Red Hat Reference Architecture Series

Scale-up Virtualization with Red Hat Enterprise Linux 5.4 on an HP ProLiant DL785 G6 (= 256 guests)

Multi-Tile, Consolidated Workload
Red Hat Enterprise Linux 5.4 Guests
Red Hat Enterprise Linux 5.4 Host (with Integrated KVM Hypervisor)
HP ProLiant DL785 G6 (8 x 6 Core/Socket = 48 AMD Opteron™ 8439SE Processor Cores)
HP StorageWorks MSA 2300fc + EVA 4400

Version 1.0
December 2009

redhat.

Scale-up Virtualization using Red Hat Enterprise Linux 5.4 on an HP ProLiant DL785 G6 (= 256 guests)

1801 Varsity Drive
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Table of Contents

Acronyms and Definitions

A short summary of the acronyms and definitions used in this document:

API	Application Programming Interface
BBWC	Battery Backed Write-back Cache
DAS	Direct Attached Storage
DHCP	Dynamic Host Configuration Protocol
Guest	An instance of a virtual machine
Host	The physical server and operating system hosting the virtual machines
KVM	Kernel Virtual Machine
LUN	A logical representation of one or more physical disks
LVM	Logical Volume Manager v2
NAT	Network Address Translation
OS	Operating System
RAID	Redundant Array of Independent Disks
RHEL 5.4 AP	Red Hat Enterprise Linux 5.4 Advanced Platform
SAN	Storage Area Network
VM	Virtual Machine
Hypervisor	A computer software/hardware platform virtualization software that allows multiple operating systems to run on a host computer concurrently
vCPU	Virtual CPU, i.e the CPU seen in the virtual machine guest
vNIC	Virtual Network Interface Card (NIC), managed by the virtual maching guest operating system
vMem	The memory in the physical host that has been allocated to and is in use by the virtual machine guest
KSM	Kernel Same-Page Merging – a technology enabling memory page sharing and thus permitting memory over-commitment for RHEL 5.4/KVM
vLUN	Virtual LUN
virt_io driver	The paravirtualized I/O driver for Linux based virtual machine guests
NUMA	Non Uniform Memory Access

1 Executive Summary

HP and Red Hat collaborate at an engineering level to ensure that our customers benefit from software and hardware solutions that are jointly tested, certified and tuned for optimal performance. HP and Red Hat have developed a variety of recommended configurations for Red Hat Enterprise Linux applications, which are appropriate for particular business and technical situations.

The goal of this study was to demonstrate the scale-up capabilities of the Red Hat Enterprise Virtualization environment – Red Hat Enterprise Linux (RHEL) 5.4 with the integrated Kernel-based Virtual Machine (KVM) hypervisor.

The configured environment successfully and **linearly scaled up to 256 virtual machine guests** (32 ”tiles” each comprising 8 virtual machines) on a single HP ProLiant DL 785 G6 server. The “Aggregate Throughput” graph below illustrates the linearity of the guest scalability while the “Average Response Time” graph below illustrates the ability of the platform to consistently maintain an average response time of under 1 second all the way to 256 simultaneously running guest instances. In the throughput graph, each color represents the throughput contribution of different tiles. Note that the KVM hypervisor enforces fair scheduling, i.e., for any configuration of “n” tiles, each tile generates ~1/n^{th} the total throughput. These graphs clearly demonstrates that host hypervisor (KVM) scaled up to the increased load by running all of the 256 guest instances while **fairly scheduling each of the guest workloads** and **ensuring reasonable response times** for externally connected clients utilizing the resources of the platform.

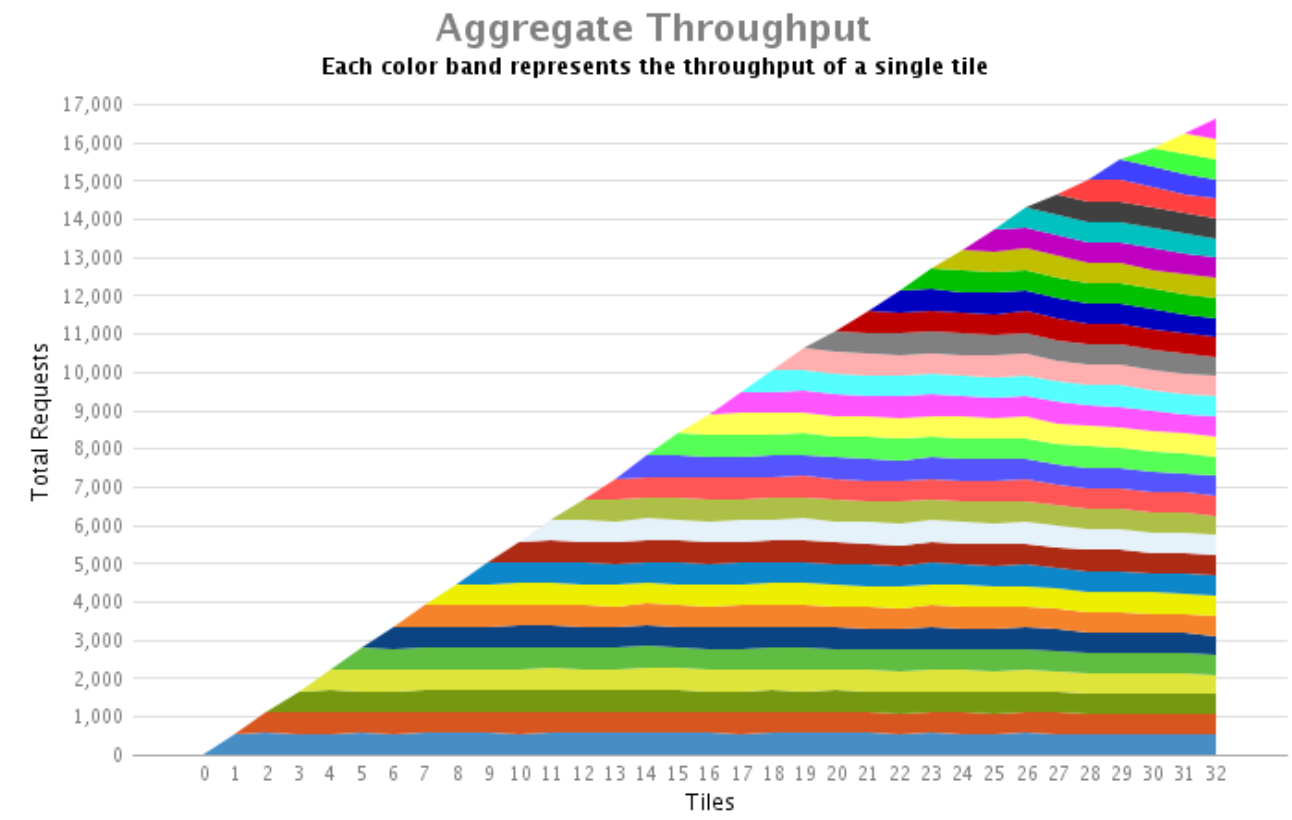

Figure 1: Scaling from 1 Tile (= 8 guests) through 32 Tiles (= 256 guests)

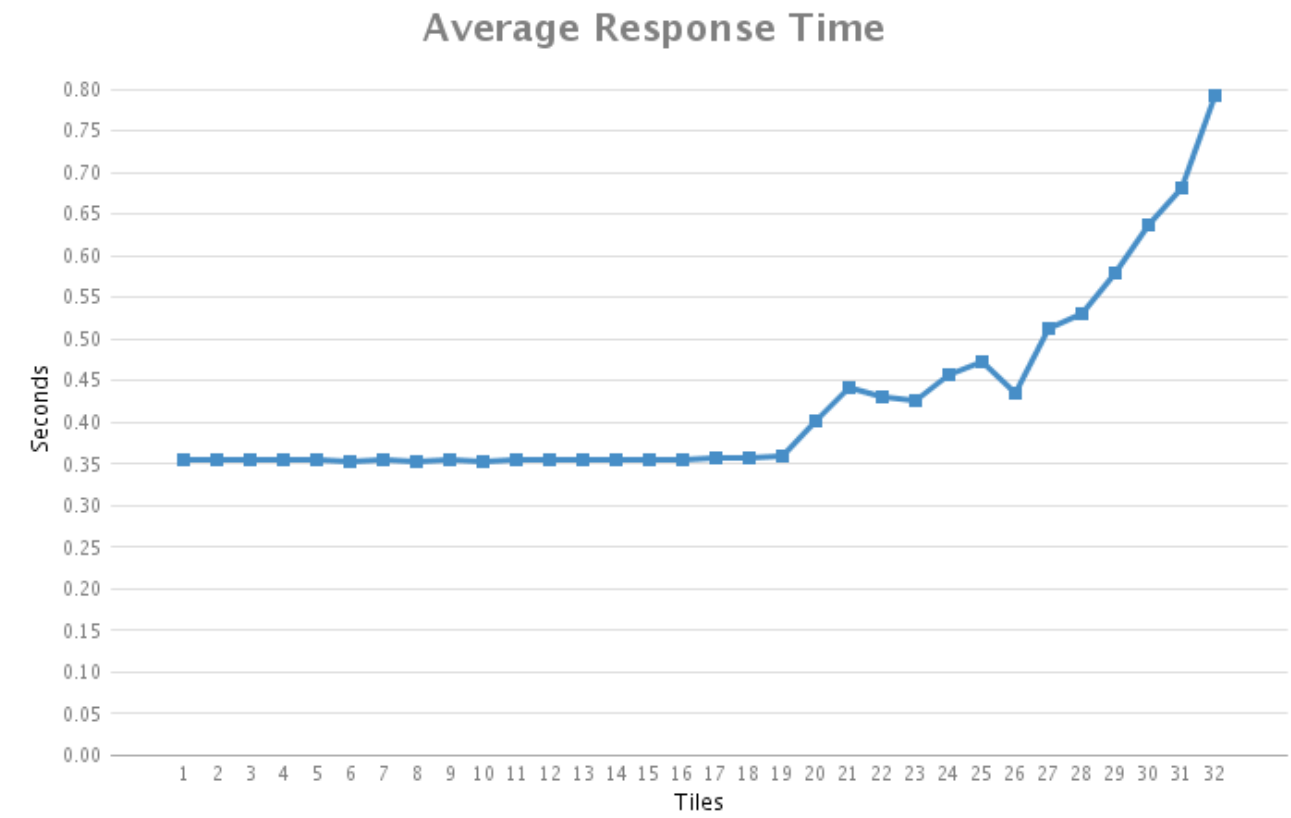

Figure 2: Scaling from 1 Tile (= 8 guests) through 32 Tiles (= 256 guests)

2 Red Hat Enterprise Virtualization – Overview

2.1 Red Hat Enterprise Virtualization - Portfolio

Server virtualization offers tremendous benefits for enterprise IT organizations – server consolidation, hardware abstraction, and internal clouds deliver a high degree of operational efficiency. However, today server virtualization is not used pervasively in the production enterprise data center. Some of the barriers preventing wide-spread adoption of existing proprietary virtualization solutions are performance, scalability, security, cost, and ecosystem challenges.

The Red Hat Enterprise Virtualization portfolio is an end-to-end virtualization solution, with use cases for both servers and desktops, designed to overcome these challenges, enable pervasive data center virtualization, and unlock unprecedented capital and operational efficiency. The Red Hat Enterprise Virtualization portfolio builds upon the Red Hat Enterprise Linux platform that is trusted by thousands of organizations on millions of systems around the world for their most mission-critical workloads. Combined with KVM, the latest generation of virtualization technology, Red Hat Enterprise Virtualization delivers a secure, robust virtualization platform with unmatched performance and scalability for Red Hat Enterprise Linux and Windows guests.

