Provisioning of Virtual Environments for Wide Area Desktop Grids through Redirect-on-write Distributed File System

Vineet Chadha, David Wolinsky, Renato J. Figueiredo
Advanced Computing and information Systems Laboratory
University of Florida, Gainesville, FL
{chadha,davidiw,renato}@acis.ufl.edu

Abstract

We describe and evaluate a thin client solution for desktop grid computing based on virtual machine appliances whose images are fetched on-demand and on a per-block basis over wide-area networks. The approach uses a distributed file system redirection mechanism which enables the use of unmodified NFS clients/servers and local buffering of file system modifications during the appliance's lifetime. The file system redirection technique is achieved through user-level proxies, and can be integrated with virtual private network overlays to provide transparent access to image servers even if they are behind firewalls. We have implemented and evaluated a prototype system which allows thin client diskless appliances to boot over a proxy VM bringing on-demand only a small fraction of the appliance image (16MB out of 900MB) and showing low runtime overhead for CPU-intensive applications. The paper also presents decentralized mechanisms to support seamless image version upgrades.

1 Introduction

Virtual execution environments based on system virtual machines address some of the fundamental issues of distributed computing – legacy support, security and site-independent computation deployment. A key challenge arising in such "Grid" infrastructures is that of data management; in particular, the provisioning of the state (e.g. virtual disk images) required to instantiate a VM-based container "appliance" which provides all the necessary runtime execution environment for an application conveniently encapsulated in a VM image. Here, a virtual "appliance" is a VM based execution environment which supports voluntary sharing of user's resources and run unmodified applications. In this paper we present a generic, user-level distributed file system virtualization framework which enables multiple VM instances to efficiently share a common set of virtual machine image files. The primary goal is to facilitate the deployment of voluntary Grids deployed as wide-area virtual networks of virtual machines [10]. Specifically, we aim at reducing download times associated with appliance images, and providing a decentralized, scalable mechanism to publish and discover upgrades to appliance images.

Thin computing paradigms offer advantages such as lower administration cost and failure management. In early computing systems, thin-client computing was successful because of two main reasons: low-cost commodity hardware was not available to end users, and a centralized approach of computing is often preferred due to easier system administration. As low-cost PCs and high-bandwidth local-area networks became widely available, thin-client computing lost ground. The advent of virtual machines have opened up new opportunities; virtual machines can be easily created, configured, managed and deployed. The virtualization approach of multiplexing physical resources not only decouples the compute resources from hardware but provides a much needed flexibility of easily movable compute resources.

For data transfers over wide-area networks, and especially in voluntary-computing environments, available bandwidth is a bottleneck. Different solutions proposed to address bandwidth limitations include caching, data compression and quality of service (QoS) provided for applications. In the environment focused in this paper, limited bandwidth hinders the voluntary deployment of appliances because these often have hundreds of MBytes of virtual disk state — a 600-MB VM image takes more than one hour to download in a 1Mbit/s link, which is a significant deterrent for end users. Optimizing the size of an appliance is time-consuming, and in many cases not possible without loss of functionality (e.g. by avoiding installation of certain packages). Nonetheless, it is often the case that only a small fraction of the virtual disk is actually "touched" by an application during run-time. We exploit this behavior through on-demand data transfers that substantially reduce

