

LPI 301 考试准备，主题 306: 容量规划

Senior Level Linux Professional (LPIC-3) 考试学习指南

Sean Walberg

网络工程师

自由职业

2008 年 5 月 26 日

在本教程中，Sean Walberg 帮助您准备 Linux Professional Institute Senior Level Linux® Professional (LPIC-3) 考试。这是 [共有 6 个教程的系列教程](#) 的最后一个，Sean 介绍如何监视系统资源、解决资源问题和分析系统容量。

[查看本系列更多内容](#)

开始之前

本节解释这些教程讲授什么内容，以及如何从这些教程获得最大的收益。

关于本系列

[Linux Professional Institute](#) (LPI) 对 Linux 系统管理员的认证分为三级：初级（也称为“认证级别 1”）、中级（也称为“认证级别 2”）和高级（也称为“认证级别 3”）。要获得认证级别 1，您必须通过 101 和 102 考试；要获得认证级别 2，您必须通过 201 和 202 考试。要获得认证级别 3，您必须已经获得中级认证，并通过 301 考试（“core”）。在高级认证中，可能还要通过其他专业考试。

developerWorks 提供教程来帮助您准备初级、中级和高级认证的 5 门考试。每门考试包含几个主题，每个主题在 developerWorks 上都有一个对应的自学教程。表 1 列出 LPI 301 考试的 6 个主题和对应的 developerWorks 教程。

表 1. LPI 301 考试：教程和主题

LPI 301 考试主题	developerWorks 教程	教程摘要
主题 301	LPI 301 考试准备：概念、体系结构和设计	了解 LDAP 的概念和体系结构，了解如何设计和实现 LDAP 目录，并了解模式。
主题 302	LPI 301 考试准备：安装和开发	了解如何安装、配置和使用 OpenLDAP 软件。
主题 303	LPI 301 考试准备：配置	学习如何详细配置 OpenLDAP 软件。
主题 304	LPI 301 考试准备：使用指南	学习如何搜索目录和使用 OpenLDAP 工具。

主题 305	LPI 301 考试准备：集成和迁移	了解如何使用 LDAP 作为系统和应用程序的数据源。
主题 306	LPI 301 考试准备：容量计划	(本教程) 度量资源、解决资源问题以及为未来的增长制定计划。参见下面详细的 目标 。

要想通过考试 301 (并获得认证级别 3)，您应该：

- 具备几年在许多计算机上为各种用途安装和维护 Linux 的经验。
- 具备使用各种技术和操作系统的集成经验。
- 具备企业级 Linux 专业人员的经验 (或者受过相关培训)，包括作为其他角色从事企业级 Linux 管理的经验。
- 了解高级和企业级 Linux 管理，包括安装、管理、安全、故障排除和维护。
- 能够使用开放源码工具评估容量计划和解决资源问题。
- 具备使用 LDAP 集成 UNIX® 服务和 Microsoft® Windows® 服务的专业经验，包括 Samba、Pluggable Authentication Modules (PAM)、电子邮件和 Microsoft Active Directory 目录服务。
- 能够计划、设计、构建和实现一个使用 Samba 和 LDAP 的完整环境，而且能够评估容量计划和服務的安全性。
- 能够创建 Bash 或 Perl 脚本，或者至少了解一种系统编程语言 (比如 C)。

要想继续准备认证级别 3，请阅读 [LPI 301 考试系列中的所有 developerWorks 教程](#) 以及 [所有 developerWorks LPI 教程](#)。

Linux Professional Institute 不为任何第三方考试准备资料或技术做担保。

关于本教程

欢迎阅读“容量规划”，这是针对 LPI 301 考试而设计的 6 篇教程中的最后一篇。在本教程中，学习如何度量 UNIX 资源、分析需求以及预测未来的资源需求。

本教程是按照这个主题的 LPI 目标组织的。大致来说，权值越高的学习目标，在考试中出的题就越多。

目标

表 2 给出本教程的详细目标。

表 2. 容量规划：本教程中涉及的考试目标

LPI 考试目标	目标权值	目标摘要
306.1 度量资源使用量	4	度量硬件和网络使用量。
306.2 解决资源问题	4	识别和解决资源问题。
306.3 分析需求	2	识别软件的容量需求。
306.4 预测未来的资源需求	1	分析使用量的变化趋势，预测应用程序什么时候将需要更多的资源，为未来制定计划。

前提条件

要想从本教程获得最大的收益，您应该具备 Linux 的高级知识并且拥有一个可以用来实践的 Linux 系统。

如果您的基本 Linux 技能有点儿生疏了，可以先复习 [针对 LPIC-1 和 LPIC-2 考试的教程](#)。

不同的程序版本可能会导致不同格式的输出，所以您在进行实践时获得的结果可能会与本教程中的清单和图不完全一样。

系统需求

为了学习这些教程中的示例，需要一个带 OpenLDAP 包并支持 PAM 的 Linux 工作站。大多数现代发行版都能够满足这些需求。

度量资源使用量

本节讨论 Senior Level Linux Professional (LPIC-3) 考试 301 的 306.1 主题的内容。这个主题的权值为 4。

在本节中，学习如何：

- 度量 CPU 使用量
- 度量内存使用量
- 度量磁盘 I/O
- 度量网络 I/O
- 度量防火墙和路由吞吐量
- 映射客户机带宽使用量

计算机依赖于硬件资源：中央处理器（CPU）、内存、磁盘和网络。通过度量这些资源，可以了解计算机目前的工作状态以及在哪些方面可能出现的问题。观察这些度量值在一段时间（比如几个月）内的变化，就可以获得一些有意义的历史信息。常常可以推断出这些指标未来的数值，这有助于预测资源会在什么时候耗尽。还可以开发系统的数学模型，使用历史信息检验这个模型，这有助于更精确地预测未来的使用量。

为了完成一个任务，服务器总是需要使用多种硬件资源。一个任务可能需要通过磁盘访问获取数据，访问内存来存储数据，通过 CPU 处理数据。如果一种资源不足，性能就会受损。在从磁盘读取信息之前，CPU 无法处理信息；如果内存满了，就无法存储信息。这些概念是相互关联的。如果内存满了，操作系统会把其他内存的内容交换到磁盘上。还会从内存中划分出缓冲区，缓冲区用来加快磁盘活动的速度。

理解资源

要想让度量发挥作用，就必须理解要度量的资源。然后，就可以收集关于系统的有用信息：当前信息、历史信息或预测信息。

CPU

计算机的 CPU 执行应用程序需要的所有计算，它会向磁盘和其他外设发出命令，并负责运行操作系统内核。CPU 每次只运行一个任务，无论这个任务是运行内核还是应用程序。一种称为中断的硬件

信号可以中断当前任务。中断可以由外部事件触发，比如收到一个网络数据包；也可以由内部事件触发，比如系统时钟（在 Linux 中称为 tick）。当发生中断时，当前运行的进程暂停，并运行一个例程来决定系统下面应该做什么。

当当前运行的进程用完分配给它的时间时，内核可以使用一个称为上下文切换（context switch）的过程切换到另一个进程。在用完分配给进程的时间之前，如果它发出任何 I/O 命令（比如读取磁盘），那么可以提前切换到另一个进程。CPU 的处理速度比磁盘快得多，所以在等待磁盘请求返回数据时，CPU 可以运行其他任务。

在讨论 Linux 系统的 CPU 时，应该关注几个指标。第一个指标是 CPU 的空闲时间与工作时间之间的百分比（实际上，CPU 总是在做某些事情——所以，如果没有任务等待执行，就认为 CPU 是空闲的）。当空闲百分比为零时，CPU 的运行时间就达到最大。CPU 时间的非空闲部分划分为系统时间和用户时间。系统时间是指花费在运行内核上的时间，用户时间是指用来执行用户所请求的工作的时间。空闲时间划分为内核由于无事可做而空闲的时间以及由于等待 I/O 而空闲的时间。

精确地度量这些指标很困难，因为这要求 CPU 花费所有时间来判断它正在做的事情！内核大约每秒检查当前状态（系统时间、用户时间、I/O 等待时间和空闲时间）100 次，并使用这些度量值计算百分比。

Linux 用来衡量 CPU 使用率的另一个指标是平均负载。这个指标并不与 CPU 使用率直接关联；它代表过去一分钟、5 分钟和 15 分钟内内核运行队列中基于指数加权的平均任务数量。后面会进一步研究这个指标。

关于内核要考虑的其他指标有中断负载和上下文切换数。对于这些指标没有上限，但是执行的中断和上下文切换越多，CPU 用来执行用户工作的时间就越少。

内存

系统有两种内存：真实内存和交换空间。真实内存是指主板上安装的 RAM 条。交换空间是当系统试图分配超出物理内存量的 RAM 时使用的临时空间。在这种情况下，RAM 的页面被交换到磁盘上，从而释放出内存空间供当前分配使用。当再次需要使用交换出去的数据时，会把它们交换回来。

RAM 可以由应用程序或系统使用，或者未被使用。系统以两种方式使用 RAM：作为原始磁盘块的缓冲区（输入或输出）和作为文件缓存。缓冲区和缓存的大小是动态的，所以如果需要的话，可以把内存让给应用程序。这就是大多数人觉得他们的 Linux 系统似乎没有空闲内存的原因：系统把未使用的内存分配给缓冲区和缓存了。