Red Hat Enterprise Virtualization consists of the following server-focused products:

1. Red Hat Enterprise Virtualization Manager (RHEV-M) for Servers: A feature-rich server virtualization management system that provides advanced management capabilities for hosts and guests, including high availability, live migration, storage management, system scheduler, and more.

2. A modern hypervisor based on KVM (Kernel-based Virtualization Machine) which can be deployed either as:

    - Red Hat Enterprise Virtualization Hypervisor (RHEV-H): A standalone, small footprint, high performance, secure hypervisor based on the Red Hat Enterprise Linux kernel.

    Or

    - Red Hat Enterprise Linux 5.4: The latest Red Hat Enterprise Linux platform release that integrates KVM hypervisor technology, allowing customers to increase their operational and capital efficiency by leveraging the same hosts to run both native Red Hat Enterprise Linux applications and virtual machines running supported guest operating systems.

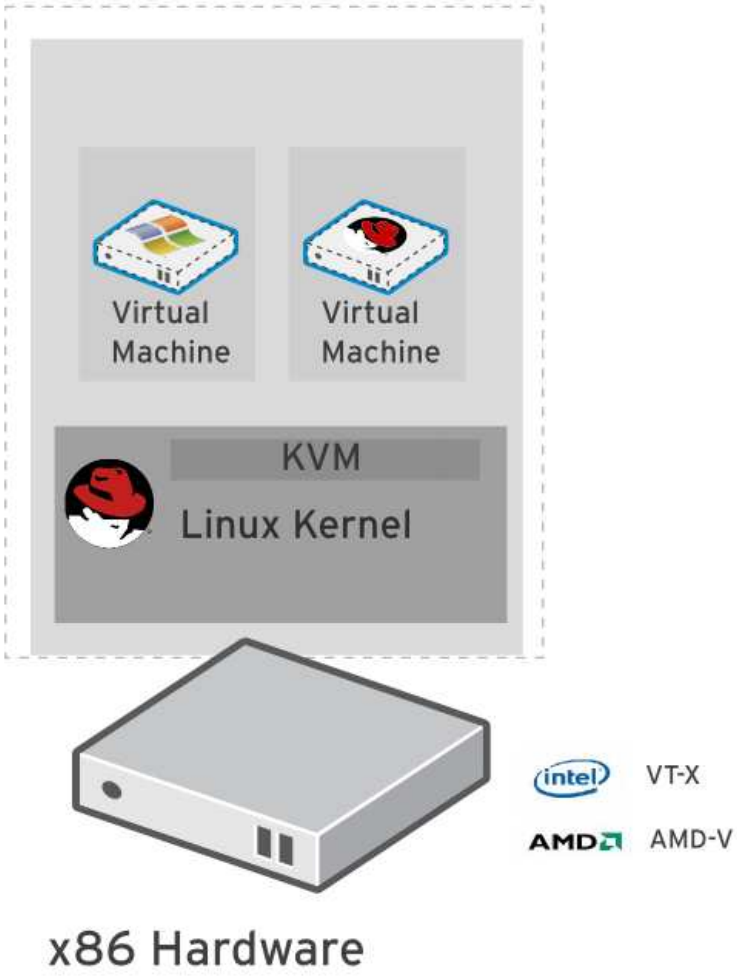

Figure 3: Red Hat Enterprise Virtualization Hypervisor

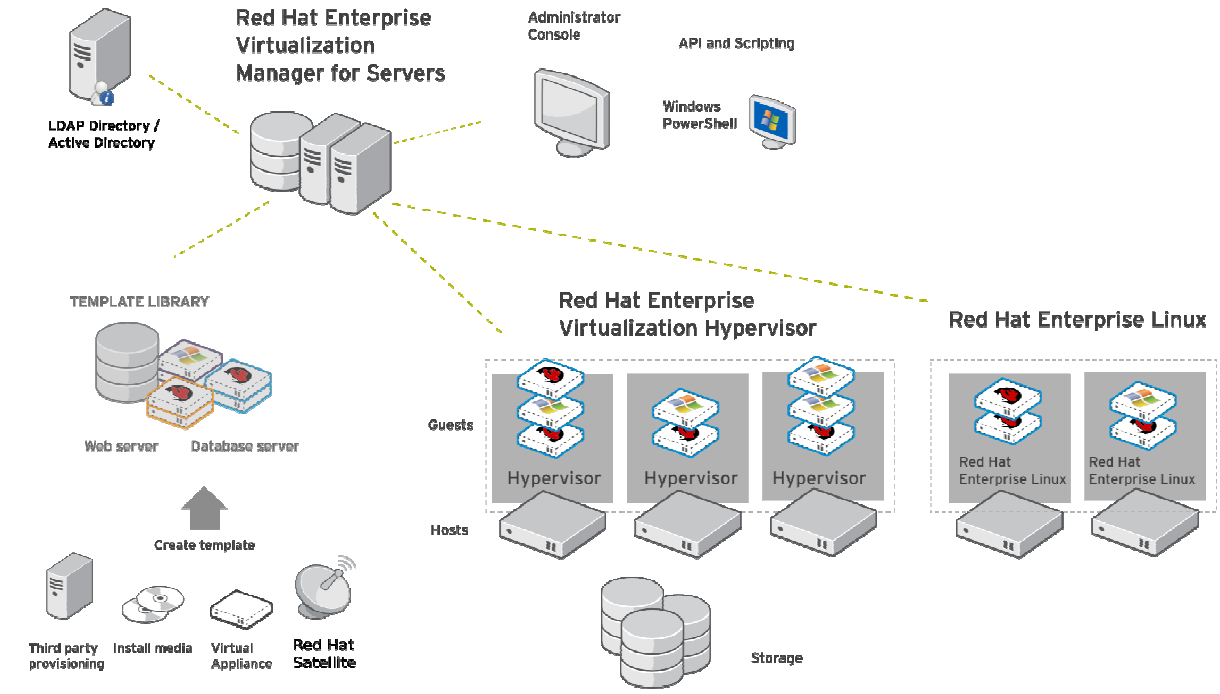

Figure 4: Red Hat Enterprise Virtualization for Servers

2.2 Kernel-based Virtualization Machine (KVM)

A hypervisor, also called virtual machine monitor (VMM), is a computer software platform that allows multiple ("guest") operating systems to run concurrently on a host computer. The guest virtual machines interact with the hypervisor that translates guest I/O and memory requests into corresponding requests for resources on the host computer.

Running fully virtualized guests, i.e., guests with unmodified guest operating systems, used to require complex hypervisors and previously incurred a performance penalty for emulation and translation of I/O and memory requests.

Over the last few years chip vendors Intel and AMD have been steadily adding CPU features that offer hardware enhancements to support virtualization. Most notable are:

1. First-generation hardware assisted virtualization: Removes the requirement for hypervisor to scan and rewrite privileged kernel instructions using Intel VT (Virtualization Technology) and AMD's SVM (Secure Virtual Machine) technology.

2. Second-generation hardware assisted virtualization: Offloads virtual to physical memory address translation to CPU/chip-set using Intel EPT (Extended Page

Tables) and AMD RVI (Rapid Virtualization Indexing) technology. This provides significant reduction in memory address translation overhead in virtualized environments.

3. Third-generation hardware assisted virtualization: Allows PCI I/O devices to be attached directly to virtual machines using Intel VT-d (Virtualization Technology for directed I/O) and AMD IOMMU. Also, SR-IOV (Single Root I/O Virtualization) which allows special PCI devices to be split into multiple virtual devices. This provides significant improvement in guest I/O performance.

The great interest in virtualization has led to the creation of several different hypervisors. However, many of these pre-date hardware-assisted virtualization, and are therefore some-what complex pieces of software. With the advent of the above hardware extensions, writing a hypervisor has become significantly easier and it is now possible to enjoy the benefits of virtualization while leveraging existing open source achievements to date.

Kernel-based Virtual Machine (KVM) turns Linux into a hypervisor. Red Hat Enterprise Linux 5.4 provides the first commercial-strength implementation of KVM, which is developed as part of the upstream Linux community.

2.2.1 Traditional Hypervisor Model

The traditional hypervisor model consists of a software layer that multiplexes the hardware among several guest operating systems. The hypervisor performs basic scheduling and memory management, and typically delegates management and I/O functions to a special, privileged guest.

Today's hardware, however, is becoming increasingly complex. The so-called “basic” scheduling operations have to take into account multiple hardware threads on a core, multiple cores on a socket, and multiple sockets on a system. Similarly, on-chip memory controllers require that memory management take into effect the Non-Uniform Memory Access (NUMA) characteristics of a system. While great effort is invested into adding these capabilities to hypervisors, we already have a mature scheduler and memory management system that handles these issues very well – the Linux kernel.

2.2.2 Linux as a Hypervisor

By adding virtualization capabilities to a standard Linux kernel, we can enjoy all the fine-tuning work that has gone (and is going) into the kernel, and bring that benefit into a virtualized environment. Under this model, every virtual machine is a regular Linux process scheduled by the standard Linux scheduler. Its memory is allocated by the Linux memory allocator, with its knowledge of NUMA and integration into the scheduler.

By integrating into the kernel, the KVM 'hypervisor' automatically tracks the latest

hardware and scalability features without additional effort.

2.2.3 KVM Summary

Leveraging new silicon capabilities, the KVM model introduces an approach to virtualization that is fully aligned with the Linux architecture and all of its latest achievements. Furthermore, integrating the hypervisor capabilities into a host Linux kernel as a loadable module simplifies management and improves performance in virtualized environments, while minimizing impact on existing systems.

An important feature of any Red Hat Enterprise Linux update is that kernel and user APIs are unchanged, so that Red Hat Enterprise Linux 5 applications are not required to be re-built or re-certified. This extends to virtualized environments: with a fully integrated hypervisor, the application binary interface (ABI) consistency offered by Red Hat Enterprise Linux means that applications certified to run on Red Hat Enterprise Linux on physical machines are also certified when run in KVM virtual machines. With this, the portfolio of thousands of certified applications for Red Hat Enterprise Linux applies to both environments.

3 Workload Description

3.1 Concept of a Tile

The general concept of a virtualization consolidation benchmark is shown in Figure 5. In this kind of benchmark **the consolidation of multiple physical servers is modeled by having a set of virtual servers (or guests), all on a single physical server, with the guests running the heterogeneous workloads which ran on the original unconsolidated physical servers.**

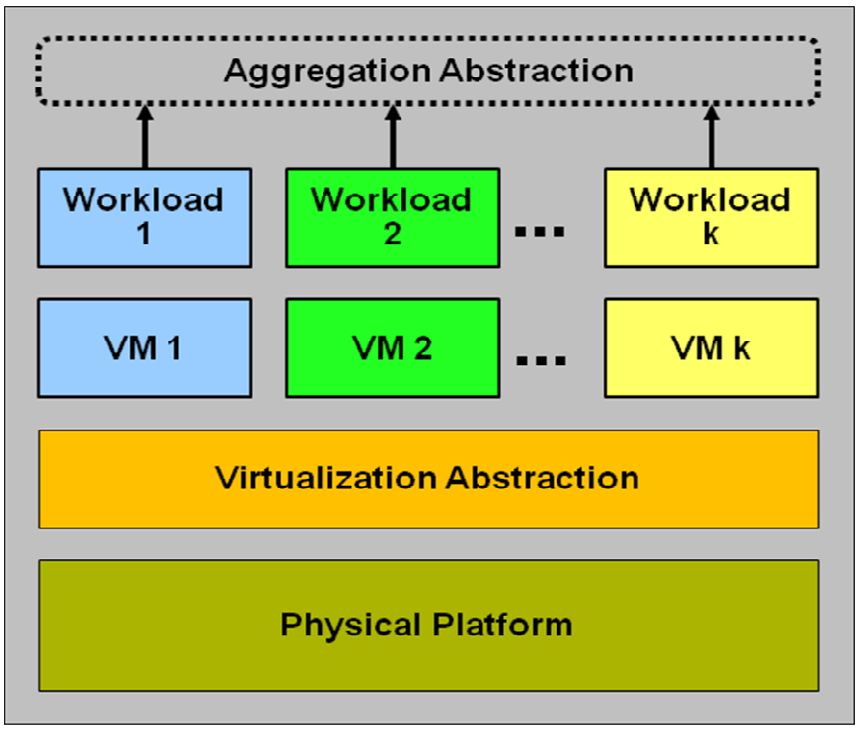

Figure 5: A Tile in a Consolidated Virtualization Workload

3.2 Composition of a Tile

The workload used here was a consolidation virtualization workload as discussed above.. Since this is targeted for a virtual environment, each component runs in its own separate KVM guest, each with its own Red Hat Enterprise Linux (RHEL) operating system. In

addition to the seven benchmark components, there is an eighth VM not running any benchmark, which simulates an idle VM. These eight virtual machines / guests collectively make up a “tile”.

