基于任务行为分析的 DVFS 机制①

陈 云[1,2], 贾刚勇[1,2], 李 曦[1,2], 张海鹏[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230026)
[2](中国科学技术大学 苏州研究院, 苏州 215028)

摘 要: 动态电压频率缩放(DVFS)技术是当前最有效的功耗调节手段之一. 本文首先分析现有 DVFS 技术存在的不足, 指出限制 DVFS 技术高效运用的核心因素; 基于现有低效的方式我们提出一种基于任务行为分析的 DVFS 机制(TC-DVFS).其具有三个层次: 一、采集任务的系统调用信息; 二、识别任务的关键系统调用, 并以关键系统调用刻画任务行为; 三、根据任务行为构建特征库, 并以任务的特征库来指导 DVFS. 我们将 TC-DVFS 添加到 linux 内核中, 并在 intel-core2 处理器平台上对不同类型的应用任务进行性能与功耗测试. 结果显示 TC-DVFS 总体获得 10%的性能提升, 并降低 5%调频失效率和 5%的系统能耗.

关键词: DVFS; 任务行为分析; 功耗优化; 系统调用

Task Behavior Based on DVFS Mechanism

CHEN Yun[1,2], JIA Gang-Yong[1,2], LI Xi[1,2], ZHANG Hai-Peng[1,2]

[1](Institute of Computer Science and Technology, The University of Science and Technology of China, Hefei 230026, China)
[2](Suzhou Institute for Advanced Study, The University of Science and Technology of China, Suzhou 215028, China)

Abstract: The dynamic voltage and frequency scaling (DVFS) is one of the most effective mechanism for power management. Firstly, we point out the most shortcomings of the current DVFS mechanisms after detailly analyzing; so we propose a task behavior based DVFS mechanism (TC-DVFS). TC-DVFS has three levels: First, gather the system call information of the task. Second, distinguish the critical system calls and characterize task behavior with them; Third, construct a feature database based on task behavior and guide DVFS according the database. Adding TC-DVFS into linux kernel, experimental results show TC-DVFS reduce 5% miss rate of the correct frequency scaling and 5% power of the system, and also improve 10% performance of the system.

Key words: DVFS; task behavior analysis; power; system call

伴随着信息时代的高速发展, 智能手机的出现, 处理器结构正逐渐趋向于多核时代, 软件设计也朝着多线程并行设计发展. 在这些追求高性能的研究背后, 如何在保证性能的前提下降低功耗已成为日益重要的课题. 动态的硬件状态调节技术是当前主要的控制手段, 其中包括 CPU 状态的切换, 内存 bank 状态调整, I/O 设备的开关等等. 这其中在 CPU 的状态控制上, 动态电压频率缩放(DVFS)技术和门控时钟技术是两种比较常用的手段[1]. DVFS 是指根据系统的当前负载信息动态的对 CPU 的频率和电压进行调整.

当前主流 linux 内核提供了 5 种可选的 CPU 控制框架: performance, powersave, userspace, ondemand, conservative[1]. 其中, performance 和 powersave 模式分别将 CPU 主频设置为最高值和最低值; userspace 模式将工作交给用户态应用程序来控制; ondemand 模式则基于负载的变化动态的调整频率; conservative 模式较 ondemand 模式更加温和一点, CPU 频率根据系统负荷在可用频率中逐渐变化, 适合于以电池供电的设备. 作为当前内核中最常用的 cpu 工作模式, ondemand 根据当前的 cpu 利用率动态的调整 cpu 频率. 它包含 3 个重要

① 基金项目:国家自然基金(61272131,61202053);江苏省自然科学基金(SBK201240198);江苏江苏省产学研前瞻性联合研究项目(BY2009128)
收稿时间:2013-03-25;收到修改稿时间:2013-05-06

参数: 采样周期 sampling_rate, 上阈值 up_threshold, 下调因子 sampling_down_factor. 在一个采样周期中获取 cpu 的利用率值, 并判断其是否大于上阈值, 若结果为真则将 cpu 频率调至最高, 在 cpu 利用率较低时, 则将 cpu 频率调低. 在 cpu 处于最高频时, 下调因子用于延长采样周期, 例如采样周期为 50ms, 下调因子为 10, 则当 cpu 处于最高频时, 采样周期将被延长到 50ms*10=500ms, 通过这样降低负载评估的开销而提高了性能, 而且能够帮助 CPU 在真正忙的时候保持在最高频率运行, 而不是在高低速度间来回切换.

