A Step Towards the Final Frontier: Lessons Learned from Acceptance Testing of the First HPE/Cray EX 3000 System at ORNL

Verónica G. Vergara Larrea, Reuben D. Budiardja, Paul Peltz, Jeffery Niles, Christopher Zimmer,
Daniel Dietz, Christopher Fuson, Hong Liu, Paul Newman, James Simmons, Christopher Muzyn
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN, USA
Email: vergaravg@ornl.gov

***Abstract*—In this paper, we summarize the deployment of the Air Force Weather (AFW) HPC11 system at Oak Ridge National Laboratory (ORNL) including the process followed to successfully complete acceptance testing of the system. HPC11 is the first HPE/Cray EX 3000 system that has been successfully released to its user community in a federal facility. HPC11 consists of two identical 800-node supercomputers, Fawbush and Miller, with access to two independent and identical Lustre parallel file systems. HPC11 is equipped with Slingshot 10 interconnect technology and relies on the HPE Performance Cluster Manager (HPCM) software for system configuration. ORNL has a clearly defined acceptance testing process used to ensure that every new system deployed can provide the necessary capabilities to support user workloads. We worked closely with HPE and AFW to develop a set of tests that used the United Kingdom's Meteorological Office's Unified Model (UM) and 4DVAR. We also included benchmarks and applications from the Oak Ridge Leadership Computing Facility (OLCF) portfolio to fully exercise the HPE/Cray programming environment and evaluate the functionality and performance of the system. Acceptance testing of HPC11 required parallel execution of each element on Fawbush and Miller. In addition, careful coordination was needed to ensure successful acceptance of the newly deployed Lustre file systems alongside the compute resources. In this work, we present test results from specific system components and provide an overview of the issues identified, challenges encountered, and the lessons learned along the way.**

***Keywords*-acceptance testing; system test;**

I. Introduction

In this paper, we summarize the deployment of the Air Force Weather (AFW) HPC11 system at Oak Ridge National Laboratory (ORNL) including the process followed to successfully complete acceptance testing of the system.

To our knowledge, HPC11 is the first HPE/Cray EX 3000 system that has been successfully released to its user community. The HPC11 system consists of two identical HPE/Cray EX 3000 supercomputers, Fawbush and Miller. HPC11 is the first deployed system at ORNL equipped with Slingshot 10 interconnect technology and also the first relying on the HPE Performance Cluster Manager (HPCM) software for system configuration. Each compute resource consists of 800 nodes each with two 64-core AMD Rome CPUs and 256GB of memory. In addition, HPC11 has access to two independent and identical high performance Lustre parallel file systems.

As described in [1], ORNL has established a clearly defined acceptance testing process that has been used to successfully deploy its last three flagship systems: Jaguar, Titan, and Summit. The acceptance testing process includes four different test elements which are used to ensure that every new system deployed can provide the necessary capabilities to support its user workloads. The four elements include a vendor test, a functionality test, a performance test, and a stability test. A key requirement of ORNL's acceptance test is to be able to simulate a realistic workload. We achieved this goal by selecting applications that are representative of the target user community's workflows. Our team worked closely with HPE and AFW subject matter experts to develop a set of tests that used United Kingdom's Meteorological Office's Unified Model (UM) and 4DVAR. We also included additional benchmarks and applications to fully exercise the HPE/Cray programming environment. We relied on standard benchmark and test suites to evaluate the functionality and performance of both the hardware and the software stack. We used the High-Performance Linpack (HPL) and STREAM benchmarks during the vendor test element. We also utilized the OpenMP Validation & Verification suite [2], the Standard Performance Corporation High Performance Group (SPEC HPG) suites[3], and the OSU Microbenchmarks [4] to evaluate the OpenMP and MPI implementations available on the system during the functionality test element. We complemented these with additional applications from the Oak Ridge Leadership Computing Facility (OLCF) portfolio including LSMS [5], GenASiS [6], minisweep [7], among others.

Acceptance testing of HPC11 required parallel independent execution of each acceptance test element on Fawbush and Miller. In addition, careful coordination was needed to ensure successful acceptance of the newly deployed Lustre file systems alongside the compute resources.

In this work, we present test results from specific system components including: network, file system, system

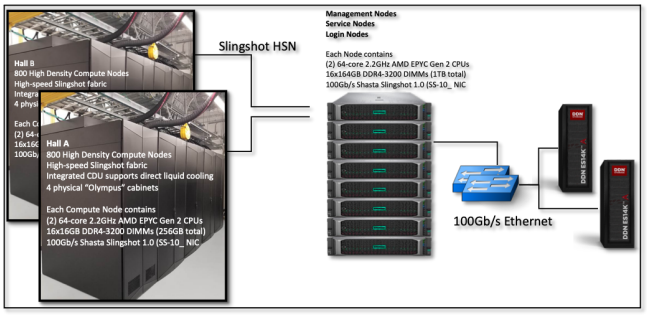

Figure 1. High Level Diagram of HPC11.

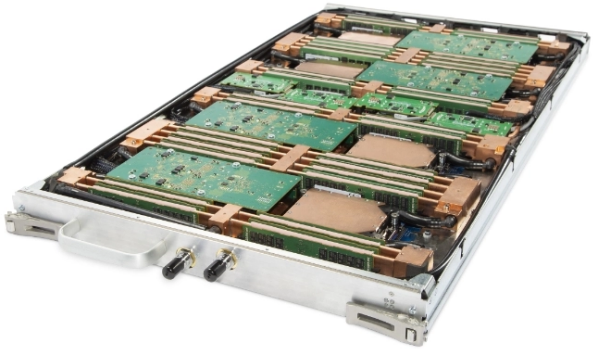

Figure 2. Node Design of HPC11.

management, programming environment, and tools. We also provide an overview of the issues identified, challenges encountered, and the lessons learned along the way that would be useful for other sites currently working on similar system deployments.

II. System Architecture

HPC11 consists of two identical, independent HPE/Cray EX 3000 supercomputers, Fawbush and Miller, with access to two identical, independent Lustre parallel file systems, Storm and Cyclone.

