软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2015,26(8):2074−2090 [doi: 10.13328/j.cnki.jos.004709]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 一种基于硬件计数器的虚拟机性能干扰估算方法*

王 卅[1,2,3], 张文博[1], 吴 恒[1,2,3], 宋云奎[1], 魏 峻[1,2], 钟 华[1,2], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术中心,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[3](中国科学院大学,北京 100049)

通讯作者: 王卅, E-mail: wangsa09@otcaix.iscas.ac.cn

**摘 要**: 虚拟化技术已成为云计算平台中的关键性支撑技术.它极大地提高了数据中心的资源利用率,降低了管理成本和能源消耗,但同时也为数据中心带来了新的问题——性能干扰.同一平台上的多虚拟机过度竞争某一底层硬件资源(如 CPU,Cache 等),会造成虚拟机性能严重下降;而出于安全性和可移植性的考虑,底层平台管理者需要尽量避免侵入式监测上层虚拟机,因而,如何透明而有效地从底层估算虚拟机性能干扰,成为虚拟化平台管理者必须面临的一个挑战.为应对以上挑战,提出了一种基于硬件计数器的虚拟机性能干扰估算方法.硬件计数器是程序运行期间产生的硬件事件信息(如 CPU 时间片、缓存失效次数等),已有工作主要利用大规模分布式系统任务相似性查找产生异常硬件计数器数据的节点,而没有探究硬件事件变化与性能干扰之间的直接关系.通过实验研究发现,硬件计数器(last level cache misses rates,简称 LLC misses rates)与不同资源需求的应用性能干扰存在不同的关联关系;以此建立虚拟机性能干扰估算模型,估算虚拟机性能.实验结果表明:该方法可以有效地预测 CPU 密集型应用和网络密集型应用的性能干扰大小,并仅为系统带来小于 10%的开销.

**关键词**: 云计算;虚拟化;性能干扰;硬件计数器;性能建模

**中图法分类号**: TP306

中文引用格式: 王卅,张文博,吴恒,宋云奎,魏峻,钟华,黄涛.一种基于硬件计数器的虚拟机性能干扰估算方法.软件学报,2015,26(8):2074−2090. http://www.jos.org.cn/1000-9825/4709.htm

英文引用格式: Wang S, Zhang WB, Wu H, Song YK, Wei J, Zhong H, Huang T. Approach of quantifying virtual machine performance interference based on hardware performance counter. Ruan Jian Xue Bao/Journal of Software, 2015,26(8):2074−2090 (in Chinese). http://www.jos.org.cn/1000-9825/4709.htm

## Approach of Quantifying Virtual Machine Performance Interference Based on Hardware Performance Counter

WANG Sa[1,2,3], ZHANG Wen-Bo[1], WU Heng[1,2,3], SONG Yun-Kui[1], WEI Jun[1,2], ZHONG Hua[1,2], HUANG Tao[1,2]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

**Abstract**: In IaaS platforms, hardware infrastructures are sliced into multiple virtual machines (VMs) to provide computing capabilities for users. Virtualization greatly improves the resource utilization, however it introduces potential risk of variation in VM performance. VMs co-located together have a high probability of performance degradation when one of the VMs behaves as a noisy neighbor competing hardware resource with other victims. How to efficiently monitor and quantify this type of performance interference thus becomes a key challenge for IaaS providers. To address these challenges, this study presents an approach which transparently monitors and quantifies

* 基金项目: 国家自然科学基金(61173003); 国家高技术研究发展计划(863)(2012AA011204)
收稿时间: 2014-04-28; 修改时间: 2014-07-30; 定稿时间: 2014-08-24

VMs interferences through low-level metrics with hardware performance counters (HPCs). The approach explores the information within HPC and LLC miss rates, builds performance prediction model and quantifies performance interference of different (CPU-bound and net-bound) VMs. Experimental results show that the proposed approach can predict the performance degradation effectively with an acceptable overhead that is lower than 10%.

**Key words**: cloud computing; virtualization; performance interference; hardware performance counter; performance model

虚拟化技术已经成为云计算平台[1,2]的主流支撑技术,它对底层硬件资源进行细粒度切分,以虚拟机 VM (virtual machine)的形式对外提供计算能力,多虚拟机共享同一硬件资源,极大地提高了硬件资源利用率.虚拟化技术实现了良好的功能隔离,即,每个虚拟机互不可知地共同运行在同一资源之上,但是性能隔离方面,虚拟化技术仍然存在问题.当共享资源的多虚拟机过度竞争某一底层硬件资源(如 CPU,cache 等)时,会造成虚拟机性能的严重下降,即性能干扰.例如,当两个计算密集型应用单独运行时,其数据可以有效利用 CPU cache 资源,提高计算效率;但当两个应用运行在同一硬件资源时,会造成 cache 数据反复替换,使得两个应用性能均出现下降[3−5].性能干扰会造成 SLA(service-level agreement)服务违约,从而给运营商带来不同程度的经济损失.因此,平台管理者需能有效监测不同应用的性能干扰大小,并结合各自 SLA 协议动态调整资源供给,实现整个数据中心的资源成本最小化[6−8]、效用最大化[9,10]、收益最大化[11].而本文所要解决的就是首当其冲的问题:如何快速而有效地估算虚拟机性能干扰大小.对于虚拟化平台管理者来说,估算虚拟机性能干扰面临很多挑战,主要包括以下 3 个方面:

- 不可知性:为了保证上层虚拟机的安全性和可移植性,虚拟化平台管理者对于上层应用是不可知的,也应该避免通过注入监测线程的方式获取虚拟机运行情况,因此,上层虚拟机和下层管理平台之间存在语义鸿沟(sematic gap)[12],需要在保证虚拟机安全性和完整性的同时,监测虚拟机运行情况.
- 复杂性:造成虚拟机性能干扰的原因是非常复杂的,不同层次的资源竞争都会造成虚拟机性能干扰.分时复用型资源,如 CPU 资源,产生资源竞争时会形成等待队列而导致性能下降;存储型资源,如 CPU cache 资源,产生资源竞争时会造成数据反复替换带来额外开销.不同类型资源的竞争方式也不同,因此带来的性能干扰也不相同,难以用统一的方法进行刻画.
- 实时性:对于虚拟化平台管理者来说,需要在发生性能干扰时第一时间做出相关资源调整的决策,以减少性能干扰带来的经济损失.与此同时,在线监测工具必须引入尽可能小的性能开销,在不影响上层虚拟机运行的前提下,保障所有虚拟机正常运行.

已有工作主要分离线建模[3−5,9,13−16]和在线监控[17−22]两种方式进行虚拟机性能干扰监控:离线模型通过组合测试或探针程序建立应用性能干扰模型,但受限于组合测试复杂度以及探针程序的有效性;在线监控主要通过获取底层硬件计数器(hardware performance counter)监测上层应用运行状况,硬件计数器是由硬件 PMU (performance monitoring unit)监测程序运行时产生的硬件资源使用信息.虚拟化平台管理者可以通过直接访问底层硬件 PMU 获取上层虚拟机运行产生的硬件计数器参数,避免了侵入式对虚拟机注入监测线程.硬件计数器被广泛应用于应用程序性能调优、错误追踪等领域[23,24],在虚拟化领域仍处于研究初期.已有工作主要侧重于不同虚拟机对 PMU 的复用问题(xenoprof[25],perfctr-xen[26]),而没有关注硬件计数器与上层应用性能之间的关系;而利用硬件计数器监测应用性能干扰的工作[18,19,27]主要利用分布式系统中多任务相似性的特点,将硬件计数器信息进行聚类和统计等方法分析异常节点,并没有深入分析硬件计数器信息与单节点应用性能干扰之间的直接关系.本文提出一种基于硬件计数器的虚拟机性能干扰估算方法,通过实验,发现和验证了硬件计数器(LLC misses rates)与不同类型应用性能干扰之间的相互关系.LLC misses rates 对不同类型应用性能干扰表现出不同的特征:对于 CPU 密集型应用,“受害”虚拟机**性能下降大小与其自身监测所得 LLC misses rates 强相关;对于网络密集型应用,“受害”虚拟机性能下降大小与“干扰”虚拟机监测所得 LLC misses rates 强相关.根据观测

** 性能干扰是相互的,每个虚拟机作为受害者的同时也是干扰者,本文仅站在待测虚拟机的视角将其称为“受害”虚拟机进行受干扰分析;反之,站在“干扰”虚拟机的角度,待测虚拟机即“干扰”虚拟机.

所得,对于新加入平台的应用,本文首先将其部署在沙箱系统上,针对不同类型应用分别建立性能干扰量化模型,然后将应用迁移至实际生产环境,实时收集应用运行过程中产生的硬件计数器信息,输入已有性能干扰量化模型,估算应用性能干扰情况.实验结果表明:该模型可以准确估算不同应用性能下降大小,误差不超过 5%;同时,该模型为系统引入不超过 10%的性能开销.

