Red Hat Enterprise Linux 9

监控和管理系统状态和性能

优化系统吞吐量、延迟和电源消耗

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 监控和管理系统状态和性能

优化系统吞吐量、延迟和电源消耗

法律通告

摘要

在不同情况下，监控和优化 Red Hat Enterprise Linux 9 上的吞吐量、延迟和功耗。

目录

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 点顶部导航栏中的 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您的改进建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 TUNED 入门

作为系统管理员，您可以使用 TuneD 应用程序来针对各种用例优化系统的性能配置集。

1.1. TUNED 的目的

TuneD 是监控您的系统并优化特定工作负载性能的服务。TuneD 的核心是 *配置集 (profiles)*，它针对不同的用例调优您的系统。

TuneD 通过很多预定义的配置集发布，它们适用于以下用例：

- 高吞吐量
- 低延迟
- 保存电源

可以修改为每个配置集定义的规则，并自定义如何调整特定设备。当您切换到另一个配置集或取消激活 TuneD 时，对之前的配置集进行的所有更改都会恢复到其原始状态。

您还可以将 TuneD 配置为响应设备使用的变化，并调整设置以提高活跃设备的性能并减少不活跃设备的功耗。

1.2. 调优配置集

系统的详细分析可能会非常耗时。TuneD 为典型的用例提供了很多预定义的配置集。您还可以创建、修改和删除配置集。

TuneD 提供的配置集被分为以下几个类别：

- 节能配置集
- 性能提升配置集

性能提升配置集包括侧重于以下方面的配置集：

- 存储和网络的低延迟
- 存储和网络的高吞吐量
- 虚拟机性能
- 虚拟化主机性能

配置集配置的语法
tuned.conf 文件可以包含一个 **[main]** 部分，其他部分用于配置插件实例。但是，所有部分都是可选的。

以 hash 符号 (**#**) 开头的行是注释。

其他资源

- **tuned.conf (5)** 手册页.

1.3. 默认 TUNED 配置集

在安装过程中，将自动选择您的系统的最佳配置集。目前，会根据以下自定义规则选择默认配置集：

环境	默认配置集	目标
Compute 节点	**throughput-performance**	最佳吞吐量性能
虚拟机	**virtual-guest**	最佳的性能。如果实现最佳性能并不是您最需要考虑的，可以将其改为 **balance** 或 **powersave** 配置集。
其他情况	**balanced**	平衡性能和能源消耗

其他资源

- **tuned.conf (5)** 手册页.

1.4. 合并的 TUNED 配置集

作为实验性功能，可以一次性选择更多配置集。**tuned** 将尝试在负载期间合并它们。

如果存在冲突，则最后指定的配置集的设置会优先使用。

> **例 1.1. 虚拟客户端中低功耗**
>
> 以下示例优化了在虚拟机中运行的系统，以获得最佳性能，并同时将其调优以实现低功耗，低功耗比高性能有更高优先级：
>
> ```
> # tuned-adm profile virtual-guest powersave
> ```

警告

合并会在不检查生成的参数组合是否有意义的情况下自动进行。因此，该功能可能会以相反的方式调整一些参数，这么做可能会影响生产效率。例如，使用 **throughput-performance** 配置集针对高吞吐量设置磁盘，但当前通过 **spindown-disk** 配置集将磁盘旋转设置为低值。

其他资源

***tuned-adm** man page。* **tuned.conf (5)** man page。

1.5. TUNED 配置集的位置

TuneD 配置集存储在以下目录中：

/usr/lib/tuned/

特定于分发的配置文件存储在目录中。每个配置集都有自己的目录。该配置集由名为 **tuned.conf** 的主配置文件以及其他文件（如帮助程序脚本）组成。

/etc/tuned/

如果您需要自定义配置集，请将配置集目录复制到用于自定义配置集的目录中。如果同一名称有两个配置集，则使用位于 **/etc/tuned/** 中的自定义配置集。

其他资源

- **tuned.conf (5)** 手册页.

1.6. RHEL 提供的调优配置集

以下是在 Red Hat Enterprise Linux 中安装 **TuneD** 的配置集列表。

注意

更特定产品的或第三方的 **TuneD** 配置集也可能会存在。这些配置集通常由单独的 RPM 软件包提供。

balanced

默认的节能配置文件。它在性能和功耗之间具有折衷。在可能的情况下尽可能使用自动扩展和自动调整。唯一缺陷是增加延迟。在当前的 **TuneD** 版本中，它启用了 CPU、磁盘、音频和视频插件，并激活了 **conservative** CPU 调控器。如果支持，**radeon_powersave** 选项使用 **dpm-balanced** 值，否则被设置为 **auto**。
它将 **energy_performance_preference** 属性改为 **normal** 能源设置。它还将 **scaling_governor** 策略属性改为 **conservative** 或 **powersave** CPU 调控器。

powersave

用于最大节能性能的配置集。它可以对性能进行调整，从而最大程度降低实际功耗。在当前的 **TuneD** 发行版本中，它为 SATA 主机适配器启用 USB 自动挂起、WiFi 节能和 Aggresive Link Power Management (ALPM) 节能。它还为使用低折率的系统调度多核功耗，并激活 **ondemand** 监管器。它启用了 AC97 音频节能，或根据您的系统，HDA-Intel 节能时间为 10 秒。如果您的系统包含启用了 KMS 支持的 Radeon 图形卡，配置集会将其配置为自动节能。在 ASUS Eee PC 上，启用了动态超级混合引擎。
它将 **energy_performance_preference** 属性改为 **powersave** 或 **power** energy 设置。它还会将 **scaling_governor** 策略属性更改为 **ondemand** 或 **powersave** CPU 调控器。

注意

在某些情况下，与 **powersave** 配置集相比，**balanced** 配置集效率更高。

请考虑存在定义的需要完成的工作，例如一个需要转码的视频文件。如果转码以全功率完成，则您的机器可能会消耗较少的能源，因为任务快速完成，因此计算机可以启动空闲，且自动缩减到非常有效的节能模式。另一方面，如果您把文件转码为节流的机器，则计算机在转码期间会消耗较少的电源，但进程会花费更长时间，且总体消耗的能源可能会更高。

这就是为什么 **balanced** 配置文件通常是一个更好的选择。

throughput-performance

针对高吞吐量优化的服务器配置文件。它禁用节能机制并启用 **sysctl** 设置，以提高磁盘和网络 IO 的吞吐量性能。CPU 调控器设置为 **performance**。
它将 **energy_performance_preference** 和 **scaling_governor** 属性设置为 **performance** 配置集。

accelerator-performance

accelerator-performance 配置集包含与 **throughput-performance** 配置集相同的调整。另外，它会将 CPU 锁定为低 C 状态，以便使延迟小于 100us。这提高了某些加速器的性能，如 GPU。

latency-performance

为低延迟优化的服务器配置文件。它禁用节能机制并启用 **sysctl** 设置来缩短延迟。CPU 调控器被设置为 **performance**，CPU 被锁定到低 C 状态（按 PM QoS）。
它将 **energy_performance_preference** 和 **scaling_governor** 属性设置为 **performance** 配置集。

network-latency

低延迟网络调整的配置集。它基于 **latency-performance** 配置集。它还禁用透明大内存页和 NUMA 平衡，并调整其他一些与网络相关的 **sysctl** 参数。
它继承 **latency-performance** 配置集，该配置集将 **power_performance_preference** 和 **scaling_governor** 属性更改为 **performance** 配置集。

hpc-compute

针对高性能计算而优化的配置集。它基于 **latency-performance** 配置集。

network-throughput

用于吞吐量网络调优的配置集。它基于 **throughput-performance** 配置集。此外，它还增加了内核网络缓冲区。
它继承 **latency-performance** 或 **throughput-performance** 配置集，并将 **energy_performance_preference** 和 **scaling_governor** 属性改为 **performance** 配置集。

virtual-guest

为 Red Hat Enterprise Linux 9 虚拟机和 VMWare 虚拟机设计的配置集基于 **throughput-performance** 配置集（除其他任务）减少了虚拟内存的交换性并增加磁盘预读值。它不会禁用磁盘障碍。
它继承 **throughput-performance** 配置集，该配置集将 **energy_performance_preference** 和 **scaling_governor** 属性更改为 **performance** 配置集。

virtual-host

基于 **throughput-performance** 配置集（除其他任务）为虚拟主机设计的配置集降低了虚拟内存交换，增加磁盘预读值，并启用更主动的脏页面回写值。
它继承 **throughput-performance** 配置集，该配置集将 **energy_performance_preference** 和 **scaling_governor** 属性更改为 **performance** 配置集。

oracle

根据 **throughput-performance** 配置集，为 Oracle 数据库负载进行了优化。它还禁用透明大内存页，并修改其他与性能相关的内核参数。这个配置集由 **tuned-profiles-oracle** 软件包提供。

desktop

根据 **balanced** 配置文件，为桌面进行了优化的配置集。此外，它还启用了调度程序自动组以更好地响应交互式应用程序。

optimize-serial-console

通过减少 printk 值，将 I/O 活动微调到串行控制台的配置集。这应该使串行控制台更快响应。此配置集用作其他配置集的覆盖。例如：

```
# tuned-adm profile throughput-performance optimize-serial-console
```

mssql

为 Microsoft SQL Server 提供的配置集。它基于 **throughput-performance** 配置集。

intel-sst

为带有用户定义的 Intel Speed Select Technology 配置的系统进行优化的配置集。此配置集用作其他配置集的覆盖。例如：

```
# tuned-adm profile cpu-partitioning intel-sst
```

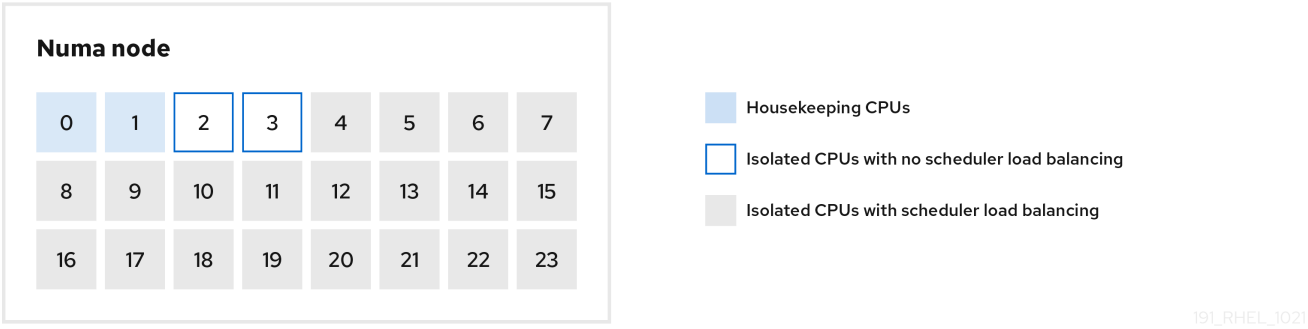
1.7. TUNED CPU-PARTITIONING 配置集

要为对延迟敏感的工作负载调整 Red Hat Enterprise Linux 9，红帽建议使用 **cpu-partitioning** TuneD 配置集。

在 Red Hat Enterprise Linux 9 之前，低延迟 Red Hat 文档描述了实现低延迟调整所需的大量低级别步骤。在 Red Hat Enterprise Linux 9 中，您可以使用 **cpu-partitioning** TuneD 配置集更有效地执行低延迟性能优化。根据个人低延迟应用程序的要求，此配置集可轻松自定义。

下图显示了如何使用 **cpu-partitioning** 配置集。这个示例使用 CPU 和节点布局。

图 1.1. cpu-partitioning 图



您可以使用以下配置选项在 **/etc/tuned/cpu-partitioning-variables.conf** 文件中配置 cpu-partitioning 配置集：

带有负载均衡的隔离 CPU

在 cpu-partitioning 图中，从 4 到 23 编号的块是默认的隔离 CPU。在这些 CPU 上启用了内核调度程序的进程负载均衡。它专为需要内核调度程序负载平衡的多个线程的低延迟进程而设计。
您可以使用 **isolated_cores=cpu-list** 选项在 **/etc/tuned/cpu-partitioning-variables.conf** 文件中配置 cpu-partitioning 配置集，它列出了 CPU 来隔离将使用内核调度程序负载平衡。

隔离的 CPU 列表用逗号分开，也可以使用一个短划线（如 **3-5** ）指定范围。这个选项是必须的。这个列表中缺少的任何 CPU 会自动被视为内务 CPU。

没有负载均衡的隔离 CPU

在 cpu-partitioning 图中，编号为 2 和 3 的块是不提供任何其他内核调度程序进程负载均衡的隔离 CPU。
您可以使用 **no_balance_cores=cpu-list** 选项在 **/etc/tuned/cpu-partitioning-variables.conf** 文件中配置 cpu-partitioning 配置集，它列出了不使用内核调度程序负载平衡的 CPU。

指定 **no_balance_cores** 选项是可选的，但此列表中的任何 CPU 都必须是 **isolated_cores** 列表中所列 CPU 的子集。

使用这些 CPU 的应用程序线程需要单独固定到每个 CPU。

日常 CPU

在 **cpu-partitioning-variables.conf** 文件中没有隔离的 CPU 会自动被视为内务 CPU。在内务 CPU 上，允许执行所有服务、守护进程、用户进程、可移动内核线程、中断处理程序和内核计时器。

其他资源

- **tuned-profiles-cpu-partitioning (7)** man page

1.8. 使用 TUNED CPU-PARTITIONING 配置集进行低延迟调整

这个步骤描述了如何使用 TuneD 的 **cpu-partitioning** 配置集为低延迟调整系统。它使用了低延迟应用的示例，它可以使用 **cpu-partitioning** 和 CPU 布局，如 cpu-partitioning 图中所述。

本例中的应用程序使用了：

- 从网络读取数据的专用的 reader 线程将固定到 CPU 2。
- 处理此网络数据的大量线程将固定到 CPU 4-23。
- 将处理的数据写入网络的专用写入器线程将固定到 CPU 3。

先决条件

- 您已以 root 用户身份，使用 **dnf install tuned-profiles-cpu-partitioning** 命令安装 **cpu-partitioning** TuneD 配置集。

步骤

1. 编辑 **/etc/tuned/cpu-partitioning-variables.conf** 文件并添加以下信息：

```
# All isolated CPUs:
isolated_cores=2-23
# Isolated CPUs without the kernel's scheduler load balancing:
no_balance_cores=2,3
```

2. 设置 **cpu-partitioning** TuneD 配置集：

```
# tuned-adm profile cpu-partitioning
```

3. 重启
   重新引导后，将根据 cpu-partitioning 图中的隔离，为低延迟调优。该应用可以使用 taskset 将读取器和写入器线程固定到 CPU 2 和 3，以及 CPU 4-23 上剩余的应用程序线程。

其他资源

- **tuned-profiles-cpu-partitioning (7)** man page

1.9. 自定义 CPU-PARTITIONING TUNED 配置集

您可以扩展 TuneD 配置集，以进行额外的性能优化更改。

例如，**cpu-partitioning** 配置集将 CPU 设置为使用 **cstate=1**。要使用 **cpu-partitioning** 配置集，但额外将 CPU cstate 从 cstate1 更改为 cstate0，以下流程描述了一个新的 TuneD 配置集，名称为 *my_profile*，它继承 **cpu-partitioning** 配置集，然后设置 C state-0。

步骤

1. 创建 **/etc/tuned/my_profile** 目录：

```
# mkdir /etc/tuned/my_profile
```

2. 在此目录中创建 **tuned.conf** 文件并添加以下内容：

```
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. 使用新配置集：

```
# tuned-adm profile my_profile
```

注意

在共享示例中，不需要重新启动。但是，如果 *my_profile* 配置集中的更改需要重新引导才能生效，则重新启动计算机。

其他资源

- **tuned-profiles-cpu-partitioning (7)** man page

1.10. RHEL 提供的实时 TUNED 配置集

实时配置集适用于运行实时内核的系统。如果没有特殊的内核构建，则不会将系统配置为实时。在 RHEL 上，配置集可从额外的软件仓库获得。

可用的实时配置集如下：

realtime

在裸机实时系统上使用。
由 **tuned-profiles-realtime** 软件包提供，该软件包可从 RT 或 NFV 存储库中获得。

realtime-virtual-host

在为实时配置的虚拟化主机中使用。
由 **tuned-profiles-nfv-host** 软件包提供，该软件包可通过 NFV 存储库获取。

realtime-virtual-guest

在为实时配置的虚拟化客户端中使用。
由 **tuned-profiles-nfv-guest** 软件包提供，该软件包可通过 NFV 存储库获取。

1.11. TUNED 中的静态和动态性能优化

在决定对于给定情况或目的使用哪种方法时，了解应用 **TuneD** 的两种系统调优（*static*和*dynamic*）之间的区别非常重要。

静态调整

主要由预定义的 **sysctl** 和 **sysfs** 设置的应用程序组成，以及激活多个配置工具（如 **ethtool** ）的一次性激活。

动态调整

监视如何在系统正常运行时间期间使用各种系统组件。**tuned** 根据监控信息动态调整系统设置。例如，硬盘驱动器在启动和登录期间大量使用，但当用户主要可能与 Web 浏览器或电子邮件客户端等应用程序工作时，通常使用。同样，CPU 和网络设备在不同时间上有所不同。**TuneD** 监控这些组件的活动，并对使用中的更改做出反应。

默认情况下禁用动态性能优化。要启用它，请编辑 **/etc/tuned/tuned-main.conf** 文件并将 **dynamic_tuning** 选项改为 **1**。然后 **TuneD** 会定期分析系统统计信息，并使用它们更新您的系统调优设置。要在这些更新之间配置时间间隔（以秒为单位），请使用 **update_interval** 选项。

目前实施了动态调优算法，尝试平衡性能和节能，因此在性能配置集中禁用。可以在 **TuneD** 配置集中启用或禁用各个插件的动态性能优化。

> **例 1.2. 工作站上的静态和动态调优**
>
> 在典型的办公室工作站上，以太网网络接口在大多数时间都不活跃。通常只会发送和接收一些电子邮件，或载入一些网页。
>
> 对于这些负载，网络接口不必像默认情况那样始终全速运行。**TuneD** 为网络设备有一个监控和调优插件，可检测此低活动，然后自动降低该接口的速度，通常会实现较低的功耗。
>
> 如果在较长的时间内接口上的活动增加，例如：因为下载了 DVD 镜像或打开了带有大量附加的电子邮件，则 **TuneD** 会检测到这个信息，并设置接口速度的最大速度，以便在活动级别高时提供最佳性能。
>
> 这个原则还用于 CPU 和磁盘的其他插件。

1.12. TUNED NO-DAEMON（非守护进程）模式

您可以在 **no-daemon** 模式下运行 **TuneD**，它不需要任何常驻内存。在这个模式中，**TuneD** 应用设置并退出。

默认情况下，**no-daemon** 模式被禁用，因为在这个模式中缺少大量 **TuneD** 功能，包括：

- D-Bus 支持
- 热插支持
- 对设置进行回滚支持

要启用 **no-daemon** 模式，请在 **/etc/tuned/tuned-main.conf** 文件中包含以下行：

```
daemon = 0
```

1.13. 安装并启用 TUNED

此流程安装并启用 **TuneD** 应用程序，安装 **TuneD** 配置集，并为您的系统预设默认 **TuneD** 配置集。

流程

1. 安装 **Tuned** 软件包：

```
# dnf install tuned
```

2. 启用并启动 **TuneD** 服务：

```
# systemctl enable --now tuned
```

3. 另外，还可为实时系统安装 **TuneD** 配置集：
对于实时系统的 **TuneD** 配置文件，启用 **rhel-9** 存储库。

```
# subscription-manager repos --enable=rhel-9-for-x86_64-nfv-beta-rpms
```

安装它。

```
# dnf install tuned-profiles-realtime tuned-profiles-nfv
```

4. 验证 **TuneD** 配置集是否活跃并应用：

```
$ tuned-adm active

Current active profile: throughput-performance
```

注意

活跃的配置文件 TuneD 会根据您的机器类型和系统设置会自动进行不同的预置。

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

1.14. 列出可用的 TUNED 配置集

此流程列出了系统中当前可用的所有 **TuneD** 配置集。

步骤

- 要列出系统中的所有可用 **TuneD** 配置集，请使用：

```
$ tuned-adm list

Available profiles:
- accelerator-performance - Throughput performance based tuning with disabled higher
latency STOP states
```

```
- 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 tuning 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: balanced
```

- 要只显示当前活跃的配置集，请使用：

```
$ tuned-adm active

Current active profile: throughput-performance
```

其他资源

- **tuned-adm (8)** 手册页.

1.15. 设置 TUNED 配置集

此流程激活系统中的所选 **TuneD** 配置集。

先决条件

- **TuneD** 服务正在运行。详情请参阅安装和启用 TuneD。

步骤

1. 另外，您可以让 **TuneD** 为您的系统推荐最合适的配置集：

```
# tuned-adm recommend

throughput-performance
```

2. 激活配置集：

```
# tuned-adm profile selected-profile
```

另外，您可以激活多个配置集的组合：

```
# tuned-adm profile selected-profile1 selected-profile2
```

例 1.3. 为低功耗优化的虚拟机

以下示例优化了在虚拟机中运行的系统，以获得最佳性能，并同时将其调优以实现低功耗，低功耗比高性能有更高优先级：

```
# tuned-adm profile virtual-guest powersave
```

3. 查看系统中当前活跃的 **TuneD** 配置集：

```
# tuned-adm active

Current active profile: selected-profile
```

4. 重启系统：

```
# reboot
```

验证步骤

- 验证 **TuneD** 配置集是否活跃并应用：

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

其他资源

- **tuned-adm (8)** 手册页

1.16. 使用 TUNED D-BUS 接口

您可以通过 TuneD D-Bus 接口在运行时直接与 TuneD 进行通信，以控制各种 TuneD 服务。

您可以使用 **busctl** 或 **dbus-send** 命令访问 D-Bus API。

注意

虽然您可以使用 **busctl** 或 **dbus-send** 命令，但 **busctl** 命令是 **systemd** 的一部分，因此已在大多数主机上存在。

1.16.1. 使用 TuneD D-Bus 接口来显示可用的 TuneD D-Bus API 方法

您可以使用 TuneD D-Bus 接口查看可与 TuneD 一起使用的的 D-Bus API 方法。

先决条件

- TuneD 服务正在运行。详情请参阅安装和启用 TuneD。

步骤

- 要查看可用的 TuneD API 方法，请运行：

```
$ busctl introspect com.redhat.tuned /Tuned com.redhat.tuned.control
```

输出应类似于以下内容：

```
NAME                    TYPE    SIGNATURE RESULT/VALUE FLAGS
.active_profile         method  -         s            -
.auto_profile           method  -         (bs)         -
.disable                method  -         b            -
.get_all_plugins        method  -         a{sa{ss}}    -
.get_plugin_documentation method s         s            -
.get_plugin_hints       method  s         a{ss}        -
.instance_acquire_devices method ss       (bs)         -
.is_running             method  -         b            -
.log_capture_finish     method  s         s            -
.log_capture_start      method  ii        s            -
.post_loaded_profile    method  -         s            -
.profile_info           method  s         (bsss)       -
.profile_mode           method  -         (ss)         -
.profiles               method  -         as           -
.profiles2              method  -         a(ss)        -
.recommend_profile      method  -         s            -
.register_socket_signal_path method s     b            -
.reload                 method  -         b            -
.start                  method  -         b            -
.stop                   method  -         b            -
.switch_profile         method  s         (bs)         -
.verify_profile         method  -         b            -
.verify_profile_ignore_missing method -   b            -
.profile_changed        signal  sbs       -            -
```

您可以在 TuneD 上游存储库 中找到不同的可用方法的描述。

1.16.2. 使用 TuneD D-Bus 接口来更改活跃的 TuneD 配置文件

您可以使用 TuneD D-Bus 接口，将活跃的 TuneD 配置文件替换为所需的 TuneD 配置文件。

先决条件

- TuneD 服务正在运行。详情请参阅安装和启用 TuneD。

步骤

- 要更改活跃的 TuneD 配置文件，请运行：

```
$ busctl call com.redhat.tuned /Tuned com.redhat.tuned.control switch_profile s profile
(bs) true "OK"
```

使用所需的配置文件的名称替换 *profile*。

验证

- 要查看当前活跃的 TuneD 配置文件，请运行：

```
$ busctl call com.redhat.tuned /Tuned com.redhat.tuned.control active_profile
s "profile"
```

1.17. 禁用 TUNED

此流程禁用 TuneD，并将所有受影响的系统设置重置为其原始状态，然后再修改 TuneD。

步骤

- 临时禁用所有调整：

```
# tuned-adm off
```

  调整会在 **TuneD** 服务重启后再次应用。

- 或者，要永久停止并禁用 **TuneD** 服务：

```
# systemctl disable --now tuned
```

其他资源

- **tuned-adm (8)** 手册页

第 2 章 自定义 TUNED 配置集

您可以创建或修改 TuneD 配置集来优化预期的用例的系统性能。

先决条件

- 安装并启用 TuneD，如安装和启用 TuneD 所述。

2.1. 调优配置集

系统的详细分析可能会非常耗时。TuneD 为典型的用例提供了很多预定义的配置集。您还可以创建、修改和删除配置集。

TuneD 提供的配置集被分为以下几个类别：

- 节能配置集
- 性能提升配置集

性能提升配置集包括侧重于以下方面的配置集：

- 存储和网络的低延迟
- 存储和网络的高吞吐量
- 虚拟机性能
- 虚拟化主机性能

配置集配置的语法
tuned.conf 文件可以包含一个 **[main]** 部分，其他部分用于配置插件实例。但是，所有部分都是可选的。

以 hash 符号 (**#**) 开头的行是注释。

其他资源

- **tuned.conf (5)** 手册页.

2.2. 默认 TUNED 配置集

在安装过程中，将自动选择您的系统的最佳配置集。目前，会根据以下自定义规则选择默认配置集：

环境	默认配置集	目标
Compute 节点	**throughput-performance**	最佳吞吐量性能
虚拟机	**virtual-guest**	最佳的性能。如果实现最佳性能并不是您最需要考虑的，可以将其改为 **balance** 或 **powersave** 配置集。
其他情况	**balanced**	平衡性能和能源消耗

其他资源

其他资源

- **tuned.conf (5)** 手册页.

2.3. 合并的 TUNED 配置集

作为实验性功能，可以一次性选择更多配置集。**tuned** 将尝试在负载期间合并它们。

如果存在冲突，则最后指定的配置集的设置会优先使用。

> **例 2.1. 虚拟客户端中低功耗**
>
> 以下示例优化了在虚拟机中运行的系统，以获得最佳性能，并同时将其调优以实现低功耗，低功耗比高性能有更高优先级：
>
> ```
> # tuned-adm profile virtual-guest powersave
> ```

警告

合并会在不检查生成的参数组合是否有意义的情况下自动进行。因此，该功能可能会以相反的方式调整一些参数，这么做可能会影响生产效率。例如，使用 **throughput-performance** 配置集针对高吞吐量设置磁盘，但当前通过 **spindown-disk** 配置集将磁盘旋转设置为低值。

其他资源

__tuned-adm__ man page。 **tuned.conf (5)** man page。

2.4. TUNED 配置集的位置

TuneD 配置集存储在以下目录中：

/usr/lib/tuned/

特定于分发的配置文件存储在目录中。每个配置集都有自己的目录。该配置集由名为 **tuned.conf** 的主配置文件以及其他文件（如帮助程序脚本）组成。

/etc/tuned/

如果您需要自定义配置集，请将配置集目录复制到用于自定义配置集的目录中。如果同一名称有两个配置集，则使用位于 **/etc/tuned/** 中的自定义配置集。

其他资源

- **tuned.conf (5)** 手册页.

2.5. TUNED 配置集之间的继承

TuneD 配置集可以基于其他配置集，仅修改其父级配置集的某些方面。

TuneD 配置集的 **[main]** 部分可以识别 **include** 选项：

```
[main]
include=parent
```

父配置集中的所有设置都会加载到此 子配置集中。在以下小节中，*child* 配置集可以覆盖从 *parent* 配置集继承的特定设置，或者添加 *parent* 配置集中没有的新设置。

您可以基于 **/usr/lib/tuned/** 中预安装的配置集，在 **/etc/tuned/** 目录中创建自己的 *child* 配置集并调整了一些参数。

如果对 *parent* 配置集（如 TuneD 升级后）进行了更新，则更改会反映在 *child* 配置集中。

例 2.2. 基于均衡的节能配置集

以下是一个可扩展 **balanced** 配置集的自定义配置集，并将所有设备的主动链路电源管理 (ALPM) 设置为最大节能项。

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

其他资源

- **tuned.conf (5)** 手册页

2.6. TUNED 中的静态和动态性能优化

在决定对于给定情况或目的使用哪种方法时，了解应用 TuneD 的两种系统调优（*static*和*dynamic*）之间的区别非常重要。

静态调整

主要由预定义的 **sysctl** 和 **sysfs** 设置的应用程序组成，以及激活多个配置工具（如 **ethtool** ）的一次性激活。

动态调整

监视如何在系统正常运行时间期间使用各种系统组件。**tuned** 根据监控信息动态调整系统设置。例如，硬盘驱动器在启动和登录期间大量使用，但当用户主要可能与 Web 浏览器或电子邮件客户端等应用程序工作时，通常使用。同样，CPU 和网络设备在不同时间上有所不同。TuneD 监控这些组件的活动，并对使用中的更改做出反应。

默认情况下禁用动态性能优化。要启用它，请编辑 **/etc/tuned/tuned-main.conf** 文件并将 **dynamic_tuning** 选项改为 **1**。然后 TuneD 会定期分析系统统计信息，并使用它们更新您的系统调优设置。要在这些更新之间配置时间间隔（以秒为单位），请使用 **update_interval** 选项。

目前实施了动态调优算法，尝试平衡性能和节能，因此在性能配置集中禁用。可以在 TuneD 配置集中启用或禁用各个插件的动态性能优化。

例 2.3. 工作站上的静态和动态调优

在典型的办公室工作站上，以太网网络接口在大多数时间都不活跃。通常只会发送和接收一些电子邮件，或载入一些网页。

对于这些负载，网络接口不必像默认情况那样始终全速运行。**TuneD** 为网络设备有一个监控和调优插件，可检测此低活动，然后自动降低该接口的速度，通常会实现较低的功耗。

如果在较长的时间内接口上的活动增加，例如：因为下载了 DVD 镜像或打开了带有大量附加的电子邮件，则 **TuneD** 会检测到这个信息，并设置接口速度的最大速度，以便在活动级别高时提供最佳性能。

这个原则还用于 CPU 和磁盘的其他插件。

2.7. TUNED 插件

插件是 **TuneD** 配置集中的模块，**TuneD** 使用它们监控或优化系统上的不同设备。

TuneD 使用两种类型的插件：

监控插件

监控插件用于从正在运行的系统中获取信息。通过调优插件进行动态调优，可以使用监控插件的输出。
当任何已启用的调优插件需要指标时，监控插件会自动实例化。如果两个调优插件需要相同的数据，则只创建一个监控插件的实例，并且数据会被共享。

调优插件

每个调优插件对单个子系统进行调优，并会获取从 TuneD 配置集填充的多个参数。每个子系统可以有多个设备，如多个 CPU 或网卡，这些设备由调优插件的单个实例处理。还支持单个设备的具体设置。

TuneD 配置集中的插件语法
描述插件实例的部分采用以下格式：

```
[NAME]
type=TYPE
devices=DEVICES
```

NAME

是插件实例的名称，在日志中使用。它可以是一个任意字符串。

TYPE

是调优插件的类型。

DEVICES

是此插件实例处理的设备列表。
devices 行可以包含一个列表、通配符 (*) 和负效果 (**!**) 。如果没有 **devices** 行，则插件实例处理所有在 *TYPE* 系统中附加的所有设备。这与使用 **devices=*** 选项相同。

例 2.4. 使用插件匹配块设备

以下示例与以 **sd** 开头的所有块设备（如 **sda** 或 **sdb** ）匹配，且不禁用这些块设备：

```
[data_disk]
type=disk
devices=sd*
disable_barriers=false
```

以下示例与 **sda1** 和 **sda2** 以外的所有块设备匹配：

```
[data_disk]
type=disk
devices=!sda1, !sda2
disable_barriers=false
```

如果没有指定插件的实例，则不会启用插件。

如果插件支持更多选项，也可以在插件部分中指定它们。如果没有指定选项，且之前未在 included 插件中指定，则使用默认值。

简短插件语法

如果您的插件实例不需要使用自定义名称，且配置文件中只有一个定义，则 **TuneD** 支持以下短语法：

```
[TYPE]
devices=DEVICES
```

在这种情况下，可以省略 **type** 行。然后，实例使用名称来指代，与类型相同。然后，前面的示例可重写为：

例 2.5. 使用简短语法匹配块设备

```
[disk]
devices=sdb*
disable_barriers=false
```

配置集中的冲突插件定义

如果使用 **include** 选项指定同一部分，则会合并设置。如果因为冲突而无法合并它们，则最后冲突的定义会覆盖上一个设置。如果您不知道之前定义的内容，您可以使用 **replace** 布尔值选项并将其设置为 **true**。这会导致之前带有相同名称的定义被覆盖，且不会出现合并。

您还可以通过指定 **enabled=false** 选项来禁用插件。这与实例从未定义的影响相同。如果您从 **include** 选项重新定义之前定义，且不想在自定义配置集中激活插件，则禁用插件会很有用。

注意

TuneD 包含了作为启用或禁用调优配置文件的一部分来运行任何 shell 命令的功能。这可让您使用尚未集成到 **TuneD** 的功能扩展 TuneD 配置集。
您可以使用 **script** 插件指定任意 shell 命令。

其他资源

- **tuned.conf (5)** 手册页

2.8. 可用的 TUNED 插件

监控插件

目前，实施了以下监控插件：

disk

每个设备获取磁盘负载（IO 操作数）和测量间隔。

net

每个网卡获取网络负载（传输数据包的数量）和测量间隔。

load

获取每个 CPU 的 CPU 负载和测量间隔。

调优**插件**
目前，实施了以下调优插件。只有其中一些插件实施动态性能优化。列出插件支持的选项：

cpu

将 CPU 调控器设置为 **governor** 选项指定的值，并根据 CPU 负载动态更改电源管理服务质量 (PM QoS) CPU Direct Memory Access (DMA) 延迟。
如果 CPU 负载低于 **load_threshold** 选项指定的值，则延迟设置为由 **latency_high** 选项指定的值，否则它将设置为 **latency_low** 指定的值。

您还可以强制对特定值强制延迟并阻止它动态更改。要做到这一点，将 **force_latency** 选项设置为所需的延迟值。

eeepc_she

根据 CPU 负载动态设置前端总线 (FSB) 速度。
此功能可在一些笔记本电脑中找到，也称为 ASUS Super Hybrid Engine (SHE) 。

如果 CPU 负载较低或等于 **load_threshold_powersave** 选项指定的值，则插件会将 FSB 速度设置为 **she_powersave** 选项指定的值。如果 CPU 负载较高或等于 **load_threshold_normal** 选项指定的值，它会将 FSB 速度设置为 **she_normal** 选项指定的值。

不支持静态调优，如果 **TuneD** 不检测到对这个功能的硬件支持，则插件会被透明禁用。

net

将 Wake-on-LAN 功能配置为 **wake_on_lan** 选项指定的值。它使用与 **ethtool** 实用程序相同的语法。它还会根据接口利用率动态更改接口速度。

sysctl

设置由插件选项指定的各种 **sysctl** 设置。
语法为 ***name=value***，其中 *name* 与 **sysctl** 实用程序提供的名称相同。

如果您需要更改 **TuneD** 中其他插件所涵盖的系统设置，请使用 **sysctl** 插件。如果某些特定插件提供了设置，首选这些插件。

usb

将 USB 设备的自动暂停超时设置为 **autosuspend** 参数指定的值。
值 **0** 表示禁用自动暂停。

vm

启用或禁用透明大内存页，具体取决于 transparent **_hugepages** 选项的值。
transparent_hugepages 选项的有效值为：

- "always"
- "never"

- "madvise"

audio

将音频解码器的 autosuspend timeout 设置为 **timeout** 选项指定的值。
目前，支持 **snd_hda_intel** 和 **snd_ac97_codec** codec。值 **0** 表示自动暂停已被禁用。您还可以通过将 布尔值选项 **reset_controller** 设置为 **true** 来强制实施控制器重置。

disk

将磁盘电梯设置为 **elevator** 选项指定的值。
它还设置：

- **apm** 选项指定的值的 APM
- 调度程序对由 **scheduler_quantum** 选项指定的值进行量化
- 磁盘 spindown 的超时值由 **spindown** 选项指定的值
- 磁盘的 readahead 会到 **readahead** 参数指定的值
- 当前磁盘 readahead 值乘以 **readahead_multiply** 选项指定的常数

此外，此插件根据当前的驱动器利用率动态地更改驱动器的高级电源管理和机超时设置。动态调优可以由布尔值选项 **动态** 控制，默认情况下是启用的。

scsi_host

SCSI 主机的选项调整。
它将积极链接电源管理 (ALPM) 设置为 **alpm** 选项指定的值。

mounts

根据 **disable_barriers** 选项的布尔值启用或禁用挂载障碍。

script

加载或卸载配置集时，执行外部脚本或二进制代码。您可以选择任意可执行文件。

重要

script 插件主要被用来与更早的版本兼容。如果其他 **TuneD** 插件涵盖所需的功能，则首选其他 TuneD 插件。

TuneD 使用以下参数之一调用可执行文件：

- 在载入配置集时 **start**
- 在卸载配置集时 **stop**

您需要在可执行文件中正确实施 **stop** 操作，并恢复您在 **start** 操作过程中更改的所有设置。否则，在更改 **TuneD** 配置集后回滚步骤将无法正常工作。

Bash 脚本可以导入 **/usr/lib/tuned/functions** Bash 库，并使用那里定义的功能。只在由 **TuneD** 原生提供的功能中使用这些功能。如果函数名称以下划线开头，如 **_wifi_set_power_level**，请考虑函数私有且不要在脚本中使用，因为它可能会在以后有所变化。

使用插件配置中的 **script** 参数指定可执行文件的路径。

例 2.6. 从配置集运行 Bash 脚本

要运行位于配置集目录中的 **script.sh** 的 Bash 脚本，请使用：

```
[script]
script=${i:PROFILE_DIR}/script.sh
```

sysfs

设置由插件选项指定的各种 **sysfs** 设置。
语法为 ***name=value***，其中 *name* 是要使用的 **sysfs** 路径。

如果需要更改其他插件未涵盖的一些设置，请使用此插件。如果插件涵盖所需的设置，则首选插件。

video

在视频卡中设置各种电源保存级别。目前，只支持 Radeon 卡。
可以使用 **radeon_powersave** 选项指定节能级别。支持的值有：

- **default**
- **auto**
- **低**
- **mid**
- **high**
- **dynpm**
- **dpm-battery**
- **dpm-balanced**
- **dpm-perfomance**

详情请查看 www.x.org。请注意，此插件是实验性的，选项可能会在以后的版本中有所变化。

bootloader

在内核命令行中添加选项。这个插件只支持 GRUB 2 引导装载程序。
grub2_cfg_file 选项指定 GRUB 2 配置文件的自定义非标准位置。

内核选项会添加到当前 GRUB 配置及其模板中。需要重新引导系统才能使内核选项生效。

切换到另一个配置集或手动停止 **TuneD** 服务会删除附加选项。如果您关闭或重启系统，则 kernel 选项会在 **grub.cfg** 文件中保留。

内核选项可使用以下语法指定：

```
cmdline=arg1 arg2 ... argN
```

例 2.7. 修改内核命令行

例如，要将 **quiet** kernel 选项添加到 **TuneD** 配置集中，请在 **tuned.conf** 文件中包括以下行：

```
[bootloader]
cmdline=quiet
```

以下是在内核命令行中添加 **isolcpus=2** 选项的自定义配置集示例：

```
[bootloader]
cmdline=isolcpus=2
```

service

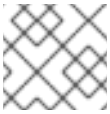
处理由插件选项指定的各种 **sysvinit**、**sysv-rc**、**openrc** 和 **systemd** 服务。
语法为 **service.*service_name*=*command*[,file:*file*]**。

支持的服务处理命令有：

- **start**
- **stop**
- **enable**
- **disable**

使用逗号(,)或分号(;)分隔多个命令。如果指令冲突，**service** 插件将使用最后列出的。

使用可选的 **file:*file*** 指令，仅为 **systemd** 安装一个覆盖配置文件 ***file***。其他 init 系统会忽略这个指令。**service** 插件将覆盖配置文件复制到 **/etc/systemd/system/*service_name*.service.d/** 目录中。卸载配置文件后，如果目录为空，**service** 插件会删除它们。



注意

service 插件仅在带有非**systemd** init 系统的当前运行级别上运行。

例 2.8. 启动并启用带有覆盖文件的sendmail sendmail 服务

```
[service]
service.sendmail=start,enable,file:${i:PROFILE_DIR}/tuned-sendmail.conf
```

内部变量 **${i:PROFILE_DIR}** 指向插件从中加载配置文件的目录。

scheduler

提供调优调度优先级的各种选项，CPU 核隔离，以及进程、线程和 IRQ 相关性。

有关可用的不同选项的具体内容，请参阅 **scheduler** TuneD 插件的功能。

2.9. SCHEDULER TUNED 插件的功能

使用 **scheduler TuneD** 插件控制并调优调度优先级、CPU 核隔离和进程、线程和 IRQ 相关性。

CPU 隔离

要防止进程、线程和 IRQ 使用某些 CPU，请使用 **isolated_cores** 选项。它更改进程和线程关联性、IRQ 关联性，并为 IRQ 设置 **default_smp_affinity** 参数。

根据 **sched_setaffinity()** 系统调用的成功，对与 **ps_whitelist** 选项匹配的所有进程和线程，会调整 CPU 关联性掩码。**ps_whitelist** 正则表达式的默认设置是 **.***，以匹配所有进程和线程名称。要排除某些进程和线程，请使用 **ps_blacklist** 选项。这个选项的值也被解释为一个正则表达式。进程和线程名称与该表达式匹配。配置文件回滚可让所有匹配的进程和线程在所有 CPU 上运行，并在配置文件应用程序之前恢复 IRQ 设置。

对于 **ps_whitelist** 和 **ps_blacklist** 选项，支持用 **;** 分隔的多个正则表达式。转义的分号 **\;** 按字面处理。

例 2.9. 隔离 CPU 2-4

以下配置隔离 CPU 2-4.与 **ps_blacklist** 正则表达式匹配的进程和线程可以使用任何 CPU，而不考虑隔离：

```
[scheduler]
isolated_cores=2-4
ps_blacklist=.*pmd.*;.*PMD.*;^DPDK;.*qemu-kvm.*
```

IRQ SMP 关联性

/proc/irq/default_smp_affinity 文件包含一个位掩码，表示系统上所有不活跃中断请求(IRQ)源的的默认目标 CPU 核。激活或分配 IRQ 后，**/proc/irq/default_smp_affinity** 文件中的值决定 IRQ 的关联性位掩码。

default_irq_smp_affinity 参数控制 TuneD 向 **/proc/irq/default_smp_affinity** 文件中写入什么。**default_irq_smp_affinity** 参数支持以下值和行为：

calc

从 **isolated_cores** 参数计算 **/proc/irq/default_smp_affinity** 文件的内容。**isolated_cores** 参数的反转计算非隔离核。
然后，将非隔离核和之前 **/proc/irq/default_smp_affinity** 文件的交集写入 **/proc/irq/default_smp_affinity** 文件中。

如果省略了 **default_irq_smp_affinity** 参数，则这是默认行为。

ignore

TuneD 不修改 **/proc/irq/default_smp_affinity** 文件。

CPU 列表

使用单个数字的形式，如 **1**，逗号分隔的列表，如 **1、3** 或范围，如 **3-5**。
解开 CPU 列表，并将其直接写到 **/proc/irq/default_smp_affinity** 文件中。

例 2.10. 使用显式 CPU 列表设置默认的 IRQ smp 关联性

以下示例使用显式 CPU 列表将默认的 IRQ SMP 关联性设置为 CPU 0 和 2：

```
[scheduler]
isolated_cores=1,3
default_irq_smp_affinity=0,2
```

调度策略

要调整一组进程或线程的调度策略、优先级和关联性，请使用以下语法：

```
group.groupname=rule_prio:sched:prio:affinity:regex
```

其中 ***rule_prio*** 定义规则的内部 **TuneD** 优先级。规则根据优先级排序。这是继承的需要，以能够重新排序之前定义的规则。应按照定义的顺序处理相同的 ***rule_prio*** 规则。但是，这是依赖于 Python 解释器。要禁用 ***groupname*** 的一个继承规则，请使用：

```
group.groupname=
```

sched 必须是以下之一：

f
: 对于先进先出(FIFO)

b
: 对于批处理

r
: 对于循环

o
: 对于其他

*
: 对于不更改

关联性 是十六进制的 CPU 关联性。对于没有更改，使用 *。

prio 是调度优先级（请参阅 **chrt -m**）。

regex 是 Python 正则表达式。它与 **ps -eo cmd** 命令的输出匹配。

任何给定的进程名称可以匹配多个组。在这种情况下，最后匹配的 ***regex*** 决定优先级和调度策略。

例 2.11. 设置调度策略和优先级

以下示例对内核线程和 watchdog 设置调度策略和优先级：

```
[scheduler]
group.kthreads=0:*:1:*:\[.*\]$
group.watchdog=0:f:99:*:\[watchdog.*\]
```

scheduler 插件使用 **perf** 事件循环来识别新创建的进程。默认情况下，它侦听 **perf.RECORD_COMM** 和 **perf.RECORD_EXIT** 事件。

将 **perf_process_fork** 参数设置为 **true**，来告知插件也侦听 **perf.RECORD_FORK** 事件，这意味着由 **fork （）** 系统调用创建的子进程已被处理。

注意

处理 **perf** 事件可能会造成大量 CPU 开销。

可以使用 scheduler **runtime** 选项来缓解scheduler 插件的 CPU 开销，并将其设置为 **0**。这会完全禁用动态 scheduler 功能，并且不会对 perf 事件进行监控和操作。这样做的缺点是，进程和线程调优只能在配置文件应用程序中完成。

例 2.12. 禁用动态 scheduler 功能

以下示例禁用了动态 scheduler 功能，同时也隔离了 CPU 1 和 3：

```
[scheduler]
runtime=0
isolated_cores=1,3
```

mmapped 缓冲用于 **perf** 事件。在负载过重时，此缓冲区可能会溢出，因此插件可能会开始丢失事件，不会处理一些新创建的进程。在这种情况下，使用 **perf_mmap_pages** 参数来增加缓冲区大小。**perf_mmap_pages** 参数的值必须是 2 的幂。如果没有手动设置 **perf_mmap_pages** 参数，则使用默认值 128。

使用 cgroups 的限制

scheduler 插件支持使用 **cgroup** v1 的进程和线程限制。

cgroup_mount_point 选项指定挂载 cgroup 文件系统的路径，或者 TuneD 预期它挂载在哪里。如果未设置，则预期为 **/sys/fs/cgroup/cpuset**。

如果 **cgroup_groups_init** 选项被设置为 **1**，则TuneD 会创建和删除使用 **cgroup*** 选项定义的所有 **cgroups**。这是默认的行为。如果 **cgroup_mount_point** 选项设为 **0**，则必须使用其他方法预设置 **cgroups**。

如果 **cgroup_mount_point_init** 选项被设置为 **1**，则TuneD 会创建和删除 cgroup 挂载点。其暗示 **cgroup_groups_init = 1**。如果 **cgroup_mount_point_init** 选项被设置为 **0**，则必须通过其他方法预设置 **cgroups** 挂载点。这是默认的行为。

cgroup_for_isolated_cores 选项是 **isolated_cores** 选项功能的 **cgroup** 名称。例如，如果系统有 4 个 CPU，**isolated_cores=1** 表示 Tuned 将所有进程和线程移到 CPU 0、2 和 3。**scheduler** 插件通过将计算的 CPU 关联性写入指定 cgroup 的 **cpuset.cpus** 控制文件来隔离指定的核，并将所有匹配的进程和线程移到这个组中。如果此选项未设置，则使用 **sched_setaffinity （）** 的经典 cpuset 关联性设置 CPU 关联性。

cgroup.*cgroup_name* 选项为任意 **cgroup** 定义关联性。您甚至可以使用层次结构的 cgroups，但您必须按正确顺序指定层次结构。**TuneD** 不会在此处做任何健全性检查，但它会强制 **cgroup** 位于 **cgroup_mount_point** 选项指定的位置。

以 **group.** 开头的 scheduler 选项的语法已被增强来使用 **cgroup.*cgroup_name***，而不是十六进制 ***关联性***。匹配的进程被移到 **cgroup *cgroup_name*** 中。您还可以使用不是由 **cgroup.** 选项定义的 cgroups，如上所述。例如，**cgroups** 不是由 TuneD 管理的。

所有 **cgroup** 名称通过使用斜杠(/)替换所有句点(.)来进行清理。这可防止插件写到 **cgroup_mount_point** 选项指定的位置外。

例 2.13. 使用带有 scheduler 插件的 cgroup v1

以下示例创建 2 个 **cgroups**，**group1** 和 **group2**。它将 cgroup **group1** 关联性设置为 CPU 2，将 **cgroup group2** 设置为 CPU 0 和 2。假有定 4 个 CPU 设置， **isolated_cores=1** 选项将所有进程和线程移到 CPU 核 0、2 和 3。**ps_blacklist** 正则表达式指定的进程和线程没有移动。

```
[scheduler]
cgroup_mount_point=/sys/fs/cgroup/cpuset
cgroup_mount_point_init=1
cgroup_groups_init=1
cgroup_for_isolated_cores=group
cgroup.group1=2
cgroup.group2=0,2

group.ksoftirqd=0:f:2:cgroup.group1:ksoftirqd.*
ps_blacklist=ksoftirqd.*;rcuc.*;rcub.*;ktimersoftd.*
isolated_cores=1
```

cgroup_ps_blacklist 选项排除了属于指定 **cgroup** 的进程。此选项指定的正则表达式与 **/proc/*PID*/cgroups** 的 **cgroup** 层次结构匹配。在正则表达式匹配前，逗号(,)将 **cgroups** v1 层次结构与 **/proc/*PID*/cgroups** 分隔开。以下是正则表达式匹配的内容的示例：

```
10:hugetlb:/,9:perf_event:/,8:blkio:/
```

多个正则表达式可以通过分号(;)分隔。分号表示逻辑"或"运算符。

例 2.14. 使用 cgroup 从调度程序中排除进程

在以下示例中，**scheduler** 插件将所有进程从核 1 移出，但属于 cgroup **/daemons** 的进程除外。**\b** 字符串是一个与单词边界匹配的正则表达式元字符。

```
[scheduler]
isolated_cores=1
cgroup_ps_blacklist=:/daemons\b
```

在以下示例中，**scheduler** 插件排除了属于层次结构 ID 为 8 和 controller-list **blkio** 的 cgroup 的所有进程。

```
[scheduler]
isolated_cores=1
cgroup_ps_blacklist=\b8:blkio:
```

最近的内核将一些 **sched_** 和 **numa_balancing_** 内核运行时参数从 **sysctl** 工具管理的 **/proc/sys/kernel** 目录移到 **debugfs**，通常挂载在 **/sys/kernel/debug** 目录下。**TuneD** 通过 **scheduler** 插件为以下参数提供抽象机制，其中，根据使用的内核，**TuneD** 将指定的值写到正确的位置：

- **sched_min_granularity_ns**
- **sched_latency_ns**,
- **sched_wakeup_granularity_ns**
- **sched_tunable_scaling**,
- **sched_migration_cost_ns**
- **sched_nr_migrate**

- **numa_balancing_scan_delay_ms**
- **numa_balancing_scan_period_min_ms**
- **numa_balancing_scan_period_max_ms**
- **numa_balancing_scan_size_mb**

例 2.15. 为迁移决策设置任务的"cache hot"值。

在旧内核上，设置以下参数意味着 **sysctl** 将值 **500000** 写到 **/proc/sys/kernel/sched_migration_cost_ns** 文件中：

```
[sysctl]
kernel.sched_migration_cost_ns=500000
```

在最近的内核上，这相当于通过 **scheduler** 插件设置以下参数：

```
[scheduler]
sched_migration_cost_ns=500000
```

意味着 TuneD 将值 **500000** 写到 **/sys/kernel/debug/sched/migration_cost_ns** 文件中。

2.10. TUNED 配置集中的变量

激活 TuneD 配置集时，在运行时扩展的变量。

使用 TuneD 变量可减少 TuneD 配置集中必要输入的数量。

TuneD 配置集中没有预定义的变量。您可以通过在配置集中创建 **[variables]** 部分并使用以下语法来定义您自己的变量：

```
[variables]

variable_name=value
```

要扩展配置集中的变量的值，请使用以下语法：

```
${variable_name}
```

例 2.16. 使用变量隔离 CPU 内核

在以下示例中，**${isolated_cores}** 变量扩展至 **1,2**; 因此内核使用 **isolcpus=1,2** 选项引导：

```
[variables]
isolated_cores=1,2

[bootloader]
cmdline=isolcpus=${isolated_cores}
```

变量可以在单独的文件中指定。例如，您可以在 **tuned.conf** 中添加以下行：

```
[variables]
```

```
include=/etc/tuned/my-variables.conf

[bootloader]
cmdline=isolcpus=${isolated_cores}
```

如果您将 **isolated_cores=1,2** 选项添加到 **/etc/tuned/my-variables.conf** 文件，则内核会使用 **isolcpus=1,2** 选项引导。

其他资源

- **tuned.conf (5)** 手册页

2.11. TUNED 配置集中的内置功能

当激活 TuneD 配置集时，内置功能会在运行时扩展。

您可以：

- 与 TuneD 变量一起使用各种内置功能
- 在 Python 中创建自定义功能，并以插件的形式将它们添加到 TuneD

要调用函数，请使用以下语法：

```
${f:function_name:argument_1:argument_2}
```

要扩展配置集和 **tuned.conf** 文件所在的目录路径，请使用 **PROFILE_DIR** 功能，它需要特殊语法：

```
${i:PROFILE_DIR}
```

例 2.17. 使用变量和内置功能隔离 CPU 内核

在以下示例中, **${non_isolated_cores}** 变量扩展至 **0,3-5**，且 **cpulist_invert** 内置函数使用 **0,3-5** 参数调用：

```
[variables]
non_isolated_cores=0,3-5

[bootloader]
cmdline=isolcpus=${f:cpulist_invert:${non_isolated_cores}}
```

cpulist_invert 功能反转 CPU 列表。对于 6-CPU 机器，inversion 为 **1,2**，内核通过 **isolcpus=1,2** 命令行选项引导。

其他资源

- **tuned.conf (5)** 手册页

2.12. TUNED 配置集中的内置功能

所有 TuneD 配置集中都有以下内置功能：

PROFILE_DIR

返回配置文件和 **tuned.conf** 文件所在的目录路径。

exec

执行进程并返回其输出。

assertion

比较两个参数。如果*不匹配*，会在日志中记录来自第一个参数的信息，并中止配置集加载。

assertion_non_equal

比较两个参数。如果*不匹配*，会在日志中记录来自第一个参数的信息，并中止配置集加载。

kb2s

将 KB 转换为磁盘扇区.

s2kb

将磁盘扇区转换为 KB。

strip

从所有传递的参数创建字符串，并删除前导和尾随空格。

virt_check

检查 **TuneD** 是否在虚拟机 (VM) 或裸机中运行：

- 在虚拟机内部，函数返回第一个参数。
- 在裸机上，函数返回第二个参数，即使出现错误。

cpulist_invert

颠倒 CPU 列表，使其补充。例如，在一个有 4 个 CPU 的系统上，从 0 到 3，列表 **0,2,3** 的反转是 **1**。

cpulist2hex

将 CPU 列表转换为十六进制 CPU 掩码。

cpulist2hex_invert

将 CPU 列表转换为十六进制 CPU 掩码并进行反转。

hex2cpulist

将十六进制 CPU 掩码转换为 CPU 列表。

cpulist_online

检查列表中的 CPU 是否在线。返回仅包含在线 CPU 的列表。

cpulist_present

检查列表中是否存在 CPU。返回只包含当前 CPU 的列表。

cpulist_unpack

解包 CPU 列表，格式为 **1-3,4** 到 **1,2,3,4**。

cpulist_pack

把包 CPU 列表，格式为 **1,2,3,5** 到 **1-3,5**

2.13. 创建新的 TUNED 配置集

此流程使用自定义性能配置集创建一个新的 **TuneD** 配置集。

先决条件

- **TuneD** 服务正在运行。详情请参阅安装和启用 TuneD。

步骤

1. 在 **/etc/tuned/** 目录中，创建一个名为您要创建的配置集的新目录：

```
# mkdir /etc/tuned/my-profile
```

2. 在新目录中，创建名为 **tuned.conf** 的文件。根据您的要求，添加一个 **[main]** 部分和插件定义。例如，查看 **balanced** 配置集的配置：

```
[main]
summary=General non-specialized TuneD profile

[cpu]
governor=conservative
energy_perf_bias=normal

[audio]
timeout=10

[video]
radeon_powersave=dpm-balanced, auto

[scsi_host]
alpm=medium_power
```

3. 要激活配置集，请使用：

```
# tuned-adm profile my-profile
```

4. 验证 **TuneD** 配置集是否活跃，并应用了系统设置：

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

其他资源

- **tuned.conf (5)** 手册页

2.14. 修改现有 TUNED 配置集

此流程根据现有的 **TuneD** 配置集创建修改后的子配置集。

先决条件

- **TuneD** 服务正在运行。详情请参阅安装和启用 TuneD。

步骤

1. 在 **/etc/tuned/** 目录中，创建一个名为您要创建的配置集的新目录：

```
# mkdir /etc/tuned/modified-profile
```

2. 在新目录中，创建一个名为 **tuned.conf** 的文件，并按如下所示设置 **[main]** 部分：

```
[main]
include=parent-profile
```

使用您要修改的配置集的名称替换 *parent-profile*。

3. 包括您的配置集修改。

例 2.18. 在 throughput-performance 配置集中降低 swappiness

要使用 **throughput-performance** 配置集的设置，并将 **vm.swappiness** 的值改为 5，而不是默认的 10，请使用：

```
[main]
include=throughput-performance

[sysctl]
vm.swappiness=5
```

4. 要激活配置集，请使用：

```
# tuned-adm profile modified-profile
```

5. 验证 **TuneD** 配置集是否活跃，并应用了系统设置：

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

其他资源

- **tuned.conf (5)** 手册页

2.15. 使用 TUNED 设置磁盘调度程序

此流程创建并启用 **TuneD** 配置集，该配置集为所选块设备设置给定磁盘调度程序。这个设置会在系统重启后保留。

在以下命令和配置中替换：

- 带有块设备名称的 *device*，如 **sdf**
- 带有您要为该设备设置的磁盘调度程序的 *selected-scheduler*，例如 **bfq**

先决条件

- **TuneD** 服务已安装并启用。详情请参阅安装和启用 TuneD。

流程

1. 可选：选择一个您的配置集将要基于的现有 **Tuned** 配置集。有关可用配置集列表，请参阅 RHEL 提供的 TuneD 配置集。
   要查看哪个配置集当前处于活跃状态，请使用：

   ```
   $ tuned-adm active
   ```

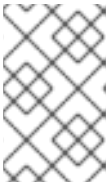
2. 创建一个新目录来保存 **TuneD** 配置集：

   ```
   # mkdir /etc/tuned/my-profile
   ```

3. 查找所选块设备系统唯一标识符：

   ```
   $ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'

   ID_WWN=0x5002538d00000000_
   ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
   ID_SERIAL_SHORT=20120501030900000
   ```

   注意

   本例中的命令将返回以 World Wide Name (WWN) 或与指定块设备关联的序列号的所有值。虽然最好使用 WWN，但给定设备始终不能使用 WWN，但 example 命令返回的任何值都可以接受用作 *device system unique ID*。

4. 创建 **/etc/tuned/*my-profile*/tuned.conf** 配置文件。在该文件中设置以下选项：

   a. 可选：包含现有配置集：

   ```
   [main]
   include=existing-profile
   ```

   b. 为与 WWN 标识符匹配的设备设置所选磁盘调度程序：

   ```
   [disk]
   devices_udev_regex=IDNAME=device system unique id
   elevator=selected-scheduler
   ```

   在这里：

   - 使用要使用的标识符的名称替换 *IDNAME*（如 **ID_WWN**）。

- 将 *device system unique id* 替换为所选标识符的值（如 **0x5002538d00000000**）。要匹配 **devices_udev_regex** 选项中的多个设备，将标识符放在括号中，并使用垂直栏来分离它们：

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. 启用您的配置集：

```
# tuned-adm profile my-profile
```

验证步骤

1. 验证 TuneD 配置集是否活跃并应用：

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See TuneD log file ('/var/log/tuned/tuned.log') for details.
```

2. 读取 **/sys/block/***设备***/queue/scheduler** 文件的内容：

```
# cat /sys/block/device/queue/scheduler

[mq-deadline] kyber bfq none
```

在文件名中，将 *device* 替换为块设备名称，如 **sdc**。

活跃的调度程序列在方括号中 (**[]**) 。

其他资源

- 自定义 TuneD 配置集。

第 3 章 使用 TUNA 接口检查系统

tuna 工具减少了执行调优任务的复杂性。使用 **tuna** 调整调度程序可调项、调优线程优先级、RRQ 处理程序以及隔离 CPU 核和套接字。通过使用 **tuna** 工具，您可以执行以下操作：

- 列出系统上的 CPU。
- 列出系统上当前运行的中断请求(IRQ)。
- 更改有关线程的策略和优先级信息。
- 显示系统的当前策略和优先级。

3.1. 安装 TUNA 工具

tuna 工具设计为在运行中的系统上使用。这允许特定应用程序的测量工具在更改后马上查看和分析系统性能。

流程

- 安装 **tuna** 工具：

```
# dnf install tuna
```

验证步骤

- 显示可用的 **tuna** CLI 选项：

```
# tuna -h
```

其他资源

- **tuna (8)** 手册页

3.2. 使用 TUNA 工具查看系统状态

您可以使用 **tuna** 命令行界面(CLI)工具查看系统状态。

先决条件

- **tuna** 工具已安装。如需更多信息，请参阅 安装 tuna 工具。

流程

1. 查看当前的策略和优先级：

```
# tuna show_threads
pid   SCHED_ rtpri affinity           cmd
1     OTHER     0      0,1            init
2      FIFO    99        0     migration/0
3     OTHER     0        0     ksoftirqd/0
4      FIFO    99        0      watchdog/0
```

或者，要查看与 PID 或与命令名称匹配的具体线程，请输入：

```
# tuna show_threads -t pid_or_cmd_list
```

pid_or_cmd_list 参数是一个用逗号分开的 PID 或 command-name 模式的列表。

2. 根据您的场景，执行以下操作之一：
   - 要使用 **tuna** CLI 调优 CPU，请完成 使用 tuna 工具调优 CPU 中的步骤。
   - 要使用 **tuna** 工具调优 IRQ，请完成 使用 tuna工具调优 IRQ 中的步骤。
3. 保存更改的配置：

```
# tuna save filename
```

这个命令只保存当前运行的内核线程。未运行的进程不会保存。

其他资源

- 系统上 **tuna (8)** 手册页

3.3. 使用 TUNA 工具调优 CPU

tuna 工具命令可以针对单个 CPU 为目标。通过使用 **tuna** 工具，您可以执行以下操作：

隔离 CPU

在指定 CPU 上运行的所有任务都移至下一个可用 CPU。通过将 CPU 从所有线程的关联性掩码中删除来隔离 CPU，使其不可用。

包括 CPU

允许任务在指定的 CPU 上运行。

恢复 CPU

将指定的 CPU 恢复到之前的配置。

先决条件

- **tuna** 工具已安装。如需更多信息，请参阅 安装 tuna 工具。

流程

1. 列出所有 CPU，并指定受命令影响的 CPU 的列表：

```
# ps ax | awk 'BEGIN { ORS="," }{ print $1 }'
PID,1,2,3,4,5,6,8,10,11,12,13,14,15,16,17,19
```

2. 在 **tuna** 界面中显示线程列表：

```
# tuna show_threads -t 'thread_list from above cmd'
```

3. 指定要受某一命令影响的 CPU 的列表：

```
# *tuna [command] --cpus cpu_list *
```

cpu_list 参数是一个用逗号分开的 CPU 号的列表，如 **--cpus *0,2***。

要将特定 CPU 添加到当前的 *cpu_list* 中，请使用例如 **--cpus +0**。

4. 根据您的场景，执行以下操作之一：

   - 要隔离一个 CPU，请输入：

     ```
     # tuna isolate --cpus cpu_list
     ```

   - 要包括一个 CPU，请输入：

     ```
     # tuna include --cpus cpu_list
     ```

5. 要使用具有四个或更多处理器的系统，请使所有 **ssh** 线程在 CPU *0* 和 *1* 上运行，使所有 **http** 线程在 CPU *2* 和 *3* 上运行：

   ```
   # tuna move --cpus 0,1 -t ssh*
   # tuna move --cpus 2,3 -t http\*
   ```

验证步骤

- 显示当前配置，并验证更改是否已应用：

  ```
  # tuna show_threads -t ssh*

  pid   SCHED_  rtpri  affinity   voluntary   nonvoluntary   cmd
  855   OTHER   0      0,1        23          15             sshd

  # tuna show_threads -t http\*
  pid   SCHED_  rtpri  affinity   voluntary   nonvoluntary   cmd
  855   OTHER   0      2,3        23          15             http
  ```

其他资源

- **/proc/cpuinfo** 文件
- 系统上 **tuna (8)** 手册页

3.4. 使用 TUNA 工具调优 IRQ

/proc/interrupts 文件记录每个 IRQ 的中断数、中断类型和位于 IRQ 的设备的名称。

先决条件

- **tuna** 工具已安装。如需更多信息，请参阅安装 tuna 工具。

流程

1. 查看当前的 IRQ 及其关联性：

   ```
   # tuna show_irqs
   # users          affinity
   ```

```
0 timer                 0
1 i8042                 0
7 parport0              0
```

2. 指定要受某一命令影响的 IRQ 的列表：

```
# tuna [command] --irqs irq_list --cpus cpu_list
```

irq_list 参数是用逗号分开的 IRQ 编号或 user-name 模式的列表。

将 [*command*] 替换为例如 **--spread**：

3. 将一个中断移到指定的 CPU 中：

```
# tuna show_irqs --irqs 128
users          affinity
128 iwlwifi          0,1,2,3

# tuna move --irqs 128 --cpus 3
```

使用 irq_list 参数替换 *128*，将 *3* 替换为 cpu_list 参数。

cpu_list 参数是一个用逗号分开的 CPU 号的列表，如 **--cpus *0,2***。如需更多信息，请参阅 使用 tuna 工具调优 CPU。

验证步骤

- 在将任何中断移到特定的 CPU 之前和之后，比较所选 IRQs 的状态：

```
# tuna show_irqs --irqs 128
   users          affinity
 128 iwlwifi               3
```

其他资源

- **/procs/interrupts** 文件
- 系统上 **tuna (8)** 手册页

第 4 章 使用 RHEL 系统角色监控性能

作为系统管理员，您可以使用 **metrics** RHEL 系统角色监控系统的性能。

4.1. 准备一个控制节点和受管节点来使用 RHEL 系统角色

在使用单独的 RHEL 系统角色管理服务和设置前，您必须准备控制节点和受管节点。

4.1.1. 在 RHEL 9 上准备一个控制节点

在使用 RHEL 系统角色前，您必须配置一个控制节点。然后，此系统根据 playbook 从清单中配置受管主机。

前提条件

- 该系统已在客户门户网站中注册。
- **Red Hat Enterprise Linux Server** 订阅已附加到系统。
- 可选：**Ansible Automation Platform** 订阅被附加到系统上。

流程

1. 创建一个名为 **ansible** 的用户，来管理并运行 playbook：

```
[root@control-node]# useradd ansible
```

2. 切换到新创建的 **ansible** 用户：

```
[root@control-node]# su - ansible
```

以这个用户身份执行其余步骤。

3. 创建一个 SSH 公钥和私钥：

```
[ansible@control-node]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Enter passphrase (empty for no passphrase): <password>
Enter same passphrase again: <password>
...
```

为密钥文件使用推荐的默认位置。

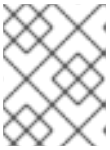
4. 可选： 要防止 Ansible 在每次建立连接时提示您输入 SSH 密钥密码，请配置一个 SSH 代理。

5. 使用以下内容创建 ~/**.ansible.cfg** 文件：

```
[defaults]
inventory = /home/ansible/inventory
remote_user = ansible

[privilege_escalation]
become = True
```

```
become_method = sudo
become_user = root
become_ask_pass = True
```



注意

~/.ansible.cfg 文件中的设置具有更高的优先级，并覆盖全局 **/etc/ansible/ansible.cfg** 文件中的设置。

使用这些设置，Ansible 执行以下操作：

- 管理指定清单文件中的主机。
- 当帐户建立到受管节点的 SSH 连接时，使用 **remote_user** 参数中设置的帐户。
- 使用 **sudo** 工具，以 **root** 用户身份在受管节点上执行任务。
- 每次应用 playbook 时，都会提示输入远程用户的 root 密码。出于安全考虑，建议这样做。

6. 创建一个列出受管主机主机名的 INI 或 YAML 格式的 **~/inventory** 文件。您还可以在清单文件中定义主机组。例如，以下是 INI 格式的清单文件，它有三个主机，以及一个名为 **US** 的主机组：

```
managed-node-01.example.com

[US]
managed-node-02.example.com ansible_host=192.0.2.100
managed-node-03.example.com
```

请注意，控制节点必须能够解析主机名。如果 DNS 服务器无法解析某些主机名，请在主机条目旁边添加 **ansible_host** 参数来指定其 IP 地址。

7. 安装 RHEL 系统角色：

- 在没有 Ansible Automation Platform 的 RHEL 主机上，安装 **rhel-system-roles** 软件包：

```
[root@control-node]# dnf install rhel-system-roles
```

