For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Red Hat
Training and Certification

Student Workbook (ROLE)

Red Hat Enterprise Linux 8.0 RH442

Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud

Edition 1

RH442-RHEL8.0-en-1-20200501

Copyright ©2019 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

RH442-RHEL8.0-en-1-20200501

Copyright ©2019 Red Hat, Inc.

Red Hat Enterprise Linux 8.0 RH442
Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud
Edition 1 20200501
Publication date 20190828

Authors: Adolfo Vazquez Rodriguez, Adrian Andrade, Herve Quatremain, Snehangshu Karmakar, Victor Costea
Editor: Philip Sweany, Nicole Muller, Jeff Tyson

Copyright © 2019 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are Copyright © 2019 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, Hibernate, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

The OpenStack® word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation in the United States and other countries, and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Douglas Shakshober, AJ Costa, Brian Atkisson, William Clark, Zoltan Molnar, Jerry Smith, and Alex Wang

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Document Conventions **ix**

Introduction **xi**
Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud ... xi
Orientation to the Classroom Environment ... xii
Internationalization ... xv

1. Introducing Performance Tuning **1**
Defining Performance Tuning Concepts ... 2
Quiz: Defining Performance Tuning Concepts ... 6
Discussing Performance Tuning Methodology ... 8
Quiz: Discussing Performance Tuning Methodology ... 12
Describing Computer Units of Measure ... 16
Guided Exercise: Converting Computer Units of Measure ... 19
Summary ... 21

2. Selecting Performance Monitoring Tools **23**
Identifying System Monitoring Tools ... 24
Guided Exercise: Identifying System Monitoring Tools ... 30
Describing the Sysstat Package Utilities ... 34
Guided Exercise: Viewing the Sysstat Package Utilities ... 43
Collecting Performance Data with Performance Co-Pilot ... 51
Guided Exercise: Collecting Performance Data with Performance Co-pilot ... 59
Lab: Selecting Performance Monitoring Tools ... 66
Summary ... 74

3. Viewing Hardware Resources **75**
Displaying Physical Resources ... 76
Guided Exercise: Displaying Physical Resources ... 93
Displaying Resources in Virtual and Cloud Instances ... 98
Guided Exercise: Displaying Resources in Virtual and Cloud Instances ... 106
Quiz: Viewing Hardware Resources ... 112
Summary ... 114

4. Configuring Kernel Tunables and Tuned Profiles **115**
Configuring Kernel Tunables ... 116
Guided Exercise: Configuring Kernel Tunables ... 123
Selecting a Tuned Profile ... 126
Guided Exercise: Selecting a Tuned Profile ... 134
Customizing Tuned Profiles ... 138
Guided Exercise: Customizing Tuned Profiles ... 143
Lab: Configuring Kernel Tunables and Tuned Profiles ... 145
Summary ... 150

5. Managing Resource Limits with Control Groups **151**
Managing Resource Limits ... 152
Guided Exercise: Managing Resource Limits ... 156
Customizing Control Groups ... 159
Guided Exercise: Customizing Control Groups ... 164
Lab: Managing Resource Limits with Control Groups ... 167
Summary ... 172

6. Analyzing Performance Using System Tracing Tools **173**
Profiling with Performance Counters ... 174
Guided Exercise: Profiling with Performance Counters ... 182
Tracing System and Library Calls ... 191
Guided Exercise: Tracing System and Library Calls ... 194
Gathering Diagnostic Data Using SystemTap ... 196

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise: Gathering Diagnostic Data Using SystemTap 203
Tracing System Events with eBPF Tools 208
Guided Exercise: Tracing System Events with eBPF Tools 215
Lab: Analyzing Performance Using Tracing Tools 220
Summary 227

7. Tuning CPU Utilization **229**
Configuring CPU Scheduling Policies and Tunables 230
Guided Exercise: Configuring CPU Scheduling Policies and Tunables 242
Configuring CPU Affinity and Isolation 247
Guided Exercise: Configuring CPU Affinity and Isolation 253
Profiling Cache Use 257
Guided Exercise: Profiling Cache Use 265
Lab: Tuning CPU Utilization 268
Summary 274

8. Tuning Memory Utilization **275**
Configuring Memory Architecture 276
Guided Exercise: Configuring Memory Architecture 284
Configuring Memory Paging and Reclamation 289
Guided Exercise: Configuring Memory Paging and Reclamation 297
Configuring NUMA Topology 300
Guided Exercise: Configuring NUMA Topology 307
Managing Memory Overcommit 311
Guided Exercise: Managing Memory Overcommit 313
Lab: Tuning Memory Utilization 316
Summary 323

9. Tuning Storage Device I/O **325**
Evaluating I/O Patterns and Scheduling Algorithms 326
Guided Exercise: Evaluating I/O Patterns and Scheduling Algorithms 330
Reviewing RAID Fundamentals 333
Guided Exercise: Reviewing RAID Fundamentals 337
Selecting I/O Analysis Tools 340
Guided Exercise: Selecting I/O Analysis Tools 343
Lab: Tuning Storage Device I/O 346
Summary 352

10. Tuning File System Utilization **353**
Managing File System Attributes 354
Guided Exercise: Managing File System Attributes 361
Managing File System Journaling 364
Guided Exercise: Managing File System Journaling 367
Lab: Tuning File System Utilization 370
Summary 376

11. Tuning Network Utilization **377**
Configuring for Network Latency and Throughput 378
Guided Exercise: Configuring for Network Latency and Throughput 383
Configuring Network Driver Parameters 386
Guided Exercise: Configuring the Network Link Layer 395
Lab: Tuning Network Utilization 404
Summary 411

12. Performing Tuning Hosts in Virtualization Environments **413**
Tuning Hosts in Virtualization Environments 414
Guided Exercise: Tuning Hosts in Virtualization Environments 425
Monitoring Performance Metrics in a Virtualized Environment 432

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise: Monitoring Performance Metrics in a Virtualized Environment 439
Lab: Performing Tuning in Virtualization Environments 445
Summary 454

13. Comprehensive Review **455**
Comprehensive Review 456
Lab: Tuning for a Computation-heavy Application 459
Lab: Tuning for a Storage Intensive Application 465
Lab: Tuning for a Large Memory Implementation 470

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Document Conventions

References
"References" describe where to find external documentation relevant to a subject.

Note
"Notes" are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important
"Important" boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss, but may cause irritation and frustration.

Warning
"Warnings" should not be ignored. Ignoring warnings will most likely cause data loss.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Introduction

Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud

Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud (RH442) is designed to teach senior Linux system administrators the methodology of performance tuning. This course discusses system architecture, with emphasis on understanding the implications of system architecture on system performance; performance adjustments, open source benchmarking utilities, networking performance, and tuning configurations for specific application loads, are discussed.

Course Objectives

- Build skills needed to be a successful senior Red Hat Enterprise Linux system administrator, responsible for tuning and maintaining settings to support specific machine workloads.
- Prepare for attendance in a Certificate of Expertise in Performance Tuning exam.

Audience

- Senior Linux system administrators responsible for maximizing resource utilization through performance tuning.
- A Red Hat Certified Engineer (RHCE) who is interested in earning a Red Hat Certification of Expertise, or a Red Hat Certified Architect(RHCA).

Prerequisites

- The prerequisites for this class are Red Hat Certified Engineer (RHCE) certification or equivalent experience.

Orientation to the Classroom Environment

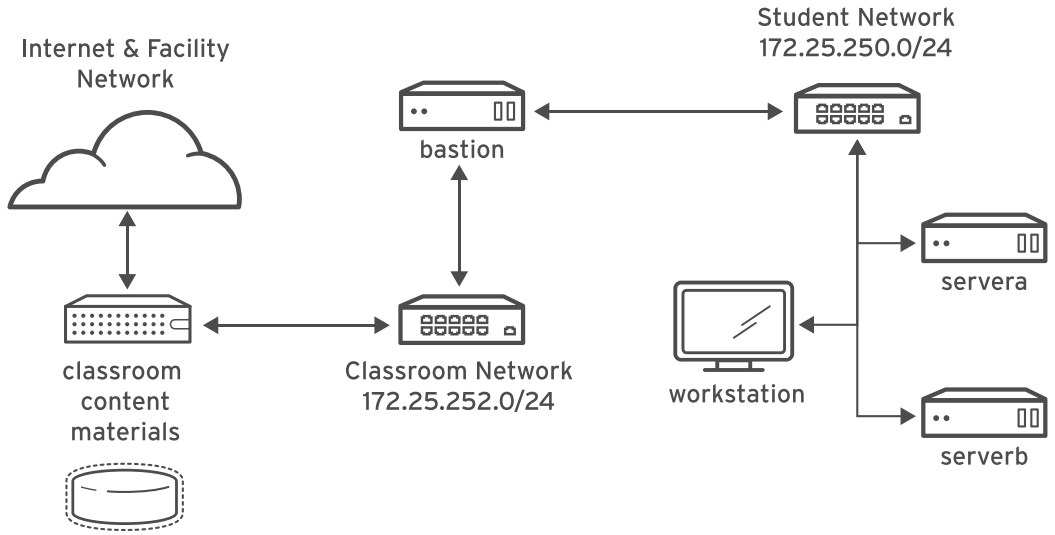


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is the **`workstation`** machine. Two other machines are also used by students for these activities: **`servera`** and **`serverb`**. All three of these systems are in the **`lab.example.com`** DNS domain.

All student computer systems have a standard user account, **`student`**, which has the password **`student`**. The **`root`** password on all student systems is **`redhat`**.

Classroom Machines

Machine name	IP address	Role
bastion.lab.example.com	172.25.250.254	Virtual router that connects the student network to the classroom server. (Must always remain running.)
workstation.lab.example.com	172.25.250.9	Graphical workstation that is used as a system administration desktop.
servera.lab.example.com	172.25.250.10	A server that is used as a tuning target.
serverb.lab.example.com	172.25.250.11	A server that is used as another tuning target.

The primary function of **`bastion`** is to act as a router between the student systems network and the classroom network. If **`bastion`** is down, then student machines can only access systems on the individual student network.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Several systems in the classroom provide supporting services. Two server URLs, **`content.example.com`** and **`materials.example.com`**, provide software and lab materials for the hands-on activities, as described in the activity's instructions. These URLs are aliases of the **`classroom.example.com`** server. Both the **`classroom`** and **`bastion`** systems must remain running for the course environment to work properly.

Note

When logging on to **`servera`** or **`serverb`**, you might see a message concerning the activation of the **`cockpit`** socket for the **`Web Console`**. The message can be ignored.

```
[student@workstation ~]$ ssh student@serverb
Warning: Permanently added 'serverb,172.25.250.11' (ECDSA) to the list of
 known hosts.
Activate the web console with: systemctl enable --now cockpit.socket
[student@serverb ~]$
```

Controlling Your Systems

Students are assigned remote computers in a Red Hat Online Learning classroom. They are accessed through a web application hosted at rol.redhat.com [http://rol.redhat.com]. Students should log in to this site using their Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through a web page. The state of each virtual machine in the classroom is displayed on the page under the **Online Lab** tab.

Machine States

Virtual Machine State	Description
STARTING	The virtual machine is in the process of booting.
STARTED	The virtual machine is running and available (or, when booting, soon will be).
STOPPING	The virtual machine is in the process of shutting down.
STOPPED	The virtual machine is completely shut down. Upon starting, the virtual machine boots into the same state as when it was shut down (the disk will have been preserved).
PUBLISHING	The initial creation of the virtual machine is being performed.
WAITING_TO_START	The virtual machine is waiting for other virtual machines to start.

Depending on the state of a machine, a selection of the following actions is available.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Classroom/Machine Actions

Button or Action	Description
PROVISION LAB	Create the ROL classroom. Creates all of the virtual machines needed for the classroom and starts them. Can take several minutes to complete.
DELETE LAB	Delete the ROL classroom. Destroys all virtual machines in the classroom. **Caution: Any work generated on the disks is lost.**
START LAB	Start all virtual machines in the classroom.
SHUTDOWN LAB	Stop all virtual machines in the classroom.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. Students can log in directly to the virtual machine and run commands. In most cases, students should log in to the `workstation` virtual machine and use `ssh` to connect to the other virtual machines.
ACTION → **Start**	Start (power on) the virtual machine.
ACTION → **Shutdown**	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION → **Power Off**	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION → **Reset**	Forcefully shut down the virtual machine and reset the disk to its initial state. **Caution: Any work generated on the disk is lost.**

At the start of an exercise, if instructed to reset a single virtual machine node, click **ACTION** → **Reset** for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION** → **Reset**

If you want to return the classroom environment to its original state at the start of the course, you can click **DELETE LAB** to remove the entire classroom environment. After the lab has been deleted, you can click **PROVISION LAB** to provision a new set of classroom systems.

> **Warning**
> The **DELETE LAB** operation cannot be undone. Any work you have completed in the classroom environment up to that point will be lost.

The Autostop Timer

The Red Hat Online Learning enrollment entitles students to a certain amount of computer time. To help conserve allotted computer time, the ROL classroom has an associated countdown timer, which shuts down the classroom environment when the timer expires.

To adjust the timer, click **MODIFY** to display the **New Autostop Time** dialog box. Set the number of hours until the classroom should automatically stop. Note that there is a maximum time of ten hours. Click **ADJUST TIME** to apply this change to the timer settings.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Internationalization

Per-user Language Selection

Your users might prefer to use a different language for their desktop environment than the system-wide default. They might also want to use a different keyboard layout or input method for their account.

Language Settings

In the GNOME desktop environment, the user might be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the Region & Language application.

You can start this application in two ways. You can run the command **`gnome-control-center region`** from a terminal window, or on the top bar, from the system menu in the right corner, select the settings button (which has a crossed screwdriver and wrench for an icon) from the bottom left of the menu.

In the window that opens, select Region & Language. Click the **Language** box and select the preferred language from the list that appears. This also updates the **Formats** setting to the default for that language. The next time you log in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications such as **`gnome-terminal`** that are started inside it. However, by default they do not apply to that account if accessed through an **`ssh`** login from a remote system or a text-based login on a virtual console (such as **`tty5`**).

Note

You can make your shell environment use the same **`LANG`** setting as your graphical environment, even when you log in through a text-based virtual console or over **`ssh`**. One way to do this is to place code similar to the following in your **`~/.bashrc`** file. This example code will set the language used on a text login to match the one currently set for the user's GNOME desktop environment:

```
i=$(grep 'Language=' /var/lib/AccountsService/users/${USER} \
  | sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanese, Korean, Chinese, and other languages with a non-Latin character set might not display properly on text-based virtual consoles.

Individual commands can be made to use another language by setting the **`LANG`** variable on the command line:

```
[user@host ~]$ LANG=fr_FR.utf8 date
jeu. avril 25 17:55:01 CET 2019
```

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to determine the current value of **LANG** and other related environment variables.

Input Method Settings

GNOME 3 in Red Hat Enterprise Linux 7 or later automatically uses the IBus input method selection system, which makes it easy to change keyboard layouts and input methods quickly.

The Region & Language application can also be used to enable alternative input methods. In the Region & Language application window, the **Input Sources** box shows what input methods are currently available. By default, **English (US)** may be the only available method. Highlight **English (US)** and click the **keyboard** icon to see the current keyboard layout.

To add another input method, click the **+** button at the bottom left of the **Input Sources** window. An **Add an Input Source** window will open. Select your language, and then your preferred input method or keyboard layout.

When more than one input method is configured, the user can switch between them quickly by typing **Super**+**Space** (sometimes called **Windows**+**Space**). A *status indicator* will also appear in the GNOME top bar, which has two functions: It indicates which input method is active, and acts as a menu that can be used to switch between input methods or select advanced features of more complex input methods.

Some of the methods are marked with gears, which indicate that those methods have advanced configuration options and capabilities. For example, the Japanese **Japanese (Kana Kanji)** input method allows the user to pre-edit text in Latin and use **Down Arrow** and **Up Arrow** keys to select the correct characters to use.

US English speakers may also find this useful. For example, under **English (United States)** is the keyboard layout **English (international AltGr dead keys)**, which treats **AltGr** (or the right **Alt**) on a PC 104/105-key keyboard as a "secondary shift" modifier key and dead key activation key for typing additional characters. There are also Dvorak and other alternative layouts available.

> **Note**
> Any Unicode character can be entered in the GNOME desktop environment if you know the character's Unicode code point. Type **Ctrl**+**Shift**+**U**, followed by the code point. After **Ctrl**+**Shift**+**U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.
>
> For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl**+**Shift**+**U**, then **03BB**, then **Enter**.

System-wide Default Language Settings

The system's default language is set to US English, using the UTF-8 encoding of Unicode as its character set (**en_US.utf8**), but this can be changed during or after installation.

From the command line, the **root** user can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it displays the current system-wide locale settings.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

To set the system-wide default language, run the command **`localectl set-locale LANG=`*`locale`***, where *locale* is the appropriate value for the **`LANG`** environment variable from the "Language Codes Reference" table in this chapter. The change will take effect for users on their next login, and is stored in **`/etc/locale.conf`**.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME, an administrative user can change this setting from Region & Language by clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the graphical login screen will also adjust the system-wide default language setting stored in the **`/etc/locale.conf`** configuration file.

Important

Text-based virtual consoles such as **`tty4`** are more limited in the fonts they can display than terminals in a virtual console running a graphical environment, or pseudoterminals for **`ssh`** sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a text-based virtual console. For this reason, you should consider using English or another language with a Latin character set for the system-wide default.

Likewise, text-based virtual consoles are more limited in the input methods they support, and this is managed separately from the graphical desktop environment. The available global input settings can be configured through **`localectl`** for both text-based virtual consoles and the graphical environment. See the **`localectl`**(1) and **`vconsole.conf`**(5) man pages for more information.

Language Packs

Special RPM packages called *langpacks* install language packages that add support for specific languages. These langpacks use dependencies to automatically install additional RPM packages containing localizations, dictionaries, and translations for other software packages on your system.

To list the langpacks that are installed and that may be installed, use **`yum list langpacks-*`**:

```
[root@host ~]# yum list langpacks-*
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Installed Packages
langpacks-en.noarch      1.0-12.el8        @AppStream
Available Packages
langpacks-af.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-am.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-ar.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-as.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-ast.noarch     1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
...output omitted...
```

To add language support, install the appropriate langpacks package. For example, the following command adds support for French:

```
[root@host ~]# yum install langpacks-fr
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Use **`yum repoquery --whatsupplements`** to determine what RPM packages may be installed by a langpack:

```
[root@host ~]# yum repoquery --whatsupplements langpacks-fr
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Last metadata expiration check: 0:01:33 ago on Wed 06 Feb 2019 10:47:24 AM CST.
glibc-langpack-fr-0:2.28-18.el8.x86_64
gnome-getting-started-docs-fr-0:3.28.2-1.el8.noarch
hunspell-fr-0:6.2-1.el8.noarch
hyphen-fr-0:3.0-1.el8.noarch
libreoffice-langpack-fr-1:6.0.6.1-9.el8.x86_64
man-pages-fr-0:3.70-16.el8.noarch
mythes-fr-0:2.3-10.el8.noarch
```

Important

Langpacks packages use RPM *weak dependencies* in order to install supplementary packages only when the core package that needs it is also installed.

For example, when installing *langpacks-fr* as shown in the preceding examples, the *mythes-fr* package will only be installed if the *mythes* thesaurus is also installed on the system.

If *mythes* is subsequently installed on that system, the *mythes-fr* package will also automatically be installed due to the weak dependency from the already installed *langpacks-fr* package.

References

`locale`(7), **`localectl`**(1), **`locale.conf`**(5), **`vconsole.conf`**(5), **`unicode`**(7), and **`utf-8`**(7) man pages

Conversions between the names of the graphical desktop environment's X11 layouts and their names in **`localectl`** can be found in the file **`/usr/share/X11/xkb/rules/base.lst`**.

Language Codes Reference

Note

This table might not reflect all langpacks available on your system. Use **`yum info langpacks-`*`SUFFIX`*** to get more information about any particular langpacks package.

Language Codes

Language	Langpacks Suffix	$LANG value
English (US)	en	en_US.utf8

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Language	Langpacks Suffix	$LANG value
Assamese	as	as_IN.utf8
Bengali	bn	bn_IN.utf8
Chinese (Simplified)	zh_CN	zh_CN.utf8
Chinese (Traditional)	zh_TW	zh_TW.utf8
French	fr	fr_FR.utf8
German	de	de_DE.utf8
Gujarati	gu	gu_IN.utf8
Hindi	hi	hi_IN.utf8
Italian	it	it_IT.utf8
Japanese	ja	ja_JP.utf8
Kannada	kn	kn_IN.utf8
Korean	ko	ko_KR.utf8
Malayalam	ml	ml_IN.utf8
Marathi	mr	mr_IN.utf8
Odia	or	or_IN.utf8
Portuguese (Brazilian)	pt_BR	pt_BR.utf8
Punjabi	pa	pa_IN.utf8
Russian	ru	ru_RU.utf8
Spanish	es	es_ES.utf8
Tamil	ta	ta_IN.utf8
Telugu	te	te_IN.utf8

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 1

Introducing Performance Tuning

Goal

Describe performance tuning concepts and goals.

Objectives

- Define performance tuning goals, expectations, and terminology.
- Discuss recommended performance tuning methods of analysis.
- Define the standard units of measure and conversion arithmetic.

Sections

- Defining Performance Tuning Concepts (and Quiz)
- Discussing Performance Tuning Methodology (and Quiz)
- Describing Computer Units of Measure (and Guided Exercise)

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Defining Performance Tuning Concepts

Objectives

After completing this section, you should be able to define performance tuning goals, expectations, and terminology.

What is Performance Tuning?

Performance Tuning is the process of adjusting settings on a system to improve the utilization of computing resources, the throughput of data, or user experience. It requires an understanding of both the hardware and the software components of a system, as well as the many interactions between systems. Performance tuning has sometimes been called a "black art" due to the sometimes arcane knowledge required to effectively tune a system.

Performance tuning is often confused with troubleshooting, but there are significant differences between the two. In a troubleshooting process, the main objective is to find and solve a problem so that a system works. In performance tuning, the main objective is to make a system work with the best possible performance by taking advantage of resources and the environment where it operates.

Performance Tuning Concepts

Tuning Goals

Setting clear goals enables measurable gains after a successful performance tuning process. When setting tuning goals, it is important to consider how the system will be used and to understand potential trade-offs. Does the system support interactive users, or it is used to perform batch processing of reports? Is low latency (faster response time) or high throughput (more operations completed) more critical? Do real-time applications require predictable latency? If so, then the system could be tuned to minimize or eliminate unpredictable latency.

Performance Bottlenecks

A performance bottleneck occurs when a resource or device reaches its maximum operating capacity, affecting the overall behavior of the system. Depending on the device type, a bottleneck can slow down the operation of a system or even put the system completely out of service. An analytical method for effectively detecting bottlenecks will be introduced in a later section.

Human Factors

Human factors must also be considered when setting performance tuning goals. For example, management may ask for improvements to perceived performance versus actual system efficiency. The microprocessor that is at the heart of a computer system does not care whether it is running at 5% utilization or 95% utilization, but administrators might have a definite opinion about which situation is preferable. Human misinterpretation of results in the process of analyzing the performance of a system can lead to bad decisions that instead of improving can worsen or confuse the final expected outcome.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Performance Management

Performance management is the process of making sure that adequate computing resources are available to accomplish the business needs of all users. However, users may have unrealistic performance expectations because they do not compensate for the physical characteristics of hardware or other factors. As part of the performance management process, it may be necessary to manage or set expectations about what is considered acceptable performance for a system.

Workload Optimization

Frequently, performance tuning targets the optimization of settings for a specific type of workload. However, processes other than the target workload may be hampered by these settings.

For example, tuning to increase the amount of memory assigned for network buffers may increase throughput to the network. However, by increasing the memory dedicated to this task, there is less system memory available for other processes. Less available system memory may lead to increased paging, or swapping, which will add to the amount of disk I/O being performed by the machine. If the benefit of increasing network throughput is the desired result, and less available system memory or increased paging is an acceptable alternative, then the change is acceptable.

Throughput and Latency

Two of the most common performance measures are throughput and latency. Performance tuning decisions require trade-offs that commonly put throughput and latency in opposition. In reality, throughput and latency should complement each other for optimal performance.

Throughput is the measure of how much data can be transferred or processed by a resource in a given time. Storage, memory, CPUs, and network devices are common resource types that can be measured for throughput performance.

Latency is the delay that a resource must wait to start data transfer or processing. Storage, memory, network devices, and CPUs can be measured for latency performance.

Capacity Planning

In the IT world, capacity planning is the process that is responsible for estimating the resources that will be necessary to support a system's workload for a specific period of time. These resources are often hardware, software, and connectivity, but they can expand to include environmental resources like power energy, thermal energy, and physical space. Effective capacity planning relies on a properly designed architecture to handle the desired workload. A good capacity planning process should focus on the following concepts:

Scalability
: The ability to scale a system is one of the key requirements for quickly adapting to different workloads.

Limitations
: It is essential to know the limitations of the system, including both common performance metrics (throughput, latency, transactions per second, storage capacity, etc.) and the escalation limits of the system.

Critical Resources
: It is important to know which resources will be critical or most used for specific workloads.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Microarchitectural Data Sampling (MDS) Vulnerabilities

Red Hat is aware of a series of microarchitectural (hardware) implementation issues that could allow an unprivileged local attacker to bypass conventional memory security restrictions to gain read access to privileged memory that would otherwise be inaccessible. Malicious code running within a container could also exploit these flaws. These issues affect many modern Intel microprocessors with Simultaneous Multithreading (SMT) or Hyper-Threading (HT) features, requiring updates to the Linux kernel, virtualization stack, and CPU microcode.

Intel microprocessor implements several predictive performance micro-optimizations that are the principal cause of these security flaws. The exploitation of the vulnerabilities allows attackers a side channel to access recently used data on the system belonging to other processes, containers, virtual machines, or to the kernel.

These vulnerabilities rely upon leveraging speculation to retrieve a data state left within internal CPU structures.

The following are the official names given to each one of these security flaws:

- Microarchitectural Store Buffer Data Sampling (MSBDS)
- Microarchitectural Load Port Data Sampling (MLPDS)
- Microarchitectural Fill Buffer Data Sampling (MFBDS)
- Microarchitectural Data Sampling Uncacheable Memory (MDSUM)

The mitigation to these security flaws have a demonstrated performance impact. Disabling SMT/HT to mitigate MDS affects the overall performance of the processor. The impact is greater in applications with high rates of user-kernel-user space transitions, such as system calls, NMIs, and interrupts.

The performance impact is variable for each workload. Red Hat testing has produced the following results:

Applications that spent a lot of time in user mode tended to show the smallest slowdown, usually in the 0-5% range. Applications that did a lot of small block or small packet network I/O showed slowdowns in the 10-25% range. Some microbenchmarks that did nothing other than enter and return from user space to kernel space showed higher slowdowns. The performance impact from MDS mitigation can be measured by running your application with MDS enabled and then disabled.

Consider the following tips about the mitigation of these security flaws:

- MDS mitigation is enabled by default in Red Hat Enterprise Linux operative systems after applying corresponding updates.
- MDS mitigation can be fully enabled, and SMT/HT disabled, by adding the **`"mds=full,nosmt"`** flag to the kernel boot command line.
- MDS mitigation can be fully disabled by adding the **`"mds=off"`** flag to the kernel boot command line.
- There is no way to disable MDS mitigation at runtime.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

For more information, refer to the *Monitoring and managing system status and performance* guide at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index

Fopr more general Red Hat Enterprise Linux information, refer to the previous *Performance Tuning Guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index

MDS - Microarchitectural Data Sampling - CVE-2018-12130, CVE-2018-12126, CVE-2018-12127, and CVE-2019-11091
https://access.redhat.com/security/vulnerabilities/mds

Capacity planning
http://www.computerbusinessresearch.com/Home/enterprise-architecture/capacity-planning

System Performance Tuning, 2nd Edition
https://www.oreilly.com/library/view/system-performance-tuning/059600284X/ch01.html

Quiz

Defining Performance Tuning Concepts

Choose the correct answers to the following questions:

1. The process of adjusting settings on a system to improve the utilization of computing resources, the throughput of data, or the user experience, is the correct definition of:

a. Performance management.
b. Workload optimization.
c. Performance tuning.
d. Tuning goals.

2. When a resource has reached its maximum operating capacity in a system, we are referring to:

a. Low throughput.
b. Long queue.
c. Performance bottleneck.
d. High latency.

3. Which three of the following concepts are important in the capacity planning process? (Choose three)

a. Scalability
b. Limitations
c. Mean time between failures
d. Critical resources

4. Which three of the following statements about MDS are correct? (Choose three)

a. MDS vulnerabilities affect only virtual machines and containers.
b. MDS vulnerabilities affect physical hosts, virtual machines, and containers.
c. MDS vulnerabilities affect AMD-based processors.
d. MDS vulnerabilities affect ARM-based processors.
e. MDS vulnerabilities can be mitigated using software-based fixes.
f. Disabling SMT/HT to mitigate MDS affects the overall performance of the processor.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Defining Performance Tuning Concepts

Choose the correct answers to the following questions:

1. The process of adjusting settings on a system to improve the utilization of computing resources, the throughput of data, or the user experience, is the correct definition of:

a. Performance management.
b. Workload optimization.
c. Performance tuning.
d. Tuning goals.

2. When a resource has reached its maximum operating capacity in a system, we are referring to:

a. Low throughput.
b. Long queue.
c. Performance bottleneck.
d. High latency.

3. Which three of the following concepts are important in the capacity planning process? (Choose three)

a. Scalability
b. Limitations
c. Mean time between failures
d. Critical resources

4. Which three of the following statements about MDS are correct? (Choose three)

a. MDS vulnerabilities affect only virtual machines and containers.
b. MDS vulnerabilities affect physical hosts, virtual machines, and containers.
c. MDS vulnerabilities affect AMD-based processors.
d. MDS vulnerabilities affect ARM-based processors.
e. MDS vulnerabilities can be mitigated using software-based fixes.
f. Disabling SMT/HT to mitigate MDS affects the overall performance of the processor.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Discussing Performance Tuning Methodology

Objectives

After completing this section, you should be able to discuss recommended performance tuning methods of analysis.

Analyzing System Performance

Many different methodologies and strategies can be applied to for system performance improvement. Before making changes, when looking for improvements, a system's current performance must be analyzed. Knowing how to begin system analysis is a critical step in this process.

The Utilization Saturation and Errors (USE) Method is highly regarded among performance tuning experts. The USE method was developed by performance expert Brendan Gregg while teaching classes in operating systems performance. The method was first published in the December 2012 issue of *acmqueue*, the journal of the Association for Computing Machinery.

The USE Method

The USE Method helps you to quickly analyze the performance of a system and find possible performance issues, or bottlenecks, without overlooking important areas. This analysis is essential to establish a baseline, and consequently be able to start making changes in the system in search of performance improvements.

The main idea is to identify each individual resource first, and then check errors, utilization, and saturation for every one of the resources.

The USE Method is most effective in complex systems with hardware resources that suffer performance degradation under high utilization or saturation. The USE Method is not effective in handling software resources, like some other methods. Therefore, focus on hardware resources when applying the USE Method. Remember that these hardware resources are the starting point for every IT system topology.

Terminology Definitions:

The following list of terms are commonly used in the USE Method:

Resources
: Functional hardware devices in a server (CPUs, memory, network interfaces, disks, controllers).

Utilization
: The average time that the resource was busy servicing work.

Saturation:
: The degree to which the resource has additional work that it can not service. This work is often queued, but sometimes rejected depending on the type of resource.

Errors
: The number of error events.

Metrics and Resource Types

Reading and understanding metrics for every resource on your system can be difficult and time consuming. Work on a subset of common resource types and make them the priority, knowing that there are other components to investigate if more in-depth analysis is needed.

For this subset of common resource types, metrics are usually expressed in the following terms:

CPU resource:
The CPU is a processing resource that handles queues and priorities. The usual metrics for measuring CPU performance are percentage of use, load average, and queue average.

Memory resource:
Memory is a capacity resource. The usual metrics for measuring performance are free capacity, throughput, and errors.

Storage resources:
Storage devices are both a capacity and an I/O resource. The usual metrics for measuring storage device performance are free space, I/O operations per second (IOPS), I/O wait time, and throughput.

Network resources:
Network devices are considered an I/O resource, and the usual metrics for measuring performance are throughput, round-trip-time, latency, packet loss, errors, and collisions.

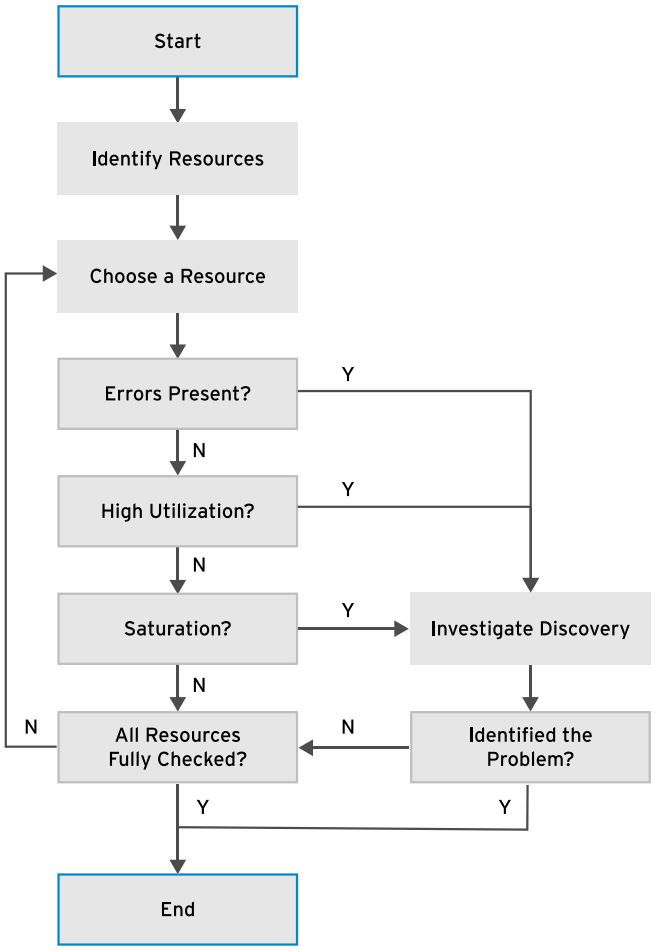


Figure 1.1: The USE Method Workflow

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Check errors first whether your analysis situation is performance-based or troubleshooting-based. High interrupts or other manifestations of problems that cause detectable errors, are a root cause of performance issues in both scenarios and should always be detected and corrected before any other analysis of saturation and utilization.

Usually, systems exhibit more than one performance problem; the first problem you find may not be the root problem. The goal is to collect evidence that will lead you to apply an action. This action can be a corrective action, or it could apply a specific tune to a device or system for a particular workload.

Interpreting the Performance Results

Some performance problems are well understood. Others depend on workload requirements, expectations, or the overall functional environment. Closing the scope of an investigation can quickly bring focus to the problem area. The following are some general suggestions for interpreting results obtained through analysis with the USE Method:

Utilization:
Full utilization (100%) is usually a sign of a bottleneck (check saturation and its effect to confirm the bottleneck). High utilization (greater than 70%) for an extended time (many seconds or minutes) can hide short bursts of 100% utilization. There are some system resources, such as hard disks, that cannot be interrupted during an operation, even for higher priority work. Once utilization is over 70%, queuing delays that impact overall system performance may become more frequent and noticeable. Other system resources, such as CPUs, can be interrupted (or "preempted") at almost any time and can handle different priorities for managing the workload.

Saturation:
Any degree of saturation can be a problem (non-zero). Saturation is measured as the length of a wait queue, or the time spent waiting in the queue before being processed.

Errors:
Non-zero error counters are worth investigating, especially if they are still increasing while facing degradation in performance.

Applying Changes to a System

Now that you have a clear process for analyzing the system's performance, you can look at how to make changes to obtain gains in performance.

Many companies utilize change management procedures. Making performance tuning changes to a system should integrate into the existing change management procedure. Usually these procedures consist of:

Setting a baseline
In a performance tuning scenario, this procedure would run the test workload established in the performance-level agreement and gather metrics from the machine, which will be remeasured and compared to the same data later. Save the metrics for a later review, as this is a crucial step.

Performing changes
Execute the changes on the system. Generally, administrators should change one item at a time and measure the effect. Make several changes that are unrelated to each other can make it difficult for the administrator to determine the effectiveness of the changes made. In some cases, a complex configuration requires modifying multiple settings with an aimed goal; in these cases, refer to the official documentation accordingly.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Verifying the effectiveness of the change

After executing the changes, rerun the same test workload on the machine. Looking at the metrics that are most important, the administrator can validate that the change was not only executed, but effective in changing the behavior of the system.

Reversing the change and comparing with the baseline

Reverting the changes and returning to the baseline gives you an extra degree of confidence in the effectiveness of the proposed changes.

Applying the definitive change

Apply and document the definitive changes. It is common to forget to document the changes, but this step is crucial for future analysis, and possible rollback procedures.

References

Linux Performance Analysis in 60,000 Milliseconds
https://medium.com/netflix-techblog/linux-performance-analysis-in-60-000-milliseconds-accc10403c55

Thinking Methodically about Performance
https://queue.acm.org/detail.cfm?id=2413037

The USE Method
http://www.brendangregg.com/usemethod.html

Quiz

Discussing Performance Tuning Methodology

Choose the correct answers to the following questions:

1. According to the USE Method flowchart, which step would comes first?

a. Review for errors
b. Identify devices
c. Review for high utilization
d. Review for saturation

2. Which two of the following resource types are storage devices, according to the USE Method? (Choose two)

a. Processing resource
b. Queue resource
c. Capacity resource
d. Input resource
e. I/O resource

3. Which three metrics are useful when analyzing performance in a CPU resource? (Choose three)

a. Queue average
b. Bus throughput
c. Percentage of use
d. Latency.
e. Load average
f. I/O operations per second (IOPS)

4. In terms of the effectiveness of the USE Method, select the correct statement.

a. The USE Method is most effective for hardware resources that suffer performance degradation under low utilization or saturation.
b. The USE Method is less effective for software resources that suffer performance degradation under low utilization or saturation.
c. The USE Method is most effective in complex systems with hardware resources that suffer performance degradation under high utilization or saturation.
d. The USE Method is most effective for resources that suffer performance degradation under low to average utilization or saturation.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Which three metrics are commonly used when analyzing performance in storage resources? (Choose three)

a. Queue average
b. I/O operations per second (IOPS)
c. Percentage of use
d. I/O wait time
e. Free space
f. I/O collisions

Solution

Discussing Performance Tuning Methodology

Choose the correct answers to the following questions:

1. According to the USE Method flowchart, which step would comes first?

a. Review for errors
b. Identify devices
c. Review for high utilization
d. Review for saturation

2. Which two of the following resource types are storage devices, according to the USE Method? (Choose two)

a. Processing resource
b. Queue resource
c. Capacity resource
d. Input resource
e. I/O resource

3. Which three metrics are useful when analyzing performance in a CPU resource? (Choose three)

a. Queue average
b. Bus throughput
c. Percentage of use
d. Latency.
e. Load average
f. I/O operations per second (IOPS)

4. In terms of the effectiveness of the USE Method, select the correct statement.

a. The USE Method is most effective for hardware resources that suffer performance degradation under low utilization or saturation.
b. The USE Method is less effective for software resources that suffer performance degradation under low utilization or saturation.
c. The USE Method is most effective in complex systems with hardware resources that suffer performance degradation under high utilization or saturation.
d. The USE Method is most effective for resources that suffer performance degradation under low to average utilization or saturation.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Which three metrics are commonly used when analyzing performance in storage resources? (Choose three)

a. Queue average

b. I/O operations per second (IOPS)

c. Percentage of use

d. I/O wait time

e. Free space

f. I/O collisions

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Describing Computer Units of Measure

Objectives

After completing this section, you should be able to define the standard units of measure and conversion arithmetic.

Decimal and Binary Units

When measuring performance, it is important to accurately measure and report quantities. The modern metric system, also known as the *International System of Units* (or by its French abbreviation, *SI*), is the most common measurement system in the world.

The International Electrotechnical Commission (IEC) adopted an alternative standard to distinguish between SI decimal prefixes and a new set of binary prefixes. They constructed the names of these new binary prefixes by taking the first two letters of the analogous decimal prefix (for example, *ki-* from kilo-) and appending the letters *-bi-* for binary.

International System of Units (SI) prefixes are *decimal*:

```
kilo- (k) = 10^3  = 1,000
mega- (M) = 10^6  = 1,000,000
giga- (G) = 10^9  = 1,000,000,000
tera- (T) = 10^12  = 1,000,000,000,000
peta- (P) = 10^15  = 1,000,000,000,000,000
exa-  (E) = 10^18  = 1,000,000,000,000,000,000
```

The International Electrotechnical Commission (IEC) prefixes are *binary*:

```
kibi- (Ki) = 2^10  = 1,024
mebi- (Mi) = 2^20  = 1,048,576
gibi- (Gi) = 2^30  = 1,073,741,824
tebi- (Ti) = 2^40  = 1,099,511,627,776
pebi- (Pi) = 2^50  = 1,125,899,906,842,624
exbi- (Ei) = 2^60  = 1,152,921,504,606,846,976
```

Consider how disk manufacturers advertise capacities. How many bytes are in a 2 TB disk?

2 TB is 2×10^{12} bytes, which is 2,000,000,000,000 bytes.

A disk is a raw, physical device with no virtual structures on it before you build a file system; hardware manufacturers use the SI decimal prefix to express unit capablilities.

However, a raw disk becomes a block disk structure when it is formatted as a file system. File systems are binary structures, some of which use a (default) block size of 4096 (4 KiB). Using this nomenclature, a terabyte is understood as 10^{12} bytes. This is why some operating system tools may report in "TB" but they are actually measuring in binary tebibytes (TiB).

A tool that reports disk size in "TB" but measures "TB" as tebibytes would report a 2 TB disk as only "1.82 TB" (equal to 1.82 *TiB*) in size, representing a difference of almost 10 percent.

The lesson is that it is very important to understand exactly what units are being reported by the tools used to analyze a system, or else significant mistakes in analysis can be made.

> **Note**
> If there is a binary prefix, then there is no ambiguity: an administrator will know the quantity is being reported using the binary interpretation. However, if a normal decimal prefix is used, an administrator may not know for sure whether the reporter meant to use the binary interpretation or the decimal one.
>
> To address this problem, some parts of the industry have standardized on particular understandings of how to interpret the SI prefixes. Memory manufacturers usually use binary interpretations; a gigabyte of RAM is considered to be 2^{30} bytes. In telecommunications, a gigabit is always 10^9 bits. The storage industry also uses decimal interpretations of prefixes, so a gigabyte of hard disk space is 10^9 bytes.

Unit conversions

When analyzing data, it is critical to convert between units correctly. Converting one unit at a time helps avoid errors and simplifies the process.

To illustrate, consider this simple example: *How many seconds are in one hour?* There are 60 seconds in one minute, and 60 minutes in one hour, so the calculation would look like this:

```
1 hour = 60 minutes/hour × 60 seconds/minute = 3600 seconds/hour
```

Next, convert 10,000 MiB/h into MiB/s.

```
10,000 MiB/h ÷ 60 min/hour = 166.7 MiB/min
166.7 MiB/min ÷ 60 seconds/min = 2.778 MiB/s
```

Ever wondered how much data a gigabit network card can transfer? Convert 1 Gb/s into MiB/s.

```
1 Gb/s × 10^9 b/Gb ÷ 8 bits/1 Byte ÷ 2^20 Bytes/1 MiB = about 119.2 MiB/s
```

The following example demonstrates the previous calculation using the **bc** and **dc** console calculators:

The **dc** calculator stands for "reverse-polish desktop calculator", emulates stack arithmetic, which is familiar to many low-level programmers.

```
[user@host ~]$ dc
8k
1
10
9^*
8/
2
20^/
p
119.20928955
Ctrl+D
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The **bc** calculator stands for "basic calculator" (with infix notation) and is implemented on top of **dc** calculator.

```
[user@host ~]$ bc
scale=8
1*10^9/8/2^20
119.20928955
Ctrl+D
```

References

Definitions of the SI Units: The Binary Prefixes
http://physics.nist.gov/cuu/Units/binary.html

Units conversion by factor-label
http://en.wikipedia.org/wiki/Units_conversion_by_factor-label

bc(1), **dc**(1), **units**(7) man pages

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Converting Computer Units of Measure

In this exercise, you will convert performance metrics from one unit of measure to another.

Outcomes

You should be able to convert performance metrics units.

Before You Begin

Log in to the **`workstation`** virtual machine as the **`student`** user with the password **`student`**.

On **`workstation`**, run the **`lab intro-measure start`** command to set up the environment for the guided exercise.

```
[student@workstation ~]$ lab intro-measure start
```

1. Convert 100 Mib/s (mebibits per second) to GiB/h (gibibytes per hour).

```
100 Mib/s × 60 (sec in 1 min) × 60 (min in 1 hour) × 2^20 (bits in 1 Mib) ÷ 8 (bits
 in 1 Byte) ÷ 2^30 (bits in 1 GiB)
```

```
[student@workstation ~]$ bc
scale=8
100*60*60*2^20/8/2^30
43.94531250
Ctrl+D
```

100 Mib/s equals 43.95 GiB/h.

2. Convert 43 KiB/sec (kibibytes per second) to MiB/min (mebibytes per minute).

```
43 KiB/sec ÷ 1024 (KiB in 1 MiB) × 60 (sec in 1 min)
```

```
[student@workstation ~]$ bc
scale=8
43/1024*60
2.51953080
Ctrl+D
```

43 KiB/sec is 2.52 MiB/min.

3. How many million transactions per day is equal to 143 tps (transactions per second)?

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
143 trans/sec × 60 (sec in 1 min) × 60 (min in 1 hr) × 24 (hr in 1 day) ÷ 1000000
 (transactions in 1 million transaction)
```

```
[student@workstation ~]$ bc
scale=8
143*60*60*24/1000000
12.35520000
Ctrl+D
```

143 tps equals 12.36 million transactions per day.

4. A fortnight is 14 days. How many hours equals 32738 microfortnights?

```
32738 microfortnights ÷ 1000000 (microfortnights in 1 fortnight) × 14 (days in 1
 fortnight) × 24 (hr in 1 day)
```

```
[student@workstation ~]$ bc
scale=8
32738/1000000*14*24
10.99996800
Ctrl+D
```

32738 microfortnights is 11 hours.

5. What is 450 Kbps (kilobits per second) when converted to GiB/day (gibibytes per day)?

```
450 Kbps × 3600 (sec in 1 hr) × 24 (hr in 1 day) × 10^3 (bits in 1 Kilobit) ÷ 8
 (bits in 1 Byte) ÷ 2^30 (bytes in 1 GiB)
```

```
[student@workstation ~]$ bc
scale=8
450*3600*24*10^3/8/2^30
4.52622771
Ctrl+D
```

450 Kbps is 4.52 GiB/day.

Finish

On **`workstation`**, run the **`lab intro-measure finish`** command to complete this exercise.

```
[student@workstation ~]$ lab intro-measure finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- Performance tuning concepts.
- The trade-offs in performance tuning when you face security vulnerabilities like MDS.
- The importance of the USE Method in performance analysis.
- The different measurement systems used in the IT world.
- How to convert performance metrics from one unit of measure to another.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 2

Selecting Performance Monitoring Tools

Goal

Evaluating the large selection of performance monitoring tools that are included with Red Hat Enterprise Linux.

Objectives

- Identify the common system monitoring tools and explain the purpose of each tool.
- Describe the sysstat architecture and explain typical sysstat utility command use.
- Describe the Performance Co-Pilot utility and its standardized data structures.

Sections

- Identifying System Monitoring Tools (and Guided Exercise)
- Describing the Sysstat Package Utilities (and Guided Exercise)
- Collecting Performance Data with Performance Co-Pilot (and Guided Exercise)

Lab

Selecting Performance Monitoring Tools

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Identifying System Monitoring Tools

Objectives

After completing this section, you should be able to identify the common system monitoring tools and explain the purpose of each tool.

System Monitoring

System monitoring is a helpful approach to provide the user with data regarding the actual timing behavior of the system. Users can perform further analysis using the data that these monitors provide. One of the goals of system monitoring is to determine whether the current execution meets the specified technical requirements.

These monitoring tools retrieve commonly viewed information, and can be used by way of the command line or a graphical user interface, as determined by the system administrator. These tools display information about the Linux system, such as free disk space, the temperature of the CPU, and other essential components, as well as networking information, such as the system IP address and current rates of upload and download.

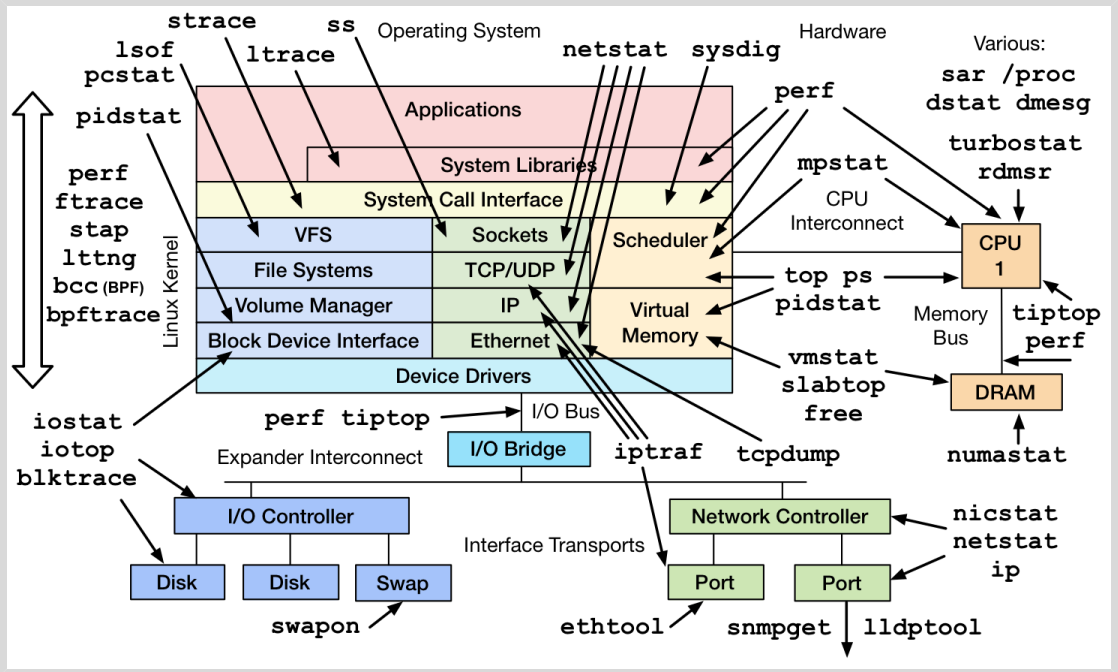


Figure 2.1: Linux Performance Observability Tools by Brendan Gregg (CC BY-SA 4.0)

Monitoring Tools

The Linux kernel maintains *counter*structures for counting events, that increment when an event occurs. For example, disk reads and writes, and process system calls, are events that increment counters with values stored as unsigned integers. Monitoring tools read these counter values. These tools provide either per process statistics maintained in process structures, or system-wide statistics in the kernel. Monitoring tools are typically viewable by non-privileged users. The **`ps`** and **`top`** commands provide process statistics, including CPU and memory.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Monitoring Processes Using the ps Command

Troubleshooting a system requires understanding how the kernel communicates with processes, and how processes communicate with each other. At process creation, the system assigns a state to the process.

Use the **ps aux** command to list all users with extended user-oriented details; the resulting list includes the terminal from which processes are started, as well as processes without a terminal. A **?** sign in the **TTY** column represents that the process did not start from a terminal.

```
[user@host]$ ps aux
USER   PID %CPU %MEM    VSZ   RSS TTY      STAT START TIME COMMAND
user  1350  0.0  0.2 233916  4808 pts/0    Ss   10:00   0:00 -bash
root  1387  0.0  0.1 244904  2808 ?        Ss   10:01 0:00 /usr/sbin/anacron -s
root  1410  0.0  0.0      0     0 ?        I    10:08   0:00 [kworker/0:2...
root  1435  0.0  0.0      0     0 ?        I    10:31   0:00 [kworker/1:1...
user  1436  0.0  0.2 266920  3816 pts/0    R+   10:48   0:00 ps aux
```

The Linux version of **ps** supports three option formats:

- UNIX (POSIX) options, which may be grouped and must be preceded by a dash.
- BSD options, which may be grouped and must not include a dash.
- GNU long options, which are preceded by two dashes.

The output below uses the UNIX options to list every process with full details:

```
[user@host]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         2     0  0 09:57 ?        00:00:00 [kthreadd]
root         3     2  0 09:57 ?        00:00:00 [rcu_gp]
root         4     2  0 09:57 ?        00:00:00 [rcu_par_gp]
...output omitted...
```

Key Columns in ps Output

PID
This column shows the unique process ID.

TIME
This column shows the total CPU time consumed by the process in hours:minutes:seconds format, since the start of the process.

%CPU
This column shows the CPU usage during the previous second as the sum across all CPUs expressed as a percentage.

RSS
This column shows the non-swapped physical memory that a process consumes in kilobytes in the resident set size, **RSS** column.

%MEM
This column shows the ratio of the process' resident set size to the physical memory on the machine, expressed as a percentage.

Use the **-p** option together with the **pidof** command to list the **sshd** processes that are running.

```
[user@host ~]$ ps -p $(pidof sshd)
  PID TTY      STAT   TIME COMMAND
  756 ?        Ss     0:00 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com...
 1335 ?        Ss     0:00 sshd: user [priv]
 1349 ?        S      0:00 sshd: user@pts/0
```

Use the following command to list of all processes sorted by memory usage in descending order:

```
[user@host ~]$ ps ax --format pid,%mem,cmd --sort -%mem
  PID %MEM CMD
  713  1.8 /usr/libexec/sssd/sssd_nss --uid 0 --gid 0 --logger=files
  715  1.8 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork --nopid
  753  1.5 /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P
  687  1.2 /usr/lib/polkit-1/polkitd --no-debug
  731  0.9 /usr/sbin/NetworkManager --no-daemon
...output omitted...
```

Various other options are available for **ps** including the **o** option to customize the output and columns shown.

Monitoring Process Using top

The **top** command provides a real-time report of process activities with an interface for the user to filter and manipulate the monitored data. The command output shows a system-wide summary at the top and process listing at the bottom, sorted by the top CPU consuming task by default. The **-n 1** option terminates the program after a single display of the process list. The following is an example output of the command:

```
[user@host ~]$ top -n 1
Tasks: 115 total,   1 running, 114 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  3.2 sy,  0.0 ni, 96.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1829.0 total,   1426.5 free,    173.6 used,    228.9 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   1495.8 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0  243968  13276   8908 S   0.0   0.7   0:01.86 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
...output omitted...
```

Useful Key Combinations to Sort Fields

RES
: Use **Shift**+**M** to sort the processes based on resident memory.

PID
: Use **Shift**+**N** to sort the processes based on process ID.

TIME+
: Use **Shift**+**T** to sort the processes based on CPU time.

Press **F** and select a field from the list to use any other field for sorting.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

> **Important**
> The **top** command imposes a significant overhead on the system due to various system calls. While running the **top** command, the process running the **top** command is often the top CPU-consuming process.

Monitoring Memory Usage

The **free** command lists both free and used physical memory and swap memory. The **-b**, **-k**, **-m**, **-g** options show the output in bytes, KB, MB, or GB, respectively. The **-s** option is passed as an argument that specifies the number of seconds between refreshes. For example, **free -s 1** produces an update every 1 second.

```
[user@host ~]$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1829         172        1427          16         228        1496
Swap:             0           0           0
```

The near zero values in the **buff/cache** and **available** columns indicate a low memory situation. If the available memory is more than 20% of the total, and the used memory is close to the total memory, then these values indicate a healthy system.

Monitoring File System Usage

One stable identifier that is associated with a file system is its UUID, a very long hexadecimal number that acts as a universally unique identifier. This UUID is part of the file system and remains the same as long as the file system is not recreated. The **lsblk -fp** command lists the full path of the device, along with the UUIDs and mount points, as well as the type of file system in the partition. If the file system is not mounted, the mount point displays as blank.

```
[user@host ~]$ lsblk -fp
NAME        FSTYPE LABEL UUID                                   MOUNTPOINT
/dev/vda
├─/dev/vda1 xfs          23ea8803-a396-494a-8e95-1538a53b821c /boot
├─/dev/vda2 swap         cdf61ded-534c-4bd6-b458-cab18b1a72ea [SWAP]
└─/dev/vda3 xfs          44330f15-2f9d-4745-ae2e-20844f22762d /
/dev/vdb
└─/dev/vdb1 xfs          46f543fd-78c9-4526-a857-244811be2d88
```

The **findmnt** command allows the user to take a quick look at what is mounted where, and with which options. Executing the **findmnt** command without any options lists out all the mounted file systems in a tree layout. Use the **-s** option to read the file systems from the **/etc/fstab** file. Use the **-S** option to search the file systems by the source disk.

```
[user@host ~]$ findmnt -S /dev/vda1
TARGET SOURCE    FSTYPE OPTIONS
/      /dev/vda1 xfs    rw,relatime,seclabel,attr2,inode64,noquota
```

The **df** command provides information about the total usage of the file systems. The **-h** option transforms the output into a human-readable form.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[user@host ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        892M     0  892M   0% /dev
tmpfs           915M     0  915M   0% /dev/shm
tmpfs           915M   17M  899M   2% /run
tmpfs           915M     0  915M   0% /sys/fs/cgroup
/dev/vda1        10G  1.5G  8.6G  15% /
tmpfs           183M     0  183M   0% /run/user/1000
```

The **du** command displays the total size of all the files in a given directory and its subdirectories. The **-s** option suppresses the output of detailed information and displays only the total. Similar to the **df -h** command, the **-h** option displays the output into a human-readable form.

```
[user@host ~]$ du -sh /home/user
16K /home/user
```

Using GNOME System Monitor

The System Monitor available on the GNOME desktop provides statistical data about the system status, load, and processes, as well as the ability to manipulate those processes. Similar to other monitoring tools, such as the **top**, **ps**, and **free** commands, the System Monitor provides both the system-wide and per-process data. These monitoring tools retrieve commonly viewed information, and can be used by way of the command line or a graphical user interface, as determined by the system administrator. Use the **gnome-system-monitor** command to access the application from a command terminal.

To view the CPU usage, go to the **Resources** tab and look at the **CPU History** chart.

Figure 2.2: CPU usage history in System Monitor

The virtual memory is the sum of the physical memory and the swap space in a system. A running process maps the location in physical memory to files on disk. The memory map displays the total virtual memory consumed by a running process, which determines the memory cost of running that process instance. The memory map also displays the shared libraries used by the process.

Memory Maps

Memory maps for process "stress" (PID 14703):

Filename	VM Start	VM End	VM Size	Flags	VM Offset	Private clean
	f7f89000	f7f8b000	8.0 KiB	rw-p	00000000	0 bytes
	f7eb4000	f7eb7000	12.0 KiB	rw-p	00000000	0 bytes
	f7c0e000	f7d0f000	1.0 MiB	rw-p	00000000	0 bytes
[heap]	58242000	58264000	136.0 KiB	rw-p	00000000	0 bytes
[stack]	ffaab000	ffacd000	136.0 KiB	rw-p	00000000	0 bytes
/usr/bin/stress	565fa000	565fb000	4.0 KiB	rw-p	00007000	0 bytes
/usr/bin/stress	565f9000	565fa000	4.0 KiB	r--p	00006000	0 bytes
/usr/bin/stress	565f7000	565f9000	8.0 KiB	r--p	00005000	0 bytes
/usr/bin/stress	565f3000	565f7000	16.0 KiB	r-xp	00001000	0 bytes
/usr/bin/stress	565f2000	565f3000	4.0 KiB	r--p	00000000	0 bytes
/usr/lib/ld-2.28.so	f7fa3000	f7fcb000	160.0 KiB	r-xp	00000000	0 bytes
/usr/lib/ld-2.28.so	f7fcb000	f7fcc000	4.0 KiB	r--p	00027000	0 bytes
/usr/lib/ld-2.28.so	f7fcc000	f7fcd000	4.0 KiB	rw-p	00028000	0 bytes
/usr/lib/libc-2.28.so	f7eb0000	f7eb1000	4.0 KiB	---p	001a1000	0 bytes
/usr/lib/libc-2.28.so	f7eb1000	f7eb3000	8.0 KiB	r--p	001a1000	0 bytes
/usr/lib/libc-2.28.so	f7eb3000	f7eb4000	4.0 KiB	rw-p	001a3000	0 bytes
/usr/lib/libc-2.28.so	f7d0f000	f7eb0000	1.6 MiB	r-xp	00000000	0 bytes
/usr/lib/libm-2.28.so	f7eb7000	f7f87000	832.0 KiB	r-xp	00000000	0 bytes
/usr/lib/libm-2.28.so	f7f87000	f7f88000	4.0 KiB	r--p	000cf000	0 bytes
/usr/lib/libm-2.28.so	f7f88000	f7f89000	4.0 KiB	rw-p	000d0000	0 bytes

Figure 2.3: Memory map of a process in System Monitor

To display the memory map of a process in System Monitor, locate a process in the **Processes** tab, right-click a process in the list, and select **Memory Maps**.

References

`ps`(1), **`top`**(1), **`free`**(1), **`df`**(1), **`findmnt`**(8), **`lsblk`**(8), and **`du`**(1) man pages

Gnome Help page for System Monitor
https://help.gnome.org/users/gnome-system-monitor/stable/

The original graphic by Brendan Gregg can be found at
Linux Performance Observability Tools
http://www.brendangregg/perf.html/
It is licensed under Attribution-ShareAlike 4.0 International
https://creativecommons.org/licenses/by-sa/4.0/legalcode

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Identifying System Monitoring Tools

In this exercise, you will analyze the output of system monitoring tools.

Outcomes

You should be able to interpret the output of **ps**, **top**, **free**, **lsblk**, **findmnt**, **df**, **du**, and **gnome-system-monitor** commands.

Before You Begin

Log in to the **workstation** VM as the **student** user with **student** as the password.

On the **workstation** VM, run the **lab perftools-system start** command. This command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the *nginx* package, starts the service, creates a **/dev/vdb1** partition, and mounts **/opt**.

```
[student@workstation ~]$ lab perftools-system start
```

1. On the **workstation** VM, use **ssh** to log in to **servera** as the **student** user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$
```

2. Use the **ps** command to record your findings. Consult the man page for the **ps** command wherever required.

   2.1. List the processes that are associated with the **nginx** service.

```
[student@servera ~]$ ps -p $(pidof nginx)
  PID TTY      STAT   TIME COMMAND
 3240 ?        Ss     0:00 nginx: master process /usr/sbin/nginx
 3241 ?        S      0:00 nginx: worker process
 3242 ?        S      0:00 nginx: worker process
```

   2.2. Retrieve information to find out how long the **nginx:master** process has been running. In the output below, **3240** is the process ID of the **nginx:master** process.

```
[student@servera ~]$ ps -p 3240 -o etime
    ELAPSED
      01:42
```

   2.3. Print the **nginx** process hierarchy, with the **PID**, **PPID**, and **CMD** columns.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@servera ~]$ ps --forest -C nginx -o pid,ppid,cmd
  PID  PPID CMD
 3240     1 nginx: master process /usr/sbin/nginx
 3241  3240  \_ nginx: worker process
 3242  3240  \_ nginx: worker process
```

2.4. List the processes using the BSD style, and sort those processes in descending order based on resident set size (RSS). The output must display the following columns: **PID**, **PPID**, **RSS**, and **%CPU**.

```
[student@servera ~]$ ps axo pid,ppid,rss,%cpu --sort=-rss
  PID  PPID   RSS %CPU
  714   685 35164  0.0
  713     1 34332  0.0
  753     1 30164  0.0
  ...output omitted...
```

2.5. List the processes running as the **student** user.

```
[student@servera ~]$ ps -U student
  PID TTY          TIME CMD
 5548 ?        00:00:00 systemd
 5742 ?        00:00:00 (sd-pam)
 6027 ?        00:04:07 stress.py
25299 ?        00:00:00 sshd
25300 pts/0    00:00:00 bash
25332 pts/0    00:00:00 ps
```

3. Use **top** to find how many processes are running and record your findings. Size the window as needed. Use the output to answer the following questions.

- Of these processes, how many are in a inactive state?
- How many processes are in the runnable state?
- How many processes are in a zombie state?
- How many are in the stopped state?

```
[student@servera ~]$ top
top - 16:45:15 up 40 min,  1 user,  load average: 0.23, 0.06, 0.02
Tasks: 119 total,   2 running, 117 sleeping,   0 stopped,   0 zombie
...output omitted...
```

Do not exit the **top** command.

4. Use **top** to record the time percentage the CPU spends for user and kernel processes.

```
top - 16:45:15 up 40 min,  1 user,  load average: 0.23, 0.06, 0.02
Tasks: 119 total,   2 running, 117 sleeping,   0 stopped,   0 zombie
%Cpu(s): 50.0 us,  0.0 sy,  0.0 ni, 50.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
...output omitted...
```

Do not exit the **top** command.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Using **top**, locate the process using the most physical memory and kill it.

5.1. Press **Shift+R** to sort the processes based on resident size (RES). Record the process ID listed under **PID** column for the process.

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
24895 student   20   0  234608   7676   4856 R  99.7   0.4   9:06.90 stress.py
24900 student   20   0  273596   4404   3748 R   0.3  99.7   9:06.89 top
...output omitted...
```

5.2. Press **k** from within the **top** interface to prompt for a PID to kill. Observe the prompt below the headers and above the columns. Ensure that the prompt defaults to the PID for the **stress.py** process at the top of the list. If not, enter the PID for the **stress.py** process at the prompt. Press **Enter** to kill the process.

```
...output omitted...
PID to signal/kill [default pid = 24895]
```

Press **Enter** again to confirm the SIGTERM signal 15.

```
...output omitted...
Send pid 24895 signal [15/sigterm]
```

Press **q** to exit the **top** command.

6. Identify the file that is filling up the **/dev/vdb1** disk.

6.1. Use **lsblk -fp** to list the UUID, file system, and mount point for **/dev/vdb1**.

```
[student@servera ~]$ lsblk -fp /dev/vdb
NAME        FSTYPE LABEL UUID                                 MOUNTPOINT
/dev/vdb
└─/dev/vdb1 xfs          18ea1c7c-b236-4009-9678-874c29781edc /opt
```

6.2. Use the **findmnt** command to verify that the **/opt** directory is mounted on **/dev/vdb1**, as well as the mount options for this directory.

```
[student@servera ~]$ findmnt -S /dev/vdb1
TARGET SOURCE    FSTYPE OPTIONS
/opt   /dev/vdb1 xfs    rw,relatime,seclabel,attr2,inode64,noquota
```

6.3. Use the **df -h** command to view the available disk space in GB for **/opt**.

```
[student@servera ~]$ df -h /opt
Filesystem      Size  Used Avail Use% Mounted on
/dev/vdb1       5.0G  5.0G   47M 100% /opt
```

6.4. Use the **du -h** command to identify the file that is causing the disk to fill. Use **sudo** to delete the file, and when prompted enter **student** as the password.

```
[student@servera ~]$ du -h /opt/*
4.9G  /opt/largefile
[student@servera ~]$ sudo rm /opt/largefile
[sudo] password for student: student
[student@servera ~]$
```

6.5. Logout from **servera**.

```
[student@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$
```

7. On **workstation**, use **gnome-system-monitor** to observe CPU usage behavior. Find and view the open files for a process.

7.1. On **workstation**, start two instances of the **sha1sum /dev/zero &** command.

```
[student@workstation ~]$ for i in $(seq 1 2); \
do sha1sum /dev/zero & done
[1] 14053
[2] 14054
```

7.2. Use the terminal to open the **gnome-system-monitor** utility.

```
[student@workstation ~]$ gnome-system-monitor
```

7.3. In **gnome-system-monitor**, click the **Resources** tab to view CPU usage. Both CPU1 and CPU2 are at 100%.

7.4. From the **Processes** tab, use the search button to find **sha1sum** processes.

Right-click on the first process and choose **Stop**. Click **Stop Process** to confirm.

Click the **Resources** tab to view CPU usage. CPU usage on one CPU is decreased.

7.5. Return to the **Processes** tab, right-click on either process, and choose **Open Files**. View the files used by the process, and then close the window.

7.6. Right-click and open **Memory Maps**. View the open files and then close the window.

7.7. Kill both **sha1sum** processes. Right-click on the process and choose **Kill**. Click **Kill Process** to confirm.

Close the **gnome-system-monitor** utility.

Finish

On the **workstation** VM, run the **lab perftools-system finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab perftools-system finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Describing the Sysstat Package Utilities

Objectives

After completing this section, you should be able to describe the *sysstat* architecture and explain typical *sysstat* utility command use.

Accessing the Sysstat Package Utilities

The *sysstat* package provides utilities for Linux system performance monitoring. The *sysstat* package utilities derive the raw data from the kernel counters, prepares them for display, and stores the data from the processes' execution history in a database. Some of these utilities store the information in a data file after collecting the data, and calculate the mean values; this makes it possible for the user to have access to the data of interest at predefined time intervals. The following is the list of utilities provided by the *sysstat* package:

`mpstat`
: Reports individual or combined CPU related statistics.

`iostat`
: Reports CPU statistics and I/O statistics for devices, partitions, and the network file system.

`pidstat`
: Reports statistics for Linux processes, including disk I/O, CPU and memory usage.

`tapestat`
: Reports statistics for tape drives connected to the system.

`cifsiostat`
: Reports statistics on shared file systems, printers, or serial ports over a network.

`sar`
: Collects, reports, and saves system activity information (such as CPU, memory, disks, and network interfaces usage statistics).

The *sysstat* package utilities accept non-mandatory options, such as **`interval`** and **`count`**. For example, in the screen below the **`iostat`** command returns data with a one second interval and an output count of three.

```
[user@host ~]$ iostat 1 3
Linux 4.18.0-80.el8.x86_64 (host.lab.example.com) ... _x86_64_ (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
❶          0.27    0.07    0.49    0.22    0.04   98.90
Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda              42.36       171.94       123.80     300358     216266
vdb               0.06         1.27         0.00       2224          0
vdc               0.06         1.27         0.00       2224          0
vdd               0.06         1.27         0.00       2224          0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
❷         0.00    0.00    1.00    1.50    0.50   97.00
```

```
Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             416.00         0.00      1040.00          0       1040
vdb               0.00         0.00         0.00          0          0
vdc               0.00         0.00         0.00          0          0
vdd               0.00         0.00         0.00          0          0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
❸       0.00    0.00    2.49    1.49    0.00   96.02
Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             432.00         0.00      1080.00          0       1080
vdb               0.00         0.00         0.00          0          0
vdc               0.00         0.00         0.00          0          0
vdd               0.00         0.00         0.00          0          0
```

❶ The first line of output of the *sysstat* package utilities is a summary since the most recent boot, which shows the mean for the entire time the system has been up.

❷❸ The subsequent lines are summaries based on the interval chosen by the user.

Features of the sysstat Package Utilities

Users have a choice to obtain system-wide information using other Linux system monitoring tools, such as the **`top`**, **`ps`**, and **`free`** commands. The utilities in the *sysstat* package provide the following features:

- Users can configure the length of statistical data history to be archived.
- Users can save statistical data in a file for future inspection, and can export in different file formats, such as CSV, JSON, and XML.
- Users can generate graphs using statistical data. For example, the **`isag`** command enables a user to plot data stored in a daily data file by a previous **`sar`** run.
- The *sysstat* package utilities automatically select the measurement units to display sizes for easier reading.
- The *sysstat* package utilities support hot-pluggable devices.

Monitoring CPU Usage

The **`mpstat`** command provides statistical data about the activities of each processor. All the data the **`mpstat`** command presents comes from **`/proc`**. For example, processor utilization and interrupts summary data is retrieved from the **`/proc/stat`** file, and the number of interrupts per seconds is retrieved from the **`/proc/interrupts`** file.

By default, the **`mpstat`** command prints only system-wide summaries. The following example prints a per-processor report with a one second interval between each report.

```
[user@host ~]$ mpstat -P ALL 1 3
Linux 4.18.0-80.1.2.el8_0.x86_64 (host.lab.example.com) ..._x86_64_(2 CPU)

01:03:22 PM  CPU    %usr   %nice    %sys ... %idle
01:03:23 PM  all   58.62    0.00   41.38 ...  0.00
01:03:23 PM    0   86.14    0.00   13.86 ...  0.00
01:03:23 PM    1   31.37    0.00   68.63 ...  0.00

01:03:23 PM  CPU    %usr   %nice    %sys ... %idle
01:03:24 PM  all   59.50    0.00   40.50 ...  0.00
01:03:24 PM    0   86.00    0.00   14.00 ...  0.00
01:03:24 PM    1   33.00    0.00   67.00 ...  0.00
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
01:03:24 PM  CPU    %usr   %nice    %sys ... %idle
01:03:25 PM  all   58.79    0.00   41.21 ...  0.00
01:03:25 PM    0   86.00    0.00   14.00 ...  0.00
01:03:25 PM    1   31.31    0.00   68.69 ...  0.00

Average:     CPU    %usr   %nice    %sys ... %idle
Average:     all   58.97    0.00   41.03 ...  0.00
Average:       0   86.05    0.00   13.95 ...  0.00
Average:       1   31.89    0.00   68.11 ...  0.00
```

Key Columns in mpstat Output

`CPU`
This column shows the logical processor ID; the entry **`all`** indicates the statistic calculated among all processors.

`%usr`
This column shows the percentage of CPU utilization to execute user-space processes.

`%sys`
This column shows the percentage of CPU utilization to execute kernel-space processes.

`%idle`
This column shows the percentage of time that the processors spent idle.

The **`-I`** option displays the statistics of the number of hardware and software interrupts, and interrupts received per second by each processor. The **`mpstat`** command with the **`-o JSON`** option displays the output in JSON format.

Monitoring Disk I/O Usage

The **`iostat`** command displays per-disk I/O statistics and provides metrics for workload characterization, utilization, and saturation. The performance administrators use the command to investigate disk I/O issues. The overhead of using the **`iostat`** command is negligible, as it uses the kernel counters for gathering statistics.

The **`iostat`** command, when executed with any arguments, displays both the CPU and disk I/O utilization metrics. The **`-c`** option displays the CPU utilization report, and the **`-d`** option displays the disk I/O utilization report. The **`-y`** option omits the first report with statistics since system boot. The **`-z`** option omits any devices for which there is no activity.

```
[user@host ~]$ iostat -dyz 1 3
Linux 4.18.0-80.el8.x86_64 (host.lab.example.com) ... _x86_64_ (2 CPU)

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             434.00         0.00      1085.00          0       1085

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             500.00         0.00      1427.50          0       1427

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             452.00         0.00      1132.00          0       1132
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Key Columns in iostat Output

`tps`
The column shows the transactions per second, referred to as IOPS.

`kB_read/s`
This column shows kilobytes read per second.

`kB_wrtn/s`
This column shows kilobytes written per second.

`kB_read`
This column shows the total kilobytes read.

`kB_wrtn`
This column shows the total kilobytes written.

The **`-x`** option displays extra columns, which include IOPS, as well as throughput metrics for performance characterization and latency analysis. Some the metrics displayed by the **`-x`** option are: average read and write request size, average I/O response time, and percentage of time the disk spent in processing I/O request.

Monitoring Process Usage

The **`pidstat`** command reports CPU, disk I/O, memory usage by processes or threads, including the percentage of CPU utilization breakdown into user-space and kernel-space. The command can also monitor the children of those processes and threads.

In the example below, **`pidstat`** displays three reports of CPU utilization for the **`23354`** process ID at one second intervals. The **`-p`** option allows a list of process IDs as arguments.

```
[user@host ~]$ pidstat -p 23354 1 3
Linux 4.18.0-80.el8.x86_64 (host.lab.example.com)  07/03/2019  _x86_64_  (2 CPU)

03:28:53 PM   UID    PID    %usr %system  %guest   %wait    %CPU  CPU  Command
03:28:54 PM  1000  23354   99.00    0.00    0.00    0.00   99.00    0  stress.py
03:28:55 PM  1000  23354  100.00    0.00    0.00    0.00  100.00    0  stress.py
03:28:56 PM  1000  23354  100.00    0.00    0.00    0.00  100.00    0  stress.py
Average:     1000  23354   99.67    0.00    0.00    0.00   99.67    -  stress.py
```

If no process ID is specified, **`pidstat`** displays all processes whose statistics change during the interval.

Options to Display Other Usage Statistics

`-t`
Displays per-thread statistics for processes.

`-d`
Displays disk statistics for processes. Only **`root`** can access statistics for processes owned by another user.

`-r`
Displays page fault and memory utilization for processes.

`-u`
Displays CPU utilization for processes. This is the default.

Monitoring Virtual Memory Usage

One of the most useful tools when troubleshooting memory-related performance issues is **`vmstat`**. The **`vmstat`** command is part of the *procps-ng* package, which includes other useful performance analysis commands such as **`free`** and **`top`**.

The **`vmstat`** command, if given no arguments, displays the averages of various system statistics since boot. The **`vmstat`** command accepts two arguments similar to the *sysstat* package utilities. The first of these arguments is the **`delay`** option, and the second is the **`count`** option.

The memory statistics are reported in KiB by default. The **`-S`** option enables changing the report to KB, MB, or MiB, using the options **`-S k`**, **`-S m`**, and **`-S M`**, respectively.

In the output below, the **`vmstat`** command displays three reports with a one second interval in reported in megabytes.

> **Important**
> The first two lines of output are header information, and the third line is the uptime average. The first line of **`vmstat`** output is an average since boot. When gathering metrics, ignore the first line and use the subsequent lines.

```
[user@host ~]$ vmstat -S m 1 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  0      0   1312      2    407    0    0     6   211  356  622  7  1 91  1  0
 1  0      0   1312      2    407    0    0     0  1185 3014 3447 50  2 48  1  0
 1  0      0   1312      2    407    0    0     0  1150 2957 3353 50  1 48  1  0
```

Overview of vmstat Columns

Category	Statistics	Definition
`process`	**`r`**	The number of processes waiting for a runtime.
	`b`	The number of processes in uninterruptible sleep.
`memory`	**`swpd`**	The amount of memory currently used in swap spaces.
	`free`	The amount of idle (immediately available) memory.
	`buff`	The amount of memory used as buffers.
	`cache`	The amount of memory used as the cache.
`swap`	**`si`**	Pages of memory swapped in per second.
	`so`	Pages of memory swapped out per second.
`io`	**`bi`**	Blocks per second received from block devices.
	`bo`	Blocks per second sent to block devices.
`system`	**`in`**	Interrupts raised per second.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Category	Statistics	Definition
	`cs`	Context switches per second.
`cpu`	`us`	Percentage of time spent running user-space code.
	`sy`	Percentage of time spent running kernel-space code.
	`id`	Percentage of time spent idle.
	`wa`	Percentage of time spent blocked while waiting for I/O to complete.
	`st`	Percentage of time where the CPU had a process ready to run, but CPU time was stolen by the hypervisor supporting this virtual machine (typically because the CPU is being used by another guest virtual machine).

Generating System Activity Reports

The System Activity Reporter, **`sar`**, monitors current system activity and can be configured to archive and report historical statistics. The **`sar`** command gathers the metrics from the kernel counter and has an agent that executes at scheduled times to record system activity. This command can also be used to collect instantaneous data about the state of the system.

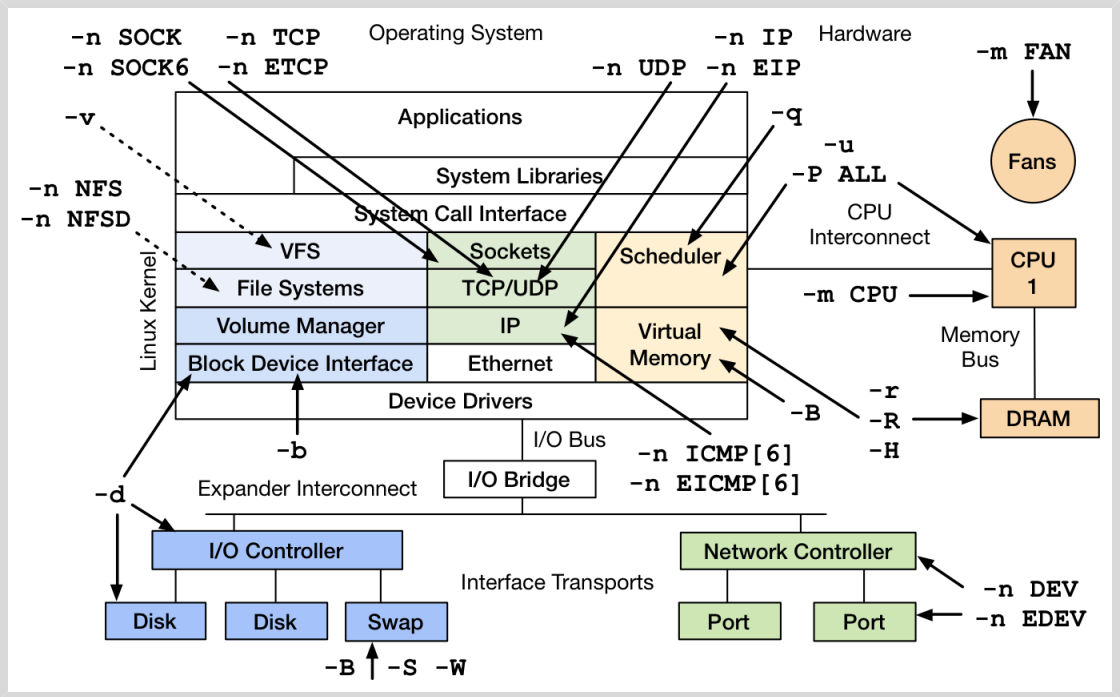


Figure 2.4: Linux Performance Observability: sar by Brendan Gregg (CC BY-SA 4.0)

In the output below, the **`sar`** command generates CPU utilization reports by default, and the **`-o`** option creates an output file to store the metrics. The command takes three samples at one second intervals, and simultaneously writes to both the output file and console.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[user@host ~]$ sar -o report.out 1 3
Linux 4.18.0-80.el8.x86_64 (host.lab.example.com)..._x86_64_(2 CPU)

09:06:41 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
09:06:42 PM     all    100.00      0.00      0.00      0.00      0.00      0.00
09:06:43 PM     all     99.01      0.00      0.99      0.00      0.00      0.00
09:06:44 PM     all     99.50      0.00      0.50      0.00      0.00      0.00
Average:        all     99.50      0.00      0.50      0.00      0.00      0.00
```

The above example uses the **sar** command interactively. To use it noninteractively, an agent is executed at a scheduled time using the **cron** command to collect data. A user configures the scheduled job in the **/etc/cron.d/sysstat** file as shown below, which runs the **/usr/lib64/sa/sa1** and **/usr/lib64/sa/sa2** commands. The **sa1** script invokes **/usr/lib64/sa/sadc** that collects data for a single one second measuring interval. By default, the **cron** command runs the **sa1** script every 10 minutes, and the **sa2** script writes a daily report to the **/var/log/sa/saDD** or the **/var/log/sa/saYYYMMDD** file, where **DD** is the current day, **MM** is the current month, and **YYYY** is the current year.

```
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib64/sa/sa1 1 1
# Generate a daily summary of process accounting at 23:53
53 23 * * * root /usr/lib64/sa/sa2 -A
```

Enable the **sysstat** service to start at system startup. The **sysstat** service sets all the metric counters to **0** by inserting a dummy record in the current daily data file to mark the time when the counters reset.

Common sar Command Options

-B
Reports memory paging statistics.

-b
Reports I/O and transfer statistics for all devices.

-d
Reports disk I/O statistics for each block device.

-n
Reports network statistics.

-r
Reports memory utilization statistics.

The **sar** command can be used to read one of the log files (or any **sar** data file in general) by using the **-f** option. In the screen below, the **sar -q -f** command displays a report of queue length and load averages to determine the saturation from the **/var/log/sa/sa03** data file.

```
[user@host ~]$ sar -q -f /var/log/sa/sa03
Linux 4.18.0-80.el8.x86_64 (host.lab.example.com) ..._x86_64_  (2 CPU)

09:35:00     LINUX RESTART  (2 CPU)

09:36:01 AM   runq-sz  plist-sz   ldavg-1   ldavg-5  ldavg-15   blocked
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
...output omitted...
10:08:01 AM         3       149      2.05      0.56      0.19         0
10:09:01 AM         4       149      2.84      1.06      0.39         1
Average:            2       148      0.16      0.07      0.03         0
```

Some configuration can be done in the **/etc/sysconfig/sysstat** file. The **/etc/sysconfig/sysstat** file is read by the **sa1** and **sa2** scripts and allows some configurable options for the **sar** command. For example, change the **HISTORY** variable to set the number of days archives are kept for the **sar** command.

Using Systemd Timer

With the advent of **systemd** in Red Hat Enterprise Linux 7, a new scheduling function is now available: **systemd** *timer units*. A **systemd** timer unit activates another unit of a different type (such as a service) whose unit name matches the timer unit name. The timer unit allows timer-based activation of other units. For easier debugging, **systemd** logs timer events in system journals.

Sample Timer Unit

The *sysstat* package provides a **systemd** timer unit called **sysstat-collect.timer** to collect system statistics every 10 minutes. The following output shows the configuration lines of **/usr/lib/systemd/system/sysstat-collect.timer**.

```
...output omitted...
[Unit]
Description=Run system activity accounting tool every 10 minutes

[Timer]
OnCalendar=*:00/10

[Install]
WantedBy=sysstat.service
```

The parameter **OnCalendar=*:00/10** signifies that this timer unit activates the corresponding unit (**sysstat-collect.service**) every 10 minutes.

> **Important**
> Do not modify any unit configuration file under the **/usr/lib/systemd/system** directory because any update to the provider package of the configuration file may override the configuration changes you made in that file. So, make a copy of the unit configuration file you intend to change under the **/etc/systemd/system** directory and then modify the copy so that the configuration changes you make with respect to a unit does not get overridden by any update to the provider package. If two files exist with the same name under the **/usr/lib/systemd/system** and **/etc/systemd/system** directories, **systemd** parses the file under the **/etc/systemd/system** directory.

After you change the timer unit configuration file, use the **systemctl daemon-reload** command to ensure that **systemd** is aware of the changes. This command reloads the **systemd** manager configuration.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# systemctl daemon-reload
```

After you reload the **systemd** manager configuration, use the following **systemctl** command to activate the timer unit.

```
[root@host ~]# systemctl enable --now <unitname>.timer
```

References

mpstat(1), **iostat**(1), **pidstat**(1), **vmstat**(8), **sar**(1), **sa1**(8), **sa2**(8), **sadc**(8), **sysstat**(5), **systemd.time**(7), and **systemd.timer**(5) man pages

Welcome to the SYSSTAT Utilities Home Page!
http://sebastien.godard.pagesperso-orange.fr/

The original graphic by Brendan Gregg can be found at
Linux Performance Observability: sar
http://www.brendangregg/linuxperf.html/
It is licensed under Attribution-ShareAlike 4.0 International
https://creativecommons.org/licenses/by-sa/4.0/legalcode

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Viewing the Sysstat Package Utilities

In this exercise, you will analyze the output of *sysstat* tool commands.

Outcomes

You should be able to:

- Interpret the output of the **`vmstat`**, **`mpstat`**, **`iostat`**, and **`pidstat`** commands.
- Configure **`sar`** on **`servera`** with an updated cron job, and then generate reports.

Before You Begin

Log in to the **`workstation`** VM as **`student`**, using **`student`** as the password.

Run the **`lab perftools-sysstat start`** command to set up the environment for the guided exercise.

```
[student@workstation ~]$ lab perftools-sysstat start
```

Note

When instructed to use the **`sudo`** command for privilege escalation, use **`student`** as the password, if prompted.

1. On the **`workstation`** VM, open two terminal windows side by side. On each terminal, use the **`ssh`** command to log in to **`servera`** as the student user.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$
```

2. On the first terminal, use the **`sudo -i`** command to switch to the **`root`** user's account.

```
[student@servera ~]$ sudo -i
[root@servera ~]#
```

3. Install the *sysstat* package. The sysstat package provides the systemd units called sysstat-collect.timer and sysstat-collect.service. Modify the timer unit configuration file to collect the system activity data once every minute.

Important

Do not edit files under the **`/usr/lib/systemd`** directory. Instead, copy the unit file to the **`/etc/systemd/system`** directory and edit that copy. The **`systemd`** process locates and uses configurations files in **`/etc/systemd/system`** first.

3.1. Install the *sysstat* package.

```
[root@servera ~]# yum install sysstat
...output omitted...
```

3.2. Copy **/usr/lib/systemd/system/sysstat-collect.timer** to **/etc/systemd/system/sysstat-collect.timer**.

```
[root@servera ~]# cp /usr/lib/systemd/system/sysstat-collect.timer \
/etc/systemd/system/sysstat-collect.timer
```

3.3. Edit **/etc/systemd/system/sysstat-collect.timer** so that the timer unit runs once every minute.

```
[root@servera ~]# vim /etc/systemd/system/sysstat-collect.timer
...output omitted...
#        Activates activity collector once every minute

[Unit]
Description=Run system activity accounting tool once every minute

[Timer]
OnCalendar=*:00/01

[Install]
WantedBy=sysstat.service
```

3.4. Use the **systemctl daemon-reload** command to make sure that **systemd** is aware of the changes.

```
[root@servera ~]# systemctl daemon-reload
```

3.5. Use the **systemctl** command to activate the **sysstat-collect.timer** timer unit.

```
[root@servera ~]# systemctl enable --now sysstat-collect.timer
```

4. On the first terminal, exit the **root** user's account to return as **student** user on **servera**.

```
[root@servera ~]# exit
logout
[student@servera ~]$
```

5. Run the **vmstat** command with a one second delay between updates. Let it run for approximately 10 seconds. Notice the CPU percentage of time spent in idle by looking at the **id** column. Output will vary between systems and command runs.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@servera ~]$ vmstat 1
procs ... ------cpu-----
 r  b ... us sy id wa st
 1  0 ...  3  0  96 0  0
 0  0 ...  0  0 100 0  0
 0  0 ...  0  0 100 0  0
 ...output omitted...
```

Press **Ctrl**+**C** to exit the **vmstat** command.

6. Execute the **/home/student/stress.py** script to generate load, and then use **vmstat** to observe the CPU usage. How would you get the system to be 100% utilized with the CPU no longer idle?

   Use the **vmstat** command to print reports with a one second interval.

   6.1. Execute the **/home/student/stress.py** script as a background process, and rerun the **vmstat** command.

```
[student@servera ~]$ ./stress.py &
[1] 24074
[student@servera ~]$ vmstat 1
 procs ... ------cpu-----
 r  b ... us sy id wa st
 4  0 ...  3  0 96  0  0
 1  0 ... 50  0 50  0  0
 1  0 ... 50  0 50  0  0
```

Observe that now the CPU percentage of time spent idle (**id**) and running user-space code (**us**) is around 50%.

   6.2. In the second terminal, execute the **/home/student/stress.py** script as a background process.

```
[student@servera ~]$ ./stress.py &
[1] 24076
```

```
 procs ... ------cpu-----
 r  b ... us sy id wa st
 2  0 ... 100  0  0  0  0
 2  0 ... 100  0  0  0  0
...output omitted...
```

In the first terminal, observe the **vmstat** command. The percentage of time spent idle (**id**) and running user-space code (**us**) is around 100%, as the process is using both the CPUs.

Press **Ctrl**+**C** to exit the **vmstat** command.

   6.3. In both terminals, kill the **stress.py** process.

```
[student@servera ~]$ pkill stress.py
[1]+  Terminated              ./stress.py
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

7. Execute the **/home/student/stress01.py** script, and then run **vmstat** and **mpstat** to check the CPU usage.

7.1. In the first terminal, execute the **/home/student/stress01.py** script as a background process.

```
[student@servera ~]$ ./stress01.py &
[1] 24079
```

7.2. In the first terminal, run the **vmstat** command with a one second delay between updates, and display four updates.

```
[student@servera ~]$ vmstat 1 4
procs ... ------cpu-----
 r  b ... us sy id wa st
...output omitted...
```

7.3. From the second terminal, run the **mpstat** command with a one second delay between updates, and display four updates.

```
[student@servera ~]$ mpstat -P ALL 1 4
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com)  ...  _x86_64_  (2 CPU)

08:35:31 PM  CPU    %usr   ...   %idle
...output omitted...
```

7.4. Observe the CPU metrics for both the **vmstat** and **mpstat** commands.

The **mpstat** command provides the same CPU utilization statistics as the **vmstat** command, but **mpstat** displays the statistics on a per-processor basis. In the below output of the **mpstat** command, the **stress01.py** process uses **CPU0** for processing, as the percentage of time spent running user-space code is around 100%. The fact that the process is running on **CPU0** might vary in your environment.

```
 procs ... ------cpu-----
 r  b ... us sy id wa st
 3  0 ...  4  0 95  0  0
 1  0 ... 50  1 50  0  0
 1  0 ... 50  0 50  0  0
...output omitted...
```

```
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com)  ...  _x86_64_  (2 CPU)

...          CPU    %usr   ...   %idle
...          all   49.75   ...   49.75
...            0   99.01   ...    0.00
...            1    0.00   ...  100.00
...output omitted...
...
...          CPU    %usr   ...   %idle
...          all   50.00   ...   50.00
...            0  100.00   ...    0.00
...            1    0.00   ...  100.00
```

```
Average:     CPU    %usr   ...   %idle
Average:     all   49.88   ...   49.88
Average:       0   99.75   ...    0.00
Average:       1    0.00   ...   99.75
```

7.5. Kill the **stress01.py** process.

```
[student@servera ~]$ pkill stress01.py
[1]+  Terminated              ./stress01.py
```

8. Execute the **/home/student/dummy.sh** script and monitor the behavior of the system with the **iostat** command while running the script. Inspect the script code to see what it does.

8.1. From the first terminal, execute the **/home/student/dummy.sh** script as a background process.

```
[student@servera ~]$ ./dummy.sh &
[1] 1276
```

8.2. Use the **iostat** command to monitor the disk I/O statistics with a one second delay between each of four updates. Use the **-y** option to ignore the first report, which provides statistics concerning the time since the system was booted. Also, use the **-z** option to omit output for any devices on which there is no activity.

```
[student@servera ~]$ iostat -yz 1 4
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com) ... _x86_64_ (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00    0.05

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             410.00         0.00      1129.00          0       1129

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    1.49    1.49    0.50   96.52

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             396.00         0.00      1087.50          0       1087

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    1.51    1.01    0.00   97.49

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             404.00         0.00      1112.50          0       1112

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    1.51    1.51    0.50   96.48

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda             400.00         0.00      1100.00          0       1100
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Observe the **tps**, **kB_read/s**, and **kB_wrtn/s** columns in the output. The amount of data read from the device in kilobytes per second is zero, which indicates there are no read transactions.

8.3. Kill the **dummy.sh** process.

```
[student@servera ~]$ kill -- -$(pgrep dummy.sh)
[student@servera ~]$
```

8.4. Open the **/home/student/dummy.sh** file and inspect the script. The script writes a 100 MB file in an infinite loop.

```
[student@servera ~]$ cat dummy.sh
#!/bin/bash

# Tell bash to exit after SIGINT
trap exit INT

# Write a 100MB file every loop
while true; do
  dd if=/dev/zero of=100MBfile bs=512 count=200000 oflag=dsync
done
```

9. Execute the **/home/student/stress.py** script and run the **pidstat** command to get the CPU usage for the process. Also, view the disk I/O statistics.

9.1. From the first terminal, execute the **/home/student/stress.py** script as a background process.

```
[student@servera ~]$ ./stress.py &
[1] 1578
```

9.2. Use the **pidstat** command to monitor the CPU usage for the **stress.py** process with a one second delay between two updates.. Use the **-p** option to pass the process ID of the **stress.py** process.

Use the **pgrep stress.py** command to retrieve the process ID. In the output below, the process ID is **1578**.

```
[student@servera ~]$ pgrep stress.py
1578
[student@servera ~]$ pidstat -p 1578 -u 1 2
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com) ... _x86_64_ 2 CPU)

...       UID   PID    %usr %system  %guest   %wait    %CPU   CPU  Command
...      1000  1578   99.00    0.00    0.00    0.00   99.00     0  stress.py
...      1000  1578  100.00    0.00    0.00    0.00  100.00     0  stress.py
Average:1000  1578   99.50    0.00    0.00    0.00   99.50     -  stress.py
```

9.3. Use the **pidstat -d** command to view the disk I/O statistics and display two reports for the **stress.py** process.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@servera ~]$ pidstat -p 1578 -d 1 2
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com) ... _x86_64_ (2 CPU)

...       UID  PID kB_rd/s kB_wr/s kB_ccwr/s iodelay  Command
...      1000 1578    0.00    0.00      0.00       0  stress.py
...      1000 1578    0.00    0.00      0.00       0  stress.py
Average:1000 1578    0.00    0.00      0.00       0  stress.py
```

9.4. Kill the **stress.py** process.

```
[student@servera ~]$ pkill stress.py
[1]+  Terminated              ./stress.py
```

10. Report the I/O and transfer rate statistics, utilization of CPU0, and the network device statistics from the current log file. Output will vary between systems and command runs. Allow additional time for the activity collector to record data.

10.1. Use the **sar -b** command to report the I/O and transfer rate statistics from the current log file.

```
[student@servera ~]$ sar -b
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com)  ...  _x86_64_  (2 CPU)

15:50:54     LINUX RESTART  (2 CPU)

03:55:01 PM       tps      rtps      wtps   bread/s   bwrtn/s
03:56:01 PM      1.65      1.48      0.17    125.64      7.39
03:57:01 PM    352.26      0.04    352.23      1.19   1937.10
03:58:01 PM    400.27      0.01    400.26      0.71   2158.94
03:59:01 PM    406.37      0.00    406.37      0.00   2032.29
04:00:01 PM    629.46      0.01    629.45      0.24   2817.26
04:01:01 PM    454.50      0.03    454.48      0.96   2102.45
04:02:01 PM    415.62      0.01    415.61      0.69   2079.67
04:03:01 PM      0.03      0.00      0.03      0.00      0.36
04:04:01 PM      0.26      0.12      0.14     15.19      1.99
Average:       314.59      0.19    314.40     16.12   1553.52
```

10.2. Use the **sar -P 0** command to report the utilization of **CPU0** for the current log file.

```
[student@servera ~]$ sar -P 0
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com)  ...  _x86_64_  (2 CPU)

15:50:54     LINUX RESTART  (2 CPU)

...      CPU %user %nice %system %iowait %steal %idle
...        0  0.01  0.08    0.03    0.05   0.01 99.83
...        0  0.03  0.00    1.59    1.05   0.41 96.92
...        0  0.03  0.00    1.67    1.07   0.46 96.77
...        0  0.04  0.00    1.69    1.03   0.48 96.76
...        0  0.07  0.00    2.29   33.53   0.46 63.65
...        0 15.12  0.00    2.09   13.96   0.40 68.43
...        0 81.08  0.00    0.95    0.17   0.09 17.71
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
...        0  0.00  0.00    0.01    0.00   0.01 99.99
...        0  0.02  0.00    0.04    0.01   0.01 99.93
Average:   0 11.61  0.01    1.22    5.99   0.27 80.90
```

10.3. Use the **`sar -n DEV`** command to report the network device statistics from the current log file.

```
[student@servera ~]$ sar -n DEV
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com)  ...  _x86_64_  (2 CPU)

15:50:54     LINUX RESTART  (2 CPU)

...  IFACE  rxpck/s txpck/s rxkB/s txkB/s rxcmp/s ... %ifutil
...     lo     0.00    0.00   0.00   0.00    0.00 ...    0.00
... enp1s0     0.91    0.31   0.07   0.05    0.00 ...    0.00
...     lo     0.00    0.00   0.00   0.00    0.00 ...    0.00
... enp1s0     1.35    0.73   0.11   0.16    0.00 ...    0.00
...     lo     0.00    0.00   0.00   0.00    0.00 ...    0.00
... enp1s0     1.07    0.53   0.07   0.13    0.00 ...    0.00
...output omitted...
Average:lo     0.00    0.00   0.00   0.00    0.00 ...    0.00
Average:enp1s0 1.40    0.70   0.11   0.16    0.00 ...    0.00
```

10.4. Logout from both **`servera`** terminal sessions to return to **`workstation`** as **`student`** user.

```
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On the **`workstation`** VM, run the **`lab perftools-sysstat finish`** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab perftools-sysstat finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Collecting Performance Data with Performance Co-Pilot

Objectives

After completing this section, you should be able to describe the Performance Co-Pilot utility and its standardized data structures.

Performance Co-Pilot Overview

Red Hat Enterprise Linux 8 includes Performance Co-Pilot (PCP), a suite of tools, services, and libraries for monitoring, visualizing, storing, and analyzing system level performance measurements. PCP provides the framework for monitoring and management of real-time data, as well as logging and retrieval of historical data. Its lightweight distributed architecture makes PCP ideal for centralized analysis of both local and remote hosts.

The PCP architecture is distributed in the sense that any PCP tool may be executing remotely. On the host (or hosts) being monitored, each domain of performance metrics, whether the kernel, a service layer, web server, or other, requires a Performance Metrics Domain Agent (PMDA) that is responsible for collecting performance measurements from that domain. All PMDAs are controlled by the Performance Metrics Collector Daemon (PMCD) on the same host.

Client applications (the monitoring tools) connect to **`pmcd`** daemon, which acts as a router for requests by forwarding requests to the appropriate PMDA and returning the responses to the clients. Clients also access performance data from sets of PCP archives (created using the **`pmlogger`** command) for retrospective analysis.

While reading through this section, it is recommended that you make a comparison of PCP tools to legacy tools. There is almost a one-to-one matching between what the PCP and legacy tools can do. An advantage of PCP is that it leverages a universal storage structure for use with many newer tools, while retaining functionality similar to the legacy tools.