3.2.1 Guest Size

Guest Size
1 vCPU, 1GB RAM
1 vNIC
2-3 vDisks (1GB Swap, 1GB data + / & /usr) - /usr is shared – read only - among guests
All virtualized hardware (no PCI or USB pass-thorough)

All of the virtual machine guests in each “tile” were configured with a single vCPU and a fairly small vMem footprint per guest instance of 1GB (except the “idle” guest environment. This environment was configured with only 512MB vMem). This test environment did not include the use of the KSM functionality although it is available with RHEL 5.4/KVM and would have permitted the use of over-commitment of vMEM relative to the physically available RAM on the host platform. Additionally, all of the virtual machine guests’ I/O adapters (vNICs and vLUNs) were configured as virtio devices.

3.2.2 Guests in Each Tile

Guest #	Workload / Application
1	Web server to measure responsiveness from external client
2	Web server for webclient1
3	The Web client to drive load “inside the box” against Web server1
4	Web load generator backend server
5	AIM7 CPU workload
6	AIM7 disk workload
7	MySQL sysbench workload
8	“Idle” workload

Figure 6 shows the basic topology used in the testing for this report. Note that only the workload in guest #1 has an external load driver.

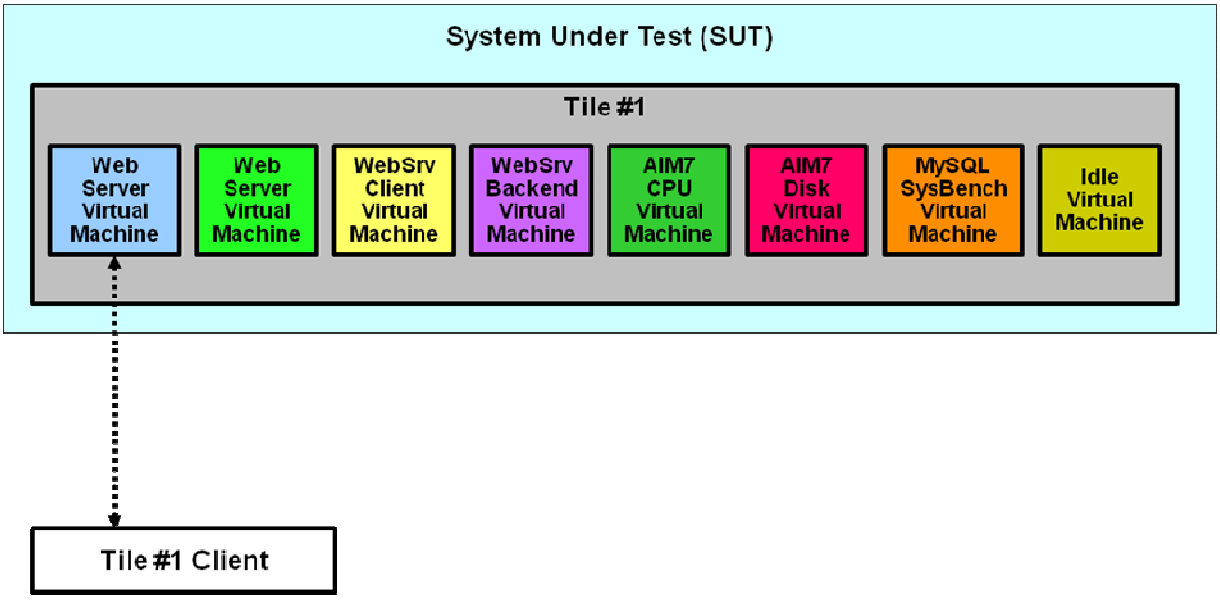

Figure 6: Topology of a Single Tile

3.2.3 Application Mix

To maximize the value of this recommended configuration, the environment consisted of a mix of high-load generating workloads in each of the guests as well as some medium and low generating workloads. For the sake of management simplicity, the virtual machine guests were created and added to the platform in 8-guest increments (a "tile") for which management operations (start/stop/create/destroy) and performance measurements were made. Each of the virtual machine guest "tiles" consisted of the following workload mix:

- 50% of the tile members ran a well-known web server benchmarking suite
- 12.5% of the tile members ran a MySQL database workload generated by SysBench
- 12.5% of the tile members ran a disk I/O workload (AIM 7 FileSystem)
- 12.5% of the tile members ran a CPU & Memory workload (AIM 7 CPU)

- 12.5% of the tile members ran an “idle” workload[1]

All of the guest workloads that were not being measured from the external client systems (over a network) were configured to maximize their imposed load on the system and perpetually run at or near 100% utilization of the in-guest virtual CPU, Memory and I/O resources available. The performance of the environment was measured against two metrics. The primary metric was the service-time per web-server request as well as the number of requests successfully responded to. The “response time” metric was collected by measuring the time elapsed between when an operation/request was submitted to one of the virtual machine web-server instances from an external client system, and reception of the resulting output by the same external client system.

To minimize the system management overhead for the test environment, all of the virtual machine guests were configured with RHEL 5.4. This permitted the test team to share the `/usr` file system between all of the virtual machine guests in a single (read-only) “disk image” file which was stored in the host environment on an ext3 file system. By configuring the environment in this manner, the test team minimized the overhead of generating new virtual machine guest “tiles” and thereby expanding the size of the tested workload. However, as a side-effect, configuring the virtual machine guests with a shared, read-only, `/usr` file system also means that updates and upgrades affect a much larger portion of the guest environment at once which may not be desirable in a production environment.

In a well-sized and configured virtual machine guest, the root, swap, `/usr` and `/var` file systems/devices will receive minimal amounts of read & write I/O operations. As a result, these four (virtual) devices can, in most cases, be successfully configured as one or more disk-image files hosted on one or more of the supported host file systems[2]. By using disk-images for the non-performance critical components of the virtual machine guest configuration, the system administrator will have a very flexible virtual machine guest environment from a migration & management perspective while minimizing the performance implications.

[1] A running instance of RHEL 5.4 with no active applications beyond those shipped and enabled in the default installation of RHEL 5.4.

[2] It is recommended that guests which will not be migrated or live-migrated use the ext3 file system as a backing device for disk-images. Guests which are expected to be live-migrated or migrated between physical hosts should be hosted on a file system which supports being shared and accessed from multiple physical hosts simultaneously such as the GFS/GFS2 file systems.

3.3 Multiple Tiles

To determine the full capabilities of a given system, additional tiles are added. Each tile added results in the adding of an additional eight virtual machines running simultaneously on the system under test.

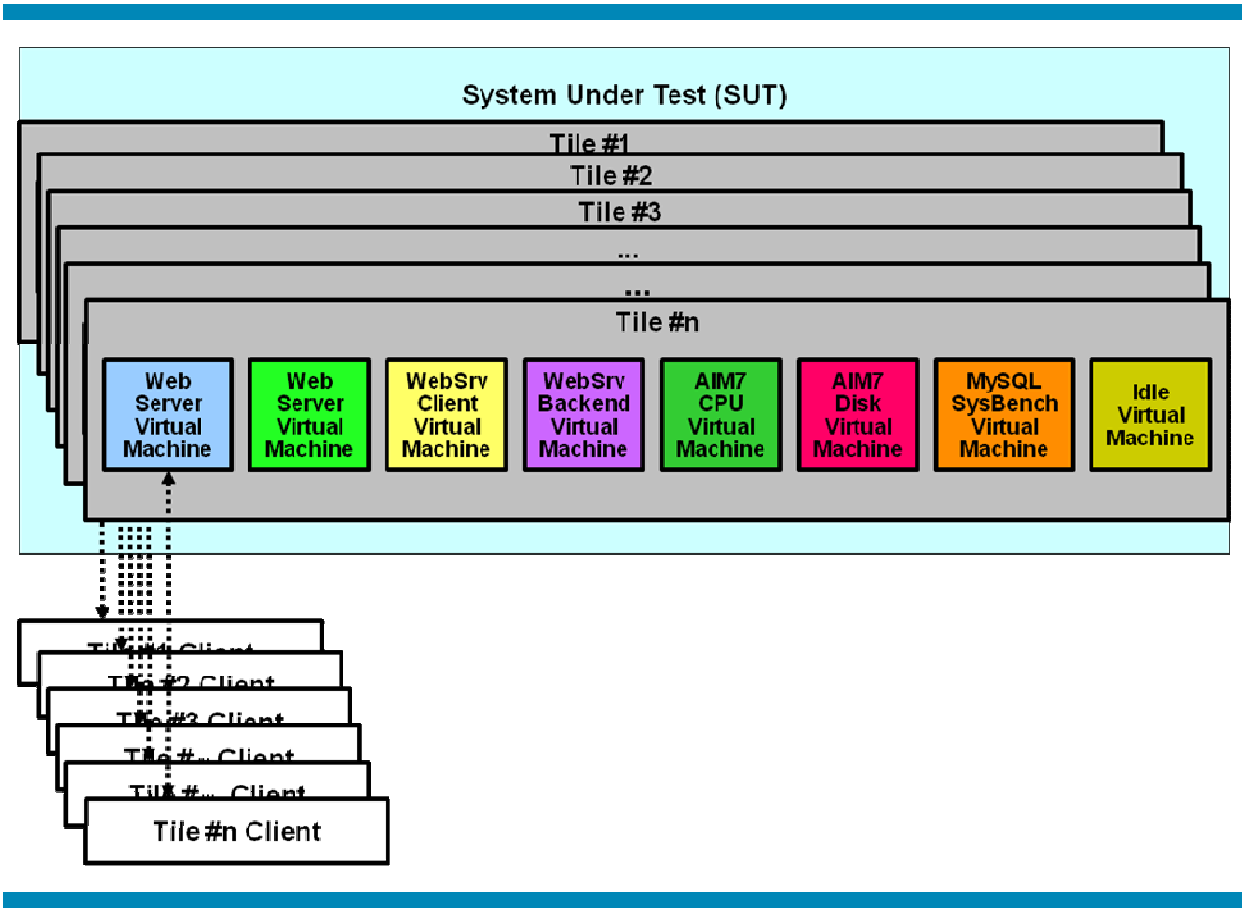

Figure 7: System Under Test (SUT) with Multiple Tiles

As tiles are added to the test, an additional client per tile is connected to the SUT. The added client runs the web workload associated with the additional tile.

The typical goal is to run enough tiles to push the server under test to its maximum capacity by either consuming close to 100 percent of the CPU capacity and / or keeping the response time below some threshold.

