数据库服务器系统中一种有效的功率封顶机制

杨良怀 阮忠孝 朱红燕 王佰心
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 数据中心的一个重要任务是功率控制,功率封顶是数据中心对服务器设置功率消耗上限的技术。关注的是数据中心节点机一级的动态功率控制机制。基于系统级功率模型构建了进程级功率模型,并将两者整合构成"软功率计",用于监控系统功率与进程功率;为实现功率封顶,软功率计被集成到闭环控制系统中,设计了功率控制的算法,该算法在控制系统功率不超预算的情况下,系统以较好的性能运行。实验结果表明所提控制机制能有效地控制系统的实时功率,且性能下降较少,同时也可以改善能效,可应用于功率感知的DBMS服务器中。

关键词 功率封顶,功率建模,软功率计,功率感知数据库系统

中图法分类号 TP315 **文献标识码** A

Effective Power Capping Scheme for Database Server

YANG Liang-huai RUAN Zhong-xiao ZHU Hong-yan WANG Zhou-xin
(School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract Power control is a critical issue in data center and power capping is the technique to keep the system within a fixed power constraint. This paper focused on the dynamic power control scheme in a data center node machine. We constructed a process-level power model based on our previous system-level power model, and integrated these two models into a gadget called soft power-meter to control system power and process power. To achieve power capping, the soft power-meter is integrated into a closed-loop control system, and a power control algorithm is devised, which keeps the system within the fixed power budget with good performance. The experiment results demonstrate that the proposed power capping scheme can effectively control the system's power with small performance degradation, and improve the energy-efficiency. It can be applied to a power-aware DBMS server.

Keywords Power capping, Power modeling, Soft power-meter, Power-aware database systems

1 引言

当今大部分三层计算结构中,数据库管理系统(DBMS)是其中重要的一类系统软件。在典型的数据中心,大部分的计算资源用于数据库服务器;所部署的软件中,DBMS是最大的能耗部件。解决计算机能耗问题可以从硬件与软件两个方面开展[11]。在软件方面,解决数据中心能耗问题,需要对服务器中的核心软件数据库管理系统开展能效研究。2008年数据库界一些有影响力的研究人员讨论了数据库领域面临的现状与挑战,提出了未来研究方向的一些建议[1],认为数据库界处在一个历史转折点,核心数据库引擎方面的重要研究议题之一是功率感知的DBMS。功率感知DBMS的研究是数据库领域今后的一个重要研究议题,是数据库界今后努力的"圣杯"[2]。

数据中心的一个重要因素是峰值功率。机架有额定供电限制,若用电超过额定限制会引起保险丝熔断。峰值功率会导致升温、超过冷却能力或大大增加冷却成本。因此,数据中心通常实施功率预算,由机架到节点机的各层协同控制器进行实施[3]。数据中心服务器的配置为处理峰值负载而设计,但并非时时刻刻都会达到峰值负载状态,因此必然存在能源浪费的问题,资源利用率较低,存在较多节能机会。因此动态功率管理技术在数据中心能效研究领域得到了广泛关注。

一个系统的能耗一般由两部分组成[14]:(1)静态(固定)部分由漏电电流引起,大小与系统规模及部件类型有关;(2)动态能耗由系统活动以及时钟速率变化引起。通过极小化静态能耗以及交付与动态能耗成正比的性能来改善能效是很活跃的两个研究领域。静态的功率管理面临诸多弊端,如能源浪费问题严重、资源利用率低,亦或者因为系统的动态性,运行时的复杂性致使系统功率出现较大的波动,从而可能使系统功率超出预算。因此,一个实际的系统必须具备运行时动态反馈机制,根据运行时参数,动态调整执行状态,从而控制系统功率。

本文关注的是数据中心中节点机一级的动态功率控制机制,即功率封顶,功率封顶是数据中心对服务器设置功率消耗上限的技术。基于原有系统级功率模型[21],进一步构建了进程级功率模型,并将两者整合构成"软功率计",用于监控系统

本文受浙江省基金项目(LY14F020017,LY13F020026),国家自然科学基金面上项目(61070042)资助。

杨良怀(1967—),男,博士,教授,主要研究方向为数据库系统,E-mail:yanglh@zjut.edu.cn;阮忠孝 硕士生,主要研究方向为数据库系统;朱红燕 硕士生,主要研究方向为数据库系统;王佰心 硕士生,主要研究方向为数据库系统。

功率与进程功率；为实现功率封顶，本文把软功率计集成到闭环控制系统中，设计了功率控制的算法，该算法在控制系统功率不超预算的情况下，以较好的性能运行。实验结果显示所提控制机制能有效地控制系统的实时功率，且性能下降较少，同时也可以改善能效，可应用于功率感知的 DBMS 服务器中。

本文第 2 节介绍相关工作；第 3 节论述功率封顶机制；第 4 节为性能评价，给出实验结果并对其进行分析；最后总结全文。

2 相关工作

能效问题是近年来学术界与工业界关注的一个重要议题。Orgerie 等[14]对改进计算与网络资源能效的技术进行了综述，讲述了从单个节点到基础设施的各种能效解决方案；Venkatachalam 与 Franz[15]对微处理器的降耗技术进行了综述；Beloglazov 等[16]对硬件层、软件层、数据中心层、虚拟化层等方面的功耗问题研究进行了分类总结；赵霞等[17,18]对软件与操作系统的功率管理作了综述；叶可江等[19]则对虚拟化云计算平台的能耗的测量、建模、管理、优化等方面进行了综述，提出了所面临的挑战性问题；Benini 等[7]综述了系统级的动态功率管理方法，包括动态功率管理的策略及其在系统中的实现；Bianchini 与 Rajamony[20]对服务器上的功率和节能的已有工作和存在的问题进行了综述，包括热点数据集中转移、集群轻负载节点机的负载集中机制，其目的是将某些设备转入低功耗状态。