本文的主要贡献包括以下 3 点:(1) 发现并验证了硬件计数器(LLC misses rates)与不同类型应用性能干扰之间的相互关系;(2) 提出一种基于硬件计数器的虚拟机性能干扰估算方法;(3) 实现了该方法的原型系统并检验了该系统的有效性以及性能开销.

本文第 1 节给出问题的研究背景.第 2 节讨论如何选择预测性能干扰的硬件计数器以及硬件事件与性能之间的相关性实验.第 3 节介绍原型系统框架的主要模块以及相关步骤.第 4 节给出实验及结果分析.第 5 节比较相关工作内容.第 6 节总结本文工作内容.

## 1 研究背景

### 1.1 虚拟化技术

虚拟化技术作用于硬件设施和上层客户端操作系统之间,如图 1 所示,CPU 资源被抽象为 VCPU(virtual CPU),由 VMM(virtual machine manager)内置调度算法对每个虚拟机的 vcpu 进行调度管理.内存和硬盘则根据不同 VM 严格划分若干区域,具有较好的隔离性;而对于 I/O 设备,不同的虚拟化机制(半虚拟化和全虚拟化)以及不同的虚拟化实现(Xen,KVM 以及 VMware 等),其共享机制都不相同.本文主要关注由剑桥大学研发的虚拟化技术 Xen[28],其首先提出将 I/O 驱动切分成前端和后端两部分,每个客户虚拟机都拥有一个自己的前端驱动(frontend),负责和后端驱动(backend)进行通信,后端驱动一方面负责接收前端驱动的请求,一方面负责和底层硬件通信.对于虚拟化平台管理者来说,他们主要通过特权虚拟机 dom0 对虚拟机进行生命周期管理,每个客户虚拟机有如一个黑盒子,管理者并不清楚内部运行情况,只能通过有限的接口获取信息.

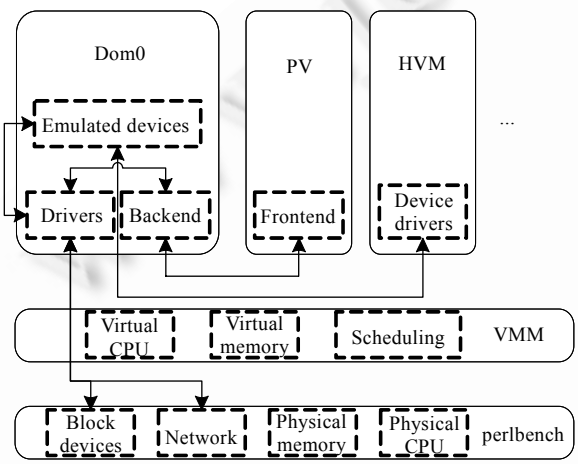



Fig.1 Xen architecture

图 1 Xen 架构

虚拟机之间严格隔离,互相不可知地运行在一起,但虚拟机性能却没有实现完全隔离,多虚拟机有可能在某一硬件资源上产生严重竞争,导致整体性能下降.如,微软的 Nathuji 在文献[3]中将 CPU 密集型应用与其他应用部署在同一硬件的不同虚拟机上,随着其他应用对 cache 资源的并发压力不断增加,CPU 密集型应用的性能不断下降.性能干扰成为虚拟化资源管理不能回避的一个关键问题.

### 1.2 硬件计数器

硬件计数器是由硬件 PMU 监视和统计所得处理器内与性能密切相关的事件发生频率,能够反映程序运行过程中出现的性能问题.硬件计数器参数种类众多,不同硬件商(Intel 和 AMD)、不同型号的硬件 PMU 能够统计的参数类型都不完全相同,表 1 给出了部分常见的硬件性能参数,分别涉及 CPU 时间片、CPU 指令以及 cache 失效等硬件信息.以 LLC_misses 为例,假设需要统计某程序运行过程中产生的 cache 失效的次数,可以设置 LLC_misses 作为统计参数,并给出参数溢出的阈值,比如 6 000,表示当程序产生 6 000 次 cache 失效时溢出,产生一次中断并计入最终结果.溢出阈值作为统计采样的频率,决定了统计的准确率以及开销大小.LLC_misses 参数

的最终统计结果反映了程序执行过程中 cache 的使用效率.

**Table 1** Hardware performance counter

**表 1** 硬件计数器

| 硬件参数名称 | 功能解释 | 统计资源类型 |
|---|---|---|
| CPU_CLK_UNHALTED | CPU clock cycles when not halted | CPU |
| INST_RETIRED | Number of instructions retired | CPU |
| BR_INST_RETIRED | Number of branch instructions retired | CPU |
| BR_MISS_PRED_RETIRED | Number of mispredicted branches retired | CPU |
| LLC_MISSES | Last level cache demand requests from this core that missed | Cache |
| LLC_REFS | Last level cache demand requests from this core | Cache |

目前,xen 虚拟化环境中统计硬件计数器参数的工具有 xenoprof[25],perfctr-xen[26]等,仍处于研究阶段.本文利用开源工具 xenoprof 并进行了简单的修改,使其能在 xen4.1 下运行并收集统计各个虚拟机运行期间产生的硬件计数器信息.

## 2 性能干扰指标

性能干扰指标是指能够有效反映虚拟机性能干扰大小的硬件计数器参数,硬件计数器参数众多且复杂,如何选择硬件计数器参数并建立该参数与虚拟机性能干扰之间的关系,是本文的关键问题.而由于监测的硬件计数器参数越多,性能开销就越大,本文主要选取一个硬件计数器参数作为性能干扰指标进行分析.虚拟机性能干扰指标的选取需要考虑以下两个问题:

- 作为性能干扰指标的硬件计数器参数应具有什么特点?

首先,该参数应该被大部分主流硬件 PMU 所支持,部分硬件计数器参数(mem_load 等)虽然能够为虚拟机运行情况提供更多、更全面的信息,但由于支持的硬件有限,不能被广泛用作性能干扰指标;其次,该参数为系统引入尽可能小的性能开销,硬件计数器所带来的性能开销取决于溢出中断的频率,对于监测 CPU 时间片和指令的参数,统计量较大,相对于其他参数来说会带来更大的性能开销.因此,尽管与 CPU 相关的参数包含丰富的运行信息,但同时也为系统带来严重的性能开销;最后,也是最重要的一点,该参数应该与虚拟机性能干扰强相关,能够有效反映或者估算虚拟机性能下降大小.

鉴于以上要求,本文选择与 CPU 缓存资源相关的硬件计数器参数 Last level cache misses rates(LLC misses rates)作为性能干扰的指标.当 CPU 发出内存访问请求时,会首先寻找 CPU 缓存是否有请求数据,如果存在,则直接返回请求数据,称为缓存命中,如果不存在,则先要把内存中的数据载入缓存当中,称为缓存失效.缓存失效的次数被许多工作[17]作为调节性能、资源调度的指标.它作为基本硬件计数器参数,几乎被所有主流硬件 PMU 所支持.其次,缓存失效是一个统计量适中的硬件计数事件,并不像其他 CPU 事件(CPU_CLK_UNHALTED)每秒都会有百万级别的统计量产生,为系统带来过多的性能开销.最后,缓存失效表现出与应用性能干扰较强的相互关系,我们将通过下一节的实验来验证 LLC misses rates 与虚拟机性能干扰之间的关系.

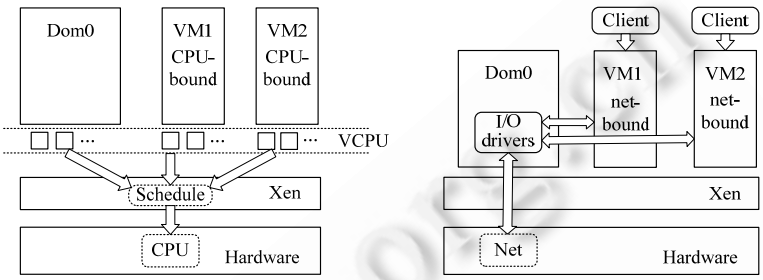
- 谁的硬件计数器参数更能反映“受害”虚拟机性能干扰?是“受害”虚拟机硬件参数,还是“干扰”虚拟机硬件参数?

