



Red Hat Reference Architecture Series

Scaling SAP in a Red Hat Enterprise Virtualization 3 Environment

John Herr, Senior Software Engineer
RHCA, RHCVA

Version 1.0
April 2012





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

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 and ABAP is/are the trademark(s) or registered trademark(s) of SAP AG in Germany and in several other countries.

UNIX is a registered trademark of The Open Group in Germany and other countries.

Intel, the Intel logo and Xeon are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

© 2012 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. 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

Send feedback to refarch-feedback@redhat.com



Table of Contents

1 Executive Summary.....	1
2 Red Hat Enterprise Virtualization.....	2
2.1 RHEV Hypervisor.....	2
2.2 Red Hat Enterprise Virtualization.....	3
3 Reference Architecture Environment.....	5
3.1 Environment.....	5
3.2 Servers.....	6
3.3 Storage Arrays.....	7
4 Test Methodology.....	9
4.1 Scaling Out.....	9
4.2 Scaling Up.....	10
4.3 Bare Metal.....	11
4.4 Tuning.....	12
4.4.1 Hypervisor.....	12
4.4.2 Test System.....	13
4.5 SLCS.....	13
4.5.1 Database Load Test.....	13
4.5.2 Memory Load Test.....	14
4.5.3 Test Execution.....	14
4.5.4 Read Test Results.....	16
5 Test Results.....	18
5.1 Factors Affecting Scaling.....	18
5.2 Scaling Out.....	19
5.2.1 One vCPU Guests.....	19
5.2.2 Two vCPU Guests.....	21
5.2.3 Four vCPU Guests.....	23
5.2.4 Eight vCPU Guests.....	24
5.3 Scaling Up.....	26
5.4 Scaling Efficiency.....	28
6 Conclusion.....	30



Appendix A: Test Configurations.....	31
A.1 One vCPU.....	31
A.2 Two vCPUs.....	32
A.3 Four vCPUs.....	32
A.4 Eight vCPUs.....	33
A.5 Sixteen vCPUS.....	33
A.6 Bare Metal.....	33
Appendix B: Scripts.....	34
B.1 run_test.sh.....	34
B.2 Kickstart %post section.....	36
B.3 run_test.sh.....	37
Appendix C: Hooks.....	39
C.1 Hypervisor Configuration.....	39
C.2 Red Hat Enterprise Virtualization Manager Configuration.....	41
C.3 Guest Configuration.....	43
Appendix D: Contributors.....	46
Appendix E: Revision History.....	47



1 Executive Summary

This paper discusses the scaling of *SAP* running in Red Hat Enterprise Linux 6 virtual machines. The virtual machines are running in a Red Hat Enterprise Virtualization 3.0 environment using a Red Hat Enterprise Linux 6 server as the hypervisor. The hypervisor is running on a Dell PowerEdge R810 server with two CPU sockets populated with 2.26GHz Intel Xeon 7560 Nehalem-EX processors each with eight cores. The Hyper-Threading support in the processors is disabled. The system contains 128 GB of system RAM.

Once all scaling tests in the virtual environment are complete, the hypervisor server is re-installed with Red Hat Enterprise Linux 6 and configured in the same manner as the virtual machines. Scaling tests are run on the bare metal system and the results are used as a basis of comparison.

Version 2.3 of the **SAP LinuxLab Certification Suite (SLCS)** running the *ERP2005 SR1* workload is used to perform the scaling tests for this paper. Two distinct tests are performed for scaling purposes, the *DATABASE LOAD* test and the *MEMORY LOAD* test. The *DATABASE LOAD* test is broken into 3 parts: database creation, database loading, and database updating.

Scaling Out

The **SLCS** tool is executed on the guests as the number of identical guests are increased. Guests containing one, two, four, and eight vCPUs are configured and the **SLCS** tool is executed on each concurrently. Each guest is allocated 7.5 GB of memory and a single storage LUN per vCPU assignment.

Scaling Up

The **SLCS** tool is executed on the virtual machines (guests) as the resources available to each guest are increased in a linear manner. The number of virtual CPUs (vCPUs) are increased from one to sixteen. Each guest is allocated 7.5 GB of memory per vCPU assignment, therefore increasing the memory assigned to each guest respectively. The quantity of storage LUNs of identical configuration assigned to each guest is increased as the vCPUs increase.

Bare Metal Comparison

The **SLCS** tool is executed on a physical server as a scale up comparison point for efficiency. A fresh install of the operating system and SLCS test suite are performed configured identically the that of the guests.



2 Red Hat Enterprise Virtualization

2.1 RHEV Hypervisor

A hypervisor 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 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 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.

Red Hat Enterprise Virtualization uses the Kernel-based Virtual Machine (KVM)¹, which turns Linux into a hypervisor. Red Hat Enterprise Linux 5.4 provided the first commercial-strength implementation of KVM, which is developed as part of the upstream Linux community. RHEV 3.0 uses the RHEL 6 KVM hypervisor, and inherits performance, scalability and hardware support enhancements from RHEL 6.

¹ <http://www.redhat.com/promo/qumranet/>



2.2 Red Hat Enterprise Virtualization

Virtualization offers tremendous benefits for enterprise IT organizations – server consolidation, hardware abstraction, and internal clouds deliver a high degree of operational efficiency.

Red Hat Enterprise Virtualization (RHEV) combines the KVM hypervisor (powered by the Red Hat Enterprise Linux kernel) with an enterprise grade, multi-hypervisor management platform that provides key virtualization features such as live migration, high availability, power management, and virtual machine life cycle management. 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 two components:

- **RHEV Manager (RHEV-M):** A feature-rich virtualization management system that provides advanced capabilities for hosts and guests.
- **RHEV Hypervisor:** A modern, scalable, high performance hypervisor based on RHEL KVM. It can be deployed as RHEV-H, a small footprint secure hypervisor image included with the RHEV subscription, or as a RHEL server (purchased separately) managed by RHEV-M.

A **host** is a physical server which provides the CPU, memory, and connectivity to storage and networks that are used for the virtual machines (VM). The local storage of the standalone host is used for the RHEV-H executables along with logs and enough space for ISO uploads.

A **cluster** is a group of hosts of similar architecture. The requirement of similar architecture allows a virtual machine to be migrated from host to host in the cluster without having to shut down and restart the virtual machine. A cluster consists of one or more hosts, but a host can only be a member of one cluster.

A **data center** is a collection of one or more clusters that have resources in common. Resources that have been allocated to a data center can be used only by the hosts belonging to that data center. The resources relate to storage and networks.

A **storage domain** is a shared or local storage location for guest image files, import/export or for ISO images. Storage domain types supported in RHEV 3.0 are NFS, iSCSI, Fibre Channel, and local disk storage.

The RHEV **network** architecture supports both guest traffic and traffic among RHEV hypervisors and the RHEV-M server. All hosts have a network interface assigned to the logical network named *rhevm*. This network is used for the communications between the hypervisor and the manager. Additional logical networks are created on the data center and applied to one or more clusters. To become operational, the host attaches an interface to the local network. While the actual physical network can span across data centers, the logical network can only be used by the clusters and hosts of the creating data center.



Figure 2.2.1: RHEV Environment provides a graphical representation of a typical Red Hat Enterprise Virtualization environment with each component listed.

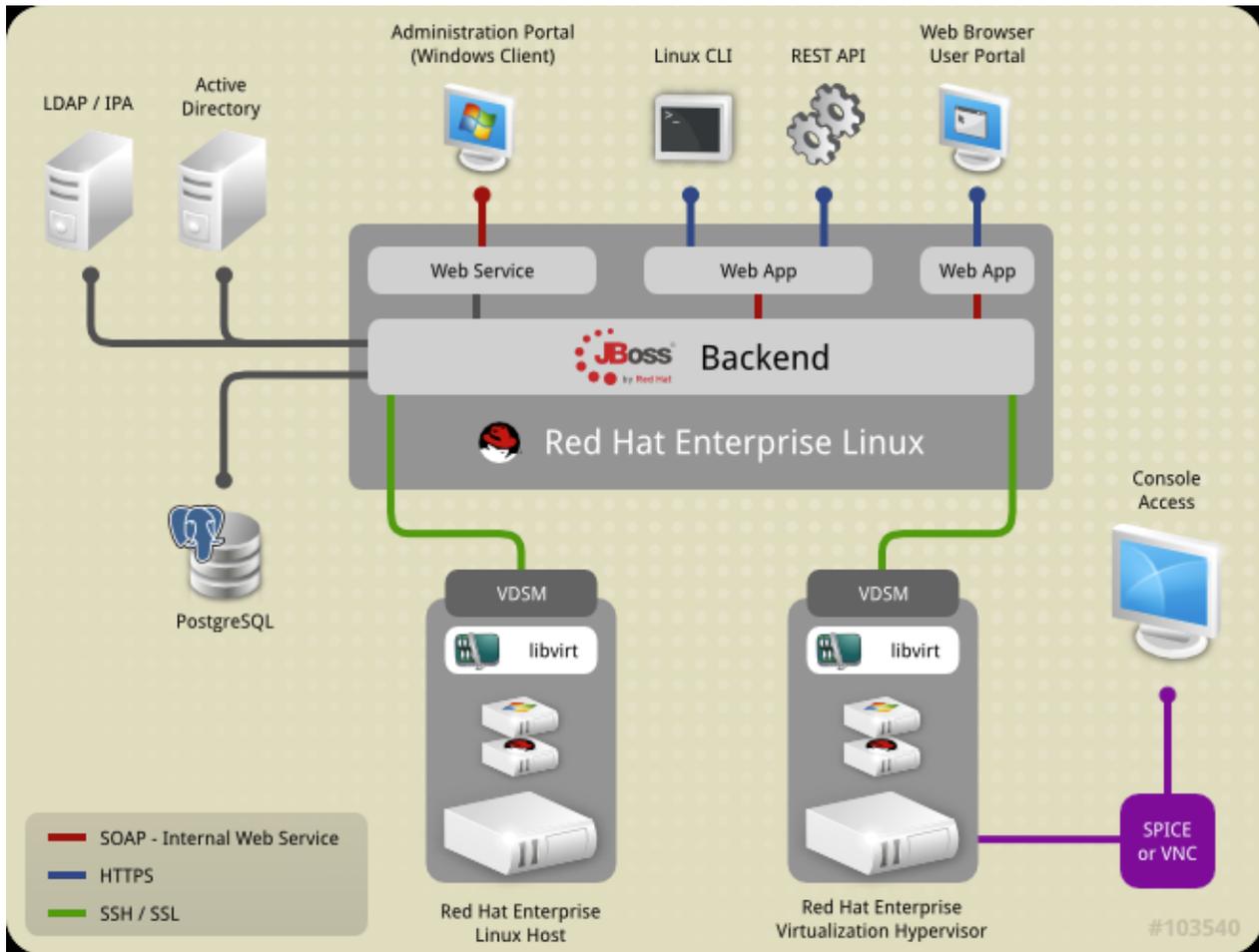


Figure 2.2.1: RHEV Environment



3 Reference Architecture Environment

3.1 Environment

Figure 3.1.1: SAP Scaling Environment depicts a simple overview of the physical environment.

