Petascale System Management Experiences

Narayan Desai, Rick Bradshaw, Cory Lueninghoener, Andrew Cherry, Susan Coghlan, and William Scullin – Argonne National Laboratory

ABSTRACT

Petascale HPC systems are among the largest systems in the world. Intrepid, one such system, is a 40,000 node, 556 teraflop Blue Gene/P system that has been deployed at Argonne National Laboratory. In this paper, we provide some background about the system and our administration experiences. In particular, due to the scale of the system, we have faced a variety of issues, some surprising to us, that are not common in the commodity world. We discuss our expectations, these issues, and approaches we have used to address them.

Introduction

High-performance computing (HPC) systems are a bellwether for computing systems at large, in multiple regards. HPC users are motivated by the need for absolute performance; this results in two important pushes. HPC users are frequently early adopters of new technologies and techniques. Successful technologies, like Infiniband, prove their value in HPC before gaining wider adoption. Unfortunately, this early adoption alone is not sufficient to achieve the levels of performance required by HPC users; parallelism must also be harnessed.

Over the last 15 years, beowulf clustering has provided amazing accessibility to non-HPC-savvy and even non-technical audiences. During this time, substantial adoption of clustering has occurred in many market segments unrelated to computational science. A simple trend has emerged: the scale and performance of high-end HPC systems are uncommon at first, but become commonplace over the course of 3-5 years. For example, in early 2003, several systems on the Top500 list consisted of either 1024 nodes or 4096-8192 cores. In 2008, such systems are commonplace.

The most recent generation of high-end HPC systems, so called petascale systems, are the culmination of years of research and development in research and academia. Three such systems have been deployed thus far. In addition to the 556 TF Intrepid system at Argonne National Laboratory, a 596 TF Blue Gene/L-based system has been deployed at Lawrence Livermore National Laboratory, and a 504 TF Opteron-based system has been deployed at Texas Advanced Computing Center (TACC). Intrepid is comprised of 40,960 nodes with a total of 163,840 cores. While systems like these are uncommon now, we expect them to become more widespread in the coming years.

The scale of these large systems impose several requirements upon system architecture. The need for scalability is obvious, however, power efficiency and density constraints have become increasingly important in recent years. At the same time, because the size of administrative staff cannot grow linearly with the system size, more efficient system management techniques are needed.

In this paper we will describe our experiences administering Intrepid. Over the last year, we have experienced a number of interesting challenges in this endeavor. Our initial expectation was for scalability to be the dominant system issue. This expectation was not accurate. Several issues expected to have minor impact have played a much greater role in system operations. Debugging, due to the large numbers of components used in scalable system operations, has become a much more difficult endeavor. The system has a sophisticated monitoring system, however, the analysis of this data has been problematic. These issues are not specific to HPC workloads in any way, so we expect them to be of general interest.

This paper consists of three major parts. First, we will provide a detailed overview of several important aspects of Intrepid's hardware and software. In this, we will highlight aspects that have featured prominently in our system management experiences. Next, we will describe our administration experiences in detail. Finally, we will draw some conclusions based on these experiences. In particular, we will discuss the implications for the non-HPC world, system managers, and system software developers.

System Background

The Blue Gene architecture is unusual among computing platforms. It is a completely integrated system, including computational resources, management infrastructure and multiple networks that interconnect them. Much of the hardware and software used in these systems are purpose built. Both of these approaches are at odds with the prevailing trends in commodity clusters. We will provide basic background relating to system management on Blue Gene systems; more detailed information can be found elsewhere [1]. In this section, we will provide an overview of Intrepid's hardware configuration. From there will proceed to a description of relevant system architecture details. Finally, we detail the control system, a nerve center for system management. The control system is the setting for all administration on the compute complex.

Intrepid Configuration

Intrepid is a 40 rack Blue Gene/P system with a pset size of 64. This system is comprised of nearly 160,000 cores, with a peak performance of 556 TF. The compute complex is managed by a single service node. A 10 gigabit data network connects I/O nodes in the compute complex with 136 file servers and several other storage resources. This network is comprised of 512 port switches assembled in a non-blocking configuration to provide 1100 client ports. The system includes 17 Data Direct 9900 storage arrays. Each array is connected to eight fileservers via direct-connected Infiniband; each can provide fail over support for the other seven. This architecture is shown in Figure 1. The service infrastructure is modestly sized, and almost exclusively uses traditional hardware and software.