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

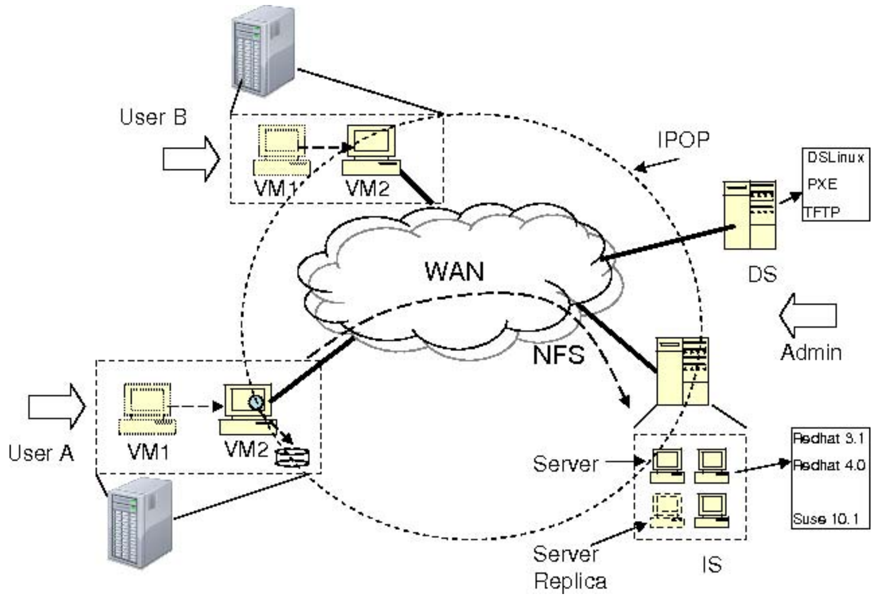

Figure 1. Illustration of O/S image management over wide area desktops: User A downloads a small ROW-Proxy configured appliance(VM2) from download server DS. VM2 appliance is configured to form a virtual overlay network using IPOP. Image Server (IS) exports read-only images to clients over NFS. VM1 is a client-configured diskless VM whose image is mounted over NFS through VM2.

the download time and bandwidth requirements for the end user.

We describe a thin client approach of deploying NFS proxies over a scalable virtual network. Our proposed environment allows sharing of read-only images among multiple clients and the re-direction of write access to a loop-back local client buffer. The solution is very generic and easily deployable as user level proxies which redirect traffic between local and remote conventional network file system servers, without requiring any kernel modifications. The approach embodies different components and technologies — hosted virtual machines, a P2P overlay network, pre-boot execution environment services, and a redirect-on-write virtual file system. Virtual Machines over P2P networks are deployed with pre-boot execution server to facilitate remote network booting. Similarly to related efforts, our approach targets applications deployed on non-persistent virtual containers [3][7] through provisioning of virtual environments with role-specific disk images [14]. Specifically, we show that our approach is suitable for low bandwidth CPU-intensive applications.

The rest of this paper is organized as follows: In section 2, we discuss motivations behind the current work. Section 3, we describe the proposed framework of provisioning images over wide-area networks. We present an evaluation of our architecture in Section 4. Section 5 describes related work. Finally, we conclude our findings and future work in section 6.

2 Motivation and Background

Grid computing is moving towards virtual execution environments as its baseline technology. For example, virtual machines have been deployed over the grid for seamless execution of scientific applications[3]. Often these experiments are transient in nature and the execution environment is dynamically setup. Even though middleware-controlled grid virtual file systems have been deployed to facilitate seamless access to data in such environments, often there are applications which require shared read-only data such as shared libraries between clients (e.g. VM instantiation) for performance improvements.

One important application of ROW-FS is supporting read-only access of shared VM disks, e.g. as supported by Xen and O/S distribution images stored in file systems to support rapid instantiation and configuration of nodes in a network. The redirect-on-write (ROW-FS [6]) capabilities, in combination with aggressive client-side caching, allow many clients to efficiently mount a system disk or file system from a single image – even if mounted across wide-area networks.

One particular use case is the provisioning of non-persistent VM environments for distributed computing. In this scenario, the goal is to have thin, generic boot-strapping

VMs that can be pushed to computational servers without requiring the full transfer or storage of large VM images. Upon instantiation, a diskless VM boots through a pre-boot execution environment (PXE) using one out of several available shared non-persistent root file system images, stored potentially across a wide area network.

This approach delivers capabilities that are not presently provided by VM monitors themselves, even if they support non-persistent environments. Without a block level distributed file system on the host, on-demand transfer of VM image files is not possible, thus the entire VM image would need to be brought to the client before a non-persistent VM could start. In voluntary Grid computing environments it is difficult to acquire privileges on the host to perform such file system mounts; in contrast, with ROW file system, the NFS-mounted file system can be kept inside a guest, and no host configuration or privileges are required to deploy the boot-strapping VM and the diskless VM.

