

The Evolution of Global Scale Filesystems for Scientific Software Distribution

Jakob Blomer, Predrag Buncic, René Meusel, and Gerardo Ganis | CERN

Igor Sfiligoi | University of California, San Diego

Douglas Thain | University of Notre Dame

Delivering complex software across a worldwide distributed system is a major challenge in high-throughput scientific computing. To address this problem in high-energy physics, a global scale filesystem delivers software to hundreds of thousands of machines around the world.

Delivering software across a worldwide distributed system is a major challenge in high-throughput scientific computing. While we often think of an application as a single executable that can be moved between machines, the reality is often much more complex: production applications are a complex assembly of scripts, configuration files, libraries, and multiple executables written in distinct languages with mutual dependencies.

This problem is particularly acute in the high-energy physics (HEP) community, which relies on a large and dynamic set of software developed over multiple decades by many authors in many different languages and environments. In any scientific effort, reproducibility is essential, and so the HEP community has developed a discipline of carefully collecting all the software that composes a given application—compilers, libraries, interpreters, and so on—into one place so that consistency is achieved across the community. A member of a particular experiment simply downloads the standard software package in a certain version and proceeds with work. However, copying the entire software stack everywhere it's needed isn't practical—it can be very large, new versions of the software stack are produced on a regular basis, and any given job only needs a small fraction of the total software. In this way, scientific software differs from general-purpose applications (such as Apache or Emacs) with

stable releases and small footprints that can be easily packaged.

Within one computing center, deploying scientific software isn't usually a significant problem: the software distribution can be made available on a distributed filesystem mounted in a known location on all the system's nodes. However, to address scientific problems at the largest scale, it becomes necessary to harness machines at locations all around the world. Distributed computing environments such as Condor¹ and Globus² have long enabled users to send jobs to other sites for execution, but users of those systems quickly discovered that their jobs couldn't function without the necessary software installed, nor did they have privileges to install the software.

To address this problem, a distributed filesystem designed specifically for distributing complex software stacks at a global scale is required. The solution must be efficient at distributing large amounts of content around the world, while maintaining the structure and semantics of a Portable Operating System Interface (POSIX)-compatible filesystem, directory tree, and metadata. It must also be entirely user level, enabling it to be accessed from any machine and in any environment without requiring special privileges. The key observation is that the software is only modified at the point of publication—all other consumers are read-only with high availability. Consistency and integrity are

Related Work in Global Data Distribution

The work described in this article combines concepts found in distributed filesystems, Web content distribution, and distributed version control.

Many distributed filesystems have aimed for global data sharing, with the Andrew Filesystem (AFS)¹ coming the closest to realizing this vision as a production system. As widely used as AFS once was, the technical matter of deploying kernel modifications limited its use within the context of distributed systems. Moreover, the possibility to read and write from any client requires servers to maintain state information per client so that each server can typically serve no more than a few hundred clients. Performance comparisons between AFS and CernVM-FS for the use case of loading software show an order of magnitude less network traffic induced by CernVM-FS and thus greatly improved startup delays, despite the HTTP protocol overhead.²

Some research prototypes with global ambitions (such as OceanStore³ and Tahoe⁴) build on the idea of peer-to-peer overlay networks to distribute files on a global scale. These systems are designed for secure and reliable data sharing among nodes that both consume and produce files. But compared to a hierarchy of Web caches, geographically distributed peer-to-peer systems are difficult to set up across firewalls and institutional boundaries; they're also technically difficult to tune for high speed and responsiveness for a workload involving many small files. As designed, these systems don't deliver POSIX filesystem semantics, and for a variety of reasons, never saw deployment beyond research prototypes.

Several filesystem designs such as Ceph⁵ and Panache⁶ have addressed metadata performance by distributing metadata across multiple servers, enabling parallel access to it. Although this increases overall system throughput, it doesn't improve a single application's metadata performance (it must request items one by one). In contrast, the approach shown in this article enables bulk delivery of metadata, which both accelerates individual applications as well as increases throughput.

In many ways, HTTP supplanted filesystems as a means of distributing data in the wide area, but it lacks any standardized form of metadata access, making it unsuitable for direct mounting from clients. The proposed WebDAV⁷ standard adds metadata access methods to HTTP for the purposes of collaborative authoring, but WebDAV isn't designed for metadatintensive workloads, and important capabilities for software distribution such as data verification, bulk metadata handling, or consistent namespace updates aren't part of the protocol. GROW-FS and CernVM-FS work around this by natively delivering metadata as HTTP objects.

The HTTP-Fuse⁸ filesystem delivered Linux operating system distributions (similar to the way the CernVM virtual

appliance uses CernVM-FS) by using content-addressable storage and data transport through the Coral content distribution network.⁹ Content delivery networks (CDNs) such as Akamai and CloudFront are now widely used for distributing HTTP objects globally. Although CernVM-FS can use these technologies, the HEP-specific proxy cache network allows for the careful segregation, monitoring, and control of heavy network loads that might otherwise conflict with commodity network access at the institutions involved.

The Merkle tree is a widely used technique for the efficient incremental computation and verification of checksums. CernVM-FS is similar to a distributed version control system such as Git, in that both use Merkle trees to name a given filesystem tree. In fact, you can use Git as a simple content distribution system for offline access. However, system objectives are very different: Git assumes that repositories are relatively small and can be duplicated in their entirety, whereas CernVM-FS is designed to handle large repositories whose contents must be distributed on demand to online filesystem operations.

