桌面操作系统性能测试研究*)

吴俊峰 戴桂兰 白晓颖 殷人昆
(清华大学计算机科学与技术系 北京 100084)

摘 要 为了进一步推动我国桌面操作系统的发展,性能测试的研究为系统改进和市场采购提供了理论与技术指导。本文从软件性能测试的基本概念出发,探讨了操作系统的性能测试,包括影响操作系统性能的主要因素、测试方法、性能信息、性能度量方法等;重点讨论了桌面操作系统的性能测试的主要特点,探讨了具有代表性的桌面操作系统性能测试方法-基于系统吞吐量的传统测试和基于用户感知性能的测试,并比较全面地评述了相关测试技术及其工具;总结了这两种性能测试方法中存在的一些问题及相应的解决方案。

关键词 性能测试,操作系统,桌面,吞吐量,用户感知性能

The Research of Performance Testing of Desktop Operating System

WU Jun-Feng DAI Gui-Lan BAI Xiao-Ying YIN Ren-Kun
(Department of Computer Science and Technology, Tsinghua University, Beijing100084)

Abstract In order to develop our national operating system, the research of performance testing provides conceptually and technically supports to the improvement of development and marketing trading. The paper begins with the essential principle of software testing. Then, the characteristics of operating system are probed , which includ the effect of operating system performance, testing method, performance information, the method of performance measuring, etc. The emphasis is on desktop operating system, so we discuss its main characteristics and introduce the two testing methods which are based on the desktop operating system and the interacting system. They are the traditional testing which is based on the system throughput and the testing of user perceptive performance. We explain their testing technology and tools in detail, also conclude the difference between them and existent problems with their solutions.

Keywords Performance testing, Operation system, Desktop, Throughput, User perceptive performance

1 引言

性能测试是系统测试的一种,是用来测试软件在集成系统中的运行性能。性能测试关注的是系统的整体。它和通常所说的强度、压力/负载测试有密切关系。性能测试的目的在于验证系统是否达到用户提出的性能指标,同时发现系统中存在的性能瓶颈,起到优化系统的目的。

操作系统是整个计算机系统的心脏,是所有软件赖以生存的基础。作为一种通用式系统,Linux 采用基于"一体化内核(Monolithic Kernel)"的开放式的结构,在内核中提供多种可独立编译、连接、运行的服务进程,并允许不同的人参与各种服务的开发、修正、改进和完善过程。Linux 的开放源代码以及国际化兼容特性,极大地促进了国产软件,尤其是系统软件的发展,各大厂家纷纷推出各种中文 Linux 操作系统,中文操作系统标准也在酝酿和完善当中。然而,在初期发展阶段,操作系统的关注点还主要集中在功能层面,鲜有性能测试的理论和实践研究,缺乏相应的方法指导和工具支持。而从长远来看,随着应用程序对图像处理、多媒体、数据交换等能力的依赖性增强,应用程序的执行时间将在很大程度上耗费在系统层操作的开销上,操作系统的性能对应用系统的性能起着越来越重要的作用。例如,一个典型的 Web 服务器,有85%的时间为操作系统运行;多媒体、GUI 等应用软件也有20%到 90%的时间消耗在操作系统上。

正如 Amdahl 所言,操作系统性能已经成为当今应用软件性能提高的关键。Linux 操作系统性能测试也成为了基于 Linux 的国产软件能够持续稳定发展,最终成功占领市场的重要支撑。本文首先简单介绍操作系统的性能、性能指标及其测量方法。然后讨论桌面操作系统性能测试的主要特点,最后详细阐述了桌面操作系统性能测试的两种技术,即基于系统吞吐量测试和基于用户感知性能的测试。

