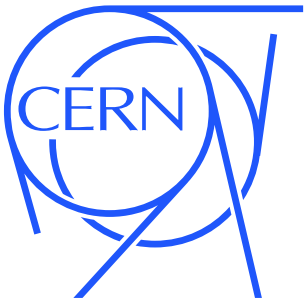

Portable Analysis Environment using Virtualization Technology (WP9)

The CernVM File System

Jakob Blomer Predrag Buncic

Revision 2.0-7

Technical Report
June 2012

Abstract

The CernVM File System is a client-server file system developed to deliver software stacks onto virtual machines in a fast, scalable, and reliable way. CernVM-FS is implemented as a FUSE module. It makes a specially prepared directory tree stored on a web server look like a local read-only file system on the virtual machine. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It transfers data file by file on demand, verifying the content by SHA-1 keys.

By means of aggressive caching and reduction of latency, we focus specifically for the software use case. Volumes hosting software consist usually of many small files that are frequently opened and read as a whole. Furthermore, we see frequent looking for files in multiple directories when search paths are examined.

We currently use CernVM-FS as part of the CernVM project. ATLAS and LHCb experiments use it to distribute software and conditions data to Grid sites. We host and distribute several hundred gigabytes of software, amongst others experiment software for ATLAS, CMS, LHCb, NA61 as well as software for the LCD collaboration.

Contents

1. Overview

The CernVM File System (CernVM-FS) is a client-server file system developed to deliver software distributions onto virtual machines in a fast, scalable, and reliable way. The delivered software is intended to reside in the form of binaries on a repository server, i. e. the repository server reflects usually the result of a `make install`.

CernVM-FS is implemented as File System in User Space (FUSE) [HS] module. Figure 1.1 shows general idea of distributing software with CernVM and CernVM-FS. Figure 1.3 shows how CernVM-FS interlocks with Fuse and a web server in order to deliver files. It has been designed to make a directory tree stored on a web server look like a local read-only file system on the virtual machine. On the client side it requires only outgoing HTTP connectivity to a web server and/or an HTTP proxy server. Assuming that the files are read only, CernVM-FS does aggressive file level caching. Both files and file metadata are cached on the local disk as well as on proxy servers, allowing the file system to scale to a very large number of clients.

The first implementation of CernVM-FS was based on grow-fs [**?**,CGL+10], which was originally provided as one of the private file system options available in Parrot. Parrot traps the system I/O calls and that is resulting in a performance penalty and occasional compatibility problems with some applications. The principal new features in CernVM-FS compared to grow-fs are:

- Using FUSE kernel module allows for in-kernel caching of file data and file attributes.
- Use of content addressable storage format resulting in immutable files and automatic file de-duplication
- Capability to work in offline mode providing that all required files are cached.
- Possibility to split a directory hierarchy into sub catalogues.
- Transparent file compression/decompression.
- Dynamical expansion of environment variables embedded in symbolic links.
- Capability to cope with digitally signed (X.509) file catalogs.
- Automatic updates of file catalogs controlled by a time to live stored inside file catalogs
- Automatic selection of mirror servers based on network round trip time.
- Random selection from a set of forward proxy servers, which results in automatic load-balancing of proxy servers

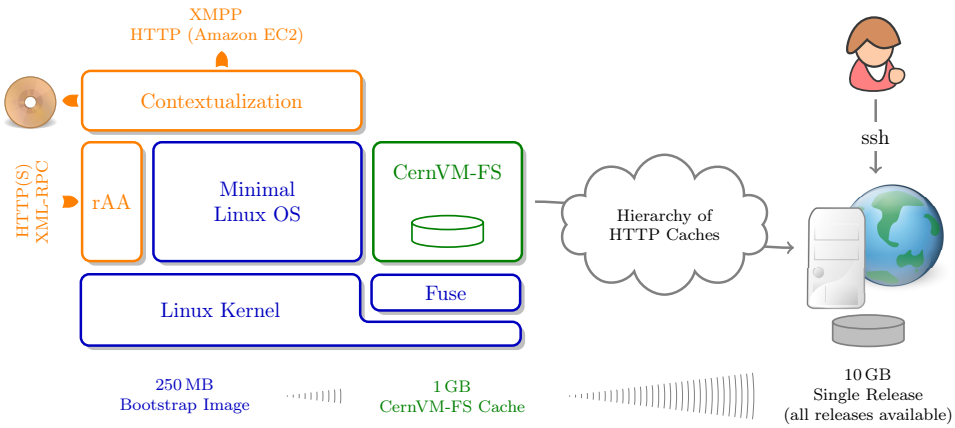

Figure 1.1.: Building blocks of CernVM 2. CernVM is built around a minimal SL5. Experiment software is loaded file by file on demand and is locally cached.

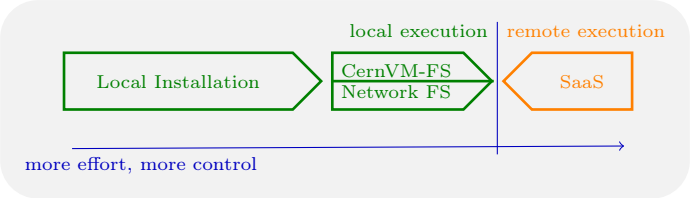

Figure 1.2.: Classification of CernVM-FS and several alternative options for software installation.

Architectural-wise, CernVM-FS is comparable to a network file system (Figure 1.2). Though the task of installing software is done remotely on the repository side, software is still executed locally under control of the user. This is in contrast to software as a service where control is completely handed over to a third party. In contrast to general purpose network file systems such as NFS, CernVM-FS is particularly crafted for fast and scalable software distribution. For instance, running and compiling software might easily overload NFS or Lustre [Sch03] shared software areas.

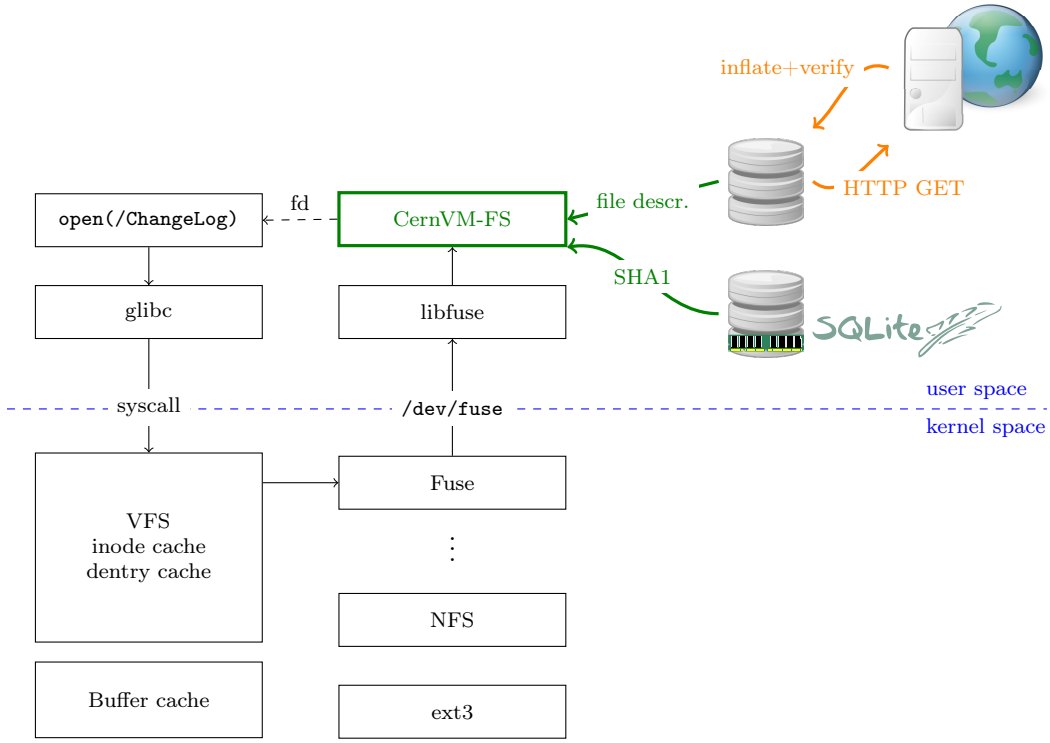

Figure 1.3.: Process of opening a file. CernVM-FS resolves the name by means of an SQLite catalog, which is prepended by a memory cache. Downloaded files are verified against the cryptographic hash of the corresponding catalog entry. The `read()` and the `stat()` system call can be entirely served from the in-kernel file system buffers.

