VOL. 1, NO. 1 – APRIL 2012

VMWARE TECHNICAL JOURNAL

Editors: Steve Muir, Rita Tavilla and Ben Verghese

TABLE OF CONTENTS

vmware®

VMWARE'S RESEARCH PARTNERSHIPS

The VMware Academic Program (VMAP) supports a number of academic research projects across a range of technical areas. We conduct an annual Request for Proposals (RFP), and also support a small number of additional projects that address particular areas of interest to VMware. The topic for our Spring 2012 Request for Proposals was Security for Virtualized and Cloud Platforms, with funding awards to be announced in May 2012. Our previous RFPs addressed Performance Management Challenges in Virtualized Environments (Spring 2011) and Cloud Computing (Spring 2010). A selection of our past and present research partnerships include:

- **Data-Mining Virtual Machines for Resource Optimization**
  Una-May O'Reilly, Saman Amarasinghe, Skye Wanderman-Milne, MIT, Computer Science and Artificial Intelligence Laboratory (CSAIL)
- **Managing Datacenter Assets under Capacity and Power Constraints**
  Karsten Schwan, Ada Gavrilovska, Georgia Institute of Technology, Center for Experimental Research in Computer Systems (CERCS)
- **A Robust Framework of Conversational Assurance for Distributed Systems based on Multiparty Session Types**
  Nobuko Yoshida, Rumyana Neykova, Imperial College, London; Kohei Honda, Queen Mary, University of London

Performance Management Challenges in Virtualized Environments (Spring 2011 RFP)

- **Energy Efficient and Reliable Server Consolidation for the Cloud**
  Ayse Coskun, Boston University
- **Hardware Performance Monitoring of Scientific Applications in VMware Environments**
  Shirley Moore, Dan Terpstra, University of Tennessee
- **Flexible Computing with VM Fork**
  Eyal de Lara, University of Toronto

Cloud Computing (Spring 2010 RFP)

- **A vCloud at Carnegie Mellon: Establishment, Automation, Measurement, and Evolution**
  Greg Ganger, Carnegie Mellon University

Hello and welcome to the inaugural issue of the VMware Technical Journal! This journal is completely driven by VMware engineers who enjoy telling the details of interesting projects and products, sharing their passion for virtualization and cloud computing, and celebrating the innovation occurring all around our industry.

It's definitely an exciting time to be at VMware. We're at a point in time when nearly 60% of all server applications in the world are running virtualized, the large majority of those on VMware's virtualization platform. We're also becoming very popular in the "desktop computing" space as enterprises rapidly move into the Post-PC era characterized by mobile devices, SaaS applications, and anywhere/anytime computing. This popularity means that many great partners want to integrate with our products, that our innovations can have a dramatic impact, and that we have an opportunity to extend our products into many adjacent technology areas. As a result, our engineering team ends up working in almost every area of system software.

And the VMware Technical Journal appropriately reflects this breadth of focus. This inaugural issue includes articles on our core vSphere technologies as well as on our newer SaaS-related management tools. We're also providing a look behind the product creation process with articles that cover the testing of vSphere quality, measuring performance, and building great user-experiences. We also close the journal with a great look at using all of these products in a very high-stress, high-demand environment… the VMworld Labs!

We sincerely hope you enjoy this first edition. We welcome your comments and ideas for future articles. Until next time, happy computing!

Steve Herrod
CTO, VMware

VisorFS: A Special-purpose File System for Efficient Handling of System Images

Olivier Cremel
VMware
ocremel@vmware.com

Abstract

In reaction to the complexity of classic VMware® ESX®, simplicity was an overarching design driver for VMware® ESXi™. A very simple runtime file system with specific features was needed to allow the use case—stateless and embedded—that was envisioned for ESXi at the time. Those features turned out to be quite useful, as ESXi evolved beyond its original intent. This paper reveals what is special about visorfs and how it enables the efficient use of memory, clean handling of system images, and proper definition of system state.

1. Introduction

The ESX architecture relies on a Service Console to handle three main tasks:

- **Booting**. The Service Console is booted in total control of the host, and is responsible for loading the ESX kernel (VMkernel) and handing control to it.
- **Handling some devices, such as USB or BMC**. Once the VMkernel is in charge of the machine, the Service Console is lifted up as a pseudo virtual machine that has direct access to the host.
- **Providing a user-level environment**. The VMkernel offers limited user-level support for the exclusive benefit of the virtual machine executable (VMX), the user-level side of the virtual machine monitor. All other user-level needs are handled by the Service Console, such as agents and shells for management.

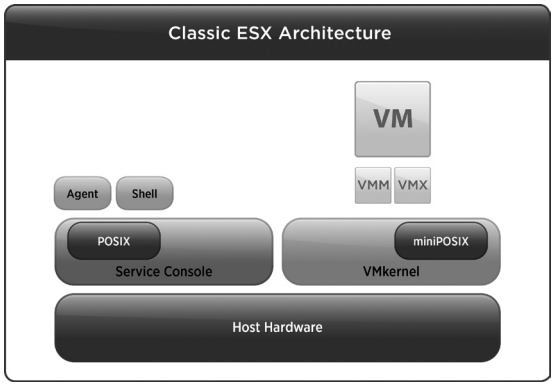

Figure 1. VMware ESX architecture

Two kernels vying for the same hardware, and needing to be synchronized through upgrade cycles, led to inherent complexity in maintaining the code base and managing the ESX system. To facilitate the deployment and management of thousands of ESX systems at one extreme, and the availability of very small turnkey configurations at the other, simplifying the ESX architecture became paramount.

The obvious angle of attack was to see how the Service Console could be excised and have all of its roles handled by the VMkernel. It became clear this required a radical rethinking of how ESX worked. ESX needed to change from being seen as just another server to a clean platform that delivers virtualization simply and easily. In practice, that meant changes, such as the absence of installation (system images baked offline) and the push for statelessness (external authority instead of local configuration) that drove the design of what became ESXi.

1.1 Taking Over the Service Console Roles

Booting was handled by adding a thin layer to the VMkernel to make it multi-boot compliant. In addition, an existing bootloader was used that understood the multi-boot standard, while handling additional devices required writing the appropriate drivers for the VMkernel.

ESX already provided user-level support for the benefit of VMX with a combination of kernel code and the GNU C library (glibc) to deliver a limited POSIX environment. That environment was expanded to include more POSIX interfaces, and an integrated set of tools (busybox) was added to allow shell functionality.

At that point, a file system was required. The only one available, and understood by the VMkernel, was the VMware® vStorage Virtual Machine File System (VMFS). VMFS was designed for virtual machine workloads, huge files that are kept open for long periods of time and minimal file system metadata activity. These characteristics made VMFS poorly suited for a user-level environment that needed something very simple:

- Support at most 256 files (glibc, busybox, agent)
- Have negligible metadata overhead
- Mostly deal with read-only existing files (from the system image)
- Handle a few small, temporary, read/write files (no persistence)

These requirements were derived from the principle of statelessness and complete control of the user environment.

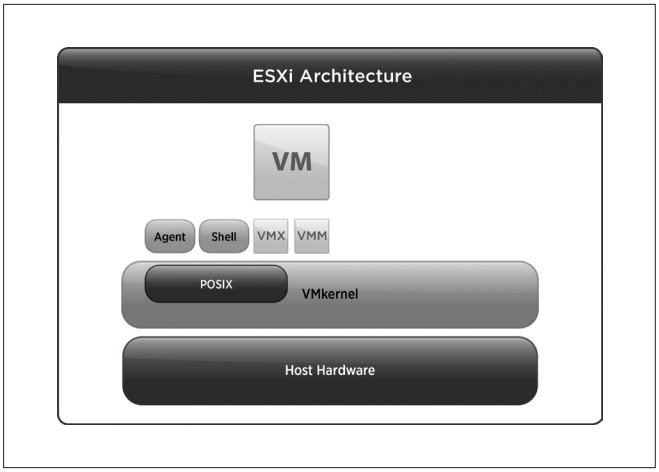

Figure 2. ESXi architecture

This article describes visorfs, the file system implemented to fulfill those requirements, and how it evolved along with ESXi. Its name is a simple contraction of hypervisor file system.

2. Basic VisorFS Concepts

2.1 Tardisk

Following the principles of no installation and statelessness, ESXi does not assume any storage. Its mode of operation is the same whether the system image comes from the network or from a local embedded flash memory device. Whatever the source, the bootloader gets a set of files and loads them into memory.

The first file is the VMkernel binary. The bootloader handles it appropriately and hands over control to it. The other files are left as is by the bootloader. Moreover, all of these files can be compressed, but the bootloader decompresses them on the fly so the content of the memory is usable directly.

The files left in memory represent the user-level environment (glibc, busybox, agents). For practical reasons, they are bundled into one or more tar archives when the system image is built, depending on the manner in which they are produced.

Once the VMkernel starts running, a handful of tar archives are resident in memory. It seemed wasteful to process and rearrange them into a different file system format while they were in memory and ready to be used. The tar archive format was kept as the underlying format for the file system. The file data remains the same, while the file tar header is used for the metadata since it already contains the name of the file and its length.

When visorfs is initialized, it starts with an empty array of descriptors (inodes). It is a small fixed array based on the requirement of handling at most 256 files.

Visorfs mounts the tar archives in order. It goes through the content of each tar file and parses it.

A file not previously encountered (with an original full pathname) is given an inode that points to the entry in the tar archive in memory. The inode also points to the parent inode (the inode of the directory in which the file resides based on its pathname). The parent must exist at that time. As a result arbitrary tar archives are not supported. Tar files must contain entries for directories prior to entries for the files in them. Complexity is pushed to offline tools so that simplicity can be kept in the running system.

A file already encountered is given the existing inode. The previous file is rendered inaccessible, but remains in memory as part of its tar archive. Although it is not prevented, this is unlikely to occur within a given tar archive. The typical use case is when the same file exists in different tar archives. There is strict ordering, and the last tar archive to be parsed wins.

In a typical file system implementation, tar archives would be untarred, their content used to populate the file system (usually resulting in copies), and then discarded. Once untarred, their individuality is lost. Unless external journaling is used, it is not possible to undo the untar operation.

In visorfs, tar archives are mounted. While their content is scattered across the namespace, they retain their integrity. In addition, tar files can be overlaid and unmounted, making all of the content disappear from the namespace at once. This is akin to a loopback mount with integration into the main namespace instead of hanging at a mount point. This is an essential characteristic of visorfs. The tar archives are referred to as tardisks due to their first-class status.

It should be noted that directory entries are just names and do not contain a list of their members, as that is the way they exist in the tar archive. Listing the contents of a directory requires walking the entire list of inodes and finding the ones pointing to it as their parent. Similarly, resolving a pathname requires walking the list for each segment. Considering the small number of inodes, this strategy is acceptable.

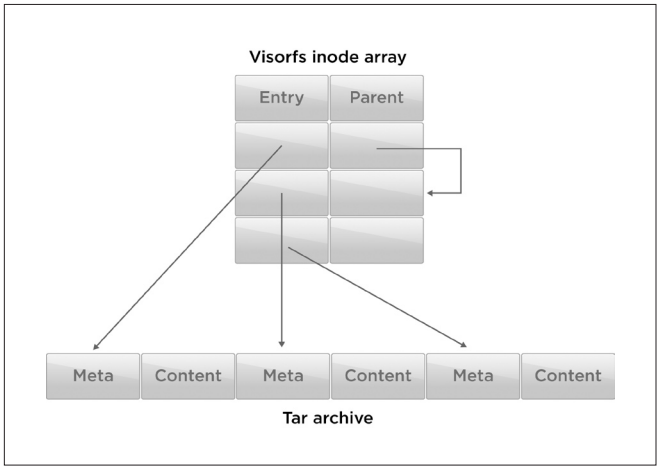

Figure 3. Tardisk

Keeping the tar archive layout implies that the file contents are contiguous and packed. It is not possible to extend a file without adding logic that stores extensions elsewhere and links them to the original entry. Since the requirements stated that the files would be read-only, visorfs simply forbids their modification and extension is not an issue.

2.2 Ramdisk

The requirements also called for the ability to create, modify, and delete a handful of temporary files.

By definition, a tardisk cannot be modified. Consequently, visorfs treats any file creation as the creation of a temporary file. An inode is allocated and pointed to the appropriate parent inode. Its entry starts as void, indicating an empty file. Whenever a write operation occurs, a sufficient amount of memory is allocated from a special visorfs pool to hold the content and the inode which points to it. On subsequent modifications, if the memory is sufficient it is reused, otherwise it is reallocated with a larger block to ensure the file contents remain contiguous.

This strategy keeps most operations from needing to differentiate between such a file or a file coming from a tardisk. The content of a file always starts at a given memory address and extends contiguously. As long as the files are few and small, it is not an issue.

The special pool of visorfs memory that contains such a file is called a ramdisk. It is possible to create different pools by mounting a ramdisk. In this case, mounting a ramdisk simply means allocating a pool of a given size and tagging the mount point inode so that any ramdisk file created under it goes into the pool.

2.3 Mounts

Externally, visorfs implements a POSIX file system. All the expected operations are available on top of the internal tardisks and ramdisks, including directory and file creation and deletion, attribute management, and even symbolic links.

Some operations might be a little disconcerting. Mounting a tardisk, and to a lesser extent mounting a ramdisk, do not follow the expected semantics. The use of "mount" should be regarded as a convenient shortcut to describe the actual operations.

Similarly, operations that report used space and free space can return counterintuitive results due to the way memory is allocated for files in ramdisks.

2.4 Branching

Quite early on, the strict separation between read-only files from the system image and read/write files as a by-product of the running system proved untenable.

Even though the system is stateless there are state files. Statelessness simply implies:

- The master state is inherited on boot from an authoritative entity.
- State changes (triggered locally or remotely) are recorded by the authority so that on shutdown and reboot the correct state is inherited.

While the system is running, the state must be maintained. Unfortunately, legacy and third-party tools keep state information in files under the /etc directory. It would have been extremely difficult to replace or adapt all of them to behave differently.

The files comprising the initial state cannot be part of a tardisk since they would not be modifiable. At first, the solution was to boot with a fixed state and obtain the actual state that could be created as files in ramdisk. However, there is a lot of state information that is set and typically does not change. This creates an undue burden on the ramdisks. Moreover, booting with a set state and switching to an actual state opens a window of non-compliance, most notably with passwords. Finally, it turns out that debugging tasks often require the ability to modify files in tardisks.

The tardisk semantics were modified to allow some files to be modified through a branching mechanism. One attribute bit, the sticky bit—which had no meaning on visorfs files—was repurposed. Any file with the sticky bit set in the tardisk could be branched.

When such a file is modified, the original content in the tardisk memory remains unchanged but is reassigned to a different inode and is renamed with a .# prefix. At the same time, a copy is made in the ramdisk that is associated with the original name and inode. From that point forward, the file is indistinguishable from a standard ramdisk file and can be modified as needed.

A safeguard prevents .# files from being created by anything but the branching mechanism. It also prevents .# files from being branched a second time.

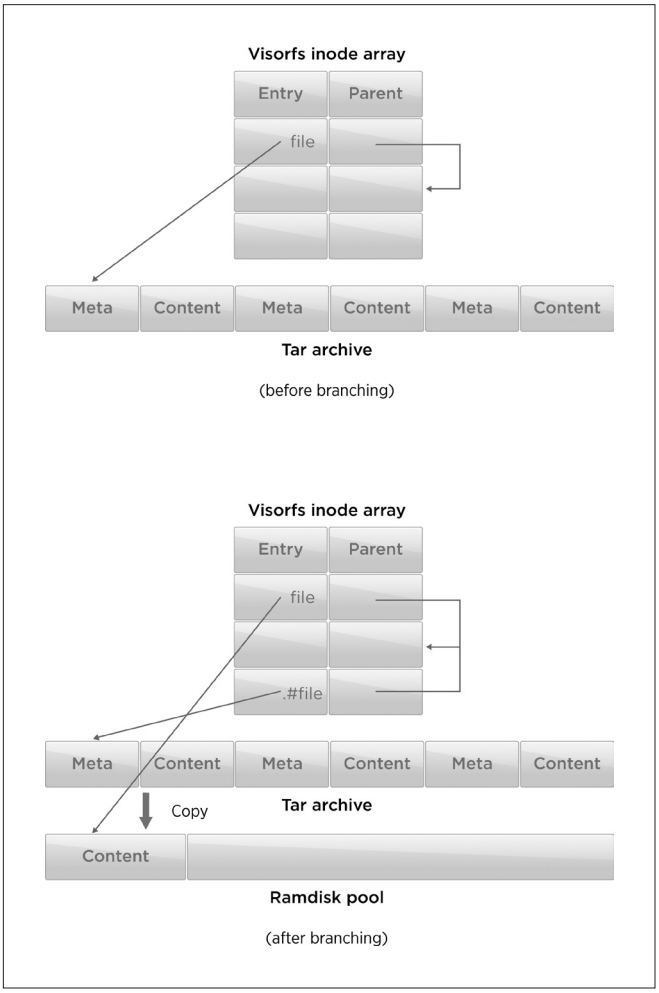

Figure 4. Branching

It was deemed useful to provide a notification mechanism when such branching occurs, enabling external entities to monitor state changes from legacy tools that normally do not report changes. State information must be properly tracked by the outside entity to ensure statelessness.

Interestingly, the notification mechanism was not used. Instead, the appearance of .# files in the namespace was monitored, as it was a more familiar way of doing things.

3. Memory Management

Keep in mind that visorfs is a completely memory-based file system. Both tardisks and ramdisks are memory resident. While this is a consequence of the file system's evolution, it means its footprint directly affects the consolidation ratio of virtual machines per host. The original system had a 32MB tardisk footprint and a 1MB ramdisk footprint, along with a small 256 inode array.

As the system evolved to become the first released ESXi version, and overtake ESX as the converged architecture, it inevitably grew. As a result, the requirements wildly inflated compared to the original ones. Today, the tardisk footprint is approximately 400MB, while the ramdisk footprint is nearly 256MB. The inodes now number almost 10,000.

Even though those numbers are one to two orders of magnitude greater than originally envisioned, the basics of visorfs have remained fundamentally the same. Implementation improvements have mitigated some drawbacks.

3.1 Page Sharing

The VMkernel implements a powerful page sharing mechanism that saves memory among virtual machines, as most run similar software and have similar content.

When a user-level agent is launched, its binary is read into memory from visorfs in order to be executed. Two copies of the binary exist in memory, one allocated page by page as the executable is loaded, the other one part of the tardisk.

In order to save memory, the VMkernel page sharing mechanism can be leveraged to avoid content duplication. However, two issues must be addressed:

- Tar archives are not naturally page aligned.
- The launcher needs to know how many pages will be shared. That cannot be the entire binary. Some pages are altered to resolve dynamic linking.

Following the precepts of moving complexity to offline tools, and keeping the running system simple, a special tool was developed. It is a tar archive post-processor named vmtar.

When the system image is built, the system tar archives that become the tardisks are constructed as usual. These tardisks are processed by the vmtar tool, which reorders the content to page align the binaries and add the number of pages needing fix-ups to their metadata header.

The resulting file is close enough to a tar archive that minimal changes to the visorfs code were needed to process it correctly. Similarly, the launcher was modified slightly to hook the pages directly into the page sharing mechanism as the binary is loaded. In the end, the footprint due to binaries in tardisks is essentially free.

3.2 Resource Pools

Another powerful feature offered by the VMkernel for virtual machines is resource pools. A memory resource pool is managed explicitly with minimum and maximum reservation settings to provide guarantees to resource pool users, for themselves and against others.

Visorfs was modified to create and associate a memory resource pool to each tardisk and ramdisk, so that they all appear in the resource pool tree. This is particularly useful for ramdisks, as they can be created on demand and assigned resource pool settings that can be tuned in the field. Creators of ramdisks suffer from both sides, and tend to:

- Overestimate the minimum reservation, as they do not want to run out of memory and deal with error cases.
- Underestimate the maximum reservation, as they forget corner cases and want to appear reasonable.

Having the ramdisks as part of the standard resource pool hierarchy enables support organizations to circumvent problems that previously required explicit patches.

3.3 Implementation Refinements

While the basic implementation of inodes quickly became a problem as their number grew, solutions were well known, such as hashing names to avoid full array walks, or splitting the array by tardisks and ramdisks.

A more annoying issue was the handling of ramdisk files. It became untenable to reallocate and copy on file growth as the number of file operations and their size increased. Eventually, the implementation switched to a more standard model in which the file backing is allocated and deallocated page by page as needed.

Doing so meant the content of ramdisk files was no longer contiguous, and could not be described by an address and a length. This created a big difference between ramdisk and tardisk files which complicated the implementation. Each content manipulation operation had to switch based on the file type.

Even so, the way in which the early requirements shaped visorfs has proven beneficial—and if not for them, ESXi system management would be very different.

4. System Organization

The organization of visorfs with tardisks and ramdisks and the concept of branching, derive from fundamental ideas about the organization of the system.

4.1 System Image

- ESXi is not installed.
- There is a system image that is created offline with any number of sophisticated tools. The system image is well defined.
- The system image is applied to a host. It is independent from the host and is unalterable. At any given time, what is running on the host is known. Because the system image is pristine, it can be dumped

at any moment. (Although a tardisk file can be branched, the tardisk remains unmodified.) This is how the ESXi installer works: ESXi is booted from media and dumped onto the local disk.

- The system image consists of a collection of tardisks that are manipulated individually as opaque entities with no direct file manipulation. While it is possible to achieve the same results with a generic file system, package management system, and extra journaling, having the concept be an intrinsic part of the file system is a great help in maintaining the discipline.

4.2 System State

- ESXi is stateless.
- Statelessness often causes confusion, since state must exist somewhere. In the case of ESXi, statelessness is defined as having a well-defined state that can be handled by an external authority.
- State is built on top of the branching concept, and must be captured in sticky bit branded files residing in tardisks. In addition, state is well defined, in the sense that its extent is known when the system image is built—and can be handled by an external authority that monitors branching activity.
- System components that do not follow the rule are quickly caught as their state is lost on reboot.

Once again, the primary value of visorfs in this case is internalizing the concept at the file system level, preventing shortcuts from being taken by some system components.

5. Conclusion

Visorfs is a product of the early vision for ESXi—no longer a complex server but a simple opaque platform geared to deliver virtual machines well. In that regard, visorfs is far from a generic file system, but is tailored closely to a very specific platform. By promoting package handling (via tardisks) and state monitoring (via branching) to first-class operations, visorfs has created an environment in which the stated goals have been easier to keep in mind and reach.

A Software-based Approach to Testing VMware® vSphere® VMkernel Public APIs

Lan Xue
lxue@vmware.com

Sreevathsa Sathyanarayana
sreevathsa@cloudphysics.com

Thorbjoern Donbaek
donbaek@vmware.com

Ramesh Pallapotu
rameshp@vmware.com

James Truong
jtruong@vmware.com

Sriram Sankaran
sriram@vmware.com

Eric Lorimer
elorimer@vmware.com

Abstract

VMware provides co-development programs[1] for partners and vendors to engage in the development of advanced virtualization features and the adoption of VMware vSphere®[2], an open platform for virtualization and cloud computing. Through these programs, VMware has published APIs for partners to integrate their products with VMware Infrastructure™. Certain functionality requires partners to access, adopt, and collaborate in the development of specific APIs. It is important to define a testing methodology to ensure the interfaces meet a certain level of quality.

This paper presents an approach to test vSphere VMkernel public APIs (vmkAPIs), a set of published APIs open to qualified partners interested in building and certifying plugins, devices, and drivers for vSphere ESX™[3]. The proposed testing approach is described, along with an example for testing a particular set of storage APIs. To assist testing, a software-simulated storage architecture was designed, providing an error injection mechanism to the storage stack, and facilitating a hardware-independent test environment.

Keywords

Error injection, storage simulation, API testing.

1. Introduction

Traditionally, developers test APIs by writing kernel-level unit tests. Although unit testing provides fundamental test coverage by checking different combinations of inputs and matching outputs of the API, certain categories of APIs are more difficult to verify since they only function within a live system. Functional or system testing, on the other hand, invokes APIs indirectly by running workloads from user space. This form of test closely resembles how customers actually use the system. However, user space workloads are hard to map to kernel APIs and the code paths that are in response. Error handling code often is tested poorly, since error conditions are difficult to generate deterministically. This paper proposes an API testing approach that addresses these questions.

The scope of this paper is the testing of vmkAPIs, a set of VMware published APIs for partners and vendors to integrate plugins and drivers to the VMkernel[1]. Traditional kernel unit testing has significant limitations in test coverage, and functional black-box testing has little awareness of kernel space APIs in response to a specific test case. The approach described here tests APIs from the kernel level and end-to-end application level in a white-box testing environment. The test suite consists of test modules and scripts that carry kernel and user space test cases, and a kernel testing (sub)system that supports these test cases. The testing (sub)system can be a storage or networking subsystem in the kernel space, providing test developers with a white-box testing environment that offers great flexibility for designing and controlling tests. Section 2 explains the testing strategy in detail.

This paper also introduces an innovative, software-based storage simulation that provides the key functions of real storage arrays. The storage simulation has two major contributions for testing vmkAPIs:

- Provides a storage driver-level error injection mechanism that helps the testing of error handling in the VMkernel storage stack
- Provides a hardware-independent test environment that is highly portable

The rest of this paper is organized as follows. Section 2 introduces the overall test strategy. Section 3 describes the software-based storage simulation used in testing. Section 4 presents an example of using the proposed test approach to test several storage-specific vmkAPIs. Section 5 lists areas for future work, and Section 6 concludes the paper.

2. Testing Methodology

The vmkAPI testing approach described here is tied closely to the VMkernel. The VMkernel is a POSIX-like operating system developed by VMware in vSphere ESX. It provides functionality similar to that found in other operating systems, such as process creation and control, signals, file systems, and process threads[5]. Figure 1 shows an overview of the VMkernel architecture.

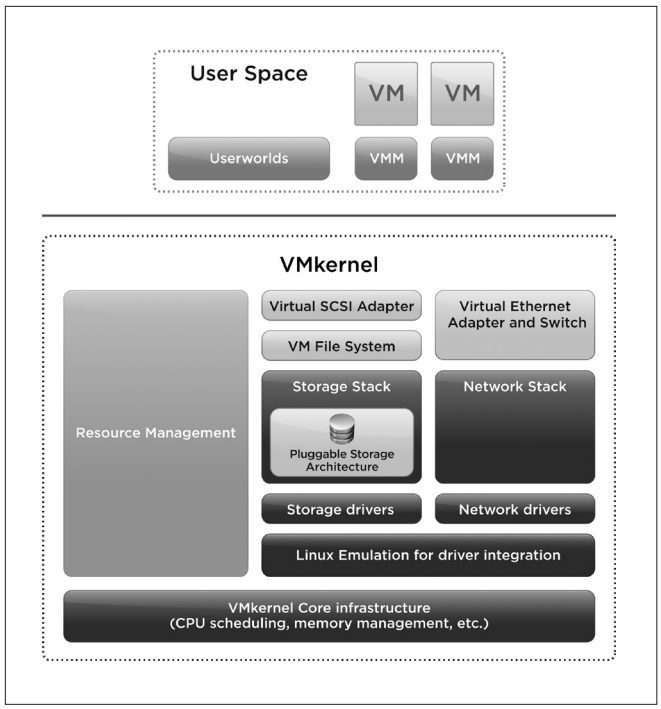

Figure 1. ESX VMkernel Architecture.

As vSphere is designed to be an open and extensible platform, the VMkernel has opened some of its subsystems for partners and vendors to integrate and co-develop certain features. For this purpose, VMware released several programming interfaces in the VMkernel, called vmkAPIs, that are currently in the following subsystems:

- Storage
- VMKcore
- Networking
- VMK Linux

The vmkAPIs are the programming interfaces between VMkernel subsystems and the external features from partners. As a result, properly functioning vmkAPIs are very important for smooth integration. Like any other part of the product, vmkAPIs must be carefully tested before they are released to partners. To test these APIs, a white-box testing approach is used. Section 2.1 gives an overview of the existing testing methods that motivate this work, and section 2.2 describes this approach.

2.1 Motivation

The most important requirement for any API is functional correctness. In this context, correctness means that each API functions properly under positive and stress conditions, has appropriate error handling under fault conditions, and follows the binary backwards compatible restriction accordingly.

Unit testing is a common, static, and ad hoc correctness verification method. Usually it means a dedicated and isolated test that validates a particular API by giving various inputs. These inputs easily can cover positive and negative conditions, thereby rendering good code coverage of the APIs under test. An API that uses or requires no specific additional hardware than CPU and memory, and does not have dependencies on side effects from other functions, can be tested very effectively with unit testing. For example, an API to copy a memory range or compute a checksum over a memory range can be tested easily in this way. Unit testing provides efficient correctness verification for APIs that function independently, without any prerequisite from the system.

There are two major limitations in using unit test for correctness verification. First, validating input/output pairs does not guarantee API semantic correctness. In many cases, semantic correctness is not straightforward to verify. Second, many APIs do not function in an isolated environment. Instead an API works in conjunction with, or follows, certain call sequences with other APIs. An API is called to respond to some events, and when it is called, it can lead to other consequences.

These kinds of APIs require a live (sub)system in order to be tested. For example, to change a path state in the kernel core storage layer, the test needs to work with an existing path in the system. It needs to call the API to set the path state, and then verify the result by querying the path state. It is impossible to test this API in a static unit test without participation of a storage device.

One way to address these limitations is to use black-box testing that runs workloads from user space in the hope that they invoke some of the kernel APIs. This approach yields an indirect and partial semantic verification and is a best-effort solution, as the testers lack knowledge of the kernel APIs being invoked by the user space workloads. Because of the disconnection between the test case and kernel code, debugging a user space test failure and root causing to kernel functions can prove difficult.

2.2 Testing Approach

The approach described in this paper can be called “API functional verification in a white-box testing environment”. The general definition of white-box testing is “a method of testing software that tests internal structures or workings of an application, as opposed to its functionality”[4].In the context of the kernel vmkAPI testing, white-box testing is used to provide a testing environment that has the following major advantages compared to the traditional testing methods discussed in Section 2.1, providing:

- A kernel working test system for vmkAPI unit testing
- A kernel test (sub)system for vmkAPI functional verification using user space workloads

- An error injection mechanism to test the error handling code
- A test infrastructure to ensure modules completely built on the vmkAPIs integrate successfully with the VMkernel

Test modules were developed that consume vmkAPIs with relevant functionality. Example includes vmkAPIs dealing with storage multipathing, or vmkAPIs handling networking packages. Such test modules do not just provide use cases for the vmkAPIs. As a whole, they can integrate with the kernel as a plugin providing specific functionality in a subsystem or a device driver. As noted earlier, some APIs require interaction with the system in order to function. After loading in the kernel, the pseudo plugin or pseudo driver creates a test (sub)system that can provide the prerequisite conditions to test those APIs.

The kernel test subsystem consisting of pseudo testing modules (plugins and drivers) also is used as the target kernel subsystem for user space workloads. This testing environment enables functional verification to the vmkAPIs, which is a very important aspect of vmkAPI testing. Executing user space workloads on the test subsystem provides the following functional verifications:

- General user workloads working on the native VMkernel should also function correctly after the VMkernel is integrated with the test subsystem.
- Test cases designed for the test subsystem should be successfully executed.
- The vmkAPI's error handling is tested under fault conditions that can be simulated as part of the white-box testing approach.
- The instrumented test modules provide additional information for debugging in the case of failure, or tracking the code path(s) invoked by a certain test case.

In addition to allowing functional verification from both kernel level and user space, the white-box testing approach can be used to verify that the API semantics of earlier release(s) is preserved. Some vmkAPIs are backward compatible from one (major) release to its previous release(s). As a result, modules written entirely on the backward compatible APIs from the earlier release(s) should be built with and loadable on the current release, and provide the same functionality as in the earlier release(s). Test modules were developed that consume APIs with the same compatibility restrictions. Therefore, porting test modules between releases, where binary compatibility is maintained, incurs no extra cost. A test breakage potentially can be caused by a breakage of the binary compatibility of the vmkAPI under test.

3. Software Storage Simulation

Deterministic error injection is one of the key values added by the software storage simulation, as described in Section 2.2. This section introduces the software simulated storage architecture, as shown in Figure 2, and a storage simulator implemented in this architecture. The software simulated storage architecture supports different types of simulated storage adapters, disk backends that can be chosen as data containers for I/O, hooks and controls for injecting errors at the storage driver level, utility functions, programing interfaces, and other services. It also provides an extensible infrastructure for integrating new features.

A storage simulator is a specific configuration of this architecture. A storage simulator can contain multiple pseudo storage devices, with each device consisting of a simulated driver and a disk backend. The device registers with the VMkernel after loading its driver, just as other storage arrays do, and is equivalent to a real hardware-based device from the VMkernel point of view. In the storage vmkAPI testing, a storage simulator was used that has one pseudo storage device with a simulated Fibre Channel driver supporting path disconnection error condition simulation and a memory-based disk backend (referred to as "ramdisk" in the rest of the paper) as a replacement for a real hardware Fibre Channel LUN.

Although most actively used for the VMkernel storage API testing, this software storage architecture is a standalone platform that provides a range of storage services, completely independent of real storage hardware, and can be a potential candidate where an actual storage device is not required. Following is a list of features supported by the software storage architecture:

- Storage adapters with different transport types, such as Parallel SCSI, Fiber Channel, and iSCSI
- Different backend disk types, such as ramdisk, nulldisk, and netdisk
- Dynamic binding between adapter and disk
- Driver-level error injection
- Application-level commands for non-kernel users to create, control, and delete simulated storage devices
- Kernel APIs that let test developers use the storage services without having to deal with the simulator's internal details
- An open architecture for new adapter and disk types, and other new features

3.1 Core Infrastructure

Figure 2 shows the software simulated storage architecture.

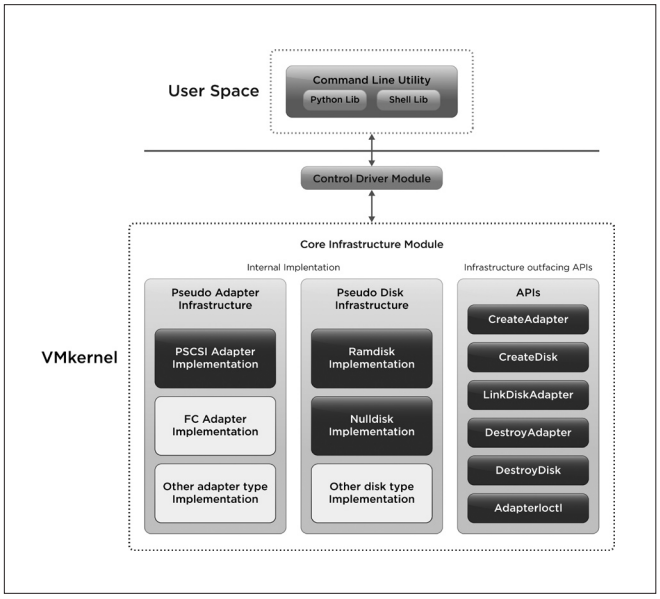

Figure 2. High-level design of the software simulated storage architecture.

The core infrastructure is implemented as a kernel module. It contains the pseudo adapter infrastructure implementation, pseudo device backend infrastructure implementation, and programmable APIs to help test developers write tests on top of the infrastructure.

3.1.1 Pseudo Adapter Infrastructure

The pseudo adapter infrastructure provides facilities to create a simulated adapter and its driver. Currently, Fibre Channel adapter simulation is implemented. The infrastructure provides interfaces to add or extend other types of adapters.

3.1.2 Pseudo Disk Infrastructure

The pseudo disk infrastructure is an extensible framework that provides facilities to create a particular type of simulated disk backend. Currently, ramdisk and null disk backends are supported.

The ramdisk is a type of backend that simulates a disk in physical memory. It is organized into grains. Each grain is an extent of machine memory (one contiguous chunk of machine pages), and is the granularity of memory allocations. During I/O, memory for grains is allocated when grains are first referenced and is never freed for the lifetime of the ramdisk. Memory is only freed when the ramdisk is destroyed.

This form of lazy allocation helps to create a sparse disk. The advantage of this technique is that large-sized LUNs (2TB+) can be created using only a small amount of memory, which mainly is used for storing the grain's metadata for internal tracking purposes. As an example, an empty 3TB ramdisk-backed LUN created with a VMFS5 partition on it consumes only 100MB of memory and approximately 48MB to store the grain's metadata.

The null disk is an experimental backend type that simulates /dev/null. It returns 0 for read requests and discards all write requests. A LUN with this type of backend takes the least amount of memory, and is useful in testing scenarios where a large number of LUNs is required.

3.1.3 Infrastructure Facility APIs

The core infrastructure exposes a set of APIs to assist test developers writing tests on top of the storage simulator. The APIs help test developers to programmatically create a test setup with a specified adapter type and a disk backend type of their choice. Once the required setup is created, the test module can use the setup to exercise the tests. For example, it is possible to choose a simulated Fibre Channel adapter, binding with a ramdisk with thin provisioning, as the storage device, along with other kernel test modules to test storage multipathing vmkAPIs, as shown in Figure 4. These APIs also give test developers the ability to control the storage simulator on demand. For example, developers can enable error injection on a specific device path during I/O to test path failover, and clear the error to simulate a path reconnection condition.

3.1.4 Error Injection

The software storage simulator supports ioctl commands to the adapter and path by exposing the ioctl APIs to the kernel callers. It also provides a command-line utility at the user level. Currently, two ioctl commands are implemented at the adapter level to simulate path connection conditions. One simulates path disconnection: the adapter processes no commands sent to the path with this condition, and each command returns with a VMK_HOST_NO_CONNECT status. The other ioctl command clears the no connection condition on the path. The adapter resumes processing all commands sent to the path after clearing the "no connection" condition. These ioctl commands have been used to simulate storage path failover and All Paths Down scenarios, as described in detail in Section 4.

3.2 Control Driver

The control driver is a character device driver that is used to provide application-level control for configuring and working with the software-based storage core infrastructure, as shown in Figure 2. This driver exposes a character device (/dev/padapterctl) that accepts a set of commands that can be used to setup and control the storage simulator. The control driver currently supports the following commands and operations:

- Create a pseudo adapter
- Destroy a pseudo adapter
- List pseudo adapters
- Add a pseudo disk to an adapter
- Send ioctl command to an adapter

Commands are received from the user space through a write to the character device. Depending on the command, the control driver uses the relevant test APIs exposed by the infrastructure to create the backend setup. Once this is done, the result is conveyed back to the user space through a read from the character device.

3.3 Command-line Utility

As shown in Figure 2, the infrastructure provides a command-line utility that is targeted mainly at users who want to create a test setup through the command line without having to perform any kernel programming. Python and shell libraries are provided that interact with the control driver.

4. Storage vmkAPI Testing

This section introduces the Pluggable Storage Architecture (PSA), an open infrastructure for partners to integrate their storage multipathing software with vSphere ESX. It describes the testing of the vmkAPIs for PSA, using kernel-level unit testing and API functional verification in a white-box testing environment. The software simulated storage layer provides error injection to test error handling in these vmkAPIs and PSA that otherwise is difficult to test without instrumenting the actual storage driver code.

4.1 Pluggable Storage Architecture in vSphere ESX

Multipathing software enables more than one physical path to be established between a server and its mass storage devices through buses, controllers, switches, and bridges. The redundant paths can be used to provide fault tolerance, dynamic load balancing, and traffic shaping. If one of these physical paths fails, data traffic can be routed to or from a storage device via one or more of the other physical paths. Usually, a specific version of multipathing software is installed on a server that is developed and tailored to provide a variety of functionality for its corresponding storage devices. If the server is connected to a storage device that does

not correspond to the server's multipathing software, the multipathing software typically provides limited functionality to that non-corresponding storage device[6].

To address this restriction, the Pluggable Storage Architecture, as part of the vSphere ESX VMkernel storage stack, was designed to provide an infrastructure that enables vSphere ESX to run multiple multipathing modules simultaneously. It comprises[6]:

- Scanning for physical devices
- Scanning for paths to each of the physical devices
- Presenting the paths to one or more multipathing software modules
- Claiming or rejecting one or more of the paths
- Creating and exposing one or more logical devices, wherein each logical device is associated with a multipathing module

The PSA provides a series of APIs that are used to write a multipathing software module to work in the VMkernel. These public APIs allow partners and third-party software vendors to port their proprietary multipathing and load balancing code to the vSphere ESX platform. SAN array vendors can develop device specific code that takes advantage of advanced array features without compromising proprietary information. In addition, vSphere ESX developers can add support for new arrays without increasing risk to existing array support code. The PSA enables several multipathing software modules to run simultaneously in the VMkernel, thereby leveraging the capabilities and functionalities available from different multipathing software. Figure 3 shows an example of VMware's Native MultiPathing (NMP) module, and EMC's PowerPath multipathing module co-existing and operating simultaneously within the VMkernel.

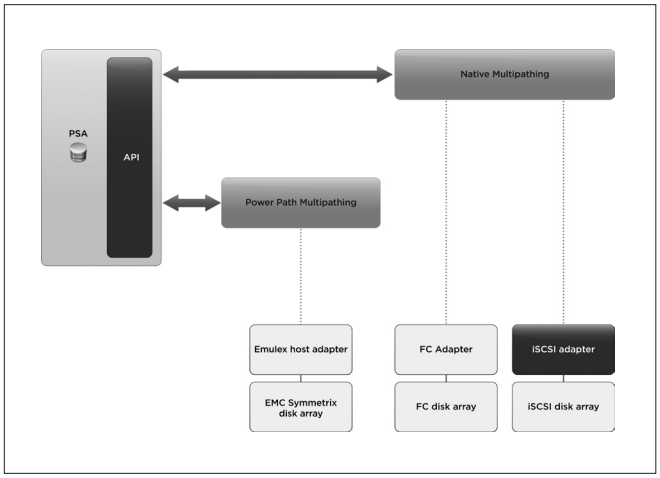

Figure 3. Example of multiple multipathing modules in PSA.

