Table of Contents

前言

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

Linux是一种由全世界人们共同开发的开源操作系统。其源代码可以自由的取得并可以在GNU General Public License下使用。有很多公司提供不同的系统发行版供用户使用，如红帽子和Novell。大部分桌面发行版都可以从网站上免费下载，但服务器版一般是需要购买的。

在过去的几年里，Linux被世界上许多公司的数据中心所使用。Linux系统被科学领域和企业用户所认可。今天，Linux已经是成为一种多种用途的操作系统。你能在多种嵌入式设备中发现它，如：防火墙、手机或电脑主机。所以Linux的性能对于科学领域和企业用户来说已经成为一个热门议题。然而一个操作系统可能被用来计算全球的天气预报或者被用来运行数据库等多种用途，Linux必须能够为各种可能的使用情境提供优良性能。大多数Linux发行版含有常规的调校参数来满足所有用户。

IBM认为Linux作为一种操作系统非常适合在IBM的系统之上运行企业级应用。大多数企业应用现在都可以运行在Linux上，包括文件及打印服务器、数据库服务器、Web服务器、以及沟通和邮件服务器。

当企业级服务器运行Linux时需要对性能实施监控，在必要时对服务器进行调优以解决影响用户的性能瓶颈。此IBM红皮书将介绍一些用来调优Linux的方法、监控分析服务器性能的工具、以及对于特定应用的关键性能参数。本文目的是说明怎样分析和调校Linux操作系统，来为在其系统上运行的各种不同应用提供优良的性能。

在我们测试环境中所使用到的性能调校参数、基准结果以及监控工具，可以在基于IBM Sysem x和System z服务器上拥有2.6内核RedHat、Novell Linxu系统中执行。然而本文的内容对于所有Linux硬件平台都会有很好的帮助。

本书结构

为了帮助Linux新手快速了解性能调优，我们将从以下几个方面进行介绍：

第一章，“了解Linux操作系统”

此章将介绍影响系统性能的因素和Linux操作系统管理系统资源的方法。你可以了解到几种重要的量化性能指标。

第二章，“监控和测量工具”

第二章介绍几种可以用来测量和分析Linux系统性能的工具。

第三章，“分析性能瓶颈”

此章介绍发现和分析系统瓶颈的过程。

第四章，“操作系统调优”

凭着操作系统运行和性能测量工具使用的基本知识，你已经具备了发现Linux性能问题的能力了。

在我们开始前，先来浏览一下Linxu操作系统是怎样控制任务来完成与硬件资源交互的。性能调优是一项富有挑战性的工作，它需要对硬件、操作系统和应用有非常深入的了解。但如果性能调优很简单，那么我们所要探究的配置参数就可以直接硬编码到韧体或操作系统中了，你今天就不会看到本文了。然而正如图1-1所示，服务器性能会受到多种因素的影响。

图1-1：不同元件交互示意图

如果一个有20000名用户的数据库服务器，却只拥有一个单IDE驱动器，这样的I/O子系统你可能花几个星期的时间都无法有效的调优。为了能带来更好的性能，通常作法是更换一个新的驱动器或者将你的应用升级。正如我们之前所讨论的，请在头脑中保持对系统性能的整体印象。了解操作系统管理资源的方法，可以帮助我们在各种各样的应用情境中找出哪个子系统需要调优。

注释：本文主要讨论Linux操作系统的性能。

本章内容包括：

1.1 Linux进程管理

1.2 Linux内存管理

1.3 Linux文件系统

1.4 硬盘I/O子系统

1.5 网络子系统

1.6 了解Linux性能量度

1.1 Linux进程管理(原创翻译)

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

进程管理对于任何一个操作系统来说都是最重要的任务之一。高效的进程管理能保证应用平稳有效的运行。

Linux的进程管理与UNIX十分相似。它包括进程调度、中断处理、信号发送、进程优先级、进程切换、进程状态、进程内存等。

在本章节中，我将讨论Linux进程管理的原理。它能帮助你更好的了解Linux内核是怎样管理进程来影响系统性能的。

1.1.1 进程是什么？

进程就是执行程序运行在处理器上的一个实例。进程可以使用Linux内核所能控制的任何资源来完成它的任务。

所有运行在Linux操作系统上的进程都使用一个名叫task_struct的结构来管理，这个结构亦被称作进程描述符(Process Descriptor)。进程描述符包括进程运行的所有信息如进程ID、进程属性和构建这个进程所需要的资源。如果你清楚进程的结构，就能了解到什么对于进程执行和效能来说是重要的。图1-2展现了进程结构的概要。

图 1-2 task_struct结构

1.1.2 进程生命周期

每个进程都有自己的生命周期如创建、执行、结束和消除。这些阶段在系统启动运行中会被重复无数次。因此从性能角度来看进程生命周期是极其重要的。

图1-3展示进程典型的生命周期

图1-3进程典型的生命周期

当进程创建一个新的进程，创建进程（父进程）发出fork()系统调用。当一个fork()系统调用被发出，它将得到一个关于新进程（子进程）的进程描述符并设置一个新的进程ID。它会将父进程的进程描述符中所有数据复制到子进程。此时父进程的整个地址空间并没有被复制的，所以父子进程会共享相同的地址空间。

exec()系统调用会复制一个新的程序到子进程的地址空间。因为父子进程共享相同的地址空间，所以当新程序写入数据时会导致分页错误(page fault)的例外发生。这时候内核会分配给子进程一个新的物理分页。

这个推迟的操作被叫做写时复制(Copy On Write)。子进程通常是执行自己的程序，与其父进程所执行的有所不同。这样的操

作可以避免没有必要的系统开销，因为复制整个地址空间是一个非常慢而且效率低的操作，它会消耗很多处理器时间和资源。

当程序执行完成时，子进程调用exit()系统调用结束。系统调用exit()会释放进程的大部分数据结构并发送信号通知父进程。此时子进程被称作僵尸进程(zombie process)。

在子进程使用wait()系统调用让父进程知道其已经结束之前，子进程是不会被清除的。当父进程得到子进程结束的通知后，会立即清除子进程的所有数据结构并释放进程描述符。

1.1.3 线程

线程是由一个单独进程产生的执行单元。它与同一进程中的其他线程并行运行。它们能共享同一资源如内存、地址空间、打开的文件等。它们能访问同样一组应用数据。线程也被称作轻量级进程（Light Weight Process LWP）。因为它们共享资源，线程不可以在同一时间修改它们共享的资源。互斥的实现、锁、序列化等是用户应用的职责。

从性能角度来说，创建线程要比创建进程的开销小，因为线程在创建时不需要复制资源。另一方面，进程和线程在调度算法方面上有很多相似的特性。内核在处理它们时都使用类似的方法。

图1-4 进程和线程

在目前Linux的实现中，线程支持可移植操作系统接口（POSIX）。在Linux操作系统中有几种线程的实现。下面列举几种最常使用的线程实现。

- LinuxThreads

LinuxThreads自从Linux内核2.0就被作为默认的线程实现。但LinuxThread有许多实现与POSIX标准不兼容。Native POSIX Thread Library（NPTL）正在取代LinuxThreads。在未来的Linux企业发行版中将不在支持LinuxThreads。

- Native POSIX Thread Libray（NPTL）本地POSIX线程库

NPTL最初是由Red Hat开发。NPTL与POSIX标准更加兼容。利用2.6内核增强特性如新的系统调用clone()、信号处理实现等，它可以提供较LinuxThreads更好的性能和伸缩性。

NPTL与LinuxThreads有很多的不兼容之处。一个应用如果依赖于LinuxThreads，可能在NPTL实现中无法工作。

- Next Generation POSIX Thread（NGPT）下一代POSIX线程

NGPT是IBM开发的POSIX线程库的版本。目前处于维护阶段，未来也没有开发计划。

使用LD_ASSUME_KERNEL环境变量，你可以设定应用使用哪个线程库。

1.1.4 进程优先级和Nice值

进程优先级(Process priority) 是一个数值，用来让CPU根据动态优先级和静态优先级来决定进程执行的顺序。一个高优先级的进程可以获得更多在处理器上运行的机会。

内核会根据进程的行为和特性使用试探算法(Heuristic Algorithm)来动态调高和调低动态优先级。用户进程可以通过进程Nice的值间接改变静态优先级。静态优先级高的进程可以获得较长的时间片【Time Slice】（进程能运行在处理器有多长时间）。

Linux中Nice值范围为19（最低优先级）到-20（最高优先级），默认值为0。要将Nice值更改为负数，必须通过登录或使用su命令由root执行。

1.1.5 上下文交换(Context switching)

在进程执行过程中，进程信息存储在处理器的寄存器和缓存中。这组为执行中进程而载入寄存器的数据被称作上下文

(Context)。为切换进程，当前执行中进程的上下文会被暂存，下一个执行进程的上下文会被还原到寄存器，进程描述符和内核模式堆栈【Kernel mode stack】的区块会被用来存储上下文，这个交换过程被叫做上下文交换【Context Switching】。发生过多的上下文交换是不好的，因为处理器每次都要刷新寄存器和缓存为新进程让出资源，这会导致性能上的问题。

图1-5 说明了上下文交换是怎样工作的。

图1-5 上下文交换

1.1.6 中断处理

中断处理是优先级最高的任务之一。中断通常是由I/O设备产生的如网卡、键盘、硬盘控制器、串行适配器等。中断控制会向内核发送一个事件通知（如键盘输入、以太帧到达等），它指示内核中断执行中的进程并尽快处理中断，因为大多数设备需要快速的回应，这对系统性能是很关键的。当一个中断信号到达内核时，内核必须切换当前执行的进程到处理中断的新进程，这意味着中断会触发上下文交换，因此大量的中断可以导致系统性能的下降。

在Linux中，有两种类型的中断。一种为硬中断(Hard Interrupt)，是由需要回应的设备产生的（硬盘I/O中断、网络适配器中断，键盘中断，鼠标中断）。另一种为软中断【Soft Interrupt】，用于可以延后执行的任务（TCP/IP操作，SCSI协议操作等）。你可以在/proc/interrupts中查看到有关硬中断的信息。

在多处理器环境下，每个处理器都可以用来处理中断。将中断绑定到某一个物理处理可以提升系统的性能。要了解更详细的内容，请参考4.4.2“关于中断处理的CPU亲和力(CPU Affinity)”。

1.1.7 进程状态

每个进程都有它自己的状态，来显示进程当前的情况。在进程执行过程中其状态会发生变化。可能状态有如下几种：

- TASK_RUNNING(运行中) : 此状态表示进程正运行在CPU上或在队列中等待运行（运行队列(Run Queue)）。
- TASK_STOPPED(停止) : 当进程接收到某些信号（例如SIGINT、SIGSTOP）后被暂停就处于此种状态，该等待的进程在收到恢复信号如SIGCONT后会重新投入运行。
- TASK_INTERRUPTIBLE(可中断) : 在这种状态下，进程被暂停运行，等待某些状态的达成。如果一个处于可中断状态的进程收到停止的信号，将变更进程的状态并中断操作。可中断状态进程的一个典型例子就是等待键盘的输入。
- TASK_UNINTERRUPTIBLE(不可中断) : 此状态基本上与可中断状态十分相似。但可中断状态进程可以被中断，而向一个不可中断进程发送信号却不会有任何反应。不可中断状态进程的一个典型例子就是等待硬盘I/O操作。
- TASK_ZOMBIE(僵尸) : 在进程使用系统调用exit()退出后，其父进程就会知道。僵尸状态的进程会等待父进程通知其释放所有的数据结构。

图1-6 进程状态

僵尸进程

当一个进程收到信号并已经终止，它通常需要一些时间来完成结束前的所有任务（如关闭打开的文件）。在这个通常很短的时间里，该进程就是一个僵尸（Zombie）。

当进程完成所有的关闭任务后，它会向父进程报告其即将终止。但有时僵尸进程并不能将自己终止，在这种情况状态会显示为Z（Zombie）。

使用kill命令是不可能结束这样一个进程的，因为这个进程被认为已经死掉了。如果你不能清除僵尸，你可以结束其父进程，这样僵尸也会随之消失。然而如果父进程是init进程，你就不可以结束它，因为init进程是一个极为重要的进程。因此你可能需要重新启动系统来清除这个僵尸进程。

1.1.8 进程内存段

一个进程需要使用自己的内存区域来执行工作。工作的变化随情况和进程用法而定。一个进程可以有不同的工作负载特性和不同的数据大小的需求，进程需要处理数据的大小多种多样。为了满足这样的需求，Linux内核使用动态内存分配机制。进程内存分配结构如图1-7。

图1-7 进程地址空间

进程的内存区有几部分组成

- 文字段 : 用来存储执行代码。
- 数据段 数据段由三块区域组成。
  - 数据(Data)：存储已初始化数据如静态变量。
  - BSS ：存储零初始化的数据，数据被初始化为零。
  - 堆(Heap)：这块区域由malloc()用来按需要分配动态内存。堆向高地址扩张。
- 堆栈段 : 用来存储本地变量、函数参数、函数返回地址。堆栈段向低地址扩张。

使用pmap命令可以显示用户进程地址空间的内存分配。你可以使用ps命令显示此内存段的大小。参考2.3.10“pmap”和2.3.4“ps和pstree”。

1.1.9 Linux CPU调度器

计算机的基本功能非常简单就是计算。为了计算，这就意味要管理计算资源或处理器和计算任务（被称作线程或进程）。Linux内核使用与过去CPU调度器使用的算法O(n)截然不同的O(1)算法,这要感谢Lngo Molnar的巨大贡献。O(1)指的是一种静态算法，意思就是不管进程的数量有多少，进程的执行时间都是不变的。

这种新的调度器的扩展性非常好，不管进程的数量或处理器的数量有多少，系统的开销都是非常小的。此算法中使用到两个进程优先级数组：

- 活动的(Active)
- 过期的(Expired)

调度器根据进程的优先级和优先拦截率(Prior Blocking Rate)分配时间片，然后它们被以优先级顺序置于活动数组【Active Array】中。当时间片耗尽，它们会被分配一个新的时间片并置于过期数组中。当活动数组中所有进程的时间片都全部耗尽，两个数组会被互换并重新执行。对于交互进程（相对于实时进程），拥有长时间片的高优先级进程可以得到比低优先级进程更多的时间，但这并不意味着低优先级的进程会被置之不理。在企业环境中，拥有很多的处理器并经常出现大量的线程和进程，这样做可以大大提升Linux内核的伸缩性。新的O(1)CPU调度器被设计用于2.6内核，但已被移植到2.4内核家族。图1-8说明了Linux CPU调度器是怎样工作的。

图1-8 Linux 2.6内核 O(1)调度器

新调度器另一个大的改进就是支持非一致性内存架构（NUMA）和对称多线程处理器，如Intel超线程技术。

改良后的NUMA支持确保只有当某个节点过载时，负载平衡才会跨越NUMA节点。This mechanism ensures that traffic over the comparatively slow scalability links in a NUMA system are minimized。尽管在每个调度节拍(tick)时负载平衡会遍历调度域群组【Scheduler Domain Group】中的处理器，但只有在节点过载并请求负载平衡时，负载才会跨越调度域【Scheduler Domain】转移。

图1-9 O(1) CPU调度器结构

译者注：在翻译过程中，深深的感觉到进程调度是一项非常复杂的工作。如果想深入了解2.6内核进程调度还需要多多查看相关资料，下面是一些关于2.6内核进程调度的文章，有兴趣的朋友不妨看看。