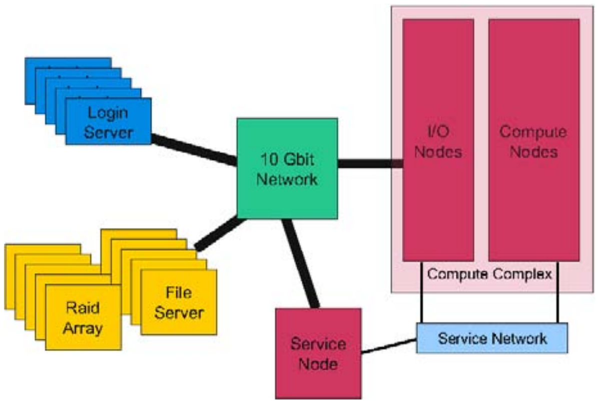

Figure 1: Intrepid system architecture.

Intrepid is a capability HPC system, operated as a part of the US Department of Energy INCITE program [2]. This means that it is intended to run large-scale jobs. Individually, these jobs often consume a large portion of the machine. It is common to see 32,768 core and 65,536 core jobs on Intrepid, occasionally at the same time.

System Architecture

Blue Gene/P compute nodes use quad core PowerPC 450 processors, with access to multiple specialized networks. Compute jobs have access to a mesh network that can be connected into a 3-D torus for collective communication. Also, there is a tree network used to connect compute nodes to dedicated I/O nodes. I/O nodes are connected to fileservers using 10 gigabit ethernet. All compute and I/O nodes also have access to a dedicated management network. This network is used for node booting, diagnostics, and monitoring.

The system is partitioned for each user job. Each partition contains a distinct set of compute and I/O nodes, and the associated mesh network resources. The mesh network must be built into a large rectangular solid. This limits the valid combinations of node resources. Provided sufficient resources are available, a mesh network can be wrapped into a full 3-D torus, providing better network performance. Each of these constraints limit the ways that the system can be allocated, and provide a complicated set of variables when debugging application problems.

The BG/P Control System

The BG/P control system has three main tasks: partition control, job control and system monitoring. It is a single, monolithic process that is run on the system service node. It stores large amounts of data into a DB2 database, run on the same service node. In this section, we will describe each function in turn.

As we mentioned earlier, partitions are the entities where jobs can be run. The architecture of the system depends on a detailed understanding of the topology of the constituents of a partition. This means that partitions are typically rebooted between jobs, upon either user or partition changes. For example, if two jobs were running on a small pools of resources, and these pools needed to be joined for a new job, all of these resources would need to be rebooted before job execution could occur.

The control system plays three main roles in the partition allocation and boot process. It stores and validates the partition configuration, it reserves that hardware used in an active partition, and it implements the partition boot process. The first two of these are straightforward data management issues, while the partition boot process is a little more subtle.

During partition boot, a partition-specific OS image is served to all nodes. This OS image contains all configuration data needed to properly configure the system. The service network internal to each pset implements a hardware broadcast, so the service node need only send the OS images to each node card in the partition; from there it can be broadcast to all nodes. Simultaneously, the control system serves a different set of OS images to the I/O nodes in the partition. When the boot process is complete, each compute node stands ready to execute a user application, and all I/O nodes have mounted all filesystems that can be used by a user job. Also, all I/O nodes run a compute node I/O daemon, *ciod*, that is responsible for executable loading and the proxying of I/O for user jobs.

Once node boot has completed, the user's job executable is loaded, via the ciod. The control system sends the user executable to each I/O node in the partition, which, in turn, loads the user executable on each compute node. Once a job starts on the partition, the ciod begins its main task of proxying I/O requests. When a user process on a compute node performs an

I/O related system call, this call is packaged up and sent to the I/O node, where the ciod performs it and sends the request back to the compute node.

The final responsibility of the control system is to monitor the service network for RAS events. Compute and I/O nodes can directly issue these events to the low-level service network. These events signal a variety of problems, including correctable and uncorrectable hardware errors, and some software errors. The data available from this interface substantially improves the visibility into the compute complex.

Service Infrastructure Management

The service infrastructure is a fairly standard set of systems, both in terms of hardware and software. All of the hosts run a standard version of Suse Enterprise Linux and are managed using a set of standard management tools. Configuration management is nicely handled by Bcfg2 [3], while Nagios [4] is used for monitoring. The service infrastructure is only about 200 systems, and doesn't pose any scalability problems. Most of those 200 or so systems are fileservers, which have a standardized set of software and services.