References

1. J.H. Howard et al., "Scale and Performance in a Distributed File System," *ACM Trans. Computer Systems*, vol. 6, no. 1, 1988, pp. 51–81.
2. J. Blomer, "Decentralized Data Storage and Processing in the Context of the LHC Experiments at CERN," PhD thesis, Dept. Computer Science, Technische Universität München, 2012.
3. J. Kubiatowicz et al., "Oceanstore: An Architecture for Global-Scale Persistent Storage," *ACM Sigplan Notices*, vol. 35, no. 11, 2000, pp. 190–201.
4. Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The Least-Authority Filesystem," *Proc. 4th ACM Int'l Workshop Storage Security and Survability*, 2008, pp. 21–26.
5. S.A. Weil et al., "Ceph: A Scalable, High-Performance Distributed File System," *Proc. 7th Symp. Operating Systems Design and Implementation*, 2006, pp. 307–320.
6. M. Eshel et al., "Panache: A Parallel File System Cache for Global File Access," *Proc. Usenix Conf. File and Storage Technologies*, 2010, pp. 155–168; www.cse.buffalo.edu/faculty/tkosar/cse710_spring13/papers/panache.pdf.
7. Y. Goland et al., "HTTP Extensions for Distributed Authoring—WEBDAV," IETF RFC 2518, 1999; www.ics.uci.edu/~ejw/authoring/protocol/rfc2518.html.
8. K. Suzuki et al., "HTTP-FUSE Xenoppix," *Proc. 2006 Linux Symp.*, vol. 2, 2006, pp. 379–392.
9. M.J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing Content Publication with Coral," *Proc. Usenix Symp. Network Systems Design and Implementation (NSDI)*, vol. 4, 2004, p. 8.

achieved via Merkle trees to checksum all content up to the filesystem's root. This enables read access even during network outages.

Traditional filesystem technologies simply can't meet these needs because they were never designed for unprivileged deployment or operation at large scale across wide area networks (see the "Related Work in Global Data Distribution" sidebar). Web content delivery systems are closer to the mark, but they don't provide the filesystem tree structures or efficient metadata delivery required by these applications. This article describes the evolution of the global filesystem concept in the HEP computing community. An early prototype (called GROW) validated the basic idea but revealed limitations in design and implementation. CERN started a new implementation (CernVM-FS; <http://cernvm.cern.ch/portal/filesystem>) with improved performance and scalability; it now serves the needs of many communities in HEP, astrophysics, and life sciences, running on hundreds of thousands of machines around the world on a daily basis.

Design Objectives

A global-scale filesystem for scientific software distribution must meet the following needs:

- *World-wide scalability.* HEP applications must run at enormous scale to enable timely processing of the data produced by the Large Hadron Collider (LHC). The filesystem should be able to scale to hundreds of thousands of machines spread across potentially thousands of sites around the world. A multilayer hierarchy is the only effective means of achieving global scale.
- *Unprivileged deployment.* Although a given user might be able to obtain administrator privileges on a few machines, or perhaps even an entire site, no one can possibly have such access at a global scale. Thus, the system must be usable by an ordinary user on the client side without requiring software installation, kernel modules, deployment of virtual machines, or similar techniques.
- *Infrastructure compatibility.* Large computing centers often operate in a restricted networking environment, with good reason. Firewalls and other network translating devices often make it difficult or impossible to employ nonstandard ports or protocols, only allowing traffic via well-known channels such as HTTP or Secure Shell (SSH). Thus, our design relies heavily on the established Web caching infrastructure.
- *Application-level consistency.* Many previous filesystem designs have struggled with the tradeoffs described by the CAP (consistency, availability, partition tolerance) theorem³ because they assume that a client's objective is always to read the most recent version of data available. In this case, software must remain consistent while the application runs, so a single snapshot of the filesystem must be delivered to the application throughout the run, greatly improving performance, availability, and reproducibility.
- *Efficient metadata access.* HEP applications often have a period of intense metadata: the application startup involves running scripts, searching multiple directories, loading configuration files, accessing libraries, and so forth. A large number of metadata lookups result in failure because the scripts are searching for entries that don't exist. A suitable solution must be efficient at supporting millions of metadata operations on a cold cache in a short period of time.

A first prototype of such a filesystem was conceived in the context of the (now concluded) Collider Detector at Fermilab (CDF) experiment.

Case Study: The CDF Experiment

CDF was typical of HEP experiments in that a computational model of the physical detector was created and then subjected to simulated particles to understand how the real device would respond to real particles. These simulations were run an enormous number of times in different configurations to fully understand the detector's behavior. Access to thousands of machines was necessary to generate results in a timely manner.

The CDF standard software distribution was composed of several hundred different software release versions, each composed of thousands of small files, plus a shared area. The files themselves in the software releases never changed; versioning was used extensively to allow long-term scientific reproducibility. Files in the shared area were allowed to change, but this was heavily discouraged. A typical user job would access a fraction of files in the shared area (such as startup scripts) plus a subset of files in one of the software release directories. Although a given user was likely to only use 1 percent of the files in the repository, the precise set would differ between users and wasn't easily determined in advance.