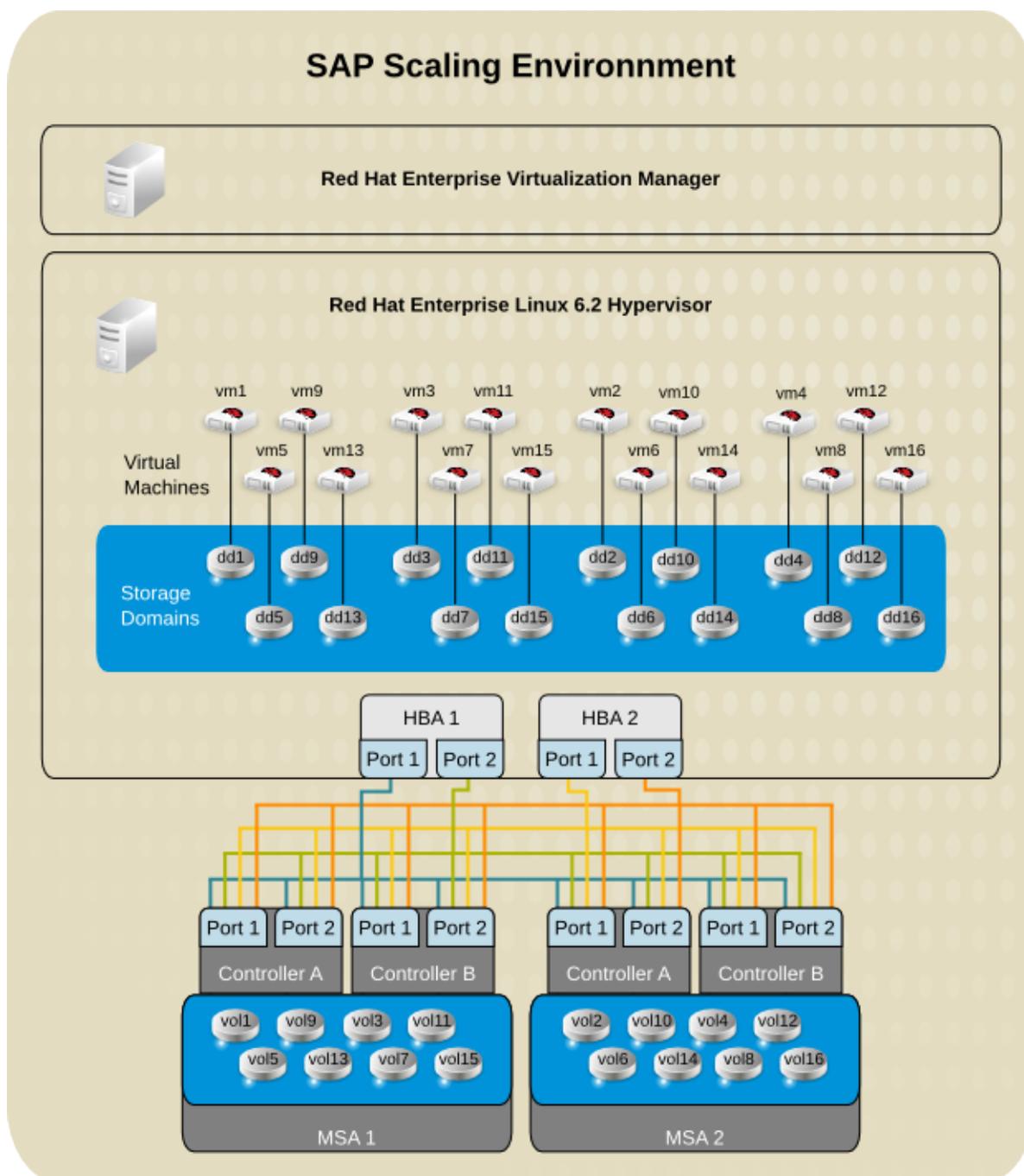


Figure 3.1.1: SAP Scaling Environment



3.2 Servers

Two physical servers are used in the environment for testing. The first server is used as the Red Hat Enterprise Virtualization Manager. The second server is used for both the hypervisor as well as the bare metal test system.

Server	Specifications
Red Hat Enterprise Virtualization Manager [HP ProLiant DL580 G5]	Red Hat Enterprise Linux 6.2 Kernel 2.6.32-220.4.1.el6.x86_64
	Red Hat Enterprise Virtualization Manager 3.0.1_0001-4.el6
	4 x Quad Core Intel Xeon X7350 CPUs @2.93GHz
	64 GB Memory
	4 x 73 GB internal Disk Drives, RAID 5
	2 x Broadcom NetXtreme II BCM5708 Gigabit Ethernet Controllers
	1 x Intel 82572EI Gigabit Ethernet Controller
Hypervisor [Dell PowerEdge R810]	Red Hat Enterprise Linux 6.2 Kernel 2.6.32-220.4.2.el6.x86_64
	VDSM 4.9-112.6.el6_2.x86_64
	2 x 8 Core Intel Xeon X7560 CPUS @2.26GHz
	128 GB Memory
	4 x 146 GB SAS Internal Disk Drives (RAID 5)
	2 x Broadcom Gigabit BASE-T MC Server Adapters
	1 x Broadcom 10 Gigabit Dual Port SFP+ Adapter

Table 3.2.1: Server Configuration



3.3 Storage Arrays

Two fibre channel storage arrays are used to hold the virtual disks used by the guests as well as the test database for the bare metal server.

Device	Specifications
2 x HP StorageWorks MSA2324fc Fibre Channel Storage Arrays	Expanded Storage: HP StorageWorks 70 Modular Smart Array with Dual Domain IO Modules Expansion Module: 2.28
	2 x Controllers
	CPLD Code Version: 8
	Hardware Version: 56
	Storage Controller Code Version: M112R14 Loader Code Version: 19.009
	Memory Controller Code Version: F300R22
	Management Controller Code Version: W441R39 Loader Code Version: 12.015
Expander Controller Code Version: 1112	
1 x HP StorageWorks 8/24 SAN Switch	Firmware Version: v6.4.0a

Table 3.3.1: Storage Configuration



The storage arrays have two virtual disks created on each of its two storage enclosures. Every virtual disk consists of a twelve physical disk *RAID 10* with a size of about 900 GB. Two 430 GB volumes are created on the virtual disks. Each storage array houses eight volumes that are presented to the hypervisor server. The volumes are mapped to allow traffic on any of the four fibre channel ports (two on each controller) to any of the four fire channel ports on the hypervisor. **Figure 3.3.1: Storage Disk Configuration** depicts the disk configuration.

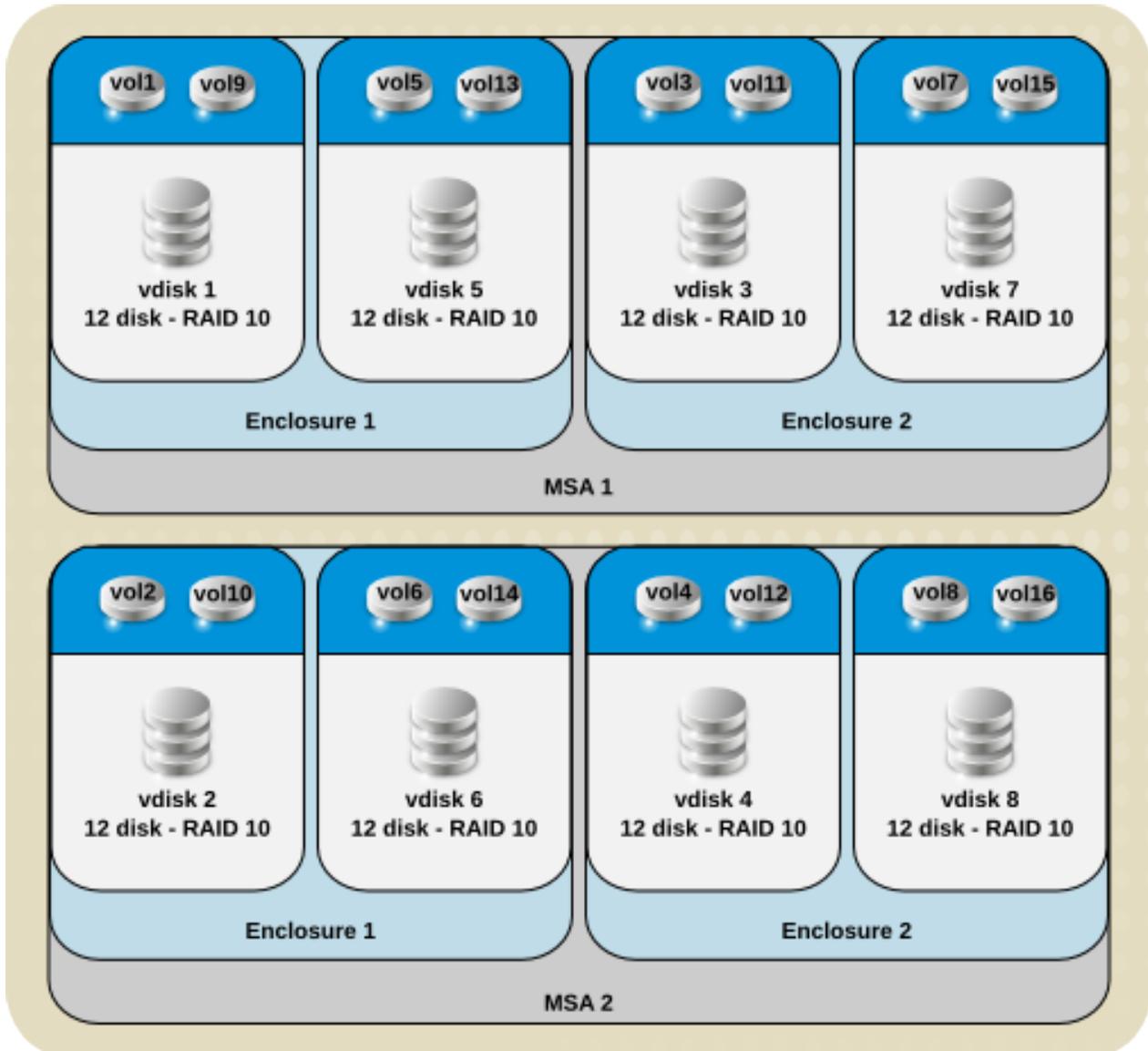


Figure 3.3.1: Storage Disk Configuration



4 Test Methodology

Using the **SLCS** tool, the effects of scaling *SAP* workloads running in a virtualized environment are observed. A base line *SAP* workload is executed on a physical server to further compare the effects of scaling.

4.1 Scaling Out

Scaling out is performed by increasing the number of identical guests concurrently running the *SAP* workload. Up to sixteen single vCPU guests can run concurrently in the testing environment, allowing the testing of 1, 2, 4, 8, and 16 guests. Only a maximum of eight guests can run if each guest is assigned two vCPUs.

The quantity of LUNs used for the test is increased as the number of vCPUs assigned to each guest is increased. Assigning more storage LUNs to the guests is done to improve disk access times by providing more disk spindles since the *DATABASE LOAD* tests do not require more storage space during the scaling tests.