A. Compute System

Each compute resource consists of 800 compute nodes each with two 64-core AMD Rome CPUs and 256 GB of memory [8]. The compute nodes are connected via Slingshot 10 interconnect technology configured in a Dragonfly topology. Figure 1 shows the HPC11 high level diagram. Figure 2 shows the HPC11 node design of an AMD Epyc Processor.

1) System Configuration: HPC11 leverages the HPE Performance Cluster Manager (HPCM) for system configuration. HPCM is a fully featured cluster management suite that is responsible for the life cycle management of a HPC system. The software provides tools for switch management, image curation and provisioning, monitoring and health

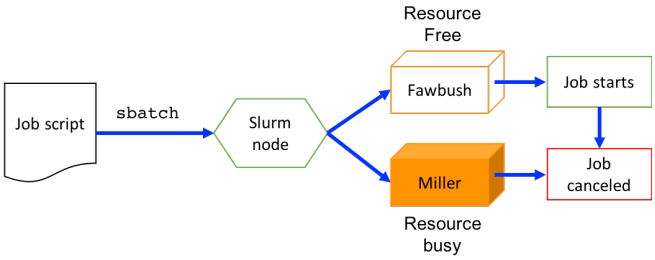

Figure 3. Slurm job federation on HPC11.

management, and cluster setup. HPCM is a more traditional cluster management suite of tools that has adopted some modern aspects of system monitoring and metric collecting through tools like Kafka, ELK, and Alerta. HPCM has a decade long history of supporting HPE and SGI clusters. When Cray was acquired by HPE, they made a decision to add Cray EX support to HPCM as well.

The HPC11 system was one of the first customer systems to utilize this new Cray EX hardware support in an early HPCM 1.4 release. We worked with the HPCM developers to test and improve the early HPCM release. There was an initial learning curve and we encountered issues reconfiguring the system from a Shasta Cray System Management (CSM) configured system to an HPCM configured system, but those were quickly resolved with assistance from the HPCM developers. We were able to make fast progress reinstalling the system and testing the usability of the HPCM solution. Because we were able to quickly deploy an HPCM configured system, engineers and project management decided to pursue HPCM as the acceptance path for the HPC11 system.

HPCM management system is comprised of one `admin` node and three `leader` nodes. The `admin` node is the single point of management of the HPCM system and the `leader` nodes are responsible for scaling those critical services up as the system size increases. HPCM does have a high-availability model that we did not choose to pursue to provide more resiliency, but since the HPC11 is actually a dual cluster, the resilience is pushed into the two hall design and federated scheduling system (see Figure 1. The `leader` nodes can also be scaled up in sets of three, if necessary, as a cluster grows in size, but for the scale of the HPC11 system three `leader` nodes is an adequate number for each system.

2) Federated Scheduling: Slurm, which is developed and supported by SchedMD, was the scheduler selected to be used on the HPC11 system. Slurm has a feature called Federated Scheduling which allows users to submit a job to a cluster and then Slurm replicates that job across all available clusters in the federation. As shown in Figure 3, the job is tracked and managed by the individual cluster scheduler and whichever cluster is able to run it first will run the job and

cancel the job on any other clusters it is also scheduled upon. This allows greater utilization of a two cluster design rather than having a production and development cluster requiring users to switch between them when the production system is down for maintenance. One cluster can also suffer from an unexpected failure and while the current running jobs will fail and requeue, the dual scheduled jobs that are currently queued on the other cluster will still have an opportunity to run on the still operational cluster. Users should not need to know about which cluster their job is scheduled to run on and simply have to submit it and let the federated scheduling system manage the job. This will allow users to not have to guess which cluster they should submit to in order to get their job to be scheduled to run first because the scheduler will handle that for them.

3) Network: HPC11 utilizes a Slingshot 10 high speed network. The network is composed of Mellanox Connect-X 5 100G ethernet network interface cards and HPE Rosetta switches. The Slingshot network brings significant advancements in state-of-the-art congestion control and adaptive routing improving the bandwidth utilization and reducing tail latency on large HPC networks. Each Hall of HPC11 is broken into a 3-Hop dragonfly network with 4 dragonfly groups per hall. Each group contains 16 Rosetta switches configured where each switch has 16 links for computes, 32 links for intra-group connectivity, and 16 links for inter-group (global connectivity). Each dragonfly group contains 200 compute nodes with a global connectivity of 51.2 Tbps.

The edge connection into the HPC11 system was a constantly evolving topic leading up to acceptance. When the system was originally designed to run the Shasta software stack there was a Customer Access Network (CAN) designed to provide an edge connection from the internal ethernet network out to the customer network. When we switched to using HPCM, that was no longer an option as the CAN was a Shasta-specific component, and a supported edge connection method was not available due to the fact that the HPCM-managed solution for Cray EX was still under active development. In order to fill this gap and to provide a resilient login and system services deployment, we designed and introduced a "Hall C" network that provides edge routing from the Slingshot network into the ORNL network.

The "Hall C" network includes of login nodes, Slurm nodes, and the Slingshot edge connections into two Arista switches. This allows login nodes and Slurm nodes to be independent from the compute Slingshot network. The advantage this provides is that each of the compute clusters can have independent fabric managers and can be taken down for maintenance without affecting the login and Slurm nodes. The login nodes have a direct Lustre connection rather than routing through the Slingshot network and the Slurm scheduler node is still available to the users even when the rest of the Slingshot network may be down.

With the change to HPCM, the network had to be redesigned to include a modified ingress. This included new switch hardware and routing design. Network topology now includes a routed link for Slurm, Slingshot, and login nodes. This change simplified the overall design and network segmentation.

4) Software Stack: At the time of acceptance, the software stack comprised of a pre-release version of HPCM 1.4.1, Cray OS 1.3, and the Slingshot fabric manager 1.3. The Cray OS and fabric manager were originally tied to Shasta releases so we needed to extract individual components from them to create the full software stack for HPCM that we would run for acceptance. HPCM was described in Section II-A1. In this section we will discuss the Cray OS and the Slingshot fabric manager.