Until the early 2000s, CDF relied on using monolithic local clusters, in which the software was mounted on a single network filesystem (NFS)

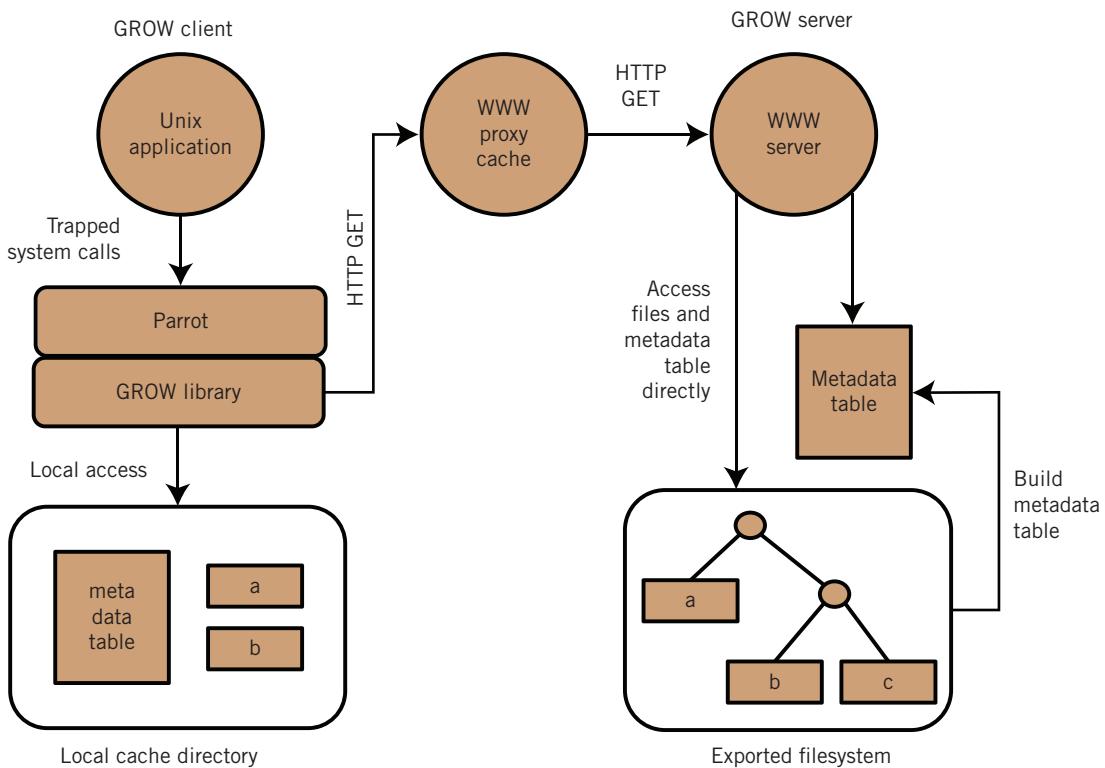


Figure 1. GROW-FS architecture. On the server side, GROW-FS is simply an ordinary webserver that exports the desired software. Using the HTTP protocol makes it easy to transit multiple networks because it's rarely blocked by firewalls.

server and accessed over a LAN. Researchers desired to access machines on the global computing grid, but NFS didn't have the performance or security to be accessed over the WAN. AFS was more suited to global access, but it proved administratively impossible to convince every site to install it, due to the challenges of deploying kernel modules and opening up firewalls.

This experience convinced the CDF computing team that it needed a solution that didn't require system privileges to work. The Parrot user-level filesystem (<http://ccl.cse.nd.edu/software/parrot>)⁴ looked like a suitable technology for mounting a filesystem, but it didn't (yet) support a protocol for caching software accessed over the WAN. The caching requirement led to consideration of the HTTP protocol because it had strong and mature support of distributed caching. The relatively small size of most CDF files also nicely fit the typical HTTP use case. The problem, however, was how to present these files in a filesystem-like manner.

The Prototype Filesystem: GROW-FS

To meet CDF's needs, a prototype filesystem called GROW-FS (<http://ccl.cse.nd.edu/software/parrot>)

was created. Figure 1 shows the system architecture. On the server side, GROW-FS is simply an ordinary webserver that exports the desired software. Using the HTTP protocol makes it easy to transit multiple networks because it's rarely blocked by firewalls. However, HTTP has one significant drawback: it doesn't offer access to filesystem metadata in any standard way. Here, metadata includes directory listings and file details such as size, owner, and permissions.

To enable efficient metadata access, GROW-FS requires the server operator to run a script that traverses the filesystem and produces a metadata table listing all files along with their metadata and (optionally) a checksum of file contents. The filesystem tree as a whole is a Merkle tree, in which each directory's checksum is computed from the checksum of the files it contains. As a result, the directory's root has a checksum that represents the entire tree's contents.

On the client side, the GROW-FS library implements a read-only filesystem by combining the metadata table and file objects. When the filesystem is first accessed, the library downloads the *entire* metadata table and holds it in memory. This allows all directory reads and metadata lookups to be

satisfied from client memory. File accesses are implemented by downloading the desired file from the server, caching it on local disk by name, and then accessing the file directly.