- Linux 2.6 调度系统分析：http://www.ibm.com/developerworks/cn/linux/kernel/l-kn26sch/index.html
- Linux 的 NUMA技术：http://www.ibm.com/developerworks/cn/linux/l-numa/index.html
- Linux Scheduling Domains：http://www.ibm.com/developerworks/cn/linux/l-cn-schldom/index.html
- Inside the Linux scheduler：http://www.ibm.com/developerworks/linux/library/l-scheduler/

1.2 Linux内存结构

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

1.2.1 物理内存与虚拟内存 1.2.2 虚拟内存管理

执行一个进程时，需要Linux 内核分配一段内存区域，用来作为工作区来执行它的工作。这就像你的办工桌，你可以用它来摆放工作需要的纸张、文件和备忘录。不同之处是Linux内核要使用动态的方法来分配内存空间。在内存大小通常是有限的情况下有时进程的数量会达到数万个，Linux内核必须有效地来使用内存。在本章节中，我们会了解到内存的结构、地址分布，以及Linux是怎样有效地管理内存空间的。

1.2.1 物理内存与虚拟内存

今天我要面对选择32位系统还是选择64位系统的这样问题。其中对于企业级用户最重要的区别就是虚拟内存地址是否可以超过4GB。从性能角度来看，弄明白32位和64位Linux内核是怎么样将物理内存映射到虚拟内存是非常有益的。

在图1-10中，你可以很明显的看出32位系统和64位系统在内存地址分配上的不同之处。关于物理内存映射到虚拟内存的详细内容已超出本文的范畴，本文将只着重介绍Linux内存结构的部分细节。

在32位架构如IA-32中，Linux内核只能直接访问物理内存的前1GB（当考虑保留部分时只有896MB）。在ZONE_NORMAL之上的内存需要映射到1GB一下，这样的映射对于应用来说是完全透明的，但分配内存页到ZONE_HIGHMEM会导致性能轻微的下降。

在64位架构如x86-64（也称作x64），ZONE_NORMAL可一直延伸到64GB，在IA-64中甚至达到128GB。正如你所见到的，在64位架构中，可以被省去将内存页从ZONE_HIGHMEM映射到ZONE_NORMAL的开销。

图1-10 32位和64位Linux内核内存布局

虚拟内存寻址布局

图1-11展示了32位和64位Linux虚拟寻址布局。

在32位架构中，单个进程能访问最大地址空间位4GB，这就限制了32位虚拟寻址能力。在标准实现中，虚拟地址空间被分为3GB的用户空间和1GB的内核空间，这有些像4 G/4 G寻址布局实现的变种。

在64位架构如X86_64和IA64中，就没有这样的限制。每个单独的进程都能获得巨大的地址空间。

图1-11 32位和64位Linux虚拟寻址布局

1.2.2 虚拟内存管理

操作系统的物理内存架构对于应用和用户来说通常是隐藏的，因为操作系统会将所有内存映射到虚拟内存。如果我们想要知道Linux操作系统调优的可能性，我就必须要明白Linux是怎样管理虚拟内存的。正如1.2.1“物理内存和虚拟内存”中讲到的，应用是不能分配到物理内存的，当向Linux内核请求一定大小的内存映射时，得到是一段虚拟内存的映射。如图1-12所示，虚拟内存不一定要映射到物理内存，如果你的应用需要很多内存，其中一部分很有可能是被映射到了硬盘上的交换文件。

如图1-12所示应用通常直接写进缓存区(Cache)或缓冲区【Buffer】而不是硬盘。当内核线程pdflush空闲或文件大小超出缓冲区，pdflush会将缓存区和缓冲区数据清空并写入硬盘。参考“清空脏缓冲”。

图1-12 Linux虚拟内存管理

Closely connected to the way the Linux kernel handles writes to the physical disk subsystem is the way the Linux kernel manages disk cache.其它操作系统只分配部分内存作为硬盘缓存，而Linux在内存资源管理上则更加有效。Linux虚拟内存管理默认将所有空闲内存空间都作为硬盘缓存。因此在拥有数GB内存的生产性Linux系统中，经常可以看到可用的内存只有20MB。

同样，Linux在管理交换空间方面也十分有效。当交换空间被使用是并不表示内存出现瓶颈，这恰恰证明Linux管理系统资源是多么有效。详见“页帧回收”

页帧的分配

页(Page)就是指在物理内存（page frame）或虚拟内存中一组连续的地址。Linux内核就是以页为单位来管理内存的，页的大小通常为4KB。当有进程请求一定数量的页时，如果存在可有的页Linux内核会立即将其分配给进程，否则要从其它的进程或页缓存中取得页。内核清楚知道有多少可以使用的内存页并知道它们都在哪里？

伙伴系统(Buddy System)

Linux内核使用一种名叫伙伴系统的机制来管理自由页(Free Pages)。伙伴系统维护自由页并尝试为有需要的进程分配页，尽可能保持内存区是连续。如果忽视分散的小页，可能会造成内存碎片。这让在连续的区块中分配一大段页变得困难。从而导致内存使用效率降低性能下降。图1-13说明了伙伴系统是如何分配页的。

图1-13 伙伴系统

当尝试分配页失败，页回收被激活。参见“页帧回收”。

你可以在/proc/buddyinfo中获得关于伙伴系统的信息。详细内容参见“Memory used in a zone”。

页帧回收

当进程请求一定数量的页但没有可用的页时，Linux内核会尝试释放某些页（之前被使用过但现在没有被使用，但基于某种原则其仍被标示为活动页）来满足请求并分配内存给新的进程。这个进程被称作页回收。kswapd内核线程和try_to_free_page()内核函数被用来负责页的回收。

kswapd通常在可中断状态中睡眠，当某一区域(Zone)的自由页低于临界值时伙伴系统将呼叫kswapd。它将尝试使用最近最少使用（LRU）原则从活动页中找出候选页。最近最少使用的页将被首先释放。活动列表和非活动列表被用来维护候选页。kswapd扫描部分活动列表，检查页最近的使用情况，将最近没有使用的页放入非活动页。你可以使用vmstat -a查看到哪些内存是活动的哪些内存是非活动的。详细信息参见2.3.2"vmstat"。

kswapd也要遵循其它原则。页的使用主要两个用途是：页缓存(Page Cache)和进程地址空间【Process Address Space】。页缓存是指将页映射到硬盘上的一个文件。一个进程地址空间的页（被称作匿名内存，因为它没有映射到任何文件也没有名字）被用作堆和堆栈。参见1.1.8“进程内存段“。当kswapd回收页时，它会尽量压缩页缓存而不是将进程的页page out（或swap out）。

Page out and swap out：page out和swap out很容易被混淆。”page out“是将某些页（整个地址空间的一部分）置入交换空间，而”swap out“是将整个地址空间置入交换空间。但有时它们可以互换使用。

大部分被回收的页缓存和进程地址空间取决于使用情境从而影响性能。你可以通过/porc/sys/vm/swappiness来控制这样的行为。调优的详细内容参见4.5.1“设置内核交换和pdflush行为”。

swap

正如之前所述，当页回收发生时，在活动列表中属于进程地址空间的候选页会被page out。交换的发生本身并不意味着出了状况。虽然在其它操作系统中交换只不过是当内存过度分配时的一种保障，但Linux在使用交换空间时则更为有效。正如你在图1-12所见，虚拟内存是由物理内存和硬盘子系统或交换空间组成。在Linux中如虚拟内存管理者发现被分配的内存页在一段时间内没有被使用，它将被移至交换空间。

你经常可以看到守护进程如getty自从系统启动后就很少被使用。显然将内存页移至交换空间释放所占用的主内存会更加有效。如果你发现交换分区已经使用了50%。并不需要惊慌，这正体现Linux是怎样管理交换的。事实上交换空间被使用并不意

味着内存出现瓶颈，相反验证了Linux使用系统资源的有效。

1.3 Linux文件系统

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

1.3.1 虚拟文件系统 1.3.2 日志记录 1.3.3 Ext2 1.3.4 Ext3 1.3.5 ReiserFS 1.3.6 日志文件系统 1.3.7 XFS

Linux作为开源操作系统其中最大的优势就是支援多种文件系统。现代Linux内核几乎支援计算机系统中所有文件系统，从基本的FAT到高性能文件系统如日志文件系统(Journaling File System)(JFS)。由于Ext2、Ext3和ReiserFS为大多数Linux发行版的原生文件系统（ReiserFS只有使用在Novell SUSE Linux上才能获得商业支持），所以我们只集中讲解这些文件系统的相关特性，至于其它文件系统只会略作说明。

关于文件系统和硬盘子系统，参见4.6 “调优硬盘子系统”

1.3.1 虚拟文件系统

虚拟文件系统（VFS）是一个位于用户进程与各种Linux文件系统之间的抽象接口层，它提供用于访问文件系统对象的通用对象模型（如i-node、文件对象、页缓存、目录条目等等）和方法，它对用户进程隐藏了各种文件系统间的差异。用户进程不需要知道使用的是哪个文件系统或不同的文件系统需要呼叫哪个系统调用。图1-14阐明了VFS的概念。

图1-14 VFS概念

1.3.2 日志记录

在一个没有日志记录的文件系统中，当文件系统执行一个写操作时，首先内核会更改文件系统的元数据，然后才写入实际的用户数据。这样的操作对于保持数据完整性会产生很高的风险。当写操作正在更改文件系统元数据时系统突然崩溃，文件系统的一致性很可能遭到破坏。在下次重启时，fsck需要通过检查所有的元数据来修复一致性。但当系统的容量很大时，完成检查需要很长的时间才能完成。在此过程中系统是不可以使用的。

一个有日志记录的文件系统是通过在数据写入实际文件系统前先将变更的数据写入日志区(Journal Area)来解决这个问题。日志区可以被放置于文件系统中也可被放置于文件系统之外。被写入日志区的数据被称作日志记录【Journal Log】。它包含文件系统元数据的变化和实际的文件数据（如果支持）。

因为日志记录在写入实际用户数据到文件系统前需要写入journal log，相对于没有日志记录的文件系统这会导致性能上的开销。维护数据的高一致性会牺牲多少的性能开销，取决写入用户数据前有多少信息需要写入硬盘。我们将在1.3.4“Ext3”中讨论此课题。

图1-15日志记录概念

1.3.3 Ext2

Ext2是Ext3文件系统的前身，一个快速、简单的文件系统。不像当今其它大多数文件系统一样，它并没有日志功能。

图1-16向我们展示了Ext2文件系统的数据结构。文件系统以引导扇区开始，紧接着是块组(Block Group)。因i-node表和存储用户数据的数据块【Data Block】可以位于磁碟片相邻位置，这样就可以减少寻道的时间，为了性能的提升整个文件系统被分为多个小块组。一个块组是由下面几项组成：

Super block ：存储文件系统信息。Super block必须位于每个块组的顶部。 Block group descriptor：存储块组的相关信息。

Data block bitmaps：用于管理未使用的数据块。 i-node bitmaps：用户管理未使用的i-node。 i-node tables：存储i-node表。每个文件都有一个相应的i-node表，用来保存文件的元数据如：文件模式、uid、gid、atime、ctime、mtime、dtime和数据块的指针。 Data blocks：存储实际的用户数据。

图1-16 Ext2文件系统数据结构

为了要找到组成文件的数据块，内核首先查找文件的i-node。当一个进程请求打开/var/log/messages时，内核解析文件路径并查找/（根目录）的目录项，获得其下的文件及目录信息。下一步内核继续查找/var的i-node并查看/var的目录项，它也包含其下的文件及目录信息。内核继续使用同样的方法直到找到所要文件的i-node。Linux内核使用文件对象缓存如目录项缓存或i-node缓存，来加快查找相应i-node的速度。

当Linux内核找到文件的i-node后，它将尝试访问实际的用户数据块。正像我们所讲述的，i-node保存有数据块的指针。通过指针内核可以获得数据块。对于大文件，Ext2提供了直接/间接的数据块参照。图1-17描绘了其是怎样运作的。

图1-17 Ext2文件系统直接/间接的数据块参照

不同的文件系统在文件系统结构和文件访问操作上存在着不同的差异，这让每个文件系统都各具特色。

1.3.4 Ext3

当前Linux企业发行版中都支持第三扩展文件系统。它作为第二扩展文件系统的升级版本被广泛使用。经管它有着与Ext2文件系统相似的基本结构，但与Ext2最大的不同之处就是其拥有日志记录功能。此文件系统亮点包括：

- 可用性：Ext3始终保持写入硬盘数据的一致性。当发生非正常关机（以外的电源故障或系统崩溃）时，服务器不需要花费时间去检查数据的一致性，从而将系统恢复时间从几小时缩短至几秒钟。
- 数据完整性：在mount命令时通过指定日志记录模式data=journal，所有数据包括文件数据及元数据都会被记录日志。
- 速度：通过指定日志记录模式data=writeback，你可以根据业务需要来决定选择速度还是完整性。在一个同时发生写操作非常频繁的环境中这是非常重要的。
- 复杂度：将一个已存在的Ext2文件系统升级到Ext3文件系统是很简单的，不需要重新格式化。通过执行tune2fs命令并修改/etc/fstab文件，你就可以很容易将Ext2升级到Ext3文件系统。此外Ext3文件系统可以被挂载为Ext2，但日志记录功能此时是不可用的。有许多第三方的产品提供调整Ext3文件系统的功能，例如PartitionMagic可以修改Ext3的分区。

日志模式

Ext3支持3种日志模式

- 日志 此种模式提供了最高等级的数据一致性，文件数据和元数据都将被记录日志。它也会造成过多的性能开销。
- 顺序 在这种模式下只有元数据会被写入，然而在此之前首先保证文件数据已被写入。此模式为默认设定。
- 回写 此种模式是在牺牲数据一致性的情况下提供访问数据最快的速度。为保证数据的一致性，元数据任会被记录。然而实际的文件数据并不会被特殊处理，这就导致当系统崩溃后旧数据的出现。

1.3.5 ReiserFS

ReiserFS是一种快速的日志文件系统，它可以优化硬盘空间的使用和加快故障恢复的速度。ReiserFS的开发很大程度上得益于Novell的支助。只有在Novell SUSE Linux上使用ReiserFS才可以得到商业支持。

1.3.6 Journal File System

JFS是一个全64为文件系统，能支持大文件和分区。JFS起初是IBM为AIX所开发的，但现在已近可以在GPL下使用。JFS对于高性能计算（HPC）和数据库环境中的大分区和文件是非常理想的文件系统。如果想了解更多，参见 http://jfs.sourceforge.net 注释：在Novel SUSE Linux Enterprise Server 10中，JFS不在被支持。

1.3.7 XFS

eXtended File System（XFS）起初是Silicon Graphics公司为其IRIX系列系统开发的高性能日志文件系统。其功能特性与来自IBM的JFS非常相似，也支持大文件和分区，因此使用情境与JFS也十分相似。

1.4 硬盘I/O子系统

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

1.4.1 I/O子系统架构 1.4.2 缓存 1.4.3 块层 1.4.4 I/O设备驱动 1.4.5 RAID及存储系统

在处理器解码和执行指令前，先从扇区中读取数据并将数据置于处理器的缓存和寄存器中。执行结果会被写回硬盘。

下面我们将简单地介绍一下Linux硬盘I/O子系统，来更好的认识这个可以对系统性能产生很大影响的组件。

1.4.1 I/O子系统架构

图1-18展示了I/O子系统架构的基本概念。

图1-18 I/O子系统架构

为了能快速的对I/O子系统运作有一个整体的认识，我们将举一个写数据到硬盘的例子。下面的步骤概述了当一个硬盘写操作执行时的基本操作。假设存储于硬盘片扇区上的数据已经被读取到分页缓存中。