直观上来讲,“受害”虚拟机的性能干扰应该与自身的硬件参数强相关,像 LLC misses rates:高缓存失效表示该虚拟机缓存数据替换频繁,从而说明虚拟机出现了性能下降;低缓存失效率,说明虚拟机缓存使用率较低或者缓存使用效率较高.但是另一方面,“干扰”虚拟机的硬件参数也能反映该虚拟机带给系统资源的压力,这个压力可用来预测受害虚拟机受到干扰的程度.因此,我们通过下一节的实验来探究谁的硬件计数器信息与“受害”虚拟机的性能干扰相关性更强.

### 2.1 实验设计

本节为研究性能干扰指标的选择问题设计实验,实验环境包括一台 DELL 主机(8 Intel Core i7-2600 3.40GHz,4GB 1333Hz DDR3,1TB 7200 RPM SATA3)、Xen 4.1,虚拟机以及 dom0 使用 linux 3.6.8 内核.每台虚拟

机分配 8 个 vcpu、1 024MB 内存以及 20GB 硬盘空间.实验架构如图 2 所示,同时部署两个虚拟机在实验环境中,每个虚拟机(标记为“VM1”和“VM2”)部署一个独立的应用.其中,我们将 VM1 作为待测虚拟机,即“受害”虚拟机,VM2 作为干扰源,实验过程中以待测虚拟机的实际运行为主,例如,当待测虚拟机程序仍在运行而干扰源运行结束,则再次启动干扰源程序直到目标虚拟机运行结束为止.




(a) CPU 密集型应用 (b) 网络密集型应用

Fig.2 Testbed

图 2 实验平台

在该实验中,我们主要观察应用 $vm_i$ 的性能下降 $D_{ij}$ 与硬件计数器参数之间的相互关系,其中,性能下降 $D_{ij}$ 是指:

$$D_{ij}=\frac{|P_{ij}-P_i|}{P_i} \tag{1}$$

$P_{ij}$ 是指 $vm_i$ 在受到 $vm_j$ 干扰后的性能,$P_i$ 是指 $vm_i$ 在单独运行过程中的性能.另外,本文使用皮尔逊相关系数(Pearson correlation coefficient)描述性能下降与硬件计数器参数之间的关系,皮尔逊相关系数的计算公式如下:

$$\rho_{X,Y}=\frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y}=\frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X\sigma_Y} \tag{2}$$

限于篇幅,本文主要关注当前云计算平台中比较流行的两种类型的应用:CPU 密集型应用和网络密集型应用.我们选取了能够代表两类应用的测试基准:SPECcpu2006[29]是由 SPEC 公司推出的一组工业级别标准的 CPU 密集型应用测试基准,被广泛应用在各项 CPU 相关测试.我们从 SPECcpu2006 的测试基准中选取了若干具有代表性的应用作为 CPU 密集型应用进行测试.Apache[30]是一个主流 HTTP 服务器,我们使用 ApacheBench 工具生成网络密集型负载访问 HTTP 服务器中的静态页面.Memcache[31]是一个分布式缓存服务器,被很多主流网站用来缓存数据库查询结果,有效地提高了网站响应速度,减轻了数据库端压力.我们使用 mem-slap 工具生成网络密集型负载获取服务器数据.TPC-W[32]是有 TPC 组织提出的工业级标准电子商务测试基准,它模拟一个在线电子书店以及用户访问电子书店搜索购买等行为模型,我们采用基于该标准的 Bench4Q[33]生成网络密集型负载访问电子书店应用.

### 2.2 CPU密集型应用

首先,我们来看 CPU 密集型应用性能干扰与 LLC misses rates 之间的关系.我们从 SPECcpu2006 中挑选了 4 个具有代表性的应用,它们分别是 mcf,lbm,libquantum 和 bzip2.其中,mcf 和 lbm 为 CPU 带来较大的缓存压力,而 libquantum 和 bzip2 带来适中的缓存压力.实验环境如图 2 所示,4 个应用先后运行于 VM1 上作为“受害”虚拟机,SPECcpu2006 中的其他应用运行于 VM2 上作为“干扰”虚拟机.我们收集 VM1 运行期间产生的 LLC misses rates 和 VM1 的性能下降情况.实验结果如图 3 所示,“受害”虚拟机的 LLC misses rates 表现出与自身的性能下降极高的相关性,相关系数高达 0.88~0.98 之间,而缓存压力较大的应用 LLC misses rates 与性能下降之间的相关性更高.我们同时收集了实验过程中“干扰”虚拟机产生的 LLC misses rates 与“受害”虚拟机性能下降之间的关系.如表 2 所示,两者之间的相关系数远小于“受害”虚拟机的 LLC misses rates 与自身性能下降之间的相关系

数.因此,对于 CPU 密集型应用,虚拟机运行期间自身产生的 LLC misses rates 能够作为性能干扰指标估算虚拟机性能下降状况.

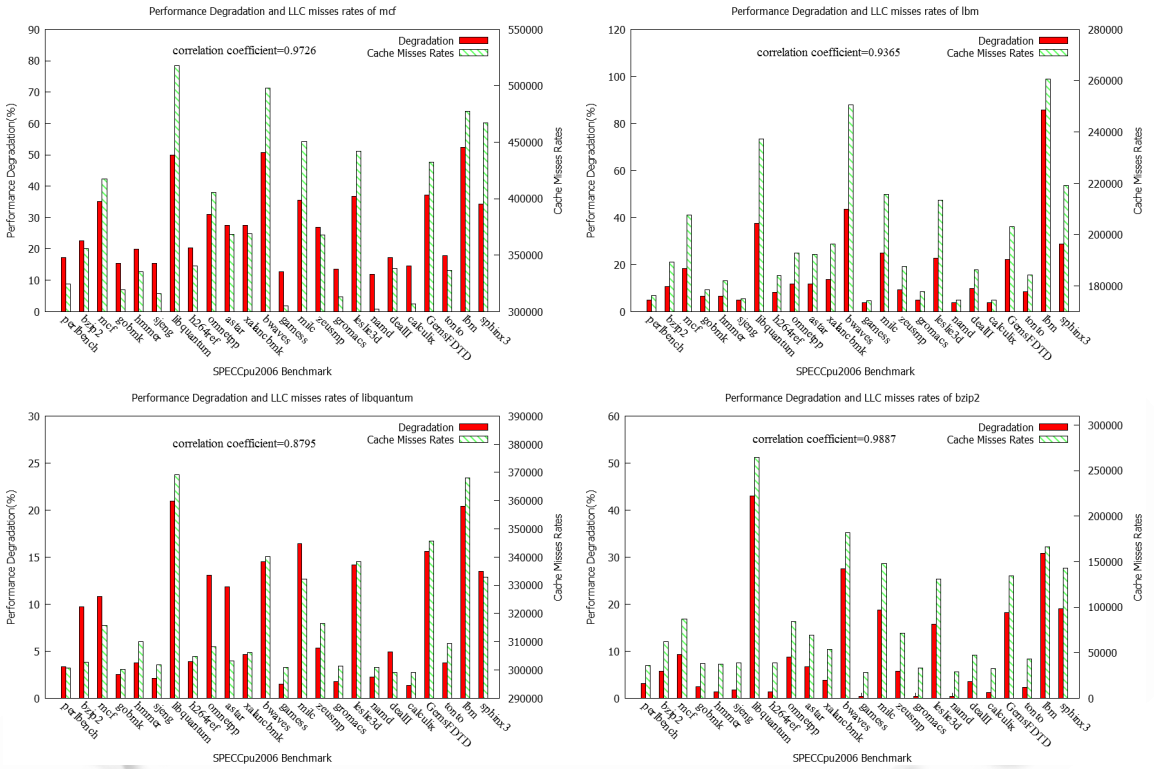



Fig.3 Performance degradation and LLC misses rates of the four workloads (mcf, lbm, libquantum, bzip2) running on VM1 when co-located with the workloads (along x-axis) running on VM2, respectively

图 3 4 个 VM1 应用(mcf,lbm,libquantum,bzip2)分别与 VM2 应用共同运行时性能下降以及 LLC misses rates

**Table 2** Correlation coefficient of victim VM performance degradation with LLC misses rates of victim VM and culprit VM

**表 2** “受害”虚拟机的性能下降与“受害”虚拟机 LLC misses rates, “干扰”虚拟机 LLC misses rates 之间的相关系数

| 测试基准 | “受害”虚拟机 | “干扰”虚拟机 |
|---|---|---|
| mcf | 0.972 6 | 0.586 4 |
| lbm | 0.932 5 | 0.473 1 |
| libquantum | 0.879 5 | 0.658 8 |
| bzip2 | 0.988 7 | 0.456 9 |

