Regression Testing on Shaheen Cray XC40: Implementation and Lessons Learned

Bilel Hadri[1], Samuel Kortas[1], Robert Fiedler[2], George S. Markomanolis[1]

[1] KAUST Supercomputing Laboratory (KSL)
King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia
email: bilel.hadri@kaust.edu,sa; samuel.kortas@kaust.edu.sa, georgios.markomanolis@kaust.edu.sa

[2] Cray Inc,
Seattle, WA, USA
e-mail:rfiedler@cray.com

***Abstract*—Leadership-class supercomputers are becoming larger and more complex tightly integrated systems consisting of many different hardware components, tens of thousands of processors and memory chips, kilometers of networking cables, large numbers of disks, and hundreds of applications and libraries. To increase scientific productivity and ensure that applications efficiently and effectively exploit a system's full potential, all the components must deliver reliable, stable, and performant service. Therefore, to deliver the best computing environment to our users, system performance assessments are critical, especially after an unplanned downtime or any scheduled maintenance session. This paper describes the design and implementation of the regression testing methodology used on the Shaheen2 XC40 to detect and track issues related to the performance and functionality of compute nodes, storage, network, and programming environment. We also present an analysis of the results over 24 months, along with the lessons learned.**

Keywords: Cray XC40, Regression Testing, Performance, Lesssons learned

I. MOTIVATION

For many years, regression testing has been an essential step of any software development or integration cycle. However, for HPC systems, regression testing is typically performed in a more ad-hoc fashion, and is focused on the basic functionality of the various hardware components. For example, for the previous HPC systems at KAUST, the actual coverage of the tests done after maintenance was not rigorously known, due to the lack of a systematic procedure. Back then, the basic functionality of some system components was checked only before releasing the system back to the users as soon as possible. Under this scenario, the performance of all components was measured only occasionally, and despite some actions taken in response to user complaints regarding functionality and performance issues, tracking down the cause of any observed degradation was challenging.

Since the acceptance of Shaheen 2 [1], a 36-cabinet Cray XC40 supercomputer installed in March 2015, a clear regression procedure has been adopted in order to identify potential hardware or software issues in a more rational and methodical way. After each maintenance session or unscheduled downtime, a set of well-defined tests is systematically run to assess the actual state of the system. A careful analysis of the obtained results is used as crucial input to the decision by the KAUST Supercomputing Lab (KSL) team on whether or not to release the system to the users, based on the criticality of any issues detected.

In the last 24 months, our use of this regression procedure has provided four essential benefits:

1. a drastic decrease of user tickets received soon after a downtime: our objective is to have not a single hardware or software ticket related to the system for the next 24 hours after it is released to users, and we have received only a few in that time frame.
2. a significant gain in performance due to the "trimming" of the nodes, as well fixing weak network links: we observed up to a 10% performance improvement on a full scale code.
3. an improved reproducibility of user experiments run at large scale.
4. a more detailed history of observed hardware and software problems, allowing us to provide more accurate data to vendors about any performance degradation

After presenting the overall regression protocol (section II), unit tests (addressing single-node performance checking and interconnect capacity), and component tests are discussed in section III and IV. Section V focuses on integration and tests: by using actual applications running at medium or full scale, we validate the performance of the system as a whole. Finally, section VI details the first implementation of an automated framework for assembling all of the tests and triggering their execution to enable 'on-the-fly' regression testing of the system.

II. TESTING PROTOCOL

From our experience, a regression testing approach is successful if the tests selected are sufficiently reproducible, and if they provide the most exhaustive coverage possible of the system features that must be tested. In this respect, a systematic testing protocol appeared essential. The set of tests we assembled allowed us to reach a decent coverage with minimum redundancy. In addition, the order in which we execute them has helped guarantee early detection and straightforward localization of any problems observed. Of course, the optimization of this protocol is a never-ending task, and having it carefully documented is of great help. Here we describe our current testing protocol.

First, we test the regular and basic of functionality of the scheduler and programming environments. Tagged as 'Component Tests', they are documented with a clear description of each command to run as well as the expected result. A comprehensive list of these tests is given in section IV. They can easily be automated, and will be included in our automated regression framework in the near future.

Second, we perform extremely well-localized performance runs with synthetic tests validating each crucial component of the system. In detail we test:

- the health and decent performance of any compute node in the system. To do so, we submit a one-node LINPACK test, wrapped into an MPI job to launch it across all nodes and check both performance and accuracy.
- the behavior of the interconnect by evaluating the bandwidth of all links in any allocation of nodes. A Cray-developed topology-aware MPI program is used along with environment variable settings that enforce minimal-path routing.
- the global throughput of the parallel file system (both Lustre and DataWarp nodes) using IOR. The goal is to check the bandwidth of the parallel file systems to be above 500 GBs/ and 1.5TB/s for Lustre and DataWarp respectively.

Last, we run some typical user jobs (real applications for only couple of iterations, such as WRF, SPECFEM, and other in-house codes) to stress the system at larger scales and guarantee good integration of all components (file system, compute nodes, and interconnect) while corroborating the synthetic test results.

Most of the time, this regression testing protocol occurs in a single session after maintenance, but it has also been designed to be run on-the-fly on Shaheen2. Additional implementation details of our regression protocol are given in Section V.