3 Architecture

The overall architecture is depicted in Figure 1. The approach is based on diskless provisioning of virtual machine environments through a virtual machine proxy. The utility of the envisioned architecture can be observed from the viewpoint of both users and system administrators. Users not only have fast and transparent access to different O/S images but also have automatic support to upgrade the O/S images. For administrators, it provides a framework for simple deployment and maintenance of new images.

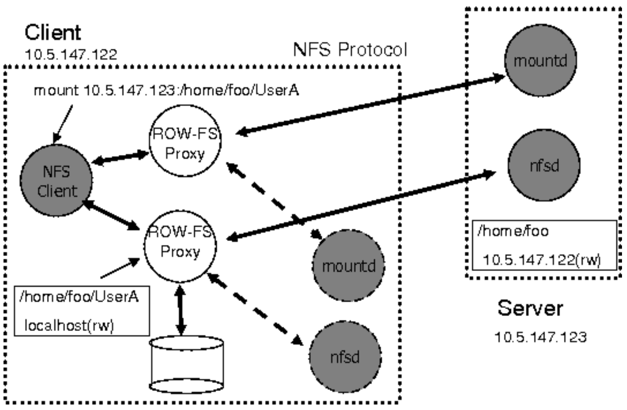

Figure 2. Redirect-on-write filesystem: User-level proxy virtualizes NFS by forwarding write calls to a shadow server. NFS client transparently access the read-only images (/home/foo/userA) from Server VM. NFS client is configured to run shadow NFS server and ROW-FS proxies.

As shown in Figure 1, an end user A downloads a small proxy appliance from Download Server DS. The proxy appliance is configured to connect to a virtual network overlay connecting it to other users (e.g. using IPOP[11][9]). The proxy appliance is also configured to run a small ftp server and DHCP server to download network bootstrap program and allocate IP address to client's working environment. The actual appliances which carry out computation can be configured with a desired execution environment and need not be downloaded in their entirety by end users - they are brought in on-demand through the proxy appliance. Each node is an independent computer which has its own IP address on a private network.

Key to this architecture is the redirect-on-write file system (ROW-FS) which is described in [6]. ROW-FS consists of user-level DFS extensions that support selective redirection of DFS calls to two servers: the main server and a copy-on-write server. The architecture is novel in the manner it overlays the ROW capabilities upon unmodified clients and servers, without requiring changes to the underlying protocol.

The ROW-FS approach relies on the opaque nature of NFS file handles to allow for virtual handles that are always returned to the client, but map to physical file handles at the main and ROW servers. The file handle hash table stores such mappings, as well as information about client modifications made to each file handle. Files whose contents are modified by the client have "shadow" files created by the ROW server in a sparse file, and block-based modifications are inserted in-place in the shadow file. ROW-FS proxy maintains the client's session state in a set of data structures - hash table and bitmap [6]. While hash table keeps the file handle mapping between main and shadow NFS server, a presence bitmap marks which blocks have been modified, at the granularity of NFS blocks. Figure 2 shows one of the possible deployments of ROW-FS proxy. A user-level proxy intercepts client's mount request and stores the mount file handle in a temporary location. This file handle is used by *nfsd* ROW-FS proxy to direct read/write calls to the servers.

A related copy-on-write approach is implemented in UnionFS [15]. The key advantages of ROW-FS over UnionFS are that the former is user-level and integrates with unmodified NFS clients/servers, while the latter is a kernel-level approach that requires support from the kernel, and that the former operates with individual file data blocks while the latter operates on whole files. UnionFS copy-on-write mechanism is based on copy-up complete to new branch on write invocation whereas ROW-FS just replicates the needed block; hence, ROW-FS has added advantage over UnionFS for instantiation of disk images, where a large copy-up is expensive.

