分类号_____ 学号 M201572752
学校代码 10487 密级_____

华中科技大学

硕士学位论文

面向延迟敏感型应用的
数据中心功率利用率提升方法

学位申请人: 陈 洋
学 科 专 业: 计算机系统结构
指 导 教 师: 吴 松 教授
答 辩 日 期: 2018 年 5 月 29 日

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Improving Power Usage of Datacenters Deploying Latency-Sensitive Applications

Candidate : Chen Yang

Major : Computer Architecture

Supervisor : Prof. Wu Song

Huazhong University of Science & Technology

Wuhan 430074, P.R.China

May, 2018

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，在_____年解密后适用本授权书。

不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月

摘要

数据中心已成为全球信息化建设的基础支持设施。功率供应是建造数据中心的最大限制之一，是数据中心昂贵的资源。然而部署了大量延迟敏感型应用的数据中心功率利用率低下问题一直困扰着其管理者。由于延迟敏感型应用对性能要求较高，数据中心管理者不得不为该类应用预留大量功率以满足其性能要求，这也成为数据中心功率利用率低下的主要原因之一。降低应用的功率预留量可以有效提升功率利用率，然而预留功率不足可能会带来不可接受的应用性能损失。

针对以上问题，提出了面向延迟敏感型应用的数据中心功率利用率提升方法，解决了两个问题，1）在必须满足应用性能要求的情况下，为服务器预留多少功率最为合理？2）在数据中心额定功率限制下，如何将服务器部署其中，可以有效提高数据中心功率利用率？针对问题 1 中预留功率不足时可能造成的性能损失，提出一种定量衡量性能损失的方法—细粒度微分方法，该方法基于微分任务量不变的原理，可细粒度精确评估给定预留功率阈值后可能的性能损失，能够在给定的性能要求限制下，得到最合适的预留功率。针对问题 2，采用“装箱问题”的形式，将各台服务器按照最合适的预留功率部署在数据中心中，能够在给定的额定功率限制下，保证数据中心的功率峰值控制并有效提升数据中心服务器部署密度和功率利用率。

对真实数据中心中 25000 多台服务器的历史数据验证结果表明，所提细粒度微分方法在评估应用的性能损失上准确度很高，误差仅为 5%。同时，与常用数据中心服务器部署方法对比，所提方法在给定性能要求下，可将服务器部署密度提高 12.6%，功率利用率增至 1.12 倍。

关键字：延迟敏感型应用，性能损失，数据中心，功率利用率

Abstract

Datacenter has become the basic support facility for global information construction. Power supply is one of the biggest limitations in building datacenters and is an expensive resource of datacenters. However, the problem of low power utilization of datacenter where deploying a large number of latency-sensitive applications has plagued its managers. Due to high performance requirements, datacenter managers have to reserve a large amount of power for this type of application to meet their performance requirements, which is one of the main reasons for low power utilization in datacenter. Reducing the power reservation amount can effectively improve the utilization of power. However, the lack of reserved power may lead to unacceptable performance loss.

Aiming at the above problems, a datacenter power utilization enhancement method for latency-sensitive applications is proposed and two problems are solved. 1) How much power should be reserved for the server? 2) How to deploy servers under datacenter rated power limit can effectively improve the data center power utilization? Aiming at Problem 1, a method for quantitatively measuring performance loss—fine-grained differential method is proposed. This method can accurately evaluate performance loss under the power threshold, thus get the most suitable reserved power with the given performance requirements. For problem 2, under the given rated power limit, each server with its most suitable reserved power is deployed as the “bin packing problem” to improve server density and power utilization.

The results for more than 25,000 servers in a real datacenter show that the proposed fine-grained differential method is highly accurate in assessing application performance loss with an error of only 5%. At the same time, compared with common datacenter server deployment methods, the proposed one can increase server density by 12.6% and increase power utilization by 1.12 times under a given performance requirement.

Keywords: Latency-Sensitive Applications, Performance Loss, Datacenter, Power Utilization

目　录

1 绪论

本文首先介绍研究背景，即目前数据中心功率利用率现状，之后介绍关于提高数据中心功率利用率的常用技术以及存在的问题，接着总结国内外对数据中心峰值功率控制的研究现状，最后给出论文的架构。

1.1 问题背景

随着全球信息化建设的快速推进，以及物联网的逐步普及，大量信息都以电子化的形式存储在互联网中，每天新产生的数据量还在以极高的速度增长。在庞大的数据面前，传统的数据处理与存储技术已经不能胜任，越来越多的政府部门、科研机构、商业公司等都纷纷将业务部署在数据中心中，数据中心已经成为数据处理及存储的基础支撑设施。尤其对像谷歌、亚马逊、微软、阿里巴巴等大型互联网内容提供商[1]来说，数据中心已成为数据处理与存储的唯一选择。

数据中心极其复杂，由多套系统及配套设备构成，包括计算机系统、通信和存储系统、制冷系统、冗余的备份系统以及辅助的监控、报警、安全装置。数据中心是一个多功能的“集装箱”，能容纳多个服务器以及通信设备，这些服务器及设备具有相同的对环境的要求以及物理安全上的需求而被放置在一起以便维护[2]。数据中心不仅仅是几台服务器的集合，它的建造是一项复杂的工程，耗时且造价昂贵。功率供应是数据中心最大的限制之一，平均每瓦特功率的造价在10~20 美元[3]，额定功率是建造数据中心最大的限制[4]，建造一个额定功率为 1 兆瓦的数据中心的成本近两千万美元，这还没有包括建造配套的冷却系统以及辅助设施，数据中心能耗问题[5]已成为全球学术界和工业界关注的焦点。

数据中心建造成本如此巨大，如何充分利用其能力成为重中之重。然而，目前数据中心的可用功率并不能被充分利用，造成资源的闲置，整体利用率较低。谷歌在提高数据中心功率利用率上做了很多研究，其数据中心的平均功率利用率达到 59.5%[7]。而大多数数据中心还达不到这个利用率，应该看到，数据中心功率利用率还存在巨大的提升空间。

本质上，提高数据中心功率利用率就是要提高数据中心服务器部署密度，也

即增加数据中心的服务器数量[3]。学术界与工业界的很多探索、研究都是在直接或间接的达到此目的。在数据中心额定功率限制下，运行的服务器越多，其总功率就会越大，功率利用率也就越高，与此同时，数据中心也会有更多计算能力的输出。而对云计算服务提供商来说，计算能力的输出直接关系到盈利的多少，所以提高数据中心服务器部署密度对云计算服务提供商尤为重要，既降低了平均成本，又增加了直接收益。

造成数据中心功率利用率不高的主要原因之一是目前数据中心在决定部署服务器的数量时，普遍采用计算功率预留量的方式，往往根据服务器的额定功率设置其预留功率。现在考虑一个简化的小型数据中心，假设其额定功率为 6KW，一台服务器的额定功率为 300W，如果我们根据每台服务器的额定功率为其预留功率，则该数据中心仅能部署 20 台服务器。然而一台服务器在其运行期间的绝大部分时间里是达不到额定功率的[6][7]，甚至永远达不到其额定功率。假设其大部分时间都低于 200W，这样就有 2KW 的功率长期处于闲置状态，33%的资源不能被利用。如果只为其预留 200W，这样整个数据中心就可以部署 30 台服务器。这极大提升了数据中心服务器部署密度，又能保证绝大部分时间里该功率预留量能够满足服务器运行需求。

降低每台服务器的功率预留量可以直接提高数据中心部署密度。也即通过为每台服务器设置一个功率阈值，将该服务器功率控制在该阈值之下，就能提高服务器部署密度，并能够保证不会超过数据中心额定功率限制。这在技术上也简单可行，目前已有比较成熟的服务器峰值功率控制技术，能够将服务器功率控制在给定值之下。然而不能一味的降低服务器预留量，因为服务器运行期间会有超过该预留量的风险，超过该预留量，峰值功率控制技术会启动，通过动态频率、电压调节等技术，将服务器功耗降低到预留量之下，而服务器降频的后果就是对运行在其上的应用产生性能损失。对有些应用来说，一定的性能损失可以接受，这时候可以调低一些预留量，争取更大的服务器部署密度，而对延迟敏感型应用来说，只有极小的性能损失是可以接受的，该类应用有严格的服务等级协议（Service Level Agreement, SLA），此时就不得不为其多预留些功率以满足其性能需求。

然而每台服务器运行的负载都不同，并且有着不同的性能要求，该如何个性

化地为每台服务器设置不同的阈值，并且如何衡量设置阈值后可能带来的性能损失，一直是没有解决的问题。

1.2 相关研究现状

和提升数据中心资源利用率相关的研究内容较多，大致可以分为三个研究方向，峰值功耗控制技术、负载调度方案以及由于功率控制或负载调度产生的性能损失定量衡量方法。

1.2.1 峰值功耗控制技术

峰值功耗控制技术（power capping）就是研究如何将数据中心的总功率控制在给定阈值之下，该给定值一般就是数据中心的额定功率，当然为了安全，可能要留出一定的功率余量，也即将给定阈值设置为比额定功率偏低一些。峰值功耗控制技术会在数据中心服务器部署密度已经较高且发生总功率超过给定阈值的情况下被触发，将总功率降低到给定阈值之下。峰值功耗控制技术一定程度上解决了数据中心服务器部署密度提高后带来的功率超过限额的问题，是提高数据中心资源利用率的强有力的保障。

峰值功耗控制技术在硬件上一般采用动态电压频率调节技术(DVFS)实现。DVFS[1]是根据芯片负载的不同，动态调节其频率和电压，从而降低功耗。当前市场上很多芯片都支持 DVFS，例如 Intel 公司支持 SpeedStep，ARM 公司支持“智能能耗管理”和“自适应的电压调节”等。

DVFS 过程一般通过四步完成。

1)利用监控模块记录的负载信息，计算当前系统的负载大小。

2)根据第一步计算的负载值，预测系统在将来一段时间内需要的功耗高低。

3)根据预测的功耗值，计算未来一段时间需要的频率，如果预测值高于设置的阈值，则按照阈值计算对应的频率，之后调整芯片的时钟设置。

[1] https://baike.baidu.com/item/DVFS/859664?fr=aladdin

4)根据第三步计算的频率计算相应的电压，之后通知电源模块调整 CPU 的电压。

峰值功耗控制技术可以应用在数据中心的四个层级中，数据中心级、PDU 级、机架级和服务器级，各个层级均有较多研究成果。

在介绍各个层级的研究现状前，先简单介绍一下数据中心的四个层级。图 1.1 是一个简化的数据中心层次结构图，一个数据中心包含多个功率配电单元（power distribution units, PDU），每个 PDU 又包含多个机架，服务器部署在各个机架内。数据中心接入市电，之后分配到各个 PDU，每个 PDU 的额定功率为 75~200KW，功率从 PDU 在分配到各个机架，一个 PDU 可以包含 20~60 个机架，一个典型的机架的额定功率为 3~6KW，根据服务器的额定功率不同，可以供给 10~80 台服务器（计算节点）运行[7]。

在数据中心层次，Facebook 在其数据中心中应用了 Dynamo[8]，是第一个经过真实生产环境检验的数据中心级功率管理系统[8]，该系统在其数据中心中实测了三年，是被验证的可靠的功率管理系统。Dynamo 为每个层级都设置了阈值，当某台服务器超过阈值时，会有机架监控，如果发现机架总功率没有超过为机架设置的阈值时，则支持该服务器的功率需求，否则采用 DVFS 等技术，将该服务器功耗限制到阈值之下。在机架级也是如此，如果某台机架总功率超过了给定阈值，而 PDU 级没有超过给定阈值，则 PDU 级监控进程会满足该机架的功率需求，否则限制该机架总功率到给定阈值之下。由于限制机架功率，最终要作用到服务器上，Dynamo 采用功率从高到低的方式按比例“惩罚”功耗高者。Wang 等人[9]在理论上设计了 SHIP 来动态的控制数据中心的功耗，并经过了模拟验证。数据中心层次的功耗管理系统由于层次较高，多是采用多级反馈控制的方式。比如，某台服务器的功耗超过了限制，服务器上的监控模块要上报给机架上的监控模块，机架的监控模块再上报给 PDU 的监控模块，最终上报给数据中心决策模块，由它决策该采取什么措施，再层层下达到服务器。整个环节有较大延迟并且稳定性欠缺[10]。

在 PDU 层次，[11]提出一种基于统计学的方法，可以在额定功率限制下部署更多的服务器，它的核心思想是宏观上整个数据中心数据不会同时处在计算高峰

期，数据峰值有高有低，利用这些统计特性，可以部署更多的服务器。

在机架层次，Wang 等人[12]设计了基于控制理论的方法优化系统性能；[13][14]提出了基于启发式算法的机架级功率控制方法。

而在服务器层次，则有更多的研究成果。[15-17]分别采用了不同的控制理论来控制单台服务器的功耗。Felter 等人[18]应用了开环控制理论，将功耗在处理器和内存之间来回切换将总功耗限制在功率阈值之下。Gandhi 等人[19]则通过在服务器运行期间插入停顿指令来降低功耗，以控制服务器的总功率在阈值之下。另外还有很多针对单台服务器上虚拟机功耗控制的研究。[20]提出了一种启发式方法控制虚拟机的总功耗；[21]则提出了针对虚拟机的动态功耗控制技术。

