动态时间片缩放:一种优化 Linux 任务公平性的方法①

高旭宏[1], 李 曦[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: 处理器动态电压频率调节技术, 对 Linux 系统中并发任务的性能产生不同程度的变化, 从而影响并发任务计算资源分配的公平性. 提出了一种利用动态时间片缩放来优化任务公平性的方法, 并基于 Linux 操作系统任务调度程序, 加入动态时间片缩放模块, 该模块通过读取 CPU 性能监控计数器, 在线计算时间片缩放系数, 并利用该系数对任务时间片长度进行动态缩放. 实验表明, 这种方法以较小的系统开销为价, 极大地提高了 Linux 中并发任务计算资源分配的公平性.

关键词: 动态电压频率调节; Linux 任务调度程序; 并发任务; 公平性; 动态时间片缩放

Dynamic Time-Slice Scaling: A Scheme to Improve Fairness of Tasks in Linux

GAO Xu-Hong[1], LI Xi[1,2]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Suzhou Institute for Advanced Study, University of Technology and Science of China, Suzhou 215123, China)

Abstract: The processor dynamic voltage and frequency scaling(DVFS)leads to unequal performance loss for concurrently running tasks in Linux, affecting the fairness of CPU sharing. This paper proposes a dynamic time-slice scaling scheme to improve the fairness of CPU sharing for concurrently running tasks. The method which is implemented based on Linux task scheduler dynamically scales the length of time-slice for each task using a time-slice scaling factor which is calculated based on the processor performance monitoring unit statistics. Experiment results show that dynamic time-slice scaling scheme can significantly improve the fairness of the CPU sharing with low overhead compared with the conventional Linux scheduler algorithm.

Key words: dynamic voltage and frequency scaling; Linux task scheduler; concurrently running tasks; fairness; dynamic time-slice scaling

随着处理器芯片集成度的不断提高, 处理器功耗问题和温度问题日益突出[1,14]. 研究人员相继提出了电源管理[15,16]、功耗管理[2,14]和温度管理[1]等手段来解决上述问题. 动态电压频率调节 DVFS(dynamic voltage and frequency scaling)[4-12]是一种有效的功耗温度管理手段, 它是基于“降低 CPU 电压可以按照电压的平方与频率的乘积的比例关系降低功耗”理论[2], 在软件运行时调整处理器电压和频率, 从而降低处理器功耗和温度. DVFS 技术分为实时 DVFS 和非实时 DVFS 两类. 实时 DVFS[11,12,14]主要通过评估任务的执行时间和最坏执行时间, 在满足硬实时要求的前提下, 降低电压和频率, 从而降低功耗, 由于任务的执行时间无法预先知道, 这种方法实际应用较少; 而非实时 DVFS[4,5,7,8,10,13]技术主要通过对系统工作负载的评估和预测来选择合适的电压和频率. 目前应用在 Linux 系统中的 DVFS 技术为 ondemand governor[6], 它根据 CPU 利用率来调整处理器电压和频率. DVFS 技术的缺点是它会造成系统性能损失, 特别是对分时操作系统中各个并发任务造成不均衡的性能损失.

① 基金项目:江苏省产学研前瞻性联合研究项目(BY2009128)
收稿时间:2012-03-03;收到修改稿时间:2012-04-10

现代分时操作系统 Linux 以应用为导向, 其系统中同时存在大量并发任务. 从微观角度看, 这些任务通过获取 Linux 内核调度程序给他们分配的时间片而得以运行, 当一个任务时间片耗尽, 转而执行下一个任务. 由于处理器上各个并发任务的指令动态特征各不相同, DVFS 机制的引入, 造成了各个并发任务的性能变化量在相同的频率变化刻度下差别较大. 以处理器最高工作频率时的性能为基准, 在下降相同频率的过程中, 不同任务的性能下降值往往不同. 图 1 给出了 Linux 操作系统中, 同时运行两个 SPEC 2000 标准测试程时, 在不同频率点下的归一化性能值. 从实验结果可以得到: 随着处理器频率的降低, 两个测试程序的性能变化差异越来越大. 这种不公平的性能下降现象, 造成某些线程执行较快, 某些线程执行较慢, 甚至出现线程饥饿以及优先级反转等问题, 从而降低系统服务质量.