1.进程使用write()系统调用发出写文件的请求。

2.内核更新映射此文件的分页缓存。

3.pdflush内核线程将分页缓存清空至硬盘。

4.文件系统层将多个块缓冲(block buffer)置于bio结构中（参见1.4.3，“块层”）并提交一个写请求给块设备层。

5.块设备层从上一层收到请求，执行一个I/O elevator操作并发出进入I/O请求队列的请求。

6.设备驱动如SCSI或其它设备的特定驱动来负责完成写操作。

7.硬盘设备韧体负责执行硬件操作像确定磁头、旋转和传输数据至磁盘片上的扇区。

1.4.2 缓存

在过去的20年里，处理器性能的提升已超过计算机中其它元件如处理器缓存、总线、RAM、硬盘等。内存和硬盘较慢的访问速度限制了系统的整体性能，所以单单提高处理器速度并不能让系统性能提升。缓存机制通过将常用数据缓存至快速内存中来解决此问题。它减少了访问较慢内存的机会。目前计算机系统中几乎所有的I/O元件都是用了这项技术如硬盘驱动器缓存、硬盘控制器缓存、文件系统缓存、各种应用程序的缓存等等。

内存层级(Memory Hierarchy)

图1-19展示了内存层级的概念。CPU寄存器和硬盘在访问速度上有着很大的差距，CPU需要花费更多的时间等待从缓慢硬盘上获取数据，这显然降低了一个快速CPU所带了的好处。内存层级结构通过在CPU和硬盘间部署一级缓存、二级缓存、RAM和其他一些缓存可以减少这种速度上的失衡。它能减少处理器访问较慢内存和硬盘的机会。靠近处理器的内存拥有更快的访问速度但容量却较小。

这种技术也可以利用局部性引用原则。快速内存中更高的缓存命中率就意味着更高的数据访问速度。

图1-19 内存层级

局部性引用(Locality of Reference)

正如我们前面在“内存层级”中所述，取得更高的缓存命中率是性能提升的关键。为取得更高的缓存命中率，“局部性引用”技术被使用。这项技术是基于下面的原则：

- 最近使用过的数据非常有可能在近期被再次使用（时间局部性）
- 位于使用过数据相邻的数据非常有可能被使用（空间局部性）

图1-20说明了这个原则。

图1-20局部性引用

这个原则被应用于Linux的许多元件中如分页缓存、文件对象缓存（i-node缓存、目录项缓存等）、预读缓冲以及更多。

清空脏缓冲(Flushing a dirty buffer)

当进程从硬盘中读取数据时，数据被复制到内存。此进程和其他进程都可以从缓存在内存中的副本取得相同数据。当有进程尝试变更数据，会先更改内存中数据。此时，硬盘中数据和内存中数据发生不一致，内存中数据被称为脏缓冲。赃缓冲会被尽快同步至硬盘，但如果系统突然崩溃内存中数据就会丢失。

同步脏数据的进程叫做flush。在Linux2.6内核中，pdflush内核线程负责清空数据到硬盘。此操作会定期（kupdate）执行或者当内存中脏缓冲比例超过临界值（bdflush）时执行。此临界值配置在/proc/sys/vm/dirty_background_ratio文件中。更多信息参见4.5.1“设置内核交换和pdflush行为”。

图1-21清空脏数据

1.4.3 块层

块层负责管理所有关于块设备操作的相关活动（参见图1-18）。块层中的关键数据结构就是bio，bio结构是位于文件系统层和块层之间的一个接口。

当执行写操作时，文件系统层尝试向由块缓冲组成的分页缓存中写入。将连续的块组成一个bio结构，然后将bio传送至块层。（参见图1-18）

块层获得bio请求并将其链接至I/O请求队列（I/O Request Queue）中。这个链接操作叫做I/O调度器(原文中使用的单词为elevator)。在Linux 2.6内核中，共有四种I/O调度器算法可以选择，他们是：

块的大小(Block Sizes)

块大小（读写硬盘数据的最小数量）能直接影响服务器的性能。最为一个指引，如果你的服务器处理非常多的小文件，较小的块大小会更有效率。如果你的服务器用于处理大文件，较大的块大小可以提升性能。已有的文件系统是不能更改块大小的，只有在重新格式化时才能修改当前块大小。

I/O调度器

Linux 2.6内核的I/O调度器使用了新模式，而Linux 2.4内核中采用的是一种简单、通用的I/O调度器，而在2.6内核中有四种调度器可供选择。因为Linux被用来执行各种各样的任务，对I/O设备和负载要求有显著的不同，一台笔记本电脑和一台有10000个用户的数据库对I/O的需求就有很大差别。为满足这样的需求，Linux提供了四种I/O调度器。

- Anticipatory

Anticipatory I/O 调度器是基于假设块设备只有一个物理寻址磁头（例如一个单SATA驱动）。Anticipatory 调度器使用了增加预测启发能力的Deadline机制（下面会详细介绍）。正像名字所暗示的，Anticipatory I/O Anticipatory会“猜测”I/O并尝试使用单

一较大的数据流写入硬盘代替多个小的随机硬盘访问。这种预测启发能力会造成写入的延迟，但它可以提高一般用途系统如个人电脑的写入吞吐量。在2.6.18内核中Anticipatory调度器被作为标准的I/O调度器，然而大多数企业发行版默认使用CFQ调度器。

- 完全公平队列（CFQ）

CFQ elevator通过为每个进程都维护单独的I/O队列来实现QoS（服务质量）策略。CFQ调度器也适用于拥有许多互相竞争进程的多用户系统。它可以避免进程饿死并降低延迟。自从2.6.18内核，CFQ调度器已被作为默认的I/O调度器。

依据系统设置和负载情况，CFQ调度器可以降低单一主应用程序的速度，例如使用其公平向导算法的大型数据库 。默认配置进程组确保公平，进程组与其它进程竞争。例如数据库所有写分页缓存（所有pdflush实例为一个群组）的操作被认为是一个应用程序与其它后台进程竞争。在这样的情况下它对于测试I/O调度器配置或Deadline调度器也是非常有用的。

- Deadline

Deadline是个使用最后期限算法的轮询调度器（Round Robin），它提供I/O子系统接近实时的操作。Deadline在维持满意的硬盘吞吐量的同时可以提供优秀的请求延迟。Deadline的算法能确保不会有进程被饿死。

- NOOP

NOOP代表不操作(No Operation)，名字就说明了其主要功能。NOOP简单且直接，它只实现一个简单的FIFO队列并不对任何数据排序。NOOP只是简单地合并数据请求，因此它增加非常低的处理器负载。NOOP假设块设备拥有自己的调度器算法如SCSI的TCQ或没有寻道的延迟的块设备如闪存卡。

注释：在2.6.18内核中可以为每个硬盘子系统设定特定的I/O调度器，不需要在系统级上设定。

1.4.4 I/O设备驱动

Linux内核使用设备的驱动程序来控制设备。设备的驱动程序通常为独立的内核模组，它让操作系统中的每个设备都可以使用。当设备驱动被加载后，它就作为内核的一部分运行并完全控制设备。这里我们将介绍一下SCSI设备驱动。

SCSI

小型计算机系统接口（SCSI）是一种最常用的I/O设备技术，特别是在企业服务器环境中。在Linux内核的实现中，SCSI设备由设备驱动模组控制。它们由下面几种模组构成。

- 上层驱动：sd_mod,sr_mod(SCSI-CDROM),st(SCSI Tape),sq(SCSI generic device)等。 提供支持几种SCSI设备的功能如SCSI-CDROM，SCSI磁带等。
- 中间层驱动：scsi_mod 实现SCSI协议和SCSI共有的功能。
- 底层驱动 提供设备的底层访问。底层驱动为每个设备提供特定的硬件驱动。例如用于IBM ServeRAID控制器的ips、用于Qlogic HBA的qla2300、用于LSI Logic SCSI控制器的mptscsih等。
- 伪驱动：ide-scsi 用于模拟IDE-SCSI。

图1-22 SCSI驱动结构

如果要设备实现特殊的功能，它需要在设备韧体和底层设备驱动中实现。支持什么功能取决于你使用什么设备和设备驱动的版本。设备自己也可以支持所希望的功能。通过设定设备驱动参数可以调整特定的功能。你可以调整/etc/modules.conf测试某些性能。关于使用方法和技巧请参见设备和驱动相关文档。

1.4.5 RAID和存储系统

就系统效能而言存储系统的选择配置与RAID类型也是重要因素。Linux支持软RAID，但此主题的详细内容已超出本文范畴，我将在4.6.1“安装Linux前硬件的选择”中介绍一些调优相关的内容。

1.5 网络子系统

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

1.5.1 网络实现 1.5.2 TCP/IP 1.5.3 减负 1.5.4 绑定

网络子系统是性能方面另一个重要的子系统。网络除了Linux以外还会与其它很多设备交互，如交换机、路由器、网关、PC客户端等等。尽管这些设备已超出Linux的控制范围，但它们仍会对系统整体性能产生影响。不要忘记我们现在工作生活在一个网络时代。

在此我们将主要关注Linux是怎样处理网络操作的。

1.5.1 网络的实现

TCP/IP协议有着与OSI模型相似的层次结构。Linux网络实现使用了类似的方法。图1-23展示了Linux TCP/IP堆栈的层次结构和TCP/IP通讯的概览。

图1-23 网络层次结构和网络运行概览

像许多UNIX系统一样Linux的TCP/IP网络使用套接字接口。套接字为用户应用程序提供使用接口。让我们将一起来了解一下网络传输时必要的操作步骤。

1. 当要传送数据到对方主机时，应用程序创建所要传送的数据。
2. 应用程序打开套接字并向套接字写入数据。
3. 使用套接字缓冲(socket buffer)接口来处理需要传输的数据。套接字缓冲拥有数据的引用并通过所有层将其向下传递。
4. 在每一层里，都会执行相应的操作如解析数据包头、添加修改数据包头、检查大小、路由操作、分片等。当套接字缓冲通过这些层向下传递时，其数据本身不会被复制到每个层，因为将实际的数据复制到每个层是很没有效率的，内核通过及时更改套接字缓冲的引用并传递到下一层来节省没有必要的开销。
5. 最后数据通过网卡传输到线路中。
6. 以太网帧到达对方主机的网络接口。
7. 如果与网卡的MAC吻合，帧会被转到网卡的缓冲。
8. 网卡最后会将数据包移到套接字缓冲并向CPU发出一个硬终端。
9. 然后CPU处理数据包并通过所有层向上传递直到其到达应用程序（如Apache）的端口。

套接字缓冲

正如前面所述，内核使用缓冲来发送和接受数据。图1-24展示了网络中所用的缓冲。它们可以通过/proc/sys/net下的文件来调整。

/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

有时它可以影响网络性能。我们将在4.7.4“增加网络缓冲”中详细介绍。

图1-24 套接字缓冲的内存分配

网络API（NAPI）

在新的网络API中网络子系统发生了一些变化。Linux中网络堆栈的标准实现更加关注可靠性和低延时而不是低开销和高吞吐量。当创建防火墙时这些特点适合的，但大多数企业应用如文件打印或数据库的运行速度要比相似情况下windows要慢。

如图1-25中蓝色箭头所描述的，在传统处理网络数据包的方法中，网卡最后将数据包转移至操作系统内核的网络缓冲中并向CPU发出一个硬中断。

虽然这只是简单描述了处理网络数据包的过程，但其反应了这种方法的弊病。每次匹配MAC地址的以太网帧到网络接口时，都会产生一个硬中断。无论何时也不管CPU在做什么都不得不要停止下来处理这个硬中断，这会导致一个上下文交换并要清空相应的处理器缓存。如果只有很少的数据包到达接口，你可能认为这并不是问题，但在千兆网和现代的应用程序每秒钟可以创建数千个数据包的情况下，这会导致大量中断和上下文交换的发生。

图1-25 Linux网路堆栈

因此为减少处理网络通讯的开销，NAPI被采用。当第一个数据包到来时，NAPI像传统方式一样发出一个中断。但此后，网路接口进入轮询模式(Polling Mode)。即使在网络接口的DMA环缓冲中有数据包，也不会产生新的中断，这可以有效减少上下文交换和相应的开销。直到最后一个数据包被处理完并且环缓冲为空时，网卡重新恢复到中断模式【Interrupt Mode】。
NAPI另外的好处就是其创建的软中断可以由多个处理器处理从而提升多处理器的扩展性。NAPI为大多数企业级多处理器系统带来明显改善，但它需要支援NAPI的驱动。正如我们在本文调优章节中所阐明的，这里有着很大的调优空间。

Netfilter

作为内核的一部分Linux有着先进的防火墙功能。这个功能由Netfilter模组实现。你可以使用iptables工具来调整和配置Netfilter。

一般而言，Netfilter提供以下功能。

- 包过滤：当数据包符合某个规则时，Netfilter接受或拒绝此数据包，或者根据拒绝规则执行一个特定动作。
- 地址转换：当数据包符合某个规则时，Netfilter将变更此数据包来符合地址转换的要求。

匹配过滤可以定义以下属性。

- 网路接口
- IP地址、IP地址范围、子网
- 协议
- ICMP类型
- 端口
- TCP标志
- 状态（参见“连接追踪”）

图1-26显示数据包是怎样通过Netfilter链的，Nefilter链是指依次应用在每个节点上的一组已定义规则。

图1-26 Netfilter中数据包的流向

如果数据包与规则匹配Netfilter会执行相应的动作，这个动作被叫做target，可能的target有：

ACCEPT：接受数据包并让其通过

DROP：默默地将数据包丢弃

REJECT：丢弃数据包，向发送端回传数据包如ICMP端口不能到达。重置源主机TCP。

LOG：记录匹配的数据包。

MASQUERADE,SNAT,DNAT,REDIRECT：地址转换。

连接追踪

为获得更精致的防火墙功能，Netfilter使用连接追踪机制记录所有网络通讯的状态。使用TCP连接状态（参考“建立连接”）和其他网络属性（如IP地址、端口、协议、序列号、ack number、ICMP类型等），Netfilter将包分为下面四种状态。

NEW：试图建立新连接的包

ESTABLISHED：通过连接的包

RELATED：与前面的包相关的包

INVALID：由于损坏或无效而未知状态的包

另外，Netfilter能使用单独的模组分析协议的特定属性和操作来获得更为详细的连接追踪。例如用于FTP、NetBIOS、TFTP、IRC等的连接追踪模组。

1.5.2 TCP/IP

TCP/IP作为默认的网络协议已经有很多年了。Linux的TCP/IP实现与标准完全兼容。为了更好的性能调优，你应该熟悉基本的TCP/IP网络。

要了解详细内容，参见《TCP/IP Tutorial and Technical Overview》，GG24-3376

连接的建立

在应用数据传输前，需要在客户端和服务器端建立连接。连接建立的过程叫做TCP/IP三次握手。图1-27展示了连接建立到结束的过程。

1. 客户端发送一个SYN包（设置SYN标志的包）给对方主机请求连接。
2. 服务器端接收到包并发送一个SYN+ACK包。
3. 然后客户端发送一个ACK包到对方完成连接的建立。

一旦连接建立，应用程序的数据就可以通过连接传输。当所有的数据传输完成，连接结束流程开始。

1. 客户端发送一个FIN包给服务器段，开始连接结束流程。
2. 服务器端发送FIN的确认，如果没有数据需要传给客户端，发送FIN包到客户端。
3. 客户端发送ACK包到服务器端结束连接。