4.2 Testing

Using the test approach described in Section 2, testing storage multipathing vmkAPIs includes kernel-level unit testing and API functional testing in a white-box testing environment with error scenarios. Figure 4 shows the high-level architecture of the PSA test suite, including a pseudo multipathing plugin module for testing, a software-based storage simulator, and user space scripts.

The test multipathing plugin, entirely built on the vmkAPIs provided by PSA, was loaded and operated simultaneously with the NMP multipathing module in the VMkernel. The software storage simulator created a Fibre Channel test device with a multipath ramdisk as a data container for I/O.

The PSA is configured to prioritize the presentation of a path to the multipathing plugin based on the received vendor, model, and version information of the device[6]. When the vendor, model, and version supplied are from the predefined test device, the PSA first checks with the test multipathing module to see if it wants to claim the path(s). The test multipathing module claims the paths and the test device. The PSA, the test multipathing plugin, together with the test device, form a live storage subsystem for testing.

The kernel test modules and user scripts in Figure 4 generate test cases to the testing system, from kernel and user space, respectively. The kernel test modules create test cases to exercise one or more of the vmkAPIs that are impossible to test alone without a working test system. For instance, testing the vmkAPI that removes a disconnected path from a given storage device requires a device with a disconnected path existing in the kernel space at the time of testing. With the pseudo multipathing plugin and the simulated storage device, it is easy to create such prerequisite conditions to test this API. On the other hand, the user space scripts create test cases to exercise the test system and the vmkAPIs from an "end-to-end" angle, which is close to how vmkAPIs are (indirectly) used by end users.

The following is an example of such a user space test case. After adding a new Fibre Channel LUN to vSphere ESX, users can use the command line to change the device default claiming multipathing plugin to a desired plugin. The user space command is interpreted through system calls and eventually reaches the multipathing vmkAPIs that matter. A successful return of the user command relies on these vmkAPIs to function properly. The test plugin is instrumented with checkpoints that verify the result at different stages along the code path, which is helpful for tracking and debugging.

With the help of the software-based storage simulator, the APIs can be tested under error conditions. For example, as mentioned in Section 3.1.4, when the ioctl command for path disconnection is set on a particular path of the simulated device, no I/O for that path is processed by the storage driver. A VMK_HOST_NO_CONNECT status is returned, indicating that the path is disconnected. VMware has tested the path failover fault tolerance feature in PSA and its multipathing module using such simulation, by injecting path disconnection on the current active and preferred path of the device. Consequently, path failover handling is invoked, since the device has multiple paths and at least one path still is connected.

If path disconnection is enforced on all paths of that device, a All Paths Down (APD) or Permanent Device Loss (PDL) scenario is created. All Paths Down is the condition where all paths to the storage device are lost or the storage device is removed. In the real world, the APD condition can be caused by changes in SAN configuration or due to accidental outages in the SAN. This type of APD is a transient APD, as the disconnected path eventually recovers. There is one special case caused by removing a storage device. This condition is called Permanent Device Loss. Testing the vmkAPIs under error conditions such as APD and PDL is possible only using functional verification testing in a system in which the error scenario is present.

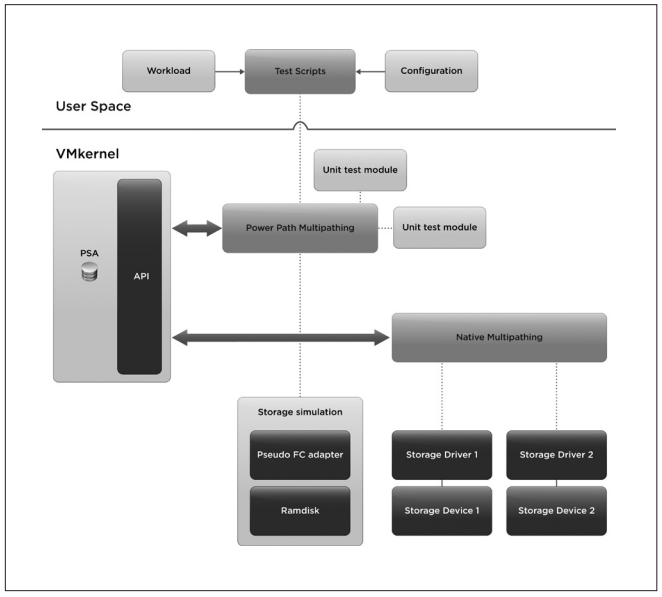

Figure 4. PSA vmkAPI Testing Architecture.

4.3 Approach Comparison

This section compares the white-box testing approach versus black-box testing approach and unit testing on vmkAPI test coverage from several aspect (Table 1). The white-box testing tests vmkAPIs within a test (sub)system. The black-box testing tests vmkAPIs by best effort, invoking the vmkAPIs from a user space workload on the native VMkernel. Unit testing is kernel-level functional verification isolated to an individual vmkAPI, validating various combinations of input/output pairs.

The white-box testing approach provides comprehensive API coverage. Using the white-box testing approach, the PSA vmkAPI test suite is able to achieve satisfactory API coverage. Since this is the first step in vmkAPI testing, there is no previous API coverage data with which to compare results.

	WHITE-BOX TESTING	BLACK-BOX TESTING	UNIT TESTING
Unit test coverage	Possible for all vmkAPIs	Nondeterministic kernel invocation	Limited to vmkAPIs without system dependency
Functional verification	Deterministic at kernel level and user space	Nondeterministic kernel invocation from user space	Limited to vmkAPIs without system dependency
Error handling coverage	Yes and fully automated	Limited coverage with manual involvement	Limited to vmkAPIs without system dependency
Real-world scenario testing	Yes for both kernel and user space scenarios	User space scenarios only	Very limited since most scenarios have system dependency
Compatibility testing	Yes, with the test modules built on vmkAPIs with the same compatibility restriction	Not applicable without kernel modules specially written for testing	Yes, with the test modules built on vmkAPIs with the same compatibility restriction

Table 1. vmkAPI Coverage for Three Test Approaches.

This section concludes with a comparison between using software simulation versus using real hardware on hardware savings, based on two storage multipathing test cases. The first is a basic storage path failover test case in vSphere ESX. The test requires a vSphere ESX host and a storage device with at least two paths that are claimed and managed by the PSA. Initially, both paths are active, and one path is the preferred path for I/O. During I/O, the preferred path failed and path failover was initiated. The standby path became the preferred path. To the vmkAPIs being tested and path failover handling, the hardware-based approach and software-based approach are equivalent. They were compared by minimum hardware requirement, test process, and side impact, as shown in Table 2.

	HARDWARE APPROACH	SOFTWARE APPROACH
Minimum hardware	1 vSphere ESX host 2 HBA ports 1 switch 1 storage array	1 vSphere ESX host
Test process	Run I/O workloads; Disable the port; Re-enable the port.	Run I/O workloads; Inject error to path; Unset the error.
Side impact	All paths associated with the port are down when the port is disabled	None

Table 2. Testing Path Failover: Hardware Approach versus Software Approach.

The second test case is based on a customer reported issue. Rescanning the host adapters took significantly longer when some storage devices were recovering from an All Paths Down (APD) condition, compared to the rescanning time before the APD event. This issue was happening on a host with a relatively large number of different types of storage adapters. A random number of them experienced an APD event. The hardware-based and software-based approaches were compared by minimum hardware requirement, test process, and side impact, as shown in Table 3.

	HARDWARE APPROACH	SOFTWARE APPROACH
Minimum hardware	1 vSphere ESX host dozens of HBA ports multiple switches dozens of FC and iSCSI LUNs	1 vSphere ESX host
Test process	Time rescan adapters; Run I/O workloads; Disable a random number of ports; Time rescan adapters; Re-enable the ports.	Time rescan adapters; Run I/O workloads; Inject error to paths; Time rescan adapters; Unset the error.
Side impact	Need a host with a big number of FC and iSCSI LUNs; All paths associated with the port are down when the port is disabled.	None

Table 3. Testing rescanning host adapters under APD condition: Hardware Approach versus Software Approach.

The comparisons in Table 3 show that the software-based approach has an obvious advantage in hardware consumption, with zero additional cost as the number of storage devices scales. The software-based approach provides particular flexibility and convenience to testing the PSA that does not rely on hardware to work properly. In addition, in this example, the pseudo multipathing plugin, software simulated storage device, and the PSA, become the white-box that enabled unit testing and API functional verification. This provides comprehensive API coverage that cannot be achieved simply by using unit testing and black-box functional testing.

5. Future Work

Testing of an API is not complete without testing it in scenarios where the API actually is used. The pseudo plugins and drivers in this paper are used to build such a white-box testing environment, where different use scenarios can be simulated and tested. Many of the use scenarios tested are synthetic, with only a handful from real customer use cases. A real-world usage scenario is valuable because it tells how the API is being consumed, and sometimes helps discover hidden bugs. It is a good resource for testing and regression testing. VMware plans to continue to improve test coverage by collecting more customer use cases and building them into the test suite.

One major advantage of using white-box over black-box testing is that test developers can design targeted test cases to exercise specific code path(s), which is only possible when test developers have knowledge of the internal implementation of the product being tested. VMware is working on enhancing the code coverage tool, which can provide a quantitative measurement of path coverage for a given test case.

The software-based storage simulation potentially can be considered in other places, in addition to API testing. For instance, it can be used in a setup for proof of concept, where storage devices are required but there are hardware resource limitations, or driver-level code instrumentation is needed to test features in the upper level of the kernel stack. VMware plans to continue to improve the storage simulation to support more adapter and disk types. Consideration is being given to extending the backend by introducing a dedicated error injection framework with enhanced features, for example, injecting specific errors on demand for a certain period of time, or for a certain number of I/O operations. A frontend GUI might be provided to configure and manage the simulator.

6. Conclusion

Static unit testing and functional black-box testing commonly are used to test APIs. Neither of these methods provides a sufficient level of verification. This paper described a new approach to testing vSphere VMkernel vmkAPIs. This approach uses kernel-level unit testing combined with API functional verification in a white-box testing environment. A kernel test (sub)system consisting of pseudo plugins and drivers and error condition simulation is used to assist testing. As part of this white-box testing approach, the software storage simulation provides an effective way to test error handling for some storage vmkAPIs and related kernel storage stacks that are otherwise difficult to test. This approach is used in the Pluggable Storage Architecture vmkAPI testing, and provides satisfactory API coverage. The software storage simulator also can be used to simplify test setup under certain conditions, making it a potential supplement to some typical hardware-based test environments.

7. Acknowledgments

The authors would like to thank our shepherd Keith Farkas for his patience and valuable feedback. We thank Tom Phelan and Ravi Soundararajan for guiding our work, Eric Lorimer for implementing thin-provisioned ramdisk and nulldisk, and Manisha Banerjee for testing the All-Paths-Down storage feature. Our thanks to everyone in the VMkernel API test development team, as well as Mike Zucca, Kapil Chowksey, Samdeep Nayak, Komal Desai, Pushkin Reddy, and Gururaja Hegdal for their contributions.

8. References

[1] VMware Co-Development Programs. http://www.vmware.com/partners/programs/alliances/co-dev

[2] VMware vSphere. http://www.vmware.com/products/vsphere/overview.html

[3] vSphere ESX. http://www.vmware.com/products/vsphere/esxi-and-esx/overview.html

[4] White-box Testing. http://en.wikipedia.org/wiki/White-box_testing

[5] Charu Chaubal. The Architecture of VMware ESXi. White Paper, 2008

[6] Thomas A. Phelan, Olivier Lecomte. Multiple multipathing software modules on a computer system. U.S. Patent 7\,831\,761 B2, 2010

Providing Efficient and Seamless Desktop Services in Ubiquitous Computing Environments

Lizhu Zhang
VMware 17th Floor, Tower C,
Raycom Info Tech Park
No 2, Kexueyuan South Road
Beijing 100190, China
lzzhang@vmware.com

Wenlong Shao
VMware 17th Floor, Tower C,
Raycom Info Tech Park
No 2, Kexueyuan South Road
Beijing 100190, China
wshao@vmware.com

Jim Grandy
VMware
3401 Hillview Avenue
Palo Alto CA94304, U.S.
jgrandy@vmware.com

Abstract

As smart phones and tablets gain in popularity, VMware® desktop virtualization technology makes it possible to access virtual desktop services from any location and any device. VMware currently provides the VMware View™ Client on most smart devices, such as Android tablets and phones, and iPads. Using VMware View Client on these devices, end users can access personal desktops on PCs, cellular phones, or tablets through any Ethernet, WiFi, or 3G network. To make the user experience just as good as direct desktop access, the VMware View Client is optimized for embedded devices. These enhancements include code optimization, better user interaction, and efficient usage of available network bandwidth.

This paper details the code optimization in the image processing SIMD technology on embedded devices based on ARM NEON technology. The number of NEON registers on the ARM CPU is limited, and a special pipeline results in different instructions possibly consuming different amounts of CPU cycles. As a result, it is important to use a good register scheduler to ensure overall CPU optimization. The optimized implementation described here improved performance by 2x when compared to the original C code, and by 75 percent compared to the ARM NEON intrinsics. In addition, it presents a translator from "touch screen actions" to "desktop actions", as well as how to tune parameters, such as sensitivity and waiting time, with a VMware internal testing program. Based on user feedback, options are kept as simple as possible while meeting most user requirements. Finally, the PCoIP network adaptive protocol was used for desktop virtualization to provide good performance in different network environments. Special rules were configured for the mobile devices to save network data for users. These enhancements solve the performance, interaction, and network issues on mobile devices to deliver a seamless and efficient desktop experience in a ubiquitous computing environment.

Categories and Subject Descriptors

J.7 [Computer Applications]: COMPUTERS IN OTHER SYSTEMS — *Consumer products*.

General Terms

Performance, Experimentation, Human Factors

Keywords

Desktop Virtualization, SIMD Optimization, Interaction

1. Introduction

Portable devices, such as smart phones and tablets, are changing the way people interact with information. While these devices enable users to access information anywhere, limited computing ability and fewer supported applications make it difficult for users to move away from traditional desktops. VMware desktop virtualization technology[1] provides a way to deploy desktops as a service from a concrete datacenter to distributed clients via VMware View[2]. VMware View supports PCs running Microsoft Windows, Linux and Mac OS X, as well as most smart devices, such as Android tablets and phones, and iPads. Users can access to desktops on PCs, cell phones, or tablets from anywhere over Ethernet, WiFi, or 3G networks.

Three factors are key to delivering a good experience to users accessing desktops from portable devices.

- The desktop must be displayed clearly and seamlessly. An unclear image or time delay of a remote desktop negatively impacts the user experience.
- Actions taken on portable devices must be translated to desktop actions naturally, so that users can perform tasks easily.
- The solution must work in different network environments. Portable devices always work in wireless environments even when the user is mobile. The user experience must remain stable when network conditions change.

The work described in this paper takes a step towards enhancing the user experience. To achieve these goals, PCoIP[3] was chosen as the display protocol and tuned specifically for ubiquitous computing with portable devices. Key contributions from this paper include:

- A proposed pipeline-based SIMD[4] schedule model to optimize the image processing code with ARM NEON[5] technology.
- Based on the optimized display protocol, a set of gestures was designed for users to interact with the remote desktop. The software was tuned with a VMware internal test program to improve the user experience.
- PCoIP can adjust image quality adaptively based on the network environment. Special rules were added for portable devices to save network data bandwidth for users.

The remainder of this paper is organized as follows. Section 2 provides an overview of the system. Section 3 details ARM NEON optimization and compares the result with two baseline methods. Section 4 presents the interaction system for touch screen devices. Section 5 introduces the PCoIP network adaptive feature and special rules for mobile clients. Finally, Section 6 summarizes the results observed and presents ideas for the future.

2. System Overview

This section provides an introduction to VMware View and describes possible scenarios for its use in ubiquitous computing environments. In addition, the technologies and designs used to improve the user experience are described briefly.

2.1 Preliminary

VMware View delivers rich, personalized, virtual desktops as a managed service from a virtualization platform built to deliver the entire desktop, including the operating system, applications, and data. As shown in Figure 1, users can access a centralized virtual desktop from a client after authentication with the VMware View Manager.

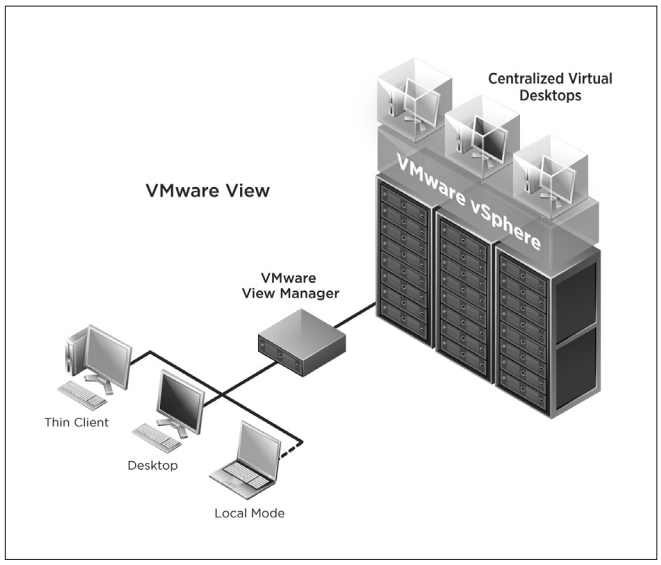

Figure 1. VMware View Architecture

There are many benefits of using VMware View in the enterprise[2]. It allows users to use portable devices to connect to desktops and create an ubiquitous computing environment. There are many scenarios for accessing desktops from portable devices. For example, users that are traveling can deal with tasks on an office system, and then access the desktop through a smart phone when remote.

The overall architecture of the VMware View Client on portable devices is shown in Figure 2. The VMware View agent, which contains the PCoIP server, is installed on the Windows operating system in a virtual machine in the datacenter. The PCoIP server redirects the desktop image data over the network. The PCoIP client receives the data from network, decodes the desktop image, and draws it on the device screen. Touch screen and on-screen keyboard events are translated to mouse and keyboard events and sent to the desktop over PCoIP. Other data, such as audio, also are transferred over the PCoIP protocol.

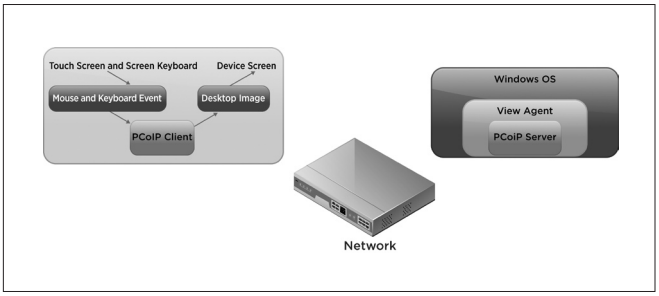

Figure 2. Architecture of the VMware View Client on Portable Devices

2.2 Technologies

Figure 3 shows the technologies used to deliver desktops in an ubiquitous computing environment. SIMD technology is used to improve the image processing performance on portable devices. The gesture-based interaction system supplies a way to control the desktop with touch screen operations. The network adaptive transfer protocol stabilizes client performance in different network environments.

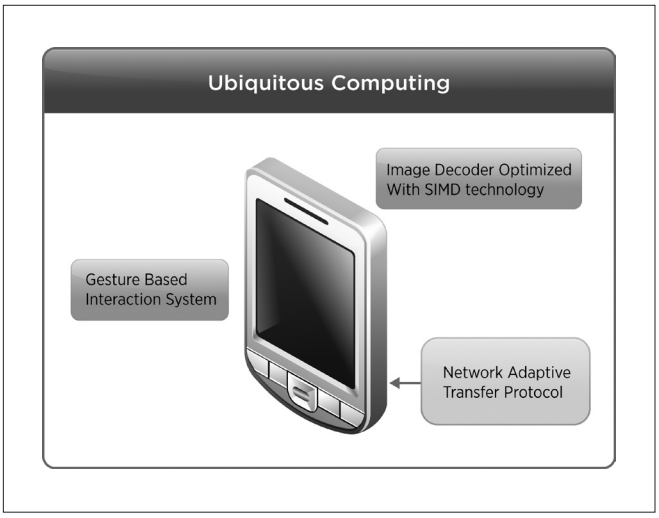

Figure 3. Technologies for delivering desktop in ubiquitous computing

3. Optimizing The Image Decoder

This section describes the use of ARM NEON technology to optimize the image decoder in the PCoIP client. It includes a brief introduction to ARM NEON technology and PCoIP protocol features. Next, the ARM NEON technology-based optimization model is described, including how to find hot spots in applications and verify optimization results. Finally, the optimized application's performance is compared to two baseline methods.

3.1 ARM NEON and PCoIP Basic Concepts

ARM NEON5 is general-purpose SIMD engine that accelerates multimedia and signal processing algorithms. It is widely used in multiple media and signal processing algorithm[6 7] optimization. On the ARM-Cotex A8 CPU8, the components of the NEON co-processor include:

- NEON register file with 32x64-bit general-purpose registers
- NEON integer execution pipeline (ALU, Shift, MAC)
- NEON dual- and single-precision floating-point execution pipelines (FADD, FMUL)
- NEON load/store and permute pipeline

As shown in Figure 4, the 64-bit registers of the NEON co-processor, named D register, can be packed in 8, 16, 32, or 64 bits. Two consecutive 64-bit registers can be combined into a 128-bit register, called the Q register. As a result, a maximum of 128-bit data can be processed for a single instruction.

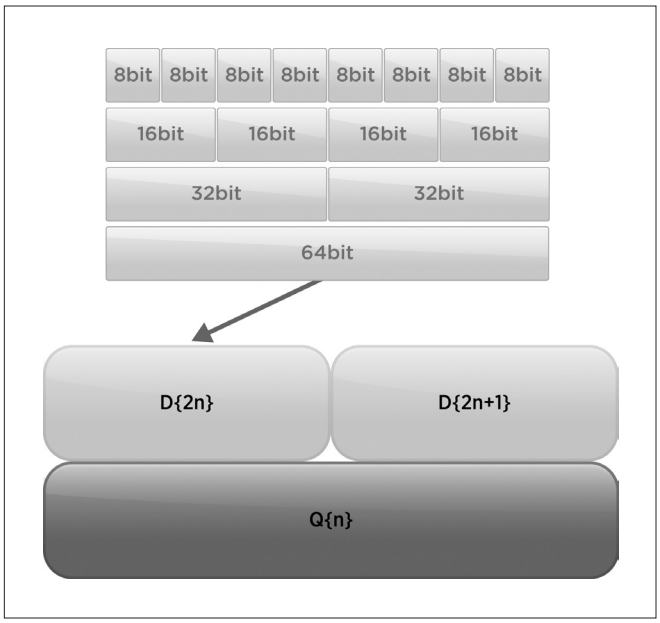

Figure 4. NEON Registers and Data Pack

The PCoIP protocol provides the adaptive display compression for connecting desktops over existing, standard IP networks[3]. The PCoIP protocol compresses, encrypts, and encodes the entire computing experience in the virtual desktop, and delivers the image to the client. The client decodes, decrypts, and decompresses the desktop image and renders it on the client's host device. Similar to other protocols, PCoIP uses the following technologies to improve image quality and compression rate.

- Detects persistent image sections and enables a sequenced build of the image dependent on human perception preferences[9].
- Offers an adaptive progressive encoding sequencer with selectable quality levels optimized for the encoding of computer display images[9].
- Provides real-time decomposition of text, objects, background, and pictures at a pixel level that is critical for the perception-free compression and reproduction of a computer display image[10].
- Decomposes images based on expected digital computer display image artifacts, resulting in optimized decomposition of the image[10].

These features complicate the PCoIP image decoder and consume more computing resources.

3.2 NEON-based Optimization Model

3.2.1 A generic example of optimization with SIMD

Using SIMD technology in a signal processing application is a very common idea. For example, the native C code shown in Table 1 adds two 64-dimensioned, 8-bit vectors (A[64], B[64]).

TWO 64-DIMENSION VECTORS ADD OPERATION

for (int i = 0; i < 63; i ++)
{
  A[i] += B[i];
}

Table 1. Native C code without SIMD optimization

Because the operations on A[i] and B[i] are the same, and assuming 128-bit SIMD registers (R0, R1, ..., R15) are available, the add operator *vadd* and load operator *vload* are used. The pseudo code with SIMD technology for the code in Table 1 results in the code shown in Table 2.

If the CPU cycles for *load, add, vload*, and *vadd* are all 1, the code in Table 2 needs 12 CPU cycles (8 load cycles and 4 add cycles), while the code in Table 1 needs 192 CPU cycles (128 load cycles and 64 add cycles). The SIMD optimized code saves 180 CPU cycles of 192, or 93.7 percent.

TWO 64-DIMENSION VECTORS ADD OPERATION	
vload R0, A[0]; vload R1, A[16]; vload R2, A[32]; vload R3, A[48]; vload R4, B[0]; vload R5, B[16]; vload R6, B[32]; vload R7, B[48];	vadd R0, R0, R4; vadd R1, R1, R5; vadd R2, R2, R6; vadd R3, R3, R7;

Table 2. Pseudo code with SIMD optimization

3.2.2 Model Description

Unfortunately for NEON technology, different instructions may consume different CPU cycles. For example, VMUL for Qd, Qn, Qm (.32 normal) requires four CPU cycles, while VADD for Qd, Qn, Qm only requires one CPU cycle[8]. As a result, the instruction sequence listed in Table 3 consumes eight CPU cycles instead of five CPU cycles. The code on line 2 must wait 3 CPU cycles for the code on line 1's result.

As described in section 3.1, "ARM NEON and PCoIP Basic Concepts," NEON instructions execute in a pipeline. If the code from lines 2 to 5, which does not depend on the Q0 register, is placed between line 1 and line 2, the code segment only consumes five CPU cycles (see Table 4). Three lines can be executed when the multiply operation is in process.

A MULTIPLE ACTION FOLLOWED BY ADD ACTIONS
VMUL.s32 Q0, Q1, Q2; — Line 1 – 4th CPU cycle
VADD.s32 Q4, Q0, Q3; — Line 2 – 5th CPU cycle
VADD.s32 Q5, Q5, Q6; — Line 3 – 6th CPU cycle
VADD.s32 Q7, Q7, Q8; — Line 4 – 7th CPU cycle
VADD.s32 Q9, Q9, Q10; — Line 5 – 8th CPU cycle

Table 3. Non-optimized code for VMUL followed by VADD

A MULTIPLE ACTION FOLLOWED BY ADD ACTIONS
VMUL.s32 Q0, Q1, Q2; — 1st CPU cycle
VADD.s32 Q5, Q5, Q6; — 2nd CPU cycle
VADD.s32 Q7, Q7, Q8; — 3rd CPU cycle
VADD.s32 Q9, Q9, Q10; — 4th CPU cycle
VADD.s32 Q4, Q0, Q3; — 5th CPU cycle

Table 4. Instruction Pipeline Optimized code for VMUL followed by VADD

ARMv7 has 16x128-bit or 32x64bit registers. The NEON instruction that consumes the most CPU cycles, *VLD4 4-reg (unaligned, @64)*[8], requires five CPU cycles. Since this is smaller than the number of registers, it is possible to intersect the instructions similar to the case in Table 3 to eliminate need for the CPU to wait. To achieve this goal, the following rules should be followed:

- Make the input data unit large enough for parallel instructions. If the input data is only 128-bit, the code intersecting operation cannot be performed.
- Use the output register of a multiple CPU cycles operator only after the result is ready.

With the rules above, functions can be optimized manually with higher performance than a typical compiler's optimization.

3.2.3 PCoIP Optimization Techniques

Intrinsically, all operations performed on contiguous data can be optimized with SIMD. For example, all memset functions act on a memory block. To improve application performance significantly, a focus on the functions consuming the most CPU cycles, or "hot spots", is required. Shark[11], a profiler for Mac OS X, is an application that helps optimize software for Mac OS X or iOS, and can be used to find the hot spots in applications.

3.2.3.1 Finding Hot Spots in Application Code

The project began by making special videos for different cases of the actions taken on the desktops, such as video that included rapid text switching, and images with high refresh rates. These videos make the image decoder work at full capacity. The video was played on the remote desktop in full screen mode. The used Shark applications was used to make a sample for the VMware View Client on an iPad. Finally, CPU call stacks and function usage were generated, with the top functions identified as hot spots. Figure 5 shows a sample of the CPU call stacks and function profile of the VMware View Client on an iPad.

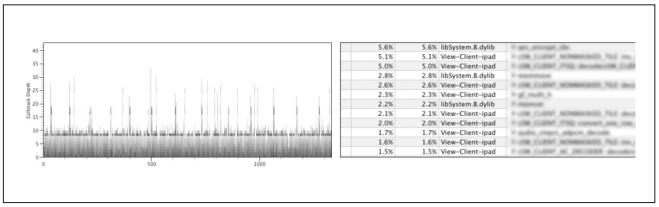

Figure 5. CPU call stacks and function usage generated by the Shark application

3.2.3.2 Using Inline Assembly to Optimize Functions

The GNU C compiler for ARM RISC processors offers a way to embed assembly language code into C programs. This feature can be used for manual optimization of time-critical sections of the software, or to use specific processor instructions that are not available in the C language[12]. Inline assembly was used to optimize the hot spots line by line, following the model in section 3.2.2. The optimization changes the original code logic to ensure no CPU cycles are wasted for NEON instructions. As a result, it is impossible to make the optimized function work correctly at one time. It is very important to have ways to verify the optimized result and easily locate code errors.

3.2.3.3 Verifying Optimization Results

Three steps were developed to ensure the result was correct, as shown in Figure 6.

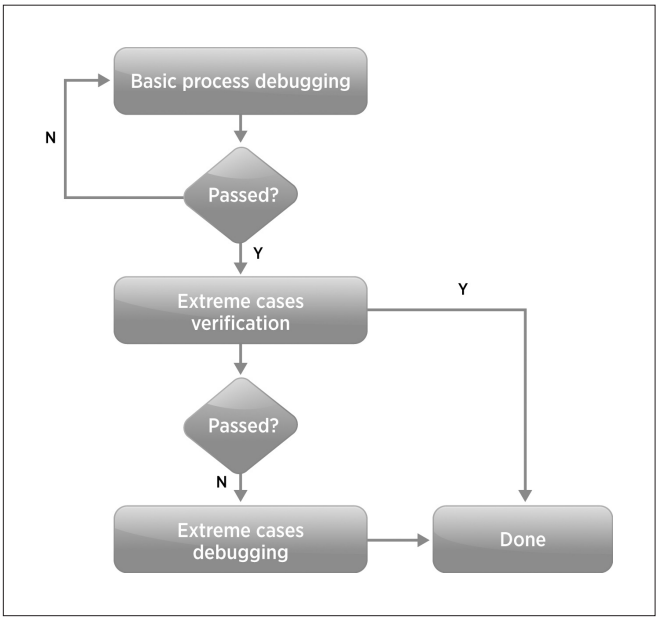

Figure 6. Optimization result verification process

- Step 1: The basic debugging process used featured input vectors, such as a vector of all 0s, all 1s, and arithmetic progression with a common difference of 1, starting from 1, as the input data for the

functions before and after the optimization. Output results were compared value by value. Using obvious input data can make it easier to locate logic errors by comparing the output before and after the optimization.

- Step 2: Because the test cases in step 1 did not cover all function branches or extreme cases such as value overflow, featured video sequences were made that contained pixels with extreme values and covered all of the branches in the image decoder. The decoding results from the original C application, and the optimized application, were written to files and compared byte by byte. If no difference was observed between the two files, the optimized function was correct and the optimization job was done. Otherwise, step 3 was used to locate the extreme case error.
- Step 3: The extreme cases identified in step 2 were debugged as follows. The original and optimized functions were placed in a single application. A copy was made of the input data to the function. The optimized function and the original C function were called at the same time, and the output of the two functions was compared.

Using these steps, the top eight hot spots that consumed nearly 40 percent of the CPU before optimization were optimized to use only 10 percent of CPU resources.

3.3 Experiments for Image Decoder Optimization

Image decoders are evaluated based on the decoding speed in MPixels per second. Using this metric, the effectiveness of the codec and code optimizations can be evaluated. This section explores the effectiveness of two baseline applications, the *original C code application and the NEON Intrinsics[13] optimized application*. The original C application does not use SIMD technology to process data. In the NEON Intrinsics application, SIMD technology is used for data processing, but the compiler schedules the NEON registers. All three implementations are compiled using GCC[14] 4.2 with the level 2 optimization option. Different devices were used to test the performance of the application, including: SAMSUNG Galaxy Tab (1 GHz CPU, Cortex A8), iPad 1 (Apple A4 1 GHz CPU, Cortex A8) and iPad 2 (Apple A5 dual-core, 1 GHz CPU, Cortex A9).

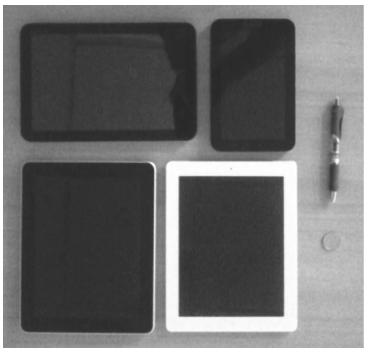

Figure 7. Devices for the performance test

Table 5 lists the decoder speed of the three applications. Comparing the result to the original C application, the performance is doubled. Compared to the NEON Intrinsics application, there is approximately a 75 percent performance improvement.

	TEST IMPLEMENTATION (MPIXELS/SEC)	NEON INTRINSICS (MPIXELS/SEC)	ORIGINAL C (MPIXELS/SEC)
SAMSUNG Galaxy Tab	1.32	0.75	0.64
iPad 1	1.87	1.13	0.93
iPad 2	4.09	2.46	1.99

Table 5. Decoder speed in MPixels/Sec of different implementations (higher is better)

4. Interaction System Design

This section details the goal of the interaction system and how it was tuned to improve the user experience.

4.1 Basic Design

Users interact with tablets and smart phones via a touch screen or on-screen keyboard. VMware View Client is used to connect to PC desktops, which use a traditional hardware keyboard and mouse as the input method. There are two main differences between touch screen input and mouse input. With a mouse it is easy to fire clicks with different buttons. A touch screen has no buttons. In addition, a mouse cursor is more accurate than fingers on a touch screen. To give users a better experience, it is important to make frequently used actions on the desktop easier to fire on a handheld device. This results in the touch screen to desktop translation map shown in Figure 8.

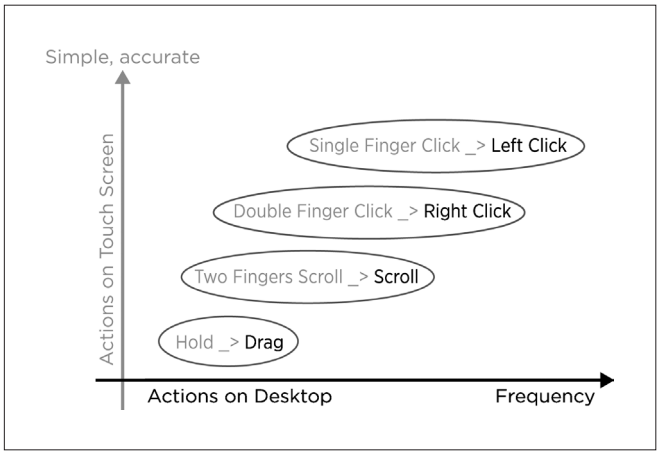

Figure 8. Action translation map

Single finger clicks on the touch screen send a mouse left-click event to the remote desktop. A double finger click sends a mouse right-click event. The position is the first finger's position. When two fingers are used to scroll on the screen, a mouse scroll event is sent. Holding a position with one finger for a period of time sends a mouse left-button hold and drag event.

In addition to the basic actions, more accessories were added to improve the user experience, as shown in Figure 9. For example, local gestures such as pinch in/out are used to zoom the screen size locally, while a long press is used to call out the loupe to magnify part of the desktop image. Local items, such as a drag tip, is used

to help with accurate drag. A soft touchpad gives the user a way to use a touchpad, while a functional key pad helps with pressing function keys.

Figure 9. Accessories to improve user experience

4.2 Tuning the Interaction System

Some operations are related to variable parameters, such as the time interval of finger press to show a drag indicator, the time interval of finger holding to show a loupe, the sensitivity of the touchpad or two finger scroll, and more. Having too many configuration options for users can be inefficient. The solution described here eliminates as many options as possible to make the application simple and easy to use.

During VMware internal test program execution, four options were shown to users:

- The time interval for a drag indicator, which determines how long a user holds a finger on the screen to initiate the drag action on the remote desktop.
- The time interval between a drag indicator and loupe, which determines how long a user holds a finger on the screen to show the loupe. The total value is equal to the time interval for the drag indicator plus the time interval between the drag indicator and loupe.
- The touchpad sensitivity, the coefficient in the touchpad track formula.
- Two fingers scroll sensitivity, the coefficient in the scroll distance formula.

After a user launches the remote desktop, the four parameters can be configured to customize the application. A button sends the configuration file to the developer to get the statistics result. Hundreds of users from different departments of the company, including marketing, product management, developers, and testers, joined the internal test program.

4.3 Internal Test Result

The statistics for the internal test program are shown in Figure 10. The results show that 85 percent of users chose to configure the time interval for a drag indicator to 0.1 to 0.2 seconds; 81 percent of users set the time interval between a drag indicator and loupe between 1.5 and 2.5 seconds; and 78 percent of users set the two fingers scroll sensitivity, which has a positive correlation for the distance of scrolling action, between 5 and 15. Because the values are concentrated, the solution sets the time interval for a drag indicator to 0.15 seconds, the time interval between a drag indicator and loupe to 2 seconds, and the two fingers scroll sensitivity to 10. No single value can satisfy every user's preference for touchpad sensitivity (the maximum speed of the pointer). As a result, the solution makes touchpad sensitivity an option, and sets the default value to 3.

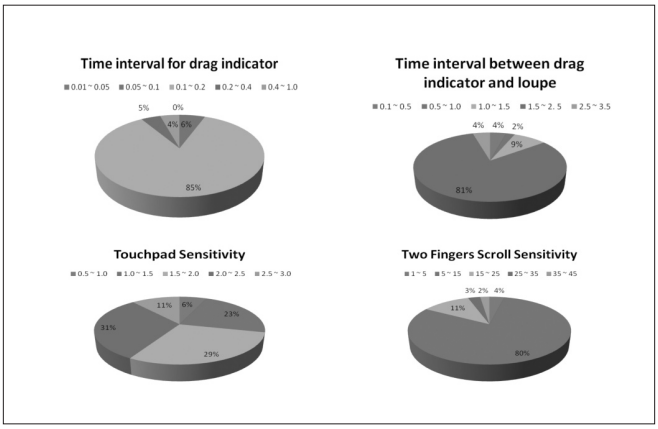

Figure 10. Parameters statistics result in Internal Test Program

5. Network Policy

This section describes the network policy for the PCoIP protocol and VMware View Client on mobile platforms. In general, PCoIP adapts to network conditions. For mobile clients, network status and network cost should be considered.

The PCoIP protocol adapts dynamically to network conditions. Its adaptive encoders automatically adjust image quality on congested networks then resume maximum image quality when the network no longer is congested.[3] In different network environments, such as WLAN, WAN, or 3G networks, the desktop can be displayed seamlessly with different image qualities. At the same time, 3G network usage is charged by network flow, and is much more expensive than a WiFi connection. Consequently, the following rule was established for mobile clients. If a user connects over WiFi and the application is in the background, the session is maintained for 10 minutes. If the user connects over a 3G network and the application is in the background, the session is closed and reconnected to the same desktop once the application is moved into foreground.

6. Conclusion

This paper provides an overview of the work performed to use VMware View Client on mobile platforms. Based on the ARM NEON CPU, a model was created to optimize the image decoder to improve mobile clients performance. Considering the difference between touch screen and PC operation, an interaction system was designed and optimized for VMware View Client for mobile platforms. Based on the network connection type, different behaviors were defined for the application when it moves to the background. As a result, VMware View Client offers users a way to access desktops in ubiquitous computing environment with good performance and a positive user experience.

Future enhancements include the addition of cache for the image decoder to improve the performance and save the network bandwidth. In addition, more features can be added as smart phone and tablet technologies progress.

Acknowledgements

The authors would like to thank Yueting Zhang, Kun Shi from Wenlong Shao's team, in helping out with the development work on different devices.

7. References

[1] VMware Desktop & End-User Computing Solutions: http://www.vmware.com/solutions/desktop/

[2] VMware View: http://www.vmware.com/products/view/overview.html

[3] PCoIP technology: http://www.teradici.com/pcoip/pcoip-technology.php

[4] SIMD: http://en.wikipedia.org/wiki/SIMD

[5] ARM NEON: http://www.arm.com/products/processors/technologies/neon.php

[6] Chirag, P., Anurag M., and Sandeep G,. 2009. H.264 Video Decoder Optimization on ARM Cortex-A8 with NEON. India Conference (INDICON), 2009 Annual IEEE. Samsung India Software Operations Pvt. Ltd.

[7] Tero R,. 2009. Optimizing H.264 Decoder for Cortex-A8 with ARM NEON OpenMax DL Implementation. Technology In-Depth. On2 Technologies.

[8] Cortex — A8 Technical Reference Manual Revision: r3p2. ARM Limited.

[9] David, V. H., Patrick, R., 2010. Methods and apparatus for encoding a digital video signal. US Patent US7822278. Teradici Corporation.

[10] David, V.H., Kimberly, M.T., 2010. Method and apparatus for generating masks for a multi-layer image decomposition. US Patent US7782339. Teradici Corporation.

[11] http://en.wikipedia.org/wiki/Apple_Developer_Tools#Shark

[12] http://www.ethernut.de/en/documents/arm-inline-asm.html

[13] http://gcc.gnu.org/onlinedocs/gcc/ARM-NEON-Intrinsics.html

[14] http://gcc.gnu.org/

Comprehensive User Experience Monitoring

Lawrence Spracklen
lspracklen@vmware.com

Banit Agrawal
banit@vmware.com

Rishi Bidarkar
rishi@vmware.com

Hari Sivaraman
hsivaraman@vmware.com

R&D Performance, VMware, Inc.