2 操作系统性能测试

影响操作系统性能的因素主要包括:执行时间、计算速度、系统吞吐量、CPU 利用率、指令执行数、浮点运算数、网络传输速率、延迟、带宽、系统开销、服务器相应速度、图形处理能力等。其关键的性能信息主要在于:CPU、IO、信道、主要存储内存、第二存储内存的使用情况,每个指令的 IO 数量,每个模块执行时间百分比,一个模块等待 IO 完工的百分比时间,模块使用在主存储上的时间百分比,指令随时间的跟踪路径,控制从一个模块到另一个模块的次数,遇到每一组指令等待的次数,每一组指令页换入和换出的次数,系统反应时间和系统吞吐量。

操作系统的性能测量方法可以分两种:一是以没有争议的方式执行任务所花费的单位时间。例如,任务在理想工作环境下执行所需要的时间。这可以通过秒表等计时器直接进行测量。另一种是在竞争的模式下,多个任务进行操作并且

*)国家自然科学基金(项目编号:60403022)资助项目。吴俊峰 硕士研究生,研究方向:软件工程。

排队请求公共资源，例如 CPU、内存、磁盘、信道、网络等，收集竞争性的系统执行时间和资源使用情况。两种收集系统执行时间和资源使用情况的方法比较常用：

(1)监视器方法(Monitoring Approach)。通过测试工具或者操作系统的时间设备进行控制，在一定的时间间隔内，以采样的方式收集系统的状态，然后进行分析与总结，以此来检测系统的性能。采样时间间隔越短，测量越精确；(2)探针方法(Probe Approach)。通过插入探针或程序段到系统程序的内部，当系统执行时通过探针记录和收集需要的信息。探针方法收集到的结果可以用于生成报告，显示语句、模块的执行时间。

2.1 操作系统性能测试技术

操作系统性能测试主要采用基准测试(benchmark)的方法，即在标准的硬件配置和测试环境下，采用标准的安装过程安装并配置操作系统，选择典型应用及测试用例，自动检测并收集系统的性能指标(如响应时间、系统资源占用等)；统计、分析测试结果，并给出测试报告。操作系统进行基准测试需要考虑以下多方面的因素，包括：

• Operational Profile，即根据历史经验和数据，总结归纳出系统应用和运行的主要模式。例如各种操作的重要性能级、执行的频繁程度、执行的顺序等。

• Benchmark 测试集应能测试出在平均负载程度下、高负载下及极端负载情况下系统的性能状况。既需要考虑瞬间高负载，也需要在疲劳状态下不同的负载测试，如连续运行 5 天、每天 24 小时后系统的性能表现。

• 运行环境的选择，可能是针对某个特定应用的性能响应，亦可能是在复杂的多个应用同时运行时测试系统的性能表现。

操作系统性能的基准测试可分为两种：一是微观测试方法(Micro-benchmark)，针对操作系统特定核心操作性能的测试，如进入/退出内核模式的时间等；二是宏观测试方法(Macro-benchmark)，针对操作系统整体性能的测试，如文件操作等。下面分别对这两种方法做简单介绍。

2.1.1 微观测试方法

微观测试方法是针对操作系统特定核心操作性能的测试，它测量系统中一些相对简单、特定事件的性能[1]。在这种测试中，端到端性能会对结果起决定性的作用。但是，由于测试的结果会因不同的 cache 行为的影响而导致不真实。因此，微观测试程序一般不能用作准确的端到端系统性能预测。

Ousterhout 于 1990 年最早提出较为完整的微观测试集，测试内核模式进入/退出、系统调用、内存拷贝、环境切换、文件打开/关闭、文件创建/删除等特性的时间延迟。采用该测试集，Ousterhout 及其高性能计算应用研究室 WRL(Western Research Lab)对 Ultrix，SunOS，RISC/os 和 Sprite4 种操作系统进行了比较。

1995 年，Mcvoy 在发展和完善 Ousterhout 测试集的基础上，提出了一种简单的、可移植的测试集 Lmbench[2]。Lmbench 是一个用于测试 Linux 系统综合性能的多平台开源基准测试程序，着重关注带宽和延迟问题，主要度量系统在处理器、内存、缓存、网络及硬盘间数据转移的能力。与 Ousterhout 测试集相比，Lmbench 测试更加深入、细致地将操作系统原语分解为硬件相关的各个组件，并提供了工具支持，以便于理解操作系统性能与硬件体系结构之间的关系。Lmbench 不足之处在于：一是缺乏对图形特性性能测试的支持；二是测试工具集缺乏系统的测试方法的指导。

