Delivering LHC Software to HPC Compute Elements with CernVM-FS

Jakob Blomer(✉), Gerardo Ganis, Nikola Hardi, and Radu Popescu

European Organization for Particle Research (CERN), Geneva, Switzerland
{jblomer,ganis,nhardi,rpopescu}@cern.ch

Abstract. In recent years, there was a growing interest in improving the utilization of supercomputers by running applications of experiments at the Large Hadron Collider (LHC) at CERN when idle cores cannot be assigned to traditional HPC jobs. At the same time, the upcoming LHC machine and detector upgrades will produce some 60 times higher data rates and challenge LHC experiments to use so far untapped compute resources. LHC experiment applications are tailored to run on *high-throughput computing* resources and they have a different anatomy than HPC applications. LHC applications comprise a core framework that allows hundreds of researchers to plug in their specific algorithms. The software stacks easily accumulate to many gigabytes for a single release. New releases are often produced on a daily basis. To facilitate the distribution of these software stacks to world-wide distributed computing resources, LHC experiments use a purpose-built, global, POSIX file system, the CernVM File System. CernVM-FS pre-processes data into content-addressed, digitally signed Merkle trees and it uses web caches and proxies for data distribution. Fuse-mounted files system clients on the compute nodes load and cache on demand only the small fraction of files needed at any given moment. In this paper, we report on problems and lessons learned in the deployment of CernVM-FS on supercomputers such as the supercomputers at NERSC in Berkeley, at LRZ in Munich, and at CSCS in Lugano. We compare CernVM-FS to a shared software area on a traditional HPC storage system and to container-based systems.

1 Introduction

Computing for High-Energy Physics (HEP) collider experiments benefits from its embarrassingly parallel workload. HEP software processes so-called "events". Events represent the data that a particle detector captured as a result of particle collisions; they can be processed independently from each other. In the case of the CERN Large Hadron Collider (LHC) this is reflected in the experiments' *high-throughput computing* (HTC) infrastructure, the World-wide LHC Computing Grid (WLCG) [1]. A federation of some 170 globally distributed data centers contributes resources in the form of commodity, Linux-based x86_64 servers. A *middleware* presents these resources as one coherent batch and data management system to hundreds of individual physics research groups. The aggregated amount of resources in WLCG, approximately half a million cores and one

J.M. Kunkel et al. (Eds.): ISC High Performance Workshops 2017, LNCS 10524, pp. 724–730, 2017.
https://doi.org/10.1007/978-3-319-67630-2_52

exabyte of storage, is comparable to a large supercomputer. Yet the computing environment comes closer to a typical Big Data installation than to an HPC system. Compute resources are considered as a set of independent CPU cores. A typical compute job runs for a few hours on a particular core. It has access to 2 GB to 4 GB memory, node-local scratch space, a GbE Internet connection to access central databases and to read and write data, and a standard Linux environment with a few custom system software packages. Substantial success was made in porting compute intensive simulation codes to special architectures found on supercomputers, including KNL and PowerPC systems [6]. Still, the use of supercomputers has been a manual, labor intensive task. A lack of bridging on the systems level between HTC and HPC worlds prevents LHC experiments to integrate supercomputers in a seamless and automated way into the general pool of resources. Custom approaches tailored to individual HPC centers are carried out in order to stage input data, write out output data, integrate with the supercomputer's job manager and to deliver codes to compute nodes. In this contribution, we will discuss the distribution of large software stacks to HPC resources.

2 Software Distribution in High-Throughput Computing

In the traditional HPC world, the distribution of codes is usually not a problem. Applications are carefully optimized, often statically linked binaries tailored to run on a specific supercomputer. They are either sent together with the compute job definition or they can reside on a shared cluster file system. In the HTC world, compute jobs can potentially be executed on any of the hundreds of the world-wide distributed data centers, whose compute nodes run different flavors of Linux with different sets of pre-installed libraries. Therefore, LHC experiments have long developed a discipline of bundling all the dependencies (compilers and system libraries) together with the application software. Overall, a single LHC software release consists of the order of ten gigabytes and hundred thousand files. New releases are produced on a daily basis.

Size and volatility of the LHC experiment software combined with the large number of compute nodes makes it difficult to use containers for software distribution. Instead, LHC experiments and other scientific collaborations use CernVM-FS, a purpose-built, global file system to provide a shared software area for compute nodes around the world. CernVM-FS pre-processes data into content-addressed, digitally signed Merkle trees and it uses web caches and proxies for data distribution. [2,3] Fuse-mounted [10,16] files system clients on the worker nodes provide access to the entire repository of precompiled software under the `/cvmfs` directory. Currently, LHC experiments provide close to half a billion (small) files in CernVM-FS.