HPCM supports the SUSE Linux Enterprise Server (SLES) 15 SP2 operating system for the compute node image, but we wanted to provide the Cray Operating System because it has a HPC optimized kernel running on SLES 15 SP1 OS. This choice allowed us to provide the Cray kernel along with all of the drivers compiled against it rather than having to manage that complete stack. With minimal effort, we combined the HPCM provided repositories with the Cray OS repositories and made a hybrid image for the compute nodes to boot.

The Slingshot fabric manager, at the time of acceptance, was only provided as a container in the Shasta release. We extracted that fabric manager from the container and developed a method for it to run natively on the `admin` node. This allowed us to provide an initial version of the standalone Slingshot fabric manager to be used for acceptance without relying on any container services.

For HPC11 acceptance testing, we used the October 2020 HPE/Cray programming environment (PE 20.10) which includes the Cray Compiler Environment 10.0.4, Cray MPICH 8.0.16, and Cray LibSci 20.10.1.2.

5) Telemetry: Metrics and logging for the HPC11 system are sent to both a remote Elasticsearch instance for long term storage and a system local short term Elasticsearch. The short-term instance receives only logs and metrics from the HPCM metric pipeline, which pertains to the system, nodes, network, and facilities. As shown in Figure 4, these metrics are also shipped out to the aforementioned remote Elasticsearch instance, in addition to metrics regarding Slurm scheduling, GPU usage (where applicable), and filesystem metrics.

The HPCM system local store is aggregated using Message Queuing Telemetry Transport (MQTT) brokers, which collect metrics from their various sources and forward them into a centralized Kafka bus. The Kafka bus serves as the central location to obtain the internal metrics. From here, we employ a kafka-elastic connector to send to the local Elasticsearch store.

For the long-term store, we employ Logstash as a buffered

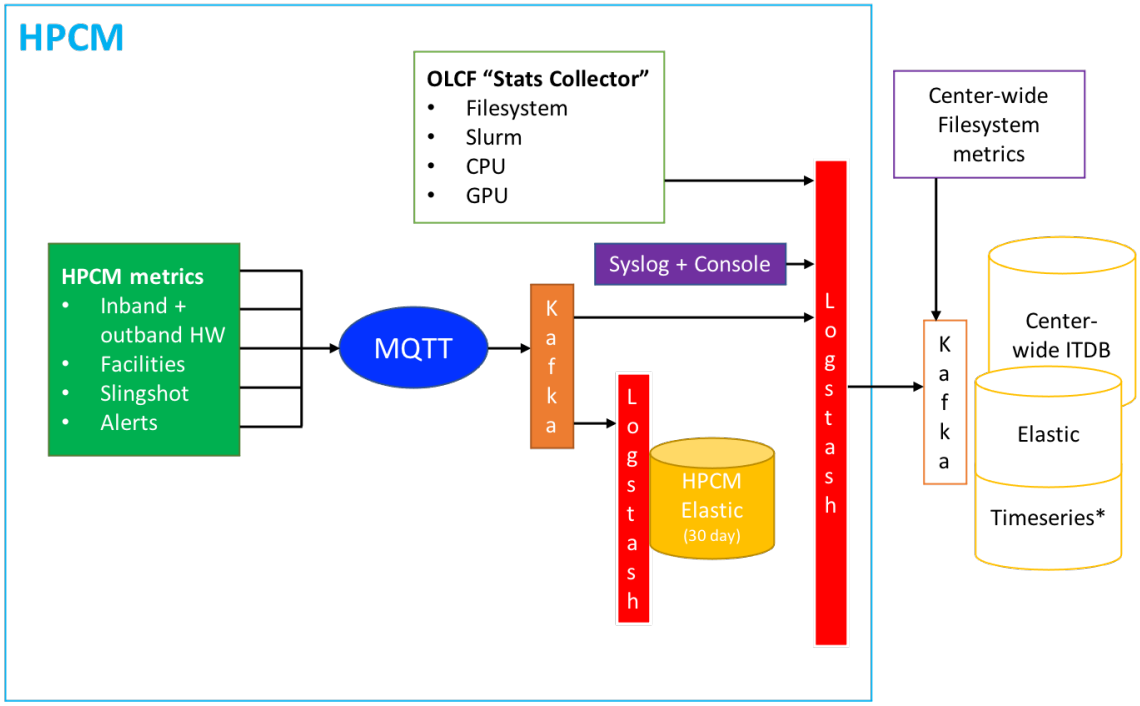

Figure 4. Telemetry system overview.

pipeline between Kafka and the remote long-term store. The Logstash buffer is intended to provide approximately 5 hours of storage in the event of connection loss. The Logstash instance is also where we inject any custom metrics that HPCM does not collect for us, using custom Python metrics collectors. Logstash is also responsible for sending data to the remote Elasticsearch store.

B. Storage System

HPC11 also includes two identical, independent Lustre parallel file systems, Storm and Cyclone. Each file system provides 7.5 PB of usable capacity and performs at approximately 45GB/s each for sequential read and write I/O operations. Additionally, each file system has 110 TB of flash capacity for future data-on-metadata (DoM) use.

1) Hardware: From a hardware perspective, each file system is largely based off a single DDN SFA14KX. The SFA14KX head unit is connected to 10 disk enclosures and presents 12 virtual disks (LUNs) to six physical OSS nodes via SRP over direct-connect InfiniBand. A single ZFS-backed object storage target (OST) is created on each of these 12 virtual disks. For metadata targets (MDTs), a single DDN SFA200NV is direct-connected to two metadata server (MDS) nodes. The SFA200NV is formatted into four virtual disks, two of which are presented as primary to each MDS. These two virtual disks are formatted into a single ZFS pool which is then used as the MDT.

Each file system node is interconnected by 2x100G bonded (active-active) diverse ethernet links to two independent network switches. These switches are directly connected to the compute fabric.

2) Software stack: Storm and Cyclone were ultimately accepted with RHEL 7.9 (3.10.0-1160.6.1 kernel), ZFS 0.8.5, and Lustre 2.12.6 (with patches). We completed testing using fio 3.7 and IOR/mdtest 3.2.1.

III. HPC11 Acceptance Test Plan

HPC11 acceptance testing consisted of two separate activities that were executed in parallel: compute and storage acceptance. Compute acceptance includes the following test elements: vendor test, functionality test, performance test, and stability test. Storage acceptance is comprised of the following test elements: hardware test, functionality test, performance test, and stability test.

