CernVM-FS Documentation

Release 2.1.20

CernVM Team

March 02, 2016

Contents

CHAPTER 1

What is CernVM-FS?

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, CernVM-FS 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 hashes.

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.

CHAPTER 2

Contents

2.1 Overview

The CernVM File System (CernVM-FS) is a read-only file system designed to deliver scientific software onto virtual machines and physical 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 *[BernersLee96] [Fielding99]* 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.

Fig. 2.1: 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.

The first implementation of CernVM-FS was based on grow-fs *[Compostella10] [Thain05]*, which was originally provided as one of the private file system options available in Parrot. Ever since the design evolved and diverged, taking into account the works on HTTP- Fuse *[Suzaki06]* and content-delivery networks *[Freedman03] [Nygren10] [Tolia03]*. Its current implementation provides the following key features:

- Use of the the Fuse kernel module that comes with in-kernel caching of file data and file attributes
- Cache quota management
- Use of a content addressable storage format resulting in immutable files and automatic file de-duplication
- Possibility to split a directory hierarchy into sub catalogs at user-defined levels
- Automatic updates of file catalogs controlled by a time to live stored inside file catalogs
- Digitally signed repositories
- Transparent file compression/decompression and transparent file chunking
- Capability to work in offline mode providing that all required files are cached
- File system versioning
- File system hotpatching
- Dynamic expansion of environment variables embedded in symbolic links
- Automatic mirror server selection based on geographic proximity
- Automatic load-balancing of proxy servers

- Support for WPAD/PAC auto-configuration of proxy servers
- Efficient replication of repositories
- Possibility to use S3 compatible storage instead of a file system as repository storage

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 is a use case general purpose distributed file systems are not optimized for. In contrast to virtual machine images or Docker images, software installed in CernVM-FS does not need to be further packaged. Instead it is distributed and versioned file-by-file. 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 *[Wright04]*. 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.

On the client, only data and metadata of the software releases that are actually used are downloaded and cached.

Fig. 2.2: Opening a file on CernVM-FS. CernVM-FS resolves the name by means of an SQLite catalog. 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.

2.2 Getting Started

This section describes how to install the CernVM-FS client. The CernVM-FS client is supported on x86, x86_64, and ARMv7 architectures running Scientific Linux 4-7, Ubuntu ≥ 12.04, SLES 11 and openSuSE 13.1, Fedora 19-21, or Mac OS X ≥ 10.8.

2.2.1 Getting the Software

The CernVM-FS source code and binary packages are available on our website. 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 and 64 bit Scientific Linux 7 are available as a yum repository. The `cvmfs-release` packages can be used to add a these yum repositories to the local yum installation. The `cvmfs-release` packages are available on our download page.

The CernVM-FS client is not relocatable and needs to be installed under /usr. On Intel architectures, it needs a gcc ≥ 4.2 compiler, on ARMv7 a gcc ≥ 4.7 compiler. In order to compile and install from sources, use the following cmake command:

```
cmake .
make
sudo make install
```

2.2.2 Installation

Linux

To install, proceed according to the following steps:

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

```
yum install cvmfs cvmfs-config-default
```

If yum does not show the latest packages, clean the yum cache by `yum clean all`. Packages can be also installed with rpm instead with the command `rpm -vi`. On Ubuntu, use `dpkg -i` on the cvmfs and cvmfs-config-default .deb packages.

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, ensure that `/cvmfs /etc/auto.cvmfs` is set in /etc/auto.master and that the autofs service is running. 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 *Network Settings*. Make sure your local proxy servers allow access to all the Stratum 1 web servers (more on *proxy configuration here*). 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`.

Mac OS X

On Mac OS X, CernVM-FS is based on OSXFuse. It is not 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
```

2.2.3 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 *[Mockapetris87]*. 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 on the fly. This directory gets automatically unmounted after some automount-defined idle time.

2.2.4 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`.

2.3 Client Configuration

2.3.1 Structure of /etc/cvmfs

The local configuration of CernVM-FS is controlled by several files in `/etc/cvmfs` listed in the table below. For every .conf file except for the files in /etc/cvmfs/default.d 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 :ref:sct_network‘), and perhaps the cache directory and the cache quota (see :ref:sct_cache‘) In a few cases, one might change a parameter for a specific domain or a specific repository, provide an exclusive cache for a specific repository (see :ref:sct_cache‘). For a list of all parameters, see Appendix "*Client parameters*".

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. The shell commands in the configuration files can use the `CVMFS_FQRN` parameter, which contains the fully qualified repository names that is being mounted.

The current working directory is set to the parent directory of the configuration file at hand.

File	Purpose
`config.sh`	Set of internal helper functions.
`default.conf`	Set of base parameters.
`default.d/$config.conf`	Adjustments to the default.conf configuration, usually installed by a cvmfs-config-... package. Read before default.local.
`domain.d/$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/`	Contains domain-specific sub directories with public keys used to verify the digital signature of file catalogs

The "Config Repository"

In addition to the local system configuration, a client can configure a dedicated "config repository". A config repository is a standard mountable CernVM-FS repository that resembles the directory structure of /etc/cvmfs. It can be used to centrally maintain the public keys and configuration of repositories that should not be distributed with rather static packages. Configuration from the config repository is overwritten by the local configuration in case of conflicts. The config repository is set by the `CVMFS_CONFIG_REPOSITORY` parameter. The default configuration sets this parameter to cvmfs-config.cern.ch.

2.3.2 Mounting

Mounting of CernVM-FS repositories is typically handled by autofs. Just by accessing a repository directory under /cvmfs (/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.cern.ch /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 is needed under several paths, use a *bind mount* or use a *private file system mount point*.

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 ignore 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 umount`.

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_CLAIM_OWNERSHIP=yes
CVMFS_RELOAD_SOCKETS=/home/user/mycache
CVMFS_SERVER_URL=http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch
CVMFS_HTTP_PROXY=DIRECT
```

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.

The private mount points can also be used to use the CernVM-FS Fuse module in case it has not been installed under /usr and /etc. If the public keys are not installed under /etc/cvmfs/keys, the directory of the keys needs to be specified in the config file by `CVMFS_KEYS_DIR=<directory>`. If the libcvmfs_fuse.so library is not installed in one of the standard search paths, the `LD_LIBRARY_PATH` variable has to be set accordingly for the `cvmfs2` command.

Docker Containers

There are two options to mount CernVM-FS in docker containers. The first option is to bind mount a mounted repository as a volume into the container. This has the advantage that the CernVM-FS cache is shared among multiple containers. The second option is to mount a repository inside a container, which requires a *privileged* container.

In both cases, autofs should not be used. Docker has often issues with autofs.

Bind mount from the host

In order to bind mount a repository from the host, turn off autofs on the host and mount the repository manually, like:

```
service autofs stop  # systemd: systemctl stop autofs
chkconfig autofs off  # systemd: systemctl disable autofs
mkdir -p /cvmfs/sft.cern.ch
mount -t cvmfs sft.cern.ch /cvmfs/sft.cern.ch
```

Start the docker container with the `-v` option to mount the CernVM-FS repository inside, like

```
docker run -i -t -v /cvmfs/sft.cern.ch:/cvmfs/sft.cern.ch centos /bin/bash
```

The `-v` option can be used multiple times with different repositories.

Mount inside a container

In order to use `mount` inside a container, the container must be started in privileged mode, like

```
docker run --privileged -i -t centos /bin/bash
```

In such a container, CernVM-FS can be installed and used the usual way provided that autofs is turned off.

2.3.3 Network Settings

CernVM-FS uses HTTP for the data transfer. Repository data can be replicated to multiple web servers and cached by standard web proxies such as Squid *[Guerrero99]*. 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. In every cluster of client machines, there should be two or more web proxy servers that CernVM-FS can use (see *Setting up a Local Squid Proxy*). 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

WPAD/PAC proxy auto configuration *[Gauthier99]*, 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.

Stratum 1 List

To 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 servers so that the closest servers are used with priority. This can be done automatically by *using geographic ordering*. Alternatively, for roaming clients (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 *Auxiliary Tools*). Otherwise, the proxy server would invalidate round trip time measurement.

The special sequence `@fqrn@` in the `CVMFS_SERVER_URL` string is replaced by fully qualified repository name (atlas.cern.cn, cms.cern.ch, ...). That allows to use the same parameter for many repositories hosted under the same domain. For instance, http://cvmfs-stratum-one.cern.ch/cvmfs/@fqrn@ can resolve to http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch, http://cvmfs-stratum-one.cern.ch/cvmfs/cms.cern.ch, and so on depending on the repository that is being mounted. The same works for the sequence `@org@` which is replaced by the unqualified repository name (atlas, cms, ...).

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*. The CernVM-FS proxies are set by the `CVMFS_HTTP_PROXY` parameter.

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]. Multiple IP addresses behind a single proxy host name (DNS *round-robin* entry) are automatically transformed into a load-balanced group. The `DIRECT` keyword for a hostname avoids using proxies. Note that the `CVMFS_HTTP_PROXY` parameter is necessary in order to mount. If you don't use proxies, set the parameter to `DIRECT`.

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.

Automatic Proxy Configuration

The proxy settings can be automatically gathered through WPAD. The special proxy server "auto" in `CVMFS_HTTP_PROXY` is resolved according to the proxy server specification loaded from a PAC file. PAC files can be on a file system or accessible via HTTP. CernVM-FS looks for PAC files in the order given by the semicolon separated URLs in the `CVMFS_PAC_URLS` environment variable. This variable defaults to http://wpad/wpad.dat.