Finally, Parrot attaches the application to the GROW-FS library. Parrot is a user-level tool for attaching standard Unix applications to a variety of remote data services such as HTTP, FTP, iRODS, and HDFS. It works by capturing all of an application's system calls via the `ptrace` debugging interface. System calls that refer to files in the GROW-FS namespace, such as `/grow/www.cern.ch/file`, are converted into calls in the GROW-FS library, thus transparently attaching the application to the remote service.

For several years, GROW-FS enabled CDF simulations to run on several thousand CPUs simultaneously across the LHC Computing Grid (LCG), including IN2P3 in France, GridKA in Germany, and Fermilab in the US.⁵ The prototype was efficient at running CDF codes and demonstrated that a user-level filesystem delivered over HTTP could be highly effective. In particular,

- GROW-FS did an excellent job of localizing I/O, particularly metadata access. Once a local cache was warmed up, similar applications could run repeatedly with little or no contact with the server. Metadata access was handled entirely within client memory, which resulted in the application's very fast initial configuration. Only the data necessary for the given application was moved over the network.
- Parrot's overhead for accessing GROW-FS was acceptable for simulation codes. At a microlevel, Parrot slowed down individual system calls, sometimes by a factor of 10 compared to accessing a local filesystem. However, when considering I/O costs against simulation runtime measured in hours, the overall slowdown was less than 5 percent compared to a local run.

But there were also significant limitations:

- Constructing the metadata table at the server became an increasing cost as software sizes grew. It took approximately 30 minutes to build the 20 Mbytes of metadata for CDF software distribution. This was acceptable for occasional updates but became a burden as software updates increased in frequency.
- Cold client startup could be quite expensive. At the first reference to a filesystem, GROW-FS

would require downloading the entire metadata table and unpacking it in memory. Again, this was acceptable for CDF but for a much larger filesystem, it resulted in traffic bursts as multiple clients began simultaneously.

- Proxy cache discovery was also much harder than expected. Many HEP sites deployed proxy caches near clusters to support a variety of applications, but there was no standard way for a running job to discover the nearest cache's location. If users couldn't determine this, the filesystem would fall back to accessing the central repository directly, which didn't scale.
- Although GROW-FS could detect (and prevent) inconsistencies, it didn't support multiple software versions simultaneously. The server could only be updated between runs of the analysis jobs.

Overall, GROW-FS demonstrated that software distribution via a global filesystem was feasible, but more work was needed to reach the next level of scalability.

The Production System: CernVM-FS

CernVM-FS derives its name from being part of CernVM, an R&D project established at CERN in 2008 to investigate how virtualization technology could improve the daily interaction of physicists with experiment software frameworks.⁶ When the project started, the way in which computing resources are provided to scientific experiments was changing. Along with the resources from managed computer centers within the grid, a substantial amount of future resources would be unmanaged and opportunistic. The cloud emerged as an interface to access resources, with commercial and academic infrastructure-as-a-service (IaaS) clouds, volunteers' computers, or special-purpose computers used in data taking that remain unused when the detector undergoes maintenance and upgrades. Such resources required a virtual machine image. Due to the size and fast rate of changes of the LHC application stack, baking the application software into the virtual machine image wasn't feasible; the only possibility was to deliver the application software on demand via a network file system.

The scale of software distribution for LHC experiments raised by one to two orders of magnitude in comparison to former HEP experiments. The ATLAS experiment, for instance, produces new versions of the experiment software almost on a daily basis. Any particular software version comprises