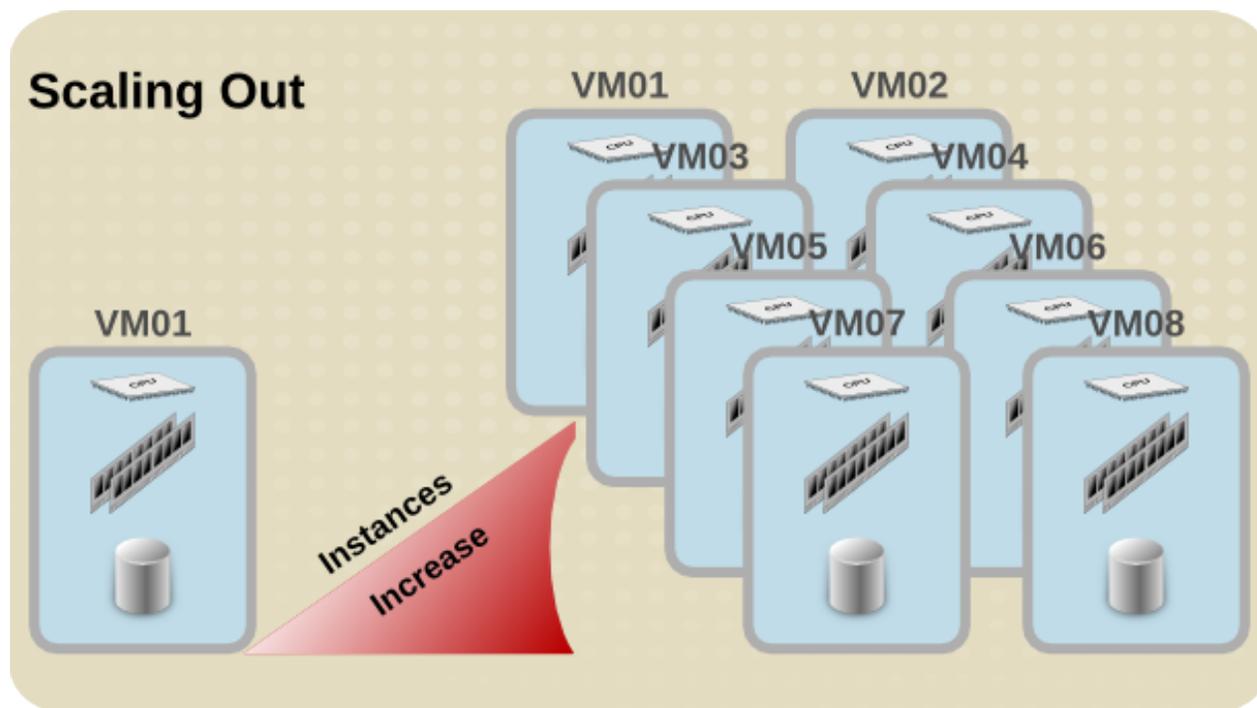


Figure 4.1.1: Scaling Out



Table 4.1.1: Scaling Out Summary shows the configuration of each guest and the tests run. See **Appendix A: Test Configurations** for more details.

# vCPUs per Guest	Memory per Guest	# LUNs per Guest	# of Guests Run per Test
1	7.68 GB	1	1, 2, 4, 8, 16
2	15.36 GB	2	1, 2, 4, 8
4	30.72 GB	4	1, 2, 4
8	61.44 GB	8	1, 2
16	122.88 GB	8	1

Table 4.1.1: Scaling Out Summary

4.2 Scaling Up

Scaling up is performed by creating a single guest and increasing the resources available to it. These resources consist of the number of vCPUs, the amount of memory, and the available storage. The quantity of LUNs used for the test is only increased to provide more disk spindles for data access since the **SLCS DATABASE LOAD** tests do not require extra storage space when scaling.

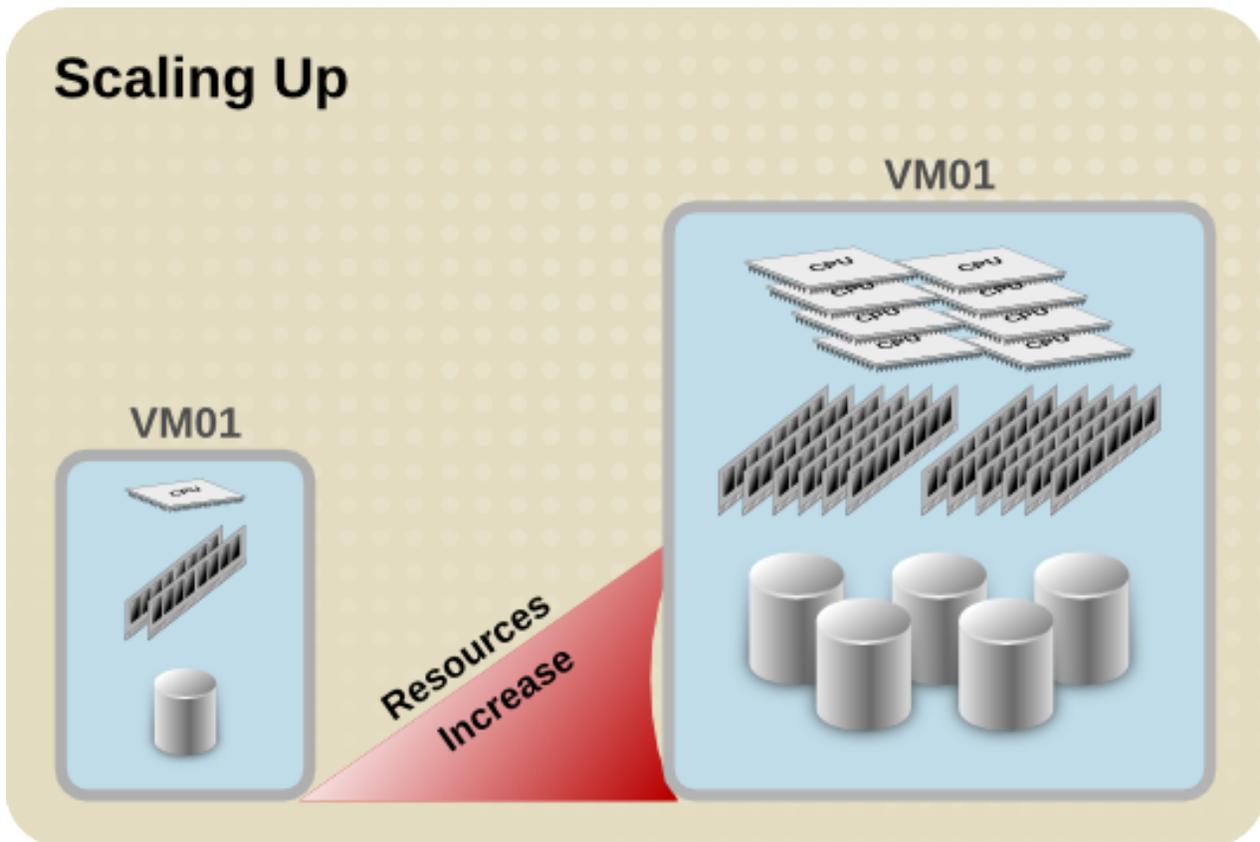


Figure 4.2.1: Scaling Up



As the vCPUs presented to the guest are increased, the amount of memory available and the storage presented is also increased. For each vCPU presented to the guest, 7.68 GB of memory is allocated. For example, a four vCPU guest is allocated 4 x 7.68 GB of memory or around 30 GB.

For each vCPU presented, a single LUN is also presented. For example, a four vCPU guest will have four luns presented to it. It is more beneficial to increase the number of spindles available to the guest instead of increasing the storage space available. Each lun presented is located on a separate virtual disk within the storage enclosure when possible, thus increasing the amount of spindles available.

Table 4.2.1: Scaling Up Summary indicates the resources allocated for each test performed. See **Appendix A:Test Configurations** for more details on the test configurations.

# vCPUs	Memory	# LUNs (spindles)
1	7.68 GB	1 (12)
2	15.36 GB	2 (24)
4	30.72 GB	4 (48)
8	61.44 GB	8 (96)
16	122.88 GB	8 (96)

Table 4.2.1: Scaling Up Summary

4.3 Bare Metal

A bare metal test provides base line information that is used to compare the efficiency of scaling. The hypervisor is re-installed with the same version of the operating system as the guests and then configured identically. The **SLCS** tests are then executed and the results recorded.

The bare metal server presents 16 physical CPUs and 128 GB of memory to the operating system. A single LUN from each virtual disk on one storage array is presented to the server. A logical volume is created from the four LUNs and is used for the test database.

Please see **Appendix A:Test Configurations** for more details on the test configurations.



4.4 Tuning

To get better performance, tuning is performed on both the hypervisor and guest operating system. **Appendix C: Hooks** contains information on configuring hooks.

4.4.1 Hypervisor

Each vCPU assigned to a guest can be assigned to run on a single physical process core. This is accomplished by using `HOOKS2` within the Red Hat Enterprise Virtualization environment and CPU pinning. This allows the workload of the guest to run on a controlled processor or set of processors.

Pinning also allows each guest to be assigned a set *NUMA* zone. To get the best performance, the guest should be pinned to the same *NUMA* zone as the physical processor core it is assigned.

The `lscpu` command is used to display the *NUMA* zones and which physical cores are assigned to each zone.

```
# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:  0-15
Thread(s) per core:    1
Core(s) per socket:    8
CPU socket(s):         2
NUMA node(s):          2
Vendor ID:              GenuineIntel
CPU family:            6
Model:                 46
Stepping:               6
CPU MHz:                2261.147
BogoMIPS:               4521.23
Virtualization:        VT-x
L1d cache:              32K
L1i cache:              32K
L2 cache:               256K
L3 cache:               24576K
NUMA node0 CPU(s):     0, 2, 4, 6, 8, 10, 12, 14
NUMA node1 CPU(s):     1, 3, 5, 7, 9, 11, 13, 15
```

² See the Red Hat Enterprise Virtualization 3.0 Administration Guide

[http://docs.redhat.com/docs/en-](http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/Administration_Guide/VDSM_Hooks.html)

[US/Red_Hat_Enterprise_Virtualization/3.0/html/Administration_Guide/VDSM_Hooks.html](http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/Administration_Guide/VDSM_Hooks.html)



4.4.2 Test System

The test system, whether it is a guest running on the virtual environment or the bare metal system, is configured to have all unnecessary services disabled and *SELINUX* enabled and set to enforcing. Only the following services are configured to run after boot.

Services		
iptables	network	sysstat
ip6tables	sshd	udev-post
lvm2-monitor	rsyslog	

Table 4.4.2.1:

The file system the **SLCS** tests run against is configured to have the *JOURNAL_DATA_WRITEBACK* option configured and to also have *WRITE BARRIERS* disabled³. This can cause data loss in production environments and is done for performance reasons in this scaling paper only. This is not recommended for the normal operation of a *SAP* environment.

4.5 SLCS

The **SAP LinuxLab Certification Suite (SLCS)** is a tool created by *SAP LinuxLab* to allow software and hardware vendors to test and benchmark *SAP* on their software and platforms. The **SLCS** tool provides several options to use for the database export, the *ERP2005SR1* export is used in this scaling paper.

Two main tests provided by the **SLCS** tool are the *DATABASE LOAD* test and the *MEMORY LOAD* test. These tests are used for hardware and virtualization certifications. Both these tests are used for this scaling paper.

4.5.1 Database Load Test

The *DATABASE LOAD* test consists of three parts: creating the database, loading of the database, and updating or initialization of the database. The *ERP2005SR1* export used for the *DATABASE LOAD* tests contains 90 GB of data that is imported into the database. This test reports its results in seconds to completion of each part.

Database Creation

Database creation involves creating the log and database volumes. This is a database write intensive operation. This is represented as *AVG CREATE* on the graphs.

Database Loading

Database loading imports the data from the *ERP2005SR1* export file. Multiple processes are started to import the data. Import processes up to two times the number of processors are started concurrently. The import process is both database read and write intensive. This is called *AVG LOAD* on the graphs.

³ http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/writebarr.html



Database Initialization

The final part of the *DATABASE LOAD* test is initializing or updating the database statistics. The **SLCS** tool executes a *MaxDB* command to update the database statistics for this read intensive operation.

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

Database initialization is represented as *AVG INIT* on the graphs.

4.5.2 Memory Load Test

ABAP, or *Advanced Business Application Programming*, reports are generated to perform the *MEMORY LOAD* test. During the test, 900 MB of memory is allocated to each CPU by the tool. An internal *ABAP* table (two dimensional array acting as a database table) of 300 MB is created. Random data is written into the table and then the table is looped over for 900 seconds. During each loop, a randomly chosen data set is read. Three *ABAP* reports are run concurrently for each available CPU in the system. On a four CPU system, twelve *ABAP* reports are run concurrently.

