红帽企业版 Linux 6

性能调节指南

在红帽企业版 Linux 6 中优化子系统流量

版 4.0

Last Updated: 2017-10-12

红帽企业版 Linux 6 性能调节指南

在红帽企业版 Linux 6 中优化子系统流量
版 4.0

红帽 主题专家

编辑

Don Domingo

Laura Bailey

法律通告

Copyright © 2011 Red Hat, Inc. and others.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

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, 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 Software Collections 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.

摘要

《性能调节指南》 论述了如何优化运行红帽企业版 Linux 6 的系统性能。它还记录了与性能有关的红帽企业版 Linux 6 升级。 虽然本指南包含实地测试和验证的步骤，红帽建议您在将其部署到产品环境前在测试环境中全面测试所有计划的配置。您还应该在调整配置前备份所有数据。

目录

第 1 章 概述

*《性能调节指南》*是红帽企业版 Linux 配置的完整参考。虽然本发行本也记录了红帽企业版 Linux 5 的性能，但所有在此提供的步骤都只用于红帽企业版 Linux 6。

本书将分几个章节论述红帽企业版 Linux 中的具体子系统。*《性能调节指南》*着重论述以下三个主要方面：

性能

各个子系统章节论述红帽企业版 Linux 6 中特有的性能。这些章节还论述了红帽企业版 Linux 6 中具体子系统性能比红帽企业版 Linux 5 的显著改善。

分析

本书还模拟每个具体子系统的性能指标。根据具体服务给出这些指标的常用值，帮助您理解其在真实产品系统中的意义。

另外，*《性能调节指南》*还演示了查询子系统性能数据（即侧写）的不同方法。注：这里演示的一些侧写工具在其他文档中有具体论述。

配置

可能本文档中大多数重要信息是告诉您如何在红帽企业版 Linux 6 中调整性能。*《性能调节指南》*解释了如何为具体服务微调红帽企业版 Linux 6 子系统。

请记住，微调子系统的性能可能会影响其他子系统性能，有时可能是负面影响。红帽企业版 Linux 6 的默认配置是为*大多数*在*中度*负载下运行的服务优化。

*《性能调节指南》*中模拟的步骤都由红帽工程师在实验室和现场进行全面测试。但红帽建议您在将您其应用到产品服务器前在安全的测试环境中正确测试所有计划的配置。您还可以在开始调节系统前备份所有数据和配置信息。

1.1. 读者

本文档适合两类读者：

系统/业务分析师

本书模拟并解释了红帽企业版 Linux 6 中包含的性能，提供大量子系统如何为具体工作负载工作的信息（包括默认和优化配置）。用来描述红帽企业版 Linux 6 性能的详细长度可帮助签字客户和销售工程师了解这个平台在可以接受的水平提供消耗资源服务的适合程度。

*《性能调节指南》*还在可能的情况下提供各个特性具体文档的链接。读者可根据这些详细的性能特点构成部署和优化红帽企业版 Linux 6 的高级策略。这样可让读者开发*并*评估架构提案。

这个注重性能的文档适合非常了解 Linux 子系统和企业级网络的读者。

系统管理员

本文档中模拟的步骤审适合有 RHCE[1]技能（或者有相当于 3-5 年部署和管理 Linux 经验）的系统管理员。*《性能调节指南》*主要为每个配置提供尽可能详细的效果，就是说论述所有可能的性能平衡。

性能调节的基本技能不是了解如何分析和调节子系统。系统管理员应该了解如何*为具体的目的*平衡和优化红帽企业版 Linux 6 系统。即*意味着*了解在尝试使用提高具体子系统性能设计的配置时会有什么性能交换和代偿。

1.2. 横向可扩展性

红帽对红帽企业版 Linux 性能的提高的重点是*可扩展性*。主要根据其如何影响负载谱（即独立网页服务器到服务器大型机）中各个区段平台性能来评估那些提高性能的功能。

关注可延展性可让红帽企业版 Linux 保持其在不同类型的负载和目的通用性。同时这也意味着随着您的业务增长和负载增加，重新配置您的服务器环境不会产生太高的费用（就人员小时工资而言），且更直观。

红帽在红帽企业版 Linux 中同时进行了*横向可扩展性*和*纵向可扩展性*改进，但横向可扩展性的应用更为普遍。横向可扩展性的改进主要是使用多台*标准计算机*分布大工作负载以便改进性能和可靠性。

在典型服务器组中，这些标准计算机以 1U 机架安装服务器和刀片服务器的形式存在。每台标准计算机都可小至简单的双插槽系统，尽管有些服务器组使用有更多插槽的大型系统。有些企业级网络是混合使用大型系统和小系统。在此类情况下，大型系统使用高性能服务器（例如：数据库服务器），同时小型系统是专用的应用程序服务器（例如：网页或者邮件服务器）。

这类可扩展性简化了您 IT 基础设施的增长：中级业务以及适当的负载只需要两个批萨盒服务器就可满足其需要。随着公司雇佣更多的员工，扩展其业务，增加其销售等等，其 IT 要求在量和复杂性方面都会增张。横向可扩展性可让 IT 部门只需部署附加机器即可，且（大多数）可使用同样的配置。

总之，横向可扩展性添加了一个提取层，简化了系统硬件管理。通过开发红帽企业版 Linux 平台使其横向增大，增强 IT 服务的功能和性能，方法就是简单地添加新的容易配置的机器即可。

1.2.1. 并行计算

从红帽企业版 Linux 横向可扩展性中获益的用户不仅是因为可以简化系统硬件管理，还因为横向可扩展性是符合硬件开发当前趋势的开发理念。

想象一下：最复杂的企业版程序同时要执行数千个任务，每个任务之间都有不同协调方法。虽然早期的计算机使用单核处理器可以完成这些任务，但当今所有可用的虚拟处理器都有多个核。现代计算机有效地将多个核放到单一插槽中，让单插槽桌面系统或者笔记本电脑也有多处理器系统。

从 2010 年开始，标准 Intel 和 AMD 处理器都有六核产品。此类处理器批萨盒或者刀片服务器中最为常见，这样的服务器现在可以有多达 40 个核。这些低成本高性能系统可为大型机提供超大系统容量和性能。

要获得最佳性能及系统使用，则必须让每个核都保持忙碌。就是说必须运行 32 个独立任务以便充分利用 32 核刀片服务器。如果一个刀片组包含十组这样的 32 核刀片服务器，那么整个配置最少可同时处理 320 个任务。如果这些任务都属于同一项任务，则必须对之进行协调。

红帽企业版 Llinux 的开发已经可以很好地适应硬件开发趋势，并确保商家可从中获取最大利益。第 1.3 节 “分布式系统” 中探讨了启用红帽企业版 Linux 横向可扩展性的技术细节。

1.3. 分布式系统

为完全利用横向延展性，红帽企业版 Linux 使用*分布式计算*的很多组件。可将组成分布式计算的技术分为三层：

通讯

横向延展需要同时（平行）执行很多任务。因此这些任务必须有*进程间通讯*以便协调其工作。另外，采用横向延展的平台应该可以跨多个系统共享任务。

存储

本地磁盘存储不足以满足横向延展的要求。需要分布式或者共享存储，一个有可允许单一存储卷容量渐变增长的提取层，另外还有额外的新存储硬件。

管理

分布式计算最重要的任务是*管理*层。这个管理层可协调所有软件和硬件组件，有效管理通讯、存储以及共享资源的使用。

以下小节论述了每一层的详细技术。

1.3.1. 通讯

通讯层可保证数据传输，它由两部分组成：

- 硬件
- 软件

多系统进行通讯最简单（也最迅速）的方法是*共享内存*。这样可以推导出类似内存读取/写入操作的用量。共享内存的带宽高，低延迟，且常规内存读取/写入操作成本低。

以太网

计算机之间最常用的通讯是使用以太网。目前系统默认提供 *Gigabit Ethernet*（GbE），且大多数服务器包括 2-4 个 Gigabit 以太网 GbE 端口。GbE 提供良好的带宽和延迟性能。这是目前使用的大多数分布式系统的基础。即使系统使用较快的网络硬件，一般也是使用 GbE 专门用于管理接口。

10GbE

Ten Gigabit Ethernet (10GbE) 是在高端甚至一些中端服务器中迅速得以广泛使用。10GbE 提供比 GbE 快 10 倍的带宽。其主要优点之一是使用现代多核处理器，它可保证通讯和计算之间的平衡。您可以将使用 GbE 的单核系统与使用 10GbE 的八核系统进行比较。以这种方法使用，10GbE 对保持整体系统性能有特殊意义，并可以避免通讯瓶颈。

遗憾的是，10GbE 很贵。虽然 10GbE NIC 的成本已经下降，但互联（特别是光纤）的价格仍然很高，且 10GbE 网络交换机的价格极为昂贵。我们可以期待在一定时间内价格可以下降，但 10GbE 现在是在服务器机房主干以及对性能至关重要的程序中使用最多的网卡。

Infiniband

Infiniband 提供比 10GbE 更高的性能。除在以太网中使用 TCP/IP 和 UDP 网络连接外，Infiniband 还支持共享内存通讯。这就允许 Infiniband 通过*远程直接内存访问* (RDMA) 在系统间使用。

使用 RDMA 可让 Infiniband 直接从系统中删除数据而无需负担 TCP/IP 或者插槽连接，继而减小延迟，这对有些程序是很重要的。

Infiniband 最常用于*高性能技术计算*（HPTC）程序，此类程序要求使用高带宽、低延迟和低负担。很多超级计算程序都得益于此，提高性能的最佳方式是使用 Infiniband，而不是快速处理器或者更多内存。

RoCCE

使用以太网的 RDMA (RoCCE) 通过 10GbE 基础设施实施 Infiniband 形式的通讯（包括 RDMA）。鉴于 10GbE 产品产量的增长带来的成本改善，我们有理由相信可能会在更大范围的系统和程序中使用更多的 RDMA 和 RoCCE。

红帽公司在红帽企业版 Linux 6 中全面支持这些通讯方法。

1.3.2. 存储

使用分布式计算的环境使用共享存储的多个实例。这可能代表以下两个含义之一：

- 多系统在单一位置保存数据
- 存储单元（例如卷）由多个存储应用组成

最熟悉的存储示例是挂载到系统中的本地磁盘驱动器。这对将所有程序都托管在一台主机中的 IT 操作很合适。但由于基础设施可能包括数十个乃至数百个系统，管理如此多的本地存储磁盘将变得困难且复杂。

分布式存储添加了一层以便为业务规模减轻并实现自动存储硬件管理。多个系统共享少量存储实例可减少管理员需要进行管理的设备数量。

将多个存储设备的存储容量强化到一个卷中对用户和管理员都有好处。此类分布式管理提供了存储池的提取层：用户看到的是单一存储单元，管理员可通过添加更多硬件很方便地增大该单元。有些启用分布式存储的技术也提供附加利益，比如故障切换以及多路径。

NFS

网络文件系统 (NFS) 可让多服务器或者用户通过 TCP 或者 UDP 挂载并使用远程存储的同一实例。NFS 一般用来保存由多个程序共享的数据。它还便于对大量数据的海量存储。

SAN

存储区网络 (SANs) 使用光纤或者 iSCSI 协议提供对存储的远程访问。光纤基础设施（比如光纤主机总线适配器、开关以及存储阵列）有高性能、高带宽和海量存储。SAN 根据处理分割存储，为系统升级提供可观的灵活性。

SAN 的其他优点还有它们可为执行主要存储硬件管理任务提供管理环境。这些任务包括：

- 控制对存储的访问
- 管理海量数据
- 供应系统
- 备份和复制数据
- 提取快照
- 支持系统故障切换
- 保证数据完整性
- 迁移数据

GFS2

红帽*全局文件系统 2*（GFS2）提供一些特别定制的功能。GFS2 的基本功能是提供单一文件系统，其中包括同时读/写访问，集群中跨多个成员的共享。即使说该集群的每个成员都可以看到 GFS2 文件系统中“磁盘上”的完全相同的数据。

GFS2 可让所有系统同时访问该“磁盘”。为维护数据完整性，GFS2 使用*分布式锁管理器*（DLM），它在具体位置一次只允许一个系统进行写入。

GFS2 对故障切换程序最合适，因为那些程序要求高存储容积。

有关 GFS2 的详情请参考*《全局文件系统 2》*。有关存储的常规信息请参考*《存储管理指南》*。您可以在 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ 找到这两本手册。

1.3.3. 聚合网络

通过网络通讯一般使用以太网进行，使用专用光纤 SAN 环境的存储流量。通常有专用网络或者串行链路进行系统管理，且甚至可能使用*心跳*管理[2]。结果是在多网络中通常使用单一服务器。

在每台服务器中提供多个连接费用高昂，且累赘，不容易管理。这样就增加了将所有连接整合到一台服务器中的需求。*使用以太网的光纤* (FCoE) 和 *Internet SCSI* (iSCSI) 可以满足这个需要。

FCoE

使用 FCoE，标准光纤命令和数据包可使用 10GbE 物理基础架构通过单一*聚合网卡*（CNA）传送。也可使用同一链接传输标准 TCP/IP 以太网流量以及光纤存储操作。FCoE 为多个逻辑网络/存储连接使用一个物理网卡（和一条线缆）。

FCoE 有以下优点：

连接数减少

FCoE 将每台服务器的网络连接减少了一半。您仍可以根据性能或者可用性目的选择使用多个连接，虽然单一连接就可以提供存储和网络连接。这对批萨盒服务器和刀片机都非常有用，因为它们的组件空间都有限。

低成本

减少的连接数立刻表现在线缆、开关以及其他联网设备的减少。以太网的历史也是规模经济的历史。网络成本急剧下降，因为市场上设备数量已从百万级上升到十亿级，明显的表现就是 100Mb 以太网和 GB 以太网设备价格的下降。

同样，10GbE 也变得比以前便宜很多。另外，因为已可将 CNA 硬件整合到单一芯片中，广泛的使用也将增加其在市场中的数量，直接造成价格大幅下降。

iSCSI

Internet SCSI (iSCSI) 是另一个聚合网络协议类型，它是 FCoE 的替换产品。与光纤相似，iSCSI 在网络中提供块级存储。但 iSCSI 不提供完整管理环境。iSCSI 比 FCoE 强的一点是提供更多光纤容量和灵活性，同时成本较低。

---

[1] 红帽认证工程师，详情请参考 http://www.redhat.com/training/certifications/rhce/。

[2] *心跳*管理是在系统间交换信息以保证各个系统都正常工作。如果系统“失去心跳”，则假设其失败或者已关闭，并使用另一个系统替换。

第 2 章 红帽企业版 LINUX 6 性能特点

2.1. 64 位支持

红帽企业版 Linux 6.4 支持 64 位处理器：这些处理器理论上可使用多达 16 *EB* 内存。从 GA 版本开始，红帽企业版 Linux 就已通过测试并证明可以支持多达 8TB 物理内存。

在更新几个次要版本后，红帽企业版 Linux 6 支持的内存大小有望继续增加，因为红帽总是不断引进并改进可使用更大内存块的功能。此类改进（从红帽企业版 Linux 6 GA 版本开始）有：

- 大页面和透明大页面
- 非均匀内存访问改进

下面的小节中将更具体地列举这些改进。

大页面和透明大页面

在红帽企业版 Linux 6 中采用*大页面*可让系统在跨不同内存负载工作时更有效地管理内存。大页面动态地使用 2MB 页面，而不是标准的 4KB 页面大小，这样可以让应用程序全面处理 GB 甚至 TB 内存。

很难手动创建、管理和使用大页面。要解决这个问题，红帽企业版 Linux 6 还采用*透明大页面*（THP）。THP 自动管理很多使用大页面时的复杂情况。

有关大页面以及 THP 的详情请参考 第 5.2 节 “大页面和透明大页面”。

NUMA 改进

很多新系统现在支持*非均匀内存访问*（NUMA）。NUMA 为大型系统简化硬件设计和创建，但它也增加了应用程序开放的复杂性。例如：NUMA 采用本地和远程内存，但访问远程内存的时间是访问本地内存的好几倍。这个功能（还有其他一些功能）有很多隐患，可能会影响操作系统、应用程序以及应部署的系统配置。

红帽企业版 Linux 6 为更好地使用 NUMA 进行了优化，这要感谢一些可以在 NUMA 系统中帮助管理用户和程序的功能。这些功能包括 CPU 亲和力、CPU pinning（芯片组）、numactl 和控制组，这些功能可将进程（亲和力）或者程序（pinning）“捆绑”到具体的 CPU 或者一组 CPU 中。

有关红帽企业版 Linux 6 中 NUMA 支持的详情请参考 第 4.1.1 节 “CPU 和 NUMA 拓扑”。

2.2. TICKET 自旋锁

系统设计的关键是要保证进程不会更改另一个进程使用的内存。无法控制的内存更改可导致数据污染和系统崩溃。要防止此类情况出现，操作系统要允许进程锁定一个内存片段，执行操作，然后解锁或者“释放”内存。

内存锁定的常见使用是通过*自旋锁*实现的，它可让进程一直检查是否有可用的锁，并在所可用时立即使用。如果有多个进程竞争同一锁，那么在该锁被释放后第一个请求该锁定进程会得到它。当所有进程有对内存相同的访问时，这个方法是“公平”的且运作良好。

遗憾的是，在 NUMA 系统中，不是所有进程都对所有对等的访问。与该锁处于同一 NUMA 节点中的进程明显可以优先获得该锁。远程 NUMA 节点中的进程可能会有锁不足和性能下降的问题。

为解决这个问题，红帽企业版 Linux 采用 *ticket 自旋锁*。这个功能为锁添加了预留队列机制，可让*所有*进程根据其提出请求的时间顺序使用锁。这样可以消除计时问题以及不公平的锁请求利益。

虽然 ticket 自旋锁负担比普通自旋锁要高，但它缩放比例更大，并可在 NUMA 系统中提供更好的性能。

2.3. 动态列表结构

该操作系统需要系统中每个处理器的一组信息。在红帽企业版 Linux 5 中是将这组信息分配到内存的固定大小阵列中。通过索引到这个阵列可获得每个独立处理器中的信息。这个方法迅速、简便，且对包含少数处理器的系统来说相对直接。

但随着系统中处理器数量的增长，这个方法会产生相当大的费用。因为内存中固定大小的阵列是一个单一共享资源，它可能成为一个瓶颈，因为更多的处理器会尝试同时访问它。

为解决这个问题，红帽企业版 Linux 6 使用*动态列表结构*提供处理器信息。这样可以动态分配用于处理器信息的阵列：如果系统中只有八个处理器，则会在该列表中生成八个条目。如果有 2048 个处理器，则会生成 2048 个条目。

使用动态列表结构可进行更细微的锁定。例如：如果要同时为处理器 6、72、183、657、931 和 1546 进行更新，则可以同时进行更新。显然在高性能系统中类似这种情况出现的频率比小系统要高得多。

2.4. 无空循环内核

在红帽企业版 Linux 以前的版本中，内核使用基于计时器的机制，可连续生成系统中断。在每个中断中，系统进行*轮循*，即它检查是否有需要进行的工作。

根据设置，这个系统中断或者*对时信号*可能会每秒出现几百或者几千次。无论系统负载如何，这种情况每秒都会发生。在轻度负载的系统中，这会因为防止处理器有效使用睡眠状态而影响*电源消耗*。处于睡眠状态时系统使用的电量最少。

系统最有效地使用电源的方法是尽快完成工作，尽可能进入深度睡眠状态，而且能睡多久睡多久。红帽企业版 Linux 使用*对时信号*实现这个目的。使用这个方法，将中断计时器从 idle 池中移除，将红帽企业版 Linux 6 转化为完全由中断驱动的环境。

无间断内核可让系统在闲置时进入深度睡眠状态，并在有工作要做时迅速反应。

有关详情请参考*《电源管理指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

2.5. 控制组

红帽企业版 Linux 6 为性能调节提供很多有用的选项。大型系统，比如几百个处理器，可调节为提供超级性能。但调节这些系统需要高级专家以及明确的工作量。当大型系统非常昂贵且数量不多，应该予以特殊对待。现在这些系统是大型机，我们需要更多有效工具。

目前使用的用来强化服务的更强大的系统可让事情并的更为复杂。以前在 4-8 台就服务器中运行的负载现在都在一台服务器在中运行。如 第 1.2.1 节 “并行计算” 所述，现在很多中级系统所含核比以前高性能机器还要多。