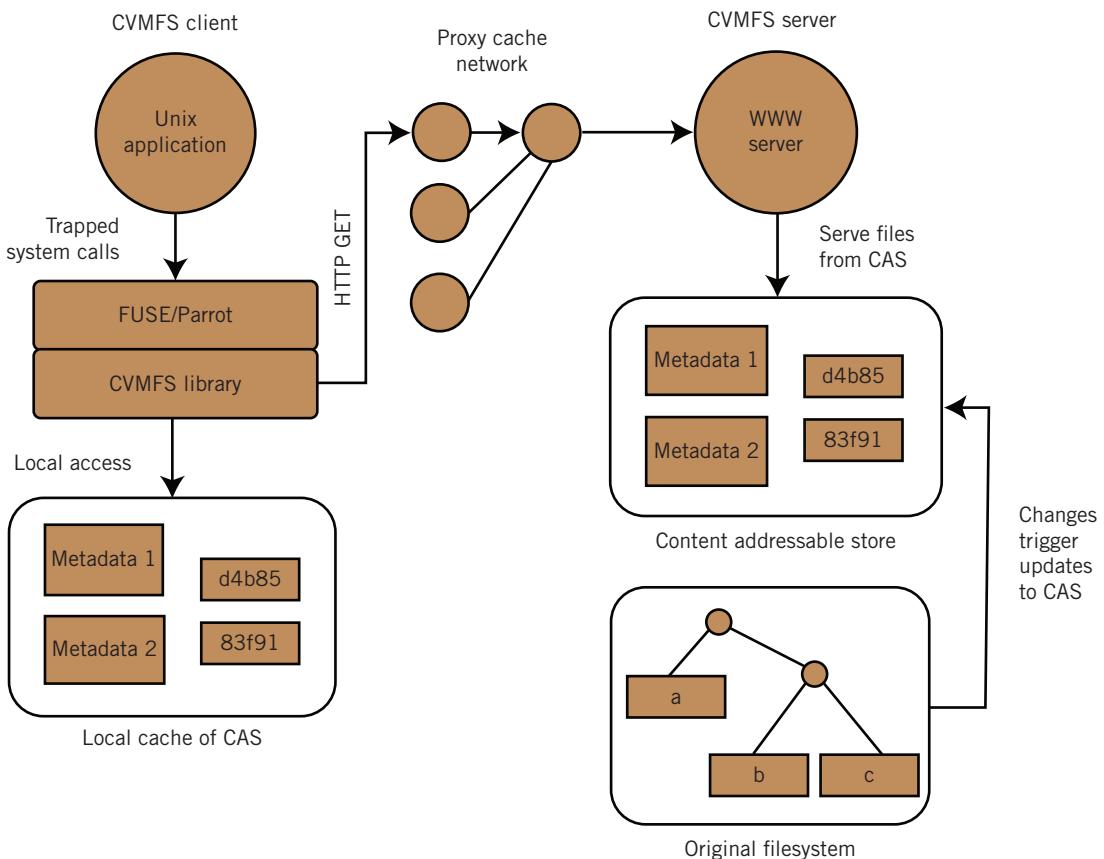


Figure 2. CernVM-FS architecture. The problem of separating the application software from a virtual machine to create a lighter virtual machine that's easier to maintain slightly differs from the software distribution problem GROW-FS addressed.

some 200,000 files and directories that sum up to 10 Gbytes, comparable to the size of an office suite or an operating system. And while new versions keep being added, older versions need to remain available for the sake of reproducibility.⁷

Figure 2 shows CernVM-FS's architecture. The problem of separating the application software from a virtual machine to create a lighter virtual machine that's easier to maintain slightly differed from the software distribution problem GROW-FS addressed. Being in full control of the environment let us move from a pure user-level interface to a FUSE (<http://fuse.sourceforge.net>) filesystem, which provides better performance due to its use of kernel-level caches. One of the new challenges, however, was disconnected operation for virtual machines that run on physicists' or volunteers' laptops.

These differences turned out to be not fundamental. A single infrastructure could serve software distribution needs for grids as well as clouds, while slight variations to the filesystem client code

offered different interfaces and modes of local caches tailored to the environment at hand. For example, Parrot could access CernVM-FS from restricted and opportunistic resources, and FUSE could be used inside virtual machines and cooperating grid sites, if necessary through an NFS interface. Another mode of operation lets CernVM-FS act as a Linux root filesystem, thereby loading both application software and the operating system binaries on demand from the network.⁸

When building the CernVM-FS content delivery network, we extended GROW-FS's scalable and robust architecture via fault-tolerant content distribution. Because HTTP connections are stateless, high availability is provided by the filesystem client in the form of server failover, provided that Web proxies and webservers are duplicated. A handful of standard Apache webservers with a full read-only data replica, the Stratum 1 servers, provide access to filesystem content at different geographical locations around the world. The central webserver that

provides access to the read/write copy of the data, the Stratum 0 server, is only used to feed Stratum 1 servers and is thus removed from the critical path. Replication from Stratum 0 to Stratum 1 exploits Merkle trees. Unlike `rsync`, which has to inspect every file and every directory, Merkle trees provide an efficient means to instantly detect where exactly in the filesystem the tree changes are made. That lets us synchronize replicas of frequently changing filesystems with a hundred million objects within less than an hour over standard, wide-area HTTP connections.

Automatic configuration of HTTP proxies is critical to both performance and usability, so CernVM-FS clients use a combination of mechanisms to find nearby servers. To select a Stratum 1 server over the WAN, geolocation Web services and direct measurement of round-trip times are performed. To select a nearby proxy server on the LAN, clients use the WPAD⁹ proxy discovery protocol or access a central registry of Web proxies.¹⁰ Other mechanisms of server discovery are conceivable, including bootstrapping server addresses from CernVM-FS itself, which turned out to be much more a matter of administrative agreement than a technical obstacle.

A fundamental change in CernVM-FS was the introduction of content addressable storage (CAS).¹¹ Instead of deriving a file's URL from the path name, the file's URL is derived from a cryptographic hash of its contents, which is stored in the metadata table. The cryptographic hash provides a short yet globally unique identifier for a file. Content-addressed files are self-verifying in the sense that their integrity can be verified without inspecting the metadata table. They're immutable, so maintaining cache consistency is trivial because data never expires. Changes to a particular path name result in a new content-addressable file that provides a building block for versioning in the filesystem: it's sufficient to remember the root hash of the Merkle tree at various points in time to go back to corresponding filesystem snapshots. The same file under different paths is deduplicated and mapped to the same content-addressable file. Indeed, the deduplicated data doesn't grow at the speed of the sheer path names: as new software versions are installed, we see only 10 to 20 percent of the files with new content. Combined with data compression, the amount of transferred data is thus reduced to a level where it becomes feasible to use even complex software stacks over WANs with consumer-grade Internet access. Yet, large multigigabyte files occasionally end up in the filesystem together with software (such as ISO images, tar archives, and SQLite indexes),

which can jam the caching infrastructure. CernVM-FS cuts these files in smaller chunks of not more than a few megabytes, using rolling checksums^{12,13} rather than fixed-sized chunks. Like a `diff` on text files, rolling checksums follow changes in binary files so that small changes to a file result in only one or two new chunks.

