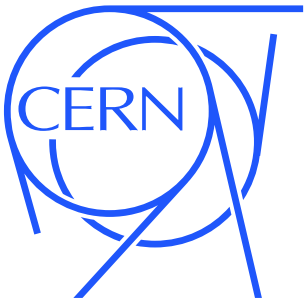

The CernVM File System

Jakob Blomer Predrag Buncic René Meusel

jblomer@cern.ch

Revision 2.1-0

Technical Report
January 2013

Abstract

The CernVM File System (CernVM-FS) provides a scalable, reliable and low-maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace `/cvmfs`. Internally, it uses content-addressable storage and Merkle trees in order to maintain file data and meta-data. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It transfers data and meta-data on demand and verifies data integrity by cryptographic a hash.

By means of aggressive caching and reduction of latency, CernVM-FS focuses specifically on the software use case. Software usually comprises many small files that are frequently opened and read as a whole. Furthermore, the software use case includes frequent look-ups for files in multiple directories when search paths are examined.

CernVM-FS is actively used by small and large HEP collaborations. In many cases, it replaces package managers and shared software areas on cluster file systems as means to distribute the software used to process experiment data.

Contents

1. Overview

The CernVM File System (CernVM-FS) is a read-only file system designed to deliver high energy physics (HEP) experiment analysis software onto virtual machines and grid worker nodes in a fast, scalable, and reliable way. Files and file metadata are downloaded on demand and aggressively cached. For the distribution of files, CernVM-FS uses a standard HTTP transport, which allows exploitation of a variety of web caches, including commercial content delivery networks. CernVM-FS ensures data authenticity and integrity over these possibly untrusted caches and connections.

The CernVM-FS software comprises client-side software to mount "CernVM-FS repositories" (similar to AFS volumes) as well as a server-side toolkit to create such distributable CernVM-FS repositories. Figure 1.1 shows an overview of software distribution with CernVM-FS. Figure 1.2 shows how CernVM-FS interlocks with Fuse and a web server in order to deliver files.

The first implementation of CernVM-FS was based on GROW-FS [TL05, 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 differences of CernVM-FS compared to GROW-FS are:

- Use of the the Fuse kernel module [HS] that comes with in-kernel caching of file data and file attributes.
- Use of a 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 at user-defined levels.
- Transparent file compression/decompression.
- Dynamical expansion of environment variables embedded in symbolic links.
- Digitally signed repositories.
- Automatic updates of file catalogs controlled by a time to live stored inside file catalogs
- Automatic server selection.
- Random selection from a set of forward proxy servers, which results in automatic load-balancing of proxy servers

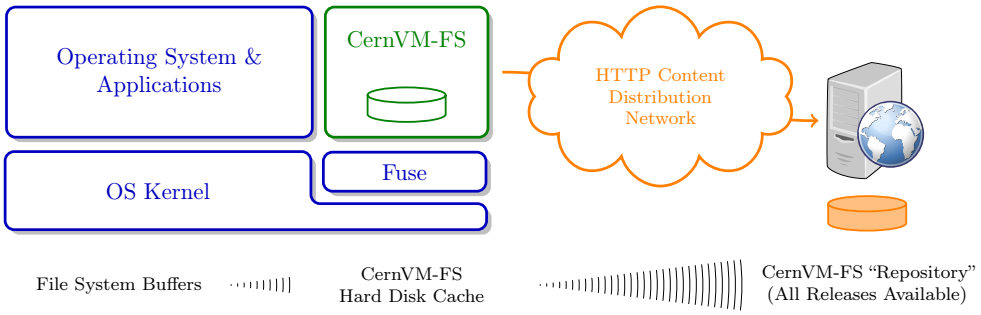

Figure 1.1.: A CernVM-FS client provides a virtual file system that loads data only on access. In this example, all releases of a sofware package (such as an HEP experiment framework) are hosted as a CernVM-FS repository on a web server.

In contrast to general purpose network file systems such as NFS or AFS, CernVM-FS is particularly crafted for fast and scalable software distribution. Running and compiling software that is hosted on shared areas is typically hard for general purpose network file systems.

In order to create and update a CernVM-FS repository, a distinguished machine, the so-called *Release Manager Machine*, is used. On such a release manager machine, a CernVM-FS repository is mounted in read/write mode by means of a union file system [WDG+04]. The union file system overlays the CernVM-FS read-only mount point by a writable scratch area. The CernVM-FS server tool kit merges changes written to the scratch area into the CernVM-FS repository. Merging and publishing changes can be triggered at user-defined points in time; it is an atomic operation. As such, a CernVM-FS repository is similar to a repository in the sense of a versioning system.

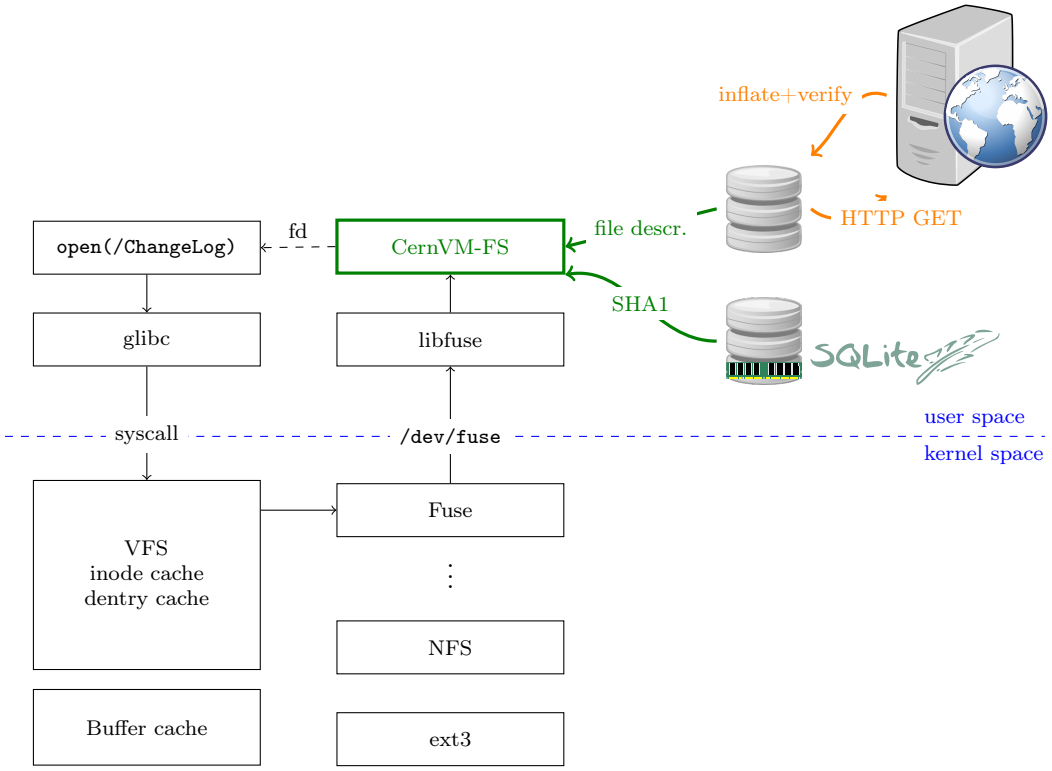

Figure 1.2.: Opening a file on CernVM-FS. CernVM-FS resolves the name by means of an SQLite catalog, which is prepended by a memory cache. Downloaded files are verified against the cryptographic hash of the corresponding catalog entry. The `read()` and the `stat()` system call can be entirely served from the in-kernel file system buffers.

2. Getting Started

This section describes how to install the CernVM-FS client. CernVM-FS is supported on Scientific Linux 5 and 6, Ubuntu 12.04, openSuSE 12.2, Fedora 17, and Mac OS X.

2.1. Getting the Software

The CernVM-FS source code and binary packages are available under https://cernvm.cern.ch/portal/downloads. Binary packages are produced for RPM, DPKG, and Mac OS X (.pkg). YUM repositories for 64 bit and 32 bit Scientific Linux 5 and 6 are available under http://cvmrepo.web.cern.ch/cvmrepo/yum. The `cvmfs-release` packages can be used to add a these YUM repositories to the local YUM installation. The `cvmfs-release` packages are available under https://cernvm.cern.ch/portal/downloads.

The CernVM-FS client is not relocatable and needs to be installed under /usr. In order to compile and install from sources, use the following CMAKE command:

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr .
make
sudo make install
```

2.2. Installation

2.2.1. Linux

To install, proceed according to the following steps:

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

```
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. On Ubuntu, use `dpkg -i` on the cvmfs .deb package.

