Bringing Up Cielo: Experiences with a Cray XE6 System

or, Getting Started with Your New 140k Processor System

Cory Lueninghoener cluening@lanl.gov

Daryl Grunau dwg@lanl.gov

Timothy Harrington toh@lanl.gov

Kathleen Kelly kak@lanl.gov

Quellyn Snead quellyn@lanl.gov

September 11, 2011

Abstract

High Performance Computing systems are complex to stand up and integrate into a wider environment, involving large amounts of hardware and software work to be completed in a fixed timeframe. It is easy for unforeseen challenges to arise during the process, especially with respect to the integration work: sites have dramatically different environments, making it impossible for a vendor to deliver a product that exactly fits everybody's needs. In this paper we will look at the standup of Cielo, a 96-rack Cray XE6 system located at Los Alamos National Laboratory. We will examine many of the challenges we experienced while installing and integrating the system, as well as the solutions we found and lessons we learned from the process.

Tags: HPC, configuration management, system integration

1 Introduction

High Performance Computing (HPC) systems are complex to stand up. They generally involve the delivery of a large amount of hardware at one time that must be installed, configured, and integrated in a fixed period of time in preparation to be run in relative steady state for five to ten years. While physical installation is usually a job for the hardware vendor, configuration and integration normally fall on the shoulders of a team of integrators and system administrators that will be charged with managing the system after it is put in production. Configuration and integration include such complex tasks as configuration management (CM) system design and implementation, parallel filesystem and network integration, and accounts system and user services integration. Standing up a new HPC resources can be similar to bringing up an entire new data center from scratch, with all of the same difficulties and pitfalls.

In this paper we will look at the installation and integration of Cielo, a 96-rack Cray XE6 compute resource sponsored by the Alliance for Computing at Extreme Scale (ACES) project and located at Los Alamos National Laboratory (LANL) and Sandia National Laboratory (Sandia). The complete Cielo family consists of four machines: Smog and Muzia, two 1/3-rack test systems; Cielito, a 1-rack development system; and Cielo, a 96-rack production system. Muzia is located at Sandia, while Smog, Cielito, and Cielo are located at LANL. In this paper we will focus on Cielo, but all four systems run the same software stack and have similar configurations, and our experiences with Cielo align with our experiences on the smaller systems.

2 System Overview

Cielo is a 96-rack, 142,304-core Cray XE6 system operated by the HPC Division at LANL. Delivery of the system began with the arrivals of Smog and Muzia at their respective laboratories in May of 2010, followed by delivery of Cielito in June of 2010 and 72 racks of Cielo in August of 2010. The final hardware delivery occurred in April of 2011, when Cielo was expanded to its full size of 96 racks. In this final configuration, Cielo consists of 8518 16-core compute nodes with 32GB of memory each; 376 16-core visualization compute nodes with 64GB of memory each; 286 IO nodes; 16 internal login nodes; and a handful of infrastructure nodes. Cielo debuted on the Top 500 List of Supercomputer Sites at position six in July of 2011.

The Cray XE6 is a massively parallel processing system that consists of compute and service blades connected by a high-speed torus network. The basic building block of the system is a *chassis* that holds eight compute or service blades. Each of these blades contains four diskless *nodes* that are designated as either *compute nodes*, where actual jobs run, or *service nodes*, which provide management, data virtualization service (DVS), or other services to the machine. Three chassis can be placed in a *rack*, and many racks can be combined in rows to create large systems. All of the nodes are connected by Cray's Gemini network, providing a three-dimensional torus high-speed network for user job communication, node management, and filesystem access.

All compute nodes are headless, diskless machines with no external network connections, but service nodes have one or two on-board PCI slots for expansion. These slots are most commonly used to provide outside network connections for login nodes or file server nodes. Two of the service nodes are designated as special: the *boot* node and the service database (*sdb*) node. These two nodes have external connections to a RAID device, the *bootraid*, and are used to provide persistent services to the rest of the nodes such as logging, root filesystems, and administrator management capabilities. One main task of the boot node is to serve the *sharedroot* from the bootraid to the other service nodes, who mount it as their root filesystem. In our case we also have a set of DVS nodes that serve this filesystem out to the compute nodes, giving them an optional complete Linux environment.

Outside of the main XE6 racks are two classes of standard rack-mounted machines: the system management workstation (SMW) and external login nodes. The SMW is the main administration server for the system, controlling low-level hardware access, system bootstrapping, and system ramdisks. The external login nodes are larger memory, diskfull analogs to the standard login nodes on the system blades that make user access more similar to standard clusters.

