基于延迟/QoS 约束的 LC 服务动态电压/频率调节

林捷 1)

摘 要 许多云服务都有服务质量(QoS)要求;大多数请求必须在给 定的延迟约束内完成。近年来，研究者开始研究在满足 QoS 的同时， 是否可以尝试在每个请求的基础上节省电力。现有的工作表明，我们确实可以为特定的云应用程序手动离线调整请求延迟预测器，并在运行时咨询它，以调节 CPU 电压和频率，从而节省大量的电力。在这里，本文介绍了近年来基于延迟/QoS 约束的 LC 服务动态电压/频率调节（DVFS）工作。

关键词 服务质量约束； 延迟关键型服务； 动态电压/频率调节

dynamic voltage/frequency scaling for LC services with latency/QoS constraints

Jie Lin1)

Abstract Many cloud services have Quality-of-Service (QoS) requirements; most requests have to to complete within a given latency constraint. Recently, researchers have begun to investigate whether it is possible to meet QoS while attempting to save power on a per-request basis. Existing work shows that one can indeed hand-tune a request latency predictor offline for a particular cloud application, and consult it at runtime to modulate CPU voltage and frequency, resulting in substantial power savings. In this paper, we introduce the work of LC service dynamic voltage/frequency regulation (DVFS) based on delay/QoS constraints in recent years.

Key words QOS; LC service; DVFS

1 介绍

仓库规模的数据中心承载着越来越多的关键延迟(LC)交 互服务，如 websearch 和社交网络。这些服务在尾部延迟方面的严格服务质量(QoS)约束下运行，这意味着，例如，请求的第 99 百分位数需要在延迟阈值内完成，以保证可预测的性能和良好的用户体验。

除了满足尾延迟要求外，能源效率对数据中心也很重要， 因为数据中心每年都会产生数十亿美元的能源账单。为了在一定程度上考虑到负载激增和意想不到的峰值，运行 LC 应用程序的服务器通常会为最坏的情况做好准备。这意味着在正常情况下，它们被严重利用不足。这同样适用于电源管理:LC 服务通常以高频率运行，以避免延迟峰值。这既不经济也没有必要，考虑到云的充分记录的资源利用不足，需要更 细粒度的电源管理，以准确捕捉交互式、LC 服务的实时性能需求。

2 发展现状

许多工作已经探索了 LC 服务的动态电压/频率调节 (DVFS) [3][4][5][7][8]。Pegasus[7]通过监测其负载和延 迟松弛来动态调整 LC 应用的频率。不幸的是，一个关键 LC 服务的特点是它们在请求延迟上的巨大变化:一些请求慢了 几个数量级，并最终定义了应用程序的尾部延迟[2][6]。因 此，在应用程序粒度上的电源管理错

收稿日期：年-月-日；最终修改稿收到日期：年-月-日 *投稿时不填写此项*. 本课题得到… …基金中文完整名称(No.项目号)、… …基金中文完整名称(No.项目号)、… … 基金中文完整名称(No.项目号)资助.作者名1(**通信作者**)，性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号）,主要研究领域为*****、****.E-mail: **********.作者名2（**通信作者**），性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号）,主要研究领域为*****、****.E-mail: **********. 作者名3（**通信作者**），性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号）,主要研究领域为*****、****.E-mail: **********.(**给出的电子邮件地址应不会因出国、毕业、更换工作单位等原因而变动。请给出所有作者的电子邮件**)
第1作者手机号码(投稿时必须提供，以便紧急联系，发表时会删除): … …, E-mail: … …*此部分6号宋体*

过了在单个请求级别上 区分频率的机会。已经有一些工作探索了细粒度的请求级电 源管理。一些提案使用启发式[3][4]将请求分为短请求和长 请求。另一些则使用统计模型[5]来估计最坏情况下的延迟， 这往往过于保守。最近的相关工作 Gemini[10]使用特定应用 的神经网络来预测 web 搜索的请求延迟，而没有泛化到其他 LC 服务 。

最新的研究 Retail[1]，是第一个自动化的框架， 它使用实用的、简单的线性技术来实现 LC 云服务的 QoS 感知的电源管理，并使用请求级延迟预测。Retail 表明，使用适当的输入特征的简单技术比将复杂的 ML 模型视为黑箱要有效得多。我们通过七个不同的 LC 应用程序的特征来激励零 售的设计，展示了从选定的请求/应用程序特性准确推断其 请求延迟的机会。