In a similar effort to avoid very large files, the metadata table can be split along the filesystem tree, but automatic splitting on every directory level was dismissed as a step backward in exploiting metadata locality. Instead, the splitting of the metadata is user-guided through creation and deletion of magic hidden files as markers for splitting and merging the metadata table. Filesystem maintainers can easily identify filesystem subtrees that have a high locality such as root directories of particular software package versions. Good experience was made with metadata partitions that have at least 1,000 entries but not more than a few hundred thousand entries. Each of these is known as a *subcatalog*.

A surprisingly large amount of work went into the efficient update management at the filesystem's central writable copy. Any given filesystem update writes or modifies a potentially large number of files (such as a new software version), which is still only a small subset of all available files. Several unsatisfying approaches were implemented to identify the changeset of filesystem modifications to publish a new snapshot. Kernel-level filesystem tracers turn out to be either difficult to maintain or unreliable for large batches of small writes.¹⁴ Interposition systems, such as a recording FUSE filesystem, have a high-performance hit for batch writings of small files, although this is constantly improving.

In the current version, CernVM-FS uses a kernel-level union file system which has a performance overhead of only a few percent. The filesystem's current read-only state is provided by CernVM-FS itself, whereas the union filesystem redirects all modifications into temporary local storage from where they can be further processed (compressed, checksummed) in a parallelized, batched fashion. As scalable back-end storage beyond the local filesystem, CernVM-FS can upload data to S3-compatible storage, which in turn might directly serve as an HTTP Stratum 0/1.

Performance

To demonstrate the effectiveness of caching on the client side, we instrumented and configured an instance of the CernVM-FS client to run with the production software and archive. We ran a standard simulation

Table 1. Cache effectiveness.

Software	Cache	Time	FUSE syscalls			CernVM-FS client ops		
			Stats ($\times 1,000$)	Opens ($\times 1,000$)	Reads (Mbytes)	HTTP requests	Downloaded data (Mbytes)	Downloaded metadata (Mbytes)
CMSSW	Cold cache	12m, 05s	2,429	11	840	4,536	895	147
	Warm cache	8m, 14s	2,429	11	772	1	0	0
Firefox	Cold cache	16s	17	1	186	268	71	1.5
	Warm cache	2s	17	1	186	1	0	0
LaTex	Cold cache	23s	150	2	85	351	19	12
	Warm cache	17s	150	2	85	1	0	0

Table 2. Caching across versions.

Software version	FUSE syscalls			CernVM-FS client ops		
	Stats ($\times 1,000$)	Opens ($\times 1,000$)	Reads (Mbytes)	HTTP requests	Downloaded data (Mbytes)	Downloaded metadata (Mbytes)
CMSSW 6_2_5	2,477	12	859	4,906	854	147
CMSSW 6_2_6	2,461	12	827	Δ 1,751	Δ 78	Δ 6
CMSSW 6_2_7	2,461	12	823	Δ 1,874	Δ 94	Δ 6
CMSSW 6_2_8	2,470	12	825	Δ 1,761	Δ 41	Δ 6
CMSSW 5_3_1	2,332	11	741	Δ 2,688	Δ 284	Δ 6
CMSSW 5_3_12	2,340	11	711	Δ 1,828	Δ 81	Δ 6

and reconstruction code using the standard software of the CMS experiment (CMSSW) release 6.2.5 twice, once with a cold cache and then the identical code a second time to observe the warm cache’s effect. We did the same exercise to compare the Firefox browser and LaTex binaries hosted on CernVM-FS.

Table 1 shows the instrumented applications’ behavior. The CMS application first configures itself by using CernVM-FS and then processes 10 events; the Firefox browser is just being opened, and the LaTex binaries compile a 70-page technical report. The first column shows the total time for execution, with the next three showing the total number of `stat()` and `open()` system calls—and the amount of data read—at the FUSE layer. (All system calls are supported, but these are the most numerous.) The next three columns show the behavior of the CernVM-FS client itself, measured in the number of HTTP requests, data downloaded, and metadata downloaded

over the network. The applications ran once with an empty cache to observe cold cache behavior and then once again to observe warm cache behavior.

Several key observations can be made from this table. First, loading large applications is extremely metadata intensive—for LHC applications, it involves issuing millions of `stat()` system calls. But it isn’t enough for the cache to have a high hit rate: it must also be efficient at filling the cold cache. Millions of `stat()` operations are reduced into a much smaller number of HTTP queries, each of which fetches a large amount of directory metadata a single time. Second, once the cache is warmed, the applications are considerably accelerated, and remote accesses are reduced to a single HTTP query that verifies the filesystem’s root checksum. (The difference between data read in each configuration is due to the action of the OS buffer cache, which is consulted before reaching the FUSE module.)

