
Los Alamos
NATIONAL LABORATORY
EST. 1943

# An Approach to Continuous Testing

Shivam Mehta, Paul Ferrell,
PRETeam, Adam Good, and Francine Lapid
Los Alamos National Laboratory (LANL)

smehta@lanl.gov

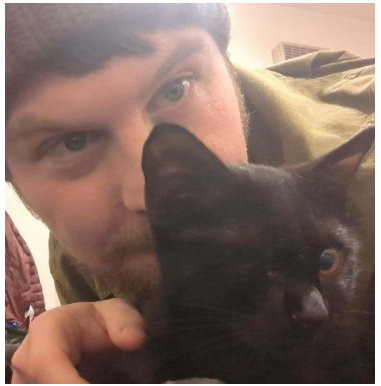

May 8, 2024
Cray User Group 2024, Perth, WA, Australia

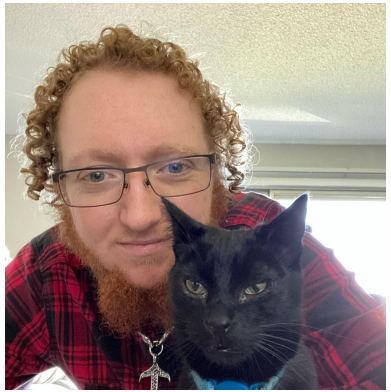
# The PRETeam: Members


Chris Dejager


David Debonis


Paul Ferrell


Timothy (Ty) Goetsch


Adam Good


Francine Lapid


Shivam Mehta


Hank Wikle

# The PRETeam: Description

- Programming Runtime and Environments Team (PRETeam) at Los Alamos National Laboratory (LANL).
  - Software environments.
  - Container support.
  - Assorted hardware and software expertise.
  - System testing.

"We don't know how anything works but we have to debug it anyway."

Adam’s Cat, Frey, diagnosing network problems.

- Problem Statement
  - Story Time
  - System Changes
  - System Monitoring
- Continuous Testing Framework
  - Pavilion Overview
  - Continuous Testing Implementation
  - Cron Job
  - Splunk Dashboard
- Conclusion
- Future Work

# Before I begin

- DST
  - Dedicated System Time
    - Time allocated for system maintenance, upgrades, and testing.
  - Synonyms:
    - Scheduled downtime.
    - Maintenance time.
    - Planned outage.

Adam's cat, Freya, turning off for maintenance.

# Why?

# A horror story in 1 slide

- Your cluster needed to be rebooted
- No changes have been made since the last scheduled downtime
- You're running post-maintenance tests to verify functionality
- Everything is running fine

Calvin's cat, Gorlock, anxiously waiting for clusters to reboot

# Except…

# The MPI Speed Tests Are Failing

Node Health Check? Fine ✅

Image? Same ✅

Hardware? Looks Good

Network? Healthy ✅

# So you start wondering

When did this actually start?

Did the machine degrade between reboots?

Was there a system change between reboots?

# The World May Never Know!

# A horror story in 1 slide

- Your cluster needed to be rebooted
- No changes have been made since the last scheduled downtime
- You're running post-downtime to verify functionality
- Everything is running fine…except…
  - Your MPI performance test is failing!
- You can't find a cause
- You can't figure out when this started happening

(Based on a true story)

# A Few More Reasons

# System Changes

- We want to move away from scheduled downtimes for changes
  - It's costly to take down the system
- Rolling strategy by sysadmins.
- Problem:
  - In the past, post-maintenance testing ensured changes did not negatively affect the system.
  - Without post-maintenance testing, we can't verify the system post change

Francine's cat, Killua, all tuckered out after a long day of not working.

# System Monitoring

- Adds performance metrics to a preexisting system monitoring setup.
- Gives us a solid flow of data to analyze for future business decisions.

Gorlock, looking at all the data she has to analyze.