III. Unit Tests Assessing Node and Interconnect Performance

In our regression process, we consider compute nodes, the interconnect, and the file system as the smallest testable parts of our environment. Indeed, during this whole testing process, we never require any system administrator privileges and always remain at a user level. In agreement with the terminology used in software engineering, we therefore named them 'unit tests'.

The purpose of the present section is to describe them further.

A. Node Performance

Shaheen Cray XC40 is composed of 6,174 dual-socket compute nodes based on 16-core Intel Haswell processors running at 2.3GHz. Each node has 128GB of DDR4 memory running at 2300MHz. In order to evaluate individual node performance and computational correctness, we use the binary xlinpack_xeon64, the Intel optimized LINPACK Benchmark for Linux [2] that solves a dense linear system of linear equations (Ax=b) in double precision and measures the required time to factor and solve the system. In the end, this time is converted into a performance rate, and the accuracy of the solution obtained is also validated.

```
Node nid00008
Intel(R) Optimized LINPACK Benchmark dataCurrent
date/time: Wed Mar 22 14:10:11 2017

CPU frequency:    3.599 GHz
Number of CPUs: 2
Number of cores: 32
Number of threads: 32

Parameters are set to:

Number of tests: 1
Number of equations to solve (problem size) : 55000
Leading dimension of array                  : 55000
Number of trials to run                     : 1
Data alignment value (in Kbytes)            : 1

Maximum memory requested that can be used=24201101024, at
the size=55000

========= Timing linear equation system solver ========

Size LDA Align. Time GFlops Residual Residual(norm) Check
55000  55000  1  113.094  980.796  1.7961e-09  2.11745e-02
pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
55000  55000  1       980.7967 980.7967

Residual checks PASSED
```

Figure 1. Intel LINPACK Benchmark output for one node.

The Linux function execl[3] is wrapped into an MPI program that runs separate, identical LINPACK benchmark on each node, and gathers and sorts the results depending on performance. This process also identifies nodes that perform significantly worse (by a specified margin) than the best (or, optionally, the average) node of a given type (i.e., nodes having same core count, clock frequency, memory frequency, etc). The interconnect is used only to determine the node on which each rank is running and to collect results for analysis and outlier identification. Since all nodes are tested concurrently, the overall test run time is determined by the slowest node in the system; it increases only slowly with number of nodes tested due to the longer time required to start up the job and to gather results. In order to have consistent performance results, and to fully stress the CPU and memory of the nodes, we have chosen a matrix size N = 55,000, even though this large problem size takes longer to run. This size tests most of the memory and consistently yields near-asymptotic performance on the Haswell nodes.

On average, the test of node performance lasts around 6 minutes for all Shaheen2 nodes at once.

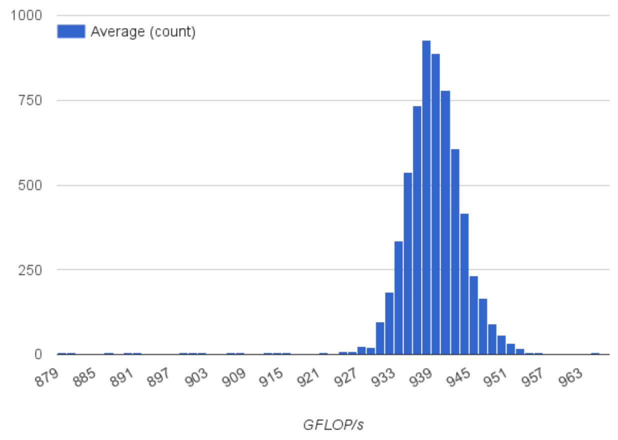

Figure 2. Node performance variability on Shaheen March 2016.

From the output of the benchmark, as shown in Figure 1, we parse the node number, the CPU frequency, the GFLOP/s performance and the residual (accuracy) test. These are the variables that help us detect weak or faulty nodes that need to be fixed. Early in production in July 2015, the node performance variability of Shaheen2 used to range from 930 GFLOP/s to 965 GFLOP/s, with an average performance of 940 GFLOP/s. Over time, it has been noticed that more and more nodes are performing below this range, and some of the nodes reached a poor performance of 879 GFLOP/s as noticed in March 2016 and plotted in Figure 2, with around 100 nodes with a performance lower than 930 GFLOP/s. This is far from the expected performance, and thus the scientists aiming at applications targeting performance would not be able to exploit fully the potential of the Cray XC40. Indeed, the HPL number per node is less than 75% of the theoretical peak of a Shaheen Haswell node (1177.9 GFLOP/s). Consequently, on April 2016, at our request, Cray on-site engineers performed the trimming procedure[4] on all Shaheen nodes, and much improved performance was reached, with an enhancement up to 10%, with a range of performance distribution from 935 to 1025 GFLOP/s with an average of 980 GFLOP/s as shown in Figure 3.

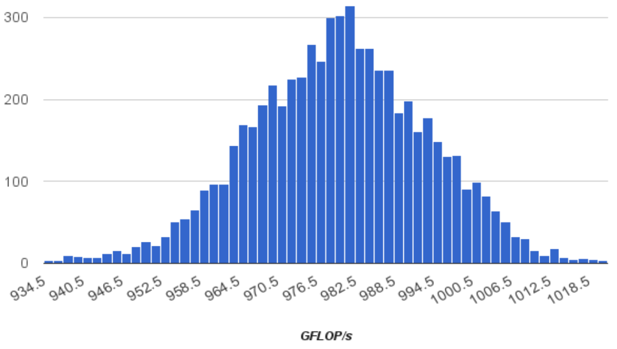

Figure 3. Node performance variability on Shaheen April 2016 after triming.