4 Test Configuration

The primary goal of the test configuration(s) described in this section and used for the testing described in this paper was to demonstrate the scale-up virtualization capability of RHEL 5.4 and KVM in a Red Hat Virtualization Environment on an HP ProLiant DL785 G6. The configuration(s) may be used as a guideline to assist the end user in building an architecture for their specific needs. However, these configuration(s) are provided as a reference only. Specific end user configurations will vary depending on customer needs. Memory, processor amount and speed, as well as I/O and storage recommendations, should be seen as a reflection of the test configuration and do not represent a recommended size. The actual sizing configuration(s) in an end user environment will be depend on the number of guests, the size of the guests and the types of workloads being hosted. HP and Red Hat strong recommend that you work with your local HP and Red Hat Sales Representatives to help determine the best solution for you.

4.1 Configuration Paradigms

This example configuration, in addition to being optimized for the rack form factor, expose some unique features provided by the selection of HP ProLiant servers and the options it contains. The individual optimization points and trade-offs are explained below.

4.1.1 Rack Optimized

The rack form factor optimized configuration consists of a production environment. The configuration is built around the ability to fit into existing data center rack environments, providing an ideal solution for customers who value future expansion capabilities and modularity over physical rack space. In addition, this configuration allows deployment of high power processor options as well as a significant amount of the storage through either Directly Attached Storage (DAS) or Area Network (SAN) connected storage options.

4.1.2 Workload Considerations

The example configuration contained in this document assumes estimated workloads based on a singular server setup, utilizing Red Hat Enterprise Linux 5.4 Advanced Platform (RHEL 5.4 AP) in a 64-bit server environment. The configuration is assumed to be running mixed workloads as virtualized guests utilizing the included Kernel Virtual Machine (KVM) virtualization technology in RHEL 5.4 AP.

Alternative installations and services may require different configurations, due to changes in server setup, service type and application interaction requirements.

4.2 Example Configuration

4.2.1 Rack Form Factor – HP ProLiant DL785 G6

Figure 8

4.2.2 Production Environment

Function	Quantity	Model	Memory	Processor	GHz [1]	Disk Drives (SAS)[2] OS/App
Application	1	DL785 G6	512 GB	8x Six-Core AMD Opteron™ model 8439	2.8	8x 146 GB 15K SAS Red Hat Enterprise Linux 5.4 Advanced Platform with KVM Virtualization
Network adapter	2 (or more)	NC364T	n/a	n/a	n/a	n/a
Storage adaptor	2 (or more)	FC1242SR	n/a	n/a	n/a	n/a
External storage adaptor	1	HP StorageWorks P400i Smart Array	n/a	n/a	n/a	n/a
Storage [3]	1	HP StorageWorks MSA2300fc	n/a	n/a	n/a	
Storage[4]	1	HP StorageWorks EVA 4400	n/a	n/a	n/a	

[1] Typically the fastest available processor should be used for production servers.
[2] Increasing the number of drive spindles improves performance, 15k Serial Attached SCSI (SAS) reference minimum.
[3] HP StorageWorks MSA2300fc with embedded HP P400i Smart Array, optionally available with BBWC and the HP Smart SAS Expander card.
[4] Optionally replace the HP StorageWorks MSA2300fc with an EVA4400

4.2.3 Storage Volumes

The disks hosting the host operating systems are attached to the system internal HP Smart Array P400 RAID controller in the HP ProLiant DL785 G6 server and should be configured as RAID-1 devices to ensure availability in the event of a disk failure. The container files hosting the guest operating systems were hosted on eight (8) HP StorageWorks 2300 Modular Smart Arrays (MSA2312fc controllers) configured with redundancy for the LUNs--RAID-5—as well as redundant data paths (multipath) from the MSA2300 RAID controllers to the host environment.

The partitions for the guest operating system contents are hosted in file-based containers that are stored on the system-external MSA2300 RAID arrays while the application data file systems are stored on Logical Volume Manager (LVM) logical volumes exported from the host environment to maximize flexibility and performance. These logical volumes are also hosted on the system-external MSA2300 RAID arrays.

Optionally, the guest operating systems and data devices could be hosted on an HP

StorageWorks EVA 4400.

4.3 General Notes

This section details the configuration and performance of a large scale-up HP ProLiant DL785 G6 using Red Hat Enterprise Linux 5.4 Advanced Platform KVM-based virtualization solution hosting up to 256 RHEL guests. After an introductory section describing the hardware and software technologies used for this solution, detailed information is provided about how to configure and optimize the HP ProLiant DL785 G6 as well as Red Hat Enterprise Linux 5.4 Advanced Platform for a virtualization solution of this magnitude.

4.3.1 Overview: HP ProLiant DL785 G6

Powered by the latest Six-Core AMD Opteron™ processors, the HP ProLiant DL785 G6 is the next generation server to the award winning HP ProLiant DL785 G5. HP ProLiant DL785 G6 is an 8-socket x86 server, supporting up to eight (8) Six-Core AMD Opteron™ processors, 512 GB of memory, eleven (11) PCI-e I/O slots or an optional I/O backplane with seven (7) PCI-e and two (2) HTx I/O slots.

With this highly expandable feature set, the HP ProLiant DL785 G6 is an ideal choice for growing enterprise-class database, consolidation and virtualization environments seeking to improve server utilization and reduce server and virtualization sprawl, while continuing to leverage all the familiar and easy to use HP ProLiant management tools and options.

The HP ProLiant DL785 G6 with latest Six-Core AMD Opteron™ processors is built to improve server utilization and reduce resource sprawl. Some of the benefits of this server include:

- 64 DIMM slots for a maximum of 512 GB of memory (with 8 GB DIMMs) and 11 expansion slots so the HP ProLiant DL785 G6 has the memory and I/O expansion capabilities to support very large virtualization and consolidation environments

- The capacity to support eight (8) Six-Core AMD Opteron™ processors the HP ProLiant DL785 G6 delivers outstanding scalability and performance for the most demanding applications

- Management features such as the HP Integrated Lights-Out 2 and HP Insight Manager agents to remotely monitor and control the environment

- Reliability features, such as redundant power supplies and fans, ASR, pre-failure warranty, ensure the HP ProLiant DL785 G6 is well suited to support an enterprise class environment

- 11 PCIe slots with three PCIe x16 slots for high performance I/O for accelerators, visualization or high speed communications

4.3.2 Overview: Red Hat Enterprise Linux 5.4 AP with KVM Virtualization

4.3.2.1 Platform Configuration

In order to ensure performance and availability for the platform:

- All OS/App disks are local hardware RAID 1 or RAID 5

- The best performance is obtained with the highest speed, highest wattage processors. Depending on specific workloads, lower power processors may provide acceptable tradeoff in performance versus power and cooling

- Refer to the Red Hat Enterprise Linux Installation documentation for general recommendations on configuring a RHEL 5.4 AP based servers

- For higher storage performance when running very demanding workloads, substitute the MSA 2300fc RAID array for an EVA4400 You can also install additional Host Bus Adapters (HBAs) in each server to improve overall disk-I/O throughput

- The host environment should be configured for multipath access to the available HP StorageWorks 2300 Modular Storage Arrays

4.3.2.2 Storage Configuration

At the time these experiments were conducted, the HP StorageWorks products used supported the KVM virtualization engine available in RHEL 5.4 as a “Technology Preview”. Although the HP StorageWorks QA team has tested (and certified) the HP Modular Storage Array 2300 (G1) and the HP Enterprise Virtual Array (EVA) 4400 for use in limited environments, it is advised that you please contact the HP StorageWorks Division in order to obtain support information for your specific HP produced and

supported RAID array configurations when wanting to use RHEL 5.4/KVM.

4.3.2.3 Hardware Optimizations

For use as a RHEL 5.4 AP Virtualization with KVM host environment, the HP ProLiant DL785 G6 server should be configured with memory interleaving *disabled*. One of the easiest ways to achieve this is by using the ROM Based Setup Utility (RBSU), available as part of the HP Insight Foundation suite for ProLiant, http://www.hp.com/go/foundation.

4.4 Configuring Red Hat Enterprise Linux 5.4 AP and KVM

The reference architecture consists of a standard installation of Red Hat Enterprise Linux 5.4 Advanced Platform with the KVM option under the Virtualization group and the "Customize now" radio button selected.

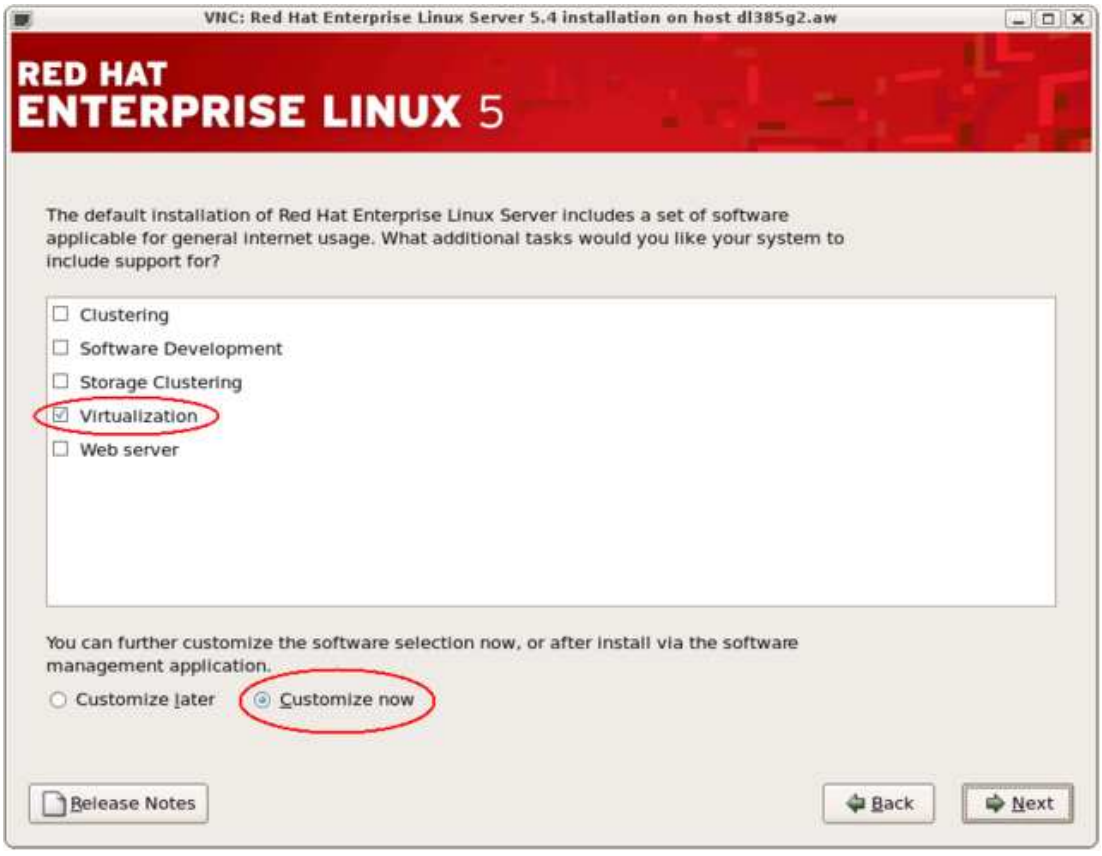

Figure 9

Then select “Next” and select the “KVM” option in the “Virtualization” section of the RHEL 5.4 AP installer customization screen:

Figure 10

NOTE: In this illustration, both the “KVM” and “Virtualization” option is selected. The “Virtualization” option will also install the Xen based hypervisor for RHEL 5.4 AP and enable the `*.el5-xen` kernel as a boot option. Should this behavior be undesirable, disable the “Virtualization” option in the above customization step and select *only* the “KVM” (circled above).

