RC23784 (W0511-053) November 7, 2005
Computer Science

IBM Research Report

Blutopia: Cluster Life-cycle Management

F. Oliveira
Rutgers University

J. Patel
University of Illinois at Urbana-Champaign

E. Van Hensbergen, A. Gheith, R. Rajamony
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Blutopia
Cluster Life-cycle Management
DRAFT

F. Oliveira
fabiool@cs.rutgers.edu
Rutgers University

J. Patel
jaypatel@uiuc.edu
University of Illinois
at Urbana-Champaign

E. Van Hensbergen *
bergevan@us.ibm.com
IBM Research

A. Gheith
ahmedg@us.ibm.com
IBM Research

R. Rajamony
rajamony@us.ibm.com
IBM Research

November 2, 2005

Abstract

We have implemented a proof-of-concept prototype named Blutopia that embodies our ideas of a new software life cycle management paradigm driven chiefly by simplicity. Aiming at facilitating management, Blutopia allows administrators to deploy multiple machines fast and effortlessly in as little time as it takes to perform a system reboot. It is equally easy to manage and augment the deployed infrastructure by assigning new roles, upgrading existing roles, and rolling back software versions and/or configuration in the event of unexpected problems arising from an administrative task. Underlying our administrative paradigm is stackable file system technology providing the ability to efficiently manipulate layers of the file system, perform instantaneous snapshots, and institute copy-on-write layers to capture local modifications to an otherwise shared system or application image. We conducted a preliminary performance analysis of our mechanism and concluded that it degraded neither the performance nor scalability of the cluster nodes on which it was deployed.

*Supported in part by the Defense Advanced Research Projects Agency under contract no. NBCH30390004.

1 Introduction

The computing infrastructure on which small and medium-sized firms rely is becoming increasingly complex. Many such companies depend on software components distributed over a number of servers running internal application suites as well as providing on-line Internet services. These distributed components include database servers, email servers, identity management, application servers, web servers, load-balancers, and many others. This diversity of mutually dependent software complicates the fundamental tasks of systems management: initial installation, software upgrade, new hardware deployment, and coherent cluster configuration.

The complexity of system deployment and configuration is mitigated to some extent by the existence of value added resellers (VAR) and regional system integrators (RSI) which bundle hardware, operating systems, middleware, applications, and administration tools. The installation and configuration of these bundles is still predominantly a manual process. End-customers must go through the tedious process of installing systems from CD-ROM or DVD media and then must manually configure the cluster.

New hardware, software upgrades, or configuration changes require the end-user to manually take machines out of service, make the changes, test the new configuration, and then redeploy the server to a production environment. If the administrator wishes to expand capacity or re-provision the cluster to balance the workload, they must repeat the previously mentioned tedious installation process making sure that all appropriate patches and configurations are applied to each node – an error prone process at best. That humans invariably make mistakes worsens the problem. Several studies have shown that human mistakes committed during system administration were the predominant cause of service disruptions and outages [5, 11, 10, 9].

Customers purchase servers to run software components of interest to them, therefore an effective systems management solution must revolve around the installation, maintenance, and management of these components. We have built a prototype to assist in the management of a cluster's software life-cycle named Blutopia. It seeks to automate many of the aforementioned tasks, provide version control of system changes, and reduce time to install, upgrade, configure, re-provision, and deploy services.

In the remainder of this paper, we first present the high-level architecture of Blutopia in Section 2. In Section 3, we explore Blutopia's functionality by unveiling how it manages roles and versions. Section 4 describes the underlying implementation that enables these features. Section 5 evaluates Blutopia's overhead, presenting the results of the preliminary performance analysis we conducted. We finish the paper by surveying related work in Section 6 and pointing out our planned extensions to this work in Section 7.

2 Design

The aforementioned notion of a software-component-centric systems management tool leads to Blutopia's basic administration unit: the *role*. Imagine a scenario in which brand new servers are delivered to the end customer by a value added reseller (VAR) or regional system integrator (RSI). With Blutopia, initial deployment can be accomplished by unpacking the hardware, plugging it into an existing network, turning it on, and assigning pre-existing roles to each system through a web-interface served from the storage appliance, management module, or administrative server. As the infrastructure evolves, new machines can be brought into service by simply assigning pre-existing roles to them. Likewise, as new roles, versions, or configurations become available, administrators can re-provision the cluster in a matter of minutes. In addition, Blutopia empowers system administrators with the ability to quickly rollback machines to checkpoints instantiated prior to the execution of administrative tasks. An arbitrary number of these checkpoint snapshots can be logged by the Blutopia server.

2.1 Definitions

Blutopia is a tool meant to manage a set of *services* that are required by the end user. The services may range from a Web-based bookstore to payroll and personnel management suites to be used internally by a business. A service is comprised of a set of *components* that work in conjunction, where each component is associated with a *role*. For example, a Web-based bookstore service might use five components: one component whose assigned role is "LVS (Linux Virtual Server) front-end load balancer", three components whose roles are "Apache Web server", and one component designated as "MySQL database server". Blutopia also supports multiple *versions* of roles. For instance, the users might be able to choose among two different versions of the role "Apache Web server".

Besides support for management and deployment of services, another function Blutopia provides is extensibility. Users may create new roles, as well as capture new versions of a role, making them available for future deployment. In a typical business setting, local IT staff would manage and deploy services, whereas contracted *publishers* would supply new roles and upgrades. The details on how Blutopia deals with roles and versions are unveiled in Section 3.

2.2 Architecture

Figure 1 depicts Blutopia's architecture. The so-called Blutopia network embodies all machines running services managed by Blutopia, as well as the manager machine running Blutopia itself.

