Application-Level Regression Testing Framework using Jenkins

Author1, Timothy Bouvet, Galen Arnold
National Institute for Computational Sciences
The University of Tennessee, Knoxville, TN 37996
{email1,tbouvet@illinois.edu,email3}@domain.edu

***Abstract*—This paper will explore the challenges of regression testing and monitoring of large scale systems such as NCSAs Blue Waters. Our goal was to come up with an automated solution for running user-level regression tests to evaluate system usability and performance. Another requirement was to find an automated solution for running a suite of test jobs that reveal the system health after upgrades and outages before returning the system to service. We evaluated test frameworks such as Inca [1] and Jenkins [2]. Jenkins was chosen for its versatility, large user base, and multitudes of plugins including plotting test results over time. We utilize these plots to track trends and alert us to system-level issues before they are reported by our partners (users). Not only does Jenkins have the ability to store historical data but it can also send customized notifications (e.g. send email or text pages) based on the result of a test. Some of the requirements we had include two-factor authentication to access Jenkins GUI with privileges to execute tests and account management through LDAP. In this paper we describe our implementation of these requirements to ensure a secure and usable deployment of a Jenkins instance.**

Our Jenkins instance was deployed on a vSphere managed VM running Centos 6.8. Security was of the highest concern since Jenkins has the ability to execute commands on our login nodes. Jenkins is set to log in as a standard user account with passwordless access (commands to the login nodes can be input via Jenkins GUI) using ssh keys scoped to the VM. The VM (bwjenkins) was closely vetted by our security team on our test and development system (TDS) before it was allowed access to Blue Waters. Iptables was used to lock bwjenkins down to a small internal ip space to ensure the highest security. An anonymous user account was enabled with read-only access to view current and historical test results. During a test, Jenkins downloads software, builds the software, submits a job, awaits completion, gathers and plots the results or returns error information during a failure. These are full end to end functionality testing of the programming environment, queueing system, license servers and provide historical metrics to quickly detect regressions.

In this paper we describe in detail our challenges and experiences in deploying Jenkins as a user-level system-wide regression testing and monitoring framework for Blue Waters. We will also show some application-based system-level tests we have implemented in Jenkins and share results of those tests. The deployment of Jenkins allow us to monitor and detect issues as early as possible from the perspective of a user on Blue Waters, eventually providing a more efficient service for better user experience.

***Keywords*-Performance; Benchmarking; Computer architecture**

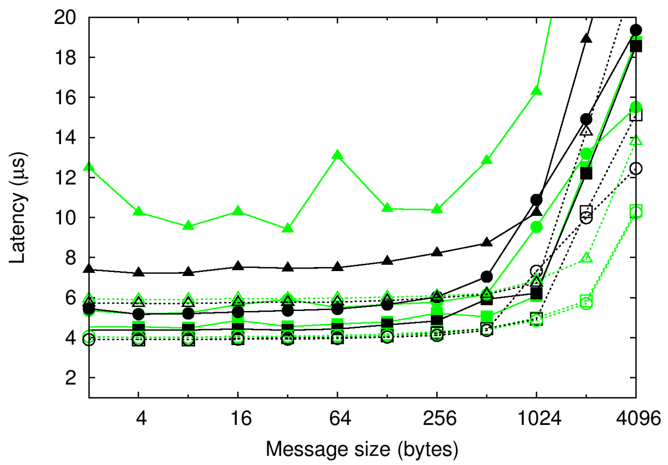


Figure 1: This is an example figure with its caption.