4.4.1 Post Install Customizations

NOTE: Some of the recommendations in this section may lead to disabling certain security measures that ship with, and are enabled by, the RHEL 5.4 AP distribution. These changes should only be considered if there are other security measures in place

to mitigate the risks recommendations pose. Additionally, HP engineering and Red Hat engineering does not recommend disabling security functionality unless doing so has been validated and approved by the appropriate authorities, and the disabling is in compliance with the security practices and policies in effect at the deployment site.

All of the customizations documented in this section were made to the environment in order to improve its performance or scalability.

In order to maximize network performance for the benchmark and testing, the host-based firewall service (iptables) included in RHEL 5.4 AP were disabled using the `system-config-secuitylevel-tui` management tool. As a result, there were no firewall/iptables rules loaded by the host operating system during the boot process. The guest environments were similarly disabled.

Additionally, some of the operating system daemons (services) that are typically enabled by default in the operating system are technically not required in the host environment and, as a result, they were disabled with the chkconfig command.

Disabled operating system daemons:

- crond
- hplip
- cups
- anacron

```
# chkconfig <service> off
```

4.4.2 I/O Scheduling for Performance

The Linux kernel, the core of the operating system, is responsible for controlling disk access by using kernel I/O scheduling. The I/O schedulers provided in Red Hat Enterprise Linux 4 and 5, embedded in version 2.6 and later of the Linux kernel, have advanced the I/O capabilities of Linux significantly. With Red Hat Enterprise Linux 5, applications can now optimize the kernel I/O by selecting one of four different I/O

schedulers to accommodate different I/O usage patterns:

- Completely Fair Queuing—elevator=cfq (default)
- Deadline—elevator=deadline
- NOOP—elevator=noop
- Anticipatory—elevator=as

Red Hat chose CFQ as the default scheduler since it offers the highest performance for the widest range of applications and I/O system designs.

However, considering the platform configuration for this advanced virtualization environment and the use of intelligent RAID controllers for both the externally and internally hosted disk devices, HP and Red Hat recommend using the deadline scheduler for similar environments to the one tested by HP.

The Deadline elevator uses a deadline algorithm to minimize I/O latency for a given I/O request. The scheduler provides near real-time behavior and uses a round robin policy to attempt to be fair among multiple I/O requests and to avoid process starvation. Using five I/O queues, this scheduler will aggressively re-order requests to improve I/O performance.

In RHEL 5.4 AP, changing the I/O scheduler/elevator can be done globally for all devices in the system as part of the boot process by setting the kernel boot parameter “elevator=deadline” or on a per-LUN basis at any point in time by setting the scheduler in the `/sys/block/<device>/queue/scheduler` file.

Example: Setting the kernel boot parameter

```
title Red Hat Enterprise Linux Server (2.6.18-160.el5)
                                        root (hd0,1)
        kernel /vmlinuz-2.6.18-160.el5 ro root=/dev/vg0/rhel5.4 elevator=deadline console=ttyS1,115200
        initrd /initrd-2.6.18-160.el5.img
```

Example: Run-time setting of the elevator per LUN

```
# for pth in /sys/block/{sd,cciss}*/queue/scheduler
do
       echo “deadline” > $pth
done
```

4.4.3 Storage Multipath Configuration

To ensure improved availability for the data accessed and managed by the guests, the host environment should always be configured for multipath failover access. HP recommends adding the following to the `/etc/multipath.conf` to ensure optimal multipath operations when using HP’s StorageWorks Enterprise Virtual Array 4400 or the Modula Storate Array 2300fc.

For the HP Storageworks MSA 2300fc family of RAID controllers:

```
# cat << __EOF__ >> /etc/multipath.conf

device
{
      vendor “HP”
      product “MSA2312fc|MSA2324fc|MSA2312i|MSA2324i”
      getuid_callout “/sbin/scsi_id –g –u –s /block/%n”
      hardware_handler “0”
      path_selector “round-robin 0”
      prio_callout “/sbin/mpath_prio_alua /dev/%n”
      path_grouping_policy group_by_prio
      failback immediate
      rr_weight uniform
      no_path_retry 18
      rr_min_io 100
      path_checker tur
}

__EOF__
```

And for the HP StorageWorks Enterprise Virtual Array family of RAID controllers:

```
# cat << __EOF__ >> /etc/multipath.conf

device
{
       vendor “COMPAQ|HP”
       product “HSV1[01]1|HSV2[01]0|HSV300|HSV4[05]0”
       getuid_callout “/sbin/scsi_id –g –u –s /block/%n”
       prio_callout “/sbin/mpath_prio_alua /dev/%n”
       hardware_handler “0”
       path_selector “round-robin 0”
       path_grouping_policy group_by_prio
       failback immediate
       rr_weight uniform
       no_path_retry 18
       rr_min_io 100
       path_checker tur
}

__EOF__
```

After adding either or both of the above entries in the `/etc/multipath.conf` file, depending on the chosen configuration, enabling multipathing is a very simple task and only requires starting the Linux multipath service and ensuring that it starts at system boot.

```
# chkconfig multipathd on
# chkconfig --list multipathd
multipathd         0:off  1:off  2:on   3:on   4:on   5:on   6:off
# service multipathd start
```

4.4.4 Network Multipath Configuration

To ensure improved availability for the services and improve total network throughput across the installed Network Interface Cards (NIC), the RHEL 5.4 AP network bonding functionality should be configured in the host environment.

For the test configuration, the eight physical ports on the two HP NC364T PCI-E Quad Port Gigabit Server Adapters and the two embedded NC371i Multifunction Gigabit Network Adapters were configured as five bonded interfaces. Four of the bonded interfaces consist of one port from each of the two NC364T NICs and the fifth consists of the two embedded NC371i NICs. This was done in an effort to maximize throughput and ensure that the guests were not starved as the network links got saturated.

To ensure support for bonded interfaces with the expected load-balancing and failover policy set, the following changes must be made, or added, to the `/etc/modprobe.conf` file:

```
# Comment: To enable bonding for multiple bonding devices, add
# one “alias bond<#> bonding” (# is a number) line for each of
# the bonding interfaces you want to configure
#

alias bond0 bonding
alias bond1 bonding
alias bond2 bonding
alias bond3 bonding
alias bond4 bonding

#
# Add the appropriate options for the bonding driver to select
# the bonding algorithm and timeout values for the bonds.
# See http://www.linuxhorizon.ro/bonding.html for a description
# of the various mode= settings and their meaning
# Additionally, the Red Hat documentation (RHEL 5 Deployment Guide)
# includes an excellent write-up:
#
# The following “options” directive applies to all bonding interfaces
#

options bonding mode=4 miimon=100 xmit_hash_policy=layer3+4

#
# Whereas this one only applies to the bondX device it references.
# This way you can do a per-interface policy if you so choose.
# options bond0 mode=4 miimon=100 xmit_hash_policy=layer3+4
```

Additionally, the network interface configuration files must be modified to enslave the physical NICs (`ethX` devices) to the appropriate logical `bondX` interface. The following is only an example of a configuring bond0 to enslave the physical NICs eth0 and eth1:

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth0

# Broadcom Corporation NetXtreme II BCM5706 Gigabit Ethernet
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
MASTER=bond0
SLAVE=yes
USERCTL=no
HWADDR=00:21:5A:DE:04:1A

# cat /etc/sysconfig/network-scripts/ifcfg-eth1
# Broadcom Corporation NetXtreme II BCM5706 Gigabit Ethernet
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
MASTER=bond0
SLAVE=yes
USERCTL=no
HWADDR=00:21:5A:DE:04:1C

# cat /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
BOOTPROTO=dhcp
ONBOOT=yes
USERCTL=no
```

4.4.5 Bridged Network Interfaces

By default, the KVM guest is given access to either a virtual network device (NAT) or a “shared physical device” (bridge). To ensure the best possible performance by minimizing the layers of the software stack that the guest traffic had to traverse, all of the guests in the environment were configured to use one of the five bonded interfaces. To configure a “shared physical device”, you must assign one of the bonded interfaces (`bond[0-4]`) to a network bridge. The RHEL 5.4 AP network startup scripts are capable of doing this automatically during the boot process if the following configuration changes are made to the RHEL 5.4 AP host environment. For this to successfully complete, the steps outlined in the “Network multipath configuration” section of this document must be successfully completed.

First, change the `/etc/sysconfig/network-scripts/ifcfg-bond[0-4]` files so they no longer are assigned an IP address from the DHCP client but instead is assigned to the bridge device:

```
# cat /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
BRIDGE=br0
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
```

Add the `/etc/sysconfig/network-scripts/ifcfg-br[0-4]` configuration file and populate it based on the following template:

```
# cat /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
# For a DHCP configuration, delete the “BOOTPROTO=none” line
# and uncomment the ‘dhcp’ line:
#BOOTPROTO=dhcp
BOOTPROTO=none
USERCTL=no
ONBOOT=yes