此命令在 **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/** 目录中安装集合，且 **ansible-core** 软件包作为依赖项。

- 在 Ansible Automation Platform 上，以 **ansible** 用户身份执行以下步骤：

i. 将 Red Hat Automation hub 定义为 **~/.ansible.cfg** 文件中内容的主要源。

ii. 从 Red Hat Automation Hub 安装 **redhat.rhel_system_roles** 集合：

```
[ansible@control-node]$ ansible-galaxy collection install
redhat.rhel_system_roles
```

此命令在 **~/.ansible/collections/ansible_collections/redhat/rhel_system_roles/** 目录中安装集合。

后续步骤

- 准备受管节点。如需更多信息，请参阅 准备一个受管节点。

其他资源

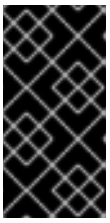
- RHEL 9 和 RHEL 8.6 及更新的 AppStream 软件仓库中包含的 Ansible Core 软件包支持范围
- 如何使用 subscription-manager 在红帽客户门户网站中注册和订阅系统
- **ssh-keygen (1)** 手册页
- 通过 ssh-agent，使用 SSH 密钥连接到远程机器
- Ansible 配置设置
- 如何构建清单

4.1.2. 准备受管节点

受管节点是在清单中列出的系统，它由控制节点根据 playbook 进行配置。您不必在受管主机上安装 Ansible。

前提条件

- 您已准备好了控制节点。如需更多信息，请参阅 在 RHEL 9 上准备一个控制节点 。
- 您从控制节点进行 SSH 访问的权限。



重要

以 **root** 用户身份进行直接的 SSH 访问是一个安全风险。要降低这个风险，您将在此节点上创建一个本地用户，并在准备受管节点时配置一个 **sudo** 策略。然后，控制节点上的 Ansible 可以使用本地用户帐户登录到受管节点，并以不同的用户身份（如 **root** ）运行 playbook。

流程

1. 创建一个名为 **ansible** 的用户：

```
[root@managed-node-01]# useradd ansible
```

控制节点稍后使用这个用户建立与这个主机的 SSH 连接。

2. 为 **ansible** 用户设置密码：

```
[root@managed-node-01]# passwd ansible
Changing password for user ansible.
New password: <password>
Retype new password: <password>
passwd: all authentication tokens updated successfully.
```

当 Ansible 使用 **sudo** 以 **root** 用户身份执行任务时，您必须输入此密码。

3. 在受管主机上安装 **ansible** 用户的 SSH 公钥：

   a. 以 **ansible** 用户身份登录到控制节点，并将 SSH 公钥复制到受管节点：

```
[ansible@control-node]$ ssh-copy-id managed-node-01.example.com
```

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/ansible/.ssh/id_rsa.pub"
The authenticity of host 'managed-node-01.example.com (192.0.2.100)' can't be
established.
ECDSA key fingerprint is
SHA256:9bZ33GJNODK3zbNhybokN/6Mq7hu3vpBXDrCxe7NAvo.
```

b. 当提示时，输入 **yes** 进行连接：

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
```

c. 当提示时，输入密码：

```
ansible@managed-node-01.example.com's password: <password>

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'managed-node-01.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

d. 通过在控制节点上远程执行命令来验证 SSH 连接：

```
[ansible@control-node]$ ssh managed-node-01.example.com whoami
ansible
```

4. 为 **ansible** 用户创建 **sudo** 配置：

a. 使用 **visudo** 命令创建并编辑 **/etc/sudoers.d/ansible** 文件：

```
[root@managed-node-01]# visudo /etc/sudoers.d/ansible
```

在普通编辑器中使用 **visudo** 的好处是，该实用程序提供基本的检查，如用于解析错误，然后再安装该文件。

b. 在 **/etc/sudoers.d/ansible** 文件中配置满足您要求的 **sudoers** 策略，例如：

- 要为 **ansible** 用户授予权限，以便在输入 **ansible** 用户密码后在此主机上以任何用户和组身份来运行所有命令，请使用：

```
ansible   ALL=(ALL) ALL
```

- 要向 **ansible** 用户授予权限，以便在不输入 **ansible** 用户密码的情况下在该主机上以任何用户和组身份来运行所有命令，请使用：

```
ansible   ALL=(ALL) NOPASSWD: ALL
```

或者，配置匹配您安全要求的更精细的策略。有关 **sudoers** 策略的详情，请查看 **sudoers (5)** 手册页。

验证

1. 验证您可以在所有受管节点上执行来自控制节点的命令：

```
[ansible@control-node]$ ansible all -m ping
BECOME password: <password>
managed-node-01.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
...
```

硬编码的所有组会动态包含清单文件中列出的所有主机。

2. 使用 Ansible **command** 模块在受管主机上运行 **whoami** 工具来验证特权升级是否可以正常工作：

```
[ansible@control-node]$ ansible managed-node-01.example.com -m command -a whoami
BECOME password: <password>
managed-node-01.example.com | CHANGED | rc=0 >>
root
```

如果命令返回 root，则您在受管节点上正确地配置了 **sudo**。

其他资源

- 在 RHEL 9 上准备一个控制节点
- **sudoers (5)** 手册页

4.2. METRICS RHEL 系统角色简介

RHEL 系统角色是 Ansible 角色和模块的集合，其提供一致的配置接口来远程管理多个 RHEL 系统。**metrics** 系统角色为本地系统配置性能分析服务，并可以选择包含要由本地系统监控的远程系统的列表。**metrics** 系统角色可让您使用 **pcp** 来监控系统的性能，而无需单独配置 **pcp**，因为 **pcp** 的设置和部署是已由 playbook 处理。

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件
- **/usr/share/doc/rhel-system-roles/metrics/** directory

4.3. 使用 METRICS RHEL 系统角色以可视化方式监控本地系统

此流程描述了如何使用 **metrics** RHEL 系统角色来监控您的本地系统，同时通过 **Grafana** 提供数据可视化。

前提条件

- 您已准备好控制节点和受管节点

- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。
- **localhost** 在控制节点上的清单文件中被配置：

```
localhost ansible_connection=local
```

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
---
- name: Manage metrics
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_graph_service: yes
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

因为 **metrics_graph_service** 布尔值被设置为 **value="yes"**，所以 **Grafana** 会被自动安装，并使用 **pcp** 置备为数据源。因为 **metrics_manage_firewall** 和 **metrics_manage_selinux** 都被设为了 **true**，所以 metrics 角色使用 **firewall** 和 **selinux** 系统角色来管理 metrics 角色使用的端口。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook：

```
$ ansible-playbook ~/playbook.yml
```

验证

- 要查看机器上收集的指标的视图，请访问 **grafanaweb** 界面，如 访问 Grafana web UI 中所述。

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件
- **/usr/share/doc/rhel-system-roles/metrics/** directory

4.4. 使用 METRICS RHEL 系统角色设置监控其自身的独立系统

此流程描述了如何使用 **metrics** 系统角色设置一组机器来监控其自身。

前提条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

步骤

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Configure a fleet of machines to monitor themselves
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

因为 **metrics_manage_firewall** 和 **metrics_manage_selinux** 都被设为了 **true**，所以 metrics 角色使用 **firewall** 和 **selinux** 角色来管理 **metrics** 角色使用的端口。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook：

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件
- **/usr/share/doc/rhel-system-roles/metrics/** directory

4.5. 使用 METRICS RHEL 系统角色使用本地机器集中监控机器的数量

此流程描述了如何使用 **metrics** 系统角色设置本地机器来集中监控一组机器，同时通过 **grafana** 提供数据的视觉化，并通过 **redis** 提供数据的查询。

前提条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。
- **localhost** 在控制节点上的清单文件中被配置：

```
localhost ansible_connection=local
```

步骤

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
- name: Set up your local machine to centrally monitor a fleet of machines
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_graph_service: yes
    metrics_query_service: yes
    metrics_retention_days: 10
    metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
    metrics_manage_firewall: yes
    metrics_manage_selinux: yes
```

因为 **metrics_graph_service** 和 **metrics_query_service** 布尔值被设置为了 **value="yes"**，所以 **grafana** 会被自动安装，并使用**pcp**置备为数据源，并将 **pcp** 数据记录索引到 **redis** 中，允许 **pcp** 查询语言用于复杂的数据查询。因为 **metrics_manage_firewall** 和 **metrics_manage_selinux** 都被设为了 **true**，所以 **metrics** 角色使用 **firewall** 和 **selinux** 角色来管理 **metrics** 角色使用的端口。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook：

```
$ ansible-playbook ~/playbook.yml
```

验证

- 要查看机器集中收集的指标的图形表示，并查询数据，请访问 **grafana** web 界面，如 访问 Grafana Web UI 中所述。

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件
- **/usr/share/doc/rhel-system-roles/metrics/** directory

4.6. 在使用 METRICS RHEL 系统角色监控系统时设置身份验证

PCP 通过简单身份验证安全层 (SASL) 框架支持 **scram-sha-256** 验证机制。**metrics** RHEL 系统角色使用 **scram-sha-256** 身份验证机制自动化设置身份验证的步骤。这个流程描述了如何使用 **metrics** RHEL 系统角色设置身份验证。

前提条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

步骤

1. 编辑现有的 playbook 文件，如 ~/**playbook.yml**，并添加与身份验证相关的变量：

```
---
- name: Set up authentication by using the scram-sha-256 authentication mechanism
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
    metrics_username: <username>
    metrics_password: <password>
```

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook：

```
$ ansible-playbook ~/playbook.yml
```

验证

- 验证 **sasl** 配置：

```
# pminfo -f -h "pcp://managed-node-01.example.com?username=<username>"
disk.dev.read
Password: <password>
disk.dev.read
inst [0 or "sda"] value 19540
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件
- **/usr/share/doc/rhel-system-roles/metrics/** directory

4.7. 使用 METRICS RHEL 系统角色为 SQL SERVER 配置并启用指标集合

此流程描述了如何使用 **metrics** RHEL 系统角色，通过本地系统上的 **pcp** 来自动化配置，并为 Microsoft SQL Server 启用指标集合。

前提条件

前提条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。
- 您已 安装了用于 Red Hat Enterprise Linux 的 Microsoft SQL Server，并建立了一个与 SQL 服务器的可信连接。
- 您已 为用于 Red Hat Enterprise Linux 的 SQL Server 安装了 Microsoft ODBC 驱动程序 。
- **localhost** 在控制节点上的清单文件中被配置：

```
localhost ansible_connection=local
```

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Configure and enable metrics collection for Microsoft SQL Server
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_from_mssql: true
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

因为 **metrics_manage_firewall** 和 **metrics_manage_selinux** 都被设为了 **true**，所以 **metrics** 角色使用 **firewall** 和 **selinux** 角色来管理 **metrics** 角色使用的端口。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook：

```
$ ansible-playbook ~/playbook.yml
```

验证

- 使用 **pcp** 命令来验证 SQL Server PMDA 代理 (mssql) 是否已加载并在运行：

```
# pcp
platform: Linux sqlserver.example.com 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23
UTC 2019 x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
     pmcd: Version 5.0.2-1, 12 agents, 4 clients
```

```
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
pmlogger: primary logger: /var/log/pcp/pmlogger/sqlserver.example.com/20200326.16.31
    pmie: primary engine: /var/log/pcp/pmie/sqlserver.example.com/pmie.log
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件
- **/usr/share/doc/rhel-system-roles/metrics/** directory

第 5 章 设置 PCP

Performance Co-Pilot (PCP) 是用于监控、视觉化、存储和分析系统级性能测量的工具、服务和库集。

5.1. PCP 概述

您可以使用 Python、Perl、C++ 和 C 接口添加性能指标。分析工具可以直接使用 Python、C++、C 客户端 API，并通过 JSON 界面探索所有可用的性能数据。

您可以通过将实时结果与存档数据进行比较来分析数据模型。

PCP 的功能：

- 轻量级分布式架构，在复杂的系统集中分析过程中非常有用。
- 它允许监控和管理实时数据。
- 它允许记录和检索历史数据。

PCP 包含以下组件：

- Performance Metric Collector Daemon (**pmcd**) 从已安装的性能指标域代理 (**pmda**) 收集性能数据。**PMDA** 可以单独加载或卸载在系统上，并由同一主机上的 **PMCD** 控制。
- **pminfo** 或 **pmstat** 等各种客户端工具可以检索、显示、存档和处理同一主机或网络上的此数据。
- **pcp** 软件包提供命令行工具和底层功能。
- **pcp-gui** 软件包提供了图形应用程序。执行 **dnf install pcp-gui** 命令安装 **pcp-gui** 软件包。如需更多信息，请参阅使用 PCP Charts 应用程序进行 Visual tracing PCP 日志归档。

其他资源

- **pcp (1)** 手册页
- **/usr/share/doc/pcp-doc/** directory
- PCP 分发的系统服务和工具
- Performance Co-Pilot (PCP) 文章、解决方案、教程以及红帽客户门户网站中的白皮书的索引
- PCP 工具与旧工具红帽知识库文章的并排比较
- PCP 上游文档

5.2. 安装并启用 PCP

要开始使用 PCP，请安装所有必需的软件包并启用 PCP 监控服务。

这个步骤描述了如何使用 **pcp** 软件包安装 PCP。如果要自动化 PCP 安装，请使用 **pcp-zeroconf** 软件包安装它。有关使用 **pcp-zeroconf** 安装 PCP 的更多信息，请参阅 使用 pcp-zeroconf 设置 PCP。

步骤

1. 安装 **pcp** 软件包：

-

```
# dnf install pcp
```

2. 在主机机器上启用并启动 **pmcd** 服务：

```
# systemctl enable pmcd

# systemctl start pmcd
```

验证步骤

- 验证 **pmcd** 进程是否在主机上运行：

```
# pcp

Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

其他资源

- **pmcd (1)** man page
- PCP 分发的系统服务和工具

5.3. 部署最小 PCP 设置

PCP 最小设置收集 Red Hat Enterprise Linux 的性能统计信息。设置涉及在产品系统中添加收集数据以便进一步分析所需的最小软件包数量。

您可以使用各种 PCP 工具分析生成的 **tar.gz** 文件和 **pmlogger** 输出存档，并将它们与其他性能信息源进行比较。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

1. 更新 **pmlogger** 配置：

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. 启动 **pmcd** 和 **pmlogger** 服务：

```
# systemctl start pmcd.service

# systemctl start pmlogger.service
```

3. 执行所需的操作来记录性能数据。
4. 停止 **pmcd** 和 **pmlogger** 服务：

```
# systemctl stop pmcd.service

# systemctl stop pmlogger.service
```

5. 保存输出并将其保存到基于主机名和当前日期和时间的 **tar.gz** 文件中：

```
# cd /var/log/pcp/pmlogger/

# tar -czf $(hostname).$(date +%F-%Hh%M).pcp.tar.gz $(hostname)
```

使用 PCP 工具提取此文件并分析数据。

其他资源

- **pmlogconf (1)**、**pmlogger (1)** 和 **pmcd (1)** man page
- 系统服务和使用 PCP 分发的工具

5.4. PCP 分发的系统服务和工具

Performance Co-Pilot (PCP)包括可用来测量性能的各种系统服务和工具。基本软件包 **pcp** 包括系统服务和基本工具。**pcp-system-tools**、**pcp-gui** 和 **pcp-devel** 软件包提供了其他工具。

PCP 分发的系统服务的角色

pmcd
Performance Metric Collector Daemon (PMCD) 。

pmie
性能指标对引擎.

pmlogger
性能指标日志记录器。

pmproxy
实时和历史性能指标代理、时间序列查询和 REST API 服务。

基本 PCP 软件包分发的工具

pcp
显示 Performance Co-Pilot 安装的当前状态。

pcp-vmstat
每 5 秒提供高级系统性能概述。显示有关进程、内存、分页、块 IO、 traps 和 CPU 活动的信息。

pmconfig
显示配置参数的值。

pmdiff

比较一个或两个存档（给定时间窗内）中每个指标的平均值，而在搜索性能回归时可能会感兴趣的更改。

pmdumplog

显示 Performance Co-Pilot 归档文件中的控制、元数据、索引和状态信息。

pmfind

在网络上查找 PCP 服务。

pmie

定期评估一组算术、逻辑和规则表达式的 inference 引擎。指标可以从 live 系统或 Performance Co-Pilot 归档文件收集。

pmieconf

显示或设置可配置的 **pmie** 变量。

pmiectl

管理 **pmie** 的非主要实例。

pminfo

显示性能指标的相关信息。指标可以从 live 系统或 Performance Co-Pilot 归档文件收集。

pmlc

交互式配置活动的 **pmlogger** 实例。

pmlogcheck

在 Performance Co-Pilot 归档文件中标识无效数据。

pmlogconf

创建和修改 **pmlogger** 配置文件。

pmlogctl

管理 **pmlogger** 的非主要实例。

pmloglabel

验证、修改或修复 Performance Co-Pilot 归档文件的标签。

pmlogsummary

计算 Performance Co-Pilot 归档文件中存储性能指标的统计信息。

pmprobe

决定性能指标的可用性。

pmsocks

允许通过防火墙访问 Performance Co-Pilot 主机。

pmstat

定期显示系统性能的简短摘要。

pmstore

修改性能指标的值。

pmtrace

提供到 trace PMDA 的命令行界面。

pmval

显示性能指标的当前值。

单独安装的 pcp-system-tools 软件包分发的工具

pcp-atop

从性能角度显示最重要的硬件资源的系统级别：CPU、内存、磁盘和网络。

pcp-atopsar

在各种系统资源使用率上生成系统级活动报告。报告从之前使用 **pmlogger** 或 **pcp-atop** 的 **-w** 选项记录的原始日志文件生成。

pcp-dmcache

显示有关配置的设备映射缓存目标的信息，例如：设备 IOP、缓存和元数据设备利用率，以及在每次缓存设备的读写率和比率。

pcp-dstat

一次显示一个系统的指标。要显示多个系统的指标，请使用 **--host** 选项。

pcp-free

报告系统中的空闲和已用内存。

pcp-htop

以类似于 **top** 命令的方式显示系统上运行的所有进程及其命令行参数，但允许您使用鼠标进行垂直和水平滚动。您还可以以树形格式查看进程，并同时对多个进程选择和实施。

pcp-ipcs

显示有关调用进程具有读取访问权限的进程间通信(IPC)工具的信息。

pcp-mpstat

报告 CPU 和与中断相关的统计信息。

pcp-numastat

显示内核内存分配器的 NUMA 分配统计信息。

pcp-pidstat

显示系统上运行的各个任务或进程的信息，如 CPU 百分比、内存和堆栈使用率、调度和优先级。报告默认情况下本地主机的实时数据。

pcp-shping

pmdashping 性能指标域代理(PMDA)导出的有关 shell-ping 服务指标的样本和报告。

pcp-ss

显示 **pmdasockets** PMDA 收集的套接字统计信息。

pcp-tapestat

报告磁带设备的 I/O 统计信息。

pcp-uptime

显示系统正在运行的时长，当前登录的用户数量，以及过去 1、5 和 15 分钟的系统负载平均值。

pcp-verify

检查 Performance Co-Pilot 收集器安装的各个方面，并报告其是否为某些操作模式进行了正确配置。

pmiostat

报告 SCSI 设备（默认）或设备映射器设备的 I/O 统计信息（使用 **-x** device-mapper 选项）。

pmrep

报告选定、易于自定义、性能指标值。

单独安装的 pcp-gui 软件包分发的工具

pmchart

通过 Performance Co-Pilot 的功能来绘制性能指标值。

pmdumptext

输出从 Performance Co-Pilot 归档收集的性能指标值。

单独安装的 pcp-devel 软件包分发的工具

pmclient

使用性能指标应用程序编程接口 (PMAPI) 显示高级系统性能指标。

pmdbg

显示可用的 Performance Co-Pilot 调试控制标记及其值。

pmerr

显示可用的 Performance Co-Pilot 错误代码及其对应的错误消息。

5.5. PCP 部署架构

Performance Co-Pilot (PCP)根据 PCP 部署规模，支持多个部署架构，并提供许多选项来完成高级设置。

可用的扩展部署设置变体基于红帽推荐的部署设置、调整因素和配置选项，包括：

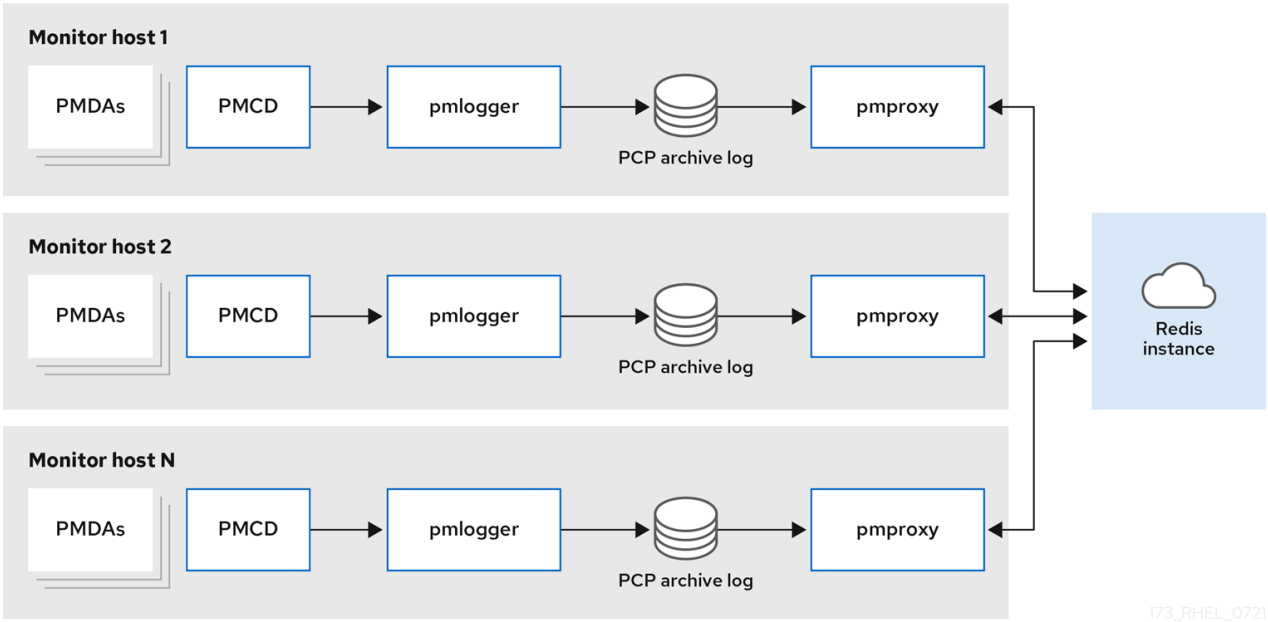
Localhost

每个服务在被监控的机器上本地运行。当您在没有配置更改的情况下启动服务时，这是默认的部署。在这种情况下无法对单个节点进行扩展。
默认情况下，Redis 的部署设置是单机 localhost。但是，Red Hat Redis 可以选择以高可用性和高度扩展的集群执行，其中数据在多个主机之间共享。另一个可行选择是在云中部署 Redis 集群，或者从云供应商中使用受管 Redis 集群。

Decentralized

localhost 和分散设置之间的唯一区别是集中式 Redis 服务。在这种模型中，主机在每个被监控的主机上执行 **pmlogger** 服务，并从本地 **pmcd** 实例检索指标。然后本地 **pmproxy** 服务将性能指标导出到中央 Redis 实例。

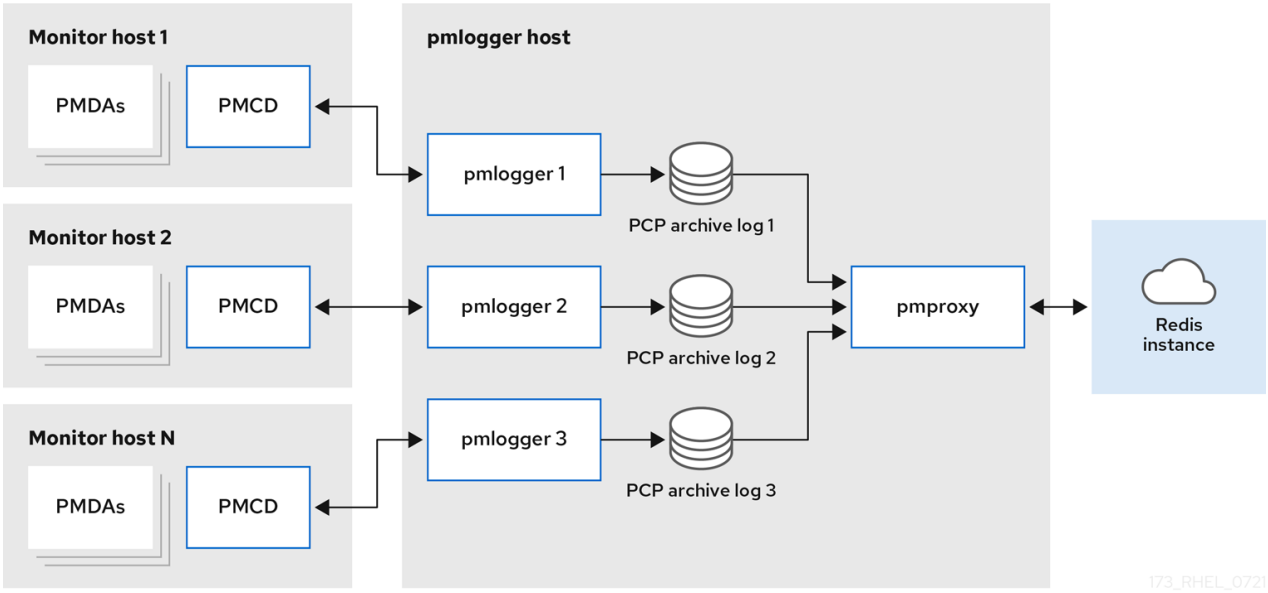
图 5.1. 分散日志记录



集中式日志记录 - pmlogger 场

当被监控主机的资源使用情况受限时，另一个部署选项是一个 **pmlogger** 场，也称为集中式日志记录。在本设置中，单个日志记录器主机执行多个 **pmlogger** 进程，各自配置为从不同的远程 **pmcd** 主机检索性能指标。集中式日志记录器主机也被配置为执行 **pmproxy** 服务，该服务发现生成的 PCP 存档日志并将指标数据加载到 Redis 实例中。

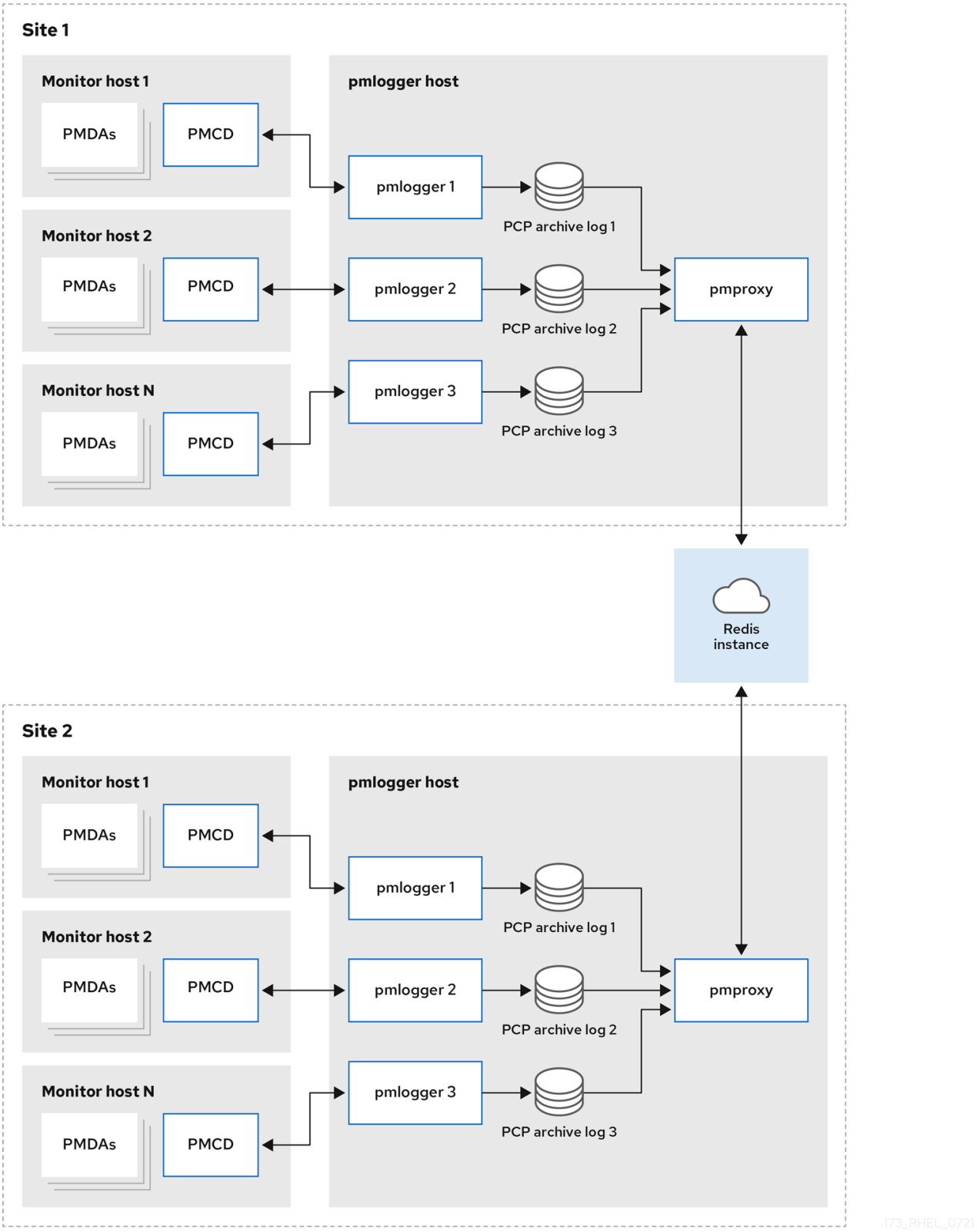
图 5.2. 集中式日志记录 - pmlogger 场



联邦 - 多个 pmlogger farms

对于大规模部署，红帽建议以联邦方式部署多个 **pmlogger** farm。例如，每个机架或数据中心一个 **pmlogger** farm。每个 **pmlogger** farm 都会将指标加载到中央 Redis 实例中。

图 5.3. 联邦 - 多个 pmlogger farms



注意

默认情况下，Redis 的部署设置是单机 localhost。但是，Red Hat Redis 可以选择以高可用性和高度扩展的集群执行，其中数据在多个主机之间共享。另一个可行选择是在云中部署 Redis 集群，或者从云供应商中使用受管 Redis 集群。

其他资源

- **pcp (1)** , **pmlogger (1)** , **pmproxy (1)** , 和 **pmcd (1)** man pages
- 推荐的部署架构

5.6. 推荐的部署架构

下表根据监控的主机数量描述了推荐的部署架构。

表 5.1. 推荐的部署架构

主机数 (N)	1-10	10-100	100-1000
pmcd 服务器	N	N	N
pmlogger 服务器	1 到 N	N/10 到 N	N/100 到 N
pmproxy 服务器	1 到 N	1 到 N	N/100 到 N
Redis 服务器	1 到 N	1 到 N/10	N/100 到 N/10
Redis 集群	否	Maybe	是
推荐的部署设置	localhost、Decentralized 或 centralized logging	Decentralized, Centralized logging 或 Federated	decentralized 或 Federated

5.7. 大小考虑因素

以下是扩展所需的大小调整因素：

远程系统大小

CPU、磁盘、网络接口和其他硬件资源的数量会影响中央日志记录主机上的每个 **pmlogger** 收集的数据量。

日志记录的指标数据

日志记录的指标的数量和类型是重要的角色。特别是，**per-process proc.*** 指标需要大量磁盘空间，例如，标准 **pcp-zeroconf** 设置、10s 日志记录间隔、11 MB、没有 proc 指标和 155 MB 的 proc 指标 - 可获得 10 倍。此外，每个指标的实例数量，如 CPU、块设备和网络接口的数量也会影响所需的存储容量。

日志记录间隔

指标的日志记录频率，会影响存储要求。预期的每日 PCP 归档文件大小会为每个 **pmlogger** 实例写入到 **pmlogger.log** 文件。这些值未压缩估算。由于 PCP 归档的压缩非常大，大约 10:1，因此可以为特定站点确定实际的长期磁盘空间要求。

pmlogrewrite

在每个 PCP 升级后，将执行 **pmlogrewrite** 工具，并重写旧的归档（如果之前版本中的指标元数据有变化）和 PCP 的新版本。这个过程持续时间使用存储的存档数扩展线性。

其他资源

- **pmlogrewrite (1)** 和 **pmlogger (1)** man page

5.8. PCP 扩展的配置选项

以下是扩展所需的配置选项：

sysctl 和 rlimit 设置

当启用归档发现时，对于每个 **pmlogger**，**pmproxy** 都需要 4 个描述符，用于监控或注销，以及服务日志和 **pmproxy** 客户端套接字的额外文件描述符（如果有）。每个 **pmlogger** 进程在远程 **pmcd** 套接字、存档文件、服务日志等中使用大约 20 个文件描述符。总的来说，这可以超过运行约 200 个 **pmlogger** 进程的系统上的默认 1024 软限制。**pcp-5.3.0** 及之后的版本中的 **pmproxy** 服务会自动将软限制增加到硬限制。在 PCP 的早期版本中，如果要部署大量 **pmlogger** 进程，则需要调优；这可以通过增加 **pmlogger** 的软或硬限制来实现。如需更多信息，请参阅 如何为 systemd 运行的服务设置限制 (ulimit) 。

本地归档

pmlogger 服务将本地和远程 **pmcds** 的指标存储在 **/var/log/pcp/pmlogger/** 目录中。要控制本地系统的日志间隔，请更新 **/etc/pcp/pmlogger/control.d/*configfile*文件**，并在参数中添加 **-t *X***，其中 *X* 是日志间隔（以秒为单位）。要配置应该记录哪些指标，请执行 **pmlogconf /var/lib/pcp/config/pmlogger/config.*clienthostname***。此命令使用一组默认指标来部署配置文件，可选择性地进行进一步自定义。要指定保留设置（指定何时清除旧的 PCP 存档），更新 **/etc/sysconfig/pmlogger_timers** 文件指定 **PMLOGGER_DAILY_PARAMS="-E -k *X*"**，其中 *X* 是保留 PCP 归档的天数。

Redis

pmproxy 服务将日志记录的指标从 **pmlogger** 发送到 Redis 实例。以下是两个选项，用于指定 **/etc/pcp/pmproxy/pmproxy.conf** 配置文件中的保留设置：

- **stream.expire** 指定应删除过时指标时的持续时间，即在指定时间内没有更新的指标，以秒为单位。
- **stream.maxlen** 指定每个主机的一个指标值的最大指标值数。此设置应是保留的时间除以日志间隔，例如如果保留时间为 14 天日志间隔是 60s，则设置为 20160 (60*60*24*14/60)

其他资源

- **pmproxy (1)** , **pmlogger (1)** , 和 **sysctl (8)** man pages

5.9. 示例：分析集中式日志记录部署

在集中式日志记录设置中收集以下结果（也称为 pmlogger 场部署），其默认 **pcp-zeroconf 5.3.0** 安装，其中每个远程主机都是在有 64 个 CPU 内核、376 GB RAM 的服务器上运行 **pmcd** 的相同容器实例。

日志记录间隔为 10s，不包含远程节点的 proc 指标，内存值则引用 Resident Set Size (RSS) 值。

表 5.2. 10s 日志间隔的详细利用率统计

主机数量	10	50
PCP 每天归档存储	91 MB	522 MB
pmlogger Memory	160 MB	580 MB

主机数量	10	50
每天 **pmlogger** Network (In)	2 MB	9 MB
pmproxy Memory	1.4 GB	6.3 GB
每天的 redis 内存	2.6 GB	12 GB

表 5.3. 根据被监控的主机提供 60 个日志记录间隔的资源

主机数量	10	50	100
PCP 每天归档存储	20 MB	120 MB	271 MB
pmlogger Memory	104 MB	524 MB	1049 MB
每天 **pmlogger** Network (In)	0.38 MB	1.75 MB	3.48 MB
pmproxy Memory	2.67 GB	5.5GB	9 GB
每天的 redis 内存	0.54 GB	2.65 GB	5.3 GB

注意

pmproxy 队列 Redis 请求，并使用 Redis pipelining 来加快 Redis 查询。这可能导致大量内存使用。有关此问题的故障排除，请参阅对高内存的使用进行故障排除。

5.10. 示例：分析联合设置部署

以下结果在联合设置中观察，也称为多个 **pmlogger** farm，由三个集中式日志记录（**pmlogger** farm）设置组成，每个 **pmlogger** farm 都监控 100 个远程主机，总计为 300 个主机。

pmlogger 场的设置与下面提到的配置一样

60s 日志记录间隔的 示例：分析集中式日志记录部署 ，但 Redis 服务器在集群模式下运行。

表 5.4. 根据联合主机进行 60s 日志记录间隔使用的资源

PCP 每天归档存储	**pmlogger** Memory	每天网络 (In/Out)	**pmproxy** Memory	每天的 redis 内存
277 MB	1058 MB	15.6 MB / 12.3 MB	6-8 GB	5.5 GB

此处，所有值都是每个主机。网络带宽较高，因为 Redis 集群的节点间通信。

5.11. 建立安全的 PCP 连接

您可以配置 PCP 收集器和监控组件，以参与安全的 PCP 协议交换。

5.11.1. 确保 PCP 连接安全

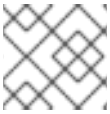
您可以在 Performance Co-Pilot (PCP)收集器和监控组件之间建立安全连接。PCP 收集器组件是 PCP 的一部分，它从不同的源收集和提取性能数据。PCP 监控组件是 PCP 的一部分，它显示从安装了 PCP 收集器组件的主机或存档中收集来的数据。在这些组件之间建立安全连接有助于防止未授权方访问或修改正在收集和监控的数据。

所有与性能指标收集器守护进程(**pmcd**)的连接都是使用基于 TCP/IP 的 PCP 协议建立的。协议代理和 PCP REST API 由 **pmproxy** 守护进程提供 - REST API 可通过 HTTPS 访问，以确保安全连接。

pmcd 和 **pmproxy** 守护进程能够在单一端口上同时进行 TLS 和非 TLS 通信。 **pmcd** 的默认端口是 44321，44322 用于 **pmproxy**。这意味着您不必为您的 PCP 收集器系统在 TLS 或非 TLS 通信之间进行选择，您可以同时使用它们。

5.11.2. 为 PCP 收集器组件配置安全连接

所有 PCP 收集器系统必须有有效的证书，以便参与 PCP 协议交换。



注意

从 TLS 的角度来看，**pmproxy** 守护进程同时作为客户端和服务器运行。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- 私有客户端密钥存储在 **/etc/pcp/tls/client.key** 文件中。如果您使用其他路径，请调整该流程的对应步骤。
  有关创建私钥和证书签名请求(CSR)的详细信息，以及如何从证书颁发机构(CA)请求证书，请参阅您的 CA 文档。
- TLS 客户端证书存储在 **/etc/pcp/tls/client.crt** 文件中。如果您使用其他路径，请调整该流程的对应步骤。
- CA 证书存储在 **/etc/pcp/tls/ca.crt** 文件中。如果您使用其他路径，请调整该流程的对应步骤。另外，对于 **pmproxy** 守护进程：
- 私有服务器密钥存储在 **/etc/pcp/tls/server.key** 文件中。如果您使用其他路径，请调整流程的相应步骤
- TLS 服务器证书存储在 **/etc/pcp/tls/server.crt** 文件中。如果您使用其他路径，请调整该流程的对应步骤。

步骤

1. 更新收集器系统上的 PCP TLS 配置文件，以使用 CA 发布的证书来建立安全连接：

```
# cat > /etc/pcp/tls.conf << END
tls-ca-cert-file = /etc/pcp/tls/ca.crt
tls-key-file = /etc/pcp/tls/server.key
tls-cert-file = /etc/pcp/tls/server.crt
```

```
tls-client-key-file = /etc/pcp/tls/client.key
tls-client-cert-file = /etc/pcp/tls/client.crt
END
```

2. 重启 PCP 收集器基础架构：

```
# systemctl restart pmcd.service
# systemctl restart pmproxy.service
```

验证

- 验证 TLS 配置：
  - 在 **pmcd** 服务上：

```
# grep 'Info:' /var/log/pcp/pmcd/pmcd.log
[Tue Feb 07 11:47:33] pmcd(6558) Info: OpenSSL 3.0.7 setup
```

  - 在 **pmproxy** 服务上：

```
# grep 'Info:' /var/log/pcp/pmproxy/pmproxy.log
[Tue Feb 07 11:44:13] pmproxy(6014) Info: OpenSSL 3.0.7 setup
```

5.11.3. 为 PCP 监控组件配置安全连接

配置 PCP 监控组件以参与安全 PCP 协议交换。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- 私有客户端密钥存储在 **~/.pcp/tls/client.key** 文件中。如果您使用其他路径，请调整该流程的对应步骤。
  有关创建私钥和证书签名请求(CSR)的详细信息，以及如何从证书颁发机构(CA)请求证书，请参阅您的 CA 文档。
- TLS 客户端证书存储在 **~/.pcp/tls/client.crt** 文件中。如果您使用其他路径，请调整该流程的对应步骤。
- CA 证书存储在 **/etc/pcp/tls/ca.crt** 文件中。如果您使用其他路径，请调整该流程的对应步骤。

步骤

1. 使用以下信息创建一个 TLS 配置文件：

```
$ home=echo ~
$ cat > ~/.pcp/tls.conf << END
tls-ca-cert-file = /etc/pcp/tls/ca.crt
tls-key-file = $home/.pcp/tls/client.key
tls-cert-file = $home/.pcp/tls/client.crt
END
```

2. 建立安全连接：

```
$ export PCP_SECURE_SOCKETS=enforce
$ export PCP_TLSCONF_PATH=~/.pcp/tls.conf
```

验证

- 验证安全连接是否已配置：

```
$ pminfo --fetch --host pcps://localhost kernel.all.load

kernel.all.load
    inst [1 or "1 minute"] value 1.26
    inst [5 or "5 minute"] value 1.29
    inst [15 or "15 minute"] value 1.28
```

5.12. 对高内存使用量进行故障排除

以下情况可能会导致内存用量：

- **pmproxy** 进程忙于处理新的 PCP 归档，且没有处理 Redis 请求和响应的备用 CPU 周期。
- Redis 节点或集群已过载，且无法在时间处理传入的请求。

pmproxy 服务守护进程使用 Redis 流并支持配置参数，这些参数是 PCP 调优参数，并影响 Redis 内存用量和密钥保留。**/etc/pcp/pmproxy/pmproxy.conf** 文件列出了 **pmproxy** 和关联的 API 的可用选项。

以下流程描述了如何对高内存使用率问题进行故障排除。

先决条件

1. 安装 **pcp-pmda-redis** 软件包：

```
# dnf install pcp-pmda-redis
```

2. 安装 redis PMDA：

```
# cd /var/lib/pcp/pmdas/redis && ./Install
```

步骤

- 要排除高内存用量的问题，请执行以下命令并观察 **inflight** 列：

```
$ pmrep :pmproxy
        backlog  inflight  reqs/s  resp/s   wait req err  resp err  changed  throttled
           byte     count   count/s  count/s  s/s  count/s   count/s  count/s   count/s
14:59:08   0        0       N/A      N/A      N/A  N/A       N/A      N/A       N/A
14:59:09   0        0       2268.9   2268.9   28   0         0        2.0       4.0
14:59:10   0        0       0.0      0.0      0    0         0        0.0       0.0
14:59:11   0        0       0.0      0.0      0    0         0        0.0       0.0
```

此列显示有多少 Redis 请求是 in-flight，这意味着它们被排队或发送，目前还没有收到回复。

数字表示以下条件之一：

- **pmproxy** 进程忙于处理新的 PCP 归档，且没有处理 Redis 请求和响应的备用 CPU 周期。
- Redis 节点或集群已过载，且无法在时间处理传入的请求。

- 要对高内存使用问题进行故障排除，请减少此场的 **pmlogger** 进程数量，再添加另一个 pmlogger 场。使用联邦 - 多个 pmlogger farm 设置。
  如果 Redis 节点使用 100% 的 CPU 延长的时间，请将其移到具有更好的性能的主机，或使用集群的 Redis 设置。

- 要查看 **pmproxy.redis.*** 指标，请使用以下命令：

```
$ pminfo -ftd pmproxy.redis
pmproxy.redis.responses.wait [wait time for responses]
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: counter  Units: microsec
    value 546028367374
pmproxy.redis.responses.error [number of error responses]
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: counter  Units: count
    value 1164
[...]
pmproxy.redis.requests.inflight.bytes [bytes allocated for inflight requests]
    Data Type: 64-bit int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: discrete  Units: byte
    value 0

pmproxy.redis.requests.inflight.total [inflight requests]
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: discrete  Units: count
    value 0
[...]
```

  要查看有多少 Redis 请求在flight 中，请参阅 **pmproxy.redis.requests.inflight.total** 指标和 **pmproxy.redis.requests.inflight.bytes** 指标来查看所有当前在flight Redis 请求中消耗的字节数。

  通常，redis 请求队列为零，但可以根据大型 pmlogger 场的使用而构建，这限制了可扩展性，并可能导致 **pmproxy** 客户端的高延迟。

- 使用 **pminfo** 命令查看有关性能指标的信息。例如，要查看 **redis.*** 指标，请使用以下命令：

```
$ pminfo -ftd redis
redis.redis_build_id [Build ID]
    Data Type: string  InDom: 24.0 0x6000000
    Semantics: discrete  Units: count
    inst [0 or "localhost:6379"] value "87e335e57cffa755"
redis.total_commands_processed [Total number of commands processed by the server]
    Data Type: 64-bit unsigned int  InDom: 24.0 0x6000000
    Semantics: counter  Units: count
    inst [0 or "localhost:6379"] value 595627069
[...]

redis.used_memory_peak [Peak memory consumed by Redis (in bytes)]
    Data Type: 32-bit unsigned int  InDom: 24.0 0x6000000
```

```
    Semantics: instant  Units: count
    inst [0 or "localhost:6379"] value 572234920
[...]
```

要查看峰值内存用量，请参阅 **redis.used_memory_peak** 指标。

其他资源

- **pmdaredis (1)**、**pmproxy (1)** 和 **pminfo (1)** man page
- PCP 部署架构

第 6 章 使用 PMLOGGER 记录性能数据

使用 PCP 工具，您可以记录性能指标值并稍后重新显示。这可让您执行改进的性能分析。

使用 **pmlogger** 工具，您可以：

- 在系统上创建所选指标的归档日志
- 指定系统中记录哪些指标以及它们的频率

6.1. 使用 PMLOGCONF 修改 PMLOGGER 配置文件

当 **pmlogger** 服务运行时，PCP 会记录主机上一组默认指标。

使用 **pmlogconf** 实用程序检查默认配置。如果 **pmlogger** 配置文件不存在，则 **pmlogconf** 会使用默认指标值创建该文件。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

1. 创建或修改 **pmlogger** 配置文件：

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. 按照 **pmlogconf** 提示启用或禁用相关性能指标组，并控制每个启用的组的日志间隔。

其他资源

- **pmlogconf (1)** 和 **pmlogger (1)** man page
- 系统服务和使用 PCP 分发的工具

6.2. 手动编辑 PMLOGGER 配置文件

要使用特定指标和给定间隔创建定制的日志配置，请手动编辑 **pmlogger** 配置文件。默认 **pmlogger** 配置文件为 **/var/lib/pcp/config/pmlogger/config.default**。配置文件指定主日志记录实例记录哪些指标。

在手动配置中，您可以：

- 记录没有列在自动配置中的指标。
- 选择自定义日志记录频率。
- 使用应用程序指标添加 PMDA。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

- 打开并编辑 **/var/lib/pcp/config/pmlogger/config.default** 文件以添加特定的指标：

```
# It is safe to make additions from here on ...
#

log mandatory on every 5 seconds {
    xfs.write
    xfs.write_bytes
    xfs.read
    xfs.read_bytes
}

log mandatory on every 10 seconds {
    xfs.allocs
    xfs.block_map
    xfs.transactions
    xfs.log

}

[access]
disallow * : all;
allow localhost : enquire;
```

其他资源

- **pmlogger (1)** man page
- 系统服务和使用 PCP 分发的工具

6.3. 启用 PMLOGGER 服务

必须启动并启用 **pmlogger** 服务，以记录本地计算机上的指标值。

这个步骤描述了如何启用 **pmlogger** 服务。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

- 启动并启用 **pmlogger** 服务：

```
# systemctl start pmlogger

# systemctl enable pmlogger
```

验证步骤

- 验证 **pmlogger** 服务是否已启用：

```
# pcp
```

```
Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents, 1 client
pmda: root pmcd proc xfs linux mmv kvm jbd2
pmlogger: primary logger: /var/log/pcp/pmlogger/workstation/20190827.15.54
```

其他资源

- **pmlogger (1)** man page
- 系统服务和使用 PCP 分发的工具
- **/var/lib/pcp/config/pmlogger/config.default** file

6.4. 为指标集合设置客户端系统

这个步骤描述了如何设置客户端系统，以便中央服务器能够从运行 PCP 的客户端收集指标。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

1. 安装 **pcp-system-tools** 软件包：

```
# dnf install pcp-system-tools
```

2. 为 **pmcd** 配置 IP 地址：

```
# echo "-i 192.168.4.62" >>/etc/pcp/pmcd/pmcd.options
```

使用客户端应侦听的 IP 地址替换 *192.168.4.62*。

默认情况下，**pmcd** 侦听 localhost。

3. 配置防火墙以永久添加公共 **zone** ：

```
# firewall-cmd --permanent --zone=public --add-port=44321/tcp
success

# firewall-cmd --reload
success
```

4. 设置 SELinux 布尔值：

```
# setsebool -P pcp_bind_all_unreserved_ports on
```

5. 启用 **pmcd** 和 **pmlogger** 服务：

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

验证步骤

- 验证 **pmcd** 是否已正确侦听配置的 IP 地址：

```
# ss -tlp | grep 44321
LISTEN   0   5     127.0.0.1:44321    0.0.0.0:*    users:(("pmcd",pid=151595,fd=6))
LISTEN   0   5  192.168.4.62:44321    0.0.0.0:*    users:(("pmcd",pid=151595,fd=0))
LISTEN   0   5        [::1]:44321       [::]:*    users:(("pmcd",pid=151595,fd=7))
```

其他资源

- **pmlogger (1)** , **firewall-cmd (1)** , **ss (8)** , 和 **setsebool (8)** man pages
- 系统服务和使用 PCP 分发的工具
- **/var/lib/pcp/config/pmlogger/config.default** file

6.5. 设置中央服务器以收集数据

这个步骤描述了如何创建中央服务器从运行 PCP 的客户端收集指标。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- 为指标集合配置了客户端。如需更多信息，请参阅为指标集合设置客户端系统。

步骤

1. 安装 **pcp-system-tools** 软件包：

```
# dnf install pcp-system-tools
```

2. 使用以下内容创建 **/etc/pcp/pmlogger/control.d/remote** 文件：

```
# DO NOT REMOVE OR EDIT THE FOLLOWING LINE
$version=1.1

192.168.4.13 n n PCP_ARCHIVE_DIR/rhel7u4a -r -T24h10m -c config.rhel7u4a
192.168.4.14 n n PCP_ARCHIVE_DIR/rhel6u10a -r -T24h10m -c config.rhel6u10a
192.168.4.62 n n PCP_ARCHIVE_DIR/rhel8u1a -r -T24h10m -c config.rhel8u1a
192.168.4.69 n n PCP_ARCHIVE_DIR/rhel9u3a -r -T24h10m -c config.rhel9u3a
```

使用客户端 IP 地址替换 *192.168.4.13*、*192.168.4.14*、*192.168.4.62* 和 *192.168.4.69*。

3. 启用 **pmcd** 和 **pmlogger** 服务：

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

验证步骤

- 确保您可以从每个目录中访问最新的归档文件：

```
# for i in /var/log/pcp/pmlogger/rhel*/*.0; do pmdumplog -L $i; done
Log Label (Log Format Version 2)
Performance metrics from host rhel6u10a.local
  commencing Mon Nov 25 21:55:04.851 2019
  ending     Mon Nov 25 22:06:04.874 2019
Archive timezone: JST-9
PID for pmlogger: 24002
Log Label (Log Format Version 2)
Performance metrics from host rhel7u4a
  commencing Tue Nov 26 06:49:24.954 2019
  ending     Tue Nov 26 07:06:24.979 2019
Archive timezone: CET-1
PID for pmlogger: 10941
[..]
```

/var/log/pcp/pmlogger/ 目录中的存档文件可用于进一步分析和显示。

其他资源

- **pmlogger (1)** man page
- 系统服务和使用 PCP 分发的工具
- **/var/lib/pcp/config/pmlogger/config.default** file

6.6. SYSTEMD 单元和 PMLOGGER

当您部署 **pmlogger** 服务时，可以作为一个单一主机监控自身或 **pmlogger** farm，且单个主机从多个远程主机收集指标，则代表自动部署几个关联的 **systemd** 服务和计时器单元。这些服务和计时器提供了常规检查，以确保 **pmlogger** 实例正在运行，重启任何缺少的实例，并执行存档管理，如文件压缩。

pmlogger 通常部署的检查和内务服务有：

pmlogger_daily.service

默认情况下，在午夜后马上运行，以聚合、压缩和轮转一个或多个 PCP 存档集合。另外，剔除比限制旧的存档，默认为 2 周。由 **pmlogger_daily.timer** 单元触发，**pmlogger.service** 单元需要该单元。

pmlogger_check

执行半小时检查 **pmlogger** 实例是否正在运行。重启任何缺少的实例并执行任何所需的压缩任务。由 **pmlogger_check.timer** 单元触发，**pmlogger.service** 单元需要该单元。

pmlogger_farm_check

检查所有配置的 **pmlogger** 实例的状态。重启任何缺少的实例。将所有非主实例迁移到 **pmlogger_farm** 服务。由 **pmlogger_farm_check.timer** 触发，**pmlogger_farm.service** 单元需要它，而 **pmlogger.service** 单元需要这个单元。

这些服务通过一系列正依赖项进行管理，这意味着在激活主 **pmlogger** 实例时都启用了这些服务。请注意，虽然 **pmlogger_daily.service** 默认被禁用，但 **pmlogger_daily.timer** 通过 **pmlogger.service** 的依赖项处于活跃状态时，将触发 **pmlogger_daily.service** 运行。

pmlogger_daily 也与 **pmlogrewrite** 集成，以便在合并前自动重写存档。这有助于确保元数据一致性将更改的生产环境和 PMDA。例如，如果在日志记录间隔期间更新一个监控主机上的 **pmcd**，则可能会更新主机上一些指标的语义，从而使新存档与来自该主机记录的存档不兼容。如需更多信息，请参阅 **pmlogrewrite (1)** man page。

管理 pmlogger触发的 systemd 服务

您可以为 **pmlogger** 实例收集的数据创建一个自动自定义归档管理系统。这可以通过控制文件来完成。这些控制文件是：

- 对于主 **pmlogger** 实例：
  - **etc/pcp/pmlogger/control**
  - **/etc/pcp/pmlogger/control.d/local**
- 对于远程主机：
  - **/etc/pcp/pmlogger/control.d/remote**
    将 *remote* 替换为您的所需文件名。

    注意

    主 **pmlogger** 实例必须与它连接到的 **pmcd** 在同一主机上运行。如果一个中央主机正在收集数据到远程主机上运行的 **pmcd** 实例，则不需要主实例。

文件应包含要记录的每个主机的一行。自动创建的主日志记录器实例的默认格式类似如下：

```
# === LOGGER CONTROL SPECIFICATIONS ===
#
#Host     P?  S?   directory      args

# local primary logger
LOCALHOSTNAME    y   n    PCP_ARCHIVE_DIR/LOCALHOSTNAME    -r -T24h10m -c
config.default -v 100Mb
```

这些字段包括：

主机

要记录的主机的名称

P?

代表"主要"此字段指示主机是主日志记录器实例 y，或不是 n。您配置中的所有文件只能有一个主日志记录器，它必须与它连接到的 pmcd 在同一个主机上运行。

S?

代表"Socks?"此字段指示此日志记录器实例是否需要使用 SOCKS 协议来通过防火墙、y 或 n 连接到 pmcd。

目录

与此行关联的所有存档都会在此目录中创建。

args

传递给 pmlogger 的参数。

args 字段的默认值有：

-r

报告存档大小和增长率。

T24h10m

指定当天结束日志的时间。这通常是 pmlogger_daily.service 运行时的时间。默认值为 24h10m，表示日志记录应当在最新的 24 小时和 10 分钟后结束。

-c config.default

指定要使用的配置文件。这实际上定义了要记录什么指标。

-v 100Mb

指定将一个数据卷被填充并创建了另一个数据的大小。切换到新存档后，之前记录的存档将由 pmlogger_daily 或 pmlogger_check 压缩。

其他资源

- pmlogger (1) man page
- pmlogger_daily(1) man page
- pmlogger_check(1) man page

- pmlogger.control(5) man page
- pmlogrewrite (1) man page

6.7. 使用 PMREP 重现 PCP 日志存档

记录指标数据后，您可以重新执行 PCP 日志存档。要将日志导出到文本文件并将其导入到电子表格中，请使用 pcp2csv、pcp2xml、pmrep 或 pmlogsummary 等。

使用 pmrep 工具，您可以：

- 查看日志文件
- 解析所选 PCP 日志存档，并将值导出到 ASCII 表中
- 通过在命令行中指定单个指标，从日志中提取整个存档日志或只从日志中选择指标值

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- pmlogger 服务已启用。如需更多信息，请参阅启用 pmlogger 服务。
- 安装 pcp-system-tools 软件包：

```
# dnf install pcp-gui
```

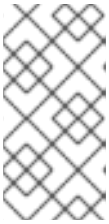
步骤

- 显示指标上的数据：

```
$ pmrep --start @3:00am --archive 20211128 --interval 5seconds --samples 10 --output csv
```

```
disk.dev.write
Time,"disk.dev.write-sda","disk.dev.write-sdb"
2021-11-28 03:00:00,,
2021-11-28 03:00:05,4.000,5.200
2021-11-28 03:00:10,1.600,7.600
2021-11-28 03:00:15,0.800,7.100
2021-11-28 03:00:20,16.600,8.400
2021-11-28 03:00:25,21.400,7.200
2021-11-28 03:00:30,21.200,6.800
2021-11-28 03:00:35,21.000,27.600
2021-11-28 03:00:40,12.400,33.800
2021-11-28 03:00:45,9.800,20.600
```

上述示例以逗号分隔值格式显示存档中以 *5 秒* 间隔收集的 disk.dev.write 指标中的数据。



注意

将此示例中的 20211128 替换为包含您要显示数据的 pmlogger 存档的文件名。

其他资源

- pmlogger (1)、pmrep (1) 和 pmlogsummary (1) man page
- PCP 分发的系统服务和工具

6.8. 启用 PCP 版本 3 归档

Performance Co-Pilot (PCP)归档存储单个主机记录的 PCP 指标的历史值，并支持回顾性性能分析。PCP 归档包含线下或站下分析所需的所有重要指标数据和元数据。这些存档可被大多数 PCP 客户端工具读取，或者由 pmdumplog 工具原始转储。

从 PCP 6.0 开始，除了版本 2 归档外，版本 3 归档也被支持。版本 2 存档保留默认值，除了得到 RHEL 9.2 及之后版本的长期支持的版本 3 之外，出于向后兼容目的，还会继续得到长期支持。

与版本 2 相比，使用 PCP 版本 3 归档提供了以下好处：

- 支持实例域更改增量

- Y2038-safe 时间戳
- 纳秒精度时间戳
- 任意时区支持
- 用于大于 2GB 的单个卷的 64 位文件偏移

前提条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

流程

1. 在您选择的文本编辑器中打开 /etc/pcp.conf 文件，并设置 PCP 归档版本：

```
PCP_ARCHIVE_VERSION=3
```

2. 重启 pmlogger 服务以应用您的配置更改：

```
# systemctl restart pmlogger.service
```

3. 使用您的新配置创建一个新的 PCP 存档日志。如需更多信息，请参阅使用 pmlogger 的日志记录性能数据。

验证

- 验证使用新配置创建的归档的版本：

```
# pmloglabel -l /var/log/pcp/pmlogger/20230208
Log Label (Log Format Version 3)
Performance metrics from host host1
        commencing Wed Feb   08 00:11:09.396 2023
        ending     Thu  Feb   07 00:13:54.347 2023
```

其他资源

- logarchive(5) 手册页
- pmlogger (1) man page
- 使用 pmlogger 记录性能数据

第 7 章 使用 PERFORMANCE CO-PILOT 监控性能

Performance Co-Pilot (PCP) 是用于监控、视觉化、存储和分析系统级性能测量的工具、服务和库集。

作为系统管理员，您可以使用 Red Hat Enterprise Linux 9 中的 PCP 应用程序监控系统性能。

7.1. 使用 PMDA-POSTFIX 监控 POSTFIX

这个步骤描述了如何使用 pmda- postfix 监控 postfix 邮件服务器的性能指标。它有助于检查每秒接收多少电子邮件。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- pmlogger 服务已启用。如需更多信息，请参阅启用 pmlogger 服务。

步骤

1. 安装以下软件包：

   a. 安装 pcp-system-tools ：

   ```
   # dnf install pcp-system-tools
   ```

   b. 安装 pmda-postfix 软件包以监控 postfix ：

   ```
   # dnf install pcp-pmda-postfix postfix
   ```

   c. 安装日志记录守护进程：

   ```
   # dnf install rsyslog
   ```

d. **安装邮件客户端进行测试：**

```
# dnf install mutt
```

2. **启用 postfix 和 rsyslog 服务：**

```
# systemctl enable postfix rsyslog
# systemctl restart postfix rsyslog
```

3. **启用 SELinux 布尔值，以便 pmda-postfix 可以访问所需的日志文件：**

```
# setsebool -P pcp_read_generic_logs=on
```

4. **安装 PMDA ：**

```
# cd /var/lib/pcp/pmdas/postfix/

# ./Install

Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Waiting for pmcd to terminate ...
Starting pmcd ...
Check postfix metrics have appeared ... 7 metrics and 58 values
```

验证步骤

- **验证 pmda-postfix 操作：**

```
echo testmail | mutt root
```

- **验证可用指标：**

```
# pminfo postfix

postfix.received
postfix.sent
postfix.queues.incoming
postfix.queues.maildrop
```

```
postfix.queues.hold
postfix.queues.deferred
postfix.queues.active
```

其他资源

- **rsyslogd (8) , postfix (1) , 和 setsebool (8) man pages**
- **系统服务和使用 PCP 分发的工具**

7.2. 使用 PCP CHARTS 应用程序可视化追踪 PCP 日志存档

记录指标数据后，您可以作为图形重新执行 PCP 日志存档。指标来源于一个或多个实时主机，可通过替代选项将 PCP 日志存档中的指标数据用作历史数据的来源。要自定义 PCP 图表 应用程序接口来显示性能指标中的数据，您可以使用行图表、栏图或利用率图形。

使用 PCP Charts 应用程序，您可以：

- 重播 PCP 图表 应用程序中的数据，并使用图形来视觉化重新内省数据以及系统的实时数据。
- 将性能指标值图表到图表中。
- 同时显示多个 chart。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- 使用 pmlogger 记录性能数据。如需更多信息，请参阅使用 pmlogger 的日志记录性能数据。
- 安装 pcp-gui 软件包：

```
# dnf install pcp-gui
```

步骤

1. 从命令行启动 PCP Charts 应用程序：

```
# pmchart
```

图 7.1. PCP Charts 应用程序

pmtime 服务器设置位于底部。可以使用 start 和 pause 按钮控制：

- PCP 轮询指标数据的时间间隔
- 历史数据指标的日期和时间

2. 点 File 然后点 New Chart，通过指定主机名或地址来选择来自本地机器和远程机器的指标。高级配置选项包括手动设置图表值的功能，以及手动选择图表颜色。

3. 记录在 PCP Charts 应用程序中创建的视图：

以下是获取镜像或记录 PCP Charts 应用程序中创建的视图的选项：

- 点 File，然后点 Export 以保存当前视图的镜像。

- 点 Record，然后 Start 启动记录。点 Record，然后 Stop 停止记录。停止记录后，会存档记录的指标，以便稍后查看。

4. 可选：在 PCP Charts 应用程序中，主配置文件称为 view，允许保存与一个或多个 chart 关联的元数据。此元数据描述了所有图表，包括所使用的指标和图表列。通过点 File 保存自定义 视图 配置，然后保存 View，稍后载入 视图 配置。

以下 PCP 图表 应用程序视图配置文件示例描述了一个堆栈图图，显示了读取和写入到给定 XFS 文件系统 loop1 的字节总数：

```
#kmchart
version 1

chart title "Filesystem Throughput /loop1" style stacking antialiasing off
    plot legend "Read rate"   metric xfs.read_bytes   instance  "loop1"
    plot legend "Write rate"  metric xfs.write_bytes  instance  "loop1"
```

其他资源

- pmchart (1) 和 pmtime (1) man page
- 系统服务和使用 PCP 分发的工具

7.3. 使用 PCP 从 SQL 服务器收集数据

SQL Server 代理在 Performance Co-Pilot (PCP) 中提供，它可帮助您监控和分析数据库性能问题。

这个步骤描述了如何通过系统中的 pcp 为 Microsoft SQL Server 收集数据。

前提条件

- 您已安装了用于 Red Hat Enterprise Linux 的 Microsoft SQL Server，并建立了与 SQL 服务器的"信任"连接。
- 您已为 Red Hat Enterprise Linux 安装了用于 SQL Server 的 Microsoft ODBC 驱动程序。

流程

1. **安装 PCP：**

```
# dnf install pcp-zeroconf
```

2. **安装 pyodbc 驱动程序所需的软件包：**

```
# dnf install python3-pyodbc
```

3. **安装 mssql 代理：**

   a. 为 **PCP 安装 Microsoft SQL Server 域名代理：**

   ```
   # dnf install pcp-pmda-mssql
   ```

   b. 编辑 **/etc/pcp/mssql/mssql.conf 文件，为 mssql 代理配置 SQL 服务器帐户的用户名和密码。请确定您配置的帐户具有性能数据的访问权限。**

   ```
   username: user_name
   password: user_password
   ```

   使用这个帐户的 SQL Server 帐户和 *user_password* 替换 *user_name*。

4. **安装代理：**

```
# cd /var/lib/pcp/pmdas/mssql
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check mssql metrics have appeared ... 168 metrics and 598 values
[...]
```

验证步骤

- **使用 pcp 命令，验证 SQL Server PMDA (mssql) 是否已加载并在运行：**

```
$ pcp
Performance Co-Pilot configuration on rhel.local:
```

```
platform: Linux rhel.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC 2019
x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
    pmcd: Version 5.0.2-1, 12 agents, 4 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
 pmlogger: primary logger: /var/log/pcp/pmlogger/rhel.local/20200326.16.31
    pmie: primary engine: /var/log/pcp/pmie/rhel.local/pmie.log
```

- 查看 PCP 可以从 SQL Server 收集的指标的完整列表：

```
# pminfo mssql
```

- 查看指标列表后，您可以报告事务的速度。例如，要报告每秒总事务数，超过 5 秒时间窗：

```
# pmval -t 1 -T 5 mssql.databases.transactions
```

- 使用 pmchart 命令查看系统中的这些指标的图形图表。如需更多信息，请参阅使用 PCP Charts 应用程序进行 Visual tracing PCP 日志归档。

其他资源

- pcp (1) , pminfo (1) , pmval (1) , pmchart (1) , and pmdamssql (1) man pages
- 带有 RHEL 8.2 Red Hat Developers Blog 的 Microsoft SQL Server 的 Performance Co-Pilot

7.4. 从 SADC 归档生成 PCP 归档

您可以使用 sysstat 软件包提供的 sadf 工具来生成来自原生 sadc 归档的 PCP 存档。

先决条件

- 已创建了 sadc 存档：

```
# /usr/lib64/sa/sadc 1 5 -
```

在本例中，sadc 是抽样系统数据 1 时间，间隔为 5 秒。outfile 指定为 -，它导致 sadc 将数据写入标准系统活动每日数据文件。此文件名为 saDD，默认情况下位于 /var/log/sa 目录中。

步骤

- 从 sadc 归档生成 PCP 存档：

```
# sadf -l -O pcparchive=/tmp/recording -2
```

在本例中，使用 -2 选项会导致 sadf 从 sadc 归档（2 天）中生成 PCP 存档。

验证步骤

您可以使用 PCP 命令检查和分析 sadc 存档生成的 PCP 存档，正如一个原生 PCP 存档一样。例如：

- 要显示 PCP 存档中从 sadc 归档生成的指标列表，请运行：

```
$ pminfo --archive /tmp/recording
Disk.dev.avactive
Disk.dev.read
Disk.dev.write
Disk.dev.blkread
[...]
```

- 要显示 PCP 归档的归档和主机名的时间范围，请运行：

```
$ pmdumplog --label /tmp/recording
Log Label (Log Format Version 2)
Performance metrics from host shard
        commencing Tue Jul 20 00:10:30.642477 2021
        ending     Wed Jul 21 00:10:30.222176 2021
```

- 要将性能指标值绘制成图形，请运行：

```
$ pmchart --archive /tmp/recording
```

第 8 章 使用 PCP 对 XFS 的性能分析

XFS PMDA 作为 pcp 软件包的一部分提供，并在安装过程中默认启用。它用于在 Performance Co-Pilot (PCP) 中收集 XFS 文件系统的性能指标数据。

您可以使用 PCP 分析 XFS 文件系统的性能。

8.1. 手动安装 XFS PMDA

如果 pcp 配置输出中没有列出 XFS PMDA，请手动安装 PMDA 代理。

这个步骤描述了如何手动安装 PMDA 代理。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

1. 进入 xfs 目录：