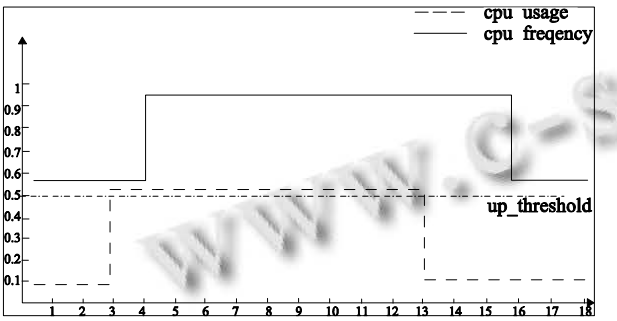

图 1 ondemand 模式调频过程

ondemand 模式的确在一定程度上降低了处理器的功耗, 但是这种模式也存在以下不足:

(1) 作为基于利用率的 DVFS 机制, ondemand 只关注了 cpu 负载参数这一外在信息, 忽略了引起 cpu 负载变化的内在因素, 即任何硬件的负载变化都是由于使用该硬件的各任务需求变化导致的. 通过负载参数无法准确预测下一阶段的负载变化, 不具有前瞻性.

(2) 调频延迟问题. 每次调频操作都是基于当前周期的负载统计信息在下一周期将 cpu 频率调整到一个合适的值. 而当前周期的 cpu 频率却并不适合实际的负载需求. 图 1 中显示了 ondemand 模式调频过程, 可以看到在负载上升后一段时间 ondemand 模式才感知到这种变化并提高 cpu 频率, 未能及时满足任务的 cpu 负载需求, 损失了一定的性能; 相应地, 在负载下降一段时间后才降低 cpu 频率, 功耗还有优化的空间.

由于 ondemand 模式对 cpu 负载变化无法提前预知, 在进行 cpu 调频时存在一定延迟, 从而对性能和功耗带来损失. 故我们提出了一种基于任务行为分析[2]的 DVFS 机制(TC-DVFS). 通过在线务在操作系统层的行为和任务对 CPU 的需求变化, 提取出影响任务 CPU 负载需求变化的关键信息, 并据此在出现关键信息时实时的行处理器调频.

与现有工作相比, 本文的贡献有:

① 提出任务行为中存在引起任务的 cpu 负载变化的关键信息, 并提出识别关键信息的算法.

② 首次提出并实现了基于任务行为分析的 DVFS 机制, 结合任务行为特征与 DVFS 技术实现性能与功耗优化.

本文使用不同类型的应用对比我们的 TC-DVFS 和默认的调频算法, 实验结果显示, 使用 TC-DVFS 机制总体上可以取得 10%的性能提升, 并降低 5%调频失效率和 5%的系统能耗.

本文章节安排如下: 第一章介绍 DVFS 与系统级行为分析的相关工作. 第二章讨论基于任务行为分析的 DVFS 出发点及前期的验证性实验. 第三章详述了 TC-DVFS 的实现机制. 第四章结合真实平台上得到的数据讨论 TC-DVFS 与 ondemand 模式的优劣性. 第五章总结我们的工作.

1 相关工作

在 DVFS 研究领域, Kihwan Choi 等提出了基于工作量划分[3]的 dvfs, 通过区分任务处于 on-chip 和 off-chip 模式来提供不同的 DVFS. 文献[4-6]则提出了基于 cache miss 的 dvfs 策略, 文献[7]中提出基于程序的 IPC 值来调整 cpu 状态. 在文献[8]中, 通过 PMU 统计的执行过程中指令条数和访存次数来选择一个最合理的频率. 文献[9]中利用 PMU 统计程序在片上执行时间和片外访存时间的比例来实现 DVFS 以达到更精细的功耗控制. Thidapat Chantem[10]等提到在计算密集型应用中如何在温度阈值限制下最大化性能, 文献[11,12]则对 DVFS 调度时机的选择进行了探讨.

在系统级行为分析中, 系统调用作为进程与 OS 交互的主要手段, 可以反映出进程对资源的申请使用情况. 操作系统层管理硬件资源时需要达到与应用层的具体实现无关性. 因此在应用程序处于黑盒情况无法跟踪应用的代码信息时, 系统调用便成为了解任务行为变化的主要手段. Abhinav Pathak[13]等人利用系统调用信息构建功耗模型. 在能耗评估时, 基于利用率的功耗模型存在 0.4%-20.3%的错误率, 而基于系统调用信息的方法可以降低到 0.2%-3.6%. Iker Burguera[14]通过探测任务的系统调用信息识别智能手机上的恶意软件. 与安全软件相比, 恶意软件在操作

系统层执行的系统调用序列具有显著区别，通过对恶意软件样本的系统调用信息进行分析，提取出恶意软件特有的系统调用序列并构建特征库，在对未知应用进行安全性评估时，将其运行时的系统调用信息与特征库匹配来识别是否为恶意软件. Timothy Sherwood 等提出任务执行具有阶段性[15]，通过识别这种阶段性特征及以往任务行为预测下一阶段的任务行为.

