STORM: Simple Tool for Resource Management

Mark Dehus and Dirk Grunwald – University of Colorado, Boulder

ABSTRACT

Virtualization has recently become very popular in the area of system engineering and administration. This is primarily due to its benefits, such as: longer uptimes, better hardware utilization, and greater reliability. These benefits can reduce physical infrastructure, space, power consumption, and management costs. However, managing a virtualized environment to gain those benefits is difficult and rife with details.

Through the use of a concept known as virtual appliances, the benefits of virtualization can be brought to organizations without sufficient knowledge or staff to install and support a complex virtual infrastructure. This same concept can also be used to provide cheap datacenter services to larger companies, or research facilities that are unable or unwilling to run a high performance computing environment.

In this paper, we describe Storm, a system designed to simplify the development, deployment and provisioning for common applications. The system is designed to be easy to configure and maintain. It can automatically react to changes in system load to deploy additional services and it dynamically powers client machines using IMPI controls to enhance energy savings. We demonstrate the utility of the system using scalable mail appliance.

Introduction

Virtualization has become very popular as a way of managing a large number of complex software systems. This is primarily due to its benefits, such as longer uptimes, better hardware utilization, and greater reliability enabled by the ability to move a virtual machine from one host computer to another. These benefits lead to reduced physical infrastructure/footprint, power consumption, and total cost of ownership [1].

However, managing virtual environments is complex; a number of management frameworks, both commercial and academic or open source projects, have been recently developed. These frameworks seek to reduce the complexity of managing a large scale deployment or infrastructure. Usually, these frameworks are complex – that complexity is introduced in a large part by their attempted generality. We argue that we can produce a simpler tool by taking a restricted view of how many information technology organizations actually conduct their operations.

We argue that the concept of *layered virtual appliances* should be central to the development and deployment of a virtual machine management framework – so much so that we are focused on a *virtual appliance management framework* rather than a *virtual machine management framework*. By adopting this focus, and using a simple, extensible framework for managing such appliances, we show how virtualization can be brought to organizations without sufficient knowledge or staff to install and support a complex virtual infrastructure. We also explore how virtualization can be used to provide cheap datacenter services to larger companies, or research facilities that are unable or unwilling to run a commercial management framework. In each case, we're focused on a simple management framework that is easy to adopt.

Server sprawl and operating system (OS) management are major concerns in the area of information technology [2]. This paper addresses these concerns by simplifying the use of virtualization and system configuration for application developers and system administrators. We also show how our simplified interface can still be used to provide scalable "on demand" computing services using standard interfaces and technologies.

As we describe in more detail later, virtual appliances [3], are a combination of operating system and application components that provide a specific computing task, such as spam filtering, mail delivery or web serving. The STORM system provides a *virtual appliance configuration and provisioning system*. The STORM management node controls a cluster of computers that use a virtualization hyperviser, such as Xen, VMWare or VirtualBox. Each node in the cluster must run a specific control program (not shown) that coordinates the STORM management node. Administrators (or programs) can cause virtual appliances to be deployed on nodes within the cluster of computers. The STORM system determines on which nodes the appliance should be run, loads the appropriate configurations and customizes them for the environment. Given system management mechanisms such as IPMI,

the STORM system can also dynamically manage the power state of different computing nodes to reduce the energy needs based on load and configuration. Using monitoring interfaces provided by common hypervisors, the STORM system can cause new appliance instances to be generated when CPU utilization or reported host demands warrant.

As we'll describe in related work section, there are many existing virtual machine management systems. Some of these are designed for specific applications, such as managing "grid" computing or clusters of machines. The approach we took in the STORM system is to focus on simplicity and ease of infrastructure maintenance. The framework is simple because it uses readily available technologies (reducing the time for installation) and presents a simple but very capable web interface for system management. To simplify ongoing management and deployment of applications, each deployed application contains four "layers":

1. A *common operating system substrate* that contains the basic components needed by all virtual appliances; the Ubuntu "Just Enough OS" (JEOS) platform is a representative example of this substrate.
2. An *appliance specific component* that provides the application and necessary libraries; an example might be the the "postfix" program, ldap and mysql libraries for remote mail deliver and other necessary libraries.
3. A *deployment specific component* that customizes the combination of the operating system substrate and the appliance specific component; an example might be the configuration files for postfix, mysql, nfs and ldap. The deployment specific component essentially captures *changes* to the underlying appliance component – for example, the appliance component would typically include "off the shelf" configurations provided by an O/S distribution. The deployment specific component would be the result of an appliance maintainer editing the specific configuration files to customize those files for the local environment.
4. An *instance specific component* that uses information provided by the STORM server to configure a specific instance of a more general appliance. For exmaple, that instance specific information may configure the domain name to be "mail.foo.com" *vs.* "mail.bar.com" and makes (minor) changes to /etc/sendmail configuration files based on data from a specific configuration file accessed by the STORM server.

On Linux, the STORM system can deploy an "initial RAM disk" that combines these four layers using the "union file system." This configuration allows a single O/S configuration to be used by multiple clients, and that single configuration can be provided by NFS or iSCSI. Using the union file system in such a structured fashion greatly simplifies the task of building the "deployment specific component" – the local administrator essentially logs in to an instance of the machine and updates the configuration files. The top "writable" layer of the union filesystem will capture those changes. Many of the changes will involve adding instance specific components, which are specified as "variables" in the configuration files that are later expanded when the instance is created.

