Red Hat Reference Architecture Series

Scaling SAP in a Red Hat® Enterprise Virtualization Environment

SAP LinuxLab Certification Suite (SLCS) 2.3
SAP ECC 6
MaxDB Database 7.6.06_09
Red Hat® Enterprise Linux 5.4 Guest
Red Hat® Enterprise Linux 5.4 (with Integrated KVM Hypervisor)
Intel Xeon X5570 (Nehalem)

Version 1.0
August 2009

redhat

Scaling SAP in a Red Hat® Enterprise Virtualization Environment

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 following terms used in this publication are trademarks of other companies as follows:

- Linux is a registered trademark of Linus Torvalds
- Red Hat, Red Hat Enterprise Linux and the Red Hat "Shadowman" logo are registered trademarks of Red Hat, Inc. in the United States and other countries
- SAP® software and all other SAP products and services mentioned herein are registered trademarks of SAP AG in Germany and other countries

All other trademarks referenced herein are the property of their respective owners.

© 2009 by Red Hat, Inc. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

The information contained herein is subject to change without notice. Red Hat, Inc., SAP AG, shall not be liable for technical or editorial errors or omissions contained herein.

Distribution of modified versions of this document is prohibited without the explicit permission of Red Hat Inc.

Distribution of this work or derivative of this work in any standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from Red Hat Inc.

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

1 Executive Summary....4

2 Red Hat Enterprise Virtualization (RHEV) - Overview....5

2.1 Red Hat Enterprise Virtualization (RHEV) - Portfolio....5
2.2 Kernel-based Virtualization Machine (KVM)....8
2.2.1 Traditional Hypervisor Model....9
2.2.2 Linux as a Hypervisor....9
2.2.3 A Minimal System....9
2.2.4 KVM Summary....9

3 Test Configuration....11

3.1 Hardware....11
3.2 Software....11
3.3 SAN....11

4 Test Methodology....13

4.1 Workload....13
4.1.1 Database Load Test....13
4.1.1.1 Create Database (Write-Only)....14
4.1.1.2 Load Database (Read/Write)....14
4.1.1.3 Update Database Statistics (Read-Only)....15
4.1.2 Memory Load Test....15
4.2 Performance Test Plan....16
4.3 Tuning & Optimizations....17

5 Test Results....18

5.1 Scaling Multiple 2-vCPU Guests....19
5.2 Scaling Multiple 4-vCPU Guests....22
5.3 Scaling-Up the Memory and vCPUs in a Single Guest....25
5.4 Consolidated Virtualization Efficiency....27

6 Conclusions....28

7 References....28

Appendix A: Sample .xml File....29

1 Executive Summary

This paper describes the performance and scaling of SAP running in Red Hat Enterprise Linux 5.4 guests on a Red Hat Enterprise Linux 5.4 host with the KVM hypervisor. The host was deployed on a dual socket, quad core, 2.93 GHz, Intel Xeon X5570 Nehalem system with 48GB of RAM and support for hyper-threading technology; comprising a total of 8 cores and 16 hyper-threads.

The workload used was the SAP LinuxLab Certification Suite (SLCS), a tool created by SAP for software and hardware vendors to ensure the smooth operation of SAP software on their platforms.

Scaling Up A Virtual Machine
First, the performance of the SLCS workload was measured by loading a single VM on the server, and assigning it two, four and eight vCPUs. The performance scales quite linearly as the VM expands from 1 hyper-thread to a complete 4 core/8 hyper-thread server.

Scaling Out Virtual Machines
A second series of tests involved scaling out multiple independent VMs each comprised of two, four or eight vCPUs to a total of 16 vCPUs on an 8 core/16 hyper-thread Nehalem server. As an example, Red Hat tested the performance one to eight concurrent 2-vCPU VMs running the SLCS workload. Four VMs produced over 386% of the performance of the single VM, with each of the four guests retaining an average 97% of the single guest rate. Even when hyperthreading is engaged (when running more than four 2-vCPU Vms), performance still gained an additional 67% with eight VMs.

The data presented in this paper establishes that Red Hat Enterprise Linux 5.4 virtual machines using the KVM hypervisor on Intel Nehalem provide an effective production-ready platform for hosting multiple virtualized SLCS workloads. The combination of low virtualization overhead and the ability to both scale-up and scale-out contribute to the effectiveness of KVM for SAP. The number of actual users and throughput supported in any specific customer situation will, of course, depend on the specifics of the customer application used and the intensity of user activity. However, the results demonstrate that in a heavily virtualized environment, good throughput was retained even as the number and size of guests/virtual-machines was increased until the physical server was fully subscribed.

2 Red Hat Enterprise Virtualization (RHEV) - Overview

2.1 Red Hat Enterprise Virtualization (RHEV) - 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 datacenter. 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, that is designed to overcome these challenges, enable pervasive datacenter 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 millions of organizations 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.