Abstract

In today's networked world, users increasingly consume multimedia content that is streamed in real time to client devices. This content can be comprised of a user's entire desktop (as is the case with Virtual Desktop Infrastructure (VDI) and desktop as a service (DaaS) solutions), specific applications (such as software as a service (SaaS), application publishing, and online gaming) or video and audio streaming. This paradigm shift is evidenced by the significant emphasis that VMware is placing on this market with its internal application development and recent acquisitions.

In all these scenarios, across different client devices and networks, users demand a near-flawless, rich, multimedia experience. To ensure this experience, audio and video quality and the synchronization between the streams (to ensure good "lip-sync"), must meet stringent criteria. In addition, a user's experience is influenced highly by the responsiveness of interactive operations, such as scrolling, panning, or window dragging (2D and 3D workloads). To date, tracking these metrics, especially at cloud scale, has been very imprecise.

This paper presents VMware techniques and tools to solve these problems and enable accurate, real-time, automated monitoring of the user multimedia experience. It highlights the potential of these tools by using them to analyze the performance of VDI solutions (including VMware View™) in a variety of network environments.

Categories and Subject Descriptors

D.m **[Miscellaneous]**: Virtual Machines, system management, performance monitoring.

General Terms

Performance, Management, Applications, Tools.

Keywords

End user computing, user experience, VDI, virtual desktop, DaaS, performance monitoring applications, video benchmarking, view planner, audio, interactive benchmark

1. Introduction

Today's networked economy is giving rise to a rapid transformation in the way users access multimedia content. In the corporate environment, users are moving away from local desktops and locally installed applications, to virtual desktop infrastructure (VDI) deployments and software-as-a-service (SaaS) models. In these environments, a user's desktop and software is hosted remotely, and the user interacts with them via a wide variety of client devices. In addition, there has been an explosion in consumer use of audio and video streaming sites. Voice over IP (VoIP) is becoming a mainstay, and smart phones and tablets are proliferating rapidly.

While these use cases span a wide application space, they all involve the streaming of multimedia content in real time. In previous years, network bandwidth limitations largely ensured that multimedia content had to be downloaded prior to viewing. Furthermore, streaming access to multimedia content is not constrained to LANs. WANs such as Wi-Fi and 3G are now sufficiently fast to support real-time streaming, albeit with additional challenges associated with network latencies, packet loss, and so on.

While there are obvious mobility advantages to these trends, users should not be constrained by the type of applications that can run in order for this access paradigm to become truly mainstream. For many users, high definition (HD) video, webcasts, VoIP, and video conferences represent the most demanding applications. Providers must ensure that sufficient computational and network resources can be made available to guarantee the user's multimedia experience is not compromised, or ensure that their solutions respond gracefully to resource bottlenecks and dynamically throttle data rates, while striving to maintain quality.

For these adaptive applications—including VDI, DaaS and even the more advanced video streaming sites—the client and remote host detect resource constraint problems and dynamically vary the level of compression applied to the streamed content to conform to the available bandwidth. As a result, there exists a real need to monitor the user experience in an accurate, repeatable, scalable, unobtrusive, and real-time manner. Such information is critical in a number of important situations.

- **Capacity planning.** Providers often undertake large-scale capacity planning exercises to characterize key workloads and understand the number of users that can be supported via various system architectures and for a wide variety of usage models. Without this data, providers lack confidence that they can meet user requirements and service-level agreements (SLAs) without the significant over-provisioning of hardware.

- **Protocol development.** In order to create more robust protocols that respond gracefully to resource constraints, developers require tools that help them accurately measure the response of their protocols to these events in the lab.
- **Experience monitoring.** While successful capacity planning should ensure users never encounter problems, in reality it can be beneficial to monitor the user experience in live deployments. The results of the analysis can be simply logged, used to alert users and administrators of performance problems, leveraged to adjust resource allocation dynamically and correct quality issues, or help aware applications to respond more gracefully to problems.

Unfortunately, accurately tracking these user experience metrics has been largely impossible to date. This paper discusses the techniques VMware has developed to allow the accurate and automated tracking of the key components of the user experience—video fidelity, audio fidelity audio-video (AV) synchronization (or lip-sync), and interactive workloads. The subsequent sections describe techniques to monitor AV content accurately, before discussing techniques to monitor interactive operations.

2. Video Benchmarking

While it is possible to monitor CPU and network utilization accurately, there is no precise way to measure video performance and quality on the client. Past approaches used incoming network data to correlate and analyze video quality and performance. This can be very inaccurate, as the server-side protocol could throttle the bit rate of the video in response to bandwidth limitations, and packets can correspond to player skins, window appearance, or other miscellaneous screen updates.

2.1 Challenges in Video Benchmarking

Since video metadata is not available when examining network packets, this problem needs to be attacked at a much higher level. One possible approach is to use a separate socket connection to inform the client about video metadata, such as frame number. Since the video frames are sent on a separate connection, packets could be lost or arrive out of order, and the latency measurement can be inaccurate. Another approach is to develop hooks into the video player, providing access to frame rate and quality information. In a remote environment, however, client video quality can be almost completely decoupled from that reported by the video player. Accordingly, an approach is required that provides an accurate way of measuring the latency and quality of remote and streaming video playback on the client. Additionally, it must be independent of the underlying streaming protocol, require no changes on the server side, and introduce no server-side perturbation.

To accurately measure the quality and latency of remote video playback, it is important to track the time at which each frame is displayed on the client accurately. To this end, this paper proposes a technique to watermark the video and encode the video metadata in each frame.

2.2 Video Watermarking Techniques

To accurately measure video quality, VMware developed a technique in which video frames are watermarked with their frame numbers. Since it is difficult to locate the watermarking data in the network layer (the packets have no notion of video frame), detection is performed at the display driver level, where all display updates are rendered. This corresponds to the time when the user actually sees the frames.

Using this technique, customized videos are created that have magic pixels in each frame followed by video meta-data, such as frame number. Since each frame contains the magic pixels and its own frame number, all that needs to be done is to locate the magic pixels in the frame buffer, read the current frame number, and record the time at which the frame was received.

While this watermarking works for lossless compression, where there is no modification of the values of the magic pixels and frame numbers, it does not function correctly with applications that leverage additional lossy compression to dynamically control bandwidth usage. Further, many remote display protocols also use caching techniques to send only those rectangles that have changed significantly. Accordingly, the technique needs to be robust enough to handle these situations.

2.2.1 Handling Lossy Compression

Due to the use of lossy compression, the pixel values observed on the client can differ from their original values. As a result, any metadata inserted into the video may be considered corrupt on the client. To overcome this problem, the solution leverages the observation that, even after lossy compression, changes in the pixel value tend to be quite small. The application must provide a similar look and feel on the client side.

In 24-bit RGB color format, each pixel is composed of three colors: red, green, and blue. Each occupies one byte, with a maximum value of 0xff. If 1 bit of metadata (value '0') is encoded in a 24-bit pixel value, 0x00 can be assigned to each byte. If, as a result of lossy compression, these bytes change to 0x3a, 0x15, 0x46, the appropriate thresholds (such as 0x80) can be used to mask the noise and recover the value '0'. Accordingly, the technique encodes the frame numbers by using the byte values 0xff and 0x00 to represent the frame number, and uses the color values of an entire pixel to encode each bit of frame number, as shown in Figure 1.

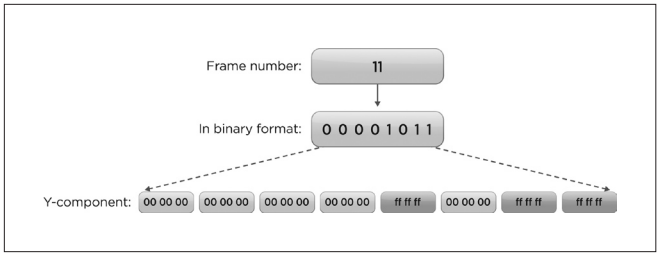

Figure 1. Frame number encoding for the video watermark, with every bit encoded using an entire pixel.

2.2.2 Bitmap Caching

Some protocols perform caching and quantization optimizations that discard very small changes. To address this problem, a different random macro-block (24x24 pixels in size) is embedded at the beginning of each frame, as shown in Figure 2. By customizing the video in this way, it is possible to ensure that the random macro-block, along with the magic pixels and frame number, are sent to the client for each frame.

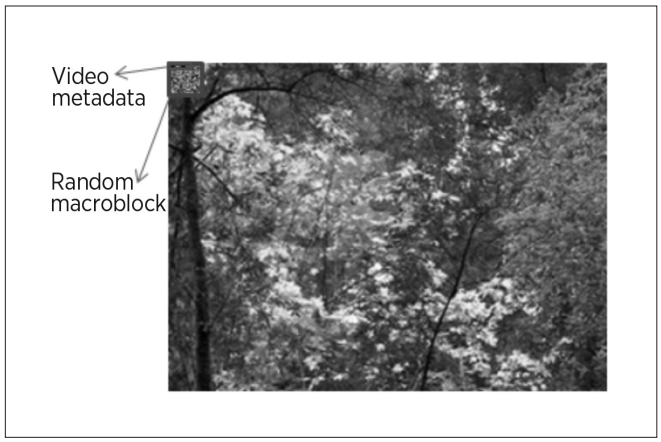

Figure 2. An example frame in a watermarked video.

2.3 Video Watermark Detection

Watermark detection consists of scanning each update for the magic pixels. The magic pixels only need to be located once, since their location does not change over the duration of the video. The following rules are used to locate the magic pixels:

(1) Nine consecutive bytes must obey the following:

(byte2, byte4, byte6) > upperThreshold **and**

(byte0, byte1, byte3, byte5, byte7, byte8) < lowerThreshold

(2) Additionally, the next 15 bytes must obey the following:

(byte[0-14] < lowerThreshold) **or**

(byte[0-14] > upperThreshold)

If these constraints are satisfied at a particular pixel location, the pixel is considered to represent the start of the magic pixels, and the location is used to find the frame numbers in all subsequent frames. The thresholds (0xa0 and 0x50 in this example) can be selected to control false positives. The frame number is recovered by scanning the next 15 bytes. If the pixel's bytes are more than 0x80, it is interpreted as '1', otherwise it is interpreted as '0'.

3. Audio Benchmarking

To accurately measure audio fidelity, it is necessary to examine the audio stream received by the client and ascertain how closely the received stream matches the original audio stream, and how a user would perceive the differences. What is the impact of host CPU constraints, network constraints, and the lossy compression utilized to control bandwidth consumption?

3.1 Audio Time Stamping

One could imagine performing fidelity analysis by capturing the audio stream on the client and comparing it with the original. Modifications during transmission, coupled with a client's potentially constrained computational resources, make this approach impractical. Timing markers can be utilized to greatly simplify the matching process.

In its most basic form, the insertion of time stamps can be achieved by periodically altering a sample in the audio stream before it is transmitted. The problem is reduced on the client to locating these markers in the audio stream. However, modifications during transmission make this difficult. A technique is required to harden the markers.

To facilitate the analysis of AV synchronization, tracking the advance of the audio stream is required with minimal 100 ms precision[1]. Above 100 ms of drift, users increasingly begin to notice lip synchronization problems. This places additional demands on the time stamps. For a typical (48 KHz PCM) audio stream, this implies it is necessary to be able to detect alignment to within 4,800 samples.

Additionally, it is not sufficient to insert a simple marker every 4,800 samples to prevent aliasing problems. The challenge is to encode, using less than 4,800 samples, locally unique time stamps that can survive the significant modifications encountered. These modifications include MP3 compression (for offline timestamp insertion); MP3 decompression (playback via an audio tool); lossy compression (to control streaming bandwidth); network congestion, packet loss, and jitter; and realization on the client device.

To achieve this time stamp hardening, spread spectrum techniques are leveraged. Spread spectrum signals use special fast codes, called pseudo-noise (PN) codes, that run at many times the information bandwidth or data rate[2]. The rate of the PN code can range from approximately 50x to over 1000x the information data rate. The faster the code, the greater the spreading, and the more robust the information encoding.

3.2 Inserting Audio Time Stamps

Using spread spectrum techniques each timestamp is spread over multiple audio samples. The technique performs the following steps:

(a) Insertion of timestamps

1) A PN code is generated.

2) The timestamp data modulates the n-bit PN code (the timestamp data is "spread", with each bit modulating an entire PN code).

3) The resulting signal modulates a carrier.

4) The resulting carrier is incorporated into the original audio stream.

5) The modified audio stream is used for playback.

(b) Recovery of timestamps:

1) The modified audio stream is received.

2) Using the same PN code, the client acquires the received code and locks to it.

3) The received signal is correlated with the PN code, extracting the time stamp data.

The client acquires the transmitted PN codes by undertaking correlation operations, where the received stream is correlated against the known PN code. These special PN codes have a critical property. The periodic autocorrelation function basically is two valued, with a peak at 0 shift and zero elsewhere. There is a very significant spike in correlation when the two codes are aligned precisely. Misalign the codes by as little as a single sample and the result of the correlation operation is diminished significantly.

Accordingly, in order to locate a PN code in the received stream, the receiver needs to advance its PN code gradually across the received stream and recompute the correlation between the two after each sample-by-sample move. When the correlation exceeds a predefined threshold, the code in the audio stream has been located or acquired. Alternatively, rather than using a preset threshold, the code can be moved across a predefined portion (window) of the audio stream and the maximum correlation observed deemed to represent the location of a code.

Once the code is detected, the client can proceed to recover the time stamps bit by bit. Due to the lack of timing synchronization between the host and client device, additional analysis must be undertaken to recover the timestamps. When the client examines the audio stream, this stream will have been subject to significant modification. Further, the local audio stream continues to advance at the defined rate, even if the client has stopped receiving audio from the host, is receiving at a significantly reduced rate, or if packets are being dropped or are arriving out of order. As a result, the client's audio stream might not contain any PN codes during certain intervals.

As a result, it is necessary to check continually for synchronization with the PN code. This can be achieved by monitoring the result of the correlation operations performed as analysis is advanced across each successive chunk of the audio stream. If the correlation result drops below a (dynamically adjusted) threshold, synchronization must be considered lost, and resynchronization with the stream is required.

Once this process is complete, the sequence of recovered bits must be transformed into a sequence of time stamps. This process starts by determining where each time stamp starts and ends. However, this sequence could contain replicated, missing, or corrupt bits. Therefore, it is necessary to use unique symbols to demarcate each time stamp. For example, if a time stamp is 8 bits long, and if the time data is constrained such that bits 0 and 7 are always zero, then a sequence of 8 ones could be used to demarcate each time stamp. When the bit sequence of 8 ones is observed, it likely corresponds to a demarcation marker.

To identify the impact of missing, corrupt, or duplicate bits, several adjacent time stamps are considered in parallel. If they are monotonically increasing, then it is highly likely that the time stamps not corrupt and can be considered. The probability of errors can be managed by changing the number of time stamps considered and inserting parity bits (or error correction (EC) codes) to guard more directly against corruption.

The biggest challenge is to insert sufficiently large time stamps frequently to provide the high-resolution timing required for tracking AV synchronization. While the PN code length can be reduced, the shorter the code the less spreading, and the less immunity there is to audio stream modification. In experimentation, 255-bit Kasami codes[3] were the shortest codes found to provide sufficient protection for the WAN environment, as illustrated in Figure 3.

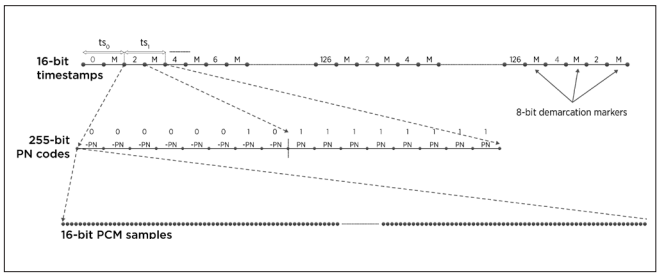

Figure 3. Using Spread Spectrum techniques to insert timestamps into a PCM audio stream.

3.3 Interpreting Audio Time Stamps

The simplest measure of audio quality that can be derived from the time stamps is an indication of whether the audio is being played back at the correct speed on the client. To fully gauge audio fidelity, several additional parameters must be considered.

1. Over what percentage of the audio stream was it possible to lock with the PN code? Failure to detect the code likely means the audio was missing, or badly corrupted during that portion of the audio stream.
2. When a code is detected, how does the correlation compare with the idealized result? The lower the score, the more audio samples were lost, replicated, or significantly modified.
3. At the larger time scale, at which audio degradation becomes noticeable to humans, how do all the various chunks of audio reassemble the original stream?

Metrics describing (1) and (2) can be extracted directly from the PN code results. For (3), analysis was achieved by performing sequence alignment[4]. This process is illustrated in Figure 4. The alignment is undertaken by scanning a chunk (a couple of seconds of audio) of the recovered time stamps, and identifying the longest sequence of monotonically increasing time stamps (Figure 4, steps 1-3). This represents the largest sequence of good quality, correctly timed audio and forms the basis of the sequence alignment.

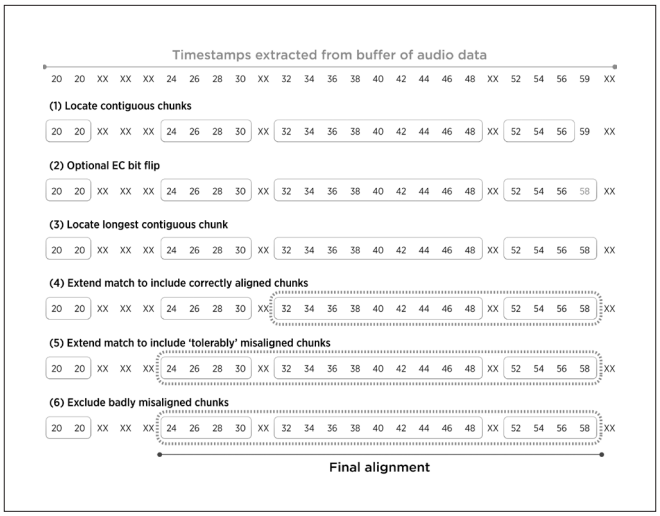

Figure 4. Computing overall audio quality via local alignment of timestamps.

Other non-contiguous groups of samples within the buffer may be correctly timed with respect to this longest contiguous match. These samples also contribute to the reconstruction of the audio stream. The initial match is then 'extended' by including these other smaller groups of audio (Figure 4, step 4).

Small slips in synchronization are not detectible by the human ear. As a result, it is possible to use these samples to further expand the set of matching samples. In this case, the degree to which each of the remaining time stamps in the analysis window is out of synchronization is considered. For those where the delta is less than a predefined threshold, they are considered part of the final match (Figure 4, steps 5 and 6). The percentage of included or good samples is computed and provides, in conjunction with the other metrics, a good measure of the user's audio experience. It also forms the basis of the quality metric in the results section of this document.

4. AV Synchronization

To fully understand a user's AV experience, it is important to understand how the audio and video streams correspond to each other. Accurately measuring AV synchronization requires determining which audio samples are being played on the client device in parallel with which video frames. To achieve this, timing information obtained from the video stream is compared with the timing information obtained from the audio stream. Ideally, the times from the two streams should match. The data from the video stream should indicate that frame X was played at time Y, and the audio data should indicate that the audio associated with frame X was played at time Y. Any difference between the two means the streams are drifting apart and imperfect AV synchronization is the result.

5. Interactive Operations

To enable monitoring of interactive operations, techniques were developed to track the responsiveness of desktop applications as they respond to user requests. In this situation, the total latency of the requested operation(s) is reported. Additional techniques were developed that provide user experience metrics for window drag, scroll, and pan operations.

5.1 Office Applications

The VMware View Planner™ tool was used to simulate typical day-to-day office user operations, including typing, launching applications, browsing Web pages and PDF documents, and checking email. VMware View Planner is an automated measurement framework that is comprised of a typical user workload, a robust technique to measure end-to-end latency accurately, and a virtual appliance to configure and manage the workload run at scale. In the workload, a variety of typical office user applications are included, such as Microsoft Outlook, Microsoft PowerPoint, Microsoft Excel, Microsoft Word, Adobe Reader, Windows Media player, and Internet Explorer.

The supported applications and their corresponding operations are shown in Table 1. To measure the response time of these operations on the client side, a watermarking technique was designed with several key criteria:

- Work under adverse network conditions.
- Smart watermark placement to ensure the watermark does not interfere with application updates.
- Adaptive watermark placement that works for any screen resolution.

APPLICATION ID	APPLICATION	OPERATIONS
1	Firefox	["OPEN", "CLOSE"]
2	Excel	["OPEN", "COMPUTE", "SAVE", "CLOSE", "MINIMIZE", "MAXIMIZE", "ENTRY"]
3	Word	["OPEN", "MODIFY", "SAVE", "CLOSE", "MINIMIZE", "MAXIMIZE"]
4	AdobeReader	["OPEN", "BROWSE", "CLOSE", "MINIMIZE", "MAXIMIZE"]
5	IE_ApacheDoc	["OPEN", "BROWSE", "CLOSE"]
6	Powerpoint	["OPEN", "RUNSLIDESHOW", "MODIFYSLIDES", "APPENDSLIDES", "SAVEAS", "CLOSE", "MINIMIZE", "MAXIMIZE"]
7	Outlook	["OPEN", "READ", "RESTORE", "CLOSE", "MINIMIZE", "MAXIMIZE", "ATTACHMENT-SAVE"]
8	7Zip	["COMPRESS"]
10	Video	["OPEN", "PLAY", "CLOSE"]
12	WebAlbum	["OPEN", "BROWSE", "CLOSE"]

Table 1: View Planner applications with their IDs and operations

To precisely determine the application response time on the client side, meta-data was overlaid on top of the start menu button that travels with the desktop display. By putting the metadata on top of the start menu, application rendering interference is avoided, and metadata can be sent without any overlapping issues. This metadata encodes each application with its operation.

As shown in Table 1, each application is assigned an application ID and has a set of operations. If there is a need to encode a PowerPoint "Open" operation in pixel values, a predefined encoding technique is applied and the value "61" is put in the metadata. On the client side, the metadata is detected and the time stamps are recorded at the start and end of the operation. An initial timestamp is generated when the client initiates the request. The application response time is computed from the difference between these timestamps. This client-side measurement more accurately reflects the operation timing observed by the user, and prevents the over-consolidation problems that can result from merely measuring the operational latencies on the server.

VMware View Planner is designed to scale from a few virtual machines to thousands of virtual machines distributed across a cluster of hosts. It is invaluable to VDI administrators, enabling them to perform scalability studies that closely resemble real-world VDI deployments and gather useful information about configuration settings, hardware requirements, and consolidation ratios.

5.2 Scrolling and Dragging

The method described works well for traditional office applications with limited mouse activity. This method can be further extended to scenarios in which the position of the window, or the visible region of the window, changes in response to mouse events. To track the responsiveness of window drag or window scroll operations, additions must be made to the existing watermarking technique.

VMware's approach to measure the responsiveness of such operations works by "painting" the test window with a well-defined, fixed pattern (Figure 5). The pattern is chosen carefully to ensure every pixel inside the window has a unique color that directly relates to the pixel's location. There is a one-to-one mapping between the color of a pixel and it's (x, y) coordinates within the window. After extensive experimentation to overcome the significant color space (and hence positional) errors that can be introduced by lossy compression, the mapping used is Pixel-Color (x, y) = (3*x + 4097 * y).

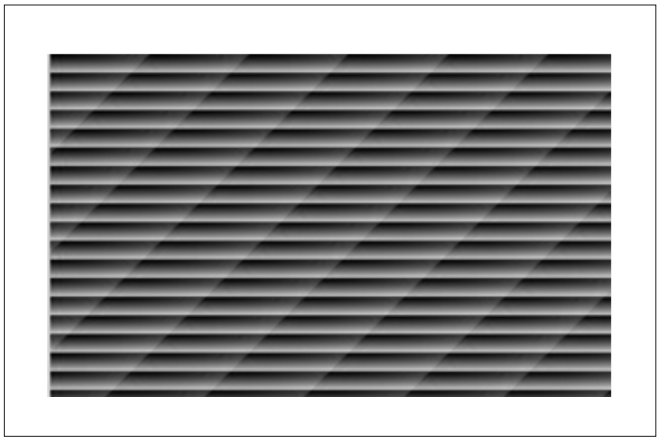

Figure 5. Sample bitmap used to "paint" the window and to track its motion.

To detect window movement, a predefined number of observation points are scattered across the screen background. As the window is dragged or scrolled, the observation points record the colors of the pixels observed. Since the colors correspond to (x, y) coordinates within the window, the colors observed by the observation points pinpoint the window's position and accurately track the trajectory of motion of the window or the sequence of scrolling operations. Potential positional errors resulting from lossy compression can be minimized further by using a voting scheme across all of the observation points that are overlapped by the window at each stage of its move.

Using this technique, the client side can accurately track the motion of the window (or the number of the pages that were displayed). This information is correlated against the client's list of mouse-clicks and user input that it issued to the remote view desktop to trigger the move and scroll operations, to determine responsiveness accurately and user experience. Accordingly, to determine the responsiveness of window drag operations, the expected trajectory of the window, the corresponding observed values, and the time stamps of the mouse clicks and screen updates are recorded. If:

- M_1, M_2, M_3..., and M_n represent mouse clicks.
- T_{M1}, T_{M2}, T_{M3}... and T_{Mn} represent time stamps for the clicks.
- U_1, U_2, U_3... and U_n represent screen updates.
- T_{U1}, T_{U2}, T_{U3}... and T_{Un} represent time stamtps for the updates.

After significant experimentation, four key metrics were targeted that impact significantly the preserved quality of the operation:

1. Run-time (**R**) = (T_{Un} - T_{M1})
2. Smoothness: **μ** and **σ** of T_{U1}, T_{U2}, T_{U3}, .., and T_{Un}
3. Responsiveness: (**μ**') = (**Σ** (T_{Ui} - T_{Mi}))/m
4. The number of frame updates (denoted **f**) seen by a VDI client.

These four metrics are combined to create a single *user experience* number (denoted *UE*) that succinctly characterizes the test and allows different test configurations and environments to be compared easily. The UE number is generated as:

$$\left(\frac{R_{SUT}}{R_o} * \frac{\mu_{SUT}}{\mu_o} * \frac{\sigma_{SUT}}{\sigma_o} * \frac{\mu'_{SUT}}{\mu'_o} * \frac{f_o}{f_{SUT}}\right)$$

where the subscript *SUT* represents a system-under-test and the subscript *0* represents a reference system, such as a top-of-the-line desktop PC. The UE number falls between 0.0 and 1.0, where a score of 1.0 equates to indistinguishable quality to the reference system. In addition to defining these metrics, an extensive suite of tests was developed that covers a wide variety of different interactive operations. The overall user experience score for a given system is defined as the geometric mean across all of the UE numbers generated by running the test suite.

6. Results

The techniques developed are broadly applicable to evaluating user experience across a wide range of applications and use cases. To provide real-world results relating to a well-known product, this section presents user experience results for VMware View, a VDI solution.

The types of results highlighted in this section illustrate how the techniques presented enable service providers, administrators, product users, and developers to generate accurate quantitative results about how their products and infrastructure perform under a wide variety of situations. Critically, this supports data-driven consolidation decisions and considered product choices, and provides developers with clear insights into key improvements for future products.

6.1 Video Quality Measurement

The video watermarking technique was used to create a suite of video files (320x240 resolution @ 25 fps) that each encompass different video attributes. In this video repository, some videos

contain rapid, wide-spread motion. In others, the movement is restricted to a small part of the frame. This coverage provides insight into how various protocols adapt and their different performance optimizations. The video files and their encoding and attributes are as follows:

- dp: Powerpoint slideshow (webinar)
- hi: freeway driving video with fast screen changes
- im: high-quality images stitched together to form a video
- le: leaves blowing in the wind, with only a few colors (green/white) and small back and forth movements
- na: fast horizontal panning over an urban setting with zoom-in and zoom-out
- pr: human face movement (news anchor)
- ra: random pixel images stitched together
- ro: rocky mountains, fast horizontal panning across a mountain scene
- st: diagonal panning across a static background image (stones on a river bed)
- te: tennis match with localized fast movement
- wa: waterfall, movement in the center with zoom-in

To illustrate the potential of the video watermarking technique, two use cases are presented. First, different VDI display protocols, Remote Desktop Protocol 7 (RDP7) and PC over IP (PCoIP), were compared under varying network conditions using these videos and the video plugin framework. Multimedia redirection is disabled for RDP7 and PCoIP in this and subsequent results. The comparative frame rate results for RDP7 and PCoIP presented in Figure 6 clearly illustrate that VDI protocols can differ widely in their ability to handle different network conditions. Critically, these techniques provide customers with the ability to research and characterize application performance easily to ensure suitability before purchase.

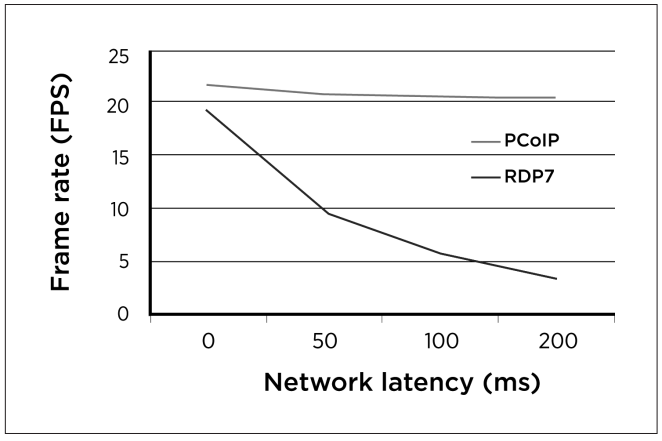

Figure 6. The effect of increasing network latency on video frame rates.

Second, the entire video suite was run using VMware View and the PCoIP VDI display protocol for two different network conditions:

(1) LAN conditions: no network constraints

(2) WAN conditions: 2 Mbps downlink with 100 ms round trip latency

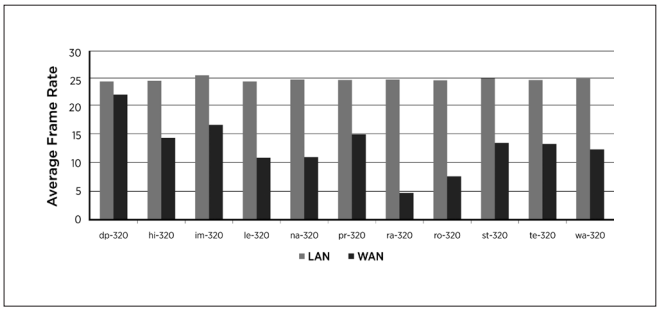

Figure 7. Video frame rates achieved for different videos under both LAN and WAN conditions.

The average frame rates observed for the different videos for LAN and WAN conditions are shown in Figure 7. As also illustrated in Figure 6, close to the full 25 frames/second is observed for almost all the videos in LAN conditions, where there is essentially unlimited available bandwidth. When network bandwidth is constrained (WAN, 2Mbps), the differences between the videos manifest in different achievable frame rates, with frame rate ranging from 7 frames/second up to 22 frames/second. This is expected, as the number of pixels modified per frame varies greatly from video to video. The highest frame rate recorded is for "dp-320", a slideshow video in which very few pixels change from frame to frame. The lowest frame rate is observed for "ra-320", in which each video frame is composed of random pixel data and every pixel value changes with every frame. The random nature of the changes makes it unsuitable for compression and even motion estimation optimizations.

6.2 Audio Quality Measurement

In VDI deployments, consolidation ratios are a key concern, and directly impact the achievable cost savings. Figure 8 depicts the impact of over consolidation on audio quality, illustrating the importance of load decisions when applications with hard real-time constraints, such as audio, are considered. Using the techniques presented, it is now possible for customers to determine safe consolidation ratios accurately and easily for any given multimedia profile of their users.

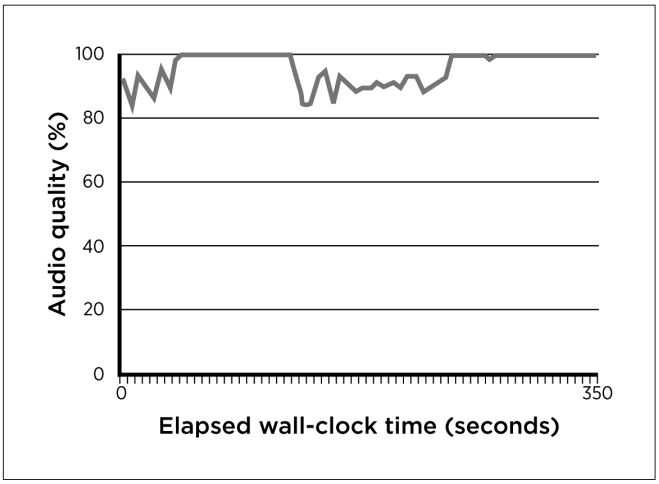

Figure 8. Audio quality problems on an overloaded core.

It is also important to ensure that VDI solutions are sufficiently robust to handle the competing demands of audio and video streams in the presence of network resource constraints. For instance, if a user is streaming audio over a low-bandwidth WAN network and then starts scrolling around image-rich Web pages, does the user experience significant audio glitches? Figure 9 illustrates a run with a 300 Kb/second bandwidth limit and highlights how PCoIP manages to maintain acceptable audio quality, even when the user is undertaking bandwidth-intensive scrolling operations. The protocol correctly prioritizes the audio stream to ensure uninterrupted audio playback and the best user experience.

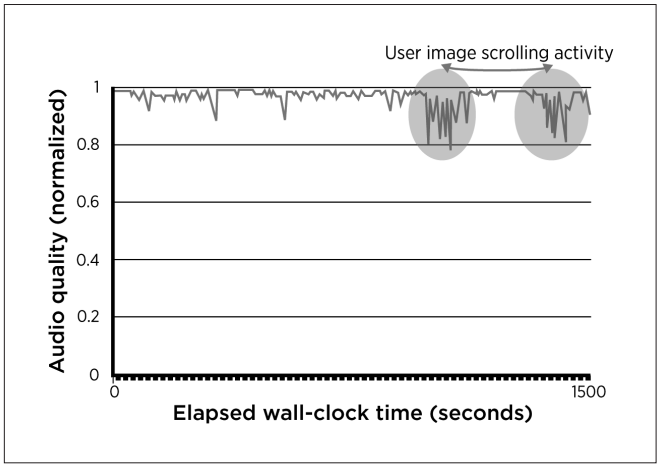

Figure 9. Audio quality in the low-bandwidth situations.

6.3 AV Synchronization Measurement

To highlight how AV streams can drift in the VDI environment, Figure 10 illustrates the AV synchronization during video playback for RDP7 (LAN). The average AV drift is approximately 180 ms, which is higher than the 100 ms at which ITU indicates drift starts to become noticeable[1].

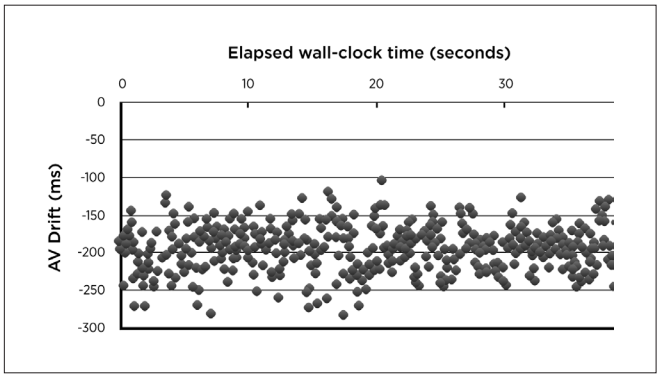

Figure 10. Typical drift in AV synchronization for RDP7.

6.4 Interactive Office Operations Measurement

VMware View Planner is used to characterize precisely the response time of application operations using VMware's novel watermarking techniques. To highlight the effectiveness of these techniques, different network conditions were simulated using a traffic shaper. The impact on the response time was investigated for different display protocols (PCoIP and RDP). In this paper, three network conditions were investigated, corresponding to LAN, WAN, and extreme WAN (very low bandwidth and high latency). The results are presented in Figure 11, and are normalized to the worst case latency. From Figure 11, it is apparent that PCoIP provides significantly faster response times in all network conditions compared to RDP. This is especially true in WAN conditions, where the RDP response time degrades rapidly. VMware View Planner enables this kind of study and provides a platform to characterize the "true" end-user experience.

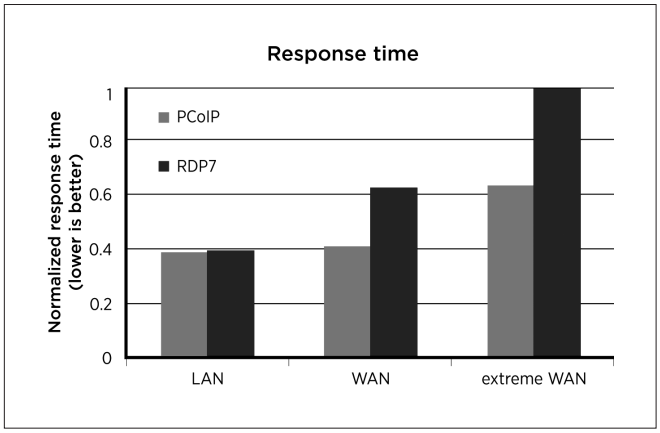

Figure 11. Operation response times for different network conditions, as measured by View Planner.

To illustrate the ability to quantify user experience at scale, VMware View Planner was used to simulate the VDI users on a host and investigate how many virtual machines a host can support while delivering an acceptable user experience. Figure 12 shows an overview of the consolidation improvement that VMware View 5 delivers in comparison to VMware View 4.5 for Windows 7 deployments. Figure 12 depicts the number of desktops that successfully passed VMware View Planner's quality of service (QoS) requirements when running the VMware View Planner workload. A run is considered passed if application response times are below 1.5 seconds, otherwise is labeled as failed. As Figure 12 shows, VMware View 4.5 could achieve 11 Windows 7 virtual machines per core, while VMware View 5 is able to achieve 14.5 virtual machines per core.

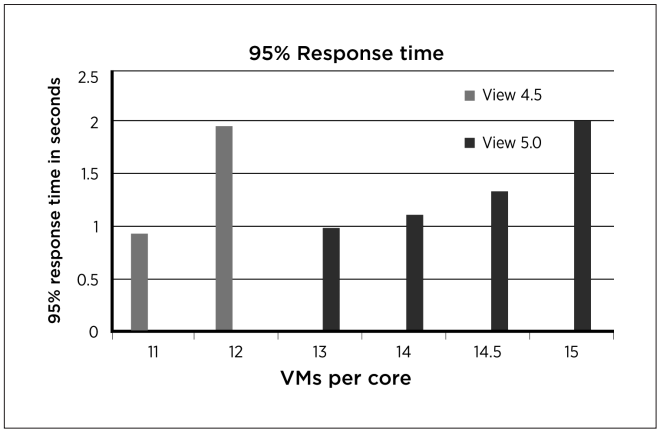

Figure 12. View Planner's 95% response times for different user consolidations (comparing View 4.5 and 5).

Finally, Figure 13 illustrates the impact of network conditions on the user experience when running the interactive test suite using VMware View and the PCoIP protocol. As expected, user experience is impacted by the progression from LAN conditions to increasingly severe WAN conditions. It is also apparent that PCoIP can adapt gracefully to these network constraints and continue to deliver a high-quality user experience even for WAN networks with 100 ms of round trip latency.

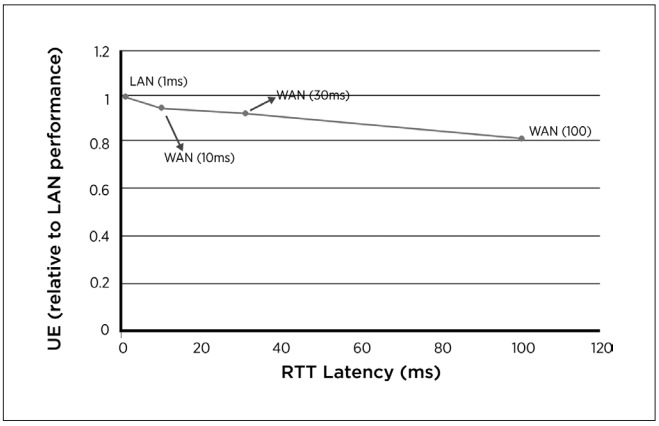

Figure 13. Impact of network conditions on Interactive performance.

The results in this section highlight the sensitivity of the user experience to resource constraints, and emphasize the importance of these techniques for service providers, such as cloud providers and content delivery network owners, to characterize their load and correctly size deployments for good performance and cost effectiveness.

7. Related Work

There is significant interest in monitoring user experience, particularly with respect to the sizing of large VDI deployments[5, 6]. Current solutions lack the precise, real-time, end-to-end quality analysis provided by the techniques presented in this paper. For many, the user experience is estimated by timing basic operations on the guest, essentially looking at latency and not the quality of operations)[7]. Others simply provide aggregate CPU and bandwidth statistics, attempting to highlight any potential resource bottlenecks in the proposed configuration. Given the adaptive nature of the lossy compression used by these applications, hitting a physical resource limit does not necessarily represent maximum user load, as quality can be traded off against an increased number of users. Accurately make these tradeoffs requires the ability to accurately analyze user experience on the client device.

One solution is to use subjective analysis and estimate quality by manually watching the client's desktop during a test run. Such an approach is inaccurate, labor-intensive, and does not scale. This is especially problematic because—due to hysteresis effects in the bandwidth estimation schemes—there can be considerable variation in the quality experienced across users, even when they are all running identical workloads. As a result, manually monitoring user experience in a couple of sessions out of a much larger deployment can not provide true QoS metrics for the entire user population. Other slightly more automated techniques involve capturing the image stream en route to the client device, attempting to reconstruct the images after the run, and estimating quality[8]. This technique does not provide end-to-end coverage, and often can be prevented by the encryption of the display data en route to the client.

