



# Red Hat

## Red Hat Enterprise Linux 8

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



## 优化系统吞吐量、延迟和电源消耗

## Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

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

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

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

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

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

在不同情况下，监控和优化 Red Hat Enterprise Linux 8 的吞吐量、延迟和功耗。

## Table of Contents

对红帽文档提供反馈 .....	8
<b>第 1 章 性能监控选项概述 .....</b>	<b>9</b>
<b>第 2 章 TUNED 入门 .....</b>	<b>11</b>
2.1. TUNED 的目的 .....	11
2.2. 调优配置文件 .....	11
2.3. 默认 TUNED 配置文件 .....	11
2.4. 合并的 TUNED 配置文件 .....	12
2.5. TUNED 配置文件的位置 .....	12
2.6. RHEL 提供的调优配置文件 .....	13
2.7. TUNED CPU-PARTITIONING 配置文件 .....	15
2.8. 使用 TUNED CPU-PARTITIONING 配置文件进行低延迟调整 .....	16
2.9. 自定义 CPU-PARTITIONING TUNED 配置集 .....	17
2.10. RHEL 提供的实时 TUNED 配置文件 .....	18
2.11. TUNED 中的静态和动态性能优化 .....	18
2.12. TUNED NO-DAEMON (非守护进程) 模式 .....	19
2.13. 安装并启用 TUNED .....	19
2.14. 列出可用的 TUNED 配置文件 .....	20
2.15. 设置 TUNED 配置文件 .....	21
2.16. 使用 TUNED D-BUS 接口 .....	22
2.17. 禁用 TUNED .....	23
<b>第 3 章 自定义 TUNED 配置文件 .....</b>	<b>25</b>
3.1. 调优配置文件 .....	25
3.2. 默认 TUNED 配置文件 .....	25
3.3. 合并的 TUNED 配置文件 .....	26
3.4. TUNED 配置文件的位置 .....	26
3.5. TUNED 配置文件之间的继承 .....	26
3.6. TUNED 中的静态和动态性能优化 .....	27
3.7. TUNED 插件 .....	28
3.8. 可用的 TUNED 插件 .....	29
3.9. SCHEDULER TUNED 插件的功能 .....	34
3.10. TUNED 配置集中的变量 .....	38
3.11. TUNED 配置集中的内置功能 .....	39
3.12. TUNED 配置集中的内置功能 .....	40
3.13. 创建新的 TUNED 配置集 .....	41
3.14. 修改现有 TUNED 配置集 .....	42
3.15. 使用 TUNED 设置磁盘调度程序 .....	43
<b>第 4 章 使用 TUNA 接口检查系统 .....</b>	<b>46</b>
4.1. 安装 TUNA 工具 .....	46
4.2. 使用 TUNA 工具查看系统状态 .....	46
4.3. 使用 TUNA 工具调整 CPU .....	47
4.4. 使用 TUNA 工具调整 IRQ .....	49
<b>第 5 章 使用 RHEL 系统角色，使用 PCP 配置性能监控 .....</b>	<b>51</b>
5.1. 使用 METRICS RHEL 系统角色配置 PERFORMANCE CO-PILOT .....	51
5.2. 使用 METRICS RHEL 系统角色配置带有身份验证的 PERFORMANCE CO-PILOT .....	52
5.3. 使用 METRICS RHEL 系统角色设置 GRAFANA，来使用 PERFORMANCE CO-PILOT 监控多个主机 .....	54
<b>第 6 章 设置 PCP .....</b>	<b>57</b>
6.1. PCP 概述 .....	57

6.2. 安装并启用 PCP	57
6.3. 部署最小 PCP 设置	58
6.4. 系统服务和使用 PCP 分发的工具	59
6.5. PCP 部署架构	62
6.6. 推荐的部署架构	65
6.7. 大小考虑因素	65
6.8. PCP 扩展的配置选项	66
6.9. 示例：分析集中式日志记录部署	66
6.10. 示例：分析联合设置部署	67
6.11. 对高内存使用量进行故障排除	68
<b>第 7 章 使用 PMLOGGER 记录性能数据</b>	<b>70</b>
7.1. 使用 PMLOGCONF 修改 PMLOGGER 配置文件	70
7.2. 手动编辑 PMLOGGER 配置文件	70
7.3. 启用 PMLOGGER 服务	71
7.4. 为指标集合设置客户端系统	72
7.5. 设置中央服务器以收集数据	73
7.6. SYSTEMD 单元和 PMLOGGER	74
7.7. 使用 PMREP 重现 PCP 日志存档	76
<b>第 8 章 使用 PERFORMANCE CO-PILOT 监控性能</b>	<b>78</b>
8.1. 使用 PMDA-POSTFIX 监控 POSTFIX	78
8.2. 使用 PCP CHARTS 应用程序可视化追踪 PCP 日志存档	79
8.3. 使用 PCP 从 SQL 服务器收集数据	81
<b>第 9 章 使用 PCP 对 XFS 的性能分析</b>	<b>83</b>
9.1. 手动安装 XFS PMDA	83
9.2. 使用 PMINFO 检查 XFS 性能指标	84
9.3. 使用 PMSTORE 重置 XFS 性能指标	85
9.4. XFS 的 PCP 指标组	86
9.5. 每个设备 PCP 指标组用于 XFS	87
<b>第 10 章 设置 PCP 指标的图形表示</b>	<b>89</b>
10.1. 使用 PCP-ZEROCONF 设置 PCP	89
10.2. 设置 GRAFANA-SERVER	89
10.3. 访问 GRAFANA WEB UI	90
10.4. 配置 PCP REDIS	92
10.5. 在 PCP REDIS 数据源中创建面板和警报	93
10.6. 为警报添加通知频道	95
10.7. 在 PCP 组件间设置身份验证	96
10.8. 安装 PCP BPFTRACE	97
10.9. 查看 PCP BPFTRACE SYSTEM ANALYSIS 仪表盘	98
10.10. 安装 PCP 向量	99
10.11. 查看 PCP 向量清单	100
10.12. 在 GRAFANA 中使用 HEATMAPS	101
10.13. GRAFANA 问题故障排除	103
<b>第 11 章 使用 WEB 控制台优化系统性能</b>	<b>105</b>
11.1. WEB 控制台中的性能调优选项	105
11.2. 在 WEB 控制台中设置性能配置文件	105
11.3. 使用 WEB 控制台监控本地系统的性能	106
11.4. 使用 WEB 控制台和 GRAFANA 监控多个系统的性能	108
<b>第 12 章 设置磁盘调度程序</b>	<b>110</b>
12.1. 可用磁盘调度程序	110

12.2. 不同用例的磁盘调度程序	110
12.3. 默认磁盘调度程序	111
12.4. 确定活跃磁盘调度程序	111
12.5. 使用 TUNED 设置磁盘调度程序	111
12.6. 使用 UDEV 规则设置磁盘调度程序	113
12.7. 为特定磁盘临时设置调度程序	114
<b>第 13 章 调整 SAMBA 服务器的性能</b>	<b>116</b>
13.1. 设置 SMB 协议版本	116
13.2. 与包含大量文件的目录调整共享	116
13.3. 可能会对性能造成负面影响的设置	117
<b>第 14 章 优化虚拟机性能</b>	<b>118</b>
14.1. 影响虚拟机性能的因素	118
14.2. 使用 TUNED 优化虚拟机性能	118
14.3. 针对特定工作负载的虚拟机性能优化	120
14.4. 配置虚拟机内存	120
14.5. 优化虚拟机 I/O 性能	125
14.6. 优化虚拟机 CPU 性能	131
14.7. 优化虚拟机网络性能	140
14.8. 虚拟机性能监控工具	141
14.9. 其它资源	143
<b>第 15 章 电源管理的重要性</b>	<b>144</b>
15.1. 电源管理基础	144
15.2. 审计和分析概述	145
15.3. 用于审计的工具	145
<b>第 16 章 使用 POWERTOP 管理能耗</b>	<b>149</b>
16.1. POWERTOP 的目的	149
16.2. 使用 POWERTOP	149
16.3. POWERTOP 统计	150
16.4. 为什么 POWERTOP 不会在一些实例中显示 FREQUENCY STATS 值	152
16.5. 生成 HTML 输出	152
16.6. 优化功耗	152
<b>第 17 章 调整 CPU 频率以优化能源消耗</b>	<b>155</b>
17.1. 支持的 CPUPOWER 工具命令	155
17.2. CPU 空闲状态	156
17.3. CPUFREQ 概述	157
<b>第 18 章 PERF 入门</b>	<b>161</b>
18.1. PERF 简介	161
18.2. 安装 PERF	161
18.3. 常见 PERF 命令	161
<b>第 19 章 使用 PERF TOP 实时分析 CPU 使用量</b>	<b>162</b>
19.1. PERF TOP 的目的	162
19.2. 使用 PERF TOP 分析 CPU 使用量	162
19.3. PERF OUTPUT 输出的解释	163
19.4. 为什么 PERF 会显示一些功能名称作为原始功能地址	163
19.5. 启用调试和源存储库	163
19.6. 使用 GDB 获取应用程序或库的 DEBUGINFO 软件包	164
<b>第 20 章 使用 PERF STAT 在进程执行过程中计算事件</b>	<b>166</b>

20.1. PERF STAT 的目的	166
20.2. 使用 PERF STAT 计数事件	166
20.3. PERF STAT 输出的解释	167
20.4. 将 PERF STAT 附加到正在运行的进程	167
<b>第 21 章 使用 PERF 记录和分析性能配置文件</b>	<b>169</b>
21.1. PERF RECORD 的目的	169
21.2. 在没有 ROOT 访问权限的情况下记录性能配置文件	169
21.3. 使用 ROOT 访问权限记录性能配置文件	169
21.4. 以针对每个 CPU 的模式记录性能档案	170
21.5. 使用 PERF RECORD 捕获调用图形数据	170
21.6. 使用 PERF REPORT 分析 PERF.DATA	171
21.7. PERF 报告输出的解释	172
21.8. 生成可在不同设备上读取的 PERF.DATA 文件	172
21.9. 分析在不同设备中创建的 PERF.DATA 文件	173
21.10. 为什么 PERF 会显示一些功能名称作为原始功能地址	173
21.11. 启用调试和源存储库	174
21.12. 使用 GDB 获取应用程序或库的 DEBUGINFO 软件包	174
<b>第 22 章 使用 PERF 调查繁忙的 CPU</b>	<b>176</b>
22.1. 显示使用 PERF STAT 时会计算哪些 CPU 事件	176
22.2. 显示使用 PERF REPORT 要使用哪些 CPU 样本	176
22.3. 使用 PERF TOP 在性能分析期间显示特定 CPU	177
22.4. 使用 PERF RECORD 和 PERF REPORT 监控特定 CPU	177
<b>第 23 章 使用 PERF 监控应用程序性能</b>	<b>179</b>
23.1. 将 PERF 记录附加到正在运行的进程	179
23.2. 使用 PERF RECORD 捕获调用图形数据	179
23.3. 使用 PERF REPORT 分析 PERF.DATA	180
<b>第 24 章 使用 PERF 创建 UPROBES</b>	<b>182</b>
24.1. 在功能级别使用 PERF 创建 UPROBES	182
24.2. 在带有 PERF 的函数内创建 UPROBES	182
24.3. 通过 UPROBES 记录的数据 PERF 脚本输出	183
<b>第 25 章 使用 PERF MEM 分析内存访问</b>	<b>184</b>
25.1. PERF MEM 的目的	184
25.2. 使用 PERF MEM 抽样内存访问	184
25.3. PERF MEM 报告输出的解释	186
<b>第 26 章 检测错误共享</b>	<b>187</b>
26.1. PERF C2C 的目的	187
26.2. 使用 PERF C2C 检测缓存行竞争	187
26.3. 视觉化使用 PERF C2C 记录所记录的 PERF.DATA 文件	188
26.4. PERF C2C 报告输出的解释	190
26.5. 使用 PERF C2C 检测错误共享	191
<b>第 27 章 FLAMEGRAPHS 入门</b>	<b>193</b>
27.1. 安装 FLAMEGRAPHS	193
27.2. 在整个系统中创建 FLAMEGRAPHS	193
27.3. 在特定进程中创建 FLAMEGRAPHS	194
27.4. 解释 FLAMEGRAPHS	195
<b>第 28 章 监控使用 PERF 环形缓冲的性能瓶颈</b>	<b>197</b>
28.1. 使用 PERF 环缓冲缓冲和特定于事件的快照	197

28.2. 收集特定数据以监控使用 PERF 环形缓冲的性能瓶颈	197
<b>第 29 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</b>	<b>199</b>
29.1. 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	199
29.2. 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	199
<b>第 30 章 使用 NUMASTAT 分析内存分配</b>	<b>201</b>
30.1. 默认 NUMASTAT 统计	201
30.2. 使用 NUMASTAT 查看内存分配	201
<b>第 31 章 配置操作系统以优化 CPU 使用率</b>	<b>203</b>
31.1. 监控和诊断处理器问题的工具	203
31.2. 系统拓扑类型	203
31.3. 配置内核空循环时间	206
31.4. 中断请求概述	207
<b>第 32 章 调优调度策略</b>	<b>210</b>
32.1. 调度策略的类别	210
32.2. 使用 SCHED_FIFO 的静态优先级调度	210
32.3. 使用 SCHED_RR 循环优先级调度	211
32.4. 使用 SCHED_OTHER 常规调度	211
32.5. 设置调度程序策略	211
32.6. CHRT 命令的策略选项	212
32.7. 在引导过程中更改服务优先级	212
32.8. 优先级映射	214
32.9. TUNED CPU-PARTITIONING 配置文件	214
32.10. 使用 TUNED CPU-PARTITIONING 配置文件进行低延迟调整	215
32.11. 自定义 CPU-PARTITIONING TUNED 配置集	217
<b>第 33 章 影响 I/O 和文件系统性能的因素</b>	<b>218</b>
33.1. 监控和诊断 I/O 和文件系统问题的工具	218
33.2. 用于格式化文件系统的可用调整选项	219
33.3. 可用于挂载文件系统的选项	221
33.4. 丢弃未使用块的类型	221
33.5. 固态磁盘调优注意事项	222
33.6. 通用块设备性能优化参数	222
<b>第 34 章 调优网络性能</b>	<b>224</b>
34.1. 调优网络适配器设置	224
34.2. 调优 IRQ 平衡	228
34.3. 改善网络延迟	230
34.4. 提高大量连续数据流的吞吐量	234
34.5. 为高吞吐量调优 TCP 连接	236
34.6. 调优 UDP 连接	241
34.7. 识别应用程序读套接字缓冲区瓶颈	245
34.8. 调优具有大量传入请求的应用程序	247
34.9. 避免侦听队列锁争用	249
34.10. 调优设备驱动程序和 NIC	253
34.11. 配置网络适配器卸载设置	255
34.12. 调优中断合并设置	257
34.13. TCP 时间戳的好处	260
34.14. 以太网网络的流控制	261
<b>第 35 章 配置操作系统以优化内存访问</b>	<b>262</b>
35.1. 监控和诊断系统内存问题的工具	262

35.2. 系统内存概述	262
35.3. 虚拟内存参数	263
35.4. 文件系统参数	265
35.5. 内核参数	266
35.6. 设置与内存相关的内核参数	266
<b>第 36 章 配置巨页 .....</b>	<b>268</b>
36.1. 可用的巨页功能	268
36.2. 在引导时保留 HUGETLB 页面的参数	268
36.3. 在引导时配置 HUGETLB	269
36.4. 在运行时保留 HUGETLB 页面的参数	271
36.5. 在运行时配置 HUGETLB	271
36.6. 管理透明巨页	272
36.7. 页面大小对转换后备缓冲区大小的影响	276
<b>第 37 章 SYSTEMTAP 入门 .....</b>	<b>278</b>
37.1. SYSTEMTAP 的目的	278
37.2. 安装 SYSTEMTAP	278
37.3. 运行 SYSTEMTAP 的权限	279
37.4. 运行 SYSTEMTAP 脚本	279
<b>第 38 章 SYSTEMTAP 交叉检测 .....</b>	<b>281</b>
38.1. SYSTEMTAP 交叉检测	281
38.2. 初始化 SYSTEMTAP 的交叉检测	281
<b>第 39 章 使用 SYSTEMTAP 监控网络活动 .....</b>	<b>283</b>
39.1. 使用 SYSTEMTAP 分析网络活动	283
39.2. 使用 SYSTEMTAP 在网络套接字代码中追踪调用的功能	284
39.3. 使用 SYSTEMTAP 监控网络数据包丢弃	284
<b>第 40 章 使用 SYSTEMTAP 分析内核活动 .....</b>	<b>286</b>
40.1. 使用 SYSTEMTAP 的计数功能调用	286
40.2. 使用 SYSTEMTAP 的追踪功能调用	287
40.3. 使用 SYSTEMTAP 确定内核和用户空间花费的时间	288
40.4. 使用 SYSTEMTAP 监控轮询应用程序	288
40.5. 与 SYSTEMTAP 跟踪最常用的系统调用	289
40.6. 使用 SYSTEMTAP 跟踪每个进程的系统调用卷	290
<b>第 41 章 使用 SYSTEMTAP 监控磁盘和 I/O 活动 .....</b>	<b>292</b>
41.1. 使用 SYSTEMTAP 总结磁盘读/写流量	292
41.2. 使用 SYSTEMTAP 跟踪每个文件的 I/O 时间	293
41.3. 使用 SYSTEMTAP 跟踪累积的 I/O 信息	293
41.4. 在使用 SYSTEMTAP 的特定设备上监控 I/O 活动	294
41.5. 监控使用 SYSTEMTAP 文件的读取和写入	295
<b>第 42 章 使用 BPF COMPILER COLLECTION 分析系统性能 .....</b>	<b>297</b>
42.1. 安装 BCC-TOOLS 软件包	297
42.2. 使用所选 BCC-TOOLS 进行性能调整	297



## 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. 单击顶部导航栏中的 **Create**。
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您对改进的建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

# 第1章 性能监控选项概述

以下是 Red Hat Enterprise Linux 8 中一些性能监控和配置工具：

- Performance Co-Pilot (**pcp**) 用于监控、可视化、存储和分析系统级性能测量。它允许监控和管理实时数据，以及记录和检索历史数据。
- Red Hat Enterprise Linux 8 提供多个工具，可从命令行用于监控外部运行级别 5 的系统。以下是内置命令行工具：
  - **top** 由 **procps-ng** 软件包提供。它可提供正在运行的系统中的进程的动态视图。它显示各种信息，包括系统摘要和当前由 Linux 内核管理的任务列表。
  - **ps** 由 **procps-ng** 软件包提供。它将捕获一组选定活跃进程组的快照。默认情况下，检查组仅限于当前用户拥有的进程，并与执行 **ps** 命令的终端关联。
  - 虚拟内存统计信息 (**vmstat**) 由 **procps-ng** 软件包提供。它为系统的进程、内存、分页、块输入/输出、中断和 CPU 活动提供即时报告。
  - System activity reporter (**sar**) 由 **sysstat** 软件包提供。它收集并报告当前发生的系统活动信息。
- **perf** 使用硬件性能计数器和内核追踪点来跟踪系统中的其他命令和应用程序的影响。
- **bcc-tools** 用于 BPF Compiler Collection (BCC)。它提供超过 100 个 **eBPF** 脚本来监控内核活动。有关这个工具的每一个脚本的更多信息，请参阅描述如何使用它以及其执行哪些功能的手册页。
- **turbostat** 由 **kernel-tools** 软件包提供。它报告了 Intel 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。
- **iostat** 由 **sysstat** 软件包提供。它监控并报告系统 IO 设备，以帮助管理员决定如何平衡物理磁盘之间的 IO 负载。
- **irqbalance** 在处理器之间分发硬件中断以提升系统性能。
- **ss** 会输出有关套接字的统计信息，允许管理员评估设备性能。红帽建议在 Red Hat Enterprise Linux 8 中使用 **ss** 而不是 **netstat**。
- **numastat** 由 **numactl** 软件包提供。默认情况下，**numastat** 显示每个节点的 NUMA 命中可能会遇到内核内存 allocator 中的系统统计信息。最佳性能由高  **numa\_hit** 值和低  **numa\_miss** 值表示。
- **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、管理以及因此系统性能。
- **SystemTap** 监控和分析操作系统活动，尤其是内核活动。
- **valgrind** 通过以 synthetic CPU 上运行应用程序，并在应用程序执行时检测现有应用程序代码来分析应用程序。然后，它将明确地识别应用程序执行中涉及的用户指定的文件、文件描述符或网络套接字的评论。查找内存泄漏也很有用。
- **pqos** 由 **intel-cmt-cat** 软件包提供。它监控并控制当前 Intel 处理器上的 CPU 缓存和内存带宽。

## 其他资源

- 您系统上的 **pcp, top, ps, vmstat, sar, perf, iostat, irqbalance, ss, numastat, numad, valgrind** 和 **pqos** 手册页
- **/usr/share/doc/** 目录
- [iostat 报告的值"await"的含义是什么？红帽知识库文章](#)
- [使用 Performance Co-Pilot 监控性能](#)

## 第 2 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文件。

### 2.1. 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 **配置文件**，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文件发布，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文件定义的规则，并自定义如何调整特定设备。当您切换到另一个配置文件或取消激活 **TuneD** 时，对之前的配置文件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。

### 2.2. 调优配置文件

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文件。您还可以创建、修改和删除配置文件。

**TuneD** 提供的配置文件被分为以下几个类别：

- 节能配置文件
- 性能提升配置文件

性能提升配置文件包括侧重于以下方面的配置文件：

- 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

配置文件配置的语法

**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

- 您系统上的 **tuned.conf (5)** 手册页

### 2.3. 默认 TUNED 配置文件

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

环境	默认配置文件	目标
计算节点	<b>throughput-performance</b>	最佳吞吐量性能
虚拟机	<b>virtual-guest</b>	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 <b>balance</b> 或 <b>powersave</b> 配置文件。
其他情况	<b>balanced</b>	平衡性能和能源消耗

## 其他资源

- 您系统上的 **tuned.conf (5)** 手册页

## 2.4. 合并的 TUNED 配置文件

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

如果存在冲突，则最后指定的配置文件的设置会优先使用。

### 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** 配置文件针对高吞吐量设置磁盘，但当前通过 **spindown-disk** 配置文件将磁盘旋转设置为低值。

## 其他资源

- 您系统上的 **tuned-adm** 和 **tuned.conf (5)** 手册页

## 2.5. TUNED 配置文件的位置

Tuned 配置文件存储在以下目录中：

**/usr/lib/tuned/**

特定于分发的配置文件存储在目录中。每个配置文件都有自己的目录。该配置文件由名为 **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文件，请将配置文件目录复制到用于自定义配置文件的目录中。如果同一名称有两个配置文件，则使用位于 /etc/tuned/ 中的自定义配置文件。

## 其他资源

- 您系统上的 **tuned.conf (5)** 手册页

## 2.6. RHEL 提供的调优配置文件

以下是在 Red Hat Enterprise Linux 中安装 TuneD 的配置文件列表。



### 注意

更特定产品的或第三方的 TuneD 配置文件也可能会存在。这些配置文件通常由单独的 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文件。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 Aggressive 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，配置文件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** 配置文件相比，**balanced** 配置文件效率更高。

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** 配置文件。

### accelerator-performance

**accelerator-performance** 配置文件包含与 **throughput-performance** 配置文件相同的调整。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** 配置文件。

### network-latency

低延迟网络调整的配置文件。它基于 **latency-performance** 配置文件。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** 配置文件，该配置文件将 **power\_performance\_preference** 和 **scaling\_governor** 属性更改为 **performance** 配置文件。

### hpc-compute

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

### network-throughput

用于吞吐量网络调优的配置文件。它基于 **throughput-performance** 配置文件。此外，它还增加了内核网络缓冲区。

它继承 **latency-performance** 或 **throughput-performance** 配置文件，并将 **energy\_performance\_preference** 和 **scaling\_governor** 属性改为 **performance** 配置文件。

### virtual-guest

为 Red Hat Enterprise Linux 8 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** 配置文件，该配置文件将 **energy\_performance\_preference** 和 **scaling\_governor** 属性更改为 **performance** 配置文件。

### virtual-host

基于 **throughput-performance** 配置文件（除其他任务）为虚拟主机设计的配置文件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** 配置文件，该配置文件将 **energy\_performance\_preference** 和 **scaling\_governor** 属性更改为 **performance** 配置文件。

### oracle

根据 **throughput-performance** 配置文件，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。这个配置文件由 **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文件。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文件。这应该使串行控制台更快响应。此配置文件用作其他配置文件的覆盖。例如：

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

### mssql

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

### 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文件。此配置文件用作其他配置文件的覆盖。例如：

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

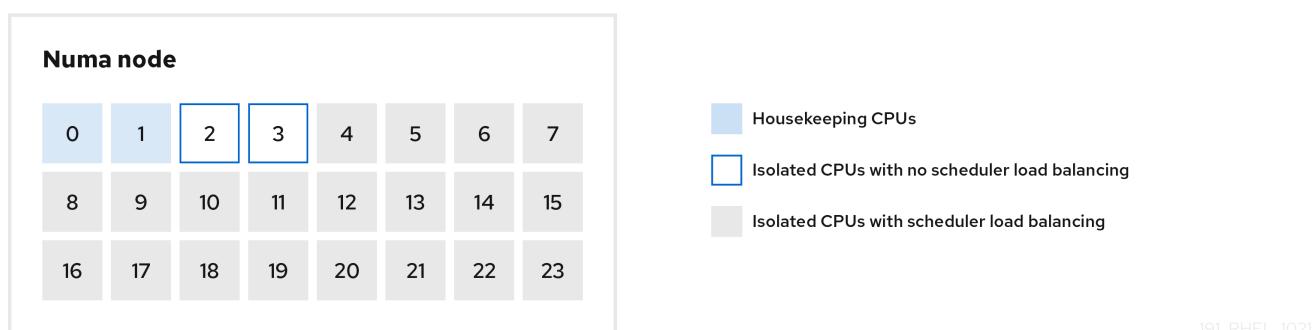
## 2.7. TUNED CPU-PARTITIONING 配置文件

要为对延迟敏感的工作负载调整 Red Hat Enterprise Linux 8，红帽建议使用 **cpu-partitioning** TuneD 配置集。

在 Red Hat Enterprise Linux 8 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 8 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，此配置文件可轻松自定义。

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

图 2.1. **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** 配置文件：

### 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** 配置文件，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** 配置文件，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

- 您系统上的 **tuned-profiles-cpu-partitioning (7)** 手册页

## 2.8. 使用 TUNED CPU-PARTITIONING 配置文件进行低延迟调整

这个步骤描述了如何使用 TuneD 的 **cpu-partitioning** 配置文件为低延迟调整系统。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 **yum install tuned-profiles-cpu-partitioning** 命令安装 **cpu-partitioning** TuneD 配置集。

### 流程

1. 使用以下更改编辑 **/etc/tuned/cpu-partitioning-variables.conf** 文件：

- a. 注释掉 **isolated\_cores=\${f:calc\_isolated\_cores:1}** 行：

```
# isolated_cores=${f:calc_isolated_cores:1}
```

- b. 为隔离的 CPUS 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 配置文件：

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

3. 重启系统。

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。

## 验证

- 验证隔离的 CPU 是否没有在 **Cpus\_allowed\_list** 字段中反映：

```
# cat /proc/self/status | grep Cpu
Cpus_allowed: 003
Cpus_allowed_list: 0-1
```

- 要查看所有进程的亲和性，请输入：

```
# ps -ae -o pid= | xargs -n 1 taskset -cp

pid 1's current affinity list: 0,1
pid 2's current affinity list: 0,1
pid 3's current affinity list: 0,1
pid 4's current affinity list: 0-5
pid 5's current affinity list: 0,1
pid 6's current affinity list: 0,1
pid 7's current affinity list: 0,1
pid 9's current affinity list: 0
...
...
```



### 注意

TuneD 无法更改某些进程的亲和性，主要是内核进程。在本例中，PID 4 和 9 的进程保持不变。

## 其他资源

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

## 2.9. 自定义 CPU-PARTITIONING TUNED 配置集

您可以扩展 TuneD 配置文件，以进行额外的性能优化更改。

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

## 流程

- 创建 **/etc/tuned/my\_profile** 目录：

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

- 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. 使用新配置文件：

```
# 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* 配置文件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

- 您系统上的 **tuned-profiles-cpu-partitioning (7)** 手册页

## 2.10. RHEL 提供的实时 TUNED 配置文件

实时配置文件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，配置文件可从额外的软件仓库获得。

可用的实时配置文件如下：

#### **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。

## 2.11. TUNED 中的静态和动态性能优化

在决定对给定情况或目的使用哪种系统调优（静态和 动态）时，了解 **TuneD** 应用的两类系统调优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，因此在性能配置文件中禁用。可以在 **Tuned** 配置文件中启用或禁用各个插件的动态性能优化。

### 例 2.2. 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。

## 2.12. 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
```

## 2.13. 安装并启用 TUNED

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

### 流程

1. 安装 **Tuned** 软件包：

```
# yum install tuned
```

2. 启用并启动 **Tuned** 服务：

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

3. 可选：为实时系统安装 **Tuned** 配置文件：

对于启用了 **rhel-8** 存储库的实时系统的 **Tuned** 配置文件。

```
# subscription-manager repos --enable=rhel-8-for-x86_64-nfv-beta-rpms
```

安装它。

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

#### 4. 验证 TuneD 配置文件是否活跃并应用：

```
$ tuned-adm active
```

```
Current active profile: throughput-performance
```



#### 注意

活跃的配置文件 TuneD 根据您的机器类型和系统自动预设不同。

```
$ tuned-adm verify
```

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

## 2.14. 列出可用的 TUNED 配置文件

此流程列出了系统中当前可用的所有 TuneD 配置文件。

### 步骤

- 要列出系统中的所有可用 TuneD 配置文件，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文件，请使用：

```
$ tuned-adm active
```

```
Current active profile: throughput-performance
```

## 其他资源

- 您系统上 **tuned-adm (8)** 手册页

## 2.15. 设置 TUNED 配置文件

此流程激活系统中的所选 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

1. 可选：您可以让 TuneD 为您的系统推荐最合适的配置文件：

```
# tuned-adm recommend
throughput-performance
```

2. 激活配置文件：

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

另外，您可以激活多个配置文件的组合：

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

### 例 2.3. 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 配置文件：

```
# tuned-adm active
Current active profile: selected-profile
```

4. 重启系统：

```
# reboot
```

### 验证

- 验证 TuneD 配置文件是否活跃并应用：

```
$ 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

- 您系统上 **tuned-adm (8)** 手册页

## 2.16. 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。

### 2.16.1. 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
```

输出应类似于如下：

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。

## 2.16.2. 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"
```

## 2.17. 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

- 您系统上 **tuned-adm (8)** 手册页

# 第 3 章 自定义 TUNED 配置文件

您可以创建或修改 TuneD 配置文件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。

### 3.1. 调优配置文件

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文件。您还可以创建、修改和删除配置文件。

TuneD 提供的配置文件被分为以下几个类别：

- 节能配置文件
- 性能提升配置文件

性能提升配置文件包括侧重于以下方面的配置文件：

- 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

#### 配置文件配置的语法

**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

- 您系统上的 **tuned.conf (5)** 手册页

### 3.2. 默认 TUNED 配置文件

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

环境	默认配置文件	目标
计算节点	<b>throughput-performance</b>	最佳吞吐量性能
虚拟机	<b>virtual-guest</b>	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 <b>balance</b> 或 <b>powersave</b> 配置文件。
其他情况	<b>balanced</b>	平衡性能和能源消耗

## 其他资源

- 您系统上的 **tuned.conf (5)** 手册页

### 3.3. 合并的 TUNED 配置文件

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

如果存在冲突，则最后指定的配置文件的设置会优先使用。

#### 例 3.1. 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** 配置文件针对高吞吐量设置磁盘，但当前通过 **spindown-disk** 配置文件将磁盘旋转设置为低值。

## 其他资源

- 您系统上的 **tuned-adm** 和 **tuned.conf (5)** 手册页

### 3.4. TUNED 配置文件的位置

TuneD 配置文件存储在以下目录中：

#### /usr/lib/tuned/

特定于分发的配置文件存储在目录中。每个配置文件都有自己的目录。该配置文件由名为 **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文件，请将配置文件目录复制到用于自定义配置文件的目录中。如果同一名称有两个配置文件，则使用位于 **/etc/tuned/** 中的自定义配置文件。

## 其他资源

- 您系统上的 **tuned.conf (5)** 手册页

### 3.5. TUNED 配置文件之间的继承

TuneD 配置文件可以基于其他配置文件，仅修改其父级配置文件的某些方面。

TuneD 配置文件的 **[main]** 部分可以识别 **include** 选项：

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

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

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

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

### 例 3.2. 基于均衡的节能配置文件

以下是一个可扩展 **balanced** 配置文件的自定义配置文件，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

- 您系统上的 **tuned.conf (5)** 手册页

## 3.6. TUNED 中的静态和动态性能优化

在决定对给定情况或目的使用哪种系统调优（静态和 动态）时，了解 TuneD 应用的两类系统调优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，因此在性能配置文件中禁用。可以在 TuneD 配置文件中启用或禁用各个插件的动态性能优化。

### 例 3.3. 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。

## 3.7. TUNED 插件

插件是 **TuneD** 配置文件中的模块，**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** 配置文件填充的多个参数。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** 配置文件中的插件语法

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 3.4. 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 3.5. 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

- 您系统上的 **tuned.conf (5)** 手册页

## 3.8. 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 3.6. 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](http://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 引导装载程序。

自定义的 GRUB 配置文件的非标准位置可以由 `grub2_cfg_file` 选项指定。

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 3.7. 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]**。

支持的处理服务的命令有：

- **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 系统忽略此指令。**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，如果这些目录为空，**service** 插件会删除它们。



### 注意

**service** 插件仅在具有非**systemd** init 系统的当前运行级别上操作。

### 例 3.8. 启动并启用带有覆盖文件的 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

提供用于调度优先级的调优、CPU 核隔离和进程、线程以及 IRQ 关联性的各种选项。

有关可用不同选项的具体内容，请参阅 [scheduler TuneD 插件的功能](#)。

### 3.9. 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、线程以及 IRQ 关联性。

#### 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 IRQs 设置 **default\_smp\_affinity** 参数。

为 **ps\_whitelist** 选项匹配的所有进程和线程调整了 CPU 关联性掩码，这取决于 **sched\_setaffinity()** 系统调用的成功。**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。

支持 **ps\_whitelist** 和 **ps\_blacklist** 选项的以 ; 分隔的多个正则表达式。转义的分号 \; 按字面处理。

#### 例 3.9. 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，代表系统上所有不活跃中断请求(IRQ)源的默认目标 CPU 核。IRQ 被激活或分配后，**/proc/irq/default\_smp\_affinity** 文件中的值决定了 IRQ 的关联性位掩码。

**default\_irq\_smp\_affinity** 参数控制 TuneD 写入 **/proc/irq/default\_smp\_affinity** 文件的内容。**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** 参数的倒置计算非隔离的核。

然后，非隔离内核和以前的 **/proc/irq/default\_smp\_affinity** 文件的内容的交集被写入 **/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

采用单数字的形式，如 1、以逗号分隔的列表，如 1、3 或范围，如 3-5。

解包 CPU 列表，并将其直接写入到 `/proc/irq/default_smp_affinity` 文件中。

### 例 3.10. 使用显式 CPU 列表设置默认 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

要为一组进程或线程调整调度策略、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。规则按优先级排序。这是继承性所需要的，以便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** 必须是以下内容之一：

**f**

用于先入先出(FIFO)

**b**

用于批处理

**r**

用于循环

**o**

用于其他

\*

用于不更改

**affinity** 是十六进制的 CPU 关联性。对于无变化，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 3.11. 设置调度策略和优先级

以下示例将调度策略和优先级设置为内核线程和 watchdog：