Figure 3 illustrates a deployment of ROW-proxy in a

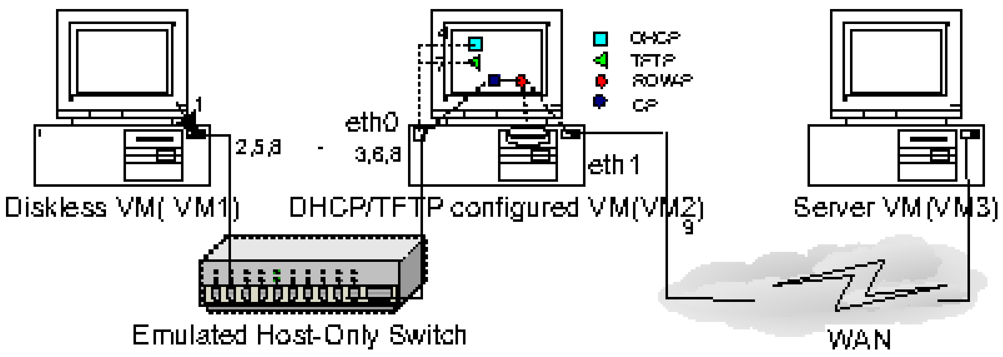

Figure 3. Illustration of the use of the ROW proxy to support PXE-based boot of a (diskless) non-persistent VM over a wide area network: VM1 and VM2 are deployed on the client's desktop - VM1 is client-configured diskless VM, VM2 is a downloadable VM appliance configured with DHCP, TFTP Server and NFS Proxies. Server VM (VM3) exports read-only O/S image.

diskless client environment. In Figure 3, VM1 is a diskless virtual machine, and VM2 is a boot proxy appliance configured with two NIC cards for communication with both host-only and public networks. VM2 is configured to execute ROW file system (ROW-FS) and NFS cache proxies. In addition, VM2 is configured to run DHCP and TFTP servers to provide diskless client an IP address and initial kernel image to VM1. Classic virtual machines such as VMware provides support for PXE-enabled BIOS and NICs; PXE is a technology to boot diskless computers using network interface cards. The server VM is configured to share a common directory through ROW-FS to clients. To illustrate the workings of our approach, consider the following steps to boot a diskless VM with an appliance image served over a wide-area network:

1. Diskless VM (VM1) invokes DHCP request for an IP address

2. DHCP request is routed through a host-only switch to gateway VM (VM2)

3. VM2 is configured to have two NICs: host-only (private IP address) and public. VM2 receives request at host-only NIC (eth0)

4. DHCP Server allocates an IP address and sends a reply back to diskless VM (VM1)

5. Diskless VM invokes TFTP request to obtain network bootstrap program and initial kernel image

6. VM2 receives TFTP request at host-only eth0

7. Kernel image is transferred to VM1 and loaded in RAM to kick start boot process

8. Diskless VM invokes mount request to mount read-only directory from Server (VM3)

9. VM2 is configured to redirect write calls to a local server. Read-only NFS calls are routed through the proxy VM2 to VM3; the connection between VM2 and VM3 is through the virtual overlay network

The primary goal of the architecture is to automate the process of publishing, discovering and mounting appliance images. Furthermore, it should be possible for images to be replicated (fully or partially) across multiple virtual servers throughout a virtual network for load-balancing and fault-tolerance. It is feasible to provide image versioning capability through maintaining the latest image state in a decentralized way using a Distributed Hash Table (DHT) — which, in the case of the IPOP virtual network [9], is already responsible for providing DHCP addresses. DHTs provide two simple primitives: put(key, value) and get(key). In order to use the DHT to track appliance image versions, the key functionality needed can be broken down into client-side and publisher-side.

- Client-side: a client should be able to query which version Vi is the most recent for an appliance A, and for the virtual IP addresses of one or more servers which have the image for A available for use. This information is used at boot-time by the proxy VM to select an appropriate server to mount a read-only image over ROW-FS, and can also be used to redirect calls in case of server failures.

- Client-side: a client should be able to periodically publish that it is using version Vi of appliance A, notifying image publishers that such version is in use and should not be removed

- Publisher-side: a developer should be able to publish that a new version Vj of appliance A has been created, along with the identifiers (IP address and mount path) of where the image is available
- Publisher-side: a developer should be able to query for the total number of sharers for a given version Vi of appliance image A in order to make decisions about garbage-collecting versions which are no longer in use. E.g. if there are no users of image versions i through j of an appliance A, and the current appliance version is $k > j$, a publisher may decide to remove versions i through j to make storage for additional versions available

A first-order approach to deal with this problem using a DHT is to support three tables. One is indexed by the appliance identifier A, assumed to be unique, which holds as a value the latest version associated with A. A second table is then indexed by a tuple (A,Vi), and stores as values the (IP,mount path) tuple pointing to available servers for this image. A third table, also indexed by (A,Vi), holds a value "1" for each client currently using the image. This value has a limited time to live (TTL) and must be refreshed by each client before the expiration of the TTL; summing up all the values associated with this key gives an estimate reference count on the number of sharers of an image. The goal is to store the state information such as reference count of the number of current users of version Vi of appliance A.

