文章编号:1001 - 9081(2006)05 - 1180 - 03

# 一种评测应用程序实际性能开销的方法

钱丽萍[1],汪立东[2]
(1. 北京建筑工程学院 计算机科学与技术系, 北京 100044;
2. 国家计算机网络应急技术处理协调中心, 北京 100029)
(wangzs@ vip. sina. com)

**摘 要**:为评测应用程序的性能开销,提出基序列时延度量法。通过选择合理的基准程序,在其中植入若干观察点,在待测应用程序运行前及运行时,分别评测基准程序各观察点的运行时延,可以测量待测应用程序在各种负载下运行时的相对性能开销。现已在 Linux 上通过可装入内核模块和系统调用截获技术验证了该方法。

**关键词**:性能; 基准; 系统调用; 时延
**中图分类号**: TP302.7 **文献标识码**:A

# Method for evaluating real performance costs of applications

QIAN Li-ping[1], WANG Li-dong[2]
(1. *Department of Computer Science & Technology, Beijing Institute of Civil Engineering and Architecture, Beijing* 100044, *China;*
2. *National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing* 100029, *China*)

**Abstract:** A method to evaluate performance cost of applications was Presented based on execution delay of benchmarking-sequence. With well-chosen benchmark programs and well-designed viewpoints planted in them, the real costs of performance of a certain application can be evaluated relatively by calculating the delay of the characteristic viewpoints before and during executing the application under test. Such method was implemented on Linux with techniques of loadable kernel module and system call interception.

**Key words:** performance; benchmark; system call; delay

## 0 引言

当前对计算机系统进行性能评估的主要方法是针对系统的功能,利用各种测试工具获得系统在各种负载下的工作能力,如测试网络系统的吞吐量、延迟、丢包率、并发连接数等。当前主流的测试工具大致可分为两类,一类是业界标准的测试基准,如 SPEC 系列、TPC 系列等,它们可以给出复杂负载情况下的可靠指标,但一般较为昂贵。另一类是开放源代码及免费的测试工具,这类工具较多,包含单一功能的测试工具和综合测试工具,前者如系统自身拥有的 ping、netstat 等基本命令,比较著名的综合测试工具有 Linpack、lmbench、webbench、netbench 和 IOzone 等。

虽然综合利用目前已有的测试工具已足够了解系统的整体运行情况,但除了一些专门用于测试 Web 服务器、邮件服务器等应用的专用测试工具外,大多数测试工具一般是给出当前系统某一方面(如 CPU 利用率、内存利用率、I/O 吞吐量、磁盘读写效率等)或整体运行性能的评价,它们生成的结果是用来证明一个系统的硬件或软件在某些方面的性能满足需要。这些测试工具往往不区分特定应用程序所需的性能开销,难以揭示应用程序的运行对整个系统的影响[1],因此不足以评估实际负载下应用程序的性能,更无法给出优化信息。

另一方面,目前越来越多的应用是基于网络和多媒体内容的,这些应用会频繁地处理语音、图像和视频信息,需要在内存中及内存和磁盘间传输大量数据,从而增加了在系统内核态的执行时间。如文献[2]曾指出:Web 服务器有 85% 的时间是在运行操作系统代码。然而目前即使是针对特定应用的测试工具,其结果也无法反映出应用程序工作流程与操作系统之间或操作系统与硬件之间的复杂交互,不能将 I/O 和睡眠时间与用户进程请求联系起来,也无法揭示应用程序在不同负载情况下对内核的使用变化情况,因此不能提供准确的机制来确定应用程序在内核态的运行时延。正是由于传统的测试基准未将重点放在依赖操作系统的应用上,也往往不去区分应用程序在内核态执行和在用户态执行时的性能影响,因此需要设计针对操作系统内核的性能测量工具。