图1-27 TCP三次握手

在此会话过程中，连接状态会发生改变。图1-28展示了TCP/IP连接的状态图。

图1-28 TCP连接的状态图

你可以使用netstat命令了解每个TCP/IP会话的连接状态。更多内容请看2.3.11“netstat”。

流量控制(Traffic control)

TCP/IP实现中有一种机制来保证数据的有效传输和确保即使在网路传输质量很差又拥堵时包也可以顺利传递。

TCP/IP传输窗口

在Linux操作系统效能中传输窗口策略是TCP/IP实现的重要方面。基本上，TCP传输窗口就是在对方请求确认前指定主机能发送和接收数据的最大数量。窗口大小由接收主机通过TCP数据包头中窗口大小栏位提供给发送主机。使用传输窗口，主机可以更有效地发送包，因为发送主机没有必要等待每个发送包的确认。它提高网络利用率。也会提升延迟确认效率。TCP窗口开始时很小，随着来自对方的每此成功确认慢慢增长。想要优化窗口大小，见4.7.4“增加网络缓冲”。

图1-29滑动窗口和延迟确认

作为一个选项，高速网络使用一个名叫窗口缩放(Windows Scaling)的技术来增加最大传输窗口大小。我们将在“TCP选项调优”分析这些实现的影响。

重传(Retransmission)

在连接建立、结束和数据传输过程中，由于一些原因（网卡故障、路由缓慢、网路堵塞、buggy network implementation等）会发生多次超时和数据重传。为解决这种状况TCP/IP将包放入队列并尝试多次发送包。

你可以通过配置参数来改变某些内核行为。你可能想当网络丢包率高时增加尝试TCP SYN连接建立包的数量。你也可以通过/proc/sys/net下的文件改变某些超时临界值。更多信息，请看“调优TCP行为”。

1.5.3 减负(offload)

如果你系统中的网络适配器支持硬件减负功能，内核可以将其部分任务转移到适配器上执行，这样可以减少CPU的使用。

- 校验和减负

执行IP/TCP/UDP校验和，通过比较协议头中校验和字段的值和计算包数据的值来为确保包被正确的传输。

- TCP分段减负（TSO）

当数据大小超过目标网络适配器的最大传输单元（MTU）时，数据被分割成MTU大小的包，这个动作由适配器代表内核来完成。

关于更多的网络进阶特性，参见“Tuning IBM System x Servers for Performance”和10.3 网路进阶特性。

1.5.4 绑定模组

Linux内核通过使用绑定(bonding)驱动程序提供将网络接口集成在一起的功能。这是设备独立的绑定驱动。（也有设备特定的驱动）绑定驱动支援802.3连接的集成规范和一些原始的负载平衡和容错实现。它可以获得更高等级的可用性和效能的提升。请参见内核相关文档Documention/networking/bonding.txt

1.6 了解Linux性能指标(原创翻译)

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

1.6.1 处理器性能指标 1.6.2 内存性能指标 1.6.3 网络性能指标 1.6.4 块设备性能指标

在我们了解Linux操作系统中各种调优参数和性能度量工具前，有必要讨论一下关于系统性能的各种可用指标和他们的意义。由于Linux是一个开源的操作系统，所以有很多性能度量工具可以使用。你最后选择哪个工具取决于你的个人喜好和数据量以及你需要的详细程度。虽然有很多工具可以使用，但所有的性能度量工具都测量相同的指标，所以理解这些指标能让你使用任何你所碰到工具。为此我们只涉及了最重要的一些指标。有许多更详细的指标可以用来做深入分析，但这些已超出了本文的范畴。

1.6.1 处理器性能指标

下面是关于处理器的性能指标。

- CPU使用率(CPU Utilization)

这可能是最直接的指标了，它表示每个处理器的整体使用率。在IBM System x架构中，如果在持续一段时间里CPU使用率超过80%，就可能预示着CPU出现了瓶颈。

- 用户时间(User Time)

表示用户进程所花费的CPU百分比，包括Nice时间。在用户时间值很高的情况下，表明系统正在执行实际的工作。

- 系统时间(System Time)

表示内核操作所花费的CPU百分比，包括硬中断(IRQ)和软中断[SoftIRQ]。系统时间值持续很高表明网络或驱动器堆栈可能存在瓶颈。通常系统只花费很少时间在内核时间上。

- 等待(Waiting)

花费在等待I/O操作所需的CPU时间总和，与阻塞(Blocked)值相似，系统不应该花费过多的时间等待I/O操作；否则你应该检查一下I/O子系统各方面性能。

- 空闲时间(Idle time)

表示CPU空闲的百分比。

- Nice时间(Nice time)

表示花费在执行re-nicing（改变进程的执行顺序和优先级）进程的CPU百分比。

- 平均负载(Load average)

平均负载不是百分比，它是下面数值之和的平均值：

– 队列中等待执行的进程数 – 等待不可中断任务执行完成的进程数。

也就是TASK_RUNNING和TASK_UNINTERRUPTIBLE之和的平均值。如果请求CPU时间的进程发生阻塞（），平均负载将

会上升。相反如果每个进程都可以立即执行不会错过CPU周期，平均负载就会降低。

- 可运行进程(Runable processes)

这个值表示准备执行的进程。这个值在持续一段时间按内应该不会超过物理处理器数量的10倍，否则CPU可能存在瓶颈。

- 堵塞(Blocked)

在等待I/O操作完成前，进程是不能继续执行。进程堵塞可能意味着I/O存在瓶颈。

- 上下文交换(Context switch)

系统中进程之间进行交换的数量。上下文交换次数过多与大量的中断有关，这可能暗示着驱动器或应用程序存在问题。通常是不需要上下文交换的，因为每次只需要刷新CPU缓存，但有些上下文交换是必要的。参见1.1.5“上下文交换”。

- 中断(Interrupts)

中断数量中包括硬中断和软中断。硬中断会对系统性能产生非常不利的影响。高中断值表明软件存在瓶颈，可能是内核或者驱动。请记住中断值中也包括CPU始终所导致的中断。参见1.1.6“中断处理”。

1.6.2 内存性能指标

下面是关于内存的性能指标。

- 空闲内存(Free memory)

与其它操作系统相比，不必过分在意空闲内存值。正如1.2.2“虚拟内存管理”所述，Linux内核将大量未使用的内存分配作为文件系统缓存使用，所以在已用内存扣除用于缓冲和缓存的数量得到实际空闲内存。

- 交换空间使用(Swap usage)

这个值表示已使用的交换空间数量。正如1.2.2“虚拟内存管理”所述，交换空间的使用只能告诉你Linux在管理内存上是多么有效。要想确定内存是否存在瓶颈，Swap In/Out的数量才以为着用来。如果Swap In/Out长时间保持在每秒钟超过200到300页以上可能表示内存存在瓶颈。

- 缓冲与缓存(Buffer and cache)

被用来作为文件系统和块设备的缓存

- Slabs

表示内核所使用的内存。注意内核的页是不能被交换到硬盘上的。

- 活动与非活动内存(Active versus inactive memory)

提供关于活动内存的相关信息。非活动内存会作为候选被kswapd交换到硬盘。参见“页帧回收”

1.6.3 网络性能指标

下面是关于网络的性能指标。

- 已收到和已传送的封包(Packets received and sent)

这个指标能告诉你特定网卡已收到和已发送的封包数量

- 已收到和已传送的字节(Bytes received and sent)

这个值表示特定网卡已收到和已发送的字节数量。

- 每秒钟冲突数(Collisions per second)

这个值提供发生在指定网卡的网络冲突的数量。持续出现冲突值表示在网络架构中存在瓶颈而不是服务器。在大多数正确配置网络中，冲突时非常罕见的，除非网络架构是由hub组成的。

- 丢弃的封包(Packets dropped)

被内核丢弃的封包数，原因可能是防火墙配置问题或缺乏网络缓冲

- Overruns

Overruns表示超出网络接口缓冲的次数。这个指标可以与丢弃的封包数量配合来确定瓶颈是出自网络缓冲还是网络队列长度。

- 错误(Errors)

被标示为失败的帧的数量。这经常是由于网络不匹配或部分网线损坏引起的。对于铜缆千兆网部分网线损坏会产生严重的性能问题。

1.6.4 块设备性能指标

下面是关于块设备的性能指标。

- IO等待(Iowait)

CPU在等待I/O操作发生所花费的时间。如果这个值持续很高，很可能表示I/O存在瓶颈。

- 队列平均长度(Average queue length)

I/O请求的数量。通常硬盘队列值在2到3为最佳；过高可能表示硬盘I/O存在瓶颈。

- 平均等待时间(Average wait)

I/O请求服务所花费的平均时间。等待时间包括实际I/O操作的时间和在I/O队列中等待的时间。单位为毫秒ms。

- 每秒钟传输的数量(Transfers per second)

表示每秒钟执行了多少次I/O操作（包括读取和写入）。与每秒钟传输字节数(kBytes per second)结合可以帮助确定系统平均传输大小。平均传输大小通常要与硬盘子系统的条带大小一致。

- 每秒钟读写块的数量(Blocks read/write per second)

这个指标表示每秒钟读写块的数量，在2.6内核中块的大小为1024字节，早期的内核可以有不同的块大小，从512字节到4KB。

- 每秒钟读写字节的数量(Kilobytes per second read/write)

表示块设备读写的实际数据的数量，单位为KB。

第二章：监控和基准工具

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

Linux操作系统的开放和灵活性使其拥有大量性能监控工具。其中有些是原来UNIX上知名工具的Linux版，还有一些是专门为Linux设计的。大多Linux性能监控工具基于虚拟proc文件系统。要度量性能，我还需要使用一些适当的基准工具。

在本章中我们将介绍讨论部分性能监控工具和一些有用的命令，同时也会介绍一些有用的基准工具。

我们讨论的大部分监控工具都可以在Linux企业发行版中找到。

2.1 介绍

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

Linux企业发行版中都装备有许多监控工具。这些工具中有些是输出易于理解的系统活动结果，有些提供某些特定性能指标（如硬盘I/O）和详细信息。

熟悉这些工具能让你清楚当前系统的运行情况，帮助你找出影响系统性能的原因。

2.2 工具功能简介

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

表2-1 列出了本章所涉及到工具的功能。

表2-1 Linux性能监控工具

工具	主要功能
top	进程活动
vmstat	系统活动，硬件及系统信息
uptime，w	系统平均负载
ps，pstree	显示进程
free	内存使用情况
iostat	CPU平均负载，硬盘活动
sar	搜集展示系统活动
mpstat	多处理器使用情况
numastat	NUMA-相关统计信息
pmap	进程内存使用情况
netstat	网络相关统计信息
iptraf	实时网络统计信息
tcpdump/ethereal	网络通讯的详细分析
nmon	搜集展示系统活动
Proc文件系统	各种内核统计信息
KDE system guard	系统图形化实时展示
Gnome System Monitor	系统图形化实时展示

表2-2 列出了本章所涉及到基准工具的功能

表2-2 基准工具

工具	主要功能
lmbench	操作系统基准
iozone	文件系统基准
netperf	网络性能基准

2.3 监控工具

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

本章我们将讨论监控工具，这些工具大部分都来源于Linux企业版。你应该熟练掌握这些工具。

2.3.1 top

top命令显示实际进程的运行情况。在默认情况下，它显示服务器上占用CPU最多的任务并每5秒刷新列表一次。你也可以按PID（数值）、age（最新的排第一）、time（累计的时间）、常驻内存或时间（自启动开始进程所占用的CPU时间）排序。

你可以使用renice命令来修改进程的优先级。如果进程挂起或占据过多的CPU，你可以使用kill命令杀死此进程。

输出的栏位有：

PID：进程ID

USER：进程所有者的用户名

PRI：进程优先级（参见1.1.4“进程优先级和Nice值”）

NI：Nice值(Whether the process tries to be nice by adjusting the priority by the number given. See below for details.)

SIZE：进程使用的内存大小（code+data+stack），单位为KB

RSS：所使用的物理内存大小，单位为KB

SHARE：与其它进程共享的内存大小，单位为KB

STAT：进程状态：S=sleeping, R=running, T=stopped or traced,D=interruptible sleep, Z=zombie。在1.1.7“进程状态”讨论关于进程的状态

%CPU：使用的CPU百分比

%MEM：物理内存百分比

TIME：进程使用的CPU时间（自从此启动）

COMMAND：启动任务所使用的命令（包括参数）

top支持多个有用的热键，如下：

t：隐藏或显示摘要信息

m：隐藏或显示内存信息

A：分类显示各种系统资源，可用于快速找出系统中影响性能的任务。

o：在交互模式下选择排序的栏位

r：renice命令

k：kill命令

2.3.2 vmstat

vmstat能提供进程、内存、分页、块I/O、traps和CPU活动相关信息。vmstat命令可以显示平均值或实际取样值。使用取样频率和取样时间等参数启用vmstat取样模式。

注意：在取样模式中需要考虑实际数据搜集中存在误差的可能，设定较低的取样频率能减少这样的可能。

实例2-2 vmstat输出示例

注释：vmstat输出结果中第一行展示的是自最后一次启动以来的平均值，所以此行可以忽略。

输出栏位如下： Process（procs）

r ：等待运行时间的进程数。 b ：处于不可中断睡眠状态的进程数。

Memory

swpd ：虚拟内存使用量（KB）。 free ：空闲内存量（KB）。 buff ：用作buffer的内存量（KB）。 cache ：用作cache的内存量（KB）。

swap

si ：从硬盘交换到内存的数量（KBps）。 so ：交换到硬盘的内存数量（KBps）。

IO

bi：发送到块设备的块的数量（blocks/s）。 bo：从块设备获取的块的数量（blocks/s）。

System

in ：每秒钟的中断数量，包括时钟中断。 cs ：每秒钟上下文交换的数量。

CPU（整个CPU时间的百分比）

us ：花费在非内核代码的CPU时间（用户时间，包括Nice时间）。 sy：花费在内核代码的CPU时间（系统时间）。 id ：空闲时间。在2.5.41内核以前，还包括I/O等待时间。 wa ：IO等待时间。在2.5.41内核以前，显示为0。

vmstat命令提供了许多命令行参数，使用man手册查看参数的详细文档。常用的参数有：

-m ：显示内核的内存使用情况（slabs）

-a ：显示活动和非活动内存分页相关信息

-n ：只显示一次栏位名称行，当在取样模式通下将输出信息存储到文件时非常有用。（例如，root#vmstat –n 2 10 以每2秒钟的频率执行10次取样）

当使用-p{分区}参数时，vmstat也可以提供I/O相关统计信息。

2.3.3 uptime

uptime命令可以用来查看服务器已经运行了多久和曾经登录过的用户有多少，以及服务器的平均负载值是多少（参见1.6.1“处理器性能指标”）。它可以显示过去1分钟、5分钟、15分钟的系统平均负载值。

平均负载的最佳值是1，这意味着每个进程都可以立即执行不会错过CPU周期。负载的正常值在不同的系统中有着很大的差别。在单处理器的工作站中，1或2都是可以接受的。然而在多处理器的服务器上你可能看到8到10。

你能使用uptime来确定是服务器还是网络出了问题。例如如果网络应用程序运行，运行uptime来了解系统负载是否很高。如果负载不高，这个问题很有可能是由于网络引起的而你也非服务器。

提示：你可以使用w命令来代替uptime。w也提供关于当前系统登录用户和用户所进行工作的相关信息。

例子2-3：uptime输出示例

2.3.4 ps和pstree

ps和pstree是用于系统分析的基本命令。ps有3中不同风格的命令选项，UNIX风格、BSD风格和GNU风格。这里我们只介绍UNIX风格选项。

ps命令可以显示当前运行的进程列表。top命令也可以显示进程信息，但ps可以提供更加详细的内容。使用相应选项可以影响进程显示的数量。ps -A命令可以列出所有进程及其相应的进程ID（PID），当我们使用如pmap或renice等工具时会用到此PID。