与现有研究工作相比，本文主要结合已有的 DVFS 技术和我们提出的任务行为分析方案对系统性能和功耗进行优化. 通过检测任务的系统调用信息，及时感知任务的 cpu 负载需求变化，并作相应的 DVFS 来降低调频延迟，以实现系统性能及功耗的优化. 图 2 中的紫色线条代表 cpu 的负载变化，蓝色线条与红色线条分别描述了 2 种 ondemand 模式在负载变化时的调频时机，它们之间的差异仅在于采样周期不同，采样周期越短，调频延迟越小，对系统性能的影响也相应越大. 绿色线条为我们预期的 TC-DVFS 工作方式，它可以对负载变化作更及时的响应.

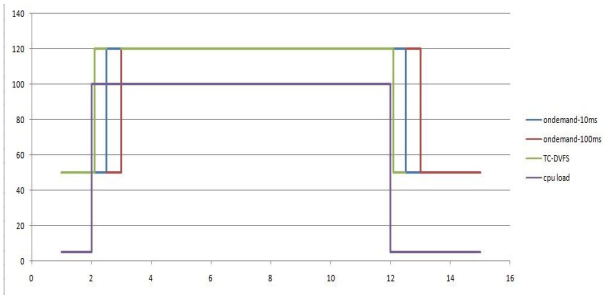

图 2 负载变化时不同模式的调频时机

2 任务行为分析

进程作为操作系统中拥有资源的最小单位,是通过操作系统提供的系统调用接口申请硬件资源，任务的系统调用信息一定程度上反映了任务对硬件资源的需求情况. 如果任务的系统调用序列和任务对硬件需求变化之间存在某种联系，那么通过检测任务的系统调用就可以预知下一阶段任务对硬件的需求情况并实时的对相应的硬件作状态调整.

通过周期性采集任务系统调用序列及任务的 cpu 占有率信息并在不同类型的应用上进行实验后，我们发现每个应用中都存在与任务 cpu 负载具有关联性的系统调用. 图 3 中描述了在 firefox 运行过程中 connect 系统调用出现后的任务负载需求变化情况. 横坐标数值代表一次采样周期，纵坐标代表 firefox 的 cpu 占有率. 出现红色线条代表发生 connect 系统调用，蓝色线条表示运行过程中 firefox 的 cpu 占有率随时间变化情况. 从图中可以看到每当出现 connect 后，总会带来任务的 cpu 需求显著上升. 相应地，图 4 中记录了 bzip2 运行时 exit 与任务 cpu 需求关系，每当任务调度到 exit 时，下一阶段任务的 cpu 需求量便会显著下降. 以上实验结果表明，任务的系统调用序列中确实存在一些系统调用，它们的出现会引起任务 cpu 需求的显著改变. 通过前期验证实验，我们认为表示识别出这些系统调用，在检测到任务调用它们时便可以预知下一阶段任务的 cpu 需求情况，从而实时的进行 cpu 调频，实现我们降低调频延迟的预期.

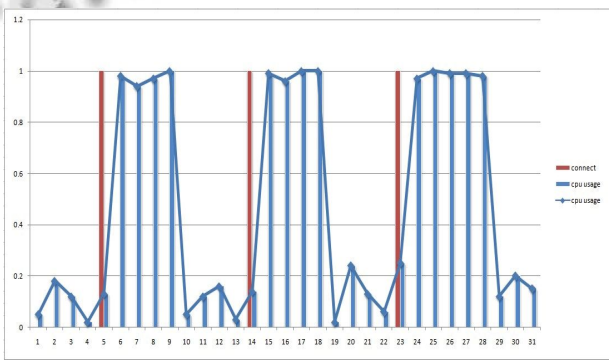

图 3 firefox 中 connect 系统调用与任务 cpu 需求关系

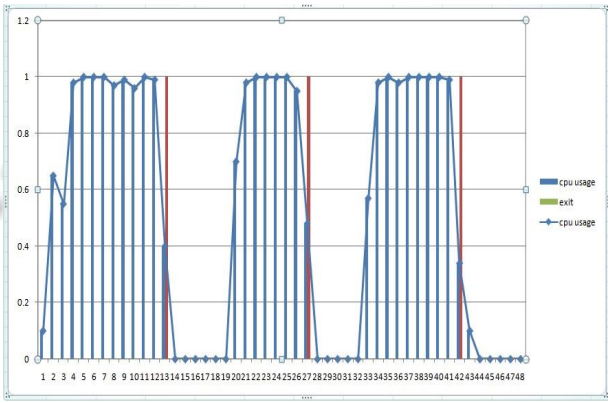

图 4 bzip2 中 exit 系统调用与任务 cpu 需求关系

3 TC-DVFS设计

通过前述分析和验证性实验证明通过检测任务的某些系统调用可以让我们提前感知到任务的 cpu 负载需求变化. 为了识别出这一类引起任务负载变化的系统调用，首先需要获得任务的系统调用与 cpu 需求信息. 研究过程中，我们使用了内核探针工具 systemtap，它以模块形式在需要探测的内核函数入口地址前添加