This critical performance test has helped the KSL team detect major issues during the regression testing:

- **Performance issue**: After each maintenance, a couple of nodes are generally detected with performance lower than 935 GFLOP/s, the threshold set by the KSL team. An individual test is performed, and if the observed performance is confirmed, the node is drained and can be returned to the pool of available nodes after a trimming procedure, provided the performance exceeds 935 GFLOP/s. The weak performance can come from the memory; it has indeed been observed that the memory issue is related to either runs with a performance around and worse than 550 GFLOP/s or when the execution time exceeds the wall clock limit of 10 minutes.
- **Power capping issue**: There was a phase (from July 2015 until December 2016) where Shaheen was running under power and cooling constraints, using initially two static queues (workq_high with uncapped nodes, and workq_low with nodes capped at 275W) and later adopting SLURM dynamic power capping [5]. During this period, we were always able to detect nodes that were not correctly configured. With a performance under 800 GFLOP/s, this correspond to a node that is behaving as capped, while it belongs to the set of uncapped nodes. These issues lead to updates on CAPMC[6] and SLURM since randomly some nodes were set to a lower frequency (below 2.3GHz, while the test should show around 3.6 GHz as shown in Figure 1).
- **CPU frequency issue**: More recently, with the release of SLURM 17.02, a critical issue has been detected thanks to this test, where at the beginning of each batch step, the command srun would inadvertently set the CPU frequency maximum to the minimum value supported on the node. The result obtained by the node performance test showed only one node that was capped at 1.2GHz, while the rest of nodes reached expected performance. Nevertheless, when testing several nodes individually, all of them reported a low performance with a CPU frequency set at 1.2GHz. This problem has been detected with the applications, like SPECFEM and WRF, where only one weak node increased considerably the time to solution of the application.
- **Correctness**: Thanks to the residual check in the LINPACK test, it sometimes occurs that the performance of a given node is in the acceptable range, however the residual is above the threshold, which means that the answer is incorrect. This typically corresponds to a faulty socket with inaccurate results that will impact dramatically any scientific results. Since

production, 12 sockets have been detected as faulty and sent back to Intel for further analysis by Cray on-site engineers.

- **Thermal issue**: examining the performance data stored so far, the overall performance of the nodes is quite stable and does not vary. Nevertheless, we observe, during the iteration of tests of the same node to validate the results, variation from 910 back to 990 GFLOP/s and again down to 910. This issue is typically linked with a thermal issue and the detected node is reported and further tests are run by Cray on-site engineers to determine the faulty socket needing to be replaced. These sockets impact application performance reproducibility.

This process has an excellent advantage to provide to particular users the list of the best nodes (above 1 TFLOP/s) for their performance study.

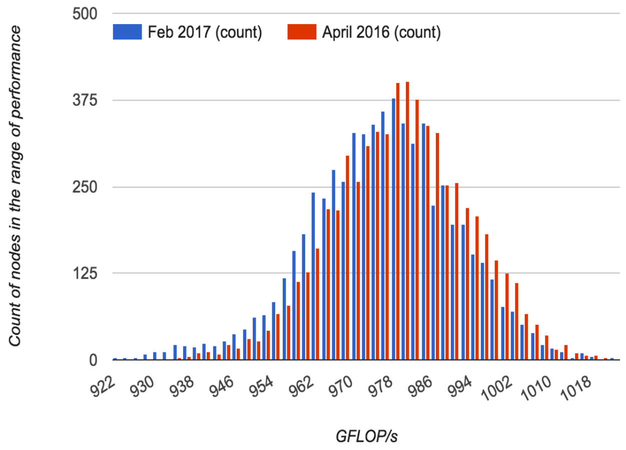

Figure 4. Node performance variability on Shaheen comparison between April 2016 and February 2017.

This test is done almost monthly following a maintenance and all the data are stored. This allows us to observe the performance variability and degradation over time as shown in Figure 4 comparing the result in April 2016 and February 2017. Even tough the average of all nodes is quite stable (only 0.4% of variation, from 980 to 976 GFLOP/s), we clearly observe a shift of the majority of nodes towards a lower value, toward the left side for the February 2017 node performance in blue as plotted in Figure 4. This is also shown in Figure 5, where the performance of each node is plotted. February 2017 performance is lower than April 2016, and on the left side of the plot, we spot several hundred of nodes that lost close to 8% of their original performance from as high as 1015 GFLOP/s down to 940 GFLOP/s.

Consequently, the threshold limit is lowered to 930 GFLOP/s, since further trimming process on the node with a performance of 931 to 935 GFLOP/s does not fix the issue. It seems that this is expected normal degradation according to Intel engineers.