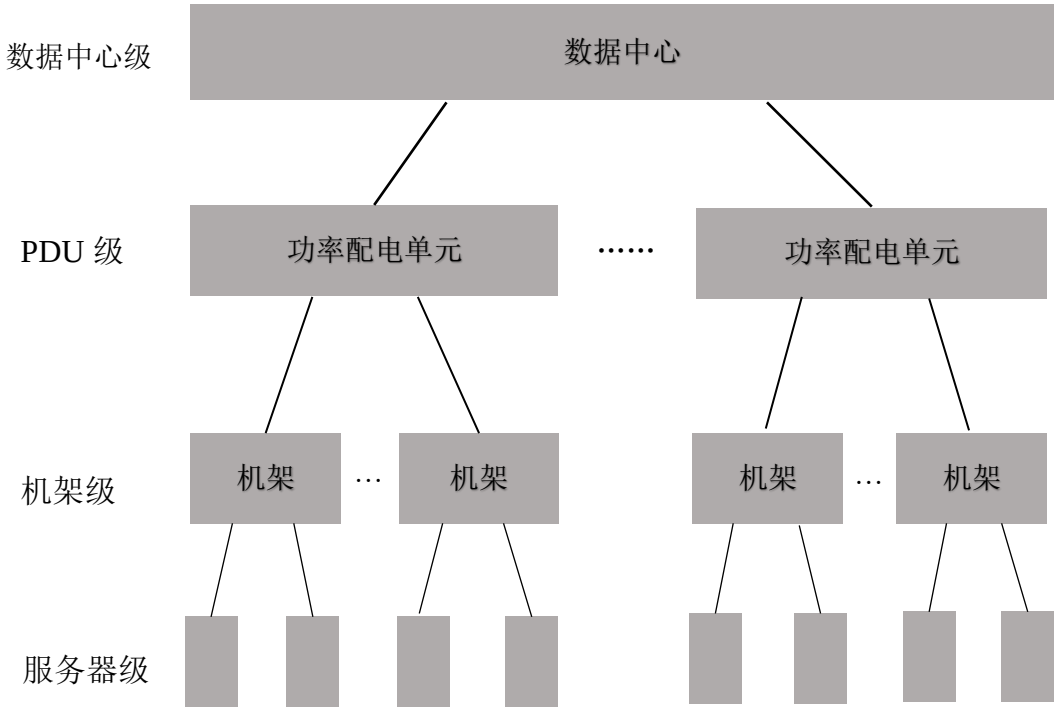


图 1.1 数据中心层次结构

当然还有附件级的功耗控制。Ma 等人[22]提出针对 CPU 核心的功耗控制技术并在多线程的应用中采用了针对每个核的 DVFS。Intel 也提出了节点管理器来支持硬件级的附件功耗控制技术[1]。

[1] http://www.intel.com/content/www/us/en/data-center/data-center-management/node-manager-general.html

另外，与峰值功耗控制技术相关的还有备用电源技术[23][24]。备用电源技术分为两种，服务器本身配置的不间断电源(Uninterruptible Power Supply , UPS)和需要额外配置的电源。该技术采用了辅助电源的形式，在数据中心超过额定功率时启用电源供电以应对突然到来的峰值。

服务器配套设置的 UPS 是为了防止市电突然断了之后，紧急的电源支持，可以保证服务器短期内不会停止工作，能够紧急保存工作进度。UPS 又分为集中式和分布式 UPS。中央式是为数据中心配置一个大的 UPS，而分布式则是根据粒度不同，为每个机架或每台服务器设置一个 UPS。Facebook 就是为机架设置 UPS，Google 是为每一个服务器设置一个 UPS。利用现有 UPS 的方法是指开发 UPS，对 UPS 进行充放电，在负载较小时利用市电对 UPS 充电保存起来，而负载较重时，再用 UPS 放电，支持服务器运行负载。这种方法的好处是利用到了平时基本闲置的 UPS 系统，也保证了负载较重时短期内也不会超过阈值，而且不会对应用的性能产生影响。缺点则是，频繁对 UPS 的充放电，会加剧 UPS 的老化，降低 UPS 寿命。这种方法同样对长而久的尖端负载没有作用。备用电源只能应对短期的峰值，一般将备用电源与功耗控制结合起来达到更好的效果。

1.2.2 负载调度

负载调度也是提功率利用率的有效手段。该方法通过将不同负载调度到一个更好的位置来提升功率利用率。举例来说，假设有三台服务器各跑了一个应用，而三台服务器 CPU 利用率都很低，这时候可以把三个应用调度到一台服务器上，使得该台服务器 CPU 利用率较高，而其他两台则可以空闲出来部署其他业务，或者直接关闭减少待机功耗。通过这种方式直接或间接的达到提升功率利用率的目的。

Guosai Wang 等人[3]通过控制往不同服务器上分发作业实现了一个动态的功率控制系统，其基本思想是监控每台服务器的功耗，将负载动态的分发到功率较低的服务器上。服务器整合技术也是提高功率利用率常用手段[25-28]。负载整合技术就是将尽量多的负载整合在尽量少的服务器上，以达到每台服务器利用率都很高的目的。这种方式在数据中心中往往会带来一些问题[29]。首先，单点故障问题

会变得越来越严重，由于一台服务器上整合了很多的应用，一旦出现故障，该台服务器上的所有应用全部瘫痪，极大增加应用运营压力。另外，服务器整合过程中带来的额外开销及不稳定性也让该方法实用性受限。

1.2.3 性能损失衡量方法

建造数据中心的目的是更好的运行业务，提升功率利用率需要在满足应用的性能要求下进行。尤其对于云计算厂商来说，保证租户应用的 SLA 是最基本的要求，也是商业往来的前提。研究如何提高功率利用率，就必须要了解在提高功率利用率过程中可能带来的应用性能损失并将它量化，之后才能够做到在满足性能要求的情况下尽量提高功率利用率。不少文献也提出了一些性能损失的衡量方法[7][21][30-33]。

[7][30]中都采用了计算应用受影响的时间比例的方式来计算性能损失。该方法的基本思想是为服务器设置一个阈值之后，统计服务器的功率超过该阈值的时间占总时间的比例。图 1.2 表示了该方法对性能损失的量化方法。该应用在t_1时刻启动，在t_2时刻超过给定功率阈值限制并一直持续到t_3，功率需求降低到阈值之下，之后运行到t_4结束运行。此方法给出的性能损失为$(t_3 - t_2)/(t_4 - t_1)$。

[21][31]则通过计算任务的最终完成时间被拖延的百分比方式量化性能损失。图 1.3 表示了该方法对性能损失的量化方法。图 1.3 中仍采用了图 1.2 中应用的功率需求示例。由于在t_2~t_3时刻对功率阈值做了限制，势必造成服务器通过降频等方式将总功率降低到阈值之下，进而造成完成总任务的时间变长。假设原来在t_4时刻能够完成的任务因功率限制拖延到t_5才执行完毕，该方法给出的量化性能损失为$(t_5 - t_4)/(t_4 - t_1)$。

从两种方法的设计中可以看出，两种方法均是针对批处理应用，没有考虑中间的任何细节，只从宏观上给出了性能损失量化值。

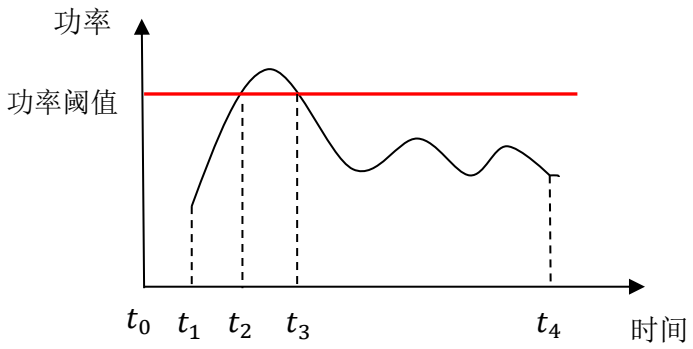


图 1.2 以应用受影响的时间比例的方式来计算性能损失。该应用性能损失为 $(t_3 - t_2)/(t_4 - t_1)$。

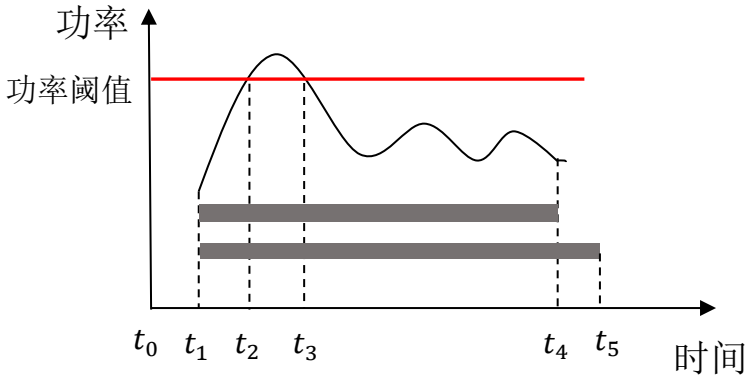


图 1.3 以任务的最终完成时间被拖延的百分比方式量化性能损失。该应用性能损失为 $(t_5 - t_4)/(t_4 - t_1)$。

1.3 课题研究内容

如前文所述，保证性能是做数据中心功率利用率提升的前提，提高数据中心服务器部署密度是提高功率利用率的根本方法。为了在满足各类应用的性能需求的前提下，尽可能提高数据中心服务器部署密度，需要有一套综合的方法，该方法需要知道每种应用负载波动的大致情况并能够量化服务器性能，从而决策出该为某台服务器预留多少功率最为合适，一般而言，该决策值应该是满足其性能需求的最低值，这样才能够保证有最大的服务器部署密度。另一方面，该方法应该能够指导服务器的部署过程，确定服务器应该部署的具体位置。而目前笔者已知的学术研究中，还没有这样一套方法，这正是本文的研究内容。

1.4 本文结构

本文大致框架如下。

第一章，绪论，介绍数据中心功率利用率低的现状，然后针对该问题总结国内外的研究现状，之后引出本文的研究内容，最后说明本文的框架结构。

第二章，功率利用率影响因素分析，详细分析了影响功率利用率的几个重要因素。

第三章，性能评估方法的设计与实现，首先介绍了细粒度微分方法的基本思想，之后给出方法设计与具体实现。

第四章，数据中心服务器部署方法，首先讲解利用细粒度微分方法求解服务器精确功率阈值，之后详细讲解服务器部署方法。

第五章，方法测试与分析，对细粒度微分方法测试，之后结合真实历史数据，在典型服务器以及大规模数据中心上分析数据中心节省功率的潜力，最后验证所提方法对提高数据中心功率利用率的有效性。

第六章，总结与展望，总结全文的研究内容，以及还存在的不足及其可能的改进方案。

最后，是致谢和参考文献。

2 功率利用率影响因素分析

本章分析一些影响数据中心功率利用率的主要因素。首先分析数据中心产生功耗的主要组成部分，之后分析了数据中心功率利用率低的主要原因—功率裕量，并分析它与提升功率利用率的具体关系。接着讨论了一种广泛的、可能导致数据中心功率利用率较低的应用—延迟敏感型应用的一些主要特征。然后讨论负载波动对功率利用率的影响。最后总结本章的主要内容。

2.1 影响功耗的主要因素

数据中心产生功耗的部件包括两大部分，IT 部件和非 IT 部件。IT 部件主要指服务器等产生计算结果的执行部件，而非 IT 部件主要指空调、水冷、机架、报警装置等提供支持服务的辅助设备。由于 IT 设备功率有较大的变化空间，本文专注于研究 IT 设备。一台典型的服务器包括 CPU、内存、硬盘、主板等耗电部件。然而很多研究表明[7][34-36]服务器 CPU 的功耗和和其总的功耗强相关，近似的存在着一个映射函数，可以拟合 CPU 利用率和总功率曲线，假设该映射函数记为 f，CPU 利用率记为 CPU，总功率记为 $power$，则$power = f(CPU)$。由于 CPU 利用率和功率间存在的映射关系，在以后的分析中，我们以分析 CPU 利用率为例，进行性能的量化分析，不再区分 CPU 利用率和功率。

2.2 功率裕量与服务器密度

谷歌通过分析自身数据中心的功耗情况，首先在其论文中揭露了数据中心广泛存在的严重的功率过度供给问题[3]。紧接着又有很多研究[7][13][31][37-39]揭露了这一严重现象。功率过度供给是指为防止可能到来的数据峰值，数据中心管理员为服务器预留的功率常常远大于其平均功耗，这直接导致了数据中心服务器部署密度的降低。图 2.1 详细说明了功率裕量的产生以及消除过程。图中(a)表示一台典型的服务器在一段时间内的功率需求曲线。(b)表示如果以服务器额定功率为其预留功率时，就有大量的功率裕量产生（图中阴影部分），该裕量导致了大量可用功率浪费，是数据中心服务器部署密度低的主要原因。(c)表示了功率裕量的消

除过程，如果为服务器设置低于额定功率的阈值，功率裕量相对于图(b)有所减少，图中标出了具体的“节省值”，该节省值可以用来提供给额外的服务器运行，从而增大数据中心服务器部署密度。图(d)是进一步压缩阈值的结果，可以看到，功率裕量被进一步消除了，这样将会有更大的服务器部署密度，然而图中同样标记出了“超限”，意思是在t_0~t_1这段时间内服务器需求功率超过阈值限制，服务器处于受限状态，应用产生性能损失。

2.3 延迟敏感型应用

延迟敏感型应用是广泛存在的、对性能要求较高的一类应用。主要包含基于网络交互的应用、在线游戏、流媒体视频网站等[41]。延迟敏感型应用是相对于批处理应用来说的。批处理应用只关心任务完成的最终时间，而对中间时刻并不关心。典型的批处理任务如 MapReduce，是离线计算任务。延迟敏感型应用通常包含了大量的请求-响应操作，一般的形式为用户从客户端发出请求，服务器处理该请求并将响应结果返回给用户。该类应用对响应延迟比较敏感，用户对每个请求的响应时间都很在意，对服务提供商有很高的要求。比如在线游戏，用户点击某个操作后，如果服务器响应较慢，用户端就会有卡顿、不流畅的体验，后果就是该游戏提供商流失大量用户。延迟敏感型应用的性能很容易受到功率限制的影响，比如一台 HTTP web 服务器发送的请求，其响应时间会因为功率的限制急剧增大[40]。

延迟敏感型应用对性能有苛刻要求，这也是应用提供商宁可闲置大量可用功率也要为运行该类应用的服务器预留大量功率的主要原因。部署大量延迟敏感型应用的数据中心的功率利用率往往偏低，由于该类应用性能要求较高，一些常用的提高数据中心功率利用率的手段都不敢轻易使用。比如负载整合技术，如果将多个延迟敏感型应用整合在一台服务器上，无疑可以提高功率利用率，但是各个延迟敏感型应用可能争抢该台服务器的资源（CPU 资源、IO 以及网络带宽等），隔离性做的不好，各个应用就会互相影响，损耗各自的性能，往往产生得不偿失的后果。