The results are reported as throughput per second per process. To calculate the overall throughput of the system, the result reported must be multiplied by 3 times the number of available processors.

Often, when using the *ERP2005SR1* export, the test script cannot locate all the result files from the tests, this is related to the nature of an *SAP* system. Therefore the *MEMORY LOAD* test does not finish completely. Running the tests multiple times alleviates this problem. It is suggested to run the *MEMORY LOAD* test at least three times to get a valid result. In this testing environment, the *MEMORY LOAD* tests are run four times on each guest during the test cycles.

4.5.3 Test Execution

Each system under test is installed using an identical kickstart file. The post section of the kickstart file performs the following tasks:

- Use the **chkconfig** command to ensure all unnecessary services are disabled upon boot.
- Tune the Ext file system to ensure the *JOURNAL_DATA_WRITEBACK* option is enabled.
- Modify the */etc/fstab* file to ensure the Ext file system is mounted with the *noatime*, *data=writeback*, and *barrier=0* options upon boot.
- Installed the **compat-libstdc++-33** package is installed.
- Unpack the **SLCS** tool in the */slcs2.3* directory.
- Copy the *run_test.sh* script to the */slcs2.3* directory.

See **Appendix B: Scripts** for the contents of the *%post* section of the kickstart file as well as the *run_test.sh* script.



After the test system boots, a shell is opened and the test is started. For test scenarios where multiple test systems need to run, the tests are started at the same time.

The testing procedure consists of the following steps:

- Change to the `/slcs2.3` directory.

```
# cd /slcs2.3
```

- Execute the `screen` command. This is to not only log the output of the tool, but to also keep the shell open in case the connection is lost. This prevents having to re-run the test if the connection is lost.

```
# screen -L
```

The following steps are performed by the `run_tests.sh` script listed in **Appendix B: Scripts**.

- Export the **EXPORT** variable. This variable defines the export to use for the tests. This can be any file name located in the `/slcs2.3/noarch` directory.

```
# export EXPORT="ERP2005SR1.tar"
```

- Export the **DATADEVS** variable. The **DATADEVS** variable defines the size and location of the database volumes. Since the `ERP2005SR1` export requires 90 GB of storage, the size of the database volume is set to 100 GB. The location of the database is a directory on the tuned Ext file system.

```
# export DATADEVS="100GB /sapdb"
```

- Create the database volume directory.

```
# mkdir /sapdb  
# chmod 777 /sapdb
```

- Execute the *DATABASE LOAD* test.

```
# ./install.sh
```

- Execute the *MEMORY LOAD* test. These tests should be run multiple times and the results from the last run should be used.

```
# ./memory_load.sh  
[ ... Content Removed ... ]
```

```
# ./memory_load.sh  
[ ... Content Removed ... ]
```

```
# ./memory_load.sh  
[ ... Content Removed ... ]
```

```
# ./memory_load.sh  
[ ... Content Removed ... ]
```



4.5.4 Read Test Results

The results of the test are displayed in the screen output and written in the `/var/log/slcs/BEN_*/time.log` file.

```
# ./run_tests.sh

Console command finished (2012-03-26 10:53:01).
[2012-03-26 10:53:01] - OK      Install SAP-System
[2012-03-26 10:53:01] - DBLOAD time: Create 1639s, Load 8191s, Initialize 2265s

[ ... Content Removed for Brevity ... ]

[2012-04-05 00:53:59] - started sapevt SAP_SD01
pf=/usr/sap/BEN/SYS/profile/BEN_DVEBMGS19_ssap-vm17
[2012-04-05 00:54:00] - started sapevt SAP_SD02
pf=/usr/sap/BEN/SYS/profile/BEN_DVEBMGS19_ssap-vm17
[2012-04-05 00:54:01] - started sapevt SAP_SD03
pf=/usr/sap/BEN/SYS/profile/BEN_DVEBMGS19_ssap-vm17
[2012-04-05 00:54:02] - started sapevt SAP_SD04
pf=/usr/sap/BEN/SYS/profile/BEN_DVEBMGS19_ssap-vm17
[2012-04-05 00:54:03] - started sapevt SAP_SD05
pf=/usr/sap/BEN/SYS/profile/BEN_DVEBMGS19_ssap-vm17
[2012-04-05 00:54:04] - started sapevt SAP_SD06
pf=/usr/sap/BEN/SYS/profile/BEN_DVEBMGS19_ssap-vm17
[2012-04-05 00:54:04] - waiting for them to finish...
[2012-04-05 01:10:44] - OK - memory load Test passed successfully
[2012-04-05 01:10:44] - CPU: 6 - Fails: 0 - Diff: 6
[2012-04-05 01:10:44] - Memory load throughput/sec per CPU: 12990.08

[ ... Content Removed for Brevity ... ]
```

The above output shows the results of the database load test and the *MEMORY LOAD* test for a two vCPU guest. The number of CPUs reported in the output is the number of ABAP reports ran and not the actual number of CPUs in the system. The results of the *MEMORY LOAD* test must be multiplied by the number of CPUs reported to get the total memory throughput on the system under test.

If any fails are reported, the test must be executed again. It is common to see fails when executing the *MEMORY LOAD* test the first or second time. See **Section 4.5.2 Memory Load Test**.



The following lists the relevant contents of the `/var/log/slcs/BEN_*/time.log` file. The `MEMORY LOAD` test was executed four times. The results for the first execution are empty, this indicates a fail during execution. The remaining three results contain values, but only the last result is used to calculate the total throughput.

[... Content Removed for Brevity ...]

```
<data>
  <runtimes>
    <time name="db load" create="561" load="3538" initialization="1836" />
    <memory_load>
      <time name="memory load" throughput="" quantity="6" />
      <time name="memory load" throughput="13056.05" quantity="6" />
      <time name="memory load" throughput="12869.88" quantity="6" />
      <time name="memory load" throughput="12990.08" quantity="6" />
    </memory_load>
    <liveCache_benchmark>
  </liveCache_benchmark>
</runtimes>
```

[... Content Removed for Brevity ...]



5 Test Results

5.1 Factors Affecting Scaling

Hardware, applications, and virtualization overhead are factors that affect scaling. These are briefly discussed below.

Hardware

Memory, processors, and storage are key factors in scaling applications. Lack of performance and resources with any of these can drastically impact scaling.

The amount of memory is an obvious limiting factor in scaling, but the speed of memory access also affects scalability. A feature common in newer systems is *NON-UNIFORM MEMORY ACCESS (NUMA)*. *NUMA* allows processors to access memory that is considered local to the processor quicker than memory that is not considered local. It also helps prevent multiple processors from trying to simultaneously access the same memory locations. All of this can improve performance and scalability.

The number of processors in the system, the processor speed, and hyper-threading can help or hurt scalability. The more processors and the faster their speed almost always increases performance and scalability. It goes to reason that enabling hyper-threading on the processors would also increase scalability since it presents the software with what seems to be twice as many processors. But this is not always the case. Processor cache and *NUMA* memory are shared between the hyper-thread processes on a processor, thus causing a resource bottleneck for cache and memory access.

Storage resources can impact scalability more than any other factor if the application is very IO intensive. Storage can also be the most complicated hardware to configure for the best performance. Configuration on the controller cards, RAID levels, the speed of the disks, and the number of disks or spindles all impact performance and scalability.

Application

The nature of the application directly impacts the scalability. Applications that require a lot of resources may scale up easily, but not scale out because of the demand on the already taxed resources. Applications that use a smaller amount of resources may not scale up well, but they may scale out very well.

Virtualization

CPU and memory resources are needed by the hypervisor when running a virtualized environment. This reduces the amount of resources available to the guests and can limit scaling.



5.2 Scaling Out

The following section discusses the results of scaling out various vCPU count guests.

5.2.1 One vCPU Guests

Figure 5.2.1.1: Scaling Out Database Time for Single vCPU Guests shows the results of the *DATABASE LOAD* times remain constant until the number of guests reach sixteen.

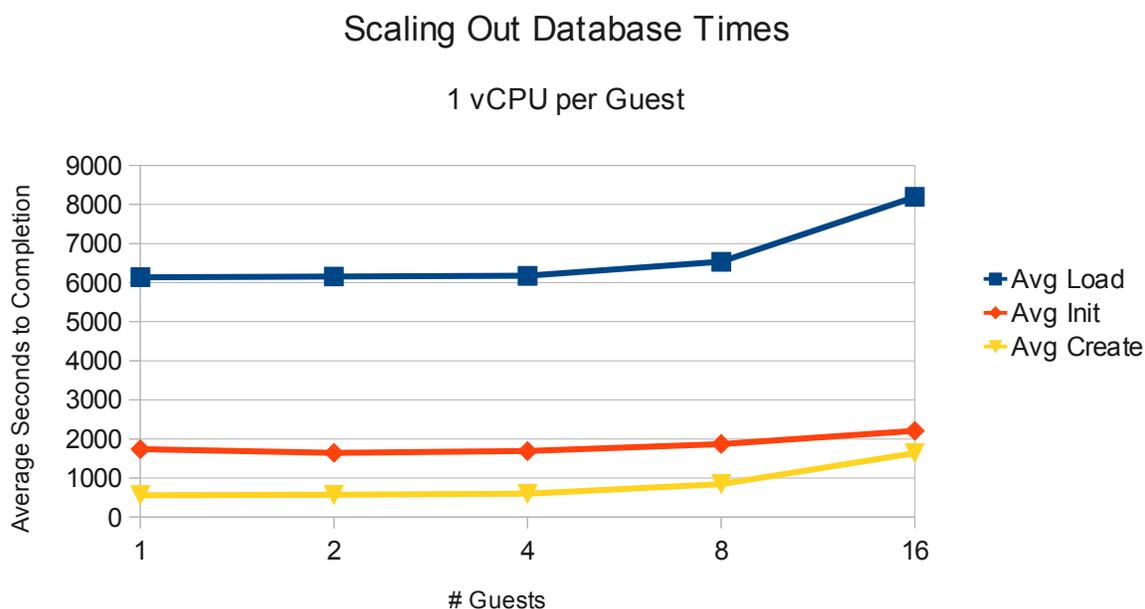


Figure 5.2.1.1: Scaling Out Database Time for Single vCPU Guests

The spike at sixteen guests is due to storage contention. Each guest has a dedicated set of storage spindles assigned for its sole use for the tests involving one, two, four, and eight guests. Once the number of guests reach sixteen, each guest shares storage spindles with one other guest. Refer back to **Figure 3.3.1: Storage Disk Configuration** and **Appendix A: Test Configurations** for the storage configuration.



Figure 5.2.1.2: Scaling Out Memory Throughput for Single vCPU Guests shows that memory throughput for each guest remains about the same as more guests are added.

