CernVM-FS at Extreme Scales

Laura Promberger[1,*], *Jakob* Blomer[1], *Valentin* Völkl[1], and *Matt* Harvey[2]

[1]CERN, Switzerland
[2]Jump Trading, LLC

Abstract. The CernVM File System (CVMFS) provides the software distribution backbone for High Energy and Nuclear Physics experiments and many other scientific communities in the form of a globally available shared software area. It has been designed for the software distribution problem of experiment software for LHC Runs 1 and 2. For LHC Run 3 and even more so for HL-LHC (Runs 4-6), the complexity of the experiment software stacks and their build pipelines is substantially larger. For instance, software is being distributed for several CPU architectures, often in the form of containers which includes base and operating system libraries, the number of external packages such as machine learning libraries has multiplied, and there is a shift from C++ to more Python-heavy software stacks that results in more and smaller files needing to be distributed. For CVMFS, the new software landscape means an order of magnitude increase of scale in several key metrics. This contribution reports on the performance and reliability engineering on the file system client to sustain current and expected future software access load. Concretely, the impact of the newly designed file system cache management is shown, including significant performance improvements for HEP-representative benchmark workloads, and an up to 25% performance increase in software built-time when the build tools reside on CVMFS. Operational improvements presented include better network failure handling, error reporting, and integration with container runtimes. And a pilot study using zstd as compression algorithm shows that it could bring significant improvements for remote data access times.

1 Introduction

CernVM File System (CVMFS) [1] is a distributed read-only file system used as a software distribution backbone for High Energy and Nuclear Physics experiments and many other scientific communities [2]. It provides a globally shared software area that allows both end users and grid jobs to stream on-demand software which is then run locally. CVMFS uses aggressive data caching, deduplication and on-demand streaming to reduce network transfer sizes and access times for consecutive data requests.

CVMFS was designed for LHC Runs 1 and 2. For HL-LHC, it is expected that growth in infrastructure and increase of complexity of software stacks will lead to the CVMFS metrics growing an order of magnitude. Already at this point in time, the increase of the complexity of experiments' software stacks and build pipelines can be seen. Contrary to LHC Runs 1 and 2, the standard nowadays is that each software stack version has a build matrix consisting of

*e-mail: laura.promberger@cern.ch

multiple architectures (x86_64, aarch64, ppc64le), multiple compilers (different versions of gcc and clang), and operating systems (EL7 – EL9, Debian-based distribution, and macOS). With CVMFS being the file system of choice to distribute the software stacks, the corresponding increase of files cannot be hidden from it. Figure 1 shows this growth of files for selected CERN-hosted repositories over the last couple of years. In 2020, there were about 400 million files in the hosted repositories. Nowadays, there are nearly 1.5 billion files. This is an increase by nearly a factor of 4 within 3 years, with exponential tendencies.

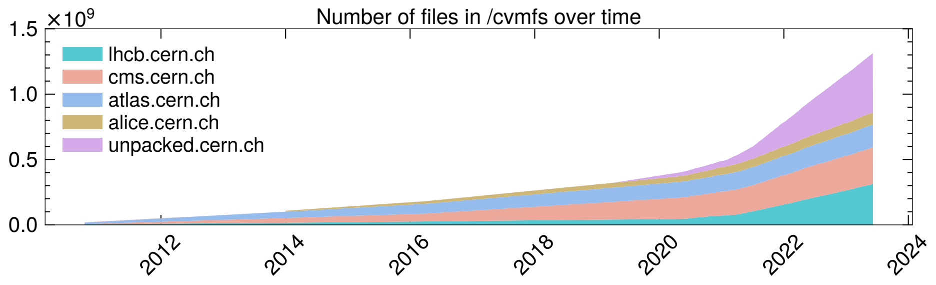


Figure 1: Increase in size of repository for selected CERN-hosted repositories.

Current CVMFS growth is also due to the automation of build pipelines and parallel publishing on CVMFS that allows for faster software release cycles. In addition, experiments tend to have now several repositories each (stable release, nightlies, ...). And, the availability of containers on CVMFS allows for more flexible usage which increases the numbers of containers being deployed as can be seen by the larger growth of `unpacked.cern.ch` compared to the other experiment-owned repositories. `unpacked.cern.ch` is a special repository that only contains container images that are synced through container registries like Harbor or GitLab.