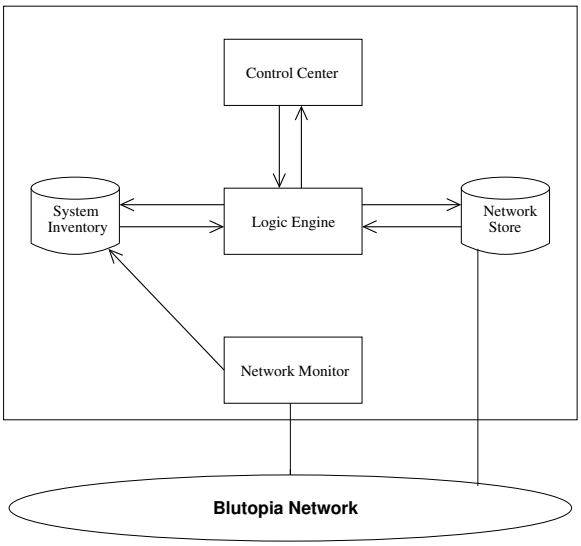

Figure 1: *Blutopia architecture.*

As shown in Figure 1, Blutopia has four core components: a Control Center, a Logic Engine, a Network Monitor, and a Network Store.

From a high-level perspective, users interact with Blutopia by means of a GUI, issuing commands to the Control Center which in turn invokes the functions exported by the Logic Engine. The latter interacts with the Network Store to perform the actions needed to carry out the user commands. The current status of the cluster is collected by the Network Monitor and is reflected in the Control Center user interface.

In the next few paragraphs, we briefly describe the internals of each Blutopia component and how they interact with each other.

Network Monitor. As one would expect, the Network Monitor performs connectivity-related actions. It constantly monitors the network in order to detect new hardware, broken links, and service disruptions. When a new machine is detected, the Network Monitor updates the System Inventory (see Figure 1), a database that stores information about all machines belonging to the Blutopia Network. Whenever the Network Monitor detects a service disruption, it reports that the corresponding machine is unreachable and in the near future will activate autonomic recovery agents within the Logic Engine. The current prototype system provides the Network Monitor functionality through scripts monitoring the DHCP server log and the *Ganglia* monitoring infrastructure [8].

Control Center. Upon receiving a command from the user, the Control Center updates the System Inventory with new configuration information and calls the corresponding Logic Engine agent.

It also displays notification messages on the GUI corresponding to the status information it receives from the Logic Engine and the Network Monitor. Our prototype Control Center is implemented as a set of PHP scripts which interface with AJAX Javascript GUI interfaces [4].

Logic Engine. The Logic Engine is Blutopia's heart. It is composed of a set of administrative agents which interact with the System Inventory and Network Store to implement the various functions which will be described in Section 3. The Logic Engine records the association of system images to machines in the System Inventory. Its agents are responsible for adjusting configurations as new applications are deployed or as services move to different physical machines. The prototype Logic Engine is currently implemented as a set of Perl scripts.

Network Store. The last and potentially most important component is the Network Store which keeps the file system and kernel images used by the machines belonging to the Blutopia Network. The key enabling feature of our Network Store implementation is the ability to efficiently stack file systems providing copy-on-write (COW) layers as well as multiple shared file system image layers providing content, configuration, applications, and system software. The same technology enables us to instantaneously snapshot the file system before each administrative change, giving us the ability to roll-back system state to correct configuration errors. We detail how stackable storage technology is leveraged by Blutopia in Section 4.

We have deployed our prototype implementation of Blutopia on an IBM Blade Center, installing our infrastructure on an IBM Total Storage appliance and targeting disk-less IBM HS20 xSeries blades. We envision migrating the Blutopia infrastructure to the integrated Blade Center management module in the near future. While our prototype targets a Blade Center embodiment running Linux, there is nothing preventing its use with more traditional Intel and PowerPC based servers running either Linux, Windows, or AIX.

3 Administrative Tasks

3.1 Component Installation

Machine Detection. Before installation, all new machines must be configured to boot over the network through the PXE boot protocol. The Network Monitor detects the addition of a new machine when the machine issues a DHCP request for an IP address to Blutopia. At this moment, the Network Monitor stores information pertinent to the new machine in the System Inventory and the machine is given a default boot image containing only limited functionality. After booting from

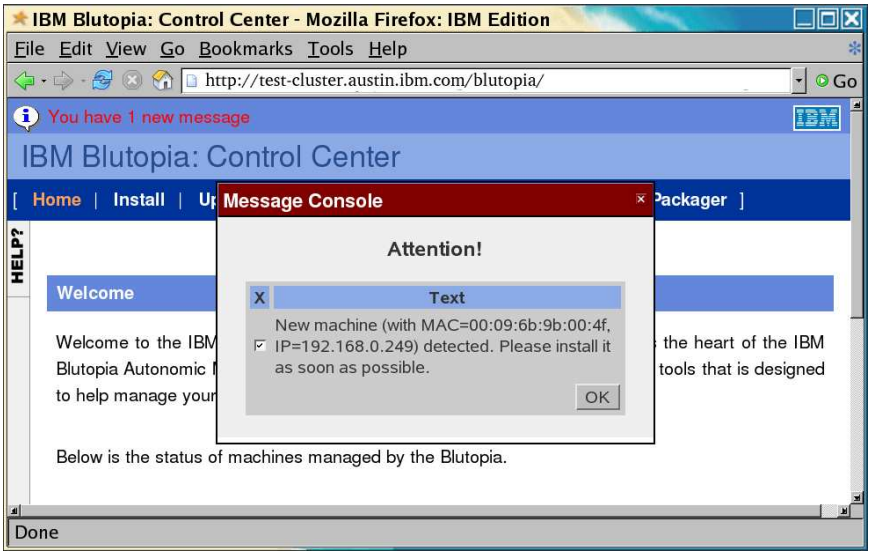

Figure 2: *Install function: detecting a new machine.*

the default image (a process which takes approximately two minutes), the new machine effectively joins the Blutopia Network, becoming available for future operations. Note that multiple machines can be added to the Blutopia Network at the same time.