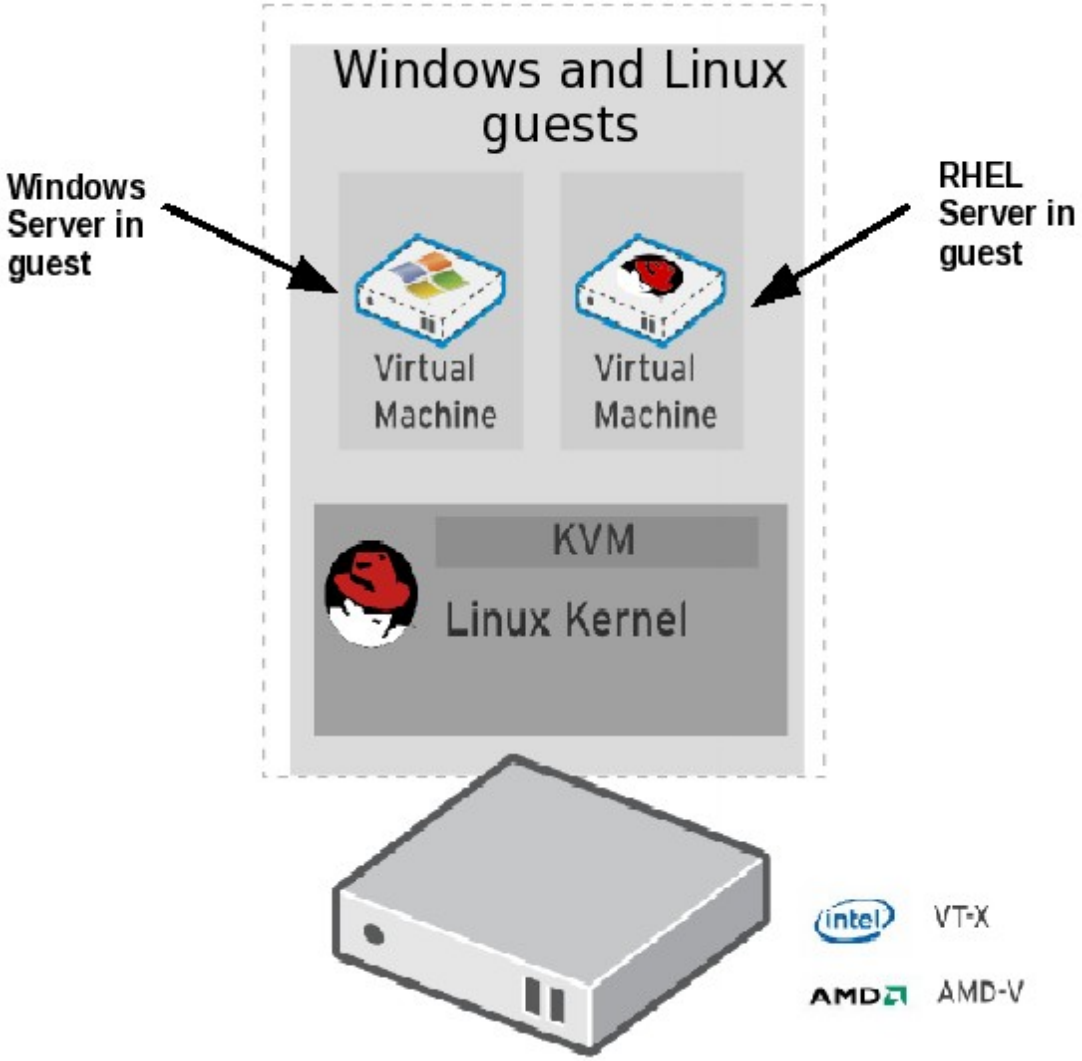

Figure 1: ***Red Hat Enterprise Virtualization Hypervisor***

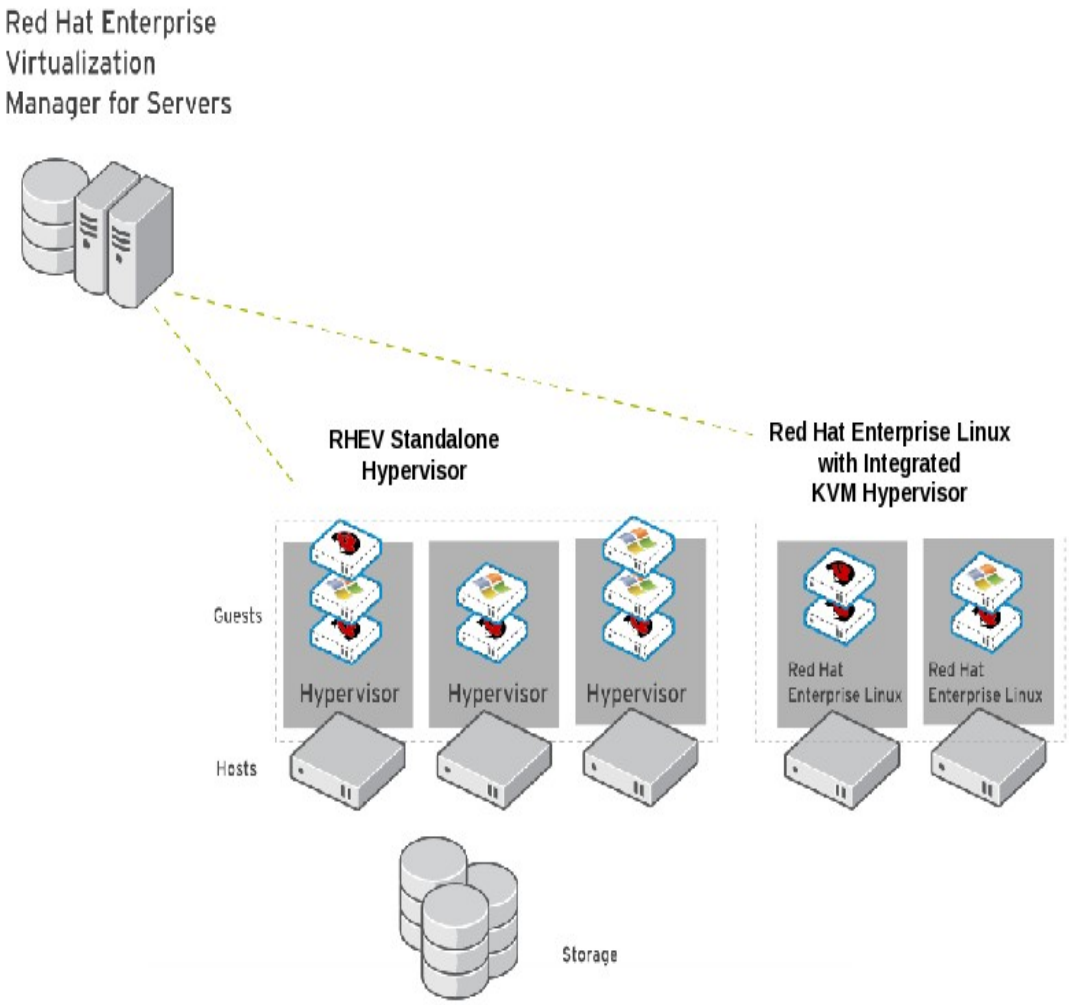

Figure 2: Red Hat Enterprise Virtualization Manager for Servers

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 which translates guest IO 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 IO and memory requests.

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

1. First generation hardware assisted virtualization: Removes the need 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 IO devices to be attached directly to virtual machines using Intel VT-d (Virtualization Technology for directed IO) and AMD IOMMU. And SR-IOV (Single Root IO Virtualization) which allows special PCI devices to be split into multiple virtual devices. This provides significant improvement in guest IO performance.

The great interest in virtualization has led to the creation of several different hypervisors. However, many of these predate hardware-assisted virtualization, and are therefore somewhat 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 a Linux kernel 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 kernel.

2.2.1 Traditional Hypervisor Model

The traditional hypervisor model consists of a software layer which multiplexes the hardware among several guest operating systems. The hypervisor performs basic scheduling and memory management, and typically delegates management and IO 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 A Minimal System

One of the advantages of the traditional hypervisor model is that it is a minimal system, consisting of only a few hundred thousands lines of code. However, this view does not take into account the privileged guest. This guest has access to all system memory, either through hypercalls or by programming the DMA hardware. A failure of the privileged guest is not recoverable as the hypervisor is not able to restart it if it fails.

A KVM based system's privilege footprint is truly minimal: only the host kernel plus a few thousand lines of the kernel mode driver have unlimited hardware access.

2.2.4 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.

Red Hat Enterprise Linux 5.4 incorporates KVM-based virtualization in addition to the existing Xen-based virtualization. Xen-based virtualization, of course, remains fully supported for the life of the Red Hat Enterprise Linux 5 family.

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 do not need to be rebuilt 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 virtual machines. So the portfolio of thousands of certified applications for Red Hat Enterprise Linux applies to both environments.

3 Test Configuration

3.1 Hardware

Dual Socket, Quad Core, Hyper Threading (Total of 16 processing threads) Intel(R) Xeon(R) CPU X5570 @ 2.93GHz
12 x 4 GB 1066MHz DDR3 RDIMMs - 48 GB total
1 x 146 GB SAS 15K dual port disk drive

Table 1: Hardware

3.2 Software

Red Hat® Enterprise Linux 5.4 beta	2.6.18-155.el5 kernel
KVM	kvm-83-80.el5
SAP	ECC 6
SLCS	2.3
MaxDB	7.6.06_09

Table 2: Software

3.3 SAN

The hypervisor host utilized three MSA2324fc and one MSA2212fc fibre channel storage arrays for this testing. The MSA2212fc was used for all of the guest OS disks while one LUN from each of the other three arrays were LVM striped at the guest and used for the SAP database traffic for each virtual machine. Additional details regarding the Storage Area Network (SAN) hardware are in **Table 3**.

(1) HP StorageWorks MSA2212fc Fibre Channel Storage Array	Storage Controller: Code Version: J200P19 Loader Code Version: 15.010 Memory Controller: Code Version: F300R21 Management Controller Code Version: W420R35 Loader Code Version: 12.013 Expander Controller: Code Version: 2042 CPLD Code Version: 27 Hardware Version: LCA 55
(3) HP StorageWorks MSA2324fc Fibre Channel Storage Array	Storage Controller: Code Version: M100R18 Loader Code Version: 19.006 Memory Controller: Code Version: F300R22 Management Controller Code Version: W440R20 Loader Code Version: 12.015 Expander Controller: Code Version: 1036 CPLD Code Version: 8 Hardware Version: 56
(1) HP StorageWorks 4/16 SAN Switch	Firmware: v5.3.0
(1) HP StorageWorks 8/40 SAN Switch	Firmware: v6.1.0a

Table 3: Storage Area Network

Device-mapper multipathing was used at the host to manage the multiple paths to each LUN.

4 Test Methodology

4.1 Workload

As Linux becomes an increasingly more important operating system running SAP applications, SAP developed a process to ensure the quality of Linux core components such as the kernel, glibc, KVM, etc.