## 1 基序列时延度量法

实际上,任何应用程序的执行,最终都是转化为调用一系列内核系统调用,它对系统资源的使用情况,最终反映在执行时间上。应用程序所生成的一系列系统调用及其调用时间,可以看作一个时间序列,该序列与应用程序的执行环境和输入参数有关,并因需要竞争系统资源而受到其他运行程序的影响。因此,如果能够以标准系统环境下某个典型程序的某个特征系统调用序列的执行时间作为参照,则可以相对地测量其他应用程序运行所造成的性能开销。即,首先选择典型

**收稿日期**:2005 - 11 - 03;**修订日期**:2006 - 01 - 09

**作者简介**:钱丽萍(1971 - ),女,安徽黄山人,讲师,硕士,主要研究方向:中文信息处理、信息安全、软件测试; 汪立东(1970 - ),男,安徽黄山人,工程师,博士,主要研究方向:网络与信息安全、系统测试.

程序(称为基准程序)的某个特征系统调用序列作为参照序列(称为基序列),并在待测应用程序运行前,测出基序列各系统调用间的平均时延作为基准时延 $D$;然后当待测程序以特定负载运行时,再次测得基序列各系统调用间的平均时延 $D'$。通过 $D$ 和 $D'$,可以在一定程度上了解应用程序以特定负载运行时的系统性能开销。该方法称为基序列时延度量法。

基序列时延度量法首先需要选择合适的基准程序。鉴于该方法并非直接去测量应用程序的性能开销,而是通过测量基序列的相对时延去间接评价应用程序对系统的性能影响,因此基准程序的选择与传统使用的性能指标(如 CPU 利用率、内存利用率、磁盘读写效率等)并无直接关系,既可以侧重简单函数延迟、进程间通信、上下文转换、内存读取、文件系统访问、网络传输等单项指标,亦可以是复合功能的程序或程序集。基准程序和基序列的选择,需要考虑承担测试的硬件平台的性能。为适于评测大中型软件,基准程序的执行时间应足够覆盖待测功能的执行;为适用于评测中小型软件,基序列可以遵循前紧后松的原则设计。基准程序和基序列一旦确定,则可以在测试平台上测量、对比各种应用的性能开销。

其次,需要定义一个性能开销函数。设 $\alpha$ 是一个基准程序,$\Psi$ 是系统调用全集。$\alpha$ 在特定执行环境和特定输入参数下执行时,生成的系统调用时间序列 $S$ 为 $< S_0(t_0, x_0), S_1(t_1, x_1), S_2(t_2, x_2), \cdots, S_n(t_n, x_n) >, S_i \in \Psi, t_0 = 0, t_i$ 是 $S_i$ 自 $t_0$ 开始的相对调用时间,$x_i$ 是 $S_i$ 的参数($i = 0, \cdots, n$)。设 $\Psi' \subseteq \Psi$,$\Psi'$ 是性能测试所关注的系统调用集。$S' = < S'_0(t'_0, x'_0), S'_1(t'_1, x'_1), S'_2(t'_2, x'_2), \cdots, S'_m(t'_m, x'_m) >$ 是 $S$ 的一个子序列,且 $S'_j \in \Psi'(j = 1, \cdots, m)$,$x'_j$ 是 $S'_j$ 的参数,$S'_j(x'_j)$ 是 $\alpha$ 的一个观察点,称 $S'$ 是 $\alpha$ 的一个基序列。设 $\gamma$ 是一个待测应用程序,当 $\gamma$ 以负载 $L$ 运行时,测得观察点 $S'_j(x'_j)$ 的自 $t'_0$ 开始的相对调用时间为 $t'_{\gamma Lj}$,$S_{\gamma L}' = < S'_{\gamma L0}(t'_{\gamma L0}, x'_0), S'_{\gamma L1}(t'_{\gamma L1}, x'_1), \cdots, S'_{\gamma Lm}(t'_{\gamma Lm}, x'_m) >$ 称为 $S'$ 一个时延序列,$\gamma$ 在基准程序 $\alpha$ 和特定负载 $L$ 下的性能开销函数表示为 $PCost(\alpha, L, \gamma)$。$PCost(\alpha, L, \gamma)$ 的构造,一方面要体现基序列与时延序列的关系,另一方面可以利用实例应用程序的执行进行调优,使其既能反映系统性能开销的变化,又能反映应用程序的实际运行特点。本文根据测试平台上多个应用程序的测试实验数据,对 $PCost(\alpha, L, \gamma)$ 定义如下:

$$PCost(\alpha,L,\gamma) = 100\% * \sum_{j=1}^{m} \frac{\Delta_{\gamma Lj} - \Delta_j}{t'_{\gamma Lj} + t'_j} \quad (1)$$