4 Experiments and Results

This section evaluates the feasibility and performance of our approach through quantitative experiments. First, we evaluated the overhead associated with the use of a proxy VM for on-demand access to the appliance image. We evaluate CPU, network and disk utilization during the appliance boot, execution of a CPU-intensive application, and reboot. Second, we profiled at the proxy VM the occurrence of RPC calls throughout the execution of an appliance to correlate with VM overhead. Third, we measured the image size replicated in the shadow server after diskless boot and the total amount of data fetched on-demand from the image server. Finally, we demonstrate a proof-of-concept implementation and measure the boot and reboot times for appliances deployed over the IPOP virtual network and in a realistic wide-area environment.

4.1 Proxy VM resource consumption

The experiments are conducted to characterize resource consumption (CPU, disk and network) by the proxy virtual machine (VM2 in Figure 3). In this experiment, virtual machines VM1, VM2 and VM3 are deployed in a host-only network using VMware's ESX Server VM monitor. Note that in this experiment we use the VM monitor as a tool to collect resource statistics summarizing the utilization of the ROW proxy appliance - we do not analyze the execution time of the application in a virtual environment. The experimental setup is as follows: Server VM3 is configured to have 2GB RAM, single virtual CPU. VM2 and VM1 are configured to have 1GB RAM, and also a single VCPU. The VMs are hosted by a dual Xeon 3.2GHz processor, 4GB memory server. The size of the image of the appliance is 934MB. We have characterized memory usage for ROW-proxies for an execution run for simplescalar application. Memory overhead for executing the ROW-Proxy application is very small (1-2MB).

Figure 4 shows time-series plots with CPU, disk and network rates for three different intervals. These values are obtained in 20-second intervals leveraging VMware ESX's internal monitoring capabilities. In the first interval, the VM is booted. In the second interval, the VM runs a CPU-intensive application (the computer architecture simulator simplescalar) which models our target workload of a typical voluntary computing execution. In the third phase, the appliance is rebooted.

VM Boot: Initially, the shadow copy-on-write server is empty; no file system state is present. During VM boot, the ROW-FS proxy accesses boot-time files from the server VM (VM3) and re-generates the file system hierarchy in the shadow server (local server) as described in [6]. The Figure shows high data write rates during VM boot execution. Clearly, we can observe a maximum of 12% CPU consumption and also a high data rate across the network to load the initial kernel image into diskless client (VM1) memory.

Application Execution: Since the application is CPU intensive, the proxy VM exhibits little run-time overhead in this phase — once the diskless VM (VM1) is booted and loads necessary files for the application execution into RAM on-demand (as shown by initial network activity). We can further observe that disk and network usage is negligible in VM2 during the execution of the simplescalar application, thus supporting our assumption of minimal overhead of proxy configured VM.

VM Reboot: During reboot, the client has replicated session state at the shadow server. We see an average boot time reduction and further spikes in network and CPU usage as some of files are fetched and read from the Server VM. In past results, we have shown that aggressive caching can further improve boot performance [6].

4.2 RPC call profile

Figure 5 provides statistics for number of RPC calls during the boot up of the diskless appliance VM1. The histogram is broken down by different types of RPC calls cor-

