HARMONY: A Harness Monitoring System for the Oak Ridge Leadership Computing Facility

Cameron Kuchta
kuchtact@beloit.edu
Beloit College
Beloit, Wisconsin

Reuben D. Budiardja
reubendb@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Verónica G. Vergara Larrea
vergaravg@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT

Acceptance of a new system requires extensive testing and is often comprised of hundreds of tests. Summit, the latest flagship supercomputer at the Oak Ridge Leadership Computing Facility (OLCF), and the number one system in the November 2018 Top500 list [2], completed its acceptance testing in 2018. To execute acceptance, the acceptance test (AT) team utilizes the OLCF test harness, a tool developed at the OLCF that automates the launch and verification of all acceptance tests. Acceptance requires analysis of test results and classification of all test failures. The sheer number of tests involved makes performing these tasks challenging. To complete these tasks more efficiently, in addition to lessen the personnel burden during acceptance testing, we developed a monitoring system for the OLCF test harness called HARMONY.

KEYWORDS

large-scale system testing, high performance computing

ACM Reference Format:

Cameron Kuchta, Reuben D. Budiardja, and Verónica G. Vergara Larrea. 2019. HARMONY: A Harness Monitoring System for the Oak Ridge Leadership Computing Facility. In *Practice and Experience in Advanced Research Computing (PEARC '19)*, July 28- August 1, 2019, Chicago, IL, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3332186.3332254>

1 INTRODUCTION

The complexities and sophistication of current supercomputing architectures continue to increase. Examples of these complexities can be observed even at a single node scale which may contain multiple accelerators such as GPUs, multi- or many-core processors, and main memory with separate high-bandwidth memory connected to the accelerators. Furthermore, the complexities of the full system are increased by multiple factors including the interconnects within and between compute nodes, the large-scale parallel file system, and the system software that must work at

scale. To ensure that a system of such scale and level of complexity functions correctly and achieves the expected performance, HPC centers commonly execute what is known as an *acceptance test* (AT) before putting the system in production and making it available to users. Additionally, as the software stack matures, it is common practice to execute regression testing on the new software before pushing a major upgrade to the full system.

Acceptance testing of a new system typically involves running a large number of tests, comprising of benchmarking software, science applications, and system software tests, over a period of several weeks. Summit [1], the latest flagship supercomputer at the Oak Ridge Leadership Computing Facility (OLCF), and the number one system in the November 2018 Top500 list, recently completed its acceptance testing in late 2018.

To execute the acceptance testing, the AT team utilizes the OLCF test harness [8], a tool developed at the OLCF that automates the launch and verification of all acceptance tests. The OLCF test harness is written in Python and has also been used for acceptance testing of other supercomputers at the OLCF including Jaguar [3] and Titan [5].

Acceptance testing requires analysis of test results, including the classification and identification of the root cause of all test failures. The sheer number of tests involved makes performing these tasks challenging. To complete these tasks more efficiently, in addition to lessen the personnel burden during an acceptance testing, we developed a new tool to complement the OLCF test harness called HARMONY (*Harness Monitoring System*).

The goal of HARMONY is to facilitate the analysis of results from individual tests by leveraging the same input system of the OLCF test harness, while providing automated monitoring and reporting capabilities. This paper therefore describes HARMONY and is organized as follows. In Section 2 we elaborate the design requirements of HARMONY and how they are implemented. Section 3 show what we have accomplished so far with the current version of HARMONY and how HARMONY was used during Summit's acceptance testing. We end with a summary of our experience in building HARMONY and concluding remarks in Section 4, followed by discussions of the next stages planned for HARMONY in 5.

2 HARMONY'S DESIGN

HARMONY is comprised of three separate modules: monitoring, recording, and reporting. Each of these modules interacts with the OLCF test harness in distinct ways, providing a way to monitor the harness during its execution, recording results from the harness execution, and reporting previously recorded results for further analysis.

Notice of Copyright. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

PEARC '19, July 28-August 1, 2019, Chicago, IL, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Practice and Experience in Advanced Research Computing (PEARC '19)*, July 28-August 1, 2019, Chicago, IL, USA, <https://doi.org/10.1145/3332186.3332254>.

