Solaris Zones: Operating System Support for Consolidating Commercial Workloads

Daniel Price and Andrew Tucker – Sun Microsystems, Inc.

ABSTRACT

Server consolidation, which allows multiple workloads to run on the same system, has become increasingly important as a way to improve the utilization of computing resources and reduce costs. Consolidation is common in mainframe environments, where technology to support running multiple workloads and even multiple operating systems on the same hardware has been evolving since the late 1960's. This technology is now becoming an important differentiator in the UNIX and Linux server market as well, both at the low end (virtual web hosting) and high end (traditional data center server consolidation).

This paper introduces Solaris Zones (*zones*), a fully realized solution for server consolidation projects in a commercial UNIX operating system. By creating virtualized application execution environments within a single instance of the operating system, the facility strikes a unique balance between competing requirements. On the one hand, a system with multiple workloads needs to run those workloads in isolation, to ensure that applications can neither observe data from other applications nor affect their operation. It must also prevent applications from over-consuming system resources. On the other hand, the system as a whole has to be flexible, manageable, and observable, in order to reduce administrative costs and increase efficiency. By focusing on the support of multiple application environments rather than multiple operating system instances, zones meets isolation requirements without sacrificing manageability.

Introduction

Within many IT organizations, driving up system utilization (and saving money in the process) has become a priority. In the lean economic times following the post dot-com downturn, many IT managers are electing to adopt server consolidation as a way of life. They are trying to improve on typical data center server utilizations of 15-30% [1] while migrating to increasingly commoditized hardware. But the cost savings promised are not always realized [12]. Consolidation can drive down initial equipment cost, but it can also increase complexity and recurring costs in several ways. In our experience, this has made many system administrators reluctant to embrace consolidation projects. We believe that when implemented effectively, consolidation can free system administrators and IT architects to pursue higher service levels, better overall performance, and other long term projects. With an appropriate solution, greater specialization (and in turn, higher expertise) can be achieved; some administrators can focus on the maintenance of the physical platforms, and others can concentrate on the deployment of applications.

Administrators currently lack an all-in-one solution for server consolidation, as existing solutions require administrators to purchase, author, or deploy additional software. This paper explains how a server consolidation facility tightly integrated with the core operating system can provide answers to these problems.

Barriers to Consolidation

Consolidation projects face a variety of technical problems. First and foremost, applications can be mutually incompatible when run on the same server. In one real-world example, two poorly written applications at a customer site both wanted to bind a network socket to port 80. While neither application was a substantial resource user, the customer resolved the conflict by buying two servers! Applications can also be uncooperative when administrators wish to run multiple instances of the same application on the same node. For example, dependencies on running as a particular user ID can make it difficult to distinguish one running instance from another. Hard-coded log file locations or other pathnames can make it difficult to deploy two distinct versions of a particular application on the same node. At the highest level, solving this problem requires some form of *namespace isolation*, allowing administrators to make applications unaware of the presence of others. In the customer's example, deploying to two separate OS instances running on two separate systems provides complete namespace isolation, but the cost is very high.

A second technical problem faced by consolidators is security isolation. If multiple applications are to be deployed on a single host, what if there is a security bug in one of the applications? Even if each application is running under a different user ID (except for the applications that demand to run as root!), a wily intruder may be able to embark on a *privilege escalation*, in which the successively achieves higher levels of privilege until the entire system is compromised. If administrators are unable to assess the extent of the damage, the consolidated system might require a rebuild. Ideally, one should be able to create namespaces that are at

PAN EX1029
Palo Alto Networks v. Croga Innovations
IPR2024-01421
0001

fundamentally reduced levels of privilege: root in such an environment should be less powerful than the traditional UNIX root.

If a particular workload is compromised, a protective mechanism that wards off denial of service and resource exhaustion attacks against the rest of the system should be in place. Similarly, consolidation projects must address quality of service guarantees and must often be able to account for resource utilization for billing or capacity planning purposes. Existing resource management solutions address many of these requirements by providing resource partitioning, advanced schedulers, and an assortment of resource caps, reservations, and controls. However, these facilities do not typically offer security and namespace isolation and in cases where both are available, they have not been closely integrated.

A Comprehensive Solution

While a range of solutions exists to each of the problems described above, we discovered that no comprehensive consolidation facility was available as a core component of a commonly available operating system. We determined that deeper integration and a more "baked in" facility for consolidation would allow administrators to approach consolidation projects without the burden of designing the infrastructure to do so from component pieces. As a design goal, we established that administrators should need only a few minutes and a very few configuration choices to instantiate and start up a new application container, which we dubbed a *zone*. We also wanted our project to be a pure software solution that would work on a variety of hardware platforms, with the least possible performance tax.

At the highest level, zones are lightweight "sandboxes" within an operating system instance, in which one or more applications may be installed and run without affecting or interacting with the rest of the system. They are available on every platform on which Solaris 10 runs: AMD64, SPARC64, UltraSPARC, and x86. Applications can be run within zones with no changes, and with no significant performance impact for either the performance of the application or the base operating system.

Outline

This paper introduces zones and explains how we built a server consolidation facility directly into a production operating system, Solaris 10. The next sections describe related work, an overview of the facility, our design principles, and the architectural components of the project. The paper then explores specific aspects of the zones implementation, including resource management, observability and performance. We also discuss experiences to date with the facility.

Related Work

Much of the previous work on support for server consolidation has involved running multiple operating system instances on a single system. This can be done either by partitioning the physical hardware components into disjoint, isolated subsets of the overall system [5, 8], or by using virtual machine technologies to create abstracted versions of the underlying hardware [2, 7, 14]. Hardware partitioning, while providing a very high degree of application isolation, is costly to implement and is generally limited to high-end systems. In addition, the granularity of resource allocation is often poor, particularly in the area of CPU assignment. Virtual machine implementations can be much more granular in how resources are allocated (even time-sharing multiple virtual machines on a single CPU), but suffer significant performance overheads. With either of these approaches, the cost of administering multiple operating system instances can be substantial.

More recently, a number of projects have explored the idea of virtualizing the operating system's application execution environment, rather than the physical hardware. Examples include the Jails facility in FreeBSD [9] and the VServer project available for Linux systems [13]. These efforts differ from virtual machine implementations in that there is only one underlying operating system kernel, which is enhanced to provide increased isolation between groups of processes. The result is the ability to run multiple applications in isolation from each other within a single operating system instance. This should result in reduced administration costs, since there is only one operating system instance to administer (patch, backup, etc.); in addition, the performance overhead is generally minimal. Such technologies can also be used to create novel system architectures, such as the distributed network testbed provided by the PlanetLab project [3].