# Continuous Testing Framework

# Continuous Testing Framework: In a Nutshell

Cron Job


Pavilion2


Splunk

https://www.flaticon.com/free-icon/calendar_2738173

# Test Requirements

- Tests must take less than 10 minutes to complete.
- Tests should evaluate a particular system component.
- Tests should attempt to simulate workload.

Ty's roommate's cat, Harold, defeated by all the test requirements.

# Current Tests

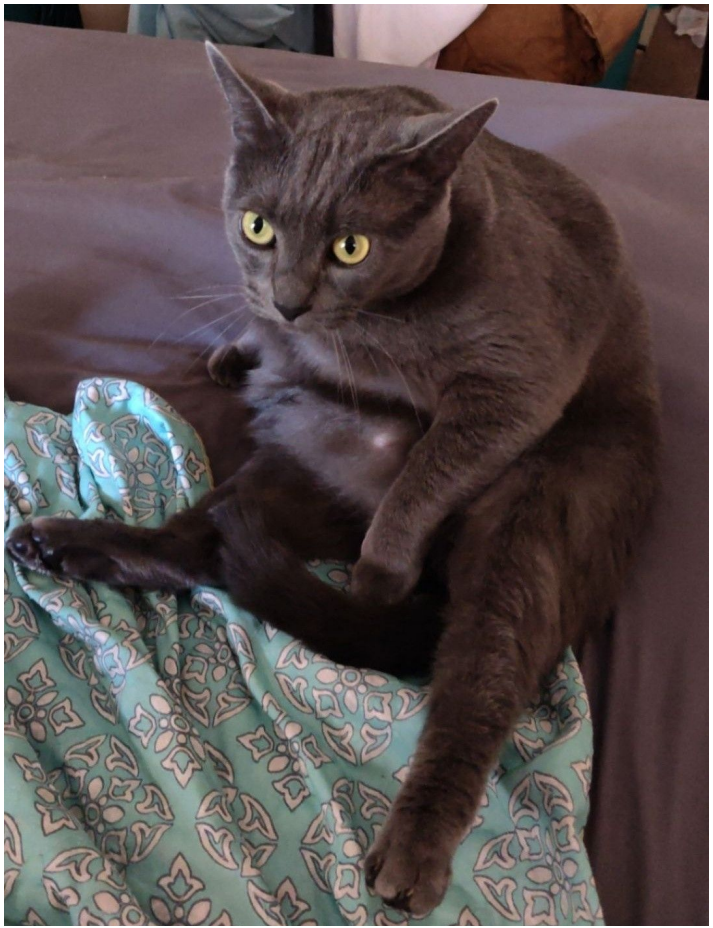
| Test Name | System Component | Description |
|---|---|---|
| Flexible I/O Tester | Filesystems | Different I/O workload simulations |
| GROMACS - Water Benchmark | CPU & GPU | Water molecule simulation |
| HPCG | CPU + MPI | Representative of modern workloads |
| Jacobi | GPU | Distributed jacobi solver |
| Mem-Info | Memory | Memory metadata |
| Module-Timing* | Environment Modules | LMOD response time |
| Perl-Perf* | CPU | CPU performance test |
| Stream | Memory | Memory bandwidth test |
| VPIC | Memory | Particle-in-cell node stress test |

* User provided

# Pavilion

# Continuous Testing: Pavilion

1. Test configs.
2. Series file.
3. Mode file.



Jake's cat, Percy, scrutinizing config files.

# Example Test Configuration - DGEMM (Base Definition)

dgemm.yaml (base)

```yaml
base:
    subtitle: 'DGEMM-OMP_{{ompnumthreads}}_{{size}}'
    summary: DGEMM problem from the APEX benchmark suite
    maintainer:
        name: Paul Ferrell
        email: pferrell@lanl.gov
    scheduler: slurm
    variables:
        size: [ 2500, 5000, 10000, 20000, 40000 ] #12665
        ompnumthreads:
            - '{{ sched.min_ppn // 2 }}'
        lib_path: '/opt/cray/pe/libsci/20.06.1.1/CRAY/9.0/x86_64/lib'
        build_opts:
            - '-DUSE_CBLAS=1 -Ofast -mcpu=native -fopenmp
-L{{lib_path}} -lsci_cray_mp'

    permute_on: [ompnumthreads, size]
```