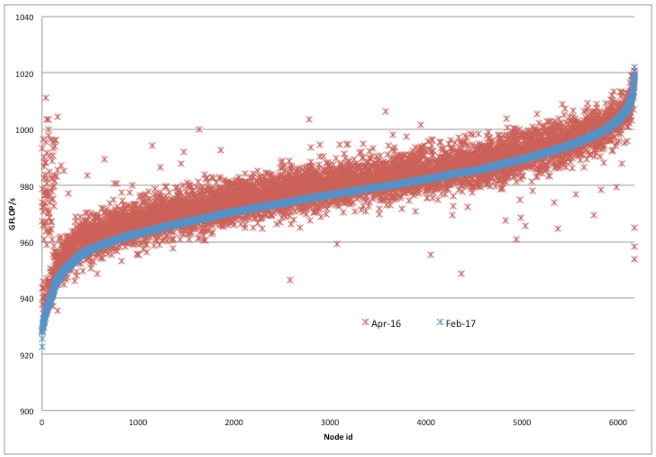

Figure 5. Node performance variability on Shaheen comparison between April 2016 and February 2017.

B. Link Performance Test

The HSN on Shaheen is configured with 8 optical network connections between every pair of cabinets achieving therefore 57% of the maximum global bandwidth between the 18 groups of two cabinets. This will allow the design of a future upgrade with additional cabinets to accommodate more optical links between all cabinets with the same level of connectivity, i.e. 8 optical network connections between every pair of cabinets.

In order to evaluate individual interconnect link performance, we employ the test_links tool developed at Cray. Test_links was originally developed for the Gemini network on Cray XE/XK systems [2], and was recently redesigned for Cray XC systems (aries interconnect). Cray has provided to KSL the test_links binary executable and batch scripts for the purposes of troubleshooting and regression testing.

Test_links evaluates the bandwidth of all interconnect links in any allocation of nodes and identifies links with lower (by a specified amount) than the best or the average bandwidth for links of the same type. Bandwidths for each link are also recorded in tables for comparison between sets of results obtained at different times, so that one may determine whether and how individual link performance has changed over time. Four different types of links are evaluated, including:

- the PCIe links from the four compute nodes on a blade to the single Aries router on that blade,
- the copper links between blades in the same chassis,
- the copper links between blades in different chassis of the same group,
- the optical links between groups.

The test_links code is a user-level topology-aware MPI program in which the physical location on the hardware of each process (MPI rank) in the allocation is obtained from Cray PMI library calls. The bandwidth of any link is determined by timing MPI calls involving messages of a specified size using an intensive communication pattern between ranks located on either end of that link. Certain environment variables are set to disable adaptive routing, i.e., the only communication paths allowed are the minimal (most direct) paths. Thus, all communication in the selected pattern passes only through the link being evaluated (plus the PCIe links from the aries routers to the nodes). Note that the performance impact of a small number of links delivering somewhat less than the nominal bandwidth (as indicated by test_links) on a real science application running with adaptive routing enabled can be expected to be considerably less than the discrepancy reported by test_links, since the adaptive routing algorithm can compensate for the presence slow links by utilizing alternate paths.

Links are evaluated independently but concurrently, in order to minimize the time it takes to measure the bandwidth of every link in the allocation. The test algorithm is designed to handle the presence of any service nodes and down compute nodes in the system. The run time to test all PCIe and copper (i.e., intra-group) links in a system is essentially independent of the number of groups in the system. The run time for N groups is proportional to N, rather than the number of optical links, which scales as N*(N-1). Thus, all link bandwidths for even a very large system can be measured in ~10 wall clock minutes or less.

When evaluating the optical links between groups, multiple blades on either end of a given link are utilized. As a result, only the aggregate bandwidth of a given optical link can be measured. Nevertheless, the test can readily identify any slow optical links, which can be further diagnosed by the system administrators using lower-level Cray-provided tools.

The implemented communication patterns include all-to-all and one-to-one, in which each core on one end of the link sends a message to the corresponding core on the other end using non-blocking point-to-point MPI Isend/Irecv calls. The one-to-one pattern exhibits higher bandwidths than all-to-all when evaluating link types at all levels above the PCIe links, and is therefore considered to be the more sensitive of the two for detecting performance degradation. We use all-to-all for the PCIe links, and typically observe wider variations in performance among PCIe links in the system compared to variations measured for the other non-optical link types. We normally use a message size of 2 MB and repeat the communication pattern thousands of times to ensure accurate timings. The threshold used for identifying a slow link is typically 10% below the average measured bandwidth for a given link type. Lane degrades for non-PCIE, non-optical link type typically reduces the measured bandwidth by 20-30% (with adaptive routing disabled), and therefore is readily detected using a 10% threshold.

Table 1 summarizes the expected performance. Over time, this test revealed several links in dimension 2 and 3 that needed to be replaced, or in some cases only a reboot of the nodes in the implicated blade was necessary to resolve the issue.

	Description	Expected Performance
Dimension 1	Optical links between groups	60 GB/s
Dimension 2	Copper links between different chassis	8.5 GB/s
Dimension 3	Backplane within a chassis	3.5 GB/s
Dimension 4	PCIe connections from nodes to aries router	5 GB/s

Table 1: Expected Performance of the different dimensions

In case of poor performance, misbehaving nodes or links are either fixed or removed from the available resource, and the test is re-executed to make sure that the remaining links are performing as expected.