3 具体设计

大多数提高电力效率的建议都集中在面向吞吐量的批处理工作上。最近研究具有延迟 /QoS 约束的 LC 服务 DVFS 的工作大致可分为两类: 粗粒度的电源管理、细粒度的电源管理系统。

3.1 粗粒度的电源管理系统

1) Pegasus

像 Pegasus[7]这样的粗粒度电源管理为整个 LC 应用动态调整频率。它解决了负载波动下的长期延迟变化，但通过不区分单个请求，将功耗节省留在了桌面上。

PEGASUS，这是一种基于反馈的控制器，它显著提高了 WSC 系统的能量比例，在 Google 搜索集群中的实际实现证明了这一点。PEGASUS 使用请求延迟统计信息以细粒度的方式动态调整服务器电源管理限制，使每台服务器的运行速度足以满足全局服务级别延迟目标。在大型集群实验中，PEGASUS 将功耗降低了 20%。我们还估计，分布式版本的 PEGASUS 可以将这些节省额增加近一倍。

3.2 细粒度电源管理

细粒度电源管理的性能优于粗粒度电源管理，它根据请求粒度进行区分，在满足 QoS 要求的同时，通常会提升长请求，减缓短请求。主要有两种方法:

3.2.1 基于分类的方法

基于分类的方法，使用各种指标将请求分为短请求和长请求，并在长类别中提升所有请求。

1) EETL[3]

EETL 的工作通过利用具有单核动态电压和频率缩放（DVFS）和非对称多核处理器（AMP）的服务器来解决这一冲突。EETL 引入了自适应慢到快调度框架，该框架将工作负载的异质性（短请求和长请求的混合）与以不同速度运行的硬件核心的异质性相匹配。调度器通过利用长请求本身的洞察力，将长请求优先分配给更快的内核。EETL 使用控制理论来设计基于阈值的调度策略，这些策略使用单个请求进度、负载、竞争和延迟目标来优化性能和能量。EETL 配置 EETL 的框架以优化 DVFS 和 AMP 的给定尾部延迟（EETL）的能效。在这个框架中，每个请求都会自行调度，从一个慢内核开始，然后将自己迁移到更快的内核。在高负载下，当一个请求没有所需的 AMP 核心速度 s，但有一个更快的核心速度时，s 核心类型上最长的请求会提前迁移，以便为其他请求腾出空间。与单核 DVFS 系统相比，用于 AMP 的 EETL 提供了相同的尾部延迟，将能量减少 18%至 50%，并将容量（吞吐量）提高 32%至 82%。

EETL 跟踪每个请求的进展;那些超过预定执行阈值的请求被标记为长请求， EETL 在这一点上提高了频率。然而，当一个请求达到进度阈值时，可能已经来不及防止尾部延迟退化。

2) Adrenaline[4]

Adrenaline（Adrenaline），这是一种利用更精细的粒度（10 纳秒）升压来有效控制尾部延迟的方法，具有查询级精度。这项工作有两个关键见解。首先，新出现的更精细的电压/频率提升是智能分配功率预算的一种使能机制，以仅精确提升导致尾部延迟的查询；第二，每个查询的特征可以用于设计用于主动定位这些查询的指标，从而触发相应的提升。基于这些见解，Adrenaline 可以有效地定位和增强可能会增加尾部分布的查询，并可以从电压/频率提升中获得更多好处。通过在各种工作负载配置下进行评估，我们证明了我们方法的有效性。在固定的提升功率预算下，Memcached 的尾部延迟提高了 2.50 倍，Web 搜索的延迟提高了 3.03 倍。在优化能量减少时，Adrenaline 在 Memcached 上的性能提高了 1.81 倍，在粗粒度 DVFS 上的 Web 搜索性能提高了 1.99

倍。

Adrenaline 引入了特征驱动的请求分类的概念。它研究了两个 LC 应用， web 搜索和键值存储，并通过人工检查，识别每个应用中的请求特征，这些特征可用于预测传入的请求为“短”或“长”。“肾上腺素从一开始就使用这种分类来促进长请求。不幸的是， Adrenaline 并不容易推广到其他 LC 服务，它们的功能会有所不同。此外，请求分类的一个缺点是，它无法对每个类别内的请求进行排序，因此，不够细粒度，无法准确地定位尾部的请求 ;相反，当只需要最长的请求时，整个类别的请求都得到了提升。