```
[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，意味着由 **fork()** 系统调用创建的子进程被处理了。



### 注意

处理 **perf** 事件可能会造成大量 CPU 开销。

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

### 例 3.12. 禁用动态调度程序功能

以下示例禁用了动态调度程序功能，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。在重负载下，此缓冲区可能会溢出，因此插件可能会开始丢失事件，并且没有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 **perf\_mmap\_pages** 参数不是手动设置的，则使用默认值 128。

### 使用 cgroups 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** 期望挂载它的位置。如果未设置，则期望为 **/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**，则必须由其他方法预设置 **cgroups**。

如果 **cgroup\_mount\_point\_init** 选项被设置为 **1**，则 **TuneD** 会创建并删除 **cgroup** 挂载点。它暗示 **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，则经典 **cpuset** 关联性使用 **sched\_setaffinity()** 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，但您必须以正确顺序指定层次结构。**TuneD** 不会在此处执行任何健全性检查，例外是它强制 **cgroup** 位于 **cgroup\_mount\_point** 选项指定的位置。

以 **group** 开头的调度程序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。您也可以使用不是由 **cgroup.** 选项定义的 cgroups，如上面所述。例如，**cgroups** 不由 **TuneD** 管理的。

通过将所有句点(.)替换为斜杠(/)来对所有 **cgroup** 名称进行清理。这可防止插件在 **cgroup\_mount\_point** 选项指定的位置之外写入。

### 例 3.13. 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** 关联性设置为 CPU 0 和 2。假定有 4 个 CPU 设置，**isolated\_cores=1** 选项将所有进程和线程移到 CPU 内核 0、2 和 3。由 **ps\_blacklist** 正则表达式指定的进程和线程不会被移动。

```
[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用分号(;)分开。分号表示逻辑"或"运算符。

### 例 3.14. 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 中移走，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 **sysctl** 工具管理的 **/proc/sys/kernel** 目录中的一些 **sched\_** 和 **numa\_balancing\_** 内核运行时参数移到了 **debugfs**, 其通常挂载在 **/sys/kernel/debug** 目录下。TuneD 通过 **scheduler** 插件为以下参数提供了一种抽象机制, 其中, 根据所使用的内核, 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 3.15.** 为迁移决策设置任务的 "cache hot" 值。

在旧内核中, 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
```

在最新内核中, 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。

### 3.10. 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 3.16. 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

- 您系统上的 `tuned.conf (5)` 手册页

## 3.11. 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 3.17. 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

- 您系统上的 **tuned.conf (5)** 手册页

## 3.12. 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**

### 3.13. 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

- 您系统上的 **tuned.conf (5)** 手册页

## 3.14. 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 3.18. 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

- 您系统上的 **tuned.conf (5)** 手册页

## 3.15. 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

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 4 章 使用 TUNA 接口检查系统

使用 **tuna** 工具调整调度程序可调项，调优线程优先级、RIRQ 处理程序，并隔离 CPU 内核和插槽。**tuna** 降低了执行调优任务的复杂性。

**tuna** 工具执行以下操作：

- 列出系统中的 CPU
- 列出系统中当前运行的中断请求 (IRQ)
- 更改线程的策略和优先级信息
- 显示系统的当前策略和优先级

### 4.1. 安装 TUNA 工具

**tuna** 工具设计为在运行的系统上使用。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：

```
# yum install tuna
```

#### 验证

- 显示可用的 **tuna** CLI 选项：

```
# tuna -h
```

#### 其他资源

- 系统上 **tuna (8)** 手册页

### 4.2. 使用 TUNA 工具查看系统状态

这个步骤描述了如何使用 **tuna** 命令行界面 (CLI) 工具查看系统状态。

#### 先决条件

- **tuna** 工具已经安装。如需更多信息，请参阅[安装 \*\*tuna\*\* 工具](#)。

#### 流程

- 查看当前的策略和优先级：

```
# tuna --show_threads
      thread
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

```

- 查看与 PID 或命令名称对应的特定线程：

```
# tuna --threads=pid_or_cmd_list --show_threads
```

*pid\_or\_cmd\_list* 参数是一个用逗号分开的 PID 或 command-name 模式的列表。

- 要使用 **tuna** CLI 调整 CPU，请参阅[使用 tuna 工具调整 CPU](#)。
- 要使用 **tuna** 工具调整 IRQs，请参阅[使用 tuna 工具调整 IRQ](#)。
- 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

## 4.3. 使用 TUNA 工具调整 CPU

**tuna** 工具命令可以针对单个 CPU 为目标。

使用 **tuna** 工具，您可以：

### 隔离 CPU

在指定 CPU 上运行的所有任务都移至下一个可用 CPU。隔离 CPU 会将其从所有线程的关联性掩码中删除使其不可用。

### 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。

这个步骤描述了如何使用 **tuna** CLI 调整 CPU。

### 先决条件

- tuna** 工具已经安装。如需更多信息，请参阅[安装 tuna 工具](#)。

### 流程

- 指定要受某一命令影响的 CPU 列表：

```
# tuna --cpus(cpu_list [command])
```

*cpu\_list* 参数是一个用逗号分开的 CPU 号列表。例如，**--cpus=0,2**。CPU 列表也可以以范围的形式指定，例如 **--cpus="1-3"**，这代表选择 CPU 1、2 和 3。

要将特定的 CPU 添加到当前的 *cpu\_list* 中，例如，使用 **--cpus=+0**。

将 [command] 替换为例如 **--isolate**：

- 隔离 CPU：

```
# tuna --cpus(cpu_list) --isolate
```

- 包括一个 CPU：

```
# tuna --cpus(cpu_list) --include
```

- 以下显示了，在带有四个或更多处理器的系统中，如何在 CPU 0 和 1 上运行所有 ssh 线程，在 CPU 2 和 3 中运行所有 http 线程：

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

这个命令会按顺序执行以下操作：

1. 选择 CPU 0 和 1。
2. 选择以 **ssh** 开头的所有线程。
3. 将所选线程移到所选 CPU。tuna 设置线程的关联掩码，从 **ssh** 开始到适当的 CPU。CPU 可以数字形式表示为 0 和 1，掩码为 0x3（十六进制），或 11（二进制）。
4. 将 CPU 列表重置为 2 和 3。
5. 选择所有以 **http** 开头的线程。
6. 将所选线程移到指定的 CPU。tuna 将以 **http** 开始的线程的关联掩码设置为指定的 CPU。CPU 可以数字化为 2 和 3，掩码为 0xC（十六进制），或 1100（二进制）。

## 验证

- 显示当前配置并验证更改是否已如预期执行：

```
# tuna --threads=gnome-sc\* --show_threads \
--cpus=0 --move --show_threads --cpus=1 \
--move --show_threads --cpus=+0 --move --show_threads

      thread      ctxt_switches
pid SCHED_rtpri affinity voluntary nonvoluntary      cmd
3861  OTHER      0    0,1    33997      58 gnome-screensav
      thread      ctxt_switches
pid SCHED_rtpri affinity voluntary nonvoluntary      cmd
3861  OTHER      0    0    33997      58 gnome-screensav
      thread      ctxt_switches
pid SCHED_rtpri affinity voluntary nonvoluntary      cmd
3861  OTHER      0    1    33997      58 gnome-screensav
      thread      ctxt_switches
pid SCHED_rtpri affinity voluntary nonvoluntary      cmd
3861  OTHER      0    0,1    33997      58 gnome-screensav
```

这个命令会按顺序执行以下操作：

1. 选择以 **gnome-sc** 线程开头的所有线程。
2. 显示所选线程，以便用户验证其关联性掩码和 RT 优先级。
3. 选择 CPU 0。
4. 将 **gnome-sc** 线程移到指定的 CPU CPU 中，CPU 0。
5. 显示移动的结果。
6. 将 CPU 列表重置为 CPU 1。
7. 将 **gnome-sc** 线程移到指定的 CPU 1 中。
8. 显示移动的结果。
9. 将 CPU 0 添加到 CPU 列表中。
10. 将 **gnome-sc** 线程移到指定的 CPU，CPU 0 和 1。
11. 显示移动的结果。

## 其他资源

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

## 4.4. 使用 TUNA 工具调整 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。

这个步骤描述了如何使用 **tuna** 工具调整 IRQ。

### 先决条件

- **tuna** 工具已经安装。如需更多信息，请参阅[安装 tuna 工具](#)。

### 流程

- 查看当前的 IRQs 及其关联性：

```
# tuna --show_irqs
# users      affinity
0 timer      0
1 i8042      0
7 parport0   0
```

- 指定要受某一命令影响的 IRQs 列表：

```
# tuna --irqs=irq_list [command]
```

*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** :

- 将中断移到指定的 CPU:

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

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

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 号列表, 例如 **--cpus=0,2**。如需更多信息, 请参阅[使用 tuna 工具调整 CPU](#)。

## 验证

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

## 其他资源

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

# 第 5 章 使用 RHEL 系统角色，使用 PCP 配置性能监控

Performance Co-Pilot (PCP)是一个系统性能分析工具包。您可以使用它来记录和分析 Red Hat Enterprise Linux 系统上许多组件的性能数据。

您可以使用 **metrics** RHEL 系统角色自动化 PCP 的安装和配置，角色可以配置 Grafana 来视觉化 PCP 指标。

## 5.1. 使用 METRICS RHEL 系统角色配置 PERFORMANCE CO-PILOT

您可以使用 Performance Co-Pilot (PCP)监控许多指标，如 CPU 使用率和内存使用率。例如，这有助于识别资源和性能瓶颈。通过使用 **metrics** RHEL 系统角色，您可以在多个主机上远程配置 PCP 来记录指标。

### 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。

### 流程

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

```
---
- name: Monitoring performance metrics
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure Performance Co-Pilot
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
      vars:
        metrics_retention_days: 14
        metrics_manage_firewall: true
        metrics_manage_selinux: true
```

示例 playbook 中指定的设置包括以下内容：

**metrics\_retention\_days: <number>**

设置多少天后 **pmlogger\_daily** systemd 计时器删除旧的 PCP 归档。

**metrics\_manage\_firewall: <true/false>**

定义角色是否应该在 **firewalld** 服务中打开所需的端口。如果要远程访问受管节点上的 PCP，请将此变量设置为 **true**。

有关 playbook 中使用的所有变量的详情，请查看控制节点上的 **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件。

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

- 查询一个指标，例如：

```
# ansible managed-node-01.example.com -m command -a 'pminfo -f kernel.all.load'
```

## 后续步骤

- 可选：[配置 Grafana 以监控 PCP 主机并视觉化指标](#)。

## 其他资源

- [/usr/share/ansible/roles/rhel-system-roles.metrics/README.md](#) 文件
- [/usr/share/doc/rhel-system-roles/metrics/](#) directory

## 5.2. 使用 **METRICS** RHEL 系统角色配置带有身份验证的 **PERFORMANCE CO-PILOT**

您可以在 Performance Co-Pilot (PCP) 中启用身份验证，以便 **pmcd** 服务和性能指标域代理(PDMA)可以决定是否允许运行监控工具的用户执行操作。已验证的用户可以访问带有敏感信息的指标。另外，某些代理需要身份验证。例如，**bpftrace** 代理使用身份验证识别是否允许用户将 **bpftrace** 脚本加载到内核来生成指标。

通过使用 **metrics** RHEL 系统角色，您可以在多个主机上远程配置带有身份验证的 PCP。

## 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。

## 流程

- 将您的敏感变量存储在一个加密文件中：

- 创建 vault：

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

- 在 **ansible-vault create** 命令打开编辑器后，以 **<key>: <value>** 格式输入敏感数据：

```
metrics_usr: <username>
```

```
metrics_pwd: <password>
```

- c. 保存更改，并关闭编辑器。Ansible 加密 vault 中的数据。
2. 创建一个包含以下内容的 playbook 文件，如 `~/playbook.yml`：

```
---
- name: Monitoring performance metrics
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Configure Performance Co-Pilot
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
      vars:
        metrics_retention_days: 14
        metrics_manage_firewall: true
        metrics_manage_selinux: true
        metrics_username: "{{ metrics_usr }}"
        metrics_password: "{{ metrics_pwd }}"
```

示例 playbook 中指定的设置包括以下内容：

#### **metrics\_retention\_days: <number>**

设置多少天后 `pmlogger_daily` systemd 计时器删除旧的 PCP 归档。

#### **metrics\_manage\_firewall: <true/false>**

定义角色是否应该在 `firewalld` 服务中打开所需的端口。如果要远程访问受管节点上的 PCP，请将此变量设置为 `true`。

#### **metrics\_username: <username>**

角色在受管节点本地创建此用户，将凭证添加到 `/etc/pcp/passwd.db` Simple Authentication and Security Layer (SASL) 数据库中，并在 PCP 中配置身份验证。另外，如果您在 playbook 中设置了 `metrics_from_bpftrace: true`，则 PCP 会使用此帐户注册 `bpftrace` 脚本。

有关 playbook 中使用的所有变量的详情，请查看控制节点上的 `/usr/share/ansible/roles/rhel-system-roles.metrics/README.md` 文件。

3. 验证 playbook 语法：

```
$ ansible-playbook --ask-vault-pass --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。

4. 运行 playbook：

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

## 验证

- 在安装了 `pcp` 软件包的主机上，查询一个需要身份验证的指标：
  - 使用您在 playbook 中使用的凭证查询指标：

```
# pminfo -fmdt -h pcp://managed-node-01.example.com?username=<user>
proc.fd.count
Password: <password>

proc.fd.count
inst [844 or "000844 /var/lib/pcp/pmdas/proc/pmdapro"] value 5
```

如果命令成功，它会返回 **proc.fd.count** 指标的值。

- b. 再次运行命令，但省略用户名，以验证命令是否对未经身份验证的用户失败：

```
# pminfo -fmdt -h pcp://managed-node-01.example.com proc.fd.count
proc.fd.count
Error: No permission to perform requested operation
```

## 后续步骤

- 可选：配置 Grafana 以监控 PCP 主机并视觉化指标。

## 其他资源

- `/usr/share/ansible/roles/rhel-system-roles.metrics/README.md` 文件
- `/usr/share/doc/rhel-system-roles/metrics/` directory
- [Ansible vault](#)

## 5.3. 使用 **METRICS** RHEL 系统角色设置 **GRAFANA**, 来使用 **PERFORMANCE CO-PILOT** 监控多个主机

如果您已在多个主机上配置了 Performance Co-Pilot (PCP)，您可以使用一个 Grafana 的实例来视觉化这些主机的指标。您可以显示实时数据，如果 PCP 数据存储在 Redis 数据库中，也可以显示过去的数据。

通过使用 **metrics** RHEL 系统角色，您可以自动化设置 Grafana、PCP 插件、可选 Redis 数据库以及数据源的配置的过程。



### 注意

如果您使用 **metrics** 角色在主机上安装 Grafana，则角色还会在此主机上自动安装 PCP。

## 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。](#)
- [在您要监控的主机上为远程访问配置了 PCP。](#)
- [在其上要安装 Grafana 的主机可以访问您计划监控的 PCP 节点上的端口 44321。](#)

## 流程

1. 将您的敏感变量存储在一个加密文件中：

a. 创建 vault：

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

b. 在 **ansible-vault create** 命令打开编辑器后，以 **<key>: <value>** 格式输入敏感数据：

```
grafana_admin_pwd: <password>
```

c. 保存更改，并关闭编辑器。Ansible 加密 vault 中的数据。

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

```
---
- name: Monitoring performance metrics
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Set up Grafana to monitor multiple hosts
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
    vars:
      metrics_graph_service: true
      metrics_query_service: true
      metrics_monitored_hosts:
        - <pcp_host_1.example.com>
        - <pcp_host_2.example.com>
      metrics_manage_firewall: true
      metrics_manage_selinux: true

    - name: Set Grafana admin password
      ansible.builtin.shell:
        cmd: grafana-cli admin reset-admin-password "{{ grafana_admin_pwd }}"
```

示例 playbook 中指定的设置包括以下内容：

**metrics\_graph\_service: true**

安装 Grafana 和 PCP 插件。另外，角色在 Grafana 中添加了 **PCP Vector**、**PCP Redis** 和 **PCP bpftrace** 数据源。

**metrics\_query\_service: <true/false>**

定义角色是否应该为集中式指标记录安装和配置 Redis。如果启用了，从 PCP 客户端收集的数据存储在 Redis 中，因此您还可以显示历史数据，而不是只显示实时数据。

**metrics\_monitored\_hosts: <list\_of\_hosts>**

定义要监控的主机的列表。在 Grafana 中，您可以显示这些主机的数据，以及运行 Grafana 的主机。

**metrics\_manage\_firewall: <true/false>**

定义角色是否应该在 **firewalld** 服务中打开所需的端口。如果将此变量设置为 **true**，则您可以远程访问 Grafana。

有关 playbook 中使用的所有变量的详情，请查看控制节点上的 **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** 文件。

### 3. 验证 playbook 语法：

```
$ ansible-playbook --ask-vault-pass --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。

### 4. 运行 playbook：

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

## 验证

1. 在您的浏览器中打开 **http://<grafana\_server\_IP\_or\_hostname>:3000**，并使用您在流程中设置的密码，以 **admin** 用户身份登录。

### 2. 显示监控数据：

- 要显示实时数据：

- i. 单击左侧导航栏中的 **Performance Co-Pilot** 图标，然后选择 **PCP Vector Checklist**。

- ii. 默认情况下，图形显示运行 Grafana 的主机的指标。要切换到其他主机，请在 **hostspec** 字段中输入主机名，然后按 **Enter** 键。

- 要显示存储在 Redis 数据库中的历史数据：[创建一个带有 PCP Redis 数据源的面板](#)。这要求您在 playbook 中设置 **metrics\_query\_service: true**。

## 其他资源

- [/usr/share/ansible/roles/rhel-system-roles.metrics/README.md](#) 文件
- [/usr/share/doc/rhel-system-roles/metrics/](#) 目录
- [Ansible vault](#)

## 第 6 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。

### 6.1. 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 **yum 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

- 您系统上的 **pcp (1)** 手册页
- [/usr/share/doc/pcp-doc/ 目录](#)
- [系统服务和使用 PCP 分发的工具](#)
- [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](#)

### 6.2. 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：

```
# yum install pcp
```

- 在主机机器上启用并启动 **pmcd** 服务：

```
# systemctl enable pmcd
# systemctl start pmcd
```

## 验证

- 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

- 您系统上的 **pmcd (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

## 6.3. 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

- 更新 **pmlogger** 配置：

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

- 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-%H%M).pcp.tar.gz $(hostname)
```

使用 PCP 工具提取此文件并分析数据。

## 其他资源

- 您系统上的 **pmlogconf (1)**, **pmlogger (1)** 和 **pmcd (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

## 6.4. 系统服务和使用 PCP 分发的工具

Performance Co-Pilot (PCP) 包括各种系统服务和可用于衡量性能的工具。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** 软件包还提供了其他工具。

### 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**

实时和历史性性能指标代理、时间序列查询和 REST API 服务。

### 使用基本 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。

**pmic**

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**

提供一个命令行界面来跟踪 PMDA。

**pmval**

显示性能指标的当前值。

## 与单独安装的 pcp-system-tools 软件包一起分发的工具

### **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。报告是从之前使用 **pmlogger** 或 **pcp-atop** 的 **-w** 选项记录的原始日志文件生成的。

### **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**

显示调用进程具有读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)导出的样本和有关 shell-ping 服务指标的报告。

### **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-upptime**

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，并报告是否为某些操作模式正确配置了。

### **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** 设备映射器选项）。

### **pmrep**

报告选定、易于自定义、性能指标值。

## 与单独安装的 pcp-gui 软件包一起分发的工具

**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。

**与单独安装的 pcp-devel 软件包一起分发的工具****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。

## 6.5. PCP 部署架构

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

根据红帽推荐的部署设置、调整因素和配置选项，可用的扩展部署设置变体包括：

**注意**

因为 PCP 版本 5.3.0 在 Red Hat Enterprise Linux 8.4 中以及之前的 Red Hat Enterprise Linux 8 次版本中不可用，红帽推荐使用 localhost 和 pmlogger farm 架构。

有关 PCP 版本早于 5.3.0 的 pmproxy 中的已知内存泄漏的更多信息，请参阅 [PCP 中的 pmproxy 中的内存泄漏](#)。

**localhost**

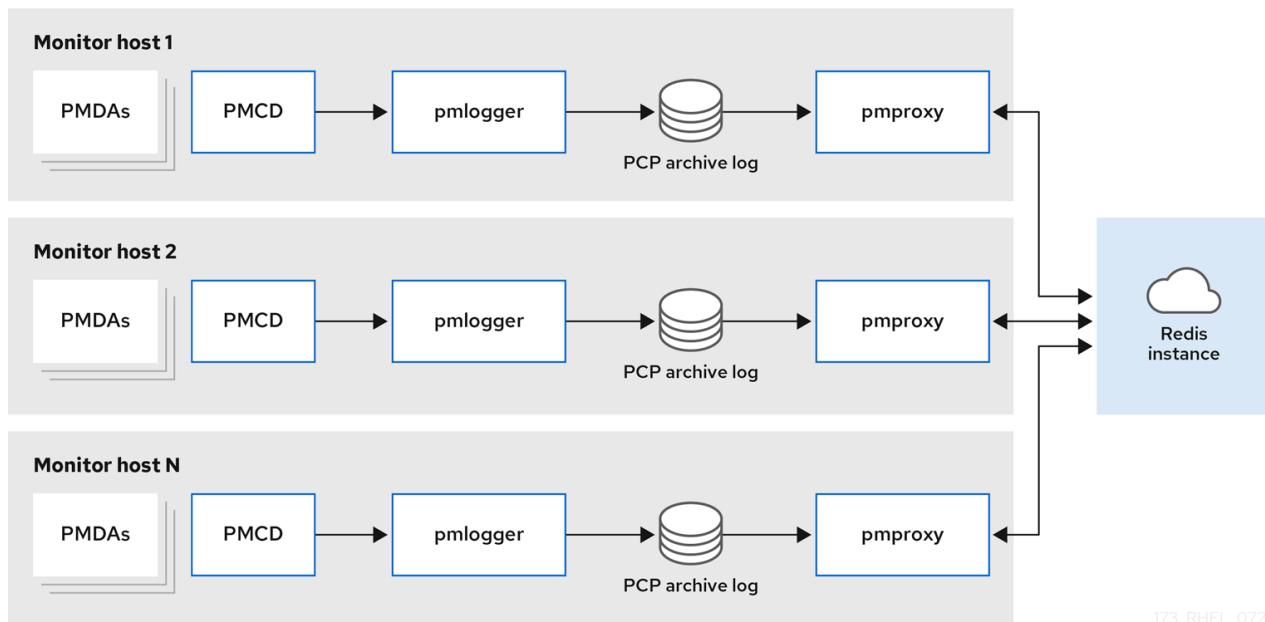
每个服务在被监控的机器上本地运行。当您在没有配置更改的情况下启动服务时，这是默认的部署。在这种情况下无法对单个节点进行扩展。

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 6.1. 分散日志记录

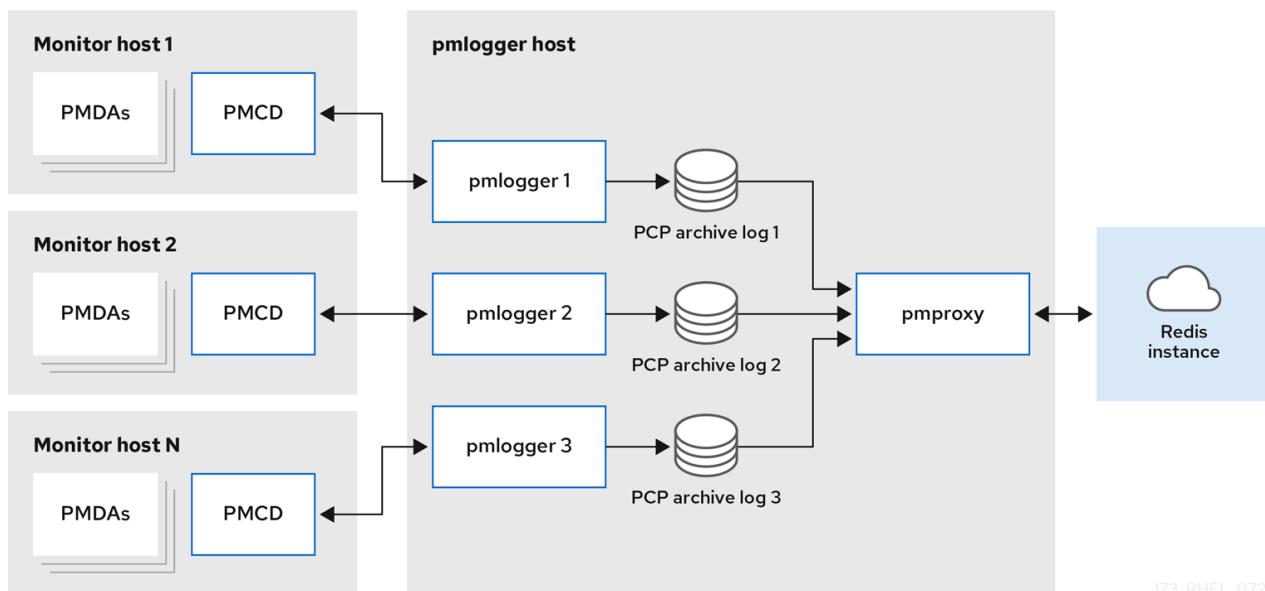


173\_RHEL\_0721

### 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 6.2. 集中式日志记录 - pmlogger 场

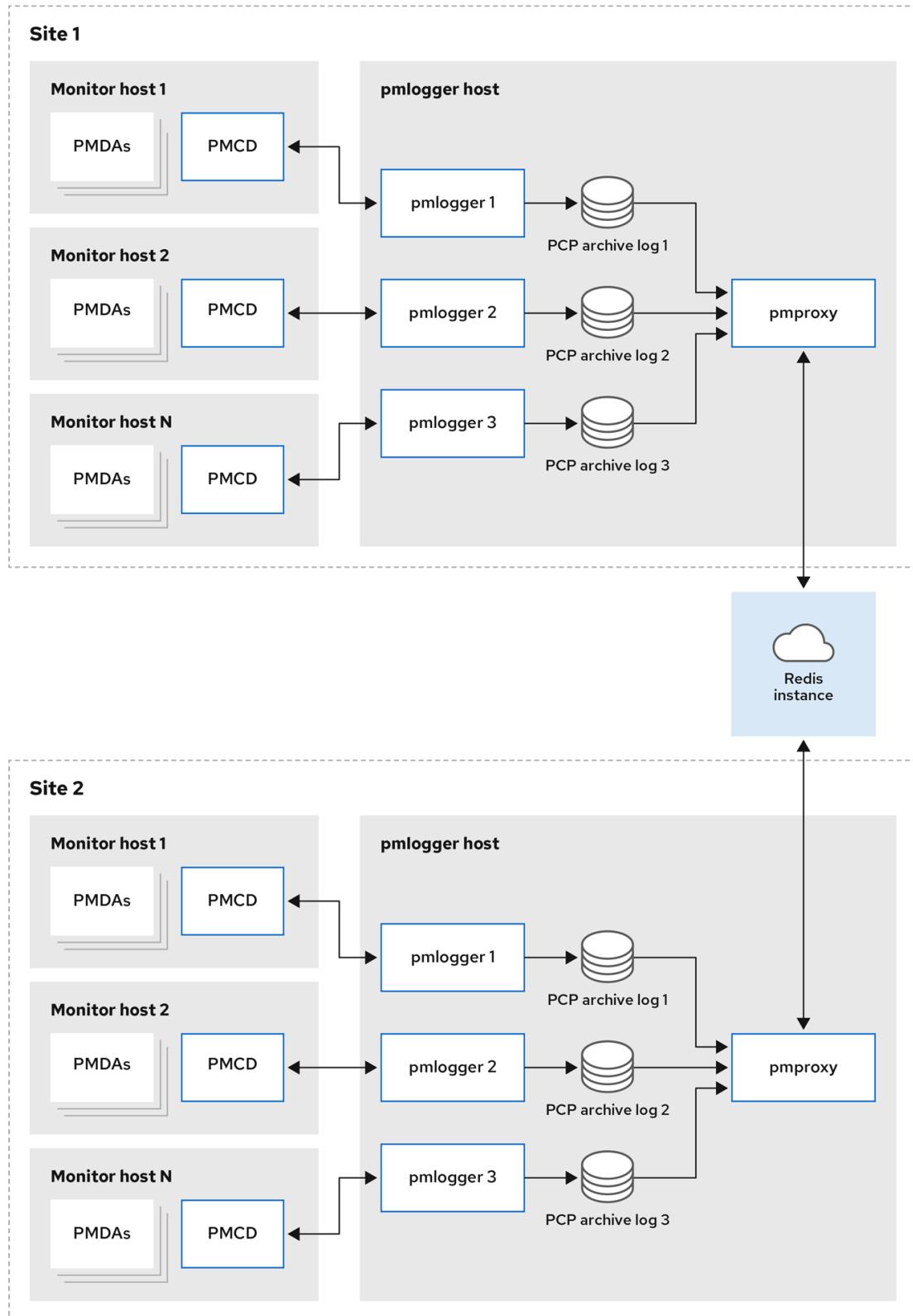


173\_RHEL\_0721

### 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 6.3. 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

- 您系统上的 **pcp (1)**, **pmlogger (1)**, **pmproxy (1)** 和 **pmcd (1)** 手册页
- [推荐的部署架构](#)

## 6.6. 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 6.1. 推荐的部署架构

主机数 (N)	1-10	10-100	100-1000
<b>pmcd</b> 服务器	N	N	N
<b>pmlogger</b> 服务器	1 到 N	N/10 到 N	N/100 到 N
<b>pmproxy</b> 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

## 6.7. 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

- 您系统上的 **pmlogrewrite (1)** 和 **pmlogger (1)** 手册页

## 6.8. 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，请参阅红帽知识库解决方案 [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/pmpoxy/pmpoxy.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

- 您系统上的 **pmpoxy (1)**, **pmlogger (1)** 和 **sysctl (8)** 手册页

## 6.9. 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 6.2. 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

主机数量	10	50
<b>pmlogger</b> Memory	160 MB	580 MB
每天 <b>pmlogger</b> Network (In)	2 MB	9 MB
<b>pmproxy</b> Memory	1.4 GB	6.3 GB
每天的 redis 内存	2.6 GB	12 GB

表 6.3. 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
<b>pmlogger</b> Memory	104 MB	524 MB	1049 MB
每天 <b>pmlogger</b> Network (In)	0.38 MB	1.75 MB	3.48 MB
<b>pmproxy</b> 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](#)。

## 6.10. 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。

此 **pmlogger** 场的设置与下面提到的配置相同

[示例：分析集中式日志记录部署](#)，日志记录间隔为 60 秒，只是 Redis 服务器在集群模式下运行。

表 6.4. 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	<b>pmlogger</b> Memory	每天网络 (In/Out)	<b>pmproxy</b> 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。

## 6.11. 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：

```
# yum 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
14:59:08  0       0       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

- 您系统上的 **pmdaredis (1)**, **pmproxy (1)** 和 **pminfo (1)** 手册页
- [PCP 部署架构](#)

## 第 7 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

### 7.1. 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

- 您系统上的 **pmlogconf (1)** 和 **pmlogger (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