```
# cd /var/lib/pcp/pmdas/xfs/
```

2. 手动安装 XFS PMDA：

```
xfs]# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check xfs metrics have appeared ... 387 metrics and 387 values
```

验证步骤

- 验证 pmcd 进程是否在主机上运行，且在配置中列出 XFS PMDA：

```
# pcp
```

```
Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

其他资源

- **pmcd (1) man page**
- 系统服务和使用 PCP 分发的工具

8.2. 使用 PMINFO 检查 XFS 性能指标

PCP 启用 XFS PMDA 以允许每个挂载的 XFS 文件系统报告特定的 XFS 指标。这样可以更加轻松地查明特定挂载的文件系统问题并评估性能。

pminfo 命令为每个挂载的 XFS 文件系统提供每个设备 XFS 指标。

此流程显示 XFS PMDA 提供所有可用指标的列表。

先决条件

- **已安装 PCP。如需更多信息，请参阅安装并启用 PCP。**

步骤

- **显示 XFS PMDA 提供的所有可用指标列表：**

```
# pminfo xfs
```

- **显示各个指标的信息。以下示例使用 pminfo 工具检查特定的 XFS 读和写指标：**
  - **显示 xfs.write_bytes 指标的简短描述：**

```
# pminfo --oneline xfs.write_bytes

xfs.write_bytes [number of bytes written in XFS file system write operations]
```

- **显示 xfs.read_bytes 指标的长描述：**

```
# pminfo --helptext xfs.read_bytes

xfs.read_bytes
Help:
This is the number of bytes read via read(2) system calls to files in
XFS file systems. It can be used in conjunction with the read_calls
count to calculate the average size of the read operations to file in
XFS file systems.
```

- **获取 xfs.read_bytes 指标的当前性能值：**

```
# pminfo --fetch xfs.read_bytes

xfs.read_bytes
    value 4891346238
```

- **使用 pminfo 获取每个设备 XFS 指标：**

```
# pminfo --fetch --oneline xfs.perdev.read xfs.perdev.write

xfs.perdev.read [number of XFS file system read operations]
inst [0 or "loop1"] value 0
inst [0 or "loop2"] value 0

xfs.perdev.write [number of XFS file system write operations]
inst [0 or "loop1"] value 86
inst [0 or "loop2"] value 0
```

其他资源

- **pminfo (1) man page**
- **XFS 的 PCP 指标组**

- 每个设备 PCP 指标组用于 XFS

8.3. 使用 PMSTORE 重置 XFS 性能指标

使用 PCP，您可以修改特定指标的值，特别是当指标充当控制变量时，如 xfs.control.reset 指标。要修改指标值，请使用 pmstore 工具。

这个步骤描述了如何使用 pmstore 工具重置 XFS 指标。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

步骤

1. 显示指标的值：

```
$ pminfo -f xfs.write

xfs.write
    value 325262
```

2. 重置所有 XFS 指标：

```
# pmstore xfs.control.reset 1

xfs.control.reset old value=0 new value=1
```

验证步骤

- 在重置指标后查看信息：

```
$ pminfo --fetch xfs.write

xfs.write
    value 0
```

其他资源

- pmstore (1) 和 pminfo (1) man page
- 系统服务和使用 PCP 分发的工具
- XFS 的 PCP 指标组

8.4. XFS 的 PCP 指标组

下表描述了 XFS 可用的 PCP 指标组。

表 8.1. XFS 的指标组

指标组	提供的指标
xfs.*	常规 XFS 指标，包括读取和写入操作计数、读取和写入字节计数。另外，索引节点的次数也会刷新、集群集群和集群失败数的计数器。
xfs.allocs.* **xfs.alloc_btree.***	有关文件系统中对象分配的指标范围，其中包括扩展的数量和块创建/自由。分配树查找和比较，以及扩展从 btree 创建和删除记录。
xfs.block_map.* **xfs.bmap_btree.***	指标包括块映射的读/写入和块删除次数，用于插入、删除和查找的扩展列表操作。另外，还可从 blockmap 进行比较、查找、插入和删除操作的操作计数器。
xfs.dir_ops.*	XFS 文件系统的目录操作计数器，用于创建、删除条目，计数为"getdent"操作。
xfs.transactions.*	meta-data 事务的数量的计数器包括同步和异步事务的数量以及空事务的数量。
xfs.inode_ops.*	操作系统在索引节点缓存中查找带有不同结果的 XFS 索引节点的次数计数器。这些计数的缓存命中，缓存未命中等。
xfs.log.* **xfs.log_tail.***	通过 XFS 文件 sytems 写入的日志缓冲区数的计数器包括写入磁盘的块数量。还提供日志清除和固定数量的指标。
XFS.xstrat.*	XFS flush deamon 清空的文件数据的字节数以及缓冲区数量计数器（清空到磁盘上连续和非相邻空间）的计数器。

xfs.attr.*	所有 XFS 文件系统的属性数量、设置、删除和列出操作的数量。
xfs.quota.*	在 XFS 文件系统上，配额操作的指标包括数字配额回收、配额缓存未命中、缓存命中和配额数据回收。
xfs.buffer.*	有关 XFS 缓冲区对象的指标范围。计数器包括请求的缓冲区调用数、成功缓冲区锁定、等待的缓冲区锁定、miss_locks、miss_retries 和 buffer hit（在查找页面时）。
xfs.btree.*	有关 XFS btree 操作的指标。
xfs.control.reset	配置用于重置 XFS 统计数据的指标计数器的配置指标。使用 pmstore 工具切换控制指标。

8.5. 每个设备 PCP 指标组用于 XFS

下表描述了适用于 XFS 的每设备 PCP 指标组。

表 8.2. 每个设备 PCP 指标组用于 XFS

指标组	提供的指标
xfs.perdev.*	常规 XFS 指标，包括读取和写入操作计数、读取和写入字节计数。另外，索引节点的次数也会刷新、集群集群和集群失败数的计数器。
xfs.perdev.allocs.* **xfs.perdev.alloc_btree.***	有关文件系统中对象分配的指标范围，其中包括扩展的数量和块创建/自由。分配树查找和比较，以及扩展从 btree 创建和删除记录。
xfs.perdev.block_map.* **xfs.perdev.bmap_btree.***	指标包括块映射的读/写入和块删除次数，用于插入、删除和查找的扩展列表操作。另外，还可从 blockmap 进行比较、查找、插入和删除操作的操作计数器。
xfs.perdev.dir_ops.*	XFS 文件系统的目录操作计数器，用于创建、删除条目，计数为"getdent"操作。
xfs.perdev.transactions.*	meta-data 事务的数量的计数器包括同步和异步事务的数量以及空事务的数量。
xfs.perdev.inode_ops.*	操作系统在索引节点缓存中查找带有不同结果的 XFS 索引节点的次数计数器。这些计数的缓存命中，缓存未命中等。

xfs.perdev.log.* **xfs.perdev.log_tail.***	通过 XFS filesytems 写入的日志缓冲区数的计数器包括写入磁盘的块数量。还提供日志清除和固定数量的指标。
xfs.perdev.xstrat.*	XFS flush deamon 清空的文件数据的字节数以及缓冲区数量计数器（清空到磁盘上连续和非相邻空间）的计数器。
xfs.perdev.attr.*	所有 XFS 文件系统的属性数量、设置、删除和列出操作的数量。
xfs.perdev.quota.*	在 XFS 文件系统上，配额操作的指标包括数字配额回收、配额缓存未命中、缓存命中和配额数据回收。
xfs.perdev.buffer.*	有关 XFS 缓冲区对象的指标范围。计数器包括请求的缓冲区调用数、成功缓冲区锁定、等待的缓冲区锁定、miss_locks、miss_retries 和 buffer hit（在查找页面时）。
xfs.perdev.btree.*	有关 XFS btree 操作的指标。

第 9 章 设置 PCP 指标的图形表示

使用 pcp、grafana、pcp redis、pcp bpftrace 和 pcp vector 的组合来提供 Performance Co-Pilot (PCP)收集的实时数据或数据的图形表示。

9.1. 使用 PCP-ZEROCONF 设置 PCP

这个步骤描述了如何在使用 pcp-zeroconf 软件包的系统中设置 PCP。安装 pcp-zeroconf 软件包后，系统会将默认指标集合记录到存档文件中。

步骤

- 安装 pcp-zeroconf 软件包：

```
# dnf install pcp-zeroconf
```

验证步骤

- 确保 pmlogger 服务处于活跃状态，并开始归档指标：

```
# pcp | grep pmlogger
 pmlogger: primary logger:
/var/log/pcp/pmlogger/localhost.localdomain/20200401.00.12
```

其他资源

- pmlogger man page
- 使用 Performance Co-Pilot 监控性能

9.2. 设置 GRAFANA-SERVER

Grafana 生成可从浏览器访问的图形。grafana-server 是 Grafana 仪表板的后端服务器。默认情况下，它监听所有接口，并提供通过 Web 浏览器访问的 Web 服务。grafana-pcp 插件与后端中的 pmproxy 协议交互。

这个步骤描述了如何设置 grafana-server。

先决条件

- **配置了 PCP。如需更多信息，请参阅使用 pcp-zeroconf 设置 PCP。**

步骤

1. **安装以下软件包：**

```
# dnf install grafana grafana-pcp
```

2. **重启并启用以下服务：**

```
# systemctl restart grafana-server
# systemctl enable grafana-server
```

3. **为到 Grafana 服务的网络流量打开服务器防火墙。**

```
# firewall-cmd --permanent --add-service=grafana
success

# firewall-cmd --reload
success
```

验证步骤

- **确定 grafana-server 正在侦听并响应请求：**

```
# ss -ntlp | grep 3000
LISTEN  0  128  *:3000  *:*  users:(("grafana-server",pid=19522,fd=7))
```

- **确保安装了 grafana-pcp 插件：**

```
# grafana-cli plugins ls | grep performancecopilot-pcp-app

performancecopilot-pcp-app @ 3.1.0
```

其他资源

- **pmproxy (1) 和 grafana-server man page**

9.3. 访问 GRAFANA WEB UI

这个步骤描述了如何访问 Grafana Web 界面。

使用 Grafana Web 界面，您可以：

- 添加 PCP Redis、PCP bpftrace 和 PCP 向量数据源
- 创建仪表板
- 查看任何有用的指标的概述
- 在 PCP Redis 中创建警报

先决条件

1. 配置了 PCP。如需更多信息，请参阅使用 pcp-zeroconf 设置 PCP。
2. grafana-server 被配置。如需更多信息，请参阅设置 grafana-server。

步骤

1. 在客户端系统中，打开浏览器，并使用 *http://192.0.2.0*:3000 链接通过端口 3000 访问 grafana-server。

   将 *192.0.2.0* 替换为您的计算机 IP。

2. 首次登录时，在 Email or username 和 Password 字段中输入 admin。

   Grafana 提示设置 新密码 以创建安全帐户。如果要稍后设置，请点 Skip。

3. 在菜单中，将鼠标悬停在

Configuration 图标上，然后单击 Plugins。

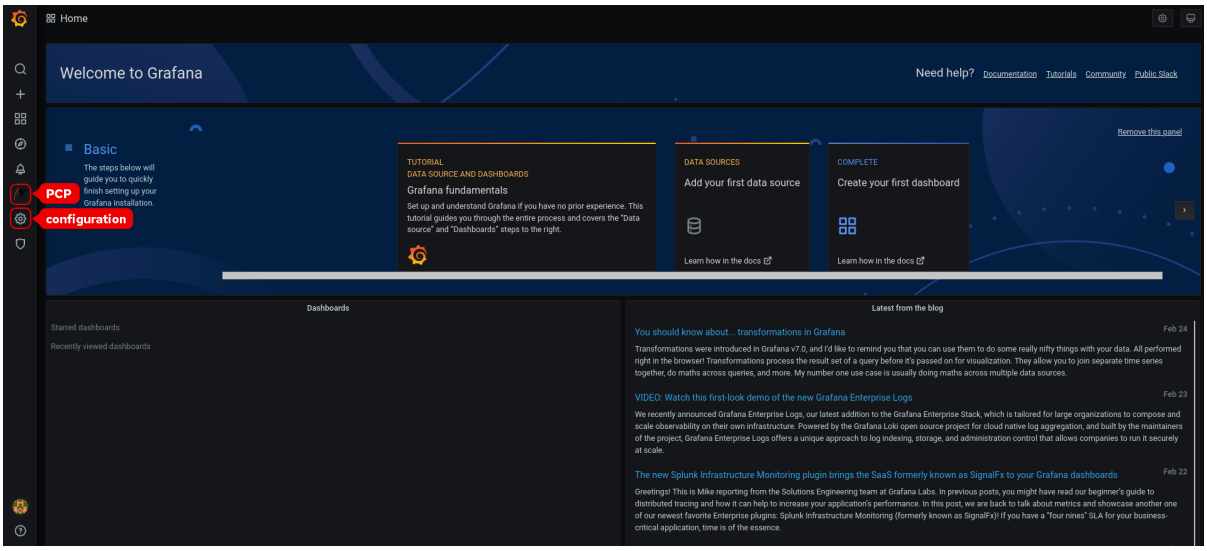
4. 在 Plugins 选项卡的 Search by name or type 中输入 performance co-pilot，然后点 Performance Co-Pilot (PCP) 插件。

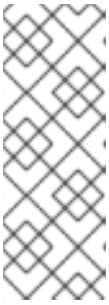
5. 在 Plugins / Performance Co-Pilot 窗格中，点 Enable。

6. 点 Grafana

图标。Grafana Home 页会被显示。

图 9.1. 仪表板主页





注意

屏幕右上角有一个类似的

图标，但它控制常规 仪表板设置。

7. 在 Grafana Home 页面中，点 Add your first data source 添加 PCP Redis、PCP bpftrace 和 PCP 向量数据源。有关添加数据源的更多信息，请参阅：

- 要添加 pcp redis 数据源，查看默认仪表板，创建面板和警报规则，请参阅在 PCP

Redis 数据源中创建面板和警报。

- 要添加 pcp bpftrace 数据源并查看默认仪表板，请参阅 PCP bpftrace System Analysis 仪表板。

- 要添加 pcp向量数据源，查看默认仪表板并查看向量清单，请参阅 查看 PCP 向量 检查列表。

8. 可选：在菜单中，将鼠标悬停在 admin profile

图标上，以更改 Preferences，包括 Edit Profile、Change Password 或 Sign out。

其他资源

- grafana-cli 和 grafana-server man page

9.4. 为 GRAFANA 配置安全连接

您可以在 Grafana 和 Performance Co-Pilot (PCP)组件之间建立安全连接。在这些组件之间建立安全连接有助于防止未授权方访问或修改正在收集和监控的数据。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。

- grafana-server 被配置。如需更多信息，请参阅设置 grafana-server。

- 私有客户端密钥存储在 /etc/grafana/grafana.key 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

  有关创建私钥和证书签名请求(CSR)的详细信息，以及如何从证书颁发机构(CA)请求证书，请参阅您的 CA 文档。

- TLS 客户端证书存储在 /etc/grafana/grafana.crt 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

步骤

1. **作为 root 用户，打开 /etc/grafana/grana.ini 文件，并调整 [server] 部分中的以下选项，以反映以下内容：**

```
protocol = https
cert_key = /etc/grafana/grafana.key
cert_file = /etc/grafana/grafana.crt
```

2. **确保 grafana 可以访问证书：**

```
# su grafana -s /bin/bash -c \
  'ls -1 /etc/grafana/grafana.crt /etc/grafana/grafana.key'
/etc/grafana/grafana.crt
/etc/grafana/grafana.key
```

3. **重启并启用 Grafana 服务，以应用配置更改：**

```
# systemctl restart grafana-server
# systemctl enable grafana-server
```

验证

1. **在客户端系统上，打开浏览器并使用 *https://192.0.2.0:3000* 链接访问 grafana-server 机器的端口 3000。将 192.0.2.0 替换为您的机器 IP。**

2. **确认**

锁图标显示在地址栏旁边。

注意

如果协议设置为 http，并且尝试 HTTPS 连接，则您将收到 ERR_SSL_PROTOCOL_ERROR 错误。如果协议设置为 https，并且尝试 HTTP 连接，则 Grafana 服务器会使用 "Client send a HTTP request to a HTTPS server" 信息响应。

9.5. 配置 PCP REDIS

使用 PCP Redis 数据源：

- 查看数据存档
- 使用 pm series 语言查询时间序列
- 分析多个主机的数据

先决条件

1. 配置了 PCP。如需更多信息，请参阅使用 pcp-zeroconf 设置 PCP。
2. grafana-server 被配置。如需更多信息，请参阅设置 grafana-server。
3. 邮件传输代理（如 sendmail 或 postfix ）已安装并配置。

步骤

1. 安装 redis 软件包：

```
# dnf install redis
```

2. 启动并启用以下服务：

```
# systemctl start pmproxy redis
# systemctl enable pmproxy redis
```

3. 重启 grafana-server ：

```
# systemctl restart grafana-server
```

验证步骤

- 确保 pmproxy 和 redis 正常工作：

```
# pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

如果没有安装 redis 软件包，这个命令不会返回任何数据。

其他资源

- pmseries (1) man page

9.6. 为 PCP REDIS 配置安全连接

您可以在 performance co-pilot (PCP)、Grafana 和 PCP redis 之间建立安全连接。在这些组件之间建立安全连接有助于防止未授权方访问或修改正在收集和监控的数据。

先决条件

- 已安装 PCP。如需更多信息，请参阅安装并启用 PCP。
- grafana-server 被配置。如需更多信息，请参阅设置 grafana-server。
- PCP redis 已安装。如需更多信息，请参阅配置 PCP Redis。
- 私有客户端密钥存储在 /etc/redis/client.key 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

  有关创建私钥和证书签名请求(CSR)的详细信息，以及如何从证书颁发机构(CA)请求证书，请参阅您的 CA 文档。
- TLS 客户端证书存储在 /etc/redis/client.crt 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。
- TLS 服务器密钥存储在 /etc/redis/redis.key 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

- TLS 服务器证书存储在 /etc/redis/redis.crt 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

- CA 证书存储在 /etc/redis/ca.crt 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

另外，对于 pmproxy 守护进程：

- 私有服务器密钥存储在 /etc/pcp/tls/server.key 文件中。如果您使用其他路径，请修改流程相应步骤中的路径。

步骤

1. 以 root 用户身份，打开 /etc/redis/redis.conf 文件，并调整 TLS/SSL 选项以反映以下属性：

```
port 0
tls-port 6379
tls-cert-file /etc/redis/redis.crt
tls-key-file /etc/redis/redis.key
tls-client-key-file /etc/redis/client.key
tls-client-cert-file /etc/redis/client.crt
tls-ca-cert-file /etc/redis/ca.crt
```

2. 确保 redis 可以访问 TLS 证书：

```
# su redis -s /bin/bash -c \
  'ls -1 /etc/redis/ca.crt /etc/redis/redis.key /etc/redis/redis.crt'
/etc/redis/ca.crt
/etc/redis/redis.crt
/etc/redis/redis.key
```

3. 重启 redis 服务器以应用配置更改：

```
# systemctl restart redis
```

验证

- 确认 TLS 配置可以正常工作：

```
# redis-cli --tls --cert /etc/redis/client.crt \
   --key /etc/redis/client.key \
   --cacert /etc/redis/ca.crt <<< "PING"
PONG
```

不成功的 TLS 配置可能导致以下错误信息：

```
Could not negotiate a TLS connection: Invalid CA Certificate File/Directory
```

9.7. 在 PCP REDIS 数据源中创建面板和警报

在添加了 PCP Redis 数据源后，您可以使用有用的指标概述控制面板，添加查询来视觉化负载图形，并创建可帮助您在系统发生后查看系统问题的警报。

先决条件

1. PCP Redis 已配置。如需更多信息，请参阅配置 PCP Redis。
2. grafana-server 可以访问。如需更多信息，请参阅 访问 Grafana Web UI。

步骤

1. 登录到 Grafana Web UI。
2. 在 Grafana Home 页面中，点 Add your first data source。
3. 在 Add data source 窗格中，在 Filter by name or type 文本框中键入 redis，然后点 PCP Redis。
4. 在 Data Sources / PCP Redis 窗格中，执行以下操作：
   a. 在 URL 字段中添加 http://localhost:44322，然后点 Save & Test。

b.

点 Dashboards tab → Import → PCP Redis: Host Overview 查看带有任何有用指标概述的仪表板。

图 9.2. PCP Redis: 主机概述

5.

添加新面板：

a.

在菜单中，将鼠标悬停在

Create icon → Dashboard → Add new panel icon 来添加一个面板。

b.

在 Query 选项卡中，从查询列表中选择 PCP Redis 而不是所选的默认选项，在 A 的文本字段中输入 metric，如 kernel.all.load 以视觉化内核负载图形。

c.

可选：添加 Panel title 和 Description，更新来自 Settings 的选项。

d.

点 Save 以应用更改并保存仪表板。添加仪表板名称。

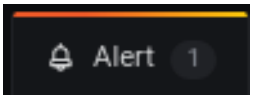
e.

点 Apply 以应用更改并返回控制面板。

图 9.3. PCP Redis 查询面板

6. 创建警报规则：

a. 在 PCP Redis 查询面板中，点



Alert，然后点 Create Alert。

b. 编辑 Rule 中的 Name, Evaluate query 和 For 项，为您的警报指定 Conditions。

c. 点 Save 以应用更改并保存仪表板。点 Apply 以应用更改并返回控制面板。

图 9.4. 在 PCP Redis 面板中创建警报

.

d. 可选：在同一面板中，向下滚动并点 Delete 图标以删除创建的规则。

e. 可选：在菜单中点击  Alerting 图标查看创建的具有不同警报状态的警报规则，从Alert Rules 选项卡编辑警报规则，或者暂停现有规则。

要为创建的警报规则添加通知频道以接收来自 Grafana 的警报通知，请参阅为警报添加通知频道。

9.8. 为警报添加通知频道

通过添加通知频道，每当满足警报规则条件且系统需要进一步监控时，可以从 Grafana 接收警报通知。

在从支持的通知程序列表中选择任意一种类型后，您可以收到这些警报，其中包括 DingDing, Discord, Email, Google Hangouts Chat, HipChat, Kafka REST Proxy, LINE, Microsoft Teams, OpsGenie, PagerDuty, Prometheus Alertmanager, Pushover, Sensu, Slack, Telegram, Threema Gateway, VictorOps, 和 webhook。

先决条件

1. grafana-server 可以访问。如需更多信息，请参阅 访问 Grafana Web UI。
2. 已创建一个警报规则。如需更多信息，请参阅在 PCP Redis 数据源中创建面板和警报。
3. 配置 SMTP 并在 grafana/grafana.ini 文件中添加有效的发件人电子邮件地址：

```
# vi /etc/grafana/grafana.ini

[smtp]
enabled = true
from_address = abc@gmail.com
```

使用有效电子邮件地址替换 ***abc@gmail.com***。

4. 重启 grafana-server

```
# systemctl restart grafana-server.service
```

步骤

1. 在菜单中，将鼠标悬停在  Alerting 图标 → 上，点 Contact Points → New contact point。

2. 在 New contact point 详情视图中，执行以下操作：

   a. 在 Name 文本框中输入您的名称

   b. 选择 Contact point type，例如 Email ，并输入电子邮件地址。您可以使用 ; 分隔符添加多个电子邮件地址。

   c. 可选：配置可选电子邮件设置和通知设置。

3. 单击 Save contact point。

4. 在警报规则中选择通知频道：

   a. 在菜单中选择 Notification policies 图标，然后点 + New specific policy。

   b. 选择您刚才创建的 Contact point

   c. 点 Save policy 按钮

其他资源

- 警报通知的上游 Grafana 文档

9.9. 在 PCP 组件间设置身份验证

您可以使用 scram-sha-256 身份验证机制来设置身份验证，该机制可通过简单身份验证安全层 (SASL) 框架获得 PCP 支持的。

步骤

1. 为 scram-sha-256 身份验证机制安装 sasl 框架：

```
# dnf install cyrus-sasl-scram cyrus-sasl-lib
```

2. 在 pmcd.conf 文件中指定支持的身份验证机制和用户数据库路径：

```
# vi /etc/sasl2/pmcd.conf

mech_list: scram-sha-256

sasldb_path: /etc/pcp/passwd.db
```

3. 创建一个新用户：

```
# useradd -r metrics
```

使用您的用户名替换 ***metrics***。

4. 在用户数据库中添加创建的用户：

```
# saslpasswd2 -a pmcd metrics

Password:
Again (for verification):
```

要添加创建的用户，您需要输入 *指标* 帐户密码。

5. 设置用户数据库的权限：

```
# chown root:pcp /etc/pcp/passwd.db
# chmod 640 /etc/pcp/passwd.db
```

6. 重启 pmcd 服务：

```
# systemctl restart pmcd
```

验证步骤

- 验证 sasl 配置：

```
# pminfo -f -h "pcp://127.0.0.1?username=metrics" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

其他资源

- saslauthd (8) , pminfo (1) , 和 sha256 man pages
- 如何在 RHEL 8.2 中在 PCP 组件（如 PMDA 和 pmcd）之间设置身份验证？

9.10. 安装 PCP BPFTRACE

安装 PCP bpftrace 代理以内省系统，并从内核和用户空间追踪点收集指标。

bpftrace 代理使用 bpftrace 脚本来收集指标。bpftrace 脚本使用增强的 Berkeley Packet Filter (eBPF) 。

这个步骤描述了如何安装 pcp bpftrace。

先决条件

1. 配置了 PCP。如需更多信息，请参阅使用 pcp-zeroconf 设置 PCP。

2. grafana-server 被配置。如需更多信息，请参阅设置 grafana-server。

3. scram-sha-256 身份验证机制被配置。如需更多信息，请参阅在 PCP 组件之间设置身份验证。

步骤

1. 安装 pcp-pmda-bpftrace 软件包：

```
# dnf install pcp-pmda-bpftrace
```

2. 编辑 bpftrace.conf 文件并添加您在 {setting-up-authentication-b between-pcp-components} 中创建的用户：

```
# vi /var/lib/pcp/pmdas/bpftrace/bpftrace.conf

[dynamic_scripts]
enabled = true
auth_enabled = true
allowed_users = root,metrics
```

使用您的用户名替换 *metrics*。

3. 安装 bpftrace PMDA：

```
# cd /var/lib/pcp/pmdas/bpftrace/
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bpftrace metrics have appeared ... 7 metrics and 6 values
```

pmda-bpftrace 现已安装，只能在验证您的用户后使用。如需更多信息，请参阅 查看 PCP bpftrace System Analysis 仪表板。

其他资源

- pmdabpftrace (1) 和 bpftrace man page

9.11. 查看 PCP BPFTRACE SYSTEM ANALYSIS 仪表板

使用 PCP bpftrace 数据源，您可以从 pmlogger 或 archive 中不以普通数据提供的源访问实时数据

在 PCP bpftrace 数据源中，您可以使用有用的指标概述查看仪表板。

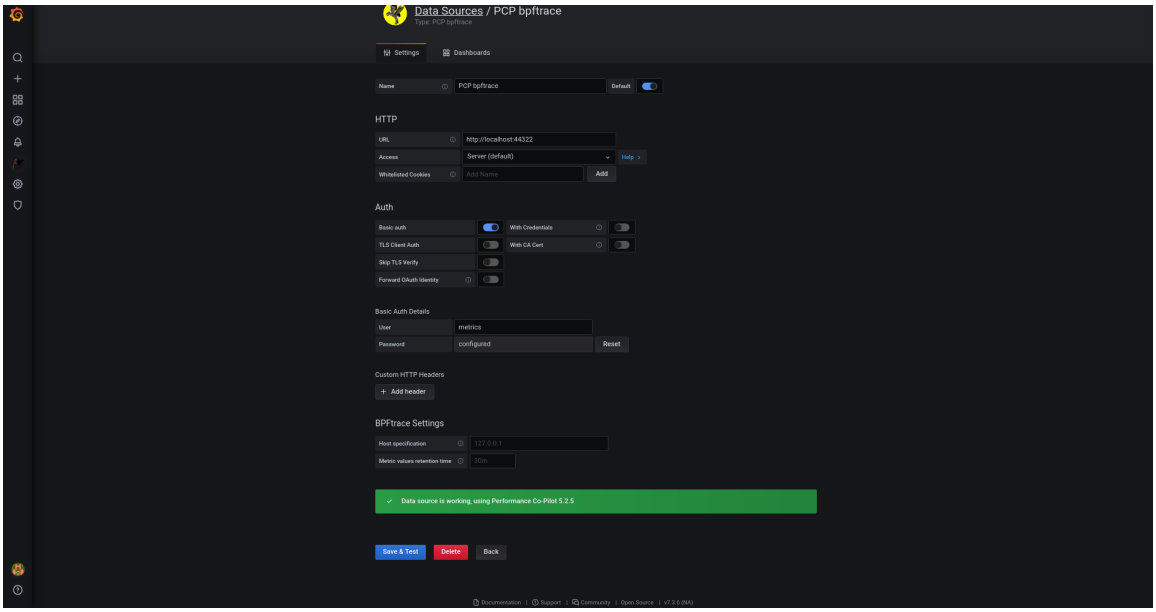
先决条件

1. 已安装 PCP bpftrace。如需更多信息，请参阅安装 PCP bpftrace。
2. grafana-server 可以访问。如需更多信息，请参阅 访问 Grafana Web UI。

步骤

1. 登录到 Grafana Web UI。
2. 在 Grafana Home 页面中，点 Add your first data source。
3. 在 Add data source 窗格中，在 Filter by name or type 文本框中键入 bpftrace，然后点 PCP bpftrace。
4. 在 Data Sources / PCP bpftrace 窗格中，执行以下操作：
   a. 在 URL 字段中添加 http://localhost:44322。
   b. 切换 Basic Auth 选项，并在 User 和 Password 字段中添加创建的用户凭证。
   c. 点 Save and Test。

图 9.5. 在数据源中添加 PCP bpftrace



d. 点 Dashboards tab → Import → PCP bpftrace: System Analysis 查看包含任何有用的指标概述的仪表板。

图 9.6. PCP bpftrace: 系统分析

9.12. 安装 PCP 向量

这个步骤描述了如何安装 pcp 向量。

先决条件

1. 配置了 PCP。如需更多信息，请参阅使用 pcp-zeroconf 设置 PCP。

2. grafana-server 被配置。如需更多信息，请参阅设置 grafana-server。

步骤

1. 安装 pcp-pmda-bcc 软件包：

```
# dnf install pcp-pmda-bcc
```

2. 安装 bcc PMDA：

```
# cd /var/lib/pcp/pmdas/bcc
# ./Install
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Initializing, currently in 'notready' state.
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Enabled modules:
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: ['biolatency', 'sysfork',
[...]
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bcc metrics have appeared ... 1 warnings, 1 metrics and 0 values
```

其他资源

- pmdabcc (1) man page

9.13. 查看 PCP 向量清单

PCP 向量数据源显示实时指标并使用 pcp 指标。它分析单个主机的数据。

在添加了 PCP 向量数据源后，您可以使用有用的指标概述来查看仪表板，并查看清单中的相关故障排除或引用链接。

先决条件

1. 已安装 PCP 向量。如需更多信息，请参阅安装 PCP 向量。

2. grafana-server 可以访问。如需更多信息，请参阅 访问 Grafana Web UI。

步骤

1. 登录到 Grafana Web UI。

2. 在 Grafana Home 页面中，点 Add your first data source。

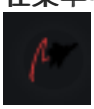
3. 在 Add data source 窗格中，在 Filter by name or type 文本框中键入 vector，然后点 PCP 向量。

4. 在 Data Sources / PCP Vector 窗格中，执行以下操作：

   a. 在 URL 字段中添加 http://localhost:44322，然后点 Save & Test。

   b. 点 Dashboards tab → Import → PCP Vector: Host Overview 查看带有任何有用指标概述的仪表板。

图 9.7. PCP Vector：主机概述

5. 在菜单中，将鼠标悬停在  Performance Co-Pilot 插件上，然后单击 PCP Vector Checklist。

在 PCP 检查列表中，点 

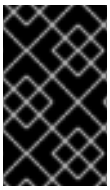
帮助或

警告图标查看相关的故障排除或参考链接。

图 9.8. Performance Co-Pilot / PCP 向量清单

9.14. 在 GRAFANA 中使用 HEATMAPS

您可以在 Grafana 中使用 heatmaps 来查看数据的直方图，识别数据中的趋势和模式，并查看它们随时间变化。heatmap 中的每列代表一个直方图，不同的颜色单元代表该直方中给定值的不同破坏程度。



重要

这个特定工作流用于 Grafana 版本 9.0.9 及之后的版本中的 heatmaps。

先决条件

- PCP Redis 已配置。如需更多信息，请参阅配置 PCP Redis。
- grafana-server 可以访问。如需更多信息，请参阅 访问 Grafana Web UI。
- PCP Redis 数据源已配置。如需更多信息，请参阅在 PCP Redis 数据源中创建面板和警报。

步骤

1. 将光标悬停在 Dashboards 选项卡上，然后单击 + New dashboard。

2. 在 Add 面板 菜单中，单击 Add a new panel。

3. 在 Query 选项卡中：

   a. 从查询列表中选择 PCP Redis，而不是所选的默认选项。

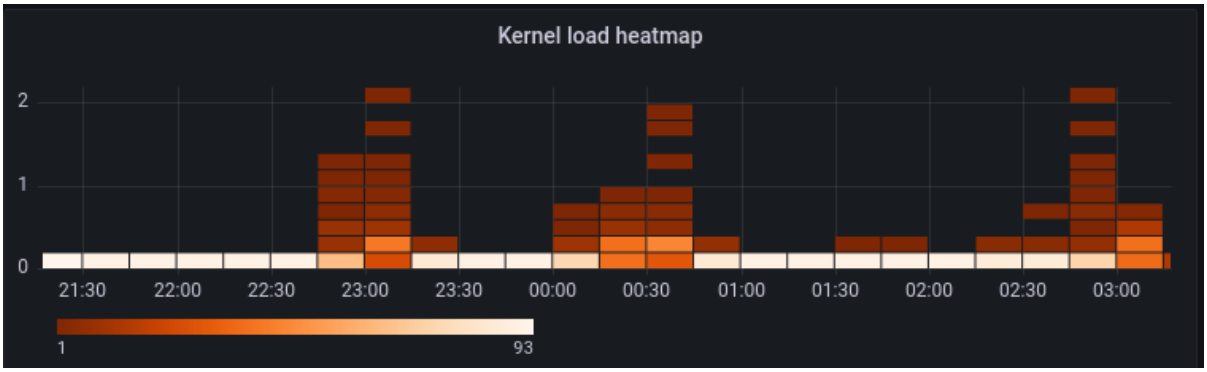
   b. 在 A 的文本字段中，输入指标，如 kernel.all.load 以视觉化内核负载图。

4. 单击默认设为 Time series 的视觉化下拉菜单，然后单击 Heatmap。

5. 可选： 在面板选项 下拉菜单中，添加一个面板 标题和描述。

6. 在 Heatmap 下拉菜单中，在 Calculate from data 设置下，单击 Yes。

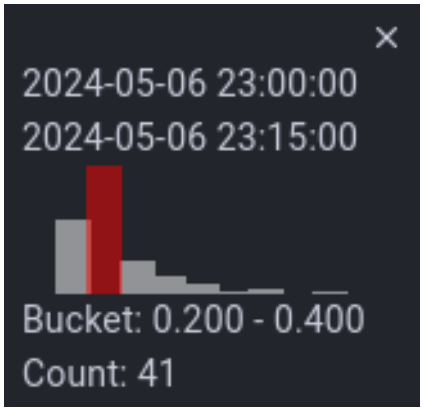
heatmap



7. 可选：在 Colors 下拉菜单中，从默认的 Orange 更改 Scheme，然后选择步骤数(color shades)。

8. 可选：在 Tooltip 下拉菜单中，在 Show histogram (Y Axis) 设置下，单击切换，在光标悬停在 heatmap 中的单元格上时显示单元的位置。例如：

显示直方图(Y Axis)单元显示



9.15. GRAFANA 问题故障排除

有时有必要对 Grafana 问题进行故障排除，如 Grafana 不显示任何数据、仪表盘是黑的或类似的问题。

步骤

- 通过执行以下命令，验证 pmlogger 服务是否已启动并正在运行：

```
$ systemctl status pmlogger
```

- 运行以下命令，验证是否在磁盘中创建或修改了文件：

```
$ ls /var/log/pcp/pmlogger/$(hostname)/ -rlt
total 4024
-rw-r--r--. 1 pcp pcp   45996 Oct 13  2019 20191013.20.07.meta.xz
-rw-r--r--. 1 pcp pcp     412 Oct 13  2019 20191013.20.07.index
-rw-r--r--. 1 pcp pcp   32188 Oct 13  2019 20191013.20.07.0.xz
-rw-r--r--. 1 pcp pcp   44756 Oct 13  2019 20191013.20.30-00.meta.xz
[..]
```

- 运行以下命令验证 pmproxy 服务是否正在运行：

```
$ systemctl status pmproxy
```

- 通过查看 /var/log/pcp/pmproxy/pmproxy.log 文件确定其包括一些内容来验证 pmproxy 是否正在运行、时间序列支持是否被启用以及到 Redis 的连接：

```
pmproxy(1716) Info: Redis slots, command keys, schema version setup
```

在这里，1716 是 pmproxy 的 PID，对于每次调用 pmproxy 时，将有所不同。

- 运行以下命令，验证 **Redis** 数据库是否包含任何密钥：

```
$ redis-cli dbsize
(integer) 34837
```

- 通过执行以下命令，验证 **Redis** 数据库和 **pmproxy** 中的任何 **PCP** 指标是否能够访问它们：

```
$ pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df

$ pmseries "disk.dev.read[count:10]"
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
   [Mon Jul 26 12:21:10.085468000 2021] 117971
70e83e88d4e1857a3a31605c6d1333755f2dd17c
   [Mon Jul 26 12:21:00.087401000 2021] 117758
70e83e88d4e1857a3a31605c6d1333755f2dd17c
   [Mon Jul 26 12:20:50.085738000 2021] 116688
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[...]
```

```
$ redis-cli --scan --pattern "*$(pmseries 'disk.dev.read')"

pcp:metric.name:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:values:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:desc:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelvalue:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:instances:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelflags:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

- 运行以下命令，验证 **Grafana** 日志中是否有错误：

```
$ journalctl -e -u grafana-server
-- Logs begin at Mon 2021-07-26 11:55:10 IST, end at Mon 2021-07-26 12:30:15 IST. --
Jul 26 11:55:17 localhost.localdomain systemd[1]: Starting Grafana instance...
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-
26T11:55:17+0530 lvl=info msg="Starting Grafana" logger=server version=7.3.6 c>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-
26T11:55:17+0530 lvl=info msg="Config loaded from" logger=settings file=/usr/s>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-
26T11:55:17+0530 lvl=info msg="Config loaded from" logger=settings file=/etc/g>
[...]
```

第 10 章 使用 WEB 控制台优化系统性能

了解如何在 RHEL web 控制台中设置性能配置集，以便为所选任务优化系统性能。

10.1. WEB 控制台中的性能调优选项

Red Hat Enterprise Linux 9 提供几个根据以下任务优化系统的性能配置集：

- 使用桌面的系统
- 吞吐性能
- 延迟性能
- 网络性能
- 低电源消耗
- 虚拟机

TuneD 服务优化系统选项以匹配所选配置集。

在 Web 控制台中，您可以设置系统使用的哪个性能配置集。

其他资源

- TuneD 入门

10.2. 在 WEB 控制台中设置性能配置集

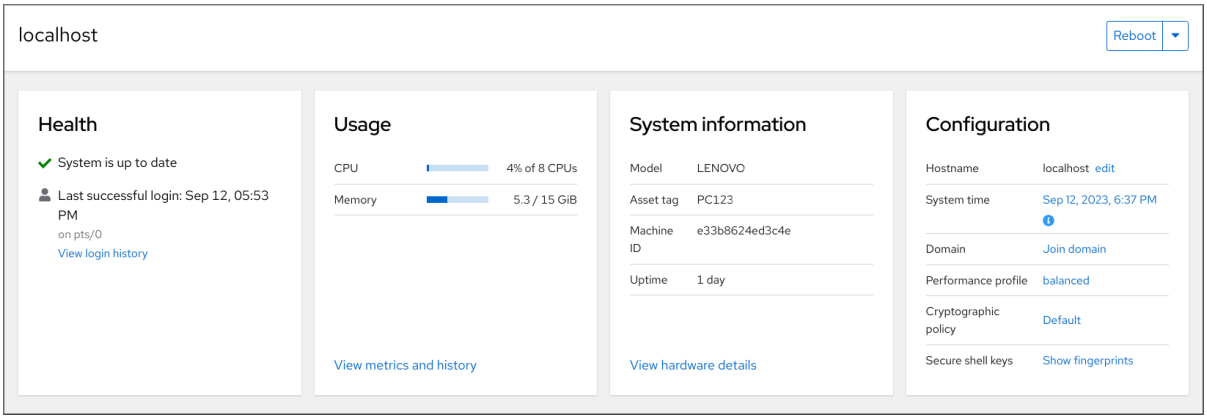
根据您要执行的任务，您可以使用 Web 控制台通过设置合适的性能配置集来优化系统性能。

先决条件

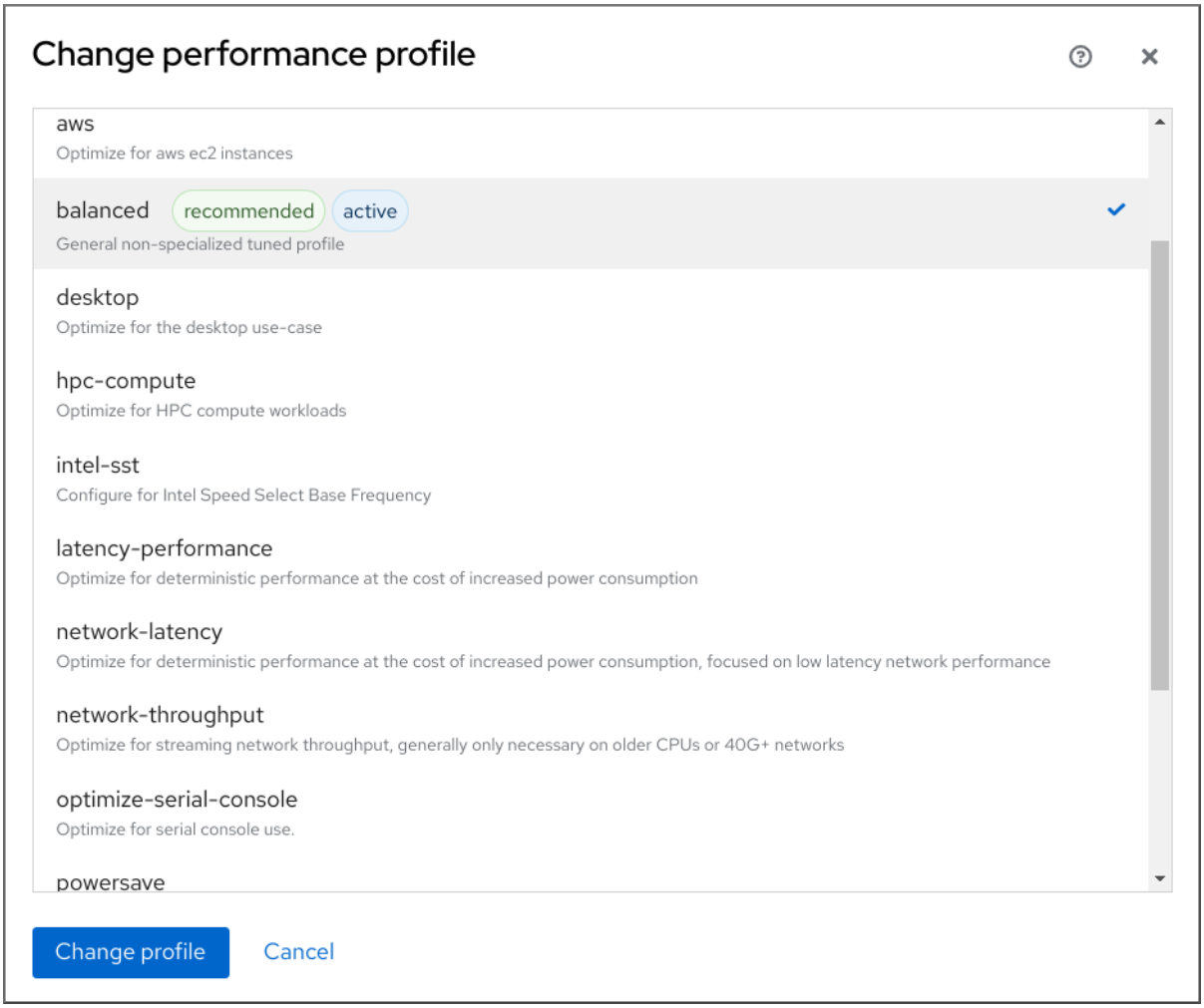
- 确保 Web 控制台已安装并可以访问。详情请参阅安装 Web 控制台。

流程

1. 登录到 web 控制台。详情请参阅 Web 控制台的日志记录。
2. 点 Overview。
3. 在 Configuration 部分中，点当前的性能配置文件。



4. 在 Change Performance Profile 对话框中设置所需的配置文件。



5. 点 Change Profile。

验证步骤

- Overview 选项卡现在在 Configuration 部分中显示所选性能配置文件。

10.3. 使用 WEB 控制台监控本地系统的性能

Red Hat Enterprise Linux Web 控制台使用 Utilization Saturation and Errors (USE) 方法进行故障排除。新的性能指标页面带有最新数据，您可以对数据进行组织化的历史视图。

在 Metrics and history 页面中，您可以查看事件、错误和资源使用率和饱和度的图形表示。

先决条件

- Web 控制台已安装并可以访问。详情请参阅安装 Web 控制台。
- cockpit-pcp 软件包（启用收集性能指标）已安装：
  a. 从 Web 控制台界面安装软件包：
    i. 使用管理权限登录到 web 控制台。详情请参阅 Web 控制台的日志记录。
    ii. 在 Overview 页面中，单击 View metrics and history。
    iii. 点 Install cockpit-pcp 按钮。
    iv. 在安装软件对话框窗口中，点安装。
  b. 要从命令行界面安装软件包，请使用：

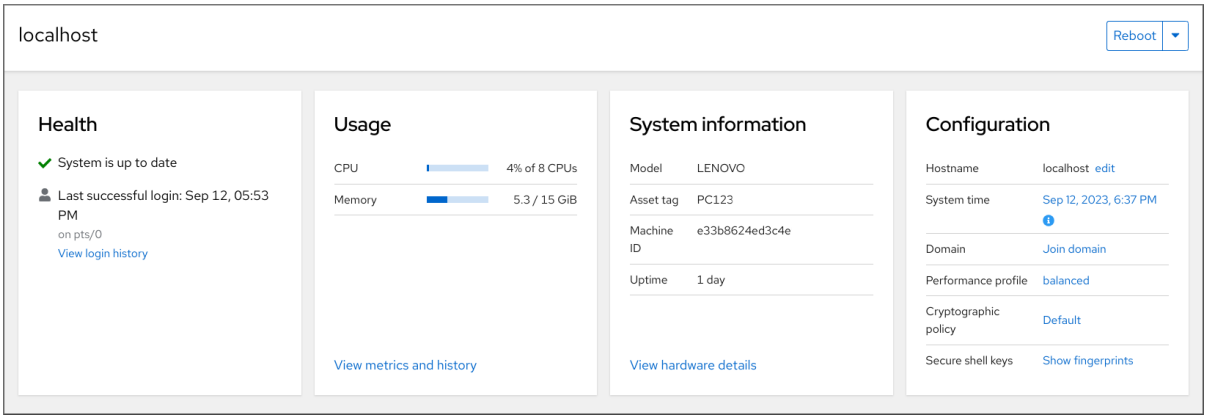
```
# dnf install cockpit-pcp
```

- 启用 Performance Co-Pilot (PCP)服务：

```
# systemctl enable --now pmlogger.service pmproxy.service
```

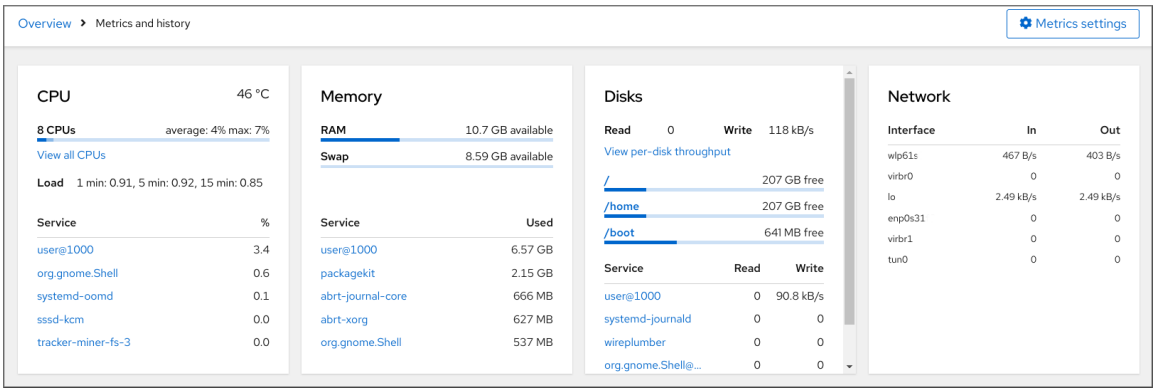
流程

1. 登录到 web 控制台。详情请参阅 Web 控制台的日志记录。
2. 点 Overview。
3. 在 Usage 部分中，点 View metrics and history。

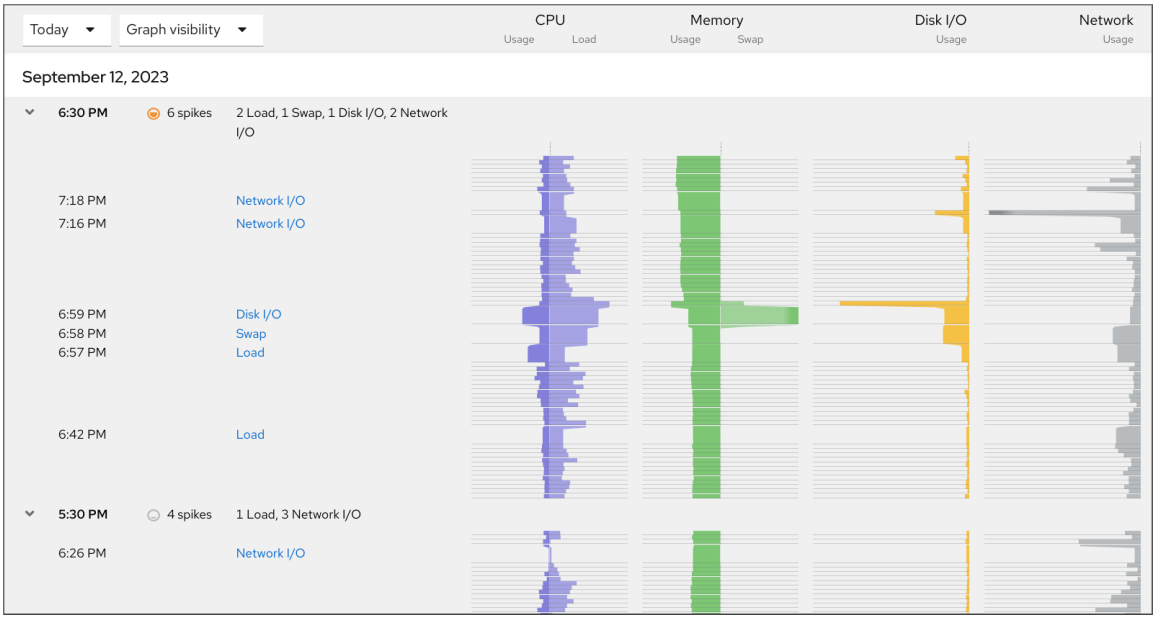


Metrics and history 部分打开：

- **当前系统配置和使用情况：**



- **用户指定的时间间隔后，图形形式的性能指标：**



10.4. 使用 WEB 控制台和 GRAFANA 监控多个系统的性能

Grafana 使您可以一次从多个系统中收集数据，并查看其收集的 Performance Co-Pilot (PCP)指标的图形表示。您可以在 web 控制台界面中的多个系统设置性能指标监控和导出。

前提条件

- 必须安装并可以访问 Web 控制台。详情请参阅链接：安装 Web 控制台。
- 安装 cockpit-pcp 软件包。

1. 在 Web 控制台界面中：
   a. 使用管理权限登录到 web 控制台。详情请参阅 Web 控制台的日志记录。
   b. 在 Overview 页面中，点 View details 和 history。
   c. 点 Install cockpit-pcp 按钮。
   d. 在安装软件对话框窗口中，点安装。

e. 注销并再次登录以查看指标历史记录。

2. 要从命令行界面安装软件包，请使用：

```
# dnf install cockpit-pcp
```

- 启用 PCP 服务：

```
# systemctl enable --now pmlogger.service pmproxy.service
```

- 设置 Grafana 仪表板。如需更多信息，请参阅设置 grafana-server。

- 安装 redis 软件包。

```
# dnf install redis
```

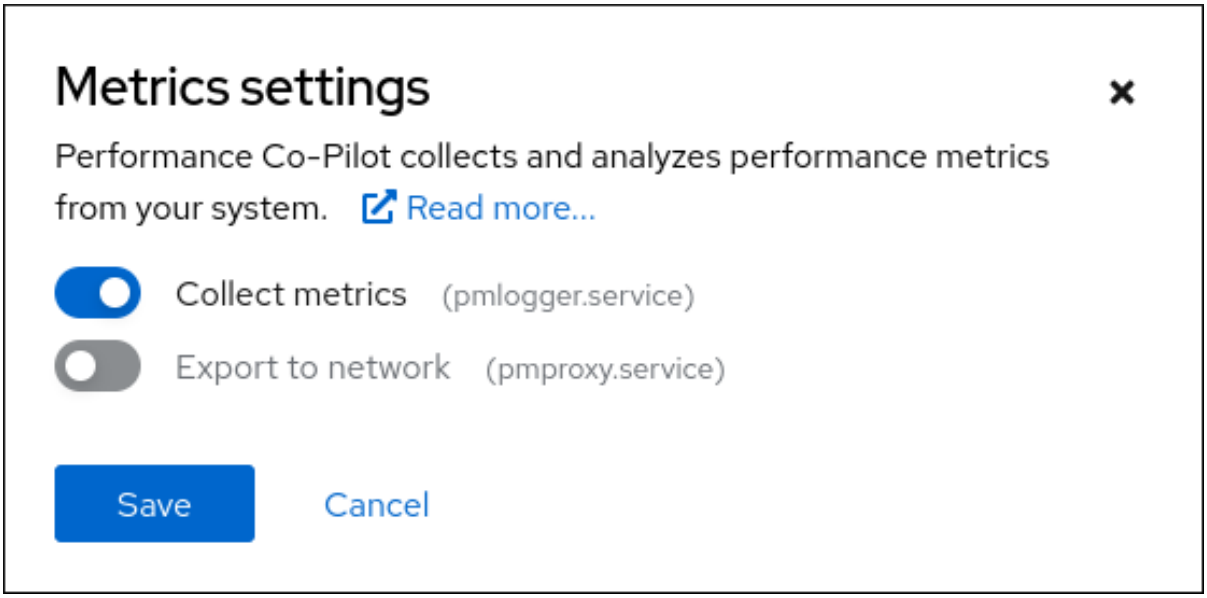
另外，您可以在稍后的流程中从 Web 控制台界面安装软件包。

流程

1. 在 Overview 页面中，点 Usage 表中的 View metrics and history。

2. 点 Metrics 设置 按钮。

3. 将 Export to network 滑块移到活跃位置。



如果您没有安装 redis 软件包，Web 控制台会提示您安装它。

4. **要打开 pmproxy 服务，请从下拉列表中选择一个区，然后点 Add pmproxy 按钮。**

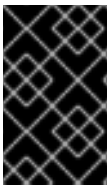
5. **点击 Save。**

验证

1. **点 Networking。**

2. **在 Firewall 表中，点 Edit rules and zones 按钮。**

3. **在您选择的区域中搜索 pmproxy。**



重要

在您要监视的所有系统中重复此步骤。

其他资源

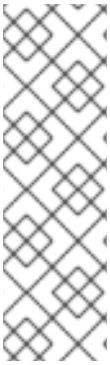
- 设置 PCP 指标的图形表示

第 11 章 设置磁盘调度程序

磁盘调度程序负责对提交至存储设备的 I/O 请求进行排序。

您可以通过几种不同方式配置调度程序：

- 使用 TuneD 设置调度程序，如使用 TuneD 设置磁盘调度程序中所述
- 使用 udev 规则设置调度程序，如使用 udev 规则设置磁盘调度程序中所述
- 在运行中的系统上临时更改调度程序，如临时 为特定磁盘设置调度程序



注意

在 Red Hat Enterprise Linux 9 中，块设备只支持多队列调度。这可让块层性能针对使用快速固态驱动器（SSD）和多核系统进行正常扩展。

Red Hat Enterprise Linux 7 和更早的版本中的传统、单一队列调度程序已被删除。

11.1. 可用磁盘调度程序

Red Hat Enterprise Linux 9 中支持以下多队列磁盘调度程序：

none

实施第一出 (FIFO) 调度算法。它将请求合并到通用块层，并通过一个简单的最近缓存来合并。

mq-deadline

尝试为请求到达调度程序的时间点提供有保证的延迟。

mq-deadline 调度程序将排队的 I/O 请求分为读取或写入批处理，然后调度它们以增加逻辑块寻址 (LBA) 顺序执行。默认情况下，读取批处理的优先级高于写入批处理，因为应用程序更有可能阻止读 I/O 操作。在 mq-deadline 批处理后，它会检查写操作在处理器时间耗尽的时间，并根据情况调度下一个读取或写入批处理。

这个调度程序适用于大多数用例，特别是那些写入操作是异步的。

bfq

以桌面系统和互动任务为目标。

bfq 调度程序可确保任何单个应用程序都不会使用所有带宽。实际上，存储设备总是像它们处于闲置时一样进行响应。在其默认配置中，bfq 注重提供最低延迟，而不是达到最大吞吐量。

BFQ 基于 cfq 代码。它不会为每个进程授予固定时间片段，但会为进程分配一个扇区数衡量的 *budget（预算）*。

在复制大型文件时，这个调度程序不适用。

kyber

调度程序调整自身，以通过计算提交到块 I/O 层的每个 I/O 请求的延迟来实现延迟目标。您可以为读取配置目标延迟，如 cache-misses 和同步写入请求。

此调度程序适用于快速设备，如 NVMe、SSD 或其他低延迟设备。

11.2. 不同用例的磁盘调度程序

根据系统执行的任务，建议在进行任何分析和调优任务前，将以下磁盘调度程序作为基准：

表 11.1. 适用于不同用例的磁盘调度程序

使用案例	磁盘调度程序
传统的使用 SCSI 接口的 HDD	使用 **mq-deadline** 或 **bfq**。
高性能 SSD 或具有快速存储的 CPU 绑定系统	使用 **none**，特别是在运行企业级应用程序时。或者，使用 **kyber**。
桌面或互动任务	使用 **bfq**。
虚拟客户端	使用 **mq-deadline**。使用可以多队列的主机总线适配器 (HBA) 驱动程序，使用 **none**。

11.3. 默认磁盘调度程序

块设备使用默认的磁盘调度程序，除非您指定了另一个调度程序。

注意

具体来说，对于 非易失性内存 Express (NVMe) 块设备，默认调度程序为 none，红帽建议不更改此设备。

内核会根据设备类型选择默认磁盘调度程序。自动选择调度程序通常是最佳设置。如果您需要不同的调度程序，红帽建议使用 udev 规则或 TuneD 应用程序来配置它。匹配所选设备并只为那些设备切换调度程序。

11.4. 确定活跃磁盘调度程序

此流程决定了哪个磁盘调度程序目前在给定块设备中活跃。

步骤

- 读取 /sys/block/*设备*/queue/scheduler 文件的内容：

```
# cat /sys/block/device/queue/scheduler
[mq-deadline] kyber bfq none
```

在文件名中，将 *device* 替换为块设备名称，如 sdc。

活跃的调度程序列在方括号中 ([ ]) 。

11.5. 使用 TUNED 设置磁盘调度程序

此流程创建并启用 TuneD 配置集，该配置集为所选块设备设置给定磁盘调度程序。这个设置会在系统重启后保留。

在以下命令和配置中替换：

- 带有块设备名称的 *device*，如 sdf

- 带有您要为该设备设置的磁盘调度程序的 *selected-scheduler*，例如 bfq

先决条件

- TuneD 服务已安装并启用。详情请参阅安装和启用 TuneD。

流程

1. 可选：选择一个您的配置集将要基于的现有 Tuned 配置集。有关可用配置集列表，请参阅 RHEL 提供的 TuneD 配置集。

   要查看哪个配置集当前处于活跃状态，请使用：

   ```
   $ tuned-adm active
   ```

2. 创建一个新目录来保存 TuneD 配置集：

   ```
   # mkdir /etc/tuned/my-profile
   ```

3. 查找所选块设备系统唯一标识符：

   ```
   $ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'

   ID_WWN=0x5002538d00000000_
   ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
   ID_SERIAL_SHORT=20120501030900000
   ```

   注意

   本例中的命令将返回以 World Wide Name (WWN) 或与指定块设备关联的序列号的所有值。虽然最好使用 WWN，但给定设备始终不能使用 WWN，但 example 命令返回的任何值都可以接受用作 *device system unique ID*。

4. 创建 /etc/tuned/*my-profile*/tuned.conf 配置文件。在该文件中设置以下选项：

a. 可选：包含现有配置集：

```
[main]
include=existing-profile
```

b. 为与 WWN 标识符匹配的设备设置所选磁盘调度程序：

```
[disk]
devices_udev_regex=IDNAME=device system unique id
elevator=selected-scheduler
```

在这里：

- 使用要使用的标识符的名称替换 ***IDNAME***（如 **ID_WWN**）。
- 将 ***device system unique id*** 替换为所选标识符的值（如 **0x5002538d00000000**）。

要匹配 **devices_udev_regex** 选项中的多个设备，将标识符放在括号中，并使用垂直栏来分离它们：

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. 启用您的配置集：

```
# tuned-adm profile my-profile
```

验证步骤

1. 验证 **TuneD** 配置集是否活跃并应用：

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
```

```
See TuneD log file ('/var/log/tuned/tuned.log') for details.
```

2. 读取 /sys/block/*设备*/queue/scheduler 文件的内容：

```
# cat /sys/block/device/queue/scheduler
[mq-deadline] kyber bfq none
```

在文件名中，将 ***device*** 替换为块设备名称，如 sdc。

活跃的调度程序列在方括号中 ([]) 。

其他资源

- 自定义 TuneD 配置集。

11.6. 使用 UDEV 规则设置磁盘调度程序

此流程使用 udev 规则为特定块设备设置给定磁盘调度程序。这个设置会在系统重启后保留。

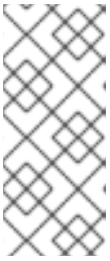
在以下命令和配置中替换：

- 带有块设备名称的 ***device***，如 sdf
- 带有您要为该设备设置的磁盘调度程序的 ***selected-scheduler***，例如 bfq

步骤

1. 查找块设备系统唯一标识符：

```
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
E: ID_WWN=0x5002538d00000000
E: ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
E: ID_SERIAL_SHORT=20120501030900000
```



注意

本例中的命令将返回以 World Wide Name (WWN) 或与指定块设备关联的序列号的所有值。虽然最好使用 WWN，但给定设备始终不能使用 WWN，但 example 命令返回的任何值都可以接受用作 *device system unique ID*。

2. **配置 udev 规则。使用以下内容创建 /etc/udev/rules.d/99-scheduler.rules 文件：**

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

在这里：

- **使用要使用的标识符的名称替换 *IDNAME*（如 ID_WWN）。**
- **将 *device system unique id* 替换为所选标识符的值（如 0x5002538d00000000）。**

3. **重新载入 udev 规则：**

```
# udevadm control --reload-rules
```

4. **应用调度程序配置：**

```
# udevadm trigger --type=devices --action=change
```

验证步骤

- **验证活跃的调度程序：**

```
# cat /sys/block/device/queue/scheduler
```

11.7. 为特定磁盘临时设置调度程序

此流程为特定块设备设置给定磁盘调度程序。系统重启后该设置不会保留。

步骤

步骤

- 将所选调度程序的名称写入 /sys/block/*device*/queue/scheduler 文件：

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

在文件名中，将 *device* 替换为块设备名称，如 sdc。

验证步骤

- 验证调度程序是否在该设备中活跃：

```
# cat /sys/block/device/queue/scheduler
```

第 12 章 调整 SAMBA 服务器的性能

了解在某些情况下，哪些设置可以提高 Samba 的性能，以及哪些设置可能会对性能造成负面影响。

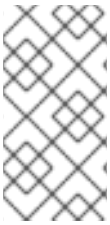
本节的部分内容来自在 Samba Wiki 中发布的 Performance Tuning 文档。许可证： CC BY 4.0。作者和贡献者：请参阅 Wiki 页面上的历史选项卡。

先决条件

- Samba 被设置为文件或打印服务器

12.1. 设置 SMB 协议版本

每个新的 SMB 版本都添加了特性并提高了协议的性能。最新的 Windows 和 Windows 服务器操作系统始终支持最新的协议版本。如果 Samba 也使用最新的协议版本，那么连接到 Samba 的 Windows 客户端将从性能改进中受益。在 Samba 中，server max protocol的默认值被设置为最新支持的稳定的 SMB 协议版本。



注意

要始终拥有最新的稳定的 SMB 协议版本，请不要设置 server max protocol 参数。如果手动设置参数，则需要修改 SMB 协议的每个新版本的设置，以便启用最新的协议版本。

以下流程解释了如何对 server max protocol 参数使用默认值。

步骤

1. 从 /etc/samba/smb.conf 文件的 [global] 部分中删除 server max protocol 参数。
2. 重新载入 Samba 配置

```
# smbcontrol all reload-config
```

12.2. 与包含大量文件的目录调整共享

Linux 支持区分大小写的文件名。因此，在搜索或访问文件时，Samba需要针对大小写文件名来扫描目录。您可以将共享配置为只以小写或大写来创建新文件，这可以提高性能。

先决条件

- Samba 配置为文件服务器

步骤

1. 将共享上的所有文件重命名为小写。

注意

使用这个过程中的设置，名称不为小写的文件将不再显示。

2. 在共享部分中设置以下参数：

```
case sensitive = true
default case = lower
preserve case = no
short preserve case = no
```

有关参数的详情，请查看 smb.conf (5) 手册页 中的描述。

3. 验证/etc/samba/smb.conf文件：

```
# testparm
```

4. 重新载入 Samba 配置：

```
# smbcontrol all reload-config
```

应用了这些设置后，此共享上所有新创建的文件的名称都使用小写。由于这些设置，Samba 不再需要针对大小写来扫描目录，这样可以提高性能。

12.3. 可能会对性能造成负面影响的设置

默认情况下，Red Hat Enterprise Linux 中的内核会根据高网络性能进行了微调。例如，内核对缓冲区大小使用自动轮询机制。在 /etc/samba/smb.conf 文件中设置 socket options 参数会覆盖这些内核设置。因此，设置此参数会在大多数情况下降低 Samba 网络性能。

要使用内核的优化的设置，请从 /etc/samba/smb.conf 中的 [global] 部分删除 socket options 参数。

第 13 章 优化虚拟机性能

与主机相比，虚拟机的性能总会有所降低。以下小节解释了这个冲突的原因，并提供了有关如何在 RHEL 9 中最小化虚拟化性能影响的说明，以便您的硬件基础架构资源可以尽可能高效地使用。

13.1. 影响虚拟机性能的因素

虚拟机作为用户空间进程在主机上运行。因此管理程序需要转换主机的系统资源，以便虚拟机可使用它们。因此，部分资源会被转换消耗，因此虚拟机无法获得与主机相同的性能效率。

虚拟化对系统性能的影响

体虚拟机性能损失的原因包括：

- 虚拟 CPU（vCPU）是主机上的线，,由 Linux 调度程序处理。
- VM 不会自动继承主机内核的优化功能，比如 NUMA 或巨页。
- 主机的磁盘和网络 I/O 设置可能会对虚拟机有显著的性能影响。
- 网络流量通常通过基于软件的网桥到达虚拟机。
- 根据主机设备及其型号，特定硬件的模拟可能会产生大量的开销。

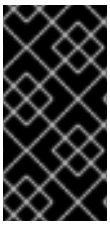
虚拟化对虚拟机性能影响的严重程度受到各种因素的影响，具体包括：

- 并行运行的虚拟机数量。
- 每个虚拟机使用的虚拟设备数量。
- 虚拟机使用的设备类型。

降低虚拟机性能损失

RHEL 9 提供了很多功能，可用于降低虚拟化的负面影响。值得注意的是：

- TuneD 服务 可以自动优化虚拟机的资源分布和性能。
- 块 I/O 调优 可以提高虚拟机块设备（如磁盘）的性能。
- NUMA 调优 可以提高 vCPU 的性能。
- 可以通过多种方式优化 虚拟网络。



重要

调整虚拟机性能会对其他虚拟化功能造成负面影响。例如，它可以使迁移修改过的虚拟机更为困难。

13.2. 使用 TUNED 优化虚拟机性能

TuneD 工具是一种调优配置文件交付机制，能够让 RHEL 适应某些工作负载特性，如 CPU 密集型任务或存储网络吞吐量响应的需求。它提供很多预先配置的调优配置文件，以便在多个特定用例中增强性能并降低功耗。您可以编辑这些配置集，或创建新配置集来创建适合您的环境的性能解决方案，包括虚拟环境。

要针对虚拟化优化 RHEL 9，请使用以下配置集：

- 对于 RHEL 9 虚拟机，请使用 virtual-guest 配置集。它基于通常适用的 *throughput-performance* 配置集，但也会降低虚拟内存的交换性。
- 对于 RHEL 9 虚拟化主机，请使用 virtual-host 配置集。这可提高脏内存页面的主动回写，这有助于主机性能。

