

目錄

序	1.1
01.理解Linux操作系统	1.2
1.1.Linux进程管理	1.2.1
1.2.Linux内存体系	1.2.2
1.3.Linux文件系统	1.2.3
1.4.磁盘IO子系统	1.2.4
1.5.网络子系统	1.2.5
1.6.理解Linux性能指标	1.2.6
02.监控和压测工具	1.3
2.1.介绍	1.3.1
2.2.工具功能概述	1.3.2
2.3.监控工具	1.3.3
2.4.压测工具	1.3.4
03.分析性能瓶颈	1.4
3.1.发现瓶颈	1.4.1
3.2.CPU瓶颈	1.4.2
3.3.内存瓶颈	1.4.3
3.4.磁盘瓶颈	1.4.4
3.5.网络瓶颈	1.4.5
04.系统调优	1.5
4.1.优化原则	1.5.1
4.2.安装考量	1.5.2
4.3.调整内核参数	1.5.3
4.4.优化处理器子系统	1.5.4
4.5.优化虚拟内存子系统	1.5.5
4.6.优化磁盘子系统	1.5.6
4.7.优化网络子系统	1.5.7

序

Linux是一个由全世界各地开发者合作打造的开源操作系统。Linux的源代码可以在互联网上自由获取，并且在GPL(GNU General Public License)协议下自由使用。像Red Hat和Novell这样的公司可以定制各式的Linux发行版本提供给用户使用。大部分Linux桌面发行版能够从网络上免费下载，而服务器版可能需要付费购买才能使用。

在过去的这些年里，Linux渗透进了许多公司的数据中心，并且在科学界和商业界都大受欢迎。今天，Linux已经是最通用的操作系统，Linux无处不在，如防火墙、电话和大型机。自然的，Linux操作系统的性能问题成为了各类用户关注一个热点。但是，在不同场景下对操作系统的要求是不一样的，譬如用作天气计算和运行数据库的需求就完全不同，这就要求Linux在所有可能的场景中都表现出卓越性能。幸好，大多数的Linux发行版都包含了通用的调优参数。

IBM把Linux作为一个企业级应用操作系统。大多数的企业应用都已经可以在Linux上运行，例如打印服务，数据库服务，Web服务和邮件服务！

作为企业系统，用户需要监控各项系统性能指标，必要的时候，还需解决影响用户的系统瓶颈。这本IBM的红皮书提供了优化Linux的办法，使用书中的工具，可以监控和分析Linux服务器性能，以及针对特定的服务调整关键参数！这本书中解释了如何分析和优化Linux，使其在各类应用场景中表现出卓越性能。

书中环境的各类参数、基准结果和监测工具都是基于运行在IBM System x系列服务器和IBM System z系列服务器上的Red Hat和Novell SUSE Linux系统，内核版本2.6。不过，书中的知识，应该同样适用于运行在各类硬件平台上的各版本Linux！

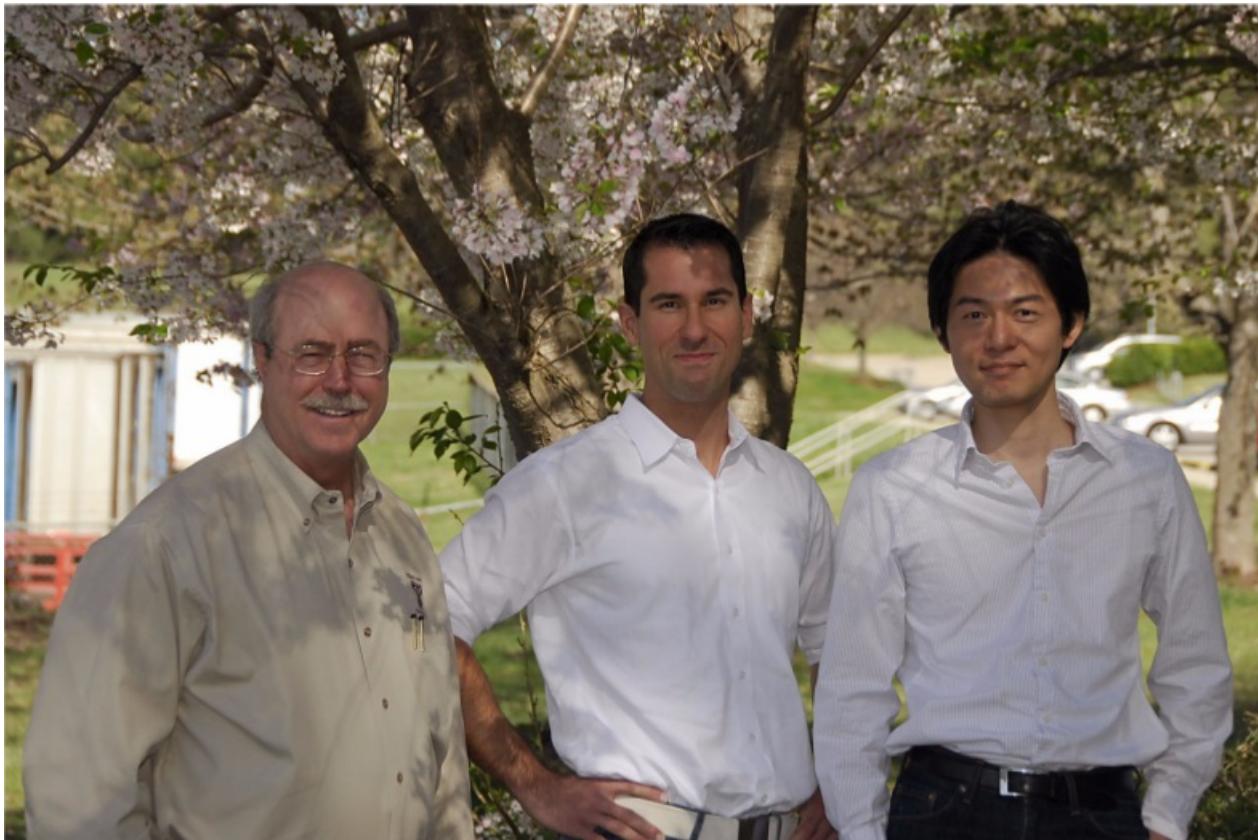
本书结构

为了方便各类Linux或者Linux调优新手迅速开始，我们以如下的方式组织本书：

- 第一章，理解Linux操作系统。
这一章介绍了影响系统性能的问题，以及Linux如何管理系统资源。你将学习到一些重要的系统性能指标！
- 第二章，监控和测试工具
介绍各类衡量和分析Linux性能的工具。
- 第三章，分析性能瓶颈
介绍如何识别出系统中的瓶颈
- 第四章，优化操作系统
基于前几章已经介绍的操作系统工作原理，和性能分析工具，你可以开始探索Linux上的各种性能优化办法！

本书作者团队

本书由来自工作在国际技术支持组织（International Technical Support Organization）的专家组完成。



The team: Byron, Eduardo, Takechika

Eduardo Ciliendo 是IT顾问专家，在IBM瑞士研究系统性能问题。在计算机领域有超过10年的经验。Eddy曾经在苏黎世大学学习商业和计算机科学，还获得了日本学学位（这是个啥子学位？）。Eddy是zChampion的成员，获得过多项IT认证，包括RHCE。作为IBM System z的系统工程师，他负责z/OS和运行在System z上的Linux的容量规划和系统性能优化。Eddy撰写了许多关于系统性能和Linux的出版物。

Takechika Kunimasa 是IBM日本全球服务中心的副IT架构师。他曾在千叶大学学习电气与电子工程学。有10年以上的IT行业经验。做过5年的网络工程师，也做过Linux技术支持。他的专业领域包括各类IBM设备上的Linux高可用系统，网络，基础架构设计。他是思科认证的网络专家（CCNP）和红帽认证工程师（RHCE）。

Byron Braswell 网络专家，在国际技术支持组织工作。他在德克萨斯A&M大学获得了物理学士学位和计算机科学硕士学位。他撰写过关于网络，应用集成中间件，和个人电脑软件的各类文章。在加入ITSO之前，Byron在IBM的networking education development的Learning Services Development里工作。

感谢如下帮助本项目的人：

Margaret Ticknor

Carolyn Briscoe

ITSO, Raleigh 中心

Roy Costa

Michael B Schwartz

Frieder Hamm

ITSO, Poughkeepsie 中心

Christian Ehrhardt

Martin Kammerer

IBM 德国

Erwan Auffret

IBM 法国

成为出版作者

经历2到6周的residency program！帮助编写针对特定产品或者解决方案的IBM红皮书，获得尖端技术的实践经验。你将接触到IBM技术专家，合作伙伴和客户！

你的工作会提高产品的验收率和客户的满意度。作为奖励，你会在IBM开发实验室发展起来社交网络，使你变得更加有生产力和市场化。

想要获得更多实习过程，浏览实习主页，在如下地址申请：

ibm.com/redbooks/residencies.html

欢迎评论

你的意见过很重要！

我们希望这本书能帮到你，把你关于本书或者其它IBM红皮书的意见通过如下方式发给我们：

- 通过在线的联系我们：ibm.com/redbooks
- 发送邮件到：redbooks@us.ibm.com
- 地址：

BM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

=====

- Linux进程管理
 - 什么是进程
 - 进程的生命周期
 - 线程
 - 进程优先级和nice级别
 - 切换上下文
 - 中断处理
 - 进程状态
 - 进程内存段
 - Linux的CPU调度

在本章开始，我们先了解一下Linux是如何处理任务以及与硬件资源进行交互的。性能调优需要在深刻理解硬件资源、操作系统和应用程序的基础上进行。如果性能优化很简单，我们将要学习到的参数就应该被硬编码进固件或者操作系统，否则，我们就没必要学习本书。如下图1-1所示，服务器性能受许多因素影响。

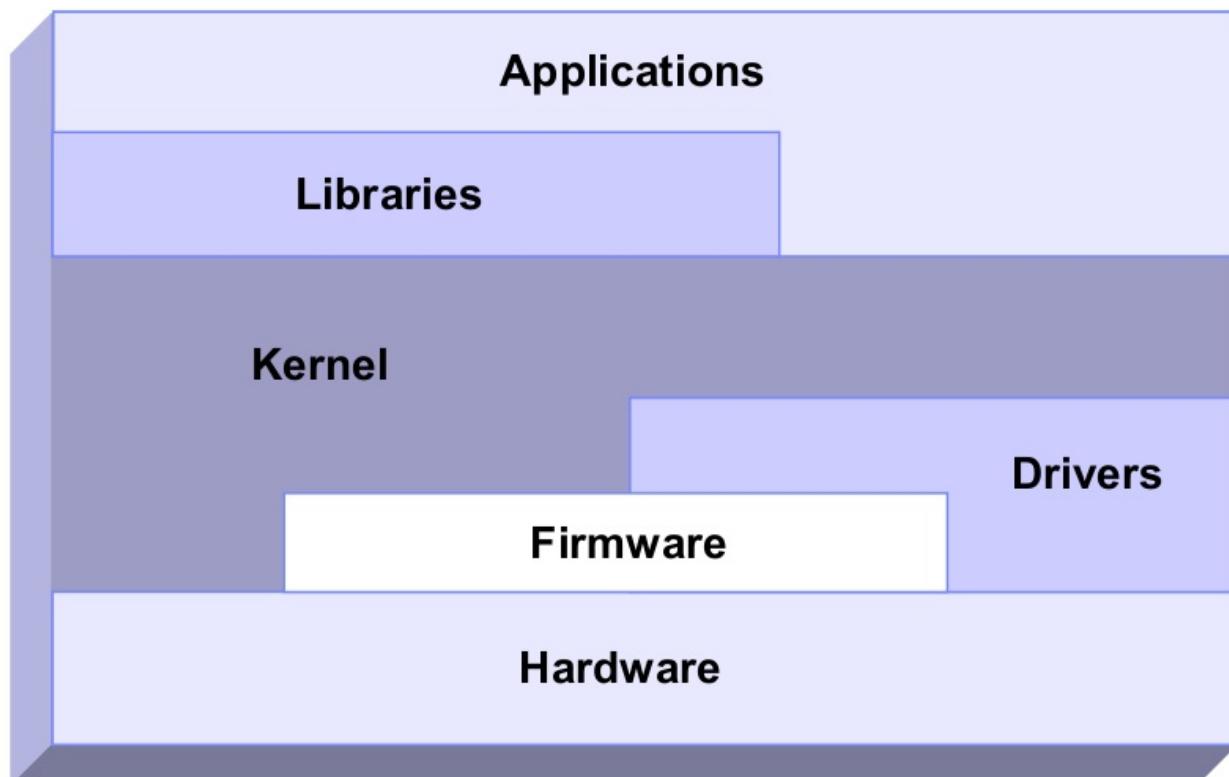


Figure 1-1 Schematic interaction of different performance components

如果一个20000人使用的数据库服务器运行在一块单独的IDE磁盘上，你即使花好几周的时间调试I/O子系统也是白搭。这种情况下，更换一个新的磁盘或者升级应用可能会获得很大的性能提升。当我们讨论具体细节的时候，请牢记上面关于整个系统性能的图片！理解操作系统管理系统资源的方式，能够帮助我们找到各种场景下应该优化的子系统。

以下的部分简短说明了Linux的架构。里面包含了完整的Linux内核分析，你可以同时参考Linux内核文档。

本书专注于讲解Linux操作系统的性能。

下面的章节包含如下内容：

- 1.1 Linux进程管理
- 1.2 Linux内存结构
- 1.3 Linux文件系统
- 1.4 磁盘I/O子系统
- 1.5 网络子系统
- 1.6 衡量Linux性能

1.1 Linux进程管理

进程管理在任何操作系统上都是最重要事情。高效的进程管理能够确保应用高效稳定的运行。

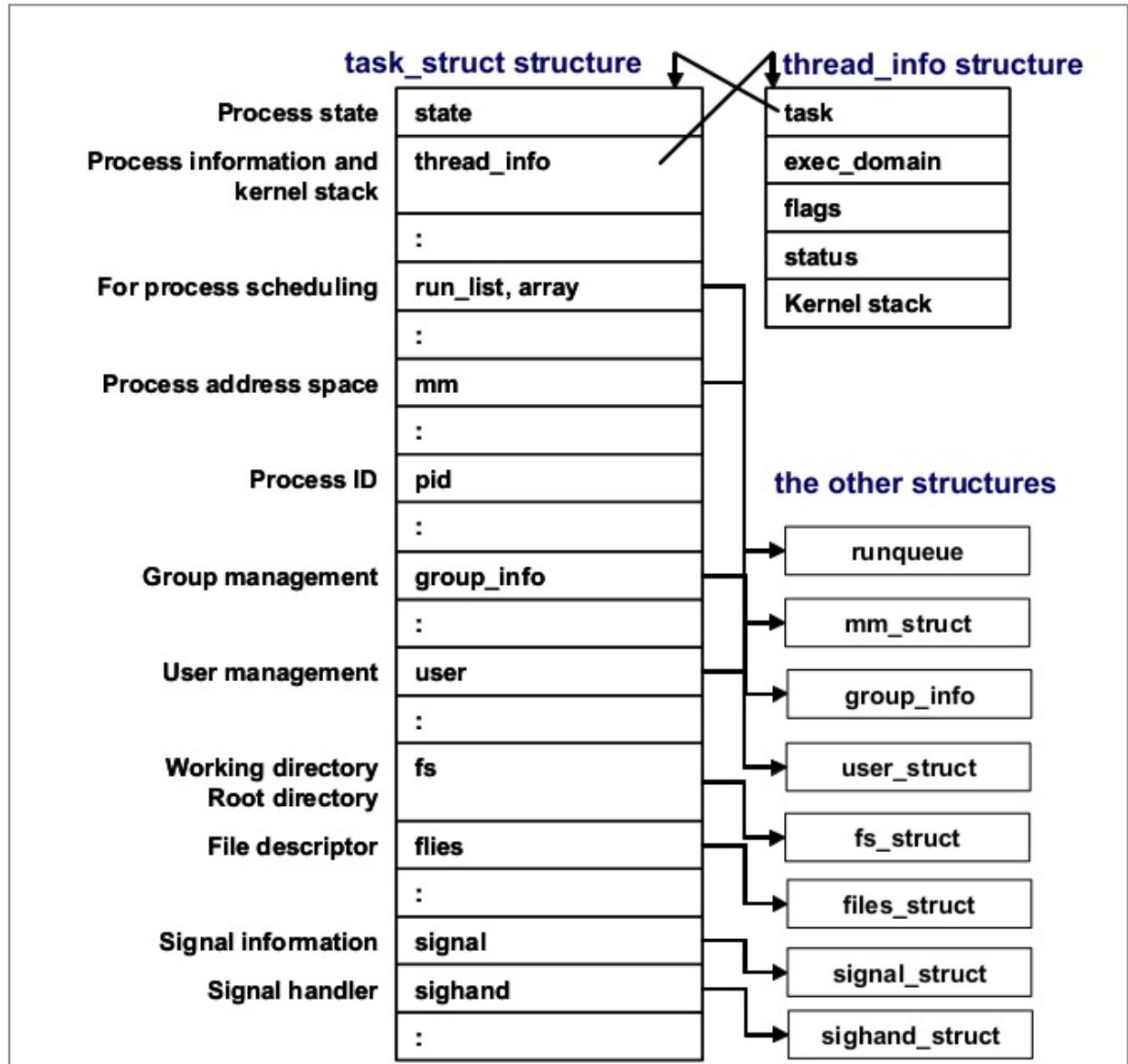
Linux的进程管理方式类似于Unix的进程管理方式，包含进程调度、中断处理、信号、进程优先级、进程切换、进程状态、进程的内存等等。

在本节中，将会讨论Linux进程管理的实现基础，这会帮助你理解Linux内核如何处理影响系统性能的进程。

1.1.1 什么是进程

进程是在处理器中执行的实例，内核调度各类资源来满足进程的需求。

所有运行在Linux操作系统的进程都被task_struct这个结构体管理，task_struct也被称为进程描述符。进程描述符包含一个进程运行所需的所有信息，比如进程的id、进程的属性以及构建进程的资源。如果你知道进程的结构，你就能知道对于进程执行和性能最重要的是什么。下图1-2展示了进程信息相关结构的概览。

Figure 1-2 `task_struct` structure

1.1.2 进程的生命周期

每个进程都有自己生命周期，比如创建、执行、终止和删除。在系统运行过程中，这些阶段反复执行成千上万次。因此，从性能的角度来看，进程的生命周期十分重要。

下图1-3展示了进程的生命周期

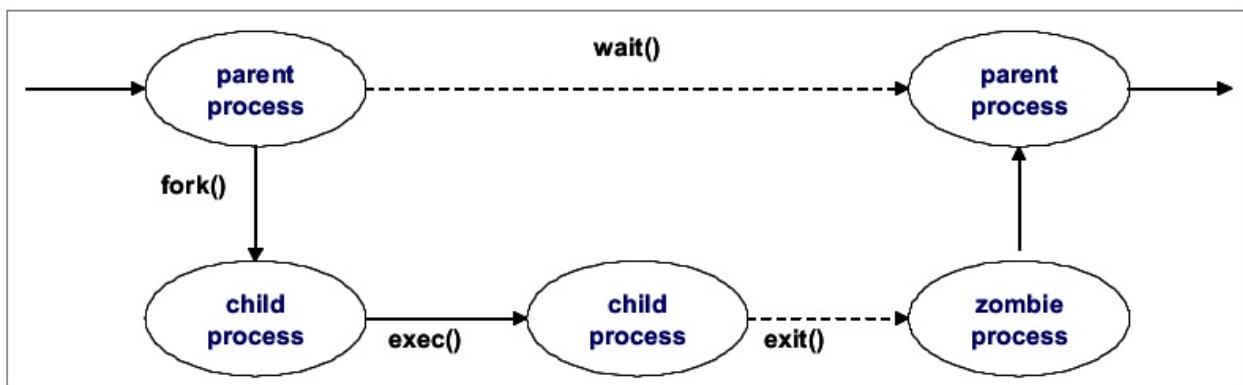


Figure 1-3 Life cycle of typical processes

当一个进程创建一个新的进程，创建进程的进程（父进程）使用名为`fork()`的系统调用。当`fork()`被调用的时候，它会为新创建的进程（子进程）获得一个进程描述符，并且设置新的进程ID。复制父进程的进程描述符给子进程。这时候，不会复制父进程的地址空间，而是父子进程使用同样的地址空间。

`exec()`系统调用把新程序复制到子进程的地址空间。由于共享同样的地址空间，写入新进程的数据会引发页错误的异常。此时，内核给子进程分配新的物理页。

这个延迟的操作叫做Copy On Write。子进程和父进程执行的程序通常不一样，它执行自己的程序。这个操作避免了不必要的开销，因为，复制整个地址空间是很慢且低效率的，还会消耗很多的处理器时间和资源。

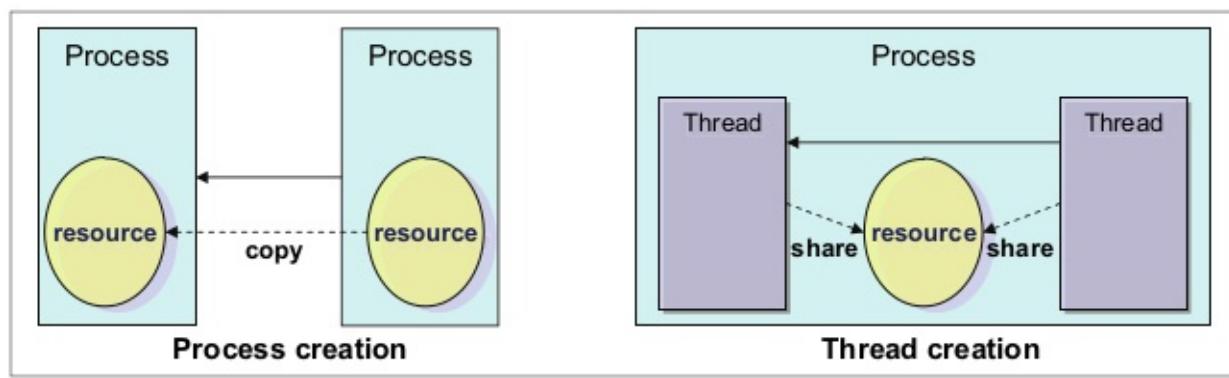
当程序执行完成，子进程使用`exit()`系统调用终止。`exit()`会释放进程的大部分数据结构，并且把这个终止的消息通知给父进程。这时候，子进程被称为zombie process(僵尸进程)。

直到父进程通过`wait()`系统调用知悉子进程终止之前，子进程都不会被完全的清除。一旦父进程知道子进程终止，它会清除子进程的所有数据结构和进程描述符。

1.1.3 线程

线程是单个进程中生成的执行单元。多个线程在同一个进程中并发运行。它们共享内存、地址空间、打开文件等等资源。还能访问同样的应用数据集。线程也被称为轻量级进程（Light Weight Process）。由于线程间共享资源，线程不能同时改变它们共享的资源。互斥、锁、序列化等等都是由用户应用程序来实现。

从性能的角度看，创建线程比创建进程更加低消耗，因为创建线程不需要复制资源。另一方面，从进程和线程在调度上看，他们拥有相似的行为。内核用类似的方法来处理他们。



在当前的Linux实现中，线程由POSIX（Portable Operating System Interface for UNIX，可移植操作系统接口）的兼容库(pthread)提供。Linux支持多线程。如下是已经被广泛使用的。

- LinuxThreads

从Linux Kernel2.0以后，LinuxThreads就是Linux上的默认线程实现方法了。LinuxThreads和POSIX标准有一些不兼容的地方。本地POSIX线程库（Native POSIX Thread Library，NPTL）正在取代LinuxThreads。LinuxThreads将会被未来的Linux企业发行版所抛弃。

- Native POSIX Thread Library (NPTL)

NPTL最开始是有红帽公司开发的，它和POSIX标准更加兼容。由于它在kernel2.6中具备增强的clone()新系统调用、信号处理实现等，它比LinuxThreads具备更好的性能和可扩展性。

NPTL与LinuxThreads有一些不兼容，依赖于LinuxThreads的应用可能没法在用NPTL实现的平台上工作。

- Next Generation POSIX Thread (NGPT)

NGPT是IBM开发的POSIX线程库，它目前处于维护中，没有更多的开发计划。

使用LD_ASSUME_KERNEL环境变量，你可以为你的应用程序选择使用哪一种线程库。

1.1.4 进程优先级和nice级别

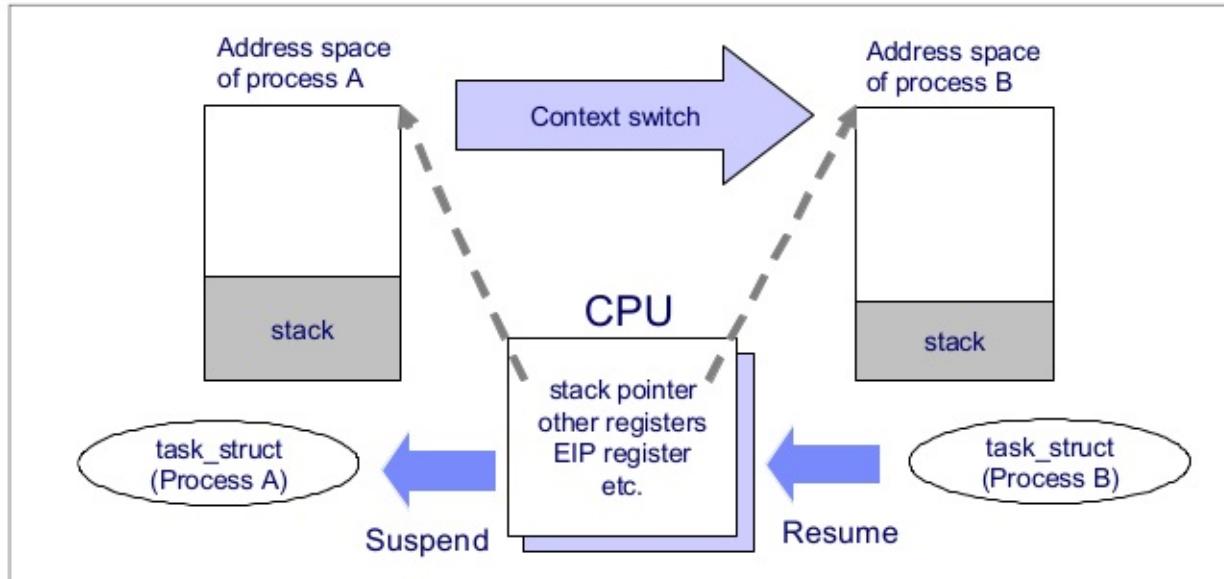
进程优先级由动态优先级和静态优先级决定，它是决定进程在CPU中执行顺序的数字。优先级越高的进程被处理器执行的机会越大。

根据进程的行为，内核使用启发式算法决定开启或关闭动态优先级。可以通过nice级别直接修改进程的静态优先级，拥有越高静态优先级的进程会获得更长的时间片（时间片是进程在处理器中的执行时间）。

Linux支持的nice级别从19（最低优先级）到-20（最高优先级），默认只是0。只有root身份的用户才能把进程的nice级别调整为负数（让其具备较高优先级）。

1.1.5 切换上下文

在进程执行过程中，进程的信息存放在处理器的寄存器和缓存中。这部分执行中进程存放在寄存器中的数据就叫做context，上下文。在切换进程中，正在处理的进程上下文被保存起来，把下一个要执行的进程的上下文恢复到寄存器。上下文通常存储在进程描述符和内核态栈中。进程切换就叫做上下文切换（context switching）。因为处理器每次上下文切换都要为新进程刷新寄存器和缓存，可能引发性能上的问题，所以应该尽量避免太多的上下文切换。下图1-5描述上下文切换是如何工作的。



1.1.6 中断处理

中断处理是最高优先级别的任务之一。中断通常由I/O设备产生，譬如网络接口、键盘、磁盘控制器。中断处理器把键盘输入、网络帧到达这类事件通知给内核，它告诉内核尽快中断进程执行，因为某些设备需要快速的回应。这对系统稳定性是一个挑战。当中断信号到达内核，内核必须切换当前执行中的进程到新的进程，处理中断。这就意味着会发生上下文切换，同时也意味着大量的中断会导致系统性能下降。

在Linux中有两类中断，硬中断是由设备产生的需要做出响应的中断，例如磁盘I/O中断，网卡中断，键盘和鼠标中断。软中断用于任务处理，可以推迟，例如TCP/IP操作、SCSI协议操作。可以在`/proc/interrupts`中看到相关的硬中断信息。

在多处理器环境中，中断由各个处理器自行解决。把中断绑定到单个物理处理器上可以增强系统性能。更多细节，参考4.4.2，CPU affinity for interrupt handling。

1.1.7 进程状态

每个进程都有自己的状态，显示进程中当前发生的事情，进程执行时进程状态发生改变。可能的状态列表如下：

- **TASK_RUNNING**

在这个状态中，进程正在CPU中执行，或者在运行队列（run queue）中等待运行。

- **TASK_STOPPED**

进程由于特定的信号（如SIGINT、SIGSTOP）而挂起就会处于这个状态，等待恢复信号，比如SINCONT。

- **TASK_INTERRUPTIBLE**

在此状态中，进程挂起并且等待一个特定的条件。假如进程处于TASK_INTERRUPTIBLE状态并且收到一个停止信号，进程状态会发生改变，操作会中断。TASK_INTERRUPTIBLE的典型例子是等待键盘中断。

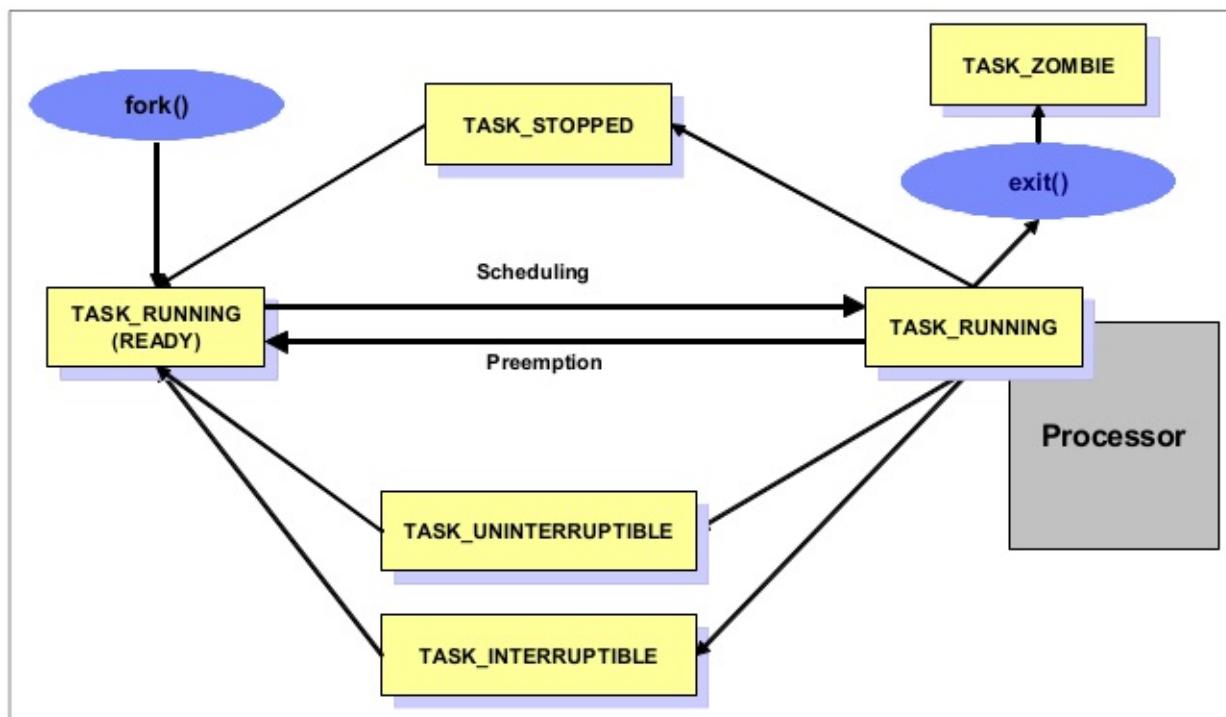
- **TASK_UNINTERRUPTIBLE**

类似于TASK_INTERRUPTIBLE。当进程处于TASK_INTERRUPTIBLE状态可以被中断，发送一个信号给TASK_UNINTERRUPTIBLE却不会有任何反应。

TASK_UNINTERRUPTIBLE最典型的例子是进程等待磁盘I/O操作。

- **TASK_ZOMBIE**

进程在使用exit()系统调用退出以后，父进程应该知道进程终结。在TASK_ZOMBIE状态中，进程在等待父进程收到通知并释放所有的数据结构。



僵尸进程

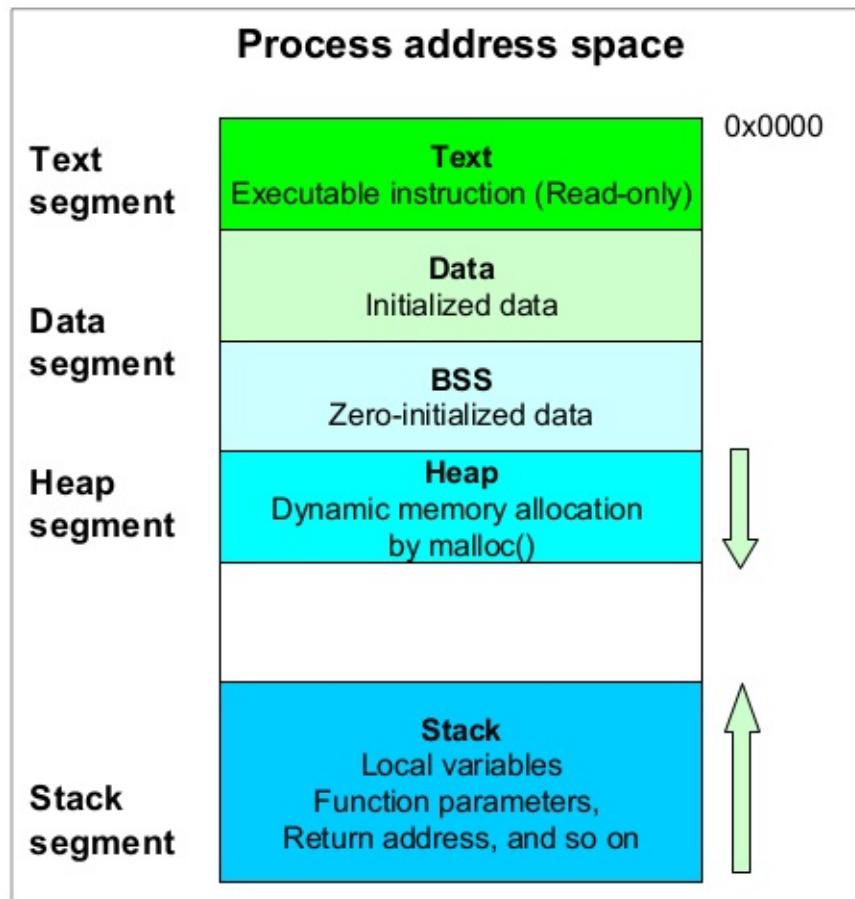
当进程已经收到信号而终止，正常情况下，完全结束之前，它有一些时间来完成所有的任务（例如关闭打开的文件）。在这个很短的正常的时间片里，这个进程是僵尸。

当进程完成了所有的关闭操作，它向父进程报告它即将终结。有时候，僵尸进程不能够结束它自己，这个状态下，它就显示状态为Z (zombie)。

因为它已经死了，所以不可能使用kill命令杀死这种进程。如果无法摆脱僵尸进程，可以杀死僵尸进程的父进程，这样僵尸进程也会消失。然后，如果僵尸进程的父进程是init，你就别这么做了，init是非常重要的进程，你可能要重启才能摆脱僵尸进程了。

1.1.8 进程内存段

进程使用自己的内存区域处理任务，任务种类由场景和进程用途决定。进程有不同的工作特性和不同数据大小要求，进程必须处理各种大小的数据。为满足这一要求，Linux内核给各个进程使用动态内存分配机制。进程内存分配结构如下图1-7。



进程内存区域包含如下段：

- 文本
存储可执行代码的区域
- 数据
数据段由如下三个区域构成
 - Data：存储初始化数据，比如静态变量
 - BSS：存储初始化0数据，数据初始化为0
 - Heap(堆)：根据需要使用malloc()分配动态内存。堆向高地址空间增长。

- 栈

该区域存储局部变量、函数参数和函数的返回地址。栈向低地址空间增长。

用户进程的地址空间分配可以使用pmap命令显示出来。你可以使用ps命令显示总的段大小。参考2.3.10，“pmap”和2.3.10，“ps和pstree”。

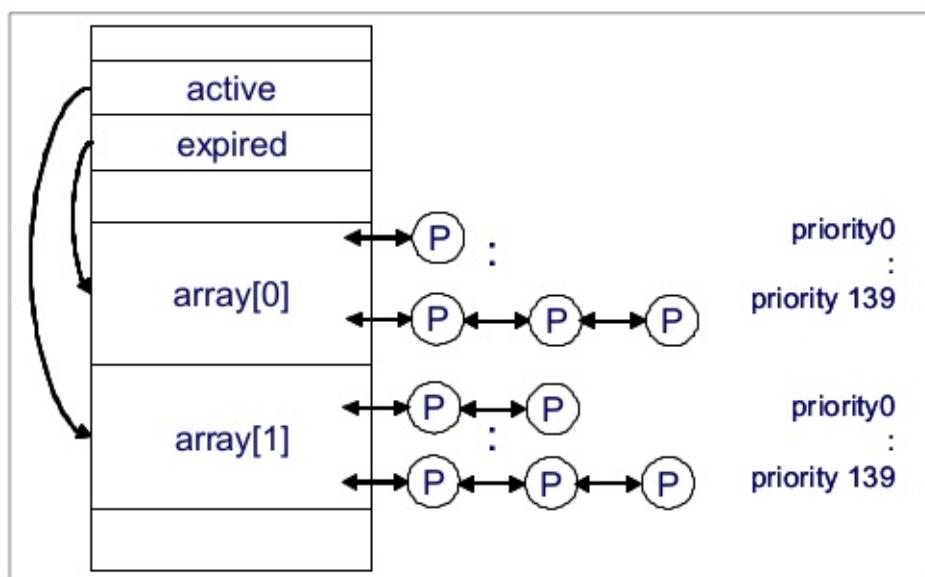
1.1.8 Linux的CPU调度

计算机的最基本功能就是计算，为了实现计算功能，必须要有办法管理计算资源、处理器、计算任务，也就是常说的进程和线程。感谢Ingo Molnar所做出的伟大贡献，Linux内核使用O(1)而不是O(n)来实现CPU调度。O(1)就是静态算法，意味着处理器选择和调用进程开始执行的时间是一个常数，不论多少进程数量的情况下都是如此。

新的调度器十分好用，不用考虑进程数量和处理器数量，强制使用很小的开销。这个算法使用两个进程优先级数组：

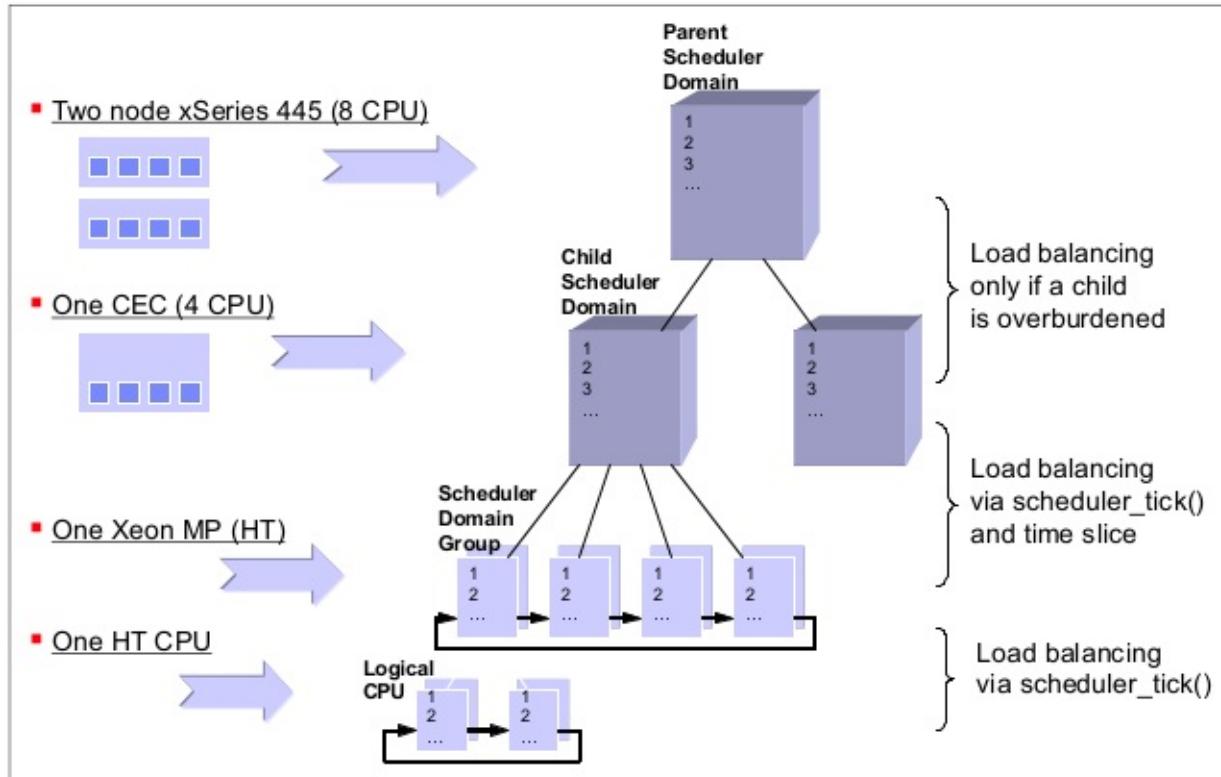
- active
- expired

由于调度器根据进程的优先级和先前阻塞率来位分配时间片，进程的优先级被放在一个active数组中。当时间片到期，它们被重新分配新的时间片，并且放置到expired数组上。当所有active数组上的进程都到期，active和expired数组发生对换，重启算法。对于一般的交互进程（对应实时进程），高优先级的进程通常会比低优先级进程分配更多的时间片，但是并不意味着完全不给低优先级进程机会。这个算法大大提高Linux内核的可扩展性，对于包含各种大量进程和线程以及处理器的工作需求，尤其如此。新的O(1)CPU调度算法是为内核2.6设计的，兼容内核2.4的版本。如下1.8展示和Linux的CPU调度器是如何工作的。



新调度器的另外一个大有有点是支持非均匀内存架构（Non-Uniform Memory Architecture，NUMA）和对称多线程处理器，例如Intel的超线程技术。

支持NUMA保证了正常情况下不会出现负载均衡的情况，除非一个节点负担过重。这个机制保证了在NUMA系统中，比较缓慢的链路负载较小。尽管在一个组中的处理器调度的每一个处理，会被负载均衡，但是调度器的组只会在节点负载过高和要求负载均衡的时候产生。



1.2 Linux内存体系

- Linux内存体系
 - 物理和虚拟内存
 - 虚拟内存管理器

进程执行过程中，Linux内核根据需要给进程分配一块内存区域。进程就把这片区域作为工作区，按要求执行操作。这就像给你分配一张自己的桌子，你可以在桌子上摆放文档，备忘录，开展自己的工作。区别在于，内核以更加动态的方式分配空间。系统上运行的进程经常是成千上万的，但是内存却是有限的。于是，Linux必须高效的处理内存问题。在本节中，将介绍Linux内存架构、地址布局、以及Linux如何高效管理内存空间。

1.2.1 物理和虚拟内存

现实中，我们经常会面临32位还是64位操作系统的选择，对用户来说，它们间最大的差别是能否支持4GB以上的虚拟内存空间。站在性能的角度来理解32位和64位的系统上，Linux映射物理内存到虚拟内存的区别是十分有趣的。如下图所示，可以明显看出内存映射方式在32位和64位系统上的区别，详尽探索物理内存映射到虚拟内存超出了本文的范围，我们重点学习Linux的内存架构的一些知识。

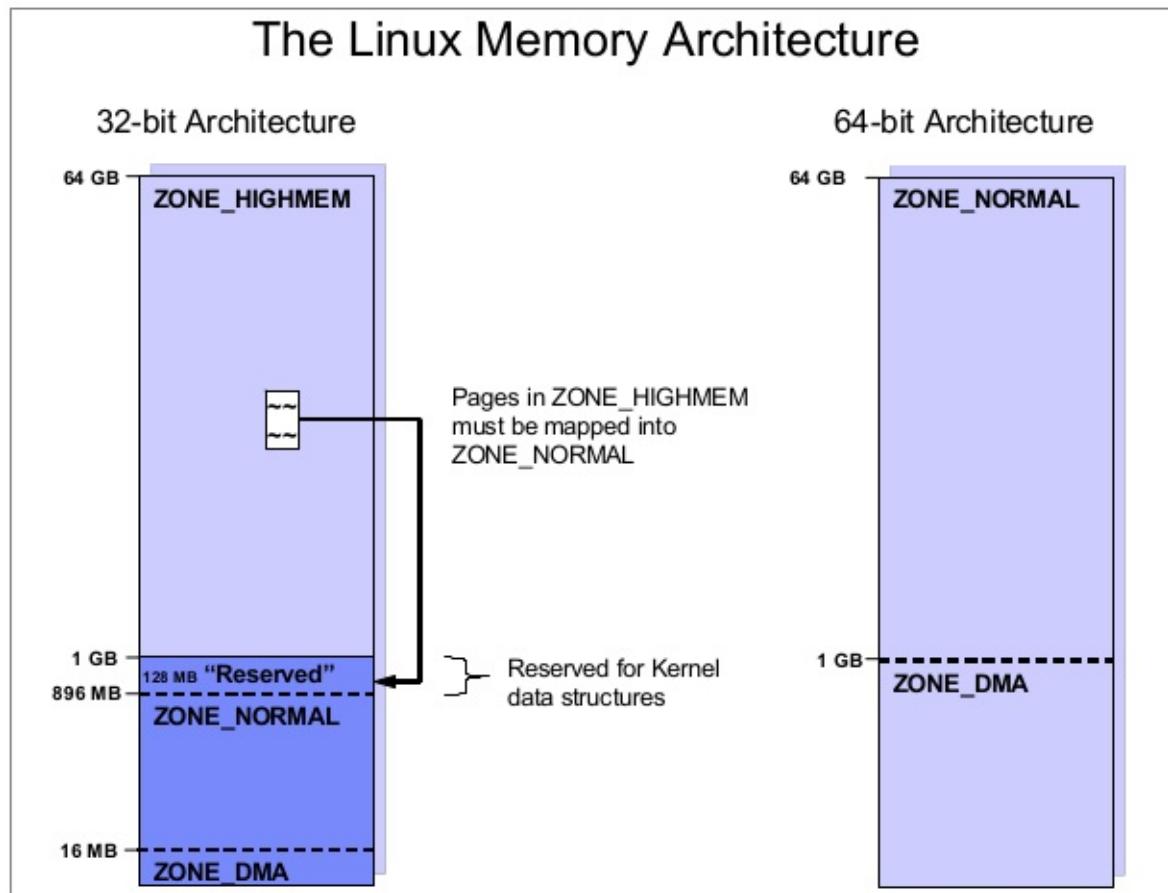


Figure 1-10 Linux kernel memory layout for 32-bit and 64-bit systems

在

32位架构的机器上，Linux内核只能直接映射第一个GB的物理内存（896M，因为还要考虑到保留的空间）。在此上的内存被称作ZONE_NORMAL，这部分空间必须映射到最下面的1GB。这种映射对应用程序是完全透明的，但是分配内存页到ZONE_HIGHMEM会造成一点点性能损耗。