交换内存存在磁盘上。大量的交换操作会减慢系统速度，这是系统内存不足的迹象之一。

磁盘

磁盘是长期存储数据的设备，比如硬盘、U 盘或磁带（统称为块设备）。惟一的例外是 RAM 磁盘，这是 RAM 中的一个区域，但是表现为块设备。当系统关闭时，RAM 磁盘上的数据会丢失。最常用的磁盘类型是硬盘，所以本教程对磁盘的讨论主要关注这种介质。

用来描述磁盘的度量值有两种：空间和速度。磁盘上的空闲空间是指磁盘上可供使用的字节数。磁盘上的开销包括由文件系统使用的空间或由于其他原因不可用的空间。请记住，大多数生产商

用 GB 表示磁盘的大小，1 GB 是 1,000,000,000 字节；但是操作系统使用以 2 为底的指数，1 GB 是 1,073,741,824 字节；这会导致磁盘的实际空间只有标称空间的 93%。这个差距并不大，但是如果没有考虑到这一点，对磁盘空间的计算就不正确。

磁盘的第二个指标是速度，也就是磁盘返回数据的速度。当 CPU 发出请求时，必须执行几个步骤，数据才能返回给 CPU：

1. 内核把请求放到一个队列中，请求在队列中等待发送到磁盘（等待时间）。
2. 命令被发送到磁盘控制器。
3. 磁盘把磁头移动到所需的块上（寻道时间）。
4. 磁头从磁盘读取数据。
5. 数据返回给 CPU。

度量这些步骤的方式各不相同，有时候根本不度量某些步骤。服务时间 包括最后三个步骤，这个指标表示在发出请求之后多长时间能够满足请求。等待时间 表示整个过程，包括请求在队列中排队的时间和服务时间。

内核执行的优化措施之一是，在第一步中对队列中的请求进行重新排列和合并，从而尽可能减少磁盘寻道数量。这称为 *elevator*，有多种不同的实现算法。

网络

Linux 在网络方面常常扮演两种角色：客户机（与服务器上的应用程序相互发送数据包）和路由器（或防火墙或桥接器）。在一个接口上接收数据包，在另一个接口上发出数据包（可能先进行某种过滤或检查）。

网络状态常常用每秒比特数（或千比特、兆比特、吉比特）和每秒数据包数来度量。度量每秒数据包数的意义往往不太大，因为计算机针对每个数据包有固定的开销量，所以数据包越小，吞吐量越差。不要误以为网卡的速度（比如 100Mbit/sec 或吉比特）就是机器预期的数据传输速度。有几个外部因素会影响数据传输速度，包括延迟和连接的远程端，还包括服务器上的设置。

队列

队列与其他资源很不一样，但是在监视性能时常常用到队列，因此必须加以考虑。队列 是请求等待处理的地方。内核在许多方面使用队列，比如运行队列（包含要运行的进程的列表）、磁盘队列、网络队列和硬件队列。在一般情况下，队列是一个内存区域，内核使用它跟踪特定的任务集。但也可以是硬件组件上的一段内存，由硬件管理。

队列在两个方面影响性能调优。首先，如果队列中有过多的工作，新的工作就会丢失。例如，如果过多的数据包到达一个网络接口，一些数据包就被丢弃（用网络术语来说，这导致 *tail drop*）。第二，如果频繁使用队列（或使用得不够），另一个组件可能达不到所需的性能。如果运行队列中的进程数量常常很高，就可能意味着 CPU 过载。

度量性能

有几个工具可以用来度量 Linux 系统的性能。其中一些工具直接度量 CPU、磁盘、内存和网络，其他工具显示队列使用量、进程创建量和错误数等指标。一些工具显示即时值，一些工具显示一段时间的平均值。理解度量的方式与理解度量的目标同样重要。

vmstat

`vmstat` 是一种很有帮助的工具，可以实时地显示最常用的性能指标。关于 `vmstat` 应该注意的最重要的一点是：它首先显示自系统启动以来的平均值，一般应该忽略这些数据。在命令行上指定重复时间（以秒为单位），就可以让 `vmstat` 使用当前数据重复地报告信息。清单 1 给出 `vmstat 5` 的输出。

清单 1. `vmstat 5` 的输出

```
# vmstat 5
procs -----memory----- ---swap-- ----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo    in   cs us sy id wa st
 0  3   17780  10304  18108 586076    0    0   2779   332    1    1  3  4 76 17  0
 1  2   17780  10088  19796 556172    0    0   7803  3940  2257 4093 25 28 14 34  0
 0  2   17780   9568  19848 577496    0    0  18060  1217  1610  910  0  3 48 49  0
 0  0   17780  51696  20804 582396    0    0   9237  3490  1736  839  0  3 55 41  0
```

清单 1 显示每 5 秒度量一次的 `vmstat` 命令的输出。第一行的值是自系统启动以来的平均值，所以应该忽略它。前两列是进程数。`r` 下面的数值是进行度量时运行队列中的进程数。运行队列中的进程正在等待 CPU。下一列是因 I/O 操作阻塞的进程数，这意味着在 I/O 操作返回数据之前这些进程处于睡眠状态，而且不能中断它们。

`memory` 下面的列是关于系统内存的即时度量值，数据以 KB（1024 字节）为单位。`swpd` 是已经交换到磁盘的内存量。`free` 是应用程序、缓冲区和缓存都没有使用的空闲内存量。如果这个数值很低，也不要吃惊（关于实际的空闲内存的更多信息，请参见后面对 [free](#) 的讨论）。`buff` 和 `cache` 分别表示分配给缓冲区和缓存的内存量。缓冲区存储原始磁盘块，缓存存储文件。

前两类度量值是即时度量值。有可能出现这样的情况：在一段短时间内，所有空闲内存都被消耗了，但是在下一次度量之前释放了。因此，在输出中没有反映出来。其他值是取样时间段内的平均值。

`swap` 是每秒从磁盘交换回（`si`）和交换到磁盘（`so`）的平均内存量；数据以 KB 为单位。`io` 是每秒对所有块设备读和写的磁盘块数量。

`system` 类别中的度量值是每秒中断数（`in`）和每秒上下文切换数（`cs`）。中断来自设备（比如网卡通知内核一个数据包正在等待处理）和系统计时器。在某些内核中，系统计时器每秒触发 1,000 次，所以这个数值可能很高。

最后一类度量值显示 CPU 的状态，数据显示为总 CPU 时间的百分比。这 5 个值的总和应该是 100。`us` 是在取样时间段内 CPU 花在用户任务上的平均时间，`sy` 是 CPU 花在系统任务上的平均时间。`id` 是 CPU 的空闲时间，`wa` 是 CPU 等待 I/O 的时间。（清单 1 中的数据来自一个出现严重 I/O 瓶颈的系统，可以看到 34-49% 的 CPU 时间花在等待从磁盘返回数据上。）最后一个值 `st`（steal 时间）适用于运行系统管理程序（hypervisor）和虚拟机的服务器。它是指系统管理程序在本来应该运行虚拟机的时间内执行其他任务的时间百分比。

在 [清单 1](#) 中可以看出，`vmstat` 提供了关于各种性能指标的大量信息。如果系统发生了什么问题，`vmstat` 是寻找问题原因的好工具。

`vmstat` 还可以显示关于每个磁盘设备的使用情况的信息，这可以进一步细化清单 1 中的 swap 和 io 度量值。`-d` 参数报告一些磁盘统计数据，包括每个磁盘的读写总数。清单 2 给出 `vmstat -d 5` 的部分输出（去掉了未使用的设备）。

清单 2. 使用 `vmstat` 显示磁盘使用情况

```
disk- -----reads----- -----writes----- -----IO-----
      total merged sectors      ms total merged sectors      ms      cur      sec
hda   186212  28646 3721794   737428 246503 4549745 38981340 8456728      0   2583
hdd   181471  27062 3582080   789856 246450 4549829 38981624 8855516      0   2652
```

每个磁盘的信息显示在单独的一行上，而且输出分为读和写。读和写进一步分为发出的请求总数、在磁盘 elevator 中合并的请求数、读或写的扇区数和总服务时间。这些数值都是累积值，所以在下一次重新引导之前会一直增加，而在不使用 `-d` 选项时显示的是平均值。

清单 2 中的最后一组度量值（在 IO 下面）显示磁盘当前的 I/O 操作数和自启动以来在 I/O 上花费的总秒数。

在清单 2 中，两个磁盘的读操作数据很相似，写操作数据几乎相同。这两个磁盘组成一个软件镜像，所以这种表现是正常的。还可以使用清单 2 中的信息识别缓慢的磁盘或者使用量比较大的磁盘。

iostat

与清单 2 中的 `vmstat -d` 示例密切相关的是 `iostat`。这个命令提供每个磁盘设备的使用情况细节。与 `vmstat -d` 相比，`iostat` 能够提供更多的细节。与 `vmstat` 一样，可以向 `iostat` 传递一个表示刷新时间间隔的数值。同样，它首先输出自系统启动以来的值，因此常常忽略这些数据。清单 3 给出 `iostat` 的输出，时间间隔为 5 秒。

清单 3. `iostat` 命令的输出

```
$ iostat 5
Linux 2.6.20-1.3002.fc6xen (bob.ertw.com)      02/08/2008

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.85    0.13   0.35   0.75    0.01   97.90

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
hda                 1.86      15.24      13351      4740568    41539964
hdd                 1.85      14.69      133.51      4570088    41540256
```