These technologies can be used as "toolkits" to assemble point solutions to virtualization problems, but at present they lack the comprehensive support required for supporting commercial workloads. The barrier to entry for administrators is also high due to the lack of tools and integration with the rest of the operating system.

Zones Overview

Zones provides a solution which virtualizes the operating system's application environment, and leverages the performance and sharing possible. At the same time, we have provided deeper and more complete system integration than is typical of such projects. We have been gratified when casual users mistake the technology for a virtual machine. This section provides a broad overview of the zones architecture and operation.

Figure 1 provides a block diagram of a system with four zones, representing a hypothetical consolidation. Zones *red*, *neutral* and *lisa* are *non-global zones*

running disjoint workloads. This example demonstrates that different versions of the same application may be run without negative consequences in different zones to match the consolidation requirements. Each zone can provide a rich (and different) set of customized services, and to the outside world, it appears that four distinct systems are available. Each zone has a distinct root password and its own administrator.

Basic process isolation is also demonstrated; a process in one non-global zone cannot locate, examine, or signal a process in another zone. Each zone is given access to at least one logical network interface; applications running in distinct zones cannot observe the network traffic of the other zones even though their respective streams of packets travel through the same physical interface. Finally, each zone is provided a disjoint portion of the file system hierarchy, to which it is confined.

The *global* zone encloses the three non-global zones and has visibility into and control over them. Practically speaking, the global zone is not different from a traditional UNIX system; root generally remains omnipotent and omniscient. The global zone always exists, and acts as the "default" zone in which all processes are run if no non-global zones have been created by the administrator.

We use the term *global administrator* to denote a user with administrative privileges in the global zone. This user is assumed to have complete control of the physical hardware comprising the system and the operating system instance. The term *zone administrator* is used to denote a user with administrative privileges who is confined to the sandbox provided by a particular non-global zone.

Managing zones is not complicated. Figure 2 shows how to create a simple, non-networked zone called *lisa* with a file system hierarchy rooted at /aux0/lisa, install the zone, and boot it. Booting a zone causes the init daemon for the zone to be launched. At that point, the standard system services such as cron, sendmail, and inetd are launched.

Design Principles

This section and the next examine the zones architecture in greater depth; before doing so it helps to examine the design principles we applied. First and foremost, our solution must *solve consolidation problems* such as those highlighted in the first section. The solution must provide namespace isolation and abstraction, security isolation, and resource allocation and management.

Second, the facility must *support commercial applications*: these are often scalable, threaded, highly connected to the network via TCP/IP, NFS, LDAP, etc. These applications come with installers and usually interoperate with the packaging subsystem on the host. More importantly, because these applications are often

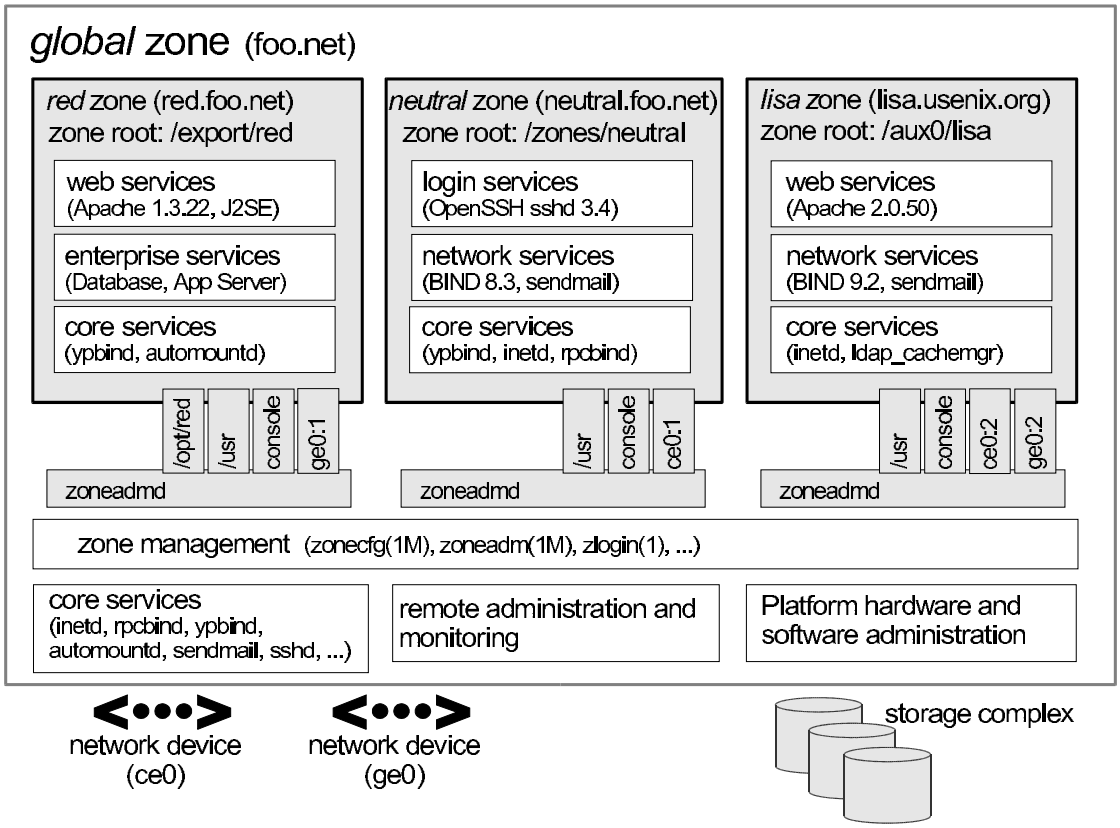

Figure 1: Zones block diagram.

opaque in operation, they should work "out of the box" within a zone whenever possible. Software developers should not need to modify applications, and administrators should not need to develop scripting wrappers or have a deep understanding of UNIX internals to deploy these applications. Similarly, administrators interacting with this facility should be *pilots, not mechanics*. As much as possible, system administrators should be able to view the application environment as a vehicle for deploying applications, not as a collection of parts to assemble. Setup should be simple and the entire system should look and feel as much like a normal host as possible. In addition, the solution should *enable delegation* wherever possible. The administrator of the global zone should be able to configure the overall system and delegate further control to zone administrators.

By *exploiting sharing and semantics* inside a single operating system instance, we can support a large number of application environments with relatively few resources. Operating in a shared environment means that monitoring application environments can be performed transparently. For example, from the pilot's seat, we should immediately be able to tell which process on the system (regardless of the application environment in which it runs) is using the most CPU cycles.