The service infrastructure is largely run of the mill. This highlights another unique aspect of this system. On typical large scale systems, standard management tools have reach across the whole system. On Intrepid, the combination of specialized and commodity hardware and software make this impossible. Different software and methods must be used on the different parts of the system. This poses a cross-training and coverage problem.

Experiences and Observations

We took delivery of our first BG/L system in January of 2005. This system was comprised of a single rack, with 1024 dual-core nodes. This system was run with a combined set of research and production computation goals and was retired mid-summer 2008. We found this system to be robust and popular with users; these good experiences provided the justification for Intrepid and other associated smaller BG/P systems.

During the summer of 2007, we got initial access to prototype Blue Gene/P systems at IBM. These systems, still under development at the time, were used to port software, both applications and system software, to the new architecture. We also used this opportunity to gain experience with the new control system and runtime environment.

In October of 2007, we took delivery of our first batch of BG/P racks, with delivery and assembly continuing until December. Friendly users have been present on the system since. At the time of this writing, a small development system, Surveyor, is open for general use, while the larger system, Intrepid has eight compute node racks in production and thirty-two more in "early science" mode. These last thirty-two racks are in the process of transitioning to production.

The administrative team of this system is fairly seasoned; many are veterans of other large HPC centers. In anticipation of Intrepid, we each had concerns about different potential system management issues. Many of these concerns focused around standard concerns in large systems: scalability, fault tolerance, and general robustness. To our surprise, several of the issues that we expected to be problematic were handled well by the system, while others proved to be difficult issues to handle. We will discuss several of these in detail throughout the rest of this section.

Control System Issues

As we mentioned earlier, we had expected to encounter scalability issues with the Blue Gene control system. Each BG/P system is managed by a single instance of the control system running on a single service node. Running a system the size of Intrepid with a single control system seemed to be pushing the limits; we had anticipated trouble. On the contrary, control system scalability, thus far, as not been an issue for us. Instead, control system serviceability has been a bigger issue.

Due to the system's reliance on a single instance of the control system, control system restarts are catastrophic; the system must be drained of jobs and no new jobs can be started until the restart has completed. We encountered several bugs in the control system early in the deployment process that caused critical failures, requiring a restart control system to reclaim resources. While these issues have been fixed, we are still vulnerable to this behavior when control system problems occur. When the control system goes south due to unexpected system behavior, the entire system must be idled.

The control system workload has three major components: partition booting, RAS messaging, and state management. Increases in system size have the expected effect on the control system workload; both booting and RAS messaging grow with the system size. Job size and length also have a dramatic impact on the control system workload. The boot process scales non-linearly (in a good way) with the number of nodes, so a single large job boots more quickly than two half-sized jobs. Job length controls how often the boot process and job setup and teardown occur, so longer jobs are gentler on the system.

In this section, we discuss issues related to control system scalability and serviceability. We proceed through the three major areas of the control system, describing our workload, decisions we have made and their combined impact.

Node Boot

One of the main tasks performed by the BG/P control system is providing the node boot infrastructure. The boot process on BG/P systems is highly parallelized, and hence scalable. However, this scalability comes at a cost. The boot process consists of a variety of tasks operating in parallel; any of these can fail. The robustness of the boot process has been improved through the use of simple mechanisms for system boot; due to the integrated nature of this system, standards like PXE need not be applied. Instead, the control system uses a Joint Test Action Group (JTAG) network to load kernels on the compute complex. This network provides low-level access capabilities to individual hardware components in the compute complex. It has a limited hardware broadcast capability which greatly improves the scalability of the boot process.

After the kernels have been loaded onto the compute complex, the compute nodes boot a custom lightweight kernel, called the CNK (compute node kernel). The I/O nodes boot Linux, start services consumed by compute nodes, bring up external networking, and mount all filesystems. A majority of boot problems occur when the compute complex is interfaced with the external service infrastructure. In particular, these problems happen during network and filesystem bringup.

As we mentioned earlier, due to limitations in the BG/P runtime system, nodes must be rebooted between jobs if their partition configuration changes. In light of this restriction, and other factors discussed later, we have opted to reboot nodes between each job. Due to this operational decision, partitions are rebooted in advance of each job. The boot process is sufficiently quick to make this convenient; a 16 rack (64K task) partition boots in less than 5 minutes. This performance is a marked departure for the boot process on commodity systems, where network boots are considerably slower and more fragile.