延迟敏感型应用的性能包括响应延迟、IO 吞吐量等，但该类应用尤其对响应

时间有苛刻的要求，因此本文主要以响应延迟作为衡量延迟敏感型应用的性能指标。一般来说，每种延迟敏感型应用都有一个对应的 SLA，定义了对各方面性能的要求，尤其是对于尾端延迟(tail latency)的要求[2]。尾端延迟是指如果对各个请求的响应时间做个排序的话，排在末尾的很少一部分请求的响应时间远远大于所有请求的平均时间，由于延迟敏感型应用对每一个请求都很关心，自然处在末尾的那部分请求是需要重点关注的对象。

而对有些延迟敏感型应用，其对响应延迟的严苛程度没有那么高，它容许有一定的延迟，只要该延迟在用户可接受的范围内。比如搜索引擎，用户输入关键词之后，搜索引擎在 100 毫秒内给出搜索结果，用户会感觉到很快，然而即使搜索引擎在 500 毫秒甚至 1 秒才给出结果，用户也可以接受，并不会对用户的主观感受造成太大影响。图 2.1(d)中显示的正是这种情况，如果在t_0~t_1这段时间内产生的性能损失，对整体性能影响不大，用户可以接受，则可以将阈值设置为该值，会比图 2.1(c)消除更多的功率裕量，带来更大的服务器部署密度。一个很直观的结论是，阈值设置的越低，造成的性能损失就越大，消除的功率裕量就越多，带来的功率利用率的提升就越大。

一方面，对某些应用来说，一些很小的性能损失可能带来很大的功率利用率的提升；另一方面，数据中心管理者因为延迟敏感型应用的 SLA 的约束，对可能产生的性能损失十分敏感。这里就牵涉到两方面权衡的问题。要做好这个权衡，必须设计一个能够精确量化衡量服务器性能损失的衡量方法。

2.4 负载波动规律

负载的波动特性是影响功率利用率的另一个重要因素。一般来说，负载的波动性越大，功率的利用率越低。从图 2.1(d)可以看出，如果波峰波谷差距越大，在设置了较低的阈值之后，对波峰处造成的性能损失将不可接受，波峰限制了对阈值的进一步压缩。

另一方面来说，如果能够掌握负载波动的大致变化，就可以设置一个更为合理的阈值，为保证应用的性能提供保障。

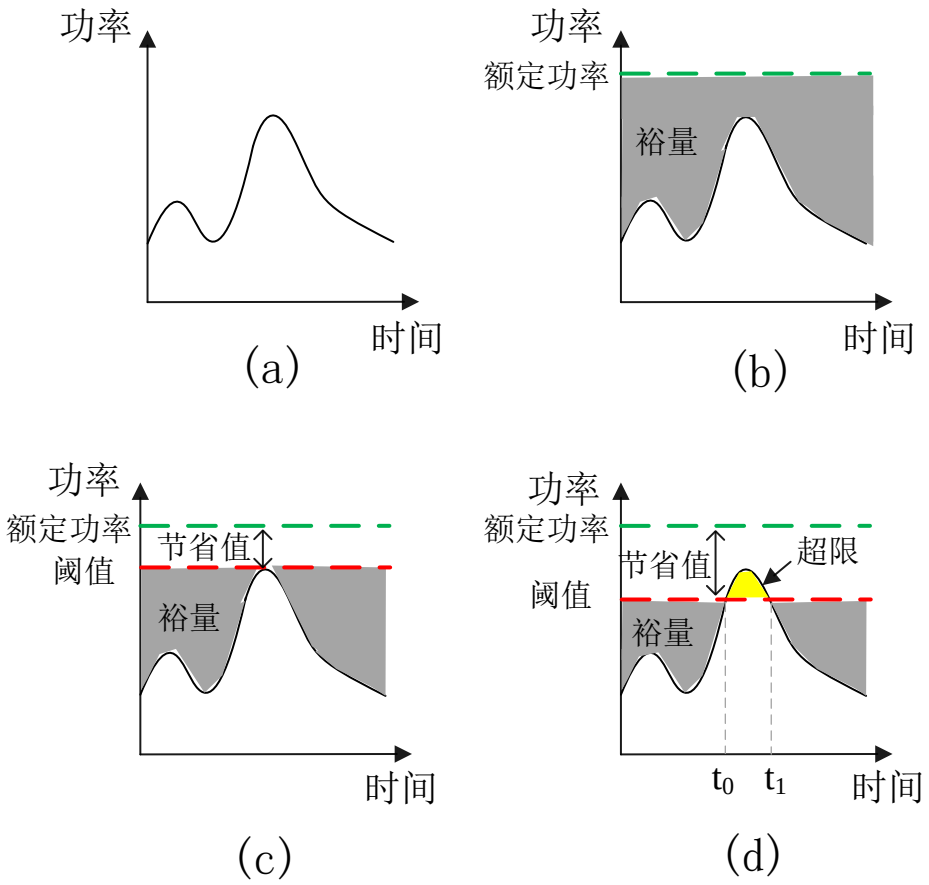


图 2.1 功率裕量的产生与清除过程

2.5 本章小结

本章分析一些影响数据中心功率利用率的主要因素。功率裕量是数据中心功率利用率低的主要原因，它的清除可以直接提升数据中心功率利用率。延迟敏感型应用是一类广泛的、可能导致数据中心功率利用率较低的应用，应该成为重点研究优化的目标。应用负载的波动对功率利用率也有决定性影响，提前了解一类应用的波动规律有助于制定更适宜的阈值。

3 细粒度微分方法

通过对功率利用率影响因素的分析，部署大量延迟敏感型应用的数据中心的功率利用率往往更低，要提高数据中心服务器部署密度，就必须要尽可能的消除功率裕量。而消除功率裕量的同时有可能带来延迟敏感型应用的性能损失。因此，提高数据中心功率利用率的关键就是要找到能够精确衡量因为服务器设置了阈值而带来的性能损失。基于此，本章讲解所提性能评估方法—细粒度微分方法（*Fine-Grained Differential Method*, FGD）的详细设计及实现。

3.1 基本思想

如绪论中分析，现存的衡量性能损失的方法多针对非延迟敏感型应用（如，批处理应用）设计，该类方法对运行过程中的细节几乎一无所知。而延迟敏感型应用必须细粒度的捕获应用运行过程中任意一个请求的响应时间，这对方法提出了很高的要求。如果把应用在运行中的每一个请求都想象为一个微元，要捕获每个微元的变化，微积分再适合不过。因此本文采用微积分的方式来获得细粒度的性能评估。

图 3.1 展示了一台服务器在一段时间内的 CPU 利用率情况。一项作业总的任务量可以用执行时间乘以 CPU 利用率来衡量，它的物理含义是需要多少个 CPU 周期来完成作业。如图 3.1，本文采用微积分的概念，把 CPU 利用率曲线覆盖的面积定义为负载量 *Workload*。该定义有两个好处。第一，微积分可以天然解决微分问题，一个请求需要消耗很少量的 CPU 资源，这在图 3.1 中正类似于曲线中的一个微元 *dt*，在 *dt* 时刻发送的请求可以被等同于 *dt* 时刻的微元参与分析，这样就满足了可以捕获运行过程中的任意请求的要求。第二，采用曲线覆盖的面积来代表负载量可以解决图 3.1 中(a)(b)所描述的问题。图 3.1(a)和(b)中，两条曲线轨迹相似，而且超过功率限制的时间相同（都在$t_1 \sim t_2$时段内），用上文描述的“计算应用受影响的时间比例的方式来计算性能损失”的方法，将得到同样的性能损失结果$(t_2 - t_1)/(t_3 - t_0)$。然而从图中可以明确的看到，(b)中超过功率限制的程度是更大的，直觉上应用所受到的性能损失也应该是更大的。方法给出相同的性

能损失结果，难以让人信服。而定义了 *Workload* 后，在t_1~t_2时段内曲线下的面积明显是不同的，$s_2 > s_1$，受影响的 *Workload* 明显更大，这与直观相符。

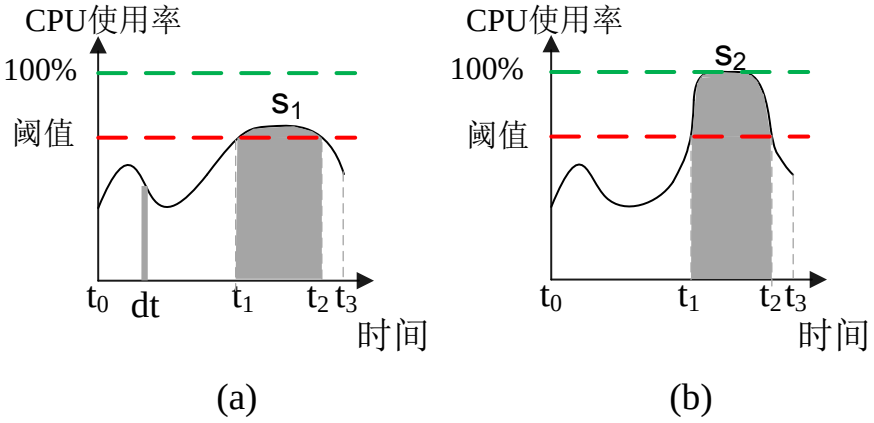


图 3.1 负载 *Workload* 的定义及作用

假设 CPU 利用率曲线为$u(t)$，则在t_1~t_2时间内的负载可以描述为：

$$Workload = \int_{t_1}^{t_2} u(t)dt \tag{3.1}$$

Workload 的定义，为细粒度衡量延迟敏感型应用的性能提供了基础。

3.2 方法设计

本章节先详细介绍几个细粒度微分方法用到的概念。

3.2.1 尾端延迟三元组

延迟敏感型应用最关心的指标就是请求的响应延迟，延迟越低性能越好。上文介绍了尾端延迟(*tail latency*)的概念，也是工业界常用的性能指标，我们就用此作为衡量性能损失的标准。顾名思义，尾端延迟包含两部分，尾端的比例和延迟时间，为了方便，本文采用二元组的形式*(请求的百分比，延迟时间)*来表示尾端延迟。举例来说，*(95%, 200ms)*表示该应用 95%的请求延迟都不高于 200 毫秒。为服务器设置了功率阈值后，可能会增加请求处理的延迟，造成性能下降。性能下降可以很明显的通过尾端延迟的二元组反映出来。比如，我们为服务器设置了功率阈值之后，该应用的性能可能会从*(95%, 200ms)*下降到*(95%, 400ms)*。通过对比设置功率阈值前后的两个二元组，就可以定量看出性能到底下降了多少。

为服务器设置功率阈值明显会对应用性能造成负面影响，功率阈值应该作为衡量性能损失的一个参数参与到设计中。因此，将二元组扩展为三元组，加入功率阈值参数，即*（请求的百分比，延迟时间，CPU 阈值）*。举例来说，*(95%, 200ms, 60%)*的意思是，在将 CPU 利用率阈值设置为 60%的情况下，95%的请求延迟都不高于 200 毫秒。当设置一个更低的 CPU 阈值时，假设为 50%，应用的性能可能从*(95%, 200ms, 60%)*下降到*(95%, 400ms, 50%)*。通过三元组就可以定量表达设置 CPU 阈值后对应用造成的性能损失。

3.2.2 性能表

细粒度微分方法的目的就是要在设置了固定的 CPU 阈值下，定量的衡量应用的性能。为了让定量结果一目了然地呈现出来，本文将三元组结果放在一张表格中，称之为性能表。

表 3.1 一个应用的性能表实例

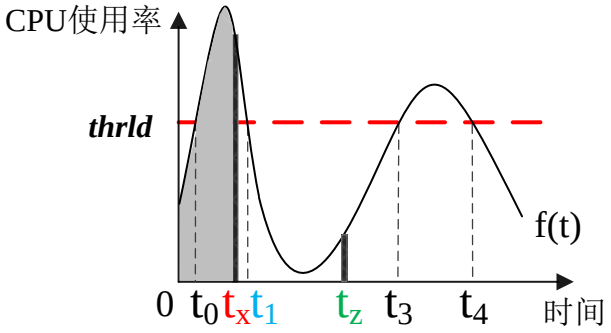
百分比 / 阈值	…	95%	…	99%	100%
…	…	…	…	…	…
31%	…	300 ms	…	800 ms	900 ms
32%	…	100 ms	…	750 ms	800 ms
…	…	…	…	…	…
99%	…	50 ms	…	100 ms	200 ms
100%	…	30 ms	…	80 ms	150 ms

表 3.1 是一个应用的性能表实例。行代表该应用的请求百分比，列代表为其设置的 CPU 利用率阈值，值代表延迟时间（单位为毫秒）。通过性能表，就可以很方便查出设置了某个阈值之后，不同百分比处的请求的延迟具体值。同时通过对比性能表中的两项，可以很容易得到应用的性能损失。得到这张表格后，所有问题都迎刃而解，那么该如何得到这张性能表呢？

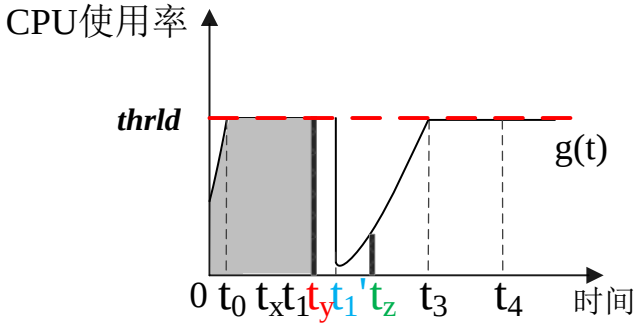
得到性能表的关键是在给定 CPU 阈值后，求出任意时刻的请求延迟。上文已经提及，对延迟敏感型应用来说，一个请求可等价于 CPU 利用率曲线上的一个微元，通过计算每个微元的延迟时间就可以得到每个请求的延迟时间。下面就分析一下如何获得每个微元的延迟。