在每个度量间隔的输出中，第一部分显示 CPU 使用情况，`vmstat` 也显示这些信息。但是，这里显示两位小数形式。输出的第二部分显示系统上的所有块设备（要想限制显示的设备数量，可以在命令行上传递设备的名称，比如 `iostat 5 hda sda`）。第一列 `tps` 表示在 elevator 合并请求之后的每秒传输数。没有指定传输的大小。后面 4 列以 512 字节的块为单位，分别表示每秒读的块数、每秒写的块数、读的总块数和写的总块数。如果希望看到以 KB 或 MB 为单位的值，应该分别指定 `-k` 或 `-m` 选项。如果需要的话，可以通过指定 `-p` 选项显示分区级的详细数据。

可以使用 `-x` 参数获得更详细的信息，见清单 4。清单 4 还把输出限制为一个驱动器。为了适应页面的宽度，调整了输出的格式。

清单 4. `iostat` 的部分输出

```
# iostat -x 5 hda
..... CPU information removed ...
Device:      rrqm/s   wrqm/s   r/s     w/s    rsec/s   wsec/s  avgrq-sz
hda          16669.31    1.49    756.93   1.49   139287.13    27.72   18369
              avgqu-sz   await   svctm    %util
              1.58      208    1.28    96.83
```

前六个值是每秒读和写的数量。`rrqm/s` 和 `wrqm/s` 是合并的读写请求数。`r/s` 和 `w/s` 表示发送到磁盘的读写数。因此，合并的磁盘请求百分比是 $16669 / (16669 + 757) = 95\%$ 。`rsec/s` 和 `wsec/s` 显示读写速率（每秒扇区数）。

后面四列显示磁盘队列和时间的相关信息。`avgrq-sz` 是发送到设备的平均请求大小（扇区数）。`avgqu-sz` 是在度量间隔内磁盘队列的平均长度。`await` 是平均等待时间（毫秒数），这表示从请求发送给内核到返回的平均时间。`svctm` 是平均服务时间（毫秒数），这是请求离开队列并发送给磁盘到返回的时间。

最后一个值 `%util` 是系统在这个设备上执行 I/O 的时间百分比，这也称为饱和度（saturation）。清单 4 报告的 96.83% 表明在这段时间内这个磁盘几乎达到了容量极限。

`mpstat`

`mpstat` 报告关于 CPU（或多处理器机器上的所有 CPU）的详细信息。其中许多信息也可以通过某种形式的 `iostat` 和 `vmstat` 命令获得，但是 `mpstat` 为所有处理器单独提供数据。清单 5 显示 `mpstat` 的输出，度量时间间隔为 5 秒。与 `iostat` 和 `vmstat` 不同，不应该忽略第一行数据。

清单 5. 用 `mpstat` 显示 CPU 信息

```
# mpstat -P 0 5
Linux 2.620-1.3002.fc6xen (bob.ertw.com)      02/09/2008

09:45:23 PM  CPU    %user   %nice    %sys %iowait  %irq   %soft  %steal   %idle  intr/s
09:45:25 PM    0    77.61   21.89    0.00   0.00   0.50   0.00   0.00   0.00   155.22
09:45:27 PM    0    68.16   30.85    1.00   0.00   0.00   0.00   0.00   0.00   154.73
```

指定 `-P 0` 表示应该显示第一个 CPU（编号从 0 开始）。还可以指定 `-P ALL`，从而单独显示所有 CPU 的数据。`mpstat` 返回的字段如下：

- `%user`：在用户任务上花费的时间的百分比，但是不包括 `nice` 任务
- `%nice`：在 `nice`（低优先级）用户任务上花费的时间的百分比
- `%sys`：在内核任务上花费的时间的百分比
- `%iowait`：等待 I/O 的空闲时间的百分比
- `%irq`：处理硬件中断的时间的百分比
- `%soft`：处理软件中断的时间的百分比
- `%steal`：系统管理程序从虚拟机偷取的时间的百分比
- `intr/s`：每秒的平均中断数

`ps tree`

在考察资源使用情况时，了解哪些进程生成了其他进程会很有帮助。了解这些情况的一种方法是使用 `ps -ef` 的输出并使用父进程的 id，直至找到 PID 1（`init`）。还可以使用 `ps -efjH`，这个命令会把输出排序为父-子树，并包含 CPU 时间使用数据。

`ps-tree` 实用程序以更图形化的格式显示进程树，还把相同进程的多个实例合并为一行。清单 6 显示 `ps-tree` 的输出（传递 Postfix 守护进程的 PID）。

清单 6. `ps-tree` 的输出

```
[root@sergeant ~]# ps-tree 7988
master--anvil
        |--cleanup
        |--local
        |--pickup
        |--proxymap
        |--qmgr
        |--2*[smtpd]
        |--2*[trivial-rewrite]
```

master 进程生成了其他几个进程，比如 `anvil`、`cleanup` 和 `local`。最后两行输出的格式是 `N*[something]`，其中的 `something` 是进程的名称，`N` 是采用这个名称的子进程的数量。如果除了方括号（`[]`）之外 `something` 还包围在花括号（`{}`）中，`N` 就表示运行的线程数（`ps` 在一般情况下不显示线程数，除非使用 `-L` 选项）。

w、uptime 和 top

这些实用程序是研究系统问题时首先使用的实用程序。清单 7 给出 `w` 命令的输出。

清单 7. `w` 命令的输出

```
# w
12:14:15 up 33 days, 15:09, 2 users, load average: 0.06, 0.12, 0.09
USER      TTY      FROM          LOGIN@      IDLE   JCPU   PCPU WHAT
root      tty2      -             17Jan08 18days 0.29s   0.04s login -- root
root      pts/0    bob           Sat22       0.00s   0.57s   0.56s -bash
```

`w` 输出的第一行提供大量信息。第一部分“12:14:15 up 33 days, 15:09”是当前时间和运行时间（33 天 15 小时 9 分钟）。第二部分“2 users”是已经登录的用户数。最后一部分是平均负载，包括 1 分钟、5 分钟和 15 分钟的平均值。

平均负载是给定的时间段内运行队列中进程数的加权平均值。平均负载越高，就表示有更多的进程在争夺 CPU。平均负载并未按照 CPU 数量进行归一化，这意味着平均负载与 CPU 数量不相关。

要想理解平均负载，就必须理解它的加权方式。平均负载每 5 秒更新一次，越陈旧的信息在计算中的作用越小。如果运行队列中的进程数从 0 提高到 1，那么下一分钟的 1 分钟平均负载并不是一条直线，而是一条曲线，这条曲线先快速上升，然后在 60 秒标志之前逐渐下降。关于平均负载计算方法的详细信息，参见 [参考资料](#)。

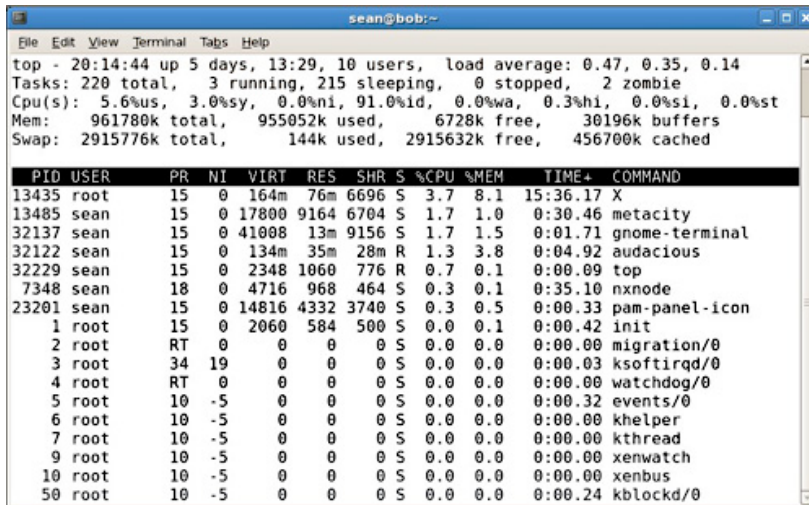
对平均负载进行加权计算的目的是，避免度量期间实际负载的变化影响数据的稳定性；但是数值更多地反映当前状态，尤其是 1 分钟平均值。