2. Getting Started

This section describes how to install CernVM-FS on Scientific Linux 5/6 for use with CernVM repositories. For a list of currently hosted repositories at CERN see Table 2.1. CernVM-FS comes as a source tarball and as an rpm for Scientific Linux on x86 and x86_64 architectures. Current versions are available from https://cernvm.cern.ch/portal/downloads.

2.1. Setting up yum for CernVM-FS

If you want to use yum, there is a repository available at http://cvmrepo.web.cern.ch/cvmrepo/yum. The `cvmfs-release` package can be used to add the CernVM-FS repository to your local yum. Use the following script to download and install the `cvmfs-release` RPM:

```
RELEASE=$(rpm -q --queryformat '%{version}\n' sl-release)
RELEASE_MAJOR=$(echo $RELEASE | cut -d. -f1)
ARCH=$(uname -m)
wget http://cvmrepo.web.cern.ch/cvmrepo/yum/cvmfs/EL/$RELEASE/$ARCH/\
cvmfs-release-2-2.el${RELEASE_MAJOR}.noarch.rpm
sudo rpm -i cvmfs-release-2-2.el${RELEASE_MAJOR}.noarch.rpm
```

For Ubuntu and SuSE, compile from sources using the following configure options:

```
--enable-sqlite3-builtin --enable-libcurl-builtin \
--enable-libfuse-builtin --enable-libfuse-builtin \
--enable-zlib-builtin --enable-mount-scripts --disable-server \
--prefix=/usr
```

2.2. Installation

To install, proceed according to the following steps:

Step 1 Install the CernVM-FS packages. With yum, run

```
1 yum install cvmfs-keys cvmfs cvmfs-init-scripts.
```

If yum does not show the latest packages, clean the yum cache by yum clean all. Use rpm –vi to install the packages using just rpm.

Repository	Description
atlas.cern.ch	ATLAS experiment software
atlas-condb.cern.ch	ATLAS conditions database
atlas-nightlies.cern.ch	ATLAS nightly builds
cms.cern.ch	CMS experiment software
lhcb.cern.ch	LHCb experiment software
lhcb-conddb.cern.ch	LHCb conditions database
na61.cern.ch	NA61 experiment software
hone.cern.ch	H1 experiment software
boss.cern.ch	BES experiment software
lcd.cern.ch	Software of the Linear Collider Studies collaboration
grid.cern.ch	Grid User Interface
sft.cern.ch	LCG application's area software
geant4.cern.ch	Geant4 software

Table 2.1.: Repositories hosted at CERN

Step 2 For the base setup, run cvmfs_config setup. Alternatively, you can do the base setup by hand: ensure that `user_allow_other` is set in /etc/fuse.conf and ensure that `/cvmfs /etc/auto.cvmfs` is set in /etc/auto.master. If you migrate from a previous version of CernVM-FS, see the release notes if there is anything special to do for migration.

Step 3 Create `/etc/cvmfs/default.local` and open the file for editing.

Step 4 Select the desired repositories by setting CVMFS_REPOSITORIES=repo1,repo2,.... For ATLAS, for instance, set

```
CVMFS_REPOSITORIES=atlas.cern.ch,atlas-condb.cern.ch,grid.cern.ch
```

Specify the HTTP proxy servers on your site with

```
1 CVMFS_HTTP_PROXY="http://myproxy1:port|http://myproxy2:port"
```

For the syntax of more complex HTTP proxy settings, see Section 6.2.1. Make sure your local Squid servers allow access to all the Stratum 1 web servers 4. For Cern repositories, the Stratum 1 web servers are listed in /etc/cvmfs/domain.d/cern.ch.conf.

Step 5 Restart the cvmfs service by service cvmfs restart.

Step 6 Check if CernVM-FS mounts the specified repositories by service cvmfs probe.

2.3. Usage

The CernVM-FS repositories are located under /cvmfs. Each repository is identified by a fully qualified repository name. That is a repository identifier and a domain name, similar to DNS [Moc87]. The domain part of the fully qualified repository name indicates the location of repository creation and maintenance. For the ATLAS experiment software, for instance, the fully qualified repository name is atlas.cern.ch.

Mounting and un-mounting of the CernVM-FS is controlled by autofs and automount. That is, starting from the base directory /cvmfs different repositories are mounted automatically just by accessing them. For instance, running the command ls /cvmfs/atlas.cern.ch will mount the ATLAS software repository. This directory gets automatically unmounted after some automount-defined idle time.

Usually, some additional configuration is necessary to make the software of a certain repository run properly. This includes symbolic links from /opt/*repository* to /cvmfs/*repository*.cern.ch. The extra configuration is done by the cvmfs service.

2.4. Debugging Hints

In order to check for common misconfigurations in the base setup, run

```
cvmfs_config chksetup
```

CernVM-FS gathers its configuration parameter from different configration files (default, domain specific, local setup, ...). To show the effective configuration for *repository*.cern.ch, run

```
1 cvmfs_config showconfig repository.cern.ch
```

In order to exclude autofs/automounter as a source of problems, you can try to mount *repository*.cern.ch manually by

```
1 mkdir /mnt/cvmfs
mount -t cvmfs repository.cern.ch /mnt/cvmfs
```

Once you sorted out a problem, make sure that you do not get the original error served from the file system buffers by

```
service cvmfs restartautofs
```

In case you need additional help, please don't hesitate to contact us at cernvm.support@cern.ch. Together with the problem description, please send the system information tarball created by `cvmfs_config bugreport`.

3. Installation from Sources

This section describes how to manually install the CernVM-FS client from the source tarball. Current versions are available from https://cernvm.cern.ch/portal/downloads. In order to compile CernVM-FS from the tarball, you need a Linux system having installed the packets listed in Table 3.1.

3.1. Installation of the CernVM-FS Fuse Module

Download and uncompress the source tarball. The CernVM-FS build system is created with automake. So compilation is done by the usual configure, make, make install procedure. We recommend to configure using

```
./configure --prefix=/usr --disable-server \
  --enable-sqlite3-builtin --enable-libcurl-builtin \
  --enable-zlib-builtin --enable-mount-scripts
```

The package installs the `cvmfs2`, `cvmfs2_debug`, `cvmfs_fsck`, `cvmfs_config`, and `cvmfs-talk` executables. In addition, it installs configuration files to /etc/cvmfs as well as the mount helpers `mount.cvmfs` and `umount.cvmfs` to /sbin and the autofs map `auto.cvmfs` to /etc.

3.2. Configuration of CernVM-FS

In the default configuration, you do not have to mount or un-mount CernVM-FS repositories manually; this is done by automount. Starting from the base directory /cvmfs the repositories are mounted on first access (ref. Section 2). The local configuration of CernVM-FS is controlled by a couple of files in `/etc/cvmfs` listed in Table 3.2. For every .conf file except for site.conf you can create a .local file having the same prefix in order to customize the configuration. The .local file will be sourced after the corresponding .conf file.

The variables you can set in /etc/cvmfs/default.local roughly correspond to mount options of the Fuse module. See Section 3.3 for a comprehensive list of mount options. Table 3.3 lists the known configuration parameters. To make changes to the parameters effective, the cvmfs service has to be restarted by `service cvmfs restart`.

3.2.1. Using Multiple Replicas

For reliability, multiple replica or mirror servers can be used by CernVM-FS. To do so, set `CVMFS_SERVER_URL` to a semicolon-separated list of known replica servers (enclose

Package	Client	Server	Compile
kernel built tree			✓
Fuse kernel module	✓		✓
fusermount utility	✓		
autofs	✓		
openssl $\geq$ 0.9.7a	✓	✓	✓
libz $\geq$ 1.23	(✓)	(✓)	(✓)
libcurl $\geq$ 7.15	(✓)		(✓)
sqlite3 $\geq$ 3.3.9	(✓)	(✓)	(✓)
libgomp $\geq$ 4.4.0		✓	✓
gcc, g++ $\geq$ 4.1			✓
autotools			✓
pkg-config			✓

Table 3.1.: System requirements for CernVM-FS. Packages with checkmarks in parantheses are shipped with CernVM-FS and can be compiled and linked statically.