其中 $\Delta_{\gamma Lj} = t'_{\gamma Lj} - t'_{\gamma L(j-1)}$,$\Delta_j = t'_j - t'_{(j-1)}$。

当选择一组基准程序 $\alpha_1$、$\alpha_2$、$\cdots$、$\alpha_z$ 时,定义 $\gamma$ 在特定负载 $L$ 下的性能开销函数 $PCost(L, \gamma)$ 为:

$$PCost(L,\gamma) = \sum_{i=1}^{z} PCost(\alpha_i,L,\gamma)/z \quad (2)$$

通过度量各种负载情况下 $PCost(L, \gamma)$ 的值,可以获得 $\gamma$ 的性能开销曲线。同时,如果选择 $\gamma$ 自身的一个系统调用序列作为基序列,则基序列时延度量法有可能发现对 $\gamma$ 执行时间影响较大的系统调用,从而有利于发现 $\gamma$ 的性能瓶颈并改进 $\gamma$ 的性能。

第三,基序列时延度量法的实现前提是截获特定系统调用并在此基础上设置合适的观察点。Linux 上的系统调用截获可以通过可装入内核模块(Loadable Kernel Module,LKM)技术实现[3],Windows 上的系统调用截获方法可参见[4]。

## 2　应用程序的性能开销评测

例子采用基序列时延度量法来评测 Linux 上 tcpdump 和 make 工具。对这类应用,传统测试工具无法揭示其对内核的使用变化情况,而基序列时延度量法特别适合对比测量不同负载情况下的应用程序的性能开销。

### 2.1　基准程序和基序列的选择

基准程序选用 UNIX 上比较典型的文件打包程序 tar,因为它与 CPU、I/O 和内存利用率均关联密切。测试平台为 RedHat Linux 9,内核版本 2.4.31,基准程序的操作目录为 /usr/src/linux-2.4.20-8,该目录下共有 12 688 个子目录或文件,占用的磁盘存储空间约 187 兆字节。实验中,基序列采用随机方法构造,在操作目录中 6 个不同位置植入指定名称的观察文件,将执行基准程序对应的 execve 系统调用以及打开植入文件的 open 系统调用作为 7 个观察点,通过系统调用截获获得基准程序到达观察点时的时间,从而获得基序列和时延序列。

### 2.2　Linux 系统调用截获

为了获得基序列,需要截获系统调用并打上时间戳。对 Linux,系统调用是从用户空间到内核空间的转换,如用户空间打开文件,对应到内核空间调用 sys_open()。Linux 上有 100 多个系统调用,为避免因截获过多系统调用造成系统性能下降,选择截获 open 和 execve 两个系统调用。

为了截获系统调用,创建了一个字符设备,采用 Linux 的 LKM 技术来实现该设备的驱动程序,在其中截获 execve 和 open 系统调用。新的 execve 和 open 调用处理程序首先获取系统时间、进程号等系统参数,并将相关信息加入一个内核先进先出(FIFO)队列中,然后再执行原处理程序。新创建的字符设备驱动程序也共享访问该内核 FIFO 队列:当有用户进程读该设备时,驱动程序中的读操作处理函数从 FIFO 队列中取出一个结点并将其复制到用户空间供用户进程使用;若队列中无数据可用,则阻塞用户进程。因此在用户空间通过打开并读取该字符设备,可以获取被截获系统调用的相关信息。

