EPJ Web of Conferences **245**, 03015 (2020) https://doi.org/10.1051/epjconf/202024503015
CHEP 2019

Distributing User Code with the CernVM FileSystem

Dave Dykstra[1,*], *Shreyas* Bhat[1], *Dennis* Box[1], *Hyun Woo* Kim[1], *Tanya* Levshina[1]

[1]Scientific Computing Division, Fermilab, Batavia, IL, USA

Abstract. The CernVM FileSystem (CVMFS) is widely used in High Throughput Computing to efficiently distributed experiment code. However, the standard CVMFS publishing tools are designed for a small group of people from each experiment to maintain common software, and the tools are not a good fit for publishing software from numerous users in each experiment. As a result, most user code, such as code to do specific physics analyses, is still sent with every job to the place the job is run. That process is relatively inefficient, especially when the user code is large. To overcome these limitations, we have built a CVMFS user code publication system. This publication system enables users to still submit their code with their jobs but the code is distributed and accessed through the standard CVMFS infrastructure. The user code is automatically deleted from CVMFS after a period of no use. Most of the software for the system is available as a single self-contained open source rpm called cvmfs-user-pub and is available for other deployments.

1 Introduction

The CernVM File System [1] (CVMFS) is widely used in High Throughput Computing (otherwise known as grid computing) to distribute the software for many experiments in the scientific community. Its highly effective use of caches, execution of metadata operations on the client, file deduplication and compression, and download on demand make it very efficient and convenient to access code on large numbers of computers worldwide.

CVMFS distributes software through a publish process that works well for small groups of people responsible for the shared code used by large experiments, but it is not designed for the much larger number of researchers ("users") that write relatively small additions to the experiment code. As a result, most user code, for example code to do physics analysis, is sent along with the jobs that users run on the grid. That process is relatively inefficient compared to CVMFS, especially when the user code is large. Many copies of the same large set of files can end up being sent across the world, even when the jobs are all running at the same site.

This paper discusses the user code distribution system based on CVMFS that we have written and deployed at Fermilab.

2 Motivation

The primary motivation to make the user code publication system at Fermilab came from the evolution of the architecture of our local computing cluster. Originally, each worker node mounted filesystems from a high bandwidth Network Attached Storage Server (NAS) using

* Corresponding author: dwd@fnal.gov

the Network File System (NFS) protocol. Even though the NAS had high bandwidth, many jobs from a single user could easily overload it and often did if users were not careful to limit the rate of data access. In addition, relying on locally mounted filesystems prevented user jobs from taking advantage of grid computing at remote sites. The NAS hardware was also aging and needed to be replaced, and the chosen replacement technology (dCache [1]) worked best when not accessed using POSIX-based semantics such as those provided by NFS. For those reasons, it was decided to remove the NFS mounts from the NAS on the worker nodes.

Removing the NFS mounts meant that many more users needed to start distributing their custom code to jobs, since previously they had executed the code directly over NFS. The recommended replacement was to put their code into compressed tar files (tarballs) on dCache and submitting them to be distributed with their jobs. That very quickly overloaded individual dCache file servers because many of the hundreds or thousands of jobs in a batch wanted to download a single file soon after the jobs started. Some of the tarballs were as large as 3GB. The temporary solution was to use dCache "resilient pools" which made identical copies of each of these files on 20 different servers. This was highly wasteful of disk space, disk server bandwidth, and network bandwidth, both the local network when jobs were run locally and the wide area network when jobs were run on the grid. A new solution was needed and basing it on CVMFS seemed promising.

3 Requirements

After discussions with users the following requirements for building a code distribution service based on CVMFS were identified:

1. Since users submit their code tarballs with jobs, publication and distribution to worker nodes must be faster than normal so as not to keep the jobs waiting for long. Code must be available on worker nodes in less than 5 minutes in most cases.
2. The system must support tarballs up to 3 GB in size.
3. The system must be performant enough to accommodate hundreds of new tarball publications per day.
4. The system must be reliable, not subject to a single point of failure.
5. Published files must be automatically removed when they are no longer used. User code is assumed to change frequently and users separately keep track of their own source code.

4 Design

In this section we will discuss several aspects of the design that was settled upon.