1997 年，哈佛大学的 Brown 和 Seltzer 修正了 Lmbench 中的错误，增强了测试集的统计、分析能力，使测试结果稳定可靠，提出了 HBench2OS，并应用该测试集测试了运行于 X86 机器上的 NetBSD 性能，进行了非常详细的分析[3]。

除了上述这些基准测试程序外，比较有名的基于微观测试方法的基准程序还有：Winbench，它是测试视窗环境下图形、磁盘和视像子系统的性能的基准程序；Bytemarks 套件，它测试系统的 CPU、FPU 和内存系统的能力。

微观测试方法最大的优点是它的结果容易理解和分析。因此微观测试方法可以很好地用于探查和了解系统某些特定的方面，以及用于支持和解释一个大规模的基准测试的结果。利用这种方法可以有效地测量延迟和吞吐量。但微观基准测试程序测量的是互相独立的以端到端性能作为决定因子的少量操作的性能。它可以作为一个有效的工具让用户了解到一个特定操作的性能，但不可能准确反映真实情况下的系统行为。

2.1.2 宏观测试方法

不同于微观测试方法，宏观测试方法针对的是一个完整的、非琐碎的计算。它让系统完成一系列复杂的运算，记录它们的运行时间和系统信息等。所以，宏观测试程序的结果是被执行的目标应用程序的系统吞吐量。

针对于桌面操作系统，使用宏观测试方法进行性能测试的工具有很多，其中较具有代表性的是 Business Winstone 和 Content Creation Winstone 测试集。

Business Winstone 主要针对基于 Windows 系统的桌面办公软件的基准测试集。通过对文字处理、空白表格程序、网页浏览格式化、文件压缩、反病毒扫描、电子邮件等一系列 Windows 应用程序运行测试，对系统性能进行全面的评价。整个测试根据运行完整脚本所用的时间进行打分，测试有着严格的评分标准，越高的得分代表着越好的性能。

Content Creation Winstone 与 Business Winstone 类似，也为系统级的基准测试集，它主要针对多媒体制作的性能进行测试和评价。在测试中，通过运行测试脚本，同时打开多个程序，切换并模拟实际用户进行操作，根据系统的响应时间给出性能得分。

由于宏观测试方法针对的是一个实际的应用，所以它的结果能够直接反映出这个系统性能的好坏。宏观测试方法是度量系统性能的一个很好的方法。然而，尽管它能表现一个真实的工作负载，但因为不可能模仿交互式用户的行为，所以这种方法不能准确测试桌面操作系统的性能。

3 桌面操作系统性能测试特点

桌面操作系统是典型的人机交互系统。与服务器系统相对固定的运行模式、相对稳定的服务软件相比，桌面用户需要频繁运行和终止应用软件。即使是对相同软件的操作，因时因地差异也很大，因而系统行为的可预测性差，处理事件比传统的批处理系统离散性强。从宏观的角度来讲，系统不可能根据用户当前处理的事件预测下一阶段要处理的事件；从微观角度来讲，由于事件之间的相互独立性和不可预测性导致指令级的分支预测功能、指令 Cache 的使用、指令 TLB 等 CPU 体系结构的作用发挥受到一定程度的影响。

另外，由于桌面操作系统的以下特点[4]，也使其不能采用传统的基准测试方法性能评测。

·交互性：由于是由用户来决定执行哪些程序，因此与传统的批处理应用程序相比，桌面操作系统上的程序的执行路径是不能预测的；另外，一个批处理工作负载可能执行了数百万次的时钟周期，而很多交互事件的持续时间会少于一个时钟周期。