3 Challenges

Before pieces of Cielo even showed up at lab, the system administration team realized that we were going to have challenges integrating a Cray system into our environment. We currently run on the order of 20 HPC clusters, ranging from tiny (tens of nodes) up to huge (thousands of nodes), and we have been very careful to put configuration management at the forefront when installing new systems. All of our existing clusters are fully managed by Cfengine[1], with a collection of dedicated or multi-cluster CM servers covering every system in every cluster. This has tremendously helped our relatively small team keep all of the systems in sync and under control.

We quickly discovered that Cray has taken a "fully managed appliance" approach with the XE system. Part of this is due to the way the system itself is designed: the compute infrastructure is tightly coupled by its torus network, with the compute and service node blade hardware fully managed by their control system. The compute nodes run a stripped down Linux-based operating system that is shipped as a Cray-provided image, while the service nodes run a Cray-branded release of SuSE's SLES 11 Linux distribution. Were we running Cielo as a stand-alone system, these features would all be fantastic: we could dedicate people to the system to keep it patched and running via Cray's methods, and that would be that. In fact, this appears to be a common way for sites to run large Cray systems: Cray will assign one or more software and hardware engineers to a site to handle much of the day-to-day management of the system, leaving the system administration team with more of a black box system to take care of.

Due to the existing usage models on our other systems, it was decided that we had a strong business case to not take the fully managed appliance route with Cielo. This way we could provide users with the most consistent environment across all of our systems while keeping our system administration team as flexible as possible. Instead, we needed to have our administrators integrate the system into our family of compute resources as tightly as we could. In our case, this was definitely the right decision: we've spent years keeping a wide variety of systems consistent to minimize the learning curve of existing users on new systems, and it has worked very well. However, it did present a series of challenges to the system integrators and administrators bringing up the systems. The challenges related to bringing up Cielo can be categorized into two broad areas: vendor relations and software.

3.1 Vendor Relations Challenges

The integration team that brought up Cielo was very fortunate to have a strong working relationship with Cray. The Cielo contract included provisions for Cray hardware, software, and application engineers to be located on-site at LANL, and through the bringup process we had access to many extra resources at Cray as we needed them. However, instead of just using the engineers as managers of the appliance, we worked with them to fully integrate the system with our already existing infrastructure. This being the first Cray system located at LANL in many years, there were some challenges on both sides as we worked out how we could work together most effectively.

Since we were looking to closely integrate Cielo into our environment, we ended up digging deeper into the inner workings of the machine than some of our Cray engineers were used to. In the process, we found that there was a lot of in-house knowledge at Cray that, although easy to get, was not always obvious to ask about. There were several times that we learned that a standard assumption about the management of the machine was incorrect, including such details as the best way to reboot nodes, the most effective way to correlate logs between the compute and service nodes, and how to update software. One of our running gags became the hypothetical response "Oh, you still do it that way?" to any question we asked of Cray.

On the bright side, we also had good challenges in this area: Cray provided written step-by-step procedures for almost everything we needed to do to the system. In many ways we weren't ready for this, especially when compared to the many commodity clusters we have in our environment. The biggest challenge here was figuring out how to make the best use of the documentation, whether it be importing the actions into Cfengine scripts, importing the documentation into our own library, or explicitly deciding to follow our own path. Cray was also very receptive to our suggestions for improvement. The challenges listed in the next few sections are often clearly related to our desire to do something our way instead of Cray's. However, they were ready to hear our suggestions and pass them on to their developers in most cases. There were several instance throughout Cielo's bringup where specific nuances we found difficult were changed in later software updates.

3.2 Software Challenges

Over the course of the year that we have been working with Cielo, the majority of the challenges we have faced have come on the software front. As mentioned before, it is very common for Cray to assign a group of software engineers to a site with one of their large systems. We believe this practice has lead to many of our difficulties - Cray wasn't ready for us to be downloading and working with many of their software products, and we didn't have the experience needed to understand some of their distribution, packaging, and installation decisions.