Listing 1: Apache Configuration Example

```
# First, we configure the "default" to be a very
# restrictive set of features.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

I. Motivation

II. Jenkins Configuration

A. Introduction to Jenkins

B. Master-Node Configuration

C. Accessing Login Nodes

III. Authentication and Authorization

A. Security Considerations

B. Configuration Choices

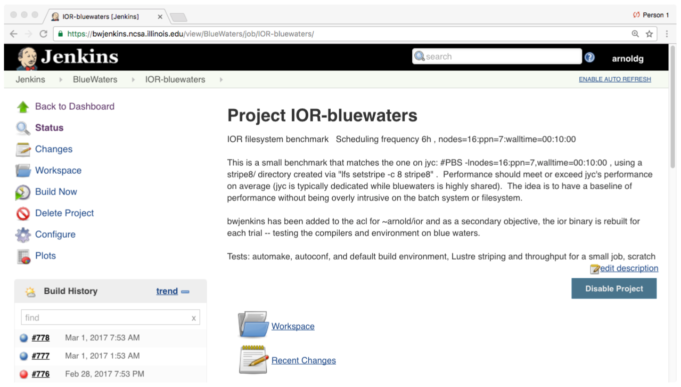
IV. Anatomy of a Test

V. SWTools Integration

VI. Use Cases

A. IOR Lustre scratch test

We created Jenkins tests to run the IOR MPI parallel filesystem test to run at a modest scale and validate filesystem functionality and performance for the scratch filesystem on a periodic basis. One of our best practices is to provide a management level summary description for casual jenkins users who want to view the test but do not need to understand the jenkins configuration and execution details. Where possible, tests build from source and exercise



Project IOR-bluewaters

IOR filesystem benchmark   Scheduling frequency 6h , nodes=16:ppn=7:walltime=00:10:00

This is a small benchmark that matches the one on jyc: #PBS -lnodes=16:ppn=7,walltime=00:10:00 , using a stripe8/ directory created via "lfs setstripe -c 8 stripe8" . Performance should meet or exceed jyc's performance on average (jyc is typically dedicated while bluewaters is highly shared). The idea is to have a baseline of performance without being overly intrusive on the batch system or filesystem.

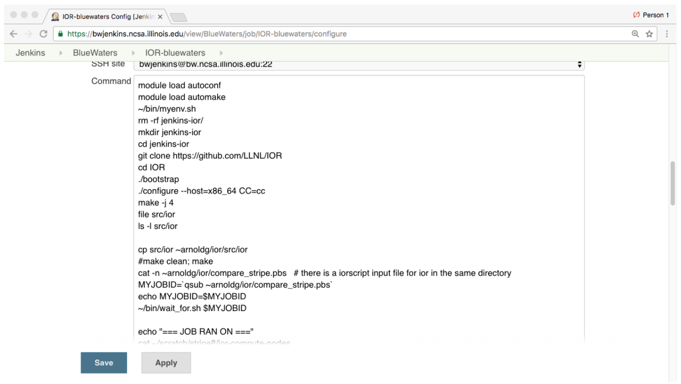
bwjenkins has been added to the acl for ~arnold/ior and as a secondary objective, the ior binary is rebuilt for each trial -- testing the compilers and environment on blue waters.

Tests: automake, autoconf, and default build environment, Lustre striping and throughput for a small job, scratch

#778	Mar 1, 2017 7:53 AM
#777	Mar 1, 2017 1:53 AM
#776	Feb 28, 2017 7:53 PM

Figure 2: IOR description in jenkins

multiple user-facing system components such as: git for external connectivity and loading modules to test the defaults in the environment. Where scripts on the SSH site are used, they are echoed and/or displayed with "cat -n" to make console output verbose and easy to debug in case of errors.


IOR-bluewaters Config [Jenkin
Person 1
https://bwjenkins.ncsa.illinois.edu/view/BlueWaters/job/IOR-bluewaters/configure
Jenkins
BlueWaters
IOR-bluewaters
Save
Apply

SSH site bwjenkins@bw.ncsa.illinois.edu:22

Command

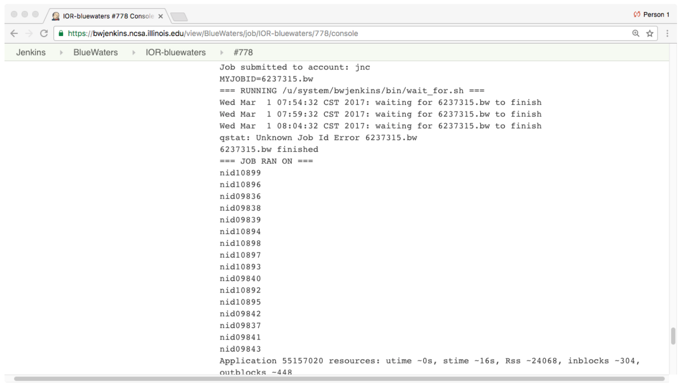
```
module load autoconf
module load automake
~/bin/myenv.sh
rm -rf jenkins-ior/
mkdir jenkins-ior
cd jenkins-ior
git clone https://github.com/LLNL/IOR
cd IOR
./bootstrap
./configure --host=x86_64 CC=cc
make -j 4
file src/ior
ls -l src/ior