4.1 System Design

These are the basic aspects of the system design:

1. Use two publishing servers for redundancy, not shared for other purposes.
2. Each server provides a web API, authenticated by the same X.509 proxy certificates used to submit jobs. Clients connect randomly to either server.
3. Each code tarball is assigned a unique Code ID (CID) by the client, based on a hash of the contents.
4. Clean out old tarballs automatically (details in section 4.4 below).

5. Publishes happen in two CVMFS repositories on each server (a total of 4 repositories), so when cleaning up happens it will not block publication. When cleanup is not happening, publishing can be done in parallel to the two repositories.
6. Clients directly upload tarballs to a publishing server.
7. Publishing servers unpack each tarball into a directory name based on the CID.
8. Minimize the distribution delays (details in section 4.5 below).
9. The CID is passed to a job wrapper script which waits for the CID directory to appear in any of the four repositories and passes the directory path to the job. The script times out with an error if it takes too long for the CID to appear.

4.2 Control Flow

Tarball publishing is integrated with the Fermilab job submission system, which is called Jobsub [3]. Jobsub itself uses a client/server model, and the client is called jobsub_submit. Figure 1 shows the control flow between all the pieces of the system. Step 4 is an optimization to avoid uploading if a given CID has already been published with a previous job submission. Copies of all four repositories are kept on both servers so they can each quickly tell whether or not a CID is already published.

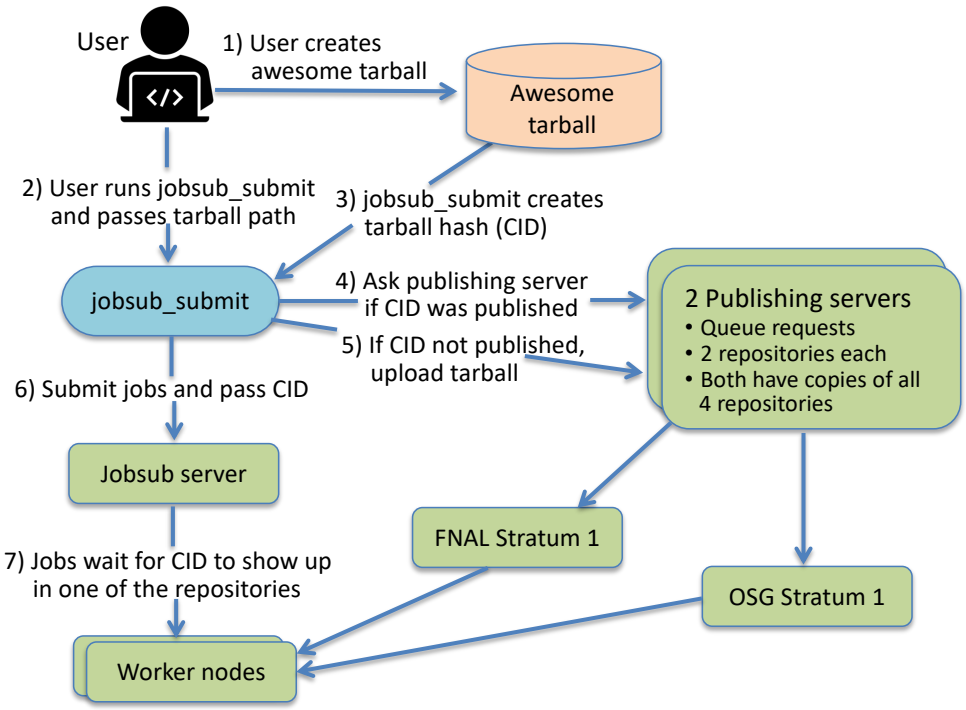

Figure 1. Control flow of user tarball publication to CVMFS

4.3 API

The API for the publishing server is simple and can be easily accessed through curl or any other https-capable web client library or tool. These are the primitives:

1. **/pubapi/publish?cid=XXX** – used with the POST method to upload a tarball to be unpacked and published in any CVMFS repository. The user's X.509 proxy must be used to authenticate the https connection, and the user's certificate Distinguished Name (DN) must be listed in the grid-mapfile on the publishing server. The body of the response is OK when the tarball is queued for publishing, or PRESENT if the CID XXX was already published. If PRESENT, the access timestamp (used for cleanup) is also updated. The CID can be any value, not verified by the server, and may include a slash to put the unpacked files into a subdirectory. The Jobsub client uses the submitter's Virtual Organization name in the top level directory so the code gets put in a group by experiment.
2. **/pubapi/exists?cid=XXX** – checks whether the CID already exists and returns PRESENT if it does or MISSING if it does not. Also requires https and the user's proxy certificate.
3. **/pubapi/update?cid=XXX** – the same as the exists API, except it also updates the access timestamp.
4. **/pubapi/config** – returns the configuration, which is currently just a list of CVMFS repository names. May be used over http or without a proxy certificate.
5. **/pubapi/ping** – simply returns OK. Used for monitoring and used by the load balancer to determine whether or not a server is responding. Also does not require https or a proxy certificate.

4.4 Repository Cleanup

Cleanups on the publishing servers happen once a day and one repository per hour beginning at a configurable hour of the night. Access timestamps are stored in any of the CVMFS repositories, not necessarily the same one in which the tarball was published. When doing the cleanup process for a repository, the server looks at each CID directory in that repository and uses the newest access timestamp in all four repositories to determine how hold the CID is. If the newest timestamp is older than a configurable number of days (default 30), the CID directory is removed. It also looks at each timestamp in that repository, and if the corresponding CID directory does not exist in any of the four repositories anymore it removes the timestamp file.

Each cleanup process is immediately followed by CVMFS garbage collection to remove deleted files from the server from the same repository. During both of the cleanup steps the repository is locked from publishing new tarballs, but other repositories can still publish.

4.5 Minimizing Distribution Delays

In addition to the dedicated publishing resources, the following steps were taken to minimize the time it takes to distribute updates to worker node clients.

1. The amount of time that a cvmfs client waits between checking for updates is set in a configuration option on the publishing server. The default is 4 minutes, but for these repositories we set it to 15 seconds.
2. Stratum ones vary in the amount of time they wait between checking for updates. The OSG stratum 1 has performant hardware, so it has been configured to start a new process serially checking all repositories for updates every 15 seconds. The FNAL stratum 1 normally only starts such a process every 5 minutes, but it has now been configured to start a separate process checking only these 4 repositories twice a minute. It normally reads repositories from the OSG stratum 1, but it reads these

4 repositories directly from the publishing servers to eliminate any extra delay. The time to transfer varies depending on the size; most take just a couple of seconds but even the largest one we have observed so far, which was the first publication from a user of a 1GB tarball, took less than 15 seconds. Most of the time a large percentage of the files are already published from a previous publication.

3. The amount of time that a squid cache waits between checking for updates is set by the apache web server on the Stratum 1s, and that time is set to 61 seconds on most Stratum 1s because that's the minimum that the default configuration of squid allows.
4. Depending on the cvmfs client version, there may be an additional 60 second delay to flush the kernel buffer cache. The most recent cvmfs client version does not have this delay.

With these minimal settings, worker node jobs should normally see updates less than 2 or 3 minutes after their tarballs are published. The time to publish is the longest step; the 1GB tarball that we observed took around 75 seconds to publish.

5 Packaging

The software for the publishing server is packaged in an rpm for Red Hat Enterprise Linux. It is available as open source code on github [4] and also as a prebuilt rpm in cvmfs-contrib [5]. The rpm is configured by a simple configuration file that primarily only contains the names of the repositories and the hosts they are on. Installing it automatically installs all other required packages. The rpm takes care of creating the cvmfs repositories for the local host and creating replicas of the repositories from other hosts. It provides the apache httpd configuration and web api, and does the automatic cleanup. The only other significant thing that a system administrator needs to set up is a grid-mapfile that lists the accepted user certificate Distinguished Names (DNs).

6 Conclusions

The user code publishing system based on CVMFS that we have described is easy to deploy and reliably distributes user code to grid jobs in a very efficient manner compared to alternatives. The software is readily available for other people to deploy in other applications.

Acknowledgements

This document was prepared using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359.

References

1. J. Blomer, et. al., J. Phys. Conf. Ser. **396**, 052013 (2012)
2. http://www.dcache.org
3. D. Box, J. Phys. Conf. Ser. **513,** 032010 (2014)
4. https://github.com/cvmfs-contrib/cvmfs-user-pub
5. https://cvmfs-contrib.github.io