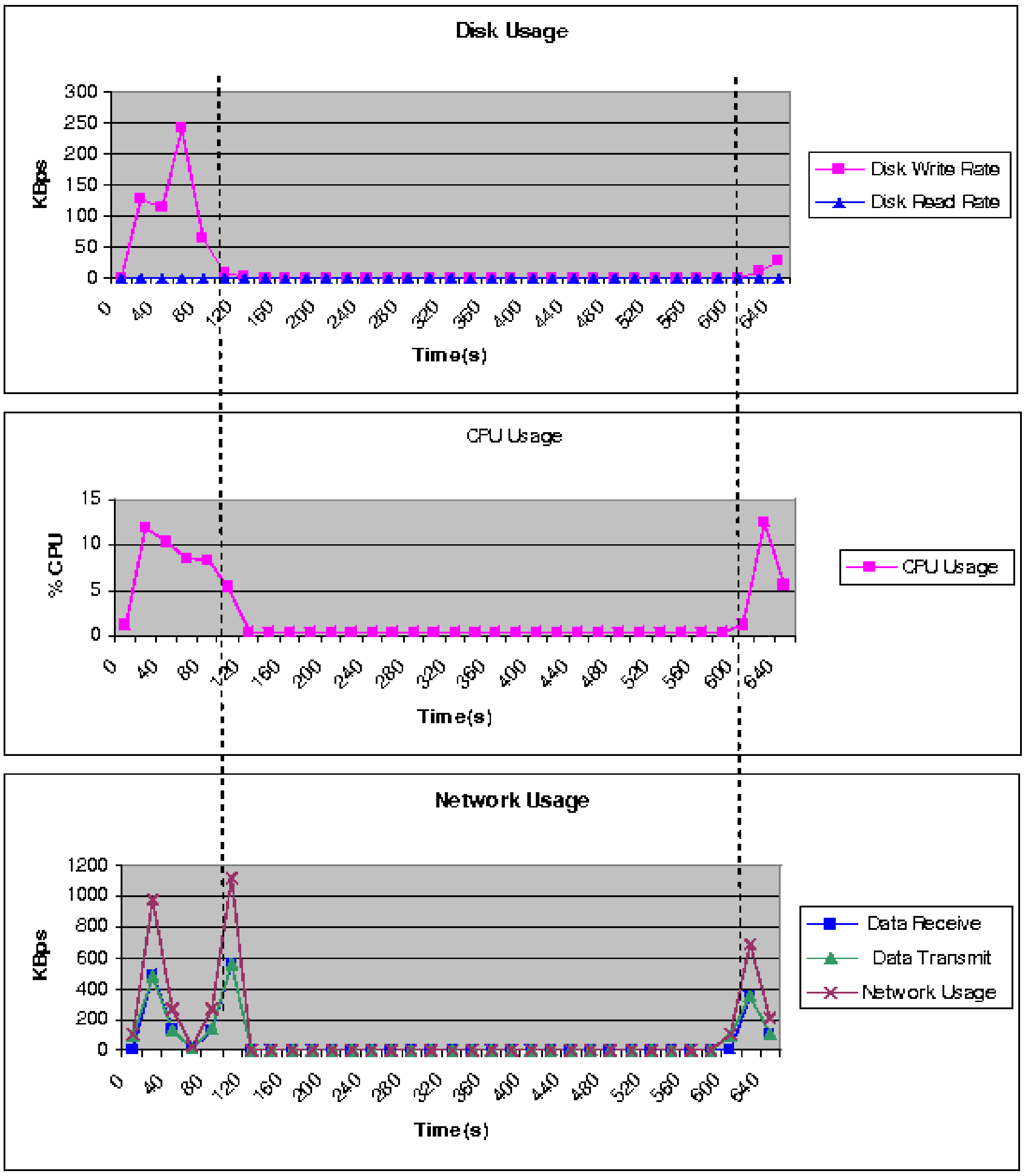

Figure 4. Proxy VM usage time series for CPU, disk and network. Results are sampled every 20 seconds and reflect data measured at the VM monitor. Three phases are shown (marked by vertical lines): appliance boot, execution of CPU-intensive application (simplescalar), and appliance reboot.

responding to NFS protocol calls, from left to right: get and set file attributes, file handle lookup, read links, read block, write block, create file, rename file, make directory, make symbolic link, and read directory.