In contrast to these existing approaches, the technique presented in this paper provides automated, real-time, end-to-end tracking of the user experience over all users, providing meaningful QoS metrics. Additionally, it is application, protocol, and network topology agnostic, and is not impacted by the use of encryption.

8. Future Work: Analysis Of Live Deployments

In the capacity planning environment, offline insertion of the timing markers is preferable as it provides true end-to-end fidelity measurements. Moving to live deployments, it is necessary to provide feedback about the user experience without necessarily having a priori access to the media and application. (Such access generally is feasible for video and audio streaming sites.) Furthermore, it is no longer acceptable to noticeably perturb audio and video when inserting timing markers. Consequently, there are a number of ways in which these techniques can be applied to real-time quality analysis in live deployments:

- Hook into applications directly and inject timing markers into the imaging streams in real time. In many applications, this can be achieved readily via the use of plugins. It is also necessary to alter the placement of the watermarks to minimize their visual appearance. Furthermore, the introduction of PN codes into the audio stream must not be perceptible. As a result, it is necessary to employ longer PN codes in order to ensure they can be correctly recovered on the client device.
- Rather than attempting to directly measure user experience within a user's virtual machine, periodically run lightweight test virtual machines that run the user experience tests described herein. Collect data that can be used to provide information about the prevailing conditions being experienced by users.
- Provide users with a tool that can benchmark their current user experience, allowing users to report, or even root cause, shortcomings.

Finally, in addition to merely providing a quality warning to administrators and users, VMware is investigating the use of quality measurements to enhance application automatic response to developing problems by providing accurate quality information to guide:

- Tuning resource usage
- Throttling frame rates
- Down sampling audio
- Dynamically correcting drift in AV synchronization
- Improving decisions about altering lossy compression levels

These optimizations can be performed directly at the application level, or resource problems can be communicated to higher-level management functionality to enable better global performance optimization. For instance, using the VMware View example, the VMware View connection server can be informed about quality issues that can, in turn, work with virtualization management

software, such as VMware vCenter Server Virtualization Management, to automatically modify resource allocation and virtual machine placement to optimize performance across all of the virtual desktops under its control.

9. Conclusions

For applications providing real-time streaming of content, including VDI deployments, SaaS solutions, and video streaming, it is critical to ensure that users have the best possible experience, even when running in the most demanding of situations. In order to ensure a positive experience, there is a requirement to accurately monitor the user experience in live deployments and initial capacity planning and provisioning experiments. This paper discussed novel techniques that provide real-time feedback on key components of user experience, including video and audio quality, audio/video synchronization, and the responsiveness of applications in user environments. To highlight the importance of these techniques, examples were presented in which the techniques provided key insight into the behavior of VMware's VDI product. They clearly illustrate the detailed quality information that can be extracted using these tools, and emphasize how this information is critical for service providers, developers, and customers.

10. References

[1] ITU-R, "Relative Timing of Sound and Vision for Broadcasting", BT. 1359-1 (1998).

[2] A. Viterbi, "CDMA: principles of spread spectrum techniques", (1995).

[3] T. Kasmi, "Weight Distribution Formula for Some Class of Cyclic Codes", Tech Report No. R-285, Univ. of Illinois, (1966).

[4] D. Mount, "Bioinformatics: Sequence Alignment and Genome Analysis", Cold Spring Press, 2004.

[5] Scapa Tech, "Scapa TPP for Thin Client".

[6] Citrix, "Benchmarking Citrix XenDesktop 4 using Login Consultants VSI 2.1", Whitepaper, 2010.

[7] J. Nieh, S. J. Yang, and N. Novik, "Measuring Thin-Client Performance Using Slow-Motion Benchmarking", ACM Trans. Comp. Sys., 21:87–115, February 2003.

[8] S. Yang, J. Nieh, M. Selsky, N. Tiwari, "The Performance of Remote Display Mechanisms for Thin-Client Computing", Proc. of the USENIX Annual Technical Conference, 2002

StatsFeeder: An Extensible Statistics Collection Framework for Virtualized Environments

Vijayaraghavan Soundararajan
ravi@vmware.com

Balaji Parimi*
balaji@cloudphysics.com

Jon Cook
cookj@vmware.com

Abstract

Although virtualization has eased the day-to-day lives of IT administrators, analyzing the performance of virtualized infrastructure remains very difficult. Admittedly, collecting statistics for a large number of servers is not a new problem. Neither is analyzing the virtualization performance of an individual virtual machine or the performance of a multi-tier virtualized application. The challenge lies in building tools to enable all three to occur at the same time and provide meaningful information for end users, many of whom are not virtualization experts.

This paper describes a scalable performance monitoring framework for virtualized environments. It first considers the pain points experienced by customers trying to do performance troubleshooting in virtualized environments. The common themes present in most customer environments include the need to gather granular data in a scalable way and to correlate data across layers, from the application to the guest operating system to the physical host. Using these requirements to guide design, the paper describes a prototype for scalable statistics collection that leverages currently existing statistics-gathering APIs and attempts to address these use cases. It describes how the authors were able to use this design to address several real-world troubleshooting use cases. It concludes with results demonstrating the scalability of the approach for typical virtualized datacenter sizes.

Categories and Subject Descriptors

D.m **[Miscellaneous]**: Virtual Machines, system management, performance monitoring.

General Terms

Performance, Management, Applications, Tools.

Keywords

Virtual Machine management, cloud computing, datacenter management tools. performance monitoring applications, diagnostic tools, datacenter management tools.

Introduction

Virtualization has eased many of the day-to-day tasks in today's datacenters. One area that has arguably become more difficult is performance debugging. Prior to virtualization, it was common to provision one operating system and application per physical host and give an end user remote access to the physical machine via an interface such as an Integrated Lights-Out (ILO) interface[12]. If an end user complained about slow application performance, the end user could use traditional operating system-based tools, such as perfmon or top, to try to determine resource bottlenecks and potential sources of performance issues. In turn, the IT administrator could examine resources beyond a single host, such as shared storage or networking, to try to help with performance debugging.

In a virtualized environment, a hypervisor layer is inserted between the hardware and the operating system, allowing multiple operating systems and applications to reside on a single physical host. A result, these tried-and-true debugging techniques must be done much more carefully. For example, a performance issue can be due to an under-provisioned virtual machine (one virtual CPU (vCPU) instead of two), or an over-committed host (20 2-vCPU virtual machines sharing eight physical cores). Even the coupling of application and virtualization management traffic can cause the issues[22], such as a migrating virtual machine consuming network traffic that is shared with application traffic.

Each of these issues could resemble the others in terms of symptoms. As a result, it is possible for neither the application owner nor the IT administrator to have a complete picture of all of the issues. These problems are magnified in today's datacenters, many of which have tens of thousands of virtual machines and separate administrators for the application owner and the infrastructure administrator.

There has been a wide body of research in tools for automated detection of performance issues[26 31]. While these tools can be quite effective for anomaly detection, they have two key limitations. First, they often lack an API for retrieving data from the platform. If a user finds a particular piece of data useful, that user does not necessarily have an easy way to retrieve and store that data. Second, these tools are typically data agnostic. While this can be extremely valuable for viewing disparate data streams and finding connections between them, it is not as helpful when trying to provide an "actionable" suggestion based on incoming data. For example, if a problem is known to be specific to virtualization, and simply adding an extra vCPU is known to solve the problem, then an automated tool working on agnostic data is not necessarily able to offer guidance to solve the problem.

*This work was performed while Balaji Parimi was employed at VMware.

This paper tries to take a simpler approach to performance troubleshooting in a virtualized environment, addressing common customer questions and pain points without necessarily requiring a user to learn a complex API for statistics collection and analysis. For example, some customers simply want to gather statistics, such as CPU usage per virtual machine, and place them in their own pre-existing performance databases. Other customers want guidance about what statistics are important and rules of thumb for detecting issues. In both cases, customers need a scalable method for retrieving data and an easy way to analyze it. The approach described here is not intended to be a replacement for advanced analytics. Instead, it is an intermediate step that tries to address the most common performance issues.

The paper describes StatsFeeder, a prototype extensible statistics framework designed with the end user in mind. It performs several key tasks:

1. Provides a mechanism for retrieving statistics without understanding the underlying virtualization API
2. Allows users to store statistics in whatever format is desired
3. Allows users to create mashups of important statistics
4. Allows users to store statistics at arbitrary granularity
5. Enables users to easily combine statistics across various virtualization layers

This paper discusses a StatsFeeder implementation and provides a number of sample use cases. Section 2 provides an overview of data collection in virtualized frameworks and the StatsFeeder architecture. Section 3 describes how the design helps address key customer pain points related to statistics collection in virtualized environments. Section 4 describes several concrete use cases to validate the design. Section 5 provides a preliminary performance evaluation of the scalability of the tool. Related work is discussed briefly in Section 6, with concluding remarks in Section 7.

2. Overview And Architecture

Before describing the statistics collection framework, it is helpful to understand virtualized datacenter management. This section provides a brief overview of management and monitoring in virtualized environments. More detail can be found in[22] and[23].

A conventional datacenter consists of physical hosts, storage and networking infrastructure, power and cooling hardware. A management and monitoring framework performs operational tasks, such as powering down servers, and assesses the basic health of datacenter components. A virtualized datacenter consists of the same hardware components as a physical datacenter. In a virtualized datacenter, however, each host contains a hypervisor layer for running virtual machines on top of the physical hosts. In addition, the management framework is extended to include the management and monitoring of virtual machines and hosts.

Figure 1 illustrates the similarities between the management infrastructures for a physical datacenter versus a virtualized datacenter. In general, a central management server or group of servers monitors the status of each host and propagates information to system administrators. In addition, configuration and statistics information is persisted in a database for auditing purposes, capacity planning, and troubleshooting. For example, if an end user complains about slow application performance over a given weekend, the administrator might access historical performance statistics about the virtual machine in which the application was hosted in order to see if the virtual machine had resource contention issues.

Management systems also provide an administrator user interface ("Admin UI" in Figure 1) to allow system administrators to view the health status of the infrastructure or perform tasks such as reconfiguring or rebooting a host. Several commercially-available tools are available for monitoring and managing both physical and virtual infrastructures, including VMware vSphere®[32].

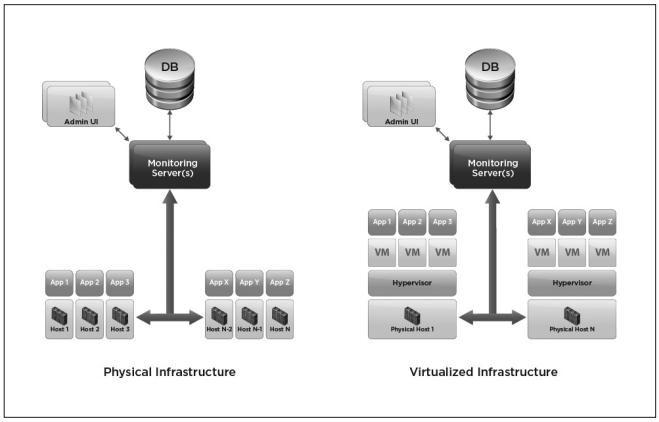

Figure 1. Contrasting Physical Datacenter Management and Virtualized Datacenter Management.

In a physical environment, each host might run a single application, and the management layer must monitor hosts, applications, storage, and networking. In a virtualized environment, applications run inside virtual machines, and the management layer also must monitor virtual machines and the hypervisor. In VSphere, the monitoring servers are called VMware vCenter™ servers, and the physical hosts run VMware® ESX®.

To put this architecture into perspective, it is important to understand the typical use case. An enterprise customer can have hundreds of hosts and thousands of virtual machines running within a datacenter. Unlike a truly cloud-scale environment, such as a warehouse-sized computer or an Amazon EC2 deployment, the applications are often legacy applications. These legacy applications require constant monitoring in case an individual instance goes down—especially if the application is a mission-critical application such as a mail server. In addition, if an individual instance fails, often it is important to determine the precise cause of the failure, rather than simply restarting the application. In contrast, for an infrastructure hosting a massively-scalable application such as Google Earth, it might be cheaper and more feasible to simply spin up a new version of an application when a node failure occurs, rather than try to do any kind of online diagnosis.

As a result of these diagnostic requirements, the collection of detailed monitoring data is extremely important, and such data often is archived in a highly available relational database. For example, in vSphere, the monitoring server (the VMware vCenter server) is responsible for periodically querying hosts, collecting performance data, and archiving it to the centralized database. This database is a SQL-compliant database such as Oracle Database or Microsoft SQL. By using a commodity relational database, IT administrators can leverage rich tool sets for backup, restoration, and data mining that are not present with existing NoSQL or key-value store alternatives.

To allow programmatic access to the per-host and per-virtual machine statistics data for a virtualized infrastructure, management solutions usually export APIs that abstract the database schema details and allow efficient searching of the stored data. In vSphere, the API for performance data is called the *PerformanceManager*. The PerformanceManager abstracts the details of how performance data is stored in the VMware vCenter database and allows users to query for data by indicating the name of an entity (such as the name of a given virtual machine or host), the metadata for the metric (a shorthand form of the metric name), and a time region for the metric (such as all data over the past hour).

A rich set of metrics are available per virtual machine and host. For example, in vSphere, a given virtual machine might have over 500 metrics[29] at any given timestamp, and a host might have over 1,000. It is important to be selective about what is stored in the database so that the database does not grow too large and queries do not become prohibitively slow. The next section talks briefly about how vSphere collects data across hosts and virtual machines, and how it decides what to store in the database.

2.1 Statistics Collection in VMware vSphere

Because StatsFeeder is built upon VMware vSphere, it is helpful to understand how vSphere collects and stores statistics. Each host stores statistics data for up to one hour. Within that one hour period, the VMware vCenter server queries each host, retrieves a subset of this data, and archives it to the centralized SQL database. Such a design allows vSphere to be deployed in existing enterprise environments, in which it is typical to deploy middleware applications that are connected to SQL databases in backend database farms.

On a host, statistics are collected at a 20 second granularity. Every 20 seconds, a daemon on each ESX host collects metrics (such as CPU usage in MHz) for each virtual machine on the host and for the host itself. The ESX host then archives this data in a file on the host. These statistics are stored for up to one hour. Since the statistics are at 20 second granularity and are stored for up to one hour, there are up to 180 samples per metric per hour. A given metric can be an aggregate metric (such as the sum of CPU usage for all CPUs on the physical host), or it can be a per-instance metric (such as CPU usage of a given physical CPU). This 20s data is referred to as *real-time* data.

As mentioned earlier, a subset of this real-time data is stored in the database. The statistics that are stored in the database are called *historical* statistics. Since there are over 500 different types of metrics (such as CPU usage in MHz or network bandwidth consumed in Mbps) per virtual machine, and there can be multiple instances for each metric, it would be prohibitively expensive to store this data in the database at 20s granularity for long time periods for every host and every virtual machine. Instead, VMware vCenter limits the amount of data it collects and stores from each host. It uses two techniques to limit the data collection: statistics rollups and statistics levels.

2.1.1 Statistics rollups

One technique that VMware vCenter uses to limit the amount of data stored in the database is to store at a coarser granularity in the database than the granularity of storage on the ESX hosts. Each ESX host stores data for its virtual machines at 20s granularity for one hour, while the VMware vCenter database stores data for all hosts and virtual machines at five-minute granularity for one day. The conversion of every group of fifteen 20s samples (such as five minutes' worth of data) into a single value for that five-minute interval is called a *rollup*.

There are four distinct intervals of storage: past day, past week, past month, and past year. Each stores data at successively coarser granularity (five minutes, 30 minutes, two hours, and one day, respectively). These intervals and granularities are borrowed from the networking community and the Multi Router Traffic Grapher (MRTG) standard[18]. By using rollups instead of storing 180 samples per hour per host or virtual machine, only 12 samples are stored per hour for past-day statistics.

2.1.2 Statistics Levels

Another technique that the VMware vCenter server uses to reduce traffic to the database is to limit which metrics are archived in the database. For example, certain statistics might be deemed more valuable than others, and therefore more valuable to be persisted. The relative importance of statistics is specified using the concept of a statistics *level*. This level varies from one to four, with one being the least-detailed statistics level and four being the most detailed. For example, statistics level one usually is confined to aggregate statistics over a host, such as the overall CPU usage of the host or the overall network bandwidth consumed by the host. Statistics level three incorporates per-device statistics, such as CPU usage of a host on a per-CPU basis, or per-virtual machine statistics.

The statistics level describes whether or not a statistic is archived in the VMware vCenter statistics database. If a statistic is a level two statistic, but the VMware vCenter statistics level is level one, this statistic is not stored in the database and users cannot query its historical values. The statistics level can be varied on a per-metric basis so that any statistic ultimately can be archived in the VMware vCenter database. In practice, the statistics levels are defined to provide increasing levels of value for minimal database cost. While this approach has its benefits and drawbacks, the basic concept behind statistics levels is that a user can learn basic information with the level one counters, and can increase the statistics level temporarily for troubleshooting purposes.

2.1.3 Statistics Queries: Real-time versus Historical

When the VMware vCenter server receives a request for a given statistic for a given virtual machine, it first determines if the statistic is a real-time (20s) statistic or an historical statistic (*past day* statistic). If the request is for a real-time statistic, then the request is forwarded directly to the ESX host where the virtual machine resides, because

only ESX hosts store 20s data. Recall that the ESX host stores all data for the virtual machine irrespective of statistics level. In contrast, if the request is for a historical statistic, VMware vCenter retrieves the data from the database. It is important to remember that only certain statistics are archived in the database, based on statistics level and desired granularity (five minutes, 30 minutes, and so on.). Users can set the statistics level of VMware vCenter to allow any statistics to be stored at five minute granularity in the database, but this comes at a cost of increased database bandwidth and disk consumption.

For many users, storing the default aggregate data at a five-minute granularity for up to a day in the centralized database is sufficient, and therefore default statistics that are collected at level one are sufficient. However, there are a number of customers who wish to collect more detailed statistics, or store them at 20s granularity. Because ESX hosts store all data at 20s granularity for an hour, one option is for users to periodically send a request to VMware vCenter for such real-time data and archive it in a custom manner. The VMware vCenter server forwards the request to the appropriate ESX host, grabs the data, and returns it to the end user. The end user can then store this data in a custom database. Because such data only is stored on the ESX host for up to one hour, users must request such data more frequently than once per hour to avoid losing data. Such an approach avoids excessive growth of the VMware vCenter database. By using a custom database and querying only the necessary data, users can control the amount of data to be stored and the granularity of storage.

While many users of vSphere have database experience, fewer of them have experience with the APIs required for data retrieval. For these users, the difficulty lies in writing client code to specify the appropriate statistics and retrieve them in a format that they can use for persistence in their custom database. StatsFeeder is designed to address these issues. StatsFeeder queries the 20-second data and allows users to write an adapter to store this data in any convenient format, such as a CSV file, database, or message bus. StatsFeeder also monitors an inventory and makes sure that the statistics related to new virtual machines or hosts also are collected. By not requiring users to increase the statistics level, StatsFeeder keeps the VMware vCenter database small. By allowing users to specify precisely which metrics to collect, StatsFeeder provides statistics specifically-tailored to a customer's use case. Moreover, StatsFeeder is extensible, allowing users to collect statistics within a guest in addition to standard virtualization-specific counters.

2.2 StatsFeeder Architecture

2.2.1 StatsFeeder Design Goals

To illustrate the value of StatsFeeder, consider a customer that wants to write a simple application to periodically collect disk latency statistics for all hosts in a virtualized datacenter. The customer wants to use this data to find out if any hosts are incurring storage-related performance issues. In order for customers to write such a tool, they must determine:

1. The precise name of the counter desired. In this case, for examining host disk latency, the counter name is "disk.maxTotalLatency.latest".
2. How to enumerate all hosts in the environment.
3. How to enumerate all disks on each host so that statistics for each disk can be retrieved.
4. How to monitor whether new hosts are added to or removed from the system, or whether new disks are added to or removed from existing hosts.
5. How frequently to collect such data.

Once all of these steps are followed, customers can use any of the performance APIs[29] available to retrieve the data. The data retrieval can work for a small development environment, yet be too slow for a large production environment. As a result, customers must design for scalability. Finally, now that all of the preceding steps are complete, users can store data, write queries to analyze data, and determine when there are performance issues.

The preceding discussion gives some insight as to the main pain points that customers experience when trying to do performance troubleshooting. Most customers are very familiar with simple shell scripting, but would prefer to avoid writing complicated, multithreaded applications to get simple data. The ideal solution is a tool that highlights all possible performance issues. There are many commercially-available tools that attempt to solve this problem. Until such tools are more robust, a helpful short-term solution is simply to make it easier for customers to get the statistics they want and let them use simple rules of thumb to do basic performance troubleshooting. This is the purpose of StatsFeeder.

2.2.2 StatsFeeder Architecture

StatsFeeder is a prototype reference application for data collection in vSphere. A modular framework was created that can be extended by end users, and mashups are available to demonstrate a number of uses cases. Users simply need to create a configuration file that lists the entities to monitor, the statistics to collect, and the frequency of monitoring. StatsFeeder collects these statistics and presents them to users in a suitable format, such as comma-separated values. The basic flow of StatsFeeder involves reading the configuration file, collecting the metadata about the statistics and entities to monitor, and setting up callbacks to collect the data at appropriate frequencies.

Figure 2 shows a block diagram of StatsFeeder. Each of the components is described in the follow paragraphs.

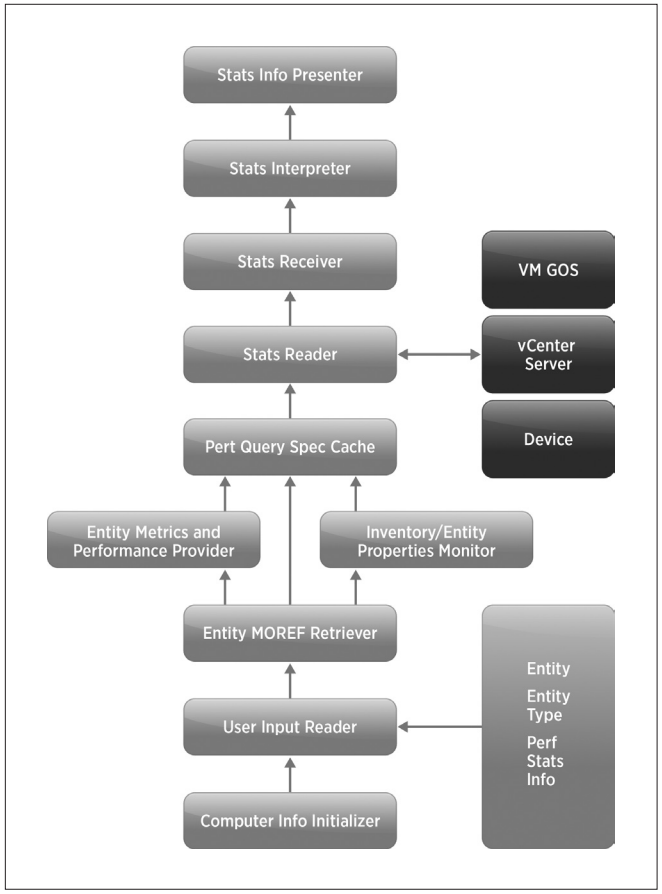

Figure 2. Block Diagram of StatsFeeder. The input to StatsFeeder is a list of entities and statistics, and the output is the set of statistics in a specified format.

- **Counter Info Initializer**. In vSphere, the *Counter Info Initializer* takes the counter names provided by users (from a configuration file), validates them, and retrieves the metadata necessary for requesting this data from the VMware vCenter server.
- **User Input Reader**. The *User Input Reader* reads a configuration file for information about the entities to monitor, the performance statistics to retrieve, and the frequency at which the data should be retrieved.
- **Entity MoRef Retriever**. Every object in a vSphere installation has a name and a unique identifier. This unique identifier is known as a managed object reference, or MoRef. Performance queries in VMware vCenter require use of the MoRef rather than the name of the object, since names can be duplicated. For example, two virtual machines in different datacenters can have the same name, but their MoRefs are unique. The *entity MoRef retriever* takes the names of the entities specified in the configuration file and retrieves their unique MoRefs.
- **Entity Metrics and Performance Provider Summary Initializer**. The preceding section noted that statistics are collected at the host at a granularity of 20s and persisted in the database at five-minute granularity. Certain infrastructure-level entities, such as clusters, are available only at five-minute granularity, not 20-second granularity. *The Entity Metrics and Performance Provider Summary Initializer* retrieves such metadata for the entities and the statistics to be monitored.
- **Inventory/Entity Properties Monitor**. When an environment is changing, such as when virtual machines are added to the system or a disk is added to a virtual machine, it is important to track such changes and ensure statistics are collected correctly for the newly-added entities or virtual disks. The *Inventory/Entity Properties Monitor* checks for changes to the inventory at any container level and for changes to all the properties that affect the statistics metadata of the entities, such as configuration changes to virtual machines that can affect what statistics can be collected for that virtual machine.
- **Perf Query Spec Cache**. *The Perf Query Spec Cache* creates specifications for all performance statistics queries, and maintains a cache of these specifications. Any changes to the inventory are reflected in this cache. For example, if a user is retrieving CPU usage of all the hosts in a cluster, and a new host is added to a cluster, a specification is created in the cache.
- **Stats Reader**. The *Stats Reader* collects performance data from the VMware vCenter server. Users can extend it to retrieve data from various guest-specific performance tools, such as xperf[16], perfmon[15], or top[25]. Examples are provided in Section 4.3.
- **Stats Receiver**. The *Stats Receiver* is a simple interface to store the output from StatsFeeder. By default, a provided File Stats Receiver writes data to a CSV file, and a Default Stats Receiver writes data to the console. Users can customize the Stats Receiver to any desired format, including CSV, MySQL, or even a key-value store.
- **Stats Interpreter.** A sample *Stats Interpreter* is provided as a rules engine to make inferences based on simple rules. For example, the rules engine might examine the ready time for a number of virtual machines on a host and conclude that the host CPU is overcommitted. The goal is to make it easy to connect StatsFeeder to existing rules engines, rather than to provide a production-quality rules engine. Customers can develop their own rules or feed data into a commercial rules engine.
- **Stats Info Presenter**. The *Stats Info Presenter* displays the interpreted statistics data. A partner can display a line chart of statistics similar to those in the vSphere client for a custom set of statistics. For example, customers can overlay guest and host statistics to see how well they correlate.

3. Customer Requirements

To make StatsFeeder most useful for customers, it is important to analyze the main concerns customers have expressed about performance monitoring in virtualized infrastructure. Ultimately, customers have four key requirements for performance monitoring:

1. Guidance about what statistics are important
2. An easy way to specify the statistics they want to gather
3. An easy way to collect and persist these statistics in a familiar format
4. The ability to use existing analysis engines or write simple ones to detect common problems

StatsFeeder aims to provide some basic tools to address these requirements.

3.1 Determining Useful Statistics

Debugging physical infrastructure is a difficult proposition. Adding a virtualized layer increases the complexity of performance debugging. With StatsFeeder, the goal is to provide an easy mechanism for users to collect statistics that are important to them and to perform simple analysis on that data.

To simplify the process of selecting which statistics are useful, standard templates are provided that are based on VMware's experience in debugging performance-related issues in virtualized environments. These templates are not comprehensive and are not intended to help debug all performance issues. Instead, they are useful for triaging some of the common issues.

While many customers are familiar with basic metrics such as CPU used for a host or virtual machine or network bandwidth consumed, the templates include several metrics that are helpful in diagnosing virtualization-specific problems. Here is a sampling of virtualization-aware counters that are have included in the default template.

1. **CPU ready time**. Ready time is the amount of time that a virtual CPU (vCPU) was ready to be scheduled, but could not be scheduled because there was no underlying physical CPU available. High CPU ready time can indicate that underlying physical CPU resources are overcommitted, perhaps because there are too many CPU-intensive virtual machines on the same physical host. CPU ready time does not have an analogous metric for a physical machine.
2. **Memory swapIn rate**. When a host is memory-constrained, ESX sometimes must reclaim machine memory allocated to one virtual machine and give this memory to another virtual machine[35]. This is called host-level swapping. In contrast to ballooning[35], host-level swapping can cause performance degradation if it occurs frequently. The swapIn rate indicates the rate at which pages requested by a virtual machine must be swapped in by ESX because the pages were previously swapped out by ESX. High swapIn rates suggest that ESX is under memory pressure and might need more memory, or some virtual machines might need to be moved to a different host.
3. **Memory swapOut rate**. The swapOut rate is similar to the swapIn rate. It reflects the rate at which ESX is swapping out machine pages allocated to one virtual machine and allocating those machine pages to another virtual machine.
4. **CPU swapwait**. Once ESX has swapped out pages to disk for a given virtual machine, and the virtual machine is rescheduled, ESX must swap in those pages from disk before the virtual machine can resume. The swapwait metric indicates the latency required to swap in these pages before the virtual machine can run. If the swapwait values are high, memory contention could be occurring on the host.
5. **Disk maxTotalLatency**. This metric indicates the highest disk latency experienced by a host during a given sampling period. This includes the latency to remote storage such as SAN devices. In many cases, when this latency exceeds 50ms or more, it can be a signal of misconfigured or failing storage, and application performance may be impacted. While such a metric is not virtualization-specific, there are such a wide variety of metrics in vSphere that metrics such as this one often can be ignored.

3.2 Specifying Statistics to be Gathered

The standard vSphere API is designed for extreme flexibility in terms of statistics gathering. For example, the list of available statistics is not pre-determined within VMware vCenter. Instead, whenever a host or virtual machine is added to the system, it registers the statistics that it supports with the VMware vCenter server. For example, certain newer hosts might store power usage statistics, while older hosts might not collect such counters.

In order to gather data, users must specify an array of entities, such as the hosts and virtual machines to be monitored. They also must specify the desired statistics, such as those mentioned in the previous section. Specifying statistics using such an array allows maximum flexibility in tuning the speed of the query. For example, if the query is time-critical, a single array with all statistics can be used. In contrast, for higher performance, multiple queries can be issued simultaneously with disjoint subsets of the virtual machines and hosts in order to make it complete faster. While this is useful from a performance perspective, it makes programming more difficult. Users must enumerate explicitly every entity for which statistics are desired. For example, suppose an environment contains 1,000 virtual machines. Then a user must retrieve the ID of each virtual machine and construct an array with each virtual machine's ID and the desired counter. If all virtual machines are in a single datacenter, there is no easy way to specify a 'container' attribute to allow users to get statistics for all virtual machines within the datacenter simply by specifying the datacenter name. Moreover, users must constantly monitor the hosts and virtual machines in an environment to see if new ones have been added or deleted. If so, those virtual machines and hosts must be added or removed from the statistics query.

To make it easy for end users to specify the entities for which metrics are to be collected, users can specify the entities and metrics in a simple XML format. StatsFeeder is invoked with this XML file as input. The format allows users to specify a container-based

list of entities. For example, users can specify certain statistics for all virtual machines in a datacenter, without enumerating virtual machines by ID. If users want to collect the CPU used time for all virtual machines within a cluster named "myDatacenter", the configuration file looks like the following:

```
<container>
        <name>myDatacenter</name>

        ...

</container>
<childType>VirtualMachine</childType>
<statsList>
        <stat>cpu.used.summation</stat>

</statsList>
```

If users want statistics for CPU used time and CPU ready time for virtual machines, and the network bandwidth consumed for each host, the configuration file is easy to modify:

```
<childType>VirtualMachine</childType>
<statsList>
        <stat>cpu.used.summation</stat>
        <stat>cpu.ready.summation</stat>
</statsList>

...

<childType>HostSystem</childType>
<statsList>
        <stat>net.usage.average</stat>
</statsList>
```

StatsFeeder takes the specified container name and generates a list of entities within the container. This list is used to create a set of parallel queries to collect the data, taking advantage of the flexibility of the API while hiding the complexity from end users. StatsFeeder also allows users to indicate specific entities to be monitored, such as virtual machine VM-1 or host Host-X, rather than requiring a container to be specified.

3.3 Collecting and Persisting Data

StatsFeeder handles the collection of data and allows users to write a custom receiver to persist the data. A default CSV receiver writes the statistics data to a file in a comma-separated value format. The CSV file can be viewed directly or additionally processed by scripts that take the data and insert it into a database for more efficient long-term storage.

To implement a new receiver, users simply implement a single receiveStats() method. Within this class, the retrieved performance data is read and sent to whatever persistence framework is desired, whether it be a file, a message bus, a database, or even a distributed hash table (DHT).

A number of different receivers were implemented during testing. For example, a MySQL receiver was created that takes statistics data and persists it directly into a MySQL database. In addition, a receiver that uses the Redis distributed hash table[21], in which data is written to an in-memory DHT, was developed. The DHT approach offers an advantage: some of the publisher/subscriber functionality of Redis can be leveraged. For example, each statistic can be published on a separate stream, and end-user applications can subscribe to specific statistics for specific entities. Redis also provides capabilities to expire data automatically beyond a certain time frame, similar to a round-robin database.

3.4 Analysis

There are a wide variety of tools available to analyze performance data and perform advanced analytics for anomaly detection[26] [31]. A goal for StatsFeeder was to allow users to employ very simple, extensible heuristics on data, rather than trying to implement sophisticated analytics. For example, some end users simply are interested in knowing when CPU usage for a given host goes above 70 percent, and have special corporate policies to provision new physical hosts if the utilization exceeds this value. Other customers want to automate troubleshooting by taking standard VMware best practices and turning them into automated detection code.

Analysis using StatsFeeder is straightforward. By default, metric values are written to a CSV file. As a proof of concept, values were instead written to a MySQL database, and a simple analyzer was implemented that runs periodic queries against the database and looks for CPU ready time and disk latency issues. The query ranks virtual machines and hosts based on these values. By looking at this sorted list, administrators can determine which virtual machines or hosts should be investigated.

In addition, simple aggregations of ready time are performed to determine if a given host has a high proportion of virtual machines with high ready times—in this case, it might be advisable to move some virtual machines off the host. For disk latency issues, the tool checks to see if any host's maximum disk latency exceeds a predefined threshold value (in this case, 20ms), and creates a sorted list of such hosts. A simple alert mechanism was implemented on top of these analyzers to send email messages with these charts as the body of the message.

4. Use Cases

This section discusses various use cases that were prototyped using StatsFeeder.

4.1 Aggregated Esxtop

Esxtop[27] is a useful ESX performance debugging tool. Users log in to an individual ESX host and invoke esxtop. Similar to the UNIX top utility, esxtop shows resource utilization for CPU, disk, memory, and network. Further, it breaks down resource usage by virtual machine as well as by device (in the case of disk and network). Esxtop includes a batch mode that allows users to archive statistics to a file on the host and read the data in offline.

One disadvantage of esxtop is that it collects data for a single host, while often it is desirable to examine multiple metrics across multiple hosts at the same time. In addition, using esxtop requires logging in to each ESX host individually. This can be onerous for large environments or potentially introduce security concerns, since many enterprise environments are locked down and restrict logins to ESX hosts to a very small set of users.

StatsFeeder can perform the batch mode functions of esxtop, and aggregate data across multiple hosts. Moreover, StatsFeeder requires login privileges to the VMware vCenter management server, rather than login privileges to each host, easing security concerns. Using StatsFeeder, statistics data can be obtained across multiple hosts and stored in a local database or a file. The data can be mined and written to a Web page so that it is aggregated in one place as opposed to distributed across multiple screens. Such a collector was implemented. Figure 3 shows an output sample.

One final advantage of using StatsFeeder for data collection is that users can customize the statistics they want to see, as described in section 3.2. For example, users can create a graph similar to the perfmon interface with multiple sets of data on the same set of axes (such as network bandwidth and CPU usage) to help see correlations between statistics.

VM Name	CPU Used	CPU Ready	Memory Ballooned	Memory Swapped	Network Data Receive Rate	Network Data Transmit Rate	Disk Read Rate	Disk Write Rate
cpuBurn-smp-clone2	1364	781	0	0	0	0	0	9
cpuBurn-smp-clone3	1348	749	0	0	0	0	0	18
cpuBurn-smp-clone4	1055	358	0	0	0	0	0	19

Figure 3: Aggregated esxtop—a statistics display similar to esxtop that shows multiple metrics per virtual machine in a single screen.

4.2 Statistics and Events Overlays: Is DRS Working?

One of the interesting value propositions of vSphere is Distributed Resource Scheduling (DRS)[32], in which virtual machines are distributed across hosts and migrated as needed using VMware® VMotion™[18] to try to achieve good load-balancing and avoid hot spots in resource usage. Customers often are curious about the effectiveness of DRS. They want to know if migrations are happening, whether performance is improving, and if utilizations really are balanced. The current vSphere platform allows users to see the frequency of DRS-initiated migrations. Currently there is no utility to overlay these migrations with the resulting changes in resource utilization on the source and destination hosts.

Using StatsFeeder to collect CPU usage for a source and destination host, and combining this data with a custom script that grabs events from a VMware vCenter server, events can be overlaid on top of statistics charts to demonstrate the impact of DRS. Figure 4 shows VMware VMotion events along with CPU utilization for a pair of hosts in a cluster, illustrating the impact of VMware VMotion on CPU utilization. In this situation, a workload is run that saturates one host and causes virtual machines to be moved by VMotion to the other host, reducing CPU utilization on the first host and increasing CPU usage on the second host. Displaying such overlay data is helpful to show customers that DRS is working as expected. One can imagine

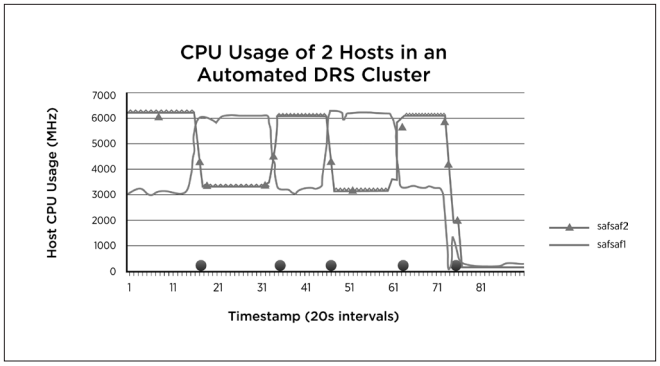

Figure 4. Does DRS Work? This figure shows how CPU usage for two hosts in a cluster varies due to DRS-initiated VMotion migrations. The dots on the x-axis represent VMotion events. As expected, when a virtual machine is moved from one host to another, the CPU utilization of the destination host increases and the utilization of the source host decreases.

grabbing application data with another custom script and overlaying it on such a chart to show a more direct measurement of the effectiveness of DRS.

4.3 Getting Data from within the Guest

The VMware vSphere Web Services SDK[29] includes the ability to query data from within a guest operating system. Using this API, data can be collected from the guest and correlated to host-level or virtual machine-level metrics. In addition, profiling tools can be executed within the guest in response to various conditions.

4.3.1 Running Profilers within a Guest: xPerf

The xPerf[1] utility in Microsoft Windows 2008 provides a profiler that shows CPU used time and callstack information for Windows executables. Often it is useful to run this tool when the CPU is saturated to see where cycles are being spent. Using the StatsFeeder infrastructure and leveraging its extensibility, such automated data collection can be performed.

First, a special StatsReader is implemented that collects statistics about per-process CPU usage within a guest. Next, StatsFeeder is configured to collect data from the custom StatsReader as opposed to the standard VMware vCenter server. A simple Stats Interpreter module is implemented that checks this CPU usage. If the CPU usage exceeds a predefined threshold for a prolonged period of time, the Stats Interpreter can use the in-guest APIs to start xperf automatically. This xperf profile can be dumped to a file, read out using the same in-guest API, and emailed to a developer for offline analysis.

4.3.2 Active Memory Comparison

One of the difficulties in a virtualized environment is that for common troubleshooting, there is no 1:1 correlation between certain metrics within a guest inside the virtual machine, virtual machine-level metrics, and host-level metrics. For example, in physical machines running Windows, end users might determine that the machine needs more memory by looking at active memory in the TaskManager display and comparing it to the memory size of the machine. If the active memory is close to the memory size, the machine might need more memory.

When these machines are virtualized, end users still can look inside the virtual machine at the active memory counters to see if the virtual machine needs more memory. Many customers often assume that the active memory of the virtual machine as measured by ESX is the same as the active memory that would be seen by examining TaskManager inside the virtual machine. The active memory of the virtual machine is not, however, the same as the active memory of applications inside the guest operating system for several reasons.

For example, the active memory of a virtual machine is approximated by examining the activity of a subset of virtual machine pages, and using statistical models to extrapolate to the overall pages used by the virtual machine. An alternative approach to determining whether a virtual machine needs more memory without looking inside the guest is to consider the virtual machine's *consumed* memory. It describes the amount of host machine memory that is mapped to a given virtual machine[8], taking into account content-based page sharing.[35] This metric is an inaccurate proxy for determining if a virtual machine needs more memory: pages can be mapped and remain unused, so a high number does not indicate that a virtual machine is using all of the memory allocated to it. A more complete discussion of in-guest virtual machine and host-level metrics is provided by Colbert and Guo[8].

StatsFeeder can be used to help overlay various statistics and demonstrate the differences between in-guest statistics and host-level statistics, as shown in Figure 5. First, a custom StatsReader is created for retrieving active memory within the guest using in-guest APIs. This StatsReader is used in addition to the default StatsReader that obtains active memory and consumed memory statistics for the host from VMware vCenter. A Stats Interpreter module is written that overlays these statistics on the same graph to demonstrate the differences.