# Example Test Configuration - DGEMM (Base Definition)

dgemm.yaml (base)

```yaml
base:
    subtitle: 'DGEMM-OMP_{{ompnumthreads}}_{{size}}'
    summary: DGEMM problem from the APEX benchmark suite
    maintainer:
        name: Paul Ferrell
        email: pferrell@lanl.gov
    scheduler: slurm
    variables:
        size: [ 2500, 5000, 10000, 20000, 40000 ] #12665
        ompnumthreads:
            - '{{ sched.min_ppn // 2 }}'
        lib_path: '/opt/cray/pe/libsci/20.06.1.1/CRAY/9.0/x86_64/lib'
        build_opts:
            - '-DUSE_CBLAS=1 -Ofast -mcpu=native -fopenmp
-L{{lib_path}} -lsci_cray_mp'

    permute_on: [ompnumthreads, size]
```

PAVILION HPC TEST HARNESS

EXPRESS TEST METADATA

HIGH PERFORMANCE COMPUTING

# Example Test Configuration - DGEMM (Base Definition)

dgemm.yaml (base)

```yaml
base:
    subtitle: 'DGEMM-OMP_{{ompnumthreads}}_{{size}}'
    summary: DGEMM problem from the APEX benchmark suite
    maintainer:
        name: Paul Ferrell
        email: pferrell@lanl.gov
    scheduler: slurm
    variables:
        size: [ 2500, 5000, 10000, 20000, 40000 ] #12665
        ompnumthreads:
            - '{{ sched.min_ppn // 2 }}'
        lib_path: '/opt/cray/pe/libsci/20.06.1.1/CRAY/9.0/x86_64/lib'
        build_opts:
            - '-DUSE_CBLAS=1 -Ofast -mcpu=native -fopenmp
-L{{lib_path}} -lsci_cray_mp'

    permute_on: [ompnumthreads, size]
```

PAVILION HPC TEST HARNESS

DEFINE VARIABLES TO BE REUSED IN THE HIERARCHY OF THE TEST CONFIG

HIGH PERFORMANCE COMPUTING

# Example Test Configuration - DGEMM (Base Definition)

dgemm.yaml (base)

```yaml
base:
    subtitle: 'DGEMM-OMP_{{ompnumthreads}}_{{size}}'
    summary: DGEMM problem from the APEX benchmark suite
    maintainer:
        name: Paul Ferrell
        email: pferrell@lanl.gov
    scheduler: slurm
    variables:
        size: [ 2500, 5000, 10000, 20000, 40000 ] #12665
        ompnumthreads:
            - '{{ sched.min_ppn // 2 }}'
        lib_path: '/opt/cray/pe/libsci/20.06.1.1/CRAY/9.0/x86_64/lib'
        build_opts:
            - '-DUSE_CBLAS=1 -Ofast -mcpu=native -fopenmp
-L{{lib_path}} -lsci_cray_mp'

    permute_on: [ompnumthreads, size]
```


IDENTIFY VARIABLES TO ITERATE OVER

# Example Test Configuration - DGEMM (Build Definition)

dgemm.yaml (build)

```yaml
    build:
        source_url: http://portal.nersc.gov/project/m888/apex/mt-
dgemm_160114.tgz
        source_path: mt-dgemm_160114.tgz
        extra_files: dgemm_omp_fixes.patch
        modules: [ '{{compilers}}', '{{mpis}}' ]
        cmds:
            - "# Patch broken openmp pragmas"
            - patch -p1 < dgemm_omp_fixes.patch
            - '${PAV_CC} -o mt-dgemm {{build_opts}} mt-dgemm.c'
```