In this section, you will learn PCP commands that accomplish functions similar to those performed by the legacy commands presented earlier in this chapter. Compare the two command versions for capability, efficiency, and functionality. See the reference links at the end of this section for additional comparisons.

Installing Performance Co-Pilot

Performance Co-Pilot is installed with the *pcp* package. The *pcp* package provides several basic tools and libraries, including the **`pmcd`** collector daemon and the **`pcp`** command and subcommands.

```
[user@host ~]$ sudo yum install pcp
```

To generate PCP graphical views, the *pcp-gui* package must be installed. The *pcp-gui* package provides additional utilities, such as the **`pmchart`** command, to generate graphical data of local and remote performance metrics.

```
[user@host ~]$ yum install pcp-gui
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Starting and Enabling Performance Co-Pilot

PCP uses the **pmcd** daemon collector to collect performance metrics on a system. There must be an instance of **pmcd** running on all systems that provide performance metrics to PCP.

```
[user@host ~]$ sudo systemctl enable --now pmcd
```

Installing Additional System Monitoring Tools

The *pcp-system-tools* package provides an extended set of PCP tools and subcommands that are passed as arguments to the **pcp** command. Included in this package are the PCP versions of legacy commands, such as **iostat**, **pidstat**, **free**, and others.

```
[user@host ~]$ sudo yum install pcp-system-tools
```

Gathering and Displaying Performance Data

The *pcp* and *pcp-system-tools* packages provides a variety of command-line utilities to gather and display performance data.

The **pcp free** command provides a summary of the total amount of free and used physical memory and swap in the system, as well as the caches used by the kernel.

```
[user@host ~]$ pcp free
             total       used       free     shared    buffers     cached
Mem:       1872932     961280     911652          0     109112     545176
-/+ buffers/cache:     306992    1565940
Swap             0          0          0
```

The **pcp-dstat** command is used as a general performance analysis tool. This command, **pcp-dstat**, enables viewing multiple system resources in the same interval. For example, you can compare disk usage in combination with interrupts from a disk controller, or compare the network bandwidth numbers directly with the disk throughput.

When no arguments are provided to the **pcp dstat** command, the default options **-cdngy** are used. The default options are equivalent to **--cpu**, **--disk**, **--net**, **--page**, and **--sys** respectively.

```
[user@host ~]$ pcp dstat
You did not select any stats, using -cdngy by default.
----total-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai stl| read  writ| recv  send|  in   out | int   csw
  0   0 100   0   0|   0     0 | 622   648 |   0     0 |  90   150
  0   0 100   0   0|   0     0 | 570  1144 |   0     0 |  82   147
  0   0 100   0   0|   0     0 |  66   102 |   0     0 |  55   109
  1   1  94   4   1|   0  1833k| 388   750 |   0     0 |1576  5219
  1   2  93   3   1|   0  1931k| 220   528 |   0     0 |1654  5536
  1   2  93   4   0|   0  2125k|  66    66 |   0     0 |1812  6085
  1   2  93   4   1|   0  2021k| 118   358 |   0     0 |1733  5794
  1   2  94   4   1|   0  1911k|  66   358 |   0     0 |1637  5473
  1   1  93   3   1|   0  2004k| 118   358 |   0     0 |1708  5737
  1   2  94   4   1|   0  2271k|  66   358 |   0     0 |1925  6507
```

Other useful options include adding a time stamp to each line of output, adding a delay between each update, and a count of updates to display. The following command displays full word options for a timestamp, with CPU and process statistics reported every two seconds for 8 intervals.

```
[user@host ~]$ pcp dstat --time --cpu --proc 2 8
----system---- ----total-usage---- ---procs---
     time     |usr sys idl wai stl|run blk new
08-07 23:33:57|                   |2.0 1.0
08-07 23:33:59| 52   1  45   3   0|1.0   0   0
08-07 23:34:01| 51   3  45   3   1|1.0   0 2.0
08-07 23:34:03| 51   1  44   4   1|1.0   0   0
08-07 23:34:05| 50   1  44   3   0|1.0   0   0
08-07 23:34:07| 50   1  44   3   0|1.0   0   0
08-07 23:34:09| 51   1  45   3   1|1.0   0   0
08-07 23:34:11| 51   1  44   3   0|1.0   0   0
```

The **pmstat** command provides information similar to **vmstat**. Like **vmstat**, **pmstat** supports options to adjust the collection interval (**-t**) and the number of samples (**-s**).

```
[user@host ~]$ pmstat -s 5
@ Sun Jul  7 21:09:45 2019
 loadavg                      memory      swap        io    system         cpu
   1 min   swpd   free   buff  cache   pi   po   bi   bo   in   cs  us  sy  id
    0.08      0 919832 109112 544940    0    0    0 819K  584  673   0  12  88
    0.15      0 919700 109112 544912    0    0    0 655K  502  574   0  10  90
    0.22      0 919652 109112 544844    0    0    0 819K  574  680   0  13  87
    0.28      0 919516 109112 544916    0    0    0 655K  485  575   0  10  90
    0.34      0 920548 109112 544916    0    0    0 614K  437  522   0   9  91
```

Another convenient utility is **pmcollectl**. This utility is a Python port of a popular open source storage statistics application called **collectl**.

```
[user@host ~]$ pmcollectl -c 5
#<--------CPU--------> <----------Disks-----------> <----------Network--------->
#cpu sys inter  ctxsw KBRead  Reads KBWrit Writes KBIn  PktIn  KBOut  PktOut
   1   0    97    122     0      0      0      0    0      2      0      1
   2   1   178    210     0      0    184     13   23     16      2     16
   1   0   161    190     0      0     96      1   23     16      2     16
   1   0   176    199     0      0      0      0   24     25      3     24
   2   1   102    134     0      0      0      0    0      1      0      1
```

Performance Co-Pilot has a text-based mechanism for querying individually tracked metrics. To obtain a list of metrics stored in the Performance Co-Pilot database, use the **pminfo** command. Use the **pmval** command with the metric to gather related data.

```
[user@host ~]$ pminfo
...output omitted...
kernel.all.nprocs
proc.nprocs
hotproc.nprocs
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[user@host ~]$ pminfo -dt proc.nprocs

proc.nprocs [instantaneous number of processes]
    Data Type: 32-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: instant  Units: none

[user@host ~]$ pmval -s 5 proc.nprocs

metric:    proc.nprocs
host:      host.example.com
semantics: instantaneous value
units:     none
samples:   5
interval:  1.00 sec
        129
        129
        129
        129
        129
```

Plotting Performance Metric Data Using a Graphic Utility

The *pcp-gui* package provides a **pmchart** graphical utility that plots performance metric values sourced from multiple live hosts. Local and remote hosts can be displayed in the same view. One or more sets of PCP archives can be used as a source of historical data.

The **pmchart** command graphs live data and data from other machines, and can replay data from archived data files.

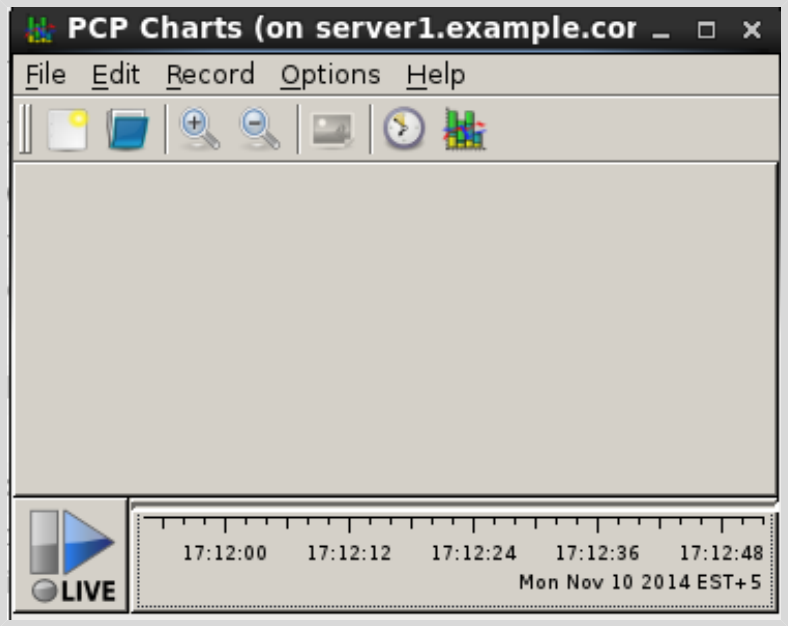


Figure 2.5: The pmchart view without any charts configured

To add a chart to **pmchart**, select **New Chart...** from the **File** menu.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

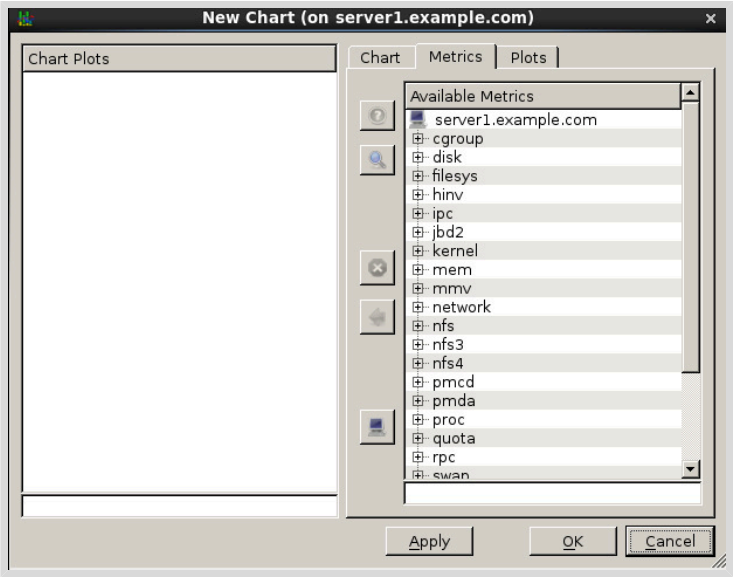


Figure 2.6: The pmchart Available Metrics item selection

The **`pmchart`** command offers a variety of available metrics for administrators to graph. Administrators can configure multiple graphs to display in the same **`pmchart`** window.

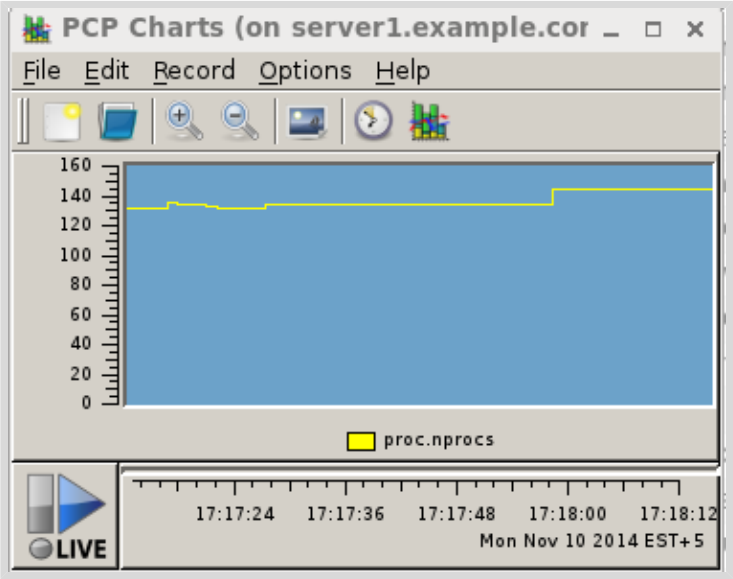


Figure 2.7: Graph displaying the number of active processes (nprocs)

By default, **`pmchart`**'s graph data updates every second. The interval can be modified on each independent graph.

Using PCP Chart to Access Remote Hosts

To access performance metrics from remote hosts:

- Install the *pcp* package on local and remote hosts.
- Grant access to port **`44321/tcp`** on each remote host.
- Start and enable the **`pmcd`** daemon on local and remote hosts.
- Install the *pcp-system-tools* package on any host requiring system monitoring tools.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

By default, **pmchart** will connect to the local host **pmcd** daemon. To connect to a remote host, provide the host name on the **pmchart** command. The following command opens a connection from the local host to the **pmcd** daemon on **remotehost**.

```
[user@host ~]$ pmchart -h remotehost
```

Alternatively, you can add a properly configured remote host to a running PCP Chart interface:

- From the **PCP Charts** interface, click on the **Edit** menu, select **Preferences ...**.
- Open the **Edit** menu, and then select **Preferences ...**.
- In the **Settings** interface, click on the **Hosts** tab.
- Click the **Add Host** computer icon.
- Enter the remote host name in the **Hostname:** field. Click the **OK** button, then the **Close** button.
- To verify the additional host, click the **New Chart** icon from the **PCP Charts** interface. Click on the **Metrics** tab to display a list of hosts connected to **pmcd** daemons on their respective hosts.

Replaying data from the log

Performance Co-Pilot also has the ability to store data in a log using the **pmlogger** command. By default, the **pmlogger** command stores logged data in the **/var/log/pcp/pmlogger/*HOSTNAME*** directory.

After the data has been collected in a **pmlogger** archive, tools, such as **pmval** or **pmchart**, are used to query and process data based on a select performance metric.

The **pmlogger** command may be started as a daemon by the **pcp** init script. It may also be started manually. By default, **pmlogger** collects data every second.

```
[user@host ~]$ pmlogger /var/log/pcp/pmlogger/host.example.com
```

In the previous example, a log was created in the **/var/log/pcp/pmlogger/host.example.com** directory. The log file name begins with the ISO formatted date. In addition to the log file, several other files are created to store metadata and index information.

Use the **pmdumplog** command to view internal details of a performance metrics archive log. Add the **-Z EST+5** option to ensure that the time stamp displays in the same format as the time stamp of the archive, essentially overriding the timezone of the local system.

```
[user@host ~]$ pmdumplog -L -Z EST+5 \
/var/log/pcp/pmlogger/host.example.com/20190708.12.24.0
Note: timezone set to "TZ=EST+5"

Log Label (Log Format Version 2)
Performance metrics from host host.example.com
    commencing Mon Jul 08 12:24:41.562205 2019
    ending     Mon Jul 08 12:54:41.500123 2019
Archive timezone: EST+5
PID for pmlogger: 9217
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Note

When exporting the log to another system, copy the log file and associated **.meta** files. If any files is missing, then the analysis tools are unable to read the log data.

Once the log has been created, command line **pcp** tools use the **-a** option to indicate that they should be run against a data archive instead of live data. The **pmval** and **pmchart** commands have additional options used to specify the start and end times, which should be used to narrow the data to a specific time span.

```
[user@host ~]$ pmval -a /var/log/pcp/pmlogger/host.example.com/20150224.00.10.0
 kernel.all.load
...output omitted...
03:03:46.197       0.1100        0.1700        0.1200
03:03:47.197       0.1100        0.1700        0.1200
03:03:48.197       0.1100        0.1700        0.1200
03:03:49.197       0.1100        0.1700        0.1200
...output omitted...
```

Use the **pmval** command to view the load averages from 03:03:00 until 03:04:00 on July 07, 2019:

```
[user@host ~]$ pmval -a /var/log/pcp/pmlogger/host.example.com/20190707.00.10.0
 kernel.all.load -S '@ Sun Jul 07 03:03:00 2019' -T '@ Sun Jul 07 03:04:00 2019'
metric:    kernel.all.load
archive:   /var/log/pcp/pmlogger/host.example.com/20190707.00.10.0
host:      host.example.com
start:     Sun Jul 07 03:03:00 2019
end:       Sun Jul 07 03:04:00 2019
semantics: instantaneous value
units:     none
samples:   61
interval:  1.00 sec

                 1 minute      5 minute     15 minute
03:03:00.000       0.3000        0.2100        0.1200
03:03:01.000       0.3000        0.2100        0.1200
03:03:02.000       0.3000        0.2100        0.1200
...output omitted...
```

Alternate time specifications can be found in the **PCPIntro**(1) man page.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

`pmstat`(1), **`pmcollectl`**(1), **`pmatop`**(1), **`pminfo`**(1), **`pmval`**(1), and **`pmchart`**(1) man pages

Performance Co-Pilot (PCP) Data Sheet
https://access.redhat.com/articles/3119481

Side-by-side comparison of PCP tools with legacy tools
https://access.redhat.com/articles/2372811

How does Performance Co-Pilot (PCP) compare with sysstat
https://access.redhat.com/articles/1145953

Index of Performance Co-Pilot (PCP) articles, solutions, tutorials and white papers
https://access.redhat.com/articles/1145953

Guided Exercise

Collecting Performance Data with Performance Co-pilot

In this exercise, you will gather and store performance metrics using the Performance Co-Pilot.

Outcomes

You should be able to install Performance Co-Pilot packages and demonstrate use of tools.

Before You Begin

Log in as the **student** user on the **workstation** VM, using **student** as the password.

On the **workstation** VM, run the **lab perftools-copilot start** command. This command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab perftools-copilot start
```

Note

When instructed to use the **sudo** command for privilege escalation, use **student** as the password.

1. On the **workstation** VM, use the **ssh** command to log in to **servera** as the **student** user. The systems are configured to use SSH keys for authentication, therefore a password is not required to log in to **servera**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$
```

2. On **servera**, install the necessary packages that provide the basic and extended sets of PCP tools. Start and enable the Performance Metrics Collector Daemon.

   2.1. Install the *pcp* and *pcp-system-tools* packages.

```
[student@servera ~]$ sudo yum install pcp pcp-system-tools
[sudo] password for student: student
...output omitted...
Complete!
```

   2.2. Start and enable the **pmcd** daemon.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@servera ~]$ sudo systemctl enable --now pmcd
...output omitted...
```

3. Gather information using the **pcp** command with subcommands that compare to legacy tools.

3.1. Use the **pcp** command without subcommand arguments to summarize the Performance Co-Pilot configuration.

```
[student@servera ~]$ pcp
Performance Co-Pilot configuration on servera.lab.example.com:

 platform: Linux servera.lab.example.com 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13
 12:02:46 UTC 2019 x86_64
 hardware: 2 cpus, 4 disks, 1 node, 1829MB RAM
 timezone: CDT+5
 services: pmcd
     pmcd: Version 4.3.0-1, 8 agents
     pmda: root pmcd proc xfs linux mmv kvm jbd2
```

3.2. Use the **pcp free** command to display memory statistics.

```
[student@servera ~]$ pcp free
             total     used     free  shared  buffers  cached
Mem:       1872932   614300  1258632       0     2104  356424
-/+ buffers/cache:   255772  1617160
Swap             0        0        0
```

3.3. Use the **pcp dstat** command to display detailed information related to disk usage for **vda**.

Use the **dd** command, in the background, to generate disk writes to the **/home/student/diskreads** file. Use the **pcp dstat** command to display a time stamp and disk statistics that collect data every second and display 10 iterations of output.

```
[student@servera ~]$ dd if=/dev/zero of=diskreads bs=512 \
count=2M oflag=dsync &
```

```
[student@servera ~]$ pcp dstat --nocolor -tdD vda 1 10
----system---- --dsk/vda--
     time     | read  writ
28-06 08:10:52|
28-06 08:10:53|   0   111M
28-06 08:10:54|   0   128M
28-06 08:10:55|   0   112M
28-06 08:10:56|   0   112M
28-06 08:10:57|   0   111M
28-06 08:10:58|   0     0
28-06 08:10:59|   0     0
28-06 08:11:00|1838k    0
28-06 08:11:01|   0    10k
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Use the **`kill $(pidof dd)`** command to terminate the **`dd`** command.

```
[student@servera ~]$ kill $(pidof dd)
[student@servera ~]$
```

4. Use the **`pmstat`** command to capture one minute load averages, swapping statistics, system interrupt rates and context switching, CPU user code, system and interrupt processing code, and idle loop statistics.

   Execute four instances of the **`sha1sum /dev/zero &`** command in a loop structure to apply workload to the system, followed immediately by the **`pmstat`** command. Observe the change in values for approximately 15 iterations. Note that PID and statistic values vary between command executions.

```
[student@servera ~]$ for i in $(seq 1 4); do sha1sum /dev/zero & done
[1] 2154
[2] 2155
[3] 2156
[4] 2157
```

```
[student@servera ~]$ pmstat
@ Sun Jun 30 15:54:33 2019
 loadavg                  memory    swap      io     system         cpu
   1 min swpd   free buff  cache  pi  po  bi  bo    in   cs  us  sy  id
    0.70    0  1229m 2104 364200   0   0   0   0  2017  307  95   5   0
    0.96    0  1229m 2104 364200   0   0   0   0  2016  305  96   4   0
    1.20    0  1229m 2104 364200   0   0   0   0  2014  304  95   5   0
    1.59    0  1229m 2104 364200   0   0   0   0  2015  307  95   5   0
    1.94    0  1229m 2104 364200   0   0   0   0  2013  306  95   5   0
    2.11    0  1220m 2104 364220   0   0   0   0  2069  404  95   5   0
    2.34    0  1228m 2104 364212   0   0   0  27  2099  458  95   5   0
    2.55    0  1228m 2104 364212   0   0   0   0  2014  304  96   4   0
    2.67    0  1228m 2104 364212   0   0   0   0  2013  303  96   4   0
    2.78    0  1228m 2104 364212   0   0   0   0  2015  304  95   5   0
    2.87    0  1228m 2104 364212   0   0   0   0  2013  304  95   5   0
    2.96    0  1228m 2104 364212   0   0   0   0  2015  299  96   4   0
    3.05    0  1228m 2104 364212   0   0   0  18  2015  308  95   5   0
    3.12    0  1228m 2104 364212   0   0   0   0  2016  306  95   5   0
    3.19    0  1228m 2104 364212   0   0   0   0  2014  306  95   5   0
...output omitted...
```

Press **`Ctrl`**+**`C`** to exit the **`pmstat`** command.

Use the **`pkill sha1sum`** command to terminate all instances of the **`sha1sum /dev/zero`** command.

```
[student@servera ~]$ pkill sha1sum
[2]   Terminated              sha1sum /dev/zero
[3]-  Terminated              sha1sum /dev/zero
[1]-  Terminated              sha1sum /dev/zero
[4]+  Terminated              sha1sum /dev/zero
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Locate the **pcp** performance metric that tracks free memory. Once you have found it, display its description, and then gather five sample outputs at a rate of 15 seconds per interval.

```
[student@servera ~]$ pminfo | grep free
mem.freemem
...output omitted...
```

```
[student@servera ~]$ pminfo -dt mem.freemem
mem.freemem
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: instant  Units: Kbyte
```

Statistic values will vary between command executions.

```
[student@servera ~]$ pmval -t 15 -s 5 mem.freemem
metric:    mem.freemem
host:      servera.lab.example.com
semantics: instantaneous value
units:     Kbyte
samples:   5
interval:  15.00 sec
              1261872
              1261904
              1261904
              1261904
              1261936
```

6. Logout from **servera**.

```
[student@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$
```

7. From the **workstation** VM, use the **pmchart** command to launch the graphical user interface. Display two charts, one for free memory and one for outbound bytes on the **en*X*** interface.

Important

Network interface names are determined by their bus type and the detection order of devices during boot. Your network interface names will vary according to the course platform and hardware in use.

On your current system, locate the interface name (such as **ens06** or **enp1s0**) associated with the Ethernet address **52:54:00:00:fa:09**. Use this interface name to replace the **en*X*** placeholder used throughout this exercise.

7.1. On the **workstation** VM, install the *pcp-gui* package.

```
[student@workstation ~]$ sudo yum install pcp-gui
[sudo] password for student: student
...output omitted...
Complete!
```

7.2. Start and enable the Performance Co-Pilot service.

```
[student@workstation ~]$ sudo systemctl enable --now pmcd
...output omitted...
```

7.3. Launch the **pmchart** utility in the background.

```
[student@workstation ~]$ pmchart &
```

7.4. Click the **New Chart** icon.

7.5. From the **Metrics** tab, select **mem** → **freemem**, and then click the **OK** button.

7.6. Click the **New Chart** icon again.

7.7. From the **Metrics** tab, select **network** → **interface** → **out** → **bytes** → **en***X* and then click the **OK** button.

7.8. Review selected charts. Experiment with a few of the other available charts.

7.9. Close the PCP Chart interface.

In the top menu bar click **File** → **Quit**.

8. From a terminal session on **workstation**, use the **ssh** command to log in to **servera** as the **student** user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$
```

9. Download the **sampleserver.tgz** archive from `http://materials.example.com/labs/perftools-copilot/sampleserver.tgz` to the **/home/student** directory on **servera**. Extract the archive.

9.1. Download the archive file.

```
[student@servera ~]$ wget \
http://materials.example.com/labs/perftools-copilot/sampleserver.tgz
...output omitted...
```

9.2. Extract the archive.

```
[student@servera ~]$ tar -xf sampleserver.tgz
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

10. Use the **pmval** command on the **sampleserver.log/20150223.12.24.0** archive to determine the exact time when the **mem.freemem** performance metric reports a saturation point of **80276**.

```
[student@servera ~]$ pmval -a sampleserver.log/20150223.12.24.0 \
mem.freemem | grep 80276 | head -n 1
12:29:42.562                80276
```

The **mem.freemem** performance metric was first logged as **80276** at 12:29:42.

11. Determine the highest number of disk reads for **vda** reported in the **sampleserver.log** archive.

11.1. Determine the Performance Co-Pilot value that could indicate disk reads.

```
[student@servera ~]$ pminfo | grep disk | grep read
disk.dev.read
disk.dev.blkread
disk.dev.read_bytes
disk.dev.read_merge
disk.dev.read_rawactive
disk.all.read
...output omitted...
```

Several of these look promising; use the **pminfo -dt** command to determine which is most appropriate.

```
[student@servera ~]$ pminfo -dt disk.dev.read
disk.dev.read [per-disk read operations]
    Data Type: 64-bit unsigned int  InDom: 60.1 0xf000001
    Semantics: counter  Units: count
```

11.2. Perhaps the data in the **disk.dev.read** performance metric will make it easy to identify the highest number of read transactions on **vda**.

```
[student@servera ~]$ pmval -a sampleserver.log/20150223.12.24.0 \
disk.dev.read | less
metric:    disk.dev.read
archive:   sampleserver.log/20150223.12.24.0
host:      server0.example.com
start:     Mon Feb 23 12:24:41 2015
end:       Mon Feb 23 12:54:41 2015
semantics: cumulative counter (converting to rate)
units:     count (converting to count / sec)
samples:   1800
interval:  1.00 sec
...output omitted...
                           vda                   vdb
12:25:41.562  No values available
12:25:42.562               0.0                   0.0
12:25:43.562               0.0                   0.0
12:25:44.562               0.0                   0.0
```

```
12:25:45.562                0.0                    0.0
...output omitted...
```

Press **q** to exit the **less** command.

11.3. The data is not easy to discern directly from the output, so use text processing tools to filter for the data that you want. Note the format of the data. Only the data in column two, for **vda**, is required to answer the question.

After cutting off the preamble information, view and sort only the data in the second column.

```
[student@servera ~]$ pmval -a sampleserver.log/20150223.12.24.0 \
disk.dev.read | tail -n +11 | awk '{print $2}' | sort -unr | head
356.0
355.0
353.0
311.0
222.0
221.0
12.00
6.000
5.000
4.000
```

The highest number of read transactions on **vda** is 356.0.

12. Logout from **servera**.

```
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On the **workstation** VM, run the **lab perftools-copilot finish** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab perftools-copilot finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Selecting Performance Monitoring Tools

Performance Checklist

In this lab, you will observe system behavior using a variety of system monitoring tools.

Outcomes

You should be able to:

- Install and configure the *sysstat* package to generate system activity reports.
- Install and configure Performance Co-Pilot to capture raw data to monitor local and remote systems.
- Display performance metrics from archive logs of particular systems using Performance Co-Pilot charts.

Before You Begin

Log in to the **`workstation`** VM as the **`student`** user with **`student`** as the password.

On **`workstation`**, run the **`lab perftools-review start`** command. This command determines if the host, **`serverb`**, is reachable on the network and copies required files to the host for this lab exercise.

```
[student@workstation ~]$ lab perftools-review start
```

1. On the **`workstation`** VM, configure the system to run Performance Co-Pilot Charts to monitor local and remote performance metrics.
2. On **`workstation`**, use **`ssh`** to log in to **`serverb`** as the **`student`** user. Escalate privileges to the **`root`** user account.
3. On **`serverb`**, install the package that provides the systemd units called **`sysstat-collect.timer`** and **`sysstat-collect.service`**. Modify the timer unit configuration file to collect system activity data once every minute.
4. On **`serverb`**, install Performance Co-Pilot packages to support basic system-level performance monitoring, and the PCP module that provides additional system monitoring tools. Enable and start the Performance Metrics Collector Daemon.
5. Configure remote access to the Performance Metrics Collector Daemon running on **`serverb`**.
6. On **`serverb`**, capture 10 samples of the **`mem.util.used`** metric from Performance Co-Pilot. Redirect the output into **`/tmp/mem-util-used-grade.data`**.
7. On **`serverb`**, download the **`sampleserver.tgz`** Performance Co-Pilot log archive from `http://materials.example.com/labs/perftools-review/sampleserver.tgz`. Determine the highest one minute load average experienced, and save this value to **`/tmp/high-1min-load.data`** When done, determine the first time that the one minute load average reaches that value, and save this value to **`/tmp/time-1min-load.data`**.

8. On **serverb**, download the pre-existing **sar** archive file from `http://materials.example.com/labs/perftools-review/sar-server.data`. Determine the highest number of network packets received per second for interface **enp1s0**, as recorded in **~/sar-server.data**. Use the format **HH:MM:SS Packets** to record the time and number of packets in the **/tmp/net-grade.data** file.
9. On the **workstation** VM, use Performance Co-Pilot Chart to create a view to the remote host **serverb** that includes the following performance metrics: **mem.util.used**, **disk.all.aveq**, and **proc.nprocs**. Name that view as **442labview**.

Evaluation

On **workstation**, run the **lab perftools-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab perftools-review grade
```

Finish

On **workstation**, run the **lab perftools-review finish** command to complete this exercise.

```
[student@workstation ~]$ lab perftools-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Selecting Performance Monitoring Tools

Performance Checklist

In this lab, you will observe system behavior using a variety of system monitoring tools.

Outcomes

You should be able to:

- Install and configure the *sysstat* package to generate system activity reports.
- Install and configure Performance Co-Pilot to capture raw data to monitor local and remote systems.
- Display performance metrics from archive logs of particular systems using Performance Co-Pilot charts.

Before You Begin

Log in to the **workstation** VM as the **student** user with **student** as the password.

On **workstation**, run the **lab perftools-review start** command. This command determines if the host, **serverb**, is reachable on the network and copies required files to the host for this lab exercise.

```
[student@workstation ~]$ lab perftools-review start
```

1. On the **workstation** VM, configure the system to run Performance Co-Pilot Charts to monitor local and remote performance metrics.

   1.1. Install the **pcp-gui** package.

```
[student@workstation ~]$ sudo yum install pcp-gui
```

   1.2. Start and enable Performance Co-Pilot Metrics Collector Daemon.

```
[student@workstation ~]$ sudo systemctl enable --now pmcd
```

2. On **workstation**, use **ssh** to log in to **serverb** as the **student** user. Escalate privileges to the **root** user account.

   2.1. Log in to **serverb** as the **student** user.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3. On **serverb**, install the package that provides the systemd units called **sysstat-collect.timer** and **sysstat-collect.service**. Modify the timer unit configuration file to collect system activity data once every minute.

3.1. Install the *sysstat* package.

```
[root@serverb ~]# yum install sysstat
...output omitted...
```

3.2. Copy **/usr/lib/systemd/system/sysstat-collect.timer** to **/etc/systemd/system/sysstat-collect.timer**.

```
[root@serverb ~]# cp /usr/lib/systemd/system/sysstat-collect.timer \
/etc/systemd/system/sysstat-collect.timer
```

3.3. Edit **/etc/systemd/system/sysstat-collect.timer** so that the timer unit runs once every minute.

```
[root@serverb ~]# vim /etc/systemd/system/sysstat-collect.timer
...output omitted...
#        Activates activity collector once every minute

[Unit]
Description=Run system activity accounting tool once every minute

[Timer]
OnCalendar=*:00/01

[Install]
WantedBy=sysstat.service
```

3.4. Use the **systemctl daemon-reload** command to make sure that **systemd** is aware of the changes.

```
[root@serverb ~]# systemctl daemon-reload
```

3.5. Use the **systemctl** command to activate the **sysstat-collect.timer** timer unit.

```
[root@serverb ~]# systemctl enable --now sysstat-collect.timer
```

4. On **serverb**, install Performance Co-Pilot packages to support basic system-level performance monitoring, and the PCP module that provides additional system monitoring tools. Enable and start the Performance Metrics Collector Daemon.

4.1. Install the *pcp* and *pcp-system-tools* packages.

```
[root@serverb ~]# yum install pcp pcp-system-tools
...output omitted...
Complete!
```

4.2. Start and enable the Performance Co-Pilot Metrics Collector Daemon.

```
[root@serverb ~]# systemctl enable --now pmcd
...output omitted...
```

5. Configure remote access to the Performance Metrics Collector Daemon running on **serverb**.

   5.1. Use the **firewall-cmd** command to add the **pmcd** service.

```
[root@serverb ~]# firewall-cmd --permanent --add-service="pmcd"
success
```

   5.2. Reload the firewall configuration.

```
[root@serverb ~]# firewall-cmd --reload
success
```

   5.3. Verify the firewall configuration.

```
[root@serverb ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: cockpit dhcpv6-client pmcd ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

6. On **serverb**, capture 10 samples of the **mem.util.used** metric from Performance Co-Pilot. Redirect the output into **/tmp/mem-util-used-grade.data**.

```
[root@serverb ~]# pmval -s 10 mem.util.used > /tmp/mem-util-used-grade.data
```

7. On **serverb**, download the **sampleserver.tgz** Performance Co-Pilot log archive from `http://materials.example.com/labs/perftools-review/sampleserver.tgz`. Determine the highest one minute load average experienced, and save this value to **/tmp/high-1min-load.data** When done, determine the first time that the one minute load average reaches that value, and save this value to **/tmp/time-1min-load.data**.

   7.1. Download the Performance Co-Pilot log data for the sample system.

```
[root@serverb ~]# wget \
http://materials.example.com/labs/perftools-review/sampleserver.tgz
```

   7.2. Untar the archive to access the Performance Co-Pilot log data.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# tar -xf sampleserver.tgz
```

7.3. Determine the parameter that will provide the one minute load average.

```
[root@serverb ~]# pminfo | grep load
...output omitted...
kernel.all.load
...output omitted...

[root@serverb ~]# pminfo -t kernel.all.load
kernel.all.load [1, 5 and 15 minute load average]
```

7.4. Determine the highest one minute load average experienced. Redirect this value to **/tmp/high-1min-load.data**.

```
[root@serverb ~]# pmval -a sampleserver.log/20150223.12.24.0 \
kernel.all.load | tail -n +11 | \
awk '{print $2}' | sort -rn | \
head -n 1 > /tmp/high-1min-load.data
```

7.5. Verify that your content matches the following:

```
[root@serverb ~]# cat /tmp/high-1min-load.data
19.12
```

7.6. Determine the first time that the one minute load average reached **19.12**. Redirect this value to **/tmp/time-1min-load.data**. You must know the time zone of the archive to maintain consistency when displaying time values on systems in different time zones.

```
[root@serverb ~]# pmdumplog -L sampleserver.log/20150223.12.24.0
Log Label (Log Format Version 2)
Performance metrics from host server0.example.com
    commencing Mon Feb 23 11:24:41.562205 2015
    ending     Mon Feb 23 11:54:41.500123 2015
Archive timezone: EST+5
PID for pmlogger: 9217

[root@serverb ~]# pmval -a sampleserver.log/20150223.12.24.0 \
-Z EST+5 kernel.all.load | grep 19\.12 | \
head -n 1 > /tmp/time-1min-load.data
```

7.7. Verify that your content matches the following:

```
[root@serverb ~]# cat /tmp/time-1min-load.data
12:32:41.562      19.12          5.060         1.860
```

8. On **serverb**, download the pre-existing **sar** archive file from `http://materials.example.com/labs/perftools-review/sar-server.data`. Determine the highest number of network packets received per second for interface **enp1s0**, as

recorded in **~/sar-server.data**. Use the format **HH:MM:SS Packets** to record the time and number of packets in the **/tmp/net-grade.data** file.

8.1. Download the `http://materials.example.com/labs/perftools-review/sar-server.data` to **/home/student**.

```
[root@serverb ~]# wget \
http://materials.example.com/labs/perftools-review/sar-server.data
...output omitted...
```

8.2. Use the **sar** command to read the first ten lines of the archive file to identify the format.

```
[root@serverb ~]# sar -n DEV -f sar-server.data | head
Linux 4.18.0-80.el8.x86_64 (serverb.lab.example.com)  07/04/2019  _x86_64_ (2 CPU)

00:04:26     LINUX RESTART (2 CPU)

12:06:01 AM   IFACE  rxpck/s  txpck/s ... %ifutil
12:07:01 AM      lo     0.00     0.00        0.00
12:07:01 AM  enp1s0     1.13     0.50        0.00
12:08:01 AM      lo     0.00     0.00        0.00
12:08:01 AM  enp1s0     0.62     0.12        0.00
12:09:01 AM      lo     0.00     0.00        0.00
```

8.3. Use the following filters to parse the time and received packets values.

- **sar -n DEV -f sar-server.data** reports network statistics from a pre-existing tar file.
- **grep '^[^A].*enp1s0'** isolates only those lines associated with **enp1s0** after eliminating lines that do not start with a time stamp, such as those that start with the word **Average**.
- **awk '{print $1, $4}' | sort -rnk 2** prints the first and third columns of the output generated by the **grep** command, and performs a numeric sort on column three in descending order.
- **head -n+1 > /tmp/net-grade.data** redirects the first line of the sorted data into the **/tmp/net-grade.data** file.

```
[root@serverb ~]# sar -n DEV -f sar-server.data | \
grep '^[^A].*enp1s0' | \
awk '{print $1, $4}' | sort -rnk 2 | \
head -n+1 > /tmp/net-grade.data
```

8.4. Verify that your results match the following:

```
[root@serverb ~]# cat /tmp/net-grade.data
01:08:01 202.23
```

8.5. Logout from **serverb**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

9. On the **`workstation`** VM, use Performance Co-Pilot Chart to create a view to the remote host **`serverb`** that includes the following performance metrics: **`mem.util.used`**, **`disk.all.aveq`**, and **`proc.nprocs`**. Name that view as **`442labview`**.

9.1. Start the **`pmchart`** interface to connect specifically to the **`serverb`** remote host.

```
[student@workstation ~]$ pmchart -h serverb &
```

9.2. Add the three performance metrics.

Click the **New Chart** button. On the **Metrics** tab, select **mem** → **util** → **used**. Click the **OK** button.

Click the **New Chart** button. On the **Metrics** tab, select **disk** → **all** → **aveq**. Click the **OK** button.

Click the **New Chart** button. On the **Metrics** tab, select **proc** → **nprocs**. Click the **OK** button.

9.3. Save your view as **`442labview`**.

Open the **File** menu, and then select **Save View ...**.

Change the **Path:** field to **`/home/student`**.

In the **Filename:** field, enter **`442labview`**.

Click the **Save** button.

9.4. Quit the graphical interface for PCP Charts.

Open the **File** menu, and then select **Quit**.

Evaluation

On **`workstation`**, run the **`lab perftools-review grade`** command to confirm success of this exercise.

```
[student@workstation ~]$ lab perftools-review grade
```

Finish

On **`workstation`**, run the **`lab perftools-review finish`** command to complete this exercise.

```
[student@workstation ~]$ lab perftools-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- One of the goals of system monitoring is to determine whether the current execution of system components meets the specified technical requirements.
- The *sysstat* package utilities derive raw data from kernel counters, and provide performance monitoring of CPU usage, disk I/O, process usage, memory usage, and more.
- Performance Co-Pilot (PCP) packages provide the framework for monitoring and management of real-time data, as well as logging and retrieval of historical data. System monitoring tools are available in both command-line and graphical user interfaces.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 3

Viewing Hardware Resources

Goal

View and interpret hardware resource listings.

Objectives

- Describe the utilities that list system hardware resources.
- Describe the techniques used to list resources in virtual and cloud instances.

Sections

- Displaying Physical Resources (and Guided Exercise)
- Displaying Resources in Virtual and Cloud Instances (and Quiz)

Quiz

Viewing Hardware Resources

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Displaying Physical Resources

Objectives

After completing this section, you should be able to describe the utilities that list system hardware resources.

Hardware Resources

Before tuning the performance of a system, it is important to understand the theoretical capabilities of that system. The first step toward that understanding is to develop a hardware profile of the system that you want to tune.

CPU

The central processing unit (CPU) processes machine instructions that are provided by the software. Cores are integrated into a single integrated circuit die, known as a chip multiprocessor, or multiple dies in a single chip package. Each server system has one or more CPUs, each residing on a motherboard socket. Although having more CPUs can provide more processing power, software must be written as multi-threaded to be able to use multiple CPUs simultaneously to efficiently leverage this additional processing capacity.

Memory

Volatile computer memory temporarily stores programs or program data while in use. Having more available memory can reduce the frequency of the system reading data from storage devices, because physical memory can be used as a storage device cache when unused memory is available. When under memory pressure, the kernel has to choose between decreasing caches or moving objects to swap areas. Virtual memory consists of both the physical memory, and the space available on swap devices, which are typically disk devices.

Storage

Nonvolatile storage devices preserve programs and data on physical recording mediums. Hard disk drives (HDD) use spinning magnetic discs and multiple read/write heads that move to where the data resides on the spinning discs. Newer solid-state drives (SSD) save data on integrated circuit assemblies, similar to memory but persistent. SSD performance properties generally match their architecture, which describes the number of bits stored per cell. Single level cells (SLC) hold 1 bit, and are generally the most reliable and fast, but also the most expensive. Multilevel (MLC) and triple level (TLC) cells hold 2 or 3 bits, respectively, with slightly decreased reliability and lower cost. The newest drives for the consumer market are quad level (QLC), holding 4 bits. The reliability and speed of SSDs compared to HDDs is significantly greater, but SSD costs remain higher per Gigabyte and therefore, cell quality remains relevant.

Networking

Computer systems share resources over connections made over cable media. Typically, Ethernet networks and interface cards are connected using network switches. Common connection capacities are 1Gbps and 10-Gbps, with 40-Gbps and 100-Gbps segments commonly found in enterprise data centers. 400-Gbps Ethernet was first demonstrated in 2017, and is ready for data center deployment. InfiniBand is another common connection technology with very high

throughput and low latency, used for high-performance computing clusters interconnects. Single InfiniBand links can provide up to 250 Gbps in each direction concurrently. InfiniBand links can be aggregated to the currently published maximum of 16, for a 4 Tbps total throughput.

Interconnects

External components are connected to the system by a local computer bus. There are many different bus types available, with characteristics such as optimizations for functional use, and bandwidth demands. The *Peripheral Component Interconnect Express (PCIe)* bus is the most commonly supported standard interface for internally attaching devices in a computer.

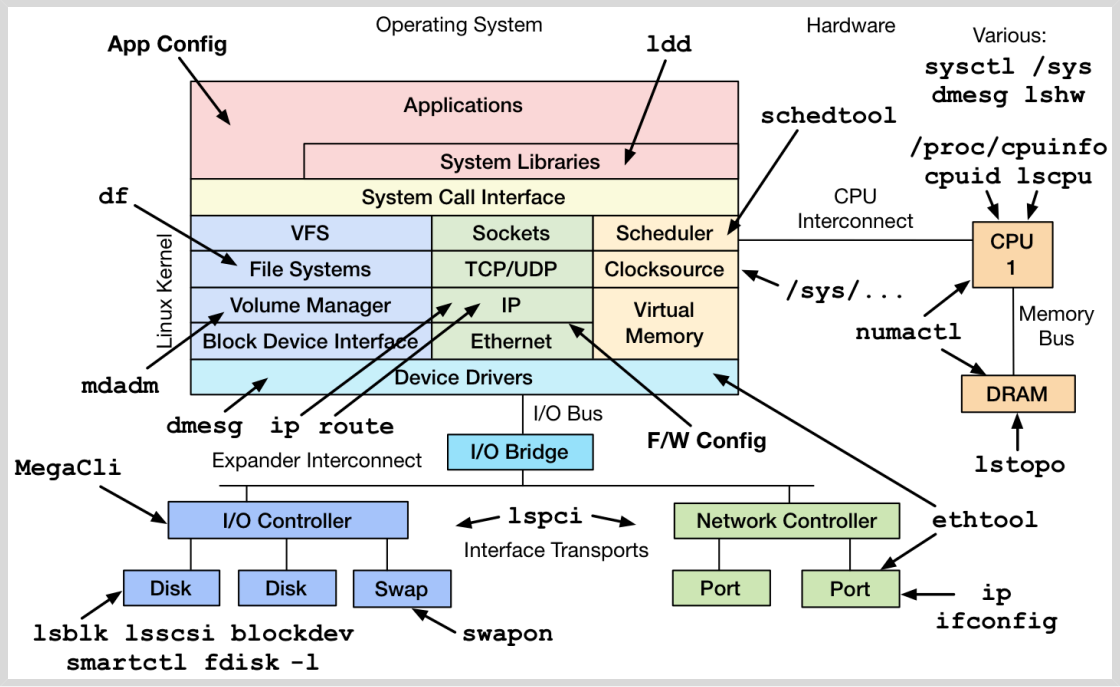


Figure 3.1: Linux Static Performance Tools by Brendan Gregg (CC BY-SA 4.0)

Reviewing Kernel Messages

The easiest and most direct source of hardware information is the running Linux kernel, which functions as the mediator of all hardware access. The kernel exposes information structures through the **`/proc`** file system, the **`/sys`** file system, and also as kernel messages and events.

Kernel messages are written to a preallocated ring buffer known as the **`dmesg`** buffer. A ring buffer is a circular buffer where data overflow simply starts again at the top of the buffer. Over time, more recent messages will fill the buffer and overwrite the original messages, but the buffer will never grow in size.

There are three common motivations for examining the **`dmesg`** buffer:

- Reviewing what hardware was detected at boot time.
- Observing messages printed by drivers as hardware is attached or detected.
- Observing warning or error message conditions as they occur.

Ring buffer messages are displayed using the **`dmesg`** command or **`journalctl`** command with the **`-k`** option. The **`journald`** service can be configured for persistent storage, and is preferred over use of **`/var/log/dmesg`**. The kernel ring buffer is captured within the **`systemd-journal`**, as well as in **`/var/log/messages`**, via the **`imjournal rsyslog`** plug-in.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

With the introduction of the system journal in RHEL 7, the **/var/log/dmesg** file is regarded as optional, and no longer created automatically in RHEL 8. This file can be restored by recreating and enabling the **/etc/systemd/system/dmesg.service** service. See Knowledgebase article *The /var/log/dmesg file is not created during boot for Red Hat Enterprise Linux 8* at https://access.redhat.com/solutions/3748981.

Exploring Kernel Messages

The first 2 lines of the **dmesg** command display the kernel version and command line boot options passed to the kernel by the boot loader. It is useful to know which command line options were used during system initialization in the event that you need to troubleshoot system behavior.

```
[root@host ~]# dmesg | head -n2
[    0.000000] Linux version 4.18.0-80.el8.x86_64
(mockbuild@x86-vm-08.build.eng.bos.redhat.com)
(gcc version 8.2.1 20180905 (Red Hat 8.2.1-3) (GCC))
#1 SMP Wed Mar 13 12:02:46 UTC 2019
[    0.000000] Command line:
BOOT_IMAGE=(hd0,msdos1)/boot/vmlinuz-4.18.0-80.el8.x86_64
root=UUID=884f47c9-a69d-4c5b-915d-6b7c9c74c923 ro console=tty0
console=ttyS0,115200n8 no_timer_check crashkernel=auto
```

System memory is organized into *Direct Memory Access (DMA)* memory zones for allocation. Many kernel subsystems and older devices use memory from particular zones.

Use the **dmesg** command to view DMA memory zones and total available memory.

```
[root@host ~]# dmesg | grep -i 'dma'
[    0.000000]   DMA      [mem 0x0000000000001000-0x0000000000ffffff]
[    0.000000]   DMA32    [mem 0x0000000001000000-0x000000007ffd7fff]
[    0.000000]   DMA zone: 64 pages used for memmap
[    0.000000]   DMA zone: 21 pages reserved
[    0.000000]   DMA zone: 3998 pages, LIFO batch:0
[    0.000000]   DMA32 zone: 8128 pages used for memmap
[    0.000000]   DMA32 zone: 520152 pages, LIFO batch:31
[    0.000000] Policy zone: DMA32
[    2.134084] [TTM] Initializing DMA pool allocator
[    2.134643] ata1: SATA max UDMA/133 abar m4096@...9000 port 0xfea19100 irq 46
[    2.134649] ata2: SATA max UDMA/133 abar m4096@...9000 port 0xfea19180 irq 46
[    2.134655] ata3: SATA max UDMA/133 abar m4096@...9000 port 0xfea19200 irq 46
[    2.134661] ata4: SATA max UDMA/133 abar m4096@...9000 port 0xfea19280 irq 46
[    2.134666] ata5: SATA max UDMA/133 abar m4096@...9000 port 0xfea19300 irq 46
[    2.134672] ata6: SATA max UDMA/133 abar m4096@...9000 port 0xfea19380 irq 46
```

```
[root@host ~]# dmesg | grep 'Memory'
[    0.000000] Memory: 261668K/2096600K available (12292K kernel code, 2100K
 rwdata, 3816K rodata, 2348K init, 3320K bss, 271600K reserved, 0K cma-reserved)
[    0.108065] x86/mm: Memory block size: 128MB
```

The single boot CPU core activates the remaining cores on all detected physical CPUs.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# dmesg | grep 'CPU'
[    0.000000] smpboot: Allowing 2 CPUs, 0 hotplug CPUs
[    0.000000] setup_percpu: NR_CPUS:8192 nr_cpumask_bits:2 nr_cpu_ids:2
 nr_node_ids:1
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1
[    0.001000] rcu:  RCU restricting CPUs from NR_CPUS=8192 to nr_cpu_ids=2.
[    0.033186] x86/cpu: Activated the Intel User Mode Instruction Prevention
 (UMIP) CPU feature
[    0.035007] mce: CPU supports 10 MCE banks
[    0.066209] smpboot: CPU0: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz (family:
 0x6, model: 0x5e, stepping: 0x3)
[    0.076516] smp: Bringing up secondary CPUs ...
[    0.078008] .... node  #0, CPUs:      #1
[    0.001000] x86/cpu: Activated the Intel User Mode Instruction Prevention
 (UMIP) CPU feature
[    0.088011] smp: Brought up 1 node, 2 CPUs
```

The mechanism for allocating huge memory pages is initialized.

```
[root@host ~]# dmesg | grep 'Huge'
[    0.124725] HugeTLB registered 1.00 GiB page size, pre-allocated 0 pages
[    0.125007] HugeTLB registered 2.00 MiB page size, pre-allocated 0 pages
[    3.829838] systemd[1]: Mounting Huge Pages File System...
```

As storage is initialized, the various I/O elevator strategies are registered.

```
[root@host ~]# dmesg | grep 'io scheduler'
[    1.032885] io scheduler mq-deadline registered
[    1.034521] io scheduler kyber registered
[    1.036059] io scheduler bfq registered
```

Attached SCSI devices, and network-attached storage on relevant systems, are itemized.

The following example shows an attached CD-ROM as **/dev/sr0**, an attached storage devices as **/dev/sdb** and **/dev/sda** with partitions **sda1** and **sda2**.

```
...output omitted...
[    0.967540] cdrom: Uniform CD-ROM driver Revision: 3.20
[    0.967651] sr 2:0:0:0: Attached scsi CD-ROM sr0
[    0.982652]  sdb:
[    0.982805] sd 1:0:0:0: [sdb] Attached SCSI disk
[    0.993686]  sda: sda1 sda2
[    0.994024] sd 0:0:0:0: [sda] Attached SCSI disk
...output omitted...
```

When an error occurs, a symbolic stack trace is often included. This is helpful information to provide to Red Hat technical support when troubleshooting potential problems.

```
...output omitted...
i5000_edac: waiting for edac_mc to get alive
EDAC MC0: Giving out device to i5000_edac.c I5000: DEV 0000:00:10.0
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
kobject_add failed for i5000_edac with -EEXIST, don't try to register things with
 the same name in the same directory.

Call Trace:
 [<ffffffff8014ebaf>] kobject_add+0x170/0x19b
 [<ffffffff8014e0e6>] cmp_ex+0x0/0x10
 [<ffffffff80041306>] load_module+0x1692/0x19f0
 [<ffffffff80155cbf>] pci_bus_read_config_byte+0x0/0x72
 [<ffffffff80128a95>] task_has_capability+0x54/0x60
 [<ffffffff8009fdcf>] autoremove_wake_function+0x0/0x2e
 [<ffffffff800a6183>] sys_init_module+0x4d/0x1f2
 [<ffffffff8005d116>] system_call+0x7e/0x83
intel_rng: FWH not detected
...output omitted...
```

Network cards are discovered as the appropriate PCI drivers are loaded. In this example, a Broadcom Tigon3 Ethernet card is detected as **eth0**.

```
...output omitted...
[    0.652792] tg3 0000:34:00.0 eth0: Tigon3 [partno(BCM95751A519FLP) rev 4201]
 (PCI Express) MAC address 00:10:18:48:2a:49
[    0.652795] tg3 0000:34:00.0 eth0: attached PHY is 5750 (10/100/1000Base-T
[    0.652797] tg3 0000:34:00.0 eth0: RXcsums[1] LinkChgREG[0] MIirq[0] ASF[0]
[    0.652799] tg3 0000:34:00.0 eth0: dma_rwctrl[76180000] dma_mask[64-bit]
...output omitted...
```

Modifying Kernel Message Output

Use the **-T** option to convert the standard **dmesg** time stamp into a human-friendly format.

```
[root@host ~]# dmesg -T
...output omitted...
[Mon Aug 12 14:13:21 2019] tsc: Detected 1861.914 MHz processor
[Mon Aug 12 14:13:21 2019] Calibrating delay loop (skipped) preset value.. 3723.82
 BogoMIPS (lpj=1861914)
[Mon Aug 12 14:13:21 2019] pid_max: default: 32768 minimum: 301
[Mon Aug 12 14:13:21 2019] Security Framework initialized
[Mon Aug 12 14:13:21 2019] SELinux:  Initializing.
...output omitted...
```

Retrieving CPU information

Modern systems typically have multiple CPUs, each with multiple cores per socket, and possibly multiple hyper-threads per core, with different levels of local and shared caches. The **lscpu** command provides a quick summary of the local configuration. The following example shows **lscpu** output, and an explanation of the pertinent information.

```
[root@host ~]# lscpu
Architecture:          x86_64                 (1)
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4                      (2)
On-line CPU(s) list:   0-3
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
Thread(s) per core:    1                    ❸
Core(s) per socket:    4                    ❹
Socket(s):             1                    ❺
NUMA node(s):          1                    ❻
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 23
Model name:            Intel(R) Core(TM)2 Quad CPU    Q9550  @ 2.83GHz
Stepping:              10
CPU MHz:               2833.000
BogoMIPS:              5665.57
Virtualization:        VT-x                 ❼
L1d cache:             32K                  ❽
L1i cache:             32K
L2 cache:              6144K
NUMA node0 CPU(s):     0-3                  ❾
```

❶ The architecture of the CPU.
❷ The number of logical cores available to the kernel for task scheduling.
❸ The number of hyper-threads per core.
❹ The number of cores per socket.
❺ The number of physical sockets.
❻ The number of distinct memory buses on NUMA systems.
❼ The hardware-assisted Virtualization support for the CPU, if any.
❽ The sizes of the relative caches.
❾ The mapping of the various logical CPUs to the NUMA address buses.

While **`lscpu`** lists the sizes of the various CPU caches, it does not answer the question of which caches are shared by which logical CPUs. However, when invoked as **`lscpu -p`**, the parsable output can be used to tease out this information.

In the following output, the last three columns indicate how the cache is distributed among the CPUs. The L1 caches are distinct for each CPU, while the L2 cache is shared between **`CPU 0`** and **`CPU 4`**, and also between **`CPU 1`** and **`CPU 2`**.

```
[root@host ~]# lscpu -p
# The following is the parse-able format, which can be fed to other
# programs. Each different item in every column has an unique ID
# starting from zero.
# CPU,Core,Socket,Node,,L1d,L1i,L2
0,0,0,0,,0,0,0
1,1,0,0,,1,1,1
2,2,0,0,,2,2,1
3,3,0,0,,3,3,0
```

The POSIX standard defines the **`getconf`** library call, used to programmatically determine local hardware aspects, which can be relevant for software optimization. These parameters are available using the **`getconf`** command.

While most **`getconf`** information relates to standards conformance for various software subsystems, some important hardware information is included. The following example demonstrates output from a different platform, showing availability of L1, L2, and L3 caches.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# getconf -a
PAGESIZE                           4096
PAGE_SIZE                          4096
...output omitted...
_AVPHYS_PAGES                      36380772
_NPROCESSORS_CONF                  24
_NPROCESSORS_ONLN                  24
_PHYS_PAGES                        37209409
...output omitted...
LEVEL1_ICACHE_SIZE                 32768
LEVEL1_ICACHE_ASSOC                4
LEVEL1_ICACHE_LINESIZE             64
LEVEL1_DCACHE_SIZE                 32768
LEVEL1_DCACHE_ASSOC                8
LEVEL1_DCACHE_LINESIZE             64
LEVEL2_CACHE_SIZE                  262144
LEVEL2_CACHE_ASSOC                 8
LEVEL2_CACHE_LINESIZE              64
LEVEL3_CACHE_SIZE                  12582912
LEVEL3_CACHE_ASSOC                 16
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC                 0
LEVEL4_CACHE_LINESIZE              0
...output omitted...
```

Retrieving SMBIOS/DMI information

The **dmidecode** utility probes the local *System Management BIOS (SMBIOS)* and *Desktop Management Interface (DMI)* data structures, and displays significant local hardware information.

In the following excerpts, the **dmidecode** command shows the CPUs in the target host.

```
[root@host ~]# dmidecode
# dmidecode 2.10
SMBIOS 2.5 present.
82 structures occupying 3794 bytes.
Table at 0x7F6BD000.
...output omitted...
Handle 0x0001, DMI type 4, 40 bytes
Processor Information
        Socket Designation: Node 1 Socket 1
        Type: Central Processor
        Family: Xeon MP
        Manufacturer: Intel(R) Corporation
        ID: C2 06 02 00 FF FB EB BF
        Signature: Type 0, Family 6, Model 44, Stepping 2
        Flags:
                FPU (Floating-point unit on-chip)
                VME (Virtual mode extension)
...output omitted...
                HTT (Hyper-threading technology)
                TM (Thermal monitor supported)
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
                PBE (Pending break enabled)
        Version: Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz
        Voltage: 1.2 V
        External Clock: 6400 MHz
        Max Speed: 4000 MHz
        Current Speed: 2670 MHz
        Status: Populated, Enabled
        Upgrade: ZIF Socket
        L1 Cache Handle: 0x0002
        L2 Cache Handle: 0x0003
        L3 Cache Handle: 0x0004
...output omitted...
        Core Count: 6
        Core Enabled: 6
        Thread Count: 12
        Characteristics:
                64-bit capable
...output omitted...
```

The **dmidecode** command lists the associated CPU caches.

```
...output omitted...
Handle 0x0002, DMI type 7, 19 bytes
Cache Information
        Socket Designation: Internal Cache Level 1
        Configuration: Enabled, Not Socketed, Level 1
        Operational Mode: Write Back
        Location: Internal
        Installed Size: 32 kB
        Maximum Size: 32 kB
...output omitted...
        Associativity: 16-way Set-associative
...output omitted...
```

Conveniently, physical system information is available, thus saving a trip to the data center.

```
...output omitted...
Handle 0x000A, DMI type 1, 27 bytes
System Information
        Manufacturer: IBM
        Product Name: BladeCenter HS22V -[7871AC1]-
        Version: 03
        Serial Number: 06ERF91
        UUID: 0D542590-CD16-11E0-8D30-5CF3FCC11164
...output omitted...
```

Various "on system board" devices are listed, such as network interfaces.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
...output omitted...
Handle 0x0019, DMI type 10, 6 bytes
On Board Device Information
        Type: Ethernet
        Status: Enabled
        Description: Ethernet 1 BroadCom 5709S Ethernet Controller
...output omitted...
```

Importantly, a full itemization of supported and installed RAM is provided.

```
...output omitted...
Handle 0x002B, DMI type 16, 15 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: Single-bit ECC
        Maximum Capacity: 288 GB
        Error Information Handle: Not Provided
        Number Of Devices: 18

Handle 0x002C, DMI type 6, 12 bytes
Memory Module Information
        Socket Designation: DIMM01
        Bank Connections: 0 0
        Current Speed: 1 ns
        Type: DIMM
        Installed Size: 8192 MB (Single-bank Connection)
        Enabled Size: 8192 MB (Single-bank Connection)
        Error Status: OK

Handle 0x002D, DMI type 17, 27 bytes
Memory Device
        Array Handle: 0x002B
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 8192 MB
        Form Factor: DIMM
        Set: None
        Locator: DIMM01
        Bank Locator: BANK00
        Type: Other
        Type Detail: Unknownof a computer
        Speed: 800 MHz
...output omitted...
```

Empty memory sockets are listed also, allowing assessment of possible additional capacity.

```
...output omitted...
Handle 0x001B, DMI type 17, 21 bytes
Memory Device
        Array Handle: 0x0017
        Error Information Handle: Not Provided
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
        Total Width: 72 bits
        Data Width: 64 bits
        Size: No Module Installed
        Form Factor: DIMM
...output omitted...
```

The **dmidecode** data is obtained through the **sysfs** file system. Administrators can query for the same kernel-provided information in **/sys/class/dmi/id**. Accessing data stored in component files for use in scripts and reports is simple.

```
[root@host ~]# ls /sys/class/dmi/id/
bios_date        board_vendor       chassis_version  product_version
bios_vendor      board_version      modalias         subsystem
bios_version     chassis_asset_tag  power            sys_vendor
board_asset_tag  chassis_serial     product_name     uevent
board_name       chassis_type       product_serial
board_serial     chassis_vendor     product_uuid
[root@host ~]# head /sys/class/dmi/id/product_*
==> /sys/class/dmi/id/product_name <==
BladeCenter HS22V -[7871AC1]-

==> /sys/class/dmi/id/product_serial <==
06ERF91

==> /sys/class/dmi/id/product_uuid <==
0D542590-CD16-11E0-8D30-5CF3FCC11164

==> /sys/class/dmi/id/product_version <==
03
```

Retrieving Peripheral Information

A system administrator may need to know what peripherals are connected to the PCI or USB system bus. The **lspci** and **lsusb** commands are useful to query these subsystems.

- **lspci**: lists devices connected to PCI buses in the system
- **lsusb**: lists devices connected to the system's USB buses

List Devices Connected by PCI Bus

The **lspci** command displays hardware devices connected to the system PCI bus. The hardware can be integrated on the motherboard, or physically connected through a PCI slot.

```
[root@host ~]# lspci
00:00.0 Host bridge: Intel Corporation 82X38/X48 Express DRAM Controller
00:01.0 PCI bridge: Intel Corporation 82X38/X48 Express Host-Primary PCI Express
 Bridge
00:1a.0 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller
 #4 (rev 02)
...output omitted...
00:1a.7 USB controller: Intel Corporation 82801I (ICH9 Family) USB2 EHCI
 Controller #2 (rev 02)
00:1b.0 Audio device: Intel Corporation 82801I (ICH9 Family) HD Audio Controller
 (rev 02)
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
00:1c.0 PCI bridge: Intel Corporation 82801I (ICH9 Family) PCI Express Port 1 (rev
 02)
...output omitted...
00:1d.0 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller
 #1 (rev 02)
...output omitted...
00:1d.7 USB controller: Intel Corporation 82801I (ICH9 Family) USB2 EHCI
 Controller #1 (rev 02)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev 92)
00:1f.0 ISA bridge: Intel Corporation 82801IR (ICH9R) LPC Interface Controller
 (rev 02)
00:1f.2 RAID bus controller: Intel Corporation 82801 SATA Controller [RAID mode]
 (rev 02)
01:00.0 VGA compatible controller: NVIDIA Corporation G86 [Quadro NVS 290] (rev
 a1)
10:0b.0 FireWire (IEEE 1394): LSI Corporation FW322/323 [TrueFire] 1394a
 Controller (rev 61)
34:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5751 Gigabit
 Ethernet PCI Express (rev 21)
3f:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5755 Gigabit
 Ethernet PCI Express (rev 02)
```

The **`lspci`** command displays devices connected to the system PCI bus in the following format:

```
Busnumber:Devicenumber.Function Device Class: Vendor Device.
```

Additional information can be displayed by adding up to three verbose switches to the **`lspci`** command. Using **`lspci -vv`** is usually sufficient to obtain all relevant hardware information. For example, to find the kernel driver used to communicate with the hardware:

```
[root@host ~]# lspci -vv
...output omitted...
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
 Subsystem: Red Hat, Inc Device 0001
 Physical Slot: 3
 Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping-
 SERR+ FastB2B- DisINTx+
 Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort-
 >SERR- <PERR- INTx-
 Latency: 0
 Interrupt: pin A routed to IRQ 10
 Region 0: I/O ports at c0c0 [size=32]
 Region 1: Memory at fc052000 (32-bit, non-prefetchable) [size=4K]
 Expansion ROM at fc000000 [disabled] [size=256K]
 Capabilities: [40] MSI-X: Enable+ Count=3 Masked-
  Vector table: BAR=1 offset=00000000
  PBA: BAR=1 offset=00000800
 Kernel driver in use: virtio-pci
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

List Devices Connected by USB Bus

The **lsusb** command displays all hardware connected to the system USB bus. The hardware can be integrated on the motherboard, or physically connected through a USB port.

```
[root@host ~]# lsusb
Bus 002 Device 022: ID 0bdb:1900 Ericsson Business Mobile Networks BV F3507g
 Mobile Broadband Module
Bus 002 Device 028: ID 090c:1000 Silicon Motion, Inc. - Taiwan (formerly Feiya
 Technology Corp.) Flash Drive
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 003: ID 17ef:4807 Lenovo UVC Camera
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 003: ID 0a5c:2145 Broadcom Corp. BCM2045B (BDC-2.1) [Bluetooth
 Controller]
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

The **lsusb** command uses the following format:

```
Bus Busnumber Device Devicenumber ID VendorID:ProductID Vendor Product
```

Additional information can be displayed by adding up to three verbose switches to the **lsusb** command. Using **lsusb -vv** is usually sufficient to obtain all relevant hardware information.

```
[root@host ~]# lsusb -vv
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.10
  bDeviceClass            9 Hub
  bDeviceSubClass         0 Unused
  bDeviceProtocol         0 Full speed (or root) hub
  bMaxPacketSize0        64
  idVendor           0x1d6b Linux Foundation
  idProduct          0x0001 1.1 root hub
  bcdDevice            3.10
  iManufacturer           3 Linux 3.10.0-123.el7.x86_64 uhci_hcd
  iProduct                2 UHCI Host Controller
  iSerial                 1 0000:00:01.2
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           25
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
    bmAttributes         0xe0
      Self Powered
      Remote Wakeup
    MaxPower                0mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           1
      bInterfaceClass         9 Hub
      bInterfaceSubClass      0 Unused
      bInterfaceProtocol      0 Full speed (or root) hub
      iInterface              0
      Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x81  EP 1 IN
        bmAttributes            3
          Transfer Type            Interrupt
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0002  1x 2 bytes
        bInterval             255
Hub Descriptor:
  bLength               9
  bDescriptorType      41
  nNbrPorts             2
  wHubCharacteristic 0x000a
    No power switching (usb 1.0)
    Per-port overcurrent protection
  bPwrOn2PwrGood        1 * 2 milli seconds
  bHubContrCurrent      0 milli Ampere
  DeviceRemovable    0x00
  PortPwrCtrlMask    0xff
 Hub Port Status:
   Port 1: 0000.0100 power
   Port 2: 0000.0100 power
Device Status:     0x0001
  Self Poweredi
```

Collecting System Component Information

Gathering a system hardware profile requires tools that provide detailed information about hardware components. Various tools are available for collecting system component information. Many tools provide both command line and graphical representation of a system, and can output to archive files.

Portable Hardware Locality

The *Portable Hardware Locality* (hwloc) package provides command line and graphical tools to gather information about system components, including cache, memory, I/O devices, and network interfaces.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The **lstopo** command creates a graphical representation of the system. If a graphical interface is not available, then the **lstopo-no-graphics** command provides detailed textual output.

The default listing for the **lstopo-no-graphics** displays memory and PCI information.

```
[root@host ~]# lstopo-no-graphics
Machine (1829MB)
  Package L#0 + L3 L#0 (16MB) + L2 L#0 (4096KB) + L1d L#0 (32KB) + L1i L#0 (32KB)
 + Core L#0 + PU L#0 (P#0)
  Package L#1 + L3 L#1 (16MB) + L2 L#1 (4096KB) + L1d L#1 (32KB) + L1i L#1 (32KB)
 + Core L#1 + PU L#1 (P#1)
  Misc(MemoryModule)
  HostBridge L#0
    PCIBridge
      PCI 1af4:1041
    PCIBridge
      PCI 1af4:1042
    PCIBridge
      PCI 1af4:1042
    PCIBridge
      PCI 1af4:1042
    PCIBridge
      PCI 1af4:1042
    PCI 1af4:1050
    PCI 8086:2922
```

Listing Hardware Configurations

The **lshw** command lists detailed information about the hardware configuration of a machine. The **lshw** command lists exact memory configuration, such as firmware version, mainboard configuration, CPU version and speed, cache configuration, and bus speed.

The **lshw** command separates hardware components into classes, such as system, memory, network, and more. Use the **lshw -short** command to list a capsulized version of hardware information, including the available hardware classes on your system.

```
[root@host ~]# lshw -short
Bus info          Device    Class      Description
==================================================
                            system     KVM
                            bus        Motherboard
                            memory     96KiB BIOS
cpu@0                       processor  Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
cpu@1                       processor  Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
                            memory     2GiB System Memory
                            memory     2GiB DIMM RAM
pci@0000:00:00.0            bridge     82G33/G31/P35/P31 Express DRAM Controller
pci@0000:00:01.0            bridge     QEMU PCIe Root port
pci@0000:01:00.0            network    Virtio network device
virtio@1          enp1s0    network    Ethernet interface
pci@0000:00:01.1            bridge     QEMU PCIe Root port
pci@0000:02:00.0            bus        QEMU XHCI Host Controller
pci@0000:00:01.2            bridge     QEMU PCIe Root port
pci@0000:03:00.0            storage    Virtio block device
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
virtio@2          /dev/vda  disk       10GB Virtual I/O device
...output omitted...
```

Use the **`lshw -C system`** command to list **`system`** class information.

```
[root@host ~]# lshw -C system
host.example.com
    description: Computer
    product: KVM
    vendor: Red Hat
    version: RHEL-7.6.0 PC (Q35 + ICH9, 2009)
    width: 64 bits
    capabilities: smbios-2.8 dmi-2.8 smp vsyscall32
    configuration: boot=normal family=Red Hat Enterprise Linux
                   uuid=A6FF4E62-FA8A-DC44-8E71-379E8766D790
  *-pnp00:00
       product: PnP device PNP0b00
       physical id: 1
       capabilities: pnp
       configuration: driver=rtc_cmos
```

Reporting Hardware Errors

The user-space daemon **`rasdaemon`** catches and handles all *reliability, availability, and serviceability (RAS)* error events generated by the kernel tracing mechanism. These trace events are logged in **`/sys/kernel/debug/tracing`**, and reported by **`syslog/journald`**.

To use **`rasdaemon`**, install the *rasdaemon* package, and then start and enable the **`rasdaemon`** service.

```
[root@host ~]# yum install rasdaemon
...output omitted...

[root@host ~]# systemctl enable --now rasdaemon
Created symlink /etc/systemd/system/multi-user.target.wants/rasdaemon.service → /
usr/lib/systemd/system/rasdaemon.service.
```

Important

The **`rasdaemon`** utilizes 100 % CPU on each enabled core if the kernel boots with the **`mds=full,nosmt`** option.

The **`ras-mc-ctl`** command provides a means to work with *error detection and correction (EDAC)* drivers. Enter the following command to see a list of command options:

```
[root@host ~]# ras-mc-ctl --help
Usage: ras-mc-ctl [OPTIONS...]
 --quiet            Quiet operation.
 --mainboard        Print mainboard vendor and model for this hardware.
 --status           Print status of EDAC drivers.
 --print-labels     Print Motherboard DIMM labels to stdout.
 --guess-labels     Print DMI labels, when bank locator is available.
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
 --register-labels  Load Motherboard DIMM labels into EDAC driver.
 --delay=N          Delay N seconds before writing DIMM labels.
 --labeldb=DB       Load label database from file DB.
 --layout           Display the memory layout.
 --summary          Presents a summary of the logged errors.
 --errors           Shows the errors stored at the error database.
 --error-count      Shows the correcte
```

The following output is an example summary of memory controller events:

```
[root@host ~]# ras-mc-ctl --summary
Memory controller events summary:
        Corrected on DIMM Label(s): 'CPU_SrcID#0_Ha#0_Chan#0_DIMM#0'
        location: 0:0:0:-1 errors: 1

No PCIe AER errors.

No Extlog errors.
MCE records summary:
        1 MEMORY CONTROLLER RD_CHANNEL0_ERR Transaction: Memory read error errors
        2 No Error errors
```

The following output is an example list of errors reported by the memory controller:

```
[root@host ~]# ras-mc-ctl --errors
Memory controller events:
1 3172-02-17 00:47:01 -0500 1 Corrected error(s): memory read error at
 CPU_SrcID#0_Ha#0_Chan#0_DIMM#0 location: 0:0:0:-1, addr 65928, grain 7, syndrome
 0  area:DRAM err_code:0001:0090 socket:0 ha:0 channel_mask:1 rank:0

No PCIe AER errors.

No Extlog errors.

MCE events:
1 3171-11-09 06:20:21 -0500 error: MEMORY CONTROLLER RD_CHANNEL0_ERR
 Transaction: Memory read error, mcg mcgstatus=0, mci Corrected_error,
 n_errors=1, mcgcap=0x01000c16, status=0x8c00004000010090, addr=0x1018893000,
 misc=0x15020a086, walltime=0x57e96780, cpuid=0x00050663, bank=0x00000007
2 3205-06-22 00:13:41 -0400 error: No Error, mcg mcgstatus=0, mci Corrected_error
 Error_enabled, mcgcap=0x01000c16, status=0x9400000000000000, addr=0x0000abcd,
 walltime=0x57e967ea, cpuid=0x00050663, bank=0x00000001
3 3205-06-22 00:13:41 -0400 error: No Error, mcg mcgstatus=0, mci Corrected_error
 Error_enabled, mcgcap=0x01000c16, status=0x9400000000000000, addr=0x00001234,
 walltime=0x57e967ea, cpu=0x00000001, cpuid=0x00050663, apicid=0x00000002,
 bank=0x00000002
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

`dmesg`(1), **`lscpu`**(1), **`lstopo`**(1), **`getconf`**(1p) **`dmidecode`**(8), **`lspci`**(8), **`lsusb`**(8), **`lshw`**(1), **`rasdaemon`**(8), and **`ras-mc-ctl`**(8) man pages

Portable Hardware Locality (hwloc)
https://www.open-mpi.org/projects/hwloc/

dmidecode Project Page
http://savannah.nongnu.org/projects/dmidecode/

The original graphic by Brendan Gregg can be found at
Linux Static Performance Tools
http://www.brendangregg/linuxperf.html/
It is licensed under Attribution-ShareAlike 4.0 International
https://creativecommons.org/licenses/by-sa/4.0/legalcode

Guided Exercise

Displaying Physical Resources

In this exercise, you will use various utilities to list your system hardware resources.

Outcome(s)

You should be able to create a profile of hardware resources for a system.

Before You Begin

Log in as the **student** user on **workstation**, using **student** as the password.

Run the **lab viewing-physical start** command. This command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab viewing-physical start
```

When prompted for a password by the **sudo** command, enter **student** as the password for the **student** user.

1. From a terminal session on **workstation**, use the **ssh** command to log in to **servera** as the **student** user. The systems are configured to use SSH keys for authentication, therefore, a password is not required to log in to **servera**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$
```

2. Create a hardware profile at **/home/student/hwprofile.txt** on **servera** that contains **DMI**, **CPU Model Name**, **L1d**, **L1i** and **L2** CPU cache size, **BIOS Revision**, the USB and PCI device lists, and the disks and memory size.

> **Note**
> The output of the following commands may vary because of the underlying hardware.

2.1. Find the **DMI** entry in the output of **dmesg**.

```
[student@servera ~]$ dmesg
...output omitted...
[    0.000000] DMI: Red Hat KVM, BIOS 1.11.1-3.module+el8+2529+a9686a4d 04/01/2014
...output omitted...
```

Record the DMI in the hardware report file **/home/student/hwprofile.txt**.

```
DMI:
Red Hat KVM, BIOS 1.11.1-3.module+el8+2529+a9686a4d 04/01/2014
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

2.2. Find the **`Model Name`** entry in the output of the **`lscpu`** command.

```
[student@servera ~]$ lscpu
...output omitted...
Model name:          Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
...output omitted...
```

Record the information in the hardware report file **`/home/student/hwprofile.txt`**.

```
CPU:
Model name:          Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
```

2.3. Find the **`L1d`**, **`L1i`**, and **`L2`** cache size entries in the output of **`lscpu`**.

```
[student@servera ~]$ lscpu
...output omitted...
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
...output omitted...
```

Record the information in the hardware report file **`/home/student/hwprofile.txt`**.

```
CPU Cache:
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
```

2.4. Find the **`BIOS Revision`** entry in the output of **`dmidecode`**.

```
[student@servera ~]$ sudo dmidecode
...output omitted...
 BIOS Revision: 0.0
...output omitted...
```

Record the information in the hardware report file **`/home/student/hwprofile.txt`**.

```
BIOS:
BIOS Revision: 0.0
```

2.5. List the USB hardware available on **`servera`**.

```
[student@servera ~]$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Record the information in the hardware report file **`/home/student/hwprofile.txt`**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
USB hardware list:
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

2.6. Display the list of PCI hardware.

```
[student@servera ~]$ lspci
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
00:01.0 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.1 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.2 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.3 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.4 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.5 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.6 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:01.7 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
00:02.0 VGA compatible controller: Red Hat, Inc. Virtio GPU (rev 01)
00:1f.0 ISA bridge: Intel Corporation 82801IB (ICH9) LPC Interface Controller (rev
 02)
00:1f.2 SATA controller: Intel Corporation 82801IR/IO/IH (ICH9R/DO/DH) 6 port SATA
 Controller [AHCI mode] (rev 02)
00:1f.3 SMBus: Intel Corporation 82801I (ICH9 Family) SMBus Controller (rev 02)
01:00.0 Ethernet controller: Red Hat, Inc. Virtio network device (rev 01)
02:00.0 USB controller: Red Hat, Inc. QEMU XHCI Host Controller (rev 01)
03:00.0 SCSI storage controller: Red Hat, Inc. Virtio block device (rev 01)
04:00.0 SCSI storage controller: Red Hat, Inc. Virtio block device (rev 01)
05:00.0 SCSI storage controller: Red Hat, Inc. Virtio block device (rev 01)
06:00.0 SCSI storage controller: Red Hat, Inc. Virtio block device (rev 01)
07:00.0 Unclassified device [00ff]: Red Hat, Inc. Virtio memory balloon (rev 01)
```

Record the PCI hardware list in the hardware report file **/home/student/hwprofile.txt**.

```
PCI hardware list:
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
00:01.0 PCI bridge: Red Hat, Inc. QEMU PCIe Root port
...output omitted...
```

2.7. Find the disks and memory size in the output of the **lshw** command. Use the **-C disk** and **-C memory** options.

```
[student@servera ~]$ sudo lshw -C disk
  *-virtio2
       description: Virtual I/O device
       physical id: 0
       bus info: virtio@2
       logical name: /dev/vda
       size: 10GiB (10GB)
       capabilities: partitioned partitioned:dos
       configuration: driver=virtio_blk logicalsectorsize=512 sectorsize=512
 signature=16a1e057
  *-virtio3
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
       description: Virtual I/O device
       physical id: 0
       bus info: virtio@3
       logical name: /dev/vdb
       size: 5GiB (5368MB)
       configuration: driver=virtio_blk logicalsectorsize=512 sectorsize=512
  *-virtio4
       description: Virtual I/O device
       physical id: 0
       bus info: virtio@4
       logical name: /dev/vdc
       size: 5GiB (5368MB)
       configuration: driver=virtio_blk logicalsectorsize=512 sectorsize=512
  *-virtio5
       description: Virtual I/O device
       physical id: 0
       bus info: virtio@5
       logical name: /dev/vdd
       size: 5GiB (5368MB)
       configuration: driver=virtio_blk logicalsectorsize=512 sectorsize=512
```

```
[student@servera ~]$ sudo lshw -C memory
  *-firmware
       description: BIOS
       vendor: SeaBIOS
       physical id: 0
       version: 1.11.1-3.module+el8+2529+a9686a4d
       date: 04/01/2014
       size: 96KiB
  *-memory
       description: System Memory
       physical id: 1000
       size: 2GiB
       capacity: 2GiB
       capabilities: ecc
       configuration: errordetection=multi-bit-ecc
     *-bank
          description: DIMM RAM
          vendor: Red Hat
          physical id: 0
          slot: DIMM 0
          size: 2GiB
```

Record the disks and memory size in the hardware report file **/home/student/hwprofile.txt**.

```
Disks size:
  *-virtio2
       description: Virtual I/O device
       physical id: 0
       bus info: virtio@2
       logical name: /dev/vda
       size: 10GiB (10GB)
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
        capabilities: partitioned partitioned:dos
        configuration: driver=virtio_blk logicalsectorsize=512 sectorsize=512
 signature=16a1e057
...output omitted...
```

```
Memory size:
  *-firmware
        description: BIOS
        vendor: SeaBIOS
        physical id: 0
        version: 1.11.1-3.module+el8+2529+a9686a4d
        date: 04/01/2014
        size: 96KiB
...output omitted...
```

2.8. Log out from the **servera** terminal session to return to workstation as student user.

```
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab viewing-physical finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab viewing-physical finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Displaying Resources in Virtual and Cloud Instances

Objectives

After completing this section, you should be able to describe the techniques used to list resources in virtual and cloud instances.

Viewing Hardware Resources From a Host

In virtualization environments, the host server provides the physical hardware used by virtual machines on that host.

KVM

Kernel-based Virtual Machine (KVM) is a loadable kernel module that provides full virtualization through the use of the Intel VT or AMD-V hardware extensions. KVM itself runs in kernel space; however, the guests run on the host as individual *Quick Emulator (QEMU)* processes in user space. KVM allows a host to make its physical hardware available to virtual machines.

The **`libvirt`** toolkit facilitates management of virtual machines and associated virtual devices. The **`virsh`** command uses the **`libvirt`** library and API to access virtual machines.

QEMU

QEMU is a multi-platform emulator that provides full system emulation, including one or more processors and peripherals. In conjunction with KVM and a processor with virtualization extensions, QEMU provides full hardware assisted virtualization. QEMU presents the virtual devices to the operating system of the guest virtual machine for enumeration, including:

- Networking PCI controller (virtio-net-pci)
- Storage PCI SCSI controller (virtio-scsi-pci)
- Storage PCI block controller (virtio-blk)
- Memory PCI controller (virtio-balloon-pci)
- Random Number Generator PCI device (virtio-rng-pci)
- USB root hubs (e.g., ich9-usb-uhci3 or pci-ohci)
- Network device drivers (e.g., e1000 or virtio)
- Video devices (e.g., cirrus-vga or qxl-vga)

Q35 is the most recent QEMU emulation. Modeled on the ICH9 chipset, this is the recommended machine type for KVM guests. Moving to the ICH9 architecture from the older i440fx design added major improvements, including an integrated AHCI controller, USB controller, network adapter, audio adapter, PCI-E and PCI bus, and LPC bus for SuperIO devices. It is significant that Q35 now supports PCIe devices, and no longer supports an ISA bus.

Use the **`virsh dumpxml`** command to determine if the guest is configured to use the QEMU Q35 machine type. The output will vary depending on machine capabilities.

```
[root@host ~]# virsh dumpxml guest |grep "q35"
<type arch='x86_64' machine='pc-q35-rhel7.6.0'>hvm</type>
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Pass-through Host Devices

Some hardware platforms support *PCI passthrough*, in which KVM guests directly access specific host hardware devices. Passthrough appears as a guest operating system device assignment, behaving as if the device is directly attached to the guest's virtual machine. PCI passthrough allows guests to have exclusive access to an assigned PCI device.

The CPU passthrough behaves differently than PCI passthrough. A KVM guest setting, *host passthrough*, uses the same CPU model features as the host on which it resides. A guest setting, *host model*, uses the same model family, but without forcing all CPU features. In both examples, the CPU is not directly assigned to the guest, but only shares the CPU feature configuration. CPU device assignment is covered in a later chapter when discussing CPU pinning.

Use the **`virsh dumpxml`** command to determine if host CPU features are visible to a guest. Verify that the **`cpu mode`** element for the guest is set to **`host-passthrough`**.

```
[root@host ~]# virsh dumpxmlguest |grep "cpu mode"
<cpu mode='host-passthrough'>
```

Single root input/output virtualization (SR-IOV) is a device assignment method by which a single physical device is shared by multiple virtual machines. SR-IOV provides an independent memory space, interrupts, and *Direct Memory Access (DMA)* streams for each guest, such that the hypervisor is not needed to manage data transfers. SR-IOV enables a *Single Root Function* where, for example, a single Ethernet port appears as multiple, unique physical devices. A physical device with SR-IOV capabilities can be configured to appear as multiple functions, with each having its own configuration space.

Memory Ballooning

The **`virtio-balloon`** driver controls the amount of memory a guest possesses. The balloon device designates the unused RAM of a virtual machine (VM), and makes it available for use by the host or other VMs. When a VM requires more memory, virtio-balloon reassigns memory back to the VM, up to the configured size of the VM. Memory ballooning efficiently manages unused guest memory, while still making the full defined size available to the guest.

Memory ballooning is ideal for applications that allocate large amounts of memory, and have the physical memory guaranteed set to the same value as the defined memory. Memory ballooning is also useful for applications that consume memory slowly, occasionally release memory, or stay idle for long periods of time, such as virtual desktops.

A virtual memory balloon device is automatically added to all KVM guest virtual machines, and managed by **`libvirt`**. In the XML configuration file, memory ballooning is represented by the **`<memballoon>`** element. It is not necessary to explicitly add this element to the XML of a guest virtual machine, unless you are assigning a specific PCI slot. If the **`<memballoon>`** device must be explicitly disabled, set the **`model='none'`** parameter in the XML file.

Use the **`lspci`** command to verify **`virtio-balloon`** support.

```
[root@host ~]# lspci | grep balloon
07:00.0 Unclassified device [00ff]: Red Hat, Inc. Virtio memory balloon (rev 01)
```

Verify the balloon device setting in the XML configuration file.

```
[root@host images]# grep balloon guest.xml
<memballoon model='virtio'>
  <alias name='balloon0'/>
</memballoon>
```

Right-sizing a Virtual Machine Instance

The process for right-sizing a virtual machine instance is to plan, install, and configure the right amount of resources to keep costs under control and ensure optimal performance and scalability. Right-sized VM instances should have the minimum number of packages installed, and run only the services necessary for the desired workload. As a cloud computing design, instances are right-sized and tuned for that particular size, and then scaled horizontally to almost any capacity.

Monitoring host and guest related issues in your virtual and cloud environments is the most effective way to optimize performance. Continuously right-sizing is how you maintain consistency, and control optimal performance and operational costs. Observing guest and host behavior, such as idle instances or heavy workloads, is an ongoing effort.

Observing Guest and Host Events

Using **`kvm_stat`** to observe guest behavior from the host perspective can identify heavy workloads that are candidates for right-sizing. The **`perf kvm`** command records performance counters for guests and host systems, providing the necessary information to analyze performance issues.

kvm_stat

The **`kvm_stat`** command diagnoses performance related issues for guests running on the host. By default, the **`kvm_stat`** command reports statistics for all running guests and the host. However, interactive keys are available to isolate statistics to a particular guest.

On the host system, use the **`kvm_stat`** command to view runtime statistics from the **`kvm`** kernel module and user space. Output varies between systems and command execution.

```
[root@host ~]# kvm_stat
kvm statistics - summary

 Event                                          Total %Total CurAvg/s
 kvm_entry                                         17   18.9       85
 kvm_exit                                          17   18.9       85
 kvm_hv_timer_state                                14   15.6       70
 kvm_msr                                           11   12.2       55
 kvm_inj_virq                                       5    5.6       25
 kvm_apic_accept_irq                                5    5.6       25
 kvm_eoi                                            5    5.6       25
 kvm_pv_eoi                                         5    5.6       25
 kvm_vcpu_wakeup                                    4    4.4       20
 kvm_apic                                           2    2.2       10
 kvm_fpu                                            2    2.2       10
 kvm_pio                                            1    1.1        5
 kvm_apic_ipi                                       1    1.1        5
 kvm_userspace_exit                                 1    1.1        5
 Total                                             90             449
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The entry and exit events show transitions between the kvm host and the user space guest. High percentage values for entry and exit events are an indication of potentially excessive CPU overhead. Press the **x** key to toggle system-wide kernel and user space statistics.

```
kvm statistics - summary

 Event                                          Total %Total CurAvg/s
 kvm_entry                                      95940   20.7      127
 kvm_exit                                       95940   20.7      127
   kvm_exit(MSR_WRITE)                          56769   59.2       84
   kvm_exit(HLT)                                17557   18.3       33
   kvm_exit(IO_INSTRUCTION)                      2905    3.0        5
   kvm_exit(MSR_READ)                            1304    1.4        2
   kvm_exit(PENDING_INTERRUPT)                    743    0.8        2
   kvm_exit(EXTERNAL_INTERRUPT)                  8791    9.2        1
   kvm_exit(CPUID)                                675    0.7        0
   kvm_exit(EPT_MISCONFIG)                         97    0.1        0
 kvm_hv_timer_state                             73413   15.9      114
 kvm_msr                                        58073   12.5       86
 kvm_apic_accept_irq                            25243    5.5       35
 kvm_inj_virq                                   25242    5.5       35
 kvm_eoi                                        25237    5.4       35
 kvm_pv_eoi                                     24588    5.3       34
 kvm_vcpu_wakeup                                17490    3.8       33
 kvm_apic                                        5261    1.1       12
 kvm_fpu                                         5822    1.3       10
 kvm_userspace_exit                              2911    0.6        5
   kvm_userspace_exit(IO)                        2905   99.8        5
   kvm_userspace_exit(MMIO)                         6    0.2        0
 kvm_pio                                         2905    0.6        5
 kvm_apic_ipi                                    2260    0.5        5
...output omitted...
```

Press the **g** key to activate the interface to select a specific guest.

```
Show statistics for specific guest or pid.
This might limit the shown data to the trace statistics.

Guest or pid [ENTER exits]: guest-vm

       Pid  Guest Name (fuzzy list, might be inaccurate!)
      5273  guest-vm
```

```
kvm statistics - pid 5273 (guest-vm)

 Event                                          Total %Total CurAvg/s
 kvm_entry                                        343   19.6      106
 kvm_exit                                         343   19.6      106
   kvm_exit(MSR_WRITE)                            225   65.6       69
   kvm_exit(HLT)                                   82   23.9       26
   kvm_exit(IO_INSTRUCTION)                        17    5.0        5
   kvm_exit(EXTERNAL_INTERRUPT)                     9    2.6        3
   kvm_exit(MSR_READ)                               8    2.3        2
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
  kvm_exit(PENDING_INTERRUPT)                    1    0.3        0
kvm_hv_timer_state                             307   17.5       95
kvm_msr                                        233   13.3       71
kvm_inj_virq                                    84    4.8       27
kvm_eoi                                         84    4.8       27
kvm_pv_eoi                                      82    4.7       26
kvm_apic_accept_irq                             82    4.7       26
kvm_vcpu_wakeup                                 82    4.7       26
kvm_fpu                                         34    1.9       11
kvm_apic                                        23    1.3        7
kvm_pio                                         17    1.0        5
kvm_userspace_exit                              17    1.0        5
  kvm_userspace_exit(IO)                        17  100.0        5
kvm_apic_ipi                                    10    0.6        3
kvm_set_irq                                      4    0.2        1
kvm_ioapic_set_irq                               4    0.2        1
kvm_pic_set_irq                                  2    0.1        1
kvm_ack_irq                                      1    0.1        0
Total                                         1752             544
```

Press the **q** key to exit the **kvm_stat** command.

Analyzing Guest Performance Metrics

Use performance metrics to view functions that generate high amounts of overhead in the code. Identifying where the excess overhead occurs is a starting point for optimizing performance. To analyze performance metrics of a guest virtual machine, copy the contents of two guest device files to the host.

/proc/modules
: Loaded kernel modules. Running the **lsmod** will display these modules in a user-friendly format.

/proc/kallsyms
: Kernel debugging symbols. Symbols are variable names, function names, and paths to source code.

> **Note**
> Directly coping device files will result in zero-length files, providing no information. Therefore, to copy the content of these files, redirect the output to temporary files before transferring those files to the host for analysis.

As **root** user on the guest VM, redirect the contents of **/proc/modules** and **/proc/kallsyms** files to arbitrary files in the **/tmp** directory. Then copy the temporary files to the host.

```
[root@guest ~]# cat /proc/modules > /tmp/modules
[root@guest ~]# cat /proc/kallsyms > /tmp/kallsyms
[root@host ~]# scp root@guest:/tmp/modules guest-modules
[root@host ~]# scp root@guest:/tmp/kallsyms guest-kallsyms
```

Use the **perf kvm record** command to record guest events from both files.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# perf kvm --guest --guestmodules=guest-modules \
--guestkallsyms=guest-kallsyms record -a
```

Press **Ctrl**+**C** to stop recording data.

Use the **perf kvm report** to generate a report based on guest events from both **guest-modules** and **guest-kallsyms** files. For easy viewing and parsing, redirect the output to another file.

```
[root@host ~]# perf kvm --guest --guestmodules=guest-modules \
--guestkallsyms=guest-kallsyms report --force > guest-analyze
```

View the generated file.

```
[root@host ~]# vim guest-analyze
# To display the perf.data header info, please use --header/--header-only options.
#
# Total Lost Samples: 0
#
# Samples: 108  of event 'cycles'
# Event count (approx.): 11370560
#
# Overhead  Command  Shared Object            Symbol
# ........  .......  .......................  ....................................
#
     4.27%  :5293    [guest.kernel.kallsyms]  [g] pvclock_clocksource_read
     3.43%  :5291    [guest.kernel.kallsyms]  [g] pvclock_clocksource_read
     2.62%  :5293    [guest.kernel.kallsyms]  [g] kmem_cache_alloc
     2.49%  :5291    [guest.kernel.kallsyms]  [g] kvm_clock_get_cycles
     2.48%  :5293    [guest.kernel.kallsyms]  [g] security_compute_sid.part.17
     2.28%  :5293    [guest.kernel.kallsyms]  [g]
 reservation_object_add_shared_fence
     2.19%  :5291    [guest.kernel.kallsyms]  [g] tick_program_event
     2.14%  :5293    [guest.kernel.kallsyms]  [g] kvm_clock_get_cycles
     2.09%  :5293    [guest.kernel.kallsyms]  [g] rcu_check_callbacks
     2.07%  :5293    [guest.kernel.kallsyms]  [g] native_write_msr
     2.01%  :5293    [drm]                    [g] drm_mm_insert_node_in_range
     1.75%  :5291    [guest.kernel.kallsyms]  [g] update_load_avg
     1.75%  :5293    [guest.kernel.kallsyms]  [g] alloc_inode
     1.73%  :5293    [guest.kernel.kallsyms]  [g] seq_put_decimal_ull_width
     1.67%  :5293    [guest.kernel.kallsyms]  [g] irqtime_account_irq
     1.67%  :5293    [guest.kernel.kallsyms]  [g] memcg_kmem_put_cache
     1.63%  :5291    [guest.kernel.kallsyms]  [g] update_wall_time
     1.44%  :5293    [guest.kernel.kallsyms]  [g] cpus_share_cache
     1.44%  :5291    [guest.kernel.kallsyms]  [g] run_timer_softirq
     1.34%  :5293    [guest.kernel.kallsyms]  [g] show_interrupts
```

Generate a second report based on guest and host events from both **guest-modules** and **guest-kallsyms** files.

```
[root@host ~]# perf kvm --guest --host --guestmodules=guest-modules \
--guestkallsyms=guest-kallsyms report --force > guest-host-analyze
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

View the generated file.

```
[root@host ~]# vim guest-host-analyze
# To display the perf.data header info, please use --header/--header-only options.
#
# Total Lost Samples: 0
#
# Samples: 550  of event 'cycles'
# Event count (approx.): 137688512
#
# Overhead  Command          Shared Object               Symbol
# ........  ...............  ..........................  ...........................
#
    13.51%  CPU 1/KVM        [kernel.kallsyms]           [k] vmx_vcpu_run
     6.49%  CPU 0/KVM        [kernel.kallsyms]           [k] vmx_vcpu_run
     1.94%  swapper          [kernel.kallsyms]           [k]
 update_blocked_averages
     1.38%  swapper          [kernel.kallsyms]           [k] apic_timer_interrupt
     1.18%  CPU 1/KVM        [kernel.kallsyms]           [k] vcpu_enter_guest
     1.14%  swapper          [kernel.kallsyms]           [k]
 tick_check_oneshot_broadcast_this_cpu
     1.05%  swapper          [kernel.kallsyms]           [k] native_write_msr
     1.01%  CPU 1/KVM        [kernel.kallsyms]           [k] native_write_msr
     0.98%  swapper          [kernel.kallsyms]           [k] _raw_spin_lock_irqsave
     0.96%  swapper          [kernel.kallsyms]           [k] pvclock_gtod_notify
     0.94%  swapper          [kernel.kallsyms]           [k] native_safe_halt
     0.91%  swapper          [kernel.kallsyms]           [k] __switch_to
     0.88%  swapper          [kernel.kallsyms]           [k] timerqueue_add
     0.72%  :5293            [guest.kernel.kallsyms]     [g] apic_timer_interrupt
     0.72%  :5293            [guest.kernel.kallsyms]     [g]
 security_compute_sid.part.17
     0.71%  swapper          [kernel.kallsyms]           [k]
 __raw_callee_save___pv_queued_spin_unlock
     0.66%  swapper          [kernel.kallsyms]           [k] interrupt_entry
     0.60%  CPU 1/KVM        [kernel.kallsyms]           [k] vmx_vcpu_load
     0.58%  swapper          [kernel.kallsyms]           [k]
 smp_apic_timer_interrupt
     0.57%  swapper          [kernel.kallsyms]           [k] enqueue_entity
     0.55%  swapper          [kernel.kallsyms]           [k] find_next_bit
     0.54%  xfsaild/vda1     [kernel.kallsyms]           [k]
 xfs_trans_ail_cursor_first
     0.54%  kworker/0:0-eve  [kernel.kallsyms]           [k] __kmalloc
     0.52%  swapper          [kernel.kallsyms]           [k] update_wall_time
     0.50%  SPICE Worker     qemu-kvm                    [.]
 xen_hvm_modified_memory
     0.50%  CPU 0/KVM        [kernel.kallsyms]           [k]
 copy_user_enhanced_fast_string
     0.48%  SPICE Worker     [kernel.kallsyms]           [k] entry_SYSCALL_64
     0.47%  CPU 1/KVM        [kernel.kallsyms]           [k] vmx_pre_block
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

`kvm_stat`(1), **`perf-kvm`**(1), and **`virsh`**(1) man pages.

KVM site
https://www.linux-kvm.org/page/Main_Page

QEMU site
https://www.qemu.org/

QEMU information wiki
https://wiki.qemu.org/Main_Page

libvirt Virtualization API
https://libvirt.org/

Virtio
https://wiki.libvirt.org/page/Virtio

For more information, refer to the *Appendices* A.4. kvm_stat Section in the *Red Hat Enterprise Linux 7 Virtualization Deployment and Administration Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/virtualization_deployment_and_administration_guide/index#sect-Troubleshooting-kvm_stat

Guided Exercise

Displaying Resources in Virtual and Cloud Instances

In this exercise, you will use various techniques to list resources in virtual and cloud instances.

Outcomes

You should be able to interpret the output of various commands that display resources of virtual and cloud instances.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab viewing-virtual start** command. The command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab viewing-virtual start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Use the **virsh** command to ensure that the **rhel8-lab** VM is available and in a **running** state.

   2.1. List available VMs and their state.

```
[root@servera ~]# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     rhel8-lab                      shut off
```

   2.2. Start the **rhel8-lab** VM.

```
[root@servera ~]# virsh start rhel8-lab
Domain rhel8-lab started
```

3. Display detailed host and guest CPU information to determine the number of host CPUs available to **guest** VMs, and then confirm that the **rhel8-lab** VM is balanced between them, and therefore, not pinned to a single CPU.

   3.1. List host CPUs available to guest VMs.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# lscpu | grep -i '^cpu'
CPU op-mode(s):      32-bit, 64-bit
CPU(s):              2
CPU family:          6
CPU MHz:             3408.000
```

3.2. There are 2 host CPUs available to guest VMs. List virtual CPU information for the **rhel8-lab** VM to determine that it is balanced between the two hosts, and not pinned to a CPU.

```
[root@servera ~]# virsh vcpuinfo rhel8-lab
VCPU:           0
CPU:            1
State:          running
CPU time:       70.1s
CPU Affinity:   yy

VCPU:           1
CPU:            1
State:          running
CPU time:       101.3s
CPU Affinity:   yy
```

The **CPU Affinity**: **yy** value means that the **rhel8-lab** VM can presently run on any CPU and is not pinned to a specific CPU.

4. Use the **top** command to observe host statistics directly related to the **rhel8-lab** VM guest running on the **servera** host. Values will vary throughout the observation period.

The following options for the **top** command will isolate output to VM guests running on the **servera** host.