### 2.3 网络密集型应用

对于网络密集型应用,我们部署 Apache 服务器于 VM2 作为“干扰”虚拟机,如图 2 所示,由外部 client 产生并发负载由 1 到 30 的负载压力请求 Apache 服务器静态页面,同时,“受害”虚拟机 VM1 上分别部署 memcached 服务器和 TPC-W 测试基准.对于 memcached 服务器,我们使用 memslap 模拟 30 并发线程获取 100 000 条目数据;而对于 TPC-W 测试基准,我们使用 Bench4q 模拟并发用户(100EB,browsing 模式,0.07s 思考时间)访问电子书店.实验结果如图 4 所示:图 4(a)为 memcached 服务器在受到 apache 干扰的情况下的性能统计结果,图 4(b)为

TPC-W 测试基准在受到apache干扰的情况下的性能统计结果.不同于CPU密集型应用,网络密集型应用的LLC misses rates 并没有表现出与自身性能下降很强的相互关系:随着负载的变化,“受害”虚拟机的性能在不断下降,但其LLC misses rates基本保持不变.但有趣的是图中另一统计数据,“干扰”虚拟机的LLC misses rates,却代替了“受害”虚拟机,表现出与“受害”虚拟机性能下降的强相关性,分别为 0.921 6 和 0.918 4.

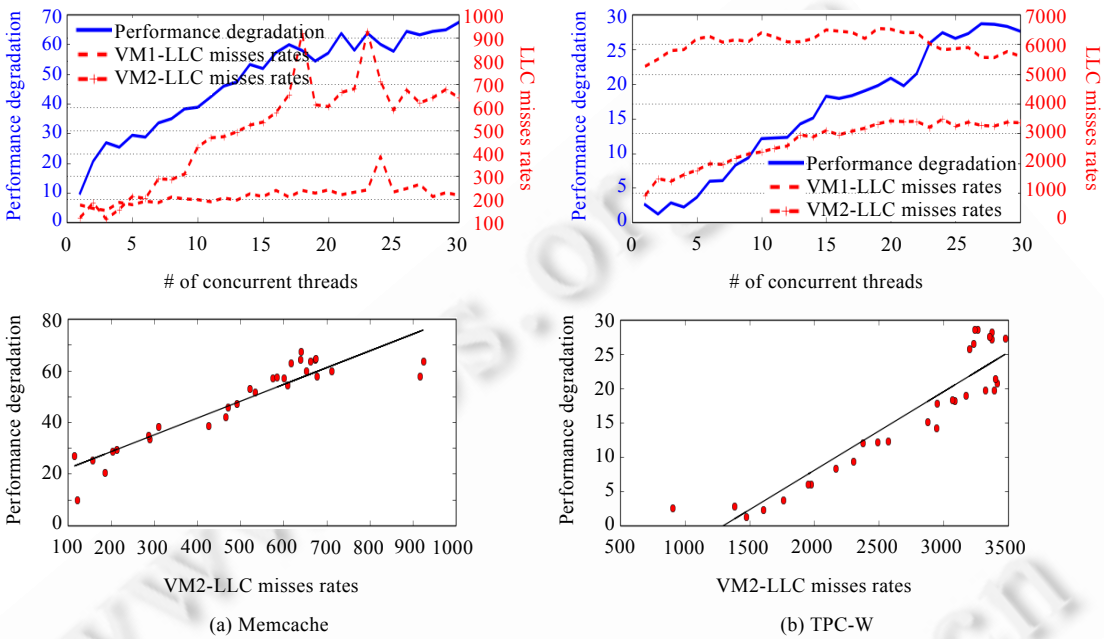



Fig 4 Performance degradation of VM1 against with LLC misses rates of VM1 and VM2 (apache); The scatter plot of performance degradation of VM1 and LLC misses rates of VM2 and fitting curve

图 4 VM1 的性能下降对比 VM1 和 VM2(apache)的 LLC misses rates; VM2 的 LLC misses rates 和 VM1 的性能下降的散点图以及拟合曲线

### 2.4 相关讨论

为什么 LLC misses rates 对于不同类型应用的性能下降表现出不同的特点?

- 对于 CPU 密集型应用,应用的运行时间主要集中在 CPU,cache,memory 和少量 I/O(程序在执行的首次,将数据一次性全部载入内存当中),发生缓存失效会使 CPU 必须从内存中读取数据,带来性能开销,而这一性能开销在 CPU 密集型应用运行时间中所占的比例较高.因此,缓存失效大小可以直接准确指示该应用的性能下降大小.
- 而对于网络密集型应用,应用的运行时间并不局限于片上,加入了网络处理开销,缓存失效带来的开销在运行时间中所占的比例降低,相关性下降,因此,应用自身的 LLC misses rates 并不能直接指示其性能下降的大小.但是对于网络密集型应用来说,每个进入系统的并发线程产生的 LLC misses rates 是相似的,随着并发负载的增加,应用产生的 LLC misses rates 统计量也线性增加,因此,LLC misses rates 可以作为应用负载压力指标.

“干扰”虚拟机的 LLC misses rates 代表该应用为系统带来的压力,所以,表现与“受害”虚拟机的性能下降强相关.

## 3 方法描述

基于上一节的实验发现,本文提出一种基于硬件计数器参数的虚拟机性能干扰量化方法,利用硬件计数器信息建立性能干扰估算模型,在不侵入虚拟机内部的前提下,监测虚拟机性能变化.图 5 给出了整体架构,本节介绍具体步骤.

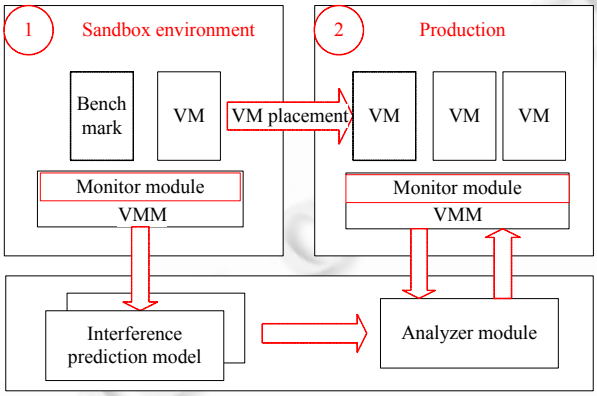



Fig 5 System architecture

图 5 系统架构

当新应用加入到虚拟化数据中心时,系统首先将应用部署在一个沙箱系统(sandbox environment),使应用与若干测试基准(benchmark)共同运行,通过监测系统(monitor system)获取运行数据,建立新应用的性能干扰量化模型(interference prediction model);然后将应用迁移到生产环境(production)中,监测系统实时收集应用运行期间产生的数据,将数据输入分析系统(analyzer system)中,结合已有的性能干扰模型监测虚拟机运行情况.当发现某虚拟机运行发生异常时,分析系统结合其他虚拟机运行状况和模糊规则对虚拟机进行调整,如虚拟机迁移.

- 对于 CPU 密集型应用,根据上一节观察的结果,应用性能干扰量与自身 LLC misses rates 的统计量线性相关,那么在沙箱环境中,只需将新应用与一组 CPU 密集型测试基准组合测试,即可建立 CPU 密集型应用的性能干扰模型.
- 对于网络密集型应用,应用性能干扰量与干扰源的 LLC misses rates 统计量线性相关,我们参考文献[13]中对干扰的定义,引入干扰度曲线和敏感度曲线.干扰度曲线是指随着负载的增加虚拟机为系统引入压力大小的度量,敏感度曲线是指虚拟机在承受一定干扰度的情况下,性能下降的度量.利用干扰度曲线和敏感度曲线为网络密集型应用建立性能干扰模型,然后将应用迁移到生产环境中,根据实时系统的总干扰度,计算不同应用的性能下降大小.

### 3.1 干扰估算模型

由于硬件计数器对不同类型应用性能干扰的特性不同,本节我们分别介绍针对 CPU 密集型应用和网络密集型应用刻画其性能干扰估算模型.而对于两类应用的性能干扰估算模型,我们均选取了线性模型:首先,第 2.2 节和第 2.3 节的实验设计与数据表明,LLC misses rates 与两种应用的性能干扰大小均有较强的线性相关性,因此,线性模型可以很好地刻画两者之间的关系;其次,作为在线监测模型,模型复杂度过高带来的计算开销会影响原有系统的运行,好的在线监测模型需要在保证原有系统正常运行的前提下,有效监测应用的性能干扰.后续的实验结果显示:该估算模型能够在保证较小性能开销的前提下,有效地监测两类应用的性能干扰大小.