额外代码来获取内核函数的执行时信息，包括调用它的进程名，函数执行时长，参数大小等等. 在固定间隔的采样周期中，systemtap 通过探测内核中系统调用的实际处理函数便可以获取到任务的 cpu 占有率以及该周期中任务发出的各系统调用情况. 以下列举出采集 bzip2 运行时信息的结果(如图 5). 图中蓝线 usage 代表 bzip2 运行过程中其 cpu 占有率的变化情况，其他每种颜色的线条代表一种系统调用在运行中的出现次数变化情况. 横坐标中的数值代表一次采样周期，蓝色线条在纵坐标的数值代表本次采样周期中 cpu 占有率，其他线条在纵坐标的数值代表一种系统调用本周期中的出现次数. 图中可以看到，brk、open 等等只出现在 cpu 占有率两次显著上升阶段. 而 bzip2 调用 exit、exit_group 和 munmap 后任务的 cpu 占有率显著降低. 掌握任务的这种行为特征后便可以跟踪这些系统调用来发现任务负载需求变化，降低调频延迟. 而传统 ondemand 模式需要在任务负载变化一段时间后才能通过 cpu 利用率感知的这些信息.

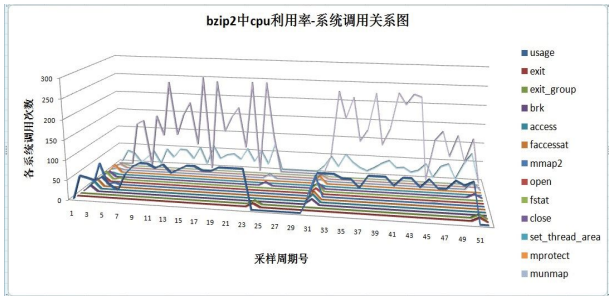

图 5　bzip2 中系统调用与 cpu 占用率关系图

3.1 识别关键系统调用

根据系统调用对任务 cpu 负载需求的影响程度将其分为 cpu 无关系统调用和 cpu 相关系统调用. 在任务运行时，我们需要智能的识别出引起任务负载变化的关键系统调用. 本文首先利用计数器法识别出任务的 cpu 无关系统调用；然后使用在线机器学习在 cpu 相关系统调用中识别出指导 DVFS 的关键系统调用.

3.1.1 无关系统调用识别

为了区分任务运行中产生的系统调用是否为无关系统调用我们引入每系统调用计数器 *light_count* 和 *heavy_count* 统计各系统调用在高负载阶段和低负载阶段下的出现次数情况. 分析某特定任务的行为时，固定时间间隔采样任务 cpu 占用率和系统调用次数情况. 在每个采样周期中，若目标任务处于低 cpu 占用率，则将本周期中出现过的系统调用 *i*, *i.light_count* 加一；相反若目标任务处于高 cpu 占用率状态，则将本周期中出现过的系统调用 *i*, *i.heavy_count* 加一. 在多次运行目标任务后对所有出现过的系统调用，分别读取其 *light_count* 和 *heavy_count*. 对于系统调用 *i*，若其 *light_count* 为较大值而 *heavy_count* 几乎为 0，则说明 *i* 出现时任务均处于低负载阶段；若其 *heavy_count* 为较大值而 *light_count* 几乎为 0，则说明 *i* 出现时任务均处于高负载；若 *heavy_count* 和 *light_count* 均不为 0，则 *i* 出现时任务可能处于低负载状态也可能处于高负载状态，说明这种系统调用对于任务负载没有关键性影响，即我们所称的无关系统调用.

我们列出 firefox 中各系统调用的 *heavy_count* 和 *light_count* 统计值，结果如图 6 所示，从中可以看出 gettimeofday、futex 等系统调用的 2 个计数器值均不为 0，firefox 执行时他们在 cpu 低负载和高负载阶段都有调用，无法判定他们与 cpu 负载的关联性，我们的算法也确实将他们归为了无关系统调用；而 brk、exit 等系统调用 2 个计数器值中只有一个为正值，调用它们后任务只在低负载或高负载中出现，它们属于和任务 cpu 需求相关的系统调用.

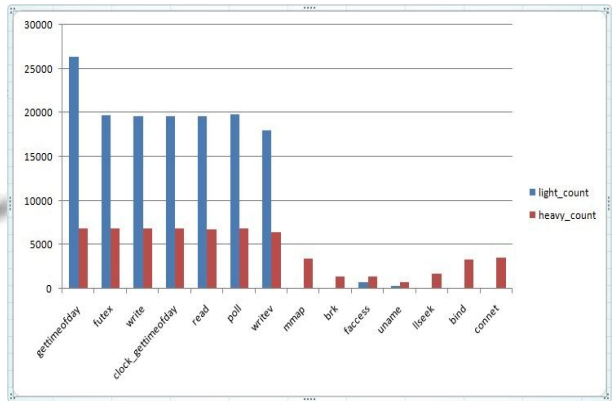

图 6　firefox 中各系统调用的计数器信息

3.1.2 关键系统调用识别

利用计数器统计信息，我们识别出了任务的无关系统调用. 然而，并不是所有的相关系统调用都对指导 DVFS 具有指导意义. 有些系统调用只出现在任务持续在稳定负载状态时，这种稳定状态下不需要进行 DVFS，需要识别出那些出现后引起任务 cpu 需求显著改变的关键系统调用.