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, check 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

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

For the syntax of more complex HTTP proxy settings, see Section **??**. 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 Check if CernVM-FS mounts the specified repositories by `cvmfs_config probe`.

2.2.2. Mac OS X

On Mac OS X, CernVM-FS is based on Fuse4X[1]. It is not yet integrated with autofs. In order to install, proceed according to the following steps:

Step 1 Install the CernVM-FS package by opening the .pkg file.

Step 2 Create /etc/cvmfs/default.local and open the file for editing.

Step 3 Select the desired repositories by setting `CVMFS_REPOSITORIES=repo1,repo2,...`. For CMS, for instance, set

```
CVMFS_REPOSITORIES=cms.cern.ch
```

Specify the HTTP proxy servers on your site with

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

If you're unsure about the proxy names, set `CVMFS_HTTP_PROXY=DIRECT`.

Step 4 Mount your repositories like

```
sudo mkdir -p /cvmfs/cms.cern.ch
sudo mount -t cvmfs cms.cern.ch /cvmfs/cms.cern.ch
```

[1] http://fuse4x.github.com

2.3. Usage

The CernVM-FS repositories are located under /cvmfs. Each repository is identified by a *fully qualified repository name*. The fully qualified repository name consists of a repository identifier and a domain name, similar to DNS records [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 although the hosting web servers are spread around the world.

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.

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 various configuration files that can overwrite each others settings (default configuration, domain specific configuration, local setup, ...). To show the effective configuration for *repository*.cern.ch, run

```
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

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

In order to exclude SELinux as a source of problems, you can try mounting after SELinux has been disabled by

```
/usr/sbin/setenforce 0
```

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

```
service autofs restart
```

In case you need additional assistance, 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. Client Configuration

3.1. Structure of /etc/cvmfs

The local configuration of CernVM-FS is controlled by several files in `/etc/cvmfs` listed in Table 3.1. For every .conf file except for site.conf you can create a corresponding .local file having the same prefix in order to customize the configuration. The .local file will be sourced after the corresponding .conf file.

In a typical installation, a handful of parameters need to be set in /etc/cvmfs/default.local. Most likely, this is the list of repositories (`CVMFS_REPOSITORIES`), HTTP proxies (see Section 3.4), and perhaps the cache directory and the cache quota (see Section 3.3) In a few cases, one might change a parameter for a specific domain or a specific repository, e. g. provide an exclusive cache for a specific repository (see Section 3.3).

The .conf and .local configuration files are key-value pairs in the form `PARAMETER=value`. They are sourced by /bin/sh. Hence, a limited set of shell commands can be used inside these files including comments, `if` clauses, parameter evaluation, and shell math (`$((...))`). Special characters have to be quoted. For instance, instead of `CVMFS_HTTP_PROXY=p1;p2`, write `CVMFS_HTTP_PROXY='p1;p2'` in order to avoid parsing errors. For a list of all parameters, see Appendix B.

3.2. Mounting

Typically, mounting of CernVM-FS repositories is handled by autofs. Just by accessing a repository directory under /cvmfs (e. g. /cvmfs/atlas.cern.ch), autofs will take care of mounting. autofs will also automatically unmount a repository if it is not used for a while.

Instead of using autofs, CernVM-FS repositories can be mounted manually with the system's `mount` command. In order to do so, use the `cvmfs` file system type, like

```
mount -t cvmfs atlas /cvmfs/atlas.cern.ch
```

Likewise, CernVM-FS repositories can be mounted through entries in /etc/fstab. A sample entry in /etc/fstab:

```
atlas.cern.ch /mnt/test cvmfs defaults 0 0
```

Every mount point corresponds to a CernVM-FS process. Using autofs or the system's mount command, every repository can only be mounted once. Otherwise multiple CernVM-FS processes would collide in the same cache location. If a repository

File	Purpose
`config.sh`	Set of internal helper functions
`default.conf`	Set of parameters reflecting the standard configuration
`site.conf`	Site specific set of parameters that overwrites the standard configuration. This file is used by the CernVM contextualization.
`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/*.pub`	Public keys used to verify the digital signature of file catalogs

Table 3.1.: List of configuration files for CernVM-FS in `/etc/cvmfs`

is needed under several paths, use a *bind mount* or use a private file system mount point (see Section 3.2.1).

3.2.1. Private Mount Points

In contrast to the system's `mount` command which requires root privileges, CernVM-FS can also be mounted like other Fuse file systems by normal users. In this case, CernVM-FS uses parameters from one or several user-provided config files instead of using the files under /etc/cvmfs. CernVM-FS private mount points do not appear as `cvmfs2` file systems but as `fuse` file systems. The `cvmfs_config` and `cvmfs_talk` commands do not affect privately mounted CernVM-FS repositories. On an interactive machine, private mount points are for instance unaffected by an administrator unmounting all system's CernVM-FS mount points by `cvmfs_config unmount`.

In order to mount CernVM-FS privately, use the `cvmfs2` command like

```
cvmfs2 -o config=myparams.conf atlas.cern.ch /home/user/myatlas
```

A minimal sample myparams.conf file could look like this:

```
CVMFS_CACHE_BASE=/home/user/mycache
CVMFS_SERVER_URL=http://cvmfs-stratum-one.cern.ch/opt/atlas
CVMFS_HTTP_PROXY=DIRECT
CVMFS_PUBLIC_KEY=/etc/cvmfs/keys/cern.ch.pub:\
  /etc/cvmfs/keys/cern-it1.cern.ch.pub:\
  /etc/cvmfs/keys/cern-it3.cern.ch.pub
```

Make sure to use absolute path names for the mount point and for the cache directory. Use `fusermount -u` in order to unmount a privately mounted CernVM-FS repository.

3.3. Cache Settings

Downloaded files will be stored in a local cache directory. The CernVM-FS cache has a soft quota; as a safety margin, the partition hosting the cache should provide 10 % more space than the soft quota limit. Once the quota limit is reached, CernVM-FS will automatically remove files from the cache according to the least recently used policy [PS06]. Removal of files is performed bunch-wise until half of the maximum cache size has been freed. Currently, CernVM-FS is not able to access files in the repository that are larger than half of the cache quota. The quota limit can be set in Megabytes by `CVMFS_QUOTA_LIMIT`. For typical repositories, a few Gigabytes make a good quota limit. For repositories hosted at CERN, quota recommendations can be found under http://cernvm.cern.ch/portal/cvmfs/examples.

The cache directory needs to be on a local file system in order to allow each host the accurate accounting of the cache contents; on a network file system, the cache can potentially be modified by other hosts. Furthermore, the cache directory is used to create (transient) sockets and pipes, which is usually only supported by a local file system such as ext3 or XFS. The location of the cache directory can be set by `CVMFS_CACHE_BASE`.

Each repository can either have an exclusive cache or join the CernVM-FS shared cache. The shared cache enforces a common quota for all repositories used on the host. File duplicates across repositories are stored only once in the shared cache. The quota limit of the shared directory should be at least the maximum of the recommended limits of its participating repositories. In order to have a repository not join the shared cache but use an exclusive cache, set `CVMFS_SHARED_CACHE=no`.

3.4. Network Settings

CernVM-FS uses HTTP [BLFF96, FGM+99] for data transfer. Repository data can be replicated to multiple web servers and cached by standard web proxies such as Squid. In a typical setup, repositories are replicated to a handful of web servers in different locations. These replicas form the CernVM-FS Stratum 1 service, whereas the replication source server is the CernVM-FS Stratum 0 server. On every cluster of machines, there should be two or more web proxy servers that CernVM-FS can use (see Section 4). These site-local web proxies reduce the network latency for the CernVM-FS clients and they reduce the load for the Stratum 1 service. CernVM-FS supports choosing a random proxy for load-balancing and automatic fail-over to other hosts and proxies in case of network errors. Roaming clients can connect directly to the Stratum 1 service.

3.4.1. Stratum 1 List

Tp specify the Stratum 1 servers, set `CVMFS_SERVER_URL` to a semicolon-separated list of known replica servers (enclose in 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. For repositories under the cern.ch domain, the Stratum 1 servers are specified in /etc/cvmfs/domain.d/cern.ch.conf.

It is recommended to adjust the order of Stratum 1 server so that the closest servers are used with priority. For roaming clients (i. e. clients not using a proxy server), the Stratum 1 servers can be automatically sorted according to round trip time by `cvmfs_talk host probe` (see Section 3.7). Otherwise, the proxy server would invalidate round trip time measurement.

The special sequence `@org@` in the `CVMFS_SERVER_URL` string is replaced by the repository name (not fully qualified). That allows to use the same parameter for many repositories hosted under the same domain. For instance, `http://cvmfs-stratum-one.cern.ch/opt/@org@` can resolve to `http://cvmfs-stratum-one.cern.ch/opt/atlas`, `http://cvmfs-stratum-one.cern.ch/opt/cms`, and so on depending on the repository that is being mounted. The same works for the sequence `@fqrn@` with fully qualified repository names (e. g. atlas.cern.cn, cms.cern.ch, ...).

3.4.2. Proxy Lists

CernVM-FS uses a dedicated HTTP proxy configuration, independent from system-wide settings. Instead of a single proxy, CernVM-FS uses a *chain of load-balanced proxy groups*. Proxies within the same proxy group are considered as a load-balance group and a proxy is selected randomly. If a proxy fails, CernVM-FS automatically switches to another proxy from the current group. If all proxies from a group have failed, CernVM-FS switches to the next proxy group. After probing the last proxy group 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.

The chain of proxy groups is specified by a string of semicolon separated entries, each group is a list of pipe separated hostnames[1]. The `DIRECT` keyword for a hostname avoids using proxies.

Multiple proxy groups are often organized as a primary proxy group at the local site and backup proxy groups at remote sites. In order to avoid CernVM-FS being stuck with proxies at a remote site after a fail-over, CernVM-FS will automatically retry to use proxies from the primary group after some time. The delay for re-trying a proxies from the primary group is set in seconds by `CVMFS_PROXY_RESET_AFTER`. The distinction of primary and backup proxy groups can be turned off by setting this parameter to 0.

3.4.3. Timeouts

CernVM-FS tries to gracefully recover from broken network links and temporarily overloaded paths. The timeout for connection attempts and for very slow downloads can be set by `CVMFS_TIMEOUT` and `CVMFS_TIMEOUT_DIRECT`. The two timeout parameters apply to a connection with a proxy server and to a direct connection to a Stratum 1 server, respectively. A download is considered to be "very slow" if the transfer rate is

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

below 100 Bytes/second for more than the timeout interval. A very slow download is treated like a broken connection.

On timeout errors and on connection failures (but not on name resolving failures), CernVM-FS will retry the path using an exponential backoff. This introduces a jitter in case there are many concurrent requests by a cluster of nodes, allowing a proxy server or web server to serve all the nodes consecutively. `CVMFS_MAX_RETRIES` sets the number of retries on a given path before CernVM-FS tries to switch to another proxy or host. The overall number of requests with a given proxy/host combination is `$CVMFS_MAX_RETRIES`+1. `CVMFS_BACKOFF_INIT` sets the maximum initial backoff in seconds. The actual initial backoff is picked with milliseconds precision randomly in the interval $[1, \$CVMFS\_BACKOFF\_INIT \cdot 1000]$. With every retry, the backoff is then doubled.

3.5. NFS Server Mode

In case there is no local hard disk space available on a cluster of worker nodes, a single CernVM-FS client can be exported via NFS [CPS95, SCR+03] to these worker nodes. This mode of deployment will inevitably introduce a performance bottleneck and a single point of failure and should be only used if necessary.

NFS export requires Linux kernel >= 2.6.27 on the NFS server. It works for Scientific Linux 6 but not for Scientific Linux 5. NFS clients can run both SL5 and SL6.

For proper NFS support, set `CVMFS_NFS_SOURCE=yes`. When this option is enabled, upon mount an additionally directory nfs_maps.$repository_name appears in the CernVM-FS cache directory. These *NFS maps* store the virtual inode CernVM-FS issues for any accessed path. The virtual inode may be requested by NFS clients anytime later. As the NFS server has no control over the lifetime of client caches, entries in the NFS maps cannot be removed.

Typically, every entry in the NFS maps requires some 150-200 Bytes. A recursive `find` on /cvmfs/atlas.cern.ch with 25 million entries, for instance, would add up 5 GB in the cache directory. For a CernVM-FS instance that is exported via NFS, the safety margin for the NFS maps needs be taken into account. It also might be necessary to monitor the actual space consumption.

For decent performance, the amount of memory given to the meta-data cache should be increased. By default, this is 16M. It can be increased, for instance, to 256M by setting `CVMFS_MEMCACHE_SIZE` to 256. Furthermore, the maximum number of download retries should be increased to at least 2 for the NFS use case.

A sample entry /etc/exports

```
/cvmfs/atlas.cern.ch 172.16.192.0/24(ro,sync,no_root_squash,\
  no_subtree_check,fsid=101)
```

A sample entry /etc/fstab entry on a client:

```
172.16.192.210:/cvmfs/atlas.cern.ch /cvmfs/atlas.cern.ch nfs \
  nfsvers=3,noatime,ac,actimeo=60 0 0
```

3.6. Hotpatching and Reloading

By hotpatching a running CernVM-FS instance, most of the code can be reloaded without unmounting the file system. The current active code is unloaded and the code from the currently installed binaries is reloaded. Hotpatching is logged to syslog. Since CernVM-FS is re-initialized during hotpatching and configuration parameters are re-read, hotpatching can be also seen as a "reload".

Hotpatching has to be done for all repositories concurrently by

```
cvmfs_config [-c] reload
```

The optional parameter `-c` specifies if the CernVM-FS cache should be wiped out during the hotpatch. Reloading of the parameters of a specific repository can be done like

```
cvmfs_config reload atlas.cern.ch
```

In order to see the history of loaded CernVM-FS Fuse modules, run

```
cvmfs_talk hotpatch history
```

The currently loaded set of parameters can be shown by

```
cvmfs_talk parameters
```

The CernVM-FS packages use hotpatching in order to update previous versions.

3.7. Auxiliary Tools

3.7.1. cvmfs_fsck

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,

```
cvmfs_fsck -j 8 /var/lib/cvmfs/shared
```

checks all the data files and catalogs in `/var/lib/cvmfs/shared` using 8 concurrent threads. Supported options are:

`-v`	Produce more verbose output.
`-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 cache database. The database will be automatically rebuilt by CernVM-FS on next mount.

3.7.2. cvmfs_config

The `cvmfs_config` utility provides commands in order to setup the system for use with CernVM-FS.

setup The `setup` command takes care of basic setup tasks, such as creating the cvmfs user and allowing access to CernVM-FS mount points by all users.

chksetup The `chksetup` command inspects the system and the CernVM-FS configuration in /etc/cvmfs for common problems.

showconfig The `showconfig` command prints the CernVM-FS parameters for all repositories or for the specific repository given as argument.

stat The `stat` command prints file system and network statistics for currently mounted repositories.

status The `status` command shows all currently mounted repositories and the process id (PID) of the CernVM-FS processes managing a mount point.

probe The `probe` command tries to access /cvmfs/$repository for all repositories specified in `CVMFS_REPOSITORIES`.

reload The `reload` command is used to reload or hotpatch CernVM-FS instances (see Section 3.6).

unmount The `unmount` command unmounts all currently mounted CernVM-FS repositories, which will only succeed if there are no open file handles on the repositories.

wipecache The `wipecache` command tries to unmount all currenlty mounted CernVM-FS repositories and, in case of success, removes all files in the CernVM-FS cache.

bugreport The `bugreport` command creates a tarball with collected system information which helps to debug a problem (see Section 2.4).

3.7.3. cvmfs_talk

The `cvmfs_talk` command provides a way to control a currently running CernVM-FS process and to extract information about the status of the corresponding mount point. Most of the commands are for special purposes only or covered by more convenient commands, such as `cvmfs_config showconfig` or `cvmfs_config stat`. Two commands might be of particular interest though.

```
cvmfs_talk cleanup 0
```

will, without interruption of service, immediately cleanup the cache from all files that are not currently pinned in the cache.