The solution must *scale and perform* with with the underlying platform. A 64-CPU application environment should "just work," as should the deployment of 20 environments on a 1-CPU system. Additionally, the solution should levy little or no performance tax on applications run inside it. Finally, minimal performance impact should be present on a system with no application environments.

To address these design principles, we divided the zones architecture into five principal components.

- A state model that describes the lifecycle of the zone, and the actions that comprise the transitions.
- A configuration engine, used by administrators to describe the future zone to the system. This allows the administrator to describe the "platform," or those parameters of the zone that are controlled by the global administrator, in a persistent fashion.
- Installation support, which allows the files that make up the zone installation to be deployed into the *zone path*. This subsystem also enables patch deployment and upgrades from one operating system release to another.
- The *application environment*, the "sandbox" in which processes run. For example, in Figure 3 each zone's application environment is represented by the large shaded box.
- The virtual platform, comprised of the set of platform resources dedicated to the zone.

We'll explore these subsystems in more depth in subsequent sections.

Zones State Model

A well-formed, observable state model that describes the zone lifecycle is an important part of the

```
# zonecfg -z lisa 'create; set zonepath=/aux0/lisa'
# zoneadm list -vc
  ID NAME             STATUS         PATH
   0 global           running        /
   - lisa             configured     /aux0/lisa

# zoneadm -z lisa install
Constructing zone at /aux0/lisa/root
Copying packages and creating contents file
...
# zoneadm list -vc
  ID NAME             STATUS         PATH
   0 global           running        /
   - lisa             installed      /aux0/lisa

# zoneadm -z lisa boot
# zoneadm list -vc
  ID NAME             STATUS         PATH
   0 global           running        /
   7 lisa             running        /aux0/lisa

# zlogin lisa
[Connected to zone 'lisa' pts/7]
zone: lisa
# ptree
1716   /sbin/init
1769   /usr/sbin/cron
1775   /usr/lib/sendmail -Ac -q15m
1802   /usr/lib/ssh/sshd
...
```

Figure 2: Zones administration.

pilot model design principle; it makes it easier for administrators to manage the zones present on the system. Figure 3 illustrates the zone state model. While this is of interest to the global administrator, zone administrators need not be aware of these states. A zone can be in one of four primary states, or in one of several secondary, or transitional, states:

CONFIGURED: A zone's configuration has been completely specified and committed to stable storage.

INSTALLED: Based on the zone's configuration, a unique root file system for the zone has been instantiated on the system.

READY: At this stage, the virtual platform for the zone has been established: the kernel has created the zsched process, network interfaces have been plumbed, file systems mounted, and devices configured. At this point, there are no user processes associated with the zone (zsched is a system process, and lacks a user address space).

RUNNING: The init daemon has been created and appears to be running. init will in turn start the rest of the processes that comprise the application environment.

SHUTTING DOWN: The zone is transitioned into this state when either the global or non-global zone administrator elects to reboot or halt the zone. The zone remains in this state until all user processes associated with the zone have been destroyed.

DOWN: The zone remains in this state until the virtual platform has been completely destroyed: filesystems and NFS shares are unmounted, IPC objects destroyed, network interfaces unplumbed, etc. At that point the zone returns to the INSTALLED state.

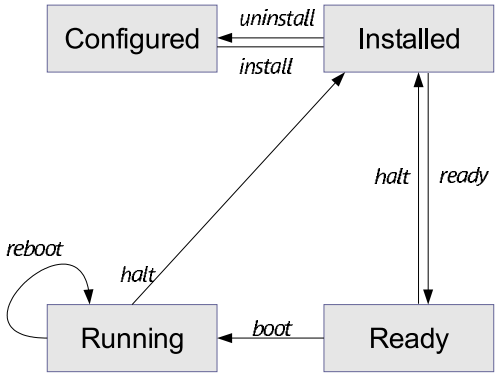

Figure 3: Zones state model.

Configuration Engine

Zones present a simple and concise configuration experience for system administrators. A command shell, zonecfg, is used by the global administrator to configure the zone's properties and describe the zone to the system. The tool can be used in interactive mode or scripted to create a new zone or edit existing zones. The configuration includes information about the location of the zone in the file system, IP addresses, file systems, devices, and resource limits. The zone configuration is retained by the system in a private repository (presently, an XML file), and keyed by zone name.

The design of zonecfg was challenging: Zone configurations can be complex, but we wanted to instantiate a new zone with a minimum number of commands and without having to navigate through a complex configuration file. Ultimately, the only mandatory parameter is the zonepath – the location in the file system where the zone should be created.

Installation Support

The zones installer is an extension to the Solaris install and packaging tools. An important goal was to be able to create a zone on an existing system, without needing to consult installation media. Binary files such as /usr/bin/ls can simply be copied from the global zone, or imported to the zone using a loopback mount to save disk space.

Files which are customizable by an administrator, such as /etc/passwd, must not be copied from the global zone into the zone being installed. Such files must be restored to their "factory default" condition. In order to accomplish this, the installer archives private, pristine copies of such volatile and editable system files when the global zone itself is installed or upgraded. The zones installer uses these archived versions when populating zones.

Because the zone installer is package-aware, the end result of zone installation is a virtual environment with an appropriately populated package database. This means that packaging utilities such as pkgadd can be used by the zone administrator to add or patch unbundled or third-party software inside the zone while also allowing the global administrator to the correctly upgrade and patch the system as a whole.

Application Environment

The application environment forms the core of the zones implementation. Using the facilities it provides, other subsystems such as NFS, TCP/IP, file systems, etc. have been "virtualized," that is, rearchitected to be compatible with the zones design.

At the most basic level, the kernel identifies specific zones in the same fashion as it does processes, by using a numeric ID. The *zone ID* is reflected in the cred and proc structures associated with each process. The kernel can thus easily and cheaply determine the zone membership of a particular process. This mapping is at the heart of the implementation. We have also found that virtualizing kernel subsystems (for example, process accounting) is often not terribly difficult if the subsystem's global variables are lifted up into a per-zone data structure (in other cases, such as TCP/IP, the virtualization required is more pervasive).

The process of booting the application environment is similar to the late stages of booting the operating system itself. In the kernel, a special process,

zsched, is created. This mimics the traditional UNIX process 0, sched. When seen from inside a zone, zsched is at the root of the process tree. zsched also acts as a container for a variety of per-zone data that is hard to express in other ways. RPC thread pools and other per-zone kernel threads, as well as resource controls and resource pool bindings, are handled in this fashion. Next, the init daemon is formed, associated with the zone, and exec'd to set it running in userspace; init then initiates the process of starting up other services that make the zone behave like a standalone computer system.

