

A Case Study in Configuration Management Tool Deployment

Narayan Desai, Rick Bradshaw, Scott Matott, Sandra Bittner,
Susan Coghlan, Rémy Evard, Cory Lueninghoener, Ti Leggett, John-Paul Navarro,
Gene Rackow, Craig Stacey, and Tisha Stacey
Mathematics and Computer Science Division, Argonne National Laboratory,
Argonne, IL 60439

September 1, 2005

Abstract

While configuration management systems are generally regarded as useful, their deployment process is not well understood or documented. In this paper, we present a case study in configuration management tool deployment. We describe the motivating factors and both the technical considerations and the social issues involved in this process. Our discussion includes in an analysis of the overall effect on the system management model and the tasks performed by administrators.

1 Introduction

System administration is, at its heart, the profession of helping people to use computers. System administrators are domain experts who provide impedance matching between users' desires and computers. The expertise of system administrators is manifested in their choices of computer hardware and software and of system configuration. Environments are constantly changing; the need for timely software updates and frequent configuration changes has never been greater. Configuration management tools provide different levels of automation and representational models, but all have the same basic goal: to help in the system configuration process.

Nevertheless, adoption of configuration management systems has lagged substantially behind tool development. The explanation rests largely with the up-front cost of building an adequate representation of an environment, but the problem can also be traced to the requirement that administrators change their administration methods. This change makes the configuration management adoption process quite costly and time consuming.

The Mathematics and Computer Science Division of Argonne National Laboratory consists of nearly 200 researchers, programmers, students, and visitors. The division is home to several hundred workstations, three large clusters, and other high-performance computing resources. One group maintains this diverse set of resources. Recognizing the need for more efficient management mechanisms, members of the system staff have contributed to a number of system management research efforts [6, 4, 7].

In the summer of 2004, the group decided to deploy Bcfg2 [3], a configuration management tool developed in house. As of May 2005, much of the deployment has been completed. Bcfg2 manages the general infrastructure and two clusters and is being deployed on another cluster and on an IBM Blue Gene/L system. The general infrastructure is extremely complex and involves the largest number of administrators, hence; we focus on this particular deployment.

This paper documents our experiences and lessons learned in adopting a configuration management tool. We discuss the goals that prompted this adoption and describe the deployment process in terms of both technical and—more important—social issues. The adoption of Bcfg2 has dramatically changed the procedures and model used by system administrators in the division. We discuss these outcomes in detail.

One cannot talk about configuration management without including technical aspects that are likely tool-specific. As the same time, we believe that many of the issues we faced and benefits we reaped in

deploying Bcfg2 are intrinsic to the adoption of any configuration management system. Where possible, we avoid the discussion of Bcfg2-specific topics.

Since configuration management has been recognized as a problem for quite some time [4, 9], numerous tools have been written to address this task. Many of the tools, in particular LCFG [1] and Quattor [8], can cause large changes in management methods; our discussion of Bcfg2 [3] is especially applicable to such tools. Other tools, such as SystemImager [5] and CFEngine [2], have a model that is not substantially different from manual system administration; for these tools our discussion of deployment issues is less applicable.

2 Architecture

In this section, we describe the overall goals that motivated our adoption of Bcfg2. We also briefly describe Bcfg2, in order to provide an understanding of the outcomes we achieved, as discussed in Section 5.

2.1 Goals

The bulk of the benefits we hoped to achieve with better configuration management were efficiency related. We were spending an inordinate amount of time performing software upgrades and applying security patches. In partial response to this problem, we found ourselves faced with the deployment of a new base operating system, Debian Sarge. This occasion provided an ideal opportunity to add a new configuration management system into the environment.

We wanted a centralized location where all configuration information for all clients could be stored and coordinated. This requirement raised a variety of expressivity issues. We wanted all aspects of the configuration specification to be as terse as possible. That is, we wanted to express all required configurations in a way that minimized redundancy in the specification. Similarly, we wanted configuration changes to be made in as simple a way as possible. For example, adding a software upgrade to all systems or uniformly changing the contents of a configuration file should be trivial.

We also wanted a high-level interface into the configuration management that allowed configuration specification based on machine role. For example, it should be easy to ask for another instance of a given role, like a webserver.

Another important goal was to eliminate configuration state local to clients. That is, all machines should match their configuration specification. Once no local configuration exists, no local configuration data will be lost in the event of a machine rebuild. Adherence to this goal enables machines to be trivially rebuilt. Furthermore, once the configuration specification contains all configuration directives needed to produce a goal, this goal can be replicated.