File	Purpose
`config.sh`	Set of helper functions for the events mount, unmount, and initialization of a repository
`default.conf`	Set of parameters reflecting the standard configuration
`site.conf`	Site specific set of parameters that overwrites the standard configuration. Do not touch. This file is used by the CernVM web interface.
`domain.d/$domain.conf`	Domain-specific parameters and implementations of the functions in `config.sh`
`config.d/$repository.conf`	Repository-specific parameters and implementations of the functions in `config.sh`
`keys/$domain.pub`	Public keys used to verify the digital signature of file catalogs

Table 3.2.: List of configuration files for CernVM-FS in `/etc/cvmfs`

Parameter	Meaning
CVMFS_REPOSITORIES	Comma-separated list of fully qualified repository names that shall be mountable under /cvmfs.
CVMFS_HTTP_PROXY	Sets the `proxies` mount option. CernVM-FS supports random selection of a listed proxy server in order to balance their load load. See Section 6.2.1 on how to use lists of proxy servers. This parameter is required.
CVMFS_TIMEOUT	Timout in seconds for HTTP requests with a proxy server.
CVMFS_TIMEOUT_DIRECT	Timout in seconds for HTTP requests without a proxy server.
CVMFS_NFILES	Sets the `nofiles` mount option.
CVMFS_CACHE_BASE	Sets the parent directory of the `cachedir` mount option. The `$CVMFS_USER` has to be owner of this directory.
CVMFS_SERVER_URL	Sets the repository url having `@org@` as placeholder for the repository. Usually set for a domain. Example: http://cernvm-webfs.cern.ch/opt/@org@
CVMFS_PUBLIC_KEY	Sets `pubkey` mount option. Usually set for a domain.
CVMFS_STRICT_MOUNT	If set to yes, only repositories listed in `CVMFS_REPOSITORIES` can be mounted.
CVMFS_FORCE_SIGNING	If set to yes, only digitally signed repositories can be mounted.
CVMFS_SYSLOG_LEVEL	Sets the `syslog_level` mount option.
CVMFS_TRACEFILE	Sets the `tracefile` mount option.
CVMFS_DEBUGLOG	Specifies a debug log file. Having this variable set, CernVM-FS runs in debug mode which causes high load. The debug log is different from normal log messages written to syslog.
CVMFS_MAX_TTL	Sets the `max_ttl` mount option.
CVMFS_USER	Sets the `gid` and `uid` mount options. Don't touch or overwrite.
CVMFS_OPTIONS	Set of standard Fuse options that are added. Don't touch or overwrite.
CVMFS_MOUNT_DIR	Sets the CernVM-FSroot directory. Don't touch or overwrite.

Table 3.3.: List of recognized parameters in `/etc/cvmfs/default.local`. For CernVM-FS mount options see Section 3.3.

in double quotes). The so defined URLs are organized as a ring buffer. Whenever download of files fails from a server, CernVM-FS automatically switches to the next mirror server. Additionally, on the first download and after every 1000 downloads, CernVM-FS orders the list of servers according to the round trip time for downloading a file; it then switches automatically to the closest server.

3.3. Manually Mounting the CernVM-FS Fuse Module

In order to mount a remote HTTP repository manually onto a local mount point, use the following basic syntax

```
mount -t cvmfs [-o $mount_options] $repository $mount_point
```

The `mount` command will invoke the CernVM-FS mount helper `/sbin/mount.cvmfs`, which in turn will invoke the `cvmfs2` binary. The `$mount_point` is a path to an empty directory as with any other mount. The CernVM-FS mount helper takes care of transforming the `$repository` pseudo device into an HTTP URL, according to the `CVMFS_SERVER_URL` parameter. Additionally it specifies a set of default mount options listed in Table 3.4. The `-f` option to the `mount` command does a *dry run*, i. e. it shows the command line that is used to invoke the `cvmfs2` process; this might be useful for debug purposes.

CernVM-FS has to be started as root; it will drop privileges to `$CVMFS_USER`. Useful mount options are

−o rebuild_cachedb	The managed cache database is rebuilt from the cache directory. This might become necessary when CernVM-FS was mounted once with quota_limit and once without.
−o deep_mount	Path prefix if a repository is mounted on a nested catalog, i.e. −o deep_mount=/software/15.0.1.
−o force_signing	CernVM-FS will accept only signed catalogs. This might create hard to find I/O errors, if a root catalog is signed but one of its nested catalogs is not. Failures due to invalid signatures are written into syslog.
−o whitelist=<url>	HTTP location of a signed white-list containing certificate fingerprints. Only file catalogs from certificates referenced in this white-list are accepted as validly signed. Defaults to `<root url>/.cvmfswhitelist`.
−o pubkey=<pemfile>	Public RSA key that is used to verify the the white-list signature.
−o syslog_level=<NUMBER>	Sets the level used for syslog to DEBUG (1), INFO (2), or NOTICE (3). Default is NOTICE. Note that the level applies only to messages written into syslog, not to the debug log. The debug log is a separate log facility that is turned on and off by the `CVMFS_DEBUGLOG` parameter.

See cvmfs2 −−help for a complete list of available mount options.

3.4. Debugging

The `cvmfs2` binary forks a watchdog process on start. Using this watchdog, CernVM-FS is able to create a stack trace in case certain signals (such as segmentation fault) are received. The watchdog writes the stack trace into syslog as well as into a file `stacktrace` in the cache directory. The `cvmfs-talk` utility can be used to determine the PID of the main process (see Section 6.5.7).

In addition to the `cvmfs2` binary, CernVM-FS comes with a `cvmfs2_debug` binary that is compiled with debug symbols and without optimization. In case there are any problems with CernVM-FS, this binary can create a detailed log file. Also, it is compiled without optimizations for better examination by gdb. In order to create a log file, use the `cvmfs2_debug` and mount with the additional mount options

```
-o debug,logfile=$absolute_path
```

You will get the standard mount options by

```
1 mount -f -t cvmfs $repository $mount_point
```

−o fsname=cvmfs2	The mount command will show cvmfs2 instead of fuse. This is for convenience.
−o ro	Marks the file system as read-only.
−o nodev	Marks the file system as unrelated to any system device.
−o grab_mountpoint	Changes the owner of the mount point to `$CVMFS_USER`.
−o kernel_cache	Uses Linux file system buffers to cache file data. This option brings probably the biggest performance boost. Used in conjunction with auto_cache.
−o auto_cache	Invalidate changed files in the Linux file system buffers. Used in conjunction with kernel_cache.
−o allow_other	Allows other users than the mounting user to access the file system.
−o entry_timeout −o negative_timeout −o attr_timeout	Specifies the maximum caching duration for file meta data in the kernel in seconds. 10 seconds is a reasonable value.
−o max_ttl	Specifies a maximum time to live for file catalogs in minutes. That way, the repsonse time to updates can be improved (default is 1 hour). Note that a shorter TTL will also stress the local Squids more.
−o quota_limit −o quota_threshold	If quota_limit is specified, CernVM-FS turns the local cache into a managed LRU cache. When the cache grows beyond the amount of MB specified by quota_limit, CernVM-FS removes files from the cache according to LRU until the size is below quota_threshold. As a side effect, this restricts the maximum file size to quota_limit-quota_threshold. In case only quota_limit=−1 is specified, the cache size is unrestricted.
−o nofiles	Sets the maximum number of open files for CernVM-FS process (soft limit) to `$CVMFS_NFILES`. Set this at least to 10 000.
−o cachedir	Sets the cache directory for the file and catalog cache. The cache directory will be created on demand; it has to be owned by `$CVMFS_USER`.
−o proxies	Specifies the semicolon-separated list of forward proxy servers to `$CVMFS_PROXY`.
−o uid −o gid	Sets the user id and group id of `$CVMFS_USER`

Table 3.4.: Mount options as added by `/etc/auto.cvmfs` with the default options defined in `/etc/cvmfs/default.conf`

Alternatively, the mount helper will do the same having the `CVMFS_DEBUGLOG` variable set to a file name.

CernVM-FS assumes that the local cache directory is trustworthy. However, it might happen that files get corrupted in the cache directory caused by errors outside the scope of CernVM-FS. CernVM-FS stores files in the local disk cache with their cryptographic content hash key as name, which makes it fairly easy to verify file integrity. CernVM-FS contains the `cvmfs_fsck` utility to do so for a specific cache directory. Its return value is comparable to the system's `fsck`. For example,

```
1 cvmfs_fsck -j 8 /var/cache/cvmfs2/atlas.cern.ch
```

checks all the data files and catalogs in `/var/cache/cvmfs2/atlas.cern.ch` using 8 concurrent threads. Supported options are:

-j #threads	Sets the number of concurrent threads that check files in the cache directory. Defaults to 4.
-p	Tries to automatically fix problems.
-f	Unlinks the managed cache database (cf. Section 6.3.1), i. e. it will be rebuilt by CernVM-FS on next mount.