This test lasts around 6 minutes for evaluating all Shaheen links. Figure 6 shares few lines of the test_links report, focusing on the chassis-chassis links where node nid01775 is reporting the lowest bandwidth. In this example, the network link associated with the low bandwidth node is indeed degraded and further troubleshooting was done to resolve the problem. Moreover, once a node is indicated as a significant outlier, its corresponding blade is disabled and another iteration of the link test is performed to confirm that the disabled blade was the cause of the issue.

```
ANALYSIS OF RESULTS FOR ARIES DIMENSION  2

 For aries with            3  available compute nodes:
 Highest  bandwidth for   3 nodes      8.19175
 Lowest   bandwidth for   3 nodes      8.10847
 Average  bandwidth for   3 nodes      8.15648
 Std. dev.     for   3 nodes     0.241659E-01

 For aries with            4  available compute nodes:
 Highest  bandwidth for   4 nodes      8.55541
 Lowest   bandwidth for   4 nodes      4.34258
 Average  bandwidth for   4 nodes      8.51178
 Std. dev.     for   4 nodes     0.772598E-01


OUTLIER(MORE THAN 5.0% BELOW AVERAGE)FOR ARIES DIMENSION 2
NID   tx ty tz  NID   tx ty tz BdwidthGB/s nodes % deviation
1644, 4,1,11,  1775,  4,3,11,   4.34258      4        48.98
1708, 4,2,11,  1775,  4,3,11,   6.59278      4        22.55
1775, 4,3,11,  1644,  4,1,11,   4.34258      4        48.98
1775, 4,3,11,  1708,  4,2,11    6.59278      4        22.55
```

Figure 6. Extract of test_link for Dimension 2 links.

C. IOR Tests

The primary Shaheen data storage solution is a Lustre Parallel file system with a usable storage capacity of 17.2 PB delivering around 500 GB/s of I/O throughput. The Cray Sonexion 2000 installation is configured using 72 Scalable Storage Units (SSU) and 144 Object Storage Servers (OSS) connected to the XC40 via 72 LNET router service nodes.

ShaheenII has 268 Cray DataWarp (DW) accelerator nodes hosting a total of 536 Intel SSD cards. This combination provides an aggregate burst buffer capacity of 1.56 PB to Shaheen users. This fast middle storage layer provides up to three times the performance of the Lustre parallel file system. Indeed, it achieves 1.54 TB/s and 1.66 TB/s in IOR write and IOR read, respectively.

For both storage systems, the IOR benchmark[8] is used for measuring the amount of data moved in a fixed time. In our case, we fix it at 60 seconds, respectively on 1152 and

5628 compute nodes on Lustre and Burst Buffer. Figure 7 shows the history of IOR performance on the Lustre parallel file system. The performance was over the expected performance of 500GB/s until September 2016 and kept decreasing down to 433 GB/s, which corresponds to a degradation of more than 20%. This was a major issue, but it is not critical enough to prohibit releasing user jobs. Nevertheless, the regression testing history of the IOR benchmark showed a correlation with the WRF application benchmark with close to a 10% increase in execution time. After diagnostic analysis, the performance decrease was determined to be mainly due to increased data usage on Lustre and fragmentation of OST usage. Indeed, 44% of the Lustre capacity was used in May 2016, and it reached close to 70% of capacity in December 2016. A variation on the load on individual OSTs of up to 20% was observed, and some of them were at close to 85 % of capacity.

IO throughput performance over 500 GB/s was achieved again in January 2017, with a decrease of usage (to around 50%) and a defragmentation was performed automatically, since backup and archiving files are enabled with a Cray Tiered Adaptive Storage (TAS) system, which consists of a tape library with a total capacity of 20 PB upgradable.

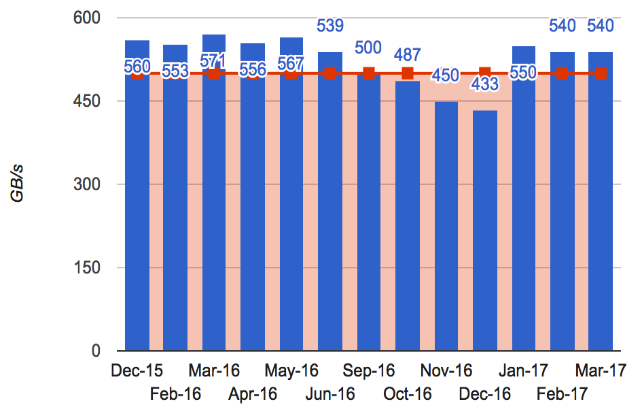

Figure 7. IOR Benchmark results on Shaheen.

IV. Component Tests

In parallel with the unit tests described in the previous section, a set of basic features can also be tested validating individually the key components of the environment. By analogy with the software engineering classical approach, we call them 'Component Tests'.

Each of these tests entails launching a simple Unix command that has been well-documented in a checklist as well as a means to verify whether or not the command was successful.

The following Table 2 gives an idea of the components tested:

Category	Purpose	How to test?
General	Connection	Try to login via ssh (do this test with each login node)
	promptness of command line	How long for a regular shell command to return?
	check X-Windows	Does an X11 window open correctly when spawned from Shaheen front-end?
	check files	Are files accessible in /home, /lustre, /project/, /scratch?
Licenses	Cray compiler	Can we compile a toy program with these compilers?
	Intel compiler	
	Commercial software	Can we run Totalview, DDT, Ansys?
Scheduler	Availability	Check that all queues are up and running and record the number of nodes down
	Nominal use	Submit (1, 4-512, 510-1000, > 1000) -node jobs Submit from /project, from /scratch
	Stress	Measure the time needed to submit a job-array of 500 jobs. When running, cancel all of them.
	Scheduling policies	It should not be possible to submit more than 800 jobs per user, more than 512 nodes occupied with jobs of 72 hours.
	Accounting	Check if the accounting is working
Programming Environment	Compilers	Compile a toy code with Cray, Intel and GNU compiler
	Libraries, modules	Link toy codes against petsc, perftools, hdf5 and netcdf libraries
	Monitoring	Check that the previous compilations have been recorded in the xalt database. Check that a toy program's IO behavior is tracked in Darshan.
Burst Buffer	Availability	Submit a job using the burst-buffer and check the queue status

Table 2: Expected Performance of various component tests

V. Integration and Performance Tests Using Actual Applications

Last but not least, at the end of our testing protocol, here are some procedures to test the whole environment. Following a software engineering analogy, they are named 'Integration tests'.