Table 3. Top 10 repositories by size.*

Repository	Files (M)	Directories (K)	Symlinks (K)	Subcatalogs	Data (Gbytes)
atlas.cern.ch	35.1	5,837	7,810	518	2,223
cms.cern.ch	31.4	3,570	1,757	789	967
lhcb.cern.ch	13.5	2,218	118	1,584	542
alice.cern.ch	6.5	533	15	487	571
sft.cern.ch	4.9	485	104	727	412
ams.cern.ch	3.0	130	618	69	1,997
geant4.cern.ch	2.6	122	2	126	115
belle.cern.ch	1.0	282	7	50	105
boss.cern.ch	0.96	106	22	9	42

* M and K stand for millions and thousands, respectively.

Table 2 shows the effect of running multiple versions of the CMS software sequentially at the same client. Each line indicates another run of the same simulation and reconstruction job, each with a different release of the software. There is considerable overlap between versions, and CernVM-FS effectively reduces the total amount of data that must be distributed whenever the software is updated.

Current Scale and Deployment

CernVM-FS is widely used today within the HEP community. Most major experimental operations maintain a CernVM-FS repository that's made available to grid sites, virtual machines in the cloud, and sometimes to supercomputer resources around the world. Although the HEP community—in particular, those running the LHC experiments—has the largest CernVM-FS repositories, scientific collaborations from other fields have recently started to operate CernVM-FS production services, too. Scientific software is distributed, for instance, for astrophysics researchers at the Pierre Auger Observatory and for researchers in bioscience and life sciences within the BIOMED, WeNMR, and GenAP projects. Somewhat in the slipstream of the infrastructure for scientific software stacks, there are currently some 10,000 CernVM machines per month that do a network boot of the operating system from CernVM-FS.

To give a sense of the scale of deployment for LHC experiments, Table 3 shows the top 10 repositories at CERN in December 2014 by size. Note that while the total amount of data in each repository is relatively modest (typically a few hundred

gigabytes) the number of metadata entries (millions) gives some sense of the complexity of these software installations.

The number of end clients can't be measured directly because they're all hidden behind Web proxy caches that, by design, hide large amounts of data from the central repositories. However, some sense of the scale can be obtained by looking at the central servers' activity logs. The Stratum 1 proxy server's cache log that covers all repositories at CERN was extracted for a one-week period in July, during which the server responded to 64 million requests for a total of 10.2 Tbytes of data over the course of a week, which is a modest 382 Kbytes/s sustained throughput.

To preserve system scalability, individual clients aren't permitted to connect to the Stratum 1 server—only proxy caches representing an entire cluster are admitted. During the instrumented week, proxy servers from 70 sites made requests from the Stratum 1 server, each representing a remote cluster of clients. Each of these clusters can range in size from a handful of machines in a closet to more than 3,800 machines in the CERN central datacenter. Given the resource pledges of grid sites, our best estimate is that those 70 sites host approximately 28,000 clients, with CernVM-FS currently running in production on some 64,000 nodes at 160 sites.

Future Opportunities

CernVM-FS has been successfully adopted by scientific organizations with centrally coordinated software stacks and a default attitude of openness toward software and data. To expand the system's scope will require tackling several fundamental challenges.

Organizational Complexity and Troubleshooting

The current design accommodates for the fact that scientific collaborations often span organizational boundaries such as multiple universities, research institutes, and industrial partners in several countries. Similar to the Web itself, anyone can create a globally accessible filesystem, anyone can mirror existing filesystems, and any filesystem client or Web proxy can connect to existing filesystem mirrors. To accommodate the independent evolution of each component, there must be minimal dependencies between the components and loose coupling between clients and servers and between independent servers.

The downside of this decoupling is that monitoring and troubleshooting the network of clients, proxies, and servers is challenging but essential to achieving good performance. Individual organizations perform monitoring within their boundaries, but no one is charged with the performance of a system as a whole. Such issues are addressed by content delivery networks (CDNs) that have carefully managed networks of edge servers.^{15,16} However, the underlying assumption of CDNs is that a large number of geographically distributed servers are interconnected and publicly reachable, which is difficult to achieve across organizations that employ firewalls liberally. More work is needed to understand how to deploy, maintain, and troubleshoot complex storage networks in this dynamic environment.

Cooperative Storage and Data Distribution

For reasons of simplicity, cooperative caching in the local network isn't supported. Provided that a few gigabytes of local hard disk space are available for a cache (an assumption that isn't always true), independent local hard disk caches are easy to maintain, and software or hardware failures on one node can't affect another node. The working sets of the nodes within a cluster, however, tend to be almost identical and could be reduced considerably by peer-to-peer data access. So the simplicity in cache management comes at the cost of a large number of duplicated caches and network transfers.

This is an interesting problem to tackle in light of a growing interest to not only distribute software but also scientific datasets using the same infrastructure and filesystem clients. The high level of POSIX compliance of the filesystem client is clearly appealing. At the same time, larger datasets typically require high data throughput and tight constraints on the effective replication factor. Neither of these

were original design goals. Slight adjustments in the implementation can possibly stretch the current limitations in a way not yet known.

Confidentiality

For datasets even more so than for software, data confidentiality can be an issue. To confidentially distribute data over wide-area links, the transfer must be encrypted for a closed user group. Two important properties for any such end-to-end encryption are the timely support for group membership changes (in particular, timely revocation of user access) and maintaining the ability to easily cache files on intermediate proxies (in encrypted form). Despite early ideas on an extension to the data distribution scheme and key management,¹⁷ much work remains to be done to support confidential data in the current system.