Despite this relatively low cost, this approach has other tradeoffs. A higher load is placed on the control system, due to the increased frequency of node boots. Also, while the boot process itself is scalable, I/O and network related problems do occasionally occur during the boot process. An increased frequency of node boots increases the possibility of this occurrence.

On the plus side, per-job node reboots reduce the ability of latent state to impact jobs; the system starts from a clean slate for each job. This makes job performance consistent and reproducible. Moreover, it minimized the occurrences of idiosyncratic software problems. On the balance, this decision has been the right one; we are convinced this policy has allowed us to avoid far more problems than it caused.

Debugging boot problems requires a wide view of the activities of all of the components included in a partition. Failures fall into two basic categories: single component failures and workload dependent failures. Single component failures, of course, are far easier to diagnose that workload dependent failures; however, even single component failures can be difficult to locate.

The most frequent sort of single component failures cause network or filesystem failures on I/O nodes. I/O nodes use a simplified Linux boot process that runs a series of configuration scripts. The boot process is fundamentally serial; I/O nodes independently start up, and are not made aware of failures on other nodes. Even if failures can be locally detected, they are not adequately communicated to other components in the system. Direct access to the RAS system is not conveniently available from this context, however, enhancements in this area are forthcoming.

Load-related errors can also cause issues on the network during large-scale partition boots. Because I/O nodes are rebooted along with compute nodes for each job, the network must be able to properly cope with large numbers of nodes frequently leaving and joining the network. All the while, I/O is being performed for other active jobs. Workloads of this sort have triggered a variety of switch firmware bugs, all of which have been difficult to replicate. When these problems occur, they are painful to troubleshoot and resolve. We currently have no mechanisms that help us with this process.

RAS Messaging

The BG/P control system includes a scalable logging framework for RAS events. These events are generated in a variety of conditions corresponding to hardware and software failures. When software failures occur, they frequently occur across an entire job. Because large jobs are common on Intrepid, these errors can cause up to 160,000 RAS messages per incident. This count corresponds to a RAS event per task on a full-system job. Even a more modest job size could easily result in upwards of 32,768 events for a single error. The control system also stores a variety of data about system hardware and current and historical jobs.

This volume of logging data cannot be retained indefinitely. Even though Intrepid has been in operation for less than a year, we already face difficult data retention policy questions. The amount of stored data has a direct impact on the performance of the control system; as more data is added, queries take longer. Ideally, data would be retained forever, to allow for long-term trend analysis; this is clearly not possible.

The introduction of a data-intensive system like this into system management is an abrupt departure from the usual approach used on HPC systems. We were frankly unprepared to deal with data and transaction volumes at this level. In the last few months, we have added a dedicated DBA to the system management staff; this appears to be improving the situation.

The scalability of the RAS messaging infrastructure itself has not been a problem; during acceptance tests we were able to successfully simulate intense storms of RAS messages without issue. The system is even responsive during these situations.

The RAS subsystem is a critical resource for debugging system issues, though the volumes of data involved make this task somewhat difficult. These issues are discussed in detail in Section RAS-based Debugging.

Partition State Management

The final role of the BG/P control system is to track compute system states through the partition and job lifecycles. This process is scalable, however, it is the largest current source of serviceability problems. In some cases, partitions end up in a state where they cannot be reclaimed. When this occurs, only a full control system restart can restore the partition to operation. In order to perform a control system restart, the machine needs to be drained of all jobs, resulting in diminished utilization. This problem has posed the largest issue of all of the control system issues mentioned in this section. While IBM has been quite responsive, and bugs have usually been fixed relatively quickly, the potential for trouble remains. This is the main detriment caused by the control system's implementation as a single scalable component.

Fault Management

Large scale systems are more sensitive to hardware failure than decoupled systems. This sensitivity occurs due to the properties of parallel workloads; many HPC applications will fail outright upon single component failure. Moreover, the increase in component counts in large-scale systems cause failures to occur more often.

Some of the fault management work on Intrepid consists of traditional component diagnosis and replacement, however, the scale and workload of the system makes failure isolation difficult. Because of this, more sophisticated techniques are to discover failing components.

Intrepid also has a number of useful features for fault management. The compute complex ships with comprehensive hardware diagnostics. The RAS system provides detailed information about unexpected hardware and software events at runtime.