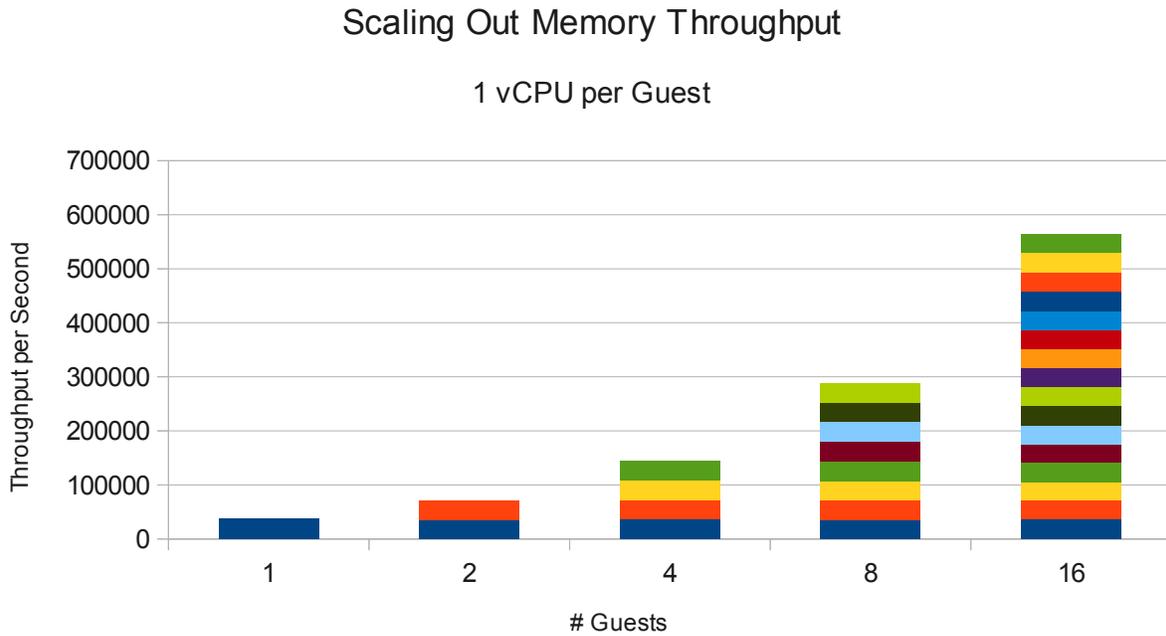


Figure 5.2.1.2: Scaling Out Memory Throughput for Single vCPU Guests

As the quantity of guests are doubled, the memory throughput doubles. This shows that a one vCPU guest scales extremely well.



5.2.2 Two vCPU Guests

Figure 5.2.2.1: Scaling Out Database Time for Two vCPU Guests shows that guests with two vCPUs scale better for the *DATABASE LOAD* test than one vCPU guests. The load time decreased significantly from that of a single vCPU guest since more vCPUs and disk spindles are presented to the guest. Thus allowing the **SLCS** tool to allocate more threads to import the database.

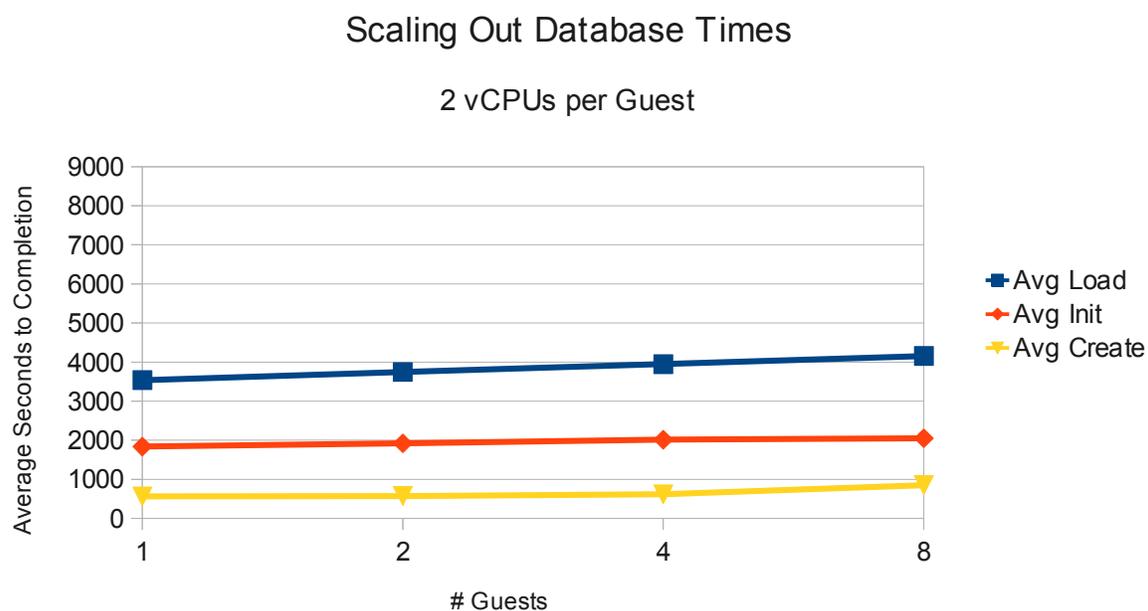


Figure 5.2.2.1: Scaling Out Database Time for Two vCPU Guests



Two vCPU guests scale very well with respect to memory throughput as shown by the following graph.

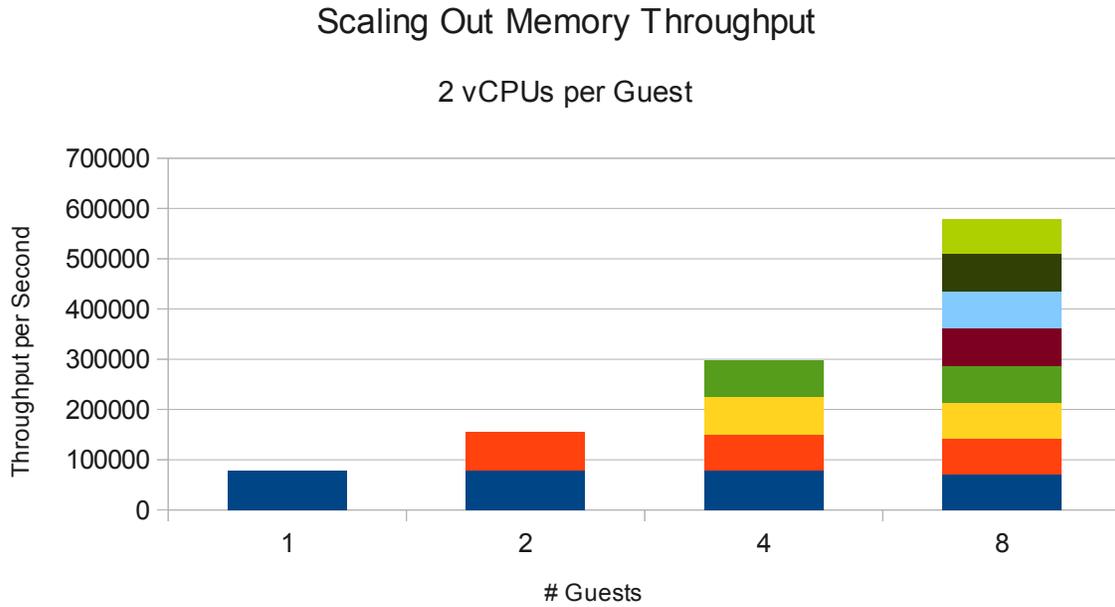


Figure 5.2.2.2: Scaling Out Memory Throughput for Two vCPU Guests



5.2.3 Four vCPU Guests

Figure 5.2.3.1: Scaling Out Database Time for Four vCPU Guests shows the load time for the database increases slightly with guests running four vCPUS. Although the time increases more sharply when running four guests, the graph still depicts a slight increase overall.

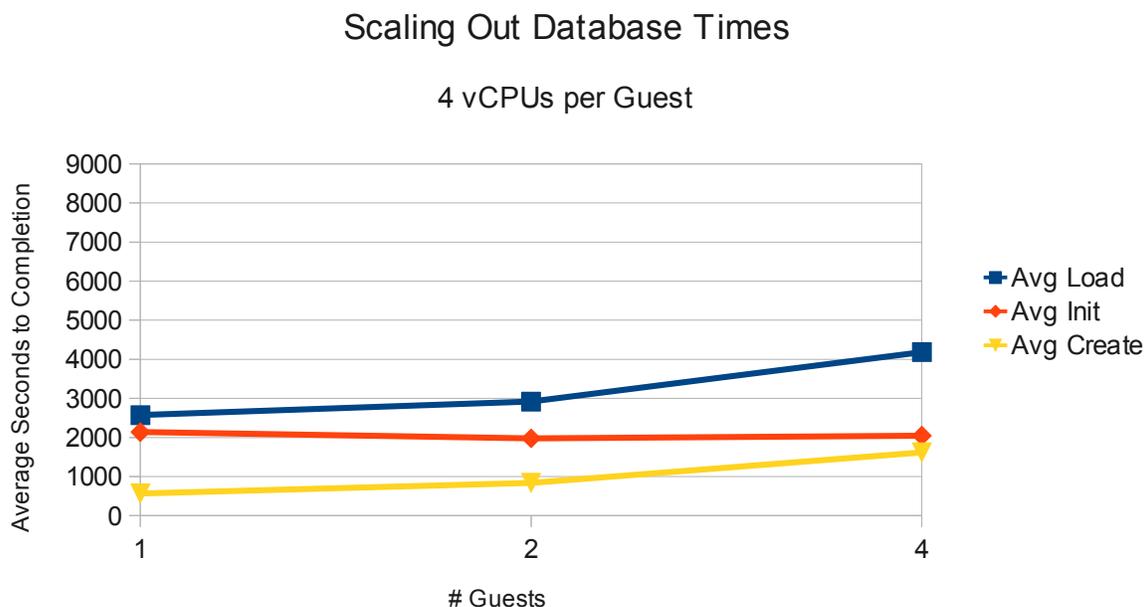


Figure 5.2.3.1: Scaling Out Database Time for Four vCPU Guests

The memory throughput scales very well for guests running four vCPUs. This is shown in the following graph.



Scaling Out Memory Throughput

4 vCPUs per Guest

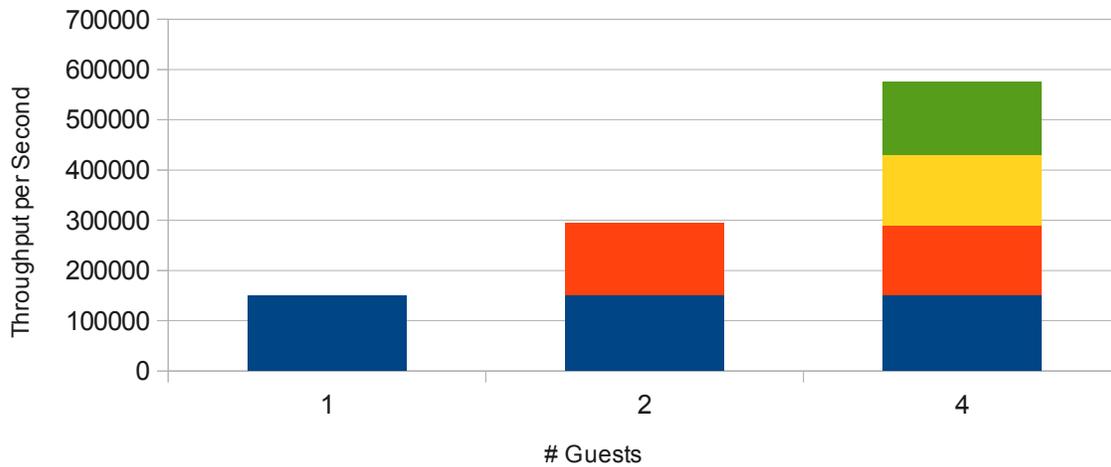


Figure 5.2.3.2: Scaling Out Memory Throughput for Four vCPU Guests

5.2.4 Eight vCPU Guests

The hardware used for the hypervisor server contained only sixteen processor cores. Because of this, the graphs for *DATABASE LOAD* and *MEMORY THROUGHPUT* only have two points for comparison.

Even with only two points for comparison, it appears the times associated with the database *AVG INIT* and *AVG CREATE* remain consistent as the *AVG LOAD* time increases only slightly.



