

# 一种基于进程执行行为分析的图形界面交互系统性能评测方法

宋 博<sup>1),2)</sup> 陈明宇<sup>1)</sup> 樊建平<sup>1)</sup>

<sup>1)</sup>(中国科学院计算技术研究所 北京 100190)

<sup>2)</sup>(中国科学院研究生院 北京 100049)

**摘 要** 传统的系统性能评测方法使用吞吐率等整体性参数作为评测手段,这类参数对于用户输入时间不确定的图形界面交互式应用程序并不适用.图形界面交互系统的评价应更侧重于考虑用户的主观感受.在多用户共享服务资源的图形界面系统中,单个用户可占用的资源受限,用户请求的处理时间可能会被延长.此时程序的“实际执行时间”,即整体执行时间与等待用户响应时间之差,能够真实地反映用户可察觉的系统处理能力.但如何提取“实际执行时间”是一个问题,文中提出了一种新的基于内核 profiling 的进程执行行为特征分析的图形界面交互系统性能评测方法,并给出了一种区间最大相关比对算法,能够从整体执行时间中准确地提取实际执行时间.为了能够在引入时空开销小的前提下获取进程执行行为,文中还设计实现了内核 trace 记录工具 Pro.对 Impress 等4个图形界面交互程序在系统内存大小不同时的性能行为进行记录和分析评测,实验结果显示了该方法的准确性和有效性.

**关键词** 实际执行时间;图形界面系统;内核 profiling;程序执行行为比对;区间最大相关算法

**中图法分类号** TP302 **DOI号**: 10.3724/SP.J.1016.2009.01393

## Evaluating GUI Systems Based on the Analysis of Process Execution Behaviors

SONG Bo<sup>1),2)</sup> CHEN Ming-Yu<sup>1)</sup> FAN Jian-Ping<sup>1)</sup>

<sup>1)</sup>(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

<sup>2)</sup>(Graduate University of Chinese Academy of Sciences, Beijing 100049)

**Abstract** Traditional methodology with throughput as a performance metric is not appropriate for evaluating interactive applications on GUI-based systems, because non-determinate user input time takes up most of the total execution time. In this case, user's perception is more important. In multi-user shared GUI systems, the resources available to a single user are limited, and the processing latency for user requests may be prolonged. So the subtraction of user input time from total time, namely the “real execution time”, can truly reflect the user-perceived performance. It is a difficult problem to extract the real executing time from the end-to-end time. This paper introduces a novel method of evaluating GUI systems based on the analysis of process execution behaviors, and presents a sectional maximum correlation algorithm, which is able to extract the real executing time accurately. In order to record kernel traces, a tool called Pro is designed and implemented, which can record pieces of trace produced during kernel profiling quickly and accurately, while introduces a quite small extra cost of time and kernel space. Four GUI-based appli-

cations on systems with different memory sizes are analyzed, and the experiment results fully demonstrate the veracity and the efficiency of this method.

**Keywords** real execution time; GUI system; kernel profiling; process behaviors contrasting; sectional maximum correlation

## 1 引 言

基准测试程序(benchmark)被广泛用于评测系统性能、识别系统瓶颈、协助系统设计等方面.传统基准测试程序<sup>[1-3]</sup>给出的评测结果一般是与吞吐率(throughput)相关的参数,如程序的整体执行时间等.对于传统的计算密集型高性能系统,这类参数可以较准确地反映系统的处理能力.但是,对于评测图形界面系统的性能,整体执行时间等评测参数就显示出其明显的局限性.

运行在图形界面系统上的交互式应用程序,如Office、Adobe Reader等,其主要特征是需要与用户进行交互,等待用户输入.用户响应时间是不确定的,因人因时而异,导致整体执行时间也不确定.人机界面研究表明对图形界面交互式系统的性能评价应更侧重于考虑用户的主观感受<sup>[4]</sup>.系统对用户请求的处理时间是这种“用户可感知”性能的量化表现形式.

所谓“实际执行时间”,是指系统实际执行应用程序所花费的时间,即从测得的整体执行时间中减去空闲并等待用户输入的时间.对于资源充分的单用户系统,实际执行时间比用户响应时间短得多.但是对于多用户共享资源的环境,如终端服务器、虚拟机、云计算、DSAG<sup>[5]</sup>等,每个用户能使用的系统资源受系统整体资源调度的限制,系统对用户请求的处理时间可能会被延长.此时考虑实际执行时间就显得十分必要.当系统对键盘和鼠标事件的响应时间分别超过150ms和195ms时,用户就能感觉到系统的处理延迟<sup>[6]</sup>.研究数据表明,用户使用计算机工作时间的38%被耗费在等待系统响应上<sup>[7]</sup>.实际执行时间代表了限定资源对单个用户造成的影响,能够真实反映系统使用有限资源处理图形界面交互式应用的能力.

目前多用户共享的处理环境已成为计算模式发展的重要趋势,共享式的图形界面系统的应用也越来越广泛.使用实际执行时间评测图形界面系统性能,能够为这类系统的设计与优化提供帮助.但是,

从最新文献检索的情况看,目前还没有一种方法能够很好地从整体执行时间中准确提取实际执行时间.

交互式应用程序由多个用户请求组成.每个请求占用的时间可以分为两部分:系统处理时间和等待用户输入时间,简称“处理时间”和“等待时间”.所有请求的处理时间之和即为实际执行时间.

获取实际执行时间的难点在于判断系统何时处理完毕用户请求,即处理时间段和等待时间段的分界点.最直接的方法是分析应用程序代码,借助profiling手段,插桩找到这个时间点.但是这种方法面临以下困难:(1)评测每个应用都要先分析源代码,非常耗时,不方便使用;(2)某些商业应用的源代码不可获取.另外,如果进程在等待时间段一直处于睡眠等待状态,就能以此为特征区分处理时间段和等待时间段.但是,对图形界面交互式应用程序的分析发现,在等待用户输入的时间段中,应用程序并不是完全处于睡眠状态,而是每隔一段随机时间就会醒来,检查是否有新的用户请求到达,如有,则结束等待时间段,转入处理用户请求;否则继续睡眠.因此,代码分析方法和简单特征判断方法均不能解决问题.