Access to scientific software often plays a key role in the ability to reproduce scientific results. Conventional software distribution is based on the assumption that, once installed, applications are used many times and that users want to receive updates to the newest version as soon as possible. In contrast, users of scientific software usually want access to a very specific version to ensure that the same input yields the same results. Most scientific software is only a vehicle for processing a specific dataset whereupon the software serves no purpose anymore and can be purged from a computing node. Instead of package managers and (lightweight) virtual machine images, the approach presented in this article uses a Web-based, global, and versioning filesystem. This approach combines the ease of administration of software as a service, the scalability of local installations, and the traceability of a version-control system. ■

Acknowledgments

This work was supported in part by the US National Science Foundation grant CNS-0643229. Blomer was supported by a Marie-Curie fellowship of the European Union.

References

1. D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, nos. 2–4, 2005, pp. 323–356.
2. I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Network and Parallel Computing*, Springer, 2005, pp. 2–13.

3. E. Brewer, "CAP Twelve Years Later: How the 'Rules' Have Changed," *Computer*, vol. 45, no. 2, 2012, pp. 23–29.
4. D. Thain and M. Livny, "Parrot: An Application Environment for Data Intensive Computing," *Scalable Computing: Practice and Experience*, vol. 6, no. 3, 2005, pp. 9–18.
5. G. Compostella et al., "CDF Software Distribution on the Grid Using Parrot," *J. Physics: Conf. Series*, vol. 219, no. 6, 2010, article no. 062009.
6. P. Buncic et al., "A Practical Approach to Virtualization in HEP," *European Physical J. Plus*, vol. 126, no. 1, 2011, pp. 1–8.
7. J. Blomer, "Decentralized Data Storage and Processing in the Context of the LHC Experiments at CERN," PhD thesis, Dept. Computer Science, Technische Universität München, 2012.
8. J. Blomer et al., "Micro-CernVM: Slashing the Cost of Building and Deploying Virtual Machines," *J. Physics: Conf. Series*, vol. 513, no. 3, 2014, article no. 032009.
9. P. Gauthier et al., "Web Proxy Auto Discovery Protocol," IETF Internet Draft Working Document, 1999; <https://tools.ietf.org/html/draft-ietf-wrec-wpad-01>.
10. I. Gable et al., "Dynamic Web Cache Publishing for IaaS Clouds Using Shoal," *J. Physics: Conf. Series*, vol. 513, no. 3, 2014, article no. 032035.
11. N. Tolia et al., "Opportunistic Use of Content Addressable Storage for Distributed File Systems," *Usenix Ann. Tech. Conf., General Track*, vol. 3, 2003, pp. 127–140.
12. R.M. Karp and M.O. Rabin, "Efficient Randomized Pattern-Matching Algorithms," *IBM J. Research and Development*, vol. 31, no. 2, 1987, pp. 249–260.
13. K. Kutzner, "The Decentralized File System Igor-FS as an Application for Overlay-Networks," PhD thesis, Dept. Computer Science, University of Karlsruhe, 2008.
14. F. Hrbata, "Callback Framework for VFS Layer," MS thesis, Dept. Information Tech., Brno University of Technology, 2005.
15. M.J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing Content Publication with Coral," *Proc. Usenix Symp. Network Systems Design and Implementation (NSDI)*, vol. 4, 2004, p. 8.
16. E. Nygren, R.K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," *ACM SIGOPS Operating Systems Rev.*, vol. 44, no. 3, 2010, pp. 2–19.
17. H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification," IETF RFC 2094, 1997; <https://tools.ietf.org/html/rfc2094>.

Jakob Blomer is a computer scientist in the scientific software group at CERN. His research interests include distributed storage systems and cloud computing. Blomer received a PhD in computer science from the Technical University of Munich. Contact him at jblomer@cern.ch.

Predrag Buncic is a senior applied physicist at CERN. His research interests include software and computing architecture and data management. Buncic received an MS in physics from the University of Belgrade, Serbia. Contact him at predrag.buncic@cern.ch.

René Meusel is a software developer at CERN. His research interests lie both in system software architecture and human-computer interaction. Meusel received an MS in IT systems engineering from the Hasso Plattner Institute in Potsdam. Contact him at rene.meusel@cern.ch.

Gerardo Ganis is a senior applied physicist at CERN. He contributed to the ROOT and XRootD projects, in particular in the areas of remote file access, distributed parallel analysis, authentication, and authorization. Ganis received an MS in physics from the University of Trieste, Italy. Contact him at gerardo.ganis@cern.ch.

Igor Sfiligoi was a senior research software developer at the University of California, San Diego, at the time of writing; he's currently a senior software engineer at Teradata. His research interests include distributed computing and big data. Sfiligoi received an MS in computer science from Universita Degli Studi di Udine and an MS in security science from the EC Council University. Contact him at igor.sfiligoi@gmail.com.

Douglas Thain is an associate professor in the Department of Computer Science and Engineering at the University of Notre Dame. His research team creates and publishes open source software that's used around the world to harness large-scale computing systems such as clusters, clouds, and grids. Thain received a PhD in computer science from the University of Wisconsin–Madison, where he contributed to the Condor distributed computing system. Contact him at dthain@nd.edu.