cp src/ior ~arnoldg/ior/src/ior
#make clean; make
cat -n ~arnoldg/ior/compare_stripe.pbs   # there is a iorscript input file for ior in the same directory
MYJOBID=`qsub ~arnoldg/ior/compare_stripe.pbs`
echo MYJOBID=$MYJOBID
~/bin/wait_for.sh $MYJOBID

echo "=== JOB RAN ON ==="
```

Figure 3: IOR configuration sample

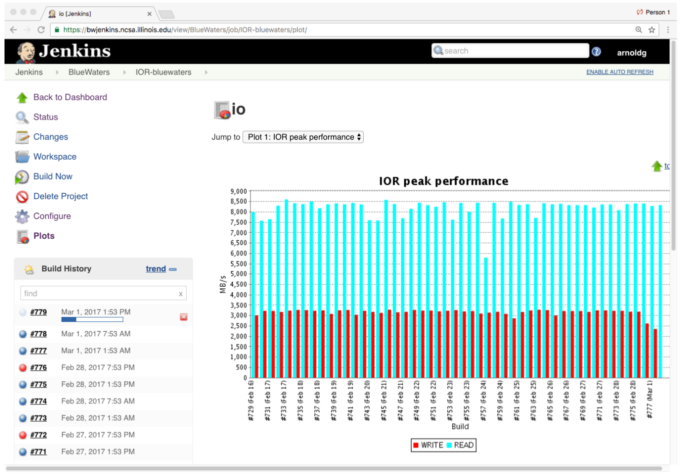
The test is set to run as scheduled by Jenkins with a best-effort through our batch system. The test blocks such that the next test will not start until the previous one has completed. We employ a watchdog script to block the exit from the test until the batch system marks it as finished.


IOR-bluewaters #778 Console
Person 1
https://bwjenkins.ncsa.illinois.edu/view/BlueWaters/job/IOR-bluewaters/778/console
Jenkins
BlueWaters
IOR-bluewaters
#778

```
Job submitted to account: jnc
MYJOBID=6237315.bw
=== RUNNING /u/system/bwjenkins/bin/wait_for.sh ===
Wed Mar  1 07:54:32 CST 2017: waiting for 6237315.bw to finish
Wed Mar  1 07:59:32 CST 2017: waiting for 6237315.bw to finish
Wed Mar  1 08:04:32 CST 2017: waiting for 6237315.bw to finish
qstat: Unknown Job Id Error 6237315.bw
6237315.bw finished
=== JOB RAN ON ===
nid10899
nid10896
nid09836
nid09838
nid09839
nid10894
nid10898
nid10897
nid10893
nid09840
nid10892
nid10895
nid09842
nid09837
nid09841
nid09843
Application 55157020 resources: utime ~0s, stime ~16s, Rss ~24068, inblocks ~304,
outblocks ~448
```

Figure 4: IOR watchdog console output

For tests that produce an interesting performance metric like IOR, we save that output to a file and use the Jenkins plotting feature to produce plots.



#779	Mar 1, 2017 1:53 PM
#778	Mar 1, 2017 7:53 AM
#777	Mar 1, 2017 1:53 AM
#776	Feb 28, 2017 7:53 PM
#775	Feb 28, 2017 1:53 PM
#774	Feb 28, 2017 7:53 AM
#773	Feb 28, 2017 1:53 AM
#772	Feb 27, 2017 7:53 PM
#771	Feb 27, 2017 1:53 PM

Figure 5: IOR plot view

A plot like this helps us with performance baselines and we can quickly react to changes in the software environment or filesystem configuration.

B. *mdtest Lustre tests*

We created Jenkins tests to run the IOR MPI parallel filesystem test to run at a modest scale and validate filesystem functionality and performance for the scratch filesystem on a periodic basis. The mdtest metadata test is also run on the home and scratch filesystems to validate metadata server performance. Both tests create plots that we review when checking recent overall filesystem performance. These user-facing tests complement the fine-grained detail we are able to glean from the backed system-side counters in our OVIS database where we record details about individual server metrics and network traffic on the Cray Gemini fabric. Both of the tests are set to run as scheduled by Jenkins with a best-effort through our batch system. The tests block such that the next test will not start until the previous one has completed through the batch system. We employ a watchdog script to block the exit from the test until the batch system marks it as finished.

VII. Conclusion

Acknowledgment

This research used resources at the National Institute for Computational Sciences, funded by the National Science Foundation (NSF).