New overlay FS features in cvmfs_server; FUSE-T on macOS

Yuriy Belikov

CernVM FS Team,

Sep 16th 2024

What is CernVM FS

The CernVM File System distributes LHC experiment software and conditions data to the world-wide LHC computing infrastructure.

- 5 billion files under management;
- 100 000 worker nodes with read-only clients installed
- Implemented as FUSE file system
- Uses a union file system: the read-only file system client is combined with a temporary scratch area to record a change set
- Relies on Overlay FS for repository updates (cvmfs_server)
- Uses content-addressable storage and Merkle trees in order to maintain file data and metadata

cvmfs_server publishing (simplified)

- Done in a transaction-based principle
  - Failed update does not break repository
- Could be executed only on dedicated publishing machines
- Utilizes overlay FS features under the hood (tracks repositories updates)
- Stores info about repo contents in SQLite file catalog database (see the file catalog table schema)
  - Stores MD5 hashes of relative entries’ paths in database
  - Stores info about file contents in a compressed (zstd) and hashed (MD5) format

CLI example of transaction on dedicated server machine:

```
[ ~ ] # cvmfs_server mkfs repo.name.org
[ ~ ] # cvmfs_server transaction repo.name.org
[ ~ ] # vim /cvmfs/repo.name.org/new_file.txt
[ ~ ] # cvmfs_server publish repo.name.org
```

What is overlay FS

- Functionality available on Linux that allows you to create a union view of multiple directories
- Particularly useful for systems where you have to maintain readonly base view while making changes that appear to be writable

Typically comprises of three layers:

- **Readonly (lower) layer**: this is the underlying filesystem that typically read-only
- **Union layer**: unified view of upper and lower layers
- **Writeable (upper) layer**: typically writeable directory; modifications to the filesystem such as adding, modifying, or deleting files are reflected in this directory

Additionally requires specifying a **working directory** which is used by OS for own needs

Mounting example:

```
mount -t overlay overlay -o\
        lowerdir=/lower,upperdir=/upper,workdir=/work  /merged
```