3.2.3 微元延迟

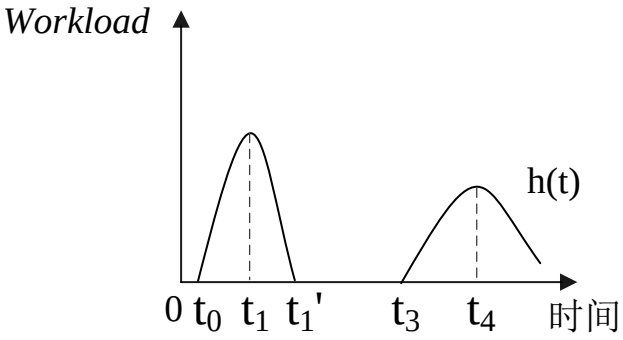
由于每个请求都需要一定的时间来处理，即使资源充足，这部分基础处理时间也不会缩短。为服务器设置功率阈值之后，在基础处理时间之上，还会产生额外的延迟。这部分额外的延迟来自于设置了功率阈值后，当服务器所需的功率超过该阈值，使用了 DVFS 等手段，让服务器功耗降低的同时也导致了其处理能力



(a) CPU 利用率需求曲线



(b) 实际 CPU 利用率曲线



(c) 积累的 *Workload* 曲线

图 3.2 微元延迟的形成与负载 *Workload* 的累积

的下降，造成请求来不及处理而排队现象。事实上，每个微元的延迟等于基础处理时间加额外排队等待时间。由于基础处理时间是必须的，可以认为是一个定值，这里所提的微元延迟，只讨论由于设置了功率阈值而额外增加的排队时间。

如图 3.2 所示，假设一台服务器在没有设置阈值时，其 CPU 利用率需求曲线 *f(t)*如(a)，为其设置 CPU 阈值后（图中虚横线 ***thrld*** 所示），服务器将在$t_0 \sim t_1$与$t_3 \sim t_4$期间功率超过 ***thrld***，触发功率限制。图 3.2(b)显示了功率限制之后的服务器真实 CPU 利用率 *g(t)*，通俗来讲，就是超过阈值 ***thrld*** 的部分被削平，部分负载被推迟到 CPU 资源充足的时候才被处理。

细看图 3.2(a)，现在考虑一个位于t_x处的微元，由于它处在$t_0 \sim t_1$期间，该阶段服务器被做了限制，CPU 处理能力不足，超过阈值的那部分负载势必需要排队等待处理，假设由于对服务器功率限制，使得原本在t_x处就能处理的微元被推迟到了 3.2(b)中的t_y处在处理，那么该微元的延迟就是$t_y - t_x$。

为了更好的衡量每个微元的延迟值，此处定义两个关键函数。

- *Delay(W_t)*

*Delay(W_t)*的含义是，在 *t* 时刻的微元 *Workload* 的延迟值。用图 3.2 中的例子来说，有$Delay(W_{t_x}) = t_y - t_x$，$Delay(W_{t_z}) = 0$，因为$t_z$处的微元没有被延迟，所以其延迟函数为 0。

- *h(t)*

*h(t)*表示负载的累积量，它的含义是因为服务器做了功率限制后，不能及时被处理而排队累积起来的负载。图 3.2(c)显示了 *h(t)*的变化过程。负载累积函数 *h(t)*是恒大于 0 的，因为物理意义上负载的累积是不可能为负的。

*h(t)*能够反映 CPU 利用率需求曲线 *f(t)*和真实 CPU 利用率曲线 *g(t)*之间实时关系。具体关系为：当 *h(t)>0*，说明此时 CPU 资源不足，有负载累积，此时 *g(t)=thrld*；当 *h(t)=0*，说明没有负载积累，CPU 资源是充足的，此时 *g(t)=f(t)*。给定 *f(t)*，要求解每个微元的延迟，必须知道 *g(t)*。而 *h(t)*充当了二者沟通的桥梁，因此 *h(t)*对求解每个微元的延迟至关重要。

*h(t)*的增长速度是由 CPU 利用率需求曲线 *f(t)*与阈值 *thrld* 的差值决定的，这很容易理解，*f(t)*与 *thrld* 差值越大，说明需求功率超过阈值越多，此时服务器对

CPU 资源的需求得不到满足的程度越严重，自然负载累积的越快。在图 3.2(a)中，负载在t_0时刻开始累积，在t_1时刻达到最大，之后由于有充足的 CPU 资源而逐渐减小。考虑到 *h(t)*不会为负值，因此 *h(t)*可由如下公式 3.2 表示。

$$h(t) = max\left\{\int_{t_0}^{t}(f(t) - thrld)\,dt, 0\right\} \tag{3.2}$$

$$\int_{t_0}^{t}(f(t) - thrld)\,dt = 0 \tag{3.3}$$

由于负载的累积，原本在图(a)中t_1之前的负载量，要推迟到图(b)中t_1'时刻处才处理完。因此在图(c)中负载累积量 *h(t)*在t_1时刻达到最大，在t_1'处重新回到 0，而在t_3又开始重复该累积过程。t_1'可由公式 3.3 计算得到，它的物理含义为负载 *h(t)*重新等于 0，即负载从累积开始再次回到零值的时刻。

如图 3.2(c)所示，*h(t)*是一个周期重复性分段函数，用公式(3.2)(3.3)可以求得其中一段的 *h(t)*表达式，并计算出该段的结束时刻（t_1'处）。因此只要重复上面的过程，就可以求出整个时间段的 *h(t)*表达式。

上文已经提及，*h(t)*能够反映 CPU 利用率需求曲线 *f(t)*和真实 CPU 利用率曲线 *g(t)*之间实时关系。具体关系为：当 *h(t)>0*，*g(t)=thrld*；当 *h(t)=0*，*g(t)=f(t)*。由于 *h(t)*已经求得，*g(t)*可用公式 3.4 表达。

$$g(t) = \begin{cases} f(t), & h(t) = 0 \\ thrld, & h(t) > 0 \end{cases} \tag{3.4}$$

负载是先来先服务的，后来的负载一定要等到前面累积的负载处理完之后，才能被处理。无论设置阈值与否，整个过程负载量是不变的，即 *f(t)*与 *g(t)*曲线下的积分面积不变。也即设置功率阈值 *thrld* 后，*t* 时刻的延迟(*Delay(t)*)必然满足公式 3.5。

$$\int_0^t f(t)dt = \int_0^{t+Delay(t)} g(t)dt \tag{3.5}$$

通过求解 3.5 中 *Delay(t)*，就可以得到任意时刻 *t* 处请求的延迟值，如公式 3.6。

$$Delay(t) = t_d \left| \int_0^t f(t)dt = \int_0^{t+t_d} g(t)dt \right. \tag{3.6}$$

至此，已经求出任意 *t* 时刻请求的延迟值 *Delay(t)*，在将所有延迟值排序，就得到在给定 *thrld* 下的服务器性能表。之后在给定另一个 *thrld*，重复该过程，就求得 CPU 阈值设置为 1%-100%时候的服务器所有性能，得到一张完整的性能表。

以上过程中，CPU 利用率曲线 *f(t)*是我们能够求解的前提条件。在真实数据中心中，都部署有监控模块，监控模块会定时采集服务器运行期间各种状态信息，包括 CPU 利用率、内存利用率、磁盘利用率等，并存储在数据库中。表 3.2 展示了 CPU 利用率历史数据示例，其可以作为离散的 *f(t)*，这就满足了我们求解的前提条件。

表 3.2 CPU 利用率采样数据示例

采样时间	CPU 利用率
12:00:00	25%
12:00:05	30%
12:00:10	58%
…	…
t_n	u_n

3.3 方法实现

上面完成了理论推导，这部分详细分析一下求解延迟敏感型应用性能表的具体流程。

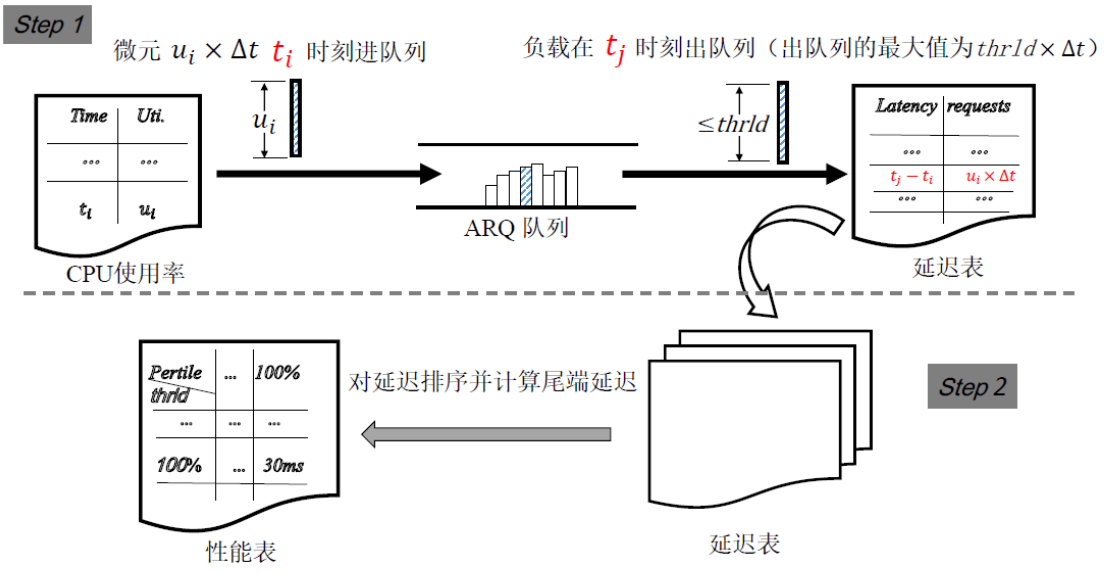


图 3.3 性能表的求解过程

求解性能表的具体流程如图 3.3 所示。求解过程包含两步。第一步，将 CPU 利用率历史数据作为数据源，给定一个 CPU 阈值，通过细粒度微分方法计算每个请求的延迟，并将结果先记录在另一张表格中，称之为延迟表。表 3.3 是延迟表的一个示例。第二步，将 CPU 阈值从 1%到 100%得到的 100 张延迟表，分别排序计算其尾端延迟，汇总成性能表。下面讲解详细过程。

表 3.3 延迟表示例

延迟值 (毫秒)	请求数
0	15000
50	7800
300	1300
500	570
700	20
…	…

- **第一步**

如图 3.3 所示，为了模拟负载的累积与消耗过程，流程中引入了一个先进先出(FIFO)的队列，记为 ARQ 队列（Accumulated Requests Queue 的简称）。给定一个 CPU 阈值 *thrld* 后，将采样的 CPU 利用率数据（i 时刻的 CPU 利用率记为u_i）转化为一个微元，其大小为$u_i \times \Delta t$，其中Δt表示采样间隔。将每个微元入队列，并记录其入队时间，与此同时，按照当前 CPU 的最大处理能力（$thrld \times \Delta t$），将在队头的微元出队列并记录其出队时间。对于特定的一个微元，其出队时间与入队时间的差值即为在队列中的排队等待延迟。一个微元需要消耗一定的资源，其可以等价于若干个请求，计算出了一个微元的延迟值，就可以认为与此微元等量的请求也具有相同的延迟值。上文已经提及，一个请求总的延迟包含了执行该请求的基本执行时间和在队列中排队等待时间。对于同一类型的请求，基本执行时间可以认为是定值，加不加该定值不会影响总延迟的尾端延迟分布。因此为简化处理过程，只考虑排队等待时间以强调做了功率限制后对服务器性能产生的影响。对于图 3.3 中 Step 1，一个微元（其值为$u_i \times \Delta t$）在t_i时刻进入队列中排队，在t_j时刻被 CPU 处理而出队列，那么该微元的延迟就是$t_j - t_i$，也即有$u_i \times \Delta t$个请求的延迟为$t_j - t_i$，之后将延迟值与对应的请求数记录到延迟表中。

- **第二步**

每给定一个阈值 *thrld*，第一步都会生成一张延迟表，而 CPU 阈值 *thrld* 的范围是从 1%变化到 100%，所以经过第一步的重复之后，总共产生了 100 张延迟表。接着，对每一张延迟表做排序，求出其尾端延迟。举个例子来说明具体的排序求解过程。如表格 3 所示，该表格已经按照延迟值做了排序，记录在表格中的总的请求数为 24690，95%请求处，也即第 23455 个请求，其延迟为 300ms。即 95%处的尾端延迟为 300ms。

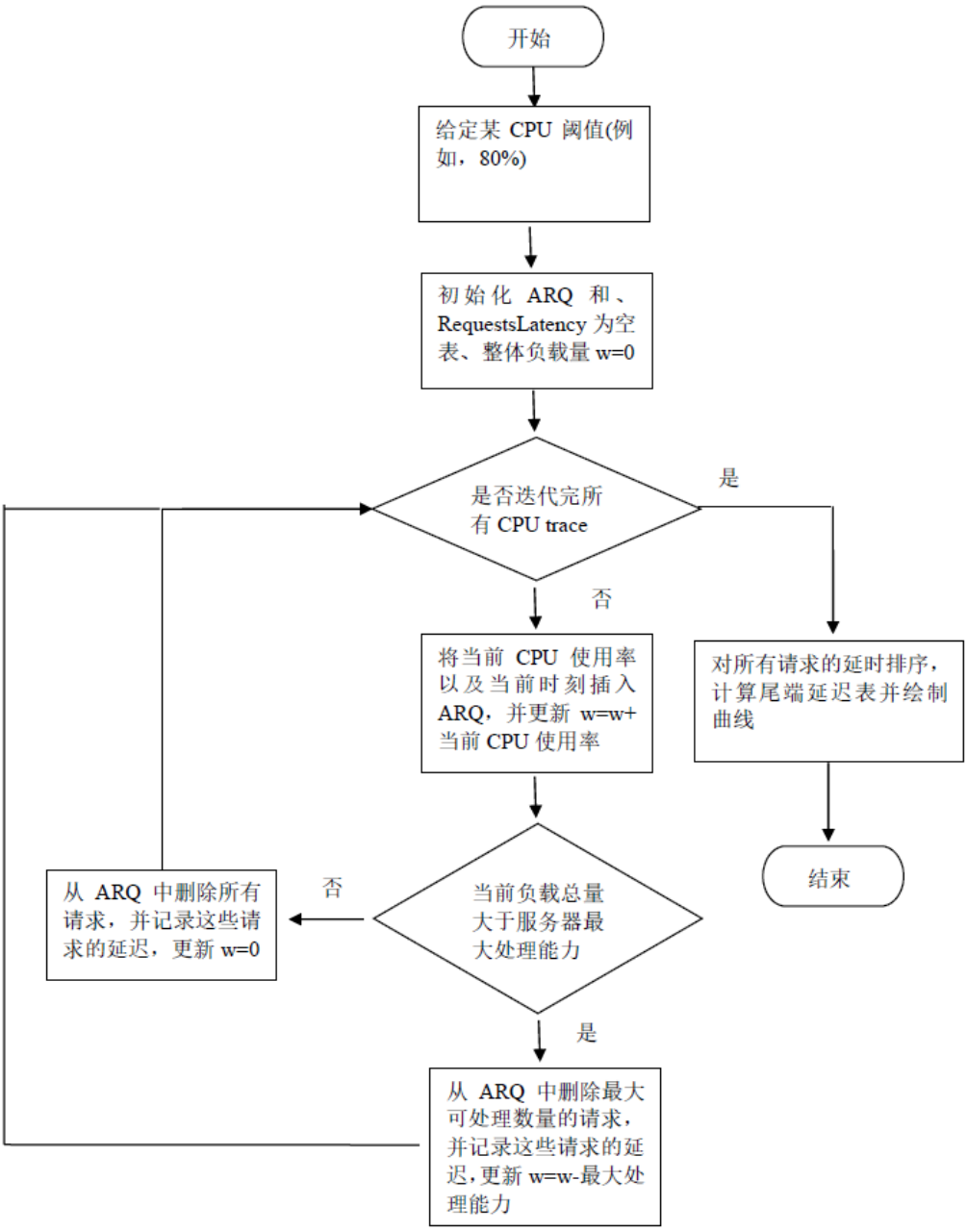


图 3.4 性能表的求解流程图

算法详细流程图见图 3.4，共包含 8 个步骤。

(1) 给定 CPU 阈值 threshold，初始化请求排队队列 *AccumulatedRequestsQueue(ARQ)*、*RequestsLatency* 表和整体负载量 w=0；