### 7.2. 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

- 您系统上的 **pmlogger (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

## 7.3. 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

- 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

- 您系统上的 **pmlogger (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)
- [/var/lib/pcp/config/pmlogger/config.default file](#)

## 7.4. 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：

```
# yum 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

- 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

- 您系统上的 **pmlogger (1)**, **firewall-cmd (1)**, **ss (8)** 和 **setsebool (8)** 手册页
- [系统服务和使用 PCP 分发的工具](#)
- [/var/lib/pcp/config/pmlogger/config.default file](#)

## 7.5. 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

- 安装 pcp-system-tools 软件包：

```
# yum install pcp-system-tools
```

- 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
```

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



## 注意

在 Red Hat Enterprise Linux 8.0, 8.1 和 8.2 中对控制文件中的远程主机使用以下格式：`PCP_LOG_DIR/pmlogger/host_name`。

### 3. 启用 `pmcd` 和 `pmlogger` 服务：

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

## 验证

- 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

- 您系统上的 `pmlogger (1)` 手册页
- [系统服务和使用 PCP 分发的工具](#)
- [/var/lib/pcp/config/pmlogger/config.default file](#)

## 7.6. SYSTEMD 单元和 PMLOGGER

当您部署 `pmlogger` 服务时，无论是作为监控自身的单个主机，还是作为一个具有从多个远程主机收集指标的单个主机的 `pmlogger` 场，都会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常规检查，以确保 `pmlogger` 实例在运行，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，每天午夜后不久运行，以聚合、压缩和轮转一个或多个 PCP 存档的集合。另外，剔除超过限制的旧存档，默認為 2 周。由 `pmlogger_daily.timer` 单元触发，这是 `pmlogger.service` 单元所需要的。

### `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，这是 **pmlogger.service** 单元所需要的。

### **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，这是 **pmlogger\_farm.service** 单元所需要的，其本身也是 **pmlogger.service** 单元所需要的。

这些服务通过一系列正依赖项来管理，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** 的依赖项被激活将触发 **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在更改生产环境和 PMDA 时的一致性。例如，如果在日志间隔期间更新一个监控的主机上的 **pmcd**，则主机上一些指标的语义可能会被更新，从而使新存档与该主机上之前记录的存档不兼容。如需更多信息，请参阅 [pmlogrewrite \(1\)](#) 手册页。

## 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的自定义归档管理系统。这可以通过使用控制文件来完成。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* 替换为您需要的文件名。

### 注意

主 **pmlogger** 实例必须与它连接的 **pmcd** 运行在同一主机上。您不需要有一个主实例，如果一个中央主机正在连接到远程主机上运行的 **pmcd** 实例的几个 **pmlogger** 实例上收集数据，则在您的配置中可能也不需要它。

文件应为要记录的每个主机包含一行。自动创建的主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：

#### **Host**

要记录的主机的名称

#### **P?**

代表"Primary?"此字段表示主机是否是主记录器实例, **y**, 或不是, **n**。在您配置中的所有文件中只能有一个主记录器, 它必须与它连接的 **pmcd** 运行在同一个主机上。

## S?

代表"Socks?"此字段表示此记录器实例是否需要使用 **SOCKS** 协议, 通过防火墙连接到 **pmcd**, **y** 或不需要, **n**。

## directory

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

指定每天何时结束记录日志。这通常是 **pmlogger\_daily.service** 运行时的时间。默认值 **24h10m** 表示日志记录最迟应当在开始后 24 小时 10 分钟结束。

## -c config.default

指定要使用哪个配置文件。这实际上定义了要记录哪个指标。

## -v 100Mb

指定一个数据卷被填充和另一个数据被创建的大小。切换到新存档后, 之前记录的存档将被 **pmlogger\_daily** 或 **pmlogger\_check** 压缩。

## 其他资源

- 您系统上的 **pmlogger (1)** 和 **pmlogrewrite (1)** 手册页
- 您系统上的 **pmlogger\_daily (1)**, **pmlogger\_check (1)**, 和 **pmlogger.control (5)** 手册页

## 7.7. 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-gui** 软件包：

```
# yum install pcp-gui
```

## 流程

- 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

- 您系统的 **pmlogger (1)**, **pmrep (1)** 和 **pmlogsummary (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

## 第 8 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 8 中的 PCP 应用程序监控系统性能。

### 8.1. 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](#)。

#### 流程

##### 1. 安装以下软件包：

###### a. 安装 **pcp-system-tools**：

```
# yum install pcp-system-tools
```

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

```
# yum install pcp-pmda-postfix postfix
```

###### c. 安装日志记录守护进程：

```
# yum install rsyslog
```

###### d. 安装邮件客户端进行测试：

```
# yum 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

- 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

- 您系统上的 **rsyslogd (8)**, **postfix (1)** 和 **setsebool (8)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

## 8.2. 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：

```
# yum install pcp-gui
```

## 流程

- 从命令行启动 **PCP Charts** 应用程序：

```
# pmchart
```

图 8.1. 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

- 您系统上的 **pmchart (1)** 和 **pmtime (1)** 手册页
- 系统服务和使用 PCP 分发的工具

### 8.3. 使用 PCP 从 SQL 服务器收集数据

在 Red Hat Enterprise Linux 8.2 或更高版本中，SQL Server 代理位于 Performance Co-Pilot (PCP) 中，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。

#### 先决条件

- 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：

```
# yum install pcp-zeroconf
```

2. 安装 **pyodbc** 驱动程序所需的软件包：

```
# yum install gcc-c++ python3-devel unixODBC-devel
# yum install python3-pyodbc
```

3. 安装 **mssql** 代理：

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

```
# yum 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

- 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

- 您系统上的 **pcp (1)**, **pminfo (1)**, **pmval (1)**, **pmchart (1)** 和 **pmdamssql (1)** 手册页
- [带有 RHEL 8.2 Red Hat Developers Blog 的 Microsoft SQL Server 的 Performance Co-Pilot](#)

# 第9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。

## 9.1. 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](#)。

### 流程

- 进入 xfs 目录：

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

- 手动安装 XFS PMDA：

```
xfs]# ./Install
```

You will need to choose an appropriate configuration for install of the “xfs” Performance Metrics Domain Agent (PMDA).

collector	collect performance statistics on this system
monitor	allow this system to monitor local and/or remote systems
both	collector and monitor configuration for this system

Please enter c(ollector) or m(onitor) or (both) [b]

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 xfs metrics have appeared ... 149 metrics and 149 values

- 通过输入 **c** 表示 collector、输入 **m** 表示 monitor 或输入 **b** 表示两者，来选择想要的 PMDA 角色。PMDA 安装脚本提示您指定以下一个 PMDA 角色：

- collector** 角色允许在当前系统上收集性能指标

- monitor** 角色允许系统监控本地系统、远程系统或两者

默认选项是 **collector** 和 **monitor**，它允许 XFS PMDA 在大多数情况下正常工作。

### 验证

- 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

- 您系统上的 **pmcd (1)** 手册页
- [系统服务和使用 PCP 分发的工具](#)

## 9.2. 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](#)。

### 流程

- 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 --helpxtext 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

- 您系统上的 **pminfo (1)** 手册页
- [XFS 的 PCP 指标组](#)
- [每个设备 PCP 指标组用于 XFS](#)

## 9.3. 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](#)。

### 流程

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

- 在重置指标后查看信息：

```
$ pminfo --fetch xfs.write
xfs.write
value 0
```

## 其他资源

- 您系统上的 [pmstore \(1\)](#) 和 [pminfo \(1\)](#) 手册页
- [系统服务和使用 PCP 分发的工具](#)
- [XFS 的 PCP 指标组](#)

## 9.4. XFS 的 PCP 指标组

下表描述了 XFS 可用的 PCP 指标组。

表 9.1. XFS 的指标组

指标组	提供的指标
<b>xfs.*</b>	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。
<b>xfs.allocs.*</b>	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。
<b>xfs.alloc_btree.*</b>	
<b>xfs.block_map.*</b>	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。
<b>xfs.bmap_btree.*</b>	
<b>xfs.dir_ops.*</b>	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。
<b>xfs.transactions.*</b>	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。
<b>xfs.inode_ops.*</b>	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。

<b>xfs.log.*</b>	通过 XFS 文件 systems 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。
<b>XFS.xstrat.*</b>	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。
<b>xfs.attr.*</b>	所有 XFS 文件系统的属性数量、设置、删除和列出操作的数量。
<b>xfs.quota.*</b>	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。
<b>xfs.buffer.*</b>	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）。
<b>xfs.btree.*</b>	有关 XFS btree 操作的指标。
<b>xfs.control.reset</b>	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。

## 9.5. 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 9.2. 每个设备 PCP 指标组用于 XFS

指标组	提供的指标
<b>xfs.perdev.*</b>	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。
<b>xfs.perdev.allocs.*</b>	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。
<b>xfs.perdev.alloc_btree.*</b>	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。
<b>xfs.perdev.dir_ops.*</b>	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。

<b>xfs.perdev.transactions.*</b>	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。
<b>xfs.perdev.inode_ops.*</b>	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。
<b>xfs.perdev.log.*</b> <b>xfs.perdev.log_tail.*</b>	通过 XFS filesystems 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。
<b>xfs.perdev.xstrat.*</b>	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。
<b>xfs.perdev.attr.*</b>	所有 XFS 文件系统的属性数量、设置、删除和列出操作的数量。
<b>xfs.perdev.quota.*</b>	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。
<b>xfs.perdev.buffer.*</b>	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）。
<b>xfs.perdev.btree.*</b>	有关 XFS btree 操作的指标。

# 第 10 章 设置 PCP 指标的图形表示

使用 **pcp**、**grafana**、**pcp redis**、**pcp bpftrace** 和 **pcp vector** 的组合提供了实时数据或由 Performance Co-Pilot (PCP) 收集的数据的图形表示。

## 10.1. 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。

### 流程

- 安装 **pcp-zeroconf** 软件包：

```
# yum install pcp-zeroconf
```

### 验证

- 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

- 您系统上的 **pmlogger** 手册页
- [使用 Performance Co-Pilot 监控性能](#)

## 10.2. 设置 GRAFANA-SERVER

Grafana 生成可从浏览器访问的图形。**grafana-server** 是 Grafana 仪表盘的后端服务器。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** 守护进程进行交互。

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](#)。

### 流程

- 安装以下软件包：

```
# yum install grafana grafana-pcp
```

- 重启并启用以下服务：

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

- 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

- 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

- 您系统上的 **pmp proxy (1)** 和 **grafana-server** 手册页

### 10.3. 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
- 创建仪表盘
- 查看任何有用的指标的概述
- 在 PCP Redis 中创建警报

## 先决条件

- 配置了 PCP。如需更多信息，请参阅[使用 pcp-zeroconf 设置 PCP](#)。
- grafana-server** 被配置。如需更多信息，请参阅[设置 grafana-server](#)。

## 步骤

- 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**。  
当从远程机器访问 Grafana web UI 时，将 `192.0.2.0` 替换为机器 IP，或者在本地访问 web UI 时替换为 **localhost**。
- 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，然后点 **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 10.1. 仪表盘主页



### 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** 配置集 **图标** 上，更改首选项，包括 [编辑配置文件](#)、[更改密码](#) 或 [注销](#)。

## 其他资源

- 您系统上的 **grafana-cli** 和 **grafana-server** 手册页

## 10.4. 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

- 配置了 PCP。如需更多信息，请参阅[使用 pcp-zeroconf 设置 PCP](#)。
- grafana-server** 被配置。如需更多信息，请参阅[设置 grafana-server](#)。
- 邮件传输代理（如 **sendmail** 或 **postfix**）已安装并配置。

### 流程

- 安装 **redis** 软件包：

```
# yum module install redis:6
```



#### 注意

从 Red Hat Enterprise Linux 8.4 起，支持 Redis 6，但 **yum update** 命令不会将 Redis 5 更新到 Redis 6。要从 Redis 5 更新到 Redis 6，请运行：

```
# yum module switch-to redis:6
```

- 启动并启用以下服务：

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

- 重启 **grafana-server**：

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

### 验证

- 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

- 您系统上的 **pmseries (1)** 手册页

## 10.5. 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盘。

**图 10.2. PCP Redis: 主机概述**

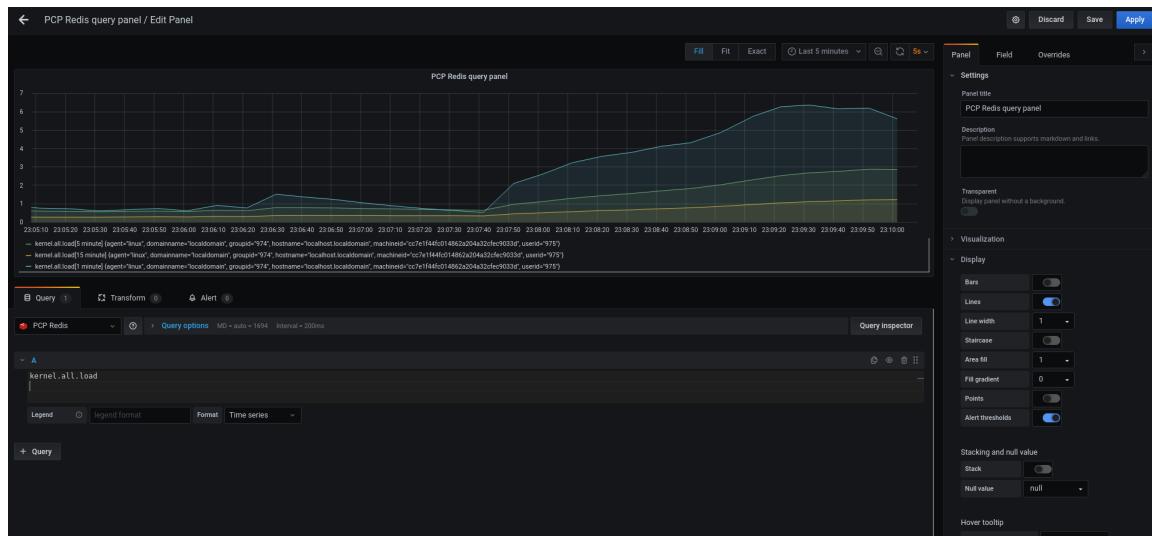


5. 添加新面板：

- +
- 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盘。添加**Dashboard name**。
- e. 点 **Apply** 以应用更改并返回控制面板。

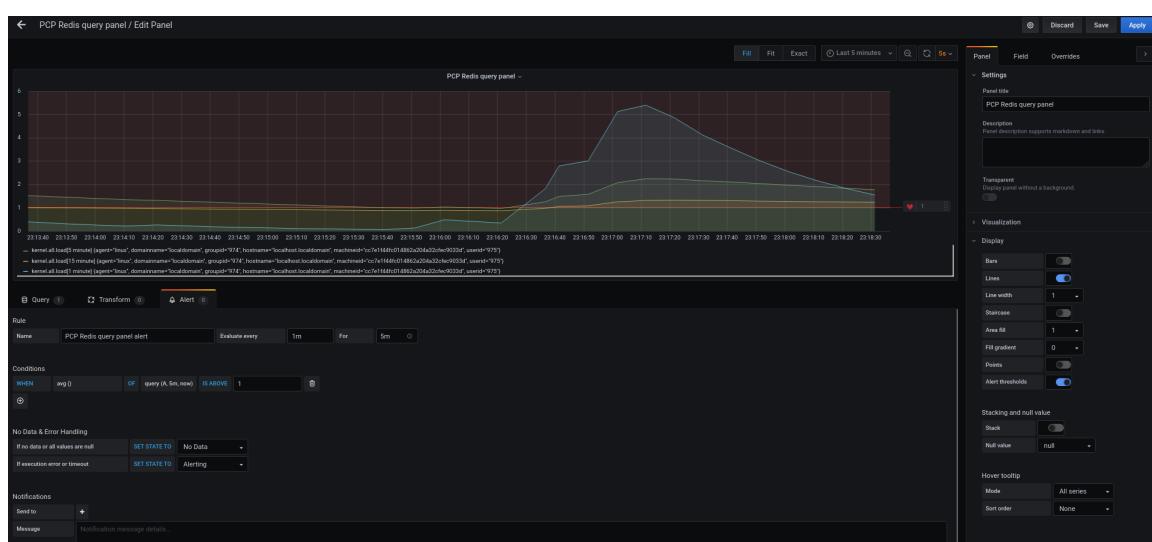
图 10.3. PCP Redis 查询面板



## 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盘。点 Apply 以应用更改并返回控制面板。

图 10.4. 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 **Alerting** 图标查看具有不同警报状态的创建的警报规则，以编辑警报规则，或者从 **Alert Rules** 选项卡中暂停现有规则。  
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](#)。

## 10.6. 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
```

### 流程



1. 在菜单中，将鼠标悬停在 **Alerting** 图标上 → 单击 **Notification channels** → **Add channel**。
2. 在 Add notification 频道详情窗格中执行以下操作：
  - a. 在 **Name** 文本框中输入您的名称
  - b. 选择通信类型，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. 点 **Save**。
4. 在警报规则中选择通知频道：



- a. 在菜单中，将鼠标悬停在 **Alerting** 图标上，然后单击 **Alert rules**。
- b. 在 **Alert Rules** 选项卡中点创建的警报规则。
- c. 在 **通知** 选项卡上，从 **Send to** 选项中选择您的通知频道名称，然后添加警报消息。
- d. 点**应用**。

## 其他资源

- [警报通知的上游 Grafana 文档](#)

## 10.7. 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。



### 注意

在 Red Hat Enterprise Linux 8.3 中，PCP 支持 **scram-sha-256** 身份验证机制。

### 流程

1. 为 **scram-sha-256** 身份验证机制安装 **sasl** 框架：

```
# yum 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/pmcld.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：

```
# saslpwdgen2 -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

- 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

- 您系统上的 **saslauthd (8)**, **pminfo (1)** 和 **sha256** 手册页
- [如何在 RHEL 8.2 中在 PCP 组件（如 PMDA 和 pmcd）之间设置身份验证？](#)（红帽知识库）

## 10.8. 安装 PCP BPFTRACE

Red Hat Enterprise Linux 中的 **pcp-bpftrace** 软件包是 Performance Co-Pilot (PCP)和 **bpftrace** 的组合，用于创建内核中扩展 Berkeley Packet Filter (eBPF)技术的自定义追踪工具。使用 **pcp-bpftrace**，您可以从 **bpftrace** 脚本收集的性能数据优化系统。

**bpftrace** 脚本使用增强的 Berkeley Packet Filter (**eBPF**)。

## 先决条件

- 配置了 PCP。如需更多信息，请参阅[使用 pcp-zeroconf 设置 PCP](#)。
- grafana-server** 被配置。如需更多信息，请参阅[设置 grafana-server](#)。
- 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

- 安装 **pcp-pmda-bpftrace** 软件包：

```
# yum install pcp-pmda-bpftrace
```

- 编辑 **bpftrace.conf** 文件，并添加您在[在 PCP 组件之间设置身份验证](#)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

- 您系统上的 **pmdabpftrace (1)** 和 **bpftrace** 手册页

## 10.9. 查看 PCP BPFTRACE SYSTEM ANALYSIS 仪表盘

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盘。

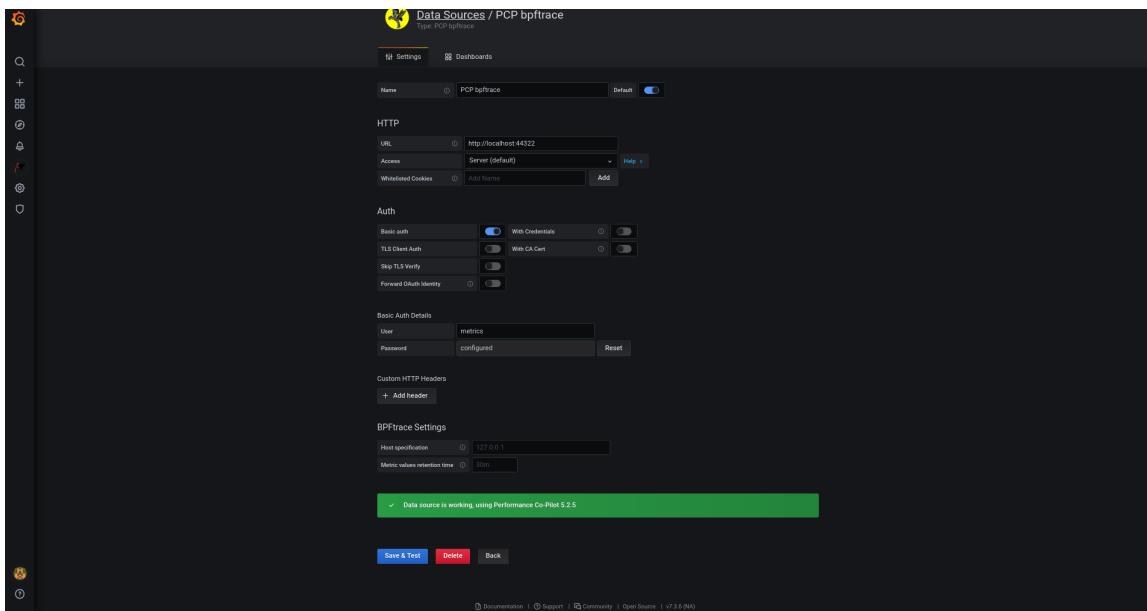
### 先决条件

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

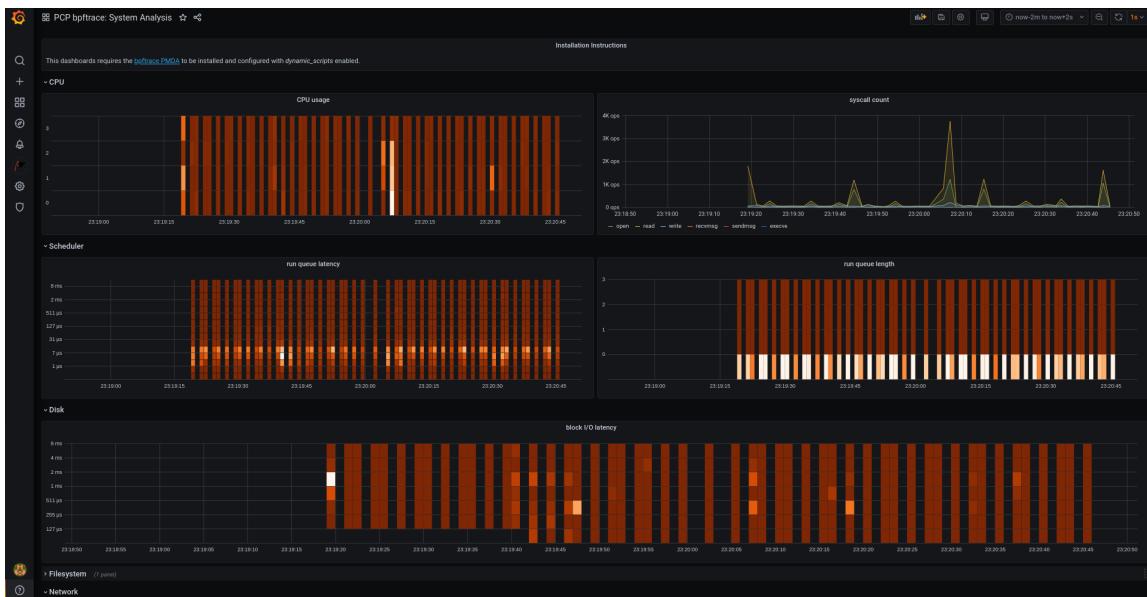
1. 登录到 Grafana Web UI。
2. 在 Grafana Home 页面中，点 **Add your first data source**。
3. 在 Add data source 窗格中，在 **Filter by name or type** 文本框中键入 `bpftrace`，然后点 **PCP bpftrace**。
4. 在 Data Sources / PCP bpftrace 窗格中，执行以下操作：
  - a. 在 URL 字段中添加 `http://localhost:44322`。
  - b. 切换 **Basic Auth** 选项，并在 **User**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

图 10.5. 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盘。

图 10.6. PCP bpftrace: 系统分析



## 10.10. 安装 PCP 向量

安装 **pcp vector** 数据源来显示实时 **pmwebapi** 接口中的实时、主机指标。此数据源旨在对单个主机进行按需性能监控，并包括容器支持。

### 先决条件

- 配置了 PCP。如需更多信息，请参阅[使用 pcp-zeroconf 设置 PCP](#)。
- grafana-server** 被配置。如需更多信息，请参阅[设置 grafana-server](#)。

### 步骤

- 安装 **pcp-pmda-bcc** 软件包：

```
# yum 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

- 您系统上的 **pmdabcc (1)** 手册页

## 10.11. 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盘，并查看清单中的相关故障排除或引用链接。

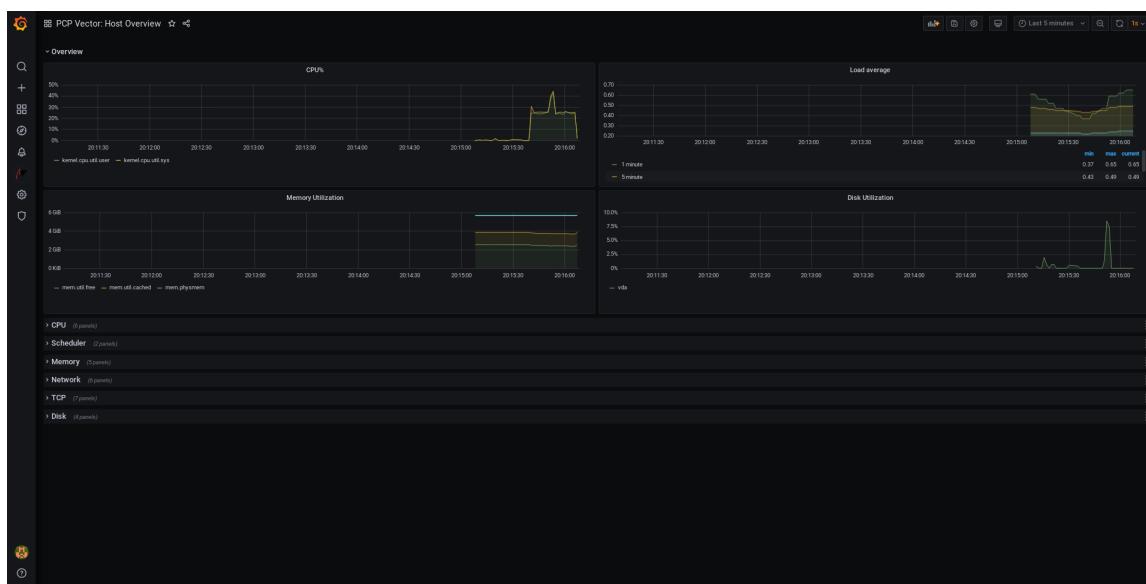
### 先决条件

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盘。

图 10.7. PCP Vector : 主机概述



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

在 PCP 检查列表中，点 help 或 警告图标查看相关的故障排除或参考链接。

图 10.8. Performance Co-Pilot / PCP 向量清单



## 10.12. 在 GRAFANA 中使用 HEATMAPS

您可以在 Grafana 中使用 heatmaps，来查看数据随时间变化的直方图，识别数据中的趋势和模式，并查看它们如何随时间而变化。heatmap 中的每一列代表一个直方图，不同的颜色单元代表该直方图中给定值的不同的观察密度。



### 重要

这个特定工作流用于 RHEL8 上 Grafana 版本 9.2.10 及之后版本中的 heatmaps。

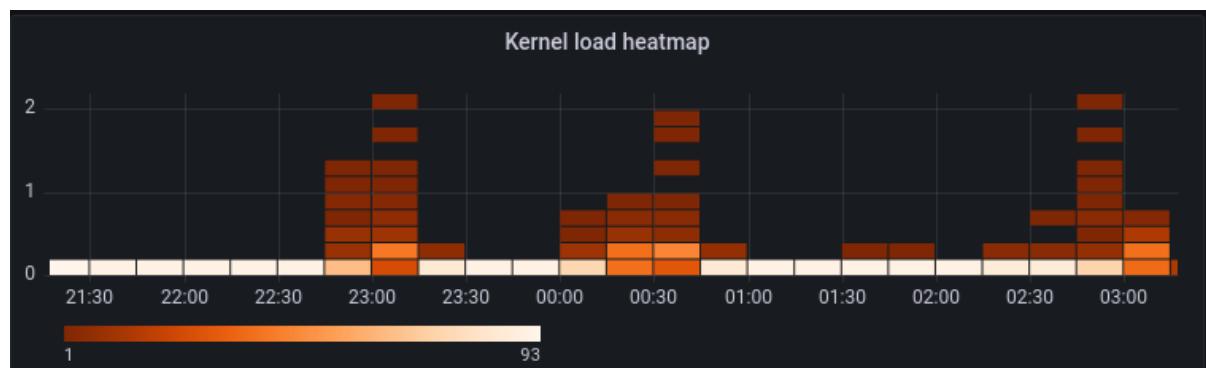
### 先决条件

- 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](#)。

## 流程

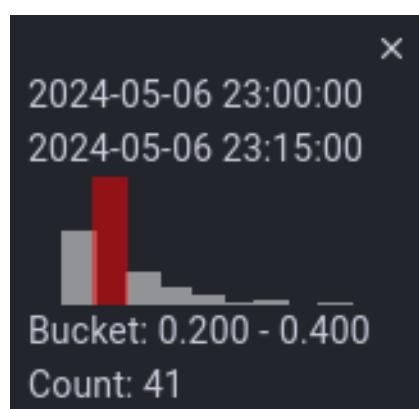
1. 将光标悬停在 **Dashboards** 选项卡上，然后单击 **+ New dashboard**。
2. 在 **Add 面板** 菜单上，单击 **Add a new panel**。
3. 在 **Query** 选项卡中：
  - a. 从查询列表，而不是所选的默认选项中选择 **PCP Redis**。
  - b. 在文本字段 **A** 中，输入指标，如 **kernel.all.load**，来视觉化内核负载图。
4. 单击视觉化下拉菜单，其默认被设为 **Time series**，然后单击 **Heatmap**。
5. 可选：在 **Panel Options** 下拉菜单中，添加 **Panel Title** 和 **Description**。
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，将默认的 **Orange** 改为 **Scheme**，然后选择步骤数(色调)。
8. 可选：在 **Tooltip** 下拉菜单中，在 **Show histogram (Y Axis)** 设置下，单击 **toggle**，以便在光标悬停在 heatmap 中的单元格上时，显示特定直方图中单元的位置。例如：

### 显示直方图(Y轴)单元显示



## 10.13. GRAFANA 问题故障排除

有时，需要对 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。

### 流程

- 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 11 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文件，以便为所选任务优化系统性能。

## 11.1. WEB 控制台中的性能调优选项

Red Hat Enterprise Linux 8 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文件。

在 Web 控制台中，您可以设置系统使用的哪个性能配置文件。

### 其他资源

- [TuneD 入门](#)

## 11.2. 在 WEB 控制台中设置性能配置文件

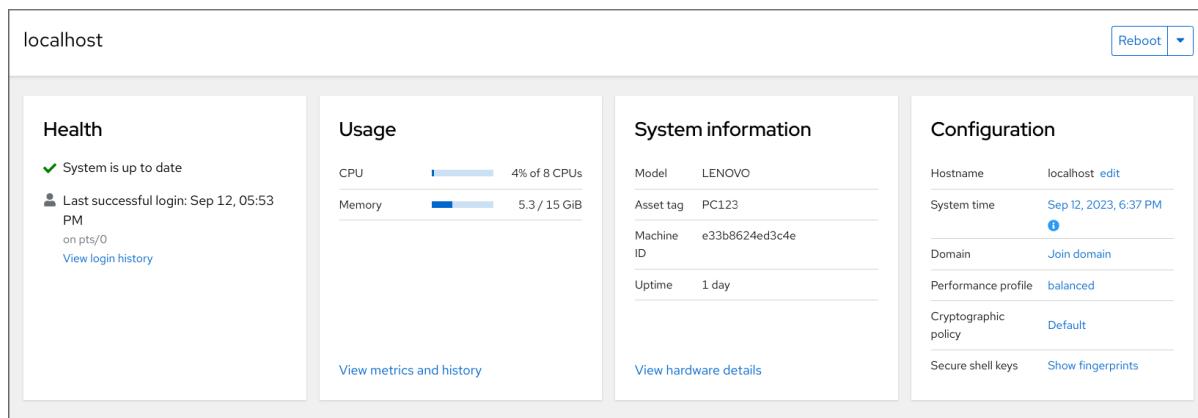
根据您要执行的任务，您可以使用 Web 控制台通过设置合适的性能配置文件来优化系统性能。