- The **-H** option displays individual threads.
- The **-u qemu** option displays processes owned by the **qemu** user.
- The **-o %CPU** option sets the sort field to **%CPU**.

When the **top** command interface opens, press the **1** key to view both CPUs in the summary section, and then press the **x** key to highlight the values in the **%CPU** column.

```
[root@servera ~]# top -H -u qemu -o %CPU
top - 19:32:48 up 1 day, 21:28,  2 users,  load average: 0.02, 0.02, 0.00
Threads: 177 total,   3 running, 174 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.3 us,  0.3 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1829.0 total,     90.4 free,    900.1 used,    838.5 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    754.2 avail Mem

  PID USER    PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
32036 qemu    20   0 2593572 680160  21444 R   0.7  36.3   0:57.95 CPU 1/KVM
32034 qemu    20   0 2593572 680160  21444 S   0.3  36.3   0:34.92 CPU 0/KVM
32012 qemu    20   0 2593572 680160  21444 S   0.0  36.3   0:02.63 qemu-kvm
32021 qemu    20   0 2593572 680160  21444 S   0.0  36.3   0:00.16 qemu-kvm
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
32031 qemu   20   0 2593572 680160  21444 S   0.0  36.3   0:00.00 IO mon_iothread
32039 qemu   20   0 2593572 680160  21444 S   0.0  36.3   0:01.43 SPICE Worker
```

Press the **q** key to exit the **top** command.

5. Use the **kvm_stat** command to view runtime statistics from the kvm kernel module and user space. Use an interactive key to toggle views for both kernel and user space statistics system-wide. Use an interactive key to activate details for the **rhel8-lab** guest.

   5.1. List statistics for the entire system.

```
[root@servera ~]# kvm_stat
 kvm statistics - summary

 Event                                          Total %Total CurAvg/s
 kvm_entry                                       9784   19.7      101
 kvm_exit                                        9784   19.7      101
 kvm_hv_timer_state                              8855   17.8       92
 kvm_msr                                         6761   13.6       70
 kvm_apic_accept_irq                             2364    4.8       25
 kvm_inj_virq                                    2363    4.7       25
 kvm_eoi                                         2363    4.7       25
 kvm_pv_eoi                                      2292    4.6       25
 kvm_vcpu_wakeup                                 2328    4.7       25
 kvm_fpu                                          968    1.9       10
 kvm_userspace_exit                               484    1.0        5
 kvm_pio                                          481    1.0        5
 kvm_apic                                         473    1.0        5
 kvm_apic_ipi                                     193    0.4        2
 kvm_set_irq                                       64    0.1        1
 kvm_ioapic_set_irq                                64    0.1        1
 kvm_pic_set_irq                                   32    0.1        1
 kvm_msi_set_irq                                   52    0.1        0
 kvm_ack_irq                                       16    0.0        0
 kvm_halt_poll_ns                                  20    0.0        0
 kvm_emulate_insn                                   6    0.0        0
 kvm_mmio                                           5    0.0        0
 kvm_fast_mmio                                      3    0.0        0
 vcpu_match_mmio                                    3    0.0        0
 Total                                          49758              519
```

   5.2. Press the **x** key to toggle details of **kvm_exit** events in kernel and user space.

```
 Event                                          Total %Total CurAvg/s
 kvm_entry                                      12443   19.7       83
 kvm_exit                                       12443   19.7       83
   kvm_exit(MSR_WRITE)                           8324   66.9       53
   kvm_exit(HLT)                                 2951   23.7       19
   kvm_exit(IO_INSTRUCTION)                       619    5.0        5
   kvm_exit(EXTERNAL_INTERRUPT)                   213    1.7        4
   kvm_exit(MSR_READ)                             270    2.2        2
   kvm_exit(PENDING_INTERRUPT)                     36    0.3        0
   kvm_exit(EPT_MISCONFIG)                          6    0.0        0
 kvm_hv_timer_state                             11250   17.8       72
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
 kvm_msr                                        8594   13.6        55
 kvm_apic_accept_irq                            3003    4.7        19
 kvm_inj_virq                                   3002    4.7        19
 kvm_eoi                                        3002    4.7        19
 kvm_vcpu_wakeup                                2950    4.7        19
 kvm_pv_eoi                                     2921    4.6        18
 kvm_fpu                                        1244    2.0        11
 kvm_userspace_exit                              622    1.0         5
   kvm_userspace_exit(IO)                        619   99.5         5
   kvm_userspace_exit(MMIO)                        3    0.5         0
 kvm_pio                                         619    1.0         5
 kvm_apic                                        595    0.9         2
 kvm_set_irq                                      80    0.1         1
 kvm_ioapic_set_irq                               80    0.1         1
 kvm_apic_ipi                                    247    0.4         1
 kvm_pic_set_irq                                  40    0.1         1
 kvm_msi_set_irq                                  66    0.1         0
 kvm_ack_irq                                      20    0.0         0
 kvm_halt_poll_ns                                 22    0.0         0
 kvm_emulate_insn                                  6    0.0         0
 kvm_mmio                                          5    0.0         0
 kvm_fast_mmio                                     3    0.0         0
 vcpu_match_mmio                                   3    0.0         0
 Total                                         63260              415
```

5.3. Press the **g** key to activate the interface to select a specific guest. Enter **`rhel8-lab`** at the prompt.

```
Show statistics for specific guest or pid.
This might limit the displayed data to the trace statistics.

Guest or pid [ENTER exits]: rhel8-lab

       Pid  Guest Name (fuzzy list, might be inaccurate!)
     29164  rhel8-lab
```

```
kvm statistics - pid 32012 (rhel8-lab)

 Event                                         Total %Total CurAvg/s
 kvm_entry                                      2258   19.7        88
 kvm_exit                                       2258   19.7        88
   kvm_exit(MSR_WRITE)                          1491   66.0        58
   kvm_exit(HLT)                                 543   24.0        20
   kvm_exit(IO_INSTRUCTION)                      105    4.7         5
   kvm_exit(EXTERNAL_INTERRUPT)                   40    1.8         3
   kvm_exit(MSR_READ)                             69    3.1         2
   kvm_exit(PENDING_INTERRUPT)                     8    0.4         0
   kvm_exit(CPUID)                                 2    0.1         0
 kvm_hv_timer_state                             2030   17.7        78
 kvm_msr                                        1560   13.6        60
 kvm_inj_virq                                    551    4.8        20
 kvm_eoi                                         551    4.8        20
 kvm_pv_eoi                                      542    4.7        20
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
 kvm_vcpu_wakeup                                    543    4.7        20
 kvm_apic_accept_irq                                541    4.7        20
 kvm_fpu                                            210    1.8         9
 kvm_pio                                            105    0.9         5
 kvm_userspace_exit                                 105    0.9         5
   kvm_userspace_exit(IO)                           105  100.0         5
 kvm_apic                                           106    0.9         3
 kvm_apic_ipi                                        47    0.4         2
 kvm_set_irq                                         12    0.1         0
 kvm_ioapic_set_irq                                  12    0.1         0
 kvm_pic_set_irq                                      6    0.1         0
 kvm_ack_irq                                          3    0.0         0
 kvm_cpuid                                            2    0.0         0
 kvm_halt_poll_ns                                     2    0.0         0
 Total                                            11444              438
```

Press the **q** key to exit the **kvm_stat** command.

6. Use the **perf kvm** command to report statistics for hardware performance events on the **servera** kvm host. The **rhel8-lab** VM guest must be running to collect performance events. Values will vary between reports.

   6.1. Verify that the **rhel8-lab** VM guest is **running**.

```
[root@servera ~]# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     rhel8-lab                      running
```

   6.2. Generate a report of statistics for performance event counters on the host, which reflect a snapshot of performance during a 10 second interval. Review various counters, such as, page faults, context switches, and others.

```
[root@servera ~]# perf kvm stat -- sleep 10

 Performance counter stats for 'sleep 10':

          0.414960      task-clock:G (msec)       #    0.000 CPUs utilized
                 1      context-switches:G        #    0.002 M/sec
                 0      cpu-migrations:G          #    0.000 K/sec
                59      page-faults:G             #    0.142 M/sec
                 0      cycles:G                  #    0.000 GHz
                 0      instructions:G
                 0      branches:G                #    0.000 K/sec
                 0      branch-misses:G           #    0.00% of all branches

      10.000960832 seconds time elapsed

       0.000676000 seconds user
       0.000000000 seconds sys
```

   6.3. Log out from **servera** terminal sessions to return to workstation as student user.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **`workstation`**, run the **`lab viewing-virtual finish`** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab viewing-virtual finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Quiz

Viewing Hardware Resources

Choose the correct answers to the following questions:

1. **Which file is modified to write kernel messages to files other than, or in addition to, the default locations?**
   a. `/etc/syslog.defaults`
   b. `/etc/kmsg.conf`
   c. `/etc/rsyslog.conf`
   d. `/etc/defaults.conf`

2. **Which command displays SMBIOS/DMI information for the system?**
   a. `sysreport`
   b. `lsbios`
   c. `bonnie++`
   d. `dmidecode`

3. **Which process, when applied to a a virtual machine instance, ensures optimal performance and scalability?**
   a. right-sizing
   b. host-scheduling
   c. vm-right-scaling
   d. virtual-load-sizing

4. **Which concept is useful for applications and loads that slowly consume memory, or stay dormant for long periods of time?**
   a. `Memory-balancing`
   b. `Memory-ballooning`
   c. `VM-ballooning`
   d. `VM-balancing`

5. **When using the `perf kvm` command to analyze performance metrics of a virtual machine guest, which two files must be copied from the virtual machine guest to the virtualization host? (Choose two.)**
   a. `/proc/cpuinfo`
   b. `/proc/modules`
   c. `/proc/vmstat`
   d. `/proc/zoneinfo`
   e. `/proc/devices`
   f. `/proc/kallsyms`

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Viewing Hardware Resources

Choose the correct answers to the following questions:

1. **Which file is modified to write kernel messages to files other than, or in addition to, the default locations?**
   a. `/etc/syslog.defaults`
   b. `/etc/kmsg.conf`
   c. **`/etc/rsyslog.conf`**
   d. `/etc/defaults.conf`

2. **Which command displays SMBIOS/DMI information for the system?**
   a. `sysreport`
   b. `lsbios`
   c. `bonnie++`
   d. **`dmidecode`**

3. **Which process, when applied to a a virtual machine instance, ensures optimal performance and scalability?**
   a. right-sizing
   b. host-scheduling
   c. vm-right-scaling
   d. virtual-load-sizing

4. **Which concept is useful for applications and loads that slowly consume memory, or stay dormant for long periods of time?**
   a. `Memory-balancing`
   b. **`Memory-ballooning`**
   c. `VM-ballooning`
   d. `VM-balancing`

5. **When using the `perf kvm` command to analyze performance metrics of a virtual machine guest, which two files must be copied from the virtual machine guest to the virtualization host? (Choose two.)**
   a. `/proc/cpuinfo`
   b. **`/proc/modules`**
   c. `/proc/vmstat`
   d. `/proc/zoneinfo`
   e. `/proc/devices`
   f. **`/proc/kallsyms`**

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- Tuning a system for performance optimization starts with a hardware profile.
- Identifying hardware on a system, and understanding the theoretical capabilities, requires familiarity with commands that discover hardware resource details. Commands that profile hardware include:
  - **`dmesg`**, **`dmidecode`**
  - **`lspci`**, **`lscpu`**, **`lsusb`**, **`lshw`**, and **`lstopo`**
- Profiling system performance is essential to right-sizing virtual system instances for optimal performance and operation costs. Commands that profile performance include:
  - **`kvm_stat`** to analyze host and guest behavior.
  - **`perf kvm`** to analyze performance counters on both host and guest systems.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 4

Configuring Kernel Tunables and Tuned Profiles

Goal

Configure the operating system to tune for different workload requirements.

Objectives

- Configure settings for both the static kernel and for dynamic loadable kernel modules and drivers.
- Select an appropriate tuned profile to use, based on a system's most common workloads.
- Create customized tuned profiles for unique workloads.

Sections

- Configuring Kernel Tunables (and Guided Exercise)
- Selecting a Tuned Profile (and Guided Exercise)
- Customizing Tuned Profiles (and Guided Exercise)

Lab

Configuring Kernel Tunables and Tuned Profiles

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring Kernel Tunables

Objectives

After completing this section, you should be able to configure settings for both the static kernel and for dynamic loadable kernel modules and drivers.

Introducing the Proc File System

The **`proc`** file system is a pseudo file system that provides an interface to kernel data structures. The kernel mounts the **`proc`** file system on the **`/proc`** directory during boot time. The files in the **`/proc`** directory have permissions that match those of the parameters to which they refer, such as read-only for static information and writable for configurable settings and kernel tunables.

Files in pseudo file systems, such as **`proc`**, represent other more complex structures. Some **`/proc`** files effectively point to single kernel parameters, while other **`/proc`** files invoke routines that gather multiple data and present that information as a stream, which looks and behaves like a normal file. When permitted, editing settings in **`/proc`** files is the same as directly editing a kernel parameter field. For example, the **`vmstat`** command reads the **`/proc/meminfo`** file, which provides various information about the system memory usage.

Before the **`proc`** file system was developed, developers wrote code to access and parse many different types of kernel structures to get and set parameters, which required understanding complex kernel structures and maintaining program code to match released kernel enhancements. By providing the **`proc`** pseudo file system as a public interface with conventional file-based semantics, such as open, read, and write, programmers no longer need to program using the internal structures. Developers now write programs to access kernel parameters using stable **`/proc`** file names that remain consistent across kernel updates.

The files inside the **`/proc`** directory are created dynamically when their corresponding kernel structures are loaded with the static kernel, or when a dynamic kernel module or driver is loaded during boot or later during runtime.

Commonly Used Files and Subdirectories in `/proc`

`/proc/cpuinfo`
: Provides information about the processor.

`/proc/meminfo`
: Provides information about memory usage, both physical and swap.

`/proc/swaps`
: Provides information about swap space utilization.

`/proc/<PID>/`
: Provides information about specific processes running on the system. The name of the subdirectory in **`/proc`** is the process ID for the process.

`/proc/cmdline`
: Provides the kernel parameters and options used to boot this kernel most recently.

Introducing Kernel Tunables

The Linux kernel exposes information and tunables to user-space through the **/proc** directory. The **/proc/sys** subdirectory contains configurable parameters for tuning kernel behavior and activity. Most files in **/proc/sys** are writable by root, inheriting file permission from the writable flag on the corresponding kernel structure.

> **Important**
> Modifying files in **/proc/sys** causes immediate changes in the running system.

Although changes made for tuning purposes are expected to improve system behavior, modifying the **/proc/sys** tunable can also have detrimental effects on other production workloads and systems. For example, increasing the size of TCP memory buffers allocated by the kernel can improve network throughput. However, this change causes the kernel to use more memory for each TCP connection made or received by the system. A system with numerous network connections will consume more memory, limiting the memory available for user-space applications.

Since all **/proc/sys** file changes write directly to the kernel in memory, changes are not persistent across reboots. Kernel tunable settings will revert to a kernel module's default settings unless configured persistently in appropriate configuration files.

The **/proc/sys** directory tree contains several subdirectories for kernel tunables.

Commonly Used Subdirectories in /proc/sys

/proc/sys/dev
: Contains tunables for system devices such as RAID devices, CD-ROMs, SCSI devices, and one or more parallel ports.

/proc/sys/fs
: Contains file system related tunables, for example, parameters for quota handling.

/proc/sys/kernel
: Contains tunables that change how the kernel works internally, for example, shared memory settings.

/proc/sys/net
: Contains tunables that change networking settings, for example, receive and send socket buffer size.

/proc/sys/vm
: Contains tunables which change the virtual memory management of the kernel, for example, the number of huge pages.

Not all files in **/proc/sys** are writable. In the example below, the **osrelease** file contains the operating system version, which would only be modified during a operating system update, never manually. The **threadmax** file is writable, and can be edited to set the maximum number of simultaneous process threads.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]#ls -l /proc/sys/kernel/{osrelease,threads-max}
-r--r--r-- 1 root root 0 Jul  8 14:18 /proc/sys/kernel/osrelease
-rw-r--r-- 1 root root 0 Jul  8 14:18 /proc/sys/kernel/threads-max
[root@host ~]# cat /proc/sys/kernel/osrelease
4.18.0-80.el8.x86_64
[root@host ~]# cat /proc/sys/kernel/threads-max
14257
```

Modifying Kernel Tunables

Administrators can modify most kernel tunables on a running system to change current system or application behavior. These parameters are modified directly in memory, and will be discarded during the next reboot. To set parameter values that persist across reboots requires creating settings in persistent configuration files.

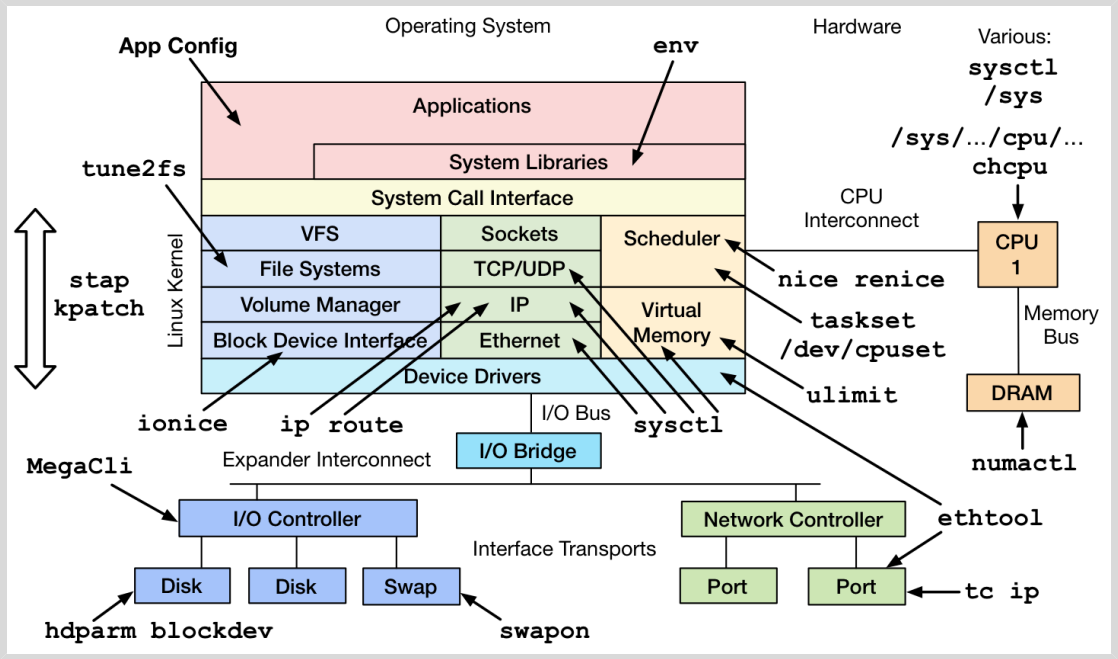


Figure 4.1: Linux Performance Tuning Tools by Brendan Gregg (CC BY-SA 4.0)

There are multiple ways to modify the kernel tunables:

- Use an editor or redirection to modify files in the **`/proc/sys`** directory.
- Use the **`sysctl`** command to set a parameter by name.
- Create entries for parameters in configuration files in the **`/etc/sysctl.d`** directory.

Modify Kernel Tunables From the Command Line

The **`root`** user can edit writable **`/proc/sys`** files, with immediate effect on the running kernel. It is common to use the **`echo`** command to redirect settings into single value tunables, but such changes will not persist across a reboot.

The dotted name of a kernel tunable matches the directory structure inside **`/proc/sys`**. For example, the **`net.ipv4.icmp_echo_ignore_all`** kernel tunable matches the **`/proc/sys/net/ipv4/icmp_echo_ignore_all`** file.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Some kernel tunables are Boolean values. For example, setting the **net.ipv4.icmp_echo_ignore_all** tunable to false (**0**) causes the kernel to process (not ignore) all ICMP echo requests and replies received. Verify the current setting of **/proc/sys/net/ipv4/icmp_echo_ignore_all**.

```
[root@host ~]# cat /proc/sys/net/ipv4/icmp_echo_ignore_all
0
```

For example, the kernel replies to ICMP echo requests by default. Send a single ICMP echo request and get a reply in return.

```
[root@host ~]# ping -c1 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.082 ms

--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.082/0.082/0.082/0.000 ms
```

Setting the net.ipv4.icmp_echo_ignore_all tunable to true (**1**) causes the kernel to ignore all ICMP requests. Edit the tunable to **1**, then send a single ICMP echo request and receive no reply in return.

```
[root@host ~]# echo '1' > /proc/sys/net/ipv4/icmp_echo_ignore_all
[rootr@host ~]# cat /proc/sys/net/ipv4/icmp_echo_ignore_all
1
[root@host ~]# ping -c1 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.

--- localhost ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Some files in **/proc/sys** hold multiple values, such as **/proc/sys/net/ipv4/tcp_rmem**. To change the entry, write all values together into the **/proc/sys** file. For example, the **net.ipv4.tcp_rmem** tunable holds multiple values:

```
[root@host ~]# cat /proc/sys/net/ipv4/tcp_rmem
4096    87380    6291456
[root@host ~]# echo '8192 87380 6291456' > /proc/sys/net/ipv4/tcp_rmem
[root@host ~]# cat /proc/sys/net/ipv4/tcp_rmem
8192 87380 6291456
```

Modify Kernel Tunables Using `sysctl`

The **sysctl** command directly modifies kernel tunables by name. The command can edit a tunable, list available tunables, and load and apply settings from a specified configuration file.

The **sysctl -a** command lists all available kernel tunables and their associated values:

```
[root@host ~]# sysctl -a
...output omitted...
vm.stat_interval = 1
vm.swappiness = 30
vm.user_reserve_kbytes = 56825
vm.vfs_cache_pressure = 100
vm.watermark_scale_factor = 10
vm.zone_reclaim_mode = 0
```

Use the **`systctl -n`** command to suppress the tunable's name and print only the value. For example, the following command prints the value of the **`vm.swappiness`** tunable:

```
[root@host ~]# sysctl -n vm.swappiness
30
```

Use the **`sysctl -w`** command to change the value of a kernel tunable in the running kernel. In the following example, the **`vm.swappiness`** tunable is set to **`10`**, overwriting the default value of **`30`**. The lower the **`vm.swappiness`** tunable value, the less aggressively the kernel writes memory content to the swap space.

```
[root@host ~]# sysctl -w vm.swappiness=10
vm.swappiness = 10
```

Modify Kernel Tunables Persistently

During booting, the kernel loads tunable settings from configuration files found in the following directories:

- **`/etc/sysctl.d/*.conf`**
- **`/run/sysctl.d/*.conf`**
- **`/usr/lib/sysctl.d/*.conf`**

Kernel tunable configuration files must end with **`.conf`** to be loaded automatically. After loading a configuration file with a specific file name from **`/etc/sysctl.d`**, the kernel ignores the same file name in **`/run/sysctl.d`** and **`/usr/lib/sysctl.d`**. The **`/usr/lib/sysctl.d`** directory contains configuration set by the vendor in the package distribution. Never modify files in **`/usr/lib/sysctl.d`**. Instead, override vendor defaults by creating matching filenames with settings in **`/etc/sysctl.d`**. Settings in these configuration files are persistent, since they are loaded during each system boot.

As an example, create a new file in the **`/etc/sysctl.d`** directory with the appropriate key-value pairs. In the below output, the **`vm.swappiness`** kernel tunable is set to a new value, **`10`**, in **`/etc/sysctl.d/swappiness.conf`**.

```
[root@host ~]# cat /etc/sysctl.d/swappiness.conf
vm.swappiness=10
```

The **`sysctl -p`** command can load a configuration file to apply settings without rebooting.

```
[root@host ~]# sysctl -p /etc/sysctl.d/swappiness.conf
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Introducing the Sysfs File System

The **sysfs** file system is also a pseudo file system, mounted on the **/sys** directory. The files under **/sys** provide access to information and parameters for devices, file systems, and other software loaded as a kernel module. Like **/proc**, the **/sys** file system presents files with permissions for the structures they represent. Some are read-only information and others are writable settings. The **sysfs** tree was once part of **/proc**, but was moved into its own file system to increase kernel stability, since device drivers are written by a wide range of contributors with different levels of experience.

Commonly Used Subdirectories in /sys

/sys/module/
: Contains information about kernel modules currently loaded in the kernel under separate subdirectories.

/sys/devices/
: Contains information about device attributes attached to a system.

/sys/bus/
: Contains information about various bus types in the kernel.

/sys/dev/
: Contains information about the block and character devices on the system.

Configuring Module Parameters

The Linux kernel is modular to allow device drivers and components to be automatically loaded only when needed, and unloaded when no longer required. Many kernel modules have modifiable settings. All kernel module settings are loaded or set to defaults when the module loads. Parameters exposed through **sysfs** can be modified on a running system.

Most kernel modules include documentation, including parameter settings, in the *kernel-doc* package. Use the **modinfo** command to query a currently loaded module. The **modinfo** command lists module information, such as its file name, license, description, and configurable parameters.

```
[root@host ~]# modinfo loop
filename:       /lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/block/loop.ko.xz
alias:          devname:loop-control
alias:          char-major-10-237
alias:          block-major-7-*
license:        GPL
rhelversion:    8.0
srcversion:     3D71AF3926DFBFA6C0C0FA9
depends:
intree:         Y
name:           loop
...output omitted...
parm:           max_loop:Maximum number of loop devices (int)
parm:           max_part:Maximum number of partitions per loop device (int)
```

Use the **-p** option to list only the module parameters.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# modinfo -p loop
parm:           max_loop:Maximum number of loop devices (int)
parm:           max_part:Maximum number of partitions per loop device (int)
```

View the current setting for a module parameter in the **/sys/module** directory when the module is loaded. In the below output, the **loop** module's **max_loop** parameter has a value of **0**. The **/sys/module/loop** directory, and its parameter files will not exist on the system until the **loop** module is loaded.

```
[root@host ~]# cat /sys/module/loop/parameters/max_loop
0
```

If a parameter file in **/sys** is writable, then a new value can be redirected into the file. To persistently set module parameters every time that the module is loaded, create a configuration file in the **/etc/modprobe.d** directory. The new file name must end in **.conf** to be loaded automatically. This file contents must follow the syntax rules found in the **modprobe.conf**(5) man page:

```
options modulename parameter=value
```

To automatically set the **loop** module's **max_loop** parameter to **5** when the module loads, add the following entry to the new **/etc/modprobe.d/loop.conf** file:

```
options loop max_loop=5
```

Parameters can also be added to the **modprobe** command line when loading modules manually. Parameters configured manually are not persistent.

```
[root@host ~]# modprobe loop max_loop=5
```

References

sysctl(1), **sysctl.conf**(5), **sysctl.d**(5), **modinfo**(8), **modprobe**(8), **modprobe.conf**(5), **proc**(5), and **sysfs**(5) man pages

The *kernel-doc* package provides descriptions and explanations for many of the available kernel tunables. Once installed, the description and possible values for the existing tunables are documented in:

/usr/share/doc/kernel-doc-*/Documentation/sysctl/*.txt

For more information, refer to the *Configuring kernel tunables* chapter in the *Red Hat Enterprise Linux 8 Managing, monitoring and updating the kernel* guide at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/managing_monitoring_and_updating_the_kernel/index

The original graphic by Brendan Gregg can be found at
Linux Performance Tuning Tools
http://www.brendangregg/linuxperf.html/
It is licensed under Attribution-ShareAlike 4.0 International
https://creativecommons.org/licenses/by-sa/4.0/legalcode

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring Kernel Tunables

In this exercise, you will configure kernel settings and loadable module parameters and make these configuration settings persistent.

Outcomes

You should be able to:

- Set the **vm.swappiness** kernel tunable persistently to **15** in the newly created file **/etc/sysctl.d/swappiness.conf** and activate without reboot on **servera**.
- Load the kernel module **usb_storage**, and set the module parameter **delay_use** to **0** persistently in the newly created file **/etc/modprobe.d/usb_storage.conf** on **servera**.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab kerneltuning-tunables start** command. The command determines if the host, **servera**, is reachable on the network.

```
[student@workstation ~]$ lab kerneltuning-tunables start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Set the **vm.swappiness** kernel tunable persistently to **15** by editing the **/etc/sysctl.d/swappiness.conf** file. Activate it without rebooting **servera**.

   2.1. Verify **vm.swappiness** is set to **30** on the **servera** system.

```
[root@servera ~]# sysctl -n vm.swappiness
30
```

   2.2. Set **vm.swappiness** to **15** in **/proc/sys/vm/swappiness**.

```
[root@servera ~]# echo 15 >> /proc/sys/vm/swappiness
```

   2.3. Verify **vm.swappiness** is set to **15** on the **servera** system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# sysctl -n vm.swappiness
15
```

2.4. Set **vm.swappiness** in **/etc/sysctl.d/swappiness.conf** file to **15** to make the configuration persistent. Create the **/etc/sysctl.d/swappiness.conf** file with the following content:

```
[root@servera ~]# vim /etc/sysctl.d/swappiness.conf
vm.swappiness=15
```

2.5. Activate the parameter set in **/etc/sysctl.d/swappiness.conf** file without rebooting **servera**.

```
[root@servera ~]# sysctl -p /etc/sysctl.d/swappiness.conf
vm.swappiness = 15
```

2.6. Verify the **vm.swappiness** tunable is set to **15** on **servera**.

```
[root@servera ~]# sysctl -n vm.swappiness
15
```

3. Load the **usb_storage** kernel module, and verify the default value of the **delay_use** parameter. Configure **delay_use** to be the value **0**.

3.1. Verify that the **usb_storage** kernel module is not loaded. The command should not return any output.

```
[root@servera ~]# lsmod | grep usb_storage
```

3.2. Verify that the **/sys/module/usb_storage** file does not exist.

```
[root@servera ~]# stat /sys/module/usb_storage
stat: cannot stat '/sys/module/usb_storage': No such file or directory
```

3.3. Load the **usb_storage** kernel module on **servera**.

```
[root@servera ~]# modprobe usb_storage
```

3.4. Verify the default value for the **delay_use** parameter. This parameter sets the time that a USB device waits to become available when first connected, to ensure it is ready.

```
[root@servera ~]# cat /sys/module/usb_storage/parameters/delay_use
1
```

3.5. Unload the **usb_storage** kernel module.

```
[root@servera ~]# modprobe -r usb_storage
```

3.6. Create the file **/etc/modprobe.d/usb_storage.conf** with the content **options usb_storage delay_use=0** to reduce the time for the USB devices to wait.

```
[root@servera ~]# vim /etc/modprobe.d/usb_storage.conf
options usb_storage delay_use=0
```

3.7. Load the **usb_storage** kernel module again on **servera**.

```
[root@servera ~]$ modprobe usb_storage
```

3.8. Verify that the value for the **delay_use** parameter is now **0**.

```
[root@servera ~]# cat /sys/module/usb_storage/parameters/delay_use
0
```

3.9. Exit **servera** to return to **workstation**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab kerneltuning-tunables finish** command to finish this exercise. The command disables the **usb_storage** kernel module on **servera**, and sets **vm.swappiness** to the default value.

```
[student@workstation ~]$ lab kerneltuning-tunables finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Selecting a Tuned Profile

Objectives

After completing this section, you should be able to select an appropriate tuned profile for use based on a system's most common workloads.

Tuning System Performance

Red Hat Enterprise Linux provides the tuning service named **`tuned`** to help system administrators optimize performance for different types of workload use cases. The **`tuned`** service can apply tuning adjustments both statically and dynamically, using tuned profiles configured to meet particular workload requirements. A number of predefined tuned profiles exist for different tuning objectives, such as power conservation or network throughput. The tuned architecture includes the ability to customize a profile's rules to meet workload requirements.

Static Tuning

The **`tuned`** service applies system settings during service startup or when switching between profiles. Static tuning consists of setting predefined **`sysctl`** and **`/sys`** parameter once. With static tuning, kernel tunables are set for overall performance expectations and are not adjusted as activity levels change.

Dynamic Tuning

With dynamic tuning, the **`tuned`** service monitors system activity and adjusts settings depending on runtime behavior changes. Dynamic tuning continuously adjusts to fit the current workload, starting with the selected tuned profile's initial settings. Dynamic tuning attempts to balance both performance and power savings, and is therefore disabled in performance boosting profiles.

For example, consider a video file that needs to be transcoded. During the transcoding job, the network connection is not needed and is mostly inactive. The network connection is required only when the transcoded video is uploaded. The network interface does not have to run at full speed all the time, as it would by default. With dynamic tuning, **`tuned`** monitors the low activity of the network interface using a tuned plug-in, and automatically lowers the interface speed to decrease power usage. When interface activity increases for an extended period, such as when the transcoded video is begins uploading, the tuned monitoring plug-in detects and resets the maximum interface speed to increase performance during the higher network activity.

Note

Dynamic tuning is disabled by default in Red Hat Enterprise Linux 8. Set **`dynamic_tuning`** to **`1`** in **`/etc/tuned/tuned-main.conf`** to enable dynamic tuning. When dynamic tuning is enabled, **`tuned`** adjusts the system tuning every 10 seconds. Set the **`update_interval`** in **`/etc/tuned/tuned-main.conf`** to change the interval, in seconds. Changes to the **`/etc/tuned/tuned-main.conf`** configuration require restarting the **`tuned`** service.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Installing and Enabling Tuned

A minimal Red Hat Enterprise Linux 8 installation includes and enables the *tuned* package by default. To install and enable the package manually:

```
[root@host ~]# yum install tuned
...output omitted...
[root@host ~]# systemctl enable --now tuned
Created symlink /etc/systemd/system/multi-user.target.wants/tuned.service → /usr/
lib/systemd/system/tuned.service.
```

Selecting a Tuning Profile

The profiles are the core of the **`tuned`** service, and consist of tunable settings configured for a specific use cases. **`Tuned`** profiles can be modified by duplicating and editing existing profiles, or by inheriting a profile and extending it. Red Hat recommends that you do not modify any predefined profiles, since some are parents to other profiles. The *tuned* package provides predefined profiles in the following categories:

Power Saving Profiles
: A power saving profile enables aggressive disk spin-down by increasing disk write-back values and disabling log syncing for the disk. On servers, it enables **`Aggressive Link Power Management`** (ALPM) power saving for SATA host adapters, disables CD-ROM polling through HAL, and activates the CPU and disk plug-ins of tuned.

Performance Boosting Profiles
: A performance boosting profile disables power saving mechanisms, enables system settings that improve the throughput performance of disk and network I/O, and disables transparent huge pages and NUMA balancing. These profiles set the CPU governor to **`performance`**.

> **Note**
> The **`performance`** CPU governor uses the highest possible CPU frequency to ensure that tasks execute faster. This governor does not make use of power saving mechanisms such as sleep or idle, and is therefore not recommended for data centers or similar large deployments.
>
> The *kernel-tools* package provides the **`cpupower`** command. Use the **`cpupower frequency-info --governors`** command to list the available CPU governors.

The following is a list of predefined profiles installed by *tuned* on Red Hat Enterprise Linux 8.

Tuned Profiles Distributed with Red Hat Enterprise Linux 8

`balanced`
: Ideal for systems that require a compromise between power saving and performance. This profile enables the CPU, disk, audio, and video plug-ins, and activates the **`conservative`** CPU governor. It uses dynamic tuning, but when activated on a system, the profile increases latency.

`desktop`
: Derived from the **`balanced`** profile. This profile provides a faster response to interactive applications.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

`throughput-performance`
Ideal for systems that require maximum throughput. This profile sets the CPU governor and energy performance bias to **`performance`**. The **`readahead`** tunable for the disk is set to a value greater than or equal to **`4096`**. The **`throughput-performance`** profile also enables kernel tunables that improve the throughput performance of disk and network I/O.

`latency-performance`
Ideal for systems that require low latency at the expense of power consumption. Like the **`throughput-performance`** profile, CPU governor and energy performance bias are set to **`performance`** to guarantee the highest clock frequency. By statically setting the clock frequency, **`latency-performance`** prevents the latency which occurs when CPU frequency switches in response to workload fluctuations.

`network-latency`
Derived from the **`latency-performance`** profile, it enables additional network tuning parameters to provide low network latency. This profile disables transparent huge pages and NUMA balancing, as well as tunes several other network-related kernel tunables to lower network latency.

`network-throughput`
Derived from the **`throughput-performance`** profile, this profile increases kernel network buffers to boost network performance for maximum network throughput.

`powersave`
Ideal for systems that require maximum power saving. This profile throttles performance to minimize actual power consumption by enabling USB auto-suspend, WiFi power saving, and **`Aggressive Link Power Management`** (ALPM) power savings for SATA host adapters.

`oracle`
Optimized for Oracle database loads based on the **`throughput-performance`** profile. The *tuned-profiles-oracle* package provides the profile.

`virtual-guest`
Tunes the system for maximum performance if it runs on a virtual machine. This profile decreases swapping processes out of virtual memory and increases disk read ahead values.

`virtual-host`
Tunes the system for maximum performance if it acts as a host for virtual machines. This profile decreases swapping processes out of virtual memory, increases disk read ahead values, and enables a more aggressive value of dirty pages write-back.

Managing Profiles from the Command Line

The **`tuned-adm`** command is used to manage the **`tuned`** service. The **`tuned-adm`** command can query current settings, list available profiles, recommend a tuned profile for the system, change profiles on a running system, and turn the tuning service on and off.

Use the **`tuned-adm active`** command to identify the currently active tuned profile on a system.

```
[root@host ~]# tuned-adm active
Current active profile: virtual-guest
```

The **`tuned-adm list`** command lists all available tuned profiles, including both built-in profiles and custom tuned profiles created by a system administrator.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# tuned-adm list
Available profiles:
- balanced
- desktop
- latency-performance
- network-latency
- network-throughput
- powersave
- throughput-performance
- virtual-guest
- virtual-host
Current active profile: virtual-guest
```

The currently active profile can be changed if it does not meet the tuning requirements for the system. Use the **`tuned-adm profile`** command to switch the active profile to a different profile based on the system's current tuning requirements.

```
[root@host ~]# tuned-adm profile throughput-performance
[root@host ~]# tuned-adm active
Current active profile: throughput-performance
```

The **`tuned-adm`** command can recommend a tuned profile for the system. This mechanism is used to determine the default profile of a system after installation.

```
[root@host ~]# tuned-adm recommend
virtual-guest
```

Note

The output of the **`tuned-adm recommend`** command is based on various system characteristics, including whether the machine is a virtual machine and other predefined machine categories selected during installation. Any virtual machine activates the **`virtual-guest`** profile, whereas any server or compute node uses the **`throughput-performance`** profile. All machines with other installation categories, such as desktop, have the **`balanced`** profile activated by default.

Any tuning activity of the **`tuned`** service can be turned off instantly with the **`tuned-adm off`** command. Use the **`tuned-adm off`** command to disable all tuned settings temporarily. On restart of the **`tuned`** service, the settings are reapplied again.

```
[root@host ~]# tuned-adm off
[root@host ~]# tuned-adm active
No current active profile.
```

Stop and disable the **`tuned`** service to permanently disable all tuned settings.

Managing Profiles with Web Console

To manage system performance profiles with web console, log in with privileged access. Click the **Reuse my password for privileged tasks** option, which permits the user to execute commands with sudo privileges to modify system performance profiles.

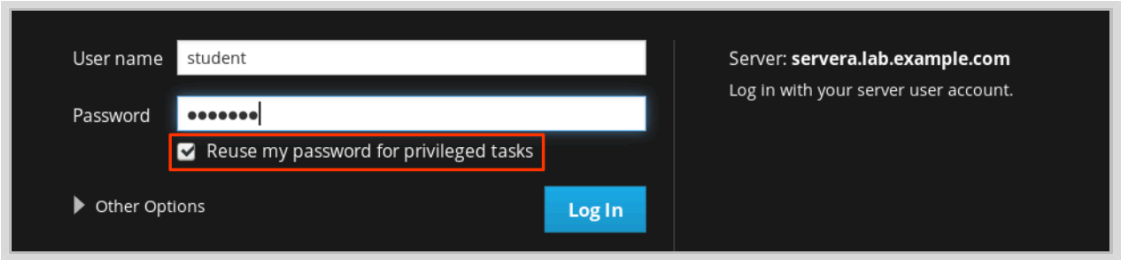


Figure 4.2: Web Console privileged login

As a privileged user, click **Systems** in the left navigation bar. The **Performance Profile** field displays the currently active tuned profile. To select a different profile, click the active profile link.

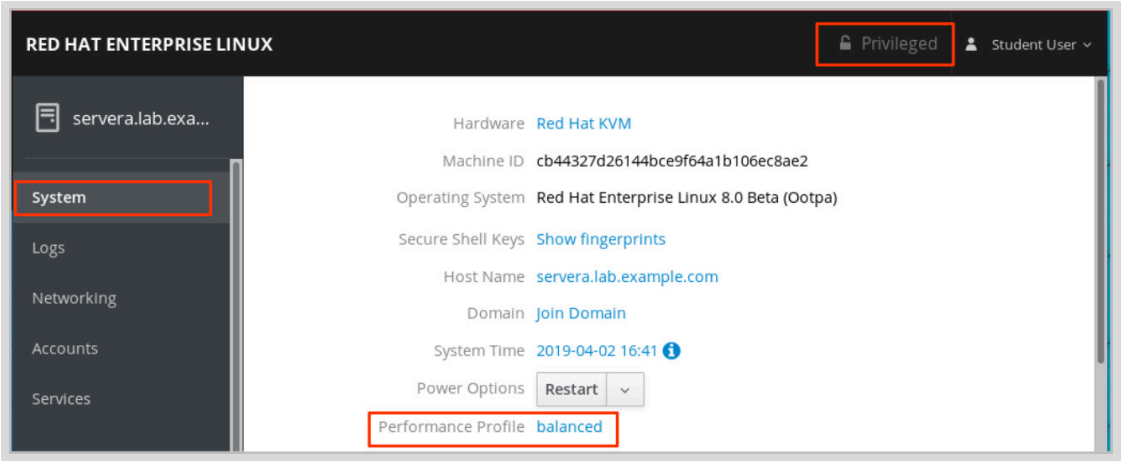


Figure 4.3: Active performance profile

In the **Change Performance Profile** user interface, scroll through the profile list to select one that best suits the system purpose.

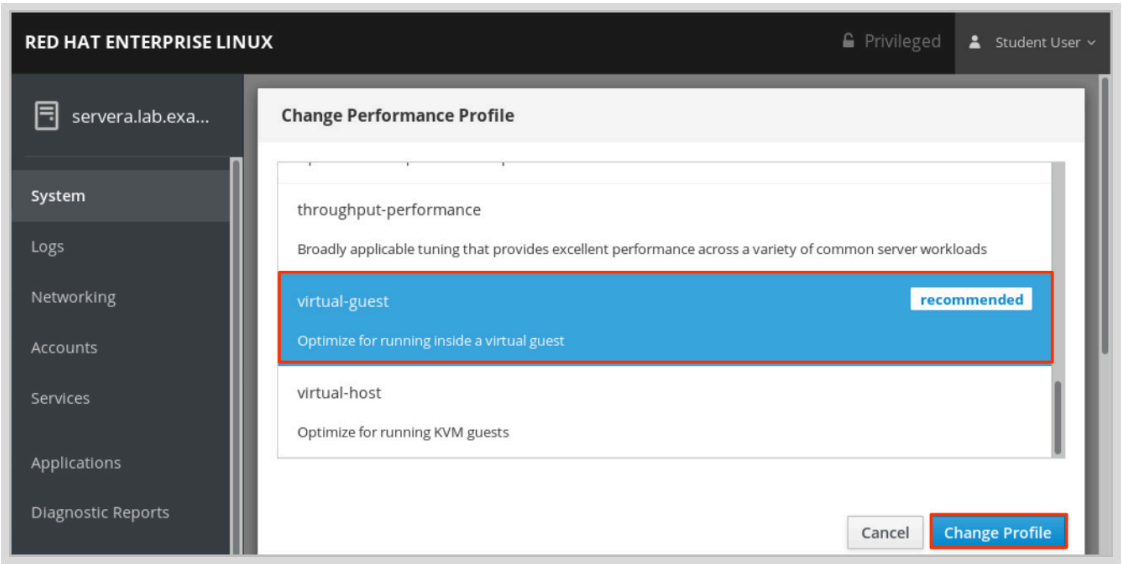


Figure 4.4: Select a preferred performance profile

To verify changes, return to the main **System** page and confirm that it displays the active profile in the **Performance Profile** field.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

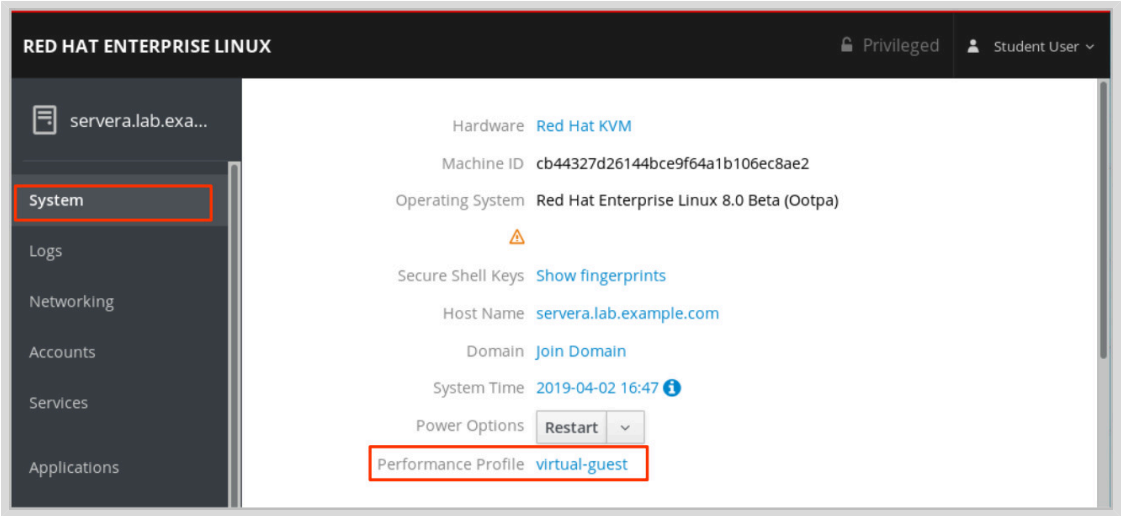


Figure 4.5: Verify active performance profile

Defining Tuned Profile

The tuned profiles support various types of mechanisms to change configuration settings, for example, **`sysctl`**, **`sysfs`**, or kernel boot parameters that are available as tuned plug-ins. A profile stores all its configurations in the following directories:

`/usr/lib/tuned/`
: This directory stores the profiles provided by Red Hat Enterprise Linux distribution. Each profile configuration is present in its own subdirectory. Each profile's main configurations is written in the **`tuned.conf`** file.

`/etc/tuned/`
: Red Hat Enterprise Linux supports custom profiles that inherit the settings of another profile and can be customized further. This directory is used to store the custom profiles. Each profile configuration is present in its own subdirectory. The **`/etc/tuned`** directory also contains the **`active_profile`** file, which contains the name of the currently active tuned profile and the **`tuned-main.conf`** configuration file for configuring dynamic tuning.

The **`tuned.conf`** file contains several sections. The first section defined is **`[main]`**. Other sections are configurations for plug-in instances. Each tuned plug-in tunes an individual subsystem and requires several parameters that populate the tuned profiles.

The output below shows the **`tuned.conf`** file of the **`oracle`** tuned profile.

```
[root@host ~]# cat /usr/lib/tuned/oracle/tuned.conf
#
# tuned configuration
#

[main]                                  (1)
summary=Optimize for Oracle RDBMS
include=throughput-performance

[sysctl]                                (2)
vm.swappiness = 10
vm.dirty_background_ratio = 3
vm.dirty_ratio = 40
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
kernel.shmmax = 4398046511104
kernel.shmall = 1073741824
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
fs.file-max = 6815744
fs.aio-max-nr = 1048576
net.ipv4.ip_local_port_range = 9000 65499
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
kernel.panic_on_oops = 1

[vm]                                  ❸
transparent_hugepages=never
```

❶ The **[main]** section allows for adding a summary, description, and inheritance of settings from an existing tuned profile. In the above example, the **oracle** profile inherits its properties from the **throughput-performance** profile.

❷ The **[sysctl]** section sets various kernel tunables managed by the **sysctl** command using the **sysctl** plug-in parameters. In the above example, the **oracle** profile sets the **vm.swappiness** tunable to **10** by modifying the **/proc/sys/vm/swappiness** file.

❸ The **[vm]** section enables or disables transparent huge pages depending on the value set for the **vm** plug-in parameter. In the above example, the **oracle** profile disables the **transparent_hugepages** parameter by modifying the **/sys/kernel/mm/transparent_hugepage/enabled** file.

The **disk** plug-in provides the **readahead** parameter which is set to tune disk I/O. The **readahead** parameter influences how much data the kernel attempts to read ahead when it sees contiguous requests come in. The amount of data the kernel attempts to read ahead starts at half of the amount configured in **/sys/block/*device*/queue/read_ahead_kb**. As the name implies, this setting is in KiB. When the kernel sees contiguous access, it increases the amount of data it reads ahead up to the maximum defined in **read_ahead_kb** or lowers the amount (going down to zero if necessary) when random access is detected.

The **blockdev** command reads and writes the **read_ahead_kb** tunable. Use the **--getra** and **--setra** options with the **blockdev** command to get and set the **readahead** values.

```
[root@host ~]# blockdev --getra /dev/vda
8192
[root@host ~]# blockdev --setra 512 /dev/vda
```

Tuned profiles support the configuration of the **readahead** parameter with sectors. The following example configures the **vdb** device with a **readahead** value of **4096** sectors.

```
[disk]
devices=vdb
readahead=4096 sectors
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Important

Unlike **`/sys/block/`*`device`*`/queue/read_ahead_kb`**, which is in KiB, the **`blockdev`** command expresses everything in *sectors*. Historically, a sector was consistently 512 bytes, but are now commonly larger. To view the sector size for a block device, use **`blockdev --getss`** ***`DEVICE`***.

References

`tuned-main.conf`(5), **`tuned-adm`**(1), and **`tuned`**(8), **`tuned.conf`**(8), **`blockdev`**(8), and **`tuned-profiles`**(7) man pages.

The Tuned Project

https://tuned-project.org/

Latest Tuned 2.0 profiles

https://github.com/redhat-performance/tuned/tree/master/profiles

For more information, refer to the *Getting Started with Tuned* chapter in the *Red Hat Enterprise Linux 8 Monitoring and managing system status and performance* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index

Guided Exercise

Selecting a Tuned Profile

In this exercise, you will list tuned profile choices and select various tuned profiles for the running system.

Outcome(s)

You should be able to:

- Configure a system to use a tuned profile.
- Verify the settings after switching to a tuned profile.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab kerneltuning-profile start** command. The command determines if the host, **servera**, is reachable on the network.

```
[student@workstation ~]$ lab kerneltuning-profile start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Verify that the *tuned* package is installed, enabled, and started.

   2.1. Use **yum** to confirm that the *tuned* package is installed.

```
[root@servera ~]# yum list tuned
...output omitted...
Installed Packages
tuned.noarch                    2.10.0-15.el8                    @anaconda
```

   2.2. Verify that the **tuned** service is enabled and started.

```
[root@servera ~]# systemctl status tuned
tuned.service - Dynamic System Tuning Daemon
 Loaded: loaded (/usr/lib/systemd/system/tuned.service; enabled; vendor preset:
 enabled)
 Active: active (running) since Tue 2019-08-20 03:47:13 EDT; 6min ago
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3. List the available tuned profiles and identify the active profile.

```
[root@servera ~]# tuned-adm list
Available profiles:
- balanced               - General non-specialized tuned profile
- desktop                - Optimize for the desktop use-case
- latency-performance    - Optimize for deterministic performance at the cost of
                           increased power consumption
- network-latency        - Optimize for deterministic performance at the cost of
                           increased power consumption, focused on low latency
                           network performance
- network-throughput     - Optimize for streaming network throughput, generally
                           only necessary on older CPUs or 40G+ networks
- powersave              - Optimize for low power consumption
- throughput-performance - Broadly applicable tuned that provides excellent
                           performance across a variety of common server workloads
- virtual-guest          - Optimize for running inside a virtual guest
- virtual-host           - Optimize for running KVM guests
Current active profile: virtual-guest
```

The default profile on **`servera`** is **`virtual-guest`** because of our classroom environment.

4. Set the current active tuned profile to **`throughput-performance`**, then confirm the results.

4.1. Set the current active tuned profile to **`throughput-performance`**.

```
[root@servera ~]# tuned-adm profile throughput-performance
```

4.2. Verify that **`throughput-performance`** is the active tuned profile.

```
[root@servera ~]# tuned-adm active
Current active profile: throughput-performance
```

5. View the profile settings for the **`throughput-performance`** profile in the **`/usr/lib/tuned/throughput-performance/tuned.conf`** file.

```
[root@servera ~]# grep "^[^#]" /usr/lib/tuned/throughput-performance/tuned.conf
[main]
summary=Broadly applicable tuned that provides excellent performance across a
 variety of common server workloads
[cpu]
governor=performance
energy_perf_bias=performance
min_perf_pct=100
[disk]
readahead=>4096
[sysctl]
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
```

```
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10
vm.swappiness=10
```

6. Verify that the **readahead** tunable value for the **/dev/vda** disk is greater then or equal to **4096**.

```
[root@servera ~]# blockdev --getra /dev/vda
8192
```

7. Verify the values of the **vm.dirty_ratio**, **vm.dirty_background_ratio**, and **vm.swappiness** tunables set by the **throughput-performance** profile.

```
[root@servera ~]# sysctl -n vm.dirty_ratio vm.dirty_background_ratio \
vm.swappiness
40
10
10
```

8. Switch the tuned profile to the recommended profile for **servera**, and then confirm the results.

   8.1. View the most suitable tuned profile for **servera**.

```
[root@servera ~]# tuned-adm recommend
virtual-guest
```

   8.2. Set the current active tuned profile to **virtual-guest**.

```
[root@servera ~]# tuned-adm profile virtual-guest
```

   8.3. Verify that **virtual-guest** is the active tuned profile.

```
[root@servera ~]# tuned-adm active
Current active profile: virtual-guest
```

   8.4. Verify the values of the **vm.dirty_ratio** and **vm.swappiness** tunables set by the **virtual-guest** profile. Compare these values with the one set by the **throughput-performance** profile.

```
[root@servera ~]# sysctl -n vm.dirty_ratio vm.swappiness
30
30
```

   8.5. Exit **servera** to return to **workstation**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Finish

On **`workstation`**, run the **`lab kerneltuning-profile finish`** command to finish this exercise. The command ensures that **`servera`** is set to the **`virtual-guest`** tuned profile.

```
[student@workstation ~]$ lab kerneltuning-profile finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Customizing Tuned Profiles

Objectives

After completing this section, you should be able to create customized tuned profiles for unique workloads.

Creating Custom Tuned Profiles

The *tuned* package provides tuning profiles suitable for most common workloads. The built-in profiles in Red Hat Enterprise Linux have the potential to be changed from regular updates or errata. Creating custom tuned profiles protects from such updates. Moreover, there are cases where more fine-tuning is required to achieve the desired performance. The **`tuned`** service supports the creation and use of custom tuned profiles. The *tuned* package ships with built-in profiles, residing in the **`/usr/lib/tuned`** directory:

```
[user@host ~]$ ls /usr/lib/tuned
balanced   latency-performance  oracle       throughput-performance
desktop    network-latency      powersave    virtual-guest
functions  network-throughput   recommend.d  virtual-host
```

The **`/etc/tuned`** directory holds the **`active_profile`** file, which contains the name of the currently active tuned profile and the **`tuned-main.conf`** tuned configuration file. The directory also stores the custom profile configurations among other files.

```
[user@host ~]$ ls /etc/tuned
active_profile  bootcmdline  profile_mode  recommend.d  tuned-main.conf
```

In Linux, the **`/usr/lib`** directory contains libraries and default configuration files distributed with an application. On updating the software package, the application files can be replaced; they are intended to be left unmodified for system-related changes and must not be modified. As the built-in tuned profiles reside in the **`/usr/lib/tuned`** directory, the custom profile must not reside in the same directory to avoid unintended changes during software updates.

Create the **`/etc/tuned/http-performance`** directory to create custom profile called **`http-performance`**.

```
[user@host ~]$ sudo mkdir /etc/tuned/http-performance
```

> **Important**
> It is recommended to use a name for the custom profile that is different than the profiles shipped with the *tuned* package. If a custom profile has the same name as a built-in profile, the custom profile takes precedence.

In the newly created directory for the custom profile, tuned expects the **`tuned.conf`** configuration file. The **`tuned.conf`** configuration file for the custom profile may contain multiple sections. All sections are optional.

Inheriting Settings from an Existing Profile

Inheriting an existing profile into custom profile allows handling the additional tuning requirements needed for an environment. One of the built-in profiles is usually a good starting point for most custom profiles. The **`[main]`** section allows for the inheritance of settings from an existing tuning profile. To inherit settings, the **`tuned.conf`** configuration file needs to include the optional section **`[main]`**, followed by **`include=`*`parent profilename`***.

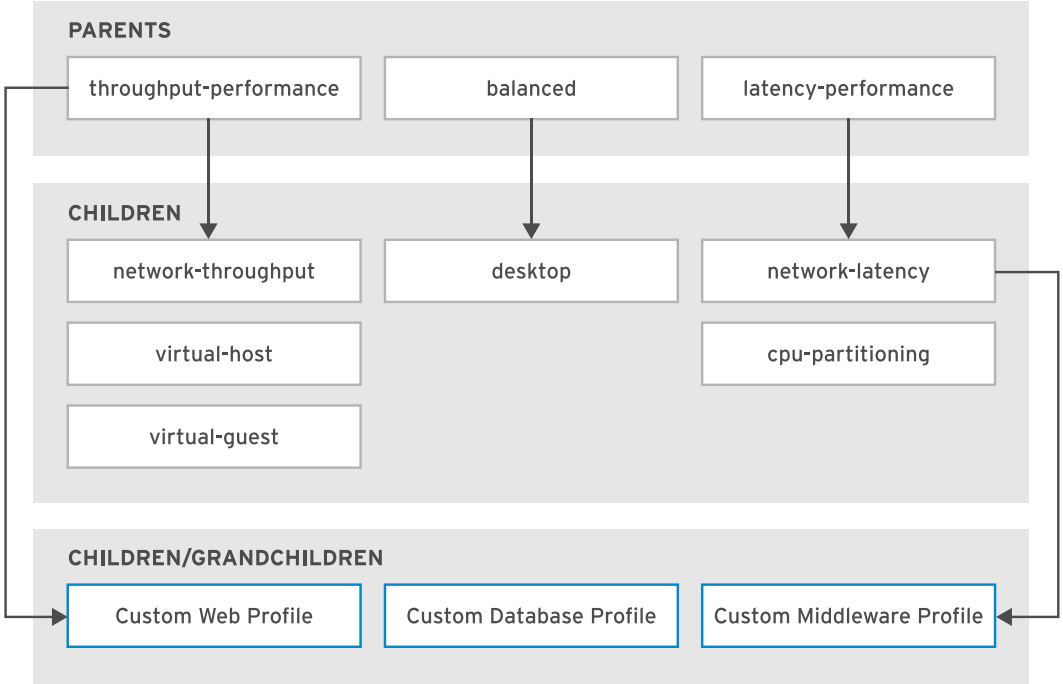


Figure 4.6: Inheritance in tuned profile

To inherit the settings of the **`network-throughput`** profile in the **`http-performance`** custom profile, the **`/etc/tuned/http-performance/tuned.conf`** configuration file must contain:

```
[main]
include=network-throughput
```

Using Plug-ins in Custom Profiles

A tuned profile uses plug-ins to monitor or tune different subsystems on the system. There are two types of plug-ins: monitoring plug-ins and tuning plug-ins.

The enabled tuning plug-ins instantiate the monitoring plug-ins whenever the tuning plug-ins require metrics from the monitoring plug-ins for dynamic tuning. The currently implemented monitoring plug-ins are: **`disk`** that provides metrics for the number of disk I/O per device, **`net`** that provides metrics for the number of transferred packets, and **`load`** that provides metrics for CPU load.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Each tuning plug-in tunes an individual subsystem and defines various attributes. An administrator sets the value of these attributes in the **`tuned.conf`** configuration file for the tuned profile. These tuning attributes are either set for all the devices, or for a particular device attached in a subsystem. The plug-ins most commonly used for server tuning are **`cpu`**, **`disk`**, **`mount`**, **`net`**, **`vm`**, **`sysctl`**, and **`script`**.

In the **`tuned.conf`** file, a section defines which tuning plug-in to instantiate and its related attributes:

```
[NAME]              ❶
type=TYPE           ❷
devices=DEVICES     ❸
```

❶ Define the section name of the plug-in instance. The section name must be unique, but it can be any desired string as tuned logging uses the section name.

❷ The attribute specifies the type of tuning plug-in. The **`type`** attribute is optional when the section name is same as the name of tuning plug-in, but is otherwise mandatory.

❸ The attribute specifies the list of devices the plug-in instance handles. The **`devices`** attribute can contain a comma-separated list of devices, the wildcard (**`*`**), and can exclude a particular device with negation (**`!`**) put in front of the device name. In case, the **`devices`** attribute is not present, **`devices=*`** is assumed, and the tuning applies to all affected devices in the target system.

The following example disables transparent huge pages with the **`vm`** plug-in:

```
[vm]
transparent_hugepages=never
```

The **`readahead`** tunable of the **`disk`** tuning plug-in causes the kernel to read ahead a specified amount of 512-byte chunks. To set the **`readahead`** tunable of the **`disk`** tuning plug-in on all hard drives where the device name matches **`sd*`** (for example, **`sda`** and **`sdb`**), add the following:

```
[all_sd_disks]
type=disk
devices=sd*
readahead=4096
```

When the tuning parameter should not be applied to the **`/dev/sda`** and **`/dev/sdc`** devices, the **`devices`** attribute excludes those devices using negation (**`!`**).

```
[not_all_sd_disks]
type=disk
devices=!sda, !sdc
readahead=4096
```

Executing a Script in Custom Profiles

The **`script`** plug-in executes an external script or executable when the profile is loaded or unloaded. In the **`[script]`** section, the **`script`** attribute specifies the script to execute. The shell script resides in the **`/etc/tuned/`***`profilename`* directory. The shell script library **`/usr/lib/tuned/functions`** may be used to import commonly used functions.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The tuning script must recognize the **`start`** and **`stop`** parameters. The **`start`** parameter enables the tuning settings when the profile is activated. The **`stop`** parameter reverts all previously applied tuning settings when the profile is deactivated. It is recommended to look at the existing scripts provided in the *tuned* package that are used by built-in tuning profiles, **`/usr/lib/tuned/powersave/script.sh`** and **`/usr/lib/tuned/sap/script.sh`**.

The following example shows an empty script template that does not tune anything, but provides a good starting point for a custom script:

```
#!/bin/sh
. /usr/lib/tuned/functions

start () {
}

stop () {
}

process $@
```

To add the **`/etc/tuned/http-performance/script.sh`** script to the **`http-performance`** profile, add the following section to **`/etc/tuned/http-performance/tuned.conf`**:

```
[script]
script=${i:PROFILE_DIR}/script.sh
```

In the above example, **`${i:PROFILE_DIR}`**, uses the **`PROFILE_DIR`** built-in function which returns the location of the profile and the **`tuned.conf`** file.

For more information on built-in functions in tuned profiles, refer to the *Variables and built-in functions in Tuned profiles* section in *Customizing Tuned profiles* chapter in the *Red Hat Enterprise Linux 8 Monitoring and managing system status and performance* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index.

Setting Sysctl Parameters in Custom Profiles

The custom tuning profile can set sysctl parameters. The **`sysctl`** tuning plug-in is used to set values for kernel tunables managed in **`/proc/sys`**. Create a section named **`[sysctl]`** in the profile configuration file. In the **`[sysctl]`** section, set the kernel tunables in the same way as those defined for a custom profile in the **`/etc/sysctl.conf`** or **`/etc/sysctl.conf.d/`** files respectively.

To set the **`net.ipv4.icmp_echo_ignore_all`** to **`1`** in the **`http-performance`** custom profile, the **`/etc/tuned/myprofile/tuned.conf`** configuration file must include the following:

```
[sysctl]
net.ipv4.icmp_echo_ignore_all=1
```

Verifying Custom Tuned Profiles

After activating a custom tuned profile, use the **`tuned-adm verify`** command to verify that the currently active profile set the tunables as expected. In case of errors, the **`/var/log/tuned/tuned.log`** log file records tuning activity and messages related to tuning events.

```
[user@host ~]$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

References

`tuned`(8), **`tuned.conf`**(5), **`tuned-main.conf`**(5), **`tuned-adm`**(1), and **`tuna`**(8) man pages

For more information, refer to the *Customizing Tuned profiles* chapter in the *Red Hat Enterprise Linux 8 Monitoring and managing system status and performance* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Customizing Tuned Profiles

In this exercise, you will create a custom tuned profile from an existing profile.

Outcome

You should be able to create a new tuned profile that inherits the settings of an existing profile.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab kerneltuning-customize start** command. The command determines if the host, **servera**, is reachable on the network.

```
[student@workstation ~]$ lab kerneltuning-customize start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Create a new custom tuned profile named **throughput-performance-alternative** in the **/etc/tuned/throughput-performance-alternative** directory. Inherit settings from the **throughput-performance** profile.

   2.1. Create the directory **/etc/tuned/throughput-performance-alternative** for the new custom profile.

```
[root@servera ~]# mkdir /etc/tuned/throughput-performance-alternative
```

   2.2. Inherit the settings for the new profile from the existing profile **throughput-performance**. Add the following two statements to the newly created file, **/etc/tuned/throughput-performance-alternative/tuned.conf**, with your favorite text editor.

```
[main]
include=throughput-performance
```

   2.3. Force the kernel to use swap space less frequently. Override the **vm.swappiness** setting inherited from the **throughput-performance** profile and set it to **5**. Add the following statements to the **/etc/tuned/throughput-performance-alternative/tuned.conf** file using the **sysctl** plug-in.

```
[main]
include=throughput-performance
[sysctl]
  vm.swappiness=5
```

Save and exit the **/etc/tuned/throughput-performance-alternative/tuned.conf** file.

3. Enable the newly created tuned profile **throughput-performance-alternative**. When done, verify that the custom settings for the profile are enabled.

   3.1. Enable the **throughput-performance-alternative** profile.

```
[root@servera ~]# tuned-adm profile throughput-performance-alternative
```

   3.2. Verify that the custom value for **vm.swappiness** is active on **servera**.

```
[root@servera ~]# sysctl -n vm.swappiness
5
```

   3.3. Exit **servera** to return to **workstation**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab kerneltuning-customize finish** command to finish this exercise. The command activates the default tuned profile, **virtual-guest** on **servera**.

```
[student@workstation ~]$ lab kerneltuning-customize finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Configuring Kernel Tunables and Tuned Profiles

Performance Checklist

In this lab, you will configure persistent kernel settings using both tuned profiles and methods for dynamic loadable modules.

Outcomes

You should be able to:

- Create a new custom tuned profile.
- Set the custom tuned profile as the active tuned profile.
- Load a kernel module with module parameters.

Before You Begin

Log in to **`workstation`** as **`student`** using **`student`** as the password.

On **`workstation`**, run the **`lab kerneltuning-review start`** command. The command determines if the host, **`serverb`**, is reachable on the network.

```
[student@workstation ~]$ lab kerneltuning-review start
```

1. On **`serverb`**, create a new tuned profile named **`virt-guest-lab`**. The profile must inherit the settings from the **`virtual-guest`** profile, and add a summary as **`Custom tuned profile`**.
2. The **`virt-guest-lab`** profile, must set the **`vm.swappiness`** kernel tunable to **`10`**, and disable the transparent huge pages. Also, the profile must set the **`readahead`** tunable on the **`/dev/vdb`** disk to **`4096`** sectors.
3. Configure **`serverb`** to use the **`virt-guest-lab`** tuned profile.
4. Load the **`usb_storage`** kernel module and set the **`delay_use`** parameter to **`0`**, and then logout from **`serverb`**.

Evaluation

On **`workstation`**, run the **`lab kerneltuning-review grade`** command to grade this lab. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab kerneltuning-review grade
```

Finish

On **`workstation`**, run the **`lab kerneltuning-review finish`** command to finish this exercise. The command ensures that **`serverb`** is set to the **`virtual-guest`** tuned profile, and unloads the **`usb_storage`** kernel module.

```
[student@workstation ~]$ lab kerneltuning-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Configuring Kernel Tunables and Tuned Profiles

Performance Checklist

In this lab, you will configure persistent kernel settings using both tuned profiles and methods for dynamic loadable modules.

Outcomes

You should be able to:

- Create a new custom tuned profile.
- Set the custom tuned profile as the active tuned profile.
- Load a kernel module with module parameters.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab kerneltuning-review start** command. The command determines if the host, **serverb**, is reachable on the network.

```
[student@workstation ~]$ lab kerneltuning-review start
```

1. On **serverb**, create a new tuned profile named **virt-guest-lab**. The profile must inherit the settings from the **virtual-guest** profile, and add a summary as **Custom tuned profile**.

   1.1. On **workstation**, use **ssh** to log in to **serverb** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

   1.2. Create the **/etc/tuned/virt-guest-lab** directory for the new custom profile.

```
[root@serverb ~]# mkdir /etc/tuned/virt-guest-lab
```

   1.3. Inherit the settings for the new profile from the predefined profile, **virtual-guest**, and add a summary as **Custom tuned profile**. In the **/etc/tuned/virt-guest-lab/tuned.conf** configuration file, create a new section called **[main]**, and add the following:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# vim /etc/tuned/virt-guest-lab/tuned.conf
[main]
summary=Custom tuned profile
include=virtual-guest
```

Do not exit the file.

2. The **virt-guest-lab** profile, must set the **vm.swappiness** kernel tunable to **10**, and disable the transparent huge pages. Also, the profile must set the **readahead** tunable on the **/dev/vdb** disk to **4096** sectors.

   2.1. Use the **sysctl** plug-in to set the **vm.swappiness** kernel tunables to **10**. In the **/etc/tuned/virt-guest-lab/tuned.conf** configuration file, create a new section called **[sysctl]**, and add the following:

```
[sysctl]
vm.swappiness=10
```

Do not exit the file.

   2.2. Use the **vm** plug-in and set the **transparent_hugepages** tunable to **never**. In the **/etc/tuned/virt-guest-lab/tuned.conf** configuration file, create a new section called **[vm]**, and add the following:

```
[vm]
transparent_hugepages=never
```

Do not exit the file.

   2.3. Use the **disk** plug-in and set the **readahead** tunable to **4096** sectors for the **/dev/vdb** disk. In the **/etc/tuned/virt-guest-lab/tuned.conf** configuration file, create a new section called **[disk]**, and add the following:

```
[disk]
devices=vdb
readahead=4096 sectors
```

Save and exit the file.

3. Configure **serverb** to use the **virt-guest-lab** tuned profile.

```
[root@serverb ~]# tuned-adm profile virt-guest-lab
```

4. Load the **usb_storage** kernel module and set the **delay_use** parameter to **0**, and then logout from **serverb**.

   4.1. Verify that the **usb_storage** kernel module is not loaded.

```
[root@serverb ~]# lsmod | grep usb_storage
```

   4.2. Create the **/etc/modprobe.d/usb_storage.conf** file with the content, **options usb_storage delay_use=0**; this sets a zero second delay before using the USB device.

```
[root@serverb ~]# vim /etc/modprobe.d/usb_storage.conf
options usb_storage delay_use=0
```

4.3. Load the **usb_storage** kernel module on **serverb**.

```
[root@serverb ~]# modprobe usb_storage
```

4.4. Verify the value for the **delay_use** parameter.

```
[root@serverb ~]# cat /sys/module/usb_storage/parameters/delay_use
0
```

4.5. Exit **serverb** to return to **workstation**.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab kerneltuning-review grade** command to grade this lab. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab kerneltuning-review grade
```

Finish

On **workstation**, run the **lab kerneltuning-review finish** command to finish this exercise. The command ensures that **serverb** is set to the **virtual-guest** tuned profile, and unloads the **usb_storage** kernel module.

```
[student@workstation ~]$ lab kerneltuning-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- Kernel tunables customize the behavior of Red Hat Enterprise Linux at boot, or on a running system. The **`sysctl`** command is used to list, read, and set kernel tunables.
- Installation of the *tuned* package also presets the profile, which should be the best available for the system.
- Tuned also monitors the use of system components, and tunes system settings dynamically, based on that monitoring information.
- A custom tuned profile can be based on other profiles, but include only certain aspects of the parent profile.
- The monitoring and tuning plug-ins, which are part of a tuned profile, enable monitoring and optimizing different devices on the system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 5

Managing Resource Limits with Control Groups

Goal

Manage resource contention and set limits for resource use on services, applications, and users using cgroup configuration.

Objectives

- Configure resource settings for individual and groups of services, applications, and users to control system resource sharing.
- Customize service, application, and user resource limits using cgroup plug-in parameters.

Sections

- Managing Resource Limits (and Guided Exercise)
- Customizing Control Groups (and Guided Exercise)

Lab

Managing Resource Limits with Control Groups

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Managing Resource Limits

Objectives

After completing this section, you should be able to configure resource settings for individual and groups of services, applications, and users to control system resource sharing.

Limiting System Resources with ulimit

The historical method for limiting system resources use is the **`ulimit`** command. Limits set are system-wide and enforced at login and session initiation by the Pluggable Authentication Module (PAM) authentication mechanism. The **`ulimit`** command is a BASH builtin that limits the resources available to the shell and its child processes This method is adequate for limiting runaway resource use, but is insufficient for handling resource contention.

Limits can be queried (and set) using the **`ulimit`** command-line tool. For a complete list of limits, refer to the **`ulimit`**(1) man page (part of the **`bash`**(1) manual page).

Configuring Persistent ulimit Rules

Using **`ulimit`** interactively restricts limits to a single user shell. The **`pam_limits`** PAM module sets POSIX resource limits on login. In the default **`/etc/pam.d/system-auth`** file, there is a **`session`** call to run **`pam_limits`**. This PAM module parses and applies **`ulimit`** settings from **`/etc/security/limits.conf`** and **`/etc/security/limits.d/*.conf`** during user log in. File syntax is described in the **`limits.conf`**(5) man page. The **`PAM`** module can also set limits other than the regular POSIX **`ulimit`** list.

For example, to limit users in the **`managers`** group to a maximum of three simultaneous logins, **`/etc/security/limits.d/managers.conf`** would include this setting:

```
@managers   hard  maxlogins   3
```

The previous example shows the configuration for a hard limit. You can also configure soft limits, which should not exceed hard limits. Those soft limits are also enforced, and a user cannot exceed them. The difference between soft and hard limits is that non-root users can configure soft limits, but just a user with **`root`** permissions can modify hard limits.

> **Note**
> As noted in the documentation, the **`rss`** limit is currently not enforceable on a Red Hat Enterprise Linux system.

Setting Limits for Services

POSIX limits are also set on **`systemd`** services, using the **`Limit*=`** entries in the unit file's **`[Service]`** block. For example, to limit the **`example.service`** to a maximum of **`30`** seconds of CPU time before being killed, the following file would be created to define the limit:

```
[root@host ~]# cat /etc/systemd/system/example.service.d/10-cpulimits.conf
[Service]
LimitCPU=30
```

Use the **systemctl daemon-reload** command to inform **systemd** of the changes. The service must also be restarted to use the configuration change.

The previous example uses a **systemd** *drop-in* file. Drop-in files are used to add or override individual settings, without modifying the original unit configuration found in **/usr/lib/systemd**. Never modify files in **/usr/lib/systemd**, because package upgrades will overwrite your custom configuration. Instead, create a subdirectory in **/etc/systemd/system/** using the original unit name with **.d** appended. Drop-in configuration files in that subdirectory directory must end in ***.conf**. Files are parsed in alphanumeric order, with files in the **/etc/systemd/system/*service.d*** directory given higher priority than the **/etc/systemd/system/*service*** unit file or the **/usr/lib/systemd/system/*service*** defaults.

The full list of **Limit*=** settings, and their meanings, are described in the **systemd.exec**(5) and **setrlimit**(2) manual pages.

Describing Control Groups

Control groups (cgroups) are a Linux kernel mechanism for fine-grained control of resources. With cgroups, resources are placed in *controllers* representing the type of resource; for example, **cpu** for CPU time, **memory** for memory usage, and **blkio** for disk I/O.

Controllers can be subdivided, as represented by a tree structure, with different weights or limits associated to branches and leaves, which are the cgroups. Each cgroup has zero or more processes associated with it. Resources are shared equally in a cgroup by default, but different limits and weights can be set on child cgroups as long as they do not exceed the parent cgroup limits. New cgroups inherit the limits set on its parent cgroup, unless explicitly overridden.

Cgroups and Systemd

Managing cgroups with **systemd** daemon is a simple way to handle complex resource configurations. By default, **systemd** subdivides the **cpu**, **cpuacct**, **blkio**, and **memory** cgroups into three equal *slices*: **system** for system services and daemons, **machine** for virtual machines and containers, and **user** for user sessions.

> **Warning**
> Previous versions of Red Hat Enterprise Linux used the deprecated **cgconfig** and **cgred** services found in the *libcgroup-tools* package. The default resource controllers now use **systemd** as already described. To avoid conflicts, only use **libcgroup** tools to manage controllers not currently supported by **systemd**, such as **net_prio**. See the **systemd-system.conf**(5) man page for information before using **libcgroup** tools.

The Three Default systemd Slices

Three parent slices are created by default. Administrators can create additional ***.slice** units, with their own limits, and assign services to these slices. To create slices as children of another slice, name them to include the parent name, a dash, and the child name, such as ***<parent>-<child>*.slice**. The new slice will be created inside the ***<parent>*.slice** slice.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

system.slice
All services started by **`systemd`** will be put into a new child group of this slice by default.

user.slice
A new child slice is created in this slice for every user that logs into the system. Each *session* for that user will be put into a new child slice of that slice.

machine.slice
Virtual machines managed by libvirt are automatically assigned a new child slice of this slice. The **`machine.slice`** slice is not created until the first virtual machine is launched. Linux containers are also placed in the **`machine.slice`** slice.

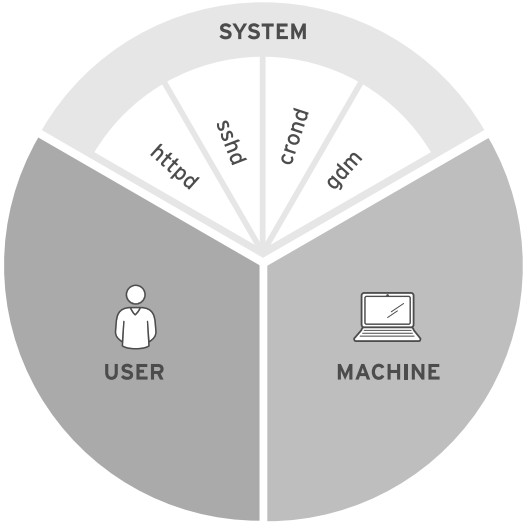


Figure 5.1: Example of systemd slices

Inspecting Cgroups

`Systemd` includes two tools to inspect the current cgroup/slice layout: **`systemd-cgtop`** and **`systemd-cgls`**. The **`systemd-cgtop`** command shows a top-like overview of the different cgroups, ordered by name, number of tasks, CPU usage, memory usage, or I/O activity. Interactive help is obtained by pressing the **?** key. The **`systemd-cgls`** command outputs a complete list of all slices, cgroups, and the processes they contain. By providing a cgroup tree path on the command line, the output is limited to the specified cgroup or slice.

> **Note**
> Unless CPU, memory, or I/O accounting is turned on for a slice or cgroup, only the number of tasks in a cgroup is reported. Turning on accounting features for a service might turn it on for other services in the same slice.

An interactive command can also be run inside a systemd slice by starting it with the **`systemd-run`** command using the **`--slice=`** option. Commands executed with **`systemd-run`** are children of the **`system.slice`** slice. The following example uses a slice called **`example.slice`**, which will be created if it does not already exist.

```
[root@host ~]# systemd-run --slice=example.slice sleep 10d
Running as unit run-4321.service
[root@host ~]# systemd-cgls /example.slice/run-4321.service
/example.slice/run-4321.service:
└─4322 /bin/sleep 10d
```

Starting ad hoc commands with **`systemd-run`** allows an administrator to control long-running commands with **`systemd`** tools like **`systemctl`**, while still limiting their resource usage.

Applying Resources Limits in Runtime

Using the **`systemctl set-property`** command sets the specified unit properties at runtime. The list of supported properties is located at **`systemd.resource-control`**(5) man page. Changes are applied immediately and stored persistently.

This example applies and verifies a resource control property to the memory subsystem for a running service:

```
[root@host ~]# systemctl set-property example.service MemoryAccounting=yes
[root@host ~]# systemctl set-property example.service MemoryLimit=2048M
[root@host ~]# cat /sys/fs/cgroup/memory/system.slice/example.service/
memory.limit_in_bytes
2147483648
```

Note

Adding the **`--runtime`** option in the **`systemctl set-property`** command applies the changes immediately, but the settings will not be retained after a service reload.

References

`ulimit`(1), **`limits.conf`**(5), **`setrlimit`**(2), **`systemctl`**(1), **`systemd.resource-control`**(5), and **`systemd.exec`**(5) man pages

Understanding the new control groups API
https://lwn.net/Articles/679786/

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Managing Resource Limits

In this exercise, you will configure resource settings for a variety of **systemd** units and users.

Outcome(s)

You should be able to:

- Configure POSIX resource limits for a specific user, group, and service.
- Explore and configure basic cgroup resource limits.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab cgroups-limits start** command. This command determines if the host, **servera**, is reachable on the network, and configures the **example** service.

```
[student@workstation ~]$ lab cgroups-limits start
```

You have been asked to limit the resources available for the following:

- The **operator1** user must be limited to a maximum of **16** processes. This should be a hard limit.
- Users in the **sysadmins** group must be limited to **128 MiB** of virtual memory. This should be a hard limit. (**128 MiB** equals **131072 KiB**)
- The **example** service must be limited to a maximum of **32** open files. This should be a hard limit.
- Configure the **example** service with a cgroup **MemoryLimit** of **1G**, using the command line and the default slice.

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Set the hard limit to **16** processes for the user **operator1**. The setting must be persistent across reboots.

   2.1. Create a new file called **/etc/security/limits.d/90-operator1.conf**, insert the following content, and save the changes:

```
[root@servera ~]# vim /etc/security/limits.d/90-operator1.conf
operator1 hard nproc 16
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3. Set a limit of **128** MiB of virtual memory for users in the **sysadmins** group.

3.1. Create a new file called **/etc/security/limits.d/10-sysadmins.conf**, insert the following content, and save the changes:

```
[root@servera ~]# vim /etc/security/limits.d/10-sysadmins.conf
@sysadmins hard as 131072
```

4. Set a hard limit of **32** open files for the **example** service. The setting must be persistent across reboots.

4.1. Create a drop-in directory for the **example** service:

```
[root@servera ~]# mkdir /etc/systemd/system/example.service.d
```

4.2. Create a new file called **/etc/systemd/system/example.service.d/10-NOFILE.conf**, insert the following content, and save the changes:

```
[root@servera ~]# vim /etc/systemd/system/example.service.d/10-NOFILE.conf
[Service]
LimitNOFILE=32
```

4.3. Reload the configuration changes, and then restart the **example** service.

```
[root@servera ~]# systemctl daemon-reload
[root@servera ~]# systemctl restart example
```

5. To configure the **example** service, set the **MemoryLimit** property of cgroups to **1G** for the default slice. Log out from **servera**.

5.1. Set the **MemoryAccounting** property to **true** for the **example** service:

```
[root@servera ~]# systemctl set-property example.service MemoryAccounting=true
```

5.2. Set the **MemoryLimit** property to **1G** for the **example** service:

```
[root@servera ~]# systemctl set-property example.service MemoryLimit=1G
```

5.3. Inspect the new files and directory created automatically after the execution of the previous commands, under the **/etc/systemd/system.control/example.service.d/** directory:

```
[root@servera ~]# cd /etc/systemd/system.control/example.service.d/
[root@servera example.service.d]$ ls -l
Total 8
-rw-r--r--. 1 root root 152 Jul 15 21:59 50-MemoryAccounting.conf
-rw-r--r--. 1 root root 154 Jul 15 22:10 50-MemoryLimit.conf
[root@servera example.service.d]# cat 50-MemoryAccounting.conf
# This is a drop-in unit file extension, created via "systemctl set-property"
# or an equivalent operation. Do not edit.
[Service]
```

```
MemoryAccounting=yes
[root@servera example.service.d]# cat 50-MemoryLimit.conf
# This is a drop-in unit file extension, created via "systemctl set-property"
# or an equivalent operation. Do not edit.
[Service]
MemoryLimit=1073741824
```

Note that the value for the **MemoryLimit** property was automatically converted from **1G** into bytes (1 GiB = 2^{30} bytes = 1073741824 bytes).

5.4. Return to your home directory.

```
[root@servera example.service.d]# cd ~
```

5.5. Use the **systemctl** command to query the properties.

```
[root@servera ~]# systemctl show -p MemoryAccounting example.service
MemoryAccounting=yes
[root@servera ~]# systemctl show -p MemoryLimit example.service
MemoryLimit=1073741824
```

5.6. Logout from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab cgroups-limits finish** command to complete this exercise.

```
[student@workstation ~]$ lab cgroups-limits finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Customizing Control Groups

Objectives

After completing this section, you should be able to customize service, application, and user resource limits using cgroup plug-in parameters.

Managing systemd cgroup Settings

Since the introduction of **systemd** in RHEL 7, the resource management settings have been moved from the process level to the application level by binding the system of cgroup hierarchies with the **systemd** unit tree. **systemd** provides the **systemctl** command and the service unit files to simplify and standardize system resource management control.

Control **systemd** slices and cgroups these three ways:

- Enable CPU, memory, and/or block I/O accounting for a service or a slice.
- Place resource limits on an individual service or slice.
- Run services in a custom slice.

Enabling Accounting

Enabling accounting in the resource controllers allows observing the accurate amount of resource usage, either by using the **systemd-cgtop** command or by viewing the cgroup virtual file system.

CPU, memory, and block I/O accounting can be enabled for a service by adding the **CPUAccounting=true**, **MemoryAccounting=true**, and **BlockIOAccounting=true** stanzas to the **[Service]** block of the desired service, or **[Slice]** block of the desired slice. Accounting does incur a slight administrative overhead on the kernel, but on regular systems, this overhead should not be noticeable.

A simple method to add these settings to a unit is to use a *drop-in* file. Drop-in files can be configured by creating a directory under **/etc/systemd/system/** that is named after the unit to be configured with **.d** appended. For example, the drop-in directory for **sshd.service** would be **/etc/systemd/system/sshd.service.d/**.

Create ***.conf** files in this directory to override or add specific options for that unit. Files are read in alphanumeric order. For example, this file and content is used to enable memory accounting for the **sshd.service** service.

```
[root@host ~]# cat /etc/systemd/system/sshd.service.d/20-accounting.conf
[Service]
MemoryAccounting=true
```

Enforcing Limits

The mechanism used to control resource usage is called enforcing limits. Limits are created in the same way as accounting is enabled, by placing the relevant **systemd** directive inside a unit drop-in file. View the full directive list in the **systemd.resource-control**(5) man page

Typical resource limitations look like this:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

CPUShares

Using cgroups CPU time is not limited directly. Instead, weights are set for individual cgroups. Inside a cgroup, CPU is shared equally, but between cgroups the number of shares determines the maximum allotment when CPU is scarce.

For example, if three cgroups are all requesting CPU time, and their **`CPUShares`** are set to **`1024`**, **`1024`**, and **`512`**, the first two cgroups will get roughly **`40%`** of the available CPU time each, and the third one will only get roughly **`20%`**. The default amount of **`CPUshares`** is **`1024`**.

Note: **`CPUShares`**, like all cgroup limits, are nested. On a system under load, if the **`system.slice`** can only get roughly **`33%`** of the available CPU time, then all cgroups (system services) running in that slice will share that **`33%`**.

CPUQuota

Assign the specified CPU time quota to the processes executed. The percentage measure is used to specify how much CPU time the unit could get at maximum load, relative to the total CPU time available on one CPU. Use percentage values greater than **`100%`** for allotting CPU time on more than one CPU.

For example, **`CPUQuota=25%`** ensures that the executed processes will never get more than **`25%`** CPU time on one CPU.

MemoryLimit

Used to set the *hard* limit of kernel memory used by a process or task in this unit. This setting can be used in cgroups, slices or services. Values are specified in bytes, but kibibytes, mebibytes, gibibytes, and tebibytes are specified by appending a **`K`**. **`M`**, **`G`**, or **`T`**.

When a cgroup hits the hard limit an *out-of-memory* (OOM) action will be triggered.

BlockIO*

Block I/O can be limited using relative weights, relative weights per device, or by specifying maximum read and write bandwidth per device.

> **Note**
> Setting the value of the **`CPUQuota`** or **`CPUShares`** parameters automatically enables **`CPUAccounting`**.

`systemd` cgroups settings can be used to limit access to specific devices. For the full list of possible settings and syntax, view the **`systemd.resource-control`**(5) man page.

> **Note**
> Enabling a limit for a specific resource will automatically enable accounting on that resource for that service.

Running Services in a Custom Slice

By default, **`systemd`** creates a new cgroup under the **`system.slice`** slice for each service, named after the unit file for the service. To run a service in a different slice, use the setting **`Slice=other.slice`** inside the **`[Service]`** block, where **`other.slice`** is the name of a custom slice. If the specified slice does not exist, it will be created when the service is started. The new slice will get default settings for all limits and accounting.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

System resource granularity and distribution equality are the main advantages of using custom slices. To gain more control over custom slices, new **`*.slice`** files can be created in **`/etc/systemd/system/`**. An example slice file can look like this:

```
[Unit]
Description=Example Custom Slice
[Slice]
CPUShares=2048
MemoryAccounting=true
```

To create a slice as a child of another slice, inheriting all parent settings unless explicitly overridden, the slice must be named ***`parent-child`*****`.slice`**. For example, to create a slice named **`flops`** as a child of the **`system.slice`**, the new unit file must be named **`system-flops.slice`**. When creating multi-level nested slices, it is not necessary to create each level manually. Creating a slice like **`system-a-b-c-d.slice`** will automatically create all needed levels: **`system-a.slice`**, **`system-a-b.slice`**, **`system-a-b-c.slice`**, and **`system-a-b-c-d.slice`**.

Important
Slices are not instantiated until at least one process/service has been assigned to it.

Note
After creating or modifying a new unit or drop-in file, run **`systemctl daemon-reload`** to instruct **`systemd`** to reread all unit definitions. Service changes typically require a restart of the affected service or slice also.

Previewing cgroups v2

RHEL8 includes cgroups v2 as a technology preview. The Linux community began cgroups v2 to improve the work accomplished by cgroups v1, and to introduce a more straightforward hierarchy for resource management.

Hierarchy in Control Groups

The hierarchy is a critical design point in each control group implementation. It is a main reason why Red Hat recommends switching from **`libcgroups`** to **`systemd`**. With the new cgroups v2, hierarchy is a main feature to simplify resource management and eliminate confusion due to the duplication of resource entries across multiple cgroup controllers.

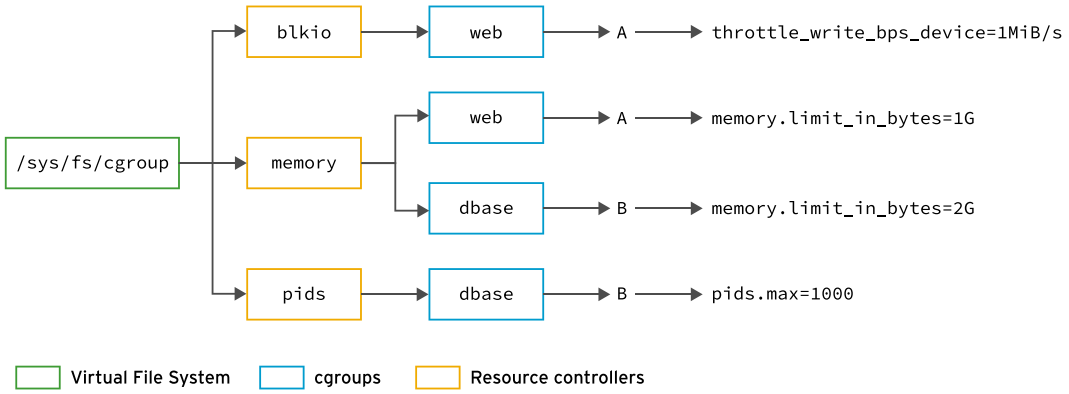


Figure 5.2: cgroups v1 Hierarchy

Hierarchy in cgroups v1 has cgroups associated with resource controllers. cgroups v1 mounts resource controllers, and cgroups are created within the controller hierarchies. In the graphic above, the **`web`** cgroup has a duplicated entry in the hierarchy, in **`memory`** and the **`blkio`** resource controllers. The same happens to the **`dbase`** cgroup that has duplicated entries, but in the **`memory`** and the **`pids`** resource controllers.

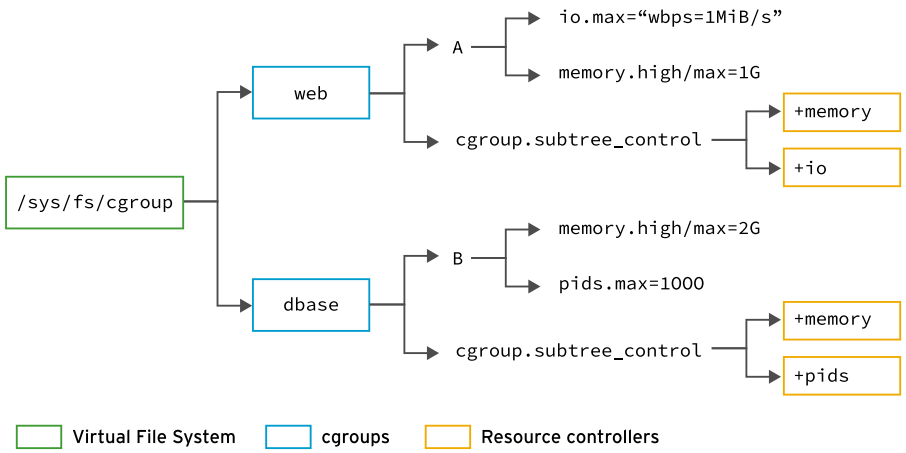


Figure 5.3: cgroups v2 Hierarchy

Hierarchy in cgroups v2 has controllers associated with cgroups. cgroups v2 creates the cgroups, then incorporates the desired controllers to the cgroups via the **`cgroup.subtree_control`** file. In the graphic above, the **`web`** and **`dbase`** cgroups exists once and they have their associated resource controllers listed inside **`cgroup.subtree_control`** file.

cgroups v2 Considerations

The following are the main cgroups v2 considerations:

- cgroups v2 provides a single control group hierarchy for all resource controllers.
- Mounting both cgroups v1 and cgroups v2 simultaneously is unsupported and not recommended.
- Enabling cgroups v2 requires adding **`systemd.unified_cgroup_hierarchy=1`** to the kernel boot parameters.
- cgroups v2 currently includes only the CPU, IO, Memory, RDMA, and PID controllers.
- cgroup v1 continues to be the default in RHEL 8 for backward compatibility.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

- The v1 **`blkio`** controller is replaced by the v2 **`io`** controller.
- **`CPUShares`**, **`MemoryLimit`**, and **`BlockIOWeight`** are deprecated in v2, and replaced by **`CPUWeight`**, **`MemoryMax`**, and **`IOWeight`**.

References

`systemd.unit`(5), **`systemd.exec`**(5), **`systemd.service`**(5), and **`systemd.resource-control`** manual pages.

Migrating from CGroups v1 in Red Hat Enterprise Linux 7 and below to CGroups v2 in Red Hat Enterprise Linux 8
https://access.redhat.com/articles/3735611

CFS Bandwidth Control
https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Customizing Control Groups

In this exercise, you will customize **systemd** unit and user resource sharing using cgroup plug-ins.

Outcome(s)

After completing this exercise, the **frontend** service must be placed in a separate slice, limiting the CPU time it can consume.

Before You Begin

Log in to **workstation** as **student**, using **student** as the password.

On **workstation**, run the **lab cgroups-customize start** command. The command determines if the host, **servera**, is reachable on the network, and also creates and configures necessary services required in the exercise.

```
[student@workstation ~]$ lab cgroups-customize start
```

Your **servera** has a new service called **frontend**. This service consumes large amounts of CPU time. To control the CPU available to the **frontend** service, place this service in its own slice called **frontend**. Make the new slice a child of the **system** slice.

Limit the new slice to **50%** of CPU time quota in 1 CPU core, and implement explicit accounting for CPU usage. Leave the CPU time quota that the **frontend** service can consume at default values.

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Create a configuration for the new slice and configure it with the previously stated values.

   2.1. Create a new file called **/etc/systemd/system/system-frontend.slice**, insert the following content, and save the changes:

```
[root@servera ~]# vim /etc/systemd/system/system-frontend.slice
[Unit]
Description=frontend Slice

[Slice]
CPUQuota=50%
CPUAccounting=yes
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3. Create a drop-in directory for the **frontend** service, with a drop-in file directing the service into the new slice.

3.1. Create the drop-in directory:

```
[root@servera ~]# mkdir /etc/systemd/system/frontend.service.d
```

3.2. Create a new file called **/etc/systemd/system/frontend.service.d/10-slice.conf**, insert the following content, and then save the changes:

```
[root@servera ~]# vim /etc/systemd/system/frontend.service.d/10-slice.conf
[Service]
Slice=system-frontend.slice
```

4. Reload and restart the affected services.

4.1. Reload all **systemd** unit files from disk.

```
[root@servera ~]# systemctl daemon-reload
```

4.2. Restart the **frontend** service.

```
[root@servera ~]# systemctl restart frontend
```

5. Verify that the **frontend** service is running in the new slice.

5.1. Inspect the output of the **systemctl status frontend** command.

```
[root@servera ~]# systemctl status frontend | grep CGroup
   CGroup: /system.slice/system-frontend.slice/frontend.service
```

5.2. Use **systemd-cgls** command to inspect the **/system.slice/system-frontend.slice** cgroup.

```
[root@servera ~]# systemd-cgls /system.slice/system-frontend.slice
Control group /system.slice/system-frontend.slice:
└─frontend.service
  └─2003 /usr/local/bin/frontend /dev/zero
```

6. Verify the settings in cgroups virtual file system, and then inspect the effect of the changes in the system. Log out from **servera**.

6.1. List the settings for the **system-frontend.slice** child slice in the cgroup virtual file system located at **/sys/fs/cgroup/cpu/system.slice/system-frontend.slice/**.

```
[root@servera ~]# cat \
/sys/fs/cgroup/cpu/system.slice/system-frontend.slice/cpu.cfs_quota_us
50000
```

```
[root@servera ~]# cat \
/sys/fs/cgroup/cpu/system.slice/system-frontend.slice/cpu.cfs_period_us
100000
```

The **cpu.cfs_quota_us** attribute has a value of **50000** microseconds of CPU time, corresponding to the **50%** of the length of a full period of CPU time, expressed in the **cpu.cfs_period_us** attribute with a value of **100000** microseconds.

6.2. Use **systemd-cgtop** command to verify the changes in the running control group.

```
[root@servera ~]# systemd-cgtop
Control Group                           Tasks   %CPU   Memory  Input/s Output/s
/                                         131   55.3   409.3M        -        -
/system.slice                              57   49.9   253.9M        -        -
/system.slice/system-frontend.slice         1   49.8   440.0K        -        -
/user.slice                                 8    1.2    22.7M        -        -
...output omitted...
```

Press **q** to exit the command.

6.3. Use the **top -n1** command to verify the changes in the running process.

```
[root@servera ~]# top -n1
top - 10:19:26 up 10:35,  1 user,  load average: 0.80, 0.55, 0.66
Tasks: 103 total,   2 running, 101 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.8 us,  3.2 sy,  0.0 ni, 67.7 id,  0.0 wa,  3.2 hi,  0.0 si,  0.0 st
MiB Mem :   1829.0 total,   1412.1 free,    181.9 used,    235.0 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   1487.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2003 root      20   0  228360   1744   1464 R  49.9   0.1   7:44.08 frontend
    1 root      20   0  180752  13880   9180 S   0.0   0.7   0:12.54 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.06 kthreadd
...output omitted...
```

6.4. Logout from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab cgroups-customize finish** command to complete this exercise.

```
[student@workstation ~]$ lab cgroups-customize finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Managing Resource Limits with Control Groups

Performance Checklist

In this lab, you will configure services, applications, and users to control system resource sharing.

Outcome(s)

After completing this exercise, your **serverb** should have a new custom slice, with two services assigned to that slice, each with its own custom limits.

Before You Begin

Log in to **workstation** as **student**, using **student** as the password.

On **workstation**, run the **lab cgroups-review start** command. This command determines if the host, **serverb**, is reachable on the network, and also creates and configures necessary services required in the lab.

```
[student@workstation ~]$ lab cgroups-review start
```

You have recently installed two new services on **serverb**: **web.service** and **dbase.service**. To control their resource usage, you will put these two services in a new slice named **workers**. This new slice should have the relative amount of CPU shares set to **2048**, and CPU accounting enabled.

To further control resource usage, the **web** service should be limited to a maximum of **64** open files, while the **dbase** service should be limited to a maximum of **32** processes.

The **web** service should have access to twice as much CPU as the **dbase** service when under load.

Do not modify neither the **/etc/systemd/system/web.service** file nor the **/etc/systemd/system/dbase.service** file on **serverb**.

1. Create the new **workers.slice** slice.
2. Configure the service limits for the **web** service. Configure the service to use a maximum of **64** open files, and **2048** CPU shares.
3. Configure the service limits for the **dbase** service. Configure the service to use a maximum of **32** processes, and **1024** CPU shares.
4. Activate your changes, and logout from **serverb**.

