Producing the Software that Runs the Most Powerful Machines in the World

The Inside Story on Cray Software Test and Release

Kelly Marquardt
Director of Engineering
Cray, Inc.
kellym@cray.com

***Abstract*— Starting from the point when new software features have been coded, a thirteen week process of testing, bug fixing, and release activities kicks into gear to produce high quality CLE and SMW releases. This paper will cover the types of testing, configurations tested, and related challenges addressed with the goal of providing early adopters with a solid understanding of the newly released system software.**

(Abstract)

Keywords-component; software test, software release, release, configurations

I. INTRODUCTION

Cray software releases are put through a full suite of testing. The software testing includes focused functional testing of new features in the release along with full regression suites covering the existing feature set. In addition, the release is put through a battery of system focused testing, including stress testing, scale testing, performance testing and reliability runs. Stress testing focuses on running multiple test suites concurrently and putting a heavy load on the system aimed at finding issues that might cause node loss or system wide outages and to evaluate overall system stability

Challenges include testing at scale, handling the complexity of configurations, and interfacing with third party and community software providers. Partnerships with key customers are an important component of the overall test strategy.

This paper will focus on the typical minor or update software release cycle Major releases that include a significant update to the base operating system follow a longer schedule to allow for the longer stabilization time that is typically required but otherwise follow the same basic process.

A. *Guiding Principles*

The mission of the Cray HPCS software team is to provide outstanding software products that enable our customers to harness the extraordinary power of Cray supercomputers and maximize scientific and engineering application performance.

Guiding principles for Cray software:

- Performance is a key metric of our Market
- Differentiate when strategically sound
- Follow industry standard solutions where they exist
- Collaborate with customers and the community
- Focus on quality in the areas of
    - Timely software releases
    - Updates with no regressions

B. *Focus on Quality*

Cray's software test and release process supports this focus on quality through a structured release methodology and rich collection of test suites. The software testing includes focused functional testing of new features in the release along with full regression suites covering the existing feature set. Cray software releases are put through a battery of system focused testing with the goal of evaluating overall stability. System test suites include stress testing, scale testing, performance testing and reliability runs. The focus of system testing is on evaluating the system's ability to run solidly in production at the customer site, to find any issues that might cause node loss or system wide outages, and to uncover problems that only occur with extended uptime.

II. TYPES OF TESTING

Cray's software test approach uses a wide variety of testing activities with different areas of focus. The following table describes the purpose and focus of the software testing activities. These will be described in more detail later when the overall release process is discussed.

TABLE I. TYPES OF TESTING

Type	Definition
Feature Testing	Functional testing of a new software feature; Tested both inside and outside the release branch; Feature tests are automated so they can be run in future regression test suites
Regression Testing	Automated test suites are run multiple times

	throughout the release cycle and include over 14,000 test cases canvasing I/O, Kernel, System Calls, Memory Management, OOM, Threads, Interconnect, Jitter, Cray-specific Features: ALPS, Core Specialization, Core Affinity, Node Health, Link Resiliency, Quotas, and many others. Also spans a wide range of programming languages; UPC, Co-Array, Shmem, MPI and others. The suites contain benchmarks, kernels, and full blown applications
Stress Testing	Test suites are run concurrently in order to put a heavy load on system. Focus is on how the system holds up under stress
Scale Testing	Scale testing involves testing features at high node count scale, application testing and testing the overall system at large scale. Focus is on evaluating the system's ability to run solidly in production at the customer site
Performance Testing	Automated performance tests are run to measure node-to-node throughput, ping-pong, multi-pong, all-to-all, HPCC latency, I/O. Focus is on ensuring that the current release runs at the same performance levels as previous releases
Reliability Testing	System reliability runs (RelRuns) are performed multiple times throughout the release cycle and involve running the system for an extended period covering multiple days under heavy load. Focus is on finding issues that might cause node loss or system wide outages and to evaluate overall system stability
Exposure Testing	Systems running the release in progress are made available to the user community within Cray including OS developers, testers, programming environement developers, benchmarking, and applications. Focus is on exposing the release to general usage in order to find issues that might not be seen by targeted testing

III. The Release Cycle

The Cray software release cycle follows a well-defined timeline and process for managing the testing and other activities that are part of generating a software release.