They all involve running typical use cases using actual applications on actual data sets representative of the Shaheen workload. All tests have in common the process of compiling and running through the scheduler, but they stress diverse components of the environment (I/O, compute power in memory use, and network). Out of the ten application tests available we usually pick 4 or 5 of them to confirm the overall stability and availability of the whole environment.

We here give two examples of these application benchmarks.

A. SPECFEM3D

Context: SPECFEM3D simulates 3D seismic wave propagation in any region of the Earth based on the spectral-element method**.** This test consists of running a reference case delivered with the source of the Fortran 90 MPI code. It uses a CMT solution for a deep earthquake (647.1 km depth) which took place in Bolivia in June, 1994 and uses the CPU version 5.1.5 of SpecFEM3D_GLOBE.

This example creates a 3D earth global mesh spectral element mesh using model S263ANI and runs 8700 time steps using an explicit integration scheme. All implementation details of this test can be found at [9].

Test Family: MPI multi-node application, medium scale

Components tested: Execution environment sustainability, interconnect and compute node overall performance, scheduler behavior.

Purpose: The executable had been built for the acceptance test of Shaheen 2 in 2015 and was not recompiled since to insure not only tracking of the performance of compute nodes and interconnect on a medium scale but it also validates the compatibility of the running environment against an executable built several months ago.

Description: Two runs can be performed on 2336 cores (73 nodes) or 16224 cores (507 nodes). Following the guidance at [6], we systematically check the accuracy of obtained results and track the global elapsed time of the whole simulation (excluding the time spent building the mesh).

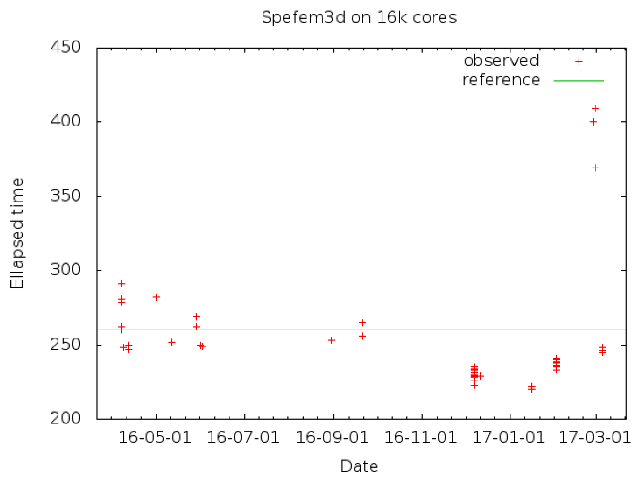

Figure 8. SPECFEM3D regression testing result over time.

Results: As shown in Figure 8, SPECFEM3D 16224-core test has been run systematically after each maintenance session. The elapsed time observed during Shaheen 2 acceptance testing was 260 seconds.

A significant improvement in the execution environment has been observed since December 2016. This corresponds to when SLURM dynamic capping was disabled. Even if the code was not recompiled, a time below 240 seconds is now observed as a nominal result.

We can also notice a classical pattern of degraded times observed at the beginning of a regression step, helping us to confirm that a problem needs to be fixed, and the nominal result obtained again once the problem has been solved.

The log run times (400 and 369 s) observed at the end of February 2017 were related to an issue with the latest SLURM 17.02 version installed. As described in section III, the CPU frequency of one node in the same job was different, leading to a performance close to the lowest capped node. This problem was immediately identified thanks to this test and was solved later by SchedMD.

B. WRF

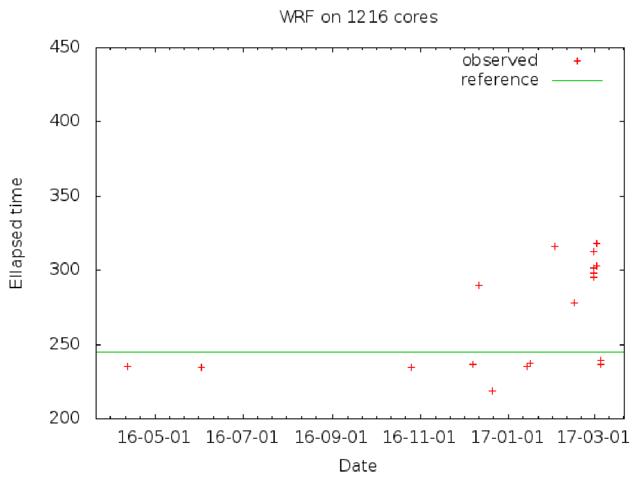

Figure 9. WRF regression testing results over time.