The client loads and caches on demand only the tiny fraction of files and file meta-data needed at any given moment. This way, most data and meta-data requests are served from the compute nodes' local caches, with cache hit rates well over 99%. A typical cache hierarchy comprises some 100 MB in the worker node RAM, 10 GB on the worker node hard disk, and 50 GB on a handful of web proxies within the data center. Caching is key to CernVM-FS' ability to scale a very meta-data intensive workload—up to the MHz range of meta-data requests per node—to tens of thousands of nodes.

3 Aspects of HPC Computing Environments

For traditional HPC storage systems, such as Lustre and GPFS, the high meta-data load from LHC software is challenging. Storage of tens of millions of small files easily exceeds the user's inode quota. The synchronization of such a large number of files into the supercomputers storage system, for instance through *rsync* invocations on the login nodes, is error-prone and time consuming. At runtime, meta-data servers can easily become overloaded.

Another problem with copying software from CernVM-FS into a shared location is that its contents are often not relocatable. The supercomputer's systems team either need to create a symbolic link on the compute nodes from `/cvmfs` into the actual location or binaries and scripts need to be post-processed after copying. For one of the LHC experiments, this post-processing affected tens of thousands of files. [18]

The straight deployment of CernVM-FS on supercomputers, on the other hand, is often difficult because

1. restrictive policies for compute nodes prevent the deployment of the CernVM-FS client,
2. compute nodes might not have outgoing Internet connectivity, which is needed to populate the caches from central CernVM-FS servers,
3. compute nodes might lack local hard disks, removing a key caching layer of CernVM-FS.

The following sections discuss these obstacles. It is worth noting that binaries can be pre-compiled or cross-compiled for a variety of destination platforms and placed on CernVM-FS beforehand. In one instance, the software pre-compiled by gcc for standard x86_64 nodes even ran 20% faster compared to the same code compiled by Cray's compiler. [8]

3.1 File System Interface

Binary files containing the scientific codes have to reside on a "real" file system ready to be loaded by the operating system kernel. This is different from data, which can in principal also be accessed from applications through user-level libraries. CernVM-FS clients are based on the *Fuse* file system toolkit (cf. Fig. 1). Fuse is a kernel level file system that forwards all calls to a user-level module. Thus errors in the file system code do not cause kernel crashes. Although part of the Linux kernel, many supercomputers disable Fuse on the compute nodes.

On such systems, individual applications can access `/cvmfs` by means of the CernVM-FS connector for *Parrot* [15]. Parrot provides virtual file systems

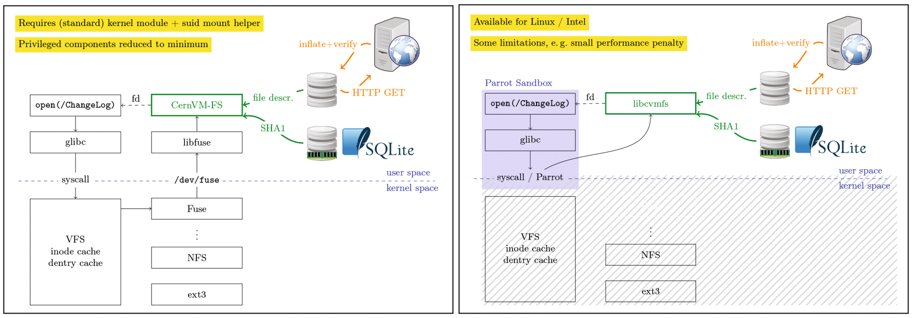
Requires (standard) kernel module + suid mount helper
Privileged components reduced to minimum
inflate+verify
HTTP GET
file descr.
SHA1
SQLite
open(/ChangeLog)
fd
CernVM-FS
glibc
libfuse
syscall
/dev/fuse
user space
kernel space
Fuse
NFS
ext3
VFS inode cache dentry cache
Available for Linux / Intel
Some limitations, e. g. small performance penalty
Parrot Sandbox
libcvmfs
syscall / Parrot

Fig. 1. CernVM-FS file system options. Left hand with Fuse upcalls to user space, right hand in pure user space with Parrot.

for Linux processes using `ptrace`-level sandboxing (cf. Fig. 1). As such, Parrot requires no special privileges but it also introduces a performance penalty. We found that the performance penalty is negligible for most compute tasks. Some HPC centers, however, reported problems with certain multi-core applications and with direct GPU access caused by the `ptrace` sandboxing. [8]

We are currently investigating Cray's Data Virtualization Service (DVS) [14] to provide network file systems to compute nodes. DVS can provide NFS volumes to compute nodes, and as such it can provide an NFS exported CernVM-FS mount point to compute nodes. In our experience, an NFS server providing `/cvmfs` scales up to a few thousand cores. Caching within DVS, however, could increase the scalability.