4. Setting up a Local Squid Proxy

If you setup CernVM-FS on your local cluster, we strongly recommend to setup a Squid forward proxy server as well. This is for two reasons: it will reduce the latency for the local worker nodes, which is critical for cold cache performance. And it reduces the load on our backend server.

From what we have seen, a Squid server on commodity hardware scales well for at least a couple of hundred worker nodes. The more RAM and hard disk you can devote for caching the better. We have good experience with 4-8 GB of memory cache and 50-100 GB of hard disk cache. We suggest to setup two identical Squids for reliability and load-balancing. Assuming the two servers are A and B, adjust your CernVM-FS client configuration as follows:

```
CVMFS_HTTP_PROXY="http://A:3128|http://B:3128"
```

Squid is very powerful and has lots of configuration and tuning options. For CernVM-FS we require only the very basic static content caching. Starting from a standard Scientific Linux 5 Squid, all we have to do is to adjust the cache size. If you're using ACLs, add ACLs allow rules for the Stratum 1 servers[1]. Browse through your /etc/squid/squid.conf and make sure the following lines appear accordingly

```
collapsed_forwarding on
max_filedesc 8192
maximum_object_size 1024 MB

# 4 GB memory cache
cache_mem 4096 MB
maximum_object_size_in_memory 128 KB
# 50 GB disk cache
cache_dir ufs /var/spool/squid 50000 16 256
```

Check your Squid configuration with `squid -k parse`. Create the hard disk cache area with `squid -z`. In order to make the increased number of file descriptors effective for Squid, execute `ulimit -n 8192` prior to starting the Squid service.

[1] For Cern repositories Stratum 1 servers are http://cvmfs-stratum-one.cern.ch:8000 (CERN), http://cernvmfs.gridpp.rl.ac.uk:8000 (RAL), and http://cvmfs.racf.bnl.gov:8000 (BNL).

5. Creating a Repository

Though in principle a CernVM-FS repository is just a directory tree, it is converted into a *repository* format first. The repository format is in particular content addressable storage. We call the original directory tree *shadow tree*. This task includes creating the file catalog(s), compressing the files and calculating content hashes. Furthermore, we store the files in the same layout as the local CernVM-FS cache on the server, i. e. as SHA1 data chunks. We do so to exploit redundancy and in order to mangle the real file name into an SHA1 key when CernVM-FS downloads files. This circumvents certain firewall restrictions. For instance, many firewalls block an HTTP request to a file called `root.exe`. Figure 5.1 outlines the repository generation.

Since the repositories may contain many file system objects[1], we cannot afford to process an entire shadow tree from scratch for every update. Instead, we choose a journal based approach supported by the redirfs kernel level framework [Hrb05]. The redirfs framework hooks into VFS[2] calls and allows to install so-called filters. By installing a CernVM-FS filter for redirfs, we create a journal of file system changes which is processed by `cvmfs_sync`. "Process the journal" means that the shadow tree is synchronized with the repository. This includes compression of new and updated files and updating of the file catalogs.

In order to create a repository, the server part of CernVM-FS and the CernVM-FS kernel modules are required. The server part and the kernel modules are available as rpms (see Section A). The server tools contain the `cvmfs_sync` and the `cvmfs_zpipe`, `cvmfs_sign`, `cvmfs_pull`, `cvmfs_scrum`, and `cvmfs_clgcmp` tools as well as the `redirfs` and `cvmfsflt` kernel modules. Create the directory structure shown in Table 5.1. The directory and file names are mostly recommandations and fit to the example command line to start the CernVM-FS server daemon, which is part of the CernVM-FS `add-ons` directory.

From the point of view of the file system, repositories are relocatable. However, many software installation tools hard-code the full path. In effect, repositories have to be mounted at the same location that was used to install it on the release manager machine. By convention, CernVM-FS repositories are mounted using fully qualified repository name under `/cvmfs`, for instance at `/cvmfs/atlas.cern.ch`.

Typically a repository publisher does the following steps in order to create or update a repository:

1. Make the necessary changes to the shadow directory, i. e. add new directories, patch certain binaries, ...

[1] For ATLAS, for example, "many" means order of 10^7 file system objects (i. e. number of regular files, symbolic links, and directories).

[2] Virtual File System Switch, an operating system abstraction layer for file systems.

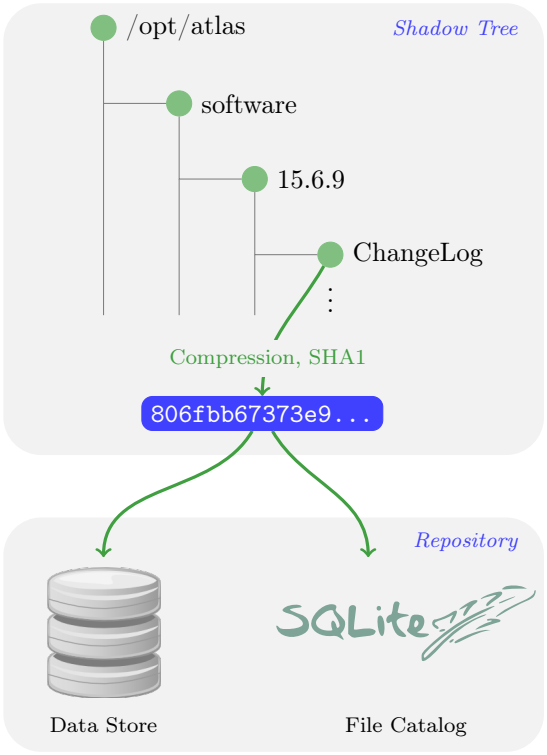

Figure 5.1.: Converting a shadow tree into a repository. The file catalog contains the directory structure as well as file metadata, symbolic links, and secure hash keys of regular files. Regular files are compressed and renamed to their cryptographic content hash before copied into the data store.

/shadow	This is a standard directory which will be watched by INOTIFY. This directory will be seen by CernVM-FS clients as read-only directory. Install your software in here.
/pub	Conatins everything that needs to be served by the webserver.
/pub/catalogs	Contains the cvmfs catalogs and symlinks to /pub/data. Mimics the directory structure in /shadow as far as necessary to provide all the entry points for catalogs. For Apache servers, it is recommended to add a .htaccess file in this directory allowing symlinks to be followed.
/pub/data	The SHA1 data cache, looks like the cache directory on clients except that files reside in a compressed form.
/ctrl	Contains zipped versions of old journals and may contain additional control files for `cvmfs_sync`.

Table 5.1.: Recommended directory layout of a repository. These directories don't have to be in the root directory. The `shadow`, the `ctrl`, and the `pub` directory don't have to be in the same parent directory.

2. Test the software installation
3. Run the `cvmfs_sync` utility and optionally the `cvmfs_sign` utility.
4. Make the web server serve the new version of the pub directory.

5.1. CernVM-FS Respository Out of the Box

Small organizations can use the `cvmfs_server` script in order to easily create a new CernVM-FS repository with reasonable defaults. Run without options for documentation. The tool expects an SL5 distribution with an `httpd` service running and *no* CernVM-FS client utilities. The server utilities and the client utilites are mutually exclusive.

The `cvmfs_server` uses the /srv/cvmfs area as storage. So if you want to use a large hard disk, mount it there upfront.

The necessary steps for a new repository are:

1. `cvmfs_server mkfs ams.iss.xz`
2. Install software in /cvmfs/ams.iss.xz as user cvmfs
3. `cvmfs_server publish`
4. Every 30 days: `cvmfs_server resign`

Note that the software signing key and the release manager machine certificate are newly created as well. In particular, they are different from CERN repositories. In order to mount the so-created repositories on CernVM-FS clients, push the public signing key /etc/cvmfs/keys/ams.iss.xz.pub from the release manager machine to the clients. A release manager machine can only manage one repository.

6. Under the Hood

CernVM-FS has a modular structure and relies on several open source libraries. Figure 6.1 shows the internal building blocks of CernVM-FS. Most of those libraries are shipped with the CernVM-FS tarball and can be linked statically to keep the system dependencies minimal.

6.1. File Catalog

A CernVM-FS repository is defined by its file catalog. The file catalog is an SQLite database [H+] having a single table that lists files and directories together with its metadata. The table layout is shown in Table 6.1.

In order to save space we do not store absolute paths. Instead we store MD5 hash values of the absolute path names. Symlinks are kept in the catalog. Symlinks may contain environment variables that will be dynamically resolved by CernVM-FS on access. The SHA1 content hash referrs to the zlib-compressed version of the file. Flags indicate the type of an directory entry (see Table 6.1).