(2) 取出 i 时刻 CPU 利用率数据，将其利用率值U_i及采集的时间点 i，插入到 ARQ 中；

(3) 将整体负载量 w 增加U_i，仍记为 w，即 w = w + U_i ；

(4) 判断整体负载量 w 是否大于当前 CPU 的最大处理能力 threshold？是，进入(5)，否则进入(6)；

(5) 从 ARQ 中删除 threshold 数量的请求，并将其延迟记录在 *RequestsLatency* 中，其延迟为目前的时刻减去进入的时刻；

(6) 从 ARQ 中删除全部的请求，并将其延迟记录在 *RequestsLatency* 中，其延迟为目前的时刻减去进入的时刻；

(7) 判断采集的数据是否全部迭代完毕，是转入(8)，否则进入(2)；

(8) 将 *RequestsLatency* 根据延迟大小排序，得到所有请求的尾端延迟（tail latency）表。

详细代码展示在算法 3.1 中。

算法 3.1 性能表求解算法

输入：有n个取样点的 CPU 利用率历史数据（记为 *trace*），u_i表示第i时刻的 CPU 利用率
输出：性能表

```
1. for thrld ← 1 to 100 do
2.     初始化 ARQ 和用于记录每个请求延迟的延迟表
3.     w = 0 // 记录积累的 Workload
4.     time = 0 // 记录入队时间
5.     for u_i ∈ trace do
6.         将包含微元负载量以当前时刻的二元组（u_i * Δt, time）插入到 ARQ 中
7.         w = w + u_i * Δt
8.         /*下一行中的thrld * Δt是 CPU 在一个周期内能够处理的最大负载量 */
9.         if w > thrld * Δt then
10.            从 ARQ 中删除thrld * Δt负载量并将延迟记录在延迟表中（延迟值为当
               前时刻减去入队时刻）
11.            w = w - thrld * Δt
12.        else //负载量小于能处理的最大值，可以一次全部处理
13.            从 ARQ 中删除全部负载，并将延迟记录在延迟表中
14.            w = 0
```

```
15.            end if
16.            time = time + 1
17.        end for
18.end for
19.对每个延迟表排序并求出尾端延迟，汇总成性能表
```

- **算法复杂度**

在算法 3.1 中，每一轮计算要迭代 *n* 个采样点，一共迭代 100 轮，总的时间复杂度为 o(100n)，也即 O(n)，其中小 o 代表了总的时间复杂度，而大 O 则代表复杂度的量级。对每台服务器，算法使用了一个 ARQ 表，和 100 张延迟表来记录每个请求的延迟情况，所以其空间复杂度也为 O(n)。对延迟表的排序操作，可以使用快排，其时间复杂度为 O(nlogn)，由于可以利用桶排序的思想，借助额外的辅助空间，排序的时间复杂度可以降低到 O(n)，因此求解性能表总的时间复杂度可以控制在 O(n)水平。另外，该算法是离线的，是在线下计算好的，一旦计算好该性能表，在数据中心中给定一台服务器的 CPU 阈值后，通过查表，仅仅需要 O(1)的时间就可以找出该服务器尾端延迟情况。

3.4 本章小结

本章详细讲解了性能评估方法—细粒度微分方法的设计与实现。细粒度微分方法采用定积分任务量不变的原理，引入微元的概念，可以计算出任意时刻微元的延迟值。之后讲解了具体的方法设计，包含了尾端延迟三元组、性能表、微元延迟等概念，最后讲解了方法的具体实现，即如果具体求解性能表，包括流程图与伪代码。

4 服务器部署方法

本章首先根据细粒度微分方法求得每台服务器满足其 SLA 的精确阈值，之后介绍数据中心服务器部署算法，以最大程度的提高功率利用率、增加数据中心服务器部署密度。

4.1 服务器精确功率阈值

在大型数据中心中，每一台服务器都分配有唯一的标识，记为 SID，以 SID 为 10001 的服务器为例，如章节 3 所述，想要评估该服务器在不同的功率阈值下的性能情况，需要该服务器的 CPU 利用率历史数据（见表 3.2）和一张 CPU 利用率到功率的映射表。表 4.1 是一张 CPU 利用率到功率的映射表的示例。一般情况下，表 4.1 可以由服务器生产厂商提供。如果厂商没有提供该信息，数据中心管理员也可以通过实测的方式自己获得。

表 4.1 CPU 利用率到功率的映射表示例

CPU 利用率	功率
100%	250W
99%	248W
98%	246W
…	…
32%	180W
…	…
0%	120W(待机功率)

将表 3.2 作为算法 3.1 的输入，就可以求解出一台服务器的性能表。工业界也经常把 SLA 表述为尾端延迟的二元组形式，要求至少多少百分比的请求的延迟不能高于某个值。比如文献[29]就提到延迟敏感型应用一般对 95%的请求延迟值有要求。用户会提出他们期望的 SLA，根据该 SLA，通过查性能表，就可以得到最合适的 CPU 阈值。假设用户所提 SLA 为*(95%, 100ms)*，对应着性能表（见表 3.1）百分比中“95%”那列，从下往上找到第一个满足延迟低于 100ms 的那行，其所对应的 CPU 阈值 32%即为所求，之后在对照 CPU 利用率到功率的映射表（见表 4.1），就可以给出最适宜的功率阈值为 180W。具体求解算法见算法 4.1。

数据中心管理员将算法 4.1 应用在所有服务器上，最终为每一台服务器都得出一个精确的满足其 SLA 要求的功率阈值，汇总为一张功率阈值建议表。表 4.2 是一张服务器功率阈值建议表示例。值得一提的是，同一个用户在不同时间段可能有不同的 SLA 要求，因此示例中展示了针对同一个 SID，有多个 SLA 要求的情况，不同的 SLA 就会有不同的建议功率阈值。有了该建议表，管理员就可以为每台服务器设置精确的功率阈值。

算法 4.1 精确功率阈值求解算法

输入：CPU 利用率历史数据、CPU 利用率到功率的映射表、用户提出的 SLA
输出：适合该服务器的精确的功率阈值
1. 以 CPU 利用率历史数据作为输入，调用算法 3.1，得到性能表
2. 根据用户提出的 SLA，查找性能表，得到最合适的 CPU 利用率阈值
3. 利用步骤 2 中得到的 CPU 利用率阈值，查找 CPU 利用率到功率的映射表，给出精确的功率阈值

表 4.2 服务器功率阈值建议表示例

SID	SLA(s)	建议功率阈值
10001	*(95%, 100ms)* *(95%, 200ms)* …	180W 175W …
10002	*(99%, 500ms)* …	208W …
10003	*(99%, 500ms)* …	195W …
…	…	…

4.2 服务器部署算法

上节中已经求得了服务器功率阈值建议表，本小节提出服务器部署算法以增大服务器部署密度，提高数据中心功率利用率。

在机架里部署服务器的过程是一个典型的装箱问题[1]。经典的装箱问题是说，有若干个容量一定的箱子，怎么把一定数量的物品放入这些箱子中，物品大小不能超过箱子的容量，使得最终所用的箱子数目最少。这里带有确定容量的箱子就是有额定功率限制的机架，一定大小的物品就是有确定建议值的服务器。装箱问

[1] https://baike.baidu.com/item/%E8%A3%85%E7%AE%B1%E9%97%AE%E9%A2%98/7749220

题本身是 NP 难题，有限时间内不能完全求解。本文采用首次适应的近似算法解决服务器部署问题，首次适应算法属于贪心算法的一种，能够得到较优解，当然以后如果有了更优的算法，功率配给算法可以很容易加以替换，有良好的拓展性。

图 4.1 展示了服务器部署算法的工作流程。对于一台新服务器，由于还没有运行过具体的应用， 没有 CPU 利用率历史数据，因此服务器功率阈值建议表中不会有针对它的记录。此时采用预估的方式，先找到一台和它运行相同应用的在建议表中有记录的服务器，用已经存在的服务器的功率阈值建议值作为这台新服务器的建议值。之后搜索所有的机架，找到第一个有足够“空间”可以容纳该新服务器建议值的机架，将新服务器部署其中。将新服务器的初始化阈值记为$\overline{P_{new}}$，机架的额定功率记为$\tilde{P}$，首次适应算法应满足公式(4.1)。

$$\overline{P_{new}} + \sum_{k=1}^{n} \overline{P_k} \le \tilde{P} \tag{4.1}$$

其中$\overline{P_k}$表示第 k 台服务器的功率阈值建议值。$\sum_{k=1}^{n} \overline{P_k}$表示已经部署在该机架上的服务器的功率阈值的总和，只有新服务器的功率阈值加上已经部署的服务器阈值总和之和小于额定功率，新服务器才允许上架，这严格保证了该机架总功率不会超过额定功率的限制。

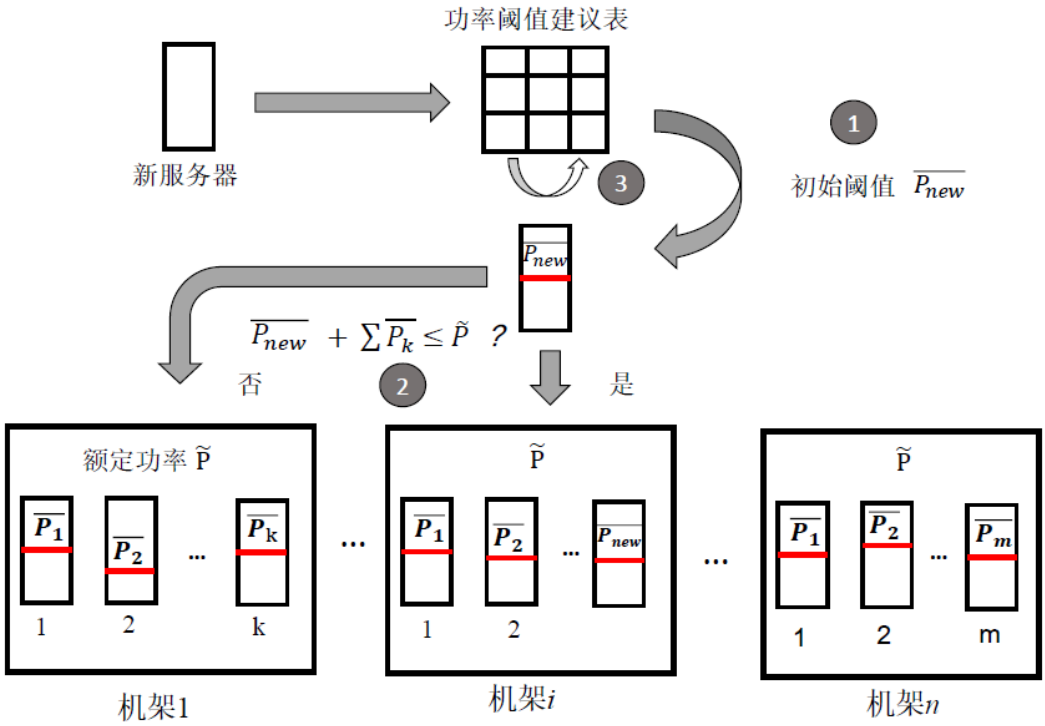


图 4.1 服务器部署算法工作流程

由于负载的特征只会在一段时间内保持较强的规律性，超过一定时间规律可能发生变化，因此未来更好的反应负载的波动特性，需要定期更新功率阈值建议表。本文的数据集全部来源于腾讯数据中心（将在下章详细说明），基于已有的数据，一周更新一次就能够较好的表达负载的波动规律。

服务器部署算法的流程图见图 4.2，共包含 3 个步骤。

(1) 根据功率阈值建议表得到运行特定应用的服务器的最优功率配额；

(2) 一台新服务器要部署时，挑选与该服务器运行相同应用的服务器，将新服务器的最优功率配额设置为与运行相同应用的服务器配额相同；

(3) 针对数据中心中所有机架，顺序判断已经在架上的服务器的配额总和加上新服务器的配额是否大于机架的额定功率，采用首次适应算法，将新服务器部署在数据中心合适的机架上。