3.2.1 Software Releases

There are three styles of software updates that we have worked with from Cray: cumulative service pack-style updates containing all previous updates for a particular product; individual patches and field notices that will eventually be rolled up in a cumulative update; and sliding window updates that contain older versions of the related software for compatibility as well as the latest update. Cumulative updates are generally released quarterly and contain new functionality, bug fixes, and other substantial updates. Individual patches and field notices are released as needed between the quarterly updates and generally fix bugs or security issues. Sliding window updates are used for the Cray programming environment and includes the both the latest and the last n releases of their supported compilers and libraries, where n is determined by Cray's release engineers based on provided functionality and customer usage. Each style of update is packaged differently, but each generally consist of a monolithic installer script, one or more directories full of RPMs, and fairly detailed instructions on how to install the update.

These individual release styles are further fragmented by individual update idiosyncrasies. Some updates are available publicly to all registered Cray users, while some are only available to Cray engineers. Most are applied by use of an included monolithic install script, but some are applied in a more manual process by following instructions in an install document. Some are applied in a way that will be preserved with future updates, while others are applied in a less stable way. Finally, versioning can be confusing: many packages include an SVN repository version number in their version string that refers to the repository revision from which it was generated. If branching happens in an unusual way, this can (and does) lead to newer software having a lower "version number" than already-existing software. All of these details are easily absorbed when the system is looked at as a standalone appliance, but in a more integrated environment that is used to a high level of update automation, they present a challenging hurdle.

3.2.2 Software Practices

Along with acclimating to Cray's software release methods described above, we ran into challenges with the software they contained. One of the biggest difficulties involved abuses of the RPM package management system. Most of the original software installs and subsequent updates provided by Cray comes in the form of RPM packages and monolithic install scripts that examine the hardware and software currently in use on the system and install required software as needed. This involves installing most packages with `--nodeps` and `--force` options that override RPM's built-in dependency and safety checks. Again, these options fit the appliance model very well: the supplied software is vetted by Cray in a specific format, and they want to replicate that format closely in the field. However, they make verification and integration into an already-existing CM system difficult.

Similarly, Cray's distributed RPMs often make use of postinstall scripts to take care of large portions of the install. This use ranges from fairly benign (creating links if some other packages are already installed) to difficult to manage (RPM only contains a .tar.gz file that is unpacked by a postinstall script). These "write-only" RPMs also fit into the appliance model, but have many problems in a wider-ranging environment: they are difficult to verify, they can seem to behave non-deterministically depending on install order, and they tend to require more hand holding than more well-behaved packages. We found these packages especially difficult to place under CM control, as discussed in the next section.

Finally, we ran into several inconsistencies with respect to how software versions were managed. The `modules`[2] package and `/etc/alternatives` system[3] are two existing packages created to ease the selection of and switch between multiple versions of equivalent software installed on a system. The Cray software stack uses both of these packages to manage its software versions, even using both on the same package in some cases. In most cases they also use a third method involving "default links": symbolic links in each package's install path that point to the install root of the version that should be treated as default. These different methods are used to varying degrees by different pieces of the overall system - the modules are mostly used for normal users of the system to choose compilers, libraries, and related packages; while the alternatives and default links are mostly used by system-level processes. However, there is overlap between the usage of each.

3.2.3 Configuration Management

We discovered many of the challenges listed above while working to put Cielo under complete configuration management control. LANL's HPC division has traditionally used Cfengine to manage its systems, relying on it to create a uniform management environment across all of its clusters. Early on we recognized that this would be more difficult on Cielo than on traditional clusters, but we knew that it was the best way to integrate the new machine into our group's management rotation.

The software practices mentioned in the previous section all made configuration management challenging: creative uses of RPM, inconsistent versioning, and multiple management methods all add layers of complexity to the CM problem. However, the biggest challenge of all was Cray's use of monolithic install scripts. While very handy when installing software and updates interactively, these scripts made it difficult to automate configuration of the machine. Some of these scripts were easy to analyze and either import into Cfengine or call directly during the install process. Others were much more problematic: one explicitly checked to make sure it was connected to a TTY and exited with an error if it wasn't, while another stopped in the middle and presented a set of commands for the administrator to run in a second window before telling the script waiting in the first to continue. In another case, the script didn't even trust itself - after running, it instructs the administrator to check its work and confirm it had written out various files correctly. It turns out this was a needed step each time we ran it.

4 Responses

The above sections may sound like a large doom-and-gloom scenario, but in the end we were able to integrate Cielo into our environment in a way that is relatively easy for our administrators to pick up quickly. While we could have run the machine as a appliance-like system and not needed to deal with most of the challenges we discussed, we concluded that closer integration with our existing systems would help our small system administration team take on the system quickly after integration was complete. That made each of the challenges into actual problems and pushed us to find solutions for them.