很多现代程序已设计为使用多线程或者进程进行平行处理以提高性能。但只有少数程序能充分利用八个以上线程。因此多程序通常要在 32-CPU 中安装以便让产能最大化。

想象这样一种情况：小型、廉价大型机系统现在与原来昂贵的高性能机器有同样的性能。廉价的高性能机器可让系统架构师可以在较少的机器在很强化更多服务。

但有些资源（比如 I/O 以及网络通讯）是共享的，不会如 CPU 计数一般快速增长。因此当某个系统使用单一资源时间过长时，托管多个程序的系统可能会有总体性能下降。

为解决这个问题，红帽企业版 Linux 6 现在支持*控制组*（cgroups）。Cgroups 可让管理员根据需要为具体任务分配资源。这意味着例如要为数据库应用程序分配四台 CPU 的80%，60GB 内存，以及 40% 的磁盘 I/O。在同一系统中运行的网页应用程序则使用两个 CPU，2GB 内存以及 50% 的可用网络带宽。

结果是数据库和网页应用程序都有良好的性能，因为该系统防止二者过度占用系统资源。另外，cgroups 的很多方面是*自我调整*，可让系统根据负载的变化做相应的调整。

cgroup 有两个主要部分：

- 分配给该 cgroup 的任务列表
- 分配给那些任务的资源

分配给 cgroup 的任务是在该 cgroup *内部*运行。所有其派生处的子任务也都将在该 cgroup 中运行。这样可以让管理员将整个程序作为一个单位进行管理。管理员还可以分配以下资源：

- CPU 组
- 内存
- I/O
- 网络（带宽）

在 CPU 组中，cgroups 可让管理员配置 CPU 数，具体 CPU 或者节点的亲和性 [3]，以及任务组使用的 CPU 时间长度。使用 cgroups 配置 CPU 在是保证总体性能良好，防止程序过度消耗资源，影响其他任务的关键，同时还可保证程序不会缺少 CPU 时间。

I/O 带宽和网络带宽由其他资源控制器管理。同样，资源控制器可让您决定 cgroup 中任务可消耗的带宽，并保证 cgroup 中的任务既不会消耗过多资源，也不会缺少资源。

Cgroups 可让管理员在较高等级确定并分配各种程序需要（和将要）消耗的系统资源。然后系统会自动管理并平衡各种程序，提供良好的可预估性能，并优化系统总体性能。

有关使用控制组的详情请参考*《资源管理指南》*，网址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

2.6. 存储和文件系统改进

红帽企业版 Linux 6 还有一些存储和文件系统管理改进。这个版本中两个最著名的优势是 ext4 和 XFS 支持。有关与存储和文件系统有关的性能改进详情请参考 第 7 章 *文件系统*。

Ext4

Ext4 是红帽企业版 Linux 6 的默认文件系统。它是 EXT 文件系统产品线的第四代产品，理论上支持的最大文件系统大小为 1ET，单一文件最大可为 16TB。红帽企业版 Linux 支持最大的文件系统大小为 16TB，单一文件最大可为 16TB。除有较大存储容量外，ext4 还包括几个新功能，比如：

- 基于扩展的元数据
- 延迟的分配
- 日志检查总数

有关 ext4 文件系统详情请参考 第 7.3.1 节 “Ext4 文件系统”。

XFS

XFS 是一个鲁棒且成熟的 64 位日志文件系统，支持超大文件和单一主机中的文件系统。这个文件系统最初由 SGI 开发，并有在超大服务器和存储阵列中运行的经验。XFS 功能包括：

- 延迟的分配

- 动态分配的内节点
- 剩余空间管理可扩展性的 B-tree 索引
- 在线碎片重组和文件系统增长
- 复杂的元数据预读算法

虽然 XFS 可以扩展到 ET 大小，但红帽支持的最大 XFS 无极限系统大小为 100TB。有关 XFS 的详情请参考 第 7.3.2 节 “XFS 文件系统”。

超大引导驱动器

传统 BIOS 支持的最大磁盘大小为 2.2TB。使用 BIOS 的红帽企业版 Linux 6 系统可以通过使用名为*全局分区表*（GPT）的新磁盘结构支持超过 2.2TB 的磁盘。GPT 只能用于数据磁盘，不能在使用 BIOS 的引导驱动器中使用，因此引导驱动器最大只能是 2.2TB。BIOS 最初是由 IBM PC 发明的，虽然 BIOS 已经有了长足的发展可以使用先进的硬件，但*统一可扩展固件界面*（UEFI）的设计是用来支持最新以及新兴的硬件。

红帽企业版 Linux 6 还支持 UEFI，它可用来替换 BIOS（仍支持）。使用 UEFI 运行红帽企业版 Linux 6 的系统允许在 boot 分区和数据分区中使用 GPT 和 2.2TB（和更大的）分区。

重要

红帽企业版 Linux 6 不支持 32 位 x86 系统的 UEFI

重要

请注意：UEFI 和 BIOS 的引导配置有很大差别。因此安装的系统必须使用安装时所用的同一固件引导。您不能在使用 BIOS 的系统中安装操作系统后，再在使用 UEFI 的系统中引导这个安装。

红帽企业版 Linux 6 支持 UEFI 规格的 2.2 版。支持 UEFI 规格版本 2.3 或者之后版本的硬件应使用红帽企业版 Linux 6 引导，但这些后面的版本中拥有的附加功能可能无法使用。UEFI 规格请参考 http://www.uefi.org/specs/agreement/。

[3] 节点通常是指插槽中一组 CPU 或者 core。

第 3 章 监控和分析系统性能

本章简要介绍了可用来监控和分析系统及程序性能的工具，并指出每个工具最合适的工具。每个工具收集的数据可找到不能达到系统最佳性能的瓶颈或者其他系统问题。

3.1. PROC 文件系统

proc"文件系统"是包含代表 Linux 内核当前状态的文件层级目录。它可让程序和用户从内核角度查看系统。

proc 目录还包含有关系统硬件以及目前正在运行的进程的信息。大多数此类文件为只读，但有些文件（主要是在 **/proc/sys** 中）是可以由用户和程序控制让内核了解配置更改。

有关查看和编辑 **proc** 目录的详情请参考 *《部署指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

3.2. GNOME 和 KDE 系统监视器

GNOME 和 KDE 桌面环境都有图形工具可以帮助您监控和调整系统行为。

GNOME 系统监视器

GNOME System Monitor 显示基本系统信息并可让您监控系统进程以及资源或者文件系统使用。在 **Terminal** 中使用 **gnome-system-monitor** 命令打开它，也可已在「**应用程序**」菜单中选择「**系统工具**」 > 「**系统监视器**」打开该程序。

GNOME System Monitor 有四个标签：

「**系统**」
显示计算机硬件和软件基本信息。

「**进程**」
显示活跃进程以及那些进程之间的关系，同时还显示每个进程的详细信息。它还让您可以过滤显示的进程并在那些进程中执行某些动作（比如启动、停止、杀死、更改优先权等等。）。

「**资源**」
显示当前 CPU 使用时间、内存以及 swap 空间用量和网络使用。

「**文件系统**」
列出所有已挂载文件系统及其每个系统的脚本信息，比如文件系统类型、挂载点以及内存使用。

有关 **GNOME System Monitor** 详情请参考程序中的「**帮助**」菜单，或者 *《部署指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

KDE 系统卫士

KDE System Guard 可让您监控当前系统负载和正在运行的进程。它还让您在这些进程中执行动作。在 **Terminal** 中使用 **ksysguard** 命令打开它，也可点击「**Kickoff Application Launcher**」，选择「**应用程序**」 > 「**系统工具**」 > 「**系统监视器**」打开该程序。

KDE System Guard 有两个标签：

「**进程表**」

显示所有运行中的进程，默认是根据字母顺序显示。您也可以根据其他属性值对进程排序，比如总 CPU 用量、物理或者共享内存用量、拥有者和优先权。您还可以过滤可视结果，搜索具体进程或者在某个进程中执行某些动作。

「系统负载」

显示 CPU 用量、内存和 swap 空间用量以及网络使用的历史记录图。鼠标放在图形上就可以看到详细分析和图形按钮。

有关 **KDE System Guard** 详情请参考程序中的「**帮助**」菜单。

3.3. 内嵌命令行监控工具

除图形监控工具外，红帽企业版 Linux 还提供几个可用来监控系统的命令行工具。这些工具的有点就是他们可在第五级以外使用。本小节对每个工具进行简要论述，并提供每个工具最适合用途的建议。

`top`

top 工具为运行中的系统提供一个动态实时的进程查看。它可以显示各种信息，其中包括系统概述以及目前由 Linux 内核管理的任务。它还有一定可以操控进程的能力。其操作和信息都是可以配置的，且所有配置详情在重启后都将被保留。

默认情况下，进程以占用 CPU 的比例数进行排列，让您可以清楚地看到消耗最多资源的进程。

有关使用 **top** 的详情请参考其 man page：**`man top`**。

`ps`

ps 工具提取活跃进程所选组的快照。默认情况下这个组仅限于当前用户以及与之关联的同一终端拥有的进程。

它可提供比 **top** 更详细的进程信息，但不是动态的。

有关使用 **ps** 的详情请参考其 man page：**`man ps`**。

`vmstat`

vmstat（虚拟内存统计）输出结果可即时报告您系统的进程、内存、页调度、块 I/O、中断以及 CPU 活动。

虽然它不象 **top** 一样提供动态结果，但您可以指定采样间隔，这样可以让您观察类似即使的系统活动。

有关使用 **vmstat** 的详情请参考其 man page：**`man vmstat`**。

`sar`

sar（系统活动报告程序）收集并报告今天到目前为止的系统信息。默认输出结果包括今天的 CPU 使用（10 分钟间隔）：

```
12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal
%idle
12:10:01 AM     all      0.10      0.00      0.15      2.96      0.00
96.79
12:20:01 AM     all      0.09      0.00      0.13      3.16      0.00
96.61
```

```
12:30:01 AM     all      0.09      0.00      0.14      2.11      0.00
97.66
...
```

这个工具是 **top** 或者类似创建系统活动周期性报告的替代工具。

有关使用 **sar** 的详情请参考其 man page：**man sar**。

3.4. TUNED 和 KTUNE

Tuned 是监控并收集各个系统组件用量数据的守护进程，并可使用那些信息根据需要动态调整系统设置。它可以对 CPU 和网络使用的更改作出反应，并调整设置以提高活动设备的性能或者降低不活跃设备的电源消耗。

伴随它的工具 **ktune** 结合 **tuned-adm** 工具提供大量预先配置的调整分析以便在大量具体使用案例中提高性能并降低能耗。编辑这些配置或者创建新配置可生成为系统定制的性能解决方案。

作为 **tuned-adm** 一部分的配置包括：

default

默认节电配置。这是最基本的节点配置。它只启用磁盘和 CPU 插件。注：这与关闭 **tuned-adm** 不同，关闭该程序会同时禁用 **tuned** 和 **ktune**。

latency-performance

典型延迟性能调试的服务器配置。它禁用 **tuned** 和 **ktune** 节能机制。**cpuspeed** 模式改为 **performance**。每个设备的 I/O 提升程序改为 **deadline**。对于服务的电源管理质量，将 **cpu_dma_latency** 设定为 **0**。

throughput-performance

用于典型吞吐性能调整的服务器侧写。如果系统没有企业级存储则建议使用这个侧写。它与 **latency-performance** 相同，只是：

- 将 ***kernel.sched_min_granularity_ns***（调度程序最小优先占用时间间隔）设定为 **10** 毫秒，
- 将***kernel.sched_wakeup_granularity_ns***（调度程序唤醒间隔时间）设定为 **15** 毫秒。
- 将 ***vm.dirty_ratio***（虚拟机脏数据比例）设定为 **40%**，并
- 启用透明超大页面。

enterprise-storage

建议最企业级服务器配置中使用这个侧写，其中包括电池备份控制程序缓存保护以及管理磁盘缓存。它与 **throughput-performance** 配置类似，只是文件系统要使用 **barrier=0** 重新挂载。

virtual-guest

建议最企业级服务器配置中使用这个侧写，其中包括电池备份控制程序缓存保护以及管理磁盘缓存。它与 **throughput-performance** 类似，只是：

- 将 ***readahead*** 值设定为 **4x**，同时
- 不使用 **barrier=0** 重新挂载的 root/boot 文件系统。

virtual-host

根据 ***enterprise-storage*** 配置，***virtual-host*** 还可减少可置换的虚拟内存，并启用更多集合脏页写回。您可以在红帽企业版 Linux 6.3 以及之后的版本中找到这个配置，同时推荐在虚拟化主机中使用这个配置，包括 KVM 和红帽企业版 Linux 虚拟化主机。

有关 **tuned** 和 **ktune** 的详情请参考红帽企业版 Linux 6 *《电源管理指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

3.5. 应用程序分析工具

程序概要分析是收集有关程序执行时其行为的过程。您可以分析一个程序以便决定可以优化程序的哪个部分以便提高该程序的总体速度，减少其内存使用等等。程序分析工具可以帮助您简化这个过程。

红帽企业版 Linux 6 支持三个分析工具：**SystemTap**、**OProfile** 和 **Valgrind**。这些分析工具的具体内容不在本指南讨论范围内，但本小节会为您提供链接，并概述每个分析工具适用的任务。

3.5.1. SystemTap

SystemTap 是一个跟踪和探测工具，可让用户监控并分析操作系统活动（特别是内核活动）的细节。它提供类似 **netstat**、**top**、**ps** 和 **iostat** 等工具的输出结果，但包含为所收集信息的额外过滤和分析选项。

SystemTap 提供深入准确的系统活动和程序行为分析，以便您可以准确包我系统和程序瓶颈。

Eclipse 的功能函数图插件使用 SystemTap 作为后端，可让其完整监控程序状态，其中包括功能调用、返回、次数以及用户空间变量，并以直观形式显示以便优化。

有关 SystemTap 的详情请参考 *《SystemTap 初学者指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

3.5.2. OProfile

OProfile（**oprofile**）是一个系统范围的性能监控工具。它使用处理器专用性能监控硬件搜索有关内核和系统可执行程序的信息，比如何时参考内存，L2 缓存要求数，以及收到的硬件中断数。它还可以用来决定处理器用量，以及使用最多的应用程序和服务。

Oprofile 还可以通过 Eclipse Oprofile 插件与 Eclipse 一同使用。这个插件可以让用户轻松确定其代码中最耗时的部分，并在执行 OProfile 的所有命令行功能时获得最丰富的直观结果。

但用户应该注意到 OProfile 的几个限制：

- 性能监控示例可能不准确因为该处理器可能没有按顺序执行指令，可能是根据最接近的指令执行，而不是触发中断的指令。
- 因为 OProfile 是系统范围内的程序，且会多次启动和停止，多次运行的示例允许有累积。就是说您需要清除以前程序运行产生的示例数据。
- 它主要是识别有 CPU 限制的问题进程，因此无法识别等待为其他事件锁定而处于睡眠状态的进程。

有关使用 OProfile 的详情请参考 *《部署指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/；或者 **/usr/share/doc/oprofile-*<version>*** 中的 **oprofile** 文档。

3.5.3. Valgrind

Valgrind 提供大量探测和分析工具以便帮助您改进性能并修正您的程序。这些工具可以探测与内存和线程有关的错误，以及堆、栈和阵列过度运行，以便您在程序代码中轻松定位并修改错误。他们还可以分析缓存，堆，以及分支预测以便识别增加程序速度，减少程序内存使用的因素。

Valgrind 通过在综合 CPU 运行分析您的程序，并检测其执行的程序代码。然后它会输出“说明”明确为用户指定的文件描述符、文件或者网络插槽鉴别出执行程序所涉及的每个进程。检测等级根据 Valgrind 工具的使用及设置而有所不同，但重要的是注意执行检测的代码的时间比一般执行代码要长 4-50 倍。

Valgrind 可以在您的程序中原封不动地使用，不需要重新编译。但因为 Valgrind 使用调试信息锁定代码中的问题，如果您的程序以及支持库无法使用启用的调试信息编译，则强烈建议您将重新编译包含在这个信息中。

从红帽企业版 Linux 6.4 开始 Valgrind 整合了 gdb (GNU Project Debugger) 以改进调试效率。

有关 Valgrind 的详情请参考 *《开发者指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。或者在安装 valgrind 软件包后查看 **man valgrind** 命令。附带的文档也可在此找到：

- **/usr/share/doc/valgrind-<version>/valgrind_manual.pdf**
- **/usr/share/doc/valgrind-<version>/html/index.html**

有关如何使用 Valgrind 分析系统内存的详情请参考 第 5.3 节 “使用 Valgrind 简要描述内存使用”。

3.5.4. Perf

perf 工具提供大量有用的性能计数器，可让用户评估其系统中其他程序的影响：

perf stat

这个命令常见性能事件的总体统计，其中包括执行的质量以及消耗的时钟周期。您可以使用选项标签收集事件中默认测量事件以外的统计数据。从红帽企业版 Linux 6.4 开始，还可以使用 **perf stat** 过滤根据一个或者多个指定的控制组（cgroup）指定的监控。有关详情请查看 man page：**man perf-stat**。

perf record

这个命令将性能数据记录到文件中，以后可以使用 **perf report** 进行分析。有关详情请查看 man page：**man perf-record**。

perf report

这个命令从文件中读取性能数据并分析记录的数据。有关详情请查看 man page：**man perf-report**。

perf list

这个命令列出具体机器中的可用事件。这些时间随性能监控硬件以及系统软件配置而有所不同。有关详情请查看 man page：**man perf-list**

perf top

这个命令与 **top** 工具的功能类似。它可以实时生成并显示性能计数器分析。有关详情请查看 man page：**man perf-top**。

有关 **perf** 的详情请查看红帽企业版 Linux *《开发者指南》*，地址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

3.6. RED HAT ENTERPRISE MRG

Red Hat Enterprise MRG 的实时组件包括 **Tuna**，它是可让用户调整其系统的可调节参数值，并查看那些更改结果。虽然这个工具是为使用实时组件而开发，但它也可以用于调整标准红帽企业版 Linux 系统。

使用 Tuna，您可以调整或者禁用不必要的系统活动，其中包括：

- 与电源管理有关的 BIOS 参数、错误探测以及系统管理中断；
- 网络设置，比如中断结合以及 TCP 使用；
- 日志文件系统中的日志活动；
- 系统活动记录；
- 中断和用户进程是由具体 CPU 还是一组 CPU 处理；
- 是否使用 swap 空间；以及
- 如何处理内存不足的意外情况。

有关使用 Tuna 界面调整 Red Hat Enterprise MRG 的概念性信息请参考 *《实时调节指南》*中“常规系统调节”一章。有关使用 Tuna 界面的步骤请参考 *《Tuna 用户指南》*。这两本指南的网址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_MRG/。

第 4 章 CPU

CPU 意为*中央处理单元*，对大多数系统来说这个名字并不恰当，因为*中央*暗指*一个*，而大多数现代系统有一个以上的处理单元或者核。通常 CPUs 是一个放在一个包装中附着在主板*插槽*。主板的每个插槽可连接到其他 CPU 插槽、内存控制器、中断控制器以及其他外部设备。连接到操作系统的插槽是一个 CPU 及相关资源的逻辑分组。这个概念是我们要讨论的 CPU 调节中大多数问题的关键。

红帽企业版 Linux 有大量关于系统 CPU 事件的统计。这些统计数据在计划调节策略以改进 CPU 性能时很有帮助。第 4.1.2 节 “调节 CPU 性能” 中讨论了一些游泳的统计，在哪里可以找到它们以及如何就性能调节对其进行分析。

拓扑

在旧的计算机系统中，每个系统只有几个 CPU，因此将其构架称为*对称多处理器*（SMP）。就是说系统中的每个 CPU 对可用内存有类似（或者对称）的访问。近年来，CPU 按插槽计数已完善到可以尝试对系统中的所有内存进行对称访问，但这项技术非常昂贵。大多数高 CPU 计数系统现在才有名为*非均匀内存访问*（NUMA）技术而不是 SMP。

AMD 处理器很久以前就已经在其*超级传输*（HT）互联中采用此架构，同时 Intel 也已最其*快速通道互联*（QPI）设计中使用此架构。NUMA 和 SMP 调节方法各异，因为最为程序分配资源时要考虑此系统的*拓扑*。

线程

Linux 操作系统内部到执行单元被称为*线程*。线程有注册上下文、栈以及在 CPU 中运行到可执行代码片段。操作系统（OS）的任务时在可用 CPU 中调度这些线程。