先决条件

- TuneD 服务已安装并启用。

流程

启用特定的 TuneD 配置集：

1. 列出可用的 TuneD 配置集。

```
# tuned-adm list

Available profiles:
- balanced             - General non-specialized TuneD profile
- desktop              - Optimize for the desktop use-case
[...]
- virtual-guest        - Optimize for running inside a virtual guest
- virtual-host         - Optimize for running KVM guests
Current active profile: balanced
```

2. 可选：创建一个新的 TuneD 配置集或编辑现有的 TuneD 配置集。

   如需更多信息，请参阅自定义 TuneD 配置集。

3. 激活 TuneD 配置集。

```
# tuned-adm profile selected-profile
```

   - 要优化虚拟化主机，请使用 *virtual-host* 配置集。

```
# tuned-adm profile virtual-host
```

   - 在 RHEL 虚拟机操作系统中，使用 *virtual-guest* 配置集。

```
# tuned-adm profile virtual-guest
```

其他资源

- 监控和管理系统状态和性能

13.3. 优化 LIBVIRT 守护进程

libvirt 虚拟化套件作为 RHEL hypervisor 的管理层，libvirt 配置对您的虚拟化主机有很大影响。值得注意的是，RHEL 9 包含两种不同类型的 libvirt 守护进程，即单体或模块化，您使用的守护进程类型会影响您可以配置单独的虚拟化驱动程序。

13.3.1. libvirt 守护进程的类型

RHEL 9 支持以下 libvirt 守护进程类型：

单体 libvirt

传统的 libvirt 守护进程 libvirtd 通过使用单个配置文件 - /etc/libvirt/libvirtd.conf 控制各种虚拟化驱动程序。

因此，libvirtd 允许使用集中的虚拟机监控程序配置，但可能会导致对系统资源的使用效率低。因此，libvirtd 在以后的 RHEL 主发行版本中将不被支持。

但是，如果您从 RHEL 8 更新至 RHEL 9，您的主机仍然默认使用 libvirtd。

模块 libvirt

RHEL 9 中新引入的模块 libvirt 为各个虚拟化驱动程序提供一个特定的守护进程。其中包括：

- virtqemud - hypervisor 管理的主要守护进程
- virtinterfaced - 用于主机 NIC 管理的辅助守护进程
- virtnetworkd - 用于虚拟网络管理的辅助守护进程
- virtnodedevd - 主机物理设备管理的辅助守护进程
- virtnwfilterd - 主机防火墙管理的辅助守护进程

- virtsecretd - 用于主机 secret 管理的辅助守护进程
- virtstoraged - 用于存储管理的辅助守护进程

每个守护进程都有单独的配置文件 - 例如 /etc/libvirt/virtqemud.conf。因此，模块化的 libvirt 守护进程可以为调优 libvirt 资源管理提供更好的选项。

如果您执行了全新的 RHEL 9 安装，则会默认配置模块化的 libvirt。

后续步骤

- 如果您的 RHEL 9 使用 libvirtd，红帽建议切换到模块化守护进程。具体步骤请参阅启用模块化 libvirt 守护进程。

13.3.2. 启用模块化 libvirt 守护进程

在 RHEL 9 中，libvirt 库使用 modular 守护进程来处理您主机上的单个虚拟化驱动程序集。例如，virtqemud 守护进程处理 QEMU 驱动程序。

如果您执行了 RHEL 9 主机的全新安装，您的虚拟机监控程序默认使用模块化 libvirt 守护进程。但是，如果您将主机从 RHEL 8 升级到 RHEL 9，您的管理程序将使用单体 libvirtd 守护进程，这是 RHEL 8 中的默认设置。

如果是这种情况，红帽建议改为启用模块 libvirt 守护进程，因为它们为微调 libvirt 资源管理提供了更好的选项。另外，libvirtd 在以后的 RHEL 主发行版本中将不被支持。

先决条件

- 您的管理程序使用单体的 libvirtd 服务。

```
# systemctl is-active libvirtd.service
active
```

如果这个命令显示 active，则代表在使用 libvirtd。

- 您的虚拟机已关闭。

流程

1. 停止 libvirtd 及其套接字。

```
$ systemctl stop libvirtd.service
$ systemctl stop libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

2. 禁用 libvirtd 以防止它在引导时启动。

```
$ systemctl disable libvirtd.service
$ systemctl disable libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

3. 启用模块 libvirt 守护进程。

```
# for drv in qemu interface network nodedev nwfilter secret storage; do systemctl unmask virt${drv}d.service; systemctl unmask virt${drv}d{,-ro,-admin}.socket; systemctl enable virt${drv}d.service; systemctl enable virt${drv}d{,-ro,-admin}.socket; done
```

4. 启动模块守护进程的套接字。

```
# for drv in qemu network nodedev nwfilter secret storage; do systemctl start virt${drv}d{,-ro,-admin}.socket; done
```

5. 可选：如果您需要从远程主机连接到主机，请启用并启动虚拟化代理守护进程。

   a. 检查libvirtd-tls.socket 服务是否已在系统上启用了。

   ```
   # grep listen_tls /etc/libvirt/libvirtd.conf

   listen_tls = 0
   ```

   b. 如果 libvirtd-tls.socket 没启用(listen_tls = 0)，请激活 virtproxyd，如下所示：

   ```
   # systemctl unmask virtproxyd.service
   # systemctl unmask virtproxyd{,-ro,-admin}.socket
   # systemctl enable virtproxyd.service
   ```

```
# systemctl enable virtproxyd{,-ro,-admin}.socket
# systemctl start virtproxyd{,-ro,-admin}.socket
```

c. 如果 libvirtd-tls.socket启用了(listen_tls = 1)，请激活 virtproxyd，如下所示：

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin,-tls}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin,-tls}.socket
# systemctl start virtproxyd{,-ro,-admin,-tls}.socket
```

要启用 virtproxyd 的 TLS 套接字，您的主机必须配置了 TLS 证书，以与 libvirt 一起工作。如需更多信息，请参阅 上游 libvirt 文档。

验证

1. 激活已启用的虚拟化守护进程。

```
# virsh uri
qemu:///system
```

2. 验证您的主机是否在使用 virtqemud 模块守护进程。

```
# systemctl is-active virtqemud.service
active
```

如果状态是 active，则您已成功启用了模块 libvirt 守护进程。

13.4. 配置虚拟机内存

要提高虚拟机 (VM) 的性能，您可以将额外的主机 RAM 分配给虚拟机。类似地，您可以减少分配给虚拟机的内存量，从而使主机内存可以分配给其他虚拟机或任务。

要执行这些操作，您可以使用 Web 控制台 或 命令行界面。

13.4.1. 使用 web 控制台添加和删除虚拟机内存

要提高虚拟机 (VM) 的性能或释放它使用的主机资源，您可以使用 Web 控制台来调整分配给虚拟机的内存量。

先决条件

- 客户端操作系统正在运行内存 balloon 驱动程序。请执行以下命令校验：

1. 确保虚拟机的配置包含 memballoon 设备：

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

如果此命令显示任何输出，并且型号未设置为 none，则存在 memballoon 设备。

2. 确保 balloon 驱动程序在客户机操作系统中运行。

   - 在 Windows 客户机中，驱动程序作为 virtio-win 驱动程序软件包的一部分安装。具体步骤请参阅为 Windows 虚拟机安装半虚拟化 KVM 驱动程序。

   - 在 Linux 客户机中，通常默认包含驱动程序，并在存在 memballoon 设备时激活。

- Web 控制台 VM 插件已安装在您的系统上。

流程

1. 可选：包含有关虚拟机最大内存和当前使用的内存的信息。这将作为您更改的基准，并进行验证。

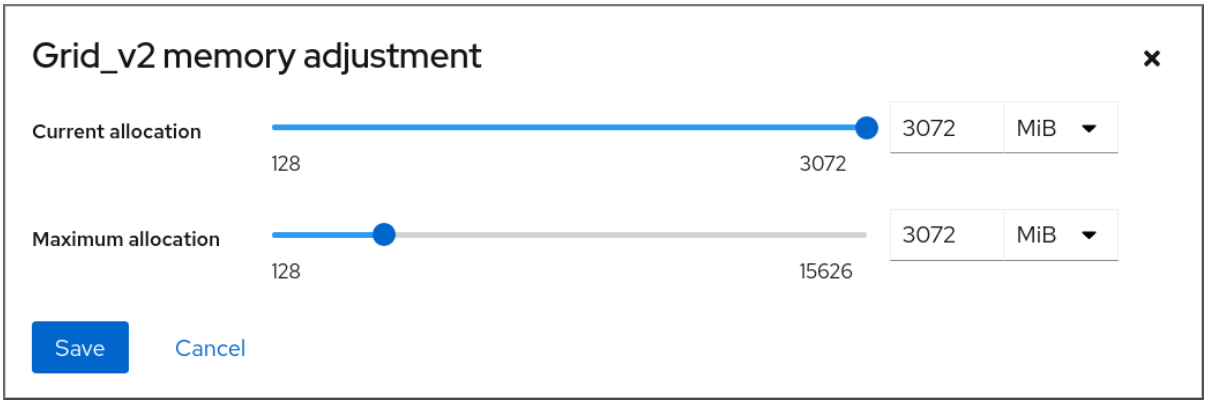
```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. 在 Virtual Machines 界面中，点您要查看其信息的虚拟机。

此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

3. 点概述窗格中 Memory 行旁边的 edit。

   此时将显示 Memory Adjustment 对话框。



4. 为所选的虚拟机配置虚拟内存。

   - 最大分配 - 设置虚拟机可用于其进程的最大主机内存量。您可以在创建虚拟机时指定最大内存，或可以在以后增大。您可以将内存指定为 MiB 或 GiB 的倍数。

     只有在关闭虚拟机上才能调整最大内存分配。

   - 当前分配 - 设置分配给虚拟机的实际内存量。这个值可以小于最大分配量，但不能超过它。您可以调整值，来控制虚拟机的进程可使用的内存。您可以将内存指定为 MiB 或 GiB 的倍数。

     如果没有指定这个值，则默认分配是 Maximum allocation 值。

5. 点 Save。

   调整了虚拟机的内存分配。

其他资源

- 使用命令行界面添加和删除虚拟机内存
- 优化虚拟机 CPU 性能

13.4.2. 使用命令行界面添加和删除虚拟机内存

若要提高虚拟机 (VM) 的性能或释放其使用的主机资源，您可以使用 CLI 来调整分配给虚拟机的内存量。

先决条件

- 客户端操作系统正在运行内存 balloon 驱动程序。请执行以下命令校验：

1. 确保虚拟机的配置包含 memballoon 设备：

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

如果此命令显示任何输出，并且型号未设置为 none，则存在 memballoon 设备。

2. 确定 ballon 驱动程序正在客户端操作系统中运行。

   - 在 Windows 客户机中，驱动程序作为 virtio-win 驱动程序软件包的一部分安装。具体步骤请参阅为 Windows 虚拟机安装半虚拟化 KVM 驱动程序。

   - 在 Linux 客户机中，通常默认包含驱动程序，并在存在 memballoon 设备时激活。

流程

1. 可选：包含有关虚拟机最大内存和当前使用的内存的信息。这将作为您更改的基准，并进行验证。

```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. 调整分配给虚拟机的最大内存。增加这个值可以提高虚拟机的性能风险，降低这个值会降低虚拟机在主机上的性能占用空间。请注意，此更改只能在关闭的虚拟机上执行，因此调整正在运行的虚拟机需要重新启动才能生效。

例如，将 *testguest* 虚拟机可以使用的最大内存更改为 4096 MiB：

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

要增加正在运行的虚拟机的最大内存，您可以将内存设备附加到虚拟机。这也被称为内存热插拔。详情请参阅将设备附加到虚拟机。

警告

不支持从正在运行的虚拟机中删除内存设备（也称为内存热插拔），因此红帽强烈不鼓励这样做。

3. 可选：您还可以调整虚拟机当前使用的内存，最多不超过最大分配数。这调整了虚拟机在主机上的内存负载，直到下一次重启为止，而不需要更改最大的虚拟机分配。

```
# virsh setmem testguest --current 2048
```

验证

1. 确认虚拟机使用的内存已更新：

```
# virsh dominfo testguest
Max memory:     4194304 KiB
Used memory:    2097152 KiB
```

2. 可选：如果您调整了当前虚拟机内存，您可以获取虚拟机的内存 balloon 统计，以评估它如何有效地控制其内存使用量。

```
 # virsh domstats --balloon testguest
Domain: 'testguest'
  balloon.current=365624
  balloon.maximum=4194304
  balloon.swap_in=0
  balloon.swap_out=0
  balloon.major_fault=306
```

```
balloon.minor_fault=156117
balloon.unused=3834448
balloon.available=4035008
balloon.usable=3746340
balloon.last-update=1587971682
balloon.disk_caches=75444
balloon.hugetlb_pgalloc=0
balloon.hugetlb_pgfail=0
balloon.rss=1005456
```

其他资源

- 使用 web 控制台添加和删除虚拟机内存
- 优化虚拟机 CPU 性能

13.4.3. 使用 virtio-mem 添加和删除虚拟机内存

RHEL 9 提供 virtio-mem 半虚拟化内存设备。此设备使得可以在虚拟机(VM)中动态添加或删除主机内存。例如，您可以使用 virtio-mem 在正在运行的虚拟机之间移动内存资源，或者根据您的当前要求在云设置中调整虚拟机内存大小。

13.4.3.1. virtio-mem 概述

virtio-mem 是一个半虚拟化内存设备，可用于在虚拟机中动态添加或删除主机内存。例如，您可以使用这个设备在运行的虚拟机之间移动内存资源，或者根据您当前的要求在云设置中调整虚拟机内存大小。

通过使用 virtio-mem，您可以将虚拟机的内存增加到其初始大小，并将其缩小到其原始大小，单位是 4 到几百字节(MiB)。但请注意，virtio-mem 也依赖于特定的客户机操作系统配置，特别是为了可靠地拔出内存。

virtio-mem 功能限制

virtio-mem 目前与以下功能不兼容：

- 对主机上的实时应用程序使用内存锁定
- 在主机上使用加密的虚拟化

- 在主机上将 virtio-mem 与 memballoon 膨胀和收缩合并
- 在虚拟机中卸载或重新载入 virtio_mem 驱动程序
- 使用 vhost-user 设备，但 virtiofs除外

其他资源

- 在虚拟机中配置内存在线
- 将 virtio-mem 设备附加到虚拟机

13.4.3.2. 在虚拟机中配置内存在线

在使用 virtio-mem 将内存附加到正在运行的虚拟机（也称为内存热插拔）之前，您必须配置虚拟机(VM)操作系统，以便可将热插拔内存自动设置为在线状态。否则，客户端操作系统无法使用额外的内存。您可以从以下内存在线的配置中选择：

- online_movable
- online_kernel
- auto-movable

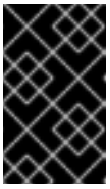
要了解这些配置之间的区别，请参阅： 简化配置的内存比较

RHEL 中默认内存在线是使用 udev 规则配置的。但是，在使用 virtio-mem 时，建议直接在内核中配置内存在线。

先决条件

- 主机具有 Intel 64 或 AMD64 CPU 架构。

- 主机使用 RHEL 9.4 或更高版本作为操作系统。
- 主机上运行的虚拟机使用以下操作系统版本之一：
  - RHEL 8.10

    重要

    在 RHEL 8.10 虚拟机中默认禁用从正在运行的虚拟机中拔出内存。
  - RHEL 9

步骤

- 要在虚拟机中设置内存在线，以使用 online_movable 配置：
  1. 将 memhp_default_state 内核命令行参数设置为 online_movable ：

     ```
     # grubby --update-kernel=ALL --remove-args=memhp_default_state --
     args=memhp_default_state=online_movable
     ```
  2. 重启虚拟机。
- 要在虚拟机中设置内存在线，以使用 online_kernel 配置：
  1. 将 memhp_default_state 内核命令行参数设置为 online_kernel ：

     ```
     # grubby --update-kernel=ALL --remove-args=memhp_default_state --
     args=memhp_default_state=online_kernel
     ```
  2. 重启虚拟机。

- 要在虚拟机中使用 **auto-movable** 内存在线策略：

1. 将 **memhp_default_state** 内核命令行参数设置为 **online** ：

```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --
args=memhp_default_state=online
```

2. 将 **memory_hotplug.online_policy** 内核命令行参数设置为 **auto-movable** ：

```
# grubby --update-kernel=ALL --remove-args="memory_hotplug.online_policy" --
args=memory_hotplug.online_policy=auto-movable
```

3. 可选：要进一步调整 **auto-movable** 在线策略，请更改 **memory_hotplug.auto_movable_ratio** 和 **memory_hotplug.auto_movable_numa_aware** 参数：

```
# grubby --update-kernel=ALL --remove-args="memory_hotplug.auto_movable_ratio" --
args=memory_hotplug.auto_movable_ratio=<percentage>

# grubby --update-kernel=ALL --remove-
args="memory_hotplug.memory_auto_movable_numa_aware" --
args=memory_hotplug.auto_movable_numa_aware=<y/n>
```

   - 与可用于任何分配的内存相比，**memory_hotplug.auto_movable_ratio** 参数设置仅可用于可移动分配的最大内存比率。比率以百分比表示，默认值为 ***301*** **(%)**，它是 **3:1** 的比率。

   - **memory_hotplug.auto_movable_numa_aware** 参数控制 **memory_hotplug.auto_movable_ratio** 参数是否应用到跨所有可用 **NUMA** 节点的内存，或仅应用到单个 **NUMA** 节点上的内存。默认值为：***y*** **(yes)**

     例如，如果最大比率设置为 **301%**，并且 **memory_hotplug.auto_movable_numa_aware** 设置为 ***y*** **(yes)**，即使在附加了 **virtio-mem** 设备的 **NUMA** 节点内也应用 **3:1** 比率。如果参数设置为 ***n*** **(no)**，则最大 **3:1** 比率仅应用于整个 **NUMA** 节点。

     另外，如果没有超过比率，则新热插拔内存将只可用于可移动分配。否则，新热拔插内存将同时可用于可移动和不可移动分配。

4. **重启虚拟机。**

验证

- **要查看 online_movable 配置是否已正确设置，请检查 memhp_default_state 内核参数的当前值：**

```
# cat /sys/devices/system/memory/auto_online_blocks
online_movable
```

- **要查看 online_kernel 配置是否已正确设置，请检查 memhp_default_state 内核参数的当前值：**

```
# cat /sys/devices/system/memory/auto_online_blocks
online_kernel
```

- **要查看 auto-movable 配置是否已正确设置，请检查以下内核参数：**
  - **memhp_default_state ：**

```
# cat /sys/devices/system/memory/auto_online_blocks
online
```

  - **memory_hotplug.online_policy:**

```
# cat /sys/module/memory_hotplug/parameters/online_policy
auto-movable
```

  - **memory_hotplug.auto_movable_ratio:**

```
# cat /sys/module/memory_hotplug/parameters/auto_movable_ratio
301
```

- memory_hotplug.auto_movable_numa_aware:

```
# cat /sys/module/memory_hotplug/parameters/auto_movable_numa_aware
y
```

其他资源

- virtio-mem 概述
- 将 virtio-mem 设备附加到虚拟机
- 配置内存热插拔

13.4.3.3. 将 virtio-mem 设备附加到虚拟机

要将额外内存附加到正在运行的虚拟机（也称为内存热插拔），之后能够调整热插拔内存的大小，您可以使用 virtio-mem 设备。特别是，您可以使用 libvirt XML 配置文件和 virsh 命令来定义并将 virtio-mem 设备附加到虚拟机(VM)。

先决条件

- 主机具有 Intel 64 或 AMD64 CPU 架构。
- 主机使用 RHEL 9.4 或更高版本作为操作系统。
- 主机上运行的虚拟机使用以下操作系统版本之一：
  - RHEL 8.10

重要

在 RHEL 8.10 虚拟机中默认禁用从正在运行的虚拟机中拔出内存。

- RHEL 9

- 虚拟机配置了内存在线。具体说明，请参阅：在虚拟机中配置内存在线

流程

1. 确保目标虚拟机的 XML 配置包含 maxMemory 参数：

```
# virsh edit testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <maxMemory unit='GiB'>128</maxMemory>
  ...
</domain>
```

在本例中，testguest1 虚拟机的 XML 配置定义了一个 maxMemory 参数，其大小为 128 gibibyte (GiB)。maxMemory 大小指定虚拟机可以使用的最大内存，其包括初始内存和热插拔内存。

2. 创建并打开 XML 文件，来在主机上定义 virtio-mem 设备，例如：

```
# vim virtio-mem-device.xml
```

3. 在文件中添加 virtio-mem 设备的 XML 定义并保存：

```
<memory model='virtio-mem'>
      <target>
            <size unit='GiB'>48</size>
            <node>0</node>
            <block unit='MiB'>2</block>
            <requested unit='GiB'>16</requested>
            <current unit='GiB'>16</current>
      </target>
      <alias name='ua-virtiomem0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
</memory>
<memory model='virtio-mem'>
      <target>
            <size unit='GiB'>48</size>
            <node>1</node>
            <block unit='MiB'>2</block>
            <requested unit='GiB'>0</requested>
```

```
            <current unit='GiB'>0</current>
        </target>
        <alias name='ua-virtiomem1'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</memory>
```

在本例中，使用以下参数定义了两个 virtio-mem 设备：

- **size** ：这是设备的最大大小。在示例中，它是 48 GiB。size 必须是 block 大小的倍数。
- **node** ：这是为 virtio-mem 设备分配的 vNUMA 节点。
- **block** ：这是设备的块大小。它必须至少是 Transparent Huge Page (THP)的大小，在 Intel 64 或 AMD64 CPU 架构上它是 2 MiB。Intel 64 或 AMD64 架构上的 2 MiB 块大小通常是好的默认选择。当使用带有 *虚拟功能 I/O(VFIO)* 或 *中介设备(mdev)* 的 virtio-mem 时，跨所有 virtio-mem 设备的总块数不能大于 32768，否则 RAM 的插入可能会失败。
- **requested** ：这是您附加到具有 virtio-mem 设备的虚拟机的内存量。但是，它只对虚拟机的请求，可能无法成功解决，例如如果虚拟机没有正确配置。requested 大小必须是 block 大小的倍数，且不能超过定义的 size最大值。
- **current**：这代表提供给虚拟机的当前 virtio-mem 设备的大小。current 大小可能与 requested 不同，例如当请求无法完成或重启虚拟机时。
- **alias** ：这是一个可选d 用户定义的别名，您可用来指定预期的 virtio-mem 设备，例如使用 libvirt 命令编辑设备时。libvirt 中所有用户定义的别名必须以 *"ua-"* 前缀开头。

除了这些特定的参数外，libvirt 像处理任何其他 PCI 设备一样处理 virtio-mem 设备。

4. 使用 XML 文件将定义的 virtio-mem 设备附加到虚拟机。例如，要将 virtio-mem-device.xml 中定义的两个设备永久附加到正在运行的 testguest1 虚拟机：

```
# virsh attach-device testguest1 virtio-mem-device.xml --live --config
```

--live 选项仅将设备附加到正在运行的虚拟机，在引导间不持久。**--config** 选项使配置更改持久。您还可以将设备附加到没有 **--live** 选项的关闭的虚拟机。

5. 可选：要动态更改附加到正在运行的虚拟机的 virtio-mem 设备的 requested 大小，请使用 virsh update-memory-device 命令：

```
# virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 4GiB
```

在本例中：

- testguest1 是您要更新的虚拟机。
- --alias ua-virtiomem0 是之前由定义的别名指定的 virtio-mem 设备。
- --requested-size 4GiB 将 virtio-mem 设备的 requested 大小更改为 4 GiB。

警告

通过减少 请求的大小，从正在运行的虚拟机中拔下内存可能不可靠。此过程是否成功取决于各种因素，如所使用的内存策略。

在某些情况下，客户机操作系统无法成功完成请求，因为目前无法更改热插内存量。

另外，在 RHEL 8.10 虚拟机中默认禁用从正在运行的虚拟机中拔出内存。

6. 可选 ： 要从关闭的虚拟机中拔出 virtio-mem 设备，请使用 virsh detach-device 命令：

```
# virsh detach-device testguest1 virtio-mem-device.xml
```

7. 可选 ：从正在运行的虚拟机中拔出 virtio-mem 设备：

a.

将 virtio-mem 设备的请求大小改为 0，否则尝试从正在运行的虚拟机中拔出 virtio-mem 设备将失败。

```
# virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 0
```

b. 从正在运行的虚拟机中拔出 virtio-mem 设备：

```
# virsh detach-device testguest1 virtio-mem-device.xml
```

验证

- 在虚拟机中，检查可用的 RAM ,并查看总内存现在是否包含热插拔内存：

```
# free -h

       total    used    free   shared  buff/cache   available
Mem:    31Gi     5.5Gi   14Gi   1.3Gi   11Gi         23Gi
Swap:   8.0Gi    0B      8.0Gi
```

```
# numactl -H

available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 29564 MB
node 0 free: 13351 MB
node distances:
node   0
  0:  10
```

- 也可以通过显示正在运行的虚拟机的 XML 配置来查看主机上当前的插入的 RAM 量：

```
# virsh dumpxml testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <currentMemory unit='GiB'>31</currentMemory>
  ...
  <memory model='virtio-mem'>
     <target>
       <size unit='GiB'>48</size>
       <node>0</node>
       <block unit='MiB'>2</block>
       <requested unit='GiB'>16</requested>
       <current unit='GiB'>16</current>
     </target>
     <alias name='ua-virtiomem0'/>
```

```
    <address type='pci' domain='0x0000' bus='0x08' slot='0x00' function='0x0'/>
  ...
</domain>
```

在本例中：

- <currentMemory unit='GiB'>31</currentMemory> 代表虚拟机中所有源的可用 RAM 总量。
- <current unit='GiB'>16</current> 代表 virtio-mem 设备提供的插入的 RAM 的当前大小。

其他资源

- virtio-mem 概述
- 在虚拟机中配置内存在线

13.4.3.4. 内存在线配置的比较

将内存附加到正在运行的 RHEL 虚拟机（也称为内存热插拔）时，您必须将热插内存设置为虚拟机 (VM)操作系统的在线状态。否则，系统将无法使用内存。

下表总结了在可用内存在线配置间选择时的主要注意事项。

表 13.1. 内存在线配置的比较

配置名称	从虚拟机中拔出内存	创建内存区不平衡的风险	一个潜在的用例	预期工作负载的内存要求
online_movable	热插内存可以可靠地拔出。	是	热插拔相对较少的内存	主要是用户空间内存
auto-movable	热插内存的可移动部分可以可靠地拔出。	最小	热插拔大量内存	主要是用户空间内存
online_kernel	热插拔内存无法可靠地拔出。	否	不可靠内存拔出是可以接受的。	用户空间或内核空间内存

区域不平衡 是某个 Linux 内存区域中缺少可用内存页。*区域不平衡* 会对系统性能造成负面影响。例如，如果内核用完了不可移动分配的可用内存，则内核可能会崩溃。通常，可移动分配主要包含用户空间内存页，不可移动分配主要包含内核空间内存页。

其他资源

- 在线和离线内存块
- 区域不平衡
- 在虚拟机中配置内存在线

13.4.4. 其他资源

- 将设备附加到虚拟机将设备附加到虚拟机。

13.5. 优化虚拟机 I/O 性能

虚拟机 (VM) 的输入和输出 (I/O) 能力可能会显著限制虚拟机的整体效率。要解决这个问题，您可以通过配置块 I/O 参数来优化虚拟机的 I/O。

13.5.1. 在虚拟机中调整块 I/O

当一个或多个虚拟机正在使用多个块设备时,可能需要通过修改虚拟设备的 I/O 优先级来调整虚拟设备的 *I/O 权重*。

增加设备的 I/O 权重会增加设备的 I/O 带宽的优先级，从而为它提供更多主机资源。同样的，降低设备的权重可使其消耗较少的主机资源。

注意

每个设备的 weight 值必须在 100 到 1000 之间。或者，该值可以是 0，它会从每个设备列表中删除该设备。

步骤

显示和设置虚拟机的块 I/O 参数：

1. 显示虚拟机当前的 <blkio> 参数：

```
# virsh dumpxml VM-name
```

```
<domain>
 [...]
 <blkiotune>
  <weight>800</weight>
  <device>
   <path>/dev/sda</path>
   <weight>1000</weight>
  </device>
  <device>
   <path>/dev/sdb</path>
   <weight>500</weight>
  </device>
 </blkiotune>
 [...]
</domain>
```

2. 编辑指定设备的 I/O 加权：

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

例如，以下命令将 *testguest1* 虚拟机中 */dev/sda* 设备的权重改为 **500**。

```
# virsh blkiotune testguest1 --device-weights /dev/sda, 500
```

13.5.2. 虚拟机中的磁盘 I/O 节流

当多个虚拟机同时运行时，它们可能会使用过量的磁盘 I/O 而干扰系统性能。KVM 虚拟化中的磁盘 I/O 节流使得能够对从虚拟机发送到主机的磁盘 I/O 请求设定限制。这可以防止虚拟机过度使用共享资源并影响其他虚拟机的性能。

要启用磁盘 I/O 节流，请对从附加到虚拟机的每个块设备发送给主机的磁盘 I/O 请求设置限制。

步骤

1.

1. 使用 virsh domblklist 命令列出指定虚拟机上所有磁盘设备的名称。

```
# virsh domblklist rollin-coal
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rollin-coal.qcow2
sda        -
sdb        /home/horridly-demanding-processes.iso
```

2. 找到您要节流的虚拟磁盘挂载的主机块设备。

例如，如果您想要从上一步中节流 sdb 虚拟磁盘，以下输出显示该磁盘挂载在 /dev/nvme0n1p3 分区上。

```
$ lsblk
NAME                                          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
zram0                                         252:0    0     4G  0 disk  [SWAP]
nvme0n1                                         259:0    0 238.5G  0 disk
├─nvme0n1p1                                     259:1    0   600M  0 part  /boot/efi
├─nvme0n1p2                                     259:2    0     1G  0 part  /boot
└─nvme0n1p3                                     259:3    0 236.9G  0 part
  └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fea 253:0    0 236.9G  0 crypt /home
```

3. 使用 virsh blkiotune 命令为块设备设置 I/O 限制。

```
# virsh blkiotune VM-name --parameter device,limit
```

以下示例将 rollin-coal 上的 sdb 磁盘节流为每秒 1000 个读写 I/O 操作，每秒的读写 I/O 操作吞吐量 50 MB。

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-
write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec
/dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

附加信息

- 磁盘 I/O 节流可用于各种情况，例如，当属于不同客户的虚拟机在同一台主机上运行时，或者为不同的虚拟机提供服务质量保障时。磁盘 I/O 节流还可用来模拟较慢的磁盘。

- I/O 节流可以独立应用于附加到虚拟机的每个块设备，并支持对吞吐量和 I/O 操作的限制。

- 红帽不支持使用 virsh blkdeviotune命令来在 VM 中配置 I/O 节流。有关使用 RHEL 9 作为虚拟机主机时不支持的功能的更多信息，请参阅 RHEL 9 虚拟化中不支持的功能。

13.5.3. 启用多队列 virtio-scsi

在虚拟机 (VM) 中使用 virtio-scsi 存储设备时，*multi-queue virtio-scsi* 特性可提高存储性能和可扩展性。它允许每个虚拟 CPU (vCPU) 使用单独的队列和中断，而不影响其他 vCPU。

流程

- 要为特定虚拟机启用 multi-queue virtio-scsi 支持，请在虚拟机的 XML 配置中添加以下内容，其中 *N* 是 vCPU 队列的总数：

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

13.6. 优化虚拟机 CPU 性能

与主机中的物理 CPU 非常相似，vCPU 对虚拟机 (VM) 性能至关重要。因此，优化 vCPU 会对虚拟机的资源效率产生重大影响。优化 vCPU：

1. 调整分配给虚拟机的主机 CPU 数。您可以使用 CLI 或 Web 控制台进行此操作。

2. 确保 vCPU 模型与主机的 CPU 型号一致。例如，将 *testguest1* 虚拟机设置为使用主机的 CPU 型号：

```
# virt-xml testguest1 --edit --cpu host-model
```

   在 ARM 64 系统上，请使用 **--cpu host-passthrough**。

3. 管理内核相同的页面合并 (KSM)。

4. 如果您的主机使用非统一内存访问 (NUMA)，您也可以为其虚拟机 配置 NUMA。这会尽可能将主机的 CPU 和内存进程映射到虚拟机的 CPU 和内存进程上。实际上，NUMA 调优为 vCPU 提供了对分配给虚拟机的系统内存更精简的访问，这可以提高 vCPU 的处理效率。

详情请参阅 在虚拟机中配置 NUMA 和 vCPU 性能调整场景示例 。

13.6.1. 使用命令行界面添加和删除虚拟 CPU

要提高或优化虚拟机 (VM) 的 CPU 性能，您可以添加或删除分配给虚拟机的虚拟 CPU (vCPU) 。

当在运行的虚拟机上执行时，这也被称为 vCPU 热插和热拔。但请注意，RHEL 9 不支持 vCPU 热拔，红帽不建议使用它。

先决条件

- 可选：查看目标虚拟机中的 vCPU 的当前状态。例如，显示 *testguest* 虚拟机上的 vCPU 数量：

```
# virsh vcpucount testguest
maximum      config         4
maximum      live           2
current      config         2
current      live           1
```

此输出显示 *testguest* 目前使用 1 个 vCPU，另外 1 个 vCPu 可以热插入以提高虚拟机性能。但是，重新引导后，vCPU *testguest* 使用的数量会改为 2，而且能够热插 2 个 vCPU。

流程

1. 调整可以附加到虚拟机的最大 vCPU 数量，其在虚拟机下次启动时生效。

   例如，要将 *testguest* 虚拟机的最大 vCPU 数量增加到 8:

   ```
   # virsh setvcpus testguest 8 --maximum --config
   ```

   请注意，最大值可能会受 CPU 拓扑、主机硬件、hypervisor 和其他因素的限制。

2. 将当前附加到虚拟机的 vCPU 数量调整到上一步中配置的最大值。例如：

   - 将附加到正在运行的 *testguest* 虚拟机的 vCPU 数量增加到 4:

```
# virsh setvcpus testguest 4 --live
```

这会增加虚拟机的性能和主机的 ***testguest*** 负载占用，直到虚拟机下次引导为止。

- 将附加到 ***testguest*** 虚拟机的 vCPU 数量永久减少至 1：

```
# virsh setvcpus testguest 1 --config
```

这会降低虚拟机的性能和 ***testguest*** 的主机负载占用。但是，如果需要可热插入虚拟机以暂时提高性能。

验证

- 确认虚拟机的 vCPU 的当前状态反映了您的更改。

```
# virsh vcpucount testguest
maximum      config         8
maximum      live           4
current      config         1
current      live           4
```

其他资源

- 使用 Web 控制台管理虚拟 CPU

13.6.2. 使用 Web 控制台管理虚拟 CPU

通过使用 RHEL 9 web 控制台，您可以查看并配置 web 控制台连接的虚拟机所使用的虚拟 CPU。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

步骤

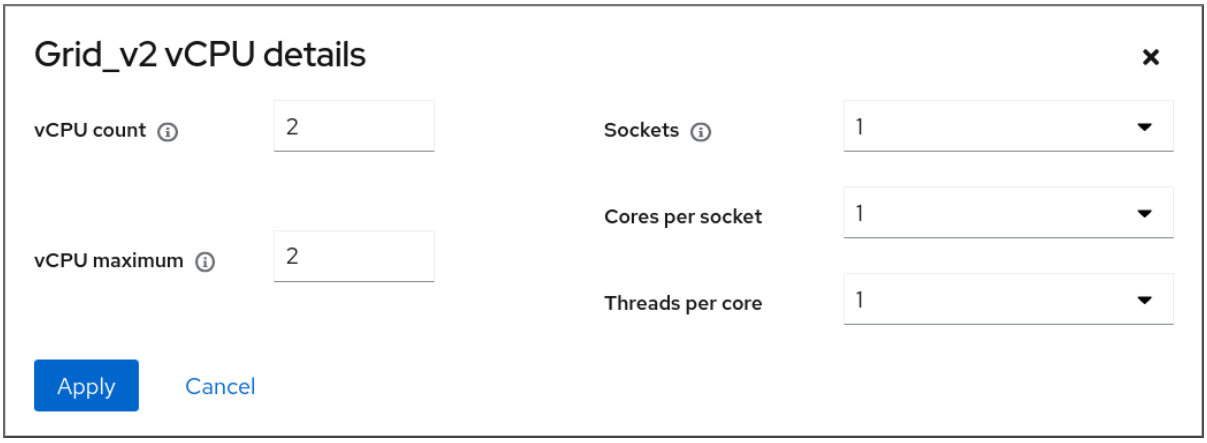
1. 在 Virtual Machines 界面中，点您要查看其信息的虚拟机。

此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟

机的图形界面的控制台部分。

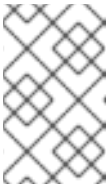
2. 点概述窗格中 vCPU 数旁边的 edit。

此时会出现 vCPU 详情对话框。



1. 为所选虚拟机配置虚拟 CPU。

- vCPU 数量 - 当前正在使用的 vCPU 数量。



注意

vCPU 数量不能超过 vCPU 的最大值。

- vCPU 最大 - 可为虚拟机配置的最大虚拟 CPU 数。如果这个值大于 vCPU Count，可以为虚拟机附加额外的 vCPU。
- 插槽 - 向虚拟机公开的插槽数量。
- 每个插槽的内核数 - 向虚拟机公开的每个插槽的内核数。
- 每个内核的线程数 - 向虚拟机公开的每个内核的线程数。

请注意， 插槽、每个插槽的内核数和每个内核的线程数选项调整了虚拟机的 CPU 拓

拓扑。这可能对 vCPU 性能有好处，但可能会影响客户机操作系统中某些软件的功能。如果您的部署不需要不同的设置，请保留默认值。

2. 点应用。

配置了虚拟机的虚拟 CPU。

注意

对虚拟 CPU 设置的更改仅在重启虚拟机后生效。

其他资源

- 使用命令行界面添加和删除虚拟 CPU

13.6.3. 在虚拟机中配置 NUMA

以下方法可用于在 RHEL 9 主机上配置虚拟机 (VM) 的非一致性内存访问 (NUMA) 设置。

先决条件

- 主机是一个与 NUMA 兼容的机器。要检测是否是这种情况，请使用 virsh nodeinfo 命令，并查看 NUMA cell (s) 行：

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              48
CPU frequency:       1200 MHz
CPU socket(s):       1
Core(s) per socket:  12
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         67012964 KiB
```

如果行的值为 2 或更高，则主机与 NUMA 兼容。

流程

为便于使用，您可以使用自动化工具和服务设置虚拟机的 NUMA 配置。但是，手动 NUMA 设置可能会显著提高性能。

自动方法

- 将虚拟机的 NUMA 策略设为 Preferred。例如，对于 *testguest5* 虚拟机要这样做：

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- 在主机上启用自动 NUMA 均衡：

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- 启动 numad 服务，来自动将 VM CPU 与内存资源对齐。

```
# systemctl start numad
```

手动方法

1. 将特定 vCPU 线程固定到特定主机 CPU 或者 CPU 范围。在非 NUMA 主机和虚拟机上也可以这样做，我们推荐您使用一种安全的方法来提高 vCPU 性能。

   例如，以下命令将 *testguest6* 虚拟机的 vCPU 线程 0 到 5 分别固定到主机 CPU 1、3、5、7、9 和 11：

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
# virsh vcpupin testguest6 4 9
# virsh vcpupin testguest6 5 11
```

   之后，您可以验证操作是否成功：

```
# virsh vcpupin testguest6
VCPU   CPU Affinity
----------------------
0      1
1      3
```

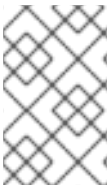
```
2     5
3     7
4     9
5     11
```

2. 固定 vCPU 线程后，您还可以将与指定虚拟机关联的 QEMU 进程线程固定到特定的主机 CPU 或 CPU 范围。例如：以下命令将 *testguest6* 的 QEMU 进程线程 固定到 CPU 13 和 15，确认成功：

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
----------------------------------
       *: 13,15
```

3. 最后，您也可以指定将哪些主机 NUMA 节点专门分配给某个虚拟机。这可提高虚拟机 vCPU 的主机内存用量。例如，以下命令将 *testguest6* 设置为使用主机 NUMA 节点 3 到 5，确认成功：

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```



注意

为了获得最佳性能，建议使用以上列出的所有手动调优方法

已知问题

- 目前无法在 IBM Z 主机上执行 NUMA 调整。

其他资源

- vCPU 性能调整场景示例
- 使用 numastat 程序查看系统的当前 NUMA 配置

13.6.4. vCPU 性能调整场景示例

要获得最佳 vCPU 性能，红帽建议同时使用手动 vcpupin、emulatorpin 和 numatune 设置，例如在以下场景中。

起始场景

- 您的主机有以下与硬件相关的信息：
  - 2 个 NUMA 节点
  - 每个节点上的 3 个 CPU 内核
  - 每个内核有 2 个线程

  此类机器的 virsh nodeinfo 输出类似于：

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              12
CPU frequency:       3661 MHz
CPU socket(s):       2
Core(s) per socket:  3
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         31248692 KiB
```

- 您打算将现有的虚拟机修改为有 8 个 vCPU，这意味着单个 NUMA 节点无法容纳它。

  因此，您应该在每个 NUMA 节点上分发 4 个 vCPU，并使 vCPU 拓扑尽可能接近主机拓扑。这意味着，作为给定物理 CPU 的同级线程运行的 vCPU 应该固定到同一核上的主机线程。详情请查看以下*解决方案*:

解决方案

1. 获取有关主机拓扑的信息：

```
# virsh capabilities
```

   输出应包含类似如下的部分：

```
<topology>
  <cells num="2">
```

```
    <cell id="0">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="10" />
        <sibling id="1" value="21" />
      </distances>
      <cpus num="6">
        <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
        <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
      </cpus>
    </cell>
    <cell id="1">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="21" />
        <sibling id="1" value="10" />
      </distances>
      <cpus num="6">
        <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
        <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
      </cpus>
    </cell>
  </cells>
</topology>
```

2. **可选：使用适用的工具和实用程序测试虚拟机的性能。**

3. **在主机上设置并挂载 1 GiB 巨页：**

注意

1 GiB 巨页可能在某些架构和配置上不提供，如 ARM 64 主机。

a. 在主机的内核命令行中添加以下行：

```
default_hugepagesz=1G hugepagesz=1G
```

b. 使用以下内容创建 /etc/systemd/system/hugetlb-gigantic-pages.service 文件：

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

c. 使用以下内容创建 /etc/systemd/hugetlb-reserve-pages.sh 文件：

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
 echo "ERROR: $nodes_path does not exist"
 exit 1
fi

reserve_pages()
{
 echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages 4 node1
reserve_pages 4 node2
```

这会从 *node1* 保留 4 个 1GiB 巨页，并在 *node2* 中保留 4 个 1GiB 巨页。

d. 使在上一步中创建的脚本可执行：

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. 在引导时启用巨页保留：

```
# systemctl enable hugetlb-gigantic-pages
```

4. 使用 virsh edit 命令编辑您要优化的虚拟机的 XML 配置，在本例中为 *super-VM* ：

```
# virsh edit super-vm
```

5. 用以下方法调整虚拟机的 XML 配置：

a. 将虚拟机设置为使用 8 个静态 vCPU。使用 <vcpu/> 元素来执行此操作。

b. 将每个 vCPU 线程固定到拓扑中镜像的对应主机 CPU 线程。为此，请在 <cputune> 部分中使用 <vcpupin/> 元素。

请注意，如上面 virsh capabilities 工具所示，主机 CPU 线程在各自的内核中不是按顺序排序的。此外，vCPU 线程应固定到同一 NUMA 节点上最多可用的主机核集合。有关表图，请查看以下 示例拓扑 部分。

步骤 a. 和 b. 的 XML 配置类似：

```
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
</cputune>
```

c. 将虚拟机设置为使用 1 GiB 巨页：

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

d.

配置虚拟机的 NUMA 节点，使其使用主机上对应的 NUMA 节点的内存。要做到这一点，请在 <numatune/> 部分中使用 <memnode/> 元素：

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e.

确保 CPU 模式设为 host-passthrough，且 CPU 在 passthrough 模式下使用缓存：

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
```

在 ARM 64 系统上，省略 <cache mode="passthrough"/> 行。

验证

1.

确认生成的虚拟机 XML 配置包含类似如下内容：

```
[...]
  <memoryBacking>
    <hugepages>
      <page size='1' unit='GiB'/>
    </hugepages>
  </memoryBacking>
  <vcpu placement='static'>8</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='1'/>
    <vcpupin vcpu='1' cpuset='4'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='5'/>
    <vcpupin vcpu='4' cpuset='7'/>
    <vcpupin vcpu='5' cpuset='10'/>
    <vcpupin vcpu='6' cpuset='8'/>
    <vcpupin vcpu='7' cpuset='11'/>
    <emulatorpin cpuset='6,9'/>
  </cputune>
  <numatune>
    <memory mode="preferred" nodeset="1"/>
    <memnode cellid="0" mode="strict" nodeset="0"/>
    <memnode cellid="1" mode="strict" nodeset="1"/>
  </numatune>
  <cpu mode="host-passthrough">
    <topology sockets="2" cores="2" threads="2"/>
    <cache mode="passthrough"/>
```

```
    <numa>
     <cell id="0" cpus="0-3" memory="2" unit="GiB">
      <distances>
       <sibling id="0" value="10"/>
       <sibling id="1" value="21"/>
      </distances>
     </cell>
     <cell id="1" cpus="4-7" memory="2" unit="GiB">
      <distances>
       <sibling id="0" value="21"/>
       <sibling id="1" value="10"/>
      </distances>
     </cell>
    </numa>
  </cpu>
</domain>
```

2. **可选：使用 适用的工具和实用程序测试虚拟机的性能，以评估虚拟机优化的影响。**

拓扑示例

- **下表演示了 vCPU 和主机 CPU 之间的连接：**

表 13.2. 主机拓扑

CPU 线程	0	3	1	4	2	5	6	9	7	10	8	11
内核	0		1		2		3		4		5	
插槽	0						1					
NUMA 节点	0						1					

表 13.3. VM 拓扑

vCPU 线程	0	1	2	3	4	5	6	7
内核	0		1		2		3	
插槽	0				1			
NUMA 节点	0				1			

表 13.4. 合并主机和虚拟机拓扑

vCPU 线程		0	1	2	3		4	5	6	7

主机 CPU 线程	0	3	1	4	2	5	6	9	7	10	8	11
内核	0		1		2		3		4		5	
插槽	0						1					
NUMA 节点	0						1					

在这种情况下，有 2 个 NUMA 节点和 8 个 vCPU。因此，应该为每个节点固定 4 个 vCPU 线程。

另外，红帽建议在每个节点上至少保留一个 CPU 线程用于主机系统操作。

因为在这个示例中，每个 NUMA 节点都有 3 个核，每个核都有 2 个主机 CPU 线程，节点 0 的设置转换如下：

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

13.6.5. 管理内核相同的页面合并

内核同页合并 (KSM) 通过在虚拟机 (VM) 间共享相同的内存页面来提高内存密度。但是，启用 KSM 会增加 CPU 使用率，并可能会根据工作负载对整体性能造成负面影响。

根据具体要求，您可以为单个会话启用或禁用 KSM，或者永久禁用 KSM。

注意

在 RHEL 9 及更高版本中，默认禁用 KSM。

先决条件

- 对主机系统有根访问权限。

流程

- **禁用 KSM：**

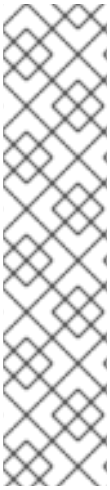
  - 要对单个会话停用 KSM，请使用 systemctl 工具停止 ksm 和 ksmtuned 服务。

```
# systemctl stop ksm

# systemctl stop ksmtuned
```

  - 要永久停用 KSM，请使用 systemctl 工具来禁用 ksm 和 ksmtuned 服务。

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```



注意

取消激活 KSM 前在虚拟机间共享的内存页将保持共享。要停止共享，请使用以下命令删除系统中的所有 PageKSM 页：

```
# echo 2 > /sys/kernel/mm/ksm/run
```

匿名页面替换了 KSM 页面后，khugepaged 内核服务将在虚拟机物理内存中重建透明的大页面。

- 启用 KSM：

警告

启用 KSM 会增加 CPU 使用率并影响整体 CPU 性能。

1. 安装 ksmtuned 服务：

```
# dnf install ksmtuned
```

2. 启动服务：

- 要为单个会话启用 KSM，请使用 systemctl 程序启动 ksm 和 ksmtuned 服务。

```
# systemctl start ksm
# systemctl start ksmtuned
```

- 要永久启用 KSM，请使用 systemctl 程序启用 ksm 和 ksmtuned 服务。

```
# systemctl enable ksm
Created symlink /etc/systemd/system/multi-user.target.wants/ksm.service →
/usr/lib/systemd/system/ksm.service

# systemctl enable ksmtuned
Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service →
/usr/lib/systemd/system/ksmtuned.service
```

13.7. 优化虚拟机网络性能

由于虚拟机的网络接口卡 (NIC) 的虚拟性质，虚拟机会丢失其分配的主机网络带宽的一部分，这可以降低虚拟机的整体工作负载效率。以下提示可最大程度降低虚拟化对虚拟 NIC (vNIC) 吞吐量的负面影响。

流程

使用以下任一方法并观察它是否对虚拟机网络性能有帮助：

启用 vhost_net 模块

在主机上，确保启用了 vhost_net 内核特性：

```
# lsmod | grep vhost
vhost_net           32768  1
vhost               53248  1 vhost_net
tap                 24576  1 vhost_net
tun                 57344  6 vhost_net
```

如果这个命令的输出为空，请启用 vhost_net 内核模块：

```
# modprobe vhost_net
```

设置 **multi-queue virtio-net**

要为虚拟机设置 ***multi-queue virtio-net*** 特性，请使用 **virsh edit** 命令来编辑虚拟机的 **XML** 配置。在 **XML** 中，将以下内容添加到 **<devices>** 部分，并将 **N** 替换为虚拟机中的 **vCPU** 个数，最多 **16** 个：

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

如果虚拟机正在运行，重启它以使更改生效。

批量网络数据包

在具有长传输路径的 **Linux VM** 配置中，在将数据包提交给内核之前对其进行批处理可以提高缓存利用率。要设置数据包批处理，请在主机上使用以下命令，并将 ***tap0*** 替换为虚拟机使用的网络接口的名称：

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

如果您的主机 **NIC** 支持 **SR-IOV**，请为您的 **vNIC** 使用 **SR-IOV** 设备分配。如需更多信息，请参阅管理 **SR-IOV** 设备。

其他资源

- 了解虚拟网络

13.8. 虚拟机性能监控工具

要确定什么消耗了最多的 **VM** 资源，以及虚拟机性能的哪方面需要优化，可以使用性能诊断工具，包括通用的和特定于虚拟机的。

默认操作系统性能监控工具

对于标准性能评估，您可以使用主机和虚拟机操作系统默认提供的实用程序：

- 在 RHEL 9 主机上，以 root 用户身份使用 top 实用程序或 系统监控 应用程序，并在输出中查找 qemu 和 virt。这显示了您的虚拟机消耗的主机系统资源量。

  - 如果监控工具显示任何 qemu 或 virt 进程消耗了大量的主机 CPU 或内存量，请使用 perf 工具进行调查。详情请查看以下信息。

  - 另外，如果 vhost_net 线程进程（如 *vhost_net-1234* ）被显示为消耗大量主机 CPU 容量，请考虑使用 虚拟网络优化功能，如 multi-queue virtio-net。

- 在客户机操作系统上，使用系统上可用的性能工具和应用程序来评估哪些进程消耗了最多的系统资源。

  - 在 Linux 系统上，您可以使用 top 工具。

  - 在 Windows 系统中，您可以使用 Task Manager 应用程序。

perf kvm

您可以使用 perf 实用程序收集有关 RHEL 9 主机性能的特定虚拟化统计。要做到这一点：

1. 在主机上安装 *perf* 软件包：

```
# dnf install perf
```

2. 使用 perf kvm stat 命令之一来显示您的虚拟化主机的 perf 统计信息：

   - 若要实时监控 hypervisor ，请使用 perf kvm stat live 命令。

   - 要记录一段时间内 hypervisor 的 perf 数据，请使用 perf kvm stat record 命令激活日志记录。在命令被取消或中断后，数据保存在 perf.data.guest 文件中，可以使用 perf kvm stat report 命令进行分析。

3. 分析 VM-EXIT 事件类型及其分发的 perf 输出。例如，PAUSE_INSTRUCTION 事件应当不常发生，但在以下输出中，此事件的频繁发生表明主机 CPU 没有很好地处理正在运行的 vCPU。

在这种场景下，请考虑关闭某些处于活动状态的虚拟机，从这些虚拟机中删除 vCPU，或者 调优 vCPU 的性能。

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:


             VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time         Avg time

  EXTERNAL_INTERRUPT     365634    31.59%    18.04%      0.42us  58780.59us    204.08us ( +-   0.99% )
            MSR_WRITE     293428    25.35%     0.13%      0.59us  17873.02us      1.80us ( +-   4.63% )
     PREEMPTION_TIMER     276162    23.86%     0.23%      0.51us  21396.03us      3.38us ( +-   5.19% )
   PAUSE_INSTRUCTION     189375    16.36%    11.75%      0.72us  29655.25us    256.77us ( +-   0.70% )
                 HLT      20440     1.77%    69.83%      0.62us  79319.41us  14134.56us ( +-   0.79% )
              VMCALL      12426     1.07%     0.03%      1.02us   5416.25us      8.77us ( +-   7.36% )
      EXCEPTION_NMI         27     0.00%     0.00%      0.69us      1.34us      0.98us ( +-   3.50% )
      EPT_MISCONFIG          5     0.00%     0.00%      5.15us     10.85us      7.88us ( +-  11.67% )

Total Samples:1157497, Total events handled time:413728274.66us.
```

perf kvm stat 输出中表明有问题的其它事件类型包括：

- INSN_EMULATION - 建议次优化的 虚拟机 I/O 配置 。

有关使用 perf 监控虚拟化性能的更多信息，请参阅 perf-kvm 手册页。

numastat

要查看系统当前的 NUMA 配置，您可以使用 numastat 工具，该工具通过安装 numactl 软件包来提供。

以下显示了一个有 4 个运行虚拟机的主机，各自从多个 NUMA 节点获取内存。这不是 vCPU 性能的最佳方案，并保证调整：

```
# numastat -c qemu-kvm
```

```
Per-node process memory usage (in MBs)
PID             Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)    68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)   245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)    62    432   1661    506   4851    136     22    445  8116
53773 (qemu-kvm)  1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total             1769    463   2024   7462  10037   2672    169   7837 32434
```

相反，以下显示单个节点为每个虚拟机提供内存，这效率显著提高。

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID             Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)     0      0      7      0   8072      0      1      0  8080
53736 (qemu-kvm)     0      0      7      0      0      0   8113      0  8120
53773 (qemu-kvm)     0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)     0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                0      0   8072      0   8072      0   8114   8110 32368
```

13.9. 其他资源

- 优化 Windows 虚拟机

第 14 章 电源管理的重要性

降低计算机系统的整体功耗有助于节省成本。有效地优化每个系统组件的能源消耗包括研究系统执行的不同任务，并配置各个组件以确保其在该作业的性能正确。降低特定组件或整个系统的功耗，可以降低产生的热量并可能会降低性能。

正确的电源管理结果包括：

- 服务器和计算中心的发热的缩减
- 降低辅助成本，包括冷却、空间、电缆、发电机和不间断电源 (UPS)
- 延长笔记本电脑的电池寿命
- 降低二氧化碳的输出
- 满足与绿色 IT 相关的政府法规或法律要求，例如，节能星级
- 使新系统满足公司的要求

这部分论述了有关 Red Hat Enterprise Linux 系统的电源管理的信息。

14.1. 电源管理基础

有效电源管理基于以下原则构建：

闲置 CPU 应该在需要时唤醒

从 Red Hat Enterprise Linux 6 开始，内核以 tickless 方式运行，这意味着，以前使用的定期计时器中断已被按需中断替代。因此，空闲的 CPU 可以在新任务排队进行处理前保持闲置状态，并且已处于较低电源状态的 CPU 可以保持这个状态更长时间。但是，如果您的系统中存在会创建不必要的计时器事件的应用程序时，此功能的好处可能会减少。轮询事件（如检查卷更改或鼠标移动）是此类事件的示例。

Red Hat Enterprise Linux 包括您可以使用的工具，您可以根据自己的 CPU 使用量识别和审核应用程序。有关更多信息，请参见审计和分析概述和工具以进行审计。

应该完全禁用未使用的硬件和设备

对于存在移动部分的设备（如硬盘）也是如此。除此之外，一些应用程序可能会留下未使用但已启用的设备"打开"；当发生这种情况时，内核会假定设备处于使用状态，这样可防止设备进入节能状态。

较少的活动应转代表低的电源消耗

然而，在很多情况下，这取决于现代的硬件以及正确的 BIOS 配置，或在现代系统中使用 UEFI，包括非 x86 架构。确保您的系统使用了最新的官方固件，且在 BIOS 的电源管理或设备配置部分中启用了电源管理功能。要查找的一些功能包括：

- 对 ARM64 的 Collaborative Processor Performance Controls (CPPC) 支持
- IBM Power 系统的 PowerNV 支持
- SpeedStep
- PowerNow!
- Cool’n’Quiet
- ACPI (C-state)
- Smart

如果您的硬件对这些功能的支持，且在 BIOS 中启用了它们，Red Hat Enterprise Linux 默认使用它们。

不同的 CPU 状态形式及其影响

现代 CPU 与高级配置和电源接口 (ACPI) 结合会提供不同的电源状态。三个不同的状态是：

- Sleep (C-states)
- Frequency and voltage (P-states)
- Heat output (T-states or thermal states)

在以最低的睡眠（sleep）状态中运行的 CPU 会消耗最小的电能，但在需要时也会花费大量时间从该状态唤醒。在非常罕见的情形中，这可能会导致 CPU 在每次将要进入睡眠状态时被立即唤醒。这种情况会导致 CPU 一直处于忙碌状态，并在已使用另一个状态时丧失一些潜在的节能好处。

关闭的机器使用最少电能

省电功能的最佳方法是关闭系统。例如，您的公司可以开发一个企业文化，专注于"绿色 IT"感知，例如在午餐休息或下班后关闭机器。您还可以将多个物理服务器整合为一个较大的服务器，并使用 Red Hat Enterprise Linux 提供的虚拟化技术虚拟化它们。

14.2. 审计和分析概述

通常，单个系统的详细手动审计、分析和调优是例外，因为通常要做的时间和成本远远超过了从这些最后一部分系统调节中获得的好处。

但是，对于相似的、可以在所有系统中重复使用相同设置的大量系统，一次执行这些任务可能会非常有用。例如，部署数千台桌面系统，或部署由几乎相同的机器组成的 HPC 集群。进行审核和分析的另一种原因是，您可以创建一个基础，用于在以后比较环境变化造成的影响。对于需要定期更新硬件、BIOS 或软件的环境，此分析结果会非常有帮助，可以帮助避免出现与功耗相关的任何意外情况。通常，全面的审计和分析让您可以更好地了解特定系统中发生的情况。

对于电源消耗，审计和分析系统相对来说比较困难，即使对于大多数现代系统也是如此。大多数系统并不会提供通过软件测量电源使用情况的方法。然而，会存在一些例外：

- Hewlett Packard 服务器系统的 iLO 管理控制台具有一个电源管理模块，您可以通过 Web 访问。
- IBM 在其 BladeCenter 电源管理模块中提供类似的解决方案。

- 在一些 Dell 系统中，IT Assistant 也提供了电源监控功能。

其他供应商可能会为其服务器平台提供类似的功能，但并没有一个统一的解决方案用于所有厂商。通常，通过直接测量功耗来进行节能只会尽力而为。

14.3. 用于审计的工具

Red Hat Enterprise Linux 8 提供用于执行系统审核和分析的工具。如果您想要验证已经发现的内容或需要某些部分的更深入的信息，则它们中的大多数都可以用作补充的信息源。

其中许多工具也用于性能调优，其中包括：

powertop

它可识别频繁绕过 CPU 的内核和用户空间应用程序的特定组件。以 root 用户身份使用 powertop 命令启动 PowerTop 工具和 powertop --calibrate 来布线电源估算引擎。有关 PowerTop 的更多信息，请参阅 使用 PowerTOP 管理功耗。

Diskdevstat 和 netdevstat

它们是 SystemTap 工具，可收集有关系统中所有应用程序的磁盘活动和网络活动的详细信息。使用这些工具收集的统计数据，您可以识别与许多小 I/O 操作（而非更小且更大型的操作）的强大应用程序。以 root 用户身份使用 dnf install tuned-utils-systemtap kernel-debuginfo 命令，安装 diskdevstat 和 netdevstat 工具。

要查看磁盘和网络活动的详细信息，请使用：

```
# diskdevstat

PID  UID  DEV  WRITE_CNT  WRITE_MIN  WRITE_MAX  WRITE_AVG  READ_CNT
READ_MIN  READ_MAX  READ_AVG  COMMAND

3575  1000  dm-2  59       0.000     0.365      0.006      5       0.000      0.000     0.000
mozStorage #5
3575  1000  dm-2  7        0.000     0.000      0.000      0       0.000      0.000     0.000
localStorage DB
[...]