A. *Train Model*

Cray software releases for CLE and SMW follow a train model to produce a new release each quarter. This release model is structured to support the focus on timely software releases. The concept of the train model is that releases happen at defined times and that features that are ready in the right timeframe can "get on the train". Release milestones define when the development and test activities for each feature need to be done in order for the feature to be included in the release.

Software features that will be included in a release are identified during the planning phase. In order to keep a release on schedule, features can be moved out to the next release if the development and test completion milestones are not met or if there are critical bugs associated with the feature that cannot be resolved in time for the release. When it isn't possible to move the feature to the next release, then other options are considered. Sometimes late features can be delivered as patches or special releases. Other times, the feature must be in the release and so we must evaluate the risk to the release. In looking at late feature risk, there are some key considerations that are evaluated in managing the risk. The primary concern is around protecting the base functionality and stability of the release. Features that touch key areas such as job launch, file systems, and memory management, to name a few, would be considered high risk. Features that are more isolated in their functionality or those that can effectively be "turned off" are more readily accepted late into the release. In those cases the risk of the feature is isolated to early adopters of the feature rather than impacting the installed base. At times, there are features identified as "defining" for the release. This means that if the feature is late, the release will be allowed to run late to include the defining feature. This approach is used rarely because it makes it very difficult to stay on the quarterly cadence for future releases.

B. *Release Timeline*

A typical minor or update software release follows a thirteen week timeline from the time that the feature code for the release is completed until the release is available for customers. The main testing focus areas during the release cycle are feature testing and system testing. Feature testing refers to focused functional tests of new feature functionality. System testing focuses on full system stack testing, feature interactions, regression testing of previously delivered features, stress testing, scale testing, and reliability runs. The system test cycle is also used to perform system level qualification of new hardware platforms, such as a new processor family.

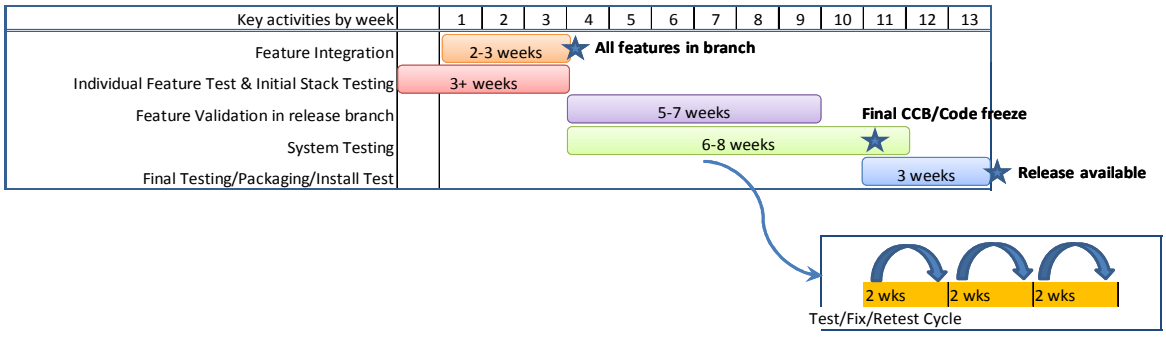

Figure 1. Release Timeline

C. *Feature Integration*

Cray's source code management methodology uses a main development repository where all developed code is stored during development along with release branches to manage the code for a release. As feature code is developed and unit tested, it is checked into the main trunk of the repository, referred to as DEV. Once checked into DEV, the feature code gets its first exposure. Feature and system testing often start in DEV when schedules support that and then continue in the release branch.