操作系统根据跨各个核到线程间到负载平衡最大化 CPU 使用。因为 OS 主要考虑的是要让 CPU 处于忙碌状态，它可能没有就程序性能作出最佳选择。将一个程序线程从一个 CPU 移动到其他插槽要比等待 CPU 对性能的影响更大，因为访问内存的操作在跨插槽时会变得非常缓慢。对于高性能程序，设计者最好可以决定要将其放在哪个线程中。第 4.2 节 “CPU 调度” 中讨论了如何以最佳方式分配 CPU 和内存以便最好地执行程序线程。

中断

中断（在 Linux 中被称为 IRQ）是一个不易觉察（但仍很重要）的系统事件，它可以影响程序性能。这些事件由操作系统处理，并用于外部设备以通知数据到达或者操作完成，比如系统写入或者计时器事件。

OS 或者 CPU 执行程序代码处理中断的方法不影响程序功能。但它可能影响程序的性能。本章还讨论了防止中断对程序性能产生负面影响。

4.1. CPU 拓扑

4.1.1. CPU 和 NUMA 拓扑

第一台计算机处理器是*单处理机*，就是说系统只有一个 CPU。让这个操作系统可以平行运行进程的幻想是希望它能够将一个 CPU 迅速从一个执行线程切换到另一个。为提高系统性能，设计者注意到采用提高时钟率提高指令执行速度的方法是有限的（通常受采用当前技术生成稳定时钟波形的限制）。在努力获得更好总体系统性能时，设计者在系统中添加了另一个 CPU，就可以让两个平行流同时运行。这个添加处理器的趋势一直延续至今。

大多数早期的单处理机系统的设计为让每个 CPU 到每个内存位置都使用同一逻辑路径（一般是平行总线）。这样每次 CPU 访问任意位置的内存时与其他系统中的 CPU 对内存的访问消耗的时间是相同的。此类架构就是我们所说的同步多处理器（SMP）系统。SMP 适合 CPU 数较少的系统，但一旦 CPU 计数超过某一点（8 或者 16），要满足对内存的平等访问所需的平行 trace 数就会使用过多的板载资源，留给外设的空间就太少。

有两个新概念整合在一起可以允许系统中有较多的 CPU：

1. 串行总线

2. NUMA 拓扑

串行总线是一个有很高时钟频率的单线通讯路径，以分组突发传送的方式传送数据。硬件设计者开始使用串行总线作为 CPU 之间、CPU 和内存控制器以及其他外设之间的高速互联。就是说不是要求在*每个* CPU 中都有 32 和 64 个板载 trace 连接到内存子系统，现在只要*一个* trace 即可，明显减少了板载空间要求。

同时，硬件设计者通过减小芯片尺寸在同样的空间中放置了更多晶体管。他们不是将独立 CPU 直接放到主板上，而是开始将其打包到处理器包中作为多核处理器。然后设计者不是为每个处理器包提供对等的内存访问，而是借助非均衡存储器访问（NUMA）策略，让每个包/插槽组合有一个或者多个专用内存区以便提供高速访问。每个插槽还有到另一个插槽的互联以便提供对其他插槽内存的低速访问。

作为简单的 NUMA 示例，假设我们有一个双插槽主板，其中每个插槽都有四核。就是说该系统中的 CPU 总数为 8，每个插槽有 4 个。每个插槽还附带 4GB 内存条，内存总数为 8GB。在这个示例中 CPU 0-3 在插槽 0 中，CPU 4-7 在插槽 1 中。这个示例中的每个插槽都对应一个 NUMA 代码。

CPU 0 访问内存条 0 大约需要三个时钟周期：一个周期是将地址发给内存控制器，一个周期是设置对该内存位置的访问，一个周期是读取或者写入到该位置。但 CPU 4 可能需要 6 个时钟周期方可访问内存的同一位置，因为它位于不同的插槽，必须经过两个内存控制器：插槽 1 中的本地内存控制器和插槽 0 中的远程内存控制器。如果在那个位置出现竞争（即如果有一个以上 CPU 同时尝试访问同一位置），内存控制器需要对该内存进行随机且连续的访问，所以内存访问所需时间会较长。添加缓存一致性（保证本地 CPU 缓存包含同一内存位置的相同数据）会让此过程更为复杂。

最新高端处理器，比如 Intel 的 Xeon 和 AMD 的 Opteron 都有 NUMA 拓扑。AMD 处理器使用我们所说的超传输（或称 HT）互联，而 Intel 使用名为快速路径（或称 QPI）的互联。根据其物理连接到其他互联、内存或者外设的情况该互联有所不同，但实际上他们就是可允许从另一台连接的设备对一个连接的设备进行透明访问的开关。在这种情况下，透明指的是使用该互联没有特别的编程 API 要求，而不是“零成本”选项。

因为系统架构千变万化，因此具体指定由于访问非本地内存所致性能代偿是不切实节的。我们可以说每个跨互联的*中继段多少*会产生一些相对恒定的性能代偿，因此参考距当前 CPU 两个互联的内存位置时至少会产生 *2N + 内存周期时间*单位访问时间，其中 N 是每个中继段的代偿。

有这个性能代偿后，对性能敏感的程序应避免常规访问 NUMA 拓扑系统中的远程内存。应将程序设定为使用特定的节点，并从那个节点为其分配内存。

要做到这一点，需要了解程序的一些情况：

1. 系统使用什么*拓扑*？

2. 该程序目前在哪里执行？

3. 最近的内存条在哪里？

4.1.2. 调节 CPU 性能

阅读本小节了解如何调整出更好的 CPU 性能，本小节中还介绍了几个用于此目的的工具。

NUMA 最初是用于将单一处理器连接到多个内存条中。因为 CPU 制造商改进了其工艺并缩小了芯片尺寸，因此可在一个包装中包括多个 CPU 核。这些 CPU 核以集群形式寻租以便每个核都有相同的访问本地内存条的时间，同时可在核之间共享缓存。但每个核、内存以及缓存中跨互联的‘中继段’都有一个小的性能代偿。

图 4.1 “NUMA 拓扑中的本地和远程内存访问” 中的示例系统包括两个 NUMA 节点。每个节点有 4 个 CPU，一个内存条和一个内存控制器，节点中的任意 CPU 都可以直接访问那个节点中的内存条。根据节点 1 中的箭头指示执行步骤如下：

1. CPU（0-3）给出到本地内存控制器的内存地址。
2. 内存控制器设置对内存地址的访问。
3. CPU 在那个内存地址执行读取或者写入操作。

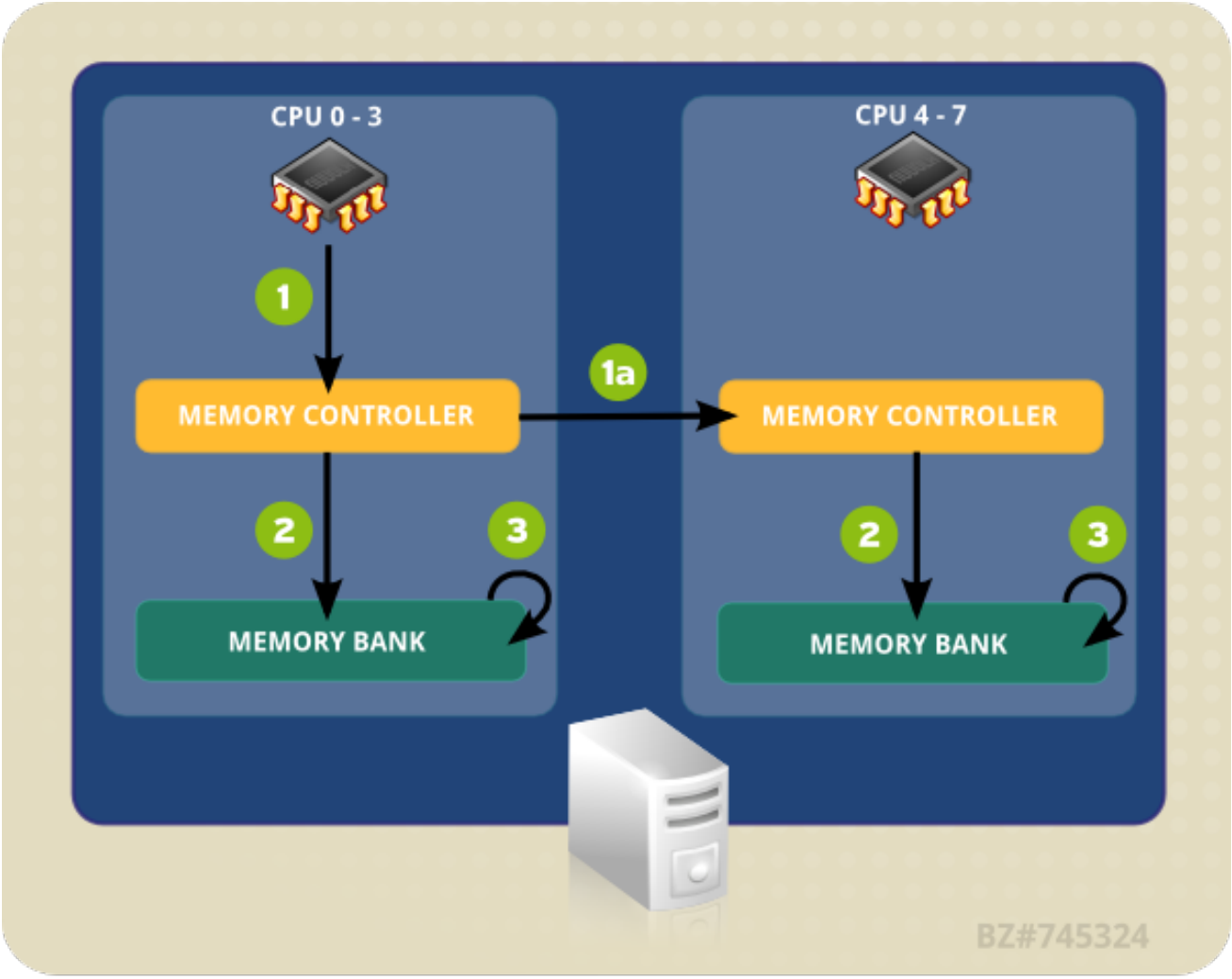


图 4.1. NUMA 拓扑中的本地和远程内存访问

但如果一个节点中的 CPU 需要访问不同 NUMA 节点的内存条中的代码，则它要使用的路径就不那么直接：

1. CPU（0-3）给出到本地内存控制器的远程地址。
    1. 会将对那个远程内存地址的 CPU 请求传递给远程内存控制器，到该节点的本地控制器包含那个内存地址。
2. 远程内存控制器设置对远程内存地址的访问。
3. CPU 在那个远程内存地址执行读取或者写入操作。

每个动作都需要通过多个内存控制器，这样访问在尝试访问远程内存地址时时间会延长两倍以上。因此多核系统中主要性能考量是保证以最有效的方式进行信息传递，即通过最短最迅速的路径。

要为优化 CPU 性能配置程序，您需要了解：

- 系统的拓扑（组件是如何连接的），
- 执行程序的核，以及
- 最接近的内存条位置。

红帽企业版 Linux 6 附带大量可以帮助您找到这个信息并根据您的发现调整系统的工具。以下小节对概述了用于 CPU 性能调节有帮助的工具。

4.1.2.1. 使用 taskset 设置 CPU 亲和性

taskset 搜索并设定运行进程的 CPU 亲和性（根据进程 ID）。它还可用于启动给定 CPU 亲和性的进程，这样就可将指定的进程与指定的 CPU 或者一组 CPU 捆绑。但 **taskset** 不保证本地内存分配。如果您需要本地内存分配的额外性能利益，我们建议您使用 **numactl**，而不是 **taskset**。详情请查看 第 4.1.2.2 节 “使用 numactl 控制 NUMA 策略”。

CPU 亲和性使用位掩码表示。最低位对应第一个逻辑 CPU，且最高位对应最后一个逻辑 CPU。这些掩码通常是十六进制，因此 **0x00000001** 代表处理器 0，**0x00000003** 代表处理器 3 和 1。

要设定运行进程的 CPU 亲和性，请执行以下命令，使用处理器或者您要捆绑到的处理器掩码替换 *mask*，使用您要更改亲和性的进程的进程 ID 替换 *pid*。

```
# taskset -p mask pid
```

要启动给定亲和性的进程，请运行以下命令，使用处理器或者您要捆绑的处理器的掩码替换 *mask*，使用程序、选项以及您要运行的程序参数替换 *program*。

```
# taskset mask -- program
```

与其将处理器指定为位码，您还可以使用 **-c** 选项提供逗号分开的独立处理器，或者一组处理器列表，类似如下：

```
# taskset -c 0,5,7-9 -- myprogram
```

有关 **taskset** 的详情请参考 man page：**man taskset**。

4.1.2.2. 使用 numactl 控制 NUMA 策略

numactl 使用指定的调度或者内存放置策略运行进程。所选策略是为那个进程及其所有子进程设定。**numactl** 还可以为共享内存片段或者文件设定永久策略，并设定 CPU 亲和性和进程的内存亲和性。它使用 **/sys** 文件系统决定系统拓扑。

/sys 文件系统包含有关 CPU、内存和外设是如何通过 NUMA 互联连接的。特别是 **/sys/devices/system/cpu** 目录中包含有关系统的 CPU 是如何互相连接的信息。**/sys/devices/system/node** 目录包含有关系统中 NUMA 节点以及那些节点间相对距离的信息。

在 NUMA 系统中，处理器和内存条之间的距离越大，处理器访问那个内存条的速度就越慢。应将对性能敏感的程序配置为可以从最接近的内存条分配内存。

还应将对性能敏感的程序配置为执行一组核，特别是在多线程程序的情况下。因为以及缓存一般都很小，如果在一个核中执行多个线程，每个线程可有可能逐出由之前线程访问的缓冲的数据。当操作系统尝试在这些线程间执行多任务，且线程继续逐出每个其他的缓存的数据时，则其执行时间的很大比例将用于缓存线替换。这个问题也称*缓存贬值*。因此建议您将多线程的程序捆绑到节点而不是单一核，因为这样可以让线程在多个层级（第一、第二和最后以及缓存）共享缓存线，并尽量减小缓存填充操作的需要。但如果所有线程都访问同一缓存的数据，则将程序捆绑到单一核可能获得高性能。

numactl 可让您将程序捆绑到特定核或者 NUMA 节点，同时要将内存分配到与那个程序关联的核或者一组核。numactl 提供的一些有用选项有：

--show

显示当前进程的 NUMA 策略设置。这个参数不需要进一步的参数，且可按以下方式使用：**numactl --show**。

--hardware

显示系统中可用节点清单。

--membind

只从指定节点分配内存。当使用这个参数时，如果这些节点中的内存不足则分配会失败。这个参数的用法为 **numactl --membind=*nodes program***，其中 *nodes* 是您要从中分配内存的节点列表，*program* 是要从那个节点分配内存的程序。节点号可以采用用逗号分开的列表、范围或者两者的结合方式提供。有关详情请参考 **numactl** man page： **man numactl**

--cpunodebind

只执行属于指定节点的 CPU 中的命令（及其子进程）。这个参数的用法为 **numactl --cpunodebind=*nodes program***，其中 *nodes* 是指定程序（*program*）要捆绑的 CPU 所属节点列表。节点号可以采用用逗号分开的列表、范围或者两者的结合方式提供。有关详情请参考 **numactl** man page： **man numactl**

--physcpubind

只执行指定 CPU 中的命令（及其子进程）。这个参数的用法为 **numactl --physcpubind=*cpu program***，其中 *cpu* 是用逗号分开的物理 CPU 号列表，这些数据在 **/proc/cpuinfo** 的 processor 字段中显示，*program* 是应只在哪些 CPU 中执行的程序。还要将 CPU 指定为与当前 **cpuset** 关联。详情请参考 **numactl** man page：**man numactl**。

--localalloc

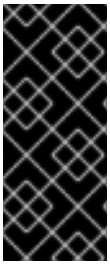
指定永远要在当前节点中分配的内存。

--preferred

在可能的情况下分配到指定节点中的内存。如果内存无法分配到指定的节点，则返回其他节点。这个选项只能有一个节点号，例如：**numactl --preferred=*node***。详情请参考 numactl man page：**man numactl**。

numactl 软件包中包含的 **libnuma** 程序库为内核支持的 NUMA 策略提供编程界面。这比 **numactl** 程序可更详细地调节系统。有关详情请参考 man page：**man numa(3)**。

4.1.3. numastat



重要

之前 **numastat** 工具是由 Andi Kleen 编写的 Perl 脚本。在红帽企业版 Linux 6.4 中对其进行了重大修改。

虽然默认命令（**numastat**，没有任何选项或者参数）可保持与之前版本的严格兼容性，但请注意在这个命令中使用选项或者参数会极大更改输出结果内容及其格式。

numastat 显示进程以及每个 NUMA 节点中操作系统的内存统计数据（比如分配成功数和失败数）。默认情况下，**numastat** 显示每个节点中的以下事件分类所占内存页数。

低 **numa_miss** 和 **numa_foreign** 值表示最佳 CPU 性能。

这个更新的 **numastat** 版本还显示是在系统间分配进程内存，还是使用 **numactl** 在具体的节点中集中使用。

numastat 输出结果与每个 CPU **top** 输出结果对比确定进程线程在分配了内存的同一节点中运行。

默认跟踪分类

numa_hit

为这个节点成功的分配尝试数。

numa_miss

由于在目的节点中内存较低而尝试为这个节点分配到另一个节点的数目。每个 **numa_miss** 事件都在另一个节点中有对应的 **numa_foreign** 事件。

numa_foreign

最初要为这个节点但最后分配个另一个节点的分配数。每个 每个 **numa_foreign** 事件都在另一个节点中有对应的 **numa_miss** 事件。

interleave_hit

成功分配给这个节点的尝试交错策略数。

local_node

这个节点中的进程成功在这个节点中分配内存的次数。

other_node

这个节点中的进程成功在另一个节点中分配内存的次数。

提供任意以下选项可将显示内存单位更改为 MB（四舍五入为两位十进制数），并将其他具体 **numastat** 行为更改如下。

-c

横向紧凑地显示信息表。这对有大量 NUMA 节点的系统很有用，但栏宽度以及栏间空间有时无法预测。使用这个选项时，会将内存值四舍五入到最接近的 MB 数。

-m

显示每个节点中系统范围内的内存使用信息，类似 **/proc/meminfo** 中的信息。

-n

显示与原始 **numastat** 命令类似的信息（numa_hit, numa_miss, numa_foreign, interleave_hit, local_node, and other_node），采用更新的格式，使用 MB 作为测量单位。

-p *pattern*

为指定的模式显示每个节点的内存信息。如果 *pattern* 值由数字组成，**numastat** 假设它是一个数字进程识别符。否则 **numastat** 会为指定的模式搜索进程命令行。

假设在 **-p** 选项值后输入的命令行参数是过滤器的附加模式。附加模式要扩大而不是缩小过滤范围。

-s

以降序模式排列显示的数据以便让最大内存消耗者（根据 **total** 栏）列在首位。

您也可以指定 *node*，并根据 *node* 栏排列表格。当使用这个选项时，*node* 值后必须立即跟上一个 **-s** 选项，如下所示：

```
numastat -s2
```

不要在该选项及其数值之间有空格。

-v

显示更详细的信息。就是说多进程的进程信息会为每个进程显示详细的信息。

-V

显示 **numastat** 版本信息。

-z

省略表格显示的信息中数值为 0 的行和列。注：有些接近 0 的值都四舍五入为 0 以方便显示，这些数值不会在显示的输出结果中省略。

4.1.4. NUMA 亲和性管理守护进程（numad）

numad 是一个自动 NUMA 亲和性管理守护进程，它监控系统中的 NUMA 拓扑以及资源使用以便动态提高 NUMA 资源分配和管理（以及系统性能）。

根据系统负载，**numad** 可对基准性能有 50% 的提高。要达到此性能优势，**numad** 会周期性访问 **/proc** 文件系统中的信息以便监控每个节点中的可用系统资源。该守护进程然后会尝试在有足够内存和 CPU 资源的 NUMA 节点中放置大量进程已优化 NUMA 性能。目前进程管理阈为至少是一个 CPU 的 50%，且至少有 300 MB 内存。**numad** 会尝试维护资源使用水平，并在需要时通过在 NUMA 节点间移动进程平衡分配。

numad 还提供预布置建议服务，您可以通过各种任务管理系统进行查询以便提供 CPU 起始捆绑以及进程内存资源的支持。这个预布置建议服务无论系统中是否运行 **numad** 都可以使用。有关为预布置建议使用 **-w** 选项的详情请参考 man page：**man numad**。