HARMONY is written in Python. It uses MariaDB as its database back end with a Django framework to build its user interface for the recording and reporting modules. HARMONY leverages other publicly available APIs as needed in a modular way. For example, it uses the Slack API [7] to send alerts to the AT team for any event it monitors, e.g. a notification about a test failure via Slack. It also uses the IBM Spectrum LSF API [4] to interface with the LSF scheduler running on the system undergoing acceptance testing. Its modular design allows it to be customizable for other HPC centers that may use a different scheduler and/or messaging service for team communication. The code is open source and publicly available [6].

In this section, we first provide an overview of the OLCF test harness workflow. We then describe each of HARMONY modules in detail.

2.1 OLCF Test Harness

The OLCF test harness [8] is designed to simulate the types of workloads observed on HPC systems. It simulates both code development activities like compiling applications, copying and moving files and directories, as well as batch jobs workloads that execute benchmarks, mini-applications, and applications. The type of workload selected varies by the specific type of test being executed.

In the harness, each application is defined by the `app_name`. Each `app_name` can have multiple tests instances that vary in size, type of problem solved, length of job, etc. Each test instance is defined by the `test_name`. To launch a test, the harness relies on the `rgt_input.txt` input file. Figure 1 shows the structure of the input file. The input file specifies the location where the tests will be executed via the `Path_to_tests` variable, the specific tests to be launched, and the commands for the harness to execute.

```
# Path to the location where tests
# will be executed.
Path_to_tests = /scratch/filesystem/testdir/

# Lists of tests to be executed
# Format used:
# Test = <app_name> <test_name>
Test = app1 test_name1
Test = app2 test_name1
Test = app2 test_name2
...

# Harness commands
# Format used:
# Harness_task = <command>
Harness_task = check_out_tests
Harness_task = start_tests
Harness_task = stop_tests
```

Figure 1: Sample of OLCF test harness input file.

For every `app_name`, `test_name` pair, the harness checks out the source code for `app_name` into a Source subdirectory. Then, for every `test_name`, the harness creates an individual Run_Archive

run directory. For each test instance that is launched, the harness defines a unique test ID and creates a subdirectory within the run directory. In addition, the harness also creates a Status directory for every test to record the status of individual instances as well as a summary of all instances in the `rgt_status.txt` file. Figure 2 shows the directory structure that the harness uses to keep track of the different test instances.

```
# app1 application with 1 test
<PT>/app1/Source
    /test1/Scripts/
        /Run_Archive/12345/buildidir
            /workdir
                /Status/rgt_status.txt

# app2 application with 2 tests
<PT>/app2/Source
    /test1/Scripts/
        /Run_Archive/67890/buildidir
            /workdir
                /67891/buildidir
                    /workdir
                        /Status/rgt_status.txt
    /test2/Scripts/
        /Run_Archive/10112/buildidir
            /workdir
                /Status/rgt_status.txt
```

Figure 2: Example of OLCF test harness directory structure (PT = Path_to_tests).

The `rgt_status.txt` file is key to track the progress of individual test instances. It stores the status of the application build, the job submission for the test instance, as well as the verification script. Figure 3 shows an example of a status file. This file is particularly important as it is used by HARMONY to determine the state of a given test instance.

2.2 Monitoring Tests

HARMONY's monitoring module ensures that test instances launched by the OLCF test harness are progressing in the batch queue and are restarted correctly after a catastrophic failure. The monitoring module can send alerts via two mechanisms: a custom Slack application and via email.

The Slack application allows staff to query status of jobs in the batch queue directly from a Slack channel without having to login to the compute system under test. Slack is also available on mobile devices which gives staff direct access to a snapshot of the batch queue instantaneously via a simple Slack message.

During a test cycle, we need to continuously monitor the system's batch queue to alert the staff when a particular test type is no longer in the queue. To monitor the batch queue, we leverage IBM's Spectrum LSF Python API [4] to connect directly to the LSF scheduler and issue status requests to monitor the entire queue, a specific job, or all jobs by a specific user.