# netdevstat

PID  UID  DEV     XMIT_CNT  XMIT_MIN  XMIT_MAX  XMIT_AVG  RECV_CNT
RECV_MIN  RECV_MAX  RECV_AVG  COMMAND
3572  991  enp0s31f6  40     0.000     0.882     0.108     0       0.000      0.000     0.000
```

```
openvpn
3575  1000 enp0s31f6    27      0.000      1.363      0.160      0      0.000      0.000      0.000
Socket Thread
[...]
```

使用这些命令，您可以指定三个参数：update_interval、total_duration 和 display_histogram。

TuneD

它是一个基于配置集的系统调整工具，它使用 udev 设备管理器监控连接的设备，并支持系统设置的静态和动态调优。您可以使用 tuned-adm recommend 命令确定红帽推荐的配置集作为最适合特定产品的配置集。如需有关 TuneD 的更多信息，请参阅 开始使用 TuneD 和 自定义 TuneD 配置文件。使用 powertop2tuned 实用程序，您可以从 PowerTOP 推荐创建自定义 TuneD 配置集。有关 powertop2tuned 工具的详情，请参考 优化功耗。

虚拟内存统计信息 (vmstat)

它由 procps-ng 软件包提供。使用这个工具，您可以查看进程、内存、分页、块 I/O、陷阱和 CPU 活动的详细信息。

要查看此信息，请使用：

```
$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r  b  swpd  free    buff   cache   si   so  bi   bo   in  cs  us  sy id  wa  st
1  0   0   5805576 380856 4852848   0    0  119  73  814  640  2   2 96   0   0
```

使用 vmstat -a 命令，可以显示活动和不活跃的内存。有关其他 vmstat 选项的详情，请查看 vmstat 手册页。

iostat

它由 sysstat 软件包提供。此工具与 vmstat 类似，但只适用于监控块设备上的 I/O。它还提供了更详细的输出和统计数据。

要监控系统 I/O，请使用：

```
$ iostat
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          2.05    0.46    1.55    0.26    0.00   95.67

Device     tps     kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
nvme0n1    53.54     899.48     616.99      3445229     2363196
```

```
dm-0      42.84    753.72    238.71     2886921     914296
dm-1       0.03      0.60      0.00        2292          0
dm-2      24.15    143.12    379.80      548193    1454712
```

blktrace

它提供有关在 I/O 子系统中花费时间的详细信息。

要以人类可读格式查看此信息，请使用：

```
# blktrace -d /dev/dm-0 -o - | blkparse -i -

253,0   1    1   0.000000000  17694  Q   W 76423384 + 8 [kworker/u16:1]
253,0   2    1   0.001926913     0   C   W 76423384 + 8 [0]
[...]
```

第一列 253,0 是设备主和次的元组。第二列 1 提供了有关 CPU 的信息，后跟用于发出 IO 进程的进程的时间戳和 PID 的列。

第六个列 Q 显示事件类型，第 7 列，W 代表写入操作，第 8 列为 76423384，是块号，+ 8 是请求的块的数量。

最后一个字段 [kworker/u16:1] 是进程名称。

默认情况下，blktrace 命令会永久运行，直到进程被明确终止。使用 -w 选项指定运行时持续时间。

turbostat

它由 kernel-tools 软件包提供。它报告了 x86-64 处理器上的处理器拓扑、频率、空闲的电源状态统计、温度和功耗。

要查看此摘要，请使用：

```
# turbostat

CPUID(0): GenuineIntel 0x16 CPUID levels; 0x80000008 xlevels; family:model:stepping
0x6:8e:a (6:142:10)
CPUID(1): SSE3 MONITOR SMX EIST TM2 TSC MSR ACPI-TM HT TM
CPUID(6): APERF, TURBO, DTS, PTM, HWP, HWPnotify, HWPwindow, HWPepp, No-
HWPpkg, EPB
[...]
```

默认情况下，turbostat 显示整个屏幕的计数器结果摘要，每 5 秒显示计数器结果。使用 -i 选项指定计数器结果之间的不同周期，例如，执行 turbostat -i 10 会每 10 秒打印结果。

Turbostat 在识别电源使用或空闲时间方面效率低下的服务器也很有用。它还有助于识别系统中发生的系统管理中断 (SMI) 的速度。它还可用于验证电源管理调整的影响。

cpupower

IT 是用于检查和调优处理器相关功能的工具集合。在 cpupower 命令中使用 frequency-info, frequency-set, idle-info, idle-set, set, info, 和 monitor 选项来显示和设置处理器相关的值。

例如，要查看可用的 cpufreq 管理器，请使用：

```
$ cpupower frequency-info --governors
analyzing CPU 0:
 available cpufreq governors: performance powersave
```

有关 cpupower 的更多信息，请参阅查看 CPU 相关信息。

GNOME Power Manager

它是作为 GNOME 桌面环境的一部分安装的守护进程。GNOME Power Manager 通知您系统的电源状态变化，例如，从电池改为 AC 电源。它还会报告电池状态，并在电池电源较低时提醒您。

其他资源

- powertop(1), diskdevstat(8), netdevstat(8), tuned(8), vmstat(8), iostat(1), blktrace(8), blkparse(8), 和 turbostat(8) man page
- cpupower(1), cpupower-set(1), cpupower-info(1), cpupower-idle(1), cpupower-frequency-set(1), cpupower-frequency-info(1), and cpupower-monitor(1) man page

第 15 章 使用 POWERTOP 管理能耗

作为系统管理员，您可以使用 PowerTOP 工具来分析和管理功耗。

15.1. POWERTOP 的目的

PowerTOP 是一个诊断与功耗相关的问题的程序，并提供如何延长生动生命周期的建议。

PowerTOP 工具可提供系统总功耗和各个进程、设备、内核工作器、计时器和中断处理器的功耗。工具还可识别频繁绕过 CPU 的内核和用户空间应用程序的特定组件。

Red Hat Enterprise Linux 9 使用 PowerTOP 版本 2.x。

15.2. 使用 POWERTOP

先决条件

- 为了可以使用 PowerTOP，请确定在您的系统中已安装了 powertop 软件包：

```
# dnf install powertop
```

15.2.1. 启动 PowerTOP

步骤

- 要运行 PowerTOP，请使用以下命令：

```
# powertop
```

重要

笔记本电脑应在运行 powertop 命令时以电能量运行。

15.2.2. 校准 PowerTOP

步骤

1. 在笔记本电脑中，您可以通过运行以下命令来布放节能引擎：

```
# powertop --calibrate
```

2. 让校准完成，而不会在此过程中与机器交互。

校准需要一些时间，因为进程执行各种测试，整个测试通过强度级别和交换机设备进行循环。

3. 在完成校准过程后，PowerTOP 会正常启动。让它运行大约一小时以收集数据。

收集足够数据后，输出表的第一列中将显示节能数据。

注意

请注意，powertop --calibrate 仅适用于笔记本电脑。

15.2.3. 设置测量间隔

默认情况下，PowerTOP 将测量间隔为 20 秒。

如果要更改此测量频率，请使用以下步骤：

步骤

- 使用 --time 选项运行 powertop 命令：

```
# powertop --time=time in seconds
```

15.2.4. 其他资源

有关如何使用 PowerTOP 的详情，请查看 powertop man page。

15.3. POWERTOP 统计

在运行期间，PowerTOP 会从系统收集统计信息。

powertop的输出提供多个标签：

- 概述
- idle stats
- 频率统计
- 设备统计
- Tunables
- WakeUp

您可以使用 Tab 和 Shift+Tab 键通过这些选项卡进行循环。

15.3.1. Overview 选项卡

在 Overview 选项卡中，您可以查看将 wakeups 发送到 CPU 最频繁或消耗最多功能的组件列表。Overview 选项卡中的项目（包括进程、中断、设备和其他资源）会根据它们的使用情况进行排序。

Overview 选项卡中的 adjacent 列提供以下信息：

使用

详细估算资源的使用情况。

Events/s

每秒的 Wakeups 数。每秒唤醒的时间数代表如何高效地执行内核的设备和驱动程序。较少的 wakeups 表示消耗较少电源。组件按照可进一步优化的电源使用量排序。

类别

组件的类别，如进程、设备或计时器。

描述

组件的描述。

如果正确校准，则会显示第一列中每个列出的项目的功耗估算。

除此之外，Overview 选项卡还包含包含摘要统计的行，例如：

- 电源消耗总数
- 剩余限制生命周期（仅在适用的情况下）
- 每秒的 GPU 操作总数（每秒为 GPU 操作）和每秒虚拟文件系统操作

15.3.2. Idle stats 标签页

Idle stats 选项卡显示所有处理器和内核的使用 C-states，而 Frequency stats 选项卡显示 P-states 的使用，包括 Turbo 模式（若适用）所有处理器和内核。C- 或 P-states 的持续时间代表 CPU 用量的优化程度。CPU 使用率更长的 CPU 处于更高的 C- 或 P-states（例如，C4 大于 C3），CPU 使用量优化越好。理想情况下，当系统闲置时，驻留的最高 C- 或 P-state 应该为 90% 或更高。

15.3.3. Device stats 标签页

Device stats 选项卡中提供与 Overview 选项卡类似的信息，但只适用于设备。

15.3.4. Tunables 选项卡

Tunables 选项卡包含 PowerTOP的建议，以优化系统以降低功耗。

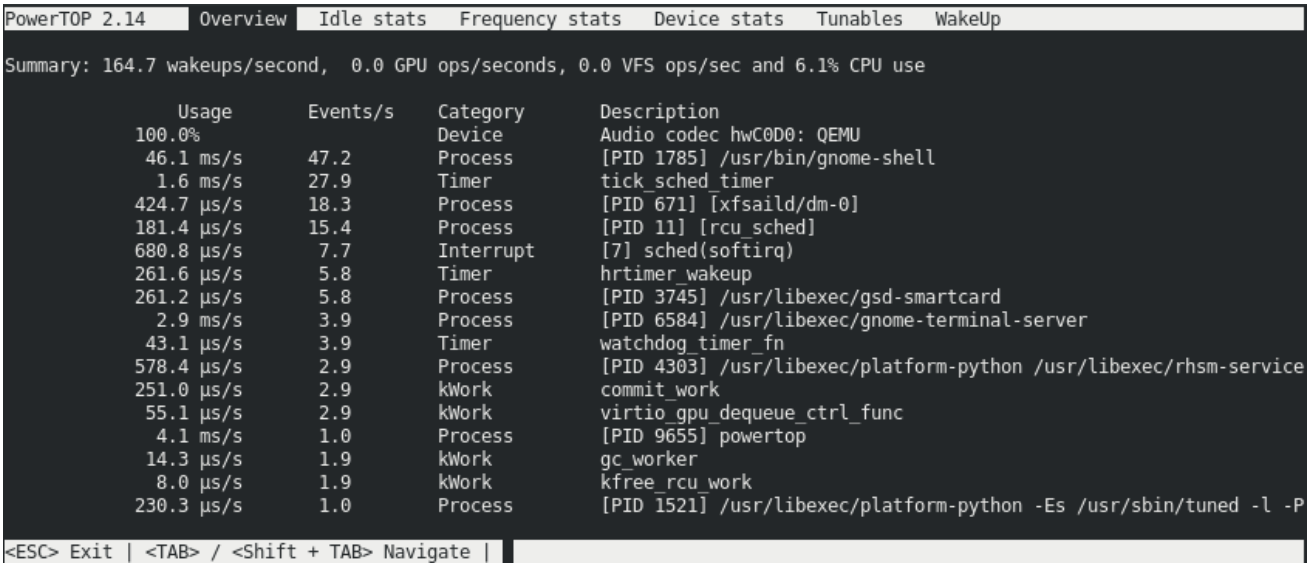
使用 up 和 down 键移动建议，使用 enter 键打开或关闭建议。

15.3.5. WakeUp 选项卡

WakeUp 选项卡显示设备 wakeup 设置，供用户根据需要更改。

使用 up 和 down 键通过可用的设置移动，并使用 enter 键启用或禁用设置。

图 15.1. PowerTOP 输出



其他资源

有关 PowerTOP 的详情，请参阅 PowerTOP 主页。

15.4. 为什么 POWERTOP 不会在一些实例中显示 FREQUENCY STATS 值

在使用 Intel P-State 驱动程序时，如果驱动程序处于被动模式，PowerTOP 仅显示 Frequency Stats 选项卡中的值。但在这种情况下，这些值可能不完整。

总而言，Intel P-State 驱动程序有三种可能模式：

- 使用硬件 P-States (HWP) 的活跃模式
- 没有 HWP 的活跃模式

- 被动模式

切换到 ACPI CPUfreq 驱动程序会导致 PowerTOP 显示的完整信息。但是，建议将您的系统保留在默认设置中。

要查看载入哪些驱动程序以及在什么模式下运行：

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
```

- 如果 Intel P-State 驱动程序被加载且处于活跃模式，则返回 intel_pstate。
- 如果 Intel P-State 驱动程序被加载且处于被动模式，则返回 intel_cpufreq。
- 如果载入 ACPI CPUfreq 驱动程序，则返回 acpi-cpufreq。

在使用 Intel P-State 驱动程序时，在内核命令行中添加以下参数，以强制驱动程序在被动模式下运行：

```
intel_pstate=passive
```

要禁用 Intel P-State 驱动程序并使用，而是使用 ACPI CPUfreq 驱动程序，在内核引导命令行中添加以下参数：

```
intel_pstate=disable
```

15.5. 生成 HTML 输出

除了终端中的 powertop 输出外，您还可以生成 HTML 报告。

步骤

- 使用 --html 选项运行 powertop 命令：

```
# powertop --html=htmlfile.html
```

将 htmlfile.html 参数替换为输出文件所需名称。

15.6. 优化功耗

要优化功耗，您可以使用 powertop 服务或 powertop2tuned 程序。

15.6.1. 使用 powertop 服务优化功耗

您可以使用 powertop 服务，从引导时的 Tunables 选项卡中自动启用所有 PowerTOP 的建议：

步骤

- 启用 powertop 服务：

```
# systemctl enable powertop
```

15.6.2. powertop2tuned 工具

powertop2tuned 程序允许您从 PowerTOP 建议创建自定义 TuneD 配置集。

默认情况下，powertop2tuned 在 /etc/tuned/ 目录中创建配置集，并在当前选择的 TuneD 配置集中基础配置集。为安全起见，所有 PowerTOP 调优最初在新配置集中被禁用。

要启用调整，您可以：

- 在 /etc/tuned/profile_name/tuned.conf 文件中 取消注释。
- 使用 --enable 或 -e 选项生成新的配置集，启用 PowerTOP 建议的大部分调优。

  某些潜在的调整（如 USB 自动暂停）在默认情况下被禁用，需要手动取消注释。

15.6.3. 使用 powertop2tuned 程序优化电源消耗

先决条件

- powertop2tuned 程序已安装在系统上：

```
# dnf install tuned-utils
```

步骤

1. 创建自定义配置集：

```
# powertop2tuned new_profile_name
```

2. 激活新配置集：

```
# tuned-adm profile new_profile_name
```

附加信息

- 要获得 powertop2tuned 支持的选项列表，请使用：

```
$ powertop2tuned --help
```

15.6.4. powertop.service 和 powertop2tuned 的比较

和 powertop.service 相比，对于优化能耗应首选 powertop2tuned，理由如下：

- powertop2tuned 实用程序代表将 PowerTOP 集成到 TuneD 中，这能够带来这两个工具的优势。
- powertop2tuned 实用程序允许对已启用的调优进行精细的控制。
- 使用 powertop2tuned 时，可能无法自动启用潜在的危险调整。
- 通过 powertop2tuned，可以在不重启的情况下进行回滚。

第 16 章 PERF 入门

作为系统管理员，您可以使用 perf 工具来收集和分析系统的性能数据。

16.1. PERF 简介

与基于内核的子系统 *Performance Counters for Linux* (PCL) 的 perf 用户空间工具接口。perf 是一个强大的工具，它使用性能监控单元 (PMU) 测量、记录和监控各种硬件和软件事件。perf 还支持追踪点、kprobes 和 uprobes。

16.2. 安装 PERF

此流程安装 perf 用户空间工具。

步骤

- 安装 perf 工具：

```
# dnf install perf
```

16.3. 常见 PERF 命令

perf stat

此命令提供常见性能事件的总体统计信息，包括执行的指令和消耗的时钟周期。选项可用于选择默认测量事件以外的事件。

perf 记录

此命令将性能数据记录到文件 perf.data 中，稍后可以使用 perf report 命令进行分析。

perf 报告

此命令从 perf 记录 创建的 perf.data 文件中读取和显示性能数据。

perf list

此命令列出特定计算机上可用的事件。这些事件将根据系统的性能监控硬件和软件配置而有所不同。

perf top

此命令执行与 top 实用程序类似的功能。它生成并实时显示性能计数器配置集。

perf trace

此命令执行与 strace 工具类似的函数。它监控指定线程或进程使用的系统调用，以及该应用收到的所有信号。

perf help

此命令显示 perf 命令的完整列表。

其他资源

- 在 子命令中添加 --help 选项以打开 man page。

第 17 章 使用 PERF TOP 实时分析 CPU 使用量

您可以使用 perf top 命令来实时测量不同功能的 CPU 使用量。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

17.1. PERF TOP 的目的

perf top 命令用于实时系统分析，功能类似于 top 实用程序。但是，当 top 实用工具通常会显示给定进程或线程使用的 CPU 时间，perf top 则显示每个特定功能使用的 CPU 时间。在其默认状态下，perf top 告知您在用户空间和内核空间中的所有 CPU 之间使用的功能。要使用 perf， 您需要 root 访问权限。

17.2. 使用 PERF TOP 分析 CPU 使用量

此流程实时激活 perf top 和实时 profile CPU 用量。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 有 root 访问权限

流程

- 启动 perf top 监控接口：

```
# perf top
```

监控接口类似如下：

```
Samples: 8K of event 'cycles', 2000 Hz, Event count (approx.): 4579432780 lost: 0/0 drop:
0/0
Overhead  Shared Object       Symbol
   2.20%  [kernel]            [k] do_syscall_64
   2.17%  [kernel]            [k] module_get_kallsym
```

```
  1.49%  [kernel]           [k] copy_user_enhanced_fast_string
  1.37%  libpthread-2.29.so  [.] pthread_mutex_lock 1.31% [unknown] [.] 0000000000000000
1.07% [kernel] [k] psi_task_change 1.04% [kernel] [k] switch_mm_irqs_off 0.94% [kernel] [k]
fget
  0.74%  [kernel]           [k] entry_SYSCALL_64
  0.69%  [kernel]           [k] syscall_return_via_sysret
  0.69%  libxul.so          [.] 0x000000000113f9b0
  0.67%  [kernel]           [k] kallsyms_expand_symbol.constprop.0
  0.65%  firefox            [.] moz_xmalloc
  0.65%  libpthread-2.29.so  [.] __pthread_mutex_unlock_usercnt
  0.60%  firefox            [.] free
  0.60%  libxul.so          [.] 0x000000000241d1cd
  0.60%  [kernel]           [k] do_sys_poll
  0.58%  [kernel]           [k] menu_select
  0.56%  [kernel]           [k] _raw_spin_lock_irqsave
  0.55%  perf               [.] 0x00000000002ae0f3
```

在这个示例中，内核功能 do_syscall_64 使用最多的 CPU 时间。

其他资源

- perf-top (1) man page

17.3. PERF OUTPUT 输出的解释

perf output 监控接口在多个列中显示数据：

"Overhead" 列

显示特点功能使用的 CPU 百分比。

"Shared Object" 列

显示使用函数的程序或库的名称。

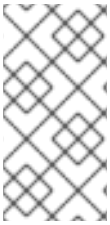
"Symbol" 列

显示功能名称或符号。在内核空间中执行的功能由 [k] 标识，在用户空间中执行的功能由 [.] 标识。

17.4. 为什么 PERF 会显示一些功能名称作为原始功能地址

对于内核功能，perf 使用 /proc/kallsyms 文件中的信息将示例映射到其相应的功能名称或符号。对于用户空间中执行的功能，您可能会看到原始功能地址，因为二进制文件被剥离。

必须安装可执行文件的 debuginfo 软件包；如果可执行文件是本地开发的应用程序，则必须通过打开的调试信息（GCC 中的 -g 选项）编译，以显示这样的情形中的功能名称或符号。



注意

安装与可执行文件关联的 debuginfo 后，不需要重新运行 perf record 命令。只需重新运行 perf report 命令。

其它资源

- 使用调试信息启用调试

17.5. 启用调试和源存储库

Red Hat Enterprise Linux 的标准安装并不会启用 debug 和 source 软件仓库。这些仓库包含调试系统组件并测量其性能所需的信息。

流程

- 启用源和调试信息软件包频道： $ (uname -i) 部分会自动替换为您系统构架的匹配值：

架构名称	Value
64 位 Intel 和 AMD	x86_64
64-bit ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

17.6. 使用 GDB 获取应用程序或库的 DEBUGINFO 软件包

调试代码需要调试信息。对于从软件包安装的代码，GNU Debugger (GDB) 会自动识别缺少的调试信息，解析软件包名称，并提供了有关如何获取软件包的建议。

先决条件

- 必须在系统上安装您要调试的应用程序或库。

- 在系统中必须安装 GDB 和 debuginfo-install 工具。详情请参阅设置调试应用程序。

- 提供 debuginfo 和 debugsource 软件包的存储库必须在系统上配置和启用。详情请参阅 启用调试和源存储库。

步骤

1. 启动附加到您要调试的应用程序或库的 GDB。GDB 会自动识别缺少的调试信息并建议一个命令运行。

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. 退出 GDB：键入 q，使用 Enter 进行确认。

```
(gdb) q
```

3. 运行 GDB 建议的命令来安装所需的 debuginfo 软件包：

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf 软件包管理工具提供了更改摘要，要求确认，一旦被您确认后会下载和安装所有需要的文件。

4. 如果 GDB 无法建议 debuginfo 软件包，请按照 手动应用程序或库获取 debuginfo 软件包中所述的步骤操作。

其他资源

- 如何为 RHEL 系统下载或安装 debuginfo 软件包？ - 红帽知识库解决方案

第 18 章 使用 PERF STAT 在进程执行过程中计算事件

您可以使用 perf stat 命令在进程执行期间计算硬件和软件事件。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

18.1. PERF STAT 的目的

perf stat 命令执行指定的命令，在命令执行过程中保留运行次数和软件事件，并生成这些计数的统计信息。如果您没有指定任何事件，则 perf stat 会计算一组通用的硬件和软件事件。

18.2. 使用 PERF STAT 计数事件

您可以使用 perf stat 计算命令执行过程中出现的硬件和软件事件，并生成这些计数的统计信息。默认情况下，perf stat 以针对每个线程的模式运行。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 计算事件数。
  - 在没有 root 访问权限的情况下运行 perf stat 命令只会计算在用户空间中出现的事件：

```
$ perf stat ls
```

例 18.1. perf stat 的输出在没有 root 访问权限的情况下运行

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

 Performance counter stats for 'ls':

              1.28 msec task-clock:u              #    0.165 CPUs utilized
                 0      context-switches:u        #    0.000 M/sec
                 0      cpu-migrations:u          #    0.000 K/sec
```

```
       104      page-faults:u             #    0.081 M/sec
 1,054,302      cycles:u                  #    0.823 GHz
 1,136,989      instructions:u            #    1.08  insn per cycle
  228,531       branches:u                #  178.447 M/sec
   11,331       branch-misses:u           #    4.96% of all branches

0.007754312 seconds time elapsed

0.000000000 seconds user
0.007717000 seconds sys
```

如上例中所示，如果 perf stat 运行没有 root 访问权限，则事件名称会加上 :u，表示这些事件只在用户空间中计算出来。

- **要计算用户空间和内核空间事件，在运行 perf stat 时必须具有 root 访问权限：**

```
# perf stat ls
```

例 18.2. 使用 root 访问权限运行 perf stat 的输出

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

 Performance counter stats for 'ls':

          3.09 msec task-clock              #    0.119 CPUs utilized
            18      context-switches        #    0.006 M/sec
             3      cpu-migrations          #    0.969 K/sec
           108      page-faults             #    0.035 M/sec
     6,576,004      cycles                  #    2.125 GHz
     5,694,223      instructions            #    0.87  insn per cycle
     1,092,372      branches                #  352.960 M/sec
        31,515      branch-misses           #    2.89% of all branches

0.026020043 seconds time elapsed

0.000000000 seconds user
0.014061000 seconds sys
```

- **默认情况下，perf stat 以针对每个线程的模式运行。要更改为 CPU 范围事件计数，请将 -a 选项传递给 perf stat。要计算 CPU 范围内的事件，您需要 root 访问权限：**

```
# perf stat -a ls
```

其他资源

- perf-stat(1) man page

18.3. PERF STAT 输出的解释

perf stat 在命令执行期间执行指定命令并计数发生事件，并在三列中显示这些计数的统计信息：

1. 给定事件的发生次数
2. 被计算的事件的名称
3. 当相关的指标可用时，右列中的 hash 符号 (#) 会显示比例或百分比。

例如，当以默认模式运行时，perf stat 会计算周期和说明，因此计算并在最接近列中显示每个周期的说明。对于缺失的分支（占所有分支的百分比），您可以看到类似行为，因为默认情况下这两个事件都会被计算。

18.4. 将 PERF STAT 附加到正在运行的进程

您可以将 perf stat 附加到正在运行的进程。这将指示 perf stat 在执行命令期间仅在指定进程中发生事件。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 将 perf stat 附加到正在运行的进程中：

```
$ perf stat -p ID1,ID2 sleep seconds
```

前面的例子计算进程 *ID1* 和 *ID2* 的事件，这些事件的时间单位是 sleep 命令使用的*秒*数。

其他资源

- **perf-stat(1) man page**

第 19 章 使用 PERF 记录和分析性能配置集

perf 工具允许您记录性能数据并在以后对其进行分析。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

19.1. PERF RECORD 的目的

perf record 命令对性能数据进行样本，并将其存储在文件 perf.data 中，可以使用其他 perf 命令来读取和视觉化。perf.data 在当前目录中生成，并可以在以后访问，可能在不同计算机上。

如果您没有为 perf record 指定命令，它将持续进行记录，直到您通过按 Ctrl+C 手动停止进程为止。您可以通过传递 -p 选项后跟一个或多个进程 ID，将 perf record 附加到特定进程。但是，您可以在没有 root 访问权限的情况下运行 perf record，因此只有用户空间中的性能数据示例。在默认模式中，perf record 使用 CPU 周期作为抽样事件，并在每个线程模式下运行，并启用了继承模式。

19.2. 在没有 ROOT 访问权限的情况下记录性能配置集

您可以在没有 root 访问权限的情况下使用 perf record 来记录用户空间中的性能数据。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 记录性能数据示例：

```
$ perf record command
```

使用您要其期间对其进行抽样的命令替换 *command*。如果没有指定命令，则 perf record 将持续对数据进行抽样，直到手动按 Ctrl+C 来为止。

其他资源

- perf-record (1) man page

19.3. 使用 ROOT 访问权限记录性能配置集

您可以使用带有 root 访问权限的 perf record 来同时在用户空间和内核空间中记录性能数据。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 有 root 访问权限。

流程

- 记录性能数据示例：

```
# perf record command
```

使用您要其期间对其进行抽样的命令替换 *command*。如果没有指定命令，则 perf record 将持续对数据进行抽样，直到手动按 Ctrl+C 来为止。

其他资源

- perf-record (1) man page

19.4. 以针对每个 CPU 的模式记录性能档案

您可以使用针对每个 CPU 的模式运行 perf record 来抽样和记录在一个被监测的 CPU 中的所有线程的用户空间和内核空间的性能数据。默认情况下，CPU 模式会监控所有在线 CPU。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 记录性能数据示例：

```
# perf record -a command
```

使用您要其期间对其进行抽样的命令替换 ***command***。如果没有指定命令，则 **perf record** 将持续对数据进行抽样，直到手动按 **Ctrl+C** 来为止。

其他资源

- **perf-record (1) man page**

19.5. 使用 PERF RECORD 捕获调用图形数据

您可以配置 **perf record** 工具，以便其记录在性能配置集中调用其他功能的功能。如果多个进程调用相同功能，这有助于识别瓶颈。

先决条件

- 已安装 **perf** 用户空间工具，如安装 perf 所述。

流程

- 使用 **--call-graph** 选项记录性能数据示例：

```
$ perf record --call-graph method command
```

  - 使用您要其期间对其进行抽样的命令替换 ***command***。如果没有指定命令，则 **perf record** 将持续对数据进行抽样，直到手动按 **Ctrl+C** 来为止。

  - 使用以下 **unwinding** 的方法之一替换 ***method***:

    fp

    使用帧指针方法。根据编译器优化，如使用 GCC 选项 **--fomit-frame-pointer** 构建的二进制文件，这可能无法取消验证堆栈。

    dwarf

使用 DWARF Call Frame 信息来 unwind 堆栈。

lbr

使用 Intel 处理器上的最后分支记录硬件。

其他资源

- perf-record (1) man page

19.6. 使用 PERF REPORT 分析 PERF.DATA

您可以使用 perf report 来显示和分析 perf.data 文件。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 当前目录中有一个 perf.data 文件。
- 如果 perf.data 文件是使用 root 访问权限创建的，则需要使用 root 访问权限运行 perf report。

流程

- 显示 perf.data 文件的内容，以进一步分析：

```
# perf report
```

这个命令显示类似如下的输出：

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead  Command          Shared Object                     Symbol
  2.36%  kswapd0          [kernel.kallsyms]                 [k] page_vma_mapped_walk
  2.13%  sssd_kcm         libc-2.28.so                      [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
  1.17%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_hash_table_lookup
```

```
  0.93%  Xorg            libc-2.28.so                 [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
  0.83%  Xorg            [kernel.kallsyms]            [k] alloc_vmap_area
  0.63%  gnome-shell     libglib-2.0.so.0.5600.4      [.] g_slice_alloc
  0.53%  gnome-shell     libgirepository-1.0.so.1.0.0 [.] g_base_info_unref
  0.53%  gnome-shell     ld-2.28.so                   [.] _dl_find_dso_for_object
  0.49%  kswapd0         [kernel.kallsyms]            [k] vma_interval_tree_iter_next
  0.48%  gnome-shell     libpthread-2.28.so           [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock
```

其他资源

- **perf-report (1) man page**

19.7. PERF 报告输出的解释

运行 **perf report** 命令显示的表会将数据排序为几列：

'Overhead' 列

指明在该特定功能中收集的整体样本的百分比。

'Command' 列

告诉您从哪个进程收集样本。

'Shared Object' 列

显示示例来自内核的 ELF 镜像的名称（当样本来自内核时使用名称 **[kernel.kallsyms]**）。

'Symbol' 列

显示功能名称或符号。

在默认模式中，功能按照降序排列，首先显示最高的开销。

19.8. 生成可在不同设备上读取的 PERF.DATA 文件

您可以使用 perf 工具将性能数据记录到 perf.data 文件中，以便在不同的设备上分析。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 已安装内核 debuginfo 软件包。如需更多信息，请参阅使用 GDB 来获取应用程序或库的 debuginfo 软件包。

流程

1. 捕获您需要进一步调查的性能数据：

```
# perf record -a --call-graph fp sleep seconds
```

这个示例会在整个系统中生成一个 perf.data，其包括的时间是使用 sleep 命令指定的 *seconds* 秒数。它还将使用数据框架指针方法捕获调用图形数据。

2. 生成包含记录数据的 debug 符号的归档文件：

```
# perf archive
```

验证步骤

- 验证存档文件是否已在您的当前活跃目录中生成：

```
# ls perf.data*
```

输出中将显示以 perf.data 开头的每个文件。归档文件将命名为：

```
perf.data.tar.gz
```

或者

```
perf.data.tar.bz2
```

其他资源

- 使用 perf 记录和分析性能配置集
- 使用 perf record 捕获调用图形数据

19.9. 分析在不同设备中创建的 PERF.DATA 文件

您可以使用 perf 工具分析在不同设备上生成的 perf.data 文件。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 使用的 perf.data 文件和相关存档文件存在于不同的设备上。

流程

1. 将 perf.data 文件和存档文件复制到您当前的活跃目录中。
2. 将存档文件提取到 ~/.debug 中：

```
# mkdir -p ~/.debug
# tar xf perf.data.tar.bz2 -C ~/.debug
```

注意

归档文件可能也命名为 ***perf.data.tar.gz***。

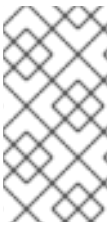
3. 打开 perf.data 文件以进一步分析：

```
# perf report
```

19.10. 为什么 PERF 会显示一些功能名称作为原始功能地址

对于内核功能，perf 使用 /proc/kallsyms 文件中的信息将示例映射到其相应的功能名称或符号。对于用户空间中执行的功能，您可能会看到原始功能地址，因为二进制文件被剥离。

必须安装可执行文件的 debuginfo 软件包；如果可执行文件是本地开发的应用程序，则必须通过打开的调试信息（GCC 中的 -g 选项）编译，以显示这样的情形中的功能名称或符号。



注意

安装与可执行文件关联的 debuginfo 后，不需要重新运行 perf record 命令。只需重新运行 perf report 命令。

其它资源

- 使用调试信息启用调试

19.11. 启用调试和源存储库

Red Hat Enterprise Linux 的标准安装并不会启用 debug 和 source 软件仓库。这些仓库包含调试系统组件并测量其性能所需的信息。

流程

- 启用源和调试信息软件包频道： $ (uname -i) 部分会自动替换为您系统构架的匹配值：

架构名称	Value
64 位 Intel 和 AMD	x86_64
64-bit ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

19.12. 使用 GDB 获取应用程序或库的 DEBUGINFO 软件包

调试代码需要调试信息。对于从软件包安装的代码，GNU Debugger (GDB) 会自动识别缺少的调试信息，解析软件包名称，并提供了有关如何获取软件包的建议。

先决条件

- 必须在系统上安装您要调试的应用程序或库。
- 在系统中必须安装 GDB 和 debuginfo-install 工具。详情请参阅设置调试应用程序。
- 提供 debuginfo 和 debugsource 软件包的存储库必须在系统上配置和启用。详情请参阅 启用调试和源存储库。

步骤

1. 启动附加到您要调试的应用程序或库的 GDB。GDB 会自动识别缺少的调试信息并建议一个命令运行。

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. 退出 GDB：键入 q，使用 Enter 进行确认。

```
(gdb) q
```

3. 运行 GDB 建议的命令来安装所需的 debuginfo 软件包：

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf 软件包管理工具提供了更改摘要，要求确认，一旦被您确认后会下载和安装所有需要的文件。

4. 如果 GDB 无法建议 debuginfo 软件包，请按照 手动应用程序或库获取 debuginfo 软件包中所述的步骤操作。

其他资源

- 如何为 RHEL 系统下载或安装 debuginfo 软件包？ - 红帽知识库解决方案

第 20 章 使用 PERF 调查繁忙的 CPU

在调查系统上的性能问题时，您可以使用 perf 工具来识别和监控总线 CPU，以便专注于您的努力。

20.1. 显示使用 PERF STAT 时会计算哪些 CPU 事件

您可以使用 perf stat 来显示因为禁用 CPU 计数聚而被计算的 CPU 事件。您必须使用 -a 标志来计算系统范围的事件，才能使用此功能。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 计算禁用 CPU 数聚合的事件：

```
# perf stat -a -A sleep seconds
```

前面的例子显示了一组默认的硬件和软件事件，覆盖的时间是 sleep 命令中指定的 *seconds* 的秒数，针对每个 CPU（从 CPU0 开始以升序进行排序）。因此，它可以用于指定一个事件（如周期）：

```
# perf stat -a -A -e cycles sleep seconds
```

20.2. 显示使用 PERF REPORT 要使用哪些 CPU 样本

perf record 命令对性能数据进行样本，并将其存储在文件 perf.data 中，您可以使用 perf report 命令来读取这个文件。perf record 命令会记录要使用哪些 CPU 样本。您可以配置 perf report 来显示此信息。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 在当前的目录中有一个使用 perf record 创建的 perf.data 文件。如果 perf.data 文件是使用 root 访问权限创建的，则需要使用 root 访问权限运行 perf report。

流程

- 显示 perf.data 文件的内容用于进一步分析，按 CPU 排序：

```
# perf report --sort cpu
```

  - 您可以按 CPU 和命令排序，以显示消耗的 CPU 时间的更多详细信息：

```
# perf report --sort cpu,comm
```

    本示例将按开销使用量降序列出所有受监控 CPU 的命令，并标识从中执行命令的 CPU。

其他资源

- 使用 perf 记录和分析性能配置集

20.3. 使用 PERF TOP 在性能分析期间显示特定 CPU

您可以在实时分析系统时，配置 perf top 来显示特定的 CPU 及其相对使用量。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 启动 perf top 接口，按 CPU 排序：

```
# perf top --sort cpu
```

  这个示例将实时列出 CPU 及其对应开销，以降序排列。

  - 您可以按 CPU 和命令排序，以了解 CPU 时间被消耗在什么地方：

```
# perf top --sort cpu,comm
```

本示例将按消耗使用量降序列出命令，并确定命令实时在哪个 CPU 上执行。

20.4. 使用 PERF RECORD 和 PERF REPORT 监控特定 CPU

您可以将 perf record 配置为仅针对目标的特定 CPU 样本，并使用 perf report 分析生成的 perf.data 文件以进一步分析。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

1. 对特定 CPU 进行抽样并记录性能数据，生成 perf.data 文件：

   - 使用以逗号分隔的 CPU 列表：

     ```
     # perf record -C 0,1 sleep seconds
     ```

     以上示例抽样并记录 CPU 0 和 1 中的数据，覆盖的时间为 sleep 命令指定的 *seconds* 秒数。

   - 使用一系列 CPU:

     ```
     # perf record -C 0-2 sleep seconds
     ```

     以上示例对 CPU 0 到 2 的所有 CPU 进行抽样并记录数据，覆盖时间为 sleep 命令中使用的 *seconds* 指定的秒数。

2. 显示 perf.data 文件的内容，以进一步分析：

   ```
   # perf report
   ```

   本例将显示 perf.data 的内容。如果您在监控多个 CPU 并想了解哪些 CPU 数据被抽样，请参阅使用 perf report 显示哪些 CPU 样本。

第 21 章 使用 PERF 监控应用程序性能

您可以使用 perf 工具来监控和分析应用程序性能。

21.1. 将 PERF 记录附加到正在运行的进程

您可以将 perf record 附加到正在运行的进程。这将指示 perf record 只对于特定的进程进行抽样并记录性能数据。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

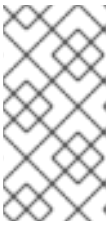
流程

- 将 perf record 附加到正在运行的进程中：

```
$ perf record -p ID1,ID2 sleep seconds
```

前面的示例抽样并记录 ID 为 *ID1* 和 *ID2* 的进程，覆盖的时间范围为 sleep 命令指定的 *seconds* 秒数。您还可以配置 perf 来记录特定线程中的事件：

```
$ perf record -t ID1,ID2 sleep seconds
```



注意

当使用 -t 标志和处理线程 ID 时，perf 会默认禁用继承功能。您可以通过添加 --inherit 选项来启用继承性。

21.2. 使用 PERF RECORD 捕获调用图形数据

您可以配置 perf record 工具，以便其记录在性能配置集中调用其他功能的功能。如果多个进程调用相同功能，这有助于识别瓶颈。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

- 使用 --call-graph 选项记录性能数据示例：

```
$ perf record --call-graph method command
```

  - 使用您要其期间对其进行抽样的命令替换 *command*。如果没有指定命令，则 perf record 将持续对数据进行抽样，直到手动按 Ctrl+C 来为止。

  - 使用以下 unwinding 的方法之一替换 *method*:

    fp

    使用帧指针方法。根据编译器优化，如使用 GCC 选项 --fomit-frame-pointer 构建的二进制文件，这可能无法取消验证堆栈。

    dwarf

    使用 DWARF Call Frame 信息来 unwind 堆栈。

    lbr

    使用 Intel 处理器上的最后分支记录硬件。

其他资源

- perf-record (1) man page

21.3. 使用 PERF REPORT 分析 PERF.DATA

您可以使用 perf report 来显示和分析 perf.data 文件。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

- 当前目录中有一个 perf.data 文件。

- 如果 perf.data 文件是使用 root 访问权限创建的，则需要使用 root 访问权限运行 perf report。

流程

- 显示 perf.data 文件的内容，以进一步分析：

```
# perf report
```

这个命令显示类似如下的输出：

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead  Command          Shared Object                     Symbol
   2.36%  kswapd0          [kernel.kallsyms]                 [k] page_vma_mapped_walk
   2.13%  sssd_kcm         libc-2.28.so                      [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
   1.17%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_hash_table_lookup
   0.93%  Xorg             libc-2.28.so                      [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
   0.83%  Xorg             [kernel.kallsyms]                 [k] alloc_vmap_area
   0.63%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_slice_alloc
   0.53%  gnome-shell      libgirepository-1.0.so.1.0.0      [.] g_base_info_unref
   0.53%  gnome-shell      ld-2.28.so                        [.] _dl_find_dso_for_object
   0.49%  kswapd0          [kernel.kallsyms]                 [k] vma_interval_tree_iter_next
   0.48%  gnome-shell      libpthread-2.28.so                [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock
```

其他资源

- perf-report (1) man page

第 22 章 使用 PERF 创建 UPROBES

22.1. 在功能级别使用 PERF 创建 UPROBES

您可以使用 perf 工具在进程或应用程序的任意点处创建动态追踪点。然后，这些追踪点可以与其他 perf 工具（如 perf stat 和 perf 记录 ）一起使用，以更好地理解进程或应用程序行为。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

1. 在对进程或应用程序感兴趣的位置监控的进程或应用程序中创建 uprobe：

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function   (on function in /path/to/executable)

You can now use it in all perf tools, such as:

        perf record -e probe_executable:function -aR sleep 1
```

其他资源

- perf-probe man page
- 使用 perf 记录和分析性能配置集
- 使用 perf stat 在进程执行过程中计算事件

22.2. 在带有 PERF 的函数内创建 UPROBES

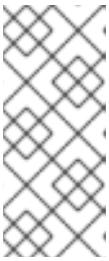
然后，这些追踪点可以与其他 perf 工具（如 perf stat 和 perf 记录 ）一起使用，以更好地理解进程或应用程序行为。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

- 您已获得可执行文件的调试符号：

```
# objdump -t ./your_executable | head
```



注意

要做到这一点，必须安装可执行文件的 debuginfo 软件包；或者，如果可执行文件是本地开发的应用程序，则必须使用调试信息（GCC 中的 -g 选项）编译应用程序。

流程

1. 查看可放置 uprobe 的功能行：

```
$ perf probe -x ./your_executable -L main
```

这个命令的输出结果类似如下：

```
<main@/home/user/my_executable:0>
          0  int main(int argc, const char **argv)
          1  {
                 int err;
                 const char *cmd;
                 char sbuf[STRERR_BUFSIZE];

                 /* libsubcmd init */
          7        exec_cmd_init("perf", PREFIX, PERF_EXEC_PATH,
EXEC_PATH_ENVIRONMENT);
          8        pager_init(PERF_PAGER_ENVIRONMENT);
```

2. 为所需的功能行创建 uprobe：

```
# perf probe -x ./my_executable main:8
Added new event:
      probe_my_executable:main_L8   (on main:8 in /home/user/my_executable)

    You can now use it in all perf tools, such as:

        perf record -e probe_my_executable:main_L8 -aR sleep 1
```

22.3. 通过 UPROBES 记录的数据 PERF 脚本输出

通过 uprobes 分析收集的数据的常用方法是使用 perf script 命令来读取 perf.data 文件，并显示记录的工作负载的详细跟踪。

在 perf 脚本示例输出中：

- uprobe 被添加到名为 my_prog 的程序中的函数 isprime（）中
- a 是添加到 uprobe 中的一个函数参数。或者，a 可以是您在添加 uprobe 的代码范围中可见的任意变量：

```
# perf script
   my_prog  1367 [007] 10802159.906593: probe_my_prog:isprime: (400551) a=2
   my_prog  1367 [007] 10802159.906623: probe_my_prog:isprime: (400551) a=3
   my_prog  1367 [007] 10802159.906625: probe_my_prog:isprime: (400551) a=4
   my_prog  1367 [007] 10802159.906627: probe_my_prog:isprime: (400551) a=5
   my_prog  1367 [007] 10802159.906629: probe_my_prog:isprime: (400551) a=6
   my_prog  1367 [007] 10802159.906631: probe_my_prog:isprime: (400551) a=7
   my_prog  1367 [007] 10802159.906633: probe_my_prog:isprime: (400551) a=13
   my_prog  1367 [007] 10802159.906635: probe_my_prog:isprime: (400551) a=17
   my_prog  1367 [007] 10802159.906637: probe_my_prog:isprime: (400551) a=19
```

第 23 章 使用 PERF MEM 分析内存访问

您可以使用 perf mem 命令对系统上的内存访问进行示例。

23.1. PERF MEM 的目的

perf 工具的 mem 子命令启用内存访问抽样（加载和存储）。perf mem 命令提供有关内存延迟、内存访问类型、导致缓存命中和丢失的功能，并通过记录数据符号（这些点和丢失的内存位置）。

23.2. 使用 PERF MEM 抽样内存访问

这个步骤描述了如何使用 perf mem 命令示例系统中的内存访问示例。该命令使用与 perf record 和 perf report 相同的选项，以及一些选项专用于 mem 子命令。记录的数据保存在当前目录中的 perf.data 文件中，以便稍后进行分析。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

1. 内存访问示例：

```
# perf mem record -a sleep seconds
```

这个示例所有 CPU 的抽样内存访问，覆盖的时间是 sleep 命令指定的 *seconds* 秒数。您可以替换您要在其中示例内存访问数据的任何命令的 sleep 命令。默认情况下，perf mem 样本会加载和存储。您可以使用 -t 选项并选择一个内存操作，并指定 perf mem 和 record 之间的 "load" 或 "store"。对于负载，捕获内存层次结构级别的信息、TLB 内存访问、总线 snoops 和内存锁定。

2. 打开 perf.data 文件进行分析：

```
# perf mem report
```

如果您使用了示例命令，输出为：

```
Available samples
35k cpu/mem-loads,ldlat=30/P
54k cpu/mem-stores/P
```

cpu/mem-loads,ldlat=30/P 行表示通过内存负载收集的数据，cpu/mem-stores/P 行表示通过内存存储收集的数据。突出显示感兴趣的类别，然后按 Enter 键以查看数据：

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 4067062
Overhead       Samples  Local Weight  Memory access            Symbol
Shared Object               Data Symbol                                  Data Object
Snoop       TLB access           Locked
  0.07%           29  98          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.06%           26  97          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.06%           25  96          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.06%           1  2325         Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e9084
[kernel.kallsyms]                None        L1 or L2 hit          No
  0.06%           1  2247         Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e8164
[kernel.kallsyms]                None        L1 or L2 hit          No
  0.05%           1  2166         L1 or L1 hit            [.] 0x00000000038140d6
libxul.so                 [.] 0x00007ffd7b84b4a8                       [stack]
None        L1 or L2 hit          No
  0.05%           1  2117         Uncached or N/A hit     [k] check_for_unclaimed_mmio
[kernel.kallsyms]           [k] 0xffffb092c1842300
[kernel.kallsyms]                None        L1 or L2 hit          No
  0.05%           22  95          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.05%           1  1898         L1 or L1 hit            [.] 0x0000000002a30e07
libxul.so                 [.] 0x00007f610422e0e0                       anon
None        L1 or L2 hit          No
  0.05%           1  1878         Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e8164
[kernel.kallsyms]                None        L2 miss               No
  0.04%           18  94          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.04%           1  1593         Local RAM or RAM hit    [.] 0x00000000026f907d
libxul.so                 [.] 0x00007f3336d50a80                       anon
Hit         L2 miss               No
  0.03%           1  1399         L1 or L1 hit            [.] 0x00000000037cb5f1
libxul.so                 [.] 0x00007fbe81ef5d78                       libxul.so
None        L1 or L2 hit          No
  0.03%           1  1229         LFB or LFB hit          [.] 0x0000000002962aad
libxul.so                 [.] 0x00007fb6f1be2b28                       anon
None        L2 miss               No
```

```
  0.03%          1  1202       LFB or LFB hit          [.] __pthread_mutex_lock
libpthread-2.29.so          [.] 0x00007fb75583ef20                                anon
None        L1 or L2 hit         No
  0.03%          1  1193       Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e9164
[kernel.kallsyms]                None        L2 miss             No
  0.03%          1  1191       L1 or L1 hit            [k] azx_get_delay_from_lpib
[kernel.kallsyms]           [k] 0xffffb092ca7efcf0
[kernel.kallsyms]                None        L1 or L2 hit        No
```

或者，您也可以对结果进行排序，以便在显示数据时调查感兴趣的不同方面。例如，在抽样周期内按类型内存访问对内存负载进行排序，以降低其帐户的开销：

```
# perf mem -t load report --sort=mem
```

例如，输出可以是：

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 40670
Overhead       Samples  Memory access
  31.53%          9725  LFB or LFB hit
  29.70%         12201  L1 or L1 hit
  23.03%          9725  L3 or L3 hit
  12.91%          2316  Local RAM or RAM hit
   2.37%           743  L2 or L2 hit
   0.34%             9  Uncached or N/A hit
   0.10%            69  I/O or N/A hit
   0.02%           825  L3 miss
```

其他资源

- perf-mem (1) man page.

23.3. PERF MEM 报告输出的解释

运行 perf mem report 命令显示的表，没有任何修饰符将数据排序为几列：

'Overhead' 列

表示该特定功能中收集的整体样本的百分比。

'Samples' 列

显示该行所指定的示例数量。

"Local Weight" 列

在处理器核心周期中显示访问延迟。

'Memory Access' 列

显示发生的内存访问类型。

'Symbol' 列

显示功能名称或符号。

'Shared Object' 列

显示示例来自内核的 ELF 镜像的名称（当样本来自内核时使用名称 [kernel.kallsyms]）。

'Data Symbol' 列

显示行目标的内存位置的地址。

重要

通常，由于被访问的内存或堆栈内存的动态分配，"Data Symbol"列将显示原始地址。

"Snoop" 列

显示总线事务。

'TLB Access' 列

显示 TLB 内存访问。

'Locked' 列

指明某个函数是或者没有内存锁定。

在默认模式中，功能按照降序排列，首先显示最高的开销。

第 24 章 检测错误共享

当 Symmetric Multi Processing (SMP) 系统上的处理器核心修改供其他处理器使用的相同缓存行上的数据项时，会发生错误。

这个初始修改要求另一个使用缓存行的处理器使其副本无效，并且请求更新后一，尽管处理器不需要，甚至可能具有修改的数据项的更新版本。

您可以使用 perf c2c 命令检测 false 共享。

24.1. PERF C2C 的目的

perf 工具的 c2c 子命令启用 Shared Data Cache-to-Cache (C2C) 分析。您可以使用 perf c2c 命令检查 cache-line contention 来检测 true 和 false 共享。

当 Symmetric Multi Processing (SMP) 系统中的处理器内核修改由其他处理器使用的同一缓存行上的数据项时，缓存行争用会出现这种情况。使用这个缓存行的所有其他处理器都必须使其副本无效，并请求更新过。这会导致性能下降。

perf c2c 命令提供以下信息：

- 缓存被检测到竞争的行
- 读取和写入数据的进程
- 导致竞争的说明
- 涉及内容的 Non-Uniform Memory Access (NUMA) 节点

24.2. 使用 PERF C2C 检测缓存行竞争

使用 perf c2c 命令检测系统中的缓存行争用。

perf c2c 命令支持与 perf record 相同的选项，以及 c2c 子命令的一些选项。记录的数据保存在当前目录中的 perf.data 文件中，以便稍后进行分析。

先决条件

- 已安装 perf 用户空间工具。如需更多信息，请参阅安装 perf。

流程

- 使用 perf c2c 检测缓存行争用：

```
# perf c2c record -a sleep seconds
```

这个示例抽样并记录所有 CPU 中的缓存行内容数据，覆盖的时间为 sleep 命令指定的 *seconds* 秒数。您可以使用您要收集缓存数据的任何命令替换 sleep 命令。

其他资源

- perf-c2c (1) man page

24.3. 视觉化使用 PERF C2C 记录所记录的 PERF.DATA 文件

此流程描述了如何视觉化 perf.data 文件，该文件使用 perf c2c 命令记录。

先决条件

- 已安装 perf 用户空间工具。如需更多信息，请参阅安装 perf。
- 使用 perf c2c 命令记录的 perf.data 文件位于当前目录中。如需更多信息，请参阅使用 perf c2c 检测到缓存行争用。

流程

1. 打开 perf.data 文件以进一步分析：

```
# perf c2c report --stdio
```

此命令可在终端中将 perf.data 文件视觉化到多个图中：

```
=================================================
        Trace Event Information
=================================================
Total records                   :    329219
Locked Load/Store Operations    :     14654
Load Operations                 :     69679
Loads - uncacheable             :         0
Loads - IO                      :         0
Loads - Miss                    :      3972
Loads - no mapping              :         0
Load Fill Buffer Hit            :     11958
Load L1D hit                    :     17235
Load L2D hit                    :        21
Load LLC hit                    :     14219
Load Local HITM                 :      3402
Load Remote HITM                :     12757
Load Remote HIT                 :      5295
Load Local DRAM                 :       976
Load Remote DRAM                :      3246
Load MESI State Exclusive       :      4222
Load MESI State Shared          :         0
Load LLC Misses                 :     22274
LLC Misses to Local DRAM        :       4.4%
LLC Misses to Remote DRAM       :      14.6%
LLC Misses to Remote cache (HIT) :      23.8%
LLC Misses to Remote cache (HITM) :      57.3%
Store Operations                :    259539
Store - uncacheable             :         0
Store - no mapping              :        11
Store L1D Hit                   :    256696
Store L1D Miss                  :      2832
No Page Map Rejects             :      2376
Unable to parse data source     :         1

=================================================
  Global Shared Cache Line Event Information
=================================================
Total Shared Cache Lines        :        55
Load HITs on shared lines       :     55454
Fill Buffer Hits on shared lines :     10635
L1D hits on shared lines        :     16415
L2D hits on shared lines        :         0
LLC hits on shared lines        :      8501
Locked Access on shared lines   :     14351
Store HITs on shared lines      :    109953
Store L1D hits on shared lines  :    109449
Total Merged records            :    126112

=================================================
            c2c details
=================================================
Events                          : cpu/mem-loads,ldlat=30/P
```

```
                              : cpu/mem-stores/P
 Cachelines sort on               : Remote HITMs
 Cacheline data groupping         : offset,pid,iaddr

=================================================
   Shared Data Cache Line Table
=================================================
#
#                        Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
Load Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index          Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1       L2       Llc       Rmt
# .....  ..................  .......  .......  .......  .......  .......  .......  .......  .......  ........  ........  .......  .......  .......
.......  .......  ........  ........
#
    0          0x602180   149904   77.09%    12103     2269     9834   109504   109036      468
727      2657    13747    40400     5355    16154        0     2875      529
    1          0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056      652
    2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
    3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
    4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0
10       25       62       11        1        0       24        7

=================================================
    Shared Cache Line Distribution Pareto
=================================================
#
#       ----- HITM -----  -- Store Refs --        Data address                          ---------- cycles
----------       cpu                                Shared
#   Num      Rmt      Lcl   L1 Hit  L1 Miss              Offset      Pid        Code address  rmt
hitm  lcl hitm      load       cnt             Symbol              Object                Source:Line
Node{cpu list}
# .....  .......  .......  .......  .......  ...................  .......  ...................  ........  ........  ........  ........
.....................  .....................  .............................  ....
#
  -------------------------------------------------------------
    0    9834     2269   109036      468           0x602180
  -------------------------------------------------------------
       65.51%   55.88%   75.20%    0.00%                0x0    14604          0x400b4f     27161
26039     26017        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:144   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
    0.41%    0.35%    0.00%    0.00%                0x0    14604          0x400b56     18088
12601     26671        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:145   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
    0.00%    0.00%   24.80%  100.00%                0x0    14604          0x400b61        0        0
0        9  [.] read_write_func  no_false_sharing.exe  false_sharing_example.c:145   0{0-
1,4}  1{24-25,120}  2{48,54}  3{169}
    7.50%    9.92%    0.00%    0.00%               0x20    14604          0x400ba7      2470
1729      1897        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:154   1{122}  2{144}
   17.61%   20.89%    0.00%    0.00%               0x28    14604          0x400bc1      2294
1575      1649        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:158   2{53}  3{170}
```

```
  8.97%  12.96%   0.00%   0.00%            0x30   14604         0x400bdb     2325
1897     1828       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:162   0{96}  3{171}

 -------------------------------------------------------------
   1    2832    1119       0       0         0x602100
 -------------------------------------------------------------
  29.13%  36.19%   0.00%   0.00%            0x20   14604         0x400bb3     1964
1230     1788       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:155   1{122}  2{144}
  43.68%  34.41%   0.00%   0.00%            0x28   14604         0x400bcd     2274
1566     1793       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:159   2{53}  3{170}
  27.19%  29.40%   0.00%   0.00%            0x30   14604         0x400be7     2045
1247     2011       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:163   0{96}  3{171}
```

24.4. PERF C2C 报告输出的解释

运行 perf c2c report --stdio 命令显示的视觉化会将数据排序为几个表格中：

跟踪事件信息

这个表提供了 perf c2c record 命令收集的所有负载和存储样本的高级概述。

全局共享缓存行事件信息

此表提供了共享缓存行的统计信息。

c2c 详情

此表提供有关记录事件的信息以及 perf c2c report 数据在视觉化中组织的方式。

共享数据缓存行表

这个表格为检测到 false 共享的热测试缓存行提供了一行摘要，并默认按每个缓存行检测到的远程 Hitm 量降序排列。

共享缓存行分发 Pareto

这个表提供有关每个缓存行遇到竞争的各种信息：

- 缓存行在 NUM 列中编号，从 0 开始。
- 每个缓存行的虚拟地址都包含在 Data address Offset 列中，随后是偏移到发生不同访问权限的缓存行中。

- Pid 列包含进程 ID。
- Code Address 列包含指令指针代码地址。
- cycles 标签下的列显示平均负载延迟。
- cpu cnt 列中显示来自多少个不同 CPU 样本（本质上，等待在给定位置索引的数据不同 CPU 数量）。
- Symbol 列显示功能名称或符号。
- Shared Object 列显示来自示例的 ELF 镜像的名称（当样本来自内核时使用 [kernel.kallsyms] ）。
- Source:Line 列显示源文件和行号。
- Node{cpu list} 列显示每个节点来自哪些特定 CPU 样本。

24.5. 使用 PERF C2C 检测错误共享

这个步骤描述了如何使用 perf c2c 命令检测错误共享。

先决条件

- 已安装 perf 用户空间工具。如需更多信息，请参阅安装 perf。
- 使用 perf c2c 命令记录的 perf.data 文件位于当前目录中。如需更多信息，请参阅使用 perf c2c 检测到缓存行争用。

流程

1. 打开 perf.data 文件以进一步分析：

```
# perf c2c report --stdio
```

这会在终端中打开 perf.data 文件。

2. 在 "Trace Event Information" 表中，找到包含 LLC Misses to Remote Cache (HITM) 的值的行：

LLC Misses to Remote Cache (HITM) 行的值列中的百分比表示被修改的 cache-lines 中的 NUMA 丢失的次数，并出现一个关键指示符错误。

```
=================================================
          Trace Event Information
=================================================
  Total records                     :     329219
  Locked Load/Store Operations      :      14654
  Load Operations                   :      69679
  Loads - uncacheable               :          0
  Loads - IO                        :          0
  Loads - Miss                      :       3972
  Loads - no mapping                :          0
  Load Fill Buffer Hit              :      11958
  Load L1D hit                      :      17235
  Load L2D hit                      :         21
  Load LLC hit                      :      14219
  Load Local HITM                   :       3402
  Load Remote HITM                  :      12757
  Load Remote HIT                   :       5295
  Load Local DRAM                   :        976
  Load Remote DRAM                  :       3246
  Load MESI State Exclusive         :       4222
  Load MESI State Shared            :          0
  Load LLC Misses                   :      22274
  LLC Misses to Local DRAM          :       4.4%
  LLC Misses to Remote DRAM         :      14.6%
  LLC Misses to Remote cache (HIT)  :      23.8%
  LLC Misses to Remote cache (HITM) : 57.3%
  Store Operations                  :     259539
  Store - uncacheable               :          0
  Store - no mapping                :         11
  Store L1D Hit                     :     256696
  Store L1D Miss                    :       2832
  No Page Map Rejects               :       2376
  Unable to parse data source       :          1
```

3. 检查 Shared Data Cache Line Table 的 LLC Load Hitm 字段的 Rmt 列：

```
=================================================
            Shared Data Cache Line Table
=================================================
#
#                        Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
Load Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index          Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1       L2       Llc       Rmt
# .....  ..................  .......  .......  .......  .......  .......  .......  .......  .......  ........  ........  .......  .......  .......
.......  .......  ........  ........
#
     0          0x602180   149904   77.09%    12103     2269     9834   109504   109036
468      727     2657    13747    40400     5355    16154        0     2875       529
     1          0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056      652
     2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
     3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
     4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0
10       25       62       11        1        0       24        7
```

这个表根据每个缓存行检测到的远程 Hitm 的数量降序排列。LLC Load Hitm 部分的 Rmt 列中的数量很高，需要进一步检查调试 false 共享活动的缓存行。

第 25 章 FLAMEGRAPHS 入门

作为系统管理员，您可以使用 flamegraphs 来创建使用 perf 工具记录的系统性能数据的可视化。作为软件开发人员，您可以使用 flamegraphs 创建使用 perf 工具记录的应用程序性能数据的视觉化。

抽样堆栈追踪是使用 perf 工具分析 CPU 性能的常见技术。不幸的是，利用 perf 的性能分析堆栈跟踪的结果可能非常详细，对其进行分析是一个人工密集型的工作。flamegraphs 是从通过 perf 记录的数据创建的视觉化呈现，以便更快、更轻松地识别热代码路径。

25.1. 安装 FLAMEGRAPHS

要开始使用 flamegraphs，请安装所需软件包。

流程

- 安装 flamegraphs 软件包：

```
# dnf install js-d3-flame-graph
```

25.2. 在整个系统中创建 FLAMEGRAPHS

这个步骤描述了如何使用 flamegraphs 来视觉化在整个系统中记录的性能数据。

先决条件

- 如安装 flamegraphs 所述 安装 flamegraphs。
- perf 工具安装如安装 perf 所述。

流程

- 记录数据并创建视觉化：

```
# perf script flamegraph -a -F 99 sleep 60
```

该命令会抽样并记录整个系统的性能数据（使用 sleep 命令确定），然后作为

flamegraph.html 形式将保存在当前活动目录中的视觉化显示。覆盖的时间为 60 秒。命令默认示例调用数据，并在特定情况下使用与 perf 工具相同的参数：

-a

保证记录整个系统的数据。

-F

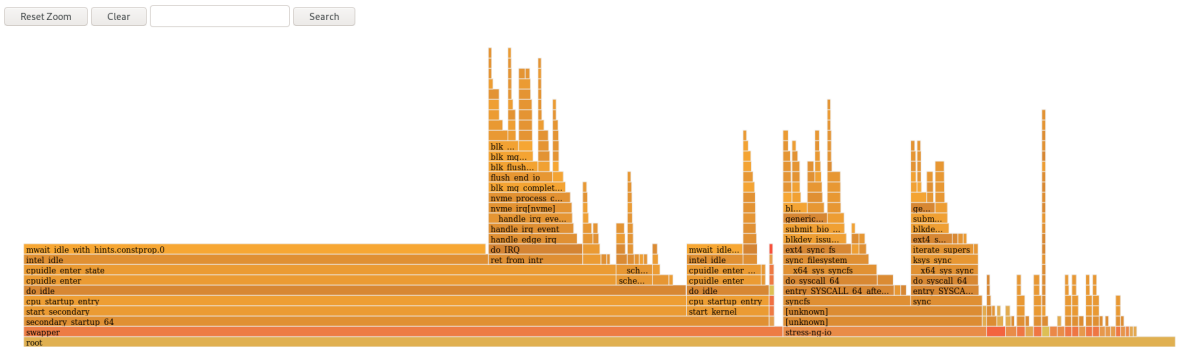
设置每秒抽样频率。

验证步骤

- **要分析，请查看生成的视觉化：**

```
# xdg-open flamegraph.html
```

这个命令在默认浏览器中打开视觉化：



25.3. 在特定进程中创建 FLAMEGRAPHS

您可以使用 flamegraphs 来视觉化在特定运行中的进程中记录的性能数据。

先决条件

- **如安装 flamegraphs 所述 安装 flamegraphs。**

- perf 工具安装如安装 perf 所述。

流程

- 记录数据并创建视觉化：

```
# perf script flamegraph -a -F 99 -p ID1,ID2 sleep 60
```

该命令会取样并记录进程 ID 为 *ID1* 和 *ID2* 的进程的 60 秒的性能数据，使用 sleep 命令确定，然后以 flamegraph.html 形式保存在当前活动目录中的视觉化。命令默认示例调用数据，并在特定情况下使用与 perf 工具相同的参数：

-a

保证记录整个系统的数据。

-F

设置每秒抽样频率。

-p

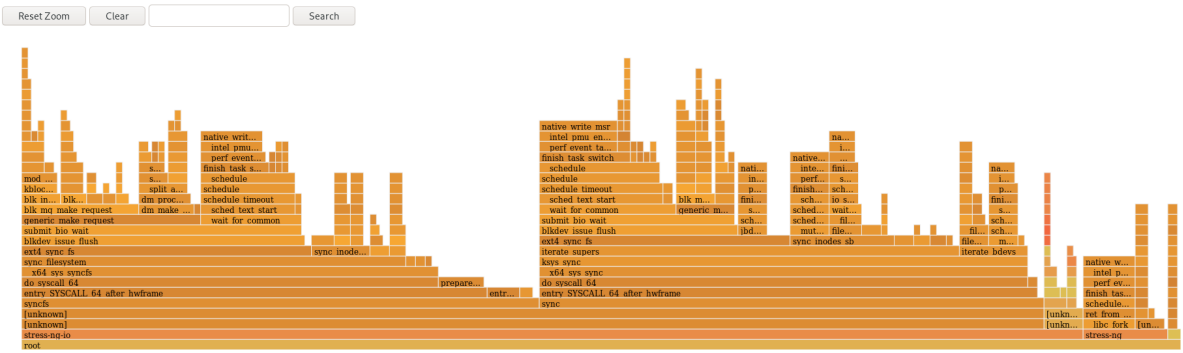
要推断特定进程 ID 的示例并记录数据。

验证步骤

- 要分析，请查看生成的视觉化：

```
# xdg-open flamegraph.html
```

这个命令在默认浏览器中打开视觉化：



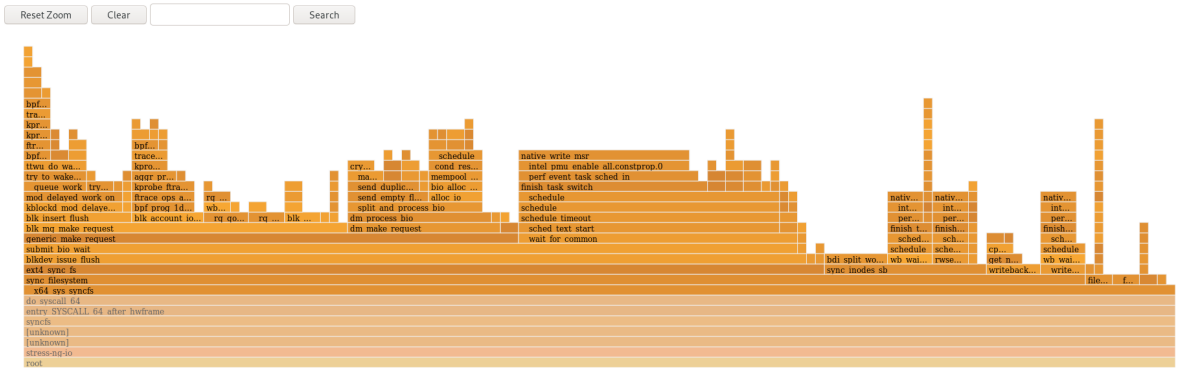
25.4. 解释 FLAMEGRAPHS

flamegraph 的每个框代表堆栈中的不同功能。y 轴显示堆栈在顶部框的深度，每个堆栈都是实际处于 CPU 的函数，以及它处于ancestry 下的一切。x 轴显示抽样的 call-graph 数据的填充。

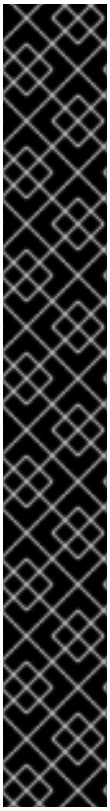
给定行中堆栈的子项会根据在 x 轴之间以降序排列的样本数量显示，x 轴并不代表传递时间。更广泛的一个单独框是，在数据被抽样时，它更频繁地处于 CPU 或一个 on-CPU ancestry 的一部分。

流程

- 要显示之前未显示的功能名称，进一步调查数据，点 flamegraph 中的方框，将缩放到指定位置的堆栈中：

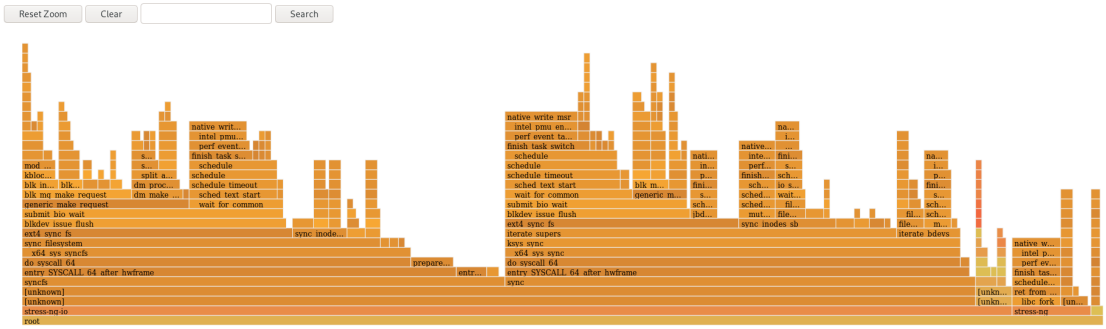


- 要返回 flamegraph 的默认视图，请点 Reset Zoom。



重要

代表用户空间功能的框可以在 flamegraphs 中被标记为 Unknown，因为函数的二进制文件被剥离。必须安装可执行文件的 debuginfo 软件包，或者如果可执行文件是本地开发的应用程序，则必须使用调试信息进行编译。使用 GCC 中的 -g 选项，在这种情形中显示功能名称或符号。



其他资源

- 为什么 perf 显示为原始功能地址
- 使用调试信息启用调试

第 26 章 监控使用 PERF 环形缓冲的性能瓶颈

您可以创建环形缓冲区，它使用 perf 工具获取特定于事件的数据快照，以监控您系统上运行的特定进程或部分性能瓶颈。在这种情况下，perf 仅将数据写入 perf.data 文件，以便在检测到指定事件时进行后续分析。

26.1. 使用 PERF 环缓冲缓冲和特定于事件的快照

在通过 perf 对进程或应用中调查性能问题时，在发生特定事件前数小时内可能无法记录数据。在这种情况下，您可以使用 perf record 来创建自定义环形缓冲，该缓冲区在特定事件后拍摄快照。

--overwrite 选项会使 perf record 将所有数据存储在可被覆盖的循环缓冲区中。当缓冲区已满时，perf record 会自动覆盖最旧的记录，因此永远不会被写入 perf.data 文件。

将 --overwrite 和 --switch-output-event 选项一起配置循环缓冲区，记录并持续转储数据，直到它检测到 --switch-output-event 触发器事件。对 perf record 的事件信号发生与用户相关的内容，并将 circular 缓冲区中的数据写入 perf.data 文件。这会收集您感兴趣的特定数据，这会同时减少运行 perf 进程的开销，因为您不需要的数据不会写入 perf.data 文件。

26.2. 收集特定数据以监控使用 PERF 环形缓冲的性能瓶颈

使用 perf 工具，您可以创建由您指定的事件触发的环形缓冲，以便仅收集您感兴趣的数据。要创建收集事件数据的 circular 缓冲，请将 --overwrite 和 --switch-output-event 选项用于 perf。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 您已在想要监测的进程或应用程序中放置了一个 uprobe：

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function   (on function in /path/to/executable)

You can now use it in all perf tools, such as:

        perf record -e probe_executable:function -aR sleep 1
```

流程

- **使用 uprobe 作为触发器事件创建环形缓冲：**

```
# perf record --overwrite -e cycles --switch-output-event probe_executable:function
./executable
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012231959 ]
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232008 ]
^C[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232082 ]
[ perf record: Captured and wrote 5.621 MB perf.data.<timestamp> ]
```

这个示例启动可执行文件并收集 cpu 周期（在 -e 选项后指定），直到 perf 检测到 uprobe（在 --switch-output-event 选项后指定的触发器事件）。此时，perf 采用循环缓冲区中所有数据的快照，并将其存储在由时间戳标识的唯一 perf.data 文件中。此示例生成总共 2 个快照，最后的 perf.data 文件是通过按 Ctrl+c 强制执行的。

第 27 章 在不停止或重启 PERF 的情况下从正在运行的 PERF 收集器添加或删除追踪点

通过使用控制管道接口在正在运行的 perf 收集器中启用和禁用不同的追踪点，您可以动态调整收集的数据，而无需停止或重启 perf。这样可确保您不会丢失在停止或重启过程中记录的性能数据。

27.1. 在没有停止或重启 PERF 的情况下向正在运行的 PERF 添加追踪点

使用控制管道接口向正在运行的 perf 收集器添加追踪点，以调整您正在记录的数据，而无需停止 perf 和丢失性能数据。

先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。

流程

1. 配置控制管道接口：

```
# mkfifo control ack perf.pipe
```

2. 使用您启用的控制文件设置和事件运行 perf 记录 ：

```
# perf record --control=fifo:control,ack -D -1 --no-buffering -e 'sched:*' -o - > perf.pipe
```

在本例中，在 -e 选项后声明 'sched:*' 使用调度程序事件启动 perf record。

3. 在第二个终端中，启动控制管道的读取权限：

```
# cat perf.pipe | perf --no-pager script -i -
```

启动控制管道的读取侧会在第一个终端中触发以下信息：

```
Events disabled
```

4. 在第三个终端中，使用 control 文件启用 tracepoint：

```
# echo 'enable sched:sched_process_fork' > control
```

此命令触发 perf，以扫描控制文件中针对声明的事件扫描当前事件列表。如果事件存在，则会启用 tracepoint，并在第一个终端中显示以下消息：

```
event sched:sched_process_fork enabled
```

启用追踪点后，第二个终端会显示 perf 检测追踪点的输出：

```
bash 33349 [034] 149587.674295: sched:sched_process_fork: comm=bash pid=33349
child_comm=bash child_pid=34056
```

27.2. 在不停止或重启 PERF 的情况下从正在运行的 PERF 收集器中除追踪点

使用控制管道接口从正在运行的 perf 收集器中删除追踪点，以减少收集的数据范围，而无需停止 perf 和丢失性能数据。

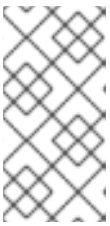
先决条件

- 已安装 perf 用户空间工具，如安装 perf 所述。
- 您已通过控制管道接口向正在运行的 perf 收集器添加追踪点。如需更多信息，请参阅在不停止或重启 perf 的情况下向正在运行的 perf 收集器添加追踪点。

流程

- 删除追踪点：

```
# echo 'disable sched:sched_process_fork' > control
```



注意

本例假设您之前已将调度程序事件加载到控制文件中，并启用 tracepoint sched_process_fork。

此命令触发 perf，以扫描控制文件中针对声明的事件扫描当前事件列表。如果事件存在，则会禁用 tracepoint，并在用于配置控制管道的终端中显示以下消息：

```
event sched:sched_process_fork disabled
```

第 28 章 使用 NUMASTAT 分析内存分配

使用 numastat 工具，您可以显示系统中内存分配的统计信息。

numastat 工具会单独显示每个 NUMA 节点的数据。您可以使用这些信息来调查系统的内存性能，或者系统上的不同内存策略的有效性。

28.1. 默认 NUMASTAT 统计

默认情况下，numastat 工具显示各个 NUMA 节点的这些类别数据的统计信息：

numa_hit

成功分配给此节点的页面数量。

numa_miss

由于预期节点上的内存较低，在此节点上分配的页面数量。每个 numa_miss 事件在另一个节点上都有对应的 numa_foreign 事件。

numa_foreign

最初用于分配给另一节点的页面数量。每个 numa_foreign 事件在另一节点上都有对应的 numa_miss 事件。

interleave_hit

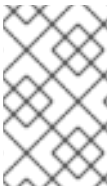
成功分配给此节点的交集策略页面数量。

local_node

此节点上的进程在这个节点上成功分配的页面数量。

other_node

通过另一节点上的进程在这个节点上分配的页面数量。



注意

高 numa_hit 值和低 numa_miss 值（相对于彼此）代表优化的性能。

28.2. 使用 NUMASTAT 查看内存分配

您可以使用 numastat 工具查看系统的内存分配。

先决条件

- 安装 numactl 软件包：

```
# dnf install numactl
```

流程

- 查看系统的内存分配：

```
$ numastat
                           node0          node1
numa_hit                76557759       92126519
numa_miss               30772308       30827638
numa_foreign            30827638       30772308
interleave_hit            106507         103832
local_node              76502227       92086995
other_node              30827840       30867162
```

其他资源

- numastat (8) 手册页

第 29 章 配置操作系统以优化 CPU 使用率

您可以配置操作系统来优化跨工作负载的 CPU 使用率。

29.1. 监控和诊断处理器问题的工具

以下是 Red Hat Enterprise Linux 9 中用于监控并诊断处理器相关性能问题的工具：

- turbostat 工具以指定间隔打印计数器结果，以帮助管理员识别服务器中的意外行为，如过量电源使用、无法进入深度睡眠状态或系统管理中断 (SMI) 创建不必要。
- numactl 实用程序提供了多个选项来管理处理器和内存关联性。numactl 软件包包含 libnuma 库，它为内核支持的 NUMA 策略提供简单编程接口，并可用于比 numactl 应用更精细的调优。
- numastat 工具显示操作系统及其进程的每个 NUMA 节点内存统计信息，并演示进程内存是否在整个系统中分散，还是集中于特定的节点上。此工具由 numactl 软件包提供。
- numad 是一个自动 NUMA 关联性管理守护进程。它监控系统中的 NUMA 拓扑和资源使用情况，以便动态改进 NUMA 资源分配和管理。
- /proc/interrupts 文件显示中断请求 (IRQ) 编号、系统中的每个处理器处理的类似中断请求数量、中断发送的类型以及响应所列中断请求的设备的逗号分隔列表。
- pqos 程序在 intel-cmt-cat 软件包中提供。它监控最近 Intel 处理器上的 CPU 缓存和内存带宽。它监控：
  - 每个周期的说明 (IPC) 。
  - 最后一次缓存 MISSES 的计数。
  - 在 LLC 中给定 CPU occupies 中程序执行的大小（以 KB 为单位）。