# Example Test Configuration - DGEMM (Build Definition)

dgemm.yaml (build)

```yaml
    build:
        source_url: http://portal.nersc.gov/project/m888/apex/mt-
dgemm_160114.tgz
        source_path: mt-dgemm_160114.tgz
        extra_files: dgemm_omp_fixes.patch
        modules: [ '{{compilers}}', '{{mpis}}' ]
        cmds:
            - "# Patch broken openmp pragmas"
            - patch -p1 < dgemm_omp_fixes.patch
            - '${PAV_CC} -o mt-dgemm {{build_opts}} mt-dgemm.c'
```

PAVILION HPC TEST HARNESS

DEFINE SOURCE AND ANY EXTRA FILES

HIGH PERFORMANCE COMPUTING

# Example Test Configuration - DGEMM (Build Definition)

dgemm.yaml (build)

```yaml
    build:
        source_url: http://portal.nersc.gov/project/m888/apex/mt-
dgemm_160114.tgz
        source_path: mt-dgemm_160114.tgz
        extra_files: dgemm_omp_fixes.patch
        modules: [ '{{compilers}}', '{{mpis}}' ]
        cmds:
            - "# Patch broken openmp pragmas"
            - patch -p1 < dgemm_omp_fixes.patch
            - '${PAV_CC} -o mt-dgemm {{build_opts}} mt-dgemm.c'
```

DEFINE ENV MODULES REQUIRED AT BUILD TIME

# Example Test Configuration - DGEMM (Build Definition)

dgemm.yaml (build)

```yaml
    build:
        source_url: http://portal.nersc.gov/project/m888/apex/mt-
dgemm_160114.tgz
        source_path: mt-dgemm_160114.tgz
        extra_files: dgemm_omp_fixes.patch
        modules: [ '{{compilers}}', '{{mpis}}' ]
        cmds:
            - "# Patch broken openmp pragmas"
            - patch -p1 < dgemm_omp_fixes.patch
            - '${PAV_CC} -o mt-dgemm {{build_opts}} mt-dgemm.c'
```

EXPRESS BUILD SHELL COMMANDS

# Example Test Configuration - DGEMM (Run Definition)

dgemm.yaml (run)

```
    run:
        timeout: 6000
        modules: [ '{{compilers}}', '{{mpis}}' ]
        env:
            OMP_NUM_THREADS: '{{ompnumthreads}}'
            OMP_PROC_BIND: 'true'
            OMP_DISPLAY_ENV: 'true'
        cmds:
            - for node in {{ sched.test_node_list }}; do
            -   if [[ $node == $SLURMD_NODENAME ]]; then
            -       continue
            -   fi
            -   srun -w $node -N1 -n1 ./mt-dgemm {{size}} &>
${node}.out &
            - done
            - while [[ $(jobs | wc -l) != 0 ]]; do
            -   jobs
            -   sleep 1
            -   echo -n "."
            - done
            - ./mt-dgemm {{size}} &> ${SLURMD_NODENAME}.out
```

# Example Test Configuration - DGEMM (Run Definition)

dgemm.yaml (run)

```yaml
run:
    timeout: 6000
    modules: [ '{{compilers}}', '{{mpis}}' ]
    env:
        OMP_NUM_THREADS: '{{ompnumthreads}}'
        OMP_PROC_BIND: 'true'
        OMP_DISPLAY_ENV: 'true'
    cmds:
        - for node in {{ sched.test_node_list }}; do
        -   if [[ $node == $SLURMD_NODENAME ]]; then
        -       continue
        -   fi
        -   srun -w $node -N1 -n1 ./mt-dgemm {{size}} &>
${node}.out &
        - done
        - while [[ $(jobs | wc -l) != 0 ]]; do
        -   jobs
        -   sleep 1
        -   echo -n "."
        - done
        - ./mt-dgemm {{size}} &> ${SLURMD_NODENAME}.out
```