Context: Weather Research and Forecasting (WRF) [9] is used by many environmental groups and weather centers to simulate, forecast the weather but also study weather phenomena. For this case we use WRF version, 3.5.1.

Test Family: MPI multi-node application, medium scale

Components tested: Execution environment sustainability, interconnect and compute node overall performance, scheduler behavior.

Purpose: The WRF version is the same with the acceptance test of Shaheen 2 in 2015 and same flags were used for the compilation to be able to compare the performance obtained during the time of acceptance.

Description: We use 304 nodes, we have 1216 MPI processes and 8 OpenMP threads per node. The domain is small because we do not want to spend a lot of time on regression testing. We have 3-nested domains; the large domain is 701x601 data points. Compared to SPECFEM3D, this application stresses the network more and the compute nodes less, and perfoms a small amount of I/O. Sometimes WRF can identify errors with high speed network (this has happened on two or more occasions). We use a triply-nested domain covering a large range outside Saudi Arabia, and the last nested domain extends close to the red sea.

Results:

As shown in Figure 9, WRF has been run systematically after each maintenance session.

The elapsed time observed during Shaheen 2 acceptance testing was 245 seconds. In the figure, we see a variation of the execution time of the WRF case, depending on compute node or network connectivity issues.

VI. AUTOMATED REGRESSION FRAMEWORK

In [1], a brief overview of the regression framework was given. The following paragraphs provide more details about its implementation. As a medium-term objective, we plan to release the complete sources of this monitoring and testing environment as well as the tests not involving confidential or commercial components to the HPC community.

A. A three-component monitoring environment

Since the first day of Shaheen 2 production, a monitoring environment has been put in place. It relies on three components hosted on a unique workstation connected via SSH to a regular user account:

1. **A Jenkins integration server** that allows the continuous monitoring of the Shaheen 2 SLURM scheduling environment. Every 3 minutes, a shell script saves in a log file:
   - a comprehensive state of the scheduling queue for compute nodes and allocated burst buffer spaces (result of squeue -l and dwstat --all command)
   - general information about the resources available: nodes made available to the queue (sinfo) or taken out (sinfo -R), list of the ongoing reservations (scontrol show reservation), priority of the jobs to be submitted (sprio) and general diagnoses about the system (sdiag).

2. **A Python extracting script** that computes the current load on Shaheen 2 from Jenkins log files, and stores this information in a MYSQL table as tuples (timestamps, load, Jenkins_job). The load is the ratio of Shaheen nodes hosting a running job to the total number of nodes available. Jenkins_job refers to the exact Jenkins log from which information was computed.

3. **A PHP/Jquery based website** allowing easy browsing over time of the load history of Shaheen 2 (see figure 10). The load of the system, taken every 3 minutes, is plotted over time, and clickinig on a location of the obtained curve leads to the opening of another window displaying the corresponding Jenkins log at that time in the history. Via a clickable arrow, one can also go back in time and browse a window from 1 h to one week at any specific date in the last 6 months.

B. A self-described testing framework

To complement these monitoring tools, KTF, (KAUST Testing framework), written in Python allows one to describe, store, run, monitor and collect the results of a given test in a very straightforward way [11].

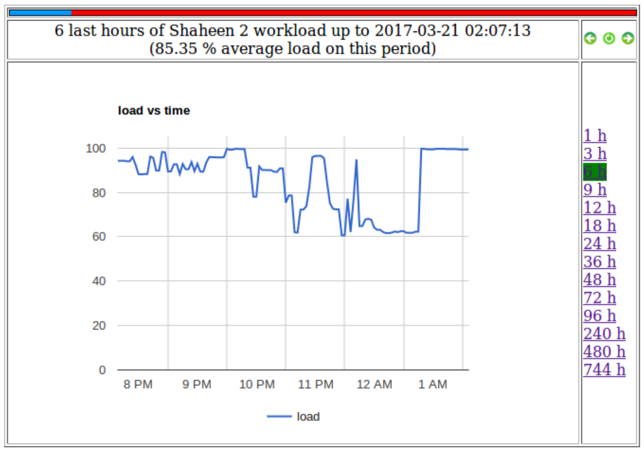
6 last hours of Shaheen 2 workload up to 2017-03-21 02:07:13
(85.35 % average load on this period)

Figure 10. Shaheen2 load monitoring website.

A test is saved as a Python class (inherited from the ktf class) where the user defines which command to run and how to extract result information from the output produced.

This result is saved into a database and examined to detect any regression of the environment.

The detail of the execution of the test is defined through a Python function, or by providing a set of sub-directories containing the executable, a job template, and all the input files required. Optionally, the Python script or job template contains parameters to be set at the test launching step: for example, the number of nodes or cores or any parameter to pass to the code (size of mesh, convergence threshold …). These files (Python scripts and/or template files) are stored in a global directory gathering all the available regression tests. In order to assure tracking and reproducibility, this directory is placed under version control with GIT.

For each run carefully indexed by a unique identifier, the test result is saved in a database as well as the precise time stamps when the test began and stopped, the GIT checksum of the source actually used, the measure of the elapsed time, the list of nodes scheduled, and all information about the environment that are systematically dumped in an output file copied in a Text field.

In case of unsatisfying observed performance, all these data are very useful in order to correlate the obtained result with the state of execution environment at this very same time. Via a combined examination of these data and Jenkins log files, one can see if another particular code was running simultaneously or if the scheduler was detecting problems on some nodes partly allocated to the job, etc.

