Distributing LHC application software and conditions databases using the CernVM file system

Jakob Blomer, Carlos Aguado-Sánchez, Predrag Buncic, Artem Harutyunyan
CERN, Genève 23, CH-1211 Switzerland
E-mail: jakob.blomer@cern.ch

Abstract. The CERNVM File System (CernVM-FS) is a read-only file system designed to deliver high energy physics (HEP) experiment analysis software onto virtual machines and Grid worker nodes in a fast, scalable, and reliable way. CernVM-FS decouples the underlying operating system from the experiment defined software stack. Files and file metadata are aggressively cached and downloaded on demand. By designing the file system specifically to the use case of HEP software repositories and experiment conditions data, several typically hard problems for (distributed) file systems can be solved in an elegant way. For the distribution of files, we use a standard HTTP transport, which allows exploitation of a variety of web caches, including commercial content delivery networks. We ensure data authenticity and integrity over possibly untrusted caches and connections while keeping all distributed data cacheable. On the small scale, we developed an experimental extension that allows multiple CernVM-FS instances in a computing cluster to discover each other and to share their file caches.

1. Introduction
Computing in high energy physics (HEP) takes place on globally distributed resources such as those installed in Computing Grids. While a lot of work has been done to distribute analysis jobs and experiment data onto these resources, current systems do not address the particular problems related to distribution of the experiment software. The analysis software of the LHC experiments reaches a scale that makes a straightforward distribution difficult. Approaches using “installation jobs” with shared software areas on common distributed file systems, such as NFS and AFS, are error-prone and manpower intensive and—in effect—costly. A typical software release is several Gigabytes in size and consists of hundreds of thousands of files. While running, hundreds of shared libraries are opened and tens of thousands of files and directories are examined. Given the frequent number of releases, the overall number of file system objects (directories, regular files, and symbolic links) in published LHC experiment software reaches the order of $10^7$. This is comparable to within an order of magnitude the number of entries in file catalogs of experiment data. In terms of file system requirements, this places the focus on efficient and scalable handling of file meta data rather than file data. Those characteristics apply similarly to file sets of experiment conditions data (calibrations, environmental data, etc.).

As shown in Figure 1, CERNVM-FS exploits the idea of on-access delivery of files using HTTP [1]. In the course of supporting the entire process of publishing, distributing, and running experiment software, CERNVM-FS tackles the efficient handling of file catalogs covering the entire history of software releases, transparent recovery from network failures, automatic file
de-duplication, exploitation of kernel-level file system caches, and reliable transport of data via insecure and possibly untrusted HTTP proxy servers.

CernVM-FS is developed in the context of CernVM [2] but it has shown to be useful outside CernVM on physical machines as well. By the end of 2010, CernVM-FS was transformed from a feasibility study into a production system. The four large LHC experiments use it to deliver experiment software and conditions data (ATLAS conditions data, planned for LHCb conditions data) to CernVM users, ATLAS and LHCb experiments use it additionally for Grid sites.

2. Nature of LHC Experiment Software
LHC experiment software consists of many small files, as can be seen in Figure 2. We see 50% of all regular files are smaller than 4 KB, and 80% of all regular files are smaller than 16 KB. In the first couple of months of LHC data-taking we saw weekly releases of experiment software with a monthly data growth rate of about $10^6$ files ($10^5$ in the repository) and 10-50 GB (1-10 GB in the repository) per experiment. New releases do not render previous ones obsolete as analysis tasks may be bound to specific software versions to preserve reproducibility. Between releases, as well as inside a release, there are a lot of duplicate files. Over 40 releases of ATLAS software, for instance, show some 8 million path names referencing only 1.6 million distinct files. Duplicates occur, for instance, when the same external packages are copied from one release to another. As experiments release their software by a small group of release managers, the number of clients requiring write access is very small compared to the order of $10^5$ potential read-only clients. Hence we do not require a read-write distributed file system, but instead we provide means to publish new and updated repositories and focus on the efficient read-only content delivery.

At runtime, each analysis job typically does not require all the experiment software but only a small fraction of the files of a particular release. However, those few path names are subject to a lot of system calls, in particular stat() calls (Table 1). Moreover we see lookups for non-existing files in about the same order of magnitude as successful lookups. Lookups for non-existing files occur, for instance, when the linker searches for libraries in a list of search paths. Hence CernVM-FS caches negative responses. Running analysis jobs on the worker nodes in a cluster might easily overload a shared software area [3]. This occurs even on parallel file systems, such as pNFS and Lustre, because the bottleneck is on meta-data operations.

3. CernVM-FS Repositories
A CernVM-FS repository is created on a specially prepared release manager machine. When new or updated releases are published, a single large directory tree containing the various releases
Figure 2. Size of LHC experiment software by October 2010 in terms number of file system objects and volume. The CernVM-FS repository format reaches compression factors of about 5-6 in both metrics, by use of compression and by eliminating duplicate files. Duplicate files are eliminated by the aid of cryptographic content hashes (see Section 3).