当系统运行java应用时，ps -A产生的输出结果很容易填满整个显示区域，这导致很难得到所有运行中进程的完整视图。在这种情况下，pstree命令就派上用场了，它使用树状显示所有运行中的进程并合并所有的子进程（例如java线程），pstree命令有助于确认原始进程。还有另一个ps变种pgrep也非常有用。

例子2-4：ps输出示例

下面我们来了解一些常用的选项

-e ：所有进程。等同于-A -l ：显示长格式 -F ：附加全格式 -H ：显示进程的层次结构 -L ：显示线程，可能出现LWP和NLWP栏位 -m ：在进程后显示线程

下面的命令演示输出进程的详细信息：

ps -elFL

例子2-5：输出进程详细信息示例

输出栏位说明： F ：进程标志 S ：进程状态。S=sleeping，R=running，T=stopped/traced，D=interruptable sleep，Z=zombie。参见1.1.7“进程状态”中关于这些状态的介绍。 UID ：进程所有者（可能是启动者）用户名 PID ：进程ID PPID ：父进程ID LWP ：LWP(轻量级进程(light weight process),也称作线程)ID C ：处理器使用率百分比 NLWP ：进程中lwp（线程）的数量。（别名thcount） PRI ：进程的优先级。（参看1.1.4“进程优先级和Nice值”） NI ：Nice值（whether the process tries to be nice by adjusting the priority by the number given; see below for details） ADDR ：进程地址空间（不显示） SZ ：进程所有内存（code+data+stack）总数，单位为KB。 WCHAN ：内核功能名称，如果进程正在运行中 RSS ：常驻集大小，任务所使用的非交换物理内存（KB） PSR ：当前执行进程的处理器 STIME ：开始时间 TTY ：终端名称 TIME ：进程所用的CPU时间总数（自从启动） CMD ：启动任务的命令行（包括参数）

线程信息

你可以使用ps -L查看线程信息。

例子2-6：使用ps -L查看线程信息

2.3.5 free

命令/bin/free显示系统中内存空闲及使用情况。它也包括内核使用的buffer与cache的相关信息。

例子2-7：free命令输出结果示例

在使用free命令时，回忆一下Linux内存架构和虚拟内存管理方法。空闲内存值的用处市非常有限的，单纯统计交换空间使用率也不表示存在内存瓶颈。

图2-1：描述了free命令输出结果的基本含义。

图2-1：free命令的输出结果

free命令常用的参数包括：

-b, -k, -m, -g ：分别使用byte、KB、MB、GB为单位显示。

-l ：区分低内存和高内存（参见1.2“Linux内存架构”）

-c ：显示输出结果的次数

内存区(Zone)中使用的内存

使用-l选项，你可以了解到在每个内存区中有多少内存被使用。例子2-8和例子2-9分别展示了在32位系统和64位系统中free -l的输出结果。注意在64位系统中不再使用高内存。

例子2-8：32位系统中free命令的输出结果

例子2-9：64位系统中free命令的输出结果

你可以从/proc/buddyinfo文件中了解到每个内存区中有多少块内存可以使用。每列数值代表所需求大小的块可用的数量。例子2-10中，在ZONE_DMA中有5块2^2*PAGE_SIZE*可用，在*ZONE_DMA32*中有*16*块*2^3*PAGE_SIZE可用。回忆一下伙伴系统是怎样分配内存分页的（参见“伙伴系统”）。下面输出结果显示系统内存碎片的具体情况并让我们了解有到底多少分页可以分配。

例子2-10：64位系统中伙伴系统相关信息

2.3.6 iostat

iostat名利可以显示自从系统启动开始的CPU平均时间（与uptime相似）。它也可以产生关于服务器硬盘子系统活动情况的报告，报告包括两部分内容：CPU使用情况和硬盘使用情况。iostat可以用来获得I/O瓶颈的详细信息和系统性能调优，参见3.4.1“查找硬盘瓶颈”。iostat是sysstat工具包的一部分。

例子2-11：iostat输出结果

CPU使用情况分为四个部分：

%user ：显示用户级别（应用）所占用CPU的百分比。 %nice ：显示拥有Nice优先级用户级别所占用CPU的百分比。（优先级和nice级别会在2.3.7“nice，renice”中做相应介绍） %sys ：显示系统级别（内核）所占用CPU的百分比。 %idle ：显示空闲CPU的百分比。

设备使用情况分为下面几个部分：

Device ：块设备名称 tps ：设备每秒钟传输的数量（每秒钟I/O请求数）。多个单独的I/O请求可以合并到一个传输请求中，因为每个传输请求可以有不同的大小。 Blk_read/s，Blk_wrtn/s ：每秒钟块设备读写块的数量。可以设置不同的块大小，一般为1024,2048,4048字节，这取决于分区的容量。例如使用下面的命令来获得/dev/sda1的块大小： dumpe2fs -h /dev/sda1 |grep -F "Block size" 产生类似如下的输出： dumpe2fs 1.34 (25-Jul-2003) Block size: 1024 Blk_read，Blk_wrtn ：显示自系统启动后读写块的总数。

iostat提供很多选项，对于性能来说最为常用的就是-x，它可以显示进阶的统计信息，示例如下。

例子2-12：iostat -x输出结果

rrqm/s，wrqm/s ：每秒钟合并的读写请求数。多个单独的I/O请求可以合并到一个传输请求中，因为每个传输请求可以有不同的大小。 r/s，w/s ：每秒钟读写请求数。 rsec/s，wsec/s ：每秒钟读写的扇区数。 rkB/s，wkB/s ：每秒钟读写了多少KB。 avgrq-sz ：请求的平均大小，此值是以扇区为单位。 avgqu-sz ：请求队列的平均长度。 await ：显示系统级别（内核）所占用CPU的百分比。 svctm ：I/O请求的平均服务时间（单位为毫秒）。 %util ：I/O请求过程中CPU时间的百分比（设备的带宽使用率）。当接近100%时设备处于饱和状态。

计算I/O平均大小对于调整硬盘子系统访问模式非常有用。如下使用iostat -x -d，只显示有关于硬盘子系统的相关信息。

例子2-13：使用iostat -x -d分析I/O平均大小

在例子2-13的输出结果中，设备dasdc每秒钟写入12300.99KB的数据（kB_wrtn/s），这些数据被在一秒钟内分为2502.97次（w/s）I/O写入硬盘。I/O平均大小或请求平均大小为9.83块（avgrq-sz），本例中块大小为512字节。异步写操作平均I/O大小通常为奇数，然而大多数应用程序是以4KB的倍数（例如4KB,8KB,16KB,32KB等）来执行读写操作的。在上面的例子中应用程序随机写操作请求的大小为4KB，然而iostat显示平均请求大小为4.915KB，这种差别主要是由于Linux文件系统引起的，尽管我们执行的是随机写操作，确发现一些I/O被合并以便更有效地向硬盘写入数据。

注释：当文件系统使用默认的异步模式时，在iostat中只有请求平均大小的显示是正确的。尽管应用程序在执行写操作时求有明确请求的大小，但I/O层仍可能将大多数请求合并从而导致I/O平均大小的改变。

2.3.7 sar

sar命令用来搜集、报告和保存系统活动信息。sar命令由3个应用程序组成：sar，用来显示数据；sa1和sa2，用来搜集和保存数据。sar有非常多的选项，关于这些选项可以参看man手册。sar是sysstat工具包的一部分。

利用sa1和sa2，可以配置获得系统相关信息并记录下来用于后续的分析。

提示：我们建议在你的绝大多数系统上运行sar。这样一旦系统出现性能方面的问题，你手头就可以有个非常详细的系统信息，而这只需要少量的系统开销而且并不会增加额外的成本。

为达成这样的目的，需要在/etc/crontab添加相应配置（例子2-14）。在系统安装sar后，会自动建立一个cron工作每天执行sar。

例子2-14：使用cron启动日志自动报告

sar的原始数据保存于/var/log/sa/并根据月日分别保存在不同的文件中。要查阅你的结果，可以选择日期和所需的性能数据。例如要想显示从21日开始的网络计数器，使用sar -n DEV -f sa21命令并通过管道传给less，如例子2-15。

例子2-15：使用sar显示系统相关信息

你也可以通过命令行执行sar获得接近实时的报告（例子2-16）。

例子2-16：监控特定CPU

通过搜集到的数据，你可以看到CPU使用率（%user,%nice,%system,%idle）、内存分页、网络I/O和传输情况、进程创建活动、块设备活动和每秒钟的中断数量。

2.3.8 mpstat

在多处理器服务器上使用mpstat命令可以显示所有可用处理器的使用情况，也会显示所有CPU活动的平均值。mpstat是sysstat工具包的一部分。

mpstat能显示每个CPU的全面信息。mpstat也可以像vmstat命令设定取样频率和取样次数来使用取样模式生成相关统计信息。在例子2-17中执行mpstat -P ALL来显示所有处理器的CPU平均使用率。

例子2-17：多处理器系统中mpstat命令的输出结果

使用下面的命令，每间隔一秒钟显示一次所有处理器的使用情况：

mpstat -P ALL 1 2

例子2-18：在两路服务器中mpstat的输出示例

要了解mpstat完整的语法规则，输入：

mpstat -?

2.3.9 numastat

使用非一致性内存架构（NUMA）的系统如IBM System x 3950，如今NUMA架构在企业数据中心已经成为主流。然而NUMA却为性能调优过程带来新的挑战。如内存局部性问题在NUMA系统出现前是不用考虑的。幸运的是，企业版Linux提供监控NUMA架构行为的工具。numastat能提供关于使用本地内存与远端内存的比例和各节点内存配置的相关信息。在numa_miss栏中显示分配失败的本地内存，在numa_foreign栏中显示分配的远端内存（较慢的内存）。过多分配远端内存会增加系统延迟并可能降低整体性能。绑定节点的进程使用本地RAM的内存映射可以大大提高系统性能。

例子2-19：numastat输出结果示例

2.3.10 pmap

pmap命令可以显示一个或多个进程所使用的内存数量。你可以使用这个工具来了解服务器上的某个进程分配了多少内存，并以此来判断这是否是导致内存瓶颈的原因。要得到更加详细的信息，使用pmap -d选项。

例子2-20：init进程相关的内存信息

在输出结果的最后一行显示了一些重要的信息，如下：

mapped ：映射到文件的内存数量

writable/private ：进程所占用的私有地址空间数量

shared ：与其它进程共享的地址空间数量

你也可以查看地址空间所存储的信息。当分别在32位和64位系统中执行pmap，你能发现一个有趣差异。关于pmap的完整语法，请执行：

pmap -?

2.3.11 netstat

netstat是常用工具之一。如果你在网络环境中工作，应该对它非常熟悉。它可以显示许多网络相关信息如套接字使用情况、路由、接口、协议、网络数据统计等。下面是以下基本的选项：

-a ：显示所有套接字信息

-r ：显示路由信息

-i ：显示网络接口信息

-s ：显示网络协议信息

还有很多其他有用的选项，请查看man手册。下面的例子显示了套接字相关信息的输出结果。

例子2-21：使用netstat显示套接字相关信息

套接字信息

Proto ：套接字使用的协议（tcp,udp,raw）。

Recv-Q ：连接此套接字的用户程序为复制的字节数量。

Send-Q ：远程主机未确认的字节数量。

Local Address ：本地套接字的地址和端口。除非使用--numeric（-n）选项，否则套接字地址将被解析成域名（FQDN）端口号则被转换成相应的服务名称。

Foreign Address ：远程主机套接字的地址和端口号。

State ：套接字状态。由于在原始模式(raw mode)和UDP没有状态，此列显示为空。关于可能显示的状态，请参见图1-28或者查看man手册。

2.3.12 iptraf

iptraf可以实时监控TCP/IP网络流量并实时产生相关报告。它可以以接口和协议来显示每个会话的TCP/IP流量的统计信息。iptraf是由iptraf工具包提供的。

iptraf为我们提供如下信息：

- IP traffic monitor: TCP连接相关的网络传输统计信息
- General interface statistics: 网络接口的网络传输统计信息
- Detailed interface statistics: 按协议分类显示网络传输统计信息
- Statistical breakdowns: 按TCP/UDP和封包大小分类显示网络传输统计信息
- LAN station monitor: 按第二层地址分类显示网络传输统计信息

下面是由iptraf产生的报告。

图2-2：使用iptraf按协议分类显示TCP/IP网络传输统计信息

图2-2：使用iptraf按封包大小分类显示TCP/IP网络传输统计信息

2.3.13 tcpdump / ethereal 翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

---

tcpdump和ethereal可以用来获取和分析网络通讯活动，他们都是使用libpcap库来捕获网络封包的。在混杂模式下他们可以监控网络适配器的所有通讯活动并捕获网卡所接收的所有帧。要想设置网络接口为混杂模式并执行这些命令来捕获所有的网络封包，需要具有超级用户的权限。

你可以使用这些工具来探究网络相关问题。你可以发现TCP/IP重发、窗口大小的缩放、名字解析的问题、网络配置错误等。注意这些工具只能监控网络适配器所接收到的帧，并不能监控到整个网络的通讯情况。

tcpdump

tcpdump是一个简单却很强大的工具。它提供基本协议的分析功能，让你获得网络使用的概况。tcpdump支持许多选项和复杂的表达式用于过滤所要捕获的帧（捕获过滤器(Capture filter)），下面让我们一起来了解一下。

选项：

-i <接口> ：网络接口

-e ：打印链路层(link-level)头信息

-s ：从每个封包中截取字节

-n ：不执行DNS解析

-w <文件名> ：写入文件

-r <文件名> ：从文件中读取

-v, -vv, -vvv ：输出详细的结果

捕获过滤器表达式：

关键字：host dst, src, port, src port, dst port, tcp, udp, icmp, net, dst net, src net, 等

下面操作符可以用来合并多个Primitive：

非（'!'或者'not'）

与（'&&'或者'and'）

或（'||'或者'or'）

常用表达式示例：

- DNS查询封包

tcpdump -i eth0 'udp port 53'

- 连接192.168.1.10的FTP控制和数据会话。

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

- 连接到192.168.2.253的HTTP会话

tcpdump -ni eth0 'dst 192.168.2.253 and tcp and port 80'

- 连接到192.138.2.0/24子网的Telnet会话

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

- 源地址与目的地址不在192.168.1.0/24子网内并且TCP SYN或TCP FIN标志开启（TCP连接建立或结束）

tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net 192.168.1.0/24'

例子2-22：tcpdump输出结果示例

关于更多详细内容参见man手册

ethereal

ethereal与tcpdump有着相似的功能但更加复杂，有进阶的协议分析和报告功能。它可以通过图形化界面或

命令行来使用，它属于ethereal工具包的一部分。

像tcpdump一样它也可以使用捕获过滤器，并且还支持显示过滤器，这样可以减少帧。这里有一些常用表达

式例子：

- 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 == 0x12 tcp.options.mss_val == 1460 and tcp.option.sack == 1

- Application

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

图2-4：ethereal图形化界面

2.3.14 nmon

nmon是Nigel's Monitor的简称，由Nigel Griffitchs开发的一款常用系统性能监控工具。因为nmon包含多种子系统的性能信息，可以用作性能监控的单一来源。使用nmon可以多得任务的相关性能信息包括处理器使用率、内存使用率、运行队列信息、硬盘I/O相关统计信息、网络I/O相关统计信息、分页活动情况和进程性能指标。

要运行nmon，只需启动该工具并输入所感兴趣子系统对应的字母即可。例如要获得CPU、内存、硬盘的相关信息，启动nmon并输入c m d。

在nmon中一个非常有用的功能就是将性能相关统计信息保存到CSV文件中用于后续分析。在表格处理软件中导入nmon输出的CSV文件来产生图形化报表。要达成这样的目的，nmon启动时需要使用-f参数（使用nmon -h查看详细内容）。例如执行nmon每30秒钟快照一次连续一个小时获取数据，可以使用例子2-23的命令。

例子2-23：使用nmon来记录性能相关数据

上面例子的输出结果将被存储在当前目录的一个文本文件中，文件名为_date_time.nmon。

关于nmon的更多信息，我们建议你访问

http://www-941.haw.ibm.com/collaboration/wiki/display/WikiPtype/nmon

下载nmon，访问

http://www.ibm.com/collaboration/wiki/display/WikiPtype/nmonanalyser

2.3.15 strace

strace命令可以拦截并记录进程所使用的系统调用和进程所接收到的信号。这是一个非常有用的诊断、指导

和调试工具。系统管理员使用其来解决应用程序相关问题。

要跟踪某个进程，需要指定被监控进程的进程ID(PID)：

strace -p

例子2-24展示了strace的输出结果。

例子2-24：strace监控httpd进程的输出结果

注意：当针对某个进程执行strace命令，此进程的性能将大大降低，所以只有在搜集数据时才执行此命令。

这里有另外一个有趣的用法。此命令可以报告在执行某个命令时被系统调用消耗了多少内核时间。

strace -c

例子2-25：strace关于系统时间的计算结果

关于strace命令的完整用法，输入：

strace -?