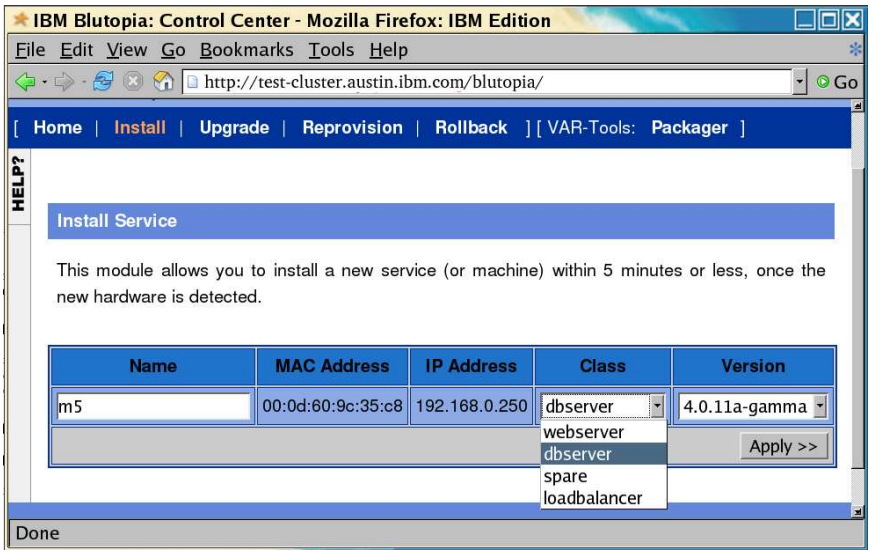

Figure 3: *Install function: assigning a role to a new machine.*

Role Assignment. After initial installation of new machines has been completed, Blutopia's messaging system, which is part of the Control Center (see Section 2.2), sends a message to the system administrator through the GUI notifying them of the presence of new machines in the Blutopia

Network, as shown in Figure 2. Afterwards, whenever the user wishes, they can give the new machines names and assign roles to them. In effect, role assignment turns the new machines into components of the service provided at the user site, as defined in Section 2.1. Figure 3 shows Blutopia's interface for machine naming and role assignment. As can be seen in the figure, the machine was given the name "m5" and the role "dbserver" was selected. The other roles available in the example are "webserver" and "loadbalancer". Furthermore, note that Blutopia provides a pre-defined role called "spare", which is automatically assigned to all new machines during initial installation. After roles are assigned to a machine, a reboot command is issued by the Logic Engine. Once the reboot completes (typically in couple of minutes), the new service is fully-available and integrated with the rest of the cluster.

The Blutopia system provides support for installation and role assignment to multiple systems simultaneously with no additional overhead. Hence, for all practical purposes, the installation of a service with multiple components takes about the same time as installing a single component. The overall time for installation and role assignment is inherently related to the speed of the machines and network conditions. In our tests, a whole service with six components was brought on-line within less than three minutes.

3.2 Upgrade

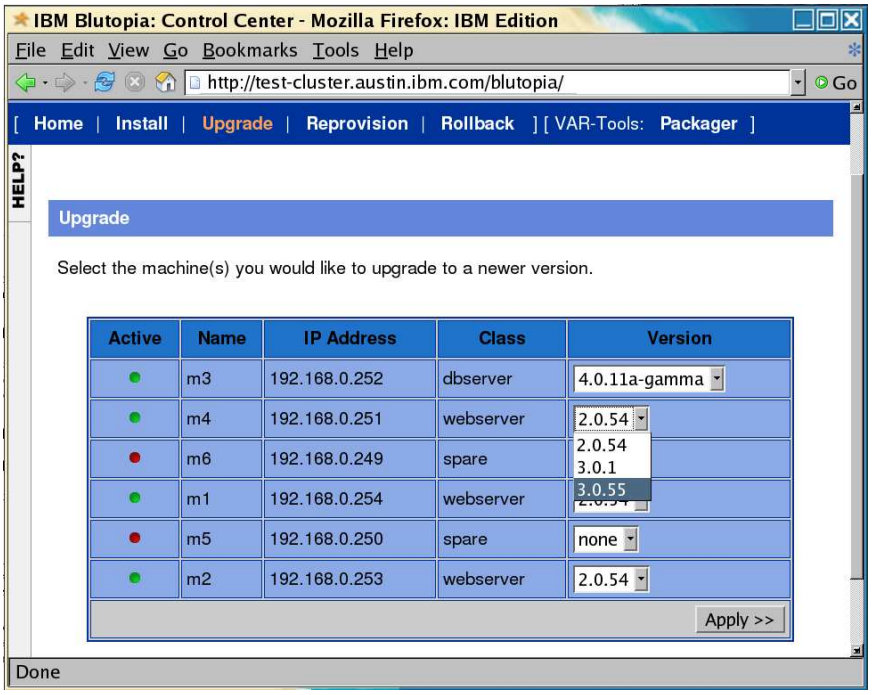

Figure 4: *Upgrade function: upgrading a component to a newer version.*

From time to time, value added resellers and regional system integrators may provide updated component images containing the latest bug fixes and security updates from an upstream software publisher. These updated images may be transported over the Internet or on conventional media such as CD-ROM or DVDs. The updated images need only be copied to the Blutopia server. Once updated images are available, Blutopia trivializes the process of upgrading a component to a newer version.

The system administrator selects an upgrade option for a given component from the Control Center GUI which provides a drop-down list with all currently available versions, as illustrated in Figure 4. If so desired, multiple components may be upgraded simultaneously.

The Blutopia Logic Engine manages upgrades by maintaining a strict order of single, linear dependencies. In other words, a new version of a role is directly dependent exclusively on only one previous version of the role. For example the version 2.0.54 of the role "Webserver" may be dependent on version 1.0.33. The way dependencies are generated will be examined in Section 3.5.

3.3 Re-provision

We define the re-provision of a machine as changing the role currently assigned to it. For instance, imagine an on-line three-tier Internet service composed of two Web servers, four application servers, and one database server. Suppose that a more powerful machine is purchased to host the database. Eventually, after the new machine is set up to run the database, the machine previously designated as the database server might be re-provisioned so that it can be re-integrated into the service as either a Web server, or an application server. Yet another situation for which re-provision is suitable is alleviating a bottleneck tier in on-line Internet services. In the example of the three-tier Internet service, if the second tier (application servers) becomes overloaded and there are no extra machines to be deployed, re-provisioning one of the Web servers by transforming it into an application server, might significantly improve the overall system performance.