#####					
#Start Time	Unique ID	Batch ID	Build	Submit	Check
#####					
2018-12-18T16:30:37.210080	1545168637.2088807	224932	0	0	0
2018-12-18T18:37:27.305139	1545176247.3040786	224961	0	0	5
2019-02-19T12:15:20.551543	1550596520.5140405	269003	0	0	-1
2019-03-04T12:12:29.692833	1551719549.6678352	285580	0	0	1

Figure 3: Example of OLCF test harness status `rgt_status.txt` file.

Once connected, we peer into the file system and make sure that each test that exists in the `rgt_input.txt` file has an instance currently in the queue. We do this by traversing the path to the instance's directory and reading the `job_id.txt` file which gives us the LSF job ID. Once retrieved, we use the job ID to check if the job is still in the queue. If an instance is present in the queue, we proceed to the next instance. Otherwise, we issue an alert via Slack and email.

The Slack alert is setup using a custom built application that installs a bot in a given Slack channel. This allows the user to send commands to the bot via instant messages. The bot connects to Slack using Slack's client API [7]. A unique key is generated for a particular channel which is used to connect the bot. Once the connection is established, we begin pulling recent messages that have appeared on the selected channel. These messages first pass through a validation process to filter potential erroneous or harmful messages. If the message received is deemed to be valid, we construct a query to generate the correct response and send it back to the user using Slack's client API.

The Slack monitoring module provides the following commands:

- `slack_help`: displays the different commands supported by the bot.
- `all_jobs`: displays the status of the entire queue. The response is equivalent to the output shown by LSF's `bjobs -u all`.
- `check_tests <inputfile>`: provides the status of tests in a specific test harness input file.
- `monitor_job <jobID>`: starts a thread that wakes up periodically to check the status of the the job that matches `jobID`. If the status of the job has changed between checks, the bot issues an alert to the user.
- `my_jobs <username>`: displays the status of all jobs owned by a specific user. The response is equivalent to the output shown by `bjobs -u <username>`.

In addition to the Slack bot, the monitoring module also has the ability to send alerts via email. This mechanism is unidirectional as the user cannot query status on demand but it is provided for convenience.

2.3 Recording Results

The recording module ingests results generated by the test harness into a database. It also automatically updates the database as test instances change state (e.g. move from *queued* to *running*, or move from *running* to *completed*). The database contains all the static

information for a given test instance including the unique test ID, the application name, and the test name. It also stores the individual status from the build script and the output of the build command of a given application, from the job submission of a test, and from the correctness verification of the test results. While this information can be retrieved directly from the file system where results are stored as described in Section 2.1, having results recorded in a database allows the team to have a centralized storage location of all test results rather than distributed over multiple run directories used by the test harness. In addition, the database can be easily exposed to staff to generate custom reports as we have done in HARMONY's reporting module as described in Section 2.4.

Once test instances have finished running, output information is recorded in their respective `Run_Archive` directories (c.f. Figure 2). We devise HARMONY to read in the same `rgt_input.txt` file as the harness, gather status and output of tests instances, and store all information in a database such that the HARMONY database has all the information that the test harness does but in a structured way afforded by our relational database.

The database used to store test information from OLCF resources consists of six tables. The test table contains both static and dynamic information about test instances. The static information includes the unique test ID, the application name, and the test name. The dynamic information includes the status of the build, submit, and verification scripts. The `test_event` table stores a record when a test instance moves from one state to the next (e.g. build completes, job submits, etc.). The `check` and `event` tables list the possible types of status a given test instance can take and the possible states a test instance can be in, respectively. The last two tables, `failure` and `test_failure`, are used to maintain a list of possible types of failures and assign a failure cause to individual test instances that failed.

To update the database, we first gather the `Path_to_tests` path to each test from the input file. We then parse the `rgt_status.txt` file which contains the information about each of the test instances. For each test instance, we must query the test table to determine if a record for this unique test ID already exists. If the test instance has not been previously processed, it is entered in the database using the instance's static information. If the unique test ID has already been recorded, we update the dynamic information to reflect the new state of the test instance.