Do 等[8]在 Linux 系统中根据处理器的活动时间、不同执行频率上的运行时间以及执行频率之间的切换次数构建处理器的能量模型，根据单位时间内磁盘/网卡的读写量来估算磁盘/网卡的能耗模型；将每个活动进程对各个硬件的占用比例与各部件额定功率相结合来分摊硬件的总能耗，得到进程的能耗模型。并基于这些模型形成工具 pTop。其缺点是系统的状态变化会影响部件的功率，简单地用额定功率来计算会降低模型的准确性。基于 pTop 的思想，Shi 等[9]构建了 Windows 环境的 pTopW 工具，pTopW 在模型中考虑了内存的影响，并实现了能耗感知框架“EnergyGuard”用于识别能耗异常的应用程序。虽然该框架准确性有待提高，但大多数情况下能预测功率的变化趋势。

实时功率与峰值功率是功率的两个方面。功率封顶与两者都有联系。功率测量方法可分为硬件测量和软件预测[10]。仪器测量的优点是精确，缺点是价格昂贵无法大规模地部署在数据中心，但这种方法可用于验证或评估模型的有效性。软件预测在准确性上不如仪器测量，但能够提供细粒度的联机功率信息，具有经济灵活的优点。Economou[23]对服务器进行了实时功率建模，模型把处理器、磁盘、内存和网卡的相关性能指数作为系统功率的自变量，建立功率与硬件性能指数之间的线性回归模型，其模型的平均误差范围为 0～15%。文献[22]给出了数据库系统中的核心操作连接算子在串行执行时的峰值功率估计方法。

文献[21]采用了 Economou[23]的模型设计思想，利用组件级的性能指数来预测系统的实时功率，但对其中的参数进行了进一步细分，包括 CPU 核的利用率及其相应的执行频率、磁盘利用率（未考虑网络通信功率）。多核架构是未来系统发展的必然趋势，从核粒度来考察更能反映系统的真实运行状态，结果表明细分后的模型相对误差更低。

Felter 等[4]指出冷却与电力供应是根据峰值功率来设计的，降低峰值功率可以缓解这方面的限制。通过检测服务器部件（主要是 CPU、内存）前一时间区间的活动状况（处理器发射指令数、内存请求数）来预测下一时间区间的活动状况，并根据事先分配的功率来确定处理器与内存的节流阈值，限制各部件的活动，进行功率调节。Meisner 与 Wenisch[5]讨论了数据中心峰值功率建模问题，意在帮助解决数据中心功率封顶。数据中心功率封顶方面的研究有很多，但在数据库方面，这方面的工作还较少。数据库系统如何调节执行机制使其工作在目标功率包络内且仍然最大化性能是其中的一个挑战[12]。

3 功率封顶机制

功率封顶是数据中心将功率消耗限制在某一范围之内的技术。一般地，对于服务器来说，处理器、内存和磁盘是主要的耗能部件，而其中处理器功率占主要部分，因此，控制处理器的功率消耗对于系统的功率控制至关重要。可以从算法实现级对系统功率进行细粒度的调节，也可以在系统级采用动态电压频率调节。目前，动态电压频率调节技术（DVFS）、时钟节流技术等已被广泛应用于实际系统中，并且效果显著。本文针对后者方法开展系统功率控制的研究。

3.1 功率封顶闭环控制系统结构

闭环控制理论在数字系统中的应用见 Franklin[6]著述。将之结合到数据库服务器系统中，图 1 给出了功率封顶闭环系统的基本架构。其中“软功率计”和“功率控制器”分别对应于控制系统中的“观察器”和“控制器”。软功率计对系统整体或者负载运行功率进行监控，将观察到的工作负载信息（功率）反馈给功率控制器，由功率控制器向系统发出功率调节指令，实施相应的执行策略，从而调整系统的运行状态，由此形成闭环反馈控制框架，构成功率控制系统。

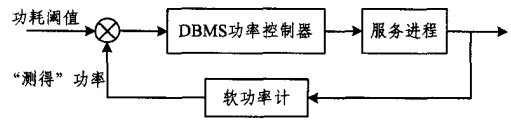


图 1 闭环反馈功率控制系统

3.2 软功率计的构建

软功率计的任务是监控系统功率。但整机系统的功率监控粒度偏粗，无法区分进程优先级，控制功率时无法区别对待。为了有更好的灵活性，软功率计除了感知系统级功率（文献[21]中的式(8)）外，还需要感知细粒度进程级功率。

3.2.1 进程级功率模型

本文所述的功率管理基于对系统和进程的功率分析，所以，在系统功率模型的基础上，进一步构建进程级的功率模型，使其能够对进程的功率变化进行实时监控。

进程功率是指程序运行过程中使用处理器、磁盘以及内存等系统部件、资源而产生的功率。通过分别计算各个进程单位时间内对各种资源的使用率，根据各进程对某一资源的使用比例来分摊该资源的总功率，将各进程在不同资源分摊

的功率求和即得到相应进程的功率。这种分摊策略具有可行性，通过设备利用率来计算消耗功率具有较好的准确性(已得到验证[22])。下面利用该策略来构建进程级功率模型。

在 Linux 系统下，伪文件系统/proc 中不仅提供了系统级资源使用信息，也提供了某一活动进程的资源使用情况。处理器、磁盘的整体活动信息可以分别从/proc/stat 和/proc/diskstats 文件中获取；活动进程的信息记录在/proc 中以进程号 *pid* 命名的文件夹，文件/proc/[*pid*]/stat 记录了进程在处理器上的使用时间，文件/proc/[*pid*]/io 则记录了进程的读写数据量信息。根据这些信息，可以计算出活动进程在处理器以及磁盘上的活动时间。