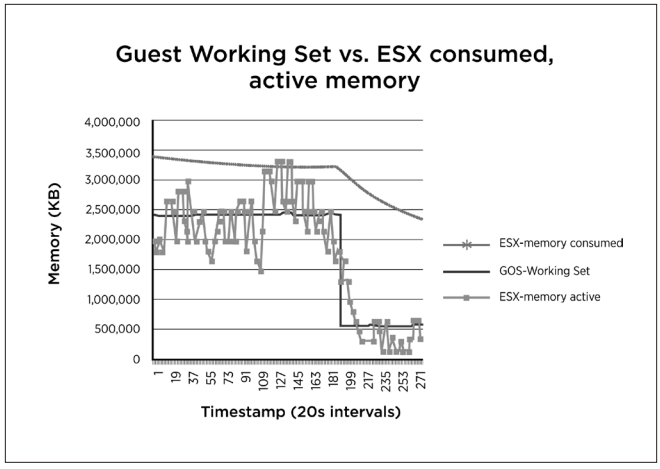

Figure 5. Guest operating system (GOS) working set versus active memory as measured by ESX (jittery line) versus consumed memory (top line). While the trends are similar, ESX active memory is not equivalent to the GOS working set.

Note that guest active memory is quite steady, while the active memory for the virtual machine as measured by ESX jitters quite a bit. In fact, the two values are different by several hundred MB at various times. Moreover, the memory consumed by the virtual machine on the host (mapped pages minus pages saved due to sharing) is quite a bit larger. The basic trends in terms of usage are similar, suggesting that the active memory for the virtual machine as measured by ESX in some cases can be an approximation for the working set of memory within the guest.

4.3.3 Application Metrics by Resource Pool

One final use case that was implemented involves statistics comparisons that are not available in the current vSphere platform. For example, customers can set up resource pools in order to provide controlled compute capability or memory to different organizations within a company. They might want to see if their resource settings are giving them the performance they expect. Typically, this would involve comparing application behavior across virtual machines in the resource pool. Such a view currently is not available in the vSphere UI. Such comparisons can be performed using the extensibility of StatsFeeder.

Figure 6 compares the time to run 10 iterations of a compilation benchmark for two different resource pools with different resource reservations and limits[32]. RP1 has a 3GHz CPU limit and RP2 has a 6GHz CPU limit. As Figure 6 shows, the average compilation time is almost twice as long for RP1 as for RP2. This is to be expected for a CPU-bound benchmark in which the CPU is throttled artificially. This was implemented in StatsFeeder in a manner similar to that used for the active memory comparison in section 4.3.2. First, the benchmark was modified to record the compilation times in a file on each virtual machine. A custom StatsReader was created that reads those files out using the in-guest APIs, as described earlier. A custom Stats Intepreter was used to examine the compilation times and create a bar chart for comparison. Such a comparison can help in determining whether to adjust resource pool settings to provide better performance for an application.

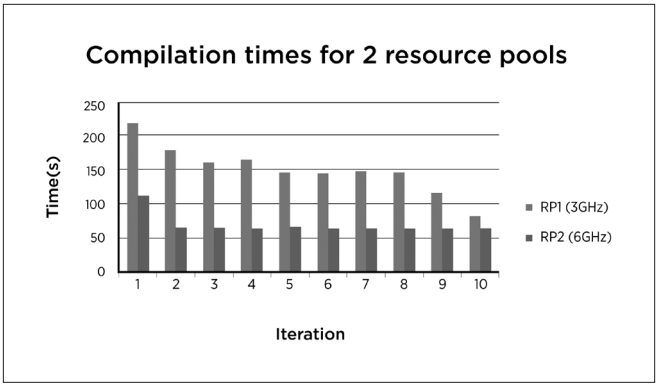

Figure 6. Compilation times for resource pools with different limit settings. StatsFeeder can compare application-level metrics across resource pools. Here, RP2 (6GHz limit) compiles 100 percent faster than RP1 (3GHz limit).

4.4 Other Use Cases

One can imagine extending the functionality of StatsFeeder in a number of other ways.

- Showing data aggregated by virtual machine type. For example, it is possible to show CPU usage for all Windows virtual machines or all Microsoft Exchange virtual machines on the same chart.
- Sending data to a performance database with a schema that is optimized on a per-customer basis.

- Getting Common Information Model (CIM)[10] data to monitor hardware health, such as examining the temperature on a motherboard to see if it exceeds a healthy threshold.
- Collecting vSCSI[28] statistics data from ESX and correlating them with application I/O data.
- Coupling this tool with a more sophisticated rules engine for diagnostic purposes.

5. Performance Evaluation

StatsFeeder was designed to make it easier for customers to retrieve statistics they want at a fine granularity, and help them to monitor virtualized infrastructure. The performance evaluation focused on three aspects of StatsFeeder design. First, the need for high-performance statistics collection was illustrated by comparing a naïve approach against the more tuned approach that StatsFeeder employs. Second, consideration was given to whether StatsFeeder meets the needs of larger environments by examining StatsFeeder scalability with the number of entities being monitored. Finally, the scalability of StatsFeeder was examined with the number of statistics collected per entity to ensure it is capable of getting a rich set of statistics that can help end users with performance troubleshooting.

5.1 Experimental Setup

The experimental setup consisted of a single VMware vCenter server connected to a varying number of virtual machines (1 to over 3000). The virtual machines were spread across up to 160 physically-distributed ESX hosts. The VMware vCenter server contained a four-vCPU virtual machine with 8GB of RAM. The StatsFeeder client consisted of a four-way Linux virtual machine with four vCPUs and 4GB of RAM. A 100Mbps network link was connected between VMware vCenter and the ESX hosts, and between VMware vCenter and StatsFeeder. StatsFeeder accumulated statistics data from VMware vCenter and persisted the data to a Redis DHT.

Tests experimented with a local Redis instance (in the same virtual machine as the StatsFeeder client), as well as a remote instance (a different virtual machine on a different ESX host). In the remote case, a 100Mbps link ran between the StatsFeeder client and the Redis DHT instance—the testing team was required to use an existing installation and could not upgrade the network. The results described in this paper are for a local Redis instance.

5.2 Baseline Comparison

For a baseline comparison, consideration was given to the task of collecting a single statistic from a virtual machine or group of virtual machines. Two approaches were compared. The first approach used the VMware vSphere PowerCLI™ [33] toolkit with built-in primitives, called *cmdlets*, for statistics collection. The second approach used StatsFeeder. While the vSphere PowerCLI toolkit is easy to use, the cmdlets are built for generality rather than for speed. This comparison highlights the importance of highly tuned statistics collection.

Table 1 shows the results for the vSphere PowerCLI and StatsFeeder approaches for a varying numbers of virtual machines in an environment. The vSphere PowerCLI implementation requires two lines of code, while configuring StatsFeeder for collection requires editing two lines of a configuration file. For a small numbers of virtual machines, performance is comparable and acceptable, in the range of milliseconds. As the number of virtual machines increases, the scripted approach scales very poorly, taking over 165 seconds for 441 virtual machines versus only 11 seconds for StatsFeeder, a 10x difference in performance.

StatsFeeder is able to achieve such performance gains using three primary optimizations. First, StatsFeeder uses aggressive multithreading when issuing performance queries, while the vSphere PowerCLI script issues queries in a serialized manner. Second, StatsFeeder leverages a more compact data transfer format than the format used with the built-in vSphere PowerCLI primitives. This data transfer format can reduce message size up to ten times. Finally, StatsFeeder caches and reuses certain metadata that is common across all virtual machines, while the default vSphere PowerCLI toolkit retrieves this data for each virtual machine and does not cache it.

NUMBER OF VMS	POWERSHELL (SINGLE-THREADED)	STATSFEEDER (JAVA, MULTITHREADED)
1	229 milliseconds	127 milliseconds
5	2.6 seconds	302 milliseconds
21	9.2 seconds	782 milliseconds
441	165 seconds	11 seconds

Table 1: Naive statistics collection versus highly tuned statistics collection. A simple shell script (vSphere PowerCLI) can take 10x longer than the highly optimized collector in StatsFeeder.

This baseline exercise is not intended to make an unfavorable comparison between vSphere PowerCLI and Java. In fact, each of the features employed in StatsFeeder (multithreading, compact data formats, metadata caching) is available in vSphere PowerCLI. Using each one, however, requires system administrators to understand the PerformanceManager API in detail and apply key optimizations—in addition to implementing more complicated features, such as multithreaded queries, rather than using the extremely simple built-in cmdlets. While the cmdlets can be optimized, doing so requires significant changes to the underlying vSphere PowerCLI runtime system to reorganize queries and cache data. Such changes would be somewhat different from the typical stateless programming model that shell scripts assume. Instead, StatsFeeder offers a highly optimized solution that is simple to use.

5.3 Statistics Scalability

The previous section compared StatsFeeder to a baseline vSphere PowerCLI implementation for varying numbers of virtual machines. This section studies the scalability of StatsFeeder with respect to the number of statistics collected per virtual machine or host. A typical large, virtualized datacenter can comprise 3,000 virtual machines. Many customers require statistics at 20s granularity to be collected every five minutes and published to a monitoring dashboard. The goal is to accommodate this use case and ensure that a reasonable number of statistics can be collected per virtual machine for up to 3,000 virtual machines in under five minutes.

METRICS PER VIRTUAL MACHINES	LATENCY FOR COLLECTION (SECONDS)				
	441 Virtual Machines	882 Virtual Machines	1762 Virtual Machines	2433 Virtual Machines	3190 Virtual Machines
1	3.4	11	19	29	93
5	3.5	9.2	20	23	116
10	5.1	14	24	30	91

Table 2: For a given number of metrics, StatsFeeder scales roughly linearly with the number of VMs. The jump in latency from 2433 VMs to 3190 VMs is due to anomalies in host loads. The latency for the largest environments never exceeds the performance goal of five minutes. (The data for one metric is different from Table 1 due to the use of different environments.)

Table 2 shows the results obtained as the number of virtual machines was varied from approximately 400 to over 3,000, and as the number of metrics collected per virtual machine was varied from one to five to 10. The data for one metric was used a baseline, while five and 10 are more typical values for the number of metrics collected per virtual machine (including some of the metrics listed in the default template described in section 3.1).

Each collection obtained the last 45 values for a given statistic. For real-time statistics that are refreshed every 20s, 45 values represents the last 15 minutes of statistics for the virtual machine. This suggests that StatsFeeder could run every 15 minutes over the entire infrastructure to obtain the desired statistics. For a given number of threads (10 in this case), near linear scaling is expected as the number of virtual machines is increased, since the overall amount of data being collected is increasing nearly linearly. Similarly, as the number of metrics being collected is increased, an approximately linear increase in collection time is expected.

As Table 2 shows, the results are quite different from expectations. When increasing the number of virtual machines, latency increases in a near linear manner as expected. Yet there is a large jump in latency from 2,433 virtual machines to 3,190 virtual machines. The main reason for this anomaly is that the collection time on a per virtual machine basis is not constant. The time to collect statistics for virtual machines can vary dramatically from 20ms to 5,000ms, depending on ongoing network activities or the load on the underlying physical host running the virtual machine. During testing a set of hosts consistently were taking an unusually long time to answer the real-time queries forwarded from VMware vCenter. This tied up some of the StatsFeeder threads and prevented other queries from being issued. More long-running queries occurred with 3,190 virtual machines than with 2,433 virtual machines. These few queries resulted in a longer overall collection time.

Another unexpected result is that the latency for collection is largely independent of the number of metrics. Fixed overhead dominates for modest numbers of metrics. On a given host, in the absence of an anomalous load, the collection time does not vary significantly for one to 10 metrics. A null statistics call—a call to retrieve zero statistics—takes approximately 2ms per host. This is the fixed cost of collecting statistics for a single virtual machine on a host. Essentially, fixed overhead dominates the total time on the host for one to 10 metrics. The total time to collect one metric is approximately 3 ms. Collecting 10 metrics takes approximately 5ms. Therefore, collecting the metrics only accounts for 1ms to 3ms of overhead for one to 10 metrics.

The total latency for collection consists of collection at the host, transfer to VMware vCenter, deserialization at VMware vCenter, transfer to StatsFeeder, and persistence to Redis, with the bulk of the latency introduced by serialization and deserialization within VMware vCenter. For a modest number of metrics, the fixed cost of serialization and persistence at each stage is on par with the cost per volume of data. Consequently, the total time does not change much with the number of metrics. When collecting a significantly larger number of metrics per virtual machine (~150), the overall latency on a host increases to approximately 30ms, and the cost per volume of data exceeds the fixed cost of serialization and deserialization. Also note that because the results are persisted to in an in-memory DHT, the persistence portion is quite small. Persisting to disk instead would potentially incur much larger overheads.

A final anomaly can be seen in the data. For 882 virtual machines, the collection time for metrics is longer than that for 10 metrics. Similar behavior occurs for 3,190 virtual machines. In the case of 2,433 virtual machines, collection of one metric takes longer than collecting five metrics. In each of these cases, anomalous long-running queries influence the results. When the anomalous queries are removed, the results are in line with expectations.

5.4 Resource Utilization versus Performance

Statistics collection in vSphere is extremely parallelizable, and StatsFeeder issues multiple concurrent statistics requests to the VMware vCenter server in order to maximize this parallelism. Each collection requires CPU utilization on the VMware vCenter server to deserialize the data from the hosts and send it to StatsFeeder. Since the primary function of VMware vCenter is virtualization management, rather than statistics collection, StatsFeeder must throttle the number of requests to avoid overwhelming the VMware vCenter server CPU.

For all but the largest environment (3,190 virtual machines), it was possible to achieve acceptable collection latency with a modest number of outstanding queries from StatsFeeder to the VMware vCenter server (10 threads). Because the latency to collect statistics varies widely among virtual machines, the server spends a lot of time waiting for requests to return from hosts. In fact, its average CPU utilization was approximately 15 percent for the duration of the statistics collection. In addition, the StatsFeeder client and its local Redis instance never exceed 50 percent CPU utilization. Without resource saturation, it is reasonable to try to increase the number of threads and improve collection latency.

Table 3 shows the results of increasing the number of StatsFeeder collection threads from 10 to 20. The collection times are significantly better when using more threads. Moreover, the average CPU usage of the VMware vCenter server increases only marginally, from 15 percent with 10 threads to 20 percent with 20 threads. With more outstanding requests from StatsFeeder, the impact of a single long-running query is reduced. The CPU usage of StatsFeeder and the local Redis instance show only modest increases with the number of threads, from approximately 45 percent to 55 percent CPU utilization.

METRICS PER VIRTUAL MACHINE	LATENCY FOR COLLECTION (SECONDS)				
	441 Virtual Machines	882 Virtual Machines	1762 Virtual Machines	2433 Virtual Machines	3190 Virtual Machines
1	3.0	8.3	16	19	69
5	3.3	8.1	15	18	66
10	4.6	8.6	14	19	63

Table 3: Latency for statistics collection with 20 StatsFeeder threads. The additional threads improve performance significantly by reducing the impact of long-running queries.

6. Related Work

First and foremost, StatsFeeder was intended to fill a specific role: easier collection of statistics for a virtualized infrastructure. While StatsFeeder was not designed intentionally for limited scale, the intended use case for an individual StatsFeeder instance is a typical deployment of approximately 300 physical ESX hosts and 3,000 virtual machines. By limiting the scope to data relevant to virtualization, it is possible to be more precise in what is collected and provide actionable information to end users. In addition, StatsFeeder can be extended easily to allow end users to retrieve and store data in any desired format, whether it be a file, a database, or a DHT.

A number of efforts are underway to help ease the monitoring and troubleshooting process in distributed environments. StatsFeeder can leverage some of the techniques they employ.

- **Apache Chukwa**, a project[6] exploring large-scale log collection and analysis. Built upon Apache Hadoop[2] and MapReduce[9], Apache Chukwa enables users to perform analytics using Apache Pig[4]. Initially, Apache Chukwa was not intended for real-time monitoring and management, but for out-of-band detection of issues.
- **Facebook's Scribe server**[11], a scalable approach for aggregation and analysis of log data. It employs Hadoop for storing data and Hive[3] for performing data analytics.
- **Cloudera Flume**[7], a Hadoop implementation that enables the efficient processing of stream-based data such as event logs. It is not specifically tailored to time-series data such as performance metrics.
- **Splunk**[24], a tool for analyzing log files and semi-structured data. Splunk indexes large amounts of data in real time and performs analytics on the data.

In contrast to Apache Chukwa, monitoring systems such as Nagios[17] and Ganglia[14] are designed for real-time collection of a small number of metrics per node. Ganglia is most similar to the work presented in this paper, as it focuses on the efficient collection of time-series data across an infrastructure. Ganglia requires users to implement a daemon that runs on each monitored node. This daemon collects the statistics that are relevant to end users.

The approach described in this paper uses a similar mechanism by leveraging agents that exist on ESX hosts. In addition, it tries to tailor metric collection to statistics that are relevant in virtualized environments. Ganglia could be coupled easily to StatsFeeder as an additional source of application-specific input data, such as application throughput metrics. Similar to Ganglia, Nagios works well with time-series data. It also can be used with other types of data, such as network or host service health data, or other semi-structured data.

Finally, various commercial tools are available for large-scale statistics collection, including VMware® vCenter™ Operations Manager™[31], VMware Hyperic®[30], and IBM Tivoli[13]. Hyperic is similar to StatsFeeder in that it provides an API for collecting statistics per host, per virtual machine, and within the guest operating system. One of the primary differences between Hyperic and StatsFeeder is that Hyperic requires an agent to be installed in the guest or on the host in order to obtain data. StatsFeeder utilizes existing management agents on a host to obtain statistics for a host or its virtual machines. VMware vCenter Operations Manager offers a rich set of analytics for performing anomaly detection and signal alerts. Currently, it does not provide an API to store this data in a database of the user's choosing.

7. Conclusions And Future Work

Typical VMware customers can perform basic performance troubleshooting in a physical environment. They are familiar with which metrics are important, and have tools to collect and analyze these metrics. When a virtualization layer is deployed, it creates an additional layer of complexity in the process of finding performance issues. Even simple metrics such as CPU usage and memory allocation have a different meaning when applied to a virtual machine as opposed to a physical machine. Ultimately, customers are looking for guidance in determining which metrics are important to monitor for assessing the health of their infrastructure—and they are asking for tools to help collect and analyze these metrics. While toolkits are available for collecting such data[33], it can be challenging to write collection scripts that scale with inventory size and deliver adequate performance.

This paper presented StatsFeeder, a prototype application that helps with the collection and analysis of data in a virtualized environment. StatsFeeder provides users with a simple way to specify the entities to monitor and the statistics to collect. StatsFeeder includes a sample rules engine with some common virtualization-specific, performance-troubleshooting rules built in. Users can create more complicated rules or connect to existing rules engines.

A wide variety of uses cases were presented that validate the approach, ranging from overlaying events and statistics to comparing guest and host data. The collection infrastructure scales with the size of the environment. For additional performance, users can run multiple instances of StatsFeeder and store the output in a common database or distributed hash table to allow comparisons across virtualized environments. For example, it might be interesting to collect power data for hosts in different virtualized datacenters to see which ones are most efficient.

There are a number of avenues for future work. The authors plan to experiment with separating the StatsFeeder client and the Redis DHT onto separate machines and sharding the Redis DHT to achieve high throughput for persistence. The aim is to increase the default number

of metrics that can be collected at a particular collection interval. On the data collection front, the plan is to extend the StatsFeeder model to include cloud-level metrics, such as statistics subdivided by cloud tenant or virtual datacenter[34].

Another plan is to couple StatsFeeder to various stream-processing engines, such as Cloudera Flume, and extend StatsFeeder to include event and alarm information so that users do not need to write external scripts to overlay statistics with event data. Moreover, customers could send this aggregated data to more advanced analytics engines[31]. Extending the rules engines for more advanced internal debugging is straightforward. By collecting such detailed data in an automated manner, StatsFeeder is useful for customers as well as internal debugging of products before release.

8. Acknowledgments

The authors would like to thank our shepherd, Banit Agrawal, for many helpful comments on earlier drafts of this paper.

9. References

[1] Amazon. Amazon EC2. http://aws.amazon.com/ec2/

[2] Apache. Hadoop. http://hadoop.apache.org/

[3] Apache. Hive. http://hive.apache.org/

[4] Apache. Pig. http://pig.apache.org/

[5] Barroso, L., et al. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers, 2009.

[6] Boulon, J., et al. Chukwa, a large-scale monitoring system. In Proceedings of CCA 08. October 2008.

[7] Cloudera. Cloudera Flume. https://github.com/cloudera/flume/wiki

[8] Colbert, K., and Guo, F. Understanding "Host" and "Guest" Memory Usage and Related Memory Management Concepts. VMworld '09. San Francisco, CA.

[9] Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, vol. 51, no. 1 (2008), pp. 107-113.

[10] DMTF. Common Information Model. http://dmtf.org/standards/cim

[11] Facebook. Facebook Scribe. https://github.com/facebook/scribe

[12] HP. Lights-out management. http://h18000.www1.hp.com/products/servers/management/ilo/

[13] IBM. IBM Tivoli. http://www.ibm.com/developerworks/tivoli/

[14] Massie, M., et al. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. In Parallel Computing. Volume 30, Issue 7, pp 817-840, 2004.

[15] Microsoft. Perfmon. http://technet.microsoft.com/en-us/library/bb490957.aspx

[16] Microsoft. xPerf. Windows Performance Analysis Developer Center. http://msdn.microsoft.com/en-us/performance/default.aspx

[17] Nagios Enterprises. Nagios. http://www.nagios.org/

[18] Nelson, M. et al. Fast Transparent Migration for Virtual Machines. In Proceedings of USENIX '05 (Anaheim, CA, April 10-15, 2005). 391-394.

[19] Oetiker, T. MRTG. http://oss.oetiker.ch/mrtg/

[20] Oracle. MySQL. http://www.mysql.com/

[21] Redis. Redis key-value store. http://redis.io/

[22] Soundararajan, V., and Anderson, J. The Impact of Management Operations on the Virtualized Datacenter. In Proceedings of ISCA 2010 (San Malo, France, June 19-23, 2010). 326-337.

[23] Soundararajan, V., and Govil, K. Challenges in Building Scalable Virtualized Datacenter Management. In ACM SIGOPS Operating Systems Review. Volume 44, Issue 4. December 2010. 95-102.

[24] Splunk. www.splunk.com.

[25] Unix top utility. http://www.unix.com/man-page/Linux/1/top/

[26] VKernel. VKernel vOperations Suite. http://www.vkernel.com/

[27] VMware. Interpreting esxtop statistics. http://communities.vmware.com/docs/DOC-9279

[28] VMware. Using vscsiStats for Storage Performance Analysis. http://communities.vmware.com/docs/DOC-10095/version/9

[29] VMware. VMware API Reference Documentation. https://www.vmware.com/support/pubs/sdk_pubs.html

[30] VMware. VMware Hyperic HQ. http://www.hyperic.com/products/vmware-monitoring

[31] VMware. VMware vCenter Operations Manager. http://www.vmware.com/products/vcenter-operations/overview.html

[32] VMware. VMware vSphere. http://www.vmware.com/products/vsphere/

[33] VMware. VMware vSphere PowerCLI. https://www.vmware.com/support/developer/PowerCLI/index.html

[34] VMware. VMware vCloud Director. http://www.vmware.com/products/vcloud-director/overview.html

[35] Waldspurger, Carl. Memory Resource Management in VMware ESX Server. In Proceedings of OSDI '02. (Boston, MA, December 9-11, 2002).

VMware Distributed Resource Management: Design, Implementation, and Lessons Learned

Ajay Gulati
VMware, Inc.
agulati@vmware.com

Anne Holler
VMware, Inc.
anne@vmware.com

Minwen Ji
Facebook, Inc.
mji@fb.com

Ganesha Shanmuganathan
VMware, Inc.
sganesh@vmware.com

Carl Waldspurger
carl@waldspurger.org

Xiaoyun Zhu
VMware, Inc.
xzhu@vmware.com

Abstract

Automated management of physical resources is critical for reducing the operational costs of virtualized environments. An effective resource-management solution must provide performance isolation among virtual machines (VMs), handle resource fragmentation across physical hosts and optimize scheduling for multiple resources. It must also utilize the underlying hardware infrastructure efficiently. In this paper, we present the design and implementation of two such management solutions: DRS and DPM. We also highlight some key lessons learned from production customer deployments over a period of more than five years.

VMware's Distributed Resource Scheduler (DRS) manages the allocation of physical resources to a set of virtual machines deployed in a cluster of hosts, each running the VMware ESX hypervisor. DRS maps VMs to hosts and performs intelligent load balancing in order to improve performance and to enforce both user-specified policies and system-level constraints. Using a variety of experiments, augmented with simulation results, we show that DRS significantly improves the overall performance of VMs running in a cluster. DRS also supports a "what-if" mode, making it possible to evaluate the impact of changes in workloads or cluster configuration.

VMware's Distributed Power Management (DPM) extends DRS with the ability to reduce power consumption by consolidating VMs onto fewer hosts. DPM recommends evacuating and powering off hosts when CPU and memory resources are lightly utilized. It recommends powering on hosts appropriately as demand increases, or as required to satisfy resource-management policies and constraints. Our extensive evaluation shows that in clusters with non-trivial periods of lowered demand, DPM reduces server power consumption significantly.

Categories and Subject Descriptors

C.4 **[Performance of Systems]**: Modeling techniques;
C.4 **[Performance of Systems]**: Measurement techniques;
C.4 **[Performance of Systems]**: Performance attributes;
D.4.8 **[Operating Systems]**: Performance—*Modeling and prediction*;
D.4.8 **[Operating Systems]**: Performance—*Measurements*;
D.4.8 **[Operating Systems]**: Performance—*Operational analysis*

General Terms

Algorithms, Design, Experimentation, Management, Measurement, Performance

Keywords

VM, Virtualization, Resource Management, Scheduling, Cluster, Hosts, Load Balancing, Power Management

1 Introduction

Initially, the rapid adoption of virtualization was fueled by significant cost savings resulting from server consolidation. Running several virtual machines (VMs) on a single physical host improved hardware utilization, allowing administrators to "do more with less" and reduce capital expenses. Later, more advanced VM capabilities such as cloning, template-based deployment, checkpointing, and live migration [43] of running VMs led to more agile IT infrastructures. As a result, it became much easier to create and manage virtual machines.

The ease of deploying workloads in VMs is leading to increasingly large VM installations. Moreover, hardware technology trends continue to produce more powerful servers with higher core counts and increased memory density, causing consolidation ratios to rise. However, the operational expense of managing VMs now represents a significant fraction of overall costs for datacenters using virtualization. Ideally, the complexity of managing a virtualized environment should also benefit from consolidation, scaling with the number of hosts, rather than the number of VMs. Otherwise, managing a virtual infrastructure would be as hard — or arguably harder, due to sharing and contention — as managing a physical environment, where each application runs on its own dedicated hardware.

In practice, we observed that a large fraction of the operational costs in a virtualized environment were related to the inherent complexity of determining good VM-to-host mappings, and deciding when to use vMotion [8], VMware's live migration technology, to rebalance load by changing those mappings. The difficulty of this problem is exacerbated by the fragmentation of resources across many physical hosts and the need to balance the utilization of multiple resources (including CPU and memory) simultaneously.

We also found that administrators needed a reliable way to specify resource-management policies. In consolidated environments, aggregate demand can often exceed the supply of physical resources. Administrators need expressive resource controls to prioritize VMs of varying importance, in order to isolate and control the performance of diverse workloads competing for the same physical hardware.

We designed *DRS* (for *distributed resource scheduler*), to help reduce the operational complexity of running a virtualized datacenter. DRS enables managing a cluster containing many potentially-heterogeneous hosts as if it were a single pool of resources. In particular, DRS provides several key capabilities:

- A *cluster* abstraction for managing a collection of hosts as a single aggregate entity, with the combined processing and memory resources of its constituent hosts.
- A powerful *resource pool* abstraction, which supports hierarchical resource management among both VMs and groups of VMs. At each level in the hierarchy, DRS provides a rich set of controls for flexibly expressing policies that manage resource contention.
- Automatic *initial placement*, assigning a VM to a specific host within the cluster when it is powered on.
- Dynamic *load balancing* of both CPU and memory resources across hosts in response to dynamic fluctuations in VM demands, as well as changes to the physical infrastructure.
- Custom *rules* to constrain VM placement, including affinity and anti-affinity rules both among VMs and between VMs and hosts.
- A host *maintenance mode* for hardware changes or software upgrades, which evacuates running VMs from one host to other hosts in the cluster.

In this paper, we discuss the design and implementation of DRS and a related technology called *DPM* (for *dynamic power management*). DRS provides automated resource-management capabilities for a cluster of hosts. DPM extends DRS with automated power management, powering off hosts (placing them in standby) during periods of low utilization, and powering them back on when needed [9].

Our experimental evaluation demonstrates that DRS is able to meet resource-management goals while improving the utilization of underlying hardware resources. We also show that DPM can save significant power in large configurations with many hosts and VMs. Both of these features have been shipping as VMware products for more than five years. They have been used by thousands of customers to manage hundreds of thousands of hosts and millions of VMs world-wide.

The remainder of the paper is organized as follows. We discuss the resource model supported by DRS in Section 2. Section 3 explains the details of the DRS algorithm, and Section 4 discusses DPM. Section 5 contains an extensive evaluation of DRS and DPM based on both a real testbed and an internal simulator used to explore alternative solutions. We discuss lessons learned from our deployment experience in Section 6. Section 7 highlights opportunities for future work. A survey of related literature is presented in Section 8, followed by a summary of our conclusions in Section 9.

2. Resource Model

In this section, we discuss the DRS resource model and its associated resource controls. A resource model explains the capabilities and goals of a resource-management solution. DRS offers a powerful resource model and provides a flexible set of resource controls. A wide range of resource-management policies can be specified by using these controls to provide differentiated QoS to groups of VMs.

2.1 Basic Resource Controls

VMware's resource controls allow administrators and users to express allocations in terms of either absolute VM allocation or relative VM importance. Control knobs for processor and memory allocations are provided at the individual host level by the VMware ESX hypervisor. DRS provides exactly the same controls for a distributed cluster consisting of multiple ESX hosts, allowing them to be managed as a single entity. The basic VMware resource controls are:

- **Reservation**: A *reservation* species a minimum guaranteed amount of a certain resource, *i.e.*, a lower bound that applies even when this resource is over-committed heavily. Reservations are expressed in absolute units, such as megahertz (MHz) for CPU, and megabytes (MB) for memory. Admission control during VMpower-on ensures that the sum of the reservations for a resource does not exceed its total capacity.
- **Limit**: A *limit* species an upper bound on the consumption of a certain resource, even when this resource is under-committed. A VM is prevented from consuming more than its limit, even if that leaves some resources idle. Like reservations, limits are expressed in concrete absolute units, such as MHz and MB.
- **Shares**: *Shares* specify relative importance, and are expressed using abstract numeric values. A VM is entitled to consume resources proportional to its share allocation; it is guaranteed a minimum resource fraction equal to its fraction of the total shares when there is contention for a resource. In the literature, this control is sometimes referred to as a *weight*.

The ability to express hard bounds on allocations using reservations and limits is extremely important in a virtualized environment. Without such guarantees, it is easy for VMs to suer from unacceptable or unpredictable performance. Meeting performance objectives would require resorting to crude methods such as static partitioning or over-provisioning of physical hardware, negating the advantages of server consolidation. This motivated the original implementation of reservations and limits in ESX, as well as their inclusion in DRS.

Although DRS focuses primarily on CPU and memory resources, similar controls for I/O resources have been validated by a research prototype [30]. VMware also offers shares and limit controls for network and storage bandwidth [10, 28]. A new *Storage DRS* feature was introduced in vSphere 5.0, providing a subset of DRS functionality for virtual disk placement and load-balancing across storage devices [16, 29, 31].

2.2 Resource Pools

In addition to basic, per-VM resource controls, administrators and users can specify flexible resource-management policies for *groups* of VMs. This is facilitated by introducing the concept of a logical *resource pool* — a container that species an aggregate resource allocation for a set of VMs. A resource pool is a named object with associated settings for each managed resource — the same familiar shares, reservation, and limit controls used for VMs. Admission control is performed at the pool level; the sum of the reservations for a pool's children must not exceed the pool's own reservation.

Resource pools may be configured in a flexible hierarchical organization; each pool has an enclosing parent pool, and children that may be VMs or sub-pools. Resource pools are useful for dividing or sharing aggregate capacity among groups of users or VMs. For example, administrators often use resource-pool hierarchies to mirror human organizational structures. Resource pools also provide direct support for delegated administration; an administrator can allocate resources in bulk to sub-administrators using sub-pools.

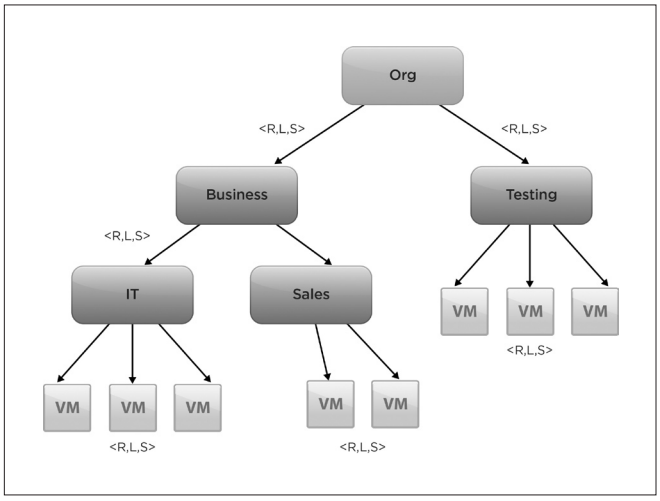

Figure 1. Resource pool tree. R, L, and S denote reservation, limit, and share values, respectively, and are specified for each internal node (pool) and leaf node (VM).

Figure 1 shows a resource pool structure defined by an example organization. Here the resources are first split across two groups, *Business and Testing*. *Business* is further sub-divided into *IT* and *Sales* groups, while *Testing* is a flat collection of VMs. Separate, per-pool allocations provide both isolation between pools, and sharing within pools.

For example, if some VMs within the *Sales* pool are idle, their unused allocation will be reallocated preferentially to other VMs within the same pool. Any remaining spare allocation flows preferentially to other VMs within *Business*, its enclosing parent pool, then to its ancestor, *Org*. Note that in a resource-pool hierarchy, shares are meaningful only with respect to siblings; each pool effectively defines a scope (similar to a currency [50]) within which share values are interpreted.

A distinguished *root resource pool* for a cluster represents the physical capacity of the entire cluster, which is divvied up among its children. All resource pools and VMs in a cluster are descendants of the root resource pool.

2.3 Resource Pool Divvy

A resource pool represents an aggregate resource allocation that may be consumed by its children. We refer to the process of computing the entitled reservation, limit and shares of its child sub-pools and VMs as *divvying*.

Divvying is performed in a hierarchical manner, dividing the resources associated with a parent pool among its children. Divvying starts from the root of the resource pool hierarchy, and ultimately ends with the VMs at its leaves. DRS uses the resulting entitlements to update host-level settings properly, ensuring that controls such as reservations and limits are enforced strictly by the ESX hypervisor. DRS also uses host-level entitlement imbalances to inform VM migration decisions.

During divvying, DRS computes resource entitlements for individual pools and VMs. Divvying computations incorporate user-specified resource controls settings, as well as per-VM and aggregate workload demands. Since pool-level resource allocations reflect VM demands, DRS allows resources to flow among VMs as demands change. This enables convenient multiplexing of resources among groups of VMs, without the need to set any per-VM resource controls.

Divvying must handle several cases in order to maintain the resource pool abstraction. For example, it is possible for the total reservation of a parent pool to be greater than the sum of its childrens' reservations. Similarly, the limit at a parent pool can be smaller than the sum of its childrens' limit values. Finally, a parent's shares need to be distributed among its children in proportion to each child's shares value. In all such cases, the parent values need to be divided among the children based on the user-set values of reservation, limit, shares and the actual runtime demands of the children.

Three divvy operations are defined: *reservation-divvy, limit-divvy,* and *share-divvy*, named for their respective resource controls. DRS carries out these divvy operations periodically (by default, every 5 minutes), reflecting current VM demands. DRS also initiates divvy operations in response to changes in resource allocation settings.

The divvy algorithm works in two phases. In the first, bottom-up phase, divvying starts with the demand values of the individual VMs, which are the leaf nodes. It then accumulates aggregate demands up the resource pool tree.

VM demands are computed by the ESX hypervisor for both CPU and memory. A VM's CPU demand is computed as its actual CPU consumption, CPU_{used}, plus a scaled portion of CPU_{ready}, the time it was ready to execute, but queued due to contention:

$$CPU_{demand} = CPU_{used} + \frac{CPU_{run}}{(CPU_{run} + CPU_{sleep})} \times CPU_{ready} \quad (1)$$

A VM's memory demand is computed by tracking a set of randomly-selected pages in the VM's physical address space, and computing how many of them are touched within a certain time interval [49]. For example, if 40% of the sampled pages are touched for a VM with 16 GB memory allocation, its *active memory* is estimated to be 0.4 X 16 = 6.4 GB.

Once the per-VM demand values have been computed, they are aggregated up the tree. Demand values are updated to always be no less than the reservation and no more than the limit value. Thus, at each node, the following adjustments are made:

$$demand = MAX(demand, reservation) \quad (2)$$

$$demand = MIN(demand, limit) \quad (3)$$

The second divvying phase proceeds in a top-down manner. Reservation and limit values at each parent are used to compute the respective resource settings for its children such that the following constraints are met:

1. Child allocations are in proportion to their shares.

2. Each child is allocated at least its own reservation.

3. No child is allocated more than its own limit.

In order to incorporate demand information, if the sum of the childrens' demands is larger than the quantity (reservation or limit) that is being parceled out, we replace the limit values of the children by their demand values:

$$limit = MIN(demand, limit) \quad (4)$$

This allows reservation and limit settings at a parent to flow among its children based on their actual demands.

Conceptually, the divvy algorithm at a single level in the tree can be implemented by first giving the reservation to every child. Then the extra reservation or limit, depending on what is being divvied out, can be allocated in small chunks to the children, by giving a chunk to the child with minimum value of current *allocation/shares*. If a child's current allocation hits its limit, that child can be taken out from further consideration and capped at the limit.

The share-divvy is performed simply by dividing a parent's shares among its children, in proportion to the shares of each child. Demands do not play any role in the share-divvy operation.

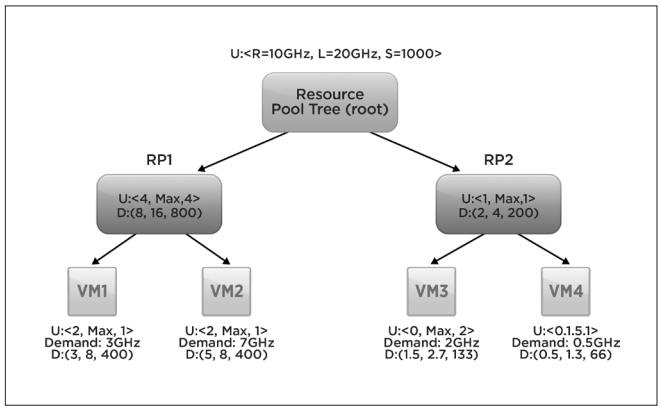

Figure 2. Resource pool divvy example. U denotes the user-set values and D denotes the divvied values. The special Max limit value indicates the allocation is not limited.

We illustrate the divvy operations using a simple example with a two-level resource pool tree. Figure 2 shows a resource pool tree with CPU reservation R = 10 GHz, limit L = 20 GHz and shares S = 1000 at the root node. Shares are unitless and the other two settings are in absolute CPU units (GHz). The user-set resource control values are denoted by U, and the final divvied values are denoted by *D*. Although this example shows only CPU divvying, a similar computation is performed for memory.

In Figure 2, there are two resource pools under the root, *RP*1 and *RP*2, each with two child VMs. First, the bottom-up divvying phase aggregates demands up the tree, starting with individual VM demands. Next, the top-down divvying phase starts at the root node, doing reservation and limit divvy recursively until reaching the leaf nodes.

The 10 GHz reservation at the root is greater than the 5 GHz sum of its childrens' reservations. Thus, the reservation is divvied based on shares, while meeting all the divvying constraints. In this case, the shares-based allocation of 8 GHz and 2 GHz satisfies the reservation and limit values of the two resource pools. Next, the reservation of 8 GHz at RP1 is divvied among VM1 and VM2. Since 8 GHz is smaller than the aggregate demand of 10 GHz from VM1 and VM2, we replace the limit values of VM1 and VM2 by their demand values of 3 GHz and 7 GHz, respectively. As a result, even though the shares of these two VMs are equal, the divvied reservations are 3 GHz and 5 GHz (instead of 4 GHz each). This illustrates how demands are considered while allocating a parent's reservation.

Note that VM demands do not act as an upper bound while divvying limit values here, since the sum of the children's demands is smaller than the limit at the parent. As a result, the actual limit values of the children are used in the divvying. Shares are simply distributed in proportion to the shares of the children. Although we have not walked through every computation in this resource pool tree, one can verify that both reservation and limit values can be divvied in a top-down pass starting from root, while considering user-set values and current demands.

3. DRS Overview and Design

DRS is designed to enforce resource-management policies accurately, delivering physical resources to each VM based on the resource model described in the previous section.

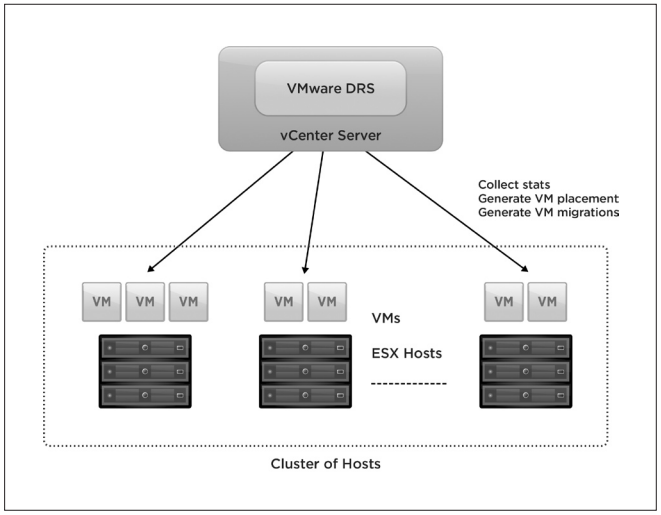

Figure 3. VMware DRS runs as part of the VMware vCenter server management software. VMware DRS collects statistics about hosts and VMs and runs periodically or on demand.