第一行后面是已经登录的用户的列表，数据包括他们的登录时间、位置和 CPU 使用信息。第一个用户 `root` 是从 `tty2`（一个本地控制台）登录的，他已经空闲了 18 天。第二个用户也是 `root`，但是他是通过网络登录的，当前在 shell 中。`JCPU` 和 `PCPU` 列表示这个用户已经使用的 CPU 时间；第一列包含过去的作业，而 `PCPU` 是用户当前使用的进程的 CPU 时间。

uptime 的输出与 w 输出的第一行完全相同，但是没有关于用户的信息。在实践中，w 更有帮助，因为它提供关于用户的信息，而且这个命令更简短。

另一个常用的命令是 top，它显示不断更新的消耗资源最多的进程列表（按照内存或 CPU 使用量排序），还显示其他一些性能指标。图 1 显示 top 的屏幕图。

图 1. top



```
top - 20:14:44 up 5 days, 13:29, 10 users, load average: 0.47, 0.35, 0.14
Tasks: 220 total, 3 running, 215 sleeping, 0 stopped, 2 zombie
Cpu(s): 5.6%us, 3.0%sy, 0.0%ni, 91.0%id, 0.0%wa, 0.3%hi, 0.0%si, 0.0%st
Mem: 961780k total, 955052k used, 6728k free, 30196k buffers
Swap: 2915776k total, 144k used, 2915632k free, 456700k cached

  PID USER   PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 13435 root    15   0   164m   76m 6696  S   3.7   8.1   15:36.17 X
 13485 sean    15   0   17800  9164 6704  S   1.7   1.0   0:30.46 metacity
 32137 sean    15   0   41008   13m 9156  S   1.7   1.5   0:01.71 gnome-terminal
 32122 sean    15   0   134m   35m 28m   R   1.3   3.8   0:04.92 audacious
 32229 sean    15   0   2348   1060 776   R   0.7   0.1   0:00.09 top
 7348  sean    18   0   4716   968 464   S   0.3   0.1   0:35.10 nxnode
23201  sean    15   0   14816  4332 3740  S   0.3   0.5   0:00.33 pam-panel-icon
   1  root    15   0   2060    584 500   S   0.0   0.1   0:00.42 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   0:00.03 ksoftirqd/0
   4  root    RT   0     0     0   0   S   0.0   0.0   0:00.00 watchdog/0
   5  root    10  -5     0     0   0   S   0.0   0.0   0:00.32 events/0
   6  root    10  -5     0     0   0   S   0.0   0.0   0:00.00 khelper
   7  root    10  -5     0     0   0   S   0.0   0.0   0:00.00 kthread
   9  root    10  -5     0     0   0   S   0.0   0.0   0:00.00 xenwatch
  10  root    10  -5     0     0   0   S   0.0   0.0   0:00.00 xenbus
  50  root    10  -5     0     0   0   S   0.0   0.0   0:00.24 kblockd/0
```

第一行与 uptime 的输出相同，比如运行时间和平均负载。第二行是进程数。Running 是运行队列中的进程数；sleeping 是正在等待唤醒的进程数。stopped 是已经暂停的进程（可能由于正在跟踪或调试这些进程）。zombie 是已经退出的进程，但是父进程还没有确认它们已经死亡。

奇怪的现象

VIRT = RES + SWAP，这意味着 SWAP = VIRT - RES。看一下 PID 13435，可以看到 VIRT 是 164m，RES 是 76m，这意味着 SWAP 必须是 88m。但是，屏幕顶部的交换统计数据表明，只使用了 144K 的交换空间！在 top 中使用 f 键，启用更多字段（比如 swap），就可以证实这一情况。

在交换过程中，不仅仅把页面交换到磁盘。应用程序的二进制代码和库不需要一直放在内存中。内核可以把一些内存页面标为不需要的；但是因为二进制代码已经保存在磁盘上的一个已知位置，所以不需要使用交换文件。这部分内存仍然算作已经交换，因为代码不再常驻在这里了。另外，应用程序可以把内存映射到一个磁盘文件。因为应用程序的总大小（VIRT）包含映射的内存，但是这个部分不是常驻的（RES），它被算作已经交换了。

第三行显示 CPU 使用信息，依次显示用户、系统、niced、空闲、I/O 等待、硬件中断、软件中断和 steal 时间。这些数值是前一个度量间隔（默认为 3 秒）中的时间百分比。

最后两行显示内存统计数据。首先是关于真实内存的信息；在图 1 中，可以看到这个系统有 961,780K 的 RAM（已经扣除了内核占用的部分）。除了 6,728K 之外所有内存都被使用了，其中缓冲区大约为 30MB，缓存为 456M（缓存显示在第二行末尾）。第二行显示交换空间的情况：这个系统有差不多 3G 的交换空间，其中只使用了 144K。

屏幕上其余的数据是当前运行的进程的信息。top 会显示尽可能多的进程，直至填满窗口。每个进程单独占一行，这个列表会在每次度量时更新，把使用 CPU 时间最多的任务列在最上面。显示的列如下：

- PID：进程的进程 id
- USER：进程的有效用户名（如果程序使用 `setuid(2)` 改变用户，就会显示新用户）
- PR：任务的优先级，内核使用优先级决定哪个进程应该首先使用 CPU
- NI：任务的 nice 级别，系统管理员通过设置 nice 级别影响哪些进程应该首先使用 CPU
- VIRT：进程的虚拟映像的大小，这是使用的 RAM 空间（常驻大小）和交换空间中的数据量（交换大小）的总和
- RES：进程的常驻大小，也就是进程使用的真实 RAM 数量
- SHR：应用程序共享的内存量，比如 SysV 共享内存或动态库（*.so）
- S：状态，比如 sleeping、running 或 zombie
- %CPU：前一个度量时间段内使用的 CPU 百分比
- %MEM：前一次度量时使用的 RAM 的百分比（不包括交换空间）
- TIME+：进程使用的时间，格式为 minutes:seconds:hundredths
- COMMAND：正在运行的命令的名称

可以通过 `top` 快速了解哪些进程使用的 CPU 时间最多，还可以全面了解系统的 CPU 和内存使用情况。在 `top` 中按 `M`，就可以让 `top` 按内存使用量排序。

free

看过 `top` 的输出之后，就应该讨论 `free` 了。清单 8 显示 `free` 的输出，这里使用 `-m` 选项以 MB 为单位报告所有值。

清单 8. 使用 `free` 命令

```
# free -m
              total        used        free      shared    buffers     cached
Mem:           939          904           34           0         107         310
-/+ buffers/cache:        486        452
Swap:          2847           0        2847
```

`free` 显示几方面的内存使用信息。第一行显示与 `top` 相同的信息。第二行显示在不考虑缓冲区和缓存的情况下已使用的内存和空闲内存。在清单 8 中，有 452M 的内存是空闲的，可供应用程序使用；这部分内存包括空闲内存（34M）、缓冲区（107M）和缓存（310M）。

最后一行显示与 `top` 相同的交换统计数据。

显示网络统计数据

获取网络统计数据的过程不如获取 CPU、内存和磁盘统计数据那么直接。主要方法是读取 `/proc/net/dev` 中的数据。这个文件记录每个接口的网络传输量，采用数据包数和字节数两种统计方法。如果希望看到传输速度，就必须自己计算：两个度量值之间的差除以度量间隔。也可以使用 `bwm` 等工具自动收集和显示总带宽。图 2 显示 `bwm` 的输出。

图 2. 使用 **bwm** 命令

```

bwm-ng v0.5 (probing every 0.500s), press 'h' for help
input: libstatgrab type: rate
-      iface      Rx      Tx      Total
=====
lo:      0.00 KB/s    0.00 KB/s    0.00 KB/s
peth0:   20.02 KB/s    1.00 KB/s   21.02 KB/s
virbr0:   0.00 KB/s    0.00 KB/s    0.00 KB/s
vif0.0:   1.00 KB/s   20.02 KB/s   21.02 KB/s
eth0:    20.02 KB/s    1.00 KB/s   21.02 KB/s
xenbr0:   0.00 KB/s    0.00 KB/s    0.00 KB/s
-----
total:   41.04 KB/s   22.02 KB/s   63.07 KB/s

```

bwm 显示各方面的接口使用情况。图 2 每半秒更新一次即时速率，还显示 30 秒平均带宽、最大带宽和字节数。在图 2 中可以看到，eth0 以大约 20K/sec 的速率接收通信流，通信流来自 vif0.0。如果不想查看每秒字节数，可以按 u 键在比特数、数据包数和错误数之间切换。

如果需要更详细地了解负责通信流的主机，就需要使用 iftop。它通过与 top 相似的界面提供关于网络通信的信息。内核并不直接提供这些信息，所以 iftop 使用 pcap 库检查网络上传输的数据包，这需要根特权。图 3 显示 iftop 的输出（连接到 eth2 设备）。

图 3. **iftop -i eth2** 的输出

```

195Kb 391Kb 586Kb 781Kb 977Kb
mybox => pub1.kernel.org 23.3Kb 19.0Kb 10.7Kb
mybox <= sbproxy01.voip.les.net 622Kb 518Kb 286Kb
mybox <= scfire-chi-aa05.stream.aol.com 77.3Kb 75.2Kb 75.8Kb
mybox <= ool-4356bfb2.dyn.optonline.net 4.62Kb 5.02Kb 5.27Kb
mybox <= lists.groupstudy.com 124Kb 129Kb 130Kb
mybox <= 501060010b57beae6.cn.shawcable.ne 0b 75.8Kb 18.9Kb
mybox <= qb-in-f17.google.com 0b 9.55Kb 2.39Kb
mybox <= 24.78.140.1 0b 690b 201b
mybox <= 6.33Kb 1.27Kb 324b
mybox <= 6.00Kb 1.20Kb 307b
mybox <= 0b 1.81Kb 911b
mybox <= 0b 354b 209b
mybox <= 0b 583b 146b
mybox <= 0b 506b 127b
255.255.255.255 <= 0b 0b 0b
1.32Kb 826b 816b

TX: cumm: 1.44MB peak: 303Kb rates: 113Kb 183Kb 122Kb
RX: 4.78MB 914Kb 831Kb 739Kb 497Kb
TOTAL: 6.22MB 1.00Mb 943Kb 922Kb 619Kb

```

iftop 显示网络上最活跃的会话。在默认情况下，每个会话有两行输出：一行显示发送信息，另一行显示接收信息。看一下从 mybox 到 pub1.kernel.org 的第一个会话，第一行显示从 mybox 发送出的通信流，第二行显示 mybox 接收的通信流。右边的数值分别表示前 2 秒、10 秒和 40 秒内的平均流量。还可以看到覆盖主机名的黑色条，这是 10 秒平均值的图形化表示（刻度显示在屏幕顶部）。