### 先决条件

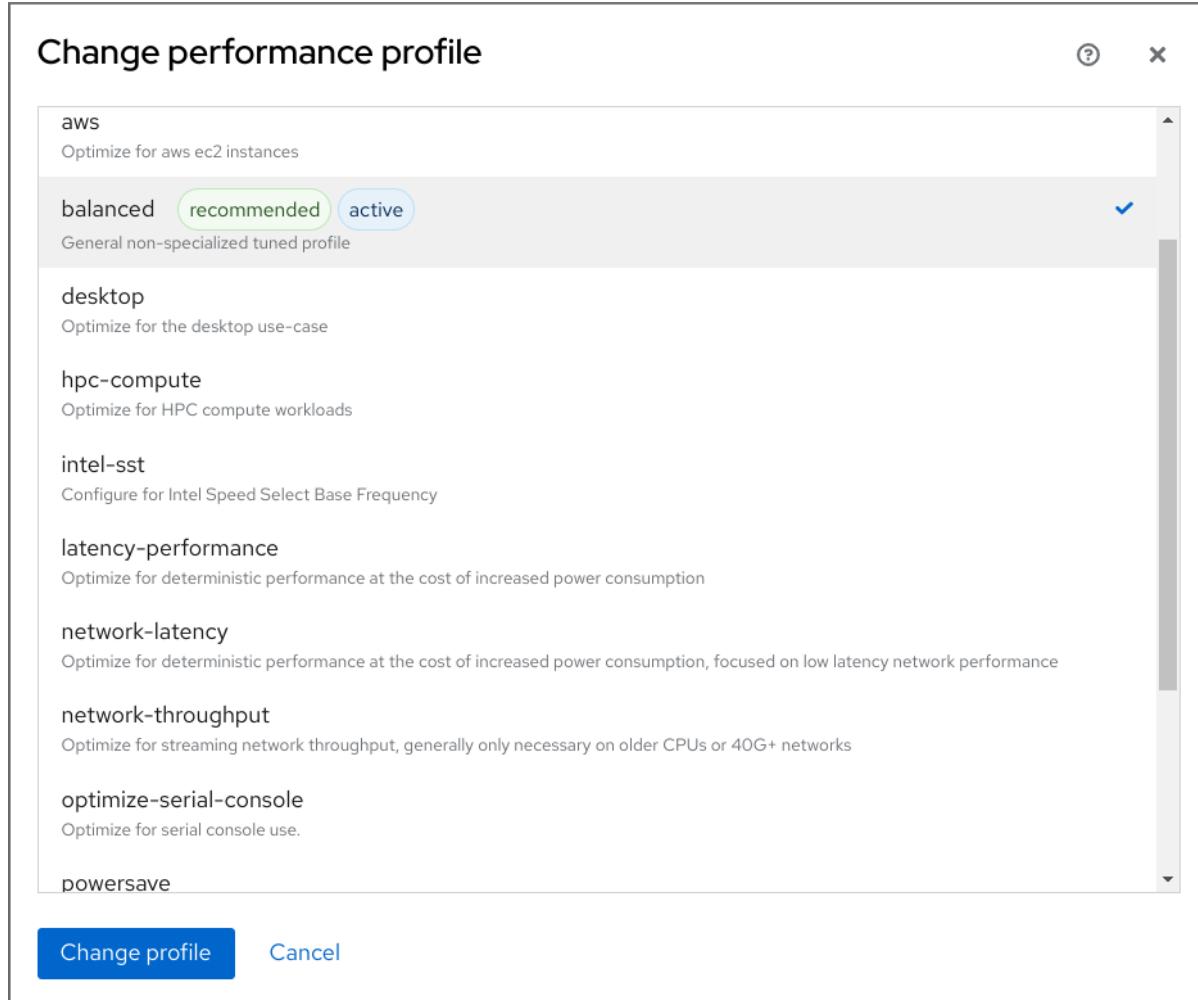
- 您已安装了 RHEL 8 web 控制台。
- 您已启用了 cockpit 服务。
- 您的用户帐户被允许登录到 web 控制台。  
具体步骤请参阅[安装并启用 Web 控制台](#)。

### 流程

1. 登录到 RHEL 8 web 控制台。  
详情请参阅[登录到 web 控制台](#)。
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

- Overview 选项卡现在在 Configuration 部分显示所选的性能配置文件。

## 11.3. 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、错误以及资源利用率和饱和度的图形表示。

## 先决条件

- 您已安装了 RHEL 8 web 控制台。
- 您已启用了 cockpit 服务。
- 您的用户帐户被允许登录到 web 控制台。  
具体步骤请参阅[安装并启用 Web 控制台](#)。
- cockpit-pcp** 软件包（其启用收集性能指标）已安装。
- 启用 Performance Co-Pilot (PCP) 服务：  

```
# systemctl enable --now pmlogger.service pmproxy.service
```

## 流程

- 登录到 RHEL 8 web 控制台。  
详情请参阅[登录到 web 控制台](#)。
- 点 Overview。
- 在 Usage 部分中，点 View metrics and history。

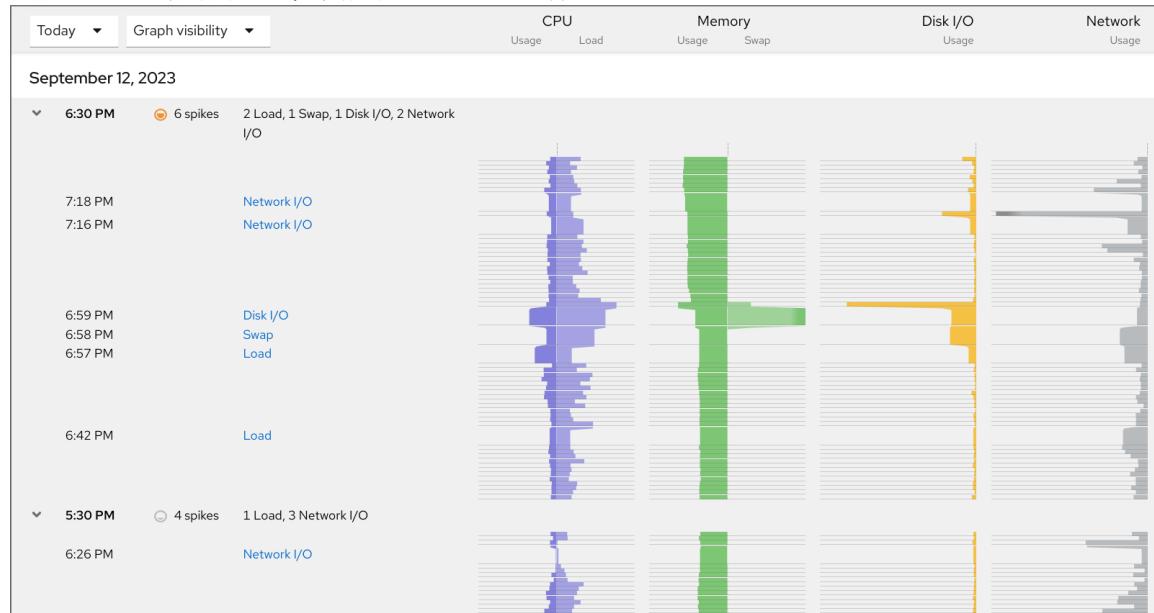
The screenshot shows the RHEL 8 web control panel's Overview page. It features four main sections: Health, Usage, System information, and Configuration. The Health section indicates the system is up-to-date with the last successful login on Sep 12, 05:53 PM. The Usage section displays CPU and Memory usage. The System information section provides details like Model (LENOVO), Asset tag (PC123), and Uptime (1 day). The Configuration section shows the hostname (localhost), system time (Sep 12, 2023, 6:37 PM), domain (Join domain), performance profile (balanced), cryptographic policy (Default), and secure shell keys (Show fingerprints). A 'Reboot' button is at the top right.

Metrics and history 部分打开：

- 当前系统配置和使用率：

The screenshot shows the Metrics and history section of the RHEL 8 web control panel. It includes four main monitoring panels: CPU, Memory, Disks, and Network. The CPU panel shows 8 CPUs with an average load of 4% and a maximum of 7%. The Memory panel shows RAM and Swap usage. The Disks panel shows disk throughput and free space for /, /home, and /boot. The Network panel shows interface statistics for interfaces like wlp6s1, virbr0, and tun0. A 'Metrics settings' button is at the top right.

- 用户指定的时间间隔后，图形形式的性能指标：



## 11.4. 使用 WEB 控制台和 GRAFANA 监控多个系统的性能

Grafana 允许您一次从多个系统收集数据，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。

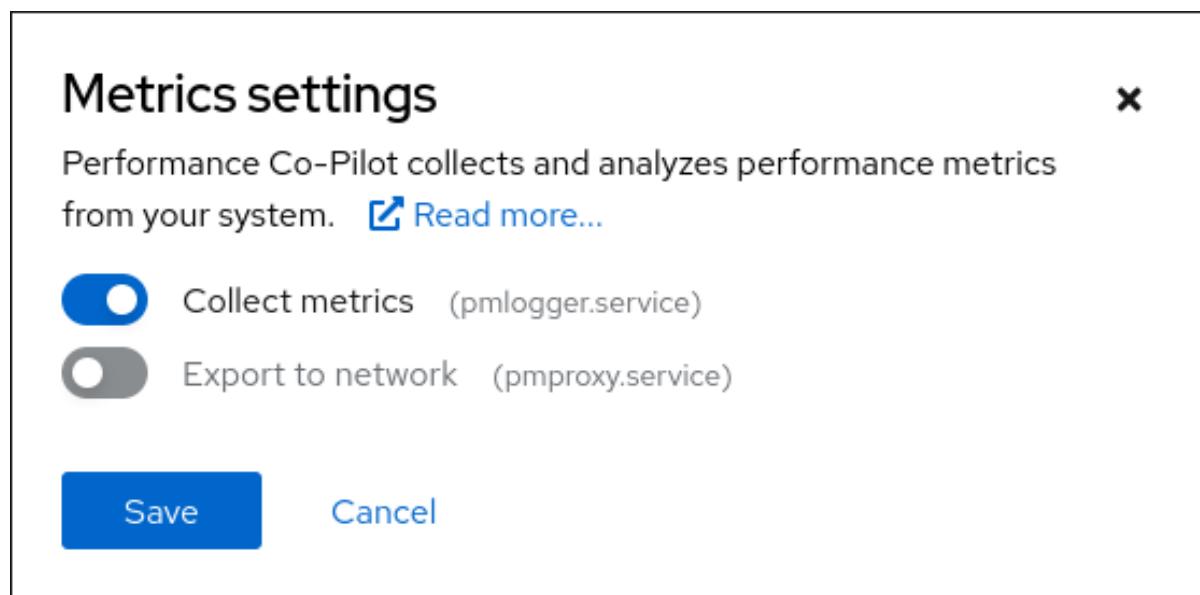
### 先决条件

- 您已安装了 RHEL 8 web 控制台。
- 您已启用了 cockpit 服务。
- 您的用户帐户被允许登录到 web 控制台。  
具体步骤请参阅[安装并启用 Web 控制台](#)。
- 您已安装了 **cockpit-pcp** 软件包。
- 您已启用了 PCP 服务：  

```
# systemctl enable --now pmlogger.service pmproxy.service
```
- 您已设置了 Grafana 仪表盘。如需更多信息，请参阅[设置 grafana-server](#)。
- 您已安装了 **redis** 软件包。  
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. 登录到 RHEL 8 web 控制台。  
详情请参阅[登录到 web 控制台](#)。
2. 在 Overview 页面中，点 Usage 表中的 View metrics and history。
3. 点 Metrics 设置 按钮。
4. 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。

5. 要打开 **pmproxy** 服务，请从下拉列表中选择一个区域，然后点 **Add pmproxy** 按钮。
6. 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 12 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 8 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。

### 12.1. 可用磁盘调度程序

Red Hat Enterprise Linux 8 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。在其默认配置后，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空闲时一样进行响应。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。它不会将磁盘授予每个进程一个固定的时间段，而是为进程分配一个以扇区数衡量的 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。

### 12.2. 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 12.1. 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 <b>mq-deadline</b> 或 <b>bfq</b> 。
高性能 SSD 或具有快速存储的 CPU 绑定系统	使用 <b>none</b> ，特别是在运行企业级应用程序时。或者，使用 <b>kyber</b> 。
桌面或互动任务	使用 <b>bfq</b> 。
虚拟客户端	使用 <b>mq-deadline</b> 。使用可以多队列的主机总线适配器 (HBA) 驱动程序，使用 <b>none</b> 。

## 12.3. 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。

## 12.4. 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。

### 流程

- 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 (**[]**)。

## 12.5. 使用 TUNED 设置磁盘调度程序

此流程创建并启用 **TuneD** 配置文件，该配置文件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

- 可选：选择一个您的配置文件将要基于的现有 **Tuned** 配置文件。有关可用配置文件列表, 请参阅[RHEL 提供的 TuneD 配置文件](#)。

要查看哪个配置文件当前处于活跃状态, 请使用：

```
$ tuned-adm active
```

- 创建一个新目录来保存 **TuneD** 配置文件：

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

- 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*。

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

- 可选：包含现有配置文件：

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

- 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)
```

- 启用您的配置文件：

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

## 验证

- 验证 TuneD 配置文件是否活跃并应用：

```
$ 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.
```

- 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 配置文件。](#)

## 12.6. 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**

## 流程

- 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*。

- 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`）。

- 重新载入 **udev** 规则：

```
# udevadm control --reload-rules
```

- 应用调度程序配置：

```
# udevadm trigger --type=devices --action=change
```

### 验证

- 验证活跃的调度程序：

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

## 12.7. 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。

### 流程

- 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

- 验证调度程序是否在该设备中活跃：

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

# 第 13 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有负面影响。

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  
请参阅[使用 Samba 作为服务器](#)。

### 13.1. 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。

## 流程

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
```

### 13.2. 与包含大量文件的目录调整共享

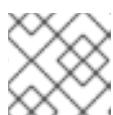
Linux 支持区分大小写的文件名。因此，在搜索或访问文件时，Samba 需要针对大小写文件名来扫描目录。您可以将共享配置为只以小写或大写来创建新文件，这可以提高性能。

## 先决条件

- Samba 配置为文件服务器

## 流程

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，请查看您系统上 **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。

## 13.3. 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 14 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。以下章节解释了导致这种恶化的原因，并提供了有关如何在 RHEL 8 中将虚拟化的性能影响降到最低的说明，以便您的硬件基础架构资源能尽可能被高效地使用。

### 14.1. 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

虚拟机性能损失的原因包括：

- 虚拟 CPU (vCPU) 是主机上的线程，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 8 提供很多功能，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

调优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。

### 14.2. 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。

要为虚拟化优化 RHEL 8，请使用以下配置文件：

- 对于 RHEL 8 虚拟机，请使用 **virtual-guest** 配置文件。它基于普遍适用的 **吞吐量性能** 配置文件，但也减少了虚拟内存的交换。
- 对于 RHEL 8 虚拟化主机，请使用 **virtual-host** 配置文件。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** 配置文件：

- 列出可用的 **TuneD** 配置文件。

```
# 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
```

- 可选：创建一个新的 **TuneD** 配置文件，或编辑现有的 **TuneD** 配置文件。  
如需更多信息，请参阅[自定义 TuneD 配置集](#)。
- 激活 **TuneD** 配置文件。

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

- 要优化虚拟化主机，请使用 **virtual-host** 配置文件。

```
# tuned-adm profile virtual-host
```

- 在 RHEL 虚拟机操作系统中，使用 **virtual-guest** 配置文件。

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

## 验证

- 显示 **TuneD** 的活动的配置文件。

```
# tuned-adm active
Current active profile: virtual-host
```

- 确保已在您的系统上应用了 **TuneD** 配置文件设置。

```
# 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

- 监控和管理系统状态和性能

### 14.3. 针对特定工作负载的虚拟机性能优化

虚拟机(VM)经常专用于执行特定的工作负载。您可以通过优化针对预期工作负载的配置来提高虚拟机的性能。

表 14.1. 针对特定用例推荐的虚拟机配置

使用案例	IOThread	vCPU pinning	vNUMA pinning	巨页	多队列
数据库	对于数据库磁盘	是*	是*	是*	是, 请查看： <a href="#">multi-queue virtio-blk、virtio-scsi</a>
虚拟化网络功能(VNF)	否	是	是	是	是, 请参阅： <a href="#">multi-queue virtio-net</a>
高性能计算(HPC)	否	是	是	是	否
备份服务器	对于备份磁盘	否	否	否	是, 请查看： <a href="#">multi-queue virtio-blk、virtio-scsi</a>
具有许多 CPU (通常超过 32 个) 的虚拟机	否	是*	是*	否	否
具有较大 RAM (通常超过 128 GB) 的虚拟机	否	否	是*	是	否

\* 如果虚拟机有足够的 CPU 和 RAM 来使用多个 NUMA 节点。



#### 注意

虚拟机可以适合多个用例类别。在这种情况下，您应该应用所有推荐的配置。

### 14.4. 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](#)。

#### 14.4.1. 内存过量使用

在 KVM hypervisor 上运行的虚拟机(VM)没有给它们分配的专用物理 RAM 块。相反，每个虚拟机都作为一个 Linux 进程，其中仅在请求时主机的 Linux 内核才分配内存。此外，主机的内存管理器可以在其自身的物理内存和交换空间之间移动虚拟机的内存。如果启用了内存过量使用，内核可以决定分配比虚拟机请求的少的物理内存，因为请求的内存量通常没有被虚拟机的进程完全使用。

默认情况下，Linux 内核中启用了内存过量使用，内核会为虚拟机请求估算安全的内存过量使用。但是，对于内存密集型工作负载，频繁的内存过量使用仍然会使系统变得不稳定。

内存过量使用要求您在主机物理机上分配足够的交换空间，以容纳所有虚拟机，并为主机物理机器的进程分配足够的内存。有关基本推荐的 swap 空间大小的说明，请参阅：[为红帽平台推荐的 swap 大小是什么？](#)

推荐的处理主机上内存短缺的方法：

- 为每个虚拟机分配较少的内存。
- 向主机添加更多的物理内存。
- 使用更大的 swap 空间。



#### 重要

如果交换频繁，虚拟机将会运行较慢。另外，过度使用会导致系统用尽内存(OOM)，这可能会导致 Linux 内核关闭重要的系统进程。

设备分配不支持内存过度使用。这是因为，在使用设备分配时，必须静态预分配所有虚拟机内存，以对分配的设备启用直接内存访问(DMA)。

#### 其他资源

- [虚拟内存参数](#)

#### 14.4.2. 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

- 您已安装了 RHEL 8 web 控制台。
- 您已启用了 cockpit 服务。
- 您的用户帐户被允许登录到 web 控制台。  
具体步骤请参阅[安装并启用 Web 控制台](#)。
- 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](#)。

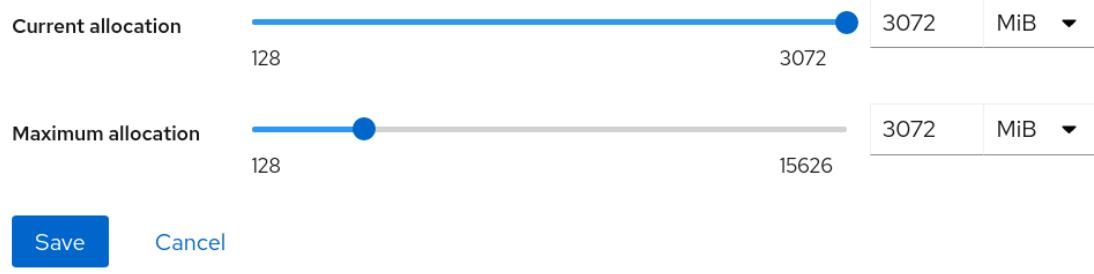
## 流程

1. 可选：获取有关虚拟机的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
```

1. 登录到 RHEL 8 web 控制台。  
详情请参阅[登录到 web 控制台](#)。
2. 在 **Virtual Machines** 界面中，点击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. 单击概述窗格中 **Memory** 行旁边的 **edit**。  
此时将显示 **Memory Adjustment** 对话框。

### Grid\_v2 memory adjustment



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

- [使用命令行添加和删除虚拟机内存](#)
- [优化虚拟机 CPU 性能](#)

### 14.4.3. 使用命令行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 CLI 来调整使用 **memballoon** 设备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 balloon 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：获取有关虚拟机的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。

- 可选：您还可以调整虚拟机当前使用的内存，最多为最大分配量。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

- 确认虚拟机使用的内存已更新：

```
# virsh dominfo testguest
Max memory: 4194304 KiB
Used memory: 2097152 KiB
```

- 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气球统计，以评估它如何有效地控制其内存使用。

```
# 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_palloc=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](#)

#### 14.4.4. 配置虚拟机以使用巨页

在某些情况下，您可以使用巨页而不是默认的 4 KiB 内存页来为虚拟机提高内存分配。例如，巨页可以提高具有高内存使用率的虚拟机的性能，如数据库服务器。

## 前提条件

- 主机被配置为在内存分配中使用巨页。具体说明请参阅：[在引导时配置 HugeTLB](#)

## 步骤

1. 如果所选的虚拟机正在运行，请关闭它。
2. 要将虚拟机配置为使用 1 GiB 巨页，请打开虚拟机的 XML 定义进行编辑。例如，要编辑 **testguest** 虚拟机，请运行以下命令：

```
# virsh edit testguest
```

3. 在 XML 定义中的 **<memoryBacking>** 部分添加以下行：

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

## 验证

1. 启动虚拟机。
2. 确认主机已成功为正在运行的虚拟机分配了巨页。在主机上运行以下命令：

```
# cat /proc/meminfo | grep Huge

HugePages_Total: 4
HugePages_Free: 2
HugePages_Rsvd: 1
Hugepagesize: 1024000 kB
```

当您将空闲的和保留的巨页数加在一起(**HugePages\_Free + HugePages\_Rsvd**)时，结果应小于总巨页数(**HugePages\_Total**)。差距是正在运行的虚拟机所使用的巨页数。

## 其他资源

- [配置巨页](#)

### 14.4.5. 其他资源

- [将设备附加到虚拟机](#)

## 14.5. 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。

### 14.5.1. 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
```

#### 验证

- 检查虚拟机的块 I/O 参数是否已正确配置。

```
# virsh blkiotune testguest1
```

Block I/O tuning parameters for domain testguest1:

```
weight          : 800
device_weight   : [
```

```

    {"sda": 500},
]
...

```



### 重要

某些内核不支持为特定设备设置 I/O 权重。如果上一步没有按预期显示权重，则可能是这个功能与您的主机内核不兼容。

## 14.5.2. 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. 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。有关在使用 RHEL 8 作为虚拟机主机时不支持的功能的更多信息，请参阅 [RHEL 8 虚拟化中不支持的功能](#)。

### 14.5.3. 在存储设备上启用多队列

当在虚拟机(VM)中使用 **virtio-blk** 或 **virtio-scsi** 存储设备时，**多队列**功能提供改进的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。

默认情况下，对 **Q35** 机器类型默认启用了 **多队列**功能，但您必须在 **i440FX** 机器类型上手动启用它。您可以调整队列数以最适合您的工作负载，但每种类型的工作负载的最佳数有所不同，您必须测试哪个队列数最适合您的情况。

#### 流程

- 要在存储设备上启用 **多队列**，请编辑虚拟机的 XML 配置。

```
# virsh edit <example_vm>
```

- 在 XML 配置中，找到预期的存储设备，并将 **queues** 参数改为使用多个 I/O 队列。将 **N** 替换为虚拟机中的 vCPU 数，最多为 16。

- 一个 **virtio-blk** 示例：

```
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' queues='N'/>
  <source dev='/dev/sda' />
  <target dev='vda' bus='virtio' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0' />
</disk>
```

- 一个 **virtio-scsi** 示例：

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

- 重启虚拟机以使更改生效。

### 14.5.4. 配置专用的 IOTreads

要提高虚拟机(VM)上磁盘的输入/输出(IO)性能，您可以配置专用的 **IOTread**，其用于管理虚拟机磁盘的 IO 操作。

通常，磁盘的 IO 操作是主 QEMU 线程的一部分，后者可以在密集型 IO 工作负载期间降低虚拟机的整体响应能力。通过将 IO 操作分离到专用的 **IOTread**，您可以显著提高虚拟机的响应和性能。

#### 流程

- 如果所选的虚拟机正在运行，请关闭它。
- 在主机上，在虚拟机 XML 配置中添加或编辑 **<iotthreads>** 标签。例如，要为 **testguest1** 虚拟机创建单个 **IOThread**：

```
# virsh edit <testguest1>

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <vcpu placement='static'>8</vcpu>
  <iotthreads>1</iotthreads>
  ...
</domain>
```



### 注意

为获得最佳结果，请对主机上的每个 CPU 只使用 1-2 个 **IOThreads**。

- 将一个专用的 **IOThread** 分配给虚拟机磁盘。例如，要将 ID 为 1 的 **IOThread** 分配给 **testguest1** 虚拟机上的一个磁盘：

```
# virsh edit <testguest1>

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native' iotread='1' />
      <source file='/var/lib/libvirt/images/test-disk.raw' />
      <target dev='vda' bus='virtio' />
      <address type='pci' domain='0x0000' bus='0x04' slot='0x00' function='0x0' />
    </disk>
    ...
  </devices>
  ...
</domain>
```



### 注意

**IOThread** ID 从 1 开始，您必须将单个 **IOThread** 专用于一个磁盘。

通常，要获得最佳性能，每个虚拟机有一个专用的 **IOThread** 就足够了。

- 使用 **virtio-scsi** 存储设备时，为 **virtio-scsi** 控制器分配一个专用的 **IOThread**。例如，要将 ID 为 1 的 **IOThread** 分配给 **testguest1** 虚拟机上的一个控制器：

```
# virsh edit <testguest1>

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <controller type='virtio-scsi' index='0' >
    <iothread id='1' />
  </controller>
</domain>
```

```

<devices>
  <controller type='scsi' index='0' model='virtio-scsi'>
    <driver iothread='1'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x0b' function='0x0' />
    </controller>
    ...
  </devices>
  ...
</domain>

```

## 验证

- 评估您的更改对虚拟机性能的影响。详情请查看：[虚拟机性能监控工具](#)

### 14.5.5. 配置虚拟磁盘缓存

KVM 提供多种虚拟磁盘缓存模式。对于密集型输入/输出(IO)工作负载，选择最佳缓存模式可以显著提高虚拟机(VM)性能。

+

#### 虚拟磁盘缓存模式概述

##### writethrough

主机页缓存仅用于读。只有在数据已提交到存储设备时，写才会被报告为完成。持续的 IO 性能会下降，但此模式有很好的写保证。

##### writeback

主机页缓存用于读和写。当数据达到主机的内存缓存，而不是物理存储时，写被报告为完成。这个模式比 **writethrough** 有更快的 IO 性能，但可能会在主机有故障时丢失数据。

##### none

主机页缓存被完全绕过。这个模式直接依赖于物理磁盘的写队列，因此它有可预测的持续的 IO 性能，并在稳定的客户机上提供良好的写保证。其也是用于虚拟机实时迁移的一种安全的缓存模式。

## 流程

- 如果所选的虚拟机正在运行，请关闭它。

- 编辑所选虚拟机的 XML 配置。

```
# virsh edit <vm_name>
```

- 查找磁盘设备，并在 **driver** 标签中编辑 **cache** 选项。

```

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native' iothread='1' />
      <source file='/var/lib/libvirt/images/test-disk.raw' />
      <target dev='vda' bus='virtio' />
      <address type='pci' domain='0x0000' bus='0x04' slot='0x00' function='0x0' />
    </disk>
  </devices>
</domain>

```

```

</disk>
...
</devices>
...
</domain>
```

## 14.6. 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
```
3. [停止内核同页合并\(KSM\)](#)。
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 [用于特定工作负载的虚拟机性能优化](#)。

### 14.6.1. vCPU 过量使用

vCPU 过量使用允许您有一个设置，其中在主机上运行的虚拟机(VM)中所有 vCPU 总和超过主机上的物理 CPU 数。但是，当在虚拟机中同时运行比主机上物理提供的更多的核时，您可能会遇到性能下降。

为了获得最佳性能，请只给虚拟机分配在每个虚拟机中运行预期的工作负载所需的 vCPU 数。

vCPU 过量使用建议：

- 分配虚拟机工作负载所需的最小 vCPU 数，以获得最佳性能。
- 避免在未经过广泛测试的情况下在生产环境中过量使用 vCPU。
- 如果过量使用 vCPU，对于 100% 以下的负载，安全比例通常是 5 个 vCPU 对 1 个物理 CPU。
- 不建议每个物理处理器核分配的总 vCPU 数超过 10 个。
- 监控 CPU 使用率，以防止高负载下出现性能下降。



#### 重要

使用 100% 内存或处理资源的应用程序可能会在过度使用的环境中变得不稳定。不要在未经过广泛测试的情况下在生产环境中过量使用内存或 CPU，因为 CPU 过量使用比率取决于工作负载。

### 14.6.2. 使用命令行添加和删除虚拟 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，vCPU 热拔在 RHEL 8 中不支持，红帽强烈反对使用它。

## 先决条件

- 可选：查看目标虚拟机中 vCPU 的当前状态。例如，要显示 *testguest* 虚拟机上 vCPU 的数量：

```
# 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

- 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。

- 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](#)

### 14.6.3. 使用 Web 控制台管理虚拟 CPU

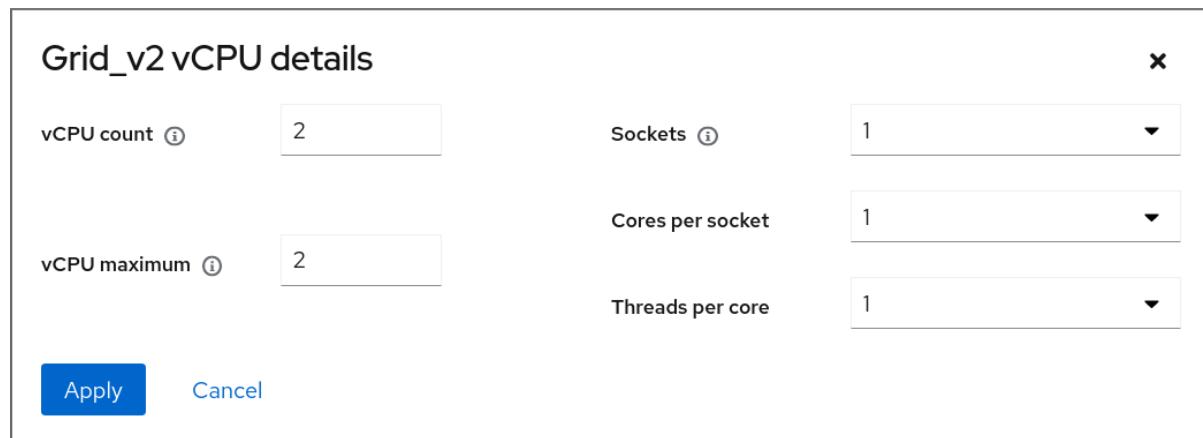
通过使用 RHEL 8 web 控制台，您可以查看并配置 web 控制台连接的虚拟机所使用的虚拟 CPU。

#### 前提条件

- 您已安装了 RHEL 8 web 控制台。
- 您已启用了 cockpit 服务。
- 您的用户帐户被允许登录到 web 控制台。  
具体步骤请参阅[安装并启用 Web 控制台](#)。
- Web 控制台 VM 插件[已安装在您的系统上](#)。

#### 步骤

- 登录到 RHEL 8 web 控制台。  
详情请参阅[登录到 web 控制台](#)。
- 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。
- 点概述窗格中 vCPU 数旁边的 **edit**。  
此时会出现 vCPU 详情对话框。



- 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

- [使用命令行添加和删除虚拟 CPU](#)

### 14.6.4. 在虚拟机中配置 NUMA

以下方法可用于配置 RHEL 8 主机上虚拟机(VM)的非统一内存访问(NUMA)设置。

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。

#### 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。

- 可选：您在主机上已安装了 **numactl** 软件包。

```
# yum install numactl
```

## 流程

### 自动方法

- 将虚拟机的 NUMA 策略设为 **Preferred**。例如，要配置 *testguest5* 虚拟机：

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- 使用 **numad** 服务自动将 VM CPU 与内存资源保持一致。

```
# 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

要手动调整 NUMA 设置，您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。

- 可选：使用 **numactl** 命令查看主机上的 NUMA 拓扑：

```
# numactl --hardware

available: 2 nodes (0-1)
node 0 size: 18156 MB
node 0 free: 9053 MB
node 1 size: 18180 MB
node 1 free: 6853 MB
node distances:
node 0 1
 0: 10 20
 1: 20 10
```

- 编辑虚拟机的 XML 配置，来将 CPU 和内存资源分配给特定的 NUMA 节点。例如，以下配置将 *testguest6* 设置为在 NUMA 节点 **0** 上使用 vCPU 0-7，在 NUMA 节点 **1** 上使用 vCPUs 8-15。两个节点也都分配了 16 GiB 内存。

```
# virsh edit <testguest6>

<domain type='kvm'>
  <name>testguest6</name>
  ...
  <vcpu placement='static'>16</vcpu>
  ...
  <cpu ...>
    <numa>
      <cell id='0' cpus='0-7' memory='16' unit='GiB'/>
      <cell id='1' cpus='8-15' memory='16' unit='GiB'/>
    </numa>
  ...
</domain>
```

- 如果虚拟机正在运行，请重启它以应用配置。



### 注意

为了获得最佳性能，建议尊重主机上每个 NUMA 节点的最大内存大小。

## 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](#)

### 14.6.5. 配置虚拟 CPU 固定

要提高虚拟机(VM)的 CPU 性能，您可以将虚拟 CPU (vCPU)固定到主机上的特定物理 CPU 线程。这确保 vCPU 有自己的专用物理 CPU 线程，这可以显著提高 vCPU 性能。

要进一步优化 CPU 性能，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。

## 流程

1. 检查主机上的 CPU 拓扑：

```
# lscpu -p=node,cpu

Node,CPU
0,0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
1,0
1,1
1,2
1,3
1,4
1,5
1,6
1,7
```

在本例中，输出中包含 NUMA 节点和主机上可用的物理 CPU 线程。

2. 检查虚拟机中的 vCPU 线程数：