4.1.4.1. numad 的优点

numad 主要可让长期运行消耗大量资源的系统受益，特别是当这些进程是包含在总系统资源子集中时尤为突出。

numad 还可以让消耗相当于多个 NUMA 节点资源的程序受益，但 **numad** 提供的优势可被由于系统增长而消耗的资源比例抵消。

numad 不太可能在进程只运行几分钟或者不会消耗很多资源时改进性能。有连续不可预测内存访问的系统，比如大型内存中的数据库也不大可能从 **numad** 使用中受益。

4.1.4.2. 操作模式

注意

内核内存审核统计数据之间可能会在大规模整合后产生冲突。如果是这样则可能会在 KSM 数据库处整合大量内存时让 **numad** 困惑。将来的 KSM 守护进程版本可能会更好地识别 NUMA。但目前，如果您的系统有大量剩余内存，则可能需要您关闭并禁用 KSM 守护进程方可获得较高的性能。

numad 有两种使用方法：

- 作为服务使用
- 作为可执行文件使用

4.1.4.2.1. 将 numad 作为服务使用

在 **numad** 服务运行时，它将会尝试根据其负载动态调节系统。

要启动该服务，请运行：

```
# service numad start
```

要让该服务在重启后仍保留，请运行：

```
# chkconfig numad on
```

4.1.4.2.2. 将 numad 作为可执行文件使用

要将 **numad** 作为可执行文件使用，请运行：

```
# numad
```

numad 将运行直到将其停止。在它运行时，其活动将被记录到 **/var/log/numad.log** 文件中。

要将 **numad** 限制为管理具体进程，请使用以下选项启动它。

```
# numad -S 0 -p pid
```

-p pid

在明确包含列表中添加指定的 *pid*。**numad** 进程将管理这个指定的进程直到达到其重要阈值。

-S mode

-S 参数指定扫描的进程类型。如示将其设定为 **0** 则将 **numad** 管理明确规定到所包含的进程。

要停止 **numad**，请运行：

```
# numad -i 0
```

停止 **numad** 不会删除它对改进 NUMA 亲和性所做的更改。如果系统使用大量更改，再次运行 **numad** 将调整清河性以便在新条件下提高性能。

有关 **numad** 可用选项的详情请参考 **numad** man page：**man numad**。

4.2. CPU 调度

*调度程序*负责保证系统中的 CPU 处于忙碌状态。Linux 调度程序采用 *调度策略*，它可以决定合适以及在具体 CPU 核中线程运行的时间。

调度策略有两个主要分类：

1. 实时策略
   - SCHED_FIFO
   - SCHED_RR
2. 一般策略
   - SCHED_OTHER
   - SCHED_BATCH
   - SCHED_IDLE

4.2.1. 实时调度策略

首先调度实时线程，所有实时线程调度完成后会调度一般线程。

*实时*策略用于必须无间断完成的关键时间任务。

SCHED_FIFO

这个策略也称作*静态优先调度*，因为它为每个线程规定固定的优先权（在 1 到 99 之间）。该调度程序根据优先权顺序扫描 SCHED_FIFO 线程列表，并调度准备好运行的最高优先权线程。这个线程会运行到它阻断、推出或者被更高的线程抢占准备运行的时候。

即使是最低优先权的实时线程也会比非实时策略线程提前被调度。如果只有一个实时线程，则 **SCHED_FIFO** 优先权值就无所谓了。

SCHED_RR

SCHED_FIFO 策略的轮循变体。也会为 **SCHED_RR** 线程提供 1-99 之间的固定优先权。但有相同优先权的线程使用特定仲裁或者时间片以轮循方式进行调度。**sched_rr_get_interval(2)** 系统调用所有时间片返回的数值，但用户无法设定时间片持续时间。这个策略在您需要以相同的优先权运行多个线程是很有帮助。

有关实时调度策略的规定的语义详情请参考系统界面 – 实时中的《*IEEE 1003.1 POSIX 标准*》，地址为 http://pubs.opengroup.org/onlinepubs/009695399/functions/xsh_chap02_08.html。

定义线程优先权的最佳实践是从低开始，并只在识别了合法延迟时才增加优先权。实时线程不象一般线程

那样是时间片。**SCHED_FIFO** 线程只有在他们阻断、退出或者由更高优先权线程占先时才停止。因此不建议将优先权设定为 99。因为这样会将您的进程放到与迁移和 watchdog 线程相同的优先权等级。如果这些线程因为您的线程进入计算池而被阻断，则他们将无法运行。单处理机系统会在这种情况下平均分配锁定。

在 Linux 内核中，**SCHED_FIFO** 策略包括一个带宽封顶机制。这样可以保护实时应用程序程序员不会受可能独占 CPU 的任务影响。这个机制可通过 **/proc** 文件系统参数进行调整：

/proc/sys/kernel/sched_rt_period_us

以毫秒为单位定义视为 100% CPU 带宽的时间段（'us'是明文中与 'µs' 最接近的表示）。默认值为 1000000µs 或者 1 秒。

/proc/sys/kernel/sched_rt_runtime_us

以毫秒为单位定义用于运行实时线程的时间段（'us'是明文中与 'µs' 最接近的表示）。默认值为 950000µs 或者 0.95 秒。

4.2.2. 一般调度策略

有三个一般调度策略：**SCHED_OTHER**、**SCHED_BATCH** 和 **SCHED_IDLE**。但 **SCHED_BATCH** 和 **SCHED_IDLE** 策略主要用于低优先权任务，因此性能调整指南对其不多做论述。

SCHED_OTHER 或者 SCHED_NORMAL

默认调度策略。该策略使用完全公平调度程序（CFS）提供对所有使用此策略线程的公平访问时间段。CFS 建立了动态优先权列表，部分是根据每个进程线程的 ***niceness*** 值。（有关此参数以及 **/proc** 文件系统的详情请参考*《部署指南》*。）这样可为用户提供一些间接控制进程优先权的权利，但这个动态优先权列表只能由 CFS 直接更改。

4.2.3. 策略选择

为程序线程选择正确的调度程序策略不总是那么直截了当的任务。通常应在关键时间或者需要迅速调度且不能延长运行时间的重要任务中使用实时策略。一般策略通常可已产生比实时策略好的数据流量结果，因为它们让调度进程更有效地运行（即他们不需要经常重新调度占先的进程。

如果您要管理大量进程，且担心数据流量（每秒网络数据包，写入磁盘等等），那么请使用 **SCHED_OTHER**，并让系统为您管理 CPU 使用。

如果您担心事件响应时间（延迟），则请使用 **SCHED_FIFO**。如果您只有少量线程，则可以考虑隔离 CPU 插槽，并将线程移动到那个插槽的核中以便没有其他线程与之竞争。

4.3. 中断和 IRQ 调节

中断请求（IRQ）是用于服务的请求，在硬件层发出。可使用专用硬件线路或者跨硬件总线的信息数据包（消息信号中断，MSI）发出中断。

启用中断后，接收 IRQ 后会提示切换到中断上下文。内核中断调度代码会搜索 IRQ 号码机器关联的注册中断服务路由（ISR）列表，并按顺序调用 ISR。ISR 会确认中断并忽略来自同一 IRQ 的多余中断，然后在延迟的句柄中排队完成中断处理，并忽略以后的中断来结束 ISR。

/proc/interrupts 文件列出每个 I/O 设备中每个 CPU 的中断数，每个 CPU 核处理的中断数，中断类型，以及用逗号分开的注册为接收中断的驱动程序列表。（详情请参考 proc(5) man page：**man 5 proc**）

IRQ 有一个关联的“类似”属性 ***smp_affinity***，该参数可以定义允许为 IRQ 执行 ISR 的 CPU 核。这个属性还用来提高程序性能，方法是为一个或者多个具体 CPU 核分配中断类似性和程序线程类似性。这可让缓存线可在指定的中断和程序线程之间共享。

具体 IRQ 数的中断近似性值是保存的相关的 **/proc/irq/*IRQ_NUMBER*/smp_affinity** 文件中，您可以作为 root 用户查看并修改该值。保存在这个文件中的值是一个十六进制字节掩码，代表系统中所有 CPU 核。

例如：要为四核服务器指定以太网驱动程序，首先要确定与该以太网驱动程序关联的 IRQ 数：

```
# grep eth0 /proc/interrupts
32:   0     140      45       850264      PCI-MSI-edge      eth0
```

使用 IRQ 数定位正确的 ***smp_affinity*** 文件：

```
# cat /proc/irq/32/smp_affinity
f
```

smp_affinity 的默认值为 **f**，即可为系统中任意 CPU 提供 IRQ。将这个值设定为 **1**，如下，即表示只有 CPU 0 可以提供这个中断：

```
# echo 1 >/proc/irq/32/smp_affinity
# cat /proc/irq/32/smp_affinity
1
```

可使用逗号为不连续的 32 位组限定 ***smp_affinity*** 值。在有 32 个以上核的系统有这个要求。例如：以下示例显示在一个 64 核系统的所有核中提供 IRQ 40。

```
# cat /proc/irq/40/smp_affinity
ffffffff,ffffffff
```

要只在 64 核系统的上 32 核中提供 IRQ 40，请执行：

```
# echo 0xffffffff,00000000 > /proc/irq/40/smp_affinity
# cat /proc/irq/40/smp_affinity
ffffffff,00000000
```

注意

在支持*中断操作*的系统中，修改 IRQ 的 ***smp_affinity*** 设置硬件以便决定在不影响内核的情况下，硬件层为具体 CPU 提供中断服务。

4.4. 红帽企业版 LINUX 6 中 NUMA 的改进

红帽企业版 Linux 6 包括大量改进以便充分利用当今高度可扩展的硬件。本小节对由红帽企业版 Linxu 6 提供的大多数与 NUMA 相关的重要性能改进进行概述。

4.4.1. 裸机和可扩展性优化

4.4.1.1. 拓扑识别改进

以下改进可让红帽企业版 Linux 探测底层硬件和架构详情，提高其在系统中进行自动优化的功能。

提高的拓扑探测

这可让操作系统探测到引导时的底层硬件详情（比如逻辑 CPU，超线程，核，插槽，NUMA 节点以及节点间访问时间），并优化系统进程。

完全公平调度程序

这个新的调度模式可以保证在有效进程间平均分配运行时间。这个模式与拓扑探测联合使用可将进程在同一插槽的 CPU 中调度以避免昂贵的远程内存访问，同时保证随时保留缓存内容。

`malloc`

`malloc` 现在已经优化，可保证分配给某个进程的内存区尽量接近执行该内存的核。这样可以增加内存访问速度。

skbuff I/O 缓存分配

与 **`malloc`** 类似，现已将其优化为使用与处理 I/O 操作（比如设备中断）的 CPU 最接近的内存。

设备中断亲和性

设备驱动程序记录的关于哪个 CPU 处理哪个中断的信息可用来限制在同一物理插槽中的 CPU 处理的中断，保留缓存亲和性并限制高容量跨插槽通讯。

4.4.1.2. 改进多核处理器同步

协调多个处理器之间的任务需要频繁、耗时的操作以便保证平行执行的进程不会损害数据完整性。红帽企业版 Linux 包括以下改进以提高此方面的性能：

读-拷贝-更新（RCU）锁

通常 90% 的锁定是用于只读目的。RCU 锁定移除了在未修改数据访问时获得独家访问锁定的要求。这个锁定模式现已用于页缓存分配：现在锁定只可用于分配或者取消分配动作。

按 CPU 以及按插槽进行计算的算法

很多算法已更新至在同一插槽的合作 CPU 之间执行锁定协作以便允许更细致地调整锁定。大量全局自旋锁已使用按插槽锁定方法替换，且更新的内存分配程序区以及相关的内存页列表可在执行分配或者取消分配操作时让内存分配逻辑贯穿更有效的内存匹配数据结构子集。

4.4.2. 虚拟化优化

因为 KVM 使用内核功能，所以基于 KVM 的虚拟机可立即受益于所有裸机优化。红帽企业版 Linux 还包括大量可让虚拟机进入逻辑性能层的改进。这些改进注重存储和网络访问中的 I/O 路径，甚至可以让加强负荷（比如数据库以及访问服务）利用虚拟的部署。可提高虚拟系统性能的针对 NUMA 的改进包括：

CPU pinning

可将虚拟机捆绑到具体插槽中以便优化本地缓存使用，并删除昂贵的插槽间通讯和远程内存访问的需要。

透明大页面（THP）

启用 THP 后，系统可为大量连续内存自动执行 NUMA 可识别的内存分配要求，减少内存锁竞争量和所需转移后备缓冲器（TLB）内存管理操作，并可在虚拟机中将性能提高达 20%。

基于内核的 I/O 实施

虚拟机 I/O 子系统现在已在内核中部署，这样可极大降低节点间通讯和内存访问，方法是避免大量上下文切换，减少同步和通讯费用。

第 5 章 内存

预读本章了解红帽企业版 Linux 中可用的内存管理功能，以及如何使用这些管理功能优化系统的内存使用。

5.1. 超大转译后备缓冲器（HUGETLB）

将物理内存地址转译为性能内存地址是内存管理的一部分。物理地址和虚拟地址的映射关系保存在名为页表的数据结构中。因为为每个地址映射读取页表会很消耗时间和资源，所以最近使用的地址都有缓存。这个缓存就称为转译后备缓冲器（TLB）。

但 TLB 只能缓存大量地址映射。如果所要求的地址映射没有在 TLB 中，则必须仍读取页表以决定物理到虚拟地址映射。这就是所谓的"TLB 缺失"。因为其内存要求与用来缓存 TLB 中地址映射的页面之间的关系，所以有大内存要求的程序比使用较少内存的程序更容易受 TLB 缺失影响。因为每个缺失都涉及页表读取，因此尽量避免这些缺失很重要。

超大转译后备缓冲器（HugeTLB）可以很大片段管理内存，以便一次可以缓存更多地址。这样可减少 TLB 缺失的可能性，进而改进有大内存要求的程序性能。

有关配置 HugeTLB 的信息可在内核文档中找到：**/usr/share/doc/kernel-doc-*version*/Documentation/vm/hugetlbpage.txt**

5.2. 大页面和透明大页面

内存是由块管理，即众所周知的*页面*。一个页面有 4096 字节。1MB 内存等于 256 个页面。1GB 内存等于 256000 个页面等等。CPU 有内嵌的*内存管理单元*，这些单元中包含这些页面列表，每个页面都使用*页表条目*参考。

让系统管理大量内存有两种方法：

- 增加硬件内存管理单元中页表数
- 增大页面大小

第一个方法很昂贵，因为现代处理器中的硬件内存管理单元只支持数百或者书签页表条目。另外适用于管理数千页面（MB 内存）硬件和内存管理算法可能无法很好管理数百万（甚至数十亿）页面。这会造成性能问题：但程序需要使用比内存管理单元支持的更多的页面，该系统会退回到缓慢的基于软件的内存管理，从而造成整个系统运行缓慢。

红帽企业版 Linux 6 采用第二种方法，即使用*超大页面*。

简单说，超大页面是 2MB 和 1GB 大小的内存块。2MB 使用的页表可管理多 GB 内存，而 1GB 页是 TB 内存的最佳选择。

差大页面必须在引导时分配。它们也很难手动管理，且经常需要更改代码以便可以有效使用。因此红帽企业版 Linux 也部署了*透明超大页面*(THP)。THP 是一个提取层，可自动创建、管理和使用超大页面的大多数方面。

THP 系统管理员和开发者减少了很多使用超大页面的复杂性。因为 THP 的目的是改进性能，所以其开发者（社区和红帽开发者）已在各种系统、配置、程序和负载中测试并优化了 THP。这样可让 THP 的默认设置改进大多数系统配置性能。

注：THP 目前只能映射异步内存区域，比如堆和栈空间。

5.3. 使用 VALGRIND 简要描述内存使用

Valgrind 是为用户空间二进制提供检测的框架。它与大量用于简要描述和分析程序性能的工具一同发布。本小节提供的工具可用于探测内存错误，比如使用未初始化内存以及不正确地分配或者取消分配内存。这些工具都包含在 valgrind 软件包中，并可使用以下命令运行：

```
valgrind --tool=toolname program
```

使用您要使用的工具名称替换 *toolname*（要对内存进行简要概述，请使用 **memcheck**, **massif** 或者 **cachegrind**），同时使用您要使用 Valgrind 进行简要概述的程序替换 *program*。请注意 Valgrind 的检测可能造成程序比正常情况运行更缓慢。

Valgrind 的功能概述请参考 第 3.5.3 节 “Valgrind”。详情，包括用于 Eclipse 的插件请参考 *《开发者指南》*，其链接为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。您还可以在安装 valgrind 软件包后使用 **man valgrind** 命令查看相关文档，或者在以下位置查找这些文档：

- **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf** 以及
- **/usr/share/doc/valgrind-*version*/html/index.html**。

5.3.1. 使用 Memcheck 简要概述内存使用

Memcheck 是默认 Valgrind 工具，且可以与 **valgrind *program*** 一同运行，而无需指定 **--tool=memcheck**。它可探测和报告大量很难探测和诊断的内存错误，比如不应该发生的内存访问，使用未定义或者未初始化值，错误释放的堆内存，重叠的光标以及内存泄漏。运行 Memcheck 时程序运行速度要比正常运行时慢 10-30 倍。

Memcheck 根据探测的问题类型返回具体错误。这些错误在 Valgrind 文档中有具体论述，文档位置为 **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**。

注：Memcheck 只能报告这些错误 – 它不能放置这些错误发生。如果您的程序采用一般会造成片段失败的的方法访问内存，片段失败仍会发生。但 Memcheck 将在失败前记录出错信息。

Memcheck 提供可用来着重检查进程的命令行选项。有些可用的选项为：

--leak-check

启用后，Memcheck 会在客户端程序完成后搜索内存泄漏。其默认值为 **summary**，它输出找到的泄漏数。其他可能的值为 **yes** 和 **full**，这两个选项都会给出每个泄漏的详细情况，且 **no** 会禁用内存泄漏检查。

--undef-value-errors

启用后（将其设定为 **yes**），Memcheck 会报告使用未定义值报告的错误。禁用时（将其设定为 **no**），则不会报告未定义值错误。默认启用这个选项。禁用该选项会稍稍提高 Memcheck 速度。

--ignore-ranges

可让用户指定一个或者多个 Memcheck 检查可寻址能力时应该忽略的范围。多个范围使用逗号分开，例如：**--ignore-ranges=0xPP-0xQQ,0xRR-0xSS**。

选项完整列表请参考 **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf** 中的文档。

5.3.2. 使用 Cachegrind 简要概述缓存使用

Cachegrind 模拟您的程序与机器缓存等级和（可选）分支预测单元的互动。它跟踪模拟的一级指令和数据缓存的用量以便探测不良代码与这一级缓存的互动；最高一级，可以是二级或者三级缓存，用来跟踪对主内存的访问。因此，使用 Cachegrind 运行的程序速度比正常运行时要慢 20-100 倍。

要运行 Cachegrind 请执行以下命令，使用您要用 Cachegrind 简要描述的程序替换 *program*。

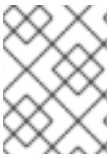
```
# valgrind --tool=cachegrind program
```

Cachegrind 可以为整个程序以及该程序中的每个功能收集统计数据：

- 一级指令缓存读取（或者执行的指令）和读取缺失，最后一级缓存指令读取缺失；
- 数据缓存读取（或者内存读取），读取缺失，以及最高一级缓存数据读取缺失；
- 数据缓存写入（或者内存写入），写ur缺失，以及最高一级缓存数据写入缺失；
- 已执行和无法预测的条件分支；以及
- 已执行和无法预测的间接分支。

Cachegrind 输出控制台的这些统计数据信息小结，并在文件（默认为 **cachegrind.out.*pid***，其中*pid* 为您运行 Cachegrind 的程序的进程 ID）中写入更详细的配置信息。该文件可由 **cg_annotate** 进行进一步的处理，比如：

```
# cg_annotate cachegrind.out.pid
```