These layers allow the base O/S and the specific applications to be split; this means that an underlying O/S image, including all the common libraries and management tools, can be patched without having to then patch each individual appliance component. Reducing the number of operating system configurations greatly reduces the security risks of multiple unpatched configurations; it also reduces the workload on the administrator. Likewise, the separation of the appliance specific component, the deployment specific component and the instance specific allows the common components to be upgraded without having to reconfigure each component; again, this assists in securing those appliance components. It also reduces the storage needed for the underlying O/S substrate – while this seems like a minor feature, having many appliances use the same O/S image means that the file server can more effectively cache common utilities, speading access and reducing needed resources.

This layering is not perfect – depending on the package management system used by the underlying O/S, it may be that an appliance component may install patches already provided by an updated or modified O/S substrate. In our experience, systems similar the "debian" package management system provide the most flexible interface – these systems stores package information in individual files, rather than in databases as is done in the common "redhat" package management system. Either packaging system works, but the database versions will consume more space.

These individual components can be automatically combined when configuring the provisioned system. STORM can also be used to exert finer levels of control should the "virtual appliance" model not be sufficient; however, these are not the focus on STORM nor of this paper. Likewise, the STORM system also provides a secure XML-RPC service that can be used by appliances themselves to control provisioning based on environmental or load conditions.

Again, by offering guidence in how a system should be configured an by limiting the scope of problems that we try to address, we believe that STORM provides a simplified workflow for an IT administrator. In this remainder of this paper, we briefly describe the virtualization technology that underpins STORM. Then, in the Method section we describe the components that make up STORM. In the Example and

Analysis section, we walk through an example of using STORM to configure an energy and load-aware mail processing system. We subject that system to artificial load and demonstrate that STORM is capable of adaptively adjusting the number of mail processing appliances. Lastly, in section Future Work, we describe similar systems and future directions for STORM.

Background

Virtualization is not a very new area of computer science. The concept of virtual machines date back to the 60's with the IBM research system titled CP-40 [4]. This system is one of the first known to be able to run multiple instances of client operating systems within it. Virtualization gained popularity as a management tool with the development and widespread deployment of VMware circa 1998. The VMware virtualization technology took a different approach than the IBM hardware – the virtualization was done by binary rewriting.

In 2003, Xen, developed at the University of Cambridge [1], introduced a *para-virtualization* system, where in the ''guest'' operating system cooperated with the virtualization system to reduce virtualization overhead. Their described an implementation of a hypervisor that made para-virtualization possible on the x86 architecture. The code for this implementation was released under an open source license, and distributed freely on the Internet, greatly increasing the popularity of virtualization on commodity platforms. Since that time, Intel and AMD have provided additional hardware support to improve virtualization performance.

With the availability of an inexpensive and high performance virtualization system, many projects have been started using this technology to provide homogeneous computing, in which the operating system is independent from the hardware.

The *hypervisor* is the software that enables multiple operating systems to run on a single physical host. It is the intermediary between the operating system being virtualized and the physical host. The hypervisor is also responsible for handling time sharing between virtual machines. There are several hypervisors currently available, a few worthwhile mentioning are: Vmware, Xen, KVM, and Virtual Box. In this paper, our description of the STORM system is based on the Xen hypervisor, but STORM is not limited to that virtualization hypervisor. STORM uses the libvirt virtualization library to interface to the hypervisor, meaning it can support Xen [1], Qemu [5], KVM and ''container systems'' [6] such as the Linux Container System and OpenVZ. The libvirt interface can be easily extended to other virtualization systems.

A *virtual machine* is the guest operating system being controlled by the hypervisor. It contains the application(s) that a specific user desires to run. Depending on the type of virtualization used, the operating system can be run on the hypervisor without any modification.

When using Xen, a virtual machine is typically referred to as Domain*U*, where *U* is a unique number to the specific virtual machine. The number 0 is reserved for a special domain that has escalated privileges for management purposes.

A *virtual appliance* [7, 8] is the definition of a virtual machine designed to performance a specific application. The definition typically includes metadata describing information about services provided, resources required, and dependencies. The metadata is typically stored in a portable format, such as XML.

There are two types of ways to describe a virtual machine. One method [7, 8, 9, 10] describes it entirely in metadata. All information regarding packages to install, ports to open, services to configure, is defined in metadata. The software creating virtual machines from the definition will take this description and do everything necessary to make sure the virtual machine created is exactly as the author intended. Configuring the system solely from metadata is very extensible; however it requires more work from appliance developers. There are tools available that can reduce this workload.

Another method takes the combination of metadata, and a disk image containing a pre-configured version of the operating system and all software desired. This method is not as extensible as the first since it requires the distribution of a hard disk image. However, it allows the developer to have more freedom in configuring the virtual appliance and requires adoption of fewer tools. This is the approach we have take in STORM, but we've extended the basic ''disk image'' approach by using *layered* appliance deployments. We describe this method in the next section.

Method

The overall STORM system consists of three primary entities:

1. the STORM manager (see the Virtual Appliance Server section)
2. channel server (see the Channel Server section),
3. a disk image server (that may be integrated with the channel server),
4. and the virtual appliance server (see the Appliance Definition section).

The interactions between these entities are shown Figure 1. The management appliance plays the most important role in the system. It is responsible for creating & controlling virtual machines, appliance servers, and for fetching appliances. In practice, the STORM manager is a virtual appliance. Having the management software implemented inside a virtual appliance provides increased security, simplified installation,

the ability to run on any available virtual appliance server, and adheres to the guidelines given by the Xen creators [1].