Zones are also assigned unique identities. The zone name, which is used to label and identify the zone, is assigned by the global administrator. Control of the node name, RPC domain name, Kerberos configuration, locale, time zone, root password and name service configuration is entirely delegated to the zone administrator. When a zone is first booted, the zone administrator is stepped through the process of setting up this configuration via an interactive tool.

Security concerns are central to the design of the application environment. Fundamentally, a zone is less powerful than the global environment, because zones take advantage of the fine-grained privilege mechanism available in Solaris 10 [11]. This mechanism changes the traditional all-or-nothing "super-user" privilege model into one with distinct privileges that can be individually assigned to processes or users.[1] A zone runs with a reduced set of privileges, and this helps to ensure that even if a process could find a way to escape namespace isolation enforced by the zone, it would still be constrained from escalating to higher privilege. For example, writing to /dev/kmem requires all privileges. All non-global zone processes and their descendants have fewer than all privileges, and are constrained from ever achieving all privileges, so the kernel will never allow such a process to write to /dev/kmem. The namespace isolation facilities provided by zones coupled with privilege containment provide a sound double-hulled architecture for secure operation. Although the set of privileges available in a zone is currently fixed, we plan to make this configurable in the future; this will allow administrators to create special-purpose zones with only the minimal set of privileges needed to run a particular service.

One significant design challenge the project faced was: how can we cross the boundary between global and non-global zones in a safe fashion? We authored the zlogin utility to allow global administrators to descend into specific zones; this command is modeled after familiar utilities such as rlogin. The process of transferring a running process from one zone to another is complex, and was a challenging aspect of the implementation. We took care to prevent any data from "leaking" from the global zone into non-global zones; this required sanitization of parent process IDs, process group IDs, session IDs, credentials, fine-grained privileges, core file settings, and other process model-related attributes. Processes whose parent process lies outside the zone (as is the case with zlogin to a zone) are faked within the zone to have zsched's PID as their parent process ID. Similarly, signals sent from the global zone to non-global zone processes appear to originate from zsched.

Virtual Platform

The virtual platform is the "bottom half" of a zone. Conceptually, it is comprised of the physical resources that have been made available to the zone. The virtual platform is also responsible for boot, reboot and halt, and is managed by the zoneadmd daemon.

The virtual platform takes a snapshot of the zone configuration from the configuration engine and follows the plan it provides to bring the zone into the READY state. This involves directing the kernel to create the central zone_t data structure and the zsched kernel process, setting up virtual network interfaces, populating devices, creating the zone console, and directing any other pre-boot housekeeping.

[1] This is similar to the capability feature available in Linux.

```
# zlogin -C lisa
[Connected to zone 'lisa' console]

lisa console login: root
Password:

# reboot
Aug 13 14:44:07 lisa reboot: rebooted by root

[NOTICE: Zone rebooting]

SunOS Release 5.10 Version s10_65 64-bit
Copyright 1983-2004 Sun Microsystems, Inc. All rights reserved.
Use is subject to license terms.
Hostname: lisa
NIS domain name is usenix.org

lisa console login: ~.
[Connection to zone 'lisa' console closed]
#
```

Figure 4: Zones console.

Uniquely, the zone console can exist even before the zone is in the ready state. This mimics a serial console to a physical host, which can be connected even when a machine is halted, and it provides a familiar experience for administrators. The console itself is a STREAMS driver instance that is dynamically instantiated as needed. It shuttles console I/O back and forth from the zone (via /dev/console) to the global zone (via /dev/zcons/<zonename>/masterconsole). zoneadmd then acts as a console server to the zlogin -C command. Figure 4 shows a typical console session. We found that the zone pseudo-console was a key to helping users see that a zone is a substantially complete environment, and perhaps more importantly, a *familiar* environment.

Virtualization of Specific Subsystems

One of the principal challenges of the zones project was making decisions about the "virtualization strategy" for each kernel subsystem. Generally, we sought to allow the global administrator to observe and control the entire system. But this was not always possible due to API restrictions (for example, APIs dictated by a particular standard), implementation constraints, or other factors. The next sections detail the virtualization that was required for each primary kernel subsystem.

Process Model

One of the basic principles of zones is that processes in non-global zones should not be able to affect the activity of processes running within another zone. This also extends to visibility; processes within one (non-global) zone should not even be able to see processes outside that zone, and by extension should not be able to observe the activity of such processes. This is enforced by restricting the process ID space exposed through the /proc file system and process-specific system calls such as kill, priocntl, and signal. If the calling process is running within a non-global zone, it will only be able to see or affect processes running within the same zone; applying the operations to process IDs in any other zone will return an error. The error code is the same as the one returned when the specified process does not exist, to avoid revealing the fact that the selected process ID exists in another zone. This policy also ensures that an application running in a zone sees a consistent view of system objects; there aren't any objects that are visible through some means (e.g., when probing the process ID space using kill) but not others (e.g., /proc).

The dual role of the global zone, acting as both the default zone for the system and as the nexus of system-wide administrative control, raises some interesting issues. Since applications within the zone have access to processes and other system objects in other zones, the effect of administrative actions may be wider than expected. For example, service shutdown scripts often use pkill to signal processes of a given name to exit. When run from the global zone, all such processes in the system, regardless of zone, will be signaled.

On the other hand, the system-wide scope is often desired. For example, an administrator monitoring system-wide resource usage would want to look at process statistics for the whole system. A view of just global zone activity would miss relevant information from other zones in the system that may be sharing some or all of the system's resources. Such a view is particularly important when the use of relevant system resources such as CPU, memory, swap, and I/O is not strictly partitioned between zones using resource management facilities.

We chose to allow any processes in the global zone to observe processes and other objects in non-global zones. This allows such processes to have system-wide observability. The ability to control or send signals to processes in other zones, however, is restricted by a fine-grained privilege, PRIV_PROC_ZONE. By default, only the root user in the global zone is given this privilege. This ensures, for example, that user tucker, whose user ID in the global zone is 1234, cannot kill processes belonging to user dp, whose user ID in the lisa zone is also 1234. Because different zones on the same system can have completely different name service configurations, this is entirely possible. The root user can also drop this privilege, restricting activity in the global zone to affect only processes in that zone.

Accounting and Auditing

Process and workload accounting provide an excellent example of both the challenges and opportunities for retrofitting virtualization into an existing subsystem. Accounting outputs a record of each process to a file upon its termination. The record typically includes the process name, user ID, exit status, statistics about CPU usage, and other billing-related items. The UNIX System V accounting subsystem, which remains in wide usage, employs fixed size records that cannot be extended with new fields. Thus, we modified the system so that accounting records generated in any zone (including the global zone) only contain records pertinent to the zone in which the process executed. System V accounting can be enabled or disabled independently for each zone.