As illustrated in Figure 3, DRS runs as part of the vCenter Server centralized management software [11]. It manages resources for a cluster of ESX hosts as well as the VMs running on them. More specifically, DRS performs four key resource-management operations: (1) It computes the amount of resources to which each VM is entitled based on the reservation, limit and shares values and the runtime demands for all related VMs and resource pools, (2) It recommends and performs migration of powered-on VMs to balance load across hosts in a dynamic environment where VMs' resource demands may change over time, (3) It optionally saves power by invoking DPM, and (4) It performs initial placement of VMs onto hosts, so that a user does not have to make manual placement decisions.

DRS load balancing is invoked periodically (by default, every 5 minutes) to satisfy cluster constraints and ensure delivery of entitled resources. It is also invoked on demand when the user makes cluster configuration changes, *e.g.*, adding a host to the cluster, or requesting that a host enter maintenance mode. When DRS is invoked, it performs the first three resource-management operations listed above, along with a pass to correct cluster constraint violations. For example, constraint correction evacuates VMs from hosts that the user has requested to enter maintenance or standby mode.

DRS initial placement assigns a VM to a host within a cluster when the VM is powered-on, resumed from a suspended state, or migrated into the cluster manually. DRS initial placement shares code with DRS load balancing to ensure that placement recommendations respect constraints and resource entitlements.

In this section, we first discuss DRS load balancing, since it forms the core of the functionality that supports the resource model discussed in the Section 2. We next outline how DRS initial placement works, highlighting how it builds on the DRS load-balancing model. We conclude by presenting the kinds of constraints DRS respects, and how they are handled. DPM is discussed in Section 4.

3.1 Load Balancing

We first examine the DRS load-balancing metric and algorithm. We then consider in more detail how DRS analyzes possible load-balancing moves in terms of their impact on addressing imbalance, their costs and benefits, and their interaction with pending and dependent actions.

3.1.1 Load Balancing Metric

The DRS load-balancing metric is *dynamic entitlement*, which differs from the more commonly-used metric of host utilization in that it reflects resource delivery in accordance with both the needs and importance of the VMs. Dynamic entitlement is computed based on the overall cluster capacity, resource controls, and the actual demand for CPU and memory resources from each VM.

A VM's entitlement for a resource is higher than its reservation and lower than its limit; the actual value depends on the cluster capacity and total demand. Dynamic entitlement is equivalent to demand when the demands of all the VMs in the cluster can be met; otherwise, it is a scaled-down demand value with the scaling dependent on cluster capacity, the demands of other VMs, the VM's place in the resource pool hierarchy, and its shares, reservation and limit.

Dynamic entitlement is computed by running the divvy algorithm over the resource pool hierarchy tree. For entitlement computation, we use the cluster capacity at the root as the quantity that is divvied out. This is done for both CPU and memory resources separately.

DRS currently uses *normalized entitlement* as its core per-host load metric, reflecting host capacity as well as the entitlements of the running VMs. For a host h, normalized entitlement N_h is defined as the sum of the per-VM entitlements E_i for all VMs running on h, divided by the host capacity C_h available to VMs: $N_h = \frac{\sum E_i}{C_h}$. If $N_h \leq 1$ then all VMs on host h would receive their entitlements. If $N_h > 1$, then host h is deemed to have insufficient resources to meet the entitlements of all its VMs, and as a result, the VMs on that host would be treated unfairly as compared to VMs running on hosts whose normalized entitlements were not above 1.

After calculating N_h for each host, DRS computes the cluster-wide imbalance, I_c, which is defined as the standard deviation over all N_h values. The cluster-wide imbalance considers both CPU and memory imbalance using a weighted sum, in which the weights depend on resource contention. If memory is highly contended, *i.e.*, its max normalized entitlement on any host is above 1, then it is weighted more heavily than CPU. If CPU is highly contended, then it is weighted more heavily than memory; equal weights are used if neither resource is highly contended. The ratio 3:1, derived by experimentation, is used when one resource is weighted more heavily than the other.

3.1.2 Load Balancing Algorithm

The DRS load-balancing algorithm, described in Algorithm 1, uses a greedy hill-climbing technique. This approach, as opposed to an exhaustive, offline approach that would try to find the best target balance, is driven by practical considerations. The live migration operations used to improve load-balancing have a cost, and VM demand is changing over time, so optimizing for a particular dynamic situation is not worthwhile.

DRS aims to minimize cluster-wide imbalance, I_c, by evaluating all possible single-VMmigrations, many filtered quickly in practice, and selecting the move that would reduce I_c the most. The selected move is applied to the algorithm's current internal cluster state so that it reflects the state that would result when the migration completes. This move-selection step is repeated until no additional beneficial moves remain, there are enough moves for this pass, or the cluster imbalance is at or below the threshold T specified by the DRS administrator. After the algorithm completes, an execution engine performs the recommended migrations, optionally requiring user-approval.

```
Input: Snapshot of entire cluster (hosts and VMs)
I_c ← σ(N_h) /* standard deviation over all hosts*/
NumMigrations ← 0
while I_c >T and NumMigrations < MaxMigrations do
    BestMigration ← NULL
    Max_δ ← 0
    foreach VM v in the cluster do
        foreach compatible destination host h do
            δ̂ ← improvement in imbalance I_c when v is migrated to h
            if benefit of migration > cost then
                if δ̂ > Max_δ then
                    BestMigration ← migrate v to h
                    Max_δ ← δ̂
    if BestMigration is NULL then
        break
    Apply BestMigration to the algorithm's internal cluster state and update I_c
    NumMigrations++
```

Algorithm 1: DRS Load Balancing

This overview of the DRS algorithm has been greatly simplified to focus on its core load-balancing metric. The actual load-balancing algorithm considers many other factors, including the impact of the move on imbalance and the risk-adjusted benefit of each move. The next sections describe these checks and other factors in more detail.

3.1.3 Minimum Goodness

DRS load balancing rejects a move if it does not produce enough benefit in terms of improvement in the standard deviation value representing imbalance. The threshold used for this filtering is computed dynamically based on the number of hosts and VMs. The threshold is reduced significantly when imbalance is very high and moves to correct it are filtered by the normal threshold, so that many low-impact moves can be used to correct high imbalance.

3.1.4 Cost-Benefit Analysis

The main check DRS uses to filter unstable moves is cost benefit analysis. Cost-benefit analysis considers the various costs involved in the move by modeling the vMotion costs as well as the cost to the other VMs on the destination host which will have an additional VM competing for resources. The benefit is computed as how much the VMs on the source and the migrating VM will benefit from the move. Cost-Benefit analysis also considers the risk of the move by predicting how the workload might change on both the source and destination and if this move still makes sense when the workload changes.

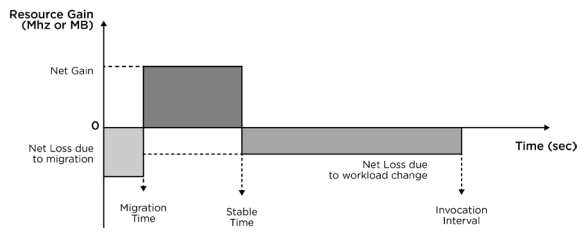

Figure 4: Overview of DRS Cost-Benet analysis

The cost is modeled by estimating how long this VM would take to migrate. The migration time mainly depends on the memory size of the VM as well as how actively the VM modifies its pages during migration. If the VM dirties its pages frequently during vMotion, they must be copied to the destination host multiple times.

DRS keeps track of the transfer rate based on the history of migration times for each VM and host. Based on the transfer rate and current memory size, it calculates the time required for single round of copying. During this time the cost is computed as the resources required for the migrating VM to exist on the destination. The vMotion cost is in the unit of resources (MHz or MB) over time.

The vMotion process itself consumes some resources on both the source and destination hosts and this is also added as a cost. DRS then computes how much the workload inside the VM would suffer due to migration. This is done by measuring how actively the VM writes to its memory pages and how much time each transfer takes. DRS approximates this by measuring the number of times the VM had to be descheduled by the CPU scheduler during past vMotions and how active the VM was at that time. The performance degradation due to the increased cost of modifying memory during migration is computed. This value is used to extrapolate how much the VM would suffer by taking into account its current consumption and added to the cost. This cost is also measured in units of resources over time, such as CPU seconds.

The vMotion benefit is also measured in resources over time. The benefit is computed by calculating how much of the demand for the candidate VM is being clipped on the source versus the increased amount of demand that is expected to be satisfied on the destination. The benefit also includes the amount of unsatisfied demand for other VMs on the source that will be satisfied after the VM moves off. To represent possible demand changes, demand is predicted for all VMs on the source and destination. Based on the workloads of all these VMs, a *stable time* is predicted after which we assume the VM workloads will change to the worst configuration for this move, given recent demand history (by default, for the previous hour). The use of this worst-case value is intended to make the algorithm more conservative when recommending migrations.

With respect to the source host, the worst possible situation is computed as one in which, after the stable time, the workloads of the VMs on the source exhibit the minimum demand observed during the previous hour. With respect to the destination host, the worst possible situation is computed as one in which the demands for the VMs on the destination including the VM being moved are the maximum values observed in the last one hour. By assuming this worst-case behavior, moves with benefits that depend on VMs with unstable workloads are filtered out.

3.1.5 Pending Recommendations

DRS considers the recommendations in flight and any pending recommendations not yet started, so that it does not correct the same constraint violation or imbalance several times. VMs that are currently migrating are treated as if they exist on both source and destination. New recommendations that would conflict with pending recommendations are not generated.

3.1.6 Move Dependency

When DRS generates a sequence of moves, some VM migrations in the sequence may depend on capacity freed up on a host by a VM migration off that host earlier in the sequence. For example, during the constraint violation correction step, DRS may generate a recommendation to move VM *x* off of host A to correct a rule violation, and then during the load-balancing step, it may generate a recommendation to move VM *y* to host A that depends on the resources freed up by the migration of VM *x*. For such sequences, the second move should be executed only after the first move succeeds. DRS can designate dependencies within its output recommendations, and these dependencies are respected by the DRS recommendation execution engine. DRS does not issue sequences of moves whose dependencies cannot be satisfied.

3.1.7 Host-level Resource Settings

As mentioned in section 2.3, in maintaining the illusion that the cluster is a single large host with the aggregate capacity of its constituent hosts, DRS breaks up the user-specified resource-pool hierarchy into per-host resource pool hierarchies with appropriate host-level resource pool settings. Thus, while the VMs are running on a host, the local schedulers on each ESX host allocate resources to VMs fairly, based on VM resource settings and on the host-level resource pool tree provided to the host by DRS.

At the beginning of each balancing invocation, DRS runs the reservation-divvy and limit-divvy algorithm discussed in Section 2.3, to capture the impact of any changes in VM demand over time. Based on any differences between the updated divvy results and those the algorithm last produced, DRS generates recommendations to adjust the resource trees and settings on hosts in the cluster, in accordance with the VMs running on that host.

3.2 VM Initial Placement

DRS VM initial placement is invoked to perform admission control and host selection for VM power-on, for resumption of a suspended VM, or for manual migration of a running VM into a cluster. DRS can be asked to place a single VM, for which it attempts to generate one or more alternative host placement recommendations, or it can be asked to place a set of VMs, for which it attempts to generate a single coordinated recommendation comprised of a placement action for each VM. The latter handles the placement of multiple VMs more efficiently and effectively than a series of individual VM placements, because it builds a single representation of the cluster to place the set of VMs and because the algorithm can order its placement of the VMs to facilitate bin-packing. Specific errors are issued for any VMs DRS cannot place.

During placement, DRS does not have an estimate of the VM's current CPU and memory demand. It makes the conservative assumption that the VM being placed will consume its maximum possible load, *i.e.*, that its memory demand will match its configured memory size and its CPU demand will be such that each of its virtual CPUs (vCPUs) will consume a physical core. DRS placement code leverages the DRS load-balancing code to evaluate the relative goodness of each possible placement.

However, one way that placement differs from load balancing is that placement considers any prerequisite moves that may be necessary. If a VM cannot be placed without a constraint violation on any powered-on host in the cluster, DRS next considers placing the VM on each of the hosts, while attempting to correct the resulting constraint violations via prerequisite moves of other VMs on the host. And if no such placement on powered-on hosts is possible, DRS considers powering-on standby hosts for the VM via DPM.

DRS also handles the initial placement of fault tolerant (FT) VMs [6]. An FT VM consists of both primary and secondary VMs, with the secondary being a replica of the primary that runs in lock-step to allow immediate failover. The primary and secondary VMs must be placed on different hosts that are vMotion compatible. Since it is computationally very expensive to consider all possible pairs of hosts, DRS places the primary VM on the best possible host in terms of goodness. Then, from the subset of hosts that are vMotion-compatible with the primary, it picks the best host for the secondary. If no secondary host can be found, it considers the second-best host for the primary and tries again. This process is repeated until a host is found for both primary and secondary or there are no more hosts left to consider for the primary VM.

3.3 Constraints

The fundamental constraint respected by DRS is VM-to-Host compatibility, i.e., the ability of a host to satisfy a VM's execution requirements. VM-to-Host compatibility information is passed to DRS from the compatibility-checking module of the vCenter management software, and DRS respects this constraint during load balancing and VM placement.

In addition, DRS provides support for enforcing a set of constraints to handle various use cases such as co-location of VMs for performance, placement of VMs on separate hardware for availability and fault isolation, and affinity to particular hardware to handle licensing issues. It also handles evacuation of hosts the user has requested to enter maintenance or standby mode, preservation of spare resources for failover, and the role of special VMs that provide services to other VMs. DRS respects constraints during VM initial placement and runs a pass prior to load balancing to correct violations of DRS-enforced constraints, reporting errors for any of those constraints that cannot be corrected. We discuss several constraints and their specific use cases next.

3.3.1 Affinity Rules

DRS supports VM-to-VM or VM-to-Host rules, which are used for a variety of common business scenarios. VM-to-VM *anti-affinity* rules define a set of VMs that are to be kept on separate hosts. These rules are typically used for availability and are mandatory, *i.e.,* DRS will not make any recommendation that would violate them. For example, avoiding a single point of failure due to running two VMs on the same host. VM-to-VM *affinity* rules define a set of VMs that are to kept on the same host and are used to enhance performance of communicating VMs, because intra-host VM-to-VM networking is optimized to perform in-memory packet transfers, without using NIC hardware. These rules are mandatory for load-balancing, but

DRS will violate them if necessary to place VM(s) during power-on. VM-to-VM rule violations are corrected during the initial phase of a load-balancing run and that corrected state is maintained during load balancing. For VM-to-VM anti-affinity, potential balancing moves introducing violations are filtered; for VM-to-VM affinity, potential balancing moves are formed by treating each set of affine VMs on a host as if it were a single large VM.

VM-to-Host rules define a set of VMs being affine or anti-affine with a set of hosts. These rules can be specified as mandatory or preferred, with the latter meaning that they are enforced unless they engender additional constraint violations or cannot be respected without causing host overutilization. VM-to-Host rules are useful for a variety of reasons. Mandatory VM-to-Host affinity rules are often used to enforce licensing, to associate VMs requiring a host-based software license with the hosts having that license. Preferred VM-to-Host affinity or anti-affinity rules are often used to manage availability and/or site locality.

Mandatory VM-to-Host rules are represented in the VM-to-Host compatibility information passed to DRS and hence are handled as a fundamental constraint. Preferred VM-to-Host rules are represented as alternative VM-to-Host compatibility information. DRS handles preferred VM-to-Host rules by running a what-if pass with the preferred rules treated as mandatory. If the resulting cluster state has no constraint violations or overutilized hosts, the result of the pass is accepted. Otherwise, DRS retries the what-if pass with the preferred rules dropped and accepts the latter result if it has fewer constraint violations or overutilized hosts, else it accepts the former result.

3.3.2 High Availability

vSphere High Availability [7] (HA) is a cluster service that handles host failures, and restarts failed VMs on remaining healthy hosts. HA runs as a decentralized service on the ESX hosts and keeps track of liveness information through heart-beat mechanisms. DRS supports HA functionality in two ways: by preserving powered-on idle resources to be used for VM restart in accordance with the HA policy and by defragmenting resources in the cluster when HA restart cannot find sufficient resources.

For DRS to preserve enough powered-on idle resources to be used for VM restart, HA needs to express to DRS the resources required based on HA policy settings. HA supports three methods by which users can express the resources they need preserved for failover: (1) the number of host failures they would like to tolerate, (2) the percentage of resources they would like to keep as spare, and (3) the designation of particular host(s) as failover hosts, which are not used for running VMs except during failover.

HA expresses the implication of these policies to DRS using two kinds of constraints: the minimum amount of CPU and memory resources to be kept powered-on and the size and location of unfragmented chunks of resources to be preserved for use during failover. The latter is represented in DRS as *spare VMs*, which act like special VMs with reservations used to ensure DRS maintains spare resource slots into which VMs can be failed over and whose VM-to-Host compatibility information is used to create those slots on appropriate hosts.

Spare VMs are also used in the unusual event that more resources are needed for failure restart than expected due to the failure of more hosts than configured by the HA policy. For any VMs that HA cannot restart, DRS is invoked with spare VMs representing their configuration, with the idea that it may be able to recommend host power-ons and migrations to provide the needed resources.

3.3.3 ESX Agent VMs

Some services that an ESX host provides for VM use (*e.g.*, vShield [15]) are encapsulated in VMs; these VMs are called ESX Agent VMs. An agent VM needs to be powered on and ready to serve prior to any non-agent VM being powered on or migrated to that host, and if the host is placed in maintenance or standby mode, the agent VM needs to be powered off after the non-agent VMs are evacuated from the host.

To support agent VMs, DRS respects their role as part of the host platform, which has a number of implications. The DRS algorithm does not produce recommendations to migrate or place non-agent VMs on hosts on which required agent VMs are not configured. On hosts with configured agent VMs, the DRS algorithm respects the agent VMs' reserved resources even when they are not in a ready-to-serve state. The DRS execution engine understands that non-agent VMs need to wait for required agent VMs to be powered on and ready to serve on the target host. And the DRS load-balancing code understands that agent VMs do not need to have evacuation recommendations produced for them when a host is entering maintenance or standby mode; the agent VMs are automatically powered off by the agent VM framework after the non-agent VMs are evacuated.

4. DPM Overview and Design

Distributed Power Management (DPM) is a feature of DRS that opportunistically saves power by dynamically right-sizing cluster capacity to match workload demands, while respecting all cluster constraints. DPM recommends VM evacuation and powering off ESX hosts when the cluster contains sufficient spare CPU and memory resources. It recommends powering ESX hosts back on when either CPU or memory resource utilization increases appropriately or additional host resources are needed to meet cluster constraints, with DRS itself recommending host power-ons for the latter reason. DPM runs as part of the DRS balancing invocation as an optional final phase.

Note that in addition to DPM, each ESX host performs Host Power Management (HPM), which uses ACPI P-states and C-states [12] on ESX hosts to reduce host power consumption while respecting VM demand. HPM works synergistically with DPM in reducing overall cluster power consumption, with DPM reducing the number of running hosts and HPM reducing the power consumed by those hosts.

DPM decouples the detection of the need to consider host power-on or power-off from the selection of the host, which allows host selection to be independent of how the hosts are currently utilized. This means that DPM host selection for power-off does not depend on finding hosts on which both CPU and memory utilization are currently low; DPM can migrate VMs to exploit low utilization of these resources

currently not manifest on the same host. And more importantly, this decoupling means that DPM host selection for power-off can consider criteria that are more critical in the longer term than current utilization, such as headroom to handle demand burst, power efficiency, or temperature.

For example, host A may currently have lower CPU and memory utilization than host B, but host B may be a better candidate for power-off, because it is smaller than host A and hence is less able to handle VM demand burst than host A, or it is less power-efficient than host A, or it is in a hotter part of the datacenter than host A.

In this section, we first discuss how DPM evaluates the utilization of the powered-on hosts to determine if it should suggest host power-off or power-on recommendations. Next, we examine how DPM evaluates possible recommendations. Finally, we explain how DPM sorts hosts for power-on or power-off consideration. Note that this section includes a number of specific factors whose defaults were chosen based on experimental data; the default values are included to illustrate how the system is typically configured, but all of these values can be tuned by the user if desired.

4.1 Utilization Evaluation

To determine whether the currently powered-on cluster capacity is appropriate for the resource demands of the running VMs, DPM evaluates the CPU and memory resource utilization (VM demand over capacity) of powered-on hosts in the cluster. Utilization can exceed 100% since demand is an estimate of VM resource consumption assuming no contention on the host. As mentioned in Section 2.3, a VM's CPU demand includes both its CPU usage and a portion of its ready time, if any. A VM's memory demand is an estimate of its working set size.

DPM considers demand over an extended time period and it uses the mean demand over that period plus two standard deviations for the utilization calculation. This makes DPM conservative with respect to host power-off, which is the bias customers prefer. In order to be relatively quick to power-on a host in response to demand increases and relatively slow to power-off a host in response to demand decreases, the time period used for host power-off is the last 40 minutes and for host power-on is the last 5 minutes.

Utilization is characterized with respect to a target range of 45-81% (*i.e.*, 63 +/-18), with utilization above 81% considered high and utilization below 45% considered low. DPM evaluates cluster utilization using a measure that incorporates CPU and memory utilization on a per-host basis in such a manner that higher utilizations on certain hosts are not offset by lower utilizations on other hosts. This is done because DRS cannot always equalize VM load across hosts due to constraints.

Use of this metric allows DPM to consider host power-on to address high utilization on a few hosts in the cluster (*e.g.*, ones licensed to run Oracle VMs) when the overall utilization of the powered-on hosts in the cluster is not high, and to consider host power-off to address low utilization on a few hosts in the cluster (e.g., ones not connected to a popular shared storage device) when the overall utilization of the powered-on hosts in the cluster is not low. Note that DPM executes on a representation of the cluster produced by a complete DRS load balancing pass.

DPM computes a cluster-wide high and a low score for each resource. Considering the hosts whose utilization for the power-on demand period is above the high-utilization threshold (default 81%):

$$High\ UtilizationScore = \sqrt{\sum DistanceAboveThreshold^2} \quad (5)$$

Considering the hosts whose utilization for the power-off demand period is below the low utilization threshold (default 45%):

$$LowUtilizationScore = \sqrt{\sum DistanceBelowThreshold^2} \quad (6)$$

If the high score is greater than zero for *either CPU or memory*, DPM considers producing host power-on recommendations. Otherwise, if the low score is greater than zero for *both CPU and memory* (possibly on different hosts), DPM considers producing host power-off recommendations. The key intuition behind this scoring is that we want to identify cases where a subset of hosts may have very high or very low utilizations that can be acted upon by suggesting power-on and power-off operations respectively.

4.2 Host Power-off Recommendations

DPM considers host power-off in the current invocation if two criteria are met: non-zero low scores for both CPU and memory and zero high scores for both. To evaluate various alternatives, DPM considers each candidate host in a sorted order. If a candidate host power-off passes both the cluster utilization and cost-benefit evaluations described below, the recommendation to power it off along with its prerequisite VM evacuation recommendations is added to the list of the recommendations that will be issued by this invocation of DRS. The internal representation of the cluster is then updated to reflect the effects of this recommendation. DPM continues to consider host power-off until either the cluster utilization scores no longer show both low CPU and memory utilization or there are no more candidate hosts to be considered.

For a candidate host, DPM runs DRS in a what-if mode to correct constraints and load balance the cluster assuming that the host were entering standby. If what-if DRS can fully evacuate the host, DPM evaluates the resulting cluster state using the utilization scores given in the previous section and compares the scores with those of the cluster state before the host power-off was considered. If the cluster low utilization score is lowered and if the cluster high utilization score is not increased, then the candidate host power-off is next subjected to DPM power-off cost-benefit analysis.

DPM cost-benefit analysis considers the risk-adjusted costs and benefits of the power-off. The costs include the CPU and memory resources associated with migrating VMs off the powering-off host and the corresponding resources associated with repopulating the host when it powers back on in the future. The costs also include the risk-adjusted resource shortfall with respect to satisfying VM demand if the host's CPU and/or memory resources are needed to meet that demand while the host is entering standby, powered-off, or rebooting. The risk-adjusted resource shortfall is computed

assuming the current demand persists for the expected stable time of that demand given an analysis of recent historical stable time, after which demand spikes to a "worst-case" value. This value is defined to be the mean of the demand over the last 60 minutes plus three standard deviations.

The benefit is the power saved during the time the host is expected to be down, which is translated into CPU and memory resource currency (the host's MHz or MB multiplied by time in standby) to be weighed against cost. By default, the benefit must outweigh the cost by 40x to have the power-off pass DPM's cost benefit filter; this value was chosen based on experimentation to match customer preference of not impacting performance to save power.

4.3 Host Power-on Recommendations

DPM evaluates host power-on given high CPU or memory utilization of any powered-on host. DPM considers each candidate host in a sorted order. If a candidate host power-on passes the cluster utilization evaluation described below, the recommendation to power it on is added to the list of recommendations generated by DRS. The internal representation of the cluster is also updated to reflect the effects of this recommendation. DPM continues to consider host power-on until either the cluster utilization scores no longer show either high CPU or memory utilization or there are no more candidate hosts to be considered.

For a candidate host, DPM runs DRS in a what-if mode to correct constraints and load balance the cluster assuming that host were powered-on. If the host power-on reduces the high utilization score, then the stability of that improvement is evaluated. Stability of the improvement involves checking that the greedy rebalancing performed by DRS in the presence of that host was better because of the addition of that host. This is checked by doing a what-if power-off of the same candidate host and checking that the improvement in the high-utilization score obtained by considering its power-on does not remain. If the host power-on benefit is stable, then the host is selected for power-on. Some extra evaluation is performed for the last host needed to address all remaining high utilization in the cluster. In that case, DPM evaluates multiple hosts until it finds one that decreases the high score and provides minimum or zero increase in the low utilization score.

4.4 Sorting Hosts for Power-on/off

As mentioned earlier, DPM considers the hosts for evaluation in a sorted order. DPM currently prefers to keep larger servers powered-on, motivated by the fact that they provide more resource headroom and that they typically are more efficient at amortizing their idle power consumption, which for datacenter-class servers often exceeds 60% of their peak power. For servers of the same size, cost of evacuation is a secondary criteria for power-off consideration and randomization for wear-leveling is a secondary criteria for power-on consideration.

As long as some host in the order satisfies the necessary conditions, it is selected and the corresponding operation is applied to that host in the internal snapshot. The rest of the DPM evaluation is conducted with that host in the new state. Thus DPM doesn't select a host that leads to maximum improvement in the scores, but uses this sorted list as the greedy order.

5. Experimental Evaluation

In this section, we present the results of an extensive evaluation of DRS and DPM using an in-house simulator and experiments on a real testbed. In the first subsection, we present the results using a cluster simulator to show how DRS load-balancing improves the amount of CPU and memory resources delivered to a set of VMs as compared to those delivered without DRS load-balancing; we also highlight the need and impact of DRS cost-benefit analysis and of DPM.

In the second subsection, we present results gathered on a real cluster to illustrate the effectiveness of the DRS algorithm using a variety of metrics. And finally, we will present the results gathered on a real cluster that includes power measurements to show the performance of DPM and to demonstrate the benefit of using DPM in addition to host power management [13].

5.1 Experimental Results from Simulation

5.1.1 Simulator Design

To develop and evaluate various algorithms, we implemented a simulator that simulates a cluster of ESX hosts and VMs. The simulator allows us to create different VM and host profiles in order to experiment with different configurations. For example, a VM can be defined in terms of a number of virtual CPUs (vCPUs), configured CPU (in MHz) and configured memory size (in MB). Similarly a host can be defined using parameters such as number of physical cores, CPU (MHz) per core, total memory size, power consumption when idle etc. In terms of workload, the simulator supports arbitrary workload specifications for each VM over time and generates CPU and memory demand for that VM based on the specification.

Based on the physical characteristics of the host, VM resource demands and specifications, the simulator mimics ESX CPU and memory schedulers and allocates resources to the VMs in a manner consistent with the behavior of ESX hosts in a real DRS cluster. The simulator supports all the resource controls supported by the real ESX hosts, including reservation, limit and shares for each VM along with the resource pools.

The simulator generates the allocation each VM receives, whenever there is any change in demand by any of the VMs on the host. The simulator supports vMotion (live migration) of VMs, models the cost of vMotion and the impact on the workload running in the VM, based on a model of how vMotion works in a physical ESX host. The simulator also takes into account the resource settings for the resource pool trees on the host when resources are divvyed out, similar to how the real ESX host divvies out the host resources based on the host-level resource pool hierarchy.

The simulator was used to study configurations of VMs and hosts and how the algorithm performs in terms of improving the resource allocation to the VMs by placing and moving the VMs at the right time. If the algorithm causes too many migrations, it will affect the workload negatively as the migrations may require a non-trivial amount of time and the workload performance will be degraded during migration.

The production DRS algorithm code is compiled into this simulator. This allows the quality of the moves can be evaluated using all the modeling done by the simulator and decisions made by DRS or DPM.

Since the simulator is deterministic, we can run a test with different algorithms and compute the overall resources allocated to the VMs. Even though the simulator supports random distributions as input for workload generations, the random numbers use a configurable seed so that the VMs would have the same load value at the same time in the simulation across different runs.

The main metric used to study how effectively resources are being allocated is a *cumulative payload metric* that can be defined as follows:

$$\sum_{t=1}^{T} \sum_{k=1}^{All\ VMs} \frac{(Satisfied\ Demand\ for\ VM_k\ at\ t)}{Cluster\ capacity\ at\ t} \times 100 \quad (7)$$

Here *T* denotes the total simulation time. This metric is calculated separately for both CPU and memory. The CPU payload is the area under the curve when the sum of the CPU utilization for all the VMs is plotted in time. If all the VMs have equal shares and have no reservation or limit, a higher number is better. Furthermore, this value is normalized by dividing the area by sum of capacity of all the hosts times length of the experiment. The CPU playload thus captures the total number of CPU cycles in the cluster (on all the hosts) and how many of those cycles were usefully spent.

The memory payload is computed similarly. If VMs have different shares, then this metric must be calculated for each share class to ensure the DRS algorithm is fair when it improves the overall utilization. The simulator also estimates power consumption of the ESX hosts based on a simple power model that involves base power and additional power for the workload it currently runs.

EXPERIMENT	MIGRATIONS	CPU	MEMORY
Without VMware DRS	0	55.68	65.94
With VMware DRS	166	73.74	94.99

Table 1: Benefits of VMware DRS. VMware DRS improved CPU and memory utilization significantly.

5.1.2 Simulator-Based Evaluation

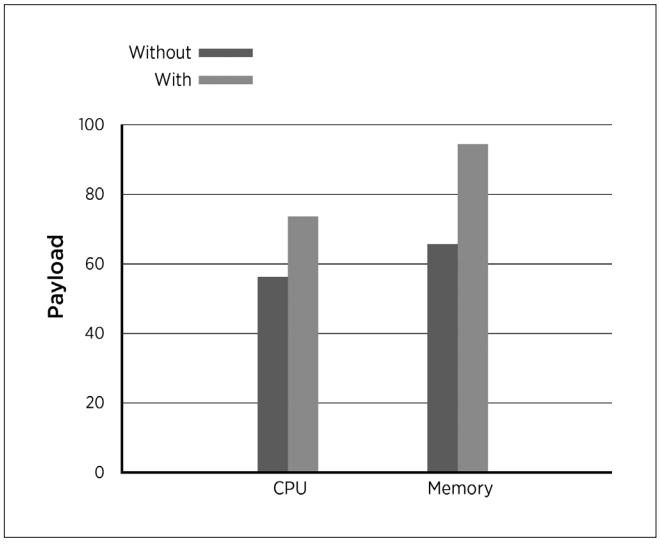

Figure 5: Difference in CPU and Memory payload when VMware DRS is enabled

Using the simulator we first show the benefits of DRS load balancing. We simulated an environment consisting of 30 hosts and 400 VMs. The simulated hosts were each configured with 8 cores and 3 GB of RAM. The VMs were all configured as uniprocessors (with a single vCPU) and had a High-Low workload pattern inducing a demand of either 600 or 1800 MHz randomly for 600 seconds, followed by an inactive period where they would demand 100 MHz for another 600 seconds. The VMs had constant memory demand of 220 MB including overhead. The VMs were randomly placed in the beginning of the experiments.

In the first experiment, DRS was disabled and the VMs could not be moved around so that active VMs could take advantage of the hosts with many inactive VMs. When the same experiment was performed with DRS, the VMs were moved when some hosts had many inactive VMs and some hosts were overcommited. The results are shown in Table 1. The CPU and memory metrics were as computed using Equation 7. The results show the percentage of the total cluster resources (in the units of cycles for CPU and memory-seconds for memory) that was used to satisfy demand.

In the next set of experiments, we illustrate the impact of DRS cost-benefit analysis. The environment consisted of 15 hosts and 25 VMs. The hosts were configured with three different types — a small host with 2 cores, 2 GB memory, a medium host containing 4 cores and 2 GB of memory and a large host containing 4 cores and 4 GB of memory. The workloads inside the VMs mimicked a diurnal pattern. For 600 seconds the VMs were busy. For experiment 1, The values were normally distributed around 1000 MHz with a variance of 800 MHz. For the next 600 seconds it was distributed around 600 MHz with a variance of 500 MHz. For the other two experiments the high value was distributed uniformly between 500 and 1500 MHz and the low value was distributed between 0 and 500 MHz. Figure 7 shows a typical workload inside VM from experiment 1. DPM was turned on for these set of experiments. The three experiments show that cost benefit significantly reduces the

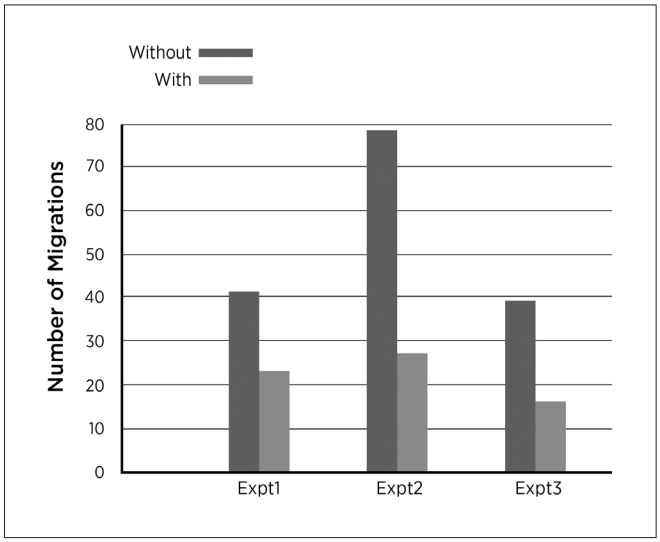

Figure 6: Difference in the number of migrations when cost-benefit analysis is enabled.

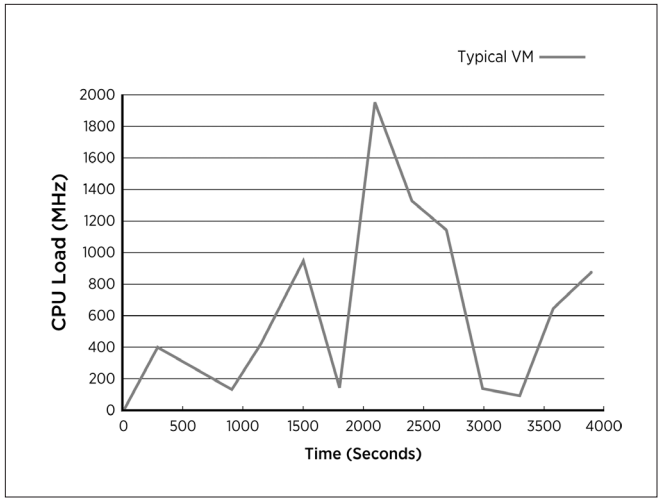

Figure 7: Typical VM workload used in the cost-benefit experiments.

number of migrations while keeping CPU and memory utilizations of the cluster the same or improving it. Table 2 presents the results of these experiments.

EXPERIMENT	MIGRATIONS	CPU	MEMORY
Expt1	41	19.16	15.84
Expt1-CB	25	19.25	15.86
Expt2	78	18.92	15.84
Expt2-CB	27	19.20	15.84
Expt3	39	6.67	42.90
Expt3-CB	16	6.67	43.05

Table 2: Impact of cost-benefit analysis.

The simulator also models the power consumption of each host and the whole cluster. We used a cluster with 32 hosts and 200 VMs. Similar to the cost benefit experiment, the VMs had a day-night workload where all the VMs were active (between 1200 and 2200 MHz) during the day and would go partially idle (workload normally distributed between 500 to 1000 MHz) during the night. DPM would shut down the hosts when the VMs became idle. Table 3 shows the results with and without DPM. Power is in the units of Kilowatt hours. The CPU and memory metrics are as described above.

EXPERIMENT	MIGRATIONS	CPU	MEMORY	POWER
Without VMware DPM	0	2.77	2.31	31.65
With VMware DPM	273	2.78	2.32	11.97

Table 3: DPM achieves signicant power savings without impacting performance.

5.2 Real Testbed: DRS Performance

We now present the results of an experiment that shows the effectiveness of the DRS algorithm on a real testbed. The experiment was run on a DRS cluster consisting of 32 ESX hosts and 1280 Red Hat Enterprise Linux (RHEL) VMs. vSphere 5.0 [4] was used to manage the cluster. The VMs belonged to 4 different resource pools (RP-HighShare, RP-NormShare-1, RP-NormShare-2, RP-LowShare) with different resource shares, with 320 VMs in each resource pool. Each VM was configured with 1 vCPU and 1 GB memory. Notice that we are using only uniprocessor VMs in our evaluation because the number of vCPUs in a VM does not affect DRS behavior in a significant way. The hardware and software setup of the testbed follows:

- **vCenter Server 5.0**: Intel Xeon E5420, 2 x 4 cores @ 2.50 GHz, 16 GB RAM, runs 64-bit Windows 2008 Server SP1.
- **vCenter Database**: AMD Opteron 2212 HE, 2 x 2 cores @ 2 GHz, 12 GB RAM, runs 64-bit Windows 2008 Server SP1 and Microsoft SQL Server 2005.
- **ESX 5.0 hosts**: Dell PoweEdge 1950II servers, Intel Xeon E5420, 2 x 4 cores @ 2.5 GHz, 32 GB RAM.
- **Network**: 1 Gb vMotion network configured on a private VLAN.
- **Storage**: 10 LUNs on an EMC Clariion CX3-80 array shared by all the hosts.

In this experiment, DRS was enabled with default settings, and DPM was disabled, so all 32 hosts remained powered on. We started with all 1280 VMs idling in the cluster. These VMs were evenly distributed across all 32 hosts, due to DRS load balancing. We then injected a CPU load of roughly 20% of a core into each of the 640 VMs in the first two resource pools (RP-HighShare and RP-NormShare-1). Since these 640 VMs were running on 16 hosts of the cluster, the VM load spike led to an overload on these 16 hosts and a significant load imbalance in the cluster.

Even though such an extreme scenario may not occur commonly in a datacenter, it serves as a stress test for evaluating how DRS handles such a significant imbalance in the cluster. We also use this example to illustrate the following key metrics for characterizing the effectiveness of the DRS algorithm.

- **Responsiveness**: How quickly can DRS respond to VM load changes?
- **Host Utilization**: Total resource entitlement from all the VMs running on a host, normalized by the host's available capacity.
- **Cluster Imbalance**: The standard deviation of the current host load across all hosts in the cluster.
- **Number of vMotions**: How many vMotions are recommended by DRS to balance the cluster?
- **VM Happiness**: Percentage of resource entitlement received by each VM.

Next we discuss the experimental results using these five metrics.

Responsiveness: In this experiment, DRS responded in the first invocation after the VM load spikes that led to a severe imbalance in the cluster. The DRS algorithm took 20 seconds to run, and gave recommendations to migrate VMs away from the overloaded hosts. This behavior demonstrates the responsiveness of the DRS algorithm when faced with significant VM load changes.

Host Utilization: Figures 8 and 9 show the normalized total CPU/memory entitlement of all hosts based on the VMs running on each host.[1] The larger number is worse in terms of load on the host. A value above one indicates that the host is overloaded. As we can see, due to the increased VM loads, 16 of the hosts had a spike in the normalized CPU utilization and reached an overload state, whereas the other 16 hosts remained lightly loaded.

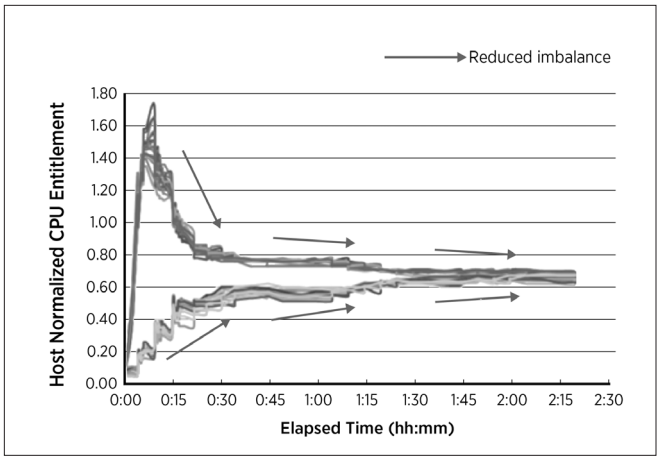

Figure 8: Per-host normalized CPU entitlement. Each line represents a host in the 32-host cluster.

After DRS started moving VMs out of the busy hosts, within 30 minutes the CPU utilization of the most-loaded hosts was reduced to around 0.8. In the remaining time, DRS continued to balance the CPU load in the cluster and reached a steady state in 2 hours and 20 minutes. As a result of balancing the host CPU loads, the host memory loads became slightly imbalanced, while staying within a range of 0:14-0:32. This is acceptable because for most applications, memory does not become a bottleneck at such low utilizations.