```
cvmfs_talk internal affairs
```

prints the internal status information and performance counters. It can be helpful for performance engineering.

3.7.4. Other

Information about the current cache usage can be gathered using the `df` utility. For repositories created with the CernVM-FS 2.1 toolchain, information about the overall number of file system entries in the repository as well as the number of entries covered by currently loaded meta-data can be gathered by `df -i`.

For the Nagios[2] [SBG+08] monitoring system, a checker plugin is available under http://cernvm.cern.ch/portal/filesystem/downloads.

3.8. Debug Logs

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 a segmentation fault) are received. The watchdog writes the stack trace into syslog as well as into a file `stacktrace` in the cache directory.

In addition to the debugging hints in Section 2.4, CernVM-FS can be started in debug mode. In the debug mode, CernVM-FS will log with high verbosity which makes the debug mode unsuitable for production use. In order to turn on the debug mode, set `CVMFS_DEBUGFILE=/tmp/cvmfs.log`.

[2] http://www.nagios.org

4. Setting up a Local Squid Proxy

For clusters of nodes with CernVM-FS, we strongly recommend to setup two or more Squid[1] forward proxy servers [Gue99,FCAB00] as well. The forward proxies will reduce the latency for the local worker nodes, which is critical for cold cache performance. They also reduce the load on the Stratum 1 servers.

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 GB to 8 GB of memory cache and 50 GB to 100 GB of hard disk cache. We suggest to setup two identical Squid servers for reliability and load-balancing. Assuming the two servers are A and B, set

```
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. If you already have a *Frontier Squid*[2] [BDLW08, DL10] installed you can use it as well for CernVM-FS.

Otherwise, cache sizes and access control needs to be configured in order to use the Squid server with CernVM-FS. In order to do so, 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
```

Furthermore, Squid needs to allow access to all Stratum 1 servers. This is controlled through Squid ACLs. For the Stratum 1 servers for the cern.ch domain, add the following lines to you Squid configuration:

```
acl cvmfs dst cvmfs-stratum-one.cern.ch
acl cvmfs dst cernvmfs.gridpp.rl.ac.uk
acl cvmfs dst cvmfs.racf.bnl.gov
acl cvmfs dst cvmfs02.grid.sinica.edu.tw
```

[1] http://www.squid-cache.org
[2] http://frontier.cern.ch

```
acl cvmfs dst cvmfs.fnal.gov
acl cvmfs dst cvmfs-atlas-nightlies.cern.ch
http_access allow cvmfs
```

The SQUID configuration can be verified by `squid -k parse`. Before the first service start, the cache space on the hard disk needs to be prepared by `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.

5. Creating a Repository (Stratum 0)

CernVM-FS is a file system with a single source of (new) data. This single source, the repository *Stratum 0*, is maintained by a dedicated *release manager machine*. A read-writable copy of the repository is available on the release manager machine. The CernVM-FS server tool kit is used to *publish* the current state of the repository on the release manager machine. Publishing is an atomic operation.

All data stored in CernVM-FS have to be converted into a CernVM-FS *repository* during the process of publishing. The CernVM-FS repository is a form of content-addressable storage. Conversion includes creating the file catalog(s), compressing new and updated files and calculating content hashes. Storing the data in a content-addressable format results in automatic file de-duplication. It furthermore simplifies data verification and it allows for file system snapshots.

In order to provide a writable CernVM-FS repository, CernVM-FS uses a union file system that combines a read-only CernVM-FS mount point with a writable scratch area [WDG+04, Oka]. Figure 5.1 outlines the process of publishing a repository.

5.1. Publishing a new Repository Revision

Since the repositories may contain many file system objects[1], we cannot afford to generate an entire repository from scratch for every update. Instead, we add a writable file system layer on top of a mounted read-only CernVM-FS repository using the union file system AUFS [Oka]. This renders a read-only CernVM-FS mount point writable to the user, while all performed changes are stored in a special writable scratch area managed by AUFS. A similar approach is used by Linux Live Distributions that are shipped on read-only media, but allow *virtual* editing of files where changes are stored on a RAM disk.

If a file in the CernVM-FS repository gets changed, AUFS first copies it to the writable volume and applies any changes to this copy (copy-on-write semantics). AUFS will put newly created files or directories in the writable volume as well. Additionally it creates special hidden files (called *white-outs*) to keep track of file deletions in the CernVM-FS repository.

Eventually, all changes applied to the repository are stored in AUFS's scratch area and can be merged into the actual CernVM-FS repository by a subsequent

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

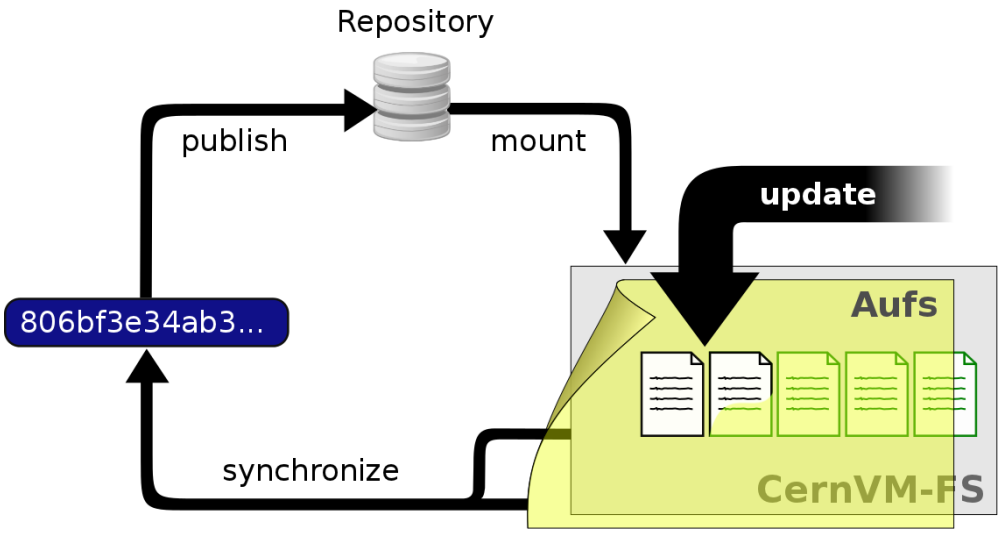

Figure 5.1.: Updating a mounted CernVM-FS repository by overlaying it with a copy-on-write AUFS volume. Any changes will be accumulated in a writable volume (yellow) and can be synchronized into the CernVM-FS repository afterwards. The file catalog contains the directory structure as well as file metadata, 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.

synchronization step. Before the actual synchronization step takes place, no changes are applied to the CernVM-FS repository. Therefore, any unsuccessful updates to a repository can be rolled back by simply clearing the writable file system layer of AUFS.

5.2. Requirements for a new Repository

In order to create a repository, the server and client part of CernVM-FS must be installed on the release manager machine. Furthermore your machine should provide an AUFS enabled Kernel as well as a running `Apache2` web server. Currently we support Scientific Linux 5 and 6 as well as Ubuntu 12.04 distributions. Please note, that Scientific Linux 6 *does not* ship with an AUFS enabled kernel, therefore we provide a compatible patched kernel as RPMs (see Appendix A).

5.3. CernVM-FS Repository Creation and Updating

The CernVM-FS server tool kit provide the versatile `cvmfs_server` utility in order to perform all operations related to repository creation, updating, deletion, replication

and inspection. Please run it without any parameters to get a short documentation of its commands.

5.3.1. Repository Creation

A new repository is created by `cvmfs_server mkfs`:

```
cvmfs_server mkfs my.repo.name
```

The utility will then ask you for a user that should act as the owner of the repository and afterwards create all the infrastructure for the new CernVM-FS repository. Additionally it will create a reasonable default configuration and generate a new release manager certificate and software signing key. You'll have to distribute the public key in /etc/cvmfs/keys/my.repo.name.pub to all your client machines. The `cvmfs_server` utility will use `/srv/cvmfs` as storage location. In case you want to use a separate hard disk, you should mount it there upfront.

Once created, you should see your repository mounted under `/cvmfs/my.repo.name` containing only a single file called `new_repository`. Following this step, you can produce the first revision by going through the repository update procedure described in the next section.

5.3.2. Repository Update

Typically a repository publisher does the following steps in order to create a new revision of a repository:

1. Run `cvmfs_server transaction` to switch to a copy-on-write enabled CernVM-FS volume
2. Make the necessary changes to the repository, e. g. add new directories, patch certain binaries, ...
3. Test the software installation
4. Do one of the following:
   - Run `cvmfs_server publish` to finalize your new repository revision *or*
   - Run `cvmfs_server abort` to clear all changes and start over again
5. Make the web server serve the new version of the repository directory.

CernVM-FS supports to have more than one repository on a single server machine. In case of a multi-repository host, you need to specify which repository you want to operate on, when running the `cvmfs_server` utility commands. Additionally you should run `cvmfs_server resign` every 30 days to update the signatures of the repository.

6. Setting up a Replica Server (Stratum 1)

While a CernVM-FS Stratum 0 repository server is able to serve clients directly, a large number of clients is better be served by a set of Stratum 1 replica servers. Multiple Stratum 1 servers improve the reliability, reduce the load, and protect the Stratum 0 master copy of the repository from direct accesses. Stratum 0 server, Stratum 1 servers and the site-local proxy servers can be seen as content distribution network. Figure 6.1 shows the situation for the repositories hosted in the cern.ch domain.

A Stratum 1 server is a standard web server that uses the CernVM-FS server toolkit to create and maintain a mirror of a CernVM-FS repository served by a Stratum 0 server. To this end, the `cvmfs_server` utility provides the `add-replica` command. This command will register the Stratum 0 URL and prepare the local web server. Periodical synchronization has to be scheduled, for instance with `cron`, using the `cvmfs_server snapshot` command. The advantage over general purpose mirroring tools such as rSync is that all CernVM-FS file integrity verifications mechansims from the Fuse client are reused. Additionally, by the aid of the CernVM-FS file catalogs, the `cvmfs_server` utility knows beforehand (without remote listing) which files to transfer.

In order to prevent accidental synchronization from a repository, the Stratum 0 repository maintainer has to create a `.cvmfs_master_replica` file in the HTTP root directory. Also keep in mind that replication will thrash any caches that might be between Stratum 1 and Stratum 0. A direct connection is therefore preferable.

6.1. Recommended Setup

The vast majority of HTTP requests will be served by the site's local proxy servers. Being a publicly available service, however, we recommend to install caching Squid frontends in front of the Stratum 1 web server. This setup is shown in Figure 6.2

We suggest the following key parameters:

Storage. RAID-protected storage. The `cvmfs_server` utility should have low latency to the storage because it runs a large number of system calls (`stat()`) against it. For the local storage backends ext3/4 filesystems are preferred (rather than XFS).

Web server. A standard Apache server. Directory listing is not required. In addition, it is a good practice to exclude search engines from the replica web server by an appropriate robots.txt. The work load is mainly covered by the Squid servers,

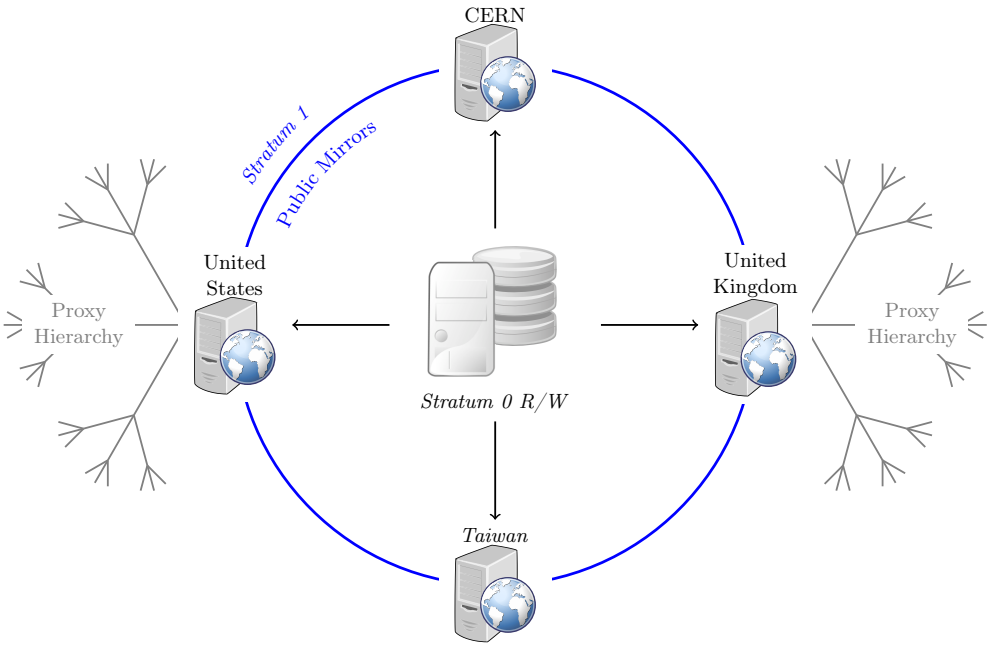

Figure 6.1.: CernVM-FS content distribution network for the cern.ch domain: Stratum 1 replica servers are located in Europe, the U.S. and Asia. One protected r/w instance (Stratum 0) is feeding up the public, distributed mirror servers. A distributed hierarchy of proxy servers fetches content from the closest public mirror server.

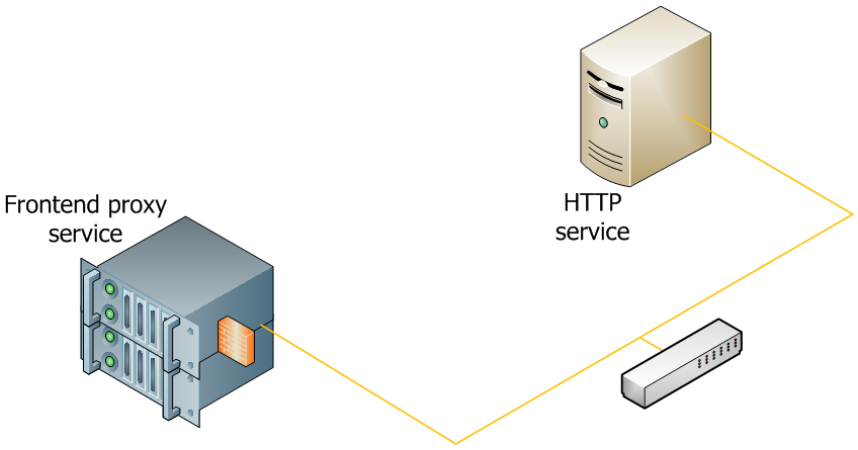

Figure 6.2.: Recommended setup: 2 DNS load-balanced (round-robin) SQUID machines in a reverse proxy mode in front of a single storage and web server.

hence performance of the web server is not critical. However, the webserver should be close to the storage in terms of latency.

Squid frontend. At least 2 SQUID servers configured in load-balance mode as reverse proxies for the web server. The Squid frontends should listen on ports 80 and 8000. The more RAM SQUID can use for caching, the better.

6.2. Squid Configuration

The SQUID configuration differs from the site-local Squids because the Stratum 1 SQUID servers are transparent to the clients (*reverse proxy*). The Squid server is configured as reverse proxy for the web server. Concurrent requests for the same URL should be collapsed into one request to the backend. Additionally, the cache sizes have to be set. As the expiry rules are set by the web server, SQUID cache expiry rules remain unchanged.

The following lines should appear accordingly in /etc/squid/squid.conf:

```
http_port 80 accel
http_port 8000 accel
http_access allow all
cache_peer <APACHE_HOSTNAME> parent 80 0 no-query originserver
collapsed_forwarding on