In addition, since Solaris 8, the system has provided a modernized "extended accounting" facility, with flexible record sizes. We modified this so that records are now tagged with the zone name in which the process executed, and are written *both* to that zone's accounting stream and to the global zone; this provides an important facility for consolidation, and, uniquely, the ability to account in detail for the activity of the application environment. The set of data collected, the location of the accounting record files, and other accounting controls may all be configured independently per-zone.

The Solaris security auditing facility has been similarly updated with the addition of a zonename

token. An audit record describes an event, such as writing to a file, and the stream of audit records is written to disk and may be processed later. Each zone can access the appropriate subset of the audit trail, and the global zone can see all audit records for all zones. Because the global zone can track audit events by zone name, a complete record of auditable events can be generated per-zone. We think this represents an exciting possibility for intrusion detection and analysis.

IPC Mechanisms

Local inter-process communication (IPC) represents a particular problem for zones, since processes in different (non-global) zones should normally only be able to communicate via network APIs, as would be the case with processes running on separate machines. It might be possible for a process in the global zone to construct a way for processes in other zones to communicate, but this should not be possible without the participation of the global administrator.

IPC mechanisms that use the file system as a rendezvous, such as pipes, STREAMS, UNIX domain sockets, doors, and POSIX IPC objects, fit naturally into the zone model without modification since processes in one zone will not have access to file system locations associated with other zones. Because the file system hierarchy is partitioned, there is no way for processes in a non-global zone to achieve rendezvous with another zone without the involvement of the global zone (which has access to the entire hierarchy).

The System V IPC interfaces allow applications to create persistent objects (shared memory segments, semaphores, and message queues) for communication and synchronization between processes on the same system. The objects are dynamically assigned numeric identifiers that can be associated with user-defined keys, allowing usage of a single object in unrelated processes. Objects are also associated with an owner (based on the effective user ID of the creating process unless explicitly changed) and permission flags that can be set to restrict access when desired. In order to prevent sharing (intentional or unintentional) between processes in different zones, a zone ID is associated with each object, based on the zone in which the creating process was running at time of creation. Non-global zone processes are only able to access or control objects associated with the same zone. An administrator in the global zone can still manage IPC objects throughout the system without having to enter each zone. The key namespace is also virtualized to be per-zone, which avoids the possibility of key collisions between zones.

Networking

As discussed earlier, each zone is configured with one or more IP addresses. For each address assigned, a logical network interface is created in the global zone when the zone is readied. This address is then assigned to the zone. The system as a whole looks like a traditional multi-home server, but internally the IP stack partitions the networking between zones in much the same way as it would be partitioned between separate servers. From the perspective of an external network observer, a system with booted zones appears to be set of separate servers.

Each IP address and its associated logical interface are dedicated for use by the assigned zone. Only processes within the zone can send packets from that address or receive packets sent to that address. Logical interfaces can share a physical network interface, however, so depending on how the zones are configured, different zones may wind up sharing network bandwidth on a single physical interface. The isolation of network traffic means that services such as sendmail, Apache, etc., can be run in different zones without worrying about IP port conflicts.

Applications in different zones on the same system can communicate using conventional networking, just as applications on different systems can communicate. This traffic is "short-circuited" within the IP stack rather than sending data over the wire, minimizing the communication overhead. One drawback is that existing firewalling products are not able to filter or otherwise act on cross-zone traffic, as it is handled entirely within IP and is not visible to any underlying firewalling products. We hope to remedy this in the future.

Sending and receiving broadcast and multicast packets is supported within any zone. Inter-zone broadcast and multicast is implemented by replicating outgoing and incoming packets as necessary, so that each zone that should receive a broadcast packet or each zone that has joined a particular multicast group receives the appropriate data.

Access to the network by non-global zones is restricted. The standard TCP and UDP transport interfaces are available, but some lower level interfaces, such as raw socket access (which allows the creation of IP packets with arbitrary contents) and DLPI are not. These restrictions are in place to ensure that a zone cannot gain uncontrolled access to the network, where it might be able to behave in undesirable ways. For example, a zone cannot masquerade as a different zone or host on the network. Access to ICMP is also supported, allowing popular utilities such as traceroute and ping to work properly.

The zones facility also provides support for manual configuration of IPv6 addresses, with support for automatic addressing planned for the future. Because much of the TCP/IP infrastructure is shared between all zones, some functionality is automatically supported and can be configured on behalf of a zone by the global administrator. For example, if IP Multipathing is configured within the global zone, the logical interfaces associated with a failed physical interface are automatically moved to a configured alternate interface. The individual zones do not need any

configuration to support this, and are not even aware of the failure.

IPsec and IPQoS facilities can be configured on behalf of a zone by the global administrator; in the future we hope to allow global administrators to delegate some of this configuration to non-global zones. It would also be convenient to provide DHCP client support so that the global zone could request IP addresses for non-global zones from a DHCP server, and work to incorporate this support is underway.

File Systems

We have seen that zones are rooted at a particular point in the file system. This is implemented in a fashion similar to the chroot system call, although that call's well known security limitations [6] are avoided and the zone is not escapable. Because a different mechanism is used, use of chroot is even possible within a zone.

When the zone boots, the configuration engine is consulted for a list of file systems to mount on behalf of the zone. These can include storage-backed file systems as well as pseudo-file systems. In particular, lofs, the Solaris loopback file system, provides a useful tool for constructing a file system namespace for a zone. It can be used to mount segments of a file system in multiple places within the namespace. For example, the /usr file system is typically loopback mounted read-only beneath the zone root. This results in a high degree of sharing of storage, and a freshly installed zone requires only about 60 MB of disk space. The use of loopback mounts also results in the sharing of process text pages in the buffer cache, further decreasing the impact of running large numbers of zones. However, this approach adds substantial complexity to the design and implementation of the packaging tools. For example, the zone installation software must be aware that a particular file system object such as /usr/bin/ls will be available, but it will not have to be copied to the zone's /usr file system.

Mounts require special handling as well. In Solaris /etc/mnttab is itself a mounted file system that, when mounted, exports the typical /etc/mnttab file. The mnttab handling code was modified so that each zone sees only the mounts accessible by it. As usual, the global zone can see everything.