4.1 Working Together

One of the most important things we did was keep a strong working relationship with Cray, our vendor. While it would be very easy for the clashes between our ideal system and their real-world products to result in deep fighting and animosity between the two groups, we were all able to keep a good relationship. Cray was eager to hear our concerns, fix problems, and submit idea cases when appropriate, and we were open to understanding their reasoning behind the design choices they made. We believe this is an important thing for both vendor and customer to keep in mind during a machine standup - both sides need each other, and keeping a good relationship is very beneficial in the long run.

Along with working closely with our vendor, we were careful to work closely across teams at LANL and Sandia. On the systems side, the Cielo bringup was a collaboration between HPC-5 (the system integrators) and HPC-3 (the system administrators) at LANL and the Cray support team at Sandia. Having these three teams work together gave us great power: we had the Cray experience from Sandia, the new system integration experience from HPC-5, and the long-term production system experience from HPC-3. While the nature of our environment made in-person collaboration the most useful form, we also made use of standard conference calls and email lists to keep each other in sync. The HPC-5/HPC-3 collaboration was especially important, as it made the transition from integration to production smooth and much less painless than a "throw it over the fence" model would have provided. Being able to work closely between all of the groups without chain of command overhead made it easy for us to make quick progress with the project.

4.2 Configuration Management

Another very important decision we made was to use a CM tool from the beginning. Although this could be seen as the source of several of our challenges, we would have had a much larger set of more difficult challenges without it. With Cielo, we ended up using a layered approach to managing the various parts of the system. Since the majority of the cluster is diskless, our final CM scheme had a small number of nodes that actually ran the Cfengine client: the SMW, the boot node, and the external login nodes. Everything else was managed by the sharedroot area (from the boot node) or the ramdisk images (from the SMW). With this design, we effectively had only a handful of Cfengine product areas to manage. This simplification made it easier to quickly grasp the design of the system and push out changes to the large number of nodes in the system.

Of course, after putting our CM system in place we still had a number of management tasks that required manual work. Most of these revolved around the monolithic install scripts mentioned previously - some of these were impossible to automate, while others just weren't worth the time.

For these we decided the best route was to document the exception and train new system administrators to recognize when they needed to do things by hand. In some cases, such as rebuilding the compute image ramdisk, we were able to have Cfengine print out a message after a successful run telling the administrator what more needed to be done by hand. In other cases we needed to rely on the carefully-maintained documentation wiki that the LANL administrators already use. By modeling the Cielo documentation off of existing documentation for other systems, we were able to fit these manual processes in to the mindset already known by the system administration team.

By implementing a complete configuration management scheme from the beginning, we were able to make several big changes to the system relatively quickly and painlessly. The first happened when we swung Cielo from our open network to our classified network: this required rebuilding the entire machine from scratch, which we were able to do in a matter of days using the configuration management system and documentation we had created. Later, we were able to quickly rebuild the system again when the upgrade from 72 to 96 racks required a large change in machine topology. Immediately after that, we made a quick upgrade between two Cray service packs that had caused problems at other sites with little trouble, mostly because we had a fully managed system and could recognize which system components had changed in incompatible ways. In short, our early effort has repaid itself several times over already.

4.3 Homegrown Tools

While bringing up Cielo, we found several system management deficiencies that weren't quite met by existing Cray management tools, but were too specialized to our environment to submit as a cases to Cray. Instead we wrote our own tools to fill in the gaps.

4.3.1 xtautorpm

Most of Cielo's compute and service nodes are diskless systems that use a ramdisk and an NFS-mounted read-only root filesystem to provide their operating system environment. The NFS filesystem provides system specialization of files through a layered approach, with the base filesystem being overlaid by *views* of node-specific files. These systems are managed by an interactive Cray tool named `xtopview` that handles package installation, file specialization, and other management tasks by presenting the administrator with a chrooted environment corresponding to the specialized view of each system or class of systems. This extra layer made our team's standard management methods difficult, as it is designed to be run interactively, only one person can run the `xtopview` utility at a time, and the utility has no provision for using tools such as `yum` to install packages.

To alleviate these restrictions, we expanded one of our already-existing tools under the name `xtautorpm`. This new tool automates acts as a layer between Cfengine and the `rpm` command, giving Cfengine the abstraction needed to use `xtopview` directly. With this extra layer of abstraction, we made the package installation procedure identical to that on our other clusters without losing the support of the vendor supplied tools.