Scaling Out Database Times

8 vCPUs per Guest

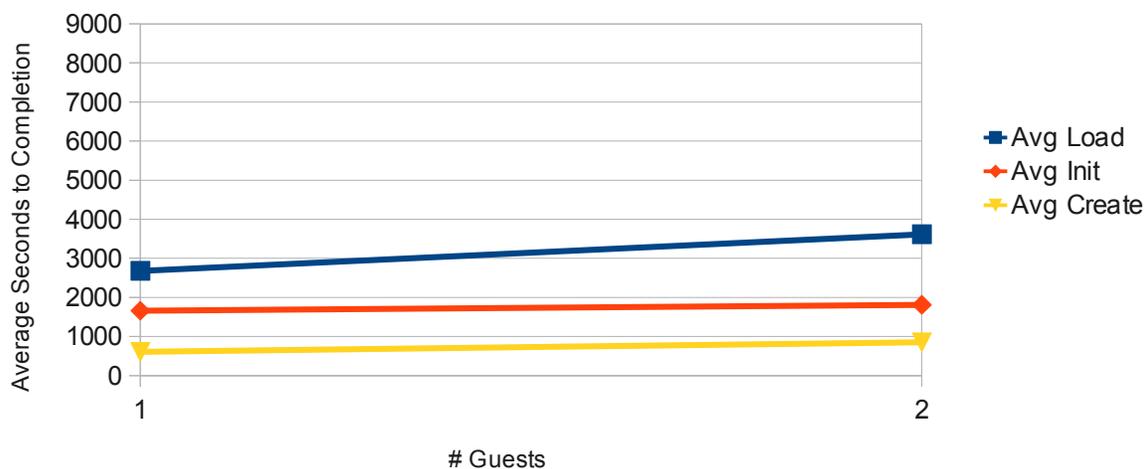


Figure 5.2.4.1: Scaling Out Database Time for Eight vCPU Guests

Memory throughput continues to scale very well on guests running eight vCPUs as well.

Scaling Out Memory Throughput

8 vCPUs per Guest

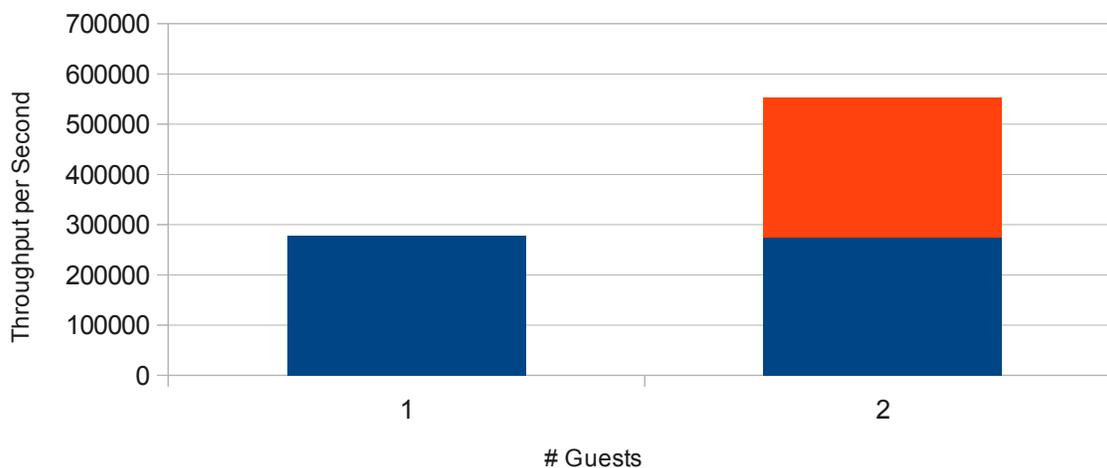


Figure 5.2.4.2: Scaling Out Memory Throughput for Eight vCPU Guests



5.3 Scaling Up

This section discusses the results of scaling up a single vCPU guest.

Figure 5.3.1: Scaling Up Database Times for a Single Guest depicts the results of the *DATABASE LOAD* test. The graph shows the *AVG INIT* and *AVG CREATE* times to remain constant throughout the testing. This is expected since neither of these two parts of the *DATABASE LOAD* test are impacted by vCPU count.

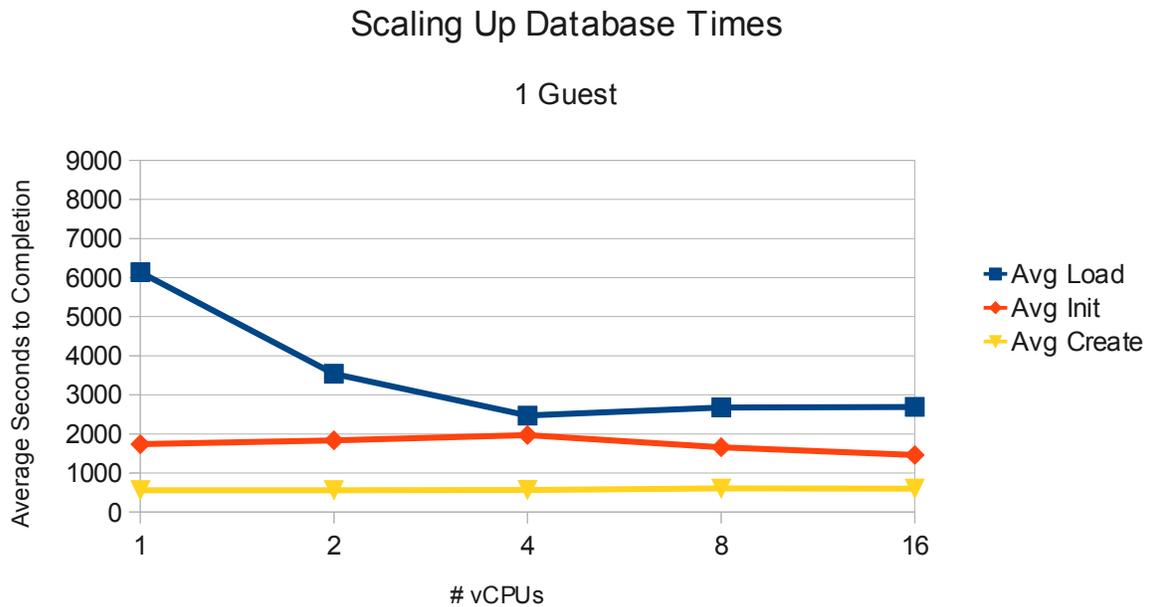


Figure 5.3.1: Scaling Up Database Times for a Single Guest

The *AVG LOAD* time decreases as the number of vCPUs presented to the guest increases. Since the load part of the test performs a multi-threaded import with a thread count based upon available vCPUs, this is expected.



Figure 5.3.2: Scaling Out Memory Throughput for a Single Guest depicts the results of the *MEMORY LOAD* test. The results show that the memory throughput increases linearly as the number of available vCPUs are increased.

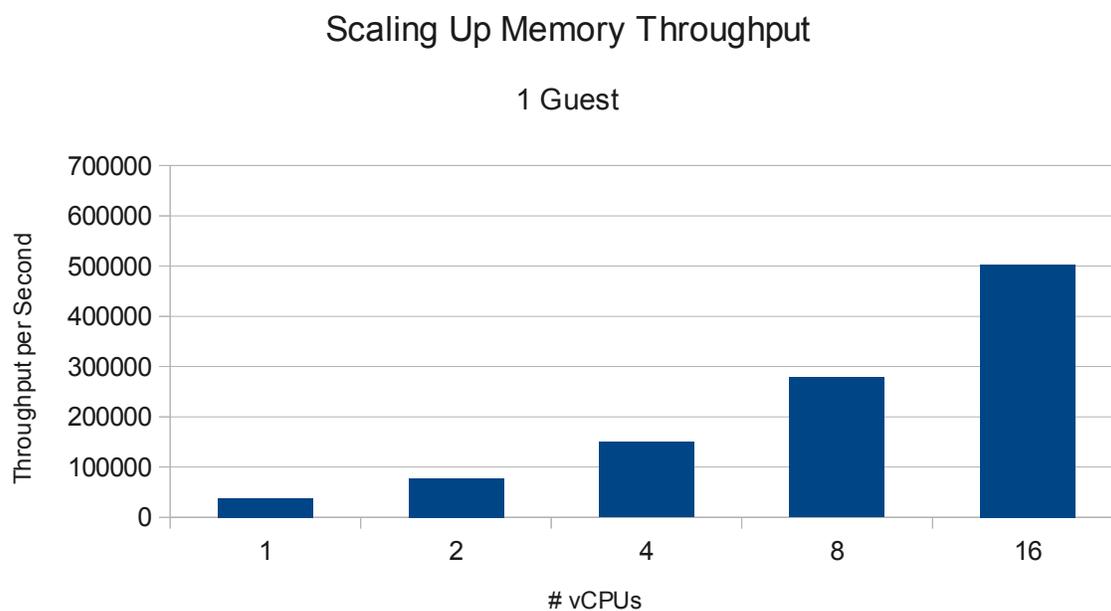


Figure 5.3.2: Scaling Out Memory Throughput for a Single Guest

The above graphs both indicate the application scales up very well.



5.4 Scaling Efficiency

The following graph compares the memory throughput of a sixteen core bare metal server to various test configuration where the total number of vCPUs allocated to guests equaled sixteen. Each test configuration represented in the graph is allocated the the same amount of memory as a whole with the exception of the bare metal system. The bare metal system has a little more memory available to it since there is no virtualization overhead.

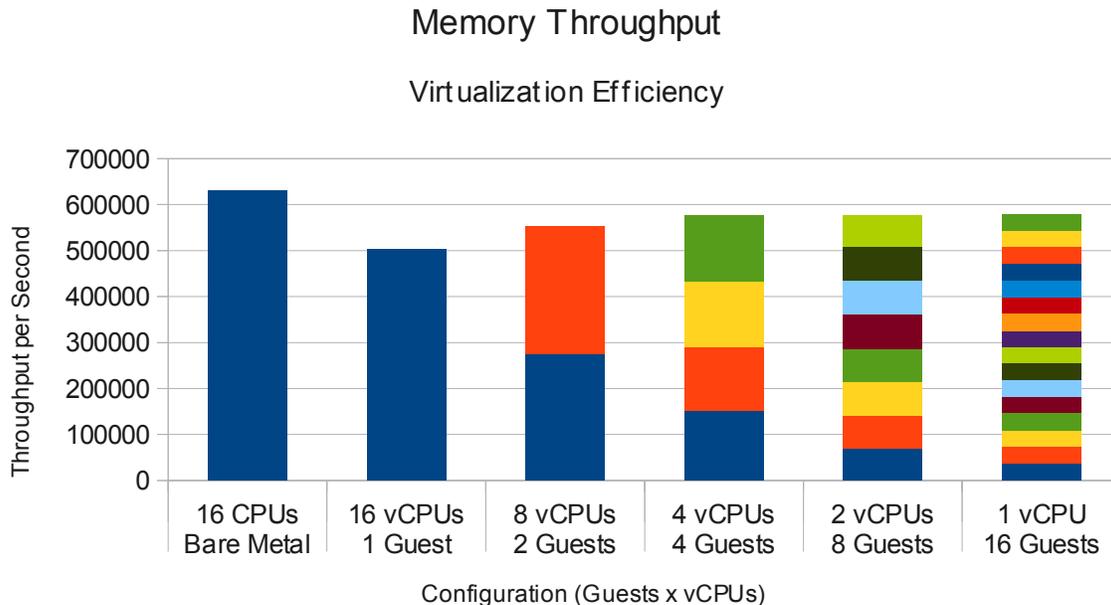


Figure 5.4.1: Virtualization Efficiency

There is a noticeable dip in the throughput when running a single sixteen vCPU guest. This is caused by *NUMA* zone traversals. When pinning all the vCPUs to physical CPUs, the sixteen vCPU guest was not able to be pinned to a single *NUMA* zone. The guests in all the other test configurations were able to be pinned to the same *NUMA* zone as the physical CPUs they were using. Without the *NUMA* pinning, the processors of the sixteen vCPU guest traversed across *NUMA* zones, causing a slower response time.