本文提出了一种新的基于内核profiling的进程执行行为特征分析方法,并给出了区间最大相关比对算法,有效地提取了实际执行时间.进程执行行为指的是进程在处理器上换入换出、睡眠、唤醒等进程调度相关行为.同一应用程序在不同系统上多次执行时,在处理时间段的执行行为是相似的;而由于等待时间的随机性,在此时间段内的行为则有较大差别.本文使用比对进程执行行为特征的方法找出相似行为和不相似行为时间段之间的分界点,作为处理时间段和等待时间段的分界点.

本文第2节简单回顾前人的相关工作;第3节阐述基于内核profiling的进程执行行为特征分析方法;第4节详细描述区间最大相关比对算法;第5节对区间最大相关比对算法的准确性进行了验证,并使用本文提出的方法提取了4个图形界面交互式应用程序Impress、Writer、Adobe Reader、Gimp在

系统内存大小不同时的实际执行时间.第6节给出结论,并对今后的工作进行了展望.

## 2 相关工作

对于图形界面交互式系统处理能力的评测,已经引起学术界越来越多的关注,国外有不少学者对此问题进行了研究.

Endo等<sup>[8]</sup>最早提出使用“单个请求延迟”,即用户从发出一个请求至看到输出结果所需的时间反映图形界面系统的处理能力.“单个延迟请求”和“实际执行时间”在本质上是一致的.图形界面应用程序包括各种各样的请求,每种请求的延迟时间都不相同,使用“单个请求延迟”作为评测参数,评测结果要分类给出各种请求的延迟时间.与“单个请求延迟”相比,实际执行时间能够从全局来考察系统对图形界面应用程序的处理能力,以更简捷的形式表示最终评测结果.

Endo<sup>[8]</sup>等提出的 idle 算法认为交互式系统大部分时间处于空闲状态.当用户请求到达时,系统转入忙碌状态,处理完用户请求后重新进入空闲状态.系统在两次空闲之间的忙碌时间就被当做单个请求延迟时间.这个方法显然不能准确给出处理时间和等待时间的分界点.首先,系统中并不是每时每刻只有一个应用程序;其次,系统处于空闲状态并不都是因为系统真正处理完用户请求,更多情况是因为等待某种资源;第三,即使系统处理完请求后,也不会总处于空闲状态,它会每隔一段时间检查一下是否有新的请求到来.

Zeldovich 等为评测 Collective 系统<sup>[9]</sup>的性能提出一种工具 VNCPlay<sup>[10]</sup>,也是使用“单个请求延迟”作为评测参数.用户在客户端执行一段图形界面的交互式操作,VNCPlay 的客户端软件把用户发出的请求和系统对每个请求的响应画面通过网络保存到服务器端.然后在不同的交互式系统上自动重放这段交互式操作,并记录每次重放时各个请求的延迟时间.VNCPlay 认为对于同一请求,每次重放时系统的响应画面应该是非常相似的.在重放时,发出一个请求后,每隔一段时间比较一次当前画面和记录的结果画面,如果相似度大于指定的阈值,则认为当前画面是此次重放时对应请求的结果画面.从发出请求至得到结果画面之间的时间就是延迟时间.通过比较同一个请求在不同系统上的延迟时间,可以比较两个系统处理图形界面负载的能力.这种方法

比 Endo 提出的解决方案要精确,但是也存在一些问题:(1)要求独占系统;(2)每次重放时都要求系统开始状态与录制时完全相同,这一点并不容易做到;(3)比较两幅画面相似程度的开销很大,而且即使得到响应画面,也不一定代表后台没有未完成的计算;(4)对键盘支持不好,对网络性能依赖较大.

为了评测基于 SWING 和 SMT 开发的 Java 应用程序的性能,Milan Jovic<sup>[11]</sup>等设计了一种基于 profiling 的工具 LiLa,通过对应用程序代码进行插桩来提取请求处理延迟.虽然这种方法的准确度较高,但可行度却很低.分析源代码相当耗时,而且商业应用程序的源代码很难获取.

另外,还有一些其它的解决方法<sup>[12-14]</sup>,它们也大多使用“单个请求延迟”来反映系统对图形界面负载的处理能力,也都没有给出有效准确的分界点定位方法.

## 3 进程执行行为分析方法

基于内核 profiling 的进程执行行为特征分析方法,简称 kernel profiling 方法,其总体思路是:(1)用进程状态的切换表示进程调度、睡眠、唤醒等行为;(2)分析 Linux 内核源代码的相关部分,找到插桩点插入记录 trace 的代码;(3)当执行到插桩点时,使用 trace 记录工具记录进程生命周期中的状态切换 trace;(4)分析 trace,获取进程调度相关行为特征,设计算法判断每个用户请求的处理时间和等待时间的分界点.

文献<sup>[15]</sup>讨论了内核源代码分析及插桩过程,在此不再赘述.内核 profiling 过程产生数据的特点是速度快,数量大.针对这个特点,在内核 profiling 过程中记录 trace 的工具应该能在保证较小时空开销的前提下,快速完整地记录这些数据.由于内核提供的消息记录函数 Printk 具有时空开销大和易丢数据的缺点<sup>[16]</sup>,不能满足记录内核 trace 的要求,本文设计并实现了一个工具 Pro,能够快速完整地记录内核 trace,并且具有较高的灵活性.

### 3.1 Trace 记录工具 Pro

Pro 是本文专为内核 profiling 开发的 trace 记录工具.它实现为一个虚拟内存字符设备的驱动程序,以模块形式加载到 Linux 操作系统中.Pro 实现的 3 个主要机制是:(1)给内核提供函数接口,实现向内核缓冲区写 trace 的功能.(2)实现内核空间与用户空间缓冲区的映射;(3)提供 trace 快速读出方式.

图 1 表示 Pro 的工作原理.对内核进行 profi-

ling 时,一旦执行到插桩点,Pro 就把产生的 trace 以二进制格式写到自己的设备空间(循环缓冲区)中.另外有一个用户进程专门负责处理 Pro 设备空间中的 trace,这里“处理”指的是读出 trace 并保存,或者实时分析等,根据 profiling 的目的而定.为了使读进程能及时读出设备空间中的数据,使用 mmap 实现了设备空间到用户空间缓冲区的映射.这样,读进程可以通过对用户空间缓冲区的操作达到读取设备空间中 trace 数据的目的.