A key security principle is that the global zone users should not be able to traverse the file system hierarchy of non-global zones. Allowing this would enable unprivileged users in the global zone to collaborate with root users in non-global zones. For example, a zone's root user might mark a binary in a zone setuid root, and collaborate with a non-root user in the global zone, who could then run the binary and gain superuser privileges. As such, the the zones infrastructure enforces that the zone root's parent directory be owned, readable, writable, and executable by root only. We were also careful to prevent zones components such as zlogin and zoneadmd from ever accessing files residing within zones, in order to avoid "traps" that might have been placed by privileged software within the zone.

Devices

A limited set of devices, accessible via the /dev hierarchy, are available to zones by default. Additionally, some devices required additional virtualization to support zones. The syslog device is a good example: each zone has a distinct /dev/log device with a separate message stream, so that syslog(3C) messages are delivered to the syslogd in the zone that generated them.

Administrators can use the configuration engine to add additional devices to the zone as needed. This carries additional security risk because device interfaces are relatively unconstrained. A single device driver can form its own subsystem of APIs and semantics. For example, writing to a disk, writing to /dev/null, and writing to /dev/kmem all have completely different effects and security implications. As a general principle, we discourage the placement of physical devices into zones, as there is wide opportunity for mischief. For example, disks or disk partitions can be assigned to a zone, but the preferred method is for the global administrator to assign only file systems, which provide more uniform, auditable semantics.

A driver bug or improperly guarded feature could allow a hacker to attack the kernel. As a result, all of the devices included in a zone by default were audited and tested for security problems. We also addressed more systemic security problems; for example, an *imported device node* may allow a hacker to attack the system. For this reason, zones lack the privilege to call mknod(2) to create device nodes. However, this problem is more pervasive. If a hacker caused an NFS server to export a device node that matched the major and minor number of /dev/kmem and caused the zone to mount this share, then the system could be compromised. To defend against this attack, all mounts initiated from within a zone are guarded by the nodevices mount option, which prevents the opening of device nodes present on the mount. Note that even without nodevices, such an attack would remain difficult, as the reduced privilege allotted to the zone does not allow writing to the kmem device under any circumstances.

A final category of attacks could be carried out against the software managing the /dev hierarchy that runs in the global zone as part of the virtual platform. In this case, both global and non-global zones require access to the /dev hierarchy. The solution is to build and manage /dev for the zone *outside* of the zone's file system hierarchy, and then use the lofs file system to loopback mount /dev into the zone. Additionally, the kernel prohibits the zone from making all but the most basic modifications to its /dev hierarchy. Permission, group, and ownership changes are permitted; other file system operations are not.

NFS

Virtualizing client-side NFS support presents a somewhat unique challenge. NFS is not only a file

system: It also has semantics that are dependent on the network identity (hostname, RPC domain, etc.) of the client. For example, an NFS share may be exported solely to a client with a specific host name. Since each zone has a separate network identity, NFS mounts in different zones on the same system must be handled separately. In particular, operations to file system mounts associated with a zone must have matching credentials. This allows lower-level code (such as the RPC transport code) to keep track of the zone associated with a specific operation, even if that operation is being performed asynchronously. As a consequence, NFS mounts in a non-global zone cannot be accessed from the global zone.

Another complication is the use of kernel threads. The Solaris NFS implementation maintains a pool of in-kernel threads to asynchronously read-ahead data before it is needed, which improves performance when large files are read sequentially. When multiple zones can be using NFS, the thread pools need to be maintained on a per-zone basis. This allows the number of threads in each pool to be managed independently (since different zones may have different requirements with respect to concurrency) and means that threads can be assigned credentials associated with the appropriate zone.

Resource Management

Most of the prior discussion has described the ways in which zones can be used to isolate applications in terms of configuration, namespace, security, and administration. Another important aspect of isolation is ensuring that each application receives an appropriate proportion of the system resources: CPU, memory, and swap space. Without such a capability, one application can either intentionally or unintentionally starve other applications of resources. In addition, there may be reasons to prioritize some applications over others, or adjust resources depending on dynamic conditions. For example, a financial company might wish to give a stock trading application high priority while the trading floor is open, even if it means taking resources away from an application analyzing overall market trends.

The zones facility is tightly integrated with existing resource management controls available in Solaris [10]. These controls come in three flavors: *entitlements*, which ensure a minimum level of service; *limits*, which bound resource consumption; and *partitions*, which allow physical resources to be exclusively dedicated to specific consumers. Each of these types of controls can be applied to zones. For example, a fair-share CPU scheduler can be configured to guarantee a certain share of CPU capacity for a zone. In addition, an administrator within a zone can configure CPU shares for individual applications running within that zone; these shares are used to determine how to carve up the portion of CPU allocated to the zone. Likewise, resource limits can be established on either a per-zone basis (limiting the consumption of the entire zone) or a more granular basis (individual applications or users within the zone). In each case, the global zone administrator is responsible for configuring per-zone resource controls and limits, while the administrator of a particular non-global zone can configure resource controls within that zone.

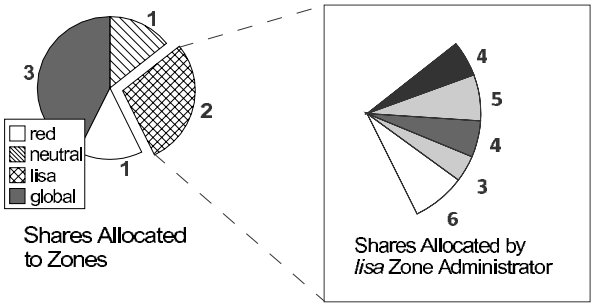

Figure 5: Zones and the fair-share scheduler.

Figure 5 shows how the fair-share CPU scheduler can be used to divide CPU resources between zones. In the figure, the system is divided into four zones, each of which is assigned a certain number of CPU shares. If all four zones contain processes that are actively using the CPU, then the CPU will be divided according to the shares; that is, the red zone will receive 1/7 of the CPU (since there are a total of seven shares outstanding), the neutral zone will receive 2/7, etc.. In addition, the lisa zone has been further subdivided into five *projects*, each of which represent a workload running within that zone. The 2/7 of the CPU assigned to the lisa zone (based on the per-zone shares) will be further subdivided among the projects within that zone according to the specified shares.

Resource partitioning is supported through a mechanism called *resource pools*, which allows an administrator to specify a collection of resources that will be exclusively used by some set of processes. Although the only resources initially supported are CPUs, this is planned to later encompass other system resources such as physical memory and swap space. A zone can be "bound" to a resource pool, which means that the zone will run only on the resources associated with the pool. Unlike the resource entitlements and limits described above, this allows applications in different zones to be completely isolated in terms of resource usage; the activity within one zone will have no effect on other zones. This isolation is furthered by restricting the resource visibility. Applications or users running within a zone bound to a pool will see only resources associated with that pool. For example, a command that lists the processors on the system will list only the ones belonging to the pool to which the zone is bound. Note that the mapping of zones to pools can be one-to-one, or many-to-one; in the latter case, multiple zones share the resources of the pool,

and features like the fair-share scheduler can be used to control the manner in which they are shared.