Evaluation

On **workstation**, run the **lab cgroups-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab cgroups-review grade
```

Finish

On **`workstation`**, run the **`lab cgroups-review finish`** command to complete this exercise.

```
[student@workstation ~]$ lab cgroups-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Managing Resource Limits with Control Groups

Performance Checklist

In this lab, you will configure services, applications, and users to control system resource sharing.

Outcome(s)

After completing this exercise, your **serverb** should have a new custom slice, with two services assigned to that slice, each with its own custom limits.

Before You Begin

Log in to **workstation** as **student**, using **student** as the password.

On **workstation**, run the **lab cgroups-review start** command. This command determines if the host, **serverb**, is reachable on the network, and also creates and configures necessary services required in the lab.

```
[student@workstation ~]$ lab cgroups-review start
```

You have recently installed two new services on **serverb**: **web.service** and **dbase.service**. To control their resource usage, you will put these two services in a new slice named **workers**. This new slice should have the relative amount of CPU shares set to **2048**, and CPU accounting enabled.

To further control resource usage, the **web** service should be limited to a maximum of **64** open files, while the **dbase** service should be limited to a maximum of **32** processes.

The **web** service should have access to twice as much CPU as the **dbase** service when under load.

Do not modify neither the **/etc/systemd/system/web.service** file nor the **/etc/systemd/system/dbase.service** file on **serverb**.

1. Create the new **workers.slice** slice.

   1.1. On **workstation**, use **ssh** to log in to **serverb** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

   1.2. Create a new file called **/etc/systemd/system/workers.slice**, insert the following content, and then save the changes:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# vim /etc/systemd/system/workers.slice
[Unit]
Description=Workers Slice

[Slice]
CPUShares=2048
CPUAccounting=yes
```

2. Configure the service limits for the **web** service. Configure the service to use a maximum of **64** open files, and **2048** CPU shares.

   2.1. Create the **/etc/systemd/system/web.service.d** directory to hold drop-in files for the **web** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/web.service.d
```

   2.2. Create a drop-in file called **/etc/systemd/system/web.service.d/10-limits.conf**, insert the following content, and then save the changes:

```
[root@serverb ~]# vim /etc/systemd/system/web.service.d/10-limits.conf
[Service]
Slice=workers.slice
CPUShares=2048
LimitNOFILE=64
```

3. Configure the service limits for the **dbase** service. Configure the service to use a maximum of **32** processes, and **1024** CPU shares.

   3.1. Create the **/etc/systemd/system/dbase.service.d** directory to hold drop-in files for the **dbase** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/dbase.service.d
```

   3.2. Create a drop-in file called **/etc/systemd/system/dbase.service.d/10-limits.conf**, insert the following content, and then save the changes:

```
[root@serverb ~]# vim /etc/systemd/system/dbase.service.d/10-limits.conf
[Service]
Slice=workers.slice
CPUShares=1024
LimitNPROC=32
```

4. Activate your changes, and logout from **serverb**.

   4.1. Reload all **systemd** unit files from disk.

```
[root@serverb ~]# systemctl daemon-reload
```

   4.2. Restart both the **web** and **dbase** services.

```
[root@serverb ~]# systemctl restart web dbase
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

4.3. Logout from **serverb**.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab cgroups-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab cgroups-review grade
```

Finish

On **workstation**, run the **lab cgroups-review finish** command to complete this exercise.

```
[student@workstation ~]$ lab cgroups-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- How to control system resources using POSIX limits with the help of **`systemd`**.
- How to activate cgroups in runtime from the command line.
- Why using cgroups integrated with **`systemd`** is the recommended way to use control groups.
- How to use customized cgroups slices for granular management of system resources.
- The significant changes in the technical preview of cgroups v2.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 6

Analyzing Performance Using System Tracing Tools

Goal

Diagnose system and application behaviors using a variety of resource-specific tracing tools.

Objectives

- Profile system events and observe performance counters with the **`perf`** command.
- Trace system and library calls for a process.
- Gather specific diagnostic data using the SystemTap infrastructure.
- Debug applications using tools instrumented by the in-kernel eBPF virtual machine.

Sections

- Profiling with Performance Counters (and Guided Exercise)
- Tracing System and Library Calls (and Guided Exercise)
- Gathering Diagnostic Data Using SystemTap (and Guided Exercise)
- Tracing System Events with eBPF Tools (and Guided Exercise)

Lab

Analyzing Performance Using Tracing Tools

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Profiling with Performance Counters

Objectives

After completing this section, you should be able to profile system events and observe performance counters with the **`perf`** command.

Introduction to Performance Counters

Performance Counters for Linux (PCL) is a kernel-based subsystem that provides a framework for collecting and analyzing performance data. PCL is included with Red Hat Enterprise Linux 8 along with *Performance Events for Linux (PERF)*. PERF is a powerful profiling infrastructure that provides a suite of userspace tools to analyze the collected performance data.

Performance data is a collection of measured hardware and software events. A performance event is an occurrence of a particular condition, such as when the processor completes a set of instructions, or when an application experiences a page fault. Computer processors use a set of performance counters to count performance events.

PERF defines a set of commonly used performance events and provides tools to list the event counts or record them in a report for later analysis.

Installing PERF

The *perf* package is included in the **`rhel-8.0-for-x86_64-baseos-rpms`** repository for registered Red Hat Enterprise Linux 8 deployments.

```
[root@host ~]# yum install perf
...output omitted...
```

Debug symbols are required for performance data that is collected on one system and analyzed on a different machine. Without the debug symbols, the data would only show address pointers (hexadecimal numbers) of the recorded functions. The debug symbols are provided in the *kernel-debuginfo* and *kernel-debuginfo-common* packages.

Administrators must subscribe their machine to the **`rhel-8-for-x86_64-baseos-debug-rpms`** channel to install the kernel debuginfo packages.

```
[root@host ~]# subscription-manager repos --enable \
rhel-8-for-x86_64-baseos-debug-rpms
```

It is important to match the version of the running kernel on the system where performance data is collected to the kernel versions of both kernel debuginfo packages. Identify your kernel version, then install the corresponding kernel debuginfo packages. For example:

```
[root@host ~]# uname -r
4.18.0-80.el8.x86_64

[root@host ~]# yum install kernel-debuginfo-4.18.0-80.el8.x86_64 \
kernel-debuginfo-common-x86_64-4.18.0-80.el8.x86_64
...output omitted...
```

Describing Frequently Used PERF Tools

The following command displays a complete list of perf commands.

```
[root@host ~]# perf help
```

The following command displays a usage statement and additional information for a specific command.

```
[root@host ~]# perf --help COMMAND
```

The following list describes frequently used **perf** commands:

perf list
: Lists the events available on a particular machine. These events vary based on performance monitoring hardware and software configuration of the system.

perf stat
: Provides overall statistics for common performance events, including instructions executed and clock cycles consumed. Options allow selection of events other than the default measurement events.

perf record
: Records performance data into a file for analysis at a later time.

perf report
: Reads the performance data from a file generated by the **perf record** command.

perf archive
: Creates an archive with object files found in the performance data file, and generated by the **perf record** command, so that the information can be analyzed on another machine.

Listing Event Types

The **perf** command measures events from multiple sources. For example, many software events are sourced from kernel counters, while typical hardware events are sourced from the processor's *Performance Monitoring Unit (PMU)*. The available events on a system vary with each processor type and model.

The following **perf list** command lists events that are supported on a physical system running the Linux kernel version **4.18.0-80.el8.x86_64**. This example displays only the hardware and software events from a long list of various event types supported on this system.

```
[root@host ~]# perf list

List of pre-defined events (to be used in -e):
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]
...output omitted...
```

Counting Events

The following **`perf stat`** command gathers performance counter statistics for a given command.

```
[root@host ~]# perf stat dd if=/dev/zero of=/dev/null bs=2048 count=10000000
10000000+0 records in
10000000+0 records out
20480000000 bytes (20 GB, 19 GiB) copied, 6.3269 s, 3.2 GB/s

 Performance counter stats for 'dd if=/dev/zero of=/dev/null bs=2048
 count=10000000':

       6327.018717      task-clock (msec)         #    1.000 CPUs utilized
                 8      context-switches          #    0.001 K/sec
                 1      cpu-migrations            #    0.000 K/sec
                78      page-faults               #    0.012 K/sec
    24,805,371,009      cycles                    #    3.921 GHz
    32,438,639,778      instructions              #    1.31  insn per cycle
     7,055,243,722      branches                  # 1115.098 M/sec
        70,121,397      branch-misses             #    0.99% of all branches

       6.327766934 seconds time elapsed

       1.759371000 seconds user
       4.562818000 seconds sys
```

Event Descriptions

`task-clock`
Reports a lock count specific to the task that is running.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

context-switches
Counts context switches. As of Linux **2.6.34**, these are reported as happening in the kernel.

cpu-migrations
Reports the number of times the process has migrated to a new CPU.

page-faults
Reports the number of page faults.

cycles
Total cycles; not affected by CPU frequency scaling.

instructions
Retired instructions (those that have completed).

branches
Retired branch instructions.

branch-misses
Mispredicted branch instructions.

For more information on event types, refer to the **perf_event_open**(2) man page.

Counting Specific Events

When used with the **perf stat** command, the following options report statistics for specific events, system wide, for 10 seconds. The **-e** option collects statistics for a comma-delimited list of events; the **-a** option collects system-wide from all CPUs; and, the **sleep 10** option stops the collection after 10 seconds.

```
[root@host ~]# perf stat \
-e cycles,instructions,cache-references,cache-misses,bus-cycles \
-a sleep 10
 Performance counter stats for 'system wide':

     1,545,881,409      cycles
       318,687,601      instructions              #    0.21  insn per cycle
        80,129,003      cache-references
        46,054,021      cache-misses              #   57.475 % of all cache refs
         9,475,367      bus-cycles

      10.002204738 seconds time elapsed
```

Recording Samples of Events for Reports

The **perf record** command records a number of samples over a period of time. The default frequency is to collect **4000** samples per second. These values are adjustable from the command line. The benefit of sampling is that when you collect enough individual samples, you will produce a statistically meaningful result and thus reduce measurement error.

Output from the **perf record** command is saved, by default, to a data file named **perf.data** in the current working directory.

The following **perf record** command records specific events for a given command.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# perf record \
-e cpu-clock,instructions,cache-misses,context-switches \
dd if=/dev/zero of=/dev/null bs=2048 count=10000000

10000000+0 records in
10000000+0 records out
20480000000 bytes (20 GB, 19 GiB) copied, 6.4065 s, 3.2 GB/s
[ perf record: Woken up 10 times to write data ]
[ perf record: Captured and wrote 2.460 MB perf.data (53337 samples) ]
```

The following **`perf report`** command reads the default **`perf.data`** file, and displays the results to the terminal. The **`--stdio`** option displays output to the terminal.

Notice the percentage of time that the execution of the **`dd`** command spent in a particular shared object, such as **`[kernel.kallsyms]`** or **`libc-2.28.so`**, during the **`cpu-clock`** sampling events.

```
[root@host ~]# perf report --stdio
...output omitted...
# Total Lost Samples: 0
#
# Samples: 25K of event 'cpu-clock'
# Event count (approx.): 6407000000
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ...............................
#
    25.41%  dd       [kernel.kallsyms]  [k] do_syscall_64
    12.84%  dd       [kernel.kallsyms]  [k] __clear_user
    11.75%  dd       libc-2.28.so       [.] __GI___libc_write
    11.55%  dd       libc-2.28.so       [.] read
...output omitted...

# Samples: 25K of event 'instructions'
# Event count (approx.): 32534668777
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................
  ..............................................
#
    33.09%  dd       [kernel.kallsyms]  [k] __clear_user
     6.80%  dd       [kernel.kallsyms]  [k] __audit_syscall_exit
     6.39%  dd       [kernel.kallsyms]  [k] selinux_file_permission
     6.26%  dd       [kernel.kallsyms]  [k] fsnotify
     5.47%  dd       [kernel.kallsyms]  [k] syscall_return_via_sysret
...output omitted...
```

The following **`perf record`** command records sampling events of **`context-switches`** from the entire system, for a period of 10 seconds. The report saves the results to a file named **`cs-syswide.data`** in the current working directory.

```
[root@host ~]# perf record -o cs-syswide.data -e context-switches -a sleep 10
```

The following **`perf report`** command displays the results to the terminal. The **`-i cs-syswide.data`** option reads the file located in the working directory created earlier by the **`perf record`** command.

Observe the overhead details, such as the percentage of time each CPU spent in context switching events.

```
[root@host ~]# perf report -i cs-syswide.data --stdio
...output omitted...
# Total Lost Samples: 0
#
# Samples: 1K of event 'context-switches'
# Event count (approx.): 12991
#
# Overhead  Command          Shared Object      Symbol
# ........  ...............  .................  ..................
#
    50.62%  swapper          [kernel.kallsyms]  [k] schedule_idle
    18.27%  qemu-kvm         [kernel.kallsyms]  [k] schedule
    11.98%  SPICE Worker     [kernel.kallsyms]  [k] schedule
     7.54%  CPU 0/KVM        [kernel.kallsyms]  [k] schedule
     3.54%  CPU 1/KVM        [kernel.kallsyms]  [k] schedule
     2.96%  pool             [kernel.kallsyms]  [k] schedule
     1.59%  httpd            [kernel.kallsyms]  [k] schedule
     0.65%  xfsaild/sda3     [kernel.kallsyms]  [k] schedule
     0.33%  kworker/5:0-eve  [kernel.kallsyms]  [k] schedule
...output omitted...
```

Creating a PERF Archive

The **`perf archive`** command reads the default **`perf.data`** file, located in the current working directory, and archives all relevant debug symbols for the latest recorded data to the **`perf.data.tar.gz`** archive file in the current working directory.

To analyze performance data collected on **`host`** on a separate system, such as **`remotehost`**, you must copy both the **`perf.data`** and **`perf.data.tar.gz`** files from **`host`** to the **`remotehost`** system.

For example, the following command records performance data on the **`host`** system.

```
[root@host ~]# perf record -e \
cpu-clock,instructions,cache-misses,context-switches -- sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.019 MB perf.data (19 samples) ]
```

The following **`perf archive`** command reads the **`perf.data`** file from the previous **`perf record`** command and creates the archive on the **`host`** system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# perf archive
Now please run:

$ tar xvf perf.data.tar.bz2 -C ~/.debug

wherever you need to run 'perf report' on.
```

Once you have copied both **`perf.data`** and **`perf.data.tar.bz2`** files to the **`remotehost`**, create the **`~/.debug`** directory in the **`root`** user's home directory on the **`remotehost`**. Extract the **`perf.data.tar.bz2`** archive file.

```
[root@remotehost ~]# mkdir ~/.debug
[root@remotehost ~]# tar xf perf.data.tar.bz2 -C ~/.debug
```

Open the report for analysis on the remote system.

```
[root@remotehost ~]# perf report --stdio
...output omitted...
# Total Lost Samples: 0
#
# Samples: 2  of event 'cpu-clock'
# Event count (approx.): 500000
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  .......................
#
    50.00%  sleep    [kernel.kallsyms]  [k] page_counter_cancel
    50.00%  sleep    libc-2.28.so       [.] _dl_addr


# Samples: 8  of event 'instructions'
# Event count (approx.): 1683087
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ..................................
#
    93.28%  sleep    ld-2.28.so         [.] get_common_indeces.constprop.3
     6.43%  perf     [kernel.kallsyms]  [k] perf_swevent_add
     0.29%  perf     [kernel.kallsyms]  [k] perf_event_exec
     0.00%  perf     [kernel.kallsyms]  [k] perf_event_addr_filters_exec
     0.00%  perf     [kernel.kallsyms]  [k] __intel_pmu_enable_all.constprop.19


# Samples: 8  of event 'cache-misses'
# Event count (approx.): 51482
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ......................................
#
    51.73%  sleep    [kernel.kallsyms]  [k] blocking_notifier_call_chain
    39.51%  sleep    ld-2.28.so         [.] dl_main
     7.94%  sleep    [kernel.kallsyms]  [k] get_random_u64
     0.59%  perf     [kernel.kallsyms]  [k] perf_event_exec
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
      0.21%  perf     [kernel.kallsyms]  [k] perf_event_addr_filters_exec
      0.02%  perf     [kernel.kallsyms]  [k] __intel_pmu_enable_all.constprop.19


# Samples: 1  of event 'context-switches'
# Event count (approx.): 1
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ............
#
   100.00%  sleep    [kernel.kallsyms]  [k] schedule
...output omitted...
```

References

`perf`(1), **`perf-list`**(1), **`perf-stat`**(1), **`perf-record`**(1), **`perf-report`**(1), and **`perf_event_open`**(1) man pages

For more information, refer to:

Tool Reference section in the *Red Hat Enterprise Linux 7 Performance Tuning Guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index#sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Tool_Reference-perf

Monitoring Performance section in the *Red Hat Enterprise Linux 7 Developer Guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/developer_guide/perf

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Profiling with Performance Counters

In this exercise, you will use the perf tool to observe various performance events.

Outcomes

Generate and analyze hardware and software events for profiling system performance.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab tracing-profile start** command. This command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab tracing-profile start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Install *perf* and kernel debuginfo packages for the version of the kernel you are running on **servera**.

   2.1. Identify the version of your currently running kernel.

```
[root@servera ~]# uname -r
4.18.0-80.el8.x86_64
```

   2.2. Install the packages necessary for profiling your system. The installation may take several minutes depending on your environment.

```
[root@servera ~]# yum install perf \
kernel-debuginfo-4.18.0-80.el8.x86_64 \
kernel-debuginfo-common-x86_64-4.18.0-80.el8.x86_64
...output omitted...
================================================================================
Package                          ...   Repository                ...          Size
================================================================================
Installing:
 perf                                  rhel-8.0-for-x86_64-baseos-rpms        2.1 M
 kernel-debuginfo                      rhel-8-for-x86_64-baseos-debug-rpms    370 M
Installing dependencies:
 kernel-debuginfo-common-x86_64        rhel-8-for-x86_64-baseos-debug-rpms     52 M
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
Transaction Summary
================================================================================
Install  20 Packages
...output omitted...
Is this ok [y/N]: y
Downloading Packages:
...output omitted...
Installed:
  perf-4.18.0-80.el8.x86_64
  kernel-debuginfo-4.18.0-80.el8.x86_64
  kernel-debuginfo-common-x86_64-4.18.0-80.el8.x86_64

Complete!
```

3. You will work with two separate **perf** data files, **program1.data** and **program2.data**. The programs that were used to generate these data files achieve the same goal, but deploy two different algorithms to perform their tasks. Analyze both data files to determine differences that would indicate which of the programs is running most efficiently. You should evaluate all events profiled in the data files.

   The following events are present in both data files:

   - branches
   - branch-misses
   - branch-loads
   - branch-load-misses
   - cache-misses
   - instructions
   - L1-dcache-loads
   - L1-dcache-load-misses
   - L1-dcache-stores
   - L1-icache-load-misses
   - LLC-loads
   - LLC-load-misses
   - LLC-stores
   - LLC-store-misses

   3.1. Download and extract the **program-data.tgz** file to the **root** user's home directory.

```
[root@servera ~]# wget \
http://materials.example.com/labs/tracing-profile/program-data.tgz
[root@servera ~]# tar -xf program-data.tgz
```

   3.2. On **workstation**, open a second terminal session and use **ssh** to log into **servera** as **student** user. Position both terminals side by side. Escalate user privileges to the **root** user's account in the second terminal.

   3.3. In the first terminal session, use the **perf** command to view the **program1.data** file as text output.

```
[root@servera ~]# perf report -i program1.data --stdio
/root/bin/program1 with build id 00de901cc91da6aee94f38b1a6df4dba6973ff9f not
 found, continuing without symbols
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 902  of event 'branches'
# Event count (approx.): 121106960
#
# Overhead  Command   Shared Object     Symbol
# ........  ........  ................  ........................
#
    99.67%  program1  program1          [.] main
     0.11%  program1  [kernel.vmlinux]  [k] __hrtimer_run_queues
     0.11%  program1  [kernel.vmlinux]  [k] cpu_load_update_active
     0.11%  program1  [kernel.vmlinux]  [k] _cond_resched
     0.00%  perf      [kernel.vmlinux]  [k] perf_event_exec
...output omitted...
```

3.4. In the second terminal session, use the **perf** command to view the **program2.data** file as text output.

```
[root@servera ~]# perf report -i program2.data --stdio
/root/bin/program2 with build id 05b970814d530c04dce9d4bea26075a82c800950 not
 found, continui>
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 5K of event 'branches'
# Event count (approx.): 127121275
#
# Overhead  Command   Shared Object     Symbol
# ........  ........  ................  ..................................
#
    98.82%  program2  program2          [.] main
     0.08%  program2  [kernel.vmlinux]  [k] read_tsc
     0.06%  program2  [kernel.vmlinux]  [k] cpu_load_update
     0.06%  program2  [kernel.vmlinux]  [k] __update_load_avg_cfs_rq.isra.39
     0.06%  program2  [kernel.vmlinux]  [k] __update_load_avg_se.isra.38
     0.06%  program2  [kernel.vmlinux]  [k] raise_softirq
     0.04%  program2  [kernel.vmlinux]  [k] tick_sched_timer
...output omitted...
```

3.5. Use the **Page Down** and **Page Up** keys to scroll through each program's output file. Compare the "Samples of event," "Event count," and top entries in the "Overhead" column (sorted in descended order) for values that indicate which of the programs is more efficient.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For example, observe the **`instructions`** events and the percentage of overhead generated within each of the shared objects.

```
# Samples: 1K of event 'instructions'
# Event count (approx.): 2404291991
#
# Overhead  Command   Shared Object     Symbol
# ........  ........  ................  ..............................
#
    99.81%  program1  program1          [.] main
     0.12%  program1  [kernel.vmlinux]  [k] handle_pte_fault
     0.06%  program1  [kernel.vmlinux]  [k] _raw_spin_lock
     0.01%  program1  [kernel.vmlinux]  [k] handle_mm_fault
     0.00%  program1  [kernel.vmlinux]  [k] rcu_start_this_gp.isra.20
     0.00%  program1  [kernel.vmlinux]  [k] perf_mux_hrtimer_handler
...output omitted...


# Samples: 8K of event 'instructions'
# Event count (approx.): 2455109339
#
# Overhead  Command   Shared Object     Symbol
# ........  ........  ................  ...................................
#
    99.62%  program2  program2          [.] main
     0.03%  program2  [kernel.vmlinux]  [k] clear_page_erms
     0.03%  program2  [kernel.vmlinux]  [k] get_page_from_freelist
     0.02%  program2  [kernel.vmlinux]  [k] __hrtimer_run_queues
     0.02%  program2  [kernel.vmlinux]  [k] native_sched_clock
     0.02%  program2  [kernel.vmlinux]  [k] _raw_spin_lock_irqsave
...output omitted...
```

3.6. After comparing the event listings, you will discover that **`program1`** is the more efficient running program of the two.

In each terminal, press **`q`** to exit the **`perf report`** command.

4. Analyze the **`dd if=/dev/zero of=/dev/null count=100000`** command. Use the **`perf`** command to display an initial reading of the **`page-faults`** event to measure the passage of time, and the number of **`page-faults`** and **`context-switches`** events, to determine if further investigation is necessary.

Output will vary between command runs.

4.1. Use the **`perf stat`** command to measure the following software events for the **`dd if=/dev/zero of=/dev/null count=100000`** command.

- cpu-clock
- page-faults
- context-switches

Running the **`stat`** command in this manner provides a quick look at information that may be a starting point to investigate performance issues. For example, the following output indicates that the number of **`page-faults`** and **`context-switches`** events may require additional investigation.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# perf stat -e cpu-clock,page-faults,context-switches \
dd if=/dev/zero of=/dev/null count=100000
100000+0 records in
100000+0 records out
51200000 bytes (51 MB, 49 MiB) copied, 0.112426 s, 455 MB/s

 Performance counter stats for 'dd if=/dev/zero of=/dev/null count=100000':

        112.729043      cpu-clock (msec)          #    0.997 CPUs utilized
                68      page-faults               #    0.603 K/sec
                 2      context-switches          #    0.018 K/sec

       0.113096801 seconds time elapsed

       0.040863000 seconds user
       0.071744000 seconds sys
```

4.2. Use the **`perf record`** command to collect profile data for the **`dd if=/dev/zero of=/dev/null count=100000`** command. Save the profile to the **`dd-report.data`** file for analysis. Include software events that provide details for the **`page-faults`** to determine if they are minor or major faults.

Measure the following software events:

- cpu-clock
- page-faults
- minor-faults
- major-faults
- context-switches

```
[root@servera ~]# perf record -o dd-report.data -e cpu-clock,page-faults,\
minor-faults,major-faults,context-switches \
dd if=/dev/zero of=/dev/null count=100000
100000+0 records in
100000+0 records out
51200000 bytes (51 MB, 49 MiB) copied, 0.113232 s, 452 MB/s
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.031 MB perf.data (467 samples) ]
```

4.3. Use the **`perf report`** command to analyze the profile in the **`dd-report.data`** file.

Scroll through **`dd-report.data`** file output, and observe the overhead percentages for the various shared objects, to determine areas where unusual activity may occur. In this particular output, there is no unusual activity during the **`dd`** command execution.

```
[root@servera ~]# perf report --stdio -i dd-report.data
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 473  of event 'cpu-clock'
# Event count (approx.): 118250000
```

```
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  ................  ...............................
#
    25.58%  dd       [kernel.vmlinux]  [k] do_syscall_64
     8.88%  dd       libc-2.28.so      [.] read
     8.25%  dd       [kernel.vmlinux]  [k] selinux_file_permission
     7.61%  dd       libc-2.28.so      [.] __GI___libc_write
     6.77%  dd       dd                [.] 0x0000000000004f12
     6.13%  dd       [kernel.vmlinux]  [k] __audit_syscall_entry
...output omitted...

# Samples: 6  of event 'page-faults'
# Event count (approx.): 175
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  ................  ............................
#
    80.57%  dd       libc-2.28.so      [.] _init
    15.43%  dd       ld-2.28.so        [.] _dl_sysdep_start
     2.29%  dd       ld-2.28.so        [.] _start
     1.14%  dd       [kernel.vmlinux]  [k] __clear_user
     0.57%  dd       [kernel.vmlinux]  [k] copy_user_generic_string

# Samples: 6  of event 'minor-faults'
# Event count (approx.): 238
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  ................  ............................
#
    78.57%  dd       libc-2.28.so      [.] _int_malloc
    18.07%  dd       ld-2.28.so        [.] dl_main
     2.10%  dd       ld-2.28.so        [.] _start
     0.84%  dd       [kernel.vmlinux]  [k] __clear_user
     0.42%  dd       [kernel.vmlinux]  [k] copy_user_generic_string

# Samples: 0  of event 'major-faults'
# Event count (approx.): 0
#
# Overhead  Command  Shared Object  Symbol
# ........  .......  .............  ......
#


# Samples: 7  of event 'context-switches'
# Event count (approx.): 7
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  ................  ............................
#
    57.14%  dd       [kernel.vmlinux]  [k] schedule
    42.86%  dd       [kernel.vmlinux]  [k] preempt_schedule_common
...output omitted...
```

Press **q** to exit the **`perf report`** command.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Compare software events between two programs, **`cache-test1`** and **`cache-test2`**, to identify the differences that qualify one of the programs as more efficient than the other. Collect profile data from each program and save it to a ***`program`***`.report` file for analysis at a later time.

Compare the following software events:

- cpu-clock
- task-clock
- page-faults
- context-switches

Output will vary between each report generation.

5.1. Create reports for each program. Notice the differences in the number of samples and amounts captured during report generation. You should notice for **`cache-test2`** a higher value on **`cpu-clock`** and **`task-clock`** events statistics. This program also reports a lower value on **`page-faults`** and **`context-switches`** events statistics.

```
[root@servera ~]# perf record -o cache-test1.report  \
-e cpu-clock,task-clock,page-faults,context-switches \
cache-test1
Starting
Finished
[ perf record: Woken up 2 times to write data ]
[ perf record: Captured and wrote 0.452 MB cache-test1.report (9670 samples) ]

[root@servera ~]# perf record -o cache-test2.report  \
-e cpu-clock,task-clock,page-faults,context-switches \
cache-test2
Starting
Finished
[ perf record: Woken up 11 times to write data ]
[ perf record: Captured and wrote 2.555 MB cache-test1.report (55613 samples) ]
```

5.2. View the event statistics in the **`cache-test1.report`** file.

```
[root@servera ~]# perf report --stdio -i cache-test1.report
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 4K of event 'cpu-clock'
# Event count (approx.): 1138750000
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ...............................
#
    95.39%  cache-test1  cache-test1       [.] main
     1.71%  cache-test1  [kernel.vmlinux]  [k] __do_page_fault
     0.64%  cache-test1  [kernel.vmlinux]  [k] clear_page_rep
...output omitted...

# Samples: 4K of event 'task-clock'
```

```
# Event count (approx.): 1138750000
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ....................................
#
    94.71%  cache-test1  cache-test1       [.] main
     1.69%  cache-test1  [kernel.vmlinux]  [k] __do_page_fault
     0.66%  cache-test1  [kernel.vmlinux]  [k] get_page_from_freelist
     0.61%  cache-test1  [kernel.vmlinux]  [k] clear_page_rep
...output omitted...

# Samples: 566  of event 'page-faults'
# Event count (approx.): 46873
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ...........................
#
    99.58%  cache-test1  cache-test1       [.] main
...output omitted...

# Samples: 11  of event 'context-switches'
# Event count (approx.): 11
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ............
#
   100.00%  cache-test1  [kernel.vmlinux]  [k] schedule
...output omitted...
```

Press **q** to exit the **perf report** command.

5.3. View the event statistics in the **cache-test2.report** file.

```
[root@servera ~]# perf report --stdio -i cache-test2.report
# Total Lost Samples: 0
#
# Samples: 26K of event 'cpu-clock'
# Event count (approx.): 6704000000
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................
  ..............................................
#
    98.99%  cache-test2  cache-test2       [.] main
     0.41%  cache-test2  [kernel.vmlinux]  [k] __do_page_fault
     0.08%  cache-test2  [kernel.vmlinux]  [k] clear_page_rep
...output omitted...

# Samples: 26K of event 'task-clock'

# Event count (approx.): 6703500000
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ....................................
#
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
    99.00%  cache-test2  cache-test2       [.] main
     0.34%  cache-test2  [kernel.vmlinux]  [k] __do_page_fault
     0.11%  cache-test2  [kernel.vmlinux]  [k] get_page_from_freelist
     0.10%  cache-test2  [kernel.vmlinux]  [k] clear_page_rep
...output omitted...

# Samples: 1K of event 'page-faults'
# Event count (approx.): 46813
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ............................
#
    98.37%  cache-test2  cache-test2       [.] main
     1.53%  cache-test2  libc-2.28.so      [.] sysmalloc
...output omitted...

# Samples: 116  of event 'context-switches'
# Event count (approx.): 116
#
# Overhead  Command      Shared Object     Symbol
# ........  ...........  ................  ...........................
#
    98.28%  cache-test2  [kernel.vmlinux]  [k] schedule
     1.72%  cache-test2  [kernel.vmlinux]  [k] preempt_schedule_common
...output omitted...
```

Press **q** to exit the **perf report** command.

5.4. Log out from both **servera** terminal sessions to return to **workstation** as the **student** user.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab tracing-profile finish** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab tracing-profile finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Tracing System and Library Calls

Objectives

After completing this section, you should be able to trace system and library calls for a process.

System Calls and Library Calls

To evaluate the performance of a program, you must measure both the tasks performed by the program, and the amount of time required to execute each task. Programs accomplish tasks with their program code. If a program calls a function that is provided by the kernel, then the execution time is spent inside the kernel. A function that is provided by a library causes the program to spend time outside the kernel. A kernel-provided function that is called from a program is a *system call*, whereas a function provided by a library is called *library call*.

System Calls

A system call processes a function provided by the Linux kernel. System calls are always specific to the operating system. When a system call is executed, the program time is spent in the kernel internal function that is called. The program execution time spent in the kernel is referred to as **`system time`**. For example, **`open`** is a typical system call that opens a file.

Library Calls

A library call is a call provided by a program library. Program libraries may be shared among programs to lower the redundancy of functionality required. Some libraries abstract operating system-specific system calls. This allows for the porting of software between different operating systems, as only the required library has to be adjusted to work with the target operating system. An example of a library that is widely used is **`glibc`**, the GNU C-Library. This library allows programmers to write and compile code in the **`C`** programming language on Linux. The program execution time spent on a library function is called **`user time`**.

Red Hat Enterprise Linux 8 includes the **`ltrace`** command to trace library calls. This command supports the tracing of binaries compiled with the position independent executables (PIE) option disabled. To check if a binary is compiled with this option, use the **`file`** command. The **`file`** command returns **`ELF 64-bit LSB executable`** as output for binaries compiled with the PIE option disabled.

```
[user@host ~]$ file grub2-editenv
/bin/grub2-editenv: ELF 64-bit LSB executable, x86-64, version 1 (SYSV)
...output omitted...
```

Tracing System Calls

The **`strace`** utility displays information about system calls made by a process. This command helps identify the kernel function on which a program spends its time.

The **`strace`** command may take another command name as its argument. It will run the provided command, displaying each system call made by the program. For example, to review the system calls executed when running **`uname`**, start the command with **`strace`**:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[user@host ~]$ strace uname
execve("/bin/uname", ["uname"], [/* 17 vars */]) = 0
brk(0)                                  = 0x87e000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
 0x7f972a615000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=67910, ...}) = 0
mmap(NULL, 67910, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f972a604000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
...output omitted...
```

Alternatively, an existing process can be traced by providing the process ID (PID), and using the **-p *PID*** option.

```
[user@demo ~]$ strace -p 3124
```

For performance tuning purposes, it is sometimes more helpful to display only counts, instead of detailed function call information. This summarizes the report and shows how often a syscall was executed, and how much time was spent on it during program execution. Use the **-c** switch to display only counts.

```
[user@host ~]$ strace -c uname
Linux
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 82.28    0.000130          43         3           open
 17.72    0.000028          14         2           munmap
  0.00    0.000000           0         1           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         5           close
  0.00    0.000000           0         4           fstat
  0.00    0.000000           0         9           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         4           brk
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           uname
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.000158                    37         1 total
```

When used with the **-f** option, **strace** follows and profiles child processes created by the **fork()** system call. By default, child processes are not followed. The following **strace** command displays system calls and follows process forks:

```
[user@host ~]$ strace -fc elinks -dump http://classroom.example.com/pub
```

At times, the output of the **strace** is very long. With the **-e *expression*** switch, the output of **strace** can be filtered to only show a specific system call. For example, to show only the count of the **open** system call with **uname**, execute the following:

```
[user@host ~]$ strace -e open -c uname
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.000052          17         3           open
------ ----------- ----------- --------- --------- ----------------
100.00    0.000052                     3           total
```

This command can be very useful for pinpointing some types of process performance bottlenecks. If a process is blocking because of lock or I/O contention, the system call where the problem is occurring will be the one displayed on the output terminal. This utility has a number of useful options, including specifying that only a particular type of system call be displayed, summarizing the number of calls made, and so on.

References

`strace`(1) man page.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Tracing System and Library Calls

In this exercise, you will analyze the system and library calls of a program.

Outcomes

Identify the slowest and most used system call of **updatedb**.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab tracing-call start** command. The command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab tracing-call start
```

Investigate **/usr/bin/updatedb** using the **strace** command to identify the slowest and most used system calls on **servera**.

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Trace the system calls of **updatedb** with **strace**. Identify the slowest system call and the most used system call.

   2.1. Trace the system calls of **updatedb** with **strace**.

```
[root@servera ~]# strace -c updatedb
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 62.09    0.558173         711       785           write
 32.09    0.288504          23     12651           lstat
  2.61    0.023490           2      9663           poll
  0.83    0.007489           3      2568           chdir
  0.71    0.006381          15       420           read
  0.69    0.006216           2      2567           fchdir
  0.43    0.003846           7       579           open
  0.37    0.003313           6       581           close
  0.08    0.000718         359         2           socket
  0.02    0.000219           5        44           brk
  0.02    0.000154           5        32           getdents
  0.01    0.000123           7        17           munmap
  0.01    0.000109           3        34           fcntl
```

```
  0.01    0.000088           3        26           mmap
  0.00    0.000044           1        36           fstat
  0.00    0.000039          39         1         1 access
...
------ ----------- ----------- --------- --------- ----------------
100.00    0.898914                30030         3 total
```

2.2. Identify the slowest system call by looking for the highest value in the **usecs/call** column. On our system, the slowest system call is **write**, with **711** usecs per call.

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 62.09    0.558173         711       785           write
...
```

2.3. Identify the most used system call by looking for the highest value in the **calls** column. On our system, the most often executed system call is **lstat**, with **12651** calls in the previous run.

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
...
 32.09    0.288504          23     12651           lstat
...
```

3. Logout from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab tracing-call finish** command to finish this exercise.

```
[student@workstation ~]$ lab tracing-call finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Gathering Diagnostic Data Using SystemTap

Objectives

After completing this section, you should be able to gather specific diagnostic data using the SystemTap infrastructure.

Introduction to SystemTap

SystemTap is a tracing and probing tool to dynamically analyze the activities of a running kernel. SystemTap provides an infrastructure to monitor system performance and pinpoint the underlying cause of a performance issue. Using the **`kprobes`** facility within the Linux kernel, SystemTap scripts specify where to attach probes and what data to collect when the probe executes.

The essential idea behind a SystemTap script is to name events and give them handlers. Whenever a specified event occurs, the kernel runs the handler as if it were a quick subroutine, and then resumes normal processing. There are several kinds of events, such as entering or exiting a function, a timer expiring, or the entire SystemTap session starting or stopping. A handler is a series of script language statements that specify the work to be done whenever the event occurs. This work normally includes extracting data from the event context, storing that data to internal variables, or printing results.

Normally, SystemTap scripts can only be run on systems where SystemTap is installed. This could mean installing the full SystemTap compiler, kernel headers, source code, and debuginfo kernel packages on every system that is to be analyzed. In many cases, it is not feasible or secure to provide compile tools on production systems, and these tools are commonly prohibited by corporate policy. Instead, SystemTap provides a capability known as *cross-instrumentation*.

Cross-instrumentation

Cross-instrumentation is the process of generating a SystemTap instrumentation kernel module from a SystemTap script on a host system, to be used on a target system. Typically, the host system is a non-production system configured with the necessary compiler tools for generating the instrumentation modules. The host system can be installed with multiple kernel versions and related kernel information packages, to match the kernel versions of all target systems where the instrumentation modules will be run.

Installing a SystemTap Host System

To prepare SystemTap, install the *systemtap* package along with the corresponding set of *kernel-devel*, *kernel-debuginfo* and *kernel-debuginfo-common-arch* packages that match the kernel. If you plan to use your SystemTap host system to compile instrumentation for multiple target kernels, install each of those kernel versions and their corresponding *devel* and *debuginfo* packages. All host kernel versions and package sets must be a matching architecture.

> **Note**
> Do not install the *kernel-debug* package, as this is not a kernel used by SystemTap, and it does not load the needed debug symbols. Instead, *kernel-debug* is a kernel with debugging code enabled for development diagnostic use.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

A SystemTap host system requires the kernel source code, debug symbols, a compiler, and other support packages. Typically, these packages are not installed on production systems. Production systems will commonly have the *BaseOS* and *Appstream* repositories already enabled. Additionally, enable the *BaseOS Debug* repository needed for the *kernel-debuginfo* packages, as shown here:

```
[root@host ~]# subscription-manager repos --enable \
rhel-8-for-x86_64-baseos-debug-rpms
```

Installing SystemTap Using stap-prep

Installing the *systemtap* package will also install compiler, header and debug package dependencies, but not the *debuginfo* packages. Use the **`stap-prep`** to locate and install any matching kernel headers and debuginfo that are still required.

```
[root@host ~]# yum install systemtap
...output omitted...
================================================================================
 Package                    Version          Repository
================================================================================
Installing:
 systemtap                  4.0-7.el8        rhel-8-for-x86_64-appstream-rpms
Installing dependencies:
 annobin                    8.64-1.el8       rhel-8-for-x86_64-appstream-rpms
 cpp                        8.2.1-3.5.el8    rhel-8-for-x86_64-appstream-rpms
 dyninst                    9.3.2-12.el8     rhel-8-for-x86_64-appstream-rpms
 gcc                        8.2.1-3.5.el8    rhel-8-for-x86_64-appstream-rpms
 isl                        0.16.1-6.el8     rhel-8-for-x86_64-appstream-rpms
 libdwarf                   20180129-4.el8   rhel-8-for-x86_64-appstream-rpms
 systemtap-client           4.0-7.el8        rhel-8-for-x86_64-appstream-rpms
 systemtap-devel            4.0-7.el8        rhel-8-for-x86_64-appstream-rpms
 systemtap-runtime          4.0-7.el8        rhel-8-for-x86_64-appstream-rpms
 glibc-devel                2.28-42.el8      rhel-8-for-x86_64-baseos-rpms
 glibc-headers              2.28-42.el8      rhel-8-for-x86_64-baseos-rpms
 kernel-headers             4.18.0-80.el8    rhel-8-for-x86_64-baseos-rpms
 libxcrypt-devel            4.1.1-4.el8      rhel-8-for-x86_64-baseos-rpms
 mokutil                    1:0.3.0-9.el8    rhel-8-for-x86_64-baseos-rpms
Installing weak dependencies:
 kernel-debug-devel         4.18.0-80.el8    rhel-8-for-x86_64-baseos-rpms
 kernel-devel               4.18.0-80.el8    rhel-8-for-x86_64-baseos-rpms
...output omitted...

Complete!
```

The required gcc compiler, library, and headers were installed, along with the kernel source code and headers. Verify that the kernel source code and header packages match the running kernel version, by using **`uname -r`** to query the running kernel's release.

```
[root@host ~]# uname -r
4.18.0-80.el8.x86_64
```

Use the **`stap-prep`** command to locate and install the remaining kernel packages. Verify that the kernel packages previously installed, and these remaining kernel packages, are the same kernel release.

```
[root@host ~]# stap-prep
...output omitted...
================================================================================
 Package                    Version          Repository
================================================================================
Installing:
 kernel-debuginfo           4.18.0-80.el8  rhel-8-for-x86_64-baseos-debug-rpms
Installing dependencies:
 kernel-debuginfo-common-x86_64
                            4.18.0-80.el8  rhel-8-for-x86_64-baseos-debug-rpms
...output omitted...
Complete!
```

Installing SystemTap and Kernel Packages Manually

You will manually install when adding additional kernel package sets to a SystemTap host system. You should verify that each kernel, header, devel, and debuginfo set are a matching kernel release. When SystemTap compiles implementation modules for releases that are not the same as the running kernel on the host, SystemTap will error and exit if it cannot locate the symbols, headers, and source code for that release. In the following example, the kernel release of a target system is checked to determine the kernel package set to install on the host system.

```
[root@target ~]# uname -r
4.17.0-2.el8.el8.x86_64

[root@host ~]# yum install kernel-debuginfo-4.17.0-2.el8.el8.x86_64 \
kernel-debuginfo-common-x86_64-4.17.0-2.el8.x86_64 \
kernel-devel-4.17.0-2.el8.x86_64 \
kernel-debug-devel-4.17.0-2.el8.x86_64 \
kernel-headers-4.17.0-2.el8.x86_64 \
systemtap
...output omitted...
Complete!
```

Using SystemTap

The SystemTap host **`stap`** command is the front-end to the SystemTap tool, used to interpret, compile and run SystemTap scripts on the host system. SystemTap caches a compiled object in the **`.systemtap`** directory, in the user's home directory, when the script is first compiled. If the source script is detected to be unmodified, **`stap`** will use the cached object on subsequent runs. The **`stap`** command processes SystemTap scripts in 5 passes:

1. **Parse** the SystemTap script to validate its syntax.
2. **Elaborate** the script to resolve symbols and include any probes and function calls.
3. **Translate** the script into C code and create a **`Makefile`**. Stopping at this pass results in the translator displaying the C code.
4. **Compile** the C code into a kernel module. Stopping at this pass creates the kernel object without running it.
5. **Run** the implementation module, using the **`staprun`** command to:
   - Load the kernel module.
   - Start execution of the program.
   - Unload the kernel module at program exit.

Running **`stap`** with the **`-v`** verbose option will display each pass as it is performed.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# stap -v /usr/share/systemtap/examples/process/syscalls_by_proc.stp
Pass 1: parsed user script and 474 library scripts...in 140usr/20sys/170real ms.
Pass 2: analyzed script: 4 probes, 2 functions...in 820usr/430sys/1260real ms.
Pass 3: using cached /root/.systemtap/cache/a5/stap_a5dd688fc36...f9_132343.c
Pass 4: using cached /root/.systemtap/cache/a5/stap_a5dd688fc36...f9_132343.ko
Pass 5: starting run.
Collecting data... Type Ctrl-C to exit and display results
^C
#SysCalls  Process Name
618        stapio
85         in:imjournal
45         sshd
44         irqbalance
34         tuned
23         systemd-journal
...output omitted...
Pass 5: run completed in 10usr/10sys/34366real ms.
```

Compiling Portable Kernel Modules

SystemTap generates universally unique identifiers to name the kernel modules created. When creating modules to distribute for target system use, modules can be given useful names instead. Because the implementation modules are previously compiled and read-only when distributed and placed correctly on target systems, unprivileged users are able to load and run these modules. As previously discussed, modules are compiled for a specific kernel release, and can only be run on target systems with that same kernel release to function correctly.

The following example uses the **stap** command to process the **/usr/share/systemtap/examples/process/syscalls_by_proc.stp** script and create the **syscalls_by_proc** module in the current directory.

```
[root@host ~]# stap -p 4 -v -m syscalls_by_proc \
/usr/share/systemtap/examples/process/syscalls_by_proc.stp
```

- The **-p 4** option processes through pass 4 (Compile).
- The **-v** option allows the user to observe the processing passes.
- The **-m syscalls_by_proc** option names the module **syscalls_by_proc.ko**, and generates it in the current directory.

Kernel module names may only contain the following characters:

- ASCII lowercase alphabetic characters (a-z)
- Digits (0-9)
- Underscores (_)

Using SystemTap as an Unprivileged User

The **staprun** command is the back-end tool for SystemTap, used to run kernel modules that were created by the front-end **stap** command and distributed to a target system. The **stap** and **staprun** commands require elevated systems privileges. Unprivileged users must be added to either the **stapusr** or the **stapdev** group to run SystemTap modules. The following list describes the associated group privileges:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

stapusr
: Group members are restricted to using only the **staprun** command. These users may not compile SystemTap modules.

  Members may run any modules placed into **/lib/modules/$(uname -r)/systemtap** by the system administrator. Members may also write and load low-privilege script modules, which are normally limited to manipulating their own processes.

stapdev
: Group members can write and load script modules with root-equivalent privileges, without particular security constraints. Members can use either the **stap** or the **staprun** commands to run SystemTap modules.

  If the user also has **stapusr** membership, they can use the **staprun** command to load a module from any directory.

For example, an administrator would compile and distribute a SystemTap instrumentation module named **/root/syscalls_by_proc.ko**, using the following procedure make the module available for **stapusr** group users:

```
[root@host ~]# mkdir /lib/modules/$(uname -r)/systemtap
[root@host ~]# ls -ld /lib/modules/$(uname -r)/systemtap
drwxr-xr-x. 2 root root 6 Jul 25 20:43 /lib/modules/4.18.0-80.el8.x86_64/systemtap
[root@host ~]# cp /root/syscalls_by_proc.ko /lib/modules/$(uname -r)/systemtap
```

An unprivileged user, who belongs to **stapusr** group, can use the **staprun syscalls_by_proc** command to run the instrumentation module.

```
[user@host ~]$ staprun syscalls_by_proc
Collecting data... Type Ctrl-C to exit and display results
...output omitted...
```

Alternatively, **staprun** can be provided with the full path to the target module. When invoked with a bare module name, it will look for the matching **.ko** file in the running kernel **/lib/modules/$(uname -r)/systemtap** directory.

A user with both **stapusr** and **stapdev** group membership can run the **stap** command with **root** privileges.

```
[user@host ~]$ stap -v \
 /usr/share/systemtap/examples/process/syscalls_by_proc.stp
```

Important

Red Hat recommends that privileged SystemTap access be restricted, since privileged access allows compiling and loading kernel modules into a running kernel.

Installing a SystemTap Target Runtime

SystemTap instrumentation modules are used by Red Hat support and customers to gather data to determine the cause of issues. Compiled modules are copied to target systems, where the SystemTap runtime environment has been installed.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Install the *systemtap-runtime* package to prepare the SystemTap runtime environment on a target. The *systemtap-runtime* package is in the RHEL 8 **AppStream** repository.:

```
[root@target ~]# yum install systemtap-runtime
...output omitted...
```

After copying a SystemTap instrumentation module to the correct kernel module release directory on a target system, use the **staprun** command to load and run the module.

Performance Analysis With the SystemTap Examples

SystemTap provides resources for becoming proficient using SystemTap. The *SystemTap Beginners Guide* and the *Tapset Reference* are located in the **/usr/share/docs/systemtap-client** directory.

The *systemtap-client* package installs many fully developed example scripts that administrators may find useful for gathering performance data on their systems. Locate the scripts in the **/usr/share/systemtap/examples** directory, which includes a web page to browse and find commentary about the examples.

```
[root@target ~]# firefox /usr/share/systemtap/examples/index.html
```

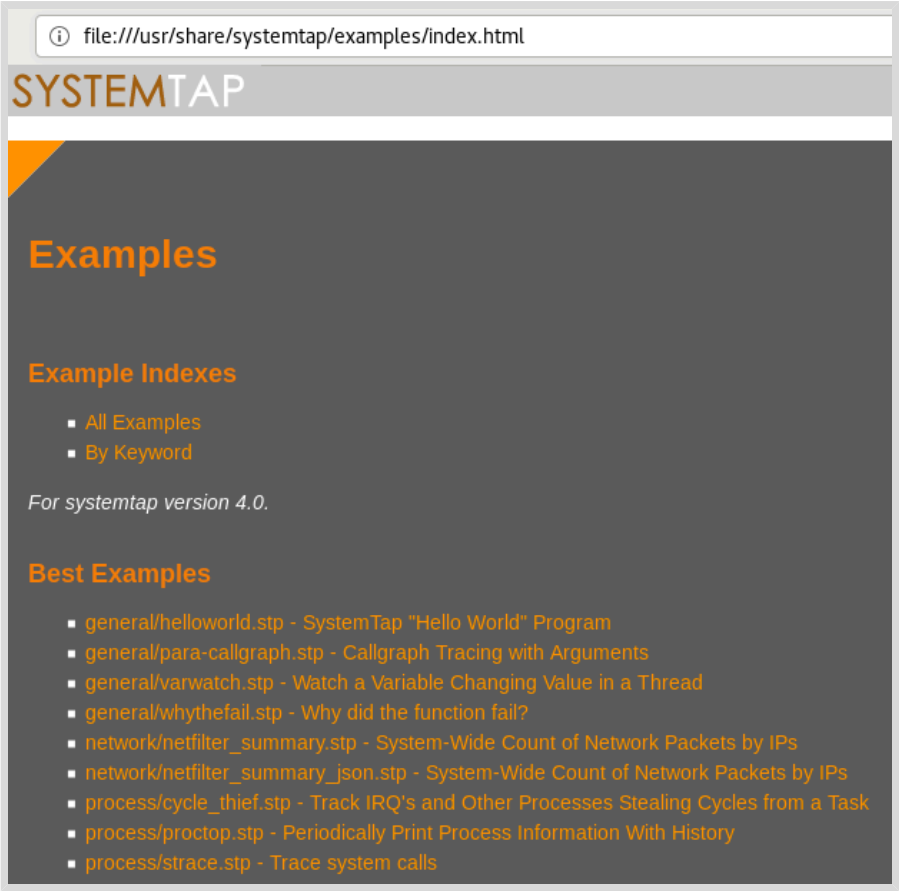


Figure 6.1: SystemTap Examples Installed Locally

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

`stap`(1) and `staprun`(8) man pages

SystemTap
https://sourceware.org/systemtap/wiki

Red Hat Enterprise Linux 7 SystemTap Beginners Guide
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/systemtap_beginners_guide/index

Knowledgebase: "How to download debuginfo packages?"
https://access.redhat.com/solutions/9907

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Gathering Diagnostic Data Using SystemTap

In this exercise, you will diagnose application behavior using existing SystemTap routines.

Outcomes

You should be able to:

- Install the packages needed to compile and run SystemTap programs.
- Build and execute a SystemTap instrumentation module.
- Deploy a compiled SystemTap module to another system.

Before You Begin

Log in as the **student** user on **workstation**, using **student** as the password.

Run the **lab tracing-diagnose start** command. This command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab tracing-diagnose start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. On **servera** identify the version of the running kernel.

```
[root@servera ~]# uname -r
4.18.0-80.el8.x86_64
```

3. Install the matching *kernel-debuginfo* and *kernel-debuginfo-common* packages for your kernel. Also install the *kernel-devel* package for your kernel, and the *systemtap* package, and any dependencies, like *gcc*.

```
[root@servera ~]# yum install kernel-debuginfo-4.18.0-80.el8.x86_64 \
kernel-debuginfo-common-x86_64-4.18.0-80.el8.x86_64 \
kernel-devel-4.18.0-80.el8.x86_64 \
systemtap
...output omitted...
Complete!
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

4. Test your configuration by running a very small SystemTap program.

```
[root@servera ~]# stap -e 'probe begin { printf("Hello World!\n"); exit()}'
Hello World!
```

The command will compile a kernel module, insert it into the kernel, run the module to print out "Hello World!", then exit. The module will unload and the SystemTap program will exit. You have validated that your system is configured properly for SystemTap.

5. Use **`stap`** with the verbose option to compile and run the SystemTap script **`/usr/share/systemtap/examples/io/disktop.stp`**.

   Once execution reaches **`Pass 5: starting run`**, generate activity by logging in to **`servera`** from **`workstation`**. Run various commands in the second terminal, such as open a man page or **`ping`** another system.

```
[root@servera ~]# stap -v /usr/share/systemtap/examples/io/disktop.stp
Pass 1: parsed user script and 474 library scripts using
 394828virt/76668res/9200shr/67104data kb, in 150usr/20sys/164real ms.
Pass 2: analyzed script: 6 probes, 35 functions, 12 embeds, 36 globals using
 591184virt/274328res/10796shr/263460data kb, in 1400usr/340sys/1759real ms.
Pass 3: using cached /root/.systemtap/cache/6d/
stap_6d72ad9b267cc9f9c84a8c8a6d80bd0b_31753.c
Pass 4: using cached /root/.systemtap/cache/6d/
stap_6d72ad9b267cc9f9c84a8c8a6d80bd0b_31753.ko
Pass 5: starting run.

Fri Jul 12 23:12:29 2019 , Average: 131Kb/sec, Read:     648Kb, Write:     10Kb

     UID      PID     PPID                       CMD   DEVICE    T        BYTES
       0    30013      742                      sshd     vda1    R       140179
       0    30030    30029                      bash     vda1    R       131361
       0    30017        1                 (systemd)     vda1    R        50942
       0    30017        1                   systemd     vda1    R        36356
       0    30016        1           systemd-user-ru     vda1    R        29265
       0    30018        2           systemd-cgroups     vda1    R        28224
       0    30019        2           systemd-cgroups     vda1    R        28224
       0    30025    30024           30-systemd-envi     vda1    R        28224
       0    30027    30017                 systemctl     vda1    R        28224
       0    30028        2           systemd-cgroups     vda1    R        28224
```

Press the **`Ctrl`**+**`C`** keys to end data collection.

6. Change your current working directory to **`/tmp`**.

```
[root@servera ~]# cd /tmp
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

7. Run the **stap** command again, but this time, stop at **Pass 4**. Write out the compiled module into a file called **disktop**.

```
[root@servera tmp]# stap -v -p 4 -m disktop \
/usr/share/systemtap/examples/io/disktop.stp
Pass 1: parsed user script and 474 library scripts using
 394696virt/76032res/8796shr/66972data kb, in 140usr/20sys/167real ms.
Pass 2: analyzed script: 6 probes, 35 functions, 12 embeds, 36 globals using
 591188virt/274036res/10508shr/263464data kb, in 1430usr/340sys/1769real ms.
Pass 3: translated to C into "/tmp/stapmZAmDS/disktop_src.c" using
 591188virt/274164res/10636shr/263464data kb, in 30usr/60sys/92real ms.
disktop.ko
Pass 4: compiled C into "disktop.ko" in 2760usr/580sys/3113real ms.
```

8. Configure the **student** user to load and execute compiled SystemTap instrumentation modules by adding this user to the **stapusr** group.

```
[root@servera tmp]# usermod -a -G stapusr student
```

9. Switch to the **student** user account and attempt to load and run the **/tmp/disktop.ko** module. The attempt should fail.

```
[root@servera tmp]# su - student
[student@servera ~]$ staprun /tmp/disktop.ko
ERROR: Members of the "stapusr" and "stapsys" groups can only use unsigned modules
  within the "/lib/modules/4.18.0-80.el8.x86_64/systemtap" directory.
  Unable to canonicalize that directory: No such file or directory
ERROR: Unable to verify the signature for the module /tmp/disktop.ko.
```

This is the expected result. As the error message indicates, members of the **stapusr** group can only load modules stored in **/lib/modules/$(uname -r)/systemtap** directory.

Return to the **root** user account on **servera**.

```
[student@servera ~]$ exit
[root@servera tmp]#
```

10. Create the **/lib/modules/$(uname -r)/systemtap** directory, and then copy the **disktop.ko** module there.

```
[root@servera tmp]# mkdir /lib/modules/$(uname -r)/systemtap
[root@servera tmp]# cp /tmp/disktop.ko /lib/modules/$(uname -r)/systemtap
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

11. Switch to the **student** user's account. Use the **staprun** command to load and execute the **disktop.ko** module. Instead of specifying the file location of the module, use the module name without the **.ko** extension.

If no activity occurs when running the **staprun** command, return go to **workstation** and open a second terminal. Use **ssh** to log in to **servera** as **student** to trigger activity in the first terminal.

```
[root@servera tmp]# su - student
[student@servera ~]$ staprun disktop

Tue Jul 23 06:27:01 2019 , Average:  72Kb/sec, Read:     355Kb, Write:      8Kb

     UID      PID     PPID                       CMD   DEVICE    T        BYTES
    1000     8050     8049                      bash     vda1    R       131494
       0     8046      732                      sshd     vda1    R       115178
       0     8048     8046               unix_chkpwd     vda1    R        12044
    1000     8066     8050                      grep     vda1    R         7774
    1000     8052     8051                        id     vda1    R         7555
    1000     8056     8055                        id     vda1    R         7555
    1000     8058     8057                        id     vda1    R         7555
       0      643        1                    auditd     vda1    W         7214
    1000     8072     8071                       sed     vda1    R         6097
    1000     8065     8064                  dircolors     vda1    R         6046
...output omitted...
```

Press **Ctrl**+**C** to end data collection.

Log out from both **servera** terminals to return to **workstation** as **student**.

12. On **workstation**, open a terminal session and use **ssh** to log in to **serverb** as the **student** user, then switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

13. Install the *systemtap-runtime* package.

```
[root@serverb ~]# yum install systemtap-runtime
...output omitted...
Complete!
```

14. Make a new **/lib/modules/$(uname -r)/systemtap** directory on **serverb**.

```
[root@serverb ~]# mkdir /lib/modules/$(uname -r)/systemtap
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

15. In a previous exercise, you compiled **disktop.ko** on **servera**. Copy this compiled module to **serverb**.

```
[root@serverb ~]# scp root@servera:/lib/modules/$(uname -r)/systemtap/disktop.ko \
/lib/modules/$(uname -r)/systemtap
root@servera's password: redhat
disktop.ko                                    100%  134KB 134.3KB/s   00:00
[root@serverb ~]#
```

16. Add the **student** user to the **stapusr** group.

```
[root@serverb ~]# usermod -a -G stapusr student
```

17. Switch to **student**, and run the **staprun** command to load and execute the **disktop** module.

    If no activity occurs when running **staprun**, go to **workstation** and open a second terminal. Use **ssh** to log in to **serverb** as **student** to trigger activity in the first terminal.

```
[root@serverb ~]# su - student
[student@serverb ~]$ staprun disktop
Mon Jul 15 00:41:04 2019 , Average:  72Kb/sec, Read:     355Kb, Write:      8Kb

     UID      PID     PPID                      CMD   DEVICE    T        BYTES
    1000    24802    24801                     bash     vda1    R       131328
       0    24798      730                     sshd     vda1    R       115178
       0    24800    24798              unix_chkpwd     vda1    R        12044
    1000    24818    24802                     grep     vda1    R         7774
    1000    24804    24803                       id     vda1    R         7555
    1000    24808    24807                       id     vda1    R         7555
    1000    24810    24809                       id     vda1    R         7555
       0      642        1                   auditd     vda1    W         7234
    1000    24824    24823                      sed     vda1    R         6097
    1000    24817    24816                 dircolors     vda1    R         6046
...output omitted...
```

    Press the **Ctrl**+**C** keys to end data collection.

18. Log out from both **serverb** terminal sessions to return to **workstation** as **student**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab tracing-diagnose finish** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab tracing-diagnose finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Tracing System Events with eBPF Tools

Objectives

After completing this section, you should be able to debug applications using tools instrumented by the in-kernel eBPF virtual machine.

Introduction to eBPF

Extended Berkeley Packet Filter (eBPF) is an in-kernel virtual machine that runs its own bytecode in a secured, restricted environment. Originally designed for capturing and filtering network packets, it has been extended to take advantage of modern hardware and provides a safe way to extend kernel functionality. eBPF is enabled from userspace, is restricted to the amount of memory it uses, and has limited access to devices.

eBPF works by attaching programs to various kernel points, such as kprobes, tracepoints, and perf events. There are many performance analysis tools that leverage the extended kernel functionality, allowing for more than just filtering packets. These enhancements allow custom analysis programs to be executed on Linux for dynamic tracing, static tracing, and profiling events.

You can write your own eBPF programs, or use existing programs available with one of the available front-end applications. One popular front-end application for BPF tracing is the *BPF Compiler Collection (BCC)*.

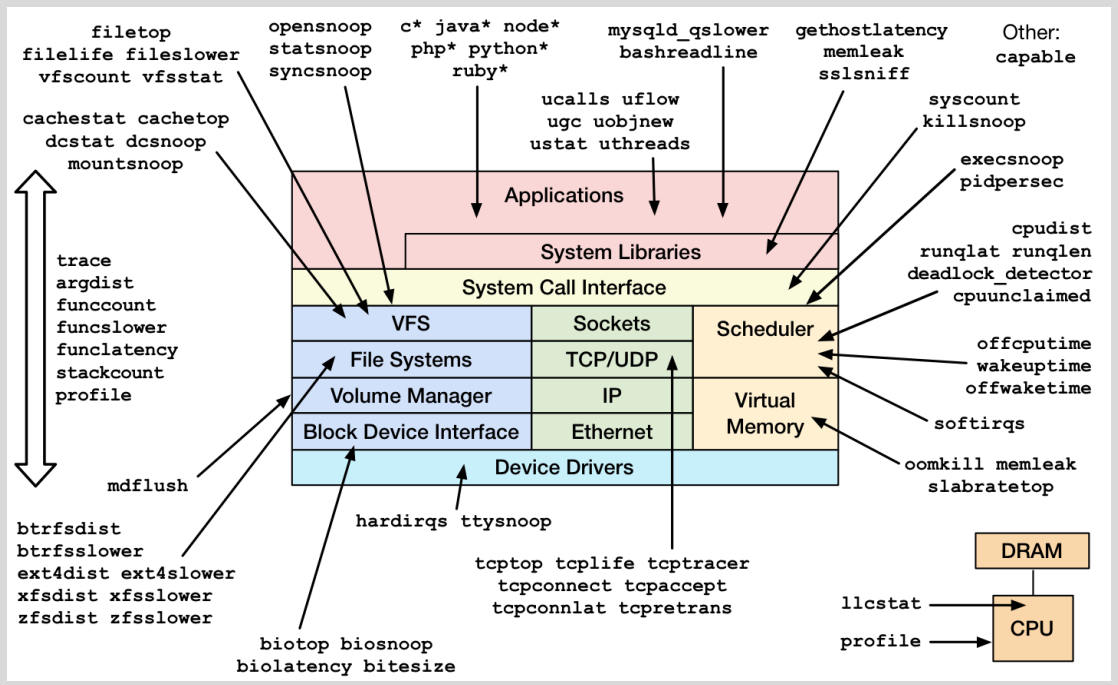


Figure 6.2: Linux bcc/BPF Training Tools by Brendan Gregg (CC BY-SA 4.0)

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Introduction to BPF Compiler Collection

Writing eBPF programs requires compiling and linking to the eBPF library from the kernel source. This works well for kernel developers, but for others, such as those who work on production systems, having preexisting programs may be the more desired approach. BCC consists of the necessary components for writing programs, and also provides example programs and preexisting tools for debugging and diagnosing performance issues.

Installing BCC

A single package, *bcc-tools*, is required to install the command line tools for the BPF compiler collection. On Red Hat Enterprise Linux 8, the *bcc-tools* package is part of the **`rhel-8.0-for-x86_64-appstream-rpms`** repository.

```
[root@host ~]# yum install bcc-tools
```

List the contents of the **`tools`** and **`doc`** directories to confirm the installation.

```
[root@host ~]# ls /usr/share/bcc/tools
```

```
[root@host ~]# ls /usr/share/bcc/tools/doc
```

Available BCC resources:

- The BCC tools are located in the **`/usr/share/bcc/tools`** directory.
- The BCC tool example files are located in the **`/usr/share/bcc/tools/doc`** directory.
- Over 100 man pages are available. Use the **`man bcc-TOOL`** command to access the man page for a given tool.

Using BCC Tools

BCC provides a large collection of tracing examples and tools. The following example programs are executing from within the **`/usr/share/bcc/tools`** directory.

Tracing New Processes

The **`execsnoop`** tool traces new processes. For example, **`execsnoop`** traces the **`execve()`** system call and shows details of the arguments and return value. It catches new processes in the **`fork->exec`** sequence. Applications that **`fork()`** but do not **`exec()`**, such as worker processes, are not included in the **`execsnoop`** output.

The following example traces the commands invoked when running the **`man ls`** command from a second terminal:

```
[root@host tools]# ./execsnoop
PCOMM            PID    PPID   RET ARGS
man              18517  18494    0 /usr/bin/man ls
preconv          18525  18517    0 /usr/bin/preconv -e UTF-8
tbl              18527  18517    0 /usr/bin/tbl
nroff            18528  18517    0 /usr/bin/nroff -mandoc -Tutf8
gzip             18531  18517    0 /usr/bin/gzip -c7
less             18530  18517    0 /usr/bin/less
locale           18534  18533    0 /usr/bin/locale charmap
```

```
groff            18535  18528    0 /usr/bin/groff -mtty-char -Tutf8 -mandoc
grotty           18537  18535    0 /usr/bin/grotty
troff            18536  18535    0 /usr/bin/troff -mtty-char -mandoc -Tutf8
```

Column Descriptions

- **PCOMM** - Command name of the parent process.
- **PID** - Process ID.
- **PPID** - Parent process ID.
- **RET** - Return value.
- **ARGS** - File name with arguments.

Tracing open() Syscalls

The **opensnoop** tool traces the **open()** syscall system-wide, and displays the process name and path name details. **opensnoop** is useful for discovering configuration and log files during application startup.

The following example traces the **open()** syscalls invoked when the **ssh** command is executed from a second terminal.

The output shows that during execution, the **ssh** command opens several library and configuration files. However, the command fails to locate the **/root/.ssh/config** file. Error number **2** is ENOENT: no such file or directory.

```
[root@host tools]# ./opensnoop
PID    COMM               FD ERR PATH
18560  ssh                 3   0 /lib64/libkeyutils.so.1
18560  ssh                 3   0 /dev/null
18560  ssh                 3   0 /proc/18560/fd
18560  ssh                 3   0 /etc/nsswitch.conf
18560  ssh                 3   0 /etc/ld.so.cache
18560  ssh                 3   0 /lib64/libnss_sss.so.2
18560  ssh                 3   0 /var/lib/sss/mc/passwd
18560  ssh                 5   0 /etc/ld.so.cache
18560  ssh                 5   0 /lib64/libnss_files.so.2
18560  ssh                 5   0 /etc/passwd
18560  ssh                 5   0 /usr/lib/locale/locale-archive
18560  ssh                -1   2 /root/.ssh/config
18560  ssh                 5   0 /etc/ssh/ssh_config
```

Column Descriptions

- **PID** - Process ID.
- **COMM** - Command name.
- **FD** - File descriptor.
- **ERR** - System error number.
- **PATH** - Path details.

Tracing XFS Operations

The **xfsslower** tool shows XFS reads, writes, opens, and fsyncs, slower than the default threshold of 10 ms.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The output shows several reads from the **`randread.pl`** command, each 8 KiB in size. Notice that all reads have over 10 ms latency.

```
[root@host tools]# ./xfsslower
Tracing XFS operations slower than 10 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
06:23:06 randread.pl    32497  R 8192    24938024   17.93 data1
06:23:06 randread.pl    32521  R 8192    13431528   18.27 data1
06:23:08 randread.pl    32497  R 8192    5070904    16.37 data1
06:23:08 randread.pl    32521  R 8192    12693016   16.06 data1
06:23:18 randread.pl    32521  R 8192    27049136   21.68 data1
06:23:18 randread.pl    32497  R 8192    257864     21.74 data1
06:23:20 randread.pl    32497  R 8192    17797208   13.37 data1
06:23:20 randread.pl    32497  R 8192    6088224    19.74 data1
```

Column Descriptions

- **`TIME`** - Time of I/O completion since the first I/O seen, in seconds.
- **`COMM`** - Process name.
- **`PID`** - Process ID.
- **`T`** - Type of operation. R for read, W for write, O for open, S for fsync.
- **`BYTES`** - Size of I/O, in bytes.
- **`OFF_KB`** - File offset for the I/O, in kilobytes.
- **`LAT(ms)`** - Latency (duration) of I/O, measured from when it was issued by VFS to the file system, to when it completed. This time is inclusive of block device I/O, file system CPU cycles, file system locks, run queue latency, and others. It is a more accurate measure of the latency suffered by applications performing file system I/O, than are measurements at the block device interface.
- **`FILENAME`** - A cached kernel file name (comes from dentry->d_iname).

Tracing Block Device Latency

The **`biolatency`** tool traces block device I/O (disk I/O), and records the distribution of I/O latency (time). Pressing the **`Ctrl`**+**`C`** keys ends tracing, and then generates a histogram that is displayed to the terminal.

The following example shows a large mode of latency from about **128** microseconds to about **32767** microseconds (**33** milliseconds). The bulk of the I/O was between **1** and **8** milliseconds, which is the expected block device latency for rotational storage devices.

The highest latency was between **65** and **131** milliseconds, as shown in the last row displayed for which there were **2** I/O.

```
[root@host tools]# ./biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
     usecs           : count     distribution
       0 -> 1        : 0        |                                        |
       2 -> 3        : 0        |                                        |
       4 -> 7        : 0        |                                        |
       8 -> 15       : 0        |                                        |
      16 -> 31       : 0        |                                        |
      32 -> 63       : 0        |                                        |
      64 -> 127      : 1        |                                        |
```

```
     128 -> 255      : 12       |********                                |
     256 -> 511      : 15       |**********                              |
     512 -> 1023     : 43       |*******************************         |
    1024 -> 2047     : 52       |****************************************|
    2048 -> 4095     : 47       |**********************************      |
    4096 -> 8191     : 52       |****************************************|
    8192 -> 16383    : 36       |**************************              |
   16384 -> 32767    : 15       |**********                              |
   32768 -> 65535    : 2        |*                                       |
   65536 -> 131071   : 2        |*                                       |
```

Tracing Block Device I/O

The **biosnoop** tool traces block device I/O (disk I/O), and prints various details in a one line summary for each I/O.

Output from the **biosnoop** tool identifies excessive writes to a disk which allows administrators to use other tools to investigate system workload.

```
[root@host tools]# ./biosnoop
TIME(s)        COMM           PID    DISK    T  SECTOR    BYTES   LAT(ms)
0.000000000    xfsaild/vda1   461    vda     R  20712288  16384      0.48
0.000099000    kworker/1:2    19277  vda     W  8203154   2048       0.35
0.002909000    xfsaild/vda1   461    vda     R  20712224  16384      2.80
0.003328000    xfsaild/vda1   461    vda     R  20712736  16384      0.34
0.009899000    xfsaild/vda1   461    vda     W  20713568  49152      0.21
1.023791000    kworker/u4:4   425    vda     W  716576    4096       0.31
30.208005000   xfsaild/vda1   461    vda     W  2176      16384      0.48
30.208023000   xfsaild/vda1   461    vda     W  2072      8192       0.50
30.208027000   xfsaild/vda1   461    vda     W  2050      512        0.50
30.214527000   kworker/1:2    19277  vda     W  8203158   2048       0.34
60.415982000   xfsaild/vda1   461    vda     W  2056      8192       0.44
60.416015000   xfsaild/vda1   461    vda     W  133952    16384      0.47
60.416019000   xfsaild/vda1   461    vda     W  2049      512        0.48
60.416030000   kworker/1:2    19277  vda     W  8203162   2560       0.26
67.302397000   dd             19324  vda     W  4743656   4096       0.28
67.305475000   dd             19324  vda     W  8203167   2560       0.17
...output omitted...
```

Column Descriptions

- **TIME** - Time of I/O completion since the first I/O seen, in seconds.
- **COMM** - Cached process name, if present. This usually identifies the responsible process for the I/O, although there are exceptions.
- **PID** - Cached process ID, if present. This usually identifies the responsible process for the I/O, although there are exceptions.
- **DISK** - Disk device name.
- **T** - Type of operation. R for read, W for write.
- **SECTOR** - Device sector for the I/O.
- **BYTES** - Size of the I/O, in bytes.
- **LAT(ms)** - Time for the I/O (latency), from the issue to the device to its completion, in milliseconds.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Tracing Cache HITS and MISSES

The **`cachestat`** tool traces kernel page cache functions, and prints a per second summary. The **`cachestat`** output is useful for general workload characterization, and for identifying patterns in operation usage over time.

The following activity was generated by running the **`dd if=/dev/`*`disk`*` of=/dev/null bs=8192`** command in a second terminal. The following command shows a file the size of *disk* (approximately 10 GiB) being read into the page cache.

```
[root@host tools]# ./cachestat
   TOTAL   MISSES     HITS  DIRTIES   BUFFERS_MB  CACHED_MB
       1        0        1        0          247        961
       0        0        0        0          247        961
     706        0      706        0          247        961
       0        0        0        0          247        961
       0        0        0        0          247        961
  644994   631836    13158        0         1113        376
 1528250  1528250        0        0         1392         67
  448566   447712      854        0         1398         56
       0        0        0        0         1398         56
...output omitted...
```

Column Descriptions

- **`TOTAL`** - Total cache accesses without counting dirties.
- **`MISSES`** - Number of page cache misses.
- **`HITS`** - Number of page cache hits.
- **`DIRTIES`** - Number of dirty pages added to the page cache.
- **`BUFFERS_MB`** - Buffers size taken from **`/proc/meminfo`**.
- **`CACHED_MB`** - Cached amount of data in current page cache taken from **`/proc/meminfo`**.

Tracing Host Name Lookup Latency

The **`gethostlatency`** tool traces host name lookup calls to **`getaddrinfo()`** and **`gethostbyname()`** functions. It displays the latency of the calls (duration) in milliseconds, and the host string. This tool is useful for identifying DNS latency by identifying which remote host name lookups were slow, and by how much.

The following activity was generated by connections from a second terminal to various hosts. Notice that the last line shows the latency of approximately **`172`** milliseconds. This was a connection attempt to a host (`materials.myu.edu`) that could not be resolved.

```
[root@host tools]# ./gethostlatency
TIME      PID    COMM                  LATms HOST
20:16:26  19490  ssh                    0.10 workstation
20:16:26  19490  ssh                    0.49 workstation
20:18:18  19496  wget                   0.02 192.168.0.254
20:19:34  19503  wget                   0.37 materials.example.com
20:19:39  19504  wget                   0.30 materials.example.com
20:20:08  19505  wget                   0.17 materials.example.com
20:21:56  19507  wget                 172.57 materials.myu.edu
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Column Descriptions

- **`TIME`** - Time of the command (HH:MM:SS).
- **`PID`** - Process ID of the client performing the call.
- **`COMM`** - Process (command) name of the client performing the call.
- **`LATms`** - Latency of the calls (duration) in milliseconds.
- **`HOST`** - Host name string: the target of the lookup.

References

`bcc`(1), and **`bcc-`*`toolname`*`(8)`** man pages, where *toolname* is any of the tools included in the bcc package, such as **`execsnoop`** and **`biolatency`**.

For more information, refer to the *Analyzing system performance with BPF Compiler Collection* chapter in the *Red Hat Enterprise Linux 8 Managing, monitoring and updating the kernel Guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_monitoring_and_updating_the_kernel/analyzing-system-performance-with-bpf-compiler_collection_managing-monitoring-and-updating-the-kernel

Source list of bcc tools
https://github.com/iovisor/bcc/tree/master/tools

The original graphic by Brendan Gregg can be found at
Linux bcc/BPF Tracing Tools
http://www.brendangregg.com/linuxperf.html
It is licensed under Attribution-ShareAlike 4.0 International
https://creativecommons.org/licenses/by-sa/4.0/legalcode

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Tracing System Events with eBPF Tools

In this exercise, you will run tools that use eBPF to debug application behavior.

Outcomes

You should be able to interpret the output of various commands provided by the **bcc** front-end to eBPF.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab tracing-debug start** command to prepare the host, **servera**, for this exercise.

```
[student@workstation ~]$ lab tracing-debug start
```

1. On **workstation**, open two terminals side by side. On each terminal, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Install the *bcc-tools* package.

```
[root@servera ~]# yum install bcc-tools
...output omitted...
Transaction Summary
=====================================================================
Install  37 Packages
...output omitted...

Complete!
```

3. Execute the **execsnoop** command to trace new processes as they make system calls. Observe details of arguments and return values as the command follows fork->exec sequences.

   3.1. In the first terminal, execute the **execsnoop** command.

```
[root@servera ~]# /usr/share/bcc/tools/execsnoop
PCOMM            PID    PPID   RET ARGS
```

Output does not display until a new process is executed.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3.2. In the second terminal, execute the **man man** command, and then observe the activity in the first terminal running the **execsnoop** command. You will see one line of output for each new process.

```
[root@servera ~]# man man
```

```
PCOMM            PID    PPID   RET ARGS
man              6847   1387     0 /bin/man man
tbl              6857   6847     0 /bin/tbl
preconv          6856   6847     0 /bin/preconv -e UTF-8
nroff            6858   6847     0 /bin/nroff -mandoc -rLL=91n -rLT=91n -Tutf8
less             6861   6847     0 /bin/less
locale           6864   6863     0 /bin/locale charmap
groff            6865   6858     0 /usr/bin/groff -mtty-char -Tutf8 -mandoc -
rLL=91n -rLT=91n
grotty           6867   6865     0 /usr/bin/grotty
troff            6866   6865     0 /usr/bin/troff -mtty-char -mandoc -rLL=91n -
rLT=91n -Tutf8
```

3.3. In the first terminal, press **Ctrl+C** to exit the **execsnoop** command.

In the second terminal, press **q** to exit the **man** command.

4. Execute the **opensnoop** command to trace all open() syscalls. Observe how the **dummy.sh** command works by tracing details such as various data files, configuration files, and other files that are open during execution.

4.1. In the first terminal, execute the **opensnoop** command.

```
[root@servera ~]# /usr/share/bcc/tools/opensnoop
PID    COMM               FD ERR PATH
...output omitted...
```

Output will display system-wide open() syscalls for running applications, such as the **irqbalance** command. However, in the next step you will execute a command that will allow you to trace specific application open() syscalls.

4.2. In the second terminal, execute the **dummy.sh** command, and then observe the activity in the first terminal running the **opensnoop** command. You will see one line of output for each open() syscall until the **100MBfile** file is opened.

```
[root@servera ~]# dummy.sh
```

```
PID    COMM               FD ERR PATH
6922   dummy.sh            3   0 /etc/ld.so.cache
6922   dummy.sh            3   0 /lib64/libtinfo.so.6
6922   dummy.sh            3   0 /lib64/libdl.so.2
6922   dummy.sh            3   0 /lib64/libc.so.6
6922   dummy.sh            3   0 /dev/tty
6922   dummy.sh            3   0 /usr/lib/locale/locale-archive
6922   dummy.sh            3   0 /usr/lib64/gconv/gconv-modules.cache
6922   dummy.sh            3   0 /root/bin/dummy.sh
6923   dd                  3   0 /etc/ld.so.cache
```

```
6923   dd                  3   0 /lib64/libc.so.6
6923   dd                  3   0 /usr/lib/locale/locale-archive
6923   dd                  3   0 /dev/zero
6923   dd                  3   0 100MBfile
```

4.3. In the first terminal, press **Ctrl+C** to exit the **opensnoop** command.

In the second terminal, press **Ctrl+C** to exit the **dummy.sh** command.

5. Execute the legacy **iostat** command to generate a device utilization report. Execute the **eBPF** front-end, **bcc biosnoop** command, to generate block device I/O with process, disk, and latency details. Compare the output of the legacy **iostat** command to that of the **bcc biosnoop** command.

5.1. In the first terminal, execute the **iostat -dxy 1 1** command. Use the **-dxy** options to display device and extended statistics, but omit the first report with statistics since system boot. Execute the command for one interval and a count of one.

```
[root@servera ~]# iostat -dxy 1 1
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com) 07/21/2019 _x86_64_ (2 CPU)

Device     r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm
vda       0.00    0.00      0.00      0.00     0.00     0.00   0.00   0.00
vdb       0.00    0.00      0.00      0.00     0.00     0.00   0.00   0.00
vdc       0.00    0.00      0.00      0.00     0.00     0.00   0.00   0.00
vdd       0.00    0.00      0.00      0.00     0.00     0.00   0.00   0.00

      r_await w_await aqu-sz rareq-sz wareq-sz  svctm  %util
         0.00    0.00   0.00     0.00     0.00   0.00   0.00
         0.00    0.00   0.00     0.00     0.00   0.00   0.00
         0.00    0.00   0.00     0.00     0.00   0.00   0.00
         0.00    0.00   0.00     0.00     0.00   0.00   0.00
```

Notice that the **iostat** command reports the system in what appears to be a completely idle state. However, in the next step you will see that there are disk I/O details that are not reported by the **iostat** command.

5.2. In the second terminal, execute the **biosnoop** command. Allow the **biosnoop** command to run for several seconds, and then press **Ctrl+C** to exit the command.

```
[root@servera ~]# /usr/share/bcc/tools/biosnoop
TIME(s)        COMM           PID    DISK    T  SECTOR    BYTES   LAT(ms)
0.000000000    kworker/1:2    8300   vda     W  8214362   2048       0.39
4.605524000    kworker/u4:1   6696   vda     W  10776     4096       0.35
30.205611000   xfsaild/vda1   462    vda     W  2176      16384      0.39
30.205628000   xfsaild/vda1   462    vda     W  2072      8192       0.41
30.205633000   xfsaild/vda1   462    vda     W  2050      512        0.42
30.208352000   kworker/1:2    8300   vda     W  8214366   2048       0.20
60.413660000   xfsaild/vda1   462    vda     W  11616     16384      0.48
60.413674000   xfsaild/vda1   462    vda     W  2056      8192       0.49
60.413677000   xfsaild/vda1   462    vda     W  2049      512        0.51
90.626166000   xfsaild/vda1   462    vda     W  2048      512        0.29
120.832344000  kworker/1:2    8300   vda     W  8214372   1024       0.33
120.833849000  xfsaild/vda1   462    vda     W  2048      512        0.34
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Press **Ctrl**+**C** to exit the **biosnoop** command.

In the above output, activity is generated by the **xfsaild** process that manages the **XFS** active items list, and the **kworker** process that conducts work for the kernel, such as processing interrupts, I/O system calls, and many others.

6. Execute the **cachestat** command to generate statistics for Linux page cache hit/miss ratios.

6.1. In the first terminal, execute the **cachestat** command. Use the **-T** option to display the timestamp.

```
[root@servera ~]# /usr/share/bcc/tools/cachestat -T
TIME        TOTAL   MISSES     HITS  DIRTIES   BUFFERS_MB  CACHED_MB
21:37:28        0        0        0        0            0       1502
21:37:33        1        0        1        0            0       1502
```

Output will vary depending on your system. Often times, there is an initial spike in cache hits but this too will vary. Notice that there is not much activity presented in the output. However, in the next step you will execute a command that will generate a large number of cache misses.

6.2. In the second terminal, execute the **cache-misses** command. The **cache-misses** command loops through four iterations that read a 2GB file, writing to **/dev/null**, and subsequently generating multiple entries of cache misses. The **cache-misses** command will terminate after the loop execution. You may want to execute the **cache-misses** command more than once to generate additional cache misses.

```
[root@servera ~]# cache-misses
Please wait for command to complete...

[root@servera ~]#
```

In the first terminal, observe the activity in the cache misses and hits columns.

```
[root@servera ~]# /usr/share/bcc/tools/cachestat -T
TIME        TOTAL   MISSES     HITS  DIRTIES   BUFFERS_MB  CACHED_MB
21:49:08        0        0        0        0            0       1484
21:49:13   727812   723196     4616        0            0       1508
21:49:18   325650   322137     3513        0            0       1482
21:49:23        0        0        0        0            0       1482
21:49:28      378      378        0        0            0       1485
21:49:33        0        0        0        0            0       1485
21:49:38        0        0        0        0            0       1485
21:49:43        0        0        0        0            0       1485
21:49:48        0        0        0        0            0       1485
21:49:53     2413      441     1972        0            0       1487
21:49:58      378        0      378        0            0       1487
21:50:03        0        0        0        0            0       1487
21:50:08     1414        0     1414        0            0       1487
21:50:13   525965   522321     3644        0            0       1488
21:50:18  1576263  1566948     9315        0            0       1484
21:50:23      752      132      620        0            0       1485
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
21:50:28        0        0        0        0            0       1485
21:50:33        0        0        0        0            0       1485
```

In the first terminal, press **Ctrl+C** to exit the **cachestat** command.

6.3. Log out from both **servera** terminal sessions to return to **workstation** as the **student** user.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab tracing-debug finish** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab tracing-debug finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Analyzing Performance Using Tracing Tools

Performance Checklist

In this lab, you will analyze various system and application behaviors using tracing tools.

Outcomes

At the end of the lab, you will be able to load and execute a compiled SystemTap module called **`page_faults.ko`** on multiple systems.

Before You Begin

Log in as the **`student`** user on **`workstation`** using **`student`** as the password.

Run the **`lab tracing-review start`** command. This command determines if the hosts, **`servera`** and **`serverb`**, are reachable on the network and prepares them for this exercise.

```
[student@workstation ~]$ lab tracing-review start
```

1. On **`workstation`**, open two terminals side by side. On one of the terminals, use **`ssh`** to log in to **`serverb`** as the **`student`** user, then use **`sudo -i`** to switch to **`root`**.
2. Install the packages necessary to compile and run SystemTap modules on **`serverb`**.
3. Compile the **`/usr/share/systemtap/examples/memory/pfaults.stp`** SystemTap script into a compiled kernel module named **`page_faults.ko`**.
4. Grant the **`student`** user the ability to run compiled SystemTap modules on **`serverb`**.
5. On **`workstation`**, from one of terminals, use **`ssh`** to log in to **`servera`** as **`student`**, then switch to **`root`**.
6. On **`servera`**, configure the **`student`** user to be able to load and execute compiled SystemTap modules. Copy the **`page_faults.ko`** module from **`serverb`** to **`servera`** to test your configuration.
7. On **`workstation`**, open two terminals and then use the **`ssh`** command to log in to **`serverb`** as the **`student`** user. Escalate user privileges to the **`root`** user's account.
8. Install the *bcc-tools* package.
9. On **`serverb`**, trace host name lookups and identify latency in calls made by various commands, such as **`curl`** and **`ping`**.

Evaluation

On **`workstation`**, run the **`lab tracing-review grade`** command to confirm success of this exercise.

```
[student@workstation ~]$ lab tracing-review grade
```

Finish

On **`workstation`**, run the **`lab tracing-review finish`** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab tracing-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Analyzing Performance Using Tracing Tools

Performance Checklist

In this lab, you will analyze various system and application behaviors using tracing tools.

Outcomes

At the end of the lab, you will be able to load and execute a compiled SystemTap module called **`page_faults.ko`** on multiple systems.

Before You Begin

Log in as the **`student`** user on **`workstation`** using **`student`** as the password.

Run the **`lab tracing-review start`** command. This command determines if the hosts, **`servera`** and **`serverb`**, are reachable on the network and prepares them for this exercise.

```
[student@workstation ~]$ lab tracing-review start
```

1. On **`workstation`**, open two terminals side by side. On one of the terminals, use **`ssh`** to log in to **`serverb`** as the **`student`** user, then use **`sudo -i`** to switch to **`root`**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

2. Install the packages necessary to compile and run SystemTap modules on **`serverb`**.

   2.1. On **`serverb`**, identify the version of the running kernel.

```
[root@serverb ~]# uname -r
4.18.0-80.el8.x86_64
```

   2.2. Install the following kernel packages that match the running kernel, followed by the *systemtap* package:

   - *kernel-debuginfo-4.18.0-80.el8.x86_64*
   - *kernel-debuginfo-common-x86_64-4.18.0-80.el8.x86_64*
   - *kernel-devel-4.18.0-80.el8.x86_64*
   - *systemtap*

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# yum install kernel-debuginfo-4.18.0-80.el8.x86_64 \
kernel-debuginfo-common-x86_64-4.18.0-80.el8.x86_64 \
kernel-devel-4.18.0-80.el8.x86_64 \
systemtap
...output omitted...
Complete!
```

2.3. Test your configuration by running a very small SystemTap program.

```
[root@serverb ~]# stap -e 'probe begin { printf("Hello World!\n"); exit()}'
Hello World!
```

3. Compile the **`/usr/share/systemtap/examples/memory/pfaults.stp`** SystemTap script into a compiled kernel module named **`page_faults.ko`**.

```
[root@serverb ~]# stap -v -p 4 -m page_faults \
/usr/share/systemtap/examples/memory/pfaults.stp
Pass 1: parsed user script and 474 library scripts using
 394696virt/76188res/8956shr/66972data kb, in 150usr/20sys/163real ms.
Pass 2: analyzed script: 3 probes, 7 functions, 5 embeds, 7 globals using
 694120virt/363508res/10616shr/366396data kb, in 2570usr/1220sys/3739real ms.
Pass 3: translated to C into "/tmp/stapwoLny5/page_faults_src.c" using
 694120virt/363636res/10744shr/366396data kb, in 20usr/60sys/82real ms.
page_faults.ko
Pass 4: compiled C into "page_faults.ko" in 1870usr/510sys/2208real ms.
[root@serverb ~]#
```

4. Grant the **`student`** user the ability to run compiled SystemTap modules on **`serverb`**.

4.1. Add **`student`** user to the **`stapuser`** group.

```
[root@serverb ~]# usermod -a -G stapusr student
```

4.2. Create the **`/lib/modules/$(uname -r)/systemtap`** directory and copy the compiled **`page_faults.ko`** module there.

```
[root@serverb ~]# mkdir /lib/modules/$(uname -r)/systemtap
[root@serverb ~]# cp page_faults.ko /lib/modules/$(uname -r)/systemtap
```

4.3. Switch to the **`student`** user's account to verify that the **`student`** user can load and execute the **`page_faults.ko`** SystemTap module.

```
[root@serverb ~]# su - student
[student@serverb ~]$ staprun page_faults
```

From the second terminal on **`workstation`**, use the **`ssh`** command to log in to **`serverb`** as the **`student`** user. This will generate page fault activity in the first terminal.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$
```

```
55311354:864:94288376525632d:w:minor:5
55311359:864:140665936370752d:w:minor:0
55311363:864:94288402105272d:w:minor:0
55311366:864:140665936381176d:w:minor:0
55311381:864:140665936364816d:w:minor:1
55311386:864:140737160224472d:w:minor:1
55311388:864:140737160220376d:w:minor:1
55311391:864:140737160216280d:w:minor:1
55311393:864:140737160212184d:w:minor:1
55311395:864:140737160208088d:w:minor:0
55311397:864:140737160203992d:w:minor:0
55311400:864:140737160199896d:w:minor:1
55311402:864:140737160195800d:w:minor:1
...output omitted...
```

Press **Ctrl**+**C** to exit the **page_faults** module.

4.4. Exit from all **serverb** terminals to return to **workstation** as **student** user.

5. On **workstation**, from one of terminals, use **ssh** to log in to **servera** as **student**, then switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

6. On **servera**, configure the **student** user to be able to load and execute compiled SystemTap modules. Copy the **page_faults.ko** module from **serverb** to **servera** to test your configuration.

6.1. Add **student** user to the **stapuser** group.

```
[root@servera ~]# usermod -a -G stapusr student
```

6.2. Create the **/lib/modules/$(uname -r)/systemtap** directory.

```
[root@servera ~]# mkdir /lib/modules/$(uname -r)/systemtap
```

6.3. Copy the **page_faults.ko** module built on **serverb** to the **/lib/modules/$(uname -r)/systemtap** on **servera**.

```
[root@servera ~]# cd /lib/modules/$(uname -r)/systemtap/
[root@servera systemtap]# scp \
root@serverb:/lib/modules/$(uname -r)/systemtap/page_faults.ko .
Are you sure you want to continue connecting (yes/no)? yes
root@serverb's password: redhat
page_faults.ko                      100% 1075KB  48.7MB/s   00:00
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

6.4. Switch to the **student** user's account to verify that the **student** user can load and execute the **page_faults.ko** SystemTap module.

```
[root@servera systemtap]# su - student
[student@servera ~]$ staprun page_faults
```

From the second terminal on **workstation**, use the **ssh** command to log in to **servera** as the **student** user. This will generate page fault activity in the first terminal.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$
```

```
55311354:864:94288376525632d:w:minor:5
55311359:864:140665936370752d:w:minor:0
55311363:864:94288402105272d:w:minor:0
55311366:864:140665936381176d:w:minor:0
55311381:864:140665936364816d:w:minor:1
55311386:864:140737160224472d:w:minor:1
55311388:864:140737160220376d:w:minor:1
55311391:864:140737160216280d:w:minor:1
55311393:864:140737160212184d:w:minor:1
55311395:864:140737160208088d:w:minor:0
55311397:864:140737160203992d:w:minor:0
55311400:864:140737160199896d:w:minor:1
55311402:864:140737160195800d:w:minor:1
...output omitted...
```

Press **Ctrl**+**C** to exit the **page_faults** module.

6.5. Exit from all terminals on **servera** to return to **workstation** as the **student** user.

7. On **workstation**, open two terminals and then use the **ssh** command to log in to **serverb** as the **student** user. Escalate user privileges to the **root** user's account.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

8. Install the *bcc-tools* package.

```
[root@serverb ~]# yum install bcc-tools
...output omitted...
Complete!
```

9. On **serverb**, trace host name lookups and identify latency in calls made by various commands, such as **curl** and **ping**.

9.1. In the first terminal, execute the **gethostlatency** command.

```
[root@serverb ~]# /usr/share/bcc/tools/gethostlatency
TIME      PID    COMM                  LATms HOST
```

Output does not display until a new host lookup is processed.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

9.2. In the second terminal, execute the **`curl http://classroom.example.com`**, **`ping -c2 workstation`**, and **`ssh servera ls`** commands, and then observe the activity in first terminal where the **`gethostlatency`** command is tracing host name lookups. Note the latency times in the **`LATms`** column.

```
[root@serverb ~]# curl http://classroom.example.com
...output omitted...
[root@serverb ~]# ping -c2 workstation
...output omitted...
[root@serverb ~]# ssh servera ls
...output omitted...
```

```
TIME      PID    COMM                 LATms HOST
14:42:39  8099   curl                  0.23 classroom.example.com
14:44:09  8102   ping                  0.55 workstation
14:44:46  8103   ssh                   0.01 servera
14:44:46  8103   ssh                   0.08 servera
14:44:50  8103   ssh                   0.03 servera
14:44:50  8103   ssh                   0.10 servera
```

9.3. In the first terminal, press **`Ctrl`**+**`C`** to exit the **`gethostlatency`** command.

9.4. Exit from all **`serverb`** terminals to return to **`workstation`** as the **`student`** user.

Evaluation

On **`workstation`**, run the **`lab tracing-review grade`** command to confirm success of this exercise.

```
[student@workstation ~]$ lab tracing-review grade
```

Finish

On **`workstation`**, run the **`lab tracing-review finish`** command to complete this exercise. This command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab tracing-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- The **`perf`** command allows administrators to observe performance counters that track hardware and software events, including:
  - Number of instructions executed.
  - Cache misses.
  - Context switches.
  - CLI to collect events in real-time, or record events for later reporting.
- The SystemTap framework allows easy probing and instrumentation of almost any component within the kernel. SystemTap scripts specify where to attach probes and what data to collect when the probe executes.
- The Extended Berkeley Packet Filter (eBPF) is a performance analysis tool that uses front-end applications, such as the BPF Compiler Collection (BCC), to perform dynamic and static tracing of profiling events.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 7

Tuning CPU Utilization

Goal

Manage CPU resource sharing and scheduling to control utilization.

Objectives

- Configure CPU scheduling policies.
- Configure CPU and interrupt affinity to control application isolation and CPU resource commitments.
- Describe how the CPU caches are used by applications.

Sections

- Configuring CPU Scheduling Policies and Tunables (and Guided Exercise)
- Configuring Affinity and Isolation (and Guided Exercise)
- Profiling Cache Use (and Guided Exercise)

Lab

Tuning CPU Utilization

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring CPU Scheduling Policies and Tunables

Objectives

After completing this section, you should be able to configure CPU scheduling policies.

Scheduling Processes

A single CPU can only execute one process at a time. Although the Linux system may appear to be running multiple processes simultaneously by *multitasking*, when several processes run concurrently on a single CPU it is by interleaving the execution of those processes.

The kernel uses the *process scheduler* to determine which process to run at any given point in time. The process scheduler must balance several options, based on the following criteria:

- Quickly determine which process gets the next turn.
- Processes get a fair share of CPU time, but permit high priority processes a larger share and to preempt lower-priority processes.
- Responsive to benefit interactive applications.
- Predictable and scalable under various workload conditions.

Describing Process Priority

In Linux, a scheduler controls the order of execution based on the scheduling policy and scheduling priority assigned to each thread or process. These scheduling policies are divided into two groups: non-real-time policies and real-time policies.

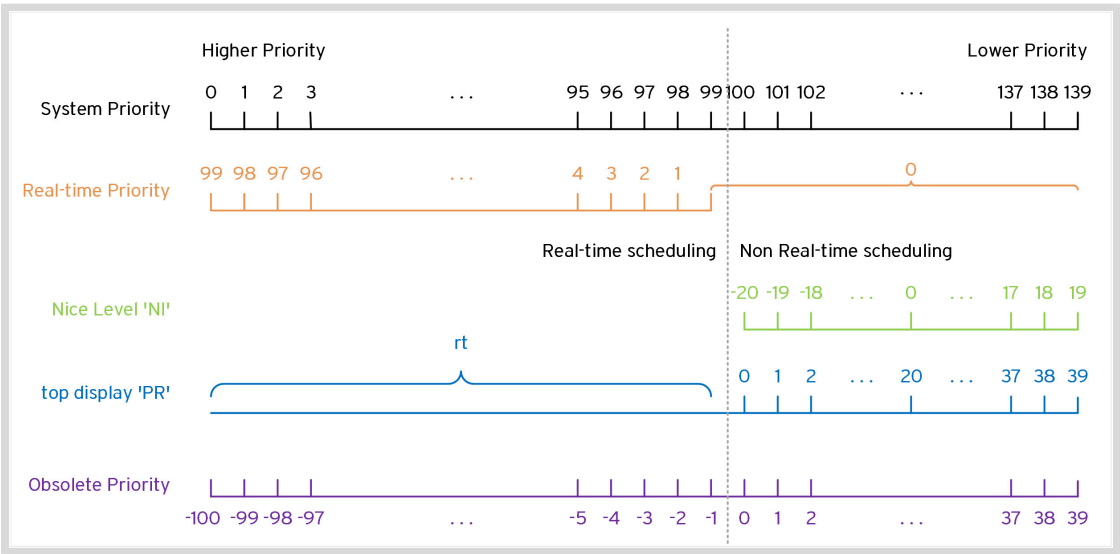


Figure 7.1: Process priorities

Static priority

A *static priority* is a default priority set at new process creation and cannot be changed. The process priority using real-time policies is a value between **1** (lowest) and **99** (highest), and is known as a *real-time priority*. The scheduler maintains a list of runnable processes in a self-balancing binary search tree. To determine which processes gets the CPU time next, the

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

scheduler looks for the process with highest static priority or real-time priority. The scheduling policy determines the ordering only within the run queue for processes with an equal static priority. Processes using non-real-time policies do not use static priorities and their static priority is set to **0**.

Dynamic priority

The process priority is reset when it is CPU starved because other processes with higher priority are being scheduled first. In this scenario, the scheduler raises the priority of the process that has the longest CPU wait time. A user can also modify a process priority using the **nice** or **renice** commands. The process priority can increase and decrease during the process lifetime, and is known as a *dynamic priority*. The value to determine which non-real-time process preempts other non-real-time processes is called a nice value. Since the static priority of non-real-time processes are set to **0**, the nice priority value determines the relative scheduling of non-real-time processes. The weight of a process controls how much CPU a process gets, and the nice value scales the process weight. For example, under system CPU saturation, a process with nice value of **10** would get twice the CPU time of a process with nice value of **5**. When the system is not CPU saturated loaded, every process gets a fair share of the CPU time. The nice value ranges from **-20** (highest) to **19** (lowest).

Use the **ps** command to list the priorities for processes; the **pri** option lists the static priority, the **rtprio** option lists real-time priority, the **ni** option lists the nice value, and the **cls** option lists the scheduling policy where **TS** represents the time-sharing, non-real-time policy, and **FF** represents the **FIFO** real-time policy:

```
[root@host ~]# ps axo pid,pri,rtprio,ni,cls,comm
  PID PRI RTPRIO  NI CLS COMMAND
  ...output omitted...
   36 139     99   -  FF watchdogd
   37  19      -   0  TS kworker/1:1-cgroup_pidlist_destroy
   39  19      -   0  TS kswapd0
   58  19      -   0  TS kworker/u4:1-events_unbound
   90  39      - -20  TS kthrotld
   91  90     50   -  FF irq/24-pciehp
   92  90     50   -  FF irq/25-pciehp
   93  90     50   -  FF irq/26-pciehp
   94  90     50   -  FF irq/27-pciehp
   95  90     50   -  FF irq/28-pciehp
   96  90     50   -  FF irq/29-pciehp
   97  90     50   -  FF irq/30-pciehp
   98  90     50   -  FF irq/31-pciehp
  ...output omitted...
```

Important

In the output of the **ps axl** command, the **PRI** column shows different priority for the same process than is shown by the **ps axo pid,pri,comm** command. The **ps axl** command shows the obsolete priority from older UNIX variants, which ranges from **-40** (lowest priority) to **99** (highest priority). The **ps axo pid,pri,comm** command shows the priority in the new style, which ranges from **0** (highest priority)to **139** (lowest priority).

Only one process can run at a specific time on any given CPU. Therefore, processes are typically stopped and started several times before completing their execution. By default, the kernel allocates a quantum of processing time to processes known as *timeslice*. A process divides its

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

timeslice into several parts, rather than running all of it at one time. When the timeslice of a process is exhausted, or the process is waiting for an event, the process can voluntarily yield the CPU time to another process. When a process involuntarily yields the CPU time to a higher-priority process, it is called *preemption*. Preemption hurts the overall performance of an application.

Use the **ps** command to list the timeslice for processes; use the **cputime** option to list the CPU time. In the below output, one process is executed with a real-time policy (**FF**), and other with a non-real-time policy (**TS**). Notice that the real-time process gets more CPU time than the non-real-time process:

```
[root@host ~]# ps axo pid,pri,rtprio,ni,cls,cputime,comm 1238 1248
  PID PRI RTPRIO  NI CLS     TIME COMMAND
 1238  78     38   -  FF 00:01:05 sha1sum
 1248  19      -   0  TS 00:00:09 sha1sum
```

When switching between processes, the kernel stores information for both the previous process, and the next process. Each time a process yields the processor, the kernel stores information on its current operating state such that when it is again scheduled to run on the processor, it can resume its operation from an identical position. This operational state data is known as context, which consists of the contents of the CPU registers and the program counter. The switching of processes for CPU time is known as *context switching*.

To view the voluntary and involuntary context switches of a process, view the contents of the **/proc/PID/status**, where PID is the process ID of the process. The following command lists the context switches of a process, where **4350** is the process ID:

```
[root@host~ ]# grep voluntary /proc/4350/status
voluntary_ctxt_switches: 1200
nonvoluntary_ctxt_switches: 1950
```

Changing the priority of a process helps to reduce involuntary context switches.

Defining Scheduling Policies

Red Hat Enterprise Linux provides six scheduling policies divided into two groups: real-time and non-real-time scheduling classes.

Available policies in each scheduling class

Class	Kernel source code	Policies
Real-Time Scheduler	**linux/kernel/sched/rt.c**	**SCHED_FIFO** **SCHED_RR**
Completely Fair Scheduler (CFS)	**linux/kernel/sched/fair.c**	**SCHED_NORMAL** (also known as **SCHED_OTHER**) **SCHED_BATCH** **SCHED_IDLE**
Deadline Scheduler	**linux/kernel/sched/deadline.c**	**SCHED_DEADLINE**

Use the **ps o pid,cls,comm** command to list the scheduling policy associated with a running process. The **policy** and **class** options are aliases of the **cls** option, and list the scheduling class of the process.