另一方面，在64位系统上，例如在IA-64上面，ZONE_NORMAL一直延伸到64GB或者128GB。如你所见，把内存页从ZONE_HIGHMEM映射到ZONE_NORMAL这种损耗在64位系统上是不存在的。

虚拟内存寻址布局

下图展示了32位和64位架构Linux系统的虚拟寻址布局。

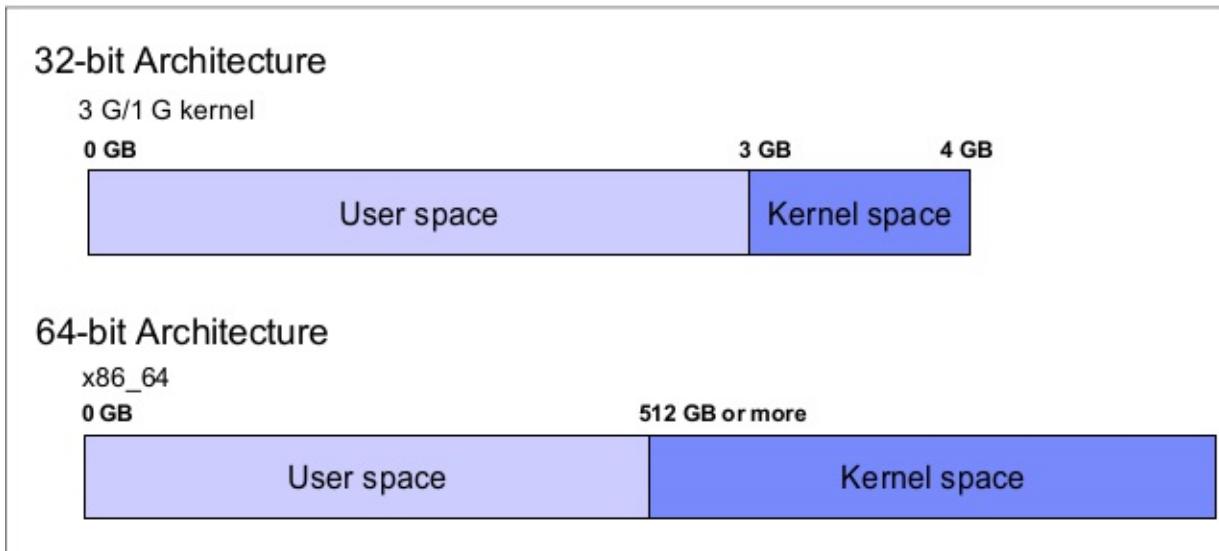


Figure 1-11 Virtual memory addressing layout for 32bit and 64-bit architecture

在32位架构上，单个进程可以利用的最大地址空间是4GB，这是受到了32位虚拟内存映射的限制。在标准的32位环境中，虚拟地址被划分为3GB的用户空间和1GB的内存空间，现实中也存在一些4GB/4GB地址布局。

再说64位架构，因为没有内存限制存在，每个进程能够都有可能使用巨大的地址空间。

1.2.2 虚拟内存管理器

由于操作系统把所有内存都映射成虚拟内存，所以，操作系统的物理内存架构对用户和应用程序通常都是不可见的。如果我们要掌握Linux内存调优的办法，就必须先理解Linux如何处理虚拟内存。如1.2.1所说的那样，应用程序不使用物理内存，而是向Linux内核请求一个特定大小的内存映射，并且收到一个虚拟内存的映射。如下图所示，虚拟内存不必要一定是物理内存的映射，如果某个应用程序使用了一块超大的虚拟内存，这虚拟内存其中某一部分可能是由磁盘上的swap空间映射来的。

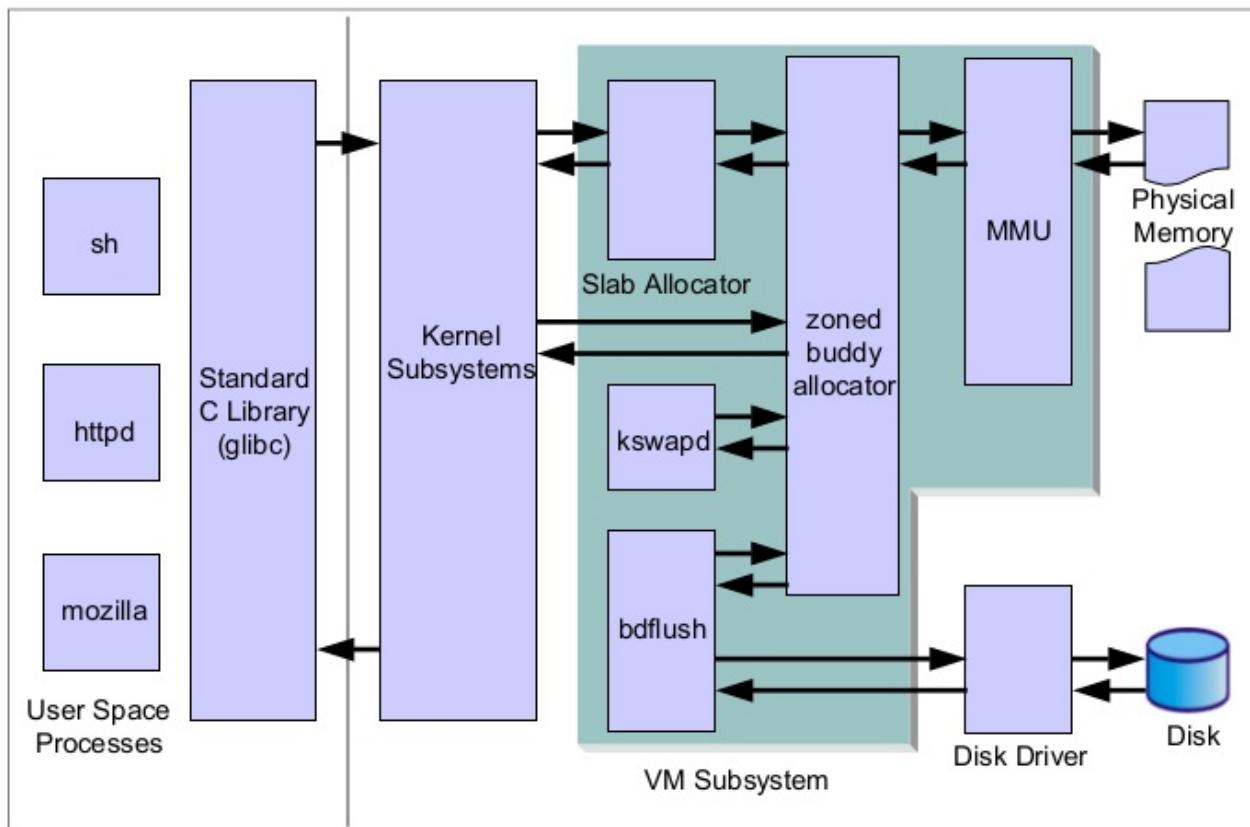


Figure 1-12 The Linux virtual memory manager

从图中可以看出来，应用程序经常不是直接写入磁盘子系统，而是首先写入cache或者buffer，然后，在pdflush空闲的时候、或者某个文件大小超出buffer和cache的时候，由pdflush内核线程把buffer或cache中的数据写入磁盘。参考后面的写入脏buffer部分。

Linux内核管理磁盘缓存的方式，和内核写数据到文件系统的方式有紧密联系。和其它操作系统都只分配特定的部分内存作为磁盘缓存的方式相比，Linux处理内存资源更加高效。虚拟内存管理器默认配置把所有的可用空闲内存空间作为磁盘缓存，所以，经常可以见到Linux系统明明拥有数GB级的内存，却只有20M处于空闲状态。

Linux同样高效利用swap空间，当操作系统开始使用swap空间的时候，并不表示系统出现了内存瓶颈，而是证明了Linux如何有效的使用系统资源。参考页帧回收（page frame reclaiming）。

页帧分配(Page frame allocation)

页是物理内存或虚拟内存中一组连续的线性地址，Linux内核以页为单位处理内存，页的大小通常是4KB。当一个进程请求一定量的页面时，如果有可用的页面，内核会直接把这些页面分配给这个进程，否则，内核会从其它进程或者页缓存中拿来一部分给这个进程用。内核知道有多少页可用，也知道它们的位置。

伙伴系统 (Buddy system)

Linux内核使用名为伙伴系统（Buddy system）的机制维护空闲页，伙伴系统维护空闲页面，并且尝试给发来页面申请的进程分配页面，它还努力保持内存区域是连续的。如果不考虑到零散的小页面可能会导致内存碎片，而且在要分配一个连续的大内存页时将变得很困难，这

就可能导致内存使用效率降低和性能下降。下图说明了伙伴系统如何分配内存页。

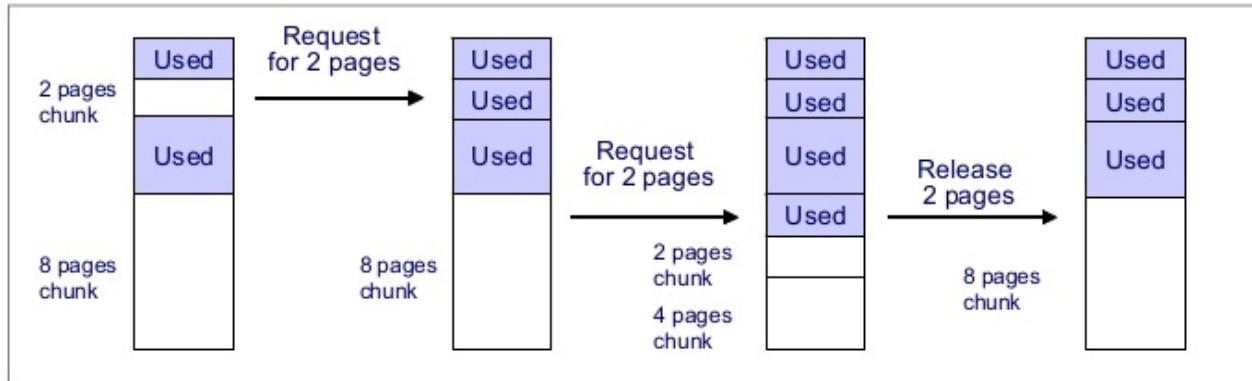


Figure 1-13 Buddy System

如果尝试分配内存页失败，就启动回收机制。参考“内存页回收(Page fram reclaiming)”

可以在/proc/buddyinfo文件看到伙伴系统的信息。详细内容参考“zone中使用的内存(Memory used in a zone)”

页帧回收

如果在进程请求指定数量的内存页时没有可用的内存页，内核就会尝试释放特定的内存页（以前使用过，现在没有使用，并且基于某些原则仍然被标记为活动状态）给新的请求使用。这个过程叫做内存回收。`kswapd`内核线程和`try_to_free_page()`内核函数负责页面回收。

`kswapd`通常在task interruptible状态下休眠，当一个区域中的空闲页低于阈值的时候，它就会被伙伴系统唤醒。它基于最近最少使用原则（LRU，Least Recently Used）在活动页中寻找可回收的页面。最近最少使用的页面被首先释放。它使用活动列表和非活动列表来维护候选页面。`kswapd`扫描活动列表，检查页面的近期使用情况，近期没有使用的页面被放入非活动列表中。使用`vmstat -a`命令可以查看有分别有多少内存被认为是活动和非活动状态。详细内容可以参考“vmstat”一节。

`kswapd`还要遵循另外一个原则。页面主要有两种用途：页面缓存(*page cache*)和进程地址空间(*process address space*)。页面缓存是指映射到磁盘文件的页面；进程地址空间的页面（又叫做匿名内存，因为不是任何文件的映射，也没有名字）使用来做堆栈使用的，参考1.1.8“进程内存段”。在回收内存时，`kswapd`更偏向于回收页面缓存。

Page out和**swap out**：“page out”和“swap out”很容易混淆。“page out”意思是把一些页面（整个地址空间的一部分）交换到swap；"swap out"意味着把所有的地址空间交换到swap。

如果大部分的页面缓存和进程地址空间来自于内存回收，在某些情况下，可能会影响性能。我们可以通过/proc/sys/vm/swappiness文件来控制这个行为，参考4.5.1“设置内核swap和pdflush行为”学习调优细节。

swap

在发生页面回收时，属于进程地址空间的处于非活动列表的候选页面会发生page out。拥有交换空间本身是很正常的事情。在其它操作系统中，swap无非是保证操作系统可以分配超出物理内存大小的空间，但是Linux使用swap的空间的办法更加高效。如图1-12所示，虚拟内存由物理内存和磁盘子系统或者swap分区组成。在Linux中，如果虚拟内存管理器意识到内存页已经分配了，但是已经很久没有使用，它就把内存页移动到swap空间。

像getty这类守护进程随着开机启动，可是却很少使用到，此时，让它腾出宝贵的物理内存，把内存页移动到swap似乎是很有益的，Linux正是这么做的。所以，即使swap空间使用率到了50%也没必要惊慌。因为swap空间不是用来说明内存出现瓶颈，而是体现了Linux的高效性。

Linux文件系统

- Linux文件系统

- 虚拟文件系统
- 日志
- Ext2
- Ext3
- ReiserFS
- 日志文件系统
- XFS

Linux作为开源操作系统，最大的优势是它可以支持各类文件系统。现代的Linux内核能够支持几乎每种文件系统，从基础的FAT到高性能的日志文件系统（JFS）都能可以。然而，由于Ext2、Ext3和ReiserFS是原生的Linux文件系统，被大多数Linux发行版（ReiserFS只受Novell Suse Linux商业支持）支持，所以，我们将重点学习它们，同时也会大概的介绍一下其它特别常用的Linux文件系统。

要了解更多文件系统和磁盘子系统的内容，可以参考4.6“磁盘子系统优化”

1.3.1 虚拟文件系统

虚拟文件系统（VFS）是一个处于用户进程和各类文件系统之间的抽象接口层，VFS提供访问文件系统对象的通用对象模型（例如，i-node、文件对象、页缓存、）和方法，它对用户进程隐藏了各种文件系统的差别。正是因为有VFS，所以用户进程不用关心使用的是那种文件系统，也更不需要知道各个文件系统应该使用哪个系统调用。下图显示了VFS的概念。

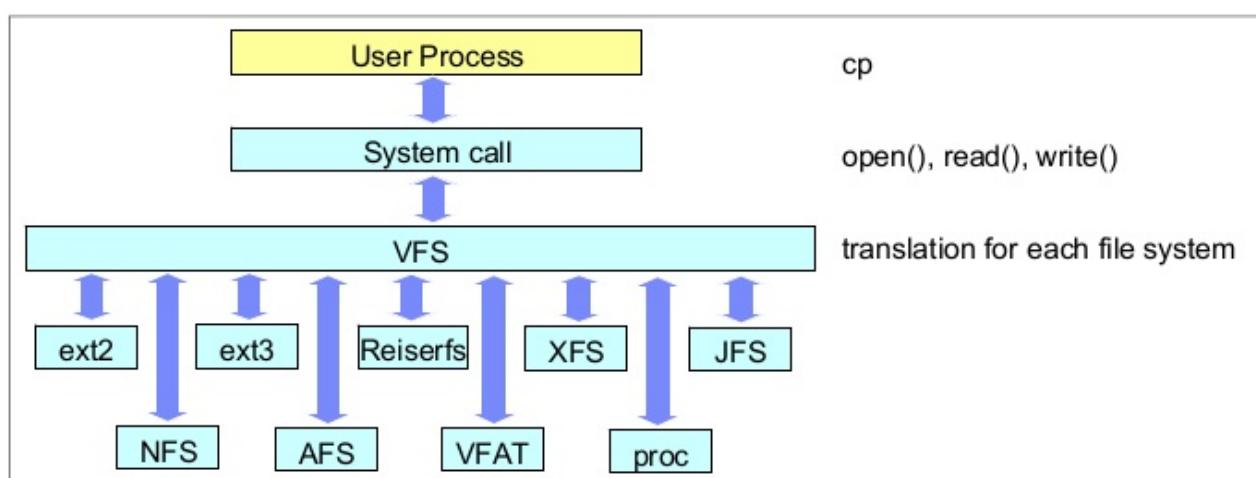


Figure 1-14 VFS concept

日志

在非日志文件系统上，当执行写操作时，内核先修改文件系统的元数据，然后写入实际的用

户数据。这个操作可能会丢失数据的完整性。如果在写入文件系统元数据的时候系统突然崩溃，文件系统的一致性就可能被破坏。`fsck`会在下次启动时检查所有的元数据，并修复文件系统上的不一致，但是如果卷特别大的时候，这个修复过程就会变得很漫长，只能干等着`fsck`工作完之后才能使用这系统。

日志文件系统解决了这个问题，在写入实际的文件系统之前，他先把要修改的数据写入一个叫做日志区域（journal area）的地方。日志区域可以在文件系统上，也可以不在文件系统上。写入日志区域的数据叫做日志记录（journal log）。如果系统支持的话，它内容包括文件系统元数据和真正的文件数据。

因为在写入真正用户数据之前要写记录日志，和非日志文件系统相比会产生性能开销。维护数据高度一致性所牺牲的性能开销大小，取决于在写入用户数据之前要写入多少信息到磁盘上。后面的1.3.4中Ext3一节中会讨论这个问题。

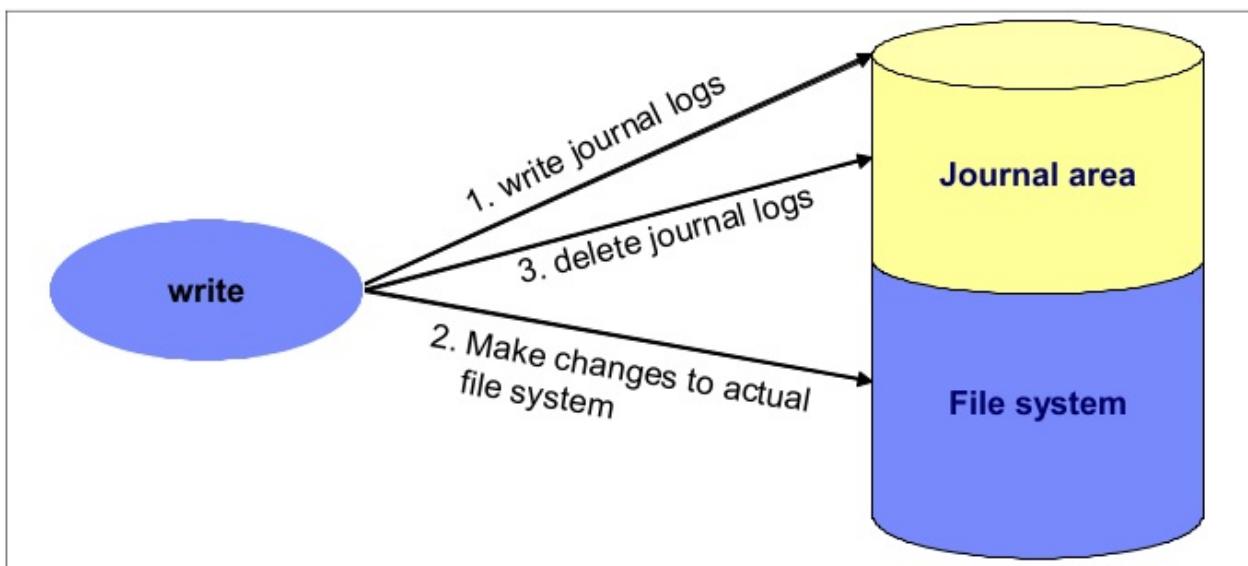


Figure 1-15 Journaling concept

Ext2

Ext2文件系统是Ext3的前身。和大多数现代文件系统不一样，Ext2是一个简单和快速的文件系统，没有日志功能。

下图展示了Ext2的文件系统数据结构，文件系统由一个引导扇区（boot sector）开始，后面跟着块组（block groups）。整个文件系统被分成许多小块组来获得高性能，因为i-node表和保存用户数据的数据块驻留在较前的磁盘，所以节约寻址时间。一个块组包含如下项目：

- + 超级块（Super block） 磁盘上信息存放在那里，超级块的副本保存在每个块组的顶部。
- + 块组描述符（Block group descriptor） 块组的信息存在这里
- + 数据块位图（Data block bitmaps） 空闲块管理
- + i-node位图（i-node bitmaps） 空闲i-node管理
- + i-node表（i-node tables） 存放i-node表。每个文件都有相应的i-node表，存放文件的元数据，譬如文件模式、uid、gid、atime、ctime、mtime、dtime和数据块的指针。
- + 数据块（Data blocks） 存放真正的用户数据。

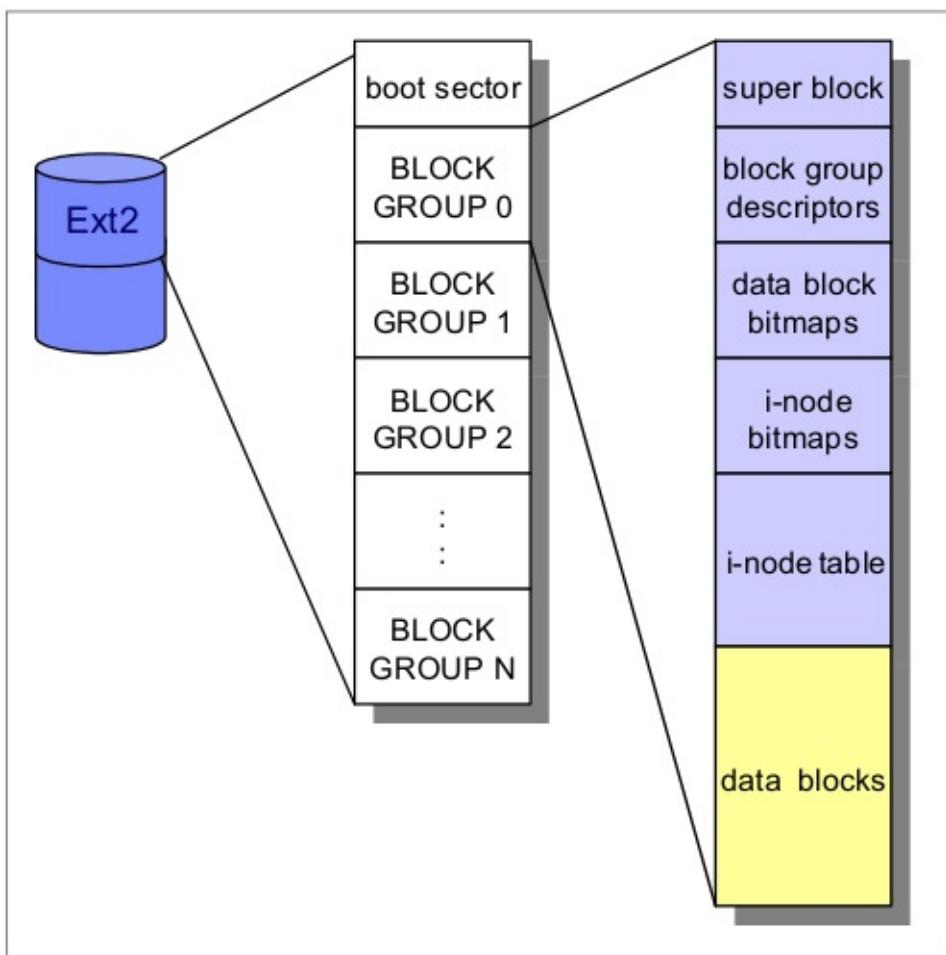


Figure 1-16 Ext2 file system data structure

为了找到组成文件的数据块，内核首先搜索文件的i-node。在接到进程打开/var/log/messages文件的请求时，内核分析文件路径，然后搜索/（根目录）的条目，其中包含了它自己目录下的文件和目录信息。这时候，内核会找到/var目录的i-node，然后再来看看/var目录，它也有自己目录下的文件和目录的信息。内核会一直执行上面的过程，直到找到/var/log/messages文件的i-node。Linux内核使用对象缓存，比如目录条目缓存或者i-node缓存来加速寻找i-node的过程。

一旦内核找到了文件的i-node，然后就试着访问真正的用户数据块。如前面解释的那样，i-node有数据块的指针。参照它，内核就能获得数据块了。对于大文件来说，Ext2提供到数据块的直接/非直接参照。下图展示了它是如何工作的。

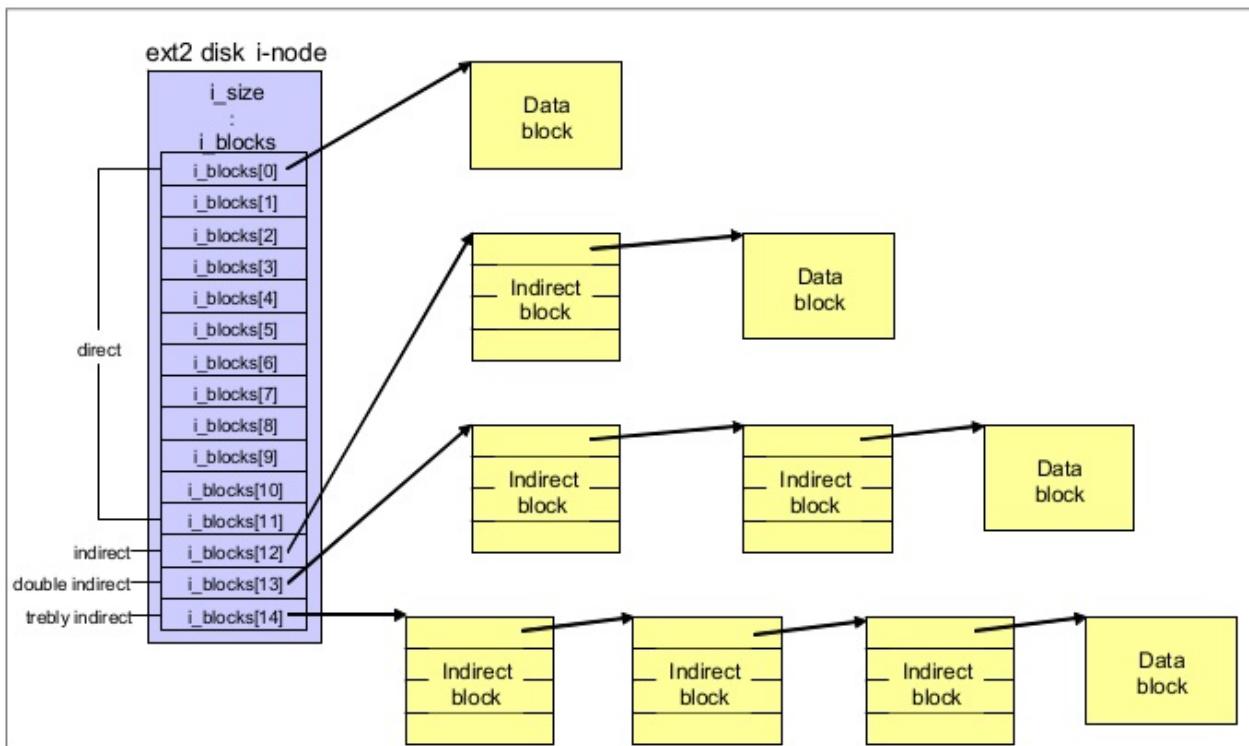


Figure 1-17 Ext2 file system direct / indirect reference to data block

不同的文件系统其文件系统结构和文件访问操作也不一样。所以，各个文件系统都有其特色。

Ext3

现在，Linux的企业发行版都已经支持Ext3（Extended 3）文件系统，它是被广泛使用的Ext2的更新版本。Ext3和Ext2的基本结构是相似的，主要区别是Ext3支持日志。Ext3有如下亮点：

- 可用性：Ext3采用一致性的方法把数据写入磁盘，所以，当系统出现意外宕机（断电或者系统崩溃），服务器再次启动时不用检查数据的一致性。可以把系统恢复时间从几小时缩短到几秒钟！
- 数据完整性：在`mount`命令中使用`data=journal`打开日志记录模式，所有的数据，包括文件和元数据都会记录日志！
- 速度：通过指定日志模式`data=writeback`，你可以在你的业务场景下，在速度和完整性之间做权衡。在有大量同步写的时候，这个效果会十分显著。
- 灵活性：把现有的Ext2很简单就能升级到Ext3，而且不用重新格式化。通过执行`tune2fs`命令和修改`/etc/fstab`文件，你能够很简单的把Ext2升级到Ext3文件系统！同样，在关掉Ext3日志的情况下，Ext3能够作为Ext2文件系统进行挂载。很多第三方工具都能修改Ext3的文件系统！例如，PartitionMagic能够修正Ext3分区。

日志模式

Ext3支持3种日志模式。

- 日志（journal） 通过记录文件数据和元数据，这个模式提供了最高的数据一致性。它的

性能消耗也是最大的。

- 有序（ordered） 该模式下只记录元数据。然而，优先保证写入文件数据！
- 回写（writeback） 这个日志选项提供最快的数据访问能力，但是牺牲了数据的一致性！
保证数据一致性的元数据也会被记录，但是没有处理确定的文件数据，在系统崩溃的时候，这可能导致旧的数据出现在文件中。

ReiserFS

ReiserFS是一个快速的日志文件系统，具备优化磁盘空间利用率和快速的崩溃恢复功能！在Novell的帮助下，ReiserFS已经变得十分好用。ReiserFS只在Novell SUSE Linux上享受商业支持！

日志文件系统

日志文件系统（Journal File System，JFS）是一个全64位文件系统，能支持超大文件和分区。JFS起初由IBM为AIX开发，现在已经在GPL许可证下发布。在高性能计算（high performance computing，HPC）和数据库这类需要支持超大分区和文件大小的情况下，JFS是一个理想的文件系统。如果想进一步了解JFS，参考 <http://jfs.sourceforge.net>

在 Novell SUSE Linux Enterprise Server 10上，JFS不再作为新的文件系统支持！

XFS

扩展文件系统（eXtended File System，XFS）是一个起初由硅图形公司（Silicon Graphics Incorporated）为他们的IRIX系列系统开发高性能日志文件系统。它和IBM的JFS的特点较为相识，也支持超大文件和分区。而且，使用场景也很相似。

磁盘I/O子系统

- 磁盘I/O子系统
 - I/O子系统结构
 - 缓存
 - 块层
 - I/O设备驱动
 - RAID和存储系统

在进程解码和执行指令之前，要把数据从盘片的扇区中恢复到进程的缓存和寄存器中。程序执行结果又被写回到磁盘中。

本节中我们会了解Linux的I/O子系统，这也是一个影响系统性能的重要部分。

I/O子系统结构

下图展示了基本的I/O子系统架构。

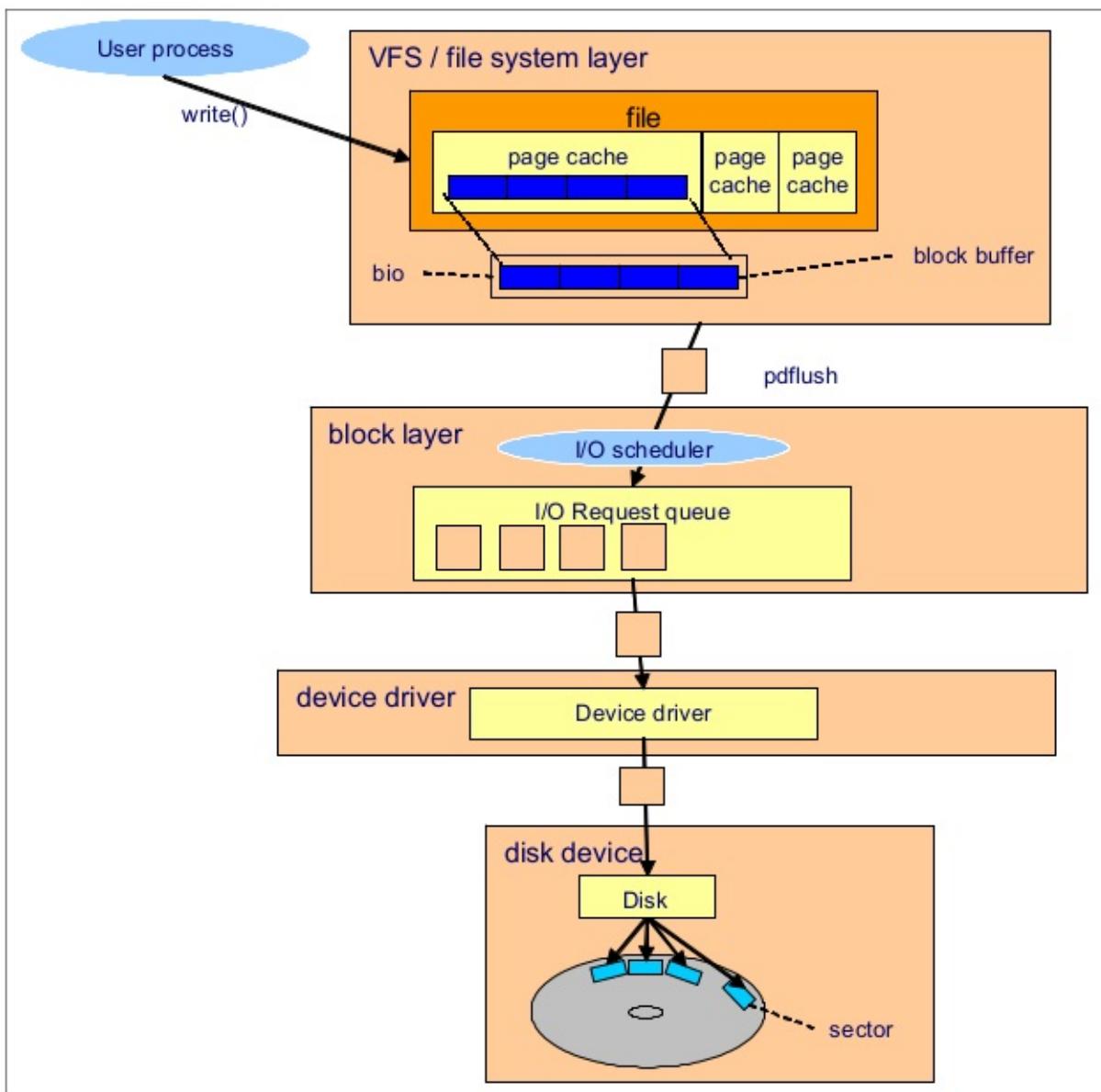


Figure 1-18 I/O subsystem architecture

为了快速的全面的了解I/O子系统操作，我们使用一个把数据写入磁盘的例子。当执行向磁盘写入数据操作的时候，会发生如下的一系列基本操作。假设文件数据存在于磁盘扇区上，并且已经被读入到页缓存中。

1. 进程使用 `write()` 系统调用写入文件。
2. 内核更新映射到文件的 **page cache**。
3. 内核线程 **pdflush** 负责把页缓存刷入到磁盘中。
4. 文件系统层把各个块缓存放入一个 **bio** 结构，并且提交一个写入到块设备层的请求。
5. 块设备层从上层获得请求，执行 **I/O elevator** 操作，把请求放入到 I/O 请求队列中。
6. 磁盘驱动，例如 SCSI 或其它特定驱动将会负责写操作。
7. 磁盘驱动固件执行硬件操作，例如寻址、旋转、数据传送到磁盘的扇区。

缓存

在过去的20年里，处理器的性能提升要大于其它计算机组件（例如处理器缓存、总线、RAM和磁盘等）。因为存储器和磁盘的速度限制了整个系统性能，所以系统的整体性能并没有因为处理器速度的提升而提升。但是，通过把常用数据放入到更快速度的内存中，以缓存机制可以解决这个问题。它减少了访问比较慢的存储器的次数。现代计算机系统在几乎所有的I/O组件中都使用了这项技术，例如硬盘驱动缓存（hard disk drive cache）、磁盘控制器缓存（disk controller cache）和文件系统缓存（file system cache），各个应用都使用到了缓存。

存储器层次

下图展示了缓存等级的概念，由于CPU寄存器和磁盘之间的访问速度差异很大，CPU会花很多时间等待磁盘中的数据，这导致CPU的高性能无用武之地。存储器层次结构通过L1 cache、L2 cache、RAM和其它在CPU和磁盘之间的缓存来消除这种影响。这减少了进程访问较慢存储器和磁盘的机会。离处理器比较近的缓存拥有更高的CPU访问速度和较小的空间。

这项技术还带来了局部引用（Locality of reference）的好处，越高的缓存命中率和越快的内存，就能越快的获取到数据。

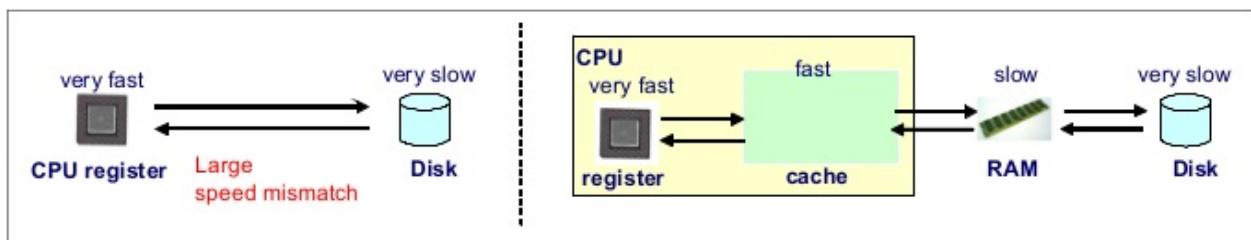


Figure 1-19 Memory hierarchy

局部性引用（Locality of reference）

如上文所说，高缓存命中率是提升性能的关键。为了获得高缓存命中率，使用“局部性引用”的技术。这个技术基于如下的原则：

- 最近使用过的数据即将被使用的可能性很高（时间局部性，temporal locality）。
- 使用过数据的附近数据被使用的可能性很高（空间局部性，spatial locality）。

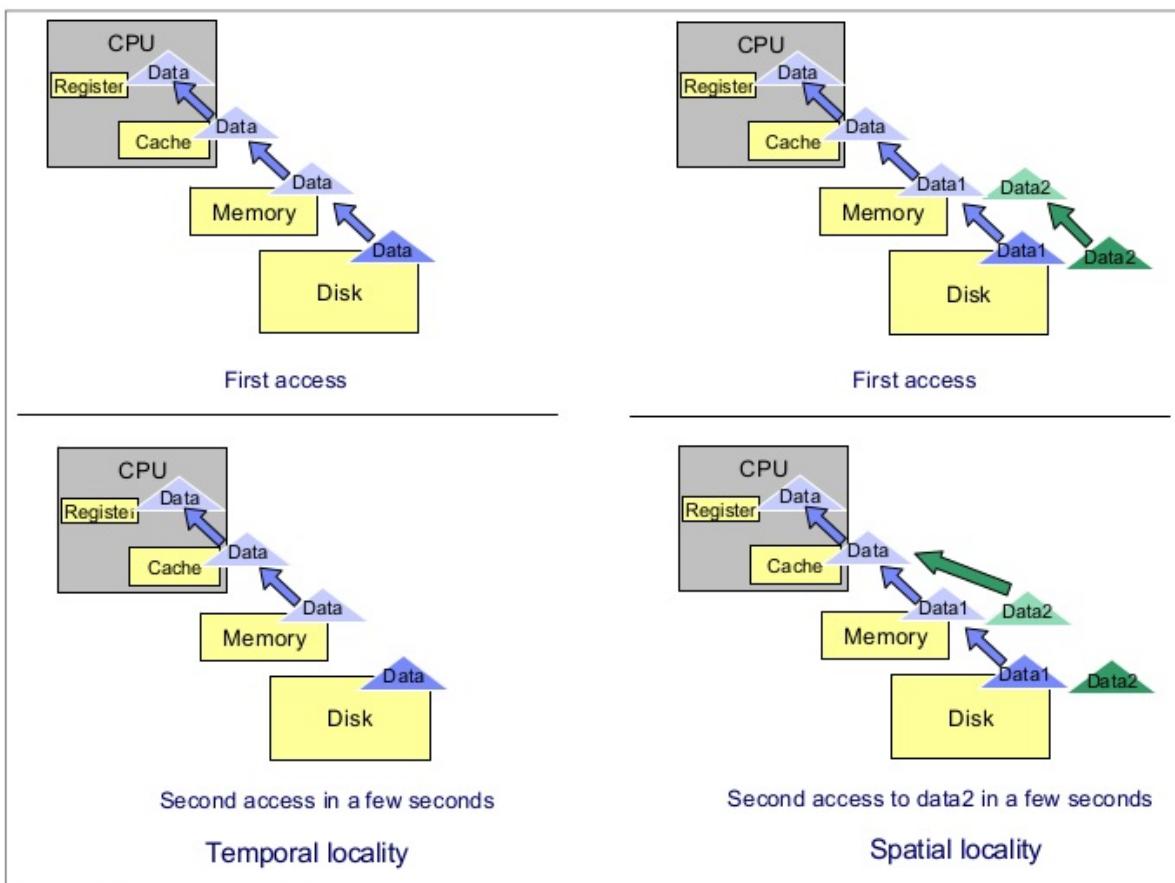


Figure 1-20 Locality of reference

Linux在很多组件中用到了这个原则，例如页缓存、文件对象缓存（i-node缓存、目录条目缓存等等）、预读缓冲区等。

刷新脏缓冲区（Flushing a dirty buffer）

在进程从磁盘中读数据时，数据被复制到内存中。该进程和其它进程都可以在内存缓存中读取同样的数据副本。当进程尝试改变数据，进程首先修改内存中的数据，这时候，磁盘和内存中的数据就不一致了，内存中的数据就叫做脏缓冲（*dirty buffer*）。脏缓冲应该尽快同步到磁盘上，否则，如果突然崩溃，内存中的数据会丢失。

同步脏缓冲的进程叫做*flush*，在Linux内核2.6中，*pdflush*内核线程负责把数据写入到磁盘上。数据会定时刷新（*kupdate*），或者当内存中的脏缓冲到了阀值的比例的时候（*bdfflush*）。这个阀值在`/proc/sys/vm/dirty_background_ratio`文件中。更多信息，请参考“设

置内核swap和pdflush行为“。

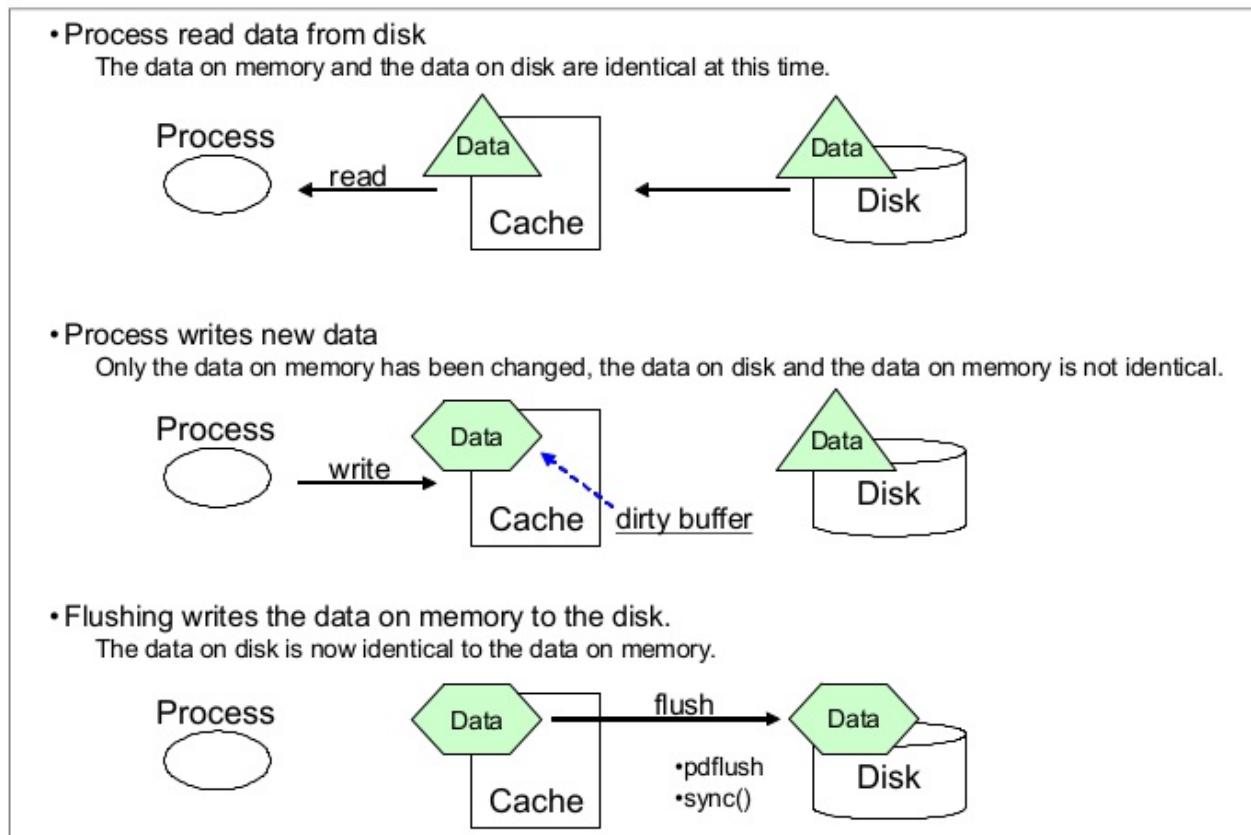


Figure 1-21 Flushing dirty buffers

块层

块层处理所有和块设备相关的操作。块层中的关键数据结构是 *bio*。*bio* 结构是文件系统层和块层之间的一个接口。

在执行写操作时，文件系统层尝试写入由块缓冲组成的页缓存。通过把相邻的块放在一起，组成 *bio* 结构，然后把 *bio* 发送给块层。

块层处理 *bio* 请求，并且把请求链接到 I/O 请求队列中。这个链接操作叫做 *I/O elevator*。在 Linux 内核 2.6 中，有四种类型的 *I/O elevator* 算法。

块大小

驱动器上可以读出和写入的最小数据量会对服务器性能有直接影响。作为参考，如果服务器要处理很多小文件，设置较小的块大小比较好。如果服务是需要处理大文件的，较大的块大小可能会提高性能。不能改变运行中的文件系统的块大小，只有重新格式化才能修改当前的块大小。

I/O elevator

Linux2.6 内核使用了一种新的I/O elevator模型。Linux2.4提供的是一种通用目标的I/O elevator，2.6则提供了四个可用的elevator。因为Linux操作系统有各种广泛的用途，在不同场景下，I/O和负载都有很大的差异。一台Linux笔记本要满足的的I/O需求可能比一个10000用户的数据库系统还多样。为了满足多样性，有四种可选的I/O elevator。

- 预期 (Anticipatory)