注意

cg_annotate 可以输出 120 字符以上，具体要看路径的长度。要让输出结果更清晰、易读，我们建议您在执行上述命令前将终端窗口至少调整到这个宽度。

您还可以比较 Cachegrind 生成的概述文件，将其简化为更改前后的图标对比图。要这样做请使用 **cg_diff** 命令，使用最初的概述输出文件替换 *first*，并使用随后的概述输出文件替换 *second*：

```
# cg_diff first second
```

这个命令提供合并的输出文件，您可以使用 **cg_annotate** 查看更详细的结果。

Cachegrind 支持大量选项注重其输出结果。有些可用选项为：

--I1

指定大小，关联性以及一级指令缓存的块大小，以逗号分开：**--I1=*size*,*associativity*,*line size***。

--D1

指定大小，关联性以及一级数据缓存的块大小，以逗号分开：**--D1=*size*,*associativity*,*line size***。

--LL

指定大小，关联性以及最后一级指令缓存的块大小，以逗号分开：**--LL=*size*,*associativity*,*line size***。

--cache-sim

启用或者禁用缓存访问和缺失计数集合。默认值为 **yes**（启用）。

注：禁用这个选项和 **--branch-sim** 选项让 Cachegrind 误信息可以收集。

--branch-sim

启用或者禁用分支指令和无法预测计数集合。默认将其设定为 **no**（禁用），因为它可让 Cachegrind 延缓 25%。

注：禁用这个选项和 **--cache-sim** 选项让 Cachegrind 误信息可以收集。

选项完整列表请参考 **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf** 中的文档。

5.3.3. 使用 Massif 查看堆和栈空间配置

Massif 使用指定的程序测量堆 空间，包括有用空间以及用于记录和对齐而分配的额外空间。它可以帮助您减少程序使用的内存量，增加程序速度，并减少程序耗尽机器 swap 空间的可能性。Massif 还可以提供有关您程序用来分配堆 内存的部分的详情。使用 Massif 运行的程序的运行速度比其一般执行速度慢 20 倍。

要给出程序堆 用量信息，请将 **massif** 指定为您要使用的 Valgrind 工具：

```
# valgrind --tool=massif program
```

Massif 收集的配置数据会写入一个文件，默认为 **massif.out.*pid***，其中 *pid* 是指定的 *program* 的进程 ID。

还可以使用 **ms_print** 命令绘制配置数据图，比如：

```
# ms_print massif.out.pid
```

这样可以生成显示程序执行期间内存消耗的图表，以及有关在程序中不同点进行分配的地点详情，其中包括峰值内存分配点。

Massif 提供大量命令行选项可用于该工具的直接输出结果。这些可用选项包括：

--heap

指定是否执行堆 分析。默认值为 **yes**。将此选项设定为 **no** 即可禁用堆 分析。

--heap-admin

指定启用堆 分析时每个块用于管理的字节数。默认每个块有 **8** 字节。

--stacks

指定是否执行栈分析。默认值为 **no**（禁用）。要启用栈分析，请将这个选项设定为 **yes**，但请注意这样做会极大降低 Massif 速度。另外还要注意 Massif 假设开始时主栈大小为 0 以便更好的显示要分析的程序的栈所控制的比例。

--time-unit

指定用来分析的时间单位。这个选项三个有效值：执行的指令（**i**），即默认值，用于大多数情况；即时（**ms**，单位毫秒），可用于某些特定事务；以及在堆（**/**或者）栈中分配/取消分配的字节（**B**），用

于很少运行的程序，且用于测试目的，因为它最容易在不同机器中重现。这个选项在使用 **ms_print** 输出结果画图是游泳。

选项完整列表请参考 **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf** 中的文档。

5.4. 容量调节

本小节总结了内存、内核以及文件系统容量，与每一部分相关的参数以及调节这些参数所涉及的交换条件。

要在调节时临时设定这些值，请将所需值 echo 到 proc 文件系统中的适当文件中。例如：要将 ***overcommit_memory*** 临时设定为 **1**，请运行：

```
# echo 1 > /proc/sys/vm/overcommit_memory
```

注：到 proc 文件系统中该参数的路径要具体看此变更所影响的系统。

要永久设定这些值，则需要使用 **sysctl** 命令。有关详情请参考 *《部署指南》*，网址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

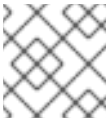
与容量相关的内存可调参数

以下参数位于 proc 文件系统的 **/proc/sys/vm/** 目录中。

overcommit_memory

规定决定是否接受超大内存请求的条件。这个参数有三个可能的值：

- **0** – 默认设置。内核执行启发式内存过量使用处理，方法是估算可用内存量，并拒绝明显无效的请求。遗憾的是因为内存是使用启发式而非准确算法计算进行部署，这个设置有时可能会造成系统中的可用内存超载。
- **1** – 内核执行无内存过量使用处理。使用这个设置会增大内存超载的可能性，但也可以增强大量使用内存任务的性能。
- **2** – 内存拒绝等于或者大于总可用 swap 大小以及 ***overcommit_ratio*** 指定的物理 RAM 比例的内存请求。如果您希望减小内存过度使用的风险，这个设置就是最好的。



注意

只为 swap 区域大于其物理内存的系统推荐这个设置。

overcommit_ratio

将 ***overcommit_memory*** 设定为 **2** 时，指定所考虑的物理 RAM 比例。默认为 **50**。

max_map_count

规定某个进程可能使用的最大内存映射区域。在大多数情况下，默认值 **65530** 就很合适。如果您的程序需要映射比这个文件数更多的文件可增大这个值。

nr_hugepages

规定在内核中配置的超大页数。默认值为 0。只有系统中有足够的连续可用页时方可分配（或者取消分配）超大页。为这个参数保留的页无法用于其他目的。安装的文件 **/usr/share/doc/kernel-doc-*kernel_version*/Documentation/vm/hugetlbpage.txt** 中有详细的内容。

与容量相关的内核可调参数

以下参数位于 proc 文件系统的 **/proc/sys/kernel/** 目录中。

msgmax

以字节为单位规定信息队列中任意信息的最大允许大小。这个值一定不能超过该队列的大小（***msgmnb***）。默认值为 **65536**。

msgmnb

以字节为单位规定单一信息队列的最大值。默认为 **65536** 字节。

msgmni

规定信息队列识别符的最大数量（以及队列的最大数量）。64 位架构机器的默认值为 **1985**；32 位架构机器的默认值为 **1736**。

shmall

以字节为单位规定一次在该系统中可以使用的共享内存总量。64 位架构机器的默认值为 **4294967296**；32 位架构机器的默认值为 **268435456**。

shmmax

以字节为单位规定内核可允许的最大共享内存片段。64 位架构机器的默认值为 **68719476736**；32 位架构机器的默认值为 **4294967295**。注：但内核支持的值比这个值要多得多。

shmmni

规定系统范围内最大共享内存片段。在 64 位和 32 位架构机器中的默认值都是 **4096**。

threads-max

规定一次内核使用的最大线程（任务）数。默认值与 ***max_threads*** 相同。使用的方程式是：

```
max_threads = mempages / (8 * THREAD_SIZE / PAGE_SIZE )
```

threads-max 的最小值为 **20**。

与容量相关的文件系统可调参数

以下参数位于 proc 文件系统的 **/proc/sys/fs/** 目录中。

aio-max-nr

规定在所有活动异步 I/O 上下文中可允许的最多事件数。默认值为 **65536**。注：更改这个值不会预先分配或者重新定义内核数据结构大小。

file-max

列出内核分配的文件句柄最大值。默认值与内核中的 ***files_stat.max_files*** 映射，该参数可将最大值设定为 **(mempages * (PAGE_SIZE / 1024)) / 10** 或者 **NR_FILE**（在红帽企业版 Linux 中是 8192）。增大这个值可解决由于缺少文件句柄而造成的错误。

Out-of-Memory Kill 可调参数

内存不足（OOM）指的是所有可用内存，包括 swap 空间都已被分配的计算状态。默认情况下，这个状态可造成系统 panic，并停止如预期般工作。但将 ***/proc/sys/vm/panic_on_oom*** 参数设定为 **0** 会让内核在出现 OOM 时调用 **oom_killer** 功能。通常 **oom_killer** 可杀死偷盗进程，并让系统正常工作。

可在每个进程中设定以下参数，提高您对被 **oom_killer** 功能杀死的进程的控制。它位于 proc 文件系统中 **/proc/*pid*/** 目录下，其中 *pid* 是进程 ID。

oom_adj

定义 **-16** 到 **15** 之间的一个数值以便帮助决定某个进程的 ***oom_score***。***oom_score*** 值越高，被 **oom_killer** 杀死的进程数就越多。将 ***oom_adj*** 值设定为 -17 则为该进程禁用 **oom_killer**。

重要

由任意调整的进程衍生的任意进程将继承该进程的 ***oom_score***。例如：如果 **sshd** 进程不受 **oom_killer** 功能影响，所有由 SSH 会话产生的进程都将不受其影响。这可在出现 OOM 时影响 **oom_killer** 功能救援系统的能力。

5.5. 调整虚拟内存

虚拟内存一般由进程、文件系统缓存以及内核消耗。虚拟内存的使用由很多因素决定，受以下参数影响：

swappiness

参数值可为 0-100，控制系统 swap 的程序。高数值可优先系统性能，在进程不活跃时主动将其转换出物理内存。低数值可优先互动性并尽量避免将进程转换处物理内存，并降低反应延迟。默认值为 **60**。

min_free_kbytes

保证系统间可用的最小 KB 数。这个值可用来计算每个低内存区的水印值，然后为其大小按比例分配保留的可用页。

警告

设定这个参数时请小心，因为该值过低和过高都有问题。

min_free_kbytes 太低可防止系统重新利用内存。这可导致系统挂起并让 OOM 杀死多个进程。

但将这个参数值设定太高（占系统总内存的 5-10%）会让您的系统很快会内存不足。Linux 的设计是使用所有可用 RAM 缓存文件系统数据。设定高 ***min_free_kbytes*** 值的结果是在该系统中花费太多时间重新利用内存。

dirty_ratio

规定百分比值。当脏数据组成达到系统内存总数的这个百分比值后开始写下脏数据（**pdflush**）。默认值为 **20**。

dirty_background_ratio

规定百分比值。当脏数据组成达到系统内存总数的这个百分比值后开始在后端写下脏数据（**pdflush**）。默认值为 **10**。

drop_caches

将这个值设定为 **1**、**2** 或者 **3** 让内核放弃各种页缓存和 slab 缓存的各种组合。

1

系统无效并释放所有页缓冲内存。

2

系统释放所有未使用的 slab 缓冲内存。

3

系统释放所有页缓冲和 slab 缓冲内存。

这是一个非破坏性操作。因为无法释放脏项目，建议在运行 **sync** 设定这个参数值。

重要

不建议在产品环境中使用 ***drop_caches*** 释放内存。

要在调节时临时设定这些值，请将所需值 echo 到 proc 文件系统中的适当文件中。例如：要将 ***swappiness*** 临时设定为 **50**，请运行：

```
# echo 50 > /proc/sys/vm/swappiness
```

要永久设定这个值，则需要使用 **sysctl** 命令。有关详情请参考 *《部署指南》*，网址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

第 6 章 输入/输出

6.1. 功能

红帽企业版 Linux 6 引进了大量 I/P 栈性能提高：

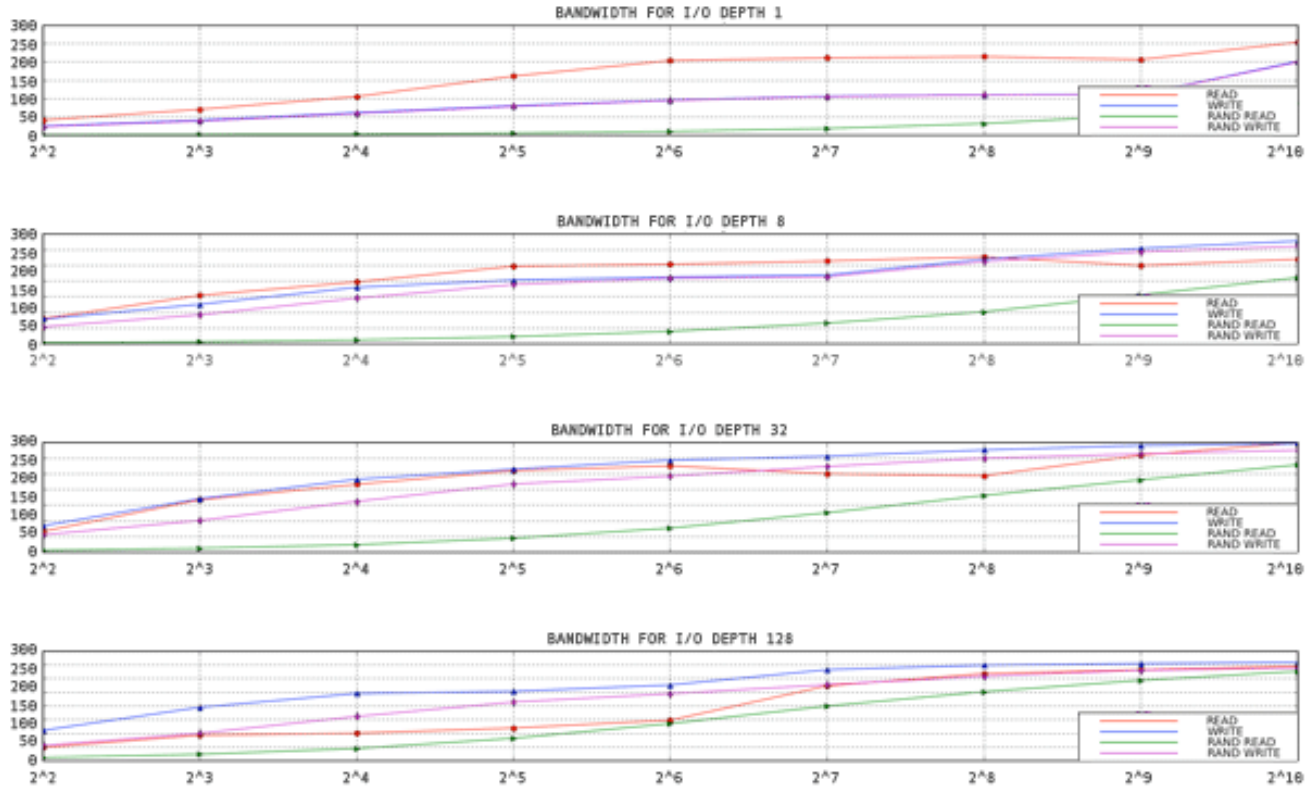
- 现在可自动识别固态磁盘（SSD），且 I/O 调度程序的性能可利用这些设备每秒可执行的 I/O（IOPS）次数较高。
- 已在内核中添加了忽略支持以便向基础存储报告未使用的块。这样可以帮助 SSD 进行耗损平衡计算。还可以帮助支持逻辑块分配的存储（类似存储的虚拟地址空间），方法是具体观察实际使用的存储量。
- 红帽企业版 Linux 6.1 会大量更改屏障使用以便提高其性能。
- 已使用 per-backing-device 清空线程替换 **pdflush**，可在使用大量 LUN 计数配置时极大改进系统灵活性。

6.2. 分析

成功调整存储栈性能需要了解数据在系统中的流程，并精通基础存储知识以及它在各种负载下的工作原理。它还要求理解要调整的实际负载。

无论合适您部署新系统时，最好彻底分析该存储。从原始 LUN 或者磁盘开始，并使用直接 I/O（绕过内核也缓存的 I/O）评估其性能。这是您可以执行的最基本测试，同时也将成为您检测栈中 I/O 性能的标准。请使用基本负载生成器（比如 **aio-stress**）可生成各种 I/O 大小和队列深度的连续和随机读取及写入。

接下来是来自 **aio-stress** 运行的系列图表，每个运行有四个阶段：顺序写入、顺序读取、随机写入和随机读取。在这个示例中，将该工具配置为以不同记录大小（**x** 轴）和队列深度（每图一个）运行。队列深度代表在给定时间进程的 I/O 操作总和。



y 轴以 MB/秒为单位显示带宽。x 轴以 Kb 为单位显示 I/O 大小。

图 6.1. 一线程、一文件的 aio-stress 输出结果

注意从左下角到右上角的流量线趋势。还请注意在给定记录大小时，您可以通过增加进程中的 I/O 数从存储中获得更多流量。

通过根据您的存储运行这些简单负载，您可以了解在此负载下的存储性能。保留这些测试生成的数据用来在分析更复杂的负载时进行比较。

如果您要使用设备映射或者 md，在下一步添加那一层并重复您的测试。如果性能损失严重，请确定那是预期的或者可以解释的。例如：如果在该栈中添加 checksumming raid 层则一定会有性能下降。意外的性能损失可能由无法同步的 I/O 操作造成。默认情况下红帽企业版 Linux 以最佳方式对齐分区和设备映射器元数据。但不是所有类型的存储都报告其最佳校准，因此可能需要手动调整。

软件设备映射或者 md 层后，在块设备顶端添加一个文件系统并根据其进行测试，仍使用直接 I/O。同样，将结果与之前测试的结果进行比较，保证您理解所有不符之处。直接写入 I/O 通常在预分配文件中性能更好，因此请确定在测试性能前预先分配文件。

您可能会认为有用的人工负载生成器包括：

- **aio-stress**
- **iozone**
- **fio**

6.3. 工具

有一些工具可用来帮助您诊断 I/O 子系统中的性能问题。**vmstat** 提供系统性能的粗略概述。以下栏与 I/O 相关：***si***（换入），***so***（换出），***bi***（阻止进入），***bo***（阻止外出），以及***wa***（I/O 等待时间）。当您的交换空间与数据分区同在一个设备中时 ***si*** 和 ***so*** 有用，且表示总体内存压力。***si*** 和 ***bi*** 是读取操

作，***so*** 和 ***bo*** 是写入操作。这些分类以 Kb 为单位报告。 ***wa*** 是停滞时间，表示在等待 I/O 完成时哪部分运行队列被阻断。

使用 **vmstat** 分析系统可让您了解 I/O 子系统是否对性能问题负责。 ***free***、***buff*** 和 ***cache*** 栏最值得关注。***cache*** 值会随着 ***bo*** 值增加，随着 ***cache*** 的降低 ***free*** 值会增大，表示系统正在执行写回操作，且无法使用页缓存。

注：***vmstat*** 报告的 I/O 数是集合了所有设备的 I/O 总数。如果您确定 I/O 子系统中可能有性能问题，则可以使用 **iostat** 做进一步的检查，该程序可以根据设备报告 I/O 情况。您还可以搜索更多详细信息，比如平均需求大小，每秒读取和写入数以及正在进行的 I/O 合并数量。

使用平均请求大小和平均队列大小（**avgqu-sz**）可预估如何使用在描述存储性能时生成的图表执行存储。可采用一些概括估计方法：例如，如果平均请求大小为 4KB，且平均队列大小为 1，则不大可能有非常高的性能。

如果性能值与您期待的性能值不同，可使用 **blktrace** 执行更详细的分析。**blktrace** 工具套件给出 I/O 子系统使用时间的详情。**blktrace** 的输出结果是一组二进制跟踪文件，可以使用其他工具进行后处理，比如 **blkparse**。

blkparse 是 **blktrace** 的配伍工具。它读取跟踪的原始输出并生成一手文字版本。

以下是 **blktrace** 输出结果示例：

```
8,64   3        1     0.000000000  4162  Q  RM 73992 + 8 [fs_mark]
8,64   3        0     0.000012707     0  m   N cfq4162S / alloced
8,64   3        2     0.000013433  4162  G  RM 73992 + 8 [fs_mark]
8,64   3        3     0.000015813  4162  P   N [fs_mark]
8,64   3        4     0.000017347  4162  I   R 73992 + 8 [fs_mark]
8,64   3        0     0.000018632     0  m   N cfq4162S / insert_request
8,64   3        0     0.000019655     0  m   N cfq4162S / add_to_rr
8,64   3        0     0.000021945     0  m   N cfq4162S / idle=0
8,64   3        5     0.000023460  4162  U   N [fs_mark] 1
8,64   3        0     0.000025761     0  m   N cfq workload slice:300
8,64   3        0     0.000027137     0  m   N cfq4162S / set_active
wl_prio:0 wl_type:2
8,64   3        0     0.000028588     0  m   N cfq4162S / fifo=(null)
8,64   3        0     0.000029468     0  m   N cfq4162S / dispatch_insert
8,64   3        0     0.000031359     0  m   N cfq4162S / dispatched a
request
8,64   3        0     0.000032306     0  m   N cfq4162S / activate rq,
drv=1
8,64   3        6     0.000032735  4162  D   R 73992 + 8 [fs_mark]
8,64   1        1     0.004276637     0  C   R 73992 + 8 [0]
```