(1)处理器功率比率

为了得到某活动进程的处理器活动时间，可从/proc/[*pid*]/stat 文件中获得相应活动进程的 *utime* 和 *stime* 两个值。*utime* 表示用户态运行时间，*stime* 表示系统内核态运行时间，两者求和得到该进程在处理器上消耗的总时间。*utime* 与 *stime* 的值是从进程启动时算起的累计值。假设以 1s 为采样周期，计算前后两次 *utime* + *stime* 的差值，可得到在 1s 内该进程的处理器使用时间；进程的处理器使用时间与系统的处理器时间(指系统中处理器被使用的时间总和)之比即进程所占处理器使用时间的比率，记作 *cpu_ratio*，用它表示进程产生的动态功率所占比重。

假设 t_1 和 t_2 为两次采样时刻($t_2>t_1$)，$utime_i$ 和 $stime_i$ 表示 t_i 时刻所获取的统计信息。令 p_time_i 表示进程在 t_i 时刻为止的 CPU 使用时间累计，run_time_i(i=1,2)分别表示在 t_1 时刻和 t_2 时刻系统级处理器活动累计时间，结合 CPU 相关统计信息(见表 1)可得 *cpu_ratio* 计算方法如下：

$$\begin{aligned} &p\_time_1 = utime_1 + stime_1 \\ &p\_time_2 = utime_2 + stime_2 \\ &run\_time_1 = user_1 + system_1 + nice_1 \\ &run\_time_2 = user_2 + system_2 + nice_2 \\ &cpu\_ratio = (p\_time_2 - p\_time_1) \\ &\qquad /(run\_time_2 - run\_time_1) \end{aligned} \tag{1}$$

表 1 处理器性能统计信息

参数	参数说明
user	从系统启动开始累计到当前时刻，用户态的 CPU 时间(单位：jiffies)，不包含 nice 值为负进程
nice	从系统启动开始累计到当前时刻，nice 值为负的进程占用的 CPU 时间
system	从系统启动开始累计到当前时刻的内核态运行时间

(2)磁盘功率比率

/proc/[*pid*]/io 文件记录了某活动进程的磁盘活动信息，但该文件并未直接给出进程的磁盘活动时间，而是记录了磁盘的读写数据量，*read_bytes* 和 *write_bytes* 分别表示到目前为止累计读盘数据量和写盘数据量；*tot_ticks* 表示从系统启动到当前时刻磁盘的活动时间。根据磁盘的平均读写速率，可将读写数据量转化为磁盘的读写时间，进而可计算进程关于磁盘功率的所占比率，记作 *dsk_ratio*。类似地，假设 t_1 和 t_2 为两次采样时刻($t_2>t_1$)，$read\_bytes_i$ 和 $write\_bytes_i$ 分别表示 t_i 时刻进行读/写的数据量，tot_ticks_i 记录 t_1 和 t_2 时刻磁盘活动累计时间，则 *dsk_ratio* 的计算方法如下：

$$\begin{aligned} &dsk\_time_1 = read\_bytes_1/KBRPERSEC + \\ &\qquad write\_bytes_1/KBWPERSEC \\ &dsk\_time_2 = read\_bytes_2/KBRPERSEC + \\ &\qquad write\_bytes_2/KBWPERSEC \\ &dsk\_ratio = (dsk\_time_2 - dsk\_time_1)/ \\ &\qquad (tot\_ticks_2 - tot\_tcks_1) \end{aligned} \tag{2}$$

式中，*KBRPERSEC*、*KBWPERSEC* 分别表示磁盘平均读盘速率与平均写盘速率。可用 *dsk_ratio* 计算某活动进程磁盘的动态功率。

(3)进程功率模型

文献[21]采用多元线性回归方法拟合执行核的执行频率和利用率、磁盘的利用率和系统功率 *SP* 之间的关系，得到了整机系统功率模型：$SP = C + \alpha \cdot \sum_{i=0}^{i=n-1} f_{ci} \cdot u_{ci} + \beta \cdot u_{disk}$，其中常数项 C 为待测机的静态功率，f_{ci} 表示系统中某一执行核的执行频率(单位 GHz)，u_{ci} 表示第 i 个执行核的利用率，系数 α 与 β 由多元线性回归模型训练得到。实验验证了该模型是有效的。结合上文分析得到的进程处理器功率比率与磁盘功率比率，可得到进程级功率模型，如式(3)所示。其中 *PP* 表示进程功率。注意进程的运行产生系统的动态功率与静态功率无关，因此，模型中仅分摊动态功率部分，而舍去静态功率 C。该模型的物理意义是根据进程对 CPU 和磁盘使用率分别对 CPU 产生的系统级动态功率 $\sum_{i=0}^{i=n-1} \alpha_i \cdot f_{ci} \cdot u_{ci}$ 和磁盘活动产生的系统级动态功率 $\beta \cdot u_{disk}$ 分别乘以 *cpu_ratio* 和 *dsk_ratio* 进行加权求和，从而将系统的动态功率分摊至各个活动进程。

$$PP = cpu\_ratio \cdot \left(\sum_{i=0}^{i=n-1} \alpha_i \cdot f_{ci} \cdot u_{ci}\right) + dsk\_ratio \cdot \beta \cdot u_{disk} \tag{3}$$