2.3.16 proc文件系统

proc文件系统并不是一个实际的文件系统，然而他却非常有用。它并不是用来存储数据的，而是提供了运行中内核的接口。proc文件系统能让管理员随时监控和调整内核。图2-5展示了proc文件系统的示例。大多数Linux性能量度工具都要依靠/proc所提供的信息。

在proc文件系统中，有几个子目录我们从名字就能辨别出他们的用途，但proc目录中的大部分信息是很难读懂的，你最好使用像vmstat这样的工具通过更易读的方式来显示各种统计信息。注意在不同的系统架构中proc文件系统的布局和所包含信息有所不同。

- /proc目录中文件

proc根目录下的文件涉及多种有关系统的统计信息。你可以使用如vmstat和cpuinfo来显示这些信息。

- 数字1到X

有一些以数字命名的子目录，代表运行中进程和相应的进程ID（PID）。目录结构总是以PID 1开始，它代表init进程。每个以数字命名的子目录都存储着所对应进程的统计信息，如进程所映射的虚拟内存。

- acpi

ACPI表示高级配置和电源管理接口，大多数台式电脑和笔记本电脑现在都支持此技术。因为ACPI主要为PC技术，在服务器系统中通常被禁用。关于更多ACPI的资料参考：http://www.apci.info

- bus

这个子目录包含总线子系统的相关信息，如PCI总线或USB接口。

- net

net子目录包含关于网络接口的大量原始统计数据，如接收的广播封包或每个网络接口的路由信息。

- scsi

此目录包含SCSI子系统相关信息，如连接的设备或驱动的版本。在大多数IBM System x服务器中会有一个名叫ips的目录用来记录IBM ServeRAID控制器相关信息。

- sys

在sys子目录下你能找到可以调整的内核参数如虚拟内存管理或网络堆栈。我将在4.3“Changing kernel parameters”中介绍一些选项和可调值。

- tty

tty子目录包含系统各虚拟终端相关信息和它们连接了哪些物理设备。

2.3.17 KDE System Guard

KDE System Guard (KSysguard)是KDE的任务管理和性能监控工具。它采用client/server架构，可以监控本机也可以监控远端主机。

图2-6：KDE System Guard默认的窗口

前端图形界面使用传感器（sensors）获得要显示的信息。传感器返回的可以是一个简单的数值或更复杂的信息如表格。针对不同的信息类型都提供了一个或多个显示界面。这些显示界面被组织在多个工作表中，工作表可以独立存储和加载。

KSysguard主窗体包括菜单栏、工具栏和状态栏、传感器浏览区以及工作区。当初次启动后，你可以看到默认的界面：本机localhost列在传感器浏览区中，在工作区中有两个标签。

每个传感器监控一个特定的系统值。所有的传感器都可以拖拽至工作区。

- 在工作区中删除和覆盖传感器
- 编辑工作表属性，增加行和列的数量
- 创建一个新的工作表，放入你所需要的传感器

工作区(Workspace)

图2-7中工作区有两个标签：

- 系统负载，首次启动KSysguard时默认的视图。
- 进程表

图2-7：KDE System Guard传感器浏览

系统负载(System Load)

系统负载工作表有四个传感器窗口：CPU负载，平均负载（1分钟），物理内存，和交换内存。多个传感器可以显示在同一窗口中。要了解窗口显示的是哪些传感器，将鼠标移动到图形上就会显示对应的说明。你也可以在图形上右键点击Properties，然后点击Sensors标签（图2-8）。这里也显示了各传感器在图形中所对应的颜色。

图2-8：传感器相关信息

进程表(Process Table)

点击Process Table标签可以显示服务器中所有运行中进程的相关信息（图2-9）。默认情况下此表格是按系统CPU使用率进行排序的，但也可以通过点击不同表头使用相应的栏位排序。

图2-9：进程表视图

设置工作表

对于你希望监控的环境或特定区域，你可能需要使用多个不同的监控传感器。最好的方法就是创建一个自定义的工作表。

在本节中，我们将一步步指导你创建图2-12那样的工作表。

1. 点击File->New，创建一个空工作表。

图2-10：新工作表的属性

1. 输入标题和行列的数量；这将决定监控窗口的最大数量，在我们的示例中为4个。填完信息后，点击OK创建一个空的工作表，如图2-11。

注释：刷新时间间隔为2秒钟。

图2-11：空工作表

1. 要增加传感器，只需从窗口左边拖拽传感器到右边空闲区域。显示的类型共有下面几种：

2. Signal Plotter：这种类型可以同时显示一个或多个传感器。如果需要显示多个传感器，会使用不同的颜色来标示。如果显示屏足够大，会使用网格来显示所显示数据的区间段。

默认情况下显示为自动范围模式(automatic range mode)，最小和最大值会自动设定。如果你想改变最小和最大值，你需要禁止自动范围模式，然后在Properties对话框（在图表中点击右键）的Scales标签中设置最小和最大值。

- Multimeter：以数字方式显示传感器的值。在Properties对话框中你可以定义上下限。如果超出限制时，使用警告色显示。
- BarGraph：以条形显示传感器的值。在Properties对话框中你可以定义上下限。如果超出限制时，使用警告色显示。
- Sensor Logger：不显示任何值，而是将上述信息和日期时间一起记录到文件中。

对于每个传感器，你必须定义一个日志文件，传感器记录的间隔时间和是否需要报警。

1. 点击File->Save保存工作表

注释：保存工作表，它将被存储在用户家目录中，其它的管理员不能访问你自定义的工作表。

图2-12：工作表示例

要获得更多关于KDE System Guard，请访问：

http://docs.kde.org/

2.3.18 Gnome System Monitor

尽管功能不像KDE System Guard那么强大，但作为Gnome桌面环境的图形化性能分析工具，Gnome System Monitor可以图形化显示性能相关系统资源用于发现可能的峰值和瓶颈。 产生的所有统计信息都是实时，要想做长时间的性能分析需使用其它工具来完成。

2.3.19 Capacity Manager

Capacity Manager是IBM Director系统管理套件中附带的工具，需要ServerPlus Pach下运行。Capacity Manager让跨系统平台长期对进行性能度量变成可能。Capacity Manager可以进行容量计划，评估未来系统容量需求。你可以将报告导出为HTML,XML和GIF文件，这些文件会被自动存储在企业内存的web服务器上。IBM Director可以在不同的系统平台中使用，这使得在复杂环境中搜集和分析数据变得容易。关于Capacity Manager的详细内容请参考Tuning IBM System x Servers for Performance, SG24-587。

要使用Capacity Manager，你首先要安装相应的RPM包。安装RPM后，在IBM Director Console中选择Capacity Manager->Monitor Activator

图2-13：IBM Director Console任务列表

拖放Monitor Activator图标到单独或一组安装有Capacity Manager包的系统上。在打开的窗口中你可以选择需要监控的子系统。Linux版Capacity Manager不支持所有可用的性能计数器。系统统计信息受一些性能参数的限制。

图2-14：启用性能监控器

Monitor Activator窗口的右面显示相关系统及当前状态，在左面显示各种可用的性能监控器。要添加新的监控器，选择所需监控器并点击On。关闭Monitor Activator窗口后所做改变立即生效。完成此步后，IBM Director开始搜集所需的性能指标并存储在各系统的临时位置中。

要创建搜集数据的报告，选择Capacity Manager->Report Generator（参见图2-13）并拖动图标到单独或一组系统上。IBM Director将询问是否要立即产生还是排定后续执行（图2-15）。

图2-15：报告排程

在生产环境中，使用Capacity Manager定期产生报告是一个不错的做法。我们的经验是在每周末的非工作时间执行。立即产生报告还是排定时间产生报告取决于你的选择。报告完成后会被存储在中央IBM Director管理服务器上，可以使用Report Viewer来查看报告。图2-16展示了Capacity Manager月报告的示例。

图2-16：Capacity Manager报告示例

在Report Viewer窗口中你可以选择不同的性能计数器。

Capacity Manager所搜集的数据可以导出成HTML或XML用来显示在内部web服务器上或后续分析使用。

2.4 基准工具

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

在本章节中我们将讨论几个主要的基准工具。要度量性能使用优秀的基准工具是必要的。这里有一些优秀的工具，它们拥有全部或部分下面的功能：

- Load generation
- 监控性能
- 监控系统使用情况
- 报告

基准只不过是某个特定工作负载的样本，它与系统运行的实际工作负载可能很接近也可能大相径庭。经管某系统自诩可以取得很好的Linpack成绩，但它仍可能不是一台理想的文件服务器。要牢记基准是不能模拟最终用户偶尔不可预知的反应。基准是不能告诉你当用户访问他们的数据或备份开始时文件服务器是怎样运作的。通常在系统中执行基准时要遵循以下规则。

- 使用服务器负载基准：服务器系统自诩拥有许多与桌面PC不同的特性，经管IBM System x平台分享了许多技术给桌面电脑。服务器基准可以生成多个线程使用系统SMP的能力来模拟现实中多用户环境。虽然PC开启Web浏览器的速度可能比高档服务器快，但服务器开启成千个Web浏览器的速度要比PC更快.
- 模拟预估的负载：所有的基准都有不同的选项，用于调整来满足系统预估未来的负载。如果应用不得不受制于硬盘的低相应，再强大的CPU都是没有的。
- 隔离基准系统：基准测试的系统首先要与其它负载尽可能的隔离。一个运行top命令的开放会话会对基准结果产生很大干扰。
- 平均结果：尽管你尝试隔离基准系统，在基准时仍可能会有未知因素影响系统效能。比较好的做法是运行基准至少三次取平均值，确保个别事件不会影响整个分析结果。

在下面的章节中，我们将基于这些原则来挑选工具。

- 工作在Linux之上：Linux为基准的目标
- 工作在所有的硬件平台之上：由于IBM就提出三种不同的硬件平台（假定IBM System p和IBM System i的硬件技术都是基于IBM POWER架构的），选择一个在所有架构上都很易用的基准是很重要的。
- 开源：Linux可运行在多个平台之上，所以如果没有源代码，二进制代码可能也无法使用。
- 完整的文档：要执行基准就必须要熟悉工具。文档可以帮助你熟悉工具。它也可以在你决定使用某个工具前，通过浏览概念、设计和详细内容，帮助评估工具是否符合你的要求，
- 维护活跃：陈旧被放弃的工具可能没有遵循最近的规范和技术。很可能产生错误的结果。
- 使用广泛：对于使用广泛的工具，你能找到更多的相关信息。
- 易用：你一定希望工具容易使用。
- 报表能力：报表能力能大大减少性能分析的工作

2.4.1 LMbench

LMbench是一套微基准，可以用来分析不同操作系统的设定，如启用SELinux与未启用SELinux的系统。包括LMbench在内的基准可以度量多种操作系统的例行程序，如上下文转换、本地通讯、内存带宽和文件操作。LMbench使用很简单，只需要

知道三个重要的命令。

- make results：首次运行LMbench时，它会提示系统配置的一些详细信息和哪些测试会被执行。
- make rerun：在初始化配置并完成首次运行基准后，使用make rerun命令可以重复基准使用配置由make results提供的。
- make see：最后在运行最少三次后，可以使用make see命令查看结果。显示的结果可以被复制到制表软件用于后续分析或图形化展示。

可以在 http://sourceforge.net/projects/lmbench/ 找到LMbench

2.4.2 IOzone

IOzone是一个文件系统的基准程序，可以用来模拟多种不同的硬盘访问模式。IOzone的配置非常丰富，所以可以比较准确的模拟特定负载。通过使用不同的块大小，IOzone可以写一个或多个不同大小的文件。

IOzone提供一个功能非常强大且自动化的基准测试模式，可以高效地定义负载属性如文件大小、I/O大小和访问模式。如果要评估满足数据库负载的文件系统，需要在IOzone创建一个随机访问模式大块的大文件代替顺序访问小块的大文件。IOzone的重要选项有：

-b

告诉IOzone将结果存储为微软Excel兼容的格式。

-C

显示每个子进程输出内容（可以用来检查所有子进程是否真正同时运行）

-f

用来告诉IOzone数据存储的位置。

-i

这个选项用来定义运行哪些测试。首次运行你不得不定义-i 0来生成测试文件。常用的测试有-i 1用于顺序读取、-i 2用于随机读写访问和-i 8用于混合随机访问。

-h

显示帮助

-r

告诉IOzone测试的记录I/O大小。记录大小要尽可能的接近目标负载。

-k

使用2.6内核的异步I/O属性，这个经常用在数据库中，如IBM DB2。

-m

如果目标应用程序使用多个内部buffer，可以使用-m来模拟这种行为。

-s

指定基准测试的文件大小。对于异步文件系统（大多数文件系统的默认挂载选项）IOzone应使用至少两倍于系统内存的文件大小，以便能真实的度量硬盘的性能。也可以使用MB或GB指定大小，只要直接在文件大小后使用m或g即可。

-+u

在测试过程中，用于度量处理器使用情况的开关。

注释：任何基准使用的文件如果合适系统的内存并存储在异步的文件系统中，所测量的将是内存吞度量而不是硬盘子系统的性能。所以你应该使用sync选项挂载文件系统或使用大小为系统内存两倍的文件。

使用IOzone测量特定硬盘子系统随机读取的性能，测试文件存储在/perf，文件大小为10GB，I/O大小为32KB（这些特性模仿一个简单的数据库）。如下：

例子2-26：IOzone命令示例

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

最后，将获得的结果导入到你的制表软件中并转换成图表。使用图形化的输出可以让分析大量数据和确定趋势变得更容易。例子2-16的输出结果图形化显示如图2-17。

图2-17：例子2-26结的图形化结果

如果IOzone所使用文件大小与系统内存或cache相当，可以获得关于cache和内存吞吐量的数据。应该注意的是由于文件系统开销IOzone只能报告系统带宽的70-80%