3.2 Local Cache Space

Much of CernVM-FS' scalability relies on the presence of node-local caches that satisfy most data and meta-data requests. When local hard disks are missing, the CernVM-FS client's cache can be placed on a cluster file system and shared by all the compute nodes. In contrast to a plain copy of the `/cvmfs`, in the CernVM-FS cache data format files are deduplicated and file meta-data is stored in larger blocks of typically a few hundred thousand files. The load from CernVM-FS clients accessing a cache on a shared file system is therefore much smaller than compute nodes directly loading software from a shared file system. At one supercomputer, the running time of codes with a shared cache on GPFS was more than three times shorter than running the software from a plain shared software area on GPFS due to inode cache thrashing in GPFS in the latter case. [17]

Even when exploiting the CernVM-FS cache format, however, millions of small files can end up on GPFS or Lustre and thousands of files can be opened concurrently by the compute nodes. To avoid the "many small files" pattern altogether, the CernVM-FS cache can be provided as a loopback device on the cluster file system. This requires one file per compute node, typically between one and ten gigabytes in size. The files are formatted with a local file system so that compute nodes are able to mount them as loopback devices. Because there is only a single file for every node, the parallelism of the cluster file system can be exploited and all the requests from CernVM-FS circumvent the cluster file system's meta-data server(s).

In our view, an efficient cache management requires flexibility in the CernVM-FS client in order to adapt to node size, network characteristics, and the storage technologies at hand. To this end, we created a plug-in interface to the client's cache subsystem so that customized cache algorithms can be independently developed and deployed. Many options are conceivable, for instance tapping burst buffers or a fully decentralized cache algorithm among the compute nodes [4]. For now, we provide a *tiered cache manager* and an *in-memory cache manager*. The two cache managers can be combined, allowing for a small hot set kept in the compute nodes' RAM and a larger warm set on a shared file system. Scale tests of these uncommon cache configurations are underway.

3.3 Internet Access

On a local cache miss, CernVM-FS clients reach out to a web server on the Internet to fetch data and populate the cache. HPC compute nodes often do not have access to the Internet but only dedicated *login nodes* have Internet connectivity.

We developed a "cache preloader" in order to pre-populate the entire content of a CernVM-FS directory tree from a login node into a location internal to the supercomputer, so that content becomes visible to the compute nodes. The cache preloader makes use of the Merkle trees to efficiently keep the data area on the shared file system synchronized. After an initial synchronization run, only change sets need to be transferred. Even for directories with hundreds of millions of files, incremental synchronization runs usually finish in a few seconds up to a few minutes. The cache preloader can furthermore prune the directory tree so that only relevant parts (for instance: the latest software versions) are copied.

4 Practical Examples

In recent years, various groups in the high-energy physics community acquired grants to run on HPC systems in the U.S. and in Europe. These included some Leadership Class Facilities such as Titan at the Oak Ridge National Lab and Mira at the Argonne National Lab. Almost all of these efforts made content from CernVM-FS available on supercomputers in one way or another. Table 1 provides an overview of code distribution approaches by different groups.

Table 1. Examples of code distribution on supercomputers used by LHC experiments.

#	HPC System	Loc	CernVM-FS Deployment
3	Piz Daint	CH	Fuse client with loopback cache [8]
4	Titan	US	Rsync of `/cvmfs` into GPFS [13]
9	Mira	US	Custom binaries [6]
20	Stampede	US	Rsync of `/cvmfs` [9]
33	HPC2	RU	Standard fuse client [11]
40	SuperMUC	DE	Parrot client with preloaded cache [17]
72	Edison	US	Shifter, parrot client with preloaded cache (tested) [7]
73	Archer	UK	Rsync of `/cvmfs` [18]
389	NEMO	DE	OpenStack virtual machines (tested) [12]

5 Related Work

A tool chain around the Shifter container system [5] was developed in order to copy the `/cvmfs` tree into a container. The content was deduplicated and compressed on a squashfs loopback device in order to reduce the size of the final container image to "only" a few hundred gigabytes. The main drawback of this approach is the time of some 24 hours it takes to produce the images. Containers in general are a promising approach to provide a commodity Linux environment on compute nodes. They can be combined with application delivery by CernVM-FS so that the container images remain small and manageable.