·图形化：当传统的基准测试程序只要求把测试结果简单地输出到纯文字的文件时，桌面程序却需要画数十到数百个图形窗口，以产生一个连续的输出环境。

·多功能：传统的基准测试应用程序常常执行单一的任务，而多数桌面应用程序除主要任务外，还支持一些别的功能。这些功能的完成带来了更大的执行档且产生了大量的跨地址空间的过程调用。

4 桌面操作系统性能测试方法

当前桌面操作系统性能测试主要有两种：一是吞吐量测试，通过系统吞吐量的量化作为性能指标，表现出操作系统的性能。二是用户感知性能测试，以用户在使用桌面操作系统时感觉到的实际性能为考察对象，以此表示桌面操作系统的性能。下面分别对这两种测试方法进行介绍。

4.1 吞吐量测试

基于系统吞吐量的性能测试是通过测量系统完成一系列用户请求或工作负载所需的时间，对操作系统进行性能评价。这种方法一个主要的特色是容易测量，只要给出一个准确的计时器和一系列固定数量的任务，执行和运算即可。吞吐量测试测量了系统对重复和同步的一系列用户请求的性能。

在吞吐量测试的过程中，有时为了阻止一些不能预知的系统交互影响了测试结果，会在测试前优化测试环境，简化系统一些不确定的活动和进程，使待测系统在一个理想的条件下进行。例如关掉某些系统进程和切断网络等。但实际上，这些预先做的行为也会影响测试结果，使吞吐量基准测试变得不真实。

4.1.1 相关研究

目前很多性能测试工具和调试技术都使用了基于吞吐量的测试方法。吞吐量测试在 Windows 和 Linux 上都有不少成果。其中评估 Windows 性能的商业基准测试工具有：eTesting Labs 的 Business Winstone 系列和 Content Creation Winstone 系列；BAPCo 的 Webmark 系列和 SysMark 系列；PC World 的 PCWorldBench；MadOnion 的 3DMark 系列。

Volanomark 是一个纯 Java 的基准测试程序，专门用于测试系统调度器和线程环境的综合性能。它建立一个模拟 Client/Server 方式的 Java 聊天室，通过获取每秒平均发送的消息数来评测宿主机综合性能（数值越大性能越好）。

TPC-H 是一个由一套面向商业的 Ad-hoc 查询和并发数据修改组成的决策支持基准测试。该基准测试通过检查大量数据，执行复杂查询，以及回答商业问题来展示决策支持系统的性能。TPC-H 性能测试包括两部分：能力（Power）测试和吞吐量（Throughput）测试。能力测试将以连续的次序执行一个数据库查询流。吞吐量测试将执行多条并发的数据库查询流，每条查询流按顺序连续地执行查询。

4.1.2 吞吐量测试存在的问题

吞吐量基准测试的结果通常只是一个数字，它代表系统完成一系列事件的时间。虽然这样能有效地给出一系列事件延迟的总和，但不能反映出各个阶段响应时间的不一致，它们对于交互式系统的性能有很大的影响。

吞吐量提供的不足信息同时会误导设计员发现系统瓶颈。因为吞吐量基准测试只提供了端到端的活跃测量，它不能分辨这些系统活跃是由短延迟事件还是长延迟事件生成，后者对于交互式系统的性能有更大的影响。如果这个吞吐量基准程序内包括足够多的短延迟事件，这些事件会增加系统运行时间，于是导致设计人员针对这些对交互式系统的性能影响很少甚至没有影响的部分进行优化。

另外，图形界面都会使用一些特别的功能，例如光标的闪烁和交互式的拼写检查等，它们本身对于交互式系统性能的影响是可以忽略的。但在应用程序的整个运行过程中，可能需要进行一些复杂运算，这些特点会影响到整个计算效率。吞吐量基准测试无法分辨这些不是经常发生但对交互性能有一定影响的特色和事件。