3.2.2 基于延迟的方法

基于延迟的方法在分类上更进一步，通过使用延迟预测来指导电源管理。最近的工作 [5][10]比基于分类的方法有了很大的改进。

1) Rubik[5]

将动态电源管理应用于延迟关键型工作负载具有挑战性。根本问题是应对其内在的短期可变性：请求在不可预测的时间到达，并且长度可变。在不了解未来的情况下，现有技术要么缓慢而保守地适应，要么依赖于特定于应用程序的启发式方法来保持尾部延迟。Rubik，一种用于延迟关键工作负载的细粒度 DVFS 方案。Rubik 通过一个新颖、通用和高效的统计性能模型来应对可变性。该模型允许 Rubik 以亚毫秒的粒度调整频率，以节省电力，同时满足目标尾部延迟。Rubik 节省了高达 66%的核心功率，大大优于现有技术，并且不需要特定于应用程序的调整。除了节省核心功率外，Rubik 还能够适应负载和系统性能的突然变化。我们使用此功能设计 RubikColoc，这是一种主机托管方案，它使用 Rubik 允许批处理和延迟关键型工作比现有技术更积极地共享硬件资源。RubikColoc 将数据中心的功耗降低了 31%，同时使用的服务器比隔离延迟关键和批处理工作的数据中心少 41%，并实现了 100%的核心利用率。

Rubik 根据当前队列长度估计每个请求的延迟分布以及请求服务时间分布 (后者是离线分析的)。然后它使用估计分布的尾部作为预测延迟。这通常太保守了，特别是当产生的延迟分布有一个长尾时。

2) Gemini[10]

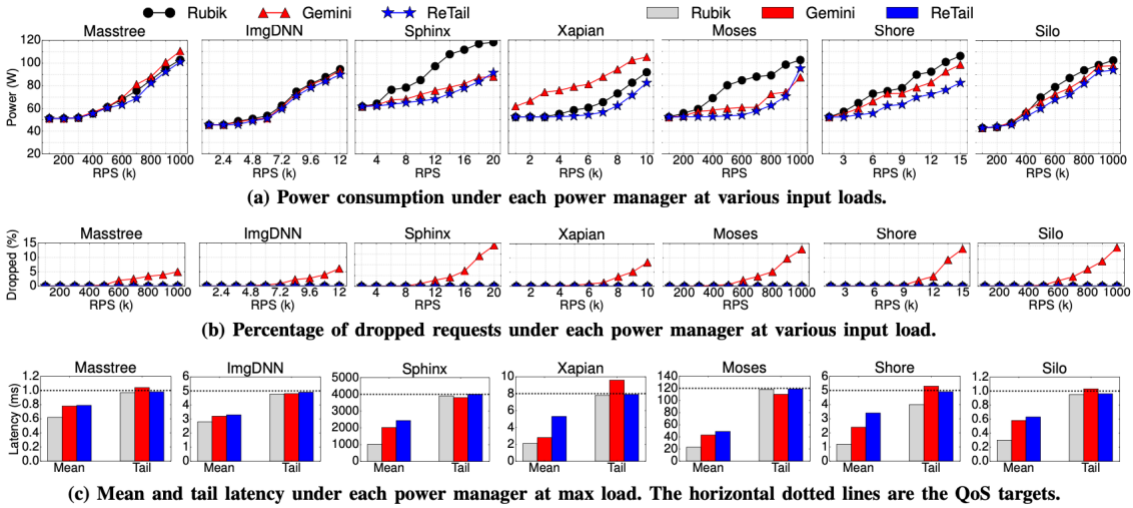
Gemini，一种用于延迟关键搜索引擎的新型电源管理框架。Gemini 有两个独特的功能来捕获每次查询的服务时间变化。首先，在没有请求排队的轻负载下，使用两步 DVFS 来管理 CPU 功率。我们的两步 DVFS 基于特定于查询的服务时间预测来选择初始 CPU 频率，然后在适当的时间明智地提高初始频率以赶上最后期限。提升时间的确定进一步依赖于估计单个查询的服务时间的预测中的误差。在高负载情况下，存在请求队列，只有当前正在执行的请求和队列中的关键请求采用两步 DVFS。中间的所有其他请求使用相同的频率以减少频率转换开销。其次，我们开发了两个独立的神经网络模型，一个用于预测服务时间，另一个用于估计预测中的误差。这两个预测因子的组合显著改善了两步 DVFS 的省电和尾部延迟结果。Gemini 是在 Solr 搜索引擎上实现的。对三个具有代表性的查询跟踪的评估表明，Gemini 节省了 41%的 CPU 功率，并且优于其他最先进的技术。