Finally, we wanted good practices encoded in the configuration management tool. The tool should perform operations in the safest manner possible. It should also make every attempt to ensure that new configuration specifications are activated.

2.2 Bcfg2 Overview

While the Bcfg2 architecture is not the main focus of this paper, several details are required to understand our deployment. Bcfg2 has a client-server architecture. The server is responsible for building configurations for clients, based on a global configuration specification. This specification contains high-level directives for clients, referred to as the metadata, and a set of configuration rules that can be used with the metadata to produce literal client configurations. That is, they contain information that needs no further processing for client use. These configurations are also assumed to be comprehensive; they contain information for all aspects of client configuration. For example, a software package or service that is active on a client will be included in its configuration. Any installed configuration entities not listed in the configuration are flagged as “extra.” The Bcfg2 client uses heuristics to discover this “extra” configuration.

The Bcfg2 client connects to the server and downloads its configuration. It then inventories the local system and compares this inventory to the configuration specification. Any discrepancies found in this process are flagged for later correction. Next, the Bcfg2 client runs its heuristics to find extra configuration. Anything located in this pass is similarly flagged for later correction. After the detection

work is done, the Bcfg2 client rectifies any conditions previously found. This behavior is tunable; dry-run and extra configuration removal modes are available.

Once the client has completed operations, it uploads statistics to the server. The information includes the overall machine state (clean or dirty) and lists of the failing and modified configuration entries. This information is stored on the server and can be used to generate nightly reports about the overall state of the network and its correspondence to the configuration specification.

3 Deployment

Deploying Bcfg2 in our environment took a substantial amount of time. Our experiences since its deployment, however, have more than justified this investment. In this section we describe the technical and social issues involved in the process. Following this, we discuss various improvements to the administrative process enabled by this deployment.

3.1 Technical Deployment

Deploying Bcfg2 took approximately four months of work performed primarily by one person. As is frequently the case with systems like this, the first 90 percent of the work was completed in the first six weeks, while numerous small issues were resolved over the course of the next 10 weeks.

Deployment was initially undertaken as part of a base OS upgrade, moving from Redhat 7.3 to Debian Sarge. We chose this occasion because our previous experience with a mid-stream switch to Bcfg1 had proved problematic. Basically, systems managed in an ad hoc fashion tend to have a lot of configuration inconsistencies. Ensuring that these machines can be cleanly rebuilt by using a configuration management tool can be quite difficult. The introduction of a configuration management tool is possible but must be carefully performed. In contrast, the deployment of new machines can be easily done, as the deployment process is available for examination.

The first goal was to get an automated build system working properly. Our environment has two main types of machines: workstations and servers. We decided to use SystemImager [5] for initial client installations, calling Bcfg2 before initial reboot. This approach ensures that machines come up properly configured and secure upon first reboot. Two profiles for Bcfg2 were created and made selectable from the initial SystemImager boot menu.

Creating the configuration profiles for workstations and servers was not difficult. This process consisted of specifying all configuration aspects of workstations, including all environment-specific modifications. A workstation configuration had been created for Bcfg1 and was easily migrated to Bcfg2. Had this not been the case, the creation of an initial configuration would have taken a few days. This process consists of recording all important aspects of configuration in the central specification. Aspects can be quickly incorporated into the specification. Our workstation configuration initially consisted of nearly 1,100 configuration aspects. Once this process was completed, we constructed a server configuration as a subset of the workstation configuration, since many of the configuration aspects of these classes are similar.

Once a simple build mechanism was in place, we rebuilt administrator desktops and some test servers using the new image and management system. In one case, the administrator ran with two desktops concurrently for several weeks, in order to find subtle aspects of needed workstation configurations not yet included in the workstation profile. The process paid a big dividend; by the time "normal" users started using workstations based on the new build, few configuration problems remained.

After the new system became full-featured and stable enough, we started to allow users to request machines with it. These early users provided the remainder of input regarding missing configuration from the new workstations. After several of these users had successfully used new-build machines for some time, we began upgrades for the rest of the division.

In this stage the simplified build process was particularly beneficial. Three people performed most of the workstation rebuilds. The initial target was to complete most machine upgrades in four weeks. Desktop rebuilds were a simple process. Each machine has a local scratch disk, which gets cleaned upon rebuild. Users save any needed data stored there. Once this data is saved, rebuilds can occur at any time and take 30 minutes. All user interactions occur in the first three minutes of the process; the rest of the process can complete unattended. In the course of a month, nearly 80 desktop machines were upgraded.