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

The `auto` keyword used as a URL in `CVMFS_PAC_URLS` is resolved to http://wpad/wpad.dat, too, in order to be compatible with Frontier *[Blumenfeld08]*.

Fallback Proxy List

In addition to the regular proxy list set by `CVMFS_HTTP_PROXY`, a fallback proxy list is supported in `CVMFS_FALLBACK_PROXY`. The syntax of both lists is the same. The fallback proxy list is appended to the regular proxy list, and if the fallback proxy list is set, any DIRECT is removed from both lists. The automatic proxy configuration of the previous section only sets the regular proxy list, not the fallback proxy list. Also the fallback proxy list can be automatically reordered; see the next section.

Ordering of Servers according to Geographic Proximity

CernVM-FS Stratum 1 servers provide a RESTful service for geographic ordering. Clients can request *http://<HOST>/cvmfs/<FQRN>/api/v1.0/geo/<proxy_address>/<server_list>* The proxy address can be replaced by a UUID if no proxies are used, and the CernVM-FS client does that if there are no regular proxies. The server list is comma-separated. The result is an ordered list of indexes of the input host names. Use of this API can be enabled in a CernVM-FS client with `CVMFS_USE_GEOAPI=yes`. That will geographically sort both the servers set by `CVMFS_SERVER_URL` and the fallback proxies set by `CVMFS_FALLBACK_PROXY`.

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 below for more than the timeout interval. The threshold can be adjusted with the `CVMFS_LOW_SPEED_LIMIT` parameter. 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 $[1, \$\text{CVMFS\_BACKOFF\_INIT} \cdot 1000]$. With every retry, the backoff is then doubled.

2.3.4 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 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 *[Panagiotou06]*. Removal of files is performed bunch-wise until half of the maximum cache size has been freed. 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. The location of the cache directory can be set by `CVMFS_CACHE_BASE`.

On SELinux enabled systems, the cache directory and its content need to be labeled as `cvmfs_cache_t`. During the installation of CernVM-FS RPMs, this label is set for the default cache directory /var/lib/cvmfs. For other directories, the label needs to be set manually by `chcon -Rv --type=cvmfs_cache_t $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`.

Alien Cache

An "alien cache" provides the possibility to use a data cache outside the control of CernVM-FS. This can be necessary, for instance, in HPC environments where local disk space is not available or scarce but powerful cluster file systems are available. The alien cache directory is a directory in addition to the ordinary cache directory. The ordinary cache directory is still used to store control files.

The alien cache directory is set by the `CVMFS_ALIEN_CACHE` option. It can be located anywhere including cluster and network file systems. If configured, all data chunks are stored there. CernVM-FS ensures atomic access to the cache directory. It is safe to have the alien directory shared by multiple CernVM-FS processes and it is safe to unlink files from the alien cache directory anytime. The contents of files, however, must not be touched by third-party programs.

In contrast to normal cache mode where files are store in mode 0600, in the alien cache files are stored in mode 0660. So all users being part of the alien cache directory's owner group can use it.

The skeleton of the alien cache directory should be created upfront. Otherwise, the first CernVM-FS process accessing the alien cache determines the ownership. The `cvmfs2` binary can create such a skeleton using

```
cvmfs2 __MK_ALIEN_CACHE__ $alien_cachedir $owner_uid $owner_gid
```

Since the alien cache is unmanaged, there is no automatic quota management provided by CernVM-FS; the alien cache directory is ever-growing. The `CVMFS_ALIEN_CACHE` requires `CVMFS_QUOTA_LIMIT=-1` and `CVMFS_SHARED_CACHE=no`.

The alien cache might be used in combination with a special repository replication mode that preloads a cache directory (Section *Setting up a Replica Server (Stratum 1)*). This allows to propagate an entire repository into the cache of a cluster file system for HPC setups that do not allow outgoing connectivity.

2.3.5 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 *[Callaghan95] [Shepler03]* 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.

NSF export requires Linux kernel >= 2.6.27 on the NFS server. For instance, exporting works for Scientific Linux 6 but not for Scientific Linux 5. The NFS server should run a lock server as well. For proper NFS support, set `CVMFS_NFS_SOURCE=yes`. Also, autofs for CernVM-FS needs to be turned off and repositories need to be mounted manually. On the client side, all available nfs implementations should work.

In the NFS mode, upon mount an additionally directory nfs_maps.$repository_name appears in the CernVM-FS cache directory. These *NFS maps* use leveldb to 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 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.

Tuning

The default settings in CernVM-FS are tailored to the normal, non-NFS use case. For decent performance in the NFS deployment, 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.

The number of NFS daemons should be increased as well. A value of 128 NFS daemons has shown perform well. In Scientific Linux, the number of NFS daemons is set by the `RPCNFSDCOUNT` parameter in /etc/sysconfig/nfs.

The performance will benefit from large RAM on the NFS server ($\geq$ 16GB) and CernVM-FS caches hosted on an SSD hard drive.

Shared NFS Maps (HA-NFS)

As an alternative to the existing, leveldb managed NFS maps, the NFS maps can optionally be managed out of the CernVM-FS cache directory by SQLite. This allows the NFS maps to be placed on shared storage and accessed by multiple CernVM-FS NFS export nodes simultaneously for clustering and active high-availablity setups. In order to enable shared NFS maps, set `CVMFS_NFS_SHARED` to the path that should be used to host the SQLite database. If the path is on shared storage, the shared storage has to support POSIX file locks. The drawback of the SQLite managed NFS maps is a significant performance penalty which in practice can be covered by the memory caches.

Example

An example entry /etc/exports (note: the fsid needs to be different for every exported CernVM-FS repository)

```
/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 nfs4 \
  ro,ac,actimeo=60,lookupcache=all,nolock,rsize=1048576,wsize=1048576 0 0
```

2.3.6 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 loaded. 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 the package upgrade process.

2.3.7 Auxiliary Tools

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 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.

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*.

umount The `umount` 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 is an alias for `reload -c`.

bugreport The `bugreport` command creates a tarball with collected system information which helps to *debug a problem*.

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.

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 monitoring system *[Schubert08]*, a checker plugin is available on our website.

2.3.8 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 *these debugging hints*, 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`.

2.4 Setting up a Local Squid Proxy

For clusters of nodes with CernVM-FS clients, we strongly recommend to setup two or more Squid forward proxy servers 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 of memory cache and 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 *[Blumenfeld08] [Dykstra10]* 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:

```
max_filedesc 8192
maximum_object_size 1024 MB

cache_mem 128 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, egi.eu, and opensciencegrid.org domains, 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
acl cvmfs dst cvmfs.fnal.gov
acl cvmfs dst cvmfs-atlas-nightlies.cern.ch
acl cvmfs dst cvmfs-egi.gridpp.rl.ac.uk
acl cvmfs dst klei.nikhef.nl
acl cvmfs dst cvmfsrepo.lcg.triumf.ca
acl cvmfs dst cvmfsrep.grid.sinica.edu.tw
acl cvmfs dst cvmfs-s1bnl.opensciencegrid.org
acl cvmfs dst cvmfs-s1fnal.opensciencegrid.org
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.

2.5 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* or *installation box*. A read-writable copy of the repository is accessible 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 *[Wright04]*. *This figure below* outlines the process of publishing a repository.

2.5.1 CernVM-FS Server Quick-Start Guide

System Requirements

- Apache HTTP server *OR* S3 compatible storage service

- aufs union file system in the kernel (see *Installing the AUFS-enabled Kernel on Scientific Linux 6*)
- Officially supported platforms
  - Scientific Linux 5 (64 bit)
  - Scientific Linux 6 (64 bit - with custom AUFS enabled kernel - Appendix "*Available RPMs*")
  - Ubuntu 13.10 and above (64 bit - with installed AUFS kernel module)

Installation

1. Install `cvmfs` and `cvmfs-server` packages
2. Ensure enough disk space in `/var/spool/cvmfs` (>50GiB)
3. For local storage: Ensure enough disk space in `/srv/cvmfs`
4. Create a repository with `cvmfs_server mkfs` (See *Repository Creation*)

Content Publishing

1. `cvmfs_server transaction <repository name>`
2. Install content into `/cvmfs/<repository name>`
3. Create nested catalogs at proper locations
   - Create `.cvmfscatalog` files (See *Managing Nested Catalogs*) or
   - Consider using a `.cvmfsdirtab` file (See *Managing Nested Catalogs with .cvmfsdirtab*)
4. `cvmfs_server publish <repository name>`

Backup Policy

- Create backups of signing key files in `/etc/cvmfs/keys`
- Entire repository content
  - For local storage: `/srv/cvmfs`
  - Stratum 1s can serve as last-ressort backup of repository content

2.5.2 Installing the AUFS-enabled Kernel on Scientific Linux 6

CernVM-FS uses the union file-system aufs to efficiently determine file-system tree updates while publishing repository transactions on the server (see Figure *below*). Note that this is *only* required on a CernVM-FS server and *not* on the client machines.

We provide customised kernel packages for Scientific Linux 6 (see Appendix "*Available RPMs*") and keep them up-to-date with upstream kernel updates. The kernel RPMs are published in the `cernvm-kernel` yum repository. Please follow these steps to install the provided customised kernel:

1. Download the latest cvmfs-release package from the CernVM website
2. Install the cvmfs-release package: `yum install cvmfs-release*.rpm`

This adds the CernVM yum repositories to your machine's configuration.

3. Install the aufs enabled kernel from `cernvm-kernel`:
   `yum --disablerepo=* --enablerepo=cernvm-kernel install kernel`
4. Install the aufs user utilities:
   `yum --enablerepo=cernvm-kernel install aufs2-util`
5. Reboot the machine

Once a new kernel version is released `yum update` will *not* pick the upstream version but it will wait until the patched kernel with aufs support is published by the CernVM team. We always try to follow the kernel updates as quickly as possible.

2.5.3 Publishing a new Repository Revision

Fig. 2.3: 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.

Since the repositories may contain many file system objects, 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. 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 synchronization step. Up until 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.

2.5.4 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 6 and 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 *Installing the AUFS-enabled Kernel on Scientific Linux 6* for details).

2.5.5 Notable CernVM-FS Server Locations and Files

There are a number of possible customisations in the CernVM-FS server installation. The following table provides an overview of important configuration files and intrinsical paths together with some customisation hints. For an exhaustive description of the CernVM-FS server infrastructure please consult Appendix "*CernVM-FS Server Infrastructure*".

File Path	Description
/cvmfs	**Repository mount points** Contains read-only AUFS mountpoints that become writable during repository updates. Do not symlink or manually mount anything here.
/srv/cvmfs	**Central repository storage location** Can be mounted or symlinked to another location *before* creating the first repository.
/srv/cvmfs/<fqrn>	**Storage location of a repository** Can be symlinked to another location *before* creating the repository <fqrn>.
/var/spool/cvmfs	**Internal states of repositories** Can be mounted or symlinked to another location *before* creating the first repository. Hosts the scratch area described *here*, thus might consume notable disk space during repository updates.
/etc/cvmfs	**Configuration files and keychains** Similar to the structure described in *this table*. Do not symlink this directory.
/etc/cvmfs/cvmfs_server_hooks.sh	**Customizable server behaviour** See "*Customizable Actions Using Server Hooks*" for further details
/etc/cvmfs/repositories.d	**Repository configuration location** Contains repository server specific configuration files.

2.5.6 CernVM-FS Repository Creation and Updating

The CernVM-FS server tool kit provides the cvmfs_server utility in order to perform all operations related to repository creation, updating, deletion, replication and inspection. Without any parameters it prints a short documentation of its commands.

Repository Creation

A new repository is created by cvmfs_server mkfs:

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

The utility will ask 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. The public key in /etc/cvmfs/keys/my.repo.name.pub needs to be distributed to all client machines.

The cvmfs_server utility will use /srv/cvmfs as storage location by default. In case a separate hard disk should be used, a partition can be mounted on /src/cvmfs or /srv/cvmfs can be symlinked to another location (see *Notable CernVM-FS Server Locations and Files*). Besides local storage it is possible to use an *S3 compatible storage service* as data backend.

Once created, the repository is mounted under /cvmfs/my.repo.name containing only a single file called new_repository. The next steps describe how to change the repository content.

Repositories for Volatile Files

Repositories can be flagged as containing *volatile* files using the -v option:

```
cvmfs_server mkfs -v my.repo.name
```

When CernVM-FS clients perform a cache cleanup, they treat files from volatile repositories with priority. Such volatile repositories can be useful, for instance, for experiment conditions data.

S3 Compatible Storage Systems

CernVM-FS can store files directly to S3 compatible storage systems, such as Amazon S3, Huawei UDS and OpenStack SWIFT. The S3 storage settings are given as parameters to `cvmfs_server mkfs`:

```
cvmfs_server mkfs -s /etc/cvmfs/.../mys3.conf \
  -w http://s3.amazonaws.com/mybucket-1-1 my.repo.name
```

The file "mys3.conf" contains the S3 settings (see :ref: *table below <tab_s3confparameters>*). The "-w" option is used define the S3 server URL, e.g. http://localhost:3128, which is used for accessing the repository's backend storage on S3. Note that this URL can be different than the S3 server address that is used for uploads, e.g. if a proxy server is deployed in front of the server. Note that the buckets need to exist before the repository is created. In the example above, a single bucket `mybucket-1-1` needs to be created beforehand.

Parameter	Meaning
`CVMFS_S3_ACCOUNTS`	Number of S3 accounts to be used, e.g. 1. With some S3 servers use of multiple accounts can increase the upload speed significantly
`CVMFS_S3_ACCESS_KEY`	S3 account access key(s) separated with `:`, e.g. KEY-A:KEY-B:...
`CVMFS_S3_SECRET_KEY`	S3 account secret key(s) separated with `:`, e.g. KEY-A:KEY-B:...
`CVMFS_S3_BUCKETS_PER_ACCOUNT`	S3 buckets used per account, e.g. 1. With some S3 servers use of multiple buckets can increase the upload speed significantly
`CVMFS_S3_HOST`	S3 server hostname, e.g. s3.amazonaws.com
`CVMFS_S3_BUCKET`	S3 bucket base name. Account and bucket index are appended to the bucket base name. If you use just one account and one bucket, e.g. named `mybucket`, then you need to create only one bucket called `mybucket-1-1`
`CVMFS_S3_MAX_NUMBER_OF_PARALLEL_UPLOADS`	Number of parallel uploads to the S3 server, e.g. 400

In addition, if the S3 backend is configured to use multiple accounts or buckets, a proxy server is needed to map HTTP requests to correct buckets. This mapping is needed because CernVM-FS does not support buckets but assumes that all files are stored in a flat namespace. The recommendation is to use a Squid proxy server (version $\geq$ 3.1.10). The squid.conf can look like this:

```
http_access allow all
http_port 127.0.0.1:3128 intercept
cache_peer swift.cern.ch parent 80 0 no-query originserver
url_rewrite_program /usr/bin/s3_squid_rewrite.py
cache deny all
```

The bucket mapping logic is implemented in s3_squid_rewrite.py file. This script is not provided by CernVM-FS but needs to be written by the repository owner. The script needs to read requests from stdin and write mapped URLs to stdout, for instance:

```
in: http://localhost:3128/data/.cvmfswhitelist
out: http://swift.cern.ch/cernbucket-9-91/data/.cvmfswhitelist
```

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, add new directories, patch certain binaries, ...
3. Test the software installation
4. Do one of the following:
    - Run `cvmfs_server publish` to finalize the new repository revision *or*

- Run `cvmfs_server abort` to clear all changes and start over again

CernVM-FS supports having more than one repository on a single server machine. In case of a multi-repository host, the target repository of a command needs to be given as a parameter when running the `cvmfs_server` utility. The `cvmfs_server resign` command should run every 30 days to update the signatures of the repository. Most `cvmfs_server` commands allow for wildcards to do manipulations on more than one repository at once, `cvmfs_server migrate *.cern.ch` would migrate all present repositories ending with `.cern.ch`.

Repository Import

The CernVM-FS 2.1 server tools support the import of a CernVM-FS file storage together with its corresponding signing keychain. With `cvmfs_server import` both CernVM-FS 2.0 and 2.1 compliant repository file storages can be imported.

`cvmfs_server import` works similar to `cvmfs_server mkfs` (described in *Repository Creation*) except it uses the provided data storage instead of creating a fresh (and empty) storage. In case of a CernVM-FS 2.0 file storage `cvmfs_server import` also takes care of the file catalog migration into the CernVM-FS 2.1 schema.

Legacy Repository Import

We strongly recommend to install CernVM-FS 2.1 on a fresh or at least a properly cleaned machine without any traces of the CernVM-FS 2.0 installation before installing CernVM-FS 2.1 server tools.

The command `cvmfs_server import` requires the full CernVM-FS 2.0 data storage which is located at /srv/cvmfs by default as well as the repository's signing keys. Since the CernVM-FS 2.1 server backend supports multiple repositories in contrast to its 2.0 counterpart, we recommend to move the repository's data storage to /srv/cvmfs/<FQRN> upfront to avoid later inconsistencies.

The following steps describe the transformation of a repository from CernVM-FS 2.0 into 2.1. As an example we are using a repository called **legacy.cern.ch**.

1. Make sure that you have backups of both the repository's backend storage and its signing keys
2. Install and test the CernVM-FS 2.1 server tools on the machine that is going to be used as new Stratum 0 maintenance machine
3. Place the repository's backend storage data in /srv/cvmfs/*legacy.cern.ch*
   (default storage location)
4. Transfer the repository's signing keychain to the machine (f.e. to /legacy_keys/)
5. Run `cvmfs_server import` like this:

```
cvmfs_server import
  -o <username of repo maintainer> \
  -k ~/legacy_keys \
  -l               \ # for 2.0.x file catalog migration
  -s               \ # for further repository statistics
  legacy.cern.ch
```

6. Check the imported repository with `cvmfs_server check legacy.cern.ch` for integrity (see *Integrity Check*)

Customizable Actions Using Server Hooks

The `cvmfs_server` utility allows release managers to trigger custom actions before and after crucial repository manipulation steps. This can be useful for example for logging purposes, establishing backend storage connections

automatically or other workflow triggers, depending on the application.

There are six designated server hooks that are potentially invoked during the *repository update procedure*:

- When running `cvmfs_server transaction`:
  - *before* the given repository is transitioned into transaction mode
  - *after* the transition was successful
- When running `cvmfs_server publish`:
  - *before* the publish procedure for the given repository is started
  - *after* it was published and remounted successfully
- When running `cvmfs_server abort`:
  - *before* the unpublished changes will be erased for the given repository
  - *after* the repository was successfully reverted to the last published state

All server hooks must be defined in a single shell script file called:

```
/etc/cvmfs/cvmfs_server_hooks.sh
```

The `cvmfs_server` utility will check the existence of this script and source it. To subscribe to the described hooks one needs to define one or more of the following shell script functions:

- `transaction_before_hook()`
- `transaction_after_hook()`
- `publish_before_hook()`
- `publish_after_hook()`
- `abort_before_hook()`
- `abort_after_hook()`

The defined functions get called at the specified positions in the repository update process and are provided with the fully qualified repository name as their only parameter (`$1`). Undefined functions automatically default to a NO-OP. An example script is located at `cvmfs/cvmfs_server_hooks.sh.demo` in the CernVM-FS sources.

2.5.7 Maintaining a CernVM-FS Repository

CernVM-FS is a versioning, snapshot-based file system. Similar to versioning systems, changes to /cvmfs/...are temporary until they are committed (`cvmfs_server publish`) or discarded (`cvmfs_server abort`). That allows you to test and verify changes, for instance to test a newly installed release before publishing it to clients. Whenever changes are published (committed), a new file system snapshot of the current state is created. These file system snapshots can be tagged with a name, which makes them *named snapshots*. A named snapshot is meant to stay in the file system. One can rollback to named snapshots and it is possible, on the client side, to mount any of the named snapshots in lieu of the newest available snapshot.

Two named snapshots are managed automatically by CernVM-FS, `trunk` and `trunk-previous`. This allows for easy unpublishing of a mistake, by rolling back to the `trunk-previous` tag.

Integrity Check

CernVM-FS provides an integrity checker for repositories. It is invoked by

```
cvmfs_server check
```

The integrity checker verifies the sanity of file catalogs and verifies that referenced data chunks are present. Ideally, the integrity checker is used after every publish operation. Where this is not affordable due to the size of the repositories, the integrity checker should run regularly.

Optionally `cvmfs_server check` can also verify the data integrity (command line flag `-i`) of each data object in the repository. This is a time consuming process and we recommend it only for diagnostic purposes.

Named Snapshots

Named snapshots or *tags* are an easy way to organise checkpoints in the file system history. CernVM-FS clients can explicitly mount a repository at a specific named snapshot to expose the file system content published with this tag. It also allows for rollbacks to previously created and tagged file system revisions. Tag names need to be unique for each repository and are not allowed to contain spaces or spacial characters. Besides the actual tag's name they can also contain a free descriptive text and store a creation timestamp.

Named snapshots are best to use for larger modifications to the repository, for instance when a new major software release is installed. Named snapshots provide the ability to easily undo modifications and to preserve the state of the file system for the future. Nevertheless, named snapshots should not be used excessively. Less than 50 named snapshots are a good number of named snapshots in many cases.

By default, new repositories will automatically create a generic tag if no explicit tag is given during publish. The automatic tagging can be turned off using the -g option during repository creation or by setting `CVMFS_AUTO_TAG=false` in the /etc/cvmfs/repositories.d/$repository/server.conf file.

Creating a Named Snapshot

Tags can be added while publishing a new file system revision. To do so, the -a and -m options for `cvmfs_server publish` are used. The following command publishes a CernVM-FS revision with a new revision that is tagged as "release-1.0":

```
cvmfs_server transaction
# Changes
cvmfs_server publish -a release-1.0 -m "first stable release"
```

Managing Existing Named Snapshots

Management of existing tags is done by using the `cvmfs_server tag` command. Without any command line parameters, it will print all currently available named snapshots. Snapshots can be inspected (`-i <tag name>`), removed (`-r <tag name>`) or created (`-a <tag name> -m <tag description> -h <catalog root hash>`). Furthermore machine readable modes for both listing (`-l -x`) as well as inspection (`-i <tag name> -x`) is available.

Rollbacks

A repository can be rolled back to any of the named snapshots. Rolling back is achieved through the command `cvmfs_server rollback -t release-1.0` A rollback is, like restoring from backups, not something one would do often. Use caution, a rollback is irreversible.

Managing Nested Catalogs

CernVM-FS stores meta-data (path names, file sizes, ...) in file catalogs. When a client accesses a repository, it has to download the file catalog first and then it downloads the files as they are opened. A single file catalog for an entire repository can quickly become large and impractical. Also, clients typically do not need all of the repository's meta-data at the same time. For instance, clients using software release 1.0 do not need to know about the contents of software release 2.0.

With nested catalogs, CernVM-FS has a mechanism to partition the directory tree of a repository into many catalogs. Repository maintainers are responsible for sensible cutting of the directory trees into nested catalogs. They can do so by creating and removing magic files named `.cvmfscatalog`.

For example, in order to create a nested catalog for software release 1.0 in the hypothetical repository experiment.cern.ch, one would invoke

```
cvmfs_server transaction
touch /cvmfs/experiment.cern.ch/software/1.0/.cvmfscatalog
cvmfs_server publish
```

In order to merge a nested catalog with its parent catalog, the corresponding `.cvmfscatalog` file needs to be removed. Nested catalogs can be nested on arbitrary many levels.

Recommendations for Nested Catalogs

Nested catalogs should be created having in mind which files and directories are accessed together. This is typically the case for software releases, but can be also on the directory level that separates platforms. For instance, for a directory layout like

```
/cvmfs/experiment.cern.ch
  |- /software
  |    |- /i686
  |    |    |- 1.0
  |    |    |- 2.0
  |    `    |- common
  |    |- /x86_64
  |    |    |- 1.0
  |    `    |- common
  |- /grid-certificates
  |- /scripts
```

it makes sense to have nested catalogs at

```
/cvmfs/experiment.cern.ch/software/i686
/cvmfs/experiment.cern.ch/software/x86_64
/cvmfs/experiment.cern.ch/software/i686/1.0
/cvmfs/experiment.cern.ch/software/i686/2.0
/cvmfs/experiment.cern.ch/software/x86_64/1.0
```

A nested catalog at the top level of each software package release is generally the best approach because once package releases are installed they tend to never change, which reduces churn and garbage generated in the repository from old catalogs that have changed. In addition, each run only tends to access one version of any package so having a separate catalog per version avoids loading catalog information that will not be used. A nested catalog at the top level of each platform may make sense if there is a significant number of platform-specific files that aren't included in other catalogs.

It could also make sense to have a nested catalog under grid-certificates, if the certificates are updated much more frequently than the other directories. It would not make sense to create a nested catalog under /cvmfs/experiment.cern.ch/software/i686/common, because this directory needs to be accessed anyway whenever its

parent directory is needed. As a rule of thumb, a single file catalog should contain more than 1000 files and directories but not contain more than ≈200000 files. See *Inspecting Nested Catalog Structure* how to find catalogs that do not satisfy this recommendation.

Restructuring the repository's directory tree is an expensive operation in CernVM-FS. Moreover, it can easily break client applications when they switch to a restructured file system snapshot. Therefore, the software directory tree layout should be relatively stable before filling the CernVM-FS repository.

Managing Nested Catalogs with `.cvmfsdirtab`

Rather than managing `.cvmfscatalog` files by hand, a repository administrator may create a file called `.cvmfsdirtab`, in the top directory of the repository, which contains a list of paths relative to the top of the repository where `.cvmfscatalog` files will be created. Those paths may contain shell wildcards such as asterisk (`*`) and question mark (`?`). This is useful for specifying patterns for creating nested catalogs as new files are installed. A very good use of the patterns is to identify directories where software releases will be installed.

In addition, lines in `.cvmfsdirtab` that begin with an exclamation point (`!`) are shell patterns that will be excluded from those matched by lines without an exclamation point. For example a `.cvmfsdirtab` might contain these lines for the repository of the previous subsection:

```
/software/*
/software/*/*
! */common
/grid-certificates
```

This will create nested catalogs at

```
/cvmfs/experiment.cern.ch/software/i686
/cvmfs/experiment.cern.ch/software/i686/1.0
/cvmfs/experiment.cern.ch/software/i686/2.0
/cvmfs/experiment.cern.ch/software/x86_64
/cvmfs/experiment.cern.ch/software/x86_64/1.0
/cvmfs/experiment.cern.ch/grid-certificates
```

Note that unlike the regular lines that add catalogs, asterisks in the exclamation point exclusion lines can span the slashes separating directory levels.

Inspecting Nested Catalog Structure

The following command visualizes the current nested file catalog layout of a repository.

