Distributed cluster testing using new virtualized framework for XRootD

Justin L Salmon\(^1,2\) and Lukasz Janyst\(^1\)

\(^1\)European Organisation for Nuclear Research (CERN), CH-1211 Geneva 23, Switzerland
\(^2\)University of the West of England, Frenchay Campus, Coldharbour Lane, Bristol, BS16 1QY, United Kingdom

E-mail: jsalmon@cern.ch, ljanyst@cern.ch

Abstract. The Extended ROOT Daemon (XRootD) is a distributed, scalable system for low-latency clustered data access. XRootD is mature and widely used in HEP, both standalone and as a core server framework for the EOS system at CERN, and hence requires extensive testing to ensure general stability. However, there are many difficulties posed by distributed testing, such as cluster set up, synchronization, orchestration, inter-cluster communication and controlled failure handling. A three-layer master/hypervisor/slave model is presented to ameliorate these difficulties by utilizing libvirt and QEMU/KVM virtualization technologies to automate spawning of configurable virtual clusters and orchestrate multi-stage test suites. The framework also incorporates a user-friendly web interface for scheduling and monitoring tests. The prototype has been used successfully to build new test suites for XRootD and EOS with existing unit test integration. It is planned for the future to sufficiently generalize the framework to encourage usage by potentially any distributed system.

1. Introduction

Testing distributed systems is significantly more complicated than testing non-distributed systems. Several new challenges are posed, such as scalability, monitoring and control, synchronization, reproducibility and fault tolerance [1].

Performing conventional unit tests alone does not provide sufficient coverage for a software system which operates in a distributed environment. It is necessary to be able to simulate real-world, functional tests for a variety of potential cluster configurations and to be able to control, or “orchestrate”, the execution of these tests across each participating machine. It is also necessary to be able to automate the testing process and to allow support for existing continuous integration platforms.

In this paper we present a system which attempts to facilitate and automate the aforementioned tests as much as possible. The framework allows for the configuration and automatic generation of clustered network setups, and for definition and controlled execution of multi-stage test suites on these clusters. It was primarily designed for use with the XRootD system [2], but is generic enough to be used with other distributed software.

We begin by describing the general architecture of our framework. We then outline the structure, configuration and operational control flow of a “test suite”. Brief implementation details and an outline of the web-based user interface are also given.
2. Architecture

The architecture of a deployed XRootD Test Framework is composed of three layers organised in a hierarchy. The bottom layer is comprised of multiple slaves, each of which has a corresponding hypervisor in the middle layer. Each hypervisor in turn is controlled by the master.

Figure 1. The XRootD Test Framework 3-layer architecture

2.1. The master layer

The master is a unix daemon responsible for synchronizing and orchestrating all system activities. It exports the current status and results of the test runs via a web interface. It fetches the cluster and test suite descriptions from a git repository.

The master also has a cron-like scheduling system for test suites, and email notifications. The master accepts connections from hypervisors and slaves, and dispatches commands to them. The daemon typically runs on a dedicated machine and can be easily configured as a standard RHEL-based system service.

2.2. The hypervisor layer

Hypervisors receive cluster (machine and network) configurations from the master and are responsible for starting/stopping/configuring their virtual machines (slaves) and virtual networks accordingly. The networks may support NAT, so the ”outside world” is visible to the slave machines. Hypervisors provide a thin translation functionality layer between the master’s requests and the libvirt[3] virtualization framework. KVM/QEMU is used as the actual virtualization technology. Hypervisors are also implemented as unix daemons.

2.3. The slave layer

The slaves are responsible for actually running tests. Each slave is run inside its own virtual machine. Their only task is to run shell scripts and send back the results, together with the relevant log files.

2.4. Cluster configuration

Each cluster is defined in its own configuration file. An example is not given here for brevity*. The cluster configuration file allows the user to specify the following:

*For examples of cluster and test suite configuration files, see https://github.com/xrootd/xrootd-test-suite/blob/master/clusters/cluster_meta_manager.py
(i) The number of slave VMs to be spawned in the cluster;
(ii) The IP and MAC addresses of each slave;
(iii) The reference disk image to boot each slave from;
(iv) The slave base architecture (x86, x64) and RAM size;
(v) Arbitrary numbers of additional disks, their sizes, and device names;
(vi) Network topology (DNS names, aliases, netmasks)

2.5. Test suite configuration

Each test suite is defined in its own configuration file, along with a collection of scripts in a predetermined directory structure. A test suite is comprised of three stages: initialization, run and finalization. The run stage carries out each test case. A test case is also divided into initialization, run and finalization stages. Figure 2 gives an overview of a test suite flow, from global initialization to finalization.

A test suite configuration file specifies which test cases are to be run within the test suite; a cron-like scheduler expression to determine when the test suite will be run; an arbitrary number of log file paths to be pulled from each slave after each test stage; and a list of email addresses to be notified on success/failure of the test suite along with corresponding alert policy specifications.

3. Implementation

The framework has been implemented using Python. It is compatible with Python versions 2.4 through 2.7 and has been tested on SLC5, SLC6 and Ubuntu. The framework has dependencies on a number of third-party Python libraries, namely the libvirt-python bindings to access libvirt functionality from Python on the hypervisors; python-cherrypy and python-cheetah for the web-based user interface; python-apscheduler for the cron-like test suite scheduling mechanism; and pyinotify to monitor changes to test suite repositories and automatically reload them.

4. Web-based user interface

The web interface that the master exports allows the current status of each test suite, as well as the status of each past test run, to be viewed. The test suite configuration allows specification
of which log files are to be collected and uploaded to the master from each slave after each test stage. These log files are then displayed, giving a detailed and highlighted view of what is happening at each stage. A list of connected hypervisors, slaves, and their hardware and network configurations are also visible. A screenshot of the XRootD Test Framework is given in Figure 3.

![Screenshot of framework web interface.](image)

**Figure 3.** Screenshot of framework web interface.

### 5. Improvements

Currently, each slave must have a separate boot image. This takes up a large amount of storage space, as well as time to clone the final disk image from the reference while initializing the cluster (although this copying need only happen once, the first time the cluster is initialized). Shared storage techniques such as unionfs/overlayfs may help reduce the amount of space required and time to boot the cluster up. IPv6 support needs to be added as well, in order to be able to cover this growing class of applications. Fortunately the newest versions of the *libvirt* framework already support IPv6.

### 6. Conclusion

We have managed to ameliorate some of the difficulties of functional testing of distributed setups with our framework. We have a working installation at CERN. This includes test suites for various kinds of XRootD setups, which have already proven themselves to be useful in the debugging of incremental builds. We will likely be using the framework for other projects as well.
References
[1] S. Ghosh and A. P. Mathur. Issues in testing distributed component-based systems. In First International ICSE Workshop on Testing Distributed Component-Based Systems, Los Angeles, 1999.
[2] XRootD home page http://xrootd.org
[3] The libvirt framework http://libvirt.org
[4] Git: distributed version control system http://http://git-scm.com/