预期 I/O elevator是基于假设一个块设备只有一个物理寻道指针（例如一块SATA盘）。预期 elevator使用期限 (deadline) 机制，并且加上了启发式的期限。正如名字所表明的，预期I/O elevator尝试往磁盘中写入一个大的流，而不是很多非常小的随机磁盘访问。启发式预期可能会导致潜在的写I/O。它一般适用于高吞吐的通用操作系统，例如pc。在内核2.6.18中，elevator是标准的I/O调度算法，但是，大多数企业Linux发行版还是默认使用CFQ elevator。

- 完全公平排队 (CFQ, Complete Fair Queuing)

通过为每个进程维护I/O队列，CFQ为进程实现QoS（服务质量，Quality of Service）策略。CFQ elevator适用于拥有很多进程竞争的极多用户的系统。它努力的避免进程饿死，具有低延时的特点。从内核2.6.18开始，加强版的CFQ elevator是默认的I/O调度器。

基于系统设置和负载模式，CFQ调度器可能拖慢一个主进程的运行，例如，一个使用公平算法的大型数据库系统。根据默认配置，会完全基于竞争处理进程组。一个单一的数据库，所有的通过页缓存（所有的pdflush实例都在一个pgroup）的写都被CFQ认为是能够和其它很多后台进程竞争的一个进程。在这个场景下，尝试I/O调度器子配置或deadline调度器可能更加有用。

- 期限 (Deadline)

期限elevator是使用dealine算法的循环elevator（轮询,round robin），提供接近实时行为的I/O子系统。在维护不错吞吐量磁盘的时候，期限elevator具有优良的请求延迟。期限算法确保不会发生进程饥饿的状况。

- NOOP

NOOP表示没有操作（No Operation），顾名思义。NOOP elevator很精简，它是简单的FIFO队列，不做任何数据排序。NOOP把相邻的数据请求做简单的合并，对磁盘I/O来说，它增加了非常小的处理器开销。NOOP elevator假设块设备拥有自己的elevator算法，例如SCSI的TCQ，或者没有寻道延迟，例如flash卡。

注意：在内核2.6.18中，可以为每个磁盘子系统选择不同的I/O elevator，而不必要对整个系统做统一的设置。

I/O设备驱动

Linux内核使用设备驱动控制设备。设备驱动通常是一个独立的内核模块，为各个设备或者一

组设备提供Linux操作系统支持。一旦设备驱动被载入，就作为Linux内核的一部分运行，并且完全控制设备。这里我们会讨论SCSI设备驱动。

SISC

小型计算机系统接口（Small Computer System Interface，SCSI）是最常用的I/O设备和技术，尤其在服务器环境。在Linux内核中，SCSI设备也受设备驱动模块控制。它由如下几个类型的模块组成。

- 上层驱动程序（upper level drivers）：`sd_mod`, `sr_mod` (SCSI-CDROM), `st` (SCSI Tape), `sq` (SCSI generic device) 等等。它们提供了各种类型SCSI设备的驱动功能，例如SCSI CD-ROM。
- 中间层驱动：`scsi_mod`。实现SCSI协议和通用的SCSI功能。
- 低级别驱动程序，提供到各设备的低级别接入。底层驱动程序，基本上是特定于硬件设备，并且提供给每个设备。例如，`ips`是IBM ServerRAID控制器，`qla2300`专门为Qlogic HBA，`mptscsih`是LSI Logic SCSI的驱动器，等等。
- 伪驱动程序：`ide-scsi`。用作IDE-SCSI仿真。

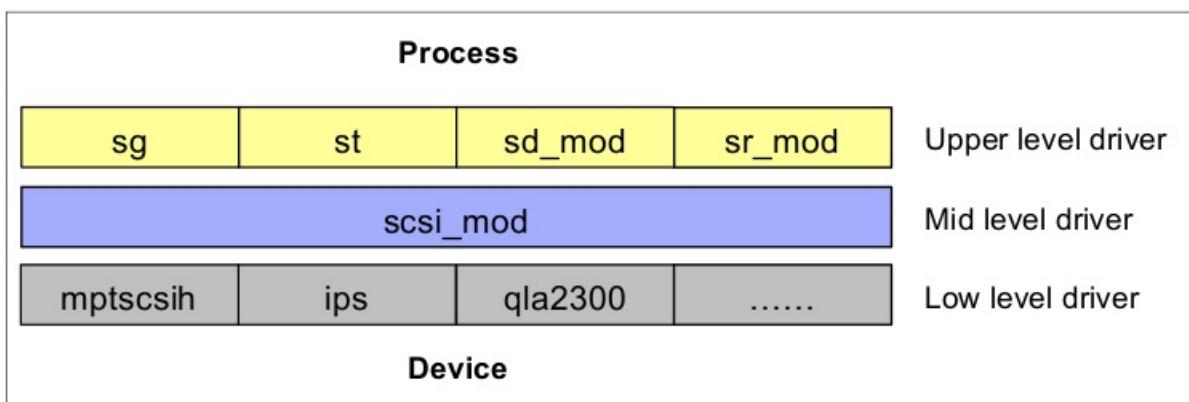


Figure 1-22 Structure of SCSI drivers

如果某一指定功能专为该款设备而设立，那么它应该在设备固件和低级别驱动程序中实现功能。所支持的功能由你所使用的硬件和设备驱动版本决定。设备本身应该也支持所需求的功能才行。指定功能通常由设备驱动参数进行优化。你可以尝试在`/etc/modules.conf`中做一些性能调优。参考设备和设备驱动器文档来获得调优指导。

RAID和存储系统

存储系统的选型和配置，以及RAID类型都是影响系统性能的重要因素。Linux支持软RAID，但是这部分的细节不在本文的讨论范围之类。我们在“安装Linux之前的硬件考虑”一章中涉及了一些调优的方法。

部分IBM存储的解决方法，如下：

- IBM System x Servers 性能调优，SG24-5287。
- IBM System Storage操作手册，SG24-5250。

- SAN（存储区域网络，Storage Area Networks）介绍，SG24-5470。

网络子系统

- 网络子系统
 - 网络实现
 - TCP/IP
 - Offload
 - 绑定模式

网络子系统是另一个影响性能的重要子系统！网络操作相关的组件有很多，例如交换机、路由器、网关、PC等等。尽管这些组件不受到Linux系统的控制，但是，他们对系统的整体性能有很大影响。请注意，你必须要和维护这个网络系统的人紧密联系。

现在，让我们主要集中注意力看看Linux是如何处理网络操作的。

网络实现

TCP/IP协议和OSI模型有类似的层级结构。Linux内核的网络实现采用了相似的办法。下图展示了Linux的TCP/IP栈的层级和TCP/IP通信概览。

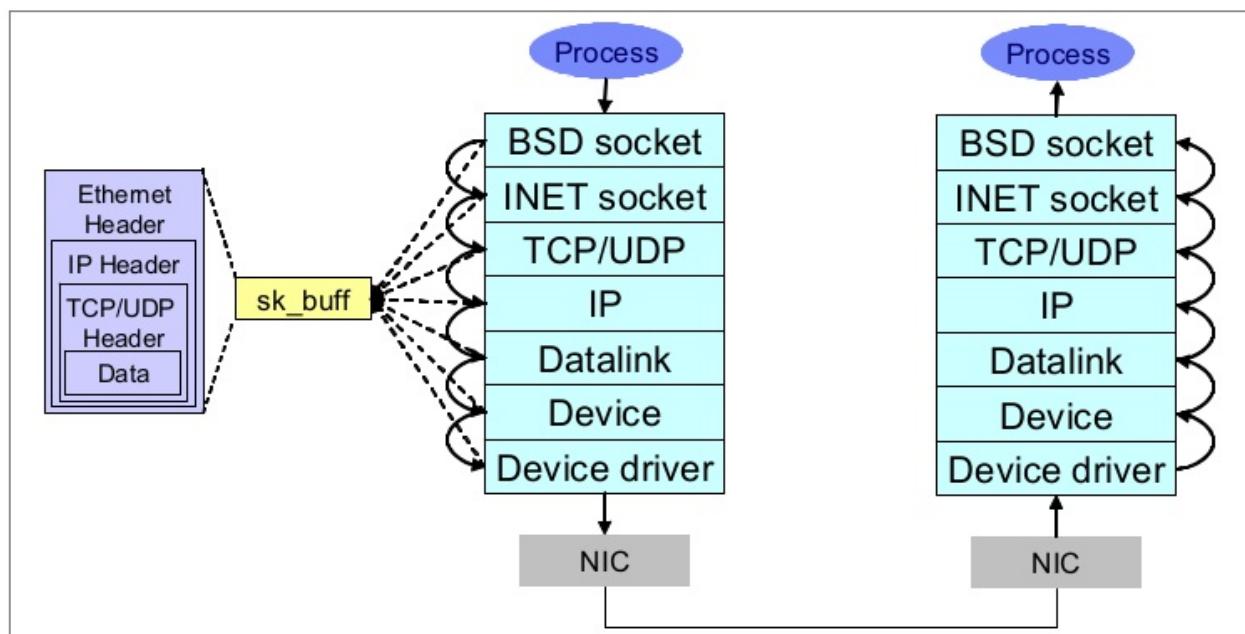


Figure 1-23 Network layered structure and overview of networking operation

和很多 Unix 操作系统一样，Linux 为 TCP/IP 网络操作提供套接字（Socket）接口。套接字为用户应用程序提供接口。下面我们看看在网络数据传递过程中，基本的数据处理顺序。

1. 当应用程序要把数据发送给其它主机，应用程序首先创建数据。
2. 应用程序打开套接字，通过套接字接口写入数据。
3. 套接字缓冲（socket buffer）是用来处理传输数据的。套接字缓冲中有数据的参考，数据穿过该层，向下传递。

4. 在每一层，都要做相应的修改，例如修改头部、添加修改包头、校验值、路由、分片等等。当套接字缓冲向下层传递，数据本身不在层之间复制。因为在各层之间复制数据的效率太低，内核只修改套接字缓冲中的参考并且向下层传递，避免不必要的损耗。
5. 最后，数据通过网卡进入网线。
6. 以太帧（Ethernet frame）到达对方的网络接口。
7. 如果MAC地址和网卡MAC地址匹配的话，分片就接收到网卡缓冲中。
8. 网卡把数据包移动到套接字缓冲中，触发一次CPU的硬中断。
9. 然后CPU处理这个数据包，把他一层层向上传递，直到抵达一个应用程序（例如进程的TCP端口），比如Apache。

套接字缓冲（Socket buffer）

正如前文所说，内核使用缓冲来发送和接收数据。下图展示了网络缓存的配置项。可以用/proc/sys/net下的文件调整。

```
/proc/sys/net/core/rmem_max  
/proc/sys/net/core/rmem_default  
/proc/sys/net/core/wmem_max  
/proc/sys/net/core/wmem_default  
/proc/sys/net/ipv4/tcp_mem  
/proc/sys/net/ipv4/tcp_rmem  
/proc/sys/net/ipv4/tcp_wmem
```

这些参数可能会影响网络性能，我们将在“增加网卡缓冲”一章中仔细阐述。

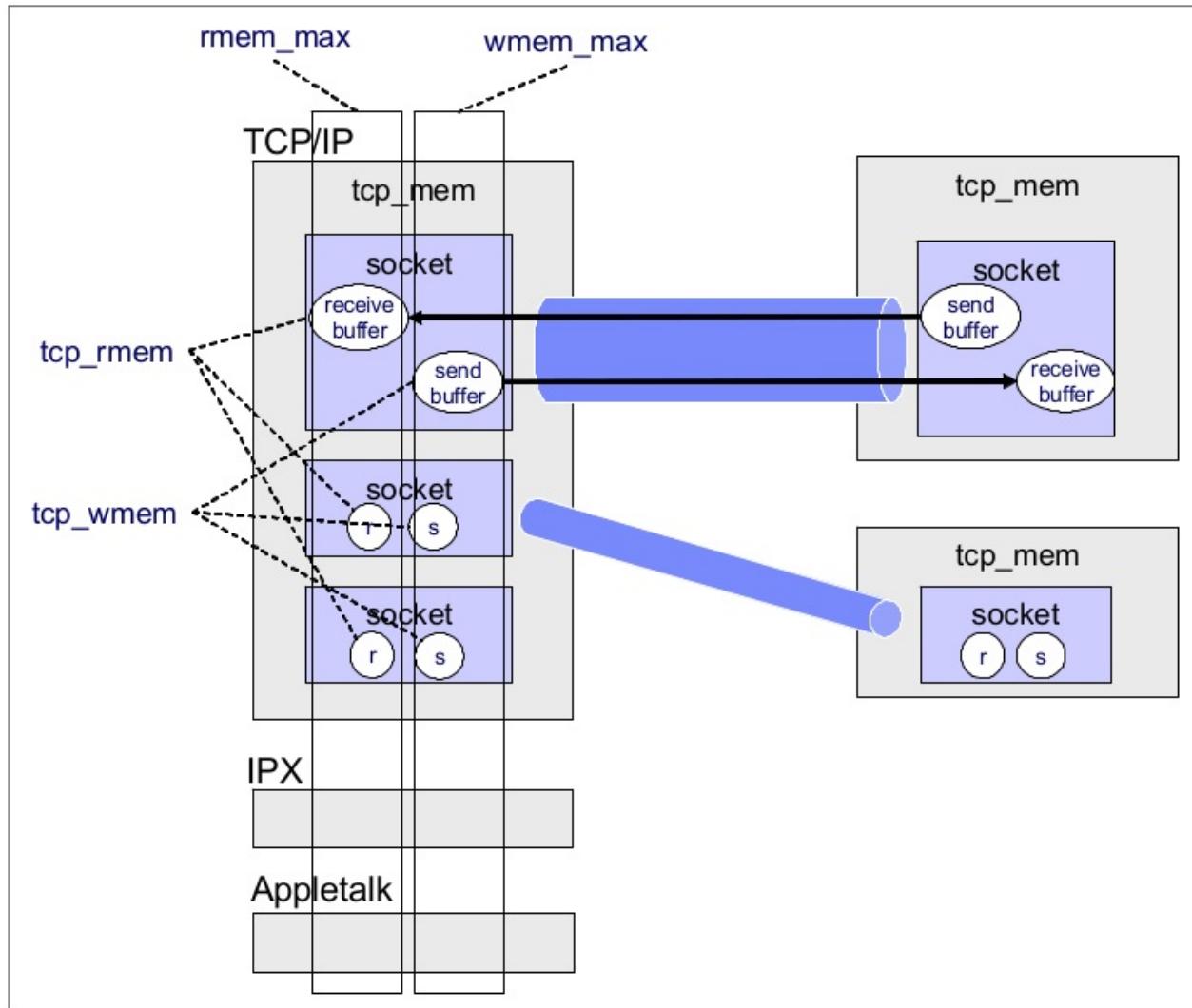


Figure 1-24 socket buffer memory allocation

网络API（Network API, NAPI）

新的网络子系统API发生了一些变化。Linux网络栈的标准实现更关注可靠性和低延时，而不是低负载和高吞吐。这个特征虽然有利于创建防火墙，但是大多数企业应用，例如文件、打印和数据库都会比相同情况下的windows慢一些。

在一个典型的网络包处理中，如下图蓝色箭头所描绘的，网卡把数据包移动到操作系统内核的网络缓冲，并且触发一个CPU的硬中断。

这只是进程处理网络包的一个简单视图，但是显示出了这种方法的一个不足。每次当一个MAC地址匹配的以太网帧到达接口，都会引起一次硬中断。无论如何，CPU必须停下正在处理的进程，然后处理这个硬中断，引发一次上下文切换和刷新处理器缓存。你可能觉得，如

果只有很少的数据包，那这就不是什么大问题。但是，GB级别的以太网卡和现代的应用程序一秒钟能产生数千个包，导致数量庞大的中断和上下文切换。

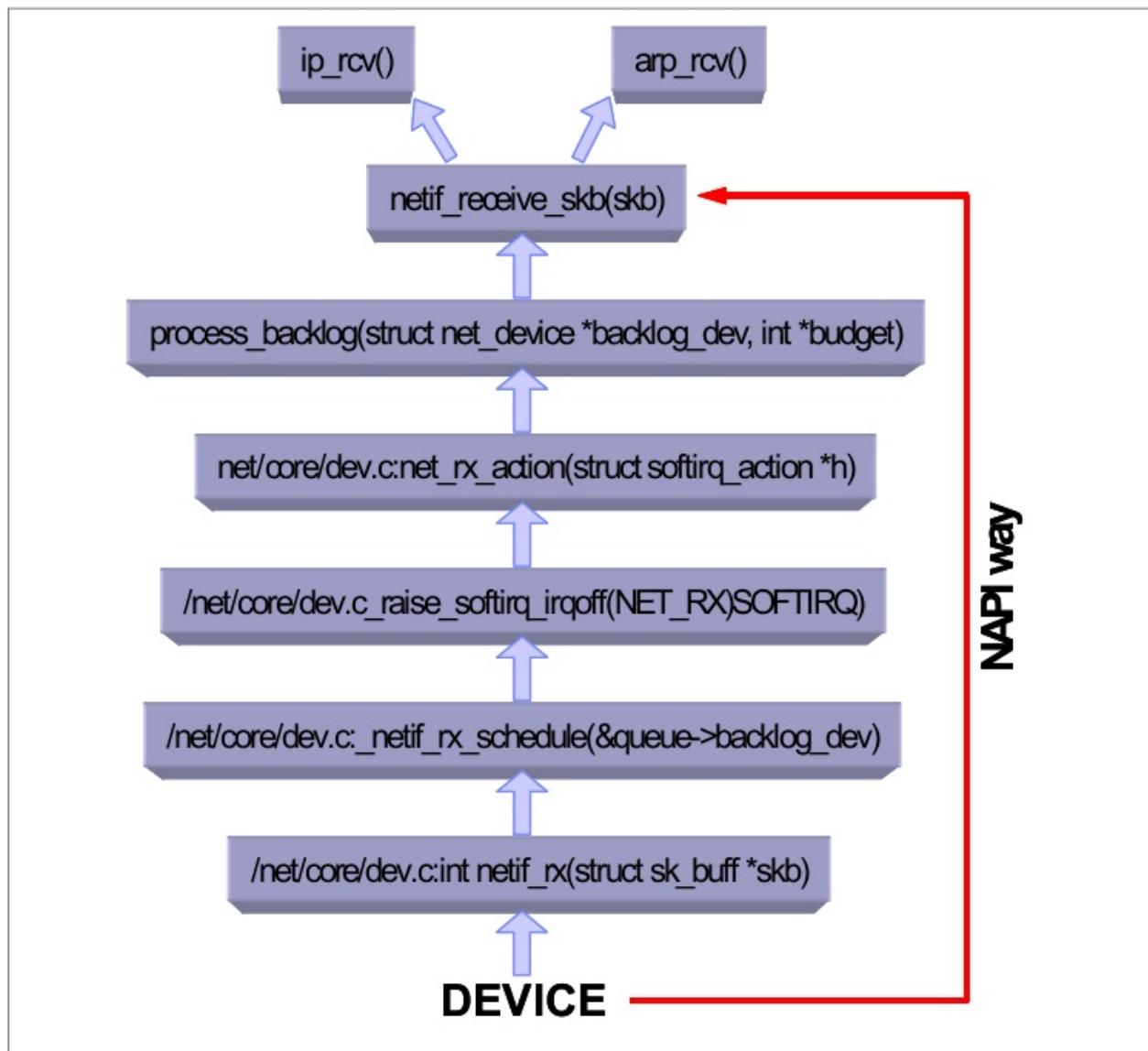


Figure 1-25 The Linux network stack

因此，Linux引入NAPI来对抗处理网络流量引发的相关开销。对于第一个包，NAPI就和传统的实现一样，触发一次中断。但是第一个包之后的包，接口进入了polling模式。只要还有数据包在网卡DMA的环状缓冲（ring buffer）中，就不会引发新的中断，从而高效的减少了上下文切换和相关的消耗。在最后的包被处理，环状缓存被清空之后，网卡又回到了中断模式。

NAPI的另一个好处是通过生成能被多处理器支持的软中断，增强了多处理器的扩展。NAPI对大多数企业级种的多处理器操作系统都是巨大的改进，需要NAPI支持的驱动。还有极大的调优空间等着我们在后面探索。

Netfilter

Linux已经把防火墙作为内核的一部分。这一功能由Netfilter模块提供。你可以使用iptables命令来管理和配置Netfilter。

一般来说，Netfilter提供了如下的几个功能。

- 数据包过滤：如果数据包匹配到一条规则，Netfilter将会接收、拒绝或者根据其它预定的操作来处理这个数据包。
- 地址翻译：如果数据包匹配到一条规则，Netfilter会根据地址翻译的需求修改这个数据包。

匹配过滤器可以定义如下的属性。

- 网络接口（Network Interface）
- IP地址，IP地址范围，子网（IP address, IP address range, subnet）
- 协议（Protocol）
- ICMP类型（ICMP Type）
- 端口（Port）
- TCP标志（TCP flag）
- 状态（state）

下图展示了数据包如何在Netfilter链中传递，以及顺序中每一个点定义的可用规则。

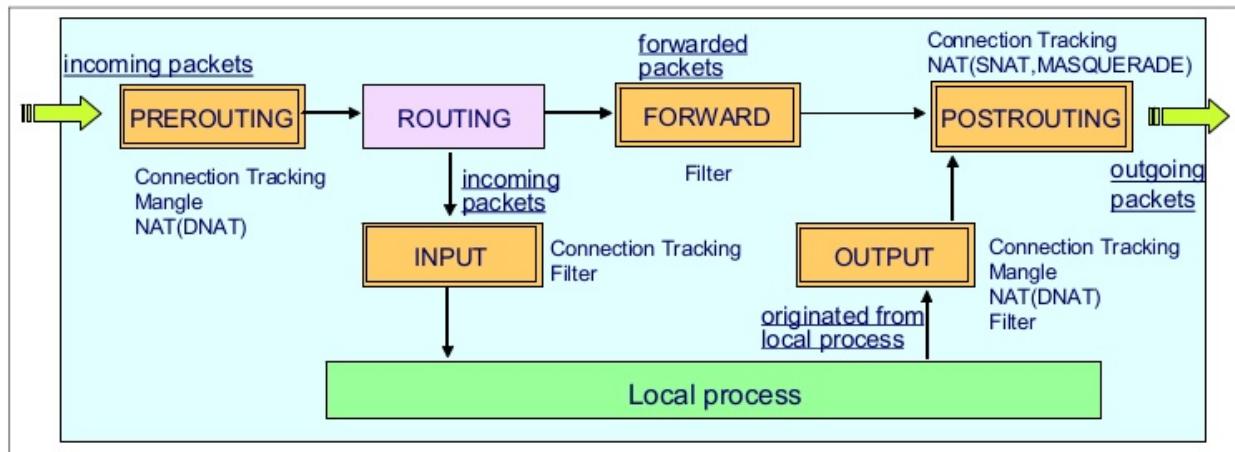


Figure 1-26 Netfilter packet flow

如果匹配到规则，Netfilter将会采取相应的操作，这个操作被叫做目标（target），下面是一些可用的target。

- ACCEPT：接受包，并且放行。
- DROP：丢弃包
- REJECT：丢弃包，并且回复一个消息，例如ICMP端口不可达、TCP重置源主机。
- LOG：记录匹配的包
- MASQUERADE, SNAT, DNAT, REDIRECT：地址翻译（NAT）

连接追踪（Connection tracking）

为了满足更多的防火墙功能，Netfilter使用连接追踪机制，跟踪所有网络流量的状态。通过TCP连接状态和其它网络属性（IP地址，端口，协议，序列号，ack，ICMP类型，等），Netfilter把每一个包分类到下面四个状态。

- NEW：尝试建立新连接的包
- ESTABLISHED：已建立连接的包

- RELATED：和前面的包相关的包
- INVALID：因为异常或者非法的不知状态的包

作为补充，Netfilter可以使用其它模块，通过分析协议特定属性和操作进行更详细的连接跟踪。例如，有FTP、NetBIOS、TFTP、IRC的连接跟踪模块。

TCP/IP

TCP/IP一直是默认的网络协议。Linux上的TCP/IP实现是十分契合TCP/IP标准的。为了更好的性能优化，应该熟悉基本的TCP/IP网络。

连接建立

在应用数据传输之前，服务器和客户端就应该建立起连接，连接建立的过程叫做TCP/IP三次握手。下图展示了基本的连接建立和中断过程。

1. 客户端发送SYN包（带有SYN标志设置的包）到服务器，请求连接
2. 服务器收到SYN请求包，回复一个SYN+ACK的包
3. 然后客户端发送ACK包给服务器完成连接建立。

一旦建立起连接，应用数据就可以通过这个连接来传送，在所有的数据传输完成之后，连接关闭。

1. 客户端发送FIN包给服务器，开始终止连接的过程。
2. 服务器回复一个ACK的确认包回去，如果再没有数据要发送给客户端，服务器然后发送一个FIN包给客户端。
3. 客户端发送一个ACK包给服务器，完成连接终止。

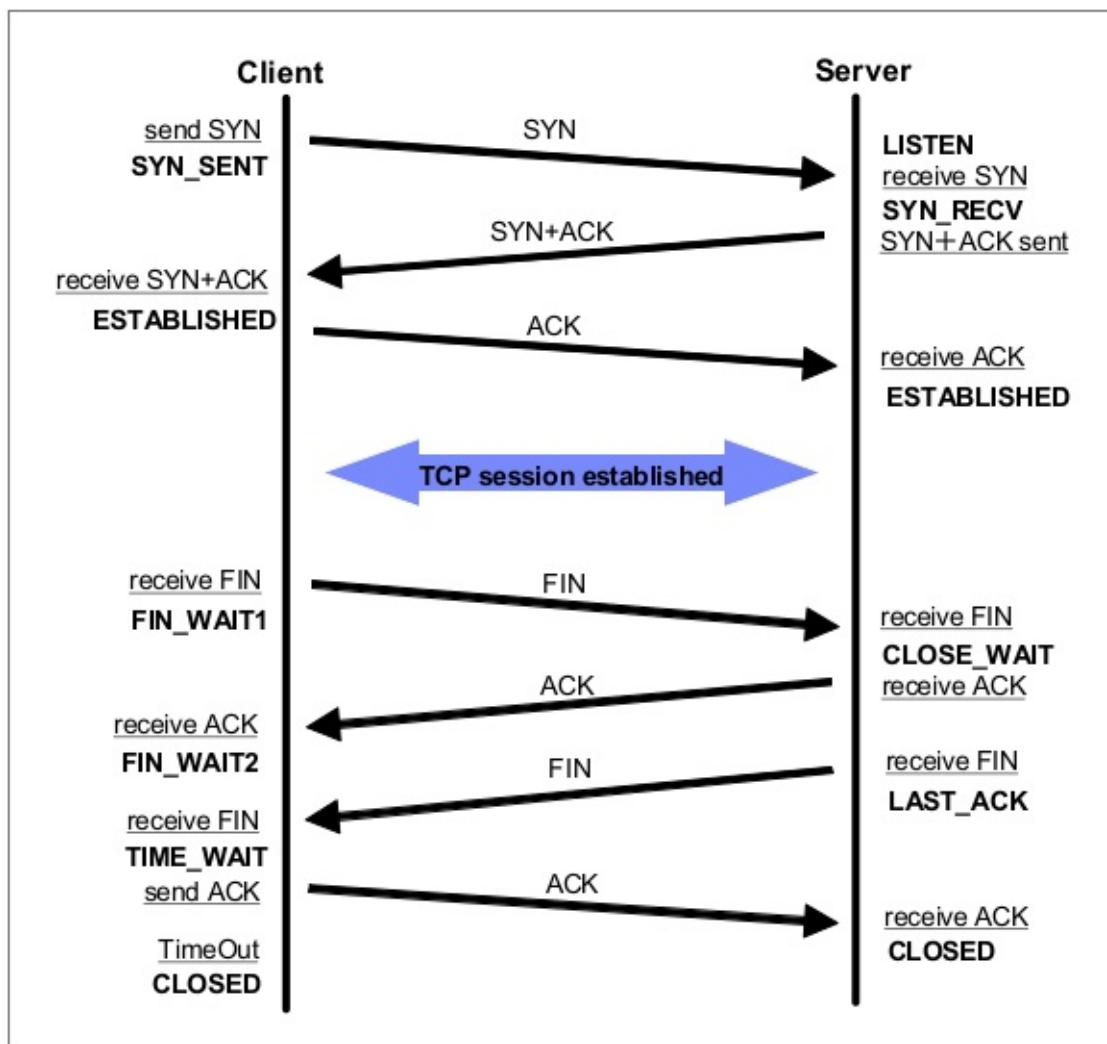
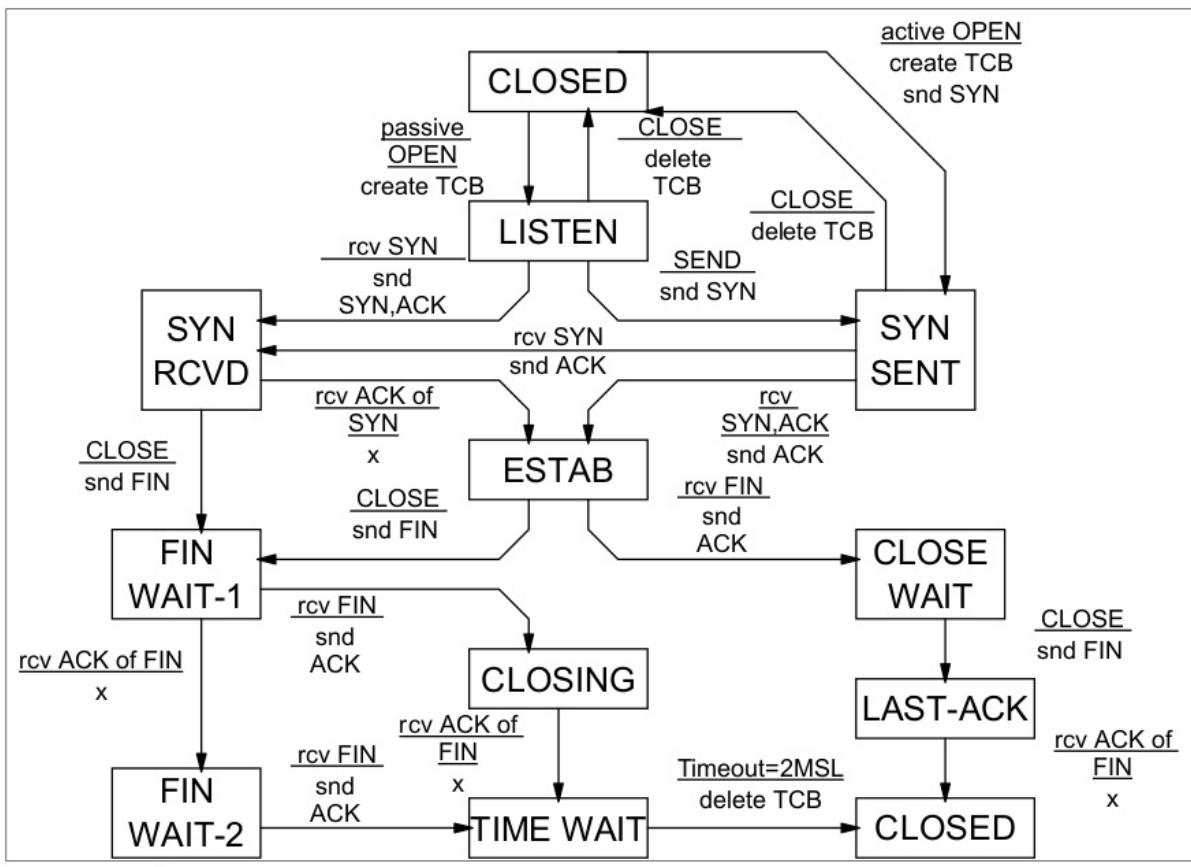


Figure 1-27 TCP 3-way handshake

在回话期间，连接状态会发生改变。下图展示了TCP/IP连接状态图。



可以使用 `netstat` 命令来查看每个 TCP/IP 会话的连接状态。

流量控制 (Traffic control)

TCP/IP 有一种机制，确保即使在拥挤时段和网络传输质量恶劣的情况下，高效数据传输！

TCP/IP 滑动窗口 (TCP/IP transfer window)

滑动窗口是 Linux 操作系统上 TCP/IP 实现的重要组成。基本上，TCP 滑动窗口就是在发送数据之前，一台机器和对方机器确认能发送和接收的最大数据量的机制。窗口大小放在 TCP 头的滑动窗口字段中，由接受端传递给发送端。使用滑动窗口可以使数据传递更加高效，因为发送主机不必等待每一个数据包的确认。这使得网络更加有效利用，也提高了延迟确认效率。在连接中，TCP 窗口从很小开始，随着收到每一个对端的确认而慢慢增长。想知道如何优化滑动窗口，请看“增加网络缓冲”一节。

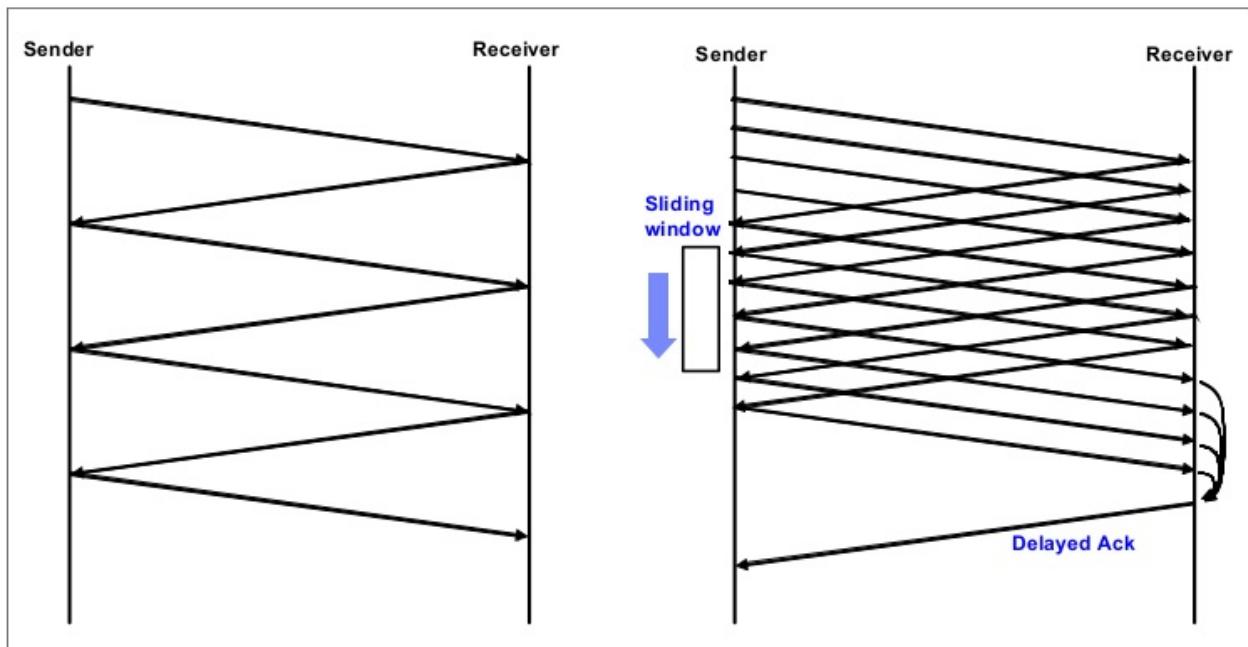


Figure 1-29 Sliding window and delayed ack

另外，高速网络可以使用叫做*window scaling*的技术来增加滑动窗口的大小。将在“调优TCP选项”一章中分析这一技术的实现和效率。

重传

在连接建立、终止和数据传输过程中，由于各种原因（网络接口故障，路由器太慢，网络拥堵，网络实现的问题等等），会发生许多超时和数据重传。TCP/IP通过数据包排队和多次重传数据包来解决这个问题。

你可以配置参数来改变内核行为。你可能想在高丢包率的网络中，增加TCP SYN建立连接包的尝试次数。你也通过/proc/sys/net下的文件改变超时阀值。更多信息参考“调优TCP行为”一节。

Offload

如果你的网卡适配器支持硬件offload功能，内核可以卸载一部分任务给适配器，从而减轻CPU的负载。

- 校验值卸载（Checksum offload）

TCP/IP/UDP校验值是协议头部的字段，由包数据计算获得，通过比较校验值可以保证数据包被正确的发送！

- TCP分片卸载（TCP segmentation offload，TSO）

如果数据比网卡支持的最大传输单元（maximum transmission unit，MTU）更大，这个数据就要分割成小于或等于MTU大小的包。适配器的这个选项由内核指定。

绑定模式

利用 *bonding* 驱动，Linux 内核提供网卡聚合功能。这是一个设备无关的绑定驱动（也有些设备需要特定的驱动）。*bonding* 驱动支持 802.3 链路聚合和一些负载均衡和容错实现。可以获得高级别的可靠性和性能提升。

理解Linux性能指标

- 理解Linux性能指标

- 处理器指标
- 内存指标
- 网络接口指标
- 块设备指标

在我们学习各类调优参数和性能分析工具之前，我们先理解各个衡量指标和它们在系统性能中所表示的含义。因为Linux是开源操作系统，所以它有很多性能测试工具。你最终选择的工具，是根据你的偏好和具体需求来决定的。尽管有很多工具可用，但是它们所衡量的指标是一样的，所以理解了指标，你可以使用任何工具。因此，我们只说最重要的指标，有很多很细的值可能在具体的分析中很有用，但是它们不在本文讨论范围。

处理器指标

处理器指标如下：

- CPU利用率（CPU utilization）

这个可能是最直接的指标，它全面展示了每个处理器的利用率。在IBM System x架构中，如果CPU利用率持续高于80%，就可能遇到了处理器瓶颈。

- 用户时间（User time）

表示CPU在用户进程上的时间百分比，包括nice时间。用户时间值高是一个较好的状态，在这种情况下，系统在处理真正的任务。

- 系统时间（System time）

表示CPU花在内核操作上的时间百分比，包括IRQ和softirq时间。持续的高系统时间可以指出网络和驱动栈的瓶颈。CPU花在内核上的时间越少越好。

- 等待（Waiting）

CPU花在等待I/O操作上的时间总和。类似**blocked**值，系统不应该把大量时间花在等待I/O操作上；否则，你应该调查I/O子系统的性能。

- 空闲时间（Idle time）

表示系统处于空闲等待任务的时间比。

- Nice时间（Nice time）

表示CPU花在re-nicing进程，改变进程执行顺序和优先级上的时间。

- 平均负载（Load average）

平均负载不是百分比，是下面的和的滚动平均值：

- 在队列中等待被处理的进程数
- 等待非中断任务完成的进程数

是TASK_RUNNING和TASK_UNINTERRUPTIBLE的和的平均值。如果进程请求CPU时间被阻塞（表示CPU没有时间处理它们），平均负载就会升高。另一方面，如果每个进程直接就能获得CPU时间并且没有CPU周期丢失，负载就会降下来。

- 可运行进程（Runnable processes）

表示已经准备好要执行的进程。这个值不应该持续超过CPU个数的10倍，否则就是出现了CPU瓶颈。

- 阻塞的（Blocked）

在等待I/O操作完成的时候，进程不能执行。阻塞进程可以指出你的I/O瓶颈。

- 上下文切换（Context switch）

系统上有大量的切换在线程间发生，在有大量中断和上下文切换发生时，表示驱动或应用程序出现了问题。一般来说，上下文切换不是好现象，因为CPU缓存需要刷新，但是有些上下文切换是必要的。

- 中断（Interrupts）

中断值包含硬中断和软中断。硬中断对系统性能有更大的影响。高中断值指示了软件瓶颈，无论是内核还是驱动程序层面的。记住中断值包含CPU时钟引起的中断。

内存指标

如下是内存度量值：

- 空闲内存（Free memory）

和其它操作系统相比，不应该过分担心Linux内存的问题。在“虚拟内存管理”一节中已经说过，Linux把大部分没用到的内存作为文件系统缓存，所以计算空闲内存的时候还得加上已用内存中的缓冲（buffer）和缓存(cache)大小。

- Swap利用率（Swap usage）

这个数值表明已经使用的swap的空间。如“虚拟内存管理”一节中所说的那样，swap使用率只是表明Linux的内存管理有多么高效。Swap In/Out才是识别内存瓶颈的手段，长时间每秒200到300以上的swap in/out次数表明可能出现内存瓶颈。

- 缓冲和缓存（Buffer and cache）

cache被用作文件系统缓存和块设备缓存。

- Slabs

描述内核的内存使用量。注意，内核页不能page out到磁盘。

- 活动和非活动内存

提供关于系统中活动的内存信息。非活动内存是可能由kswapd守护进程交换到磁盘的候选。参考“页帧回收”。

网络指标

以下是网络指标：

- 接收和发送的包（**Packets received and sent**）
这个指标告诉你指定网络接口的接收和发送网络包的数量。
- 接收和发送的字节（**Bytes received and sent**）
这个值是指定网卡的发送和接收的字节数。
- 每秒碰撞（**collisions per second**）
这个值提供了各个网络接口所连接网络的所发生的冲突数量。持续的冲突可能是由于网络基础设施导致的，而不是服务器。在大多数正确配置的网络中，碰撞很少发送，除非网络是由集线器组成的。
- 丢包
这是被内核丢弃的包的数量，可能是防火墙配置导致的，也可能是由于缺少网络缓冲。
- 过载（**Overruns**）
过载表示网络接口用光缓冲空间的次数。这个指标应该和丢包联合起来使用，来判断瓶颈是由网络缓冲还是网络队列长度导致的。
- 错误（**Errors**）
被标识为故障的帧数目。这通常是由于网络不匹配或者部分网线损坏导致的。在铜基千兆中，部分损坏网线可能导致显著的网络性能问题。

块设备指标

以下是块设备的相关指标：

- IO等待（**lowait**）
CPU花在等待I/O操作发生上的时间。该值长时间飙高预示着可能出现了I/O瓶颈。
- 平均队列长度（**Average queue length**）
未完成的I/O请求数量。通常，2到3的磁盘队列是很理想的；太高可能表示出现了I/O瓶颈。
- 平均等待（**Average wait**）
一个I/O请求被服务的平均等待时间，以毫秒计算。等待时间由真实的I/O操作时间和I/O队列的等待时间组成。
- 每秒传输（**Transfers per second**）
表示每秒有多少个I/O操作被执行（读和写）。*transfers per second*和*kBytes per second*可以联合使用，来表示系统每秒的平均传输大小。平均传输大小通常应该和所使用的磁盘子系统的条带大小相匹配。
- 每秒读写块（**Blocks read/write per second**）
在内核2.6中，它表示每秒读取和写入1024字节块的数目。更早的内核，块大小可能不一样，从512字节到4K字节不等。
- 每秒读写的千字节（**Kilobytes per second read/write**）
从块设备读和写的千字节，表示从块设备中读取和写入的实际大小。

监控和压测工具

Linux操作系统开放和自由的环境，催生了很多性能监控工具。有些是原先就已知名的UNIX系统工具的Linux版本，还有些是专为Linux而开发的。虚拟的proc文件系统为大多数工具提供重要支持。为了衡量性能，我们还得掌握压力测试的工具。

在本章中，我们将讨论和Linux性能有关的监控工具和相关命令的用法。也会介绍一些有用的压力测试工具。

大多数监控工具都是基于Linux企业发行版来讨论的。

介绍

Linux企业发行版都附带了很多监控工具。有些工具能处理多个指标，并且提供方便理解的输出格式。有些工具只负责特定的性能指标（如磁盘I/O），并且提供详尽的信息。

熟悉这些工具可以加强你对系统情况的了解，发现存在的性能问题。

工具功能概览

下表列出了本章涉及到的监控工具及其功能。

Linux性能监控工具

工具	常用功能
top	所有进程情况
vmstat	系统活动，硬件和系统信息
uptime, w	系统平均负载
ps, pstree	显示进程
free	内存使用情况
iostat	CPU负载和磁盘活动
sar	收集和报告系统状态
mpstat	多处理器使用情况
numastat	NUMA相关统计
pmap	进程内存情况
netstat	网络统计
iptraf	实时网络统计
tcpdump, ethereal	详细网络流量分析
nmon	收集和报告系统活动
strace	系统调用
proc文件系统	各种内核统计信息
KDE system guard	实时的系统图形报告
Gnome System Monitor	实时的系统图形报告

以下是本章设计的压测工具和功能

压测工具

工具	常用功能
lmbench	微型系统功能评测工具
iozone	文件系统压测
netperf	网络性能测试

监控工具

- 监控工具
 - [top](#)
 - [vmstat](#)
 - [uptime](#)
 - [ps and pstree](#)
 - [free](#)
 - [iostat](#)
 - [sar](#)
 - [mpstat](#)
 - [numastat](#)
 - [pmap](#)
 - [netstat](#)
 - [iptraf](#)
 - [tcpdump / ethereal](#)
 - [nmon](#)
 - [strace](#)
 - [Proc文件系统](#)
 - [KDE System Guard](#)
 - [Gnome System Monitor](#)
 - [Capacity Manager](#)

在本小节，我们将讨论常用监控工具。大多数Linux发行版中都携带了这些监控工具，你应该熟练掌握。

top

`top`命令会展示进程的实际活动。默认情况下，它会列出系统上所有cpu密集型任务，并且每5秒钟刷新一次列表。你可以对PID（数值），生存时间（最新的排最前面），时间（累计时间）以及常驻内存使用率和时间（进程启动开始占用cpu的时间）进行排序。

Example 2-1 Example output from the top command

```
top - 02:06:59 up 4 days, 17:14, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 62 total, 1 running, 61 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.2% us, 0.3% sy, 0.0% ni, 97.8% id, 1.7% wa, 0.0% hi, 0.0% si
Mem: 515144k total, 317624k used, 197520k free, 66068k buffers
Swap: 1048120k total, 12k used, 1048108k free, 179632k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
13737	root	17	0	1760	896	1540	R	0.7	0.2	0:00.05	top
238	root	5	-10	0	0	0	S	0.3	0.0	0:01.56	reiserfs/0
1	root	16	0	588	240	444	S	0.0	0.0	0:05.70	init
2	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
4	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
5	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/1
6	root	5	-10	0	0	0	S	0.0	0.0	0:00.02	events/0
7	root	5	-10	0	0	0	S	0.0	0.0	0:00.00	events/1
8	root	5	-10	0	0	0	S	0.0	0.0	0:00.09	kblockd/0
9	root	5	-10	0	0	0	S	0.0	0.0	0:00.01	kblockd/1
10	root	15	0	0	0	0	S	0.0	0.0	0:00.00	irqd
13	root	5	-10	0	0	0	S	0.0	0.0	0:00.02	khelper/0
14	root	16	0	0	0	0	S	0.0	0.0	0:00.45	pdflush
16	root	15	0	0	0	0	S	0.0	0.0	0:00.61	kswapd0
17	root	13	-10	0	0	0	S	0.0	0.0	0:00.00	aio/0
18	root	13	-10	0	0	0	S	0.0	0.0	0:00.00	aio/1

你也可以通过renice命令给进程设置一个优先级，如果一个进程挂了，或者占用太多CPU，你可以杀死（kill命令）这个进程。

输出中的各列：

- PID 进程号
- USER 进程所有者的名字。
- PRI 进程优先级
- NI nice级别
- SIZE 进程使用的内存（代码、数据和栈），kb单位
- RSS 物理RAM使用量，kb单位
- SHARE 和其它进程共享的内存，kb单位
- STAT 进程状态：S=睡眠，R=运行，T=停止或跟踪，D=不可中断的睡眠，Z=僵尸。请参考前文中的“进程状态”一节。
- %CPU CPU使用量。
- %MEM 物理内存用量
- TIME 进程使用的总CPU时间（从启动开始算）
- COMMAND 进程的命令行启动命令（包括参数）

top命令还有如下几个常用的快捷键：

- t 关闭和开启进程汇总信息的展示

- **m** 关闭和开启内存信息的显示
- **A** 排序系统上各类资源的排序。对于快速找出系统上的性能问题的任务很有帮助。
- **f** 进入**top**的交互配置模式，对于给**top**设置特定的进程很有用。
- **o** 让你交互的选择**top**的排序。
- **r** 使用**renice**命令
- **k** 使用**kill**命令

vmstat

vmstat显示关于进程，内存，页，块I/O，traps和CPU的信息。**vmstat**既可以展示平均值，也可以是实时数据。通过提供采样频率和采样时间就可以开启**vmstat**的采样模式。

注意：考虑到采样模式中的短时高峰情况，把采样频率设置为一个较低的值可以避免这样的问题。

Example 2-2 Example output from vmstat