如您所见，该输出结果非常紧凑且很难读懂。在这个结果中您可以看到负责向您的设备发出 I/O 的进程，是很有用的。但 **blkparse** 会在其概述中以容易理解的方式给出一些附加信息。**blkparse** 的概述信息在输出结果的最后面：

```
Total (sde):
Reads Queued:          19,       76KiB  Writes Queued:     142,183,
568,732KiB
Read Dispatches:       19,       76KiB  Write Dispatches:   25,440,
568,732KiB
Reads Requeued:         0               Writes Requeued:       125
Reads Completed:       19,       76KiB  Writes Completed:   25,315,
568,732KiB
```

```
Read Merges:            0,        0KiB  Write Merges:      116,868,
467,472KiB
IO unplugs:        20,087               Timer unplugs:           0
```

概述显示平均 I/O 速度、合并活动，并对比读负载和写负载。但 **blkparse** 输出结果因太过繁琐而变得没有意义。还好有一些工具可以帮助您解析这些数据：

btt 提供对 I/O 在 I/O 栈中不同区域所花费时间的分析。这些区域为：

- Q – 将块 I/O 排队
- G – 获得请求

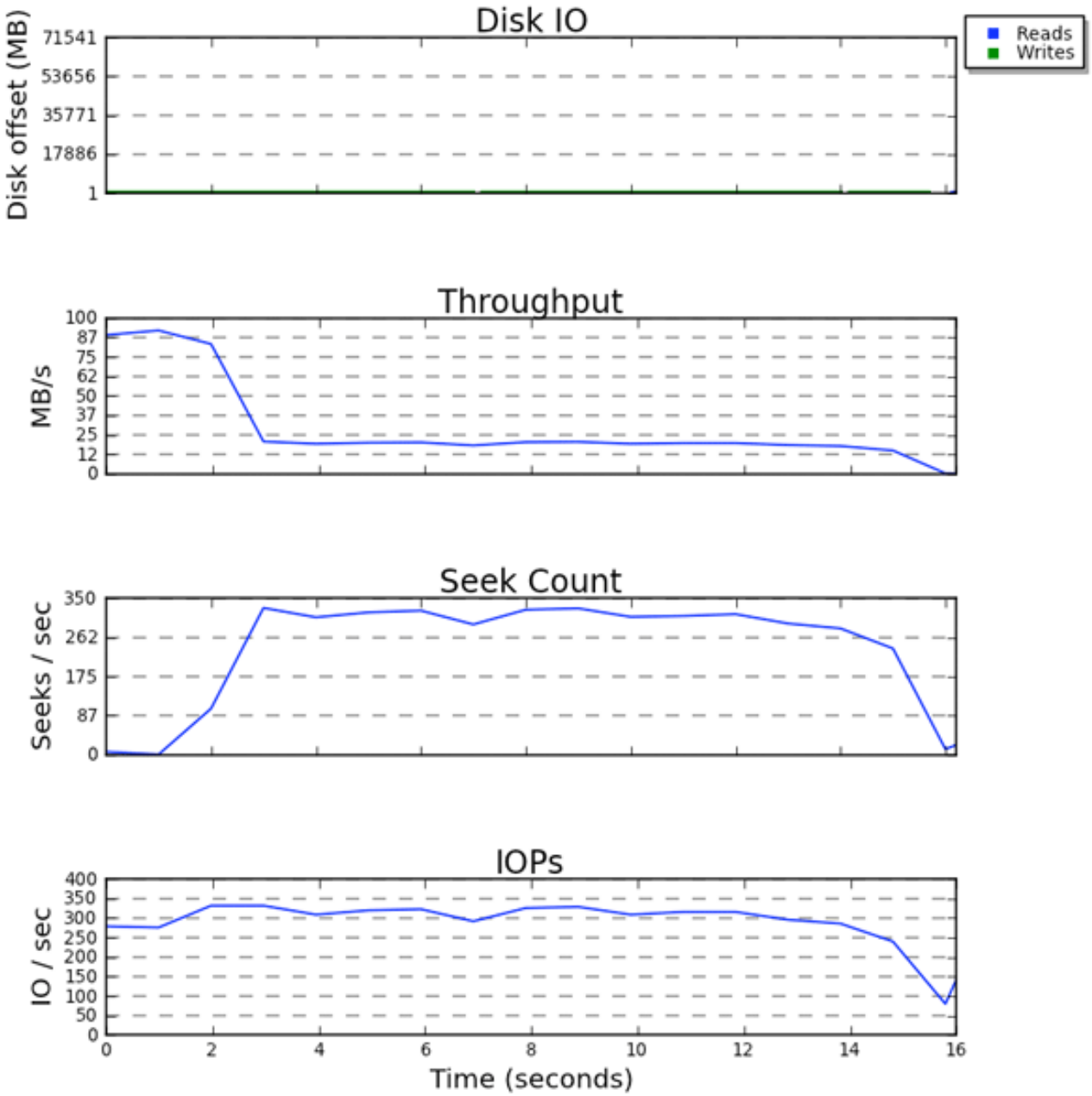
  新排队的块不能作为与现有请求合并的人选，因此会分配一个新的块层请求。
- M – 将 I/O is 与现有请求合并。
- I – 在设备队列中插入一个请求。
- D – 已向设备发出一个请求。
- C – 驱动程序已完成请求。
- P – 已插上块设备队列以便允许整合请求。
- U – 已撤销设备队列，允许向该设备发出整合的请求。

btt 将消耗在每个区域以及在各个区域间转换的时间分段，比如：

- Q2Q – 将请求发送到块层的时间
- Q2G – 从将块 I/O 排队到为其分配一个请求之间所需要的时间
- G2I – 从为其分配一个请求到将其插入设备队列之间所需时间
- Q2M – 将块 I/O 排队到将其与现有请求合并之间所需时间
- I2D – 从将请求插入设备队列到实际向该设备发出请求之间所需时间
- M2D – 从将块 I/O 与现有请求合并到向该设备发出该请求之间所需时间
- D2C – 该设备请求的服务时间
- Q2C – 为一个请求消耗在块层中的时间总量

您可以从上述表格中推断处大量有关负载的信息。例如：如果 Q2Q 比 Q2C 大很多，则意味着程序没有以快速连续法式发出请求。因此您的性能问题可能与 I/O 子系统无关。如果 D2C 很高，那么该设备服务请求的时间就很长。这可能表示该设备只是超载了（可能是由共享资源造成的），或者是因为发送给给设备的负载未经优化。如果 Q2G 很高则意味着队列中同时有大量请求。这可能代表该存储无法承担 I/O 负载。

最后，**seekwatcher** 消耗 **blktrace** 二进制数据并生成一组绘图，其中包括逻辑块地址（LBA），流量，每秒查找次数以及每秒的 I/O 量（IOPS）。



Avg Seeks/s	Avg MB/s	Avg IO/s	Run time (s)
252.57	31.72	305.99	16.21

图 6.2. seekwatcher 输出结果示例

所有绘图都使用时间作为 X 轴。LBA 图用不同的颜色显示读操作和写操作。关注吞吐量与每秒查询次数图很有意思。对查询敏感的存储这两个图形是反的。如果您没有从设备中获得预期的流量，但已达到 IOPS 极限，则需要查看 IOPS 图。

6.4. 配置

首先要决定使用那个 I/O 调度程序。本小节提供主要调度程序概述以帮助您确定最适合您的负载的那一款。

6.4.1. 完全公平调度（CFQ）

CFQ 尝试根据启动 I/O 的进程决定公平的 I/O 调度。可提供三个不同的调度等级：实时（RT），最佳效果（BE）和闲置。可使用 **ionice** 命令手动分配调度等级，或者使用 **ioprio_set** 系统调用编程分配。默认情况下将进程设定为最佳效果调度等级。在实时调度等级和最佳效果调度等级中有分为八个 I/O 优先级，其中 0 代表最高优先权，7 代表最低优先权。采用实时调度等级的进程比采用最佳效果和闲置等级的进程会被更频繁地调度，因此所有调度的实时 I/O 都要在最佳效果或者闲置 I/O 前执行。这意味着实时优先权 I/O 可耗尽最佳效果和闲置等级。最佳效果调度是默认调度等级，且给等级中的默认优先权为 4。停滞调度等级中的进程只有在系统中没有其他等待处理的 I/O 时才会执行。因此请记住只有在进程 I/O 完全不需要向前进行时方可将进程调度等级设定为闲置。

CFQ 通过为每个执行 I/O 的进程分配时间片段提供公平。在其时间片段中，进程每次最多可有八个请求（默认）。调度程序会尝试根据历史数据估计某个程序是否会在近期发出更多 I/O，然后 CFQ 会闲置，等待那个 I/O，即使有其他进程正在等待发出 I/O。

由于 CFQ 执行的闲置操作通常并不适合哪些不会受大量查询惩罚（seek penalty）影响的硬件，比如快速外置存储阵列或者固态硬盘。如果要求在此类存储中使用 CFQ（例如：如果您还喜欢使用 cgroup 加权 I/O 调度程序），则需要调节一些设置以改进 CFQ 性能。请在 **/sys/block/*device*/queue/iosched/** 中同一名称的文件中设定以下参数：

```
slice_idle = 0
quantum = 64
group_idle = 1
```

将 ***group_idle*** 设定为 1 后，可能会产生 I/O 停止（而由于闲置后端存储并不繁忙）。但这些停止的情况并不比系统队列中的闲置情况出现得频繁。

CFQ 是非工作守恒（non-working-conserving）I/O 调度程序，就是说即使在有等待处理的请求时也可以闲置（如前所述）。非工作守恒调度程序栈可在 I/O 路径中产生很大延迟。此类栈的示例为在基于主机的硬件 RAID 控制器顶层使用 CFQ。该 RAID 可部署其自身的非工作守恒调度程序，因此可在该栈的两级中造成延迟。非工作负载调度程序可在有尽可能多的数据供其做决定时操作最佳。在使用该调度算法的栈中，最底层的此奥读程序只能看到上层调度程序发送的数据。因此底层看到的 I/O 模式并不完全代表实际负载。

可调参数

back_seek_max

反向查询通常对性能有负面影响，因为它比正向查询标头重置时间要长很多。但如果负载较小，则 CFQ 仍执行此查询。这个可调参数以 KB 为单位控制 I/O 调度程序允许反向查询的间距。默认为 **16** KB。

back_seek_penalty

由于反向查询的效率低，每项反向查询都有惩罚与之关联。惩罚是一个乘数。例如：视磁头位置在 1024KB。假设在队列中有两个请求，一个在 1008KB，一个在 1040KB。这两个请求到当前磁头位置登距。但如果应用反向查询惩罚（默认：2），则磁盘中距离较远的请求现在与较近的请求的距离缩短了一倍。因此磁头将向前移动。

fifo_expire_async

这个可调参数控制异步（缓存写入）请求等待的时间长度。过期后（以毫秒计）会将无法满足的异步请求移动到调度表中。默认为 **250** 毫秒。

fifo_expire_sync

这个参数与 fifo_expire_async 相同，用于同步请求（读取和 O_DIRECT 写入）。默认为 **125** 毫秒。

group_idle

设定后，CFQ 会在最后一个进程 cgroup 中发出 I/O 后闲置。当使用加权 I/O cgroup 并将 ***slice_idle*** 设定为 **0** 后将其设定为 **1**（通常用于快速存储）。

group_isolation

如果启用组隔离（设定为 **1**），它可以吞吐量为代价提供组群间更强大的隔离。一般来说，如果禁用组隔离，则只为连续负载提供公平机制。启用组隔离则会为连续以及随机负载提供公平机制。默认值为 **0**（禁用）。详情请参考 **Documentation/cgroups/blkio-controller.txt**。

low_latency

启用低延迟后（设定为 **1**），CFQ 会尝试为设备中每个发出 I/O 的进程提供最长 300 ms 的等待时间。禁用低延迟（设定为 **0**）可忽略目标延迟，这样就可允许系统中的每个进程获得全部时间片段。默认启用低延迟。

quantum

quantum 参数控制 CFQ 每次向该存储发出的 I/O 数，主要是限制设备队列深度。默认情况下将其设定为 **8**。该存储可能支持更深的队列深度，但增加 ***quantum*** 还将对延迟产生负面影响，特别是有大量连续写操作的时候。

slice_async

这个可调参数控制分配给每个发出异步（缓存写入）I/O 的进程的时间片段。默认将其设定为 **40** 毫秒。

slice_idle

这个参数指定 CFQ 在等待进一步请求时应闲置的时间。红帽企业版 Linux 6.1 以及更早版本中的默认值为 **8** 毫秒。在红帽企业版 Linux 6.2 以及之后的版本中默认值为 **0**。这个 0 值可通过删除队列以及服务树层中的所有闲置提高外置 RAID 存储流量。但 0 值可降低内置非 RAID 存储的流量，因为它会增加查询总量。对于非 RAID 存储建议您将 ***slice_idle*** 值设定在 0 以上。

slice_sync

这个可调参数专门用于发出同步（读取或者直接写入）I/O 进程的时间片段。默认值为 **100** 毫秒。

6.4.2. 最后期限 I/O 调度程序

最后期限 I/O 调度程序尝试为请求提供保证的延迟。请注意只有当请求进入 I/O 调度程序后方开始计算延迟（这个区别非常重要，因为可能会让程序进入睡眠等待模式以便释放请求描述符）。默认情况下读取比写入的优先权高，因为程序更容易因读取 I/O 而被阻断。

最后期限调度以批形式分派 I/O。一批是一些列连续的读或者写 I/O，按 LBA 顺序递增（单向递增）。处理完每批进程后，I/O 调度程序会检查是否有写请求已等待太久，然后决定是否开始新一批读或者写操作。只在开始新一批时检查过期请求的请求 FIFO 列表。因此，如果选择批写入，且同时有过期的读取请求，那么只有在批写入完成后方可执行读取请求。

可调参数

fifo_batch

这样可以决定单一批次中发出的读取或者写入数。默认为 **16**。设为更高的数值可获得更好的流量，但也会增加延迟。

front_merges

如果您找到负载永远不会生成前合并，则您可以将这个可调参数设定为 **0**。除非您已了解这个检查的代价，建议将其设定为默认值，即 **1**。

read_expire

这个可调参数可让您已毫秒为单位设定读取操作速度。默认将其设定为 **500** 毫秒（即半秒）。

write_expire

这个可调参数可让您已毫秒为单位设定写入操作速度。默认将其设定为 **5000** 毫秒（即五秒）。

writes_starved

这个可调参数控制处理单一写入批之前可以处理多少读取批。这个值越高，越倾向于读取操作。

6.4.3. Noop

Noop I/O 调度程序采用先入先出（FIFO）调度算法。合并原始块层中的请求，但只是一个最后命中缓存（last-hit cache）。如果系统与 CPU 捆绑，且使用高速存储，这就是可以使用的最佳 I/O 调度程序。

以下是块层中可以使用的可调参数。

/sys/block/sdX/queue 可调参数

add_random

在某些情况下，熵池中用于 **/dev/random** 的 I/O 事件成本是可以测量的。在某些情况下要求将其设定为 0。

max_sectors_kb

默认将发送到磁盘的最大请求设定为 **512** KB。这个可调参数可用来增大或者减小该值。最小值为逻辑块大小；最大值由 ***max_hw_sectors_kb*** 设定。有些 SSD 会在 I/O 大小超过内部删除块大小时性能下降。在此类情况下建议将 ***max_hw_sectors_kb*** 降低到删除块大小。您可以使用类似 **iozone** 或者 **aio-stress** 的 I/O 生成程序对此进行测试，记录大小可从 **512** 字节到 **1** MB 不等。

nomerges

这个可调参数主要用于故障排除。大多数负载都可从请求合并中获益（即使类似 SSD 的告诉存储也是如此）。但在有些情况下要求禁用合并，比如当您要查看存储后端可处理多少 IOPS 而无需禁用预读或者执行随机 I/O 时。

nr_requests

每个请求队列都有可为每个读和写 I/O 分配的请求描述符总数限制。这个数的默认值为 **128**，即在将某个进程转入睡眠模式时可将 128 个读和 128 个写放入队列。转入睡眠模式的进程是下一个要分配请求的进程，不一定是已分配所有可用请求的进程。

如果您一个对延迟敏感的程序，则应考虑在您的请求队列中降低 ***nr_requests*** 值，并将存储中的命令队列深度降低到较低的数值（甚至可以降低为 **1**），这样写回 I/O 就无法分配所有可用请求描述符，并使用写入 I/O 设备队列填满该设备。分配 ***nr_requests*** 后，所有其他尝试执行 I/O 的进程都会转入睡眠模式等待请求可用。这样更为公平，因为这样会以轮循模式分配请求（而不是让一个进程很快消耗完所有资源）。注：只有在使用最后期限或者 noop 调度程序时才会有此问题，因为默认 CFQ 配置可防止出现此类情况。

optimal_io_size

在有些情况下，底层存储会报告最佳 I/O 大小。这在硬件和软件 RAID 中很常见，其中最佳 I/O 大小是条大小。如果报告该值，则程序应该发出以及最佳 I/O 大小相当会长成倍数的大小的 I/O。

read_ahead_kb

操作系统可探测到程序何时从文件或者磁盘中连续读取数据。在这种情况下，它可执行智能预读算法，因此用户可能会要求从磁盘中读取更多数据。因此当用户下一步尝试读取数据块时，它已经在操作系统的页缓存中了。可能的缺点是操作系统可能从磁盘中读取过多数据，这样就会占用页缓存直到高内存压力将其清除。如果有多个进程执行错误预读就会增加这种情况下的内存压力。

对于设备映射器设备，一般应该增大 ***read_ahead_kb*** 值，比如 **8192**。理由是设备映射器设备通常有多个基础设备组成。将其设定为默认的值（**128** KB）然后乘以要映射的设备数是个好的调整起点。

rotational

传统硬盘一般都采用轮换模式（比如转盘）。但 SSD 不是。大多数 SSD 会以适当的方式进行宣传。但如果您遇到设备没有说明有此功能，则可能需要手动将轮换模式设定为 **0**；禁用轮换模式后，I/O 提升程序就不使用要减少查询的逻辑，因为在非轮换介质中会有少量查询操作罚分。

rq_affinity

可在与发出 I/O 不同的 CPU 中处理 I/O。将 ***rq_affinity*** 设定为 **1** 可让内核向发出 I/O 的 CPU 传递完成信息。这样可以改进 CPU 数据缓存效果。

第 7 章 文件系统

阅读本章对支持使用红帽企业版 Linux 的文件系统有一个大致了解，并了解如何优化其性能。

7.1. 为文件系统调整注意事项

在所有文件系统有一些通用的注意事项：文件系统中选择的格式化和挂载选项，程序可使用的提高其在所在系统中性能的动作。

7.1.1. 格式化选项

文件系统块大小

可在执行 **mkfs** 时选择块大小。不同的系统其有效范围各有不同：上限为主机系统的最大页大小，下限取决于所使用的文件系统。默认块大小适用于大多数情况。

如果您希望创建大量小于默认块大小的块，您可以设定较小的块大小以尽量减少磁盘空间浪费。注：但设定较小的块大小可能会限制该文件系统中的最大块，并可以造成额外运行费用，特别是对那些比所选块大小更大的块。

文件系统几何学

如果您的系统使用带状存储，比如 RAID5，您可以通过在执行 **mkfs** 时将数据和元数据与基础存储几何对其提高其性能。对于软件 RAID（LVM 或者 MD）以及有些企业级存储，可查询并自动设置这些信息，但在很多情况下必须由管理员在命令行中使用 **mkfs** 手动设定。

有关创建和维护这些文件系统的信息请参考 *《存储管理指南》*。

外部日志

需要大量使用元数据的负载意味着日志文件系统（比如 ext4 和 XFS）的 log 部分会非常频繁地更新。要尽量减少文件系统查询日志的时间，您可以将日志放在专用存储中。注：如果将日志放在速度比主文件系统慢外部存储中可抵消所有可能的与使用外部存储有关的优势。

警告

确定您的外部日志是可靠的。丢失任何外部日志文件都可能造成文件系统死机。

外部日志在运行 **mkfs** 时创建，并要在挂载时指定日志设备。有关详情请参考 **mke2fs(8)**、**mkfs.xfs(8)** 和 **mount(8)** man page。