What is overlay FS

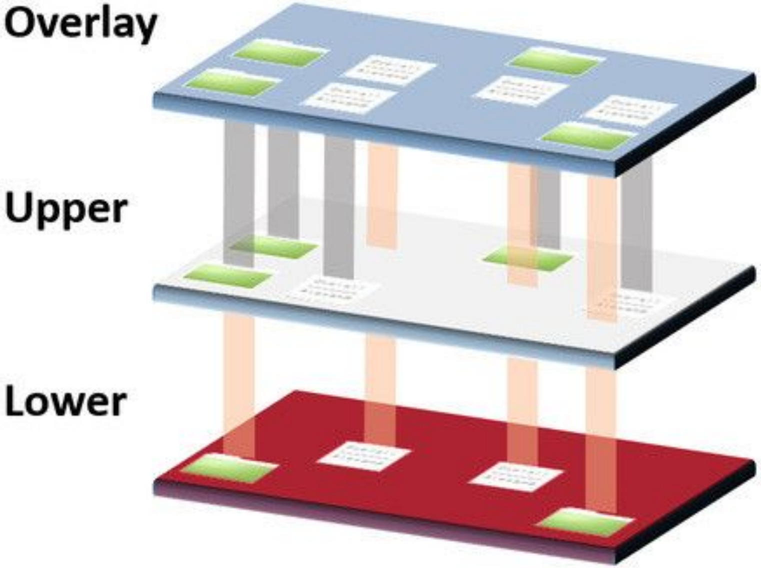


Image source:
https://commons.wikimedia.org/wiki/File:OverlayFS_Image.png

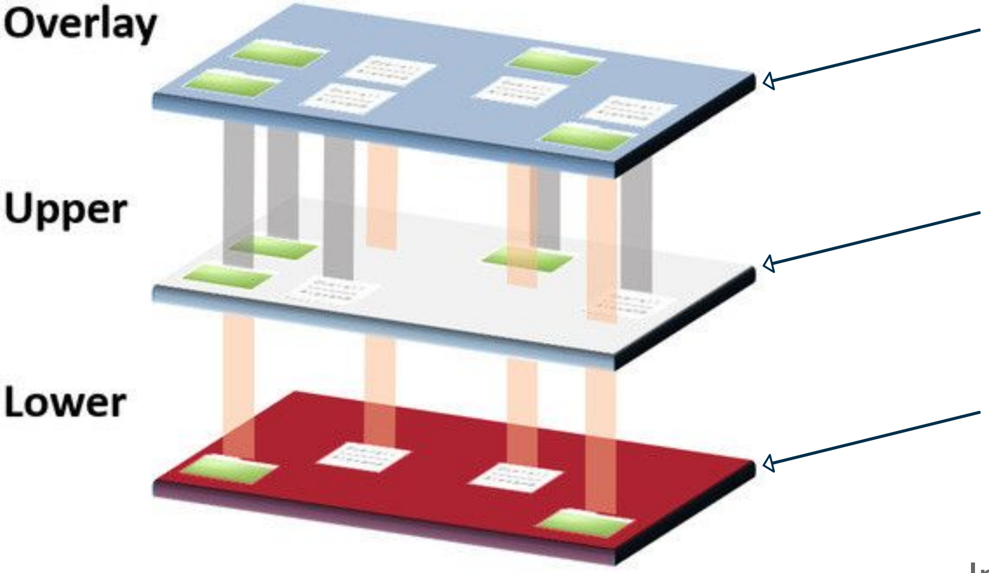
How overlay FS is utilized in cvmfs_server

Overlay FS plays important role in cvmfs_server as it allows:

- Tracking the changes made on each transaction: they are aggregated in the scratch area (which is an upper layer directory)
- Deducing the type of removed file system object in a repo (by comparing scratch area entry with the corresponding one in readonly layer)
  - Catalog database records for a directory and its nested entries are removed in a right way
- Representing a repo as union view

The last point is not as important as the other two, though. Overlay FS allows storing a previous (pre-transaction) state of a repository in /var/spool/cvmfs/<fqrn>/rdonly directory and updates made by a publisher in the “staging” (scratch) area in /var/spool/cvmfs/<fqrn>/scratch directory

How overlay FS is utilized in cvmfs_server



A repository directory (union filesystem mount point)

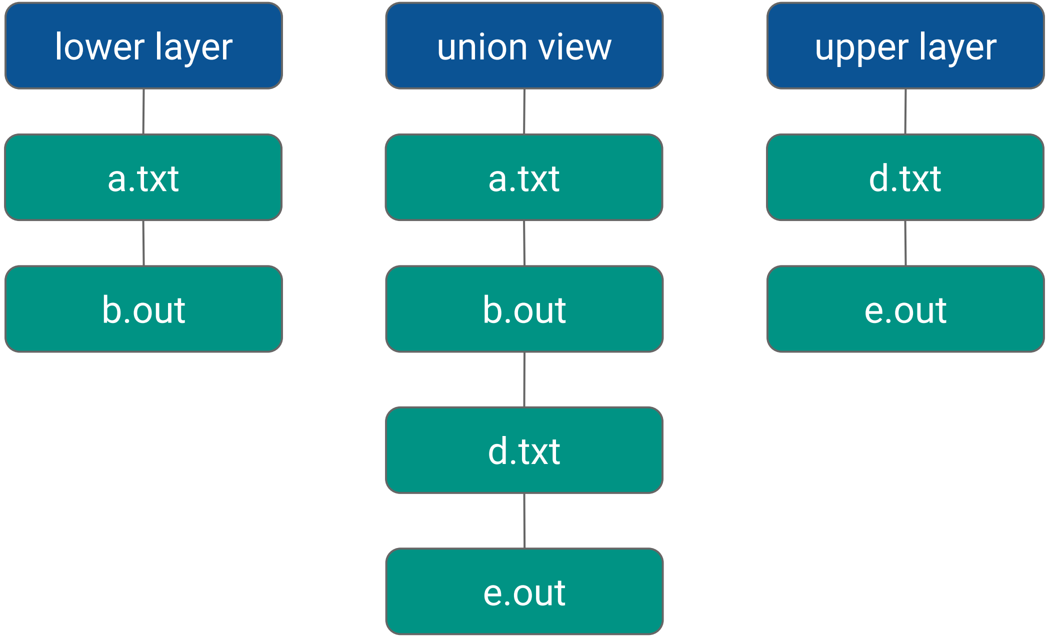
Scratch area (comprises of changes made under transaction)