仔细看一下图 3，第一个传输可能是下载操作，因为接收的流量很大（在前 10 秒内平均速率大约是每秒 0.5 MB），而上传量很小。第二个会话的发送和接收量大致相同，速率稳定在大约 75-78k/sec。这是通过 les.net（我的 VoIP 提供商）的 G.711 语音呼叫。第三个会话显示 128K 的下载速率，上传速率很小：这是一个因特网广播流。

选择连接的接口很重要。图 3 使用防火墙上的一个外部接口，这个接口可以看到通过 IP masquerading 传递之后的所有数据包。这会丢失内部地址。使用另一个接口（比如一个内部接口）可以保留这一信息。

sar

sar 是一个很大的主题（参见 [参考资料](#)）。sar 每 10 分钟对几十个指标进行一次度量，并提供获取度量值的方法。可以使用前面讨论的工具判断目前发生的情况；而 sar 可以显示本周发生的情况。注意，sar 只保留最近 7 天的数据。

配置数据收集过程需要在根的 crontab 中添加两行。清单 9 显示 sar 的典型 crontab 配置。

清单 9. sar 数据收集的根 crontab 配置

```
# Collect measurements at 10-minute intervals
0,10,20,30,40,50 * * * * /usr/lib/sa/sa1 -d 1 1
# Create daily reports and purge old files
0 * * * * /usr/lib/sa/sa2 -A
```

第一行每 10 分钟执行一次 sa1 来收集数据；这个命令通过运行 sadc 进行实际的数据收集。这个作业是自包含的：它知道向哪个文件写入数据，不需要配置。第二行在午夜调用 sa2，这会清空旧的数据文件并把当前的数据收集到一个可读的文本文件中。

在依靠 sar 的数据判断系统状态之前，有必要检查系统运行 sar 的方式。一些系统禁止收集磁盘统计数据；为了纠正这个问题，必须在对 sa1 的调用中添加 -d 选项（清单 9 已经添加了这个选项）。

收集了一些数据之后，现在可以不带任何选项运行 sar，这会显示当天的 CPU 使用情况。清单 10 给出部分输出。

清单 10. sar 的输出示例

```
[root@bob cron.d]# sar | head
Linux 2.6.20-1.3002.fc6xen (bob.ertw.com)      02/11/2008

12:00:01 AM      CPU      %user      %nice      %system      %iowait      %steal      %idle
12:10:01 AM      all       0.18        0.00        0.18        3.67        0.01      95.97
12:20:01 AM      all       0.08        0.00        0.04        0.02        0.01      99.85
12:30:01 AM      all       0.11        0.00        0.03        0.02        0.01      99.82
12:40:01 AM      all       0.12        0.00        0.02        0.02        0.01      99.83
12:50:01 AM      all       0.11        0.00        0.03        0.05        0.01      99.81
01:00:01 AM      all       0.12        0.00        0.02        0.02        0.01      99.83
01:10:01 AM      all       0.11        0.00        0.02        0.03        0.01      99.83
```

您现在应该已经熟悉清单 10 中的数值：它们就是 top、vmstat 和 mpstat 显示的各种 CPU 统计数据。通过使用表 3 中的命令行参数，可以显示更多信息。

表 3. sar 的选项

选项	示例	说明
-A	sar -A	显示所有数据。除非打算把结果保存到一个文本文件中，否则不太可能需要使用这个选项。如果需要这个选项，这个进程会在夜间在执行 sa2 期间运行。
-b	sar -b	显示发送到块设备和从块设备读取的事务数和块数，与 iostat 的输出很相似。

-B	<code>sar -B</code>	显示换页（交换）统计数据，比如 <code>vmstat</code> 报告的那些数据。
-d	<code>sar -d</code>	显示磁盘活动，与 <code>iostat -x</code> 的输出很相似，包括等待时间、服务时间和队列长度。
-n	<code>sar -n DEV # sar -n NFS</code>	在使用 DEV 时，显示接口活动（与 <code>bwm</code> 相似）；在使用 NFS 关键字时，显示 NFS 客户机统计数据（使用 <code>NFSD</code> 关键字获取 NFS 服务器守护进程统计数据）。EDEV 关键字显示来自网卡的错误信息。
-q	<code>sar -q</code>	显示关于运行队列、进程列表总大小和平均负载的信息，比如 <code>vmstat</code> 和 <code>uptime</code> 报告的数据。
-r	<code>sar -r</code>	显示关于内存、交换空间、缓存和缓冲区使用情况的信息（与 <code>free</code> 相似）。
-f	<code>sar -f /var/log/sa/sa11</code>	从另一个文件读取信息。文件根据日期命名。
-s	<code>sar -s 08:59:00</code>	从给定时间之后的第一次度量开始显示信息。如果指定 09:00:00，那么第一次度量发生在 09:10，所以应该在所需的时间中减去 1 分钟。
-e	<code>sar -e 10:01:00</code>	指定显示度量的截止时间。应该在所需的时间中加 1 分钟以确保包含最后一次度量。

还可以组合使用多个参数，从而获得多个报告或另一个带有指定时间段的文件。

df

硬盘是一种有限的资源。如果一个分区的空间用光了，就会出现問題。`df` 命令可以显示磁盘空间状态。清单 11 显示 `df -h` 的输出，这个命令采用一种更友好的输出格式。