7.1.2. 挂载选项

Barriers

写入 barrier 是保证在永久存储中正确写入并排列文件系统元数据的内核机制，即使在存储设备会经常断电的情况也不例外。启用了写入 barrier 的文件系统还可以保证在断电时保存使用 **fsync()** 进行的所有数据传输。红帽企业版 Linux 默认在所有支持此功能的硬件上启用 barrier。

但启用写入 barrier 可显著延缓一些程序的速度，特别是使用很多 **fsync()** 的程序，或者延缓创建和删除大量小文件的速度。对于没有不稳定写入缓存的存储，或者罕见的文件系统不一致的情况以及断电后出现可以承受的数据丢失，可使用 ***nobarrier*** 挂载选项禁用 barrier。有关详情请参考 *《存储管理指南》*。

访问时间（noatime）

以前在读取文件时，对那个文件的访问时间（***atime***）必须在内节点元数据中更新，这样就造成额外的 I/O 写入操作。如果不需要准确的 ***atime*** 元数据，则请使用 ***noatime*** 选项挂载该文件系统以便消除这些元数据更新。但在大多数情况下，鉴于红帽企业版 Linux 6内核的默认相对 atime（或者 ***relatime***）行为，***atime*** 不是一个大的消耗。***relatime*** 行为只在原有 ***atime*** 比修改时间（***mtime***）或者状态更改时间（***ctime***）旧时更新 ***atime***）。

注意

启用 ***noatime*** 选项还可以启用 ***nodiratime*** 行为。但不需要同时设置 ***noatime*** 和 ***nodiratime***。

增加的预读支持

预读可通过预先附加数据并将其载入页面缓存以便提前在内存中而不是磁盘中可用，籍此提高文件访问速度。有些负载，比如那些涉及连续 I/O 大量流操作的负载可得益于高的预读值。

tuned 工具以及使用 LVM 条带功能可提高预读值，但对有些负载还是不够的。另外，红帽企业版 Linux 不总是可以根据它可以探测到的您的文件系统设定恰当的预读值。例如：如果一个强大的存储阵列在红帽企业版 Linux 中只作为单一强大 LUN 出现，则操作系统会将其视为强大的 LUN 阵列，并因此默认不会充分利用该存储可以使用的预读优势。

请使用 **blockdev** 命令查看并编辑预读值。要查看某个块设备的当前预读值，请运行：

```
# blockdev -getra device
```

要修改那个块设备的预读值，请运行以下命令。*N* 代表 512 字节扇区中的数值。

```
# blockdev -setra N device
```

注：使用 **blockdev** 命令选择的值重启后不会保留。我们建议创建一个运行等级 **init.d** 脚本在引导时设定这个值。

7.1.3. 文件系统维护

放弃不使用的块

批丢弃和在线丢弃操作是根据文件系统的功能，可丢弃那些文件系统没有使用的块。这些操作在固态硬盘和精简配置存储中很有帮助。

*批忽略操作*由用户明确使用 **fstrim**命令运行。这个命令忽略文件系统中所有与该用户标准匹配到未使用块。在还没企业版 Linux 6.2 以及之后 OS 到 XFS 和 ext4 文件系统中支持这两种操作类型，条件是文件系统到基础块设备支持物理忽略操作。只要 **/sys/block/*device*/queue/discard_max_bytes** 不为零就支持物理忽略操作。

*在线忽略操作*是在挂载时使用 **-o discard** 选项指定（可以是在 **/etc/fstab** 中或者使用 **mount** 命令），并实时运行而无需任何用户互动。在线忽略操作只忽略那些从已使用转换到可用状态的块。红帽企业版 Linux 6.2 以及之后到版本中的 ext4 文件系统以及还没企业版 Linux 6.4 以及之后版本中的 XFS 文件系统支持在线忽略操作。

红帽建议使用批忽略操作除非系统负载不可使用此类批忽略，或者需要使用在线忽略操作保持其性能。

7.1.4. 应用程序注意事项

预分配

ext4、XFS 和 GFS2 文件系统支持使用 **fallocate(2)** glibc 调用有效预分配空间。在由于写入模式造成到大量碎片的文件中可导致读取性能极差。预写入可将磁盘空间标记为已分配给某个文件而无需在那个空间中写入任何数据。最将实际数据写入预写入到块中钱，读取操作将返回 0。

7.2. 文件系统性能侧写

tuned-adm 工具可让用户轻松地在已设计成为具体使用案例提高性的大量侧写间切换。特别用来提高存储性能的侧写为：

latency-performance

用于典型延迟性能调整的服务器侧写。它可禁用 **tuned** 和 **ktune** 节能机制。**cpuspeed** 模块改为 **performance**。每个设备的 I/O 提升程序改为 **deadline**。***cpu_dma_latency*** 参数使用数值 **0**（最小延迟）注册管理电源服务质量以便尽可能减小延迟。

throughput-performance

用于典型吞吐性能调整的服务器侧写。如果系统没有企业级存储则建议使用这个侧写。它与 **latency-performance** 相同，只是：

- 将 ***kernel.sched_min_granularity_ns***（调度程序最小优先占用时间间隔）设定为 **10** 毫秒，
- 将***kernel.sched_wakeup_granularity_ns***（调度程序唤醒间隔时间）设定为 **15** 毫秒。
- 将 ***vm.dirty_ratio***（虚拟机脏数据比例）设定为 40%，并
- 启用够名超大页面。

enterprise-storage

建议最企业级服务器配置中使用这个侧写，其中包括电池备份控制程序缓存保护以及管理磁盘缓存。它与 **吞吐量性能**类似，只是：

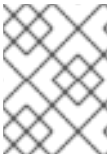
- 将 ***readahead*** 值设定为 **4x**，同时
- 不使用 **barrier=0** 重新挂载的 root/boot 文件系统。

有关 **tuned-adm** 的详情请查看其 man page（**man tuned-adm**），或者*《电源管理指南》*，网址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

7.3. 文件系统

7.3.1. Ext4 文件系统

ext4 文件系统是红帽企业版 Linux 5 中默认的 ext3 文件系统的扩展。现在最红帽企业版 Linux 6 中默认黑色用 Ext4，同时支持到最大文件系统为 16TB，单一文件最大值为 16TB。它还去除了 ext3 中最多只能有 32000 个子系统的限制。



注意

对于超过 16TB 到文件系统，我们建议您使用弹性高容量文件系统，比如 XFS。详情请查看第 7.3.2 节 “XFS 文件系统”。

ext4 文件系统默认是大多数负载的最佳系统，但如果性能分析显示文件系统行为影响到性能，则可以使用以下几个调节选项：

内节点表初始化

对于超大文件系统，**mkfs.ext4** 进程要花很长时间初始化文件系统中到所有内节点表。可使用 ***-E lazy_itable_init=1*** 选项延迟这个进程。如果使用这个选项，内核进程将在挂载文件系统后继续初始化该文件它。可使用 **mount** 命令的 ***-o init_itable=n*** 选项控制发生初始化到比例，其中执行这个后台初始化的时间约为 1/n。**n** 的默认值为 **10**。

Auto-fsync 行为

因为在重命名、截取或者重新写入某个现有文件后，有些应用程序不总是可以正确执行 **fsync()**，在重命名和截取操作后，ext4 默认自动同步文件。这个行为与原有到 ext3 文件系统行为大致相同。但 **fsync()** 操作可能会很耗时，因此如果不需要这个自动行为，请在 **mount** 命令后使用 ***-o noauto_da_alloc*** 选项禁用它。这意味着该程序必须明确使用 **fsync()** 以保证数据一致。

日志 I/O 优先权

默认情况下，日志注释 I/O 比普通 I/O 的优先权稍高。这个优先权可使用 **mount** 命令的 ***journal_ioprio=n*** 选项控制。默认值为 **3**。有效值范围为 0-7，其中 0 时最高优先权 I/O。

其他 **mkfs** 和调节选项详情请参考 **mkfs.ext4(8)** 和 **mount(8)** man page，同时 kernel-doc 软件包的 **Documentation/filesystems/ext4.txt** 文件也有它到信息。

7.3.2. XFS 文件系统

XFS 是一个鲁棒、高度弹性单一主机 64 位日志文件系统。它完全基于扩展，因此可支持超大文件和文件系统。XFS 系统可拥有的文件数量只受该文件系统中可用空间的限制。

XFS 支持元数据日志，这样可从崩溃中迅速恢复。XFS 文件系统还可以最挂载和激活时去除并放大碎片。另外，红帽企业版 Linux 6 支持专门用于 XFS 到备份和恢复工具。

XFS 使用基于扩展到分配，并有大量分配方案可用，比如延迟分配和直接预分配。基于扩展到分配可提供更多简洁、有效到方法跟踪文件系统中使用到空间，并通过减少碎片化和元数据使用到空间提高大文件性能。延迟分配可提高将文件写入连续块组到机会，减少碎片化，提高性能。预分配可用于在程序事先知道它需要写入的数据量到情况下完全防止碎片化。

XFS 提供卓越的 I/O 灵活性，方法是使用 b-tree 检索所有用户数据和元数据。检索中所有操作的对象计数增长都继承基础 b-tree 的对数伸缩特性。有些 XFS 调节选项提供 **mkfs** 时的各种 b-tree 宽度，这样就可以改变不同子系统的伸缩特性。

7.3.2.1. XFS 到基本调节

通常默认的 XFS 格式和挂载选项对大多数负载都是最佳选择。红帽建议使用默认值除非具体配置更改可以对文件系统负载有帮助。如果使用软件 RAID，**mkfs.xfs** 命令可自动使用正确到条单位和宽度自行配置以便与硬件对应。如果使用硬件 RAID 就需要手动进行配置。

在大容量（多 TB）文件系统中建议使用 **inode64** 挂载选项，除非是使用 NFS 和传统 32 位 NFS 客户端导出到文件系统需要对该文件系统到访问。

建议在经常修改或者迅速增长的文件系统中使用 **logbsize** 选项。默认值为 **MAX**（32 KB，日志条单位），同时最大值为 256 KB。建议最有大量修改的文件系统中使用 256 KB。

7.3.2.2. XFS 的高级调节

更改 XFS 参数前，您需要理解为什么默认 XFS 参数会造成性能问题。这包括理解您的程序在做什么，以及该文件系统如何应对那些操作。

可观察到的性能问题可以通过一般由文件碎片或者文件系统中到资源限制造成的调整修正或者减少。处理这些问题有不同的方法，但在有些情况下修复问题需要修改程序配置，而不是修改文件系统配置。

如果您以前没有处理过这个进程，建议您咨询您到本地红帽支持工程师。

优化大量文件

XFS 引入文件系统可以拥有的文件数随机限制。通常这个限制会高到根本无法达到的高度。如果您知道默认限制无法满足未来的需要，您可以使用 **mkfs.xfs** 命令增加可使用的内节点文件系统空间的百分比。如果您在创建文件系统后达到文件限制（通常在尝试创建文件或者目录时出现 ENOSPC 错误信息，即使有可用空间），您可以使用 **xfs_growfs** 命令调整该限制。

最单一目录中优化大量文件

文件系统的目录块是固定的，且无法更改，除非最初使用 **mkfs** 格式化。最小目录块时文件系统块大小，默认为 **MAX**（4 KB，文件系统块大小）。通常没有理由减少目录块大小。

因为该目录结构是基于 b-tree，更改块大小影响每个物理 I/O 可检索或者修改的命令信息量。目录越大，在给定块大小的每个操作需要的 I/O 就更多。

但当使用较大的目录块时，相比使用较小目录块到文件系统，同样的修改操作可能要消耗更多的 CPU。就是说相比小目录，大目录块的修改性能较差。当目录达到 I/O 的性能-限制因数，大块目录性能更佳。

默认配置的 4 KB 文件系统块大小和 4 KB 目录块大小是最多有 1-2 百万条目，每个条目名称最 20-40 字节之间到目录到最佳选择。如果您的文件系统需要更多条目，更大的目录块以便有更佳性能，则 16 KB 块大小时有 1-10 百万目录条目文件系统的最佳选择，64 KB 块大小是超过 1 千万目录条目到文件系统的最佳选择。

如果负载使用随机目录查询而不是修改（即目录读取比目录写入更常用或者重要），那么以上增加块大小的幅度就要减少一个数量级。

并行优化

与其他文件系统不同，XFS 可以最非共享对象中同时执行很多种类的分配和取消分配操作。扩展的分配或者取消分配可同时进行，即可在不同的分配组中同时进行。同样，内节点的分配和取消分配也可以同时进行，即同时进行的操作影响不同的分配组。

使用有多个 CPU 的机器以及多线程程序尝试同时执行操作时分配组的数量就变得很重要。如果只有 4 个分配组，那么可持续的、平行元数据操作将只限于那四个 CPU（该系统提供的并发性限制）。对小文件系统，请确保该系统提供的并发性支持分配组的数量。对于大文件系统（10TB 以上），默认格式化选项通常可生成足够多的分配组以避免限制并发性。

应用程序必须意识到限制点以便使用 XFS 文件系统结构中固有的并行性。不可能同时进行修改，因此创建和删除大量文件的应用程序应避免最一个目录中保存所有文件。应将每个创建的目录放在不同的分配组，这样类似哈希文件的计数就可以最多个子目录中提供比使用单一大目录更灵活的存储形式。

使用扩展的属性优化程序

如果内节点中有可用空间，则 XFS 可以直接在内节点中保存小属性。如果该属性符合该内节点的要求，那么可以最不需要额外 I/O 检索独立属性块的情况下检索并修改它。内嵌属性和外部属性之间的性能差异可以简单归结为外部属性的数量级要低。

对于默认的 256 字节内节点，约有 100 字节属性空间可用，具体要看也保存最内节点中的数据扩展指针数。默认内节点大小只在保存少量小属性时有用。

在执行 mkfs 时增加内节点的大小可以增大用来保存内嵌属性的可用空间量。512 字节的内节点约可增加用于保存属性的空间 350 字节，2 KB 的内节点约有 1900 字节的空间可用。

但对于每个独立可以保存到内嵌书香则有一个大小限制：即属性名称和值占用空间最多为 254 字节（即如果属性的名称长度和值长度都在 254 字节以内则为内嵌属性）。超过这个限制则会将属性强制保存为外部属性，即使在该内节点中有足够到空间保存所有属性。

持续元数据修改优化

日志的大小时决定可持续元数据修改等级的主要因素。日志设备是循环使用的，因此最可以覆盖 tail 命令的结果钱，必须将日志中的所有修改写入磁盘的实际位置中。这可能会涉及大量寻求写入所有脏元数据的操作。默认配置会让日志大小与文件系统总体大小成比例，因此在大多数情况下不需要调整日志大小。

小日志设备可导致非常频繁的元数据写回操作，即日志会一直从结尾写入以便释放空间，会经常将如此频繁修改的元数据写入磁盘，导致操作变缓。

增加日志大小会增加从结尾处 push 事件的间隔。这样可以更好地聚合脏元数据，形成更好的元数据写回模式，减少频繁修改的元数据的写回。代价是较大的日志需要更多内存方可跟踪所有内存中突出的修改。

如果您的机器内存有限，大日志就没有好处，因为内存限制可导致元数据写回延长抵消了释放大日志带来的好处。在这些情况下，通常较小的日志比较大的日志性能更好，因为缺少空间的日志元数据写回比内存重新回收形成的写回更有效。

您应该永远都保持将日志与包含该文件系统的设备底层条单位同步。**mkfs** 命令可自动为 MD 或者 DM 设备完成此功能，但如果是硬件 RAID 则需要手动指定。设定这个功能目前可以避免在磁盘写入修改时所有可能的由于未同步 I/O 以及后续读取-修改-写入操作造成的日志 I/O。

通过编辑挂载选项可进一步提高日志操作性能。增大内存嵌入的日志缓存（**logbsize**）的大小可增加写入日志的更改速度。默认日志缓存大小为 **MAX**（32 KB，日志条单元），且最高可达 256 KB。通常该数值越大性能越快。但如果是在 fsync 负载较重的环境中，小日志缓存比使用大条单元的大缓存速度明显快很多。

delaylog 挂载选项也可以改进不变的元数据修改性能，方法是减少日志更改次数。它通过在将其写入日志前整合每个内存更改：频繁修改的元数据是阶段性写入日志，而不是每次修改时都写入。这个选项增加了跟踪脏元数据的内存用量，同时也增加了崩溃发生时可能损失的操作量，但可以将元数据修改速度和延展性提高一个等级。使用这个选项不会在使用 **fsync**, **fdatasync** 或者 **sync** 时减少数据或者元数据完整性，从而保证可以将数据和元数据写入磁盘。

7.4. 集群

集群的存储可为集群中的所有服务器提供一致的文件系统映像，让服务读取和写入单一文件或者共享的文件系统。这样可以通过限制类似在一个文件系统中安装和为程序打补丁的方法简化集群管理。集群范围内的文件系统还不需要程序数据的冗余副本，这样也简化了备份和系统恢复的过程。

红帽高可用性附加组件除提供红帽全局文件系统 2（单行存储附加组件）外还提供集群的存储。

7.4.1. 全局文件系统 2

全局文件系统 2 是自带的文件系统，可直接与 Linux 内核文件系统互动。它可允许多台计算机（节点）同时分享集群中的同一存储设备。GFS2 文件系统一般采用自我调节，但也可以手动调整。本小节列出了尝试手动调节性能时应注意的事项。

红帽企业版 Linux 6.4 引进了 GFS2 中改进文件系统碎片管理的方法。在红帽企业版 Linux 6.3 或者之前的版本中生成文件时，多个进程同时写入多个文件很容易造成碎片化。这个碎片化可让系统运行缓慢，特别是在有超大文件的负载时。而使用红帽企业版 Linux 6.4，同时写入的结果是产生较少的文件碎片，并籍此获得更好的性能。

虽然红帽企业版 Linux 中没有用于 GFS2 的碎片清除工具，您可以使用 **filefrag** 工具，通过文件碎片工具识别它们，将其复制到临时文件中，并重新命名该临时文件以替换原始文件，这样就可以清除碎片。（只要写入是按顺序进行的，这个步骤还可以用于红帽企业版 Linux 6.4 以前的版本。）

因为 GFS2 使用全局锁定机制，可能会需要集群中节点间的通讯，当将您的系统设计成可避免这些节点间文件和目录竞争时即可获得最佳性能。这些可避免竞争的方法为：

- 在可能的情况下使用 **`fallocate`** 预分配文件和目录以便优化分配过程并避免锁定资源页。
- 尽量减小多节点间共享的文件系统区域以便尽量减小跨节点缓存失效并提高性能。例如：如果多个节点挂载同一文件系统，但访问不同的子目录，则您可以通过将一个子目录移动到独立的文件系统中而获得更好的性能。
- 选择可选资源组大小和数量。这要依赖传统文件大小以及系统中的可用空间，并可能在多个节点同时尝试使用同一资源组时产生影响。资源组过多可延缓块分配，尽管已定位分配空间，而资源组过少也可在取消分配时造成锁竞争。通常最好是测试多种配置以便确定您负载的最佳方案。

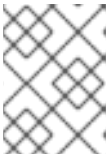
但竞争并不是可影响 GFS2 文件系统性能的唯一问题。其他可提高总体性能的最佳实践为：

- 根据集群节点的预期 I/O 模式和文件系统的性能要求选择存储硬件。
- 在可以减少查询时间的地方使用固态存储。
- 为您的负载创建适当大小的文件系统，并保证该文件系统不会超过容量的 80%。较小的文件系统的备份时间会根据比例缩短，且需要较少时间和内存用于文件系统检查，但如果相对负载过小，则很有可能生成高比例的碎片。
- 为频繁使用元数据的负载设定较大的日志，或者或者记录到日志中的数据正在使用中。虽然这样会使用更多内存，但它可以提高性能，因为在写入前有必要提供更多可用日志空间以便存储数据。
- 请保证 GFS2 节点中的时钟同步以避免联网程序问题。我们建议您使用 NTP（网络时间协议）。
- 除非文件或者目录访问次数对您的程序操作至关重要，请使用 **`noatime`** 和 **`nodiratime`** 挂载选项。