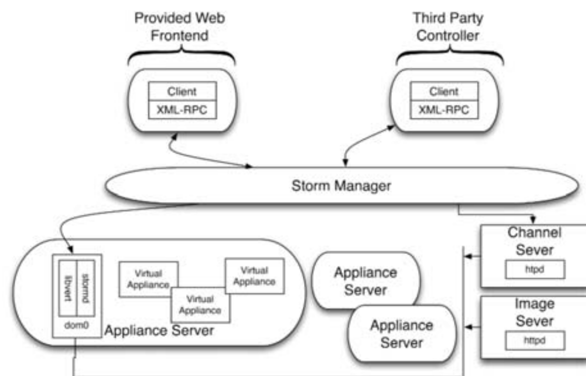

Figure 1: STORM system components and interactions.

STORM provides an easy-to-use web interface which is programmed in Python with the help of a framework called TurboGears [11]. This web interface gives administrators the ability to control the current state (running, stopped, paused) of any virtual machine, install new appliances, and manage available appliance channels.

Each appliance in the STORM cloud receives a DHCP address from the STORM appliance, or if configured, from an external DHCP server. This address can be dynamic or statically configured. Running a DHCP server on the management appliance prevents other appliance developers from having to worry about network configuration.

The STORM appliance also provides Kerberos and LDAP services. This allows for the customer to have a centralized user and password database against which virtual machines can authenticate. Again, our emphasis on simplicity of system configuration and common IT tasks led us to provide such a centeralized authentication and authorization service. That service also gives granular control over which virtual machines users have access to. For example; Bob can be detailed to have access to upload files to the web server but not make alterations to the MySQL database. The services provided are similar to Microsoft Active Directory, but use open source software and database schemas. If desired an external or appliance based active directory server can be used instead.

Lastly, the STORM appliance can control the power state of the client machines using the Intelligent Platform Management Interface (IPMI) [12]. Client machine hardware requirements depend on the number of virtual machines desired to be ran concurrently. The amount of RAM should always be 128 MB greater then the amount required to run the desired virtual machines. This ensures that domain 0 has enough memory available to operate the STORM control daemon. There should also be sufficient cores available to meet the requirements of each virtual machine.

Virtual Appliance Server

In order to accomplish these tasks the management software communicates with two daemons running within Domain0 on any given virtual appliance server. One of the daemons is libvirtd, which provides remote access to the Xen API [13]. The other (stormd) is specifically designed for this project and provides access to services and information that cannot currently be obtained from the virtual appliance server through libvirtd. These services include: appliance retrieval, upgrades, virtual machine creation, removal, virtual appliance IP address reassignment, shutdown and restart. The daemon is a XML-RPC server implemented in python and it extends the built-in XML-RPC server class. It listens for and handles connections from authorized STORM virtual machines. It is the sole piece of software responsible for handling the services described above.

Both of these daemons authenticate and encrypt communications from the STORM virtual machine using SSL. Each virtual appliance server has a local certificate authority that is responsible for generating, signing and providing a public/private key pair to the management appliance. The appliance then uses that key pair when connecting to either daemon when requesting to perform tasks.

Channel Server

A channel server is responsible for distributing and providing virtual appliances to STORM users. The user points the STORM appliance server to the desired channel, and then all information regarding appliances available is cached. After this step is completed, the user may create virtual machines based off the appliances on the channel server.

The information about available appliances is stored in an RSS feed, which are currently unlocked due to the scope of the project. The RSS feed provides URL's to the location of each appliance, allowing for load to be distributed across multiple servers. The feed and appliances are accessed via the HTTP protocol. Any standard HTTP server is suitable to act as a channel server, however Lighttpd is recommended. Lighttpd offers several performance enhancements over others and also requires very few resources to run.

Appliance Definition

The metadata that goes with a given STORM appliance is defined entirely in XML. This allows other systems to easily recognize and parse an appliance. It also makes it much simpler for a developer to create an appliance as they do not have to obtain knowledge of a proprietary format.

A key difference between the STORM appliance definition and other appliance definitions is simplicity. Formats such as the Open Virtual Machine [14], or CVL [8] are either too complex or non-trivial to parse. The STORM appliance definition contains 'bare

minimum' metadata to describe a virtual appliance. The remaining information required is stored within the appliance image itself. Some sample fields of the data supplied are: label, description, resource requirements, dependencies, and control panel URL. A sample configuration is shown in Figure 2 A full description of the specification can be found in Appendix A.

API Implementation

Upon connection to either the libvirt or storm daemon, the management appliances verifies the authenticity of the daemon it is connecting to, and the daemon verifies the authenticity of the management appliance. Once the connection has been established, the management appliance may issue nurmerous procedure calls. The following highlights procedure calls that are important to the opperation of the system as a whole:

- get_downloadprogress(): Returns the progress of a current operating system, or appliance in a percentage number less than 1. This function will return -1 if nothing is currently being downloaded.
- get_downloadspeed(): Returns the speed (in kilobytes per second) at which a current operating system or appliance is downloading at. Returns -1 if nothing is currently being downloaded.
- get_diskspace_used(): Gives amount of physical disk space allocated in megabytes for virtual machine usage. Actual usage information is available only from the virtual machine itself, and is currently the responsibility of the appliance developer to give per virtual machine disk usage. Future work includes developing a method for obtaining per virtual machine disk usage.
- get_virt_uptime(virtual_machine_id): Provides uptime information for a given virtual machine.
- server_info(): Returns the version of the server, list of capabilities, and other general information such as uptime.
- server_upgrade(): Updates all programs running on the virtual appliance server.
- server_set_network(self,...): Sets the network configuration for the virtual appliance server identified by the specific network configuration (IP address, netmask, gateway, primary and secondary DNS server). If the DHCP flag is set then all other network parameters are ignored and information is taken from DHCP.
- disk_create(): Create a blank or empty disk for the instance about to be deployed; this image is used to store automatically modified configuration files and data files for the instance.
- app_install: Downloads an appliance image (e.g., a sendmail appliance) from the channel server; instances can be created from this image.
- app_check: Checks to see if an appliance image exists (if it does not, the image is retrieved using app_install).
- app_upgrade: Forces an upgrade for an appliance, downloading a fresh image from the channel server and replacing the current appliance image with that updated image.
- os_install: This is similar to the app_install method, but it specific to the operating system layer rather than the application layer.
- Additional interfaces: There are a number of interfaces that provide services complimentary to the ones described above. We omit the detailed description for these interfaces: server_ reboot(), server_shutdown(), disk_remove, app_ remove, os_check, os_upgrade, and os_ remove.

Scalability

With the addition of network attached storage, or a storage area network the STORM system will scale to support a very large number physical hosts and virtual machines. It can theoretically address up to 2^{32} (size

```
<?xml version='1.0' encoding='utf-8'?>
<appliance version='1.0'>
    <label>Simple Web Server</label>
    <description>A very simple and efficient test web server.</description>
    <version>1.0.0-0</version>
    <size>1024</size>
    <url>http://cs.colorado.edu/appliances/webserv.tgz</url>
    <sig>http://cs.colorado.edu/appliances/webserv.asc</sig>
    <provides>webserv</provides>
    <webpanel>/upload.php</webpanel>
    <hardware>
        <cpus>1</cpus>
        <memory>524288</memory>
        <ostype>linux</ostype>
        <disk name="hda1" type="system" file="sys.img" />
        <disk name="hda3" type="swap" size="128" />
        <disk name="hda2" type="user" size="128" />
    </hardware>
    <dependency>emailserv:1.0.0-0</dependency>
    <dependency>firewall</dependency>
</appliance>
```

Figure 2: Sample appliance configuration.

of an unsigned integer) physical hosts on a 32-bit architecture.

The channel server can also be scaled to meet demand from multiple clients. This can be done by having a single server maintain the RSS feed, and an index of appliances available. The actual appliances themselves can be stored across multiple image servers with different URL's or on a single URL with an HTTP load balancer to dynamically redirect traffic to servers which are least busy.

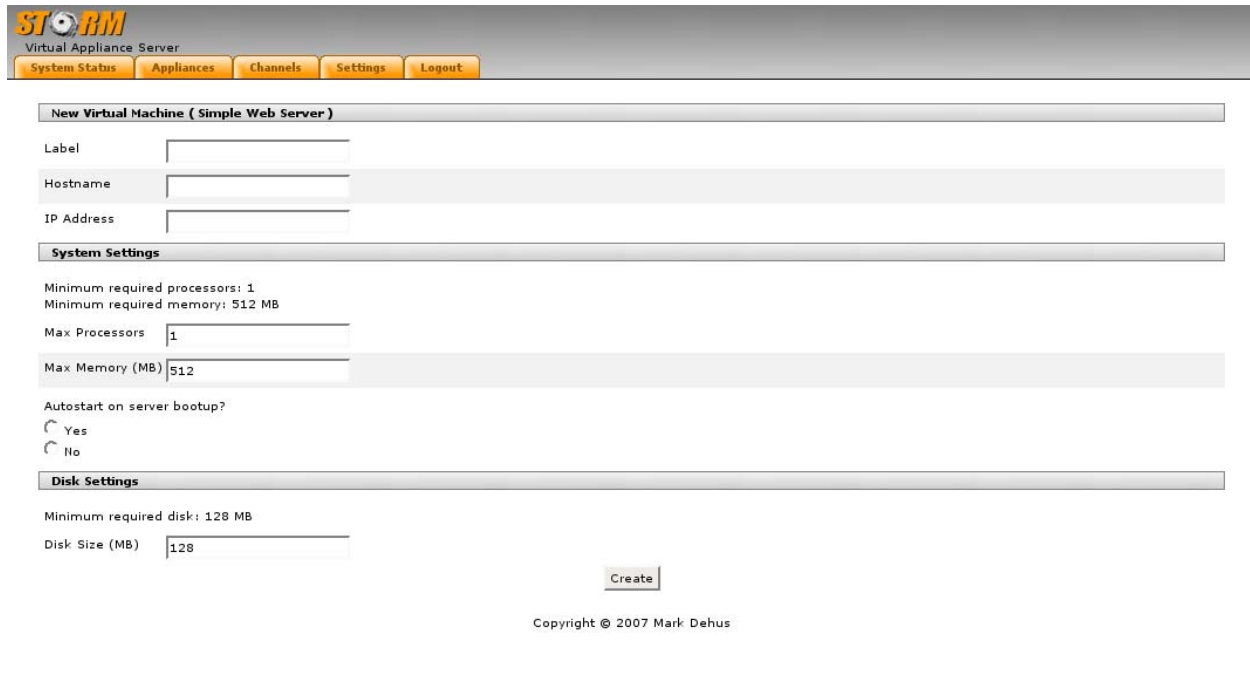

Figure 3: Web panel used to create virtual appliance.

