An Approach to Continuous Testing

Francine Lapid
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, USA
lapid@lanl.gov

Shivam Mehta
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, USA
smehta@lanl.gov

***Abstract*—The National Nuclear Security Administration Department of Energy supercomputers at Los Alamos National Laboratory (LANL) are integral to supporting the lab's mission and therefore need to be reliable and performant. To identify potential problems ahead of time while minimizing the interruption to the users' work, the High Performance Computing (HPC) Division at LANL implemented a Continuous Testing framework and the necessary infrastructure to be able to automatically and frequently run a series of tests and proxy applications. The tests, which benchmark various system components, were integrated into the Pavilion2 testing framework and are launched in small Slurm jobs across each machine every weekend. The results of the tests are summarized in a comprehensive Splunk dashboard, enabling continuous monitoring of the health of the machines over time without having to parse through each run's output and logs. This project is currently running on all of LANL's newest fleet of Cray Shasta machines: Chicoma, Crossroads, Razorback, Rocinante, and Tycho. This paper details the different components of the Continuous Testing framework, the resulting setup, and the impact it has on our HPC workflow.**

***Index Terms*—high performance computing, continuous testing, Splunk, Pavilion2**

I. Introduction

The National Nuclear Security Administration Department of Energy high performance supercomputers at Los Alamos National Laboratory (LANL) run mission critical codes related to national security. To ensure the supercomputers are reliable and performant, the High Performance Computing Division (HPC) needs to periodically test the machines to identify potential problems. The Programming and Runtime Environments Team (PRETeam), the team responsible for system testing, runs full-scale tests to identify problems prior to returning the system to the users after taking the system down for Dedicated System Time (DST)[1]. While it is widely agreed upon that thorough testing is an important step of the maintenance process, it does take time away from users to be able to do their work.

DST Testing often took 2-3 hours on top of several hours-to-days maintenance work. When LANL supported multiple different clusters with the same operating systems and file system mounts, users were able to do their mission-critical work on a different machine if their preferred machine was down for maintenance. However, over the past year, LANL's fleet of supercomputers has been drastically reduced in favor of a small number of fast and efficient supercomputers. These new supercomputers perform 4 to 8 times better than previously commissioned systems at LANL. [1] At the same time, due to the small number of supercomputers, the DSTs had to be reduced both in frequency and duration to ensure maximum supercomputer availability. While the system administrators are able to implement and apply updates using the rolling reboot strategy[2], system testers needed to find a new way to validate the systems.

To overcome the system validation issues, the PRETeam implemented the Continuous Testing framework. The framework applies the Pavilion2 test harness and combines it with cron jobs to automatically run a series of tests on a weekly basis on LANL's newest fleet of Cray Shasta machines: Chicoma, Crossroads, Razorback, Rocinante, and Tycho. This project also included creating Splunk dashboards to facilitate the evaluation of the data from the tests.

This paper discusses the test selection criteria, development of tests and series using the Pavilion2 test harness, refining the current infrastructure to allow the benchmarks to run, and forwarding the test results to a Splunk Indexer for results evaluation.

II. Test Selection

The purpose of the Continuous Testing project is to run a predetermined set of tests periodically that properly monitors the health of the systems over time while minimizing the disruption to users' work. The tests should somewhat adhere to the following standards:

- Each test should attempt to simulate a users' workload, hence, the preference for benchmarks and proxy applications.
- Each test should evaluate a particular system component.
- Each test should be completed within 10 minutes so as to not take too many compute cycles from the users.

Initially, the list was composed of benchmarks from the Crossroads Acceptance Testing[3] effort as those tests were already configured to work on Shastas and integrated into

[1]Time allocated for system maintenance, upgrades, and testing.