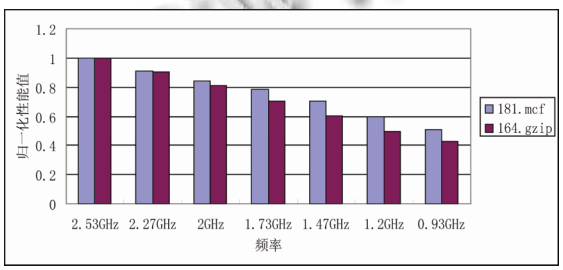

图 1 164.gzip+181.mcf 在不同频率下性能变化

因此需要一种手段来对各个并发任务的性能下降趋势进行平衡控制. Linux 操作系统内核调度程序自 2.6.23 版本以后, 采用了完全公平调度程序 CFS(completely fair scheduler)[17,18]. 该调度程序对并发任务计算资源分配的公平性, 体现在以经验性的任务时间片分配权重来进行时间片分配. 在一个调度周期内, 所有任务的时间片长度和权重成正比. 然而 CFS 调度器并没考虑到处理器 DVFS 技术对公平性的破坏.

本文基于 Linux 任务调度程序, 加入动态时间片缩放模块. 该模块通过动态计算任务时间片缩放系数, 然后与原始的任务时间片分配权重相乘, 使得那些性能下降较多的任务执行较长的时间片, 性能下降较少的任务执行较短的时间片, 从而抵消处理器 DVFS 技术对并发任务计算资源分配公平性的影响. 对比加入动态时间片缩放模块前后实验数据发现, 该方法以较小的开销为代价, 极大地降低了 DVFS 技术对并发任务公平性的影响.

1 基于Linux的动态时间片缩放模块设计

1.1 动态时间片缩放模块设计概述

动态时间片缩放模块的设计思想是: 在 Linux 操作系统运行过程中, 在线计算 DVFS 对各个任务造成的性能下降因子; 然后根据各个任务的性能下降因子, 让性能下降较多的任务执行较长的时间片; 反之, 执行较短的时间片. 系统实现框架如图 2 所示, 划分为两个子模块:

(1) 时间片缩放系数 TSF(time-slice scaling factor)在线计算模块. 该模块通过在线评估 Linux 中任务性能下降因子来计算得到 TSF, TSF 是动态时间片缩放程度的量化指标.

(2) 时间片缩放模块. 该模块使用 TSF 与原始时间片长度的分配权重相乘, 从而实现动态时间片调整.

系统运行过程中, 时间片缩放系数 TSF 在线计算模块, 在每次任务切换过程中, 读取 CPU 性能监控计数器、当前 CPU 工作频率以及任务时间片长度, 然后计算时间片缩放系数 TSF; 在任务调度过程中, 时间片缩放模块则利用 TSF 与原始的时间片分配权重(该值由任务优先级决定)相乘得到新的时间片分配权重, 然后 Linux CFS 调度器依据新的时间片分配权重来分配任务时间片, 从而实现动态时间片缩放功能.

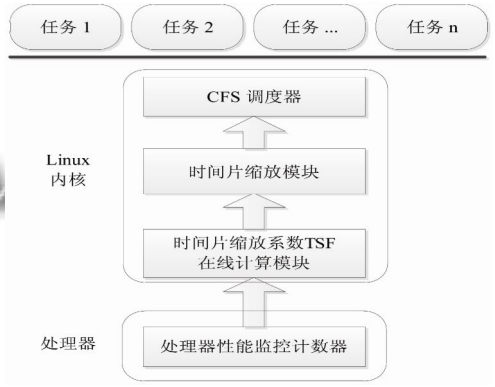

图 2 动态时间片缩放模块系统框架

1.2 Linux 内核完全公平调度程序(CFS)简介

Linux 内核自 2.6.23 开始引入了由 Ingo Molnar 开发的完全公平调度程序 CFS(Completely Fair Scheduler). 其基本思想是, 根据任务的优先级, 给每个任务分配一个经验性的权重, 然后按照该权重给各个任务分配时间片. 在一个调度周期内, 任务的时间片长度与任务权重成正比, 从而实现基于优先级的公平性. 然而 CFS 调

度程序对 DVFS 技术造成的任务性能不均衡损失一无所知, 因此无法保证 DVFS 情况下的公平性.