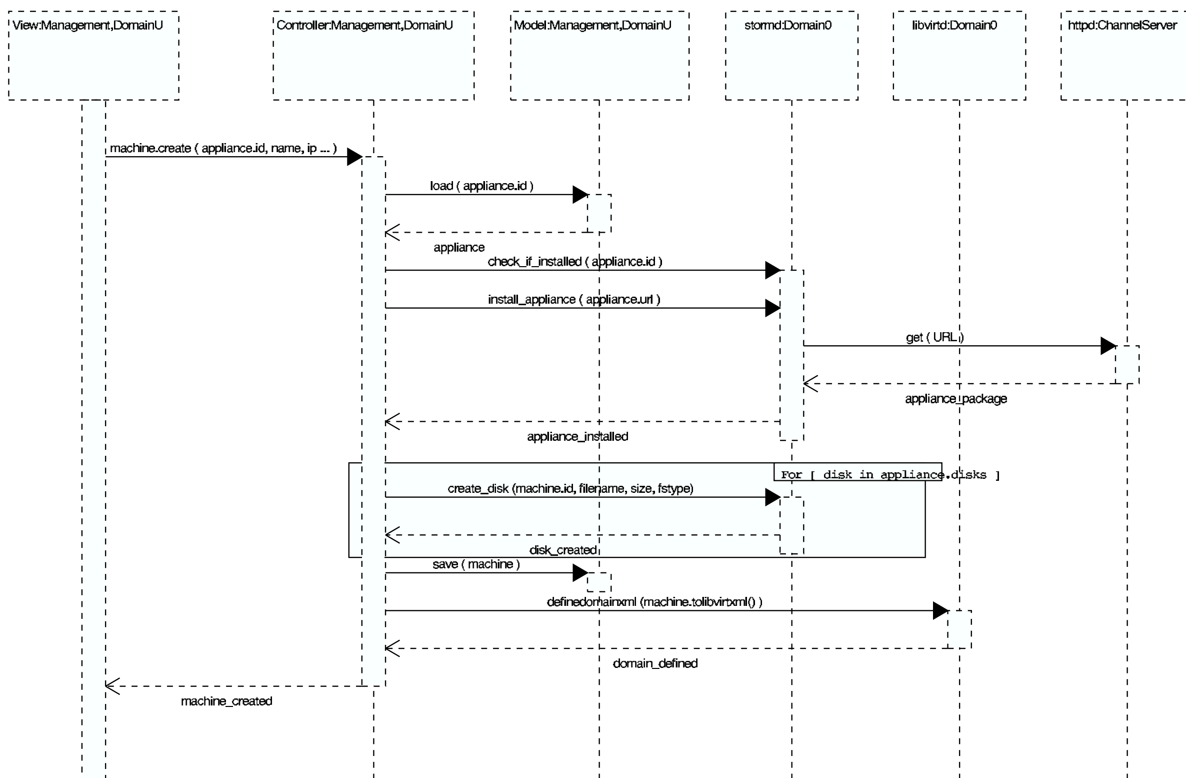

Figure 4: UML description of system component interactions.

Both the physical hosts and channel servers can easily scale without any problems and the only potential bottleneck is within the management software. Any performance issues in the software can simply be resolved by fixing the problem area within our code. We did not have sufficient hardware to test a large scale installation, but did not encounter any problems on the setup with our 16 node cluster.

Usability

The web interface for the STORM system was designed with Palmer's five constructs [15] for website usability in mind. The interface was also designed to meet the following standards: XHTML4, CSS1, and 508. Each of these standards helps to ensure that the interface will be portable, readable, and easily accessible in all browsers.

Security

In order for the STORM system to be secure two important areas must be resistant to attacks: communications and software. Secure communications are required to ensure that an attacker cannot listen in or hijack any connections between a user and the web interface, or between the web interface and management daemons running on the hypervisor. Secure software is required to prevent an attacker from exploiting bugs in code to gain unauthorized access, or to make the software behave in an undesired manner.

Secured communications are achieved through the use of TLSv1, which at the time of writing is a known to be secure protocol [16, 17]. Each session is authenticated using SHA1 signed certificates, and encrypted under AES128. This includes: the session between the user and the web interface, and between the web interface and hypervisor. For the purpose of testing self-signed certificates were used, however in a production environment all certificates would be signed by a commonly known certificate authority such as VeriSign.

The primary area of concern with software security is in the code that provides network functionality. Standard python libraries were used to provide networking functionality within the storm daemon. These libraries are xmlrpclib [18] and m2crypto [19]. They are both open source, community maintained projects. No custom code was written to provide network functionality; therefore the storm daemon is secure as long as the libraries used are also secure. At the time of writing there are no known exploits for the versions used in either library. Furthermore, once libvirt receives more development in the area of virtual appliance management it could eliminate the requirement for the storm daemon altogether.

Component Interaction

The interaction of the components can best be explained by describing the interactions during specific operations, such as virtual machine creation and configuration, as shown in Figures 3 and 4.

In that example, the Web interface (Figure 3) is used to create a specific machine instance. The UML descripton, shown in Figure 4, indicates that request is relayed to the STORM controller written in Python, running on the management appliance; the controller then communicates with the STORM daemon using XML-RPC to determine if the appliance is already available on that specific system; if not, the appliance is installed by retrieving the required disk image from the channel server. The STORM daemon then contacts the STORM management controller to inform it that the appliance has been installed; the controller then directs the storm daemon to create disks and other resources as directed by the appliance specification. Once those resources are created, the controller defines a new Xen domain, configures the network resources indicates to the the user that the machine has been created. Note that the entire process can be controlled by XML-RPC and does not need to be driven by the web interface.