清单 11. 用 `df -h` 检查磁盘空间使用情况

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                225G  169G   44G   80% /
/dev/hda1        99M   30M   64M   32% /boot
tmpfs            474M    0   474M    0% /dev/shm
```

清单 11 显示在根上有一个文件系统，大小为 225G，有 44G 的空闲空间。`/boot` 分区是 99M，有 64M 的空闲空间。`tmpfs` 是一个特殊的文件系统，不对应任何设备。如果一个分区满了，就没有可用空间，使用率为 100%。

解决资源问题

本节讨论 Senior Level Linux Professional (LPIC-3) 考试 301 的 306.2 主题的内容。这个主题的权值为 4。

在本节中，学习如何：

- 把系统现象与可能的问题联系起来
- 识别系统中的瓶颈

前一节解释了如何显示 Linux 系统中的各种性能指标。现在要应用这些信息解决系统中与资源相关的问题。

解决问题的方法

在讨论资源约束和系统的细节之前，先谈谈解决问题的策略。解决问题的许多策略分为四个步骤：

1. 识别现象。
2. 判断根源。
3. 实施纠正措施。
4. 对结果进行评估。

识别现象

解决问题的第一步是识别现象。现象的例子包括“电子邮件发送缓慢”或“我把磁盘空间用光了”。这些现象让您的用户不满意，在这些现象消失之前他们会不断地抱怨。但是，不要把现象与问题混为一谈：问题导致了现象的产生，但是问题往往与用户报告的不同。

知道了现象之后，要尽可能了解现象的严重程度和发生这些现象的条件。对于电子邮件系统缓慢，您可能了解到，过去发送电子邮件只需要花费几秒，而现在需要几小时。用光磁盘空间的用户可能正在执行某种操作，比如保存文件或处理一个批作业。

了解这些信息有两个目的。首先是为了重现问题，这在以后有助于判断是否已经解决了问题。第二个目的是从用户那里获取更多细节，帮助判断问题的根源。例如，如果了解到一个作业过去只需 5 分钟就能够执行完，而现在需要一小时，那么就应该问问这个作业是什么性质的。您可能会由此了解到它从另一个服务器上的数据库获取信息，那么在寻找根源时就必须考虑到这个数据库。

判断根源

判断根源需要使用在 [度量资源使用量](#) 一节中学习的命令寻找问题的原因。为此，必须研究 CPU、内存、磁盘和网络等资源。在理想情况下，可以在发生问题的时候用 `vmstat`、`iostat` 和 `top` 等实时工具收集数据。如果做不到，就需要用 `sar` 等工具生成历史信息。

如果问题与资源相关，就会出现两种结果之一：一种（或多种）资源的使用率达到 100%，那么问题的原因就很明显了；或者没有发现被过度使用的资源。

在第二种情况下，应该参考一个基线（baseline）。基线是一组参照数据，可以用它对比“正常”情况和您目前看到的情况。基线可以是一系列显示正常活动的图形或存档 `sar` 报告。基线还有助于预测增长趋势。

在使用管理工具的过程中，您会逐渐发现问题的根源。例如，邮件服务器在处理一个邮件时出现阻塞，导致它停止处理其他邮件。或者，一个批作业占用了系统上的所有 CPU 时间。

您必须确认自己真的找到了问题的根源。例如，某个应用程序会生成巨大的日志文件，导致磁盘空间被用光了。如果您认为日志文件就是问题的根源并决定删除它，那么在未来的某个时候这个应用程序仍然可能用光磁盘空间。

实施纠正措施

纠正一个问题常常有多种方法。以占用所有 CPU 资源的批作业为例。如果杀死这个作业，那么其他用户的工作会恢复正常，但是运行这个作业的用户可能会无法接受。可以降低这个作业的优先级，让其他进程获得更多的 CPU 时间。常常需要根据业务的需要和情况的紧急程度做出公正的决定。

对结果进行评估

实施了纠正措施之后，必须检查问题是否真的解决了。电子邮件能够立即发送出去吗？用户能够登录吗？如果问题仍然存在，就必须重新寻找根源和其他纠正措施。如果纠正措施失败了，还必须确认您没有把事情弄得更糟糕！

纠正了问题之后，还要决定是否需要采取任何长期措施。需要更大的硬盘吗？需要把用户的批作业转移到另一台主机上吗？如果机器上出现莫名其妙的进程，那么可能应该考虑对服务器进行更深入的安全检查，以确保服务器没有被入侵。

复合问题

某些性能问题的原因很明显。一个用户抱怨某些东西运行缓慢，您就登录进系统并运行 `top`。然后发现一个进程不必要地大量占用 CPU 时间，杀死这个进程，系统就恢复正常了。用户把您夸奖了一番，老板也表扬您，美好的一天结束了。

但是，如果问题不这么明显，应该怎么办呢？有时候，问题是由多种原因造成的，或者现象是由初看上去不相关的问题导致的。

交换旋涡

内存的速度很快，而且您的系统中可能有大量内存。但是有时候，一个应用程序或几个进程需要的内存量超过了系统中的物理内存量。在这种情况下，就会使用虚拟内存。内核会在磁盘上分配一个区域，并把驻留的内存页面交换到磁盘上，从而让活跃的应用程序可以使用这些内存空间。当需要交换出去的内存数据时，再把它们取回来，这时可能需要把其他内存交换出去以便让出空间。

这个过程的问题是磁盘是缓慢的。如果只执行少量的交换操作，可能不会注意到性能问题。但是，如果对内存的需求持续增长，为了满足需求，系统开始频繁地把内存交换到磁盘，就会出现问题了。您会发现磁盘 I/O 操作急剧增加，而且系统似乎没有反应了。实际上，系统可能真的没有反应了，这是因为应用程序正在等待它们的内存从磁盘传输回 RAM。

UNIX 管理员把这称为交换旋涡（swap spiral），有时候称为交换死亡旋涡（swap death spiral）。最终，磁盘忙于交换内存，I/O 吞吐量到达极限，系统接近终止状态。如果交换空间与数据放在相同的物理磁盘上，那么情况甚至会更糟糕。由于交换活动频繁，当应用程序获得 CPU 时间并发出 I/O 请求之后，它不得不等待更长时间。

交换旋涡的明显现象是做任何事情都要长时间等待，甚至是查看运行时间这样的简单活动。还会看到很高的平均负载，因为在运行队列中有许多进程等待运行。如果 CPU 时间被大量占用也会有相似的现象，为了区分这两个问题，可以运行 `top`，检查是否有进程大量占用 CPU；也可以运行 `vmstat`，查看是否有大量交换活动。尽管根据问题的性质，问题可能自行消失，但解决方案常常是一个个地杀死进程，直到系统恢复正常。

用光磁盘空间

应用程序不一定会检查错误。许多应用程序总是假设每次磁盘访问都会顺利且迅速地完成。磁盘容量被用光常常会导致应用程序表现异常。例如，应用程序可能会重复执行一个操作，而没有意识到这个操作无法完成，这就会占用所有可用的 CPU 时间。可以使用 `strace` 命令查看应用程序正在做什么（如果它使用系统调用的话）。

有时候，应用程序只是停止工作。如果 Web 应用程序无法访问它的数据库，它可能会返回空页面。

登录并检查可用磁盘（用 `du`）是检查磁盘空间的最快的方法。

由于 I/O 而阻塞

当一个进程请求某种形式的 I/O 时，内核会让这个进程进入睡眠状态，直到 I/O 请求返回。如果磁盘出了某些问题（比如交换旋涡、磁盘故障或网络文件系统上的网络故障），那么会有许多应用程序同时处于睡眠状态。

睡眠状态包括可中断睡眠和不可中断睡眠两种。可以通过信号杀死处于前一种睡眠状态的进程，但是对于后一种睡眠无法这样做。运行 `ps aux` 显示状态。清单 12 显示一个进程处于不可中断睡眠，另一个进程处于可中断睡眠。

清单 12. 两个进程处于睡眠状态

apache	26575	0.2	19.6	132572	50104	?	S	Feb13	3:43	/usr/sbin/httpd
root	8381	57.8	0.2	3844	532	pts/1	D	20:46	0:37	dd

清单 12 中的第一个进程 `httpd` 处于可中断睡眠状态，这由问号后面的字母 `S` 表示。第二个进程 `dd` 处于不可中断睡眠状态。不可中断睡眠常常与硬盘访问相关，而可中断睡眠用于执行时间比较长的操作，比如 NFS 和套接字操作。

如果发现平均负载很高（这意味着运行队列中有大量进程），而且大量进程处于不可中断睡眠状态，那么可能是在硬盘 I/O 方面出现了问题，原因可能是设备故障或者是同时执行过多的读写操作。

分析需求

本节讨论 Senior Level Linux Professional（LPIC-3）考试 301 的 306.3 主题的内容。这个主题的权值为 2。

在本节中，学习如何：

- 识别容量需求
- 详细描述程序的容量需求
- 判断程序的 CPU/内存需求
- 把程序需求合并为完整的分析

纠正当前的问题是系统管理员的关键任务之一。另一个任务是分析系统当前的运行情况，预测并提前解决资源约束。本节讨论如何分析当前需求，下一节讨论如何预测未来的使用情况。

可以以两种方式分析当前的需求：在一段时间内度量当前需求（与基线相似），或者建立系统的模型并设置一组参数，使模型反映当前的情况。第一种方式比较容易，效果也相当好。第二种方式更

精确，但是需要做大量工作。建模方式的真正价值在于预测未来的情况。如果有了系统的模型，就能够根据增长趋势修改某些参数，查看性能的变化。

在实践中，常常同时使用这两种方式。在某些情况下，很难对系统进行建模，所以度量值是需求和增长趋势的惟一基础。在建立模型时也需要度量值。

系统行为建模

计算机中的活动可以建模成一系列队列。请求进入队列的一端，排队等待某种资源可用。当这种资源可用时，执行这个任务，任务退出队列。

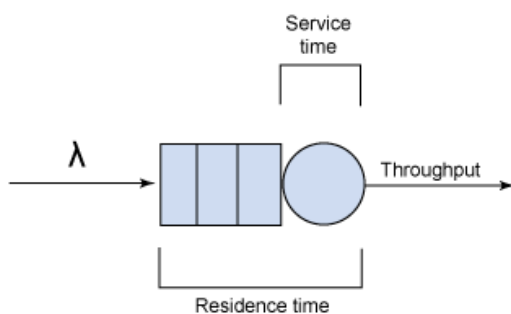
可以把多个队列组合在一起，形成更大的系统。一个磁盘可以建模成一个队列，磁盘请求在这里进入一个缓冲区。当磁盘准备为这个请求服务时，把这个请求传递给磁盘。请求一般来自 CPU，多个任务会争用 CPU 时间。对队列及其应用的研究称为队列理论。

Analyzing Computer System Performance with Perl::PDQ 这本书（参见 [参考资料](#) 中的链接）介绍了队列理论并解释如何把计算机系统建模成一系列队列。它还介绍一个称为 PDQ 的 C 库和一个相关联的 Perl 接口，可以用它们定义队列来估计性能。可以通过修改参数来估计变化对系统的性能。

队列简介

图 4 给出一个队列。请求从左边进入队列。随着请求得到处理，它们会离开队列。左边的方块表示排队的对象。

图 4. 一个简单的队列



队列的行为按照时间、速率和大小来度量。到达速率表示为 Λ (lambda)，常常表达为每秒请求数。决定 Λ 的方法是在一段合理的时间段内观察系统，统计到达的请求。合理的时间段至少是服务时间的 100 倍，服务时间是处理请求的时间长度。驻留时间是请求花费在队列中的总时间，包括处理请求花费的时间。

到达速率描述请求进入队列的速率，而吞吐量定义请求离开队列的速率。在更复杂的系统中，节点吞吐量 (nodal throughput) 定义单一队列节点的吞吐量，而系统吞吐量是指系统的总吞吐量。

在大多数情况下，缓冲区大小的意义不大，因为只要满足以下条件，缓冲区就具有有限的可预测的大小：

- 缓冲区足以容纳排队的对象。
- 队列不会无限地增长。

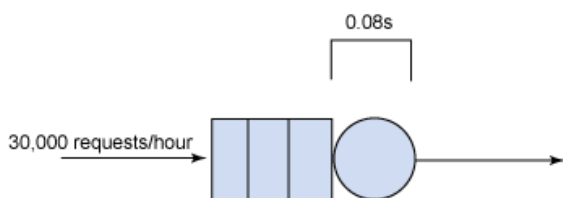
第二个约束是最重要的。如果一个队列以每秒一个请求的速率分派请求，但是请求到达的速率超过每秒一个请求，队列就会无限地增长。在实际情况中，到达速率是波动的，但是性能分析关心的是稳定状态，所以要使用平均值。可能在某个时间段每秒有 10 个请求到达，而在其他时候没有请求到达。只要平均到达速率小于每秒一个请求，队列的长度就是有限的。如果平均到达速率超过了分派请求的速率，队列长度就会持续增长，无法到达稳定状态。

图 4 中的队列称为开放队列，因为它对到达的请求数量没有限制，而且在处理请求之后请求不需要返回。封闭队列会对输入施加反馈；系统中的请求数是有限的。在处理请求之后，请求回到到达队列中。

队列的典型例子是商店的交款队列。进入队列的人数除以度量时间间隔就是到达速率。离开队列的人数除以度量时间间隔就是吞吐量。一位收款员处理一位顾客花费的平均时间就是服务时间。顾客在队列中等待的平均时间加上服务时间就是驻留时间。

为了了解 PDQ 的概念，请考虑以下场景。一个 Web 服务每小时要处理 30,000 个请求。通过观察低负载时的系统，发现服务时间是 0.08 秒。图 5 给出这个队列的情况。

图 5. 建模为队列的 Web 服务



PDQ 能够提供什么信息呢？清单 13 给出所需的 PDQ 程序及其输出。

清单 13. PDQ 程序及其输出

