MARK DEHUS AND DIRK GRUNWALD

STORM: simple tools for resource management

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, and his current research includes virtualization management systems, content delivery/distribution networks, and applications of virtualization toward computer science education.

mark.dehus@colorado.edu

Dirk Grunwald received 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.

dirk.grunwald@colorado.edu

CLOUD COMPUTING HAS BECOME ALL the rage, because it allows an organization to forget about the added management associated with having a single operating system tied to a single physical piece of hardware. In this article we discuss a cloud system built using standard tools to further reduce the amount of overhead in computer administration by simplifying software configuration. We also introduce the notion of layered virtual appliances, an approach we consider to be critical for the success of any large-scale cloud management software.

STORM [1] is a system built with the overworked system administrator in mind. It is designed to simplify the configuration and management involved with running applications as much as possible. Using the provided Web interface, an administrator can quickly provision desired software without having to worry about the middle layers such as Apache installation and configuration.

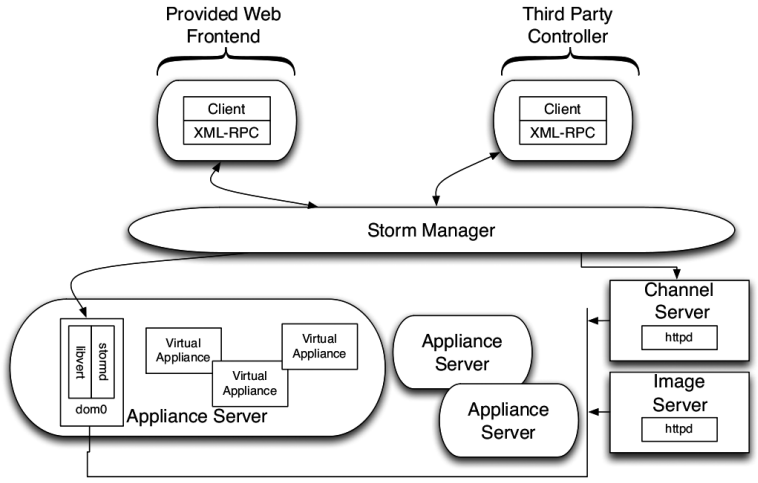

FIGURE 1: THE LOGICAL ARRANGEMENT OF THE FOUR PRIMARY ENTITIES IN STORM

The overall STORM system, illustrated in Figure 1, consists of four primary entities:

- The STORM Manager: The control subsystem for the cloud, which is itself an instance of a virtual appliance. This allows for the management framework to receive the benefits that come along with being an appliance.
- The Virtual Appliance server: The physical hardware running any hypervisor supported by our underlying API, libvirt [2].
- A Channel Server: The server responsible for

serving RSS to the STORM manager. The RSS feed contains information about available appliances.

- An Image Server: The server that strictly serves up disk images. It may be integrated with the channel server if scalability is not required.

The STORM manager, being the central control system for the cloud, is responsible for handling incoming requests from either the administrator via the provided Web front end or from an authorized appliance instance (i.e., a virtual machine). These requests can range from creating new appliance instances to registering MX records.

Several types of requests require the STORM manager to communicate with any given appliance server; this communication is accepted by libvirtd or a custom daemon called stormd, depending on the request. To authenticate the STORM manager, a signed certificate is presented by the manager upon connection. The daemons then verify this signature against a locally stored CA certificate. Libvirt does this authentication internally, and for consistency reasons we reproduced this same scheme for stormd using M2Crypto.

To provide a complete solution, we implemented services such as DNS, DHCP, LDAP, and Kerberos, all of which are handled by the STORM manager and can be controlled through the provided Web interface. Appliances can reach the authentication and authorization mechanisms through the reserved internal DNS record "storm.local" and required configuration items can be obtained through the secure XML-RPC interface.

Traditional and layered appliances are distributed to individual cloud systems by the appliance developers. Each appliance developer should have at least one server available to feed RSS, disk images, and associated XML. For scalability, disk images should be provided from separate servers, or some kind of HTTP load balancer should be used.

The RSS supplied by the channel server is very generic and does not require many attributes to define a channel (see Figure 1). This same principle also applies to the XML associated with an appliance. We do not currently have any mechanism for developers to automatically generate the required RSS and XML. However, given the simplicity of the RSS and XML, it would be trivial to develop a tool to do this.

Layered Appliances

The traditional approach to virtual appliances replicates data by having a single disk image for each given application. When one obtains two traditional appliances that have software in common, this software is stored on the system twice.

For example, take an appliance that provides Wordpress and another that provides MediaWiki. Each requires an underlying substrate of software, such as MySQL, Apache, and PHP. With a traditional appliance the underlying software ends up being stored twice on separate disk images. To address this issue, we invented the concept of a layered appliance.