```
[root@lnxsu4 ~]# vmstat 2
procs -----memory----- ---swap-- -----io---- --system-- ----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa
0 1 0 1742264 112116 1999864 0 0 1 4 3 3 0 0 99 0
0 1 0 1742072 112208 1999772 0 0 0 2536 1258 1146 0 1 75 24
0 1 0 1741880 112260 1999720 0 0 0 2668 1235 1002 0 1 75 24
0 1 0 1741560 112308 1999932 0 0 0 2930 1240 1015 0 1 75 24
1 1 0 1741304 112344 2000416 0 0 0 2980 1238 925 0 1 75 24
0 1 0 1741176 112384 2000636 0 0 0 2968 1233 929 0 1 75 24
0 1 0 1741304 112420 2000600 0 0 0 3024 1247 925 0 1 75 24
```

注意：第一行展示了从上次重启之后的平均值，所以应该屏蔽掉。

各列的含义如下：

- **进程**

r:等待执行时间的进程数
b:在不可中断睡眠中的进程数

- **内存**

swpd: 已使用的虚拟内存量
free: 空闲内存量
buff: 作为缓冲的内存
cache: 作缓存的内存

- **Swap**

si : 从交换分区写到内存的量
so : 从内存写到交换分区的大小

- IO

bi : 发往块设备的数目(blocks/s)
bo : 从块设备接收的块数目 (blocks/s)

- System

in : 每秒钟的中断次数，包括时钟
cs : 每秒的上下文切换次数

- CPU (总CPU时间的百分比) :

us : 运行非内核代码的时间 (用户时间，包括nice时间)
sy : 运行内核代码的时间 (系统时间)
id : 空闲时间，早先的Linux2.5.41版本，包含了I/O等待时间
wa : 等待IO的时间，早先的Linux2.5.41版本，这个值为0

vmstat支持很多的命令行参数，vmstat的man手册中写的很详细。其中有如下常用的几个：

- -m 显示内核的内存使用 (slabs)
- -a 显示活动和非活动的内存页
- -n 只显示一个标题行，如果vmstat运行在采样模式，并且使用输出到管道和文件的时候，这选项很有用。（例如，root# vmstat -n 2 10 间隔2秒采样一次，共收集10次数据）
当使用-p {分区} 标志的时候，vmstat会显示I/O的统计。

uptime

uptime 命令可以用来查看服务器运行了多长时间，有多少用户登录在服务器上，以及服务器的平均负载。分别展示过去1分钟、5分钟和15分钟的系统瓶颈负载值。

平均负载最理想的值是1，意味着每个进程可以直接使用CPU，没有发生CPU周期丢失。不同系统的负载有很大差别。对单处理器工作站来说，1或2的负载值是勉强可以接受的，而在多处理器服务器上，平均负载为8或者10的时候，系统依旧运行良好。

使用uptime或许可以找出服务器或网络的问题。例如，当网络服务运行不佳时，你就可以用uptime命令查看系统负载情况。如果负载不高，问题可能出现在你的网络中，而不是服务器系统上。

小提示：你可以使用w替代uptime。w也可以查看当前登录系统的用户，以及他们在做什么。

Example 2-3 Sample output of uptime

```
1:57am  up 4 days 17:05,  2 users,  load average: 0.00, 0.00, 0.00
```

ps 和 pstree

在系统分析中，ps和pstree是最基础的命令，ps有三种不同的命令选项，UNIX、BSD和GNU风格。我们来看看GNU风格的ps选项。

ps命令展示所有进程列表。top命令展示了进程活动，而且ps显示的信息更加详细。ps所显示出来的进程数量取决于所使用的命令参数。简单的ps -A命令会列出所有的进程和他们各自的PID，我们可以使用PID做更多的事情。在使用pmap，renice等工具的时候，就需要用到PID。

在运行java应用的服务器上，使用ps -A命令可能一下子就把显示器全部占满了，很难清楚查看运行进程的完整列表。在这个情况下，pstree命令可能就会派上用场，它把运行进程以树形结构展示，把子进程合并展示（例如java线程）。pstree可以识别出原始进程。ps还有另外一个变种pgrep，也十分有用。

Example 2-4 A sample ps output

```
[root@bc1srv7 ~]# ps -A
  PID TTY      TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 migration/0
    3 ?        00:00:00 ksoftirqd/0
  2347 ?        00:00:00 sshd
  2435 ?        00:00:00 sendmail
  27397 ?        00:00:00 sshd
  27402 pts/0    00:00:00 bash
  27434 pts/0    00:00:00 ps
```

其它的命令选项：

- -e 所有进程，和-A一样
- -l 显示长格式
- -F 额外的全格式，包括参数和选项。
- -H 显示进程等级
- -L 显示线程，可能带有LWP和NLWP列
- -m 在进程后面显示线程

使用如下命令可以看到详细的进程信息：

```
ps -elFL
```

Example 2-5 An example of detailed output

```
[root@lnxsu3 ~]# ps -eLF
F S UID      PID  PPID  LWP  C NLWP PRI  NI ADDR SZ WCHAN   RSS PSR STIME TTY          TIME CMD
4 S root      1    0    1  0    1  76  0 -  457 -      552  0 Mar08 ?          00:00:01 init [3]
1 S root      2    1    2  0    1 -40  - -  0 migrat  0  0 Mar08 ?          00:00:36 [migration/0]
1 S root      3    1    3  0    1  94  19 -  0 ksoftti  0  0 Mar08 ?          00:00:00 [ksoftirqd/0]
```

输出的字段含义：

- F 进程标志
- S 进程状态：S=睡眠，R=运行，T=停止或跟踪，D=不可中断的睡眠，Z=僵尸。
- UID 拥有进程的用户名字。
- PID 进程ID
- PPID 父进程ID
- LWP LWP号（light weight process，or thread，轻量级进程，或线程）。
- C 处理器使用的百分比。
- NLWP 进程中的lwps（线程）个数。
- PRI 进程优先级
- NI nice级别（进程是否通过nice改变优先级，见下文）
- ADDR 进程地址空间（例子中没展示）
- SZ 进程使用的内存大小（代码+数据+栈），单位kb。
- WCHAN 睡眠进程的内核函数名字，如果进程在运行，显示“-”，如果显示为“*”，则表示是多线程。
- RSS 驻留内存大小，任务所使用的非swap物理内存大小，单位是kb。
- PSR 分配给进程的处理器个数。
- STIME 命令开始时间
- TTY 终端
- TIME 进程从启动开始，使用CPU的总时间
- CMD 开启任务的命令（包含参数）

线程信息

可以使用ps -L选项看到进程信息：

Example 2-6 thread information with ps -L

```
[root@edam ~]# ps -eLF | grep -E "LWP|/usr/sbin/httpd"
UID      PID  PPID  LWP  C NLWP  SZ RSS PSR STIME TTY          TIME CMD
root    4504    1  4504  0    1  4313 8600  2 08:33 ?
apache  4507  4504  4507  0    1  4313 4236  1 08:33 ?
apache  4508  4504  4508  0    1  4313 4228  1 08:33 ?
apache  4509  4504  4509  0    1  4313 4228  0 08:33 ?
apache  4510  4504  4510  0    1  4313 4228  3 08:33 ?

[root@edam ~]# ps -eLF | grep -E "LWP|/usr/sbin/httpd"
UID      PID  PPID  LWP  C NLWP  SZ RSS PSR STIME TTY          TIME CMD
root    4632    1  4632  0    1  3640 7772  2 08:44 ?
apache  4635  4632  4635  0    27 72795 5352  3 08:44 ?
apache  4635  4632  4638  0    27 72795 5352  1 08:44 ?
apache  4635  4632  4639  0    27 72795 5352  3 08:44 ?
apache  4635  4632  4640  0    27 72795 5352  3 08:44 ?
```

free

`free`命令显示了系统所有已用和可用内存（包括swap）量。也包括被内核使用的缓冲和缓存信息。

Example 2-7 Example output from the free command

	total	used	free	shared	buffers	cached
Mem:	1291980	998940	293040	0	89356	772016
-/+ buffers/cache:	137568	1154412				
Swap:	2040244	0	2040244			

使用`free`命令的时候，记住Linux内存架构和虚拟内存管理器的工作方式。空闲内存是受限使用的，使用swap也不表示出现了内存瓶颈。

下图展示了`free`命令的基本原理。

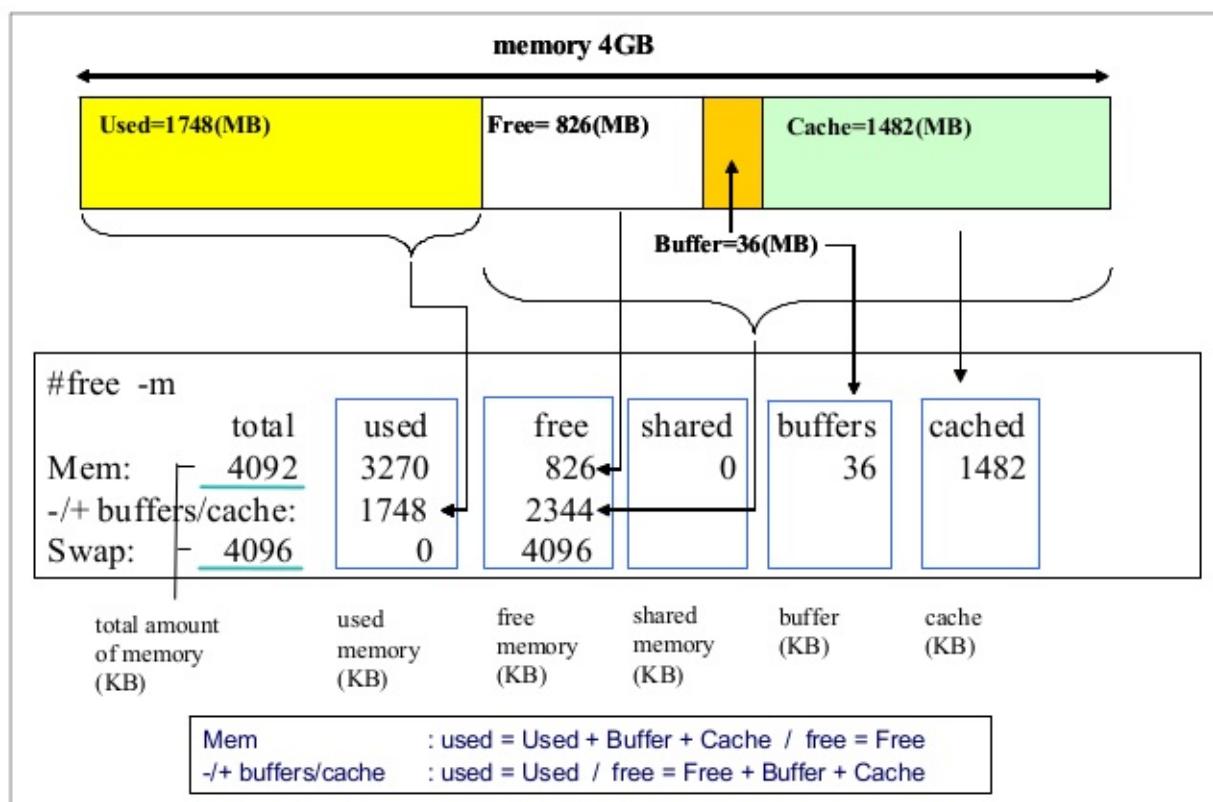


Figure 2-1 free command output

`free`命令的常用参数：

- `-b`, `-k`, `-m`, `-g` 以字节b, 千字节kb, 兆字节mb和吉字节gb为单位展示。
- `-l` 显示详细的高低内存统计
- `-c` 输出`free`的次数

Memory used in a zone

使用`-l`选项，可以看到在各个内存区域中使用的内存大小。

Example 2-8 Example output from the free command on 32 bit version kernel

```
[root@edam ~]# free -l
total        used         free      shared  buffers   cached
Mem:    4154484     2381500    1772984          0    108256  1974344
Low:    877828     199436    678392
High:  3276656    2182064   1094592
-/+ buffers/cache:    298900   3855584
Swap:    4194296          0    4194296
```

Example 2-9 Example output from the free command on 64 bit version kernel

```
[root@lnxsu4 ~]# free -l
total        used         free      shared  buffers   cached
Mem:    4037420     138508    3898912          0    10300   42060
Low:    4037420     138508    3898912
High:      0         0         0
-/+ buffers/cache:    86148   3951272
```

我们可以使用/proc/buddyinfo文件来决定每个区域中有多少个可用的内存块。每列数字意味着该列中可用的页数。在下面的例子中，在ZONE_DMA中有5块 $2^{2\text{PAGE_SIZE}}$ 可用，在ZONE_DMS32中有16块 $2^{3\text{PAGE_SIZE}}$ 可用。记住伙伴系统是如何分配内存页的。这些信息展示了内存中的分片，以及有多少页可以安全分配。

Example 2-10 Buddy system information for 64 bit system

```
[root@lnxsu5 ~]# cat /proc/buddyinfo
Node 0, zone    DMA     1     3     5     4     6     1     1     0     2     0     2
Node 0, zone  DMA32    56    14     2    16     7     3     1     7    41    42   670
Node 0, zone  Normal    0     6     3     2     1     0     1     0     0     1     0
```

iostat

iostat命令显示从系统启动依赖的平均CPU时间（和uptime类似）。它会生成服务器磁盘子系统的活动报告：CPU和磁盘设备利用情况。使用iostat找出详细的I/O瓶颈，进行性能优化，详见“找到磁盘瓶颈”一节内容。iostat是sysstat包里的一个组件。

Example 2-11 Sample output of iostat

```
Linux 2.4.21-9.0.3.EL (x232)  05/11/2004
```

avg-cpu:	%user	%nice	%sys	%idle							
	0.03	0.00	0.02	99.95							
Device:	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn						
dev2-0	0.00	0.00	0.04	203	2880						
dev8-0	0.45	2.18	2.21	166464	168268						
dev8-1	0.00	0.00	0.00	16	0						
dev8-2	0.00	0.00	0.00	8	0						
dev8-3	0.00	0.00	0.00	344	0						

CPU使用报告有4个部分：

- %user 显示CPU在用户级执行应用程序所花时间的百分比。
- %nice 显示带有nice优先级的用户级程序占用的CPU时间百分比（详见“nice，renice一节”）。
- %sys 显示显示执行系统级（内核）任务所占用CPU时间的百分比。
- %idle 显示CPU空闲的时间百分比。

设备使用报告包括如下部分：

- Device 块设备的名字
- tps 设备上的每秒传输次数（每秒的I/O请求数）。多个单I/O请求可以合成一个传输请求，因为每个传输请求的大小可以是不一样的。
- Blk_read/s,Blk_wrtn/s 每秒块读写显示了每秒从设备读或者写的数据。块也可以有不同的大小。常见的是1024,2048和4096字节，这是取决于分区大小。例如，/dev/sda1的块大小可以计算：

```
dumpe2fs -h /dev/sda1 | grep -F "Block size"
```

输出内容可能如下：

```
dumpe2fs 1.34 (25-Jul-2003)
Block size: 1024
```

- Blk_read, Blk_wrtn

指示系统启动以来读和写的总块数。

iostat有很多选项，在性能调试中最有用的是-x，它能显示扩展的统计信息。下面是输出样例。

Example 2-12 iostat -x extended statistics display

```
[root@lnxsu4 ~]# iostat -d -x sdb 1
Linux 2.6.9-42.ELsmp (lnxsu4.itso.ral.ibm.com) 03/18/2007

Device:    rrqm/s wrqm/s   r/s   w/s   rsec/s   wsec/s    rkB/s    wkB/s   avgrq-sz avgqu-sz  await  svctm %util
sdb        0.15    0.00  0.02  0.00     0.46    0.00     0.23     0.00    29.02     0.00    2.60   1.05   0.00
```

- rrqm/s, wrqm/s

每秒向设备发出的合并读写请求的数目。多个单一的读写请求可以合并为一个传输请求，因为传输请求的大小是可变的

- r/s, w/s 设备上的每秒读/写请求次数。
- rsec/s, wsec/s 设备上每秒的读/写扇区数。
- rkB/s, wkB/s 每秒从设备上读取的kb数。
- avgrq-sz 向设备发出的请求的平均大小，显示为扇区。
- avgqu-sz 向设备发出的请求的平均队列长度
- await CPU执行系统任务的时间百分比
- svctm I/O请求的平均服务时间（毫秒）。
- %util I/O请求发出到设备的时间占用CPU的百分比（设备的带宽利用率）。该值接近

100%时，设备能力几乎饱和。

在将磁盘子系统向访问模式调整时，计算平均I/O大小是否是有用的。如下命令使用iostat的-x和-d选项，用来展示我们感兴趣的磁盘：

Example 2-13 Using iostat -x -d to analyze the average I/O size

Device:	rrqm/s	wrrqm/s	r/s	w/s	rsec/s	wsec/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
dasdc	0.00	0.00	0.00	2502.97	0.00	24601.98	0.00	12300.99	9.83	142.93	57.08	0.40	100.00

上面的表格中，在kB_wrttn一列下显示了每秒写入12300.99KB数据到设备dasdc中。在w/s下显示，这些数据以2502.97次IO每秒的速度向磁盘发送。在上面的例子中，平均IO大小或者说平均请求大小显示在avgrq-sz下，是9.83块，块大小是512字节。异步写的平均I/O大小通常都很奇怪，大多数应用读写I/O都是4KB的倍数（例如，4KB，8KB，16KB，32KB等）。在上面例子的应用中，是4KB的随机写请求，但是iostat显示平均请求大小是4.915KB。这个差别是由文件系统导致的，即使执行随机写，有些I/O可以通过合并到一起，更加高效率的写入到磁盘子系统。

当文件系统使用默认异步写模式时，iostat中只有平均请求大小是显示正确的。即使应用程序执行不同大小的写请求，Linux的I/O层将会合并，并且改变平均I/O的大小。

sar

使用sar命令可以收集、展示和保存系统信息。sar命令由三个部分组成：sar，显示数据，sa1和sa2，收集和存储数据。sar工具有很多选项，请多使用man手册确认用法。sar工具是sysstat包的一部分。

因为有sa1和sa2，可以配置来收集和记录系统信息，用来以后分析。

小贴士：我们建议您尽可能的系统上运行sar，以防遇到性能问题的时候，手头可以有很详细的信息。而且，sar只消耗很少的系统资源。

在/etc/crontab中添加一行。记住，在系统上安装好sar后，cron默认每天运行sar（译者：不同的发行版可能有区别）。

Example 2-14 Example of starting automatic log reporting with cron

```
# 8am-7pm activity reports every 10 minutes during weekdays.
*/10 8-18 * * 1-5 /usr/lib/sa/sa1 600 6 &
# 7pm-8am activity reports every an hour during weekdays.
0 19-7 * * 1-5 /usr/lib/sa/sa1 &
# Activity reports every an hour on Saturday and Sunday.
0 * * * 0,6 /usr/lib/sa/sa1 &
# Daily summary prepared at 19:05
5 19 * * * /usr/lib/sa/sa2 -A &
```

sar记录的原始数据保存在/var/log/sa/下，每个文件代表月份下的每天。可以专门挑选工作日的日志记录来查看。例如，要显示21号的网络计数器，可以使用命令sar -n DEV -f sa21，然后使用管道传递给less。

Example 2-15 Displaying system statistics with sar

```
[root@linux sa]# sar -n DEV -f sa21 | less
Linux 2.6.9-5.ELsmp (linux.itso.ral.ibm.com) 04/21/2005

12:00:01 AM      IFACE    rxpck/s   txpck/s   rxbyt/s   txbyt/s   rxcmp/s   txcmp/s   rxmcst/s
12:10:01 AM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:10:01 AM      eth0     1.80      0.00    247.89      0.00      0.00      0.00      0.00
12:10:01 AM      eth1     0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

也可以使用sar命令行显示几乎实时的统计。

Example 2-16 Ad hoc CPU monitoring

```
[root@x232 root]# sar -u 3 10
Linux 2.4.21-9.0.3.EL (x232) 05/22/2004
```

02:10:40 PM	CPU	%user	%nice	%system	%idle
02:10:43 PM	all	0.00	0.00	0.00	100.00
02:10:46 PM	all	0.33	0.00	0.00	99.67
02:10:49 PM	all	0.00	0.00	0.00	100.00
02:10:52 PM	all	7.14	0.00	18.57	74.29
02:10:55 PM	all	71.43	0.00	28.57	0.00
02:10:58 PM	all	0.00	0.00	100.00	0.00
02:11:01 PM	all	0.00	0.00	0.00	0.00
02:11:04 PM	all	0.00	0.00	100.00	0.00
02:11:07 PM	all	50.00	0.00	50.00	0.00
02:11:10 PM	all	0.00	0.00	100.00	0.00
Average:	all	1.62	0.00	3.33	95.06

从收集的数据中，你可以看到详细的CPU使用信息（%nice，%user，%system，%idle），内存页，网络和I/O传输统计，进程创建、活动、块设备活动和每秒中断。

mpstat

mpstat是一个可以展示多处理器服务器上每个可用CPU活动信息的命令。所有CPU的平均活动情况也会显示出来。mpstat也是sysstat包的一部分。

mpstat工具可以全面展示系统或者CPU的统计信息。通过给mpstat传递采样频率和采样次数，可以模拟vmstat的使用。下图展示了通过mpstat -P ALL 来输出每个CPU的平均使用率。

Example 2-17 Output of mpstat command on multiprocessor system

```
[root@linux ~]# mpstat -P ALL
Linux 2.6.9-5.ELsmp (linux.itso.ral.ibm.com) 04/22/2005
```

03:19:21 PM	CPU	%user	%nice	%system	%iowait	%irq	%soft	%idle	intr/s
03:19:21 PM	all	0.03	0.00	0.34	0.06	0.02	0.08	99.47	1124.22
03:19:21 PM	0	0.03	0.00	0.33	0.03	0.04	0.15	99.43	612.12
03:19:21 PM	1	0.03	0.00	0.36	0.10	0.01	0.01	99.51	512.09

把每个处理器的统计以每秒一次的频率输出三次，使用如下命令：

```
mpstat -P ALL 1 2
```

Example 2-18 Output of mpstat command on two-way machine

```
[root@linux ~]# mpstat -P ALL 1 2
Linux 2.6.9-5.ELsmp (linux.itso.ral.ibm.com) 04/22/2005

03:31:51 PM CPU %user %nice %system %iowait %irq %soft %idle intr/s
03:31:52 PM all 0.00 0.00 0.00 0.00 0.00 0.00 100.00 1018.81
03:31:52 PM 0 0.00 0.00 0.00 0.00 0.00 0.00 100.00 991.09
03:31:52 PM 1 0.00 0.00 0.00 0.00 0.00 0.00 99.01 27.72

Average: CPU %user %nice %system %iowait %irq %soft %idle intr/s
Average: all 0.00 0.00 0.00 0.00 0.00 0.00 100.00 1031.89
Average: 0 0.00 0.00 0.00 0.00 0.00 0.00 100.00 795.68
Average: 1 0.00 0.00 0.00 0.00 0.00 0.00 99.67 236.54
```

要获得mpstat的完整语法，使用：

```
mpstat -?
```

numastat

在企业数据中心，非统一内存架构（Non-Uniform Memory Architecture，NUMA）已经变成主流，例如IBM System x3950，然而，NUMA系统给调优带来了新的挑战。在NUMA出现之前，我们从来不需要关心内存的位置。幸好，企业Linux发行版为监测NUMA架构行为提供了工具。`numastat`命令提供本地和远程内存使用率和所有节点的整体内存配置。本地内存分配失败的信息在`numa_miss`一行展示，远程内存（shower memory）分配信息在`numa_foreign`一行展示。过度的使用远程内存会增加风险，可能导致整体性能下降。把进程绑定映射本地内存的节点会增加性能。

Example 2-19 Sample output of the numastat command

```
[root@linux ~]# numastat
```

	node1	node0
<code>numa_hit</code>	76557759	92126519
<code>numa_miss</code>	30772308	30827638
<code>numa_foreign</code>	30827638	30772308
<code>interleave_hit</code>	106507	103832
<code>local_node</code>	76502227	92086995
<code>other_node</code>	30827840	30867162

pmap

`pmap`命令会展示一个或多个进程正在使用的内存量。使用这一工具，你可以确定服务器上的哪一个进程正在分配内存，还有是否这部分内存导致了内存瓶颈。更多信息，使用`pmap -d`选项。

```
pmap -d <pid>
```

Example 2-20 Process memory information the init process is using

```
[root@lnxsu4 ~]# pmap -d 1
1: init [3]
Address          Kbytes Mode Offset           Device Mapping
0000000000400000      36 r-x-- 0000000000000000 0fd:00000 init
0000000000508000       8 rw--- 0000000000008000 0fd:00000 init
000000000050a000     132 rwx-- 000000000050a000 000:00000 [ anon ]
0000002a95556000       4 rw--- 0000002a95556000 000:00000 [ anon ]
0000002a95574000       8 rw--- 0000002a95574000 000:00000 [ anon ]
00000030c3000000      84 r-x-- 0000000000000000 0fd:00000 ld-2.3.4.so
00000030c3114000       8 rw--- 00000000000014000 0fd:00000 ld-2.3.4.so
00000030c3200000    1196 r-x-- 0000000000000000 0fd:00000 libc-2.3.4.so
00000030c332b000    1024 ----- 000000000012b000 0fd:00000 libc-2.3.4.so
00000030c342b000       8 r---- 000000000012b000 0fd:00000 libc-2.3.4.so
00000030c342d000      12 rw--- 000000000012d000 0fd:00000 libc-2.3.4.so
00000030c3430000      16 rw--- 00000030c3430000 000:00000 [ anon ]
00000030c3700000      56 r-x-- 0000000000000000 0fd:00000 libsepol.so.1
00000030c370e000    1020 ----- 0000000000e000 0fd:00000 libsepol.so.1
00000030c380d000       4 rw--- 0000000000d000 0fd:00000 libsepol.so.1
00000030c380e000      32 rw--- 00000030c380e000 000:00000 [ anon ]
00000030c4500000      56 r-x-- 0000000000000000 0fd:00000 libselinux.so.1
00000030c450e000    1024 ----- 0000000000e000 0fd:00000 libselinux.so.1
00000030c460e000       4 rw--- 0000000000e000 0fd:00000 libselinux.so.1
00000030c460f000       4 rw--- 00000030c460f000 000:00000 [ anon ]
0000007fbfffc000      16 rw--- 0000007fbfffc000 000:00000 [ stack ]
ffffffffff600000  8192 ----- 0000000000000000 000:00000 [ anon ]
mapped: 12944K   writeable/private: 248K   shared: OK
```

最后一行显示的信息最为有用：

- mapped 该进程映射到文件的内存量。
- writable/private 该进程使用的私有地址空间。
- shared 该进程和其它进程共享的地址空间量。

你也可以查看存储信息的地址空间。在32位和64位系统上，pmap有些有趣的差别。使用如下命令查看完整的pmap语法：

```
pmap -?
```

netstat

netstat 是最常用的工具之一，如果你从事网络工作，你应该对这个命令很熟悉。它会展示网络相关的信息，例如socket使用，路由，接口，协议和其它网络统计。有如下的基础选项：

- -a 显示所有的socket信息

- -r 显示路由信息
- -i 显示网络接口统计
- -s 显示网络协议统计

还有很多其它有用的选，请查阅man手册。如下的例子中展示了socket信息的样例。

Example 2-21 Showing socket information with netstat

Active Internet connections (servers and established)					
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:111	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:25	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:2207	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:36285	127.0.0.1:12865	TIME_WAIT
tcp	0	0	10.0.0.5:37322	10.0.0.4:33932	TIME_WAIT
tcp	0	1	10.0.0.5:55351	10.0.0.4:33932	SYN_SENT
tcp	0	1	10.0.0.5:55350	10.0.0.4:33932	LAST_ACK
tcp	0	0	10.0.0.5:64093	10.0.0.4:33932	TIME_WAIT
tcp	0	0	10.0.0.5:35122	10.0.0.4:12865	ESTABLISHED
tcp	0	0	10.0.0.5:17318	10.0.0.4:33932	TIME_WAIT
tcp	0	0	:::22	:::*	LISTEN
tcp	0	2056	::ffff:192.168.0.254:22	::ffff:192.168.0.1:3020	ESTABLISHED
udp	0	0	0.0.0.0:111	0.0.0.0:*	
udp	0	0	0.0.0.0:631	0.0.0.0:*	
udp	0	0	:::5353	:::*	

Socket信息解释：

- Proto socket使用的协议（tcp，udp，raw）。
- Recv-Q 表示收到的数据已经在本地接收缓冲，但是还有多少没有被进程取走，单位是字节。
- Send-Q 对方没有收到的数据或者说没有Ack的，还是本地缓冲区，单位字节。
- Local Address socket的本地地址和端口。除非使用--numeric(-n)选项，socket地址会被解释成主机名（FQDN），端口号会被转成相应的服务名字。
- Foreign Address 远端socket的端口和地址。
- State socket的状态。因为raw和UDP通常是没有状态的，所以这列可能是空白。

iptraf

iptraf监控和展示TCP/IP的实时流量。它可以根据各个session、接口、协议展示TCP/IP流量统计。iptraf组件是由iptraf包提供。

iptraf给我们展示如下的报告：

- IP流量监控：通过TCP连接的网络流量统计
- 接口一般统计：网络接口流量统计
- 接口详细统计：根据端口的网络流量统计
- 统计分析：根据TCP/UDP端口和包大小的网络流量统计。
- 局域网统计：根据网络2层地址的网络流量统计。

下面是使用iptraf收集的几个报告。

Figure 2-2 iptraf output of TCP/IP statistics by protocol

Figure 2-3 iptraf output of TCP/IP traffic statistics by packet size

tcmpdump / ethereal

`tcpdump`和`ethereal`通常用来抓取和分析网络流量。这两个工具都会用到`libpcap`库来抓取包。在混杂模式下，它们会监控网卡上的所有流量，并且抓取所有网卡上收到的分片。为了抓取所有包，这些命令应该使用超级用户权限执行，以便开启网卡混杂模式。

你可以使用这些工具来找到和网络相关的问题。可以发现TCP/IP重传，滑动窗口大小变化，名字解析问题、网络错误配置等。记住，这些工具只能监控所有到达网卡的分片，而不是所有的网络流量。

tcpdump

tcpdump是一个简单和强大的工具。它拥有基本的协议分析能力，可以获得网络上的大体情况。tcpdump可以使用很多选项和扩展表达式来过滤要抓取的包。入门可以看看如下的几个选项：

- -i 指定网络接口
- -e 打印数据链路层头
- -s 抓取每个包的字节
- -n 避免DNS解析
- -w 写入文件
- -r 从文件读取
- -v, -vv, -vvv 详细输出

抓取过滤器的表达式：

- 关键字：
源目主机，源目端口，tcp，udp，icmp，源目网络等等
- 联合逻辑使用
非 ('!'或者'not')
与 ('&&'或者'and')
或 ('||'或者'or')

十分有用的过滤表达式样例：

- DNS查询包

```
tcpdump -i eth0 'udp port 53'
```

- 目的主机为192.168.1.10的FTP传输和FTP数据会话

```
tcpdump -i eth0 'dst 192.168.1.10 and (port ftp or ftp-data)'
```

- 目标为192.168.2.253的HTTP会话

```
tcmpdump -ni eth0 'dst 192.168.2.253 and tcp and port 80'
```

- 到192.168.2.0/24子网的telnet会话（译者：例子中貌似是ssh会话）

```
tcmpdump -ni eth0 'dst net 192.168.2.0/24 and tcp and port 22'
```

- 抓取源目地址都不在192.168.1.0/24子网，并且带有TCP SYN或者TCP FIN标志（建立或者中断TCP连接）的数据。

```
tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin)!=0 and not src and dst net 192.168.1.
0/24'
```

Example 2-22 Example of tcpdump output

```
21:11:49.555340 10.1.1.2542 > 66.218.71.102.http: S 2657782764:2657782764(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
21:11:49.671811 66.218.71.102.http > 10.1.1.2542: S 2174620199:2174620199(0) ack 2657782765 win 65535 <mss 1380>
21:11:51.211869 10.1.1.18.2543 > 216.239.57.99.http: S 2658253720:2658253720(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
21:11:51.332371 216.239.57.99.http > 10.1.1.1.2543: S 3685788750:3685788750(0) ack 2658253721 win 8190 <mss 1380>
21:11:56.972822 10.1.1.1.2545 > 129.42.18.99.http: S 2659714798:2659714798(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
21:11:57.133615 129.42.18.99.http > 10.1.1.1.2545: S 2767811014:2767811014(0) ack 2659714799 win 65535 <mss 1348>
21:11:57.656919 10.1.1.1.2546 > 129.42.18.99.http: S 2659939433:2659939433(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
21:11:57.818058 129.42.18.99.http > 9.116.198.48.2546: S 1261124983:1261124983(0) ack 2659939434 win 65535 <mss 1348>
```

通过man手册可以获取更多tcpdump的用法。

ethereal

ethereal有和tcpdump相似的功能，但是更加复杂，并且拥有更高级的协议分析和报告能力。它还拥有一个GUI接口和ethereal命令行界面。

和tcpdump类似，ethereal也可以使用过滤抓取，从而缩小抓取分片的范围。如下是一些常用的表达式。

- IP

```
ip.version == 6 and ip.len > 1450
ip.addr == 129.111.0.0/16
ip.dst eq www.example.com and ip.src == 192.168.1.1
not ip.addr eq 192.168.4.1
```

- TCP/UDP

```
tcp.port eq 22
tcp.port == 80 and ip.src == 192.168.2.1
tcp.dstport == 80 and (tcp.flags.syn == 1 or tcp.flags.fin == 1)
tcp.srcport == 80 and (tcp.flags.syn == 1 and tcp.flags.ack == 1)
tcp.dstport == 80 and tcp.flags == 0x21
```

- 应用层

```
http.request.method == "POST"
smb.path contains \\SERVER\SHARE
```

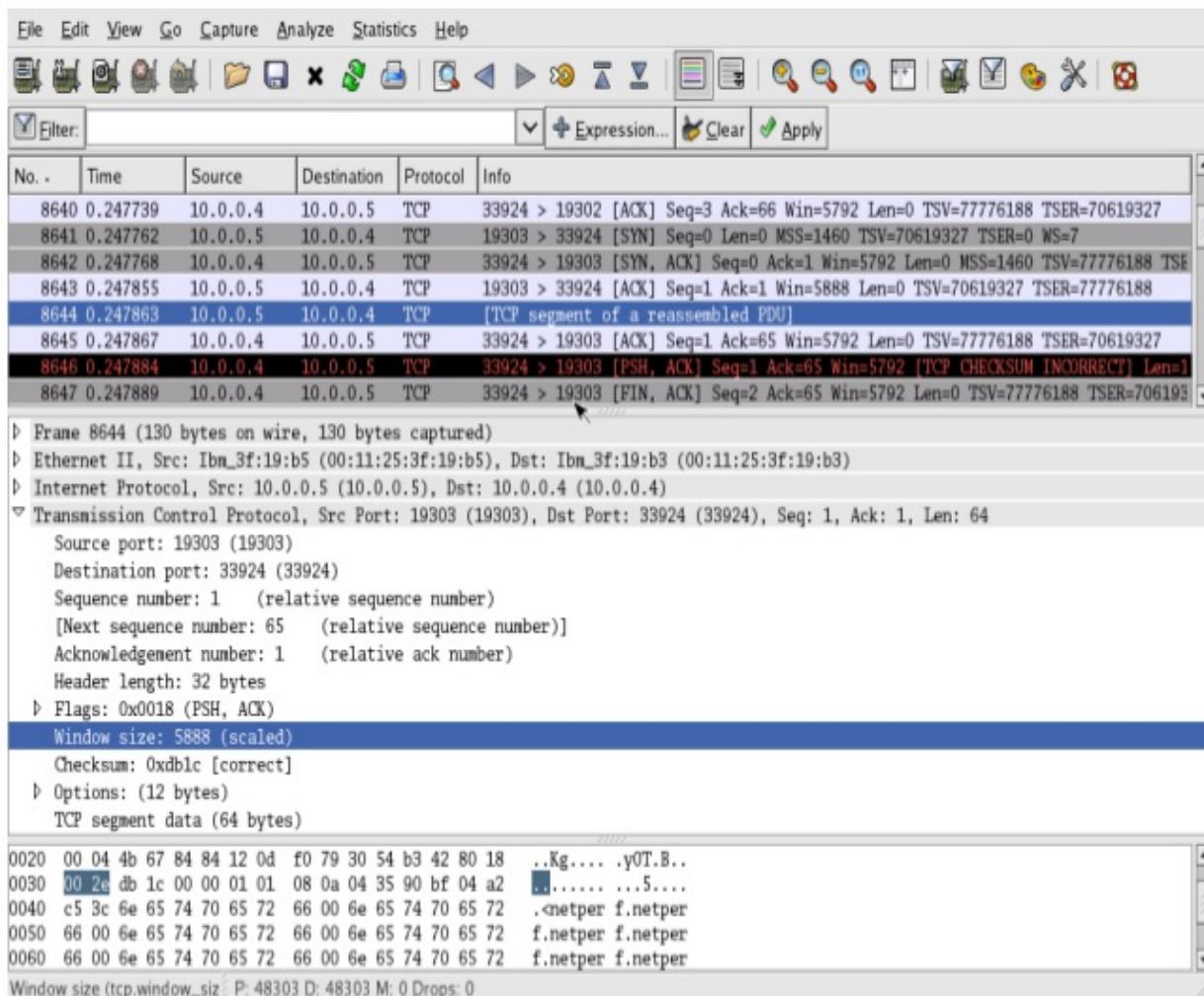


Figure 2-4 ethereal GUI

nmon

Nigel's Monitor简称nmon，是由Nigel Griffiths开发的监控Linux系统性能的常用工具。由于nmon能监控多个子系统的性能信息，所以，可以把它作为性能监控的唯一工具。通过nmon可以获取的信息有：处理器利用率、内存利用率、运行队列信息、磁盘I/O统计和网络I/O统计，页活动信息和进程指标。

运行nmon，只需要通过输入感兴趣的子系统的首字母，获取相关信息。例如，获得CPU，内存、磁盘统计，首先运行nmon，然后输入c m d。

nmon的一个很有用的特性是能够使用逗号分割的CSV文件保存性能统计，以便以后观察。

nmon输出的CSV文件可以导入电子表格应用中，生成可视化图形报告，要使用该功能，启动nmon的时候需要带上-f选项。例如使用如下命令，让nmon生成30秒钟为频率，总时长1小时的报告。

```
# nmon -f -s 30 -c 120
```

上面的命令将把统计生成的文本文件存储在当前目录下，名字格式为`_date_time.nmon`。更多信息，参考：

[<http://www-941.haw.ibm.com/collaboration/wiki/display/WikiPtype/nmon>(<http://www-941.haw.ibm.com/collaboration/wiki/display/WikiPtype/nmon>)

strace

`strace`命令会拦截和记录进程的系统调用或进程接收到的信号。这是一个有用的诊断、教学和调试工具。它在解决程序遇到的问题方面很有价值。

使用时，需要指定要监控的进程ID：

```
strace -p <pid>
```

下图是`strace`的输出实例：

Example 2-24 Output of strace monitoring httpd process

```
[root@x232 html]# strace -p 815
Process 815 attached - interrupt to quit
semop(360449, {0xb73146b8, 1})           = 0
poll([{fd=4, events=POLLIN}, {fd=3, events=POLLIN, revents=POLLIN}], 2, -1) = 1
accept(3, {sa_family=AF_INET, sin_port=htons(52534), sin_addr=inet_addr("192.168.1.1")}, [16]) = 13
semop(360449, {0xb73146be, 1})           = 0
getsockname(13, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("192.168.1.2")}, [16]) = 0
fcntl64(13, F_GETFL)                      = 0x2 (flags O_RDWR)
fcntl64(13, F_SETFL, O_RDWR|O_NONBLOCK)   = 0
read(13, 0x8259bc8, 8000)                 = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=13, events=POLLIN, revents=POLLIN}], 1, 300000) = 1
read(13, "GET /index.html HTTP/1.0\r\nUser-Agent: ..., 8000) = 91
gettimeofday({1084564126, 750439}, NULL) = 0
stat64("/var/www/html/index.html", {st_mode=S_IFREG|0644, st_size=152, ...}) = 0
open("/var/www/html/index.html", O_RDONLY) = 14
mmap2(NULL, 152, PROT_READ, MAP_SHARED, 14, 0) = 0xb7052000
writev(13, [{"HTTP/1.1 200 OK\r\nDate: Fri, 14 Mar 2014 264}, {"<html>\n<title>\n      RedPaper Per..., 152}], 2) = 416
mmap(0xb7052000, 152)                     = 0
socket(PF_UNIX, SOCK_STREAM, 0)             = 15
connect(15, {sa_family=AF_UNIX, path="/var/run/.nscd_socket"}, 110) = -1 ENOENT (No such file or directory)
close(15)                                  = 0
```

注意：当对一个进程执行`strace`命令时，该进程的性能急剧下降。

`strace`还有一个有趣的用法，下面这个命令会展示，在执行一个命令时，每个系统调用在内核中所用的时间。

```
strace -c <command>
```

Example 2-25 Output of strace counting for system time

```
[root@lnxsu4 ~]# strace -c find /etc -name httpd.conf
/etc/httpd/conf/httpd.conf
Process 3563 detached
% time      seconds   usecs/call     calls    errors syscall
----- -----
25.12    0.026714       12     2203      getdents64
25.09    0.026689        8     3302      lstat64
17.20    0.018296        8     2199      chdir
 9.05    0.009623        9     1109      open
 8.06    0.008577        8     1108      close
 7.50    0.007979        7     1108      fstat64
 7.36    0.007829        7     1100      fcntl64
 0.19    0.000205      205       1      execve
 0.13    0.000143       24       6      read
 0.08    0.000084       11       8      old_mmap
 0.05    0.000048       10       5      mmap2
 0.04    0.000040       13       3      munmap
 0.03    0.000035       35       1      write
 0.02    0.000024       12       2      1 access
 0.02    0.000020       10       2      mprotect
 0.02    0.000019        6       3      brk
 0.01    0.000014        7       2      fchdir
 0.01    0.000009        9       1      time
 0.01    0.000007        7       1      uname
 0.01    0.000007        7       1      set_thread_area
-----
100.00   0.106362      12165      1 total
```

如下命令获取strace的完整语法：

```
strace -?
```

Proc文件系统

proc文件系统不是真实的文件系统，但是它真的十分有用。它不是存储数据的；而是提供运行内核的监控和操作接口。proc文件系统让管理员可以监控和修改运行中的内核。下图展示了一个简单的proc文件系统。大多数Linux性能工具都要依赖于/proc提供的信息。

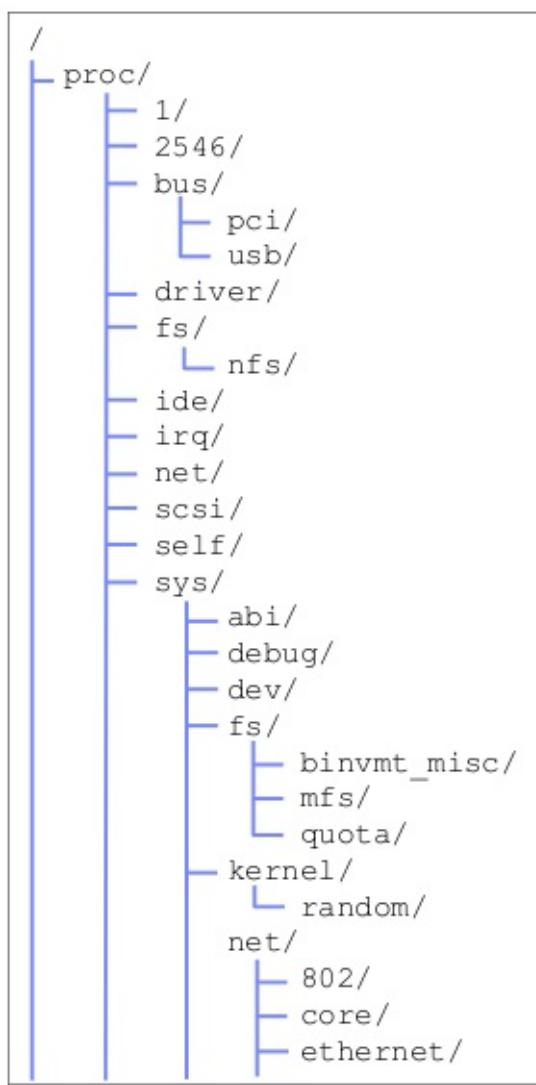


Figure 2-5 A sample /proc file system

观察这个`proc`文件系统，我们可以分辨出各个子目录的用途，但是由于`proc`目录下的大多数信息对人类来说比较难以理解，所以推荐使用类似于`vmstat`这类工具，以更高可读的方式展示统计数据。注意，在不同的系统架构中，`proc`文件系统下的信息和布局有存在差异。

- `/proc`目录下的文件
`/proc`根目录下的各种文件里面包含相关系统的统计。你可以找到Linux工具使用的信息源，例如`vmstat`和`cpuinfo`文件。
- 数字1到X
各个数字的子目录指向的是运行进程或者它们的进程ID（PID）。目录结构总是已PID 1开始，指向的是`init`进程，然后是系统上运行的各个PID。每个数字子目录下保存进程相关的统计信息。例如进程映射的虚拟内存。
- acpi
ACPI意思是高级配置与电源接口（advanced configuration and power interface），，受到大多数现代桌面和笔记本系统支持。由于ACPI主要是PC技术，所以在服务器上通常是禁用状态。ACPI的更多信息，查看
<http://www.apci.info>
- 总线（bus）