为了实现在任务运行中的智能的识别关键系统调用，我们使用人工智能的机器学习算法. 机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法. 通常来讲，机器学习分为学习以及推理两个阶段. 在学习阶段，机器学习算法从数据中自动分析获取规律；在推理阶段，算法利用学习阶段学习到的规律对未知数据进行预测.

我们为每个系统调用设置可信度因子 α (初值为0),其值在(-1,1)范围内波动. 通过循环性的采集固定时间间隔 T 中任务系统调用信息与任务负载信息，并对采集到的样本信息进行学习. 在任务的 cpu 需求显著上升时，将样本中出现的每个系统调用 i 的 α_i 增加与负载增长量相应的值；反之，在任务 cpu 需求显著下降时将 α_i 减少与负载下降量相应的值. 对于关键系统调用，由于它出现后任务 cpu 需求或始终增加或始终减少，故它们的可信度因子呈单调递增或单调递减趋势，最终它们的可信因子值会趋向-1 或 1；对于不属于这种变化趋势的可信度因子进行一定的惩罚以降低其向两极延伸的趋势. 在经历一段学习阶段，与任务负载的关联度较大系统调用，其可信度因子会因持续增长或降低而达到阈值-1 或 1 后，提取出这些系统调用来推理下一阶段的期望负载，并对期望负载与下一阶段采集的实际负载进行比较，评估各系统调用推理结果的准确率，当某一系统调用的推理正确率低于 90% 时，说明其推理的效果不佳，永久性放弃使用该系统调用推理下一阶段期望负载. 在推理起步阶段，预测的总体准确率会比较低，随着时间的推移，不断丢弃预测效果不佳的系统调用，总体的预测准确率也在逐渐上升. 在总体的预测准确率在我们设定的阈值(30min)内持续稳定在 90%以上后，说明基于当前的系统调用集预测任务 cpu 负载已经可以取得很好的收益. 将此时的系统调用添加到该任务的行为特征库，指导处理器的调频. 至此对该任务的分析已经完成，停止对该任务的学习.

我们以 bzip2 为例介绍在线学习并识别关键系统调用的过程. 图 7 中列举了学习阶段 brk、exit_group 和 read 的可信因子值在运行时的变化情况. 从图中各系统调用的可信因子值变化趋势可以看到 brk 和 exit_group 可信因子值呈单调递增或递减的特征，最后到达阈值后被用于任务负载预测，而 read 出现时任务 cpu 负载时而增长时而下降，无法基于它对任务负载进行预测，在我们算法中通过不断惩罚，其可信因子也始终处于 0 值附近，也就不会被用于负载预测.

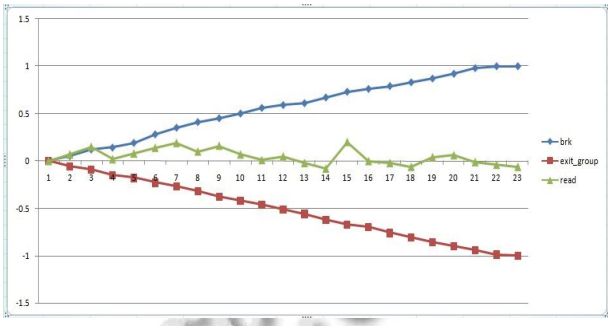

图 7 不同系统调用可信因子运行时值变化趋势

在 brk、exit_group 等系统调用的可信因子到达 1 或-1 后，通过不断的使用它们预测任务下一阶段负载，并与实际结果比较以统计其准确率，丢弃预测准确率较低的系统调用，在任务总体预测率持续稳定在 90% 以上，停止对该任务的学习，并将保留下预测任务负载的系统调用添加到 bzip2 的行为特征库中. 稳定时各系统调用预测准确率信息如图 8 所示，可以看到它们的预测准确率都超过 90%. 添加机器学习前后运行一次 bzip2 的总时间分别为 63s 和 65s，机器学习带来额外的 2s 时间开销，性能损失约 3%. 由于任务中的执行系统调用种类是有限的，在不断淘汰预测效果不佳的系统调用中，任务最后总会收敛于只剩关键系统调用的状态. 当达到这种收敛后，只需简单跟踪任务的关键系统调用进行 DVFS 即可，不在需要对该任务进行学习，因此额外开销只存在未知任务的学习阶段，总体上对系统性能的影响有限.