We update an instance by first collecting all the events that the instance has already completed. We then add these events to the `test_event` table. Then, we get the LSF job ID for the job and query the scheduler to check if there is an exit status for the job. This exit

Figure 4: HARMONY database structure.

status only exists on LSF for one hour so we need to update the database at least every hour in order to get that information from the active log. For each of the possible status fields, we look at the entry in the `rgt_status.txt` that corresponds to this instance and also get the information from the output files for each status. Once all of this information has been collected, we update the instance's entry in the `test` table.

2.4 Reporting Results

HARMONY also includes a Django-based website that provides an interface for its reporting module to filter through tests, allowing the AT team to analyze, describe, and categorize any test failure.

Once a test instance has finished running and has entered the database, we need a simple way to show the results to the test administrator. If a test failed, the administrator needs to be able to assign a failure code to the test and a description of why it occurred. Currently, each instance needs to be manually entered into a spreadsheet and then assigned a failure type.

The website created displays the various instances to the user with multiple filtering capabilities. It also allows the user to look into more detail on a specific instance and add failure types to each with a simple interface.

The interface is written primarily using Django and is hosted on the internal OpenStack server at the OLCF. In addition, it also uses HTML, CSS, and JavaScript.

Pages are constructed in a modular format with one generic template that can be reused for all pages and another sub-template that is used for pages that incorporate lists of test instances. Each separate page has its own HTML file that inherits from the more general templates.

The pages with lists are paginated and have different filters that can be applied to the instances. We have separate pages that contain: all the instances, all instances for a given application, and all instances for a given test.

To add failure descriptions, we added a button that calls a modal form that updates the database. To store this information, we use the `failure` and `test_failure` tables. The `failure` table contains the possible failures that an instance could undergo and a basic

Figure 5: Site map: Each page containing lists of tests can access the page containing tests details. Each page other than the home page also links to the form for creating and deleting failure labels and descriptions.

description of each. The `test_failure` table links a test instance to the failure and includes a more detailed description of the root cause for the specific failure.

3 CASE STUDY: SUMMIT ACCEPTANCE

During acceptance of the Summit system, we used the OLCF test harness to execute tests for a period of several weeks. Throughout the acceptance test period, we monitored the test queue to ensure the system had a sufficient workload using HARMONY's Slack bot through an acceptance specific Slack channel.

The continuous execution of tests resulted in tens of thousands of individual jobs. An early prototype of the HARMONY recording module was used to ingest all the test result data and display it via

Application	Testname	Harness UID	Job ID	Start Time	Status	Failure
ACME	ne120_2days_pgi	1529820666.1015716	103315	June 24, 2018, 2:11 a.m.	255	0 0 +
ACME	ne16_5days_pgi	1529820757.455453	103316	June 24, 2018, 2:12 a.m.	255	0 0 +
ACME	ne30_5days_pgi	1529820810.4976442	103317	June 24, 2018, 2:13 a.m.	255	0 0 +
ACME	ne4_5days_ibm	1529820864.7294726	103326	June 24, 2018, 2:14 a.m.	0	0 0 +
ACME	ne4_5days_ibm	1529763469.0708644	103179	June 23, 2018, 10:17 a.m.	0	0 0 +
ACME	ne4_5days_pgi	1529821384.1503432	103327	June 24, 2018, 2:23 a.m.	255	0 0 +
ACME	ne4_5days_pgi	1529763998.5466886	103181	June 23, 2018, 10:26 a.m.	255	0 0 +
ALCF_MPL_Benchmark	test_aggregate_0002node	1529859622.2334898	103387	June 24, 2018, 1 p.m.	0	0 1 +
ALCF_MPL_Benchmark	test_bisection_0002node	1529859624.2377784	103388	June 24, 2018, 1 p.m.	0	0 1 +

Figure 6: Harmony Django-based GUI

the web interface as shown in Figure 6. The columns in the web interface are designed to match the format from the `rgt_status.txt` file which is familiar to the AT team. To analyze test results and classify test failures, we used HARMONY’s web interface. The Failure column allows the user to select a failure type from the available options. In addition, for any test failure (shown in red), the user can select a test instance and view the corresponding output. For instance, if there is a build failure, the interface displays the output created when the harness launches the specific application build script.