[2]A strategy used by administrators to apply updates and perform a node reboot after completion of an active job. From the user's perspective, the nodes take slightly longer to return to the queue causing almost no disruption to their work. [2]

[3]https://mission.lanl.gov/advanced-simulation-and-computing/platforms/crossroads/benchmarks/

TABLE I
CURRENT LIST OF TESTS[a] IN THE CONTINUOUS TESTING FRAMEWORK

Test Name	System Component	Description
Flexible I/O Tester	Filesystems	Simulates different I/O workloads.
Gromacs - Water Benchmark	CPU & GPU	Water molecule simulation timings.
HPCG	CPU & Network	Represents workload of modern applications.
Jacobi	GPU	Distributed Jacobi solver.
Mem-Info	Memory	Records memory information.
Module-Timing	Environment Modules	Module commands timings.
Perl-Perf	CPU	Runs and times math calculations in Perl.
Stream	Memory	Memory bandwidth test.
VPIC	Memory	Particle-In-Cell plasma simulation.

[a]Some integrated user codes are also part of the framework, however, they are not listed here due to internal policies.

Pavilion2. However, of those tests, only HPCG, VPIC, and STREAM[4] comply with the requirements above. Although these tests do check the performance of the CPU, memory, and network, they do not test the GPUs and the filesystems, hence the addition of Gromacs, Jacobi, and the Flexible I/O Tester tests. The list was further expanded by evaluating common issues faced by the users and adding workload specific tests.

III. THE FRAMEWORK

For the Continuous Testing framework, there were a few required components that allowed for automatic job scheduling, execution, and evaluation. Pavilion2, the cron daemon, and Splunk were chosen as these components were already used in PRETeam's workflow in some capacity.

A. Pavilion2

Pavilion2 is an open-source Python 3-based testing framework largely developed at LANL for running and analyzing tests for HPC systems. [3] It is regularly used at LANL to verify the production-readiness of a system after it has undergone maintenance. Configurations for selected tests, a series file, and a mode file were created for Pavilion2.[5]

1) Test Configurations: The test configuration step involves writing yaml definitions to generate build, scheduler, run, and parser scripts. The following code blocks will briefly explain the yaml definitions used to generate each component for the HPCG test.

a) Variables: The variables section is used to replace test configuration values using double curly brackets. For HPCG, it is used to define the input deck, openmp argument, compiler command, and the mpich directory. The input deck is used to pass arguments to the executable while the rest of the variables are used to generate the makefile.

```
variables:
  input_deck:
    npx: 6
    npy: 6
    npz: 3
    nx: 88
    ny: 88
    nz: 88
  openmp: '-qopenmp'
  cxx: 'CC'
  mpi_dir: '$(CRAY_MPICH_DIR)'
```

b) Build: The build section is used to generate the build script. The `source*` keys informs Pavilion2 of the location of the source code (github url), when to download (only when the source code is missing), and the download location (hpcg/src directory). The `modules` key contains a list of modules that Pavilion needs to load to set up the build environment. The `templates` key is used to generate a system specific makefile. The template make file replaces some of the values with the ones defined in the variables sections and saves the file as `Make.<system_name>` which is used by the make command in the `cmds` section to build the executable.

```
build:
  source_url: https://github.com/hpcg-benchmark\
              /hpcg/archive/refs/heads/master.zip
  source_download: 'missing'
  source_path: 'hpcg/src'
  templates:
    hpcg/Make.tmpl: ./setup/Make.{{sys_name}}
  modules:
    - 'intel'
    - 'cray-mpich'
  cmds:
    - 'make arch={{sys_name}}'
```

c) Scheduler: The scheduler section is used to define the scheduler and pass scheduler options. In HPCG's case, it specifies to use the Slurm scheduler while passing the number of nodes, tasks per node, the partition, and the quality of service as the arguments.

```
scheduler: slurm
schedule:
  nodes: 1
  tasks_per_node: 108
  partition: 'standard'
  qos: 'standard'
```

[4]Originally, the IOR micro-benchmark was chosen for filesystems tests, however, it was quite aggressive and caused filesystems unavailability for the user's jobs during its run. It was later removed and replaced by the Flexible I/O Tester.