Table 1. Overall number of system calls (in thousands), distinct path names (in thousands), and cache hit rates for different jobs. Catalog Cache Hitrate refers to the meta-data memory cache in CernVM-FS. In the first data column, only system calls not covered by the kernel cache reach the catalog memory cache.

| Benchmark                      | Syscall | stat( ) | open( ) | read( ) |
|--------------------------------|---------|---------|---------|---------|
|                                |         | all uniq | all uniq | all uniq |
| Linux Kernel Compilation       |         | 438.8    | 4.2     | 426.9    | 2.4     | 426.2   | 2.4     |
| Kernel Cache Hitrate           |         | 98%      | —       | 99%      | —       |         |         |
| Catalog Cache Hitrate          |         | 56%      | 95%     | —        | —       |         |         |
| ATLAS Examples Compilation     |         | 4987.7   | 43.5    | 111.1    | 2.3     | 119.5   | 2.3     |
| Kernel Cache Hitrate           |         | 91%      | —       | 96%      | —       |         |         |
| Catalog Cache Hitrate          |         | 4%       | 94%     | —        | —       |         |         |
| LHCb Analysis                  |         | 75.6     | 11.0    | 5.8      | 1.2     | 12.3    | 1.2     |
| Kernel Cache Hitrate           |         | 81%      | —       | 41%      |         |         |         |
| Catalog Cache Hitrate          |         | 13%      | 96%     | —        |         |         |         |

of an experiment software stack (the shadow tree) is converted into so called content addressable storage, a distribution-friendly format. During this conversion regular files are compressed, renamed according to their cryptographic content hash (thus ensuring file de-duplication), and are copied into a data store. As we use a cryptographic hash, we have in particular a collision-free hash. The directory structure, symbolic links and file meta data, including the content hashes, are stored in an SQLite [4] database (the file catalog). The shadow tree is partitioned into multiple file catalogs on possibly many levels of the tree in order to keep the size of each individual file catalog manageable. The partitioning is controlled by the release managers and is naturally done on the release level.

Ultimately, the content hash of the root file catalog defines an entire repository. On the client side, such a root hash, together with the root file catalog, is used as a starting point to incrementally download and cache meta-data (file catalogs) and file data (chunks from the data store) on demand. Using SQLite catalogs makes it easy to control memory consumption simply by restricting the SQLite page cache. For minimal maintenance, files and file meta data in the
Figure 3. The trust chain that is established when CERNVM-FS downloads data.

CERNVM-FS repositories are mounted using a fully qualified repository name under /cvmfs, for instance at /cvmfs/atlas.cern.ch.

4. Transport and Security

As CERNVM-FS repositories consist of compressed static files only, we obtain the efficient and scalable distribution just by adding HTTP proxy servers. An early version of CERNVM-FS already proved to perform equally well or better than AFS and NFS for typical HEP software [2]. The CERNVM-FS content delivery network (CDN) consists of CERNVM’s central HTTP servers feeding a tree of globally distributed HTTP proxies. In addition, commercial CDN servers can be used. The CERNVM-FS repository format makes the problem of cache consistency trivial because all data chunks from the data store are immutable. As CERNVM-FS efficiently distributes experiment software to worker nodes, the very same code base is used to maintain local and remote read-only replicas of the repositories.

From the client side, only an outgoing HTTP connection is required. CERNVM-FS is still able to automatically fetch updated repository versions using an expiration date stored in the file catalogs. CERNVM-FS atomically switches to new repository versions at runtime. The client is designed to deal with network failures and supports automatic host/proxy fail-over as well as proxy load-balancing for cluster setups. In CERNVM [5] a list of close-by proxy servers is automatically selected depending on the IP address and geographical location. In effect, we gain a high-available, low-latency access to the repositories just by using already deployed infrastructures such as the CERN’s Squid servers used for proxy access from Frontier clients to Frontier servers [6].
Figure 4. Changes to the shadow tree are logged to a character device on the release manager machine. During synchronization or when the call buffer is full, writing VFS calls are blocked.

However, HTTP is an insecure protocol. In order to ensure file integrity and authenticity whilst preserving the ability to cache all data, we use cryptographically signed file catalogs. File catalogs are signed by a digital certificate. It is sufficient to sign just the file catalog; since every file is listed with a cryptographic content hash inside the catalog, we gain a secure chain and may speak of a signed repository. This allows verification of file integrity on download of files and file catalogs.

In order to validate file catalog signatures, CernVM-FS uses a white-list procedure (Figure 3). The white-list contains the fingerprints of known publisher certificates and a timestamp. A white-list has a limited lifetime and is bound to its file catalogs by the fully qualified repository name. It is signed by a private RSA key, which we refer to as a master key. The public part of the CernVM master key is distributed with the CernVM-FS sources. While using a full PKI would increase the complexity, it did not replace any of the described verification measures.

