

Systems Management Software for Extremely Large and Dynamic Computing Systems

(Formerly “High-Performance Computing Systems Research”, FWP #56692)

Rémy Evard, evard@mcs.anl.gov, PI
Rick Stevens, stevens@mcs.anl.gov, Co-PI

1 Summary

We propose a program of research in support of the design and implementation of system management software for future large-scale parallel computers. Much of the work will be applicable to (and tested on) existing large-scale clusters, but the target systems for this research are petascale computers.

The first petascale systems are expected to be feasible within six years. It is anticipated that in order to support these systems, regardless of their specific architecture, system software must increase in capability and scale by three orders of magnitude. Today’s approaches to systems management barely scale to terascale systems. Research into new methods of systems management are critical to the successful deployment of petascale systems.

The research proposed here will focus on configuration and software management, the management infrastructure of systems, the ramifications of supporting heterogeneous capabilities, and the ability to scale to a system size several orders of magnitude larger than today’s systems. Our work will be released in a toolkit of open-source programs that demonstrate capabilities and that are used to support systems management on real clusters.

2 Background

Systems software is the software needed to manage and operate computing systems and to support applications. This is an expansive set of software, including areas such as kernels and operating systems, file systems, schedulers, management systems, and monitoring mechanisms.

Much research during the past decade has demonstrated that the primary barrier to achieving systems scalability is scalability of systems software. Researchers investigating paths to petaflops-capable systems in the early and mid-1990s identified multiple possible hardware technology paths to petascale performance. Each of these hardware paths, regardless of the technology base, had one thing in common: the need for large degrees of concurrency in future systems. Little’s law ($c = bw \times l$) relates concurrency to bandwidth and latency, where bandwidth is proportional to floating-point performance. It suggests that all future systems, whether they target teraflops, petaflops, or higher, will need to support increased concurrency even as bandwidth increases, because latency is fixed by the speed of light and by the process technology used for the components.

The first petascale systems, which are expected to be feasible within six years, will most likely build on innovations in cluster based systems technology, CPU microarchitectures, and custom-designed memory and processor integration. These computers likely will have millions of individual systems components that must be managed by systems software. Current systems software can handle, at best, thousands of system elements. Thus, in the next five years, systems software must increase in capability by at least three orders of magnitude.

The work described in this proposal focuses on the systems management portions of systems software, as described in detail in lower sections. Effective and scalable systems management is critical to the petascale effort. It must be possible to build, configure, and manage petascale computers. The approaches that we have today barely scale up to teraflop systems, and petascale systems will be much more complex.

This analysis of the likely path to petascale systems does not necessarily assume that petascale systems will look exactly like today’s common clusters, i.e. systems that have a single point of control with fully-capable nodes. Alternative models are also being explored and all are likely to play a part in the goal of petascale functionality. These include:

- Grid-based systems, in which many distinct systems are loosely connected in order to support a collective operation.

- Augmented-node systems, in which a portion (or a majority) of a system is optimized to support specific operations. Examples of such computers would be PIM-augmented systems or the IBM Blue Gene project.
- Minimal-node systems, in which the “backend” of the cluster is highly tuned to optimize performance and reduce demands on the system. The Sandia Microkernel project, among others, is exploring this approach.
- Single-system images, in which the details of the backend nodes are completely hidden. The “bproc” system under development by Scyld Computing and LANL is an example of this approach.

Regardless of the actual path to petascale functionality, systems management remains a key issue. All of the above models introduce some degree of heterogeneity into a system, which increases the complexity of management. When a particular approach reduces the complexity of management in one part of system (such as the backend nodes), it increases the complexity somewhere else (such as the management nodes for the backend nodes). Particularly at large-scale, the difficulties of systems management will not be removed by a change in the computing model, although the emphasis may shift from one aspect of it to another. Thus, while the work here is described in the context of a single cluster with fully capable nodes (because such a model is the most general), the results will be applicable and critical to any of the possible paths to petascale computing.

3 Expected Benefits

The work described here will have two primary benefits.

First, the results of this work will improve the community-wide understanding of the role and capabilities of systems management on very large systems. The first petascale systems will not be built by individuals or single institutions, but will emerge from a community effort. The research proposed here will advance that cause.