The important conclusion taken from the data in Figure 5 is that ROW-FS, while increasing the total number of calls routed to the *local* shadow server (Proxy VM), substantially reduces the number of RPC calls that *cross domains* (calls to server VM). We observe that the number of get attribute (getattr) calls are increased due to invocation of getattr procedure to virtualize read calls. The read call virtualization is important to maintain consistency between returned attributes to the client from read call (from Server VM) and post-read getattr call (from proxy-VM). Since all getattr calls go to the local-area shadow server (Proxy VM), the overhead of extra getattr calls is small compared to getattr calls over WAN. For example, while the number of getattr RPC calls to shadow server is approximately 12500 for a boot sequence of diskless Linux, the number of attribute calls is nearly 1800 more than traditional NFS. This is because of 1800 read calls. Also, note that all write calls are directed to the shadow server.

4.3 Data transfer size

The RPC statistics confirm that the amount of data needed to boot the appliance and execute an application is far smaller than the entire appliance image. The total number of read calls is roughly 2000; at 8KB per block, the total amount of data brought in from the server is approximately 16MB, which is less than 2% of the image size at the server (934MB). Also, we observe that for VM boot with, only 646KB of data is created and redirected to the local shadow server. Because the VM proxy includes the ROW-FS redirection capabilities, the server VM3 is mounted read-only and NFS blocks can be aggressively cached in VM2's local disk.

4.4 Wide-area experiment

In the final experiment, we have measured appliance boot and reboot times in an actual WAN deployment. In the experiment, VM3 is deployed in one domain, and VM1 and VM2 are deployed on a different domain. The VMs are connected by the IPOP [9] virtual network, and the server and client VMs are behind NATs. The proxy VM2 is equipped with both the ROW-FS proxy and a NFS cache proxy. Table 1 summarizes the results from this experiment. Notice that the boot times reduce to less than half, becoming comparable to the LAN PXE/NFS boot time of approximately 2 minutes.

Table 1. Appliance boot/reboot times over WAN. ISP is a VM behind a residential ISP provider; UFL is a desktop machine at the University of Florida; VIMS is a server machine in Virginia.

VM1/VM2, VM3	Boot seconds	2nd Boot seconds	Ping latency
ISP, UFL	291	116	23ms
UFL, VIMS	351	162	68ms

5 Related Work

The Shark file system [4] provides mechanisms for scalable access to read-only data over the wide area network through cooperative caching and NFS proxies; our approach complements it by enabling redirect-on-write capabilities, which is a requirement to support our target application environment of NFS-mounted diskless VM clients. SFS advocated the approach of read-only file system for untrusted clients[8]. There have been upcoming commercial products which either provide thin-client solutions based on pre-boot execution environment [2] or provides cache based solution as a viable thin-client approach for scalable computing [1].

Distributed computing approach based on stackable virtual machine sandboxes is advocated in [16]. Stackable storage based framework is also used to automate cluster management as means to reduce administerial complexity and cost [14].The approach advocated in [14] is reprovisioning of application environment (Base OS, Servers, libraries and application) through role-specific(read or write) disk images. A framework to manage cluster of virtual machines is proposed in [13]. Stork package management tool provides mechanism to share files such as libraries and binaries between virtual machines[5]. A copy-on-write file server is deployed to share immutable template images for operating systems kernels and file-systems in [12]. This approach uses a combination of traditional NFS for read-only mount and AFS for aggressive caching of shared images [12].

6 Summary and Conclusions

In this paper, we describe a framework that automates appliance updates without requiring administrator intervention and enables on-demand transfer of appliance state with local buffering of modifications. We propose a ROW-FS capability to support read-write operations over virtual IP network overlays. Experiments show that proxy-configured VMs consume few resources except during bootstrapping. We have deployed an initial prototype and determined the