最后，在吞吐量测试中，常常会为系统给出一个系统能接受多快就有多快的用户输入，相当于模仿一个速度无限快的用户。这样，一个输入流是不真实的，并且很容易影响到结果。这类错误其中一个是来源于批处理。在 C/S 系统中，例如 Windows NT 和 X-Window，在一条信息内批处理地要求执行多个客户请求，这时需要把它们发送到服务器。这会使系统的通信超支，服务器只能对请求流进行一些优化处理，例如移除那些由于较后提出请求而被撤消的操作。用户在现实中永远不可能生成这样的一个输入流或实现一个相同级别的批处理，系统就在这种没有实际意义的模式下运行得到了测量结果。

综上，吞吐量测试在桌面操作系统或交互式系统中常常得不到准确的结果。

4.2 用户感知性能测试

桌面操作系统和一般基于命令行的操作系统或一些用作科学计算或事务处理的系统相比，用户需要更频繁地与计算机进行交互操作。对于这种交互式系统的性能评价，更取决于用户的感觉。这种受用户的主观评价和物理因素限制的性能称之为“用户感知性能（User-perceived performance）”。

用户感知性能是指用户在使用桌面操作系统时感觉到的性能，它的指标是基于响应时间和用户的主观期待。要量化用户感知性能是非常困难的事，因为它受到用户的主观因素影响。但一般可以通过反应时间和响应时间的可变性来对用户主观的敏感程度做评价。目前，用户感知性能没有一个相对通用和明确的测试方法，还在进行各种相关的研究。

基于用户感知性能的测试相关的内容有两个：一是启动延迟；二是事件处理延迟。下面将分别对这两个内容做介绍。

4.2.1 启动延迟

在用户的角度，“启动”事件的延迟决定用户对应用软件的第一感觉。桌面用户经常需要启动软件，而服务器应用则不必要。在系统分析的角度，软件启动能比较全面地反映系统性能。由于软件启动涉及到系统的诸多环节，如目标文件大小、共享库布局、文件系统性能、内存管理算法、进程调度算法等因素，测试启动延迟有助于全面分析操作系统对桌面应用的适应性。

在测试的实用性角度，由于桌面环境以图形界面为主，窗口环境下的工作集很大，交互性能对内存的动态变化相对敏感；磁盘访问速度的提升远远落后于 CPU，内存成为影响性能的瓶颈。

启动延迟主要受初始化时软件占用内存资源的影响，其次受目标文件大小的影响。初始化占用内存越多，意味着与

内存相关的初始化工作越多，相应地存储访问指令也越多，启动延迟越大，在同等紧张的系统内存下，启动触发的 swap 操作越多，则启动延迟时间增长越快。

Linux 操作系统的 swap 操作首先由存储管理模块中的核心算法决定需要 swap 读或写的页面，然后将读写请求交给负责 I/O 操作的 swap 设备管理模块。设 T_{swap} 为页面 swap 读写的总时间，T_{mm} 为存储管理模块的处理时间开销，$T_{i/o}$ 为 swap 模块的处理时间开销，$T_{startup}$ 表示启动延迟，T_0 表示没有 swap 发生时的启动时间，则有：$T_{swap}=T_{mm}+T_{i/o}$ 以及 $T_{swap}=T_{startup}-T_0$。由这两式可以得到：$T_{mm}=T_{startup}-T_0-T_{i/o}$，那么 T_{mm} 在 swap 操作中占用的时间开销比例为：

$$e=(T_{startup}-T_0-T_{i/o})/(T_{startup}-T_0),$$

由此可以看出 $T_{i/o}$ 对开销的影响。

软件在启动时的性能不会影响到运行的流畅性，这表明各应用软件作为独立的个体本身，性能表现卓越。但由于软件启动涉及到的因素较多，包括系统层面和应用层面，任何环节间的不协调都会对启动产生负面影响。

4.2.2 事件处理延迟

桌面应用是典型的交互式应用，系统资源动态变化时的交互性能反映了桌面应用的持续运行性能，事件处理延迟是交互性能的重要表现[5]。

现在很多的应用程序的速度是依赖于系统反映异步独立流和多变的由交互式用户的输入或网络包的到达等的事件，我们这个叫做事件处理延迟。基于吞吐量的测试在这方面完全是没作用的，更具体地说，吞吐量无法提供足够的信息评估交互式系统的性能。