After completing the workstation upgrades, we shifted our focus to the server machines. While the server profile had been used to build new machines, many existing servers remained unmanaged by Bcfg2. As these servers were replaced, we encountered a whole new set of issues relating to tool deployment—issues that were generally related not to tool completeness but rather to the way configuration changes were propagated to machines. For example, certain machines were deemed too important to perform automatic changes.

This realization necessitated a major change in our deployment strategy. Initially, all machines had called Bcfg2 each night, and all required changes were performed. On workstations, this model was good enough; while these machines are important, they don't cause congestive failures when problems occur. For servers, however, we needed to run Bcfg2 in dry-run mode each night and send its state to the relevant administrator. While examining this issue, we also realized that cluster nodes should run Bcfg2 in yet another way: between jobs, in order to prevent interference with user calculations.

These experiences shifted our focus from a tool-based one to an administrator experience-based one. Initially, we were quite concerned about tool correctness and completeness. As Bcfg2 proved itself and bugs were fixed, our confidence in its correctness greatly increased. Also, our workstation configuration proved to be as complicated as any other in our environment, so the configuration specification process for our servers was fairly simple.

Our change in focus amounted to the realization that the tool client-side functionality was not sufficient. The tool must also provide enough information for administrators to make effective decisions as conveniently as possible. Moreover, it must supply configuration state information in a convenient way. From this point onward, nearly all development focused on an information presentation system to provide a sort of scoreboard for the entire network and its configuration state.

3.2 Administrative Improvements

The deployment culminated with the implementation of a robust reporting infrastructure. This infrastructure provides periodic information about the current configuration state of all clients, the time of their last contact with Bcfg2, and a list of pending configuration changes. The information allows administrators to observe the logic employed by Bcfg2 during normal operations. This single factor had more impact on administrator trust in Bcfg2 than all others.

The reporting system results in emails, Web pages, or RSS feeds that contain information about either specific hosts or the overall status of all Bcfg2 clients. Administrators can subscribe to reports about particular clients of interest, or all systems, enabling the detection of two common problems: clients not receiving configuration updates and clients unable to perform needed configuration updates. Administrators can also observe the operations taken by Bcfg2 over time.

These reports create a picture of the entire network that gives discussions about configuration a basis in fact. The reports greatly improved our understanding of our systems, their configuration, and its modification patterns. More important, administrators grew to trust Bcfg2, and hence allow the remainder of our network, composed of our most important machines, to be rebuilt and managed by Bcfg2.

4 Social Issues

While the technical aspects of Bcfg2 deployment were complex, managing the social aspects of the deployment was far more difficult. This process occurred in an ad hoc way in our group and could have been substantially improved had we initially known the related considerations.

Adoption of any configuration management system is a stressful process. Configuration management tools affect the whole of the system management process. All administrators are required to adopt new processes for achieving the same tasks they already know how to accomplish. Since such changes can directly impact the services system administrators are expected to provide reliably, tensions can run high.

Communication also becomes an issue, because of the variety of perspectives held by different administrators. We found that three main factors motivated most of our disagreements during the deployment process: trust in the tool, a belief in the benefits provided by the tool, and the perception of a complexity increase or decrease caused by the tool.

Trust is certainly the most important of these factors. If a tool remains untrusted by administrators, it will never be substantially used in their environment. The trust-earning process certainly varies from person to person, but we can discuss the issues we observed. In general, as users gain more experience with the configuration management tool, they begin to trust it more. Two aspects of trust are important. The first is that the tool can properly represent any configuration state that the administrators may desire. This problem is largely technical and is solved as the administrator gains experience and familiarity with the tool. The second aspect of trust is harder to earn. Administrators must trust that the tool will perform the specified changes appropriately. This sort of trust is earned only through a long period of experience running the tool and observing the resulting configuration changes. The process can be accelerated, however, through the exposure of tool decision information. If administrators can easily examine the decision process each time they run the tool, then their trust will grow more rapidly.

The second factor that guides the adoption process is the *perception of benefit*. All administrators in our group were already overburdened with tasks, so expecting them to spend time learning something new was difficult. If a management tool didn't present a clear case for rapid improvement in efficiency, learning about it wouldn't be given high priority—and rightly so. Lack of time to experiment with a tool clearly has a detrimental effect on overall trust in the tool. Hence, adoption can be greatly hampered simply because of a lack of information. This factor can be minimized, however, by providing improvements that quickly benefit all administrators. Easy-to-adopt solutions to common problems provide a good incentive to try out a tool.