This section describes the two primary test loads which the SAP LinuxLab and its partners agreed would be mandatory for certifying hardware or virtualization on Linux. The tests are the database load and memory load. The database load focuses mainly on IO performance and the memory load test focuses on memory throughput.

The following two tests which are part of the SAP LinuxLab Certification Suite were used to measure performance and scalability:

- **Database Load Test** - The IO scenario is realized by importing an SAP ERP2005SR1 system into a database. This phase is divided into three minor phases which cover the following workloads:
    1. write only (Database Create)
    2. read/write (Database Load)
    3. read only (Database Update/Initialization)

- **Memory Load Test** - The memory scenario is realized by allocating and working on huge memory areas. In particular, this mostly done by excessive use of mmap() and mprotect() functions.

4.1.1 Database Load Test

[Note: The SLCS database load scenario described below represents a database migration necessitated by increasing load over time. One of the goals of Red Hat Enterprise Virtualization, in response to increasing load, is to offer applications & databases flexibility and longevity by enabling them to be hosted on larger virtual machines, on larger servers using the latest hardware and software.]

The database load is a task every customer has to perform at least once at the installation of the SAP system. After the initial installation, the system may run for years without any change to the system infrastructure. But after several years of running the database on one server, as new users are added to the system, the hardware could reach its capacity. In order to avoid exceeding capacity, customers must change the hardware of the database server either by migrating to a faster system or by upgrading the hardware in place. Upgrading the hardware may be difficult after several years, because the CPU sockets or the layout of the memory modules may have changed over time, thus a migration is the only way to speed up the database server.

The migration of an existing SAP system is done by exporting the current database content to a database and operating system of independent format known as ‘database export’. After

creating a new database on a new system, this database export is loaded into the new database. During this time the SAP system must be offline because the database is unavailable because of exporting and importing the content. Such export and import scenarios are normally performed over a weekend where the SAP system may not be as needed. The important and essential part of the migration is that it must be completed as quickly as possible since the system's users will want to connect to the SAP system. Furthermore, the SAP consultants doing the migration can be quite expensive. Every hour saved due to a fast import can be worth a lot of money.

The MaxDB database used in this workload was version 7.6.06_09, which has two IO threads per volume. Note that the focus in this phase is the IO throughput and not the virtual CPU (vCPU) usage. Although the vCPU usage in this phase may also be interesting, the current configuration does not allow specific IO and CPU monitoring for every virtual machine.

There are two exports available, a smaller database (NW70SR2) and a larger one (ERP2005). For an effective IO test, it is recommended to use the ERP2005 export, which is approximately 90GB in size. All further explanations use the following layout of the database, which is meant to perform well in the case where all data files are different LUNs:

```
LOGDEVS=”4GB /usr/sap/${SAPSYSTEMNAME}/devspaces”

ONE_DATADEVS="20GB /usr/sap/${SAPSYSTEMNAME}/devspaces"

DATADEVS="${ONE_DATADEVS} ${ONE_DATADEVS} ${ONE_DATADEVS} $
{ONE_DATADEVS} ${ONE_DATADEVS}"
```

An export consists of different files for each SAP content (i.e., application content is in a different file than documentation content). Furthermore, each content file is separated into four different types of files. These four types are TOC-, STR-, EXT-, and data files. The first three types are command files with no real application data content. The data files hold the compressed content of the database.

4.1.1.1 Create Database (Write-Only)

The first portion is creating the log volume having 2GB and later creating the six data volumes, each having 15GB of size concurrently. This leads into six MaxDB kernel threads performing write operations on the physical devices simultaneously. This first phase is basic IO write performance.

4.1.1.2 Load Database (Read/Write)

The second portion is the database load phase (the main import) with lots of writes for the data and also reads needed for index creation. The import of the data in this scenario is as follows. During the script based installation, a script (import.sh) is executed. This file checks if all export files are available. If so, it generates the R3load command files which contain the instructions for the import. Afterward, it starts an R3load process for each different command file (e.g., documentation content, application content). The R3load process imports the data into the database. The number of concurrent R3load processes is determined by the number

of available CPUs. The number of concurrent R3load processes can double the amount of available CPUs. In this case, we use nine R3load processes run at one time (CPUs plus 1). After a job is imported, the script starts a new R3load process which then imports the next one. During the last minutes of the load, it may happen that only one R3load process is running because only one job is left. During the import, a few computing resources are used to unpack the content data and to modify the unpacked data to fit into the database. Besides the small overhead of computing the IO, throughput is still the main factor to determine the time of the load phase.

4.1.1.3 Update Database Statistics (Read-Only)

Updating the database statistics, which are database reads only, is the last of the three sub measurements. The MaxDB command used is:

```
sql_updatestat ${DB_SCHEMA}.* ESTIMATE SAMPLE 1000 ROWS
```

This command updates the statistics for the SQL optimizer.

While it can be argued that some of the operations are done using the database cache instead of accessing the data volumes. This would invalidate the argument that loading a database with SAP content is a good IO benchmark. But the way in which the data is imported is very IO intensive because after each table and its associated content is imported, a MaxDB “save point” is triggered. Such a save point writes the entire cache content physically to disk. Furthermore, the log writer logs every action into the log volume, which is set to auto overwrite to prevent a log full situation. At the moment, not many systems have a greater than 90GB filesystem cache available. Furthermore, the reads for creating the indexes and updating the statistics cannot be fulfilled by the cache.

4.1.2 Memory Load Test

Note that there is no connection or even a correlation between the SLCS memory load test and the SD-benchmark. It was developed to provide a test requiring no user input that applies a heavy load on the system. This new test should be started by typing a single command in the shell. As there is no user interaction, the memory load gets its information about work processes and memory distribution itself, restarts the system, and starts a defined number (dependent on CPUs present) of ABAP reports doing the load. There are no adjustable variables such as user number, response time, or duration time ... but there is another metric: throughput per work process. The duration of the memory load runs does not matter because the measurement is stopped exactly 15 minutes after the start of the memory load process and the throughput is displayed in the command line after the run. The value is the throughput of one dialog process. As there are three dialog processes configured for every available CPU, one must multiply the result by three and the number of CPUs to get the overall system CPU throughput.

The ABAP memory allocation report generates an internal ABAP table of a defined size by writing random data into this table. It then loops over this table for a defined period of time. During each loop, it reads one data set randomly. The overall number of loops per second is displayed after approximately 18 minutes.

The most important factors are the defined size, defined time, and number of reports started

for each available CPU. The defined size of each internal table is 300MB. The defined run time is 15 minutes, which means after that time elapses the execution is stopped and the overall loops are written to file. For every available CPU in the system, the script configures three dialog work processes. For every available dialog work process, one report is started. For example, having a two CPU virtual machine, the script will configure a total of six work processes, restarts the system and launches six *cpu_load* reports. Allocating 900MB of memory per CPU also sets the minimum amount of available memory to approximately 1GB per CPU.

4.2 Performance Test Plan

Scale-out:
The scale-out data set highlights the results of scaling a number of independent, concurrent 2-vCPU, 4-vCPU, or 8-vCPU guests executing the SAP SLCS workload.

Scale-up:
The scale-up data set was collected by increased the number of vCPUs and guest memory while repeating the workload on a single guest.

Virtualization Efficiency:
Efficiency is shown by comparing the data when all the physical CPUs are allocated to executing the workload using the bare metal host (no virtualization), eight 2-vCPU guests, four 4-vCPU guests, and two 8-vCPU guests.

4.3 Tuning & Optimizations

The host OS installed was Red Hat Enterprise Linux 5.4 beta, made available via RHN. The primary purpose of this system is to provide a KVM hypervisor for guest virtual machines. Several processes deemed unnecessary for this purpose were disabled using the `chkconfig` command on the host.

auditd	iscsi	rpcgssd
avahi-daemon	iscsid	rpcidmapd
bluetooth	isdn	rpcsvcgssd
cmirror	kdump	saslauthd
cpuspeed	libvirtd	sendmail
cups	mcstrans	setroubleshoot
gpm	mdmonitor	smartd
haldaemon	modclusterd	xend
hidd	pcscd	xendomains
hplip	restorecond	xfs
ip6tables	rhnsd	xinetd
iptables	ricci	yum-updatesd

The host itself also had the same list of services disabled, excluding libvirtd which is required for the virtualization manager user interface.