这个子目录包含总线子系统的信息，例如PCI总线或者系统USB接口。

- **irq**

irq目录包含系统中断的信息。这个目录下的每个子目录代表一次中断，也可能是一个附加设备，例如网卡。在**irq**子目录下，你可以修改一个给定中断的CPU关联（**affinity**）。

- **net**

网络子目录下包含网络接口的原始统计数据，例如收到的多播包或接口的路由。

- **scsi**

scsi子目录包含系统上关于SCSI子系统的信息，例如附加设备或者驱动调整。

- **sys**

在**sys**子目录下，是可调整的内核参数，例如虚拟内存管理器或者是网络栈的行为。**/proc/sys**这部分内容和可调优的值在“修改内核参数”一节中会详细讲解。

- **tty**

虚拟终端和附加的物理设备信息都包含在**tty**子目录中

KDE System Guard

KDE System Guard（KSysguard）是一个KDE任务管理器和性能监控器。它使用C/S（client/Server）结构，可以监控本地和远程主机。

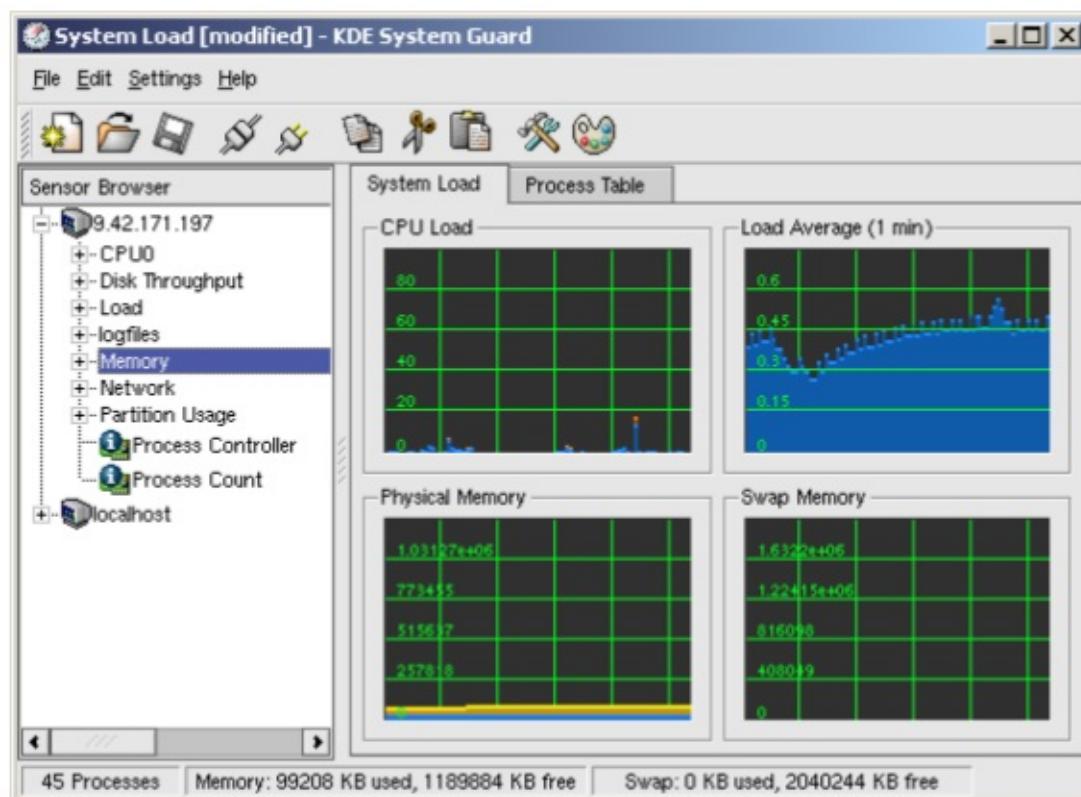


Figure 2-6 Default KDE System Guard window

前段图像系统使用传感器取回要展示的信息。传感器可以返回简单的数值，也可以返回更加复杂的信息，比如一张表。对各类信息，提供一种或者多种展示，展示通过图表显示出来，可以各自独立的加载和保存。

KSysguard主窗口包含一个菜单，一个可选工具栏，一个状态栏，传感器浏览器和工作区。第一次启动时，可以看到默认的设置：本机在传感器浏览器的localhost中，工作区中还有两个标签页。

每个传感器都监视一个特定的系统值，所有的传感器都可以拖放到工作区中。有三个选项：

- 可以替换和删除工作区中的传感器
- 可以编辑工作表属性，增加行和列数
- 可以创建新的工作表，根据需求拖拽传感器到里面

工作区

下图的工作区有两个标签页：

- 系统负载，是第一次启动Ksysguard的默认视图
- 进程表

![KDE system guard传感器浏览器](kde-system-guard-sensor-browser.png)

系统负载

系统负载工作表中展示死歌传感器窗口：CPU负载，平均负载（1分钟），物理内存和Swap。一个窗口中可以展示多个传感器。要查看一个窗口中监视的传感器，可以把鼠标移到图像上，就会出现相关描述。你也可以在图像上使用右键单击，然后选择属性，然后点击传感器标签，如下图。这地方展示了图像中每个颜色代表的项目。

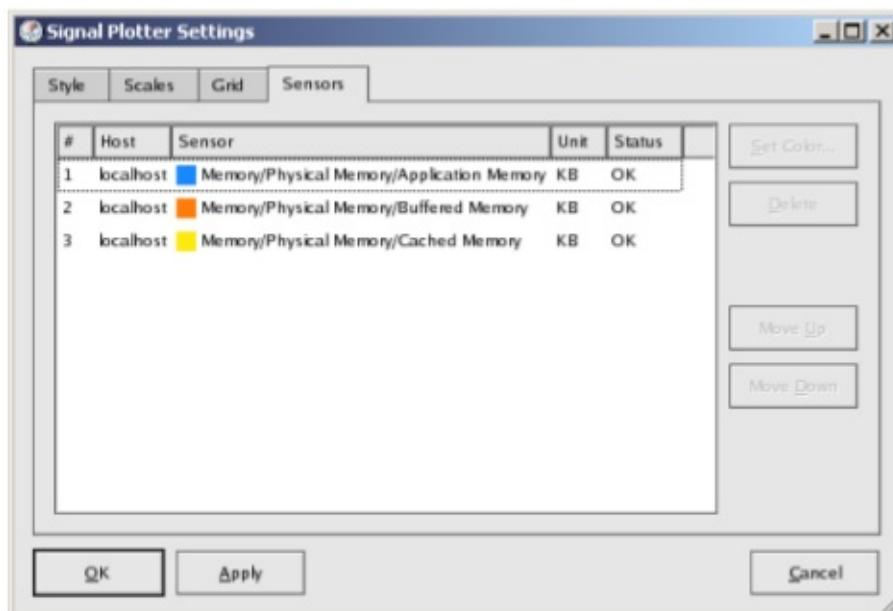


Figure 2-8 Sensor Information, Physical Memory Signal Plotter

进程表

点击进程表标签页，展示服务器上运行的所有进程信息。默认情况下，这张表以系统CPU使用率排序，也可以通过点击其它栏目，选择排序根据。

配置工作表

对于你需要的特定环境的监控，就要用到不同的传感器。最好的办法是创建一个定制的工作表。这里将演示如何一步步创建最终的工作表。

- 1. 通过点击 文件->新建打开一个如下窗口

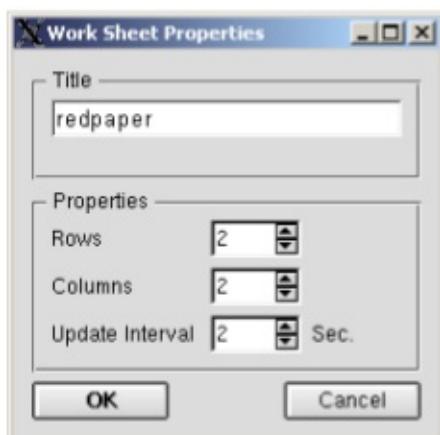


Figure 2-10 Properties for new work sheet

- 1. 输入一个标题和行列数，这就是指定的最大窗口数目，演示中是4个窗口。完成上面的信息之后，单击确定，创建空白的工作表，如下图：

注意，更新的最快频率是2秒钟一次

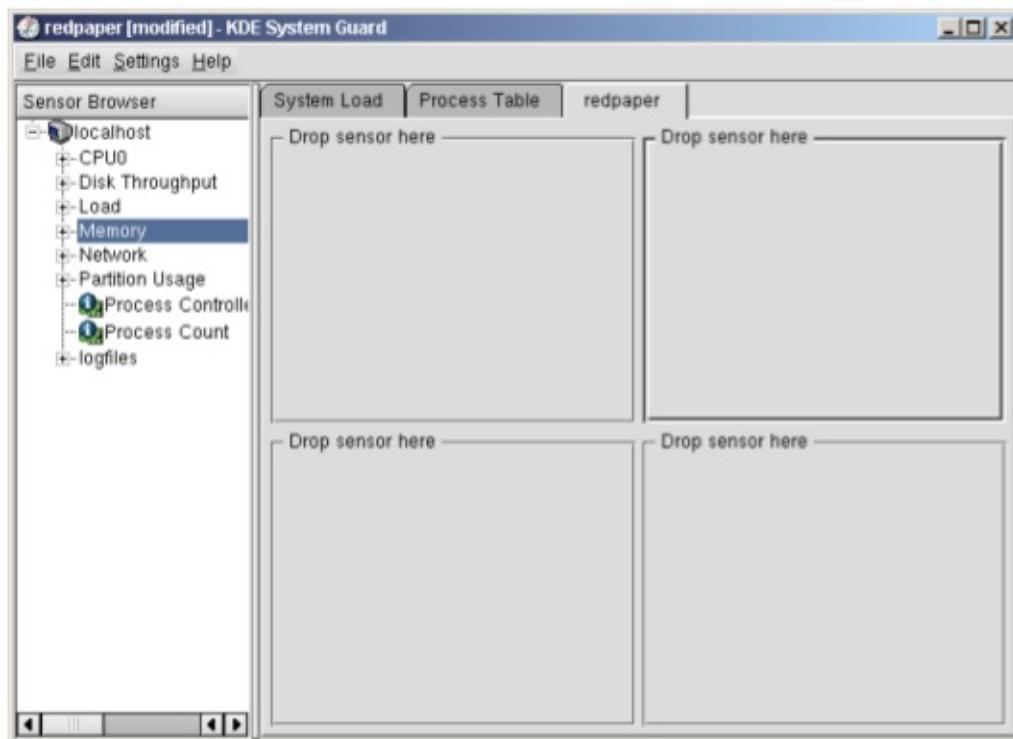


Figure 2-11 Empty work sheet

- 1. 拖拽左边窗口的传感器到右边的空格子中，显示类型如下：

■ 信号图：一次展示一个或多个传感器。如果展示多个传感器，值以不同的颜色表示。如果显示足够大，格子会显示采样的范围。

默认情况下，会开启自动范围模式，最小值和最大值会自动设置。如果你想修改最小和最大值，你可以取消自动模式，并且在属性窗口的范围标签页（在图

像中右键进入) 中设置。

- **Multimeter**：把传感器的值作为数字度量，在属性窗口中，可以指定最大和最小限制。如果超出范围，会使用警告颜色显示。
- **直方图**：使用**dancing bar**展示传感器的值，在属性窗口中，可以指定最大和最小值，以及一个最大和最小的限制。如果超出限制，以警告颜色展示。
- **传感器记录仪**：这个不展示任何值，而是带上日期和时间信息记录到文件中。对每个传感器来说，你要定义目标日志文件，传感器记录的时间间隔，以及是否开启警报。

- 1. 单击文件->保存，保存工作区表的修改。

在保存一个工作表的时候，它会保存到用户的家目录下，这会导致其它人不能使用你自定义的工作表

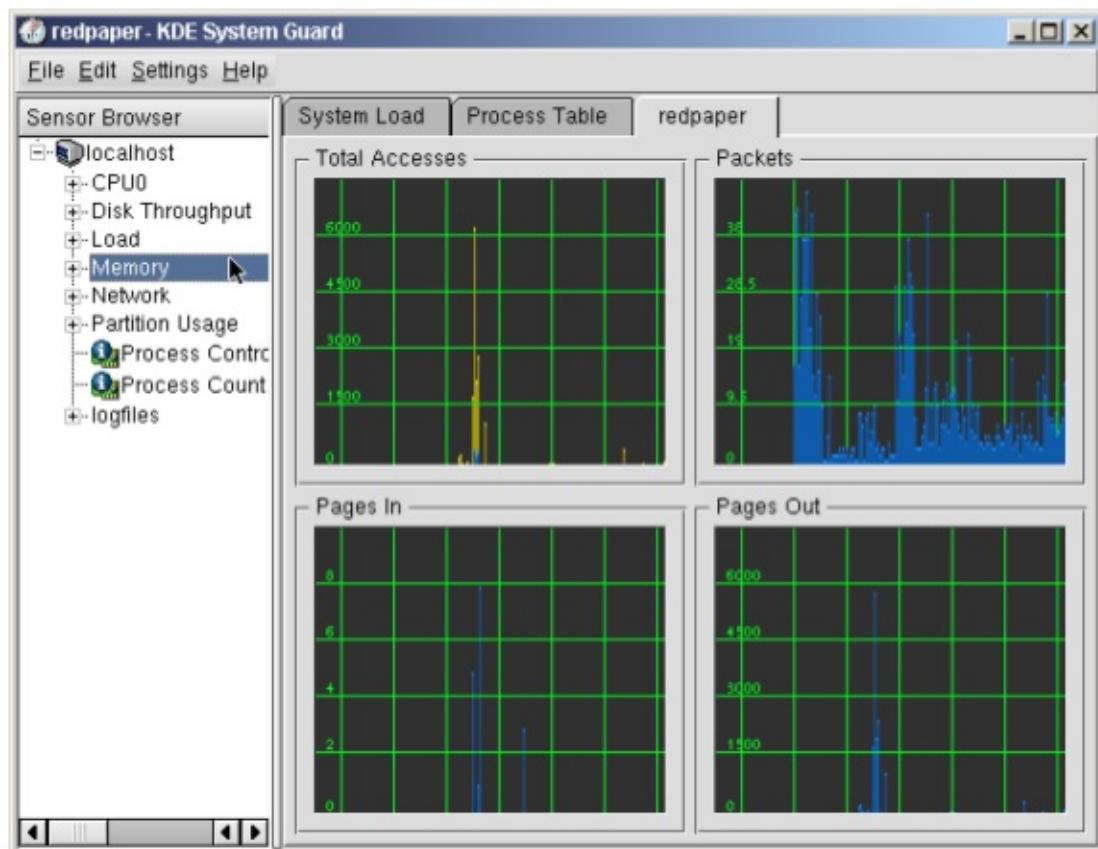


Figure 2-12 Example work sheet

获取更多关于KDE System Guard的信息：

<http://docs.kde.org/>

Gnome System Monitor

虽然没有KDE System Guard强大，桌面环境提供了一个图形性能分析工具。Gnome System Monitor会把性能相关的资源图形化展示，把可能的瓶颈可视化。注意，所有的统计数据，都

是实时的。长时期的性能分析应该使用其它工具。

性能管理器

Capacity Manager，是IBM System下附加于IBM Director系统管理套件，包含在IBM System x系统的Server Plus包中。Capacity Manager提供长期的性能管理，横跨多个系统和平台。Capacity Manager可以使用容量计划，帮你估计未来的容量需求。你可以把报告导出为HTML、XML和GIF文件，并且自动保存到web服务器上。IBM Director可以用在不同的操作系统平台，所以Capacity可以很容易的收集和分析各个环境的数据。Capacity Manager将会在“调优IBM System x Server性能”一节中详细讨论。

要使用Capacity Manager和它的先进功能，首先你要系统上安装相应的RPM包。安装完RPM之后，在IBM Director Console中选择Capacity Manager->Monitor Activator。

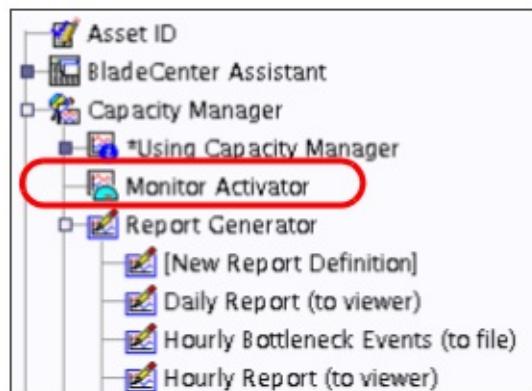


Figure 2-13 The task list in the IBM Director Console

拖放Monitor Activator中一个或多个图标，将会打开窗口让你选择监控哪个子系统。Capacity Manager for Linux还不支持全部功能。系统统计只能包含性能参数的基本子集。

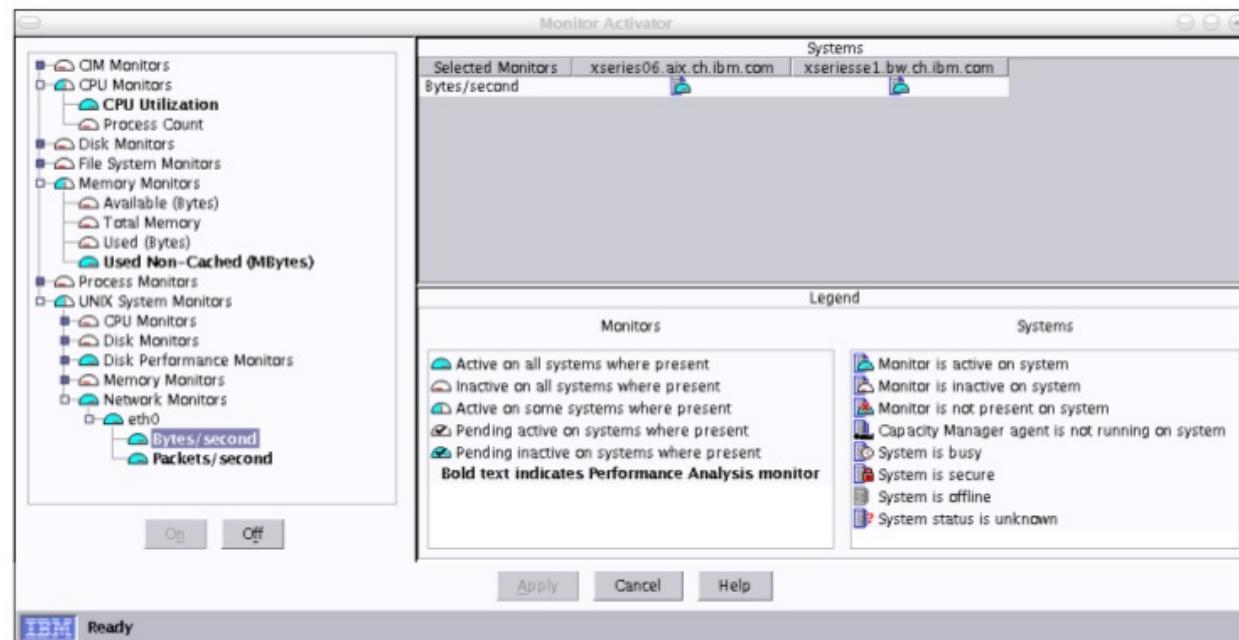


Figure 2-14 Activating performance monitors multiple systems

Monitor Activator窗口在右边展示了各自系统的当前状态，在左边是各类可用的系统监控。要

添加一个新的监视器，选择相应的，然后点击On，在关闭Monitor Activator之后不久，修改就会生效。在上面的一步之后，IBM Director开始收集请求的性能值，并且保存在各个系统上的临时存放点。

创建收集数据的报告，选择Capacity Manager->Report Generator，把他拖到你想要观察性能统计的，单个机器或者一组机器里。IBM Director会问你是直接开始收集，还是稍后执行。

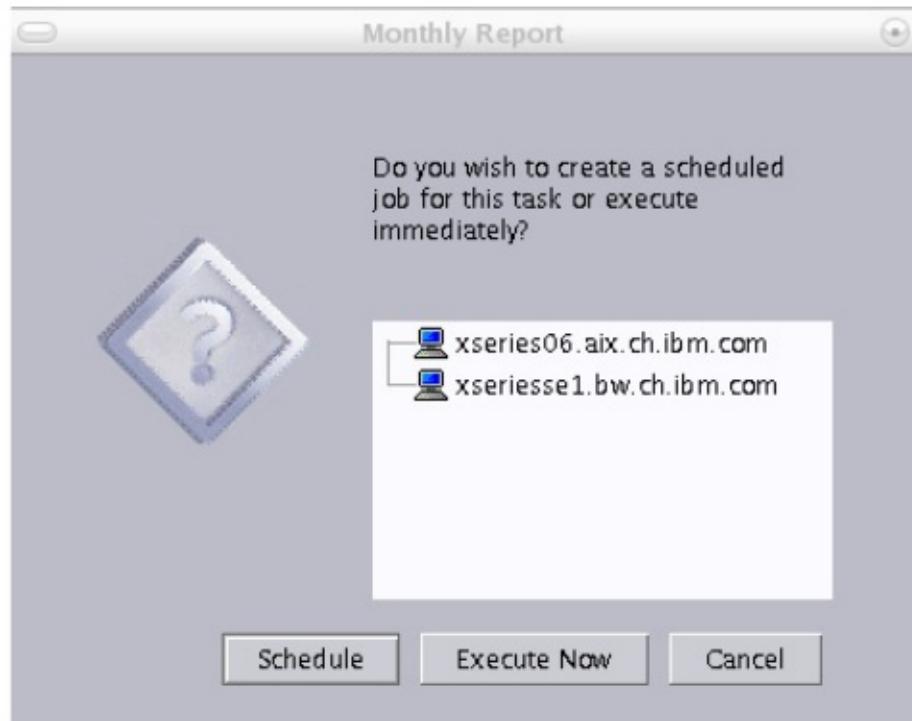


Figure 2-15 Scheduling reports

在生成环境中，最好让Capacity Manager定期生成报告。经验告诉我们，在周末生成每周报告是很有用的。你可以选择直接生成，或者按计划生成。一旦报告完成，就会保存在中心IBM Director管理服务器上，然后可以使用Report Viewer来查看。下面是Capacity Manager的报告样例。

2.3. 监控工具

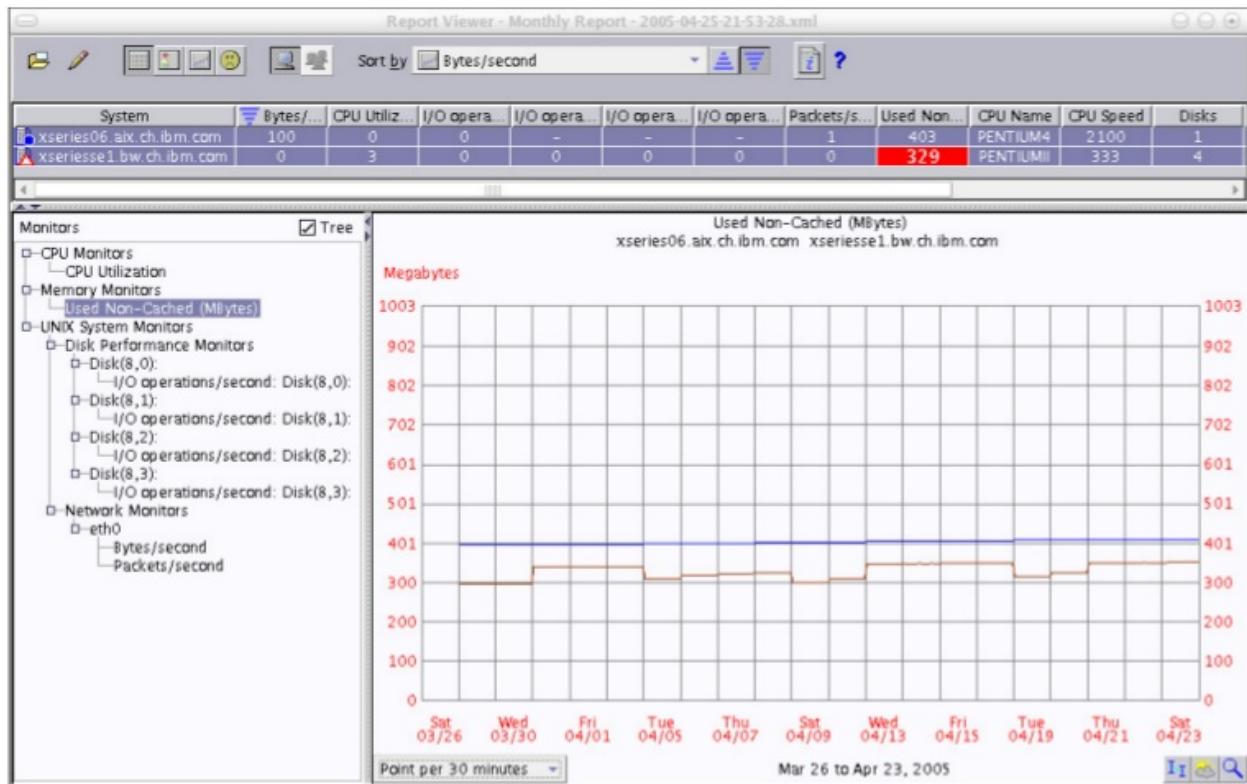


Figure 2-16 A sample Capacity Manager report

Report Viewer窗口让你可以选择不同的性能收集数据，并且关联到一个或多个选择的系统上。

通过Capacity Manager获得的数据可以以HTML或XML的格式导出到Web服务器上，或者留待以后分析。

压测工具

- 压测工具
 - [LMbench](#)
 - [iozone](#)
 - [netperf](#)
 - 其它有用工具

在这里，我们会讨论主要的压力测试工具，选择合适的压测工具，才能准确衡量系统性能。好的压测工具有很多，可能的功能如下：

- 发起压力
- 监控性能
- 监控系统使用率
- 生成报告

基准测试无非就是看系统能力是否达到对系统预期的能力。如果一款系统拥有较高的 Linpack 分数，但它未必是理想的文件服务器。需要记住的是，基准测试不可能模拟出终端用户的行为，因为他们是不可预测的。基准测试不会告诉你，在用户访问他们文件的时候或者备份开始的时候，服务器会做什么反应。一般来说，基准测试应该遵循如下几个原则：

- 为服务器负载做基准测试：服务器系统和典型的PC完全不同。为了利用系统的SMP能力和模拟一个真实的多用户环境，服务器测试使用多线程。PC打开一个网页浏览器的速度可能比高端服务器更快，但是服务器打开1000个网页浏览器的速度会比PC快很多。
- 模拟预期的负载：所有的基准测试都有不同的配置选项，我们应该朝着未来预期运行的负载方向定制测试。如果应用需要低延迟的磁盘，那么强大的CPU性能也没什么大的用处。
- 隔离测试系统：如果要对系统进行基准测试，首先要把它尽可能的从其它负载中隔离。在系统上运行一个top命令也会极大的影响基准测试结果。
- 结果平均：即使你试着隔离测试系统，可能还会有未知的因素在压测时影响系统性能。经验表明，至少执行三次测试，然后将结果平均，以保证偶然因素不会影响全面分析。

下面是基于以上标准选择的工具：

- 在Linux上测试：Linux是基准测试的目标
- 能运行在所有的硬件平台上测试：由于IBM提供3种不同的硬件平台（把IBM System p和IBM System i看做都是基于IBM Power架构），所以选择不用做太多修改就能在所有架构上使用的工具很重要。
- 开源：Linux运行在很多平台上，如果需要到处运行，就可能需要源代码。

- 文档全面：必须了解你用来做性能测试的工具，所以你需要文档。在使用某工具之前，先了解它的理念、设计和细节，以确定它是否真的合适。
- 维护中的：如果使用已经停止维护的软件，可能无法跟上最新的技术和标准，导致错误的结果。
- 广泛使用的：你很方便找到相关信息。
- 使用简单的：简单更好用。
- 能产生报告：具有报告功能能极大的减轻性能分析工作。

lmbench

LMbench是一套微测试工具，用来分析不同操作系统设置，比如SELinux是否启用。

LMbench可以用来测试操作系统的各个方面，如上下文切换、本地通信、内存带宽、文件操作等。LMbench使用起来很简单，仅有三个重要命令：
 + make results：第一次启动LMbench时，它会提示一些操作系统的各种信息，还有要执行哪些测试。
 + make return：在完成初始化配置和第一次运行LMbench之后，使用make return命令，重复make results时的配置进行测试。
 + make see：在至少三次运行之后，可以使用make see命令查看结果。可以使用图表的方式展示和分析。LMbench项目地址是：[\[http://sourceforge.net/projects/lmbench/\]](http://sourceforge.net/projects/lmbench/)
[\(http://sourceforge.net/projects/lmbench/\)](http://sourceforge.net/projects/lmbench/)

iozone

IOzone是一个文件系统测试工具，可以用它模拟各种磁盘访问方式。由于IOzone的配置很详细，所以可以用来精确的模拟目标文件系统的负载。IOzone可以使用不同的块大小写入各种大小的文件。

IOzone有一种非常好用的自动测试模式，方便测试者自定义各种测试行为，如文件大小、I/O大小和访问方式等等。如果文件系统是为数据库服务做准备的，应该重点测试大block size的大文件随机访问，而不是小block size的大文件。IOzone重要的选项如下：

- -b 让IOzone以微软Excel兼容格式保存成电子表格。
- -C 显示每个子进程的输出（用来检查是否每个子进程都在运行模拟程序）
- -f 告诉IOzone写入文件的地方
- -i 使用这个选项指定要运行什么测试。使用-i 0 指定第一时间写入到测试文件。使用-i 1是流式读，-i 8是混合随机访问负载。
- -h 显示帮助
- -r 指定测试类型和I/O大小。测试文件大小应该尽量和目标系统以后的工作负载相似，以便测试出真实的负载能力。
- -k 使用内核2.6的异步I/O特性。经常用在数据库服务器的测试，例如IBM的DB2。

- **-m** 如果目标应用使用多个内部缓冲，那么可以使用**-m**选项模拟这个行为。
- **-s** 指定测试文件的大小。对于异步文件系统（大多数文件系统的默认挂载选项），**IOzone**应该使用至少两倍内存大小的测试文件才能真正衡量出磁盘性能。测试文件大小可以MB或者GB为单位，直接在大小后面跟上m或者g的单位即可。
- **-+u** 实验性的选项。可以衡量在测试期间处理器的使用率。

注意：保存在异步文件系统，并且小于系统内存大小的测试文件，只能测试内存吞吐量，而不是磁盘子系统的性能。所以，你的目标文件系统应该使用**sync**选项挂载，并且测试文件是系统内存的至少两倍。

如下的例子展示了使用**IOzone**来衡量挂载在`/perf`目录上的磁盘子系统的随机读性能，测试文件大小为10GB，32KB的block size。

```
./iozone -b results.xls -R -i 0 -i 2 -f /perf/iozone.file -r 32 -s 10g
```

最后，可以把获得的结果导入到电子表格中，并且转换为图形格式。使用图形输出数据可以更简单的分析大文件数据并且看出趋势。下图就是上面的命令可能的一种输出情况。

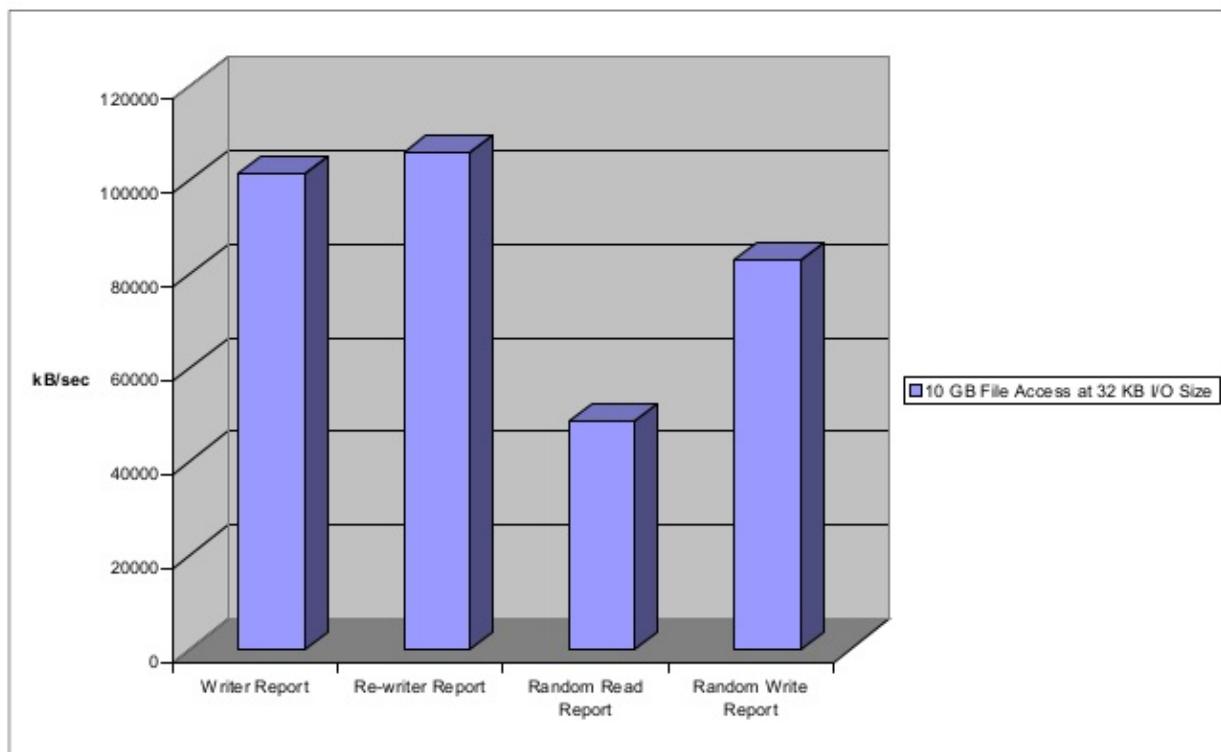


Figure 2-17 A graphic produced out of the sample results of Example 2-26

如果**IOzone**的文件大小和系统内存或者缓存一致，可以用来获取缓存和内存的吞吐量。应该注意的是，由于文件系统开销，**IOzone**只会获得70-80%的系统能力。

IOzone的地址：<http://www.iozone.org/>

netperf

netperf是专注于TCP/IP网络性能的测试工具。它支持UNIX的socket和SCTP测试。

netperf基于C/S（client-server）模型设计。netserver运行在目标服务器上，netperf运行在客户机上。netperf控制netserver，netperf把配置数据发送到netserver，产生网络流量，从一个独立于测试连接的控制连接获取netserver的结果。在测试过程中，在控制连接中没有数据交流，所以不会对结果产生影响。netperf压测工具也有提供报表的功能，包括CPU使用率。本书写作期间，稳定版本是2.4.3(本书翻译时候，netperf稳定版是2.6.0)。

netperf能生成多种类型的流量。基本分为两种：大块传输数据的流量和请求/响应的流量。netperf一次只使用一个socket。在下一版本的netperf（netperf4）中，将会全面支持并发会话。当前，我们可以使用如下办法进行多会话的测试。

- 大块数据传输(Bulk data transfer)

大块数据传输能力是网络测试中最常用的指标，通过一秒钟传送的数据量来衡量。它可以模拟大文件传输，例如多媒体流和FTP的数据传输。

- 请求/响应 (Request/response type)

模拟请求/响应的流量，衡量指标是一秒钟完成的请求数量。请求/响应是最常见的在线应用流量，例如web、数据库、邮件服务、文件服务（中小文件），目录服务。在真实的环境中，会话的建立终止和数据交换的方式是一样的。为了模拟它，需要使用到TCP_CRR类型。

- 并发会话(Concurrent session)

在当前的稳定版中，netperf还不真正支持并发会话测试，但是，我们可以以如下的方式开启多个netperf实例：

```
for i in `seq 1 10`; do netperf -t TCP_CRR -H target.example.com -i 10 -P 0; done
```

我们可以看看如下一些有用和有趣的选项，

全局选项：

- -A 设置本地接收和发送缓冲的调整
- -b 爆发大量流测试包
- -H 远程机器
- -t 测试流量类型
 - TCP_STREAM 大量数据传输测试

- TCP_MAERTS 和TCP_STREAM很像，只是流的方向相反
- TCP_SENDFILE 和TCP_STREAM很像，只是使用sendfile()，而不是send()。会引发zero-copy操作
- UDP_STREAM 和TCP_STREAM很像，只不过是UDP
- TCP_RR 请求响应报文测试
- TCP_CC TCP的连接/关闭测试。不产生请求和响应报文。
- TCP_CRR 执行连接/请求/响应/关闭的操作。和禁用HTTP keepalive的HTTP1.0/1.1相似。
- UDP_RR 和TCP_RR一样，只不过是UDP。
- -l 测试长度。如果是一个正值，netperf会执行testlen秒。如果值为负，netperf一直执行，直到大量数据传输测试中交换testlen字节，或者在请求/响应次数达到testlen。
- -c 本地CPU使用率报告
- -C 远程服务器CPU使用率报告

在某些平台上，CPU使用率的报告可能不准确。在性能测试之前，请确保准确性。

- -l 这个选项是用来维护结果可信度的。可信级别应该设置为99%或者95%。为了保证结果可信度级别，netperf会把多次重复测试。例如-l 99 5，代表在100次的99次中，测试结果和真实情况有5% (+-2.5%) 的浮动区间。
- -i 这个选项限制了最大和最小的重复次数。-i 10 3表示，netperf重复同样的测试，最多10次，最少3次。如果重复次数超过最大值，结果就不在-l指定的可信级别中，将在结果中显示一个警告。
- -s , -S 修改发送和接收的本地和远程缓冲大小。这个会影响到窗口大小。

TCP_STREAM,TCP_MAERTS,TCP_SENDFILE,UDP_STREAM的选项

- -m , -M 指定传给send()和recv()函数的缓冲大小。分别控制每个调用的发送和接收大小。

TCP_RR,TCP_CC,TCP_CRR,UDP_RR的选项：

- -r ,-R 分别指定请求和响应的大小。例如-r 128,8129意思是netperf发送128字节包到netserver，然后它响应一个8129字节的包给netperf。

如下是netperf的一个TCP_CRR的测试

Example 2-27 An example result of TCP_CRR benchmark

Testing with the following command line:

```
/usr/local/bin/netperf -l 60 -H plnxsu4 -t TCP_CRR -c 100 -C 100 -i ,3 -I 95,5 -v  
1 -- -r 64,1 -s 0 -S 512
```

```
TCP Connect/Request/Response TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to plnxsu4  
(10.0.0.4) port 0 AF_INET
```

Local /Remote

Socket	Size	Request	Resp.	Elapsed	Trans.	CPU	CPU	S.dem	S.dem
Send	Recv	Size	Size	Time	Rate	local	remote	local	remote
bytes	bytes	bytes	bytes	secs.	per sec	%	%	us/Tr	us/Tr
16384	87380	64	1	60.00	3830.65	25.27	10.16	131.928	53.039
2048	1024								

当执行性能测试的时候，最好的做法是使用netperf的简单测试脚本。通过修改脚本的变量，你可以定制自己的测试。脚本的路径在你的netperf包的doc/example/目录下。

详细信息见:<http://www.netperf.org/>

其它有用工具

这里还有其它一些有用的测试工具。记住，你必须了解压测工具的特性，然后才能选择你需要的工具。

工具	主要功能
bonnie	磁盘IO和文件系统测试 http://www.textuality.com/bonnie/
bonnie++	磁盘IO和文件系统测试 http://www.coker.com.au/bonnie++/
NetBench	文件服务器测试，运行在Windows上
dbench	文件系统测试，通常用来文件服务器压测 http://freshmeat.net/projects/dbench/
iometer	磁盘I/O和网络压测 http://www.iometer.org/
ttcp	简单的网络测试
iperf	网络测试 http://dast.nlanr.net/projects/Iperf/
ab (Apache Bench)	简单的web服务器测试，包含在Apache服务器中 http://httpd.apache.org/
WebStone	Web服务器测试 http://www.mindcraft.com/webstone/
Apache Jmeter	主要用来web服务器性能测试。也支持其它协议，例如SMTP，LDAP，JDBC等，有很好的报告功能 http://jakarta.apache.org/jmeter/
fsstone , smtpstone	邮件服务器测试，包含在postfix中 http://www.postfix.org/
nhfsstone	网络文件系统测试，包含在nfs-utils包中
DirectoryMark	LDAP测试 http://www.mindcraft.com/directorymark/

分析性能瓶颈

本章讲解如何发现影响服务器性能的瓶颈。我们描述了一系列的步骤来引导你，帮你找到问题，并将性能恢复到可接受的水平。

本章包含如下内容：

- 3.1 发现瓶颈
- 3.2 CPU瓶颈
- 3.3 内存瓶颈
- 3.4 磁盘瓶颈
- 3.5 网络瓶颈

寻找瓶颈

- 寻找瓶颈
 - 收集信息
 - 分析服务器性能

如下步骤是我们快速调优的办法：

- 1 了解你的系统
- 2 备份系统
- 3 监控和分析系统性能
- 4 缩小瓶颈范围，找到原因
- 5 通过修改一个因素，修复瓶颈
- 6 回到第3步，直到系统性能恢复到正常水平

| 你应该记录下每一步，尤其是你的修改，以及对性能的影响

收集信息

通常，你获得的第一手信息都是“服务器有问题”，关键是利用可能性因素来厘清和记录问题。有如下几个问题可以帮你更好的了解系统。

- 你能全面描述一下服务器的问题吗？
 - 模式
 - 使用时间
 - 配置
 - 外围设备
 - 操作系统版本和更新级别
- 请告诉我具体的问题是什么？
 - 症状是什么？
 - 任何错误信息

有些人不能回答这些问题，但是任何相关信息都有可能帮助你找到问题。例如，别人可能告诉你，“拷贝大文件到服务器的时候很慢”。这表明网络或磁盘子系统可能存在问题是。

- 谁遇到这个问题？

是单一个人，一个用户组，还是整个部门？这些信息可以帮助判断问题是部分网络出了问题，还是因为应用程序的问题，等等。如果只有一个用户出现问题，那很可能是这个用户的PC有不对劲的地方。

出现问题的客户端常常是解决问题的关键。由此看来，性能问题可能不是直接和服务器相关的：服务器和客户端之间的网络路径很容易导致问题。这个路径包括可网络设备和其它提供服务的服务器，例如域控。

- 问题能重现吗？所有能重现的问题都可以解决。如果你对系统足够了解，你应该能够缩小问题范围到病根上，并且决定采取什么措施。

如果能重现，可以帮你更好的理解问题。记录操作对重现问题很有用：

- 重现问题的步骤？知道步骤，可能会帮你在另外一台条件相同的机器上重现相同的问题。如果可行，那就可以使用测试环境的机器调试，避免生产服务器崩溃。
- 如果问题是断续出现的，第一件要做的事情就是收集信息，找到重现问题的条件。目的是使用脚本让问题在命令行出现。
- 问题是否在一天中特定的时间或者一周中特定的一天出现？这可能帮助你确定导致问题的原因。问题可能是在所有人开始上班或者去吃午饭的时候出现。寻找修改定时的办法（让它更常或者更少出现）；那么，问题就重现了。
- 是否常见？如果问题是不能重现的，你可能推断是特殊条件下的问题，而且把问题作为已修复的。在现实中，它重新出现的可能性很大。一个处理不可重现问题的好办法是在服务器上执行一般的维护操作：重启，升级系统的补丁和驱动。
- 问题是什么时候开始的？是缓慢浮现，还是迅速闪现？如果性能问题是逐渐浮现出来，那可能是空间问题。如果出现得很突然，可能是由于服务器某个修改或者外围设备导致的。
- 是否对服务器做过修改（无论大小），或者改变了客户端使用服务器的方式？是否是用户修改过服务器，或者外围设备导致的问题？是否有日志记录了所有网络修改？

业务修改可能导致需求的改变，也可能导致服务器和网络系统需求的改变。

- 是否和其它服务器或者硬件组件相关。
- 是否有日志可以查询
- 问题的优先级如何，需要在什么时候修复？
 - 是否在接下来的几分钟要修复，或者一天内？你可能有一些时间修复它，或者已经进入恐慌模式。
 - 问题有多大？
 - 问题的相关开销是多少？

分析服务器性能

重要：在做排错之前，备份所有数据和配置信息，以免丢失。

你应该要监控服务器，最简单的办法是在被监控服务器上运行监控工具。

应该在操作高峰记录服务器的性能日志，例如早9点到晚上5点，这取决于服务器提供的是什么服务，和谁在使用这些服务。在创建日志的时候，应该尽力把如下的数据记录在里面：

- 处理器
- 系统
- 服务器工作队列
- 内存
- 页文件
- 物理磁盘
- 重定向器
- 网卡

在开始之前，请注意性能调优的办法很重要。我们推荐的服务器性能调优过程如下：

- 1 理解影响服务器性能的因素
- 2 根据当前服务器性能表现，设立基准线，用来和未来的测试数据做比较，找出性能瓶颈。
- 3 使用监控工具来找出性能瓶颈。通过接下来的章节中的方法，你就可以把瓶颈范围缩小到部分子系统上。
- 4 通过对造成瓶颈的组件执行某些操作，提升服务器性能。

注意：很重要的是，在其它组件具有足够能力维持高性能的时候，通过升级造成瓶颈的组件，可以获得极大的性能提升。

- 1. 衡量新的性能。这一步可以帮助你比较调优前和调优后的性能差别。

在尝试修复性能问题的时候，注意：

- 应用程序应该用适当的方法来编译，减少路径长度。
- 在做任何升级和修改之前做性能衡量，这样就可以知道修改是否有效。
- 除了检查新添加的硬件，还要检查重新配置过的旧硬件。

CPU瓶颈

- CPU瓶颈
 - 找出CPU瓶颈
 - SMP
 - 性能优化选项

对于应用或数据库服务器来说，CPU是十分关键的资源，也常常是性能瓶颈的源头。需要明白的是，高的CPU利用率并不总是意味着CPU正在繁忙的工作；也可能是正在等着其它子系统完成工作。正确的判断，需要把整个系统作为一个整体，并且观察到每一个子系统，因为子系统可能存在关联的反应。

在普遍的认知中，总是误以为CPU是服务器最重要的部件。但情况并不总是这样，在服务器上配置的CPU总是好于磁盘、内存、网络等子系统组件。只有特定的CPU密集型程序才能真正利用到今天的高端处理器。

找出CPU瓶颈

有好几种办法可以确认瓶颈出现在CPU上。已经在第二章"监控和测试工具"中讨论过了，Linux有各类工具来帮助我们。关键是选择什么工具。

可以使用`uptime`。通过分析`uptime`的输出，可以粗略知道在过去15分钟里，服务器上发生了什么。关于这个命令的更多解释，参考`uptime`这一节。

Example 3-1 uptime output from a CPU strapped system

```
18:03:16  up 1 day,  2:46,  6 users,  load average: 182.53, 92.02, 37.95
```

使用KDE System Guard和CPU传感器可以看到当前CPU负载。

小贴士：一次不要运行太多工具，避免增加CPU的问题。一次运行太多不同的监控工具，可以导致CPU负载飙升。

使用`top`工具，你可以看到CPU利用率和哪个进程是消耗CPU的大户。如果设置了`sar`，你会收集到很多信息，比如一段时间内的CPU利用率。分析这些信息的办法可能不同，使用`isag`，可以对`sar`的输出画出图形。你还可以通过脚本和表格来分析信息，看看CPU利用率的走向。你也可以在命令行使用`sar -u`或者`sar -U processnumber`。要想获得系统更全面情况，而不只是CPU子系统，`vmstat`是个好帮手。