As shown in Figure 5, in order to optimize the schedule, we executed a single combined stability test period on each cluster to evaluate the stability of the compute and file system resources.

Each test element for compute and storage acceptance is described in this section.

A. Compute Acceptance Test Elements

1) Vendor testing (VT): This element is executed by the vendor and includes collecting results from component hardware diagnostics, benchmark results, and contractual Figure-of-Merit (FOM) deliverables. For HPC11, application-level performance requirements were set for the two primary applications: UM and 4DVAR.

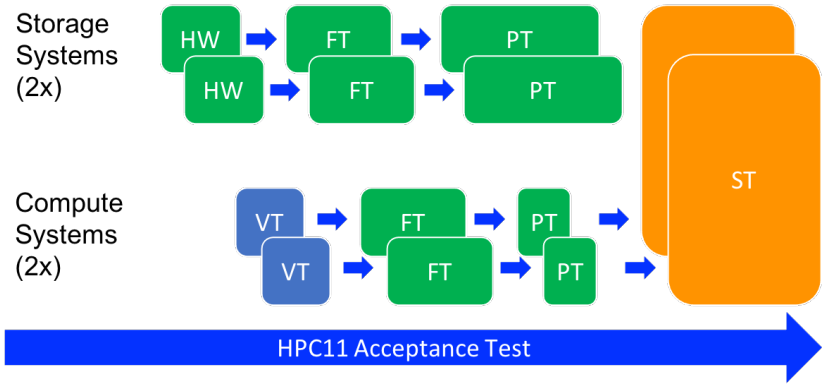

Figure 5. HPC11 Acceptance Test workflow.

Table I
DEFECT SEVERITY LEVEL CLASSIFICATION

Severity	Description
Level 1	A critical component of the system is down or an issue causes significant disruption to workloads (e.g., scheduling is interrupted, incorrect answers produced, data corruption).
Level 2	Service is partially interrupted or impaired and a workaround is not available.
Level 3	A problem that impacts specific workloads or batch jobs without interruption to service.
Level 4	A problem that has minimal impact to users and can be circumvented.

2) Functionality testing (FT): This element ensures individual components of the hardware and software stack are working correctly. It also verifies correct functionality of the selected realistic workloads. The following tests are executed in this element:

- System Administration: cold and warm boot of the full system, failure injection, telemetry data capture, among others
- Network test: injection bandwidth per node, latency, global bandwidth
- Scheduler and job launch tests: SLURM layout, job federation
- Component tests: HPL, STREAM
- Programming Environment tests: compilers, MPI, tools
- Realistic workloads: math and I/O libraries

3) Performance testing (PT): This element focuses on workloads specific to the individual program as well as contractual benchmarks. In this element, tests are executed in isolation to obtain reference values on a quiet system.

4) Stability testing (ST): This element simulates a realistic environment by combining code development activities with batch workloads. During this element, the OLCF Test Harness [9], [10] is used to maintain a continuous stream of jobs running on the system. The stability test period for each cluster, Fawbush and Miller, was 14 days.

During ST, we must identify a root cause for all failures. In addition, all issues are classified based on severity as specified in Table I. If a critical issue emerges during ST, the entire ST period must be restarted after the issue is addressed.

B. Storage Acceptance Test Elements

1) Hardware testing (HW): Hardware testing typically includes a verification against the bill of materials (BOM) and requires successful completion of power-on self-test (POST) procedures. At this stage, all dead-on-arrival (DOA) hardware is identified and replaced.

2) Functionality testing (FT): Functionality testing of the storage system ensures that specific features of the hardware function as expected. In particular, we focus on block device setup, power failure scenarios, and component failure scenarios.

3) Perormance testing (PT): Performance testing of the storage system ensures that the system meets the performance criteria required. This includes block level benchmarks, client level benchmarks, and full file system benchmarks.

4) Stability testing (ST): Stability testing of the storage system includes maintaining a known consistent workload for an extended period of time. Frequently, this workload is complemented with edge cases that stress the file system.

IV. COMPUTE ACCEPTANCE TEST

For HPC11, parallel acceptance tests were executed independently on each of the clusters: Fawbush and Miller. In this section, we summarize the results from functionality, performance, and stability test elements.

A. Functionality Test Results

In this section we will highlight results from key areas of the acceptance test including network validation, job and application launch tests, individual component testing, and application tests.

Table II
HPC11 ACCEPTANCE TEST CONTENT

Test	Description
OSU Microbenchmarks	MPI bandwidth and latency
ALCF MPI	ALCF MPI benchmarks
HPL	High-Performance Linpack
STREAM	Measures memory bandwidth
OpenMP 3.1 verification and validation	OpenMP 3.1 specification
SPEC OMP2012	OpenMP 3.1 functionality and performance
LSMS	Locally Self-consistent Multiple Scattering application
GenASiS	General Astrophysics Simulation System
minisweep	Sn radiation transport miniapp for Denovo
UM	Unified Model
4DVAR	Four dimensional variational data assimilation

Table III
HPC11 NETWORK FUNCTIONAL TEST RESULTS: PROJECTIONS AND MEASUREMENTS

Test	Projection	Measured
Global Bandwidth	5.6 TB/s	9.3 TB/s
Bi-section Bandwidth	2.8 TB/s	3.1 TB/s
MPI Latency	1.7μs	1.8μs
MPI Bandwidth	12 GB/s	11.8 GB/s

1) Network validation: We evaluated the high speed network (HSN) of HPC11 to determine whether the bandwidth and latency at varying scopes achieved the performance requirements of the system. We measured global and bi-section bandwidths using the ALCF MPI benchmarks [11]. As shown in Table III, the bandwidths measured on the system exceeded the initial projections. In the case of global bandwidth, the result was 66% faster than the original target set. We also measured MPI point-to-point latency and bandwidth using the OSU microbenchmarks [4] and showed reasonable performance results, which were very close to the expected performance of the Slingshot 10 projections.

2) Job launch and layout tests: To ensure adequate support for the various use cases expected on HPC11, we designed a comprehensive set of job launch and layout tests.