Example and Analysis

To illustrate the capabilities of the STORM system, we use the XML-RPC interface and the Xen kernel monitoring utilities to implement a scalable email processing system. In this scenario, we configured an email appliance using the ''postfix'' MTU. To demonstrate the power control and automatic provisioning made possible by STORM, we used two slots or blades in a Dell M1000 cluster with M605 blades (dual socket, four cores, 3.0 GHz, 16 GB RAM) as the mail processing engines. We used an additional four blades as load generators to subject the mail processing engines to extreme load. The Dell M605 blades provide an IPMI interface that allows us to measure power usage (in Watts) as well as control the power state of individual blades. We used information from the Xen virtual machines to estimate server loads; an alternate mechanism would be to monitor SNMP data from individual operating systems, but we focused on O/S-independent mechanisms and mechanisms that would be available even if the guest O/S is subject to intense service loads. We determined that a guest O/S is overwhelmed when the assigned CPU utilization is at 90%, and that a physical host is overwhelmed when the overall CPU utilization of the host is at 90%. The CPU utilization is based off the number of CPU seconds used as provided by Xen.

In the original configuration, we deployed a single mail processing utility; each mail agent simply discarded email's that were successfully delivered. Following the start of the mail processing system, we enabled the load generating programs (which were also configured as STORM appliances on alternate blades); those programs produced 25 MB mail messages at a rate of 2000 per second.

CPU Frequency scaling was not deployed on the test system, thus the increases in power usage as appliances are being brought online is not very visible until

a new node is turned on. Power fluctuation is mostly attributed to the boot process of the second node. During POST all fans are spun at full speed, disks are spun up, and initialization procedures are run for the underlying hypervisor and Domain.

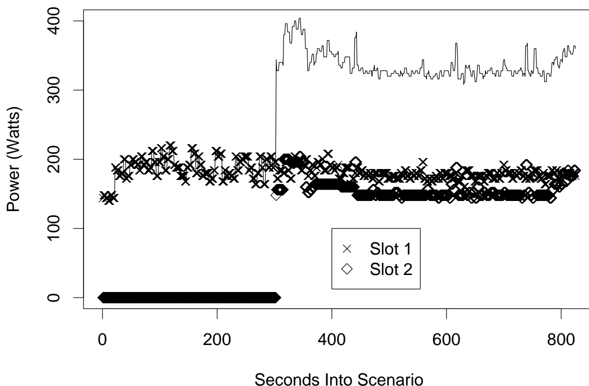

Figure 5: Power Usage Under Increasing Load. This diagram shows measurements of instantaneous power usage collected using an IPMI interface as two blade slots are used for mail processing. The individual data points indicate the power for the individual blades and the line indicates the power for the combined set of blades.

Figure 5 shows a time series plot of the power consumed by the individual mail processing systems. At approximately 300 seconds into the experiment (shortly after the load generating programs were enabled), the reported load for "Slot 1" (the primary mail processing program) was sufficiently large that the STORM monitoring component elected to configure a second mail processing node. The system was configured and deployed using the mechanisms described earlier. The mail processing systems have multiple MX and A associated records. Every time Storm brings up a new instance of the mail appliance, the necessary records are automatically added to the domain if STORM is designated as the primary DHCP & DNS server. This effectively balances the load across all running mail servers. As you can see in Figure 5, while the demand is low, total system power is low because one of the processing nodes is shut off. As the demand rises, more instances of the mail appliance are created on the first eight-core processing node, causing an increase in power. Eventually the first appliance server starts to reach maximum capacity, causing the second eight-core blade slot to be turned on. When the second appliance server becomes available then new instances are created on whichever blade slot has the lowest load. Figure 5 shows the second processor ("Slot 2") being enabled at about 300 seconds. There is a short burst of maximum power as the system undergoes self-test and then individual cores are allocated for for processing tasks.

This entire process is not automatic – in particular, the configuration of our round-robin DNS server is an afterthought and somewhat grafted to the other infrastructure. However, using the XML-RPC interface, constructing even this extension to the existing system was a days work. More complex was actually determining what interfaces could export "system load" in a reliable fashion when a system was actually being severely loaded. This example demonstrates both the capabilities of the underlying STORM system and the benefits accrued from those capabilities. We've found in practice that we can finely control the power demands of applications without extensive system augmentation – this provides a valuable infrastructure to system administrators seeking to reduce operating costs without impacting operations reliability.

Related Work

Virtual machine management has been touched upon by several groups. As we briefly mentioned in our introduction, the focus of STORM is simplicity and ease of infrastucture maintenance. In this section we will compare and contrast STORM to other management systems. VMware currently offers several management products for its hypervisor (ESX Server), the two most relevant to our work on STORM are VirtualCenter (VC), and Distributed Resource Scheduler (DRS) [20].

VirtualCenter is a centralized management tool that allows an administrator to provision, deploy, and manage virtual machines across a cluster of ESX servers. These virtual machines can be custom built or downloaded in an appliance-like fashion. STORM offers the same functionality as VC, however there are key differences in approach. VC assumes that the administrator has a general knowledge of virtualization, while STORM is more designed towards simplicity and assumes no knowledge of virtualization at all. VC and STORM also greatly differ on their approach to Virtual Appliances. Vmware offers appliances that may be manually downloaded from their web site [21], and ran within VC. Unlike STORM, they provide no real distinction between a virtual appliance or machine because of their non-layered approach. An appliance in VC terms contains both the application and operating system, which leads to redundant data.