SMP

基于SMP的系统呈现出来的问题可能是很难检测到的。在SMP环境中，有一个CPU affinity的概念，表示绑定进程到CPU上。这么做的主要好处是CPU缓存优化，可以让同样的进程运行在

一个CPU上，而不是在多个CPU上切换。当进程在CPU间切换的时候，要刷新新的CPU的缓存。进程在处理器间切换的时候会导致很多缓存刷新，那样，一个独立的进程要花费更多的时间才能处理完。而探测器很难检测到，在监控中，CPU负载会十分均衡，不会在任何一个上出现高峰。在基于NUMA的系统上，比如IBM System x3950上，CPU affinity也很有用，重要的是保持内存、缓存和CPU访问都是本地的。

性能优化选项

第一步是要确保，系统性能问题是由CPU引起的，而不是其它子系统。如果处理器是服务器瓶颈，可以采取如下的办法来增强性能：

- 使用`ps -ef`来确保没有不必要的进程程序在后台运行，如果找到了这样的程序，关掉它，或者使用`cron`让它在非高峰的时候运行。
- 通过`top`找到非关键的、CPU密集型进程，然后用`renice`修改它的优先级。
- 在基于SMP的机器上，尝试使用`taskset`命令绑定进程到CPU上，避免进程在多个处理器之间切换，引起cache刷新。
- 基于运行的应用，确认你的应用是否能高效的利用多处理器。来决定是否应该使用更强劲的CPU而不是更多的CPU。例如，单线程应用，会从更快的CPU中受益，增加值CPU个数也没用。
- 还有其它办法，比如，确保你使用的是最新的驱动和固件，这能影响到他们在系统上的负载。

内存瓶颈

- 内存瓶颈
 - 找到内存瓶颈
 - 性能调优选项

Linux系统上，有很多进程在同时运行。这些程序支持多用户，有些进程比另外的用的更多一些。当一个应用访问缓存的时候，会保持较高的性能，因为它从内存中取数据，而不是速度很慢的磁盘中。

操作系统使用算法来控制哪些程序使用物理内存，而哪些程序要page out。这对用户程序是透明的。Page空间是由操作系统在当前没有使用的磁盘上，创建的用来存放用户程序的一个文件。通常，page size是4KB或者8KB。在Linux上，在include/asm-/param.h内核头文件中，EXEC_PAGESIZE变量定义了Page size的大小。把程序page out到磁盘的过程叫做pageout。

找到内存瓶颈

从列出在服务器上运行的应用开始分析。看看每个程序用了多少物理内存和swap空间。下图展示KDE System Guard监控内存的使用情况。![[kde-system-guard-memory-monitoring](kde-system-guard-memory-monitoring.png)]下面的表格也可你帮你定义内存的问题：| 内存指示器 | 分析 | --- | --- | --- | 可用内存(Memory available) | 表示还有多少内存可以使用。如果在开启你的应用之后，这个值明显降低了，可能是因为内存泄露。检查应用，做出适当的调整。使用free -l -t -o来查看额外信息 | 页错误 (Page faults) | 有两类页错误：软页错误，在内存中发现页；硬页错误，在内存中没有发现页，而必须从磁盘中获取。访问磁盘会显著的使应用变慢。sar -B命令可以提供观察页错误的信息。, 尤其是pgpgin/s和pgpgout/s两列。| 文件系统缓存(File system cache) | 被文件系统缓存使用的内存空间。使用free -l -t -o命令查看信息。| 进程独占缓存(Private memory for process) | 服务器上各个进程使用的内存。可以使用pmap命令查看给特定的进程分配了多少内存 | **页和交换指示器** 和所有基于UNIX的系统一样，在Linux中，分页(paging)和交换(swapping)是有区别的，分页是把独立的页移动到磁盘上的交换分区；交换是一个大型操作，把进程所有的地址空间一次全部移动到交换分区。交换可能是由两种因素引起的：+ 进程进入睡眠模式。这种事情经常发生，因为进程依赖于交互操作、编辑器、shell等等，应用程序把大多数时间花在等待用户输入数据上。在此期间，进程是非激活的。+ 进程行为异常。分页可能是严重的性能问题，当空闲内存量小于预设的值时，因为分页机制不能处理对物理内存页的请求，于是调用swap机制释放更多的页，把内存数据放入到swap中。这会导致增加I/O，并且明显的性能降低。如果服务器总是分页到磁盘（page-out很高）上，应该考虑增加更多的内存。然而，如果系统的page-out很低，可能是性能利用不充分。

性能调优选项

如果确定是内存瓶颈，可以执行下面的操作：

- 使用bigpages、hugetlb和共享内存调优swap空间。
- 增加或者减少页大小。
- 改善活动和非活动的内存处理
- 调整page-out率
- 限制服务器上每个用户可使用的资源
- 关掉用不到的服务
- 增加内存

磁盘瓶颈

- 磁盘瓶颈
 - 找到磁盘瓶颈
 - 性能调优选项

磁盘子系统通常是服务器性能的最重要方面，是瓶颈问题的高发部件。但是，磁盘问题表现的有时候并不是那么直接，比如说可能是内存不足。如果CPU周期浪费在等待I/O任务完成，应用程序可能被认为是I/O密集型。

最常见的磁盘问题是磁盘太少。大多数磁盘配置是基于容量需求，而非性能。最廉价的办法可能是购买大空间磁盘。然后，这样每张盘上都存放了更多的用户数据，磁盘上会产生更大的I/O速度和风险。

第二常见的问题是在一个整列中划分太多的逻辑分区，会增加寻道时间，降低性能。

找到磁盘瓶颈

服务器表现出如下的症状，可能是磁盘出现了瓶颈：

- 磁盘慢的表现：
 - 内存缓冲中填满了写数据（或者在等待读数据），因为没有可用的空闲内存缓冲供写（或者是在等待磁盘队列中的读数据响应），拖慢了所有请求。
 - 内存不足，在没有可以为网络请求分配足够内存缓冲的时候，会产生同步磁盘I/O。
- 磁盘或者控制器使用率变高。
- 大多数网络传输都是在磁盘I/O完成之后。表现形式为极长的响应时间和非常低的网络利用率。
- 磁盘I/O花费相当长的时间，并且磁盘队列变满，因为处理请求时间变长，所以CPU利用率变得很低。

磁盘子系统可能是最难配置的子系统。除了查看磁盘接口速度和磁盘容量，还要理解磁盘负载。访问磁盘是随机的还是顺序的？I/O是大还是小？为了充分利用磁盘，需要回答上面的这些问题。

厂商会一般会给你展示它们设备的吞吐量上限。但是，花时间来了解你的工作负载吞吐量将会帮你找到你所需要的磁盘子系统。

下表展示了不同驱动在8KB I/Os下的真实吞吐。

磁盘速度	延时	寻道时间	完全随机访问时间	单盘每秒 I/O	8 KB I/O的吞吐
15000 RPM	2.0 ms	3.8 ms	6.8 ms	147	1.15 Mbps
10000 RPM	3.0 ms	4.9 ms	8.9 ms	112	900 KBps
7200 RPM	4.2 ms	9 ms	13.2 ms	75	600 KBps

- a. 如果处理命令 + 传输数据 < 1ms，完全随机访问 = 延时 + 寻道时间 + 1ms
- b. 以1/随机访问时间为吞吐量。

随机读写负载通常需要多个磁盘决定。SCSI或者光纤的带宽不太要关注。大量随机访问负载的数据库最好有多块磁盘。大的SMP服务器最好配置多块磁盘。通常磁盘可以简单的平均划分为70%的读和30%的写，RAID10的性能比RAID5要高出50%到60%。

顺序读写需要看磁盘子系统的总线带宽。需要最大吞吐量的时候，需要特别关注SCSI总线或者光纤控制器的数量。在阵列中，为每个盘指定相同的数量，RAID-10、RAID-0和RAID-5的读写吞吐流很相似。

分析磁盘瓶颈的办法：实时监控和跟踪。

- 在问题发生的时候一定要做实时监控。在动态系统负载和问题不可重现的情况下，这可能是不实际的。然而，如果问题是可重现的，通过这个办法就可以增加对象和计数器使问题更清晰。
- 跟踪是通过收集一段时间的性能数据来诊断问题。这是远程性能分析的好办法。缺点是在问题不可重现的时候，需要分析大量的文件，如果没有跟踪到所有的关键对象和参数，必须等待下一次问题出现来获取额外的数据。

vmstat命令

跟踪磁盘的一种办法是使用vmstat工具。vmstat中关于I/O最重要的列是bi和bo。这两个字段监控了各个时刻进出磁盘的块。设置了基线就可以找到随着时间的变化。

Example 3-2 vmstat output

```
[root@x232 root]# vmstat 2
r b swpd free buff cache si so bi bo in cs us sy id wa
2 1 0 9004 47196 1141672 0 0 0 950 149 74 87 13 0 0
0 2 0 9672 47224 1140924 0 0 12 42392 189 65 88 10 0 1
0 2 0 9276 47224 1141308 0 0 448 0 144 28 0 0 0 100
0 2 0 9160 47224 1141424 0 0 448 1764 149 66 0 1 0 99
0 2 0 9272 47224 1141280 0 0 448 60 155 46 0 1 0 99
0 2 0 9180 47228 1141360 0 0 6208 10730 425 413 0 3 0 97
1 0 0 9200 47228 1141340 0 0 11200 6 631 737 0 6 0 94
1 0 0 9756 47228 1140784 0 0 12224 3632 684 763 0 11 0 89
0 2 0 9448 47228 1141092 0 0 5824 25328 403 373 0 3 0 97
0 2 0 9740 47228 1140832 0 0 640 0 159 31 0 0 0 100
```

iostat命令

在反复同时打开、读、写、关闭太多文件的时候可能遇到性能问题。这可能会以寻道时间（把磁头移动到数据存储位置的时间）变长的现象表现出来。使用iostat工具，可以监控I/O设备的实时负载。不同的选项可以帮你挖掘到更深更多有用的数据。

下图展示了在/dev/sdb1设备上潜在的I/O瓶颈。输出显示平均等待时间(await)是2.7秒，服务时间(svctm)是270ms。

Example 3-3 Sample of an I/O bottleneck as shown with iostat 2 -x /dev/sdb1

```
[root@x232 root]# iostat 2 -x /dev/sdb1
avg-cpu: %user %nice %sys %idle
          11.50    0.00   2.00   86.50

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz
avgqu-sz await svctm %util
/dev/sdb1 441.00 3030.00 7.00 30.50 3584.00 24480.00 1792.00 12240.00 748.37
          101.70 2717.33 266.67 100.00

avg-cpu: %user %nice %sys %idle
          10.50    0.00   1.00   88.50

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz
avgqu-sz await svctm %util
/dev/sdb1 441.00 3030.00 7.00 30.00 3584.00 24480.00 1792.00 12240.00 758.49
          101.65 2739.19 270.27 100.00

avg-cpu: %user %nice %sys %idle
          10.95    0.00   1.00   88.06

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz
avgqu-sz await svctm %util
/dev/sdb1 438.81 3165.67 6.97 30.35 3566.17 25576.12 1783.08 12788.06 781.01
          101.69 2728.00 268.00 100.00
```

更多字段的详细信息，参考iostat(1)的man手册。

下一章中会有如何修改elevator算法和avgrq-sz（average size of request，平均请求大小），以及qvgqu-sz（average queue length，平均队列长度）的内容。修改elevator设置使得延时变低，avgrq-sz会变小。通过监控rrqm/s和wrqm/s的变化，可以看出调优对磁盘能管理的读写合并数产生的影响。

性能调优选项

在确定磁盘子系统瓶颈之后，有如下可能的解决方法：

- 如果负载是顺序的，压力在控制器带宽上，办法就是添加更快的磁盘控制器。然而，如果负载是随机的，瓶颈可能在磁盘上，增加更多多的磁盘可以帮助增加性能。
- 在RAID中添加更多的磁盘，把数据分散到多块物理磁盘，可以同时增强读和写的性能。增加磁盘会提升每秒的读写I/O数。另外，请使用硬件RAID而不是Linux提供的RAID软件。如果是硬件RAID，RAID级别对操作系统是不可见的。
- 考虑使用Linux逻辑卷分区，而不是没有分区的单块大磁盘或者逻辑卷。
- 把处理负载转移到网络中的其它系统（用户，应用程序或者服务）。
- 添加RAM。添加内存会提升系统磁盘缓冲，增强磁盘响应速度。

网络瓶颈

- 网络瓶颈
 - 找到网络瓶颈
 - 网络性能调优

网络的性能问题会导致很多其它问题，比如内核崩溃（kernel panic）。每个Linux发行版都包含了流量分析工具来检测网络瓶颈。

找到网络瓶颈

因为易用性和友好的图形界面，我们推荐KDE System Guard。上文已经讨论过，它的安装包包含在发行的CD中。

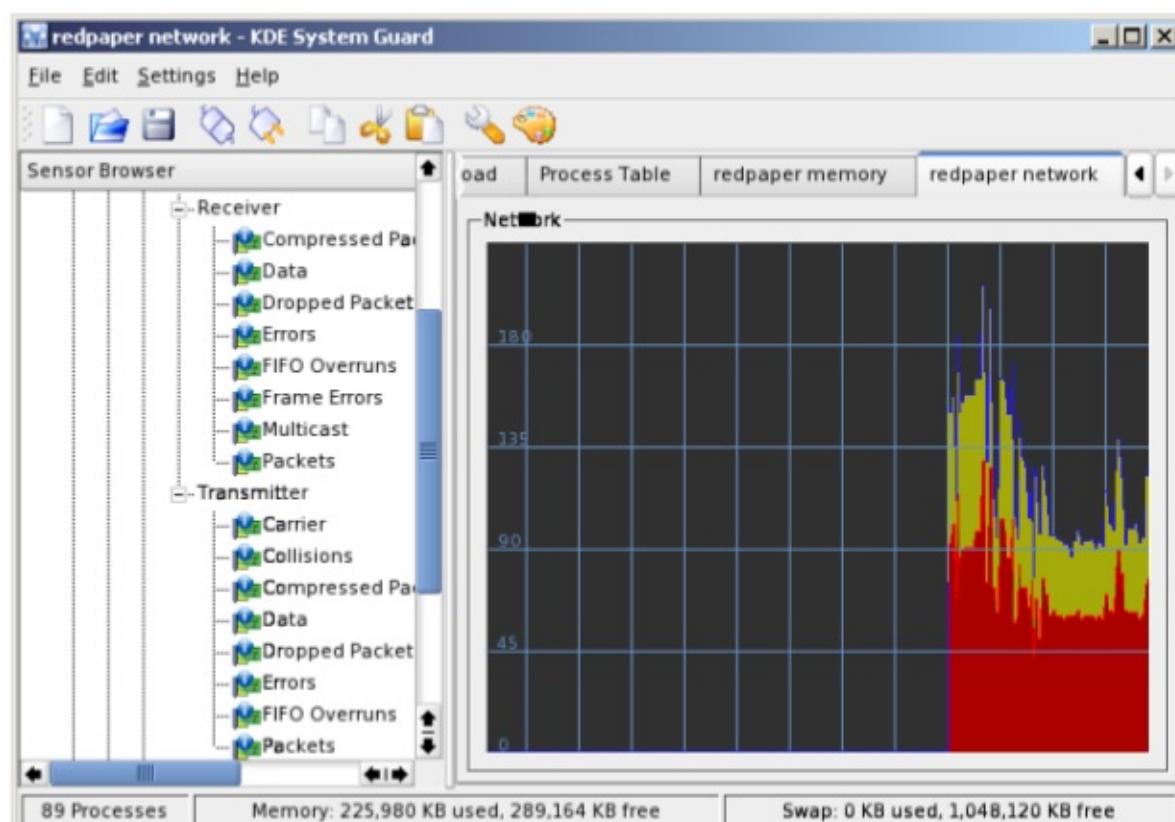


Figure 3-2 KDE System Guard network monitoring

导致性能问题的原因可能有很多，有时候问题是一起发生的，使得解决起来更加棘手。下图解释了对指示器的分析，可能帮你判断网络问题。

网络指示器	分析
接收和发送包	显示进出特定网卡的包数目。同时检查内网和外网网卡
碰撞包	在一个域中有很多机器时，使用中继器可能导致许多碰撞
丢弃包	系统丢弃包的原因可能有很多，可能影响到性能。比如，如果服务器网卡配置为100M全双工模式，但是交换机网卡配置为10M，路由器的访问控制列表（ACL）过滤器可能会丢弃包。例如： <pre>iptables -t filter -A FORWARD -p all -i eth2 -o eth1 -s 172.18.0.0/24 -j DROP</pre>
错误	如果网络线路（例如，手机线路）太差会出现错误。在这种情况下，会重发包，导致降低网络吞吐量。
有问题的适配器	网络问题经常可能是由于坏的网卡导致的。当发生这种硬件问题时，可能会广播发送垃圾包

网络性能调优

当网络瓶颈出现时，你应该试试如下的办法：

- 确保网卡配置和路由器交换机配置相匹配。
- 修改子网的组织方式
- 使用更快的网卡
- 适当调整IPv4的TCP内核参数。有些安全相关的参数调整会提升性能，详见下一章。
- 如果可能的话，更换网卡，然后重新检测性能。
- 如果可能的话，增加网卡，绑定成一个网卡组。

Linux发行版和内核提供了各种参数和选项给Linux用户优化系统性能，将软硬件性能发挥到最大。正如前文所说，没有适用于所有应用的提升系统性能的优化魔法。下文中所讨论的提升性能的办法只适用于特定的硬件和应用。某些配置会提升web服务器性能，但是配置在数据库系统上可能适得其反。

这节中的步骤可以用来优化基于内核2.6的Linux发行版。由于在本书写作时，基于2.6内核的发行版从2.6.9到2.6.19的版本都有，有些优化选项可能只适用于特定的内核版本。本章内容如下。

- Linux内核管理
- 系统清理
- 使用**sysctl**优化性能
- 网络优化

章节如下：

- 4.1 优化原则
- 4.2 安装考量
- 4.3 修改内核参数
- 4.4 优化处理器子系统
- 4.5 优化虚拟内存子系统
- 4.6 优化磁盘子系统
- 4.7 优化网络子系统

优化原则

- 优化原则
 - 管理变更

优化任何系统都应该基于一些简单的原则，最重要的是变更管理。大体上，系统调整的第一步是分析和评估当前系统配置。确保所有系统设备运行在硬件厂家所宣称的最佳状态，有助于为接下来的调整提供坚实基础。还有，优良的业务系统上应该没有任何不必要的服务和子系统在运行。最后，落实到具体的系统优化，需要注意的是，调优通常是针对特定应用负载而进行的。所以，根据系统负载行为的调优使得系统更高效，但是对其他负载模式可能是不利的。例如，对系统低延时的调整，通常来说会降低系统的吞吐量。

管理变更

管理变更和性能优化并不直接相关，但可能是成功性能调优最重要的因素。如下可能是第二位考虑的，但是作为提醒，我们强调一下：

- 在调优之前，实施合理的管理流程变更
- 永远不要在生产系统上调优
- 在调优过程中，每次只修改一个变量
- 反复测试提升性能的参数，有时候，统计来的结果更加可靠
- 把成功的参数调整整理成文档，和社区分享，即使你觉得它们微不足道。生产环境中获得的任何结果对Linux性能都有很大用处。

安装考量

- 安装考量
 - 安装
 - 检查当前配置
 - 最小化资源使用
 - SELinux
 - 编译内核

理想情况下，针对特定性能目标的系统优化应该从系统设计和安装阶段开始。朝着特定的负载模式合理安装和剪裁系统可以为以后的优化节省大量的时间。

安装

在理想的世界中，对某一给定系统的优化应该在非常早就开始，系统已经根据应用需要和预期负载精心剪裁。虽然在出现瓶颈的时候管理员必须去做优化，但是我们还是应该注意在初始化安装操作系统时候的优化可能。

在开始安装系统之前，应该确定如下几个问题：

- 选择处理器技术
- 选择磁盘技术
- 应用

然而，这些内容超出了本书范围。

理想情况下，应该在开始安装前应该回答如下问题：

- 使用什么版本的Linux？在确定业务和应用需求之后，决定使用什么版本的Linux。企业往往有规定，只能使特定的Linux，在这种情况下，出于财务和规定的原因，会决定能够使用什么系统。但是如果你有完全的选择自由，你可以考虑如下的几个问题：

- 选择受商业支持的Linux还是定制的发行版

在科学研究领域中，可以选择不受支持的Linux，比如Fedora。对于企业来说，我们强烈推荐受到商业支持的发行版，例如Red Hat Enterprise Linux或者Novell SUSE Enterprise Linux。

- 商业发行版的什么版本？

不同的商业Linux发行版在内核版本、支持的软件包或者特性，以及最重要的硬件支持上都有区别。在安装之前，先仔细检查支持的硬件配置，避免浪费硬件能力。

- 选择正确的内核。下表中列出了企业Linux发行版提供的几个内核包。由于性能原因，确保选择为你的系统选择最合适的选择。然而，在大多数情况下这都是由安装规范决定的。请注意，在不同发行版中内核包名字的区别。

内核类型	描述
标准版	单处理器机器
SMP	内核支持SMP和超线程机器。某些还同时支持NUMA结构。可能有一些变种，取决于内存和CPU个数，等等
Xen	包含一个运行在Xen虚拟机的内核版本

许多最新内核有一种叫做SMP选择的能力，在启动时优化自己。具体参考详细的发行说明。

- 选择什么样的分区布局？

通常人们都是根据应用需求、系统管理、个人偏好等选择规划磁盘分区，而不是性能。多数情况下，磁盘分区规划都是规定好的。唯一的建议是，应该分出一个swap分区来。swap分区和swap文件相比，减少了swap文件带来的文件系统消耗。交换分区很简单，在必要的时候，可以使用额外的交换分区和交换文件进行扩展。

- 选择什么文件系统？

不同文件系统在数据完整性和性能上有不同的表现。某些文件系统可能不受到其它发行版或者要运行的程序支持。对大多数服务器来说，默认推荐安装的文件系统会提供足够的性能。如果你对最小延时或者最大吞吐量有特别的要求，还是推荐你根据需要选择不一样的文件系统。

- 包选择：最小化，还是所有软件包？

在安装Linux的时候，管理员会面临是最小化安装还是要安装所有软件包的选择。有些人倾向于安装所有包，这样就会解决许多包依赖问题。

考虑到如下的问题：在安装所有包的系统上可能产生更多安全问题。在生产环境安装所有开发工具显然也更可能导致的安全问题。越少安装软件，磁盘可用空间越多。包安装管理器会帮我们解决软件依赖问题，例如红帽公司的包管理器rpm和yum。我们推荐你选择安装少的软件包，只针对特别的应用选择必须的软件包。

- 防火墙配置

是否开启防火墙也是需要考虑的问题。防火墙的作用是保护系统免受恶意攻击，然而，在大数据流的情况下，太多复杂的防火墙规则会降低性能。

- SELinux

在某些Linux发行版，例如RHEL（Red Hat Enterprise Linux，红帽企业版Linux）4.0中，安装向导会让你选择安装SELinux。SELinux会带来显著的性能消耗，你应该谨慎的评估，SELinux提供的安全保护是否真的有必要。

- 运行级别

最后一项是选择系统的默认运行级别。除非你的应用需要你的系统运行在级别5（用户图形模式），我们强烈推荐在所有服务器系统上使用3级别来运行。通常，在数据中心里的服务器上运行GUI是毫无必要的，而且会损耗性能。如果安装向导没有提供运行级别选项，我们建议你在系统初始化完成之后手动选择级别3。

检查当前配置

如介绍中所说，在优化之前全面了解系统十分重要。全面了解指的是确保所有的子系统按照设计的方式工作，并且没有异常。异常指的是，比如一个GB网卡的服务器出现网络性能瓶颈的情况。如果网卡自动协商为100MB半双工，即使调优Linux内核的TCP/IP实现也没用。

dmesg

dmesg的作用是显示内核信息。**dmesg**会提供硬件问题或者内核加载的问题信息。

还有，使用**dmesg**，你可以确定在服务器上安装什么硬件。在启动的时候，Linux会检查硬件，并且记录相关信息。可以使用**/bin/dmesg**命令查看这些日志。

Example 4-1 Partial output from dmesg

```
Linux version 2.6.18-8.el5 (brewbuilder@ls20-bc1-14.build.redhat.com) (gcc version 4.1.1
20070105 (Red Hat 4.1.
1-52)) #1 SMP Fri Jan 26 14:15:14 EST 2007
Command line: ro root=/dev/VolGroup00/LogVol00 rhgb quiet
```

```
No NUMA configuration found
Fakeing a node at 0000000000000000-0000000140000000
Bootmem setup node 0 0000000000000000-0000000140000000
On node 0 totalpages: 1029288
DMA zone: 2726 pages, LIFO batch:0
DMA32 zone: 768002 pages, LIFO batch:31
Normal zone: 258560 pages, LIFO batch:31
```

```
Kernel command line: ro root=/dev/VolGroup00/LogVol00 rhgb quiet
Initializing CPU#0
```

```
Memory: 4042196k/5242880k available (2397k kernel code, 151492k reserved, 1222k data, 196k
init)
Calibrating delay using timer specific routine.. 7203.13 BogoMIPS (1pj=3601568)
Security Framework v1.0.0 initialized
SELinux: Initializing.
SELinux: Starting in permissive mode
```

```
CPU: Trace cache: 12K uops, L1 D cache: 16K
CPU: L2 cache: 1024K
using mwait in idle threads.
CPU: Physical Processor ID: 0
CPU: Processor Core ID: 0
CPU0: Thermal monitoring enabled (TM2)
SMP alternatives: switching to UP code
ACPI: Core revision 20060707
Using local APIC timer interrupts.
result 12500514
Detected 12.500 MHz APIC timer.
SMP alternatives: switching to SMP code
```

```

sizeof(vma)=176 bytes
sizeof(page)=56 bytes
sizeof(inode)=560 bytes
sizeof(dentry)=216 bytes
sizeof(ext3inode)=760 bytes
sizeof(buffer_head)=96 bytes
sizeof(skbuff)=240 bytes

io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)

SCSI device sda: 143372288 512-byte hdwr sectors (73407 MB)
sda: assuming Write Enabled
sda: assuming drive cache: write through

eth0: Tigon3 [partno(BCM95721) rev 4101 PHY(5750)] (PCI Express) 10/100/1000BaseT Ethernet
00:11:25:3f:19:b4
eth0: RXcsums[1] LinkChgREG[0] MIirq[0] ASF[1] Split[0] WireSpeed[1] TSOcap[1]
eth0: dma_rwctrl[76180000] dma_mask[64-bit]

EXT3 FS on dm-0, internal journal

kjournald starting. Commit interval 5 seconds
EXT3 FS on sda1, internal journal
EXT3-fs: mounted filesystem with ordered data mode.

```

ulimit

这个命令是bash内建的，用来控制shell和由它启动的进程的可用资源，等等之类。

使用-a选项列出所有可以设置的参数：

Example 4-2 Output of ulimit

```
[root@x232 html]# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) 4
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
pipe size               (512 bytes, -p) 8
stack size              (kbytes, -s) 10240
cpu time                (seconds, -t) unlimited
max user processes      (-u) 7168
virtual memory          (kbytes, -v) unlimited
```

-S和-H用来设置特定资源的软硬限制。如果到达软限制，系统管理员会收到一个警告。硬限制是在用户收到”Out of file handles“错误消息之前达到的最大值。

例如，可以设置打开最大文件数（-n）的硬限制：

```
ulimit -Hn 4096
```

打开最大文件数的软限制：

```
ulimit -Sn 1024
```

查看软硬限制的值：

```
ulimit -Hn  
ulimit -Sn
```

可以使用如下的命令限制Oracle用户资源，开机生效的办法，在文件/etc/security/limits.conf输入如下行：

```
soft nofile 4096  
hard nofile 10240
```

此外，确保默认的pam配置文件（RHEL中是/etc/pam.d/system-auth，SUSE中是/etc/pam.d/common-session）有如下一行：

```
session required pam_limits.so
```

有了这一行，系统才能做上面的资源限制

查看完整的ulimit语法：

```
ulimit -?
```

最小化资源使用

为了发挥系统最好的性能，必须把资源浪费降到最低。为了获得更快的速度，赛车不会和普通汽车一样舒服，因为舒服的座椅也会浪费资源。同样的道理也适用于服务器系统。运行GUI和不必要的守护进程都是在降低整体性能。本部分内容是关于如何优化系统资源。

Daemons

在按照默认规则安装系统之后，可能会有几个不需要的服务和守护进程在运行。禁用掉用不到的守护进程，降低整体系统的内存占用、运行进程数以及上下文切换，更重要的是，降低暴露安全问题的风险，禁用掉不必要的守护进程还能提升系统启动速度。

默认情况下，某些系统中，有些已经启动的守护进程可以安全的停止并禁用。下表中列出了在各个发行版启动的守护进程。如果合适的话，你应该考虑把它们禁用。在各个系统上运行的进程数量可能是不一样的。关于这些守护进程的详细解释，参考 [system-config-services](#)。

守护进程	描述
apmd	高级电源管理守护进程，服务器上通常不用。
arpables_jf	arp表网络过滤器的用户空间程序。除非需要使用atp表，否则你可以安全的禁用这个进程
autofs	根据需要自动挂载文件系统（例如，自动挂载CD-ROM）。服务器上，通常不需要自动挂载文件系统
cpuspeed	自动调整CPU频率。在服务器环境，建议关闭
cups	通用UNIX打印系统。如果计划提供打印服务，不要禁用这个服务
gpm	文本console的鼠标服务。如果需要在本地console环境使用鼠标，不要禁用。
hpoj	惠普印表机支持。如果要在服务器上使用惠普印表机就不要禁用它
irqbalance	平衡多个处理器之间的中断，如果只有一个CPU，或者你希望静态的平衡IRQ，禁用掉它
isdn	ISDN调制解调器支持。如果在服务器上使用拨号就不要禁用它
kudzu	检测和配置新硬件，在硬件变更时手动运行
netfs	用来支持NFS网络共享，如果要支持NFS，不要关闭它
pcmcia	PCMCIA支持。很少有服务器系统用到PCMCIA适配器，基本上都应该禁用
portmap	RPC服务（NIS和NFS）的动态端口分配。如果系统不提供基于RPC的服务，没必要启动
rawdevices	raw设备绑定支持。如果不打算使用raw设备，关闭它
rpc*	各类远程过程调用守护进程，主要是NFS和Samba。如果系统不支基于RPC的服务，没必要开启
sendmail	邮件传输代理。如果要提供邮件服务，不要关掉它
smartd	自我监控和报告守护进程。监控S.M.A.R.T兼容设备的错误
xfs	X Windows的前端服务。如果系统要运行在runlevel 5，不要禁用它

注意：关闭xfs守护进程会阻止服务器运行X程序。只能在不使用GUI的服务器上禁用。在使用startx的命令开启桌面之前，确保xfs已经启动。

在SUSE和RHEL中，/sbin/chkconfig命令可以方便的管理各类守护进程的开机启动选项。在使用chkconfig之前，首先要使用如下的命令检查运行的守护进程。

```
/sbin/chkconfig --list | grep on
```

如果你希望在下次启动系统的时候不要运行某个守护进程，使用root运行下面的命令。下面的命令的结果是等价的，区别是第二条命令在所有的运行级别禁用了守护进程，使用--level可以指定具体的运行级别。

```
/sbin/chkconfig --levels 2345 sendmail off  
/sbin/chkconfig sendmail off
```

小贴士：为了避免把宝贵时间浪费在等待完成重启的过程，只需把run level分别设置为1、3或者5。

还有另外一个有用的系统命令，**/sbin/service**，管理员可以用来直接修改已注册服务的状态。第一步，管理员应该要确定服务的当前状态（以**sendmail**为例），命令如下：

```
/sbin/service sendmail status
```

关闭**sendmail**的命令：

```
/sbin/service sendmail stop
```

service命令十分有用，可以用来验证守护进程是否必要。使用**chkconfig**做出的修改只会在修改启动级别和重启的时候才会生效。然而，通过**service**禁用的守护进程在重启之后又会重新启动。如果你的发行版中没有**service**命令，你可以通过**init.d**目录对停止或启动守护进程。检查**cups**的命令如下：

```
/etc/init.d/cups status
```

也可以使用基于**GUI**的程序修改守护进程启动，如下图。在**RHEL**的**GUI**中，点击**主菜单->系统设置->服务器设置->服务**，或者使用如下的命令：

```
/usr/bin/redhat-config-services
```

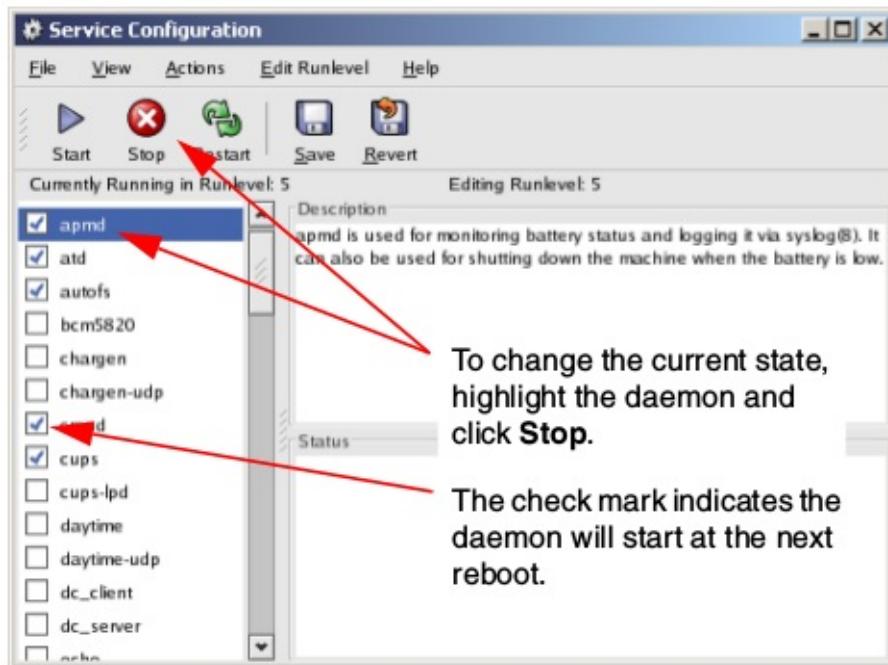


Figure 4-1 Red Hat Service Configuration interface

SUSE系统也通过YaST提供了同样的功能。在YaST中，服务配置在System->System Services（Runlevel）中。进入到服务配置中，推荐你使用专家模式，可以精确的设置各个守护进程状态。在runlevel 3中YaST的样子如下图：

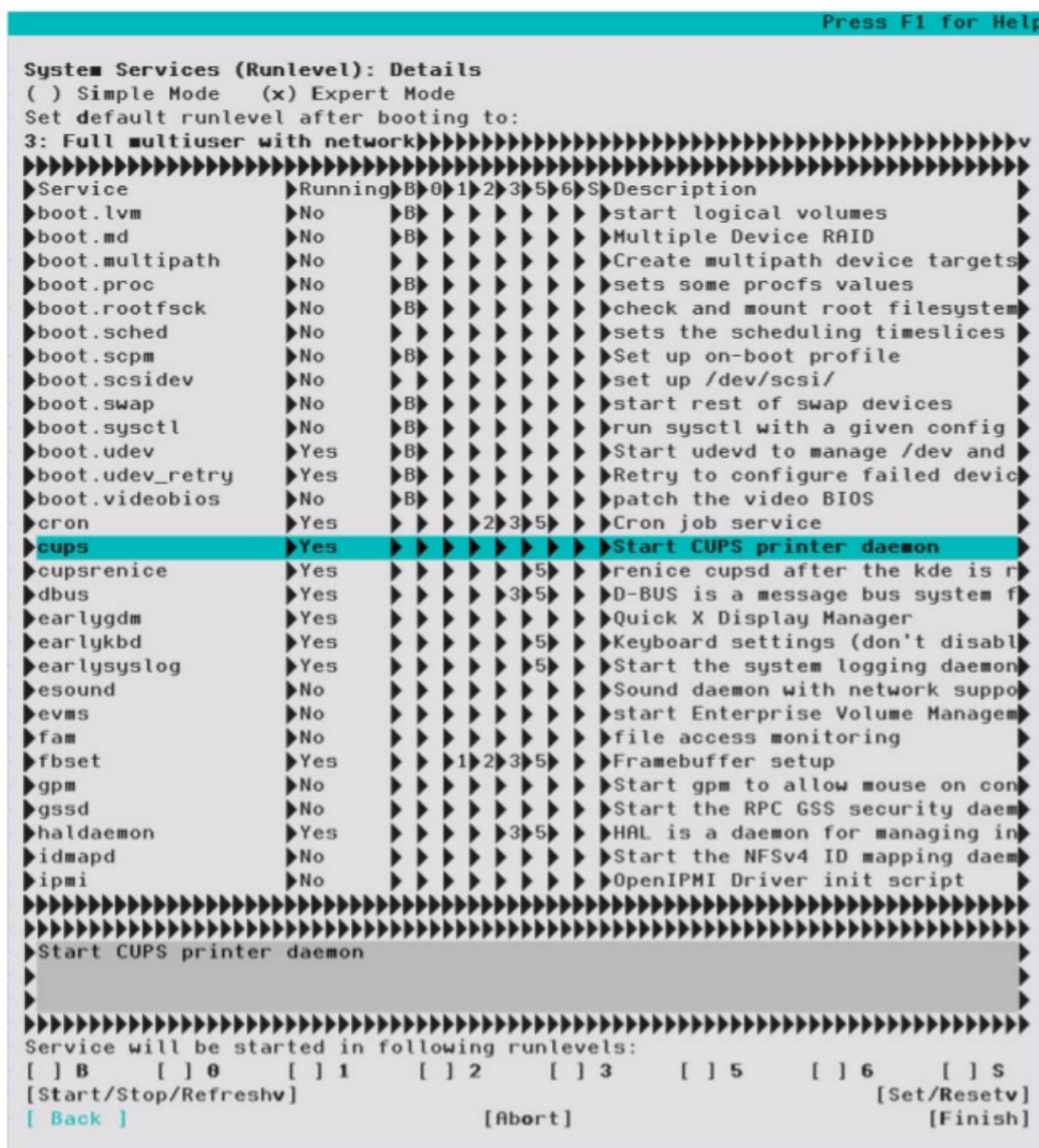


Figure 4-2 The System Services panel in YaST

在上面的图中，开启了专家模式，就可以设置服务基于各个运行级别来开启或者关闭。

修改runlevels

无论何时，不要在Linux服务器上运行GUI图形桌面。正如所有Linux管理员向你愉快保证的那样，GUI在Linux上完全没有必要。所有的任务都可以通过命令行轻松高效的完成，或者通过重定向X显示，或者通过浏览器。如果你离不开图形窗口，有几个有用的基于web的工具可以尝试，例如webmin，Linuxconf和SWAT。

小贴士：即使服务器本地禁用了GUI，你还是可以通过远程连接使用GUI。在ssh中使用-X参数。

如果必须要使用GUI，根据需要开启或者关闭，而不是让它一直运行。大多数情况下，服务器应该运行在3级别，机器开机时不会启动X服务。如果想重启X服务，从命令行使用startx。

1. 使用runlevel查看系统的运行级别

会显示出之前和当前的运行级别。例如，N 5意思是之前运行级别（N），当前运行级别是5。

2. 使用init命令在各个运行级别中切换，例如，使用init 3进入运行级别3。

Linux的运行级别如下：

0 停止（不要把initdefault设置为这个级别，否则系统启动之后会立刻关机）

1 单用户模式

2 多用户，无NFS（和没有网络的3级别一样）

3 全功能多用户模式

4 未使用，保留

5 X11

6 重启（也不要把initdefault设置为这个级别，机器会在启动的时候不断持续的重启）

设置系统启动初始运行级别的办法是，修改/etc/inittab文件，如下行：

```
id:3:initdefault:
```

```

... (lines not displayed)

# The default runlevel is defined here
id:3:initdefault:                                To start Linux without starting
                                                    the GUI, set the run level to 3.

# First script to be executed, if not booting in emergency (-b) mode
si::bootwait:/etc/init.d/boot

# /etc/init.d/rc takes care of runlevel handling
#
# runlevel 0 is System halt (Do not use this for initdefault!)
# runlevel 1 is Single user mode
# runlevel 2 is Local multiuser without remote network (e.g. NFS)
# runlevel 3 is Full multiuser with network
# runlevel 4 is Not used
# runlevel 5 is Full multiuser with network and xdm
# runlevel 6 is System reboot (Do not use this for initdefault!)
#

... (lines not displayed)

# getty-programs for the normal runlevels
# <id>:<runlevels>:<action>:<process>
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
#3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
#
#S0:12345:respawn:/sbin/agetty -L 9600 ttyS0 vt102
...
... (lines not displayed)

```

Figure 4-3 /etc/inittab, modified (only part of the file is displayed)

限制本地终端

默认情况下，系统有好几个本地虚拟终端。虽然虚拟终端所使用的内存是可以忽略不计的；但是我们锱铢必较。减少运行的进程数量可以简化排错过程和进程分析，这就是为什么把本地终端限制到2。

做法是注释掉不要的tty行。如上图所示，本地终端数量限制为2。如果你正在使用的shell被杀死了，你还有一个备用的终端。

SELinux

RHEL4引进了一项重要的安全措施，SELinux（Security Enhanced Linux）。SELinux引入的强制策略模型解决了标准的Linux访问模型的局限性。SELinux在用户和进程级别增强安全，所以，如果任何进程出现问题，只会影响到分配给该进程的资源，而不会是整个系统。

SELinux的工作很像虚拟机。例如，如果恶意攻击者利用了apache的提权漏洞，只有分配给Apache守护进程的资源可以被攻击者利用。

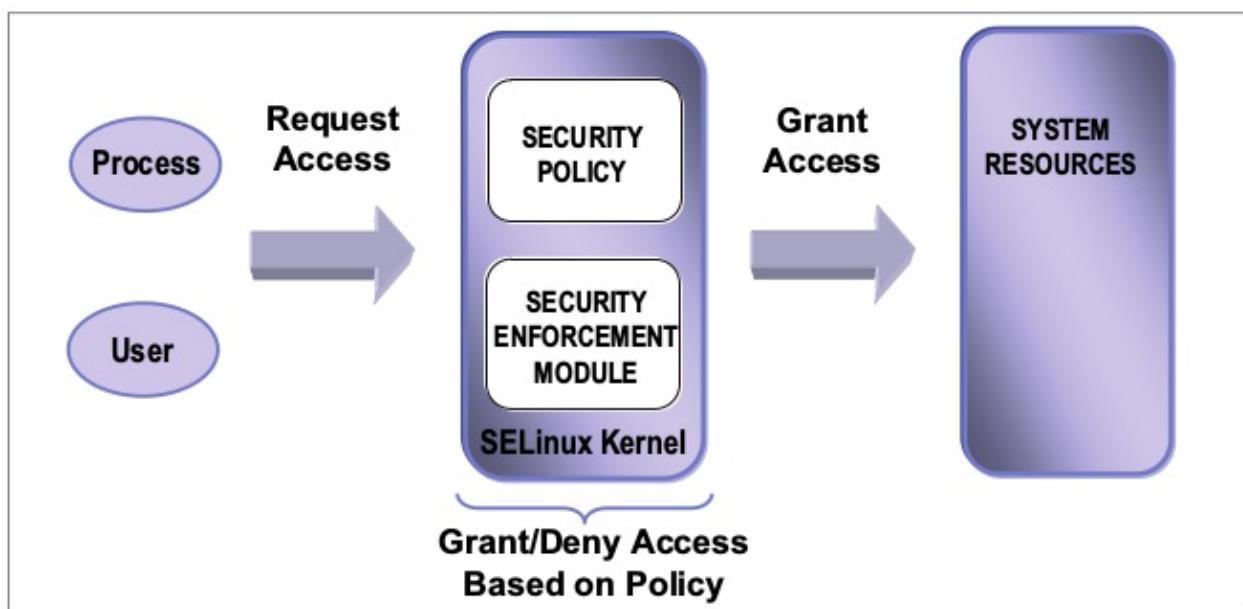


Figure 4-4 Schematic overview of SELinux

当然，使用这种强制安全策略也有代价，任何进程或者用户在访问系统资源（例如I/O设备）的时候都受到SELinux的控制。确认权限许可的过程可能会导致10%的损耗。在对外服务器上使用SELinux是很有必要的，例如防火墙或者Web服务器，但是在后端的数据库服务器上使用SELinux可能就会白白的损失性能而已。

通常，禁用SELinux最好的办法就是在最开始就不要安装它。但是，系统一般都是已经默认安装好了，而且没有考虑到SELinux会影响到性能。在GRUB boot loader的当前运行内核行中添加`selinux=0`就可以禁用掉已经安装的SELinux，如下图。

Example 4-3 Sample grub.conf file with disabled SELinux

```
default=0
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux AS (2.6.9-5.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-5.ELsmp ro root=LABEL=/ selinux=0
    initrd /initrd-2.6.9-5.ELsmp.img
```

另一个禁用SELinux的办法是修改SELinux的配置文件`/etc/selinux/config`，修改过后的该文件如下图：

```
SELINUX=disabled
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
```

如果你决定在你的服务器上使用SELinux，它的设置可以帮你更好的适应你的系统环境。在运行中的服务器，检查工作集的LSM（Linux Security Modules，Linux安全模块）缓存是否超过了默认AVC（Access Vector Cache，访问向量缓存）的512项。

在`/selinux/avc/hash_stats`中检查最大的链长度，任何超过10的都可能是瓶颈。

提示：你也可以使用`avcstat`工具来检查访问向量缓存的使用统计。

编译内核

创建和编译自己的内核对提升性能的作用其实没有人们所认为的那么大。现代大多数发行版所携带的内核都已经模块化---只加载需要的部分。重新编译内核可以降低内核的大小，改变整体的行为。修改源代码中的某些内核参数或许也可以节省一些性能。然而，非标准的内核不受商业Linux发行版的订阅支持。此外，为商业Linux发行版提供的ISV应用和IBM硬件认证，在检测到非标准内核的时候也会无效。

可以通过定制内核改善性能，但是，在企业环境中运行不受支持的内核也面临这很大的挑战。在商业环境确实如此，如果是科学计算领域，在要承受高性能计算任务的场景下，定制内核可能是很有帮助的。

在编译内核的时候不要指定`-C09`这类特殊编译器标志。Linux内核源代码已经是针对GNU的C编译器优化过的。使用特殊的编译器标志，最好的情况是降低内核性能，最差的情况下可能会破坏代码。