Re-provision, as implemented in Blutopia, is actually a generalization of assignment of a role to a new machine (see Section 3.1). During installation of a new machine, when the system administrator assigns a role to it she is, in effect, replacing the default "spare" role with another role. Similarly, in re-provisioning, the system administrator replaces the role of a machine that has been previously installed. While it is not currently implemented in the prototype, support for migrating components between physical machines through re-provisioning is planned for the near future.

3.4 Rollback

Blutopia provides the administrator with the ability to rollback any server to a previously saved checkpoint. As mentioned earlier, all Blutopia operations result in an instantaneous checkpoint of the system prior to the action being taken. As such, rollback can be used to back-out a component upgrade, reverse a re-provision operation, and correct operator error.

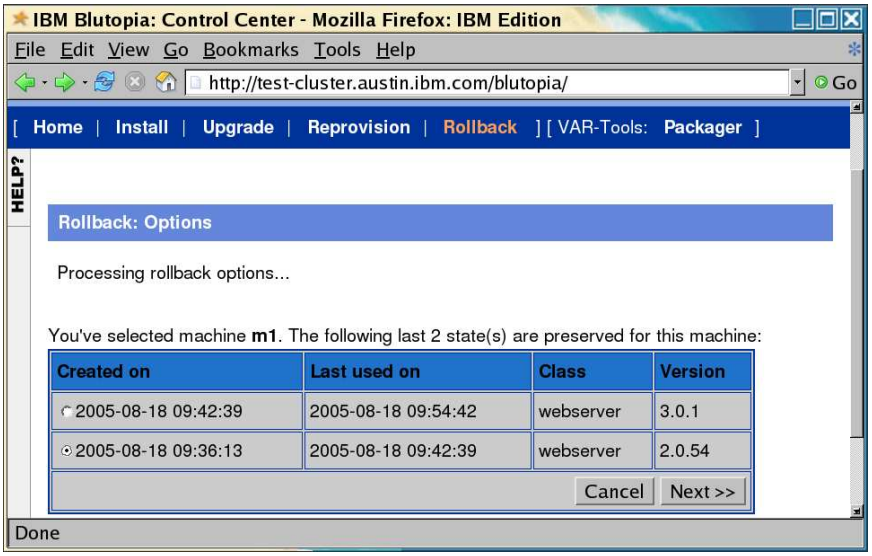

Figure 5: *Rollback function: selecting a previous state.*

To rollback a system, the user needs to choose a historical state from a list of the recorded checkpoints for that particular node. Figure 5 illustrates what the list of saved states looks like. In the figure, the user selected a state in which the component had been assigned the role "webserver" whose version was 2.0.54. Note that each entry of the list has two additional fields: one time stamp that indicates when the component was assigned the corresponding role, and another time stamp showing when the component was last used with that role. It is possible to have a pair of role and version appearing in the list of states multiple times, provided that the entries have different time stamps. This same facility could be used to checkpoint content and manual configuration changes or even provide nightly snapshots for backup purposes.

3.5 Packager

The packager function allows a software publisher, value added reseller, or even the local IT staff to create new roles, customized roles, or new versions of a role.

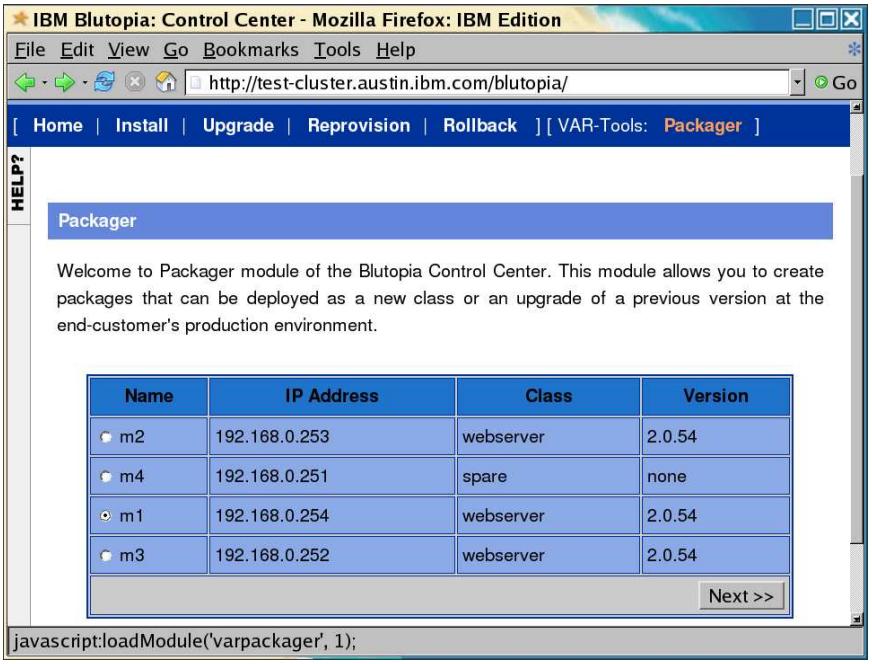

Figure 6: *Packager function: selecting a machine.*

Creating upgrades. The first step in the creation of an upgrade is to choose a machine to derive the upgrade from; eligible machines are those running the role for which an upgrade is desired. To illustrate how upgrade creation works, let us assume that we want an upgrade for the "webserver" role whose current version is 2.0.54. The administrator or software publisher would log on the machine running "webserver" version 2.0.54 and manually upgrade the software. After doing so, the administrator informs Blutopia that a new version is available on the chosen machine. This is a two step process: first, the machine previously operated upon has to be selected in Blutopia's GUI, as shown in Figure 6; second, the role name and version have to be given to Blutopia, as in Figure 7. When the Packager operation completes, the new version of the role will be available to the installation and upgrade operations.

Creating roles. The process of creating a new role is identical to that of upgrade creation, except that the publisher has to initiate it on a machine running the "spare" (default) role. This restriction is not actually required, but it simplifies role creation. A Live-CD utility disk may be provided in the future to allow creation of roles from systems outside of an existing Blutopia network.