A file catalog contains a (TTL), stored in seconds. The catalog TTL advises clients to check for a new version of the catalog, when expired. Checking for a new catalog version takes place with the first file system operation on a CernVM-FS volume after the TTL has expired. The default TTL is one hour. If a new catalog is available, CernVM-FS delays it's loading for the period of the CernVM-FS kernel cache life time (default: 1 minute). During this drain-out period, the kernel caching is turned off. The first file system operation on a CernVM-FS volume after that additional delay will apply a new file catalog and kernel caching is turned back on.

6.1.1. Nested Catalogs

In order to keep catalog sizes reasonable[1], repository subtrees may be cut and stored as separate *nested catalogs*. There is no limit on the level of nesting. A reasonable approach is to store separate software versions as separate nested catalogs. Figure 6.2 shows the directory structure which we use for the ATLAS repository.

When a subtree is moved into a nested catalog, its entry directory serves as transition point for nested catalogs. This directory appears as empty directory in the parent catalog with Flags set to 2. The same path appears as root-directory in the nested catalog with Flags set to 33. Because the MD5 hash values refer to full absolute paths, nested catalogs store the root path prefix. This prefix is prepended transparently

[1] As a rule of thumb, file catalogs up to 25 MB (compressed) are reasonably small.

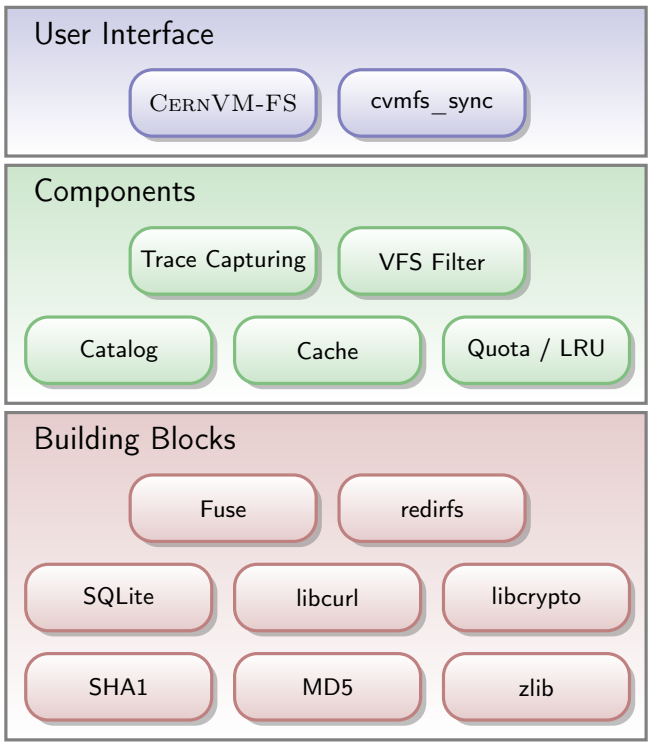

Figure 6.1.: CernVM-FS block diagram.

Field	**Type**
Path MD5	128 Bit Integer
Parent Path MD5	128 Bit Integer
inode	Integer
SHA1 Content Hash	160 Bit Integer
Size	Integer
Mode	Integer
Last Modified	Timestamp
Flags	Integer
Name	String
Symlink	String

Flags	**Meaning**
1	Directory
2	Transition point to a nested catalog
33	Root directory of a nested catalog
3	Regular file
4	Symbolic link

Table 6.1.: Metadata information stored per directory entry.

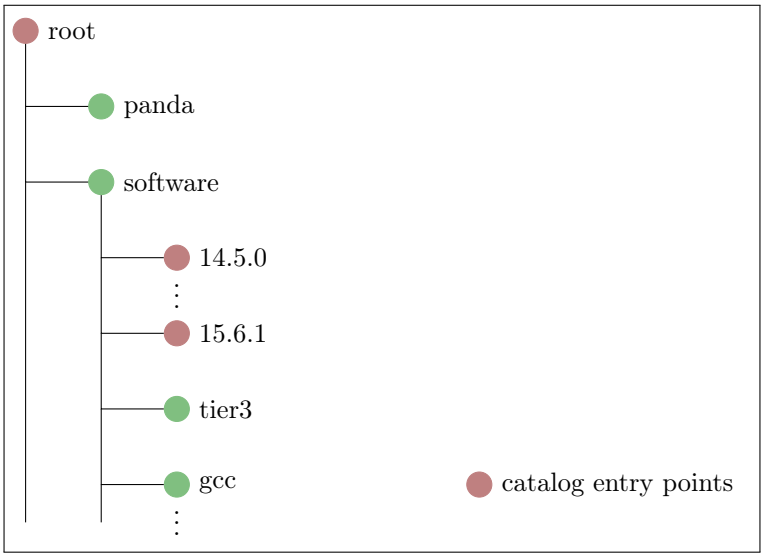

Figure 6.2.: Directory structure useds for the ATLAS repository.

by CernVM-FS. The SHA-1 key of nested catalogs is stored in the parent catalog. Therefore, the root catalog fully defines an entire repository.

Loading of nested catalogs happens on demand by CernVM-FS on the first attempt to access of anything inside, i. e. a user won't see the difference between a single large catalog and several nested catalogs. While this usually avoids unnecessary catalogs to be loaded, recursive operations like `find` can easily bypass this optimization. Nested catalogs are also separate mount points. This allows for mounting CernVM-FS deep into a repository, for instance one might mount a specific software release. Note that in this case the deep_mount mount option has to be set accordingly.

6.1.2. μ-Catalogs

The μ-catalogs can be constructed in addition to the normal file catalogs. The μ-catalogs can be considered as a special case of nested catalogs, in which each directory is a nested catalog. Essentially they are a precalculated `ls -la` for all directories. They are implemented as SQlite catalogs as well having the same structure as normal catalogs. The SHA-1 content hashes of the μ-catalogs are stored in the hash field of the directory entries in the catalog table.

6.1.3. Catalog Signature

In order to provide authoritative information about a repository publisher, file catalogs may be signed by an X.509 certificate. It is important to note that it is sufficient to

sign just the file catalog. Since every file is listed with its SHA1 content hash inside the catalog, we gain a secure chain and may speak of a "signed repository".

The signature is created using the `cvmfs_sign` utility. The `cvmfs_sign` utility takes a X.509 certificate together with its private key in order to sign a catalog and its nested catalogs. On the client side, CernVM-FS supports a *trusted mode* in which only validly signed catalogs are mounted.

In order to validate file catalog signatures, CernVM-FS uses a white-list procedure. The white-list contains the SHA1 fingerprints of known publisher certificates and a timestamp. A white-list is valid for 30 days. It is signed by a private RSA key, which we refer to as CernVM *master key*. The master key is kept offline on a secure smart card at CERN. So the security of the master key is ensured by 2-factor authentication (possession of the smart card and knowledge of the pin). The smart card is used with a smart card reader with pin pad. Neither the key nor the pin are ever exposed outside the scope of those secure devices. The public RSA key that corresponds to the master key is distributed with the CernVM-FS sources and the CernVM-FS rpm as well as with every instance of CernVM.

In addition, CernVM-FS checks certificate fingerprints against a local blacklist (default location is `/etc/cvmfs/blacklist`). The blacklisted fingerprints have to be in the same format than the fingerprints on the white-list. The blacklist has precedence over the white-list.

As crypto engine, CernVM-FS use LIBCRYPTO from the OpenSSL project [The]. Figure 6.3 shows the trust chain with a signed repository.

6.2. Exploited HTTP Features

The particular way of using the HTTP protocol has significant impact on the performance and usability of CernVM-FS. The benefit from extended HTTP/1.1 features like keep-alive, cache-control and pipelining [FGM+99] requires the intermediate proxy servers and web servers to support them, too. In any way, CernVM-FS has to work using just plain old HTTP/1.0 [BLFF96]. Internally, CernVM-FS uses the LIBCURL library [Dan].

The HTTP behaviour affects a "cold cached" CernVM-FS system. As soon as all necessary files are cached, there is only network traffic when a catalog TTL expires. Usually we'll see network traffic right after booting a CernVM for the first time, after switching to another experiment environment, and after a new software version has been published.

Downloading files is serialized by CernVM-FS. This avoids wasting the network channel by parallel downloads of the same file.