- CPU 密集型应用

相对于网络密集型应用,CPU 密集型应用的性能干扰模型比较简单,性能下降与自身的 LLC misses rates 变量线性相关,我们将待测应用与一组 CPU 密集型测试基准(如 SPECcpu2006)共同运行,获取性能干扰模型.如下

所示,虚拟机 $vm_i$ 性能下降 $d_{vm_i}$ 可以表示为

$$d_{vm_i} = \alpha_{vm_i} \times R_{vm_i} \tag{3}$$

其中, $\alpha_{vm_i}$ 是相关系数, $R_{vm_i}$ 是虚拟机 $vm_i$ 的 LLC misses rates 统计量.

- 网络密集型应用

网络密集型应用的性能干扰由其他应用对资源的压力所决定,因此,我们首先需要确定每个应用为系统带来的干扰度 $P$:$\{p_1,p_2,\ldots,p_M\}$,其中,$M$ 是指共享同一硬件资源的虚拟机的个数.干扰度越大,代表该应用带给系统的压力越大.对于网络密集型应用来说,干扰度主要指该应用对网络硬件带来的压力,本文中,简单地将干扰度认定为网络吞吐量.根据上一节分析,硬件计数器参数 LLC misses rates 可以从侧面反映虚拟机中的负载大小,则虚拟机 $vm_i$ 干扰度 $p_{vm_i}$ 可以表示为

$$p_{vm_i} = \alpha_{vm_i} \times R_{vm_i} \tag{4}$$

其中, $\alpha_{vm_i}$ 是相关系数, $R_{vm_i}$ 是虚拟机 $vm_i$ 的LLC misses rates统计量.虚拟机 $vm_i$ 所承受的总干扰度 $P_{vm_i}$ 则可以表示为

$$P_{vm_i} = \sum_{j \neq i}^{M} p_{vm_j} = \sum_{j \neq i}^{M} \alpha_{vm_j} \times R_{vm_j} \tag{5}$$

其中,$M$ 是指共享硬件资源的虚拟机的个数.于是,网络密集型应用的性能下降 $d_{vm_i}$ 可以表示为

$$d_{vm_i} = \beta_{vm_i} \times P_{vm_i} \tag{6}$$

其中, $\beta_{vm_i}$ 是相关系数.

我们设计了网络密集型应用测试基准,它能够产生不同干扰度等级的负载压力,同时能够监测其他应用为系统带来的干扰度等级.首先,让待测应用依次增加其负载压力,监测测试基准感受到的待测应用干扰度,绘制待测应用干扰度曲线;然后,使测试基准依次增加其负载压力,监测待测应用在不同干扰度干扰的情况下的性能下降状况,绘制待测应用敏感度曲线.干扰度曲线和敏感度曲线共同构成待测应用的性能干扰模型.

### 3.2 监测系统

监测系统直接与硬件 PMU 进行交互,负责将配置信息写入 PMU 中,如,设置 PMU 监测硬件计数器参数为 LLC misses rates,溢出参数为 6 000.当监测开始时,监测模块(monitor module)启动 PMU 进行计数,当 LLC misses 次数累计超过 6 000 时,PMU 产生硬件中断.监测模块在收到中断后,将当前中断发生的上下文信息保存到内存中,其中包括产生此次事件的线程、虚拟机等信息,用来后期对数据进行分析统计.当监测结束后,翻译模块(translate module)负责将内存中的二进制不可读信息结合编译信息翻译成可读文本信息,统计不同虚拟机分别产生了多少硬件计数器时间.监测系统的整个执行过程如图 6 所示.

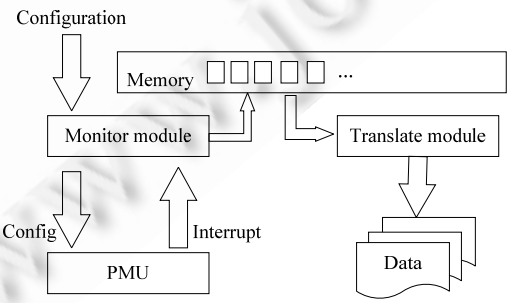



Fig 6 Architecture of monitor system

图 6 监测系统架构

我们的监测系统是在已有工具 xenoprof 修改的基础上得到的,在实际运行过程中,监测系统每分钟统计 10 秒运行信息,监测频率可以根据实际运行情况进行调整,如果在相当长的一段时间内应用都处于一个低负载状

态,则可以将监测频率降低;如果出现监测数据异常,发现虚拟机出现性能下降现象,则可以将监测频率提高,并结合其他数据,如 CPU 利用率、网络流量等信息,由分析系统决策进一步处理方案.

### 3.3 分析系统

当应用从沙箱系统迁移到生产环境中以后,由分析系统(analyzer system)接收监测系统收集到的数据,根据应用运行期间产生的数据以及已有的性能干扰模型,监测虚拟机运行情况.当虚拟机性能出现下降并低于用户预设的性能指标时,分析系统启动,收集整个系统上虚拟机的性能状况,做出决策.

对于CPU密集型应用,由于性能下降可以直接由自身的LLC misses rates计算得到,分析系统定时地将监测系统收集到的 LLC misses rates 代入性能干扰模型,获取每个虚拟机性能下降的大小,当出现虚拟机性能下降超过用户预设指标时,根据不同情况进行处理.

(1) 当系统中超过一半的虚拟机均出现性能严重下降(超过了用户预设指标)时,说明系统资源负荷过大,我们利用贪心算法的原理,将虚拟机产生的 LLC misses rates 从大到小进行排序,依次将系统中 LLC misses rates最大的虚拟机迁移到其他环境,检测其他虚拟机性能状况,直到剩下的虚拟机性能恢复正常为止.

(2) 当系统中少于一半的虚拟机出现性能严重下降时,我们依旧利用贪心算法的原理,将虚拟机按照性能下降从大到小排序,依次将性能下降最大的虚拟机迁移到其他环境,检测剩余虚拟机性能运行状况,直到剩下的虚拟机性能恢复正常为止.具体步骤见算法 1.

**算法 1**. CPU 密集型应用.

Input:

*R*: The LLC misses rates array of all CPU-bound applications on the system;

*m*: The number of CPU-bound applications on the system;

*flag*: The flag array to indicate the application performance anomaly.

```
1.  while true do
2.    UpdateMonitorData(R,flag)
3.    for each i in [1,m] do
4.      Deg=EstimatePerfDeg(R[i],D[i])
5.      if Deg>T[i] then
6.        flag[i]=true
7.      end if
8.    end for
9.  if NumOfTrue(flag)>NumOfFalse(flag) then
10.     culprit=RankOrderHPC(R)
11.     MigrateApp(culprit)
12.   else
13.     victim=ObtainVictimApp(flag)
14.     MigrateApp(victim)
15.   end if
16.   Wait(T)   //wait for T time
17. end while
```

对于网络密集型应用,虚拟机的性能下降由其他虚拟机产生的 LLC misses rates 计算得到,分析系统由每个虚拟机 LLC misses rates 代入干扰度曲线确定虚拟机对系统产生的压力 $p_{vm_i}$,然后,由此计算每个虚拟机承受的总干扰度 $P_{vm_i}$,代入虚拟机的敏感度模型中得出其性能下降的大小.当网络密集型应用性能下降超过用户预设指标时,需要对虚拟机进行调整.由于网络密集型应用性能随负载变化,一些研究认为,负载剧烈变化的应用应

该被迁移;另一些研究认为,为系统带来最大性能开销的应用应该被迁移.因此,可以根据用户定义的调度算法进行进一步的策略,其算法如下:

**算法 2**. 网络密集型应用.

Input:

*R*: The LLC misses rates array of all Net-bound applications on the system;

*m*: The number of Net-bound applications on the system.

```
1.  while true do
2.    UpdateMonitorData(R,flag,P)
3.    for each i in [1,m] do
4.      p[i]=PressureCurve(R[i])
5.      P=p[i]+P
6.    end for
7.    for each i in [1,m] do
8.      Deg=SensitivityCurve(P–p[i])
9.      if Deg>Threshold(i) then
10.       flag=true
11.       break
12.     end if
13.   end for
14.   if flag then
15.     RescheduleApp()   //user-defined schedule algorithm
16.   end if
17.   Wait(T)   //wait for T time
18. end while
```

## 4 实验分析

本节针对前文所述方法设计了一系列的实验,评价该方法对估算 CPU 密集型应用和网络密集型应用性能下降的准确性,以及该方法为系统引入的开销.实验环境如第 2.1 节所示,我们分别对 CPU 密集型应用和网络密集型应用进行干扰测试.对于 CPU 密集型应用,我们从 SPECcpu 2006 中另外挑选了 4 个具有代表性的负载:perlbench,omnetpp,milc,bwaves,将其作为待测应用进行性能监测.对于网络密集型应用,我们利用 Bench4Q 生成动态负载对待测应用电子书店进行干扰测试.