```
cvmfs_server list-catalogs
```

Additionally this command allows to spot degenerated nested catalogs. As stated *here* the recommended maximal file entry count of a single catalog should not exceed ≈200000. One can use the switch `list-catalogs -e` to inspect the current nested catalog entry counts in the repository. Furthermore `list-catalgos -s` will print the file sizes of the catalogs in bytes.

Migrate File Catalogs

In rare cases the further development of CernVM-FS makes it necessary to change the internal structure of file catalogs. Updating the CernVM-FS installation on a Stratum 0 machine might require a migration of the file catalogs.

It is recommended that `cvmfs_server list` is issued after any CernVM-FS update to review if any of the maintained repositories need a migration. Outdated repositories will be marked as "INCOMPATIBLE" and `cvmfs_server` refuses all actions on these repositories until the file catalogs have been updated.

In order to run a file catalog migration use `cvmfs_server migrate` for each of the outdated repositories. This will essentially create a new repository revision that contains the exact same file structure as the current revision. However, all file catalogs will be recreated from scratch using the updated internal structure. Note that historic file catalogs of all previous repository revisions stay untouched and are not migrated.

After `cvmfs_server migrate` has successfully updated all file catalogs repository maintenance can continue as usual.

2.5.8 Repository Garbage Collection

Since CernVM-FS is a versioning file system it is following an insert-only policy regarding its backend storage. When files are deleted from a CernVM-FS repository, they are not automatically deleted from the underlying storage. Therefore legacy revisions stay intact and usable forever (cf. *Named Snapshots*) at the expense of an ever-growing storage volume both on the Stratum 0 and the Stratum 1s.

For this reason, applications that frequently install files into a repository and delete older ones - for example the output from nightly software builds - might quickly fill up the repository's backend storage. Furthermore these applications might actually never make use of the aforementioned long-term revision preservation rendering most of the stored objects "garbage".

CernVM-FS supports garbage-collected repositories that automatically remove unreferenced data objects and free storage space. This feature needs to be enabled on the Stratum 0 and automatically scans the repository's catalog structure for unreferenced objects both on the Stratum 0 and the Stratum 1 installations on every publish respectively snapshot operation.

Garbage Sweeping Policy

The garbage collector of CernVM-FS is using a mark-and-sweep algorithm to detect unused files in the internal catalog graph. Revisions that are referenced by named snapshots (cf. *Named Snapshots*) or that are recent enough are preserved while all other revisions are condemned to be removed. By default this time-based threshold is *three days* but can be changed using the configuration variable `CVMFS_AUTO_GC_TIMESPAN` both on Stratum 0 and Stratum 1. The value of this variable is expected to be parseable by the `date` command, for example `3 days ago` or `1 week ago`.

Enabling Garbage Collection

Creating a Garbage Collectable Repository

Repositories can be created as *garbage-collectable* from the start by adding `-z` to the `cvmfs_server mkfs` command (cf. *Repository Creation*). It is generally recommended to also add `-g` to switch off automatic tagging in a garbage collectable repository.

Enabling Garbage Collection on an Existing Repository (Stratum 0)

Existing repositories can be reconfigured to be garbage collectable by adding `CVMFS_GARBAGE_COLLECTION=true` and `CVMFS_AUTO_GC=true` to the `server.conf` of the repository. Furthermore it is recommended to switch off automatic tagging by setting `CVMFS_AUTO_TAG=false` for a garbage collectable repository. The garbage collection will be enabled with the next published transaction.

Enabling Garbage Collection on an Existing Replication (Stratum 1)

In order to use automatic garbage collection on a stratum 1 replica `CVMFS_AUTO_GC=true` needs to be added in the `server.conf` file of the stratum 1 installation. This will only work if the upstream stratum 0 repository has garbage collection enabled.

2.5.9 Limitations on Repository Content

Because CernVM-FS provides what appears to be a POSIX filesystem to clients, it is easy to think that it is a general purpose filesystem and that it will work well with all kinds of files. That is not the case, however, because CernVM-FS is optimized for particular types of files and usage. This section contains guidelines for limitations on the content of repositories for best operation.

Data files

First and foremost, CernVM-FS is designed to distribute executable code that is shared between a large number of jobs that run together at grid sites, clouds, or clusters. Worker node cache sizes and web proxy bandwidth are generally engineered to accommodate that application. The total amount read per job is expected to be roughly limited by the amount of RAM per job slot. The same files are also expected to be read from the worker node cache multiple times for the same type of job, and read from a caching web proxy by multiple worker nodes.

If there are data files distributed by CernVM-FS that follow similar access patterns and size limits as executable code, it will probably work fine. In addition, if there are files that are larger but read slowly throughout long jobs, as opposed to all at once at the beginning, that can also work well if the same files are read by many jobs. That is because web proxies have to be engineered for handling bursts at the beginning of jobs and so they tend to be lightly loaded a majority of the time.

In general, a good rule of thumb is to calculate the maximum rate at which jobs typically start and limit the amount of data that might be read from a web proxy to per thousand jobs, assuming a reasonable amount of overlap of jobs onto the same worker nodes. Also, limit the amount of data that will be put into any one worker node cache to . Of course, if you have a special arrangement with particular sites to have large caches and bandwidths available, these limits can be made higher at those sites. Web proxies may also need to be engineered with faster disks if the data causes their cache hit ratios to be reduced.

Also, keep in mind that the total amount of data distributed is not unlimited. The files are stored and distributed compressed, and files with the same content stored in multiple places in the same repository are collapsed to the same file in storage, but the storage space is used not only on the original repository server, it is also replicated onto multiple Stratum 1 servers. Generally if only executable code is distributed, there is no problem with the space taken on Stratum 1s, but if many large data files are distributed they may exceed the Stratum 1 storage capacity. Data files also tend to not compress as well, and that is especially the case of course if they are already compressed before installation.

Tarballs, zip files, and other archive files

If the contents of a tarball, zip file, or some other type of archive file is desired to be distributed by CernVM-FS, it is usually better to first unpack it into its separate pieces first. This is because it allows better sharing of content between multiple releases of the file; some pieces inside the archive file might change and other pieces might not in the next release, and pieces that don't change will be stored as the same file in the repository. CernVM-FS will compress the content of the individual pieces, so even if there's no sharing between releases it shouldn't take much more space.

File permissions

Care should be taken to make all the files in a repository readable by "other". This is because permissions on files in the original repository are generally the same as those seen by end clients, except the files are owned by the "cvmfs" user and group. The write permissions are ignored by the client since it is a read-only filesystem. However, unless the client has set

```
CVMFS_CHECK_PERMISSIONS=no
```

(and most do not), unprivileged users will not be able to read files unless they are readable by "other" and all their parent directories have at least "execute" permissions. It makes little sense to publish files in CernVM-FS if they won't be able to be read by anyone.

Hardlinks

By default CernVM-FS does not allow hardlinks of a file to be in different directories. If there might be any such hardlinks in a repository, set the option

```
CVMFS_IGNORE_XDIR_HARDLINKS=true
```

in the repository's `server.conf`. The file will not appear to be hardlinked to the client, but it will still be stored as only one file in the repository just like any other files that have identical content. Note that if, in a subsequent publish operation, only one of these cross-directory hardlinks gets changed, the other hardlinks remain unchanged (the hardlink got "broken").

2.6 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. The *figure below* 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 mechanisms 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. This file is created by default when a new repository is created. Note that replication can thrash caches that might exist between Stratum 1 and Stratum 0. A direct connection is therefore preferable.

2.6.1 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 a Squid frontend in front of the Stratum 1 web server.

We suggest the following key parameters:

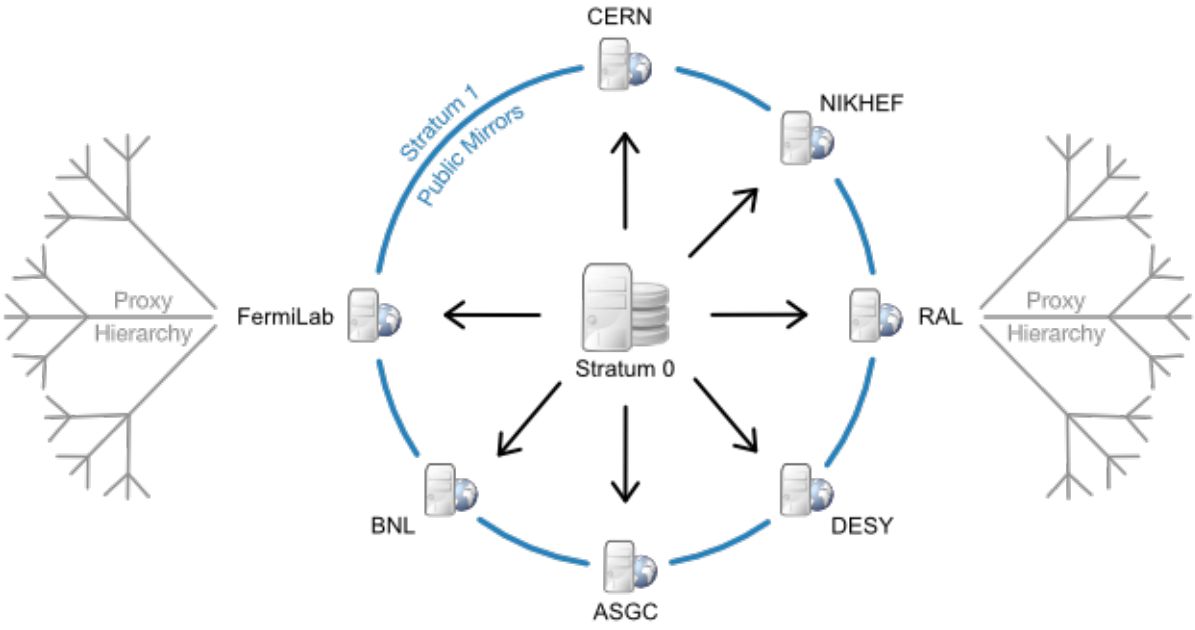

Fig. 2.4: CernVM-FS content distribution network for the cern.ch domain: Stratum1 replica servers are located in Europe, the U.S. and Asia. One protected read/write instance (Stratum 0) is feeding up the public, distributed mirror servers. A distributed hierarchy of proxy servers fetches content from the closest public mirror server.

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 webserver should be close to the storage in terms of latency.

Squid frontend Squid should be used as a frontend to Apache, configured as a reverse proxy. It is recommended to run it on the same machine as Apache to reduce the number of points of failure. Alternatively, separate Squid server machines may be configured in load-balance mode forwarding to the Apache server, but note that if any of them are down the entire service will be considered down by CernVM-FS clients. The Squid frontend should listen on ports 80 and 8000. The more RAM that the operating system can use for file system caching, the better.

DNS cache A Stratum 1 does a lot of DNS lookups, so we recommend installing a DNS caching mechanism on the machine such as `dnsmasq` or `bind`. We do not recommend `nscd` since it does not honor the DNS Time-To-Live protocol.

2.6.2 Squid Configuration

The Squid configuration differs from the site-local Squids because the Stratum 1 Squid servers are transparent to the clients (*reverse proxy*). 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 <APACHE_PORT> 0 no-query originserver
```

```
cache_mem <MEM_CACHE_SIZE> MB
cache_dir ufs /var/spool/squid <DISK_CACHE_SIZE in MB> 16 256
maximum_object_size 1024 MB
maximum_object_size_in_memory 128 KB
```

Note that `http_access allow all` has to be inserted before (or instead of) the line `http_access deny all`. If Apache is running on the same host, the `APACHE_HOSTNAME` will be `localhost`. Also, in that case there is not a performance advantage for squid to cache files that came from the same machine, so you can configure squid to not cache files. Do that with the following lines:

```
acl CVMFSAPI urlpath_regex ^/cvmfs/[^/]*/api/
cache deny !CVMFSAPI
```

Then the squid will only cache API calls. You can then set `MEM_CACHE_SIZE` and `DISK_CACHE_SIZE` quite small.