Security Enhanced Linux (SELinux) was also disabled.

Each guest was started using the `virt-manager` user interface. An example .xml file, used to start each guest, can be found in **Appendix A**.

5 Test Results

Multiple factors can effect scaling. Among them are hardware characteristics, application characteristics and virtualization overhead.

Hardware:
The most prominent hardware characteristics relevant to the tests in this paper include limited storage throughput and system architecture. While the disk IO requirements of a single database instance are not extreme, the situation changes quickly as multiple systems are executed in parallel.

The system architecture includes hyper-threading technology which provides a boost in performance beyond eight cores. However, the performance of the two threads on a hyper threaded core is not expected to be equal that of to two non-hyper threaded cores as Linux treats each processing thread as a separate CPU. The impact of hyper-threading can often be minimized by assigning (aka: pinning) vCPUs to complete cores to avoid crossing memory boundaries. For the purposes of this testing, however, no CPU pinning was used.

The system architecture also includes NUMA, which allows faster access to nearby memory, albeit slower access to remote memory. This architecture has two NUMA nodes, one for each processor. Restricting a process to a single NUMA node allows cache sharing and memory access performance boosts.

Application:
The specific scaling, up (increased amounts of memory and CPU) or out (multiple instances of similar sized systems), can effect various applications in different ways. The added memory and CPU power of scaling up will typically help applications that do not contend with a limited resource, where scaling out may provided a multiple of the limited resource. Conversely, scaling out may not be suited for applications requiring a high degree of coordination for the application, which could occur in memory for a scale-up configuration. Additionally, virtualization can be used to consolidate multiple independent homogenous or heterogeneous workloads onto a single server.

Virtualization Overhead:
As it is not completely running directly on physical hardware and requires the hypervisor layer (which consumes processing cycles), some performance cost is associated with any virtualized environment. The amount of overhead can vary depending on the efficiency of the hypervisor and of the assorted drivers used.

5.1 Scaling Multiple 2-vCPU Guests

This section presents the results obtained when running multiple 2-vCPU guests (each running an independent SLCS workload) on a two-socket, quad-core, Intel Nehalem host having 8 cores = 16 hyper-threads. Note: 1 vCPU = 1 hyper-thread.

Figure 3 is a schematic illustrating the configuration as multiple 2-vCPU guests are added.

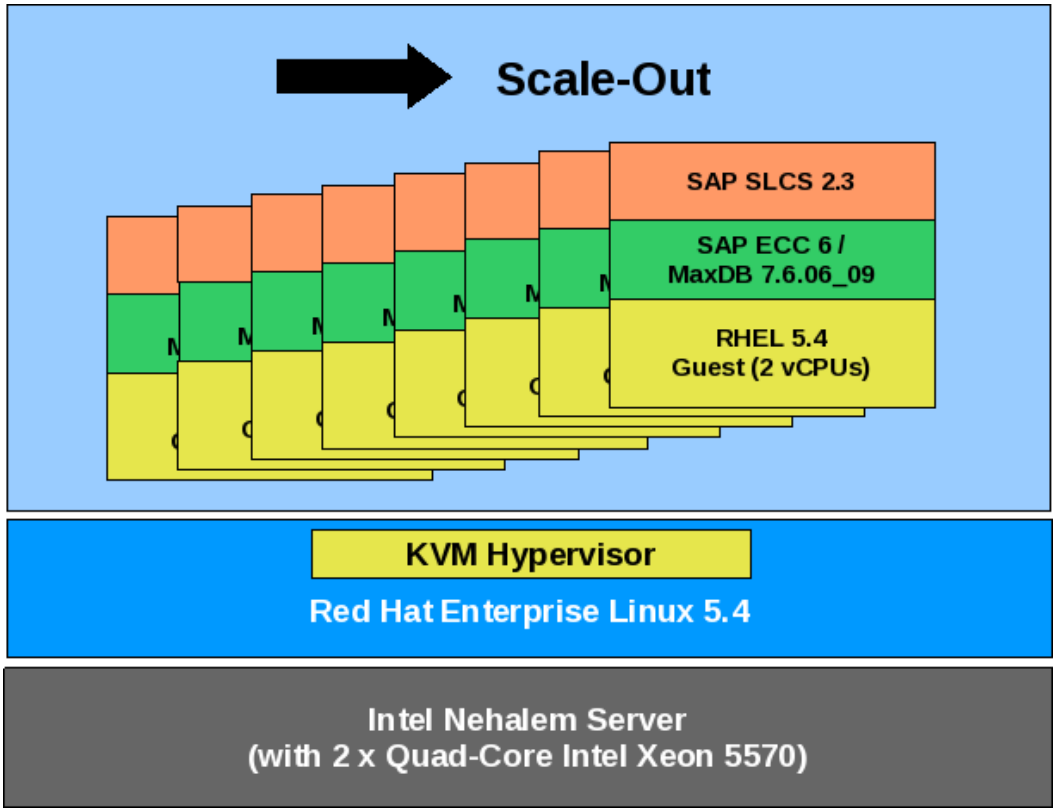

Figure 3: Scaling Multiple 2-vCPU Guests

Figure 4 plots the performance of the three phases of IO for the SLCS database load test; creation (write-only), initialization (read-only), and load (read-write). Several factors can influence these numbers including the amount of disks, their type, RAID settings, storage subsystem types, type of controllers, type of storage adapters and their respective speeds.