The batch scheduler provides a mechanism for users to allocate and access a system's compute resources. The allocation created through the scheduler is the only route through which users can access the system's compute resources.

The batch scheduler also provides the center with a method to control the flow of allocation requests on the system's limited compute resources. Through batch job priorities and limits, a center can manage access to a compute system's resources based on center policies. Organizing the queue of allocation requests is important as user requests for compute resources are often larger than the available resources. Because the allocation and control of the request queue is an important step in the use of compute resources, it is also important for a center to test the mechanisms through which the access and limits will be enforced.

Fawbush and Miller use the Slurm workload manager. As part of system acceptance, we performed tests to verify Slurm's job scheduler features. During FT, we tested the ability for a user to submit, hold, alter, remove, control order with dependencies, as well as other scheduler flags. We also performed tests to verify scheduling functionality. For example, only eligible jobs were scheduled for allocation, scheduling performed based on priority, lower priority jobs were backfilled, jobs exited the queue once the specified walltime was reached, and other similar tests.

Once jobs are submitted to the batch queue, it is important for users to monitor the submitted jobs as they progress through the queue. We executed tests to verify a user's ability to view not only their queued batch jobs, but also, all jobs in the queue. Viewing the full queue provides the ability to monitor progress over time as the queue changes due to new job allocations getting scheduled, jobs completing, and new job submissions. We tested Slurm commands such as `sacct` and `squeue` to verify the ability to view the full queue by job state and priority. We also tested the ability to view the entire queue and common details of each job including job ID, username, account, partition, walltime, nodes, and priority.

Batch job accounting is also important to help the center, programs, and users understand usage of the valuable and limited compute resources. We performed tests to ensure the ability to view details of completed batch jobs. Example per job details included job ID, username, account, submission time, start time, end time, partition, and allocated nodes. We also tested the ability to feed completed Slurm batch job details to the center's tracking and reporting systems.

Once the needed compute nodes have been allocated through the scheduler, the job launcher is used to execute a program in parallel on the allocated resources. The job launcher allows a user to control layout within and across compute nodes based on requirements of the parallel program. The launcher also provides users with the flexibility to launch multiple parallel executables either serially or simultaneously depending on the workflow. We tested the job launcher used by Slurm, `srun`, to verify inter-node

and across node task placement and binding. On both Fawbush and Miller, we performed tests to verify job launch performance, task and thread layout, job memory access, and job step binding to bind task/thread to physical core, hyperthread, and a NUMA domain. We performed additional tests to validate simultaneous job steps including single node steps, multi-node steps, non-uniform steps, and steps queued tests.

3) Individual node component testing: Before proceeding with realistic workload tests, we executed a subset of tests on every single node on the system to ensure that all nodes were healthy. These tests included standard benchmarks like HPL and STREAM, as well as, ORNL mini-applications like Minisweep, described in Section IV-A5, which has been able to identify problems in the past.

4) LSMS: LSMS (Locally Self-consistent Multiple Scattering) is a computational material sciences application that uses first principles Density Functional Theory to solve the Kohn-Sham equation and calculate electronic properties of materials. LSMS is written in C/C++ and supports MPI, OpenMP, and CUDA [5]. LSMS has been used for acceptance testing of previous systems at ORNL, including Titan and Summit. For HPC11 acceptance testing, we used the MPI version of LSMS with OpenMP support enabled. The tests executed ranged from 1 to up to 512 nodes.

5) minisweep: Minisweep is a mini-application developed as a proxy for the Denovo Sn radiation transport code. Minisweep replicates the 3-D sweep wavefront calculation in Denovo, which represents the most computationally intensive kernel in the code. Minisweep is written primarily in C++ and supports OpenMP, OpenACC, CUDA, and HIP [**?**] programming models. The mini-application has been able to pinpoint correctness issues on new systems and why we include it as part of the set of tests that we run on every node of the system. For HPC11 acceptance testing, we used the MPI-only version of Minisweep and ran tests starting with 2 nodes up until 784 nodes, each using 4 processes per node. For Minisweep tests, we exercised the GCC compiler and Cray MPICH from the October 2020 Programming Environment as mentioned in Section II.

6) GenASiS RiemannProblem Test: GENASIS (*Gen*eral *A*strophysics *Si*mulation *S*ystem) is a developing multi-physics code to perform simulations of astrophysical phenomena on large-scale supercomputers. Currently, its primary aim is the simulation and modeling of three-dimensional core-collapse supernovae with radiation transport. GENASIS is written primarily in modern Fortran. Leveraging object-oriented features afforded by the standard, GENASIS manages code complexities by using abstractions and encapsulations. GENASIS adopts test-driven development as one of its sofware development practices. As such, GENASIS has unit tests to exercise its classes and application drivers that implement known benchmark problems utilizing its solvers. For this acceptance testing, we used the multi-dimensional `RiemannProblem` in GENASIS fluid dynamics solvers. The `RiemannProblem` starts with two fluid regions separated by a discontinuity in pressure and density. As the simulation progresses, shock develops and propagates within the fluid. The problem benchmarks the code ability to correctly evolved the shock with its associated contact discontinuity. GENASIS `RiemannProblem` exercises the Fortran compiler maturity in the system. Since GENASIS uses OpenMP for both CPU and GPU parallelism, its test problems also benchmark the compiler and runtime performance for OpenMP for the particular CPU on the system.

7) UM: The United Kingdom Meteorological (UK Met) office's Unified Model (UM) is a numerical model used for both climate and weather applications. UM can be used to model at different time and geographical scales [12]. For HPC11 acceptance testing, we used an optimized version based on UM 10.9 on a specific problem size requiring 135 HPC11 nodes that was provided by UK Met. The benchmark case was setup such that 5 simultaneous copies will run concurrently on each cluster, which each one meeting the reference performance.

8) 4DVAR: 4-Dimensional Variational Data Assimilation (4DVAR) is a general method and specific implementation of data assimilation for global numerical weather prediction (NWP). To initialize a global NWP model, such as UM, to predict future real-world conditions, modelers must assimilate data sources that describe current global weather conditions. These data sources may include output from surveillance satellites and surface observations. 4DVAR considers these data, which are spaced sporadically in 3-D space and in time, and interpolates them into a homogenous grid at a single point in time that is used to initialize the global model.