6.2.1. Forward Proxies

CernVM-FS has a dedicated HTTP proxy configuration, independent from system-wide settings. We encourage sites to setup a proxy that mirrors the CernVM-FS repository, in order to decrease latency and increase reliability. Instead of a single proxy,

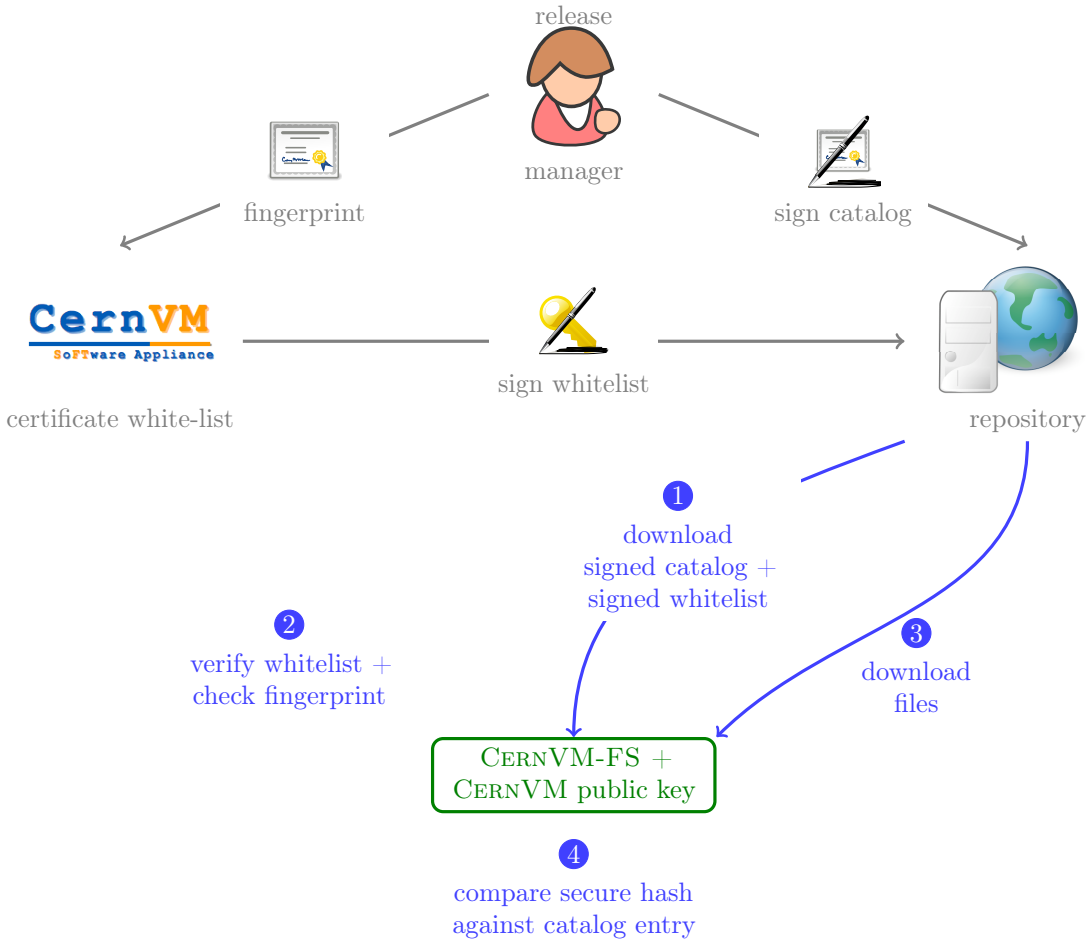

Figure 6.3.: Trust chain with a signed repository.

CernVM-FS uses a *chain of load-balanced proxy groups*. If a proxy fails, CernVM-FS automatically switches to the next group in the chain (*automatic fail-over*). The chain is internally treated as ring buffer, i. e. after probing the last proxy in the chain, the first proxy is probed again. To avoid endless loops, for each file download the number of switches is restricted by the total number of proxies. Each of the proxy groups is itself a list of proxy servers from which one of the servers is selected randomly. This leads to a *load-balancing* of proxy servers. Within a load-balanced group, fail-over takes place to another random server of that group until all proxy servers in the group failed in a row.

The chain of proxy groups is specified by a string of semicolon separated entries, each group is a list of pipe separated hostnames[2]. The `DIRECT` keyword for a hostname avoids using proxies.

6.2.2. Timeouts

CernVM-FS does an effort to recover from broken network links. There is a configurable timeout for connection attempts and for very slow downloads (mount option −o timeout), which is 10 seconds by default. A download is considered to be "very slow" if the transfer rate is below 100 Bytes/second for more than the timeout interval. A very slow download is treated like a broken connection. However, stalled DNS requests may still make the CernVM-FS process look hanging. Therefore, we recommend a local DNS caching server, such as a BIND caching server or `nscd`.

On timeout, CernVM-FS switches to the next proxy server and/or host if defined. Otherwise it returns with an EIO error to the application. CernVM-FS distinguishes between a proxied connection and a direct connetion and allows different timouts for the two cases.

CernVM-FS uses exponential backoff for download failures. That prevents request storms to web servers from applications trying to open a file in an endless loop. The backoff is triggered by consequtive download errors within 10 seconds.

6.2.3. Keep-Alive

Although the HTTP protocol overhead is small in terms of data volume, in high latency networks we suffer from the bare number of requests: Each request-response cycle has a penalty of at least the network round trip time. Using plain HTTP/1.0, this results in at least 3x(round trip time) additional running time per file download for TCP handshake, HTTP GET, and TCP connection finalisation. By including the `Connection: Keep-Alive` header into HTTP requests, we advise the HTTP server end to keep the underlying TCP connection opened. This way, overhead ideally drops to just round trip time for a single HTTP GET. The impact of the keep-alive feature is shown in Figure 6.4.

This feature, of course, somewhat sabotages a server-side load balancing. However, exploiting the HTTP keep-alive feature does not affect scalability per se. The servers

[2] The usual proxy notation rules apply, like `http://proxy1:8080|http://proxy2:8080;DIRECT`

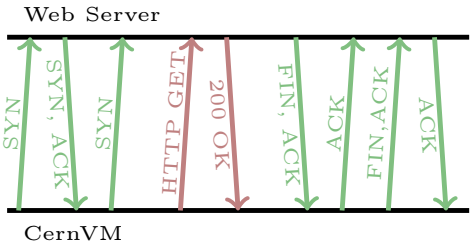

Figure 6.4.: Impact of keep-alive header on multiple file downloads.

and proxies may safely close idle connections anytime, in particular if they run out of resources. In practice, the maximum connection duration has to be set carefully for the HTTP deamon.

6.2.4. Cache Control

In a limited way, CernVM-FS advises intermediate web caches how to handle its requests. Therefor it uses the `Pragma: no-cache` and the `Cache-Control: no-cache` headers in certain cases. These cache control headers apply to both, forward proxies as well as reverse proxies. However, this is by no means a guarantee that intermediate proxies fetch a fresh original copy (though they should).

By including these headers, CernVM-FS tries to not fetch outdated cache copies. This has to be handled with care, of course, in order to not overload the repository source server. In case CernVM-FS downloads a corrupted file from a proxy server, it retries having the HTTP `no-cache` header set. This way, the corrupted file gets replaced in the proxy server by a fresh copy from the backend.

6.2.5. Identification Header

CernVM-FS sends a custom header (X−CMFS2) to be identified by the web server. In CernVM we use this header to rewrite URL's to the respective repository format for CernVM-FS version 1 or CernVM-FS version 2. If you have set the CernVM GUID, this GUID is also transmitted.

6.3. Disk Cache

The disk cache stores files with a name according to their SHA-1 content hash. Thereby the disk cache is decoupled from any specific directory hierarchy. Identical files stored in different directories are stored only once.

Each running CernVM-FS instance has a separate cache directory. The local cache directory (directories `00`, . . . , `ff`) can be accessed in parallel to a running CernVM-FS,

Field	Type
SHA1	160 Bit Integer
Size	Integer
Access Sequence	Integer
Pinned	Integer
File type (chunk or file catalog)	Integer

Table 6.2.: Cache catalog table structure.

i. e. files can be deleted for instance anytime[3]. During the download, files reside in the `txn` directory. At the very latest point they are renamed into their content addressable SHA1 names atomically by `rename()`.

6.3.1. Managed Disk Cache

The traditional CernVM-FS disk cache just grows until the system runs out of disk space. Files may be deleted manually from the cache directories, but this is a cumbersome job.