DRS provides the ability to dynamically allocate virtual machines in a cluster of ESX servers. It will load balance virtual machines based upon utilization. For example, if a virtual machine is allocated a large amount of resources on an ESX server but currently is not using them then DRS will allow other machines to execute on that ESX server. When the load increases, it will adjust accordingly. Using DRS in the scenario we described in our analysis would result in an unresponsive mail server as it cannot increase the amount of resources available to that virtual machine. STORM operates in a similar manner, but offers the ability to address application specific load and scale accordingly by creating more virtual machines with the applications to handle the excess load. In an ideal world, both

STORM and DRS would increase the amount of processors and/or RAM allocated to the virtual machine. While some operating systems [22] in combination with certain hypervisors may offer the ability to dynamically adjust resources, we opted not to restrict STORM to any hypervisor/operating system pair.

Another company taking advantage of this concept is called Enomalism. They have written a web based Xen management system that also has a definition of virtual appliances [23].

There are a number of academic projects focused on managing virtual cluster system. The Collective [8, 7] is a system designed using a metadata-rich specification system; this work is notable for introducing the notion of ''virtual appliances'' and designing a system to manage such appliances. The goal was to manage collections of virtual appliances using the rich CVL (Collective Virtual appliance Langiage). A portion of this project appears to have led to the Moka5 virtual appliance company, which takes a similar approach but focuses on desktop virtualization.

Managing Large Networks (MLN) [10] took a similar approach as the Collective, and used a scripting language (Perl) and extensible metalanguage to configure collections of nodes. MLN was focused on managing *networks* of nodes, and offered a rich configuration infrastructure for that. MLN has been used for projects making use of virtualization for academic infrastructure [24], an application domain we have also targeted. Usher [9] extended this approach to further simplify the management of clusters of related virtual machines.

Most of these system used a common infrastructure (e.g., libvirt) or a similar design. Each used a configuration language – this becomes increasingly important when deploying a *network* of nodes, but is more complex to deploy and manage in smaller appliance-oriented installations.

Although some of the commercial management tools provide integrated power management and scaling options, few of the academic systems have focused on these capabilities. Sandpiper [25] studied the value of different approaches to migrating virtual machines; similar mechanisms would be useful in controlling power, because one goal that we have not implemented would be to ''pack'' virtual machines into as few physical systems as possile. Sandpiper used service level agreement (SLA) specifications to guide their migration strategy; a similar policy specification would be appropriate for power control.

There are fewer projects that have examined *desktop virtualization*; as mentioned, Moka5 is one commercial offering. The Internet suspend/resume Project [26] is one example of a project that has been using virtualization technology to simplify system administration tasks and the way people think about portable personal computing. For example: instead of carrying around a laptop with operating system and applications, the ISR project stores that environment as a virtual machine on the Internet; users carry data with on small device such as a USB drive.

Future Work

There are numerous applications of the STORM system in small to medium business, however it is important to note that there are also several other applications as well. One of the most notable is cluster and datacenter management. In the case of cluster and datacenter management, CPU time could be sold to a customer. The customer would package an appliance designed to execute their application. The appliance would then be uploaded to a channel server, and then it would be install by the datacenter administrators. The STORM system is capable of providing this functionality; however it currently lacks an accounting infrastructure. Creating this functionality is trivial and could be completed in a minimal amount of time. Existing commercial systems such as Amazon's *EC 2* employ similar mechanisms, but the simplicity (and availability) of the Storm infrastructure should allow smaller and regional service providers to offer similar capabilities as similar costs through the automation offered by Storm.

Several additions can be made to the STORM system. Some have already been discussed in this paper, such as an accounting system capable of keeping track of virtual machine CPU usage. This capability could be used to sell time on a datacenter to customers who would find it more cost effective then purchasing and running one of their own. Completedly automated load balancing and service distribution can be added to the STORM system. We demonstrated a version of this capability, but improvements are possible – in particular, accurately estimating load independent of the specific virtual appliance is a difficult task.

In order to accomplish the difficult task of determining need for more resources, or need for more virtual machines assigned to a given task, things such as process load, queue lengths, available I/O operations, and amount of free space in various kernel buffers could be taken account.

For example, a customer running a virtual spam filter suddenly receives a massive amount of incoming spam. This increase causes the spam filter to become overwhelmed. The filter would report this information to the STORM system, which would then respond by either increasing the amount of resources available to the filter or spawning more filters.

Service discovery is another feature that should eventually be added to the STORM system. This would also require client software installed on each virtual machine. It would allow a virtual machine to easily find and contact services provided by other virtual machines. These could be standard services such as

DNS, or services developed specifically for an application.

Since the current system only supports Xen, it would be beneficial to add in support for additional hypervisors; this was the intent of the libvirt project. A virtualization environment can consist of many different hypervisors in order to meet a specific customer's needs. It would also be beneficial to support additional appliance formats. This will allow the customer greater accessibility to a wider range of services, especially ones geared towards other hypervisors.