1.3 时间片缩放系数 *TSF* 在线计算模块

任务性能与 CPU 频率之间并非严格的线性关系,本节通过理论分析得出任务性能和 CPU 频率之间的关系, 进而给出 TSF 的理论计算方法和在线计算方法.

1.3.1 时间片缩放系数 TSF 的理论计算方法

为了分析性能和频率之间的关系, 假定处理器的工作频率和最大工作频率分别为 f 和 f_{max}. $T(f)$代表执行任务时间片的长度, 因此这段时间内的时钟周期数为 $f*T(f)$. CPU 的时钟周期数可以划分为两个部分[18]: 频率敏感部分 C_{sense}, 这部分受 CPU 频率影响很大; 非频率敏感部分 C_{non_sense}, 这部分不受 CPU 频率影响.

因此有:

$$f*T(f)=C_{sense}+C_{none\_sense} \tag{1}$$

定义 T_{non_sense} 为不受频率影响的执行时间, 可得:

$$T(f)=\frac{C_{sense}}{f}+T_{none\_sense} \tag{2}$$

于是这段任务在最高频率下执行时间为:

$$T(f_{max})=\frac{C_{sense}}{f_{max}}+T_{none\_sense} \tag{3}$$

由于 T_{non_sense} 表示不受频率影响的时钟周期数消耗的时间, 故公式(2)(3)中的 T_{non_sense} 相同[18]. 以这段程序在最高频率 f_{max} 时的性能(执行时间的倒数)作为参考值, 程序在频率为 f 时的归一化性能为:

$$P(f)=\frac{T(f_{max})}{T(f)} \tag{4}$$

当 $f=f_{max}$ 时, $P(f)=1$; 当 $f<f_{max}$ 时, $T(f)>T(f_{max})$, 故 $0<P(f)<1$. 将公式(2)(3)带入公式(4)得到:

$$P(f)=\frac{C_{sense}+T_{none\_sense}*f_{max}}{C_{sense}+T_{none\_sense}*f}*\frac{f}{f_{max}}=\theta*\frac{f}{f_{max}} \tag{5}$$

结合公式(1)(2)(5), θ 可以进一步表示为:

$$\theta=\frac{f^2*T(f)-f*C_{none\_sense}+f_{max}*C_{none\_sense}}{f^2*T(f)} \tag{6}$$

因此, θ 与当前处理器频率 f、任务时间片长度 $T(f)$ 以及 C_{none_sense} 有关. 由于各个任务时间片的指令特征各不相同, 导致 C_{none_sense} 也不相同, 因此不同任务的 θ 也各不相同. θ 反映了不同任务对处理器调频的敏感程度.

考虑多任务并发情况, 假定当前系统中有 n 个任务在并发执行, 任务集定义为: $Task\_Set=\{task_1, task_2, \cdots, task_n\}$. 时间片按照权重比例进行分配, 每个任务的原始时间片权重设为: $TASK\_SLICE\_WEIGHT=\{w_1, w_2, \cdots, w_n\}$, 所有任务时间片都执行一次的时间为一个调度周期 S_{epoch}. 因此当频率为 f 时, $task_i$ 在一个调度周期内获得的计算资源 $cr\_share\_task_i$ 为:

$$cr\_share\_task_i\{f\}=P(f)*\frac{w_i}{\sum_{i=1}^{n}w_i}*S_{epoch} \tag{7}$$

当 $f=f_{max}$ 时, $P(f)=1$, 因此有:

$$cr\_share\_task_i\{f_{max}\}=\frac{w_i}{\sum_{i=1}^{n}w_i}*S_{epoch} \tag{8}$$

以频率为 f_{max} 时的 $cr\_share\_task_i$ 为基准, 当频率下降到 f 时, 任务的计算资源下降因子 $CRSR$ 为:

$$CRSR_{task_i}\{f\}=\frac{cr\_share\_task_i\{f\}}{cr\_share\_task_i\{f_{max}\}}=\frac{P(f)*\frac{w_i}{\sum_{i=1}^{n}w_i}*S_{epoch}}{\frac{w_i}{\sum_{i=1}^{n}w_i}*S_{epoch}}=P(f) \tag{9}$$

根据公式(5)(6)(9)可得, 在并发任务时间片长度不变的情况下, 频率下降而导致的任务性能下降因子与任务的 C_{non_sense} 参数相关, 而 C_{non_sense} 随任务不同而不同, 从而导致不同并发任务的性能下降比例不同.