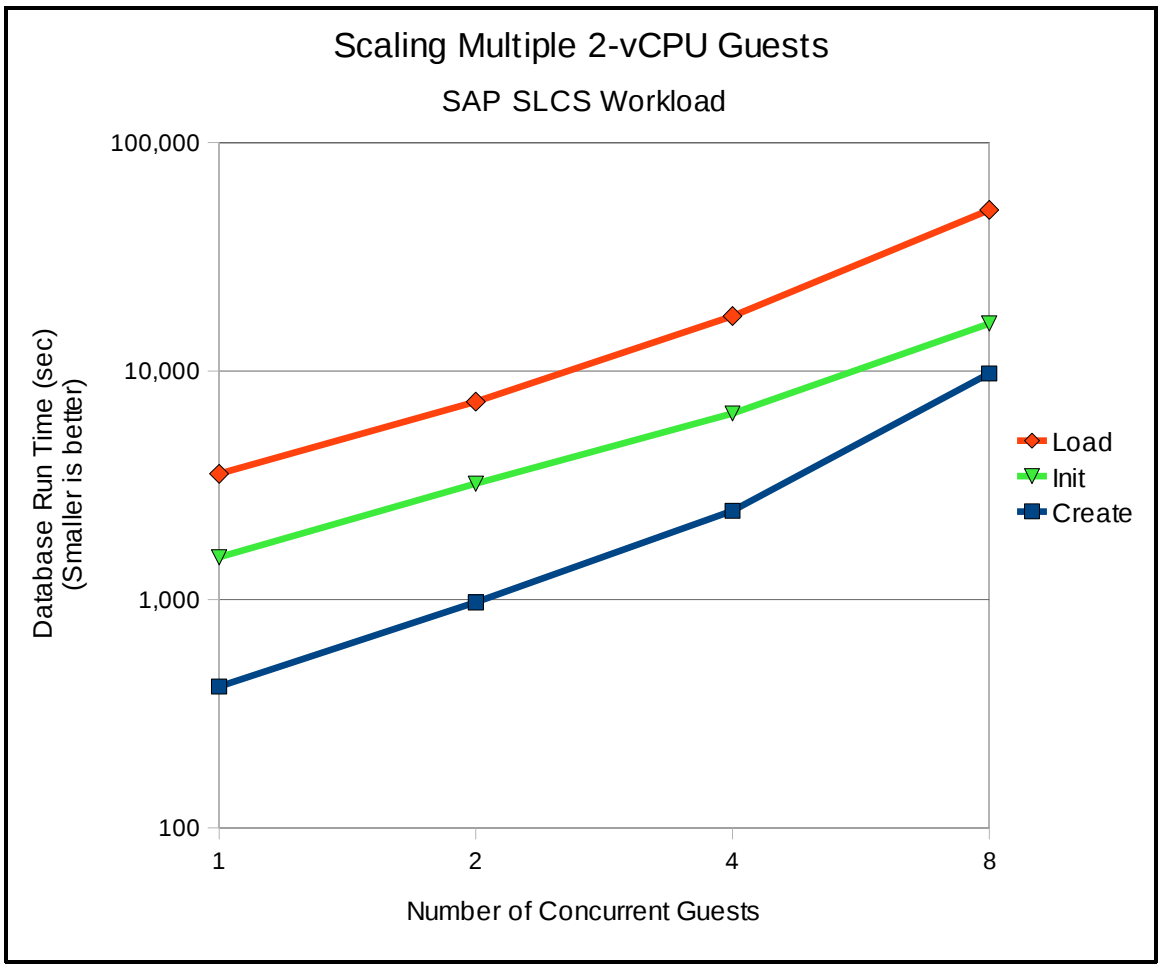

Figure 4: Scaling Multiple 2-vCPU Guests - Database Run Times

The graph shows the load times increasing as concurrent guests each perform the same activity. Note that the resulting slope is somewhat misleading given that values for three, five, six, and seven guests are not plotted. The increased load times as more guests are added are expected due to the increased IO contention as more guests increasingly utilize the host's IO bandwidth.

Figure 5 graphs the scalability achieved by increasing the number of 2-vCPU guests from one up to eight. The throughput demonstrates near-linear scalability up to four guests. Beyond four guests, the scalability is slightly lower with increased reliance on hyper-threading on this two-socket, quad-core server with 8 cores and 16 hyper-threads (Note 1 vCPU = 1 hyper-thread).

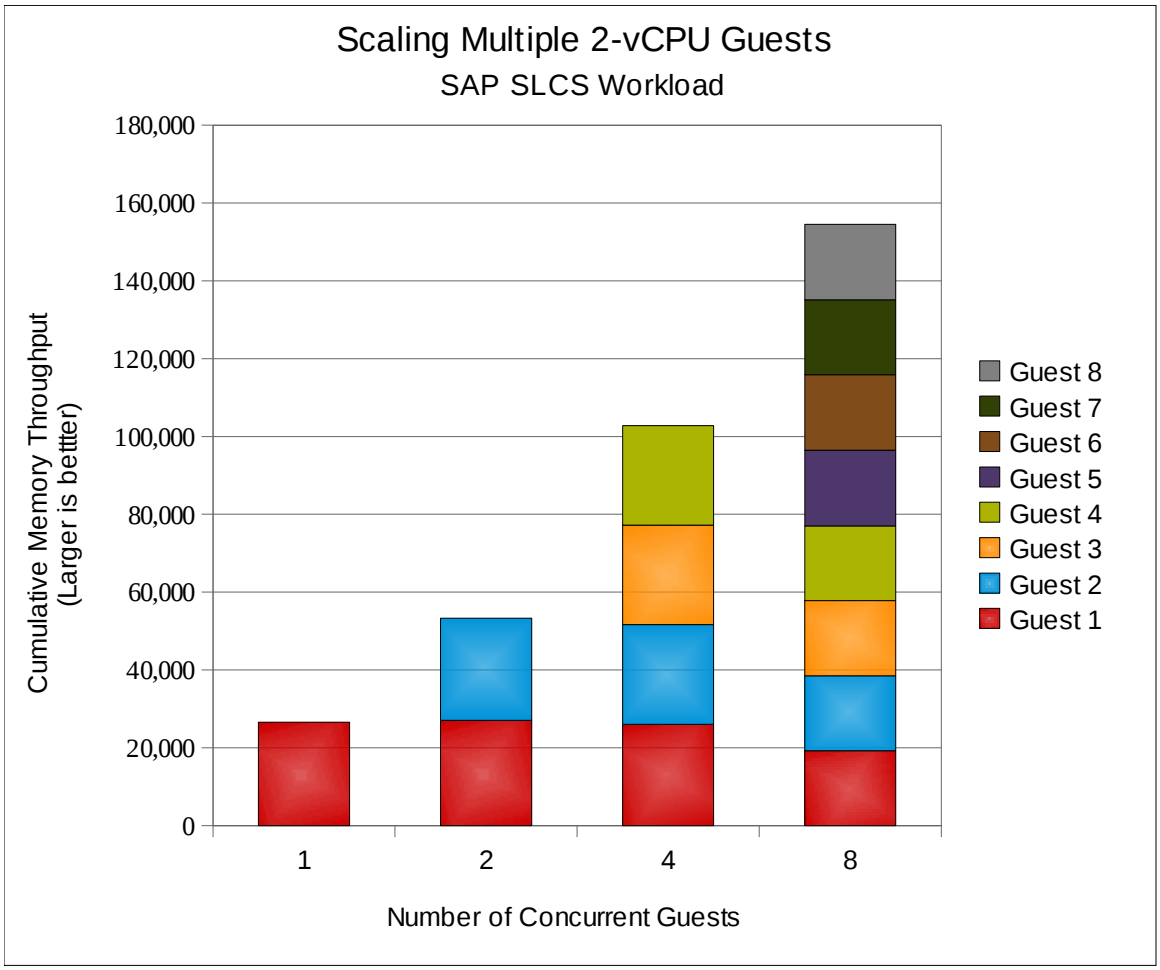

Figure 5: Results of Scaling Multiple 2-vCPU Guests

5.2 Scaling Multiple 4-vCPU Guests

This section presents the results obtained when running multiple 4-vCPU guests (each running an independent SLCS workload) on a two-socket, quad-core, Intel Nahalem host having 8 cores = 16 hyper-threads. Note: 1 vCPU = 1 hyper-thread.

Figure 6 illustrates the configuration as multiple 4-vCPU guests are added.