The web interface simplifies the process of classifying failures and gives the user a comprehensive view of the tests’ status. Being able to record results from different test cycles in the same database also allows us to identify improvements (or regressions) in the stack.

HARMONY’s modules were all exercised during Summit’s acceptance. As expected, the recording module was measured to be the most time consuming component of HARMONY. That said, even when over 3,000 job instances were being processed, the update procedure was measured to take less than 30 seconds. Because the dynamic status information is collected directly from the parallel file system where the jobs run, the performance varies and partly depends on the current load of the file system.

4 CONCLUSIONS

In this paper we describe HARMONY as a complementary system for the OLCF test harness. HARMONY is designed to automate test monitoring and simplify test analysis primarily during acceptance testing of a large system. HARMONY can be used in conjunction with

the OLCF test harness to execute and monitor test cycles for distinct purposes and aggregate the results in a centralized database.

HARMONY is written in Python and has a modular design that provides monitoring, recording, and reporting capabilities. Its monitoring capabilities allow the user to monitor the batch queue on an HPC system by receiving notifications via email and by requesting notifications on-demand via Slack instant messages. HARMONY currently supports connections to LSF batch schedulers but can be extended to support alternative schedulers and resource managers.

HARMONY has already proven to be useful during Summit’s acceptance testing. The AT team was able to monitor tests directly from the Slack mobile application. In addition, the test failures could be easily classified from the web interface prototype which reduced the effort required to analyze all the results. Given that this is an early prototype, it was in combination with the OLCF standard test analysis procedures.

The monitoring, recording, and reporting capabilities in HARMONY provide a full picture about the status of a test period at any given point in time.

Because HARMONY is open source, it can be useful for other centers interested in building a similar system. In addition, its modular design allows it to be easily extended to provide additional features.

5 FUTURE WORK

The HARMONY instance used for Summit’s acceptance was deployed as a proof of concept on an internal test server. While all the modules were exercised, we were only able to use the recording module offline given that the tests had already completed. Currently, the recording module flags any instances that it is unable to process. Improvements are on-going to better handle these edge cases.

As the next step, HARMONY will be deployed to a production server on the internal OLCF OpenStack resources. This will allow us to expose the web interface so that it is accessible from outside the ORNL internal network. This will require integration with the OLCF RSA authentication servers and will allow us to assign different roles to individual users.

We also have plans to extend the reporting module to generate specific types of reports that are of interest to management and other stakeholders, as well as download results in CSV format.

ACKNOWLEDGMENTS

We would like to thank Don Maxwell, Jason Kincl, Arnold Tharrington, and Wayne Joubert for their contributions to this project.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This research was supported in part by an appointment to the Oak Ridge National Laboratory Oak Ridge Science Semester Program sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education.

REFERENCES

- [1] [n. d.]. Summit: Scale new heights. Discover new Solutions. <https://www.olcf.ornl.gov/summit/>
- [2] [n. d.]. TOP500 Supercomputer Sites. <https://www.top500.org>.
- [3] Arthur S Bland, Wayne Joubert, Ricky A Kendall, Douglas B Kothe, James H Rogers, and Galen M Shipman. 2010. Jaguar: The world's most powerful computer system—an update. *Cray Users Group* (2010).
- [4] IBM. [n. d.]. IBM Spectrum Computing LSF Python API. <https://github.com/IBMSpectrumComputing/lsf-python-api>.
- [5] Wayne Joubert, Richard K. Archibald, Mark A. Berrill, W. Michael Brown, Markus Eisenbach, Ray Grout, Jeff Larkin, John Levesque, Bronson Messer, Matthew R. Norman, and et al. 2015. Accelerated Application Development: The ORNL Titan Experience. *Computers and Electrical Engineering* 46 (May 2015). <https://doi.org/10.1016/j.compeleceng.2015.04.008>
- [6] OLCF. [n. d.]. Harmony. <https://github.com/olcf/harmony>.
- [7] Slack. [n. d.]. Slack client API. <https://github.com/slackapi/python-slackclient>.
- [8] Arnold N. Tharrington. 2015. NCCS Regression Test Harness, Version 00. <https://www.osti.gov/servlets/purl/1232564>