[5]The following sub-sections provides a brief introduction to writing Pavilion2 test configurations, mode files, and series files. For detailed information, please refer to: https://pavilion2.readthedocs.io/en/latest/index.html

d) Run: The run section contains the information to set up the run environment and then run the binary. The env key specifies the environment variables to export, the modules to load and the commands to run. The `cmds` key after variable replacements would be: `srun -N 1 -p standard -q standard --tasks_per_node=108 ./bin/xhpcg --npx=6 --npy=6 --npz=3 --nx=88 --ny=88 --nz=88`.

```
run:
  env:
    OMP_NUM_THREADS: 2
  modules:
    - 'intel'
    - 'cray-mpich'
  cmds:
    - "{{sched.test_cmd}} ./bin/xhpcg \
       --npx={{input_deck.npx}} \
       --npy={{input_deck.npy}} \
       --npz={{input_deck.npz}} \
       --nx={{input_deck.nx}} \
       --ny={{input_deck.ny}} \
       --nz={{input_deck.nz}}"
```

e) Results: The result parse section is mainly used to capture and parse the results. For HPCG, 3 important metrics are captured: the GFLOP/s, the bandwidth, and the time. By default, Pavilion2 captures the run log and will only parse information from there. However, using the `files` key inside a regex key will make Pavilion2 look into that file to parse results. While the GFLOP/s, the bandwidth and the time capture numerical data, the result key is used to evaluate the run and provide a pass/fail verdict.

```
result_parse:
  regex:
    gflops:
      files: 'HPCG-Benchmark*'
      regex: 'GFLOP/s\srating\sof=(\d+.\d+)'
    bandwidth:
      files: 'HPCG-Benchmark*'
      regex: 'Raw\sTotal\sB/W=(\d+.\d+)'
    time:
      files: 'HPCG-Benchmark*'
      regex: 'Time\sSummary::Total=(\d+.\d+)'
    result:
      action: store_true
      files: 'HPCG-Benchmark*'
      regex: 'HPCG\sresult\sis\sVALID'
```

2) Mode file: The second sub-step within the Pavilion2 setup is the mode file. A mode file provides a way to override something that is defined in a test configuration. The test configuration written in the previous sub-step works fine, however, it only runs on 1 node once. It's great for verifying the test functionality, however, the test needs to run on all available nodes with the quality of service set to high.

```
# Iterate over all the chunk ids
permute_on: sched.chunk_ids
# List of chunk ids
chunk: '{{sched.chunk_ids}}'
schedule:
  # Nodes from the 'up' (available + idle) state
  node_state: 'up'
  # Use all of the 'up' nodes
  nodes: 'all'
  # Chunking size: 1 node per chunk
  chunking:
    size: 1
  # Quality of Service set to high
  qos: high
```

The mode file uses pavilion-provided system variables to permute over all available nodes and run 1 test per node. The choice of 1 node runs allows the execution of tests without causing a backlog of slurm jobs due to requested nodes being unavailable. Although the permutations and chunking can be written inside a test configuration, since it has to be applied to all the tests, it is easier to write a mode file which can be applied to all the tests.

3) Series file: To simplify the Pavilion run command, a series file is written that groups the tests and automatically applies the mode file to all the tests listed.

```
test_sets:
  basic:
    tests:
      - "gromacs-water.base"
      - "hpcg.base"
      - "mem-info.base"
      - "module-timing"
      - "perl-perf"
      - "stream.base"
      - "vpic.gnu"
      - "vpic.intel"
      - "vpic.shasta"
      - "vpic.shasta-small"
    modes: ['continuous']

simultaneous: 100 # Run 100 tests at a given time.
repeat: 1 # Run this series only once.
```

The series file makes the continuous testing framework modular. The tests can easily be added and removed from the series file without any modification to other files.

B. Cron Daemon

The Cron Daemon runs commands at specific times. [4] Although Pavilion2 series can be run with just one command, the Continuous Framework needs to do more, such as cancellation of old series, rescheduling new series, and logging of the weekly execution. Instead of writing multiple crontab entries that performed the tasks, a script was written to complete the tasks. The script would source the activation script, cancel the previously run series and reschedule a new one while logging the standard out to a log file.