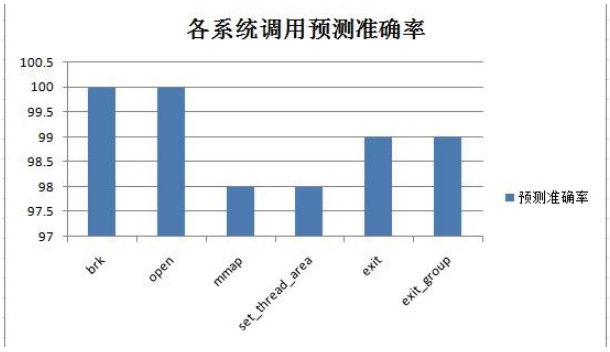

图 8 稳定时各系统调用的预测准确率

3.2 TC-DVFS 总体设计

我们已经描述了如何通过机器学习识别关键系统

调用并构建任务特征库, 在该任务以后的运行中, 只需检测其任务特征库中的系统调用, 并作相应的处理器调频, 图 9 中以 bzip2 为例描述调频过程, 其中 brk 和 exit 分别引起任务负载显著上升和下降:

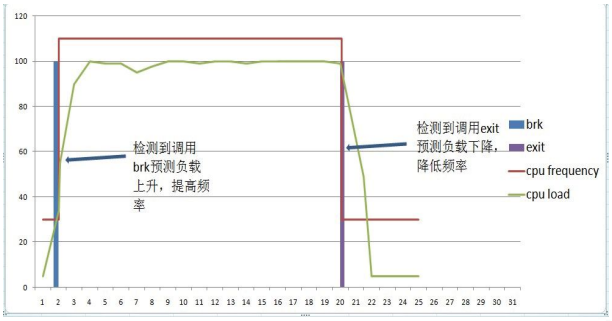

图 9　运用特征库进行处理器调频

由此 TC-DVFS 总体流程如下: 当未知任务到来时, 利用计数器法和在线学习提取该任务的关键系统调用并构建特征库, 完成任务分析后, 通过跟踪任务特征库中的关键系统调用, 检测到调用它们后执行相应 DVFS. TC-DVFS 总体工作原理图如图 10 所示.

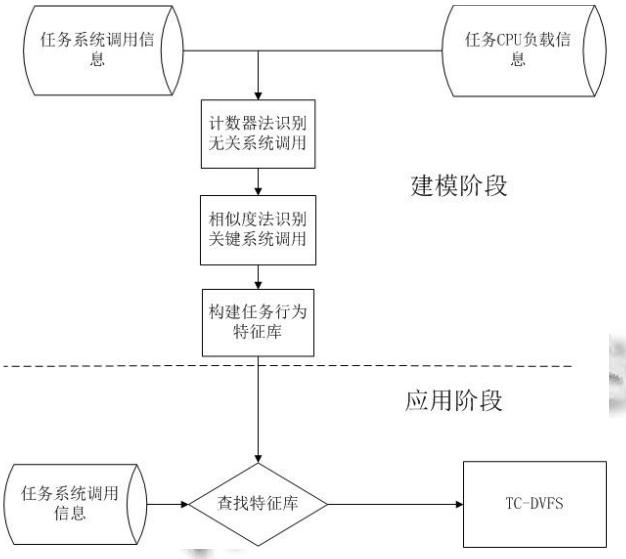

图 10　TC-DVFS 工作原理图

4　实验结果与分析

本文基于 Intel 双核处理器,可调频率为 1.8GHZ, 2.0GHz 与 2.3GHz, 操作系统内核使用 linux-2.6.38 内核. 在 ondemand 模式与 TC-DVFS 模式下分别对 5 种不同类型应用测试各项参数.

4.1 性能优化

在性能测试中对于非交互式任务, 例如 gcc 和 bzip2, 统计两种模式下的任务的完成时间. 对于交互式任务 gedit 和 firefox, 统计负载波动阶段的任务响应时间. 例如浏览器开启耗时, 网址打开时间等. 表 1 中显示了针对不同应用 TC-DVFS 较 ondemand 的性能提升. 实验结果显示TC-DVFS对于cpu负载需求频繁波动的应用提升效果更好, 总体可以达到10%的性能提升.

表 1　TC-DVFS 与 ondemand 的性能对比

	ondemand	TC-DVFS	加速比
firefox	71.661ms	51.557ms	14.102%
bzip2	395.108s	374.669s	4.89%
gcc	241.562s	226.701s	6.12%
gedit	92.52ms	78.959ms	14.64%
mplayer	141.062ms	141.187ms	0%

我们从中提取出 2 种模式下 firefox 的响应时间比较, 并添加 powersave 模式和 performance 模式下的测试结果作为参照, 如图 11 和表 2. 图 11 中为不同模式下的浏览器响应时间, 表 2 中取各模式下响应时间的均值. 结果显示TC-DVFS较ondemand-10ms降低了约15%的响应时间, 提高了系统的反映速度, 证明 TC-DVFS 可以更快的对任务负载变化做出响应以满足任务对 CPU 的需求.