In this section, we will discuss role of system diagnostics in system management operations, the processes employed between runs of the diagnostic system and use of the RAS system in debugging.

System Diagnostics

The Blue Gene system ships with a set of diagnostic routines. These tests verify the proper function of all components in the system. While the full set of tests are quite comprehensive and accurate, they are quite time consuming to run. A single rack full diagnostics takes on the order of an hour to run diagnostics and currently a maximum of two racks may be run in parallel. The cost to run regular full diagnostics on a system this size is simply unreasonable at this point. In order to guide regular diagnostics, IBM has developed a smaller, general health check that can be run in parallel across the full 40 rack system in a reasonable amount of time (under two hours). This general test can call out problem areas which can later have a full diagnostics test suite run against them.

These diagnostics greatly improve our lives; at the same time, they present us with a difficult choice. Diagnostics are able to discover marginal hardware before it has failed outright, so we benefit from frequent diagnostics. On the other hand, diagnostics monopolize system resources, reducing the number of cycles delivered to users. This issue is similar to the one faced by users choosing a checkpoint frequency in parallel jobs [5].

The addition of the general health check capability and more accurate assessments of hardware failure rates will allow us to pick a reasonable frequency for system diagnostics. That said, hardware failures will always occur in the intervals between diagnostic runs, so manual troubleshooting techniques are also needed.

Diagnostic Search

Regardless of the quality and frequency of system diagnostics, unexpected failures will still occur. When they do, users report erroneous system behavior. This behavior must be diagnosed by the system administrators. Parallel systems and workloads are problematic in this regard. Large numbers of components operate in parallel to accomplish a given task; a single failure in this context frequently causes overall process failure.

Much of the BG/P's scalability results from aggregation. While aggregation is good for scalability, it can have a detrimental effect on diagnostic procedures. Grouping individual components into parallel, aggregated process often obscures the source of failures. Incorrect collective behavior is observed, but that alone does not provide enough data to isolate the cause.

In this case, we use a binary search of system hardware to isolate the failing component. Luckily, most tasks can be run on partitions of arbitrary size, so we can use the same test on smaller partitions until a cause emerges. This technique is quite effective; it provides a fast path to isolate a faulty midplane. Once a midplane is isolated, system diagnostics can be performed without affecting other midplanes. This combination of search processes and system diagnostics has been an effective tool for faulty hardware isolation, even when problems occur in large groups of components.

RAS-based Debugging

Problem identification is substantially harder on Intrepid due to the system architecture and workload. Many operational issues on Intrepid do not result in a single log message that describes the issue. Rather, problem identification consists of the correlation of log data from a variety of locations. In addition to RAS events and control system data, many other components in the system also log information in a variety of formats. I/O nodes and service infrastructure nodes run Linux and produce standard logs.

Identification of problems in real-time or even near-real-time requires the correlation of events from a large number of sources potentially entering the system at a high rate. While several correlation frameworks exist and are publicly available [6, 7], correlation at this scale is an open research issue. Our current approaches to these problems use small volumes of data, heuristics, and intuition, to a reasonably good effect. However, progress on this front would greatly improve our ability to recognize subtle problems in a more timely manner.

Management Effort

Intrepid is administered by a team of three full-time system administrators and a DBA. Assistance from others provides for off-hours coverage and user support. Two of these administrators are primarily concerned with the compute complex and control system, while the other manages the service infrastructure. This division is purely practical; the service infrastructure requires much more daily administration effort, scaled for system size, than the compute complex does.

Considering the relative sizes of the systems, this ratio of administration effort is surprising; the 40960 node compute complex only takes twice the operation effort as a 200 node commodity system. We have determined a variety of factors that play a part in the decreased administration costs of the compute complex. In this section, we will contrast each of these with the traditional model used in commodity systems.

Persistence

One major difference between the service infrastructure and the compute complex is the workload. The compute complex is a consumer of services, and is frequently rebooted. On the other hand, the service infrastructure provides all services consumed by the compute complex. In particular, the compute complex interacts with the control system and several file systems served from storage nodes.

Frequent reboots of the compute complex, as new jobs are run, minimize the amount of problematic state that can be accumulated. This approach is not possible on the service infrastructure as the services provided must persist beyond a single job. This need for continuity makes the service infrastructure far more difficult to administer than the compute complex.