并发任务性能下降公平性, 体现在调频情况下, 各个并发任务的 $CRSR$ 是否一致: 如果各个任务 $CRSR$ 相同, 表明他们获取的计算资源按照相同比例下降, 此时各个任务是完全公平的; 因此使用各个任务的 $CRSR$ 参数集来计算标准差, 从而定义 DVFS 公平度为:

$$Fair\_Degree(f_{max}\rightarrow f)=\sigma\{CRSR_{task_1}\{f\},\ldots,CRSR_{task_n}\{f\}\} \tag{10}$$

公式(10)中, σ 为标准差运算符. 当 $Fair\_Degree$ 的计算结果为 0 时, 表示完全公平的情况; 反之, $Fair\_Degree$ 的计算结果越大, 表明各个任务的 $CRSR$ 差值也越大, 公平性越差.

根据前面的分析, 当处理器频率从 f_{max} 下降到 f 时, 各个任务的 θ 参数不同, 从而导致不公平现象. 为了达到完全公平性, 本文提出了基于 CFS 调度程序的时间片动态缩放方法. 其基本思想是: 让性能下降较多的任务执行较长的时间片, 性能较少的任务执行较短的时间片. 具体方法是: 在每个调度周期内, 给每个任务计算一个时间片缩放系数 TSF, 然后用 TSF 乘以

任务的原始时间片分配权重. 时间片缩放参数 *TSF* 按照公式(11)计算.

$$TSF(task_i)=\frac{1}{\theta_i}=\frac{f^2*T(f)}{f^2*T(f)-f*C_{none\_sense}+f_{max}*C_{none\_sense}} \quad (11)$$

经过时间片缩放之后, 新的任务计算资源为:

$$cr\_share\_task_i'\{f\}=P(f)*\frac{TSF(task_i)*w_i}{\sum_{i=1}^{n}(TSF(task_i))*w_i}*S_{epoch} \quad (12)$$

将公式(5)(6)(11)带入公式(12)得到:

$$cr\_share\_task_i'\{f\}=\frac{f}{f_{max}}*\frac{w_i}{\sum_{i=1}^{n}(TSF(task_i))*w_i}*S_{epoch} \quad (13)$$

当频率为 f 时, 新的计算资源下降因子 *CRSR'*为:

$$CRSR_{task_i}'\{f\}=\frac{cr\_share\_task_i'\{f\}}{cr\_share\_task_i'\{f_{max}\}}=\frac{\frac{f}{f_{max}}*\frac{w_i}{\sum_{i=1}^{n}(TSF(task_i))*w_i}*S_{epoch}}{\frac{w_i}{\sum_{i=1}^{n}w_i}*S_{epoch}}=\frac{f}{f_{max}}*\frac{\sum_{i=1}^{n}w_i}{\sum_{i=1}^{n}(TSF(task_i))*w_i} \quad (14)$$

根据公式(14)可知, 此时 *CRSR'* 和 C_{non_sense} 无关, 在处理器下降到频率 f 时, 各个任务的 *CRSR'*值相同. 在理论上, 可以实现任务性能下降比例因子 *CRSR'*的完全公平性.

1.3.2 *TSF* 的在线计算方法

TSF 依赖于三个参数: 当前 CPU 频率 f、任务时间片长度 *T(f)*以及 C_{none_sense}. 当前 CPU 频率 f 记录在 Linux 内核结构体 *cpufreq_policy* 的 *cur* 字段中. 任务时间片长度 *T(f)*通过 Linux 内核 *sched_entity* 结构体成员 *sum_exec_runtime* 和 *prev_sum_exec_runtime* 之差得到. C_{none_sense} 的计算基于 CPU 性能监控计数器. 现代 CPU 提供了大量的性能监控计数器, 这些计数器可以统计程序执行时的各种特征参数, 例如 IPC、cache miss 等. 研究[13,18]表明, 不受处理器频率影响的 cycle 数主要是由于计算指令访存而引起的处理器停顿 cycle 数, 因此该 cycle 数主要依赖于访存次数和访存延迟时间. 本文基于 intel 处理器性能监控计数器[19]来读取指令访存次数, 访存延迟时间使用 MaxxMEM2[20] 来测量得到.