In addition, the properties of every possible different test are characterized in a single table. There, apart from a unique identifier, the following items are specified:

[1] the list of components stressed during this test (compute nodes, interconnect, Luster file system, Burst Buffer, Scheduler, Compiling or Running environment),

[2] the classification of the test: short, medium or large scale, sequential or MPI executable, load dependent or not, etc.

[3] the possible combinations of resources and parameters on which that test can be run. For example if a test can be run on 1, 16 and 64 nodes, on a small, medium or large size problem, this field could be [(1, small), (1, medium), (16, small), (16, medium), (64, medium), (64, large)]

[4] the pool of resources on which the tests can be run. For the LINPACK and SPECFEM3D tests, it consists of a set composed of all available nodes. For the test_links code, it may be a set of all the point-to-point connections in the interconnect. A complete regression cycle is completed when all the resource components appearing in the pool have been used at least one time by a regression test.

D. *The on-the-fly regression process triggered by the observed load of Shaheen2*

Based on the information gathered, processed and stored in the monitoring components and the testing framework, the regression tests can then be launched automatically.

- After each run of the Jenkins script gathering the scheduling environment of Shaheen, a threshold test on the current load is performed and triggers a regression test if Shaheen is under 75% utilized for the last 1 hour.
- When this trigger is activated, a Python script scans the tests available in KTF database and decides which one to submit to the scheduler. Priority is given to test mobilizing resources from the available pool that have been unsolicited so far. In the example of the LINPACK test described at section III, this program tests the performance of each node executing LINPACK. By taking the intersection of nodes currently idle and nodes that have not been tested yet, the script gathers the subset of candidate compute nodes on which to run LINPACK next.
- At the end of each job, results are gathered along with information about the environment. These are made available in the result database where a consolidated view of the current fraction of the resource pool already tested is also available.

As this approach is still under testing, the database schemes described here are not definitive, but they give a rough idea of what can be achieved automatically. The script computing the next test to schedule is a function attached to each test encoded in the KTF framework.

VII. Conclusion

In this paper, we present the immense benefit of following a regression testing protocol after every maintenance or unscheduled downtime on an HPC environment. By assembling a coherent set of tests and applying them systematically, it helped us to detect sooner various problems in the hardware or software environment and allowed the release of a more reliable and performant environment to the users.

With this protocol, the KSL team has successfully provided a clean environment with near-zero ticket issue received related to software and hardware within 24 hours following a maintenance. This protocol takes on average one hour and 30 minutes, and does not require any special system privileges. Indeed, the tests are performed by the computational scientist team members as regular users.

Our rigorous approach for measuring key aspects of system performance lead to a gain of up to 10% in application performance along with better reproducibility of user experiments on Shaheen.

Though some of these tests have already been included in the Cray testing suite and adopted by Cray on-site engineers, our goal was to give an exhaustive view of the whole protocol applied. An automated version of this process is currently under testing, enabling 'on-the-fly' performance evaluation and even earlier detection of potential issues. A first version of the components of this framework is planned to be released to the community as soon as it is stable and documented.

Acknowledgment

For computer time, this research used the resources of the Supercomputing Laboratory at King Abdullah University of Science & Technology (KAUST) in Thuwal, Saudi Arabia

References

[1] B Hadri, S Kortas, S Feki, R Khurram, G Newby , "Overview of the KAUST's Cray X40 System–Shaheen II," In proceeding of Cray User Group 2015, Chicago, 2015

[2] Intel Optimized LINPACK Benchmark for Linux: https://software.intel.com/en-us/node/528615

[3] https://linux.die.net/man/3/execl

[4] Jeff Brooks, "Shifts in the Marketplace And the Exascale Era HPC SAUDI 2017 Conference

http://www.hpcsaudi.org/graphics/uploads/plenary/day2/5.%20KAUST%20Exascale%20Talk%20for%20PDF-%20Brooks.pdf

[5] ScheMD: Power Management Guide (power capping) - Slurm Workload Manager https://slurm.schedmd.com/power_mgmt.html

[6] S. J. Martin, D. Rush, and M. Kappel. Cray advanced platform monitoring and control (CAPMC). In Proc. Cray User Group Conference (CUG) , 2015

[7] C. Mendez, G. Bauer, W. Kramer, and R. Fiedler, "Expanding Blue Waters with Improved Acceleration Capability", In Proceedings of the Cray User Group 2014, CUG 2014, Lugano, 2014

[8] IOR benchmark: http://www.csm.ornl.gov/essc/io/IOR-2.10.1.ornl.13/USER_GUIDE

[9] SPEFEM3D Globe Tutorial 1: global simulation: https://wiki.geodynamics.org/software:specfem3d_globe:start

[10] Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, M. G Duda, X.-Y. Huang, W. Wang, and J. G. Powers, 2008: A Description of the Advanced Research WRF Version 3. NCAR Tech. NoteNCAR/TN-475+STR, 113pp. doi:10.5065/D68S4MVH

[11] Samuel Kortas, “KTF (KAUST Testing Framework) presentation at the workshop 'Boost your efficiency when dealing with multiple jobs on the Cray XC40 Supercomputer Shaheen II', June 2016, KAUST, https://www.hpc.kaust.edu.sa/sites/default/files/files/public/many_jobs.pdf