```
# lscpu -p=node,cpu

Node,CPU
0,0
0,1
0,2
0,3
```

在本例中，输出中包含 NUMA 节点和虚拟机中可用的 vCPU 线程。

3. 将虚拟机中特定的 vCPU 线程固定到特定的主机 CPU 或 CPU 范围。建议作为一种安全的提高 vCPU 性能的方法。

例如，以下命令将 `testguest6` 虚拟机的 vCPU 线程 0 到 3 分别固定到主机 CPU 1, 3, 5, 7：

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
```

4. 可选：验证 vCPU 线程是否已成功固定到 CPU。

```
# virsh vcpupin testguest6
VCPU  CPU Affinity
-----
0    1
1    3
2    5
3    7
```

5. 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。这可进一步帮助 QEMU 进程在物理 CPU 上更有效地运行。  
例如，以下命令将 *testguest6* 的 QEMU 进程线程固定到 CPU 2 和 4，并验证是否成功：

```
# virsh emulatorpin testguest6 2,4
# virsh emulatorpin testguest6
emulator: CPU Affinity
-----
*: 2,4
```

#### 14.6.6. 配置虚拟 CPU 上限

您可以使用虚拟 CPU (vCPU) 上限来限制虚拟机(VM)可以使用的 CPU 资源量。vCPU 上限可通过防止单个虚拟机过度使用主机的 CPU 资源来提高整体性能，并使 hypervisor 更轻松地管理 CPU 调度。

##### 流程

1. 查看主机上当前的 vCPU 调度配置。

```
# virsh schedinfo <vm_name>

Scheduler      : posix
cpu_shares    : 0
vcpu_period   : 0
vcpu_quota    : 0
emulator_period: 0
emulator_quota : 0
global_period  : 0
global_quota   : 0
iothread_period: 0
iothread_quota : 0
```

2. 要为虚拟机配置一个绝对 vCPU 上限，请设置 **vcpu\_period** 和 **vcpu\_quota** 参数。两个参数都使用一个数字值，其代表以微秒为单位的持续时间。

- a. 使用 **virsh schedinfo** 命令设置 **vcpu\_period** 参数。例如：

```
# virsh schedinfo <vm_name> --set vcpu_period=100000
```

在本例中，**vcpu\_period** 被设置为 100,000 微秒，这意味着调度程序在此时间间隔内强制实施 vCPU 上限。

您还可以使用 **--live --config** 选项来配置一个正在运行的虚拟机，而无需重启它。

- b. 使用 **virsh schedinfo** 命令设置 **vcpu\_quota** 参数。例如：

```
# virsh schedinfo <vm_name> --set vcpu_quota=50000
```

在本例中，**vcpu\_quota** 被设置为 50,000 微秒，它指定虚拟机在 **vcpu\_period** 时间间隔内可以使用的最大 CPU 时间。在本例中，**vcpu\_quota** 被设置为 **vcpu\_period** 的一半，因此虚拟机在此间隔内可使用最多 50% 的 CPU 时间。

您还可以使用 **--live --config** 选项来配置一个正在运行的虚拟机，而无需重启它。

## 验证

- 检查 vCPU 调度参数是否有正确的值。

```
# virsh schedinfo <vm_name>

Scheduler      : posix
cpu_shares    : 2048
vcpu_period   : 100000
vcpu_quota    : 50000
...
...
```

### 14.6.7. 调整 CPU 权重

**CPU 权重**（或 **CPU 共享**）设置控制与其他正在运行的虚拟机相比，虚拟机(VM)接收的 CPU 时间。通过增加特定虚拟机的 **CPU 权重**，您可以确保相对于其他虚拟机，此虚拟机获得更多的 CPU 时间。要在多个虚拟机之间优先分配 CPU 时间，请设置 **cpu\_shares** 参数

可能的 CPU 权重值范围从 0 到 262144，新 KVM 虚拟机的默认值为 1024。

## 流程

1. 检查虚拟机的当前 **CPU 权重**。

```
# virsh schedinfo <vm_name>

Scheduler      : posix
cpu_shares    : 1024
vcpu_period   : 0
vcpu_quota    : 0
emulator_period: 0
emulator_quota : 0
global_period : 0
global_quota  : 0
iothread_period: 0
iothread_quota : 0
```

2. 将 **CPU 权重** 调整为希望的值。

```
# virsh schedinfo <vm_name> --set cpu_shares=2048

Scheduler      : posix
cpu_shares    : 2048
vcpu_period   : 0
vcpu_quota    : 0
emulator_period: 0
emulator_quota : 0
global_period  : 0
global_quota   : 0
iothread_period: 0
iothread_quota : 0
```

在本例中，**cpu\_shares** 被设置为 2048。这意味着，如果所有其他虚拟机的值被设置为 1024，则此虚拟机大约获得两倍的 CPU 时间。

您还可以使用 **--live --config** 选项来配置一个正在运行的虚拟机，而无需重启它。

#### 14.6.8. 禁用内核同页合并

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，使用 KSM 会增加 CPU 使用率，并可能会对总体性能造成负面影响，具体取决于工作负载。

在 RHEL 8 中，KSM 默认被启用。因此，如果虚拟机部署中的 CPU 性能不是最优的，您可以通过禁用 KSM 来改进它。

##### 先决条件

- 对主机系统的 root 访问权限。

##### 流程

- 监控主机上虚拟机的性能和资源消耗，以评估 KSM 的好处。具体来说，请确保 KSM 使用的额外的 CPU 不会抵消内存改进，且不会造成额外的性能问题。在对延迟敏感的工作负载中，还要注意跨 NUMA 页合并。
- 可选：如果 KSM 没有提高虚拟机性能，请禁用它：

- 要为单个会话禁用 KSM，请使用 **systemctl** 工具停止 **ksm** 和 **ksmtuned** 服务。

```
# systemctl stop ksm
# systemctl stop ksmtuned
```

- 要永久禁用 KSM，请使用 **systemctl** 工具禁用 **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
```

但是，这个命令会增加内存使用率，并可能导致主机或虚拟机上出现性能问题。

## 验证

- 监控主机上虚拟机的性能和资源消耗，以评估停用 KSM 的好处。具体说明请查看 [虚拟机性能监控工具](#)。

## 14.7. 优化虚拟机网络性能

由于虚拟机的网络接口控制器(NIC)的虚拟性质，虚拟机会丢失一部分其分配的主机网络带宽，这会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](#)。

## 其它资源

- [了解虚拟网络](#)

## 14.8. 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

要进行标准性能评估，您可以使用主机和客户机操作系统默认提供的工具：

- 在 RHEL 8 主机上，以 root 用户身份使用 **top** 工具或 **system monitor** 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** 工具收集并分析有关 RHEL 8 主机性能的特定于虚拟化的统计。要做到这一点：

1. 在主机上安装 **perf** 软件包：

```
# yum 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，请参阅您系统上的 **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
```

## 14.9. 其它资源

- [优化 Windows 虚拟机](#)

## 第 15 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。

### 15.1. 电源管理基础

有效电源管理基于以下原则构建：

#### 空闲 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空闲状态，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。

## 15.2. 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。

## 15.3. 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更深入地了解某些部分的信息，则大部分工具可以用作补充信息源。

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 **yum 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

它是一个基于配置文件的系统调整工具，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文件作为最适合特定产品的配置文件。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 配置文件。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 2 296 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** 选项的更多信息，请参阅您系统上的 **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

- 您系统上的 **powertop (1)**, **diskdevstat (8)**, **netdevstat (8)**, **tuned (8)**, **vmstat (8)**, **iostat (1)**, **blktrace (8)**, **blkparse (8)** 和 **turbostat (8)** 手册页
- 您系统上的 **cpupower (1)**, **cpupower-set (1)**, **cpupower-info (1)**, **cpupower-idle (1)**, **cpupower-frequency-set (1)**, **cpupower-frequency-info (1)** 和 **cpupower-monitor (1)** 手册页

# 第 16 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。

## 16.1. 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、内核工作者、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 8 使用 PowerTOP 版本 2.x。

## 16.2. 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：

```
# yum install powertop
```

### 16.2.1. 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。

### 16.2.2. 校准 PowerTOP

#### 步骤

- 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
```

- 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。

- 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。

### 16.2.3. 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：

#### 流程

- 使用 **--time** 选项运行 **powertop** 命令：

```
# powertop --time=time in seconds
```

### 16.2.4. 其他资源

有关如何使用 PowerTOP 的详情，请查看您系统上的 **powertop** 手册页

## 16.3. 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。

### 16.3.1. 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

### 16.3.2. 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，当系统空闲时，驻留的最高 C- 或 P-state 应该为 90% 或更高。

### 16.3.3. 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。

### 16.3.4. 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。

### 16.3.5. 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 16.1. PowerTOP 输出

Usage	Events/s	Category	Description
100.0%		Device	Audio codec hwC0D0: QEMU
46.1 ms/s	47.2	Process	[PID 1785] /usr/bin/gnome-shell
1.6 ms/s	27.9	Timer	tick_sched_timer
424.7 μs/s	18.3	Process	[PID 671] [xfsaild/dm-0]
181.4 μs/s	15.4	Process	[PID 11] [rcu_sched]
680.8 μs/s	7.7	Interrupt	[7] sched(softirq)
261.6 μs/s	5.8	Timer	hrtimer_wakeup
261.2 μs/s	5.8	Process	[PID 3745] /usr/libexec/gsd-smartcard
2.9 ms/s	3.9	Process	[PID 6584] /usr/libexec/gnome-terminal-server
43.1 μs/s	3.9	Timer	watchdog_timer_fn
578.4 μs/s	2.9	Process	[PID 4303] /usr/libexec/platform-python /usr/libexec/rhsm-service
251.0 μs/s	2.9	kWork	commit_work
55.1 μs/s	2.9	kWork	virtio_gpu_dequeue_ctrl_func
4.1 ms/s	1.0	Process	[PID 9655] powertop
14.3 μs/s	1.9	kWork	gc_worker
8.0 μs/s	1.9	kWork	kfree_rcu_work
230.3 μs/s	1.0	Process	[PID 1521] /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P

### 其他资源

有关 PowerTOP 的详情，请参阅 [PowerTOP 主页](#)。

## 16.4. 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。

总而言之，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
```

## 16.5. 生成 HTML 输出

除了终端中的 **powertop** 输出外，您还可以生成 HTML 报告。

### 流程

- 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。

## 16.6. 优化功耗

要优化功耗，您可以使用 **powertop** 服务或 **powertop2tuned** 程序。

### 16.6.1. 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：

## 流程

- 启用 **powertop** 服务：

```
# systemctl enable powertop
```

### 16.6.2. powertop2tuned 工具

**powertop2tuned** 程序允许您从 PowerTOP 建议创建自定义 Tuned 配置文件。

默认情况下，**powertop2tuned** 在 **/etc/tuned/** 目录中创建配置文件，并在当前选择的 Tuned 配置文件中基础配置文件。为安全起见，所有 PowerTOP 调优最初在新配置文件中被禁用。

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** 选项生成新的配置文件，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。

### 16.6.3. 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：

```
# yum install tuned-utils
```

## 流程

- 创建自定义配置文件：

```
# powertop2tuned new_profile_name
```

- 激活新配置文件：

```
# 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
```

### 16.6.4. 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 17 章 调整 CPU 频率以优化能源消耗

您可以使用可用的 **cpupower** 命令优化系统的功耗，以便在设置所需的 CPUfreq 管理后，根据您的要求在系统上设置 CPU 速度。

## 17.1. 支持的 CPUPOWER 工具命令

**cpupower** 工具是检查和调优有关处理器相关功能的工具集合。

**cpupower** 工具支持以下命令：

### idle-info

使用 **cpupower idle-info** 命令显示 CPU idle 驱动程序的可用空闲状态和其他统计信息。如需更多信息，请参阅 [CPU Idle States](#)。

### idle-set

以 root 用户身份使用 **cpupower idle-set** 命令启用或禁用特定的 CPU 空闲状态。使用 **-d** 禁用，**-e** 启用特定的 CPU 空闲状态。

### frequency-info

使用 **cpupower frequency-info** 命令显示当前的 **cpufreq** 驱动程序以及可用的 **cpufreq** 调控器。如需更多信息，请参阅 [CPUfreq drivers](#), [Core CPUfreq Governors](#), 和 [Intel P-state CPUfreq governors](#)。

### frequency-set

以 root 用户身份使用 **cpupower frequency-set** 命令设置 **cpufreq** 和 governors。如需更多信息，请参阅 [设置 CPUfreq 调控器](#)。

### set

以 root 用户身份使用 **cpupower set** 命令设置处理器节能策略。

使用 **--perf-bias** 选项，您可以在支持的 Intel 处理器上启用软件来确定最优性能并节省功率。分配的值范围从 **0** 到 **15**，其中 **0** 是最优性能，**15** 是最佳节能。默认情况下，**--perf-bias** 选项适用于所有内核。要将它应用到各个内核，请添加 **--cpu cpulist** 选项。

### info

显示处理器电源和相关硬件配置，该配置已使用 **cpupower set** 命令启用。例如，如果您将 **--perf-bias** 值指定为 **5**：

```
# cpupower set --perf-bias 5
# cpupower info
analyzing CPU 0:
perf-bias: 5
```

### monitor

使用 **cpupower monitor** 命令显示空闲的统计数据和 CPU 需求。

```
# cpupower monitor
| Nehalem    || Mperf   ||Idle_Stats
CPU| C3  | C6  | PC3 | PC6 || C0  | Cx  | Freq || POLL | C1  | C1E | C3  | C6  | C7s | C8  |
C9  | C10
 0| 1.95| 55.12| 0.00| 0.00|| 4.21| 95.79| 3875|| 0.00| 0.68| 2.07| 3.39| 88.77| 0.00| 0.00|
 0.00| 0.00
[...]
```

使用 **-l** 选项，您可以列出系统上的所有可用监视器，并使用 **-m** 选项来显示与特定监控器相关的信息。例如，要监控与 **Mperf** 监控相关的信息，以 root 用户身份使用 **cpupower monitor -m Mperf** 命令。

## 其他资源

- 您系统上的 **cpupower (1)**, **cpupower-idle-info (1)**, **cpupower-idle-set (1)**, **cpupower-frequency-set (1)**, **cpupower-frequency-info (1)**, **cpupower-set (1)**, **cpupower-info (1)** 和 **cpupower-monitor (1)** 手册页

## 17.2. CPU 空闲状态

具有 x86 架构的 CPU 支持各种状态，比如，会停用 CPU 的部分或使用较低性能设置（称为 C-states）。

通过此状态，您可以通过部分取消没有使用的 CPU 来省电。不需要配置 C-state，这与需要管理器的 P-states 不同，一些设置可能会避免不必要的电源或性能问题。C-states 是从 C0 开始增长的值，数值越大代表 CPU 功能越低，节省效果更好。给定数量的 C-states 在处理器之间大体相似，但在不同处理器系列间特定功能集的具体细节可能会有所不同。C-states 0-3 的定义如下：

### C0

在这个状态中，CPU 正常运行，根本不会处于空闲状态。

### C1, Halt

在这一状态中，处理器没有执行任何指令，但通常不处于低电源状态。CPU 可能会在实际并没有延迟的情况下继续处理。提供 C-states 的所有处理器都需要支持此状态。Pentium 4 处理器支持名为 C1E 的改进的 C1 状态，它实际上是功耗更低的状态。

### C2, Stop-Clock

在这个状态中，时钟会对此处理器进行冷冻处理，它会保留其寄存器和缓存的完整状态，因此在时钟再次启动它后，可立即开始进行处理。这是一个可选状态。

### C3, Sleep

在这种状态下，处理器进入睡眠状态，不需要保持其缓存最新状态。因此，从这个状态开始的时间要长于 C2 状态。这是一个可选状态。

您可以使用以下命令查看 CPUidle 驱动程序的可用空闲状态和其他统计：

```
$ cpupower idle-info
CPUidle governor: menu
analyzing CPU 0:

Number of idle states: 9
Available idle states: POLL C1 C1E C3 C6 C7s C8 C9 C10
[...]
```

具有 "Nehalem" 微架构的 Intel CPU 具有 C6 状态，可将 CPU 的供电减少为 0，但通常会降低 80% 到 90% 之间的功耗。Red Hat Enterprise Linux 8 中的内核包括这个新 C-state 的优化。

## 其他资源

- 您系统上的 **cpupower (1)** 和 **cpupower-idle (1)** 手册页

## 17.3. CPUFREQ 概述

在系统上减少功耗和散热输出的最有效方法是 CPUfreq，在 Red Hat Enterprise Linux 8 中的 x86 和 ARM64 架构的支持。CPUfreq（也称为 CPU 加快扩展）是 Linux 内核中的基础架构，它可以扩展 CPU 频率以省电。

CPU 扩展可根据系统负载、响应高级配置和电源接口(ACPI)事件进行自动完成，或者由用户空间程序手动完成，并且允许实时调整处理器的时钟速度。这可让系统以较低的时钟速度运行来省电。CPUfreq 调控器定义切换频率（无论是快速还是较慢的时钟速度）的规则。

您可以以 root 用户身份使用 **cpupower frequency-info** 命令来查看 **cpufreq** 信息。

### 17.3.1. CPUfreq 驱动程序

以 root 用户身份使用 **cpupower frequency-info --driver** 命令，您可以查看当前的 CPUfreq 驱动程序。

以下是可用于 CPUfreq 的两种可用驱动程序：

#### ACPI CPUfreq

高级配置和电源接口 (ACPI) CPUfreq 驱动程序是一个内核驱动程序，它通过 ACPI 控制特定 CPU 的频率，这样可确保内核与硬件之间的通信。

#### Intel P-state

在 Red Hat Enterprise Linux 8 中，支持 Intel P-state 驱动程序。驱动程序提供了一个界面，用于根据 Intel Xeon E 系列架构或更新的架构控制处理器上的 P-state 选择。

目前，在支持的 CPU 中默认使用 Intel P-state。您可以通过在内核命令行中添加 **intel\_pstate=disable** 命令来切换到 ACPI CPUfreq。

Intel P-state 实施 **setpolicy()** 回调。驱动程序根据 **cpufreq** 内核请求的策略决定使用 P-state 的 P-state。如果处理器可以在内部选择其下一个 P-state，则驱动程序会将这一责任卸载到处理器。如果没有，则驱动程序实施算法来选择下一个 P-state。

Intel P-state 提供自己的 **sysfs** 文件来控制 P-state 选择。这些文件位于 **/sys/devices/system/cpu/intel\_pstate/** 目录中。对文件所做的任何更改都适用于所有 CPU。

该目录包含以下用于设置 P-state 参数的文件：

- **max\_perf\_pct** 限制驱动程序请求的最大 P-state，以可用性能百分比表示。可以通过 **no\_turbo** 设置来减少可用的 P-state 性能。
- **min\_perf\_pct** 限制驱动程序请求的最少 P-state，以最大 **no-turbo** 性能级别表示的百分比。
- **no\_turbo** 将驱动程序限制为在 turbo frequency 频率范围内选择 P-state。
- **turbo\_pct** 显示 turbo 范围内硬件支持的总性能百分比。这个号码独立于 **turbo** 已被禁用。
- **num\_pstates** 显示硬件支持的 P-states 数量。这个号码与 turbo 是否已被禁用无关。

#### 其他资源

- 您系统上的 **cpupower-frequency-info (1)** 手册页

### 17.3.2. Core CPUfreq governors

CPUfreq governor 定义系统 CPU 的电源特征，后者反过来会影响 CPU 性能。每个 governor 在工作负载方面都有自己的独特行为、目的和适用性。以 root 用户身份使用 **cpupower frequency-info --governor** 命令，您可以查看可用的 CPUfreq governor。

Red Hat Enterprise Linux 8 包括多个 core CPUfreq governor：

#### **cpufreq\_performance**

它强制 CPU 使用最高可能时钟频率。这个频率是静态设置的，不会更改。因此，这一特定监管器不提供节能功能。它只适用于在较长时间内有大量负载，且在 CPU 很少或永不空闲时才出现。

#### **cpufreq\_powersave**

它强制 CPU 使用最低可能的时钟频率。这个频率是静态设置的，不会更改。该 governor 提供最大节能效果，但以最低 CPU 性能为代价。术语 "powersave" 有时可能并不准确，因为在原则上，在一个慢的 CPU 中有完全负载会比一个没有负载的快速 CPU 消耗更多能源。因此，虽然建议将 CPU 设置为在预期的低活动期间使用 **powersave** governor，但该期间任何意外的高负载都可能导致系统实际消耗更多电源。Powersave governor 对 CPU 速度的限制比对节能更有效果。这对于可能会有过度负载问题的系统和环境中最有用。

#### **cpufreq\_ondemand**

这是一个动态监管器，您可以使用它启用 CPU 以获得系统负载高时的最大时钟频率，以及系统空闲时的最小时钟频率。虽然这允许系统根据系统负载而相应地调整功耗，但在频率切换之间会牺牲延迟。因此，如果系统在空闲和高工作负载间切换过度频频时，通过 **ondemand** governor 获得的性能或节能方面的好处会因为延迟问题而降低。对于大多数系统，**ondemand** governor 可以在散热、功耗、性能和可管理性之间提供最佳折衷。当系统只在一天的特定时间段忙碌时，**ondemand** governor 会根据负载自动切换最大和最小频率，而不进行进一步的干预。

#### **cpufreq\_userspace**

它允许用户空间程序或以 root 身份运行的任何进程来设置频率。在所有 governor 中，**用户空间**是最可自定义的，具体取决于其配置方式，它可以为您的系统提供在性能和功耗的最佳平衡。

#### **cpufreq\_conservative**

与 **ondemand** governor 类似，**conservative** governor 还根据使用情况调整时钟频率。但是，**conservative** governor 会以更加渐进的方式切换时钟频率。这意味着，**conservative** governor 会通过评估对负载的最佳效果来调整时钟频率，而不是只在最大和最小值间进行选择。虽然这可能会在节能方面带来显著效果，但它可能会比 **ondemand** governor 有更多延迟。



#### 注意

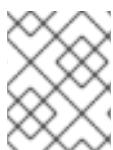
您可以使用 **cron** 任务启用一个 governor。这可让您在一天的指定时间自动设置特定的 governor。因此，您可以在空闲时间（如工作时间后）指定一个低频率 governor，并在有大量负载时返回到更高频率的调控器。

有关如何启用特定监管器的步骤，请参阅[设置 CPUfreq governor](#)。

### 17.3.3. Intel P-state CPUfreq governor

默认情况下，Intel P-state 驱动程序使用活动模式运行，并带有硬件 p-state (HWP)，具体取决于 CPU 支持 HWP。

以 root 用户身份使用 **cpupower frequency-info --governor** 命令，您可以查看可用的 CPUfreq governor。



## 注意

与相同名称的内核 CPUfreq governor 相比，**performance** 和 **powersave** Intel P-state CPUfreq governor 的功能是不同的。

Intel P-state 驱动程序可在以下三种不同的模式下运行：

### 带有硬件管理的 P-states 的活跃模式

当使用 HWP 的活跃模式时，Intel P-state 驱动程序指示 CPU 执行 P-state 选择。驱动程序可以提供频率提示。但是最终选择取决于 CPU 内部逻辑。在带有 HWP 的活跃模式中，Intel P-state 驱动程序提供了两个 P-state 选择算法：

- **performance**：使用 **performance** governor 时，该驱动程序指示内部 CPU 逻辑性能为面向性能的。允许的 P-states 范围限制为驱动程序允许使用的范围的上限。
- **powersave**：使用 **powersave** governor，该驱动程序指示内部 CPU 逻辑为节能。

### 没有硬件管理的 P-states 活跃的模式

在使用没有 HWP 的活跃模式中，Intel P-state 驱动程序提供了两个 P-state 选择算法：

- **performance**：使用 **performance** governor 时，该驱动程序选择允许使用的最大 P-state。
- **powersave**：使用 **powersave** governor，驱动程序选择 P-states proportional to current CPU 使用率。此行为与 **ondemand** CPUfreq core governor 类似。

### 被动模式

当使用 **passive** 模式时，Intel P-state 驱动程序的功能与传统的 CPUfreq 扩展驱动程序相同。所有可用的通用 CPUFreq 内核控制器都可使用。

## 17.3.4. 设置 CPUfreq 调控器

所有 CPUfreq 驱动程序作为 **kernel-tools** 软件包的一部分构建，并自动选择。要设置 CPUfreq，您需要选择一个监管器。

### 先决条件

- 要使用 **cpupower**，请安装 **kernel-tools** 软件包：

```
# yum install kernel-tools
```

### 流程

1. 查看哪些 governor 可用于特定 CPU：

```
# cpupower frequency-info --governors
analyzing CPU 0:
available cpufreq governors: performance powersave
```

2. 在所有 CPU 上启用其中一个 governor：

```
# cpupower frequency-set --governor performance
```

根据您的要求，将 **performance** governor 替换为 **cpufreq** governor 名称。

要只在特定内核上启用 governor，请使用 **-c**，其范围或以逗号分隔的 CPU 编号列表。例如，要为 CPU 1-3 和 5 启用 **userspace** governor，请使用：

```
# cpupower -c 1-3,5 frequency-set --governor cpufreq_userspace
```



### 注意

如果没有安装 **kernel-tools** 软件包，可以在 **/sys/devices/system/cpu/cpuid/cpufreq/** 目录中查看 CPUfreq 设置。可以通过写入这些可调项来更改设置和值。例如，要将 **cpu0** 的最小时钟速度设置为 360 MHz，请使用：

```
# echo 360000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

### 验证

- 验证 governor 是否已启用：

```
# cpupower frequency-info
analyzing CPU 0:
driver: intel_pstate
CPUs which run at the same hardware frequency: 0
CPUs which need to have their frequency coordinated by software: 0
maximum transition latency: Cannot determine or is not supported.
hardware limits: 400 MHz - 4.20 GHz
available cpufreq governors: performance powersave
current policy: frequency should be within 400 MHz and 4.20 GHz.

The governor "performance" may decide which speed to use within this range.
current CPU frequency: Unable to call hardware
current CPU frequency: 3.88 GHz (asserted by call to kernel)
boost state support:
Supported: yes
Active: yes
```

当前策略显示当前启用的 **cpufreq** governor。在这种情况下，它是 **performance**。

### 其他资源

- 您系统上的 **cpupower-frequency-info (1)** 和 **cpupower-frequency-set (1)** 手册页

# 第 18 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。

## 18.1. 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。

## 18.2. 安装 PERF

此流程安装 **perf** 用户空间工具。

### 流程

- 安装 **perf** 工具：

```
# yum install perf
```

## 18.3. 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文件。

### **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 19 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。

## 19.1. **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。

## 19.2. 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     [...] 0x00000000113f9b0
  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     [...] 0x00000000241d1cd
  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

- 您系统上的 **perf-top (1)** 手册页

### 19.3. 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。

### 19.4. 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](#)

### 19.5. 启用调试和源存储库

Red Hat Enterprise Linux 的标准安装不会启用调试和资源存储库。这些存储库包含调试系统组件并测量其性能所需的信息。

## 流程

- 启用源并调试信息软件包通道：

```
# subscription-manager repos --enable rhel-8-for-$uname -i-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$uname -i-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$uname -i-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$uname -i-appstream-source-rpms
```

**\$uname -i** 部分会自动替换为您系统构架的匹配值：

架构名称	订阅价值
64 位 Intel 和 AMD	x86_64
64-bit ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

## 19.6. 使用 GDB 获取应用程序或库的 DEBUGINFO 软件包

需要调试信息来调试代码。对于从软件包安装的代码，GNU Debugger(GDB)自动识别缺少的调试信息，解析软件包名称并提供有关如何获取软件包的具体建议。

### 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](#)。
- 必须在系统上配置并启用提供 **debuginfo** 和 **debugsource** 软件包的存储库。详情请参阅[启用调试和源存储库](#)。

### 流程

- 启动连接到要调试的应用程序或库的 GDB。GDB 自动识别缺少的调试信息，并建议要运行的命令。

```
$ 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)
```

- 退出 GDB：输入 **q** 并使用 **Enter** 进行确认。

```
(gdb) q
```

- 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。

- 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）

# 第 20 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。

### 20.1. **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。

### 20.2. 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 20.1. **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 20.2. 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

- 您系统上的 **perf-stat (1)** 手册页

## 20.3. 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：

- 给定事件的发生次数
- 被计算的事件的名称
- 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。

## 20.4. 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

- 您系统上的 **perf-stat (1)** 手册页

# 第 21 章 使用 PERF 记录和分析性能配置文件

**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。

### 21.1. 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。

### 21.2. 在没有 ROOT 访问权限的情况下记录性能配置文件

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

- 您系统上的 **perf-record (1)** 手册页

### 21.3. 使用 ROOT 访问权限记录性能配置文件

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

- 您系统上的 **perf-record (1)** 手册页

## 21.4. 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

- 您系统上的 **perf-record (1)** 手册页

## 21.5. 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文件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。

### ibr

使用 Intel 处理器上的最后分支记录硬件。

## 其他资源

- 您系统上的 **perf-record (1)** 手册页

## 21.6. 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_erm 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_erm 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_erm
  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 [.]
0x00000000000013b1d 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

- 您系统上的 **perf-report (1)** 手册页

## 21.7. 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。

## 21.8. 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

- 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文件](#)
- [使用 perf record 捕获调用图形数据](#)

## 21.9. 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
```

## 21.10. 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](#)

## 21.11. 启用调试和源存储库

Red Hat Enterprise Linux 的标准安装不会启用调试和资源存储库。这些存储库包含调试系统组件并测量其性能所需的信息。

### 流程

- 启用源并调试信息软件包通道：

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

**\$(uname -i)** 部分会自动替换为您系统构架的匹配值：

架构名称	订阅价值
64 位 Intel 和 AMD	x86_64
64-bit ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

## 21.12. 使用 GDB 获取应用程序或库的 DEBUGINFO 软件包

需要调试信息来调试代码。对于从软件包安装的代码，GNU Debugger(GDB)自动识别缺少的调试信息，解析软件包名称并提供有关如何获取软件包的具体建议。

### 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](#)。
- 必须在系统上配置并启用提供 **debuginfo** 和 **debugsource** 软件包的存储库。详情请参阅[启用调试和源存储库](#)。

### 流程

- 启动连接到要调试的应用程序或库的 GDB。GDB 自动识别缺少的调试信息，并建议要运行的命令。

```
$ 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)
```

- 退出 GDB：输入 **q** 并使用 **Enter** 进行确认。

```
(gdb) q
```

- 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。

- 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)

## 第 22 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。

### 22.1. 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，它可用于指定一个事件（如周期）：

```
# perf stat -a -A -e cycles sleep seconds
```

### 22.2. 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文件

### 22.3. 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。

### 22.4. 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 23 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。

## 23.1. 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。

## 23.2. 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文件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。

### ibr

使用 Intel 处理器上的最后分支记录硬件。

## 其他资源

- 您系统上的 **perf-record (1)** 手册页

### 23.3. 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_erm 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_erm 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_erm
 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 [.]
0x00000000000013b1d 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

- 您系统上的 **perf-report (1)** 手册页

## 第 24 章 使用 PERF 创建 UPROBES

### 24.1. 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

- 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

- 您系统上的 **perf-probe** 手册页
- [使用 perf 记录和分析性能配置文件](#)
- [使用 perf stat 在进程执行过程中计算事件](#)

### 24.2. 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

- 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);
```

- 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
```

### 24.3. 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 的一个函数参数。或者，**a** 可以是您在添加 uprobe 的代码范围内可见的任意变量：

```
# 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 25 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。

### 25.1. **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）。

### 25.2. 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

- 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。

- 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
Snoop        TLB access       Locked
  0.07%      29 98      L1 or L1 hit     [...] 0x000000000000a255
libspeexdsp.so.1.5.0          [...] 0x00007f697a3cd0f0
None         L1 or L2 hit      No
  0.06%      26 97      L1 or L1 hit     [...] 0x000000000000a255
libspeexdsp.so.1.5.0          [...] 0x00007f697a3cd0f0
```

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] 0xfffffb092c06e9084		[kernel.kallsyms]
None	L1 or L2 hit	No		
0.06%	1 2247	Uncached or N/A hit	[k] pci_azx_readl	
[kernel.kallsyms]		[k] 0xfffffb092c06e8164		[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] 0xfffffb092c1842300		[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] 0xfffffb092c06e8164		[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] 0xfffffb092c06e9164		[kernel.kallsyms]
None	L2 miss	No		
0.03%	1 1191	L1 or L1 hit	[k] azx_get_delay_from_lpib	
[kernel.kallsyms]		[k] 0xfffffb092ca7efcf0		[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

- 您系统上的 **perf-mem (1)** 手册页

### 25.3. 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 26 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。

### 26.1. 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

### 26.2. 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

- 您系统上的 **perf-c2c (1)** 手册页

## 26.3. 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 解析缓存行争用](#)。

### 流程

- 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
```



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}					
<hr/>							
1	2832	1119	0	0	0x602100		
<hr/>							
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}					

## 26.4. 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。

## 26.5. 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 解析缓存行争用](#)。

### 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	----	<b>LLC Load Hitm</b>	----	----	Store Reference	----	----	----	----
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	0	0	65
200	3749	12128	5096	108	0	2056	652					
2	0xfffff883ffb6a7e80	260	0.09%	15	3	12	161	161	0	0	1	
1	15	99	25	50	0	6	1					
3	0xffffffff81aec000	157	0.07%	9	0	9	1	0	1	0	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 27 章 FLAMEGRAPHS 入门

作为系统管理员，您可以使用 **flamegraphs** 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。