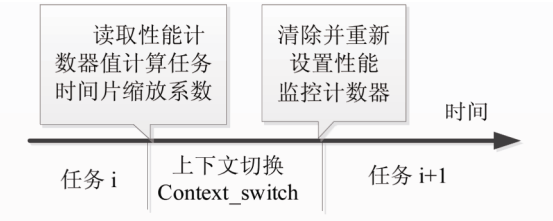

图 3 *TSF* 在线计算过程

TSF 的在线计算过程如图 3 所示, 任务执行上下文切换时调用 *context_switch* 函数, 该函数在开始和结尾处分别调用了 *prepare_task_switch* 和 *finish_task_switch*. 因此首先在 *prepare_task_switch* 中插入桩代码, 对前一个任务 i 产生的性能监控计数器值进行读取并计算 *TSF*. 最后在 *finish_task_switch* 中清空并重新设置性能计数器. 实验中使用 *rdmsr* 和 *wrmsr* 两条汇编指令来读取 CPU 性能监控计数器.

1.4 时间片缩放模块

在计算得到 *TSF* 之后, 通过给 Linux 的任务控制块数据结构 *task_struct* 添加一个变量来存放 *TSF*. 然后在下一个调度周期, 使用 *TSF* 对任务时间片长度进行缩放. 以 $se.weight_{taski}$ 表示任务 i 的原始时间片分配权重, 这个权重存放在 linux 内核数组 *prio_to_weight* 中. 图 4 描述了时间片缩放过程: 首先根据任务优先级索引 *prio_to_weight* 表, 得到任务原始时间片分配权重 $se.weight_{taski}$, 然后按照公式(15)所示, 用 *TSF* 乘以 $se.weight_{taski}$ 得到新的权重 $weight\_task_i$.

$$weight\_task_i=se.weight_{taski}*TSF_{taski} \quad (15)$$

新的任务总权重按照公式(16)所示求和得到.

$$total\_weight=\sum_{i=1}^{n}(se.weight_{taski}*TSF_{taski}) \quad (16)$$

最后任务新的时间片 $task\_slice_{taski}$ 按照公式(17)所示, 依据新的时间片分配权重 $weight\_task_i$ 来分配.

$$time\_slice_{taski}=\frac{weight\_task_i}{total\_weight}*total\_time\_slice \quad (17)$$

2 实验与结果分析

实验目的是为了测试动态时间片缩放模块带来的任务公平性优化和系统额外开销. 从 SPEC CPU 2000 标准测试程序集中选择了 3 个典型的测试程序进行组合实验, 3 个测试程序介绍如表 1 所示. 实验环境配置

如表 2 所示.

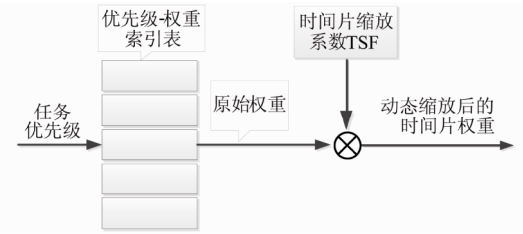

图 4　时间片缩放过程

表 1　测试程序简介

164.gzip	流行的压缩工具gzip
177.mesa	3-D 图形库
181.mcf	运输调度优化程序

表 2　实验环境配置

CPU	Intel i3
RAM	2GB
DISK	500GB
L1 cache大小	32KB
L2 cache 大小	256KB
L3 cache 大小	3MB
操作系统	Ubuntu10.10+Linux-2.6.35.9
Benchmark	SPEC2000

2.1 任务公平性实验与结果分析

对 3 个测试程序进行两两组合运行, 共进行三组实验. 每组实验中, 每个测试程序分别产生 8 个任务, 共 16 个并发任务.分别测试在处理器频率为 2.53GHz、2.27GHz、2GHZ、1.73GHz、1.47GHz、1.2GHz、0.93GHz 的性能, 以 2.53GHz 时的性能为基准, 计算其他各个频率点的归一化性能参数. 然后使用各个任务的归一化性能参数, 按照 1.3.1 节所述, 利用标准差来计算并发任务计算资源分配公平性.