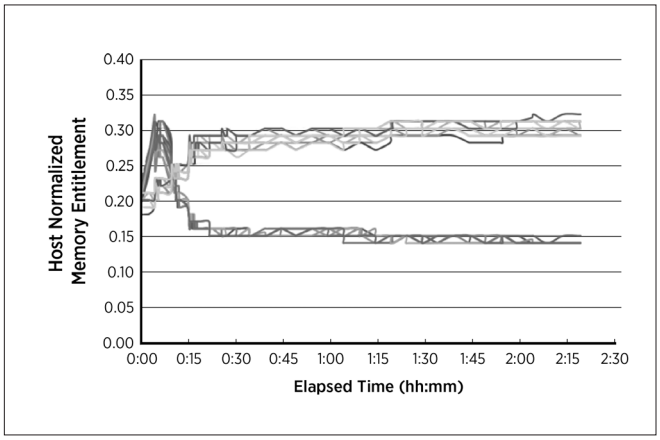

Figure 9: Per-host normalized memory entitlement. Each line represents a host in the 32-host cluster.

Cluster Imbalance: Figure 10 plots the cluster imbalance over time. The target imbalance value is 0.05 in this case. As a result of the VM load increase, the cluster imbalance initially reached a high value of 0.5. DRS quickly brought the imbalance to below 0.1 within 30 minutes, and within 90 minutes the cluster was fairly close to being "balanced." In the remaining 50 minutes, DRS further reduced the imbalance gradually and finally brought the cluster imbalance down below its target.

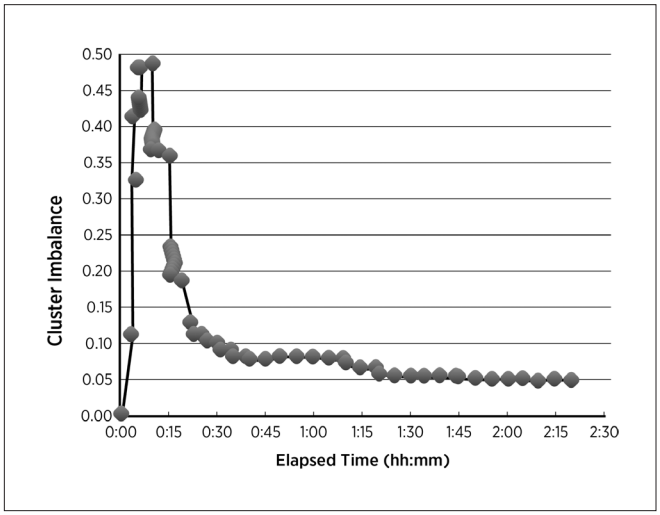

Figure 10: Cluster imbalance.

Number of vMotions: Figure 11 reports the number of vMotions recommended during each DRS invocation. A total of 267 vMotions were executed, and among these, 186 vMotions occurred in the first three rounds of DRS invocations (> 40 in each round), due to the severe imbalance in the cluster. This caused the cluster imbalance to drop quickly. In the remaining 25 rounds of invocations, the imbalance became more subtle and DRS behaved relatively conservatively, taking into account the migration cost in the cost-benefit analysis.

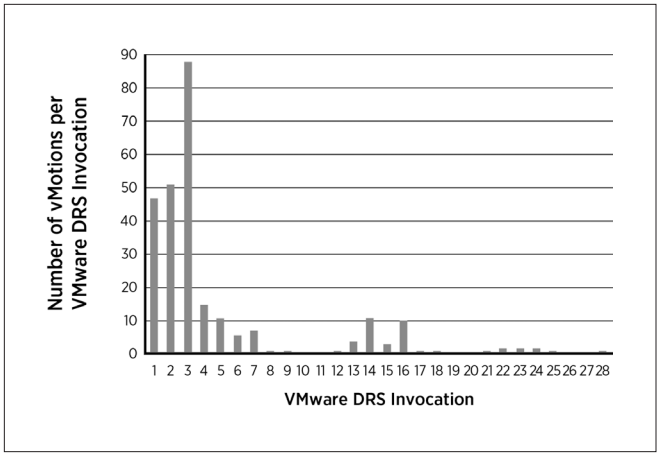

Figure 11: Number of vMotion operations per VMware DRS invocation.

VM Happiness: Finally, we summarize the overall VM happiness during the course of the experiment. Since memory never became a bottleneck in this experiment, we focus on CPU and define *happiness* for an individual VM as the percentage of CPU entitlement received by this VM. For a resource pool, we characterize its overall happiness using two metrics — *average happiness and percentage of happy VMs*. The former is the individual VM happiness averaged across all the VMs in the resource pool, and the latter is the percentage of VMs in the resource pool receiving at least 95% of entitled CPU.

[1] In most of the figures in this subsection, the x-axis represents the elapsed time T in the format of hh:mm, where T=0:00 indicates when the VM load increase occurred.

The values of these two metrics for the 4 resource pools are displayed in Figure 12 and Figure 13, respectively. In the first 30 minutes, the happiness of the VMs in the first two resource pools (RP-HighShare and RP-NormShare-1) suffered, because the VMs in these two resource pools experienced a load spike and started contending for CPU time on the 16 overloaded hosts. However, it is worth noting that under such resource contention, the average happiness of the first resource pool (RP-HighShare) is much higher than that of the second resource pool (RP-NormShare-1), because the former has a higher shares value than the latter ("High" vs. "Normal").

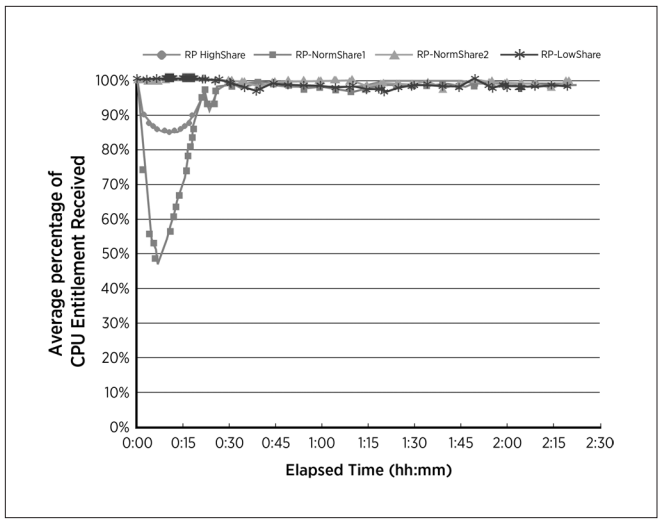

Figure 12: Average happiness (percentage of CPU entitlement received) for each of the 4 resource pools.

This result also validates the fairness of the DRS algorithm with respect to resource shares. Over time, DRS was able to improve the happiness of all the VMs, resulting in an average happiness of close to 100% for all the resource pools. Similarly, the percentage of happy VMs reached 100% for the first two resource pools (RP-HighShare and RP-NormShare-1), and was above 90% for the other two resource pools.

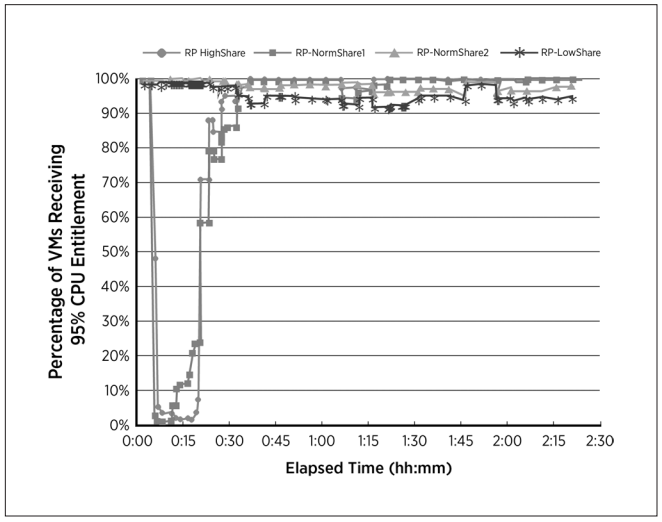

Figure 13: Percentage of happy VMs (receiving at least 95% of CPU entitlement) for each of the 4 resource pools.

5.3 Real Testbed: DPM Performance

We present the results of an experiment that evaluates the performance of DPM and compares DPM with host power management (HPM) [13]. These results demonstrate that DPM is able to provide a fairly significant reduction in power consumption for clusters with sufficiently long periods of low utilization. In addition, combining DPM with host power management provides the most power savings as compared with using either policy alone.

The experiment was run on a DRS cluster consisting of 8 ESX 4.1 hosts and 400 RHEL VMs. vSphere 4.1 [4] was used to manage the cluster. Each VM was configured with 1 vCPU and 1 GB memory. The hardware setup for the testbed is similar to that of the DRS experiment, except that we used a later generation of Dell servers (PowerEdge R610) for the ESX hosts, so that we could obtain host power measurements from the iDRAC controller [2].

The experiment was run for 90 minutes. In the first 30 minutes, all 400 VMs were idle. After the 30th minute, each VM had a surge in the CPU demand resulting in higher cluster utilization. The increased load lasted until the 60th minute, after which all the VMs became idle again. The experiment was repeated three times using three power management policies: *HPM only*, *DPM only*, and *DPM+HPM combined*.

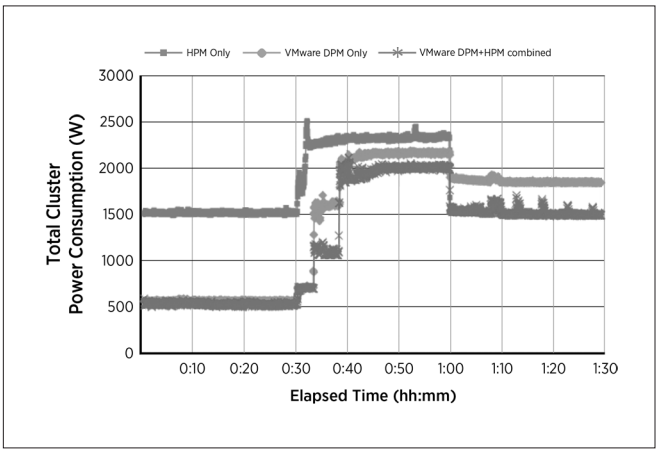

Figure 14: Total power consumption of the 8-host cluster.

Figure 14 shows the total power consumption from the eight hosts in the cluster as a function of time for the three different policies. Next, we describe the result from each policy.

HPM only: In this case, the HPM "Balanced" policy [13] was used. Since DPM was disabled, all eight hosts remained powered on. When all the VMs were idle (first and last 30 minutes), each host had a low (about 25%) CPU utilization. HPM was able to reduce the per-host power consumption by leveraging the lower processor P-states, resulting in 1500W of total cluster power. When the VMs were busy (second 30 minutes), host CPU utilization became high enough (> 60%) such that all the processor cores were running at the highest P-state (P0) most of the time. As a result, the total cluster power was between 2200-2300W.

DPM only: In this case, the HPM policy was set to "High Performance", effectively disabling HPM. Since DPM had been enabled, six of the eight hosts remained in the powered-off state during the initial 30 minutes of the idle period. This led to a total cluster power consumption of about 566W, a 60% reduction as compared with the 1500W in the HPM-only case. Notice the power saving is not exactly 75% here because in the DPM case, the two hosts that remained powered on had a much higher CPU utilization, resulting in higher per-host power consumption. After the VM load increase, DPM first powered on four of the standby hosts in the 33rd minute, and then powered on two additional hosts in the 38th minute, bringing the cluster back to full capacity. The total cluster power consumption increased to approximately 2100W after all the hosts were powered on. When the VMs became idle again, DPM (by design) kept all the hosts powered on for more than 30 minutes, resulting in a total cluster power consumption of approximately 1900W.

DPM+HPM combined: This case is similar to the DPM-only case, except for the following three observations. First, after the VM load increase, DPM initially powered on two of the standby hosts in the 33rd minute, and then powered on the four remaining hosts in the 38th minute. Second, the total cluster power after all the hosts were powered on was roughly 2000W, 100W lower compared to the DPM-only case. This was because the newly powered-on hosts had fewer VMs running on them, resulting in lower host utilization and providing HPM with an opportunity to reduce the power consumption on these hosts. Third, in the last 30 minutes when the cluster was idle, the DPM+HPM combined policy provided an additional 400W of power reduction (1500W vs. 1900W) compared to the DPM-only case. Overall, this combined policy provides the maximum power savings among the three power management policies we tested.

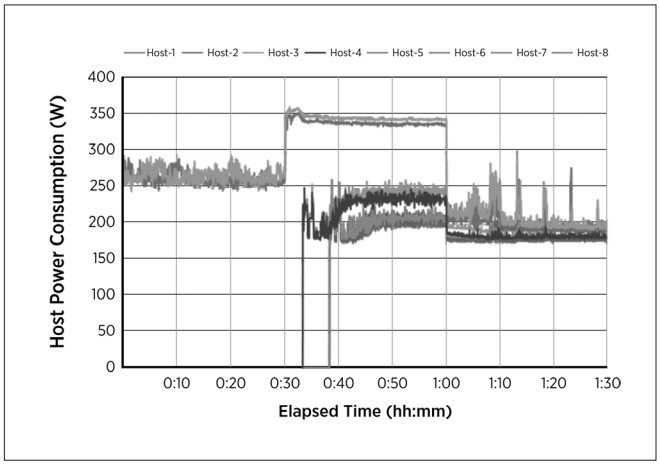

Figure 15: Per-host power consumption for the 8 hosts in the cluster.

Figure 15 shows the per-host power consumption as a function of time for the DPM+HPM combined case. We can clearly see how each host's power consumption varied as the host power state or the VM load level changed. In the last 30 minutes, periodic spikes are visible in the host power consumption, due to vMotion-induced higher CPU utilization on these hosts. These vMotions were recommended by DRS every five minutes to balance the load in the cluster.

6. DRS and DPM in the Field

When DRS was introduced in early 2006, vMotion was just beginning to gain widespread adoption, but customers were wary of automated migration of virtual machines. In the early days, one request from customers was support for manual-move recommendations. A human administrator would inspect the recommendations and apply the moves only if they made sense. Some administrators would run DRS in manual mode and if the same move was recommended over a substantial number of DRS invocations, then the administrator would apply the move. The use of DRS manual mode diminished over time as the DRS algorithm became more mature and as administrators became more comfortable with automated VM migration; as of vSphere 5.0, the use of DRS manual mode is very low.

The first version of DRS did not have cost-benefit analysis turned on, as the code was considered experimental. This led to the problem that DRS could make recommendations that moved VMs back and forth in response to frequently changing demand. In the very next release, cost-benefit feature was enabled by default, leading to higher-quality moves and fewer migrations.

The DRS algorithm tries to get the biggest bang for its vMotion buck, *i.e.*, to minimize the total number of moves needed for load-balancing. Moving the largest, most-active VMs can have the highest impact on correcting imbalance, and hence DRS would favor such moves. While choosing such VMs for vMotion in order to issue fewer moves seemed good in theory, some customers did not like this selection since their largest, most active VMs were also their most important and performance-sensitive VMs, and vMotioning those VMs could adversely impact their performance during the migration.

To address this issue, DRS cost-benefit analysis was changed to take into account the impact of vMotion on the workload which it had not done previously. As vSphere's vMotion continued to be improved, the cost modeling of that impact required updating as well. Over time we learned that the modeling aspects of the algorithm should be separated from the parts of the algorithm that use the model, to ease the maintenance of the algorithm code as the technology changes. For example, we moved to having the algorithm consider the vMotion time, with the details of the parameters relevant to that generation of vMotion technology handled in modeling-specific code.

Earlier versions of DRS did not support affinity between VMs and hosts and it was thought that affinities between VMs should be sufficient. We also wanted administrators to think less about individual hosts and more about aggregate clusters. While VM-to-VM affinity was sufficient for most technical use-cases, there were other requirements such as software licensing that made administrators want to isolate VMs onto a set of hosts. Administrators started rolling out their own solutions to pinning VMs to a set of hosts, such as adding dummy networks to the VMs and adding the networks only to a subset of hosts, making the other hosts incompatible.

VM-to-Host rules were added to DRS in vSphere 4.1 and have been used for licensing and other use cases such as supporting availability zones. Similarly, partners asked for DRS to support ESX agent VMs, which did not need to be migrated off hosts entering maintenance or standby mode and which needed to be ready to serve on an active host before non-agent VMs could placed or migrated to that host; DRS support for agent VMs was introduced in vSphere 5.0.

Another area where administrators wanted improvement was in reporting and tracking the reasons for DRS recommendations. We added reason descriptions to the recommendations and descriptive faults for situations that prevented recommendations from being issued. Users also wanted to know how DRS evaluates the cluster as the cluster-wide entitlement of a particular VM is not always apparent. Since VM entitlement depends on the demands and resource settings of all the VMs and resource pools in the cluster, we added graphical displays of metrics to the DRS user interface, showing demand and entitlement for VMs and resource pools, as well as cluster-wide metrics including overall imbalance and the percentage of a VM's entitled resources being delivered to it.

When DPM was first introduced, the only technology it supported to bring hosts out of standby was Wake On Lan (WoL). This made some administrators uncomfortable, since WoL packets were required to be sent over the vMotion network (on the same subnet) from another host in the cluster, hence requiring at least one connected powered-on host in the cluster to power the others on. Subsequent versions of DPM supported IPMI and iLO, allowing vCenter to power-on a host by talking directly with its baseboard controller, improving adoption.

Administrators were also uncomfortable when DPM shut down many hosts even when they were idle. Based on this concern, multiple options were added to DPM to make it more conservative as well as take into account the intentions of administrators regarding how many hosts can be powered down.

A third area that was improved over several releases was the interoperability between DPM and the VMware high availability (HA) product. Since DPM reduced the number of powered-on hosts HA could choose from for restarting failing-over VMs, HA needed to convey to DPM the user's configured VM failover coverage. Spare VMs were introduced to express this information, and they prevented consolidation beyond their reservation requirements.

7. Future Directions

In this section, we highlight several of the future directions that we are actively exploring for DRS and DPM.

7.1 DRS

One important area for improvement is to expand DRS scaling for cloud-scale resource management. In vSphere 5.0, the supported maximum DRS cluster size is 32 hosts and 3000 VMs, which satisfies enterprise department deployments but falls short of cloud scale. In [32], we proposed three techniques for increasing DRS scale: hierarchical scaling (build a meta-load-balancer on top of DRS clusters), at scaling (build an overlay network on ESX hosts and run DRS as a decentralized algorithm), and statistical scaling (run DRS on a selected subset of the hosts in the cluster), and we argued for statistical scaling as being the most robust.

Another future direction is to expand the computing resources that DRS manages beyond CPU and memory. In the area of network resource management, vSphere currently supports Network I/O control [10], which runs on ESX hosts to provide shares and limits for traffic types, but there is the additional opportunity for DRS to provide cross-host network management, including vMotion to respect NIC bandwidth reservations, avoid NIC saturation, or locate communicating VMs closer together.

In the area of storage resource management, vSphere recently introduced SIOC [28] (vSphere 4.1) to manage data-store I/O contention and Storage DRS [16, 29, 31] (vSphere 5.0) to place and redistribute virtual machine disks using storage vMotion across a cluster of datastores for out-of-space avoidance and I/O load balancing, but there is the additional opportunity for DRS to support cross-host storage management, including storage vMotion perhaps coupled with vMotion to allow VM power-on placement to respect I/O bandwidth reservations.

A third future direction is to support proactive operation of DRS and DPM. Currently DRS and DPM operate reactively, with the last one hour's worth of behavior used in making its calculations of current demand more conservative. The reactive model works well in ensuring that the recommendations made by DRS and DPM are worthwhile.

However, when there is a sudden steep increase in demand, a reactive operation can result in undesirably-high latency to obtain resources (*e.g.*, time to power on a host or to reclaim memory resources from other VMs) or difficulty in obtaining the resources needed to respond while those resources are being highly contended. When such sudden steep increases in demand are predictable (*e.g.*, an 8am spike in VM usage), proactive operation can allow preparation for the demand spike to occur, to hide the latency of obtaining the resources and to avoid competing for resources with the demand spike itself.

7.2 DPM

One future direction for DPM is to revise DPM's host sorting criteria for choosing host power-on/-off candidates. The host sorting currently does not use host power efficiency, which is becoming more practical as a wider variety of hosts report server power consumption in comparable industry-standard ways. Interestingly, this has not been a frequent request from DPM users for two reasons: 1) many use hosts from the same hardware generation in a cluster, so the hosts' efficiency is similar, and 2) in clusters with mixed server generations, the newer servers are both larger and more efficient, so the existing sorting based on host size happens to coincide with host efficiency.

DPM's sorting could also consider host temperature; powering off hosts with measurably higher ambient temperature reduces the usage of hosts that could be more likely to fail.

Another future direction is to revise DPM (and DRS) to understand host and cluster power caps such as those in HP Dynamic Power Capping [1] and Intel Node Manager [3]. Power caps are a mechanism

to allow provisioning of power with respect to a specified peak value, which can limit the effective MHz capacity of the hosts governed by the cap. Given knowledge of the power cap and the power efficiency of the hosts, DPM and DRS can decide how to distribute VMs on hosts with the goal of meeting the VM's CPU resource needs in accordance with the caps.

Another area for investigation is to allow the user to tune how much consolidation is appropriate for DPM. For some users, consolidation based on CPU and memory demand is too aggressive, and they would prefer that the utilization metric be based on values derived for the VM configuration.

8. Related Work

Virtualization has introduced many interesting and challenging resource-management problems due to sharing of underlying resources and extra layers of indirection (see [26, 40, 48]). Success of a highly consolidated environment depends critically on the ability to isolate and provide predictable performance to various tenants or virtual machines. A recent study [22] performed a cost comparison of public vs. private clouds and distilled its results into the mantra *don't move to the cloud, but virtualize*. Part of the reason is the ability to better consolidate VMs in a private virtual datacenter as compared to a public cloud, yet a higher consolidation ratio requires better management of the shared infrastructure. This has led to a flurry of recent research by both industry and academia in building more efficient and scalable resource management solutions.

Many of the resource-management tasks have their roots in well-known, fundamental problems from the extensive literature on bin packing and job-scheduling. We classify the related literature in this area in two broad categories: *bin packing & job-scheduling literature* and *VM-based resource optimizations*. The first category covers classical techniques that are widely applicable in many scenarios and the second covers the algorithms and techniques specific to virtual machine based environments.

8.1 Bin Packing and Job Scheduling Algorithms

Bin packing is a well-known combinatorial NP-hard problem. Many heuristics and greedy algorithms have been proposed for this problem [17, 18, 24, 37, 47]. The simple greedy algorithm of placing the item in a bin that can take it or using a new bin if none is found, provides an approximation factor of 2. Heuristics such as first-fit decreasing (FFD) [54, 33], best-fit decreasing and MFFD [36] have led to competitive bounds of (11=9 *OPT* + 6=9) and (71=60 *OPT* + 1), where *OPT* is the number of bins given by the optimal solution.

The DRS problem is closer to multidimensional bin packing without rotation or multidimensional vector bin packing. This problem is even harder than the one-dimensional bin packing and many bounds have been proved in the literature [19, 20, 35, 38]. These bounds are less tight in most cases. Bansal and Sviridenko [20] showed that an asymptotic polynomial time approximation scheme does not exist for the two-dimensional case. Similar result has been proven for two-dimensional vector bin packing by Woeginger [51].

The problem faced by DRS and DPM is more complicated since many of the proposed techniques and lower bounds make several assumptions in terms of bin sizes and object dimensions. DRS cannot directly use such techniques because there is a cost of migration and optimizations need to be solved in an online manner. For online bin packing again, one can use algorithms such as Next-Fit, First-Fit [23] and Harmonic [41]. These algorithms are shown to have a worst-case performance ratio of 2, 1.7 and 1.636 respectively. DRS does not map directly to any such algorithm because it tries all possible one-step migrations and compares them using cost and benefit analysis. The cost-benefit analysis is very dependent on the underlying virtual machine migration technology and hypervisor overheads. But one can classify DRS as an online, greedy hill-climbing algorithm that evaluates all possible one-step moves, chooses the best one, and repeats this process after applying that move.

Fair job scheduling and fair queuing [21, 25, 55] techniques are also relevant to the DRS solution. Most of these techniques provide fairness across a single dimension or resource. In DRS, we use standard deviation as a measure of imbalance and combine standard deviation across multiple resources based on dynamic weights. These weights are again determined based on the actual resource utilization and we favor the balancing of the resource with higher utilization.

8.2 VM-based Resource Optimizations

More recently there has been considerable interest in automated resource scaling for individual virtual machines. For example, both [53] and [44] proposed a two-layered control architecture for adaptive, runtime optimization of resource allocations to co-hosted VMs in order to meet application-level performance goals. The former employs fuzzy logic and the latter adopts a feedback-control based approach.

These approaches are complementary to methods that utilize live VM migration in other work to alleviate runtime overload conditions of virtualized hosts. For example, in [39], the dynamic VM migration problem was solved using a heuristic bin packing algorithm, evaluated on a VMware-based testbed. In [52], it was discovered that using information from OS and application logs in addition to resource utilization helps the migration controller make more effective decisions.

Workload migration has also been utilized in other work to consolidate workloads onto fewer hosts in order to save power. In [27], a trace-based workload placement controller and a utilization-based migration controller were combined to minimize the number of hosts needed while meeting application quality of service requirements. In [45], a coordination framework was proposed to integrate five resource and power controllers from the processor level up to the data center level to minimize overall power consumption while preventing individual servers from being overloaded or exceeding power caps.

For private clouds, resource-management solutions like Microsoft PRO [14] provide a similar high-level functionality, but the details of their approach are not known. Similar approaches such as LBVM [5] have been proposed for Xen and OpenVZ-based virtualized environments. LBVM consists of a number of scripts that are configurable and has a very different architecture. It allows a

configurable load balancing rule per VM. It is unclear how well LBVM works in terms of overall cluster-level metrics. Neither Microsoft PRO nor LBVM support a rich resource model like DRS.

In an IaaS-based cloud many approaches have been proposed for elastic scaling and provisioning of VMs based on demand [32, 42, 46]. These techniques are complementary to DRS and can run as a service on top. Based on actual application monitoring one can either power-on more VMs (*i.e., horizontal scaling*) or change the settings of an individual VM (*i.e., vertical scaling*). DRS can then be used to place the new VMs more efficiently or to migrate VMs in order to respect new resource control settings.

9. Conclusions

In this paper, we presented the design and implementation of DRS (Distributed Resource Scheduler) along with our experience of improving it over the last five years. DRS provides automated management for a group of virtualized hosts by presenting them as a single cluster. DRS provides a very rich resource model in terms of controls such as reservations, limits, shares and resource pools in order to specify user intent in terms of performance isolation and prioritization. Various user operations like initial placement of virtual machines, load balancing based on dynamic load, power management and meeting business rules are carried out without any human intervention. We also presented the design of DPM (Distributed Power Management), which provides power savings without impacting application workloads.

Our extensive performance evaluation based on an in-house simulator and real experiments shows that DRS is able to increase CPU and memory consumption of VMs by performing automatic load balancing. Similarly, DPM is able to save power without negatively impacting running workloads. DRS has been shipping as a feature for the last six years and has been extensively deployed by our user base. Our own experiments and user feedback have led to several enhancements over the initial design. We highlighted some of these optimizations and demonstrated their benefit in improving the overall accuracy and effectiveness of DRS.

Acknowledgments

We would like to thank John Zedlewski, Ramesh Dharan, Andy Tucker, Shakari Kalyanaraman, Tahir Mobashir, Limin Wang, Tim Mann, Irfan Ahmad, Aashish Parikh, Chirag Bhatt, Ravi Soundararajan, Canturk Isci, Rajasekar Shanmugam and others for their contributions to DRS and related components in the overall system and for valuable discussions over all these years.

We are also grateful to Haoqiang Zheng, Rajesh Venkatasubramanian, Puneet Zaroo, Anil Rao, Andrei Dorofeev and others who have always listened patiently to our requests and happily provided the necessary support needed from ESX CPU and memory scheduling layers.

We are also indebted to Mike Nelson, Kit Colbert, Ali Mashtizadeh, Gabriel Tarasuk-Levin and Min Cai, for providing us with the core primitive of *vmotion* that DRS uses and enhancing it further over all these years. Without the help, support and guidance of all these people, DRS would not have been such a successful, fun and exciting project to work on.

10. References

[1] HP Power Capping and HP Dynamic Power Capping for ProLiant Servers. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01549455/c01549455.pdf.

[2] Integrated Dell Remote Access Controller (iDRAC). http://support.dell.com/support/edocs/software/smdrac3/idrac/.

[3] Node Manager – A Dynamic Approach To Managing Power In The Data Center. http://communities.intel.com/docs/DOC-4766.

[4] VMware vSphere. http://www.vmware.com/products/vsphere/.

[5] Load Balancing of Virtual Machines, 2008. http://lbvm.sourceforge.net/.

[6] VMware Fault Tolerance – Deliver 24 X 7 Availability for Critical Applications, 2009. http://www.vmware.com/files/pdf/VMware-Fault-Tolerance-FT-DS-EN.pdf.

[7] VMware High Availability: Easily Deliver High Availability for All of Your Virtual Machines, 2009. http://www.vmware.com/files/pdf/VMware-High-Availability-DS-EN.pdf.

[8] VMware vMotion: Live Migration for Virtual Machines Without Service Interruption, 2009. http://www.vmware.com/files/pdf/VMware-VMotion-DS-EN.pdf.

[9] VMware Distributed Power Management Concepts and Use, 2010. http://www.vmware.com/resources/techresources/1080.

[10] VMware Network I/O Control: Architecture, Performance, and Best Practices, 2010. http://www.vmware.com/resources/techresources/10119.

[11] VMware vCenter Server Performance and Best Practices, 2010. http://www.vmware.com/resources/techresources/10145.

[12] Advanced Configuration and Power Interface Specification, 2011. http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf.

[13] Host Power Management in vSphere 5, 2011. http://www.vmware.com/resources/techresources/10205.

[14] Microsoft Performance and Resource Optimization (PRO), 2011. http://technet.microsoft.com/en-us/library/cc917965.aspx.

[15] VMware vShield Product Family, 2011. http://vmware.com/products/vshield/.

[16] VMware vSphere Storage Distributed Resource Scheduler, 2011. http://www.vmware.com/products/datacenter-virtualization/vsphere/vsphere-storage-drs/features.html.

[17] S. Albers and M. Mitzenmacher. Average-case analyses of first fit and random fit bin packing. In *In Proc. of the Ninth Annual ACM SIAM Symposium on Discrete Algorithms*, pages 290—299, 1998.

[18] B. S. Baker and J. E. G. Coman. A tight asymptotic bound for next-fit-decreasing bin-packing. *In SIAM J. Alg. Disc. Meth.*, pages 147—152, 1981.

[19] N. Bansal, A. Caprara, and M. Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *IEEE Symposium on Foundations of Computer Science (FOCS '06)*, pages 697—708, 2006.

[20] N. Bansal and M. Sviridenko. New approximability and inapproximability results for 2-dimensional bin packing. In *In Proc. of the ACM SIAM Symposium on Discrete Algorithms,* pages 196{203, 2004.

[21] J. C. R. Bennett and H. Zhang. WF2Q: Worst-case fair weighted fair queueing. In *Proc. of INFOCOM '96*, pages 120—128, March 1996.

[22] G. Clarke. Mckinsey: Adopt the cloud, lose money: Virtualize your datacenter instead. http://www.theregister.co.uk/2009/04/15/mckinsey cloud report.

[23] E. G. Coman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-Hard Problems, D. Hochbaum (editor), PWS Publishing, Boston (1997), 46-93. A survey of worst- and average-case results for the classical one-dimensional bin packing problem,* 1997.

[24] W. F. de la Vega and G. S. Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349—355, 1981.

[25] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. Journal of *Internetworking Research and Experience*, 1(1):3{26, September 1990.

[26] U. Drepper. The cost of virtualization. *ACM Queue*, Feb. 2008.

[27] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *DSN'08*, June 2008.

[28] A. Gulati, I. Ahmad, and C. Waldspurger. PARDA: Proportionate allocation of resources for distributed storage access. In *Proc. Conference on File and Storage Technology (FAST '09)*, Feb. 2009.

[29] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar. BASIL: Automated IO load balancing across storage devices. In *Proc. Conference on File and Storage Technology (FAST '10)*, Feb. 2010.

[30] A. Gulati, A. Merchant, and P. Varman. mClock: Handling throughput variability for hypervisor IO scheduling. In *9th Symposium on Operating Systems Design and Implementation (OSDI '10)*, October 2010.

[31] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal. Pesto: Online storage performance management in virtualized datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, 2011.

[32] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad. Cloud scale resource management: Challenges and techniques. In *USENIX HotCloud'11*, June 2011.

[33] György Dósa. The tight bound of First Fit Decreasing bin-packing algorithm is FFD(I)=(11/9)OPT(I)+6/9. In *ESCAPE*, 2007.

[34] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[35] M. L. J. Csirik, J. B. G. Frenk and S. Zhang. On the multidimensional vector bin packing. In *Acta Cybernetica*, pages 361—369, 1990.

[36] D. S. Johnson and M. R. Garey. A 71/60 theorem for bin packing. *J. Complexity*, 1(1):65—106, 1985.

[37] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. *In IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 312—320, 1982.

[38] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *ACM Symposium on Theory of Computing (STOC '84)*, pages 289{298, 1984.

[39] G. Khana, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In I*EEE/IFIP NOMS'06*, April 2006.

[40] E. Kotsovinos. Virtualization: Blessing or curse? *ACM Queue*, Jan. 2011.

[41] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32:562—572, July 1985.

[42] C. Liu, B. T. Loo, and Y. Mao. Declarative automated cloud resource orchestration. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[43] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Usenix Annual Technical Conference (Usenix ATC '05)*, April 2005.

[44] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Shin. Automated control of multiple virtualized resources. In *EuroSys '09*, April 2009.

[45] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In I*nternational Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '08)*, March 2008.

[46] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[47] R. R. T. Batu and P. White. Fast approximate PCPs for multidimensional bin-packing problem. In *LNCS*, pages 245—256, 1999.

[48] W. Vogels. Beyond server consolidation. *ACM Queue*, Feb. 2008.

[49] C. A. Waldspurger. Memory resource management in VMware ESX Server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Dec. 2002.

[50] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *1st Symposium on Operating Systems Design and Implementation (OSDI '94)*, 1994.

[51] G. J. Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Inf. Process. Lett.*, 64(6):293—297, 1997.

[52] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-Box and gray-box strategies for virtual machine migration. In *Symposium on Networked Systems Design and Implementation (NSDI '07)*, April 2007.

[53] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. S. Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing Journal*, 11:213—227, 2008.

[54] M. Yue. A simple proof of the inequality FFD (L) 11/9 OPT (L) + 1, for the FFD bin-packing algorithm. In *ACTA MATHEMATICAE APPLICATAE SINICA, Volume 7, Number 4*, pages 321—331, October 1991.

[55] L. Zhang. VirtualClock: A new traffic control algorithm for packet-switched networks. *ACM Trans. Comput. Syst.*, 9(2):101{124.

Identity, Access Control, and VMware Horizon

Will Pugh
VMware
3401 Hillview Ave
Palo Alto, CA. 94309
wpugh@**vmware.com**

Kyle Austin
VMware
3401 Hillview Ave
Palo Alto, CA. 94309
kyle@**vmware.com**

Abstract

The identity landscape is changing with the move to cloud. Applications that do not change with it risk becoming unmanageable and creating difficult security holes. Two big disruptions occur when more applications run:

1. Outside the firewall as software as a service (SaaS) applications
2. On mobile devices, and many of these applications save user names and passwords

Users are adapting to these changes in ways that address the inconveniences, but not necessarily the security concerns. For example, users have no qualms about giving passwords to mobile applications, even though criminals can retrieve passwords when phones are lost. Often, users work around the need for multiple passwords for SaaS applications by reusing the same password on several sites—even though the hacking of one site means that password can be used to gain access to any of the user's other accounts.

The industry is reacting to these changes with a slew of new authentication and access control standards that are gaining traction. Such standards include Security Assertions Markup Language (SAML), Open Authorization (OAuth) 2.0, OpenID Connect, and Simple Cloud Identity Management (SCIM). This paper provides an overview of these emerging standards and discusses how they fit into the VMware Horizon™ vision of unifying application access and control.

Categories and Subject Descriptors

K.1 [**Computing Milieux**]: The Computing Industry – *Standards*.

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection – *Authentication and Unauthorized Access*.

General Terms

Management, Security, Standardization

Keywords

OAuth2, SCIM, OpenID, SAML, Token, Cloud, Authentication, Authorization

1. Introduction

VMware Horizon is a platform for managing end-user applications. This is a broad charter. The first release aimed to solve enterprise management issues around SaaS applications. These applications operate in the cloud by providers outside the corporate firewall, such as Salesforce, Mozy, Box.net, and Google Apps.

SaaS applications are gaining in popularity, in part because they allow customers to use software applications without having to manage the servers and operations around them. This allows enterprises to focus less on infrastructure maintenance and more on their core business.

At the same time, end users are seeing the productivity benefits of using their own devices. As a result, a myriad of mobile devices and personal laptops have found their way into the enterprise.

What mobile and SaaS applications have in common is that neither is joined to the corporate domain. Existing methods for managing identity and machine policy fail, and the corporate security perimeter expands past the firewall. This creates several problems.

- **SaaS applications cannot authenticate through the corporate directory.**

  SaaS applications reside outside the firewall and cannot talk directly to the corporate directory, which can be dangerous for off-boarding a fired employee. A single place no longer exists for deactivating accounts. If someone forgets to deactivate an account in Salesforce or SlideRocket, a disgruntled ex-employee still may be able to get access to those accounts.

  In addition, there is a slightly more subtle issue associated with not sharing a common directory. Users need to have passwords for each of the SaaS applications they use. Unfortunately, many users reuse the same passwords. If they reuse their Active Directory password for a SaaS application, for example, an attacker that hacks the application could reuse the password to attack the corporation. The more SaaS applications for which a user needs a password, the more places they potentially put their Active Directory password, and the more places that can be compromised.

- **No delegated authorization.**

  Kerberos allows delegation within the organization. If a service or machine wants to execute an operation on behalf of a user, it does not need to store a password. In the cloud, applications often need to store user names and passwords to do the same

thing. For example, if Zimbra wants to add a new feature that automatically creates a WebEx presentation with new meetings, they need to store a user password to have the credentials to do so. As a result, hacking Zimbra could yield access to WebEx. In addition, if the Zimbra user has administrative privileges to WebEx, Zimbra could, in theory, be able to add or remove user accounts instead of just creating new meetings.

This same issue exists with mobile devices. Many mobile applications need to maintain some authorization with a SaaS application to provide alerts, background operations, or make them easier to use. Some of these store passwords, and many phones have inconsistent protections built into them. This helps to create a bigger attack surface, and pulls much of it outside the firewall.

- **No central way to manage accounts.**

  In many traditional applications, the corporate directory manages account privileges and provisioning. Different groups in a Lightweight Directory Access Protocol (LDAP) or Active Directory server may determine whether a user has access to a particular application as well as the extent of their privileges.

1.1 Attack Surface

1.1.1 Password proliferation increases attack surface

Many passwords are flowing outside the firewall through reuse in SaaS applications and in use on mobile devices. This creates opportunities for criminals to gain access. A security problem in a small SaaS application can turn into a widespread problem due to password reuse.

Another difficult thing about passwords: if one is compromised, users must go to all of their applications and change their passwords. This can be a labor-intensive and error-prone process.

1.1.2 Manual provisioning provides additional time for fired employees to access third-party SaaS applications

Since SaaS application provisioning is not tied to the internal directory, deactivating a user in the directory does not deactivate them in SaaS applications. This can provide a window of time in which a fired employee can log into the SaaS applications. In addition, manual de-provisioning processes are labor-intensive and prone to errors.

2. Authentication Approaches

The VMware Horizon approach divides authentication and access control into two parts:

1. Authentication and entitlements required for launching applications
2. Profile and policy information required by SaaS applications for determining the permissions a user can have within an application

For example, assume Bob is a developer and has access to SlideRocket for creating internal presentations. There are two parts to enforcement. The first is letting SlideRocket know Bob is really Bob. The next part is letting SlideRocket know that Bob can only publish internally. When Bob logs into SlideRocket from his work desktop, he is redirected to VMware Horizon where his Web browser automatically uses Kerberos to authenticate with the on-premise connector. The connector creates a SAML message that authenticates Bob to the VMware Horizon service. VMware Horizon creates a SAML message that authenticates Bob to SlideRocket. SlideRocket now trusts that Bob is really Bob, yet never saw a password (and possibly Bob's Active Directory user name). This process provides a real advantage over an approach where SlideRocket has its own account for Bob that requires Bob to have a unique username and password. When Bob is disabled in Active Directory, he is unable to log into SlideRocket as well.

There is no way for SlideRocket to know that Bob should only be allowed to make internal presentations. SlideRocket has a number of policies internally to restrict who can create public presentations. VMware Horizon provides a way to tell SlideRocket what Bob is allowed to do.

The VMware Horizon solution allows application managers to map attributes from their internal directories to attributes in applications, such as SlideRocket. This provides a way to create *user profiles* for an application that are driven by the corporate directory. The application manager sets the property to allow a user to publish presentations based on their VMware Horizon or Active Directory groups. For example, assume Alice is the application manager for SlideRocket. She can decide that only people in VMware marketing groups can publish presentations externally.

When Bob was given access to SlideRocket, VMware Horizon's provisioning operations were able to run and set up his privileges automatically. If Bob moves into marketing, his privileges will be updated automatically. Alice does not have to do anything, because she already set up a rule.

2.1 Authentication

SAML is the primary standard VMware Horizon uses for exchanging security information. While SAML has been around for many years, it started gaining acceptance with SAML 1.1 and has increased traction with SAML 2.0.