## 27.1. 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：

```
# yum install js-d3-flame-graph
```

## 27.2. 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。

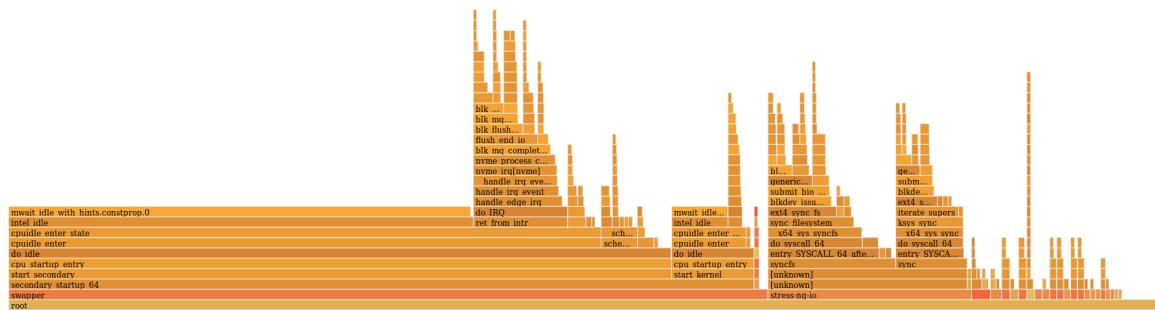
### 验证

- 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：

[Reset Zoom](#) [Clear](#)  [Search](#)



## 27.3. 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**

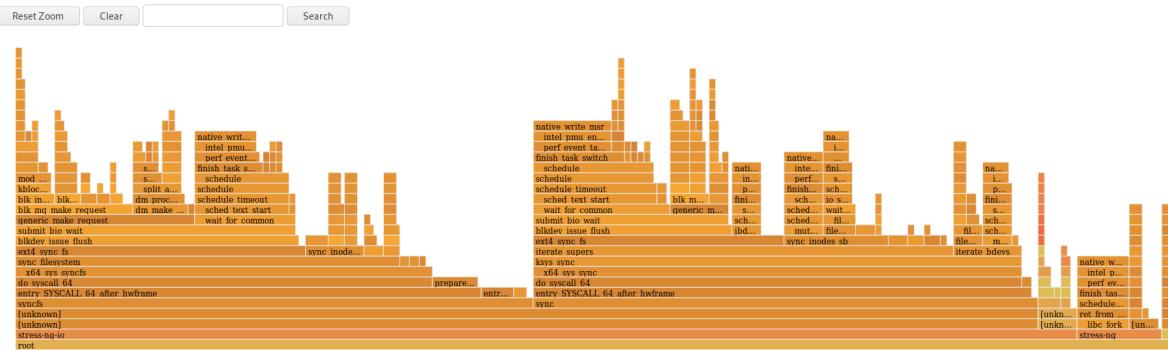
要推断特定进程 ID 的示例并记录数据。

### 验证

- 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：



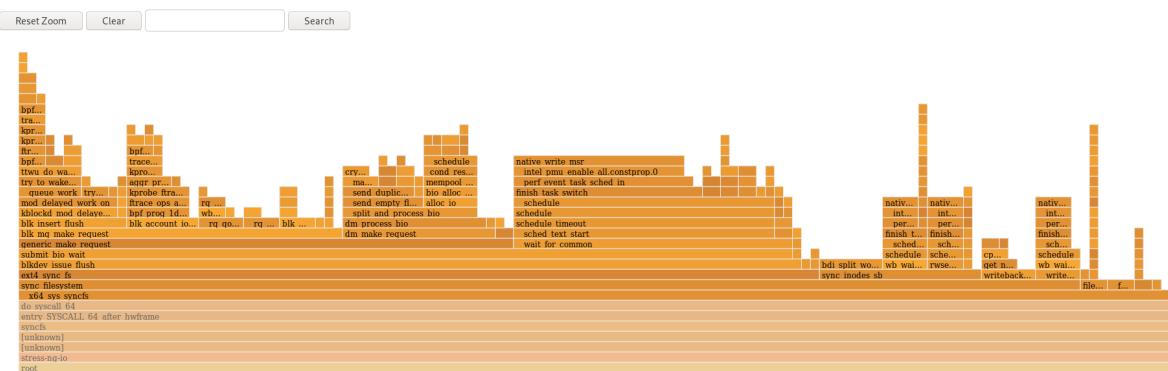
## 27.4. 解释 FLAMEGRAPHS

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

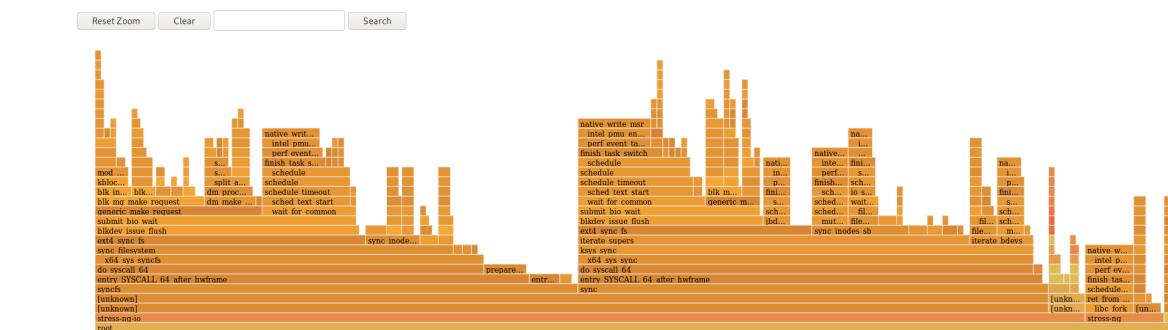
- 要显示之前未显示的功能名称，进一步调查数据，点 flamegraph 中的方框，将缩放到指定位置的堆栈中：



- 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 28 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。

### 28.1. 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。

### 28.2. 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）。

**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 29 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。

## 29.1. 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

- 配置控制管道接口：

```
# mkfifo control ack perf.pipe
```

- 使用您启用的控制文件设置和事件运行 **perf record**：

```
# 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**。

- 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
```

- 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
```

## 29.2. 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 30 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。

## 30.1. 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。

## 30.2. 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：

```
# yum 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

- 您系统上的 **numastat (8)** 手册页

# 第 31 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。

## 31.1. 监控和诊断处理器问题的工具

以下是 Red Hat Enterprise Linux 8 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

- 您系统上的 **turbostat (8)**, **numactl (8)**, **numastat (8)**, **numa (7)**, **numad (8)**, **pqos (8)**, **x86\_energy\_perf\_policy (8)** 和 **taskset (1)** 手册页

## 31.2. 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它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。

#### 31.2.1. 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

```

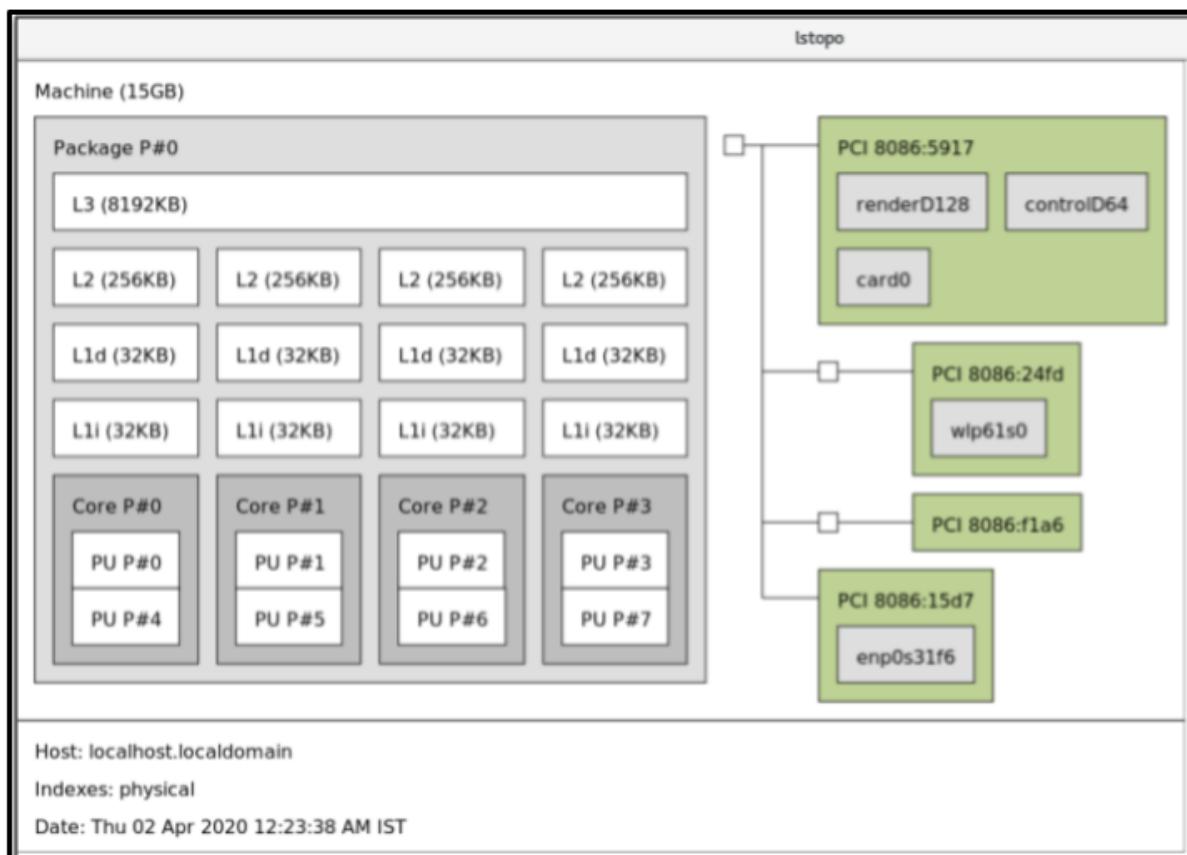
- 查看系统的图形表示：

```

# yum install hwloc-gui
# lstopo

```

图 31.1. lstopo 输出



- 查看详细文本输出：

```

# yum 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

- 您系统上的 **numactl (8)**, **lscpu (1)** 和 **lstopo (1)** 手册页

### 31.3. 配置内核空循环时间

默认情况下，Red Hat Enterprise Linux 8 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 8 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

- 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**）。

- 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
```

- 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。

- 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

- 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

- 您系统上的 **perf (1)** 和 **cpuset (7)** 手册页
- [所有关于 nohz\\_full 内核参数红帽知识库文章](#) (红帽知识库)
- [如何验证 sysfs 中的"隔离"和"nohz\\_full" CPU 信息列表？红帽知识库文章](#) (红帽知识库)

## 31.4. 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。

### 31.4.1. 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

- 检查哪些设备对应于您要配置的中断请求。
- 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** 掩码的更多信息，请参阅 [设置 smp\\_affinity 掩码](#)。

## 其他资源

- 您系统上的 **journalctl (1)** 和 **taskset (1)** 手册页

### 31.4.2. 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 31.1. CPU 的二进制位

CPU	1 5	1 4	1 3	1 2	11 0	1 0	9 0	8 0	7 1	6 0	5 0	4 0	3 0	2 0	1 0	0 1
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 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

- 您系统上的 **journalctl (1)**, **irqbalance (1)** 和 **taskset (1)** 手册页

## 第 32 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。

### 32.1. 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

- 您系统上的 `sched (7)`, `sched_setaffinity (2)`, `sched_getaffinity (2)`, `sched_setscheduler (2)` 和 `sched_getscheduler (2)` 手册页

### 32.2. 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**。

### 32.3. 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**。

### 32.4. 使用 SCHED\_OTHER 常规调度

**SCHED\_OTHER** 是 Red Hat Enterprise Linux 8 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。

### 32.5. 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，或更正正在运行的进程的属性。它还可用于在运行时设置策略。

#### 流程

- 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。

- 检查特定进程的调度策略、PID 和优先级：

```
# chrt -p 468
pid 468s current scheduling policy: SCHED_FIFO
pid 468s 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：

- 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
```

- 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
```

- 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
```

- 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

- 您系统上的 **chrt (1)** 手册页
- [chrt 命令的策略选项](#)
- [在引导过程中更改服务优先级](#)

## 32.6. 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 32.1. chrt 命令的策略选项

短选项	长选项	描述
-f	--fifo	将调度设置为 <b>SCHED_FIFO</b>
-o	--other	将调度设置为 <b>SCHED_OTHER</b>
-r	--rr	将调度设置为 <b>SCHED_RR</b>

## 32.7. 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：

```
# yum 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：

```
# tunctl --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

- 显示 **systemd** 问题设置的 **mcelog** 优先级：

```
# tunctl -t mcelog -P
thread      ctxt_switches
pid SCHED_rt pri affinity voluntary nonvoluntary      cmd
826    FIFO   20  0,1,2,3     13       0      mcelog
```

#### 其他资源

- 您系统上的 **systemd** (1) 和 **tunctl** (8) 手册页
- [优先级范围的描述](#)

## 32.8. 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 32.2. 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 <b>SCHED_OTHER</b> 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。

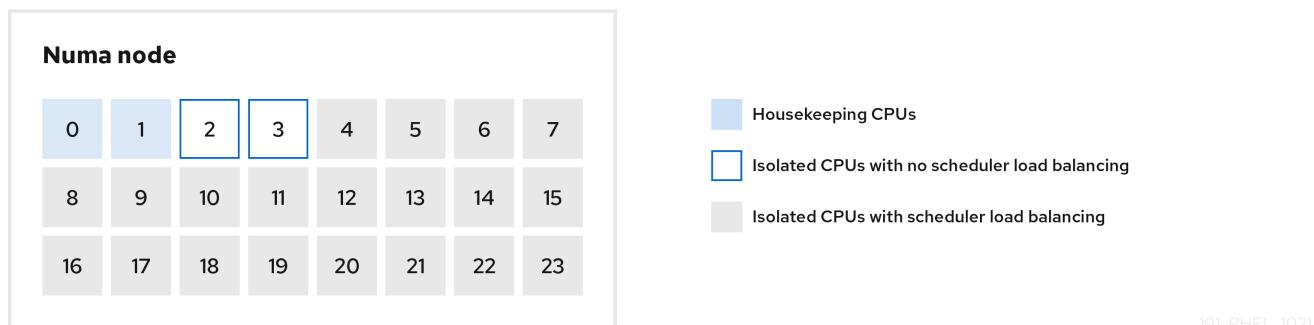
## 32.9. TUNED CPU-PARTITIONING 配置文件

要为对延迟敏感的工作负载调整 Red Hat Enterprise Linux 8，红帽建议使用 **cpu-partitioning** TuneD 配置集。

在 Red Hat Enterprise Linux 8 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 8 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，此配置文件可轻松自定义。

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

图 32.1. **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** 配置文件：

#### 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** 配置文件，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** 配置文件，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

- 您系统上的 **tuned-profiles-cpu-partitioning (7)** 手册页

## 32.10. 使用 TUNED CPU-PARTITIONING 配置文件进行低延迟调整

这个步骤描述了如何使用 Tuned 的 **cpu-partitioning** 配置文件为低延迟调整系统。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 **yum install tuned-profiles-cpu-partitioning** 命令安装 **cpu-partitioning** Tuned 配置集。

## 流程

1. 使用以下更改编辑 **/etc/tuned/cpu-partitioning-variables.conf** 文件：

- 注释掉 **isolated\_cores=\${f:calc\_isolated\_cores:1}** 行：

```
# isolated_cores=${f:calc_isolated_cores:1}
```

- 为隔离的 CPUS 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 配置文件：

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

3. 重启系统。

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。

## 验证

- 验证隔离的 CPU 是否没有在 **Cpus\_allowed\_list** 字段中反映：

```
# cat /proc/self/status | grep Cpu  
Cpus_allowed: 003  
Cpus_allowed_list: 0-1
```

- 要查看所有进程的亲和性，请输入：

```
# ps -ae -o pid= | xargs -n 1 taskset -cp
```

```
pid 1's current affinity list: 0,1  
pid 2's current affinity list: 0,1  
pid 3's current affinity list: 0,1
```

```
pid 4's current affinity list: 0-5
pid 5's current affinity list: 0,1
pid 6's current affinity list: 0,1
pid 7's current affinity list: 0,1
pid 9's current affinity list: 0
...
...
```



### 注意

TuneD 无法更改某些进程的亲和性，主要是内核进程。在本例中，PID 4 和 9 的进程保持不变。

## 其他资源

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

## 32.11. 自定义 CPU-PARTITIONING TUNED 配置集

您可以扩展 TuneD 配置文件，以进行额外的性能优化更改。

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

## 流程

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. 使用新配置文件：

```
# 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* 配置文件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

- 您系统上的 **tuned-profiles-cpu-partitioning (7)** 手册页

## 第 33 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

### 33.1. 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 8 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：
  - **desktop.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

- 您系统上的 **vmstat (8)**, **iostat (1)**, **blktrace (8)**, **blkparse (1)**, **btt (1)**, **bpftrace** 和 **iowatcher (1)** 手册页
- [使用 BPF Compiler Collection 分析系统性能](#)

## 33.2. 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

- 您系统上的 **mkfs (8)** 和 **mount (8)** 手册页
- [可用文件系统概述](#)

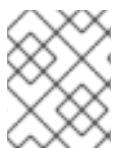
### 33.3. 可用于挂载文件系统的选项

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，并不是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 配置文件会提高预读值，但这些调整并不总是足以满足所有工作负载。

#### 其他资源

- 您系统上的 **mount (8)**、**xfs (5)** 和 **ext4 (5)** 手册页

### 33.4. 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 8 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 8 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 8 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

- 您系统上的 **mount (8)** 和 **fallocate (2)** 手册页

## 33.5. 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。

## 33.6. 通用块设备性能优化参数

此处列出的通用调优参数可在 **/sys/block/sdX/queue/** 目录中找到。

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 34 章 调优网络性能

调优网络设置是一个复杂的进程，有很多因素需要考虑。例如，这包括 CPU 到内存架构、CPU 核数等。Red Hat Enterprise Linux 使用针对大多数场景优化的默认设置。然而，在某些情况下，有必要调优网络设置，以提高吞吐量或延迟或解决问题，如数据包丢弃。

### 34.1. 调优网络适配器设置

在 40 Gbps 及更快的高速网络中，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。调优这些设置可能会阻止此类问题发生。

#### 34.1.1. 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。设备驱动程序排空 RX 环，通常是使用 SoftIRQ，其将传入的数据包放在名为 **sk\_buff** 或 **skb** 的内核数据结构中，以通过内核开始其过程，直到拥有相关套接字的应用程序。

内核使用 TX 环缓冲区来存放应发送到网络的传出数据包。这些环缓冲区位于堆栈的底部，是可能发生数据包丢弃的关键点，这反过来会对网络性能造成负面影响。

#### 流程

- 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可以处理数据包的速度。增加环缓冲有助于避免此类丢失。

- 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（红帽知识库）](#)
- [我是否应该关注 0.05% 的数据包丢弃率？（红帽知识库）](#)
- [您系统上的 \*\*ethtool \(8\)\*\* 手册页](#)

### 34.1.2. 调优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，内核会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，则内核会丢弃将来 **netif\_receive\_skb ()** 内核函数分配给此队列的所有传入的数据包。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，请调优积压队列大小，以避免此问题发生。

## 先决条件

- 一个 10 Gbps 或更快或多个 1 Gbps 网络适配器

## 流程

- 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 0
192058677 18862 0 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。每行代表核 0 开始的 CPU 核。

相关的列有：

- 第一列：收到的总帧数
- 第二列：由于积压队列满了而丢弃的帧数
- 最后一列：CPU 核数

- 如果 **/proc/net/softnet\_stat** 文件的第二列中的值随着时间而递增，请增加积压队列的大小：

- 显示当前积压队列大小：

```
# sysctl net.core.netdev_max_backlog
net.core.netdev_max_backlog = 1000
```

- 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。

- 从 **/etc/sysctl.d/10-netdev\_max\_backlog.conf** 文件中载入设置：

```
# 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，请再次加倍 **net.core.netdev\_max\_backlog** 值。重复此过程，直到数据包丢弃计数不再增加。

### 34.1.3. 增加 NIC 的传输队列长度，以减少传输错误数

内核在传输数据包前，将它们存储在传输队列中。默认长度(1000 个数据包)通常对于 10 Gbps 足够了，通常对于 40 Gbps 网络也足够了。但是，在更快的网络中，或者遇到适配器上传输错误数增加，请增加队列长度。

## 流程

- 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**。

- 监控网络接口的软件传输队列丢弃的数据包计数：

```
# 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)
...
```

- 如果您遇到较高或不断增加的传输错误计数，请设置更高的传输队列长度：

- 识别使用此接口的 NetworkManager 连接配置文件：

```
# nmcli connection show
NAME           UUID             TYPE      DEVICE
Example-Connection  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1s0
```

- 使用以下内容创建 **/etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up** NetworkManager 分配程序脚本：

```
#!/bin/bash
# Set TX queue length on enp1s0 to 2000

if [ "$1" == "enp1s0" ] && [ "$2" == "up" ] ; then
    ip link set dev enp1s0 txqueuelen 2000
fi
```

- 在 **/etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up** 文件上设置可执行位：

```
# chmod +x /etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up
```

- 应用更改：

```
# nmcli connection up Example-Connection
```

## 验证

- 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. 监控丢弃的数据包计数：

```
# tc -s qdisc show dev enp1s0
```

如果 **丢弃** 的计数还在增加，请再次加倍传输队列长度。重复此过程，直到计数不再增加。

## 34.2. 调优 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。

### 34.2.1. 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，它会使用直接内存访问(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。这些中断是由中断处理程序处理的，这些处理程序只做最少的工作，因为它们已中断了另一个任务，处理程序不能中断自己。硬中断在 CPU 使用率方面可能代价高昂，特别是在使用内核锁时。

然后，硬中断处理器将大多数接收的数据包留给软件中断请求(SoftIRQ)进程。内核可以更公平地调度这些进程。

#### 例 34.1. 显示硬件中断

内核将中断计数存储在 **/proc/interrupts** 文件中。要显示特定 NIC 的计数，如 **enp1s0**，请输入：

```
# grep -E "CPU|enp1s0" /proc/interrupts
      CPU0   CPU1   CPU2   CPU3   CPU4   CPU5
 105: 141606     0     0     0     0 IR-PCI-MSI-edge  enp1s0-rx-0
 106:     0 141091     0     0     0 IR-PCI-MSI-edge  enp1s0-rx-1
 107:     2     0 163785     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 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，或者用户载入 NIC 驱动程序模块时，内核会初始化这些向量。会给每个接收(RX)和传输(TX)队列分配一个唯一的向量，其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。

### 34.2.2. 软件中断请求

软件中断请求(SoftIRQ)清除网络适配器的接收环缓冲。内核调度 SoftIRQ 例程在其他任务没有被中断的时候运行。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 内核的 SoftIRQ 计数，请输入：

```
# watch -n1 'grep -E "CPU|NET_RX|NET_TX" /proc/softirqs'
      CPU0   CPU1   CPU2   CPU3   CPU4   CPU5   CPU6   CPU7
NET_RX:  49672  52610  28175  97288  12633  19843  18746  220689
NET_TX:    96   1615    789     46     31   1735   1315  470798
```

命令会动态更新输出。按 **Ctrl+C** 中断输出。

### 34.2.3. NAPI 轮询

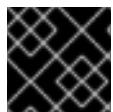
新的 API (NAPI)是设备驱动程序数据包处理框架的扩展，以提高传入网络数据包的效率。硬中断非常昂贵，因为它们通常会导致上下文从内核空间切换到用户空间，然后再切换回来，并且不能中断自己。即使中断合并，中断处理器也会完全垄断一个 CPU 核。使用 NAPI，驱动程序可以使用轮询模式，而不是内核为接收的每个数据包进行硬中断。

在正常操作下，内核会发出一个初始硬中断，后跟一个软中断请求(SoftIRQ)处理程序，该处理程序使用 NAPI 例程轮询网卡。为防止 SoftIRQ 垄断一个 CPU 核，轮询例程有一个确定 SoftIRQ 可以使用的 CPU 时间的预算。完成 SoftIRQ 轮询例程后，内核会退出例程，并将其安排在稍后运行，以重复从网卡接收数据包的过程。

### 34.2.4. irqbalance 服务

在具有和不具有非统一内存访问(NUMA)架构的系统上，**irqbalance** 服务根据系统状况有效地平衡跨 CPU 核的中断。**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 CPU 核 0 通常会处理大多数中断。即使在中等负载下，这个 CPU 核可能会变得非常忙碌，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、数据包丢失以及其他潜在的问题。



#### 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** 服务不提供任何好处，并自行退出。

默认情况下，**irqbalance** 服务在 Red Hat Enterprise Linux 上启用并运行。要重新启用该服务（如果您禁用了它），请输入：

```
# systemctl enable --now irqbalance
```

#### 其他资源

- [我们是否需要 irqbalance？](#) (红帽知识库)
- [如何配置网络接口 IRQ 通道？](#) (红帽知识库)

### 34.2.5. 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快速排空缓冲区的能力。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 轮询周期中检索来自接口的所有数据包，则表示 SoftIRQ 没有足够的 CPU 时间。这种情况可能出现在具有快速 NIC 的主机上，如 10 Gbps 和更快。如果您增加了 **net.core.netdev\_budget** 和 **net.core.netdev\_budget\_usecs** 内核参数的值，则您可以控制 **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** 文件中的计数：

-

```
# 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。每行代表核 0 开始的 CPU 核。

相关的列有：

- 第一列：收到的总帧数。
- 第三列：**softirqd** 进程不能在一个 NAPI 轮询周期中检索来自接口的所有数据包的次数。
- 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，请调优系统：

- 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，**softirqd** 进程在一个轮询周期内最多有 2000 微秒来处理最多 300 个来自 NIC 的消息。轮询根据首先满足哪个条件而结束。

- 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
```

如果值还在增加，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。

## 34.3. 改善网络延迟

CPU 电源管理功能可能会在时间敏感的应用程序中导致不必要的延迟。您可以禁用一些或所有这些电源管理功能，以改善网络延迟。

例如，如果延迟在服务器空闲时比负载过重时高，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损失。

#### 34.3.1. 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，处理器完全加电并执行。在 C1 中，处理器完全加电，但没有执行。C-state 的值越大，CPU 关闭的组件越多。

每当 CPU 核空闲时，内置节能逻辑步骤就会介入，并尝试通过关闭各种处理器组件将核从当前 C-state 移到更高状态。如果 CPU 核必须处理数据，则 Red Hat Enterprise Linux (RHEL)会将一个中断发送给处理器，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不同组件通电。在多核系统上，也可能发生许多核同时空闲，因此处于更深的 C-states。如果 RHEL 尝试同时唤醒它们，则内核可以会产生大量进程间中断(IPI)，同时所有核都从深度 C-states 返回。由于处理中断时需要锁定它，所以系统在处理所有中断过程中可能会停滞一段时间。这可能会导致应用程序在响应事件方面有大量延迟。

#### 例 34.2. 显示每个内核处于 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 花费的时间：

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%
C6 (pc6)	0.0%	C6 (cc6)	8.3%	C6	5.2%	0.6 ms	6.0%
C7 (pc7)	0.0%	C7 (cc7)	76.6%	C7s	0.0%	0.0 ms	0.0%
C8 (pc8)	0.0%			C8	6.3%	0.9 ms	5.8%
C9 (pc9)	0.0%			C9	0.4%	3.7 ms	2.2%
C10 (pc10)	0.0%			C10	80.8%	3.7 ms	79.4%
					4.4 ms		
				C1E	0.1%	0.1 ms	0.1 ms
	...						

#### 其他资源

- 使用 PowerTOP 管理能耗

#### 34.3.2. 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，空闲驱动程序决定内核是否使用固件中的设置：

- intel\_idle**：这是具有 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 **intel\_idle** 被禁用了，RHEL 在具有来自 Intel 以外的厂商的 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`

### 34.3.3. 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。内核空闲驱动程序可以与此接口进行通信，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一个最大 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
```

此自定义配置文件从 **network-latency** 配置文件继承所有设置。**force\_latency** TuneD 参数指定以微秒(μs)为单位的延迟。如果 C-state 延迟大于指定的值，则 Red Hat Enterprise Linux 中的空闲驱动程序会阻止 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，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 配置文件](#)

### 34.3.4. 使用内核命令行选项禁用 C-states

**processor.max\_cstate** 和 **intel\_idle.max\_cstate** 内核命令行参数配置可以使用的最大消耗状态(C-state) CPU 内核。例如，将参数设置为 1 确保 CPU 永远不会请求 C1 以下的 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。要不硬编码一个特定状态，请考虑使用更动态的解决方案。请参阅 [使用一个自定义 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** 来覆盖这些 C-state 设置。

3. 在独立于 CPU 厂商的每个主机上，设置 CPU 内核应该可以使用的最高 C-state：

```
# grubby --update-kernel=ALL --args="processor.max_cstate=0"
```



#### 重要

如果您除了设置 **intel\_idle.max\_cstate=0** 之外还设置了 **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，则显示最大 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？](#) (红帽知识库)
- [kernel-doc 软件包提供的 /usr/share/doc/kernel-doc-<version>/Documentation/admin-guide/pm/cpuidle.rst](#)

## 34.4. 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有 1518 字节的最大大小。这些帧的每一个都包含一个 18 字节标头，给有效负载保留 1500 字节。因此，对于服务器通过网络传输的每个 1500 字节数据，18 字节(1.2%)以太网帧标头是开销，且也会被传输。第 3 层和 4 协议中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，如备份服务器或存有大量巨大文件的文件服务器，则请考虑使用巨型帧来节省开销。巨型帧是非标准化的帧，其最大传输单位(MTU)比 1500 字节的标准以太网有效负载大小大。例如，如果您使用最大允许的 9000 字节有效负载配置 MTU 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）中启用巨型帧会很有帮助。这可避免数据包碎片。

### 34.4.1. 配置巨型帧前的注意事项

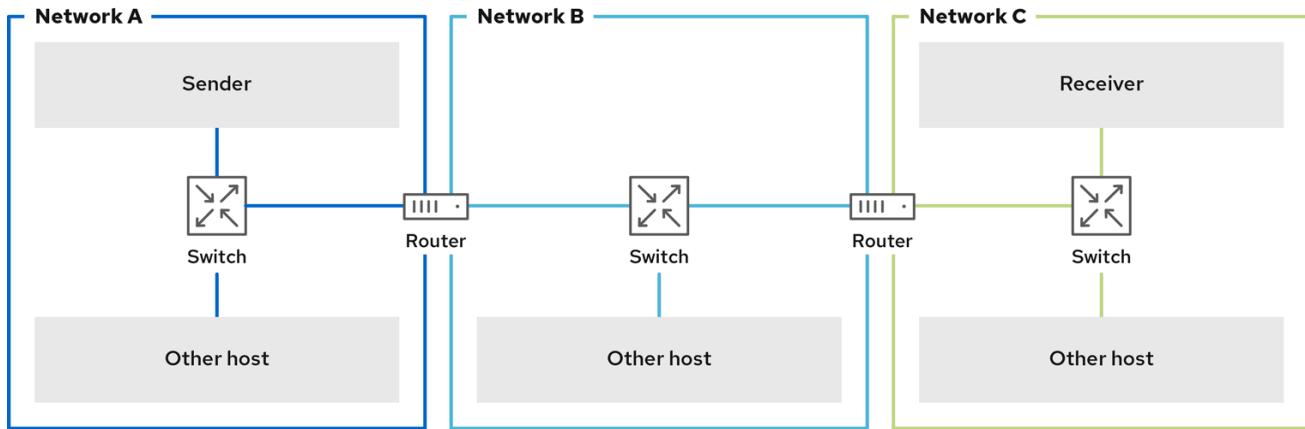
根据您网络中的硬件、应用程序和服务，巨型帧可能有不同的影响。仔细决定启用巨型帧是否能在您的场景中提供好处。

#### 先决条件

传输路径上的所有网络设备都必须支持巨型帧，并使用相同的最大传输单元(MTU)大小。相反，您可能遇到以下问题：

- 丢弃的数据包。
- 由于碎片数据包，导致更高的延迟。
- 增加了碎片导致的数据包丢失的风险。例如，如果路由器将一个 9000 字节帧划分为六个 1500 字节帧，且任何一个 1500 字节帧丢失了，则整个帧都丢失了，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 中的一个主机发送了一个数据包，则这三个子网中的所有主机都必须使用相同的 MTU：



308\_RHEL\_0223

## 巨型帧的好处

- 较高的吞吐量：每个帧包含更多的用户数据，而协议开销是固定的。
- 较低的 CPU 使用率：巨型帧导致较少的中断，因此节省了 CPU 周期。

## 巨型帧的缺陷

- 较高的延迟：大型帧延迟后面的数据包。
- 增加了内存缓冲区使用率：大型帧可以更快地填充缓冲区队列内存。

### 34.4.2. 在现有 NetworkManager 连接配置文件中配置 MTU

如果您的网络需要不同于默认的最大传输单元(MTU)，您可以在相应的 NetworkManager 连接配置文件中配置此设置。

巨型帧是有效负载在 1500 核 9000 字节之间的网络数据包。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相同的 MTU。
- 您知道网络的 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. 在管理到具有分散 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. 验证传输路径上没有主机对数据包进行碎片处理：

- 在接收端，显示内核的 IP 重新组装统计信息：

```
# 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**，则不会重新组装数据包。

- 在发送端，传输一个带有 prohibit-fragmentation-bit 的 ICMP 请求：

```
# ping -c1 -Mdo -s 8972 destination_host
```

如果命令成功，则数据包没有被进行碎片处理。

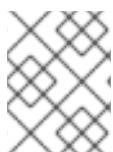
按如下所示计算 **-s** 数据包大小选项的值：MTU 大小 - 8 字节 ICMP 标头 - 20 字节 IPv4 标头 = 数据包大小