Configuration Complexity

Each half of the system uses a different configuration methodology. When a partition is booted, the same OS image is sent to all compute nodes; a (different) single OS image is sent to all I/O nodes as well. Each of these OS images contains the union of all configuration data needed for all nodes, yet still remains quite small. Frequently these images are less than 16 MB.

By contrast, commodity systems running standard distributions of Linux have a higher configuration requirement. Bcfg2 [3] configurations for Linux systems configured in the service infrastructure contain information about nearly 800 aspects of system configuration; these configuration specifications can be upwards of a megabyte in total. This difference is quite striking; the configuration burden on compute nodes is substantially lower than that of commodity systems.

The reduced complexity of compute complex configurations has both positive and negative repercussions. The compute complex is substantially less agile than we, as administrators, have become accustomed to with commodity systems. In an HPC workload, with relatively small numbers of large jobs, this shortcoming has not yet posed a serious problem to us. We anticipate this to become more of an issue as sites start to use BG/P-style systems for varied workloads.

Moreover, for the reasons described above, all configuration management must be performed incrementally on the service infrastructure, due to service continuity requirements. Incremental reconfiguration processes are clearly more error-prone and resource intensive. Incremental approaches have a much higher burden in terms of compliance monitoring. Due to the frequency of partition reboots in the compute complex, incremental approaches are not needed as drastically.

Hardware Robustness

The compute complex hardware is substantially more robust than even server-grade commodity hardware. The addition of robust diagnostics make the operation of the compute complex considerably easier than the service infrastructure, or any other commodity system. While we face issues in isolating failing hardware in the compute complex, this process still takes less time than maintenance activities in the service infrastructure. As purely anecdotal evidence, we have replaced as many compute nodes as we have serviced fileservers during system operations. This demonstrates the drastic difference between the failure rates in the different parts of the system.

Conclusions

In this paper, we have provided a discussion of our experiences managing Intrepid, a large-scale Blue Gene/P system. Our main operational issues focus around serviceability and fault management. One striking finding was the relative level of effort required to operate the halves of the system. Due to the capabilities provided by the BG/P control system and RAS infrastructure system management efficiency is drastically higher in the compute complex compared with the service infrastructure. For this reason, we anticipate that design choices similar to those made by the BG/P design team will become common in other systems as well. System administrators will have no choice but to cope.

Applicability to Other Environments

System sizes have been on the rise for the last two decades and this trend shows no sign of slowing. Meanwhile, the relative costs of administration and maintenance of these systems has continued to grow.

The relatively low operations cost and high scalability of integrated systems like the IBM Blue Gene/P or other integrated MPP systems make them appealing for wider scale use. However, these benefits come at a substantial complexity cost. Nonetheless, IBM has begun to adapt Blue Gene/P systems to non-HPC workloads [8].

Virtualization and cloud computing have become popular recently. Large scale systems, whether composed of real hardware or virtual nodes pose many similar problems in administration. These systems, once they grow to a sufficient size, will experience many of the same management and problem determinations issues that we have seen on Intrepid.

Consolidated systems such as these will introduce complex interdependencies between components and even between nodes in some cases. I/O starvation has long been a problem in the HPC space. It has become a pressing issue in virtualized environments as well. As the number of entities competing for resources in virtualized environments grows, administrators must be able to diagnose multi-system, workload-dependent issues in an effective way.

Moreover, administrators are frequently tasked with the operation of systems of increasing size without corresponding increases in staffing. These issues will be exacerbated in virtual environments, where new virtual machine instances are effectively free.

Implications for System Software and Tools

Our experiences have a variety of implications for system management tool developers. Scalability has long been the the boogy-man of the system software world. Perhaps it isn't as frightening as everyone has been assuming. Certainly there are cases where scalability is a dire concern, but this is not universally the case. A more nuanced understanding of service scalability is needed.

At the same time, in some cases, scalable operations are required. These mechanisms cost. For example, the boot process uses a hardware broadcast capability in order to scale. In terms of configuration, this approach is a step backwards to the days of shared NFS root file systems. In this case, we have lost a substantial amount of flexibility, compared to modern Linux systems.

Data analysis is much more difficult than we had initially anticipated. Realtime analysis of our volume of data is not possible with the current generation of publicly available analysis tools such as Sec [6]. We are unsure that commercial tools will scale to this level, although large scale web service providers probably have home-grown tools in this space. This is one area where a scalable, most likely parallel, approach will be needed.