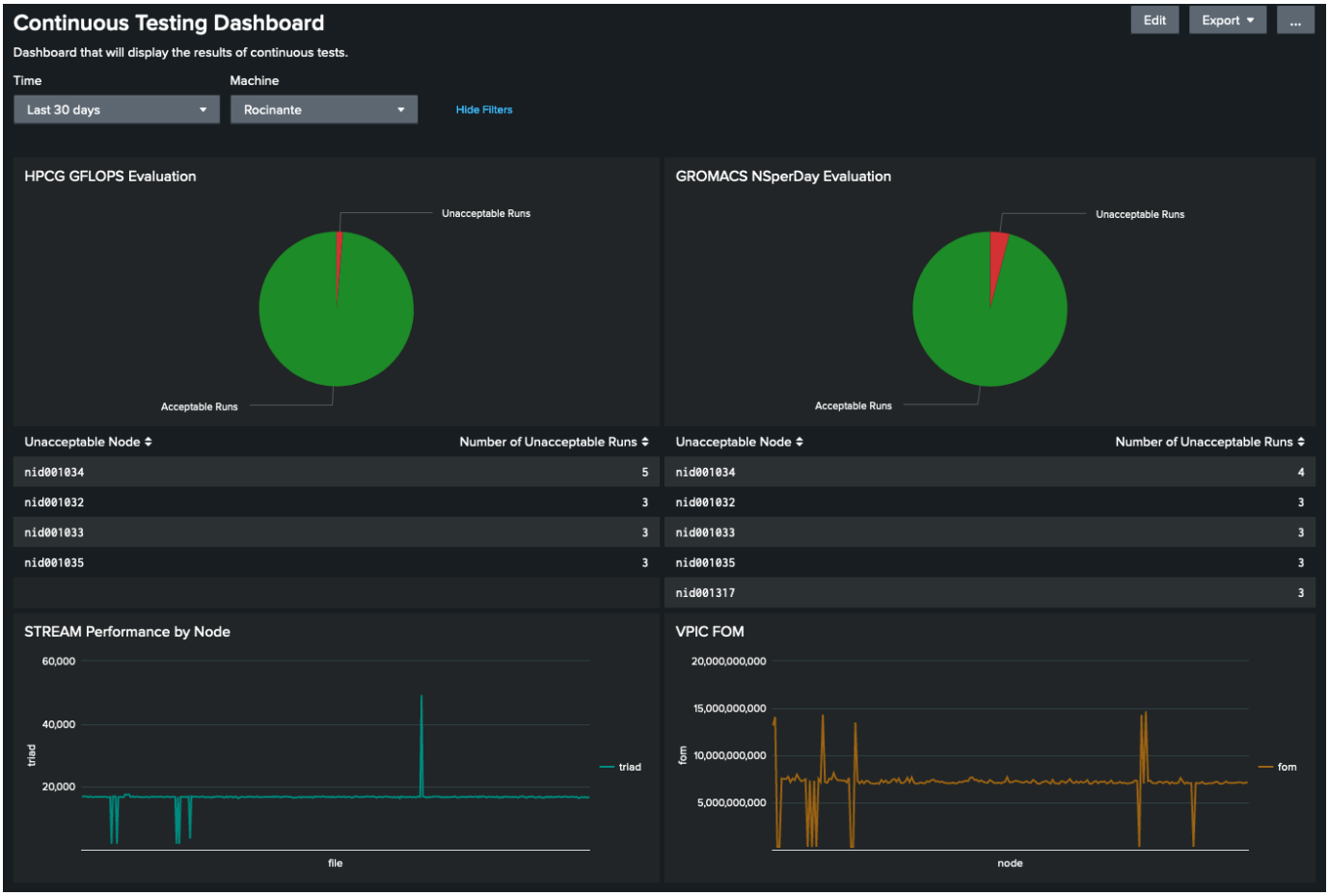

Fig. 1. Continuous Testing Splunk dashboard summarizing Rocinante supercomputer's test results from the last 30 days.