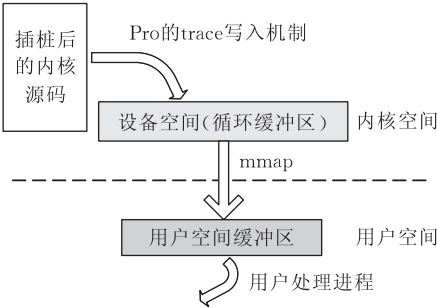


图 1 Pro 设备的工作原理

Pro 在时空开销方面比 Printk 要优越.原因在于:(1)由于内核循环缓冲区的数据是被转移到用户缓冲区而非磁盘文件保存,Pro 不需要频繁访问磁盘;(2)使用二进制格式记录 trace 也使得 Pro 占用比 Printk 少得多的磁盘空间.

Pro 能够完整地记录进程执行期间状态切换的 trace,从而准确地反应程序执行行为.以查看 CPU 占用率为例,Linux 系统中通常使用命令 top.其原理可归结为“以点代面”,即把单位时间分成若干时间片,在每个时间片内均对进程状态进行一次采样,采样时进程的忙碌或空闲状态代表其在整个时间片内对 CPU 的占用情况.单位时间内的 CPU 占用率定义为这段时间内进程占用 CPU 的时间片数与总时间片数的比值.使用采样方法获得的 CPU 占用率是不精确的,特别是当选取的单位时间粒度较小或采样频率较低时.

图 2 比较了使用内核 profiling 方法和使用 top 命令获得的某进程的 CPU 占用率曲线.图 2(a)和图 2(b)分别表示单位时间粒度为 1s 和 100ms 时,进程在第 27~32s 的 CPU 占用率曲线.图 2(a)中,由于单位时间粒度较大,使用 Pro 和 top 得到的曲线没有明显差别,尤其是采样频率为 1000 时.图 2(b)中,当采样频率为 100 时,使用 top 得到的 CPU 占用率曲线与 Pro 得到的曲线已有较大差别.图 2(c)进一步细化,表示单位时间粒度为 10ms 时,进程在第 29.2s~29.5s 的 CPU 占用率曲线.此时能明显看出 3 条曲线之间的差别.由此可以验证,使用 Pro 获取的 trace 能准确反映程序执行行为.

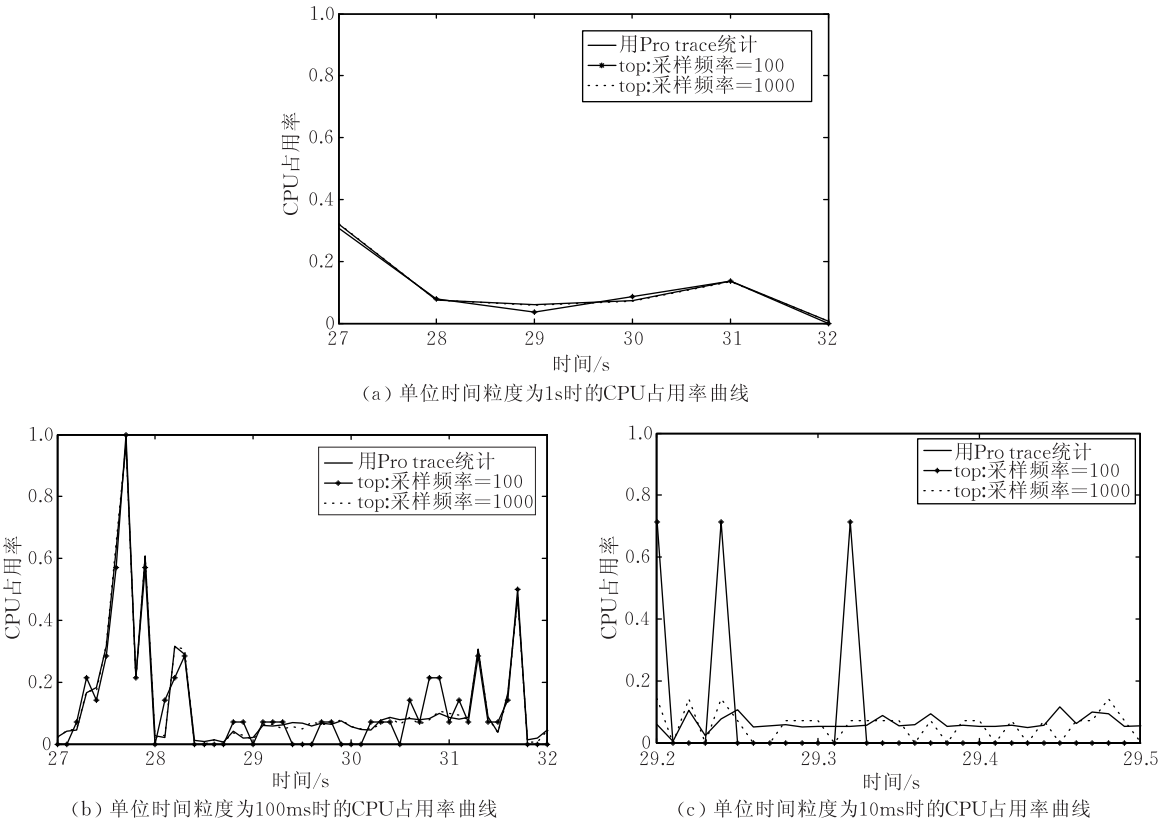


图 2 使用 Pro 和 top 得到的 CPU 占用率曲线对比

### 3.2 Trace 分析

交互式应用中,每个请求占用的时间分成处理时间和等待时间两部分.实际执行时间是所有请求的处理时间之和.

交互式进程在执行过程中会因为很多原因睡眠,主要有资源争用、磁盘访问和等待用户输入.前两者发生在处理时间段,而后者则发生在等待时间段.后者导致的睡眠特点是频率高,睡眠时间长度较随机,变化范围大,有较大值出现.

通过 Pro 获得的 trace 能够真实地表现应用程序执行行为. Trace 分析的任务是:设计算法,根据 trace 反映的执行行为特征,找出每个用户请求的处理时间段和等待时间段的分界点.