Secondly, all work described here will be carried out in the context of real systems implementation. This approach is necessary in order to ensure that we are tackling and solving real problems. The software created as a part of this work will not be written for general use or hardened for operation in a production environment, but will instead be focused on demonstrating capability and for use as a platform for exploration. All of the software that we develop will be released as a part of the “City” toolkit, thus allowing others to use these tools or to build on them. As in the past, we anticipate that the basic tools themselves (or the concepts demonstrated) will be incorporated into other efforts that will package them for widespread deployment or use concepts from them in other tools.

4 Proposed Work

Here we describe in detail the research issues we have identified and the approach we intend to take to address them.

4.1 Configuration Management

Every component of a system – the node operating systems, the management computers, the file systems, the schedulers, the installed software, and so on – must be configured to support operation. Each of these components usually includes subcomponents also requiring configuration. For example, the operating system on a node of a cluster often includes hundreds of software packages, each of which must be configured for that node to operate correctly. The collective management of all of these configurations, and the ability to change them all over time as necessary, is referred to as configuration management.

The field of systems administration has long recognized configuration management as a challenge. Currently, large-scale systems handle configuration management by a series of ad-hoc and site-specific mechanisms combined with a number of capable tools that are more appropriate for organizational computing environments. On petascale systems, heterogeneity and specialized hardware are likely to be the norm, so the issues in configuration management will become even more critical.

We propose research in four specific areas of configuration management: image description, configuration distribution, validation, and flexible configuration.

4.1.1 Image Description

Problem: Node configurations are often defined in terms of “images”. Images include descriptions of hardware (such as disk geometries), software packages to be installed, and modifications to be made after installation. This approach works well when images are extremely static or when a system only uses a few images. However, large systems tend to need to support many different types of images that change frequently. Image management becomes remarkably difficult, and can easily lead to system software failure due to misconfiguration.

Approach: We will explore a language-based approach to image description. Many of the concepts prevalent in modern programming languages such as inheritance and encapsulation could be used to dramatically reduce the complexity of image description.

4.1.2 Configuration and Software Distribution

Problem: Configuration information and software packages are typically stored in a central repository and are distributed to components at configuration time. This approach can lead to serious scalability and performance problems, particularly when a large number of nodes are validating or installing configurations, such as during a system boot.

Approach: We will explore two mechanisms for scaling configuration distribution: multicast delivery and configuration caching.

4.1.3 Validation

Problem: If a difference in the configuration exists between a central configuration repository and the actual configuration on an end-node, most systems assume the end-node is incorrect and reconfigure it. In some cases, particularly when administration responsibility is distributed, the end-node is actually configured correctly, while the central repository is out of date.

Approach: Initially, we will investigate methods of validating configurations – both on end-nodes and in a configuration repository. Then, should differences appear, various heuristics will be used to determine which configuration is actually correct, and decision action will be recommended to the administrator. In the long run, we would like to explore mechanisms for having end-nodes register their configuration with a repository server. The server would then be able to modify the central configurations as necessary. (This approach is closely related to the description capabilities of the language mentioned in section 4.1.1.)

4.1.4 Flexible Configuration

Problem: All configuration management techniques that we are aware of follow a rigid approach to configuration description and enforcement. Every detail of every system must be described, and every system must follow all configuration requirements in order for the system to work. This situation can lead to system failures because describing the entire system is a complex task, and the slightest mistake can have serious ramifications.

Approach: We will carry out research in the possibility of flexible configurations, in which as little as possible about the configuration is described formally. We are considering two options to explore:

1. Neighbor-based configuration. In this approach, an administrator would configure a single system correctly by hand (which is a fairly simple process). Then, all “neighbors” of that system would compare configuration details with that neighbor, and adjust their own configurations accordingly. In a future version of such an approach, this might include voting to determine correctness.
2. Late-binding configuration. In this approach, which is similar to late-binding of programming language variables to values, a system would only be given enough initial configuration information to boot. All other configuration information would only be delivered when it was necessary, thus reducing the potential for misconfiguration in unnecessary components.