The *NUMA* zone traversal was seen using the **numastat** command. The command was executed before the testing and then after the testing. The difference of the **numa_miss** values were then calculated. Since only one guest was running in the hypervisor, it can be assumed that most (if not all) traversals were caused and therefore experience by the guest.



The following is the output of the **numastat** command before the sixteen vCPU test was ran.

```
# numastat
                node0          node1
numa_hit        158441478      145801899
numa_miss      1546                2978
numa_foreign    2978          1546
interleave_hit  32280          32275
local_node     158438357      145763223
other_node     4667           41654
```

The following is the output of the **numastat** command after the sixteen vCPU test was ran.

```
# numastat
                node0          node1
numa_hit        197852240      179430143
numa_miss      2381                1770537
numa_foreign    1770537        2381
interleave_hit  32280          32275
local_node     197849093      179390948
other_node     5528           1809732
```

As can be seen, over one and a half million traversals occurred for *NUMA* node 1.



6 Conclusion

This paper discussed the scaling of a *SAP* workload running in Red Hat Enterprise Linux 6 guests. These guests were running in a Red Hat Enterprise Virtualization 3.0 environment using a Red Hat Enterprise Linux 6 server for the hypervisor.

This paper demonstrated how a Red Hat Enterprise Virtualization solution can be the ideal environment for scaling *SAP* systems. Guest creation and resource allocation is quick and easy using the Red Hat Enterprise Manager interface. The REST API can be used to script the processes involved in guest creation and resource allocation and allows the ability to easily scale out the environment in an on-demand fashion.

Many variables affect the scalability of *SAP* deployments in a customer environment. These include storage hardware, server hardware, and the load placed on the system by users. However, careful planning and tuning of storage and server hardware can help increase the scalability of any specific *SAP* workload. The use of hooks to pin vCPUs to physical CPUs as well pinning the guest to specific *NUMA* zones can drastically increase the performance gained when scaling.

Overall, the data presented in this paper clearly shows that *SAP* workloads scale easily and extremely well when running in a Red Hat Enterprise Virtualization 3.0 environment. Memory and CPU throughput scale linearly until all available physical resources are exhausted.



Appendix A: Test Configurations

A.1 One vCPU

Guest	CPU Pinning	NUMA Pinning	Memory	LUN Configuration
vm01	0	0	7.68 GB	vol1
vm02	1	1	7.68 GB	vol2
vm03	2	0	7.68 GB	vol3
vm04	3	1	7.68 GB	vol4
vm05	4	0	7.68 GB	vol5
vm06	5	1	7.68 GB	vol6
vm07	6	0	7.68 GB	vol7
vm08	7	1	7.68 GB	vol8
vm09	8	0	7.68 GB	vol9
vm10	9	1	7.68 GB	vol10
vm11	10	0	7.68 GB	vol11
vm12	11	1	7.68 GB	vol12
vm13	12	0	7.68 GB	vol13
vm14	13	1	7.68 GB	vol14
vm15	14	0	7.68 GB	vol15
vm16	15	1	7.68 GB	vol16

Table 6.1: One vCPU Test Configuration



A.2 Two vCPUs

Guest	CPU Pinning	NUMA Pinning	Memory	LUN Configuration
vm01	0,8	0	15.36 GB	vol1, vol9
vm02	1,9	1	15.36 GB	vol2, vol10
vm03	2,10	0	15.36 GB	vol3, vol11
vm04	3,11	1	15.36 GB	vol4, vol12
vm05	4,12	0	15.36 GB	vol5, vol13
vm06	5,13	1	15.36 GB	vol6, vol14
vm07	6,14	0	15.36 GB	vol7, vol15
vm08	7,15	1	15.36 GB	vol8, vol16

Table 6.2: Two vCPU Test Configuration

A.3 Four vCPUs

Guest	CPU Pinning	NUMA Pinning	Memory	LUN Configuration
vm01	0, 4, 8, 12	0	30.72 GB	vol1, vol2, vol3, vol4
vm02	1, 5, 9, 13	1	30.72 GB	vol5, vol6, vol7, vol8
vm03	2, 6, 10, 14	0	30.72 GB	vol9, vol10, vol11, vol12
vm04	3, 7, 11, 15	1	30.72 GB	vol13, vol14, vol15, vol16

Table 6.3: Four vCPU Test Configuration



A.4 Eight vCPUs

Guest	CPU Pinning	NUMA Pinning	Memory	LUN Configuration
vm01	0, 2, 4, 6, 8, 10, 12, 14	0	61.44 GB	vol1, vol2, vol3, vol4, vol5, vol6, vol7, vol8
vm02	1, 3, 5, 7, 9, 11, 13, 15	1	61.44 GB	vol1, vol2, vol3, vol4, vol5, vol6, vol7, vol8

Table 6.4: Eight vCPU Test Configuration

A.5 Sixteen vCPUS

Guest	CPU Pinning	NUMA Pinning	Memory	LUN Configuration
vm01	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	N/A	122.88 GB	vol1, vol3, vol5, vol7, vol9, vol11, vol13, vol15

Table 6.5: Sixteen vCPU Test Configuration

A.6 Bare Metal

Host	CPU Pinning	NUMA Pinning	Memory	LUN Configuration
Reinstalled Hypervisor	N/A	N/A	128 GB	vol1, vol3, vol5, vol7

Table 6.6: Bare Metal Test Configuration



Appendix B: Scripts

B.1 run_test.sh

```
#!/bin/bash

vm=$1
cpu=$2

if [ -z "${vm}" -o -z "${cpu}" ]
then
    echo "Usage: $0 <# of VMs during test> <# of CPUs per VM>"
    exit
fi

export DATADEVS="100GB /sapdb"
export EXPORT="ERP2005SR1.tar"

if [ ! -d /sapdb ]
then
    mkdir -p /sapdb
    chmod 777 /sapdb
fi

hostname $( hostname -s )

sec_start=$( date +%s )

clear

rpm -q --quiet compat-libstdc++-33
RC=$?

if [ ${RC} -ne 0 ]
then
    echo
    echo "Package compat-libstdc++-33 not installed"
    echo "Exiting..."
    exit 1
else
    echo
    echo "Required packages are installed."
    echo
fi

echo -e "--- \c"
lscpu | grep "^CPU(s)"

echo -e "--- \c"
cat /proc/meminfo | grep MemTotal

echo -e "--- \c"
mount | grep myvg-rootvol
```



```
echo "Press <Enter> to continue"
read

acpu=$( lscpu | grep "^CPU(s)" | awk '{print $2}' )
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Running Tests for ${vm} VMs with $
{cpu}(${acpu}) cpus."

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Install Test"
./install.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Install Test"

sleep 30

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 1"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 1"

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 2"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 2"

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 3"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 3"

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 4"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 4"

sec_end=$( date "+%s" )

sec_total=$(( sec_end - sec_start ))

minutes=$(( sec_total / 60 ))
seconds=$(( sec_total % 60 ))

hours=$(( minutes / 60 ))
minutes=$(( minutes % 60 ))

printf " --- %s Total Run Time %02d:%02d:%02d\n" "$(date '+%m/%d/%Y %H:%M:%S
%s')" ${hours} ${minutes} ${seconds}
```



B.2 Kickstart %post section

```
# Disable unnecessary services
#
needed=":iptables:ip6tables:lvm2-monitor:network:sshd:rsyslog:sysstat:udev-
post:"

for service in $( chkconfig --list | grep ":on" | awk '{print $1}' )
do

    echo ${needed} | grep -q ":$service:"
    RC=$?

    if [ ${RC} -ne 0 ]
    then
        echo "Disabling service ${service}"
        chkconfig --level 12345 ${service} off
    fi

done

# Tune the Ext filesystem.
#
echo "Tuning filesystem"
tune2fs -o journal_data_writeback /dev/mapper/myvg-rootvol

# Modify fstab for tuned file system
#
sed -i -e 's:\(^/dev/mapper/myvg-rootvol .*\)defaults\
(.*$):\1noatime,data=writeback,barrier=0\2:' /etc/fstab

# Install the needed packages
#
echo "Installing needed packages"
yum -y install compat-libstdc++-33

# Get and unpack the SLCS tool and test script.
#
echo "Extracting testing application"
PDIR=$( pwd )
cd /
wget -O /root/slcs.tgz http://share/pub/kits/sap/
slcs2.3.tar.gz

tar zxvf /root/slcs.tgz
cd /slcs*

wget -O ./run_test.sh http://share/pub/kits/sap/r un_test.sh
chmod +x ./run_test.sh

cd ${PDIR}
```



B.3 run_test.sh

```
#!/bin/bash

vm=$1
cpu=$2

if [ -z "${vm}" -o -z "${cpu}" ]
then
    echo "Usage: $0 <# of VMs during test> <# of CPUs per VM>"
    exit
fi

export DATADEVS="100GB /sapdb"
export EXPORT="ERP2005SR1.tar"

if [ ! -d /sapdb ]
then
    mkdir -p /sapdb
    chmod 777 /sapdb
fi

hostname $( hostname -s )

sec_start=$( date +%s )

clear

rpm -q --quiet compat-libstdc++-33
RC=$?

if [ ${RC} -ne 0 ]
then
    echo
    echo "Package compat-libstdc++-33 not installed"
    echo "Exiting..."
    exit 1
else
    echo
    echo "Required packages are installed."
    echo
fi

echo -e "--- \c"
lscpu | grep "^CPU(s)"

echo -e "--- \c"
cat /proc/meminfo | grep MemTotal

echo -e "--- \c"
mount | grep myvg-rootvol

echo "Press <Enter> to continue"
read
```



```
acpu=$( lscpu | grep "^CPU(s)" | awk '{print $2}' )
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Running Tests for ${vm} VMs with $
{cpu}(${acpu}) cpus."

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Install Test"
./install.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Install Test"

sleep 30

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 1"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 1"

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 2"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 2"

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 3"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 3"

echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Starting SAP Memory Test -- Run 4"
./memory_load.sh
echo "--- $(date "+%m/%d/%Y %H:%M:%S %s") Finished SAP Memory Test -- Run 4"

sec_end=$( date "+%s" )

sec_total=$(( sec_end - sec_start ))

minutes=$(( sec_total / 60 ))
seconds=$(( sec_total % 60 ))

hours=$(( minutes / 60 ))
minutes=$(( minutes % 60 ))

printf " --- %s Total Run Time %02d:%02d:%02d\n" "$(date '+%m/%d/%Y %H:%M:%S
%s')" ${hours} ${minutes} ${seconds}
```



Appendix C: Hooks

Hooks are a mechanism that allows custom actions to be performed based upon VDSM events. There are three basic steps to configuring hooks.

- Create the hook script on Red Hat Enterprise Linux Hypervisor.
- Configure the Red Hat Enterprise Virtualization Manager for the new hook.
- Apply the hook to the guest.

Each of the steps is discussed in detail below.

C.1 Hypervisor Configuration

The scripts to execute when a VDSM event occurs must be placed in the directory representing the event under the `/usr/libexec/vdsm/hooks` directory on the hypervisor.

The script to pin vCPUs to physical CPUs is placed in the `/usr/libexec/vdsm/hooks/before_vm_start` directory since the pinning of the vCPUs must be done prior to starting the guest.