你可以在http://www.iozone.org/找到IOzone。

2.4.3 netperf

netperf是一个专注于TCP/IP网络性能的基准工具。它支持UNIX domain socket和SCTP基准测试。

netperf是基于客户端-服务器模型。netserver运行在目标系统而netperf运行在客户端。netperf控制netserver并传送配置信息到netserver，产生网络通讯，通过一个控制链接从netserver得到结果，这个控制链接与实际的基准链接是分开的。在基准测试过程中，控制链接是不会产生任何流量的，所以不会对结果产生任何影响。netperf也具有报表功能包括CPU使用情况报表。本文在写作时稳定版本为2.4.3。

netperf可以产生几种类型的网络流量。大体上可以将它们分为两类：批量数据传输和请求/响应。你应该注意到netperf每次只能使用一个套接字。下个版本的netperf（netperf4）将全面支持并发会话的基准测试。此时我们可以执行多个会话的基准测试，正如下面所描述的。

- 批量数据传输

批量数据传输为网络基准测试中最为常用的量度指标。通过一秒钟传输的数据总量来测量批量数据传输。它模拟大文件的传输，如多媒体流和FTP数据传输。

- 请求/响应类型

这模拟了请求/响应类型的网络流量，测量每一秒钟交易的次数。对于在线交易应用程序请求/响应流量类型是典型的，如web服务器、数据库服务器、邮件服务器、文件服务器（管理小文件或中等文件）和目录服务器。在实际的环境中，同数据交换一样会话的建立和终结也会被执行。要模拟这个，需要采用TCP_CRR类型。

- 并发会话

netperf当前稳定版是不支持并发多会话的基准测试，但我们可以通过同时产生多了netperf实例来执行多个基准测试，如下：

---

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

&; done

我们来了解一些常用选项。

全局选项：

-A 变更远端系统发送和接收buffer alignment

-b Burst of packet in stream test

-H 远端主机

-t 测试的流量类型

TCP_STREAM 批量数据传输基准 TCP_MAERTS 与TCP_STREAM相似，除了流的方向相反。 TCP_SENDFILE 与TCP_STREAM相似，除了使用sendfile()代替send()，这导致零复制操作。 UDP_STREAM 与TCP_STREAM相似，除了使用UDP。 TCP_RR 请求/相应类型流量基准。 TCP_CC TCP连接/关闭基准。没有请求和响应包被交换。 TCP_CRR 执行连接/请求/相应/关闭操作。 与禁用HTTP keepalive的HTTP1.0/1.1会话相似。 UDP_RR 与TCP_RR相同除了使用UDP。

-l 基准测试长度。如果是正数，netperf将执行基准测试testlen秒钟。如果是负数，对于批量数据传输基准测试，会交换testlen字节数据或对于请求/响应类型，会完成testlen个交易。

-c 本地CPU使用率报告

-C 远端CPU使用率报告

注释：CPU使用率报告在一些平台上可能不是很准确。请在你执行基准测试前确认其准确性。

-I 这个选项用来确保结果的可信度。可信度级别应为99或95（百分比）和区间（百分比）可以设定。要保证结果达到特定的可信度级别，netperf会重复执行多次相同的基准测试。例如-I 99,5意味结果接近99%的真实结果区间在5%（+-2.5%）以内。

-i 测试循环次数的最大和最小值。此选项能限制循环的次数。-i 10,3意味着netperf执行相同的基准测试至少3次最多10次。如果循环超过最大值，结果将不能达到-I选项定义的可信度级别，在结果中将显示一个警告。

-s ,-S 改变本地和远端系统发送和接收buffer大小。这将影响窗口大小。

用于TCP_STREAM, TCP_MAERTS, TCP_SENDFILE, UDP_STREAM的选项

-m , -M 指定send(),recv()函数buffer的大小，控制每次呼叫发送和接收的大小。

用于TCP_RR, TCP_CC, TCP_CRR, UDP_RR的选项

-r , -R 指定请求和响应的大小。例如，-r 128,8129意味着netperf发送128字节数据包给netserver，然后netserver发送8129字节数据包回netperf。

下面为netperf的TCP_CRR类型基准的输出示例。

例子2-27：TCP_CRR基准结果示例

当你执行基准测试时，使用netperf自带的例子测试脚本是非常聪明做法。变更脚本中变量，你可以得到想要的基准测试。这些脚本存储在netperf包doc/examples/目录下。

关于更多详细内容，参见http://www.netperf.org/

2.4.4 其它有用工具

还有一些有用的基准工具。你必须要熟悉这些基准工具的特性，以便选择到符合你需求的工具。

表2-3：其它基准测试工具

第三章：分析性能瓶颈

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

本章将讲解怎样发现影响服务器的性能问题。我们总结了一系列步骤来指引你完成恢复服务器到一个可以接受的性能等级的具体方案。

本章报告以下内容：

- 3.1，“确认瓶颈”
- 3.2，“CPU瓶颈”
- 3.3，“内存瓶颈”
- 3.4，“硬盘瓶颈”
- 3.5，“网络瓶颈”

3.1 确认瓶颈 翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

下面的步骤可用来作为快速调优的策略：

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，找出原因。
5. 修复导致瓶颈的原因，每次只做一个变更。
6. 回到第3步，直到系统性能达到满意。

提示：你应该将每一个步骤记录在文件中，特别是你针对性能所做的变更和及其影响。

3.1.1 搜集信息

大多数情况下，你所能得到的第一手信息就是“服务器出现了问题”。所以利用一些探索性问题来弄清和记录下问题是非常重要的。这里列出一些问题用来帮助你更好的了解系统。

- 你能给我关于所涉及服务器的完整信息吗？
- 型号
- 使用年限
- 配置
- 外围设备
- 操作系统版本和更新等级
- 你能告诉我究竟出现了什么问题？
- 都有哪些症状？
- 描述一下错误信息。

对于有些人回答这个问题有些困难，但用户提供的任何额外信息都有可能帮助你找到问题。例如，用户可能说“当我拷贝文件到服务器时速度真的很慢。”。这就暗示网络有问题或硬盘子系统有问题。

- 谁受到这个问题的影响？

一个人、某些人还是整个组织受这个问题的影响？这可以帮助你判断问题是否出现在某段特定的网络中还是与应用相关等等。如果只有一个用户受此问题影响，很可能是用户的PC出了问题（或者是他们想象出来的）。

The perception clients have of the server is usually a key factor.从这个观点来说，性能问题并不一定与涉及服务器直接有关：位于服务器和客户端的网络极易成为导致问题的原因，这包括网络设备和其他提供服务的服务器，例如域控制器。

- 问题是否可以重现？

所有可以重现的问题都能被解决。如果你在系统方面经验丰富，你应该能找出问题根源并采取相应的措施。

问题的重现可以让你更好的了解和明白此问题。记录重现问题的相关步骤是十分必要的：

- 重现问题需要哪些步骤？

知道这些步骤可以帮助你在相同条件下不同的机器中再现相同的问题。如果可以，你将有机会使用测试机来代替崩溃的生产服务器。

- 是一个间歇性问题吗？

如果问题间歇性发生，第一件要做的是就是搜集信息并找到可以重现问题的规律，目标就是构建一个情境让问题可以随时发生。

- 问题在每天的特定时间或每周的特定某天发生吗？

这可能帮助你查明问题是由什么引起的。问题可能发生在大家上午上班或下午上班时，想办法改变现有作息时间（这可能减少问题发生的机会或发生的更加频繁）；以便让问题可以重现。

- 问题很少见吗？

如果问题不可以重现，你可能得出结果在特殊情况下问题才会发生并将其归类为已解决。在现实生活中，此问题还是极有可能再次发生的。

在排除难以重现的问题时，有效的措施就是：重启或将机器的驱动程序和补丁升级到最新。

- 问题是什么时候开始的？是渐渐的还是突然发生的？

如果性能问题是渐渐出现的，这很像是一个容量规划问题；如果它是突然出现的，很可能是由于服务器或外围设备的变更引起的。

- 服务器是否有做过变更（小的或大的）或客户端使用服务器的方法有改变吗？
- 客户是否改变过服务器或外围设备而导致了问题的发生？有网络变更的所有记录吗？

需求会随着业务的改变而改变，影响对服务器和网络系统的需求。

- 还涉及了其它的服务器或硬件吗？
- 有日志可以用吗？
- 问题的优先级是什么？什么时候问题必须解决？
- 必须在几分钟内解决还是允许在几天内解决？你可能有充分时间来解决问题；或已启动应急方案。
- 问题有多大？
- 相关的损失有哪些？

3.1.2 分析服务器性能

重要提示：在执行任何故障排除动作前，备份所有的数据和配置信息，防止其部分或全部丢失。

此时，你应开始监控服务器。最简单的方法就是在需要分析的服务器上运行监控工具。（参看第二章“监控和基准工具”）。

在运行高峰时（例如，上午9点到下午5点）记录服务器的性能日志；取决于有提供哪些服务和有哪些人在使用这些服务。在记录日志时如果可以应该包含下列字段：

处理器(Processor) 系统(System) 服务器工作队列(Server work queues) 内存(Memory) 分页文件(Page file) 物理硬盘(Physical disk) 重定向器(redirector) 网络接口(Network interface)

在你开始前，要牢记井然有序的进行性能调优是非常重要的。你可以使用我们推荐的流程为你的服务器进行调优，流程如下：

1.清楚影响服务器性能的因素。

2.测量出当前的性能作为基线，用于与后来的测量数据比较来识别出系统的瓶颈。

3.使用监控工具来识别性能瓶颈。按照下节介绍，你可以缩小瓶颈的范围到子系统级。

4.针对导致瓶颈的元件执行相应调整，提升服务器性能以满足需要。

注释：当服务器其它元件都有足够的能力来维持性能在一个较高的级别时，通过升级存在瓶颈的元件可以获得最好的效果。

5.对性能进行新的测量，对比调优前后的性能差异。

当尝试处理性能问题时，请记住下列事项：

- 应用程序应该使用适当的优化级别进行编译，这样可以少走弯路。
- 在你做任何升级和修改前执行测量，以便于确定变更是否有效果。（换句话说就是执行基线测量）
- 检查项目不应只有新增加的硬件，还要包括配置有更改的现有设备。

3.2 CPU瓶颈

翻译：飞哥 (http://hi.baidu.com/imlidapeng)

原文名称：《Linux Performance and Tuning Guidelines》

原文地址：http://www.redbooks.ibm.com/abstracts/redp4285.html

对服务器来说主要的角色就是应用服务器或数据库服务器，CPU作为关键资源经常成为性能瓶颈的根源。CPU使用率高并不总是意味着CPU工作繁忙，它有可能是正在等待其他子系统。在进行性能分析时，将所有子系统当做一个整体来看是非常重要的，因为在子系统中可能会出现瀑布效应。

注释：有种常见的错误观念认为CPU是服务器中最重要的。情况不总是这样，服务器经常是CPU的配置高，硬盘、内存和网络子系统是低配置。只有一些特定对CPU要求高的应用程序才能真正充分利用当今的高端处理器。

3.2.1 发现CPU瓶颈

有多种方法可以来确认CPU瓶颈。在第二章“监控和基准工具”中介绍到，Linux有很多工具帮助我们确认瓶颈，问题是使用哪一个。

其中一个工具是uptime。通过分析uptime输出，我能对在过去15分钟所发生的事情有个粗略的了解。关于此工具的更多说明，参见2.3.3“uptime”。

例子3-1：一个系统CPU资源紧张的uptime输出结果

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传感器可以让你了解当前CPU的工作负载。

提示：小心不要因为同时运行过多的工具而导致CPU问题。你可能发现当同时使用多个不同监控工具时会使CPU负载过高。

使用top，你可以看到CPU使用率及主要是哪些进程引起问题（例子2-1）。如果你已安装sar，搜集了包括CPU使用率的信息。但分析这些信息是很困难的，所以要使用isag，它可以将sar的输出转换成图形。否则你可以通过脚本解析这些信息并使用电子表格绘制CPU使用率的趋势图。你也可以在命令行中输入sar -u或者sar -U processornumber。要获得比单单CPU子系统更多关于系统及当前使用率的信息，一个不错的工具就是vmstat（参见2.3.2，“vmstat”）

3.2.2 SMP

基于SMP的系统会出现其特有且难于检测的问题。在SMP环境中，有个叫CPU亲和力(affinity)的概念，它允许你将一个进程绑定到指定的CPU。

主要用途是这有利于CPU cache的优化，它通过让进程在同一CPU运行代替在处理器间移动来实现。当进程在CPU间移动时，新CPU的cache会被清空。因此一个进程在处理器间移动会发生多次cache清空，这意味着一个单独的进程会花费更多的时间才能完成。这种情况非常难于发现，因为在监控时CPU负载可能非常均衡，不一定会出现某个CPU达到峰值的情况。亲和力在基于NUMA的系统中也很有用如IBM System x3950，where it is important to keep memory, cache, and CPU access local to one another.

3.2.3 性能调校选项

首先要确认系统性能问题是由CPU导致的而不是其他子系统。如果处理器为服务器瓶颈，可以通过相应调整来改善性能，这

包括：

- 使用ps -ef命令确保没有不必要的程序在后台运行。如果发现有不必要的程序，将其停止并使用cron将其安排在非高峰期运行。
- 通过使用top命令找出非关键性且消耗CPU较多的进程，并使用renice命令修改它们的优先级。
- 在基于SMP的机器中，尝试使用taskset将进程绑定到指定的CPU，确保进程不需要在处理器间忙碌，从而导致多次cache清空。
- 对于正在运行的应用程序，最好的办法是纵向升级（提升CPU频率）而不是横向升级（增加CPU数量）。这取决于你的应用程序是否能使用到多个处理器。例如一个单线程应用程序的升级方式最好是更换成更快的CPU而不是增加为多个CPU。
- 通常的做法还包括确认你所使用的是最新的驱动程序和韧体，因为这会影响CPU的负载。

3.3 内存瓶颈 On a Linux system, many programs run at the same time. These programs support multiple users, and some processes are more used than others. Some of these programs use a portion of memory while the rest are “sleeping.” When an application accesses cache, the performance increases because an in-memory access retrieves data, thereby eliminating the need to access slower disks. 在linux系统中，在同一时间有支持多个用户的多个程序在运行，它们对内存使用有多有少； 应该程序访问内存数据的效率要比磁盘高；

The OS uses an algorithm to control which programs will use physical memory and which are paged out. This is transparent to user programs. Page space is a file created by the OS on a disk partition to store user programs that are not currently in use. Typically, page sizes are 4 KB or 8 KB. In Linux, the page size is defined by using the variable EXEC_PAGESIZE in the include/asm-/param.h kernel header file. The process used to page a process out to disk is called pageout. 操作系统使用一定的策略来决定哪些程序将使用物理内存，哪些程序将被从内存页面中替出； 这些操作对用户程序来说是透明的； 页面空间是由操作系统在磁盘划分上创建的文件，用来存储当前未在使用的用户程序； 通常，页面大小为 4KB 或 8KB; 在Linux系统中，页面大小由内核的头文件: include/asm-/param.h 中的宏 EXEC_PAGESIZE 来定义； 进程使用的页面被替换到磁盘称为换页 或 页面溢出；

3.3.1 Finding memory bottlenecks 寻找内存瓶颈 文章的这部分使用了KDE的一个工具KDE System Guard来做内存检测；

KDE，K桌面环境(Kool Desktop Environment)的缩写。 一种著名的运行于 Linux、Unix 以及FreeBSD 等操作系统上面自由图形工作环境， 整个系统采用的都是 TrollTech 公司所开发的Qt程序库（现在属于Digia公司）。 KDE 和 Gnome 都是 Linux 操作系统上最流行的桌面环境系统。

Start your analysis by listing the applications that are running on the server. Determine how much physical memory and swap each application needs to run. Figure 3-1 on page 83 shows KDE System Guard monitoring memory usage.