# To set a static IP address, uncomment the following lines
#IPADDR=111.222.333.444
#NETMASK=111.222.333.444
#GATEWAY=111.222.333.444
#SEARCH=domain.tld
#DNS1=111.222.333.444
#DNS2=111.222.333.444
#DNS3=111.222.333.444
```

5 Performance Testing Results

5.1 Performance Summary

The goal of this study was to demonstrate the scale-up capabilities of the Red Hat Enterprise Virtualization environment – Red Hat Enterprise Linux (RHEL) 5.4 with the integrated Kernel-based Virtual Machine (KVM) hypervisor.

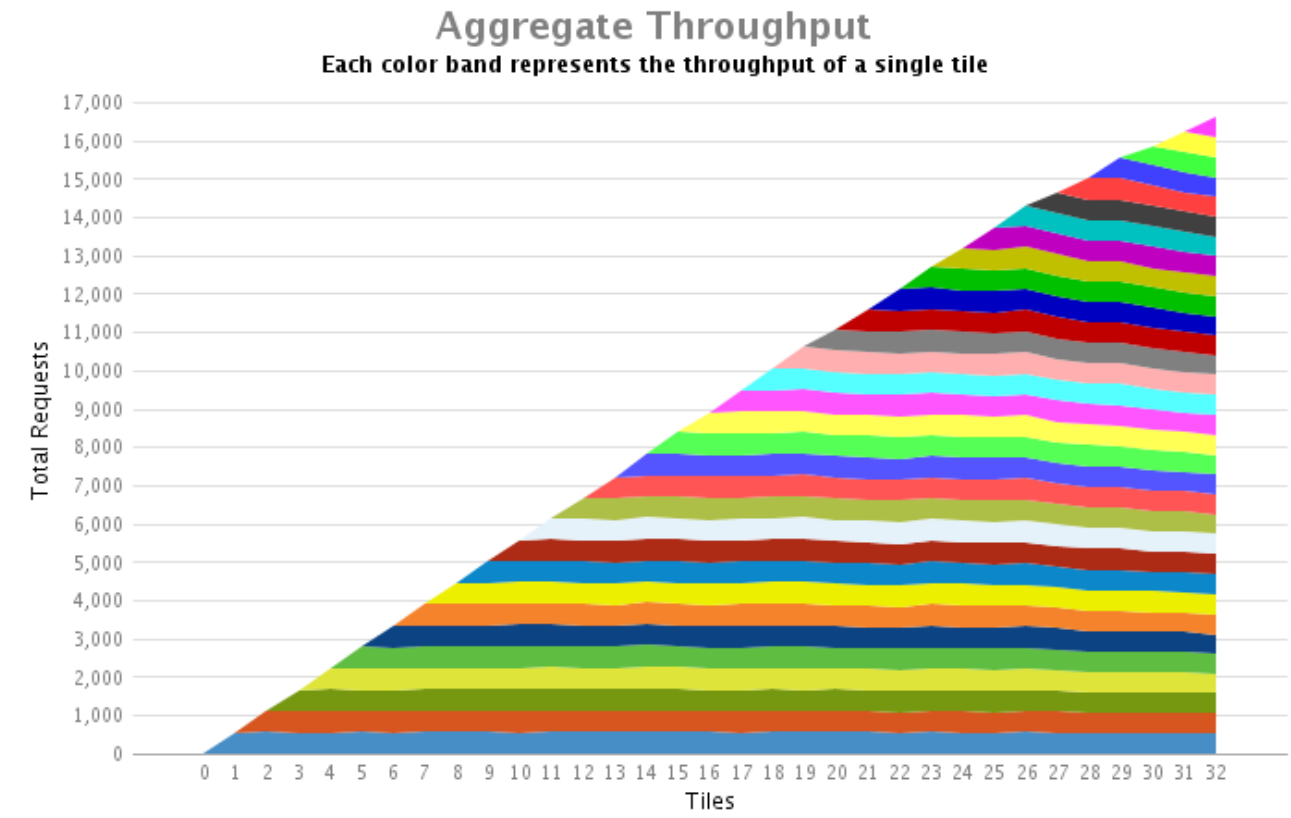

Figure 11

The configured environment successfully and **linearly scaled up to 256 virtual machine guests** (32 ”tiles” each comprising 8 virtual machines) on a single ProLiant DL 785 G6 server. The “Aggregate Throughput” graph below illustrates the linearity of the guest

scalability while the “Average Response Time” graph below illustrates the ability of the platform to consistently maintain an average response time of less than one second all the way to 256 simultaneously running guest instances. In the throughput graph, each color represents the throughput contribution of different tiles. Note that the KVM hypervisor enforces fair scheduling, i.e., for any configuration of “n” tiles, each tile generates ~1/nth the total throughput. These graphs clearly demonstrates that the host hypervisor (KVM) scaled up to the increased load by running all of the 256 guest instances while **fairly scheduling each of the guest workloads** and **ensuring reasonable response times** for externally connected clients utilizing the resources of the platform.

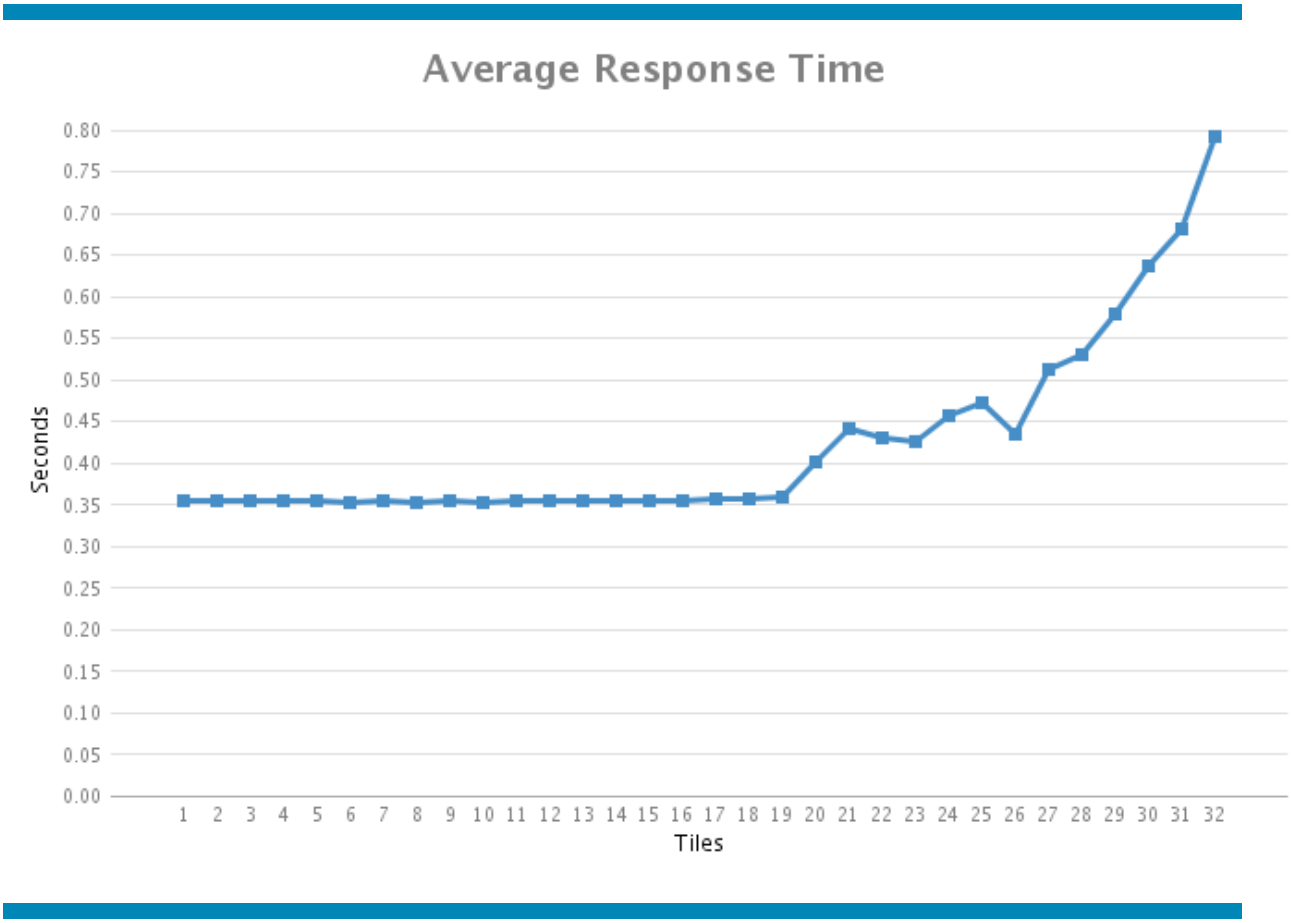

Figure 12

5.2 Tuning the Host Environment for Performance

This sub-section contains the information necessary to tune the operating system environment on the host platform in order to optimize the network throughput (number of

requests serviced by the virtual machine guests) and maintain an acceptable response time per request. Graphics are used to demonstrate the changes in system behavior as the `sysctl` tunables were applied to the environment.

5.2.1 Netfilter and Guest Traffic over Bridge Interfaces

As mentioned previously, the host environment is configured with several pairs of bonded devices (see: Network Multipath Configuration) which were configured as network bridges for the guest vNICs. This, combined with the preservation of the default firewall software as configured "out-of-the-box" in RHEL 5.4 implies that all guest traffic is processed by the netfilter functionality in the Linux TCP/IP software stack of the host environment. This has rather significant impact on the performance of the guest network throughput as illustrated in the graph below.

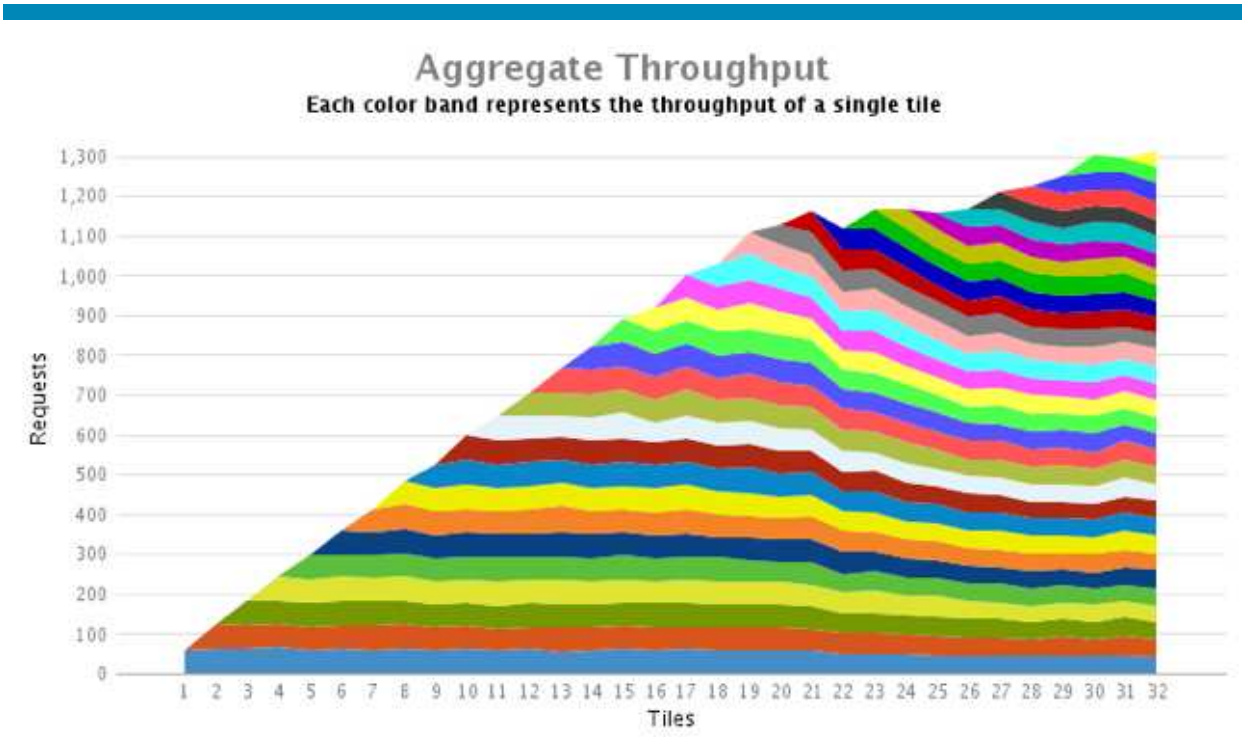

Figure 13

The above graph depicts a workload with only 1/10th the number of request of the full workload, yet it should be obvious that both the throughput and the saturation point for requests made to the guest instances in the depicted graph occurs much sooner than

what is illustrated in the “Performance Summary” section.

In order to mitigate this behavior, it will be necessary to by-pass the netfilter processing of traffic on the bridged interfaces by dynamically tuning the environment. The following entries must be added to the `/etc/sysctl.conf` file and the settings must be reloaded as illustrated below.