While traditionally CVMFS has a storage backend that is directly connected to and managed by the stratum 0[1], other configurations are possible. One of them allows the data externally stored and managed, e.g. in cloud storage. CVMFS then only provides the POSIX-API as frontend access to it and stores the required metadata to be able to redirect data access requests to the external storage [3]. This is how our knowledge transfer partner Jump Trading uses CVMFS. With an CVMFS setup running already now at similar scales as anticipated for WLCG during HL-LHC, many new areas were identified of improvements and rare bugs. The Jump Trading deployment shows that CVMFS performance scales well even to very large infrastructures. At Jump Trading, CVMFS provides access to more than 100 PB of data (as of Q1 2023), although in a context slightly different from typical HEP use cases [4].

Overall, the growth of software complexity predicted for HL-LHC implies for CVMFS an increase of an order of magnitude for several key metrics. To handle this growth and the extension of its use case CVMFS has set the following goals for improvement to ensure proper quality of service at HL-LHC scale

- ***Caching performance***: Optimize application startup time for warm and hot caches
- ***Download improvements***: Optimize application startup time for cold cache
- ***Operational improvements***: Improve robustness and error diagnostics

[1]The origin server; single source of data for all CVMFS clients and replication servers

2 Improvements to CVMFS to work with extreme data scales

This section gives an overview of planned or already implemented improvements for the three work packages: caching performance, download improvements, and operational improvements.

2.1 Caching Performance

Caching performance has a huge impact on the CVMFS overall performance. CVMFS caches both data and meta-data on different cache levels: kernel cache, local worker node disk and site-wide proxy caches (squid).

Depending on from where CVMFS client retrieves the data, the following terminology will be used

- ***Cold cache***: client has no data cached on local disk
- ***Warm cache***: client has data cached on local disk
- ***Hot cache***: client has data cached on local disk and kernel cache

The recently added feature *Page Cache Tracker* allows for a much better use of kernel page cache (see section 3, Figure 4). In addition, support for in-place replacement of files was added. This prevents crashing long-running software that uses (compared to the current file system state) an old version of these files.

Before, it was needed to install immutable software stacks into their own directory each (which is still the general recommendation) in order to prevent crashes, e.g. when updating *glibc* to a new version, but some software is still running and using the old *glibc* version.

CVMFS now also supports kernel caching of resolved symlinks for `fuse3` and linux kernel 6.2. A backporting request was opened with RedHat to make this available also to older kernel versions. Furthermore, `statfs` calls can now be cached. This was added to workaround software that would call `statfs` multiple times in short succession, e.g. during startup.

Ongoing work is to provide custom load balancing policies to be able to reduce data duplication when multiple site proxies are used. These will improve cold cache access times as it is more likely that the data is already cached within the requested site proxy. Future work will focus on an efficient prefetching policy for known file clusters (e.g. Python *cluster*, ROOT *cluster*, ...) to reduce cold cache access times.

2.2 Download Improvements

This work package only consists of two items but offers a chance for large performance improvements. The first one being parallel file decompression during download. With the current implementation downloads are executed in parallel, but decompression is done sequentially [5]. This will require large refactoring work for the download part.

The second one is to introduce a new compression algorithm `zstd` [6]. This algorithm is a good replacement for `zlib` [7] as it achieves similar compression ratios while being multiple factors faster. In section 3.3 a first exploration of using `zstd` for CVMFS cached data is performed.

2.3 Operational Improvements

This section summarizes recent efforts to streamline operations and user experience, and to increase the support for automatic error detection and handling, even for rare failure cases.

Containers make-up a large portion of data on CVMFS. CVMFS provides tooling to load and extract container images from registries to a repository. As an operational improvement, it is now supported to unpack container images through Harbor registry proxies. Harbor [8] allows for offloading aspects related to image management and image checking, such as role-based access control, vulnerability scanning and to pull through docker images with safety check.