### 4.1 CPU密集型应用

我们从 SPECcpu2006 测试基准中选取 12 个应用与待测应用(perlbench,omnetpp,milc,bwaves)共同运行在沙箱系统中,建立性能预测模型,如图 7(a)~图 7(d)所示,散点图为 LLC misses rates 和性能下降之间的关系图,直线图是由散点图模拟建立的性能预测模型.然后,我们将待测应用迁移到生产环境中,将待测应用与 SPECcpu-2006 测试基准中另外 12 个应用共同运行,观察待测应用性能变化.如图 7(e)~图 7(h)所示,图中给出了实际监控待测应用性能下降大小与性能估算模型估算结果的对比图,两个结果越接近,说明估算结果越准确.从图中可以看出,性能预测模型基本准确地预测出待测应用在与不同应用共同运行时性能下降的大小,误差不超过 5%.

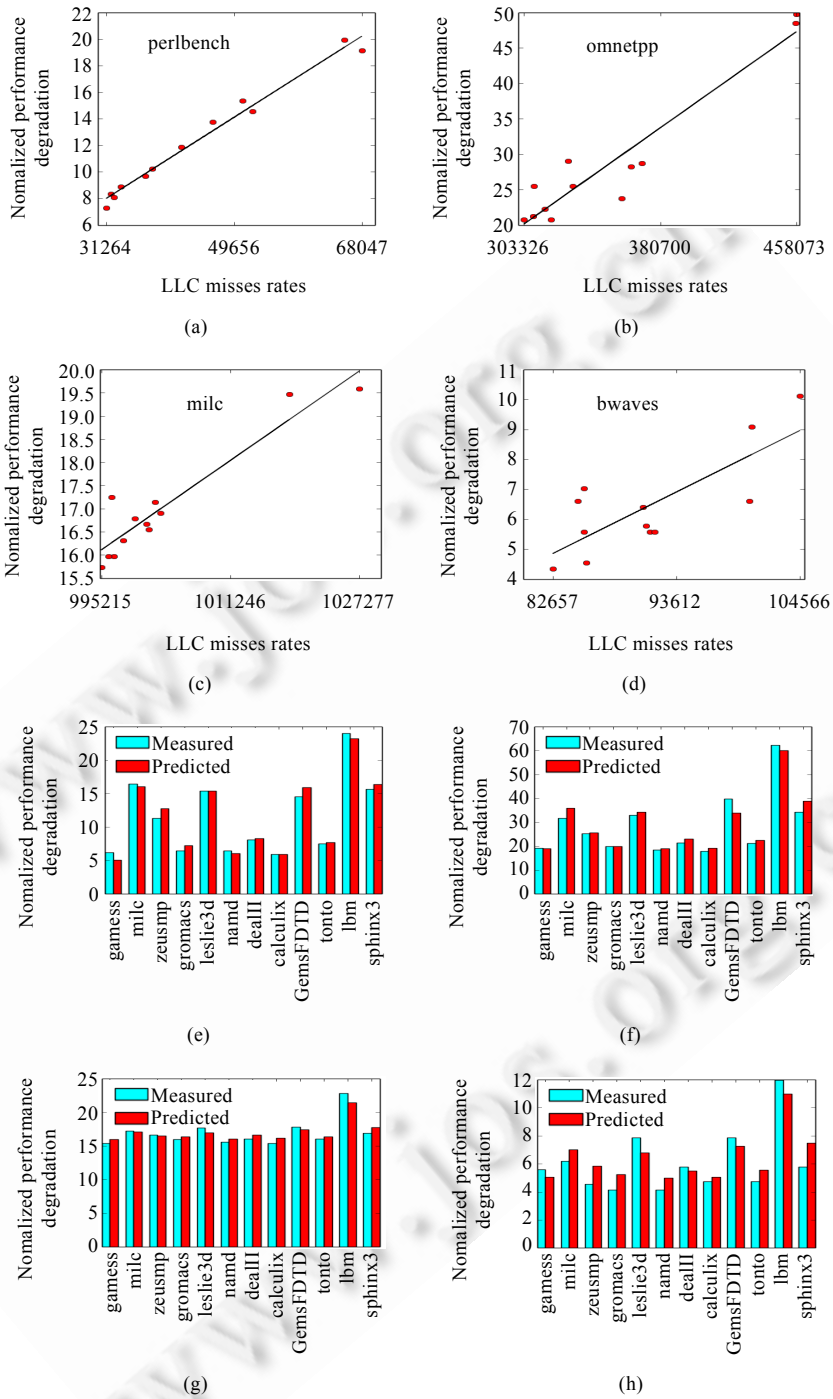


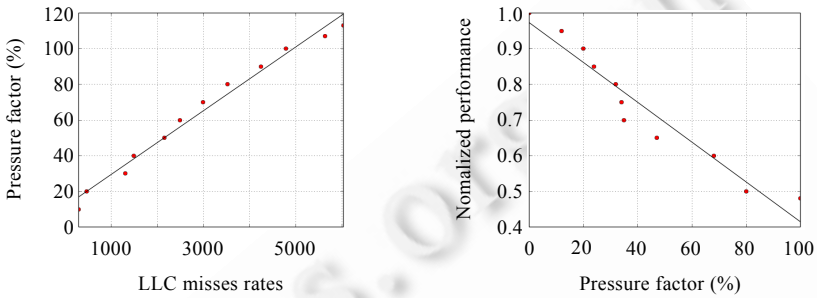

Fig.7 Interference predication models and prediction results of CPU-bound applications

图 7 CPU 密集型应用的干扰预测模型和预测结果

### 4.2 网络密集型应用

网络密集型应用的性能干扰模型包括两部分:干扰曲线和敏感曲线.如图 8 所示,我们在沙箱系统中为 Bench4Q 测试基准建立干扰曲线和敏感曲线,然后将 Bench4Q 测试基准迁移到生产环境,与 ApacheBench 生成

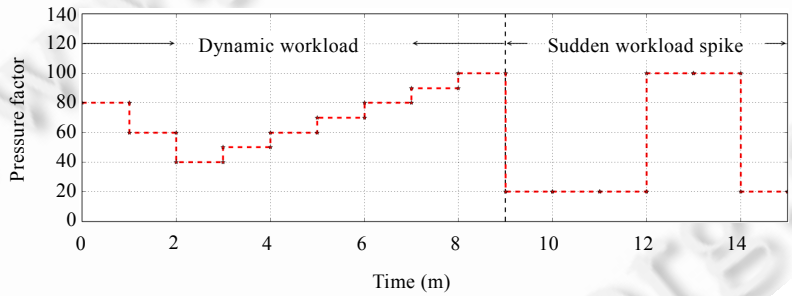
的动态负载共同运行.实验结果如图 9 所示:图 9(a)为 ApacheBench 生成的负载大小,在第 9 分钟之前系统生成的是缓慢变化的动态负载(dynamic workload),第 9 分钟之后系统生成的是瞬时负载高峰(sudden workload spike);图 9(b)为 Bench4Q 测试基准性能变化的曲线图,实线为监测到的性能大小,虚线为利用干扰曲线和敏感曲线预测得到的性能大小.可以看出,性能预测模型虽然略微滞后于监测结果,但依然较为准确地预测了性能变化,误差不超过 10%.