Gemini 它使用神经网络预测每个请求的延迟。 Gemini 是专门为 web 搜索引擎设计的，它的特性和预测模型都是为特定的服务精心挑选的，类似于之前的工作，也预测了 web 搜索引擎的请求延迟。它也不提供 QoS 保证，因为在某些情况下仍然会发生 QoS 冲突，例如在错误的延迟预测/高负载时。

3) Retail [1]

Retail。我们提出了一个系统的过程，以选择任何给定应用程序的特征，与其请求延迟最相关。Retail 利用这些特性预测延迟，并调整 CPU 的功耗。Retail 预测器在运行 时进行了充分的培训。我们展示了与之前的发现不同的是，当使用 正确的输入特征时，简单的技术比复杂的机器学习模型表现更好。 对于 web 搜索引擎来说，Retail 的性能优于先前基于该应用领域复杂 的手工调优预测器的机制。此外，Retail 的系统性方法还在一系列不 同 的云应用程序中产生了卓越的电力节省。

这是第一个自动化的框架， 它使用实用的、简单的线性技术来实现 LC 云服务的 QoS 感知的电源管理，并使用请求级延迟预测。Retail 表明，使用适 当的输入特征的简单技术比将复杂的 ML 模型视为黑箱要有 效得多。我们通过七个不同的 LC 应用程序的特征来激励零 售的设计，展示了从选定的请求/应用程序特性准确推断其 请求延迟的机会。Retail 有三个主要组成部



(a) Power consumption under each power manager at various input loads.

(b) Percentage of dropped requests under each power manager at various input load.

(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

图 1 Rubik，Gemini， ReTail 对比图

分：

a) 特征选择自动识别与给定服务的请求延迟最密切相关的 特征。Retail 通过包括在请求到达时无法立即获得的功能 来扩大功能空间。

b) 延迟预测使用线性回归根据所选的特征来预测请求级延 迟。与神经网络等更复杂的模型相比，线性回归准确、 简单、通用。它以明显较低的开销实现了类似的精度， 并且不需要每个应用程序进行手动调优。

c) 运行时功耗管理利用延迟预测器的输出估计不同频率下的 端到端延迟，并确定满足端到端 QoS 要求的最小请求执行频率。

表格 1 延迟关键应用电源管理提案的定性比较

	Rubik	Gemini	ReTail
Method	Statistical model	Neural net	Linear regression
Feature space	N/A	Request	Request&Application
Feature selection	N/A	Hand-picked	Systematic
Request-accurate	✗	✓	✓
General applications	✓	✗	✓
Training overhead	Low	High	Low
Inference overhead	Low	High	Low
QoS guarantee	✓	✗	✓
No dropped requests	✓	✗	✓
Adapt to model drift	✗	✗	✓

4 性能对比

表 1 显示了 Retail 方法和两种基于延迟的方法之间的主要区别。与 Rubil 和 Gemini 相比，Retail 针对一般 LC 应用，采用简单得多的线性回归进行延迟预测，自动选择对延迟影响最大的请求特征。Retail 动态检测并调整模型漂移，不需要丢弃请求。

图 1a 显示了平均功耗 ，Rubik 利用估计的延迟分布的尾部来计算目标频率。由于实 际延迟通常小于尾部，Rubik 下的 RMSE 最大，导致保守的频 率调整和节能。

Gemini 和 ReTail 通过利用请求特性来执行请求级延迟预测， 提高预测准确性，从而降低了 Rubik 的功耗。与 Rubik 相比，Gemini 有三大缺点:

1)Gemini 的电源管理算法会丢弃所有预计会错过最后期限 的请求。丢弃请求的百分比随负载呈超线性增长(图 1b)， 在最大负载时达到 16%(平均 9.2%)。由于 Gemini 搜索负载 的独特特性，多个请求共同贡献一个搜索查询;搜索聚合器 即使有一些被丢弃的请求，仍然可以形成一个响应(尽管搜 索质量有所下降)。然而，这个特性对其他 LC 应用是不可 推广性的。掉率直接影响用户体验，应该尽量保持低。 ReTail 不会删除请求。由于不需要为丢弃的请求做任何工作， 能源可以自然地节省，ReTail 有时在高负载下比 Gemini 消耗 更多的电力。例如，在狮身人面像的 RPS=20 时，Gemini 减 少了 16%的请求，即。在美国，Gemini 比 ReTail 少做大约 16% 的工作，但只消耗 3.5%的电力。在大多数情况下，Gemini 消耗更多的电力，同时也放弃了许多请求。这主要是由于 电源管理算法的两个低效。首先，Gemini 假设请求是 100% 计算密集型的;目标频率是根据估计的周期数和时间预算来 计算的。忽略无法通过调整 CPU 频率来改变的内存周期，

Gemini 倾向于高估频率，从而节省较少的功耗。其次，Gemini 的两步 DVFS 几乎对每个请求的频率进行两次调整， 一次是低初始频率，后来是高频率，以满足预测错误的最 后期限。由于功率与频率呈超线性增长，这比 ReTail 的”一 步”方法消耗更多的功率，该方法设置了单一的频率每个请求大部分时间。相反，ReTail 依赖于延迟监视器来调整 内部请求延迟目标，以消除预测错误。

2)Gemini 的特征空间只包括在请求到达时随时可用的特征。 对于需要应用程序特性的应用程序，如 Xapian，Gemini 甚至 比 Rubik 消耗更多的电力。ReTail 开放了功能空间，包括在请求到达时不一定可用的功能，从而导致 Xapian 和 Shore 的功能 与请求服务时间更好地相关，这最终导致比 Gemini 低得多的 预测误差(RMSE)。对于 Shore 来说，Gemini 和 ReTail 之间的差 异较小，因为 Shore 中只有两种请求类型需要特性，因此 Gemini 对于其他两种请求类型仍然有益。和 Shore 一样，筒 仓的两种请求类型也需要应用功能，但筒仓的三种电源管理 器之间的差异就更小了。这是因为 Si lo 的请求延迟处于亚毫 秒粒度，与请求延迟(Section VII-F)相比，调整频率的开销 (100s 微秒)是不可忽略的，这使得每个请求频率的调整效果 更差。

3)Gemini 的高推理开销(如表 IV 所示，每个请求超过 300 μ s)在具有亚毫秒级请求延迟的应用中是显著的(如表 IV 所示)。mastree 和 Silo)。对于 mastree，由于额外的推理开销， Gemini 比 Rubik 消 耗 更 多 的 能 量 。

简而言之，与 Rubik 和 Gemini 相比，ReTail 在不放弃任何请求 的情况下，分别减少了 36.2%和 35.6%(平均 11.5%和 8.9%)的 功耗。

参 考 文 献

[1]. Chen S, Jin A, Delimitrou C, et al. ReTail: Opting for Learning Simplicity to Enable QoS-Aware Power Management in the Cloud[C]//2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2022: 155-168.

[2]. J. Dean and L. A. Barroso, “The tail at scale,” Communications of the ACM, vol. 56, no. 2, pp. 74–80, 2013.

[3]. M. E. Haque, Y. He, S. Elnikety, T. D. Nguyen, R. Bianchini, and K. S. McKinley, “Exploiting heterogeneity for tail latency and energy efficiency,” in 50th International Symposium on Microarchitecture (MICRO), 2017.

[4]. C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, “Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting,” in IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), 2015.

[5]. H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, “Rubik: Fast analytical power management for latency-critical systems,” in 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2015.

[6]. J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble, “Tales of the tail: Hardware, os, and application-level sources of tail latency,” in the 2014 ACM Symposium on Cloud Computing (SoCC), 2014.

[7]. D.Lo,L.Cheng,R.Govindaraju,L.A.Barroso,andC.Kozyrakis,“Towards energy proportionality for large-scale latency-critical workloads,” in 41st International Symposium on Computer Architecture (ISCA), 2014.

[8]. D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, “Power management of online data-intensive services,” in 38th Annual International Symposium on Computer architecture (ISCA), 2011.

[9]. A. Sriraman and A. Dhanotia, “Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale,” in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 733750.

[10]. L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan, “Gemini: Learning to manage cpu power for latency-critical search engines,” in Proceedings of the 53rd IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020.