By using a *managed disk cache*, CernVM-FS maintains cache size restrictions and replaces files according to the least recently used (LRU) strategy [PS06]. In order to keep track of files sizes and relative file access times, CernVM-FS sets up another SQLite database in the cache directory, the *cache catalog*. The cache catalog contains a single table; its structure is shown in Table 6.2.

Note that CernVM-FS does not enforce a certain cache size. Instead CernVM-FS works with two customizable soft limits, the *cache quota* and the *cache threshold*. When exceeding the cache quota, files are deleted until the overall cache size is less than or equal to the cache theshold. The cache quota is for data files as well as file catalogs. Currently loaded catalogs are pinned in the cache, i. e. they will not be deleted. On unmount, pinned file catalogs are updated with the highest sequence number.

The cache catalog can be constructed from scratch on mount. Re-constructing the cache catalog is necessary when the managed cache is used for the first time and every time when "unmanaged" changes occurred to the cache directory. This happens, for instance, if CernVM-FS is mounted with the managed cache feature turned off. Re-construction has to be triggered manually.

For performance reasons, the cache management runs as a separate thread. This thread takes care of updating access times and deleting of files when necessary. The CernVM-FS `open` function talks via pipes to that thread. A typical compilation benchmark on CernVM-FS shows 1%-2% additional running time caused by the managed cache.

[3] While the design supports it, we however strongly suggest not to tamper with the cache directory because it will make the local LRU database inconsistent.

```
"1481074921.015","1","/root/i686-pc-linux-gnu/include/TQObject.h","OPEN"
"1481074931.030","1","/root/i686-pc-linux-gnu/include/KeySymbols.h","OPEN"
"1481074931.220","3","/root/i686-pc-linux-gnu/include/KeySymbols.h","READ (TRY)"
"1481074964.565","3","/root/i686-pc-linux-gnu/include/KeySymbols.h","READ 7407@0"
"1481074965.005","1","/root/i686-pc-linux-gnu/include/TRootCanvas.h","OPEN"
```

Figure 6.5.: Example snippet of a trace log. The first columns stores time stamps as number of microseconds starting from the Unix epoch. The second column stores the event type. Negative event types are reserved for CernVM-FS internal events.

6.4. File System Traces

CernVM-FS has an optional file system operations tracer. The tracer creates logs of usage, which can—for instance—be used as profiling information for pre-fetching. The trace file is created in CSV format (see Figure 6.5).

The tracing runs in a separate thread and adapts the *tread-safe trace buffer*, a technique used for multi-thread debugging [AR06, Chapter 8]. Since traces are kept in a memory ring buffer[4] and written to disk in blocks of thousands of lines, the performance overhead for tracing is almost negligible; for our application benchmarks it is below measurement error.

6.5. System Interface

Since CernVM-FS is a read-only file system, there are only few non-trivial call-back functions to implement. These call-back functions provide the system interface.

6.5.1. `mount` / `re-mount`

On mount, the file catalog has to be loaded. First, the file catalog checksum `.cvmfspublished` is loaded. If CernVM-FS runs in trusted mode or if it finds a signature in the checksum, the catalog is only accepted on successful validation of the signature. In order to validate the signature, the certificate and the white-list are downloaded in addition if not found in cache. If the download failes for whatever reason, CernVM-FS tries to load a local file catalog copy. As long as all requested files are in the disk cache as well, CernVM-FS continues to operate even without network access (*offline mode*).

If there is no local copy of the catalog checksum or the downloaded checksum and the cache copy differ, CernVM-FS downloads a fresh copy of the file catalog. If the checksum matches the file catalog, the cache copies are replaced by the downloaded versions and the new catalog is mounted. Otherwise, the procedure is repeated avoiding intermediate proxy servers (see Section 6.2.4).

[4] Usually, each trace record requires two atomic `fetch-and-add` operations.

6.5.2. `stat`

Requests for file attributes are entirely served from the mounted catalog, i. e. there is no network traffic involved. This function is called as pre-requisite to other file system operations and therefore the most frequently called FUSE callback. In order to minimize relatively expensive SQLite queries, CernVM-FS uses a 2-way associative / direct-mapped negative and positive hybrid cache, having the first bits of the MD5 path name hash as key. The size of the cache is determined according to compilation benchmarks monitored with Valgrind [Wei] and according to LHCb application benchmarks.

Additionally, this callback takes care of the catalog TTL. If the TTL is expired, the catalog is re-mounted on the fly. Note that a re-mount might possibly break running programs. We rely on careful repository publishers that produce more or less immutable directory trees, i. e. new repository versions just add files.

If a directory with a nested catalog is accessed for the first time, the respective catalog is mounted in addition to the already mounted catalogs. Loading nested catalogs is transparent to the user.

6.5.3. `readlink`

A symlink is served from the file catalog. As a special extension, CernVM-FS detects environment variables in symlink strings written as `$(VARIABLE)`. Those variables are expanded by CernVM-FS dynamically on access (in the context of the cvmfs process). This way, a single symlink can point to different locations depending on the environment. This is helpful, for instance, to dynamically select software package versions residing in different directories.

6.5.4. `readdir`

A directory listing is served by an SQL query on the file catalog. Though the "parent"-column is indexed (cf. Table 6.1), this is a relatively slow function. We expect directory listing to happen rather seldom.

6.5.5. `open / read`

The `open()` call has to provide Fuse with a file handle for a given path name. In CernVM-FS file requests are always served from the disk cache. The Fuse file handle is a file descriptor valid in the context of the CernVM-FS process. It points into the disk cache directory. Read requests are translated into the `pread()` system call.

Opening a path name is done in the following steps:

1. The path name is hashed with MD5
2. Using this MD5 hash, the file catalog is asked for the SHA1 key for the given path name
3. If the file is not in the disk cache, it is downloaded from the repository

6.5.6. `getxattr`

CernVM-FS uses extended attributes to display additional repository information. Currently there are two supported attributes:

hash Shows the SHA-1 hash of a regular file as listed in the file catalog. For a directory, shows the SHA-1 hash of the μ-catalog, if available.

lhash Shows the SHA-1 hash of a regular file as stored in the local cache, if available.

revision Shows the file catalog revision of the mounted root catalog, an auto-increment counter increased on every synchronization of shadow tree and repository. The value is the same for all directories, symbolic links and regular files of the mount point.

pid Shows the CernVM-FS process id.

version Shows the version of the loaded CernVM-FS binary.

expires Shows the remaining life time of the hosting (nested) file catalog in minutes.

maxfd Shows the maximum number of file descriptors available to file system clients.

usedfd Shows the number of file descriptors currently issued to file system clients.

nioerr Shows the total number of I/O errors encoutered since mounting.

proxy Shows the currently active HTTP proxy.

host Shows the currently active HTTP replica server.

uptime Shows the time passed since mounting in minutes.

nclg Shows the number of currently loaded nested catalogs.

nopen Shows the overall number of `open()` calls since mounting.

ndwonload Shows the overall number of downloaded files since mounting.

timeout Shows the timeout for proxied connections in seconds.

timeout_direct Shows the timeout for direct connections in seconds.

rx Shows the overall amount of downloaded kilobytes.

speed Shows the average download speed.

Extended attributes can be queried using the `attr` command. For instance, `attr -g hash /cvmfs/atlas.cern.ch/ChangeLog` returns the SHA-1 key of the file at hand. The extended attributes are used by the `cvmfs_config stat` command in order to show a current overview of health and performance numbers.

6.5.7. Dynamic Configuration

CernVM-FS is capable of communicating to the user at runtime. Thereby certain parameters can be configured and changed while the file system is mounted. To do so, CernVM-FS sets up a temporary socket in the cache directory, `cvmfs_io`. Communication is based on a simple command-response scheme. The enclosed `cvmfs-talk` script takes care of writing to and reading from the socket. Currently CernVM-FS handles the following commands:

`flush` Flushes the entries in the trace buffer from memory to disk.

`cache size` Returns the combined size of the data files and file catalogs in the (managed) disk cache.

`cache list` Returns the list of downloaded files that reside in the (managed) disk cache.

`cache list catalog` Returns the list of downloaded file catalogs that reside in the (managed) disk cache.

`cache list pinned` Returns the list of pinned (loaded) file catalogs that reside in the (managed) disk cache.

`cleanup <MB>` Unlinks files from the (managed) disk cache until cache size is less than or equal to the specified size in MB.

`clear file <path>` Removes <path> from local cache.

`mountpoint` Gets the mount point.

`remount` Checks for a new root file catalog.

`revision` Gets the currently mounted repository revision.