伪代码见算法 4.2。通过算法 4.2，就能够在保证总功率不超过数据中心额定功率并且满足服务器的性能要求的前提下，提高数据中心服务器部署密度。

算法 4.2 服务器部署算法

1. 调用算法 4.1，得到服务器功率阈值建议值
2. 初始化新服务器功率阈值建议值，记为$\overline{P_{new}}$
3. **for** each *rack* in *all racks* **do**
4. **if** $\overline{P_{new}} + \sum_{k=1}^{n} \overline{P_k} \leq \tilde{P}$ **then**
5. /*首次适应算法*/
6. 将服务器部署在该 *rack*
7. **break**
8. **end if**
9. **end for**
10. 定期（比如一周）调用算法 4.1，更新服务器功率阈值建议表

4.3 本章小结

本章详细讲解了如何使用细粒度微分方法得到每台服务器精确的功率阈值，即根据性能表和用户 SLA 要求，通过查表得到精确的功率阈值，之后通过流程图及伪代码，详细讲解了采用首次适应算法的服务器部署方法。

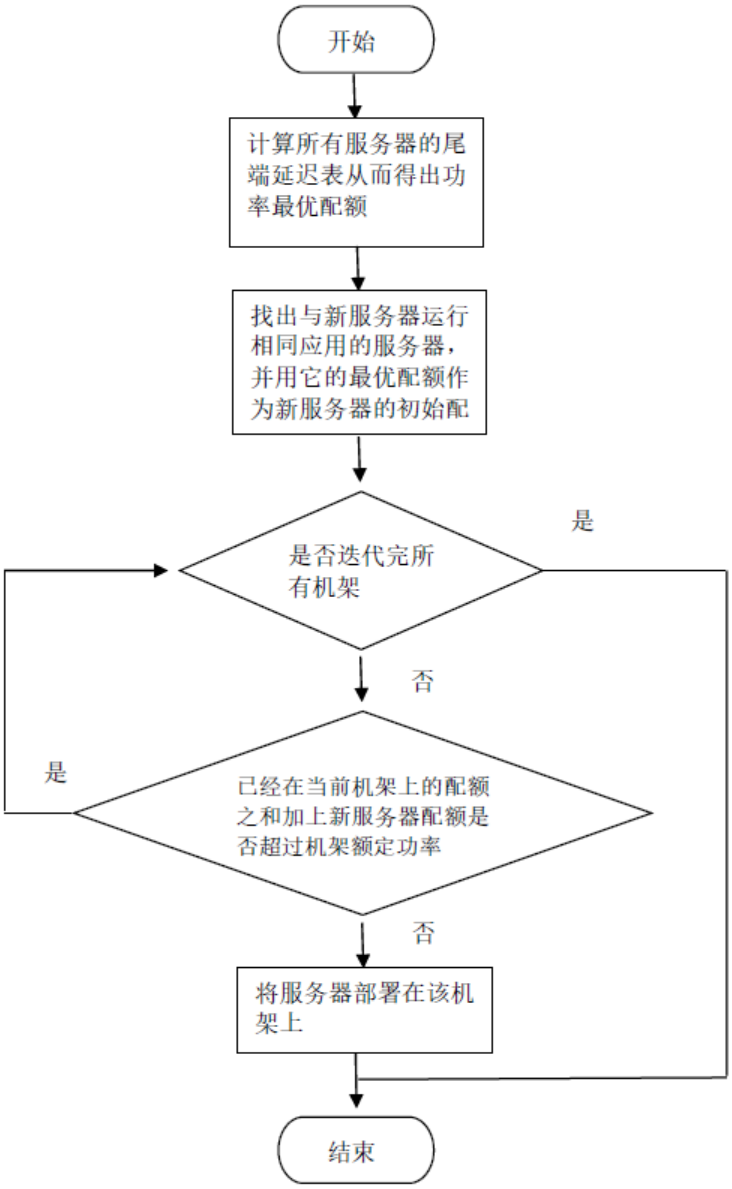


图 4.2 服务器部署算法流程图

5 测试与分析

测试章节包含四个部分，1)验证细粒度微分方法（FGD）评估性能损失的准确度；2)将细粒度微分方法应用在真实服务器上，展示其在不同的功率阈值下性能损失情况，并进一步探究如果将 FGD 应用在数据中心大量的服务器上能够有多大的功率节省空间；3)展示 FGD 及服务器部署方法在保证性能上的优势；4)最终展示该套方法能够提升多大的服务器部署密度。

5.1 细粒度微分方法的有效性

上文已经简述目前常用的两种衡量性能损失的方法，计算任务受影响的时间比例（*percentage of budget violation*, PBV）、计算任务的最终完成时间被拖延的比例（*percentage of performance loss*, PPL）。本小节将细粒度微分方法与以上两种方法的衡量结果与真实延迟测量值对比，看哪种方式能够更精确的描述服务器的真实性能损失情况。

实验硬件环境为两个 16 核的计算节点，配置为 16 × Intel(R) Xeon(R) CPU E5-2670 0@2.60GHz, 64G RAM，详细配置见表 5.1。两个计算节点分别充当客户端和 web 服务器，模拟从客户端发送请求到服务器，服务器处理之后将结果返回的过程。Web 服务器运行 Apache HTTP Server 应用，不断处理来自客户端的 HTTP 请求。试验中设置每个请求都访问一个相同的 PHP 网页，以模拟消耗一定 CPU 资源的过程。客户端则是一个 HTTP 请求的生成器，能够按照指定的速度发送 HTTP 请求，通过控制客户端的 HTTP 发送速度，就可以控制服务器端的 CPU 利用率。之后将每个请求的处理结果记录在文件里。如上文所述，每个请求都有基本的执行时间，本文首先测量该基本执行时间，用总的执行时间减去该基本执行时间作为因设置功率阈值而造成的请求排队延迟。

实验使用真实生产环境中数据集，Mini-Challenge 3[1]，该数据集记录了某服务器的网络请求、资源利用率及网络安全状态等信息。实验中仅提取 CPU 利用率信息，详细 CPU 利用率见图 5.1。

[1] http://www.vacommunity.org/VAST+Challenge+2013%3A+Mini-Challenge+3

表 5.1 实验环境参数配置表

参数名	参数值
硬件配置	CPU：Intel(R) Xeon(R) CPU E5-26700@2.60GHz 核心数：16 内存：64GB
操作系统	X86_64 centos enterprise Linux server release 6.0
编程语言	Python
真实数据集	Mini-Challenge 3，腾讯数据中心数据

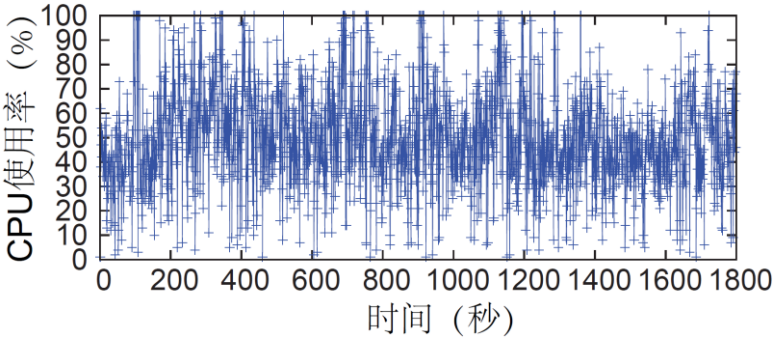


图 5.1 Mini-Challenge 3 中真实 CPU 利用率

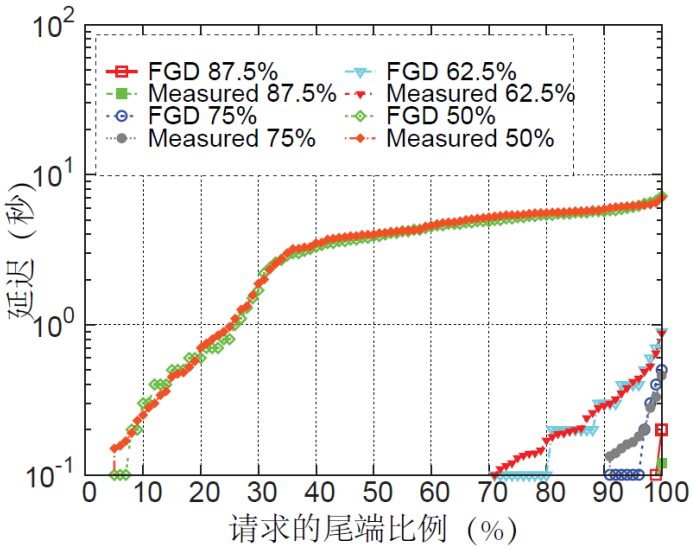


图 5.2 不同 CPU 阈值下尾端延迟的计算结果与实测结果

图 5.2 是实测结果与细粒度微分方法计算结果对比图。实验设置了四级 CPU 阈值，分别为 50%，62.5%，75%和 87.5%。图中 FGD 87.5%代表细粒度微分方法在 CPU 阈值为 87.5%时尾端延迟的计算结果；Measured 87.5%则代表 CPU 阈值为 87.5%时尾端延迟的实测结果。其他图例含义与此相同。对应于四级 CPU 阈值，计算结果与实测结果之间的误差分别为 6.9%，4.5%，3.6%和 0.5%。

进一步，图 5.3 展示了针对上述四级 CPU 阈值，实测性能损失与三种方法（FGD、

PBV、PPL）计算的性能损失对比。针对 50%，62.5%，75%和 87.5%的 CPU 阈值，实测的性能损失分别为 95.0%，30.0%，10.0%和 2.0%。FGD 给出的性能损失分别是 91.1%，29.1%，9.7%和 2.0%；PBV 给出的结果是 49.4%，26.7%，8.3%和 1.7%；PPL 给出的结果是 6.0%，0.0%，0.0%和 0.0%。可以看出，FGD 的计算结果与实测结果很接近，而其余两种方法给出的结果与实测值差距较大。整体上 FGD、PBV 和 PPL 与实测结果之间的误差为 5.0%、44.0%和 98.4%。这也说

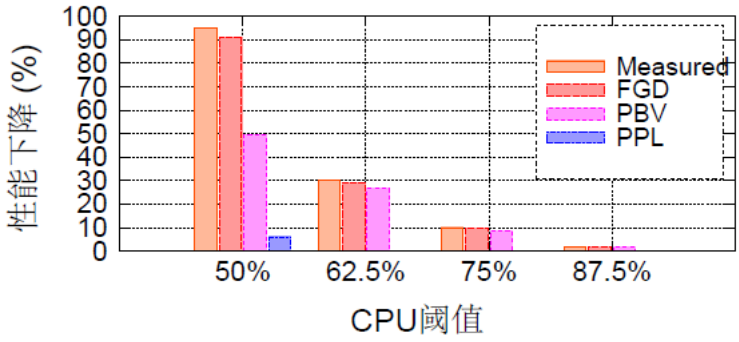


图 5.3 三种衡量性能损失方法给出的性能损失与实测性能损失对比

明了细粒度微分方法在衡量延迟敏感型应用上比其他两种方法更精确。同时该实验也验证了其他 PBV 和 PPL 两种方法并不适用于衡量延迟敏感型应用的性能损失，因为它们没有考虑应用的 SLA，而只在宏观上求了受影响或被延迟的比例。因此在后续的实验中，我们只采用细粒度微分方法也衡量性能损失，不在采用其他两种方法。

5.2 数据中心节省功率的潜力

上一节验证了细粒度微分方法的准确性，这一小节，首先介绍来自腾讯数据中心的大量真实历史数据；然后根据不同的应用类型以及负载高低选择一些有代表性的服务器展示其在不同 CPU 阈值下的性能损失情况；之后分析如何根据可接受的性能损失为这些服务器设置一个合理的 CPU 阈值。最后将 FGD 应用到数据中心 25328 台服务器上，看在大规模运行着延迟敏感型应用的数据中心中，在可接受的性能损失下，有多大的功率节省空间。

5.2.1 真实历史数据

实验数据来源于运行着大量延迟敏感型应用的腾讯数据中心真实历史数据。这些数据包含了 10166 台部署 web 应用的服务器，7824 台部署在线游戏的服务器和 7338 台部署流媒体（例如在线视频）的服务器。为方便下面的分析，此处将三种类型的应用记为 *web*, *game* 和 *stream*。该历史数据是由部署在服务器上的监控模块采集的，存储在 MySQL 数据库里。采集的数据包括 CPU 利用率、内存利用率、硬盘利用率以及网络 IO 带宽等。上文提到限制 CPU 利用率几乎与限制功率有相同的效果，因此为简化分析，本文仅提取出 CPU 利用率信息，分析 CPU 利用率对性能损失的影响。

5.2.2 在典型服务器上评估

很多研究[7][34][35]表明服务器 CPU 利用率和其总功耗强相关，可以用映射函数近似表达。本小节先探索腾讯数据中心中服务器 CPU 利用率的大致分布情况。图 5.4 展示了运行不同类型应用（*web*, *game* 和 *stream*）的服务器平均 CPU 利用率的累积分布函数（*Cumulative Distribution Function*, CDF）。