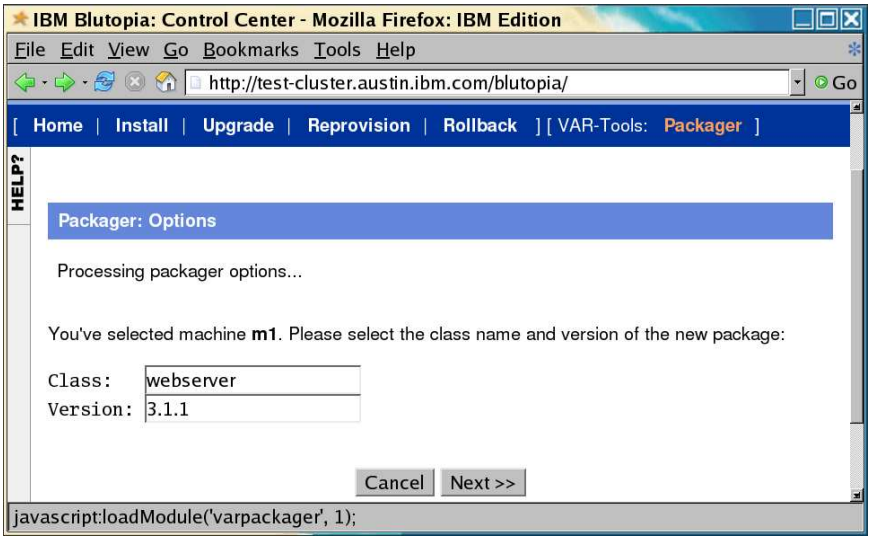

Figure 7: *Packager function: creating a new class or version.*

4 Implementation

This section describes the details underlying Blutopia's architecture. It gives some background information on Unionfs [14], the stackable file system which is the foundation of Blutopia, and discusses how it is used to execute the tasks discussed in the previous section.

4.1 Background: Unionfs

Unionfs is a stackable file system designed for the Linux kernel series 2.4 and 2.6 that lies between the VFS (Virtual File System) layer and the lower-level file systems — e.g. ext2, ext3, NFS, ISO9660, etc. It captures calls made to VFS and modifies the behavior of the corresponding operations before issuing them to the underlying file systems in order to provide the functionality of names pace unification. The main idea behind names pace unification is to recursively merge different directories (possibly belonging to different file systems) into a single unified view. In Unionfs terminology, each merged directory is called a *branch*, and the resulting merged file system is called a *union*. For instance, one could build a union based on both the root directory of an ext3 file system on disk, and the root directory of an ISO9660 file system on a CD. These two branches are then treated as a single file system.

To cope with the possibility of a given file name appearing in more than one branch, Unionfs associates a precedence level with each branch. Given a set of branches containing a conflicting file name, the branch with the highest precedence dictates what the file contents will be.

Although conceptually simple, names pace unification is a powerful feature that provides many benefits. In particular, Blutopia exploits Unionfs names pace unification in three ways: to share common base and role images, to obtain instantaneous system snapshots, and to provide fast re-provisioning of systems.

Blutopia provides each machine with access to a network root file system. While systems have the illusion of exclusive access to a unique root file system, parts of it are actually shared among all machines. Each newly detected machine is given a Unionfs file system comprised of two branches: one read-only branch containing the base operating system, and one read-write copy-on-write (CoW) branch. All machines share the read-only branch, whereas each one has its own CoW branch. Unionfs branches will be hereafter referred to as layers. A Unionfs file system, as used by Blutopia, can be thought of as a stack of layers where the topmost layer is the only read-write branch and, obviously, the one with highest precedence.

After the system administrator has assigned a role to a spare machine, Blutopia gives the machine a three-layer union with the following composition, from the bottom to the top: a base operating system layer, a role-specific layer, and a unique CoW layer. Such a stack can be seen on the left hand side of Figure 8. As one might have realized, all machines share the bottom-most layer; similarly, machines designated with a common role share the role-specific layer, yet each one has a unique CoW layer. After preparing the new union as a result of role assignment, Blutopia records the binding of the new union to the machine and reboots it so as to enforce the file system changes.

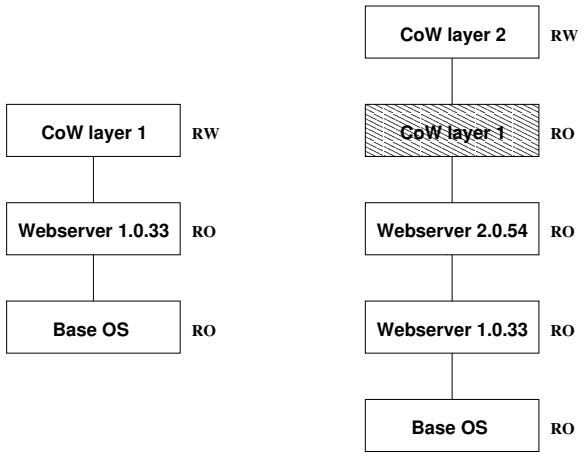

Figure 8: *Component running "Webserver" is upgraded from version 1.0.33 to 2.0.54.*

Upgrades of a component create a new union which differs from the base install in two aspects: (1) the role-specific layer corresponding to the new version is inserted between the role-specific layer of the previous version and the original CoW layer; (2) the previous CoW layer is marked as read-only, and a new CoW layer is pushed on top of the stack. Figure 8 illustrates a component that

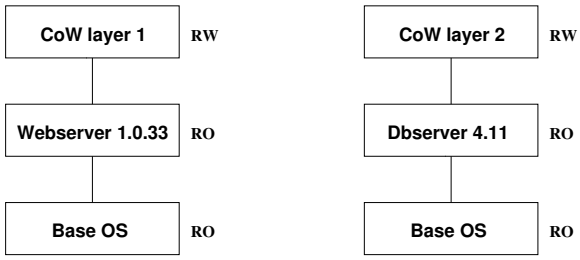

Figure 9: *Component running "Webserver" is re-provisioned as "Dbserver".*

runs the role "Webserver" being upgraded from version 1.0.33 to 2.0.54. The diagram on the left side shows the original Unionfs composition, and the resulting union is shown on the right side. Note the creation of a new CoW layer and the insertion of the role's new version layer.