max_filedesc 8192

cache_mem <MEM_CACHE_SIZE> MB
```

```
cache_dir aufs /var/spool/squid <DISK_CACHE_SIZE in MB> 32 256
maximum_object_size 1024 MB
maximum_object_size_in_memory 8 MB
```

Note that `http_access allow all` has to be inserted before (or instead of) the line `http_access deny all`.

Check the configuration syntax by `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.

6.3. Monitoring

The `cvmfs_server` utility reports status and problems to `stdout` and `stderr`.

For the infrastructure, standard NAGIOS HTTP checks do the job. Configure it with the URL `http://$replica-server/cvmfs/$repository_name/.cvmfspublished`. This file can also be used to monitor if the same repository revision is served by the Stratum 0 server and all the Stratum 1 servers. In order to tune the hardware and cache sizes, keep an eye on the Squid server's CPU and I/O load.

Keep an eye on HTTP 404 errors. For normal CernVM-FS traffic, such failures should not occur. Traffic from CERNVM-FS clients is marked by an `X-CVMFS2` header.

7. Implementation Notes

CernVM-FS has a modular structure and relies on several open source libraries. Figure 7.1 shows the internal building blocks of CernVM-FS. Most of these libraries are shipped with the CernVM-FS sources and are linked statically in order to facilitate debugging and to keep the system dependencies minimal.

7.1. File Catalog

A CernVM-FS repository is defined by its *file catalog*. The file catalog is an SQLite[1] database [AO10] having a single table that lists files and directories together with its metadata. The table layout is shown in Table 7.1.

In order to save space we do not store absolute paths. Instead we store MD5 [Riv92, TC11] hash values of the absolute path names. Symbolic links are kept in the catalog. Symbolic links may contain environment variables in the form `$(VAR_NAME)` that will be dynamically resolved by CernVM-FS on access. Hardlinks are emulated by CernVM-FS. The hardlink count is stored in the lower 32bit of the hardlinks field, a *hardlink group* is stored in the higher 32 bit. If the hardlink group is greater than zero, all files with the same hardlink group will get the same inode issued by the CernVM-FS Fuse client. The emulated hardlinks work within the same directory, only. The SHA-1 [?] content hash referrs to the zlib-compressed [DG96] version of the file. Flags indicate the type of an directory entry (see Table 7.1).

A file catalog contains a *time to live* (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.

7.1.1. Nested Catalogs

In order to keep catalog sizes reasonable[2], 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 7.2 shows the directory structure which we use for the ATLAS repository.

[1] https://www.sqlite.org

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

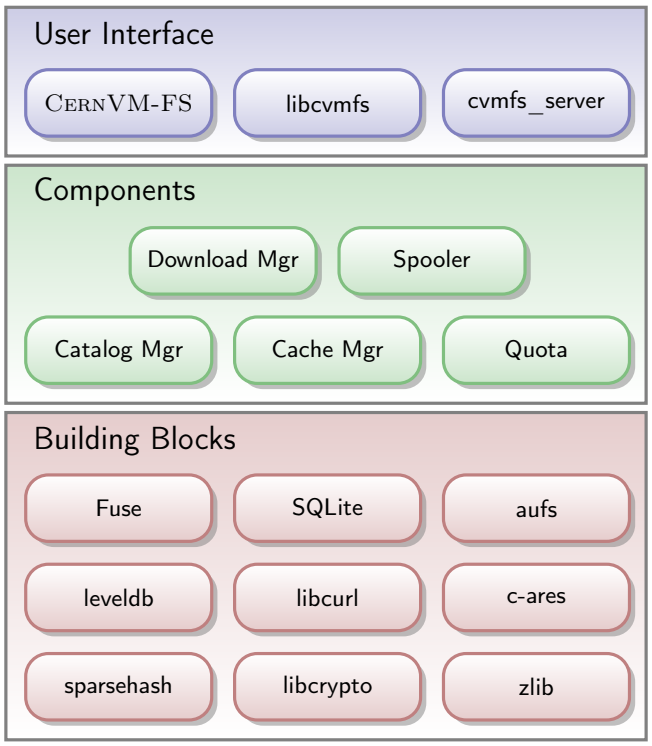

Figure 7.1.: CernVM-FS block diagram.

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

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 7.1.: Metadata information stored per directory entry. The inode is dynamically issued by CernVM-FS at runtime.

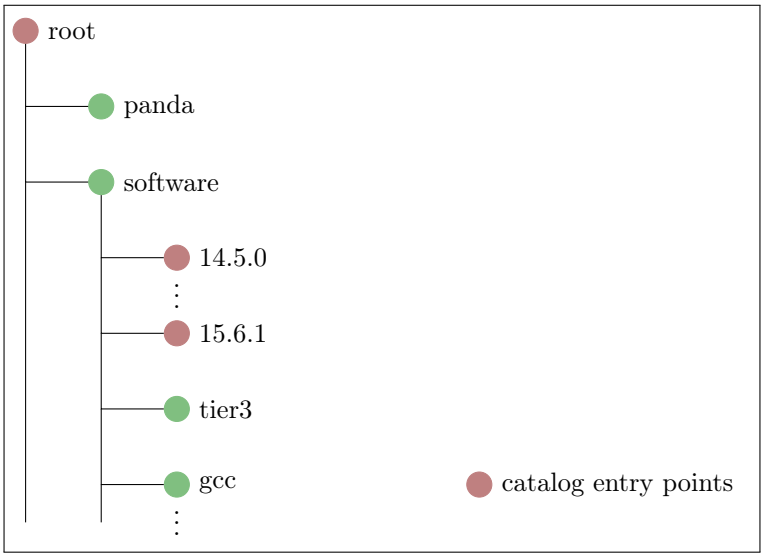

Figure 7.2.: Directory structure useds for the ATLAS repository (simplified).

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 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.

7.1.2. Catalog Statistics

A CernVM-FS file catalog maintains several counters about its contents and the contents of all of its nested catalogs. The idea is that the catalogs know how many entries there are in their sub catalogs even without opening them. This way, one can immediately tell how many entries, for instance, the entire ATLAS repository has. The numbers are shown using the number of inodes in `statvfs`. So `df -i` shows the overall number of entries in the repository and (as number of used inodes) the number of entries of currently loaded catalogs. Nested catalogs create an additional entry (the transition directory is stored in both the parent and the child catalog). File hardlinks

are still individual entries (inodes) in the cvmfs catalogs. The following counters are maintained for both a catalog itself and for the subtree this catalog is root of:

- Number of regular files
- Number of symbolic links
- Number of directories
- Number of nested catalogs

7.1.3. Catalog Signature

In order to provide authoritative information about a repository publisher, file catalogs are signed by an X.509 certificate. The CernVM-FS server tool kit uses a X.509 certificate together with its private key in order to sign a catalog and its nested catalogs. It is important to note that it is sufficient to sign just the file catalog. Since every file is listed with its SHA-1 content hash inside the catalog, we gain a secure chain and may speak of a "signed repository".

In order to validate file catalog signatures, CernVM-FS uses a white-list of valid publisher certificates. The white-list contains the SHA-1 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 *master key*. The public RSA key that corresponds to the master key is distributed with the CernVM-FS sources and the `cvmfs-keys` RPM as well as with every instance of CernVM.

In addition, CernVM-FS checks certificate fingerprints against the local blacklist /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 7.3 shows the trust chain with a signed repository.

7.2. Use of HTTP

The particular way of using the HTTP protocol has significant impact on the performance and usability of CernVM-FS. If possible, CernVM-FS tries to benefit from the HTTP/1.1 features keep-alive and cache-control. Internally, CernVM-FS uses the libcurl library [Dan].

The HTTP behaviour affects a system with cold caches only. As soon as all necessary files are cached, there is only network traffic when a catalog TTL expires.

The CernVM-FS download manager runs as a separate thread that handles download requests asynchronously in parallel. Concurrent download requests for the same URL are collapsed into a single request.

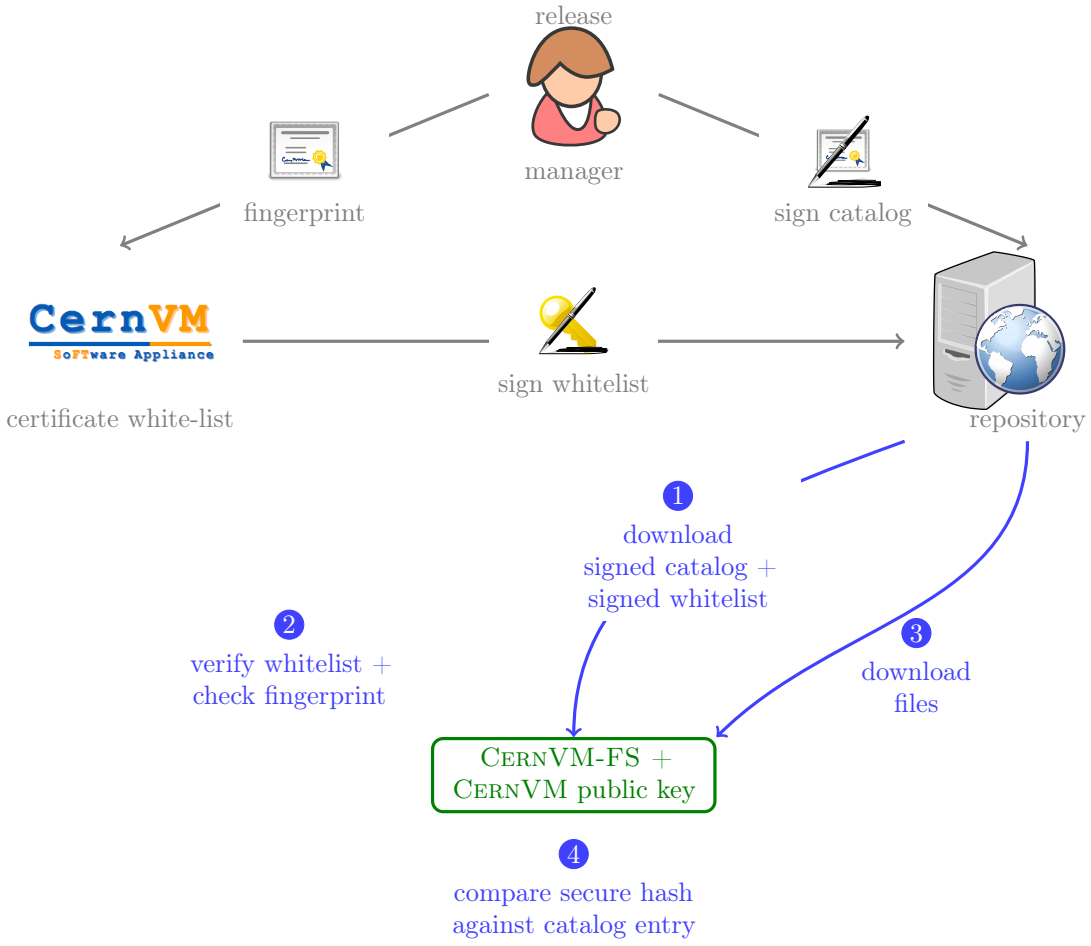

Figure 7.3.: Trust chain with a signed repository.

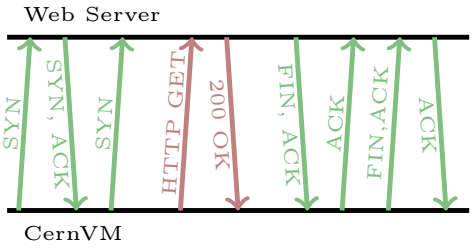

Figure 7.4.: Impact of keep-alive header on multiple file downloads.

7.2.1. DoS Protection

A subtle denial of service attack (DoS) can occur when CernVM-FS is successfully able to download a file but fails to store it in the local cache. This situation escalates into a DoS when the application using CernVM-FS remains in an endless loop and tries to open a file over and over again. Such a situation is prevented by CernVM-FS by re-trying with an exponential backoff. The backoff is triggered by consequtive filaures to cache a downloaded file within 10 seconds.

7.2.2. 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 $3 \cdot$ 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 7.4.

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 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.

7.2.3. 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).

Field	Type
SHA-1	String (hex notation)
Size	Integer
Access Sequence	Integer
Pinned	Integer
File type (chunk or file catalog)	Integer

Table 7.2.: Cache catalog table structure.

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. Only 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.

7.2.4. Identification Header

CernVM-FS sends a custom header (X−CMFS2) to be identified by the web server. If you have set the CernVM GUID, this GUID is also transmitted.

7.3. Disk Cache

Each running CernVM-FS instance requires a local cache directory. Data are downloaded into a temporary files. Only at the very latest point they are renamed into their content-addressable SHA-1 names atomically by `rename()`.

The hard disk cache is managed, i. e. 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 7.2.

CernVM-FS does not strictly enforce a the cache limit. 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 threshold. The cache threshold is currently hard-wired to half of the cache quota. 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 until unmount or until a new repository revision is applied. On unmount, pinned file catalogs are updated with the highest sequence number.

The cache catalog can be re-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, e. g. when

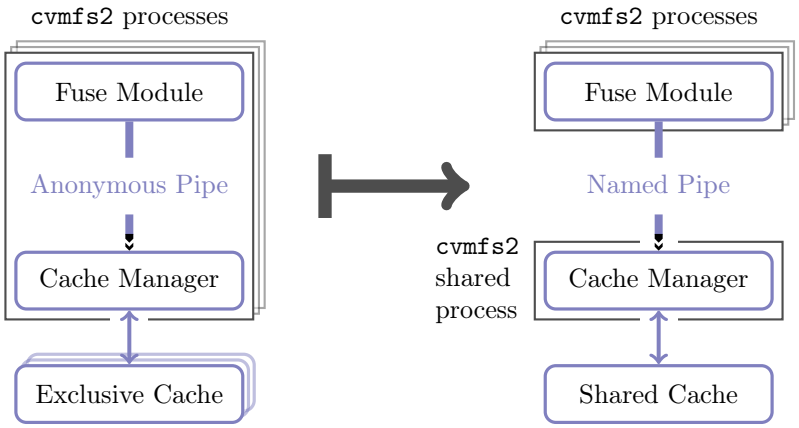

Figure 7.5.: The CernVM-FS shared local hard disk cache.

CernVM-FS was terminated unexpectedly. Re-construction has to be triggered manually.

In case of an exclusive cache, the cache manager runs as a separate thread of the `cvmfs2` process. This thread gets notified by the Fuse module whenever a file is opened or inserted. Notification is done through a pipe. The shared cache uses the very same code, except that the thread becomes a separate process (see Figure 7.5). This cache manager process is not another binary but `cvmfs2` forks to itself with special arguments, indicating that it is supposed to run as a cache manager. The cache manager does not need to be started as a service. The first CernVM-FS instance that uses a shared cache will automatically spawn the cache manager process. Subsequent CernVM-FS instances will connect to the pipe of this cache manager. Once the last CernVM-FS instance that uses the shared cache is unmounted, the communication pipe is left without any writers and the cache manager automatically quits.

7.4. 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 7.6).

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[3] and written to disk in blocks of thousands of lines, the performance overhead for tracing is low.

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