4.3.2 xtfixdefault

As mentioned earlier, Cray uses several software version management schemes on their systems. We found it time consuming to manually manage both the `modules` environment (which we understood well) and the "default links" system that Cray introduced. To prevent version skew, we wrote a tool named `xtfixdefault` to keep the two systems synchronized. Since we were already familiar with the `modules` system, we decided to use it as the base for our versioning. When Cfengine runs `xtfixdefault`, the utility checks all of Cray's default links and confirms that they are pointing to the same software versions as the `modules` environment's default version. When run interactively, the tool can also be used to update the modulefile from the default link and report on which modulefiles and default links are not the same. With this one utility we can both enforce our will over the software versions with Cfengine *and* report changes performed by Cray's monolithic software installers. This utility has made software updates much less time consuming.

4.3.3 ethcfg

The file specialization provided by the `xtopview` command is generally used at a class level to cover a large group of service nodes at once or at the individual node level to make one node stand out from the others. Both of these cases are simple and straightforward to manage. However, there is one specialization case that requires every node to have its own file: the static network management files. Standard configuration management systems avoid the need for hundreds of node-specific files by using templates, DHCP, or other similar solutions, but these did not fit the Cray model well. Instead we wrote a simple-but-powerful init script dubbed `ethtool` that configures the nodes' network interfaces by reading a flat configuration file at boot time. This file contains the network interface information for all of the service nodes in the system, meaning it can live in the default overlay view and requires no specialization for each node. The number of nodes included in the file is small enough that we found no performance problems with a flat file, giving us ease of maintainability over a more complex system using something like SQLite.

5 Lessons

After bringing up Cielo, we were able to put together a few lessons we learned along the way.

Keep good relations with your vendors : It is all too easy for vendor relations to break down when you don't see eye to eye with them. Keeping a good relationship makes it much easier to keep all sides progressing throughout the project.

Get test systems early : Although they were only mentioned briefly at the beginning of this report, our three smaller systems (Smog, Muzia, and Cielito) were instrumental in getting us experience with Cray's way of doing things early. When building a new system, getting access to representative hardware early in the process fills the knowledge pipeline much faster.

Use configuration management, even if it takes effort : The upfront cost of configuration management is easier to see than the long-term gains, but those gains are real. Whether you should work to fit a system into an existing CM scheme or not is a site-specific question, but using some tool is the best choice in any complex case.

When standing up a system of a new design, plan for "Murphy Time" : Murphy's Law will assert itself as often as it can, especially with new systems. Be ready for that. Finishing early is much more impressive than finishing late.

Work as a team : Today's systems are too complex for one person to fully understand. There are too may pieces: hardware, software, networking, filesystems, system management, and the list goes on. Working as a team is important for sharing responsibilities and areas of expertise with a new system and for keeping everybody interested in the project. Resist the urge to designate "the guy that knows it all", as he will inevitably win the lottery and leave the group.

6 Conclusions

As we stated in the beginning, standing up a new HPC resource is a complex task. While integrating Cielo, we ran into an expected breadth of challenges: managing the vendor/customer relationship, working with integrating an appliance-like system into an already-existing environment, designing a configuration management system around an imperfect software distribution design, and other more minor challenges. In our case these were all framed within the desire to make the system behave similarly to an already extensive set of HPC resources, a requirement from both the user and administrator points of view.

We were able to respond to these challenges with a combination of technical and social solutions involving, among more minor solutions, a close working relationship between our vendor and local teams, using strong configuration management and careful documentation when appropriate, and writing custom tools to fill in gaps as needed. The combination of solutions we found kept us flexible enough to make good decisions each time while getting the work done in the needed timeframe.

In the end, we were able to put together a short list of lessons that we thought were important from our experience. On the top of that list was the need to keep strong working relationships with all of the groups involved. Closely following this was the need for configuration management from the beginning. The list was rounded out with other lessons that are obvious in hindsight, but easy to lose track of in the heat of getting work done.

The final result of the work described in this report is a very manageable system. Like all systems of Cielo's complexity, there will always be work to be done, but we have a strong foundation on which to continue building and we are confident in the work we have done to integrate it into our environment.

References

[1] Cfengine. `http://www.cfengine.org/`.

[2] Modules – Software Environment Management. `http://modules.sourceforge.net/`.

[3] S. Kemp. Using the Debian alternatives system. `http://www.debian-administration.org/articles/91`.