A utility called *uncvmfs* has been used to provide a more efficient copy of the `/cvmfs` tree. With uncvmfs, files are deduplicated by means of hardlinks. Unlike the CernVM-FS cache format, directories and symbolic links are not grouped into larger blocks, preserving many of the scalability issues of plain copies of the `/cvmfs` tree.

6 Summary

We have shown several options to approach code distribution of typical HTC applications onto supercomputers. While software stacks for LHC experiments are particularly large and volatile, we believe that typical Big Data applications will face similar challenges as HPC centers become more open for non-traditional workloads. While there are a number of successful efforts to use HPC resources in LHC computing, a generally applicable and automated approach to code distribution would be highly desirable. Beyond the scope of code distribution, using HPC resources for HTC workloads raises several other open questions, such as the integration into experiments' global data management, job management, and identity federation systems.

References

1. Bird, I., et al.: LHC computing grid: Technical design report. Technical report LCG-TDR-001, CERN (2005)
2. Blomer, J., Aguado-Sanchez, C., Buncic, P., Harutyunyan, A.: Distributing LHC application software and conditions databases using the CernVM file system. J. Phys. Conf. Ser. **331**, 042003 (2011). http://iopscience.iop.org/article/10.1088/1742-6596/331/4/042003/meta
3. Blomer, J., Buncic, P., Meusel, R., Ganis, G., Sfiligoi, I., Thain, D.: The evolution of global scale filesystems for scientific software distribution. Comput. Sci. Eng. **17**(6), 61–71 (2015)
4. Blomer, J., Fuhrmann, T.: A fully decentralized file system cache for the CernVMFS. In: Proceedings 10th International Conference on Computer and Communications Networks (ICCCN), August 2010
5. Canon, S., Jacobsen, D.: Shifter: Containers for hpc. In: Proceedings of the Cray User Group (2016)
6. Childersa, J.T., Gerhardt, L.: Developments in architectures and services for using high performance computing in energy frontier experiments. In: Proceedings 38th International Conference on High Energy Physics (ICHEP 2016) (2016)
7. Fasel, M.: Using nersc high-performance computing (hpc) systems for high-energy nuclear physics applications with alice. J. Phys. Conf. Ser. **762**, 012031 (2016). IOP Publishing
8. Filipcic, A., Haug, S., Hostettler, M., Walker, R., Weber, M.: Atlas computing on cscs hpc. J. Phys. Conf. Ser. **664**, 092011 (2015). IOP Publishing
9. Gardner, R.: Xsede integration. In: US ATLAS Physics Support, Software and Computing Technical Planning Meeting (2016)
10. Henk, C., Szeredi, M.: Filesystem in Userspace (FUSE). http://fuse.sourceforge.net, http://fuse.sourceforge.net/
11. Mashinistov, R.: Panda @ nrc ki. Talk at the PanDA Workshop (2016)
12. Meier, K., Fleig, G., Hauth, T., Janczyk, M., Quast, G., von Suchodoletz, D., Wiebelt, B.: Dynamic provisioning of a hep computing infrastructure on a shared hybrid hpc system. J. Phys. Conf. Ser. **762**, 012012 (2016). IOP Publishing
13. Nilsson, P., Panitkin, S., Oleynik, D., Maeno, T., De, K., Wu, W., Filipcic, A., Wenaus, T., Klimentov, A.: Extending atlas computing to commercial clouds and supercomputers. PoS, p. 034 (2014)
14. Sugiyama, S., Wallace, D.: Cray dvs: Data virtualization service. In: Cray User Group Annual Technical Conference (2008)
15. Thain, D., Livny, M.: Parrot: an application environment for data-intensive computing. Scalable Comput. Pract. Experience **6**(3), 9–18 (2005)
16. Vangoor, B.K.R., Tarasov, V., Zadok, E.: To FUSE or Not to FUSE: performance of user-space file systems. In: Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST 2017) (2017)
17. Walker, R.: Hep software on supercomputers. In: Talk at the CernVM Users Workshop (2016)
18. Washbrook, A.: Processing lhc workloads on archer. In: Talk at GridPP35 Conference (2015)