为评估系统调用截获是否显著影响系统性能,利用 Linux 的 time 程序计算出基准程序 tar 在截获系统调用前后的执行时间,见表 1。为降低受系统中其他定时或突发进程影响的可能性,表 1 中每种评估均执行 11 次,去掉最高值后取平均值。可以看出,系统调用截获对系统性能的影响可以忽略。

**表 1　基准程序 tar 的执行时间**

| | 总时间/s | 用户态时间/s | 系统态时间/s |
|---|---|---|---|
| 截获前 | 154.219 | 61.348 | 7.199 |
| 截获后 | 154.458 | 61.049 | 7.301 |

### 2.3　tcpdump 的性能开销评测

为利用基序列时延度量法测试 tcpdump 处理不同流量网络数据时的性能开销,专门定制了一个流量发生器,以不同速率发送 UDP 数据包,包长为 1 064 字节(含包头)。待测应用的运行方式为"tcpdump -eX -s0 -i eth0 'udp'",即以最大长度捕获所有 UDP 包,并打印包头和包内容。表 2 是通过系统调用截获记录的基序列和时延序列。

表 2 tar 对 tcpdump 的基序列和时延序列

| 流量 | $t0$ | $t1$ | $t2$ | $t3$ | $t4$ | $t5$ | $t6$ |
|---|---|---|---|---|---|---|---|
| 0(基序列) | 0 | 3 | 16 | 56 | 61 | 70 | 155 |
| 800kbps | 0 | 3 | 16 | 57 | 62 | 72 | 156 |
| 8Mbps | 0 | 3 | 18 | 60 | 65 | 76 | 166 |
| 24Mbps | 0 | 3 | 18 | 60 | 66 | 77 | 168 |
| 40Mbps | 0 | 4 | 18 | 62 | 68 | 79 | 174 |
| 80Mbps | 0 | 4 | 20 | 67 | 73 | 85 | 183 |

由表 2 及 $PCost(\alpha, L, \gamma)$ 的定义，可以计算出 tcpdump 以“-eX -s0 -i eth0 'udp'”参数运行时，在不同网络流量下的性能开销，如图 1 所示。

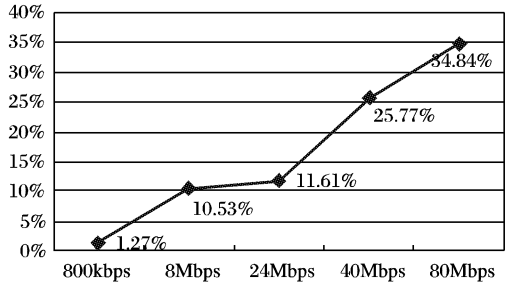


图 1 $PCost(tar, L, tcpdump)$

从图 1 可以看出，tcpdump 所捕获的网络流量自 0 至 8Mbps 时，性能开销上升较大，而从 8Mbps 到 24Mbps 的开销增长很缓，超过 24Mbps 后性能开销又显著增加。这一结果符合网络数据包捕获的特点[5]——当捕包速度低于 8Mpbs 时，系统有较充足的时间和内核内存将数据包从内核空间拷贝到用户空间供捕包程序分析；随着网络流量的增加，操作系统执行的系统调用数线性增加，内核内存开始成为瓶颈，特别是当网络流量超过 20 多 Mbps 时，丢包率逐渐上升，系统开销显著增大。

基准程序在系统态的执行特征可以相对反映待测应用在系统态的执行特征。表 3 是用 time 得到的在 tcpdump 运行情况下基准程序 tar 的执行时间。随着 tcpdump 处理的网络流量越来越大，tar 的执行时间也越来越长。特别是系统态执行时间显著增加，从无数据包捕获到 80Mbps 的数据包，tar 所需的系统态时间增加了一倍多，而总执行时间增加不到 20%，用户态时间增加不到 10%，且并非严格随流量的增长而增加。tar 在系统态执行时间的变化是 tcpdump 竞争系统态执行时间的直接结果。