图 5、6、7 给出了三组实验加入动态时间片缩放模块前后的公平性对比(数值越小, 公平性越大; 2.53GHz 性能参数作为计算基准, 实验前后公平度都为 0, 故未列出). 从实验结果可以得到: 加入动态时间片缩放模块后, 并发任务的归一化性能差异维持在较低的水平. 三组实验 18 对数据对比中, 两组数据中公平性(图 6、7 中频率为 2.27GHz 时)略有降低, 其余实验组公平性均有不同程度的提升. 以公平度数值的下降比例来衡量公平性的提升率, 最小公平性提升为 22.6%, 最大公平性提升 81.2%. 因此实验论证了动态时间片缩放方法优化任务公平性的有效性. 由于在线计算 *TSF* 是根据局部性原理, 将上一个调度周期得到的 *TSF* 参数, 用来对下一个调度周期内的任务时间片进行缩放, 程序局部性的好坏将影响时间片缩放效果.

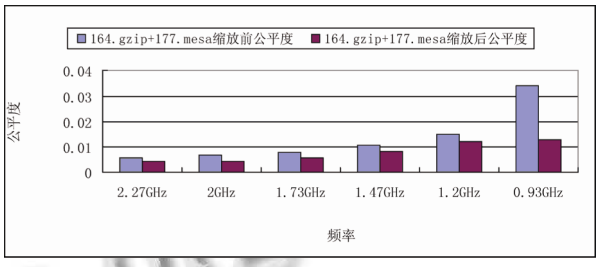

图 5　164.gzip+177.mesa 动态时间片缩放前后公平度

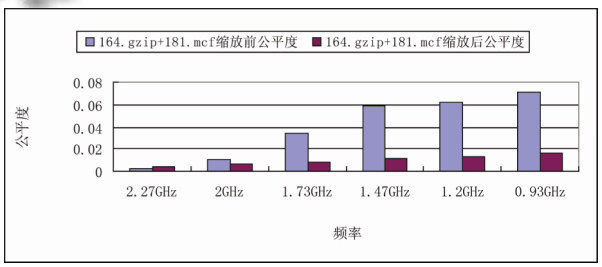

图 6　164.gzip+181.mcf 动态时间片缩放前后公平度

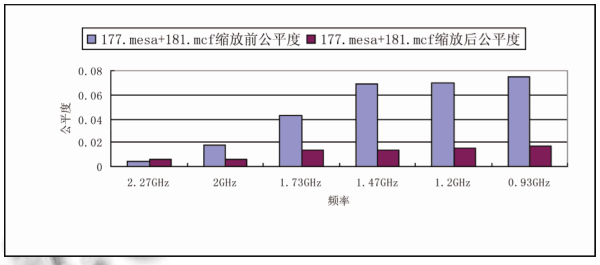

图 7　177.mesa+181.mcf 动态时间片缩放前后公平度

2.2 系统开销分析

动态时间片缩放模块的开销, 主要来自两方面: 读取 Linux 性能监控计数器并计算 TSF 的开销, 以及时间片缩放计算开销. 为了评价动态时间片缩放模块引入的额外开销, 需要逐个运行测试程序来进行评价. 分别测试了 164.gzip、177.mesa、181.mcf 在加入动态时间片缩放模块前后的执行时间. 图 8 给出了运行三个测试程序时, 在不同频率下的系统额外开销. 实验中测得最大开销为 2.56%, 最小开销为 2.17%, 运行三个测试程序的平均系统开销分别为 2.34%、2.36%、2.42%, 可见动态时间片缩放模块对系统带来的额外开销较小.