Source code for releases is managed with a release branch of the repository. During the release cycle, the release branch is managed by a change control board (CCB) that meets weekly to make decisions about which code changes, referred to as mods, are promoted to the release branch. The release branch for a major release generally branches directly from DEV, while update releases may branch from an earlier release branches, with significant merges of new feature code from DEV. The branching plan for specific releases varies depending on the nature of the features of the release.

During the feature integration phase, feature code is integrated into the release branch according to plans created to manage the level of instability in the code base. As portions of functionality are integrated, feature and system level testing is done to evaluate the initial level of stability of the branch. Once all feature code for the release is merged to the release branch, the feature integration phase is complete.

D. *Individual Feature Test and Initial Stack Testing*

Individual feature testing happens both in DEV and in the release branch while feature integration is underway. Initial stack testing happens during this phase as well. This testing is the first time that all layers of the candidate release software are tested together along with third party software such as work load managers. The goal of the initial stack testing is establishing a base level of stability and functionality to go forward with the release.

E. *Feature Validation in the Release Branch*

Feature testing in the release branch is done to ensure that newly developed feature functionality is working as intended. Tests cover the functionality across the full range of use cases, including error cases as well as designed interactions with other features. As part of developing feature tests, the tests are automated so that they can be included in the regression suites for feature coverage in future releases.

F. *System Testing*

System testing of a release includes many different types of testing. System testing is looking to find interaction issues with features, interactions with system hardware or software, edge cases, usability, and system issues that occur after extended uptime. System testing includes automated regression testing of feature functionality existing in the code base. Also included are stress testing, performance testing, scale testing, and reliability runs.

The workhorse of the system test cycle is the reliability testing, or "RelRun" as it is referred to within Cray. The goal of the RelRun is to put a sustained load on the system for an extended period of time, typically 48-72 hours, in order to ensure that the system is stable and ready for production use. The RelRun workload is designed to simulate a demanding customer workload. During a typical release cycle, RelRuns are performed approximately every other weekend. This cadence allows for bugs found in a RelRun to be triaged, fixes to be implemented and exposed in DEV, and then the mods to be promoted to the release branch so that the next RelRun will not hit the same problems. Before spending the machine time to do a RelRun, there must be a critical mass of fixes or new feature functionality available.

RelRuns are generally performed on our largest available machines. One of the measurements tools for the RelRun is its achieved level of testing. This level indicates how well the RelRun kept the system busy, the job mix achieved on the system and the size of applications used in this system mix. The level of a RelRun is described by using the following 3 elements and adding them together to obtain a RelRun level. The best possible RelRun level would be a 15.

- System Load: The system shall have an average compute node utilization of 80%. Example of things that can decrease this number; job launch issues, file system being slow, jobs not exiting, system being down for extended periods of time, and others. Desired level: 5
- Job Mix: Containing all possible feature tests, benchmarks, applications, and OS and I/O jobs. Example of things that can decrease this number; having to disable tests due to system issues, features not working or being available, system components not being available (like data virtualization service (DVS), Cluster compatibility mode (CCM), or Lustre for example) Desired level: 5
- Application Size: A range of application sizes from 1 core to approximately 1/3 of the machine size. Example of things that can decrease this number; Attempting to do a reltun on only a 1 cabinet system, jobs not working at desired size, or job being limited because of system issues. Desired level: 5

During the system testing phase, release metrics are gathered and evaluated on a weekly or biweekly basis. These metrics show performance against the goals for the release and are used to measure progress.

Release Criteria Description	Release Criteria	Attempted For This Run	Actual
RelRun Level (Describes load, mix and size. See below for description)	15	15	12
Longest System Uptime	48	48	32.7
Compute Node Loss	0.66 compute nodes/cabinet/day	Same as release criteria	6.04
System Interrupts	0	0	1
Service Node Loss	0	0	0

Figure 2. RelRun Metrics

G. Scale Testing

Testing of Cray's software releases at the extreme scale of our customer's systems is always a challenge. Two aspects of scale are important. First, we look for functionality at scale – does the system and all the software operate as intended at extreme node counts? The second aspect of scale is somewhat more subtle and is focused on issues that occur very rarely. These types of issues can be hard to see until a very large system is deployed and they start occurring frequently enough to be noticed and pursued and so are considered an aspect of scale testing.