For the acceptance testing, we ran an implementation provided by the UK Met office and that will be used in production to generate the initial conditions for the UM model. The Met Office also provided a standard benchmark set with reference scaling and performance data gathered from a Cray XC40 supercomputer. We ran this benchmark with several decompositions across 16 to 128 nodes, in the same configurations as provided in the reference benchmark performance data.

B. Performance Test Results

We executed several key scientific applications as part of the performance tests including GENASIS LSMS, Minisweep, UM, and 4DVAR. We concentrated in the UM and 4DVAR workloads given that both applications are representative of the types of production workloads that will be executed on the clusters. In addition, both applications have contractual figures-of-merit that had to be met as part of acceptance.

1) UM: We executed the UM test case first in isolation using 135 nodes on each cluster. Then we used the average execution time obtained in a quiet system as the reference value to measure runtime variability in isolation and under a realistic workload.

In isolation, we were able to easily reproduce the runtime reported by HPE on the 135-node UM case. Running 5 simultaneous copies of UM did not significantly impact performance of individual runs. On average, the target UM dataset ran in approximately 1240 seconds.

No performance issues were detected executing this application.

2) 4DVAR: Although the performance did not scale to as many nodes as the UK Met office's reference XC40, this was not a problem because HPC11 outperformed that result by nearly 2 times on the smallest decomposition core count. As shown in Figure 6, the scaling curve flattens between 32-48 nodes versus 128-192 nodes on the XC40. Approximately the same runtime was achieved on 48 nodes of our system compared to 192 nodes of the XC40. The increased core-to-core performance likely results in a sharper flattening of the scaling curve. On this test case, the cores are not saturated and we are unable to scale further with this dataset. More importantly for acceptance, the performance and output were consistent between runs and with the reference system.

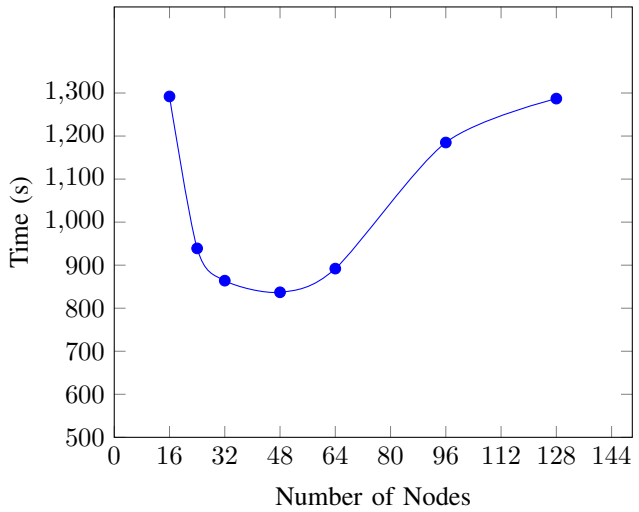

Figure 6. Scaling results of 4DVAR on HPC11.

When this dataset was run at 96 and 128 nodes, we frequently encountered hangs in the code. We conducted extensive debugging and consulted with UK Met to identify the cause and a solution. The UK Met office suggested to try a newer release of 4DVAR, however, for acceptance we needed to use the specific version provided by HPE that would be run in production, so we did not pursue this. Given the excellent performance on a much smaller set of nodes, it is unlikely the code will be run in production with 96 or more nodes, so we did not classify as a critical issue. As shown in Figure 6, peak scaling performance is achieved with a quarter of the nodes, so we speculate this number of nodes is overly decomposing the dataset and would require a larger dataset to reliably run at this scale.

C. Stability Test Results

To ensure HPC11 can support realistic workloads on the system, we conducted a 14-day stability test on each cluster, in parallel. We used the OLCF Test Harness (OTH) to launch a mixed workload that included both the contractual benchmarks, UM and 4DVAR, but also the subset of OLCF applications that were selected for acceptance: GENASIS Minisweep, and LSMS.

In total, over the 14-day stability period, we launched 5,783 jobs on Miller and 5,892 jobs on Fawbush and obtained a pass rate of 98.86% and 99.19%, respectively. Figures 7 and 8 summarize the types of failures we observed. The types of failures we encountered are shown in Table IV.

The majority of failures on both clusters were attributed to the application hitting the walltime requested before completing. Most of these jobs were for 4DVAR jobs using 96 or more nodes which, as mentioned above, is now known to be an unstable configuration. On Miller, we also saw approximately 1% of jobs fail due to node failures. This was expected as Miller became online close to the start of acceptance and had a shorter burn-in period than Fawbush. The 14-day ST is exactly designed to catch these "early life" failures.

Throughout the stability test, we used the OTH to measure the runtime of individual jobs for the two key applications. Overall, between the two clusters, we saw a runtime variability of $\sim 2.5\%$ and $\sim 4\%$ for UM and 4DVAR, respectively. These values are well within the contractual requirements for runtime variability.

V. STORAGE ACCEPTANCE TEST RESULTS

For acceptance, we ran all tests on both file systems. With the exception of some early hardware issues that were corrected, all test results were extremely similar, as expected for identical systems. As such, the results below are given from the perspective of a single file system.

A. HW results

Hardware testing was straight-forward, and no major issues were identified.