It is worth mentioning the reasons why a new CoW layer is created and the previous one is kept as read-only in the union. Inserting a new CoW layer provides a safe rollback state (originally discussed in section 3.4), whereas preserving the old CoW layer ensures that all customizations and configuration changes made to the previous version will be kept, thereby relieving the system administrator from the burden of reconfiguring the upgraded software from scratch.

In a nutshell, upgrade leads to the creation of a union that contains a new CoW layer on top of any previous CoWs, followed by as many role-specific layers as the number of nodes in the dependency list of the new version, terminating with the base operating system layer. After the union is set up, the machine is rebooted.

As a result of re-provisioning, the machine being operated upon is given a Unionfs file system comprised of a new CoW layer, the role-specific layer, and the base operating system layer. Figure 9 depicts the re-provision operation on a machine originally assigned the role "Webserver". The diagram on the left side shows the Unionfs file system before re-provision, and the resulting union is shown on the right side, after the machine was assigned the role "Dbserver". Note the differences between the two CoW layers.

One should notice a fundamental difference between upgrade and re-provision. While the former operation always keeps older CoW layers in the union, the latter does not. During re-provision, keeping the previous CoWs in the new union is not important because the machine will be assigned a totally different role; on the other hand, preserving the CoWs after an upgrade has the benefit of preserving all configurations and customizations made to the previous version, which can be re-used by the system administrator in their entirety most of the time. For instance, if a component is upgraded after the software publisher advertised a new version whose configuration files and format are backwards compatible, the system administrator would not need to do any work besides a few mouse clicks to select the machine and the new version.

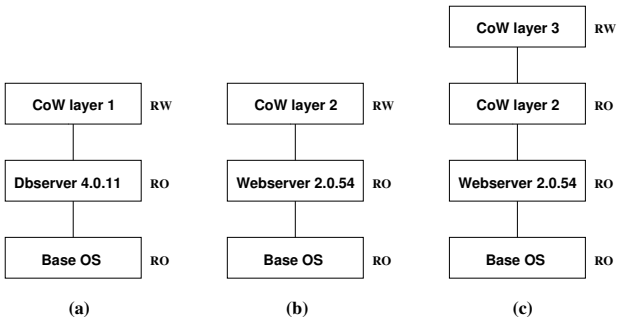

Figure 10: *Unions handled during rollback from "Dbserver" to "Webserver": (a) union before rollback; (b) union corresponding to the state chosen for rollback; (c) new union created as a result of rollback.*

In order to implement rollback, Blutopia records the state of a component right before the user performs an upgrade, re-provision, or rollback on it. The state of a component as recorded by Blutopia is defined by the following pieces of information: (1) the role and version of the component; (2) the configuration of the Unionfs file system used by that component, i.e., the names of the layers and their attributes — read-only or read-write; and (3) the actual contents of each Unionfs layer. One should notice that saving a component's state does not impose a severe space overhead, given that the base operating system layer (shared by all machines in the Blutopia Network) and any role-specific layer (shared by the machines assigned the same role) are not replicated, i.e., only one instance of such layers is kept. The contents of the aforementioned layers do not need to be saved for rollback purposes because those layers are not writable. The disk-space overhead due to rollback support stems from all CoW layers that are created in the system, but they tend to occupy far less space than do the shared layers. Blutopia never re-uses a CoW layer; every time a new CoW layer is needed for installation, upgrade, re-provision, or rollback, an empty directory is created and used as a new Unionfs layer.

Let us suppose that the component selected by the user is currently assigned the role "Dbserver" version 4.0.11, and that its current Unionfs file system has the composition shown in Figure 10(a). Moreover, let us assume that the state selection performed by the user is done as in Figure 5, and that the corresponding layers of the Unionfs file system are the ones shown in Figure 10(b). After the user chooses the state and applies the rollback function, the Logic Engine will do the following actions: (1) it adds an entry for the current state of the component — "Dbserver" version 4.0.11 — into the list of states that the component can be rolled back to; (2) it creates a new union based on the composition of the Unionfs file system that the component was using when the selected state was saved, i.e., a new union identical to the one in Figure 10(b); and (3) it inserts a new CoW layer on top of the newly created union, and transforms the previous CoW layer into read-only, the result

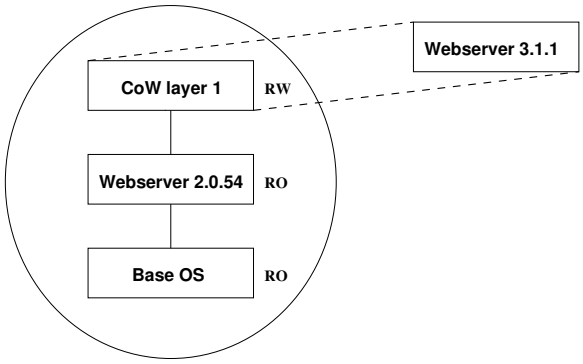

Figure 11: *An upgrade is created based on "Webserver" 2.0.54. The resulting version is named 3.1.1.*

being the union shown in Figure 10(c). The machine is then rebooted.

The step (3) described above ensures that the selected state will be preserved: all modifications made to it after rollback will be isolated in the new CoW layer. The benefit of this approach is to allow the user to rollback to any state multiple times.

The nature of stackable file systems makes the packaging of new roles and upgrades quite trivial. Upon receiving the role name and version from the user interface, Blutopia copies the contents of the CoW layer of the target machine where the upgrade was carried out, and stores it as a new role version available for future deployments, as depicted in Figure 11.

In essence, the CoW layer contains only the changes made to the file system since the machine was assigned the role "Webserver" 2.0.54. For that reason, Blutopia internally marks the new version of the role "Webserver" (in our example, 3.1.1) as dependent on the version from which it was derived (2.0.54, in our example). Whenever the version 3.1.1 is assigned to a machine, the new union created as a result will always contain the layer "Webserver" 3.1.1 on top of "Webserver" 2.0.54 because of the dependency (see Section 3.2).