本文使用 GUITest<sup>①</sup> 代替真实用户发出请求,自动执行图形界面交互程序.由于在每个请求的等待时间段,进程会发生较长时间的睡眠, GUITest 发出一个用户请求后,跟踪进程的每次睡眠时间,如果睡眠时间超过一个极大值  $sleep\_max$ ,则 GUITest 发出下一个用户请求.根据统计数据,进程发生在处理时间段的睡眠时间远小于  $sleep\_max$ .所以可以保证进程从时间长度为  $sleep\_max$  的睡眠中醒来时,已处于等待阶段,即用户请求已经执行完毕,并等待了一段时间.  $sleep\_max$  根据具体的应用和系统环境设置.每个区间的末尾都是一段长度超过  $sleep\_max$  的睡眠,以此为依据把进程的整体执行时间划分成多个区间,每个区间代表系统对一个请求的响应过程:

(1) 处理用户请求阶段.在处理的过程中可能因为等待资源而睡眠,但睡眠时间肯定远小于极大值;

(2) 处理完毕后的等待阶段.进程在该阶段反复地随机睡眠,检查是否有新的请求.如果在该阶段中某次随机睡眠时间超过极大值,则由 GUITest 发出下一个请求信号.

Trace 分析算法多种多样.例如,借鉴并优化 Endo 等<sup>[8]</sup>提出的 idle 算法,若系统处于空闲状态的时间大于某阈值(如 100ms),则认为进入等待时间段.此算法显然较为粗糙,5.1 节的实验证明了其不准确性.本文提出一种区间最大相关比对算法,能够准确判断处理时间段和等待时间段的分界点.

## 4 区间最大相关比对算法

交互式应用程序由多个用户请求组成.每个请

求占用的时间分成处理时间和等待时间两部分.同一应用程序多次执行时,在处理时间段的执行行为是相似的;而等待时间段行为则比较随机.

### 4.1 睡眠率曲线

在应用程序执行过程中,使用 Pro 工具记录内核进程调度行为相关的 trace. 每条 trace 表示为一个六元组  $\{pid, pname, switch\_time, from, to, cpu\_no\}$ ,表示某个进程何时从某个状态切换到另一个状态.定义“睡眠率”为进程在单位时间内的睡眠时间所占比例,基于记录的 trace 可做出每个区间的进程“睡眠率”曲线.单位时间粒度根据应用程序的实际情况指定.

图 3 表示执行 impress 播放幻灯片时,处理一个 page\_down 请求所对应区间的睡眠率曲线.选取的单位时间粒度为 50ms.两条曲线分别表示在不同系统中处理同一页幻灯片对应的 page\_down 信号时的睡眠率曲线,其末尾部分均为一段时间长度超过阈值的睡眠.这段睡眠结束后,进程将收到下一个 page\_down 信号.可以看出两条曲线在分隔线前的部分是非常相似的.除了最后的睡眠阶段,在分隔线后的部分差别较大,表示等待阶段进程的行为是非常随机的.比对算法的目的就是在每个区间中找出分隔线的位置,从而提取每个区间的处理时间,即单个请求的实际执行时间.

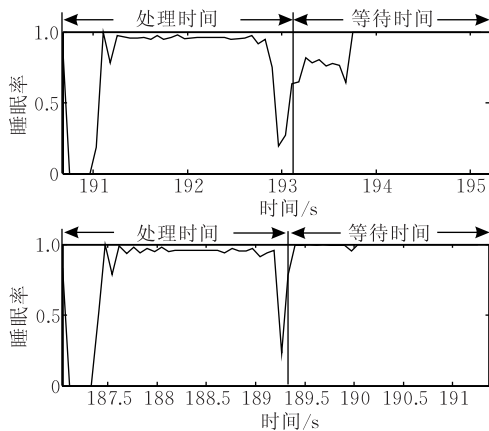


图 3 睡眠率曲线示意图

### 4.2 曲线匹配和最大相似程度 $s$

给定两条曲线  $c_1$  和  $c_2$ ,曲线匹配是指从  $c_2$  中找出与  $c_1$  相似的子曲线  $c'_2$  的过程.两段曲线相似,不仅要求对应点的值相近,而且要求变化趋势也一致.本文定义一个二元组  $(s, p)$  来描述曲线匹配的结果.

① GUITest. <http://sourceforge.net/projects/x11guitest>

引入两个向量:值向量  $x$  和变化特征向量  $a$ .  $x$  表示睡眠率曲线在每一点的取值,即该点所代表的单位时间的睡眠率; $a$  表示曲线的变化特征,每个元素的取值为  $-1$  或  $1$ . 对于  $x[1 \sim N]$  表示的曲线,向量  $a[1 \sim N]$  的元素取值定义为

$$\begin{cases} a(i) = -1, & x(i) \leq x(i-1) \\ a(i) = 1, & x(i) \geq x(i-1) \end{cases}, 1 < i \leq N \quad (1)$$

在曲线  $c_2$  中找与曲线  $c_1$  匹配的子曲线  $c'_2$  的过程如下:

1. 对于曲线  $c_1$  和  $c_2$ ,分别用  $x_1[1 \sim N_1]$  和  $x_2[1 \sim N_2]$  表示值向量,用  $a_1[1 \sim N_1]$  和  $a_2[1 \sim N_2]$  表示曲线的变化特征向量,其中  $N_1 < N_2$ . 根据式(1)计算向量  $a_1$  和  $a_2$  的值.

2. 用  $cx[1 \sim N_1 + N_2 - 1]$  表示  $x_1$  和  $x_2$  的互相关向量,  $ca[1 \sim N_1 + N_2 - 1]$  表示  $a_1$  和  $a_2$  的互相关向量,  $cx$  和  $ca$  中的元素都是归一化后的值. 即

$$cx[1 \sim N_1 + N_2 - 1] = xcorr(x_1, x_2),$$

$$ca[1 \sim N_1 + N_2 - 1] = xcorr(a_1, a_2),$$

其中,  $0 \leq cx[i] \leq 1$ ,  $0 \leq ca[i] \leq 1$ ,  $1 \leq i \leq N_1 + N_2 - 1$ .

$$\text{令 } rx[1 \sim N_2 - N_1] = cx[N_1 \sim N_2],$$

$$ra[1 \sim N_2 - N_1] = ca[N_1 \sim N_2],$$

对于  $N_1 \leq i \leq N_2$ , 用  $c'_2[i]$  表示从  $c_2$  的第  $i$  点开始, 长度为  $N_1$  的子曲线, 则  $rx[i]$  表示曲线  $c_1$  和  $c'_2[i]$  的值的相似程度,  $ra[i]$  表示变化特征的相似程度.