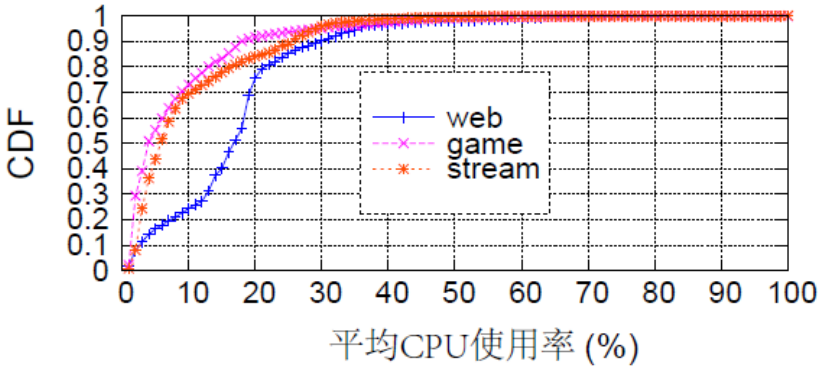
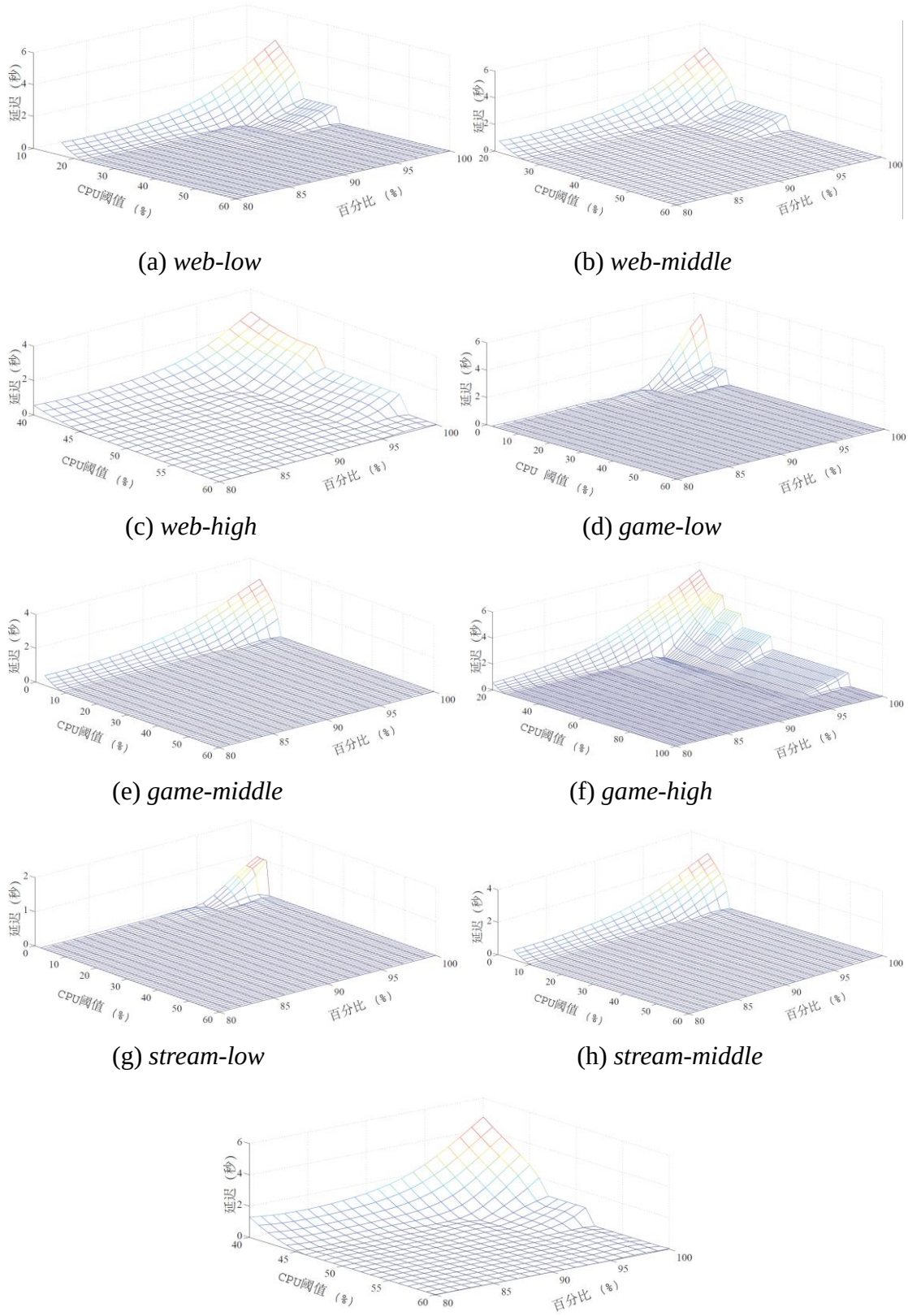


图 5.4 不同类型服务器的平均 CPU 利用率 CDF 图

从图 5.4 可以看出，对 *web* 服务器来说，80%的服务器 CPU 利用率在 20%以下，这似乎完美印证了二八原理。对 *game* 服务器和 *stream* 服务器来说，平均 CPU 利用率更低，80%的服务器平均 CPU 利用率都低于 13%。一个可能的解释是在线游戏和在线视频不是 CPU 密集型应用。

不失一般性地，本文基于应用类型和负载高低，选择九台典型的服务器，展



(a) *web-low*　(b) *web-middle*

(c) *web-high*　(d) *game-low*

(e) *game-middle*　(f) *game-high*

(g) *stream-low*　(h) *stream-middle*

(i) *stream-high*

图 5.5 九台典型服务器在不同 CPU 阈值下的尾端延迟图

示其性能表。选择的标准是，首先将服务器按照应用类型不同分为三类，*web, game, stream*。对于每一类应用，再根据平均 CPU 利用率（即负载高低）分为三类，选出平均 CPU 利用率在第 10%，40%和 70%处的服务器，分别记为 *high, middle, low*。这样选出的 9 台典型的服务器就可以分别记为：*web-high, web-middle, web-low; game-high, game-middle, game-low; stream-high, stream-middle, stream-low*。

图 5.5 展示了细粒度微分方法给出的这九台服务器在不同 CPU 阈值下的尾端延迟图，通过图 5.5，可以对性能表有一个直观的理解。举例来说，在图 5.5(b)中，当 CPU 阈值设置为 26%时，所有请求的 95%处尾端延迟为 200 毫秒。也就是说，如果该应用的 SLA 要求为(95%, 200ms)，就可以对应的设置 CPU 阈值为 26%。

在图 5.5 中，表现出来的一个最基本特征是：CPU 阈值设置的越低，请求的尾端百分比比例越大，延迟就越高。这个基本特征与主观预测完全一致。如果一个很严格 SLA 要求 99%的请求都不能受影响，也即 99%处的尾端延迟应该为 0，此时对应于图 5.5 中的服务器，*web-low, game-low, stream-low, web-middle, game-middle, stream-middle, web-high, game-high* 和 *stream-high* 分别是 33%, 15%, 6%, 40%, 10%, 12%, 57%, 87%和 52%。这些数据在管理员为运行不同应用、不同负载的服务器设置 CPU 阈值时提供了很好的建议。

通过图 5.5 可以进一步发现，在不损害应用性能的前提下，CPU 利用率阈值也有较大的压缩空间。比如对于 *web-high* 服务器（见图 5.5(c)），在没有性能损失的情况下，可以把 CPU 阈值设置为 57%；对于 *web-middle* 和 *web-low* 服务器，可以设置的更低，空闲出来的功率可以供应给额外的服务器运行。这说明在运行延迟敏感型应用的数据中心中，有很大的潜力可以增加服务器的部署密度。

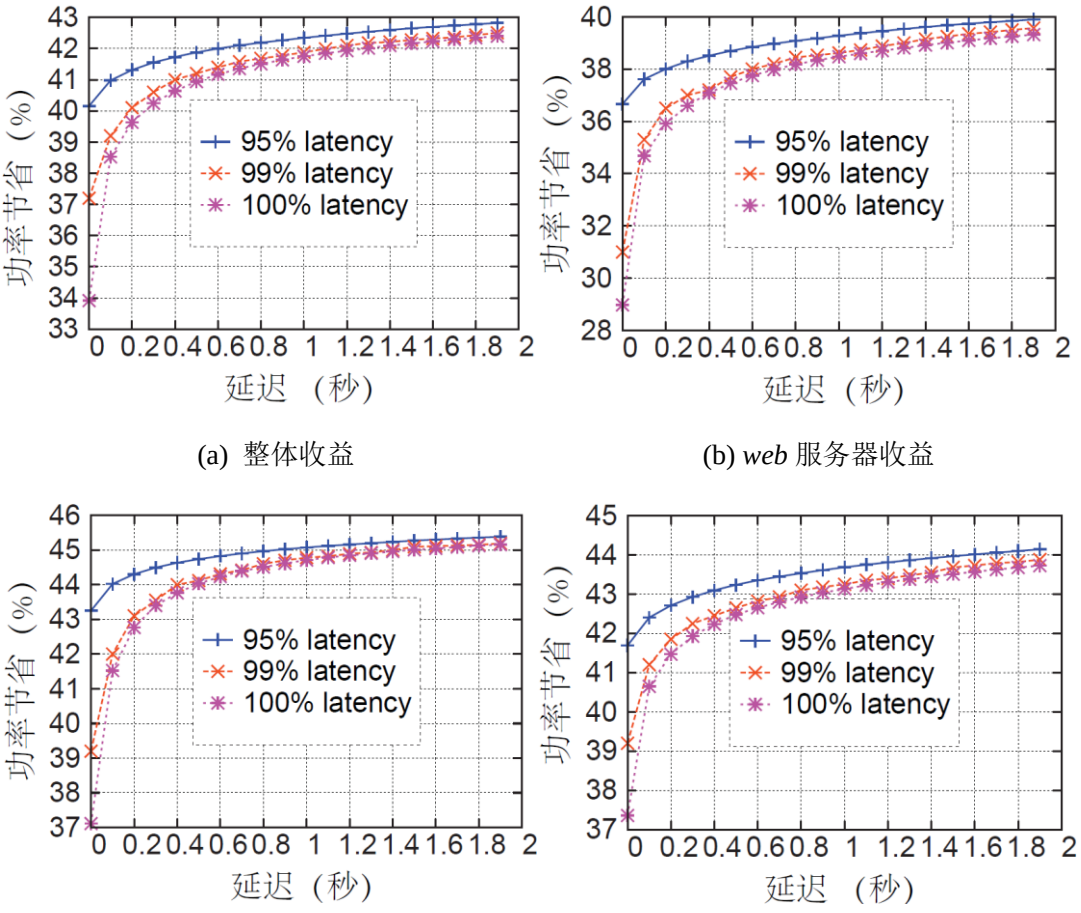
5.2.3 在大规模数据中心上评估

上一节完成了在典型服务器上的评估，证实了数据中心有很大提升服务器部署密度的潜力，这一节就具体验证一下整个数据中心有多大的功率节省空间。实验采用了腾讯数据中心中 25328 台服务器为时 7 天的历史数据。

上文提到服务器 CPU 利用率和功率直接有很强的关联，可以用映射函数近似表示，本文采用常用的线性模型[7]，如公式(5.1)。

$$p = p_{idle} + (p_{busy} - p_{idle}) \times u_i \tag{5.1}$$

其中p_{idle}和p_{busy}分别代表待机状态和满负荷运行状态，也即 CPU 利用率为 0 和 100%时对应的功率；u_i代表 i 时刻的 CPU 利用率。为了简化问题分析，文中设置p_{idle}和p_{busy}分别为 150W 和 300W[7]。在这种设置下，如果 CPU 利用率为 50%，那么它的功率 $p = 150 + (300 - 150) \times 50\% = 225$W。线性模型对于简化计算数据中心有多大功率节省空间很有用且具有较高的准确度。



(a) 整体收益　　(b) *web* 服务器收益

(c) *game* 服务器收益　　(d) *stream* 服务器收益

图 5.6 三种典型尾端比例下功率节省百分比与延迟时间关系

上文提到，针对不同的 SLA，CPU 阈值的设置是不同的，自然对于能够压缩的功率节省空间也是不同的，因此本实验选择三种典型的尾端比例，95%，99%和 100%，看看在这三种尾端比例下，随着要求的延迟值的变化，功率的节省空

间有何具体变化。图 5.6 展示了变化结果。

从图 5.6(a)可以看出，即使不允许有任何的额外延迟，也即 100%的请求额外延迟都为 0，此时也有 34%的功率节省空间，这是相当可观的。如果允许有少量的性能损失，可以获得更大的功率节省空间，比如，SLA 要求为(95%, 200ms)，此时有 41.3%的节省空间。

图 5.6 中(b)、(c)、(d)分别展示了仅仅分析运行 *web*、*game*、*stream* 单一应用服务器的收益。从图中可以看出 *web* 服务器的功率节省空间要比 *game* 和 *stream* 小一些。如果 SLA 要求为(95%, 200ms)，*web*、*game* 和 *stream* 各自能够节省的空间分别为 38.0%、44.3%和 42.7%。这反映出腾讯数据中心中 *game* 和 *stream* 应用有更大功率节省空间。

总结来说，数据中心至少有 34%的功率节省空间。如果能够接受少量的性能损失，该节省空间会进一步增大。也就是说，通过细粒度微分方法，在保证了应用性能的情况下，可以额外供应更多的服务器，增大数据中心计算能力的输出。

5.3 对应用性能的保证

这一小节分析一下细粒度微分方法在实际应用中是如何保证应用性能的。本文将所提数据中心功率利用率提升方法与当前“最先进”的 Facebook 的 Dynamo 系统[8]作对比。在实验之前，先大致介绍一下 Dynamo 系统。

Dynamo 是一个多级反馈的功率管理系统，监控着数据中心多个功率层次（服务器、机架、PDU、数据中心），它的最底层，系统称之为叶子控制器，该控制器为每种应用设置了优先级，优先级低者将被优先限制功率。比如在线游戏类应用就比批处理应用有更高的优先级，也就是说，在整个机架要超过额定功率时，将优先限制运行批处理应用的服务器功耗。而在同一个优先级内，使用了启发式的“高位桶优先”算法来决定每台服务器应该削减多少功率。Dynamo 划分桶是将服务器的功率按照从高到低每隔“一个桶大小”划分为一个桶，一般“一个桶大小”设置为 20W。举例来说，假设三台服务器功率分别为 190W，170W 和 160W，而桶记录的最大功率为 200W，那么 180~200W 的服务器都落在该桶内，该桶处于最高位，下个桶能记录的功率为 160~180W，例子中 170W 和 160W 的服务器

都将落在第二个桶中。“高位桶优先”就是指当超过额定功率需要限制服务器的功率时，优先限制在高位桶中的服务器（即功耗高的服务器）。

实验中采用三台 16 核的物理节点充当服务器，具体配置为 16×Intel(R) Xeon(R) CPU E5-2670 0@2.60GHz, 64G RAM，以模拟一台机架。为了公平对比，实验选取了三台优先级相同的 *web* 服务器，并重放他们的历史 CPU 利用率数据。图 5.7 是三台服务器随时间变化的功率情况。实验中设置该模拟机架的额定功率为 720W，*web* 应用的 SLA 设置为*(95%, 200ms)*。从图中可以看出，在大约 1700 分钟时，三台服务器的功率分别为 292.5W，234.5W，223W，总功率为 750W，超过了额定功率 30W。细粒度微分方法通过查这三台服务器的性能表，根据他们的 SLA，给出 280.5W，226.5W，213W 的功率阈值建议。也就是说，这三台服务器分别需要削减 12W，8W，10W 功率。Dynamo 采用的启发式“高位桶优先”算法，显然第一台服务器功率为 292.5W，属于最高位桶，而第二、三台服务器与一相比则属于低位桶。由于第一台服务器超过第二三台服务器几乎三个桶大小，且 Dynamo 没有采用任何性能损失评估机制，因此超过额定功率的这 30W，全部会在第一台服务器上削减。