B. FT results

The system was first configured consistent with the production block layout. Once complete, redundant power was removed from the system to verify that it would survive a half-power (single side) failure scenario using either power input. Additionally, a total power failure was simulated to

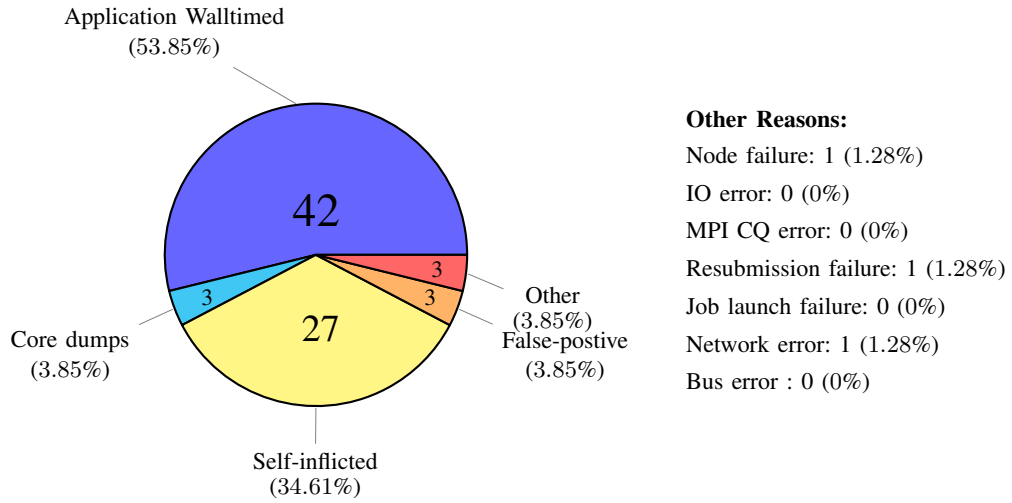

Figure 7. Failure distribution for Fawbush ST (336-hour period).

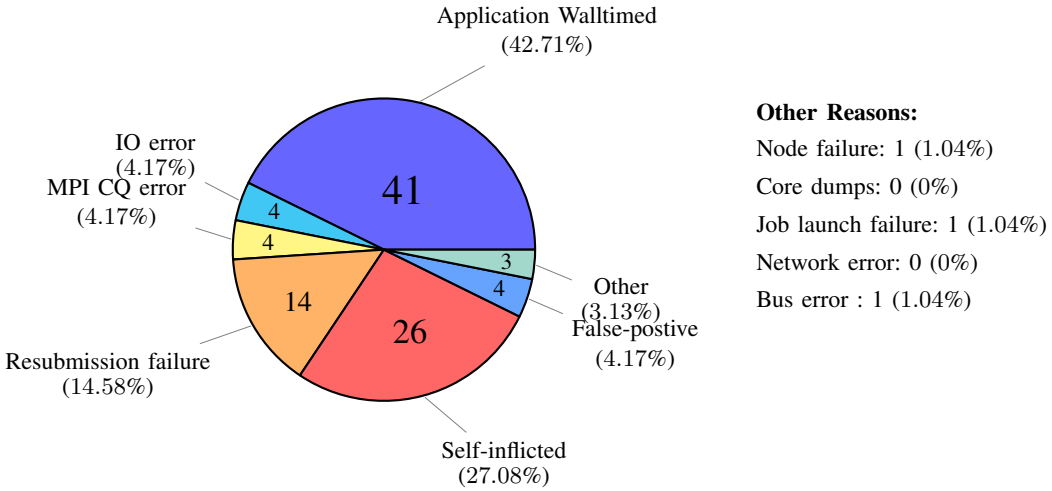

Figure 8. Failure distribution for Miller ST (336-hour period).

Table IV
TYPES OF FAILURES ENCOUNTERED DURING HPC11 ACCEPTANCE TEST

Failure Type	Description
Application walltimed	Job does not complete in the requested walltime.
Node failure	Job fails due to a node failure as reported by Slurm.
I/O error	Job fails due to an `io error` as reported by Slurm.
MPI CQ error	Job fails due to an MPI completion queue error.
Resubmission failure	Job is unable to submit the chained job.
Core dumps	Application fails and left a core file.
Self-inflicted	Job fails due to a script error.
False positive:	Value reported as a failure but reviewing output shows the correct result.

ensure that data was not lost or corrupted. Finally, each component was individually tested for failover (if applicable) and hot-swap capability; this included power supplies, fans, controllers, media, cabling, etc.

No issues were identified in functionality testing.

C. PT results

First, we ran block level benchmarks on the SFA14KX to determine what the maximum theoretical performance was. Using `fio`, it was determined that the SFA14KX maxed out at 52GB/s write and 65GB/s read with our hardware setup and layout. We also benchmarked the SFA200NV, and found the maximum block performance to be 18.0GB/s write and 23.8GB/s. Both were benchmarked with sequential and random workloads; maximum numbers stated are sequential. Initial tests were also performed to determine the best raw MDT performance, eventually settling on the production layout that combines both block devices into a single pool, as described earlier. All block level benchmark tests completed satisfactorily.

Moving on to client and file system level benchmarking, we encountered many significant performance issues. In short, our initial system layout included heavy use of Data on Metadata (DoM), where we encountered significant performance problems. Issues with the native client and DoM were identified, so we migrated to a patched, custom client. This mitigated the issue somewhat, but still did not achieve the desired performance. We have since disabled DoM. We also identified significant performance issues with ZFS integration. Some of this was mitigated, but we still have outstanding issues with ZFS related performance losses.

The following key issues highlight the type of performance degradation observed. Note that this is a subset of all issues encountered, but reflects the highest impact issues:

- Initial "hero" benchmark numbers were ∼35GB/s write and ∼30GB/s read, single client performance ∼2GB/s read/write
- Poor `ksocklnd` performance on our bonded ethernet interface (LU-14293)
  - `iperf` between two nodes was achieving 98Gbps and we were able to demonstrate 190Gbps with a 2→1 setup
  - `lnet` selftest could only hit ∼20Gbps in a node to node test
  - We backported a multiple-socket patch (LU-12815) to Lustre client version 2.12 which resolved this issue
- Client hangs when using DoM with a fixed `mdc lru_size` (LU-14221)
  - We typically set fixed `lru_size` to avoid the potential for a large memory footprint if using dynamic
  - Clients would reliably hang with a reproducer that included large amounts of metadata operations
  - Resolved by backporting LU-11518
- Multitude of grant related issues requiring a custom client, deviating from Cray client

After we backported several patches (both client and server) to Lustre version 2.12, we were able to resolve some of the observed performance issues. After patching, the observed performance is now:

- ∼6GB/s read/write on clients
- ∼45GB/s write/read hero numbers

VI. Lessons Learned

During the deployment and acceptance of HPC11, we discovered several issues that would have negatively impacted user workloads. In this section, we discuss a few high priority issues.