Collective approaches to system management need to be developed. While some initial work in this area has occurred [9, 10], collective approaches to system management have not yet become convenient enough to deploy in practice. For example, a reasonable distributed control mechanism could be used to provide fine-grained configuration management capabilities in a scalable fashion. Similarly, distributed approaches to data correlation could provide the scaling needed on systems of this sort.

Finally, much of the configuration complexity present in traditional systems have been eliminated outright from this architecture. In some cases, we find functionality missing, however, on balance, we have found the system quite usable. Might we be near the end of our collective configuration nightmare?

Author Biographies

Rick Bradshaw is currently a Senior Systems Administrator for the Mathematics and Computer Science Division of Argonne National Laboratory. His interests are mainly focused around configuration management, and experimental and HPC computing. He holds a bachelors of Computer Science from Edinboro University of Pennsylvania, and is currently working on a Masters in Computer Science at the University of Chicago.

Andrew Cherry is a UNIX system administrator who has worked on a variety of systems, in both commercial and non-commercial environments. He is currently an HPC system administrator at Argonne National Laboratory, where he is responsible for day-today management of Argonne's Blue Gene/P supercomputer.

Susan Coghlan has worked on parallel and distributed computers for 20 years, from developing scientific applications, such as her work on a model of the human brain at the Center for NonLinear Science at Los Alamos to managing ultra-scale supercomputers like ASCI Blue Mountain, a 6144 processor supercomputer at Los Alamos National Laboratory. In her current role as Associate Division Director and Director of Operations for the Argonne Leadership Computing Facility, she is responsible for the installation and operation of the world's fastest open science computer (Top500, June 2008) the ALCF's 557TF Blue Gene/P production system. She is well known within the HPC community, and has presented numerous tutorials, lectures, and papers on her work. When not fiddling with some of the world's largest computers, she does so with other Irish traditional musicians.

Narayan Desai is a researcher in the MCS Division of Argonne National Lab. He specializes in system software, particularly as it relates to system management, fault tolerance and scheduling.

Cory Lueninghoener is an HPC System Administrator with the Leadership Computing Facility at Argonne National Laboratory. When not keeping a 40-rack BlueGene/P system running, he spends time

hacking at tools like Bcfg2. Cory can be reached at lueningh@alcf.anl.gov .

William Scullin is a HPC systems administrator at Argonne National Lab's Leadership Computing Facility. His favorite systems administration tools after Bcfg2 and BlueGene Navigator are macromancy and python, though that may be repetitive. Outside of work, he seeks to raise awareness of tyrophagia.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

This work for the Argonne Leadership Computing Facility was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Bibliography

[1] Gara, A., M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L System Architecture," *IBM Journal of Research and Development*, Vol. 49, Num. 2-3, pp. 195-212, 2005.

[2] US Department of Energy, *Innovative and Novel Computational Impact on Theory and Experiment (Incite) Program*, http://www.er.doe.gov/ascr/incite/index.html .

[3] Desai, N., *Bcfg2 web site*, Argonne National Laboratory, http://trac.mcs.anl.gov/projects/bcfg2 .

[4] *Nagios web site*, Nagios Enterprises, LLC, http://www.nagios.org .

[5] Oliner, A. J., R. K. Sahoo, J. E. Moreira, and M. S. Gupta, "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems," in *IPDPS*. IEEE Computer Society, 2005.

[6] Rouillard, J. P., "Real-time Log File Analysis Using the Simple Event Correlator (Sec)," *LISA*. USENIX, pp. 133-150, 2004.

[7] *Splunk web site*, Splunk, Inc. http://www.splunk.com .

[8] Appavoo, J., V. Uhlig, and A. Waterland, "Project Kittyhawk: Building a Global-Scale Computer: Blue Gene/P as a Generic Computing Platform," *SIGOPS Operating System Review*, Vol. 42, Num. 1, pp. 77-84, 2008.

[9] McEniry, C., "Moobi: A Thin Server Management System Using Bittorrent," *LISA*, USENIX, pp. 253-260, 2007.

[10] Desai, N., R. Bradshaw, A. Lusk, and E. Lusk, "MPI Cluster System Software," *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Springer Lecture Notes in Computer Science, D. Kranzlmuller, P. Kacsuk, and J. Dongarra, Eds., Num. 3241, Springer, pp. 277-286, 2004.