事件处理延时是测量交互式负载性能的一个很好的方法。它可以通过 Idle 循环指令得到。在交互系统中，CPU 很多时候都是处于空闲状态。当一个交互式事件到达时，CPU 变成忙碌；当事件处理完成时，回到空闲状态。当处理程序结束并回到空闲状态时进行记录，这样可以测量到 CPU 处理交互事件的时间以及用户等待的时间。

这里通过使用 CPU 忙碌时间表示事件延迟。但存在一个问题，CPU 忙碌时间和 CPU 空闲时间不直接等同于等待时间和思考时间。首先，即使在这些操作时 CPU 是空闲状态，同步 I/O 的请求仍然会占用等待时间。其次在后台运行时，即使 CPU 是处于忙碌状态，用户也不一定是在等待状态。

为了解决这个问题，必须考虑在系统中有多少个事件正在处理。通过监控 API 消息队列，检视在队列进行中的事件、它们加入和离开队列的时间。当用户按键或点击鼠标时，它们会在一个消息队列中排队等待处理。因此，当有事件排队时，可以假定用户是在等待中。通过对 CPU 状态（忙碌或空闲）、信息队列状态（空或非空）和同步 I/O（忙碌或空闲）的组合关系，可以推断在哪个时间间隔内用户是处在等待状态。

对于交互式系统的评测，可从人机交互方面进行分析，先作如下定义：T_{total}——事件总延迟时间；T_{user}——用户感觉到的延迟时间；T——用户对某一事件延迟的期望值；β——合适的指数值，是用户感觉的函数。于是有：

$$T_{user}=\begin{cases}0, T_{total}\leqslant T\\ (T_{total}-T)^{\beta}, T_{total}>T\end{cases}$$

对用户来说，假如一个事件的延时是低于他预期的阈值，那么这个延迟对用户来说并没有影响，甚至可能根本察觉不到。而对于同样的一个延迟的时间，用户当前不同的操作对这个延迟时间的感觉也不一样。所以上式中 T 这个期望值很难量化，因为它和人的心理因素有关，它是一个习惯于使用该系统的用户对其性能特征的一个反映。只有当 T_{total} 超过了用户的某个期望值，用户才能感知到延迟的存在，并且随着延迟的增大，用户感受到的延迟时间会“大于”实际存在的时间值，这个可以在上式中的 β 指数表现出来。

4.2.3 用户感知性能测试的监控技术

因为桌面操作系统是人机交互系统，每个事件都是在一个连续时间内发生。同时，引起性能问题的延迟可能很偶然才出现，而它们对于系统吞吐量的测试是没有任何影响的，但它们确实影响了用户感知性能。

由于这些问题并不会在固定的时间里出现，所以需要利用各种方式进行收集或采样。例如通过连续监控技术收集关于系统状态和数据储存到磁盘的相关信息。当用户遇到问题时，便能根据他们提供的信息分析和推断导致这些问题发生的原因。下面将会介绍几种适用于用户感知性能测试的监控技术。

4.2.3.1 TIPME

交互性能监控环境（The Interactive Performance Monitoring Environment，简记为 TIPME）是一种新的基准测试的基本结构，用以提升交互式系统的性能[6]。它的目标有两个：一是当用户遇到感知性能问题时，要能识别出问题出现时所在的时间间隔段；二是当性能问题出现时，要能准确地得出系统的相关状态。

TIPME 要完成这样的目标，需要用户在使用过程中，如果遇到了一个不可接受的延迟反应，立刻告知 TIPME 这个出现性能问题的进程是什么。

TIPME 记录了进程状态（进程被阻档的原因和时间）、上下文切换的信息、事件通过 X Server 运作的时间和方式、高竞争内核资源的用户等。这些信息存储在内存的一个非页面环缓冲区内，它的大小可以保持 30 到 40 秒的有价值数据，让用户有足够的时间指出性能问题的所在，并且得到相关的状态信息。