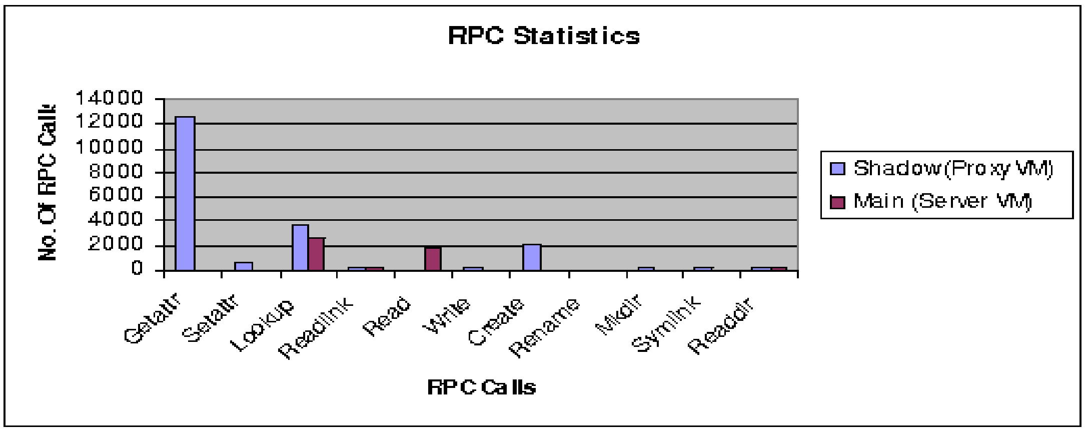

Figure 5. RPC statistics for diskless boot

feasibility of the proposed approach. Current work is investigating the integration of DHT-based publish-query of appliance location and version, garbage collection, and transparent fail-over to replica servers.

References

[1] 2x computing. http://www.2x.com/.

[2] Moka5. http://www.moka5.com/.

[3] S. Adabala, V. Chadha, P. Chawla, R. J. O. Figueiredo, J. A. B. Fortes, I. Krsul, A. M. Matsunaga, M. O. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From virtualized resources to virtual computing grids: The in-vigo system. *Future Generation Computing Systems,special issue on Complex Problem-Solving Environments for Grid Computing*, 21(6), Apr 2005.

[4] S. Annapureddy, M. J. Freedman, and D. Mazires. Shark: Scaling file servers via cooperative caching. In *2nd USENIX/ACM Symposium on Networked Systems Design and Implementation*, May 2005.

[5] J. Cappos, S. Baker, J. Plichta, D. Nyugen, J. Hardies, M. Borgard, J. Johnston, and J. H. Hartman. Stork: Package management for distributed vm environments. In *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, Nov 2007.

[6] V. Chadha and R. J. Figueiredo. Row-fs: A user-level virtualized redirect-on-write disttributed file system for wide area applications. In *International Conference on high Performance Computing(HiPC)*, Goa, India, Dec 2007.

[7] J. Chase, D. E.Irwin, L. E.Grit, J. D.Moore, and S. E.Sprenkle. Dynamic virtual clusters in a grid site manger. In *Proceedings of 12th HPDC*, Jun 2003.

[8] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.

[9] A. Ganguly, A. Agrawal, P. Boykin, and R. J. Figueiredo. Ip over p2p: Enabling self-configuring virtual ip networks for grid computing. In *IEEE International Parallel Distributed Processing Symposium (IPDPS)*, Rhode Island, Greece, Apr 2006.

[10] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. Wow: Self-organizing wide area overlay networks of virtual workstations. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 30–41, June 2006.

[11] A. Ganguly, D. Wolinsky, P. O. Boykin, and R. Figueiredo. Decentralized dynamic host: Configuration in wide-area overlay networks of virtual workstations. In *Workshop on Large-Scale and Volatile Desktop Grids (PCGrid)*, Mar 2007.

[12] E. Kotsovinos, T. Moreton, I. Pratt, R. Ross, K. Fraser, S. Hand, and T. Harris. Global-scale service deployment in the xenoserver platform. In *Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS '04)*, Dec 2004.

[13] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An Extensible Framework for Managing Clusters of Virtual Machines. In *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November 2007.

[14] F. Oliveira, G. Guardiola, J. A. Patel, and E. V. Hensbergen. Blutopia: Stackable storage for cluster management. In *Proceedings of the IEEE cluster computing*, Sep 2007.

[15] D. P. Quigley, J. Sipek, C. P. Wright, and E. Zadok. UnionFS: User- and Community-oriented Development of a Unification Filesystem. In *Proceedings of the 2006 Linux Symposium*, volume 2, pages 349–362, Ottawa, Canada, July 2006.

[16] D. Wolinsky, A. Agrawal, P. O. Boykin, J. Davis, A. Ganguly, V. Paramygin, P. Sheng, and R. Figueiredo. On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations. In *First Workshop on Virtualization Technologies in Distributed Computing (VTDC)*, Nov 2006.