至此，软功率计基于整机系统功率模型和进程级功率模型按照抽样间隔输出系统功率 *SP* 以及活动进程功率 *PP* 至功率控制器，实施功率控制。

3.3 功率控制器

功率控制器是反馈系统的核心组件之一。功率控制器比较系统功率 *SP* 与系统功率阈值之间的大小，当检测到 *SP* 持续时间 T 超出给定系统功率阈值时，触发频率下调操作，检测活动进程的当前功率，对超出阈值的进程所在的执行核实施降频处理；而当系统功率 *SP* 持续时间 T' 低于系统阈值时，则触发频率上调操作，检测所有活动进程的当前功率，对低于阈值的进程所在核实施增频处理。

为使控制器能够精确且有效地控制处理器频率，下面给出单核频率与进程功率之间的关系。由 CPU 功率公式 $P_{cpu} \approx cv^2 f$ 可知，执行核的功率与频率成正比。因执行核的功率占进程功率的绝大部分，可将该核的功率近似为绑定在该核运行的进程的动态功率；而将由进程运行产生的磁盘和内存的动态功率部分作为背景功率。基于以上假设，由系统功率 $\alpha \cdot \sum_{i=0}^{i=n-1} f_{ci} \cdot u_{ci} + \beta \cdot u_{disk}$ 公式，在单核模式下变为 $SP = b + \alpha \cdot u_{ci_0} \cdot f_{ci_0}$，其中所在核是 ci_0，u_{ci_0} 是进程在该核上产生的利用率，b 是背景功率。预测进程下一时刻的 CPU 利用率是较为困难的问题，本文采用前一时刻的利用率来代替下一时刻的利用率，即做了进程当前行为仍将维持到下一时刻的假设。该假设的可行性在于断路器具有一定范围的承受能力，一旦过低估计下一时刻的利用率，可在下一时刻调整纠正。这样

对于某一时刻 t，单核上运行的功率可按如下线性模型近似：

$$PP=a\cdot f+b \tag{4}$$

其中，f 表示该核的执行频率(GHz)，PP 表示进程功率(W)，频率的系数 $a=\alpha\cdot u_{\alpha_0}$，常数项 b 是背景功率。

假设在功率封顶中，每个进程都进行了相应功率预算，即可用功率的上限，其功率阈值记作 PP_s。根据式(4)给出的频率与进程功率的线性模型，若按进程当前状态(这里主要指利用率)运行，设对应进程功率预算 PP_s 的频率是 f'，则 $PP_s=a\cdot f'+b$。结合式(4)得到：$PP-PP_s=a\cdot(f-f')$。用 ΔPP 表示进程功率与进程功率预算之差 $PP-PP_s$，用 Δf 表示频率之差，则该式化为 $\Delta PP=a\cdot\Delta f$。若已知 PP，可算得频率调节的幅度 $\Delta f=\dfrac{\Delta PP}{a}$，在当前频率的基础上调节 Δf 大小，即得到目标频率。

3.3.1 进程功率预算

闭环反馈控制系统的目标在于实现对数据库服务器系统中各个查询任务的功率和系统功率的管理，因此，需要结合查询的属性特征，为任务设定合理的功率上限，从而更有效地控制进程功率。为此，引进查询任务的两个属性：优先级和峰值功率的 CPU 密集度(简称 CPU 密集度，用 s 表示)。

一条查询任务记作 Q，每当提交一条查询时，可获知该查询任务的优先级和 CPU 密集度，一条完整的查询记作 $Q(p,s)$。

1) p：表示查询任务的优先级，数值越小，优先级越高，反之亦然。

2) s：CPU 密集度定义为单位时间内处理器的使用频繁程度，以时间占用百分比来表示，即每秒 CPU 的利用率。

表 2 给出了查询属性特征的变化对功率需求的影响。

表 2 查询属性的变化对功率需求的影响

查询属性	属性变化	性能需求	功率需求
p	小(大)	大(小)	大(小)
s	小(大)	小(大)	小(大)

由表 2 可知，p 值越小，优先级越大，对性能的需求越高，从而功率需求越高，即优先级与功率呈反相关；s 值越小，即 CPU 密集度越小，对性能的需求就越小，从而功率减小，两者正相关。根据任务的属性以及属性与功率的关系，可为各个进程实施功率预算。

假设系统中某一时刻共有 $numCores$($numCores>1$)个查询在运行，系统的功率预算为 P_s 瓦。$numCores$ 个查询进程记作 $Q_i(p_i,s_i)$，$i\in[1,numCores]$，优先级和 CPU 密集度的权重分别为 α,β，且 $\alpha+\beta=1$。据此，针对一个查询任务，采用归一化算法，将优先级和 CPU 密集度两个属性归一化为一个属性，记作 w_i，下标 i 表示查询任务的编号，如式(5)所示。

$$w_i=\frac{\alpha}{p_i}+\beta\cdot s_i,\ i\in[1,numCores] \tag{5}$$

令 $w=\sum_{i=1}^{numCores}w_i$，根据进程的属性权重 w_i/w，$i\in[1,numCores]$，可将系统的动态功率预算(P_s-P_{static})分配至所有进程，即得到单一进程的功率预算 $PP_{si}=(P_s-P_{static})\cdot\dfrac{w_i}{w}$，$i\in[1,numCores]$，将该值作为进程的功率上限。

对于系统中进程数超过 $numCores$，可以将上述方法加以推广。设进程数为 N 个($N>numCores$)，则仍按上述方法计算各个进程权重，对这些进程的权重进行排序，然后按序顺次把这些进程分成 $numCores$ 组，每组$\lceil numCores/N\rceil$个，每组将绑定在同一个核上运行。每组选择权重最大的进程为代表，参与计算该组进程的功率预算。这样的好处是，仍可以按核进行调节(功率封顶)，且某种程度上保证进程的优先顺序。