We use several strategies to cover scale testing. Extensive testing is done on our in-house scale systems. Stress testing is done with the goal of mimicking the load of a larger machine. Simulators are also used to provide scale coverage. An example of this is a simulator that was built for use in testing the hardware supervisory system (HSS) software during the Cascade development timeframe. This simulator allows for the simulation of a range of system configurations from a few densely populated cabinets up to 200 sparsely populated cabinets. This capability has proven invaluable for finding software scaling bugs prior to deployment of the software on internal or customer systems. There are also opportunities for Cray R&D to do scale testing on customer machines in house during the manufacturing timeframe, which is very valuable for proving operation at scale before large machines get to the field.

Finally, Cray R&D works with Cray and customer field service personnel to do early testing of software releases on customer systems. This is often done in partnership with customers who have a special interest in a new feature, such as a particular Lustre version. Pre-releases can be made available after code freeze for this type of testing. Planning for this type of testing is done strategically based on customer interest and the risk of significant new features operating at scale. These cooperative testing experiences have been mutually beneficial to both Cray and the customers involved.

H. Release Readiness, Final CCB, and Code Freeze

A few days before the final CCB meeting and code freeze date, a release readiness review is conducted to review the status of the release against plans and determine if the release will freeze on the planned date. The release readiness review looks at the final set of metrics, the list of release critical bugs and plans for fixes, feature deferments, if there are any, along with other release criteria, such as the status of documentation, plans for installation testing, and so on. The outcome of a successful release readiness review is approval to freeze the release on the planned date.

I. Final Testing, Packaging and Installation Testing

After code freeze, final system testing is done. This testing typically includes a final RelRun, along with whatever other specific testing is needed to verify fixes for release critical bug fixes that went in at the final CCB. Final steps are taken to create the release packages and then installation testing is performed. Installation testing covers fresh installs and the most typical upgrade paths and includes coverage for standard features such as Lustre, DVS, CCM, boot, and system database (SDB) node failover as well as any new features in the release. While exhaustive installation testing is very time consuming, Cray's installation testing is as extensive as possible to ensure customers will not encounter issues with their upgrades.

IV. MACHINES, MACHINES, MACHINES

One of the most challenging aspects of developing, testing, and delivering new features is managing the use of our internal machines. This has become increasingly challenging in the last several years as new Cray software features have been introduced that require unique machine configurations, such as high availability SMWs and support for a collection of workload managers, across an assortment of processor types and SKUs, different cabinet types, along with a variety of storage options. The goal of machine configuration and usage management is to maximize development, test and data center access to all of these configurations and combinations within the bounds of data center limitations.

Cray has a significant investment in internal systems for development and testing. The figure below shows our internal XC30 machines, their primary purpose, what version of software they run, their storage and any special configurations in place

Cray uses a three stage strategy for deploying newly developed software on internal machines. The goal of carefully controlling deployment is to minimize the impacts of problems with new code and especially to prevent the larger machines with a broad user group from becoming disabled by problems with the new code. New mods are first exposed and tested on DEV platforms. The DEV platforms are small, typically between four and twelve compute nodes, and have a limited user community. Regression tests are run nightly on DEV systems to find any regressions caused by new mods. Once mods have been in DEV for a week, they are eligible to be promoted to the release branch for the release that is in progress.

On a weekly basis the CCB meets to approve mods to be promoted into the weekly release branch for build and deployment. Once mods are in the release branch, they will then get installed onto the test machines that are used for testing the release in progress. These machines are larger, typically around a single cabinet, and have a wider assortment of blade types, more storage types and other configuration options. The release branch machines are used for both feature and system testing throughout the cycle. The full regression suites are run at least weekly on these machines, so mods in the release branch get good regression exposure within a week.