- 到本地内存的带宽 (MBL) 。
- 远程内存的带宽 (MBR) 。

- x86_energy_perf_policy 工具让管理员能够定义与性能和能源效率相对的重要性。然后，当这些信息选择在性能和能源效率之间权衡的选项时，可以使用这些信息来影响支持此功能的处理器。

- taskset 工具由 util-linux 软件包提供。它允许管理员检索和设置正在运行的进程的处理器关联，或启动具有指定处理器关联性的进程。

其他资源

- turbostat (8) , numactl (8) , numastat (8) , numa (7) , numad (8) , pqos (8) , x86_energy_perf_policy (8) , 和 taskset (1) man page

29.2. 系统拓扑类型

在现代计算中，CPU 的概念有误导，因为大多数现代系统具有多个处理器。系统的拓扑是这些处理器互相连接并与其他系统资源的连接的方式。这可能会影响系统和应用程序性能，以及系统的调优注意事项。

以下是现代计算中使用的两种主要拓扑类型：

对称多进程 (SMP) 拓扑

SMP 拓扑允许所有处理器在相同时间内访问内存。但是，因为共享和相同的内存访问本质上会阻止所有 CPU 进行序列化内存访问，SMP 系统扩展限制通常被视为不可接受的。因此，所有现代服务器系统都是 NUMA 机器。

非统一内存访问 (NUMA) 拓扑

NUMA 拓扑是比 SMP 拓扑更久而开发的。在 NUMA 系统中，多个处理器通过插槽被物理分组。每个套接字都有一个专用的内存和处理器区域，它们对该内存进行本地访问，它们统称为节点。同一节点上的处理器对该节点的内存银行具有高速度访问，而对内存银行而非其节点上的内存银行而言较慢。

因此，访问非本地内存时会有一个性能损失。因此，具有 NUMA 拓扑的系统上性能敏感的应用程序应该访问与执行应用程序的处理器相同的内存，并应尽可能避免访问远程内存。

对于性能敏感的多线程应用程序，这些应用程序可能会被配置为在特定 NUMA 节点上执行，而不是特定处理器。这是否适当地取决于您的系统和应用程序的要求。如果多个应用程序线程访问同一缓存的数据，那么可将这些线程配置为在同一处理器上执行。但是，如果多个访问和缓存不同数据在同一处理器上执行的线程，每个线程可能会驱除由上一线程访问的缓存数据。这意味着，每个线程"misses"缓存，浪费执行时间从内存中获取数据并在缓存中替换。使用 perf 工具检查是否有过多的缓存未命中。

29.2.1. 显示系统拓扑

有许多命令可帮助了解系统拓扑。这个步骤描述了如何确定系统拓扑。

步骤

- 显示系统拓扑概述：

```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
[...]
```

- 要收集有关 CPU 架构的信息，如 CPU、线程、内核、插槽和 NUMA 节点的数量：

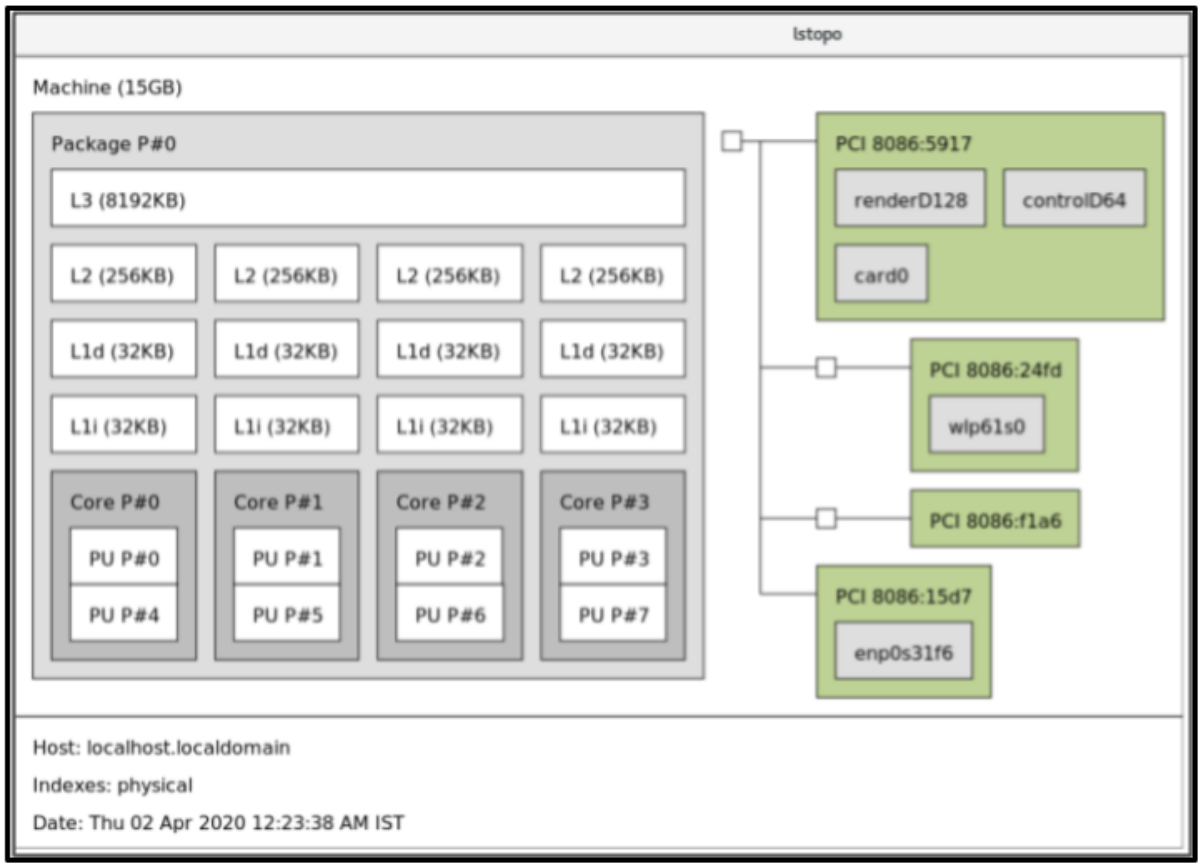
```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
Socket(s):             4
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 47
Model name:            Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:              2
CPU MHz:               2394.204
BogoMIPS:              4787.85
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
```

```
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):     2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):     1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):     3,7,11,15,19,23,27,31,35,39
```

- 查看系统的图形表示：

```
# dnf install hwloc-gui
# lstopo
```

图 **29.1. lstopo 输出**



- 查看详细文本输出：

```
# dnf install hwloc
# lstopo-no-graphics
Machine (15GB)
  Package L#0 + L3 L#0 (8192KB)
    L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
        PU L#0 (P#0)
        PU L#1 (P#4)
      HostBridge L#0
    PCI 8086:5917
        GPU L#0 "renderD128"
        GPU L#1 "controlD64"
        GPU L#2 "card0"
    PCIBridge
```

```
      PCI 8086:24fd
        Net L#3 "wlp61s0"
    PCIBridge
      PCI 8086:f1a6
    PCI 8086:15d7
      Net L#4 "enp0s31f6"
```

其他资源

- **numactl (8) , lscpu (1) , 和 lstopo (1) man pages**

29.3. 配置内核空循环时间

默认情况下，Red Hat Enterprise Linux 9 使用无空循环内核，它不会中断空闲的 CPU 来降低功耗，并允许新处理器利用深度睡眠状态。

Red Hat Enterprise Linux 9 还提供了动态无空选项，这对于对延迟敏感型工作负载（如高性能计算或实时计算）非常有用。默认情况下禁用动态无空选项。红帽建议使用 cpu-partitioning TuneD 配置集为指定为 isolated_cores 的内核启用动态无空选项。

这个步骤描述了如何永久启用动态无数性行为。

步骤

1. 要在特定内核中启用动态无空行为，在内核命令行中使用 nohz_full 参数指定这些核心。在 16 核系统上，启用 nohz_full=1-15 内核选项：

```
# grubby --update-kernel=ALL --args="nohz_full=1-15"
```

这启用了内核 1 到 15 的动态无数性行为，将所有时间保留到唯一未指定的内核（内核 0）。

2. 当系统引导时，手动将 rcu 线程移到非延迟敏感的内核中，在本例中是 core 0 ：

```
# for i in `pgrep rcu[^c]` ; do taskset -pc 0 $i ; done
```

3. 可选：在内核命令行中使用 isolcpus 参数，将特定内核与用户空间任务隔离开来。

4. **可选：将内核的 write-back bdi-flush 线程的 CPU 关联性设置为 housekeeping 内核：**

```
echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
```

验证步骤

- **系统重启后，验证是否启用了 dynticks ：**

```
# journalctl -xe | grep dynticks
Mar 15 18:34:54 rhel-server kernel: NO_HZ: Full dynticks CPUs: 1-15.
```

- **验证动态无空配置是否正常工作：**

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
```

这个命令会在 CPU 1 上测量升级，同时告诉 CPU 1 休眠 3 秒。

- **默认内核计时器配置在常规 CPU 上显示大约 3100 勾号：**

```
# perf stat -C 0 -e irq_vectors:local_timer_entry taskset -c 0 sleep 3

 Performance counter stats for 'CPU(s) 0':

             3,107      irq_vectors:local_timer_entry

       3.001342790 seconds time elapsed
```

- **配置动态无数内核后，您应该会看到大约 4 个空循环：**

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3

 Performance counter stats for 'CPU(s) 1':

                 4      irq_vectors:local_timer_entry

       3.001544078 seconds time elapsed
```

其他资源

- **perf (1) 和 cpuset (7) man page**

- 关于 nohz_full 内核参数红帽知识库文章
- 如何验证 sysfs 中的"隔离"和"nohz_full" CPU 信息列表？红帽知识库文章

29.4. 中断请求概述

中断请求或 IRQ 是从硬件立即发送到处理器的信号。系统中的每个设备都会被分配一个或多个 IRQ 编号，允许它发送唯一的中断。启用中断后，接收中断请求的处理器会立即暂停当前应用程序线程执行，以处理中断请求。

由于中断中断会停止正常操作，因此高中断率可能会严重降低系统性能。通过配置中断的关联性，或者向批处理中发送多个较低优先级中断（协调多个中断），这可以减少中断所花费的时间。

中断请求具有关联的关联性属性 smp_affinity，它定义了处理中断请求的处理器。要提高应用性能，请将中断关联和进程关联分配到同一处理器，或分配到同一内核上的处理器。这允许指定的中断和应用程序线程共享缓存行。

在支持中断中断的系统上，修改中断请求的 smp_affinity 属性可设置硬件，以便决定使用特定处理器在硬件级别提供中断，而无需在内核中干预。

29.4.1. 手动平衡中断

如果您的 BIOS 导出它的 NUMA 拓扑，则 irqbalance 服务可自动为节点上对请求服务的硬件进行中断请求。

步骤

1. 检查哪些设备对应于您要配置的中断请求。
2. 查找平台的硬件规格。检查您系统上的芯片组是否支持分发中断。
   a. 如果这样做，您可以按照以下步骤中的内容配置中断交付。另外，检查您的芯片组用来平衡中断的算法。有些 BIOS 有一些选项来配置中断交付。
   b.

如果没有，您的芯片组总会将所有中断路由到单个静态 CPU。您无法配置使用哪些 CPU。

3. 检查系统上使用了 Advanced Programmable Interrupt Controller (APIC) 模式：

```
$ journalctl --dmesg | grep APIC
```

在这里，

- 如果您的系统使用 flat 以外的模式，您可以看到一个类似于 Setting APIC routing to physical flat 的行。
- 如果看不到这个信息，代表您的系统使用 flat 模式。

如果您的系统使用 x2apic 模式，您可以在 引导装载程序配置 的内核命令行中添加 nox2apic 选项来禁用它。

只有非物理平面模式（flat）支持将中断分发到多个 CPU。这个模式仅适用于最多 8 个 CPU 的系统。

4. 计算 smp_affinity 掩码。有关如何计算 smp_affinity mask 的更多信息，请参阅 设置 smp_affinity 掩码。

其他资源

- journalctl (1) 和 taskset (1) man page

29.4.2. 设置 smp_affinity 掩码

smp_affinity 值存储为代表系统中所有处理器的十六进制位掩码。每个位配置不同的 CPU。最重要的位是 CPU 0。

掩码的默认值为 f，这意味着可在系统中的任何处理器上处理中断请求。将此值设置为 1 表示只有处理器 0 可以处理中断。

步骤

1. 二进制代码中，将值 1 用于处理中断的 CPU。例如，要设置 CPU 0 和 CPU 7 以处理中断，请使用 0000000010000001 作为二进制代码：

表 29.1. CPU 的二进制位

CPU	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
二进制	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

2. 将二进制代码转换为十六进制代码：

例如，使用 Python 转换二进制代码：

```
>>> hex(int('0000000010000001', 2))
'0x81'
```

在有 32 个处理器的系统上，您必须限制离散的 32 位组的 smp_affinity 值。例如，如果您只想 64 位处理器系统的第一个 32 个处理器来服务中断请求，请使用 0xffffffff,00000000。

3. 特定中断请求的中断关联性值存储在关联的 /proc/irq/irq_number/smp_affinity 文件中。在此文件中设置 smp_affinity mask：

```
# echo mask > /proc/irq/irq_number/smp_affinity
```

其他资源

- journalctl (1) , irqbalance (1) , 和 taskset (1) man pages

第 30 章 调优调度策略

在 Red Hat Enterprise Linux 中，进程执行的最小单元称为线程。系统调度程序决定哪个处理器运行线程，以及线程的运行时间。但是，因为调度程序的主要关注是保持系统忙碌，因此可能无法为应用程序性能最佳调度线程。

例如，当 Node B 上的处理器可用时，NUMA 系统上的应用程序在 Node A 上运行。为了将处理器保持在节点 B忙碌上，调度程序会将其中一个应用的线程移至节点 B。但是，应用程序线程仍然需要访问 Node A 上的内存。但是，这个内存会更长时间访问，因为线程现在在 Node B 和 Node A 内存上运行，而 Node A 内存不再是线程本地的。因此，线程完成在 Node B 上运行的时间可能要长于在 Node B 上运行，等待 Node A 上的处理器变得可用，然后在具有本地内存访问的原始节点上执行线程。

30.1. 调度策略的类别

性能敏感的应用程序通常受益于设计人员或管理员确定运行线程的位置。Linux 调度程序实施多个调度策略，用于决定线程的运行时间和时长。

以下是调度策略的两个主要类别：

普通策略

常规线程用于普通优先级的任务。

实时策略

实时策略用于不需要中断时必须完成的时间敏感任务。实时线程不会受到时间分片。这意味着线程会运行直到它们 block、exit、voluntarily yield 或被较高优先级线程抢占。

任何具有一般策略线程的线程前，会调度最低优先级实时线程。如需更多信息，请参阅使用 SCHED_FIFO 的静态优先级调度和使用 SCHED_RR 进行循环优先级调度。

其他资源

- sched (7) , sched_setaffinity (2) , sched_getaffinity (2) , sched_setscheduler (2) , 和 sched_getscheduler (2) man pages

30.2. 使用 SCHED_FIFO 的静态优先级调度

SCHED_FIFO 也称为静态优先级调度，是一种实时策略，为每个线程定义固定优先级。此策略允许管

理员提高事件响应时间并缩短延迟。建议您不要在时间敏感时执行此策略。

当使用 SCHED_FIFO 时，调度程序会按照优先级顺序扫描所有 SCHED_FIFO 线程的列表，并调度可随时运行的最高优先级线程。SCHED_FIFO 线程的优先级级别可以是从 1 到 99 的任何整数，其中 99 被视为最高优先级。红帽建议仅在识别延迟问题时以较低数量开始并增加优先级。

警告

因为实时线程不会受到时间分片，因此红帽不推荐将优先级设置为 99。这与迁移和 watchdog 线程相同的优先级级别保持您的进程；如果您的线程进入计算循环，并且这些线程被阻止，则它们将无法运行。具有单一处理器的系统最终会在这种情况下挂起。

管理员可以限制 SCHED_FIFO 带宽，以防止实时应用程序程序员启动对处理器进行单调执行的实时任务。

以下是此策略中使用的一些参数：

/proc/sys/kernel/sched_rt_period_us

此参数以微秒为单位定义时间，它被视为处理器带宽的 10%。默认值为 1000000 InventoryServices，或 1 秒。

/proc/sys/kernel/sched_rt_runtime_us

此参数以微秒为单位定义运行实时线程的时间周期。默认值为 950000 µs，即 0.95 秒。

30.3. 使用 SCHED_RR 循环优先级调度

SCHED_RR 是 SCHED_FIFO 的循环变体。当多个线程需要在同一优先级级别上运行时，此策略很有用。

与 SCHED_FIFO 一样，SCHED_RR 是一个实时策略，用于为每个线程定义固定优先级。调度程序会按照优先级顺序扫描所有 SCHED_RR 线程的列表，并调度可随时运行的最高优先级线程。但是，与 SCHED_FIFO 不同，在特定时间片段中以 round-robin 样式调度具有相同优先级的线程。

您可以使用 /proc/sys/kernel/sched_rr_timeslice_ms 文件中的 sched_rr_timeslice_ms 内核参数以毫秒为单位设置这个时间片段的值。最低值为 1 毫秒。

30.4. 使用 SCHED_OTHER 常规调度

SCHED_OTHER 是 Red Hat Enterprise Linux 9 中的默认调度策略。此策略使用完全公平调度程序 (CFS)，允许对使用该策略调度的所有线程进行公平处理器访问。当有大量线程或数据吞吐量是优先级时，此策略最有用，因为它可以更有效地调度线程。

当使用此策略时，调度程序会根据每个进程线程的 niceness 值创建动态优先级列表。管理员可以更改进程的 niceness 值，但不能直接更改调度程序的动态优先级列表。

30.5. 设置调度程序策略

使用 chrt 命令行工具检查并调整调度程序策略和优先级。它可以启动具有所需属性的新进程，或更改正在运行的进程的属性。它还可用于在运行时设置策略。

步骤

1. 查看活跃进程的进程 ID (PID)：

```
# ps
```

在 ps 命令中使用 --pid 或 -p 选项来查看特定 PID 的详细信息。

2. 检查特定进程的调度策略、PID 和优先级：

```
# chrt -p 468
pid 468's current scheduling policy: SCHED_FIFO
pid 468's current scheduling priority: 85

# chrt -p 476
pid 476's current scheduling policy: SCHED_OTHER
pid 476's current scheduling priority: 0
```

在这里，*468* 和 *476* 是进程的 PID。

3. 设置进程的调度策略：

a. 例如，要将 PID 为 *1000* 的进程设置为 *SCHED_FIFO*，其优先级为 *50* ：

```
# chrt -f -p 50 1000
```

b. 例如，要将 PID 为 *1000* 的进程设置为 *SCHED_OTHER*，其优先级为 *0* ：

```
# chrt -o -p 0 1000
```

c. 例如，要将 PID 为 *1000* 的进程设置为 *SCHED_RR*，其优先级为 *10* ：

```
# chrt -r -p 10 1000
```

d. 要启动具有特定策略和优先级的新应用，请指定应用程序的名称：

```
# chrt -f 36 /bin/my-app
```

其他资源

- **chrt (1) man page**
- chrt 命令的策略选项
- 在引导过程中更改服务优先级

30.6. CHRT 命令的策略选项

使用 chrt 命令，您可以查看和设置进程的调度策略。

下表描述了适当的策略选项，可用于设置进程的调度策略。

表 30.1. chrt 命令的策略选项

短选项	长选项	描述
-f	**--fifo**	将调度设置为 **SCHED_FIFO**
-o	**--other**	将调度设置为 **SCHED_OTHER**
-r	**--rr**	将调度设置为 **SCHED_RR**

30.7. 在引导过程中更改服务优先级

使用 systemd 服务时，可以在引导过程中为启动的服务设置实时优先级。*单元配置指令* 用于在引导过程中更改服务的优先级。

引导过程优先级更改通过使用 service 部分中的以下指令进行：

CPUSchedulingPolicy=

设置已执行进程的 CPU 调度策略。它被用于设置 other, fifo, 和 rr 策略。

CPUSchedulingPriority=

设置已执行进程的 CPU 调度优先级。可用的优先级范围取决于所选的 CPU 调度策略。对于实时调度策略，可以使用 1（最低优先级）和 99 （最高优先级）之间的整数。

下面的步骤描述了如何使用 mcelog 服务在引导过程中更改服务的优先级。

先决条件

1. 安装 TuneD 软件包：

```
# dnf install tuned
```

2. 启用并启动 TuneD 服务：

```
# systemctl enable --now tuned
```

流程

1. 查看正在运行的线程的调度优先级：

```
# tuna --show_threads
                    thread       ctxt_switches
   pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 1     OTHER    0    0xff      3181          292          systemd
 2     OTHER    0    0xff       254            0        kthreadd
 3     OTHER    0    0xff         2            0          rcu_gp
 4     OTHER    0    0xff         2            0      rcu_par_gp
 6     OTHER    0       0         9            0 kworker/0:0H-kblockd
 7     OTHER    0    0xff      1301            1 kworker/u16:0-events_unbound
 8     OTHER    0    0xff         2            0    mm_percpu_wq
 9     OTHER    0       0       266            0     ksoftirqd/0
[...]
```

2. 创建附加 mcelog 服务配置文件，并在该文件中插入策略名称和优先级：

```
# cat << EOF > /etc/systemd/system/mcelog.service.d/priority.conf

[Service]
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=20
EOF
```

3. 重新载入 systemd 脚本配置：

```
# systemctl daemon-reload
```

4. 重启 mcelog 服务：

```
# systemctl restart mcelog
```

验证步骤

- 显示 systemd 问题设置的 mcelog 优先级：

```
# tuna -t mcelog -P
thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
826     FIFO    20  0,1,2,3         13            0          mcelog
```

其他资源

- systemd (1) 和 tuna (8) man page
- 优先级范围的描述

30.8. 优先级映射

优先级在组中定义，有一些组专用于特定内核功能。对于实时调度策略，可以使用 1（最低优先级）和 99 （最高优先级）之间的整数。

下表描述了优先级范围，可在设置进程的调度策略时使用。

表 30.2. 优先级范围的描述

Priority	线程	描述
1	低优先级内核线程	此优先级通常为需要超过 **SCHED_OTHER** 的任务保留。
2 - 49	可供使用	用于典型的应用程序优先级的范围。
50	默认 hard-IRQ 值	
51 - 98	高优先级线程	对定期执行的线程使用此范围，且必须快速响应时间。不要将此范围用于 CPU 密集型线程，因为您将中断。
99	Watchdogs 和 migration	必须以最高优先级运行的系统线程。

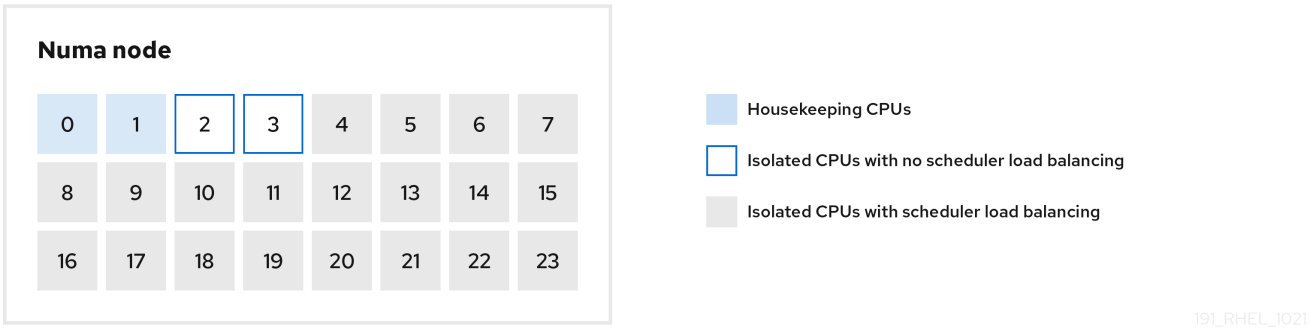
30.9. TUNED CPU-PARTITIONING 配置集

要为对延迟敏感的工作负载调整 Red Hat Enterprise Linux 9，红帽建议使用 cpu-partitioning TuneD 配置集。

在 Red Hat Enterprise Linux 9 之前，低延迟 Red Hat 文档描述了实现低延迟调整所需的大量低级别步骤。在 Red Hat Enterprise Linux 9 中，您可以使用 cpu-partitioning TuneD 配置集更有效地执行低延迟性能优化。根据个人低延迟应用程序的要求，此配置集可轻松自定义。

下图显示了如何使用 cpu-partitioning 配置集。这个示例使用 CPU 和节点布局。

图 30.1. cpu-partitioning 图



您可以使用以下配置选项在 /etc/tuned/cpu-partitioning-variables.conf 文件中配置 cpu-partitioning 配置集：

带有负载均衡的隔离 CPU

在 cpu-partitioning 图中，从 4 到 23 编号的块是默认的隔离 CPU。在这些 CPU 上启用了内核调度程序的进程负载均衡。它专为需要内核调度程序负载平衡的多个线程的低延迟进程而设计。

您可以使用 isolated_cores=cpu-list 选项在 /etc/tuned/cpu-partitioning-variables.conf 文件中配置 cpu-partitioning 配置集，它列出了 CPU 来隔离将使用内核调度程序负载平衡。

隔离的 CPU 列表用逗号分开，也可以使用一个短划线（如 3-5 ）指定范围。这个选项是必须的。这个列表中缺少的任何 CPU 会自动被视为内务 CPU。

没有负载均衡的隔离 CPU

在 cpu-partitioning 图中，编号为 2 和 3 的块是不提供任何其他内核调度程序进程负载均衡的隔离 CPU。

您可以使用 no_balance_cores=cpu-list 选项在 /etc/tuned/cpu-partitioning-variables.conf 文件中配置 cpu-partitioning 配置集，它列出了不使用内核调度程序负载平衡的 CPU。

指定 no_balance_cores 选项是可选的，但此列表中的任何 CPU 都必须是 isolated_cores 列表中所列 CPU 的子集。

使用这些 CPU 的应用程序线程需要单独固定到每个 CPU。

日常 CPU

在 cpu-partitioning-variables.conf 文件中没有隔离的 CPU 会自动被视为内务 CPU。在内务 CPU 上，允许执行所有服务、守护进程、用户进程、可移动内核线程、中断处理程序和内核计时器。

其他资源

- tuned-profiles-cpu-partitioning (7) man page

30.10. 使用 TUNED CPU-PARTITIONING 配置集进行低延迟调整

这个步骤描述了如何使用 TuneD 的 cpu-partitioning 配置集为低延迟调整系统。它使用了低延迟应用的示例，它可以使用 cpu-partitioning 和 CPU 布局，如 cpu-partitioning 图中所述。

本例中的应用程序使用了：

- 从网络读取数据的专用的 reader 线程将固定到 CPU 2。
- 处理此网络数据的大量线程将固定到 CPU 4-23。
- 将处理的数据写入网络的专用写入器线程将固定到 CPU 3。

先决条件

- 您已以 root 用户身份，使用 dnf install tuned-profiles-cpu-partitioning 命令安装 cpu-partitioning TuneD 配置集。

步骤

1. 编辑 /etc/tuned/cpu-partitioning-variables.conf 文件并添加以下信息：

```
# All isolated CPUs:
isolated_cores=2-23
# Isolated CPUs without the kernel's scheduler load balancing:
no_balance_cores=2,3
```

2. 设置 **cpu-partitioning TuneD** 配置集：

```
# tuned-adm profile cpu-partitioning
```

3. 重启

重新引导后，将根据 **cpu-partitioning** 图中的隔离，为低延迟调优。该应用可以使用 **taskset** 将读取器和写入器线程固定到 **CPU 2** 和 **3**，以及 **CPU 4-23** 上剩余的应用程序线程。

其他资源

- **tuned-profiles-cpu-partitioning (7) man page**

30.11. 自定义 CPU-PARTITIONING TUNED 配置集

您可以扩展 **TuneD** 配置集，以进行额外的性能优化更改。

例如，**cpu-partitioning** 配置集将 **CPU** 设置为使用 **cstate=1**。要使用 **cpu-partitioning** 配置集，但额外将 **CPU cstate** 从 **cstate1** 更改为 **cstate0**，以下流程描述了一个新的 **TuneD** 配置集，名称为 ***my_profile***，它继承 **cpu-partitioning** 配置集，然后设置 **C state-0**。

步骤

1. 创建 **/etc/tuned/my_profile** 目录：

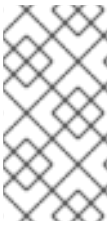
```
# mkdir /etc/tuned/my_profile
```

2. 在此目录中创建 **tuned.conf** 文件并添加以下内容：

```
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. 使用新配置集：

```
# tuned-adm profile my_profile
```



注意

在共享示例中，不需要重新启动。但是，如果 ***my_profile*** 配置集中的更改需要重新引导才能生效，则重新启动计算机。

其他资源

- **tuned-profiles-cpu-partitioning (7) man page**

第 31 章 调整网络性能

调整网络设置是一个复杂的过程，需要考虑很多因素。例如，这包括 CPU 到内存的架构、CPU 核数等。对于大多数场景，Red Hat Enterprise Linux 使用优化的默认设置。然而，在某些情况下，有必要调整网络设置，以提高吞吐量或延迟或解决问题，如数据包丢弃。

31.1. 调整网络适配器设置

在 40 Gbps 及更快的高速度网络中，一些与网络适配器相关的内核设置的默认值可能是数据包丢弃和性能降低的原因。调整这些设置可能会防止此类问题。

31.1.1. 使用 nmcli 增加环缓冲区的大小，以减少该数据包丢弃率

如果数据包丢包率导致应用程序报告数据丢失、超时或其他问题，请增加以太网设备的环缓冲的大小。

接收环缓冲在设备驱动程序和网络接口控制器(NIC)之间共享。该卡分配一个传输(TX)和接收(RX)环缓冲。名称意味着，环缓冲是循环缓冲区，溢出会覆盖现有数据。可以通过两种方法将数据从 NIC 移至内核，硬件中断和软件中断也称为 SoftIRQ。

内核使用 RX 环缓冲区来存储传入的数据包，直到设备驱动程序可以处理它们为止。设备驱动程序排空 RX 环，通常通过使用 SoftIRQ，其将传入的数据包放到名为 sk_buff 或 skb 的内核数据结构中，通过内核并直至拥有相关套接字的应用程序来开始其过程。

内核使用 TX 环缓冲区来存放应发送到网络的传出数据包。这些环缓冲位于堆栈的底部，是可能会发生数据包丢弃的关键点，进而会对网络性能造成负面影响。

步骤

1. 显示接口的数据包丢包统计信息：

```
# ethtool -S enp1s0
  ...
  rx_queue_0_drops: 97326
  rx_queue_1_drops: 63783
  ...
```

请注意，命令的输出取决于网卡和驱动程序。

discard 或 drop 计数器中的高值表示可用缓冲区的填满速度快于内核处理数据包的速度。增加环缓冲有助于避免此类丢失。

2. 显示最大环缓冲大小：

```
# ethtool -g enp1s0
Ring parameters for enp1s0:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       16320
TX:             4096
Current hardware settings:
RX:             255
RX Mini:        0
RX Jumbo:       0
TX:             255
```

如果 Pre-set maximums 部分中的值大于 Current hardware settings 部分的值，您可以在下一步中更改设置。

3. 确定使用接口的 NetworkManager 连接配置文件：

```
# nmcli connection show
NAME                UUID                                  TYPE      DEVICE
Example-Connection  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1s0
```

4. 更新连接配置文件，并增加环缓冲：

- 要增加 RX 环缓冲区，请输入：

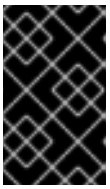
```
# nmcli connection modify Example-Connection ethtool.ring-rx 4096
```

- 要增加 TX 环缓冲区，请输入：

```
# nmcli connection modify Example-Connection ethtool.ring-tx 4096
```

5. 重新载入 NetworkManager 连接：

```
# nmcli connection up Example-Connection
```



重要

根据 NIC 使用的驱动程序，环缓冲中的更改可能会短暂中断网络连接。

其他资源

- ifconfig 和 ip 命令报告数据包丢弃
- 我是否应该关注 0.05% 的数据包丢弃率？
- ethtool(8) 手册页

31.1.2. 调整网络设备积压队列以避免数据包丢弃

当网卡接收数据包并在内核协议堆栈处理它们之前，内核将这些数据包存储在积压队列中。内核为每个 CPU 核维护一个单独的队列。

如果核的积压队列满了，则内核丢弃 netif_receive_skb（） 内核功能分配给此队列的所有将来传入的数据包。如果服务器包含 10 Gbps 或更快的网络适配器或多个 1 Gbps 适配器，请调整积压队列大小以避免此问题。

先决条件

- 一个 10 Gbps 或更快的或多个 1 Gbps 网络适配器

步骤

1. 确定是否需要调整积压队列，显示 /proc/net/softnet_stat 文件中的计数器：

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
221951548  0      0  0  0  0  0  0  0  0  0  0
192058677  18862  0  0  0  0  0  0  0  0  0  1
455324886  0      0  0  0  0  0  0  0  0  0  2
...
```

此 awk 命令将 /proc/net/softnet_stat 中的值从十六进制转换为十进制格式，并以表格式显示它们。每行表示以核 0 开始的 CPU 核。

相关的列有：

- 第一列：收到的总帧数
- 第二列：因为积压队列满了而丢弃的帧数
- 最后一列：CPU 核数

2. 如果 /proc/net/softnet_stat 文件中第二列中的值随着时间递增，请增加积压队列的大小：

   a. 显示当前积压队列大小：

   ```
   # sysctl net.core.netdev_max_backlog
   net.core.netdev_max_backlog = 1000
   ```

   b. 使用以下内容创建 /etc/sysctl.d/10-netdev_max_backlog.conf 文件：

   ```
   net.core.netdev_max_backlog = 2000
   ```

   将 net.core.netdev_max_backlog 参数设置为当前值的两倍。

   c. 从 /etc/sysctl.d/10-netdev_max_backlog.conf 文件载入设置：

   ```
   # sysctl -p /etc/sysctl.d/10-netdev_max_backlog.conf
   ```

验证

- 监控 /proc/net/softnet_stat 文件中的第二列：

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
```

如果值还在增加，请将 net.core.netdev_max_backlog 值再次增加一倍。重复此过程，直到数据包丢弃数不再增加。

31.1.3. 增加 NIC 的传输队列长度，以减少传输错误的数量

内核在传输它们前将数据包存储在传输队列中。默认长度(1000 个数据包)通常对于 10 Gbps 足够，通常对于 40 Gbps 网络也足够。但是，在更快的网络中，或者遇到适配器上的传输错误增加，请增加队列长度。

流程

1. 显示当前传输队列长度：

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
...
```

在本例中，enp1s0 接口的传输队列长度(qlen)是 1000。

2. 监控网络接口的软件传输队列丢弃的数据包数：

```
# tc -s qdisc show dev enp1s0
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5ms
interval 100ms memory_limit 32Mb ecn drop_batch 64
 Sent 16889923 bytes 426862765 pkt (dropped 191980, overlimits 0 requeues 2)
...
```

3. 如果您遇到较高的或不断增加的传输错误数，请设置更高的传输队列长度：

   a. 确定使用此接口的 NetworkManager 连接配置文件：

```
# nmcli connection show
NAME               UUID                                  TYPE      DEVICE
Example-Connection  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1s0
```

   .

b. 更新连接配置文件，并增加传输队列长度：

```
# nmcli connection modify Example-Connection link.tx-queue-length 2000
```

将队列长度设置为当前值的两倍。

c. 应用更改：

```
# nmcli connection up Example-Connection
```

验证

1. 显示传输队列长度：

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 2000
...
```

2. 监控丢弃的数据包数：

```
# tc -s qdisc show dev enp1s0
```

如果 dropped 数还在增加，请再次将传输队列长度增加一倍。重复此过程，直到数不再增加。

31.2. 调整 IRQ 平衡

在多核主机上，您可以通过确保 Red Hat Enterprise Linux 平衡中断队列(IRQ)来在 CPU 核之间分发中断来提高性能。

31.2.1. 中断和中断处理程序

当网络接口控制器(NIC)接收传入数据时，它使用直接内存访问(DMA)将数据复制到内核缓冲区中。然后，NIC 通过触发一个硬中断来向内核通知此数据。这些中断由中断处理程序处理的，这些处理程序做最少的工作，因为它们已中断了另一个任务，处理程序无法中断自己。硬中断在 CPU 用量方面代价高昂，特别是如果它们使用内核锁。

然后，硬中断处理器会将大多数数据包接收留给软件中断请求(SoftIRQ)进程。内核可以更公平地调度这些进程。

例 31.1. 显示硬件中断

内核将中断数存储在 /proc/interrupts 文件中。要显示特定 NIC （如 enp1s0）的计数，请输入：

```
# egrep "CPU|enp1s0" /proc/interrupts
       CPU0    CPU1    CPU2    CPU3    CPU4   CPU5
105:  141606       0       0       0       0      0  IR-PCI-MSI-edge      enp1s0-rx-0
106:       0  141091       0       0       0      0  IR-PCI-MSI-edge      enp1s0-rx-1
107:       2       0  163785       0       0      0  IR-PCI-MSI-edge      enp1s0-rx-2
108:       3       0       0  194370       0      0  IR-PCI-MSI-edge      enp1s0-rx-3
109:       0       0       0       0       0      0  IR-PCI-MSI-edge      enp1s0-tx
```

每个队列在分配给它的第一列中有一个中断向量。当系统引导时，或者当用户载入 NIC 驱动程序模块时，内核会初始化这些向量。每个接收(RX)和传输(TX)队列都会被分配一个唯一的向量，来告知中断处理程序中断来自哪个 NIC 或队列。列代表每个 CPU 核的传入中断数。

31.2.2. 软中断请求

软中断请求(SoftIRQ)清除网络适配器的接收环缓冲。当其他任务不会被中断时，内核调度 SoftIRQ 例程运行。在 Red Hat Enterprise Linux 上，名为 ksoftirqd/*cpu-number* 的进程运行这些例程，并调用特定于驱动程序的代码函数。

要监控每个 CPU 核的 SoftIRQ 数，请输入：

```
# watch -n1 'egrep "CPU|NET_RX|NET_TX" /proc/softirqs'
                CPU0       CPU1   CPU2       CPU3 CPU4    CPU5       CPU6  CPU7
    NET_TX:    49672      52610  28175      97288      12633   19843      18746     220689
    NET_RX:       96       1615       789        46        31    1735       1315     470798
```

命令动态更新输出。按 Ctrl+C 中断输出。

31.2.3. NAPI 轮询

新的 API (NAPI)是处理框架的设备驱动程序数据包的扩展，以提高传入网络数据包的效率。硬中断非常昂贵，因为它们通常会导致上下文在内核空间和用户空间之间来回切换，并且不能中断自己。即使中断

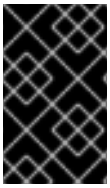
合并，中断处理器也会完全独占 CPU 核。使用 NAPI 时，驱动程序可以对每个接收的数据包使用轮询模式，而不是内核发起的硬中断。

在正常操作下，内核会发出一个初始硬中断，后跟软中断请求(SoftIRQ)处理程序，该处理程序使用 NAPI 例程轮询网卡。为防止 SoftIRQ 独占 CPU 核，轮询例程有一个预算来确定 SoftIRQ 可以使用的 CPU 时间。完成 SoftIRQ 轮询例程后，内核会退出例程，并安排它稍后再次运行，以重复从网卡接收数据包的过程。

31.2.4. irqbalance 服务

在具有和没有非统一内存访问(NUMA)架构的系统上，irqbalance 服务根据系统状况有效地在 CPU 核间平衡中断。irqbalance 服务在后台运行，每 10 秒监控一次 CPU 负载。当 CPU 的负载过高时，服务会将中断移到其他 CPU 核。因此，该系统表现良好，并更有效地处理负载。

如果 irqbalance 没有运行，则通常 CPU 核 0 处理大多数中断。即使在中等负载，这个 CPU 核可能会变得很忙碌，试图处理系统中所有硬件的工作负载。因此，中断或基于中断的工作可能会丢失或延迟。这可能导致网络和存储性能较低、数据包丢失和其他潜在问题。



重要

禁用 irqbalance 可能会对网络吞吐量造成负面影响。

在只有一个 CPU 核的系统上，irqbalance 服务没有提供任何好处，并自行退出。

默认情况下，irqbalance 服务在 Red Hat Enterprise Linux 上启用并运行。要重新启用服务（如果您禁用了它），请输入：

```
# systemctl enable --now irqbalance
```

其他资源

- 是否需要 irqbalance？解决方案
- 如何配置网络接口 IRQ 通道？解决方案

31.2.5. 增加 SoftIRQ 可在 CPU 上运行的时间

如果 SoftIRQ 没有运行足够长时间，则传入数据的速度可能会超过内核排空缓冲区的速度。因此，网络接口控制器(NIC)缓冲区溢出，数据包丢失。

如果 softirqd 进程无法在一个 NAPI 轮询周期中内检索来自接口的所有数据包，则它是一个表示 SoftIRQ 没有足够 CPU 时间的指示器。在具有快速 NIC（如 10 Gbps 和更快）的主机上，可能会出现这种情况。如果您增加 net.core.netdev_budget 和 net.core.netdev_budget_usecs 内核参数的值，您可以控制softirqd在一个轮询周期内处理数据包的时间和数量。

流程

1. 要确定是否需要调整 net.core.netdev_budget 参数，请显示 /proc/net/softnet_stat 文件中计数：

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
221951548  0  0      0  0  0  0  0  0  0  0  0  0
192058677  0  20380  0  0  0  0  0  0  0  0  0  1
455324886  0  0      0  0  0  0  0  0  0  0  0  2
...
```

此 awk 命令将 /proc/net/softnet_stat 中的值从十六进制转换为十进制格式，并以表格式显示它们。每行表示以核 0 开始的 CPU 核。

相关的列有：

- 第一列：收到的总帧数。
- 第三列：softirqd 进程在一个 NAPI 轮询周期内无法从接口检索所有数据包的次数。
- 最后一列：CPU 核号。

2. 如果 /proc/net/softnet_stat 文件的第三列中的计数随着时间而递增，请调整系统：

   a. 显示 net.core.netdev_budget_usecs 和 net.core.netdev_budget 参数的当前值：

```
# sysctl net.core.netdev_budget_usecs net.core.netdev_budget
net.core.netdev_budget_usecs = 2000
net.core.netdev_budget = 300
```

使用这些设置时，softirqd 进程最多有 2000 微秒来在一个轮询周期内处理最多 300 个来自 NIC 的消息。轮询根据哪个条件最先满足而结束。

b. 使用以下内容创建 /etc/sysctl.d/10-netdev_budget.conf 文件：

```
net.core.netdev_budget = 600
net.core.netdev_budget_usecs = 4000
```

将参数设置为其当前值的两倍。

c. 从 /etc/sysctl.d/10-netdev_budget.conf 文件中加载设置：

```
# sysctl -p /etc/sysctl.d/10-netdev_budget.conf
```

验证

- 监控 /proc/net/softnet_stat 文件中的第三列：

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
```

如果值仍然增加，请将 net.core.netdev_budget_usecs 和 net.core.netdev_budget 设置为更高的值。重复此过程，直到计数不再增加。

31.3. 提高网络延迟

CPU 电源管理功能可能会导致对时间敏感的应用程序处理造成不必要的延迟。您可以禁用一些或全部电源管理功能，以提高网络延迟。

例如，如果服务器闲置时的延迟比其在高负载时高，则 CPU 电源管理设置可能会影响延迟。

重要

禁用 CPU 电源管理功能可能会导致更高的功耗和热量损失。

31.3.1. CPU 电源状态如何影响网络延迟

CPU 的消耗状态(C-states)优化并减少计算机的功耗。C-states 从 C0 开始编号。在 C0 中，处理器完全通电并执行。在 C1 中，处理器完全通电，但没有执行。C-state 的数越大，CPU 关闭的组件越多。

每当 CPU 核闲置时，内置的节能逻辑步骤就会介入，并尝试通过关闭各种处理器组件来将核从当前 C-state 移到更高状态。如果 CPU 核必须处理数据，则 Red Hat Enterprise Linux (RHEL)会向处理器发送一个中断，以唤醒核，并将其 C-state 设置回 C0。

从深度 C-states 移回 C0 需要一些时间，因为需要给处理器的各个组件通电。在多核系统上，也可能发生许多核同时闲置的情况，因此处于深度 C-states 。如果 RHEL 尝试同时唤醒它们，则内核可能产生大量处理器间中断(IPI)，同时所有核都从深度 C-states 返回。由于处理中断时需要锁，所以在处理所有中断时系统可能会停滞一段时间。这可能会导致应用程序响应事件的大量延迟。

例 31.2. 显示每个核 C-state 的次数

PowerTOP 应用程序中的 Idle Stats 页面显示CPU 核在每个 C-state 中花费多少时间：

```
        Pkg(HW)  |          Core(HW) |          CPU(OS) 0   CPU(OS) 4
                 |                   | C0 active   2.5%       2.2%
                 |                   | POLL        0.0%   0.0 ms  0.0%   0.1 ms
                 |                   | C1          0.1%   0.2 ms  0.0%   0.1 ms
C2 (pc2)   63.7%   |                 |
C3 (pc3)    0.0%   | C3 (cc3)    0.1%   | C3          0.1%   0.1 ms  0.1%   0.1 ms
C6 (pc6)    0.0%   | C6 (cc6)    8.3%   | C6          5.2%   0.6 ms  6.0%   0.6 ms
C7 (pc7)    0.0%   | C7 (cc7)   76.6%   | C7s         0.0%   0.0 ms  0.0%   0.0 ms
C8 (pc8)    0.0%   |                   | C8          6.3%   0.9 ms  5.8%   0.8 ms
C9 (pc9)    0.0%   |                   | C9          0.4%   3.7 ms  2.2%   2.2 ms
C10 (pc10)  0.0%   |                   |
                 |                   | C10        80.8%   3.7 ms 79.4%   4.4 ms
                 |                   | C1E         0.1%   0.1 ms  0.1%   0.1 ms
...
```

其他资源

- 使用 PowerTOP 管理能耗

31.3.2. EFI 固件中的 C-state 设置

在大多数带有 EFI 固件的系统中，您可以启用和禁用单个消耗状态(C-states)。但是，在 Red Hat Enterprise Linux (RHEL)上，闲置的驱动程序决定内核是否使用固件中的设置：

- intel_idle ：这是带有 Intel CPU 的主机上的默认驱动程序，并忽略 EFI 固件中的 C-state 设置。
- acpi_idle ：如果禁用了 intel_idle，RHEL 在带有 Intel 以外的供应商 CPU 的主机上使用此驱动程序。默认情况下，acpi_idle 驱动程序使用 EFI 固件中的 C-state 设置。

其他资源

- kernel-doc 软件包提供的 /usr/share/doc/kernel-doc-*<version>*/Documentation/admin-guide/pm/cpuidle.rst

31.3.3. 使用自定义 TuneD 配置文件禁用 C-states

TuneD 服务使用内核的电源管理服务质量(PMQOS)接口来设置消耗状态(C-states)锁定。内核闲置驱动程序可以与此接口通信，以动态限制 C-states。这可防止管理员必须使用内核命令行参数来硬编码最大 C-state 值。

先决条件

- tuned 软件包已安装。
- tuned 服务已启用并运行。

流程

1. 显示活跃的配置文件：

```
# tuned-adm active
Current active profile: network-latency
```

2. 为自定义 TuneD 配置文件创建一个目录：

```
# mkdir /etc/tuned/network-latency-custom/
```

3. 使用以下内容创建 /etc/tuned/*network-latency-custom*/tuned.conf 文件：

```
[main]
include=network-latency

[cpu]
force_latency=cstate.id:1|2
```

此自定义配置文件继承来自 network-latency 配置文件的所有设置。force_latency TuneD 参数指定以微秒为单位(DSLs)的延迟。如果 C-state 延迟大于指定的值，则 Red Hat Enterprise Linux 中的空闲驱动程序阻止 CPU 移到更高的 C-state。使用 force_latency=cstate.id:1|2 ，TuneD 首先检查 /sys/devices/system/cpu/cpu_<number>_/cpuidle/state_<cstate.id>_/ 目录是否存在。在这种情况下，TuneD 从这个目录中的 latency 文件中读取延迟值。如果该目录不存在，则 TuneD 使用 2 微秒作为回退值。

4. 激活 *network-latency-custom* 配置文件：

```
# tuned-adm profile network-latency-custom
```

其他资源

- TuneD 入门
- 自定义 TuneD 配置集

31.3.4. 使用内核命令行选项禁用 C-states

processor.max_cstate 和 intel_idle.max_cstat 内核命令行参数配置 CPU 核可以使用的最大消耗状态(C-state) 。例如，将参数设置为 1 确保 CPU 永远不会请求低于 C1 的 C-state。

使用此方法测试主机上应用程序的延迟是否受到 C-states 的影响。不要硬编码一个特定状态，考虑使用更动态的解决方案。请参阅 使用自定义 TuneD 配置文件禁用 C-states。

先决条件

- tuned 服务没有运行或配置为不更新 C-state 设置。

流程

1. 显示系统使用的空闲驱动程序：

```
# cat /sys/devices/system/cpu/cpuidle/current_driver
intel_idle
```

2. 如果主机使用 intel_idle 驱动程序，请设置 intel_idle.max_cstate 内核参数，以定义 CPU 核应该能够使用的最高 C-state：

```
# grubby --update-kernel=ALL --args="intel_idle.max_cstate=0"
```

设置 intel_idle.max_cstate=0 禁用 intel_idle 驱动程序。因此，内核使用 acpi_idle 驱动程序，该驱动程序使用 EFI 固件中设置的 C-state 值。因此，还要设置 processor.max_cstate，以覆盖这些 C-state 设置。

3. 在独立于 CPU 厂商的每个主机上，设置 CPU 核应该能够使用的最高 C-state：

```
# grubby --update-kernel=ALL --args="processor.max_cstate=0"
```

重要

除了 intel_idle.max_cstate=0，如果您设置了 processor.max_cstate=0，acpi_idle 驱动程序会覆盖 processor.max_cstate 的值，并将其设置为 1。因此，使用 processor.max_cstate=0 intel_idle.max_cstate=0，内核将使用的最高 C-state 为 C1，而不是 C0。

4. 重启主机以使更改生效：

```
# reboot
```

验证

1. 显示最大 C-state ：

```
# cat /sys/module/processor/parameters/max_cstate
1
```

2. 如果主机使用 intel_idle 驱动程序，显示最大 C-state ：

```
# cat /sys/module/intel_idle/parameters/max_cstate
0
```

其他资源

- 什么是 CPU "C-states" 以及如何在需要时禁用它们？
- kernel-doc 软件包提供的 /usr/share/doc/kernel-doc-*<version>*/Documentation/admin-guide/pm/cpuidle.rst

31.4. 提高大量连续数据流的吞吐量

根据 IEEE 802.3 标准，没有 Virtual Local Area Network (VLAN)标签的默认以太网帧最大大小为 1518 字节。这些帧的每一个都包含一个 18 字节标头，为有效负载保留 1500 字节。因此，对于服务器通过网络传输的每个 1500 字节数据，就有 18 字节(1.2%)以太网帧标头的开销，并也会被传输。第 3 层和 4 协议的标头进一步增加了每个数据包的开销。

如果您网络上的主机经常发送大量连续的数据流，如备份拥有多个巨型文件的服务器或文件服务器，则考虑使用巨型帧来节省开销。巨型帧是非标准帧，其比 1500 字节的标准以太网有效负载有更大的最大传输单位(MTU)。例如，如果您使用最大允许 9000 字节有效负载的 MTU 配置巨型帧，则每个帧的开销减少到 0.2%。

根据网络和服务，仅在网络的特定部分（如集群的存储后端）启用巨型帧会很有帮助。这可避免数据包碎片。

31.4.1. 配置巨型帧前的注意事项

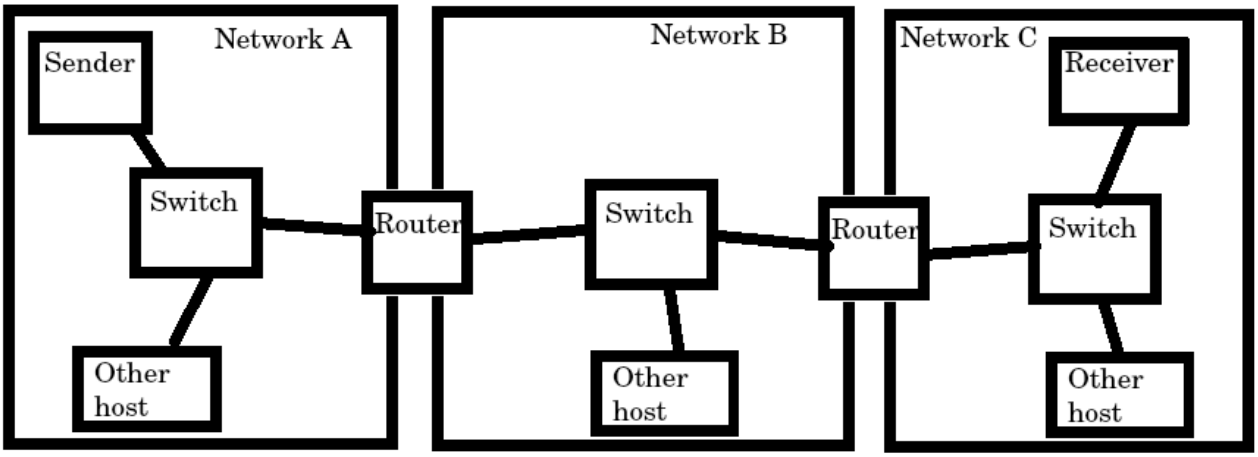
根据您的硬件、应用程序及网络中的服务，巨型帧可能有不同的影响。仔细决定在您的场景中启用巨型帧是否会带来好处。

先决条件

传输路径上的所有网络设备必须支持巨型帧，并使用相同的最大传输单元(MTU)大小。否则，您可能遇到以下问题：

- 丢弃的数据包。
- 由于碎片数据包，延迟更高。
- 增加由碎片导致的数据包丢失的风险。例如，如果路由器将一个 9000 字节帧分为六个 1500 字节帧，并且所有这些 1500 字节帧都丢失了，则整个帧都会丢失，因为它无法重新组装。

在下图中，如果来自网络 A 的主机向网络 C 中的主机发送数据包，则这三个子网中的所有主机都必须使用同样的 MTU：



巨型帧的好处

- 更高的吞吐量：每个帧包含更多的用户数据，同时修复了协议开销。
- CPU 使用率较低：巨型帧导致较少的中断，因此可以节省 CPU 周期。

巨型帧的缺陷

- 更高的延迟：大帧会延迟后面的数据包。
- 增加了内存缓冲区使用率：大帧可以更快地填充缓冲区队列内存。

31.4.2. 在现有 NetworkManager 连接配置文件中配置 MTU

如果您的网络需要与默认值不同的最大传输单元(MTU)，您可以在相应的 NetworkManager 连接配置文件中配置此设置。

巨型帧是，有效负载为 1500 到 9000 字节的网络数据包。同一广播域中的所有设备都必须支持这些帧。

先决条件

- 广播域中的所有设备都使用同样的 MTU。
- 您了解网络的 MTU。
- 您已为具有发散 MTU 的网络配置了连接配置文件。

流程

1. 可选：显示当前的 MTU：

```
# ip link show
...
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. 可选：显示 NetworkManager 连接配置文件：

```
# nmcli connection show
NAME     UUID                                  TYPE      DEVICE
Example  f2f33f29-bb5c-3a07-9069-be72eaec3ecf  ethernet  enp1s0
...
```

3. 在管理到带有分散 MTU 的网络的连接的配置文件中设置 MTU：

```
# nmcli connection modify Example mtu 9000
```

4. 重新激活连接：

```
# nmcli connection up Example
```

验证

1. 显示 MTU 设置：

```
# ip link show
...
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. 验证传输路径上的主机没有对数据包进行碎片处理：

- 在接收方一侧，显示内核的 IP 重组统计信息：

```
# nstat -az IpReasm*
#kernel
IpReasmTimeout 0 0.0
IpReasmReqds 0 0.0
IpReasmOKs 0 0.0
IpReasmFails 0 0.0
```

如果计数器返回 0，则数据包不会重组。

- 在发送方一侧，使用 prohibit-fragmentation-bit 传输 ICMP 请求：

```
# ping -c1 -Mdo -s 8972 destination_host
```

如果命令成功，则不会对数据包进行碎片处理。

按如下所示计算 -s 数据包大小选项的值：MTU size - 8 字节 ICMP 标头 - 20 字节 IPv4 header = 数据包大小

31.5. 为高吞吐量调整 TCP 连接

在 Red Hat Enterprise Linux 上调整与 TCP 相关的设置，以提高吞吐量、减少延迟或防止问题（如数据包丢失）。

31.5.1. 使用 iperf3 测试 TCP 吞吐量

iperf3 工具提供了一个服务器和客户端模式，来在两个主机之间执行网络吞吐量测试。

注意

应用程序的吞吐量取决于许多因素，如应用程序使用的缓冲大小。因此，使用测试工具（如 iperf3）测量的结果可能与生产工作负载下服务器上应用程序的测量结果有很大不同。

先决条件

- iperf3 软件包安装在客户端和服务器上。
- 两个主机上没有其他服务导致严重影响测试结果的网络流量。
- 对于 40 Gbps 和更快的连接，网卡支持 Accelerated Receive Flow Steering (ARFS)，且该功能在接口上被启用。

流程

1. 可选：在服务器和客户端上显示网络接口控制器(NIC)的最大网络速度：

```
# ethtool enp1s0 | grep "Speed"
  Speed: 100000Mb/s
```

2. 在服务器中：

   a. 在 firewalld 服务中临时打开默认的 iperf3 TCP 端口 5201 ：

```
# firewall-cmd --add-port=5201/tcp
# firewall-cmd --reload
```

   b. 在服务器模式下启动 iperf3 ：

```
# iperf3 --server
```

服务现在正在等待传入的客户端连接。

3. 在客户端中：

a. 开始测量吞吐量：

```
# iperf3 --time 60 --zerocopy --client 192.0.2.1
```

- **--time <seconds>**：定义客户端停止传输时的时间（以秒为单位）。

  将此参数设置为您希望正常工作的值，并在稍后的测量中增加它。如果服务器以比传输路径上的设备更快的速度发送数据包，或者客户端可以处理，则可以丢弃数据包。

- **--zerocopy** ：启用零复制方法，而不使用 write（） 系统调用。只有在您想模拟具有零复制的应用程序或在单一流上达到 40 Gbps 和更高时，才需要这个选项。

- **--client <server>** ：启用客户端模式并设置运行 iperf3 服务器的服务器的 IP 地址或名称。

4. 等待 iperf3 完成测试。服务器和客户端都显示每秒的统计信息，并在结尾显示总结。例如，以下是在客户端上显示的总结：

```
[ ID] Interval           Transfer     Bitrate         Retr
[  5] 0.00-60.00  sec  101 GBytes   14.4 Gbits/sec   0   sender
[  5] 0.00-60.04  sec  101 GBytes   14.4 Gbits/sec       receiver
```

在这个示例中，平均比特率为 14.4 Gbps。

5. 在服务器中：

a. 按 Ctrl+C 停止 iperf3 服务器。

b. 关闭 firewalld 中的 TCP 端口 5201：

```
# firewall-cmd --remove-port=5201/tcp
# firewall-cmd --reload
```

其他资源

- iperf3 (1) 手册页

31.5.2. 系统范围的 TCP 套接字缓冲区设置

套接字缓冲区临时存储内核已收到或应发送的数据：

- 读套接字缓冲区保存内核已收到但应用程序尚未读取的数据包。
- 写套接字缓冲区保存应用程序已写到缓冲区，但内核尚未将它们传给 IP 堆栈和网络驱动程序的数据包。

如果 TCP 数据包太大，且超过缓冲区大小或数据包以很快的速度发送或接收，则内核会丢弃任何新传入的 TCP 数据包，直到数据从缓冲区中删除为止。在这种情况下，增加套接字缓冲区可以防止数据包丢失。

net.ipv4.tcp_rmem（读）和 net.ipv4.tcp_wmem（写）套接字缓冲区内核设置包含三个值：

```
net.ipv4.tcp_rmem = 4096  131072  6291456
net.ipv4.tcp_wmem = 4096  16384   4194304
```

显示的值以字节为单位，Red Hat Enterprise Linux 通过以下方式使用它们：

- 第一个值是最小缓冲区大小。新套接字不能有更小的尺寸。
- 第二个值是默认的缓冲区大小。如果应用程序没有设置缓冲区大小，则这是默认值。
- 第三个值是自动调优缓冲区的最大大小。在应用程序中使用带有 SO_SNDBUF 套接字选项

的setsockopt()函数禁用这个最大缓冲区大小。

请注意，net.ipv4.tcp_rmem 和 net.ipv4.tcp_wmem 参数为 IPv4 和 IPv6 协议设置套接字大小。

31.5.3. 增加系统范围的 TCP 套接字缓冲区

系统范围的 TCP 套接字缓冲区临时存储内核已收到或应发送的数据。net.ipv4.tcp_rmem（读）和 net.ipv4.tcp_wmem（写）套接字缓冲区内核设置各包含三个设置：最小值、默认值和最大值。

重要

设置太大的缓冲区大小浪费内存。每个套接字可以设置为应用程序所请求的大小，内核会加倍这个值。例如，如果应用程序请求 256 KiB 套接字缓冲区大小，并打开 100 万个套接字，则系统只能将最多 512 GB RAM (512 KiB x 100万)用于潜在的套接字缓冲空间。

另外，最大缓冲区大小的值太大可能会增加延迟。

先决条件

- 您遇到 TCP 数据包丢弃率很高。

流程

1. 确定连接的延迟。例如，从客户端 ping 服务器以测量平均 Round Trip Time (RTT)：

```
# ping -c 10 server.example.com
...
--- server.example.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 117.208/117.056/119.333/0.616 ms
```

在本例中，延迟为 117 ms。

2. 使用以下公式为您要调整的流量计算 Bandwidth Delay 产品(BDP)：

```
connection speed in bytes * latency in ms = BDP in bytes
```

例如，要计算有 117 ms 延迟的 10 Gbps 连接的 BDP：

```
(10 * 1000 * 1000 * 1000 / 8) * 117 = 10683760 bytes
```

3. 根据您的要求，创建 /etc/sysctl.d/10-tcp-socket-buffers.conf 文件，并设置最大取或写缓冲区大小：

```
net.ipv4.tcp_rmem = 4096 262144 21367520
net.ipv4.tcp_wmem = 4096 24576 21367520
```

指定值（以字节为单位）。当您尝试识别您环境的优化值时，请使用以下经验法则：

- 默认缓冲区大小（第二个值）：仅稍微增加这个值或将其最多设为 524288 (512 KiB)。太高的默认缓冲区大小可能会导致缓冲区崩溃，从而导致延迟激增。
- 最大缓冲区大小（第三个值）：通常 BDP 的两倍到三倍的值就足够了。

4. 从 /etc/sysctl.d/10-tcp-socket-buffers.conf 文件中载入设置：

```
# sysctl -p /etc/sysctl.d/10-tcp-socket-buffers.conf
```

5. 将应用程序配置为使用更大的套接字缓冲区大小。net.ipv4.tcp_rmem 和 net.ipv4.tcp_wmem 参数中的第三个值定义应用程序中 setsockopt （） 函数可以请求的最大缓冲区大小。

详情请查看应用程序的编程语言的文档。如果您不是应用程序的开发人员，请联系开发人员。

6. 如果您更改了 net.ipv4.tcp_rmem 或 net.ipv4.tcp_wmem 参数中的第二个值，请重新启动应用程序以使用新的 TCP 缓冲区大小。

如果您只更改了第三个值，则不需要重启应用程序，因为自动调整会动态应用这些设置。

验证

1. 可选：使用 iperf3 测试 TCP 吞吐量。

2. 使用您遇到数据包丢弃时使用的同样方法来监控数据包丢弃统计信息。

   如果数据包仍然以较低的速率发生，请进一步增加缓冲区大小。

其他资源

- 更改套接字缓冲区大小的含义是什么？
- tcp(7) 手册页
- socket (7) 手册页

31.5.4. TCP 窗口扩展

在 Red Hat Enterprise Linux 中默认启用的 TCP 窗口扩展功能是 TCP 协议的扩展，可显著提高吞吐量。

例如，在具有 1.5 毫秒 Round Trip Time (RTT)的 1 Gbps 连接上：

- 启用 TCP 窗口扩展后，大约 630 Mbps 是可行的。
- 禁用 TCP 窗口扩展后，吞吐量下降到 380 Mbps。

TCP 提供的一个功能是流控制。通过流控制，发送方可以发送接收方所能接收的数据，但不能发送更多数据。为达到此目的，接收方公布一个 window 值，这是发送方可以发送的数据量。

TCP 最初支持最多 64 KiB 的窗口大小，但在高带宽延迟产品(BDP)下，这个值变为一种限制，因为发送方无法一次发送超过 64 KiB 的数据。高速连接可以在给定时间内传输超过 64 KiB 的数据。例如：系统间一个有 1 毫秒延迟的 10 Gbps 链接在给定时间内可传输 1 MiB 的数据。如果主机只发送 64 KiB，然后暂停，直到其他主机接收 64 KiB，则效率很低。

要删除这个瓶颈，TCP 窗口扩展允许 TCP 窗口值算术左移，以将窗口大小增加到 64 KiB 以上。例如，最大窗口值为 65535 将 7 个位置向左移动 7 位，从而产生大约 8 MiB 的窗口大小。这可在给定时间内传输更多数据。

TCP 窗口扩展会在三向 TCP 握手中进行协商，该握手会打开每个 TCP 连接。发件方和接收方必须支持 TCP 窗口扩展功能才能正常工作。如果两个参与者没有在握手中发布窗口扩展功能，则连接将恢复为使用最初的 16 位 TCP 窗口大小。

TCP 窗口扩展默认在 Red Hat Enterprise Linux 中启用：

```
# sysctl net.ipv4.tcp_window_scaling
net.ipv4.tcp_window_scaling = 1
```

如果服务器上禁用了 TCP 窗口扩展(0)，用与您设置它的相同方式恢复设置。

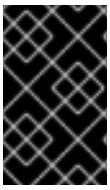
其他资源

- RFC 1323：高性能的 TCP 扩展
- 在运行时配置内核参数

31.5.5. TCP SACK 如何降低数据包丢弃率

在 Red Hat Enterprise Linux (RHEL)中默认启用的 TCP Selective Acknowledgment(TCP SACK)功能是 TCP 协议的一种改进，提高了 TCP 连接的效率。

在 TCP 传输中，接收方为其接收的每个数据包将一个 ACK 数据包给发送方。例如，客户端向服务器发送了 TCP 数据包 1-10，但数据包号 5 和 6 丢失。如果没有 TCP SACK，服务器会丢弃数据包 7-10，客户端必须从丢失点重新传输所有数据包，而这效率较低。如果两个主机上都启用了 TCP SACK，客户端只需重新传输丢失的数据包 5 和 6。



重要

禁用 TCP SACK 会降低性能，并给 TCP 连接中的接收方造成较高的数据包丢弃率。

TCP SACK 默认在 RHEL 中启用。要验证：

```
# sysctl net.ipv4.tcp_sack
1
```

如果您的服务器上禁用了 TCP SACK (0)，用与您设置它的同样的方式恢复设置。

其他资源

- RFC 2018：TCP Selective Acknowledgment 选项
- 我是否应该关注 0.05% 的数据包丢弃率？
- 在运行时配置内核参数

31.6. 调优 UDP 连接

在开始调优 Red Hat Enterprise Linux 以改进 UDP 流量的吞吐量之前，有一个现实的预期很重要。UDP 是一个简单协议。与 TCP 相比，UDP 不包含如，流控制、拥塞控制和数据可靠性等功能。这样很难在吞吐量率接近网络接口控制器(NIC)的最大速度的 UDP 上达到可靠的通信。

31.6.1. 检测数据包丢弃

在内核可以丢弃数据包的网络堆栈中有多个级别。Red Hat Enterprise Linux 提供了不同的工具来显示这些级别的统计信息。使用它们来识别潜在的问题。

请注意，您可以忽略非常小的数据包丢弃率。但是，如果您遇到非常大的丢失率，请考虑调优措施。

注意

如果网络堆栈无法处理传入的流量，则内核会丢弃网络数据包。

流程

1. 确定网络接口控制器(NIC)是否丢弃了数据包：

   a. 显示 NIC 和特定于驱动程序的统计信息：

   ```
   # ethtool -S enp1s0
   NIC statistics:
       ...
       rx_queue_0_drops: 17657
       ...
   ```

   统计的命名以及其是否可用取决于 NIC 和驱动程序。

   b. 显示接口统计信息：

   ```
   # ip -s link show enp1s0
   2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
   state UP mode DEFAULT group default qlen 1000
       link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
       RX:  bytes  packets errors dropped  missed   mcast_
       84697611107 56866482      0   10904       0       0
       TX:  bytes  packets errors dropped carrier collsns_
        5540028184  3722234      0       0       0       0
   ```

   RX 代表接收的数据包和传输的数据包 TX 的统计信息。

2. 识别由于套接字缓冲太小，或者应用程序处理速度较慢导致的特定于 UDP 协议的数据包丢弃：

   ```
   # nstat -az UdpSndbufErrors UdpRcvbufErrors
   #kernel
   UdpSndbufErrors          4    0.0
   UdpRcvbufErrors    45716659    0.0
   ```

   输出中的第二列列出了计数。

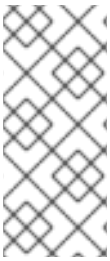
其他资源

- RHEL 网络接口丢弃数据包 解决方案

- 我是否应该关注 0.05% 的数据包丢弃率？

31.6.2. 使用 iperf3 测试 UDP 吞吐量

iperf3 工具提供了一个服务器和客户端模式，来在两个主机之间执行网络吞吐量测试。



注意

应用程序的吞吐量取决于许多因素，如应用程序使用的缓冲区大小。因此，使用测试工具（如 iperf3）测量的结果可能与生产工作负载下服务器上应用程序的测量结果有很大不同。

先决条件

- iperf3 软件包安装在客户端和服务器上。
- 两个主机上没有其他服务导致严重影响测试结果的网络流量。
- 可选：增加服务器和客户端上最大 UDP 套接字大小。详情请参阅 增加系统范围的 UDP 套接字缓冲区。

流程

1. 可选：在服务器和客户端上显示网络接口控制器(NIC)的最大网络速度：