## 34.5. 为高吞吐量调优 TCP 连接

在 Red Hat Enterprise Linux 上调优与 TCP 相关的设置以提高吞吐量、缩短延迟或防止诸如数据包丢失等问题。

### 34.5.1. 使用 iperf3 测试 TCP 吞吐量

**iperf3** 工具提供了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，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** 软件包已安装在客户端和服务器上。
- 任何一个主机上都没有其他服务导致对测试结果产生严重影响的网络流量。
- 对于 40 Gbps 和更快的连接，网卡支持 Accelerated Receive Flow Steering(ARFS)，且该功能在接口上已启用。

#### 流程

1. 可选：显示服务器和客户端上网络接口控制器(NIC)的最大网络速度：

```
# ethtool enp1s0 | grep "Speed"
Speed: 100000Mb/s
```

2. 在服务器上：

- a. 临时在 **firewalld** 服务中打开默认的 **iperf3** TCP 端口 5201：

```
# firewall-cmd --add-port=5201/tcp
```

- b. 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. 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工作的值，并在稍后的测量中增加它。如果客户端以比传输路径上的设备快的速度，或者比服务器可以处理数据包快的速度发送数据包，则数据包可以丢弃。
- **--zerocopy**：启用零复制方法，而不是使用 **write()** 系统调用。只有在您要模拟可零复制的应用程序或在单个流上达到 40 Gbps 及更多时，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. 在服务器上：

- 按 **Ctrl+C** 停止 **iperf3** 服务器。
- 关闭 **firewall** 中的 TCP 端口 5201：

```
# firewall-cmd --remove-port=5201/tcp
```

## 其他资源

- 您系统上的 **iperf3 (1)** 手册页

### 34.5.2. 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的或应发送的数据：

- 读套接字缓冲区保留内核已收到的，但应用程序尚未读取的数据包。
- 写套接字缓冲区保留应用程序已写到缓冲区，但内核尚未传给 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太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阻止数据包丢失。

**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 以下方法使用它们：

- 第一个值是最小缓冲区大小。新套接字不能有更小的大小。
- 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的缓冲区的最大大小。在应用程序中使用带有 **SO\_SNDBUF** 套接字选项的 **setsockopt()** 函数可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。

### 34.5.3. 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接收或应发送的数据。**net.ipv4.tcp\_rmem**（读）和 **net.ipv4.tcp\_wmem**（写）套接字缓冲区内核设置各自包含三个设置：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

- 您遇到了大量的 TCP 数据包丢包率。

## 流程

- 确定连接的延迟。例如，从客户端 ping 服务器来测量平均 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。

- 使用以下公式为您要调优的流量计算 Bandwidth Delay Product(BDP)：

connection speed in bytes \* latency in ms = BDP in bytes

例如，要计算具有 117 ms 延迟的 10 Gbps 连接的 BDP：

$(10 * 1000 * 1000 * 1000 / 8) * 117 = 10683760 \text{ bytes}$

- 根据您的要求，创建 `/etc/sysctl.d/10-tcp-socket-buffers.conf` 文件，并设置最大读或写缓冲区大小，或者两者：

```
net.ipv4.tcp_rmem = 4096 262144 21367520
net.ipv4.tcp_wmem = 4096 24576 21367520
```

指定以字节为单位的值。当您尝试识别为您的环境优化的值时，请使用以下经验法则：

- 默认缓冲区大小（第二个值）：仅稍微增加此值或将其实设为 **524288** (512 KiB)。太高的默认缓冲区大小可能会导致缓冲区崩溃，因此延迟激增。
- 最大缓冲区大小（第三个值）：两到三倍 BDP 的值通常就足够了。

- 从 `/etc/sysctl.d/10-tcp-socket-buffers.conf` 文件载入设置：

```
# sysctl -p /etc/sysctl.d/10-tcp-socket-buffers.conf
```

- 配置应用程序以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。如果您不是应用的开发人员，请联系开发人员。

6. 如果您已更改了 **net.ipv4.tcp\_rmem** 或 **net.ipv4.tcp\_wmem** 参数中的第二个值, 请重启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, 因为自动调优会动态应用这些设置。

## 验证

1. 可选 : 使用 **iperf3** 测试 TCP 吞吐量。
2. 使用与您遇到数据包丢弃时相同的方法来监控数据包丢弃统计信息。  
如果数据包丢弃仍然发生, 但比率较低, 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影响是什么?](#) (红帽知识库)
- 您系统上的 **tcp(7)** 和 **socket (7)** 手册页

### 34.5.4. 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, 它显著提高了吞吐量。

例如, 在具有 1.5 ms 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, 吞吐量会下降到 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可以接收的尽可能多的数据, 但不能发送太多。为此, 接收方通告一个 **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, 但对于高 Bandwidth Delay Products(BDP), 这个值变为一个限制, 因为发送方一次不能发送超过 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: 一个在系统之间有 1 毫秒延迟的 10 Gbps 链接, 在给定时间内可以传输超过 1 MiB 的数据。如果主机只发送 64 KiB, 然后暂停, 直到其他主机收到这 64 KiB, 则这将是低效的。

要删除这个瓶颈, TCP 窗口扩展允许 TCP 窗口值算术左移, 来将窗口大小增加到 64 KiB 以上。例如, 最大窗口值 **65535** 向左移 7 个位置, 会产生几乎 8 MiB 的窗口大小。这可在给定时间内传输更多的数据。

TCP 窗口扩展在打开每个 TCP 连接的三向 TCP 握手中进行协商。发件方和接收方都必须支持 TCP 窗口扩展, 才能使其正常工作。如果两个参与者没有在握手中通告窗口扩展功能, 则连接恢复为使用最初的 16 位 TCP 窗口大小。

默认情况下, TCP 窗口扩展在 Red Hat Enterprise Linux 中被启用 :

```
# sysctl net.ipv4.tcp_window_scaling
net.ipv4.tcp_window_scaling = 1
```

如果在服务器上禁用了 TCP 窗口扩展(**0**), 则以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

### 34.5.5. 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，对于其接收的每个数据包，接收方都向发送方发送一个 ACK 数据包。例如，客户端向服务器发送 TCP 数据包 1-10，但数据包号 5 和 6 丢失了。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传所有数据包，这效率较低。如果两个主机上都启用了 TCP SACK，则客户端必须只重传丢失的数据包 5 和 6。



#### 重要

禁用 TCP SACK 会降低性能，并在 TCP 连接中，在接收端造成很高的数据包丢弃率。

RHEL 中默认启用了 TCP SACK。要验证：

```
# sysctl net.ipv4.tcp_sack
1
```

如果在您的服务器上禁用了 TCP SACK (0)，请以与您设置它的相同方式恢复设置。

#### 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？（红帽知识库）](#)
- 在运行时配置内核参数

## 34.6. 调优 UDP 连接

在开始调优 Red Hat Enterprise Linux 以改进 UDP 流量的吞吐量前，有一个现实的期望很重要。UDP 是一个简单协议。与 TCP 相比，UDP 不包含这些功能，如流控制、阻塞控制和数据可靠性。这样很难与接近网络接口控制器(NIC)的最大速度的吞吐量率通过 UDP 达到可靠的通信。

### 34.6.1. 检测数据包丢弃

网络堆栈中有多个级别，内核可以丢弃数据包。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丢弃的数据包速率。但是，如果您遇到很高的速率，请考虑调优措施。



#### 注意

如果网络堆栈无法处理传入的流量，则内核丢弃网络数据包。

#### 流程

- 识别特定于 UDP 协议的数据包丢弃，因为套接字缓冲区太小，或者应用程序处理速度太慢：

```
# 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](#) (红帽知识库)
- [我是否应该关注 0.05% 的数据包丢弃率？](#) (红帽知识库)

### 34.6.2. 使用 iperf3 测试 UDP 吞吐量

**iperf3** 工具提供了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测量的结果有很大不同。

## 先决条件

- **iperf3** 软件包已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：您在服务器和客户端上增加了最大 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：显示服务器和客户端上网络接口控制器(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，并记录此值：

```
# 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
```

请注意，**iperf3** 只在服务器上打开一个 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：

- 显示客户端将用来连接到服务器的接口的最大传输单元(MTU)，并记录此值：

```
# ip link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
...
...
```

- 显示最大 UDP 套接字写缓冲区大小，并记录此值：

```
# sysctl net.core.wmem_max
net.core.wmem_max = 16777216
```

显示的值以字节为单位。

- 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都一样。如果不一样，则将此参数设置为较小的值：客户端上的 **net.core.wmem\_max** 或服务器上的 **net.core.rmem\_max**。
- **--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未碎片的有效负载。按如下所示计算理想的值：MTU - IP 标头(20 字节用于 IPv4， 40 字节用于 IPv6) - 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工作的值，并在稍后的测量中增加它。如果客户端以比传输路径上的设备快的速度，或者比服务器可以处理数据包快的速度发送数据包，则数据包可以丢弃。
- **--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，平均波特率为 2 Gbps，没有数据包丢失。

## 5. 在服务器上：

- 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
```

## 其他资源

- 您系统上的 **iperf3 (1)** 手册页

### 34.6.3. 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有 1518 字节的最大大小。这些帧的每一个都包含一个 18 字节标头，给有效负载保留 1500 字节。因此，对于服务器通过网络传输的每 1500 字节数据，18 字节(1.2%)是开销。

巨型帧是非标准化的帧，其最大传输单位(MTU)比 1500 字节的标准以太网有效负载大小大。例如，如果您使用最大允许的 9000 字节有效负载配置 MTU 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所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，数据包碎片和组装减少了网络吞吐量。

不同的连接类型有一些 MTU 限制：

- 以太网：MTU 限制为 9000 字节。
- 数据报模式下的 IP over InfiniBand (IPoIB)：MTU 限制为比 InfiniBand MTU 少 4 个字节。
- 内存网络通常支持大型 MTU。详情请查看相应的文档。

### 34.6.4. CPU 速度对 UDP 流量吞吐量的影响

在批量传输中，UDP 协议比 TCP 效率低很多，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 UDP Fragmentation Offload (UFO) 功能。因此，CPU 频率可以限制在高速链路上批量传输的 UDP 吞吐量。

例如，在具有较高最大传输单元(MTU)和大型套接字缓冲区的调优后主机上，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 速度损失大约 1-2 Gbps 速度。另外，如果 3 GHz 的 CPU 速度可以取得接近 10 Gbps，则相同的 CPU 将 40 Gbit NIC 上的 UDP 流量限制在大约 20-25 Gbps。

### 34.6.5. 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的或应发送的数据：

- 读套接字缓冲区保留内核已收到的，但应用程序尚未读取的数据包。

- 写套接字缓冲区保留应用程序已写到缓冲区，但内核尚未传给 IP 堆栈和网络驱动程序的数据包。

如果 UDP 数据包太大，且超过缓冲区大小或数据包以太快的速度发送或接收，则内核会丢弃任何新传入的 UDP 数据包，直到数据从缓冲区中删除为止。在这种情况下，增加套接字缓冲区可以阻止数据包丢失。



## 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请求的大小，内核会加倍这个值。例如，如果应用程序请求一个 256 KiB 套接字缓冲区大小，并打开 100 万个套接字，则系统只需要 512 GB RAM (512 KiB × 100万)用于潜在的套接字缓冲区空间。

## 先决条件

- 您遇到了明显的 UDP 数据包丢弃率。

## 流程

- 创建 `/etc/sysctl.d/10-udp-socket-buffers.conf` 文件，并根据您的要求设置最大读或写缓冲区大小，或者两者：

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
```

指定以字节为单位的值。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)。

- 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
```

- 将应用程序配置为使用大型套接字缓冲大小。

`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。如果您不是应用的开发人员，请联系开发人员。

- 重启应用程序以使用新的 UDP 缓冲区大小。

## 验证

- 使用与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。  
如果数据包丢弃仍然发生，但比率较低，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影响是什么？](#) (红帽知识库)
- 您系统上的 `udp(7)` 和 `socket (7)` 手册页

## 34.7. 识别应用程序读套接字缓冲区瓶颈

如果 TCP 应用程序无法频繁清除读套接字缓冲区，则性能可能会受到影响，数据包可能会丢失。Red Hat Enterprise Linux 提供了不同的工具来识别此类问题。

### 34.7.1. 识别接收缓冲区折叠和修剪

当接收队列中的数据超过接收缓冲区大小时，TCP 堆栈尝试从套接字缓冲区中删除不必要的元数据来释放一些空间。此步骤被称为折叠。

如果折叠无法为额外的流量释放足够的空间，则内核修剪到达的新数据。这意味着内核从内存中删除数据，数据包丢失。

为避免折叠和修剪操作，请监控 TCP 缓冲区是否在服务器上发生了折叠和修剪，在这种情况下，请调优 TCP 缓冲区。

#### 流程

- 使用 **nstat** 工具查询 **TcpExtTCPRecvCollapsed** 和 **TcpExtRcvPruned** 计数器：

```
# nstat -az TcpExtTCPRecvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned      0    0.0
TcpExtTCPRecvCollapsed   612859  0.0
```

- 等待一些时间，重新运行 **nstat** 命令：

```
# nstat -az TcpExtTCPRecvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned      0    0.0
TcpExtTCPRecvCollapsed   620358  0.0
```

- 如果与第一次运行时相比计数器的值增加了，则需要调优：

- 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优其大小的能力。
- 如果应用程序没有使用 **setsockopt (SO\_RCVBUF)** 调用，请调优 TCP 读套接字缓冲区的默认值和最大值。

- 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 ...
...
```

- 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** 列中只列出一个高值，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** 持续增长期间保持恒定，或者 **Recv-Q** 随着时间的推移而持续增加，则以下几个问题是原因：

- 应用程序检查其套接字缓冲区的频率不够。请联络应用程序厂商来了解有关如何解决这个问题的详细信息。
- 应用程序没有获得足够的 CPU 时间。要进一步调试这个问题：
  - 显示应用程序在哪个 CPU 核上运行：

```
# 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** 列显示进程当前分配给的 CPU 核。

- 识别运行在同一核上的其他进程，并考虑将它们分配给其他核。

## 其他资源

- [增加系统范围的 TCP 套接字缓冲区](#)

## 34.8. 调优具有大量传入请求的应用程序

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，则有必要调整 Red Hat Enterprise Linux 来优化性能。

### 34.8.1. 调优 TCP 倾听积压日志，来处理大量 TCP 连接尝试

当应用程序打开 **LISTEN** 状态的 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，则新的连接会丢失，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>** 错误消息。
- 大量连接尝试来自有效源，不是攻击造成的。

## 流程

1. 要验证是否需要调优，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:*
```

如果积压中的当前连接数(**Recv-Q**)大于套接字积压(**Send-Q**)，则侦听积压仍不够大，需要调优。

- 可选：显示当前 TCP 侦听积压限制：

```
# sysctl net.core.somaxconn
net.core.somaxconn = 4096
```

- 创建 **/etc/sysctl.d/10-socket-backlog-limit.conf** 文件，并设置更大的监听积压限制：

```
net.core.somaxconn = 8192
```

请注意，应用程序可以请求比 **net.core.somaxconn** 内核参数中指定的大的侦听积压，但内核将应用程序限制为您在此参数中设置的数值。

- 从 **/etc/sysctl.d/10-socket-backlog-limit.conf** 文件中载入设置：

```
# sysctl -p /etc/sysctl.d/10-socket-backlog-limit.conf
```

- 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一个配置选项，请更新它。例如，Apache HTTP 服务器提供了 **ListenBacklog** 配置选项，以便为该服务设置监听积压限制。
- 如果无法配置限制，请重新编译应用程序。



### 重要

您必须始终更新 **net.core.somaxconn** 内核设置和应用程序的设置。

- 重新启动应用程序。

## 验证

- 监控 Systemd 日志，来查看以后是否出现 **possible SYN flooding on port <port\_number>** 错误消息。
- 监控积压中的当前连接数，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:*
```

如果积压中的当前连接数(**Recv-Q**)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 泛滥。发送 cookie
- 监听 TCP 服务器会忽略新的连接握手的 SYN 或 ACK（红帽知识库）
- 您系统上的 **listen (2)** 手册页

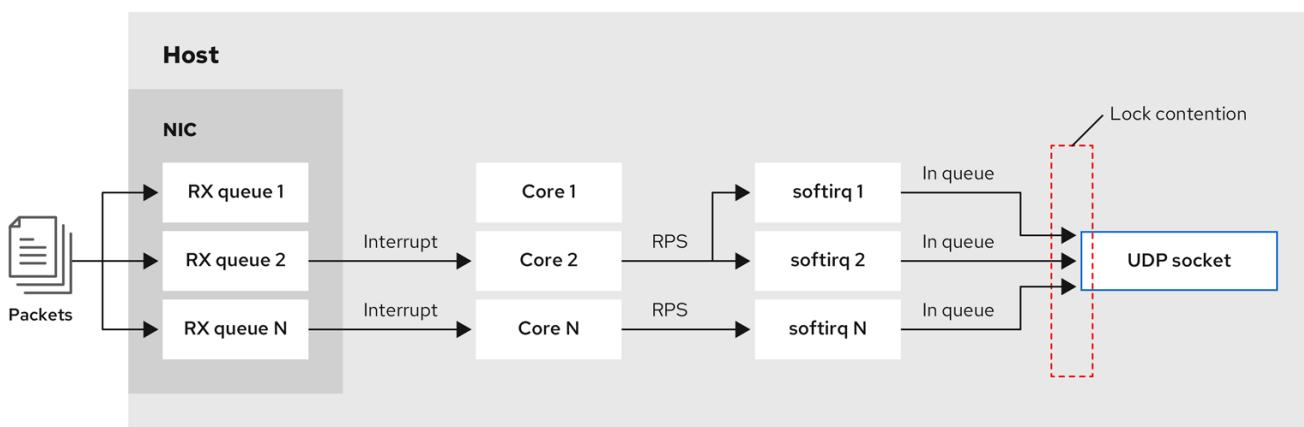
## 34.9. 避免侦听队列锁争用

队列锁争用可能导致数据包丢弃和更高的 CPU 使用率，因此导致更高的延迟。您可以通过调优应用程序并使用传输数据包控制，来避免接收(RX)和传输(TX)队列上的队列锁争用。

### 34.9.1. 避免 RX 队列锁争用 : SO\_REUSEPORT 和 SO\_REUSEPORT\_BPF 套接字选项

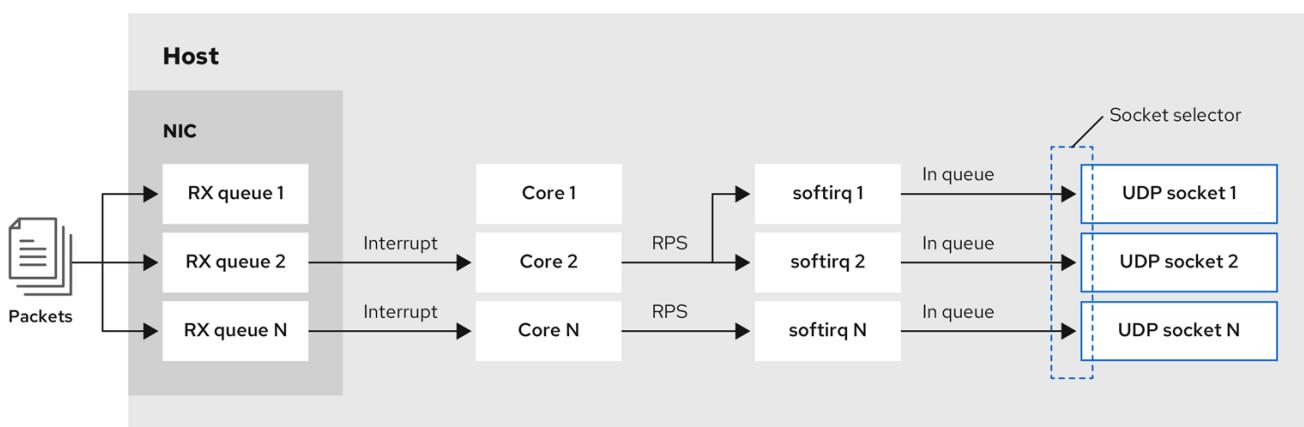
在多核系统上，如果应用程序使用 **SO\_REUSEPORT** 或 **SO\_REUSEPORT\_BPF** 套接字选项打开了端口，您可以提高多线程网络服务器应用程序的性能。如果应用程序不使用其中一个套接字选项，则所有线程都被强制共享单个套接字来接收传入流量。使用单个套接字会导致：

- 对接收缓冲区的显著竞争，这可能会导致数据包丢弃和更高的 CPU 使用率。
- CPU 使用率的显著增加
- 可能的数据包丢弃



316\_RHEL\_0323

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：



316\_RHEL\_0323

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-8-for-x86\_64-baseos-debug-rpms** 存储库：

```
# subscription-manager repos --enable rhel-8-for-x86_64-baseos-debug-rpms
```

- 安装 **kernel-debuginfo-common-x86\_64** 软件包：

```
# yum install kernel-debuginfo-common-x86_64
```

- 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

- 您系统上的 **socket (7)** 手册页
- `/usr/src/debug/kernel-<version>/linux-<version>/tools/testing/selftests/net/reuseport_bpf_cpu.c`

### 34.9.2. 避免 TX 队列锁争用：传输数据包控制

在具有支持多个队列的网络接口控制器(NIC)的主机中，传输数据包控制(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 处理传出网络流量，并避免传输队列锁争用，从而避免数据包丢弃。

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，请咨询 NIC 驱动程序文档。请咨询您的 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 自动调优，您可以手动将 CPU 核分配给传输队列。



#### 注意

Red Hat Enterprise Linux 不提供一个将传输队列永久分配给 CPU 核的选项。使用在接口被激活时执行的 NetworkManager 分配程序脚本中的命令。详情请参阅 [如何编写一个 NetworkManager 分配程序脚本，以便在接口启动时应用命令](#)。

## 先决条件

- NIC 支持多个队列。
- numactl** 软件包已安装。

## 流程

- 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。

- 可选：如果您需要特定通道上的队列，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
```

- 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。

- 如果主机是 NUMA 系统，显示哪些 CPU 被分配给了哪个 NUMA 节点：

```
# lscpu | grep NUMA
NUMA node(s): 2
NUMA node0 CPU(s): 0-3
NUMA node1 CPU(s): 4-7
```

- 在上例中，NIC 有 4 个队列，NIC 被分配给 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 队列上的任何类型的争用。否则，如果您在同一 TX 队列上映射多个 CPU，则不同 CPU 上的传输操作将导致 TX 队列锁竞争，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

- 识别发送流量的服务的进程 ID (PID)：

```
# pidof <process_name>
12345 98765
```

- 将 PID 固定到使用 XPS 的核：

```
# numactl -C 0-3 12345 98765
```

- 在进程发送流量时监控 **queues** 计数器：

```
# tc -s qdisc
qdisc fq_codel 0: dev enp1s0u1 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** 计数器不再以显著速率增加，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](#)

### 34.9.3. 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，且无法添加该功能。
- TCP 吞吐量不相关。

#### 警告

禁用 GRO 可显著降低 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，请在 NetworkManager 配置文件中重新启用 GRO。

## 其他资源

- 在 RHEL 8.5 中提高 UDP 性能

## 34.10. 调优设备驱动程序和 NIC

在 RHEL 中，内核模块为网络接口控制器(NIC)提供驱动程序。这些模块支持调优和优化设备驱动程序及 NIC 的参数。例如，如果驱动程序支持延迟接收中断的生成，您可以降低相应参数的值，以避免耗尽接收描述符。



### 注意

并非所有模块都支持自定义参数，功能依赖于硬件，以及驱动程序和固件版本。

### 34.10.1. 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

如果您在内核模块上设置了参数，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

- 已在主机上安装了 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与用于您要更改参数的驱动程序不同的网络接口。

## 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



卸载活跃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. 显示内核消息：

```
# 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。

- 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。这些文件的每一个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>
```

## 34.11. 配置网络适配器卸载设置

要减少 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 执行 ESP 操作来加快 IPsec 连接，并减少 CPU 负载。

默认情况下，Red Hat Enterprise Linux 中的大多数卸载功能都被启用了。只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。

### 34.11.1. 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，您可以通过临时启用或禁用它来尝试缩小原因范围，具体取决于其当前状态。

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之前的值。

#### 先决条件

- 网卡支持卸载功能。

#### 流程

- 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]** 的功能状态。

- 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. 在更改卸载功能之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，问题是否不再存在：
  - 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。
- 如果在禁用了特定卸载功能后问题仍然存在：
  - i. 使用 `ethtool -K <interface> <feature> [on/off]` 命令将设置重置回其之前的状态。
  - ii. 启用或禁用不同的卸载功能来缩小问题范围。

## 其他资源

- 您系统上的 [ethtool \(8\)](#) 手册页

### 34.11.2. 永久设置卸载功能

如果您已确定了在主机上限制性能的特定卸载功能，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了卸载功能，则 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在主机上限制性能。

## 流程

1. 识别使用您要更改卸载功能状态的网络接口上的连接配置文件：

```
# 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** 连接配置文件中永久禁用 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** 连接配置文件中永久启用 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

- 您系统上的 **nm-settings-nmcli (5)** 手册页

## 34.12. 调优中断合并设置

中断合并是一种减少网卡产生的中断数的机制。通常，较少的中断可能会提高网络的延迟和整体性能。

调优中断合并设置涉及调整其控制的参数：

- 合并到单个中断的数据包数量。
- 产生中断前的延迟。



### 重要

最佳合并设置取决于特定的网络条件和在使用的硬件。因此，可能需要多次尝试才能找到最适合您环境和需要的设置。

### 34.12.1. 为延迟或吞吐量敏感的服务优化 RHEL

合并调优的目的是最小化给定工作负载所需的中断数量。在高吞吐量情况下，目标是有尽可能少的中断，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，以增加或减少组合到单个中断的数据包数。因此，您可以提高流量的吞吐量或延迟。

## 流程

- 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
```

识别在其名称中包含 **drop**、**discard** 或 **error** 数据包计数器。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 NIC 合并造成的。

- 监控您在上一步中标识的数据包计数器的值。

将它们与网络的期望值进行比较，以确定任何特定接口是否遇到了瓶颈。网络瓶颈的一些常见迹象包括但不限于：

- 网络接口上的许多错误
- 高数据包丢失
- 网络接口的大量使用



### 注意

在识别网络瓶颈时，其他重要因素包括 CPU 使用率、内存使用率和磁盘 I/O。

- 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送方在产生中断前等待的微秒数。
- frames** 值指的是接收方或传送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** 值代表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 **Adaptive RX/TX** 启用时，NIC 驱动程序会自动计算合并值（每个 NIC 驱动程序的算法都不一样）。

4. 根据需要修改合并设置。例如：

- 在 **ethtool.coalesce-adaptive-rx** 被禁用时，请将 **ethtool.coalesce-rx-usecs** 配置，来在产生中断前将 RX 数据包的延迟设为 100 微秒：

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-usecs 100
```

- 启用 **ethtool.coalesce-adaptive-rx**，而 **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**，而不会造成任何损害。如果他们不希望使用自适应中断合并机制，他们可以尝试将 **ethtool.coalesce-rx-usecs** 设置为大值，如 100Us 或 250Us。
- 在出现问题前，不确定其需求的用户不应修改此设置。

5. 重新激活连接：

```
# nmcli connection up enp1s0
```

### 验证

- 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 或接近于 0，取决于网络流量和系统资源。这些字段中的高值表示有网络问题。您的计数器可以有不同的名称。严格监控其名称中包含“drop”、“discard”或“error”的数据包计数器。

**rx\_packets**、**tx\_packets**、**rx\_bytes** 和 **tx\_bytes** 的值应该随时间的推移而增加。如果值没有增加，则可能有网络问题。数据包计数器可以有不同的名称，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 和驱动程序而变化。

专注于非常低延迟的用户可以使用应用程序级指标或内核数据包时间截 API 来进行监控。

## 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？（红帽知识库）](#)
- [如何使 NIC ethtool 设置持久（在引导时自动应用）（红帽知识库）](#)
- [时间截](#)

### 34.13. TCP 时间截的好处

TCP 时间截在 TCP 标头中是可选信息，是 TCP 协议的扩展。TCP 时间截在 Red Hat Enterprise Linux 中默认启用，内核使用 TCP 时间截来更好地估算 TCP 连接中的往返时间(RTT)。这导致更准确的 TCP 窗口和缓冲区计算。

另外，TCP 时间截提供了一种替代方法来确定片段的寿命和顺序，并防止包裹的序列号。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 秒后包裹。没有 TCP 时间截，接收方无法决定带有包裹的序列号的片段是否是一个新的片段或旧的重复片段。但是，使用 TCP 时间截，接收方可以做出正确的选择来接收或丢弃片段。因此，在带有快速网络接口的系统上启用 TCP 时间截非常重要。

**net.ipv4.tcp\_timestamps** 内核参数可以有以下值之一：

- **0**：TCP 时间截被禁用。
- **1**：TCP 时间截被启用（默认）。
- **2**：TCP 时间截被启用，但没有随机偏移。



### 重要

如果每个连接没有随机偏移，则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。

默认在 Red Hat Enterprise Linux 中启用了 TCP 时间截，并为每个连接使用随机偏移，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), 则以与您设置其相同的方式恢复设置。

## 其他资源

- [RFC 1323 : 高性能的 TCP 扩展](#)

### 34.14. 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持续的数据传输可能会导致缓冲区容量满了。缓冲区容量满了会导致网络拥塞。在这种情况下, 当发送方传输数据的速度高于接收方的处理能力时, 可能会因为链接另一端 (其是交换机端口) 上网络接口的低数据处理能力而发生数据包丢失。

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, 以便管理交换机端口的高传输率。请注意, 交换机不会转发交换机端口之外的暂停帧。

当接收(RX)缓冲区变满时, 接收方会向传送方发送暂停帧。然后, 传送方会停止数据传输一个亚秒时间段, 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提供了足够时间来清空其接口缓冲区, 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个端发送暂停帧。如果网络接口的接收缓冲区已满, 网络接口将向交换机端口发送暂停帧。类似, 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 中的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
...
```

与您的交换机厂商确认您的交换机是否支持暂停帧。

## 其他资源

- 您系统上的 **ethtool (8)** 手册页
- [什么是网络链路流控制, 以及它在 Red Hat Enterprise Linux 中如何工作? \(红帽知识库\)](#)

## 第 35 章 配置操作系统以优化内存访问

您可以配置操作系统，来使用 RHEL 中包含的工具优化工作负载间的内存访问。

### 35.1. 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 8 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 **yum 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

- 您系统上的 **vmstat (8)** 和 **valgrind (1)** 手册页
- **/usr/share/doc/valgrind-version/valgrind\_manual.pdf** 文件

### 35.2. 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

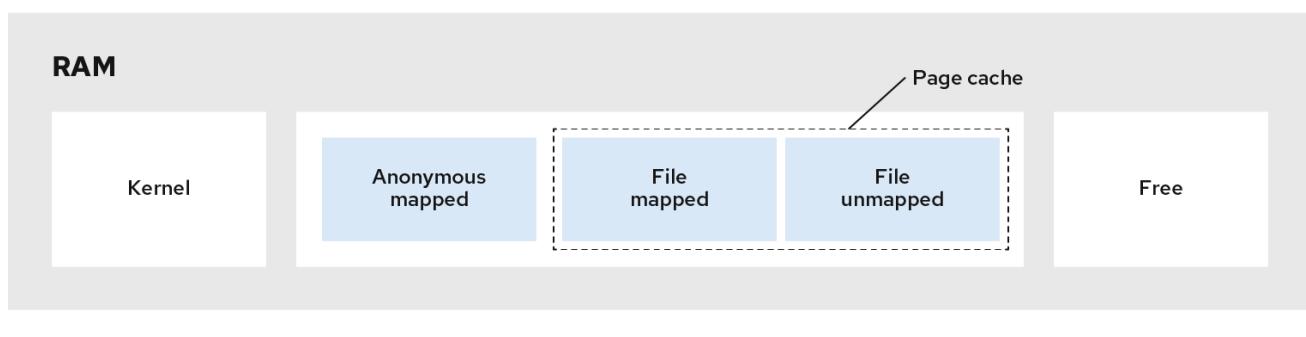
这个类别中添加的页面代表持久性存储中的部分文件。这些页面缓存中的页面可以在应用程序的地址空间中映射或取消映射。您可以使用应用程序将文件映射到其地址空间，或使用 **mmap** 系统调用，或通过缓冲的 I/O 读写系统调用处理文件。

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 35.1. 内存用量模式



133\_RHEL\_0121

## 35.3. 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。当总系统内存的此百分比被修改时，系统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。
- 如果您使用 **cgroupsV1**，则 **cgroupsV1** 独有的每个 cgroup swappiness 值将导致 **vm.swappiness** 参数配置的系统范围的 swappiness 对系统的交换行为几乎没有影响。此问题可能导致意外的和不一致的交换行为。  
在这种情况下，请考虑使用 **vm.force\_cgroup\_v2\_swappiness** 参数。

如需更多信息，请参阅红帽知识库解决方案 [在仍有大量页缓存需要回收时，过早使用 swappiness=0 进行交换](#)。

### force\_cgroup\_v2\_swappiness

此控件用于弃用仅 **cgroupV1** 中可用的每个组 swappiness 值。所有系统和用户进程在 cgroup 中运行。cgroup 交换值默认为 60。这可能会导致系统交换值对系统的交换行为的影响很小。如果用户不关心每个组交换传递功能，他们可以使用 **force\_cgroup\_v2\_swappiness=1** 配置其系统，以便在整个系统中具有更加一致的交换支配行为。