During our validation of performance for UM, we conducted a sequence of runs using the 135-node case to ensure that all 800 nodes in each cluster would participate in a UM job. In one of the jobs, the OLCF Test Harness detected a divergence. Further investigation showed that the divergence was reproducible but only when run on the original set of nodes. This was the smallest job size available for UM and, as a result, we had to bisect the set of 135 nodes until we were able to identify one specific processor in one specific node that consistently produced a silent incorrect answer. The processor was an early AMD test escape and the root cause was identified and remedied in manufacturing. This is one example of the value of using a realistic workload for acceptance testing. If we had included a reduced set of applications, this issue may not have been caught in time.

Prior to the start of the acceptance testing, GENASIS uncovered a compiler bug in Cray Compiling Environment (CCE) Fortran compiler. The bug seemed to have existed in several versions of the compiler. It causes the compiler to generate executable that produced wrong results when the Fortran pointer remapping feature is used to modify the bounds of higher-rank arrays pointing to a contiguous section of a rank-one array. Since this language feature is used in several places in GENASIS, the compiler bug needed to be fixed before the start of the acceptance testing. A new compiler version was released that included the appropriate fix for the bug. This was verified with GENASIS during FT.

During FT, we identified an issue that prevents GDB4HPC from starting successfully. The issue is currently being investigated and was tracked down to `sattach` being unable to connect to a job step in a multi-cluster environment (CASE #275853).

There were a few issues encountered with HPCM that were discovered during acceptance, but most of them have been addressed in newer versions of HPCM released since then. The majority of these issues were fairly minor (severity level 3 and 4), for example, timestamps and logging data being collected and stored correctly for a specific length of

time. We also discovered issues when we introduced our Puppet configuration management onto the HPCM administration server and making sure we were capturing configuration files that needed to be managed and not overwriting configurations that HPCM expected to have in place. This work is still ongoing and we are working closely with the HPCM developers to address these issues.

HPC11 provides a resilient set of compute and storage systems for users. However, the simultaneous acceptance testing of identical systems, significantly increases the complexity of the tests and the coordination required between system administrators, HPC engineers, and the vendor. In addition, simultaneously accepting compute and storage resources results in unavoidable dependencies. As discussed in the paper, when issues were identified on the storage systems, we had to interrupt compute acceptance test until they were resolved. For future procurements, we recommend using a previously available storage system, if possible.

Because we utilize an already well-established acceptance test procedure and testing framework, the development effort invested to port tests to a Cray EX system and Slurm-based resource will be useful for future acceptance testing of future deployments, including Frontier.

VII. Conclusion

In this paper, we have described the entire process followed to successfully execute acceptance testing of the HPC11 system. This system includes two identical compute resources, Fawbush and Miller, and two identical file system resources, Storm and Cyclone.

In addition, we have provided a summary of the results obtained, issues encountered, workarounds developed, and open issues that still remain.

The information provided here could be helpful to other centers interested in transitioning from a Shasta CSM-based system to a HPCM-based one.

While the HPCM solution was still in development in the beginning of acceptance, since the system transition to operations, we continued working with HPE to formalize the designs and procedures described here.

Acknowledgment

The authors would like to thank the Cray/HPE team for their invaluable contributions to acceptance testing of HPC11. In particular, Jeff Beckleheimer, Adam Sachitano, Cathy Willis, Pete Johnsen, Eric Dolven, Dave Londo, and Kim Kafka. We would also like to thank Matt Ezell and Don Maxwell from ORNL for lending their expertise to make the transition to HPCM a success.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

References

[1] V. G. V. Larrea, W. Joubert, M. J. Brim, R. D. Budiardja, D. Maxwell, M. Ezell, C. Zimmer, S. Boehm, W. Elwasif, S. Oral et al., "Scaling the summit: deploying the world's fastest supercomputer," in International Conference on High Performance Computing. Springer, 2019, pp. 330–351.

[2] C. Wang, S. Chandrasekaran, and B. Chapman, "An OpenMP 3.1 Validation testsuite," in International Workshop on OpenMP. Springer, 2012, pp. 237–249.

[3] "SPEC OMP2012 Benchmark Suite," https://www.spec.org/omp2012/.

[4] "OSU Microbenchmarks," http://mvapich.cse.ohio-state.edu/benchmarks.

[5] M. Eisenbach, C.-G. Zhou, D. Nicholson, G. Brown, J. Larkin, and T. C Schulthess, "Thermodynamics of magnetic systems from first principles: Wl-lsms," in Proceedings of the 2010 SciDAC conference, 04 2010.

[6] C. Y. Cardall, R. D. Budiardja, E. Endeve, and A. Mezzacappa, "GENASIS: GENERAL ASTROPHYSICAL SIMULATION SYSTEM. i. REFINABLE MESH AND NONRELATIVISTIC HYDRODYNAMICS," The Astrophysical Journal Supplement Series, vol. 210, no. 2, p. 17, jan 2014. [Online]. Available: https://doi.org/10.1088%2F0067-0049%2F210%2F2%2F17

[7] O. B. Messer, E. D'Azevedo, J. Hill, W. Joubert, M. Berrill, and C. Zimmer, "Miniapps derived from production hpc applications using multiple programing models," The International Journal of High Performance Computing Applications, vol. 32, no. 4, pp. 582–593, 2018. [Online]. Available: https://doi.org/10.1177/1094342016668241

[8] "AFW HPC11 User Documentation," https://docs.afw.ornl.gov/.

[9] V. G. V. Larrea, M. J. Brim, A. Tharrington, R. Budiardja, and W. Joubert, "Towards acceptance testing at the exascale frontier," in Proceedings of the Cray User Group 2020 conference, 2020.

[10] "The OLCF Test Harness repository," https://github.com/olcf/olcf-test-harness, 2021.

[11] V. Morozov, J. Meng, V. Vishwanath, J. R. Hammond, K. Kumaran, and M. E. Papka, "Alcf mpi benchmarks: Understanding machine-specific communication behavior," in 2012 41st International Conference on Parallel Processing Workshops, 2012, pp. 19–28.

[12] "Unified Model," https://www.metoffice.gov.uk/research/approach/modelling-systems/unified-model/.