```
# ethtool enp1s0 | grep "Speed"
  Speed: 10000Mb/s
```

2. 在服务器中：

   a. 显示最大 UDP 套接字读缓冲区大小，并记录值：

```
# sysctl net.core.rmem_max
net.core.rmem_max = 16777216
```

显示的值以字节为单位。

b.

在 firewalld 服务中临时打开默认的 iperf3 端口 5201 ：

```
# firewall-cmd --add-port=5201/tcp --add-port=5201/udp
# firewall-cmd --reload
```

请注意，iperf3 只打开服务器上的一个 TCP 套接字。如果客户端希望使用 UDP，它首先连接到此 TCP 端口，然后服务器在同一端口号上打开一个 UDP 套接字，以执行 UDP 流量吞吐量测试。因此，您必须在本地防火墙中为 TCP 和 UDP 协议打开端口 5201。

c.

在服务器模式下启动 iperf3 ：

```
# iperf3 --server
```

服务现在等待传入的客户端连接。

3.

在客户端中：

a.

显示客户端用于连接服务器的接口的最大传输单元(MTU)，并记录值：

```
# ip link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
state UP mode DEFAULT group default qlen 1000
...
```

b.

显示最大 UDP 套接字写缓冲区大小，并记录值：

```
# sysctl net.core.wmem_max
net.core.wmem_max = 16777216
```

显示的值以字节为单位。

c.

开始测量吞吐量：

```
# iperf3 --udp --time 60 --window 16777216 --length 1472 --bitrate 2G --client
192.0.2.1
```

- **--udp** ：使用 UDP 协议进行测试。
- **--time *<seconds>***：定义客户端停止传输时的时间（以秒为单位）。
- **--window *<size>*** ：设置 UDP 套接字缓冲区大小。理想情况下，客户端和服务器上的大小相同。如果不同，则将此参数设置为：比客户端上的net.core.wmem_max或服务器上的 net.core.rmem_max 更小的值。
- **--length *<size>*** ：设置要读和写的缓冲区的长度。将这个选项设置为最大的未碎片的有效负载。按如下所示计算理想的值：MTU - IP 标头(IPv4 为 20 字节，IPv6 为 40 字节) - 8 字节 UDP 标头。
- **--bitrate *<rate>*** ：将比特率限制为指定的值，单位是比特每秒。您可以指定单位，如 2G 代表 2 Gbps。

  将此参数设置为您希望正常工作的值，并在稍后的测量中增加它。如果服务器以比传输路径上的设备更快的速度发送数据包，或者客户端可以处理它们，则数据包可以被丢弃。
- **--client <server>** ：启用客户端模式并设置运行 iperf3 服务器的服务器的 IP 地址或名称。

4. 等待 iperf3 完成测试。服务器和客户端都显示每秒的统计信息，并在结尾显示总结。例如，以下是在客户端上显示的总结：

```
[ ID] Interval       Transfer     Bitrate         Jitter    Lost/Total Datagrams
[ 5] 0.00-60.00 sec  14.0 GBytes   2.00 Gbits/sec  0.000 ms  0/10190216 (0%) sender
[ 5] 0.00-60.04 sec  14.0 GBytes   2.00 Gbits/sec  0.002 ms  0/10190216 (0%) receiver
```

在这个示例中，平均比特率为 2 Gbps，没有丢失数据包。

5. 在服务器中：

a. 按 Ctrl+C 停止 iperf3 服务器。

b. 关闭 firewalld 中的端口 5201 ：

```
# firewall-cmd --remove-port=5201/tcp --remove-port=5201/udp
# firewall-cmd --reload
```

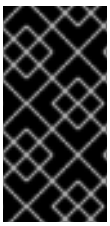
其他资源

- iperf3 (1) 手册页

31.6.3. MTU 大小对 UDP 流量吞吐量的影响

如果您的应用程序使用大型 UDP 消息大小，则使用巨型帧可以提高吞吐量。根据 IEEE 802.3 标准，没有 Virtual Local Area Network (VLAN)标签的默认以太网帧最大大小为 1518 字节。这些帧的每一个都包含一个 18 字节标头，为有效负载保留 1500 字节。因此，对于服务器通过网络传输的每 1500 字节数据，有 18 字节(1.2%)是开销。

巨型帧是非标准帧，其比 1500 字节的标准以太网有效负载有更大的最大传输单位(MTU)。例如，如果您使用最大允许的 9000 字节有效负载的 MTU 配置巨型帧，则每个帧的开销减少到 0.2%。



重要

传输路径上的所有网络设备和涉及的广播域都必须支持巨型帧，并使用相同的 MTU。由于传输路径上的 MTU 设置不一致，所以数据包碎片和重组会减少网络吞吐量。

不同的连接类型有一些 MTU 限制：

- Ethernet ：MTU 限制为 9000 字节。
- datagram 模式中的 IP over InfiniBand (IPoIB)：MTU 限制为比 InfiniBand MTU 少 4 字节。
- 内存网络通常支持更大的 MTU。详情请查看相应的文档。

31.6.4. CPU 速度对 UDP 流量吞吐量的影响

在批量传输中，UDP 协议比 TCP 效率要低得多，主要是因为 UDP 中缺少数据包聚合。默认情况下，不会启用 Generic Receive Offload (GRO)和 Transmit Segmentation Offload (TSO)功能。因此，对于高速链路上的批量传输，CPU 频率可能会限制 UDP 吞吐量。

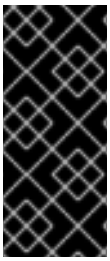
例如，在具有高最大传输单元(MTU)和大型套接字缓冲区的调优的主机上，3 GHz CPU 可以处理全速发送或接收 UDP 流量的 10 GBit NIC 的流量。但是，当您传输 UDP 流量时，您可以预计在 3 GHz 下，每 100 MHz CPU 速度会损失大约 1-2 Gbps 速度。另外，如果 3 GHz 的 CPU 速度可以达到接近 10 Gbps，则在 40 GBit NIC 上，相同的 CPU 将 UDP 流量限制为大约 20-25 Gbps。

31.6.5. 增加系统范围的 UDP 套接字缓冲区

套接字缓冲区临时存储内核已收到或应发送的数据：

- 读套接字缓冲区保存内核已收到但应用程序尚未读取的数据包。
- 写套接字缓冲区保存应用程序已写到缓冲区，但内核尚未将它们传给 IP 堆栈和网络驱动程序的数据包。

如果 UDP 数据包太大，且超过速度缓冲区大小或数据包以太快速度发送或接收，则内核会丢弃任何新传入的 UDP 数据包，直到数据从缓冲区删除为止。在这种情况下，增加套接字缓冲区可以防止数据包丢失。



重要

设置太大的缓冲区大小浪费内存。每个套接字可以设置为应用程序所请求的大小，内核会加倍这个值。例如，如果应用程序请求 256 KiB 套接字缓冲区大小，并打开 100 万个套接字，则对于潜在的套接字缓冲区空间，系统只需要 512 GB RAM (512 KiB x 100万)。

先决条件

- 您遇到了明显的 UDP 数据包丢弃率。

流程

1. 根据您的要求，创建 /etc/sysctl.d/10-udp-socket-buffers.conf 文件，并设置最大读或写缓冲区大小，或两个都设置：

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
```

指定值（以字节为单位）。本例中的值将缓冲区的最大大小设置为 16 MiB。这两个参数的默认值都为 212992 字节(208 KiB)。

2. 从 /etc/sysctl.d/10-udp-socket-buffers.conf 文件中载入设置：

```
# sysctl -p /etc/sysctl.d/10-udp-socket-buffers.conf
```

3. 将应用程序配置为使用更大的套接字缓冲大小。

net.core.rmem_max 和 net.core.wmem_max 参数定义应用程序中 setsockopt() 函数可以请求的最大缓冲区大小。请注意，如果您将应用程序配置为不使用 setsockopt() 函数，则内核将使用 rmem_default 和 wmem_default 参数的值。

详情请查看应用程序的编程语言的文档。如果您不是应用程序的开发人员，请联系开发人员。

4. 重启应用程序以使用新的 UDP 缓冲区大小。

验证

- 使用您在遇到数据包丢弃时使用的相同方法来监控数据包丢弃统计信息。

  如果数据包仍然以较低的速率发生，请进一步增加缓冲区大小。

其他资源

- 更改套接字缓冲区大小的含义是什么？
- UDP (7) 手册页

- socket (7) 手册页

31.7. 识别应用程序读套接字缓冲区瓶颈

如果 TCP 应用程序没有频繁清除读套接字缓冲区，则性能可能会会受到影响，数据包可能会丢失。Red Hat Enterprise Linux 提供了不同的工具来识别这些问题。

31.7.1. 识别接收缓冲区崩溃和修剪

当接收队列中的数据超过接收缓冲区大小时，TCP 堆栈会尝试通过从套接字缓冲区中删除不必要的元数据来释放一些空间。此步骤被称为崩溃。

如果崩溃无法为额外的流量释放足够的空间，则内核会修剪到达的新数据。这意味着内核从内存中删除数据，数据包丢失。

为避免崩溃和修剪操作，请监控 TCP 缓冲区是否在服务器上发生了崩溃和修剪，在这种情况下，调整 TCP 缓冲区。

流程

1. 使用 nstat 程序查询 TcpExtTCPRcvCollapsed 和 TcpExtRcvPruned 计数：

```
# nstat -az TcpExtTCPRcvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned          0        0.0
TcpExtTCPRcvCollapsed    612859   0.0
```

2. 等待一些时间，然后重新运行 nstat 命令：

```
# nstat -az TcpExtTCPRcvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned          0        0.0
TcpExtTCPRcvCollapsed    620358   0.0
```

3. 与第一次运行相比，如果计数的值增加了，则需要调整：

   - 如果应用程序使用 setsockopt (SO_RCVBUF) 调用，请考虑将其删除。使用这个调

用，应用程序仅使用调用中指定的接收缓冲区大小，并关闭套接字自动调整其大小的能力。

- 如果应用程序没有使用 setsockopt (SO_RCVBUF) 调用，请调整 TCP 读套接字缓冲区的默认值和最大值。

4. 显示接收积压队列(Recv-Q)：

```
# ss -nti
State   Recv-Q   Send-Q   Local Address:Port   Peer Address:Port   Process
ESTAB   0        0        192.0.2.1:443        192.0.2.125:41574
     :7,7 ... lastrcv:543 ...
ESTAB   78       0        192.0.2.1:443        192.0.2.56:42612
     :7,7 ... lastrcv:658 ...
ESTAB   88       0        192.0.2.1:443        192.0.2.97:40313
     :7,7 ... lastrcv:5764 ...
...
```

5. 多次运行 ss -nt 命令，每次运行之间等待几秒钟。

如果输出在 Recv-Q 列中只出现一次高值，则应用程序介于两个接收操作之间。但是，如果 Recv-Q 中的值在 lastrcv 持续增长期间保持不变，或者 Recv-Q 随着时间的推移不断增加，则以下几个问题可能是原因：

- 应用程序不会检查其套接字缓冲区是否足够。有关如何解决这个问题的详细信息，请联络应用程序厂商。

- 应用程序没有获得足够的 CPU 时间。要进一步调试这个问题：

  i. 显示应用程序运行在哪个 CPU 核上：

```
# ps -eo pid,tid,psr,pcpu,stat,wchan:20,comm
   PID    TID PSR %CPU STAT WCHAN               COMMAND
...
 44594  44594   5  0.0 Ss   do_select           httpd
 44595  44595   3  0.0 S    skb_wait_for_more_pa httpd
 44596  44596   5  0.0 Sl   pipe_read           httpd
 44597  44597   5  0.0 Sl   pipe_read           httpd
 44602  44602   5  0.0 Sl   pipe_read           httpd
...
```

PSR 列显示当前进程被分配给哪个 CPU 核。

ii. 识别在同一核上运行的其他进程，并考虑将它们分配到其他核。

其他资源

- 增加系统范围的 TCP 套接字缓冲区

31.8. 使用大量传入请求调整应用程序

如果您运行一个处理大量传入请求的应用程序，如 Web 服务器，则可能需要调整 Red Hat Enterprise Linux 以优化性能。

31.8.1. 调整 TCP 侦听积压，以处理大量 TCP 连接尝试

当应用程序以 LISTEN 状态打开 TCP 套接字时，内核会限制此套接字可以处理的接受的客户端连接的数量。如果客户端尝试建立比应用程序可以处理的连接更多的连接，则新的连接将丢失，或者内核向客户端发送 SYN cookie。

如果系统处于正常工作负载下，且来自合法客户端的连接导致内核发送 SYN cookie，请调优 Red Hat Enterprise Linux (RHEL)以避免它们。

先决条件

- RHEL 在 Systemd 日志中记录 possible SYN flooding on port *<ip_address>*:*<port_number>* 错误信息。
- 大量连接尝试来自有效源，不是由攻击造成的。

流程

1. 要验证是否需要调整，请显示受影响的端口的统计信息：

```
# ss -ntl '( sport = :443 )'
State    Recv-Q   Send-Q   Local Address:Port   Peer Address:Port  Process
LISTEN   650      500      192.0.2.1:443        0.0.0.0:*
```

如果积压 (Recv-Q)中的当前连接数大于套接字积压(Send-Q)，则侦听积压仍然不够大，需要调优。

2. 可选：显示当前 TCP 侦听积压限制：

```
# sysctl net.core.somaxconn
net.core.somaxconn = 4096
```

3. 创建 /etc/sysctl.d/10-socket-backlog-limit.conf 文件，并设置更大的侦听积压限制：

```
net.core.somaxconn = 8192
```

请注意，应用程序可以请求比 net.core.somaxconn 内核参数中指定的更大的侦听积压，但内核会将应用程序限制为您在此参数中设置的数值。

4. 从 /etc/sysctl.d/10-socket-backlog-limit.conf 文件载入设置：

```
# sysctl -p /etc/sysctl.d/10-socket-backlog-limit.conf
```

5. 重新配置应用程序，以使用新的侦听积压限制：

- 如果应用程序为限制提供了配置选项，请更新它。例如，Apache HTTP 服务器提供了 ListenBacklog 配置选项，来为该服务设置侦听积压限制。
- 如果无法配置限制，请重新编译应用程序。

6. 重新启动应用程序。

验证

1. 监控 Systemd 日志，以确认以后是否有possible SYN flooding on port *<port_number>* 错误消息出现。

2. 监控积压中当前连接的数量，并将其与套接字积压进行比较：

```
# ss -ntl '( sport = :443 )'
State    Recv-Q   Send-Q   Local Address:Port   Peer Address:Port  Process
LISTEN   0        500      192.0.2.1:443        0.0.0.0:*
```

如果当前积压 (Recv-Q)中的连接数大于套接字积压(Send-Q)，则侦听积压不够大，需要进一步调优。

其他资源

- 内核：端口上可能的 SYN 洪灾.发送 Cookie 解决方案
- 对于新的连接握手，侦听 TCP 服务器会忽略 SYN 或 ACK解决方案
- listen (2) 手册页

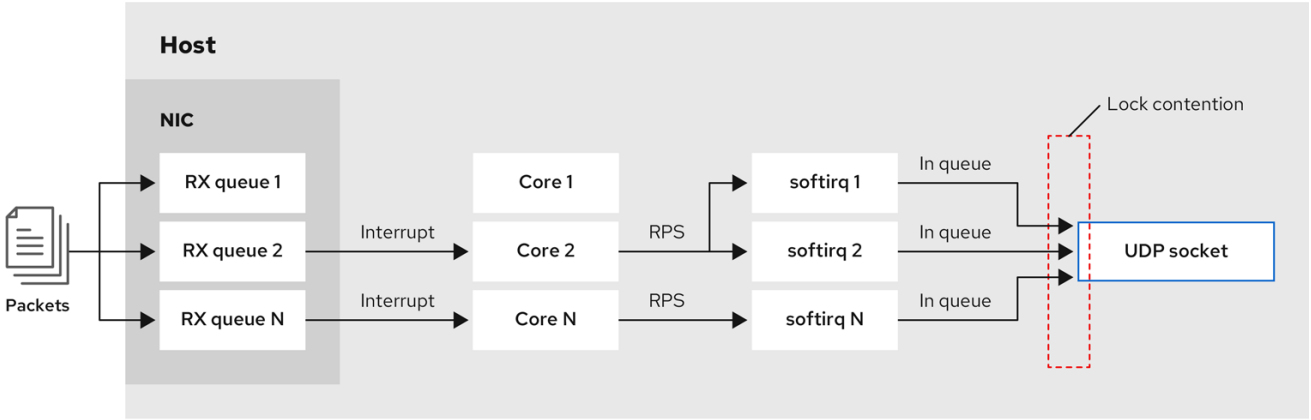
31.9. 避免侦听队列锁争用

队列锁争用可能会导致数据包丢弃和更高的 CPU 使用用率，因此导致更高的延迟。您可以通过调整应用程序并使用传输数据包转向，来避免接收(RX)和传输(TX)队列上的队列锁争用。

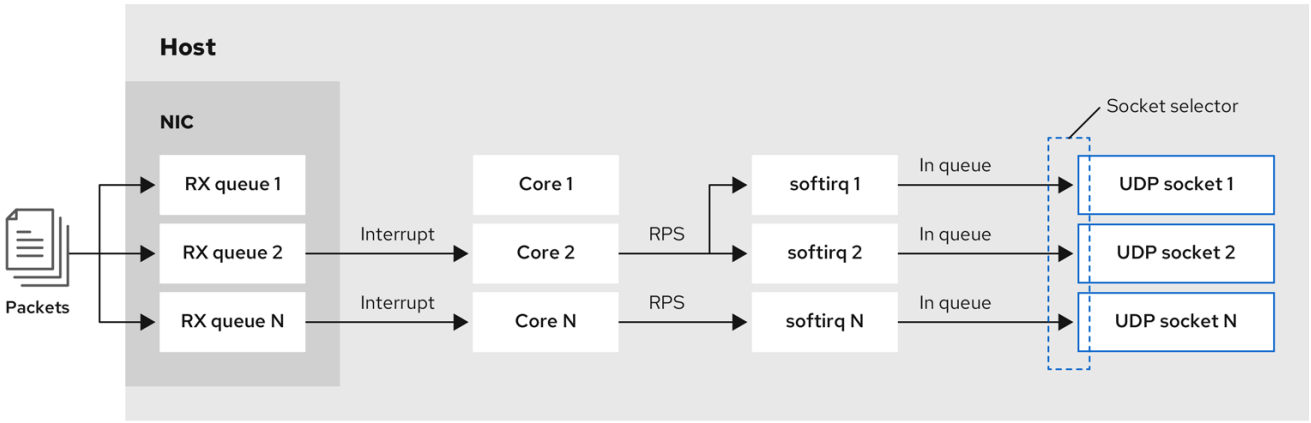
31.9.1. 避免 RX 队列锁争用：SO_REUSEPORT 和 SO_REUSEPORT_BPF 套接字选项

在多核系统上，如果应用程序使用 SO_REUSEPORT 或 SO_REUSEPORT_BPF 套接字选项打开端口，则您可以提高多线程网络服务器应用程序的性能。如果应用程序不使用其中一个套接字选项，则所有线程都会被强制共享一个套接字来接收传入流量。使用一个套接字会导致：

- 接收缓冲区上的重大竞争，这可能会导致数据包丢弃和更高的 CPU 使用率。
- CPU 使用率显著增加
- 可能的数据包丢弃



使用 SO_REUSEPORT 或 SO_REUSEPORT_BPF 套接字选项，一个主机上的多个套接字可以绑定到同一端口：



Red Hat Enterprise Linux 提供了一个如何在内核源中使用 SO_REUSEPORT 套接字选项的代码示例。要访问代码示例：

1. 启用 rhel-9-for-x86_64-baseos-debug-rpms 存储库：

```
# subscription-manager repos --enable rhel-9-for-x86_64-baseos-debug-rpms
```

2. 安装 kernel-debuginfo-common-x86_64 软件包：

```
# dnf install kernel-debuginfo-common-x86_64
```

3. 代码示例现在在 /usr/src/debug/kernel-*<version>*/linux-

<version>/tools/testing/selftests/net/reuseport_bpf_cpu.c 文件中。

其他资源

- socket (7) 手册页
- /usr/src/debug/kernel-*<version>*/linux-<version>/tools/testing/selftests/net/reuseport_bpf_cpu.c

31.9.2. 避免 TX 队列锁争用：传输数据包转向

在具有支持多个队列的网络接口控制器(NIC)的主机中，传输数据包转向(XPS)将传出网络数据包的处理分发到多个队列。这可让多个 CPU 处理传出的网络流量，并避免传输队列锁争用，因此避免了数据包丢弃。

某些驱动程序，如 ixgbe、i40e 和 mlx5 会自动配置 XPS。要识别驱动程序是否支持此功能，请参阅 NIC 驱动程序的文档。请参考您的 NIC 驱动程序文档来识别驱动程序是否支持此功能。如果驱动程序不支持 XPS 自动调整，您可以手动将 CPU 核分配给传输队列。

注意

Red Hat Enterprise Linux 没有提供一个将传输队列永久分配给 CPU 内核的选项。在脚本中使用命令，并在系统启动时运行它。

先决条件

- NIC 支持多个队列。
- numactl 软件包已安装。

流程

1. 显示可用队列的数量：

```
# ethtool -l enp1s0
Channel parameters for enp1s0:
Pre-set maximums:
RX:  0
```

```
TX:  0
Other:  0
Combined: 4
Current hardware settings:
RX:  0
TX:  0
Other:  0
Combined: 1
```

Pre-set maximums 部分显示队列的总数，Current hardware settings 显示当前分配给接收、传输、其他或组合队列的队列数。

2. 可选：如果您在特定通道上需要队列，请相应地分配它们。例如，要将 4 个队列分配给 Combined 通道，请输入：

```
# ethtool -L enp1s0 combined 4
```

3. 显示 NIC 被分配给了哪个 Non-Uniform Memory Access (NUMA)节点：

```
# cat /sys/class/net/enp1s0/device/numa_node
0
```

如果文件未找到，或者命令返回 -1，则主机不是 NUMA 系统。

4. 如果主机是 NUMA 系统，显示哪些 CPU 分配给了哪个 NUMA 节点：

```
# lscpu | grep NUMA
NUMA node(s):      2
NUMA node0 CPU(s):  0-3
NUMA node1 CPU(s):  4-7
```

5. 在上例中，NIC 有 4 个队列，NIC 被分配给了 NUMA 节点 0。此节点使用 CPU 核 0-3。因此，将每个传输队列映射到 CPU 核 0-3 中的一个：

```
# echo 1 > /sys/class/net/enp1s0/queues/tx-0/xps_cpus
# echo 2 > /sys/class/net/enp1s0/queues/tx-1/xps_cpus
# echo 4 > /sys/class/net/enp1s0/queues/tx-2/xps_cpus
# echo 8 > /sys/class/net/enp1s0/queues/tx-3/xps_cpus
```

如果 CPU 核数和传输(TX)队列相同，请使用 1 对 1 映射，以避免 TX 队列上任何类型的争用。否则，如果您在同一 TX 队列上映射了多个 CPU，则不同 CPU 上的传输操作将导致 TX 队列

锁争用，并对传输吞吐量造成负面影响。

请注意，您必须将包含 CPU 核号的位图传给队列。使用以下命令计算位图：

```
# printf %x $((1 << <core_number> ))
```

验证

1. 识别发送流量的服务的进程 ID (PID)：

```
# pidof <process_name>
12345 98765
```

2. 将 PID 固定到使用 XPS 的核：

```
# numactl -C 0-3 12345 98765
```

3. 在进程发送流量时监控 requeues 计数：

```
# tc -s qdisc
qdisc fq_codel 0: dev enp10s0u1 root refcnt 2 limit 10240p flows 1024 quantum 1514
target 5ms interval 100ms memory_limit 32Mb ecn drop_batch 64
 Sent 125728849 bytes 1067587 pkt (dropped 0, overlimits 0 requeues 30)
 backlog 0b 0p requeues 30
 ...
```

如果 requeues 计数不再以明显的速率增加，则 TX 队列锁争用不再发生。

其他资源

- /usr/share/doc/kernel-doc-_<version>/Documentation/networking/scaling.rst

31.9.3. 在具有高 UDP 流量的服务器上禁用 Generic Receive Offload 功能

使用高速度 UDP 批量传输的应用程序应该在 UDP 套接字上启用和使用 UDP Generic Receive Offload (GRO)。但是，如果满足以下条件，您可以禁用 GRO 来提高吞吐量：

- 应用程序不支持 GRO，此功能无法添加。
- 与 TCP 吞吐量无关。

警告

禁用 GRO 可显著减少 TCP 流量的接收吞吐量。因此，不要在与 TCP 性能相关的主机上禁用 GRO。

先决条件

- 主机主要处理 UDP 流量。
- 应用程序不使用 GRO。
- 主机不使用 UDP 隧道协议，如 VXLAN。
- 主机不运行虚拟机(VM)或容器。

流程

1. 可选：显示 NetworkManager 连接配置文件：

```
# nmcli connection show
NAME     UUID                                  TYPE      DEVICE
example  f2f33f29-bb5c-3a07-9069-be72eaec3ecf  ethernet  enp1s0
```

2. 在连接配置文件中禁用 GRO 支持：

```
# nmcli connection modify example ethtool.feature-gro off
```

3. 重新激活连接配置文件：

```
# nmcli connection up example
```

验证

1. 验证 GRO 是否已禁用：

```
# ethtool -k enp1s0 | grep generic-receive-offload
generic-receive-offload: off
```

2. 监控服务器上的吞吐量。如果设置对主机上的其他应用程序有负面影响，则在 NetworkManager 配置文件中重新启用 GRO。

31.10. 调整设备驱动程序和 NIC

在 RHEL 中，内核模块为网络接口控制器(NIC)提供驱动程序。这些模块支持调整和优化设备驱动程序和 NIC 的参数。例如，如果驱动程序支持延迟接收中断的产生，您可以减少相应参数的值，以避免接收描述符不足。

注意

并非所有模块都支持自定义参数，功能取决于硬件，以及驱动程序和固件版本。

31.10.1. 配置自定义 NIC 驱动程序参数

许多内核模块支持调优驱动程序和网络接口控制器(NIC)的设置参数。您可以根据硬件和驱动程序自定义设置。

重要

如果您对内核模块设置参数，则 RHEL 会将这些设置应用到所有使用这个驱动程序的设备。

先决条件

- 主机上已安装了 NIC。

- 为 NIC 提供驱动程序的内核模块支持所需的调优功能。
- 您在本地登录或使用的网络接口与您要为其更改参数的驱动程序的网络接口不同。

流程

1. 确定驱动程序：

```
# ethtool -i enp0s31f6
driver: e1000e
version: ...
firmware-version: ...
...
```

请注意，某些功能可能需要特定的驱动程序和固件版本。

2. 显示内核模块的可用参数：

```
# modinfo -p e1000e
...
SmartPowerDownEnable:Enable PHY smart power down (array of int)
parm:RxIntDelay:Receive Interrupt Delay (array of int)
```

有关参数的详情，请查看内核模块的文档。有关 RHEL 中的模块，请参阅 kernel-doc 软件包提供的 /usr/share/doc/kernel-doc-*<version>*/Documentation/networking/device_drivers/ 目录中的文档。

3. 创建 /etc/modprobe.d/*nic-parameters.conf* 文件，并为模块指定参数：

```
options <module_name> <parameter1>=<value> <parameter2>=<value>
```

例如，要启用端口节能机制，并将接收中断的产生设置为 4 个单元，请输入：

```
options e1000e SmartPowerDownEnable=1 RxIntDelay=4
```

4. 卸载模块：

```
# modprobe -r e1000e
```

警告

卸载活动网络接口使用的模块，会立即终止连接，您可以将自己锁定在服务器之外。

5. 载入模块：

```
# modprobe e1000e
```

6. 重新激活网络连接：

```
# nmcli connection up <profile_name>
```

验证

1. 显示内核信息：

```
# dmesg
...
[35309.225765] e1000e 0000:00:1f.6: Transmit Interrupt Delay set to 16
[35309.225769] e1000e 0000:00:1f.6: PHY Smart Power Down Enabled
...
```

请注意，并非所有模块都将参数设置记录到内核环缓冲区中。

2. 某些内核模块为 /sys/module/*<driver>*/parameters/ 目录中的每个模块参数创建文件。每个文件都包含此参数的当前值。您可以显示这些文件以验证设置：

```
# cat /sys/module/<driver_name>/parameters/<parameter_name>
```

31.11. 配置网络适配器卸载设置

为减少 CPU 负载，某些网络适配器使用卸载功能，其将网络绑定负载移到网络接口控制器(NIC)。例

如，使用 Encapsulating Security Payload(ESP)卸载时，NIC 会执行 ESP 操作，来加快 IPsec 连接，并减少 CPU 负载。

Red Hat Enterprise Linux 中的大多数卸载功能默认启用。只有在以下情况下禁用它们：

- 临时禁用卸载功能以进行故障排除。
- 当特定功能对您的主机造成负面影响时，永久禁用卸载功能。

如果在网络驱动程序中默认没有启用与性能相关的卸载功能，您可以手动启用它。

31.11.1. 临时设置卸载功能

如果您预计卸载功能会导致问题或降低主机的性能，您可以根据其当前状态，通过临时启用或禁用它来尝试缩小原因的范围。

如果您临时启用或禁用了卸载功能，它会在下次重启时返回到之前的值。

先决条件

- 网卡支持卸载功能。

流程

1. 显示接口的可用卸载功能及其当前状态：

```
# ethtool -k enp1s0
...
esp-hw-offload: on
ntuple-filters: off
rx-vlan-filter: off [fixed]
...
```

输出取决于硬件及其驱动程序的能力。请注意，您无法更改标记为 **[fixed]** 的功能的状态。

2. 临时禁用卸载功能：

```
# ethtool -K <interface> <feature> [on|off]
```

- 例如，要在 enp10s0u1 接口上临时禁用 IPsec Encapsulating Security Payload (ESP)卸载，请输入：

```
# ethtool -K enp10s0u1 esp-hw-offload off
```

- 例如，要在 enp10s0u1 接口上临时启用加速的 Receive Flow Steering (aRFS)过滤，请输入：

```
# ethtool -K enp10s0u1 ntuple-filters on
```

验证

1. 显示卸载功能的状态：

```
# ethtool -k enp1s0
...
esp-hw-offload: off
ntuple-filters: on
...
```

2. 在更改卸载功能前测试您遇到的问题是否仍然存在。

- 在更改特定的卸载功能后，如果问题不再存在：
  i. 联系红帽支持，并报告问题。
  ii. 考虑 永久设置卸载功能，直到有修复可用。
- 如果禁用特定卸载功能后问题仍然存在：
  i. 使用 ethtool -K *<interface> <feature> [on|off]* 命令将设置重置为之前的状态。

ii. 启用或禁用不同的卸载功能来缩小问题范围。

其他资源

- ethtool(8) 手册页

31.11.2. 永久设置卸载功能

如果您确定了在主机上限制性能的特定的卸载功能，您可以永久启用或禁用它，具体取决于其当前状态。

如果您永久启用或禁用卸载功能，NetworkManager 确保在重启后该功能仍然具有此状态。

先决条件

- 您确定了特定的卸载功能来限制主机上的性能。

流程

1. 识别使用您要在其上更改卸载功能状态的网络接口的连接配置文件：

```
# nmcli connection show
NAME     UUID                                  TYPE      DEVICE
Example  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1ss0
...
```

2. 永久更改卸载功能的状态：

```
# nmcli connection modify <connection_name> <feature> [on|off]
```

- 例如，要在 Example 连接配置集中永久禁用 IPsec Encapsulating Security Payload (ESP)卸载，请输入：

```
# nmcli connection modify Example ethtool.feature-esp-hw-offload off
```

- 例如，要在 Example 连接配置集中永久启用 accelerated Receive Flow Steering (aRFS)过滤，请输入：

```
# nmcli connection modify Example ethtool.feature-ntuple on
```

3. 重新激活连接配置文件：

```
# nmcli connection up Example
```

验证

- 显示卸载功能的输出状态：

```
# ethtool -k enp1s0
...
esp-hw-offload: off
ntuple-filters: on
...
```

其他资源

- nm-settings-nmcli(5) 手册页

31.12. 调整中断合并设置

中断合并是一种减少网卡产生的中断数量的机制。通常，较少的中断可能会提高网络的延迟和整体性能。

调整中断合并设置涉及调整控制的参数：

- 合并成一个中断的数据包的数量。
- 产生中断前的延迟。

重要

最佳合并设置取决于使用的特定网络条件和硬件。因此，可能需要进行一些尝试，来找到最适合您的环境和需要的设置。

31.12.1. 为延迟或吞吐量敏感服务优化 RHEL

合并调优的目标是最小化给定工作负载所需的中断的数量。在高吞吐量情况下，目标是尽可能有少的中断，同时保持高数据率。在低延迟情况下，可以使用更多的中断来快速处理流量。

您可以调整网卡上的设置，以增加或减少组合成一个中断的数据包数。因此，您可以提高流量的吞吐量或延迟。

流程

1. 识别遇到瓶颈的网络接口：

```
# ethtool -S enp1s0
NIC statistics:
    rx_packets: 1234
    tx_packets: 5678
    rx_bytes: 12345678
    tx_bytes: 87654321
    rx_errors: 0
    tx_errors: 0
    rx_missed: 0
    tx_dropped: 0
    coalesced_pkts: 0
    coalesced_events: 0
    coalesced_aborts: 0
```

识别名称中包含"drop"、"discard"或"error"的数据包数。这些特定统计测量网络接口卡(NIC)数据包缓冲区的实际数据包丢失，这可能是由 NIC 合并造成的。

2. 监控您在上一步中识别的数据包数的值。

将它们与网络的预期值进行比较，以确定任何特定接口是否遇到瓶颈。网络瓶颈的一些常见迹象包括：

- 网络接口上有很多错误
- 高数据包丢失
- 网络接口使用率过高

注意

在识别网络瓶颈时，其他重要因素是 CPU 使用率、内存使用率和磁盘 I/O。

3. 查看当前的合并设置：

```
# ethtool enp1s0
Settings for enp1s0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Supports Wake-on: g
        Wake-on: g
        Current message level: 0x00000033 (51)
                               drv probe link
        Link detected: yes
```

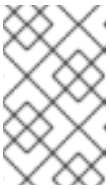
在此输出中，监控 **Speed** 和 **Duplex** 字段。这些字段显示有关网络接口操作的信息，以及它是否以预期的值运行。

4. 检查当前的中断合并设置：

```
# ethtool -c enp1s0
Coalesce parameters for enp1s0:
        Adaptive RX: off
        Adaptive TX: off
        RX usecs: 100
        RX frames: 8
        RX usecs irq: 100
        RX frames irq: 8
        TX usecs: 100
```

```
TX frames: 8
TX usecs irq: 100
TX frames irq: 8
```

- usecs 值指的是接收方或传输方在产生中断前等待的微秒数。
- frames 值指的是接收方或传输方在产生中断前等待的帧数。
- irq 值用于在网络接口已经处理中断时配置中断调节。



注意

不是所有网络接口卡都支持报告和更改示例输出中的所有值。

- Adaptive RX/TX 值表示自适应中断合并机制，它动态调整中断合并设置。根据数据包条件，当启用 Adaptive RX/TX 启用时，NIC 驱动程序会自动计算合并值（每个 NIC 驱动程序的算法不同）。

5. 根据需要修改合并设置。例如：

- 在 ethtool.coalesce-adaptive-rx 被禁用时，请配置 ethtool.coalesce-rx-usecs，来将 RX 数据包的产生中断前的延迟配置为 100 微秒：

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-usecs 100
```

- 启用 ethtool.coalesce-adaptive-rx，同时 ethtool.coalesce-rx-usecs 设为其默认值：

```
# nmcli connection modify enp1s0 ethtool.coalesce-adaptive-rx on
```

修改 Adaptive-RX 设置，如下所示：

  - 关注低延迟(低于 50us)的用户不应启用 Adaptive-RX。
  -

关注吞吐量的用户可能启用 Adaptive-RX，而没有伤害。如果他们不希望使用自适应中断合并机制，他们可以尝试对 ethtool.coalesce-rx-usecs 设置大值，如 100us 或 250us 。

- 在出现问题前，不清楚自己需求的用户不应修改此设置。

6. 重新激活连接：

```
# nmcli connection up enp1s0
```

验证步骤

- 监控网络性能并检查丢弃的数据包：

```
# ethtool -S enp1s0
NIC statistics:
    rx_packets: 1234
    tx_packets: 5678
    rx_bytes: 12345678
    tx_bytes: 87654321
    rx_errors: 0
    tx_errors: 0
    rx_missed: 0
    tx_dropped: 0
    coalesced_pkts: 12
    coalesced_events: 34
    coalesced_aborts: 56
...
```

rx_errors、rx_dropped、tx_errors 和 tx_dropped 字段的值应该为 0 或接近 0（最多几百，取决于网络流量和系统资源）。这些字段中的高值表示网络问题。您的计数可以有不同的名称。仔细监控其名称中包含"drop"、"discard"或"error"的数据包计数。

rx_packets、tx_packets、rx_bytes 和 tx_bytes 的值应该随着时间而增加。如果值没有增加，则可能会有网络问题。数据包计数可以有不同的名称，具体取决于您的 NIC 驱动程序。

重要

ethtool 命令输出可能会因使用的 NIC 和驱动程序而异。

专注于非常低延迟的用户出于监控目的，可以使用应用程序级指标或内核数据包时间戳 API 。

其他资源

- 对任何性能问题的初始调查
- 用于网络调优的内核参数是什么？
- 如何使 NIC ethtool 设置持久（在引导时自动应用）
- 时间戳

31.13. TCP 时间戳的好处

TCP 时间戳在 TCP 标头中是可选的信息，是 TCP 协议的扩展。默认情况下，TCP 时间戳在 Red Hat Enterprise Linux 中启用，内核使用 TCP 时间戳来更好地估算 TCP 连接中的往返时间(RTT)。这会导致更准确的 TCP 窗口和缓冲区计算。

另外，TCP 时间戳提供了一种替代方法来确定片段的年龄和顺序，并防止包装序列号。TCP 数据包标头在 32 位字段中记录序列号。在 10 Gbps 连接中，此字段的值可以在 1.7 秒后结束。如果没有 TCP 时间戳，接收方无法决定带有包装的序列号的段是一个新段还是一个旧的复制段。但是，使用 TCP 时间戳，接收方可以做出正确的选择来接收或丢弃段。因此，在带有快速网络接口的系统上启用 TCP 时间戳是非常重要的。

net.ipv4.tcp_timestamps 内核参数可以有以下值之一：

- 0 ：TCP 时间戳被禁用。
- 1： TCP 时间戳被启用（默认）。
- 2： TCP 时间戳被启用，但没有随机偏移量。

重要

如果每个连接没有随机偏移量，就可以大概确定主机的运行时间和指纹，并在攻击中使用此信息。

默认情况下，TCP 时间戳在 Red Hat Enterprise Linux 中启用，并为每个连接使用随机偏移量，而不是只存储当前时间：

```
# sysctl net.ipv4.tcp_timestamps
net.ipv4.tcp_timestamps = 1
```

如果 net.ipv4.tcp_timestamps 参数有一个与默认值不同的值(1)，请用与您设置它的相同的方式恢复设置。

其他资源

- RFC 1323：高性能的 TCP 扩展

31.14. 以太网网络的流控制

在以太网链路上，网络接口和交换机端口之间的持续数据传输可能会导致缓冲区满了。缓冲区满了会导致网络阻塞。在这种情况下，当发送方以高于接收方处理能力的速度传输数据时，可能会因为链接的另一端（其是一个交换机端口）上网络接口的数据处理能力较低而发生数据包丢失。

流控制机制管理以太网链路之间的数据传输，其中每个发送方和接收方都有不同的发送和接收能力。为避免数据包丢失，以太网流控制机制会临时暂停数据包传输，以便从交换机端口管理更高的传输率。请注意，路由器不会在交换机端口外转发暂停的帧。

当接收(RX)缓冲区已满时，接收方会向传输方发送暂停帧。然后，传输方会在较短的亚秒时间帧内停止数据传输，同时继续在此暂停期间缓冲传入的数据。此持续时间为接收方清空其接口缓冲区提供了足够的时间，并防止缓冲区溢出。

注意

以太网链接的任一端都可以向另一端发送暂停帧。如果网络接口的接收缓冲区已满，则网络接口将向交换机端口发送暂停帧。同样，当交换机端口的接收缓冲区已满时，交换机端口会向网络接口发送暂停帧。

默认情况下，Red Hat Enterprise Linux 中大多数网络驱动程序都启用了暂停帧支持。要显示网络接口的当前设置，请输入：

```
# ethtool --show-pause enp1s0
Pause parameters for enp1s0:
...
RX:     on
TX:     on
...
```

与您的交换机厂商进行验证，以确认您的交换机是否支持暂停帧。

其他资源

- ethtool(8) 手册页
- 什么是网络链路控制以及它如何在 Red Hat Enterprise Linux 中工作？

第 32 章 影响 I/O 和文件系统性能的因素

存储和文件系统性能的适当设置高度依赖于存储目的。

I/O 和文件系统性能可能会受到以下任意因素的影响：

- 数据写入或读取特征
- 顺序或随机
- buffered 或 Direct IO
- 数据与底层 geometry 保持一致
- 块大小
- 文件系统大小
- 日志大小和位置
- 记录访问时间
- 确保数据可靠性
- 预抓取数据
- 预分配磁盘空间

- 文件碎片
- 资源争用

32.1. 监控和诊断 I/O 和文件系统问题的工具

以下工具包括在 Red Hat Enterprise Linux 9 中用于监控系统性能并诊断与 I/O、文件系统及其配置相关的性能问题：

- vmstat 工具报告整个系统的进程、内存、分页、块 I/O、中断和 CPU 活动。它可帮助管理员确定 I/O 子系统是否负责任何性能问题。如果使用 vmstat 进行分析显示，I/O 子系统负责降低性能，管理员可以使用 iostat 工具来确定负责的 I/O 设备。
- iostat 报告您系统中的 I/O 设备负载。它由 sysstat 软件包提供。
- blktrace 提供有关 I/O 子系统花费时间的详细信息。比较实用程序 blkparse 读取来自 blktrace 的原始输出，并生成由 blktrace 记录的输入和输出操作的人类可读摘要。
- btt 分析 blktrace 输出并显示 I/O 堆栈各个区域所花费的时间，从而更轻松地发现 I/O 子系统中的瓶颈。该实用程序作为 blktrace 软件包的一部分提供。由 blktrace 机制跟踪并由 btt 分析的一些重要事件有：
  - I/O 事件队列 (Q)
  - I/O 的发送至驱动程序事件 (D)
  - 完成 I/O 事件 (C)
- iowatcher 可以使用 blktrace 输出来随着时间的推移图形 I/O。它关注磁盘 I/O 的逻辑块地址 (LBA) 、每秒吞吐量每秒的吞吐量、每秒寻道数数、I/O 操作数。这有助于识别何时达到设备的操作每秒限值。
- BPF Compiler Collection（BCC）是一个库，可帮助创建扩展的 Berkeley Packet

Filter（eBPF）程序。eBPF 程序在事件中触发，如磁盘 I/O、TCP 连接和进程创建。BCC 工具安装在 /usr/share/bcc/tools/ 目录中。以下 bcc-tools 可帮助分析性能：

- biolatency 总结了块设备 I/O（磁盘 I/O）中延迟的问题。这允许研究发行版，包括用于设备缓存命中以及缓存未命中的两种模式，以及延迟延迟。
- biosnoop 是基本的块 I/O 追踪工具，用于显示每个 I/O 事件以及发出的进程 ID，以及 I/O 延迟。使用这个工具，您可以调查磁盘 I/O 性能问题。
- biotop 用于内核中的块 i/o 操作。
- filelife 工具跟踪 stat() 系统调用。
- fileslower 跟踪文件同步的读写速度比较慢。
- filetop 按进程显示文件读取和写入。
- ext4slower、nfsslower 和 xfsslower 是显示文件系统操作比特定阈值慢的工具，默认值为 10ms。

  如需更多信息，请参阅使用 BPF Compiler Collection 分析系统性能。

- bpftace 是用于分析性能问题的 eBPF 追踪语言。它还提供类似 BCC 的 trace 工具，用于调查 I/O 性能问题。
- 以下 SystemTap 脚本在诊断存储或文件系统性能问题可能很有用：
  - disktop.stp ：每 5 秒检查一次读取或写入磁盘的状态，并在该期间输出前 10 个条目。
  - iotime.stp: 显示读和写操作上花费的时间长度，以及读取和写入的字节数。

- traceio.stp ：根据所观察到的累积 I/O 流量打印前十大可执行文件。
- traceio2.stp: 将可执行名称和进程标识符作为读写操作发生。
- Inodewatch.stp ：每次对指定主设备或次要设备中的指定内节点进行读或写时，均会打印可执行文件名称和进程标识符。
- inodewatch2.stp ：每次在指定主设备或次要设备上更改属性时，均会打印可执行文件名称、进程标识符和属性。

其他资源

- vmstat (8) 、 iostat (1) 、 blktrace (8) 、 blkparse (1) 、 btt (1) 、 bpftrace 和 iowatcher (1) man page
- 使用 BPF Compiler Collection 分析系统性能

32.2. 用于格式化文件系统的可用调整选项

在格式化设备后，无法更改某些文件系统配置决策。

以下是在格式化存储设备前可用的选项：

大小

为您的工作负载创建一个适当化的文件系统。较小的文件系统需要较少的时间和内存来进行文件系统检查。但是，如果文件系统太小，其性能将会受到影响。

块大小

块是文件系统的工作单元。块大小决定了一个块中可以存储的数据量，因此一次写入或读取的最小数据量。

默认块大小适用于大多数用例。但是，如果块大小或多个块的大小与通常一次读取或写入的数据量相比，您的文件系统可以更有效地执行并存储数据。小文件仍使用整个块。文件可分布到多个块中，但这样可创建额外的运行时开销。

另外，某些文件系统仅限于特定数量块，这限制了文件系统的最大大小。使用 mkfs 命令格式化设备时，将块大小指定为文件系统选项的一部分。指定块大小的参数因文件系统而异。

Geometry

文件系统 geometry 关心文件系统中分布数据。如果您的系统使用条状存储（如 RAID），您可以在格式化该设备时将数据和元数据与底层存储 geometry 保持一致来提高性能。

许多设备导出建议的 geometry，然后在设备使用特定文件系统格式化时自动设置。如果您的设备没有导出这些建议，或者想要更改推荐的设置，您必须在使用 mkfs 命令格式化该设备时手动指定 geometry。

指定文件系统 geometry 的参数因文件系统而异。

外部日志

日志文件系统记录了在执行操作前在日志文件写入操作前的更改。这降低了存储设备在系统崩溃或电源故障时损坏的可能性，并加快恢复过程。

注意

红帽不推荐使用外部日志选项。

元数据密集型工作负载需要对日志进行非常频繁的更新。较大的日志会使用更多内存，但减少了写操作的频率。另外，您可以将一个元数据密集型工作负载的设备的查找时间放在专用存储上，或者比主存储快得多。

警告

确保外部日志可靠。丢失外部日志设备会导致文件系统崩溃。外部日志必须在格式化时创建，日志设备在挂载时指定。

其他资源

- mkfs (8) 和 mount (8) man page
- 可用文件系统概述

32.3. 可用于挂载文件系统的选项

以下是大多数文件系统可用的选项，并可以指定为挂载该设备：

访问时间

每次读取文件时，每次读取文件时，其元数据都会使用哪个时间更新（atime）。这包括额外的写入 I/O。relatime 是大多数文件系统的默认 atime 设置。

但是，如果更新此元数据会消耗大量时间，如果不需要准确的访问时间数据，您可以使用 noatime 挂载选项挂载文件系统。当文件读取时，该操作将禁用对元数据的更新。它还启用 nodiratime 行为，它会在目录读取时禁用对元数据的更新。

注意

使用 noatime mount 选项禁用 atime 更新可能会破坏依赖于它们的应用程序，例如备份程序。

Read-ahead

Read-ahead 行为通过预先获取可能需要的数据加快文件访问速度，并将其加载到页面缓存中，如果其位于磁盘上，可以比它更快获得文件。read-ahead 值越大，代表系统预抓取数据更进一步。

Red Hat Enterprise Linux 尝试根据您检测到的文件系统的内容设置适当的 read-ahead 值。但是，并不总是可以使用准确的检测。例如，如果存储阵列以单个 LUN 形式为系统提供自己，系统检测到单个 LUN，则不会为数组设置适当的 read-ahead 值。

涉及大量后续 I/O 的工作负载通常受益于高预读值。与使用 LVM 条带化一样，Red Hat Enterprise Linux 提供的与存储相关的 tuned 配置集会提高预读值，但这些调整并不总是足以满足所有工作负载。

其他资源

- mount (8) , xfs (5) , 和 ext4 (5) man pages

32.4. 丢弃未使用块的类型

定期丢弃文件系统没有使用的块是固态磁盘和精简置备存储的建议方法。

以下是丢弃未使用块的两种方法：

批量丢弃

这种丢弃是 fstrim 命令的一部分。它丢弃与管理员指定条件匹配的文件系统中所有未使用的块。Red Hat Enterprise Linux 9 支持在支持物理丢弃操作的 XFS 和 ext4 格式的设备中批量丢弃。

在线丢弃

这种类型的丢弃操作是在挂载时通过 discard 选项进行配置，并在用户不干预的情况下实时运行。但是，它只丢弃从已使用到空闲的块。Red Hat Enterprise Linux 9 支持 XFS 和 ext4 格式化的设备的在线丢弃。

红帽建议批量丢弃，除了需要在线丢弃才能保持性能，或者在系统工作负载中不可行批量丢弃。

预分配将磁盘空间标记为分配给文件，而不将数据写入那个空间。这可用于限制数据碎片和读性性能。Red Hat Enterprise Linux 9 支持在 XFS、ext4 和 GFS2 文件系统中预先分配空间。应用程序也可以通过使用 fallocate (2) glibc 调用从预先分配空间中受益。

其他资源

- mount (8) 和 fallocate (2) man page

32.5. 固态磁盘调优注意事项

固态磁盘 (SSD) 使用 NAND 闪存芯片而不是轮转磁盘来存储持久数据。SSD 为它们的完整逻辑块范围提供数据的持续访问时间，而且不会像轮转相对应的项一样实现可测量的查找成本。它们每 GB 存储空间的成本更大，且存储密度较低，但其延迟比 HDD 更高的吞吐量。

通常，性能会降级为 SSD 上使用的块，对磁盘的容量造成影响。降级程度因供应商而异，但所有设备在这种情形中都降级。启用丢弃行为有助于减少此降级。如需更多信息，请参阅丢弃未使用块的类型。

默认的 I/O 调度程序和虚拟内存选项适合与 SSD 一起使用。在配置可影响 SSD 性能的设置时，请考虑以下因素：

I/O 调度程序

任何 I/O 调度程序都应该能与大多数 SSD 一起正常工作。但是，与任何其他存储类型一样，红帽建议采用基准测试来确定给定工作负载的最佳配置。在使用 SSD 时，红帽建议仅针对特定工作负载更改 I/O 调度程序。有关如何在 I/O 调度程序间切换的步骤，请参阅 /usr/share/doc/kernel-version/Documentation/block/switching-sched.txt 文件。

对于单队列 HBA，默认的 I/O 调度程序为 deadline。对于多队列 HBA，默认的 I/O 调度程序为 none。有关如何设置 I/O 调度程序的详情，请参考 设置磁盘调度程序。

虚拟内存

与 I/O 调度程序一样，虚拟内存 (VM) 子系统不需要特殊的调优。由于 SSD 上的 I/O 的快速性质，尝试关闭 vm_dirty_background_ratio 和 vm_dirty_ratio 设置，因为增加了 write-out 活动通常对磁盘其他操作的延迟造成负面影响。但是，这个调整可以生成更多整体 I/O，因此通常不建议在不进行特定工作负载测试的情况下使用。

交换

SSD 也可以用作交换设备，并且可能会生成良好的页面和页面性能。

32.6. 通用块设备性能优化参数

此处列出的通用调优参数位于 /sys/block/sdX/queue/ 目录中。

以下列表的调优参数与 I/O 调度程序调整不同，适用于所有 I/O 调度程序：

add_random

有些 I/O 事件贡献到 /dev/random 的熵池。如果这些贡献的开销成为可衡量，则此参数可设为 0。

iostats

默认情况下，iostats 被启用，默认值为 1。将 iostats 值设置为 0 可禁用为设备收集 I/O 统计信息，从而消除 I/O 路径的少量开销。将 iostats 设置为 0 可能会稍提高高性能设备的性能，比如某些 NVMe 固态存储设备。建议启用 iostats，除非供应商为给定的存储模型指定。

如果您禁用 iostats，则 /proc/diskstats 文件中不再存在该设备的 I/O 统计信息。/sys/diskstats 文件的内容是监控 I/O 工具的 I/O 信息的来源，如 sar 或 iostats。因此，如果您为设备禁用 iostats

参数，则 I/O 监控工具输出中不再存在该设备。

max_sectors_kb

以 KB 为单位指定 I/O 请求的最大大小。默认值为 512 KB。此参数的最小值由存储设备的逻辑块大小决定。此参数的最大值由 max_hw_sectors_kb 的值决定。

红帽建议 max_sectors_kb 始终是最佳 I/O 大小和内部清除块大小的倍数。如果一个参数为零或者未由存储设备指定，则使用 logical_block_size 值。

nomerges

大多数工作负载都受益于请求合并。但是，禁用合并对调试很有用。默认情况下，nomerges 参数设置为 0，它启用合并。要禁用简单的单校合并，nomerges 被设置为 1。要禁用所有类型的合并，将 nomerges 设置为 2。

nr_requests

它是队列的 I/O 的最大允许数。如果当前 I/O 调度程序是 none，则此数值只能减少；否则，该数值可以被增加或减少。

optimal_io_size

有些存储设备通过这个参数报告最佳 I/O 大小。如果报告这个值，红帽建议应用程序问题 I/O，并尽可能使用最佳 I/O 大小的多个问题。

read_ahead_kb

定义在后续读操作期间操作系统可提前读取的最大 KB 数。因此，在内核页面缓存中已存在所需信息，用于改进 I/O 性能。

设备映射程序通常受益于高 read_ahead_kb 值。每个要映射的设备为 128 KB 是一个很好的起点，但将 read_ahead_kb 的值增加到磁盘的请求队列的 max_sectors_kb 值可在大量文件按顺序读取的应用程序环境中提高性能。

rotational

有些固态磁盘无法正确公告其固态状态，并被挂载为传统的旋转磁盘。手动将 rotational 值设置为 0 以禁用调度程序中的不必要的搜索逻辑。

rq_affinity

rq_affinity 的默认值为 1。它会在一个 CPU 内核中完成 I/O 操作，它位于发布的 CPU 内核的同一 CPU 组中。要只对发出 I/O 请求的处理器执行完成，请将 rq_affinity 设置为 2。要禁用上述两个功能，请将它设置为 0。

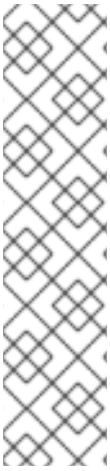
scheduler

要为特定存储设备设置调度程序或调度程序首选项顺序，请编辑 /sys/block/*devname*/queue/scheduler 文件，其中 *devname* 是您要配置的设备的名称。

第 33 章 使用 SYSTEMD 管理应用程序使用的资源

RHEL 9 通过将 cgroup 层次结构的系统与 systemd 单元树绑定，将资源管理设置从进程级别移到应用程序级别。因此，您可以使用 systemctl 命令或通过修改 systemd 单元文件来管理系统资源。

要做到这一点，systemd 从单元文件或者直接通过 systemctl 命令获取各种配置选项。然后，systemd 使用 Linux 内核系统调用及 cgroups和namespaces 这样的功能将这些选项应用到特定的进程组中。



注意

您可以在以下手册页中查看 systemd 的完整配置选项：

- systemd.resource-control (5)
- systemd.exec (5)

33.1. 资源管理中的 SYSTEMD 角色

systemd 的核心功能是服务管理和监管。systemd 系统和服务管理器：

- 确保受管服务在正确时间启动，并在启动过程中按正确的顺序启动。
- 确保受管服务平稳运行，以最优地使用底层硬件平台。
- 提供定义资源管理策略的能力。
- 提供调整各种选项的能力，这可以提高服务的性能。

重要

通常，红帽建议您使用 systemd 来控制系统资源的使用。您应该只在特殊情况下手动配置 cgroups 虚拟文件系统。例如，当您需要使用在 cgroup-v2 层次结构中没有对应的 cgroup-v1 控制器时。

33.2. 系统源的分发模型

要修改系统资源的发布，您可以应用一个或多个以下分发模型：

Weights（权重）

您可以通过增加所有子组的权重并为每个子组群分配资源，使其与总和总的比例匹配。

例如，如果您有 10 个 cgroups，则每个权重值为 100，sum 为 1000。每个 cgroup 会收到十分之一的资源。

权重通常用于分发无状态资源。例如， *CPUWeight=* 选项是此资源分布模型的实现。

Limits

cgroup 可以最多消耗配置的资源量。子组限值总和不能超过父 cgroup 的限值。因此，可以过量使用此模型中的资源。

例如， *MemoryMax=* 选项是此资源分发模型的实现。

Protections（保护）

您可以为 cgroup 设置受保护的资源量。如果资源使用量低于保护边界，内核将尝试不以竞争同一资源的 cgroup 替代其他 cgroup。可以过量使用。

例如， *MemoryLow=* 选项是此资源分发模型的实现。

Allocations（分配）

独占分配有限资源的绝对数量。不能过量使用。Linux 中这种资源类型的一个示例就是实时预算。

单元文件选项

资源控制配置的设置。

例如，您可以使用 *CPUAccounting=* 或 *CPUQuota=* 等选项配置 CPU 资源。同样，您可以使用 *AllowedMemoryNodes=* 和 *IOAccounting=* 等选项配置内存或 I/O 资源。

33.3. 使用 SYSTEMD 分配系统资源

流程

要更改服务的单元文件选项所需的值，您可以调整单元文件中的值，或使用 systemctl 命令：

1. 检查您选择的服务的分配值。

```
# systemctl show --property <unit file option> <service name>
```

2. 设置 CPU 时间分配策略选项的必要值：

```
# systemctl set-property <service name> <unit file option>=<value>
```

验证步骤

- 检查您选择的服务新分配的值。

```
# systemctl show --property <unit file option> <service name>
```

其他资源

- systemd.resource-control (5) 和 systemd.exec (5) 手册页

33.4. CGROUPS 的 SYSTEMD 层次结构概述

在后端，systemd 系统和服务管理器使用 slice、scope，以及 service 单元来整理和构建控制组中的进程。您可以通过创建自定义单元文件或使用 systemctl 命令来进一步修改此层次结构。另外，systemd 会在 /sys/fs/cgroup/ 目录中自动挂载重要内核资源控制器的层次结构。

对于资源控制，您可以使用以下三种 systemd 单元类型：

Service

systemd 根据单元配置文件启动的一个进程或一组进程，。

服务封装指定的进程，以便它们可以作为一个集启动和停止。服务使用以下方法命名：

```
<name>.service
```

影响范围

外部创建的一组进程。范围封装通过 fork() 函数由任意进程启动和停止的进程，然后在运行时由 systemd 注册。例如，用户会话、容器和虚拟机被视为范围。范围命名如下：

```
<name>.scope
```

slice

一组分级组织的单元。片段组织了一个分级，其中放置范围和服务。

实际的进程包含在范围或服务中。slice 单元的每个名称对应层次结构中的位置的路径。

短划线(-)字符充当路径组件与 -.slice root 片段中片段的分隔符。在以下示例中：

```
<parent-name>.slice
```

parent-name.slice 是 parent.slice 的子分片，它是 -.slice root 片段的子分片。parent-name.slice 可以有自己的子slice 名为 parent-name-name2.slice，以此类推。

service、scope 和 slice 单元直接映射到控制组层次结构中的对象。激活这些单元后，它们直接映射到从单元名称构建的控制组路径。

以下是控制组群分级的缩写示例：

```
Control group /:
-.slice
```

```
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 967 gdm-session-worker [pam/gdm-launch-environment]
│ │ │ ├─1035 /usr/libexec/gdm-x-session gnome-session --autostart
/usr/share/gdm/greeter/autostart
│ │ │ ├─1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -
background none -noreset -keeptty -verbose 3
│ │ │ ├─1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
│ │ │ ├─1369 /usr/bin/gnome-shell
│ │ │ ├─1732 ibus-daemon --xim --panel disable
│ │ │ ├─1752 /usr/libexec/ibus-dconf
│ │ │ ├─1762 /usr/libexec/ibus-x11 --kill-daemon
│ │ │ ├─1912 /usr/libexec/gsd-xsettings
│ │ │ ├─1917 /usr/libexec/gsd-a11y-settings
│ │ │ ├─1920 /usr/libexec/gsd-clipboard
…
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
  ├─rngd.service
  │ └─800 /sbin/rngd -f
  ├─systemd-udevd.service
  │ └─659 /usr/lib/systemd/systemd-udevd
  ├─chronyd.service
  │ └─823 /usr/sbin/chronyd
  ├─auditd.service
  │ ├─761 /sbin/auditd
  │ └─763 /usr/sbin/sedispatch
  ├─accounts-daemon.service
  │ └─876 /usr/libexec/accounts-daemon
  ├─example.service
  │ ├─ 929 /bin/bash /home/jdoe/example.sh
  │ └─4902 sleep 1
  …
```

上面的例子显示，服务和范围包含进程，并放置在不含自己进程的片段中。

其他资源

- 在 Red Hat Enterprise Linux 中*使用 systemctl 管理系统服务*
- 什么是内核资源控制器
- systemd.resource-control (5), systemd.exec (5), cgroups (7), fork （）, fork (2) 手册页

- 了解 cgroups

33.5. 列出 SYSTEMD 单元

使用 systemd 系统和服务管理器列出其单元。

流程

- 使用 systemctl 工具列出系统上所有活动的单元。终端返回一个类似于以下示例的输出：

```
# systemctl
UNIT                                LOAD   ACTIVE SUB       DESCRIPTION
…
init.scope                           loaded active running   System and Service Manager
session-2.scope                      loaded active running   Session 2 of user jdoe
abrt-ccpp.service                    loaded active exited    Install ABRT coredump
hook
abrt-oops.service                    loaded active running   ABRT kernel log watcher
abrt-vmcore.service                   loaded active exited    Harvest vmcores for
ABRT
abrt-xorg.service                    loaded active running   ABRT Xorg log watcher
…
-.slice                             loaded active active    Root Slice
machine.slice                        loaded active active    Virtual Machine and
Container Slice system-getty.slice                                    loaded
active active    system-getty.slice
system-lvm2\x2dpvscan.slice                  loaded active active    system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice                  loaded active active    system-
sshd\x2dkeygen.slice
system-systemd\x2dhibernate\x2dresume.slice       loaded active active    system-
systemd\x2dhibernate\x2dresume>
system-user\x2druntime\x2ddir.slice             loaded active active    system-
user\x2druntime\x2ddir.slice
system.slice                         loaded active active    System Slice
user-1000.slice                       loaded active active    User Slice of UID 1000
user-42.slice                        loaded active active    User Slice of UID 42
user.slice                           loaded active active    User and Session Slice
…
```

UNIT

还反映控制组层次结构中单元位置的单元名称。与资源控制相关的单元是 *slice*、*scope* 和 *service*

LOAD

指示单元配置文件是否被正确加载。如果单元文件加载失败，该字段包含状态 *error* 而

不是 *loaded*。其他单元负载状态为： *stub* 、 *merge* 和 *masked*。

ACTIVE

高级单元激活状态，其是 SUB 的一个泛论。

SUB

低级单元激活状态。可能的值的范围取决于单元类型。

DESCRIPTION

单元内容和功能的描述。

- 列出所有活跃的和不活跃的单元：

```
# systemctl --all
```

- 限制输出中的信息量：

```
# systemctl --type service,masked
```

--type 选项需要一个以逗号分隔的单元类型列表，如 *service* 和 *slice*，或者单元载入状态，如 *loaded* 和 *masked*。

其他资源

- 在 RHEL 中*使用 systemctl 管理系统服务*
- systemd.resource-control (5), systemd.exec (5) 手册页

33.6. 查看 SYSTEMD CGROUPS 层次结构

显示控制组(cgroup)层次结构以及在特定 cgroups 中运行的进程。

流程

- 使用 systemd-cgls 命令显示系统上整个 cgroups 层次结构。

```
# systemd-cgls
Control group /:
-.slice
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 965 gdm-session-worker [pam/gdm-launch-environment]
│ │ │ ├─1040 /usr/libexec/gdm-x-session gnome-session --autostart
/usr/share/gdm/greeter/autostart
...
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
  ...
  ├─example.service
  │ ├─6882 /bin/bash /home/jdoe/example.sh
  │ └─6902 sleep 1
  ├─systemd-journald.service
  │ └─629 /usr/lib/systemd/systemd-journald
  ...
```

示例输出返回整个 cgroups 层次结构，其中最高级别由 *slices* 组成。

- 使用 systemd-cgls <*resource_controller*> 命令显示资源控制器过滤的 cgroups 层次结构。

```
# systemd-cgls memory
Controller memory; Control group /:
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 965 gdm-session-worker [pam/gdm-launch-environment]
...
└─system.slice
  |
  ...
  ├─chronyd.service
  │ └─844 /usr/sbin/chronyd
  ├─example.service
  │ ├─8914 /bin/bash /home/jdoe/example.sh
  │ └─8916 sleep 1
  ...
```

示例输出列出了与所选控制器交互的服务。

- 使用 systemctl status <*system_unit*> 命令显示某个单元及其 cgroups 层次结构部分的详细信息。

```
# systemctl status example.service
● example.service - My example service
  Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset:
disabled)
  Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
Main PID: 17737 (bash)
   Tasks: 2 (limit: 11522)
  Memory: 496.0K (limit: 1.5M)
  CGroup: /system.slice/example.service
          ├─17737 /bin/bash /home/jdoe/example.sh
          └─17743 sleep 1
Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST
2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST
2019
```

其他资源

- 什么是内核资源控制器
- systemd.resource-control (5) 和 cgroups (7) 手册页

33.7. 查看进程的 CGROUP

您可以了解进程属于哪一个 *控制组* (cgroup)。然后，您可以检查 cgroup，以查找其使用哪个控制器和特定于控制器的配置。

流程

1. 要查看某个进程所属的 cgroup，请运行 # cat proc/<*PID*>/cgroup 命令：

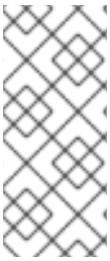
```
# cat /proc/2467/cgroup
0::/system.slice/example.service
```

输出示例与关注进程相关。在这种情况下，它是由 PID 2467 来标识的进程，它属于 example.service 单元。您可以确定该过程是否放置在 systemd 单元文件规格定义的正确控制组中。

2. 要显示 cgroup 使用哪些控制器和对应的配置文件，请检查 cgroup 目录：

```
# cat /sys/fs/cgroup/system.slice/example.service/cgroup.controllers
memory pids

# ls /sys/fs/cgroup/system.slice/example.service/
cgroup.controllers
cgroup.events
…
cpu.pressure
cpu.stat
io.pressure
memory.current
memory.events
…
pids.current
pids.events
pids.max
```



注意

cgroup 版本 1 层次结构使用每个控制器模型。因此，/proc/*PID*/cgroup 文件中的输出显示，PID 所属的每个控制器下的 cgroups。您可以在控制器目录（/sys/fs/cgroup/*<controller_name>*/）中查找相应的 cgroup。

其他资源

- cgroups (7) 手册页
- 什么是内核资源控制器
- /usr/share/doc/kernel-doc-<kernel_version>/Documentation/admin-guide/cgroup-v2.rst 文件的文档（安装 kernel-doc 软件包）

33.8. 监控资源消耗

查看当前运行的控制组(cgroup)的列表及其实时资源消耗。

流程

1. 使用 systemd-cgtop 命令显示当前运行的 cgroups 的动态帐户。