cvmfs_server read-only layer

Image source

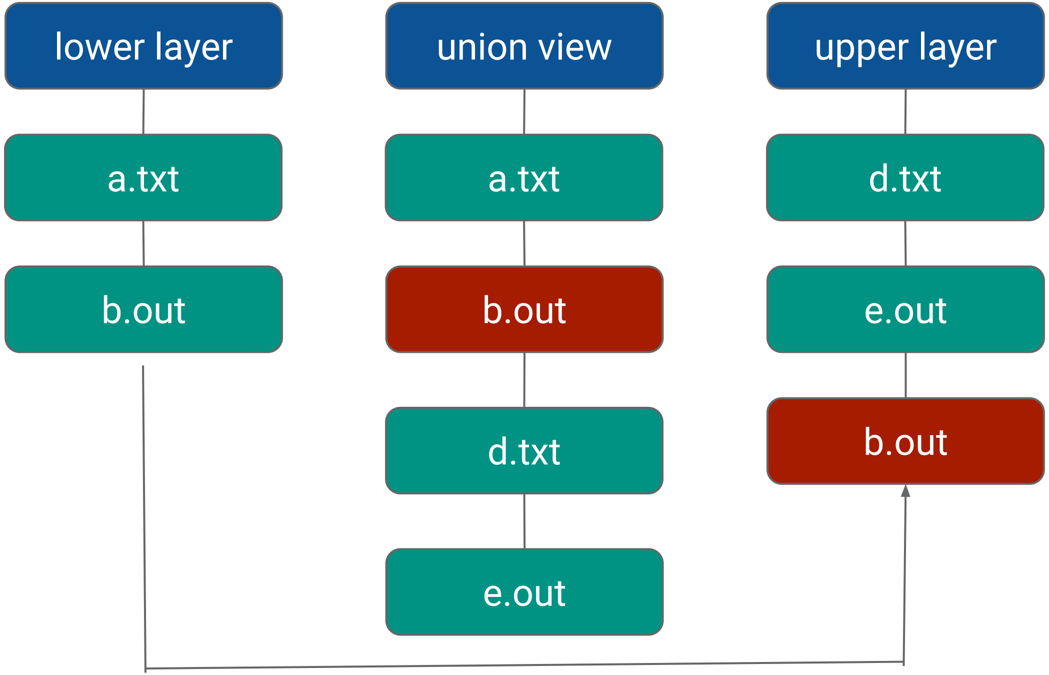
What is copy-up

Let’s consider the following setup:



What is copy-up

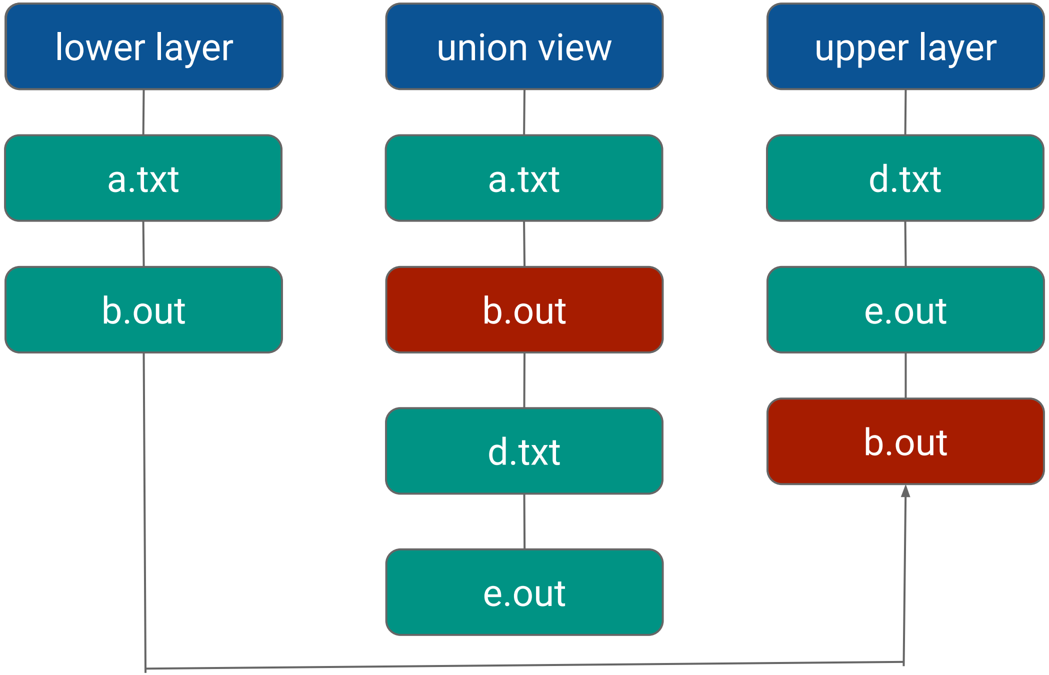
Let’s imagine that a user invokes an arbitrary metadata-modifying utility and updates metadata of **b.out**



What is copy-up

Let’s imagine that a user invokes an arbitrary metadata-modifying utility and updates metadata of **b.out**

b.out data is copied from lower layer to upper layer and that's what is called copy-up.



Metadata-only copying

Operations that utilize metadata-modifying calls actually do not affect the file content itself. When a user performs such operations during ***cmvfs_server transaction*** they involve accumulating changes in a scratch area (an upper-layer). Hence it might be beneficial to avoid copying a full file to an upper-layer and copy only metadata info instead in terms of performance considerations.

Kernel config option	CONFIG_OVERLAY_FS_METACOPY
Mount option	metacopy
Possible values	{on, off}

Mounting example: mount -t overlay overlay -o\
lowerdir=/lower,upperdir=/upper,workdir=/work,metacopy=on, redirect_dir=on /merged

Metadata-only copying. Important notes

According to OverlayFS documentation note: *redirect_dir={off|nofollow|follow[*]} and nfs_export=on mount options conflict with metacopy=on, and will result in an error.*

What is a filesystem object that gets created on the upper-layer on metadata-modifying operation?

- Regular file that contains no data (a.k.a sparse file) with an xattr **“trusted.overlay.metacopy”** that is used as an indication that this upper file contains no data and that data copy-up should still be performed before the overlayfs file is opened for write

- If you read such a file it will contain all zeroes, but those zeroes are not actually stored on disk

Directory redirect

Operations that change directory name lead to copying the whole directory with its subtree to the upper layer. So, we are dealing with the same unnecessary copy-up problem that influence operations during ***cvmfs_server transaction***.

Kernel config option	OVERLAY_FS_REDIRECT_DIR
Mount option	redirect_dir
Possible values	{on, off, follow, no_follow}

Directory redirectory. Important notes

- Renaming is not allowed by default through rename(2)
- mv(1) works by default
- After renaming a whiteout with the old name is created in the upper-layer
- “Zero-copied” directory has **trusted.overlay.redirect** extended attribute that holds the previous name of the renamed directory
- Some systems support this option by default (e.g. Manjaro Linux)