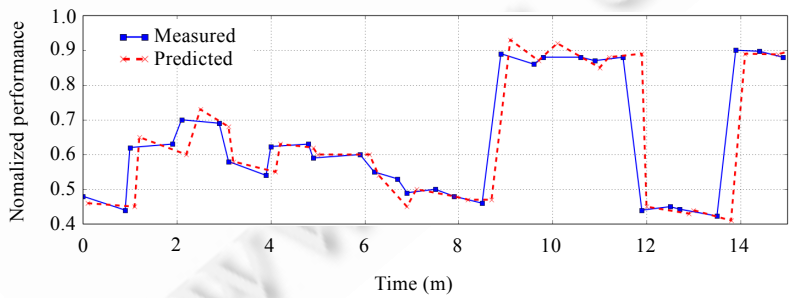

(a) 干扰曲线 (b) 敏感曲线

Fig.8 Pressure and sensitivity curve of Bench4Q benchmark

图 8 Bench4Q 测试基准的干扰曲线和敏感曲线




(a) 由 ApacheBench 产生的动态负载以及瞬时负载高峰




(b) 网络密集型应用性能预测结果

Fig.9 Generated workload and predicted performance degradation against with measured performance degradation

图 9 生成的负载和预测性能下降对比监测性能下降

### 4.3 性能开销

系统的性能开销主要来自两个方面:建立性能干扰模型和在线收集硬件计数器参数.建立性能干扰模型需

要将待测应用与不同的测试基准进行组合测试,由于该过程是在线下进行的,仅需要在应用加入系统之前测试一次,并且每次测试时间以分钟为量级,总的时间不会超过 1 个小时,因此建立性能干扰模型的性能开销在可接受范围内.所以,主要的性能开销来源于在线收集硬件计数器参数带来的开销,而该开销主要由溢出中断的频率所决定,溢出中断会打断应用运行并带来上下文切换等开销.对于 LLC misses rates,溢出中断的频率取决于应用产生缓存失效的次数,如第 2.2 节和第 2.3 节所示,CPU 密集型应用产生了远大于网络密集型应用产生的缓存失效量.本节我们主要验证一下监测系统为 CPU 密集型应用带来的开销,我们将 SPECcpu2006 测试基准集中的应用单独运行时间与在监测系统运行下的应用运行时间进行对比,如图 10 所示.图中黑色直方图为应用单独运行时间,而多出的白色部分为监测系统运行下应用的运行时间需要的额外时间.从图中可以看出,监测系统带来的性能开销均不超过 10%.因此,监测系统带来的性能开销均在可接受范围之内.

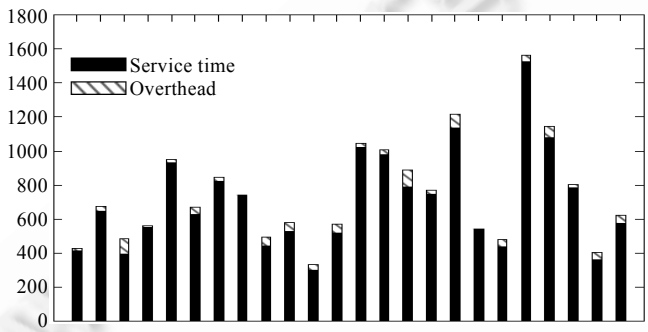



Fig.10 Performance overhead

图 10 性能开销

## 5 进一步的讨论

本方法主要针对 CPU 密集型和网络密集型这两种具有代表性应用的性能干扰问题:

- CPU 密集型应用的 CPU 利用率高,缓存失效次数直接影响应用性能大小,因此,硬件计数器 LLC misses rates 可以作为性能干扰指标估算应用性能干扰,并对 SPECcpu2006 测试基准性能下降估算表现出较高的准确率(不大于 5%的误差).
- 网络密集型应用的性能由当前系统承受的网络负载大小所决定,缓存失效对应用整体性能影响较小,但却从侧面反映应用当前并发负载大小,因此,LLC misses rate 也可以作为性能干扰指标估算其性能干扰.但由于性能并未与指标直接相关,因此在对 Bench4Q 测试基准的性能估算中,准确率略低(不大于 10%的误差).另外,LLC misses rates 指标无法分辨不同类型组合的负载,如 Bench4Q 测试基准分为 browsing,ordering,shopping 这 3 种类型的负载混合,对不同类型的负载需要分别建立模型,而混合负载模型的准确性则会大为降低.因此,我们将在随后的工作中为模型加入对负载类型的描述.

## 6 相关工作

由于共享资源而造成的虚拟机性能干扰问题,已经成为阻碍客户将其应用迁移到云计算平台上的主要因素.因此,性能干扰问题已成为产业和学术界关注的热点.已有关于性能干扰方面的工作主要分为两类:基于模型的解决方案和基于监测的解决方案.

基于模型的解决方案[3−5,9,13−16]主要通过回归模型和机器学习的方法建立性能干扰模型,预测性能干扰的大小.性能干扰模型的参数或者通过离线训练的方式确定,或者通过在线反馈的方式不断调整.I/O 竞争在虚拟化环境下非常普遍,Kundu 等人[16]通过实验发现,随机 I/O 和顺序 I/O 对性能干扰带来不同程度的影响.于是,他们在已有模型的基础上进一步细化了 I/O 参数输入,并且同时对比分析了机器学习方法中支持向量机模型和神

经网络模型对性能干扰的建模效果.Nathuji 等人[3]利用实验验证了缓存资源的竞争导致虚拟机性能下降,并建立了一个基于MIMO模型的在线反馈系统来预测应用性能.还有一些工作[4,13]试图通过刻画应用资源使用行为进行性能干扰的预测.Govindan 等人[4]和 Mars 等人[13]都尝试利用探针程序探测应用资源使用行为. Govindan 等人通过将待测应用与探针程序共同运行,模拟待测应用的缓存访问行为,然后将待测应用缓存访问行为映射到有限的缓存访问行为库,将两两测试的组合爆炸问题缩小到有限的缓存访问行为库中,有效地预测了不同应用资源竞争的性能干扰大小.Mars 等人首次提出了从 sensitivity 和 contentiousness 两个方面刻画应用对资源的使用状态,利用探针程序建立应用的两个模型,并且利用它们预测应用性能干扰.基于模型的解决方案大多需要先验知识,比如两两测试,在实际生产环境中会出现问题.应用的负载是动态变化的,离线测试的方法能够刻画的情况是有限的,而本文中,负载的变化直接反映于硬件计数器参数,能够有效刻画负载动态变化的应用.

基于监测的解决方案不需要先验知识,而是通过收集系统实时数据,判断应用运行情况.该方法的主要挑战是如何选择性能干扰指标.LLC misses rates[17],IPC(instructions per cycle)[18,19]和 memory bandwidth[20−22]均被作为性能干扰指标使用过,其中,IPC 被使用得更为广泛.Kambadur 等人[19]主要利用分布式环境中大量相似的任务并发运行,将这些相似任务的 IPC 正态分布曲线作为性能干扰指标,当收集到的在线服务 IPC 过度偏离其正态分布曲线期望值时,则认为该服务出现了性能干扰.Novakovic 等人[27]则收集不同维度的硬件计数器参数建立状态空间,利用聚类算法将应用的正常状态进行聚类,没有落入聚类的状态被认定为干扰状态.Zhang 等人[18]同样利用类似于 IPC 的 CPI 参数建立正态分布曲线,并且进一步通过对出现性能干扰的服务器上的应用 CPI 进行排序,找到最有可能干扰他人的应用进行资源限制.基于监测的解决方案需要在没有先验知识的前提下区分正常状态和异常状态.以上方法均利用分布式系统多任务相似性,将产生不同硬件计数器信息的节点认定为异常节点,而本文侧重于研究硬件计数器信息与普通单节点应用性能干扰之间的内在关系.

## 7 结束语

虚拟化技术已经迅速成为云计算平台的主流支撑技术,它有效地将硬件资源细粒度地切分,封装成虚拟机的形式提供给用户,虚拟机之间实现了良好的功能和安全隔离,但在性能隔离方面仍存在很大问题.共享同一资源的不同虚拟机会由于资源竞争造成性能干扰,如何快速而有效地监测估算虚拟机性能干扰,成为平台管理人员必须面对的一个关键问题.

本文提出一种基于硬件计数器参数的虚拟机性能干扰估算模型,通过实验我们发现,硬件计数器 LLC misses rates 对 CPU 密集型应用和网络密集型应用表现出不同的特征.基于这些发现,我们为不同应用建立各自的性能干扰估算模型;然后,将虚拟机从沙箱环境放入实验环境,通过实时监测应用运行过程中产生的数据输入性能干扰估算模型,判断虚拟机的性能干扰状况.实验结果表明:性能模型可以准确地预测虚拟机性能下降大小,误差控制在 10%以下,并且为系统带来不超过 10%的性能开销.