3. 引入向量  $rs[1 \sim N_2 - N_1]$ .

$$rs[i] = \begin{cases} (rx[i] - R_x) \times (ra[i] - R_a), & rx[i] \geq R_x \text{ 且 } ra[i] \geq R_a \\ -\infty, & \text{其它} \end{cases}$$

其中  $1 \leq i \leq N_2 - N_1$ .

$rs[i]$  综合表示了  $c_1$  与  $c_2$  中从  $i$  点开始, 长度为  $N_1$  的子曲线  $c'_2[i]$  的相似程度,  $R_x$  和  $R_a$  分别是  $rx$  和  $ra$  向量中元素的阈值, 只有满足  $rx[i] \geq R_x$  且  $ra[i] \geq R_a$ ,  $c'_2[i]$  才有与  $c_1$  匹配的可能.  $R_x$  和  $R_a$  的值根据应用程序实际情况指定.

4. 曲线  $c_1$  和  $c_2$  中各条子曲线的最大相似程度  $s$  定义为

$$s = \max(rs[i]), 1 \leq i \leq N_2 - N_1,$$

若  $s < 0$ , 则认为曲线  $c_2$  中找不到与曲线  $c_1$  匹配的子曲线; 若  $s > 0$  且  $rs[j] = \max(rs[i])$ , 即当  $i = j$  时  $rs[i]$  取得最大值, 则用二元组  $(rs[j], j)$  表示匹配结果, 表示  $c_2$  中从第  $j$  点开始, 长度为  $N_1$  的一段子曲线就是与  $c_1$  匹配的  $c'_2$ .  $c_1$  与  $c'_2$  的相似程度为  $rs[j]$ .

显然, 如果两段曲线匹配, 那么, 它们的开始点处  $x$  向量和  $a$  向量互相关函数的值都应该大于阈值. 向量  $a$  的引入是非常必要的, 如图 4 所示的上下两条曲线, 如果只比较值的相似程度, 与  $c'_1$  匹配的曲线是  $c'_2$ , 同时比较值和变化特征得到的匹配结果则是  $c'_2$ , 显然  $c'_2$  是正确的匹配.

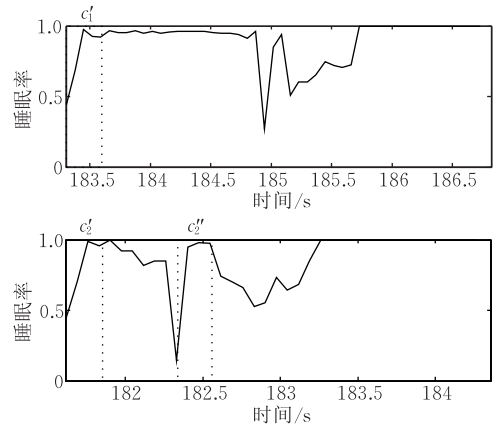


图 4 变化特征向量重要性示意图

### 4.3 比对算法

同一应用程序执行两次后, 得到两条睡眠率曲线. 把曲线划分成多个区间, 每个区间表示处理一个用户请求并等待下一个请求输入的过程. 比对算法的目标是找出各个区间中处理阶段和等待阶段的分界点. 每个区间都是以一段长度超过 `sleep_max` 的睡眠结束. 简单起见, 可以不考虑这段长时间睡眠对应的曲线部分. 如图 5(a) 所示, 要比对的是  $t_0 \sim t_{end}$  之间和  $t'_0 \sim t'_{end}$  之间的曲线.

比对算法以 3.2 节所述的曲线匹配为基础. 设两个区间的睡眠率曲线分别为  $l$  和  $l'$ , 两段曲线的开始时间点分别为  $t_0$  和  $t'_0$ , 结束时间点分别为  $t_{end}$  和  $t'_{end}$ . 比对算法流程描述如下:

1.  $t_1 = t_0$ ,  $t'_1 = t'_0$ ;  
 $t_{sep} = t_0$ ,  $t'_{sep} = t'_0$ ;
2. while ( $t_1 < t_{end}$ )  
     if ( $t_1 + \Delta t < t_{end}$ )  
          $t_2 = t_1 + \Delta t$ ;  
     else  
          $t_2 = t_{end}$ ;  
     end if  
      $l_s = t_1 \sim t_2$  之间的曲线;  $l'_s = t'_1 \sim t'_{end}$  之间的曲线;  
     if 在  $l'_s$  中能找到一个与  $l_s$  的匹配曲线  
          $t_{sep} = t_2$ ;  
          $t'_1 = l'_s$  中与  $l_s$  匹配的子曲线的结束位置;  
     end if  
      $t_1 = t_1 + \Delta t$ ;  
   end while
3.  $t'_{sep} = t'_1$ ;  
     则  $t_{sep}$  和  $t'_{sep}$  分别是曲线  $l$  和  $l'$  的分界点.

比对结果的精确度由统计睡眠率的单位时间粒度决定. 比对算法的具体过程如图 5 所示.