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.

2.6.3 Monitoring

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

For the web server infrastructure, we recommend standard Nagios HTTP checks. They should be configured 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.

2.7 Implementation Notes

CernVM-FS has a modular structure and relies on several open source libraries. Figure *below* 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.

2.7.1 File Catalog

A CernVM-FS repository is defined by its *file catalog*. The file catalog is an SQLite database *[Allen10]* having a single table that lists files and directories together with its metadata. The table layout is shown in the table below:

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

In order to save space we do not store absolute paths. Instead we store MD5 *[Rivest92]*, *[Turner11]* hash values of the absolute path names. Symbolic links are kept in the catalog. Symbolic links may contain environment variables in the form `$(VAR_NAME)` or `$(VAR_NAME:-/default/path)` 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 cryptographic content hash refers to the zlib-compressed *[Deutsch96]* version of the file. Flags indicate the type of an directory entry (see table *below*).

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

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 15 minutes. If a new catalog is available, CernVM-FS delays the 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.

Content Hashes

CernVM-FS can use SHA-1 *[Jones01]*, RIPEMD-160 *[Dobbertin96]* and SHAKE-128 *[Bertoni09]* as cryptographic hash function. The hash function can be changed on the Stratum 0 during the lifetime of repositories. On a change, new and updated files will use the new cryptographic hash while existing files remain unchanged. This is transparent to the clients since the hash function is stored in the flags field of file catalogs for each and every file. The default hash function is SHA-1. New software versions might introduce support for further cryptographic hash functions.

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. The figure *below* shows the simplified directory structure which we use for the ATLAS repository.

Fig. 2.5: Directory structure useds for the ATLAS repository (simplified).

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

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 cryptographic hash 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, 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.

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. Some of 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
- Number of chunked files
- Number of individual file chunks
- Overall file content size
- File content size stored in chunked files

2.7.2 Repository Manifest (.cvmfspublished)

Every CernVM-FS repository contains a repository manifest file that serves as entry point into the repository's catalog structure. The repository manifest is the first file accessed by the CernVM-FS client at mount time and therefore must be accessible via HTTP on the repository root URL. It is always called **.cvmfspublished** and contains fundamental repository meta data like the root catalog's cryptographic hash and the repository revision number as a key-value list.

Internal Manifest Structure

Below is an example of a typical manifest file. Each line starts with a capital letter specifying the meta data field, followed by the actual data string. The list of meta information is ended by a separator line (`--`) followed by signature information further described *here*.

```
C64551dccfbe0a48de7618dd7deb290200b474759
B1442336
Rd41d8cd98f00b204e9800998ecf8427e
D900
S42
Nexample.cern.ch
X731cca9476eb882f5a3f24aaa38001105a0e35eb
```

```
T1390301299
--
edde5308e502dd5e8fe405c56f5700f7477dc319
[...]
```

Please refer to table below for detailed information about each of the meta data fields.

Field	Meta Data Description
C	Cryptographic hash of the repository's current root catalog
R	MD5 hash of the repository's root path (usually always d41d8cd98f00b204e9800998ecf8427e)
B	File size of the root catalog in bytes
X	Cryptographic hash of the signing certificate
H	Cryptographic hash of the repository's named tag history database
T	Unix timestamp of this particular revision
D	Time To Live (TTL) of the root catalog
S	Revision number of this published revision
N	The full name of the manifested repository
L	currently unused (reserved for micro catalogs)

Repository Signature

In order to provide authoritative information about a repository publisher, the repository manifest is signed by an X.509 certificate together with its private key.

Signing a Repository

It is important to note that it is sufficient to sign just the manifest file itself to gain a secure chain of the whole repository. The manifest refers to the cryptographic content hash of the root catalog which in turn recursively references all sub-catalogs with their cryptographic content hashes. Each catalog lists its files along with their cryptographic content hashes. This concept is called a merkle tree and eventually provides a single hash that depends on the *complete* content of the repository.

The top level hash used for the repository signature can be found in the repository manifest right below the separator line (-- / see above). It is the cryptographic hash of the manifest's meta data lines excluding the separator line. Following the top level hash is the actual signature produced by the X.509 certificate signing procedure in binary form.

Signature Validation

In order to validate repository manifest signatures, CernVM-FS uses a white-list of valid publisher certificates. The white-list contains the cryptographic 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 cvmfs-config-\cdots RPMs 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 uses libcrypto from the OpenSSL project.

2.7.3 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.

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.

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.

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 here.

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.

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. This is not 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. 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.

Identification Header

CernVM-FS sends a custom header (`X-CVMFS2`) to be identified by the web server. If you have set the CernVM GUID, this GUID is also transmitted.

Redirects

Normally, the Stratum-1 servers directly respond to HTTP requests so CernVM-FS has no need to support HTTP redirect response codes. However, there are some high-bandwidth applications where HTTP redirects are used to transfer requests to multiple data servers. To enable support for redirects in the CernVM-FS client, set `CVMFS_FOLLOW_REDIRECTS=yes`.

2.7.4 Name Resolving

Round-robin DNS entries for proxy servers are treated specially by CernVM-FS. Multiple IP addresses for the same proxy name are automatically transformed into multiple proxy servers within the same load-balance group. So the usual rules for load-balancing and fail-over apply to the different servers in a round-robin entry. CernVM-FS resolves all the proxy servers at once (and in parallel) at mount time. From that point on, proxy server names are resolved on demand, when a download takes place and the TTL of the active proxy expired. CernVM-FS resolves using /etc/host (resp. the file referenced in the `HOST_ALIASES` environment variable) or, if a host name is not resolvable locally, it uses the c-ares resolver. Proxy servers given in IP notation remain unchanged.

CernVM-FS uses the TTLs that come from DNS servers. However, there is a cutoff at 1 minute minimum TTL and 1 day maximum TTL. Locally resolved host names get a TTL of 1 minute. The host alias file is re-read with every attempt to resolve a name. Failed attempts to resolve a name remain cached for 1 minute, too. If a name has been successfully resolved previously, this result stays active until another successful attempt is done. If the DNS entries change for a host name, CernVM-FS adjust the corresponding load-balance group and picks a new server from the group at random.

The name resolving silently ignores errors in individual records. Only if no valid IP address is returned at all it counts as an error. IPv4 addresses have precedence if available. If the `CVMFS_IPV4_ONLY` environment variable is set,CernVM-FS does not try to resolve IPv6 records.

The timeout for name resolving is hard-coded to 2 attempts with a timeout of 3 seconds each. This is independent from the `CVMFS_TIMEOUT` and `CVMFS_TIMEOUT(_DIRECT)` settings. The effective timeout can be a bit longer than 6 seconds because of a backoff.

The name server used by CernVM-FS is looked up only once on start. If the name server changes during the life time of a CernVM-FS mount point, this change needs to be manually advertised to CernVM-FS using `cvmfs_talk nameserver set`.

2.7.5 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 names atomically by `rename()`.

The hard disk cache is managed, CernVM-FS maintains cache size restrictions and replaces files according to the least recently used (LRU) strategy *[Panagiotou06]*. 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 here:

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

CernVM-FS does not strictly enforce 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, 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. As a pre-caution against a cache that is blocked by pinned catalogs, all catalogs except the root catalog are unpinned when the volume of pinned catalogs exceeds of the overall cache volume.

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, when CernVM-FS was terminated unexpectedly.

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 *below*). 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.

The CernVM-FS cache supports two classes of files with respect to the cache replacement strategy: *normal* files and *volatile* files. The sequence numbers of volatile files have bit 63 set. Hence they are interpreted as negative numbers and have precedence over normal files when it comes to cache cleanup. On automatic rebuild the volatile property of entries in the cache database is lost.

2.7.6 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 <https://github.com/google/leveldb>*, 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 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.

2.7.7 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.

2.7.8 File System Interface

CernVM-FS implements the following read-only file system call-backs.

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.

getattr and lookup

Requests for file attributes are entirely served from the mounted catalogs, 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 for this memory cache is determined according to benchmarks with LHC experiment software.

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, 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.

readlink

A symbolic link is served from the file catalog. As a special extension, CernVM-FS detects environment variables in symlink strings written as `$(VARIABLE)` or `$(VARIABLE:-/default/path)`. 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.

readdir

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

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.

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.

inode_max Shows the highest possible inode with the current set of loaded catalogs.

hash Shows the cryptographic hash of a regular file as listed in the file catalog.

host Shows the currently active HTTP server.

host_list Shows the ordered list of HTTP servers.

lhash Shows the cryptographic 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.

rawlink Shows unresolved variant symbolic links; only accessible as root.

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

root_hash Shows the cryptographic 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.

rawlink Shows the unresolved variant symlink.

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 cryptographic hash 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.

2.7.9 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 out of . 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.