Figure 6 shows the use of the resource pool facility to partition CPUs among zones. Note that processes in the global zone can actually be bound to more than one pool; this is a special case, and allows the use of resource pools to partition workloads even without zones. Non-global zones, however, can be bound to only one pool (that is, all processes within a non-global zone must be bound to the same pool).

Performance and Observability

As noted in the related work section, one of the advantages of technologies like zones that virtualize the operating system environment over a traditional virtual machine implementation is the minimal performance overhead. In order to substantiate this, we have measured the performance of a variety of workloads when running in a non-global zone, when compared to the same workloads running without zones (or in the global zone). This data is shown in Figure 7 (in each case, higher numbers represent a faster run). The final column shows the percentage degradation (or improvement) of the zone run versus the run in the global zone. As can be seen, the impact of running an application in a zone is minimal. The 4% degradation in the time-sharing workload is primarily due to the overhead associated with accessing commands and libraries through the lofs file system.

Workload	Base	Zone	Diff (%)
Java	38.45	38.29	99.6
Time-sharing	23332.58	22406.51	96.0
Networking	283.30	284.24	100.3
Database	38767.62	37928.70	97.8

Figure 7: Performance impact of running in a zone.

We also measured the performance of running multiple applications on the system at the same time in different zones, partitioning CPUs either with resource pools or the fair share scheduler. In each case, the performance when using zones was equivalent, and in some cases better, than the performance when running each application on separate systems.

Since all zones on a system are part of the same operating system instance, processes in different zones can actually share virtual memory pages. This is particularly true for text pages, which are rarely modified. For example, although each zone has its own init process, each of those processes can share a single copy of the text for the executable, libraries, etc.. This can result in substantial memory savings for commonly used executables and libraries such as libc. Similarly, other parts of the operating system infrastructure, such as the directory name lookup cache (or DNLC), can be shared between zones in order to minimize overheads.

Observability Tools and Debugging

Because of the transparent nature of zones, all of the traditional Solaris /proc tools may be applied to processes running inside of zones, both from inside the non-global zone, and from the global zone. Additionally, numerous utilities such as ps, priocntl, ipcs, and prstat (shown in Figure 9) have been enhanced for zone-awareness.

In addition, we were able to enhance the DTrace [4] facility to provide zone context. In the following example, we can easily discover which zone is causing the most page faults to occur; see Figure 11.

We were pleasantly surprised when a customer pointed out to us that he could employ zones and DTrace together to better understand and debug a three-tiered architecture by deploying the tiers together on a single host in separate zones in the test environment, and making specific queries using DTrace.

Experience

Zones is an integrated part of the Solaris 10 operating system, which is still under development. Through pre-release programs, Zones has seen adoption both within Sun and by a variety of customers.

In one "pilot" deployment, Sun's IT organization has consolidated a variety of business applications. A four-CPU server with six non-global zones is hosting:

- **Zone 1** The web front-end (Java System Web Server version 6.1) to Sun's host database.
- **Zone 2** The web front-end (Java System Web Server version 6.0) to the 'orgtool' website, providing Sun's online organization chart.
- **Zone 3** The Oracle database that provides the backend for Sun's online organization chart.
- **Zone 4** A database reporting tool, which interfaces with Peoplesoft and corporate tax databases; this is monitored by software from TeamQuest.
- **Zone 5** A security hardened CVS server, using LDAP and DNS name services (the other zones use NIS).
- **Zone 6** A Sun-internal application that utilizes Apache and MySQL.

This consolidation is probably typical of both large and small IT organizations; a wide variety of heterogeneous software (including different versions

```
# dtrace -n 'vminfo:::as_fault{@[zonename]=count()}'
dtrace: description 'vminfo:::as_fault' matched 1 probe
^C

  global                4303
  lisa                  29867
```

Figure 8: Enhanced DTrace facility with zone context.

of the same application) is in play. In order to provide more predictable quality of service, the deployment team assigned different amounts of CPU shares to the various zones, to represent the relative importance of each workload.

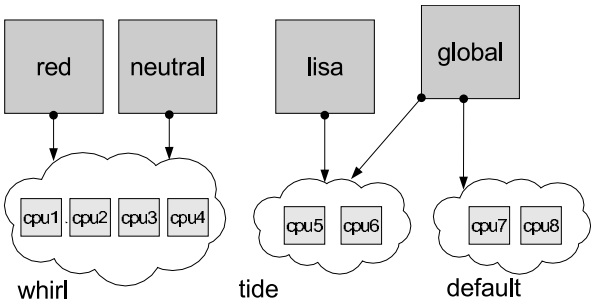

Figure 6: Zones and resource pools.

Security Experience

Any new systems architecture is rightfully viewed with suspicion by security conscious administrators; this was true during the project's development inside Sun. In order to better understand the security environment in which zones would need to operate, we created a non-global zone on an otherwise locked-down system. We then created a /SECRET file in the global zone and distributed the root password to the non-global zone far and wide within Sun, creating a "zones hacking" contest. This was extremely successful both for the contestants and the zones development team.

The system was compromised in the first few hours, using an exploit that we knew existed, but had considered very obscure. We realized that we had underestimated our adversaries. As we corrected the security problems our hackers found, we learned a lot about the sorts of attack techniques and vectors to expect. A positive result was that the reduced privileges associated with zone processes meant that attackers who managed to read the /SECRET file were usually unable to perform other sorts of mischief such as writing to /dev/kmem. We responded to the attacks by adding new system-level protections that prevent all of the exploits found. For example, mounts performed by a zone transparently have the nodevices mount option applied. This prevents using imported device files (for example, from an NFS share) as an attack vector.

Other Applications and Future Directions

In the course of developing this facility we considered the many other situations in which technologies such as Jails have been deployed. While the primary focus of the design is server consolidation, zones are well-suited for application developers, and may help organizations with large internal software development efforts to provide a multitude of "test systems." Many customers we have encountered spend substantial sums buying servers solely for this purpose.

Zones are also a useful solution for web hosting and other Internet-facing applications, in which creating a large number of application environments (perhaps administered by different departments) on modest hardware is important. We are also hopeful that advanced networking architectures such as PlanetLab will eventually include support for zones. At Sun, work is underway to prototype a version of Trusted Solaris based on the isolation provided by zones. We expect other novel uses for zones will emerge as researchers, developers, and administrators adopt them.