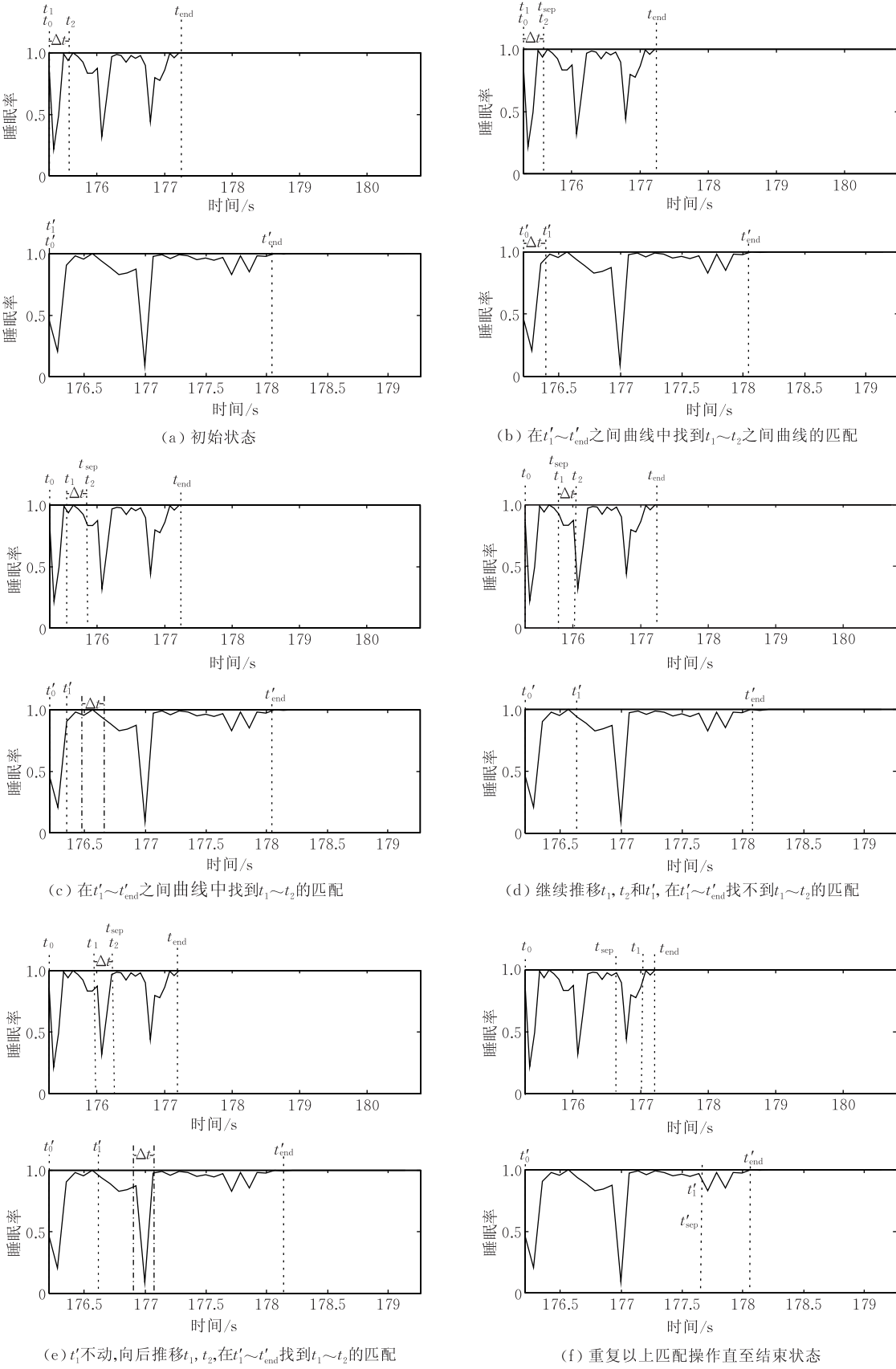


图 5 比对算法具体过程示意图

5 实验与分析

本文设计了两组实验:第 1 组实验验证区间最大相关比对算法的准确性;第 2 组实验用基于内核 profiling 的进程执行行为特征分析方法分别获取 4 个图形界面交互应用程序 Impress、Writer、Adobe Reader、Gimp 在实验机和虚拟机上的实际执行时间.

5.1 区间最大相关比对算法精度分析

3.2 节中提到,处理应用程序的每个请求占用的时间分成处理时间和等待时间两部分.处理时间段的睡眠主要是磁盘访问等操作导致的,而等待时间段的睡眠原因则是等待用户输入.

为了验证区间最大相关比对算法能够准确判断处理和等待时间段的分界点,本组实验构造了 6 个

应用程序.每个程序包括两个阶段:前一阶段除了包括依赖 CPU 的计算任务,还掺杂若干可能导致较长睡眠时间的磁盘访问操作,如从文件中读出数万行至缓冲区,把缓冲区中的数据写入指定文件等;后一阶段只包括不规则的随机睡眠.

执行 6 个应用程序,使用区间最大相关算法分析 Pro 记录的 trace 文件,判断前后两个阶段的分界点.因为分界点是预先可知的,本组实验可以判断区间最大相关算法的准确性.实验结果如表 1 所示.表 1 同时还列出了使用 Endo 等<sup>[8]</sup>提出的 idle 算法分析 trace 得到的结果.因为在等待阶段常出现较长时间的睡眠,本文优化了 idle 算法,认为若系统处于 idle 状态的时间超过某阈值,则进入等待时间段.从表 1 可以看出,区间最大相关比对算法的误差基本可控制在 10%以内,且准确性远远高于 idle 算法.

表 1 区间最大相关比对算法准确性验证结果

| 测试程序编号 | 整体执行时间/s | 前一阶段执行时间/s | 使用区间最大相关比对算法提取结果 |       | 使用 idle 算法提取结果 |       |
|--------|----------|------------|------------------|-------|----------------|-------|
|        |          |            | 前一阶段执行时间/s       | 误差/%  | 前一阶段执行时间/s     | 误差/%  |
| 1      | 16.471   | 9.777      | 9.158            | 6.33  | 4.369          | 55.31 |
| 2      | 38.947   | 23.719     | 21.832           | 7.96  | 13.253         | 44.12 |
| 3      | 45.335   | 28.211     | 29.784           | 5.58  | 11.476         | 59.32 |
| 4      | 76.238   | 38.556     | 42.438           | 10.07 | 58.924         | 52.83 |
| 5      | 128.698  | 80.480     | 88.693           | 10.21 | 26.735         | 66.78 |
| 6      | 182.579  | 141.284    | 135.826          | 3.86  | 25.679         | 81.82 |

5.2 提取应用程序的实际执行时间

本组实验用进程执行行为特征分析方法提取 4 个图形界面交互应用程序 Impress、Writer、Adobe Reader、Gimp 在实验机和虚拟机上的实际执行时间.对于每个应用程序,通过修改 grub.conf 文件限制系统可用的物理内存(在 0~1GB 之间取不同值),并挂载足够大的 swap 分区使其可以顺利执行,评测系统物理内存对应用程序执行性能造成的影响.

实验机环境为 2.0GHz Pentium IV,1GB 内存,7200rpm 160GB SATA 硬盘.操作系统内核为 Linux-2.6.14.虚拟机选用 vmware-6.5 版本,运行的操作系统内核为 Linux-2.6.14.其宿主机配置为 2 颗 4 核处理器(Quad-core 2.33GHz Intel Xeon),4G 内存,7200rpm 120G SATA 硬盘.