```
# cat << _EOF_ >> /etc/sysctl.conf

# Disable netfilter on the bridge
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0

_EOF_

# sysctl -p
```

An administrator making the above illustrated optimizations to the environment need to realize that network traffic traveling across the bridged network devices will no longer be inspected by the netfilter capability and should ensure that network traffic to the virtual machine guests is afforded similar filtering capabilities via a dedicated hardware based firewall or that the virtual machine guests have not had their iptables/firewall capabilities disabled as well.

With the proper application of the `syctl` settings, the same workload displays a much more positive performance behavior in terms of scalability and throughput/response times:

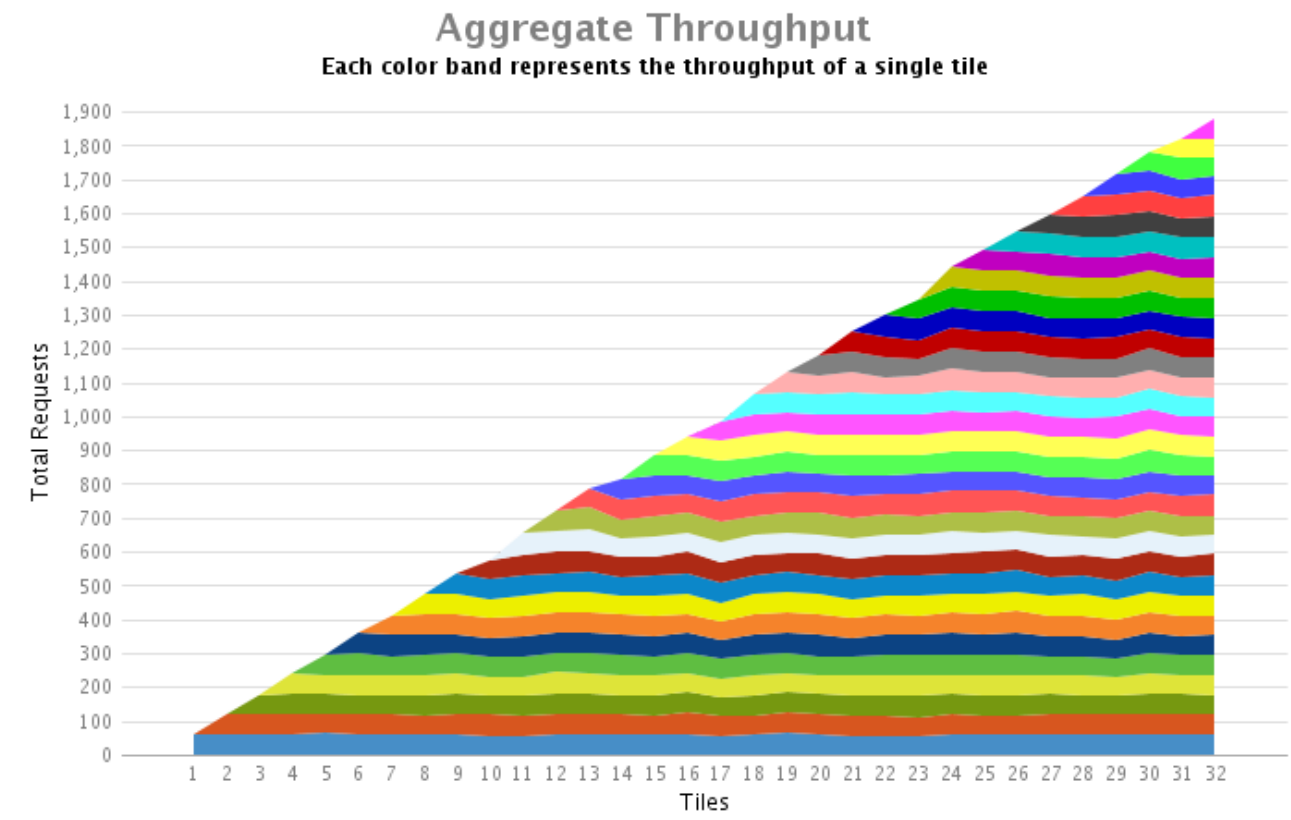
Aggregate Throughput
Each color band represents the throughput of a single tile
Total Requests
1,900
1,800
1,700
1,600
1,500
1,400
1,300
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Tiles

Figure 14

5.2.2 Supporting Large Amounts of Network Traffic to the Guests

Although the removal of netfilter processing for the bridged network interfaces being used by the virtual machine guests significantly improved the scalability of the workload, the environment still has ample free hardware resources. Since the tested environment consisted of large numbers of small guests as a consolidation workload, the availability of underutilized host hardware resources means the tested workloads ought to be increased and the host environment be tuned further. As a result, the test number of requests was increased 10-fold from each of the test clients. The resulting platform behavior is illustrated below:

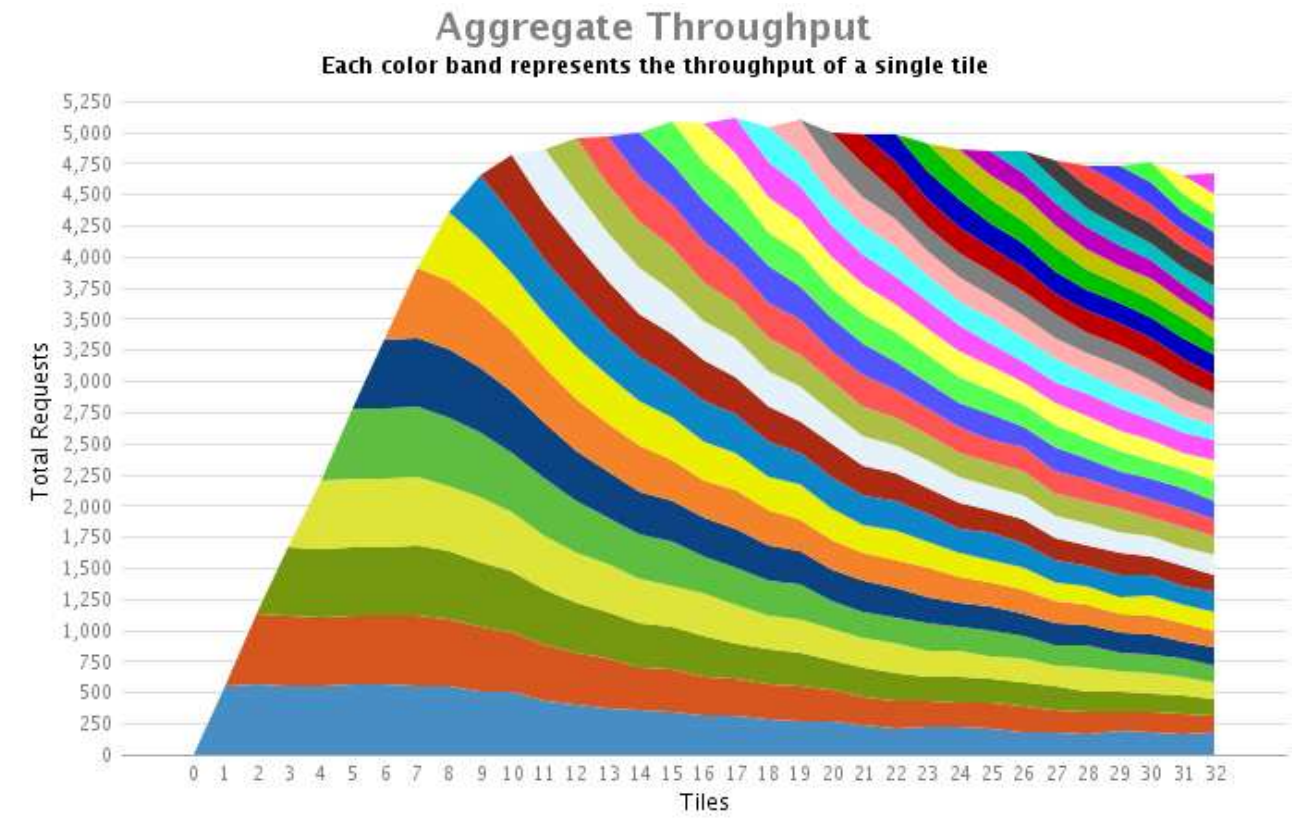

Figure 15

As can be seen from the graph, the per-tile request load has increased significantly, however the environment is no longer scaling linearly and thus represents an opportunity

for improvement. Note that while the system is clearly overloaded, the KVM hypervisor continues to enforce fair scheduling, i.e., for any configuration of “n” tiles, each tile generates ~1/nth the total throughput. The test environment was subsequently tuned using the following settings in order to increase the possible throughput and achieve linear scalability for the number of guests being hosted while preserving an acceptable per-request response time (measured from a 3rd party client system):

```
# cat << _EOF_ >> /etc/sysctl.conf

net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_max_tw_buckets = 4000000
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.core.optmem_max = 67108864
net.ipv4.tcp_rmem = 4096 16777216 67108864
net.ipv4.tcp_wmem = 4096 16777216 67108864
net.ipv4.tcp_mem = 67108864 67108864 67108864
net.ipv4.tcp_dsack = 0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_window_scaling = 0
net.core.somaxconn = 640000
net.core.netdev_max_backlog = 250000
net.ipv4.tcp_max_syn_backlog = 20000
net.ipv4.ip_local_port_range = 4096 65535

_EOF_
# sysctl -p
```

As a result of tuning the network stack as illustrated above, the environment is capable of processing around three times (3x) the number of requests across 240 virtual machine guests, all while maintaining a sub-second average response time.

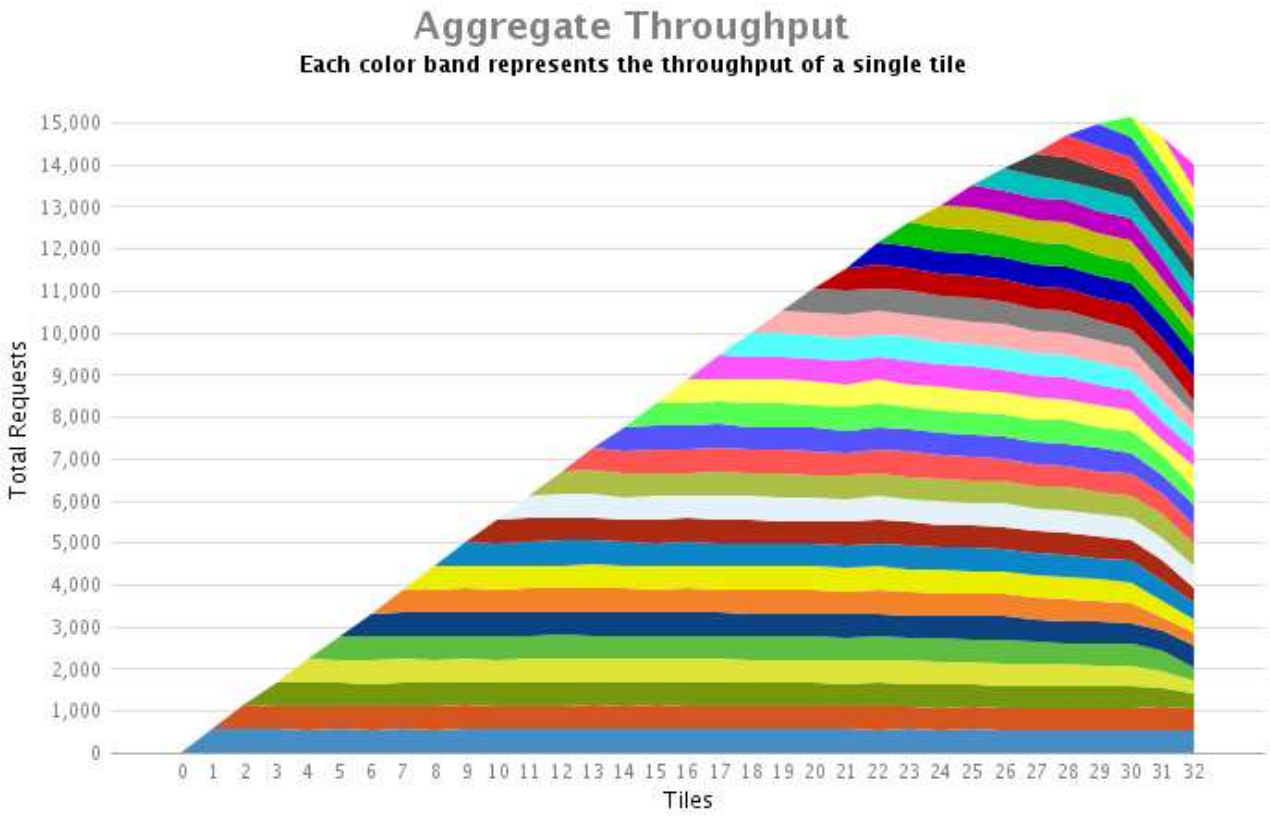
Aggregate Throughput
Each color band represents the throughput of a single tile
Total Requests
15,000
14,000
13,000
12,000
11,000
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Tiles

Figure 16

5.3 Virtual Machine Guest Placement and the Platform Architecture

NUMA is a computer memory design used in some multiprocessor systems. One of the attributes of this design is that the time it takes to access a page of memory is dependent the memory location relative to the processor. On a NUMA based system, a processor can access its own local memory faster than non-local memory, i.e. memory local to another processor or memory shared between processors. As a consequence, programs that are accessing local memory tends to be out-performed by the same program on a system only accessing non-local (remote) memory.

Although the HP ProLiant DL 785 G6 implements a low latency NUMA architecture, it was decided to validate the relative performance of the environment by testing two distinct configuration models. One relied on the operating system to optimize the workload (automated locality management – *unpinned*) and the other utilized a manual distribution (manual locality management – *pinned*) of the virtual machine guests where the placement policy was based on the administrators intimate knowledge of the workload types that were being run as well as a good understanding of the hardware configuration upon which the workloads were running. The latter is obviously a more labor intensive policy to implement and as a result, any performance difference between the two policies should be weighed against this.

For the test environment, a scripted approach was used to manage the guest workload placement.

NOTE: Using the following script, in a production environment, should only be considered if the system administrator has a firm understanding of the workloads being virtualized and their behavior in the virtualized environment, as well as fully understands both the amount, as well as the layout, of memory and I/O resources relative to the various platform CPU sockets and the resource requirements of the virtual machine guests. Failure to do so may result in a significant degradation of system performance when compared to an "automated locality management" approach.

The script "shim" created for the test environment is detailed below. The testers created a placement policy by creating an `/etc/libvirt/quest-pinning.conf` file. This file specifies which "node" (i.e. the CPU socket) that a specific guest or group of guests should be started on and isolated to. By saving the "shim" script as `/usr/libexec/qemu-kvm` and re-naming the qemu-kvm binary to `/usr/libexec/qemu-kvm.real`, the standard Linux KVM guest management tools

will be able to leverage the placement policy.