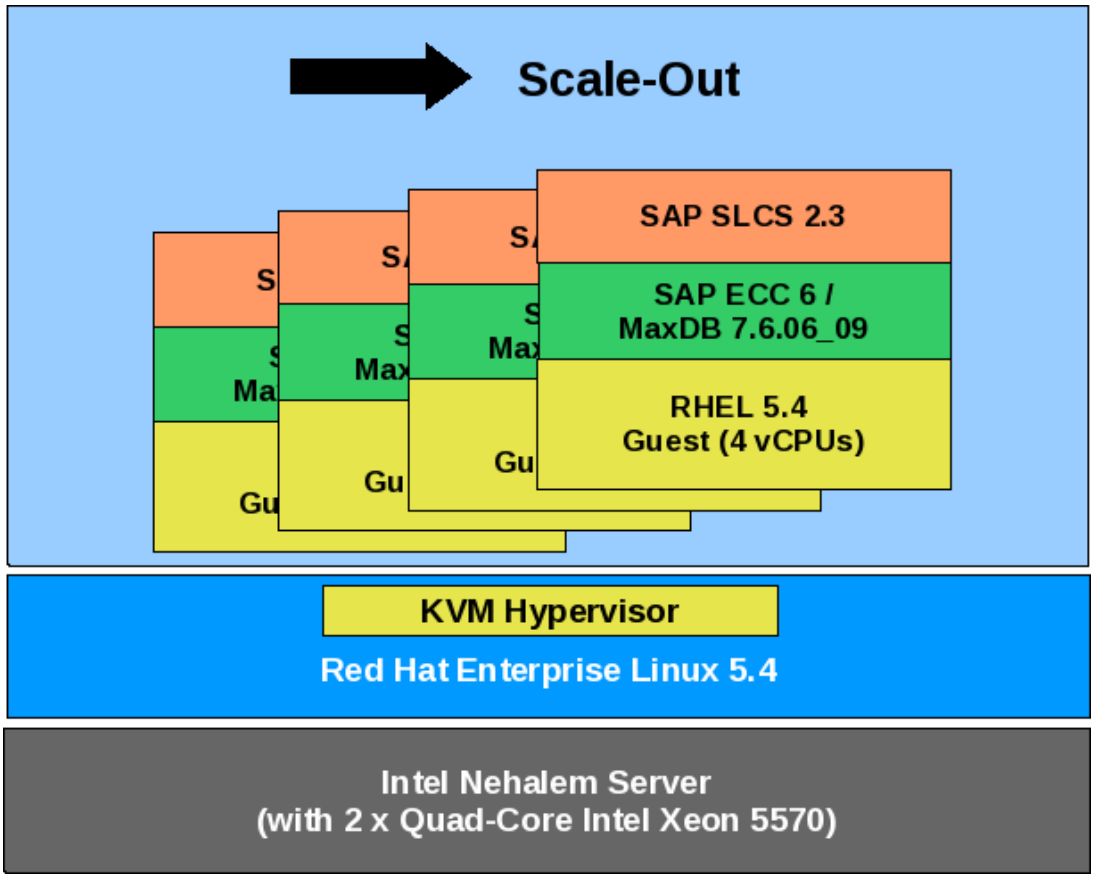

Figure 6: Scaling Multiple 4-vCPU Guests

Figure 7 plots the performance of the IO for the 4-vCPU SLCS database load test. Again limited bandwidth to storage plays a factor as all guests (individual processes on the host) contend for IO bandwidth.

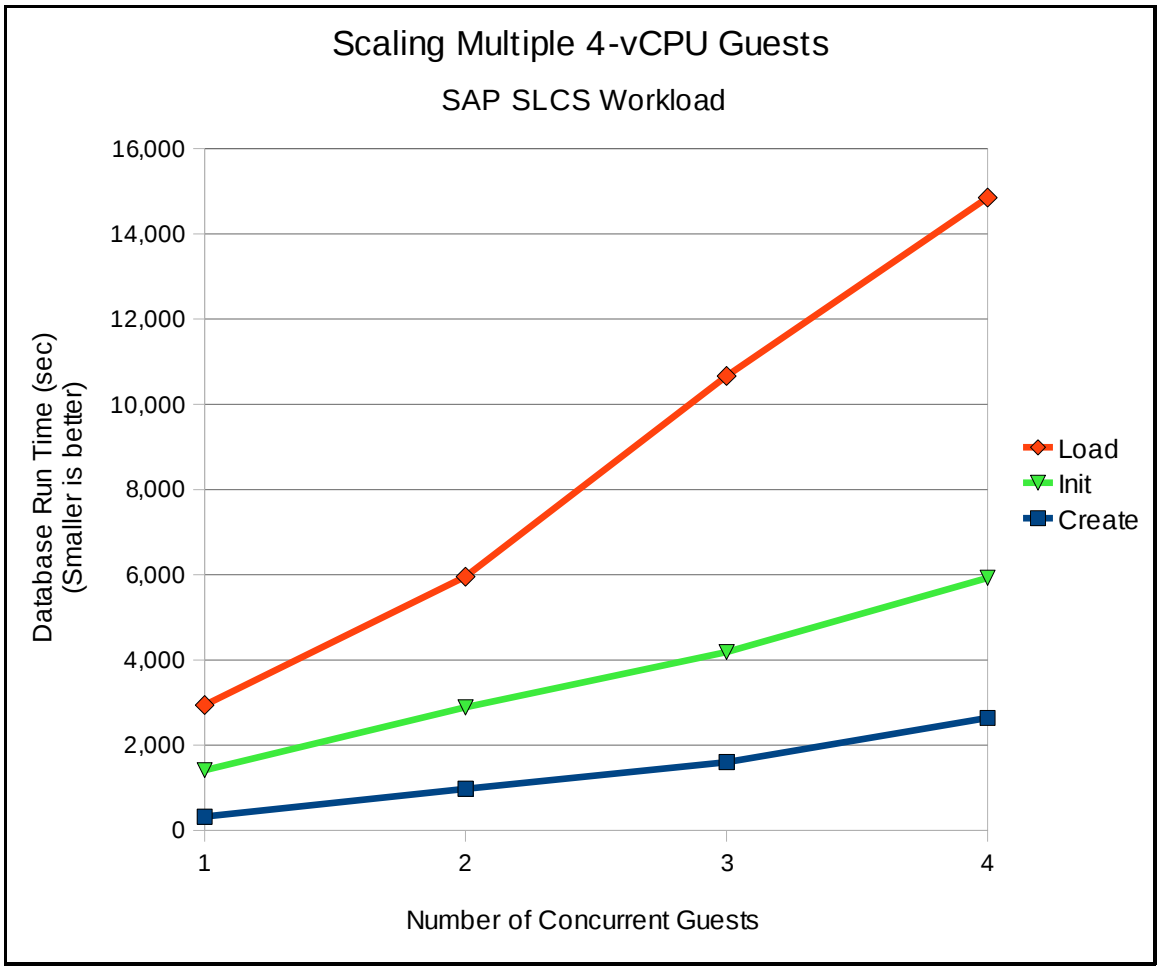

Figure 7: Scaling Multiple 4-vCPU Guests - Database Run Times

The testbed configured for this testing was not equipped with a large storage farm and consequently not designed for maximum IO bandwidth with each guest possessing only three 60GB LUNs, LVM striped to a single volume for /usr/sap.

Figure 8 graphs the scalability achieved by increasing the number of 4-vCPU guests running the independent SLCS workloads. The throughput demonstrates good scaling. However, after three guests, the throughput per guest decreases slightly due to increased use of hyperthreads and virtualization overhead.