3.3.2 功率控制算法

通常 CPU 核执行频率之间的切换时间为毫秒级，但频繁地切换会造成资源的浪费，也可能削弱任务的执行性能。为避免频繁地调节处理器的执行频率，需要了解断路器的工作原理。数据中心配电单元的断路器对不同功率上限可容忍的时间是不同的。Meisner[5] 给出了断路器的工作原理，如图 2 所示。当负载电流是原来的 10 倍时，断路器持续运行时间小于 1s 时不会触发断电；系统能够持续运行超过 1s 并少于 10s 的工作电流则要小一些；断路器能容忍超过 10s 以上的电流则更小；当工作电流超出上限 100 倍甚至 1000 倍时，断路器能够维持运行的时间缩短到 10～100ms 之间。从这里可以看出，电流超出阈值越多，降低工作电流的任务越迫切。

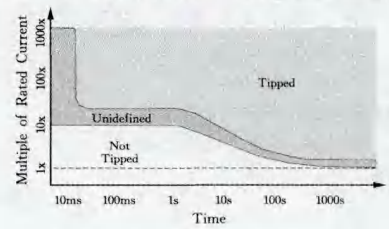


图 2 配电单元(PDU)断路器工作原理[5]

根据断路器的工作原理，软功率计监测到系统功率持续 Ts 超过或低于给定阈值时，则触发功率控制。T 根据实际运行环境和负载特性来设定。当系统功率 SP 持续 Ts 超出系统功率阈值 SP_s 时，则依次计算 $numCores$ 个活动进程的动态功率与相应阈值之间的差值，如果进程功率大于阈值，则根据差值的大小计算得到频率调节的幅度，在当前频率的基础上下降计算所得的幅度，得到调节的目标频率；当系统功率持续 $T's$ 低于阈值时，经过类似的方法找出即时功率小于阈值的进程，将该进程所在执行核的执行频率调高至目标频率，这样在功率封顶的同时兼顾系统性能。一般 CPU 允许执行的频率是离散的，目标功率取最接近的频率档值。功率控制算法描述如图 3 所示。

算法 1 PowerControl

输入：系统功率预算 SP_s，低于阈值持续时间 T'，模型有关参数

输出：设置 CPU 各核运行频率

```
Repeat
1. if 进程发生变更(数目、新进程进入、旧进程退出)
2.     for i ← 0 to numCores
3.         计算进程的功率预算 PPsi;
4.     endfor
5. endif
6. 根据采样周期获取 numCores 个进程的即时功率 PPi 以及系统功率 SP;
7. 断路器可承受当前系统功率 SP 的最大持续时间 T;
   /* 持续监测系统功率，若系统功率持续 Ts 超出阈值则调低频率 */
8. if SP > SPs lasts for T seconds/**/
   /* 对 numCores 个核，计算核绑定的进程产生的功率与阈值之差 */
```

```
9.      for i ← 0 to numCores
10.         ΔPPi← PPi－PPsi；/* PPsi表示进程的功率预算 */
11.         if ΔPPi＞0
12.             获取核 i 的运行频率 fi；
                /* 据 ΔPPi 计算频率调节幅度，a 是模型(4)中频率
                的系数 */
13.             a←α×uci；/* uci是核 i 上进程利用率 */
14.             Δf←PPi/ a；
15.             fi←(fi－Δf)；
16.             fi←选择离 fi最近的 CPU 运行频率；
17.             设置核 i 的运行频率为 fi；
18.         endif
19.     endfor
        /* 系统功率持续 T′s 低于阈值，则调高频率，功率封顶时兼顾
        性能 */
20. else if SP ＜ SPslasts for T′ seconds
21.     for i ← 0 to numCores
22.         ΔPPi← PPi－PPsi；
23.         if ΔPPi＜0
24.           获取核 i 的运行频率 fi；
25.             a←α×uci；/* uci是核 i 上进程利用率 */
26.             Δf ← abs(ΔPPi) / a；
27.               fi←(fi＋Δf)；
28.     fi←选择离 fi最近的 CPU 运行频率；
29.             设置核 i 的运行频率为 fi；
30.         endif
31.     endfor
32. endif
```

图 3　闭环控制算法伪代码

功率控制算法每次循环执行的最大开销是对 *numCores* 个活动进程的功率预算和频率的计算，以及定位合适的 CPU 频率档。设 CPU 频率档位数为 F，查找合适档位时间开销是 O(log*F*)。由此可知该算法的时间复杂度为 O(*numCores*×log*F*)。算法中需要存储 *numCores* 个进程的即时功率与功率预算，因此空间复杂度为 O(*numCores*)。

4　性能评价

本节对所提闭环反馈功率控制系统开展实验评价。

4.1　实验环境

采用杭州远方光电信息有限公司的 PF9808B 数字功率计，搭建实时功率建模的实验平台，如图 4 所示。数字功率计每秒 1 次采样被测系统的整机功率，另一台监控机通过串口读取相应采样值。实验测试用服务器的配置如表 3 所列。其中 CPU 共有 15 档频率可供调节，1.6～3.1GHz。软功率计与功率控制器在 Linux 下采用 C++语言实现。实验中，闭环控制系统假定断路器在超出系统功率 $T=3$s 时必须进行调节。