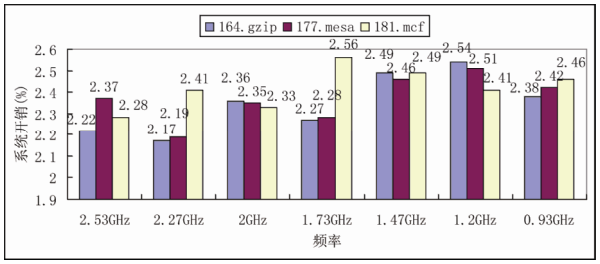

图 8 系统开销

3 结语

本文针对处理器 DVFS 技术对 Linux 操作系统中并发任务带来的计算资源分配的公平性损失, 提出了动态时间片缩放的解决方案. 基于 Linux 任务调度程序, 加入动态时间片缩放模块. 对比传统的 Linux 任务调度程序, 这种方法以较小的代价, 极大的提高了并发任务计算资源分配的公平性.

参考文献

1 Donald J, Martonosi M. Techniques for multicore thermal management:classification and new exploration. Proc. of 33rd International Symposium on Computer Architecture. 2006. 78 –88.

2 Benini L, De Micheli G. System-Level power optimization: Techniques and tools. ACM Trans. on Design Automation of Electronic Systems, 2000: 115–192.

3 Jejurikar R,Gupta R. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems Proc. of the 42nd annual Design Automation Conference. New York, 2005. 111–116.

4 Zanini F, Jones CN. Multicore thermal management using approximate explicit Model Predictive Control. Proc. of IEEE International Symposium on Circuits and Systems. 2010. 3321–3324.

5 Cai Q, Gonzalez J, Magklis G, Chaparro P, Gonzalez A. Thread shuffling: Combining DVFS and thread migration to reduce energy consumptions for multi-core systems. Proc. of 2011 International Symposium on Low Power Electronics and Design. Aug. 2011. 379–384.

6 Pallipadi V, Starikovskiy A.The ondemand governor. Proc. of the Linux Symposium, 2006, 2:223–238.

7 Hughes C, Li T. Optimizing throughput/power trade-offs in hardware transactional memory using DVFS and intelligent scheduling. Proc. of the International Conference on Supercomputing, 2011. 141–150.

8 Dhiman G, Rosing TS. Dynamic voltage frequency scaling for multi-tasking systems using online learning. Proc. of the 2007 International Symposium on Low Power Electronics and Design. New York, 2007. 207–212.

9 Spiliopoulos V, Keramidas G, Kaxiras S, Efstathiou K. Poster: DVFS management in real-processors. Proc. of the International Conference on Supercomputing, 2011.

10 Kolpe T, Zhai SA, et al. Enabling improved power management in multicore processors through clustered DVFS. Proc. of Design, Automation & Test in Europe Conference & Exhibition. 2011. 1–6.

11 Jayaseelan R, Mitra T. Temperature aware task sequencing and voltage scaling. Proc. of the 2008 IEEE ACM International Conference on Computer-Aided Design, Piscataway. NJ, IEEE Press, 2008. 618–623.

12 Hanumaiah V, Vrudhula S. Temperature-aware DVFS for Hard Real-time Applications on Multi-core Processors. IEEE Trans. on Computers, 2011.

13 Huang S, feng W. Energy-Efficient Cluster Computing via Accurate Workload Characterization. Proc. of 9th IEEE/ ACM International Symposium on Cluster Computing and the Grid. 2009. 68–75.

14 Venkatachalam V, Franz M. Power reduction techniques for microprocessor systems. ACM Comput. Surv., 2005,37(3): 195–237.

15 Benini L, Bogliolo A, De Micheli G. A survey of design techniques for system-level dynamic power management IEEE Trans. on Very Large Scale Integration Systems, June 2000,8(3):299–316.

16 Lu YH, De Micheli G. Stanford Univ, CA Comparing system level power management policies. Of Proc. of Design & Test of Computers. IEEE, Apr 2001. 10–19.

17 Pabla CS. Completely fair scheduler. Linux Journal archive, Volume 2009 Issue 184, Aug. 2009.

18 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. Newport Beach, California, USA, Aug., 2004. 174–179.

19 Intel®64 and IA-32 Architectures Software Developer's Manual Volume 3B:System Programming Guide, Part 2. Available at :http://www.intel.com.

20 maxxmem2:http://www.maxxpi.net/pages/downloads/ maxxmemsup2---preview.php.