保持警惕，除非你真的知道自己在做什么，否则你添加的参数可能是在降低系统性能。

调整内核参数

- 调整内核参数

- 内核参数存储路径
- 使用`sysctl`命令

尽管不推荐用户修改和重编译内核代码，但还是有另外调整内核参数的手段。`proc`文件系统提供了观察和修改运行中的内核的接口。

要观察当前内核配置，找到`/proc/sys/`目录中某个内核参数文件，使用`cat`查看内容。在下图中，我们修改系统的内存过量策略。输出0表示系统在对应用程序分配内存前先检查可用内存量。我们通过`echo`命令为该参数提供一个新值，修改内核的这个默认操作，在这里，把0修改为1，1表示内核直接分配内存，不去检查是否有充足的内存可以分配。

Example 4-5 Changing kernel parameters via the proc file system

```
[root@linux vm]# cat overcommit_memory
0
[root@linux vm]# echo 1 > overcommit_memory
```

从上面的演示中可以看到，使用`echo`和`cat`可以很方便修改内核参数，几乎在所有带`/proc`文件系统的操作系统上都可以使用，但是有两个明显的问题。

- `echo`命令不能检查参数的一致性。
- 在重启系统之后，所有的内核修改都会丢失。

为了解决上面的问题，管理员应该使用`sysctl`来修改内核参数。

提示：默认，系统提供了必要模块，让你使用`sysctl`而不用重启系统。如果你在安装系统的时候去掉这项功能，你就必须重启Linux使得`sysctl`的修改生效。

另外，RHEL和SUSE提供图形界面修改`sysctl`参数。如下图

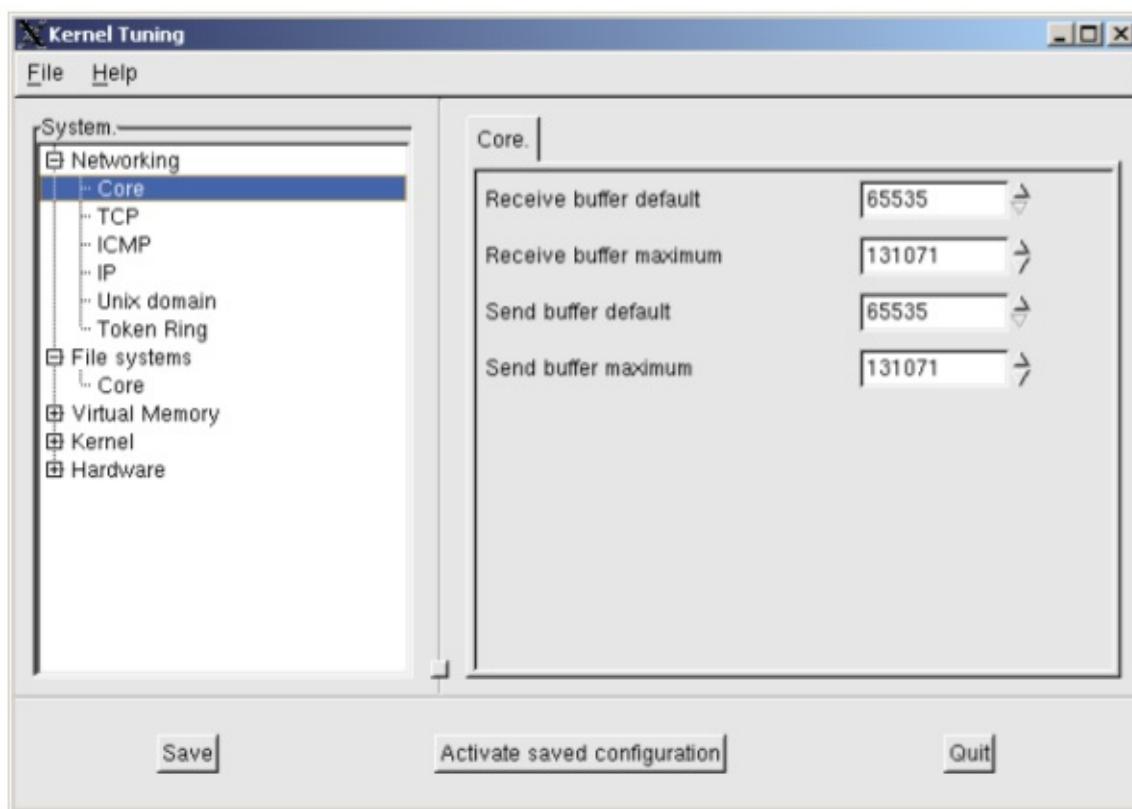


Figure 4-5 Red Hat kernel tuning

在基于SUSE的系统，YaST的powertweak优化工具可以用来调优内核参数。

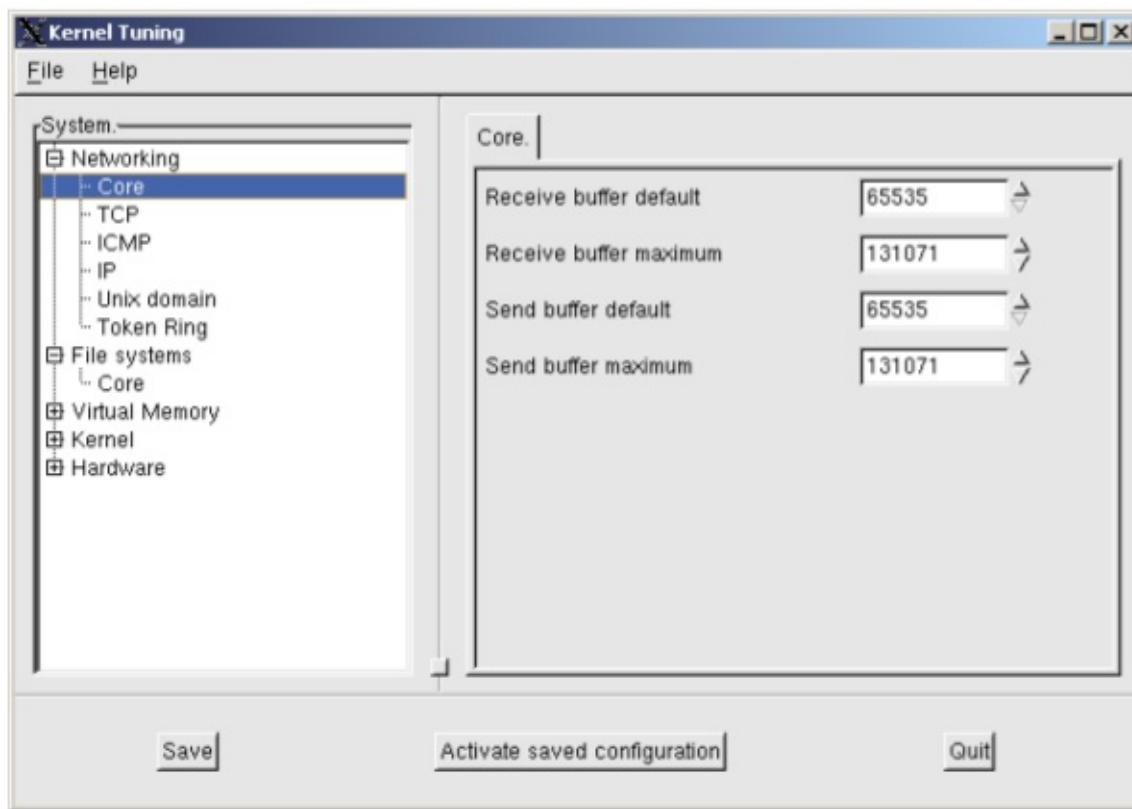


Figure 4-5 Red Hat kernel tuning

使用powertweak的好处是，所有调优参数都有简短的说明。所有使用powertweak做出的修改都会存在/etc/powertweak/tweaks文件中。

内核参数存储路径

内核参数存储在/proc（一般来说是/proc/sys）目录下。

通过/proc目录树下的文件，可以简单的了解与内核、进程、内存、网络以及其它组件相关的参数配置。每一个进程在/proc目录下都有一个以它的PID命令的目录。下表是部分文件所包含内核信息说明。

文件/目录	作用
/proc/sys/abi/*	用于提供对外部二进制的支持，比如在类UNIX系统，SCO UnixWare 7、SCO OpenServer和SUN Solaris 2上编译的软件。默认情况下是安装的，也可以在安装过程中移除。
/proc/sys/fs/*	设置系统允许的打开文件数和配额等。
/proc/sys/kernel/*	可以启用热插拔、操作共享内存、设置最大的PID文件数和syslog中的debug级别。
/proc/sys/net/*	优化网络，IPV4和IPV6
/proc/sys/vm/*	管理缓存和缓冲

使用**sysctl**命令

sysctl使用/proc/sys目录树中的文件名作为参数。例如，shmmmax内核参数保存在/proc/sys/kernel/shmmmax中，你可以使用cat来读取、echo来修改：

```
#cat /proc/sys/kernel/shmmmax
33554432
#echo 33554430 > /proc/sys/kernel/shmmmax
#cat /proc/sys/kernel/shmmmax
33554430
```

然而，使用echo很容易弄错，所以我们推荐使用**sysctl**命令，因为它会修改前检查数据一致性，如下：

```
#sysctl kernel.shmmmax
kernel.shmmmax = 33554432
#sysctl -w kernel.shmmmax=33554430
kernel.shmmmax = 33554430
#sysctl kernel.shmmmax
kernel.shmmmax = 33554430
```

上面的修改会在下次重启之后消失。如果你想做永久修改，你应该编辑/etc/sysctl.conf文件，添加参数如下：

4.3. 调整内核参数

```
kernel.shmmax = 33554439
```

下次重启系统的时候会读取/etc/sysctl.conf文件，你也可以通过如下的命令，不用重启就让配置生效：

```
# sysctl -p
```

优化处理器子系统

- 优化处理器子系统
 - 进程优先级
 - CPU中断处理
 - 考虑NUMA系统

在所有计算机中，无论手持设备还是科学计算集群，最重要的都是用来做计算的处理器。在摩尔定律的影响下，处理器子系统比其它所有子系统的发展都要快很多。结果是系统瓶颈很少出现在CPU上，除非系统是专门做运算的。通过一个Intel处理器系统的平均CPU使用率低于10%就很容易得出这个结论。理解处理器瓶颈和如何优化CPU性能参数是很重要的。

进程优先级

前面已经说过，我们不可能修改某个进程的优先级。唯一的办法是间接的通过调整进程nice级别，但这也并非总是可行。如果某个进程运行太慢，你就可以调低它的nice级别，给它更多的CPU时间。当然，这就意味着其它进程拥有更低的处理器时间，将会变得更慢。

Linux支持的nice级别从19（最低级别）到-20（最高级别）。默认值是0。需要使用root权限，才能把进程的nice级别设置为负数（更高的优先级）。

以nice级别-5启动xyz程序，使用如下命令：

```
sudo nice -n -5 xyz
```

修改运行中的程序优先级，使用如下命令：

```
renice level pid
```

把PID为2500的进程优先级设置为10：

```
renice 10 2500
```

CPU中断处理

在中断处理中，如下两个原则是十分有效的：

- 把产生大量中断的进程绑定到一个CPU上。

CPU亲和力使得管理员可以把中断绑定到某个或者某组物理处理器上（当然，这对单CPU系统无效）。要更改某个IRQ的亲和力，进入到`/proc/irq/%{irq号码}/`目录，修改`smp_affinity`文件的CPU mask值。把19号IRQ的亲和力设置到第三个CPU的命令如下（不适用SMT）：

```
echo 03 > /proc/irq/19/smp_affinity
```

- 让物理处理器处理中断

在对称多线程（SMT，symmetric multi-threading）系统中，例如IBM POWER 5+处理器支持多线程，它就推荐将中断处理绑定到物理处理器上，而不是SMT实例。在双路多线程系统中，物理处理器通常都是较低的CPU数，CPU 编号 0和2通常是物理CUP，而1和3通常是指多线程实例。

如果你不使用`smp_affinity`标志，你就没必要担心这个了！

考虑NUMA系统

非统一内存架构（NUMA，Non-Uniform Memory Architecture）系统正在获取市场份额，被看做是传统统一多处理系统的自然演化。尽管当前Linux发行版的CPU调度也同样适用于NUMA系统，但是应用程序可能就不一定了。由non-NUMA导致的性能降低是很难识别的。可以使用numactl包中的numastat工具找到在NUMA架构上有问题进程。

numastat统计数据保存在`/sys/devices/system/node/%{node number}/%/numastat`文件中，可以用来帮助找到瓶颈。`numa_miss`和`other_node`字段偏高可能是MUMA引起的。如果你发现给进程分配的内存不是在进程的本地节点上（运行程序的处理器节点），尝试对这个进程renice或者采用NUMA。

优化虚拟内存子系统

- 优化虚拟内存子系统
 - 设置kernel swap和pdflush
 - Swap分区
 - HugeTLBfs

优化内存子系统需要持续的观察，确保所做出的修改不会对服务器上的其它子系统产生不良影响。如果你要修改/proc/sys/vm下的虚拟内存参数，建议你一次只修改一个参数，然后观察服务器性能变化。

Linux上的大多数应用并不直接把数据写入磁盘，它们先写入由虚拟内存管理器维护的文件系统缓存中，最后刷入数据到磁盘。如果你使用了IBM ServerRAID控制器或者IBM TotalStorage磁盘子系统，你应该降低刷入的频率，增加每次I/O的效率。因为，高性能磁盘控制器处理大的I/O流比多个小I/O流更加高效。

Linux内核2.6的改进虚拟内存子系统，管理员可以使用简单的接口调整内核的交换操作。

- /proc/sys/vm/swappiness 中的参数可以用来定义如何把内存页交换到磁盘。前面已经介绍了Linux的虚拟内存管理器和Linux的swap空间的大致用法。Linux会把一段时间没有使用的内存移动到是swap中，即使在还有可用内存的情况下。通过修改/proc/sys/vm/swappiness中的百分比，你可以控制swap行为。如果不希望发生交换，/proc/sys/vm/swappiness应该是很小的值。在存限制严格的运行批量任务的系统（程序休眠很长时间）上，交换行为是很有用的。可以使用echo和sysctl命令改变交换行为。

```
# sysctl -w vm.swappiness=100
```

- 特别是对于快的磁盘子系统，可能会希望发生脏内存页大刷新。/proc/sys/vm/dirty_background_ratio中定义了达到内存的占用百分比时，pdflush守护进程把数据写入到磁盘。如果希望更大写入，把默认值增加到10%，值越大，写入的频率越低。可是使用如下的命令操作：

```
# sysctl -w vm.dirty_background_ratio=25
```

- 另外一个和虚拟内存子系统相关的设置是应用程序生成的脏页被刷入磁盘的百分比。前面章节已经说过，写入行为并不是直接发生，而是先写入页缓存，以后再刷入到磁盘子系统。/proc/sys/vm/dirty_ratio中是主内存的百分比。10的意思是，在文件系统缓存达到

服务器内存的10%时，数据会写入系统。和之前的例子一样，设置写入磁盘脏页面百分比的20%的方法是：

```
# sysctl -w vm.dirty_ratio=20
```

Swap分区

在物理内存用完，系统需要额外的内存的时候，就会使用swap。系统还会把一段时间内没有使用的内存存放到swap空间。当系统上没有空闲内存，会把最少使用的内存换出到磁盘中。swap空间在系统安装的时候创建，大小应该是物理内存的两倍。Linux内核2.4支持每个分区的swap最大为24G，32位系统上，理论的最大swap是8TB。交换分区应该分散在不同的磁盘中。

如果服务器上添加了内存，也应该配置额外的swap空间。在系统安装完成之后，还有如下的办法配置swap空间。

- 磁盘上空闲的分区可以创建为swap分区。如果磁盘子系统上没有空闲的地方，会比较难办，那种情况下，我们可以创建swap文件。
- 如果可以选择，swap分区是优于swap文件的。swap分区的I/O可以绕过文件系统，节省了写入文件的开销，比写入swap文件的性能更好。

另外一种提升swap分区和文件性能的办法是建立多个swap分区。Linux可以并行的读写多个swap分区。在创建了额外的swap分区或文件之后，/etc/fstab可能包含如下的内容：

Example 4-10 /etc/fstab file

/dev/sda2	swap	swap	sw	0 0
/dev/sdb2	swap	swap	sw	0 0
/dev/sdc2	swap	swap	sw	0 0
/dev/sdd2	swap	swap	sw	0 0

在正常的情况下，Linux首先把/dev/sda2作为swap分区，然后是/dev/sdb2，依次往下等等，直到具有足够的swap空间。这意味着，如果不需要更大的swap空间，那可能只有/dev/sda2被使用。

因为所有的读写是同时对所有选定的分区的，所有把数据分散在多个swap分区可以提升性能。按照下图修改/etc/fstab文件，给前三个swap分区分配更高的优先级。

Example 4-11 Modified /etc/fstab to make parallel swap partitions

/dev/sda2	swap	swap	sw,pri=3	0 0
/dev/sdb2	swap	swap	sw,pri=3	0 0
/dev/sdc2	swap	swap	sw,pri=3	0 0
/dev/sdd2	swap	swap	sw,pri=1	0 0

swap分区是从最高优先级到最低优先级使用的（32767优先级最高，0的优先级最低）。给头三个磁盘分配相同的优先级，数据就会写入这三个磁盘，系统不会等第一个磁盘写满才使用第二个。由于系统并行的使用3个**swap**分区，会提升整体的性能。

只有在前三个**swap**分区都满了之后，还需要额外的**swap**，系统才会使用第四个**swap**分区/**dev/sdd2**。也可以把给所有的分区分配相同的优先级，把数据分散在所有**swap**分区，但面临的问题是，如果其中一个分区很慢比其它分区慢，性能会降低。最好的办法是，在最快的磁盘上配置**swap**分区。

重要：尽管这是优化内存子系统的手段，但是应该尽力避免经常的page out。**swap**分区不是内存的替代品，因为**swap**的速度会比内存慢很多。所以，page out（或者**swap** out）不是什么好事情。在调整**swap**之前，首先确定服务器上内存是充足的，而且没有内存泄露发生。

HugeTLBfs

对使用大量虚拟地址空间的应用来说，内存管理是很有用的。尤其是对数据库应用。

CPU的旁路转换缓冲是保存虚拟到物理映射信息的小缓存区。使用TLB，可以不用参考内存页表的虚拟地址映射就能完成转换。然而，为了保证转换速度够快，TLB通常很小。经常有大内存的应用超过了TLB的映射能力。

HugeTLBs特性让应用使用比通常更大的页大小，所以。单个TLB条目可以映射到更大的地址空间。**HugeTLB**条目的大小不一样。例如，在Itanium 2系统中，大页可以是正常页的1000倍。意味着TLB映射的虚拟地址空间是正常进程的1000倍，而不会TLB缓存未命中。简单起见，这个功能是通过文件系统接口向应用程序开放的。

可以通过配置/**proc/sys/vm/nr_hugepages**的值，设置hugepage的个数来分配hugepage

```
sysctl -w vm.nr_hugepages=512
```

如果应用程序使用**mmap()**系统调用使用大页，你必须挂载**hugetlbfs**类型的文件系统：

```
mount -t hugetlbfs none /mnt/hugepages
```

/**proc/meminfo**会提供**hugetlb**页的信息，如下图：

Example 4-12 Hugepage information in /proc/meminfo

```
[root@lnxsu4 ~]# cat /proc/meminfo
MemTotal:      4037420 kB
MemFree:       386664 kB
Buffers:        60596 kB
Cached:         238264 kB
SwapCached:     0 kB
Active:         364732 kB
Inactive:       53908 kB
HighTotal:      0 kB
HighFree:       0 kB
LowTotal:       4037420 kB
LowFree:        386664 kB
SwapTotal:      2031608 kB
SwapFree:       2031608 kB
Dirty:          0 kB
Writeback:      0 kB
Mapped:         148620 kB
Slab:           24820 kB
CommitLimit:   2455948 kB
Committed_AS:  166644 kB
PageTables:     2204 kB
VmallocTotal:  536870911 kB
VmallocUsed:   263444 kB
VmallocChunk:  536607255 kB
HugePages_Total: 1557
HugePages_Free: 1557
Hugepagesize:  2048 kB
```

更多详细信息，请参考内核文档：[/vm/hugetlbpage.txt](#)

优化磁盘子系统

- 优化磁盘子系统
 - 安装前的硬件考虑
 - I/O elevator的调整和选择
 - 文件系统的选择和优化

最终，所有数据都会保存到磁盘中。磁盘访问通常是毫秒级的，而且比其它组件（例如，内存和PCI操作，都是纳秒或微秒级别的）至少慢上千倍。Linux文件系统就是数据在磁盘上存储和管理的方式。

Linux上有性能和扩展性各异的各类文件系统可以选择。除了保存和管理磁盘上的数据，文件系统还得保证数据的完整性。最新的Linux发行版都把日志文件系统作为默认组件。日志可以避免在系统崩溃的时候，数据不一致。所有对文件系统元数据的修改都保存在一个分割的日志中，在系统崩溃之后，可以用来恢复到数据一致的状态。日志还会节约恢复时间，因为系统重启的时候不用检查文件系统。和其它问题一样，你必须在性能和完整性之间做一个平衡。然而，进入到企业数据中心和商业环境的Linux服务器，会被提出高可用的要求。

作为对各类文件系统的补充，Linux内核2.6有4种I/O调度算法，可以在不同任务和场景下使用。每个I/O elevator都有不同的特性，可能只合适特定的硬件配置和任务。有些elevator采用流式I/O，通常用在多媒体或者桌面PC环境中，有些elevator则偏重于低延时需要的数据库一类工作。

在本节中，我们学习ReiserFS和Ext3这类基本文件系统的行为和优化方法，以及kernel 2.6中I/O elevator的优化潜力。

安装前的硬件考虑

在Linux发行版文档中，通常指明安装该操作系统的CPU速度要求和最小内存，以及完成安装所需的最小磁盘空间！但是没有说明如何初始化安装磁盘子系统。Linux服务器可能工作在任何环境中，所以第一个问题是：这个服务器是用来做什么事情的？

磁盘子系统会影响到整个服务器的性能，决定I/O是否会成为性能瓶颈的关键是理解服务器的用途。

在下面的例子中，I/O是最重要的子系统：

- 文件和打印服务器需要把文件快速在用户和磁盘间转移。因为文件服务器是目的是把文件传给客户端，所以服务器必须首先从磁盘中读取所有数据。

+ 数据库服务器的最终目标是从磁盘上的仓库里搜索和检索数据。及时内存足够，大多数数据库服务器还是会执行把大量磁盘I/O，把数据记录读入内存或者把修改写入磁盘。

以下场景中，磁盘I/O不是最重要的系统：

- 邮件服务器中，服务器是一个电子邮件的仓库和路由器，瓶颈通常在通信负载。网络比较重要。
- web服务器是处理Web页面（静态，动态，或者二者皆有）的，对其影响较大的是网络和内存。

驱动器数量

磁盘驱动的数量会显著影响性能，因为每个驱动器都会增加系统的总带宽。通常大家都只考虑到磁盘的容量需求，吞吐量需求通常不那么被看重，或者直接被忽视。优化磁盘性能的关键是增加I/O请求的读写磁头数量。

在RAID（redundant array of independent disks，独立磁盘冗余阵列）技术中，你可以把I/O分散到多个磁盘。Linux下有两种RAID：软件RAID和硬件RAID。除非服务器硬件支持硬件RAID，否则你只能选择软RAID。如果需求增加，你可以进一步选择效率更高的硬件RAID方案。

如果需要实施硬件RAID阵列，首先你需要RAID控制器。在这种情况下，磁盘子系统由多块磁盘和控制器一起组成。

贴士：通常，增加驱动器是改善服务器性能的最有效办法。

记住，磁盘子系统的性能是由给定设备所能处理的输入输出请求数决定的。一旦操作系统或磁盘缓存不能容纳读写请求的数据大小，磁盘的物理主轴就开始工作。比如下面的例子。某磁盘一秒能处理200个I/O。而你的应用执行4KB的随机写操作，所以流和请求合并都不适合。磁盘的最大吞吐计算方法是：

$$\text{物理磁盘每秒I/O数量} * \text{请求大小} = \text{最大吞吐量}$$

按照这个计算方法，上面例子中的结果是：

$$200 * 4 \text{ KB} = 800 \text{ KB}$$

800 KB是物理最大值，改善性能的办法是增加更多磁盘或者让应用一次写入更大的数据。在DB2数据库中，可以配置使用更大的请求大小，大多数情况下这都提升磁盘吞吐。

分区

分区是把磁盘上一些相邻块作为独立磁盘使用。现在的Linux发行版中，默认情况下就可以通过划分逻辑卷来动态分区。

Linux中，关于最佳磁盘分区有很大的争论。只分根分区的做法可能导致问题，因为不能满足的重新定义的新需求。太多磁盘分区又可能导致文件系统管理的困难。在安装过程中，Linux会提供创建多个分区的选项。

在Linux上划分多分区或者逻辑卷的好处：

- 通过更细粒度的文件系统属性增强安全。

例如，`/var`和`/tmp`分区是系统上所有用户和进程有权限访问的，很容易发生恶意行为。如果有为他们设置单独的分区，在这些分区需要重建或者恢复的时候可能减小对系统的影响。

- 加强数据完整性，磁盘崩溃导致的数据丢失只会影响相关的分区。

例如，如果系统上没有做软硬RAID，服务器磁盘崩溃了，只有坏磁盘上的分区需要替换和恢复。

- 可以在不影响更多静态分区的情况下安装和升级系统。

例如，如果`/home`没有单独分区，将会在系统升级的时候被覆盖掉，会丢失存储的所有用户文件。

- 更高效率的备份过程

在设计分区布局的时候需要充分考虑备份工具。了解备份工具是基于分区还是更细粒度，比如说文件系统。

下表中列出了可能需要从根分区独立的分区，可以提升灵活度和更好的性能。

分区	内容和服务器环境
<code>/home</code>	文件服务环境中最好单独给 <code>/home</code> 分区，这是系统上所有用户的 <code>home</code> 目录，这个分区上的空间很容易塞满，所以隔离这个目录，避免对整个磁盘空间造成太大影响
<code>/tmp</code>	如果服务器运行在高性能计算环境中，计算过程中需要很多临时空间，在计算完成后释放
<code>/usr</code>	该目录下存放了内核源码树和Linux文档，以及大量的可执行文件。 <code>/usr/local</code> 目录下存放了系统上所有用户都可以访问的可执行文件，也是存放自己的脚本的最佳目录。如果将 <code>/usr</code> 单独分区，在重新安装和升级的过程中只要不重新格式分区，文件都不需要重新安装。
<code>var</code>	在mail、Web和打印服务环境中， <code>/var</code> 分区很重要，因为它保存了这些应用日志，以及所有的系统日志。数据会慢慢的填满这个分区。如果这个分区不是独立的，在 <code>/var</code> 被填满的时候，服务可能会受到影响。根据运行的服务不同，可以把邮件服务器的 <code>/var/spool/mail</code> 和 <code>/var/log</code> 单独划分
<code>/opt</code>	第三方软件，比如Oracle的数据库通常是在这个分区中。如果没有分区，会被安装到 <code>/</code> 分区，如果没有足够的空间，会安装失败

关于Linux如何处理文件系统的更多信息，查看文件系统层级的标准主页：

<http://www.pathname.com/fhs>

I/O elevator的调整和选择

随着Linux内核2.6新的I/O调度算法，系统在处理不同类型I/O的时候有了更高的灵活性。系统管理员要为指定的硬件和软件选择最合适的elevator。此外，每个I/O elevator有一组调优选项，可以把系统朝着特定的需求调整。

选择正确的I/O elevator

对于多数服务器负载，CFQ（Complete Fair Queuing）elevator和deadline elevator都是不错的选择，都为典型多用户、多进程服务器环境做过优化。企业发行版通常默认是CFQ elevator。在IBM System z的Linux中，默认的是deadline 调度。特定的环境应该选择特定的I/O elevator。在RHEL 5.0和Novell SUSE Linux Enterprise Server 10，可以为单独的磁盘子系统设置I/O调度器，而在它们的上一版本中，只能做系统全局设置。由于可以为每个磁盘子系统设置I/O elevator，管理员可以在磁盘子系统上隔离不同的I/O类型（例如密集型负载），并且选择合适的elevator 算法。

- 同步文件系统访问

某些类型应用需要同步的文件系统操作。比如在数据库应用中可能用上raw文件系统，或者缓存异步磁盘访问的非常大的磁盘子系统上。在这种情况下，默认的elevator通常具有最低的吞吐和最高的时延。在大约16KB的场景下，其它三个算法的表现差不多好，之后，CFQ和NOOP开始具有比deadline更高的性能（除非磁盘访问需求竞争十分强烈），结果如下图。

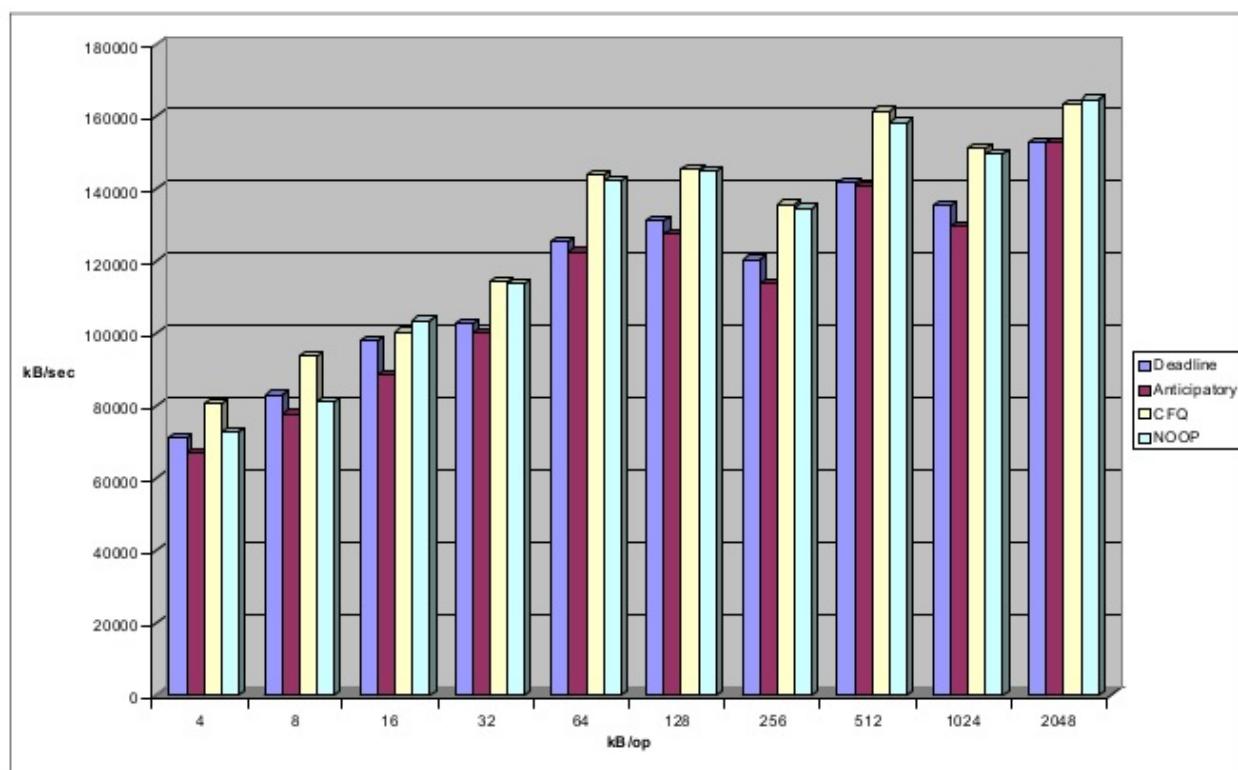


Figure 4-7 Random read performance per I/O elevator (synchronous)

- 复杂磁盘子系统

测试表明，NOOP elevator在高端服务器环境是个有意思的选择。在使用ServerRAID或TotalStorage DS等复杂的磁盘配置时，NOOP elevator的无序性成为了它的优势。企业类磁盘子系统可能包含多个SCSI或者光纤磁盘，每个都包含独立的磁盘头和数据带。在这种复杂环境中，I/O elevator很难正确预测I/O行为，所以，你可能会看到使用NOOP /IO elevator获得同等性能，只产生较少的开销。数百块磁盘的大型的基准测试，大多数使用NOOP elevator。

- 数据库系统

由于数据库系统的自然寻址特性，选择deadline elevator可以获得较好性能。

- 虚拟机

虚拟机，无论在VMware还是System z，大多数都是通过一个虚拟层和底层硬件进行通信。所以，虚拟机意识不到分配的磁盘是单个SCSI盘还是TotalStorage DS8000上的一组光纤通道磁盘。虚拟层负责必要的I/O重排序以及和物理块设备的通信。

- 绑定CPU的应用

有些I/O调度器可以提供卓越的吞吐量，同时也产生更大的系统开销。CFQ和deadline所产生的开销来自于对I/O queue的聚合和排序。有时CPU、性能对系统的限制大于磁盘性能的影响。比如说在科学计算或者处理数据仓库非常复杂的查询的领域。在这种场景下，NOOP elevator比其它elevator有更多的优势，因为它只产生较少的CPU负载，如下图。但是，需要注意的是，从吞吐量和CPU负载来看，deadline和CFQ依旧是大多数异步文件系统的最好选择。

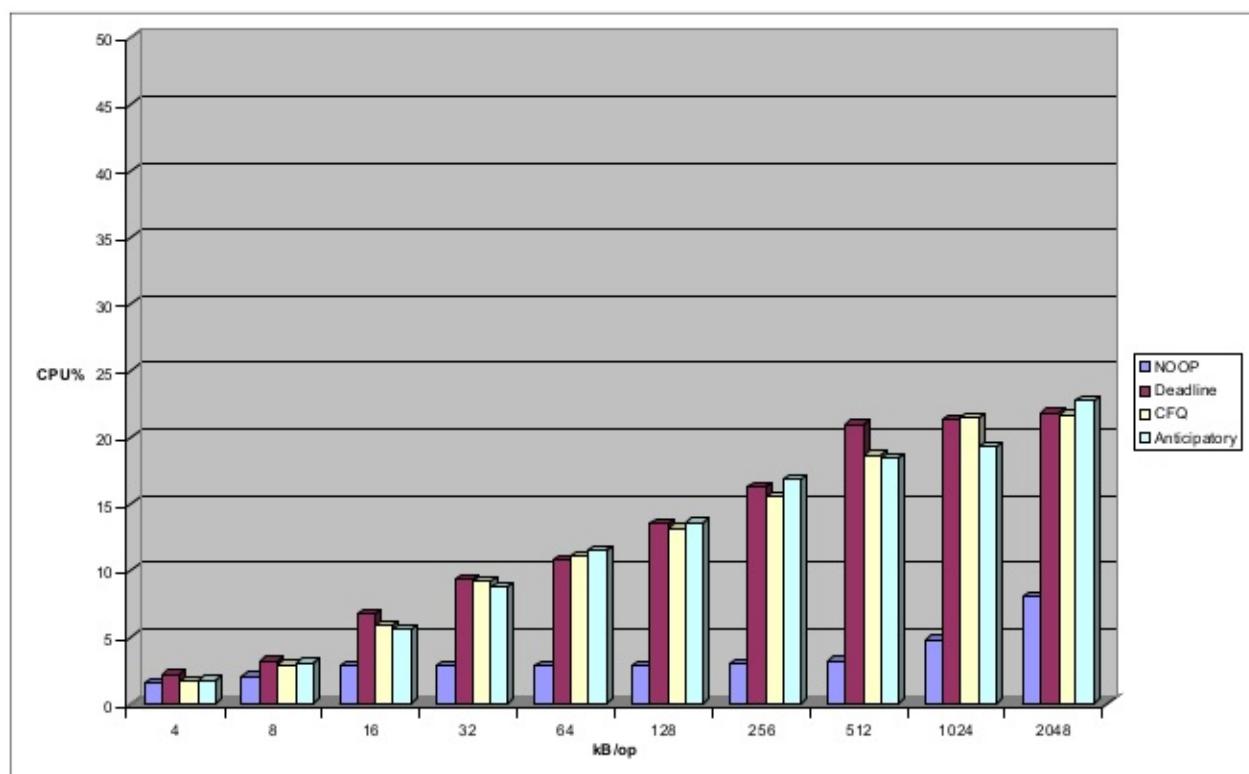


Figure 4-8 CPU utilization by I/O elevator (asynchronous)

- 单ATA或SATA磁盘子系统

如果你使用的是单个ATA或者SATA磁盘，考虑使用预期I/O elevator吧，重新排序写I/O，以适应设备的单磁头。

nr_request

内核2.6的可插拔I/O调度器实现还有增加和减少向磁盘发起请求数量的功能。`nr_requests`和许多其它优化参数一样，没有一个最佳的设置。正确的取值更多的是依赖于底层磁盘系统，而不是I/O负载行为。`request_nr`的不同取值产生的影响，也不如不同组件和I/O调度器所产生的影响如上面的图那样显而易见。如下图所示，`nr_requests`对deadline elevator的影响比对CFQ elevator的影响要小。

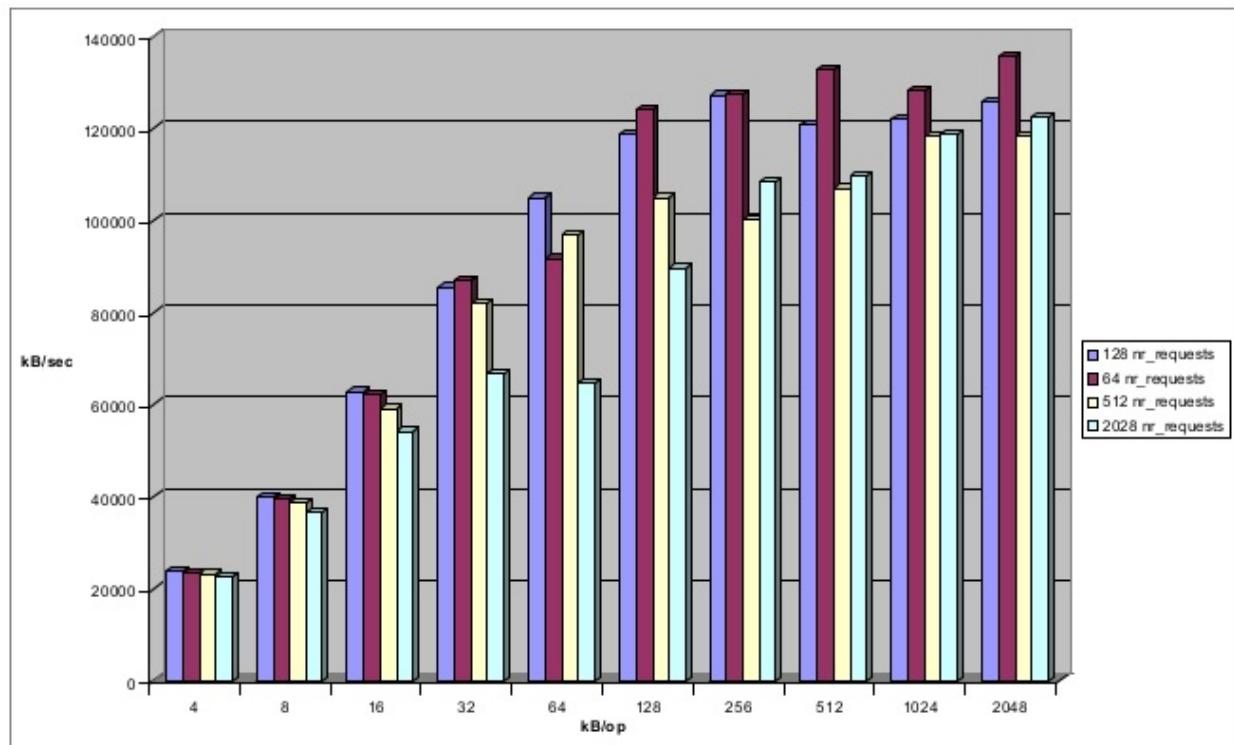


Figure 4-9 Impact of `nr_requests` on the Deadline elevator (random write ReiserFS)

大请求队列可能会为写入大量小文件的负载提供很高的吞吐量。如下图所示，设置8192提供最高级别性能的I/O大小是16KB。在I/O大小为64KB的时候，`nr_requests`值从64到8192都是差不多的性能。随着I/O大小的增加，低级别的`nr_requests`通常都会产生卓越性能。可以使用如下命令修改请求数：

```
# echo 64 > /sys/block/sdb/queue/nr_requests
```

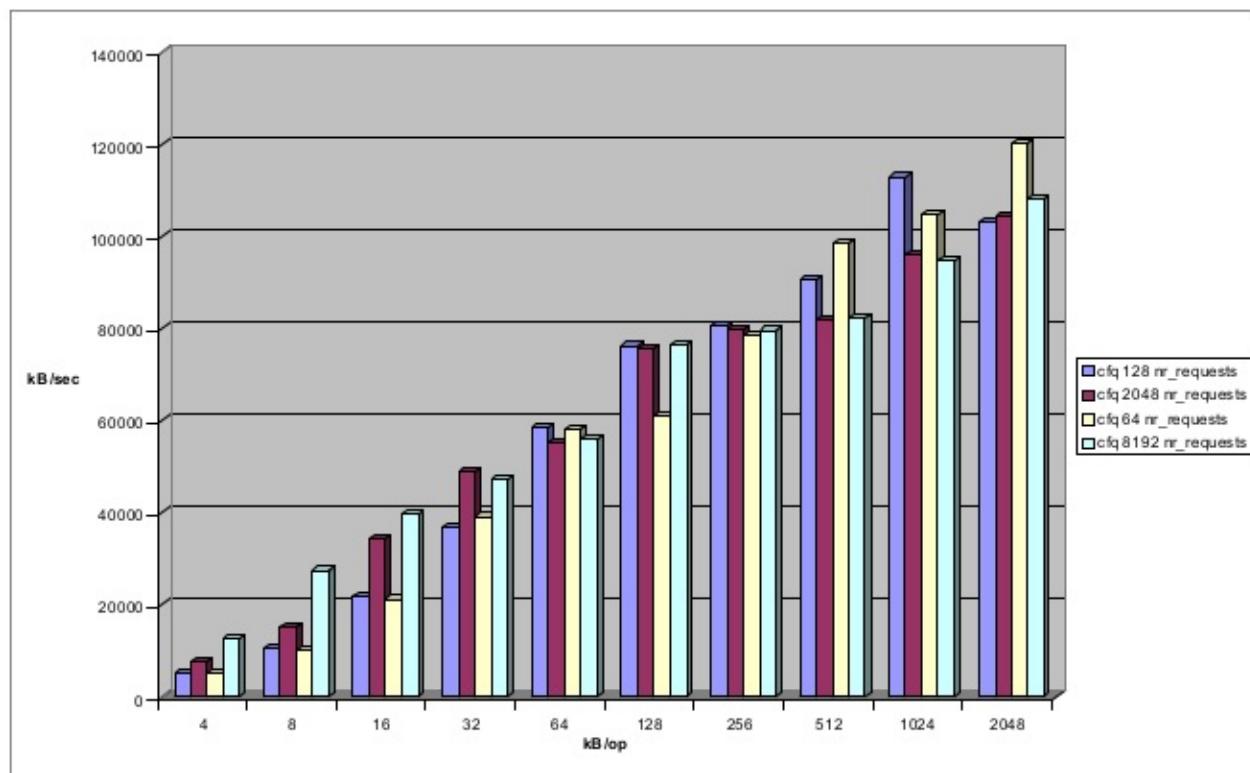


Figure 4-10 Impact of nr_requests on the CFQ elevator (random write Ext3)

重要的是，当前企业版Linux提供了基于单个磁盘子系统的nr_requests选项。所以，可以针对不同I/O模式分别优化调整。例如在数据库系统中，日志和数据应该存放在专门的磁盘或者磁盘子系统。在这个例子中，应该给日志分区设置很大的nr_requests值，用来执行很多的小尺寸写I/O，给数据分区设置很小的nr_requests值，可能就会看到128KB大的读I/O。

贴士：衡量和计算平均I/O大小的方法，参考前面的iostat。

read_ahead_kb

在大的读I/O流中，增加提前读取缓冲的大小会提升性能。注意，因为大多数都是随机I/O的原因，所以在一般情况下，增加这个值不会增强性能。read_ahead_kb定义了预读多少数据。这个值保存在/sys/block//queue/read_ahead_kb中，单位是KB。可以使用cat或者echo命令修改：

```
# cat /sys/block/<disk_subsystem>/queue/read_ahead_kb
# echo 64 > /sys/block/<disk_subsystem>/queue/read_ahead_kb
```

文件系统的选择和优化

前面已经说过，不同类型的文件系统适用于不同的工作负载场景。如果你有选择文件系统的自由，你应该好好考察下Ext、JFS、ReiserFS、XFS，比较看哪种比较符合你的需求。一般来说，ReiserFS适用于很小的I/O请求，XFS和JFS适合很大的文件系统和很大的I/O大小。Ext3的能力恰好在ReiserFS和JFS/XFS之间，它能处理小I/O请求，并且提供很好的多处理扩展性。

JFS和XFS最适合高端数据仓库，科学计算和大的SMP服务器，以及流媒体服务器。

ReiserFS和Ext3主要用在文件、Web和邮件服务器等场景。对于写频繁、而且I/O大小在64KB以下的环境，ReiserFS相比Ext3的优势在默认的日志模式上，如下图。但是，这仅仅是在同步文件操作的时候。

Ext2也是一个选择。由于不具备日志功能，无论是访问模式，还是I/O大小，Ext2在同步文件系统上的表现都优于ReiserFS和Ext3。所以，在性能比数据完整性还重要的时候，Ext2也是个不错的选择。

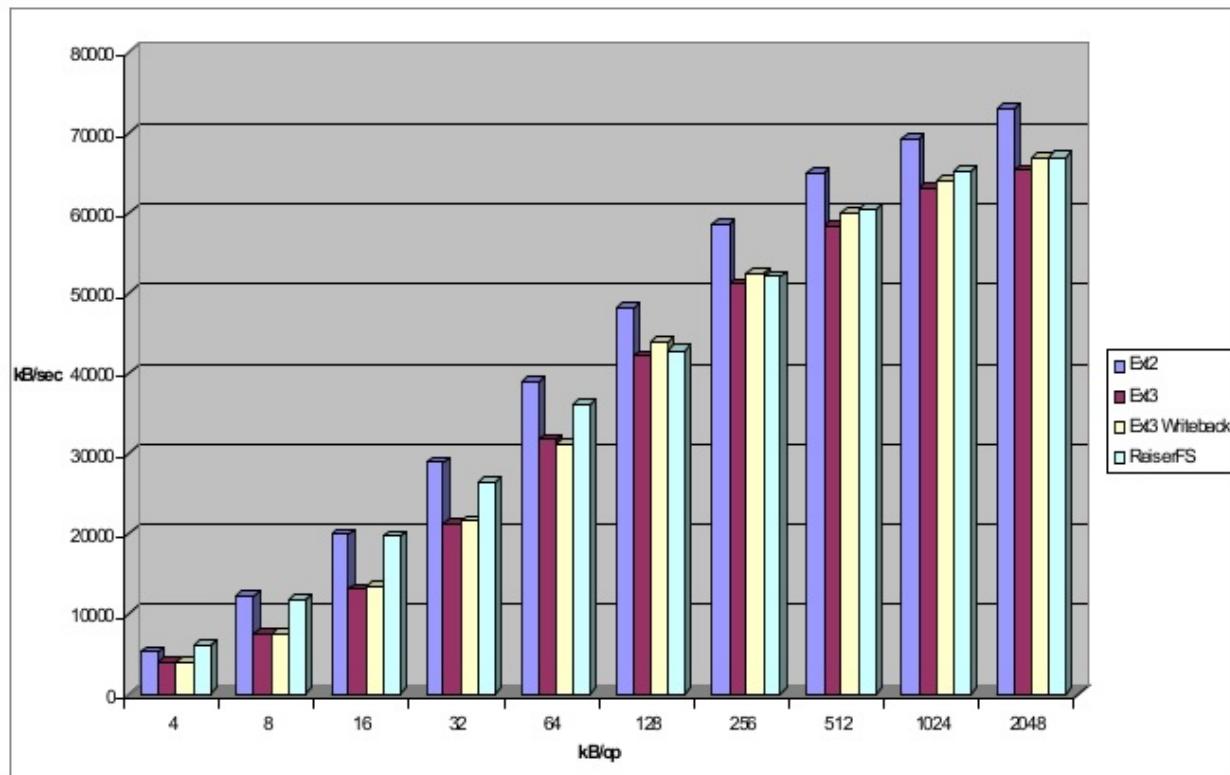


Figure 4-11 Random write throughput comparison between Ext and ReiserFS (synchronous)