5. Repository Maintenance

While the repository format is the key to the system’s performance, creating a repository is a costly step, as all modified files and directories have to be processed. At the scale of LHC experiment software, processing an entire repository on each publish is unfeasible. Each repository update, however, touches only a small fraction; typically a single software release is added. So we require a change log of the shadow tree in order to synchronize incrementally. We explored the following mechanisms to create such a change log:

**Intercepting system calls.** An interception of system calls operates on the wrong level as changes to the shadow tree might be caused by the kernel, for instance by the NFS daemon.

**A Fuse module.** The Fuse module has to handle a lot of writing calls to small files so that the performance drops to unacceptable levels. A more thorough discussion on Fuse’s write performance has been done by Kolbeck [7].

**Kernel notifications.** Kernel monitoring facilities, such as INOTIFY [8] and SYSTEMTAP [9], do not allow to block calls. However, this is necessary in case the event buffer overflows or synchronization takes place.

To create a shadow tree change log, CernVM-FS intercepts virtual file system (VFS) calls in the kernel by providing a filter for the VFS callback framework REDIRFS [10]. It exports a shadow tree change log through a character device while also allows blocking of the writing VFS calls (Figure 4).
6. Peer to Peer Extension

For cluster deployments, an experimental extension allows the CernVM-FS instances to share their caches. Effectively this distributed memory cache replaces a cluster-wide central proxy. The peers locate each other automatically by multicast IP. Data chunks are distributed according to a distributed hash table (DHT) and stored in a dedicated memory space on the peers, managed by Memcached [11]. As the traditional consistent hashing [12] is too rigid for the potentially high number of virtualized worker nodes that arrive and depart (high peer churn), we developed a customized DHT algorithm [13].

The DHT algorithm requires the fast exchange of small peer state excerpts. While in principle this can be done by IP broadcasts, an increasing number of peers will produce an unacceptable level of network traffic. We tackle this problem by a new broadcast protocol. All the peers maintain a message queue that allows to group messages in larger network packets. The message flow is along ad-hoc trees of the peer network which results in a logarithmic number of hops required to spread new messages. The protocol regularly probes the peers for availability while also supporting a “burst mode” in order to quickly spread new messages during busy periods. The evaluation of the overall system will be subject to future research.

7. Conclusion

For the use case of large collections of small files, CernVM-FS is able to efficiently and securely host and distribute large, meta-data intense file systems. Here it outperforms common distributed file systems such as AFS on wide area networks and NFS on local networks. While approaching the issues of software delivery in a novel way, on a technical level it leverages only standard protocols. In combination with the content addressable storage format, the delivery of files is greatly simplified. In practice, the file de-duplication and the aggressive on access caches reduce the required storage for installed software on each client by at least an order of magnitude. An in-depth evaluation for conditions data will be subject to future research. As CernVM-FS enables light-weight virtual machine images for LHC applications, it provides a key building block in CernVM to harvest Cloud resources and Volunteer Computing resources.

Acknowledgments

Many thanks to John Harvey and Pere Mato for the careful review of the manuscript.

References

[1] Compostella G, Griso S P, Lucchesi D, Sfiligoi I and Thain D 2009 Computing in High Energy Physics (CHEP)
[2] Buncic P, Sanchez C A, Blomer J, Franco L, Harutyunyan A, Mato P and Yao Y 2010 Journal of Physics: Conference Series 219
[3] Lanciotti E et al. An alternative model to distribute VO specific software to WLCG sites: a prototype at PIC based on CernVM file system. In these proceedings.
[4] Hipp R et al. SQLite http://www.sqlite.org
[5] Buncic P et al. CernVM: Minimal maintenance approach to the virtualization. In these proceedings.
[6] Dykstra D and Lueking L 2010 Journal of Physics: Conference Series 219
[7] Kolbeck B Fuse performance URL http://xtreemfs.blogspot.com/2008/08/fuse-performance.html
[8] Love R 2005 Linux Journal 8
[9] Eigler F C, Prasad V, Cohen W, Nguyen H, Hunt M, Keniston J and Chen B 2005 Architecture of SystemTAP: a Linux trace/probe tool Tech. rep. http://sourceware.org/systemtap
[10] Hrbata F 2005 Callback Framework for VFS layer Master’s thesis Brno University of Technology, Faculty of Information Technology
[11] Fitzpatrick B 2004 Linux Journal 2004
[12] Karger D, Lehman E, Leighton T, Panigrahy R, Levine M and Lewin D 1997 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing pp 654 – 663
[13] Blomer J and Fuhrmann T 2010 Proc. 10th int. conf. on Computer and Communications Networks (ICCCN)