The third factor in the adoption of configuration management systems is the *perception of complexity*. The complexity increase may be minor; but to users unfamiliar with particular tool, this added complexity will be unattractive because it will lead to decreased efficiency for some tasks. Over time, as the administrators become more familiar with the tool, the added cost of this complexity is reduced. Once the deployment is complete, complexity is compartmentalized, as administrators can focus on their task and ignore others. For example, an administrator responsible for web servers can focus on `apache` configuration without worrying about upgrading `ssh`.

All of these factors are heavily interrelated. Throughout the deployment process, their influence was felt through each of the issues we encountered.

4.1 Disagreements

Throughout the process of tool adoption, each administrator in the group internalizes more information about the new tool, building an opinion about the tool's value and potential use cases. Each administrator will trust the tool to a different extent and will have different ideas about the potential benefits to be gained and the complexity costs involved. The following discussion describes the different points of contention that arose in our group. Each of the major issues is described in the abstract, along with the factors that turned out to be important. While we believe that these issues are representative, the list is by no means comprehensive. People like to argue about all sorts of issues.

- **Buy-in.** Initially, everyone must agree that a given tool should be used. This issue is largely influenced by the trust and comfort levels administrators have with a tool, not to mention its technical correctness. Learning how a tool works is essential, but this process can require substantial amounts of time, especially in large groups.
- **Existing investments.** A working environment typically has a sizable investment in technical methodologies. Generally, a set of utilities has been developed to automate particular tasks, and a large body of institutional knowledge has evolved around specific tools and methodologies. Moreover, the creation of these processes and tools usually implies an emotional investment. Overcoming these investments without alienating members of the group is difficult—but possible.
- **Level of control.** Any tool will have a fundamental set of assumptions or functionality that guides its behavior. Even if the tool is reliable, it can be difficult to convince everyone that a change is beneficial—especially if the methods that a proposed new tool uses to implement a given task differ from those historically used. This issue can be overcome only with a large amount of empirical evidence that these methods are equivalent. As when programmers made the initial switch to high-level languages, administrators are accustomed to making complex low-level changes to systems, rather than using high-level specifications of functionality.

Our goal is not to diminish these concerns in any way; in fact, all of these considerations are quite reasonable. In some ways, these concerns illustrate the core essence of system administration. That is, the adoption of a tool that will radically affect every aspect of system maintenance is not a decision that should be taken lightly. Administrators, after all, shoulder the brunt of system failures, misconfigurations, and software configuration problems.

Rather, our intent is to document these concerns. We believe we have gained a deeper understanding of our requirements and also provided a comprehensive vetting process. Moreover, through this documentation, we hope to aid other groups attempting the same sort of transition.

The highest-level problem can be most easily summarized as “buy-in.” Once that problem is resolved, the tool deployment will achieve critical mass and no longer serves as a point of contention.

4.2 Hindsight

Despite the social issues we confronted, we managed to implement a configuration management system. We attribute our success to three factors.

- Our group was predisposed to recognizing the value of configuration management. This attitude removed an important initial hurdle from the process. Had we simultaneously needed to convince the group of both the need and mechanism for configuration management, the outlook would have been far worse.
- One administrator was involved in both the Bcfg2 development activities and maintenance of the division infrastructure. Without his work as liaison, many arguments wouldn’t have carried nearly the weight that they did; adoption could have easily stalled.
- Our group is quite amicable, and not particularly sentimental. These characteristics allowed easier discussion of contentious subjects and the replacement of existing mechanisms and tools.

Even with these factors, considerable perseverance and evangelism were required. The outcome of this process was in doubt throughout much of the deployment process. At many points, administrators did not seem to find the model compelling enough and did not trust the tool. Fortunately, everything worked out well in the end.

4.3 Recommendations

While there is a great amount of social variance among groups of system administrators, we can make several recommendations based on our experiences.