With the more varied client deployment modes, monitoring and scoping user access rights becomes more important. For this it is now possible to protect magic extended attributes[2] so that only users with specific `group ids` can access them. Furthermore, a new telemetry feature allows for the internal counters of clients to be exposed as InfluxDB format to a socket[3].

Further improvements include the speed-up of garbage collection and `cvmfs_server check`. Disk operations within those commands will only be performed exactly once per data chunk, independent of to how many files the chunk actually belongs to. For the OSG Nebraska backup stratum 1 this reduced the `cvmfs_server check` runtime from greater than 3 months to less than 3 days [9].

In addition, for future works the goal is to achieve feature parity between remote publishers (with gateway) and local publishers [10]. Right now remote publishers are limited in their functionality and actions performed on the gateway node require due diligence because no automatic locking mechanism is available to prevent remote publishers performing actions at the same time.

3 Results

This section evaluates performance benchmarks of different CVMFS versions and configurations, and it provides a comparison between compression algorithms.

For the performance benchmarks, two dedicated physical servers were used. The CVMFS client is run on a dual socket AMD EPYC 7302 server, that has 16 cores per processor and 2 threads per core, totaling up to 64 virtual cores. It has 256 GB RAM. The dedicated remote cache (squid proxy) is run on single socket Intel Core i7-7820X, with 8 cores and 2 threads per core, totaling up to 16 virtual cores. It has 32 GB RAM. Both servers were connected with a 10 Gbit link. Testing with `iperf3`[11], the connection consistently achieved a transfer rate over 9.4 Gbit/s.

Both servers used AlmaLinux9 as operating system. The CVMFS client was build from scratch, manually linked to the latest libfuse 3 (built from source) to be able to enable the symlink caching feature of CVMFS.

In total four different commands were benchmarked. The selection was based on common software used by experiments and standard workflows they are executing. All software and data is loaded from CVMFS.

- **Tensorflow**: Starting a (local) Python3 session that imports Tensorflow and numpy
- **ROOT**: Creating a RDataFrame with 100 random entries and creating a 1D histogram of it
- **DD4hep**: Load detector description into ROOT

[2]Magic extended attributes are synthetic extended attributes custom to CVMFS (e.g. client version, repository revision, proxy list, ...)

[3]Other formats can be created via a source code plugin

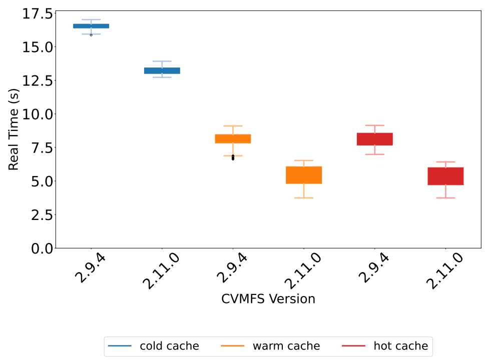
- **CMS**: TTbar_14 physics analysis of CMS. Nightly build of `CMSSW_13_1_0_pre3` was used to be able to run on EL9.

The benchmarks measure the majority of metrics provided by `time` (user time, system time, real time, page faults, ...) and save the CVMFS internal counters for the different data access patterns: cold cache, warm cache and hot cache[4].

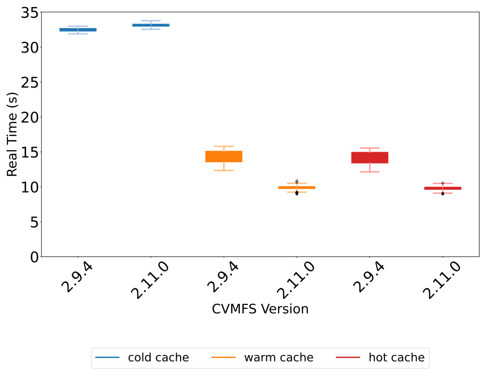
The benchmarks were run on two versions: 2.9.4 and 2.11 (Spring 2023) [12][5]. This 2.11 (Spring 2023) was a work-in-progress version which achieved the same performance as the final 2.11 release version. For each run of a command, a process of it was started on every real core, totaling up to 64 processes being run in parallel on the server running the CVMFS client.