The third level of deployment is to Cray's internal data center. Data center machines are the largest machines that Cray has for internal use. The data center is run independently from the R&D organization, with a separate administrative team. This separation is designed to create an environment similar to that of our customers and to give us an independent view of the usability and readiness of software releases. These machines are used extensively by

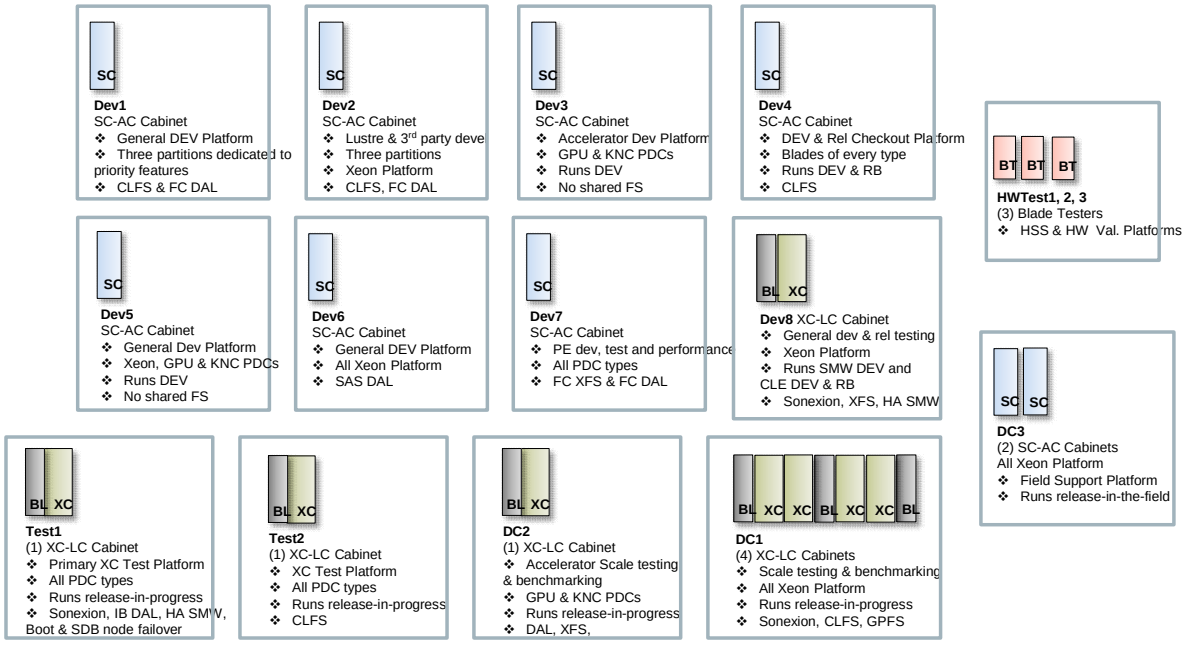

Figure 3. Internal XC30 Machines

the R&D test team but are also used by a wider community within Cray, including benchmarking and applications engineers. The data center machines generally run the release in progress and deploy the latest release branch code a week after it has been exposed on the test machines.

V. CONCLUSIONS

Cray's software test and release process supports our focus on software quality and timely releases through a structured release methodology and rich collection of test suites. The rigorous system testing methodology puts focus on system stability and suitability for a production environment. While testing at the extreme scale of Cray systems is always challenging, the use of our extensive in-house systems, stress testing, and the use of simulators are some of the ways that Cray meets this challenge. Testing at scale is enhanced by partnerships with interested customers. The process and test strategies that Cray uses support our commitment to software quality.

ACKNOWLEDGMENT

The author thanks Janet Lebens, Dennis Arason, Larry Kaplan and many others who contributed to this paper.