- The tool under consideration must have an advocate who is technically respected by the group. His role will be to assess the various tools and select one that seems best suited to the environment. He will also need to convince other members of the group that the chosen tool is the proper one.
- Administrator concerns should be addressed, not ignored. These concerns are generally based on experience and reflect potential technical issues that could occur later. Once all concerns have been addressed, the group will more readily accept the tool into daily operations.
- Tool advocacy will be most compelling when improvements mentioned are useful in the short or medium term. Long-term improvements, while desirable, do not often provide short-term motivation. Hence, long-term benefits secure a position on the “when we have time” list for tool deployment.

5 Outcomes

In spite of the difficulties described above, this project has succeeded beyond our expectations. We have several new capabilities we could not have predicted even six months ago. All of these capabilities have resulted from three major shifts in architecture. First, we now have a centralized configuration specification and statistics that allow reasoning about the desired (and actual) configuration states of our entire environment. Second, we now have an abstraction barrier between our specifications of machine function and the implementation of that function. This barrier simplifies both parts of the configuration specification and allows easier interactions. Third, several operations have been completely implemented

by the configuration management system, thereby reducing the cost in time, and improving the economies of many processes. Each of these improvements contribute to a fundamental alteration of the management model for our environment.

5.1 Configuration Specification

Many configuration management tools have a centralized specification that describes the desired state of the network. Where Bcfg2 departs from this model is the addition of detailed statistics about client state 3.2. The addition of these statistics to the central specification causes what would be a static set of rules to become a living document, automatically updated by Bcfg2. The desired configuration and all deviations from it are available in a central location.

The existence of a configuration oracle for an entire network alters the administration mindset in that global notions of state can now be constructed. Many data mining techniques can now be utilized. Reports describing network configuration state, reporting on the frequency and success of client configuration processes can give a thorough picture of unexpected client states before users are affected. Similarly, reports automatically generated from the configuration specification can provide insight into current software revisions, client functionality, and potentially even system interdependence. Reports such as these can prove useful for auditing purposes, the training of new employees, and high-level descriptions of services provided.

5.2 Function Abstraction

The configuration specification used by Bcfg2 splits information into three layers. The first, called the metadata layer, describes function information in terms of clients and classes. For example, the metadata may describe a class of clients that include webserver functionality. The second layer, called the repository, ascribes meaning to those descriptions. In the same example, the repository would contain information describing what “being a webserver” means. The third layer, implementing client reconfiguration operations, receives configurations from the higher two layers and reports on execution results. While this architecture is Bcfg2 specific, many other complex configuration management tools are structured in a similar way[1].

This layered specification provides an abstraction barrier isolating function assignment from function description. For example, after “being a webserver” is defined, one can easily and reliably add new webserver instances. Once this operation is possible, programs that generate these changes become possible. The addition of logic into this function determination process introduces dramatic flexibility into the network. It also allows common function shifting operations to be automated.

Repository semantics benefit from this abstraction barrier as well. Several simple context-specific formats can be used to describe implementation behavior. Similarly, scripts can be written to autogenerate these files, if desired. Important functionality, ranging from automated patch integration to service reconfiguration, can be implemented.

The client-side tool receives a literal set of configuration directives from the Bcfg2 server. It compares the current operational state with the desired state and performs a set of state transitions, focusing on safely transitioning into the goal state. The client tool implements a small number of operations that have been well tested. Upon completion, the Bcfg2 client returns a set of statistics about the configuration state of the client and modifications performed. In conjunction, these two features allow administrators to focus on configuration goals, while allowing the client tool to determine a reasonable set of operations to implement these goals. In this way, the client tool isolates the upper-level users from some low-level complexity.

5.3 Tool-Based Simplification

Tools are intended to make tasks easier. Using Bcfg2, we found three major tasks that were vastly simplified.

System rebuilds have become trivial. Statistics are used to verify that the configuration specification matches the running state of the machine. After this match has been verified, the machine can be rebuilt at the appropriate time. The previous process consisted of a lot of manual verification; a second system would typically be used to verify functionality and swapped in after everything worked.

The new machine build process has also been greatly accelerated and simplified. Several stock profiles are available; the user is presented with a menu at the beginning of the build process. No other setup is required. Hence, nonroot users are now able to build machines quite easily.

The class-based system allows new profiles to be easily created by composing existing function groups. Hence, users can create new profiles whenever needed, without shoehorning functions into the same profile. In this fashion, the configuration specification becomes more concise and representative of the actual running configuration.

Each of these tasks has become easier with the addition of a configuration management system. While these tasks could previously be performed, more expertise and privileges were frequently required. The net impact of these changes is to empower users to be able to perform tasks that were not previously possible.