3.1 Performance Comparison: 2.9 vs 2.11

The performance comparison was performed between version 2.9 and 2.11 shown in Figure 2. Three caching scenarios were evaluated: *cold cache* means loading from a dedicated (remote) proxy, no data is currently on the local machine; *warm cache* means the data is on the local (disk) CVMFS cache, and *hot cache* means the data is additionally also in the kernel page cache. In all caching scenarios but for Tensorflow, the new version is significantly faster. In the Tensorflow cold cache scenario, the difference is within the error margin.



(a) CMS



(b) Tensorflow

Figure 2: Real time performance of the CVMFS versions 2.9 and 2.11 for benchmarks (a) CMS and (b) Tensorflow. The performance is visualized for the different caching strategies (cold = blue, warm = orange, hot = red) using boxplots. The box represents the 25th to 75th percentile.

3.2 Performance Comparison Symlink Caching

Figure 3 shows the performance increase for 2.11 once with symlink caching being enabled and once being disabled. Symlink caching refers to keeping the resolved symlinks[6] in the

[4]The benchmarking tools and commands are becoming part of CVMFS. More information can be found on the CVMFS GitHub repository.

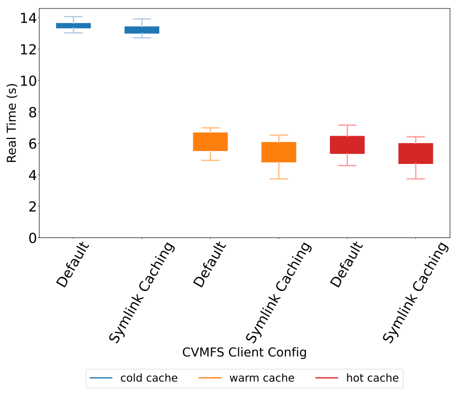
[5]This CHEP2023 branch consists of commit: f28e0e16a7 + single patch 37e876eee4

[6]Resolving a symlink means to find out to which absolute location within the file system it points to.

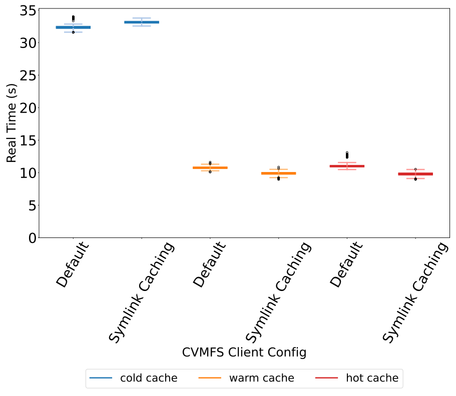
Table 1: Compression performance of zlib and zstd

Library	uncompressed	`zlib`	`zstd`
#Files	1004	1004	1004
Size (MB)	2300	999	866
Time (min)	-	1:36	0:15
Compression Ratio	-	2.30	2.66

kernel page cache. This reduces access time of the data to which the symlink points. Like in the previous performance comparison, using symlink caching increases the performance in all cases but the cold cache scenario.



(a) CMS



(b) Tensorflow

Figure 3: Performance comparison of version 2.11 (Spring 2023) with and without symlink caching for benchmarks (a) CMS and (b) Tensorflow. The performance is visualized for the different caching strategies (cold = blue, warm = orange, hot = red) using boxplots. The box represents the 25th to 75th percentile.

3.3 Pilot Study: `Zstd` Compression

As part of the download improvements (section 2.2) it has been foreseen to make `zstd` as an additional compression algorithm available. For this a first study was performed where around 2.2 GB of (uncompressed) data was downloaded by CVMFS. These chunks are once compressed with the CVMFS internal `zlib` compression tool and once with the command line tool for `zstd` using default parameters.

Table 1 lists the measurements. On average, `zstd` was 6 times faster than `zlib` and needed 15% less space.

3.4 Performance Improvements in ATLAS build system

ATLAS has all its build tools on CVMFS to build its software stack. With the introduction of the CVMFS page cache tracker feature, ATLAS reported that they noticed a significant performance increase. In Figure 4, when building ATLAS Athena with the build tools being on CVMFS, it gained 10% in performance for building with `gcc11` and even 25% for building with `clang14`.