4.2.3.2 GPROF

Gprof 是标准的 GNU 性能测量工具。它是一个执行描绘器，通过分等级地描绘信息，显示测试程序的性能。Gprof 统计了程序中各个函数被调用的次数以及花费的时间，作为优化程序的依据。它让用户能剖析程序，从而知道程序的哪一个部分在执行时最费时间，并告诉用户程序里每个函数被调用的次数和每个函数执行时所占的时间百分比。

Gprof 和 TIPME 一样都是在正常环境下连续地进行监控。它们最大的不同在于测量时间间隔的方法。Gprof 的结果反映了所有活跃的、发生在目标程序的生命周期或描绘启用的时候（如果目标程序是操作系统的内核部分）。这个结果可看成是测量间隔的平均值。而 TIPME 是在特定的子区间内进行收集、选择和评估。

4.2.3.3 DCPI

DCPI（Digital Continuous Profiling Infrastructure）是一种连续监控技术，它利用连续描绘硬件的统计值，测量系统在正常执行条件下的性能。

DCPI 是由一组低开销的连续监控效率工具组成，监控对象包括所有正在执行的文件和内核。它是基于周期取样的方法，利用了 Alpha 性能计算硬件的状态信息。它会针对每一个执行文件生成图像，以便进行分析。

DCPI 和 TIPME 都利用了连续监控技术，但它们关心的

部分却不尽相同。DCPI 捕获关于硬件事件的信息，例如 cache 不命中和分枝预测错误等，而 TIPME 捕获的信息是 GUI 事件和操作系统状态的转换。由于它们关注的目标不一样，所以最终得到的结果也不一样。TIPME 着重于确认和补救操作系统中引起用户感知性能问题出现的部分，而 DCPI 的重点是得知当前硬件在系统执行时对性能的影响。

4.2.3.4 ARM

应用程序反应测量(Application Response Measurement，简记为 ARM)直接地测量了应用程序的反应时间。它是通过让客户端程序在操作前和后分别调用一次 API 函数得到反应时间的结果。现在，很多大型、内部或商业的应用程序都已利用 ARM 的 API 进行测试，而在不久的将来，更多日常用到的普通应用程序也会利用这个 API 协助进行测试。

4.2.3.5 其它研究

Endo 等人在 MS-Windows 的客户端和服务器间通过 CPU 的活跃和信息交换来推断反应时间。Cota-Robles 和 Held 也认为只靠吞吐量不足以代表计算机系统迅速处理交互式用户请求以及其他实时服务，例如视频和音频等回放的能力。他们关心的是 Windows NT 和 Windows 98 操作系统对于处理实时工作的能力，测量了这两个系统传送硬件中断到相关处理器的速度和可靠性[7]。Windows NT 和 Windows 98 有一个共享的视窗驱动模型(Windows Driver Model)。在这个模型内，传统上由中断处理器处理的计算会改由内核线程或通过延迟过程调用(Deferred Procedure Calls)完成。它们都要求操作系统进行调度决定，以及当操作系统遇到较长的中断传送延迟时要作出适当的处理。实验表明，利用吞吐量的基准测试结果不能够正确地表现出实时交互系统的性能。虽然 Windows NT 比起 Windows 98 能够提供至少一个数量级更好的实时反应，但在基于吞吐量的基准测试(Winstone benchmark)中显示的结果，这两个系统的得分相差却不超过 10 到 20 个百分比。

4.2.4 用户感知性能测试存在的问题

当前有些测试目标尽管是交互性能，但也仅仅限于应用软件的使用性能。测试在静态的系统环境下进行，这不能动态反映系统资源变化对交互性能的影响。另外，利用 CPU 空闲区间或事件集合状态间接获取延迟时间的测试手段容易受进程调度和 I/O 量大等行为复杂事件的影响，不适合测试延迟时间长的交互式事件。这些问题导致测试结果不能有效指导设计人员从最终用户的角度同时优化系统与应用。