```
[root@host ~]# ps o pid,cls,rtprio,comm $(pidof httpd)
  PID CLS RTPRIO COMMAND
 2205  RR     50 httpd
 2206  RR     50 httpd
 2207  RR     50 httpd
 2208  RR     50 httpd
 2209  RR     50 httpd
```

Real-time Scheduling Policies

SCHED_FIFO
The policy uses the first-in, first-out scheduling algorithm without timeslices. It runs until it is blocked by I/O, calls **sched_yield**, or is preempted by a higher-priority process. The **ps o cls** command output displays the process using the **SCHED_FIFO** policy as **FF**.

SCHED_RR
The policy uses the round-robin scheduling algorithm with timeslices. Each process can run until it exhausts a predetermined timeslice. Processes at the same priority level, using the round-robin scheduling policy, are only allowed to run for a maximum time quantum. The **ps o cls** command output displays the process using the **SCHED_RR** policy as **RR**.

Both real-time scheduling policies implement fixed priority. The kernel does not calculate dynamic priority values for real-time tasks. The fixed real-time priority ensures that a real-time process at a given priority always preempts a process at a lower priority. With real-time scheduling policies, the kernel tries to schedule processes within timing deadlines, but the kernel does not promise always to achieve these goals.

Non-Real-time Scheduling Policies

SCHED_NORMAL (also known as SCHED_OTHER)
The policy defines the round-robin style time-sharing schedule. The policy is a *default* time-sharing schedule used by the majority of processes on Linux. The **ps o cls** command output displays the process using the **SCHED_NORMAL** policy as **TS**.

SCHED_BATCH
The policy benefits batch-oriented workloads. It does not preempt nearly as often as **SCHED_NORMAL** does, so the tasks run longer and make better use of caches. The **ps o cls** command output displays the process using the **SCHED_BATCH** policy as **B**.

SCHED_IDLE
The policy benefits running low priority applications. The priority of processes running with **SCHED_IDLE** is lower than a process running with a nice value of **19**. The **ps o cls** command output displays the process using the **SCHED_IDLE** policy as **IDLE**.

Applications must use the default scheduling policy, unless a specific application requires the use of a fixed-priority scheduling policy. The overall performance of the system reduces when using a high-priority process that runs using **SCHED_FIFO**. For example, processes doing intensive calculations, such as inverting a large matrix, must never run with **SCHED_FIFO**. Most of the kernel threads use **SCHED_FIFO** to avoid preemption by user-level applications. For example, the **migration** kernel process, which moves processes between CPUs for load-balancing, uses **SCHED_FIFO**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Completely Fair Scheduler

Older preemptive schedulers used fixed timeslices and multiple scheduling queues, one per process priority. Fixed timeslice schedulers pose a risk when an unwanted real-time process starts execution, because preemption by another higher-priority process is the only way to schedule out a real-time process. The *Completely Fair Scheduler* (CFS) was introduced in Linux kernek 2.6.23 and is the default RHEL scheduler. With CFS, the compute time for a running process is calculated dynamically as the scheduling context changes over the system's lifetime. CFS usea the **`SCHED_NORMAL`** scheduling policy, the default for all user-space applications.

INstead of using run queues, CFS guarantees a fair approach by maintaining runnable processes in a time-ordered, self-balancing, binary search **`red-black tree`**. To learn about a **`red-black`** self-balancing binary search tree, refer to *Red-black tree* at https://en.wikipedia.org/wiki/Red-black_tree. The three main features of the CFS scheduler are:

Scheduling Classes
: Scheduling classes extends CFS functionality using scheduler modules. The scheduler core handles these modules, which contain scheduling policy details.

Fair Scheduling
: Maintain CPU time sharing fairness for processes, based on how many processes currently contend for CPU time. CFS tracks the amount of time each task has already consumed, known as its *virtual runtime*.

Group Scheduling
: The CFS scheduler can provide fair CPU time to defined process groups.

Selecting Processes in CFS

CFS uses a self-balancing binary search tree to manage runnable processes and determine the process with the longest CPU waiting time. The process with the highest virtual time, the longest time waiting, gets to use the CPU. As that process uses CPU cycles, its virtual time decreases. When CFS next checks, a process with a higher virtual time preempts the running process. CFS kernel code is simple, scales and behaves well, and it is harder for "pathological" user processes to affect the interactive system behavior.

Kernel Tunables for Scheduling Policies

The **`sysctl`** command can tune CFS scheduler tunables. These tunables are located in the **`/proc/sys/kernel/`** directory Some are listed here:

`sched_latency_ns`
: Defines the targeted preemption latency. Increasing this variable value increases the timeslice of a CPU bound process. The targeted latency is the period in which all run queue tasks are scheduled at least once. The duration is defined in nanoseconds (ns).

`sched_min_granularity_ns`
: Defines the minimum amount of time any scheduled process must be allowed to run before being preempted by another process. The duration is defined in nanoseconds (ns). This variable value must be smaller than **`sched_latency_ns`**. If the number of running tasks in the queue is greater than **`sched_latency_ns`** divided by **`sched_min_granularity_ns`**, then there are too many tasks on the system, and the **`sched_latency_ns`** variable cannot satisfy all the minimum granularity time requirements for a task. To accommodate such cases, the scheduler increases the scheduling period by multiplying the number of running of tasks by the **`sched_min_granularity_ns`** value.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

`sched_migration_cost_ns`
Defines the require elapsed time after a process' last execution to be considered for migration to another CPU. The duration is defined in nanoseconds (ns). Increasing this variable value reduces process migration across CPUs.

`sched_rt_period_us`
Defines the timeslice that provides runtime to CPU bound processes. The duration is defined in microseconds (µs). The default value of is equivalent to 100% CPU bandwidth.

`sched_rt_runtime_us`
Defines maximum CPU time that can be used by all real-time tasks. The duration is defined in microseconds (µs). The scheduler reserves the difference between the **`sched_rt_period_us`** and **`sched_rt_runtime_us`** variable value for processes using non-real-time scheduling policies. The default value for **`sched_rt_period_us`** is **`1000000`** µs (**`1`** second) and for **`sched_rt_runtime_us`** is **`950000`** µs (**`0.95`** second). This provides **`0.05`** second to be consumed by processes using non-real-time policies. These default values prevent a real-time process from locking up the system. This safeguard mechanism is known as the *real-time scheduler throttling*.

> **Important**
> Setting **`sched_rt_runtime_us`** to **`-1`** disables time reservation for processes. Setting **`sched_rt_runtime_us`** to **`0`** disallows allocating time periods to real-time processes.

`sched_rr_timeslice_us`
Defines the timeslice that a **`SCHED_RR`** process is allowed to run before the process is preempted and put at the end of the CPU bound process queue. The duration is defined in microseconds (µs).

Video Transcoding Example Using Real-time Runtime

View the *Figure 7.2* graphic with this example.

In a video transcoding process, the CPU must deliver 25 frames per second, which is one frame every 0.04 seconds. The system performing the transcoding must also handle other **`SCHED_NORMAL`** processes, so the engineer wants to limit the video transcoding to only 80% of the CPU time. As calculated below, the **`sched_rt_runtime_us`** tunable is set to 0.032 seconds.

```
80% (utilization) * 0.040s (period allowed for one frame) = 0.032s (32000µs)
```

In the folowing graphic, **`sched_rt_period_us`** is set to 0.040 seconds (40000 µs), and **`sched_rt_runtime_us`** is set to 0.032 seconds (32000 µs) for the video transcoding process.

Videos also have audio, with a memory buffer needing to be regularly refilled so as to not empty during a transcoding runtime. The engineer has determined that refilling every 5ms is sufficient. To refill the audio buffer requires minimal CPU, which the engineer limits to 3% CPU time. As calculated below, the **`sched_rt_runtime_us`** tunable is set to 0.00015s seconds.

```
3% (utilization) * 0.005s (period allowed for one refill) = 0.00015s (150µs)
```

In the folowing graphic, **`sched_rt_period_us`** is set to 0.005 seconds (5000 µs), and **`sched_rt_runtime_us`** is set to 0.00015 seconds (150 µs) for audio buffer refilling.

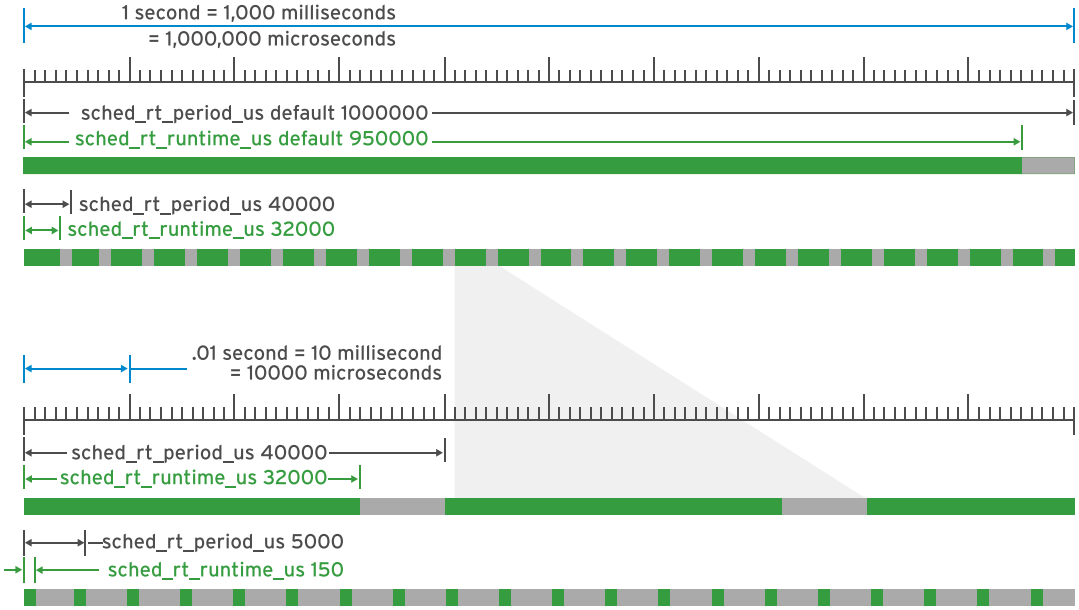


Figure 7.2: Real-time runtime scheduling

Introducing Deadline Scheduler

Real-time systems must respond to requests within a deadline. If the system fails to respond by the deadline, the system is considered to have failed in delivering results. RHEL 8 adds support for a new real-time scheduler named *deadline scheduler* or **`SCHED_DEADLINE`**. The scheduler enables specifying the task requirement parameters, allowing the system to make better scheduling decisions. The scheduler guarantees real-time task scheduling, even under high load conditions. The real-time scheduler, which provides **`SCHED_FIFO`** and **`SCHED_RR`** scheduling policies, schedules tasks based on fixed priorities. In contrast, the deadline scheduler schedules tasks according to a process deadline, and schedules the process with the earliest deadline first.

The deadline scheduler uses the three parameters: period, deadline, and runtime to define a task in nanoseconds. In the below figure, the period is the activation pattern of a real-time task. Consider an example of a video encoding task that processes 60 frames per second, and delivers a new frame at every 16 milliseconds, so the period is this case is 16 milliseconds. The runtime is the amount of CPU time that the application needs to produce the output. In the above example, if the video encoding tool consumes five milliseconds to process the image, then the runtime is 5 milliseconds. The deadline is the maximum amount of time in which the result must be delivered by the task, relative to the period. In the above example, the task must deliver a processed frame is 10 milliseconds, then the deadline is 10 milliseconds.

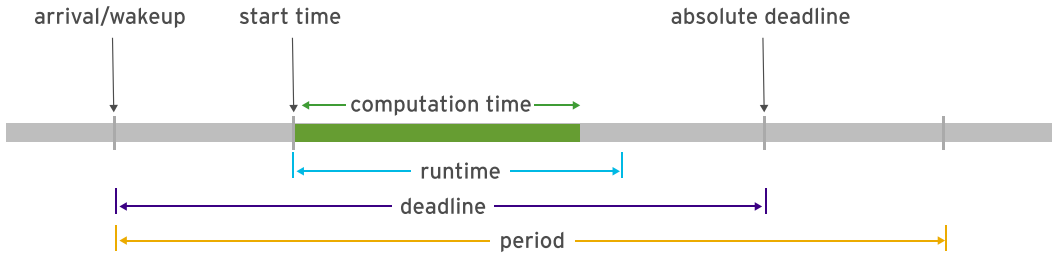


Figure 7.3: Deadline scheduler parameters

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

In the below output, the task uses the **SCHED_DEADLINE** policy to guarantee 5 ms of CPU time every 16.6 ms, and all CPU time will be available for the task within 10 ms.

```
[root@host ~]# chrt -d --sched-runtime 5000000 --sched-deadline 10000000 \
 --sched-period 16666666 0 simple_task
```

The **chrt** command with the **--sched-runtime**, **--sched-deadline** and the **--sched-period** options defines the runtime, the relative deadline, and the period in nanoseconds. The **-d** option schedules the task using the **SCHED_DEADLINE** scheduling policy. The **ps o cls** command output displays the process using the **SCHED_DEADLINE** policy as **DLN**.

When using the real-time scheduler, the user must define the correct fixed priority of the task to avoid involuntary preemption. Thus, in a system with a lot of running processes, the usage of the real-time scheduler can be more complicated than using the deadline scheduler.

The main advantages of the deadline scheduler are that the user needs to pass the parameter to meet the task deadline, and does not need to analyze other tasks running on the system. Context switches are fewer with deadline scheduler. On uni-processor systems, the deadline scheduling can schedule more tasks than fixed priority scheduling, while meeting all tasks deadlines.

Setting Scheduling Options for Processes

Linux provides a family of system calls for the management of scheduler parameters, which allow manipulation of process priority, scheduling policy, and processor affinity, yielding the processor to other tasks. An application programmatically uses the **sched_setscheduler()**, and **sched_getscheduler()** system calls to set and get a given scheduling policy and real-time priority for a process. The **sched_setscheduler**(2) manual page provides information about the **sched_setscheduler()** and **sched_getscheduler()** system calls.

Red Hat Enterprise Linux provides administrators two different ways to configure and observe process scheduler configurations: the **chrt** command, and the *tuna* interface.

Changing Scheduler Options Using Command Line

A system administrator uses the **chrt** command to view the scheduling policy and priority, and also to execute an application with the selected scheduling policy. To view the attributes of a particular process, use the following command:

```
[root@host ~]# chrt -p 5467
pid 5476's current scheduling policy: SCHED_OTHER
pid 5476's current scheduling priority: 0
```

The **chrt** command starts a program with a specified scheduler and a given priority. The command also changes the scheduler, and optionally the priority, on an already running process.

When starting a new program with **chrt**, the system administrator needs to provide the desired scheduler; otherwise the command uses the **SCHED_RR** scheduling policy as default. Additionally, the **chrt** command uses real-time priority as one of the arguments to execute the command:

```
[root@host ~]# chrt -f 38 /bin/simple_task
[root@host ~]# chrt -p 5890
pid 5890's current scheduling policy: SCHED_FIFO
pid 5890's current scheduling priority: 38
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Options for setting scheduler policy using **chrt** are:

-b sets the scheduling policy to use **SCHED_BATCH**
-f sets the scheduling policy to use **SCHED_FIFO**
-i sets the scheduling policy to use **SCHED_IDLE**
-o sets the scheduling policy to use **SCHED_NORMAL** or **SCHED_OTHER**
-r sets the scheduling policy to use **SCHED_RR**
-d sets the scheduling policy to use **SCHED_DEADLINE**

Use the **chrt -p** command to change the policy and the priority of a running process.

```
[root@host ~]# chrt -o -p 5890
[root@host ~]# chrt -p 5890
pid 5890's current scheduling policy: SCHED_OTHER
pid 5890's current scheduling priority: 0
```

Changing Scheduling Options Using Tuna

The *tuna* package provides a command-line utility and a GUI application to adjust the scheduler options such as scheduler policy, real-time priority, and CPU affinity. Use the **tuna --show_threads** command to view the current policies and priorities for all threads on a system:

```
[root@host ~]# tuna --show_threads
                     thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  1      OTHER     0      0,1      1415         1048         systemd
  2      OTHER     0      0,1       153            0        kthreadd
  3      OTHER     0      0,1         2            1          rcu_gp
  4      OTHER     0      0,1         2            1      rcu_par_gp
  5      OTHER     0        0       244            0 kworker/0:0-events
```

To view the information for a particular thread, use the following command:

```
[root@host ~]# tuna --threads 91 --show_threads
                      thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  91      FIFO    50        0         2            0   irq/24-pciehp
```

The **-t** and **-P** options are the short-form notation for the **--threads** and **--show_threads** options respectively.

Use the **--priority** option with the **tuna** command to change policy and real-time priority information for a particular thread:

```
[root@host ~]# tuna --threads 7861 --priority=RR:40
```

The policy can be either **RR**, **FIFO**, or **OTHER**.

Changing Scheduler Options for Services Using Systemd

A system administrator may change the scheduling policy and priority of a **systemd** managed service launched during the system boot. The unit configuration directives are used to change

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

the priority of a service during the boot process. Use the following directives in the **[Service]** section to change the scheduling policy and priority:

Nice
Sets the default nice level for the service. The nice level is set to a number between **-20** (highest priority) to **19** (lowest priority).

CPUSchedulingPolicy
Sets the CPU scheduling policy for the service. The policy is set to one of these values:**other**, **batch**, **idle**, **fifo** and **rr**.

> **Important**
> The **CPUSchedulingPolicy** variable currently does not support setting the **SCHED_DEADLINE** policy. A request for enhancement was submitted; refer at https://github.com/systemd/systemd/issues/10034.

CPUSchedulingPriority
Sets the CPU scheduling priority for the service. The priority range is dependent on the selected scheduling policy. For real-time scheduling policies, the value is set to numbers between **1** (lowest priority) and **99** (highest priority).

Change the **sshd** service scheduler policy to the round-robin, real-time scheduler with priority **10**, and then create the **/etc/systemd/system/sshd.service.d/10-scheduler.conf systemd** drop-in file with the following content:

```
[root@host ~]# cat /etc/systemd/system/sshd.service.d/10-scheduler.conf
[Service]
CPUSchedulingPolicy=rr
CPUSchedulingPriority=10
```

After adding the **systemd** drop-in file, reload **systemd** to read the new configuration files.

```
[root@host ~]# systemctl daemon-reload
```

After that, the service has to be started or restarted, respectively, for the changes to take effect.

> **Note**
> The **Nice**, **CPUSchedulingPolicy** and **CPUSchedulingPriority** directives can not be changed dynamically with **systemctl set-property**.

Viewing Process Scheduler Statistics

The **/proc** directory provides scheduler statistics information. The below list includes some of the **/proc** files that provide statistical information:

/proc/sched_debug
Contains the current values of all kernel tunables that affect the task scheduler behavior, statistics, and information about the run queues related to various scheduling policies on all available processors.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# cat /proc/sched_debug
Sched Debug Version: v0.11, 4.18.0-80.el8.x86_64 #1
ktime                                   : 4868733.883032
sched_clk                               : 4864409.396498
cpu_clk                                 : 4863903.145683
jiffies                                 : 4299536030
sched_clock_stable()                    : 1

sysctl_sched
  .sysctl_sched_latency                    : 12.000000
  .sysctl_sched_min_granularity            : 10.000000
  .sysctl_sched_wakeup_granularity         : 15.000000
  .sysctl_sched_child_runs_first           : 0
  .sysctl_sched_features                   : 4118331
  .sysctl_sched_tunable_scaling            : 1 (logaritmic)
...output omitted...
rt_rq[0]:/
  .rt_nr_running                 : 0
  .rt_nr_migratory               : 0
  .rt_throttled                  : 0
  .rt_time                       : 0.000808
  .rt_runtime                    : 950.000000

dl_rq[0]:
  .dl_nr_running                 : 0
  .dl_nr_migratory               : 0
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 0

runnable tasks:
 S           task   PID         tree-key  switches  prio     wait-time
 sum-exec        sum-sleep
-------------------------------------------------------------------------------------------------------------
 S       kthreadd     2     42428.774826       168   120         0.000000
 2.128856         0.000000 0 0 /
 I         rcu_gp     3        15.983287         3   100         0.000000
 0.004644         0.000000 0 0 /
...output omitted...
```

/proc/schedstat

Displays statistics relevant to the run queue. Use the **/usr/share/doc/kernel-doc-*/Documentation/scheduler/sched-stats.txt** file for more information.

```
[root@host ~]# cat /proc/sched_debug
version 15
timestamp 4299791167
cpu0 0 0 0 0 0 0 12163699489 5757880024 129647
domain0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
cpu1 0 0 0 0 0 0 18567509943 7081565709 86636
domain0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

/proc/PID/sched

Displays scheduling information for the process, where PID is the process ID.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# cat /proc/91/sched
irq/24-pciehp (91, #threads: 1)
-------------------------------------------------------------------
se.exec_start                                :           907.835303
se.vruntime                                  :             0.000000
se.sum_exec_runtime                          :             0.030325
se.nr_migrations                             :                    0
nr_switches                                  :                    2
nr_voluntary_switches                        :                    2
nr_involuntary_switches                      :                    0
se.load.weight                               :              1048576
se.runnable_weight                           :              1048576
se.avg.load_sum                              :                47481
se.avg.runnable_load_sum                     :                47481
se.avg.util_sum                              :             20621504
se.avg.load_avg                              :                 1024
se.avg.runnable_load_avg                     :                 1024
se.avg.util_avg                              :                  434
se.avg.last_update_time                      :            907806720
se.avg.util_est.ewma                         :                  108
se.avg.util_est.enqueued                     :                  435
policy                                       :                    1
prio                                         :                   49
clock-delta                                  :                   17
numa_pages_migrated                          :                    0
numa_preferred_nid                           :                   -1
total_numa_faults                            :                    0
current_node=0, numa_group_id=0
numa_faults node=0 task_private=0 task_shared=0 group_private=0 group_shared=0
```

References

sched(7), **sched_setscheduler**(2), **chrt**(1), **systemd.exec**(5), and **tuna**(8) man pages.

The description of scheduling algorithms and implementation details for the schedulers are documented in the **/usr/share/doc/kernel-doc-*/Documentation/scheduler/** directory.

For more information on tuning scheduler options, refer to the *Realtime-Specific Tuning* chapter in the *Red Hat Enterprise Linux for Real Time 8 Tuning Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/8/html/tuning_guide/chap-realtime-specific_tuning

For more information on deadline scheduler, refer to the *Deadline scheduling part 1 – overview and theory* at https://lwn.net/Articles/743740/

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring CPU Scheduling Policies and Tunables

In this exercise, you will use **systemd** and **chrt** to set real-time priorities for services and processes.

Outcome(s)

You should be able to:

- Configure the **rngd** service to run using batch scheduling policy.
- Execute the **simple_loop_task** task using the deadline scheduler and observe the utilization.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab cputuning-policies start** command. This command determines if the host, **servera**, is reachable on the network. The command also copies resource files required during the exercise to **/root** on **servera**

```
[student@workstation ~]$ lab cputuning-policies start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Create a **systemd** drop-in directory for the **rngd** service. Create a drop-in file called **20-batch.conf** with the following two settings:

   - **CPUSchedulingPolicy=batch**
   - **CPUSchedulingPriority=0**

   2.1. Create a drop-in directory for the **rngd** service.

```
[root@servera ~]# mkdir /etc/systemd/system/rngd.service.d
```

   2.2. Create the **/etc/systemd/system/rngd.service.d/20-batch.conf** file with the following contents:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# vim /etc/systemd/system/rngd.service.d/20-batch.conf
[Service]
CPUSchedulingPolicy=batch
CPUSchedulingPriority=0
```

3. Reload the **systemd** service and restart the **rngd** service.

   3.1. Reload **systemd** to reread the configuration changes.

```
[root@servera ~]# systemctl daemon-reload
```

   3.2. Restart the **rngd** service.

```
[root@servera ~]# systemctl restart rngd
```

4. Use the **ps** command to verify that the **rngd** process is now running using the batch scheduler and a priority of **0**.

```
[root@servera ~]# ps o pid,cls,rtprio,comm $(pidof rngd)
  PID CLS RTPRIO COMMAND
 4489  B     0 rngd
```

5. Use the **chrt** command to execute the real-time process using the **SCHED_DEADLINE** policy. View the current configuration and utilization of the process.

   5.1. Use the **chrt** command with the **--sched-runtime**, **--sched-deadline**, and **--sched-period** options to execute the **simple_loop_task** task using **SCHED_DEADLINE** policy. The task must guarantee 5 ms of CPU time every 16.6 ms, and the task must execute within the deadline of 10 ms.

```
[root@servera ~]# chrt -d --sched-runtime 5000000 --sched-deadline 10000000 \
--sched-period 16666666 0 ./simple_loop_task &
[1] 4508
```

   Note the process ID.

   5.2. Use the **chrt -p** command to view the configured scheduler parameters for the deadline process. Use the process ID from the previous step to execute the command.

```
[root@servera ~]# chrt -p 4508
pid 4508's current scheduling policy: SCHED_DEADLINE
pid 4508's current scheduling priority: 0
pid 4508's current runtime/deadline/period parameters: 5000000/10000000/16666666
```

   5.3. View the current utilization of the deadline process from the **/proc/*PID*/sched** file, where *PID* is the process ID.

   In the output, **dl.runtime** and **dl.deadline** show the current runtime and deadline parameter for the deadline process.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# cat /proc/4508/sched
simple_loop_tas (4508, #threads: 1)
-------------------------------------------------------------------
se.exec_start                                :       2121799.313493
se.vruntime                                  :             0.000000
se.sum_exec_runtime                          :           215.309522
se.nr_migrations                             :                    1
nr_switches                                  :                28583
nr_voluntary_switches                        :                    2
nr_involuntary_switches                      :                28581
se.load.weight                               :              1048576
se.runnable_weight                           :              1048576
se.avg.load_sum                              :                46155
se.avg.runnable_load_sum                     :                46155
se.avg.util_sum                              :             24123113
se.avg.load_avg                              :                 1001
se.avg.runnable_load_avg                     :                 1001
se.avg.util_avg                              :                  508
se.avg.last_update_time                      :        1646045637632
se.avg.util_est.ewma                         :                  221
se.avg.util_est.enqueued                     :                  509
policy                                       :                    6
prio                                         :                   -1
dl.runtime                                   :              4992804
dl.deadline                                  :        2123887383891
clock-delta                                  :                   25
mm->numa_scan_seq                            :                    0
numa_pages_migrated                          :                    0
numa_preferred_nid                           :                   -1
total_numa_faults                            :                    0
current_node=0, numa_group_id=0
numa_faults node=0 task_private=0 task_shared=0 group_private=0 group_shared=0
```

5.4. Open the **/root/calc_deadline_util.sh** file to view the calculation of:

- Configured utilization percentage using the **runtime** and **period** variables listed by the **chrt -p** command

```
...output omitted...
a=$(chrt -p $pid | awk '$(NF-1) == "parameters:" { print $NF }')
runtime=$(echo $a | awk '{split($0,a,"/"); print a[1]}')
deadline=$(echo $a | awk '{split($0,a,"/"); print a[2]}')
period=$(echo $a | awk '{split($0,a,"/"); print a[3]}')
util_setting=$(awk "BEGIN { printf \"%.8f\n\", ($runtime / $period) * 100 }")
total_util_setting=$(awk "BEGIN { printf \"%.8f\n\", $total_util_setting +
 $util_setting }")
echo "Configured utilisation of " $pid ": runtime =" $runtime, "deadline ="
 $deadline, "period =" $period, "util=" $util_setting"%"
...output omitted...
```

- Actual utilization percentage using the **dl.runtime** and **dl.deadline** variables from the **cat /proc/*PID*/sched** file.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
...output omitted...
for file in /proc/*/sched;
do
  ...output omitted...
  runtime=$(awk '$1 == "dl.runtime" {print $3}' $file)
  deadline=$(awk '$1 == "dl.deadline" {print $3}' $file)
  actual_util=$(awk "BEGIN { printf \"%.8f\n\", ($runtime / $deadline) * 100 }")
  actual_total_util=$(awk "BEGIN { printf \"%.8f\n\", $actual_total_util +
 $actual_util }")
  echo "Current utilisation of " $pid "   : runtime =" $runtime, "deadline ="
 $deadline, "util = "$actual_util"%"
...output omitted...
```

5.5. Execute the **`calc_deadline_util.sh`** file to calculate the configured utilization and the actual utilization of the deadline process.

```
[root@servera ~]# ./calc_deadline_util.sh
------- PID 4508 -------
Current utilisation of  4508    : runtime = 4998401, deadline = 2428870705025,
 util = 0.00020579%
Configured utilisation of  4508 : runtime = 5000000, deadline = 10000000, period =
 16666666, util= 30.00000120%
------------ TOTAL -------------
Current utilisation    : 0.00020579%
Configured utilisation : 30.00000120%
```

6. Execute more tasks using the deadline scheduler, observe that the deadline scheduler denies the addition of more tasks. The deadline scheduler guarantees that all deadline tasks must have CPU time to execute the task.

6.1. View the number of tasks configured to launch using the **`chrt`** command in the **`/root/run_deadline.sh`** script.

```
[root@servera ~]# cat run_deadline.sh | grep chrt | wc -l
6
```

6.2. Execute the **`run_deadline.sh`** script to launch six more tasks using the deadline scheduler. Observe that the deadline scheduler denies the addition of one task out of every six tasks.

Notice the error thrown on executing the sixth task because the deadline scheduler can not guarantee CPU time to execute the sixth task in the **`run_deadline.sh`** script.

```
[root@servera ~]# ./run_deadline.sh
[root@servera ~]# chrt: failed to set pid 0's policy: Device or resource busy
```

6.3. Recalculate the configured utilization and the actual utilization for all the deadline tasks. Observe that only *five* tasks execute from the **`run_deadline.sh`** script.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# ./calc_deadline_util.sh
------- PID 4508 -------
Current utilisation of  4508    : runtime = 4998305, deadline = 2798370690245,
 util = 0.00017861%
Configured utilisation of  4508 : runtime = 5000000, deadline = 10000000, period =
 16666666, util= 30.00000120%
------- PID 6390 -------
Current utilisation of  6390    : runtime = 997895, deadline = 2798416002044, util
 = 0.00003566%
Configured utilisation of  6390 : runtime = 1000000, deadline = 5000000, period =
 5000000, util= 20.00000000%
------- PID 6392 -------
Current utilisation of  6392    : runtime = 2997881, deadline = 2798432234462,
 util = 0.00010713%
Configured utilisation of  6392 : runtime = 3000000, deadline = 10000000, period =
 10000000, util= 30.00000000%
------- PID 6393 -------
Current utilisation of  6393    : runtime = 3997410, deadline = 2798447647659,
 util = 0.00014284%
Configured utilisation of  6393 : runtime = 4000000, deadline = 13000000, period =
 13000000, util= 30.76923077%
------- PID 6394 -------
Current utilisation of  6394    : runtime = 4997368, deadline = 2798463036249,
 util = 0.00017858%
Configured utilisation of  6394 : runtime = 5000000, deadline = 15000000, period =
 15000000, util= 33.33333333%
------- PID 6395 -------
Current utilisation of  6395    : runtime = 5997071, deadline = 2798482548006,
 util = 0.00021430%
Configured utilisation of  6395 : runtime = 6000000, deadline = 20000000, period =
 20000000, util= 30.00000000%
------------ TOTAL -------------
Current utilisation    : 0.00085712%
Configured utilisation : 174.10256530%
```

6.4. Log out from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab cputuning-policies finish** command to complete this exercise.

```
[student@workstation ~]$ lab cputuning-policies finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring CPU Affinity and Isolation

Objectives

After completing this section, you should be able to configure CPU and interrupt affinity to control application isolation and CPU resource commitments.

Pinning Processes

In typical operation, the kernel determines the CPUs a process runs. Every time the scheduler reschedules a process, it can go to any of the available CPUs. While this is fine for most workloads, sometimes it is desirable to limit on which CPU(s) a process is allowed to run. For example, limiting a memory-intensive process to just one or two CPUs increases the chances of a cache hit, thus increasing overall performance.

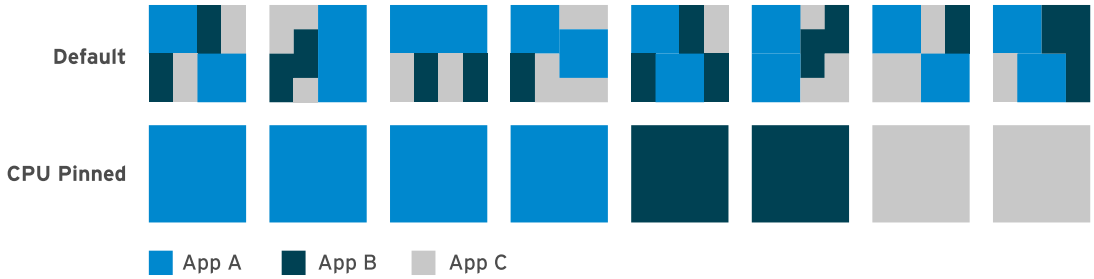


Figure 7.4: CPU Affinity Can Increase Cache and Memory Effectiveness

The **`systemd`** based services offer a convenient way of limiting the CPUs available to a service using the **`CPUAffinity`** setting inside a **`[Service]`** section of a unit definition. This setting takes a space-separated list of CPU indexes on which the service is allowed to run, with the first CPU being index **`0`**, and the second index **`1`**. When the service unit definition defines more than one **`CPUAffinity`** parameter, their values are merged.

Setting **`CPUAffinity`** to an empty string resets the service to use all CPUs:

```
[Service]
CPUAffinity=""
```

Managing CPU Affinity with Cgroups

On a system with NUMA architecture, limiting both the available CPUs and memory nodes ensures that a process always has the fastest access to memory. By default, the system automatically manages memory and CPU placement of services, but you can manually organize placement. The screen below shows a system with four NUMA nodes.

```
[root@host ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                192
```

```
On-line CPU(s) list:   0-191
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             4
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E7-8890 v4 @ 2.20GHz
Stepping:              1
CPU MHz:               2959.069
CPU max MHz:           3400.0000
CPU min MHz:           1200.0000
BogoMIPS:              4389.61
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              61440K
NUMA node0 CPU(s):     0-23,96-119
NUMA node1 CPU(s):     24-47,120-143
NUMA node2 CPU(s):     48-71,144-167
NUMA node3 CPU(s):     72-95,168-191
...output omitted...
```

With larger systems, pinning a process to use a specific CPU and memory belonging to the same NUMA node reduces memory access times and CPU contention. An example use case is a NUMA system running a sizable high-performance cluster with a high performance application. A scalable way of doing CPU and memory placement is by using the **cpuset** cgroup.

The **cpuset** cgroup controller assigns individual CPUs on a multicore system and memory nodes to tasks in a control group. The **cpuset** directory is present under the **/sys/fs/cgroup** virtual file system. The files available under the **/sys/fs/cgroup/cpuset** directory:

```
[root@host ~]# ls /sys/fs/cgroup/cpuset
cgroup.clone_children  cpuset.memory_pressure
cgroup.procs           cpuset.memory_pressure_enabled
cgroup.sane_behavior   cpuset.memory_spread_page
cpuset.cpu_exclusive   cpuset.memory_spread_slab
cpuset.cpus            cpuset.mems
cpuset.effective_cpus  cpuset.sched_load_balance
cpuset.effective_mems  cpuset.sched_relax_domain_level
cpuset.mem_exclusive   notify_on_release
cpuset.mem_hardwall    release_agent
cpuset.memory_migrate  tasks
```

Each running process has a **cpuset** file in **/proc** that lists the associated directory in the cgroup hierarchy. The default is **/**, the top-level directory in a hierarchy.

```
[root@host ~]# pgrep syslog
1088
[root@host ~]# cat /proc/1088/cpuset
/
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Administrators can manually read and write to the control files in the **/sys/fs/cgroup** cgroup virtual file system. The *libcgroup-tools* package provides tools, such as **cgcreate**, **cgexec**, and **cgclassify**, which can be used to manage cgroups.

Each **cpuset** cgroup defines parameters that are mandatory and must be set before moving a task into the cgroup. Here are some parameters related to CPU and memory affinity:

cpuset.cpus
: The parameter specifies the permitted CPUS that a task in the cgroup can access. This parameter is mandatory, and allows a comma-separated list, a range with (**-**), or a combination of both.

```
[root@host ~]# cat /sys/fs/cgroup/cpuset/cpuset.cpus
0-191
```

The above screen shows that CPUs 0 to 191 are available.

cpuset.mems
: The parameter specifies the permitted NUMA memory nodes that a task in the cgroup can access. This parameter is mandatory, and allows a comma-separated list range with (**-**), or a combination of both.

```
[root@host ~]# cat /sys/fs/cgroup/cpuset/cpuset.mems
0-3
```

The above screen shows that memory from 0 to 3 nodes are available.

cpuset.cpu_exclusive and cpuset.mem_exclusive
: The parameter specifies whether **cpusets**, other than the current one and its parents and children, can share the CPU and memory nodes specified for the **cpuset**. This parameter is optional, and by default is set to **0** to disable sharing CPU and memory nodes. Setting to **1** enables sharing CPU and memory nodes among parents and children.

> **Note**
> The **cpuset.mems** parameter has a value of **0** on systems that do not have a NUMA architecture.

Currently, the **systemd** service does not directly manage the **cpuset** controller. The **cpuset** controller can be managed either manually, or by using the tools provided by the *libcgroup-tools* package.

One way of managing the **cpuset** is with an **ExecStartPost** script inside a unit. The following procedure assigns the **httpd** service to a new **cpuset** cgroup named **cpuset0**:

> **Note**
> For production, use more advanced scripts with error handling.

1. Create the **/usr/local/bin/cpuset0** script in order to create a **cpuset** with the name **cpuset0**. The script assigns CPUs from **2** to **8**, and memory from NUMA node **0** to **1**, to the new **cpuset**.

```
[root@host ~]# vim /usr/local/bin/cpuset0
#!/bin/bash
mkdir -p /sys/fs/cgroup/cpuset/cpuset0
echo 2-8 > /sys/fs/cgroup/cpuset/cpuset0/cpuset.cpus
echo 0-1 > /sys/fs/cgroup/cpuset/cpuset0/cpuset.mems
for PID in $(pgrep httpd); do
  echo ${PID} > /sys/fs/cgroup/cpuset/cpuset0/tasks
done
```

2. Add the following to a drop-in file for the **`httpd`** service:

```
[root@host ~]# vim /etc/systemd/system/httpd.service.d/cpuset.conf
[Service]
ExecStartPost=/usr/local/bin/cpuset0
```

3. Reload **`systemd`** configuration and restart the **`httpd`** service.

```
[root@host ~]# systemctl daemon-reload
[root@host ~]# systemctl restart httpd
```

4. View the content of the **`/proc/PID/status`** file to list the allocated CPUs and memory, where PID is the process ID.

```
[root@host ~]# pgrep httpd
125203
125204
125205
125207
125208
125209
[root@host ~]# egrep "Cpus*|Mems*" /proc/12520?/status
/proc/125203/status:Cpus_allowed:
 00000000,00000000,00000000,00000000,00000000,000001fc
/proc/125203/status:Cpus_allowed_list:  2-8
/proc/125203/status:Mems_allowed:
 00000000,00000000,00000000,00000000,00000000,...,00000003
/proc/125203/status:Mems_allowed_list:  0-1
/proc/125204/status:Cpus_allowed:
 00000000,00000000,00000000,00000000,00000000,000001fc
/proc/125204/status:Cpus_allowed_list:  2-8
/proc/125204/status:Mems_allowed:
 00000000,00000000,00000000,00000000,00000000,...,00000003
/proc/125204/status:Mems_allowed_list:  0-1
...output omitted...
```

The CPU affinity is expressed in the bitmask, represented in hexadecimal, and is a series of **n** bits, where each bit represents whether the kernel pins the given process to the corresponding CPU. To calculate the value associated with a specific CPU, use the formula: value = 2^n , where n is the CPU number, starting at **0** for the first CPU. For example, to pin process to just CPUs 0-1, 3, and 5-8, calculate the CPU affinity mask as show below:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# printf '%032x' $((2**0 + 2**1 + 2**3 + 2**5 + 2**6 + 2**7 + 2**8))
00000000000000000000000000000001eb
```

Thus the CPU affinity mask for 16 cores and 2 threads per core system is **00000000,00000000,00000000,000001eb**.

In the above output, the **systemd** configuration pins the service to CPUs **2** to **8**; the bitmask on a 24 cores and 2 thread per core system is: **00000000,00000000,00000000,00000000,00000000,000001fc**.

Balancing Interrupts

An interrupt is a signal originating from hardware (hard-interrupt) or software (soft-interrupt) sent to the processor, indicating an event that requires immediate attention. An interrupt is sometimes also called an **IRQ**, which stands for Interrupt ReQuest. The kernel handles the interrupt by choosing a CPU to execute the corresponding application code. The kernel distributes the interrupts evenly across CPUs in a multiprocessor system. The delay in interrupt processing when writing new data and instruction caches across CPUs creates conflicts with other processing occurring on the CPU. In order to overcome the problem of writing across CPUs, the kernel allows administrators to pin the time-critical interrupts and processes to a specific CPU or a range of CPUs.

The **/proc/interrupts** file details how often the kernel handles specific interrupts on a specific CPU.

The kernel uses the **/proc/irq/number/smp_affinity** file, where the number is the IRQ number, to execute a given interrupt handler on a CPU. The file specifies the CPU as a bitmask value, represented as a hexadecimal number. For every CPU that handles an interrupt, the corresponding bit is set to **1**. When the bit is set to **0**, the kernel does not allocate the given CPU for interrupt handling.

For example, to limit interrupts with a value of **50** to CPUs 0, 2, and 7, calculate: $2^0 + 2^2 + 2^7 = 133$ = **0x85**. To make this calculation directly from the shell, and set the affinity mask on a 32-bit system, use:

```
[root@host ~]# printf '%032x' $((2**0 + 2**2 + 2**7)) > /proc/irq/50/smp_affinity
```

Alternatively, edit the affinity mask in the **/proc/irq/number/smp_affinity_list** file. This file takes a comma separated list of CPU indexes (starting at **0**) that can be used to handle the interrupt.

In Red Hat Enterprise Linux, the **irqbalance** service is enabled by default on new installs. The **irqbalance** service adjusts the **smp_affinity** file for all interrupts every 10 seconds. In a NUMA based system, this enables the interrupt handler with the highest chance of getting cache hits during execution. On single-core systems and dual-core systems that share their L2 cache, **irqbalance** does not have much role to play.

An administrator uses the following settings in the **/etc/sysconfig/irqbalance** file to tune the behavior of the **irqbalance** service:

IRQBALANCE_ONESHOT
: Set to **yes**, the **irqbalance** service sleeps for one minute after startup, rebalances interrupts once, and then exits. This variable is useful for avoiding the overhead of waking up the service every 10 seconds.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

`IRQBALANCE_BANNED_CPUS`
: Set to CPUs bitmask, the **`irqbalance`** service does not assign any interrupts to the CPUs listed. The value is a hexadecimal bitmask of CPUs, which the **`irqbalance`** service ignores when deciding which CPU to use for a specific interrupt. For example, set the value to **`000000fe`** to ban CPUs 1-15 on a 32-bit system from interrupt handling, and allow only CPU0 for interrupt processing (and any CPU with an index of 16 or higher).

CPU Partitioning

One crucial technique for real-time tuning is to isolate a group of CPU cores for the use of one or more real-time applications running on the system. CPU partitioning takes into account the CPU topology, so that related process threads reside on the same core in a NUMA node, reducing jitters and kernel interruptions. Red Hat Enterprise Linux provides the **`cpu-partitioning`** tuned profile that partitions the CPU automatically upon boot. Install the *tuned-profiles-cpu-partitioning* package to deploy the tuned profile on a system. Customize the **`cpu-partitioning`** tuned profile using the **`/etc/tuned/cpu-partitioning-variables.conf`** file. There are two configuration options:

`isolated_cores`
: List of CPUs to isolate. This option is mandatory. The rest of the CPUs that are not part of the list are considered for running housekeeping process.

`no_balance_cores`
: List of CPUs to be ignored by the kernel when load-balancing processes. In most of the scenarios, this list is same as that of **`isolated_cores`**. This option is not required.

References

`systemd.exec`(5), **`irqbalance`**(1), and **`tuned-profiles-cpu-partitioning`**(7) man pages.

The IRQ affinity is documented in the **`/usr/share/doc/kernel-doc-*/Documentation/IRQ-affinity.txt`** file.

For more information on configuring the CPU usage of a service, refer to the *Realtime-Specific Tuning* chapter in the *Red Hat Enterprise Linux for Real Time 8 Tuning Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/8/html/tuning_guide/chap-realtime-specific_tuning

Guided Exercise

Configuring CPU Affinity and Isolation

In this exercise, you will configure and verify application isolation using CPU and interrupt affinity tuning.

Outcomes

You should be able to:

- Pin the **`httpd`** service to the first CPU on the system.
- Assign interrupts to a specific CPU.

Before You Begin

Log in to **`workstation`** as **`student`** using **`student`** as the password.

On **`workstation`**, run the **`lab cputuning-affinity start`** command. The command determines if the host, **`servera`**, is reachable on the network and ensures the *tuna* package is installed.

```
[student@workstation ~]$ lab cputuning-affinity start
```

1. On **`workstation`** open two terminal windows side by side. On each terminal, use **`ssh`** to log in to **`servera`** as the **`student`** user. On the first terminal, use **`sudo -i`** to switch to **`root`**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Install, enable, and start the **`httpd`** service.

   2.1. Install the *httpd* package.

```
[root@servera ~]# yum install httpd
...output omitted...
```

   2.2. Enable and start the **`httpd`** service.

```
[root@servera ~]# systemctl enable httpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/
lib/systemd/system/httpd.service.
```

3. View the CPU assignments for all **`httpd`** processes, and all of their threads. Observe that the CPU affinity for all threads are distributed across both the processors.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# tuna -t httpd -P
                      thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  24095  OTHER     0      0,1        34            3           httpd
  24096  OTHER     0      0,1         6            0           httpd
  24097  OTHER     0      0,1         5            0           httpd
  24098  OTHER     0      0,1         7            0           httpd
  24099  OTHER     0      0,1         5            0           httpd
```

4. Create a **systemd** drop-in file for the **httpd** service to set the CPU affinity to use the first processor. Remember that **systemd** starts with **0** when counting CPUs, and that the **CPUAffinity** parameter takes a comma-separated list of CPU numbers.

   4.1. Create a **systemd** drop-in directory for the **httpd.service** service.

```
[root@servera ~]# mkdir /etc/systemd/system/httpd.service.d
```

   4.2. Create a new file called **/etc/systemd/system/httpd.service.d/10-CPUAffinity.conf** with the following content:

```
[root@servera ~]# vim /etc/systemd/system/httpd.service.d/10-CPUAffinity.conf
[Service]
CPUAffinity=0
```

5. Reload the configuration changes.

   5.1. Reload the **systemd** service to reload all unit definitions from disk.

```
[root@servera ~]# systemctl daemon-reload
```

   5.2. Restart the **httpd** service.

```
[root@servera ~]# systemctl restart httpd.service
```

6. View the CPU assignments for all **httpd** processes, and all of their threads. Observe that the **CPU0** processor handles all the **httpd** processes.

```
[root@servera ~]# tuna -t httpd -P
                      thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  24344  OTHER     0        0        11            3           httpd
  24346  OTHER     0        0         6            0           httpd
  24347  OTHER     0        0         7            0           httpd
  24348  OTHER     0        0         7            0           httpd
  24349  OTHER     0        0         7            0           httpd
```

7. From the second terminal with an open SSH connection to **servera**, run the following command, and keep it running for the duration of this exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@servera ~]$ watch -dn1 cat /proc/interrupts
```

You may want to increase the size of the terminal window to see more interrupts. Using the **-d** option with the **watch** command highlights the differences since the last run.

8. From the first terminal, use the **irqbalance** service to have all interrupts automatically assigned to the first processor.

   8.1. From the first terminal, edit the **/etc/sysconfig/irqbalance** file to uncomment the **IRQBALANCE_BANNED_CPUS** parameter, and edit it with the following content:

```
[root@servera ~]# vim /etc/sysconfig/irqbalance
...output omitted...
IRQBALANCE_BANNED_CPUS=00000002
...output omitted...
```

   The parameter takes the bit mask for CPUs as the value. Any CPU that has its bit set will not be eligible to handle interrupts when the **irqbalance** service rebalances the interrupts.

   8.2. Restart the **irqbalance** service.

```
[root@servera ~]# systemctl restart irqbalance
```

9. From the second terminal, view the **watch** command for **/proc/interrupts**, and notice that the second processor handles very few interrupts. The reasons for a processor with a bit mask set for the **IRQBALANCE_BANNED_CPUS** parameter still handling interrupts are:

   - Some of the hardware interrupts are tied to a specific CPU.
   - The **irqbalance** service does not reassign interrupts to a CPU that shares the same L2 cache, because there is no performance improvement.

10. Disable and stop the **irqbalance.service** service.

```
[root@servera ~]# systemctl disable irqbalance.service --now
Removed /etc/systemd/system/multi-user.target.wants/irqbalance.service.
```

11. Set the CPU mask for every IRQ so that it will only ever be handled by the first processor (**CPU0**).

```
[root@servera ~]# for IRQ in /proc/irq/*/smp_affinity; do \
echo 1 > ${IRQ} 2>/dev/null; done
```

12. Observe the **watch** command to determine the impact of rebalancing.

> **Note**
> If you see few or no interrupts, move the mouse around inside the console window of the **servera** machine. This should show some USB-related interrupts.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Notice that all hardware interrupts are now handled by your first processor. Both the processors handle the timer interrupts, function call interrupts, and rescheduling interrupts.

13. Repeat the manual rebalancing step, but now have all interrupts handled by the second processor.

```
[root@servera ~]# for IRQ in /proc/irq/*/smp_affinity; do \
echo 2 > ${IRQ} 2>/dev/null; done
```

14. Observe the **watch** command to determine the impact of rebalancing.

    All hardware interrupts must now be handled by the second processor, but as before, both the processors handle the timer interrupts, function call interrupts, and rescheduling interrupts.

    Press **Ctrl**+**C** to exit the **watch** command.

15. Log out from **servera** from both the terminals.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab cputuning-affinity finish** command to complete this exercise.

```
[student@workstation ~]$ lab cputuning-affinity finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Profiling Cache Use

Objectives

After completing this section, you should be able to describe how the CPU caches are used by applications.

Describing CPU Caches

Computer systems use different memory tiers to ensure that processors have their data as close as possible when ready to start processing. The fastest storage on any system are memory registers on the processor that operate at the processor's clock speed. Processor registers offer a very limited amount of storage, and in terms of bytes per dollar, the storage is quite expensive. System designers must use slower memory for storing larger amounts of data than available in processor registers.

Cache memory is faster than main memory. Whereas main memory has access times of 8+ nanoseconds, caches have access times as little as a few CPU clock cycles. Most systems have two to three levels of cache. Level 1 (L1) cache is very small, on the order of a few KiB, and is typically located on the processor chip itself. Level 2 (L2) cache is larger than L1 cache (typically hundreds of KiB), and is somewhat slower. Some systems may implement a third or even fourth level of cache, external to the processor, that is slower and larger than L2 cache (typically several MiB in size), but still faster than main memory.

For example, on a Intel i7 CPU package, each core may have its own private L1 and L2 caches on-die, and all four may share a large common L3 cache on-die or in the CPU package. The below graphics are the output of the **`lstopo --no-legend --no-io`** command, which shows the hardware topology of a system.

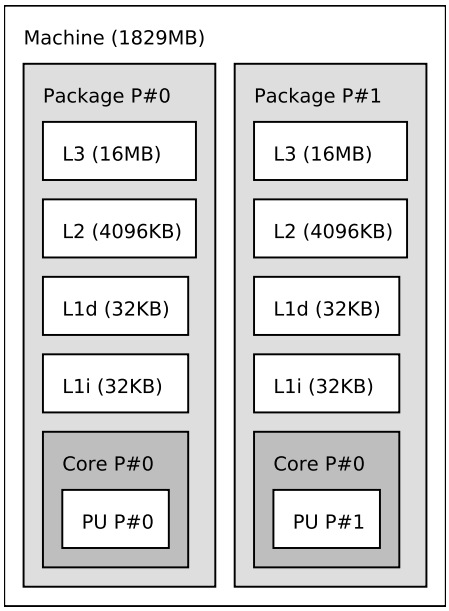


Figure 7.5: Hardware topology of Intel Core2 cache diagram

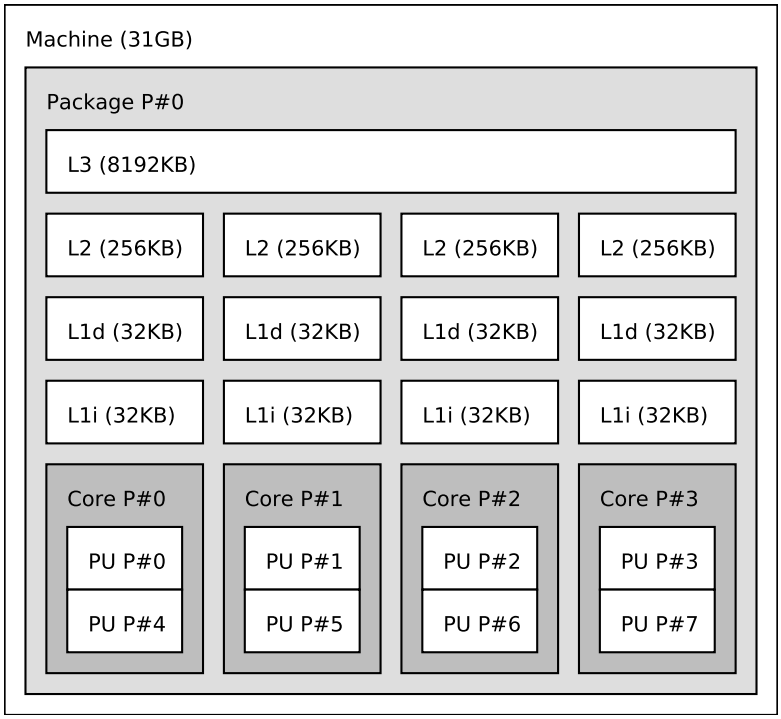


Figure 7.6: Hardware topology of Intel i7 cache diagram

Use the **`lscpu`** or **`lshw -class memory`** to list the amount of caches on a system.

```
[root@host ~]# lscpu
...output omitted...
Model name:            Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
...output omitted...
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              8192K
...output omitted...
```

```
[root@host ~]# lshw -class memory
...output omitted...
  *-cache:0
       description: L2 cache
       physical id: 3e
       slot: CPU Internal L2
       size: 1MiB
       capacity: 1MiB
       capabilities: internal write-back unified
       configuration: level=2
  *-cache:1
       description: L1 cache
       physical id: 3f
       slot: CPU Internal L1
       size: 256KiB
       capacity: 256KiB
       capabilities: internal write-back
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
        configuration: level=1
   *-cache:2
        description: L3 cache
        physical id: 40
        slot: CPU Internal L3
        size: 8MiB
        capacity: 8MiB
        capabilities: internal write-back unified
        configuration: level=3
 ...output omitted...
```

Cache Access Latency

Multiple levels of CPU caches helps a processor to deliver optimum configuration of size and latency. The access latency for data from the L1 cache is 1 ns. The access latency for the larger L2 cache is typically 10 ns, and that for main memory is around 60 ns. The above values are a rough estimation and depend on core frequencies and memory speeds.

In multiprocessor architecture, processors are connected using a shared system bus or a dedicated interconnect. The use of shared bus, also known as front-side bus, was prevalent in earlier multiprocessor architectures. With the advent of Non-Uniform Memory Architecture (NUMA), the CPU interconnect connects multiple processor and I/O controllers, for example, Intel's Ultra Path Interconnect (UPI) and AMD's HyperTransport (HT). The interconnect between processors allows higher bandwidth and discontented access in comparison to shared system bus.

Cache architecture

Cache memory is organized into a *cache line*. Each cache line can be used to cache a specific location in memory. The storage in cache line is a range of bytes, defined as *cache line size*, which a compiler uses to optimize a program performance. Most computer systems have separate caches for instructions for the processor, the I-cache, data, and D-cache. This is sometimes referred to as the *Harvard memory architecture*.

On a multiprocessor system, each CPU has its own separate cache. Each cache has an associated cache controller. When a processor makes a reference to main memory, the cache controller first checks to see if the requested address is present in cache to fulfill the request from the processor. This is referred to as a *cache hit*. If the requested memory is not present in cache, then a *cache miss* occurs; the requested location must be read from main memory, and the processor stores the data in the cache. This is referred to as a *cache line fill*.

On multiprocessor systems, the need to maintain cache coherency arises. That is, if a processor updates a line of cache in memory, it must let the other processors on the system know that the location in memory has been updated, in case they are caching it also. This is referred to as *cache snooping* and is performed in the system hardware.

Note
NUMA systems group memory into *nodes*, and then assign those nodes to individual CPU sockets.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

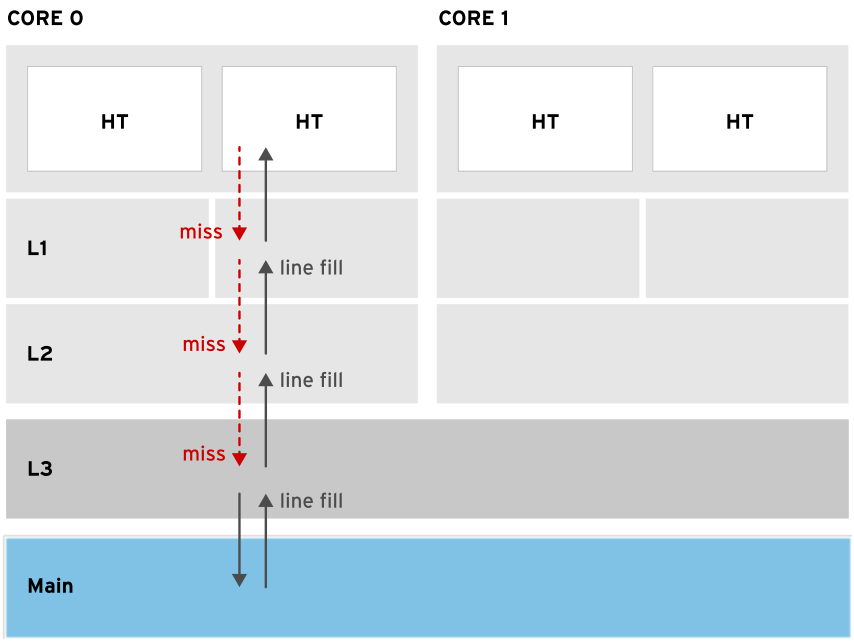


Figure 7.7: Cache miss and cache line fill

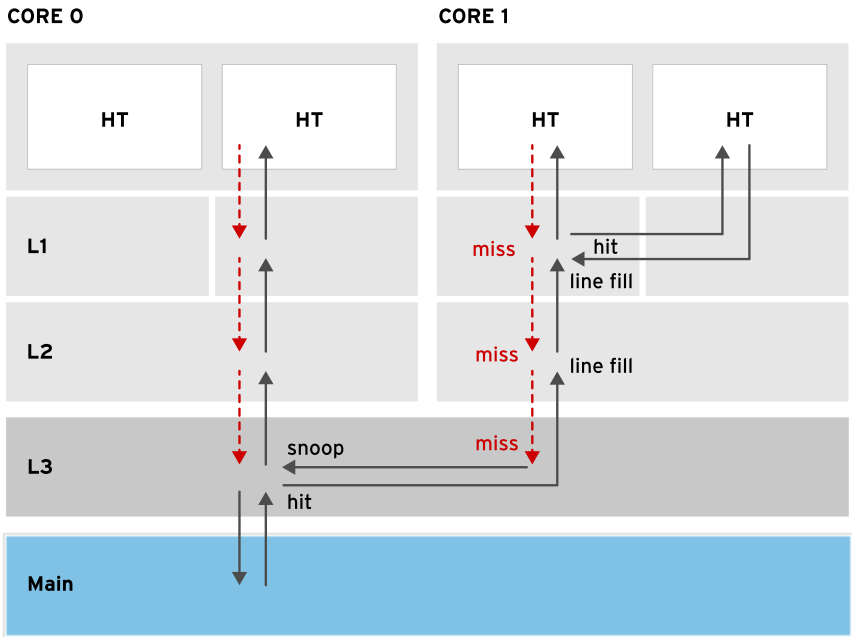


Figure 7.8: Cache miss, cache line fill, cache snoop, and cache hit

The cache controller contains an array of cache entries for each line of cache. In addition to the cached memory contents, each cache entry consists of a *tag* and flags describing the status of that cache entry. The cache controller uses the tag to identify which main memory location has been cached for an entry.

The processor both reads from and writes to cache memory. In the case of write operations, cache memory can be configured as either write-through or write-back. If write-through caching is enabled, then when a particular line of cache memory is updated, the corresponding location in

main memory must be updated as well. If write-back caching is enabled, a write to a cache line does not get written back to main memory until the cache line is reallocated. Write-back caching is more efficient than write-through caching. On x86 platforms, each page of memory contains control bits that can be used to disable caching of pages, and to disable write-back caching. The Linux kernel clears both bits on all memory pages, so all memory accesses are cached with write-back caching.

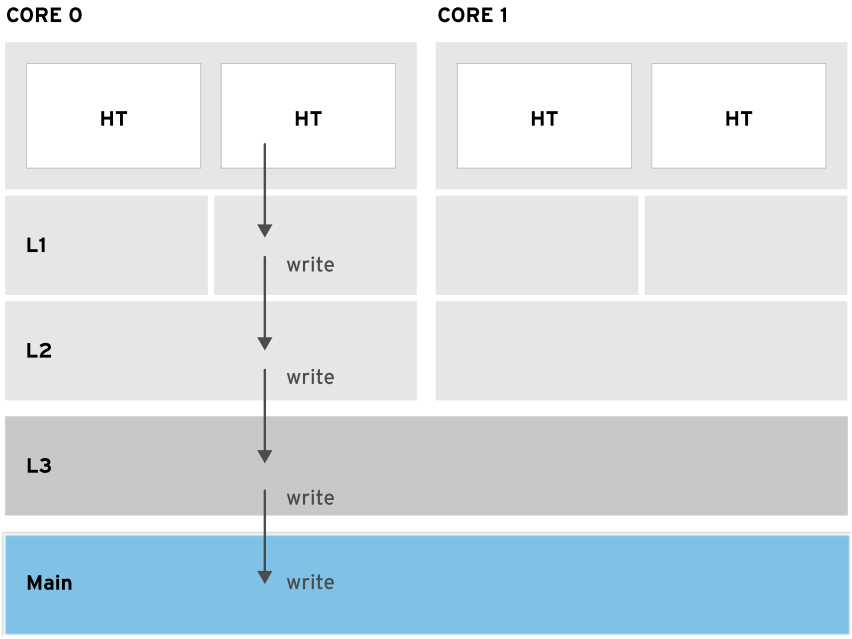


Figure 7.9: Cache write through

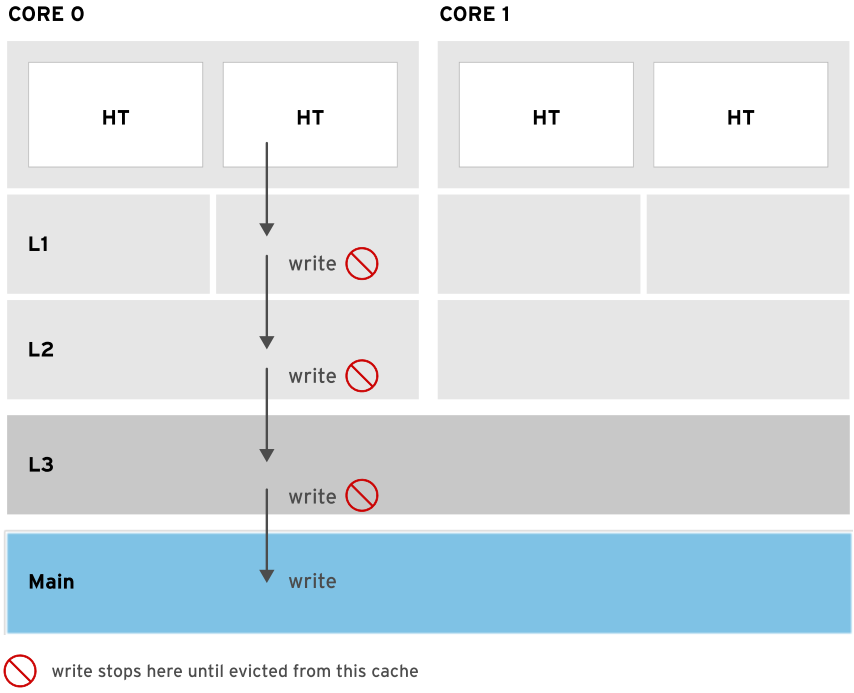


Figure 7.10: Cache write back

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Cache Associativity

Cache asssociativity is a cache characteristic describing the constraint on locating places to store new entries in the cache. The replacement policy decides where in the cache it copies a particular entry of main memory. There are mainly three types of cache associativity:

Direct mapped cache
Direct mapped cache is the least expensive type of cache memory. Each line of direct mapped cache can only cache a specific location in main memory. With direct mapped cache, the search time is faster because cache data maps to one specific memory location, but the cache hit rate is low.

Fully associative cache
Fully associative cache memory is the most flexible type of cache memory, and consequently the most expensive because it requires the most circuitry to implement. A fully associative cache memory line can cache any location in main memory. With fully associative cache, the search time is extremely long because the CPU has to look through the entire cache to find out if the data is present before searching main memory, but the cache hit rate is high.

Set associative cache
Set associative cache memory provides a good compromise between direct mapped cache and fully associative cache. This cache memory is usually referred to as *n-way set associative*, where **n** is some power of **2**. Set associative cache memory allows a memory location to be cached into any one of **n** lines of cache. Most systems compromise and use set associative cache memory.

Profiling CPU Cache Usage

Runtime performance of a particular program largely depends on the efficiency of cache usage, since caches are faster than system memory. To decide which of two programs has the best runtime performance, it is helpful to compare the efficiency of their cache usage. One instruction cache read in L1 cache is equivalent to one executed instruction, the profiling tool helps a developer to find how many instructions are executed per line, which is useful for profiling and test coverage.

Profiling CPU Cache Using Valgrind

The *valgrind* utility collection provides a set of tools for memory debugging and profiling. One of the tools provided in the **`valgrind`** toolset is the **`cachegrind`** tool. The **`cachegrind`** tool allows users to do cache simulation, and to annotate the source code with the number of cache misses. This tool records the following:

- L1 instruction cache reads and misses.
- L1 data cache read load, read misses, write load, and write misses.
- L2 cache reads load and read misses, as well as write load and write misses.

To run a program with the **`valgrind`** command, the program requires debugging info. Red Hat Enterprise Linux provides the *packagename-debuginfo* packages for all architecture-dependent RPMs bundled with the operating system. Use the **`gcc`** command with the **`-g`** option to generate debugging information for your program.

Use the **`valgrind`** command with the **`--tool=cachegrind`** option to display cache performance for the **`cache1`** program: The performance-relevant numbers are the cache miss rates of the application. The command also collects line-by-line information in the **`cachegrind.out.PID`**, where PID is the program's process ID.

```
[root@host ~]# gcc -o cache1 -g cache1.c
[root@host ~]# valgrind --tool=cachegrind cache1
...output omitted...
==3258==
==3258== I   refs:      194,476
==3258== I1  misses:        932
==3258== LLi misses:        920
==3258== I1  miss rate:    0.48%
==3258== LLi miss rate:    0.47%
==3258==
==3258== D   refs:       59,025  (45,928 rd   + 13,097 wr)
==3258== D1  misses:      3,113  ( 2,502 rd   +    611 wr)
==3258== LLd misses:      2,606  ( 2,041 rd   +    565 wr)
==3258== D1  miss rate:     5.3% (   5.4%     +    4.7%  )
==3258== LLd miss rate:     4.4% (   4.4%     +    4.3%  )
==3258==
==3258== LL refs:         4,045  ( 3,434 rd   +    611 wr)
==3258== LL misses:       3,526  ( 2,961 rd   +    565 wr)
==3258== LL miss rate:      1.4% (   1.2%     +    4.3%  )
...output omitted...
```

In the above output, the L1 I-cache is summarized first, where **I** represents L1 I-cache, and **LLi** represents lowest level cache instruction, which is cache above L1 in the hierarchy. The L1 data cache is summarized next, where **D** represents L1 D-cache and **LLd** represents lowest level data instruction.

Profiling CPU Cache Using Perf

Another tool that is useful for analyzing memory performance and cache usage is **perf**. The **perf** command utilizes hardware counters to determine cache statistics. The hardware counters are not available in virtual machines. Therefore, **perf** cannot be used to track hardware performance counters inside virtual machines, but must run directly on bare-metal installations.

The **perf list** command displays a list of available events. For tracking cache misses, the name of the cache miss hardware event is required for executing the **perf stat** command.

```
[root@host ~]# perf list
...output omitted...
  cache-misses                                       [Hardware event]
...output omitted...
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
...output omitted...mphasis>
```

Run the **perf stat** command to display the cache misses for executing the **tar** command to create an archive:

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# perf stat -e instructions,cycles, \
L1-dcache-loads,L1-dcache-load-misses,LLC-load-misses,LLC-loads \
tar cvf backup.tar /etc
...output omitted...
 Performance counter stats for 'tar cvf backup.tar /etc':

        95,733,574      instructions              #    0.93  insn per cycle
    (65.23%)
       103,078,536      cycles
    (82.63%)
        26,498,868      L1-dcache-loads
    (81.25%)
         2,842,868      L1-dcache-load-misses     #   10.73% of all L1-dcache hits
    (36.12%)
           250,096      LLC-load-misses           #   38.66% of all LL-cache hits
    (32.52%)
           646,860      LLC-loads
    (46.71%)

       0.028904566 seconds time elapsed

       0.005115000 seconds user
       0.022559000 seconds sys
```

In the above output, the **`perf`** command, apart from **`instructions`** and **`cycles`**, measures the following hardware cache events:

`L1-dcache-load-misses`
: The value shows the L1 D-cache misses.

`LLC-load-misses`
: The value shows misses at the last level cache, which is a larger cache furthest away from the CPU core. The difference between **`L1-dcache-load-misses`** and **`LLC-load-misses`** provides the effectiveness of the CPU caches.

References

`valgrind`(1), **`perf`**(1), and **`perf-list`**(1) man pages

Valgrind
http://www.valgrind.org/

For more information on **`valgrind`**, refer to documentation at **`/usr/share/doc/valgrind/html/manual.html`**.

Memory part 2: CPU caches
https://lwn.net/Articles/252125/

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Profiling Cache Use

In this exercise, you will analyze cache usage of programs.

Outcomes

You should be able to determine the program that utilizes the caches most efficiently with **valgrind**.

Before You Begin

Log in to **workstation** as **student**, using **student** as the password.

On **workstation**, run the **lab cputuning-cpucache start** command. This command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the **cache1** and **cache2** programs that you use to evaluate the program that exhibits the better cache usage. This program has been developed specifically for this course.

```
[student@workstation ~]$ lab cputuning-cpucache start
```

1. On **workstation**, use **ssh** to log in to **servera** as the **student** user, then use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Install the *valgrind* package.

```
[root@servera ~]# yum install valgrind
...output omitted...
```

3. Analyze the cache usage of the **cache1** and **cache2** programs with the **valgrind** command, using the built-in **cachegrind** tool.

   3.1. Use the **valgrind --tool=cachegrind** command to assess the cache usage of the **cache1** program.

```
[root@servera ~]# valgrind --tool=cachegrind cache1
==6000== Cachegrind, a cache and branch-prediction profiler
==6000== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==6000== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==6000== Command: cache1
==6000==
--6000-- warning: L3 cache found, using its data for the LL simulation.
Starting
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
Finished
==6000==
==6000== I    refs:      6,750,720,315
==6000== I1   misses:            1,010
==6000== LLi misses:               998
==6000== I1   miss rate:          0.00%
==6000== LLi miss rate:           0.00%
==6000==
==6000== D    refs:      3,937,859,248  (3,375,271,104 rd   + 562,588,144 wr)
==6000== D1   misses:       35,159,405  (        2,521 rd   +  35,156,884 wr)
==6000== LLd misses:       35,158,986  (        2,133 rd   +  35,156,853 wr)
==6000== D1   miss rate:           0.9% (          0.0%     +         6.2%  )
==6000== LLd miss rate:            0.9% (          0.0%     +         6.2%  )
==6000==
==6000== LL refs:          35,160,415  (        3,531 rd   +  35,156,884 wr)
==6000== LL misses:        35,159,984  (        3,131 rd   +  35,156,853 wr)
==6000== LL miss rate:             0.3% (          0.0%     +         6.2%  )
```

3.2. Use the **`valgrind --tool=cachegrind`** command to assess the cache usage of the **`cache2`** program.

```
[root@servera ~]# valgrind --tool=cachegrind cache2
==6001== Cachegrind, a cache and branch-prediction profiler
==6001== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==6001== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==6001== Command: cache2
==6001==
--6001-- warning: L3 cache found, using its data for the LL simulation.
Starting
Finished
==6001==
==6001== I    refs:      6,750,720,315
==6001== I1   misses:            1,010
==6001== LLi misses:               998
==6001== I1   miss rate:          0.00%
==6001== LLi miss rate:           0.00%
==6001==
==6001== D    refs:      3,937,859,248  (3,375,271,104 rd   + 562,588,144 wr)
==6001== D1   misses:      562,503,145  (        2,521 rd   + 562,500,624 wr)
==6001== LLd misses:       35,164,610  (        2,133 rd   +  35,162,477 wr)
==6001== D1   miss rate:          14.3% (          0.0%     +       100.0%  )
==6001== LLd miss rate:            0.9% (          0.0%     +         6.3%  )
==6001==
==6001== LL refs:         562,504,155  (        3,531 rd   + 562,500,624 wr)
==6001== LL misses:        35,165,608  (        3,131 rd   +  35,162,477 wr)
==6001== LL miss rate:             0.3% (          0.0%     +         6.3%  )
```

4. Based on the previous output, the D1 cache miss-rate of the **`cache1`** program is 0.9%, while the **`cache2`** program has a D1 miss-rate of 14.3%. This also results in slower execution of the **`cache2`** program.

5. Exit **`servera`** to return to **`workstation`**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab cputuning-cpucache finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab cputuning-cpucache finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning CPU Utilization

Performance Checklist

In this lab, you will use a variety of CPU tuning techniques to manage CPU resource contention.

Outcomes

You should be able to:

- Configure the **irqbalance** service to exclude first processor from IRQ balancing.
- Pin the **httpd** service to the first processor.
- Assign non-real-time scheduling policy and priority to the **vsftpd** service.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab cputuning-review start** command. The command determines if the host, **serverb**, is reachable on the network, and copies required files to the host for this lab exercise.

```
[student@workstation ~]$ lab cputuning-review start
```

You are tuning your system for two CPU-intensive services. Pin the **httpd** service to the first processor, and configure the **vsftpd** service to use the **batch** scheduler and a priority of **0**. Configure the system such that no interrupts interfere with the service running on the first processor, and the second processor handles all the interrupts.

> **Note**
> Because both processors in the classroom system share an L2 cache, the **irqbalance** command will not distribute hardware interrupts across processors. For this lab, the configuration is enough, but in a production environment, all interrupts must have their CPU affinity set manually to achieve this goal.

1. Configure the **irqbalance** service on **serverb** to never assign interrupts to the first processor. The CPU mask for first CPU in a 2-processor system is **00000001**.
2. Verify that the **irqbalance** service ignores the first processor, **CPU0**, for IRQ balancing.
3. Install the *httpd* package. Configure the **httpd** service to pin the service to the first processor. Enable and start the **httpd** service.
4. Install the *vsftpd* package, and then configure the **vsftpd** service to use the **SCHED_BATCH** scheduler with a priority of **0**. Enable and start the **vsftpd** service.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Evaluation

On **workstation**, run the **lab cputuning-review grade** command to confirm the success of this exercise.

```
[student@workstation ~]$ lab cputuning-review grade
```

Finish

On **workstation**, run the **lab cputuning-review finish** command to complete this exercise.

```
[student@workstation ~]$ lab cputuning-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning CPU Utilization

Performance Checklist

In this lab, you will use a variety of CPU tuning techniques to manage CPU resource contention.

Outcomes

You should be able to:

- Configure the **irqbalance** service to exclude first processor from IRQ balancing.
- Pin the **httpd** service to the first processor.
- Assign non-real-time scheduling policy and priority to the **vsftpd** service.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab cputuning-review start** command. The command determines if the host, **serverb**, is reachable on the network, and copies required files to the host for this lab exercise.

```
[student@workstation ~]$ lab cputuning-review start
```

You are tuning your system for two CPU-intensive services. Pin the **httpd** service to the first processor, and configure the **vsftpd** service to use the **batch** scheduler and a priority of **0**. Configure the system such that no interrupts interfere with the service running on the first processor, and the second processor handles all the interrupts.

> **Note**
> Because both processors in the classroom system share an L2 cache, the **irqbalance** command will not distribute hardware interrupts across processors. For this lab, the configuration is enough, but in a production environment, all interrupts must have their CPU affinity set manually to achieve this goal.

1. Configure the **irqbalance** service on **serverb** to never assign interrupts to the first processor. The CPU mask for first CPU in a 2-processor system is **00000001**.

   1.1. On **workstation**, open two terminal windows side by side. On each terminal, use **ssh** to log in to **serverb** as **student**. In the first terminal, use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

1.2. From the second terminal, generate a network traffic on **serverb**.

```
[student@serverb ~]$ ping localhost
...output omitted...
```

1.3. From the first terminal, view the processors the **irqbalance** service uses to handle the network device interrupts. The **virtio1-input.0** is the input queue for the first network device attached to the system.

Observe that the second column value increases, indicating that the **CPU0** handles all the interrupts from the network device.

```
[root@serverb ~]# watch -dn1 "cat /proc/interrupts |\
egrep 'virtio.*(input|output)'"
...output omitted...
44:        800      76847   PCI-MSI 524289-edge      virtio1-input.0
45:          0          1   PCI-MSI 524290-edge      virtio1-output.0
```

Press **Ctrl**+**C** to exit the command on both the terminals.

1.4. Edit **/etc/sysconfig/irqbalance** to include the following line:

```
[root@serverb ~]# vim /etc/sysconfig/irqbalance
...output omitted...
IRQBALANCE_BANNED_CPUS=00000001
...output omitted...
```

1.5. Restart the **irqbalance** service.

```
[root@serverb ~]# systemctl restart irqbalance
```

2. Verify that the **irqbalance** service ignores the first processor, **CPU0**, for IRQ balancing.

2.1. Generate network traffic on **serverb** from the second terminal.

```
[student@serverb ~]$ ping localhost
...output omitted...
```

2.2. View the processors the **irqbalance** service uses to handle the network device interrupts. The **virtio1-input.o** is the input queue for the first network device attached to the system.

Observe that the second column value does not increase, which verifies the **irqbalance** service ignores the **CPU0** processor for IRQ balancing.

```
[root@serverb ~]# watch -dn1 "cat /proc/interrupts |\
egrep 'virtio.*(input|output)'"
...output omitted...
44:        800      77990   PCI-MSI 524289-edge      virtio1-input.0
45:          0          1   PCI-MSI 524290-edge      virtio1-output.0
```

Press **Ctrl**+**C** to exit the command on both the terminals.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3. Install the *httpd* package. Configure the **`httpd`** service to pin the service to the first processor. Enable and start the **`httpd`** service.

3.1. Install the *httpd* package.

```
[root@serverb ~]# yum install httpd
...output omitted...
```

3.2. Create a **`systemd`** drop-in directory for the **`httpd`** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/httpd.service.d
```

3.3. Create a new **`systemd`** drop-in file for the **`httpd`** service, **`/etc/systemd/system/httpd.service.d/50-cpuaffinity.conf`**, with the following content:

```
[root@serverb ~]# vim /etc/systemd/system/httpd.service.d/50-cpuaffinity.conf
[Service]
CPUAffinity=0
```

3.4. Reload the configuration changes.

```
[root@serverb ~]# systemctl daemon-reload
```

3.5. Enable and start the **`httpd`** service.

```
[root@serverb ~]# systemctl enable httpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/
lib/systemd/system/httpd.service.
```

3.6. Verify that the **`httpd`** service is pinned to use the first processor.

```
[root@serverb ~]# tuna -t httpd -P
                          thread       ctxt_switches
   pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
 2137   OTHER     0        0        34            4           httpd
 2138   OTHER     0        0         6            0           httpd
 2139   OTHER     0        0         7            0           httpd
 2140   OTHER     0        0         7            0           httpd
 2141   OTHER     0        0         7            0           httpd
```

4. Install the *vsftpd* package, and then configure the **`vsftpd`** service to use the **`SCHED_BATCH`** scheduler with a priority of **`0`**. Enable and start the **`vsftpd`** service.

4.1. Install the *vsftpd* package.

```
[root@serverb ~]# yum install vsftpd
...output omitted...
```

4.2. Create a **`systemd`** drop-in directory for the **`vsftpd`** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/vsftpd.service.d
```

4.3. Create a new **systemd** drop-in file for the **vsftpd** service, **/etc/systemd/system/vsftpd.service.d/50-batch.conf**, with the following content:

```
[root@serverb ~]# vim /etc/systemd/system/vsftpd.service.d/50-batch.conf
[Service]
CPUSchedulingPolicy=batch
CPUSchedulingPriority=0
```

4.4. Reload the configuration changes.

```
[root@serverb ~]# systemctl daemon-reload
```

4.5. Enable and start the **vsftpd** service.

```
[root@serverb ~]# systemctl enable vsftpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/vsftpd.service → /usr/
lib/systemd/system/vsftpd.service.
```

4.6. After reboot, verify that the **vsftpd** service uses the **batch** scheduler with a priority of **0**.

```
[root@serverb ~]# tuna -t vsftpd -P
                      thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  2945   BATCH     0      0,1         1            0          vsftpd
```

4.7. Log out from **serverb**.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab cputuning-review grade** command to confirm the success of this exercise.

```
[student@workstation ~]$ lab cputuning-review grade
```

Finish

On **workstation**, run the **lab cputuning-review finish** command to complete this exercise.

```
[student@workstation ~]$ lab cputuning-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- CFS is the scheduling class for non-real-time processes, named **`SCHED_NORMAL`**. Meant for interactive applications to provide better desktop experience.
- The **`SCHED_DEADLINE`** scheduler guarantees the scheduling of real-time tasks, even in higher load systems.
- The **`irqbalance`** service adjusts CPU affinities to increase cache-hits for interrupts. The **`CPUAffinity`** parameter limits the CPUs available to a service.
- Runtime performance of a particular program largely depends on the efficiency of cache usage, since caches are faster than system memory.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 8

Tuning Memory Utilization

Goal

Manage settings for efficient memory utilization for different types of workloads.

Objectives

- Describe memory architecture, including virtual memory, caches, huge pages and process space.
- Describe system paging behavior and configure paging for server workload requirements.
- Describe the non-uniform memory architecture and the utilities for viewing and configuring NUMA topology.
- Describe memory overcommitment in typical application design.

Sections

- Configuring Memory Architecture (and Guided Exercise)
- Configuring Memory Paging and Reclamation (and Guided Exercise)
- Configuring NUMA Topology (and Guided Exercise)
- Managing Memory Overcommit (and Guided Exercise)

Lab

Tuning Memory Utilization

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring Memory Architecture

Objectives

After completing this section, you should be able to describe memory architecture, including virtual memory, caches, huge pages, and process space.

Explaining Virtual and Physical Memory

Management of system memory is one of the more sophisticated things that the kernel does. Efficient memory management is critical for good process performance on the system.

Computer systems organize memory into fixed-size chunks called *pages*. The size of a page depends on the processor architecture; for x86_64, the standard *page size* is 4 KiB. The physical RAM on a system is divided into *page frames*; one page frame holds one page of data.

Processes do not address physical memory directly. Instead, each process has a *virtual address space*. When a process requests memory, the kernel maps the physical address of a page frame to the virtual address in the process address space. From the perspective of the process, it has a private memory space and can only access the physical page frames that the kernel has mapped into its virtual addresses. Among other things, this helps enforce security restrictions and clear divisions between processes.

The size of a process's virtual address space does not depend on the installed physical RAM but rather depends on the processor architecture. On a 64-bit x86-64 system, the address space is 2^{64} bytes (16 EiB) in size.

An individual process generally does not use its entire address space. Much of it is unallocated and not mapped to any actual physical memory.

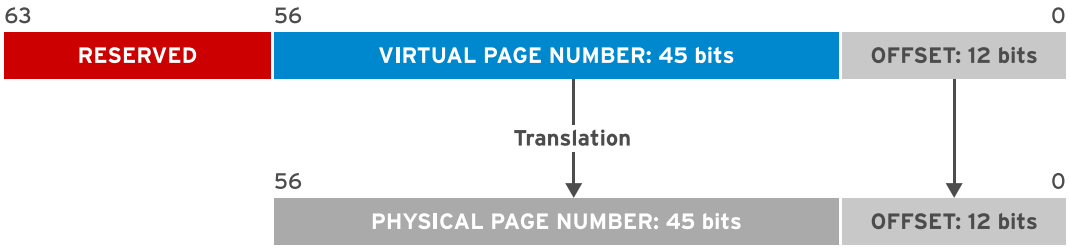


Figure 8.1: Virtual to physical memory translation

The diagram above shows the memory organization on a 64-bit x86-64 system with a page size of 4 KiB (2^{12}).

Each process has a 64-bit private virtual address space. The high 52 bits (63 to 12) represent the *page number*, which uniquely identifies the page in memory. The low 12 bits (11 to 0) identify the *offset* within the page where the data is stored.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Note

The latest generation of processors reserves the highest 7 bits of the virtual address, which causes a large block of virtual addresses to be unavailable for use by processes. Only 57 bits are left. With 57 bits, the kernel can address 128 PiB (2^{57}) of RAM. CPU manufacturers use this architecture to simplify the processors hardware designs.

Older processors reserve the highest 16 bits of the virtual address; the system can only use 48 bits for the virtual addresses. The addressable virtual memory is 256 TiB.

You can retrieve the virtual address size of your system by reviewing the **/proc/cpuinfo** file.

```
[user@host ~]$ cat /proc/cpuinfo
...output omitted...
address sizes : 52 bits physical, 57 bits virtual
...output omitted...
```

```
[user@host ~]$ cat /proc/cpuinfo
...output omitted...
address sizes : 46 bits physical, 48 bits virtual
...output omitted...
```

To access data, a process must first retrieve the associated physical page. Out of the 57 bits, the system uses 45 bits to address the page frames. The kernel converts (or translates) the private virtual page number (45 bits) to the actual physical page in RAM that contains the data.

With the physical page address, the process uses the lower 12 bits of the address as the offset to access the data in the page.

Note

The number of physical pages on a system depends on the available RAM and is usually not the same as the number of virtual pages a process can address.

Monitoring Processes Memory Usage

Most processes request more memory than they consume. When a process requests memory, it reserves virtual memory addresses but does not map them to physical page frames. The kernel only allocates physical page frames when the process starts using its memory.

Tools such as **ps** and **top** distinguish between two statistics: **VIRT** (or **VSZ**), the total amount of virtual memory a process has asked for, and **RES** (or **RSS**), the total amount of virtual memory that a process is currently mapping to physical memory. Normally, **RSS** is the more critical value because it represents memory actually allocated and mapped.

```
[user@host ~]$ ps -o pid,vsz,rss,comm -C firefox
  PID     VSZ    RSS COMMAND
 2790 2479004 258712 firefox
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Additionally, processes can *share* physical page frames. For instance, processes that use the same library functions usually share the physical page frames that store the library code. Some interprocess communication mechanisms also make related processes share physical page frames as an efficient way to exchange data.

The system counts a shared page against the **RSS** value of each process that uses it.

Restricting a Process Memory Usage

Starting with Red Hat Enterprise Linux 7, you can restrict the memory an application can consume with cgroups.

For your application, create a systemd drop-in file and define the **`MemoryLimit`** parameter under the **`[Service]`** section. You can use a **`K`**, **`M`**, **`G`**, or **`T`** suffix to specify a memory size in kilobytes, megabytes, gigabytes, or terabytes.

```
[root@host ~]# systemctl set-property sshd.service MemoryLimit=1G
[root@host ~]# systemctl cat sshd.service
...output omitted...
# /etc/systemd/system.control/sshd.service.d/50-MemoryLimit.conf
# This is a drop-in unit file extension, created via "systemctl set-property"
# or an equivalent operation. Do not edit.
[Service]
MemoryLimit=1073741824
[root@host ~]# systemctl daemon-reload
```

Monitoring Page Faults

Because each process maintains its own virtual address space, each process has a *page table* that tracks mappings of its virtual pages to physical pages. Each virtual page has an entry in the table.

When a process accesses a page that the system has not yet mapped to a physical page frame in the table, the kernel generates a page fault event.

The event occurs when the process accesses a page for the first time after allocation. In that situation, the kernel generates a *minor page fault* and allocates a new physical page frame from RAM. A minor page fault causes a small amount of overhead.

However, if the physical page frame is missing and the system needs to retrieve that page of memory from disk, the kernel generates a *major page fault* event. This event occurs for example when the kernel has swapped the page out, or when it needs to load the program and libraries code from disk to memory. A major page fault has a significant impact on performance because the kernel must load the page from disk.

To view minor and major page faults per process, use the **`ps`** command, asking for the **`minflt`** and **`majflt`** columns:

```
[user@host ~]$ ps -o pid,minflt,majflt,comm -C firefox
  PID MINFLT MAJFLT COMMAND
 2790 227093    210 firefox
```

You can also monitor the **`minor-faults`** and **`major-faults`** **`perf`** events for a process.

```
[user@host ~]$ perf stat -e minor-faults,major-faults firefox

 Performance counter stats for 'firefox':

           142,115      minor-faults:u
               136      major-faults:u

      11.779854436 seconds time elapsed

       6.576191000 seconds user
       2.161528000 seconds sys
```

Introducing Page Tables and the TLB

Remember that each process has a page table that tracks mappings of its virtual pages to physical pages. The problem with this mechanism is that if each process allocates one big table in memory to track these mappings, the table itself would be huge. On an x86-64 system, it would require 2^{45} entries, potentially hundreds of GiB *per process*, even if the process only needs a few mappings.

To reduce memory use, the system breaks up the page table into a hierarchy of page directories, which point to smaller page tables, which finally point to page table entries. The system only needs to allocate the page directories or page tables that contain addresses in use by the process, reducing the size of the total table of mappings in memory.

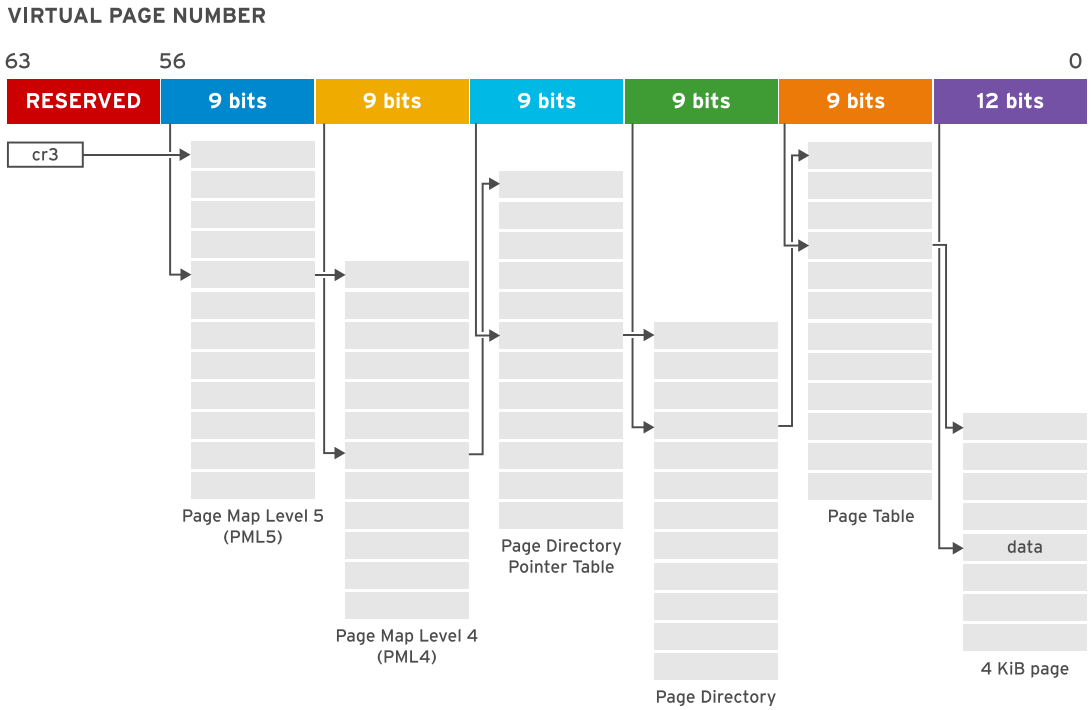


Figure 8.2: Actual page table hierarchy on x86-64

The process of looking up a page mapping on a hierarchical page table can be expensive. Therefore, when the system adds a mapping from a virtual address to a physical address into the page table, it also caches the entry in a dedicated hardware cache called the *Translation Look-*

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

aside Buffer, or *TLB*. This CPU cache speeds up address translation by caching page mappings the process has recently used.

Since programs tend to access their memory sequentially, if a program accesses two addresses on the same page then the second access gets the mapping quickly from the TLB rather than making a long lookup in the page table.

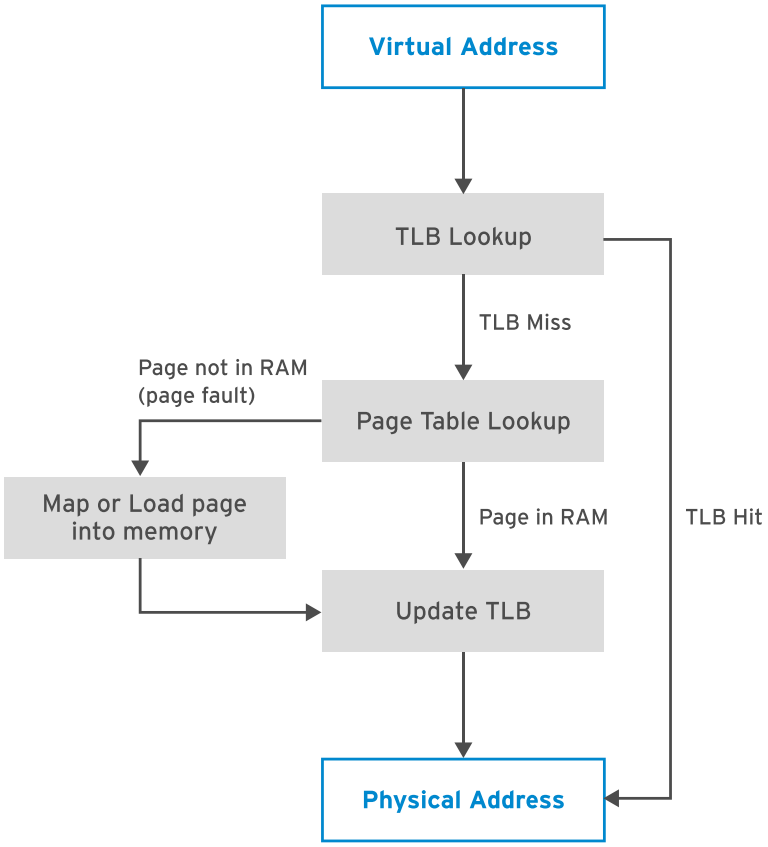


Figure 8.3: TLB flowchart

To inspect the TLB misses in your system, use **`perf`** to monitor the **`dTLB-load-misses`** event. This event is not available for virtual machines.

```
[user@host ~]$ perf stat -e dTLB-load-misses firefox

 Performance counter stats for 'firefox':

         4,583,971      dTLB-load-misses:u

      18.022801842 seconds time elapsed

       2.621693000 seconds user
       0.548131000 seconds sys
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Managing Huge Pages

The number of TLB entries in a processor is fixed and not very big. Processes that access a large amount of memory, such as databases, may incur a considerable number of TLB misses which impacts performance.

In addition to 4 KiB pages, the Linux kernel supports large-sized memory pages through the *huge pages* mechanism. The x86_64 architecture supports multiple page sizes, usually 4 KiB, 2 MiB, and 1 GiB. Red Hat Enterprise Linux uses a default huge page size of 2 MiB, but you can change that size with the **`hugepagesz=`** kernel parameter.

Look in the **`/proc/meminfo`** file to determine the size of the huge pages on a particular system.

```
[user@host ~]$ cat /proc/meminfo
...output omitted...
Hugepagesize:       2048 kB
...output omitted...
```

With huge pages, rather than addressing a standard 4 KiB page, each TLB entry can now address a 2 MiB page. Having fewer TLB entries that point to more memory means that a TLB hit is more likely to occur.

Whenever the system performs a context switch, the kernel must usually flush TLB entries for the process that is being scheduled out.

To allocate huge memory pages, set the total number of required huge pages using the **`vm.nr_hugepages`** **`sysctl`** parameter:

```
[root@host ~]# sysctl -w vm.nr_hugepages=20
vm.nr_hugepages = 20
```

In the previous example, the administrator requests the kernel to allocate 20 huge memory pages. With a huge page size of 2 MiB, the system allocates a total of 40 MiB of huge pages.

The system only allocates the requested pages if there is enough free, contiguous memory to satisfy the request. If the kernel is unable to satisfy the request, it allocates as many huge memory pages as it can out of the requested amount.

To verify the amount of allocated huge pages, check the contents of **`/proc/meminfo`**.

```
[root@host ~]# cat /proc/meminfo
...output omitted...
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
...output omitted...
```

Because the kernel must allocate huge pages as contiguous memory, it can be challenging to allocate them during runtime, as memory becomes more fragmented over time. In that situation, define the **`vm.nr_hugepages`** parameter in **`/etc/sysctl.conf`** and reboot the system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

> **Note**
> By allocating memory as huge pages, only huge page-aware applications may utilize the memory. The kernel removes the huge page memory from the generally available system memory, and most processes cannot use it.

In order to use the huge pages, processes must request them using either the **`mmap()`** system call or the **`shmat()`** and **`shmget()`** system calls. If a process uses the **`mmap()`** system call, then you need to mount the **`hugetlbfs`** file system.