```
#!/bin/bash
#
# /usr/libexec/qemu-kvm shim for pinning KVM guests with numactl
#
# To use this, create a configuration file
# /etc/libvirt/guest-pinning.conf that contains guest to
# node associations.  Guest names can be globs.
#
# Valid node specifiers are as described in numactl(8).
#
# Example configuration:
#
#   postgres1 0-2
#   postgres2 3-5
#   jack      6
#   jill      7
#   node0-*   0
#   node1-*   1
#
#

qemu=/usr/libexec/qemu-kvm.real
conf=/etc/libvirt/guest-pinning.conf
numactl=
guest=

if [[ -s $conf ]]; then
    # find the guest name in the positional parameters
    for ((i = 1; i <= $#; i++)); do
        if [[ ${!i} == -name ]]; then
            let i++
            guest=${!i}
            break
        fi
    done

    # match against /etc/guest-pinning
    if [[ -n $guest ]]; then
        while read line; do
            line=${line%%#*} # trim comments
            read glob node extra <<<"$line"
            if [[ $guest == $glob ]]; then
                numactl="numactl --cpunodebind=$node --membind=$node"
                break
            fi
        done < $conf
    fi
fi

exec $numactl $qemu "$@"
```

5.3.1 Results from Tests of Manual Locality Management Policy

As can be observed in the previous section, the automated locality management policy delivers linear scalability up to about 30 tiles from a perspective of the number of requests serviced. By utilizing the above mentioned “shim” and thus using the manual locality management policy, the number of requests processed by the platform increased by approximately 8% as illustrated in the graphic below.

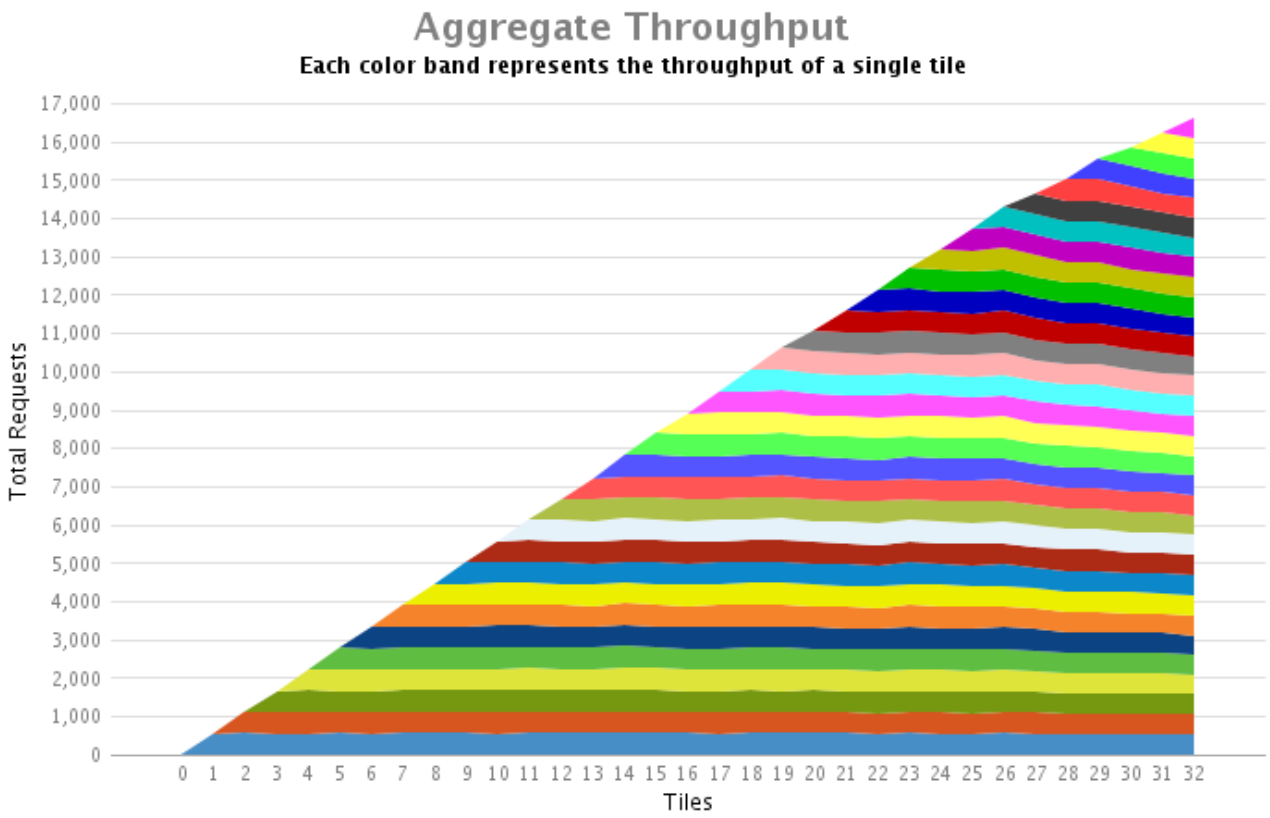

Figure 17

Additionally, in a side-by side, comparison of the average response times for the two test runs, the average response time per request demonstrates a more well-behaved system when increasing the guest load from 240 guests to 256 and it is obvious that the platform load can be increased further before the response time per request as well as any increases in the numbers of requests serviced will exhibit diminishing returns.

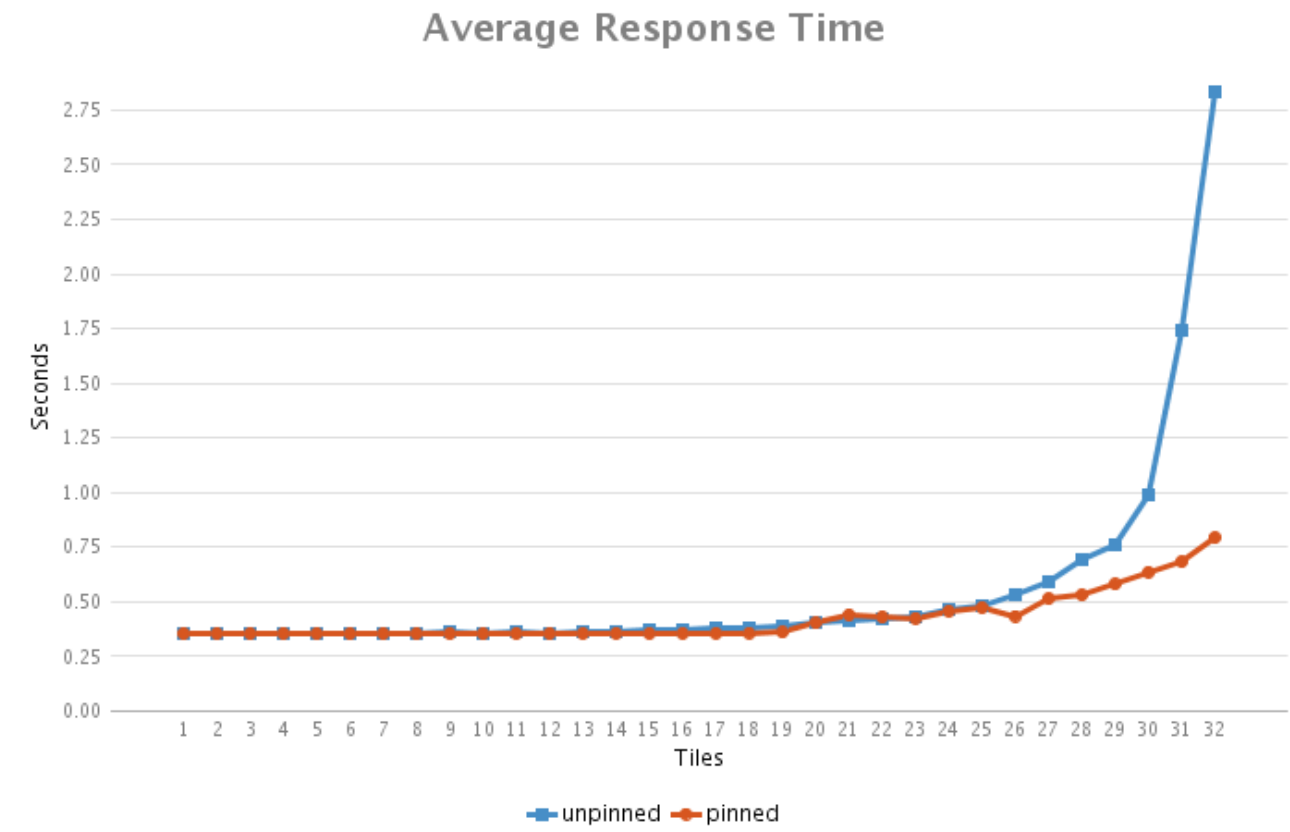

Figure 18

5.4 In-distro Virtualization Tool Optimizations

In order to successfully use the libvirtd based management tools that come bundled with RHEL 5.4 AP when testing with more than 20 tiles, the team had to increase several values in the libvirtd configuration file:

```
#################################################################
#
# Processing controls
#

# The maximum number of concurrent client connections to allow
# over all sockets combined.
#max_clients = 20
max_clients = 300 <-- Changed

# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20
max_workers = 300 <-- Changed

# Total global limit on concurrent RPC calls. Should be
# at least as large as max_workers. Beyond this, RPC requests
# will be read into memory and queued. This directly impact
# memory usage, currently each request requires 256 KB of
# memory. So by default upto 5 MB of memory is used
#
# XXX this isn't actually enforced yet, only the per-client
# limit is used so far
#max_requests = 20
max_requests = 300 <-- Changed

# Limit on concurrent requests from a single client
# connection. To avoid one client monopolizing the server
# this should be a small fraction of the global max_requests
# and max_workers parameter
#max_client_requests = 5
max_client_requests = 300 <-- Changed
```

/etc/libvirt/libvirtd.conf