**References**:

[1] Amazon elastic compute cloud. 2006. http://aws.amazon.com/ec2/

[2] Windows azure platform. 2008. http://www.microsoft.com/windowsazure/

[3] Nathuji R, Kansal A, Ghaffarkhah A. Q-Clouds: Managing performance interference effects for QoS-aware clouds. In: Proc. of the 5th European Conf. on Computer Systems. ACM Press, 2010. 237−250. [doi: 10.1145/1755913.1755938]

[4] Govindan S, Liu J, Kansal A, Sivasubramaniam A. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In: Proc. of the 2nd ACM Symp. on Cloud Computing. ACM Press, 2011. 1−14. [doi: 10.1145/2038916.2038938]

[5] Zhuravlev S, Blagodurov S, Fedorova A. Addressing shared resource contention in multicore processors via scheduling. In: Proc. of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems. ACM Press, 2010. 129−142. [doi: 10.1145/1736020.1736036]

[6] Liu X, Zhu X, Singhal S, Arlitt M. Adaptive entitlement control of resource containers on shared servers. In: Proc. of 2005 the 9th IFIP/IEEE Int'l Symp. on Integrated Network Management (IM 2005). IEEE Computer Society, 2005. 163−176. [doi: 10.1109/inm.2005.1440783]

[7] Urgaonkar B, Pacifici G, Shenoy P, Spreitzer M, Tantawi A. An analytical model for multi-tier internet services and its applications. In: Proc. of the 2005 ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems. ACM Press, 2005. 291−302. [doi: 10.1145/1064212.1064252]

[8] Urgaonkar B, Shenoy P, Chandra A, Goyal P, Wood T. Agile dynamic provisioning of multi-tier Internet applications. ACM Trans. on Autonomous and Adaptive Systems, 2008,3(1):1−39. [doi: 10.1145/1342171.1342172]

[9] Jung G, Joshi K, Hiltunen M, Schlichting R, Pu C. A cost-sensitive adaptation engine for server consolidation of multitier applications. In: Proc. of the Middleware. ACM Press, 2009. 163−183. [doi: 10.1007/978-3-642-10445-9_9]

[10] Gueyoung J. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: Proc. of the IEEE 30th Int'l Conf. on Distributed Computing Systems (ICDCS). IEEE Computer Society, 2010. 62−73. [doi: 10.1109/ICDCS.2010.88]

[11] Wu H, Zhang W, Zhang J, Wei J, Huang T. A benefit-aware on-demand provisioning approach for multi-tier applications in cloud computing. Frontiers of Computer Science, 2013,7(4):459−474. [doi: 10.1007/s11704-013-2201-8]

[12] Kim H, Lim H, Jeong J, Jo H, Lee J. Task-Aware virtual machine scheduling for I/O performance. In: Proc. of the 2009 ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. ACM Press, 2009. 101−110. [doi: 10.1145/1508293.1508308]

[13] Mars J, Tang L, Hundt R, Skadron K, Soffa ML. Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In: Proc. of the 44th Annual IEEE/ACM Int'l Symp. on Microarchitecture. ACM Press, 2011. 248−259. [doi: 10.1145/ 2155620.2155650]

[14] Lim SH, Huh JS, Kim Y, Shipman GM, Das CR. D-Factor: A quantitative model of application slow-down in multi-resource shared systems. In: Proc. of the 12th ACM SIGMETRICS/PERFORMANCE Joint Int'l Conf. on Measurement and Modeling of Computer Systems. ACM Press, 2012. 271−282. [doi: 10.1145/2254756.2254790]

[15] Vasic N, Novakovic D, Miuvcin S, Kostic D, Bianchini R. DejaVu: Accelerating resource allocation in virtualized environments. In: Proc. of the 17th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM Press, 2012. 423−436. [doi: 10. 1145/2150976.2151021]

[16] Kundu S, Rangaswami R, Gulati A, Zhao M, Dutta K. Modeling virtualized applications using machine learning techniques. In: Proc. of the 8th ACM SIGPLAN/SIGOPS Conf. on Virtual Execution Environments. ACM Press, 2012. 3−14. [doi: 10.1145/2151024.2151028]

[17] Blagodurov S, Zhuravlev S, Fedorova A. Contention-Aware scheduling on multicore systems. ACM Trans. on Computer Systems, 2010,28(4):Article 8. [doi: 10.1145/1880018.1880019]

[18] Zhang X, Tune E, Hagmann R, Jnagal R, Gokhale V, Wilkes J. CPI2: CPU performance isolation for shared compute clusters. In: Proc. of the 8th ACM European Conf. on Computer Systems. ACM Press, 2013. 379−391. [doi: 10.1145/2465351.2465388]

[19] Kambadur M, Moseley T, Hank R, Kim MA. Measuring interference between live datacenter applications. In: Proc. of the Int'l Conf. on High Performance Computing, Networking, Storage and Analysis. ACM Press, 2012. 1−12. [doi: 10.1109/SC.2012.78]

[20] Mutlu O, Moscibroda T. Stall-Time fair memory access scheduling for chip multiprocessors. In: Proc. of 2007 the 40th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO 2007). ACM Press, 2007. 146−160. [doi: 10.1109/MICRO.2007.21]

[21] Nesbit KJ, Aggarwal N, Laudon J, Smith JE. Fair queuing memory systems. In: Proc. of the 39th Annual IEEE/ACM Int'l Symp. on Microarchitecture. ACM Press, 2006. 208−222. [doi: 10.1109/micro.2006.24]

[22] Kim Y, Papamichael M, Mutlu O, Harchol-Balter M. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In: Proc. of 2010 the 43rd Annual IEEE/ACM Int'l Symp. on Microarchitecture. ACM Press, 2010. 65−76. [doi: 10.1109/micro.2010.51]

[23] Arulraj J, Chang PC, Jin G, Lu S. Production-Run software failure diagnosis via hardware performance counters. In: Proc. of the 18th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM Press, 2013. 101−112. [doi: 10.1145/2451116.2451128]

[24] Yoo W, Larson K, Baugh L, Kim S, Campbell RH. ADP: Automated diagnosis of performance pathologies using hardware events. In: Proc. of the 12th ACM SIGMETRICS/PERFORMANCE Joint Int'l Conf. on Measurement and Modeling of Computer Systems. ACM Press, 2012. 283−294. [doi: 10.1145/2254756.2254791]

[25] Menon A, Santos JR, Turner Y, Janakiraman GJ, Zwaenepoel W. Diagnosing performance overheads in the xen virtual machine environment. In: Proc. of the 1st ACM/USENIX Int'l Conf. on Virtual Execution Environments. ACM Press, 2005. 13−23. [doi: 10.1145/1064979.1064984]

[26] Ruslan N, Godmar B. Perfctr-Xen: A framework for performance counter virtualization. In: Proc. of the 7th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. ACM Press, 2011. 15−26. [doi: 10.1145/1952682.1952687]

[27] Dejan N, Vasic N, Novakovic S, Kostic D, Bianchini R. DeepDive: Transparently identifying and managing performance interference in virtualized environments. In: Proc. of the 2013 USENIX Annual Technical Conf. San Jose: USENIX Association, 2013. 219−230.

[28] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proc. of the 19th ACM Symp. on Operating Systems Principles. San Jose: ACM Press, 2003. 164−177. [doi: 10.1145/945445.945462]

[29] SPEC cpu2006. 2006. http://www.spec.org/cpu2006/

[30] Apache. 1995. http://httpd.apache.org/

[31] Memcached. 2004. http://memcached.org/

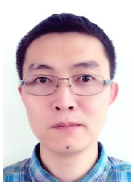
[32] TPC-W e-commerce benchmark. 2000. http://www.tpc.org/tpcw/

[33] Wenbo Z, Sa W, Wei W, Hua Z. Bench4Q: A QoS-oriented e-commerce benchmark. In: Proc. of 2011 IEEE the 35th Annual Computer Software and Applications Conf. (COMPSAC). IEEE Computer Society, 2011. 38−47. [doi: 10.1109/COMPSAC.2011.14]

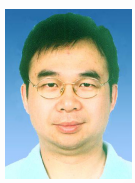
**王卅**(1986－),男,陕西渭南人,博士生,主要研究领域为网络分布式计算,软件工程.

**张文博**(1976－),男,博士,副研究员,CCF会员,主要研究领域为网络分布式计算,软件工程.

**吴恒**(1982－),男,博士,助理研究员,主要研究领域为网络分布式计算,软件工程.



**宋云奎**(1979－),男,助理研究员,主要研究领域为网络分布式计算,软件工程.



**魏峻**(1970－),男,博士,研究员,博士生导师,CCF高级会员,主要研究领域为网络分布式计算,软件工程.

**钟华**(1971－),男,博士,研究员,博士生导师,CCF高级会员,主要研究领域为网络分布式计算,软件工程.

**黄涛**(1965－),男,博士,研究员,博士生导师,CCF高级会员,主要研究领域为网络分布式计算,软件工程.