注意

红帽强烈推荐您在 GFS2 中使用 **`noatime`** 选项。

- 如果您需要使用配额，请尝试减少配额同步传送的频率，或者使用模糊配额同步以便防止常规配额文件更新中的性能问题。



注意

模糊配额计算可允许用户和组稍微超过其配额限制。要尽量减少此类问题，GFS2 会在用户或者组接近其配额限制时动态减少同步周期。

有关 GFS2 性能调整各个方面的详情请参考 *《全局文件系统 2 指南》*，网址为 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/。

第 8 章 联网

随着时间的推移红帽企业版 Linux 的网络栈已有了大量自动优化功能。对于大多数负载，自动配置的网络设定可提供优化的性能。

在大多数情况下联网性能问题是由硬件故障或者出错的基础设施造成的。这些原因不在本文档讨论范围。本章所讨论的性能问题及解决方案对优化完全正常工作的系统有帮助。

联网是一个专用子系统，以敏感的连接包含不同部分。这是为什么开源社区以及红帽都致力于使用自动优化网络性能的方式。因此，对于大多数负载，您根本不需要为性能重新配置联网设置。

8.1. 网络性能改进

红帽企业版 Linux 6.1 提供以下网络性能改进：

接收数据包操控（RPS）

RPS 启用单一 NIC **rx** 队列使其接收在几个 CPU 之间发布的 **softirq** 负载。这样可以帮助防止单一 NIC 硬件队列中的网络流量瓶颈。

要启用 RPS，请在 **/sys/class/net/*ethX*/queues/*rx-N*/rps_cpus** 中指定目标 CPU 名称，使用 NIC 的对映设备名称（例如 **eth1, eth2**）替换 *ethX*，使用指定的 NIC 接受队列替换 ***rx-N***。这样可让在该文件中指定的 CPU 处理 ***ethX*** 中 ***rx-N*** 队列中的数据。指定 CPU 时，请注意该队列的 *缓存亲和力* [4]。

接收流程操控

RFS 是 RPS 的延伸，可让管理员配置在程序接收数据并整合至网络栈中时自动填充的哈希表。这可决定哪个程序接受网络数据（根据 source:destination 网络信息）。

使用此信息，网络栈可调度最佳 CPU 区接收每个数据包。要配置 RFS 请使用以下可调参数：

/proc/sys/net/core/rps_sock_flow_entries

这个参数控制内核可以操控的任意指定 CPU 可控制的最多栈/流程数。这是一个系统参数，有限共享。

/sys/class/net/*ethX*/queues/*rx-N*/rps_flow_cnt

这个参数控制可操控某个 NIC（***ethX***）中指定接受队列（***rx-N***）的最大栈/流程数。注：所有 NIC 中这个参数的各个队列值之和应等于或者小于 **/proc/sys/net/core/rps_sock_flow_entries**。

与 RPS 不同，RFS 允许接收队列和程序在处理数据包流程时共享同一 CPU。这样可以在某些情况下改进性能。但这种改进依赖类似缓存阶层、程序负载等因素。

TCP-thin 流的 getsockopt 支持

Thin-stream 是用来描述程序用来发送数据的传输协议的名词，在这种低速率下，协议的重新传输机制并未完全饱和。使用 thin-stream 协议的程序通常使用可靠协议传输，比如 TCP。在大多数情况下此类程序提供对时间敏感的服务（例如股票交易、在线游戏、控制系统）。

对时间敏感的服务，丢失数据包对服务质量是致命的。要放置此类情况出现，已将 **getsockopt** 调用改进为支持两个附加选项：

TCP_THIN_DUPACK

这个布尔值在 thin stream 的一个 duupACK 后启用动态重新传输。

TCP_THIN_LINEAR_TIMEOUTS

这个布尔值为 thin stream 线性超时启用动态起动。

这两个选项都可由该程序特别激活。有关这些选项的详情请参考 **file:///usr/share/doc/kernel-doc-*version*/Documentation/networking/ip-sysctl.txt**。有关 thin-stream 的详情请参考 **file:///usr/share/doc/kernel-doc-*version*/Documentation/networking/tcp-thin.txt**。

传输代理服务器（**TProxy**）支持

内核现在可以处理非本地捆绑的 IPv4 TCP 以及 UDP 插槽以便支持传输代理服务器。要启用此功能，您将需要配置相应的 iptables。您还需要启用并正确配置路由策略。

有关传输代理服务器的详情请参考 **file:///usr/share/doc/kernel-doc-*version*/Documentation/networking/tproxy.txt**。

8.2. 优化的网络设置

性能调节通常采用优先方式进行。通常我们会在运行程序或者部署系统前调整已知变量。如果调整不起作用，则会尝试调整其他变量。此想法的逻辑是*默认情况下*，系统并不是以最佳性能水平运作；因此我们*认为*需要相应对系统进行调整。在有些情况下我们根据计算推断进行调整。

如前所述，网络栈在很大程度上是自我优化的。另外，有效调整网络要求网络栈有深入的理解，而不仅仅值直到网络栈是如何工作，同时还要直到具体系统的网络资源要求。错误的网络性能配置可能会导致性能下降。

例如：*缓存浮点问题*。增加缓存队列深度可导致 TCP 连接的拥塞窗口比允许连接的窗口更大（由深层缓存造成）。但那些连接还有超大 RTT 值，因为帧在队列中等待时间过长，从而导致次佳结果，因为它可能变得根本无法探测到拥塞。

当讨论网络性能时，建议保留默认设置，*除非*具体的性能问题变得很明显。此类问题包括帧损失，流量极大减少等等。即便如此，最佳解决方法通常是经过对问题的细致入微的研究，而不是简单地调整设置（增加缓存/队列长度，减少中断延迟等等）。

要正确诊断网络性能问题请使用以下工具：

netstat

这是一个命令行程序可以输出网络连接、路由表、接口统计、伪连接以及多播成员。它可在 **/proc/net/** 文件系统中查询关于联网子系统的信息。这些文件包括：

- **/proc/net/dev**（设备信息）
- **/proc/net/tcp**（TCP 插槽信息）
- **/proc/net/unix**（Unix 域插槽信息）

有关 **netstat** 及其在 **/proc/net/** 中的参考文件的详情请参考 **netstat** man page: **man netstat**

dropwatch

监控内核丢失的数据包的监视器工具。有关详情请参考 A monitoring utility that monitors packets dropped by the kernel. For more information, refer to the **dropwatch** man page: **man dropwatch**

ip

管理和监控路由、设备、策略路由及通道的工具。有关详情请参考 **ip** man page: **man ip**

ethtool

显示和更改 NIC 设置的工具。有关详情请参考 **ethtool** man page: **man ethtool**

/proc/net/snmp

显示 IP、ICMP、TCP 以及 UDP 根据 **snmp** 代理管理信息所需 ASCII 数据的文件。它还显示实时 UDP-lite 统计数据。

*《SystemTap 初学者指南》*中包含一些示例脚本，您可以用来概括和监控网络性能。您可在 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ 找到本指南。

收集完网络性能问题的相关数据后，您就可以形成一个理论 – ，同时也希望能有一个解决方案。[5]例如：在 **/proc/net/snmp** 中增加 UDP 输入错误表示当网络栈尝试将新帧排入程序插槽时，一个或者多个插槽接受队列已满。

这代表数据包*至少*在一个插槽队列中被瓶颈，就是说插槽队列输送数据包的速度太慢，或者对于该插槽队列该数据包过大。如果是后者，那么可验证任意依赖网络程序的日志查看丢失的数据以便解决这个问题，您应该需要优化或者重新配置受到影响的程序。

插槽接收缓存大小

插槽发送和接收大小都是动态调节的，因此基本不需要手动编辑。如果进一步分析，比如 SystemTap 网络示例中演示的分析，**sk_stream_wait_memory.stp** 认为该插槽队列的排放速度过慢，那么您可以增大该程序插槽队列深度。要做到这一点，请增大插槽接收缓存，方法是配置以下值之一：

rmem_default

控制插槽使用的接收缓存*默认*大小的内核参数。要配置此参数，请运行以下命令：

```
sysctl -w net.core.rmem_default=N
```

使用所需缓存大小以字节为单位替换 ***N***。要确定这个内核参数值请查看 **/proc/sys/net/core/rmem_default**。请记住 **rmem_default** 值不得大于 **rmem_max**）；如果需要请增大 **rmem_max** 值。

SO_RCVBUF

控制插槽接收缓存*最大值*的插槽选项，单位为字节。有关 **SO_RCVBUF** 的详情请参考其 man page：**man 7 socket**。

要配置 **SO_RCVBUF**，请使用 **setsockopt** 工具，您可以使用 **getsockopt** 查询当前 **SO_RCVBUF** 值。有关这两个工具的详情请参考 **setsockopt** man page: **man setsockopt**。

8.3. 数据包接收概述

为更好地分析网络瓶颈和性能问题，您需要直到数据包接收的原理。数据包接收对网络性能调节来说很重要，因为接收路径是经常会丢帧的地方。在接收路径丢帧可能会造成对网络性能的极大负面影响。

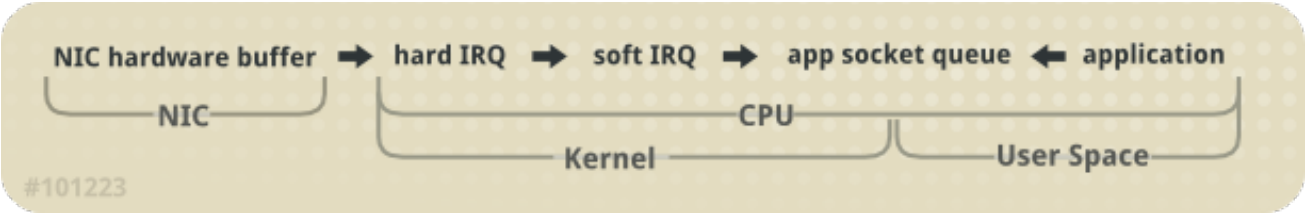


图 8.1. 网络接收路径图表

Linux 内核接收每一帧，并将其送入四步处理过程：

1. *硬件接收*：*网卡*（NIC）接收传送的帧。根据其驱动程序配置，NIC 可将帧传送到内部硬件缓冲内存或者指定的环缓存。

2. *Hard IRQ*：NIC 通过中断 CPU 插入网络帧。这样可让 NIC 驱动程序意识到该中断并调度 *soft IRQ 操作*。

3. *Soft IRQ*：这个阶段采用实际接收进程，并在 **`softirq`** 环境中运行。就是说这个阶段会预先清空所有在指定 CPU 中运行的程序，但仍允许插入 hard IRQ。

   在这个环境中（与 hard IRQ 在同一 CPU 中运行，以便尽量减少锁定消耗），该内核会删除 NIC 硬件缓存以及它通过网络栈的进程中的帧。从那里开始，可将帧转发、忽略或者传递给目标侦听插槽。

   传递给插槽后，该帧就会被附加到拥有该插槽的程序中。这个过程会以互动方式进行直到 NIC 硬件缓存超出帧外，或者直到达到*设备加权*（**`dev_weight`**）。有关设备加权的详情请参考 第 8.4.1 节 “NIC 硬件缓冲”。

4. *程序接收*：程序接受帧并使用标准 POSIX 调用（**`read`**, **`recv`**, **`recvfrom`**）从任意拥有的插槽中退出队列。此时从网络中接收到的数据不再存在于网络栈中。

CPU/缓存亲和性

要维护接收路径的高流量，建议您让 L2 缓存处于*热*状态。如前所述，网络缓冲由作为 IRQ 的显示其存在的同一 CPU 接收。就是说该缓存数据将位于接收 CPU 的 L2 缓存中。

要利用这个功能，请在要接收共享 L2 缓存同一核的 NIC 中的大多数数据的程序中配置进程亲和性。这样可已最大化缓存成功率，并籍此提高性能。

8.4. 解决常见队列/帧丢失问题

到目前为止，帧丢失最常见的原因是*队列超限运转*。该内核设定了队列长度限制，且在有些情况下队列填充的速度超过排出的速度。出现这种情况时间过长，则会开始出现掉帧的情况。

如 图 8.1 “网络接收路径图表” 所示，在接收路径中有两种主要队列：NIC 硬件缓冲和插槽队列。这两种队列都需要进行配置以放置队列超限运转。

8.4.1. NIC 硬件缓冲

NIC 使用帧填充其硬件缓冲；然后该缓冲会被 **`softirq`** 排干，通过中断肯定 NIC。要询问这个队列的状态，请使用以下命令：

```
ethtool -S ethX
```

使用 NIC 的对应设备名称替换 ***ethX***。这样会显示在 ***ethX*** 中已丢失的帧数。丢帧经常是因为该队列超过保存那些帧的缓冲空间所致。

解决这个问题有一些不同方法，即：

输入流量

您可以通过放缓下行输入流量放置队列超限运转。方法是过滤、减少联合多播组数、降低广播流量等等。

队列长度

另外，您也可以增加队列长度。这包括在指定队列中将缓冲数增加到该驱动程序最多可以承担的数量。方法是编辑 ***ethX*** 的 **rx/tx** 环参数，命令为：

```
ethtool --set-ring ethX
```

在前面所说的命令中附加恰当的 **rx/tx** 值。详情请参考 **man ethtool**。

设备加权

您还可以增加排空队列的速度。方法是相应调整 NIC 的*设备加权*。这个属性指的是 **softirq** 上下文必须产生 CPU 并重新调度其本身前 NIC 可以接收的最多帧数。它由 **/proc/sys/net/core/dev_weight** 变量控制。

大多数管理员有选择第三个选项的倾向。但请注意这样做会有一定的后果。在一个迭代中增加可以从 NIC 接收的帧数会造成额外的 CPU 周期，在此期间那个 CPU 无法调度任何应用程序。

8.4.2. 插槽队列

和 NIC 硬件队列一样，插槽队列是由来自 **softirq** 上下文的网络栈填充。然后程序通过调用清空其对应插槽的队列以便进行 **read**、**recvfrom** 等等。

要监控这个队列的状态请使用 **netstat** 程序。**Recv-Q** 列显示队列大小。一般来说对插槽队列中的超限运转的处理与对 NIC 硬件缓冲超限运转的处理相同（例如：第 8.4.1 节 “NIC 硬件缓冲”）：

输入流量

第一个方法是延缓输入流量，方法为填充队列配置速度。具体步骤可以是过滤帧或者抢先丢掉它们。您还可以通过降低 NIC 的设备加权[6]延缓输入流量。

队列深度

您还可以通过增大队列深度避免插槽队列超限运转。方法是增大 **rmem_default** 内核参数或者 **SO_RCVBUF** 插槽选项值。有关详情请参考 第 8.2 节 “优化的网络设置”。

程序调用频率

尽可能优化程序以便更频繁地执行调用。这包括修改或者重新配置网络程序以便执行更频繁的 POSIX 调用（比如 **recv**、**read**）。反过来，这也可以让程序更快地排空队列。

很多管理员更喜欢使用增加队列深度的方法。这是最简单的解决方案，但并不总是能够长期使用。因为联网技术发展迅速，插槽队列将继续以更快的速度填充。随着时间的推移这意味着要相应重新调整队列深度。

最好的解决方法是提高或者将程序配置为更迅速地从内核中排空数据，即使需要让数据在程序空间排队也无妨。这样数据的保存就变得更灵活，因为可以根据需要置换出数据或者缓存到页中。

8.5. 多播注意事项

当有多个程序侦听多播组时，要求将处理多播帧的内核代码设计为为每个独立插槽复制网络数据。这个复制很耗时且要在 **softirq** 上下文中进行。

因此在单一多播组中添加多个侦听程序会直接影响 **softirq** 上下文的执行时间。在多播组中添加侦听程序意味着内核必须为那个组接收的每个帧生成额外的副本。

这对低流量卷且侦听程序数小时影响最小。但当多个插槽侦听一个高流量多播组时，增加的 **softirq** 上下文执行时间导致在网卡和插槽多列中掉帧。增加 **softirq** 运行时会导致降低程序在高负载系统中运行的机会，那样的话多播帧丢失的比例会随着侦听高容量多部组程序是的增加而增大。

如 第 8.4.2 节 “插槽队列” 或者 第 8.4.1 节 “NIC 硬件缓冲” 所属通过优化您的插槽队列和 NIC 硬件缓冲解决丢帧问题。另外，您还可以优化程序的插槽使用。方法是将程序配置为控制单一插槽，并将接收的网络数据迅速传播到其他用户空间进程中。

---

[4] 保证 CPU 和 NIC 之间的缓存亲和力意味着将其配置为共享同一 L2 缓存。有关详情请参考 第 8.3 节 “数据包接收概述”。

[5] 第 8.3 节 “数据包接收概述”中包含数据包行程概述，应该可以帮助您定位并对应网络栈中的瓶颈区域。

[6] 设备加权是由 **/proc/sys/net/core/dev_weight** 控制。有关设备加权以及调整它的方法详情请参考 第 8.4.1 节 “NIC 硬件缓冲”。

附录 A. 修订记录

修订 4.0-22.2.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

修订 4.0-22.2 **Mon July 1 2013** **Wei Lliu**
完成翻译、校对

修订 4.0-22.1 **Thu Apr 18 2013** **Chester Cheng**
与 XML 源 4.0-22 版本同步的翻译文件

修订 4.0-22 **Fri Feb 15 2013** **Laura Bailey**
为红帽企业版 Linux 6.4 发布。

修订 4.0-19 **Wed Jan 16 2013** **Laura Bailey**
为保持一致进行的小修改（BZ#868404）。

修订 4.0-18 **Tue Nov 27 2012** **Laura Bailey**
为红帽企业版 Linux 6.4 Beta 发布。

修订 4.0-17 **Mon Nov 19 2012** **Laura Bailey**
已添加 SME 反馈 re. numad 小节（BZ#868404）。

修订 4.0-16 **Thu Nov 08 2012** **Laura Bailey**
已在 numad 中添加草稿小节（BZ#868404）。

修订 4.0-15 **Wed Oct 17 2012** **Laura Bailey**
采用 SME 反馈限制删除的讨论，并将本小节移到挂载选项下（BZ#852990）。
更新性能策略论述（BZ#858220）。

修订 4.0-13 **Wed Oct 17 2012** **Laura Bailey**
更新性能策略论述（BZ#858220）。

修订 4.0-12 **Tue Oct 16 2012** **Laura Bailey**
改进本书浏览体验（BZ#854082）。
修改 ***file-max*** 定义（BZ#854094）。
修改 ***threads-max*** 定义（BZ#856861）。

修订 4.0-9 **Tue Oct 9 2012** **Laura Bailey**
在文件系统一章添加 FSTRIM 建议（BZ#852990）。
根据用户反馈更新 ***threads-max*** 参数描述（BZ#856861）。
更新有关 GFS2 碎片管理改进备注（BZ#857782）。

修订 4.0-6 **Thu Oct 4 2012** **Laura Bailey**
在 numastat 工具中添加新的一节（BZ#853274）。

修订 4.0-3 **Tue Sep 18 2012** **Laura Bailey**
添加备注，新的 perf 功能（BZ#854082）。
修正 file-max 参数描述（BZ#854094）。

修订 4.0-2 **Mon Sep 10 2012** **Laura Bailey**
添加 BTRFS 一节以及对该文件系统的基本介绍（BZ#852978）。
记录使用 GDB 整合 Valgrind（BZ#853279）。

修订 3.0-15 **Thursday March 22 2012** **Laura Bailey**
添加并更新 tuned-adm 侧写描述（BZ#803552）。

修订 3.0-10 **Friday March 02 2012** **Laura Bailey**

更新 threads-max 和 file-max 参数描述（BZ#752825）。

更新 slice_idle 参数默认值（BZ#785054）。

修订 3.0-8 **Thursday February 02 2012** **Laura Bailey**

重新调整并添加有关任务组和使用 numactl 捆绑 CPU 和内存分配的详情为 第 4.1.2 节 “调节 CPU 性能”（BZ#639784）。

修改内部链接使用（BZ#786099）。

修订 3.0-5 **Tuesday January 17 2012** **Laura Bailey**

第 5.3 节 “使用 Valgrind 简要描述内存使用” 的小修改（BZ#639793）。

修订 3.0-3 **Wednesday January 11 2012** **Laura Bailey**

保证内部和外部超链接的一致性（BZ#752796）。

添加 第 5.3 节 “使用 Valgrind 简要描述内存使用”（BZ#639793）。

添加 第 4.1.2 节 “调节 CPU 性能” 并重新调整 第 4 章 *CPU*（BZ#639784）。

修订 1.0-0 **Friday December 02 2011** **Laura Bailey**

红帽企业版 Linux 6.2 的 GA 发行版本。