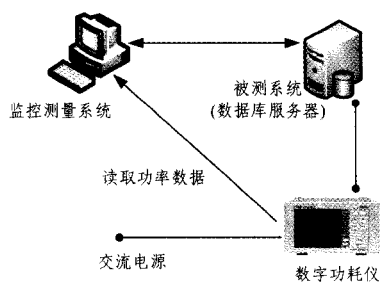


图 4　功耗测试环境示意图

表 3　实验设备规格

组件	型号	功率
操作系统	Ubuntu10.04，内核版本：2.6.33	—
CPU	Intel(R) Core(TM)i5-2400 3.1GHz	95
内存	记忆科技 DDR3 1600MHz 2G＊2	3
硬盘	西数 WD5000AAKX/7200RPM	7

4.2　实验负载

本文采用 4 种常见的数据库连接操作，即块嵌套循环连接算法(BNL)、排序归并连接算法(SMJ)、Grace 哈希连接算法(GHJ)、混合哈希连接算法(HHJ)，以及冒泡排序算法(BUBBLE)和存储密集型负载(MLoad)。用 C++实现这几种算法。

测试机中的处理器共有 4 个执行核，每次运行 4 种负载，分别绑定在 4 个核上，故对 6 种负载按照密集度的不同组合，构成 3 组综合负载，考察 3 种不同功耗需求环境下，闭环控制系统的效果(见表 4)。实验中考察 4 个任务并行运行的情况，故优先级共设置 4 个等级。

表 4　综合负载的属性与核绑定

测试负载	单核负载	核绑定	优先级	密集度
任务 1	BNL	0	4	0.15
	SMJ	1	1	0.3
	GHJ	2	2	0.23
	HHJ	3	3	0.24
任务 2	BNL	0	4	0.15
	SMJ	1	3	0.3
	MLoad	2	2	0.7
	BUBBLE	3	1	0.95
任务 3	BUBBLE	0	1	0.95
	BUBBLE	1	2	0.95
	BUBBLE	2	3	0.95
	BUBBLE	3	4	0.95

任务 1 分别将 4 种连接算法绑定到 4 个核上，绑定的顺序对功率不会产生影响，连接查询的优先级可根据用户喜好设定，计算强度取自各负载的单核利用率，且 4 种负载的计算强度相近，都较低；任务 2 采用两种连接算法，BNL 和 SMJ，计算强度较小，MLoad 和 BUBBLE 两种负载的计算强度分别为 0.7 和 0.95，对性能的需求逐渐增高；任务 3 采用 4 个核绑定同样的排序任务 BUBBLE，计算强度都为 0.95，构成高功率运行环境。数据库连接算法的数据采用 TPC-H 测试基准的中 customer 和 orders 基本表，表的规模 *Scale* 取为 3。

为考察闭环控制系统对功率控制的效果，对各任务设定了不同档次的系统功率阈值 SP_s，设定时适当考虑了不同的负载对性能需求的差异，相应地反映到功率需求上，各任务的系统功率阈值(预算)见表 5，系统测得静态功率为 $P_{static}=38.5$W；算法优先级和计算强度的权重值 α,β 取相同的值 0.5，没有考虑倾向性；根据表 4 负载优先级和 CPU 密集度，各个进程按动态功率，即 SP_s-P_{static}，进行各个进程的功率阈值 PP_{si} 的分配，分配结果如表 5 所列。

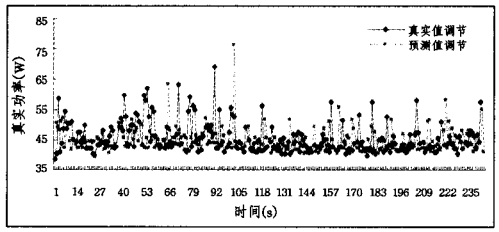
表 5　进程功率预算

负载	SP_s(W)	进程功率阈值 PP_{si}(W)			
任务 1	45	BNL	SMJ	GHJ	HHJ
		0.87	2.81	1.58	1.24
任务 2	50	BNL	SMJ	MLoad	BUBBLE
		1.1	1.74	3.3	5.36
任务 3	80	BUBBLE	BUBBLE	BUBBLE	BUBBLE
		13.78	10.22	9.05	8.45

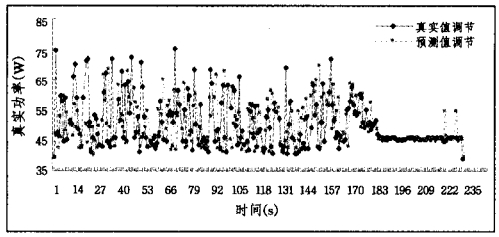
4.3 软功率计与功耗仪在功率控制中有效性比较

系统实时功率是功率控制器判断处理器核频率的调节方向与大小的决策依据。功率控制器的实时功率由软功率计提供。本节比较根据功耗仪测得实时功率的真实值进行调节与根据软功率计预测值进行调节对系统各指标的影响，包括任务的执行时间、完成任务所需的总能耗。

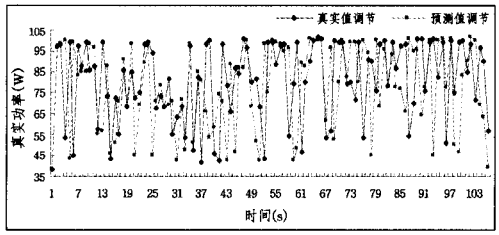
该组实验运行了表 4 所列的 3 个负载，功率控制系统分别采用功耗仪和软功率计进行调节，并通过功耗仪测得不同负载运行中在不同时间经控制器调节后的真实功率。其功率图谱如图 5 所示，x 轴表示负载运行的时间，y 轴表示功耗仪测得功率。