Conclusions

The resurgence of virtualization has greatly impacted the information technology infrastructure. Companies that lack sufficient knowledge to capitalize on the advantages provided by virtualization are unable to move towards it. The Storm system successfully allows these companies to capitalize on virtualization and reduce or eliminate the need for in house technical support. In general the Storm system allows application developers to provide a single pre-configured virtual appliance to which a customer may deploy virtual machines from. This eliminates the need for each customer to maintain their own operating systems to run the desired application. It does this in a secure and efficient manner by avoiding the common pitfalls that similar solutions suffer from.

We plan to make the STORM system available on SourceForge [27] before the end of 2008.

Author Biographies

Dirk Grunwald recived his Ph.D. from the University of Illinois in 1989 and has been a faculty member at the University of Colorado since that time. He is interested in the design of digital computer systems, including aspects of computer architecture, runtime systems, operating systems, networking and storage. His current research addresses resource and power control in microprocessor systems, power-efficient wireless networking and managing very large storage systems.

Mark Dehus recently graduated with his M.S. in Computer Science from the University of Colorado at Boulder. His focus is virtualization and large-scale systems administration, but is also interested in systems engineering, operating systems, and networking. Some of his current research includes, optimized course capture for web distribution, and applications of virtualization toward computer science education.

Bibliography

[1] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the Art of Virtualization," *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 164-177, New York, NY, USA, 2003.

[2] Khanna, G., K. Beaty, G. Kar, and A. Kochut, "Application Performance Management in Virtualized Server Environments," *10th IEEE/IFIP Network Operations and Management Symposium, 2006 (NOMS 2006)*, pp. 373-381, 2006.

[3] Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. Lam, and M. Rosenblum, *Virtual Appliances for Deploying and Maintaining Software*, 2003.

[4] Varian, Melinda, "VM and the VM community: Past, Present, and Future," *SHARE 89 Sessions*, August, 1997.

[5] Bellard, Fabrice, "QEMU, A Fast and Portable Dynamic Translator," *Proceedings of the USENIX Annual Technical Conference 2005*, p. 41, 2005.

[6] Soltesz, Stephen Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson, "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," *SIGOPS Operating Systems Revew*, Vol. 41, Num. 3, pp. 275-287, 2007.

[7] Sapuntzakis, Constantine and Monica S. Lam, "Virtual Appliances in the Collective: A Road to Hassle-Free Computing," *HOTOS'03: Proceedings of the Ninth conference on Hot Topics in Operating Systems*, p. 10, 2003.

[8] Sapuntzakis, Constantine, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum, "Virtual Appliances for Deploying and Maintaining Software," *LISA '03: Proceedings of the 17th USENIX Conference on System Administration*, pp. 181-194, 2003.

[9] McNett, Marvin, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker, "Usher: An Extensible Framework for Managing Clusters of Virtual Machines," *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November, 2007.

[10] Begnum, Kyrre M., "Managing Large Networks of Virtual Machines," *LISA '06*, pp. 205-214, 2006.

[11] *Turbogears Framework*. http://turbogears.com .

[12] Intel, *Intelligent Platform Management Interface Specifications*, Feb., 2006, http://www.intel.com/design/servers/ipmi/ .

[13] *The Virtualization API*, http://libvirt.org .

[14] VMware, *Open Virtual Machine Format*, http://vmware.com/appliances/learn/ovf.html .

[15] Palmer, Jonathan W., "Web Site Usability, Design, and Performance Metrics," *Information Systems Research*, Vol. 13, Num. 2, pp. 151-167, 2002.

[16] Wagner, David and Bruce Schneier, "Analysis of the SSL 3.0 Protocol," *WOEC'96: Proceedings of the Second USENIX Workshop on Electronic Commerce*, p. 4, 1996.

[17] Paulson, Lawrence C., "Inductive Analysis of the Internet Protocol TLS," *ACM Transactions on Information System Security*, Vol. 2, Num. 3, pp. 332-351, 1999.

[18] *Python 2.5 XML-RPC*, http://docs.python.org/lib/module-xmlrpclib.html .

[19] *Me Too Crypto 0.19 (m2crypto)*, http://chandler-project.org/bin/view/Projects/MeTooCrypto .

[20] *VMware Infrastructure: Resource Management With DRS*, http://www.vmware.com/pdf/vmware_drs_wp.pdf .

[21] VMware, *Vmware Virtual Marketplace*, http://vmwware.com/appliances .

[22] Pinter, S. S., Y. Aridor, S. Shultz, and S. Guenender, "Improving Machine Virtualization with 'Hotplug Memory'," *17th International Symposium on Computer Architecture and High Performance Computing*, pp. 168-175, Oct., 2005.

[23] Enomolism, Inc., *Enomolism Elastic Computing Platform*, http://enomolism.com .

[24] Gaspar, Alessio, Sarah Langevin, and William D. Armitage, "Virtualization Technologies in the Undergraduate IT Curriculum, *IT Professional*, Vol. 9, Num. 4, pp. 10-17, 2007.

[25] Wood, Timothy, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif, *Black-Box and Gray-Box Strategies for Virtual Machine Migration*.

[26] Satyanarayanan, Mahadev, Benjamin Gilbert, Matt Toups, Niraj Tolia, Ajay Surie, David R. O'Hallaron, Adam Wolbach, Jan Harkes, Adrian Perrig, David J. Farber, Michael A. Kozuch, Casey J. Helfrich, Partho Nath, and H. Andres Lagar-Cavilla, "Pervasive Personal Computing in an Internet Suspend/Resume System," *IEEE Internet Computing*, Vol. 11, Num. 2, pp. 16-25, 2007.

[27] *Sourceforge: Open Source Software Development Web Site*, http://sourceforge.net .