Impress 和 Writer 分别是 OpenOffice 软件包中的幻灯片和文档处理软件.实验中,选取一个 50MB 的幻灯片作为 impress 的输入,使用 GUITest 脚本启动 Impress 进程,打开幻灯片文件后连续播放 53 页. Writer 实验中的操作包括:启动 Writer,新建一个文档,输入 5000 个字符后,插入一幅 gif 图像,保存文档. Adobe Reader 是 Adobe 公司发布

的 Linux 环境下的 pdf 文件处理工具.实验中,启动 Adobe Reader,打开一个 pdf 文件,依次显示随机选择的 100 页. Gimp 是一个以 GPL 协议发布的图形编辑软件.对 Gimp 执行的操作是:启动 Gimp,打开一幅 jpg 图片,调整显示比例为 100%,使用 gimp 的插件改变图片的背景色后保存图片.

图 6、图 7 分别给出了 Impress 在实验机和虚拟机上执行时,对应于不同物理内存大小的整体执行时间和实际执行时间.整体执行时间是指手动执行应用程序时,从开始到结束的时间.这个时间与用户

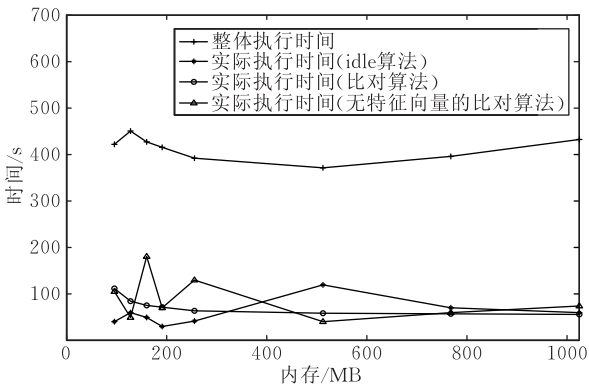


图 6 Impress 在实验机上的执行时间曲线



反应速度、操作熟练程度等主观因素有很大关系. 使用 GUITest 脚本自动执行应用程序, 执行过程中使用 Pro 记录进程调度相关的内核 trace. 分别使用 3 种不同算法分析 trace, 得到 3 种实际执行时间: (1) 参照 Endo<sup>[8]</sup>等提出的 idle 算法分析 trace 提取的实际执行时间; (2) 使用区间最大相关比对算法提取的实际执行时间; (3) 使用无特征向量的区间最大相关算法提取的实际执行时间.

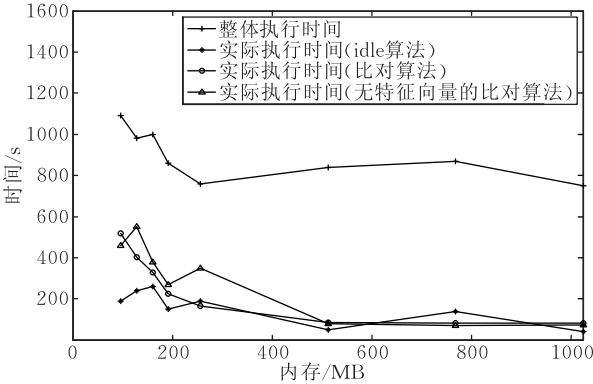


图 7 Impress 在虚拟机上的执行时间曲线

从实验结果可以看出, 整体执行时间和实际执行时间的值存在显著差异. 因为不确定的用户响应时间占据了整体执行时间的很大比例, 整体执行时间并不能反应图形界面系统的真实处理能力. 如图 6 所示, 内存为 1GB 时的整体执行时间反而比内存为 256MB 时大. 图 7 中内存从 512MB 变化到 1GB 的过程中, 实际执行时间基本不变, 说明在实验机上执行此幻灯片文件翻页时, Impress 最多需要约 512MB 内存, 此外再增加内存并不能对性能提升有明显帮助. 可见, 与整体执行时间相比, 用区间最大相关比对算法提取的实际执行时间能够准确反映系统性能, 可以作为分析识别系统瓶颈时的有效判断依据.

从图 6、图 7 可看出, 使用 idle 算法和无特征向量的比对算法得到的实际执行时间曲线都远不如比对算法得到的曲线精确. 注意到只有比对算法的曲线整体上保持了随内存容量增加的单调下降(符合直观判断)的趋势. 图 8、图 9 分别给出了 4 种应用程序在实验机和虚拟机上执行时, 使用比对算法得到的实际执行时间曲线. 比对算法单调下降的特点在这两幅图中得到充分验证. 前两种方法得到的曲线在系统内存较小时都呈现波动, 说明误差相对较大. 由此可知 idle 算法不能准确判断处理和等待两个时间段的分界点, 同时也可看出特征向量  $a$  在区间最大相关算法中的重要性.

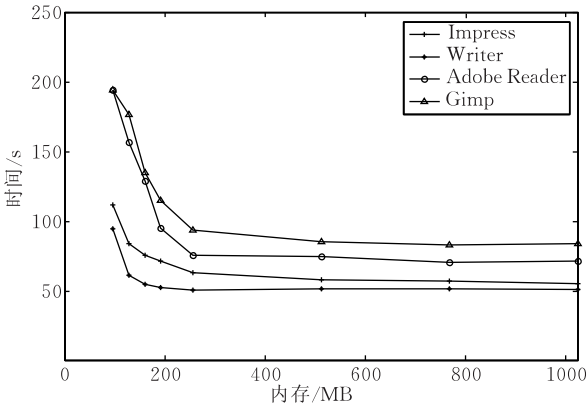


图 8 4 个应用程序在实验机上的实际执行时间曲线

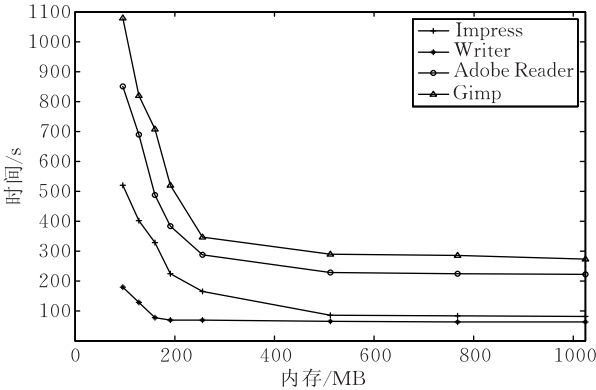


图 9 4 个应用程序在虚拟机上的实际执行时间曲线

6 总结与展望

传统的使用整体执行时间的系统性能评测方法对交互式系统性能评价具有明显不足. 基于图形界面系统的交互式应用程序, 大多数时间处于等待用户响应的状态. 排除了不确定性等待时间的“实际执行时间”能够充分代表用户可感知的性能, 从而真实反映系统对图形界面应用程序的处理能力. 应用程序的实际执行时间提取是一个挑战性的问题. 本文提出了一种新的基于内核 profiling 的进程执行行为分析的方法. 该方法采用在内核源代码中插桩, 采集进程执行过程中的状态转换 trace, 并利用所提出的区间最大相关比对算法对这些 trace 进行分析, 得到实际执行时间. 为了记录内核执行过程中产生的 trace, 设计并实现了一个能够快速记录完整的内核 trace, 并保证较小开销和较强灵活性的 profiling 工具——Pro. 实验部分以 4 个图形界面应用为例, 对 Pro 采集到的 trace 进行了分析, 得到可以反映系统真实性能的实际执行时间. 实验结果显示了该方法的准确性和有效性.