(a) 任务 1 功率图谱



(b) 任务 2 功率图谱



(c) 任务 3 功率图谱

图 5　基于真实值调节与基于预测值调节的任务功率图谱

根据功率图谱可知，两种情形下功率的变化走势较为相近，对 3 种任务的执行时间与总能耗两项指标进行分析，其对比结果如表 6、表 7 所列。

表 6　基于真实值调节与基于预测值调节两种情形下的时间对比

综合负载	真实值调节执行时间(t)	预测值调节执行时间(t)
任务 1	241	243
任务 2	230	231
任务 3	106	106

表 7　基于真实值调节与基于预测值调节两种情形下的总能耗对比

综合负载	真实值调节总能耗(J)	预测值调节总能耗(J)
任务 1	10953.1	10797.3
任务 2	11593.8	11538.7
任务 3	8591.2	8236.7

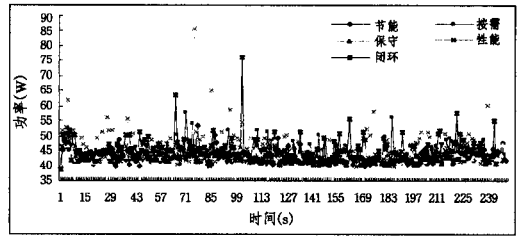
由表中数据可发现，两种情况下的执行时间相近，而完成任务所需的总能耗存在差异，3 种任务在基于预测值调节环境下得到的总能耗均低于基于真实值调节环境，分别低了 1.42%、0.48%以及 4.13%。因此，软功率计能够替代硬件功耗仪为闭环反馈控制系统提供决策依据。

4.4 闭环控制对系统能效的影响

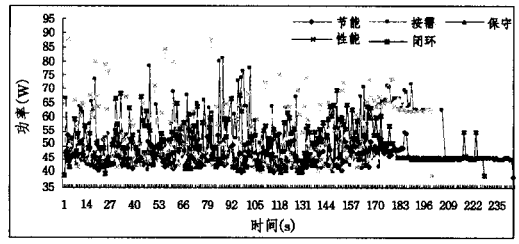
闭环控制系统的设计目的在于控制系统的峰值功率，同时最小化性能损失。另一方面，Linux 系统中的 cpufreq 模块提供了对 CPU 的频率和电压的控制功能，该模块提供了 4 种调节机制。其中一个问题是，所提功率封顶机制在性能与平均功率两个方面与其他调节机制相比有什么特点是值得关心的。

Linux 系统所提供的能效调节方案有：(1)性能(performance)：处理器一直处于最优性能状态，以最高频率执行，测试机中以 3.1GHz 持续运行；(2)节能(powersave)：与性能相反，处理器一直处于最节能状态，即持续以最低频率 1.6GHz 运行；(3)按需(ondemand)：系统根据当前 CPU 的利用率动态调节运行频率，当利用率低于预设下限时，降低频率来节能，反之，则频率升至预设值来保证性能；(4)保守(conservative)：采用类似按需的调节方式，但规定电压/频率升降必须逐级转换。若 CPU 利用率超过上限阈值，调节器调高一级频率，反之则降低一级频率；(5) 用户自定义(userspace)：用户自己提供策略控制机器运行。所提闭环反馈控制系统使用该模式来实现功率控制。

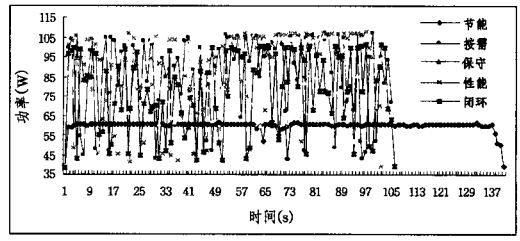
前面 3 种负载分别在 5 种控制模式下运行，其功率图谱如图 6 所示。其中，闭环模式下由软功率计代替功耗仪估计实时功率。横坐标表示时间，纵坐标表示系统功率，功率值由硬件功耗仪监测得到，即功率真实值。根据功率图谱，对负载的执行时间和总能耗两项指标进行统计分析，得到表 8 中的数据。



(a)任务 1 功率图谱



(b)任务 2 功率图谱



(c)任务 3 功率图谱

图 6　3 种任务在 5 种控制模式下的功率图谱

表 8　执行时间与总能耗两项指标的对比

控制模式	执行时间(s)			总能耗(J)		
	任务 1	任务 2	任务 3	任务 1	任务 2	任务 3
节能	249	247	141	10725.1	11600.4	8422.9
按需	248	209	105	10806	11004.4	8993
保守	248	207	106	10704	10658.5	8855.3
性能	238	202	102	10806.8	11067.9	8826
闭环	243	231	106	10797.3	11538.3	8236.7

4.4.1 性能分析与对比

观察图6(a),任务1在5种模式下的功率都比较稳定,波动范围为40～55W。任务1在节能模式下的执行时间最长为249s,性能模式和闭环模式下执行时间分别为238s和243s,其余两种均为248s。由此可知,在该环境下闭环控制模式不具有很好的性能优势。

由图6(b)以及表8中的数据可知,任务2在节能模式下运行时间最长为247s,在性能模式下执行时间最短为202s,按需模式、保守模式执行时间分别为209s和207s,闭环控制下为231s。经计算可知,闭环控制在执行时间指标上比按需模式、保守模式以及性能模式分别下降10.53%、11.59%和14.36%,而与节能模式相比则提高了6.48%。