```
[root@host ~]# mkdir /largepage
[root@host ~]# mount -t hugetlbfs none /largepage
```

On Red Hat Enterprise Linux 8, systemd automatically mounts this file system during boot.

Hugepages on NUMA Systems

On NUMA systems, the kernel divides the huge pages across all NUMA nodes uniformly.

When an application such as a database needs all its huge pages on only one node, where the process is running, use the **`/sys/devices/system/node/node`*N*`/hugepages/hugepages-2048kB/nr_hugepages`** tunable instead of the global **`vm.nr_hugepages sysctl`** parameter.

For example, to allocate 20 pages on the NUMA node 2, use the following command.

```
[root@host ~]# echo 20 > \
/sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
```

You can review on which node the kernel has allocated the huge pages with the following **`numastat`** command (from the *numactl* package).

```
[root@host ~]# numastat -cm

Per-node system memory usage (in MBs):
                 Node 0 Node 1 Node 2 Node 3  Total
                 ------ ------ ------ ------ ------
...output omitted...
AnonHugePages        32     18      2      0     52
HugePages_Total       0      0     40      0     40
HugePages_Free        0      0     40      0     40
HugePages_Surp        0      0      0      0      0
```

The **`numastat`** command displays the values in megabytes: 20 huge pages * 2 MiB page size = 40 MiB.

A later section introduces the NUMA topology in more details.

Using Transparent Huge Pages

Since RHELinux 6.2, the kernel enables *Transparent Huge Pages (THP)* by default.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

THP differs from the standard huge pages in that the kernel allocates and manages them automatically. The system can also swap them out of memory, unlike standard huge pages.

The THP tunables are in the **/sys** file system in the **/sys/kernel/mm/transparent_hugepage/** directory. The value for the **/sys/kernel/mm/transparent_hugepage/enabled** tunable can be one of the following:

- **always** – enable THP
- **never** – disable THP
- **madvise** – disable automatic THP allocation. Processes that wish to use THP must inform the kernel through the **madvise()** system call.

The **khugepaged** daemon automatically starts when **transparent_hugepage/enabled** is set to **always** or **madvise** and it automatically shuts down when the parameter is set to **never**. The **khugepaged** daemon scans the memory, defragments the memory, and merges standard pages into huge pages.

To check the THP usage, execute the following command:

```
[root@host ~]# grep AnonHugePages /proc/meminfo
AnonHugePages:    75776 kB
```

To disable or enable THP in a tuned profile, add the **transparent_hugepages** parameter under the **[vm]** section of the **tuned.conf** configuration file.

```
[vm]
transparent_hugepages = never
```

You can also append the **transparent_hugepage=** parameter to the kernel command line.

Transparent huge pages are not recommended for use with database workloads; instead, allocate huge pages with the **sysctl vm.nr_hugepages** parameter.

References

ps(1), **top**(1), **systemd.resource-control**(5), and **perf-stat**(1) man pages

/usr/share/doc/kernel-doc-*/Documentation/x86/x86_64/5level-paging.txt

/usr/share/doc/kernel-doc-*/Documentation/admin-guide/mm/hugetlbpage.rst

/usr/share/doc/kernel-doc-*/Documentation/admin-guide/mm/transhuge.rst

How to disable transparent hugepage (THP) on Red Hat Enterprise Linux 8?
https://access.redhat.com/solutions/3799821

What every programmer should know about memory, Part 1
https://lwn.net/Articles/250967/

Guided Exercise

Configuring Memory Architecture

In this exercise, you will view and configure system memory settings.

Outcomes

You should be able to inspect your processes memory usage and allocate transparent huge pages as well as preallocated huge pages.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab memtuning-arch start** command. The command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the **bigmem** program that you use to allocate huge pages. This program has been developed specifically for this course.

```
[student@workstation ~]$ lab memtuning-arch start
```

1. On **workstation**, open two terminal windows. On each terminal, use **ssh** to log in to **servera** as **student**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$
```

2. In the first terminal, execute the following **watch** command to begin watching for the **bigmem** processes.

```
[student@servera ~]$ watch ps -o vsz,rss,comm -C bigmem
```

Running this command, you can monitor memory usage, virtual (**vsz**) and physical (**rss**) of the **bigmem** program. No **bigmem** process should be running now.

3. In the second terminal, run the **bigmem** command to allocate 512 MiB of memory.

3.1. Run **bigmem** with the requested memory size as an argument.

```
[student@servera ~]$ bigmem 512M
```

The **bigmem** program allocates, then uses the requested memory.

In the first terminal, observe that the **VSZ** and **RSS** columns show **bigmem** using 512 MiB of virtual and physical memory. The **ps** command displays **VSZ** and **RSS** in kilobytes.

3.2. Press **Enter** to exit **bigmem**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
...output omitted...
Press <Enter> to exit   Enter
```

4. Run **bigmem** again to allocate 512 MiB of virtual memory. Use the **--virtual** (or **-v**) option to only allocate virtual memory without using it.

   4.1. Run **bigmem** with the **--virtual** option and the requested memory size as an argument.

```
[student@servera ~]$ bigmem --virtual 512M
```

   In the first terminal, **bigmem** only consumes virtual memory (**VSZ**).

   4.2. Press **Enter** to exit **bigmem**.

```
...output omitted...
Press <Enter> to exit   Enter
```

5. Exit **watch** in the first terminal and list the memory consumption of all processes running on your system.

   5.1. In the first terminal, press **Ctrl**+**C** to exit **watch** and return to the shell prompt.

   5.2. Run the **ps** command to list all the processes running on your system.

```
[student@servera ~]$ ps -eo vsz,rss,comm
  RSS    VSZ COMMAND
...output omitted...
 93324  8092 systemd-journal
109424 11548 systemd-udevd
 66424  5132 rpcbind
149320  2052 auditd
 47832  2052 sedispatch
 16868  2128 mcelog
1911236 23684 polkitd
 26064  4136 smartd
100384  3164 irqbalance
 63464  5092 dbus-daemon
394488 10796 sssd
...output omitted...
```

   Generally, processes consume far less physical memory (**RSS**) than requested (**VSZ**).

6. In the second part of this exercise, inspect and use huge pages on your system.

   In the first terminal, ensure transparent huge pages are enabled.

```
[student@servera ~]$ cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

7. View the current number of anonymous, or transparent, huge pages of memory. The value in your system will be different.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@servera ~]$ grep AnonHugePages /proc/meminfo
AnonHugePages:     22528 kB
```

8. In the second terminal, run **bigmem** to allocate 512 MiB of memory. Use the **--transparent** (or **-T**) option to allocate big chunks of memory assigned to transparent huge pages.

```
[student@servera ~]$ bigmem --transparent 512M
```

9. In the first terminal, review the current number of anonymous huge pages of memory. The value in your system will be different and you may have to wait a few seconds and run the following command several times for the system to report the updated value.

```
[student@servera ~]$ grep AnonHugePages /proc/meminfo
AnonHugePages:    542720 kB
```

The number has increased. The **bigmem** process is using the additional huge pages.

10. Press **Enter** to exit **bigmem**.

```
...output omitted...
Press <Enter> to exit   Enter
```

11. Compare the number of minor page faults when **bigmen** uses regular pages and when it uses huge pages.

11.1. In the first terminal, use **sudo -i** to switch to **root**.

```
[student@servera ~]$ sudo -i
[root@servera ~]#
```

11.2. Install the *perf* package.

```
[root@servera ~]# yum install perf
...output omitted...
Complete!
```

11.3. Retrieve the name of the event for monitoring minor page faults.

```
[root@servera ~]# perf list | grep minor
  minor-faults                       [Software event]
```

11.4. Run **perf stat** to monitor **bigmem** when it uses regular pages. Press **Enter** to exit the program when prompted.

```
[root@servera ~]# perf stat -e minor-faults bigmem 512M
...output omitted...
Press <Enter> to exit   Enter
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
 Performance counter stats for 'bigmem 512M':

           131,634      minor-faults:u
...output omitted...
```

Note the number of minor page faults. The value will be different on your system.

11.5. Run **bigmem** again, but this time allocate transparent huge pages. Press **Enter** to exit the program when prompted.

```
[root@servera ~]# perf stat -e minor-faults bigmem --transparent 512M
...output omitted...
Press <Enter> to exit   Enter


 Performance counter stats for 'bigmem --transparent 512M':

            33,072      minor-faults:u
...output omitted...
```

When **bigmem** uses huge pages, the number of minor page faults is lower, because the process needs fewer pages to allocate the requested memory.

12. In this part of the exercise, allocate standard huge pages and test your configuration.

First, review the standard huge pages memory information.

```
[root@servera ~]# grep HugePages /proc/meminfo
AnonHugePages:     22528 kB
ShmemHugePages:        0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
```

No standard huge pages are available.

13. Use the **systcl** command to enable 512 huge pages.

```
[root@servera ~]# sysctl -w vm.nr_hugepages=512
vm.nr_hugepages = 512
```

14. View the huge pages memory information again.

```
[root@servera ~]# grep HugePages /proc/meminfo
AnonHugePages:     22528 kB
ShmemHugePages:        0 kB
HugePages_Total:     512
HugePages_Free:      512
HugePages_Rsvd:        0
HugePages_Surp:        0
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

15. In the second terminal, run **bigmem** to allocate 256 MiB of shared memory backed by huge pages. Use the **--hugepages** (or **-H**) option to allocate huge pages using the **shmat()** and **shmget()** system calls.

```
[student@servera ~]$ bigmem --hugepages 256M
```

16. In the first terminal, view the huge pages memory information again.

```
[root@servera ~]# grep HugePages /proc/meminfo
AnonHugePages:     22528 kB
ShmemHugePages:        0 kB
HugePages_Total:     512
HugePages_Free:      384
HugePages_Rsvd:        0
HugePages_Surp:        0
```

Only 384 huge pages are available now. The **bigmem** process is consuming 128 huge pages (512 - 384 = 128).

17. Press **Enter** to exit **bigmem**.

```
...output omitted...
Press <Enter> to exit   Enter
```

18. Log off from **servera** in the two terminals.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab memtuning-arch finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab memtuning-arch finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring Memory Paging and Reclamation

Objectives

After completing this section, you should be able to describe system paging behavior and configure paging for server workload requirements.

Explaining the System Memory and Page Cache

The kernel uses the system memory for two primary purposes:

Page cache
: The kernel uses most unallocated memory as a cache to store data being read from or written to disk. The next time that a process needs that data, the system fetches it from RAM rather than the disk. This caching mechanism causes a significant performance improvement in general because storage is generally much slower than physical memory.

Anonymous **pages**
: Anonymous pages are not associated with data on disk. Anonymous pages are the pages that processes allocate and use to store their working data.

You can inspect the memory usage with the **`free`** and the **`vmstat`** commands.

```
[user@host ~]$ free -m
              total        used        free      shared  buff/cache   available
Mem:         120556       11788         353        1611      108414      106143
Swap:          4095           0        4095
```

In this output, notice that even though the system has 120 GiB of RAM, only 353 MiB are free. The page cache (in the **`buff/cache`** column) consumes most of the memory, probably because the system has recently performed a lot of storage I/O.

If necessary, the system has 104 GiB (106143 MiB) of free memory (from the **`available`** column) if it releases data from its caches. However, dropping the caches harms I/O performance. You can monitor memory usage in real time with the **`vmstat`** command.

```
[user@host ~]$ vmstat --unit M 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0    362 105754   2649    0    0     4     0    1    0  0  0 100  0  0
 0  1      0    339 105777   2649    0    0 143360     0 1162  743  0  0 99  0  0
 0  1      0    331 105785   2649    0    0 143360    24 1047  770  0  0 99  0  0
 0  1      0    342 105774   2650    0    0 147456     4 1227  871  0  0 99  0  0
 0  1      0    353 105763   2650    0    0 147456     0 1240 1075  0  0 99  0  0
...output omitted...
```

In the **`vmstat`** output, the size of the page cache is the sum of the **`buff`** and **`cache`** columns. The **`buff`** column represents the buffers, which are part of the page cache. These buffers store blocks of data directly read from a block device, without going through a file system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Bypassing the Page Cache

When accessing data on disk, some applications use *direct I/O*, that bypasses the page cache. For example, some database systems provide their own internal process cache and use direct I/O for disk access. Some use direct I/O to access dedicated raw block devices. Others perform direct I/O through a file configured with direct I/O attributes.

For applications repetitively accessing the same files or the same parts of files, such as file servers, the page cache provides a significant performance improvement. Applications that read a file only once sequentially, such as a video player on your laptop, get little value from the page cache.

Reclaiming Anonymous Pages

When the system is under memory pressure, the kernel can choose to either reclaim anonymous pages or pages from the page cache. Anonymous pages, which are not backed by files, are moved to the swap area. Having swap increases the amount of effective memory. As free memory drops, the kernel can *page out* memory pages that processes are not actively using. These reclaimed pages are then used for other requests.

When a process needs a page in the swap area, a major page fault, the kernel *pages in* the page from disk and assigns it back to the process in memory. The swap area is slower since most storage devices are very slow compared to memory. Excessive paging can slow down a system significantly.

The **`vmstat`** utility provides information on whether a system is paging to swap or not. The critical columns in the output of **`vmstat`** are **`si`** and **`so`**, pages swapped in per second and pages swapped out per second.

```
[user@host ~]$ vmstat --unit M 5
procs -----------memory--------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free  buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0   1484 104355     0    688    0    0     4     0    0    0  0  0 100  0  0
 1  0   1484   9735     0    695    0    0     6     0 1546  405  0  1 99  0  0
 1  0   1483   7614     0    695    0    0    13     0 1372  365  0  1 99  0  0
 1  0   1483   5493     0    695    0    0     0     0 1414  394  0  1 99  0  0
 1  0   1483   3391     0    695    0    0    13    12 1472  408  0  1 99  0  0
 0  6   1564    272     0    658    0   16     6 16295 2238  376  0  0 99  0  0
 0  6   2156    298     0    535    0  118   435 120484 27539  746  0  0 97  3  0
 0  9   2755    271     0    462    0  119   273 122048 25179  763  0  0 97  3  0
 0  9   3363    275     0    462    0  121   259 124290 22535  717  0  0 96  3  0
 0  7   3903    278     0    458    0  108   226 110926 27968  775  0  0 96  3  0
...output omitted...
```

The previous output shows a system that is rapidly swapping pages out to disk. In the latest lines, values in the **`so`** column show that the system is swapping pages out rapidly.

> **Important**
> The **`swpd`** value represents the used swap space amount and is not significant if enough free swap space exists. Having pages be located in the swap area does not impact performance, and may help by moving unneeded pages out of main memory, freeing memory for other use. Performance is impacted by frequent paging activity in and out of swap areas, as reported by the **`si`** and **`so`** metrics.

Introducing Process Working Set

A process may allocate and fill a large amount of memory at startup, but only be using a portion of that memory at any time. This set of active physical memory pages is called the process' *working set*. The working set will increase and decrease during the process lifetime.

The system tracks those active pages at a global level. Under memory pressure, the kernel uses that information to choose which memory to page, while preserving the active pages.

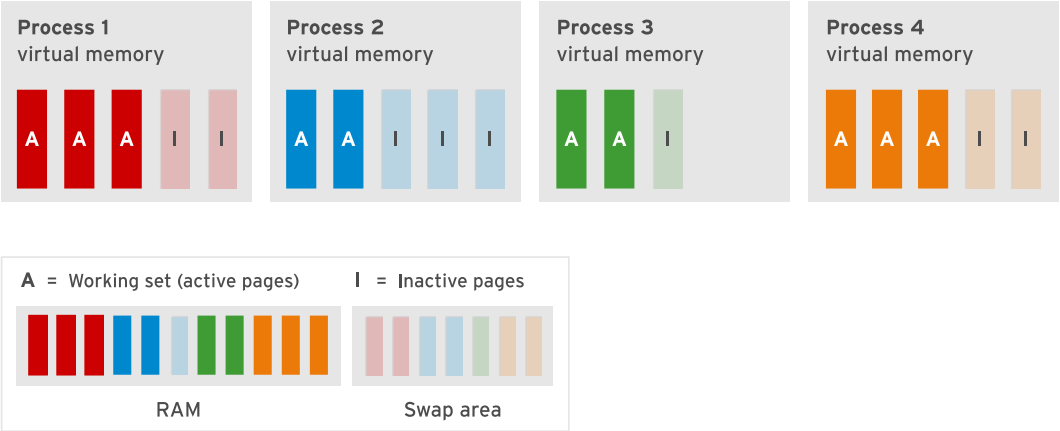


Figure 8.4: Memory pressure and working set

The diagram above shows that the available RAM is running low. The system has already paged out some of the processes anonymous pages in the swap area. Notice however that all the pages from the processes working set are still in RAM. In that situation, the running applications do not incur performance degradation, at least as long as their working set does not change and does not need a page from the swap area.

Creating Swap Space Remains Recommended

Administrators should size the swap area based on the memory workload on the system. Application vendors sometimes provide recommendations on that subject. The following table provides some guidance based on the total amount of physical memory.

System can have multiple partitions, which can be prioritized in **`/etc/fstab`** configuration. SSDs are preferred as swap due to their increased throughput and decreased latencies compared to rotational media. Swap space is necessary in scenarios where direct I/O requires large areas of contiguous memory, as opposed to being memory mapped. The system can handle contiguous memory requirements, but only if enough swap exists to move file-backed and anonymous pages out to create sufficient contiguous memory space.

Basic Guidance on Swap Area Sizing

RAM	Swap Space	With Hibernation
2 GiB or less	2 time the RAM	3 times the RAM
Between 2 GiB and 8 GiB	Equal to RAM	2 time the RAM
Between 8 GiB and 64 GiB	At least 4 GiB	1.5 times the RAM
More than 64 GiB	At least 4 GiB	Not recommended

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Reclaiming Page Cache and Tuning Swappiness

When the kernel needs to free a memory page, it has two choices. It can page out a process anonymous page to the swap area, or it can drop a page from the page cache. Because pages in the page cache store data from disk, the system can write the data back to disk, if the data has changed, and reclaim the page. By tuning the **`vm.swappiness`** **`sysctl`** parameter, you can influence that kernel choice. The parameter accepts a value between 0 and 100.

Set the **`vm.swappiness`** parameter to **`100`**, and the system almost always prefers to page anonymous pages out to the swap area over reclaiming pages from the page cache. With this value, the kernel preserves the page cache, which can significantly increase performance for an I/O-heavy workload. In a file server environment, for example, it is better to keep file data in the page cache than dropping those data and having to reread them from disk later.

Setting the **`vm.swappiness`** parameter to **`1`**, on the other hand, forces the system to swap as little as possible. This tuning might result in a more responsive system, but at the cost of file system performance. Applications that do not use the page cache usually benefit from this setting. For example, a database system that needs to preserve its tables and data structures in memory benefits from a low **`vm.swappiness`** value. Some database software makers have specific values that should be used for this tunable.

> **Note**
> Beginning in RHEL 6.4, setting **`vm.swappiness`** to **`0`** causes the machine to use the swap space only as the last resort.

With cgroups, you can set different swappiness values for different cgroups using the **`memory.swappiness`** tunable of the **`memory`** cgroup controller.

Using SSD as Swap Area

SSD devices are significantly faster than rotating disks. Reads from swap are random operations because the system cannot predict when an application will need its paged out pages. With rotating disks, random operations are expensive, but not with SSDs.

When using rotating disks, it is common to tune the system to avoid swapping at all cost. By allowing the system to use a swap area on SSD, the kernel can better balance between the page cache and the anonymous pages without too much performance loss.

This configuration is particularly useful when the page cache comes from rotating disks. It is faster to page out and page in some anonymous pages to an SSD swap area than dropping the page cache and having to reread data from the rotating disks.

Therefore, when the system uses SSDs for swap areas, increase the value of the **`vm.swappiness`** parameter for the kernel to manage its memory better and move the inactive pages to swap.

With multiple swap areas, the system uses them in series. The kernel uses the first activated swap area until it is full, then it starts using the next one. However, you can set the mount option **`pri=`*`value`*** in the **`/etc/fstab`** file to specify the priority of each space. The system fills up first the swap area with the highest priority before moving on to one with a lower priority. This way you can prioritize a faster disk over a slower one.

Tuning Dirty Pages Cleaning

A physical page can be in different states:

- **`Free`** – The page is available for immediate allocation.
- **`Active`** – The page is in active use and not a candidate for being freed.
- **`Inactive clean`** – The page is not in active use, and its content matches the content on disk.
- **`Inactive dirty`** – The page is not in active use, but the page content has been modified since being read from disk and has not yet been written back.

To review the amount of active, inactive, and dirty memory, inspect the **`/proc/meminfo`** file.

```
[user@host ~]$ cat /proc/meminfo
MemTotal:       123449580 kB
MemFree:        121526520 kB
MemAvailable:   121336512 kB
Buffers:               0 kB
Cached:           246468 kB
SwapCached:         9400 kB
Active:            87320 kB
Inactive:         230776 kB
Active(anon):      52576 kB
Inactive(anon):    78848 kB
Active(file):      34744 kB
Inactive(file):   151928 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:       4194300 kB
SwapFree:        2458084 kB
Dirty:                 8 kB
Writeback:             0 kB
AnonPages:         64860 kB
Mapped:            64792 kB
...output omitted...
```

For a per-process view, use the **`/proc/`*`PID`*`/smaps`** file. This file details every memory segment allocated to a process, including the sizes of clean and dirty memory. To parse the file, write a small **`awk`** program like the following:

```
[user@host ~]$ cat /proc/$$/smaps | awk '
/Shared_Clean/ {SHCL+=$2}
/Shared_Dirty/ {SHDT+=$2}
/Private_Clean/ {PRCL+=$2}
/Private_Dirty/ {PRDT+=$2}
END {
  print "Total Clean:", SHCL + PRCL
  print "Total Dirty:", SHDT + PRDT
}'
Total Clean: 3480
Total Dirty: 1624
```

Reclaiming pages in the inactive clean state is cheap because the data is already on disk. However, reclaiming pages in the inactive dirty state first requires to write the data back to disk.

Also, because memory is volatile (contents are lost when power fails), the system writes out dirty pages to disk at regular intervals to prevent data loss in case of a power failure.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The system handles writing out dirty pages through the **`flush-`*`MAJOR:MINOR`*** kernel threads. The kernel only starts those threads when needed, and it starts one thread per block device, identified by its major and minor number.

A set of **`sysctl`** parameters control when the flush threads start writing data to disk.

- **`vm.dirty_expire_centisecs`** – How old (in 1/100ths of a second) dirty data must be before the kernel considers writing the page out to disk. This parameter stops the kernel from writing out the same page multiple times in quick succession just because a process keeps modifying the page contents.
- **`vm.dirty_writeback_centisecs`** – How often (in 1/100ths of a second) the kernel wakes up the flushing threads to write out data. Setting this parameter to **`0`** disables periodic write back completely.
- **`vm.dirty_background_ratio`** – The percentage of total system memory being dirty at which the kernel starts writing out data in the background.
- **`vm.dirty_ratio`** – The percentage of total system memory being dirty at which a process generating writes blocks and writes out dirty pages.

There are two more parameters, **`vm.dirty_background_bytes`** and **`vm.dirty_bytes`**, that you can use instead of the **`vm.dirty_background_ratio`** and **`vm.dirty_ratio`** settings. The value of these new parameters is in bytes instead of a percentage. When you set the **`vm.dirty_{background_,}bytes`** parameter, the system automatically sets the accompanying ratio to **`0`** and vice versa.

Setting lower ratios results in more frequent but shorter writes, suited for an interactive system. Setting higher ratios results in fewer but bigger writes, causing less overhead in total but possibly resulting in higher response times for interactive applications.

Most tuned profiles modify at least one of the previous settings. For example, the **`oracle`** tuned profile, from the *tuned-profiles-oracle* package, makes the following adjustment:

```
[user@host ~]$ cat /usr/lib/tuned/oracle/tuned.conf
#
# tuned configuration
#

[main]
summary=Optimize for Oracle RDBMS
include=throughput-performance

[sysctl]
vm.swappiness = 10
vm.dirty_background_ratio = 3
vm.dirty_ratio = 40
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
...output omitted...
```

With this configuration, the machine starts writing out dirty pages from the page cache if they are over 3% of system memory, but does not force writes until dirty pages are over 40% of system memory.

Dirty pages that have not been modified in the last 5 seconds (500 centiseconds) become candidates for flushing to the I/O subsystem.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Every second (100 centiseconds), the system checks to see if pages have expired and need to be written to disk.

With these settings, the system performs small but frequent writes.

Handling Out-of-memory Events

When a minor page fault happens but there are no free pages available, the kernel tries to reclaim memory to satisfy the request. If it cannot reclaim sufficient memory, an out-of-memory condition occurs. This condition often happens when there is no more memory and swap space available.

By default, a kernel subsystem called the *Out-of-Memory killer (OOM killer)* selects and kills one process to free memory so that the system can satisfy the request. If the condition is still present, the system triggers the OOM killer again to kill a second process and so on.

Because killing some processes of an application this way may damage or corrupt application data, you may set the **`vm.panic_on_oom`** **`sysctl`** tunable to **`1`** instead of **`0`**. With this value, the kernel panics instead of killing processes.

> **Note**
> Once the system reaches a point where it has an out-of-memory condition, there are not many reasonable options for recovery. Killing a process to free memory or giving up and killing the system are the possible choices. In general, it is best to avoid allowing the system to get into an out-of-memory condition if at all possible.

Every process has a *badness* score that you can view in the **`/proc/`*`PID`*`/oom_score`** file. The kernel maintains that score and uses it to determine which process the OOM killer should kill. The higher the score, the more likely the OOM killer will kill the process. The kernel itself and PID 1 (**`systemd`**) are immune from the OOM killer.

The system uses a number of factors to calculate this score:

- The virtual memory size.
- The cumulative virtual memory of any children the process has.
- Nice values, positive nice values give a higher score.
- Total run time, a longer total run time reduces the score.
- The running user, **`root`** user processes get slight protection.
- Direct hardware access by the process, reduces the score as well.

You can manually adjust that score with the **`/proc/`*`PID`*`/oom_score_adj`** tunable. The tunable takes a value between -1000 and 1000 where 0 is the default. A value of -1000 means immunity. The OOM killer never kills the process. The kernel automatically adds any other value to the process **`oom_score`** value.

> **Note**
> Another deprecated tunable is still available: **`/proc/`*`PID`*`/oom_adj`**. This tunable works in a similar fashion but accepts a value between -17 (immunity) and 15.

The system can be in an out-of-memory condition while **`free`** still reports memory available. This can occur when a memory limit is set in a cgroup, and the processes in the group consume all that memory. On a NUMA system, when no more memory is available in a node and a process requests memory from that node, the system also triggers the OOM killer.

A later section introduces the NUMA topology in more details.

Adjusting the Out-of-memory Score with Systemd

To adjust the score of an application at startup, create a systemd drop-in file and define the **`OOMScoreAdjust`** parameter under the **`[Service]`** section.

```
[root@host ~]# mkdir /etc/systemd/system/sssd.service.d
[root@host ~]# vi /etc/systemd/system/sssd.service.d/10-OOMscore.conf
[Service]
OOMScoreAdjust=-1000
[root@host ~]# systemctl daemon-reload
[root@host ~]# systemctl restart sssd
```

The **`OOMScoreAdjust`** parameter accepts a value between -1000 (immunity) and 1000.

References

`free`(1), **`vmstat`**(8), **`systemd.exec`**(5), and **`swapon`**(8) man pages

`/usr/share/doc/kernel-doc-*/Documentation/sysctl/vm.txt`

Knowledgebase: What is the recommended swap size for Red Hat platforms?
https://access.redhat.com/solutions/15244

For more information, refer to the *Configuring System Memory Capacity* section in the *Red Hat Enterprise Linux 7 Performance Tuning Guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index#sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Configuration_tools-Configuring_system_memory_capacity

Page Frame Reclamation
https://www.kernel.org/doc/gorman/html/understand/understand013.html

Reconsidering swapping
https://lwn.net/Articles/690079/

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring Memory Paging and Reclamation

In this exercise, you will tune memory paging and reclamation parameters.

Outcomes

You should be able to observe and tune how the kernel handles dirty pages, and tune the OOM killer to protect an application process.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab memtuning-paging start** command. The command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the **bigmem** program that you use to allocate memory.

```
[student@workstation ~]$ lab memtuning-paging start
```

1. On **workstation** open four terminal windows. In each terminal, use **ssh** to log in to **servera** as **student**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$
```

2. In the first terminal, monitor the dirty pages on your system. Execute the following loop and keep it running. For your convenience, you can copy and paste those commands from the **/home/student/watch_dirty.sh** script.

```
[student@servera ~]$ while :
do
   grep Dirty /proc/meminfo
   sleep 1
done
Dirty:                24 kB
Dirty:                24 kB
Dirty:                 0 kB
Dirty:                 0 kB
...output omitted...
```

3. In the second terminal, use **sudo -i** to switch to **root**.

```
[student@servera ~]$ sudo -i
[root@servera ~]#
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

4. Review the value of the **vm.dirty_background_ratio** tunable.

```
[root@servera ~]# sysctl vm.dirty_background_ratio
vm.dirty_background_ratio = 10
```

With this value, the kernel starts writing out data to disk when dirty pages consume 10% of total memory. Use the **free** command to retrieve the total memory.

```
[root@servera ~]# free
              total        used        free      shared  buff/cache   available
Mem:        1872936      177632     1551744       16804      143560     1540312
Swap:             0           0           0
```

The kernel will start writing out dirty pages when 187293 KiB (10% of 1872936 KiB) of memory is dirty.

5. Start writing random data to the available **/dev/vdc** device.

```
[root@servera ~]# dd if=/dev/urandom of=/dev/vdc
```

In the first terminal, observe that the volume of dirty pages increases then stabilizes, and does exceed 10% of total memory, because the kernel started writing out dirty pages.

Press **Ctrl**+**C** to exit **dd** and return to the shell prompt.

6. Configure the system to start writing out dirty pages when 20% of the memory is dirty.

6.1. Set the value of the **vm.dirty_background_ratio** tunable to **20**.

```
[root@servera ~]# sysctl -w vm.dirty_background_ratio=20
vm.dirty_background_ratio=20
```

6.2. To verify, write random data to **/dev/vdc**. Observe that the volume of dirty pages does not exceed 20% (374587 KiB) of total memory.

```
[root@servera ~]# dd if=/dev/urandom of=/dev/vdc
```

Press **Ctrl**+**C** to exit **dd** and return to the shell prompt. In the first terminal, press **Ctrl**+**C** to exit the loop and return to the shell prompt.

7. In the second part of this exercise, tune the OOM killer to protect a process.

In the first terminal, execute a **watch** command to begin watching for **bigmem** processes.

```
[student@servera ~]$ watch ps -o pid,rss,args -C bigmem
```

No **bigmem** process should be running yet.

8. In the third terminal, start a **bigmem** instance claiming 1 GiB of memory. Keep it running.

```
[student@servera ~]$ bigmem 1G
```

Note the PID of the new **bigmem** process appears in the first terminal's **watch** output.

9. In the second terminal, configure that **bigmem** to never be killed by the OOM killer.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# echo -1000 > /proc/PID/oom_score_adj
```

10. In the fourth terminal, start another **bigmem** instance claiming just 400 MiB of memory. Keep it running.

```
[student@servera ~]$ bigmem 400M
```

Note the PID of this new **bigmem** process in the first terminal's **watch** output.

11. In the second terminal, configure the **bigmem** process so that the OOM killer will prefer killing it over other processes.

```
[root@servera ~]# echo 1000 > /proc/PID/oom_score_adj
```

12. Start another instance of **bigmem**. Make it claim 410 MiB of memory. Monitor the first terminal's **watch** output.

```
[root@servera ~]# bigmem 410M
```

As expected, the OOM killer kills the second **bigmem** process.

13. In the second terminal, press **Enter** to exit the **bigmem** application.

```
Press <Enter> to exit   Enter
```

14. Review the system log.

```
[root@servera ~]# journalctl | grep "Out of memory"
Jul 16 03:27:34 servera.lab.example.com kernel: Out of memory: Kill process 3776
 (bigmem) score 220 or sacrifice child
```

15. Stop the **bigmem** and the **watch** processes. Log off from **servera** in the four terminals.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab memtuning-paging finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab memtuning-paging finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring NUMA Topology

Objectives

After completing this section, you should be able to describe the non-uniform memory architecture and the utilities for viewing and configuring NUMA topology.

Introducing the Non-Uniform Memory Access architecture

On older multiprocessor systems, all memory is equally accessible by the CPUs. That means that access times for all memory addresses are the same, no matter which CPU is performing the operation. A shared front-side bus (FSB) connects the CPUs to main memory. This memory architecture is called *Uniform Memory Access (UMA)*.

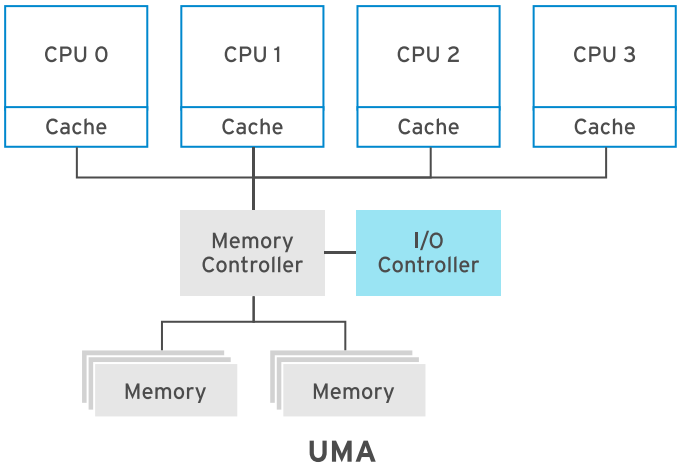


Figure 8.5: UMA architecture

On more modern multiprocessor systems, this is no longer the case. On a *Non-uniform Memory Access (NUMA)* system, each CPU package (or physical CPU) has direct access to a set of memory banks, and high-speed links *interconnect* the CPUs. The whole memory is still accessible from any CPU, but accessing the local memory is faster than accessing remote memory through the interconnect. In that sense, the memory access time is not uniform.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

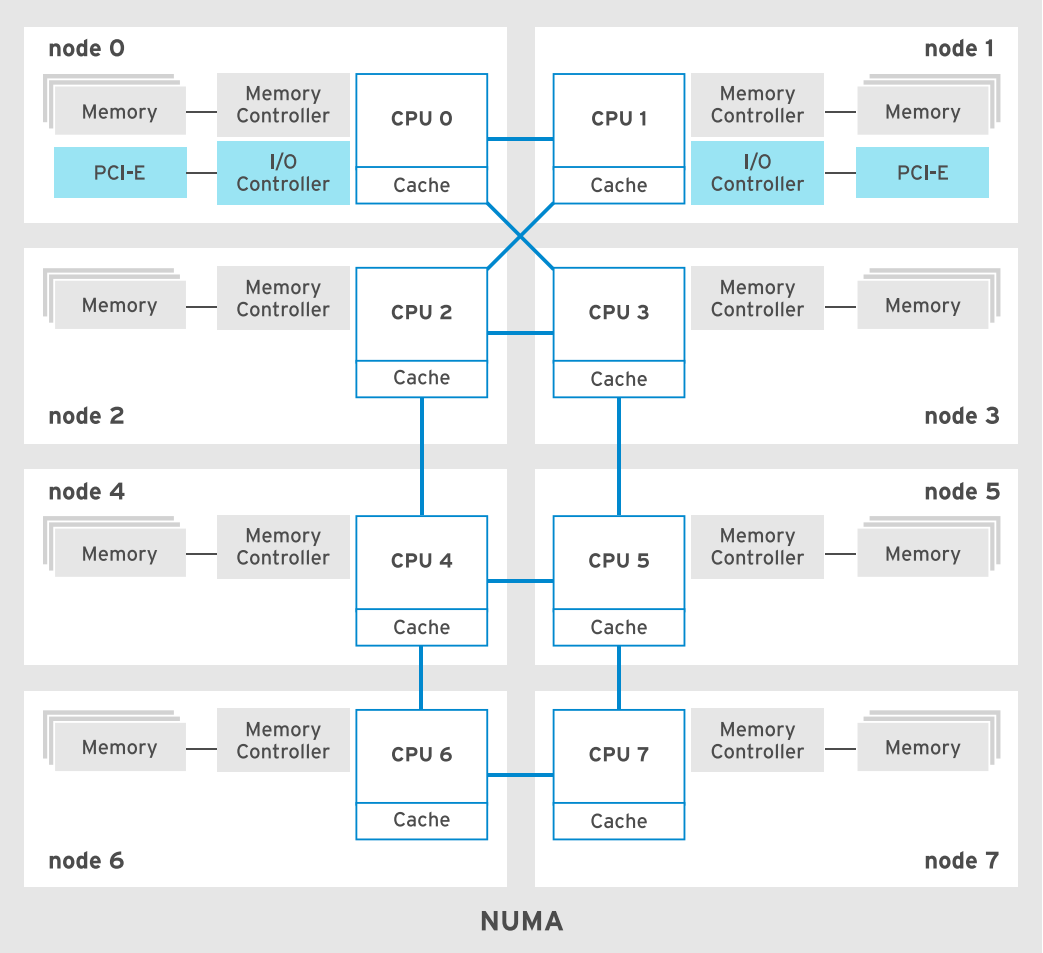


Figure 8.6: 8-nodes NUMA topology

To manage the NUMA topology, Linux divides the system in *nodes*. A node groups a physical CPU and its local memory. The diagram above shows a system with eight physical CPUs and eight NUMA nodes.

The CPUs are not always directly interconnected to each other. In the diagram, notice that CPU 0 has a direct connection with CPU 3, but not with CPU 5. For CPU 0 to access memory on node 5, it has to go through two links, slowing down further the memory access time.

In a NUMA topology, I/O controllers have a direct connection with some of the CPUs. An application dealing with a specific PCI-e device, such as a high-speed network card for example, usually benefits from running on the same NUMA node as the associated device.

Managing Non-Uniform Memory Access

Red Hat Enterprise Linux 7 introduced the *NUMA-Aware Scheduling and Memory Allocation* feature. This automatic NUMA affinity management feature monitors the NUMA topology and resource usage to dynamically improve NUMA resource allocation and management. The algorithm regularly inspects the system and relocates processes, or their memory, to improve locality.

On Red Hat Enterprise Linux 6, the **numad** user-space daemon provides the same functionality. Although the *numad* package is still available in Red Hat Enterprise Linux 7 and 8, you must disable the automatic NUMA balancing algorithm to use it. To do so, set the **kernel.numa_balancing sysctl** tunable to **0**.

Some workloads work better with automatic NUMA balancing, others with the **numad** service. The benefits of those mechanisms are most significant on systems with long-running processes that consume the system resources. They are unlikely to improve performance on systems where processes are not resource-intensive and only run for a few seconds.

Inspecting the NUMA Topology

To see how a system is divided into nodes, use the **numactl --hardware** command from the *numactl* package.

```
[root@host ~]# numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 96 97
 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
 119
node 0 size: 32641 MB
node 0 free: 30588 MB
node 1 cpus: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
 47 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
 139 140 141 142 143
node 1 size: 32768 MB
node 1 free: 30648 MB
node 2 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
 71 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
 163 164 165 166 167
node 2 size: 32768 MB
node 2 free: 31627 MB
node 3 cpus: 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94
 95 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186
 187 188 189 190 191
node 3 size: 24576 MB
node 3 free: 23090 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

This command shows which processors are in which nodes, and the amount of memory that is available and that is in use per node. At the bottom of the output, the matrix indicates the relative weight of memory access between nodes, where lower is faster.

The **lstopo** command, from the *hwloc-gui* package, provides a graphical view of the topology.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

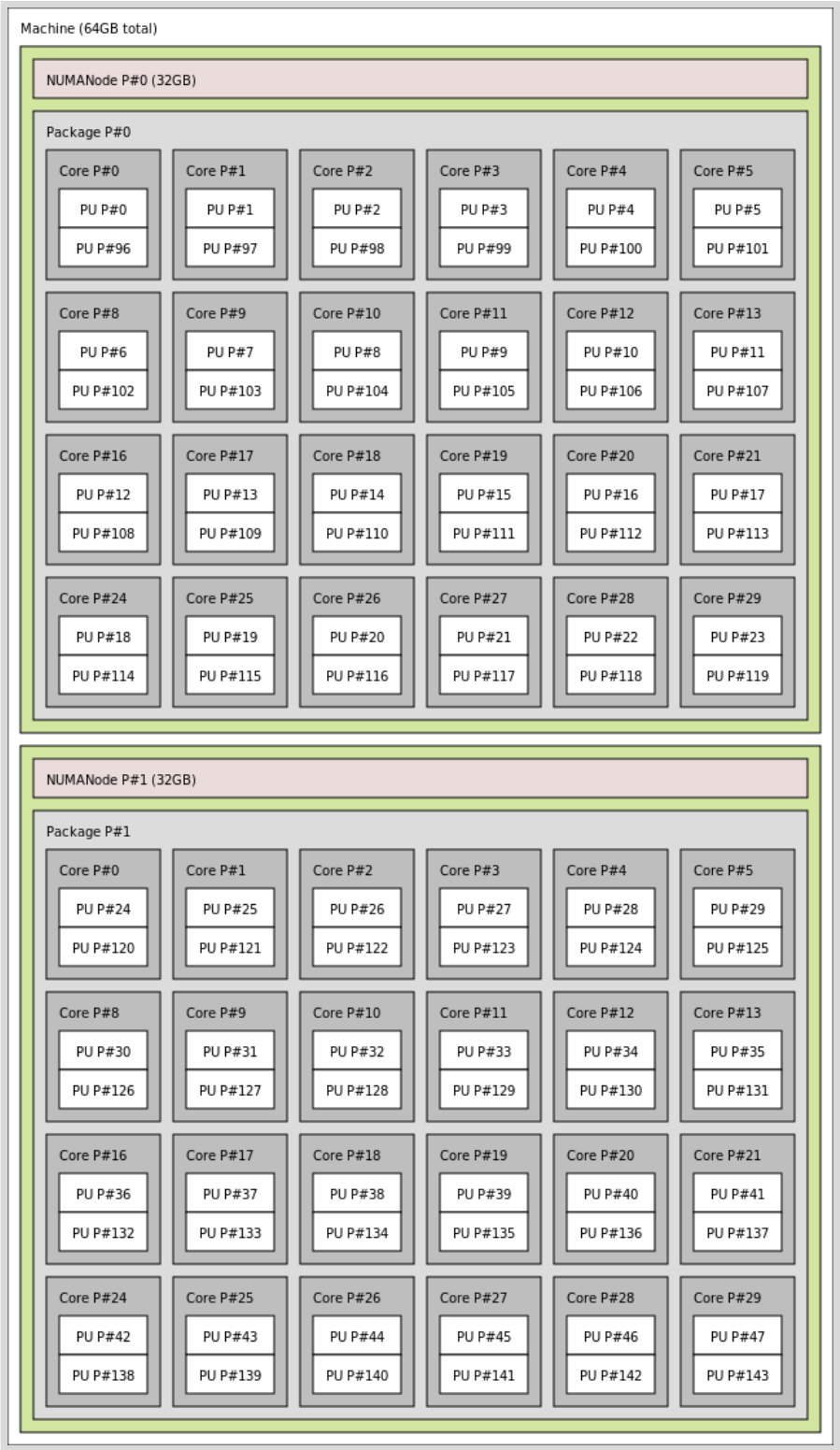


Figure 8.7: 4-nodes NUMA system (showing only two nodes)

The previous output only shows two nodes of a 4-nodes NUMA system (for image size constraints). Notice how the command displays the cores and threads (or PU for Processing Unit) of each CPU package.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The **lstopo** command also shows the CPU connections with the I/O controllers and devices (not displayed in the previous screen capture).

Monitoring NUMA Memory Usage

The **numastat** command, also from the *numactl* package, provides some memory usage statistics.

With the **-m** option, the **numastat** command displays the memory usage per node, in a similar way as the **/proc/meminfo** file. The **-c** option allows for a more compact view.

```
[root@host ~]# numastat -c -m

Per-node system memory usage (in MBs):
                 Node 0 Node 1 Node 2 Node 3  Total
                 ------ ------ ------ ------ ------
MemTotal          32642  32768  32768  24576 122754
MemFree           31409  31034  31728  23527 117698
MemUsed            1233   1734   1040   1049   5055
Active               19    546    128     87    780
Inactive             43    259     32     26    361
Active(anon)          4    359     68     82    514
Inactive(anon)       11    190     12     23    236
Active(file)         14    187     60      6    267
Inactive(file)       32     68     20      4    124
Unevictable           0      0      0      0      0
Mlocked               0      0      0      0      0
Dirty                 0      0      0      0      0
Writeback             0      0      0      0      0
FilePages            57    799     87    112   1056
Mapped                8      8      4      9     28
AnonPages             5      8     73      2     87
Shmem                 9    539      3    102    653
KernelStack           7      6      6      6     26
PageTables            3      1      2      1      7
NFS_Unstable          0      0      0      0      0
Bounce                0      0      0      0      0
WritebackTmp          0      0      0      0      0
Slab                124    128    121    118    491
SReclaimable         28     42     38     40    148
SUnreclaim           96     86     83     78    344
AnonHugePages         0      2     62      0     64
HugePages_Total       0      0      0      0      0
HugePages_Free        0      0      0      0      0
HugePages_Surp        0      0      0      0      0
```

The **numastat** command also accepts the PID or the name of a process as an argument. In that case, it lists the per-node memory usage of the process.

```
[root@host ~]# numastat -c rsyslogd

Per-node process memory usage (in MBs) for PID 20002 (rsyslogd)
         Node 0 Node 1 Node 2 Node 3 Total
         ------ ------ ------ ------ -----
Huge          0      0      0      0     0
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
Heap           0      0      0      0     0
Stack          0      0      0      0     0
Private        1      7     71      4    82
-------  ------ ------ ------ ------ -----
Total          1      7     71      4    82
```

The special **/proc/*PID*/numa_maps** file also provides that information in more details. See the numa(7) man page for a description of that file.

Associating Processes with NUMA Nodes

With some workloads, you may want to organize your processes on nodes manually, rather than relying on the automatic NUMA balancing algorithm.

To run a process in a specific node, use the **numactl** command to start it, passing the allowed CPU and memory nodes.

The following command starts the **myprogram** process on a CPU in node 2. The system should also allocate the memory from node 2 if possible. The **--** characters alone indicate the end of the **numactl** options. What follows is the application name and its options.

```
[user@host ~]$ numactl --cpunodebind=2 --preferred=2 -- myprogram
```

The following command starts the **myprogram** process on a CPU in node 2 and *requires* memory from node 2 or 3. If the system cannot allocate memory from one of those nodes, the OOM killer kills the process.

```
[user@host ~]$ numactl --cpunodebind=2 --membind=2,3 -- myprogram
```

NUMA can lead to some unexpected performance variations. If the memory footprint of an application does not fit into the local node, performance can vary or be hard to predict. You may face this situation with databases that consume the majority of the system memory. Depending on the location of the accessed data, the performance may vary drastically.

For such workloads, to get more predictable performance, use the **numactl** command to interleave all access to memory across all nodes.

```
[user@host ~]$ numactl --interleave all -- mydatabase
```

As an alternative, you can disable NUMA at the BIOS level, often through an option called "Socket Interleave" or similar, and add the **numa=off** option to the kernel command line.

Managing Non-Uniform Memory for Virtual Machines

KVM (Kernel-based Virtual Machine) on Red Hat Enterprise Linux, Red Hat OpenStack Platform, or Red Hat Virtualization lets you define a NUMA topology for your virtual machines. For example, you can create a virtual machine with four NUMA nodes, and another one with only two nodes. The topology you define for those machines does not have to match the NUMA architecture of the physical host.

One benefit of creating virtual machines with a NUMA topology is that you can associate, or pin the virtual NUMA nodes with the physical NUMA nodes. This way, you can organize the virtual machines running on a system to use the resources better and improve performances.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The Red Hat Virtualization web admin portal provides a graphical interface to configure the virtual machines' NUMA topology and pinning.

On Red Hat Enterprise Linux, the **`virt-install`** command that you use to create virtual machines provides several useful options. The **`--cpu`** option defines the virtual machine NUMA topology, and the **`--numatune`** and **`--cputune`** options associate virtual NUMA nodes to physical NUMA nodes and CPUs.

References

`numactl`(8) and **`numastat`**(8) man pages

Knowledgebase – What is NUMA and how does it work on Linux ?
https://access.redhat.com/solutions/700683

Knowledgebase – Automatic NUMA balancing in RHEL7
https://access.redhat.com/solutions/1533463

Knowledgebase – Enabling automatic NUMA balancing and numad together
https://access.redhat.com/solutions/3952761

For more information, refer to the *Configuring Virtual NUMA* section in the *Red Hat Virtualization 4.3 Virtual Machine Management Guide* at
https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html-single/virtual_machine_management_guide/index#Configuring_virtual_numa

For more information, refer to the *Configure CPU Pinning with NUMA* section in the *Red Hat OpenStack Platform 14 Instances and Images Guide* at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/14/html-single/instances_and_images_guide/index#ch-cpu_pinning

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring NUMA Topology

In this exercise, you will configure NUMA topology to provide efficient memory resource access to critical applications.

Outcomes

You should be able to query the NUMA topology of your system and use the tools to inspect and tune your processes for NUMA.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab memtuning-numa start** command. The command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the **bigmem** program that you use to allocate memory. This program has been developed specifically for this course. The command also configures the system to simulate NUMA nodes and reboots **servera**.

```
[student@workstation ~]$ lab memtuning-numa start
```

1. On **workstation** open two terminal windows. On each terminal, use **ssh** to log in to **servera** as **student**. You may have to wait for **servera** to finish rebooting. In the first terminal, use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Install the *numactl* package.

```
[root@servera ~]# yum install numactl
...output omitted...
Complete!
```

3. Use the **numactl --hardware** command to query the NUMA topology.

```
[root@servera ~]# numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 1
node 0 size: 503 MB
node 0 free: 384 MB
node 1 cpus: 0 1
node 1 size: 343 MB
node 1 free: 223 MB
node 2 cpus: 0 1
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
node 2 size: 479 MB
node 2 free: 465 MB
node 3 cpus: 0 1
node 3 size: 502 MB
node 3 free: 498 MB
node distances:
node   0   1   2   3
  0:  10  10  10  10
  1:  10  10  10  10
  2:  10  10  10  10
  3:  10  10  10  10
```

The system has four NUMA nodes. Each node has ~500 MiB of memory and two CPUs.

Note

The classroom environment does not provide NUMA systems. To simulate a NUMA architecture the **`lab memtuning-numa start`** command configured **`servera`** with the **`numa=fake=4`** kernel parameter. Because it is a simulation, the **`numactl --hardware`** output shows varying memory sizes for each node and fictitious CPU associations. The distance matrix shows the same distance between all nodes.

4. In the second terminal, run **`bigmem`** to allocate 100 MiB of memory. Verify on which NUMA node s the kernel allocated the memory.

4.1. In the second terminal, run **`bigmem`** with the requested memory size.

```
[student@servera ~]$ bigmem 100M
```

4.2. In the first terminal, use **`numastat`** to inspect the node allocation for **`bigmem`**.

```
[root@servera ~]# numastat -c bigmem

Per-node process memory usage (in MBs) for PID 1320 (bigmem)
         Node 0 Node 1 Node 2 Node 3 Total
         ------ ------ ------ ------ -----
Huge          0      0      0      0     0
Heap          0      0      0      0     0
Stack         0      0      0      0     0
Private       0    102      0      0   102
-------  ------ ------ ------ ------ -----
Total         0    102      0      0   102
```

The **`numastat -c`** option displays a more compact output.

The kernel allocated all the memory for the **`bigmem`** process on a single node. On your system, **`bigmem`** may be running on a different node, or on two nodes if the kernel cannot fit the process on a single node.

4.3. In the second terminal, press **`Enter`** to exit **`bigmem`**.

```
Press <Enter> to exit   Enter
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Run **bigmem** but allocate 600 MiB of memory. Verify on which NUMA nodes the kernel allocated the memory. Use **numactl** to try forcing the kernel to allocate the memory on a single node.

5.1. In the second terminal, run **bigmem** with the requested memory size.

```
[student@servera ~]$ bigmem 600M
```

5.2. In the first terminal, use **numastat** to view the node allocation for **bigmem**.

```
[root@servera ~]# numastat -c bigmem

Per-node process memory usage (in MBs) for PID 1328 (bigmem)
         Node 0 Node 1 Node 2 Node 3 Total
         ------ ------ ------ ------ -----
Huge          0      0      0      0     0
Heap          0      0      0      0     0
Stack         0      0      0      0     0
Private     197    210    196      0   604
-------  ------ ------ ------ ------ -----
Total       197    210    196      0   604
```

Because the **bigmem** request cannot fit on a single node, the kernel allocated it on multiple nodes. The memory access time will vary depending on the distance from the process node to the node for its memory.

5.3. Press **Enter** to exit the **bigmem** application.

```
Press <Enter> to exit   Enter
```

5.4. Start **bigmem** again but use **numactl --membind 1** to force the kernel to use the memory from only node 1.

```
[student@servera ~]$ numactl --membind 1 -- bigmem 600M
Process PID: 1392
Allocating 600 MiB of resident memory (in 1 MiB chunks)...
Killed
```

The OOM killer killed the process because not enough memory was available on node 1 to satisfy the request.

5.5. Inspect the system log for confirmation.

```
[student@servera ~]$ journalctl -n 20
...output omitted...
Jul 18 09:30:34 servera.lab.example.com kernel: Out of memory: Kill process 1392
 (bigmem) score 471 or sacrifice child
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Note

Use the **`--preferred`** option, instead of the **`--membind`** option, to allow the kernel to fall back to allocating memory on other nodes if available memory on the requested node is insufficient. On a real NUMA system, provide the **`--cpunodebind=`*`nodes`*** option, in addition to the **`--membind`** and **`--preferred`** options, to force the process to run on the CPUs of the given nodes.

6. For a more constant memory access time, use the **`numactl`** command to better spread the **`bigmem`** memory between the nodes.

6.1. Use the **`numactl`** command to interleave the memory of the **`bigmem`** process between all the nodes.

```
[student@servera ~]$ numactl --interleave all -- bigmem 600M
```

6.2. In the first terminal, use the **`numastat`** command to confirm that the memory is spread better between the nodes.

```
[root@servera ~]# numastat -c bigmem

Per-node process memory usage (in MBs) for PID 1359 (bigmem)
         Node 0 Node 1 Node 2 Node 3 Total
         ------ ------ ------ ------ -----
Huge          0      0      0      0     0
Heap          0      0      0      0     0
Stack         0      0      0      0     0
Private     151    151    152    151   604
-------  ------ ------ ------ ------ -----
Total       151    151    152    151   604
```

6.3. Press **`Enter`** to exit the **`bigmem`** application.

```
Press <Enter> to exit   Enter
```

7. Log off from **`servera`** in the two terminals.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **`workstation`**, run the **`lab memtuning-numa finish`** command to complete this exercise. The command deletes the resources created during the exercise, disable the NUMA simulation, and reboots **`servera`**.

```
[student@workstation ~]$ lab memtuning-numa finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Managing Memory Overcommit

Objectives

After completing this section, you should be able to describe memory overcommitment in typical application design.

Introducing Memory Overcommitment

The Linux kernel allows applications to allocate more memory than is available on the system. This feature is called *memory overcommitment*.

Remember that when a process requests memory, it only reserves virtual memory, and does not consume any RAM. The kernel only allocates physical page frames when the process starts using its memory. Because most processes request more memory than they consume, the system can overcommit memory.

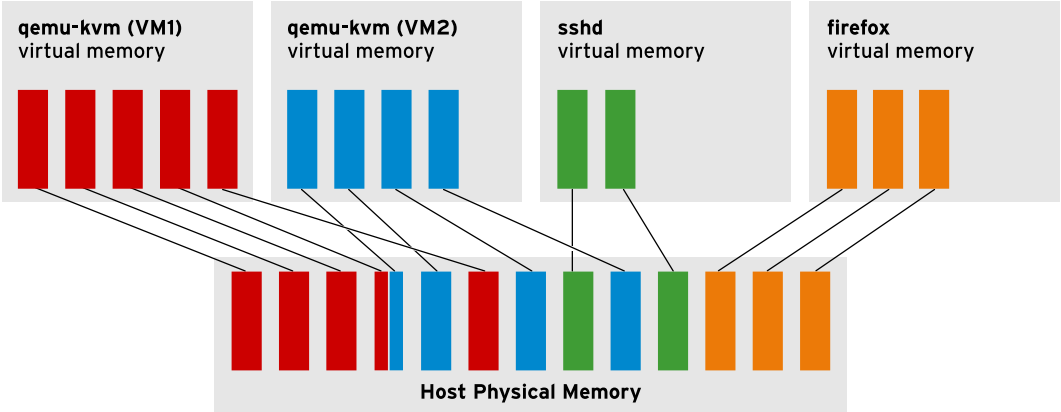


Figure 8.8: Virtual memory overcommit

The diagram above shows that the sum of all the process virtual memory is greater than the available physical memory The system is overcommitting memory. However, the sum of the memory actually used by the processes is less than the physical memory.

Virtual machines using KVM (Kernel-based Virtual Machine) on Red Hat Enterprise Linux, Red Hat OpenStack Platform, or Red Hat Virtualization, behave like normal processes on the host system. In the diagram, the two **`qemu-kvm`** processes correspond to two virtual machines.

If you configure a guest with 16 GiB of RAM, the host system only allocates physical page frames when the virtual guest starts using its memory. Memory overcommitment is an essential feature for virtualization because it allows the consolidation of more virtual systems on your host and uses your hardware resources more efficiently.

The same goes for containers started with Red Hat OpenShift or the **`podman`** command on a Red Hat Enterprise Linux system. For the kernel, a container is a set of processes running on the host.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

If the free RAM is running low, the system reclaims pages the usual way: from the page cache or by paging out inactive anonymous pages to the swap area.

When a machine is in a situation where it overcommits too much and now has to deliver more memory than is possible, the kernel invokes the OOM killer which kills some processes.

Tuning Memory Overcommitment

You control the memory overcommitment policy of your system with the **`vm.overcommit_memory`** **`sysctl`** tunable. Set the tunable to one of the following values:

- **`0`** – The kernel uses a heuristic overcommit algorithm. If an application tries to allocate more memory than could obviously be granted, the system refuses the allocation. However, the kernel usually accepts to overcommit memory for processes that perform a large number of small allocations.
- **`1`** – The kernel always overcommits memory. It always grants memory allocations, regardless of whether sufficient free memory exists.
- **`2`** – The kernel strictly controls memory overcommitment. It only commits an amount of memory equal to the amount of swap space plus a percentage (the default is 50) of physical memory. You define that percentage with the **`vm.overcommit_ratio`** tunable.

The **`Committed_AS`** field in the **`/proc/meminfo`** file shows an estimate of how much RAM the system requires to avoid an out-of-memory condition for the current workload.

Memory overcommitment is useful for scientific and engineering applications written in FORTRAN. FORTRAN programs do not allocate memory for data structures until they actually need that memory. Memory overcommitment makes it possible to run programs that allocate huge data structures, but only utilize a tiny amount of that data.

Since Red Hat Enterprise Linux 6, the default value for the **`vm.overcommit_memory`** tunable is **`0`**, or heuristic overcommit.

References

`/usr/share/doc/kernel-doc-*/Documentation/sysctl/vm.txt`

`/usr/share/doc/kernel-doc-*/Documentation/vm/overcommit-accounting.rst`

Guided Exercise

Managing Memory Overcommit

In this exercise, you will configure memory overcommit policies and observe out-of-memory system behavior.

Outcomes

You should be able to configure your machine to overcommit memory up to twice the system's RAM plus swap space.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab memtuning-overcommit start** command. The command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the **bigmem** program that you use to allocate memory. This program has been developed specifically for this course.

```
[student@workstation ~]$ lab memtuning-overcommit start
```

1. On **workstation** open two terminal windows. On each terminal, use **ssh** to log in to **servera** as **student**. In the first terminal, use **sudo -i** command to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Review the current memory overcommit policy.

```
[root@servera ~]# sysctl vm.overcommit_memory
vm.overcommit_memory = 0
```

0 means that the system uses the heuristic overcommit algorithm.

3. Use **bigmem** to allocate 20 GiB of virtual memory, which is more than the available physical memory. Use the **--virtual** (or **-v**) option to allocate memory in 1 MiB blocks.

```
[root@servera ~]# bigmem --virtual 20G
Process PID: 1981
Allocating 20480 MiB of virtual memory (in 1 MiB chunks)...
Allocated 10240 MiB
Allocated 20480 MiB
Done

Press <Enter> to exit   Enter
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The kernel accepts the memory overcommit because **bigmem** requests memory in small allocations.

Press **Enter** to exit the application.

> **Note**
> With the **--virtual** option, **bigmem** allocates the memory but does not use it. Otherwise, the OOM killer would likely kill the process.

4. Use the **bigmem** command to allocate 20 GiB of virtual memory in one big request. Use the **--block** (or **-b**) option.

```
[root@servera ~]# bigmem --block 20G
Process PID: 1984
Allocating 20480 MiB of virtual memory...
malloc: Cannot allocate memory
```

This time, the kernel refuses the allocation request.

5. Select the memory overcommit policy for the kernel to always grant memory allocations.

```
[root@servera ~]# sysctl -w vm.overcommit_memory=1
vm.overcommit_memory=1
```

6. Use **bigmem --block** again to allocate 20 GiB of virtual memory in one big request.

```
[root@servera ~]# bigmem --block 20G
Process PID: 2017
Allocating 20480 MiB of virtual memory...
Done

Press <Enter> to exit   Enter
```

Now, the kernel accepts to overcommit the memory.

Press **Enter** to exit the application.

7. Configure the system to allow the kernel to overcommit memory up to twice the system's RAM plus swap space.

7.1. Change the kernel tunable never to allow overcommitting memory.

```
[root@servera ~]# sysctl -w vm.overcommit_memory=2
vm.overcommit_memory=2
```

7.2. Configure the kernel to overcommit twice (200%) the amount of physical RAM.

```
[root@servera ~]# sysctl -w vm.overcommit_ratio=200
vm.overcommit_ratio=200
```

8. Run **watch** to monitor the current committed address space and the commit limit.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# watch  grep Commit /proc/meminfo
Every 2.0s: grep Commit /proc/meminfo ...

CommitLimit:     3745872 kB
Committed_AS:     412696 kB
```

Your values may be different. From this output, it appears possible to request no more than 3745872 - 412696 = 3333176 KiB (3 GiB) of memory. You may have to calculate your limit from your output.

9. In the second terminal, start **bigmem** in the background to allocate 2.5 GiB (2560 MiB) of virtual memory. That amount is slightly below the limit.

```
[student@servera ~]$ bigmem --virtual 2560M &
```

The command succeeds because the value is below the commit limit. In the **watch** output, the committed address space is very close to the commit limit.

10. Use the **bigmem** command to allocate an additional gigabyte of virtual memory.

```
[student@servera ~]$ bigmem --virtual 1G
Process PID: 3111
Allocating 1024 MiB of virtual memory (in 1 MiB chunks)...
malloc: Cannot allocate memory
```

The command fails because the request is above the commit limit.

11. Press **Ctrl**+**C** to exit **watch** and return to the shell prompt. Kill the **bigmem** process running in the background.

```
[student@servera ~]$ pkill bigmem
```

12. Log off from **servera** in the two terminals.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab memtuning-overcommit finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab memtuning-overcommit finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning Memory Utilization

Performance Checklist

In this lab, you will configure memory settings for a variety of performance tuning use cases.

Outcomes

You should be able to:

- Create a new tuned profile and use a set of system parameters to tune the memory of your server.
- Update a systemd service to control and restrict the memory usage of a process.
- Update a service configuration to adapt its memory usage to the NUMA topology.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab memtuning-review start** command. The command determines if the host, **serverb**, is reachable on the network. On **serverb**, the command installs a PostgreSQL database, the **memload** program, and the **memload** service that you use in the exercise. The **memload** program has been developed specifically for this course. The command also configures the system to simulate NUMA nodes and reboots **serverb**.

```
[student@workstation ~]$ lab memtuning-review start
```

1. A PostgreSQL database is running on **serverb**. To improve its performances, create a new tuned profile as follows:

   - Name the new profile **enterprise-database**.
   - Base your new profile on the existing **throughput-performance** profile.
   - Allocate 256 MiB of huge pages.
   - Disable transparent huge pages.
   - The machine can use swap space but prefer to use it as little as possible. Ignore the fact that **serverb** has no swap space for this step.
   - Start background syncing dirty pages when they consume 3% of the total memory.
   - Start actively syncing dirty pages when they consume 15% of the total memory.

   Do not enable your new profile at that stage.

2. A new **memload** service is installed on **serverb**. That service fails to start. The **memload** program makes a huge memory request but only uses a few megabytes. Configure the **enterprise-database** profile so that this program can start on the system.

3. Now that the system does not restrict the virtual memory a process can request, configure so that the **memload** process cannot consume more than 512 MiB of physical memory.

   To do so, update the **memload** systemd service configuration to kill the process when it consumes more than 512 MiB of RAM.

4. Update the **memload** systemd service configuration to spread the memory of the **memload** process equally across NUMA nodes. The **ExecStart** systemd service parameter gives the path to the program that the service starts.
5. Enable the new **enterprise-database** profile. Restart the **postgresql** and **memload** services. Check that the system uses some of the allocated huge pages, and that the kernel spreads the **memload** memory across NUMA nodes.

Evaluation

On **workstation**, run the **lab memtuning-review grade** command to grade this lab. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab memtuning-review grade
```

Finish

On **workstation**, run the **lab memtuning-review finish** command to finish this exercise.

```
[student@workstation ~]$ lab memtuning-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning Memory Utilization

Performance Checklist

In this lab, you will configure memory settings for a variety of performance tuning use cases.

Outcomes

You should be able to:

- Create a new tuned profile and use a set of system parameters to tune the memory of your server.
- Update a systemd service to control and restrict the memory usage of a process.
- Update a service configuration to adapt its memory usage to the NUMA topology.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab memtuning-review start** command. The command determines if the host, **serverb**, is reachable on the network. On **serverb**, the command installs a PostgreSQL database, the **memload** program, and the **memload** service that you use in the exercise. The **memload** program has been developed specifically for this course. The command also configures the system to simulate NUMA nodes and reboots **serverb**.

```
[student@workstation ~]$ lab memtuning-review start
```

1. A PostgreSQL database is running on **serverb**. To improve its performances, create a new tuned profile as follows:

   - Name the new profile **enterprise-database**.
   - Base your new profile on the existing **throughput-performance** profile.
   - Allocate 256 MiB of huge pages.
   - Disable transparent huge pages.
   - The machine can use swap space but prefer to use it as little as possible. Ignore the fact that **serverb** has no swap space for this step.
   - Start background syncing dirty pages when they consume 3% of the total memory.
   - Start actively syncing dirty pages when they consume 15% of the total memory.

   Do not enable your new profile at that stage.

   1.1. On **workstation**, use **ssh** to log in to **serverb** as **student**. You may have to wait for **serverb** to finish rebooting. Use **sudo -i** to switch to the **root** user's account.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

1.2. Create the **`enterprise-database`** tuned profile directory.

```
[root@serverb ~]# mkdir /etc/tuned/enterprise-database
```

1.3. Create **`/etc/tuned/enterprise-database/tuned.conf`** with the following:

```
[main]
include = throughput-performance

[vm]
# Disabling transparent huge pages
transparent_hugepages = never

[sysctl]
# Allocating 256 MiB of huge pages (256 MiB / 2 MiB page = 128)
vm.nr_hugepages = 128

# Using swap as little as possible
vm.swappiness = 1

# Starting background writeback at 3%
vm.dirty_background_ratio = 3

# The process generating dirty data starts writeback at 15%
vm.dirty_ratio = 15
```

Do not activate the profile. Additional configuration is required.

2. A new **`memload`** service is installed on **`serverb`**. That service fails to start. The **`memload`** program makes a huge memory request but only uses a few megabytes. Configure the **`enterprise-database`** profile so that this program can start on the system.

2.1. Retrieve the status of the **`memload`** the service.

```
[root@serverb ~]# systemctl status memload.service
* memload.service - Allocate memory
   Loaded: loaded (/etc/systemd/system/memload.service; enabled; vendor preset:
 disabled)
   Active: failed (Result: exit-code) since Fri 2019-07-19 08:04:24 EDT; 12s ago
  Process: 1284 ExecStart=/usr/local/bin/memload 40G (code=exited, status=1/
FAILURE)
 Main PID: 1284 (code=exited, status=1/FAILURE)

Jul 19 08:04:24 serverb... systemd[1]: Started Allocate memory.
Jul 19 08:04:24 serverb... memload[1284]: Allocating 40960 MiB of virtual memory
Jul 19 08:04:24 serverb... memload[1284]: malloc: Cannot allocate memory
...output omitted...
```

The process fails to allocate 40 GiB (40960 MiB) of virtual memory. The system does not have this amount of physical memory and refuses to overcommit.

2.2. Configure the system to always overcommit memory when requested. Edit the configuration file of the **`enterprise-database`** profile, **`/etc/tuned/enterprise-database/tuned.conf`**

```
...output omitted...
# Starting background writeback at 3%
vm.dirty_background_ratio = 3

# The process generating dirty data starts writeback at 15%
vm.dirty_ratio = 15

# Always accept memory overcommit
vm.overcommit_memory = 1
```

Do not activate the profile or start **`memload`**. The service requires additional configuration.

3. Now that the system does not restrict the virtual memory a process can request, configure so that the **`memload`** process cannot consume more than 512 MiB of physical memory.

To do so, update the **`memload`** systemd service configuration to kill the process when it consumes more than 512 MiB of RAM.

3.1. Create a drop-in file for the **`memload`** systemd service. Set the **`MemoryLimit`** parameter to 512 MiB.

```
[root@serverb ~]# systemctl set-property memload.service MemoryLimit=512M
```

As an alternative, create the drop-in directory manually:

```
[root@serverb ~]# mkdir /etc/systemd/system/memload.service.d
```

Create the **`/etc/systemd/system/memload.service.d/00-limit.conf`** drop-in file with the following contents.

```
[Service]
MemoryLimit=512M
```

3.2. Instruct systemd to reload its configuration.

```
[root@serverb ~]# systemctl daemon-reload
```

Do not start the service yet.

4. Update the **`memload`** systemd service configuration to spread the memory of the **`memload`** process equally across NUMA nodes. The **`ExecStart`** systemd service parameter gives the path to the program that the service starts.

4.1. Install the *numactl* package.

```
[root@serverb ~]# yum install numactl
...output omitted...
Complete!
```

4.2. Edit the **`memload`** systemd service to prefix the command by **`/bin/numactl --interleave all`**. The service configuration file is **`/etc/systemd/system/memload.service`**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[Unit]
Description=Allocate memory
After=network.target

[Service]
ExecStart=/bin/numactl --interleave all -- /usr/local/bin/memload 40G

[Install]
WantedBy=multi-user.target
```

Provide the full path to the **`numactl`** command (**`/bin/numactl`**).

4.3. Instruct systemd to reload its configuration.

```
[root@serverb ~]# systemctl daemon-reload
```

5. Enable the new **`enterprise-database`** profile. Restart the **`postgresql`** and **`memload`** services. Check that the system uses some of the allocated huge pages, and that the kernel spreads the **`memload`** memory across NUMA nodes.

5.1. Enable the **`enterprise-database`** tuned profile.

```
[root@serverb ~]# tuned-adm profile enterprise-database
```

5.2. Restart the **`postgresql`** service.

```
[root@serverb ~]# systemctl restart postgresql
```

5.3. Start the **`memload`** service and verify that the **`memload`** process is running.

```
[root@serverb ~]# systemctl start memload
[root@serverb ~]# systemctl status memload
* memload.service - Allocate memory
   Loaded: loaded (/etc/systemd/system/memload.service; enabled; vendor preset:
 disabled)
  Drop-In: /etc/systemd/system.control/memload.service.d
           └─50-MemoryLimit.conf
   Active: active (running) since Fri 2019-07-19 08:34:21 EDT; 4s ago
 Main PID: 1572 (memload)
    Tasks: 1 (limit: 11399)
   Memory: 10.2M (max: 512.0M)
   CGroup: /system.slice/memload.service
           └─1572 /usr/local/bin/memload 40G

Jul 19 08:34:21 serverb... systemd[1]: Started Allocate memory.
Jul 19 08:34:21 serverb... numactl[1572]: Allocating 40960 MiB of virtual memory
Jul 19 08:34:21 serverb... numactl[1572]: Memory allocation done
Jul 19 08:34:21 serverb... numactl[1572]: Using 10 MiB of memory
Jul 19 08:34:21 serverb... numactl[1572]: Using memory done
```

5.4. Check that the system uses some of the huge pages.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# grep Huge /proc/meminfo
AnonHugePages:     12288 kB
ShmemHugePages:        0 kB
HugePages_Total:     128
HugePages_Free:      121
HugePages_Rsvd:       64
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:          262144 kB
```

Out of the 128 available pages, the system uses seven pages (128 - 121 = 7). The number of used pages may be different on your system.

5.5. Verify that the kernel spread the **memload** memory between all NUMA nodes.

```
[root@serverb ~]# numastat -c memload

Per-node process memory usage (in MBs) for PID 1572 (memload)
         Node 0 Node 1 Node 2 Node 3 Total
         ------ ------ ------ ------ -----
Huge          0      0      0      0     0
Heap          0      0      0      0     0
Stack         0      0      0      0     0
Private       3      3      3      3    11
-------  ------ ------ ------ ------ -----
Total         3      3      3      3    11
```

5.6. Log off from **serverb** when done.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab memtuning-review grade** command to grade this lab. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab memtuning-review grade
```

Finish

On **workstation**, run the **lab memtuning-review finish** command to finish this exercise.

```
[student@workstation ~]$ lab memtuning-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- Memory is organized and allocated in pages, which are normally 4 KiB in size.
- Memory limits can be enforced on systemd units through systemd configuration files.
- The **`vm.nr_hugepages`** sysctl parameter defines the number of huge pages to allocate.
- The **`vm.dirty_background_ratio`** and **`vm.dirty_ratio`** sysctl tunables control when per-BDI flush threads start writing data to disk.
- The **`/proc/`*`PID`*`/oom_score_adj`** tunable adjusts the **`oom_score`** for a specific process.
- The **`numastat`** tool displays process memory statistics on a per-NUMA node basis. **`numactl`** controls the process affinity for processor and memory.
- The **`vm.overcommit_memory`** sysctl parameter defines the system overcommit mode.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 9

Tuning Storage Device I/O

Goal

Manage settings for efficient disk utilization in various use cases.

Objectives

- Describe transaction-oriented and sequential I/O patterns and disk scheduling algorithms.
- Describe basic RAID architecture to understand the correlation between application writes and RAID sizing alignment.
- Describe multiple tool choices for analyzing disk I/O behavior.

Sections

- Evaluating I/O Patterns and Scheduling Algorithms (and Guided Exercise)
- Reviewing RAID Fundamentals (and Guided Exercise)
- Selecting I/O Analysis Tools (and Guided Exercise)

Lab

Tuning Storage Device I/O

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Evaluating I/O Patterns and Scheduling Algorithms

Objectives

After completing this section, you should be able to describe transaction-oriented and sequential I/O patterns and disk scheduling algorithms.

Transitioning to Solid State Drives

Solid-state drives (SSD) have replaced magnetic hard disk drives as the standard for disk storage. SSDs use intergated-circuit memory with no moving parts. Many behaviors associated with hard disks, such as spin-up time, seek time, rotational delay, mechanical failure, and magnetic susceptibility do not occur with SSDs. As access times for SSDs are uniform across the whole device, making fragmentation and track optimization procedures unnecessary. SSDs are generally smaller, weigh less, and are more energy-efficient than hard disks.

SSDs do have limitations, however, such as higher cost. Solid-state drives are limited to a determined number of writes, although software design compensation, such as wear leveling, allows SSD write lifetimes to be similar to hard drive failure expectations.

Most legacy general tuning procedures for hard disks do not apply to SSDs. For example, SSDs do not require the use of read ahead and write behind caches. Caches should be configured as write-through. Red Hat does not recommend using journaling on SSD devices, because of increased SSD wear and the slowness caused by the unnecessary double writing.

Selecting an I/O Scheduler

RHEL 8 includes new multiple queue I/O schedulers replacing single-queue based I/O schedulers. The new schedulers use the Multi-Queue Block I/O Queuing Mechanism (blk-mq), which enables I/O operations mapped to multiple hardware or software request queues.

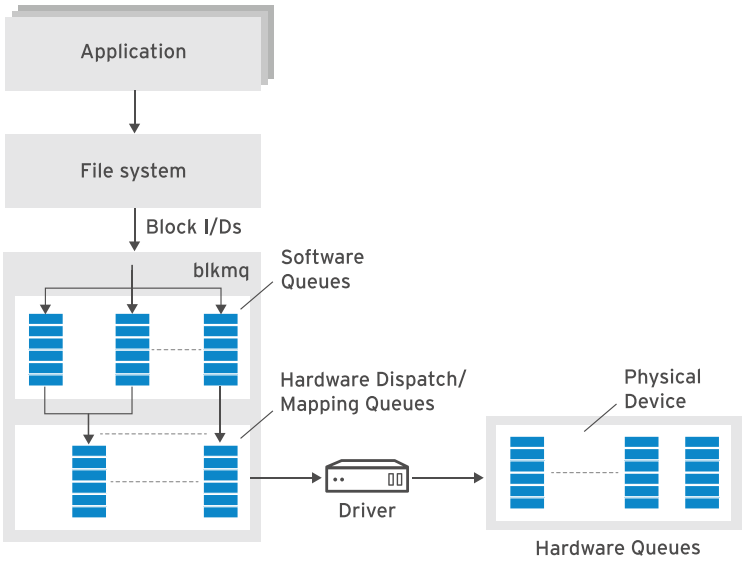


Figure 9.1: Multiple queue I/O scheduling

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

`Blk-mq` maps I/O operations to different queues and execution threads. Threads are executed by different cores on each CPU, reducing the lock contention when multiple threads of execution perform I/O to a single device. Newer device interfaces, such as the Non-Volatile Memory Express (NVMe) protocol, take advantage of this feature with native support for multiple hardware submission and completion queues and low-latency performance characteristics. Performance gains achieved depend on hardware characteristics and workload behavior.

RHEL 8 includes four multiqueue I/O scheduler settings: **`mq-deadline`**, **`kyber`**, **`bfq`**, and **`none`**. Each has performance advantages on specific devices, and choosing a scheduler must be based on benchmark tests on production storage systems.

`mq-deadline`
: The **`mq-deadline`** scheduler implements a multiple queue version of the legacy deadline I/O scheduler. This scheduler mitigates I/O operation starvation by using two queues (one for write, the other for read) sorted by expiration time, to ensure that an I/O operation (either write or read) is executed before it expires. If no expiration time is imminent, I/O operations are executed from a third queue that sorts requests by sector. The **`mq-deadline`** scheduler is the RHEL 8 default, providing ordering guarantees for arriving write operations that the other multiqueue schedulers do not yet handle.

`kyber`
: Kyber supports fast devices with two queues, one for synchronous (read) requests and another one for asynchronous (write) requests. These queues manage the per-request latency by placing strict limits on the number of request operations sent to the queues. This limits the time waiting for requests to be dispatched, and provides quick completion time for requests that are high priority. The **`kyber`** scheduler is preferred for throughput-sensitive server loads, especially on SSDs, primarily because it is a simpler scheduler, which should result in faster request processing.

`bfq`
: BFQ is an improvement on the Completely Fair Queuing (CFQ) scheduler, where fair sharing is based on the number of sectors requested and heuristics rather than a time slice. BFQ provides consistent interactive response, especially on slower devices hard disk devices. BFQ has a relatively high per-operation overhead, which may be acceptable when I/O operations are slow and costly. BFQ is not desired where I/O operations are cheap and throughput is a priority, such as with slow CPUs. This I/O scheduler is a replacement for the **`cfq`** single-queue based I/O scheduler. When using SSDs for server workloads, a simpler scheduler is preferred, such as Kyber. Users concerned with interactive system response, and possibly using slower devices, will would choose BFQ.

`none`
: This setting supports fast random I/O devices like NVMe disks, by not reordering I/O requests. This I/O scheduler is a replacement for the **`noop`** single-queue based I/O scheduler.

> **Note**
> NVMe disks use the **`none`** I/O scheduler by default. You cannot change the I/O scheduler for an NVMe disk.

Tuning Schedulers in `sysfs`

In the **`sys`** file system, the **`/sys/block/`*`disk_device`*`/queue/scheduler`** lists the available I/O schedulers for a device. The current active I/O scheduler is listed enclosed with square brackets. The following example shows that **`mq-deadline`** is the active I/O scheduler for the **`vda`** device.

```
[root@host ~]# cat /sys/block/vda/queue/scheduler
[mq-deadline] kyber bfq none
```

The tunables available for the active I/O scheduler on a disk are available in the **/sys/block/*disk_device*/queue/iosched** directory. You can find more information about the tunables available for each I/O scheduler in the references. The following example shows the tunables available for the **mq-deadline** scheduler on **vda**.

```
[root@host ~]# ls /sys/block/vda/queue/iosched
fifo_batch  front_merges  read_expire  write_expire  writes_starved
```

Note
You can enable the legacy single-queue I/O schedulers (**deadline**, **cfq**, **noop**) at boot time with the **scsi_mod.use_blk_mq=0** kernel parameter.

Tuning Storage with Tuned Profiles

The **tuned** service has predefined tuned profiles to support a variety of use cases. Storage examples include the **latency-performance** and **throughput-performance** profiles. You can create custom tuned profiles to support your environment-specific requirements. Several **tuned** plugins support storage tuning, such as:

- The **disk** plugin configures an I/O scheduler (elevator) to use on a device, with the **elevator** parameter.
- The **sysfs** plugin configures I/O scheduler tunables, using key=value definitions. The following example configures the **fifo_batch** tunable on the **vda** device to be **1**.

```
[sysfs]
/sys/block/vda/queue/iosched/fifo_batch=1
```

Simulating workloads with the fio tool

Testing a storage system requires simulating real world workloads. The **fio** command simulates workloads using multiple threads and processes. Scenarios include sequential, random reading and writing, or a mix of I/O types. The utility supports creating multiple-thread I/O requests on a single file.

Use this utility in combination with your chosen measurement tools. Use **fio** to create a consistent workload repetitively, while you gather metrics that quantify the resource configuration changes being tested. The **/usr/share/doc/fio/HOWTO** is recommended for learning how **fio** is designed to be used.

The following example executes a series of random writes (**--rw=randwrite**) on a file (**--filename=/tmp/testfile**) up to a file size (**--size=512M**). It also does direct I/O (**--direct=1**) with two jobs performing the same operation (**--numjobs=2**).

```
[root@host ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 \
--group_reporting --filename=/tmp/testfile
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Useful FIO Parameters

–filename
The utility will write to the filename specified. Be cautious, as using a block device file will overwrite the device during testing.

–name, -size
If existing files are not specified with –filename, **`fio`** will create files using the name and sizes specified with these parameters.

-rw = read/write/randread/randwrite/readwrite/randrw
Specifies whether workload should be sequential or random access. Options, as listed, include sequential reading, sequential writing, random reading, random writing, sequential mixed and random mixed. Additional options are available to set an exact percentage of read and write workloads when mixed.

-bs, -bsrange
Specifies an exact block size or a range of sizes to be used for reading and writing

–direct = 1
Sets the use of direct I/O, which bypasses the page cache, and writes directly to disk. When this option is set to **`0`**, **`fio`** uses the page cache and memory, and returns statistics that are essentially a measurement of memory speeds, not disk.

-ioengine = libaio, –iodepth
The libaio library enables asynchronous application workloads, which can be used to generate parallel requests. The **`-iodepth`** option specifies the number of requests. Using the option mandates the use of the direct I/O option, since the page cache is not asynchronously addressable. The ioengine also has parameters for synchronous I/O in multiple forms, memory mapped file activity, rdma memory activity, and even network activity..

-numjobs, -group_reporting
Specifies the number of processor threads performing the same workload for this testing. Defaults to only 1. If **`numjobs`** is specified, then use **`group_reporting`** to display consolidated statistics instead of per-job statistics.

References

`fio` manual page

`/usr/share/doc/fio/HOWTO`

Knowledgebase: "What tuning parameters are available for the mq-deadline I/O scheduler?"
https://access.redhat.com/solutions/3790801

Two new block I/O schedulers for 4.12
https://lwn.net/Articles/720675/

For more information, refer to the *Customizing Tuned profiles* chapter in the *Red Hat Enterprise Linux 8 Monitoring and managing system status and performance guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance

Guided Exercise

Evaluating I/O Patterns and Scheduling Algorithms

In this exercise, you will configure I/O scheduler tunables for different types of application behavior.

Outcomes

You should be able to first review the default I/O scheduler configuration and available tunables, then create the **reduce-latency** tuned profile to customize a tunable of the I/O scheduler to reduce latency.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab storage-patterns start** command. The command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab storage-patterns start
```

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Verify the active I/O scheduler, and review the tunables available for this scheduler.

   2.1. Verify that the active I/O scheduler is **mq-deadline**.

```
[root@servera ~]# cat /sys/block/vda/queue/scheduler
[mq-deadline] kyber bfq none
```

   2.2. Review the tunables available for the **mq-deadline** scheduler.

```
[root@servera ~]# ls /sys/block/vda/queue/iosched
fifo_batch  front_merges  read_expire  write_expire  writes_starved
```

   2.3. Verify the value for the **fifo_batch** tunable. This tunable determines the size for the batches of operations. A small value for this tunable reduces the latency, and a higher value increases throughput.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# cat /sys/block/vda/queue/iosched/fifo_batch
16
```

3. Determine the latency for **vda** with the **fio** command. When done, remove the **/tmp/testfile** file.

   3.1. Install the **fio** utility.

```
[root@servera ~]# yum install fio
...output omitted...
```

   3.2. Create a 512 MB file named **/tmp/testfile** with **fio**. Review the current value for **avg** in **clat (usec)**. This value may vary in your environment.

```
[root@servera ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 \
--group_reporting --filename=/tmp/testfile
...output omitted...
    clat (usec): min=0, max=51154k, avg=262584.65, stdev=314735.28
...output omitted...
```

   3.3. Remove the **/tmp/testfile** file.

```
[root@servera ~]# rm -f /tmp/testfile
```

4. Create a new custom tuned profile named **reduce-latency**. Inherit settings from the **latency-performance** profile.

   4.1. Create the directory **/etc/tuned/reduce-latency** for the new custom profile.

```
[root@servera ~]# mkdir /etc/tuned/reduce-latency
```

   4.2. Inherit the settings for the new profile from the existing **latency-performance** profile. Create the **/etc/tuned/reduce-latency/tuned.conf** file, and add the following two statements to the newly created file.

```
[main]
include=latency-performance
```

   4.3. Decrease the batch size for operations to be sent to **vda** to reduce the latency for this disk. Set the **/sys/block/vda/queue/iosched/fifo_batch** tunable to **1**. Add the following statements to the **/etc/tuned/reduce-latency/tuned.conf** file using the **sysfs** plug-in.

```
[sysfs]
/sys/block/vda/queue/iosched/fifo_batch=1
```

5. Enable the newly created tuned profile **reduce-latency**. When done, verify that the custom settings for the profile are enabled.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5.1. Enable the **reduce-latency** profile.

```
[root@servera ~]# tuned-adm profile reduce-latency
```

5.2. Verify that the custom value for **/sys/block/vda/queue/iosched/fifo_batch** is active on **servera**.

```
[root@servera ~]# cat /sys/block/vda/queue/iosched/fifo_batch
1
```

6. Create a 512 MB file named **/tmp/testfile** with **fio**. Review the current value for **avg** in **clat (usec)**. The latency should now be lower.

```
[root@servera ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 --group_reporting \
--filename=/tmp/testfile
...output omitted...
    clat (usec): min=0, max=41126k, avg=215403.66, stdev=239728.96
...output omitted...
```

7. Log out from **servera** to return to **workstation** as **student** user.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab storage-patterns finish** command to finish this exercise. The command activates the default tuned profile, **virtual-guest** on **servera**.

```
[student@workstation ~]$ lab storage-patterns finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Reviewing RAID Fundamentals

Objectives

After completing this section, you should be able to describe basic RAID architecture to understand the correlation between application writes and RAID sizing alignment.

Reviewing RAID

The Redundant Array of Inexpensive Disks (RAID) is a virtualized data storage technology, combining storge drives as a single logical storage unit to support increased performance, redundancy, and fault tolerance. RAID is an alternative to storage systems based on single, large and expensive disks. RAID supports the failure of a RAID disk without data downtime or data loss.

A RAID breaks the data in small chunks which are written to different disks of the array. Red Hat Enterprise Linux 8 defines a chunk size of 512 KB by default. The data is distributed among the RAID disks to meet one or more of the following data strategies: data striping, data mirroring, and data parity.

Data striping divides the data in stripes that are distributed among several disks inside of the RAID array. This strategy increases throughput because data is retrieved from several disks instead of a single disk.

Data mirroring provides redundancy copying the stripes to at least two different RAID disks, so in case of a disk failure, the RAID array can provide data with other data copies available in the array with no data downtime or loss. Read throughput of RAID 1 is slower than the fastest drive. Write throughput is always slower because every drive is updated, and the slowest drive determines the write performance.

Data parity supports data redundancy by adding parity to the stripes so other stripes related to the same data volume can be reconstructed using the working data and its associated parity bits.

A RAID array supports one or more of these strategies on different RAID levels. The following table includes some of the most used RAID levels.

Level	Data Strategy
RAID 0	Striping, no redundancy. Increased throughput.
RAID 1	Data mirroring. Redundant, double writing means slower throughput.
RAID 4	Block-level striping with dedicated parity disk. Small transfer parallelism.
RAID 5	Block-level striping with distributed parity. Redundant, but long rebuild times.
RAID 6	Block-level striping with double distributed parity. Mitigates RAID 5 problems.
RAID 10	Striped set of mirrors. Sustain multiple losses. Provides sufficient performance.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Tuning SSD for RAID

SSD-based RAIDs require some specific tuning compare to HDD-based RAIDs. If the RAID is a hardware-based RAID, use the latest firmware version available for the hardware. Recent RAID firmware versions provide better SSD support. Many hardware-based RAID arrays also include caches to enhance read (write cache), and write (read ahead). SSD drives do not benefit from using such caches, since SSDs are made of similar components as cache and have no rotational latency. In general, write behind and read ahead should be disabled for SSD drives to eliminate the redundant overhead.

SSD drives include a spare area of extra blocks to substitue for bad blocks that fail. The bigger the spare area the better the performance of the SSD on random writes. You can extend the spare area creating the RAID with only a part of the SSD available space.

As a best practice, make sure that your SSD manufacturer provides appropriate SMART attributes that ease SSD health monitoring, and make sure you check RAID consistency periodically.

Creating RAID arrays

Red Hat Enterprise Linux supports the creation and management of software RAID with the **mdadm** utility. The Multiple Disk and Device Administration (**mdadm**) utility supports the creation of software RAID arrays of any RAID level. The following command creates a RAID 0 into **/dev/md0** with two RAID disks, **vdb** and **vdc**.

```
[root@host ~]# mdadm -C /dev/md0 -l raid0 -n 2 /dev/vd[b-c]1
```

The disks used as RAID disks should not be accessed directly when the RAID array is ready. To remove a RAID array, you need to first stop the RAID array with the **mdadm --stop command**.

```
[root@host ~]# mdadm --stop /dev/md0
```

When done, you need to remove the RAID, as follows.

```
[root@host ~]# mdadm --remove /dev/md0
```

When the RAID is removed, you need to remove the superblock on each RAID disk. The **mdadm** utility uses this header to identify a disk as a RAID disk.

```
[root@host ~]# mdadm --zero-superblock /dev/vdb
[root@host ~]# mdadm --zero-superblock /dev/vdc
```

Note
Red Hat Enterprise Linux 8 also supports the management of software RAID arrays through its web console.

Stripe unit and width

Performance is significantly impacted by file system metadata alignment on disks in striped arrays (RAID 0, RAID 4, RAID 5, RAID 6). On a striped array, a chunk of data of *CHUNKSIZE* is written to each disk sequentially. If the write request is wider than the strip size, the write continues again at the first disk. If the file system's write behavior is not tuned to match the RAID layout, I/O

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

requests could require two writes per disk instead of one, or having all metadata ends up on one disk, causing that disk to become a hot spot.

To counter this, lay out your file system during creation to match the layout of the RAID array that it resides on. For XFS file systems, optimization for striped allocation requires the following information.

- The chunk size of the RAID array.
- The number of disks in the RAID array.
- The number of parity disks in the RAID array.

Subtracting the number of parity disks from the total number of disks will provide the number of data disks in the array. The chunk size of the RAID array and the number of data disks in the array can then be passed to the **`mkfs`** command to align an XFS file system during its creation.

The **`su`** and **`sw`** data section options are used to specify the RAID array chunk size and the number of data disks, respectively.

```
[root@host ~]# mkfs -t xfs -d su=64k,sw=4 /dev/san/lun1
```

For ext4 file systems, the following information is required for file system alignment.

- The file system block size desired
- The chunk size of the RAID array
- The number of disks in the RAID array
- The number of parity disks in the RAID array

This information is then used to calculate the file system stride and stripe-width.

The stride is the number of file system blocks that fit inside one chunk. As an example, for a file system block size of 4 KiB and a chunk size of 64 KiB, the stride will be 64 KiB/4 KiB = 16 blocks.

Likewise, stripe-width is the number of file system blocks that fit on one stripe of the RAID array. For example, imagine a six disk RAID 6 array. By definition, in each stripe of a RAID 6 array, two disks contain parity data. For the stripe-width, it is necessary to know how many disks in each stripe actually carry data blocks, so that will be 6 disks - 2 parity disks = 4 data disks. Each of those four disks will have stride number of file system blocks per chunk, so calculate 4 (disks) × 16 (*stride*) = 64 file system blocks per stripe.

These numbers are provided to **`mkfs`** at file system creation time to lay out the file system using the **`stride`** and **`stripe-width`** extended options.

```
[root@host ~]# mkfs -t ext4 -E stride=16,stripe-width=64 /dev/san/lun1
```

Configuring RAID on Logical Volumes

Red Hat Enterprise Linux 8 supports the creation of logical volumes that use the RAID approach. Logical volumes of type RAID use physical volumes, by the associated volume group, as their RAID disks. Each RAID logical volume has a number of stripes and a stripe size associated. Red Hat Enterprise Linux 8 supports RAID 0, 1, 4, 5, 6, and 10 for logical volumes.

When a RAID logical volume is created with a certain number of stripes, LVM creates the same number of data subvolumes as stripes defined, plus a parity subvolume if the RAID type enables partity. In addition to those data and parity subvolumes, LVM creates a metadata subvolume per each data or parity subvolume.

The following command creates a RAID 0 logical volume with a size of 3 GB. This logical volume uses three stripes, so it assumes that the underlying volume group, **raidvg** in this example, uses three disks as their physical volumes.

```
[root@host ~]# lvcreate --type raid0 -L 3G --stripes 3 --stripesize 4 \
-n raidlv raidvg
```

Note

The **lvconvert** command converts non-RAID logical volumes into RAID logical volumes.

References

mdadm manual page

For more information, refer to the *Configuring RAID logical volumes* chapter in the *Red Hat Enterprise Linux 8 Configuring and managing logical volumes guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_logical_volumes

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Reviewing RAID Fundamentals

In this exercise, you will calculate efficient RAID structures to match application transaction size.

Outcomes

You should be able to configure a RAID 0 device to optimize the data writing rate.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab storage-raid start** command. The command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab storage-raid start
```

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Write some data to **vda** to determine the writing performance for **vda**. When done, remove the **/tmp/testfile** file.

   2.1. Create a 512 MB file named **/tmp/testfile** with **fio**. Review the current **bw** value in **WRITE** for the data writing rate. Your value may vary.

```
[root@servera ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 --group_reporting \
--filename=/tmp/testfile
...output omitted...
  WRITE: bw=20.1MiB/s (21.0MB/s), 20.1MiB/s-20.1MiB/s (21.0MB/s-21.0MB/s),
 io=1024MiB (1074MB), run=51045-51045msec
...output omitted...
```

   2.2. Remove the **/tmp/testfile** file.

```
[root@servera ~]# rm -f /tmp/testfile
```

3. Create a RAID 0 with **vdb** and **vdc**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3.1. Create a GPT partition on **vdb** and **vdc**. Use the **--align optimal** option to align the partition for best performance. Run the following commands for both disk devices.

```
[root@servera ~]# parted /dev/vdb mklabel gpt
[root@servera ~]# parted /dev/vdc mklabel gpt
[root@servera ~]# parted -a optimal /dev/vdb mkpart primary 0% 100%
[root@servera ~]# parted -a optimal /dev/vdc mkpart primary 0% 100%
```

3.2. Reload the partition table.

```
[root@servera ~]# partx -uv /dev/vdb
...output omitted...
[root@servera ~]# partx -uv /dev/vdc
...output omitted...
```

3.3. Create a RAID 0 device, named **/dev/md0**, with the new partitions on **vdb** and **vdc**.

```
[root@servera ~]# mdadm -C /dev/md0 -l raid0 -n 2 /dev/vd[b-c]1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

3.4. Verify the chunk size for the RAID 0 device is **512 K**.

```
[root@servera ~]# mdadm --detail /dev/md0
...output omitted...
        Chunk Size : 512K
...output omitted...
```

4. Create an XFS filesystem on the RAID 0 device. Mount it in the **/raid** directory

4.1. Create an XFS filesystem on the RAID 0 device with the previous chunk size.

```
[root@servera ~]# mkfs -t xfs -d su=512k,sw=2 /dev/md0
...output omitted...
```

4.2. Create the **/raid** directory.

```
[root@servera ~]# mkdir /raid
```

4.3. Mount the RAID 0 device in the **/raid** directory.

```
[root@servera ~]# mount /dev/md0 /raid
```

5. Write some data to the RAID 0 device to determine its writing performance.

5.1. Create a 512 MB file named **/raid/testfile** with **fio**. Review the current **bw** value in **WRITE** for the data writing rate. Your value may vary. Note the increase in the data writing rate.

```
[root@servera ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 --group_reporting \
--filename=/raid/testfile
...output omitted...
  WRITE: bw=23.1MiB/s (24.2MB/s), 23.1MiB/s-23.1MiB/s (24.2MB/s-24.2MB/s),
 io=1024MiB (1074MB), run=44414-44414msec
...output omitted...
```

5.2. Log out from **servera** to return to **workstation** as **student**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab storage-raid finish** command to finish this exercise. The command removes the RAID 0 configuration on **vdb** and **vdc**.

```
[student@workstation ~]$ lab storage-raid finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Selecting I/O Analysis Tools

Objectives

After completing this section, you should be able to describe multiple tool choices for analyzing disk I/O behavior.

- Analyze the root cause of a storage I/O bottleneck.
- Identify the storage device and processes impacted by a storage I/O bottleneck.

Diagnosing Storage I/O Scenarios

Diagnosing performance issues on storage systems is a complex task that may require the involvement of more than one team. Usually, a problem with storage is detected at the application level, with extremely high execution times. The application may be using a SAN or NAS device or system, on which the response times may look normal. To determine the root cause for the performance problem, RHEL 8 includes tools to help identify issues from the client system accessing the storage, either local or remote.

First, identify the current status of the system using, for example, the **top** command. Storage system related issues typically show high CPU load averages compared to the number of cores available, while at the same time there is low CPU usage (**us** field), and high CPU wait (**wa** field), as in this two-core example.

```
[root@host ~]# top
top - 10:21:50 up 21 min,  1 user,  load average: 3,25, 3,74, 3,92
Tasks: 101 total,   2 running,  99 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,3 us,  1,7 sy,  0,0 ni, 99,2 id,  95,2 wa,  0,3 hi,  0,2 si,  0,2 st
...output omitted...
```

The example behavior may indicate an I/O bottleneck in the storage system, either local or remote. The storage is not able to deliver sn appropriate I/O operations rate for the data read and write operations performed by the operating system. Those backlog of I/O operations causes CPU load to increase, because Linux load averages include disk waiting activity. This behavior impacts all applications that use the affected storage system.

When the I/O bottleneck is confirmed, identify the storage device impacted, and which applications are causing the bottleneck (e.g. excessive writing of log entries to a storage device). RHEL 8 provides a collection of tools to analyze storage I/O, and identify both the impacted device and application.

The iostat Utility

Next, identify the storage device impacted by the I/O bottleneck. Verify that the storage system is operating properly by performaing generic tests using tools provided by the storage system. If the storage server responds normally, then focus on the client system. Use a tool that provides an insight of I/O load per storage device mapped to the system. RHEL8 includes the iostat utility. The -x option provides an extended report of I/O statistics per storage device.

```
[root@host ~]# iostat -x
...output omitted...
Device   ...output.omitted...   %util
vda      ...output.omitted...   111,51
...output omitted...
```

Note
You can also restrict statistics per device with the -d option.

The first line of output of the sysstat package utilities is a summary since the most recent boot, which shows the mean for the entire time the system has been up. Subsequent lines are displayed as per second statistics, calculated using the interval specified on the command line.

With **iostat**, look at the **%util** statistic. This statistic reflects the utilization for the storage device. Utilization over 100% may reflect an issue with that storage device, and may be an I/O bottleneck behavior. The previous output shows a storage device with a utilization higher than 100% and appears to indicate the I/O bottleneck.