The file `/usr/libexec/vdsm/hooks/before_vm_start/50_pincpu` is created with the following contents.

```
#!/usr/bin/python

import os
import sys
import hooking
import traceback

'''
pincpu usages
=====
pincpu=0 (use the first cpu)
pincpu=1-4 (use cpus 1-4)
pincpu=^3 (dont use cpu 3)
pincpu=1-4,6 (or all together)
'''

if os.environ.has_key('pincpu'):
    try:
        domxml = hooking.read_domxml()

        vcpu = domxml.getElementsByTagName('vcpu')[0]

        if not vcpu.hasAttribute('cpuset'):
            sys.stderr.write('pincpu: pinning cpu to: %s\n' %
os.environ['pincpu'])
            vcpu.setAttribute('cpuset', os.environ['pincpu'])
            hooking.write_domxml(domxml)
    else:
```



```
        sys.stderr.write('pincpu: cpuset attribute is present in vcpu,
doing nothing\n')
    except:
        sys.stderr.write('pincpu: [unexpected error]: %s\n' %
traceback.format_exc())
        sys.exit(2)
```

The script to pin guest to NUMA zones is placed in the `/usr/libexec/vdsm/hooks/before_vm_start` directory since the NUMA pinning must be done prior to starting the guest.

The file `/usr/libexec/vdsm/hooks/before_vm_start/50_numa` is created with the following contents.

```
#!/usr/bin/python

import os
import sys
import hooking
import traceback

'''
numa hook
=====
add numa support for domain xml:

<numatune>
  <memory mode="strict" nodeset="1-4,^3" />
</numatune>

memory=interleave|strict|preferred

numaset="1" (use one NUMA node)
numaset="1-4" (use 1-4 NUMA nodes)
numaset="^3" (don't use NUMA node 3)
numaset="1-4,^3,6" (or combinations)

syntax:
  numa=strict:1-4
'''

if os.environ.has_key('numa'):
    try:
        mode, nodeset = os.environ['numa'].split(':')

        domxml = hooking.read_domxml()

        domain = domxml.getElementsByTagName('domain')[0]
        numas = domxml.getElementsByTagName('numatune')

        if not len(numas) > 0:
            numatune = domxml.createElement('numatune')
            domain.appendChild(numatune)
```



```
memory = domxml.createElement('memory')
memory.setAttribute('mode', mode)
memory.setAttribute('nodeset', nodeset)
numatune.appendChild(memory)

hooking.write_domxml(domxml)
else:
    sys.stderr.write('numa: numa already exists in domain xml')
    sys.exit(2)
except:
    sys.stderr.write('numa: [unexpected error]: %s\n' %
traceback.format_exc())
    sys.exit(2)
```

The scripts should be owned by root, be a member of the root group, and have the appropriate permissions to execute.

```
# cd /usr/libexec/vdsm/hooks/before_vm_start

# ls -l
total 12
-rw-r--r--. 1 root root 1251 Apr 30 19:52 50_numa
-rw-r--r--. 1 root root 817 Apr 30 19:51 50_pincpu

# chmod 755 50_numa 50_pincpu

# ls -l
total 12
-rwxr-xr-x. 1 root root 1251 Apr 30 19:52 50_numa*
-rwxr-xr-x. 1 root root 817 Apr 30 19:51 50_pincpu*
```

Care should be taken when using hooks. Guests can crash and data may be lost if a VDSM hook script contains a bug.

C.2 Red Hat Enterprise Virtualization Manager Configuration

The Red Hat Enterprise Virtualization Manager must be made aware of the newly available hooks and the options they take. This is done using the **rhevms-config** command and passing the appropriate information. See the *Extending VDSM with Hooks* section of the *Red Hat Enterprise Virtualization 3.0 Administration Guide*⁴ for more information.

Check to see what hooks are enabled using the **rhevms-config** command with the **-g** option.

```
# rhevms-config -g UserDefinedVMPProperties
```

⁴ http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/Administration_Guide/VDSM_Hooks.html



```
UserDefinedVMProperties: version: 3.0  
UserDefinedVMProperties: version: 2.2
```

The above shows that there are no hooks defined in the environment.

The `-s` option allows the hooks to be defined. All hooks including those already defined must be listed as an option on the command line. If any are excluded, they will be removed and not available to the system. Each hook definition is separated using a semi-colon. The `--cver` option must also be specified.

```
# rhvm-config -s UserDefinedVMProperties='pincpu=^[\\^]?\\d+(-\\d+)?(, [\\^]?\\d+  
(-\\d+)?)*$;numaset=(interleave|strict|preferred):[\\^]?\\d+(-\\d+)?(, [\\^]?\\d+  
(-\\d+)?)*$' --cver=3.0
```

The system is now configured to use the new hooks. The Red Hat Enterprise Virtualization Service must be restarted before the hooks can be used.

```
# rhvm-config -g UserDefinedVMProperties  
UserDefinedVMProperties: version: 2.2  
UserDefinedVMProperties: pincpu=^[\\^]?\\d+(-\\d+)?(, [\\^]?\\d+  
(-\\d+)?)*$;numaset=(interleave|strict|preferred):[\\^]?\\d+(-\\d+)?(, [\\^]?\\d+  
(-\\d+)?)*$ version: 3.0  
  
# service jbossas restart  
Stopping jbossas: [ OK ]  
Starting jbossas: [ OK ]
```



C.3 Guest Configuration

Hooks for the guests can be configured using either the Red Hat Enterprise Virtualization Web Portal or using the REST API.

The **Custom Properties** tab of the **Edit Virtual Machine** dialog window allows the configuration of hooks using the portal. This is a good method to test newly implemented hooks since the **Custom Properties** entry box changes color if there is an incorrect or unknown entry specified. Placing the mouse over the box when it changes color results in a pop-up dialog that displays all known hooks.

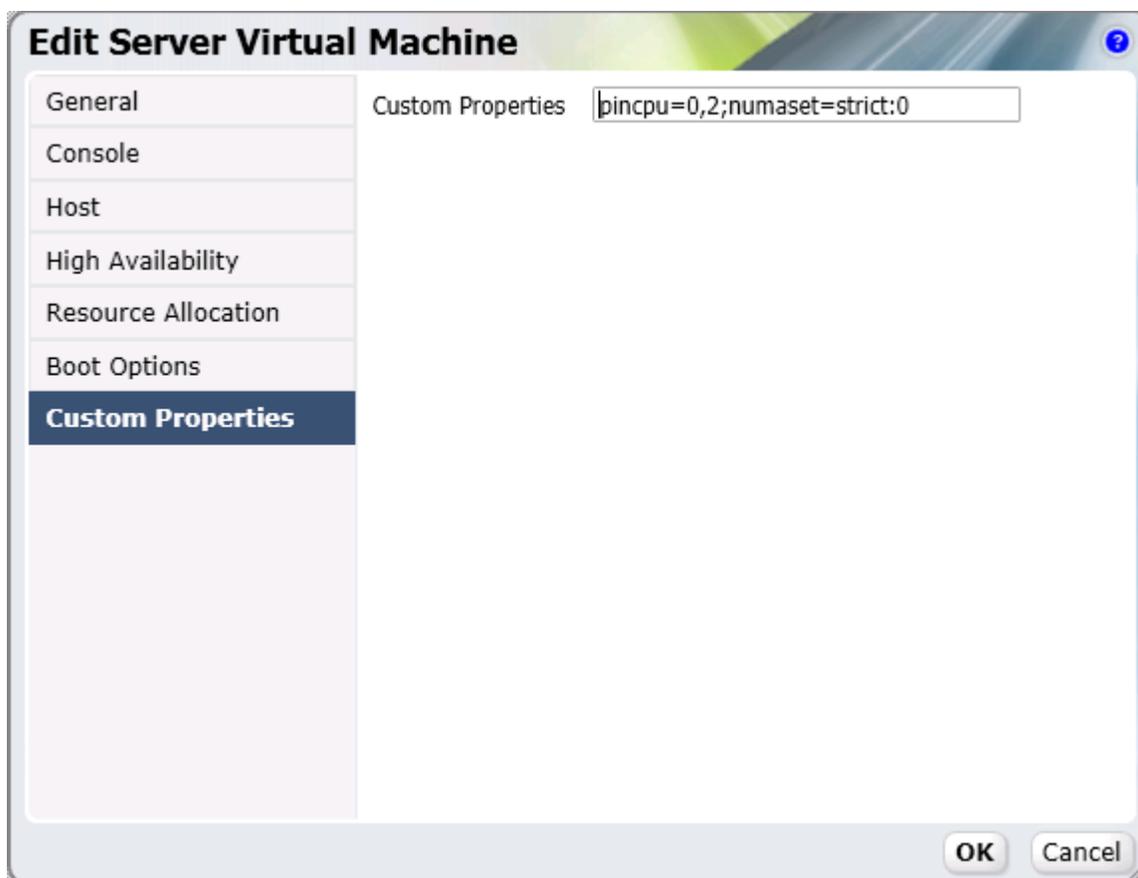


Figure 6.1: Custom Properties

The REST API may also be used for configuring hooks on a guest. This section demonstrates configuring hooks using the REST API, but it does not discuss the configuration and use of the REST API. See the *Red Hat Enterprise Virtualization 3.0 REST API Guide*⁵ for detailed information concerning the configuration and use of the REST API.

The certificate is needed for the Red Hat Enterprise Virtualization Manager. It is downloaded using the **curl** command.

⁵ http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/REST_API_Guide/index.html



```
# curl -o rhvm.cer http://ssap-rhev:8080/ca.crt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             0         0             0           0             0             0             0
100  3634 100  3634  0      0    8332           0  --:--:--  --:--:--  --:--:--  23907
```

Accessing the REST API requires authentication. The authentication is passed to the **curl** command using the **--user** option. The parameter to the option is specified in the form of **USER@DOMAIN:PASSWORD**.

The UUID of the guest is used when accessing the guest configuration using the REST API. The UUID of the guests is obtained using the REST API.

```
# PASSWORD="admin@internal:[PASSWORD]"

# curl --silent --cacert rhvm.cer \
  --header "Content-Type: application/xml" \
  --user "${PASSWORD}" \
  --request "GET" \
  https://ssap-rhev:8443/api/vms \
  | xpath "/vms/vm/@id|/vms/vm/name"

Found 2 nodes:
-- NODE --
id="7f70e41e-c041-49c6-868f-16c70e30410f"-- NODE --
<name>vm01</name>
```

Once the UUID is known, the hooks are specified using the following XML code.

```
<vm>
  <custom_properties>
    <custom_property value="0,2" name="pincpu"/>
    <custom_property value="strict:0" name="numaset"/>
  </custom_properties>
</vm>
```

The curl command is used to submit the appropriate XML code. Once the XML code is submitted, an XML response indicating success or failure is returned.

```
$ curl --silent --cacert rhvm.cer \
  --header "Content-Type: application/xml" \
  --user "${PASSWORD}" \
  --request "PUT" \
  --data '<vm><custom_properties><custom_property value="0,2"
name="pincpu"/><custom_property value="strict:0"
name="numaset"/></custom_properties></vm>' \
  https://ssap-rhev:8443/api/vms/7f70e41e-c041-49c6-868f-16c70e30410f

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vm id="7f70e41e-c041-49c6-868f-16c70e30410f" href="/api/vms/7f70e41e-c041-
49c6-868f-16c70e30410f">
  <name>vm01</name>

[ ... Content Removed for Brevity ... ]
```



```
<custom_properties>
  <custom_property value="0,2" name="pincpu"/>
  <custom_property value="strict:0" name="numaset"/>
</custom_properties>
<placement_policy>
  <affinity>migratable</affinity>
</placement_policy>
<memory_policy>
  <guaranteed>536870912</guaranteed>
</memory_policy>
<usb>
  <enabled>>true</enabled>
</usb>
</vm>
```



Appendix D: Contributors

Contributer	Title	Contribution
David Dumas	Senior Software Engineer	Content, Review
Bill Gray	Principal Software Engineer	Content, Review
Brett Thurber	Principal Software Engineer	Content, Review



Appendix E: Revision History

Revision 1.0

Monday May 3, 2012

John Herr

Initial Release