Layered appliances reduce redundancy by sharing common substrate. This provides several other benefits as well, including the ability to simultaneously apply updates across multiple instances, caching of data across multiple virtual machines, and the capability of taking snapshots of individual layers.

Our virtual appliance approach consists of four layers:

- A common operating system substrate: The substrate contains the basic components needed by all virtual appliances; the Ubuntu "Just Enough OS" (JEOS) platform is a representative example of this.

- An appliance-specific component: This component provides the application and necessary libraries; an example might be the Postfix program, LDAP and MySQL libraries for remote mail delivery, and other necessary libraries.
- A deployment-specific component: This 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 (e.g., the appliance component would typically include off-the-shelf configurations provided by an OS 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.
- An instance-specific component: This uses information provided by the STORM server to configure a specific instance of a more general appliance. For example, that instance-specific information may configure the domain name to be "mail.foo.com" instead of "mail.bar.com."

STORM and the UnionFS

In STORM we used a project called UnionFS [3] to provide the desired functionality for layering. UnionFS allows one to specify a series of directories and have them presented as a single virtual directory. We chose to implement the layering within the base substrate (operating system layer), allowing developers to build nonlayered appliances if so desired.

When a layered virtual appliance is deployed, several disk images are presented as devices and set as writable or read-only devices depending on the layer they provide. The init script within the initrd image mounts each device to a directory and then unions all of the directories appropriately to a single root. It will do its best to detect the write/read-only status of all devices; however, if the detection fails, then it relies on the following logic:

- /dev/sdaX mounts as read-only.
- /dev/sdbX mounts as read-write.
- /dev/sdcX mounts as read-only.
- /dev/sddX mounts as read-write, excluding /dev/sdd4, which is always reserved for swap.

An important thing to note is that UnionFS is not designed for I/O-intensive applications, and it adds significant overhead in these applications. These types of applications already suffer a substantial impact when being virtualized [4, 5]; therefore we suggest considering alternatives for any I/O-intensive application.

So Why Use UnionFS?

An alternative to UnionFS that immediately comes to mind is directly mounting disk images to the proper locations within the filesystem. We decided against doing this for several reasons, but the most important ones were:

- When using directly mounted disk images the developer would have to specify where the disk images should be mounted. The STORM manager would then have to get mapping information (i.e., specifying which disk image is mapped to what device) for every instance. Furthermore, conflicts can occur when using direct mounts. An example of this would be multiple disk images both needing to be mapped to /usr/bin.
- To share the operating system between multiple instances, the root directory would have to be marked as read-only. This can potentially lead to

data loss if an application attempts to write to a location that has no writable disk image mounted to it. This is somewhat of a limitation for current hypervisors, as they do not currently have locking mechanisms that would allow for better sharing.

- Developers and system administrators would have to learn yet another custom configuration management tool.

Overall, using UnionFS allowed for a much cleaner solution, because we didn't have to build the extra infrastructure to maintain and distribute directory-mapping information. It allowed us to have a single instance layer that sat on top of all the others with write capabilities only given to that specific instance.

Going Forward

The resurgence of virtualization and the construction of fiber networks have greatly impacted the information technology landscape. These two advances have made cloud computing competitive and reliable. Corporations, universities, and institutions that lack sufficient knowledge to capitalize on the advantages provided by virtualization are unable to move toward it. The STORM system successfully allows these entities to capitalize on virtualization without requiring large expenditures in virtual machine manager expertise.

One may ask, what things will cloud computing allow us to do next? Once things become more established, I imagine we will see capabilities for cross-cloud computing. Administrators will watch their virtual machines "pack their bags" and move out west for more sun (assuming data centers become solar powered [6]). Sophisticated and yet simple tools will be required to manage virtual machines in cross-cloud computing. It is our plan to continue STORM development as one of these management tools.

For more information and details about STORM, we suggest reading our paper published in the LISA '08 Proceedings [1].

REFERENCES

[1] Mark Dehus and Dirk Grunwald, "STORM: Simple Tools for Resource Management," *Proceedings of LISA '08: 22nd Large Installation System Administration Conference* (USENIX Association, 2008).

[2] The virtualization API: http://libvirt.org.

[3] David Quigley, Josef Sipek, Charles P. Wright, and Erez Zadok, "UnionFS: User- and Community-Oriented Development of a Unification File System," *Proceedings of the 2006 Ottawa Linux Symposium*, pp. 349–362, 2006.

[4] D.I. Wolinsky, A. Agrawal, P.O. Boykin, J.R. Davis, A. Ganguly, V. Paramygin, Y.P. Sheng, and R.J. Figueiredo, "On the Design of Virtual Machine Sandboxes for Distributed Computing in Wide-Area Overlays of Virtual Workstations," *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, p. 8.

[5] Paul Barham, 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, 2003.

[6] Stacey Higginbotham, "Data Centers Will Follow the Sun and Chase the Wind": http://earth2tech.com/2008/07/25/data-centers-will-follow-the-sun-and-chase-the-wind.