The iotop Utility

The **iotop** utility can help correlate I/O statistics with processes or threads currently running on the system. Invoking **iotop** without options displays the percentage of total system I/O operations per process.

```
[root@host ~]# iotop
Total DISK READ: 7.00 M/s | Total DISK WRITE: 19.33 M/s
TID PRIO USER DISK READ DISK WRITE SWAPIN IO> COMMAND
12738 be/4 root 7.91 M/s 8.03 M/s 0.00 % 62.93 % fio ...output omitted...
```

Use options to focus the **iotop** output. The **-o** (**--only**) option shows processes and threads currently performing I/O operations, **-P** (**--processes**) shows processes, and **-a** (**--accumulated**) shows the total I/O operations performed by a process or thread since **iotop** started. In the previous example, review the disk read and write speeds to determine if the bottleneck root cause is a low-performance disk, or an application writing in a non-sustainable way to the storage device (e.g. keeping large log files).

The blktrace Toolset

The **blktrace** toolset is used to observe I/O operations inside a block I/O request. This toolset consists of **blktrace**, to log, in binary format, events related to each I/O operation in a block I/O request, the **blkparse** utility to parse the binary files generated by blktrace, and the **btt** utility to generate aggregated statistics from the **blkparse** output.

The **blktrace** toolset provides useful information to identify the I/O operations for a specific device, by using the the **-d** option. The **btt** command provides the useful queue-to-completion **Q2C** aggregate that reports total I/O time, including the I/O operation wait time, the total I/O operation service time, and the device-to-completion **D2C** aggregate, which is the service time for the device. Another tool, **blkiomon**, generates per-device request size and latency from the **blktrace** output.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Analyzing Storage Performance with Performance Co-pilot

The Performance Co-pilot toolset also provides some utilities to analyze the storage system performance. Tools in the PCP toolset are typically more powerful that their similarly named versions in the **`systat`** utilities. Red Hat

pcp atop

Provides information similar tp both the **`iotop`** and **`atop`** commands. **`pcp-atop`** is a port of the actual atop source to the PCP APIs. The command lists the processes that are performing I/O, along with the disk bandwidth they are using. The command distinguishes between total values that are the amount of data read from or written to the disk block device driver, and actual values that are the numbers for the actual hardware disk I/O. File system caching is one of the reasons for the difference in the displayed values.

`pcp-atop` is good for quickly grasping changes occuring on a system. It does an excellent job of summarizing changes in each interval. **`pcp-atop`** can list all the processes that caused any system-level changes, like doing disk I/O, during the specified interval.

pmiostat

The **`pmiostat`** command reports live disk I/O statistic for the local host, but can also report for a remote host when the **`-h`** option is used, or can replay PCP archive logs with the **`-a`** option. Use the **`-R`** option to specify a regular expression to limit statistics to storage devices matching the expression.

The command reports on SCSI devices by default. Use the **`-x dm`** option to report statistics for device-mapper logical devices. Eliminate idle devices from the output by using the **`-x noidle`**

References

`iostat`, **`iotop`**, **`blktrace`**, **`blkparse`**, **`btt`**, **`blkiomon`** manual pages

Knowledgebase: "Side-by-side comparison of PCP tools with legacy tools"
https://access.redhat.com/articles/2372811

Introduction to storage performance analysis with PCP
https://access.redhat.com/articles/2450251

Guided Exercise

Selecting I/O Analysis Tools

In this exercise, you will analyze disk behavior in various use cases.

Outcomes

You should be able to measure storage performance with the **iotop**, **iostat**, and Performance Co-pilot utilities.

Before You Begin

Log in as the **student** user on **workstation** using **student** as the password.

Run the **lab storage-analysis start** command. The command determines if the host, **servera**, is reachable on the network and prepares it for this exercise.

```
[student@workstation ~]$ lab storage-analysis start
```

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Install the **iotop** utility.

```
[root@servera ~]# yum install iotop
...output omitted...
```

3. Run **iotop** to display disk I/O workload for all processes.

```
[root@servera ~]# iotop
Total DISK READ : 0.00 B/s | Total DISK WRITE :     393.18 M/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:     413.39 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
...output omitted...
```

Click **Ctrl**+**C** to exit **iotop**.

4. Install the **iostat** utility. The *sysstat* package includes this utility.

```
[root@servera ~]# yum install sysstat
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Run the **iostat** command to identify which disks have I/O workload. The output shows that **vda** has I/O workload.

```
[root@servera ~]# iostat
Linux 4.18.0-80.el8.x86_64 (servera.lab.example.com)  26/07/19  _x86_64_(2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,28    0,05    0,85    0,15    0,03   98,65

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda               3,88        57,12      1339,08     636092   14912427
vdb               0,01         0,20         0,00       2224          0
vdc               0,01         0,20         0,00       2224          0
vdd               0,01         0,20         0,00       2224          0
```

6. Run the **iostat** command with the **-x** option to display the extended statistics. Use the **-d** option to display the statistics for **vda**.

```
[root@servera ~]# iostat -xd vda
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm
vda              1,50    2,28     55,57   1302,70     0,00     0,20   0,29

%wrqm r_await w_await aqu-sz rareq-sz wareq-sz  svctm  %util
8,24    4,23    6,20   0,02    37,11   570,69   0,65   0,24
```

7. Use the Performance Co-pilot utility to retrieve live I/O statistics for **vda**.

   7.1. Install the *pcp-system-tools* package.

```
[root@servera ~]# yum install pcp-system-tools
...output omitted...
```

   7.2. Start and enable the **pmcd** service.

```
[root@servera ~]# systemctl enable --now pmcd
```

   7.3. Retrieve live disk I/O statistics with the **pmiostat** command. Use the **-R** option to limit the live I/O statistics to devices matching the regular expression **vda**.

```
[root@servera ~]# pmiostat -R vda
# Device      rrqm/s  wrqm/s     r/s    w/s    rkB/s    wkB/s avgrq-sz avgqu-sz
 await r_await w_await   %util
vda             0.00    0.00    0.00   0.00     0.00     0.00    0.000    0.000
 0.00    0.00    0.00    0.00
vda             0.00    0.00    0.00   1.00     0.00   191.20  192.000    0.001
 1.00    0.00    1.00    0.20
vda             0.00    0.00    0.00   0.00     0.00     0.00    0.000    0.000
 0.00    0.00    0.00    0.00
vda             0.00   15.95    0.00  42.86     0.00   690.74   16.116    0.031
 0.72    0.00    0.72    0.50
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Click **Ctrl+C** to exit **pmiostat**.

7.4. Run the **pcp atop** command to retrieve I/O workload per process, and additional system performance information useful to troubleshoot performance issues. Use the **-d** option to display disk-related output.

```
[root@servera ~]# pcp atop -d
ATOP - servera.lab.example.com 2019/07/26  08:55:55 - 1s elapsed
...output omitted...
DSK |          vda  | busy 0%  |               | read   19869 |  write  31042 |
 KiB/r     35 | KiB/w    522  |               | MBr/s  696.5  | MBw/s 15847.8 |
 avq     7.18 |               |  avio 0.63 ms
DSK |          vdd  | busy 0%  |               | read 99 |  write  0 |  KiB/r
 22 | KiB/w 0  |               | MBr/s    2.2  | MBw/s    0.0 |  avq     1.98 |
            |  avio 0.55 ms |
DSK |          vdb  | busy 0%  |               | read 99 |  write  0 |  KiB/r
 22 | KiB/w 0  |               | MBr/s    2.2  | MBw/s    0.0 |  avq     0.00 |
            |  avio 0.40 ms |
DSK |          vdc  | busy 0%  |               | read 99 |  write  0 |  KiB/r
 22 | KiB/w 0  |               | MBr/s    2.2  | MBw/s    0.0 |  avq     0.00 |
            |  avio 0.33 ms |
...output omitted...
        *** system and process activity since boot ***
  PID TID RDDSK WRDSK WCANCL DSK CMD 1/4
...output omitted...
```

Click **q** to exit **pcp atop**.

8. Log out from **servera** to return to **workstation** as **student** user.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab storage-analysis finish** command to finish this exercise. The command removes the utilities for this exercise.

```
[student@workstation ~]$ lab storage-analysis finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning Storage Device I/O

Performance Checklist

In this lab, you will configure I/O tunables for a variety of storage performance tuning scenarios.

Outcomes

You should be able to:

- Measure latency and data writing rate for a disk.
- Use a tuned profile to customize a tunable for a disk I/O scheduler.
- Configure a RAID 0 with two disks.

Before You Begin

Log in to **`workstation`** as **`student`** using **`student`** as the password.

On **`workstation`**, run the **`lab storage-review start`** command. The command determines if the host, **`serverb`**, is reachable on the network and ensures that the default tuned profile is active.

```
[student@workstation ~]$ lab storage-review start
```

1. On **`workstation`**, use **`ssh`** to log in to **`serverb`** as **`student`**. Use **`sudo -i`** to switch to **`root`**.
2. Determine the latency for **`vda`**.
3. Create a new custom tuned profile named **`reduce-latency`**. Decrease the batch size for operations to reduce latency. Inherit settings from the **`latency-performance`** profile.
4. Enable the newly created tuned profile **`reduce-latency`**. Verify that the custom settings for the profile are enabled.
5. Write some data to **`vda`** to verify that the latency is now lower.
6. Create a RAID 0 with **`vdb`** and **`vdc`**.
7. Create an XFS filesystem on the RAID 0 device, and mount it in the **`/raid`** directory
8. Write some data to the RAID 0 device to determine its writing performance.

Evaluation

On **`workstation`**, run the **`lab storage-review grade`** command to confirm success of this exercise.

```
[student@workstation ~]$ lab storage-review grade
```

Finish

On **`workstation`**, run the **`lab storage-review finish`** command to complete this exercise.

```
[student@workstation ~]$ lab storage-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning Storage Device I/O

Performance Checklist

In this lab, you will configure I/O tunables for a variety of storage performance tuning scenarios.

Outcomes

You should be able to:

- Measure latency and data writing rate for a disk.
- Use a tuned profile to customize a tunable for a disk I/O scheduler.
- Configure a RAID 0 with two disks.

Before You Begin

Log in to **`workstation`** as **`student`** using **`student`** as the password.

On **`workstation`**, run the **`lab storage-review start`** command. The command determines if the host, **`serverb`**, is reachable on the network and ensures that the default tuned profile is active.

```
[student@workstation ~]$ lab storage-review start
```

1. On **`workstation`**, use **`ssh`** to log in to **`serverb`** as **`student`**. Use **`sudo -i`** to switch to **`root`**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

2. Determine the latency for **`vda`**.

   2.1. Install the **`fio`** utility.

```
[root@serverb ~]# yum install fio
...output omitted...
```

   2.2. Create a 512 MB file named **`/tmp/testfile`** with **`fio`**. Review the current value for **`avg`** in **`clat (nsec)`**.

```
[root@serverb ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 --group_reporting \
--filename=/tmp/testfile
...output omitted...
    clat (usec): min=0, max=51154k, avg=262584.65, stdev=314735.28
...output omitted...
```

2.3. Remove the **/tmp/testfile** file.

```
[root@servera ~]# rm -f /tmp/testfile
```

3. Create a new custom tuned profile named **reduce-latency**. Decrease the batch size for operations to reduce latency. Inherit settings from the **latency-performance** profile.

3.1. Create the directory **/etc/tuned/reduce-latency** for the new custom profile.

```
[root@serverb ~]# mkdir /etc/tuned/reduce-latency
```

3.2. Inherit the settings for the new profile from the existing profile **latency-performance**. Create the **/etc/tuned/reduce-latency/tuned.conf** file, and add the following two statements.

```
[main]
include=latency-performance
```

3.3. Decrease the batch size for operations sent to **vda**, to reduce latency. Set the **/sys/block/vda/queue/iosched/fifo_batch** tunable to **1**. Add the following to the **/etc/tuned/reduce-latency/tuned.conf** file using the **sysfs** plug-in.

```
[sysfs]
/sys/block/vda/queue/iosched/fifo_batch=1
```

4. Enable the newly created tuned profile **reduce-latency**. Verify that the custom settings for the profile are enabled.

4.1. Enable the **reduce-latency** profile.

```
[root@serverb ~]# tuned-adm profile reduce-latency
```

4.2. Verify that the custom value for **/sys/block/vda/queue/iosched/fifo_batch** is active on **serverb**.

```
[root@serverb ~]# cat /sys/block/vda/queue/iosched/fifo_batch
1
```

5. Write some data to **vda** to verify that the latency is now lower.

5.1. Create a 512 MB file named **/tmp/testfile** with **fio**. Review the current value for **avg** in **clat (nsec)**. The latency should now be lower.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 --group_reporting \
--filename=/tmp/testfile
...output omitted...
    clat (nsec): min=0, max=41126k, avg=215403.66, stdev=239728.96
...output omitted...
```

6. Create a RAID 0 with **vdb** and **vdc**.

6.1. Create a GPT partition on **vdb** and **vdc**. Use the **`--align optimal`** option to align the partition for best performance. Run the following commands for both disk devices.

```
[root@serverb ~]# parted /dev/vdb mklabel gpt
[root@serverb ~]# parted /dev/vdc mklabel gpt
[root@serverb ~]# parted -a optimal /dev/vdb mkpart primary 0% 100%
[root@serverb ~]# parted -a optimal /dev/vdc mkpart primary 0% 100%
```

6.2. Reload the partition table.

```
[root@serverb ~]# partx -uv /dev/vdb
...output omitted...
[root@serverb ~]# partx -uv /dev/vdc
...output omitted...
```

6.3. Create a RAID 0 device, named **`/dev/md0`**, with the new partitions on **vdb** and **vdc**.

```
[root@serverb ~]# mdadm -C /dev/md0 -l raid0 -n 2 /dev/vd[b-c]1
```

6.4. Verify the chunk size for the RAID 0 device is **`512 K`**.

```
[root@serverb ~]# mdadm --detail /dev/md0
...output omitted...
        Chunk Size : 512K
...output omitted...
```

7. Create an XFS filesystem on the RAID 0 device, and mount it in the **`/raid`** directory

7.1. Create an XFS filesystem on the RAID 0 device with the previous chunk size.

```
[root@serverb ~]# mkfs -t xfs -d su=512k,sw=2 /dev/md0
...output omitted...
```

7.2. Create the **`/raid`** directory.

```
[root@serverb ~]# mkdir /raid
```

7.3. Mount the RAID 0 device in the **`/raid`** directory.

```
[root@serverb ~]# mount /dev/md0 /raid
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

8. Write some data to the RAID 0 device to determine its writing performance.

8.1. Create a 512 MB file named **/raid/testfile** with **fio**. Review the current **bw** value in **WRITE** for the data writing rate.

```
[root@serverb ~]# fio --name=randwrite --ioengine=libaio --iodepth=1 \
--rw=randwrite --bs=4k --direct=1 --size=512M --numjobs=2 --group_reporting \
--filename=/raid/testfile
...output omitted...
  WRITE: bw=23.1MiB/s (24.2MB/s), 23.1MiB/s-23.1MiB/s (24.2MB/s-24.2MB/s),
 io=1024MiB (1074MB), run=44414-44414msec
...output omitted...
```

8.2. Log out from **serverb** to return to **workstation** as **student** user.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab storage-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab storage-review grade
```

Finish

On **workstation**, run the **lab storage-review finish** command to complete this exercise.

```
[student@workstation ~]$ lab storage-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- Red Hat Enterprise Linux 8 includes by default a new collection of multiple queue I/O schedulers for disks which replace the old single-queue based I/O schedulers.
- The Tuned profiles support the tuning of storage systems through plugins like **`disk`** or **`sysfs`**.
- RAID is an array of disks, usually cheap disks, configured together as a single logical storage unit to support increased performance, redundancy, and fault tolerance.
- Red Hat Enterprise Linux supports the creation and management of software RAID with the **`mdadm`** utility.
- Several utilities are available to diagnose I/O: **`iostat`**, **`iotop`**, **`btrace`**, **`blkparse`**, **`btt`**, and **`blkiomon`**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 10

Tuning File System Utilization

Goal

Manage application efficiency for file system utilization.

Objectives

- Describe the file system choices and tunable attributes.
- Configure file system journaling and discuss appropriate use cases.

Sections

- Managing File System Attributes (and Guided Exercise)
- Managing File System Journaling (and Guided Exercise)

Lab

Tuning File System Utilization

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Managing File System Attributes

Objectives

After completing this section, you should be able to describe the file system choices and tunable attributes.

Local File Systems

Red Hat Enterprise Linux offers system administrators a choice of two local file systems to suit the needs of their system's workload requirements, the *XFS* and the *ext4* file systems.

In RHEL 7, XFS was the default file system selection and ext4 was offered as another option. In RHEL 8, XFS continue as the default file system selection. Ext4 is still available as a file system option on RHEL 8.

The XFS File System

Originally created by Silicon Graphics, Inc., XFS is a robust and highly scalable high-performance file system. Like RHEL 7, XFS in RHEL 8 is the default file system for boot, root, and user data partitions.

With support for maximum file system and file sizes of up to 1 PB as compared to ext4's 50 TB file system limit and 8 EB file size limit, XFS is the obvious choice for large servers and high-end storage devices. Well-tuned for larger systems with several numbers of CPUs and cores, XFS will offer better performance for most workloads than ext4.

XFS offers several allocation schemes, such as extent-based allocation, stripe-aware allocation policies, delayed allocation, space pre-allocation, and dynamically allocated inodes. XFS also supports metadata journaling, which can aid in quicker crash recovery.

Unlike most other file systems, XFS does not execute file system integrity check during boot. The XFS file system can be defragmented and enlarged while mounted and active. However, it unfortunately cannot be reduced in size, even when offline.

The Ext4 File System

Ext4 is a scalable extension of the ext3 file system. The ext4 driver transparently provides for continued support of both the ext2 and ext3 file systems. In RHEL8, like in RHEL 7, the ext4 file system supports a maximum size of 50 TB for the file system and maximum file size of 16 TB.

Ext4 uses extent allocation and offers the following allocation schemes: persistent pre-allocation, delayed allocation, multi-block allocation, and stripe-aware allocation. While XFS is regarded as the reigning file system champion for large systems and high-performance storage devices, ext4 can deliver better performance than XFS for some specific workloads, such as single-threaded, single-disk workloads with synchronous updates, and small files in size.

Like XFS, ext4 file systems can be grown in size while online. However, while the XFS file system currently does not support shrinking of the file system in either online or offline states, ext4 file systems can be reduced in size when unmounted.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Local File System Recommendation

In general, Red Hat recommends that you use XFS unless you have a specific use case for ext4.

In the following table, you will have a summary guide to evaluate which file system suits better for your use case. Remember that you should also measure the performance of your specific application to make sure that you choose the appropriate type of file system.

Local File System Recommendations

Scenario	Recommended File System
No special use case	XFS
Large server	XFS
Large storage devices	XFS
Large files	XFS
Multi-threaded I/O	XFS
Small files	ext4
Single-threaded I/O	ext4
Limited I/O capability (under 1000 IOPS)	ext4
Limited bandwidth (under 200 MB/s)	ext4
CPU-bound workload	ext4
Support for offline shrinking	ext4

SSD File System Recommendations

Solid state drives can apply the same recommendations that we listed in the previous table. Both the ext4 and XFS file systems support the TRIM commands or discard operations.

In RHEL, you can choose three methods to apply the discard operations, batch discard, scheduled batch discard, and online discard.

Batch discard

Batch discard operations are run on demand by the user executing the **`fstrim`** command.

To run a batch discard operations on the **`/mymountpoint`** mount point, execute the following command:

```
[root@host ~]# fstrim /mymountpoint
```

Scheduled batch discard

Scheduled batch discard operations benefit from using the **`systemd`** unit **`fstrim.timer`** to run a **`fstrim`** command weekly in all mount points available. The ones that have an underlying device with TRIM support would have a discard operation of the unused data blocks. This scheduled discard operations runs without user intervention

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

To enable the scheduled batch discard operations with the **fstrim.timer systemd** unit, execute the following command:

```
[root@host ~]# systemctl enable --now fstrim.timer
```

Online discard

Online discard operations are specified at mount time. This online discard operations run in real time without user intervention.

To enable the online discard operations at mount time with the **mount** command, execute the following:

```
[root@host ~]# mount -o discard /dev/mydevice /mymountpoint
```

To enable the online discard operation at mount time persistently, you can add a line like the following in your **/etc/fstab** configuration file:

```
[root@host ~]# vim /etc/fstab
/dev/mydevice  /mymountpoint  xfs  defaults,discard  0 0
```

Even if the use of the discard operations improves the overall performance of the specified device, the overall performance of the device is impacted while it is performing this process. For this reason, as well as the wear on the SSD, Red Hat recommends the use of the scheduled batch discard (once a week) or the batch discard (by demand) over the online discard if the workload or the maintenance of the performance allows it.

File System Formatting Options

The file system creation options are, in some cases, the only way to tune or set a file system for a specific use case or workload. The ext4 and the XFS file systems each have their own sets of options.

XFS Formatting Options

The following formatting options are the most common for a use case of the XFS file system:

Inode size

If the target use case make extensive use of extended attributes, it is recommended to increase the XFS inode size to 512 bytes. By default, the inode size is set in 256 bytes.

Use the following command to set the inode size at file system creation time:

```
[root@host ~]# mkfs.xfs -i size=512 /dev/mydevice
```

Logical Block Size for the Directory

Increasing the logical block size for the directories from the default 4096 Bytes will decrease the directory I/O and will improve the performance of directory operations.

Use the following command to set the logical block size for the directories at file system creation time:

```
[root@host ~]# mkfs.xfs -n size=8192 /dev/mydevice
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

RAID Alignment

To align the I/O at the file system layer, it is important to set the correct stripe unit (stripe element size) and stripe width (number of data disks) while formatting the file system. These options are sometimes auto-detected but manual configuration is required for many of the hardware RAID volumes.

For example, to align a RAID-0 device with two data disks and a chunk size of 512 K with the XFS file system on top, execute the following command:

```
[root@host ~]# mkfs.xfs -d su=512k,sw=2 /dev/myraid
```

ext4 Formatting Options

The following formatting options are the most common for a use case of the ext4 file system:

Inode size

In a case of use of the ext4 file system where small files predominate and there is no use of extended attributes, inode size can be reduced to optimize disk space. The default inode size is 256K.

Use the following command to set a reduced inode size at file system creation time:

```
[root@host ~]# mkfs.ext4 -I 128 /dev/mydevice
```

`extra_isize`

The **`extra_isize`** feature reserves a specific amount of space in each inode for extended metadata. If we use the reduced inode size (less than 256K), this option has no effect and should be disabled. This option is enabled by default.

Use the following command to disable the **`extra_isize`** feature at file system creation time:

```
[root@host ~]# mkfs.ext4 -O ^extra_size /dev/mydevice
```

`large_dir`

The **`large_dir`** feature increases the limit of the number of files per directory by raising the maximum size of directories. This option is disabled by default.

Use the following command to enable the **`large_dir`** feature at file system creation time:

```
[root@host ~]# mkfs.ext4 -O large_dir /dev/mydevice
```

`huge_file`

The **`huge_file`** feature allows files to be larger than 2 TB in size. For a typical use case of the ext4 file system where the small files are predominant, this feature should be disabled. This option is enabled by default.

Use the following command to disable the **`huge_file`** feature at file system creation time:

```
[root@host ~]# mkfs.ext4 -O ^huge_file /dev/mydevice
```

RAID Alignment

To align the I/O at the file system layer, it is important to set the correct stripe unit (stripe element size) and stripe width (number of data disks) while formatting the file system. These

options are sometimes auto-detected but manual configuration is required for many of the hardware RAID volumes.

For example, to align a RAID-0 device with two data disks and a chunk size of 512 K with the ext4 file system on top, execute the following command:

```
[root@host ~]# mkfs.ext4 -E stride=128,stripe-width=256 /dev/mydevice
```

File System Mount Options

Another group of settings that can be tuned on ext4 and XFS file systems are the mount options. There are some common options and also specific options for each file system.

Common Mount Options

Some of the common mount options for the XFS and the ext4 file system are related to the update of the inode access time to files and directories.

`atime`
This mount option grants control of the update inode access time to the kernel default behavior.

`relatime`
This mount option updates the inode access times if the files or directories are modified or changed, if not it would update the access time once in a day and not every time you access the files or directories. This option is enabled by default on XFS and ext4 file systems.

`noatime`
This mount option does not update the inode access times on this filesystem, applies for files and directories.

`nodiratime`
This mount option does not update the inode access times on this filesystem, only applies for directories.

XFS Mount Options

The XFS file systems in RHEL 8 comes with a set of predefined options, which make their performance optimal. However, a few specific options that we can recommend depending on the use case are:

`inode64`
With the **`inode64`** allocation strategy, inodes are placed near the data which minimizes disk seeks. The **`inode64`** option is enabled by default in RHEL 8.

`logbsize`
The **`logbsize`** handles the log buffer size on XFS file systems. The larger the **`logbsize`** is, the fewer I/O operations it takes to write all changes to the log. A larger log buffer can improve performance on systems with I/O-intensive workloads that do not have a nonvolatile write cache.

The default log buffer size is 32 KB. Other common log buffer sizes that can be configured are 64 KB, 128 KB, and a maximum of 256 KB.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

ext4 Mount Options

Like XFS, the ext4 file systems in RHEL 8 come with a set of predefined options, which make their performance optimal. However, a few specific options that we can recommend depending on the use case are:

`i_version`
: The **`i_version`** mount option will enable 64-bit inode version support, which is useful for extensive use cases of extended attributes.

  The **`i_version`** is disabled by default.

`journal_ioprio`
: The **`journal_ioprio`** mount option controls the priority of journal I/O. Values from 0 to 7, with 0 being the highest priority I/O.

  The **`journal_ioprio`** mount option has a default value of 3, which is slightly higher than the priority of normal I/O.

Benchmarking File System Performance

With the file system choices available in RHEL 8, system administrators can either accept the default XFS file system selection or explore the ext4 file system when allocating file systems onto new disk devices. While XFS offers better performance than ext4 for most workloads, ext4 will yield superior performance in certain scenarios.

Systems allocated as file servers and database servers can have varying workload characteristics. These characteristics can include things such as the size of files created or accessed, the frequency and concurrency of I/O operations, and the ratio of read operations to write operations. Even servers allocated to the same role may impose very different workloads at the file system layer due to different transaction volumes and configuration differences, such as debugging or logging levels of hosted services. To accurately evaluate which file system is best suited for the workload on a particular system, administrators need to devise a method for benchmarking file system performance for that system's workload.

While utilities such as **`dd`**, **`hdparm`**, and **`bonnie++`** are useful for profiling the performance of storage devices, the results of their throughput testing have little relevance to the real-world workload imposed by the services running on a system. For benchmarking results to be relevant, they must be obtained using benchmarking routines that simulate real-world workloads at the file system layer. In order to do this, administrators need to develop tests that simulate the nature of the I/O operations that will be imposed by the system's workload. For a file server, this may involve the upload or download of large files. For a database server, this may involve a dump of databases, execution of a large number of concurrent database queries, or the execution of a large database query with complex joins.

If benchmark tests are well-designed, they will yield measurable, relevant, and meaningful results. By running such tests to impose simulated workloads on different file systems, administrators can compare the results and clearly identify which file system yields the better performance, as well as calculate the margin of the performance difference. Depending on how large the performance difference is, administrators may or may not choose the higher-performing file system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

`ext4`(5), **`mount`**(8), **`mkfs.ext4`**(8), **`mkfs.xfs`**(8), **`tune2fs`**(8), **`umount`**(8), and **`xfs`**(5) manual pages

Knowledgebase: "What are some of best practices for tuning XFS filesystems"
https://access.redhat.com/solutions/203173

Knowledgebase: "Red Hat Enterprise Linux technology capabilities and limits"
https://access.redhat.com/articles/rhel-limits

For more information, refer to the *Overview of available file systems* chapter in the *Red Hat Enterprise Linux 8 Managing File Systems guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/managing_file_systems

For more information, refer to the *Storage and File Systems* chapter in the *Red Hat Enterprise Linux 7 Performance Tuning guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index#chap-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Storage_and_File_Systems

XFS FAQ
http://xfs.org/index.php/XFS_FAQ

XFS FITRIM/discard
http://xfs.org/index.php/FITRIM/discard

Guided Exercise

Managing File System Attributes

In this exercise, you will configure attributes for an XFS file system.

Outcomes

You should be able to configure a tuned XFS file system for a specific workload.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab fs-attributes start** command. The command determines if the host, **servera**, is reachable on the network, and prepares it for this exercise.

```
[student@workstation ~]$ lab fs-attributes start
```

You need to configure an XFS file system that has a specific use case with the following characteristics:

- The file system would be used to process raw video files that are mainly large in size.
- The processes that use the data inside the file system allows the use of multi-threaded I/O.
- Increased inode size for better support to extended attributes.
- The logging of access times in directories is not needed.

For this exercise, your system will work with a preconfigured RAID 0 **md0** device, with **vdb** and **vdc** disks as arrangement members. Configure the tuned XFS file system as follows:

- The provisioned RAID 0 has two disk members and a chunk size of 512 K.
- The XFS file system should have a block size of 4096 bytes.
- Ensure the new XFS file system has the **inode** size increased to 512 bytes for extended attributes compatibility.
- Persistently mount the newly XFS file system on **/xfsraid**.
- Use the **nodiratime** mount option to disable the logging of inode access times on directories inside the XFS file system.
- Configure the logical block size for directories to 8192 bytes to decrease the directory I/O and improve the performance of directory operations.

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Create an XFS file system aligned to the underlying RAID 0 device, according to the requirements.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

2.1. Create an XFS file system on the **md0** device. Configure the file system to be aligned to the provisioned info of the underlying RAID 0 device. Set the block size to 4096 bytes, the logical block size for directories to 8192 bytes, and the **inode** size to 512 bytes.

```
[root@servera ~]# mkfs.xfs -f -d su=512k,sw=2 -b size=4096 \
-n size=8192 -i size=512 /dev/md0
...output omitted...
```

3. Create the **/xfsraid** mount point and temporarily mount the new file system.

3.1. Create the mount point at **/xfsraid**.

```
[root@servera ~]# mkdir /xfsraid
```

3.2. Mount the new file system to **/xfsraid**.

```
[root@servera ~]# mount /dev/md0 /xfsraid
```

3.3. Unmount the new file system.

```
[root@servera ~]# umount /dev/md0
```

4. Mount the new filesystem persistently using the required mounting options, then logout from **servera**.

4.1. Configure an entry in **/etc/fstab** to persistently mount the newly created filesystem, include the **nodiratime** mount option.

```
[root@servera ~]# vim /etc/fstab
...output omitted...
/dev/md0  /xfsraid  xfs  nodiratime  0 0
```

4.2. Mount the newly created XFS file system.

```
[root@servera ~]# mount /dev/md0
```

4.3. Verify the mounting options of the newly created XFS file system.

```
[root@servera ~]# findmnt --target /xfsraid
TARGET   SOURCE   FSTYPE OPTIONS
/xfsraid /dev/md0 xfs
 rw,nodiratime,relatime,seclabel,attr2,inode64,sunit=1024,swidth=2048,noquota
```

4.4. Logout from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Finish

On **`workstation`**, run the **`lab fs-attributes finish`** command to complete this exercise.

```
[student@workstation ~]$ lab fs-attributes finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Managing File System Journaling

Objectives

After completing this section, you should be able to configure file system journaling and discuss appropriate use cases.

Journal Placement

Having a journal in a file system can speed up file system recovery, something that becomes extremely important when dealing with larger file systems. A journal speeds file system recovery by being a sort of *logbook* for the file system. Whenever a change is about to be made to the file system, the transaction will be recorded in the journal. After the operation is completed, the journal entry is removed. This means that after a power loss, the recovery process only needs to check the journal (and replay the journal if necessary) and the parts of the file system affected by the journal, instead of the entire file system.

XFS offers a **`norecovery`** option for the **`mount`** command. When this option is specified for an XFS file system during mounting, log recovery will be disabled. If the file system is not cleanly mounted, it is likely to be inconsistent, and may possibly result in some files or directories becoming inaccessible. Therefore, XFS file systems mounted with the **`norecovery`** option can only be mounted read-only.

The **`ext3/ext4`** journal can work in three different modes, chosen by passing the **`data=`*`mode`*** option to the file system at mount time. The default mode is **`ordered`**. The three modes are:

ordered
: Only metadata is recorded in the journal. Journal entries will only be committed after all file data has been flushed to disk.

writeback
: Only metadata is recorded in the journal, but data ordering is not preserved. This is rumored to be the fastest method, and internal file system integrity is guaranteed. But old data may show up inside files after a crash and recovery.

journal
: All data is first stored in the journal before being written out to disk. This gives the best reliability for the file system but can be the worst performer in most workloads. In workloads with many small writes, this could help increase performance since the disk elevator gets a chance to group writes together.

Barriers

Write barriers are a way of enforcing write ordering on devices with a supported on-device write cache. When barriers are enabled, the commit ordering between data and journal entries is guaranteed, ensuring a recoverable file system in case of a crash.

Both the XFS and ext4 file systems turn on barriers by default. Since kernel v4.10, the **`nobarrier`** mount option in XFS filesystem is deprecated. Since that kernel version, integrity operations are always performed and even specify that the **`nobarrier`** mount option has no effect. Now barriers are always enabled by default and can not be disabled in XFS file systems.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

To enable or disable barriers at mount time on ext4 file systems, use the **barrier/nobarrier** mounting option. You can still use the old **barrier=1** or **barrier=0** mount options, as well. On devices with a battery-backed cache and using ext4 file systems, **barriers** can be disabled for a performance improvement while still guaranteeing data integrity.

External Journal

By default, when XFS and ext4 file systems are created, their journals are placed on the same block device as the file system data. File system write operations are preceded by updates of corresponding metadata in the file system journal. With certain workloads, such as those involving a large number of random writes, there may be disk contention between the write operations and their accompanying journal updates.

In some storage configurations like the ones that are still using rotational disks, one way to improve journal performance is to place the journal on a separate device from the one that contains the file system. Placing the journal on low-latency devices such as an SSD, NVMe, or battery-backed ram-disk can significantly enhance performance. The act of moving the journal away from the main disk can reduce the number of disk seeks and improve performance for the main disk.

External Journal in XFS File Systems

To create a new XFS file system with an external journal, use the **logdev** log section option.

In the following example, assume that **/dev/vdc1** is a partition for the main file system and that **/dev/vdd1** is a partition for the journal device. The main file system and its external journal can be created with the following command.

```
[root@host ~]# mkfs.xfs -l logdev=/dev/vdd1 /dev/vdc1
```

The journal device location needs to again be specified when the file system is mounted. The location is specified using the **logdev** mount option.

```
[root@host ~]# mount -o logdev=/dev/vdd1 /dev/vdc1 /mnt
```

External Journal in ext4 File Systems

To create a new ext4 file system with an external journal, you need to create the journal device first.

> **Warning**
> The external journal for an ext4 file system must use the same file system block device size, so it is a best practice to specify the block when creating the journal device and the main file system.

In the following example, assume that **/dev/vdc1** is a 4 GiB partition for the main file system and that the **/dev/vdd1** is a 128 MiB partition for the journal device. Start by creating the ext4 journal using a block size of 4 KiB:

```
[root@host ~]# mkfs.ext4 -O journal_dev -b 4096 /dev/vdd1
```

Now that the journal device is ready, create the file system, specifying the journal device and the block size:

```
[root@host ~]# mkfs.ext4 -J device=/dev/vdd1 -b 4096 /dev/vdc1
```

Unlike the XFS file system, in an ext4 file system, you do not specify the external journal device at mounting time.

To convert an existing ext4 file system with an internal journal to an external journal, first find the existing file system block size:

```
[root@host ~]# tune2fs -l /dev/vdc1
...
Block size:               4096
...
```

Create the external journal device with the same block size as the target file system.

```
[root@host ~]# mkfs.ext4 -O journal_dev -b 4096 /dev/vdd1
[root@host ~]# umount /dev/vdc1
```

Unmount the file system and then remove the existing journal and add the external journal.

```
[root@host ~]# umount /dev/vdc1
[root@host ~]# tune2fs -O '^has_journal' /dev/vdc1
[root@host ~]# tune2fs -j -J device=/dev/vdd1 /dev/vdc1
```

Important

Make sure that the existing journal is empty before removing it, you can do that unmounting the file system cleanly or running the **`fsck`** command on the file system first.

References

`ext4`(5), **`mount`**(8), **`mkfs.ext4`**(8), **`mkfs.xfs`**(8), **`tune2fs`**(8), **`umount`**(8), and **`xfs`**(5) manual pages

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Managing File System Journaling

In this exercise, you will configure journaling for an ext4 file system.

Outcomes

You should be able to create an ext4 file system utilizing external journaling.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab fs-journal start** command. The command determines if the host, **servera**, is reachable on the network, and also validates if additional hard drives are available for the exercise.

```
[student@workstation ~]$ lab fs-journal start
```

For this exercise, your system will work with the **vdb** and **vdc** disks. Configure the file system on the disks according to the following directives:

- Create a primary 5 GiB partition on the **vdb** disk for an ext4 file system.
- Create a primary 1 GiB partition on the **vdc** disk for external journaling.
- Format the 5 GiB partition with an ext4 file system. Ensure its journal is logged on the 1 GiB partition.
- Persistently mount the new 5 GiB ext4 file system on **/journal**.

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Create the new partitions on the **vdb** and **vdc** disks.

   2.1. Create the partition label standard on the **vdb** and **vdc** disks.

```
[root@servera ~]# parted /dev/vdb mklabel gpt
[root@servera ~]# parted /dev/vdc mklabel gpt
```

   2.2. Create a primary partition of 5 GiB (100 % of available disk space) on the **vdb** disk:

```
[root@servera ~]# parted --align=optimal /dev/vdb mkpart primary ext4 0% 100%
```

   2.3. Create a primary partition of 1 GiB on the **vdc** disk.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# parted --align=optimal /dev/vdc mkpart primary ext4 0% 1GiB
```

2.4. Force a reread of the partition table.

```
[root@servera ~]# partx -uv /dev/vdb
...output omitted...
[root@servera ~]# partx -uv /dev/vdc
...output omitted...
```

2.5. List the actual block devices on the system.

```
[root@servera ~]# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0     11:0    1 1024M  0 rom
vda    252:0    0   10G  0 disk
└─vda1 252:1    0   10G  0 part /
vdb    252:16   0    5G  0 disk
└─vdb1 252:17   0    5G  0 part
vdc    252:32   0    5G  0 disk
└─vdc1 252:33   0 1023M  0 part
vdd    252:48   0    5G  0 disk
```

3. Create an ext4 file system, using an external device as journal.

3.1. First create the ext4 journal device with a block size of 4096 Bytes, on the **vdc1** 1 GiB partition

```
[root@servera ~]# mkfs.ext4 -O journal_dev -b 4096 /dev/vdc1
```

3.2. Create the ext4 file system on the 5 GiB partition. Specify the 1 GiB partition as the location for the external journal. Configure the same block size of 4096 Bytes used in the journal creation.

```
[root@servera ~]# mkfs.ext4 -J device=/dev/vdc1 -b 4096 /dev/vdb1
```

3.3. List the ext4 file system properties and look for the external journal device.

```
[root@servera ~]# tune2fs -l /dev/vdb1
...output omitted...
Journal UUID:             16946538-6aed-4b59-ad7e-bd954f395609
Journal device:           0xfc21
...output omitted...
```

3.4. Compare the **Journal UUID** property with the **lsblk** command output:

```
[root@servera ~]# lsblk -f /dev/vdc1
NAME FSTYPE LABEL UUID                                 MOUNTPOINT
vdc1 jbd          16946538-6aed-4b59-ad7e-bd954f395609
```

4. Create the mount point, and temporary mount the newly created file system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

4.1. Create the mount point at **/journal**.

```
[root@servera ~]# mkdir /journal
```

4.2. Temporary mount the new ext4 file system to **/journal**.

```
[root@servera ~]# mount -o data=journal /dev/vdb1 /journal
```

4.3. Unmount the new file system.

```
[root@servera ~]# umount /dev/vdb1
```

5. Mount the newly created filesystem persistently and use the required mounting options, after that logout from **servera**.

5.1. Configure an entry in **/etc/fstab** to persistently mount the newly created filesystem, include **data=journal** mount option.

```
[root@servera ~]# vim /etc/fstab
...output omitted...
/dev/vdb1  /journal  ext4  data=journal  0 0
```

5.2. Mount the newly created ext4 file system.

```
[root@servera ~]# mount /dev/vdb1
```

5.3. Verify the mounting options of the newly created ext4 file system.

```
[root@servera ~]# findmnt --target /journal
TARGET   SOURCE    FSTYPE OPTIONS
/journal /dev/vdb1 ext4   rw,relatime,seclabel,nodelalloc,data=journal
```

5.4. Logout from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab fs-journal finish** command to complete this exercise.

```
[student@workstation ~]$ lab fs-journal finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning File System Utilization

Performance Checklist

In this lab, you will tune file system utilization for different use cases.

Outcome(s)

You should be able to:

- Configure an ext4 file system for a specific workload.
- Configure an XFS file system with external journaling for a specific workload.

Before You Begin

Log in to **`workstation`** as **`student`** using **`student`** as the password.

On **`workstation`**, run the **`lab fs-review start`** command. The command determines if the host, **`serverb`**, is reachable on the network.

```
[student@workstation ~]$ lab fs-review start
```

You need to configure an XFS file system, that has a specific use case with the following characteristics:

- The file system would be used to process multimedia files, that are mainly large in size.
- The processes that use the data inside the file system allows the use of multi-threaded I/O.
- Increased inode size for better support to extended attributes.
- The logging of inode access times relative to modify or change time, would be needed.
- The external journaling in the file system is a requirement.
- The **`vdb`** and **`vdc`** disks will be used.
- Create a 5 GiB primary partition on the **`vdb`** disk, and use it as the required XFS file system.
- Create a 1 GiB primary partition on the **`vdc`** disk, and use it as external journaling for the XFS file system.
- The standard block size should be used in the XFS file system.
- Persistently mount the XFS file system on **`/bigfiles`** .

You also need to configure an ext4 file system, that has a specific use case with the following characteristics:

- The file system would be used to process text files, that are mainly small in size.
- The main process that use the data inside the file system use single-threaded I/O.
- The inode size should be reduced for better file system usage.
- The logging of inode access times in directories and files, would not be needed.
- The **`vdd`** disk will be used.
- Create a 5 GiB primary partition on the **`vdd`** disk, and use it as the required ext4 file system.
- The **`large_dir`** feature should be enabled for better compatibility with large directories in the ext4 file system.

- The **`huge_file`** and **`extra_isize`** features should be disabled as they do not be used in the ext4 file system.
- The standard block size should be used in the ext4 file system.
- Persistently mount the ext4 file system on **`/smallfiles`** .

1. Configure the XFS file system for the specified use case.
2. Persistently mount the XFS filesystem on **`/bigfiles`**.
3. Configure the ext4 file system for the specified use case.
4. Persistently mount the ext4 filesystem on **`/smallfiles`**. At the end log out from **`serverb`**.

Evaluation

On the **`workstation`** VM, run the **`lab fs-review grade`** command to confirm success of this exercise.

```
[student@workstation ~]$ lab fs-review grade
```

Finish

On the **`workstation`** VM, run the **`lab fs-review finish`** command to complete this exercise.

```
[student@workstation ~]$ lab fs-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning File System Utilization

Performance Checklist

In this lab, you will tune file system utilization for different use cases.

Outcome(s)

You should be able to:

- Configure an ext4 file system for a specific workload.
- Configure an XFS file system with external journaling for a specific workload.

Before You Begin

Log in to **`workstation`** as **`student`** using **`student`** as the password.

On **`workstation`**, run the **`lab fs-review start`** command. The command determines if the host, **`serverb`**, is reachable on the network.

```
[student@workstation ~]$ lab fs-review start
```

You need to configure an XFS file system, that has a specific use case with the following characteristics:

- The file system would be used to process multimedia files, that are mainly large in size.
- The processes that use the data inside the file system allows the use of multi-threaded I/O.
- Increased inode size for better support to extended attributes.
- The logging of inode access times relative to modify or change time, would be needed.
- The external journaling in the file system is a requirement.
- The **`vdb`** and **`vdc`** disks will be used.
- Create a 5 GiB primary partition on the **`vdb`** disk, and use it as the required XFS file system.
- Create a 1 GiB primary partition on the **`vdc`** disk, and use it as external journaling for the XFS file system.
- The standard block size should be used in the XFS file system.
- Persistently mount the XFS file system on **`/bigfiles`** .

You also need to configure an ext4 file system, that has a specific use case with the following characteristics:

- The file system would be used to process text files, that are mainly small in size.
- The main process that use the data inside the file system use single-threaded I/O.
- The inode size should be reduced for better file system usage.
- The logging of inode access times in directories and files, would not be needed.
- The **`vdd`** disk will be used.
- Create a 5 GiB primary partition on the **`vdd`** disk, and use it as the required ext4 file system.
- The **`large_dir`** feature should be enabled for better compatibility with large directories in the ext4 file system.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

- The **`huge_file`** and **`extra_isize`** features should be disabled as they do not be used in the ext4 file system.
- The standard block size should be used in the ext4 file system.
- Persistently mount the ext4 file system on **`/smallfiles`** .

1. Configure the XFS file system for the specified use case.

1.1. On **`workstation`**, use **`ssh`** to log in to **`serverb`** as **`student`**. Use the **`sudo -i`** command to switch to the **`root`** user's account.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[sudo] password for student: student
[root@serverb ~]#
```

1.2. Create the partition label standard on both **`vdb`** and **`vdc`** disks.

```
[root@serverb ~]# parted /dev/vdb mklabel gpt
```

```
[root@serverb ~]# parted /dev/vdc mklabel gpt
```

1.3. Create a primary partition of 5 GiB on the **`vdb`** disk and a primary partition of 1 GiB on the **`vdc`** disk.

```
[root@serverb ~]# parted --align=opt /dev/vdb mkpart primary xfs 0% 100%
```

```
[root@serverb ~]# parted --align=opt /dev/vdc mkpart primary xfs 0% 1GiB
```

1.4. Force a reread of the partition table.

```
[root@serverb ~]# partx -uv /dev/vdb
...output omitted...
[root@serverb ~]# partx -uv /dev/vdc
...output omitted...
```

1.5. Create the XFS file system on the 5 GiB partition, specify the 1 GiB partition as the location for the external journal, set the extended inode size to 512 bytes and the block size to 4096 bytes.

```
[root@serverb ~]# mkfs.xfs -i size=512 -b size=4096 -l logdev=/dev/vdc1 /dev/vdb1
```

2. Persistently mount the XFS filesystem on **`/bigfiles`**.

2.1. Create the mount point at **`/bigfiles`**.

```
[root@serverb ~]# mkdir /bigfiles
```

2.2. Configure an entry in **`/etc/fstab`** to persistently mount the newly created filesystem, include the external journaling and the **`relatime`** mount options.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# vim /etc/fstab
...output omitted...
/dev/vdb1  /bigfiles  xfs  relatime,logdev=/dev/vdc1  0 0
```

2.3. Mount the newly created XFS file system.

```
[root@serverb ~]# mount /dev/vdb1
```

2.4. Verify the mounting options of the newly created XFS file system.

```
[root@serverb ~]# findmnt --target /bigfiles
TARGET    SOURCE    FSTYPE OPTIONS
/bigfiles /dev/vdb1 xfs    rw,relatime,seclabel,attr2,inode64,logdev=/dev/
vdc1,noquota
```

3. Configure the ext4 file system for the specified use case.

3.1. Create the partition label standard on **vdd** disk.

```
[root@serverb ~]# parted /dev/vdd mklabel gpt
```

3.2. Create a primary partition of 5 GiB on the **vdd** disk.

```
[root@serverb ~]# parted --align=opt /dev/vdd mkpart primary ext4 0% 100%
```

3.3. Force a reread of the partition table.

```
[root@serverb ~]# partx -uv /dev/vdd
...output omitted...
```

3.4. Create the ext4 file system on the 5 GiB partition, set the reduced inode size to 128 bytes, the block size to 4096 bytes. Enable the **large_dir** feature and disable **huge_file** and **extra_isize** features.

```
[root@serverb ~]# mkfs.ext4 -I 128 -b 4096 \
-O large_dir,^huge_file,^extra_isize /dev/vdd1
```

4. Persistently mount the ext4 filesystem on **/smallfiles**. At the end log out from **serverb**.

4.1. Create the mount point at **/smallfiles**.

```
[root@serverb ~]# mkdir /smallfiles
```

4.2. Configure an entry in **/etc/fstab** to persistently mount the newly created filesystem, include the **noatime** mount option.

```
[root@serverb ~]# vim /etc/fstab
...output omitted...
/dev/vdd1  /smallfiles  ext4  noatime  0 0
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

4.3. Mount the newly created ext4 file system.

```
[root@serverb ~]# mount /dev/vdd1
```

4.4. Verify the mounting options of the newly created ext4 file system.

```
[root@serverb ~]# findmnt --target /smallfiles
TARGET      SOURCE    FSTYPE OPTIONS
/smallfiles /dev/vdd1 ext4   rw,noatime,seclabel
```

4.5. Logout from **serverb**.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On the **workstation** VM, run the **lab fs-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab fs-review grade
```

Finish

On the **workstation** VM, run the **lab fs-review finish** command to complete this exercise.

```
[student@workstation ~]$ lab fs-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- The different use cases of the XFS and ext4 file systems.
- How to tune the XFS and ext4 file systems for specific workloads, with formatting and mount options.
- How to configure the XFS and ext4 file systems with external journaling.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 11

Tuning Network Utilization

Goal

Manage application efficiency for network utilization.

Objectives

- Describe latency and throughput definitions and discuss the contrasting requirements for different network workloads.
- Configure network device parameters using **`ethtool`**.

Sections

- Configuring for Network Latency and Throughput (and Guided Exercise)
- Configuring the Network Link Layer (and Guided Exercise)

Lab

Tuning Network Utilization

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring for Network Latency and Throughput

Objectives

After completing this section, you should be able to describe latency and throughput definitions and discuss the contrasting requirements for different network workloads.

Packet Transmission and Reception

Network is one of the more complex and key elements when tuning a system. Most of the network stack is self-organizing, but there are some network elements that can impact performance. To identify which network elements are impacting the system performance and tune them properly, you need to know what network elements are involved in the flow that a packet follows both when transmitted or received by a system, and understand what tuning options are available to those that can become a bottleneck.

Packet Transmission

The following outline shows the steps of network transmission.

- Data is written to a socket (a file-like object) and is then put in the transmit buffer.
- The kernel encapsulates the data into a protocol data unit (PDU).
- The PDUs move to the per-device transmit queue.
- The network device driver copies the PDU from the head of the transmit queue to the NIC.
- The NIC sends the data and raises an interrupt when transmitted.

Packet Reception

The following outline shows the steps of network reception.

- The NIC receives a frame and uses DMA to copy the frame into the receive buffer.
- The NIC raises a hard interrupt.
- The kernel handles the hard interrupt and schedules a soft interrupt to handle the packet.
- The soft interrupt is handled and moves the packet to the IP layer.
- If the packet is destined for local delivery, the PDU is decapsulated and put in a socket receive buffer. If a process was waiting on this socket, it processes the data from the receive buffer.

The kernel management of the hard interrupt raised by the NIC prevents a phenomenon known as livelock. Livelock is based on the fact that a hard interrupt preempts everything, including other interrupt handlers. This could cause the receive buffers to fill up during heavy receive loads, if packets are not moved out of the receive buffers quickly enough.

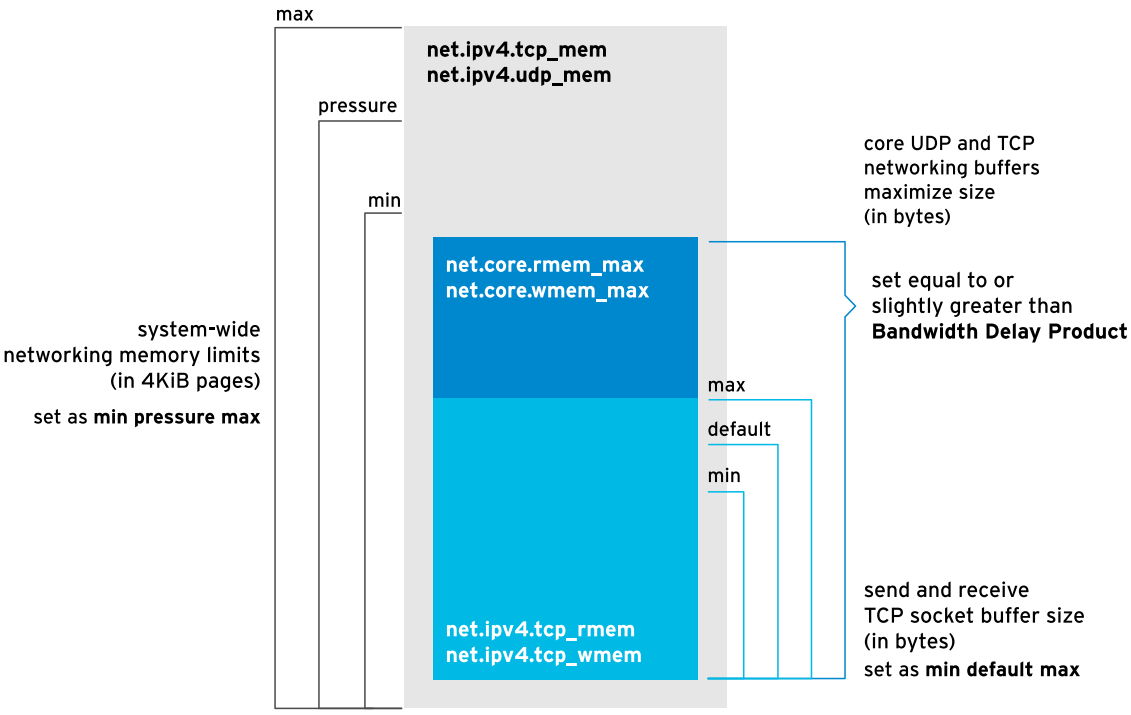
Another potential bottleneck is the NIC hardware buffer, also known as the ring buffer. This buffer may be a problem if a large number of packets are dropped by a NIC. Verify the number of dropped packets per NIC with the **`ethtool`** command. To solve this bottleneck, slow down the input traffic or increase the size of the hardware buffer queue.

How processors handle interrupts can be also a system bottleneck. If the process generating the interrupt and the interrupt itself are not handled by the same processor and core, the system may face latency and processor contention issues.

Finally, a bottleneck in an application receive queue can be identified by a large number of packets not copied to the process, or an increase in UDP input errors. To solve this, decrease the incoming traffic rate or increase the application's socket queue depth.

Network Kernel Tunables

The networking buffers (or queues) consist of the core networking read and write buffers (used for UDP and TCP), per-socket TCP read and write buffers, fragmentation buffers, and DMA buffers for the network card. The kernel automatically adjusts the size of these buffers based on the current network utilization, but within the limits specified by kernel tunables.



* squares symbolize system memory pages

Figure 11.1: Kernel Tunables for Networking

The defaults for these tunables are calculated at boot time based on the amount of available memory. On x86_64 machines, **`max`** is set to near the amount of system memory. On x86 systems, however, this might need to be increased.

These buffer sizes can be managed with some **`sysctl`** tunables.

`net.ipv4.tcp_mem`, **`net.ipv4.udp_mem`**

The systemwide memory limits for TCP and UDP, respectively. These settings consist of three fields: **`min`**, **`pressure`**, and **`max`**.

- **`min`** is the amount of memory in pages below which the kernel is not interested in TCP/UDP memory usage.
- **`pressure`**: When the amount of memory used by TCP/UDP exceeds this number of pages, the kernel will start to moderate its memory usage until it drops below **`min`** again.
- **`max`**: The maximum amount of memory allowed for queuing by all TCP/UDP sockets (in pages).

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Note
The values of **net.ipv4.tcp_mem** and **net.ipv4.udp_mem** are in pages, not bytes. View the page size (in bytes) by running **getconf PAGESIZE** (often 4096).

net.core.rmem_max, net.core.wmem_max
The core networking maximum socket receive/send (read/write) buffers. Values are in bytes.

net.ipv4.tcp_rmem, net.ipv4.tcp_wmem
Values are in bytes. A socket buffer starts at the size of **default** bytes (second value) and be automatically adjusted between **min** (first value) and **max** (third value) based on need.

Note
The **max** values of the **net.ipv4.tcp_rmem** and **net.ipv4.tcp_wmem** buffers cannot exceed the values of the core networking values (**net.core.rmem_max** and **net.core.wmem_max**). Be sure to adjust core networking values when increasing the TCP **max** read or write settings.

Calculating Bandwidth Delay Product

A calculation called the *bandwidth delay product* (BDP) is used to verify that buffers are correctly sized. The buffer size is calculated by estimating the amount of data that can fit in the network pipe between the source and the destination for a dedicated packet flow. The size of the network pipe is the capacity of the pipe (measured in bytes per second throughput) multiplied by the length of the pipe (measured in seconds as the round trip latency.)

To calculate the buffer sizes needed for maximum throughput, determine the amount of time that it takes for a packet to travel from the local system to the remote system. This can be used to calculate the amount of data that can be sent before the first packet has been received and acknowledged. First, figure out the round trip time of a packet, the time it takes for a packet to be sent from the local machine and returned from the remote machine. The **ping** command can be used to find the average round trip time.

```
[root@host ~]# ping host2
PING host2.lab.example.com (192.168.0.212) 56(84) bytes of data.
64 bytes from 192.168.0.212: icmp_seq=1 ttl=64 time=0.335ms
64 bytes from 192.168.0.212: icmp_seq=2 ttl=64 time=0.342 ms
64 bytes from 192.168.0.212: icmp_seq=3 ttl=64 time=0.347 ms
^C
--- host2.lab.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2092ms
rtt min/avg/max/mdev = 0.335/0.341/0.347/0.015 ms
```

The example shows an average round trip time (**rtt**) of 0.341 ms, or 0.000341 seconds. With a network speed (capacity) of 1 Gigabit per second:

```
1 Gb/s × 0.341 ms =
10^9  b/s × 0.000341 s =
341000 bits × 1/8 B/b  =
42625 Bytes = 41.63 KiB
```

This example results in a bandwidth delay product (BDP) of 41.63 KiB.

If the *BDP* goes above 64 KiB, TCP connections can utilize *window scaling*. A TCP window is the amount of data sent to the remote system that has not yet been acknowledged. If unacknowledged data grows to the window size, the sender will stop sending until previous data has been acknowledged.

By default, the window is limited to a maximum of 64 KiB. *Window scaling* can be negotiated, if both sides support it, allowing the window to grow. If the sysctl **`net.ipv4.tcp_window_scaling`** parameter is set to **`1`**, the kernel will attempt to negotiate window scaling. Window scaling will be disabled if this value is **`0`**.

> **Note**
> If a system will be handling multiple simultaneous connections, the per-socket buffer sizes only need to be large enough to handle the BDP of an individual socket.

Setting the TCP buffer size too large will result in a phenomenon known as buffer bloat. Buffer bloat happens when the rate at which packets are processed out of the buffer is lower than the rate at which the packets are sent to the buffer. This can seriously affect network speeds and latencies for connections that send small amounts of data (such as HTTP or SSH).

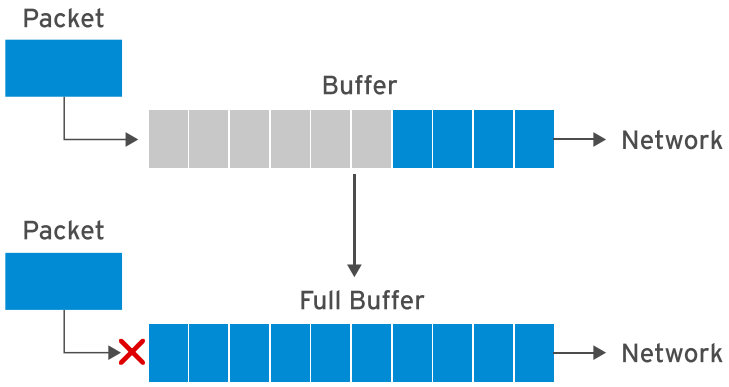


Figure 11.2: Buffer bloat

Enabling Jumbo Frames

Whenever a packet is transmitted across the network, the packet consists of headers and a payload. A typical TCP/IP packet header includes an Ethernet header, an IP header, and a TCP header. All of these headers are included in the maximum transmission unit (MTU) used on the network, the maximum size a single packet can have. For instance, a normal TCP connection using TCP timestamps uses 52 bytes for the protocol headers. With a default MTU of 1500 bytes, that is almost 3.5% of the total capacity lost to overhead.

One method to reduce the overhead incurred due to headers is to switch the protocol to another with less overhead. For example, switching from TCP to UDP will reduce the headers from 52 bytes to 28 bytes (less than 1.9% overhead with an MTU of 1500). However, this may not always be feasible.

Another method is to increase the size of the MTU. When increasing the MTU above the Ethernet standard of 1500 bytes, the resulting packets are called jumbo frames.

The usage of jumbo frames benefits not only the percentage of data included on every packet, but also the lower processing required on the NIC. Jumbo frames have to be supported in all

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

the network elements involved in its transmission. This requirement is hard to meet when so many different organizations, vendors, and network infrastructures are involved. That is the reason the key use case for jumbo frames is storage area networks. The network infrastructure related to a SAN is usually isolated, so the jumbo frame can be easily be enabled in all the network components. Other use cases for jumbo frames are server clusters, backup infrastructure, and high-performance private networks.

Before configuring a system to use jumbo frames with a specific NIC, make sure that every piece of networking equipment on the network, including but not limited to the NICs, switches, and routers, supports jumbo frames. The official maximum size for a jumbo frame is 9000 bytes, but some equipment supports even larger frames.

Taking the same 52 bytes of headers as previously mentioned, and assuming a 9000-byte MTU:

```
52 ÷ 9000 × 100% = 0.58%
```

This is a significant improvement over the previous 3.5% overhead.

You can use the **`nmcli`** command to modify the MTU size. The following example sets a MTU of **`9000`** for **`ens3`**.

```
[root@host ~]# nmcli connection modify ens3 802-3-ethernet.mtu 9000
```

References

For more information, refer to the *Networking* chapter in the *Red Hat Enterprise Linux 7 performance tuning guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/performance_tuning_guide

Red Hat Knowledge Base: How to enable jumbo frames for network interfaces in Red Hat Enterprise Linux
https://access.redhat.com/kb/docs/DOC-3644

Red Hat Knowledge Base: How do I tune RHEL for better TCP performance over a specific network connection?
https://access.redhat.com/solutions/168483

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring for Network Latency and Throughput

In this exercise, you will configure applications for efficient network utilization for different use cases.

Outcomes

You should be able to notice the effects of tuning the network buffer sizes on network throughput.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab network-latency start** command. The command determines if the host, **servera**, is reachable on the network. On **servera**, the command installs the Apache HTTP Server and creates a fake network device for the exercise.

```
[student@workstation ~]$ lab network-latency start
```

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Ping the 192.168.0.254 IP address. Notice the round-trip time.

```
[root@servera ~]# ping 192.168.0.254
PING 192.168.0.254 (192.168.0.254) 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=2001 ms
64 bytes from 192.168.0.254: icmp_seq=2 ttl=64 time=2000 ms
64 bytes from 192.168.0.254: icmp_seq=3 ttl=64 time=2000 ms
64 bytes from 192.168.0.254: icmp_seq=4 ttl=64 time=2000 ms
64 bytes from 192.168.0.254: icmp_seq=5 ttl=64 time=2000 ms
^C
--- 192.168.0.254 ping statistics ---
7 packets transmitted, 5 received, 28.5714% packet loss, time 60ms
rtt min/avg/max/mdev = 2000.176/2000.312/2000.523/1.792 ms, pipe 2
```

The connection to 192.168.0.254 has a two seconds round-trip time.

3. Assuming a 100 Mbps bandwidth, calculate the BDP based on your measured delay.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
                      100 × 10^6
100 Megabits/s × 2s = --------- × 2 = 25000000 bytes
                          8
```

4. Use the **`time`** command to determine how long it takes the **`wget`** command to download the `http://192.168.0.254/bigfile` file. When the download starts, observe the transfer rate and estimated time remaining.

```
[root@servera ~]# time wget http://192.168.0.254/bigfile
--2019-07-30 08:11:24--  http://192.168.0.254/bigfile
Connecting to 192.168.0.254:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5242880 (5.0M)
Saving to: 'bigfile'

bigfile             100%[=====================>]   5.00M   183KB/s    in 28s

2019-07-30 08:11:44 (320 KB/s) - 'bigfile' saved [5242880/5242880]


real  0m32.046s
user  0m0.010s
sys  0m0.030s
```

The time it takes to download the file is probably different on your system.

5. Temporarily set both the core networking maximum receive buffer and the maximum TCP receive buffer size to the BDP. For the **`net.ipv4.tcp_rmem`** parameter, you can keep the middle value (the initial buffer size) to its default value, **`87380`**, or increase it to start with a bigger initial buffer. The following command sets it to **`12500000`**.

```
[root@servera ~]# sysctl -w net.core.rmem_max=25000000
net.core.rmem_max = 25000000
[root@servera ~]# sysctl -w net.ipv4.tcp_rmem="4096 12500000 25000000"
net.ipv4.tcp_rmem = 4096 12500000 25000000
```

6. Run the **`time wget`** command again.

```
[root@servera ~]# time wget http://192.168.0.254/bigfile
--2019-07-30 08:16:21--  http://192.168.0.254/bigfile
Connecting to 192.168.0.254:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5242880 (5.0M)
Saving to: 'bigfile.1'

bigfile.1           100%[=====================>]   5.00M   320KB/s    in 16s

2019-07-30 08:16:41 (320 KB/s) - 'bigfile.1' saved [5242880/5242880]
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
real  0m20.038s
user  0m0.007s
sys  0m0.041s
```

Your recorded download time should now be lower than the previously recorded time.

7. Log off from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab network-latency finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab network-latency finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Configuring Network Driver Parameters

Objectives

After completing this section, you should be able to configure network device parameters using **ethtool**.

Network Performance Tuning

Network throughput can depend on multiple factors: the network cards used, the type of cabling, number of hops in a connection, the size of the packets being sent, et cetera. While some of these are properties of the hardware being used, others can also be influenced by a system.

One of the most drastic settings to change is the link speed of the network card being used. To query the current setting, use a tool called **ethtool**:

```
[root@host ~]# ethtool ens3
Settings for ens3:
 Supported ports: [ TP ]
 Supported link modes:   10baseT/Half 10baseT/Full ❶
                         100baseT/Half 100baseT/Full
                         1000baseT/Half 1000baseT/Full
 Supports auto-negotiation: Yes ❷
 Advertised link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Half 1000baseT/Full
 Advertised pause frame use: No
 Advertised auto-negotiation: Yes
 Speed: 1000Mb/s ❸
 Duplex: Full
 Port: Twisted Pair
 PHYAD: 1
 Transceiver: internal
 Auto-negotiation: on
 MDI-X: Unknown
 Supports Wake-on: g
 Wake-on: g
 Current message level: 0x000000ff (255)
 Link detected: yes
```

❶ **ens3** supports 10Mb/s, 100Mb/s, and 1000Mb/s links in both full- and half-duplex.
❷ Auto-negotiation is enabled.
❸ The connected port or device is up, and the link was set to 1000Mb/s full-duplex.

Auto-negotiation is a feature of Ethernet networks where a network card can communicate to the next hop to negotiate the highest speed. While auto-negotiation can normally be trusted, there are times when it does not work properly. To manage auto-negotiation, either limit the speeds that the network card advertises, or turn auto-negotiation off completely and set a speed manually.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

The following examples use the **ethtool** command with the **--change** (or **-s**) option to set the Ethernet options manually.

To limit link speed choices, use the **advertise** option with a bitmask of the desired speeds. To limit advertised speeds to 100 Mb/s full-duplex and 1000 Mb/s full-duplex for the **ens3** network interface:

```
[root@host ~]# ethtool -s ens3 advertise 0x28
```

The 0x28 value was obtained from the **ethtool**(8) man page, combining the values for 100baseT Full (0x008) and 1000baseT Full (0x020).

To turn off auto-negotiation and set a fixed speed, use the **autoneg**, **speed**, and **duplex** options on a single command, like this:

```
[root@host ~]# ethtool -s ens3 autoneg off speed 1000 duplex full
```

Note
Typically these settings do not work for virtual machines, as their network link speed depends on a physical network card in the hypervisor.

The Ethernet options you set with **ethtool** are not persistent.

In Red Hat Enterprise Linux 6, to configure those settings persistently, add **ETHTOOL_OPTS="*OPTIONS*"** to the interface's **/etc/sysconfig/network-scripts/ifcfg-*** file:

```
ETHTOOL_OPTS="-s ${DEVICE} autoneg off speed 1000 duplex full"
```

Red Hat Enterprise Linux 7 still supports, but deprecates, the **ETHTOOL_OPTS** parameter when NetworkManager is disabled. Consult the Knowledgebase – How to make NIC ethtool settings persistent (apply automatically at boot) [https://access.redhat.com/solutions/2127401] article for more information.

With NetworkManager (Red Hat Enterprise Linux 7 and 8), you persistently define the settings with the **802-3-ethernet.*** connection properties:

```
[root@host ~]# nmcli connection modify "Wired connection 1" \
802-3-ethernet.auto-negotiate off \
802-3-ethernet.speed 1000 \
802-3-ethernet.duplex full
```

The **ethtool** command can also report and modify more advanced network device parameters. For example, the **--show-offload** (or **-k**) option lists the device's features and the operations that the kernel can delegate to the device, such as packet segmentation or checksum computation. Use the **--offload** (or **-K**) option to modify those parameters. For persistence, NetworkManager groups those parameters in the **ethtool.feature*** connection properties.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Virtual Interfaces

The **ethtool** command can be used on virtual devices. Virtual devices are emulated Ethernet bridges, switches, routers, and NICs operating only in memory. A virtual NIC connects to a host virtual bridge, not the physical host NIC, operated by an emulated network driver, such as **virtio**. Querying a vNIC returns the emulated driver response. Many emulations default to fake 100Mbps or 1000Mbps speeds, which are not speed limits. Virtual networks operate at memory speed, unless Quality of Service (QoS) restrictions have been configured on the virtual segments.

Measuring Throughput

There are a variety of tools for measuring network throughput. On Ethernet networks, the **qperf** command can measure TCP, UDP, RDS, SCTP, and SDP socket throughputs and latencies. The **qperf** command can also perform measurements on RDMA and InfiniBand networks.

qperf needs to run on two machines. One (the listener) runs **qperf** without any options. The other (the sender) invokes **qperf** with the name of the first host as the first argument, followed by the test options. For example, to run TCP and UDP bandwidth tests between **demo** and **demo2** servers, start a **qperf** listener on **demo2**:

```
[root@host2 ~]# qperf
```

The server listens on TCP port 19765 by default, but can be set using the **--listen_port** option. While **qperf** is listening on **host2**, start the test on **host1**:

```
[root@host1 ~]# qperf host2 tcp_bw udp_bw
tcp_bw:
    bw  =  118 MB/sec
udp_bw:
    send_bw  =  120 MB/sec
    recv_bw  =  120 MB/sec
```

Note

Most modern NICs can be used to offload some operations from the CPU. This typically includes handling checksumming and fragmentation amongst others. To query and set these offload parameters, use **ethtool -k *device*** and **ethtool -K *device***, respectively.

Network Teaming

Network teaming is a method for linking NICs together logically to allow for failover or higher throughput. Teaming is a new implementation that does not affect the older bonding driver in the Linux kernel. It offers an alternate implementation. Red Hat Enterprise Linux 8 supports channel bonding for backward compatibility. Network teaming provides better performance and is more extensible because of its modular design.

A comparison of bonding and teaming can be viewed at the Red Hat Blog post"If You Like Bonding, You Will Love Teaming" [https://www.redhat.com/en/blog/if-you-bonding-you-will-love-teaming].

Red Hat Enterprise Linux 8 implements network teaming with a small kernel driver and a user-space daemon, **teamd**. The kernel handles network packets efficiently and **teamd** handles logic

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

and interface processing. Software, called *runners*, implement load balancing and active-backup logic, like **`roundrobin`**. The following runners are available to **`teamd`**:

- **`broadcast`**: a simple runner that transmits each packet from all ports.
- **`roundrobin`**: a simple runner that transmits packets in a round-robin fashion from each of the ports.
- **`activebackup`**: this is a failover runner that watches for link changes and selects an active port for data transfers.
- **`loadbalance`**: this runner monitors traffic and uses a hash function to try to reach a perfect balance when selecting ports for packet transmission.
- **`lacp`**: implements the 802.3ad Link Aggregation Control Protocol. Can use the same transmit port selection possibilities as the **`loadbalance`** runner.

All network interaction is done through a *team* interface, composed of multiple *network port* interfaces. When controlling teamed port interfaces with NetworkManager, and especially when fault finding, keep the following in mind:

- Starting the teamed interface does not automatically start the port interfaces.
- Starting a port interface always starts the teamed interface.
- Stopping the teamed interface also stops the port interfaces.
- A teamed interface without ports can start static IP connections.
- A team without ports waits for ports when starting DHCP connections.
- A team with a DHCP connection waiting for ports completes when a port with a carrier is added.
- A team with a DHCP connection waiting for ports continues waiting when a port without a carrier is added.

Configuring Interface Teams

The **`nmcli`** command can be used to create and manage team and port interfaces. The following steps are used to create and activate a network team interface:

- Create the team interface.
- Determine the IPv4 and IPv6 attributes of the team interface.
- Assign the port interfaces.
- Bring the team and port interfaces up/down.

Create the Team Interface

Use the **`nmcli`** command to create a connection for the network team interface with the following syntax:

```
nmcli connection add type team con-name CNAME ifname INAME team.OPTION OPTIONVALUE
```

CNAME is the name used to refer to the connection, *INAME* is the interface name, and *OPTION* is either **`runner`** or **`link-watchers`**. The *OPTIONVALUE* for **`team.runner`** is any of the runners available (e.g. **`loadbalance`**).

```
[root@host ~]# nmcli connection add type team con-name team0 ifname team0
 team.runner loadbalance
```

If you are configuring a link watcher with **`team.link-watchers`**, you need to at least configure the link watcher with the **`name`** property.

```
[root@host ~]# nmcli connection modify team0 team.link-watchers "name=ethtool"
```

```
nmcli connection add type team con-name CNAME ifname INAME team.OPTION OPTIONVALUE
```

Where *CNAME* is the connection name, *INAME* is the interface name, and *OPTION* is either **runner** or **link-watchers**. The *OPTIONVALUE* for **team.runner** is any of the runners available (for example, **loadbalance**).

```
[root@host ~]# nmcli connection add type team con-name team0 ifname team0
 team.runner loadbalance
```

If you are configuring a link watcher with **team.link-watchers**, you need at least to configure the link watcher with the **name** property.

```
[root@host ~]# nmcli connection modify team0 team.link-watchers "name=ethtool"
```

Determine the IPv4/IPv6 Attributes of the Team Interface

When the network team interface is created, IPv4 and IPv6 attributes can be assigned to it. If DHCP is available, this step is optional because the default attributes configure the interface to get its IP settings using DHCP.

The following example demonstrates how to assign a static IPv4 address to the **team0** interface:

```
[root@host ~]# nmcli connection modify team0 ipv4.addresses '1.2.3.4/24'
[root@host ~]# nmcli connection modify team0 ipv4.method manual
```

Note that the **ipv4.addresses** have to be assigned before the **ipv4.method** can be set to **manual**.

Assign the Port Interfaces

Use the **nmcli** command to assign each of the port interfaces with the following syntax:

```
nmcli connection add type ethernet slave-type team con-name CNAME ifname INAME
 master TEAM
```

CNAME is the name used to refer to the port, *INAME* is the name of an existing interface, and *TEAM* specifies the connection name of the network team interface. The following example configures **ens3** and **ens4** as ports for the **team0** interface.

```
[root@host ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port1 ifname ens3 master team0
[root@host ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port2 ifname ens4 master team0
```

```
nmcli connection add type ethernet slave-type team con-name CNAME ifname INAME
 master TEAM
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

where *CNAME* is the port name, *INAME* is the an existing interface name, and *TEAM* specifies the connection name of the team interface. The following example configures **ens3** and **ens4** as ports for the **team0** interface.

```
[root@host ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port1 ifname ens3 master team0
[root@host ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port2 ifname ens4 master team0
```

Bring the Team and Port Interfaces Up/Down

nmcli command can also be used to manage the connections for the team and port interfaces with the following syntax:

```
nmcli dev dis INAME
nmcli connection up CNAME
```

INAME is the device name of the team or port interface to be managed. *CNAME* is the connection name of that interface.

```
[root@host ~]# nmcli connection up team0
[root@host ~]# nmcli dev dis ens3
```

When the team interface is up, the **teamdctl** command can be used to display the team's state. The output of this command includes the status of the port interfaces.

```
[root@host ~]# teamdctl team0 state
setup:
  runner: roundrobin
ports:
  ens3
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  ens4
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
```

Persistent Team Configuration

NetworkManager creates configuration files for network teaming in the **/etc/sysconfig/network-scripts** directory the same way it does for other interfaces. A configuration file is created for each of the interfaces, for the team, and each of the ports.

The configuration file for the team interface defines the IP settings for the interface. The **DEVICETYPE** variable informs the initialization scripts this is a network team interface. Parameters

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

for **teamd** configuration are defined in the **TEAM_CONFIG** variable. Note that the contents of **TEAM_CONFIG** use JSON syntax.

```
DEVICE=team0
DEVICETYPE=Team
TEAM_CONFIG="{\"runner\": {\"name\": \"broadcast\"}}"
BOOTPROTO=none
IPADDR=172.25.5.100
PREFIX=24
NAME=team0
ONBOOT=yes
...output omitted...
```

The following is an example configuration file for a team port interface.

```
DEVICE=ens3
DEVICETYPE=TeamPort
TEAM_MASTER=team0
NAME=team0-port1
ONBOOT=yes
...output omitted...
```

The **DEVICETYPE** variable informs the initialization scripts this is a team port interface. The **TEAM_MASTER** variable defines which team device it is a port for.

Troubleshooting Team Interfaces

The **teamnl** and **teamdctl** commands are very useful for troubleshooting network teams. These commands only work on network teams that are up. The following examples show some typical uses for these commands.

Display the team ports of the **team0** interface:

```
[root@host ~]# teamnl team0 ports
 4: ens4: up 0Mbit HD
 3: ens3: up 0Mbit HD
```

Display the currently active port of **team0**:

```
[root@host ~]# teamnl team0 getoption activeport
3
```

Set the option for the active port of **team0**:

```
[root@host ~]# teamnl team0 setoption activeport 3
```

Use **teamdctl** to display the current state of the **team0** interface:

```
[root@host ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
  ens4
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  ens3
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: ens3
```

Use **`teamdctl`** to display the current JSON configuration for **`team0`**:

```
[root@host ~]# teamdctl team0 config dump
{
    "device": "team0",
    "mcast_rejoin": {
        "count": 1
    },
    "notify_peers": {
        "count": 1
    },
    "ports": {
        "ens3": {
            "link_watch": {
                "name": "ethtool"
            },
            "prio": -10,
            "sticky": true
        },
        "ens4": {
            "link_watch": {
                "name": "ethtool"
            },
            "prio": 100
        }
    },
    "runner": {
        "name": "activebackup"
    }
}
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

References

`qperf`(1) and **`ethtool`**(8) man pages

`nmcli-examples`(5), **`teamdctl`**(8), **`teamd`**(8), **`teamd.conf`**(5), and **`teamnl`**(8) man pages

For more information, refer to the *configuring networking with nmcli* chapter in the *Red Hat Enterprise Linux 8 configuring and managing networking guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_networking

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring the Network Link Layer

In this exercise, you will configure network devices using **ethtool**.

Outcomes

You should be able to:

- Measure network performance between two systems.
- Notice the effects of tuning the network devices.
- Combine two network links using network teaming to provide higher throughput.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab network-driver start** command. The command determines if the hosts, **servera** and **serverb**, are reachable on the network. On **servera**, the command creates fake network devices for the exercise.

```
[student@workstation ~]$ lab network-driver start
```

1. On **workstation**, open two terminal windows. In the first terminal, use **ssh** to log in to **servera** as **student**. In the second terminal, use **ssh** to log in to **serverb** as **student**. In both terminals, use **sudo -i** to switch to **root**.

   1.1. In the first terminal, log in to **servera** as **student**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

   1.2. In the second terminal, log in to **serverb** as **student**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[sudo] password for student: student
[root@serverb ~]#
```

2. In the second terminal, on **serverb**, prepare the system to run the **qperf** command in server mode. Open the firewall, install the *qperf* package, and start the **qperf** daemon.

   2.1. Open the firewall TCP port 19765, which is the default listening port for the **qperf** daemon. The **qperf** client also needs a data port of your choice to run its tests. Open TCP port 19766 for the exercise.

```
[root@serverb ~]# firewall-cmd --add-port=19765/tcp --add-port=19766/tcp
success
[root@serverb ~]# firewall-cmd --add-port=19765/tcp --add-port=19766/tcp \
--permanent
success
```

2.2. Install the *qperf* package.

```
[root@serverb ~]# yum install qperf
...output omitted...
```

2.3. Start the **qperf** daemon and let it run.

```
[root@serverb ~]# qperf
```

3. In the first terminal, on **servera**, install the *qperf* package and run the TCP bandwidth test, **tcp_bw**.

3.1. Install the *qperf* package.

```
[root@servera ~]# yum install qperf
...output omitted...
```

3.2. Run the **qperf** command with the **--ip_port** option to provide the data port to use, 19766. Display the result of the test in bits per second with the **--use_bits_per_sec** option. Run the test for 20 seconds with the **--time** option.

```
[root@servera ~]# qperf serverb --ip_port 19766 --use_bits_per_sec \
--time 20 tcp_bw
tcp_bw:
    bw  =  5.89 Gb/sec
```

Depending on the classroom environment, the bandwidth may be very different.

4. Use the **ethtool** command to query the offload features of the network device.

4.1. Before running the **ethtool** command, use the **ip address show** command to identify the network device associated with **servera** IP address: 172.25.250.10/24.

```
[root@servera ~]# ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 default qlen 1000
    link/ether 52:54:00:00:fa:0a brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.10/24 brd 172.25.250.255 scope global noprefixroute ens3
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
        valid_lft forever preferred_lft forever
    inet6 fe80::2937:3aa3:ea8d:d3b1/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
...output omitted...
```

Depending on the classroom environment, the interface name may be different on your system. In that case, use your device name in the following steps.

4.2. Run the **ethtool** command with the **--show-offload** (or **-k**) option to list the network operations that the system offloads to the network device.

```
[root@servera ~]# ethtool -k ens3
Features for ens3:
rx-checksumming: on [fixed]
tx-checksumming: on
  tx-checksum-ipv4: off [fixed]
  tx-checksum-ip-generic: on
  tx-checksum-ipv6: off [fixed]
  tx-checksum-fcoe-crc: off [fixed]
  tx-checksum-sctp: off [fixed]
scatter-gather: on
  tx-scatter-gather: on
  tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
  tx-tcp-segmentation: on
  tx-tcp-ecn-segmentation: on
  tx-tcp-mangleid-segmentation: off
  tx-tcp6-segmentation: on
udp-fragmentation-offload: off
...output omitted...
```

Notice that the network stack offloads the TCP packet segmentation to the network device. This usually leads to better large packet performance.

5. Disable TCP segmentation offloading and rerun the **qperf** test.

5.1. Use the **ethtool** command with the **--offload** (or **-K**) option to disable offloading of the TCP segmentation.

```
[root@servera ~]# ethtool -K ens3 tx-tcp-segmentation off
```

5.2. Confirm that TCP segmentation offloading is now disabled.

```
[root@servera ~]# ethtool -k ens3
Features for ens3:
rx-checksumming: on [fixed]
tx-checksumming: on
  tx-checksum-ipv4: off [fixed]
  tx-checksum-ip-generic: on
  tx-checksum-ipv6: off [fixed]
  tx-checksum-fcoe-crc: off [fixed]
  tx-checksum-sctp: off [fixed]
scatter-gather: on
  tx-scatter-gather: on
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
  tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
  tx-tcp-segmentation: off
  tx-tcp-ecn-segmentation: on
  tx-tcp-mangleid-segmentation: off
  tx-tcp6-segmentation: on
udp-fragmentation-offload: off
...output omitted...
```

5.3. Rerun the **qperf** test.

```
[root@servera ~]# qperf serverb --ip_port 19766 --use_bits_per_sec \
--time 20 tcp_bw
tcp_bw:
    bw  =  1.05 Gb/sec
```

Compare this value with the result from the first test. Notice that the bandwidth is significantly reduced.

6. Make the offloading configuration persistent.

6.1. Use the **nmcli** command to retrieve the connection name associated with the network device.

```
[root@servera ~]# nmcli connection
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  6bc56692-0f3b-3bf9-941f-8bc9f5ff7941  ethernet  ens3
```

In the previous output, the connection name associated with **ens3** is **Wired connection 1**. The name may be different on your system.

6.2. Edit the NetworkManager connection to add your offloading configuration.

```
[root@servera ~]# nmcli connection modify "Wired connection 1" \
ethtool.feature-tx-tcp-segmentation off
```

This configuration is for demonstration purpose only. In production, you would not disable TCP segmentation offloading.

7. In this exercise section, configure a team interface called **team0** built upon two port interfaces: **eno1** and **eno2**.

In the second terminal, exit the **qperf** command, log off from **serverb**, and log back in to **servera** as the **student** user. Escalate user privileges to the **root** user's account.

7.1. In the second terminal, on **serverb**, press **Ctrl**+**C** to exit the **qperf** daemon.

```
[root@serverb ~]# qperf
^C
```

7.2. Log off from **serverb**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

7.3. Use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[sudo] password for student: student
[root@servera ~]#
```

8. Create an active-backup teaming interface called **team0** and assign it the 192.168.0.100/24 address. Assign the **eno1** and **eno2** network interfaces as port interfaces for the **team0** connection.

8.1. Use the **nmcli** command to create the **team0** connection.

```
[root@servera ~]# nmcli connection add type team con-name team0 ifname team0 \
team.runner activebackup
Connection 'team0' (f333e6e6-7d30-47f5-87e0-5ae762f313b1) successfully added.
```

8.2. Assign the 192.168.0.100/24 address to the **team0** connection.

```
[root@servera ~]# nmcli connection modify team0 ipv4.addresses 192.168.0.100/24
[root@servera ~]# nmcli connection modify team0 ipv4.method manual
```

8.3. Assign **eno1** and **eno2** as port interfaces for the **team0** connection.

```
[root@servera ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port1 ifname eno1 master team0
Connection 'team0-port1' (d2db5c16-2905-4142-9a47-60e50c7d4dc1) successfully
 added.
[root@servera ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port2 ifname eno2 master team0
Connection 'team0-port2' (90c4b8af-5fc5-4209-990b-86026d4b43c7) successfully
 added.
```

8.4. Activate the **team0** connection and the two slave ports.

```
[root@servera ~]# nmcli connection up team0
Connection successfully activated (master waiting for slaves) (D-Bus active
 path: /org/freedesktop/NetworkManager/ActiveConnection/6)
[root@servera ~]# nmcli connection up team0-port1
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/7)
[root@servera ~]# nmcli connection up team0-port2
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/8)
```

8.5. Control the state of the teamed interface.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eno1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  eno2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
runner:
  active port: eno1
```

Notice that the **eno1** port is active and receives all the traffic. The system uses **eno2** as a backup in case of failure.

The active interface may be different on your system.

8.6. To confirm the IP connectivity, ping the network gateway 192.168.0.254 on the 192.168.0.0/24 network through the **team0** interface.

```
[root@servera ~]# ping -I team0 -c 4 192.168.0.254
PING 192.168.0.254 (192.168.0.254) from 192.168.0.100 team0: 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=0.423 ms
64 bytes from 192.168.0.254: icmp_seq=2 ttl=64 time=0.095 ms
64 bytes from 192.168.0.254: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 192.168.0.254: icmp_seq=5 ttl=64 time=0.108 ms

--- 192.168.0.254 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 122ms
rtt min/avg/max/mdev = 0.083/0.178/0.423/0.128 ms
```

9. Confirm that the TCP traffic only uses the active slave interface.

9.1. Use **ping** to generate traffic. Let the command keep running.

```
[root@servera ~]# ping 192.168.0.254
PING 192.168.0.254 (192.168.0.254) 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=0.388 ms
64 bytes from 192.168.0.254: icmp_seq=2 ttl=64 time=0.127 ms
64 bytes from 192.168.0.254: icmp_seq=3 ttl=64 time=0.113 ms
64 bytes from 192.168.0.254: icmp_seq=4 ttl=64 time=0.118 ms
```

9.2. In the first terminal, use the **tcpdump** command to monitor the active slave interface.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# tcpdump -i eno1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes
10:21:30 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:21:30 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:21:31 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:21:31 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:21:32 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:21:32 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:21:33 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
```

Notice the ICMP echo packets from the **ping** command.

9.3. Press **Ctrl**+**C** to exit the **tcpdump** command.

```
...output omitted...
10:21:33 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:21:34 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
^C
14 packets captured
14 packets received by filter
0 packets dropped by kernel
```

9.4. Rerun the **tcpdump** command to monitor the backup slave interface this time.

```
[root@servera ~]# tcpdump -i eno2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno2, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Notice that **tcpdump** does not report traffic related to **ping**.

9.5. Press **Ctrl**+**C** to exit the **tcpdump** command.

```
...output omitted...
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

9.6. In the second terminal, press **Ctrl**+**C** to exit the **ping** command.

```
...output omitted...
64 bytes from 192.168.0.254: icmp_seq=183 ttl=64 time=0.125 ms
64 bytes from 192.168.0.254: icmp_seq=184 ttl=64 time=0.122 ms
^C
--- 192.168.0.254 ping statistics ---
184 packets transmitted, 184 received, 0% packet loss, time 598ms
rtt min/avg/max/mdev = 0.070/0.113/0.275/0.035 ms
```

10. Modify the **team0** connection to use the round-robin runner. With that runner, both slave interfaces receive traffic.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

10.1. Update the **team0** connection with the new runner.

```
[root@servera ~]# nmcli connection modify team0 team.runner roundrobin
```

10.2. Restart the **team0** connection.

```
[root@servera ~]# nmcli connection down team0
Connection 'team0' successfully deactivated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/6)
[root@servera ~]# nmcli connection up team0
Connection successfully activated (master waiting for slaves) (D-Bus active
 path: /org/freedesktop/NetworkManager/ActiveConnection/9)
```

11. Test the round-robin runner by generating traffic with **ping** and monitoring the slave interfaces with **tcpdump**.

11.1. Rerun the **ping** command to generate some traffic. Let the command running.

```
[root@servera ~]# ping 192.168.0.254
PING 192.168.0.254 (192.168.0.254) 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=0.101 ms
64 bytes from 192.168.0.254: icmp_seq=2 ttl=64 time=0.080 ms
64 bytes from 192.168.0.254: icmp_seq=3 ttl=64 time=0.207 ms
64 bytes from 192.168.0.254: icmp_seq=4 ttl=64 time=0.129 ms
```

11.2. In the first terminal, use the **tcpdump** command to monitor the first slave interface.

```
[root@servera ~]# tcpdump -i eno1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes
10:34:41 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:34:41 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:34:43 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:34:43 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:34:45 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
```

Notice the ICMP echo packets from the **ping** command.

11.3. Press **Ctrl**+**C** to exit the **tcpdump** command.

```
...output omitted...
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

11.4. Rerun the **tcpdump** command to monitor the second slave interface.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# tcpdump -i eno2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno2, link-type EN10MB (Ethernet), capture size 262144 bytes
10:38:15 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:38:15 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:38:17 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
10:38:17 IP 192.168.0.254 > servera.lab.example.com: ICMP echo reply, ...
10:38:19 IP servera.lab.example.com > 192.168.0.254: ICMP echo request, ...
```

Notice that this slave interface also receives some of the ICMP traffic. Both interfaces are active.

11.5. Press **Ctrl**+**C** to exit the **tcpdump** command.

```
...output omitted...
^C
12 packets captured
12 packets received by filter
0 packets dropped by kernel
```

11.6. In the second terminal, press **Ctrl**+**C** to exit the **ping** command.

```
...output omitted...
64 bytes from 192.168.0.254: icmp_seq=377 ttl=64 time=0.125 ms
64 bytes from 192.168.0.254: icmp_seq=378 ttl=64 time=0.126 ms
^C
--- 192.168.0.254 ping statistics ---
378 packets transmitted, 378 received, 0% packet loss, time 415ms
rtt min/avg/max/mdev = 0.060/0.120/0.969/0.056 ms
```

12. Log off from **servera** in the two terminals.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab network-driver finish** command to complete this exercise. The command deletes the resources created during the exercise.

```
[student@workstation ~]$ lab network-driver finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning Network Utilization

Performance Checklist

In this lab, you will tune network efficiency for different use cases.

Outcomes

You should be able to:

- Combine two network links using network teaming to provide higher throughput.
- Create a new tuned profile and use a set of system parameters to tune the network buffers of your server.
- Tune the network devices.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab network-review start** command. The command determines if the host, **serverb**, is reachable on the network. On **serverb**, the command creates fake network devices for the exercise.

```
[student@workstation ~]$ lab network-review start
```

1. On **serverb**, create a team connection as follows:

   - Name the team interface **team0**.
   - Use the **loadbalance** runner.
   - Configure the connection with the 10.42.0.42/24 IP address.
   - Attach the **eno1** and **eno2** interfaces to the **team0** connection.

   When done, you should be able to ping the 10.42.0.254 address. Because the **eno1** and **eno2** interfaces are slow, you may have to wait up to a minute for the connection to come up.

2. On **serverb**, calculate the BDP using the remote host at 10.42.0.254 as a reference point and assuming a 100 Mbps bandwidth.

   Use that BDP to create a new tuned profile as follows:

   - Name the new profile **enterprise-network**.
   - Base your new profile on the existing **network-throughput** profile.
   - Set the size of the core send and receive buffers to the BDP.
   - Set the max size of the TCP send and receive buffers to the BDP.

   Activate your new profile.

3. One of your colleagues has disabled offloading TCP segmentation to the network device. Identify the network device associated with **serverb** main IP address, 172.25.250.11, and persistently re-enable TCP segmentation offloading.

Evaluation

On **`workstation`**, run the **`lab network-review grade`** command to grade this lab. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab network-review grade
```

Finish

On **`workstation`**, run the **`lab network-review finish`** command to finish this exercise.

```
[student@workstation ~]$ lab network-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning Network Utilization

Performance Checklist

In this lab, you will tune network efficiency for different use cases.

Outcomes

You should be able to:

- Combine two network links using network teaming to provide higher throughput.
- Create a new tuned profile and use a set of system parameters to tune the network buffers of your server.
- Tune the network devices.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab network-review start** command. The command determines if the host, **serverb**, is reachable on the network. On **serverb**, the command creates fake network devices for the exercise.

```
[student@workstation ~]$ lab network-review start
```

1. On **serverb**, create a team connection as follows:

   - Name the team interface **team0**.
   - Use the **loadbalance** runner.
   - Configure the connection with the 10.42.0.42/24 IP address.
   - Attach the **eno1** and **eno2** interfaces to the **team0** connection.

   When done, you should be able to ping the 10.42.0.254 address. Because the **eno1** and **eno2** interfaces are slow, you may have to wait up to a minute for the connection to come up.

   1.1. On **workstation**, use **ssh** to log in to **serverb** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

   1.2. Use the **nmcli** command to create the **team0** connection with the **loadbalance** runner.

```
[root@serverb ~]# nmcli connection add type team con-name team0 ifname team0 \
team.runner loadbalance
Connection 'team0' (ff4293f4-8a8f-4314-91c4-3ebcd456d13f) successfully added.
```

1.3. Assign the 10.42.0.42/24 address to the **team0** connection.

```
[root@serverb ~]# nmcli connection modify team0 ipv4.addresses 10.42.0.42/24
[root@serverb ~]# nmcli connection modify team0 ipv4.method manual
```

1.4. Assign **eno1** and **eno2** as port interfaces for the **team0** connection.

```
[root@serverb ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port1 ifname eno1 master team0
Connection 'team0-port1' (d2db5c16-2905-4142-9a47-60e50c7d4dc1) successfully
 added.
[root@serverb ~]# nmcli connection add type ethernet slave-type team \
con-name team0-port2 ifname eno2 master team0
Connection 'team0-port2' (90c4b8af-5fc5-4209-990b-86026d4b43c7) successfully
 added.
```

1.5. Activate the **team0** connection and the two slave ports.

```
[root@serverb ~]# nmcli connection up team0
Connection successfully activated (master waiting for slaves) (D-Bus active
 path: /org/freedesktop/NetworkManager/ActiveConnection/6)
[root@serverb ~]# nmcli connection up team0-port1
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/7)
[root@serverb ~]# nmcli connection up team0-port2
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/8)
```

1.6. To confirm the IP connectivity, ping the 10.42.0.254 address through the **team0** interface. Because the **eno1** and **eno2** interfaces are slow, you may have to wait up to a minute for the connection to come up. In that case, retry the **ping** command.

```
[root@serverb ~]# ping -I team0 10.42.0.254
PING 10.42.0.254 (10.42.0.254) from 10.42.0.42 team0: 56(84) bytes of data.
64 bytes from 10.42.0.254: icmp_seq=1 ttl=64 time=2001 ms
64 bytes from 10.42.0.254: icmp_seq=2 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=3 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=4 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=5 ttl=64 time=1000 ms
^C
--- 10.42.0.254 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 14ms
rtt min/avg/max/mdev = 1000.310/1200.468/2000.932/400.233 ms, pipe 2
```

2. On **serverb**, calculate the BDP using the remote host at 10.42.0.254 as a reference point and assuming a 100 Mbps bandwidth.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Use that BDP to create a new tuned profile as follows:

- Name the new profile **`enterprise-network`**.
- Base your new profile on the existing **`network-throughput`** profile.
- Set the size of the core send and receive buffers to the BDP.
- Set the max size of the TCP send and receive buffers to the BDP.

Activate your new profile.

2.1. Ping the 10.42.0.254 IP address to retrieve the round-trip time.

```
[root@serverb ~]# ping 10.42.0.254
PING 10.42.0.254 (10.42.0.254) 56(84) bytes of data.
64 bytes from 10.42.0.254: icmp_seq=1 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=2 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=3 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=4 ttl=64 time=1000 ms
64 bytes from 10.42.0.254: icmp_seq=5 ttl=64 time=1000 ms
^C
--- 10.42.0.254 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 1000.214/1000.298/1000.485/0.899 ms
```

The connection to 10.42.0.254 has a round-trip time of one second.

2.2. Assuming a 100 Mbps bandwidth, calculate the BDP based on your measured delay.

```
                      100 × 10^6
100 Megabits/s × 1s = --------- × 1 = 12500000 bytes
                          8
```

2.3. Create the **`enterprise-network`** tuned profile directory.

```
[root@serverb ~]# mkdir /etc/tuned/enterprise-network
```

2.4. Create the **`/etc/tuned/enterprise-network/tuned.conf`** file with the following contents.

```
[main]
include = network-throughput

[sysctl]
```

2.5. Use the current values of the **`sysctl`** tunables as a starting point.

```
[root@serverb ~]# sysctl net.core.rmem_max net.core.wmem_max \
net.ipv4.tcp_rmem net.ipv4.tcp_wmem >> /etc/tuned/enterprise-network/tuned.conf
```

2.6. Edit the **`/etc/tuned/enterprise-network/tuned.conf`** file again and use the BDP value to update the buffer sizes.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[main]
include = network-throughput

[sysctl]
net.core.rmem_max = 12500000
net.core.wmem_max = 12500000
net.ipv4.tcp_rmem = 4096      87380   12500000
net.ipv4.tcp_wmem = 4096      16384   12500000
```

2.7. Enable the **`enterprise-network`** tuned profile.

```
[root@serverb ~]# tuned-adm profile enterprise-network
```

3. One of your colleagues has disabled offloading TCP segmentation to the network device. Identify the network device associated with **`serverb`** main IP address, 172.25.250.11, and persistently re-enable TCP segmentation offloading.

3.1. Use the **`ip address show`** command to identify the network device associated with **`serverb`** IP address: 172.25.250.11.

```
[root@serverb ~]# ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 default qlen 1000
    link/ether 52:54:00:00:fa:0b brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.11/24 brd 172.25.250.255 scope global noprefixroute ens3
    inet6 fe80::867f:9910:1149:4302/64 scope link noprefixroute
...output omitted...
```

Depending on the classroom environment, the interface name may be different on your system. In that case, use your device name in the following steps.

3.2. Confirm that offloading TCP segmentation to the network device is disabled for the **`ens3`** interface.

```
[root@serverb ~]# ethtool -k ens3
Features for ens3:
rx-checksumming: on [fixed]
tx-checksumming: on
 tx-checksum-ipv4: off [fixed]
 tx-checksum-ip-generic: on
 tx-checksum-ipv6: off [fixed]
 tx-checksum-fcoe-crc: off [fixed]
 tx-checksum-sctp: off [fixed]
scatter-gather: on
 tx-scatter-gather: on
 tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
 tx-tcp-segmentation: off
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
 tx-tcp-ecn-segmentation: on
 tx-tcp-mangleid-segmentation: off
 tx-tcp6-segmentation: on
...output omitted...
```

3.3. Use the **ethtool** command to enable offloading of the TCP segmentation.

```
[root@serverb ~]# ethtool -K ens3 tx-tcp-segmentation on
```

3.4. To make that configuration persistent, first use the **nmcli** command to retrieve the connection name associated with the network device.

```
[root@serverb ~]# nmcli connection
NAME                UUID                                  TYPE      DEVICE
team0               ff4293f4-8a8f-4314-91c4-3ebcd456d13f  team      team0
team0-port1         3630d1b9-5a07-47ad-955a-f3cab139b34c  ethernet  eno1
team0-port2         efbc563e-0641-4efb-9f6f-cb15f04b1786  ethernet  eno2
Wired connection 1  b5980b14-7629-3b49-a5f2-154eca358706  ethernet  ens3
```

In the previous output, the connection name associated with **ens3** is **Wired connection 1**. The name may be different on your system.

3.5. Edit the NetworkManager connection to add your offloading configuration.