A crontab entry is then added to run this script every Saturday at midnight. The reasoning behind scheduling jobs at midnight on Saturday is due to one of the requirements for the Continuous Testing framework: The testing should not hinder scientists' work. During weekdays scientists are actively allocating nodes to run and test their simulation. However, prior to leaving for the weekends, scientists would schedule long running jobs and won't check in until Monday. This would be the perfect time to run the Continuous Testing framework since it would schedule and run tests as soon as a node becomes available following a scientist's job completion.

C. Splunk

Following test completion, Pavilion2's parsed results are automatically stored in a json log file. The result log file can be continuously monitored by a Splunk forwarder that sends results immediately to an indexer for transformation of data into searchable events. [5] Using Splunk's Search Processing Language (SPL), the PRETeam can query the data for a particular test event and evaluate the results. [6]

By creating a threshold for the performance metrics, a "passing" criteria can be generated and saved as a report. The reports are then used to create a Splunk dashboard which allows for system evaluation in a concise manner. [7] Each panel on the dashboard represents the performance of a different test as shown in Fig. 1.

IV. Results & Conclusions

The Continuous Testing framework has been implemented in all of LANL's Shasta machines: Chicoma, Razorback, Rocinante, Tycho, and Crossroads. The series automatically runs on the weekend and processes the results for evaluation. The Splunk dashboard provides a simple way to view the summarized results without having to manually parse through each test's output. The additions to the infrastructure that allow the tests to run automatically simplify the steps to find issues when they arise without much interference from the PRETeam.

While the Continuous Testing project has only just been implemented the past few months as of April 2024, the modularity of the framework allows the PRETeam to easily add additional tests and features as the users' needs change. Since the implementation of this project, the PRETeam has received and completed requests from code teams and system administrators to integrate several benchmarks that they have identified to be necessary.

In addition, because of the Continuous Testing project, the PRETeam has been able to reduce the list of tests run during DSTs to a few simple sanity checks. As a result,

this has shortened the amount of time the systems need to be unavailable to the users. On average, the DST testing now takes 30-45 minutes depending on the number of nodes on a particular system compared to 2-3 hours prior to the implementation. The results collected from the memory tests has also been helpful in determining the threshold for the Node Health Check to reboot low memory nodes after job completion. This prevents interruptions to scientists' work due to out-of-memory issues.

V. Future Work

The Continuous Testing framework is an on-going project. In the future, the PRETeam would like to add more tests, curating the list to maintain robustness. For example, some network tests are initially omitted as they would need to run on multiple nodes. As the list of tests grows, ensuring that the entire list of tests completes on every node before the next iteration starts the following week becomes a challenge and a bottleneck. Initially, all of the continuous tests would complete within a few days. However, with the increase in the number of tests, large machines like Crossroads and Tycho are sometimes unable to complete the entire set of tests within a week. There are several open issues in the Pavilion2 repository that will help mitigate a few issues, including this one.

Due to the novelty of these Shasta systems, the PRETeam will eventually need to determine a good baseline to set as a pass/fail threshold for the performance tests since the current thresholds generate a lot of false positives. Data is currently being collected to determine proper thresholds for these tests.

VI. Acknowledgements

Special thanks to the Advanced Simulation and Computing program, the PRETeam, Ben Santos, Anna Pietarila Graham, Jennifer Green, Jim Williams and Conor Robinson, without their support and expertise, this project would not have been possible.

References

[1] Brian Keenan. Welcome to Crossroads, Aug 2023.
[2] AI Integration User. Rolling Restart and What It Implies, Oct 2023.
[3] Paul Ferrell. Pavilion2, Oct 2022.
[4] Marcela Maslano Paul Vixie. cron(8) - linux man page.
[5] Splunk Enterprise - Getting Data In, Sep 2022.
[6] Splunk Enterprise - Search Manual, May 2021.
[7] Splunk Cloud Platform - Search Tutorial, Mar 2024.