5.4 Overall Efficiency Gains

All of these factors contributed to an impressive increase in efficiency. We estimate that before conversion three FTEs of time were spent on the maintenance of our workstation and server environment. These administrators performed a variety of tasks, ranging from security updates to new software installation, and user-requested reconfigurations. After conversion, between one-third and one-half of an FTE is consumed by these activities.

The time freed by these improvements is now available for a variety of activities. Large-scale infrastructure improvement projects are under way. More time for interactions with users has resulted in more satisfying services for users and more accurate assessments of user needs.

Basic administration tasks remain split across several administrators. The use of configuration management has also reduced the cost of task distribution. The existence of a central location for configuration specification imposes a set of expectations that allow administrators to find one another's work and synchronize when needed. Most simply, all administrators are kept on the same page.

6 Conclusions and Future Work

Our main goal in this paper is to document the process and results of deploying a configuration management tool. While the process is difficult, the outcomes are worthwhile. Indeed, the outcomes we experienced have more than justified the effort involved. While some of the difficulties faced were certainly peculiar to our group, we feel that the ones documented here are indicative of the fundamental issues in a change of this magnitude.

While our discussion may suggest that this task is too difficult for many groups to consider, we strongly believe that configuration management is an important technology to deploy in nearly any environment. We hope that our discussion of difficulties will not dissuade other groups but, rather, will be used to navigate an admittedly difficult process.

Many administrative tasks have been vastly simplified, and much useful data can be mined from the configuration specification and statistics. The availability of this data has enabled a higher level of reporting and comprehension of our environment than was previously attainable.

While many improvements have already been realized, further substantial efficiency gains can be achieved. Certainly, Bcfg2 could be improved. Several technical improvements relating to information representation require attention. Also, an interface to force client reconfigurations would be useful. The user interface will also continue to improve with more experience.

As a group, we expect administrative model changes to continue, although with a less disruptive effect. Many processes remain that could be automated. Also, a service that allows reconfiguration delegation would be useful, allowing users to update aspects of configuration as appropriate. Moreover, we hope that administration streamlining will continue.

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 W-31-109-ENG-38.

References

- [1] Paul Anderson and Alastair Scobie. Large scale Linux configuration with LCFG. In *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, October 10–14, 2000, Atlanta, Georgia, USA* [10], pages 363–372.
- [2] Mark Burgess. Cfengine: A site configuration engine. *USENIX Computing systems*, 8(3):309–402, 1995.
- [3] N. Desai, R. Bradshaw, R. Evard, and A. Lusk. Bcfg: A configuration management tool for heterogeneous environments. In *Proceedings of the 5th IEEE International Conference on Cluster Computing (CLUSTER03)*, pages 500–503. IEEE Computer Society, 2003.
- [4] Rémy Evard. An analysis of UNIX system configuration. In *Proceedings of the Eleventh Systems Administration Conference (LISA XI), October 26–31, 1997, San Diego, CA, USA*, pages 179–194, Berkeley, CA, USA, 1997. USENIX, USENIX.
- [5] Brian Elliot Finley. VA SystemImager. In *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, October 10–14, 2000, Atlanta, Georgia, USA* [10], pages 181–186.
- [6] Michail Gomberg, Rémy Evard, and Craig Stacey. A comparison of large-scale software installation methods on NT and UNIX. In *Proceedings of the Large Installation System Administration of Windows NT Conference, August 5–8, 1998*, pages 37–47, Berkeley, CA, USA, 1998. USENIX, USENIX.
- [7] J. P. Navarro, R. Evard, D. Nurmi, and N. Desai. Scalable cluster administration - Chiba City I approach and lessons learned. In *Proceedings of the 4th IEEE International Conference on Cluster Computing (CLUSTER02)*, pages 215–221. IEEE Computer Society, 2002.
- [8] Piotr Poznański, German Cancio Meliá, Rafael García Leiva, and Lionel Cons. Quattor - a framework for managing grid-enabled large-scale computing fabrics. In *Proceedings of the Krakow Grid Workshop '04*, Krakow, December 2004.
- [9] Steve Traugott and Joel Huddleston. Bootstrapping an infrastructure. In *Proceedings of the Twelfth Systems Administration Conference (LISA XII)*, pages 181–196, Berkeley, CA, USA, 1998. USENIX.
- [10] USENIX. *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, October 10–14, 2000, Atlanta, Georgia, USA*, Berkeley, CA, 2000. USENIX.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (“Argonne”) under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.