RUNTIME ENV CONFIGS


# Example Test Configuration - DGEMM (Run Definition)

dgemm.yaml (run)

```yaml
run:
    timeout: 6000
    modules: [ '{{compilers}}', '{{mpis}}' ]
    env:
        OMP_NUM_THREADS: '{{ompnumthreads}}'
        OMP_PROC_BIND: 'true'
        OMP_DISPLAY_ENV: 'true'
    cmds:
        - for node in {{ sched.test_node_list }}; do
        -   if [[ $node == $SLURMD_NODENAME ]]; then
        -       continue
        -   fi
        -   srun -w $node -N1 -n1 ./mt-dgemm {{size}} &>
${node}.out &
        - done
        - while [[ $(jobs | wc -l) != 0 ]]; do
        -   jobs
        -   sleep 1
        -   echo -n "."
        - done
        - ./mt-dgemm {{size}} &> ${SLURMD_NODENAME}.out
```

BASH SHELL EXECUTION

# Example Test Configuration - DGEMM (Results Parsing Definition)

dgemm.yaml (results)

```
    result_parse:
#=============================================================
#Final Sum is:          12665.033333
# -> Solution check PASSED successfully.
#Memory for Matrices:   3671.315575 MB
#Multiply time:         31.769945 seconds
#FLOPs computed:        121899274911000.000000
#GFLOP/s rate:          3836.936920 GF/s
#=============================================================
        constant:
            size:
                const: "{{size}}"
            omp_num_threads:
                const: "{{ompnumthreads}}"
        regex:
            result:
                regex: 'Solution check PASSED successfully'
                per_file: all
                files: '*.out'
            memory:
                regex: '^Memory for Matrices:\s+(.*)\s+MB'
                per_file: name
                files: '*.out'
            gflops:
                regex: '^GFLOP\/s\s+rate:\s+(.*)\s+GF\/s'
                per_file: name
                files: '*.out'

    result_evaluate:
        gflops_avg: avg(per_file.*.gflops)
        gflops_outliers: outliers(per_file.*.gflops, keys(per_file),
2)
```

# Example Test Configuration - DGEMM (Results Parsing Definition)

dgemm.yaml (results)

```
    result_parse:
#=============================================================
#Final Sum is:          12665.033333
# -> Solution check PASSED successfully.
#Memory for Matrices:   3671.315575 MB
#Multiply time:         31.769945 seconds
#FLOPs computed:        121899274911000.000000
#GFLOP/s rate:          3836.936920 GF/s
#=============================================================
        constant:
            size:
                const: "{{size}}"
            omp_num_threads:
                const: "{{ompnumthreads}}"
        regex:
            result:
                regex: 'Solution check PASSED successfully'
                per_file: all
                files: '*.out'
            memory:
                regex: '^Memory for Matrices:\s+(.*)\s+MB'
                per_file: name
                files: '*.out'
            gflops:
                regex: '^GFLOP\/s\s+rate:\s+(.*)\s+GF\/s'
                per_file: name
                files: '*.out'

    result_evaluate:
        gflops_avg: avg(per_file.*.gflops)
        gflops_outliers: outliers(per_file.*.gflops, keys(per_file),
2)
```

CAPTURE UNIQUE KEYS

# Example Test Configuration - DGEMM (Results Parsing Definition)

dgemm.yaml (results)

```
    result_parse:
#=============================================================
#Final Sum is:          12665.033333
# -> Solution check PASSED successfully.
#Memory for Matrices:   3671.315575 MB
#Multiply time:         31.769945 seconds
#FLOPs computed:        121899274911000.000000
#GFLOP/s rate:          3836.936920 GF/s
#=============================================================
        constant:
            size:
                const: "{{size}}"
            omp_num_threads:
                const: "{{ompnumthreads}}"
        regex:
            result:
                regex: 'Solution check PASSED successfully'
                per_file: all
                files: '*.out'
            memory:
                regex: '^Memory for Matrices:\s+(.*)\s+MB'
                per_file: name
                files: '*.out'
            gflops:
                regex: '^GFLOP\/s\s+rate:\s+(.*)\s+GF\/s'
                per_file: name
                files: '*.out'

    result_evaluate:
        gflops_avg: avg(per_file.*.gflops)
        gflops_outliers: outliers(per_file.*.gflops, keys(per_file),
2)
```