Both of these approaches are extremely exploratory, but may be particularly useful in systems with an extremely large number of nodes (i.e. greater than 1M nodes).

4.2 Management Infrastructure

The management infrastructure of a system is the hardware and software used to manage the system. On advanced parallel computers, the infrastructure has the ability to install operating systems on nodes, monitor all aspects of all systems (ranging from temperature to compute load), the ability to power cycle systems, and some form of console management. In general, simple systems have simple management infrastructures while highly capable and very large systems require complex management infrastructures. In all cases, the role of the management infrastructure is to reduce and automate the administrative load on the system administrators.

Despite management infrastructures being fairly common, there is no community understanding of the real issues in management infrastructure design. Most management infrastructures are either ignored (i.e. the minimal system supplied by the vendor) or very specific to a single site or computer.

We will carry out research in the following areas:

4.2.1 Management System Organization

Problem: The community does not understand the issues in the organization of system management servers. Existing schemes are generally ad-hoc or highly customized. None are general, and none provide the base knowledge needed to design petascale management infrastructure.

Approach: In the past, we have experimented with a rigidly hierarchical management scheme, in which a single management node manages units of thirty-two nodes, and then a single management server manages all management nodes. We have observed that while this approach works for some aspects of management, others do not scale in this rigid way. We propose two areas of exploration:

- Further investigation and measurement of the issues and approaches in a rigid hierarchical system.
- Developing ways to dynamically map management functionality into a pool of management servers. This would result in having a “community” of management functionality rather than a hierarchy.

4.2.2 Boot Scheme and OS Installation

Problem: One of the primary functions of the management infrastructure is to deploy operating systems onto nodes of a parallel computer. Although there are many industry standards that are a part of this process (e.g. PXE-based booting, DHCP address assignment), it is still surprisingly difficult to install an arbitrary operating system on a node, ensure that it was installed correctly, and then assign the correct set of network addresses to that node. On simple systems, on which operating systems are very rarely changed, this is not a substantial problem. On more dynamic systems such as those that support OS development or those that are large enough to require regular hardware modification, this becomes a significant issue.

Approach: Ideally, the management infrastructure could be told that a node in a certain physical location should have a particular OS and a particular network address, and the system would then carry out the necessary actions. In past work, we developed a boot scheme that allowed us to dynamically install operating systems onto nodes. It relied on specific hardware being available, but demonstrated the capability. We will generalize this mechanism so that it does not rely on specific hardware, and we will further explore the issues of accurate network ID assignment.

4.2.3 Physical Node Management

Problem: The management infrastructure must be able to closely manage a node. Today, many clusters have the ability to monitor and control the physical aspects of a node such as its temperature, its power state, and its serial console. However, no standards have been developed for such management. We believe that this is largely due to a lack of common understanding of the issues in this area and an inability to programmatically describe these capabilities. Having a better understanding of the requirements for physical node management will greatly increase our ability to deploy and manage very large petascale computers.

Approach: We have already developed a set of tools that demonstrate the capabilities of remote power and serial management. We will formalize the protocols in use for these tools in order to initiate the development of an interface for physical node management. We will also explore additional capabilities such as temperature monitoring and analyze many of the vendor-specific solutions. Ideally we will be able to identify a core set of

interface capabilities that will feed into other existing efforts, such as the interface mechanisms in the Scalable Systems Software ISIC, the Supermon protocols in the Science Appliance project, and into generalized SNMP monitoring.

4.3 Virtual System Management

Petascale computers are likely to be built from millions of individual systems elements. Because existing systems are generally no larger than thousands of nodes, scientists are turning to virtual systems as a way to explore software scalability issues. Using virtual operating systems, a physical node can run anywhere from one to perhaps a hundred operating systems simultaneously. (The bounding factor is the amount of RAM on the physical node.) Thus, on a thousand node system, one might be able to test software on a hundred-thousand separate system images.

This scenario presents all sorts of interesting challenges to the management of a system. The virtual operating systems themselves must be configured correctly. A system must be built that can launch and disable the virtual operating systems. Management tools must be able to treat the entire set of operating systems (both real and virtual) as one computer, but must be able to differentiate between them when necessary. In short – while we know how to build such a system, we do not know how to effectively manage it. Until we can manage it, we cannot use it.