目前，针对事件处理延迟，最大的问题在于如何确定 T 和 β 值。由于它们涉及到人的心理学，现在只能通过收集大量的用户统计得出一个平均值。用户感知性能测试最终的度量指标都会包含有用户的主观因素，只有能够找到合适的方法对这些主观点进行定义和赋值，才可以更有效和准确地对桌面操作系统和交互式系统进行性能的度量。

结束语 从桌面操作系统的性能测试技术来看，日后的重点还应放在基于吞吐量和基于用户感知性能两种测试方法的研究上。目前比较广泛使用的基准程序都是基于吞吐量测试的方法。其特点是容易进行测试，而且能有效地表现出某些系统的性能，例如科学计算或编译过程等。但缺点是它能提供的信息太少，不能反映桌面操作系统的特性并且有效地测试交互式系统的性能。而基于用户感知性能的测试，可以更真实地反映用户在使用桌面操作系统时的性能。但目前没有很完善的方法进行测试，现在主要是依靠连续监控技术和事后分析进行度量。

桌面操作系统性能测试研究现状中另一个明显不足是理论研究和工程应用相脱节。日后应该进一步改善这两种测试方法目前的不足。还有更重要的一点是结合这两种测试方法，使开发出来的性能测试基准程序能够更有效地满足应用领域的需要。

参 考 文 献

1 Chen JB, Endo Y, et al. The measured performance of personal computer operating systems. ACM Transactions on Computer Systems, 1996. 3～40

2 McVoy LW, Staelin C. Lmbench: Portable Tools for Performance Analysis. In: Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, 1996

3 Brown AB, Seltzer MI. Operating system benchmarking in the wake of lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture. In: Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Seattle, Washington, United States, 1997

4 Lee DC, Crowley PJ. Execution Characteristics of Desktop Applications on Windows NT. In: Proceedings of the 25th Annual International Symposium on Computer Architecture, Barcelona, Spain, 1998

5 Endo Y, Wang Z, et al. Using latency to evaluate interactive system performance. In: Proceedings of the Second USENIX Symposium on OperatingSystems Design and Implementation, Seattle, Washington, United States, 1996

6 Endo Y, Seltzer M. Improving Interactive System Performance Using TIPME. In: Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Santa Clara, California, United States, 2000

7 Cota-Robles E, Held JP. A comparison of Windows driver model latency performance on Windows NT and Windows 98. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, Louisiana, United States, 1999

(上接第 200 页)

器官的边缘细节信息，再利用 ICA 的方法提取出相互独立的 ICA 器官基，然后只用对应的器官基来进行图像重建，从而达到对人脸特征的定位。此方法不适用于静态和没有表情的人脸，但是对于针对存在面部表情的低比特率视频编码和表情识别非常有效。下一步的研究工作将是对 ICA 算法进行改进，提高收敛的速度，期望能够达到实时的特征定位。

参 考 文 献

1 Craw I, Tock D, Bennett. Finding face features. In: Proc. of the Second European Conference on Computer Vision, 1992. 92～96

2 Wong K H, Law H M, Tsang P W M. A system for recognizing human faces. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, 1989. 1638～1642

3 Brunelli R, Poggio T. Face recognition: features versus templates. IEEE Transaction on Pattern Analysis and Machine Intelligence, 1993, 15(10): 1042～1052

4 Vincent J M. Face finding in images. Application of Neural Network. In: Murray N F, ed., 1995. 35～70

5 Takacs B, Wechsler H. Locating facial features using SOFM. ICPR1994 B: 55～60

6 Bartlett M S, Lades H, Sejnowski T J. Independent component representations for face recognition. In: Proc. SPIE Conf. Human Vision. Electronic Imaging III, Vol 2399, 1998. 528～539

7 Mallat S. A theory of multi resolution signal decomposition: The wavelet representation. IEEE Transaction on PAMI, 1989, 11: 674～693

8 Bingham E, Hyvärine A. A fast fixed-point algorithm for independent component analysis of complex valued signals. International Journal of Neural Systems, 2000, 1: 1～8