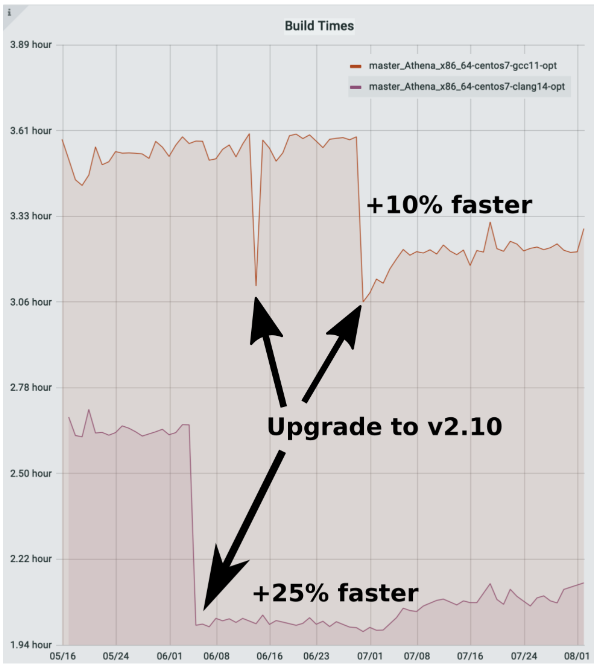


Figure 4: Speed-up for ATLAS Athena build after upgrade from CVMFS version 2.9 to 2.10. Version 2.10 has the page cache tracker enabled. The y-axis shows the build times in units of hours for the two different Athena build flavors and the x-axis show the different builds during the different days over time [13].

4 Discussion and Conclusion

The upcoming HL-LHC and its software requirements mean for CVMFS unprecedented challenges. The projected growth of the WLCG infrastructure and increase of complexity of software stacks imply the need for CVMFS to scale by an order of magnitude in several key metrics.

By now, a rich set of performance and operational improvements are well underway for CVMFS. They include optimizing performance by smarter caching strategies, increasing ease-of-use for end users and operators, and optimizing download bottlenecks. They will ensure proper quality of service at HL-LHC scales. First performance studies and feedback from experiments using newer CVMFS versions in production confirm significant performance improvements. In addition, the pilot study using `zstd` as compression algorithm shows that using `zstd` could bring significant improvements for cold cache application startup times.

References

[1] CVMFS, *cvmfs/cvmfs: The CernVM File System* (Last accessed: July 13, 2023), `https://github.com/cvmfs/cvmfs`

[2] J. Blomer, P. Buncic, R. Meusel, G. Ganis, I. Sfiligoi, D. Thain, Computing in Science & Engineering (2015)

[3] D. Weitzel, B. Bockelman, D. Dykstra, J. Blomer, R. Meusel, *Accessing Data Federations with CVMFS*, in *Journal of Physics: Conference Series* (IOP Publishing, 2017), Vol. 898, p. 062044

[4] M. Harvey, *Large-scale data processing with CVMFS at Jump Trading (6 February 2023) · Indico* (Last accessed: November 29, 2023), `https://indico.cern.ch/event/1247861/`

[5] R.N. Virtan, *CernVM-FS Profiling* (2022), `https://cds.cern.ch/record/2826081`

[6] Zstandard, *Zstandard - Real-time data compression algorithm* (Last accessed: July 13, 2023), `https://facebook.github.io/zstd/`

[7] M. Adler, G. Roelofs, *zlib Home Site* (Last accessed: July 13, 2023), `https://www.zlib.net/`

[8] Harbor Authors, *Harbor* (Last accessed: September 4, 2023), `https://goharbor.io/`

[9] D. Dykstra, personal communication

[10] R. Popescu, J. Blomer, G. Ganis, *Towards a responsive CernVM-FS architecture*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214

[11] ESnet, L.B.N. Laboratory, *iperf3 - iperf3 3.14 documentation* (Last accessed: September 4, 2023), `https://software.es.net/iperf/`

[12] CVMFS, *Release CHEP 2023 · cvmfs/cvmfs* (Last accessed: September 19, 2023), `https://github.com/cvmfs/cvmfs/releases/tag/chep23`

[13] J. Elmsheuser, personal communication