```
"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 7.6.: 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.

7.5. NFS Maps

In normal mode, CernVM-FS issues inodes based on the row number of an entry in the file catalog. When exported via NFS, this scheme can result in inconsistencies because CernVM-FS does not control the cache lifetime of NFS clients. A once issued inode can be asked for anytime later by a client. To be able to reply to such client queries even after reloading catalogs or remounts of CernVM-FS, the CernVM-FS *NFS maps* implement a persistent store of the path names $\mapsto$ inode mappings. Storing them on hard disk allows for control of the CernVM-FS memory consumption (currently $\approx$ 45 MB extra) and ensures consistency between remounts of CernVM-FS. The performance penalty for doing so is small. CernVM-FS uses Google's LevelDB [DG], a fast, local key value store. Reads and writes are only performed when meta-data are looked up in SQLite, in which case the SQLite query supposedly dominates the running time.

A drawback of the NFS maps is that there is no easy way to account for them by the cache quota. They sum up to some 150-200 Bytes per path name that has been accessed. A recursive `find` on /cvmfs/atlas.cern.ch with 25 million entries, for instance, would add up 5 GB in the cache directory. This is mitigated by the fact that the NFS mode will be only used on few servers that can be given large enough spare space on hard disk.

7.6. Loader

The CernVM-FS Fuse module comprises a minimal *loader* loader process (the `cvmfs2` binary) and a shared library containing the actual Fuse module (`libcvmfs_fuse.so`). This structure makes it possible to reload CernVM-FS code and parameters without unmounting the file system. Loader and library don't share any symbols except for two global structs `cvmfs_exports` and `loader_exports` used to call each others functions. The loader process opens the Fuse channel and implements stub Fuse callbacks that redirect all calls to the CernVM-FS shared library. Hotpatch is implemented as unloading and reloading of the shared library, while the loader temporarily queues all file system calls in-between. Among file system calls, the Fuse module has to keep

very little state. The kernel caches are drained out before reloading. Open file handles are just file descriptors that are held open by the process. Open directory listings are stored in a Google dense_hash that is saved and restored.

7.7. File 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.

7.7.1. `mount`

On mount, the file catalog has to be loaded. First, the file catalog *manifest* `.cvmfspublished` is loaded. The manifest 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 fails 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 manifest or the downloaded manifest and the cache copy differ, CernVM-FS downloads a fresh copy of the file catalog.

7.7.2. `getattr` and `lookup`

Requests for file attributes are entirely served from the mounted catalogs, 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 hash table to store negative and positive query results. The default size of 16 MB for this memory cache is determined according to compilation benchmarks.

Additionally, the 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.

7.7.3. `readlink`

A symbolic link is served from the file catalog. As a special extension, CernVM-FS detects environment variables in symlink strings written as `$(VARIABLE)`. These variables are expanded by CernVM-FS dynamically on access (in the context of the `cvmfs2` 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.

7.7.4. `readdir`

A directory listing is served by a query on the file catalog. Although the "parent"-column is indexed (cf. Table 7.1), this is a relatively slow function. We expect directory listing to happen rather seldom.

7.7.5. `open / read`

The `open()` call has to provide a file descriptor 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.

7.7.6. `getxattr`

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

expires Shows the remaining life time of the mounted root file catalog in minutes.

fqrn Shows the fully qualified repository name of the mounted repository.

hash Shows the SHA-1 hash of a regular file as listed in the file catalog.

host Shows the currently active HTTP server.

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

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

nclg Shows the number of currently loaded nested catalogs.

ndiropen Shows the overall number of opened directories.

ndownload Shows the overall number of downloaded files since mounting.