Moving forward, we know that networking poses key challenges to zones; groups of zones will cooperate in multi-tier architectures, and administrators will expect to be able to cluster, migrate, and failover zones from one host to another. Today these technologies are the unique domain of virtual machine solutions.

```
$ prstat -Z 10
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 12008 60028     191M  167M cpu18    1    0   0:00:31 1.1% ns-httpd/75
 28163 root       17M   10M sleep   59    0   4:40:37 0.5% ns-httpd/2
 12047 70002     296M  270M sleep   59    0   0:00:06 0.4% oracle/1
 10485 101       190M  101M sleep   59    0   1:37:20 0.2% webservd/82
 14058 root     6928K 5072K sleep   59    0   0:00:00 0.2% sshd/1
  1098 root     1736K  856K sleep   59    0   0:33:00 0.0% tqrtap.v9/1
   994 root     6848K 5512K sleep   59    0   0:23:08 0.0% tqwarp.ext/1
 12049 70002     296M  270M sleep    1    0   0:00:03 0.0% oracle/1
   804 root     4096K 3616K sleep   59    0   0:00:25 0.0% nscd/51

ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     2       39  374M  272M   1.6%   4:45:01 2.1% lisa
     1       55 8025M 7217M    45%   0:05:20 0.9% red
     0       56  212M  130M   0.7%   2:28:18 0.2% global
     3       36  463M  211M   1.3%   1:48:55 0.2% neutral
     6       47  940M  372M   2.2%   0:24:52 0.0% euro
     5       38  330M  246M   1.5%   0:10:47 0.0% end

Total: 261 processes, 1356 lwps, load averages: 0.12, 0.13, 0.14
```

Figure 9: Monitoring Zones Using prstat. The top half of this split view shows the individual processes consuming the most CPU cycles. The bottom half shows a view of CPU usage aggregated by zone.

One of the strengths of zones is its integration with the base operating system. To provide a comprehensive solution, pervasive integration with the wider systems management software stack is necessary, and will be a major part of our future work.

Availability

Solaris Zones, which has been productized under the name *N1 Grid Containers*, is an integrated part of the Solaris 10 Operating System. Pre-release versions are available as part of the Software Express for Solaris Program at http://www.sun.com/solaris/10. A clearinghouse of information about Solaris Zones is available at http://www.sun.com/bigadmin/content/zones. Documentation is available at http://docs.sun.com.

Conclusions

A successful server consolidation must drive down both initial and recurring costs and day-to-day complexity for all involved. Having less hardware to manage is an important goal. However, the ability to maintain less software – fewer operating system instances – can have an even greater impact on the long-term cost reduction realized. The savings in operating system licenses and service contracts alone can be substantial. The best consolidations also allow a site to split the platform administration and application administration tasks. This capability allows the IT organization to delegate certain work responsibilities while maintaining control over the server itself, so areas of specialization can be exploited.

Solutions that create a hierarchy of control on a single host *without sacrificing observability* allow IT organizations to act as infrastructure providers who can provide compute resources, not just networks and SANs. Simultaneously, application expertise can remain with the department deploying or developing the application.

Solaris Zones offer the first fully realized facility for server consolidation built directly into a commodity operating system. Zones provides the namespace, security and resource isolation needed to drive effective consolidation in the real world.

Author Information

Daniel Price received a Bachelor of Science with honors in Computer Science from Brown University. At Brown, he got his first taste of UNIX systems administration serving as a SPOC (systems programmer, operator, consultant) for the Computer Science Department. He joined Sun Microsystems' Solaris Kernel Development Group in 1998 and has worked on several I/O frameworks, the Zones project, and on administering the OS group's development labs.

Andrew Tucker is a Distinguished Engineer in the Solaris Data and Kernel Services group at Sun Microsystems. He has been at Sun since 1994 working on a variety of projects related to the Solaris operating system, including scheduling, multiprocessor support, inter-process communication, clustering, resource management, and server virtualization. Most recently, he was the architect and technical lead for Solaris Zones. Andrew received a Ph.D. in Computer Science from Stanford University in 1994.

Acknowledgments

The Zones project was the work of a much larger group of engineers and staff than the authorship of this paper represents, and we are grateful to everyone who worked to make this project possible. For this paper, Allan Packer and Priya Sethuraman provided performance data, and Karen Chau and Norman Lyons provided information about the pilot deployment of zones in Sun's IT organization. In addition, a number of people contributed substantially to the writing of this paper. Penelope Cotten provided extensive editorial assistance. Dave Linder, John Beck, David Comay, Liane Praza, Bryan Cantrill and our USENIX shepherd, Snoopy, provided valuable feedback that made the paper much improved.

References

[1] Andrzejak, Artur, Martin Arlitt, and Jerry Rolia, *Bounding the Resource Savings of Utility Computing Models*, Technical Report HPL-2002-339, HP Labs, 2002.

[2] Barham, Paul, et al., "Xen and the Art of Virtualization," *Proceedings of the 19th Symposium on Operating Systems Principles*, 2003.

[3] Bavier, Andy, et al., "Operating system support for planetary-scale services," *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[4] Cantrill, Bryan, Mike Shapiro, and Adam Leventhal, "Dynamic instrumentation of production systems," *USENIX Annual Technical Conference*, 2004.

[5] Charlesworth, Alan, et al., "The Starfire SMP interconnect," *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing*, 1997.

[6] *How to Break Out of a chroot() Jail*, http://www.bpfh.net/simes/computing/chroot-break.html .

[7] Gum, P. H., "System/370 Extended Architecture: Facilities for Virtual Machines," *IBM Journal of Research and Development*, Vol. 27, Num. 6, 1983.

[8] IBM Corp., *Partitioning for the IBM eServer p-Series 690 system*, http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/lpar.html .

[9] Kamp, Poul-Henning and Robert Watson, "Jails: Confining the Omnipotent Root," *Second International System Administration and Networking Conference (SANE 2000)*, May 2000.

[10] Sun Microsystems, Inc., *Solaris 9 Resource Manager Software*, http://wwws.sun.com/software/whitepapers/solaris9/srm.pdf .

[11] Sun Microsystems, Inc., *System Administration Guide: Security Services*, http://docs.sun.com/db/doc/816-4557 .

[12] Thibodeau, Patrick, ''Data Center, Server Consolidations Remain Top IT Issue,'' *Computerworld*, March 2004.

[13] http://www.linux-vserver.org .

[14] Waldspurger, Carl, ''Memory Resource Management in VMware ESX Server,'' *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, 2002.