SAML provides a general mechanism for exchanging information. It defines a document, a definition of assertions, and the cryptographic operations required to keep it secure. A SAML *assertion* is the core document containing a signature, issuer information, condition for use (typically only expiration date), and more. Within an assertion, SAML can include attributes, arbitrary pieces of data that can be used for authorization decisions.

The SAML standard is described in terms of *profiles* and *bindings*. Profiles describe business cases and how to use the SAML assertions and protocols to solve them. Bindings describe the transports for how assertions are passed around. Although there are many profiles in SAML, the only one commonly implemented is Web single sign-on (SSO), and is the one VMware Horizon currently supports. The most common bindings for Web SSO are HTTP POST and HTTP Redirect.

SAML defines two major roles: the Identity Provider (IDP) and the Service Provider (SP). An IDP is a service that is able to determine the identity of a user in a trusted fashion. Through SAML, the IDP can return assertions about who the user is, as well as attributes that describe the user, such as the user's cost center. The response contains an integrity-preserving signature that lets the SP know it is valid.

Typically a SP is a SaaS application that needs to have a user authenticated for it, such as Google Apps, SalesForce, or SlideRocket. These need to be setup with the keys to validate assertions. If the SP is able to initiate an authentication, it also must have an Internet Datagram Protocol (IDP) URL set.

When using VMware Horizon, there are two important flows for authentication:

1. Authentication from within the firewall
2. Authentication from outside the firewall

2.1.1 Authentication from within the firewall

When performing an authentication from within the firewall, credentials never leave the LAN, even when the hosted VMware Horizon service is used. In this scenario, VMware Horizon recognizes the Enterprise Connector as an IDP, and users are directed there to log in. The Enterprise Connector is located on premise, and understands the Kerberos and NT LAN Manager (NTLM) protocols.

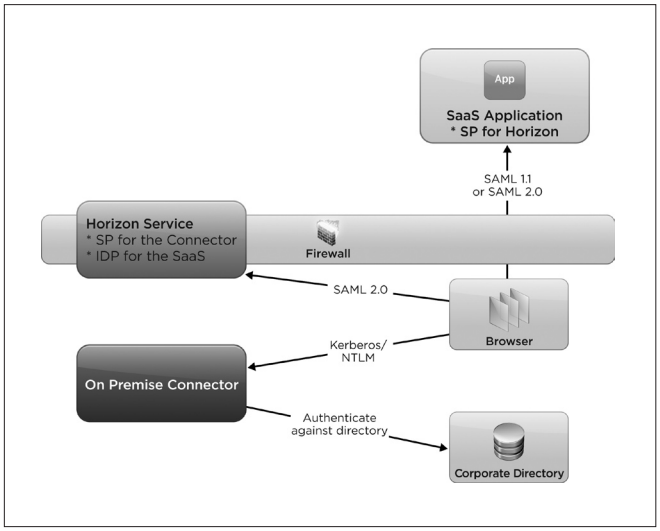

Figure 1. Authentication inside the firewall.

This provides a seamless authentication. A user's browser authenticates to the connector using Kerberos (so the user never needs to be prompted), which redirects them to VMware Horizon with a SAML 2.0 assertion. This is used by VMware Horizon to log in the user and generate a new SAML assertion for the SaaS application. At the end of the day, users get signed in to their SaaS application with no more credentials than the ones they used to log in to their desktop. Since these credentials are never directly used again, there is no way for third parties to steal them.

2.1.2 Authentication from outside the firewall

When doing an authentication outside the firewall, the browser needs to authenticate to the VMware Horizon service. In this scenario, it is important for the on-premise connector to be accessible from outside the firewall.

Users go to VMware Horizon, which redirects them to the connector for a credentialed authentication using a name and password, or SecurID. After authenticating users, the connector authenticates them to the VMware Horizon service using OAuth 2.0. The VMware Horizon service authenticates the user to the SaaS application using an appropriate authentication method supported by the application.

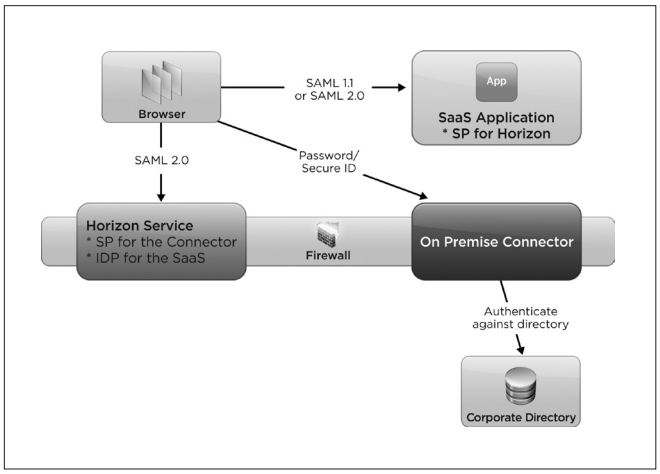

Figure 2. Authentication outside the firewall.

The advantage of this approach is that passwords never leave the enterprise. Even when using the hosted VMware Horizon service, all credentials go through the on-premise connector, so they never leave the premises.

2.2 Delegated Authentication

VMware Horizon uses SAML to authenticate a user and pass attributes to the service provider. The SAML assertion used to authenticate a user is short lived. After approximately one minute, the SAML document is considered invalid. This prevents attacks, such as replay attacks, in which users gain access to the SAML document that came across the wire and "replay" it later on to fake an authentication. This is not the only protection for replay attacks, but it is a good secondary defense.

There are a number of cases where it is important to be able to perform authentication on behalf of a user, and it is important to have a longer-lived token. This is a delegated trust scenario. A mechanism is needed for giving a third party a token and allowing them to use that for authentication. Two entities are important for VMware Horizon to delegate trust to:

1. *Mobile devices*. It would be irritating if users had to type in their name and passwords every time they used a mobile device. It would also be irritating if a user had to change their Active Directory password every time they lost a device.

2. *Other services*. Imagine if Zimbra had a feature that allowed it to create a WebEx meeting automatically when setting up a meeting in Zimbra. If Zimbra needed to store every user's WebEx or Active Directory password, it would create a security liability. Instead, OAuth 2.0 lets them store a token that can be disabled without forcing a user to change passwords. It also does not allow an exploit of one service to help determine the passwords for another service.

OAuth 2.0 solves these problems by defining flows for securely passing tokens that can be used for authentication and authorization. The main idea in OAuth 2.0 is to provide a way of exchanging tokens instead of passwords. In the example, Zimbra can send a user through a set of steps to obtain a token from Salesforce. After Zimbra gets the token, it can access Salesforce on behalf of the user.

2.2.1 Enterprise Connector

The Enterprise Connector takes this approach. It needs to be able to authenticate with VMware Horizon for performing operations such as directory synchronization.

One way to do this is to put a user name and password in the configuration. It could use this to communicate with VMware Horizon. However, doing so creates more of the same password proliferation problems VMware Horizon is trying to solve. If a user broke into a connector, he could get access to a user account in VMware Horizon.

Instead, the connector used an OAuth 2.0 token for authorization. This way, each connector has a completely different token. None of these tokens is connected in any way to any user's password. As a result, if a connector falls into the wrong hands, its token can be invalidated without having to change account passwords, or change any of the credentials on the other system.

2.2.2 Other Clients

VMware Horizon provides APIs to allow other clients to use OAuth 2.0 when trying to make programmatic calls to VMware Horizon. This standard is expected to prove to be important, and lay the foundation for a new trust model across SaaS applications. Scenarios such as the Zimbra one used in this document are becoming more prevalent. If the only way to delegate access is through saving passwords, these applications are likely to become dangerous.

3. Enforcing Application Privileges

3.1 Creating Application Attributes

In addition to whether or not a user can log in to a SaaS application, there are certain privileges within a SaaS application to which they need access. An example is whether a user can publish a presentation in SlideRocket. While this is a privilege that needs to be passed to SlideRocket to enforce, often it should be given to a user based on their participation in groups.

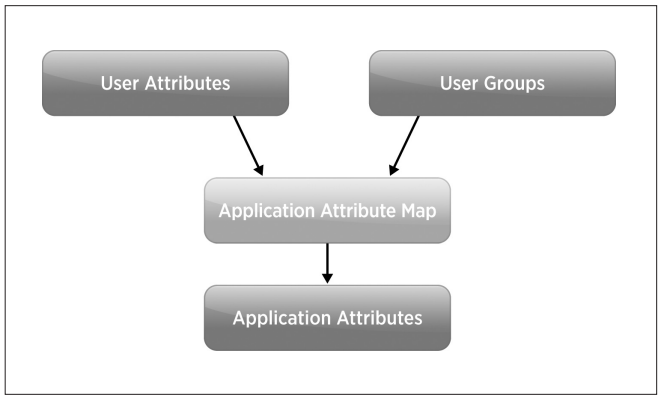

Figure 3. User Attributes and User Groups are mapped into application attributes.

VMware Horizon provides mapping and provisioning features for accomplishing this task. VMware Horizon provides attributes that can be created for an application that take a user's attributes and groups and creates a new attribute for them. In the case of SlideRocket, if the user's group is marketing it returns true, otherwise it returns false.

When these attributes change, VMware Horizon takes care to update the SaaS application with these new properties. If Bob moves into marketing, VMware Horizon's dynamic evaluation engine determines that application attributes need to be updated and a re-provision event occurs to do so.

3.2 Getting the Attributes to the Applications

One challenge of provisioning user accounts in the cloud is that there are no reasonable standards for doing so. VMware Horizon has built a generic model for creating the right user attributes for applications. Without a way to communicate the attributes to applications, they are less than useful.

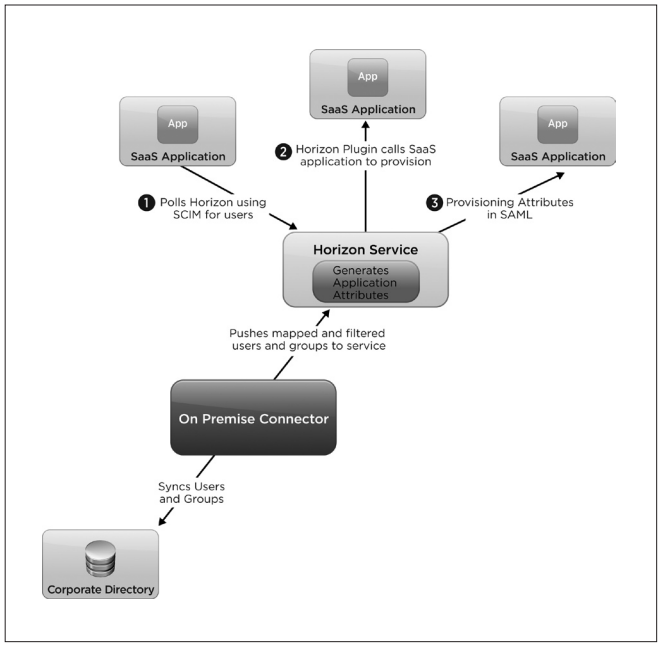

Figure 4. VMware Horizon has three ways of communicating application attributes.

VMware Horizon has three ways attributes can get to SaaS applications:

1. VMware Horizon invokes an application-specific provisioning adaptor that makes the appropriate Representational State Transfer (REST) or Simple Object Access Protocol (SOAP) calls to the SaaS application to add the account and set the attributes.
2. If the SaaS provider supports just-in-time provisioning, VMware Horizon puts the application attributes in the SAML. In this case, the application must recognize the attributes in the SAML and update their own user account. While this is popular with some providers, there is not a great answer for how to de-provision in this case.
3. VMware Horizon provides the attributes through REST API calls. This allows applications to poll for the user entitlements, and soon to poll for the application attributes as well. SCIM is an emerging standard in this space.

4. Conclusion

The security perimeter is expanding due to SaaS applications, mobile devices, and users bringing their own PCs into the corporate environment. Enterprises that do not adapt are bound to slowly increase their attack surface without really thinking about it.

VMware Horizon helps solve these issues by providing SSO and application management to SaaS applications and the newer application paradigm. This market is still young, and both the proliferation and complexity of how SaaS applications interact is expected to continue to expand. As it does so, VMware Horizon is poised to expand as well to provide management and security through open standards.

There are several important security standards emerging that solve many of these issues. VMware Horizon bridges traditional application and new cloud deployments by building on a mix of ratified and up-and-coming standards.

5. References

[1] http://saml.xml.org/saml-specifications

[2] http://www.simplecloud.info/

[3] http://oauth.net/2/

[4] http://openid.net/connect/

VMworld 2011 Hands-On Labs: Implementation and Workflow

Adam Zimman
Integration Engineering, R&D
VMware
adam@vmware.com

Clair Roberts
Integration Engineering, R&D
VMware
croberts@vmware.com

Mornay Van Der Walt
Integration Engineering, R&D
VMware
mornay@vmware.com

Abstract

At VMworld 2011 in Las Vegas, Nevada, more than 6,000 attendees completed 13,000+ labs over the course of four days, driving the creation and destruction of 145,000+ virtual machines. To say the least, it is a significant accomplishment—one that resulted from many iterations of design and years of working to deliver a constantly improving experience to end users.

Since 2004, Hands-On Labs (HOLs) have been a part of VMworld. Over the years, HOLs evolved along with VMware's technology portfolio. The goal of HOLs has always been to provide hands-on access to the latest VMware technologies in a manner that straddles the fence between technical marketing and training. The VMworld event provides a forum to share refreshed HOL content every year with some of VMware's most enthusiastic and technical users.

This paper discusses the evolution of HOL, and takes a look inside the HOL environment and the workflow service built on top of VMware's cloud infrastructure, known as LabCloud.

1. Introduction

This section introduces the high-level goals of the VMworld Hands-On Labs, as well as the technologies and people required to enable them to take place annually at the VMworld conference.

1.1 Hands-On Labs: It's All About the User

The annual VMworld conference attracts some of the most enthusiastic and engaged customers in the IT industry. The VMworld Hands-On Labs aim to provide an optimal and consistent user experience for showcasing VMware technologies. This premise continues to be the driving influence for the iteration and improvement of HOL offerings.

The conference attracts individuals with a high level of technical competence and a passion for experimenting with new VMware products and technologies.

Hands-On Labs try to give as many users as possible the opportunity to have their hands on the keyboard. This basic notion of maximizing the number of participants sets the target duration for individual labs at 60 to 90 minutes. In addition, it helps drive the continued refinement of the process of moving a user successfully through a lab, resetting the environment to a pristine state, and moving the next user through a lab. Every user that sits down to take part in a specific lab is guided through the same tasks. While users can walk away with differentiated knowledge, their paths through the lab are the same.

1.2 The HOL Virtual Army

Staging the Hands-On Labs is a herculean annual effort that pushes the boundaries of VMware technologies. An all-volunteer army participates and contributes to the effort in addition to their day-to-day responsibilities.

The sheer number of people who have contributed to the Hands-On Labs over the years is tremendous. In 2011, a virtual team of 270 contributors was led by a smaller group known as the HOL Core Team.

At a functional level, the HOL Core Team is comprised of several functional roles:

- HOL Architects - Responsible for the physical hardware, virtualization infrastructure, portal and automation services, core services, and onsite operations
- HOL Content Leads – In charge of the overarching lab scenarios and training for lab staff, as well as serving as leaders for all content development
- HOL Program Leads – In charge of program management, room flow for users, and event logistics

The effort of these individuals is magnified further by the self-imposed desire to work with pre-released code to showcase the latest VMware technologies to users. Additional groups listed rounded out HOL virtual army for VMworld 2011.

- Content Leads – Five individuals, each responsible for the oversight of a given number of labs to ensure quality and consistency (reports to the Content Manager on the Core Team)
- Lab Captains Labs – Two individuals per lab, responsible for the complete build out of the Lab Manual and Lab vPod
- Proctors for Labs – Five individuals per lab, subject matter experts responsible for assisting individual customers during the event

2. Evolution of the Hands-On Labs

This section describes the evolution of the Hands-On Labs while introducing key technologies developed by VMware.

2.1 VMworld 2004 – 2008

VMworld HOLs were first offered in 2004 as a way to provide hands-on experience with advanced concepts to the VMware user community. From 2004 through 2008, the lab experience took the form of instructor-led classrooms providing a high-touch user experience. A desirable effect of the labs, from a user perspective, was the opportunity to gain hands-on experience with sophisticated environments. The labs gave users this opportunity without the need to stage systems or spend money to build a similar environment for experimentation.

Instructor Led Labs (ILL) typically consisted of a room with seating capacity for 40 to 120 users. Each user station included a laptop and a mouse. A room was built out as its own encapsulated environment, including datacenter, isolated network, and user access points. Each lab team consisted of two to four Captains and four to eight Technical Operators responsible for the physical equipment. The Captains were responsible for building the lab environment, creating lab exercises, and leading lab sessions.

During the sessions, one or two Captains gave a guided tour of the lab while the remaining Captains circulated the room and assisted individuals. Sessions were offered only at set times and were scheduled or reserved per attendee during event hours. The frequency at which each session was offered depended on the complexity of the lab content and how long resetting the systems to a fresh state required. Resetting the environment to its initial state involved restoring snapshots on storage arrays, redeployment of images to laptops, and running scripts to stage the initial state of the environment.

The ILLs were extremely popular, with pre-registration conflicts often resulting in long waiting lines. Many individuals stood in line hoping that a pre-registered attendee would not attend.

Along with the ILL labs, simpler Self-Paced Labs (SPLs) were built using VMware Workstation and staged on individual laptops using non-persistent disks. Instructor interaction was reduced due to simplified lab topics and manuals.

Introduced in 2005, SPLs were completed by following steps in a manual with little or no interaction from a Lab Captain. Users could come and take a lab without pre-registration on a first-come, first-served basis. This resulted in constant user turnover and led to the consolidation of multiple concurrent lab topics in a single room. SPL content was limited to what could run on a laptop. After a user completed a lab, the reimaging process took 5 to 20 minutes, depending on the size of the image and the manual effort required.

2.2 Nested VMware® ESX®

Between VMworld 2008 and VMworld 2009, lab teams began experimenting with nested VMware® ESX®.[1] Nested ESX involves running ESX on top of ESX within a virtual machine. As shown in Figure 1, a typical deployment of ESX and virtual machines consists of layers 0 through 2. In a nested ESX deployment a third layer is introduced, creating one level of recursion within the nested ESX hypervisor at layer 2.

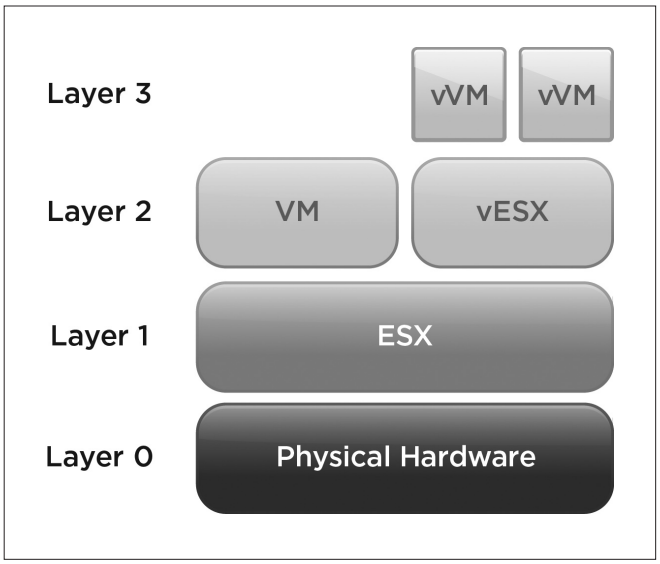

Figure 1. Block diagram showing the layers of nested virtualization.

Nested ESX provided the HOL team with three key benefits:

- A fully encapsulated virtual datacenter representing the lab environment, enabling a build-once, deploy-many methodology
- Fast and reliable lab environment resets
- A significant reduction in the underlying compute, network, and storage infrastructure, resulting in improved operational efficiencies

2.3 VMworld 2009

By VMworld 2009, the labs teams had worked out how to combine nested ESX, VMware® vCenter™ Lab Manager™, and a custom orchestration portal to create an enhanced Self-Paced Labs experience. This new format ran in addition to the tried-and-true ILL model. This was the first year the SPLs had content on par with the level of complexity of ILLs.

The combination of nested ESX and Lab Manager made it possible to deliver complex Self-Paced Labs. Lab Manager contributed three key values:

- Configurations
- Fenced networks
- Linked Clones

[1] Nested ESX and nested virtualization are discussed in listed in the references section.

Lab Manager configurations allowed a combination of virtual machines to be treated as a lab, referred to as a vPod. With nested ESX, the vPods were virtualized datacenters. Each vPod encapsulated the inner workings of a datacenter: ESX hosts, network configurations, multitiered applications, and anything else needed for a stand-alone lab environment. The fenced network functionality provided isolated networking for each vPod. This isolation ensured the user of one lab environment could not accidentally or intentionally impact the lab environment of another user. Since so much of the overall notion of the labs requires exact copies of environments, Linked Clones provided extreme storage savings and rapid deployments.

LabCloud brought an end-user portal delivering self-service lab selection, and behind-the-scenes Lab Manager orchestration. This eliminated the manual overhead of bringing labs to the user. It was the first time users could access labs on their own and have a complete lab experience at their own pace.

This new model using LabCloud on top of Lab Manager allowed SPLs to offer enterprise relevant labs in a self-paced format.

SPLs offered enterprise content side-by-side with ILLs for the first time in 2009. Seeing the two formats next to each other made the scale of the SPL model even more compelling. With just 78 seats, the SPL room completed over 4,500 labs. In comparison, ILLs utilized 10 rooms, each with 40 to 80 seats, and completed approximately 4,200 labs. The hardware scale required to run 78 SPL seats included:

- Four racks of computing hardware, providing 112 compute nodes with a total of 896 compute cores and 2.6 TB of RAM
- One storage array with 30 TB of usable storage
- Gigabit Ethernet network running the Network File System (NFS)
- One instance of Lab Manager linked to each rack and tied to its own instance of VMware® vCenter™
- 78 thin clients, each with 256MB RAM, an Internet Explorer 6 compliant Web browser, and a built-in RDP client
- LabCloud 0.1 (Beta)
  - Nine lab topics present in an HTML Web menu
  - A lab environment deployed on demand: only a 5 to 7 minute wait for the environment to be available to a user

In comparison, the ILLs completed fewer overall labs and required 10x more hardware than SPLs. The success of SPLs in the 2009 event was a pivotal turning point. The results proved that SPLs were the right direction to take to scale to the next level of user availability and access.

2.4 VMworld 2010

The labs offered in 2010 were completely self-paced, offering labs as a service (LaaS) for the first time. A large room was staged for all labs, offering 480 concurrent users access to 30 unique SPLs.

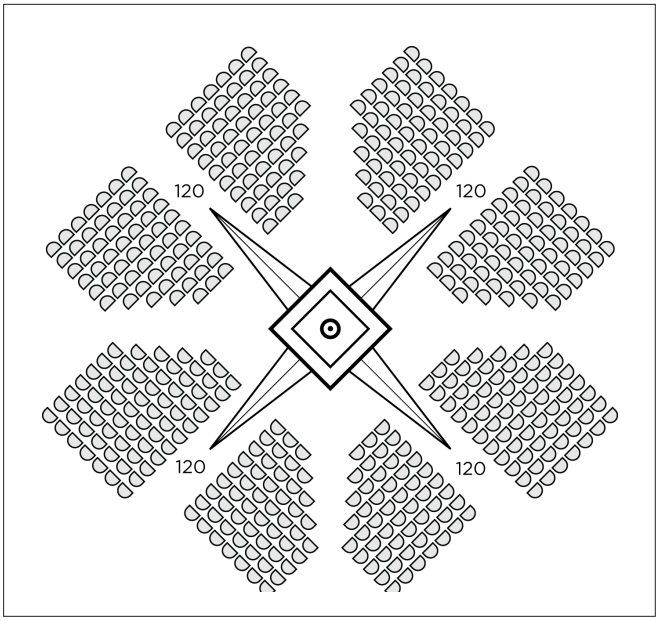

Figure 2. Clover leaf 480-seat room layout

Looking at the environment from a high level, each user station was built with a PC-over-IP (PCoIP) zero client, keyboard, mouse, and two monitors. Each zero client was connected to the cloud for access to a menu of the 30 lab topics. The topics were displayed in an interactive menu as part of the HOL user portal. The portal ran inside a VMware® View™ desktop residing in one of three datacenters. Two of the datacenters were located on the eastern seaboard, with the third onsite in San Francisco at the event. Users were unaware which datacenter served their environment, as the user experience was the same from each location.

The LaaS approach allowed for an exponential growth in HOLs consumed by the user community, resulting in over 15,000 completed HOLs by the close of VMworld 2010. Everything from the physical user environment to the logical layout of the compute infrastructure experienced a massive increase in scaling. At a high level, the following items supported the environment:

- 13 racks of computing hardware with a total of 416 compute nodes, a total of 3,328 compute cores, and 19.9TB of RAM
- Each rack was linked to one instance of Lab Manager (13 total), which in turn was tied to its own instance of VMware vCenter (13 total)
- Six storage arrays with 180TB of usable storage
- A converged fabric NFS network running a combination of 10 Gigabit Ethernet and InfiniBand
- Two remote datacenter locations, one on premises datacenter, with public Internet connections to remote sites
- 480 PCoIP hardware-enabled zero clients

- VMware® View™ Security Server across the public Internet to remote sites[2].
- LabCloud 1.0
  - Converted client interface from HTML to Flex UI
  - 30 Lab topics
  - Pre-population of lab environments to provide instant provisioning
  - Help Request feature
  - Built-in, online satisfaction surveys
  - Dynamic overhead statistics screens
  - Seat map and seating waitlist application to facilitate user flow in the large room

The scaling of the self-paced environment was a tremendous success. The overall capacity of labs completed increased by more than 2x while utilizing less physical space and hardware.

In many ways, 2010 marked a turning point for the labs, particularly with respect to the user experience. The 480 seat, complete self-paced model was determined to be an optimal physical configuration for lab delivery. The user interaction model was deemed to need only minor enhancements, with the majority of functionality staying the same as the previous year. The real heavy lifting for 2011 took place in two key areas: lab content and a major overhaul to LabCloud.

2.5 VMworld 2011

In 2011, there was a conscious effort to revamp the way in which content was conceived and presented to lab participants. It was the first time an overarching scenario or background story was developed and used in all of the individual labs to tie the content together. This helped to force the individual labs to be oriented to business IT solutions rather than point products or features. The team believed the VMware user community would be more engaged learning solutions to business IT needs rather than simply learning how to install and configure VMware software products.

Scenario-driven labs had an unexpected impact. The average time to complete a lab in 2010 was 45 to 60 minutes. In 2011, the average time to complete an HOL ranged from 60 to 90 minutes. This resulted in a slight decrease in the total number of labs completed.

From a LabCloud perspective, the majority work involved migrating the interface from orchestrating Lab Manager to VMware vCloud® Director™. The decision to perform this migration was based on the desire to move to released technologies from VMware and take advantage of the benefits of vCloud Director.

For comparison, the following items supported the 2011 environment:

- Eight racks of computing hardware with a total of 544 compute nodes, a total of 3,520 compute cores, and 29.95TB of RAM
- Six storage arrays with 1 petabyte (PB) of usable storage
- A converged fabric NFS network running 10 Gigabit Ethernet
- Each datacenter was linked to one instance of vCloud Director (3 total),
- 16 VMware vCenter instances
  - Three management zone deployments
  - Three View deployments
  - 13 compute node deployments
- Three remote datacenter locations with public Internet connections to remote sites
- 480 PCoIP hardware-enabled zero clients
- VMware View Security Server across the public Internet to remote sites
- LabCloud 1.5
  - Evolved to orchestrate vCloud Director instead of Lab Manager

YEAR	NUMBER OF LAB TOPICS	NUMBER OF SELF-PACED LABS	NUMBER OF LAB SEAT HOURS	NUMBER OF VIRTUAL MACHINES CREATED	NUMBER OF LABS COMPLETED
2004*	3	1	400	~1,200	400
2005*	8	1	1,200	~3,600	1,200
2006*	12	2	1,600	~4,000	1,600
2007*	15	4	~2,000	~7,000	~2,000
2008*	15	5	~3,000	~12,000	~3,000
2009*	23	11	~9,200	~62,000	~9,200
2010	30	30	21,120	145,097	15,344
2011	31**	31	24,000	144,083	13,056

Figure 3. Table showing progression of the scale of the labs offered at VMworld from 2004 through 2011.

*Some numbers are approximations due to a lack of hard data.

**While the number of distinct labs was 27, two of the labs were offered in German and Japanese, bringing the total number of choices to 31.

3. A Peek Under the Covers

This section dives deeper into the architecture of the environment and explores the various layers, from the datacenter to the user experience.

3.1 Software Technologies Used in 2011

In 2011, the infrastructure and architecture for servicing the HOLs were enhanced with the addition of several VMware products and technologies:

- VMware vSphere® 5
- VMware® vShield™ 5
- VMware vCloud Director 1.5
- VMware View 5
- VMware® vCenter™ Operations Manager™ 5

[2] The first time this software was tested at scale was at this event.

3.2 The vPod

Earlier in this paper, the vPod was introduced as an encapsulated virtual datacenter. Key motivations for using vPods as the constructs for the labs included the ability to group and coordinate multiple virtual machines, isolation, and reproducibility.

For VMworld 2011, vPods consisted of the following basic components:

- Control center virtual machine – provided an inbound landing zone for users to enter the vPod and access its resources
- VMware® vShield Edge™ virtual machine – provided fire-wall services, network address translation (NAT) services, DHCP, and inbound port forwarding to the control center
- Lab-specific virtual machines – varied based on the needs of the lab content (such as VMware vCenter, View, databases, and so on)
- Nested ESX virtual machines – more advanced labs often included ESX virtual machines with nested virtual machines
- VMware vSphere® Storage Appliance virtual machines – a storage appliance virtual machine that provided NFS exports for the ESX virtual machines to mount as data stores for nested virtual machines.

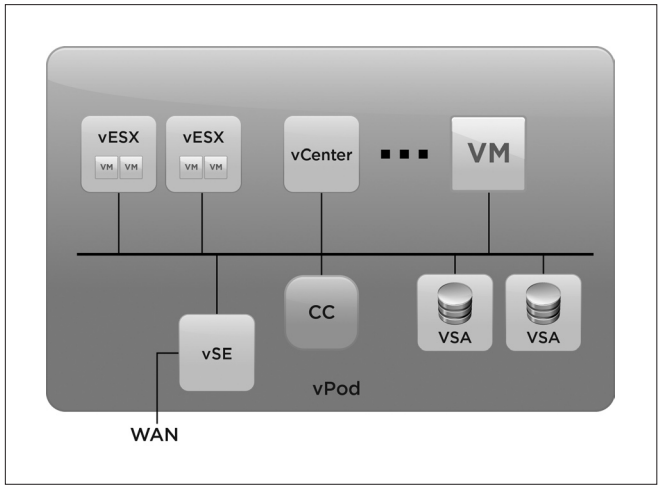

Figure 4. vPod block diagram

3.3 High Availability Designs

The HOL architecture expects failures to occur. It is not a question of if something will fail, but *when* it will fail. In an ideal world, the HOL environment would experience zero failures or interruptions. In the real world, the unexpected happens every day. Over the years, VMware teams have witnessed many unexpected events, from equipment falling off trucks and 400 amp circuit breakers blowing, to forklifts running into servers and unexpected software and hardware failures.

The implication of these events is that no single point of failure can exist that allows for a complete outage of any service. There can be one of nothing. This design strategy has been applied end-to-end, from the hardware to application workloads. Over the years, this practice has been refined based on lessons learned and the realities of the corporate world.

The goal of graceful service degradation does not mean that services never go offline. In fact, a portion of all services always remain *online*. Capacity can be diminished due to incremental failures, but service interruption does not occur. The justification of this approach: it is better to run fewer stations rather than no stations.

This practice translates to redundancy in all blades, power and network connections, switches, storage controllers, storage disks, and so on. Logical groupings span multiple components to avoid any single point of failure. Blade chassis are mapped logically across frames to ensure entire racks are not lost if one blade chassis fails. The strategy is carried through the software stack with the implementation of multiple Domain Name Service (DNS) servers, Active Directory servers, load balancers, vCloud Director cells, and so on. The architecture includes multiple VCs, each associated with segmented portions of the environment such that one instance is not at risk of taking down the entire infrastructure.

3.4 LabCloud

The LabCloud solution delivers the HOL experience. It is comprised of a few components that provide the following functionality:

- End-user portal – provides the opportunity to view and select available labs, presents the lab dashboard environment (lab timer, lab manual, and an automatically connecting RDP session)
- Infrastructure API bridges – the command and control orchestration component that dictates the when, where, and how many to send to the underlying infrastructure
- Workload placement engine – a component that load-balances the *where* aspect of the lab placement on the infrastructure based on what is currently running in the environment
- Station state machine - manages the state machine for each station, ensuring each station adheres to the well-defined attendee use case workflow at all times
- Room flow tools – a series of tools that assist with the overall flow of users throughout the room (waiting list, seating map, and help desk)

From an implementation perspective, LabCloud was written primarily in Python and leverages the Django framework. In recent years, it was extended to include Flex client-side components. A Postgres database provides backend data persistence. The basic structure of the LabCloud solution is built on the following components:

- Database – tracks all of the objects and statistics of the system
- Adobe Message Framework (AMF) – provides messaging between LabCloud server-side and client-side components
- Django – provides a server-side model, view, and controller framework
- Apache – serves static content and executes LabCloud workflows

These component layers work together to support the orchestration of the underlying infrastructure.

4. Demystifying the HOL Station Workflow

The lab station workflow is analogous to the workflow found in a restaurant. Customers enter a room and are shown to their seat. They peruse the menu, make a selection, receive their food, consume and enjoy. Once they are done, they get up and leave.

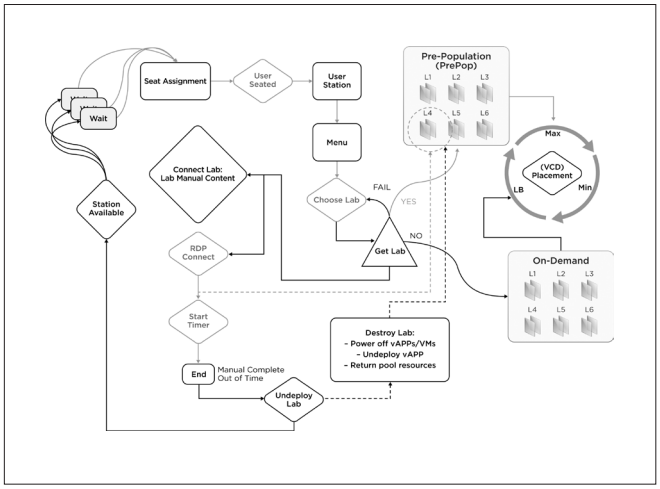

Figure 5. HOL Workflow

Finally, the host cleans up after them. In both endeavors the goal is the same, rotate many customers through the seats as possible in a fixed amount of time.

Starting from the beginning: The initial state is a waiting user or, a wait, state. When a seat is available, it is assigned to the next user in line. The user receives a *seat assignment*. For VMworld HOLs, users are taken to an assigned seat where the user portal is ready for their personal identification. Once at the user station, *user seated*, the user can review the menu and select a lab. Once a selection is made, *choose lab*, the user waits for the HOL to be presented via the user portal.

Getting the selected lab: The *get lab* logic checks to see if any instances of the selected lab are available in the pre-populated *pre-pop* pool. If an instance is available, it is assigned immediately to the user. This best-case scenario offers the user instant gratification, as the lab is presented within a few seconds. In the event a request cannot be serviced from the *pre-pop* pool, an *on-demand* provisioning of the selected HOL is triggered.

Pre-pop pool: The pre-pop pool is a repository of pre-built HOL instances that are staged prior to user requests being placed. The popularity of a given HOL and the number of pre-pops (pre-populated HOL instances) is managed, dictated, and monitored by the LabCloud administrator. Ideally, all requests are served from the *pre-pop* pool. This allows user requests to be serviced within seconds instead of minutes. However, space in the *pre-pop* pool is expensive, as each fully running instance consumes compute resources on the system, including CPU, memory, storage, and network cycles. These are wasted resources from an operational perspective—the systems are running but are not servicing any users. This balance can make it difficult to predict which HOLs are likely to be the most popular and require the most pre-pops. Over time, trends begin to emerge that allow this to become more predictable. LabCloud is built to replenish the *pre-pop* pool automatically after values drop below a predetermined minimum threshold.

On-demand request: *On-demand* creation is significantly less expensive from an operations perspective. No resources are consumed when the lab is not in use. There is, however, a significant penalty to users in the form of wait time if an *on-demand* HOL creation is triggered. The deployment process can range from three to nine minutes when the environment is not under load. This can be longer if the storage controllers are hard at work. While this does not seem slow in the grand scheme of things, it is a significant wait compared to the ideal case of the request being serviced from the *pre-pop* pool. In comparison, it is interesting to consider that an HOL is a fully encapsulated, virtual datacenter. In a pre-virtualization world, the effort to set up the services represented in this fully encapsulated virtual datacenter could take weeks to months.

Connection to the HOL manual and content: Once the *get lab* event is serviced, the user portal immediately establishes a connection to the manual and begins to establish an RDP session. If the lab needs to be deployed and the RDP session is not ready, the user receives a notification message that the lab is being deployed and will be visible upon completion.

RDP session: The RDP session is the visible entry point into the lab environment.

Start Timer: Once the lab content is delivered, a timer starts that limits the amount of time the user has to complete the lab.

Clean-up time: Once the lab ends—the user indicates the lab is complete or the timer expires—the *undeploy lab* process initiates. The provisioned lab is undeployed and the seat is made available.

Station available: One the seat is available, the entire process starts again.

5. VMworld 2011 HOL by the Numbers

By adopting a LaaS model with LabCloud and a geographically distributed VMware powered IaaS public cloud, the HOL team was able to continue to innovate and provide a very rich HOL user experience to VMworld attendees. In addition, the user community was able to complete significantly more HOLs during VMworld 2011 than in 2004 through 2009. This is a testament to the tremendous scale, agility, resiliency—and most importantly consumer and customer satisfaction—that can be realized when coupling a LaaS model with a VMware powered IaaS cloud.

VMWORLD 2011 HANDS ON LABS BY THE NUMBERS	
Number of HOL in the VCD service catalog	31
Number of HOL hours	50
Number of HOL seats	480
Number of lab seat hours	24,000
Total labs deployed	13,056
Total IVMs deployed and destroyed	144,083
Rate of churn; 1 VM created every 1.25 seconds	

Figure 6. Overview of statistics from the VMworld 2011 Hands-On Labs

6. Future Work

Project Fiji is an ongoing project that involves the redesign and rewrite of the LabCloud software. The motivation behind this effort is two-fold:

- Drive deeper integration with the VMware® vFabric™ family of products to provide enhanced resiliency and HOL workflow automation.
- Drive an enhanced HOL user experience over previous years through the integration of the next-generation LabCloud and VMware technologies.

7. Conclusions

The VMworld HOLs have come a long way since their debut at the VMworld 2004 conference. The evolution and advancement of VMware technology has contributed greatly to the evolution of the HOLs over the years. Purpose-built software, such as LabCloud integrated with VMware cloud infrastructure and other products, has taken HOLs to the next level and truly enabled the notion of LaaS.

While this paper focused on the VMworld 2011 HOLs, the HOL Core Team started work in early January 2012 on plans for the VMworld 2012 HOL. The plan is ambitious , and the team is looking to push the boundaries even further by integrating more of VMware's extensive portfolio into the end-to-end HOL solution. In the end, it is all about providing the best possible HOL experience for VMware's most enthusiastic and technical users.

Acknowledgments

This paper is dedicated to the memory of Fred Stark (June 25, 1968 to September 27, 2011). Fred was appointed the HOL Content Manager for 2011, and was the driving force behind the scenario based HOLs at VMworld 2011. Fred, you will be missed, but never forgotten.

VMworld HOLs would not be possible if it was not for the sacrifices of those who volunteered their time and technical expertise while still taking care of their daily responsibilities. It is a long list, far too many to list individually. More importantly, you know who you are—thank you!

In addition to the HOL Virtual Army, the authors would like to acknowledge the members of the VMware Partner Ecosystem who have sponsored the VMworld HOLs over the years.

References

[1] Bounding the Running Time of Interrupt and Exception Forwarding in Recursive Virtualization for the x86 Architecture; Wing-Chi Poon (VMware) wpoon@vmware.com, Aloysius K. Mok (UT Austin) mok@cs.utexas.edu; Technical Report VMware-TR-2010-003, Oct 20th 2010

[2] Keith Adams, Ole Agesen "A Comparison of Software and Hardware Techniques for x86 Virtualization", 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Oct 2006, San Jose, California, USA

vmware®

VMWARE ACADEMIC PROGRAMS

VMware has created rich global education and research offerings with programs to allow access and support to the latest virtualization technologies for both research and instruction purposes:

VMware Academic Programs
If you represent an academic institution, the VMware's Academic Programs group can provide access to cutting-edge virtualization and cloud technology, along with access to experts for research and teaching purposes.

- Research Funding
- Conference Sponsorships
- Research Collaboration
- Guest Lectures
- Scholar in Residence Program
- VMware Graduate Fellowship

Academic Affiliate Partners

- Algorithms, Machines, People Laboratory (AMP) Laboratory at the University of California, Berkeley
- Parallel Data Laboratory (PDL) at Carnegie Mellon University
- Center for Experimental Research in Computer Systems (CERCS) at Georgia Institute of Technology
- Computer Science and Artificial Intelligence Laboratory (CSAIL) at Massachusetts Institute of Technology

Visit labs.vmware.com to learn more about VMware's education and research opportunities.

SPECIAL THANKS

Stephen Alan Herrod, Ph.D.
Chief Technology Officer and Senior Vice President of Research & Development

Julia Austin
Vice President Innovation Programs

Questions and Comments can be sent to vmtj@vmware.com

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com