We will explore the issues of virtual system management by building a small virtual system testbed on existing clusters. In this testbed, we will aim to manage on the order of a hundred virtual nodes as if they were a real cluster. We will develop an understanding the issues in creating virtual nodes, configuring them, monitoring them, and then disabling them. In the long run, we will be able to demonstrate tools that can adapt to a highly dynamic system configuration. This capability will not only be necessary for clusters of virtual nodes, but for petascale systems.

4.4 Related Tools

As noted above, all of the previous research is carried out in the context of tool development and real system management. We find that building real tools, real prototypes, and running real systems is the best way to ensure that this work is truly relevant and useful in real situations.

As a part of this, we must sometimes develop tools and capabilities to support system functionality but that are not in and of themselves research areas at the present time. As a part of this work, we will continue to develop these tools when necessary. Because these are potentially useful to others, we will distribute them as a part of the results of this work. We anticipate three specific areas of tool development:

- Account system tools. The clusters that we test on must have ways to manage user accounts.
- Allocation system tools. The systems must have ways to record and report allocation of the system.
- Scheduler modifications. Inevitably, schedulers need to be modified to support complex systems and test some of these research concepts such as virtual clusters.

5 Relationship to Other Projects

This work is related to a number of other DOE-funded efforts.

- **Scalable Systems Software (SciDAC ISIC).** The Scalable System Software project, which is a joint project across a number of the DOE Laboratories, will define a standard interface between systems management functions and other aspects of the cluster systems software. The research in this proposal will clarify the needs of this interface from the systems management point of view and “find the path” for the SSS effort by carrying out research beyond the scope of SSS.
- **The Science Appliance (Base).** The Science Appliance project, which is joint between LANL and ANL, aims to develop an advanced and simplified infrastructure for supporting focus computational science, and to release it for general use. Some of the systems management concepts in this proposal, once proven, may be appropriate for eventual hardened development as a part of the Science Appliance project.
- **The Chiba City Project (Proposed).** Chiba City is a 512-CPU Linux cluster installed at ANL that is focused on supporting scalable testing and computer science development. Most of the research proposed here will be carried out on Chiba City. An upgrade to Chiba City has been proposed that, if funded, will result in a 1024-node cluster with operational support. The effort described here is related to that upgrade in that much of the systems software that operates Chiba City will track the results of this project. To some

degree, the work proposed here is the “intellectual and research component” of Chiba City, while the Chiba City funding itself supports the deployment and maintenance of the software and operations of the cluster.

- **MPICH and Scalable I/O.** We collaborate with the MPICH group actively. Much of their research is closely associated with this research, particularly since both activities fit within the “systems software” categorization. None of that work will be supported by any of the activities described here, but much of it will likely be used to test the system management functionality that we are exploring.
- **Grid Computing.** We collaborate with the Distributed Systems Laboratory on many cluster- and grid-related projects, but all of these are funded separately. Some of the results of this project will no doubt be useful to those projects in the same way that activity in the grid arena has been instrumental to some of our thinking.

6 The P.I.s

Here will go brief bios of Rémy Evard (PI) and Rick Stevens (co-PI).

7 Related References and Software

R. Evard, "Chiba City: A Case Study of a Large Linux Cluster", in *Beowulf Cluster Computing with Linux*, by Thomas Sterling. MIT Press, 2001.

R. Evard, “An Analysis of UNIX System Configuration”, in *Eleventh USENIX Systems Administration Conference Proceedings*, pages 179-194, October 1997.

R. Evard, J. Navarro, D. Nurmi, “City”, a software toolkit for large-scale cluster management, first released in March 2000.

R. Evard, J. Navarro, D. Nurmi, G. Rackow, “Msys”, a software toolkit for large-scale computing environment management, first released in March 2000.

R. Evard, “cfg”, a tool for configuration management of large computing environments, first released in October 1997.

8 Budget Outline

The budget request for this proposal is \$800K. This covers four FTEs, that will include a combination of the P.I.s, research staff, and programming support. It includes all other direct costs as well.