在大多数同步文件系统场景中，默认日志模式的ReiserFS文件系统（数据有序）通常提供比Ext3更好的性能。但是，在默认日志模式为调整为writeback的时候，Ext3和ReiserFS具有同等的性能，如下图。

使用ionice分配I/O优先级

CFQ I/O elevator的一个新特性是可以在进程级别分配优先级。ionice可以限制特定进程的磁盘利用率。在撰写本文的时候，可以使用ionice分配三种优先级：

- **Idle**：分配Idle优先级的进程只有在没有其它进程被分配best-effort或者更高访问级别的时候，才能访问磁盘。对某些进程，只有在系统有空闲资源的时候才能运行，如updatedb，这个设置就很有用。
- **Best-effort**：所有没有指定I/O优先级的进程，默认都会分配这个优先级。有8（0~7）种有限级别，数值越小，级别越高。

- **Real time**：最高的I/O优先级是**real time**，意味着进程总是可以有限访问磁盘子系统。
real time也有8个级别。需要注意的是，给某个进程分配了**real time**级别之后，可能导致其它进程处于饥饿状态。

ionice有如下的选项：

- c <#> I/O 优先级1是**real time**，2是**best-effort**，3是**idle**
- n <#> I/O 优先级别从0到7
- p <#> 正在运行进程的进程号，不使用-p，开启一个使用该优先级的任务

下面的例子中，给PID为113的进程分配**idle** I/O优先级。

```
# ionice -c3 -p113
```

更新访问时间

Linux文件系统会记录文件何时被创建、更新和访问。在读写文件的时候会更新**last-time-read**的属性。由于写操作很浪费，取消不必要的I/O可以提升整体性能。但是，在大多数情况下，禁用文件访问时间更新只会带来很小的性能提升，

使用**noatime**选项挂载文件系统可以取消更新**inode**访问时间。如果文件或者目录的更新时间不是很重要，比如在**web**服务器上，管理员可能会在/etc/fstab挂载选项中添加**noatime**标志。禁用访问时间更新可以获得0%~10%的性能提升，可以减轻3%的文件服务器负载。

```
/dev/sdb1 /mountlocation ext3 defaults,noatime 1 2
```

注意：通常，给/var独立分区，并且设置**noatime**极好的做法。

选择文件系统的日志模式

大多数文件系统都支持如下三种和日志相关的参数，也就是**mount**中的**data**选项。但是，由于日志模式对**Ext3**文件系统的性能影响最大，所以我们建议对**Red Hat**的默认文件系统参数做调整：

- **data=journal**

这个日志选项通过记录数据和元数据，提供给最高的数据一致性。也带来最大的性能损耗。

- **data=ordered (default)**

在这个模式下，只会记录元数据。文件数据优先写入。这是默认的设置。

- **data=writeback**

该选项以数据一致性为代价，提供了最快的数据访问。虽然，通过记录元数据来保证数据一致性。但是，没有对真正的文件数据做任何处理，如果系统崩溃，旧数据可能出现在文件在文件中。使用**writeback**模式的元数据日志记录方式，可以和默认的**ReiserFS**、**JFS**、**XFS**相媲美。**writeback**日志模式增强了**Ext3**的性能，尤其是对小I/O，如下图所示。随着I/O大小的增长，回写模式的优势变小。注意，日志模式只影响写入性能。如果主要是读数据的负载（如**web**服务器），那么修改日志模式也没有用。



有三种修改日志模式的办法：

- 在**mount**中指定：

```
mount -o data=writeback /dev/sdb1 /mnt/mountpoint
```

- 在**/etc/fstab**文件中指定：

```
/dev/sdb1 /testfs ext3 defaults,data=writeback 0 0
```

- 如果你想修改根分区默认的**data=ordered**选项，先在**/etc/fstab**中做上面的修改，然后执行**mkinitrd**命令扫描**/etc/fstab**文件中的修改，创建新的镜像。更新**grub**或者**lilo**指向新的镜像。

块大小

块大小是读写到磁盘的最小单位，会直接影响到服务器性能。如果你的服务器是需要处理很多小文件的，那么，把块大小设置小一点，效率会比较高。如果服务器是用来处理大文件的，就应该把块大小设置大一点，性能会比较高。不能对正在运行的服务器设置块大小，只能在重新格式化的时候修改。大多数Linux发行版允许块大小为1K、2K和4K。测试表明，修改块大小并不能给文件系统带来多少性能的提升，所以，一般保持为默认的4K就好了。

如果使用了硬件**RAID**方案，一定要仔细考虑阵列的条带大小（在光纤通道中是段）。**stripe-unit size**是在数据被存储在阵列中的一个驱动器，后续数据被存储在阵列的下一个驱动器上的间隔。选择正确的条带大小需要理解特定应用的请求大小。硬件阵列的条带大小和文件系统的块大小一样，都会限制影响整体磁盘性能。

流和连续内容通常会比较倾向于大的条带大小，可以减少磁头寻址时间，提升吞吐量。但是在数据库这类随机活动中，条带大小设置成和记录大小一致，可以获得较好的性能。

优化网络子系统

- 优化网络子系统
 - 流量行为
 - 速度和双工
 - MTU大小
 - 增加网络缓冲
 - 额外的TCP/IP调整
 - 防火墙的影响
 - 卸载配置
 - 增加包队列
 - 增加发送队列长度
 - 减少中断

在系统安装之初和感觉到网络子系统瓶颈的时候，我们应该对其进行优化。子系统间可能导致相互的影响：例如，网络问题可以影响到CPU利用率，尤其当包大小特别小的时候；当TCP连接数太多的时候，内存使用率会变高。

流量行为

在网络性能调优中，最重要的是尽可能准确的理解网络流量模式。性能很大程度上取决于流量特征。

例如，下面的两张图是使用netperf收集的，说明了流量模式对性能的影响。两张图唯一的区别是流量类型，第一张图是TCP_RR的结果，第二张图是TCP_CRR类型的流量。性能差别主要是由TCP session连接和关闭操作带来的损耗，以及Netfilter连接跟踪导致的。

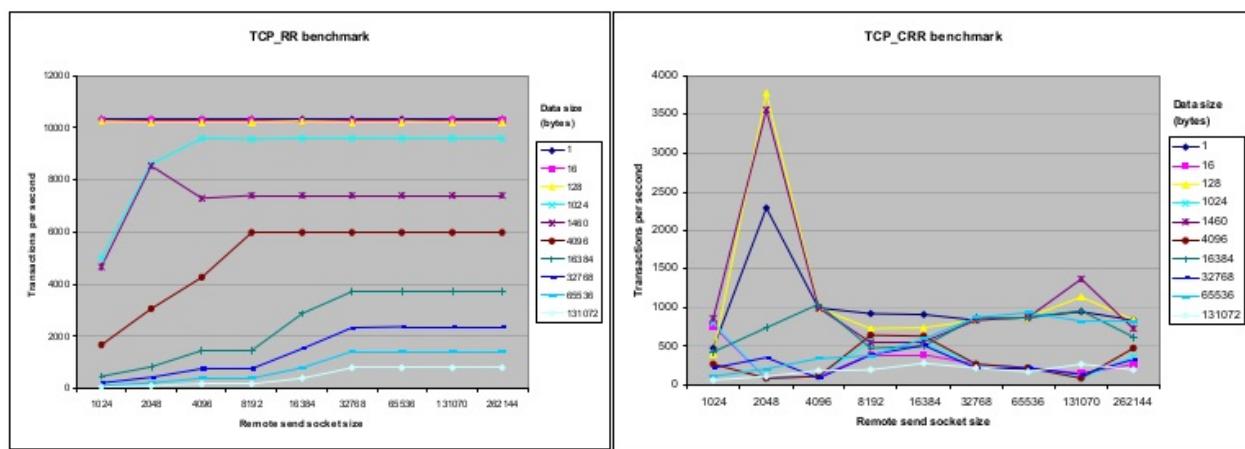


Figure 4-14 An example result of netperf TCP_RR and TCP_CRR benchmarks

如图所示，在相同的配置的情况下，即使细微的流量行为差别也能导致巨大的性能差别。请熟悉下面的网络流量行为和要求：

- 事务吞吐需求（峰值和平均值）
- 数据传输的吞吐需求（峰值和平均值）
- 延时需求
- 传输数据大小
- 接收和发送的比例
- 连接建立和关闭的频率，还有连接并发数
- 协议（TCP，UDP，应用协议，比如HTTP，SMTP，LDAP等等）

使用netstat，tcpdump，ethereal等工具可以获得准确的行为。

速度和双工

增强网络性能的最简单办法是检查网卡的实际速度，找到网络组件（比如交换机或hub）和网卡之间的问题。错误的配置可以导致很大的性能问题，如下图：

Example 4-17 Using ethtool to check the actual speed and duplex settings

```
[root@linux ~]# ethtool eth0
Settings for eth0:
  Supported ports: [ MII ]
  Supported link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Half 1000baseT/Full
  Supports auto-negotiation: Yes
  Advertised link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Half 1000baseT/Full
  Advertised auto-negotiation: Yes
  Speed: 100Mb/s
  Duplex: Full
```

从下图可以看出来，在网速被错误协商后，小数据传输比大数据传输受到的影响小。大于1KB的数据传输受到极大影响（吞吐减少50~90%）。一定要确保速度双工配置正确！

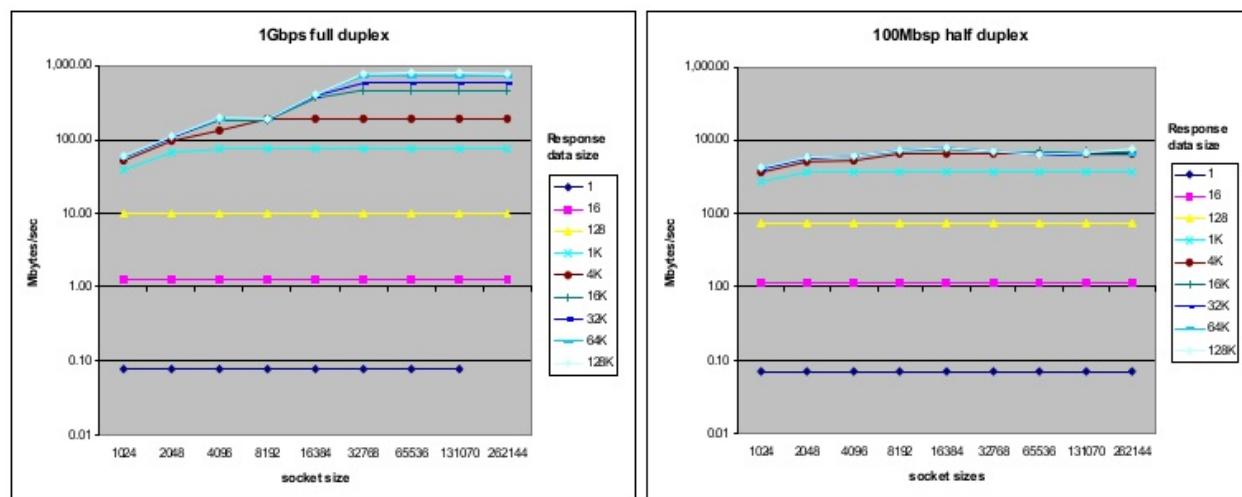


Figure 4-15 Performance degradation caused by auto negotiation failure

许多网络设备在自动协商中错误的默认为100Mb半双工模式，使用ethtool检查网络真实的速度和双工设置。

许多网络专家认为，最好的办法是在网卡和交换机或者hub上同时指定静态速度。只要设备驱动支持ethtool命令，可以使用ethtool修改配置。而有些设备可能要修改/etc/modules.conf的内容。

MTU大小

在Gb网络中，最大传输单元（maximum transmission units，MTU）越大，网络性能越好。问题是，太大的MTU可能不受大多数网络的支持，大量的网卡不支持大MTU。如果要在Gb速度上传输大量数据（例如HPC环境），增加默认MTU可以导致明显的性能提升。使用/sbin/ifconfig修改MTU大小。

Example 4-18 Changing the MTU size with ifconfig

```
[root@linux ~]# ifconfig eth0 mtu 9000 up
```

要使大的MTU值正常生效，这个值必须同时在网卡和网络组件上受到支持。

增加网络缓冲

Linux网络栈在分配内存资源给网络缓冲时十分谨慎！在服务器所连接的现代高速网络中，如下的参数应该增大，使得系统能够处理更多网络包。

- TCP的初始内存大小是根据系统内存自动计算出来的；可以在如下文件中找到这个值：

```
/proc/sys/net/ipv4/tcp_mem
```

- 调高默认以及最大接收Socket的内存值：

```
/proc/sys/net/core/rmem_default  
/proc/sys/net/core/rmem_max
```

- 调高默认和最大发送Socket的内存值：

```
/proc/sys/net/core/wmem_default  
/proc/sys/net/core/wmem_max
```

- 调高最大缓存值：

```
/proc/sys/net/core/optmem_max
```

调整窗口大小

可以通过上面的网络缓冲值参数来优化最大窗口大小。可以通过BDP（时延带宽积，bandwidth delay product）来获得理论上的最优窗口大小。BDP是导线中的传输的数据量。BDP可以使用如下的公式计算得出：

```
BDP = Bandwidth (bytes/sec) * Delay (or round trip time) (sec)
```

要使网络管道塞满，达到最大利用率，网络节点必须有和BDP相同大小的缓冲区。另一方面，发送方还必须等待接收方的确认，才能继续发送数据。

例如，在1ms时延的GB级以太网中，BDP等于：

```
125Mbytes/sec (1Gbit/sec) * 1msec = 125Kbytes
```

在大多数发行版中，`rmem_max`和`wmem_max`的默认值都是128KB，对一般用途的低时延网络环境已经足够。然而，如果时延很大，这个默认值可能就太小了！

再看看另外一个例子，假如一个samba文件服务器要支持来自不同地区的16个同时在线的文件传输会话，在默认配置下，平均每个会话的缓冲大小就降低为8KB。如果数据传输量很大，这个值就会显得很低了！

- 把所有协议的系统最大发送缓冲（`wmem`）和接收缓冲（`rmem`）设置为8MB。

```
sysctl -w net.core.wmem_max=8388608
sysctl -w net.core.rmem_max=838860
```

指定的这个内存值会在TCP socket创建的时候分配给每个TCP socket。

- 此外，还得使用如下的命令设置发送和接收缓冲。分别指定最小、初始和最大值。

```
sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"
```

第3个值必须小于或等于`wmem_max`和`rmem_max`。在高速和高质量网络上，建议调大第一个值，这样，TCP窗口在一个较高的起点开始。

- 调高`/proc/sys/net/ipv4/tcp_mem`大小。这个值的含义分别是最小、压力和最大情况下分配的TCP缓冲。

可以使用`tcpdump`查看哪些值在socket缓冲优化中被修改了。如下图所示，把socket缓冲限制在较小的值，导致窗口变小，引起频繁的确认包，会使网络效率低下。相反，增加套接字缓冲会增加窗口大小。

Example 4-19 Small window size (rmem, wmem=4096)

```
[root@lnxsu5 ~]# tcpdump -ni eth1
22:00:37.221393 IP plnxsu4.34087 > plnxsu5.32837: P 18628285:18629745(1460) ack 9088 win 46
22:00:37.221396 IP plnxsu4.34087 > plnxsu5.32837: . 18629745:18631205(1460) ack 9088 win 46
22:00:37.221499 IP plnxsu5.32837 > plnxsu4.34087: . ack 18629745 win 37
22:00:37.221507 IP plnxsu4.34087 > plnxsu5.32837: P 18631205:18632665(1460) ack 9088 win 46
22:00:37.221511 IP plnxsu4.34087 > plnxsu5.32837: . 18632665:18634125(1460) ack 9088 win 46
22:00:37.221614 IP plnxsu5.32837 > plnxsu4.34087: . ack 18632665 win 37
22:00:37.221622 IP plnxsu4.34087 > plnxsu5.32837: P 18634125:18635585(1460) ack 9088 win 46
22:00:37.221625 IP plnxsu4.34087 > plnxsu5.32837: . 18635585:18637045(1460) ack 9088 win 46
22:00:37.221730 IP plnxsu5.32837 > plnxsu4.34087: . ack 18635585 win 37
22:00:37.221738 IP plnxsu4.34087 > plnxsu5.32837: P 18637045:18638505(1460) ack 9088 win 46
22:00:37.221741 IP plnxsu4.34087 > plnxsu5.32837: . 18638505:18639965(1460) ack 9088 win 46
22:00:37.221847 IP plnxsu5.32837 > plnxsu4.34087: . ack 18638505 win 37
```

Example 4-20 Large window size (rmem, wmem=524288)

```
[root@lnxsu5 ~]# tcpdump -ni eth1
22:01:25.515545 IP plnxsu4.34088 > plnxsu5.40500: . 136675977:136677437(1460) ack 66752 win 46
22:01:25.515557 IP plnxsu4.34088 > plnxsu5.40500: . 136687657:136689117(1460) ack 66752 win 46
22:01:25.515568 IP plnxsu4.34088 > plnxsu5.40500: . 136699337:136700797(1460) ack 66752 win 46
22:01:25.515579 IP plnxsu4.34088 > plnxsu5.40500: . 136711017:136712477(1460) ack 66752 win 46
22:01:25.515592 IP plnxsu4.34088 > plnxsu5.40500: . 136722697:136724157(1460) ack 66752 win 46
22:01:25.515601 IP plnxsu4.34088 > plnxsu5.40500: . 136734377:136735837(1460) ack 66752 win 46
22:01:25.515610 IP plnxsu4.34088 > plnxsu5.40500: . 136746057:136747517(1460) ack 66752 win 46

22:01:25.515617 IP plnxsu4.34088 > plnxsu5.40500: . 136757737:136759197(1460) ack 66752 win 46
22:01:25.515707 IP plnxsu5.40500 > plnxsu4.34088: . ack 136678897 win 3061
22:01:25.515714 IP plnxsu5.40500 > plnxsu4.34088: . ack 136681817 win 3061
22:01:25.515764 IP plnxsu5.40500 > plnxsu4.34088: . ack 136684737 win 3061
22:01:25.515768 IP plnxsu5.40500 > plnxsu4.34088: . ack 136687657 win 3061
22:01:25.515774 IP plnxsu5.40500 > plnxsu4.34088: . ack 136690577 win 3061
```

socket缓冲大小

在服务器处理许多大文件并发传输的时候，小socket缓冲可能引起性能降低。如下图所示，在小socket缓冲的情况下，导致明显的性能下降。在rmem_max和wmem_max很小的情况下，即使对方拥有充足的socket缓冲可用，还是会影响可用缓冲大小。这使得窗口变小，成为大数据传输时候的瓶颈。下图中没有包含小数据（小于4KB）传输的情况，实际中，小数据传输不会受到明显的影响。

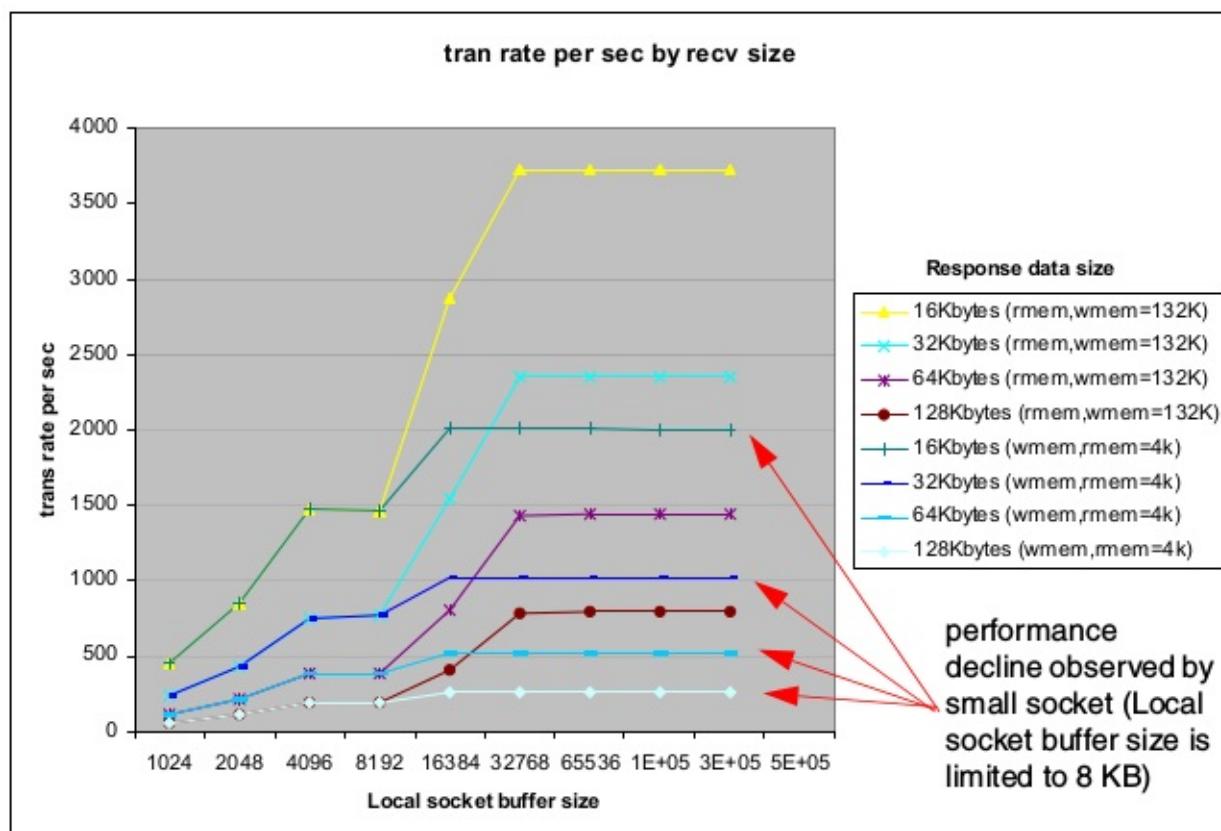


Figure 4-16 Comparison with socket buffer 4 KB and 132 bytes

额外的TCP/IP调整

还有很多其它增加或降低网络性能的配置。如下的参数可能会帮助提升网络性能。

优化IP和ICMP

如下的sysctl命令可以优化IP和ICMP：

- 禁用如下的参数可以阻止骇客进行针对服务器IP的地址欺骗攻击。

```
sysctl -w net.ipv4.conf.eth0.accept_source_route=0
sysctl -w net.ipv4.conf.lo.accept_source_route=0
sysctl -w net.ipv4.conf.default.accept_source_route=0
sysctl -w net.ipv4.conf.all.accept_source_route=0
```

- 如下的服务器配置用来忽略来自网关机器的重定向。重定向可能是攻击，所以我们值允许来自可信来源的重定向。

```
sysctl -w net.ipv4.conf.eth0.secure_redirects=1
sysctl -w net.ipv4.conf.lo.secure_redirects=1
sysctl -w net.ipv4.conf.default.secure_redirects=1
sysctl -w net.ipv4.conf.all.secure_redirects=1
```

- 你可以设置网卡是否接收ICMP重定向。ICMP重定向是路由器向主机传达路由信息的一种机制。例如，当路由器从某个接口收到发往远程网络的数据时，发现源ip地址与下一跳属于同一网段，这是路由器会发送ICMP重定向报文。网关检查路由表获取下一跳地址，下一个网关把网络包发给目标网络。使用如下的命令静止重定向：

```
sysctl -w net.ipv4.conf.eth0.accept_redirects=0
sysctl -w net.ipv4.conf.lo.accept_redirects=0
sysctl -w net.ipv4.conf.default.accept_redirects=0
sysctl -w net.ipv4.conf.all.accept_redirects=0
```

- 如果服务器不是网关，它就没必要发送重定向包，可以禁用如下的参数

```
sysctl -w net.ipv4.conf.eth0.send_redirects=0
sysctl -w net.ipv4.conf.lo.send_redirects=0
sysctl -w net.ipv4.conf.default.send_redirects=0
sysctl -w net.ipv4.conf.all.send_redirects=0
```

- 配置服务器忽略广播ping和smurf攻击：

```
sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=1
```

- 忽略所有类型的icmp包和ping：

```
sysctl -w net.ipv4.icmp_echo_ignore_all=1
```

- 有些路由器会发送错误的广播响应包，内核会对每一个包都会生成日志，这个响应包可以忽略：

```
sysctl -w net.ipv4.icmp_ignore_bogus_error_responses=1
```

- 还应该设置ip碎片参数，尤其是NFS和Samba服务器来说。可以设置用来做IP碎片整理的最大和最小缓冲。当以bytes为单位设置ipfrag_high_thresh的值之后，分配处理器会丢掉报文，直到达到ipfrag_low_thresh的值。

当TCP包传输中发生错误时，就会产生碎片。有效的数据包会放在缓冲中，而错误的包会重传。

例如，设置可用内存范围为256M到384M，使用：

```
sysctl -w net.ipv4.ipfrag_low_thresh=262144
sysctl -w net.ipv4.ipfrag_high_thresh=393216
```

优化TCP

这里讨论通过调整参数，修改TCP的行为。

如下的命令可以用来调整服务器支持巨大的连接数。

- 对于并发连接很高的服务器，TIME-WAIT套接字可以重复利用。这对web服务器很有用：

```
sysctl -w net.ipv4.tcp_tw_reuse=1
```

如果使用了上面的参数，还应该开启TIME-WAIT的套接字快速回收参数：

```
sysctl -w net.ipv4.tcp_tw_recycle=1
```

下图显示了在启用这些参数之后，连接数明显降低了。这有益于提升服务器性能，因为每个TCP连接都需要维护缓存，存放每个远端服务器信息。在这个缓存中会存放往返时间，最大分片大小和拥塞窗口。

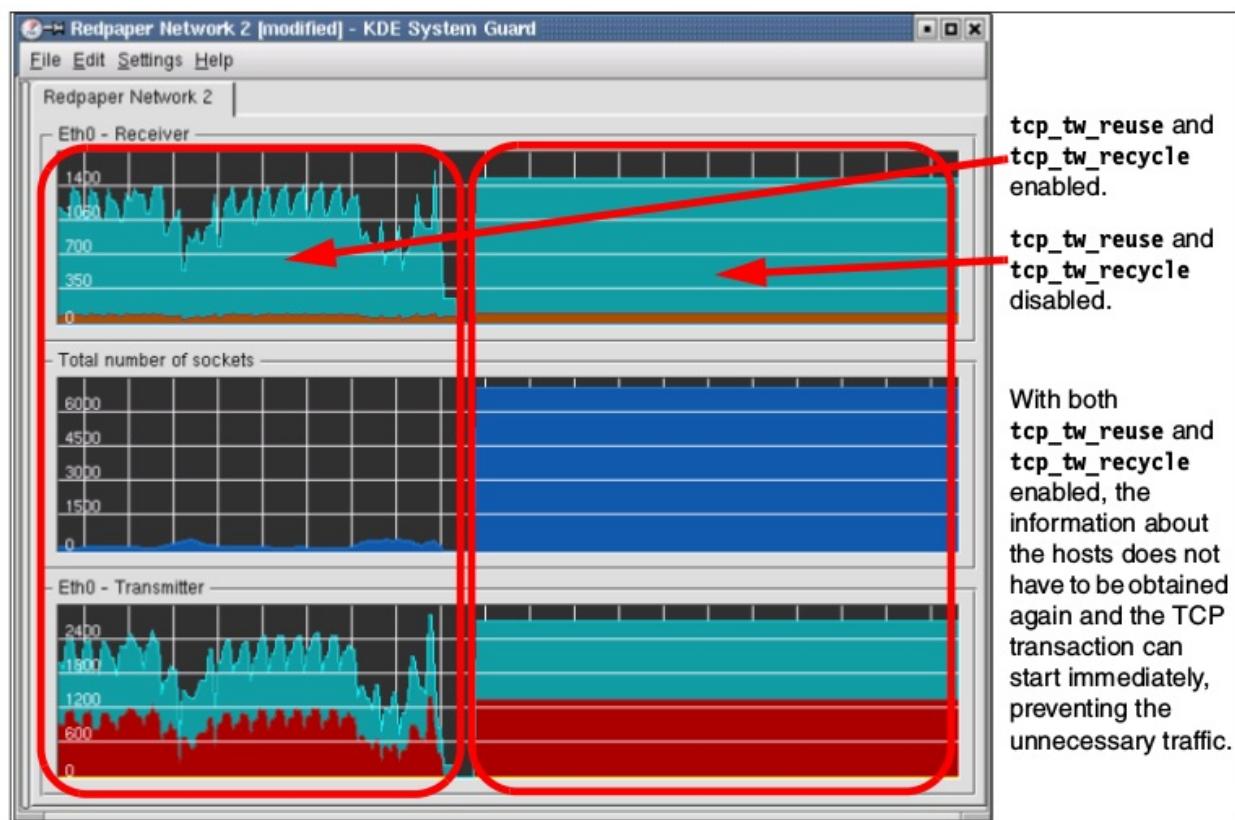


Figure 4-17 Parameters reuse and recycle enabled (left) and disabled (right)

- `tcp_fin_timeout`是socket在服务器上关闭后，socket保持在FIN-WAIT-2状态的时间。

TCP连接以三次握手同步syn开始，以3次FIN结束，过程中都不会传递数据。通过修改`tcp_fin_timeout`值，定义何时FIN队列可以释放内存给新的连接，由此提升性能。因为死掉的socket有内存溢出的风险，这个值必须在仔细的观察之后才能修改：

```
sysctl -w net.ipv4.tcp_fin_timeout=30
```

- 服务器可能会遇到大量的TCP连接打开着，却没有使用的问题。TCP有keepalive功能，探测这些连接，默认情况下，在7200秒（2小时）后释放。这个时间对服务器来说可能太长了，还可能导致超出内存量，降低服务器性能。

把这个值设置为1800秒（30分钟）：

```
sysctl -w net.ipv4.tcp_keepalive_time=1800
```

- 当服务器负载很高，拥有很多坏的高时延客户端连接，会导致半开连接的增长。在web服务器中，这个现象很常见，尤其是在许多拨号用户的情况下。这些半开连接保存在 backlog connections队列中。你应该该值最小为4096（默认是1024）。

即使服务器不会收到这类连接，也应该设置这个参数，他可以保护免受DoS（syn-flood）攻击。

```
sysctl -w net.ipv4.tcp_max_syn_backlog=4096
```

- TCP SYN cookies可以帮助保护服务器免受syn-flood攻击，无论是DoS（拒绝服务攻击，denial-of-service）还是DDoS（分布式拒绝服务攻击，distributed denial-of-service），都会对系统性能产生不利影响。建议只有在明确需要TCP SYN cookies的时候才开启。

```
sysctl -w net.ipv4.tcp_syncookies=1
```

注意，只有在内核编译了CONFIG_SYNCOOKIES选项的时候，上面的命令才是正确的
优化TCP选项

如下的TCP选项可以进一步优化Linux的TCP协议栈。

- 选择性确认可以在相当大的程度上优化TCP流量。然而，SACK和DSACK可能对Gb网络产生不良影响。tcp_sack和tcp_dsack默认情况下是启用的，但是和优化TCP/IP性能背道而驰，在高速网络上应该禁用这两个参数。

```
sysctl -w net.ipv4.tcp_sack=0
sysctl -w net.ipv4.tcp_dsack=0
```

- 每个发往Linux网络栈的以太帧都会收到一个时间戳。这对于防火墙、Web服务器这类系统是很有效且必要的，但是后端服务器可能会从禁用TCP时间戳，减少负载中获益。可以使用如下的命令禁用TCP时间戳：

```
sysctl -w net.ipv4.tcp_timestamps=0
```

- 我们已经知道缩放窗口可以增大传输窗口。然而，测试表明，窗口缩放对高网络负载的环境不合适。另外，某些网络设备不遵守RFC指导，可能导致窗口缩放故障。建议禁用窗口缩放，并且手动设置窗口大小。

```
sysctl -w net.ipv4.tcp_window_scaling=0
```

防火墙的影响

Netfilter提供TCP/IP连接跟踪和包过滤功能，在某些环境下，可能对性能产生很大影响。在连接数很高时，Netfilter的影响很明显。下图展示了测试不同连接情况下的结果，清楚表明了Netfilter的影响。

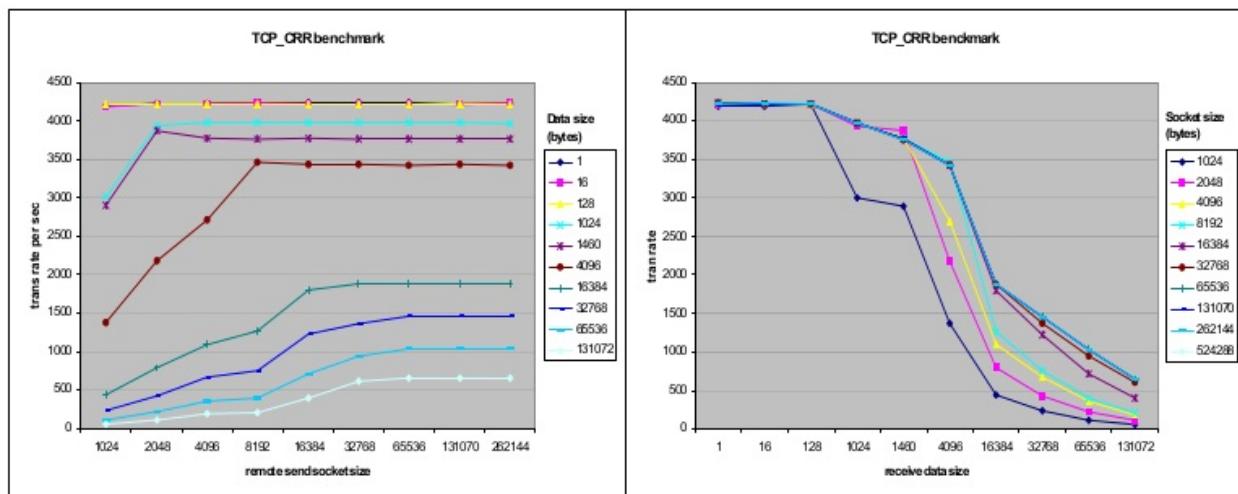


Figure 4-18 No Netfilter rule applied

在开启Netfilter之后，情况发生了明显的变化。

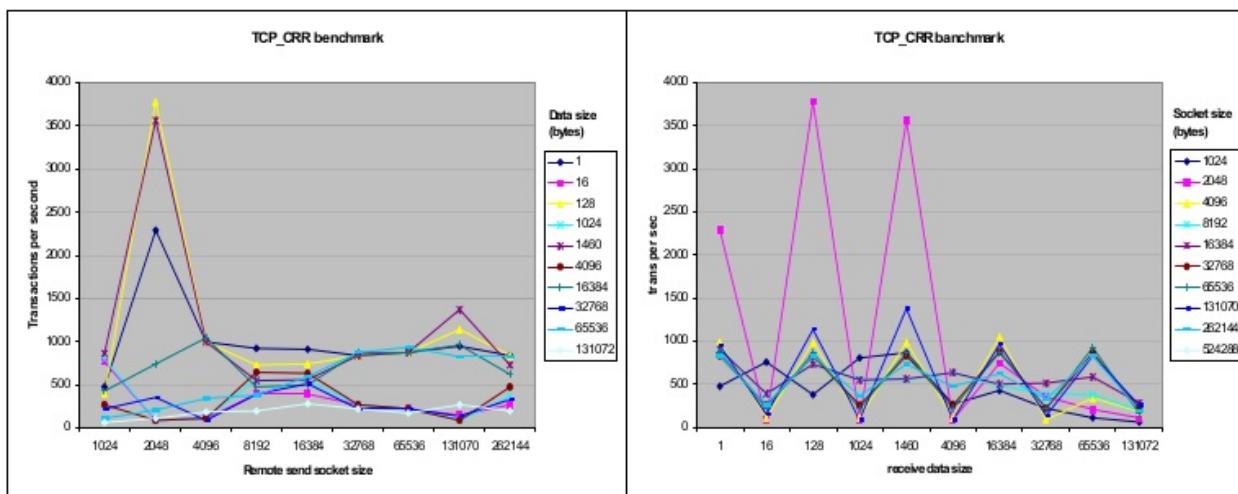


Figure 4-19 Netfilter rules applied

然而，Netfilter提供包过滤功能，增强了网络安全性。这是一个安全和性能之间的权衡考虑。Netfilter对性能的影响取决于如下的因素：

- 防火墙规则数量
- 防火墙规则顺序
- 防火墙规则复杂度
- 连接跟踪级别（取决于协议）
- Netfilter内核参数配置

卸载配置

如果网卡支持的话，有些网络操作可以卸载到网卡上。可以使用ethtool命令确认当前的卸载配置。

Example 4-21 Checking offload configurations

```
[root@lnxsu5 plnxsu4]# ethtool -k eth0
Offload parameters for eth0:
rx-checksumming: off
tx-checksumming: off
scatter-gather: off
tcp segmentation offload: off
udp fragmentation offload: off
generic segmentation offload: off
```

修改配置命令的语法如下：

```
ethtool -K DEVNAME [ rx on|off ] [ tx on|off ] [ sg on|off ] [ tso on|off ] [ ufo on|of
f ] [ gso on|off ]
```

Example 4-22 Example of offload configuration change

```
[root@lnxsu5 plnxsu4]# ethtool -k eth0 sg on tso on gso off
```

对offload能力的支持受因网卡设备、Linux发行版、内核版本已经平台差异而不同。如果你使用一个不支持的offload参数，可能会获得错误信息。

offloading

测试表明，网卡offloading会降低CPU利用率。下图展示了在大块数据传输中CPU利用率的提升。大数据包从校验码offloading中获得优势，因为校验码需要计算整个数据包，所以，数据增加，处理能力就被消耗掉。

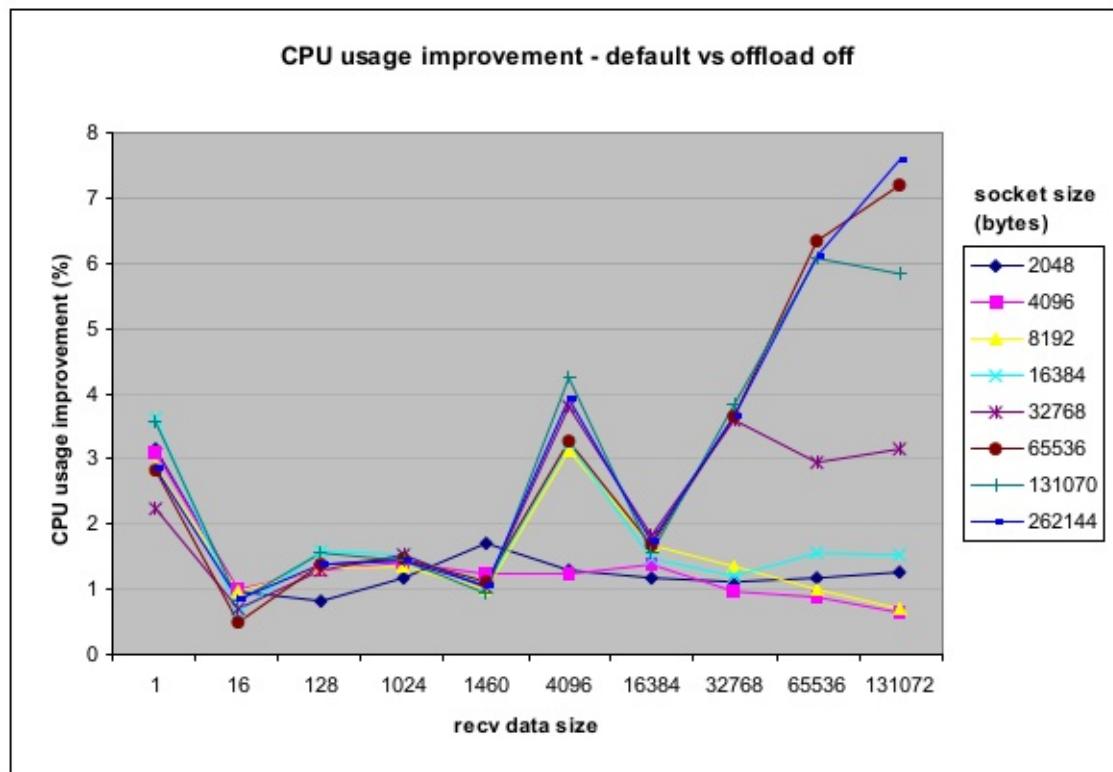


Figure 4-20 CPU usage improvement by offloading

下图，在offloading中也产生了明显的性能下降。在网卡上，对如此大量的包传输率做校验，对局域网卡处理器有很大压力。随着包大小的增加，每秒种处理的包数量降低（因为要花大量时间发送和接收所有数据），所以谨慎使用网卡做校验操作。

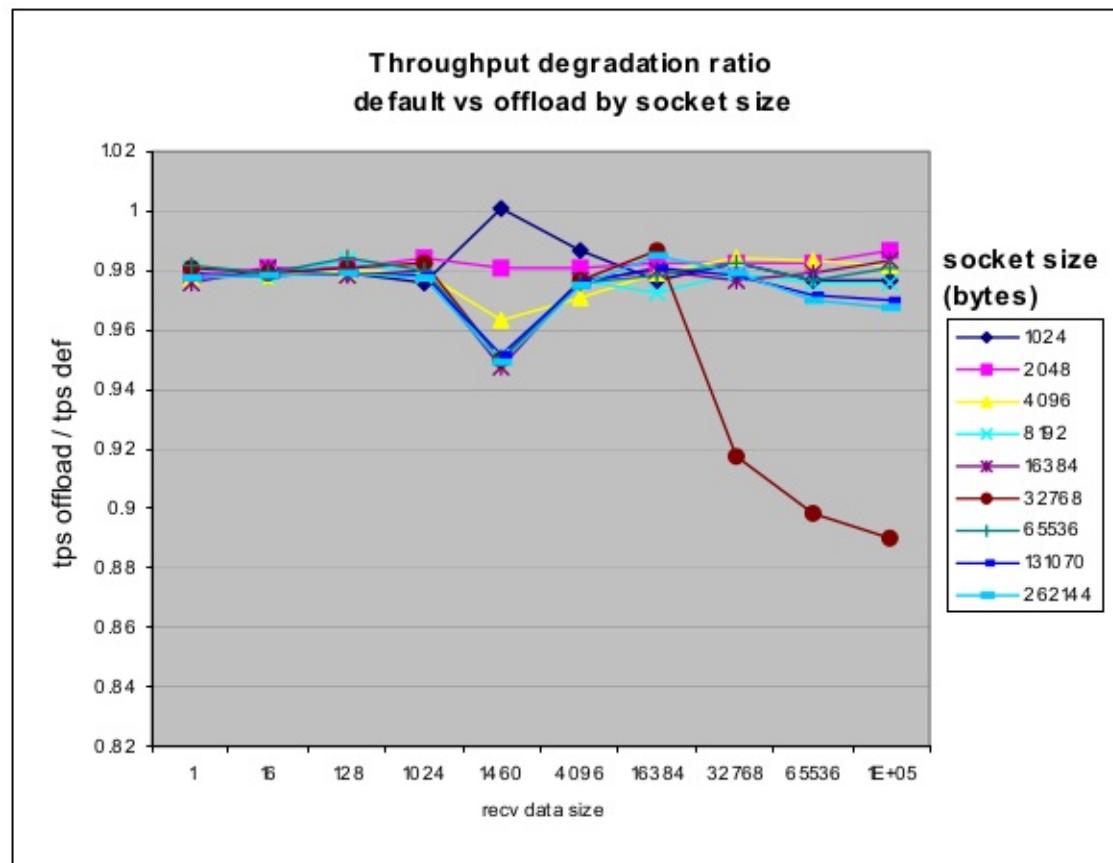


Figure 4-21 Throughput degradation by offloading

在网络应用为大帧请求数据生成请求的时候，**LAN**适配器很有效率。请求小块数据的一个应用需要**LAN**适配器处理器花费大量的时间处理每个数据传输自己的开销。这就是为什么大多数**LAN**适配器不能为所有帧大小维持全速。

增加包队列

在增加各种网络缓冲大小之后，建议增加未处理包的数量，那么，花更长的时间，内核才会丢弃包。可以通过修改/proc/sys/net/core/netdev_max_backlog来实现。

增加发送队列长度

把每个接口的txqueuelength参数值增加到1000至20000。这对数据均匀且量大的高速网络连接十分有用。可以使用ifconfig命令调整传输队列长度。

```
[root@linux ipv4]# ifconfig eth1 txqueuelen 2000
```

减少中断

除非使用**NAPI**，处理网络包的时候需要Linux内核处理大量的中断和上下文切换。对Intel e1000-based网卡来说，要确保网卡驱动编译了CFLAGS_EXTRA -DCONFIG_E1000_NAPI标志。Broadcom tg3模块的最新版本内建了**NAPI**支持。

如果你需要重新编译e1000驱动支持**NAPI**，你可以通过在你的系统做如下操作；

```
make CFLAGS_EXTRA -DCONFIG_E1000_NAPI
```

此外，在多处理器系统中，把网卡中断绑定到物理CPU可能带来额外的性能提升。为实现这个目标，首先要识别特定网卡的IRQ。可以通过ifconfig命令获得中断号。

Example 4-24 Identifying the interrupt

```
[root@linux ~]# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:11:25:3F:19:B3
          inet  addr:10.1.1.11  Bcast:10.255.255.255  Mask:255.255.0.0
          inet6 addr: fe80::211:25ff:fe3f:19b3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51704214 errors:0 dropped:0 overruns:0 frame:0
          TX packets:108485306 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4260691222 (3.9 GiB)  TX bytes:157220928436 (146.4 GiB)
          Interrupt:169
```

获取中断号之后，你可以使用在`/proc/irq/%{irq number}`中`mp_affinity`参数把中断和CPU绑定在一起。如下图示范了如何把之前获取的eth1网卡的169号中断，绑定到系统的第二个处理器。

Example 4-25 Setting the CPU affinity of an interrupt

```
[root@linux ~]# echo 02 > /proc/irq/169/smp_affinity
```