```
[root@serverb ~]# nmcli connection modify "Wired connection 1" \
ethtool.feature-tx-tcp-segmentation on
```

3.6. Log off from **serverb** when done.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab network-review grade** command to grade this lab. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab network-review grade
```

Finish

On **workstation**, run the **lab network-review finish** command to finish this exercise.

```
[student@workstation ~]$ lab network-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- Network buffer sizes can be manipulated through **`sysctl`** tunables such as **`net.core.rmem_max`**, **`net.core.wmem_max`**, **`net.ipv4.tcp_rmem`**, and **`net.ipv4.tcp_wmem`**.
- Protocol overhead can account for a significant percentage of a data packet's content. Overhead can be reduced by increasing the MTU to make use of jumbo frames.
- The **`ethtool`** utility can be used to display and modify network card settings.
- The **`qperf`** utility can be used to measure network performance between two systems.
- Network teaming configuration is done with NetworkManager.
- Some teaming runners improve throughput.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 12

Performing Tuning Hosts in Virtualization Environments

Goal

Distinguish the requirements for tuning in virtualized environments.

Objectives

- Explain the differences between tuning hosts in a virtualized environment and a cloud environment.
- Gather performance metrics from virtual machines using open source tools.

Sections

- Tuning Hosts in Virtualization Environments (and Guided Exercise)
- Monitoring Performance Metrics in a Virtualization Environment (and Guided Exercise)

Lab

Performing Tuning in Virtualization Environments

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Tuning Hosts in Virtualization Environments

Objectives

After completing this section, you should be able to explain the differences between tuning hosts in a virtualized environment and a cloud environment.

Tuning Virtualization Hosts

Tuning a virtualized environment differs from tuning an operating system because multiple virtual machines can coexist on the same physical system. This is known as *multitenancy*. Because the tenants share resources, performance issues can arise due to over-consumption of resources by a virtual machine. For example, a virtual machine on the same host performs a full database backup during the peak load interval, thus interfering with the network and disk I/O.

To minimize such performance issues, you can perform resource isolation on the host to provide better performance to all virtual machines running on the host. Resource isolation applies to network, memory, block I/O, and CPU for both the host and the virtual machine.

Selecting a Tuned Profile

Tuned profiles are preconfigured to enhance performance and reduce power consumption in specific use cases. When choosing virtualization tuned profiles, choose one profile for the hypervisor and one for use on the virtual machine. Red Hat Enterprise Linux provides two default tuned profiles, **`virtual-host`** and **`virtual-guest`**.

The **`virtual-host`** profile is the recommended profile for virtualization hosts running with KVM. The profile inherits the **`throughput-performance`** profile. The profile configures the dirty page syncing threshold to 5% of system memory instead of the default 10%.

Tuning CPU with NUMA

In RHEL, automatic NUMA balancing is enabled by default. This feature allows processes for automatic NUMA placement and periodic unmapping of memory, which leads to migration to different NUMA nodes and thereby increasing NUMA faults. Manual NUMA placement override automatic NUMA balancing.

When tuning KVM guests for better performance, Red Hat recommends to limit the guest memory size to the amount of physical memory on a single NUMA node, which prevents the splitting of resources across NUMA nodes. Use the **`numastat -c qemu-kvm`** command to view the memory alignment across NUMA nodes.

The **`libvirtd`** daemon provides a mechanism to tune the performance of a NUMA host by controlling NUMA policy for the virtual machine processes. The **`virsh numatune`** command returns the memory mode policy used by the host to map memory to the guest and a list of NUMA nodes eligible for memory mapping to the guest.

```
[root@host ~]# virsh numatune
numa_mode      : strict
numa_nodeset   : 0
```

A **`strict`** mode policy means that the guest can access memory from a specific node set.

Tuning Virtual CPU

In KVM, guest virtual CPUs run as a user-space process on the KVM host. The virtual CPU process can be pinned to a host physical CPU to increase performance. Pinning a process increases the CPU cache hit ratio for that process, saving additional memory accesses. Use the **`virsh vcpupin`** command to set the CPU affinity between the virtual CPU and the physical CPU of the host.

Use the **`virsh vcpupin`** command to view the current pinning configuration for a guest.

```
[root@host ~]# virsh vcpupin rhel8-demo
VCPU: CPU Affinity
----------------------------------
   0: 0-1
   1: 0-1
```

In the above output, the guest virtual CPU process can run on any of the physical CPUs (0-1). To pin both the virtual CPUs to the first physical CPU, use the following command:

```
[root@host ~]# virsh vcpupin rhel8-demo 0 0
[root@host ~]# virsh vcpupin rhel8-demo 1 0
```

Use the **`virsh vcpuinfo`** command to inspect the CPU affinity. The **`CPU Affinity: y-`** value indicates that the CPU affinity of the virtual CPU is the first physical CPU of the host.

```
[root@host ~]# virsh vcpuinfo rhel8-demo
VCPU:           0
CPU:            0
State:          running
CPU time:       32.3s
CPU Affinity:   y-

VCPU:           1
CPU:            1
State:          running
CPU time:       46.1s
CPU Affinity:   y-
```

The **`<cputune>`** tag in **`libvirtd`** XML file defines the virtual CPU pinning.

```
[root@host ~]# virsh dumpxml rhel8-demo | sed -n '/<cputune>/,/<\/cputune>/p'
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='0'/>
  </cputune>
```

If the number of virtual CPUs is higher than the number of physical CPUs, virtual CPU pinning must be avoided. On a NUMA-based system **`<vcpupin>`** and **`<numatune>`** must be configured to achieve optimal performance.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Tuning the QEMU Emulator

KVM uses QEMU to run virtual machines at near-native speeds by emulating the host's CPU for hardware-assisted virtualization. QEMU uses multiple threads, used for the QEMU main event loop, asynchronous I/O operation completion, migration data transfer, and SPICE display I/O. The **`libvirtd`** driver uses a placement policy to ping QEMU emulator threads to the same physical CPUs as used for the guest virtual CPUs. To assign a dedicated physical CPU for a guest's emulator threads, use the following command:

```
[root@host ~]# virsh emulatorpin rhel8-demo 1
```

The above command pins the emulator thread to the **`CPU1`** physical CPU. Use the **`virsh emulatorpin`** command to view the emulator threads CPU affinity.

```
[root@host ~]# virsh emulatorpin rhel8-demo
emulator: CPU Affinity
----------------------------------
       *: 1
```

The XML file **`<emulatorpin>`** tag defines the QEMU emulator threads' CPU pinning.

```
[root@host ~]# virsh dumpxml rhel8-demo | sed -n '/<cputune>/,/<\/cputune>/p'
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='0'/>
    <emulatorpin cpuset='1'/>
  </cputune>
```

Limiting Host Memory

When optimizing KVM memory performance, ensure that guests do not allocate more memory than required. A KVM virtual machine has two allocation values: maximum allocation and current allocation. The maximum allocation is the maximum memory allocated to the guest when it is running. The current allocation is the actual allocated memory, and is less than or equal to the maximum allocation. Use the **`virsh dominfo`** command to view allocation:

```
[root@host ~]# virsh dominfo rhel8-demo
Id:             1
Name:           rhel8-demo
UUID:           81a7a4c1-7c0e-49b7-8f8a-a8e79b579879
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       111.1s
Max memory:     2097152 KiB
Used memory:    2097152 KiB
...output omitted...
```

Use the **`virsh setmem`** and **`virsh setmaxmem`** commands to set the memory allocation.

The **`virsh memtune`** command sets the limit on the physical memory consumed by a guest virtual machine. The hard limit sets the maximum physical memory a guest virtual machine can consume. The soft limit sets the memory limit to enforce memory restriction during memory

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

contention. Setting the hard limit to a low value may result in the kernel killing the guest virtual machine. Bear in mind the following memory overheads before setting the memory limits for a guest: the guest virtual machine, QEMU emulator threads, and the guest video RAM, because memory limits are applied as a whole.

```
[root@host ~]# virsh memtune rhel8-demo --hard-limit 2G --soft-limit 1G
```

Use the **`virsh memtune`** command to view the memory limits. The output of the command is always in KiB.

```
[root@host ~]# virsh memtune rhel8-demo
hard_limit     : 2097152
soft_limit     : 1048576
swap_hard_limit: unlimited
```

The **`<memtune>`** tag in **`libvirtd`** XML file defines the memory limits for a guest virtual machine.

```
[root@host ~]# virsh dumpxml rhel8-demo | sed -n '/<memtune>/,/<\/memtune>/p'
  <memtune>
    <hard_limit unit='KiB'>2097152</hard_limit>
    <soft_limit unit='KiB'>1048576</soft_limit>
  </memtune>
```

Allocating Huge Pages to KVM Guests

Huge pages provide a substantial performance improvement for large, critical, virtualized deployment use cases, such as NFV applications. A KVM guest requests huge pages explicitly through kernel boot parameters or using the memory backing store. Memory backing allows the host to manage huge pages within the guest virtual machine properly.

The **`<memoryBacking>`** tag in the **`libvirtd`** XML file defines the memory backing for a guest virtual machine. The **`<hugepages>`** tag instructs the host that the guest virtual machine must have the memory allocated using huge pages. The following output from the **`libvirtd`** XML file defines the use of huge pages in a NUMA-based host.

```
<memoryBacking>
  <hugepages>
    <page size='2' unit='M' nodeset='0-3,5'/>
  </hugepages>
</memoryBacking>
```

To allocate huge memory pages, set the total number of required huge pages using the **`vm.nr_hugepages`** tunable:

```
[root@host ~]# sysctl -w vm.nr_hugepages=20
vm.nr_hugepages = 20
```

Restart the guest virtual machine with the huge pages defined in the **`libvirtd`** XML file. Use the **`cat /proc/meminfo | grep -i huge`** command to view the allocated huge pages.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Describing Kernel Samepage Merging

When guest virtual machines are running identical operating systems or workloads, there is a high chance that many memory pages have the same content. *Kernel Samepage Merging (KSM)*, also known as *Kernel Shared Memory*, reduces the total memory usage by merging identical pages into one memory page. When a guest must modify a shared memory page by writing to it, KSM clones the shared page to give the guest a new, unshared copy. This procedure is known as *Copy on Write (COW)*.

Using KSM requires two services, **`ksm`**, the services that performs the memory scanning and page merging, and **`ksmtuned`** which controls when and how aggressively **`ksm`** scans memory.

Configure KSM manually using files in the **`/sys/kernel/mm/ksm/`** directory. The files control **`ksm`**, but **`ksmtuned`** adjusts these settings as necessary while running.

```
[root@host ~]# ls /sys/kernel/mm/ksm/*
/sys/kernel/mm/ksm/full_scans
/sys/kernel/mm/ksm/max_page_sharing
/sys/kernel/mm/ksm/merge_across_nodes
/sys/kernel/mm/ksm/pages_shared
/sys/kernel/mm/ksm/pages_sharing
/sys/kernel/mm/ksm/pages_to_scan
/sys/kernel/mm/ksm/pages_unshared
/sys/kernel/mm/ksm/pages_volatile
/sys/kernel/mm/ksm/run
/sys/kernel/mm/ksm/sleep_millisecs
/sys/kernel/mm/ksm/stable_node_chains
/sys/kernel/mm/ksm/stable_node_chains_prune_millisecs
/sys/kernel/mm/ksm/stable_node_dups
/sys/kernel/mm/ksm/use_zero_pages
```

KSM Tunables

`run`
: When set to **`1`**, **`ksm`** actively scans memory. When set to **`0`**, scanning is disabled.

`pages_to_scan`
: The number of memory pages to scan in the next cycle.

`sleep_millisecs`
: The number of milliseconds to sleep between cycles.

`pages_shared`
: The total number of pages shared.

`pages_sharing`
: The number of pages currently shared.

`full_scans`
: The number of times the entire memory has been scanned.

`merge_across_nodes`
: Whether or not pages from different NUMA nodes can be merged.

Use the **`virsh node-memory-tune`** command to configure the KSM tunables. For example, the following command specifies the number of pages to scan before the scanning process sleeps for 10 milliseconds.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@host ~]# virsh node-memory-tune \
--shm-pages-to-scan 100 \
--shm-sleep-millisecs 10
```

To tune KSM, it is often more useful to use the **ksmtuned** service than to tune by hand. To configure the **ksmtuned** service, use the **/etc/ksmtuned.conf** file.

To specify the number of pages that KSM shares from the host memory, add the **KSM_MAX_KERNEL_PAGES** parameter to **/etc/sysconfig/ksm**. When the parameter is absent or set to **0**, KSM shares up to half the physical memory. Shareable memory increases performance, but at the cost of less flexibility when it comes to memory assignments.

```
[root@host ~]# cat /etc/sysconfig/ksm
# The maximum number of unswappable kernel pages
# which may be allocated by ksm (0 for unlimited)
# If unset, defaults to half of total memory
# KSM_MAX_KERNEL_PAGES=
```

To disable KSM, both stop and disable the **ksmtuned** and **ksm** services on the host. The **<nosharepages/>** tag instructs the host to disable KSM for the guest virtual machine.

```
<memoryBacking>
  <nosharepages/>
</memoryBacking>
```

Tuning Disk and Block I/O

A guest virtual disk can be either a block device or an image file. A block device is the preferred choice for a virtual disk. The file-based image file increases additional resource demand for I/O operations in comparison to the block device. Image files require more resources for I/O operations compared to block devices, thus making disk tuning a necessity.

KVM supports two image formats, **raw** and **qcow2**. The **raw** format provides better performance compared to **qcow2**. The **qcow2** format has greater overhead because it performs additional tasks. For example, when a **qcow2** virtual image grows, it is allocated new *clusters* of space that must be filled with data from the backing file. The **qcow2** format provides the following features:

- Uses a sparse image file that helps to over-allocate host storage.
- Stores the changes to another disk image, without actually affecting the contents of the original image. The image is known as a *copy-on-write* image.
- Supports snapshots. Snapshots are images that refer to an original image using *Redirect-on-Write* to avoid changing the original image. Snapshots are read-only, while original images are writable.
- Supports compression and allows each cluster to be compressed using zlib. Uses clusters to store the contents of the device.
- Supports encryption of clusters.

Use the **qemu-img create** command to create a **qcow2** image.

```
[root@host ~]# qemu-img create -f qcow2 -o preallocation=metadata /var/lib/
libvirt/images/disk.qcow2 1G
Formatting '/var/lib/libvirt/images/second-disk.qcow2', fmt=qcow2 size=1073741824
 cluster_size=65536 preallocation=metadata lazy_refcounts=off refcount_bits=16
```

The **qcow2** format supports four level of preallocation:

preallocation=metadata
: This option allocates space required by the metadata and not for data while creating the image. This results in the quick provision of an image, but because the image is sparse, the guest writes are slower.

preallocation=falloc
: This option allocates space required by the metadata and data while creating the image, but marks the blocks unallocated. This results in the quick provision of an image, but is slower than using the **metadata** option. The disk writes are faster.

preallocation=full
: This option allocates the full file size during image creation, and does not use any sparse techniques. This results in slower initial provisioning of the image. During use, however, the disk write performance is the same as a normal **falloc** routine.

When tuning disk I/O on a KVM host, there are three options to consider:

- Disk cache mode
- Disk I/O mode
- I/O tuning

Disk Cache Mode

KVM allows for various disk caches to be specified when configuring the guest virtual machine. The disk cache mode influences how to read and write data that is cached in the host page cache. Each virtual disk can be configured to use the following cache modes:

cache=none
: The disk I/O from the guest is not cached on the host but stored in the physical disk cache. This option is generally the best choice and supports migration.

cache=writethrough
: The disk I/O from the guest is cached on the host and written to the physical disk. The write is considered complete only when the data is committed to the physical disk. This option ensures data integrity, but writes are slower.

cache=writeback
: The disk I/O from the guest is cached on the host. The write is considered complete only when the data is committed to the host cache. The host disk cache management commits to the physical disk.

cache=directsync
: The disk I/O from the guest writes to the physical disk, and bypasses the host cache when desirable. The write is considered complete only when the data is committed to the physical disk. This option ensures data integrity, but writes are slower. This option is useful for guests that do not flush the disk I/O very often.

cache=unsafe
: This mode is similar to **writeback** but all disk I/O flush commands from the guest are ignored. This mode does not ensure data integrity and increases disk performance. For

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

example, during a guest installation, the cache mode is the best choice, but not on production workloads.

Guests use the host default cache mode setting if the mode is not defined. The below output shows a use the **cache=none** option for a virtual disk.

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none'/>
  <source file='/var/lib/libvirt/images/disk.qcow2'/>
  <target dev='vdb' bus='virtio'/>
</disk>
```

Disk I/O Threads

I/O threads are dedicated threads for supported disk devices to perform block I/O requests to improve scalability. Use the **virsh iothreadinfo** command to get the number of I/O threads and CPU affinity of each thread configured for a guest.

```
[root@host ~]# virsh iothreadinfo rhel8-demo
 IOThread ID     CPU Affinity
---------------------------------------------------
 1               0-1
```

No more than one or two I/O threads must be pinned to a host CPU, and one I/O thread per virtual block device must be configured for good performance. Use the **virsh iothreadadd** command to add I/O threads to a guest.

```
[root@host ~]# virsh iothreadadd rhel8-demo 2
```

The **libvirtd** XML file defines the I/O threads using the **<iothreads>** tag. The following output shows two I/O threads assigned to a guest, and are identified by IDs **1** and **2**.

```
<iothreads>2</iothreads>
<iothreadids>
  <iothread id='1'/>
  <iothread id='2'/>
</iothreadids>
```

The I/O thread mode defines the policy on I/O operations, namely, **io='threads'** and **io='native'**. The **io='threads'** option sets the I/O mode to host user-mode based threads. The **io='native'** option uses kernel asynchronous I/O with direct I/O options. When no I/O mode is passed to a virtual disk it uses the kernel default. The following output demonstrates the **io=native** option for a virtual disk.

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none' io=native/>
  <source file='/var/lib/libvirt/images/disk.qcow2'/>
  <target dev='vdb' bus='virtio'/>
</disk>
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Virtual Disk I/O Tuning

It is important to throttle I/O so that no guest consumes more I/O from a physical disk than allocated, which provides Quality of Service (QoS) for all running guests on the host. Use the **virsh blkdeviotune** command to view and set limits on throughput and I/O operations on each block device attached to a guest.

The following example demonstrates throttling a guest's block device (**vdb**) to 1000 I/O operations per second and 20 MB throughput per second:

```
[root@host ~]# virsh blkdeviotune rhel8-demo vdb --total-iops-sec 1000 --total-
bytes-sec 20MB
[root@host ~]# virsh dumpxml rhel8-demo | grep -B4 -A2 iotune
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none' io='native'/>
  <source file='/var/lib/libvirt/images/disk.qcow2'/>
  <target dev='vdb' bus='virtio'/>
  <iotune>
    <total_iops_sec>1000</total_iops_sec>
    <total_bytes_sec>20971520</total_bytes_sec>
  </iotune>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
</disk>
```

Setting Limits on KVM Guests

By default, **libvirtd** configures control groups for virtual machines running on the host. Use the **lscgroups** command to view the cgroups definitions.

```
[root@host ~]# lscgroup
cpu,cpuacct:/
cpu,cpuacct:/user.slice
cpu,cpuacct:/system.slice
cpu,cpuacct:/machine.slice
cpu,cpuacct:/machine.slice/machine-qemu\x2d1\x2drhel8\x2ddemo.scope
cpu,cpuacct:/machine.slice/machine-qemu\x2d1\x2drhel8\x2ddemo.scope/vcpu1
cpu,cpuacct:/machine.slice/machine-qemu\x2d1\x2drhel8\x2ddemo.scope/vcpu0
cpu,cpuacct:/machine.slice/machine-qemu\x2d1\x2drhel8\x2ddemo.scope/emulator
net_cls,net_prio:/
net_cls,net_prio:/machine.slice
net_cls,net_prio:/machine.slice/machine-qemu\x2d1\x2drhel8\x2ddemo.scope
...output omitted...
memory:/
memory:/machine.slice
memory:/machine.slice
memory:/machine.slice/machine-qemu\x2d1\x2drhel8\x2ddemo.scope
...output omitted...
```

The **libvirtd** daemon provides an API for interacting with cgroup limits assigned to individual running guests. Use the **virsh schedinfo** command to view and set CPU related limits on virtual machines.

```
[root@host ~]# virsh schedinfo rhel8-demo
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
```

The **virsh schedinfo** command sets the CPU limits for guests. For example, to set the **cpu_shares** limit to **2048** on a virtual machine, use the following command:

```
[root@foundationX ~]# virsh schedinfo desktop cpu_shares=2048
Scheduler      : posix
cpu_shares     : 100
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
```

A guest's CPU shares are calculated from the CPU weight of all guests running on the host. For example, a guest with **cpu_shares=2048** receives twice as much CPU time as one configured with **cpu_shares=1024**.

The **libvirtd** daemon also provides an API to tune block I/O cgroup tunables. The block I/O **weight** specifies the relative proportion of block I/O access available by default to a guest, in the range from 100 to 1000. The block device I/O weight (**device_weight**) specifies the relative proportion of block I/O access on a specific device available by default to a guest, in the range from 100 to 1000. The device I/O throttling is read and write I/O operations per seconds allowed on a specific block device of the host. Use the **virsh blkiotune** command to view and set block I/O weights and I/O throttling for a guest accessing the disk on the host.

```
[root@host ~]# virsh blkiotune rhel8-demo
weight         : 500
device_weight  :
```

Tuning Virtual Networks

Apart from setting network kernel tunables on the host, the **virtio_net** network driver provides better performance compared to the standard emulated device. The **vhost_net** driver provides enhanced performance benefits in mission-critical Network Function Virtualization (NFV) applications. The device using the **vhost_net** driver makes fewer system calls compared to **virtio_net**. The **vhost_net** devices operate entirely in user space without assistance from a kernel module. This approach skips the Linux network stack and delivers packets directly to a user-space application. The network throughput increases when using the user-space data path because there is no need for context switching between user space and kernel space.

```
[root@host ~]# modinfo vhost_net
filename:       /lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/vhost/
vhost_net.ko.xz
alias:          devname:vhost-net
alias:          char-major-10-238
description:    Host kernel accelerator for virtio net
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
author:         Michael S. Tsirkin
license:        GPL v2
version:        0.0.1
rhelversion:    8.0
srcversion:     582E764C2D0D60894ECA060
depends:        vhost,tun,tap
...output omitted...
```

Single Root I/O Virtualization

One of the other network tuning options is device assignment and Single Root I/O Virtualization (SR-IOV) to a guest. SR-IOV is a feature available in some PCIe network interface cards (NICs) that allows the physical NIC to be presented as several virtual NICs. This means that a single physical NIC can be shared between multiple guests, removing the usual 1:1 mapping limitation when passing PCI devices through from the host to the virtual guests. In SR-IOV terms, a physical NIC is called a Physical Function (PF), and a virtual NIC is called a Virtual Function (VF).

References

`virsh`(1) man pages.

For more information on optimizing virtual environments, refer to the *Virtualization Tuning and Optimization Guide* at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/virtualization_tuning_and_optimization_guide/index

For more information on **`libvirtd`** XML elements and attributes, refer to the *Domain XML format* at
https://libvirt.org/formatdomain.html

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Tuning Hosts in Virtualization Environments

In this exercise, you will tune a virtual machine to optimize performance.

Outcomes

You should be able to tune a KVM virtual machine's CPU, memory, and disk performance.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab virt-host start** command. This command determines if the host, **servera**, is reachable on the network, installs necessary packages, and defines a virtual guest on **servera** named **rhel8-lab**.

```
[student@workstation ~]$ lab virt-host start
```

1. On **workstation**, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. Use the **virsh** command to ensure that the **rhel8-lab** VM is available and **shut off**.

```
[root@servera ~]# virsh list --all
Id    Name                           State
----------------------------------------------------
-     rhel8-lab                      shut off
```

3. Display detailed host and guest information to determine the number of available host CPUs and confirm that the **rhel8-lab** VM is balanced between CPUs, not pinned to a single CPU.

   3.1. List host CPUs available to guest VMs.

```
[root@servera ~]# lscpu | grep -i '^cpu'
CPU op-mode(s):      32-bit, 64-bit
CPU(s):              2
CPU family:          6
CPU MHz:             3591.678
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

3.2. Two physical CPUs available to the host. List vCPU information for the **rhel8-lab** VM to determine the CPU placement.

```
[root@servera ~]# virsh vcpuinfo rhel8-lab
VCPU:           0
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   yy

VCPU:           1
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   yy
```

The **CPU Affinity: yy** value means that **rhel8-lab** can run on any CPU and is not pinned.

4. Pin both **rhel8-lab** vCPUs to use the first CPU on **servera**.

4.1. Use the **virsh vcpupin** command to pin the **rhel8-lab** vCPU to the first CPU on **servera**. The **--config** option sets the value persistently.

```
[root@servera ~]# virsh vcpupin rhel8-lab --config 0 0
[root@servera ~]# virsh vcpupin rhel8-lab --config 1 0
```

4.2. Use the **virsh vcpuinfo** command to view the **rhel8-lab** CPU affinity.

```
[root@servera ~]# virsh vcpuinfo rhel8-lab
VCPU:           0
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   y-

VCPU:           1
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   y-
```

The **CPU Affinity: y-** value means that **rhel8-lab** is using the first CPU on **servera**.

4.3. View the CPU parameters in the **libvirtd** XML file for **rhel8-lab**.

```
[root@servera ~]# virsh dumpxml rhel8-lab | sed -n '/<cputune>/,/<\/cputune>/p'
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='0'/>
  </cputune>
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5. Pin the emulator thread processes to use the second CPU on **servera**.

5.1. Use the **virsh emulatorpin** command to view the **rhel8-lab** CPU affinity for emulator thread processes.

```
[root@servera ~]# virsh emulatorpin rhel8-lab
emulator: CPU Affinity
----------------------------------
       *: 0-1
```

The **rhel8-lab** emulator threads are balanced between both CPUs on **servera**.

5.2. Use the **virsh emulatorpin** command to pin the emulator thread processes to the second CPU on **servera**.

```
[root@servera ~]# virsh emulatorpin rhel8-lab --config 1
```

5.3. View the **rhel8-lab** CPU affinity for emulator threads.

```
[root@servera ~]# virsh emulatorpin rhel8-lab
emulator: CPU Affinity
----------------------------------
       *: 1
```

5.4. View the CPU affinity configuration for emulator threads in the **libvirtd** XML file.

```
[root@servera ~]# virsh dumpxml rhel8-lab | sed -n '/<cputune>/,/<\/cputune>/p'
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='0'/>
    <emulatorpin cpuset='1'/>
  </cputune>
```

6. Set a limit of 2048 CPU shares for **rhel8-lab** on **servera**.

6.1. Use the **virsh schedinfo** command to view the available CPU time shared between the running virtual machines and the **rhel8-lab** VM.

```
[root@servera ~]# virsh schedinfo rhel8-lab
Scheduler      : posix
cpu_shares     : 0
vcpu_period    : 0
vcpu_quota     : 0
emulator_period: 0
emulator_quota : 0
global_period  : 0
global_quota   : 0
iothread_period: 0
iothread_quota : 0
```

6.2. Use the **virsh schedinfo** command to set the CPU shares to **2048** for **rhel8-lab**.

```
[root@servera ~]# virsh schedinfo rhel8-lab --config cpu_shares=2048
Scheduler      : posix
cpu_shares     : 2048
vcpu_period    : 0
vcpu_quota     : 0
emulator_period: 0
emulator_quota : 0
global_period  : 0
global_quota   : 0
iothread_period: 0
iothread_quota : 0
```

The **rhel8-lab** VM gets twice as much CPU time as other virtual machines that have a default CPU share of **1024**.

6.3. View the CPU share configuration in the **rhel8-lab** XML file.

```
[root@servera ~]# virsh dumpxml rhel8-lab | sed -n '/<cputune>/,/<\/cputune>/p'
  <cputune>
    <shares>2048</shares>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='0'/>
    <emulatorpin cpuset='1'/>
  </cputune>
```

7. Set the soft limit to 512 MiB for the **rhel8-lab** VM. Stop and disable Kernel Samepage Merging on **servera**.

7.1. Use the **virsh memtune** command to view the currently configured memory limits.

```
[root@servera ~]# virsh memtune rhel8-lab
hard_limit     : unlimited
soft_limit     : unlimited
swap_hard_limit: unlimited
```

7.2. View the current memory allocated to the **rhel8-lab** VM.

```
[root@servera ~]# virsh dominfo rhel8-lab
Id:             -
Name:           rhel8-lab
UUID:           81a7a4c1-7c0e-49b7-8f8a-a8e79b579879
OS Type:        hvm
State:          shut off
CPU(s):         2
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
```

The current allocated memory is 1024 MiB.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

7.3. Set the soft limit for the **rhel8-lab** VM to use up to 512 MiB of physical host memory.

```
[root@servera ~]# virsh memtune rhel8-lab --config --soft-limit 512MiB
```

7.4. View the memory tuning configuration in the **libvirtd** XML file of the **rhel8-lab** VM.

```
[root@servera ~]# virsh dumpxml rhel8-lab | sed -n '/<memtune>/,/<\/memtune>/p'
  <memtune>
    <soft_limit unit='KiB'>524288</soft_limit>
  </memtune>
```

7.5. Stop and disable the **ksm** and **ksmtuned** services on **servera**.

```
[root@servera ~]# systemctl disable ksm --now
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
[root@servera ~]# systemctl disable ksmtuned --now
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```

7.6. Disable KSM for the **rhel8-lab** VM. Use the **virsh edit** command to edit the XML file and add the following content below the **<currentMemory>** block:

```
[root@servera ~]# virsh edit rhel8-lab
...output omitted...
  <memoryBacking>
    <nosharepages/>
  </memoryBacking>
...output omitted...
```

8. Restrict the total number of I/O operations per second on the **vdb** device attached to the **rhel8-lab** VM to **1000**.

8.1. Use the **virsh domblklist** command to list the block devices attached to the **rhel8-lab** VM.

```
[root@servera ~]# virsh domblklist rhel8-lab
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rhel8-lab.ovl
```

8.2. Use the **qemu-img** command to create a new QCOW2 virtual disk.

```
[root@servera ~]# qemu-img create -f qcow2 -o preallocation=metadata \
/var/lib/libvirt/images/second-disk.qcow2 1G
Formatting '/var/lib/libvirt/images/second-disk.qcow2', fmt=qcow2 size=1073741824
 cluster_size=65536 preallocation=metadata lazy_refcounts=off refcount_bits=16
```

The **-o preallocation=metadata** option allocates QCOW2 metadata and keeps the image sparse. Notice that the size on the file system is 836 KB and the allocated disk size is 1.1 GB.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# ls -lsh /var/lib/libvirt/images/second-disk.qcow2
836K -rw-r--r--. 1 root root 1.1G Aug  6 17:44 /var/lib/libvirt/images/second-
disk.qcow2
```

8.3. Attach the new disk to the **rhel8-lab** VM as **vdb**. Use the **virsh edit** command and edit the **libvirtd** XML file with the following content. Add the following content below the existing **<disk>** block:

```
[root@servera ~]# virsh edit rhel8-lab
    ...output omitted...
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native'/>
      <source file='/var/lib/libvirt/images/second-disk.qcow2'/>
      <target dev='vdb' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </disk>
    ...output omitted...
```

8.4. Use the **virsh blkdeviotune** command to set the total number of I/O operations per second to **1000** for the **vdb** device attached to the **rhel8-lab** VM.

```
[root@servera ~]# virsh blkdeviotune rhel8-lab vdb --config --total-iops-sec 1000
```

8.5. View the block device tuning configuration in the **libvirtd** XML of the **rhel8-lab** VM.

```
[root@servera ~]# virsh dumpxml rhel8-lab | grep -B4 -A2 'iotune'
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native'/>
      <source file='/var/lib/libvirt/images/second-disk.qcow2'/>
      <target dev='vdb' bus='virtio'/>
      <iotune>
        <total_iops_sec>1000</total_iops_sec>
      </iotune>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </disk>
```

9. Start the **rhel8-lab-tuned** VM using the tuned **libvirtd** XML file.

9.1. Rename the **rhel8-lab** VM to **rhel8-lab-tuned**.

```
[root@servera ~]# virsh domrename rhel8-lab rhel8-lab-tuned
Domain successfully renamed
```

9.2. Create a new **libvirtd** XML file named **/var/lib/libvirt/images/rhel8-lab-tuned.xml**.

```
[root@servera ~]# virsh dumpxml rhel8-lab-tuned > \
/var/lib/libvirt/images/rhel8-lab-tuned.xml
```

9.3. Remove the **rhel8-lab-tuned** VM definition.

```
[root@servera ~]# virsh undefine rhel8-lab-tuned
Domain rhel8-lab-tuned has been undefined
```

9.4. Define the **rhel8-lab-tuned** VM using the **/var/lib/libvirt/images/rhel8-lab-tuned.xml** file.

```
[root@servera ~]# virsh define /var/lib/libvirt/images/rhel8-lab-tuned.xml
Domain rhel8-lab-tuned defined from /var/lib/libvirt/images/rhel8-lab-tuned.xml
```

9.5. Start the **rhel8-lab-tuned** VM.

```
[root@servera ~]# virsh start rhel8-lab-tuned
Domain rhel8-lab-tuned started
```

9.6. Log off from **servera**.

```
[root@servera ~]# exit
[student@servera ~]$ exit
[student@workstation ~]$
```

Finish

On **workstation**, run the **lab virt-host finish** command to complete this exercise. This command removes the **rhel8-lab** VM running on **servera**.

```
[student@workstation ~]$ lab virt-host finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Monitoring Performance Metrics in a Virtualized Environment

Objectives

After completing this section, you should be able to use open source tools to gather performance metrics from virtual machines.

Retrieving Performance Metrics

The **virsh** command provides many tools to retrieve the performance metrics of guests running on a host. The metrics are retrieved based on an individual guest or the devices attached to the guest. For example, you can use the **virsh domblkstat** command to retrieve the block I/O statistics of a block device attached to a guest:

```
[root@host ~]# virsh domblkstat rhel8-demo vda --human
Device: vda
 number of read operations:      8970
 number of bytes read:           293372928
 number of write operations:     6140
 number of bytes written:        151969280
 number of flush operations:     1958
 total duration of reads (ns):   16302637384
 total duration of writes (ns):  11898334383
 total duration of flushes (ns): 10498237420
```

The **virsh domifstat** command provides network statistics of an interface: Use **virsh domiflist** to query the interface name.

```
[root@host ~]# virsh domiflist rhel8-demo
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:ed:95:42
[root@host ~]# virsh domifstat rhel8-demo vnet0
vnet0 rx_bytes 10495245
vnet0 rx_packets 48857
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 92558
vnet0 tx_packets 1073
vnet0 tx_errs 0
vnet0 tx_drop 0
```

The following example shows displaying memory statistics for a guest:

```
[root@host ~]# virsh dommemstat rhel8-demo
actual 1048576
swap_in 0
swap_out 0
```

```
major_fault 415
minor_fault 108622
unused 665376
available 840748
usable 647060
last_update 1565261177
rss 722528
```

Using the **`virsh`** command to retrieve the performance metrics does not scale when the number of guests is large. This method also does not provide historical data for analyzing performance.

Retrieving Metrics Using Prometheus

Prometheus is an open source system monitoring and alerting project, initially built at SoundCloud. The project is a part of Cloud Native Computing Foundation (CNCF) graduate projects. As a metrics-based monitoring system, Prometheus is designed to track overall system health, behavior, and performance, rather than tracking individual events. Prometheus uses a time-series database that can handle over a million time series in a single instance. The time-series database stores data in structured key-value pairs on a local disk, and uses the Go language for the server code. For example, the following data stores the average amount of CPU time spent in system mode, per second, over the last minute.

```
rate(node_cpu_seconds_total{mode="system"}[1m])
```

Prometheus Architecture

Figure 12.1 shows the overall architecture of Prometheus.

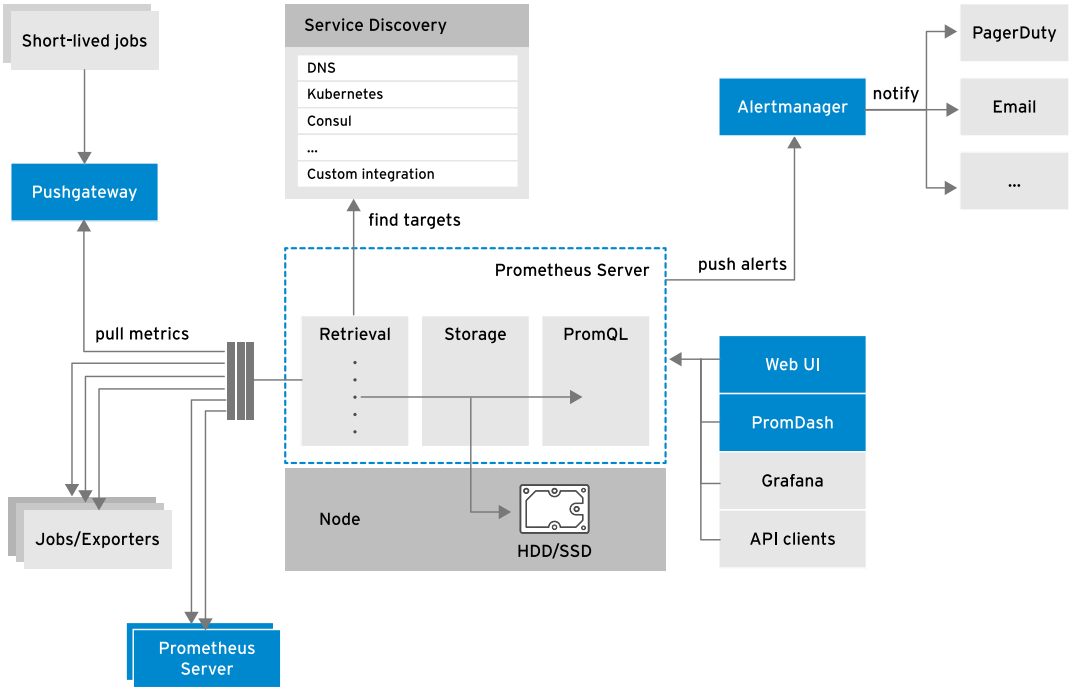


Figure 12.1: Prometheus architecture

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Service Discovery

Prometheus discovers monitoring targets using service discovery, which adds targets dynamically. It also supports adding targets manually in the Prometheus configuration file. The `http://localhost:9090/targets` endpoint lists the current targets.

Scrape

Prometheus use a pull method to collect metrics from monitored systems, known as a *scrape*. A scrape configuration defines the targets and parameters to be collected in a single task. Gathered metric samples are grouped into blocks of two hours. Each two-hour block consists of a directory containing one or more chunk files that contain all time series samples for that window of time. The Prometheus data pulling approach is different from many monitoring tools, such as Graphite, that wait on targets to push their metrics to the monitoring server.

Prometheus provides a functional query language called PromQL (Prometheus Query Language) that lets the user select and aggregate time series data in real time. The result of an expression can either be shown as a graph, viewed as tabular data in Prometheus's expression browser, or consumed by external systems via the HTTP API. The Prometheus server API is located at `http://localhost:9090/metrics`, and returns a complete list of metrics and their values.

Alerting

Prometheus allows adding alerting rules for the metrics collected. The Alertmanager component of Prometheus stores the alert levels can send notifications through multiple channels such as SMS, email, or chat.

Exporter

Exporters are client libraries that define metrics to be gathered from targets, formatted as Prometheus metrics samples. Exported metrics can be for database, hardware, messaging, issue tracker, storage, HTTP, or other software systems. Refer to https://prometheus.io/docs/instrumenting/exporters/ for the list of existing exporters. To export the operating system and hardware related metrics use the **`node_exporter`** exporter.

Push Gateway

When targets are behind a firewall such that Prometheus cannot directly access them to pull metrics, or when target processes are too short-lived to be located by Prometheus, targets can push metrics instead. A metrics cache called a *Pushgateway* is configured to store pushed metrics, and to expose the metrics to Prometheus for scraping.

Installing and Configuring Prometheus

Download and extract **`node_exporter`** on all the targets. Run **`node_exporter`**, either by using the binary in the extracted directory or by creating a **`systemd`** startup script. The following example shows **`node_exporter`** using the binary file from the extracted directory.

```
[root@host node_exporter-0.18.1.linux-amd64]# ./node_exporter
INFO[0000] Starting node_exporter (version=0.18.1, branch=HEAD,
 revision=3db77732e925c08f675d7404a8c46466b2ece83e)  source="node_exporter.go:156"
INFO[0000] Build context (go=go1.12.5, user=root@b50852a1acba,
 date=20190604-16:41:18)  source="node_exporter.go:157"
INFO[0000] Enabled collectors:              source="node_exporter.go:97"
INFO[0000]  - arp                           source="node_exporter.go:104"
...output omitted...
INFO[0000]  - cpu                           source="node_exporter.go:104"
INFO[0000]  - cpufreq                       source="node_exporter.go:104"
INFO[0000]  - diskstats                     source="node_exporter.go:104"
INFO[0000]  - edac                          source="node_exporter.go:104"
```

```
INFO[0000]  - entropy                        source="node_exporter.go:104"
INFO[0000]  - filefd                         source="node_exporter.go:104"
INFO[0000]  - filesystem                     source="node_exporter.go:104"
INFO[0000]  - hwmon                          source="node_exporter.go:104"
INFO[0000]  - infiniband                     source="node_exporter.go:104"
INFO[0000]  - ipvs                           source="node_exporter.go:104"
INFO[0000]  - loadavg                        source="node_exporter.go:104"
INFO[0000]  - mdadm                          source="node_exporter.go:104"
INFO[0000]  - meminfo                        source="node_exporter.go:104"
...output omitted...
INFO[0000]  - uname                          source="node_exporter.go:104"
INFO[0000]  - vmstat                         source="node_exporter.go:104"
INFO[0000]  - xfs                            source="node_exporter.go:104"
INFO[0000]  - zfs                            source="node_exporter.go:104"
INFO[0000] Listening on :9100                source="node_exporter.go:170"
```

Ensure that the **`firewalld`** service allows packets destined for port 9100 on targets. When **`node_exporter`** is running, verify that metrics are accessible at `http://localhost:9100/metrics`. Use the following command to verify that the target is exposing the metrics:

```
[root@host ~]# curl http://localhost:9100/metrics | grep "node_"
...output omitted...
node_boot_time_seconds 1.565585336e+09
# HELP node_context_switches_total Total number of context switches.
# TYPE node_context_switches_total counter
node_context_switches_total 364832
# HELP node_cpu_guest_seconds_total Seconds the cpus spent in guests (VMs) for
 each mode.
# TYPE node_cpu_guest_seconds_total counter
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 14.25
node_cpu_guest_seconds_total{cpu="1",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="1",mode="user"} 18.39
...output omitted...
```

Download and extract the Prometheus distribution from the Prometheus download page. Configure the scrape configuration for all the targets in the **`prometheus.yml`** file present under the extracted directory.

The **`prometheus.yml`** file uses YAML format, with white space indentation characters to indicate its data structure. The **`global`** section defines the scrape interval for all the scrape configurations in the file. A collection of targets with the same purpose are configured as a *job*job and are given a **`job_name`**. The **`scrape_config`** section specifies a set of targets and parameters describing how to scrape them, defining static targets using **`static_configs`** and dynamic targets using service discovery. In the following example, a static target using **`node_exporter`** running on the 9100 port is added to the **`prometheus.yml`** file.

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s
  # scrape_timeout is set to the global default (10s).
scrape_configs:
- job_name: 'demo'
  static_configs:
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
      - targets:
          - host.lab.example.com:9100
        labels:
          name: demo-targets
```

For file-based service discovery configuration, refer to *Configuration* at https://prometheus.io/docs/prometheus/latest/configuration/configuration/#file_sd_config.

Use the **`./promtool check config`** command to verify that the **`prometheus.yml`** configuration file is valid.

```
[root@host prometheus-2.11.1.linux-amd64]# ./promtool check config prometheus.yml
Checking prometheus.yml
 SUCCESS: 0 rule files found
```

The following example shows Prometheus running from the binary file.

```
[root@host prometheus-2.11.1.linux-amd64]# ./prometheus \
--storage.tsdb.path /var/lib/prometheus/
level=info ts=2019-08-12T05:16:23.301Z caller=main.go:293 msg="no time or size
 retention was set so using the default time retention" duration=15d
level=info ts=2019-08-12T05:16:23.301Z caller=main.go:329
 msg="Starting Prometheus" version="(version=2.11.1, branch=HEAD,
 revision=e5b22494857deca4b806f74f6e3a6ee30c251763)"
...output omitted...
level=info ts=2019-08-12T05:16:23.302Z caller=main.go:652 msg="Starting TSDB ..."
level=info ts=2019-08-12T05:16:23.302Z caller=web.go:448 component=web msg="Start
 listening for connections" address=0.0.0.0:9090
level=info ts=2019-08-12T05:16:23.308Z caller=main.go:667 fs_type=XFS_SUPER_MAGIC
level=info ts=2019-08-12T05:16:23.308Z caller=main.go:668 msg="TSDB started"
level=info ts=2019-08-12T05:16:23.308Z caller=main.go:738 msg="Loading
 configuration file" filename=prometheus.yml
level=info ts=2019-08-12T05:16:23.309Z caller=main.go:766 msg="Completed loading
 of configuration file" filename=prometheus.yml
level=info ts=2019-08-12T05:16:23.309Z caller=main.go:621 msg="Server is ready to
 receive web requests."
```

By default, Prometheus stores the time-series data in the **`data`** subdirectory under the current working directory. Use the **`--storage.tsdb.path`** option to specify a path to store the time-series database.

A Prometheus service can be verified by querying `http://localhost:9090`. Available metrics are listed at the `http://localhost:9090/metrics` endpoint, and the scrape targets at the `http://localhost:9090/targets` endpoint.

Using Grafana for Visualization

Grafana is an open source project that queries and visualizes metrics data from various data sources. The metrics data is visualized using dynamic dashboards, which are made reusable using template variables. In Grafana, dashboards are a set of one or more panels organized as rows. Grafana supports the following data sources: Graphite, InfluxDB, OpenTSDB, Prometheus, Elasticsearch, and CloudWatch.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Install Grafana by downloading the package from the following URL: https://grafana.com/grafana/download?platform=linux. Start the Grafana server using the **`systemd`** startup script provided by the *grafana* package.

```
[root@host ~]# systemctl daemon-reload
[root@host ~]# systemctl start grafana-server
[root@host ~]# systemctl status grafana-server
● grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Mon 2019-08-12 10:52:56 IST; 5s ago
     Docs: http://docs.grafana.org
 Main PID: 4045 (grafana-server)
    Tasks: 10 (limit: 11274)
   Memory: 49.6M
...output omitted...
... grafana-server[4045]: t=2019-08-12T10:52:56+0530 lvl=info msg="Initializing
 Stream Manager"
... grafana-server[4045]: t=2019-08-12T10:52:56+0530 lvl=info msg="HTTP Server
 Listen" logger=http.server address=0.0.0.0:3000 protocol=http subUrl= socket=
... systemd[1]: Started Grafana instance.
```

By default, Grafana listens on `http://localhost:3000`. Log in to the Grafana web UI as **`admin`** using **`admin`** as the password.

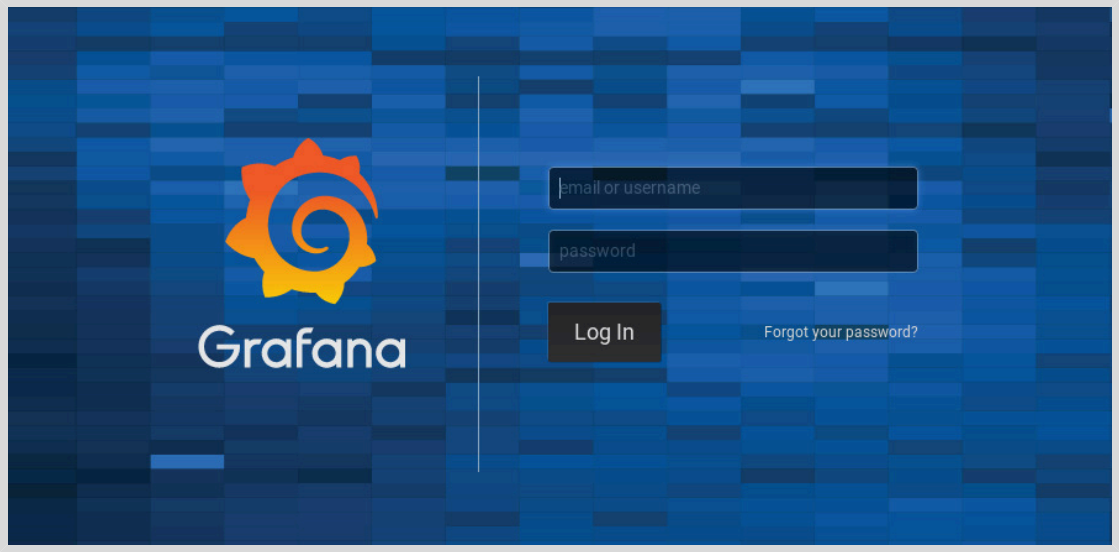


Figure 12.2: Grafana web UI

Grafana uses data sources to fetch information used by the graphs. A Grafana dashboard can have graphs from multiple data sources in the same graphic panel. To add Prometheus as the data source, use the Grafana web UI to locate the **Choose data source type** page. Click the **Prometheus** button and enter the address of the Prometheus server.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

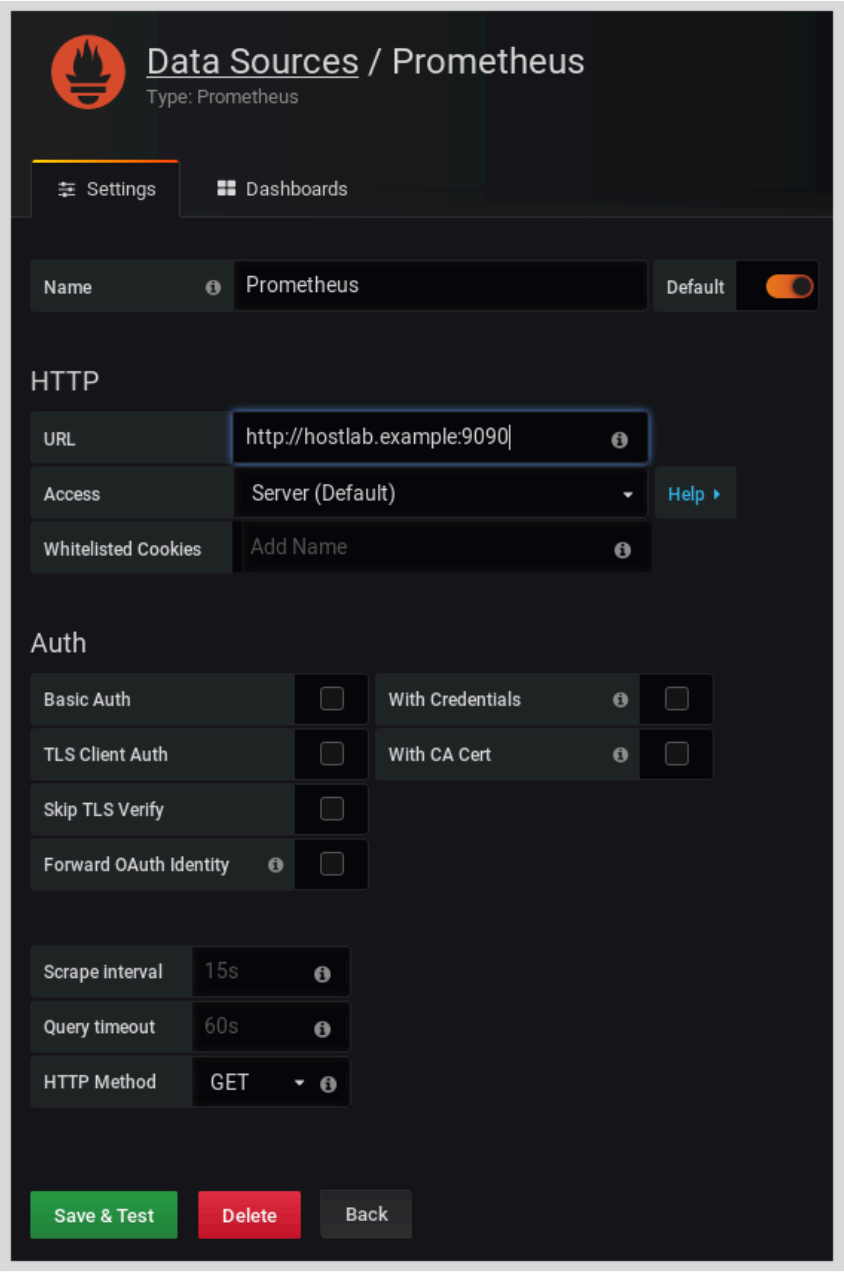


Figure 12.3: Adding a Prometheus data source to Grafana

Grafana maintains a collection of shareable, reusable dashboards that can be customized using a JSON file. Download the associated JSON file at https://grafana.com/grafana/dashboards. The `datasource:` entry reflects the currently configured Grafana data source. Navigate to **Dashboards** → **Manage** to add a dashboard. Click **Import** to import the configured JSON file. Select **Prometheus** from the **Select a Prometheus data source** menu and click **Import**.

References

For more information on Prometheus, refer to the various guides at https://prometheus.io/docs/guides/

For more information on Grafana, refer to https://grafana.com/docs/installation/configuration/

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Guided Exercise

Monitoring Performance Metrics in a Virtualized Environment

In this exercise, you will configure performance monitoring tools for virtual machines.

Outcomes

You should be able:

- Install and configure Node Exporter to expose a wide variety of hardware and kernel-related metrics on **servera** and **serverb**.
- Install and configure Prometheus on **servera** to collect metrics from targets; **servera** and **serverb**.
- Install and configure Grafana on **workstation** to visualize the metrics using Prometheus as the data source.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab virt-monitor start** command. This command determines if the host, **servera**, is reachable on the network. On **servera**, the command downloads Prometheus and Node Exporter archive, and the *grafana* package. This command also launches a virtual guest on **servera** named **rhel8-lab**.

```
[student@workstation ~]$ lab virt-monitor start
```

1. On **workstation** open two terminal windows. From the first terminal, use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

2. From the second terminal, log in to the **rhel8-lab** virtual machine, accessed from the **servera** host as **root**.

   2.1. Use **ssh** to log in to **servera** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@servera
[student@servera ~]$ sudo -i
[root@servera ~]#
```

   2.2. Retrieve IP address of the **rhel8-lab** virtual machine.

```
[root@servera ~]# virsh domifaddr rhel8-lab
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet0      52:54:00:ed:95:42    ipv4         192.168.122.12/24
```

2.3. Use **ssh** with the **rhel8-lab** IP address to log in as **root** user using **redhat** as the password.

```
[root@servera ~]# ssh root@192.168.122.12
root@192.168.122.12's password: redhat
[root@localhost ~]#
```

3. On **servera** and **rhel8-lab**, install, configure, and start the **node_exporter** service.

3.1. On **servera**, change to the **/usr/lib** directory. Extract the **/root/node_exporter-0.18.1.linux-amd64.tar.gz** file.

```
[root@servera ~]# cd /usr/lib
[root@servera lib]# tar xzvf /root/node_exporter-0.18.1.linux-amd64.tar.gz
...output omitted...
[root@servera lib]# cd
```

3.2. Create a new **systemd** unit file with the following content:

```
[root@servera ~]# vim /etc/systemd/system/node_exporter.service
[Unit]
Description=node exporter
After=network.target

[Service]
ExecStart=/usr/lib/node_exporter-0.18.1.linux-amd64/node_exporter

[Install]
WantedBy=default.target
```

3.3. Reload the **systemd** configuration. Enable and start the **node_exporter** service.

```
[root@servera ~]# systemctl daemon-reload
[root@servera ~]# systemctl start node_exporter
[root@servera ~]# systemctl enable node_exporter
Created symlink /etc/systemd/system/default.target.wants/node_exporter.service → /
etc/systemd/system/node_exporter.service.
```

3.4. Use the second terminal to copy the **/usr/lib/node_exporter-0.18.1.linux-amd64** directory and the **/etc/systemd/system/node_exporter.service** file from **servera** to **rhel8-lab**. Use **redhat** as the password for connecting to **servera** as **root**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@localhost ~]# scp -r \
root@servera:/usr/lib/node_exporter-0.18.1.linux-amd64 /usr/lib
root@servera's password: redhat
...output omitted...
[root@localhost ~]# scp \
root@servera:/etc/systemd/system/node_exporter.service /etc/systemd/system
root@servera's password: redhat
...output omitted...
```

3.5. On **rhel8-lab**, reload the **systemd** configuration. Enable and start the **node_exporter** service.

```
[root@localhost ~]# systemctl daemon-reload
[root@localhost ~]# systemctl start node_exporter
[root@localhost ~]# systemctl enable node_exporter
Created symlink /etc/systemd/system/default.target.wants/node_exporter.service → /
etc/systemd/system/node_exporter.service.
```

3.6. Open the **9100** port on **servera**. On **rhel8-lab**, because the **firewalld** service is not running, all ports are open in the classroom environment.

```
[root@servera ~]# firewall-cmd --add-port=9100/tcp --permanent
success
[root@servera ~]# firewall-cmd --reload
success
```

3.7. Verify that the **node_exporter** service is exposing the metrics. On **workstation**, open a browser to view the metrics from **servera** at `http://servera.lab.example.com:9100/metrics`.

4. On **servera**, install, configure, and start the **prometheus** server service.

4.1. Change to the **/usr/lib** directory. Extract the **/root/prometheus-2.11.1.linux-amd64.tar.gz** file.

```
[root@servera ~]# cd /usr/lib
[root@servera lib]# tar xzvf /root/prometheus-2.11.1.linux-amd64.tar.gz
...output omitted...
```

4.2. Configure Prometheus to collect metrics from **node_exporter** running on **servera** and **rhel8-lab**. Change to the **/usr/lib/prometheus-2.11.1.linux-amd64** directory. Edit the **prometheus.yml** file to add **servera** and **rhel8-lab** scrape configuration under the **scrape_configs:** section.

Use the IP address associated with **rhel8-lab**.

Important

The **prometheus.yml** file is written in YAML format and uses indentation with space characters to indicate the structure of its data.

```
[root@servera lib]# cd prometheus-2.11.1.linux-amd64
[root@servera prometheus-2.11.1.linux-amd64]# cat prometheus.yml
...output omitted...
scrape_configs:
  - job_name: 'virt-hosts'
    static_configs:
      - targets:
          - servera.lab.example.com:9100
        labels:
          name: hypervisor
  - job_name: 'virt-guests'
    static_configs:
      - targets:
          - 192.168.122.12:9100
        labels:
          name: virtual-machines
```

4.3. Verify that the Prometheus configuration file is valid. Resolve any errors before proceeding.

Change to the **root** user's home directory.

```
[root@servera prometheus-2.11.1.linux-amd64]# ./promtool check config \
prometheus.yml
Checking prometheus.yml
 SUCCESS: 0 rule files found
[root@servera prometheus-2.11.1.linux-amd64]# cd
[root@servera ~]#
```

4.4. Create a directory named **/var/lib/prometheus** to store the time-series database.

```
[root@servera ~]# mkdir /var/lib/prometheus
```

4.5. Create a new **systemd** unit file for the **prometheus** service with the following content:

```
[root@servera ~]# vim /etc/systemd/system/prometheus.service
[Unit]
Description=prometheus
After=network.target

[Service]
WorkingDirectory=/usr/lib/prometheus-2.11.1.linux-amd64
ExecStart=/usr/lib/prometheus-2.11.1.linux-amd64/prometheus \
          --storage.tsdb.path /var/lib/prometheus/

[Install]
WantedBy=default.target
```

4.6. Reload the **systemd** configuration. Enable and start the **prometheus** service.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@servera ~]# systemctl daemon-reload
[root@servera ~]# systemctl start prometheus
[root@servera ~]# systemctl enable prometheus
Created symlink /etc/systemd/system/default.target.wants/prometheus.service → /
etc/systemd/system/prometheus.service.
```

4.7. Verify that the Prometheus server is collecting the metrics. On **`workstation`**, open a browser to view configured targets at `http://servera.lab.example.com:9090/targets`. All the targets must be in the **`UP`** state.

5. On **`workstation`**, install, configure, and start the **`grafana-server`** service for visualization.

5.1. Install the **`/home/student/grafana-6.2.5-1.x86_64.rpm`** file.

```
[student@workstation ~]$ sudo yum localinstall grafana-6.2.5-1.x86_64.rpm
[sudo] password for student: student
...output omitted...
```

5.2. Start the **`grafana-server`** service.

```
[student@workstation ~]$ sudo systemctl start grafana-server
```

5.3. Verify that the **`grafana`** server is available. On **`workstation`**, open a browser to view configured targets at `http://workstation.lab.example.com:3000`. Log in to the **`grafana`** web UI as **`admin`** using **`admin`** as the password.

Set the new password to **`redhat`**.

6. Use the **`grafana`** web UI to configure data source.

6.1. Click **Add data source** to add Prometheus running on **`servera.lab.example.com`** as the data source.

6.2. Click **Prometheus** on the **Choose data source type** page. Enter `http://servera.lab.example.com:9090` in the **URL** text field. Click **Save & Test**.

Click **Back**.

7. Use the **`grafana`** web UI to configure a dashboard.

7.1. On **`workstation`**, download the JSON file from `http://materials.example.com/labs/virt-monitor/node-exporter-full_rev14.json` to **`/home/student`**.

```
[student@workstation ~]$ wget \
http://materials.example.com/labs/virt-monitor/node-exporter-full_rev14.json
```

7.2. From the **`grafana`** web UI, navigate to **Dashboards** → **Manage** to add a dashboard.

7.3. Click **Import**. On the **Import** page, click **Upload .json File**. Browse to and select the **`/home/student/node-exporter-full_rev14.json`** file to upload.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

7.4. Enter **`Performance Metrics`** in the **Name** text field. Select **Prometheus** from the **Select a Prometheus data source** menu. Click **Import**.

7.5. Use the refresh drop-down list in the upper-right corner to set the refresh period to 5 seconds.

7.6. You must now be able to visualize the metrics of **`rhel8-lab`** on the dashboard. Select **virt-hosts** from **Job** to view the performance metrics of **`servera`**.

Scroll down to view other performance metrics.

7.7. Click the icon at the bottom above the help icon and select **Sign out** to log off from the Grafana web UI.

7.8. Log off from **`servera`** and **`rhel8-lab`** from both the terminals.

Finish

On **`workstation`**, run the **`lab virt-monitor finish`** command to complete this exercise. This command stops and disables the **`prometheus`**, **`grafana-server`**, and **`node_exporter`** services. This command also removes **`rhel8-lab`** running on **`servera`**.

```
[student@workstation ~]$ lab virt-monitor finish
```

This concludes the guided exercise.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Performing Tuning in Virtualization Environments

Performance Checklist

In this lab, you will tune a virtual environment.

Outcomes

You should be able to tune both a KVM virtual machine and a host to optimize performance in a virtual environment.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab virt-review start** command. This command determines if the host, **serverb**, is reachable on the network and defines the **rhel8-lab** VM.

```
[student@workstation ~]$ lab virt-review start
```

You will tune your virtual environment consisting of the **serverb** hosts and the virtual machines running on it. You will optimize CPU, disk, and memory settings and launch a virtual machine named **rhel8-lab-tuned**. Use the **rhel8-lab** VM to create an optimized libvirtd XML file to define and launch the **rhel8-lab-tuned** VM.

1. On **serverb**, set the current active tuned profile to **virtual-host**.
2. Pin the emulator thread processes to use the first CPU on **serverb**.
3. Pin the virtual machine CPUs to the second CPU on **serverb**.
4. Set the CPU share such that the **rhel8-lab-tuned** VM is allocated twice the CPU time as all other running virtual machines.
5. Set the memory soft limit to 512 MiB and the hard limit to 1024 MiB for the **rhel8-lab-tuned** VM.
6. Enable KSM on **serverb**. Set the **shm_pages_to_scan** shared memory parameter to **100** and the **shm_sleep_millisecs** shared memory parameter to **10**. Enable merging of shared memory across NUMA nodes.
7. Attach the **/dev/vdb1** disk from **serverb** to the **rhel8-lab-tuned** VM as the **vdb** device. Set the **vdb** device total I/O operations per second to **1000**.

   Use the following XML snippet to attach the **vdb** disk to the virtual machine.

```
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native'/>
      <source dev='/dev/vdb1'/>
      <target dev='vdb' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </disk>
```

8. Launch the **rhel8-lab-tuned** VM.

Evaluation

On **workstation**, run the **lab virt-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab virt-review grade
```

Finish

On **workstation**, run the **lab virt-review finish** command to complete this lab. The command removes all the configured VMs running on **serverb**.

```
[student@workstation ~]$ lab virt-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Performing Tuning in Virtualization Environments

Performance Checklist

In this lab, you will tune a virtual environment.

Outcomes

You should be able to tune both a KVM virtual machine and a host to optimize performance in a virtual environment.

Before You Begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab virt-review start** command. This command determines if the host, **serverb**, is reachable on the network and defines the **rhel8-lab** VM.

```
[student@workstation ~]$ lab virt-review start
```

You will tune your virtual environment consisting of the **serverb** hosts and the virtual machines running on it. You will optimize CPU, disk, and memory settings and launch a virtual machine named **rhel8-lab-tuned**. Use the **rhel8-lab** VM to create an optimized libvirtd XML file to define and launch the **rhel8-lab-tuned** VM.

1. On **serverb**, set the current active tuned profile to **virtual-host**.

   1.1. Log in to **serverb** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

   1.2. Set the current active tuned profile to **virtual-host**.

```
[root@serverb ~]# tuned-adm profile virtual-host
```

2. Pin the emulator thread processes to use the first CPU on **serverb**.

   2.1. Use the **virsh** command to ensure that the **rhel8-lab** VM is available and **shut off**.

```
[root@serverb ~]# virsh list --all
Id    Name                           State
----------------------------------------------------
-     rhel8-lab                      shut off
```

2.2. List the host CPUs available to guest VMs.

```
[root@serverb ~]# lscpu | grep -i '^cpu'
CPU op-mode(s):      32-bit, 64-bit
CPU(s):              2
CPU family:          6
CPU MHz:             3591.678
```

2.3. Use the **`virsh emulatorpin`** command to view the CPU affinity for emulator thread processes for the **`rhel8-lab`** VM.

```
[root@serverb ~]# virsh emulatorpin rhel8-lab
emulator: CPU Affinity
----------------------------------
       *: 0-1
```

The CPU affinity of emulator threads for the **`rhel8-lab`** VM is balanced between both the CPUs on **`serverb`**.

2.4. Use the **`virsh emulatorpin`** command to pin the emulator thread processes to use the first CPU on **`serverb`**.

```
[root@serverb ~]# virsh emulatorpin rhel8-lab --config 0
```

2.5. View the CPU affinity for emulator threads for the **`rhel8-lab`** VM.

```
[root@serverb ~]# virsh emulatorpin rhel8-lab
emulator: CPU Affinity
----------------------------------
       *: 0
```

3. Pin the virtual machine CPUs to the second CPU on **`serverb`**.

3.1. List CPU information for the **`rhel8-lab`** VM to determine that it is balanced between them and not pinned to a CPU.

```
[root@serverb ~]# virsh vcpuinfo rhel8-lab
VCPU:           0
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   yy

VCPU:           1
CPU:            N/A
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
State:          N/A
CPU time        N/A
CPU Affinity:   yy
```

3.2. Use the **`virsh vcpupin`** command to pin the **`rhel8-lab`** VM virtual CPUs to the second CPU on **`serverb`**.

```
[root@serverb ~]# virsh vcpupin rhel8-lab --config 0 1
[root@serverb ~]# virsh vcpupin rhel8-lab --config 1 1
```

3.3. Use the **`virsh vcpuinfo`** command to view the CPU affinity for the **`rhel8-lab`** VM.

```
[root@serverb ~]# virsh vcpuinfo rhel8-lab
VCPU:           0
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   -y

VCPU:           1
CPU:            N/A
State:          N/A
CPU time        N/A
CPU Affinity:   -y
```

4. Set the CPU share such that the **`rhel8-lab-tuned`** VM is allocated twice the CPU time as all other running virtual machines.

4.1. Because the default value of CPU shares is **`1024`**, to provide twice the CPU time as all other running virtual machines, the **`rhel8-lab`** VM must be set to **`2048`**. Use the **`virsh schedinfo`** command to set the CPU shares to **`2048`** for the **`rhel8-lab`** VM.

```
[root@serverb ~]# virsh schedinfo rhel8-lab --config cpu_shares=2048
Scheduler      : posix
cpu_shares     : 2048
vcpu_period    : 0
vcpu_quota     : 0
emulator_period: 0
emulator_quota : 0
global_period  : 0
global_quota   : 0
iothread_period: 0
iothread_quota : 0
```

4.2. View the CPU tuning parameters in the libvirtd XML file of the **`rhel8-lab`** VM.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# virsh dumpxml rhel8-lab | sed -n '/<cputune>/,/<\/cputune>/p'
  <cputune>
    <shares>2048</shares>
    <vcpupin vcpu='0' cpuset='1'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <emulatorpin cpuset='0'/>
  </cputune>
```

5. Set the memory soft limit to 512 MiB and the hard limit to 1024 MiB for the **`rhel8-lab-tuned`** VM.

5.1. Use the **`virsh memtune`** command to view the currently configured memory limits.

```
[root@serverb ~]# virsh memtune rhel8-lab
hard_limit     : unlimited
soft_limit     : unlimited
swap_hard_limit: unlimited
```

5.2. View the current memory allocated to the **`rhel8-lab`** VM.

```
[root@serverb ~]# virsh dominfo rhel8-lab
Id:             -
Name:           rhel8-lab
UUID:           81a7a4c1-7c0e-49b7-8f8a-a8e79b579879
OS Type:        hvm
State:          shut off
CPU(s):         2
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
```

The current allocated memory is 1024 MiB.

5.3. Set the soft limit for the **`rhel8-lab`** VM to 512 MiB of the physical host memory.

```
[root@serverb ~]# virsh memtune rhel8-lab --config --soft-limit 512MiB
```

5.4. Set the hard limit for the **`rhel8-lab`** VM to 1024 MiB of the physical host memory.

```
[root@serverb ~]# virsh memtune rhel8-lab --config --hard-limit 1024MiB
```

5.5. View the memory tuning parameters in the libvirtd XML file of the **`rhel8-lab`** VM.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# virsh dumpxml rhel8-lab | sed -n '/<memtune>/,/<\/memtune>/p'
  <memtune>
    <hard_limit unit='KiB'>1048576</hard_limit>
    <soft_limit unit='KiB'>524288</soft_limit>
  </memtune>
```

6. Enable KSM on **serverb**. Set the **shm_pages_to_scan** shared memory parameter to **100** and the **shm_sleep_millisecs** shared memory parameter to **10**. Enable merging of shared memory across NUMA nodes.

6.1. Start and enable the **ksm** and **ksmtuned** services on **serverb**.

```
[root@serverb ~]# systemctl enable ksm --now
[root@serverb ~]# systemctl enable ksmtuned --now
```

6.2. Use the **virsh node-memory-tune** command to set the **shm_pages_to_scan** shared memory parameter to **100** and the **shm_sleep_millisecs** shared memory parameter to **10**. Enable merging of shared memory across NUMA nodes.

```
[root@serverb ~]# virsh node-memory-tune \
--shm-pages-to-scan 100 \
--shm-sleep-millisecs 10 \
--shm-merge-across-nodes 1
```

6.3. Verify that the KSM tunables are set in **/sys/kernel/mm/ksm/**.

```
[root@serverb ~]# cat /sys/kernel/mm/ksm/pages_to_scan
100
[root@serverb ~]# cat /sys/kernel/mm/ksm/sleep_millisecs
10
[root@serverb ~]# cat /sys/kernel/mm/ksm/merge_across_nodes
1
```

7. Attach the **/dev/vdb1** disk from **serverb** to the **rhel8-lab-tuned** VM as the **vdb** device. Set the **vdb** device total I/O operations per second to **1000**.

Use the following XML snippet to attach the **vdb** disk to the virtual machine.

```
  <disk type='block' device='disk'>
    <driver name='qemu' type='raw' cache='none' io='native'/>
    <source dev='/dev/vdb1'/>
    <target dev='vdb' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
  </disk>
```

7.1. Use the **virsh domblklist** command to list the block devices attached to the **rhel8-lab** VM.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# virsh domblklist rhel8-lab
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rhel8-lab.ovl
```

7.2. Attach the **/dev/vdb1** disk on **serverb** to the **rhel8-lab** VM as **vdb**. Use the **virsh edit** command and edit the libvirtd XML file with the following content. Add the following to the existing **<disk>** block:

```
[root@serverb ~]# virsh edit rhel8-lab
    ...output omitted...
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native'/>
      <source dev='/dev/vdb1'/>
      <target dev='vdb' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </disk>
    ...output omitted...
```

7.3. Use the **virsh blkdeviotune** command to set the total number of I/O operations per second to **1000** for the **vdb** device attached to the **rhel8-lab** VM.

```
[root@serverb ~]# virsh blkdeviotune rhel8-lab vdb --config --total-iops-sec 1000
```

8. Launch the **rhel8-lab-tuned** VM.

8.1. Rename the **rhel8-lab** VM to **rhel8-lab-tuned**.

```
[root@serverb ~]# virsh domrename rhel8-lab rhel8-lab-tuned
Domain successfully renamed
```

8.2. Create a new libvirtd XML file named **/var/lib/libvirt/images/rhel8-lab-tuned.xml**.

```
[root@serverb ~]# virsh dumpxml rhel8-lab-tuned > \
/var/lib/libvirt/images/rhel8-lab-tuned.xml
```

8.3. Remove the **rhel8-lab-tuned** definition.

```
[root@serverb ~]# virsh undefine rhel8-lab-tuned
Domain rhel8-lab-tuned has been undefined
```

8.4. Define the **rhel8-lab-tuned** VM using the **/var/lib/libvirt/images/rhel8-lab-tuned.xml** file.

```
[root@serverb ~]# virsh define /var/lib/libvirt/images/rhel8-lab-tuned.xml
Domain rhel8-lab-tuned defined from /var/lib/libvirt/images/rhel8-lab-tuned.xml
```

8.5. Start the **rhel8-lab-tuned** VM.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# virsh start rhel8-lab-tuned
Domain rhel8-lab-tuned started
```

8.6. Log off from **serverb**.

```
[root@servera ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab virt-review grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab virt-review grade
```

Finish

On **workstation**, run the **lab virt-review finish** command to complete this lab. The command removes all the configured VMs running on **serverb**.

```
[student@workstation ~]$ lab virt-review finish
```

This concludes the lab.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Summary

In this chapter, you learned:

- The performance impact of cache misses on a NUMA-based system are significant. Therefore virtual CPU pinning and NUMA tuning must be configured together.
- Kernel Samepage Merging (KSM) scans the identical memory pages and consolidates a Copy On Write (COW) shared pages, which helps to reduce physical memory consumption.
- Prometheus determines what to scrape using a static configuration defined in the **`static_configs`** section of the **`prometheus.yml`** file.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 13

Comprehensive Review

Goal

Review tasks from *Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud*

Objectives

- Review tasks from *Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud*

Sections

- Lab: Tuning for a Computation-heavy Application
- Lab: Tuning for a Storage Intensive Application
- Lab: Tuning for a Large Memory implementation

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Comprehensive Review

Objectives

After completing this section, students should have reviewed and refreshed the knowledge and skills learned in *Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud*.

Reviewing Red Hat Performance Tuning: Linux in Physical, Virtual, and Cloud

Before beginning the comprehensive review for this course, students should be comfortable with the topics covered in each chapter.

Students can refer to earlier sections in the textbook for extra study.

Chapter 1, Introducing Performance Tuning

Describe performance tuning concepts and goals.

- Define performance tuning goals, expectations, and terminology.
- Discuss recommended performance tuning methods of analysis.
- Define the standard units of measure and conversion arithmetic.

Chapter 2, Selecting Performance Monitoring Tools

Evaluating the large selection of performance monitoring tools that are included with Red Hat Enterprise Linux.

- Identify the common system monitoring tools and explain the purpose of each tool.
- Describe the sysstat architecture and explain typical sysstat utility command use.
- Describe the Performance Co-Pilot utility and its standardized data structures.

Chapter 3, Viewing Hardware Resources

View and interpret hardware resource listings.

- Describe the utilities that list system hardware resources.
- Describe the techniques used to list resources in virtual and cloud instances.

Chapter 4, Configuring Kernel Tunables and Tuned Profiles

Configure the operating system to tune for different workload requirements.

- Configure settings for both the static kernel and for dynamic loadable kernel modules and drivers.
- Select an appropriate tuned profile to use, based on a system's most common workloads.
- Create customized tuned profiles for unique workloads.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Chapter 5, Managing Resource Limits with Control Groups

Manage resource contention and set limits for resource use on services, applications, and users using cgroup configuration.

- Configure resource settings for individual and groups of services, applications, and users to control system resource sharing.
- Customize service, application, and user resource limits using cgroup plug-in parameters.

Chapter 6, Analyzing Performance Using System Tracing Tools

Diagnose system and application behaviors using a variety of resource-specific tracing tools.

- Profile system events and observe performance counters with the **`perf`** command.
- Trace system and library calls for a process.
- Gather specific diagnostic data using the SystemTap infrastructure.
- Debug applications using tools instrumented by the in-kernel eBPF virtual machine.

Chapter 7, Tuning CPU Utilization

Manage CPU resource sharing and scheduling to control utilization.

- Configure CPU scheduling policies.
- Configure CPU and interrupt affinity to control application isolation and CPU resource commitments.
- Describe how the CPU caches are used by applications.

Chapter 8, Tuning Memory Utilization

Manage settings for efficient memory utilization for different types of workloads.

- Describe memory architecture, including virtual memory, caches, huge pages and process space.
- Describe system paging behavior and configure paging for server workload requirements.
- Describe the non-uniform memory architecture and the utilities for viewing and configuring NUMA topology.
- Describe memory overcommitment in typical application design.

Chapter 9, Tuning Storage Device I/O

Manage settings for efficient disk utilization in various use cases.

- Describe transaction-oriented and sequential I/O patterns and disk scheduling algorithms.
- Describe basic RAID architecture to understand the correlation between application writes and RAID sizing alignment.
- Describe multiple tool choices for analyzing disk I/O behavior.

Chapter 10, Tuning File System Utilization

Manage application efficiency for file system utilization.

- Describe the file system choices and tunable attributes.
- Configure file system journaling and discuss appropriate use cases.

Chapter 11, Tuning Network Utilization

Manage application efficiency for network utilization.

- Describe latency and throughput definitions and discuss the contrasting requirements for different network workloads.
- Configure network device parameters using **`ethtool`**.

Chapter 12, Performing Tuning Hosts in Virtualization Environments

Distinguish the requirements for tuning in virtualized environments.

- Explain the differences between tuning hosts in a virtualized environment and a cloud environment.
- Gather performance metrics from virtual machines using open source tools.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning for a Computation-heavy Application

In this review, you will limit CPU resources for a service with a slice, and pin a service to a specific CPU. You will also modify the CPU scheduling algorithm and priority for a service.

Outcomes

You should be able to:

- Create a slice to limit CPU resources for a service.
- Pin a service to a specific CPU.
- Configure the CPU scheduling algorithm and priority for a service.

Before You Begin

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-cpu start
```

Instructions

Demonstrate that you can configure the services specified in the list of criteria to limit their CPU resources on **serverb** as expected:

- Limit the CPU resources for the **frontend** service with a slice. Configure the slice to enable CPU accounting, and use a CPU time quota of **50 %**.
- Pin the **httpd** service to the first CPU.
- Configure the **rngd** service to use the **batch** CPU scheduler, and a CPU scheduling priority of **0**.

Evaluation

As **student** on **workstation**, run **lab compreview-cpu grade** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-cpu grade
```

Finish

On **workstation**, run the **lab compreview-cpu finish** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-cpu finish
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

This concludes the comprehensive review.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning for a Computation-heavy Application

In this review, you will limit CPU resources for a service with a slice, and pin a service to a specific CPU. You will also modify the CPU scheduling algorithm and priority for a service.

Outcomes

You should be able to:

- Create a slice to limit CPU resources for a service.
- Pin a service to a specific CPU.
- Configure the CPU scheduling algorithm and priority for a service.

Before You Begin

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-cpu start
```

Instructions

Demonstrate that you can configure the services specified in the list of criteria to limit their CPU resources on **serverb** as expected:

- Limit the CPU resources for the **frontend** service with a slice. Configure the slice to enable CPU accounting, and use a CPU time quota of **50 %**.
- Pin the **httpd** service to the first CPU.
- Configure the **rngd** service to use the **batch** CPU scheduler, and a CPU scheduling priority of **0**.

1. On serverb, create a slice for a CPU intensive service, named **frontend**. Configure that slice, named **system-frontend.slice**, to use **50 %** of the CPU time, and enable CPU accounting.

   1.1. On **workstation**, use **ssh** command to log in to **serverb** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

   1.2. Create a new file called **/etc/systemd/system/system-frontend.slice**, insert the following content, and save the changes.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# vim /etc/systemd/system/system-frontend.slice
[Unit]
Description=frontend Slice

[Slice]
CPUQuota=50%
CPUAccounting=yes
```

1.3. Create a drop-in directory for the **frontend** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/frontend.service.d
```

1.4. Create a drop-in file called **/etc/systemd/system/frontend.service.d/10-slice.conf**, to direct the service into the new slice. Insert the following content and save the changes.

```
[root@serverb ~]# vim /etc/systemd/system/frontend.service.d/10-slice.conf
[Service]
Slice=system-frontend.slice
```

1.5. Reload all **systemd** unit files from disk. Restart the **frontend** service.

```
[root@serverb ~]# systemctl daemon-reload
[root@serverb ~]# systemctl restart frontend
```

2. Verify that the **frontend** service is running in the new slice.

2.1. Inspect the output of the **systemctl status frontend** command.

```
[root@serverb ~]# systemctl status frontend | grep CGroup
   CGroup: /system.slice/system-frontend.slice/frontend.service
```

2.2. Use **systemd-cgls** command to inspect the **/system.slice/system-frontend.slice** cgroup.

```
[root@serverb ~]# systemd-cgls /system.slice/system-frontend.slice
Control group /system.slice/system-frontend.slice:
└─frontend.service
  └─2003 /usr/local/bin/frontend /dev/zero
```

3. Pin the **httpd** service to the first CPU. Remember than the **systemd** ID for the first CPU is **0**.

3.1. Install the *httpd* package.

```
[root@serverb ~]# yum install httpd
...output omitted...
```

3.2. Enable and start the **httpd** service.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[root@serverb ~]# systemctl enable httpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/
lib/systemd/system/httpd.service.
```

3.3. View the CPU assignments for all **`httpd`** processes and all of their threads. Observe that the CPU affinity for all threads is distributed across both the processors.

```
[root@serverb ~]# tuna -t httpd -P
                      thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  24095  OTHER     0      0,1        34            3           httpd
  24096  OTHER     0      0,1         6            0           httpd
  24097  OTHER     0      0,1         5            0           httpd
  24098  OTHER     0      0,1         7            0           httpd
  24099  OTHER     0      0,1         5            0           httpd
```

3.4. Create a **`systemd`** drop-in directory for the **`httpd.service`** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/httpd.service.d
```

3.5. Create a **`systemd`** drop-in file, called **`/etc/systemd/system/httpd.service.d/10-CPUAffinity.conf`**, for the **`httpd`** service to set the CPU affinity to use the first processor. Remember that **`systemd`** starts with **`0`** when counting CPUs, and that the **`CPUAffinity`** parameter takes a comma-separated list of CPU numbers.

```
[root@serverb ~]# vim /etc/systemd/system/httpd.service.d/10-CPUAffinity.conf
[Service]
CPUAffinity=0
```

3.6. Reload the **`systemd`** service to reload all unit definitions from disk. Restart the **`httpd`** service.

```
[root@serverb ~]# systemctl daemon-reload
[root@serverb ~]# systemctl restart httpd.service
```

4. Verify that the **`httpd`** service uses the first CPU.

4.1. View the CPU assignments for all **`httpd`** processes and all of their threads. Observe that the **`CPU0`** processor handles all the **`httpd`** processes.

```
[root@serverb ~]# tuna -t httpd -P
                      thread       ctxt_switches
    pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
  24344  OTHER     0        0        11            3           httpd
  24346  OTHER     0        0         6            0           httpd
  24347  OTHER     0        0         7            0           httpd
  24348  OTHER     0        0         7            0           httpd
  24349  OTHER     0        0         7            0           httpd
```

5. Configure the **`rngd`** service to use the **`batch`** scheduler, and a priority of **`0`**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5.1. Create a drop-in directory for the **rngd** service.

```
[root@serverb ~]# mkdir /etc/systemd/system/rngd.service.d
```

5.2. Create the **/etc/systemd/system/rngd.service.d/20-batch.conf** file with the following content.

```
[root@serverb ~]# vim /etc/systemd/system/rngd.service.d/20-batch.conf
[Service]
CPUSchedulingPolicy=batch
CPUSchedulingPriority=0
```

5.3. Reload **systemd** to reread the configuration changes. Restart the **rngd** service.

```
[root@serverb ~]# systemctl daemon-reload
[root@serverb ~]# systemctl restart rngd
```

6. Verify that the **rngd** process is now running using the **batch** scheduler and a priority of **0**. When done, log out from **serverb**.

6.1. Use the **ps** command to verify that the **rngd** process is now running using the **batch** scheduler and a priority of **0**.

```
[root@serverb ~]# ps o pid,cls,rtprio,comm $(pidof rngd)
  PID CLS RTPRIO COMMAND
 4489   B      0 rngd
```

6.2. Exit from all **serverb** terminals to return to **workstation** as the **student** user.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

As **student** on **workstation**, run **lab compreview-cpu grade** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-cpu grade
```

Finish

On **workstation**, run the **lab compreview-cpu finish** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-cpu finish
```

This concludes the comprehensive review.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning for a Storage Intensive Application

In this review, you will create a tuned profile based on the **virtual-guest** profile to customize the **readahead** tunable for a disk. You will also configure an XFS file system with external journaling.

Outcomes

You should be able to:

- Create a Tuned profile to modify a tunable for a disk.
- Configure an XFS file system with external journaling.

Before You Begin

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-disk start
```

Instructions

Demonstrate on **serverb** that you can tune a disk, and configure an XFS file system with external journaling:

- Create a custom Tuned profile, named **virt-guest-lab**, based on the **virtual-guest** profile. Set the **readahead** tunable to **4096** sectors for the **/dev/vdb** disk. Configure the summary as **Custom tuned profile**.
- Configure an XFS file system with external journaling with a partition on **vdb** for the data disk, and a partition on **vdc** for the journal. Use a block size of **4096** bytes, and an inode size of **512** bytes. Mount that file system in the **/bigfiles** directory.

Evaluation

As **student** on **workstation**, run **lab compreview-disk grade** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-disk grade
```

Finish

On **workstation**, run the **lab compreview-disk finish** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-disk finish
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

This concludes the comprehensive review.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning for a Storage Intensive Application

In this review, you will create a tuned profile based on the **virtual-guest** profile to customize the **readahead** tunable for a disk. You will also configure an XFS file system with external journaling.

Outcomes

You should be able to:

- Create a Tuned profile to modify a tunable for a disk.
- Configure an XFS file system with external journaling.

Before You Begin

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-disk start
```

Instructions

Demonstrate on **serverb** that you can tune a disk, and configure an XFS file system with external journaling:

- Create a custom Tuned profile, named **virt-guest-lab**, based on the **virtual-guest** profile. Set the **readahead** tunable to **4096** sectors for the **/dev/vdb** disk. Configure the summary as **Custom tuned profile**.
- Configure an XFS file system with external journaling with a partition on **vdb** for the data disk, and a partition on **vdc** for the journal. Use a block size of **4096** bytes, and an inode size of **512** bytes. Mount that file system in the **/bigfiles** directory.

1. On **serverb**, create a tuned profile named **virt-guest-lab**. Inherit the profile settings from the **virtual-guest** profile, and add **Custom tuned profile** as the summary.

   1.1. On **workstation**, use **ssh** to log in to **serverb** as **student**. Use **sudo -i** to switch to **root**.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

   1.2. Create the **/etc/tuned/virt-guest-lab** directory for the new custom profile.

```
[root@serverb ~]# mkdir /etc/tuned/virt-guest-lab
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

1.3. Inherit settings for this new profile from the **virtual-guest** profile. Add **Custom tuned profile** as the summary. In the **/etc/tuned/virt-guest-lab/tuned.conf** file, create a new section called **[main]** and add the following.

```
[root@serverb ~]# vim /etc/tuned/virt-guest-lab/tuned.conf
[main]
summary=Custom tuned profile
include=virtual-guest
```

Do not exit the file.

2. In the **virt-guest-lab** profile, set **readahead** to **4096** sectors on **/dev/vdb**.

2.1. Use the **disk** plug-in and set **readahead** to **4096** sectors for **vdb**. In the **/etc/tuned/virt-guest-lab/tuned.conf** file, create a new section called **[disk]** and add the following.

```
[disk]
devices=vdb
readahead=4096 sectors
```

Save and exit the file.

3. Apply the **virt-guest-lab** profile on **serverb**. Verify the current **readahead** is **4096**.

3.1. Configure **serverb** to use the **virt-guest-lab** profile.

```
[root@serverb ~]# tuned-adm profile virt-guest-lab
```

3.2. Verify the current value for the **readahead** tunable.

```
[root@serverb ~]# blockdev --getra /dev/vdb
4096
```

4. Create a partition on **vdb** for XFS data. Create a partition on **vdc** for an XFS external journal.

4.1. Create the partition label on both **vdb** and **vdc**.

```
[root@serverb ~]# parted /dev/vdb mklabel gpt
[root@serverb ~]# parted /dev/vdc mklabel gpt
```

4.2. Create a primary partition of 5 GiB on **vdb** and a primary partition of 1 GiB on **vdc**.

```
[root@serverb ~]# parted --align=opt /dev/vdb mkpart primary xfs 0% 100%
[root@serverb ~]# parted --align=opt /dev/vdc mkpart primary xfs 0% 1GiB
```

4.3. Force the partition table to reload.

```
[root@serverb ~]# partx -uv /dev/vdb
...output omitted...
[root@serverb ~]# partx -uv /dev/vdc
...output omitted...
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

4.4. Create an XFS file system on the 5 GiB partition, specifying the 1 GiB partition as the location for the external journal. Set the extended inode size to 512 bytes and the block size to 4096 bytes.

```
[root@serverb ~]# mkfs.xfs -i size=512 -b size=4096 -l logdev=/dev/vdc1 /dev/vdb1
```

5. Persistently mount the XFS file system on **/bigfiles**. Log out from **serverb** when done.

5.1. Create the mount point at **/bigfiles**.

```
[root@serverb ~]# mkdir /bigfiles
```

5.2. Configure an entry in **/etc/fstab** to persistently mount the newly created file system. Include the external journaling and the **relatime** mount options.

```
[root@serverb ~]# vim /etc/fstab
...output omitted...
/dev/vdb1  /bigfiles  xfs  relatime,logdev=/dev/vdc1  0 0
```

5.3. Mount the newly created XFS file system.

```
[root@serverb ~]# mount /dev/vdb1
```

5.4. Verify the mounting options of the newly created XFS file system.

```
[root@serverb ~]# findmnt --target /bigfiles
TARGET    SOURCE    FSTYPE OPTIONS
/bigfiles /dev/vdb1 xfs    rw,relatime,seclabel,attr2,inode64,logdev=/dev/
vdc1,noquota
```

5.5. Exit from all **serverb** terminals to return to **workstation** as the **student** user.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

As **student** on **workstation**, run **lab compreview-disk grade** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-disk grade
```

Finish

On **workstation**, run the **lab compreview-disk finish** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-disk finish
```

This concludes the comprehensive review.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Lab

Tuning for a Large Memory Implementation

In this review, you will create a custom tuned profile which sets various tunable values on a system running a database server.

Outcomes

You should be able to:

- Create a custom tuned profile that configures system parameters to tune memory for a database server.
- Restrict memory usage of a process.
- Configure a service to distribute memory usage across all NUMA nodes.

Before You Begin

Set up your computers for this exercise by logging in to **`workstation`** as **`student`**, and run the following command:

```
[student@workstation ~]$ lab compreview-memory start
```

Instructions

A pre-installed PostgreSQL database is running on **`serverb`**. On **`serverb`**, create a tuned profile named **`enterprise-database`** that is based on the **`throughput-performance`** profile and configures the following memory settings for a database server:

- Allocate 256 MiB of huge pages.
- Disable transparent huge pages.
- Configure the system to use swap space but use as little as possible. Ignore the fact that **`serverb`** has no swap space for this step.
- Start background syncing dirty pages when they consume 3% of the total memory.
- Start actively syncing dirty pages when they consume 15% of the total memory.
- Do not restrict the virtual memory a process can request. Allow programs that request massive amounts of memory but use very little to start on the server.

On **`serverb`**, configure the **`memload`** systemd service to:

- Not allow the **`memload`** process to consume more than 512 MiB of physical memory.
- Kill the **`memload`** process when it consumes more than 512 MiB of RAM.
- Have the kernel spread the memory of the **`memload`** process equally across all NUMA nodes.

Evaluation

As **`student`** on **`workstation`**, run **`lab compreview-memory grade`** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-memory grade
```

Finish

On **`workstation`**, run the **`lab compreview-memory finish`** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-memory finish
```

This concludes the comprehensive review.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Solution

Tuning for a Large Memory Implementation

In this review, you will create a custom tuned profile which sets various tunable values on a system running a database server.

Outcomes

You should be able to:

- Create a custom tuned profile that configures system parameters to tune memory for a database server.
- Restrict memory usage of a process.
- Configure a service to distribute memory usage across all NUMA nodes.

Before You Begin

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-memory start
```

Instructions

A pre-installed PostgreSQL database is running on **serverb**. On **serverb**, create a tuned profile named **enterprise-database** that is based on the **throughput-performance** profile and configures the following memory settings for a database server:

- Allocate 256 MiB of huge pages.
- Disable transparent huge pages.
- Configure the system to use swap space but use as little as possible. Ignore the fact that **serverb** has no swap space for this step.
- Start background syncing dirty pages when they consume 3% of the total memory.
- Start actively syncing dirty pages when they consume 15% of the total memory.
- Do not restrict the virtual memory a process can request. Allow programs that request massive amounts of memory but use very little to start on the server.

On **serverb**, configure the **memload** systemd service to:

- Not allow the **memload** process to consume more than 512 MiB of physical memory.
- Kill the **memload** process when it consumes more than 512 MiB of RAM.
- Have the kernel spread the memory of the **memload** process equally across all NUMA nodes.

1. The start script reboots **serverb**, you may have to wait for it to fully initialize.

   On **workstation**, use **ssh** to log in to **serverb** as **student**. Use **sudo -i** to switch to **root**.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
[student@workstation ~]$ ssh student@serverb
[student@serverb ~]$ sudo -i
[root@serverb ~]#
```

2. Create a tuned profile named **`enterprise-database`**.

2.1. Create the **`enterprise-database`** tuned profile directory.

```
[root@serverb ~]# mkdir /etc/tuned/enterprise-database
```

2.2. Create the **`/etc/tuned/enterprise-database/tuned.conf`** file with the following contents.

```
[main]
include = throughput-performance

[vm]
# Disabling transparent huge pages
transparent_hugepages = never

[sysctl]
# Allocating 256 MiB of huge pages (256 MiB / 2 MiB page = 128)
vm.nr_hugepages = 128

# Using swap as little as possible
vm.swappiness = 1

# Starting background writeback at 3%
vm.dirty_background_ratio = 3

# The process generating dirty data starts writeback at 15%
vm.dirty_ratio = 15
```

Do not activate the profile. Additional configuration is required.

3. A **`memload`** systemd service is installed on **`serverb`**. The service fails to start. The **`memload`** program requests a massive amount of memory but only really uses a few megabytes. Configure your **`enterprise-database`** tuned profile so that this program can start.

3.1. Retrieve the status of the **`memload`** the service.

```
[root@serverb ~]# systemctl status memload.service
* memload.service - Allocate memory
   Loaded: loaded (/etc/systemd/system/memload.service; enabled; vendor preset:
 disabled)
   Active: failed (Result: exit-code) since Fri 2019-07-19 08:04:24 EDT; 12s ago
  Process: 1284 ExecStart=/usr/local/bin/memload 40G (code=exited, status=1/
FAILURE)
 Main PID: 1284 (code=exited, status=1/FAILURE)

Jul 19 08:04:24 serverb... systemd[1]: Started Allocate memory.
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

```
Jul 19 08:04:24 serverb... memload[1284]: Allocating 40960 MiB of virtual memory
Jul 19 08:04:24 serverb... memload[1284]: malloc: Cannot allocate memory
...output omitted...
```

The process fails to allocate 40 GiB (40960 MiB) of virtual memory. The system does not have this amount of physical memory and refuses to overcommit.

3.2. Configure the system to always overcommit memory when it is requested. Edit the configuration file of your **enterprise-database** tuned profile, **/etc/tuned/enterprise-database/tuned.conf**

```
...output omitted...
# Starting background writeback at 3%
vm.dirty_background_ratio = 3

# The process generating dirty data starts writeback at 15%
vm.dirty_ratio = 15

# Always accept memory overcommit
vm.overcommit_memory = 1
```

Do not activate the profile yet or start **memload** because the service requires additional configuration.

4. Now that the system does not restrict the virtual memory a process can request, make sure that the **memload** process cannot consume more than 512 MiB of physical memory.

Update the **memload** systemd service configuration to kill the process when it consumes more than 512 MiB of RAM.

4.1. Create a drop-in file for the **memload** systemd service. Set the **MemoryLimit** parameter to 512 MiB.

```
[root@serverb ~]# systemctl set-property memload.service MemoryLimit=512M
```

As an alternative, you can create the drop-in directory manually:

```
[root@serverb ~]# mkdir /etc/systemd/system/memload.service.d
```

And create the **/etc/systemd/system/memload.service.d/00-limit.conf** drop-in file with the following content.

```
[Service]
MemoryLimit=512M
```

4.2. Instruct systemd to reload its configuration.

```
[root@serverb ~]# systemctl daemon-reload
```

Do not start the service yet.

5. Update the **memload** systemd service configuration again, so the kernel spreads the memory of the **memload** process equally across all your NUMA nodes. For your information, the **ExecStart** systemd service parameter gives the path to the program that the service executes.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

5.1. Install the *numactl* package.

```
[root@serverb ~]# yum install numactl
...output omitted...
Is this ok [y/N]: y
...output omitted...
Complete!
```

5.2. Edit the **memload** systemd service to prefix the command by **/bin/numactl --interleave all**. The service configuration is located in the **/etc/systemd/system/memload.service** file.

```
[Unit]
Description=Allocate memory
After=network.target

[Service]
ExecStart=/bin/numactl --interleave all -- /usr/local/bin/memload 40G

[Install]
WantedBy=multi-user.target
```

You must provide the full path to the **numactl** command (**/bin/numactl**).

5.3. Instruct systemd to reload its configuration.

```
[root@serverb ~]# systemctl daemon-reload
```

6. Enable your new **enterprise-database** tuned profile and restart the **postgresql** and the **memload** services. To verify your work, check that the system uses some of the allocated huge pages. Verify that the kernel spreads the memory that **memload** uses across all the NUMA nodes.

6.1. Enable the **enterprise-database** tuned profile.

```
[root@serverb ~]# tuned-adm profile enterprise-database
```

6.2. Restart the **postgresql** service.

```
[root@serverb ~]# systemctl restart postgresql
```

6.3. Start the **memload** service and verify that the **memload** process is running.

```
[root@serverb ~]# systemctl start memload
[root@serverb ~]# systemctl status memload
* memload.service - Allocate memory
   Loaded: loaded (/etc/systemd/system/memload.service; enabled; vendor preset:
 disabled)
  Drop-In: /etc/systemd/system.control/memload.service.d
           └─50-MemoryLimit.conf
   Active: active (running) since Fri 2019-07-19 08:34:21 EDT; 4s ago
 Main PID: 1572 (memload)
```

```
     Tasks: 1 (limit: 11399)
    Memory: 10.2M (max: 512.0M)
    CGroup: /system.slice/memload.service
            └─1572 /usr/local/bin/memload 40G

Jul 19 08:34:21 serverb... systemd[1]: Started Allocate memory.
Jul 19 08:34:21 serverb... numactl[1572]: Allocating 40960 MiB of virtual memory
Jul 19 08:34:21 serverb... numactl[1572]: Memory allocation done
Jul 19 08:34:21 serverb... numactl[1572]: Using 10 MiB of memory
Jul 19 08:34:21 serverb... numactl[1572]: Using memory done
```

6.4. Check that the system uses some of the huge pages.

```
[root@serverb ~]# grep Huge /proc/meminfo
AnonHugePages:     12288 kB
ShmemHugePages:        0 kB
HugePages_Total:     128
HugePages_Free:      121
HugePages_Rsvd:       64
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:          262144 kB
```

Out of the 128 available pages, the system uses seven pages (128 - 121 = 7). The number of used pages may be different on your system.

6.5. Verify that the kernel has spread the memory that **memload** is using between all the NUMA nodes.

```
[root@serverb ~]# numastat -c memload

Per-node process memory usage (in MBs) for PID 1572 (memload)
         Node 0 Node 1 Node 2 Node 3 Total
         ------ ------ ------ ------ -----
Huge          0      0      0      0     0
Heap          0      0      0      0     0
Stack         0      0      0      0     0
Private       3      3      3      3    11
-------  ------ ------ ------ ------ -----
Total         3      3      3      3    11
```

6.6. Log off from **serverb** when done.

```
[root@serverb ~]# exit
[student@serverb ~]$ exit
[student@workstation ~]$
```

Evaluation

As **student** on **workstation**, run **lab compreview-memory grade** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-memory grade
```

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

Finish

On **`workstation`**, run the **`lab compreview-memory finish`** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-memory finish
```

This concludes the comprehensive review.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.

For use by Jagadish Honnappa jagadish.h@hcl.com jagadish.h@hcl.com Copyright © 2023 Red Hat, Inc.