5 Performance Analysis

The primary overhead of Blutopia is imposed by our use of stackable file systems. Long-lived components may accumulate dozens of layers from upgrades, rollbacks, and configuration changes. It is vital to understand how the depth of a union can impact component performance. Furthermore, since Blutopia is based on a centralized disk-storage, we must also evaluate the interplay between deep unions and remote disk access. To that end, in this section we present the results of running the widely used Postmark [6] benchmark, along with a simple stat micro-benchmark of our own

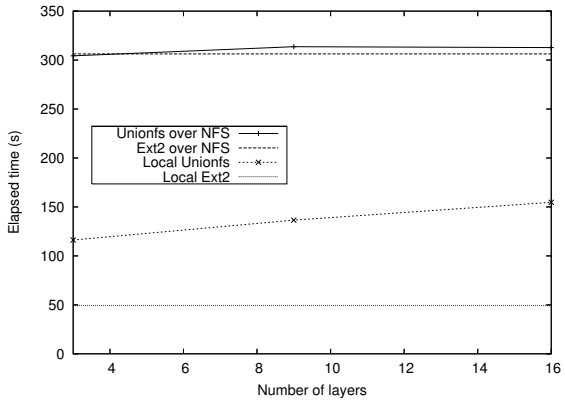

Figure 12: *Postmark: 10,000 files ranging from 500 and 1,024 bytes, 100,000 transactions.*

design.

All of our experiments were conducted with Blutopia server running on a server equipped with a single 1.6 GHz Intel Xeon processor, 4 GB of RAM, and 10K RPM IBM Ultrastar SCSI disks. This machine provided the centralized disk-storage by means of an NFS server and Unionfs version 1.0.13. The other machines belonging to the Blutopia network were disk less IBM HS20 blade servers equipped with two 2.8 GHz Intel Xeon processors and 1 GB of RAM. The Blutopia server and blades were connected through a Gigabit Ethernet network. All machines ran Red Hat Enterprise Linux release 4 with kernel version 2.6.12.2.

5.1 Postmark Benchmark

Postmark stresses the storage subsystem by creating an initial pool of files with a configurable range of sizes. It then randomly deletes files, creates new files, reads and writes data to the files. The intent of the workload is to mimic the constant use of multiple short-lived files which is common in electronic commerce sites.

Figure 12 shows the results of Postmark for our system. We configured Postmark to generate an initial pool of 10,000 files, whose sizes range from 500 to 1,024 bytes, and to perform 100,000 operations. The same random number generator seed was used throughout all benchmark runs to guarantee reproducibility. The graph shows how the runtime varies as we increase the number of Unionfs layers. Each point in the graph, corresponding to 3, 9, and 16 Unionfs layers, was determined by performing 20 runs of the benchmark, discarding the greatest and the smallest values, and taking the average of the 18 remaining samples. Before each experimental run, we warmed up the buffer cache so that disk-access time did not dominate the results. We configured Unionfs to manage file deletion by means of “whiteouts”. Also, note that the distribution of the

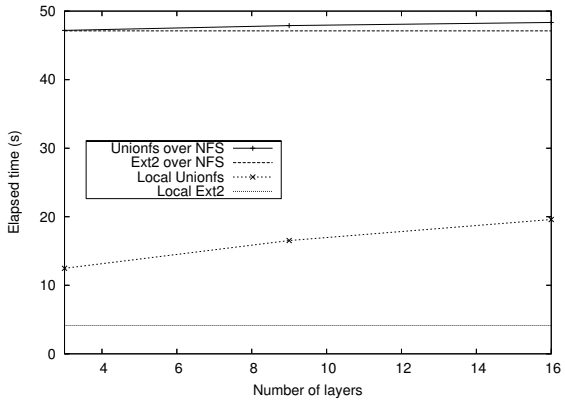

Figure 13: *Results for the stat micro-benchmark.*

original files among the Unionfs branches reflects the typical evolution of a component from initial installation (3 layers) to multiple upgrades (9 and 16 layers), but the files created and operated on by Postmark are obviously located on the topmost CoW layer, the only writable Unionfs branch.

There are two pairs of related curves in the graph. The curves on the bottom show the results of applying Postmark to the Ext2 file system and to Unionfs (on top of Ext2) locally on the Blutopia server machine. For 3 layers, Unionfs exhibits an overhead of 136% as compared with Ext2. The overhead increases linearly, with a subtle slope, as the number of layers increases, demonstrating Unionfs's reasonable scalability. Our result was different from that presented by Wright et al. [14] who claimed Unionfs overhead does not increase with the number of layers. They ran their Postmark with a slightly different configuration: 20,000 files, ranging from 500 to 10K bytes, and 200,000 transactions. We attempted to duplicate their configuration and discovered the larger the data set, the less visible the overhead (and its variation) becomes since the results are dominated by disk-access time.

The two curves on the upper portion of Figure 12 show the results of running Postmark on one blade server accessing Blutopia's Ext2 and Unionfs file systems through NFS over a Gigabit Ethernet network. This scenario is more representative since it portrays how the storage will be accessed by the Blutopia Network. The network latency dominates the results making the Unionfs overhead negligible. This outcome confirms that Unionfs will not pose any scalability problem for the remote storage-access approach taken by Blutopia.

5.2 Stat Micro-benchmark

Since Postmark was limited to writing and accessing files on the top layer of Unionfs (where they were created), we wrote a micro-benchmark that performs `stat` on a number of files at a

lower layer of the stack. The files to be operated on are supplied to the micro-benchmark from a pre-computed list; in particular, our data set is comprised of 202,014 files, spread over 1,097 directories, amounting to 3.9 GB. For runs with Unionfs the data set was completely placed on the second layer, counting bottom up.

The results for this micro-benchmark are presented in Figure 13. We show the accumulated time to perform the stat operations. As in the previously commented benchmark, we used 3, 9, and 16 Unionfs layers. The graph shows the same pairs of related curves that were shown in the previous graph, the one on the bottom being generated by running the micro-benchmark on the local Ext2 and Unionfs file systems of the Blutopia server. In this case, Unionfs demonstrates an overhead of 202% as compared with Ext2, and the overhead increases linearly with the number of layers. Running the micro-benchmark from a blade accessing the Blutopia Ext2 and Unionfs file systems via NFS leads to a negligible Unionfs overhead for the aforementioned reason: the result is dominated by the network latency.