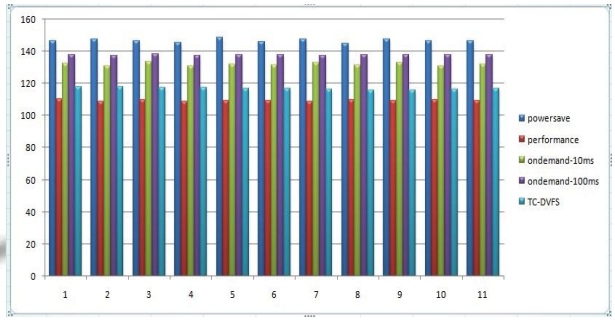

图 11　各 cpu 工作模式下的浏览器响应时间比较

表 2　各 cpu 工作模式下的浏览器响应时间比较(均值)

Powersave	Performance	ondemand-10ms	ondemand-100ms	TC-DVFS
146.6356ms	109.2777ms	131.895ms	137.7559ms	116.7543ms

4.2 调频命中率

理想状态下, CPU 频率会随着 CPU 负载的变化实时调整. 真实系统上, 负载与频率之间的契合度是衡量一种 DVFS 机制优劣的重要标准. cpu 频率工作在一个不合理的值时间越长, 则其调频效果越差. 这种不合理是指在系统存在多个可调频率时, 如果当前 cpu

利用率已经高于上限值, 但 cpu 频率仍然可以有上调的空间(称为 up_miss); 或者 cpu 利用率低于下限值但频率仍然有下调的空间(称为 down_miss). 总 miss 率即为(up_miss 数+down_miss 数)/总采样周期数.

图 12 中显示, 在 CPU 负载和频率之间分别出现了一次 up_miss 和 down_miss. 当 up_miss 出现较多时表示没有及时的保证性能, 而当 down_miss 出现较多时表示功耗还可以进一步优化, 能耗的使用效率不高.

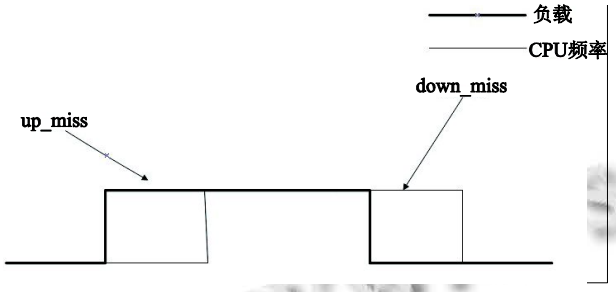

图 12 cpu 频率与负载契合度

为减少 up_miss 在任务出现 cpu 负载需求时 TC-DVFS 机制将处理器频率调至最高值以满足应用需求, 而为减少 down_miss, 在任务负载下降时根据预测任务负载在所有可选频率中一次性选择出可以保证 CPU 工作在 80%以上负荷的频率.

通过统计不同应用的调频命中率, 即负载与频率一致的比例, 结果如表 3 所示. 可以看到对于表中应用, TC-DVFS 较 ondemand 模式在调频的命中率上平均约有 5%左右的提升. 尤其是在 firefox 这类应用中, 由于负载切换频繁, ondemand 的调频延迟带来了更多的不匹配次数. 因此, TC-DVFS 在命中率的提升上也更加显著.

表 3 TC-DVFS 与 ondemand 的调频命中率对比

	ondemand	TC-DVFS
firefox	85.1%	96.8%
bzip2	94.7%	99.3%
gcc	95.2%	98.9%
gedit	85.7%	90.0%
mplayer	100%	100%

4.3 能耗优化

通过感知任务在操作系统层的行为可以预测任务下一阶段的负载变化, 从而可以很快的将频率调整到一个合适的值, 在降频阶段时, ondemand 模式通常会持续 1-2 个周期的高频后才会降低频率, 带来了不必要的功耗浪费. 表 4 是对不同应用测试得到的两种模式功耗对比结果, TC-DVFS 较 ondemand 平均节省 5%的系统功耗.

表 4 TC-DVFS 与 ondemand 功耗对比

	ondmeand	TC-DVFS	功耗优化
firefox	59.52w	57.13w	4.15%
bzip2	77.13w	72.73w	5.97%
gcc	72.92w	63.72w	11.98%
gedit	56.24w	53.43w	5.01%
mplayer	56.87w	56.85w	0.1%

5 总结

本文提出了一种基于任务行为分析的 DVFS 机制 TC-DVFS, 通过分析任务的负载变化与任务系统级行为的关系, 发现影响任务 cpu 负载的关键信息, 从而利用该信息实时对处理器调频, 实验结果显示与传统 ondemand 模式相比, 本文所提出的方法能够对性能和功耗带来可观的收益, 证明了通过对任务的行为分析掌握其资源需求变化, 从而对硬件资源状态进行调整的方法是有效的. 这种分析方法未来也将其应用到内存, I/O 等硬件资源的状态调整中.

参考文献

1 kernel.org/pub/linux/kernel/linux/2.6.38/Documentation/cpu-freq/governors.txt.