图6(c)是任务3的功率图谱,4个核运行同样的负载,且利用率高达95%,故系统功率较高,峰值功率可达107W左右。节能模式下运行时间为141s,性能模式下时间最短为102s,按需模式、保守模式和闭环模式下运行时间相近,为106s。经计算,闭环控制在执行时间指标上比性能模式下降3.92%,但比节能模式提高了24.8%。

4.4.2 能效分析与对比

能效定义为单位能量完成的有效工作量,即能效=完成的工作量/能量。对于给定的工作负载,能耗越小,则说明能效越高。对任务1,闭环控制系统的总能耗与其余4种模式相差不大;对于任务2,其能效比节能模式提高了0.54%,而小于其他3种模式,按序分别降低4.63%,7.62%,4.08%;对于任务3,则分别提高了2.26%、9.18%、7.51%与7.15%。

综上所述,对于处理器利用率较低的情况,如任务1与任务2,所提控制算法对提高系统的能效并不明显。原因之一是该算法的目的是实施功率封顶,没有考虑其他因素;原因之二是任务1与任务2情形处理器利用率低,没有过多功率控制的余地。对于核利用率较高的情形,如任务3,闭环功率控制系统实施功率封顶的同时也减少总能耗,使得整体能效提高达6.53%,性能下降3.92%。

结束语 本文提出了数据中心数据库服务器的动态功率控制机制。通过把系统级功率模型与进程级功率模型组合形成了软功率计,并集成到闭环控制系统中,提出了功率控制算法。一系列实验表明所提控制机制能有效地控制系统的实时功率,同时也可改善能效,可应用于功率感知的DBMS服务器中。

进一步工作,可以在功率控制中考虑系统能效的因素,提高系统的能效;同时,可以结合断路器原理设计更灵活的控制算法。

参 考 文 献

[1] Agrawal R, Ailamaki A, Bernstein P A, et al. The Claremont report on database research[J]. SIGMOD Record, 2008, 37(3): 9-19

[2] Harizopoulos S, Shah M, Ranganathan P. Energy Efficiency: The New Holy Grail of Data Management Systems Research[C]// Proceedings of Conference on Innovative Data Systems Research (CIDR). 2009

[3] Raghavendra R, Ranganathan P, Talwar V, et al. No Power Struggles: A Unified Multi-level Power Management Architecture for the Data Center[C]// Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems. 2008: 48-59

[4] Felter W, Rajamani K, Keller T, et al. A performance-conserving approach for reducing peak power consumption in server systems[C]// Proceedings of International Conference on Supercomputing. 2005: 293-302

[5] Meisner D, Wenisch T F. Peak Power Modeling for Data Center Servers with Switched-Mode Power Supplies[C]// Proceedings of International Symposium on Low Power Electronics and Design, 2010: 319-324

[6] Franklin G F, Powell J D, Workman M. Digital control of dynamic systems(3rd edition)[M]. Boston: Addition-Wesley, 1997

[7] Benini L, Bogliolo A, Micheli G D. A survey of design techniques for system-level dynamic power management[J]. IEEE Transactions on Very Large Scale Integration(VLSI) Systems, 2000, 8(3): 299-316

[8] Do T, Rawshdeh S, Shi W S. pTop: A process-level power profiling tool[C]// Proceedings of the Workshop on Power Aware Computing and Systems. 2009

[9] Chen H, Li Y, Shi W S. Fine-grained power management using process-level profiling [J]. Sustainable Computing: Informatics and Systems, 2012, 2(1): 33-42

[10] Poess M, Nambiar R O. Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results[C]// Proceedings of the VLDB Conference. 2008: 1229-1240

[11] Siddha S, Pallipadi V, Ven A V D. Getting Maximum Mileage Out of Tickless[C]// Proceedings of the Linux Symposium. 2007: 201-208

[12] Tsirogiannis D, Harizopoulos S, Shah M. Analyzing the energy efficiency of a database server[C]// Proceedings of SIGMOD Conference. 2010: 231-242

[13] Lefurgy C, Wang X, Ware M. Server-level Power Control[C]// Proceedings of International Conference on Autonomic Computing. 2007: 4-13

[14] Orgerie A-C, Assuncao M, Lefevre L. A Survey on Techniques for Improving the Energy Efficiency of Large Scale Distributed Systems[J]. ACM Computing Surveys, 2014, 46(4): 1-31

[15] Venkatachalam V, Franz M. Power reduction techniques for microprocessor systems [J]. ACM Computing Survey, 2005, 37(3): 195-237

[16] Beloglazov A, Buyya R, Lee Y C, Zomaya A Y. A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems[J]. Advances in Computers, 2011, 82: 47-111

[17] 赵霞,郭耀,陈向群. 软件能耗优化技术研究进展[J]. 计算机研究与发展,2011,48(12):2308-2316

[18] 赵霞,陈向群,郭耀,等. 操作系统电源管理研究进展[J]. 计算机研究与发展,2008,45(5):817-824

[19] 叶可江,吴朝晖,姜晓红,等. 虚拟化云计算平台的能耗管理[J]. 计算机学报,2012,35(6):1262-1285

[20] Bianchini R, Rajamony R. Power and Energy Management for Server Systems[J]. IEEE Computer, 2004, 37(11): 68-76

[21] 杨良怀,朱红燕. 整机系统实时功率剖析与建模[J]. 计算机科学,2014,41(9):32-37

[22] Yang L H, Zhao Y, Fan Y, et al. Peak Power Modeling for Join Algorithms in DBMS[J]. Journal of Computer and System Sciences, 2015, 81(3): 599-614

[23] Economou D, Rivoire S, Kozyrakis C. Full-system power analysis and modeling for server environments[C]// In Workshop on Modeling, Benchmarking and Simulation. 2006