PASS/FAIL RESULTS

# Example Test Configuration - DGEMM (Results Parsing Definition)

dgemm.yaml (results)

```
    result_parse:
#=============================================================
#Final Sum is:          12665.033333
# -> Solution check PASSED successfully.
#Memory for Matrices:   3671.315575 MB
#Multiply time:         31.769945 seconds
#FLOPs computed:        121899274911000.000000
#GFLOP/s rate:          3836.936920 GF/s
#=============================================================
        constant:
            size:
                const: "{{size}}"
            omp_num_threads:
                const: "{{ompnumthreads}}"
        regex:
            result:
                regex: 'Solution check PASSED successfully'
                per_file: all
                files: '*.out'
            memory:
                regex: '^Memory for Matrices:\s+(.*)\s+MB'
                per_file: name
                files: '*.out'
            gflops:
                regex: '^GFLOP\/s\s+rate:\s+(.*)\s+GF\/s'
                per_file: name
                files: '*.out'

    result_evaluate:
        gflops_avg: avg(per_file.*.gflops)
        gflops_outliers: outliers(per_file.*.gflops, keys(per_file),
2)
```

PERFORMANCE CAPTURE

# Example Test Configuration - DGEMM (Results Parsing Definition)

dgemm.yaml (results)

```yaml
    result_parse:
#=============================================================
#Final Sum is:          12665.033333
# -> Solution check PASSED successfully.
#Memory for Matrices:   3671.315575 MB
#Multiply time:         31.769945 seconds
#FLOPs computed:        121899274911000.000000
#GFLOP/s rate:          3836.936920 GF/s
#=============================================================
        constant:
            size:
                const: "{{size}}"
            omp_num_threads:
                const: "{{ompnumthreads}}"
        regex:
            result:
                regex: 'Solution check PASSED successfully'
                per_file: all
                files: '*.out'
            memory:
                regex: '^Memory for Matrices:\s+(.*)\s+MB'
                per_file: name
                files: '*.out'
            gflops:
                regex: '^GFLOP\/s\s+rate:\s+(.*)\s+GF\/s'
                per_file: name
                files: '*.out'

    result_evaluate:
        gflops_avg: avg(per_file.*.gflops)
        gflops_outliers: outliers(per_file.*.gflops, keys(per_file),
2)
```

RESULTS EVALUATIONS

# Example Test Output

NEAT COMMAND LINE INTERFACE

```
roci
agood@ro-rfe1 hpctools:pav2-lanl ζ pav results s6 -k ~name,mean_open_secs,mean_write_secs,mean_read_
secs,mean_stat_secs
Test Results: s6.
+----+----------------+--------+----------------+-----------------+----------------+---------------+
| Id | Started        | Result | Mean open secs | Mean write secs | Mean read secs | Mean stat secs |
+----+----------------+--------+----------------+-----------------+----------------+---------------+
| 7  | Mar 07 14:05:31 | PASS   | 1.8e-05        | 0.021728        | 1.242366       | 0.001009      |
| 6  | Mar 07 14:04:25 | PASS   | 0.002163       | 0.000343        | 0.050254       | 0.000471      |
agood@ro-rfe1 hpctools:pav2-lanl ζ
```

TEST SERIES METADATA

DATA PARSED FROM RESULTS

!! All Pavilion Results Are Output as JSON !!
(perfect for exporting to visualization software)

HIGH PERFORMANCE COMPUTING

# Continuous Testing: Test Series

vim series/continuous.yaml

continuous.yaml

```
1  test_sets:
2    basic:
3      tests:
4        - "gromacs-water.base"
5        - "hpcg.base"
6        - "mem-info.base"
7        - "module-timing"
8        - "perl-perf"
9        - "stream.base"
10       - "vpic.gnu"
11       - "vpic.intel"
12       - "vpic.shasta"
13       - "vpic.shasta-small"
14     modes: ['continuous']
15
16   simultaneous: 100 # Run 100 tests at a given time.
17   repeat: 1 # Run this series only once.
```