表 3 tcpdump 运行时 tar 的执行时间

| 流量 | 总时间/s | 用户态时间/s | 系统态时间/s |
|---|---|---|---|
| 0 | 154.458 | 61.049 | 7.301 |
| 800kbps | 156.575 | 59.502 | 7.608 |
| 8Mbps | 166.609 | 64.382 | 8.916 |
| 24Mbps | 168.357 | 63.631 | 9.792 |
| 40Mbps | 174.133 | 65.090 | 11.914 |
| 80Mbps | 183.464 | 65.750 | 15.668 |

### 2.4 make 工具的性能开销评测

make 是 Linux 系统上的主要编译工具，可以自动确定一个工程中需要重新编译的部分并执行编译。实验中编译的目标工程分别为开源测试工具 hbench-OS、httperf-0.8、ltp-full-20041007 以及 Linux 内核重编译，工程大小分别约为 350kB、700kB、41MB 和 187MB，表 4 是捕获的基序列和时延序列。

表 4 tar 对 make 的基序列和时延序列

| make | $t0$ | $t1$ | $t2$ | $t3$ | $t4$ | $t5$ | $t6$ |
|---|---|---|---|---|---|---|---|
| 基序列 | 0 | 3 | 16 | 56 | 61 | 70 | 155 |
| hbench-OS | 0 | 3 | 19 | 59 | 64 | 73 | 158 |
| httperf-0.8 | 0 | 3 | 20 | 60 | 65 | 74 | 159 |
| ltp-ful | 0 | 4 | 21 | 73 | 78 | 87 | 200 |
| OS kernel | 0 | 4 | 21 | 72 | 77 | 87 | 198 |

由表 4 及 $PCost(\alpha, L, \gamma)$ 的定义，计算出 make 在编译以上工程时的性能开销分别为：10%、13%、55%、58%。

make 重编译内核时，通过 top 观察系统的 CPU 和内存利用情况，发现 make 相关进程合计占用的 CPU 在 1% ~45% 之间变动，合计占用内存在十几兆左右。此时传统评测工具无法度量编译不同大小工程时的性能开销，而基序列时延度量法的 $PCost(\alpha, L, \gamma)$ 却能够反映它们之间的对应关系。在编译大型工程时，系统一方面需要频繁进行 IO 以在磁盘和内存间交换文件，另一方面编译器需要进行大量 CPU 计算和内存操作以完成编译及优化。结合其他应用的评测结果，当 $PCost(\alpha, L, \gamma)$ 的值超过 25% 后，系统性能开销较大，此时再启动其他进程时系统响应明显迟缓。

## 3 结语

基序列时延度量法通过测量基准程序生成的特征系统调用序列的运行时延，去相对地测量应用程序的性能开销。实验表明：该方法可以有效地评测应用程序在不同负载情况下运行时的性能开销，且评测结果能够反映系统当时的运行情况和被测应用程序的运行特征。下一步工作包含两方面：一是设计一个轻载的基准程序，以更准确地测量应用程序的性能；二是设计一个 API 函数库，用于在应用软件开发测试过程中设置观察点，从而可以尽早发现应用软件的性能瓶颈。

**参考文献：**

[1] BROWN AB. A decompositional approach to computer system performance evaluation[R]. Technical Report TR-03-97, Center for Research in Computing Technology, Harvard University, http://www.flashsear.net/fs/prof/papers/harvard-thesis-tr03-97.pdf, 1997.

[2] CHEN B, ENDO Y, CHAN K, *et al*. The measured performance of personal computer operating systems[A]. Proceedings of the Fifteenth Symposium on Operating System Principles[C]. Copper Mountain, Colorado: Published as ACM Operating Systems Review, SIGOPS, 1995, 29(5): 299 - 313.

[3] 汪立东，方滨兴. Linux 安全审计机制的扩展[J]. 软件学报. 2002, 13(1): 80.

[4] 汪立东. 操作系统安全评估与审计增强[D]. 哈尔滨：哈尔滨工业大学, 2002.

[5] 王佰玲，方滨兴，云晓春. 零拷贝报文捕获平台的研究[J]. 计算机学报, 2005, 28(1): 46 - 52