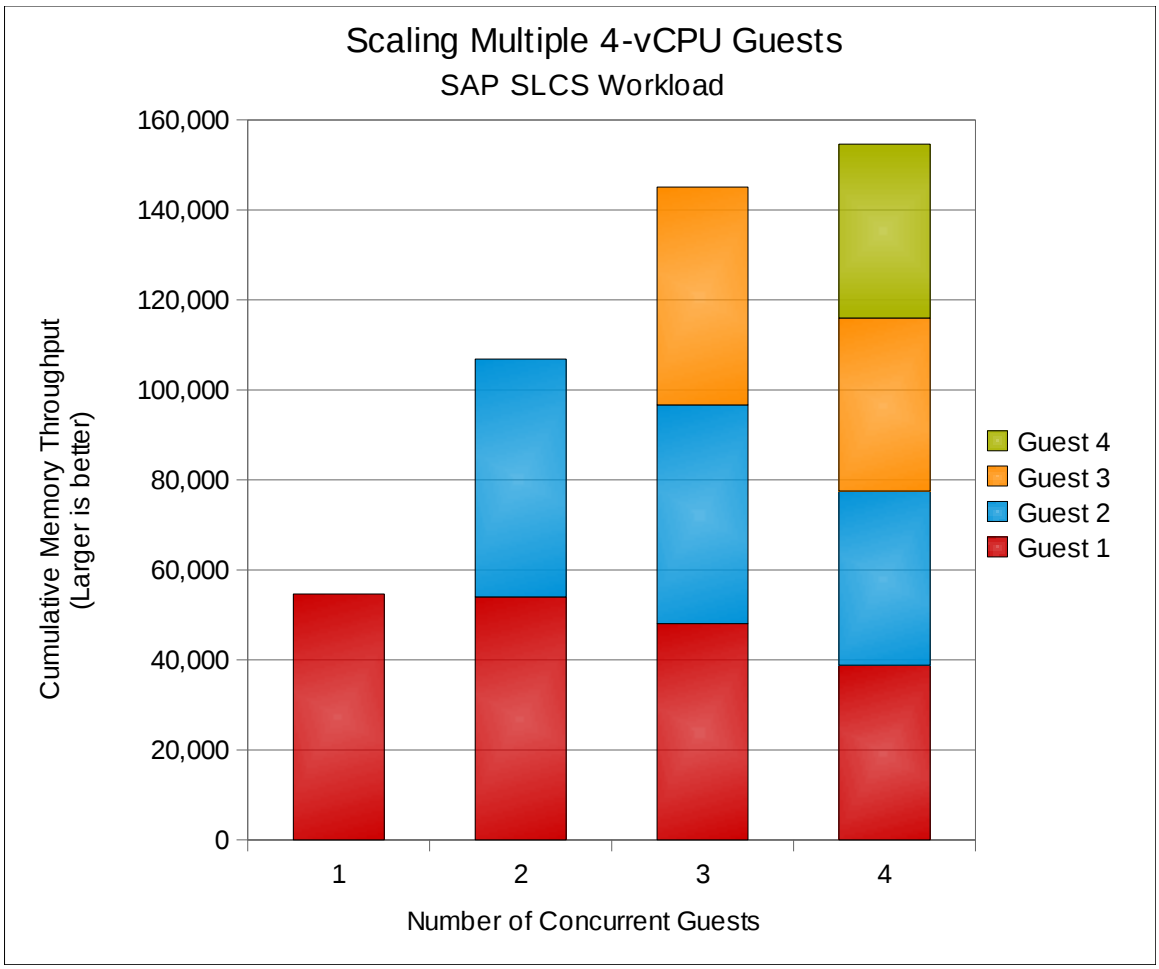

Figure 8: Results of Scaling Multiple 4-vCPU Guests - Memory Throughput

5.3 Scaling-Up the Memory and vCPUs in a Single Guest

This section presents the results obtained when running an SLCS workload on a single guest with increasing amounts of memory and vCPUs.

Figure 9 illustrates the configuration as vCPUs and memory are added.

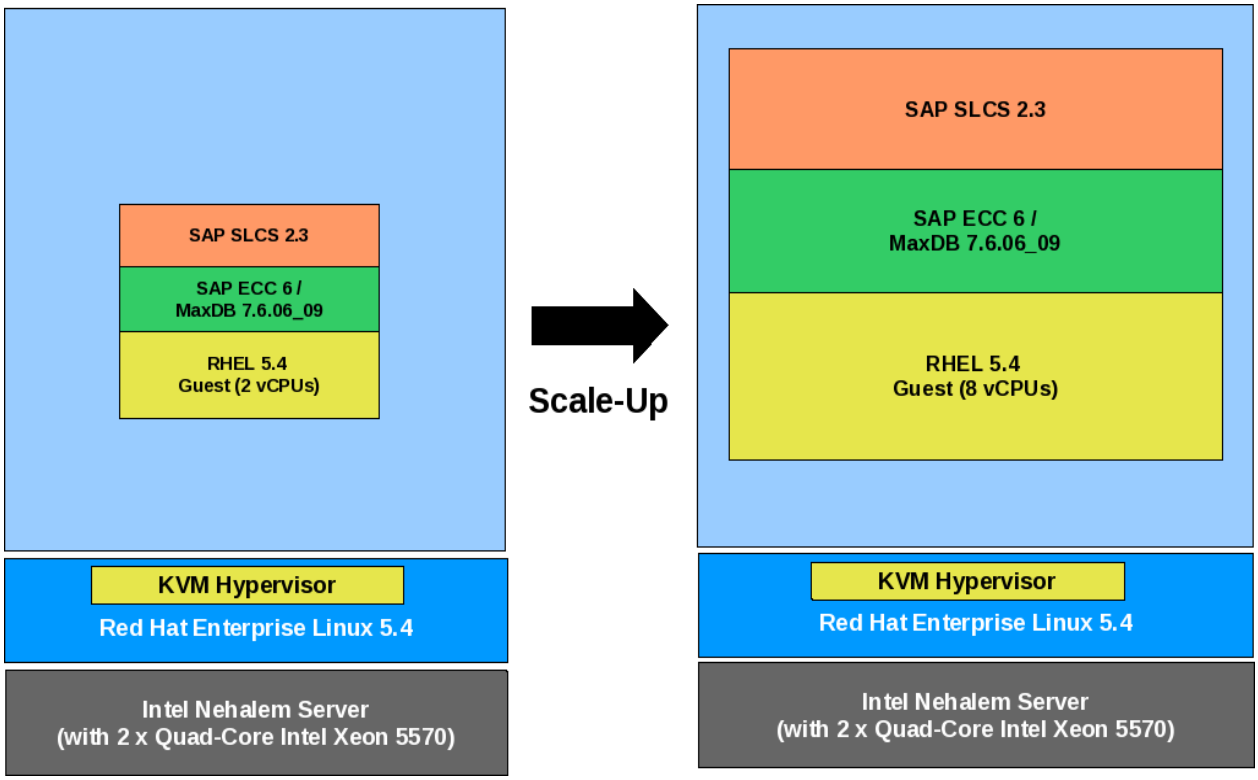

Figure 9: Scaling the Memory and vCPUs in a Single Guest

Figure 10 graphs the results when the SLCS workloads were executed on a guest using 2, 4, and 8 vCPUs with 2.5GB of memory for each vCPU. The results demonstrates excellent scaling of throughput as vCPUs are added to a guest.

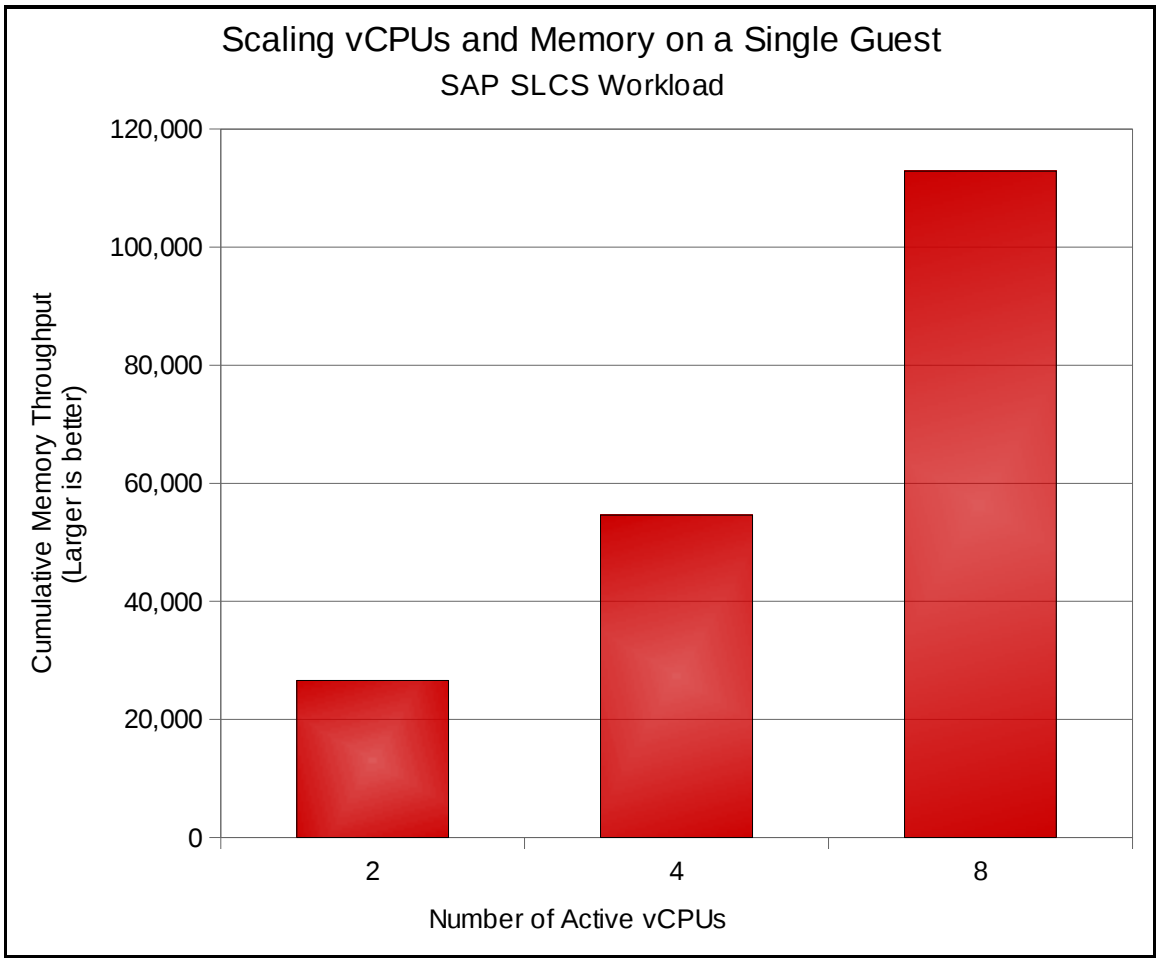

Figure 10: Results of Scaling the Memory and vCPUs in a Single Guest – SAP Memory Throughput