TEST SET DEFINITION

MODE SPECIFICATION

TEST SERIES RUN PARAMETERS

HIGH PERFORMANCE COMPUTING

# Continuous Testing: Mode

vim modes/continuous.yaml

continuous.yaml

```
1  permute_on: sched.chunk_ids
2  chunk: '{{sched.chunk_ids}}'
3  schedule:
4    share_allocation: 'max'
5    node_state: 'up'
6    nodes: 'all'
7    chunking:
8      size: 1
9    account: hpctest
10   qos: high
```

CHUNKING PARAMETERS

ALLOCATION PARAMETERS

PRIORITY PARAMETERS

Shivam’s cat, Koko, patiently waiting for pavilion jobs to start running.

# Cron Job

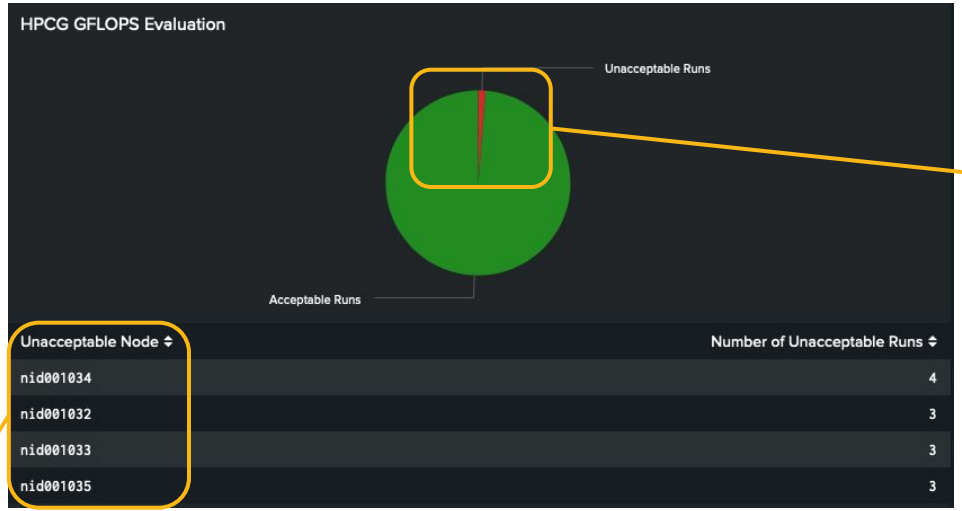
- Runs a script at midnight Saturday.
  - Sources Pavilion.
  - Cancels the old series.
  - Runs a new series.
  - Generates logs.

# Splunk

- Splunk Forwarder forwards json results to the Indexer.
- Searchable events on Splunk Enterprise.
  - Save the searches as a panel on a dashboard.

Boo and Harold, monitoring the splunk panels very intently

# Splunk Dashboard




PROBLEM DETECTED!

POTENTIALLY BAD NODES


GOLDEN NODE?

# Conclusion

# Conclusion

- Post-maintenance testing time reduced from 2-3 hours to 45 minutes.
- Data for analysis.
  - Threshold
  - Procurement
- Users are happy.

> “Just wanted to say how excited I am (and my team) to see ATS# machines back so quickly. Thank you for all your hard work!

# Future Work

# Future Work

1. Refine To Run *More* Continuously Throughout the Week
   - Currently Running in a batch on a weekly basis
   - A more random sample of nodes and kickoff time would be preferable
2. Refine Test Set to Better Model a Cluster via it's Metrics
   - Most of the current test set is essentially tests were available
   - Writing new tests specifically for this use case could get better data
3. Expand the Splunk Dashboard
4. Statistics / Data Science to Gain Better Information on Cluster Health over Time
   - But first we *Need More Data*
5. Optimize Job Queuing to Maximize Data Gained and Minimize User Impact

# Questions?

smehta@lanl.gov
preteam@lanl.gov


Over 70 years at the forefront of supercomputing.