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

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

pid Shows the process id of the CernVM-FS Fuse process.

proxy Shows the currently active HTTP proxy.

revision Shows the file catalog revision of the mounted root catalog, an auto-increment counter increased on every repository publish.

root_hash Shows the SHA-1 hash of the root file catalog.

rx Shows the overall amount of downloaded kilobytes.

speed Shows the average download speed.

timeout Shows the timeout for proxied connections in seconds.

timeout_direct Shows the timeout for direct connections in seconds.

uptime Shows the time passed since mounting in minutes.

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

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

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.

7.8. Repository Publishing

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 not to re-process an entire repository on every update, we create a read-write file system interface to a CernVM-FS repository where all changes are written into a distinct scratch area.

7.8.1. Read-write Interface using a Union File System

Union file systems combine several directories into one virtual file system that provides the view of merging these directories. These underlying directories are often called *branches*. Branches are ordered; in the case of operations on paths that exist in multiple branches, the branch selection is well-defined. By stacking a read-write branch on top of a read-only branch, union file systems can provide the illusion of a read-write file system for a read-only file system. All changes are in fact written to the read-write branch.

Preserving POSIX semantics in union file systems is non-trivial; the first fully functional implementation has been presented by Wright et al. [WDG+04]. By now, union file systems are well established for "Live CD" builders, which use a RAM disk overlay on top of the read-only system partition in order to provide the illusion of a fully read-writable system. CernVM-FS uses the AUFS union file system. Another union file system with similar semantics can be plugged in if necessary.

Union file systems can be used to track changes on CernVM-FS repositories (Figure 7.8.1). In this case, the read-only file system interface of CernVM-FS is used in conjunction with a writable scratch area for changes.

Based on the read-write interface to CernVM-FS, we create a feed-back loop that represents the addition of new software releases to a CernVM-FS repository. A repository

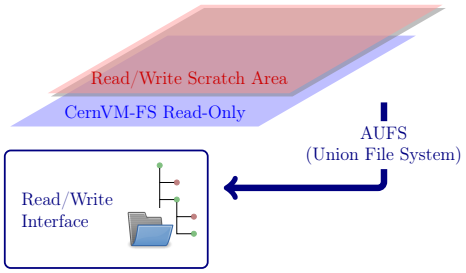

Figure 7.7.: A union file system combines a CernVM-FS read-only mount point and a writable scratch area. It provides the illusion of a writable CernVM-FS mount point, tracking changes on the scratch area.

in base revision r is mounted in read-write mode on the publisher's end. Changes are written to the scratch area and, once published, are re-mounted as repository revision $r + 1$. In this way, CernVM-FS provides snapshots. In case of errors, one can safely resume from a previously committed revision.

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 special settings for some of the repositories. For instance, the ATLAS nightly builds are provided by a URL other than the other repositories in cern.ch. 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-server Contains the CernVM-FS server tool kit for maintaining Stratum 0 and Stratum 1 servers. Depends on cvmfs-keys and httpd.

kernel-aufs21-$\cdots$ Scientific Linux 6 kernel with aufs. Required for SL6 based Stratum 0 servers.

B. CernVM-FS Parameters

B.1. Client parameters

Parameters recognized in configuration files under /etc/cvmfs:

Parameter	Meaning
CVMFS_AUTO_UPDATE	If set to "no", disables the automatic update of file catalogs.
CVMFS_BACKOFF_INIT	Seconds for the maximum initial backoff when retrying to download data.
CVMFS_BACKOFF_MAX	Maximum backoff in seconds when retrying to download data.
CVMFS_CACHE_BASE	Location (directory) of the CernVM-FS cache.
CVMFS_CHECK_PERMISSIONS	If set to "no", disable checking of file ownership and permissions (open all files).
CVMFS_DEBUGLOG	If set, run CernVM-FS in debug mode and write a verbose log the the specified file.
CVMFS_DEFAULT_DOMAIN	The default domain will be automatically appended to repository names when given without a domain.
CVMFS_HTTP_PROXY	Chain of HTTP proxy groups used by CernVM-FS.
CVMFS_IGNORE_SIGNATURE	When set to "yes", don't verify CernVM-FS file catalog signatures.
CVMFS_KCACHE_TIMEOUT	Timeout for path names and file attributes in the kernel file system buffers.
CVMFS_MAX_RETRIES	Maximum number of retries for a given proxy/host combination.
CVMFS_MAX_TTL	Maximum file catalog TTL in minutes. Can overwrite the TTL stored in the catalog.
CVMFS_MEMCACHE_SIZE	Size of the CernVM-FS meta-data memory cache in Megabyte.
CVMFS_NFILES	Maximum number of open file descriptors that can be used by the CernVM-FS process.
CVMFS_NFS_SOURCE	If set to "yes", act as a source for the NFS daemon (NFS export).
CVMFS_PROXY_RESET_AFTER	Delay in seconds after which CernVM-FS will retry the primary proxy group in case of a fail-over to another group.
CVMFS_PUBLIC_KEY	Colon-separated list of repository signing keys.

CVMFS_QUOTA_LIMIT	Soft-limit of the cache in Megabyte.
CVMFS_RELOAD_SOCKETS	Directory of the sockets used by the CernVM-FS loader to trigger hotpatching/reloading.
CVMFS_REPOSITORIES	Comma-separated list of fully qualified repository names that shall be mountable under /cvmfs.
CVMFS_ROOT_HASH	Hash of the root file catalog, implies CVMFS_AUTO_UPDATE=no.
CVMFS_SERVER_URL	Semicolon-separated chain of Stratum 1 servers.
CVMFS_SHARED_CACHE	If set to "no", makes a repository use an exclusive cache.
CVMFS_STRICT_MOUNT	If set to "yes", mount only repositories that are listed in CVMFS_REPOSITORIES.
CVMFS_SYSLOG_FACILITY	If set to a number between 0 and 7, uses the corresponding LOCAL*n* facility for syslog messages.
CVMFS_SYSLOG_LEVEL	If set to 1 or 2, sets the syslog level for CernVM-FS messages to LOG_DEBUG or LOG_INFO respectively.
CVMFS_TIMEOUT	Timeout in seconds for HTTP requests with a proxy server.
CVMFS_TIMEOUT_DIRECT	Timeout in seconds for HTTP requests without a proxy server.
CVMFS_TRACEFILE	If set, enables the tracer and trace file system calls to the given file.
CVMFS_USER	Sets the gid and uid mount options. Don't touch or overwrite.

B.2. Server parameters

tbd

Bibliography

[AO10] Grant Allen and Mike Owens. *The Definitive Guide to SQLite*. Apress, 2nd edition edition, 2010.

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

[BDLW08] Barry Blumenfeld, David Dykstra, Lee Lueking, and Eric Wicklund. CMS conditions data access using FroNTier. *Journal of Physics: Conference Series*, 119, 2008.

[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.

[CPS95] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. RFC 1813, Internet Engineering Task Force, June 1995.

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

[DG] Jeffrey Dean and Sanjay Ghemawat. leveldb. http://code.google.com/p/leveldb/.

[DG96] P. Deutsch and J-L. Gailly. ZLIB Compressed Data Format Specification version 3.3. RFC 1950, Internet Engineering Task Force, May 1996.

[DL10] D. Dykstra and L. Lueking. Greatly improved cache update times for conditions data with frontier/squid. *Journal of Physics: Conference Series*, 219, 2010.

[FCAB00] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[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.

[Gue99] David Guerrero. Caching the web, part 2. *Linux Journal*, (58), Feburary 1999.

[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.

[Oka] Junjiro R. Okajima. Aufs - Advanced multi layered Unification FileSystem. http://aufs.sourceforge.net/.

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

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Internet Engineering Task Force, April 1992.

[SBG+08] Max Schubert, Derrick Bennett, Jonathan Gines, Andrew Hay, and John Strand. *Nagios 3 Enterprise Network Monitoring*. Syngress, 2008.

[SCR+03] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 Protocol. RFC 3530, Internet Engineering Task Force, April 2003.

[TC11] S. Turner and L. Chen. Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms. RFC 6151, Internet Engineering Task Force, 2011.

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

[TL05] Douglas Thain and Miron Livny. Parrot: an application environment for data-intensive computing. *Scalable Computing: Practice and Experience*, 6(3):9, 18 2005.

[WDG+04] Charles P. Wright, Jay Dave, Puja Gupta, Harikesavan Krishnan, Erez Zadok, and Mohammad Nayyer Zubair. Versatility and unix semantics in a fan-out unification file system. Technical Report FSL-04-01b, Stony Brook University, 2004.