5.4 Consolidated Virtualization Efficiency

Figure 11 compares the throughput performance of an eight-core (16 hyper-thread) bare-metal configuration to various virtual machine configurations totaling 16 vCPUs. In the virtual environment, this test was run with eight 2-vCPU guests, four 4-vCPU guests, and two 8-vCPU guests.

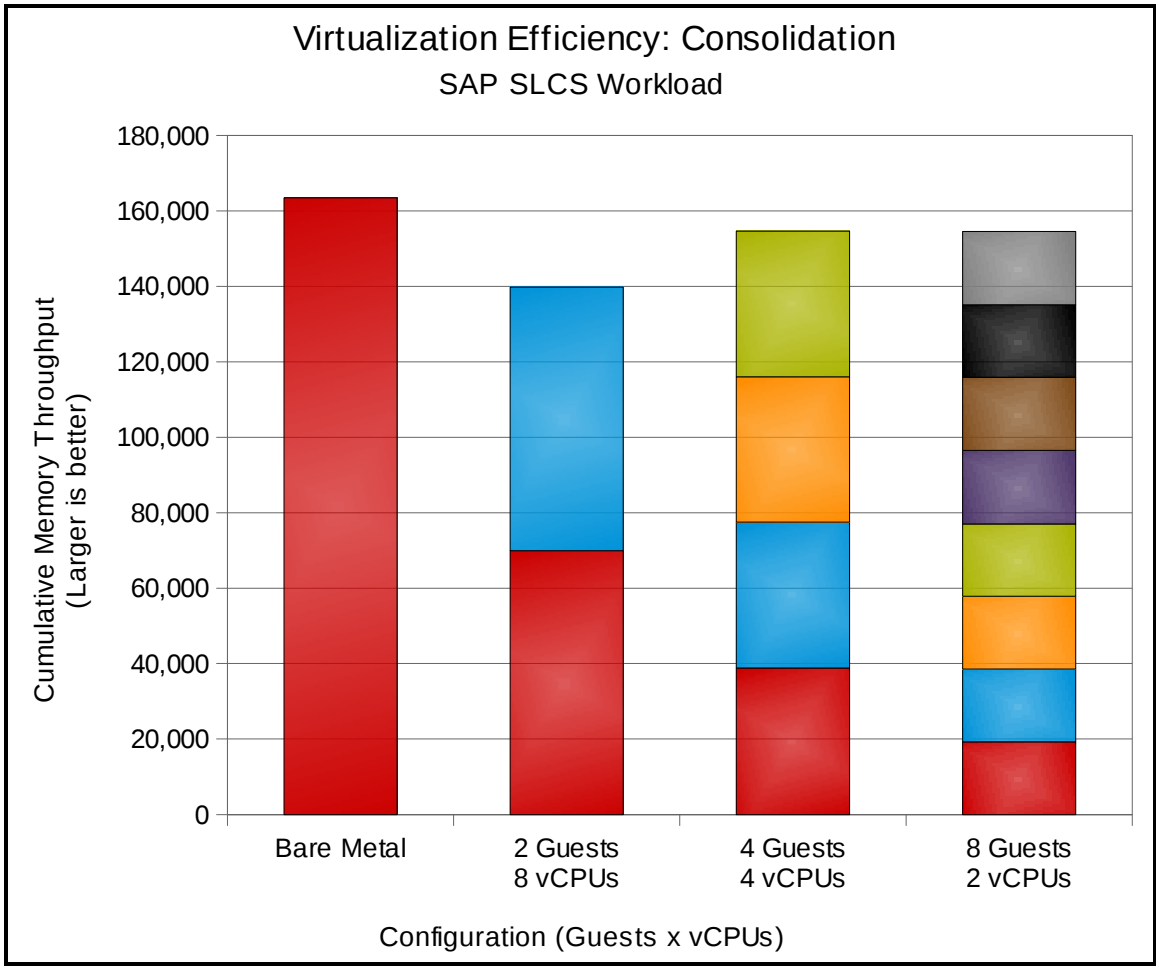

Figure 11: Virtualization Efficiency

The results indicate that comparable performance in virtualized environments can be achieved by scaling the number of guests.

6 Conclusions

This paper describes the performance and scaling of the SAP SLCS workload running in Red Hat Enterprise Linux 5.4 guests on a Red Hat Enterprise Linux 5.4 host with the KVM hypervisor. The host system was deployed on a dual socket, quad core, 2.93 GHz, Intel Xeon X5570 Nehalem system with 48GB of RAM and support for hyper-threading technology, comprising a total of 8 cores and 16 hyper-threads.

The data presented in this paper clearly establishes that Red Hat Enterprise Linux 5.4 virtual machines using the KVM hypervisor on a HP ProLiant DL370 provide an effective production-ready platform for hosting multiple virtualized SAP workloads. The combination of low virtualization overhead and the ability to both scale-up and scale-out contribute to the effectiveness of KVM for SAP. The number of actual users and throughput supported in any specific customer situation will, of course, depend on the specifics of the customer application used and the intensity of user activity. However, the results demonstrate that in a heavily virtualized environment, good throughput was retained even as the number and size of guests/virtual-machines was increased until the physical server was fully subscribed.

7 References

1. Qumranet White paper: KVM – Kernel-bsed Virtualization Machine
2. Documentation for SLCS v2.3.1, Hannes Kuhnemund, LinuxLab, SAP AG, Walldorf

Appendix A: Sample .xml File

The example content below was used to start a 4-vCPU guest with 10GB of memory.

```
<domain type='kvm' id='1'>
  <name>slcs1</name>
  <uuid>1c00c62f-1e57-b46e-66db-c7997bf851b4</uuid>
  <memory>10000000</memory>
  <currentMemory>10000000</currentMemory>
  <vcpu>4</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='block' device='disk'>
      <source dev='/dev/mapper/slcs_os1'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='block' device='disk'>
      <source dev='/dev/mapper/slcs_data51'/>
      <target dev='vdb' bus='virtio'/>
      <shareable/>
    </disk>
    <disk type='block' device='disk'>
      <source dev='/dev/mapper/slcs_data101'/>
      <target dev='vdc' bus='virtio'/>
      <shareable/>
    </disk>
    <disk type='block' device='disk'>
      <source dev='/dev/mapper/slcs_data151'/>
      <target dev='vdd' bus='virtio'/>
      <shareable/>
    </disk>
    <interface type='bridge'>
      <mac address='54:52:00:02:12:5d'/>
      <source bridge='breth0'/>
      <target dev='vnet0'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
```

```
      <source path='/dev/pts/4'/>
      <target port='0'/>
    </serial>
    <console type='pty' tty='/dev/pts/4'>
      <source path='/dev/pts/4'/>
      <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='5900' autoport='yes' keymap='en-us'/>
    <sound model='es1370'/>
  </devices>
</domain>
```