下一步的工作主要包括以下方面: (1) 利用更

多典型应用对本文算法进行进一步验证。(2)多种行为比对的融合,拟同时采用其它行为特征,如访存、I/O 等,以得到更精确的结果。(3)多用户系统下的分析方法。

## 参 考 文 献

- [1] McVoy Larry, Staelin Carl. Lmbench: Portable tools for performance analysis//Proceedings of the 1996 USENIX Technical Conference. San Diego, CA, USA, 1996: 179-294
- [2] Henning J. SPEC CPU2000: Measuring CPU performance in the new millennium. IEEE Computer, 2000, 33(7): 28-35
- [3] Brian N B, Richard P D, Alessandro F. Using microbenchmarks to evaluate system performance//Proceedings of the 3rd Workshop on Workstation Operating System. Key Biscayne, FL, USA, 1992: 148-153
- [4] Shneiderman B, Plaisant C. Designing the User Interface: Strategies for Effective Human-Computer Interface. New York: Addison Wesley, 2005
- [5] Fan Jian-Ping, Chen Ming-Yu. Dynamic self-organized computer architecture based on grid-component (DSAG). Journal of Computer Research and Development, 2003, 40(12): 1737-1742(in Chinese)  
(樊建平, 陈明宇. 网格化的动态自组织高性能计算机体系结构 DSAG. 计算机研究与发展, 2003, 40(12): 1737-1742)
- [6] James R D, Ethan V M. Is 100 milliseconds too fast?//Proceedings of the CHI'01 Extended Abstracts on Human Factors in Computing Systems. Seattle, WA, USA, 2001: 317-318
- [7] Ceaparu I, Lazar J, Bessiere K, Robinson J, Shneiderman B. Determining causes and severity of end-user frustration. International Journal of Human-Computer Interaction, 2004, 17(3): 333-356
- [8] Endo Y, Wang Zheng, Chen J B, Seltzer M. Using latency to evaluate interactive system performance//Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation. Seattle, WA, USA, 1996: 185-199
- [9] Chandra R, Zeldovich N, Sapuntzakis C, Lam M S. The Collective: A cache-based system management architecture//Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation. Boston, MA, USA, 2005: 1-11
- [10] Zeldovich N, Chandra R. Interactive performance measurement with VNCplay//Proceedings of the Annual Conference on USENIX Annual Technical Conference. Anaheim, CA, USA, 2005: 54
- [11] Milan Jovic, Matthias Hauswirth. Measuring the performance of interactive applications with listener latency profiling//Proceedings of the 6th International Symposium on Principles and Practice of Programming in Java. Modena, Italy, 2008: 137-146
- [12] Endo Yasuhiro, Seltzer Margo. Improving interactive performance using TIPME//Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. Santa Clara, CA, USA, 2000: 240-251
- [13] Nieh J, Yang S Jae, Naomi Novik. Measuring thin-client performance using slow-motion benchmarking. ACM Transactions on Computer Systems (TOCS), 2003, 21(1): 87-115
- [14] Brian K S, Monica S L, Northcutt J D. The interactive performance of SLIM: A stateless, thin-client architecture. ACM SIGOPS Operating Systems Review, 1999, 33(5): 32-47
- [15] Song Bo, Tang Huan, Chen Ming-Yu, Fan Jian-Ping. Analysis of the behaviors related to process scheduling based on kernel profiling//Proceedings of the 9th National Symposium of Graduate Students on Computer Science. Tsingtao, Shandong, 2006: 77(in Chinese)  
(宋博, 唐欢, 陈明宇, 樊建平. 基于内核 profiling 的进程行为分析//第 9 届全国研究生学术研讨会. 青岛, 山东, 2006: 77)
- [16] Jonathan Corbet, Hartman G K, Alessandro Rubini. Linux Device Drivers. 3rd Edition. Cambridge: O'Reilly Press, 2005



**SONG Bo**, born in 1983, Ph. D. candidate. Her research interests include operating system and high performance computing.

**CHEN Ming-Yu**, born in 1972, Ph. D., professor, Ph. D. supervisor. His main research interests include high performance computer architecture and parallel processing.

**FAN Jian-Ping**, born in 1963, Ph. D., professor, Ph. D. supervisor. His main research interests include high performance computing and grid computing.

Background

The traditional system evaluating methodology which focuses on throughput-related metrics is appropriate for benchmarking computing dense systems. However, it has great limitation for analyzing the performance of interactive applications on GUI-based systems, because on-determinate user response time takes up most of the total execution time. In multi-user shared GUI systems, the resources available to each user are quite limited, and the system response time for one request can be user-perceived. User interface studies indicate that a user’s perception should be the arbiter of performance.

Interests in the research of quantifying interactive performance have been really recent. Endo et al. put forward event-latency as an evaluation metric in 1996. This metric was accepted by the subsequent research and much work has been done on extracting event-processing latency from the to-

tal time, e. g. , the idle algorithm by Endo et al. and the VNCPlay tool by Zeldovic et al. However, to the best of our knowledge, there have been no methods that can do accurate extraction.

In the paper, the authors use the “real execution time”, which is the sum of all event-latency, as a parameter to evaluate how efficiently interactive applications work on GUI-based systems. The authors introduce a novel method of evaluating GUI systems based on the analysis of process executing behaviors, and present a sectional maximum correlation algorithm, which can accurately extract the real executing time from the total time. Experiment results fully demonstrate the veracity and the efficiency of this method.

This work is supported by the National Natural Science Foundation of China under grant No. 60633040 with the title of “Hyper Parallel Computer Architecture Research”.