图 3.1 KDE System Guard内存监测

The indicators in Table 3-1 can also help you define a problem with memory.

表3-1

Paging and swapping indicators 换页和交换的意义 In Linux, as with all UNIX-based operating systems, there are differences between paging and swapping. Paging moves individual pages to swap space on the disk; swapping is a bigger operation that moves the entire address space of a process to swap space in one operation. 在Linux系统中，分布和交换是不同的； 换页是将独立的页面移到磁盘的交换空间； 交换是更大的操作，它将整个进程的地址空间一次性移到磁盘的交换空间；

Swapping can have one of two causes: 导致交换的原因有两个 . Aprocess enters sleep mode. 进程进入 sleep 模式 This usually happens because the process depends on interactive action and editors, shells, and data entry applications spend most of their time waiting for user input. During this time, they are inactive. 这种情况通常是因为进程需要和编辑器，shell等交互，以及应用程序需要等待用户的数据输入； 在这是，它是非活动的；

. Aprocess behaves poorly. 进程的异常行为 Paging can be a serious performance problem when the amount of free memory pages falls below the minimum amount specified, because the paging mechanism is not able to handle the requests for physical memory pages and the swap mechanism is called to free more pages. This significantly increases I/O to disk and will quickly degrade a server’s performance. 当空闲内存页面过小时，换页会导致严重的性能问题， 因为换页机制不能处理物理内存页面的请求，而交换机制将会调用更多的空闲内存页面。 这会急剧增加磁盘的I/O, 并快速地拉低服务器的性能

If your server is always paging to disk (a high page-out rate), consider adding more memory. However, for systems with a low page-out rate, it might not affect performance. 如果服务器总是在换页(一个很高的换页率)，就需要考虑增加内存； 当然，系统换页率低时，它不会影响性能；

3.3.2 Performance tuning options 性能调整选项 It you believe there is a memory bottleneck, consider performing one or more of these actions: 如果你确认是内存的瓶颈，可以考虑用下面的办法来解决

. Tune the swap space using bigpages, hugetlb, shared memory. 调整交换空间，使用大页面，大块内存，共享内存

. Increase or decrease the size of pages. 增加或降低页面的大小；

. Improve the handling of active and inactive memory. 改进对活动和非活动内存的处理；

. Adjust the page-out rate. 调整换页率；

. Limit the resources used for each user on the server. 限制服务器上每个用户的可用资源；

. Stop the services that are not needed, as discussed in “Daemons” on page 97. 停掉不需要的服务

. Add memory. 增加内存；

3.4 Disk bottlenecks 磁盘瓶颈

The disk subsystem is often the most important aspect of server performance and is usually the most common bottleneck. However, problems can be hidden by other factors, such as lack of memory. Applications are considered to be I/O-bound when CPU cycles are wasted simply waiting for I/O tasks to finish. 磁盘子系统是服务器性能最重要的方面，通常它也是最常见的瓶颈； 然而，它的问题通常隐藏在其它因素之下，如缺少内存； 当CPU周期都浪费在等待I/O任务完成时，这样的应用程序被认为是I/O密集型；

The most common disk bottleneck is having too few disks. Most disk configurations are based on capacity requirements, not performance. The least expensive solution is to purchase the smallest number of the largest capacity disks possible. However, this places more user data on each disk, causing greater I/O rates to the physical disk and allowing disk bottlenecks to occur. 最常见的磁盘瓶颈是硬盘太小； 大多数的磁盘配置是基于容量需求，而不是性能； 最便宜的解决方案可能是少量的大容量磁盘；

The second most common problem is having too many logical disks on the same array. This increases seek time and significantly lowers performance. The disk subsystem is discussed in 4.6, “Tuning the disk subsystem” on page 112. 第二个常见的问题是在同一个陈列中有过多的逻辑磁盘， 这会增加寻道时间而显著降低性能；

3.4.1 Finding disk bottlenecks 寻找磁盘瓶颈 Aserver exhibiting the following symptoms might be suffering from a disk bottleneck (or a hidden memory problem): 如果服务器出现了下列现象就有可能是遇到了磁盘瓶颈(或隐含的内存问题): . Slow disks will result in 缓慢的磁盘会导致: – Memory buffers filling with write data (or waiting for read data), which will delay all requests because free memory buffers are unavailable for write requests (or the response is waiting for read data in the disk queue). 内存buffer将会被写数据填充(或等待读数据)，这会导致所有的响应变慢， 因为对于写请求来说，没有可用的空间内存buffer(或这个响应在等待从磁盘队列中读数据)。

– Insufficient memory, as in the case of not enough memory buffers for network requests, will cause synchronous disk I/O. 内存不足，在没有足够的内存缓冲用于网络请求的情况下，将导致磁盘I/O 同步；

. Disk utilization, controller utilization, or both will typically be very high. 磁盘利用率，控制器利用率，或者这两者都很有可能；

. Most LAN transfers will happen only after disk I/O has completed, causing very long response times and low network utilization. 大多数的的LAN传输都会在磁盘I/O完成后会发生，会造成非常长的响应时间和很低的网络选用率；

. Disk I/O can take a relatively long time and disk queues will become full, so the CPUs will be idle or have low utilization because they wait long periods of time before processing the next request. 磁盘I/O会消耗很长的时间并导致磁盘队列变满，因此CPU将会变得很闲或很低的利用率， 因为它们在处理下个响应之前，要等待很长的时间；

The disk subsystem is perhaps the most challenging subsystem to properly configure. Besides looking at raw disk interface speed and disk capacity, it is also important to understand the workload. Is disk access random or sequential? Is there large I/O or small I/O? Answering these questions provides the necessary information to make sure the disk subsystem is adequately tuned. 磁盘子系统可能是实现合适配置的最有挑战的子系统； 除了要关注原始磁盘接口速度和硬盘容量外，工作负载也同样重要； 磁盘访问是随机的还是序列的？是重I/O，还是轻I/O的？ 只有回答了这些问题后才能提供必要的信息来调整磁盘子系统；

Disk manufacturers tend to showcase the upper limits of their drive technology’s throughput. However, taking the time to understand the throughput of your workload will help you understand what true expectations to have of your underlying disk subsystem. 磁盘制造商倾向于炫耀他们驱动技术的吞吐量的上限， 然而，理解你工作负载的吞吐量将有助于理解你对磁盘子系统的真正期望；

Random read/write workloads usually require several disks to scale. The bus bandwidths of SCSI or Fibre Channel are of lesser concern. Larger databases with random access workload will benefit from having more disks. Larger SMP servers will scale better with more disks. Given the I/O profile of 70% reads and 30% writes of the average commercial workload, a RAID-10 implementation will perform 50% to 60% better than a RAID-5. 随机读/写工作负载通常请求多个磁盘扩展； SCSI的bus的带宽或Fibre通常不怎么关注； 随机访问方式的大数据库在多个磁盘情况下很有好处； 大型SMP服务器

在扩展上要好于多磁盘； 假如平均工作负载 是70%的读和30%的写，则RAID-10要比RAID-5效率高50%~60%；

Sequential workloads tend to stress the bus bandwidth of disk subsystems. Pay special attention to the number of SCSI buses and Fibre Channel controllers when maximum throughput is desired. Given the same number of drives in an array, RAID-10, RAID-0, and RAID-5 all have similar streaming read and write throughput. 序列工作负载对bus带宽和磁盘子系统有很大的压力； 假如在陈列中有相同数量的磁盘，RAID-10, RAID-0, RAID-5有相同的流读写吞吐量；

There are two ways to approach disk bottleneck analysis: real-time monitoring and tracing. 有两种方法来对磁盘瓶颈进行分析: 实时监测和跟踪； . Real-time monitoring must be done while the problem is occurring. This might not be practical in cases where system workload is dynamic and the problem is not repeatable. However, if the problem is repeatable, this method is flexible because of the ability to add objects and counters as the problem becomes clear. 实时监测必须在问题出现时做，这在系统负载是动态的且问题不能重现时很难操作， 然而，如果问题可以重现，这个方法就很有用了；

. Tracing is the collecting of performance data over time to diagnose a problem. This is a good way to perform remote performance analysis. Some of the drawbacks include the potential for having to analyze large files when performance problems are not repeatable, and the potential for not having all key objects and parameters in the trace and having to wait for the next time the problem occurs for the additional data. 跟踪是通过长时间收集性能数据来诊断问题，对于远程性能分析来说，这是一个很好的方法； 但这个方法也有一些缺点，分析大文件时，性能问题不可重复； 未对重要对象和参数进行跟踪等；

vmstat command One way to track disk usage on a Linux system is by using the vmstat tool. The important columns in vmstat with respect to I/O are the bi and bo fields. These fields monitor the movement of blocks in and out of the disk subsystem. Having a baseline is key to being able to identify any changes over time. Linux系统的硬盘跟踪工具可以使用vmstat, 它是最重要的列是bi和bo, 它们监测了有磁盘子系统的块的输入和输出的移动， 有一个基准线来标识改变很重要；

iostat command Performance problems can be encountered when too many files are opened, read and written to, then closed repeatedly. This could become apparent as seek times (the time it takes to move to the exact track where the data is stored) start to increase. Using the iostat tool, you can monitor the I/O device loading in real time. Different options enable you to drill down even deeper to gather the necessary data. iostats可以用于监测实时的I/O设备负载，不同的选项可做进一步的分析；

Example 3-3 shows a potential I/O bottleneck on the device /dev/sdb1. This output shows average wait times (await) of about 2.7 seconds and service times (svctm) of 270 ms.

For a more detailed explanation of the fields, see the man page for iostat(1). Changes made to the elevator algorithm as described in 4.6.2, “I/O elevator tuning and selection” on page 115 will be seen in avgrq-sz (average size of request) and avgqu-sz (average queue length). As the latencies are lowered by manipulating the elevator settings, avgrq-sz will decrease. You can also monitor the rrqm/s and wrqm/s to see the effect on the number of merged reads and writes that the disk can manage. 更多的细节可见iostat的man页面；

3.4.2 Performance tuning options 性能调整选项 After verifying that the disk subsystem is a system bottleneck, several solutions are possible. 当确认磁盘子系统是系统瓶颈时，有多个解决办法可用；

These solutions include the following: . If the workload is of a sequential nature and it is stressing the controller bandwidth, the solution is to add a faster disk controller. However, if the workload is more random in nature, then the bottleneck is likely to involve the disk drives, and adding more drives will improve performance. 如果工作负载是序列式的且控制器的带宽压力很大时，解决办法是添加更快速的磁盘控制器； 如果工作负载是随机式的时，瓶颈可能是硬盘驱动，增加驱动可以改善性能；

. Add more disk drives in a RAID environment. This spreads the data across multiple physical disks and improves performance for both reads and writes. This will increase the number of I/Os per second. Also, use hardware RAID instead of the software implementation provided by Linux. If hardware RAID is being used, the RAID level is hidden from the OS. 在RAID环境中添加更多的磁盘驱动，这能使数据跨越多个物理磁盘并改善读写性能； 同样，使用硬件RAID替代软件RAID也有效果；

. Consider using Linux logical volumes with striping instead of large single disks or logical volumes without striping. 考虑使用Linux条带方式的逻辑卷；

. Offload processing to another system in the network (users, applications, or services). 卸载进程到网络的另一个系统；

. Add more RAM. Adding memory increases system memory disk cache, which in effect improves disk response times. 增加更多的RAM，添加内存增加了系统的内存磁盘cache, 可以改善磁盘的响应时间；

3.5 Network bottlenecks Aperformance problem in the network subsystem can be the cause of many problems, such as a kernel panic. To analyze these anomalies to detect network bottlenecks, each Linux distribution includes traffic analyzers. 网络子系统的性能问题可能的原因有很多，如内核的小问题； linux的分析工具也很多；

3.5.1 Finding network bottlenecks We recommend KDE System Guard because of its graphical interface and ease of use. The tool, which is available on the distribution CDs, is discussed in detail in 2.3.17, “KDE System Guard” on page 62. Figure 3-2 on page 88 shows it in action. 可以使用KDE System Guard进行图形化分析 ；

图3-2 KDE system Guard 网络监测

It is important to remember that there are many possible reasons for these performance problems and that sometimes problems occur simultaneously, making it even more difficult to pinpoint the origin. The indicators in Table 3-3 can help you determine the problem with your network.

Table 3-3

3.5.2 Performance tuning options 性能调整选项

These steps illustrate what you should do to solve problems related to network bottlenecks: 下面列出解决网络瓶颈的的可选项：

. Ensure that the network card configuration matches router and switch configurations (for example, frame size). 确认网点的配置是否匹配；

. Modify how your subnets are organized. 修改子网

. Use faster network cards. 使用高速网卡；

. Tune the appropriate IPV4 TCP kernel parameters. (See Chapter 4, “Tuning the operating system” on page 91.) Some security-related parameters can also improve performance, as described in that chapter. 调整IPV4的TCP内核参数；

. If possible, change network cards and recheck performance. 如果可能，改变网卡和重检查性能；

. Add network cards and bind them together to form an adapter team, if possible. 绑定网卡成一个自适应组；

4.1 Tuning principles

Tuning any system should follow some simple principles of which the most important is change management as described below. Generally the first step in systems tuning should be to analyze and evaluate the current system configuration. Ensuring that the system performs as stated by the hardware manufacturer and that all devices are running in their optimal mode will create a solid base for any later tuning. Also prior to any specific tuning tasks a system designed for optimal performance should have a minimum of unnecessary tasks and subsystems running. Finally when moving towards specific systems tuning, it should be noted that tuning often tailors a system towards a specific workload. So, the system will perform better under the intended load characteristics but it will probably perform worse for different workload patterns. An example would be tuning a system for low latency which most of the time has an adverse effect on throughput.

4.1.1 Change management

While not strictly related to performance tuning, change management is probably the single most important factor for successful performance tuning. The following considerations might be second nature to you, but as a reminder we highlight these points: Implement a proper change management process before tuning any Linux system. Never start tweaking settings on a production system. Never change more than one variable during the tuning process. Retest parameters that supposedly improved performance; sometimes statistics come into play.

Document successful parameters and share them with the community no matter how trivial you think they are. Linux performance can benefit greatly from any results obtained in production environments.

Tuning the operating system

Linux distributions and the Linux kernel offer a variety of parameters and settings to let the Linux administrator tweak the system to maximize performance. As stated earlier in this paper, there is no magic tuning knob that will improve systems performance for any application. The settings discussed in the following chapter will improve performance for certain hardware configurations and application layouts. The settings that improve performance for a Web server scenario might have adverse impacts on the performance of a database system.

This chapter describes the steps you can take to tune kernel 2.6 based Linux distributions. Since the current kernel 2.6 based distributions vary from kernel release 2.6.9 up to 2.6.19 (at the time of this paper) some tuning options might only apply to a specific kernel release. The objective is to describe the parameters that give you the most improvement in performance and offer a basic understanding of the techniques that are used in Linux, including:

Linux memory management

System clean up

Disk subsystem tuning

Kernel tuning using sysctl

Network optimization

This chapter has the following sections:

4.1, “Tuning principles” 4.2, “Installation considerations” 4.3, “Changing kernel parameters” 4.4, “Tuning the processor subsystem” 4.5, “Tuning the vm subsystem” 4.6, “Tuning the disk subsystem” 4.7, “Tuning the network subsystem”