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. *[Wright04]*. 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. OverlayFS is supported as an experimental alternative.

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

Fig. 2.6: 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.

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 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.

CHAPTER 3

Additional Information

3.1 CernVM-FS Parameters

3.1.1 Client parameters

Parameters recognized in configuration files under /etc/cvmfs:

Parameter	Meaning
CVMFS_ALIEN_CACHE	If set, use an alien cache at the given location
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_CLAIM_OWNERSHIP	If set to *yes*, allows CernVM-FS to claim ownership of files and directories.
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 witho
CVMFS_FALLBACK_PROXY	List of HTTP proxies similar to `CVMFS_HTTP_PROXY`. The fallback proxies are added t
CVMFS_FOLLOW_REDIRECTS	When set to *yes*, follow up to 4 HTTP redirects in requests.
CVMFS_HOST_RESET_AFTER	See `CVMFS_PROXY_RESET_AFTER`.
CVMFS_HTTP_PROXY	Chain of HTTP proxy groups used by CernVM-FS. Necessary. Set to `DIRECT` if you don
CVMFS_IGNORE_SIGNATURE	When set to *yes*, don't verify CernVM-FS file catalog signatures.
CVMFS_INITIAL_GENERATION	Initial inode generation. Used for testing.
CVMFS_KCACHE_TIMEOUT	Timeout for path names and file attributes in the kernel file system buffers.
CVMFS_KEYS_DIR	Directory containing *.pub files used as repository signing keys. If set, this parameter has
CVMFS_LOW_SPEED_LIMIT	Minimum transfer rate a server or proxy must provide.
CVMFS_MAX_IPADDR_PER_PROXY	Limit the number of IP addresses a proxy names resolves into. From all registered address
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_MOUNT_RW	Mount CernVM-FS as a read/write file system. Write operations will fail but this option c
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_NFS_SHARED	If set a path, used to store the NFS maps in an SQlite database, instead of the usual Levell
CVMFS_PAC_URLS	Chain of URLs pointing to PAC files with HTTP proxy configuration information.
CVMFS_PROXY_RESET_AFTER	Delay in seconds after which CernVM-FS will retry the primary proxy group in case of a
CVMFS_PUBLIC_KEY	Colon-separated list of repository signing keys.

Table 3.1 – continued from previous page

Parameter	Meaning
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 /c
CVMFS_REPOSITORY_TAG	Select a named repository snapshot that should be mounted instead of `trunk`.
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 LOCALn facility for syslog
CVMFS_SYSLOG_LEVEL	If set to 1 or 2, sets the syslog level for CernVM-FS messages to LOG_DEBUG or LOG_
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_USE_GEOAPI	Request order of Stratum 1 servers and fallback proxies via Geo-API.
CVMFS_USER	Sets the `gid` and `uid` mount options. Don't touch or overwrite.
CVMFS_USYSLOG	All messages that normally are logged to syslog are re-directed to the given file. This file

3.1.2 Server parameters

Parameter	Meaning
CVMFS_CREATOR_VERSION	The CernVM-FS version that was used to create this repository (do not change manually).
CVMFS_IGNORE_XDIR_HARDLINKS	If set to *yes*, do not abort the publish operation when cross-directory hardlinks are found. Instead automatically break the hardlinks across directories.
CVMFS_REPOSITORY_NAME	The fully qualified name of the specific repository.
CVMFS_REPOSITORY_TYPE	Defines if the repository is a master copy (*stratum0*) or a replica (*stratum1*).
CVMFS_SPOOL_DIR	Location of the upstream spooler scratch directories; the read-only CernVM-FS moint point and copy-on-write storage reside here.
CVMFS_UPSTREAM_STORAGE	Upstream spooler description defining the basic upstream storage type and configuration.
CVMFS_STRATUM0	URL of the master copy (*stratum0*) of this specific repository.
CVMFS_STRATUM1	URL of the Stratum1 HTTP server for this specific repository.
CVMFS_AUTO_REPAIR_MOUNTPOINT	Set to *true* to enable automatic recovery from bogus server mount states.
CVMFS_UNION_DIR	Mount point of the union file system between CernVM-FS and AUFS. Here, changes to the repository are performed (see *CernVM-FS Repository Creation and Updating*).
CVMFS_UNION_FS_TYPE	Defines the union file system to be used for the repository. (currently AUFS is fully supported)
CVMFS_AUFS_WARNING	Set to *false* to silence AUFS kernel deadlock warning.
CVMFS_HASH_ALGORITHM	Define which secure hash algorithm should be used by CernVM-FS for CAS objects (supported are: *sha1* and *rmd160*)
CVMFS_CATALOG_ENTRY_WARN_THRESHOLD	Threshold for catalog entry count before triggering a warning message.
CVMFS_USER	The user name that owns and manipulates the files inside the repository.
CVMFS_USE_FILE_CHUNKING	Allows backend to split big files into small chunks (*true* | *false*)
CVMFS_MIN_CHUNK_SIZE	Minimal size of a file chunk in bytes (see also *CVMFS_USE_FILE_CHUNKING*)
CVMFS_AVG_CHUNK_SIZE	Desired Average size of a file chunk in bytes (see also *CVMFS_USE_FILE_CHUNKING*)
CVMFS_MAX_CHUNK_SIZE	Maximal size of a file chunk in bytes (see also *CVMFS_USE_FILE_CHUNKING*)
CVMFS_MAXIMAL_CONCURRENT_WRITES	Maximal number of concurrently processed files during publishing.
CVMFS_NUM_WORKERS	Maximal number of concurrently downloaded files during a Stratum1 pull operation (Stratum~1 only).
CVMFS_PUBLIC_KEY	Path to the public key file of the repository to be replicated. (Stratum~1 only).
CVMFS_AUTO_TAG	Creates a generic revision tag for each published revision (if set to *true*).
CVMFS_GARBAGE_COLLECTION	Enables repository garbage collection (Stratum~0 only | if set to *true*)
CVMFS_AUTO_GC	Enables the automatic garbage collection on *publish* and *snapshot*
CVMFS_AUTO_GC_TIMESPAN	Date-threshold for automatic garbage collection (For example: *3 days ago*, *1 week ago*, ...)

3.2 CernVM-FS Server Infrastructure

This section provides technical details on the CernVM-FS server setup including the infrastructure necessary for an individual repository. It is highly recommended to first consult "*Notable CernVM-FS Server Locations and Files*" for a more general overview of the involved directory structure.

3.2.1 Prerequisites

A CernVM-FS server installation depends on the following environment setup and tools to be in place:

- aufs support in the kernel (see Section *Installing the AUFS-enabled Kernel on Scientific Linux 6*)
- Backend storage location available through HTTP
- Backend storage accessible at `/srv/cvmfs/...` (unless stored on S3)
- **cvmfs** and **cvmfs-server** packages installed

3.2.2 Local Backend Storage Infrastructure

CernVM-FS stores the entire repository content (file content and meta-data catalogs) into a content addressable storage (CAS). This storage can either be a file system at `/srv/cvmfs` or an S3 compatible object storage system (see "*S3 Compatible Storage Systems*" for details). In the former case the contents of `/srv/cvmfs` are as follows:

File Path	Description
`/srv/cvmfs`	**Central repository storage location** Can be mounted or symlinked to another location *before* creating the first repository.
`/srv/cvmfs/<fqrn>`	**Storage location of a specific repository** Can be symlinked to another location *before* creating the repository `<fqrn>`. This location needs to be both writable by the repository owner and accessible through an HTTP server.
`/srv/cvmfs/<fqrn>/.cvmfspublished`	**Manifest file of the repository** The manifest provides the entry point into the repository. It is the only file that needs to be signed by the repository's private key.
`/srv/cvmfs/<fqrn>/.cvmfswhitelist`	**List of trusted repository certificates** Contains a list of certificate fingerprints that should be allowed to sign a repository manifest (see .cvmfspublished). The whitelist needs to be signed by a globally trusted private key.
`/srv/cvmfs/<fqrn>/data`	**CAS location of the repository** Data storage of the repository. Contains catalogs, files, file chunks, certificates and history databases in a content addressable file format. This directory and all its contents need to be writable by the repository owner.
`/srv/cvmfs/<fqrn>/data/00` ... `/ff`	**Second CAS level directories** Splits the flat CAS namespace into multiple directories. First two digits of the file content hash defines the directory the remainder is used as file name inside the corresponding directory.
`/srv/cvmfs/<fqrn>/data/txn`	**CAS transaction directory** Stores partial files during creation. Once writing has completed, the file is committed into the CAS using an atomic rename operation.

3.2.3 Server Spool Area of a Repository (Stratum0)

The spool area of a repository contains transaction infrastructure and scratch area of a Stratum0 or specifically a release manager machine installation. It is always located inside `/var/spool/cvmfs` with directories for individual repositories. Note that the data volume of the spool area can grow very large for massive repository updates since it contains the writable AUFS branch and a CernVM-FS client cache directory.