`max ttl info` Gets the maximum file catalog TTL.

`max ttl set <minutes>` Sets the maximum file catalog TTL.

`host info` Gets the host chain and their RTT, if already probed.

`host probe` Orders the host chain according to round trip time. This happens automatically every 1000 downloads.

`host switch` Switches to the next host in the chain.

`host set <host chain>` Sets a new host chain.

`proxy info` Gets the currently active proxy server.

`proxy rebalance` Randomly selects a new proxy server from the current load-balancing group.

`proxy group switch` Switches to the next load-balance proxy group in the chain.

`proxy set <proxy list>` Sets a new chain of load-balance proxy groups.

`timeout info` Gets the network timeouts with and without proxy.

`timeout set <proxy> <direct>` Sets the network timeouts in seconds.

`reset error counters` Resets the internal counter for failed I/O operations.

`pid` Returns the process id of the `cvmfs2` main process (not the watchdog).

`version` Returns the version number of the running CernVM-FS instance.

`version patchlevel` Returns the version patchlevel of the running CernVM-FS instance.

`open catalogs` Returns mount point, last modified timestamp, and expiry date of currently loaded catalogs. These are not necessarily all cached catalogs.

The `cvmfs-talk` command is also used by the cvmfs service in order to reload the configuration. Reloadable are all parameters with a "set" variant.

6.6. Repository Synchronization

Repositories are not immutable, every now and then they get updated. This might be installation of a new release or a patch for an existing release. But, of course, each time only a small portion of the repository is touched, say 2 GB out of 100 GB. In order to not process an entire shadow tree on each synchronization, we use a file system change log as basis of the synchronization. To this end, we use a CernVM-FS *filter* for redirfs [Hrb05] (cf. Fig. 6.6). The CernVM-FS filter intercepts VFS system calls that might change file data or file meta data on a given directory sub-tree. We refer to such VFS calls as *writing calls*.

CernVM-FS ships with two kernel modules, `redirfs` and `cvmfsflt`. These modules have to be loaded in this order. Once loaded, they are configured via sysfs [Moc05]. See Table 6.3 for a list of control files.

The CernVM-FS VFS filter has three modes of operation:

Normal Operation Capture writing VFS calls to the call buffer, whose tail is connected to a character device.

Call Buffer Full Block writing VFS calls. If the `O_NONBLOCK` flag is set, do not block but return `EAGAIN`.

Synchronizing Repository Forbid writing VFS calls (return `EPERM`).

In order to set up file system capturing, one can use the following commands:

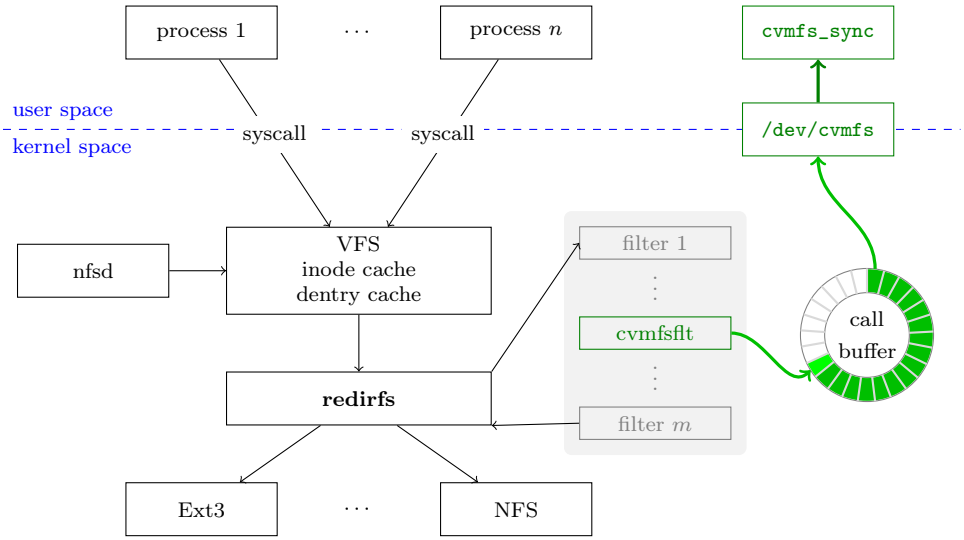

Figure 6.6.: CernVM-FS filter for redirfs: Writing VFS calls are captured and written to a character device to create a file system change log.

File	Permission	Purpose
`lockdown`	r/w	Forbid writing VFS calls (during synchronization)
`noll`	r	Number of remaining entries in the VFS call buffer
`nowops`	r	Number of remaining writing VFS calls

Table 6.3.: Sysfs control files for `cvmfsflt` in `/sys/fs/redirfs/filters/cvmfsflt`.

```
modprobe redirfs
modprobe cvmfsflt
if [ ! -c /dev/cvmfs ]; then
  major=`grep cvmfs /proc/devices | awk '{print $1}'`
  mknod /dev/cvmfs c $major 0
fi
echo -n "a:i:$MY_FILTER_PATH" > /sys/fs/redirfs/filters/cvmfsflt/paths
```

The data stream of the character device has to be stored into a log file for further processing with `cvmfs_sync`. If the character device is not read, the call buffer will eventually become full and writing VFS calls to the filtered path are blocked by the CernVM-FS filter.

The following steps have to be done before starting a synchronization run:

1. Lock the filtered path down using the sysfs control file (cf. Table 6.3)
2. Wait for the remaining writing VFS call counter to become zero
3. Wait for the remaining call entry buffer counter to become zero

In order to unload, one can use the following commands:

```
echo -n "c\0" > /sys/fs/redirfs/filters/cvmfsflt/paths
echo -n "1\0" > /sys/fs/redirfs/filters/cvmfsflt/unregister
rmmod cvmfsflt
```

A. Available RPMs

The CernVM-FS software is available in form of several RPM packages:

cvmfs-release Adds the CernVM-FS yum repository.

cvmfs-keys Contains the public key for signature verification of repositories in the cern.ch domain.

cvmfs Contains the Fuse module and additional client tools. It has dependencies to cvmfs-keys, fuse, and autofs.

cvmfs-init-scripts Contains compatibility scripts for specific repositories. Mainly, the symlinks from /opt/*repository* to /cvmfs/*repository*.cern.ch are maintained. All repositories are supposed to migrate to the /cvmfs namespace in the near future. Depends on cvmfs.

cvmfs-auto-setup Only available through yum. This is a wrapper for cvmfs_config setup. This is supposed to provide automatic configuration for the ATLAS Tier3s. Depends on cvmfs.

cvmfs-replica Contains the tools to replicate CernVM-FS repositories.

cvmfs-server Contains the CernVM-FS server tools for creating new repositories. Depends on cvmfs-keys and httpd.

redirfs, kmod-redirfs Kernel module framework used by the CernVM-FS file system change log filter.

cvmfsflt, kmod-cvmfsflt The CernVM-FS file system change log filter.

Bibliography

[AR06] Shameem Akhter and Jason Roberts. *Multi-Core Programming*. Intel Press, 2006.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, Internet Engineering Task Force, May 1996.

[CGL+10] G. Compostella, S. Pagan Griso, D. Lucchesi, I. Sfiligoi, and D. Thain. CDF software distribution on the Grid using Parrot. *Journal of Physics: Conference Series*, 219, 2010.

[Dan] Daniel Stenberg et al. libcurl. http://curl.haxx.se/libcurl.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[H+] Richard Hipp et al. SQLite. http://www.sqlite.org.

[Hrb05] Frantisek Hrbata. Callback framework for VFS layer. Master's thesis, Brno University of Technology, 2005.

[HS] Csaba Henk and Miklos Szeredi. Filesystem in Userspace (FUSE). http://fuse.sourceforge.net.

[Moc87] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035, Internet Engineering Task Force, November 1987.

[Moc05] Patrick Mochel. The sysfs filesystem. In *Proc. of the Annual Linux Symposium*, 2005.

[PS06] Konstantinos Panagiotou and Alexander Souza. On adequate performance measures for paging. *Annual ACM Symposium on Theory Of Computing*, 38:487–496, 2006.

[Sch03] Philip Schwan. Lustre: Building a file system for 1,000-node clusters. In *Proc. of the 2003 Linux Symposium*, pages 380–386, 2003.

[The] The OpenSSL Software Foundation. OpenSSL. http://www.openssl.org/docs/crypto/crypto.html.

[Wei] Josef Weidendorfer. Callgrind. http://valgrind.org/info/tools.html#callgrind.