2 Wiseman Y, Jiang S. Advanced Operating Systems and Kernel Applications: Techniques and Technologies. 1st ed. New York.Hershey. 2009. 156–199.

3 Choi K, Soma R, Pedram M. Dynamic voltage and frequency scaling based on workload decomposition. Proc. of the 2004 International Symposium on Low Power Electronics and Design. New York. ACM. 2004. 174–179.

4 Marculescu D. On the use of microarchitecture-driven dynamic voltage scaling. Workshop on Complexity-Effective Design. Wisconsin. 2000.151–156.

5 Vijaykrishnan N, Kandemir M, Irwin MJ, Kim HS, Ye W. Energy-driven integrated hardware-software optimizations using simple power. Vancouver. Proc. of the 27th International Symposium on. Canada. June 2000. 95–106.

6 Pering T, Burd T, Brodersen R. Dynamic voltage scaling and the design of a low-power microprocessor system.in power-driven microarchitecture workshop conjunction with int. Symposium on Computer Architecture. Barcelona, Spain.

(下转第 38 页)

的迁移，同时，设计一种准入控制和任务调度算法，实现负载发生资源的共享管理和动态分配，满足多租户管理需求.

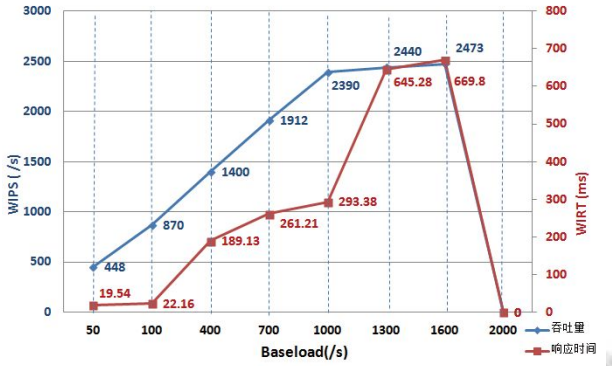

图 5 使用迁移后的 Bench4Q 进行性能测试

参考文献

1 Reuters. (2009)SOASTA Leverages the Cloud to Test 1,000,000 Users on MySpace. http://cn.reuters.com/ article/ pressRelease/idUS121904%2B17-Nov-2009%2BMW20091117.

2 Zhang WB, Wang S, Wang W, Zhong H. Bench4Q: A QoS-oriented E-commerce benchmark. Proc. of 35th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2011). Munich, Germany, 2011: 38–47.

3 Wikipedia.(2012) HP LoadRunner. http://en.wikipedia.org/wiki/HPLoadRunner.

4 Zhang Q, Cheng L, Boutaba R. Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applicatons, 2010, 1(1): 7–18.

5 林海略,韩燕波.多租户应用的性能管理关键问题研究.计算机学报,2010,33(10): 1881–1895.

6 OW2,(2012)Bench4Q.:http://forge.ow2.org /projects/jaspte.

7 Wu LL, Garg SK, Rajkumar Buyya. SLA-based admission control for a software-as-a-service provider in cloud computing environments. Journal of Computer and System Sciences, 2012, 78(5): 1280–1299.

8 Popek GJ, Goldberg RP. Formal requirements for virtualizable third generation Architectures. Commun. ACM, 1974, 17: 412–421.

(上接第 7 页)

June1998:14–19.

7 Ghiasi S, Casmira J, Grunwald D. Using IPC variation in workloads with externally specified rates to reduce power consumption. Workshop on Complexity Effective Design. Wisconsin. 2000. 96–101.

8 Weissel A, Bellosa F. Process cruise control: event-driven clock scaling for dynamic power management. CASES. Grenoble. France. October 2002. 234–246.

9 Choi K, Soma R, Pedram M. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. Automation and Test in Europe Conference and Exhibition, Proceedings. 2004, 1: 4–9.

10 Chantem T, Hu XS, Dick RP. Online work maximization under a peak temperature constraint. ISLPED'09, August 2009:19–21.

11 Wang L, von Laszewskiy G, Dayaly J, Hey X, Andrew J, Younge Y, Furlani TR. Towards thermal aware workload scheduling in a data center. Pervasive Systems, Algorithms, and Networks(ISPAN). 2009 10th International Symposium. 14-16 Dec. 2009.

12 Mukherjee T, Tang Q, Ziesman C, Gupta SKS, Cayton P. Software architecture for dynamic thermal management in datacenters.COMSWARE. Bangalore, India. 2007. 1–11.

13 Pathak A, Hu YC, Zhang M. Fine-grained power modeling for smartphones using system call tracin. EuroSys'11. Salzburg, Austria. 2011.

14 Burguera I, Zurutuza U, Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for android. SPSM'11. Chicago, Illin. October 17, 2011.

15 Sherwood T, Perelman E, Hamerly G, Sair S, Calder B. Discovering and exploiting program phases. IEEE Micro, 2003, 23(6): 84–93.