6 Related Work

Systems management issues have started drawing the attention of the systems research community. Brown and Patterson have proposed a framework for providing an enterprise-wide undo functionality [2]. They implemented a proof-of-concept prototype applicable to an e-mail store hosted by a single machine. In their work, the focus is solely on the undo capability; the proposed tool does not address the system administration tasks themselves.

Another piece of research targeted at systems management is component validation [9]. The main idea behind validation is to make sure that each component operated upon by the system administrator is introduced into the service only after the system observes that it behaves as expected. Components under operation/validation are isolated in a virtual network; transitioning from the virtual to the real network and vice-versa is done transparently and without reconfiguration. The autonomic validation of software upgrades and configuration changes would be a powerful feature to add to Blutopia's existing workload. Integration of such a automatic administrative work flow is something we plan to explore in future work.

Whitaker *et al.* [13] propose debugging configuration errors by rolling back the system to a point in time where the error is not present. The main idea of their solution is to search the space of logged disk states in order to identify the cause of the error. This search is carried out by booting a virtual machine off each relevant state previously logged and determining if that state makes the system work. This approach exhibits some resemblance to Blutopia's rollback. They use a time-travel disk to log every file system change, whereas Blutopia relies on Unionfs CoW layers to get file

system snapshots.

Like Blutopia, there are other research efforts targeted at making system administration easier. The Collective [12, 3] is a computing utility that groups related machines as a single unit of system administration called a virtual appliance. Collective assigns virtual appliances to hardware automatically and dynamically. It is able to capture the configuration of an appliance and keep it when its software is updated, so that it can be transparently reapplied. Also, Ajmani *et al.* [1] proposed a methodology and an infrastructure to promote fully automatic software upgrades. Finally, the Microvisor system [7] provides a virtual machine infrastructure that allows system administrators to perform maintenance tasks on machines without requiring them to be taken out of the on-line service.

7 Future Work

In pursuing our grand vision of making Blutopia a truly autonomic systems management solution we foresee an exciting research agenda. In the next paragraphs we describe some avenues that we plan on exploring over the next few years.

Self-configuration. In the autonomic computing realm we envision component self-configuration as a key feature of Blutopia. A self-configuration framework would allow Blutopia to automatically and transparently reconfigure the whole service as machines are assigned roles, upgraded, and rolled back. We are primarily concerned with configuration related to the interaction between service components. It is obvious that if a number of machines are used to provide a service, they must interact with one another to exchange information. For instance, in a typical multi-tier Internet service, Web servers forward requests to application servers which, in turn, contact a database server; the replies flow in the opposite direction from the database server towards the Web servers.

Blutopia's self-configuration system should be generic enough to perform automatic configuration of all roles, even those that have yet to be created and published. As far as the end customer is concerned, nothing changes. The GUI given to the end customer remains the same; the available operations are installation, role assignment, upgrade, and rollback. Whenever roles "X" and "Y" are assigned, Blutopia would transparently modify the necessary configuration files to guarantee the appropriate interaction between machines running those roles.

Component validation. In some situations, the system administrator working at the customer site might want to perform actions not supported by Blutopia. For instance, a specialized system administrator might start changing configuration parameters to tune the performance of compo-

nents. While performing actions not supported by Blutopia, the user might make mistakes that cause the whole service to misbehave or to become unavailable. To cope with human mistakes we'd like to incorporate an automated validation infrastructure [9] as part of Blutopia. Before operating on a machine, the administrator would ask the Blutopia to take the machine out of service and into the validation environment. Once the user has performed the administrative actions, they would request the Blutopia to validate and re-deploy the component into the on-line service. The machine would then be automatically checked for correctness and re-integrated only if validation succeeds; otherwise the machine is automatically rolled back before being redeployed. As described in a previous work [9], the correctness checking could be done by driving the component with real workload and inspecting the output it produces. The separation between the on-line service and the validation environment can be accomplished by means of network virtualization. The main benefit of validation is to avoid exposing human mistakes to the on-line service.

Dynamic policy caching. Currently, the machines managed by Blutopia rely completely on the Network Store to access the file system. To be more precise, Blutopia is currently oblivious to the existence of local disks. Hence, another avenue to improve Blutopia performance and scalability would be to take advantage of local storage by using them for caching purposes. The rationale behind this idea is to avoid overloading the centralized Network Store and reduce access latency. This feature may be more valuable for medium-sized companies with larger cluster installations.

References

[1] S. Ajmani, B. Liskov, and L. Shrira. Scheduling and Simulation: How to Upgrade Distributed Systems. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, May 2003.

[2] A. B. Brown and D. A. Patterson. Undo for Operators: Building an Undoable E-mail Store. In *Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.

[3] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The collective: A cache-based system management architecture. In *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI '05)*, May 2005.

[4] J. J. Garret. Ajax: A new approach to web applications. February 2005.

[5] J. Gray. Why do Computers Stop and What Can Be Done About It? In *Proceedings of 5th Symposium on Reliability in Distributed Software and Database Systems*, Jan. 1986.

[6] J. Katcher. Postmark: A New Filesystem Benchmark. Technical Report TR3022, Network Appliance, 1997. http://www.netapp.com/tech_library/3022.html.

[7] D. E. Lowell, Y. Saito, and E. J. Samberg. Devirtualizable Virtual Machines - Enabling General, Single-Node, Online Maintenance. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, Oct. 2004.

[8] B. N. C. Matthew L. Massie and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7), July 2004.

[9] K. Nagaraja, F. Oliveira, R. Bianchini, R. P. Martin, and T. D. Nguyen. Understanding and Dealing with Operator Mistakes in Internet Services. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*, Dec. 2004.

[10] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do Internet Services Fail, and What Can Be Done About It. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS'03)*, Mar. 2003.

[11] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. Technical Report UCB//CSD-02-1175, University of California, Berkeley, Mar. 2002.

[12] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the Seventeenth Large Installation Systems Administration Conference (LISA '03)*, October 2003.

[13] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration Debugging as Search: Finding the Needle in the Haystack. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*, Dec. 2004.

[14] C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair. Versatility and unix semantics in namespace unification. *ACM Transactions on Storage (TOS)*, 1(4), November 2005.