File Path	Description
/var/spool/cvmfs	**CernVM-FS server spool area** Contains administrative and scratch space for CernVM-FS repositories. This directory should only contain directories corresponding to individual CernVM-FS repositories.
/var/spool/cvmfs/<fqrn>	**Individual repository spool area** Contains the spool area of an individual repository and might temporarily contain large data volumes during massive repository updates. This location can be mounted or symlinked to other locations. Furthermore it must be writable by the repository owner.
/var/spool/cvmfs/<fqrn>/cache	**CernVM-FS client cache directory** Contains the cache of the CernVM-FS client mounting the r/o branch (i.e. /var/spool/cvmfs/<fqrn>/rdonly) of the AUFS mount point located at /cvmfs/<fqrn>. The content of this directory is fully managed by the CernVM-FS client and hence must be configured as a CernVM-FS cache and writable for the repository owner.
/var/spool/cvmfs/<fqrn>/rdonly	**CernVM-FS client mount point** Serves as the mount point of the CernVM-FS client exposing the latest published state of the CernVM-FS repository. It needs to be owned by the repository owner and should be empty if CernVM-FS is not mounted to it.
/var/spool/cvmfs/<fqrn>/scratch	**Writable aufs scratch area** All file system changes applied to /cvmfs/<fqrn> during a transaction will be stored in this directory. Hence, it potentially needs to accommodate a large data volume during massive repository updates. Furthermore it needs to be writable by the repository owner.
/var/spool/cvmfs/<fqrn>/tmp	**Temporary scratch location** Some CernVM-FS server operations like publishing store temporary data files here, hence it needs to be writable by the repository owner. If the repository is idle this directory should be empty.
/var/spool/cvmfs/<fqrn>/client.local	**CernVM-FS client configuration** This contains client configuration variables for the CernVM-FS client mounted to /var/spool/cvmfs/<fqrn>/rdonly. Most notibly it needs to contain CVMFS_ROOT_HASH configured to the latest revision published in the corresponding repository. This file needs to be writable by the repository owner.

3.2.4 Repository Configuration Directory

The authoritative configuration of a CernVM-FS repository is located in /etc/cvmfs/repositories.d and should only be writable by the administrator. Furthermore the repository's keychain is located in /etc/cvmfs/keys and follows the naming convention <fqrn>.crt for the certificate, <fqrn>.key for the repository's private key and <fqrn>.pub for the public key. All of those files can be symlinked somewhere else if necessary.

File Path	Description
/etc/cvmfs/repositories.d/	**CernVM-FS server config directory** This contains the configuration directories for individual CernVM-FS repositories. Note that this path is shortened using /.../repos.d/ in the rest of this table.
/.../repos.d/<fqrn>/	**Config directory for specific repo** This contains the configuration files for one specific CernVM-FS repository server.
/.../repos.d/<fqrn>/server.conf	**Server configuration file** Authoritative configuration file for the CernVM-FS server tools. This file should only contain *valid server configuration variables* as it controls the behaviour of the CernVM-FS server operations like publishing, pulling and so forth.
/.../repos.d/<fqrn>/client.conf	**Client configuration file** Authoritative configuration file for the CernVM-FS client used to mount the latest revision of a Stratum 0 release manager machine. This file should only contain *valid client configuration variables*. This file must not exist for Stratum 1 repositories.
/.../repos.d/<fqrn>/replica.conf	**Replication configuration file** Contains configuration variables for Stratum 1 specific repositories. This file must not exist for Stratum 0 repositories.

3.2.5 Environment Setup

Apart from file and directory locations a CernVM-FS server installation depends on a few environment configurations. Most notably the possibility to access the backend storage through HTTP and to allow for mounting of both the CernVM-FS client at /var/spool/cvmfs/<fqrn>/rdonly and aufs on /cvmfs/<fqrn>.

Granting HTTP access can happen in various ways and depends on the chosen backend storage type. For an S3 hosted backend storage, the CernVM-FS client can usually be directly pointed to the S3 bucket used for storage (see "*S3 Compatible Storage Systems*" for details). In case of a local file system backend any web server can be used for this purpose. By default CernVM-FS assumes Apache and uses that automatically.

Internally the CernVM-FS server uses a SUID binary (i.e. cvmfs_suid_helper) to manipulate its mount points. This is necessary since transactional CernVM-FS commands must be accessible to the repository owner that is usually different from root. Both the mount directives for /var/spool/cvmfs/<fqrn>/rdonly and /cvmfs/<fqrn> must be placed into /etc/fstab for this reason. By default CernVM-FS uses the following entries for these mount points:

```
cvmfs2#<fqrn> /var/spool/cvmfs/<fqrn>/rdonly fuse \
allow_other,config=/etc/cvmfs/repositories.d/<fqrn>/client.conf: \
/var/spool/cvmfs/<fqrn>/client.local,cvmfs_suid 0 0

aufs_<fqrn> /cvmfs/<fqrn> aufs br=/var/spool/cvmfs/<fqrn>/scratch=rw: \
/var/spool/cvmfs/<fqrn>/rdonly=rr,udba=none,ro 0 0
```

3.3 Available RPMs

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

cvmfs-release Adds the CernVM-FS yum repository.

cvmfs-config-default Contains a configuration and public keys suitable for nodes in the Worldwide LHC Computing Grid. Provides access to repositories in the cern.ch, egi.eu, and opensciencegrid.org domains.

cvmfs-config-none Empty package to satisfy the cvmfs-config requirement of the cvmfs package without actually installing any configuration.

cvmfs Contains the Fuse module and additional client tools. It has dependencies to at least one of the cvmfs-config-··· packages.

cvmfs-devel Contains the `libcvmfs.a` static library and the `libcvmfs.h` header file for use of CernVM-FS with Parrot *[Thain05]*.

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.

kernel-:math:‘cdots‘-.aufs21 Scientific Linux 6 kernel with aufs. Required for SL6 based Stratum 0 servers.

kernel-:math:‘cdots‘-.aufs3 Scientific Linux 7 kernel with aufs. Required for SL7/CC7 based Stratum 0 servers.

cvmfs-unittests Contains the `cvmfs_unittests` binary. Only required for testing.

3.4 References

CHAPTER 4

Contact and Authors

Visit our website on cernvm.cern.ch.

Authors of this documentation:

- Jakob Blomer
- Predrag Buncic
- Dave Dykstra
- René Meusel

Bibliography

[Blumenfeld08] Blumenfeld, B. et al. 2008. CMS conditions data access using FroNTier. *Journal of Physics: Conference Series*. 119, (2008).

[Callaghan95] Callaghan, B. et al. 1995. *NFS Version 3 Protocol Specification*. Technical Report #1813. Internet Engineering Task Force.

[Gauthier99] Gauthier, P. et al. 1999. *Web proxy auto-discovery protocol*. IETF Secretariat.

[Guerrero99] Guerrero, D. 1999. Caching the web, part 2. *Linux Journal*. 58 (Feburary 1999).

[Panagiotou06] Panagiotou, K. and Souza, A. 2006. On adequate performance measures for paging. *Annual ACM Symposium on Theory Of Computing*. 38, (2006), 487-496.

[Schubert08] Schubert, M. et al. 2008. *Nagios 3 enterprise network monitoring*. Syngress.

[Shepler03] Shepler, S. et al. 2003. *Network File System (NFS) version 4 Protocol*. Technical Report #3530. Internet Engineering Task Force.

[Jones01] 3rd, D.E. and Jones, P. 2001. *US Secure Hash Algorithm 1 (SHA1)*. Technical Report #3174. Internet Engineering Task Force.

[Dobbertin96] Dobbertin, H. et al. 1996. RIPEMD-160: A strengthened version of RIPEMD. Springer. 71-82.

[Bertoni09] Bertoni, G., Daemen, J., Peeters, M. and Van Assche, G., 2009. Keccak sponge function family main document. Submission to NIST (Round 2), 3, p.30.

[Rivest92] Rivest, R. 1992. *The MD5 Message-Digest Algorithm*. Technical Report #1321. Internet Engineering Task Force.

[Turner11] Turner, S. and Chen, L. 2011. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. Technical Report #6151. Internet Engineering Task Force.

[Deutsch96] Deutsch, P. and Gailly, J.-L. 1996. *ZLIB Compressed Data Format Specification version 3.3*. Technical Report #1950. Internet Engineering Task Force.

[Allen10] Allen, G. and Owens, M. 2010. *The definitive guide to SQLite*. Apress.

[Wright04] Wright, C.P. et al. 2004. *Versatility and unix semantics in a fan-out unification file system*. Technical Report #FSL-04-01b. Stony Brook University.

[BernersLee96] Berners-Lee, T. et al. 1996. *Hypertext Transfer Protocol - HTTP/1.0*. Technical Report #1945. Internet Engineering Task Force.

[Fielding99] Fielding, R. et al. 1999. *Hypertext Transfer Protocol - HTTP/1.1*. Technical Report #2616. Internet Engineering Task Force.

[Compostella10] Compostella, G. et al. 2010. CDF software distribution on the Grid using Parrot. *Journal of Physics: Conference Series*. 219, (2010).

[Thain05] Thain, D. and Livny, M. 2005. Parrot: an application environment for data-intensive computing. *Scalable Computing: Practice and Experience*. 6, 3 (18 2005), 9.

[Suzaki06] Suzaki, K. et al. 2006. HTTP-FUSE Xenoppix. *Proc. of the 2006 linux symposium* (2006), 379-392.

[Freedman03] Freedman, M.J. and Mazières, D. 2003. Sloppy hashing and self-organizing clusters. M.F. Kaashoek and I. Stoica, eds. Springer. 45-55.

[Nygren10] Nygren, E. et al. 2010. The Akamai network: A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*. 44, 3 (2010), 2-19.

[Tolia03] Tolia, N. et al. 2003. Opportunistic use of content addressable storage for distributed file systems. *Proc. of the uSENIX annual technical conference* (2003).

[Mockapetris87] Mockapetris, P. 1987. *Domain names - implementation and specification*. Technical Report #1035. Internet Engineering Task Force.

[Dykstra10] Dykstra, D. and Lueking, L. 2010. Greatly improved cache update times for conditions data with frontier/Squid. *Journal of Physics: Conference Series*. 219, (2010).