```
# systemd-cgtop
Control Group                          Tasks   %CPU   Memory  Input/s Output/s
/                                        607   29.8     1.5G        -        -
/system.slice                            125      -   428.7M        -        -
/system.slice/ModemManager.service         3      -     8.6M        -        -
/system.slice/NetworkManager.service       3      -    12.8M        -        -
/system.slice/accounts-daemon.service      3      -     1.8M        -        -
/system.slice/boot.mount                   -      -    48.0K        -        -
/system.slice/chronyd.service              1      -     2.0M        -        -
/system.slice/cockpit.socket               -      -     1.3M        -        -
/system.slice/colord.service               3      -     3.5M        -        -
/system.slice/crond.service                1      -     1.8M        -        -
/system.slice/cups.service                 1      -     3.1M        -        -
/system.slice/dev-hugepages.mount          -      -   244.0K        -        -
/system.slice/dev-mapper-rhel\x2dswap.swap -      -   912.0K        -        -
/system.slice/dev-mqueue.mount             -      -    48.0K        -        -
/system.slice/example.service              2      -     2.0M        -        -
/system.slice/firewalld.service            2      -    28.8M        -        -
...
```

示例输出显示当前运行的 cgroups，按照资源使用量排序（CPU、内存、磁盘 I/O 负载）。这个列表默认每 1 秒刷新一次。因此，它提供了一个动态洞察每个控制组的实际资源使用情况。

其他资源

- systemd-cgtop (1) 手册页

33.9. 使用 SYSTEMD 单元文件为应用程序设置限制

systemd 服务管理器监督每个现有或正在运行的单元，并为它们创建控制组。该单元在 /usr/lib/systemd/system/ 目录中有配置文件。

您可以手动将单元文件修改为：

- 设置限制。
- 优先顺序。

- 控制对进程组的硬件资源的访问。

先决条件

- 有 root 权限。

流程

1. 编辑 /usr/lib/systemd/system/example.service 文件，来限制服务的内存使用：

```
...
[Service]
MemoryMax=1500K
...
```

配置限制控制组中的进程不能超过的最大内存。example.service 服务是此类控制组群的一部分，它有一定的限制。使用后缀 K、M、G 或 T 将 Kilobyte、Megabyte、Gigabyte 或 Terabyte 作为一个测量单位。

2. 重新载入所有单元配置文件：

```
# systemctl daemon-reload
```

3. 重启服务：

```
# systemctl restart example.service
```

验证

1. 检查更改是否生效：

```
# cat /sys/fs/cgroup/system.slice/example.service/memory.max
1536000
```

示例输出显示，内存消耗会限制在大约 1,500 KB。

其他资源

- 了解 cgroups
- 在 Red Hat Enterprise Linux 中*使用 systemctl 管理系统服务*
- systemd.resource-control (5), systemd.exec (5), 和 cgroups (7) 手册页

33.10. 使用 SYSTEMCTL 命令将限制设置为应用程序

CPU 关联性设置可帮助您将特定进程的访问限制到某些 CPU。实际上，CPU 调度程序永远不会将进程调度到不在进程的关联性掩码中的 CPU 上运行。

默认 CPU 关联性掩码应用到 systemd 管理的所有服务。

要为特定的 systemd 服务配置 CPU 关联性掩码，systemd 提供了 CPUAffinity= 作为：

- 单元文件选项。
- /etc/systemd/system.conf 文件的 [Manager] 部分中的配置选项。

CPUAffinity= 单元文件选项设置 CPU 或 CPU 范围列表，它们被合并并用作关联性掩码。

流程

要使用 CPUAffinity 单元文件选项为特定的 systemd 服务设置 CPU 关联性掩码：

1. 在您选择的服务中检查 CPUAffinity 单元文件选项的值：

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. 以 root 用户身份，为用作关联性掩码的 CPU 范围设置 CPUAffinity 单元文件选项所需的值：

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. 重新启动服务以应用更改。

```
# systemctl restart <service name>
```

其他资源

- systemd.resource-control (5), systemd.exec (5), cgroups (7) 手册页

33.11. 通过管理器配置设置全局默认 CPU 关联性

/etc/systemd/system.conf 文件中的 CPUAffinity 选项为进程识别号(PID) 1 和从 PID1 分叉的所有进程定义一个关联性掩码。然后，您可以基于每个服务覆盖 CPUAffinity。

要使用 /etc/systemd/system.conf 文件为所有 systemd 服务设置默认的 CPU 关联性掩码：

1. 在 /etc/systemd/system.conf 文件的 [Manager] 部分中设置 CPUAffinity= 选项的 CPU 号。

2. 保存编辑的文件并重新载入 systemd 服务：

```
# systemctl daemon-reload
```

3. 重启服务器以应用更改。

其他资源

- systemd.resource-control (5) 和 systemd.exec (5) 手册页。

33.12. 使用 SYSTEMD 配置 NUMA 策略

非统一内存访问 (NUMA) 是一种计算机内存子系统设计，其中内存访问时间取决于处理器的物理内存位置。

接近 CPU 的内存的延迟（外部内存）比其他 CPU 本地内存低，或者在一组 CPU 间共享。

就 Linux 内核而言，NUMA 策略管理内核为进程分配物理内存页面的位置（例如，在哪些 NUMA 节点上）。

systemd 提供单元文件选项 NUMAPolicy 和 NUMAMask，以控制服务的内存分配策略。

流程

通过 NUMAPolicy 单元文件选项设置 NUMA 内存策略：

1. 在您选择的服务中检查 NUMAPolicy 单元文件选项的值：

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2. 作为根目录，设置 NUMAPolicy 单元文件选项所需的策略类型：

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3. 重新启动服务以应用更改。

```
# systemctl restart <service name>
```

要使用 [Manager] 配置选项设置全局 NUMAPolicy 设置：

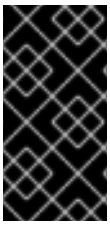
1. 在 /etc/systemd/system.conf 文件中搜索文件的 [Manager] 部分中的 NUMAPolicy 选项。

2. 编辑策略类型并保存文件。

3. 重新载入 systemd 配置：

```
# systemd daemon-reload
```

4. 重启服务器。



重要

当您配置严格的 NUMA 策略时，例如 bind，请确保也适当地设置 CPUAffinity= 单元文件选项。

其他资源

- 使用 systemctl 命令将限制设置为应用程序
- systemd.resource-control (5), systemd.exec (5) 和 set_mempolicy (2) 手册页。

33.13. SYSTEMD 的 NUMA 策略配置选项

Systemd 提供以下选项来配置 NUMA 策略：

NUMAPolicy

控制已执行进程的 NUMA 内存策略。您可以使用这些策略类型：

- default
- preferred
- bind
- interleave
- local

NUMAMask

控制与所选 NUMA 策略关联的 NUMA 节点列表。

请注意，您不必为以下策略指定 NUMAMask 选项：

- default
- local

对于首选策略，列表仅指定单个 NUMA 节点。

其他资源

- systemd.resource-control (5), systemd.exec (5) 和 set_mempolicy (2) 手册页

33.14. 使用 SYSTEMD-RUN 命令创建临时 CGROUP

临时 cgroup 设置运行时期间由单元（服务或范围）消耗的资源的限制。

流程

- 要创建一个临时控制组群，使用以下格式的 systemd-run 命令：

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

此命令会创建并启动临时服务或范围单元，并在此类单元中运行自定义命令。

  - --unit=<name> 选项为单元取一个名称。如果未指定 --unit，则会自动生成名称。
  - --slice=<*name*>.slice 选项使您的服务或范围单元成为指定片段的成员。将 <*name*>.slice 替换为现有片段的名称（如 systemctl -t slice 输出中所示），或通过传递唯一名称来创建新片段。默认情况下，服务和范围作为 system.slice 的成员创建。

- 使用您要在服务或范围单元中输入的命令替换 *<command>*。

  此时会显示以下信息，以确认您已创建并启动了该服务，或者已成功启动范围：

  ```
  # Running as unit <name>.service
  ```

- *可选* ：在进程完成后保持单元运行，以收集运行时信息：

  ```
  # systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
  ```

  命令会创建并启动临时服务单元，并在单元中运行自定义命令。**--remain-after-exit** 选项可确保服务在其进程完成后继续运行。

其他资源

- 什么是控制组
- 在 RHEL 中*管理 systemd*
- systemd-run (1) 手册页

33.15. 删除临时控制组群

如果您不再需要限制、确定或控制对进程组的硬件资源的访问，您可以使用 systemd 系统和服务管理器删除临时控制组 (cgroup) 。

当服务或范围单元包含的所有进程完成后，临时 cgroup 会自动释放。

流程

- 要停止带有所有进程的服务单元，请输入：

  ```
  # systemctl stop <name>.service
  ```

- 要终止一个或多个单元进程，请输入：

```
# systemctl kill <name>.service --kill-who=PID,... --signal=<signal>
```

命令使用 --kill-who 选项从您要终止的控制组中选择进程。要同时终止多个进程，请传递以逗号分隔的 PID 列表。--signal 决定要发送到指定进程的 POSIX 信号的类型。默认信号是 *SIGTERM*。

其他资源

- 什么是控制组
- 什么是内核资源控制器
- systemd.resource-control (5) 和 cgroups (7) 手册页
- 了解控制组群
- 在 RHEL 中*管理 systemd*

第 34 章 了解控制组群

使用控制组(cgroups)内核功能，您可以控制应用程序的资源使用情况来更有效地使用它们。

您可以为以下任务使用 cgroups ：

- 为系统资源分配设置限制。
- 将硬件资源优先分配给特定的进程。
- 防止某些进程获取硬件资源。

34.1. 控制组简介

使用 *控制组* Linux 内核功能，您可以将进程组织为按层排序的组 - cgroups。您可以通过为 cgroup 虚拟文件系统提供结构来定义层次结构（控制组树），默认挂载到 /sys/fs/cgroup/ 目录。

systemd 服务管理器使用 cgroups 来组织它管理的所有单元和服务。您可以通过创建和删除 /sys/fs/cgroup/ 目录中的子目录来手动管理 cgroups 的层次结构。

然后，内核中的资源控制器通过限制、优先处理或分配这些进程的系统资源来在 cgroups 中修改进程的行为。这些资源包括以下内容：

- CPU 时间
- 内存
- 网络带宽
- 这些资源的组合

cgroups 的主要用例是聚合系统进程，并在应用程序和用户之间划分硬件资源。这样可以提高环境的效率、稳定性和安全性。

控制组群版本 1

控制组版本 1 (cgroups-v1) 提供按资源控制器层次结构。这意味着每个资源（如 CPU、内存或 I/O）都有自己的控制组层次结构。您可以组合不同的控制组层次结构，使一个控制器可以与另一个控制器协调管理各自的资源。但是，当两个控制器属于不同的进程层次结构时，合适的协调会被限制。

cgroups-v1 控制器在很长的时间跨度开发，因此其控制文件的行为和命名不一致。

控制组群版本 2

控制组版本 2 (cgroups-v2)提供单一控制组层次结构，用于挂载所有资源控制器。

控制文件行为和命名在不同控制器之间保持一致。

重要

RHEL 9 默认挂载并使用 cgroups-v2。

其他资源

- 内核资源控制器简介
- cgroups (7) 手册页
- cgroups-v1
- cgroups-v2

34.2. 内核资源控制器简介

内核资源控制器启用控制组的功能。RHEL 9 支持适用于*控制组版本 1* (cgroups-v1) and *控制组版本 2* (cgroups-v2) 的不同控制器。

资源控制器也称为控制组子系统，是一个代表单一资源的内核子系统，如 CPU 时间、内存、网络带宽或磁盘 I/O。Linux 内核提供由 systemd 服务管理器自动挂载的一系列资源控制器。您可以在 /proc/cgroups 文件中找到当前挂载的资源控制器的列表。

cgroups-v1 提供的控制器：

blkio

设置对块设备的输入/输出访问的限制。

cpu

调整控制组任务的完全公平调度程序(CFS)的参数。cpu 控制器与 cpuacct 控制器一起挂载在同一挂载上。

cpuacct

创建控制组群中任务所使用的有关 CPU 资源的自动报告。cpuacct 控制器与 cpu 控制器一起挂载在同一挂载上。

cpuset

将控制组任务限制为仅在指定 CPU 子集上运行，并指示任务仅使用指定内存节点上的内存。

devices

控制控制组群中任务对设备的访问。

freezer

暂停或恢复控制组中的任务。

内存

设置控制组中任务对内存使用的限制，并对这些任务使用的内存资源生成自动报告。

net_cls

使用类标识符(classid)标记网络数据包，使 Linux 流量控制器(tc 命令)能够识别来自特定控制组任务的数据包。net_cls 子系统 net_filter (iptables) 也可使用此标签对此类数据包执行操作。net_filter 使用防火墙标识符(fwid)标记网络套接字，它允许 Linux 防火墙识别来自特定控制组任务的数据包（通过使用 iptables 命令）。

net_prio

设置网络流量的优先级。

pids

为控制组中的多个进程及其子进程设置限制。

perf_event

通过 perf 性能监控和报告工具对监控的任务进行分组。

rdma

对控制组群中远程直接内存访问/InfiniBand 特定资源设置限制。

hugetlb

可用于限制控制组中按任务的大量虚拟内存页的使用率。

cgroups-v2 提供的控制器：

io

设置对块设备的输入/输出访问的限制。

内存

设置控制组中任务对内存使用的限制，并对这些任务使用的内存资源生成自动报告。

pids

为控制组中的多个进程及其子进程设置限制。

rdma

对控制组群中远程直接内存访问/InfiniBand 特定资源设置限制。

cpu

为控制组的任务调整完全公平调度程序(CFS)的参数，并创建控制组中任务所使用的 CPU 资源的自动报告。

cpuset

将控制组任务限制为仅在指定 CPU 子集上运行，并指示任务仅使用指定内存节点上的内存。仅支持具有新分区功能的核心功能(cpus{,.effective}, mems{,.effective})。

perf_event

通过 perf 性能监控和报告工具对监控的任务进行分组。perf_event 在 v2 层次结构上自动启用。

重要

资源控制器可以在 cgroups-v1 层次结构或 cgroups-v2 层次结构中使用，不能同时在两者中使用。

其他资源

- cgroups (7) 手册页
- /usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/ 目录（安装 kernel-doc 包后）的文档。

34.3. 命名空间简介

命名空间是整理和识别软件对象的最重要的方法之一。

命名空间将全局系统资源（例如挂载点、网络设备或主机名）封装在抽象中，使名称空间中的进程看起来拥有自己的全局资源的隔离实例。使用命名空间的最常见技术是容器。

对特定全局资源的更改仅对该命名空间中的进程可见，不影响系统或其他命名空间的其余部分。

要检查进程所属的命名空间，您可以在 /proc/<*PID*>/ns/ 目录中检查符号链接。

表 34.1. 支持的命名空间以及它们隔离的资源：

命名空间	Isolates
Mount	挂载点
UTS	主机名和 NIS 域名
IPC	系统 V IPC, POSIX 消息队列
PID	进程 ID
Network	网络设备、堆栈、端口等
User	用户和组群 ID

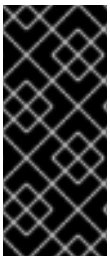
命名空间	Isolates
Control groups	控制组群根目录

其他资源

- **namespaces (7) 和 cgroup_namespaces (7) 手册页**
- **控制组简介**

第 35 章 使用 CGROUPFS 手动管理 CGROUP

您可以通过在 cgroupfs 虚拟文件系统中创建目录来管理系统上的 cgroup 层次结构。文件系统默认挂载到 /sys/fs/cgroup/ 目录中，您可以在专用控制文件中指定所需的配置。



重要

通常，红帽建议您使用 systemd 来控制系统资源的使用。您应该只在特殊情况下手动配置 cgroups 虚拟文件系统。例如，当您需要使用在 cgroup-v2 层次结构中没有对应的 cgroup-v1 控制器时。

35.1. 在 CGROUPS-V2 文件系统中创建 CGROUP 和启用控制器

您可以通过创建和删除目录，并通过写入 cgroup 虚拟文件系统中的文件来管理 *控制组* (cgroups) 。文件系统默认挂载到 /sys/fs/cgroup/ 目录中。要使用 cgroups 控制器中的设置，您还需要为子 cgroup 启用所需的控制器。在默认情况下，root cgroup 会为其子 cgroups 启用 memory 和 pids。因此，红帽建议在 /sys/fs/cgroup/ root cgroup 中创建至少两个级别的子 cgroup。这样，您可以选择从子 cgroup 中删除 memory 和 pids 控制器，并更好地组织 cgroup 文件。

先决条件

- 有 root 权限。

流程

1. 创建 /sys/fs/cgroup/Example/ 目录：

```
# mkdir /sys/fs/cgroup/Example/
```

/sys/fs/cgroup/Example/ 目录定义一个子组。当您创建 /sys/fs/cgroup/Example/ 目录时，目录中会自动创建一些 cgroups-v2 接口文件。/sys/fs/cgroup/Example/ 目录还包含针对 内存和 pids 控制器的特定于控制器的文件。

2. （可选）检查新创建的子组：

```
# ll /sys/fs/cgroup/Example/
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.procs
```

```
...
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
-r—r—r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r—r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r—r--. 1 root root 0 Jun  1 10:33 memory.low
...
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.current
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 pids.max
```

示例输出显示常规 cgroup 控制接口文件，如 cgroup.procs 或 cgroup.controllers。无论启用控制器是什么，这些文件都是所有控制组通用的。

memory.high 和 pids.max 等文件与 memory 和 pids 控制器有关，它们是 root 控制组 (/sys/fs/cgroup/) ，默认情况下会被 systemd 启用。

默认情况下，新创建的子组从父 cgroup 继承所有设置。在这种情况下，来自 root cgroup 没有限制。

3. 验证 /sys/fs/cgroup/cgroup.controllers 文件中是否有所需的控制器：

```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

4. 启用所需的控制器。在本例中是 cpu 和 cpuset 控制器：

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

这些命令为 /sys/fs/cgroup/ root 控制组的直接子组启用 cpu 和 cpuset 控制器。包含新创建的 Example 控制组。*子组* 是可以指定进程，并根据标准对每个进程应用控制检查的位置。

用户可以在任何级别上读取 cgroup.subtree_control 文件的内容，以了解哪些控制器将在直接子组中启用。

注意

默认情况下，root 控制组群中的 /sys/fs/cgroup/cgroup.subtree_control 文件包含 内存和 pids 控制器。

5. 为 Example 控制组群的子 cgroup 启用所需的控制器：

```
# echo "+cpu +cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

这些命令可确保，直接的子组仅具有与 CPU 时间分发相关的控制器，而不是 memory 或 pids 控制器。

6. 创建 /sys/fs/cgroup/Example/tasks/ 目录：

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

/sys/fs/cgroup/Example/tasks/ 目录定义了一个子组，它带有只与 cpu 和 cpuset 控制器相关的文件。现在，您可以将进程分配到此控制组，并将 cpu 和 cpuset 控制器选项用于您的进程。

7. （可选）检查子控制组群：

```
# ll /sys/fs/cgroup/Example/tasks
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.controllers
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.events
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.freeze
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.max.depth
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.max.descendants
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.procs
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.stat
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.subtree_control
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.threads
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.type
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.max
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.pressure
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus
-r—r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.effective
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.partition
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.mems
-r—r—r--. 1 root root 0 Jun  1 11:45 cpuset.mems.effective
-r—r—r--. 1 root root 0 Jun  1 11:45 cpu.stat
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.weight
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.weight.nice
-rw-r—r--. 1 root root 0 Jun  1 11:45 io.pressure
-rw-r—r--. 1 root root 0 Jun  1 11:45 memory.pressure
```

重要

只有相关的子组至少有 2 个进程在单个 CPU 上竞争时，才会激活 cpu 控制器。

验证步骤

- 可选：确认您已创建了一个新 cgroup，且只有所需控制器活跃：

```
# cat /sys/fs/cgroup/Example/tasks/cgroup.controllers
cpuset cpu
```

其他资源

- 了解控制组群
- 什么是内核资源控制器
- 挂载 cgroups-v1
- cgroups (7)、sysfs (5) 手册页

35.2. 通过调整 CPU 权重来控制应用程序的 CPU 时间

您需要为 cpu 控制器的相关文件分配值，以规范特定 cgroup 树下的应用程序分布 CPU 时间。

先决条件

- 有 root 权限。
- 您有要控制 CPU 时间分布的应用程序。
- 您在 /sys/fs/cgroup/ *root 控制组群*中创建两个级别的*子控制组群*，如下例所示：

```
...
├── Example
```

```
│   ├── g1
│   ├── g2
│   └── g3
...
```

- 您已在父控制组和子控制组中启用 cpu 控制器，类似于在 cgroups-v2 文件系统中创建 cgroups 并启用控制器。

流程

1. 配置所需的 CPU 权重以便在控制组群内实施资源限制：

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2. 将应用程序的 PID 添加到 g1、g2 和 g3 子组中：

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

示例命令可确保所需的应用程序成为 Example/g*/ 子 cgroup 的成员，并获取按照这些 cgroup 配置分发的 CPU 时间。

已在运行的子 cgroups (g1, g2, g3) 的权重在父 cgroup (Example) 一级计算其总和。然后，CPU 资源会根据对应的权重按比例分发。

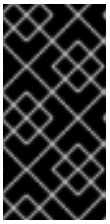
因此，当所有进程都同时运行时，内核会根据相应 cgroup 的 cpu.weight 文件为每个进程分配相应比例的 CPU 时间：

子 cgroup	cpu.weight 文件	CPU 时间分配
g1	150	~50% (150/300)
g2	100	~33% (100/300)
g3	50	~16% (50/300)

cpu.weight 控制器文件的值不是一个百分比。

如果一个进程停止运行，造成 cgroup g2 中没有运行的进程，则计算将省略 cgroup g2，仅根据 cgroup g1 和 g3 进行计算：

子 cgroup	cpu.weight 文件	CPU 时间分配
g1	150	~75% (150/200)
g3	50	~25% (50/200)



重要

如果子 cgroup 有多个正在运行的进程，分配给相应 cgroup 的 CPU 时间将平均分配到该 cgroup 的成员进程。

验证

1. 验证应用程序是否在指定的控制组群中运行：

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

命令输出显示了在 Example/g*/ 子 cgroups 中运行的特定应用程序的进程。

2. 检查节流应用程序的当前 CPU 消耗：

```
# top
top - 05:17:18 up 1 day, 18:25,  1 user,  load average: 3.03, 3.03, 3.00
Tasks:  95 total,   4 running,  91 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.1 us, 81.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
MiB Mem :   3737.0 total,   3233.7 free,    132.8 used,    370.5 buff/cache
MiB Swap:   4060.0 total,   4060.0 free,      0.0 used.   3373.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
33373 root      20   0   18720   1748   1460 R  49.5   0.0 415:05.87 sha1sum
33374 root      20   0   18720   1756   1464 R  32.9   0.0 412:58.33 sha1sum
33377 root      20   0   18720   1860   1568 R  16.3   0.0 411:03.12 sha1sum
  760 root      20   0  416620  28540  15296 S   0.3   0.7   0:10.23 tuned
```

```
    1 root      20   0  186328  14108   9484 S   0.0   0.4   0:02.00 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthread
...
```

注意

为了明确演示，我们强制所有示例进程在单个 CPU 上运行。在多个 CPU 中使用时，CPU 权重也会应用同样的原则。

请注意，PID 33373、PID 33374 和 PID 33377 的 CPU 资源根据权重 150、100、50 分配的。权重对应于每个应用程序的 50%、33% 和 16% 的 CPU 时间。

其他资源

- 了解控制组群
- 什么是内核资源控制器
- 在 cgroups-v2 文件系统中创建 cgroup 和启用控制器
- 资源分配模型
- cgroups (7)、sysfs (5) 手册页

35.3. 挂载 CGROUPS-V1

在引导过程中，RHEL 9 默认挂载 cgroup-v2 虚拟文件系统。要在限制应用程序的资源中使用 cgroup-v1 功能，请手动配置系统。

注意

内核中完全启用了 cgroup-v1 和 cgroup-v2。从内核的角度来看，没有默认的控制组版本，并且由 systemd 决定在启动时挂载。

先决条件

- 有 root 权限。

流程

1. 将系统配置为，在系统引导过程中，默认由 systemd 系统和服务管理器挂载 cgroups-v1 ：

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

这会在当前引导条目中添加所需的内核命令行参数。

在所有内核引导条目中添加相同的参数：

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

2. 重启系统以使更改生效。

验证

1. （可选）验证已挂载 cgroups-v1 文件系统：

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs
(ro,nosuid,nodev,noexec,seclabel,size=4096k,nr_inodes=1024,mode=755,inode64)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/syste
md-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpuset)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,memory)
```

```
cgroup on /sys/fs/cgroup/blkio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,blkio)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,devices)
cgroup on /sys/fs/cgroup/misc type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,misc)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,freezer)
cgroup on /sys/fs/cgroup/rdma type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,rdma)
```

与各种 cgroup-v1 控制器对应的 cgroups-v1 文件系统已被成功挂载到 /sys/fs/cgroup/ 目录中。

2. （可选）检查 /sys/fs/cgroup/ 目录的内容：

```
# ll /sys/fs/cgroup/
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 blkio
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpu → cpu,cpuacct
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpuacct → cpu,cpuacct
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 cpu,cpuacct
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 cpuset
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 devices
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 freezer
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 hugetlb
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 memory
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 misc
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_cls → net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 net_cls,net_prio
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_prio → net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 perf_event
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 pids
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 rdma
dr-xr-xr-x. 11 root root  0 Mar 16 09:34 systemd
```

默认情况下，/sys/fs/cgroup/ 目录（也称为 *root 控制组*）包含特定于控制器的目录，如 cpuset。另外，还有一些与 systemd 相关的目录。

其他资源

- 了解控制组群
- 什么是内核资源控制器

- cgroups (7) 、sysfs (5) 手册页
- RHEL 9 默认启用 cgroup-v2

35.4. 使用 CGROUPS-V1 为应用程序设置 CPU 限制

要使用 *控制组版本 1* (cgroups-v1)配置对应用程序的 CPU 限制，请使用 /sys/fs/ 虚拟文件系统。

先决条件

- 有 root 权限。
- 您有一个要限制其 CPU 消耗的应用程序。
- 您将系统配置为，在系统引导过程中，默认由 systemd 系统和服务管理器挂载 cgroups-v1 :

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

这会在当前引导条目中添加所需的内核命令行参数。

流程

1. 识别您要限制 CPU 消耗的应用程序的进程 ID (PID) :

```
# top
top - 11:34:09 up 11 min,  1 user,  load average: 0.51, 0.27, 0.22
Tasks: 267 total,   3 running, 264 sleeping,   0 stopped,   0 zombie
%Cpu(s): 49.0 us,  3.3 sy,  0.0 ni, 47.5 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
MiB Mem :   1826.8 total,    303.4 free,   1046.8 used,    476.5 buff/cache
MiB Swap:   1536.0 total,   1396.0 free,    140.0 used.    616.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6955 root      20   0  228440   1752   1472 R  99.3   0.1   0:32.71 sha1sum
 5760 jdoe      20   0 3603868 205188  64196 S   3.7  11.0   0:17.19 gnome-shell
 6448 jdoe      20   0  743648  30640  19488 S   0.7   1.6   0:02.73 gnome-terminal-
```

```
  1 root      20   0  245300   6568   4116 S   0.3   0.4   0:01.87 systemd
505 root      20   0       0      0      0 I   0.3   0.0   0:00.75 kworker/u4:4-events_unbound
...
```

这个 top 程序的示例输出显示， PID 6955 的说明性应用程序 sha1sum 消耗了大量 CPU 资源。

2. 在 cpu 资源控制器目录中创建子目录：

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

此目录代表控制组，您可以在其中放置特定进程，并向进程应用某些 CPU 限制。同时，目录中将创建多个 cgroups-v1 接口文件和 cpu 特定于控制器的文件。

3. *可选* ：检查新创建的控制组：

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r—r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r—r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.shares
-r—r—r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r—r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r—r--. 1 root root 0 Mar 11 11:42 tasks
```

这个示例输出显示文件，如 cpuacct.usage、cpu.cfs._period_us，它们代表特定的配置和/或限值，可以为 Example 控制组中的进程设置。请注意，对应的文件名以它们所属的控制组控制器的名称为前缀。

默认情况下，新创建的控制组继承对系统整个 CPU 资源的访问权限，且无限制。

4. 为控制组群配置 CPU 限制：

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

- cpu.cfs_period_us 文件表示以微秒为单位（这里表示为"us"）的时段，用于控制组对 CPU 资源的访问权限应重新分配的频率。上限为 1 000 000 微秒，下限为 1 000 微秒。

- cpu.cfs_quota_us 文件表示以微秒为单位的总时间量，控制组中的所有进程都可以在一个期间（如 cpu.cfs_period_us 定义）。当控制组中的进程在单个期间内使用完配额指定的所有时间时，在周期的剩余时间内，它们将被节流，并且不允许在下一个期间前运行。较低限制为 1 000 微秒。

上面的示例命令设定 CPU 时间限值，使得 Example 控制组中的所有进程仅能每 1 秒（cpu.cfs_quota_us 定义）每 1 秒（由 cpu.cfs_period_us 定义）运行 0.2 秒。

5. *可选*：验证限制：

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6. 将应用程序的 PID 添加到 Example 控制组群中：

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs
```

此命令确保特定应用成为 Example 控制组的一员，因此不超过为 Example 控制组配置的 CPU 限值。PID 应该代表系统中的一个已存在的进程。这里的 PID 6955 分配给进程 sha1sum /dev/zero &，用于说明 cpu 控制器的用例。

验证

1. 验证应用程序是否在指定的控制组群中运行：

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
```

```
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-
server.service
```

此示例输出显示在 Example 控制组中运行的所需应用程序的进程，后者将 CPU 限制应用到应用程序的进程。

2. 确定节流应用程序的当前 CPU 消耗：

```
# top
top - 12:28:42 up  1:06,  1 user,  load average: 1.02, 1.02, 1.00
Tasks: 266 total,   6 running, 260 sleeping,   0 stopped,   0 zombie
%Cpu(s): 11.0 us,  1.2 sy,  0.0 ni, 87.5 id,  0.0 wa,  0.2 hi,  0.0 si,  0.2 st
MiB Mem :   1826.8 total,    287.1 free,   1054.4 used,    485.3 buff/cache
MiB Swap:   1536.0 total,   1396.7 free,    139.2 used.    608.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6955 root      20   0  228440   1752   1472 R  20.6   0.1  47:11.43 sha1sum
 5760 jdoe      20   0 3604956 208832  65316 R   2.3  11.2   0:43.50 gnome-shell
 6448 jdoe      20   0  743836  31736  19488 S   0.7   1.7   0:08.25 gnome-terminal-
  505 root      20   0       0      0      0 I   0.3   0.0   0:03.39 kworker/u4:4-events_unbound
 4217 root      20   0   74192   1612   1320 S   0.3   0.1   0:01.19 spice-vdagentd
...
```

请注意，PID 6955 的 CPU 消耗从 99% 减少到 20%。

注意

cpu.cfs_period_us 和 cpu.cfs_quota_us 在 cgroups-v2 的对应部分是 cpu.max 文件。cpu.max 文件可以通过 cpu 控制器获得。

其他资源

- 了解控制组群
- 内核资源控制器

- **cgroups (7)** 、 **sysfs (5)** 手册页

第 36 章 使用 BPF COMPILER COLLECTION 分析系统性能

作为系统管理员，您可以使用 BPF Compiler Collection (BCC) 库创建用于分析 Linux 操作系统性能和收集信息的工具，这些信息可能难以通过其他接口获得。

36.1. 安装 BCC-TOOLS 软件包

安装 bcc-tools 软件包，该软件包还会将 BPF Compiler Collection (BCC)库作为依赖项安装。

流程

1. 安装 bcc-tools。

```
# dnf install bcc-tools
```

BCC 工具安装在 /usr/share/bcc/tools/ 目录中。

2. （可选）检查工具：

```
# ll /usr/share/bcc/tools/
...
-rwxr-xr-x. 1 root root  4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root  3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root  7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root  8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root  7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root  6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

上表中的 doc 目录包含每个工具的文档。

36.2. 使用所选 BCC-TOOLS 进行性能调整

使用 BPF Compiler Collection (BCC)库中的某些预先创建的程序来根据每个事件高效、安全地分析系统性能。BCC 库中预创建的程序集可作为创建其他程序的示例。

先决条件

- 安装 bcc-tools 软件包
- 根权限

使用 execsnoop 检查系统进程

1. 在一个终端中运行 execsnoop 程序：

```
# /usr/share/bcc/tools/execsnoop
```

2. 在另一个终端中运行，例如：

```
$ ls /usr/share/bcc/tools/doc/
```

以上可创建 ls 命令的短时间进程。

3. 运行 execsnoop 的终端显示类似如下的输出：

```
PCOMM PID    PPID   RET ARGS
ls    8382   8287     0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop 程序打印出每个占用系统资源的新进程的输出行。它甚至会检测很快运行的程序（如 ls ）的进程，大多数监控工具也不会进行注册。

execsnoop 输出显示以下字段：

PCOMM

父进程名称。(ls)

PID

进程 ID。(8382)

PPID

父进程 ID。(8287)

RET

exec（） 系统调用的返回值(0)，其将程序代码加载到新进程中。

ARGS

启动的程序的参数的位置。

要查看 execsnoop 的详情、示例和选项，请参阅 /usr/share/bcc/tools/doc/execsnoop_example.txt 文件。

有关 exec () 的详情，请查看 exec (3) 手册页。

使用 opensnoop 跟踪命令打开的文件

1. 在一个终端中运行 opensnoop 程序：

```
# /usr/share/bcc/tools/opensnoop -n uname
```

以上列出了文件的输出，这些文件仅由 uname 命令的进程打开。

2. 在另一个终端中，输入：

```
$ uname
```

以上命令会打开某些在下一步中捕获的文件。

3. 运行 opensnoop 的终端显示类似如下的输出：

```
PID    COMM  FD ERR PATH
8596   uname  3  0   /etc/ld.so.cache
8596   uname  3  0   /lib64/libc.so.6
8596   uname  3  0   /usr/lib/locale/locale-archive
...
```

opensnoop 程序在整个系统中监视 open () 系统调用，并为 uname 尝试打开的每个文件打印一行输出。

opensnoop 输出显示以下字段：

PID

进程 ID。(8596)

COMM

进程名称。(uname)

FD

文件描述符 - open（） 返回的值，以指向打开的文件。(3)

ERR

任何错误。

PATH

open（） 试图打开的文件的位置。

如果命令尝试读取不存在的文件，则 FD 列返回 -1，ERR 列将打印与相关错误对应的值。因此，Opennoop 可以帮助您识别行为不正确的应用程序。

要查看 opensnoop 的更多详细信息、示例和选项，请参阅 /usr/share/bcc/tools/doc/opensnoop_example.txt 文件。

有关 open () 的更多信息，请参阅 open (2) 手册页。

使用技术检查磁盘上的 I/O 操作

1. 在一个终端中运行 biotop 程序：

```
# /usr/share/bcc/tools/biotop 30
```

该命令可让您监控在磁盘中执行 I/O 操作的主要进程。参数确保命令生成 30 秒概述。

注意

如果未提供任何参数，则默认情况下输出屏幕会每 1 秒刷新一次。

2. 在另一个终端中输入，例如：

```
# dd if=/dev/vda of=/dev/zero
```

以上命令从本地硬盘设备读取内容，并将输出写入 /dev/zero 文件。此步骤会生成特定的 I/O 流量来演示 biotop。

3. 运行 biotop 的终端显示类似如下的输出：

```
PID    COMM             D MAJ MIN DISK       I/O  Kbytes     AVGms
9568   dd               R 252 0   vda      16294 14440636.0  3.69
48     kswapd0          W 252 0   vda       1763 120696.0    1.65
7571   gnome-shell      R 252 0   vda        834 83612.0     0.33
1891   gnome-shell      R 252 0   vda       1379 19792.0     0.15
7515   Xorg             R 252 0   vda        280  9940.0     0.28
7579   llvmpipe-1       R 252 0   vda        228  6928.0     0.19
9515   gnome-control-c  R 252 0   vda         62  6444.0     0.43
8112   gnome-terminal-  R 252 0   vda         67  2572.0     1.54
7807   gnome-software   R 252 0   vda         31  2336.0     0.73
9578   awk              R 252 0   vda         17  2228.0     0.66
7578   llvmpipe-0       R 252 0   vda        156  2204.0     0.07
9581   pgrep            R 252 0   vda         58  1748.0     0.42
7531   InputThread      R 252 0   vda         30  1200.0     0.48
7504   gdbus            R 252 0   vda          3  1164.0     0.30
1983   llvmpipe-1       R 252 0   vda         39   724.0     0.08
1982   llvmpipe-0       R 252 0   vda         36   652.0     0.06
...
```

biotop 输出显示以下字段：

PID

进程 ID。(9568)

COMM

进程名称。(dd)

DISK

执行读操作的磁盘。(vda)

I/O

执行的读操作的数量。(16294)

Kbytes

读操作达到的 Kbytes 量。(14,440,636)

AVGms

读操作的平均 I/O 时间。(3.69)

要查看 biotop 的详情、示例和选项，请参阅 /usr/share/bcc/tools/doc/biotop_example.txt 文件。

有关 dd 的更多信息，请参阅 dd (1) 手册页。

使用 xfsslower 来公开意料外的慢文件系统操作

1. 在一个终端中运行 xfsslower 程序：

```
# /usr/share/bcc/tools/xfsslower 1
```

以上命令测量 XFS 文件系统执行读取、写入、打开或同步 (fsync) 操作的时间。1 参数可确保程序仅显示比 1 ms 较慢的操作。

注意

如果未提供任何参数，xfsslower 默认会显示比 10 ms 慢的操作。

2. 在另一个终端中输入以下内容：

```
$ vim text
```

以上命令在 vim 编辑器中创建了一个文本文件，用于启动与 XFS 文件系统的某些互动。

3. 运行 xfsslower 的终端显示在保存上一步中的文件时：

```
TIME     COMM          PID    T BYTES   OFF_KB   LAT(ms) FILENAME
13:07:14 b'bash'       4754   R 256     0          7.11 b'vim'
13:07:14 b'vim'        4754   R 832     0          4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'        4754   R 32      20         1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'        4754   R 1982    0          2.30 b'vimrc'
13:07:14 b'vim'        4754   R 1393    0          2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'        4754   S 0       0          6.71 b'text'
13:07:45 b'pool'       2588   R 16      0          5.58 b'text'
...
```

上面的每一行代表文件系统中的一个操作，其用时超过特定阈值。xfsslower 非常适合公开可能的文件系统问题，这可能会导致意外的慢速操作。

xfsslower 输出显示以下字段：

COMM

进程名称。(b'bash')

T

操作类型。(R)

- Read
- Write
- Sync

OFF_KB

KB 为单位的文件偏移。(0)

FILENAME

被读、写或同步的文件。

要查看 xfsslower 的详情、示例和选项，请参阅 /usr/share/bcc/tools/doc/xfsslower_example.txt 文件。

有关 fsync 的详情请参考 fsync (2) 手册页。

第 37 章 配置操作系统以优化内存访问

您可以使用 RHEL 中包含的工具来配置操作系统，以优化跨工作负载的内存访问

37.1. 监控和诊断系统内存问题的工具

以下工具包括在 Red Hat Enterprise Linux 9 中，用于监控系统性能并诊断与系统内存相关的性能问题：

- vmstat 工具由 procps-ng 软件包提供，显示系统的进程、内存、分页、块 I/O、陷阱、磁盘和 CPU 活动的报告。它自计算机上次打开或自上次启动以来，提供这些事件平均的即时报告，或者自上次报告起。

- valgrind 框架提供了用户空间二进制文件的工具。使用 dnf install valgrind 命令安装此工具。它包括很多工具，可用于对程序性能进行性能分析和分析，例如：

  - memcheck 选项是默认的 valgrind 工具。它检测并报告一些可能很难检测和诊断的内存错误，例如：

    - 不应该发生的内存访问
    - 未定义或未初始化的值使用
    - 空闲的堆内存不正确
    - 指针重叠
    - 内存泄漏

注意

Memcheck 只能报告这些错误，它无法防止它们发生。但是，memcheck 会在错误发生前立即记录错误消息。

- cachegrind 选项模拟与系统的缓存层次结构和分支预测应用程序交互。它收集应用的执行持续时间的统计信息，并输出控制台的摘要。

- massif 选项测量指定应用程序使用的堆空间。它测量有用的空间以及为预订和协调目的而分配的额外空间。

其他资源

- vmstat (8) 和 valgrind (1) man page

- /usr/share/doc/valgrind-version/valgrind_manual.pdf file

37.2. 系统内存概述

Linux 内核旨在最大程度提高系统内存资源 (RAM) 的利用率。由于这些设计特性，根据工作负载的内存要求，系统内存部分在内核内代表工作负载使用，而一小部分内存可用。这个空闲内存可用于特殊系统分配，也保留用于其他低或者高优先级系统服务。

系统内存的其余部分专用于工作负载本身，并分为以下两类：

文件内存

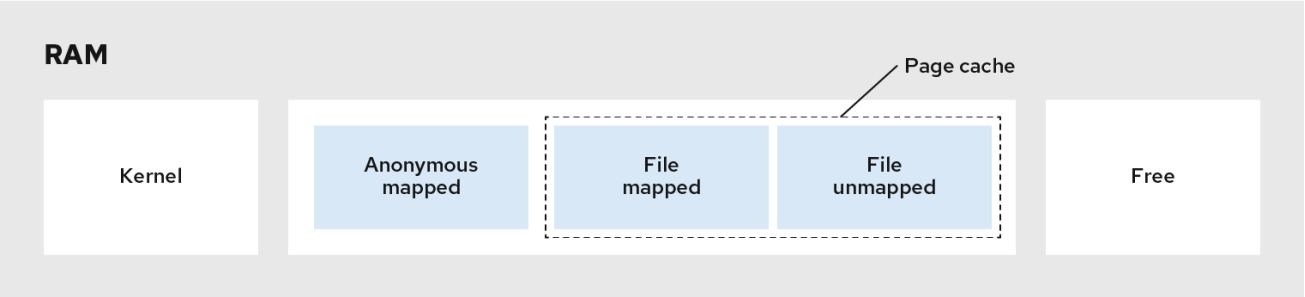
这个类别中添加的页面代表持久性存储中的部分文件。这些页面缓存中的页面可以在应用程序的地址空间中映射或取消映射。您可以使用应用程序将文件映射到其地址空间，或使用 mmap 系统调用，或通过缓冲的 I/O 读写系统调用处理文件。

缓冲区的 I/O 系统调用以及直接映射页面的应用程序可以重新使用未映射的页面。因此，这些页面会由内核存储在缓存中，特别是当系统没有运行任何内存密集型任务时，以避免对同一组页面造成昂贵的 I/O 操作。

匿名内存

此类别的页面由动态分配的进程使用，或者与持久性存储中的文件无关。这组页面会备份每个任务的内存中控制结构，如应用堆栈和堆区域。

图 37.1. 内存用量模式



37.3. 虚拟内存参数

虚拟内存参数列在 /proc/sys/vm 目录中。

以下是可用的虚拟内存参数：

vm.dirty_ratio

是一个百分比值。修改系统内存占总内存的百分比时，系统通过 pdflush 操作开始向磁盘写入修改。默认值为 20 百分比。

vm.dirty_background_ratio

一个百分比值。修改系统内存占总内存的百分比时，系统会在后台开始向磁盘写入修改。默认值为 10（百分比）。

vm.overcommit_memory

定义决定接受或拒绝大型内存请求的条件。默认值为 0。

默认情况下，内核会执行检查虚拟内存分配请求是否适合于存在的内存量 (total + swap)，并只拒绝大型请求。否则会授予虚拟内存分配，这就意味着它们允许内存过量使用。

设置 overcommit_memory 参数的值：

- 当这个参数设置为 1 时，内核不会执行内存过量使用处理。这会增加内存过载的可能性，但提高了内存密集型任务的性能。
- 当这个参数设定为 2 时，内核拒绝对内存的请求（等于或大于可用交换空间总量）以及

overcommit_ratio 中指定的物理 RAM 百分比。这可降低过量使用内存的风险，但建议只针对交换区大于其物理内存的系统。

vm.overcommit_ratio

当 overcommit_memory 设为 2 时，指定物理 RAM 的百分比。默认值为 50。

vm.max_map_count

定义进程可以使用的最大内存映射区域数。默认值为 65530。如果您的应用程序需要更多内存映射区域，则提高这个值。

vm.min_free_kbytes

设置保留可用页面池的大小。它还负责设置管理 Linux 内核页面回收算法行为的 min_page、low_page 和 high_page 阈值。它还指定在系统间保留的最小 KB 数。这会为每个低内存区计算一个特定值，每个值都会被分配一个保留的空闲页面的大小。

设置 vm.min_free_kbytes 参数的值：

- 增加参数值可有效减少应用程序工作集可用内存。因此，您可能希望将其仅用于内核驱动的工作负载，其中驱动程序缓冲区需要在原子上下文中分配。
- 减少参数值可能会导致内核无法服务系统请求，如果内存在系统中发生大量处理。

警告

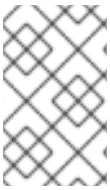
极端的值可能会降低系统性能。将 vm.min_free_kbytes 设置为非常低的值可防止系统有效地回收内存，这可能会导致系统崩溃并失败服务中断或其他内核服务。但是，设置 vm.min_free_kbytes 太大地增加系统回收活动，从而导致分配延迟因为假的直接重新声明状态而造成分配延迟。这可能导致系统立即进入内存不足状态。

vm.min_free_kbytes 参数还设置页面重新声明水位线，名为 min_pages。在确定两个其他内存水位线、low_pages 和 high_pages 时，这个水位线被用作一个因素，它管理页面重新声明算法。

/proc/*PID*/oom_adj

如果系统内存不足，并且 panic_on_oom 参数设置为 0，oom_killer 功能会终止进程，从具有最高 oom_score 的进程开始，直到系统恢复为止。

oom_adj 参数决定了进程的 oom_score。这个参数会根据进程标识符设置。值 -17 可禁用该进程的 oom_killer。其他有效的值范围从 -16 到 15。



注意

由调整的进程创建的进程将继承该进程的 oom_score。

vm.swappiness

swappiness 值范围从 0 到 200，控制系统从匿名内存池回收内存或页面缓存内存池的程度。

设置 swappiness 参数值：

- 高的数值代表，优先选择文件映射驱动的工作负载，同时交换出不活跃访问的进程的 RAM 匿名映射内存。这对文件服务器或流式应用（来自存储中的文件）很有用，从而驻留在内存上，以减少服务请求的 I/O 延迟。

- 低值优先选择匿名映射驱动的工作负载，同时回收页面缓存（文件映射内存）。这个设置对于不依赖于文件系统信息的应用程序很有用，并且主要利用动态分配和私有内存，如数学和数字缩小应用程序，以及某些硬件虚拟化超级visors （如 QEMU）。

vm.swappiness 参数的默认值为 60。

警告

将 vm.swappiness 设置为 0 会积极避免将匿名内存交换出磁盘，这会在内存不足或 I/O 密集型工作负载时，增加了进程被 oom_killer 函数终止的风险。

其他资源

- sysctl(8) man page
- 设置与内存相关的内核参数

37.4. 文件系统参数

文件系统参数在 /proc/sys/fs 目录中列出。以下是可用的文件系统参数：

aio-max-nr

定义所有活跃异步输入/输出上下文中允许的最大事件数。默认值为 65536，修改这个值不会预先分配或调整任何内核数据结构。

file-max

决定整个系统的最大文件句柄数。Red Hat Enterprise Linux 9 的默认值为 8192 或在内核启动时可用的空闲内存页的十分之一（以较大值为准）。

增加这个值可能会解决缺少可用文件句柄造成的错误。

其他资源

- sysctl(8) man page

37.5. 内核参数

内核参数的默认值位于 /proc/sys/kernel/ 目录中。它们由内核提供或通过 sysctl 指定的值设置默认值。

以下是用于为 msg* 和 shm* System V IPC (sysvipc) 系统调用设置限制的可用内核参数：

msgmax

定义消息队列中任何单个消息允许的最大大小（以字节为单位）。这个值不得超过队列的大小 (msgmnb)。使用 sysctl msgmax 命令确定系统中的当前 msgmax 值。

msgmnb

定义单个消息队列的最大大小（以字节为单位）。使用 **sysctl msgmnb** 命令确定系统中的当前 **msgmnb** 值。

msgmni

定义消息队列标识符的最大数量，因此定义队列的最大数量。使用 **sysctl msgmni** 命令确定系统中的当前 **msgmni** 值。

shmall

定义系统一次可以使用的共享内存页面总量。例如，AMD64 和 Intel 64 构架中的页面是 4096 字节。使用 **sysctl shmall** 命令确定系统中的当前 **shmall** 值。

shmmax

定义内核允许的单个共享内存段的最大大小（以字节为单位）。现在在内核中支持共享内存片段最多 1Gb。使用 **sysctl shmmax** 命令确定系统中的当前 **shmmax** 值。

shmmni

定义系统范围共享内存段的最大数量。所有系统上的默认值为 4096。

其他资源

- **sysvipc(7) 和 sysctl(8) man pages**

37.6. 设置与内存相关的内核参数

临时设置参数有助于确定参数在系统上具有的影响。当您确保参数值具有所需的效果时，您可以永久设置该参数。

这个步骤描述了如何临时和永久设置与内存相关的内核参数。

流程

- 要临时设置与内存相关的内核参数，请编辑 **/proc** 文件系统或 **sysctl** 工具中的相应文件。

  例如，要临时将 **vm.overcommit_memory** 参数设置为 ***1***:

```
# echo 1 > /proc/sys/vm/overcommit_memory
# sysctl -w vm.overcommit_memory=1
```

- **要永久设置与内存相关的内核参数，请编辑 /etc/sysctl.conf 文件并重新载入设置。**

  例如，要永久将 vm.overcommit_memory 参数设置为 *1*:

  - **在 /etc/sysctl.conf 文件中添加以下内容：**

    ```
    vm.overcommit_memory=1
    ```

  - **重新载入来自 /etc/sysctl.conf 文件的 sysctl 设置：**

    ```
    # sysctl -p
    ```

其他资源

- **sysctl(8) man page**
- **proc(5) man page**

其他资源

- 为 **IBM DB2 调整 Red Hat Enterprise Linux**

第 38 章 配置巨页

物理内存以固定大小的块的形式进行管理，称为页。在 x86_64 架构中，由 Red Hat Enterprise Linux 9 支持，默认的内存页大小为 4 KB。此默认页面大小被证明适用于常规用途的操作系统，如 Red Hat Enterprise Linux，它支持许多不同类型的工作负载。

但是，在某些情况下，特定应用程序可能会从使用更大的页面中受益。例如，在使用 4 KB 页面时，如果应用程序需要处理大量的、相对固定的数据集合时（有几百兆字节甚至数千兆字节数据）可能会遇到性能问题。这种数据集可能需要大量的 4 KB 页面，这可能导致操作系统和 CPU 的大量开销。

这部分提供有关 RHEL 9 中巨页的信息以及如何配置它们。

38.1. 可用的巨页功能

在 Red Hat Enterprise Linux 9 中，您可以使用巨页来处理大数据集的应用程序，并改进此类应用程序的性能。

以下是 RHEL 9 支持的巨页方法：

HugeTLB 页

HugeTLB 页面也称为静态巨页。有两种方法可以保留 HugeTLB 页面：

- 在引导时：这可以增加成功的几率，因为在引导时内存还没有很大的碎片。但是，在 NUMA 机器上，页面数量会自动在不同的 NUMA 节点间进行分隔。

有关影响引导时 HugeTLB 页行为的参数的更多信息，请参阅 在引导时保留 HugeTLB 页的参数 ，以及如何使用这些参数在引导时配置 HugeTLB 页，请参阅 在引导时配置 HugeTLB 。

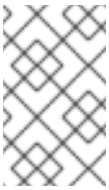
- 在运行时：它允许您为每个 NUMA 节点保留巨页。如果在引导过程中以早期方式进行运行时保留，则可能会降低内存碎片的可能性。

有关在运行时影响 HugeTLB 页行为的参数的更多信息，请参阅 在运行时保留 HugeTLB 页的参数，以及如何使用这些参数在运行时配置 HugeTLB 页，请参阅 在运行时配置 HugeTLB 。

透明巨页 (THP)

使用 THP 时，内核会自动将巨页分配给进程，因此不需要手动保留静态巨页。以下是 THP 中的两种操作模式：

- 系统范围 ：内核会在可以分配巨页时尝试为进程分配巨页，进程会使用一个大型连续的虚拟内存区域。
- 针对每个进程 ：内核仅将巨页分配给单个进程的内存区域，您可以使用 madvise () 系统调用来指定。



注意

THP 功能只支持 2 MB 页面。

有关在引导时影响 HugeTLB 页行为的参数的更多信息，请参阅 启用透明巨页 和 禁用透明巨页。

38.2. 在引导时保留 HUGETLB 页面的参数

使用以下参数来在引导时影响 HugeTLB 页面行为。

有关如何在引导时使用这些参数配置 HugeTLB 页面的更多信息，请参阅在引导时配置 HugeTLB 页面。

表 38.1. 用于在引导时配置 HugeTLB 页面的参数

参数	描述	默认值
hugepages	定义在引导时在内核中配置的持久性巨页数量。 在 NUMA 系统中，定义了这个参数的巨页在节点间平均划分。 您可以通过更改 **/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages** 文件中的节点值，在运行时为特定节点分配巨页。	默认值为 **0**。 要在引导时更新这个值，在 **/proc/sys/vm/nr_hugepages** 文件中更改这个参数的值。

参数	描述	默认值
hugepagesz	定义在引导时在内核中配置的持久性巨页的大小。	有效值为 **2 MB** 和 **1 GB**。默认值为 **2 MB**。
default_hugepagesz	定义在引导时在内核中配置的持久性巨页的默认大小。	有效值为 **2 MB** 和 **1 GB**。默认值为 **2 MB**。

38.3. 在引导时配置 HUGETLB

HugeTLB 子系统支持的页大小取决于具体架构。x86_64 架构支持 2 MB 巨页和 1 GB gigantic 节页。

这个步骤描述了如何在引导时保留 1 GB 页面。

流程

1. **要为 1 GB 页创建一个 HugeTLB 池，请启用 default_hugepagesz=1G 和 hugepagesz=1G 内核选项：**

```
# grubby --update-kernel=ALL --args="default_hugepagesz=1G hugepagesz=1G"
```

2. **在 /usr/lib/systemd/system/ 目录中创建一个名为 hugetlb-gigantic-pages.service 的新文件，并添加以下内容：**

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

3. **在 /usr/lib/systemd/ 目录中创建一个名为 hugetlb-reserve-pages.sh 的新文件，并添加以下内容：**

在添加以下内容时，使用您要保留的 1GB 页面数替换 ***number_of_pages***，并使用您要保留的节点的名称替换 ***node***。

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
   echo "ERROR: $nodes_path does not exist"
   exit 1
fi

reserve_pages()
{
   echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages number_of_pages node
```

例如，要在 ***node0*** 上保留两个 1 GB 页面，在 ***node1*** 上保留 1GB 页面，将 ***number_of_pages*** 替换 2（对于 ***node0***）和 1（对于 ***node1***）：

```
reserve_pages 2 node0
reserve_pages 1 node1
```

4. 创建可执行脚本：

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

5. 启用早期引导保留：

```
# systemctl enable hugetlb-gigantic-pages
```

注意

- 您可以通过在任何时候写入 nr_hugepages，来尝试在运行时保留更多 1 GB 页。但是，为了避免因为内存碎片而导致的失败，在引导过程中的早期保留 1 GB 页。
- 保留静态巨页可有效地减少系统可用的内存量，并阻止它正确利用内存容量。虽然正确大小的保留巨页池可能会对使用它的应用程序有用，但保留巨页的过度或未使用的池最终会降低整体系统性能。当设置保留的巨页池时，请确定系统可以正确地利用其完整的内存容量。

其他资源

- systemd.service (5) 手册页
- /usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt file

38.4. 在运行时保留 HUGETLB 页面的参数

在运行时使用以下参数来影响 HugeTLB 页面的行为。

有关如何在运行时使用这些参数来在运行时配置 HugeTLB 页的更多信息，请参阅 在运行时配置 HugeTLB。

表 38.2. 在运行时配置 HugeTLB 页面的参数

参数	描述	文件名
nr_hugepages	定义分配给指定 NUMA 节点的指定大小的巨页数量。	/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages
nr_overcommit_hugepages	定义系统通过过量使用内存来创建和使用的最大额外巨页数。 在此文件中写入任何非零值表示，如果持久的巨页池耗尽，系统会从内核的正常页面池中获取巨页数量。随着这些多余的巨页变得未使用，它们会被释放并返回到内核的正常页面池。	/proc/sys/vm/nr_overcommit_hugepages

38.5. 在运行时配置 HUGETLB

这个步骤描述了如何在 *node2* 中添加 *20* 个 *2048 kB* 巨页。

要根据您的要求保留页面，请替换：

- *20*，使用您要保留的巨页数，

- ***2048kB***，使用巨页的大小，

- ***node2***，使用您要保留页面的节点。

流程

1. 显示内存统计信息：

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total add
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0      0      0      0
HugePages_Free        0      0      0      0      0
HugePages_Surp        0      0      0      0      0
```

2. 将指定大小的巨页数量添加到节点：

```
# echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
```

验证步骤

- 确保添加了巨页数量：

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0     40      0     40
HugePages_Free        0      0     40      0     40
HugePages_Surp        0      0      0      0      0
```

其他资源

- **numastat (8)** 手册页

38.6. 启用透明巨页

在 **Red Hat Enterprise Linux 9** 中默认启用 **THP**。但是，您可以启用或禁用 **THP**。

这个步骤描述了如何启用 THP。

流程

1. 检查 THP 的当前状态：

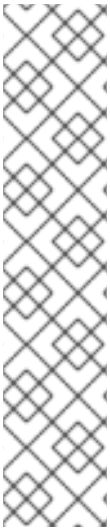
```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. 启用 THP：

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3. 要防止应用程序分配了比必要的内存资源更多的内存资源，禁用系统范围的透明巨页，并仅通过 **madvise** 明确请求来为应用程序启用它们：

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```



注意

有时，为短期分配提供低延迟的优先级比立即实现长时间分配的性能要高。在这种情况下，您可以在启用 THP 时禁用直接压缩。

直接压缩是在巨页分配过程中同步的内存压缩。禁用直接压缩功能无法保证保存内存，但可能会降低频繁页面错误期间延迟更高的风险。请注意，如果工作负载从 THP 有很大的好处，则性能会降低。禁用直接压缩：

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

其他资源

- **madvise (2)** 手册页
- 禁用透明巨页。

38.7. 禁用透明巨页

在 Red Hat Enterprise Linux 9 中默认启用 THP。但是，您可以启用或禁用 THP。

这个步骤描述了如何禁用 THP。

流程

1. 检查 THP 的当前状态：

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. 禁用 THP：

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

38.8. 对翻译的缓冲大小的影响

从页表中读取地址映射非常耗时且需要消耗大量资源，因此 CPU 会对最近使用的地址进行缓存，这称为 Translation Lookaside Buffer (TLB)。但是，默认的 TLB 只能缓存特定数量的地址映射。

如果请求的地址映射不在 TLB 中，称为 TLB miss（TLB未命中），系统仍然需要读取页表以确定到虚拟地址映射的物理。由于应用程序内存要求和用于缓存地址映射的页面大小的关系，具有大内存要求的应用程序可能会比内存要求小的应用程序造成性能下降。因此，务必要尽可能避免 TLB 丢失。

HugeTLB 和 Transparent Huge Page 功能可让应用程序使用大于 4 KB 的页面。这允许存储在 TLB 中的地址引用更多内存，这可以降低 TLB 未命中的情况并改进应用程序性能。

第 39 章 SYSTEMTAP 入门

作为系统管理员，您可以使用 SystemTap 来识别正在运行的 RHEL 系统上错误或性能问题的根本原因。

作为应用程序开发人员，您可以使用 SystemTap 来监控应用程序在 RHEL 系统中的行为。

39.1. SYSTEMTAP 的目的

SystemTap 是跟踪和探测工具，可用于详细研究和监视操作系统（特别是内核）的活动。SystemTap 提供与 netstat、ps、top 和 iostat 等工具输出相似的信息。但是，SystemTap 提供了更多用于收集的信息的过滤和分析选项。在 SystemTap 脚本中，您可以指定 SystemTap 收集的信息。

SystemTap 旨在通过为用户提供基础架构来跟踪内核活动并将此功能与两个属性相结合来补充现有 Linux 监控工具套件：

灵活性

SystemTap 框架允许您开发简单的脚本，以调查和监控内核空间中的各种内核功能、系统调用和其他事件。因此，SystemTap 并不是一个工具，它是一个系统，您可以自行开发特定于内核的知识和监控工具。

易用性

SystemTap 可让您监控内核的活动，而无需重新编译内核或重启系统。

39.2. 安装 SYSTEMTAP

要开始使用 SystemTap，请安装所需的软件包。要在安装多个内核的内核上使用 SystemTap，为 *每个*内核版本安装对应的内核软件包。

先决条件

- 您已启用了 debug 软件仓库，如启用 debug 和源存储库中所述。

流程

1. 安装所需的 SystemTap 软件包：

```
# dnf install systemtap
```

2. **安装所需的内核软件包：**

   a. **使用 stap-prep ：**

   ```
   # stap-prep
   ```

   b. **如果 stap-prep 无法正常工作，请手动安装所需的内核软件包：**

   ```
   # dnf install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-$(uname -r) kernel-devel-$(uname -r)
   ```

   $ (uname -i) 会自动替换为您系统的硬件平台，$ (uname -r) 会自动替换为正在运行的内核的版本。

验证步骤

- **如果当前使用 SystemTap 探测内核，请检查您的安装是否成功：**

```
# stap -v -e 'probe kernel.function("vfs_read") {printf("read performed\n"); exit()}'
```

成功 SystemTap 部署会生成类似如下的输出：

```
Pass 1: parsed user script and 45 library script(s) in 340usr/0sys/358real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 0 embed(s), 0 global(s) in
290usr/260sys/568real ms.
Pass 3: translated to C into
"/tmp/stapiArgLX/stap_e5886fa50499994e6a87aacdc43cd392_399.c" in
490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aacdc43cd392_399.ko" in
3310usr/430sys/3714real ms.
Pass 5: starting run. 1
read performed 2
Pass 5: run completed in 10usr/40sys/73real ms. 3
```

输出的最后三行（以 Pass 5 开始）代表：

1

SystemTap 已成功创建了检测机制，以探测内核并运行检测程序。

2

SystemTap 检测到指定的事件（本例中为 VFS 读取）。

3

SystemTap 执行有效的处理程序（打印的文本，然后关闭它且无错误）。

39.3. 运行 SYSTEMTAP 的权限

运行 SystemTap 脚本需要提升系统特权，但在某些实例上，非特权用户可能需要在其计算机上运行 SystemTap 检测。

要允许用户在无需 root 访问权限的情况下运行 SystemTap，向这两个用户组添加用户：

stapdev

此组的成员可以使用 stap 运行 SystemTap 脚本或 staprun 来运行 SystemTap 检测模块。

运行 stap 涉及将 SystemTap 脚本编译到内核模块中，并将其加载到内核中。这要求系统升级的特权，这被授予 stapdev 成员。不幸的是，此类特权还会向 stapdev 成员授予有效的 root 访问权限。因此，仅向可信任的用户授予 stapdev 组成员资格。

stapusr

这个组的成员只能使用 staprun 来运行 SystemTap 检测模块。另外，它们只能从 /lib/modules/*kernel_version*/systemtap/ 目录中运行这些模块。该目录必须仅归 root 用户所有，并且只能由 root 用户写入。

39.4. 运行 SYSTEMTAP 脚本

您可以从标准输入或从文件运行 SystemTap 脚本。

SystemTap 安装附带的示例脚本可在 SystemTap 脚本或 /usr/share/systemtap/examples 目录中找到。https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance

/getting-started-with-systemtap_monitoring-and-managing-system-status-and-performance#useful-examples-of-systemtap-scripts

先决条件

1. SystemTap 和关联的内核软件包已安装，如安装 Systemtap 所述。
2. 要以普通用户身份运行 SystemTap 脚本，请将该用户添加到 SystemTap 组：

```
# usermod --append --groups
stapdev,stapusr user-name
```

流程

- 运行 SystemTap 脚本：
  - 从标准输入中：

```
# stap -e "probe timer.s(1) {exit()}"
```

此命令指示 stap -e 在标准输入中运行括号中的脚本。

  - 从文件中：

```
# stap file_name.stp
```

39.5. SYSTEMTAP 脚本的有用示例

SystemTap 安装附带的示例脚本示例可在 /usr/share/systemtap/examples 目录中找到。

您可以使用 stap 命令来执行不同的 SystemTap 脚本：

追踪功能调用

您可以使用 para-callgraph.stp SystemTap 脚本跟踪函数调用和函数返回。

```
# stap para-callgraph.stp argument1 argument2
```

脚本采用两个命令行参数：您要跟踪的 entry/exit 的功能名称。可选的触发器功能，用于在每个线程上启用或禁用追踪。只要触发器功能还没有退出，每个线程中的追踪将继续。

监控轮询应用程序

您可以使用 timeout.stp SystemTap 脚本来识别和监控哪些应用程序正在轮询。了解这一点，您可以跟踪不必要的或过度的轮询，这有助于确定 CPU 使用量和节能方面的改进。

```
# stap timeout.stp
```

此脚本跟踪每个应用使用轮询的次数，选择,e poll ,itimer,futex,nanosleep 和 Signal 系统调用

跟踪每个进程的系统调用卷

您可以使用 syscalls_by_proc.stp SystemTap 脚本来查看哪些进程正在执行最高系统调用的卷。它显示执行最多系统调用的 20 个进程。

```
# stap syscalls_by_proc.stp
```

在网络套接字代码中追踪调用的功能

您可以使用 socket-trace.stp 示例 SystemTap 脚本跟踪从内核的 net/socket.c 文件中调用的功能。这有助于您识别每个进程在内核级别上与网络交互的方式会详细介绍。

```
# stap socket-trace.stp
```

跟踪每个文件的 I/O 时间

您可以使用 iotime.stp SystemTap 脚本监控每个进程从任何文件读取或写入到任何文件所需的时间。这有助于您确定系统上载入哪些文件。

```
# stap iotime.stp
```

跟踪 IRQ 和其他进程从任务中窃取周期

您可以使用 cycle_thief.stp SystemTap 脚本来跟踪任务运行的时间量及其没有运行的时间量。这有助于您识别任务中控制哪些进程。

```
# stap cycle_thief.stp -x pid
```

注意

您可以在 /usr/share/systemtap/examples/index.html 文件中找到有关 SystemTap 脚本的更多示例和信息。在 Web 浏览器中打开它，以查看所有可用脚本及其描述的列表。

其他资源

- /usr/share/systemtap/examples 目录

第 40 章 SYSTEMTAP 交叉检测

SystemTap 的交叉检测是从一个系统中的 SystemTap 脚本创建 SystemTap 检测模块，以在未完全部署 SystemTap 的其他系统上使用。

40.1. SYSTEMTAP 交叉检测

运行 SystemTap 脚本时，将从该脚本构建内核模块。然后，SystemTap 会将模块加载到内核中。

通常，SystemTap 脚本只能在部署了 SystemTap 的系统中运行。要在 10 个系统上运行 SystemTap，需要将 SystemTap 部署到所有这些系统上。在某些情况下，这可能并不可行。例如，企业策略可能会禁止您安装提供有关特定机器的编译器或调试信息的软件包，这会阻止 SystemTap 的部署。

要临时解决这个问题，请使用 *交叉检测*。交叉检测（Cross-instrumentation）是从系统中的 SystemTap 脚本生成 SystemTap 检测模块以在另一个系统上生成 SystemTap 检测模块的过程。这个过程具有以下优点：

- 可以在单一主机上安装各种机器的内核信息软件包。

重要

内核打包错误可能会阻止安装。在这种情况下，*host system* 和 *target system* 的 kernel-debuginfo 和 kernel-devel 软件包必须匹配。如果发生错误，请向 https://bugzilla.redhat.com/ 报告这个错误。

- 每个*目标机器*都需要一个软件包才能使用生成的 SystemTap 检测模块： systemtap-runtime。

重要

*主机系统*必须与*目标系统*具有相同的构架，并运行相同的 Linux 发行版本，*检测模块*才能正常工作。

术语

检测模块

SystemTap 脚本构建的内核模块; SystemTap 模块在*主机系统*上构建，并将在*目标系统*的*目标内核*中载入。

主机系统

编译检测模块（来自 SystemTap 脚本）的系统，以便在*目标系统*上加载。

目标系统

正在构建*检测模块*的系统（来自 SystemTap 脚本）。

目标内核

*目标系统*的内核。这是载入并运行*检测模块*的内核。

40.2. 初始化 SYSTEMTAP 的交叉检测

初始化 SystemTap 的交叉检测，以从一个系统上的 SystemTap 脚本构建 SystemTap 检测模块，并在另一个系统上使用未完全部署 SystemTap 的系统上构建 SystemTap 检测模块。

先决条件

- 在*主机系统*上安装 SystemTap，如安装 Systemtap 所述。
- systemtap-runtime 软件包安装在每个 *目标系统*中 ：

```
# dnf install systemtap-runtime
```

- *主机系统*和*目标系统*都是相同的架构。
- *主机系统*和*目标系统*都使用相同的 Red Hat Enterprise Linux 主版本（如 Red Hat Enterprise Linux 9）。

重要

内核打包错误可能会阻止在一个系统中安装多个 **kernel-debuginfo** 和 **kernel-devel** 软件包。在这种情况下，*主机系统*和*目标系统*的次版本必须匹配。如果发生错误，将其报告为 **https://bugzilla.redhat.com/**。

流程

1. 确定在每个*目标系统*中运行的内核：

```
$ uname -r
```

为每个*目标系统*重复此步骤。

2. 在*主机系统*中，根据**安装Systemtap** 中描述的方法，为每个*目标系统*安装*目标内核*和相关的软件包。

3. 在*主机系统*中构建检测模块，将这个模块复制到*目标系统*中并在其中运行：

   a. 使用远程实现：

```
# stap --remote target_system script
```

   这个命令在*目标系统*中远程实施指定的脚本。您必须确保能够通过 SSH 从*主机系统*连接到*目标系统*。

   b. 手动：

      i. 在*主机系统*中构建检测模块：

```
# stap -r kernel_version script -m module_name -p 4
```

      此处 ***kernel_version*** 是指在第 1 步中确定的*目标内核*版本，***script*** 是要转换为*检测模块*的脚本，***module_name*** 是*检测模块*的名称。**p4** 选项告知 SystemTap 不要加载并运行已编译的模块。

ii. 编译*检测模块*后，将其复制到目标系统并使用以下命令载入它：

```
# staprun module_name.ko
```