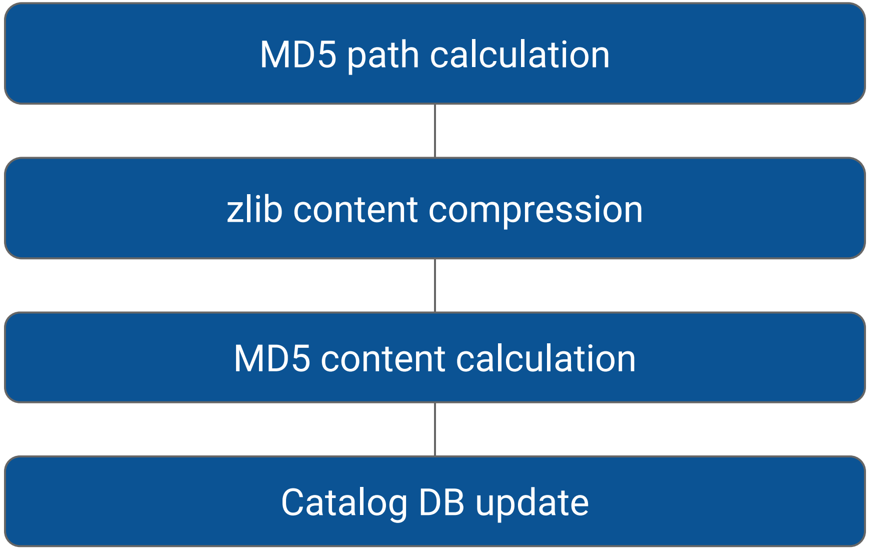
How these options could help us?

- On ***cvmfs_server publish*** command, the utility traverses scratch area and stores information about the contents of repository in file catalogs and the content itself in a compressed manner.
- Since compression, hashing and updating file catalogs is performed for full copies of the modified files modern overlay FS features have a potential to improve performance of cvmfs_server transactions via avoiding the unnecessary data copying to scratch area.

However, integrating zero-copy directory renames appeared not as trivial as was firstly expected:

- As initial logic of cvmfs transactions relies on the fact that scratch area contains full copies of file system objects zero-copy rename leads to wiping out old directory with all its contents
- Subdirectories removal creates a different footprint in the upper-layer directory: a whiteout appears instead of the removed subdirectory which is not the case for a usual setup

CVMFS server flow for FS entries



MD5 path calculation

zlib content compression

MD5 content calculation

Catalog DB update

Implementation objectives

- Enable metacopy and redirect_dir features for CVMFS repositories mounting
- Update scratch area traversal routine accordingly
- Implement catalog entries renaming (avoid remove + add sequence)
- Implement metadata-only update for catalog entries
- Expand integration tests with new cases that cover various renaming scenarios
- Cover new functionality with compile-time flag

What was done?

1. The algorithm for handling whiteouts properly

2. New integration test for checking the implemented logic

3. Metadata-only files tracking

4. Overlay FS documentation got a patch from me that elaborates existence of trusted.overlay.metacopy xattr

Unanswered questions

1. How to separate new files from updated files in renamed directories (in principle such entries are absent in ***/rdonly***)

2. Is it possible to update only file content hash instead of the whole entry in a catalog DB table?

Google Summer of Code and FUSE-T

- Currently CVMFS client for macOS fully relies upon MacFUSE module which is implemented as a kernel extension (kext)
- Apple explicitly stated that kexts are going to be deprecated in the near future
- kext requires reducing startup security level on the end-user’s side (which could be done only in recovery mode) and several reboots

Google Summer of Code and FUSE-T

- FUSE-T is a user-space library (https://www.fuse-t.org/)
- **Claimed** to be a drop-in replacement for macFUSE
- Utilizes local NFSv4 Golang server that works through a connection with the implemented filesystem process and libfuse
- **Claimed to have no significant performance drops in comparison with macFUSE**

Google Summer of Code and FUSE-T

The progress could be tracked in my PR:

What has been already done:

- CVMFS build update to overcome issues with dyld failing to find dynamic libraries
- Updated CMake build files use FUSE-T
- Updated FUSE-T installation check
- Achieved integration tests passing on a reduced tests set
- Extended GitHub Actions pipeline with updated macOS CI support

Encountered issues

- FUSE-T invokes listxattr before calling getxattr:
  - If your filesystem doesn't expect this you are in trouble: in our case magic attributes doesn't work properly.
- Hidden extended attributes are not supported; not a big issue since they are utilized by cvmfs_server (which is not supported on macOS)
- Mounting takes a time frame (usually a few seconds) long enough to fail immediate subsequent commands (such as calling ***ls <repo>*** right after mount)
- **Sometimes we get directory "loops" inside directories where usually regular files are stored: nested directory refers a parent directory**

Thank you!