### 其他资源

- 您系统上的 **sysctl (8)** 手册页
- [设置与内存相关的内核参数](#)

## 35.4. 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 8 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

- 您系统上的 **sysctl (8)** 手册页

## 35.5. 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。

### **shmmmax**

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 shmmmax** 命令确定系统中的当前 **shmmmax** 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

- 您系统行 **sysvipc (7)** 和 **sysctl (8)** 手册页

## 35.6. 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

- 您系统上 **sysctl (8)** 和 **proc (5)** 手册页

## 其他资源

- [为 IBM DB2 调优 Red Hat Enterprise Linux \(红帽知识库\)](#)

## 第 36 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 8 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 8 中巨页的信息以及如何配置它们。

### 36.1. 可用的巨页功能

在 Red Hat Enterprise Linux 8 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 8 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。

有关在引导时影响 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来在引导时配置 HugeTLB 页，请参阅 [在引导时配置 HugeTLB 页](#)。

- 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 页](#)。

#### 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](#)。

### 36.2. 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 36.1. 用于在引导时配置 HugeTLB 页面的参数**

参数	描述	默认值
<b>hugepages</b>	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 <code>/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages</code> 文件中的节点值，在运行时为特定节点分配巨页。	默认值为 <b>0</b> 。  要在引导时更新这个值，在 <code>/proc/sys/vm/nr_hugepages</code> 文件中更改这个参数的值。
<b>hugepagesz</b>	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 <b>2 MB</b> 和 <b>1 GB</b> 。默认值为 <b>2 MB</b> 。
<b>default_hugepagesz</b>	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 <b>2 MB</b> 和 <b>1 GB</b> 。默认值为 <b>2 MB</b> 。

### 36.3. 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

- 要为 **1 GB** 页创建一个 HugeTLB 池，请启用 **default\_hugepagesz=1G** 和 **hugepagesz=1G** 内核选项：

```
# grub2 --update-kernel=ALL --args="default_hugepagesz=1G hugepagesz=1G"
```

- 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
```

- 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
```

- 创建可执行脚本：

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

- 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，为了避免由于内存碎片而导致失败，请在引导过程的早期保留 **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

- 您系统上的 **systemd.service (5)** 手册页

- /usr/share/doc/kernel-doc-kernel\_version/Documentation/vm/hugetlbpage.txt 文件

## 36.4. 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 36.2. 在运行时配置 HugeTLB 页面的参数

参数	描述	文件名
<b>nr_hugepages</b>	定义分配给指定 NUMA 节点的指定大小的巨页数量。	/sys/devices/system/node/no_de_id/hugepages/hugepages-size/nr_hugepages
<b>nr_overcommit_hugepages</b>	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

## 36.5. 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

- 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

- 您系统上的 **numastat (8)** 手册页

## 36.6. 管理透明巨页

透明巨页(THP)在 Red Hat Enterprise Linux 8 中默认启用。但是，您可以启用、禁用或使用运行时配置、TuneD 配置文件、内核命令行参数或 systemd 单元文件将透明巨页设置为 **madvise**。

### 36.6.1. 使用运行时配置管理透明巨页

可在运行时管理透明巨页(THP)，以优化内存使用率。系统重启后，运行时配置不是持久的。

## 流程

- 检查 THP 的状态：

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
```

- 配置 THP.

- 启用 THP：

```
$ echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

- 禁用 THP：

```
$ echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

- 将 THP 设置为 **madvise**：

```
$ echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

要防止应用程序分配比需要多的内存资源，请禁用系统范围的透明巨页，并只为通过 **madvise** 系统调用明确请求它的应用程序启用它们。



## 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。另外，禁用直接压缩只允许 **madvise** 中突出显示的虚拟内存区域(VMAs)的异步压缩。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 never > /sys/kernel/mm/transparent_hugepage/defrag
```

## 其他资源

- 您系统上的 **madvise (2)** 手册页。

### 36.6.2. 使用 TuneD 配置文件管理透明巨页

您可以使用 TuneD 配置文件管理透明巨页(THP)。**tuned.conf** 文件提供 TuneD 配置文件的配置。这个配置在系统重启后保持不变。

#### 先决条件

- **TuneD** 软件包已安装。
- **TuneD** 服务已启用。

#### 流程

1. 将活跃配置文件复制到同一目录中：

```
$ sudo cp -R /usr/lib/tuned/my_profile /usr/lib/tuned/my_copied_profile
```

2. 编辑 **tune.conf** 文件：

```
$ sudo vi /usr/lib/tuned/my_copied_profile/tuned.conf
```

- 要启用 THP，请添加行：

```
[bootloader]
cmdline = transparent_hugepage=always
```

- 要禁用 THP，请添加行：

```
[bootloader]
cmdline = transparent_hugepage=never
```

- 要将 THP 设置为 **madvise**，请添加行：

```
[bootloader]
cmdline = transparent_hugepage=madvise
```

3. 重启 **Tuned** 服务：

```
$ sudo systemctl restart tuned
```

4. 将新配置文件设置为活跃：

```
$ sudo tuned-adm profile my_copied_profile
```

## 验证

1. 验证新配置文件是否处于活跃状态：

```
$ sudo tuned-adm active
```

2. 验证是否设置了所需的 THP 模式：

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
```

### 36.6.3. 使用内核命令行参数管理透明巨页

您可以通过修改内核参数，在引导时管理透明巨页(THP)。这个配置在系统重启后保持不变。

#### 前提条件

- 您在系统上具有 root 权限。

#### 流程

1. 获取当前的内核命令行参数：

```
# grubby --info=$(grubby --default-kernel)
kernel="/boot/vmlinuz-4.18.0-553.el8_10.x86_64"
args="ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX console=tty0 console=ttyS0"
root="UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
initrd="/boot/initramfs-4.18.0-553.el8_10.x86_64.img"
title="Red Hat Enterprise Linux (4.18.0-553.el8_10.x86_64) 8.10 (Ootpa)"
id="XXXXXXXXXXXXXXXXXXXXXXXXXXXX-4.18.0-553.el8_10.x86_64"
```

2. 通过添加内核参数来配置 THP。

- 要启用 THP：

```
# grubby --args="transparent_hugepage=always" --update-kernel=DEFAULT
```

- 要禁用 THP：

```
# grubby --args="transparent_hugepage=never" --update-kernel=DEFAULT
```

- 要将 THP 设置为 **madvise** :

```
# grubpy --args="transparent_hugepage=madvise" --update-kernel=DEFAULT
```

3. 重启系统以使更改生效 :

```
# reboot
```

## 验证

- 要验证 THP 的状态, 请查看以下文件 :

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

```
# grep AnonHugePages: /proc/meminfo
AnonHugePages:      0 kB
```

```
# grep nr_anon_transparent_hugepages /proc/vmstat
nr_anon_transparent_hugepages 0
```

### 36.6.4. 使用 **systemd** 单元文件管理透明巨页

您可以使用 **systemd** 单元文件, 在系统启动时管理透明巨页(THP)。通过创建一个 **systemd** 服务, 您可以在系统重启后获得一致的 THP 配置。

#### 前提条件

- 您在系统上具有 root 权限。

#### 流程

1. 创建新的 **systemd** 服务文件, 以启用、禁用 THP, 并将 THP 设置为 **madvise**。例如 : **/etc/systemd/system/disable-thp.service**。
  2. 通过向新的 **systemd** 服务文件中添加以下内容来配置 THP。
- 要启用 THP, 请将以下内容添加到 **<new\_thp\_file>.service** 文件中 :

```
[Unit]
Description=Enable Transparent Hugepages
After=local-fs.target
Before=sysinit.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/bin/sh -c 'echo always > /sys/kernel/mm/transparent_hugepage/enabled

[Install]
WantedBy=multi-user.target
```

- 要禁用 THP，请将以下内容添加到 `<new_thp_file>.service` 文件中：

```
[Unit]
Description=Disable Transparent Hugepages
After=local-fs.target
Before=sysinit.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/bin/sh -c 'echo never > /sys/kernel/mm/transparent_hugepage/enabled

[Install]
WantedBy=multi-user.target
```

- 要将 THP 设置为 `madvise`，请将以下内容添加到 `<new_thp_file>.service` 文件中：

```
[Unit]
Description=Madvise Transparent Hugepages
After=local-fs.target
Before=sysinit.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/bin/sh -c 'echo madvise > /sys/kernel/mm/transparent_hugepage/enabled

[Install]
WantedBy=multi-user.target
```

### 3. 启用并启动服务：

```
# systemctl enable <new_thp_file>.service
# systemctl start <new_thp_file>.service
```

### 验证

- 要验证 THP 的状态，请查看以下文件：

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
```

### 36.6.5. 其他资源

- 您还可以通过设置 TuneD 配置文件或使用预定义的 TuneD 配置文件来禁用透明巨页(THP)。请参阅 [与 RHEL 一起分发的 TuneD 配置文件](#) 和 [可用的 TuneD 插件](#)。

## 36.7. 页面大小对转换后备缓冲区大小的影响

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 37 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 Linux 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 Linux 系统中的行为。

### 37.1. 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。

### 37.2. 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：

```
# yum 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：

```
# yum 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

- 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）。

### 37.3. 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。

### 37.4. 运行 SYSTEMTAP 脚本

您可以从标准输入或从文件运行 SystemTap 脚本。

SystemTap 安装附带的样本脚本可在 **/usr/share/systemtap/examples** 目录中找到。

#### 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：

```
# echo "probe timer.s(1) {exit()}" | stap -
```

此命令指示 **stap** 运行通过 **echo** 传递给标准输入的脚本。要添加 **stap** 选项，请在 **-** 字符前面插入它们。例如，要使这个命令的结果更加详细，该命令是：

```
# echo "probe timer.s(1) {exit()}" | stap -v -
```

- 从文件中：

```
# stap file_name
```

# 第 38 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。

## 38.1. 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，企业策略可能禁止您安装提供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。

## 38.2. 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：

```
# yum 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 8），它们可以在不同的次版本中运行（如 8.1 和 8.2）。



## 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

- 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。

- 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。
- 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：

- 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。

- 手动：

- 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。

- 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
```

# 第 39 章 使用 SYSTEMTAP 监控网络活动

在安装 **systemtap-testsuite** 软件包时，您可以使用 **/usr/share/systemtap/testsuite/systemtap.examples/** 目录中提供的实用示例 SystemTap 脚本来监控和调查您系统的网络活动。

## 39.1. 使用 SYSTEMTAP 分析网络活动

您可以使用 **nettop.stp** 示例 SystemTap 脚本来对网络活动进行性能分析。脚本会跟踪系统中生成网络流量的进程，并提供有关每个进程的以下信息：

### PID

列出进程的 ID。

### UID

用户 ID。用户 ID 0 代表 root 用户。

### DEV

进程用于发送或接收数据的以太网设备（如 eth0、eth1）。

### XMIT\_PK

进程传输的数据包数。

### RECV\_PK

进程接收的数据包数。

### XMIT\_KB

进程发送的数据量（以 KB 为单位）。

### RECV\_KB

服务接收的数据量（以 KB 为单位）。

## 先决条件

- 如[安装 SystemTap](#) 所述，已安装了 SystemTap。

## 流程

- 运行 **nettop.stp** 脚本：

```
# stap --example nettop.stp
```

**nettop.stp** 脚本每 5 秒提供网络配置集抽样。

**nettop.stp** 脚本的输出类似如下：

```
[...]
PID  UID  DEV   XMIT_PK  RECV_PK  XMIT_KB  RECV_KB  COMMAND
 0    0  eth0     0      5      0      0  swapper
11178  0  eth0     2      0      0      0  synergyc
PID  UID  DEV   XMIT_PK  RECV_PK  XMIT_KB  RECV_KB  COMMAND
2886  4  eth0    79      0      5      0  cups-polld
11362  0  eth0     0     61      0      5  firefox
 0    0  eth0     3     32      0      3  swapper
2886  4  lo       4      4      0      0  cups-polld
```

```

11178 0 eth0      3   0   0   0 synergyc
PID  UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
  0  0 eth0      0   6   0   0 swapper
2886  4 lo       2   2   0   0 cups-polld
11178  0 eth0      3   0   0   0 synergyc
3611  0 eth0      0   1   0   0 Xorg
PID  UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
  0  0 eth0      3   42   0   2 swapper
11178  0 eth0      43   1   3   0 synergyc
11362  0 eth0      0   7   0   0 firefox
3897  0 eth0      0   1   0   0 multiload-apple

```

## 39.2. 使用 SYSTEMTAP 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、详细地识别每个进程如何与内核级别的网络交互。

### 先决条件

- 如[安装 SystemTap](#)所述，已安装了 SystemTap。

### 流程

- 运行 **socket-trace.stp** 脚本：

```
# stap --example socket-trace.stp
```

**socket-trace.stp** 脚本输出的 3 秒摘录类似如下：

```

[...]
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 gnome-terminal(11106): -> sock_poll
5 gnome-terminal(11106): <- sock_poll
0 scim-bridge(3883): -> sock_poll
3 scim-bridge(3883): <- sock_poll
0 scim-bridge(3883): -> sys_socketcall
4 scim-bridge(3883): -> sys_recv
8 scim-bridge(3883): -> sys_recvfrom
12 scim-bridge(3883):-> sock_from_file
16 scim-bridge(3883):<- sock_from_file
20 scim-bridge(3883):-> sock_recvmsg
24 scim-bridge(3883):<- sock_recvmsg
28 scim-bridge(3883): <- sys_recvfrom
31 scim-bridge(3883): <- sys_recv
35 scim-bridge(3883): <- sys_socketcall
[...]
```

## 39.3. 使用 SYSTEMTAP 监控网络数据包丢弃

Linux 中的网络堆栈可以丢弃由于各种原因的数据包。有些 Linux 内核包括一个追踪点 **kernel.trace ("kfree\_skb")**，其跟踪数据包在哪里被丢弃。

**dropwatch.stp** SystemTap 脚本使用 **kernel.trace ("kfree\_skb")** 来跟踪数据包丢弃；脚本总结了每 5 秒间隔丢弃数据包的位置。

## 先决条件

- 如[安装 SystemTap](#)所述，已安装了 SystemTap。

## 流程

- 运行 **dropwatch.stp** 脚本：

```
# stap --example dropwatch.stp
```

运行 **dropwatch.stp** 脚本 15 秒的结果类似如下：

```
Monitoring for dropped packets
51 packets dropped at location 0xffffffff8024cd0f
2 packets dropped at location 0xffffffff8044b472
51 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
97 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
Stopping dropped packet monitor
```

## 注意

要使数据包的位置更有意义，请参阅 **/boot/System.map-\$ (uname -r)** 文件。此文件列出了每个功能的起始地址，允许您将 **dropwatch.stp** 脚本输出中的地址映射到特定功能名称。以下的 **/boot/System.map-\$ (uname -r)** 文件的代码片段中，地址 **0xffffffff8024cd0f** 映射到功能 **unix\_stream\_recvmsg**，地址 **0xffffffff8044b472** 映射到功能 **arp\_rcv**：

```
[...]
ffffffff8024c5cd T unlock_new_inode
ffffffff8024c5da t unix_stream_sendmsg
ffffffff8024c920 t unix_stream_recvmsg
ffffffff8024cea1 t udp_v4_lookup_longway
[...]
ffffffff8044addc t arp_process
ffffffff8044b360 t arp_rcv
ffffffff8044b487 t parp_redo
ffffffff8044b48c t arp_solicit
[...]
```

## 第 40 章 使用 SYSTEMTAP 分析内核活动

您可以使用以下脚本，通过监控函数调用来分析内核活动。

### 40.1. 使用 SYSTEMTAP 的计数功能调用

您可以使用 `functioncallcount.stp` SystemTap 脚本来计算特定的内核功能调用。您还可以使用这个脚本来目标多个内核功能。

#### 先决条件

- 如[安装 Systemtap](#)所述，已安装了 SystemTap。

#### 流程

- 运行 `functioncallcount.stp` 脚本：

```
# stap --example functioncallcount.stp 'argument'
```

此脚本使用目标内核功能作为一个参数。您可以使用参数通配符将多个内核功能作为特定扩展的目标。

脚本的输出按字母顺序排列，包含调用的功能的名称，以及在抽样期间调用的次数。

考虑以下示例：

```
# stap -w -v --example functioncallcount.stp "*@mm*.c" -c /bin/true
```

其中：

- `-w`：压制警告。
- `-v`：使启动内核的输出可见。
- `-c` 命令：告知 SystemTap 在执行命令期间用于计数功能调用，在这个示例中是 `/bin/true`。输出应类似于以下内容：

```
[...]
__vma_link 97
__vma_link_file 66
__vma_link_list 97
__vma_link_rb 97
__xchg 103
add_page_to_active_list 102
add_page_to_inactive_list 19
add_to_page_cache 19
add_to_page_cache_lru 7
all_vm_events 6
alloc_pages_node 4630
alloc_slabmgmt 67
anon_vma_alloc 62
anon_vma_free 62
anon_vma_lock 66
anon_vma_prepare 98
```

```

anon_vma_unlink 97
anon_vma_unlock 66
arch_get_unmapped_area_topdown 94
arch_get_unmapped_exec_area 3
arch_unmap_area_topdown 97
atomic_add 2
atomic_add_negative 97
atomic_dec_and_test 5153
atomic_inc 470
atomic_inc_and_test 1
[...]

```

## 40.2. 使用 SYSTEMTAP 的追踪功能调用

您可以使用 `para-callgraph.stp` SystemTap 脚本来跟踪函数调用和函数返回。

### 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

### 流程

- 运行 `para-callgraph.stp` 脚本。

```
# stap --example para-callgraph.stp 'argument1' 'argument2'
```

脚本 `para-callgraph.stp` 有两个命令行参数：

- 您需要跟踪其 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。

考虑以下示例：

```
# stap -wv --example para-callgraph.stp 'kernel.function("@fs/proc.c")' 'kernel.function("vfs_read")' -c "cat /proc/sys/vm/* || true"
```

其中：

- `-w`：压制警告。
- `-v`：使启动内核的输出可见。
- `-c 命令`：告知 SystemTap 在执行命令期间用于计数功能调用，在这个示例中是 `/bin/true`。

输出应类似于以下内容：

```
[...]
267 gnome-terminal(2921): <-do_sync_read return=0xfffffffffffffff5
269 gnome-terminal(2921):<-vfs_read return=0xfffffffffffffff5
0 gnome-terminal(2921):->fput file=0xffff880111eebbc0
2 gnome-terminal(2921):<-fput
0 gnome-terminal(2921):->fget_light fd=0x3 fput_needed=0xffff88010544df54
```

```

3 gnome-terminal(2921):<-fget_light return=0xffff8801116ce980
0 gnome-terminal(2921):->vfs_read file=0xffff8801116ce980 buf=0xc86504 count=0x1000
pos=0xffff88010544df48
4 gnome-terminal(2921): ->rw_verify_area read_write=0x0 file=0xffff8801116ce980
ppos=0xffff88010544df48 count=0x1000
7 gnome-terminal(2921): <-rw_verify_area return=0x1000
12 gnome-terminal(2921): ->do_sync_read filp=0xffff8801116ce980 buf=0xc86504 len=0x1000
ppos=0xffff88010544df48
15 gnome-terminal(2921): <-do_sync_read return=0xffffffffffffffffffff5
18 gnome-terminal(2921):<-vfs_read return=0xffffffffffffffffffff5
0 gnome-terminal(2921):->fput file=0xffff8801116ce980

```

## 40.3. 使用 SYSTEMTAP 确定内核和用户空间花费的时间

您可以使用 `thread-times.stp` SystemTap 脚本来确定给定线程在内核或用户空间中的花费的时间长度。

### 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

### 流程

- 运行 `thread-times.stp` 脚本：

```
# stap --example thread-times.stp
```

这个脚本会在 5 秒期间显示 CPU 时间的前 20 个进程，以及样本中进行的 CPU 选项总数。这个脚本的输出还记录每个进程使用的 CPU 时间的百分比，以及时间是在内核空间或用户空间中消耗的。

```

tid %user %kernel (of 20002 ticks)
0 0.00% 87.88%
32169 5.24% 0.03%
9815 3.33% 0.36%
9859 0.95% 0.00%
3611 0.56% 0.12%
9861 0.62% 0.01%
11106 0.37% 0.02%
32167 0.08% 0.08%
3897 0.01% 0.08%
3800 0.03% 0.00%
2886 0.02% 0.00%
3243 0.00% 0.01%
3862 0.01% 0.00%
3782 0.00% 0.00%
21767 0.00% 0.00%
2522 0.00% 0.00%
3883 0.00% 0.00%
3775 0.00% 0.00%
3943 0.00% 0.00%
3873 0.00% 0.00%
```

## 40.4. 使用 SYSTEMTAP 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正在轮询。这样，您可以跟踪不必要的或过度的轮询，这有助于在 CPU 使用量和节能方面得到改进。

## 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

## 流程

- 运行 **timeout.stp** 脚本：

```
# stap --example timeout.stp
```

此脚本将跟踪每个应用程序随着时间的推移使用以下系统调用的次数：

- **poll**
- **select**
- **epoll**
- **itimer**
- **futex**
- **nanosleep**
- **signal**

在这个示例输出中，您可以看到使用哪个进程调用哪个进程以及次数。

uid	poll	select	epoll	itimer	futex	nanosle	signall	process
28937	148793	0	0	4727	37288	0	0	firefox
22945	0	56949	0	1	0	0	0	scim-bridge
0	0	0	36414	0	0	0	0	swapper
4275	23140	0	0	1	0	0	0	mixer_applet2
4191	0	14405	0	0	0	0	0	scim-launcher
22941	7908	1	0	62	0	0	0	gnome-terminal
4261	0	0	0	2	0	7622	0	escd
3695	0	0	0	0	0	7622	0	gdm-binary
3483	0	7206	0	0	0	0	0	dhcdbd
4189	6916	0	0	2	0	0	0	scim-panel-gtk
1863	5767	0	0	0	0	0	0	iscsid

## 40.5. 与 SYSTEMTAP 跟踪最常用的系统调用

您可以使用 **topsys.stp** SystemTap 脚本列出系统每 5 秒间隔使用的前 20 个系统调用。它还列出了该期间每个系统调用使用的次数。

## 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

## 流程

- 运行 `topsys.stp` 脚本：

```
# stap --example topsys.stp
```

考虑以下示例：

```
# stap -v --example topsys.stp
```

其中 `-v` 使启动内核的输出结果可见。

输出应类似于以下内容：

SYSCALL	COUNT
gettimeofday	1857
read	1821
ioctl	1568
poll	1033
close	638
open	503
select	455
write	391
writev	335
futex	303
recvmsg	251
socket	137
clock_gettime	124
rt_sigprocmask	121
sendto	120
setitimer	106
stat	90
time	81
sigreturn	72
fstat	66

## 40.6. 使用 SYSTEMTAP 跟踪每个进程的系统调用卷

您可以使用 `syscalls_by_proc.stp` SystemTap 脚本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。

### 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

### 流程

- 运行 `syscalls_by_proc.stp` 脚本：

```
# stap --example syscalls_by_proc.stp
```

`syscalls_by_proc.stp` 脚本的输出类似如下：

```
Collecting data... Type Ctrl-C to exit and display results
#SysCalls Process Name
1577 multiload-apple
692 synergyc
408 pcscd
376 mixer_applet2
299 gnome-terminal
293 Xorg
206 scim-panel-gtk
95 gnome-power-man
90 artsd
85 dhcddb
84 scim-bridge
78 gnome-screensav
66 scim-launcher
[...]
```

## 第 41 章 使用 SYSTEMTAP 监控磁盘和 I/O 活动

您可以使用以下脚本监控磁盘和 I/O 活动：

### 41.1. 使用 SYSTEMTAP 总结磁盘读/写流量

您可以使用 `desktop.stp` SystemTap 脚本来识别执行了哪些进程读取和写入系统。

#### 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

#### 流程

- 运行 `desktop.stp` 脚本：

```
# stap --example desktop.stp
```

脚本显示对磁盘读或写消耗最多的前十个进程。

输出包括每个列出进程的以下数据：

#### UID

用户 ID。用户 ID **0** 代表 root 用户。

#### PID

列出进程的 ID。

#### PPID

列出进程的父进程的进程 ID。

#### CMD

列出进程的名称。

#### DEVICE

列出进程从其中读取或写入的存储设备。

#### T

由列出的进程执行的操作类型，其中 **W** 代表写入，**R** 代表读取。

#### BYTES

从磁盘读取或写入的数据量。

`desktop.stp` 脚本的输出类似如下：

```
[...]
Mon Sep 29 03:38:28 2008 , Average: 19Kb/sec, Read: 7Kb, Write: 89Kb
UID PID PPID          CMD DEVICE T BYTES
0 26319 26294        firefox sda5 W 90229
0 2758 2757         pam_timestamp_c sda5 R 8064
0 2885 1            cupsd   sda5 W 1678
Mon Sep 29 03:38:38 2008 , Average: 1Kb/sec, Read: 7Kb, Write: 1Kb
UID PID PPID          CMD DEVICE T BYTES
0 2758 2757         pam_timestamp_c sda5 R 8064
0 2885 1            cupsd   sda5 W 1678
```

## 41.2. 使用 SYSTEMTAP 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。

### 先决条件

- 如[安装 Systemtap](#)所述，已安装了 SystemTap。

### 流程

- 运行 `iotime.stp` 脚本：

```
# stap --example iotime.stp
```

脚本会在每次系统调用打开、关闭、读取以及写入文件时进行跟踪。对于每一个系统调用访问的文件，它会计算出任何读取或写入到完成并跟踪数据量（以字节为单位）的微秒数，以字节为单位，读取或写入文件。

输出包含：

- 微秒中的时间戳
- 进程 ID 和进程名称
- access** 或 **iotime** 标志
- 已访问的文件

如果某个进程能够读取或写入任何数据，一对 **access** 和 **iotime** 行应同时出现。**access** 行指的是给定进程开始访问文件的时间。访问行的末尾将显示读取或写入的数据量。**iotime** 行显示进程执行读和写操作所使用的时间，以微秒为单位。

**iotime.stp** 脚本的输出类似如下：

```
[...]
825946 3364 (NetworkManager) access /sys/class/net/eth0/carrier read: 8190 write: 0
825955 3364 (NetworkManager) iotime /sys/class/net/eth0/carrier time: 9
[...]
117061 2460 (pcscd) access /dev/bus/usb/003/001 read: 43 write: 0
117065 2460 (pcscd) iotime /dev/bus/usb/003/001 time: 7
[...]
3973737 2886 (sendmail) access /proc/loadavg read: 4096 write: 0
3973744 2886 (sendmail) iotime /proc/loadavg time: 11
[...]
```

## 41.3. 使用 SYSTEMTAP 跟踪累积的 I/O 信息

您可以使用 `traceio.stp` SystemTap 脚本来跟踪系统的累积数量。

### 先决条件

- 如[安装 Systemtap](#)所述，已安装了 SystemTap。

## 流程

- 运行 `traceio.stp` 脚本：

```
# stap --example traceio.stp
```

该脚本打印一段时间内生成 I/O 流量的前十个可执行文件。它还会跟踪由那些可执行文件执行的 I/O 的读取和写入数量。该信息会被跟踪，并以 1 秒的间隔进行打印，以降序排列。

`traceio.stp` 脚本的输出类似如下：

```
[...]
Xorg r: 583401 KiB w: 0 KiB
floaters r: 96 KiB w: 7130 KiB
multiload-apple r: 538 KiB w: 537 KiB
sshd r: 71 KiB w: 72 KiB
pam_timestamp_c r: 138 KiB w: 0 KiB
staprund r: 51 KiB w: 51 KiB
snmpd r: 46 KiB w: 0 KiB
pcscd r: 28 KiB w: 0 KiB
irqbalance r: 27 KiB w: 4 KiB
cupsd r: 4 KiB w: 18 KiB
Xorg r: 588140 KiB w: 0 KiB
floaters r: 97 KiB w: 7143 KiB
multiload-apple r: 543 KiB w: 542 KiB
sshd r: 72 KiB w: 72 KiB
pam_timestamp_c r: 138 KiB w: 0 KiB
staprund r: 51 KiB w: 51 KiB
snmpd r: 46 KiB w: 0 KiB
pcscd r: 28 KiB w: 0 KiB
irqbalance r: 27 KiB w: 4 KiB
cupsd r: 4 KiB w: 18 KiB
```

## 41.4. 在使用 SYSTEMTAP 的特定设备上监控 I/O 活动

您可以使用 `traceio2.stp` SystemTap 脚本来监控特定设备的 I/O 活动。

### 先决条件

- 如[安装 Systemtap](#) 所述，已安装了 SystemTap。

## 流程

- 运行 `traceio2.stp` 脚本。

```
# stap --example traceio2.stp 'argument'
```

该脚本使用整个设备号作为参数。要找到这个数字，您可以使用：

```
# stat -c "0x%D" directory
```

其中 `directory` 位于要监控的设备中。

输出包含以下内容：

- 执行读取或写入进程的任何进程的名称和 ID
- 它正在执行的功能 (**vfs\_read** 或 **vfs\_write**)
- 内核设备号

请考虑 # **stap traceio2.stp 0x805** 的输出

```
[...]
synergyc(3722) vfs_read 0x800005
synergyc(3722) vfs_read 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
[...]
```

## 41.5. 监控使用 SYSTEMTAP 文件的读取和写入

您可以使用 **inodewatch.stp** SystemTap 脚本来实时从文件中读取和写入。

### 先决条件

- 如[安装 Systemtap](#)所述，已安装了 SystemTap。

### 流程

- 运行 **inodewatch.stp** 脚本。

```
# stap --example inodewatch.stp 'argument1' 'argument2' 'argument3'
```

脚本 **inodewatch.stp** 使用三个命令行参数：

1. 文件的主设备号。
2. 文件的副设备号。
3. 文件的 inode 号。

您可以使用以下方法获取这些信息：

```
# stat -c '%D %i' filename
```

其中 *filename* 是绝对路径。

考虑以下示例：

```
# stat -c '%D %i' /etc/crontab
```

输出应类似于如下：

```
805 1078319
```

其中：

- **805** 是基于16(十六进制)的设备号。最后两个数字是副设备号码，剩余的数字是主号码。
- **1078319** 是 inode 号。

要启动监控 **/etc/crontab**，请运行：

```
# stap inodewatch.stp 0x8 0x05 1078319
```

在前两个参数中，您必须为具有 16 的编号使用 0x 前缀。

输出包含以下内容：

- 执行读取或写入进程的任何进程的名称和 ID
- 它正在执行的功能 (**vfs\_read** 或 **vfs\_write**)
- 内核设备号

这个示例的输出应该类似如下：

```
cat(16437) vfs_read 0x800005/1078319
cat(16437) vfs_read 0x800005/1078319
```

# 第 42 章 使用 BPF COMPILER COLLECTION 分析系统性能

BPF Compiler Collection (BCC)通过组合 Berkeley Packet Filter (BPF)的功能来分析系统性能。使用 BPF，您可以安全地运行内核中的自定义程序，以访问系统事件和数据，以性能监控、追踪和调试。BCC 简化了 BPF 程序的开发和部署，用户可使用工具和库从其系统中提取重要见解。

## 42.1. 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

- 安装 **bcc-tools**。

```
# yum install bcc-tools
```

BCC 工具安装在 **/usr/share/bcc/tools/** 目录中。

### 验证

- 检查安装的工具：

```
# ls -l /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
...
```

列表中的 **doc** 目录提供了每个工具的文档。

## 42.2. 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在每个事件基础上高效、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

- [安装的 bcc-tools 软件包](#)
- 根权限

### 流程

#### 使用 **execsnoop** 检查系统进程

- 在一个终端中运行 **execsnoop** 程序：

```
# /usr/share/bcc/tools/execsnoop
```

- 要创建一个短暂的 **ls** 命令的进程，请在另一个终端中输入：

```
$ ls /usr/share/bcc/tools/doc/
```

- 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** 程序为消耗系统资源的每个新进程打印一行输出。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** 跟踪命令打开了哪些文件

- 在一个终端中，运行 **opensnoop** 程序，以打印仅由 **uname** 命令进程打开的文件的输出：

```
# /usr/share/bcc/tools/opensnoop -n uname
```

- 在另一个终端中，输入命令来打开某些文件：

```
$ uname
```

- 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 **biotop** 监控在磁盘上执行 I/O 操作的排在前面的进程

1. 在一个终端中使用参数 **30** 运行 **biotop** 程序来生成 30 秒概述：

```
# /usr/share/bcc/tools/biotop 30
```



#### 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。

1. 在另一个终端中，输入命令从本地硬盘设备读取内容，并将结果写入 **/dev/zero** 文件：

```
# dd if=/dev/vda of=/dev/zero
```

此步骤会生成特定的 I/O 流量来演示 **biotop**。

2. 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 dbus         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)

有关 **biotop** 的详情、示例和选项，请查看 [/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** 来公开意外慢的文件系统操作

**xfsslower** 测量 XFS 文件系统执行读、写、打开或同步(**fsync**)操作所花费的时间。**1** 参数可确保程序仅显示比 1ms 较慢的操作。

1. 在一个终端中运行 **xfsslower** 程序：

```
# /usr/share/bcc/tools/xfsslower 1
```



#### 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，输入命令在 **vim** 编辑器中创建一个文本文件，以开始与 XFS 文件系统进行交互：

```
$ vim text
```

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'
...							

每行代表文件系统中的一个操作，它花费的时间超过特定阈值。**xfsslower** 检测可能的文件系统问题，其表现为意外的慢操作。

**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。(O)

#### 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。