```
#!/usr/bin/perl
use strict;
use pdq;
# Observations
my $arrivals = 30000; # requests
my $period = 3600; # seconds
my $serviceTime = 0.08; # seconds

# Derived
my $arrivalRate = $arrivals / $period;
my $throughput = 1 / $serviceTime;
# Sanity check -- make sure arrival rate < throughput
if ($arrivalRate > $throughput) {
    die "Arrival rate $arrivalRate > throughput $throughput";
}

# Create the PDQ model and define some units

pdq::Init("Web Service");
pdq::SetWUnit("Requests");
pdq::SetTUnit("Seconds");
# The queuing node
pdq::CreateNode("webservice", $pdq::CEN, $pdq::FCFS);

# The circuit
pdq::CreateOpen("system", $arrivalRate);

# Set the service demand
```

```
pdq::SetDemand("webservice", "system", $serviceTime);

# Run the report
pdq::Solve($pdq::CANON);
pdq::Report();

..... output ..

***** Pretty Damn Quick REPORT *****
*****
***   of : Sat Feb 16 11:24:54 2008   ***
***  for: Web Service                ***
*** Ver: PDQ Analyzer v4.2 20070228 ***
*****
*****

***** PDQ Model INPUTS *****
*****

Node Sched Resource   Workload   Class   Demand
---- -
CEN  FCFS  webservice system    TRANS   0.0800

Queueing Circuit Totals:

      Streams:      1
      Nodes:       1

WORKLOAD Parameters

Source      per Sec      Demand
-----
system      8.3333      0.0800

***** PDQ Model OUTPUTS *****
*****

Solution Method: CANON

***** SYSTEM Performance *****

Metric      Value      Unit
-----
Workload: "system"
Mean Throughput      8.3333      Requests/Seconds
Response Time        0.2400      Seconds

Bounds Analysis:
Max Demand          12.5000      Requests/Seconds
Max Throughput      12.5000      Requests/Seconds
```

***** RESOURCE Performance *****				
Metric	Resource	Work	Value	Unit
-----	-----	----	-----	----
Throughput	webservice	system	8.3333	Requests/Seconds
Utilization	webservice	system	66.6667	Percent
Queue Length	webservice	system	2.0000	Requests
Residence Time	webservice	system	0.2400	Seconds

清单 13 首先定义程序其余部分使用的解释器。前两行 Perl 代码调用 PDQ 模块和 strict 模块。PDQ 提供 PDQ 函数，而 strict 模块强制实施良好的 Perl 编程方法。

清单 13 中后面一部分代码定义观察系统所需的变量。给出这些信息之后，计算到达速率和吞吐量。后者是服务时间的倒数 — 如果用 N 秒处理一个请求，那么每秒就可以处理 1/N 个请求。

安装 PDQ

可以从作者的网站下载 PDQ 压缩文件（参见 [参考资料](#)）。用 `tar -xzf pdq.tar.gz` 命令把它解压到一个临时目录中，用 `cd pdq42` 命令进入新创建的目录。然后运行 `make` 对 C 代码和 Perl 模块进行编译。最后，运行 `cd perl5` 和 `./setup.sh`，从而构建 Perl 模块并把它安装在系统目录中。

健康检查（sanity check）确保队列长度是有限的。大多数 PDQ 函数会报告遇到的错误，但是模块的作者建议进行显式的检查。如果请求的到达速率超过离开的速率，那么这个程序会因错误而终止。

程序的其余部分直接调用 PDQ 函数。首先，用模型的标题对模块进行初始化。然后，设置时间单位和工作单位，让报告按照您希望的方式显示信息。

用 `CreateNode` 函数创建每个队列。在清单 13 中，创建了一个称为 `webservice` 的队列（合适的名称有助于理解最终的报告）。队列类型为 `CEN`（队列中心），与之相反的是延迟节点，延迟节点不做任何工作。这个队列是标准的先进先出（first in, first out, FIFO）队列，PDQ 把这种队列称为先到先服务队列（first-come first-served queue）。

接下来，调用 `CreateOpen` 创建回路（circuit），回路是队列的集合。回路的到达速率已经计算出来了。最后，用 `SetDemand` 设置队列的需求。`SetDemand` 定义完成某个工作负载（回路中的一个队列）所花的时间。

最后，用 `Solve` 函数计算回路并用 `Report` 函数报告结果。注意，PDQ 接收您的模型，把它转换为一系列算式，然后计算它们。PDQ 并不以任何方式模拟模型。

输出是很容易理解的。报告首先显示一个标题和对模型的总结。**WORKLOAD Parameters** 部分提供一些比较有意义的信息。回路的服务时间定义为 0.08 秒。每秒速率是输入速率。

SYSTEM Performance 部分计算系统的总体性能。这里的回路能够应付每秒 8.3333 个请求的输入速率。响应时间是 0.24 秒，这个时间包括 0.08 秒的服务时间和在队列中花费的时间。回路的最高性能是每秒 12.5 个请求。

仔细看一下报告，可以看到它的使用率是 66.6667%。队列的平均长度是两个请求。这意味着，当一个请求进入队列时，平均有两个请求排在它前面，再加上正在处理的请求。服务时间为每个请求 0.08 秒，平均驻留时间为前面报告的 0.24 秒。

可以扩展这个模型，从而显示这个 Web 服务的组件。在这种情况下，不使用单一队列代表 Web 服务，而是建立多个队列：一个队列处理请求，一个队列访问数据库，一个队列对响应进行打包。如果模型是有效的，那么系统性能应该是相同的，但是现在能够看到 Web 服务的内部工作情况。在此基础上，可以做一些试验性的研究，比如设置更快的数据库或更多 Web 服务器，看看对响应有什么样的影响。可以通过各个资源数据了解特定队列是否是瓶颈，以及需要增加多少空间。

清单 13 只是使用 PDQ 库的基本示例。请阅读 *Analyzing Computer System Performance with Perl::PDQ* (参见 [参考资料](#) 中的链接)，了解如何构建更复杂的模型。

预测未来的资源需求

本节讨论 Senior Level Linux Professional (LPIC-3) 考试 301 的 306.4 主题的内容。这个主题的权值为 1。

在本节中，学习如何：

- 预测配置的容量断点
- 观察使用量的增长速度
- 用图形显示使用量的变化趋势

前一节 介绍了 PDQ 库和一个示例报告。这个报告显示一个队列的使用率和最大负载的计算值，反映了系统的总体性能。可以使用相同的方法预测配置的断点。还可以用图形显示系统使用量随时间增长的趋势，预测什么时候会达到容量极限。

再论 PDQ

清单 14 对与 [清单 13](#) 相同的 Web 服务进行建模，但是这一次把系统分成两个队列：一个队列代表用来处理请求和响应的 Web 服务器上的 CPU 时间，另一个队列显示等待数据库请求返回的时间。

清单 14. 针对示例 Web 服务的新的 PDQ 程序

```
#!/usr/bin/perl
use strict;
use pdq;
# Observations
my $arrivals = 30000; # requests
my $period = 3600; # seconds

# Derived
my $arrivalRate = $arrivals / $period;

# Create the PDQ model and define some units

pdq::Init("Web Service");
pdq::SetWUnit("Requests");
pdq::SetTUnit("Seconds");

# The queuing nodes
pdq::CreateNode("dblookup", $pdq::CEN, $pdq::FCFS);
pdq::CreateNode("process", $pdq::CEN, $pdq::FCFS);

# The circuit
```

```
pdq::CreateOpen("system", $arrivalRate);

# Set the service demand

pdq::SetDemand("dblookup", "system", 0.05);
pdq::SetDemand("process", "system", 0.03);

# Solve
pdq::Solve($pdq::CANON);
pdq::Report();
```

清单 14 中的代码在系统中增加了一个队列。总服务时间仍然是 0.08 秒，其中数据库查询时间为 0.05 秒，CPU 处理时间为 0.03 秒。清单 15 给出生成的报告。

清单 15. 清单 14 生成的 PDQ 报告

```
*****
*****  Pretty Damn Quick REPORT  *****
*****
***   of : Sun Feb 17 11:35:35 2008   ***
***   for: Web Service                 ***
***   Ver: PDQ Analyzer v4.2 20070228***
*****
*****
*****  PDQ Model INPUTS  *****
*****

Node Sched Resource  Workload  Class    Demand
----  -
CEN  FCFS  dblookup  system   TRANS    0.0500
CEN  FCFS  process   system   TRANS    0.0300

Queueing Circuit Totals:

Streams:      1
Nodes:        2

WORKLOAD Parameters

Source      per Sec      Demand
-----
system      8.3333        0.0800

*****
*****  PDQ Model OUTPUTS  *****
*****

Solution Method: CANON

*****  SYSTEM Performance  *****
```

Metric	Value	Unit	
-----	-----	----	
Workload: "system"			
Mean Throughput	8.3333	Requests/Seconds	
Response Time	0.1257	Seconds	
Bounds Analysis:			
Max Demand	20.0000	Requests/Seconds	
Max Throughput	20.0000	Requests/Seconds	
***** RESOURCE Performance *****			
Metric	Resource	Work	Value
-----	-----	----	-----
Throughput	dblookup	system	8.3333
Utilization	dblookup	system	41.6667
Queue Length	dblookup	system	0.7143
Residence Time	dblookup	system	0.0857
Throughput	process	system	8.3333
Utilization	process	system	25.0000
Queue Length	process	system	0.3333
Residence Time	process	system	0.0400

看一下报告的输出部分。与清单 13 相比，平均响应时间降低了，每秒最大请求数从 12.5 提高到了 20。这是因为新的模型支持流水线化（pipelining）。在把一个请求分派给数据库的同时，CPU 可以处理另一个请求。在原来的模型中，因为只使用一个队列，所以这是不可能的。

更重要的是，数据库的使用率是 42%，CPU 的使用率只有 25%。因此，随着系统负载的增加，数据库会首先达到容量极限。

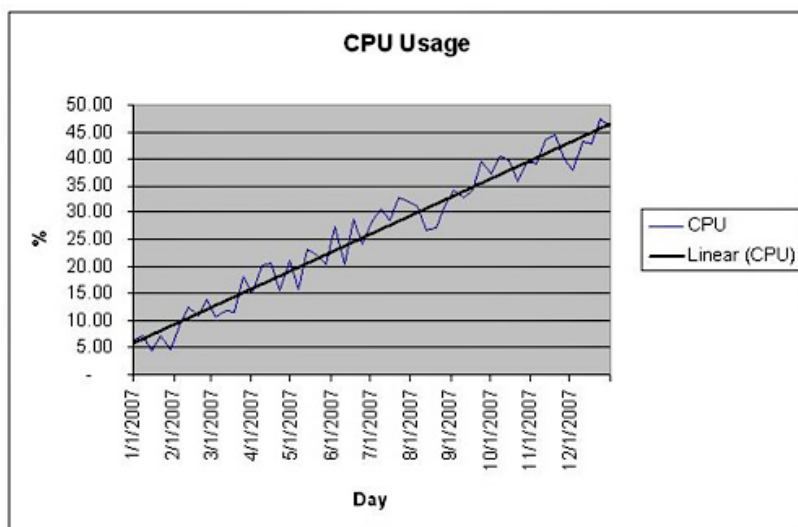
把到达速率改为每小时 60,000 个，平均响应时间就会增加到 0.36 秒，数据库使用率增加到 83%。另外，在 0.36 秒的响应时间中，0.30 秒花费在等待数据库上。因此，在进行性能调优时，主要精力最好放在加快数据库访问上。

可以以不同方式定义最大容量。当每秒请求数为 20 时（见报告的顶部），系统达到 100% 容量。还可以根据平均响应时间来定义容量。当每秒请求数大约为 15 时，响应时间将超过 0.25 秒。如果您的目标是把响应时间控制在 0.25 秒以下，那么系统在这个速率下就达到了容量极限，尽管硬件仍然能够支持更大的速率。

用图形帮助分析

图形是显示历史信息的出色手段。可以通过图形查看一段时期（比如 6 个月或一年）的统计数据，了解增长的速度。图 6 给出一个应用服务器在一年中的 CPU 使用率。平均每日使用率数据被记录在一个电子表格中，并绘制成图形。还添加了一条趋势线来显示增长趋势。

图 6. 服务器的 CPU 使用率图形



可以通过这个图形预测未来的使用量（假设增长速度保持稳定）。图 6 表明，这台服务器上的负载增长大约为每 3 个月 10%。在更高的 CPU 使用率时队列效应更加明显，所以可能需要在达到 100% 使用率之前进行升级。

如何绘制图形

电子表格方式不适合处理大量服务器和度量指标。更合适的一种方法是，获取 `sar` 的输出并传递给 `GNUplot` 等图形绘制工具。有许多图形绘制工具，其中一部分是开放源码的。开放源码工具包括基于 `RRDTool` 包的一系列工具。

`RRDTool` 是一系列程序和库，它们把数据输入一个循环式数据库（`RRD`）。`RRD` 会连续地对数据进行存档，所以可以保存时间跨度很不同的数据，比如去年的每小时数据和本周的 5 分钟平均值。这种数据库不断地把旧数据存档，所以数据库不会增长。`RRDTool` 还附带图形绘制工具。

[参考资料](#) 中介绍了几种出色的图形绘制工具。

绘制哪些数据

应该把对于您的服务最重要的信息绘制成图形，包括可能帮助您做出决策的任何信息。图形有助于了解过去发生的情况，甚至可以把风扇速度绘制成图形。但是，在一般情况下，注意力应该集中在 CPU、内存、磁盘和网络统计数据的绘制上。如果可能，应该绘制服务的响应时间。这不但有助于根据用户体验更好地做出决策，还有助于开发系统的模型。

结束语

在本教程中，学习了如何度量和分析性能。还学习了如何用度量值解决性能问题。

Linux 提供与系统状态相关的大量信息。`vmstat`、`iostat` 和 `ps` 等工具提供实时信息。`sar` 等工具提供长期信息。请记住，在 `vmstat` 和 `iostat` 的输出中，第一行数据不是实时的。

在解决系统问题时，应该首先识别问题的现象，这有助于理解问题以及在采取纠正措施之后确认问题是否已经解决。然后，在发生问题的同时度量系统资源（如果可能的话），从而判断问题的原因。找到纠正措施之后，就实施它，然后评估结果。

可以使用 PDQ Perl 模块计算队列问题。在把系统建模为一系列队列之后，可以使用 PDQ 函数编写一个 Perl 脚本。然后，使用系统模型根据当前使用量和预测的未来使用量计算性能。

可以使用模型和图形预测增长。在理想情况下，应该同时使用这两种方法并对比其结果。

现在，这个为准备 LPIC 3 考试而设计的系列结束了。如果您正要参加考试，我祝您成功，希望本系列对您有帮助。

参考资料

学习

- 回顾这个 301 系列中的前一个教程“[LPI 301 考试准备，主题 305：集成和迁移](#)”（developerWorks，2008 年 4 月）或 301 系列中的[所有教程](#)。
- 回顾 developerWorks 上的整个 [LPI 考试准备系列教程](#)，学习 Linux 的基础知识并准备系统管理员认证考试。
- 在 [LPIC Program](#) 中，可以找到 Linux Professional Institute 的三个 Linux 系统管理认证级别的任务清单、问题示例和详细的学习目标。
- 阅读 Neil Gunther 所写的 [UNIX Load Average Part 1: How It Works](#) 和 [UNIX Load Average Part 2: Not Your Average Average](#)。Neil Gunther 经营着 [Performance Dynamics Company](#)，还撰写了 [其他一些文章](#) 以及两本书 *Analyzing Computer System Performance with Perl::PDQ* 和 *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services*。
- Sean 的文章“[使用 SAR 进行简单的系统监视](#)”（developerWorks，2006 年 2 月）介绍了 SAR。这篇文章是针对 Solaris 系统撰写的，所以命令行参数有所不同，但是理论是相同的。
- Sean 的教程“[Expose Web performance problems with the RRDTool](#)”（developerWorks，2006 年 3 月）讲解了如何用 RRDTool 监视 Web 站点的性能并用图形显示结果。
- [sysstat FAQ](#) 是帮助解决 SAR 问题的好地方。
- 在 [developerWorks Linux 专区](#) 中可以找到为 Linux 开发人员准备的更多参考资料，包括[Linux 教程](#)以及上个月[读者最喜欢的 Linux 文章和教程](#)。
- 随时关注 [developerWorks 技术事件和网络广播](#)。

获得产品和技术

- 获取 [PDQ 的源代码](#) 以及针对各种操作系统的安装说明。
- [RRDTool](#) 是大多数开放源码网络监视系统的基础，它的前身是著名的 [Multi Router Traffic Grapher](#) 包。RRDTool 可以作为另一个包的组成部分，也可以在您的脚本中使用它。
- [Cacti](#) 是最好的开放源码监视包之一。它主要用于监视网络设备，但是也能够执行各种系统任务。Cacti 拥有一个活跃的用户社区，您可以在论坛中得到热情的帮助。
- [ZABBIX](#) 是另一种值得关注的开放源码系统监视包。
- 查阅 developerWorks 的 [Tivoli](#) 区域，了解关于 IBM 的企业系统、网络、安全性和应用程序管理工具的更多信息。特别是，[Tivoli Composite Application Management](#) 解决方案有助于提高业务关键的复合应用程序的性能和可用性，包括门户和基于 SOA 的技术。
- 使用 [IBM 试用软件](#) 构建您的下一个 Linux 开发项目，这些软件可以从 developerWorks 直接下载。

讨论

- 通过 [新的 developerWorks spaces](#) 中的 blog、论坛、podcast 和社区主题，加入 [developerWorks 社区](#)。

关于作者

Sean Walberg



从 1994 开始，Sean Walberg 就一直在学术、企业和 Internet 服务提供者环境中从事 Linux 和 UNIX 系统的研究。在过去几年里，他撰写了大量有关系统管理的文章。

© 版权所有 IBM 公司 2008

(www.ibm.com/legal/copytrade.shtml)

商标

(www.ibm.com/developerworks/cn/ibm/trademarks/)