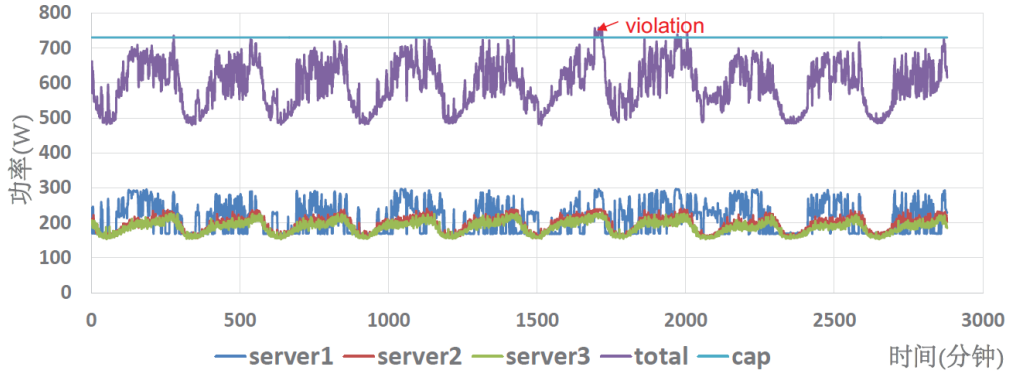


图 5.7 三台 *web* 服务器的各自功耗与总功耗波动情况

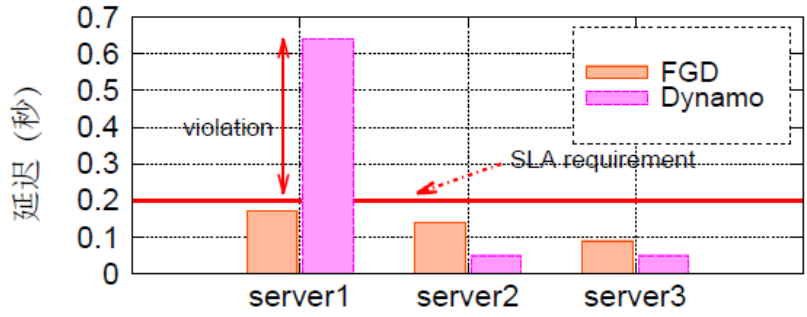


图 5.8 FGD 和 Dynamo 两种方法下三台服务器尾端 95%处的延迟值

图 5.8 展示了三台服务器在应用细粒度微分方法和 Dynamo 下，所有请求的 95%尾部比例处延迟值对比。其中虚线箭头所指横线为 SLA 要求。从图中可以看出，应用细粒度微分方法，三台服务器 95%尾部比例处延迟值均低于 200 毫秒，均满足 SLA 要求；而应用 Dynamo，由于超过额定功率的那部分功率全由第一台服务器削减，导致第一台服务器 95%尾部比例处延迟值严重违背 SLA 要求。

5.4 提高数据中心功率利用率

前文已经提到，在部署延迟敏感型应用的数据中心中，对 SLA 的保证应该优于对功率利用率的提升，保证应用的性能应该放在首位，在此基础上才能考虑对功率的更充分利用。由于 Facebook 的 Dynamo 系统没有有效措施保证延迟敏感型应用的 SLA（其采用启发式算法），虽然能够很有效的控制数据中心总功耗，其在部署大量延迟敏感型应用的数据中心中的应用中仍然是很受限的。本小节将所提整套功率利用率提升方法与能够保证延迟敏感型应用的被广泛应用的另一种常用方法—观测峰值法[10]作对比。观测峰值法是根据服务器运行期间观测到的峰值为其分配功率阈值。举例来说，如果一台服务器在过去一段时间内（例如一周），监控模块记录到的其最高达到的功率为 270W，那么就为其分配 270W 的功率阈值，如果后期超过了 270W，就使用 DVFS 等技术将其功耗降低到 270W 以下，由于 270W 已经是一段时期内观测到峰值，能够代表该服务器一段时间内负载的最高水平，后期再超过 270W 的概率是很小的，一定程度上保证了该服务器不会触发 DVFS 等限电措施，保证了服务器上应用的性能需求。这种方法比直接采用额定功率设置功率阈值要合理一些，如果观测的峰值比额定功率小很多，其能够达到的功率节省也较大。然而，一段时期内的峰值毕竟也是不常达到的，该方法仍有较大改进空间。

细粒度微分方法是为每台服务器设置一个精确的功率阈值以满足其预先签署好的 SLA 要求，并提高数据中心功率利用率及服务器部署密度。图 5.10 展示了不同 SLA 要求下，两种方法数据中心机架的平均功率利用率累积分布函数(CDF)。不失一般性地，为方便计算 CDF，实验中假设所有服务器均有相同的 SLA 要求。

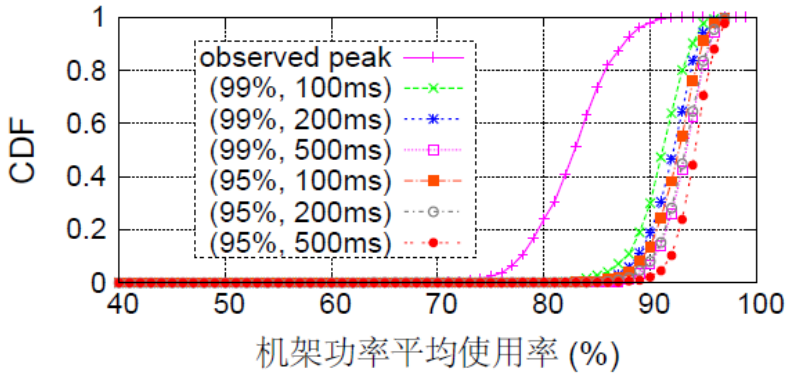


图 5.9 不同 SLA 要求下两种方法机架的平均功率利用率 CDF

从图 5.9 可以看出，观测峰值法（图中用 observed peak 标识）能够将机架的功率利用率提升到 75%~90%，而本文所提方法能够将机架功率利用率提升到 85%~96%。另外，对细粒度微分方法而言，SLA 要求越低，功率利用率能够提升的程度就越高。比如，细粒度微分方法在 SLA 要求为*(95%, 200ms)*时，其机架功率平均利用率是观测峰值法的 1.12 倍。

表 5.2 不同 SLA 要求下对比观测峰值法能够提升的服务器密度百分比

服务器类型	SLAs 要求	对比观测峰值法能够提升的服务器密度百分比
Web	*(99%, 100ms)*	9.7%
	(99%, 200ms)	11.9%
	(95%, 100ms)	13.8%
	(95%, 200ms)	14.6%
game	*(99%, 100ms)*	8.6%
	(99%, 200ms)	10.7%
	(95%, 100ms)	12.5%
	(95%, 200ms)	13.1%
stream	*(99%, 100ms)*	6.6%
	(99%, 200ms)	7.9%
	(95%, 100ms)	8.8%
	(95%, 200ms)	9.4%
entirety	*(99%, 100ms)*	8.7%
	(99%, 200ms)	10.3%
	(95%, 100ms)	12.0%
	(95%, 200ms)	12.6%

表 5.2 是采用细粒度微分方法与采用观测峰值法部署服务器对比，能够增加的服务器部署密度。从表中可以看出采用细粒度微分方法能够有效提升数据中心

服务器部署密度。例如，当 SLA 要求为*(95%, 200ms)*时，采用细粒度微分方法能够提高 12.6%的服务器部署密度，也就是说，在一个大型数据中心中，在现有额定功率不变的情况下，能够额外运行上千台新服务器。

5.5 本章小结

本章实验包含了四部分内容。

(1) 首先用真实数据集 Mini-Challenge 3 验证细粒度微分方法的有效性，实验结果证明细粒度微分方法比目前存在的衡量性能损失的方法精确很多。

(2) 用腾讯数据中心真实历史数据验证。首先介绍了数据来源与内容，之后选择了九台典型的服务器，求解其性能表，并分析实验结果，之后将细粒度微分方法应用在数据中心 25000 多台服务器上，验证了数据能够节省功率的潜力（至少 34%）。

(3) 将细粒度微分方法与 Facebook Dynamo 系统作对比，验证了细粒度微分方法在保证应用性能上的优势。

(4) 将采用细粒度微分方法的服务器部署算法应用在数据中心中，验证了该方法对数据中心服务器部署密度及功率利用率的提升的有效性。

6 总结与展望

本章先总结研究工作，在分析一些存在的不足及可能的改进措施。

6.1 研究内容总结

减少数据中心摊销成本，增加数据中心功率利用率以及计算能力的输出一直是云计算领域的热门研究内容。用少量用户能够接受的性能损失换来功率利用率的提高是简单且有效手段。SLA 规定了用户对应用性能的要求，由于云计算付费用户更关心 SLA 是否被满足，对云计算提供商来说，满足 SLA 要求应该被放在首位考虑，只是目前还没有有效的能够定量衡量延迟敏感型应用性能损失的办法，这给提升功率利用率带来了障碍。本文分析了为什么现有衡量性能损失的方法不适用于延迟敏感型应用，继而提出针对定量衡量延迟敏感型应用的细粒度微分方法，该方法采用了微积分的基本原理，能够细粒度精确评估任意时刻请求的延迟情况。在此基础上，进一步提出了基于首次适配的服务器部署方法，能够有效提高数据中心功率利用率，增大服务器部署密度。整套方案大致流程如图 6.1 中四步所示。

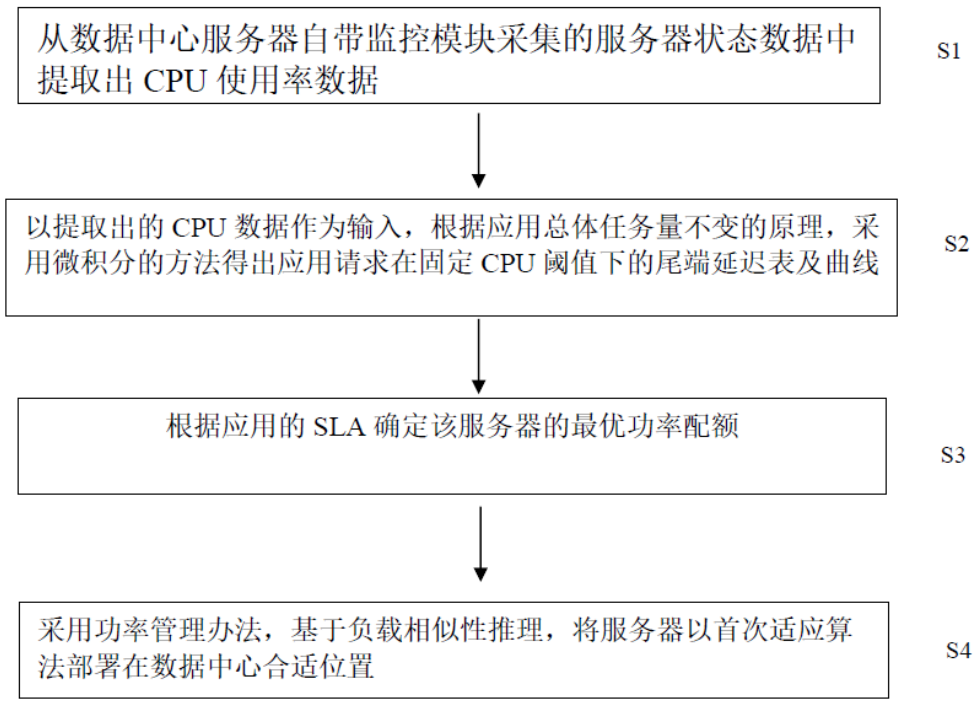


图 6.1 本文研究方法大致流程

实验中采用腾讯数据中心真实的历史数据，验证了方法的有效性，能够在保证应用性能的前提下，提升数据中心功率利用率。例如，该方案在 SLA 要求为 *(95%, 200ms)*时，能够将机架平均功率利用率提升到现有常用方案的 1.12 倍，能够将数据中心服务器部署密度提升 12.6%。

6.2 不足及改进

- CPU 历史数据采样间隔较大

细粒度微分方法引入了负载微元的感念，其大小表示为$u_i \times \Delta t$，其中u_i代表 CPU 利用率，Δt则表示 CPU 采样间隔。如果Δt过大，直观的表现就是微积分中“锯齿感”加强，对曲线下覆盖的面积描述不够精确。Δt是由数据中心监控模块决定的，采样数据以及数据的存储是笔较大的开销，Δt越小，对数据的描述就会越精确，同时带来的计算与存储开销也越大，同时密集的采样数据可能会对应用本身带来性能上的影响。这是一个需要权衡的过程。本文采用的腾讯数据中心中采样间隔为 5 分钟，5 分钟采集一次数据相对来讲是比较粗粒度的，这也是上述过程权衡之后的决定，如果能够改进采样过程，使得采样粒度更细，而不至影响应用的性能，细粒度微分方法会更精确。

- 方法依赖于历史数据

细粒度微分方法依赖于服务器的 CPU 利用率历史数据，如果采集到的历史数据不能正确的反应该类应用一段时间内的负载波动情况，或者历史数据根本拿不到，细粒度微分方法将无用武之地。实验中选取了连续 7 天作为分析，能够较好的反应腾讯数据中心负载波动情况。这个时间段针对不同的数据中心会有所不同，需要使用者去甄别。

- 对延迟不敏感的应用适用性不强

细粒度微分方法针对延迟敏感型应用设计，填补了定量衡量延迟敏感型应用方面的部分空缺，然而该方法对延迟不敏感的应用适用性不强，无法简单的往所有应用上推广使用，这是因为细粒度微分方法要捕捉任意时刻的请求延迟，而延迟不敏感应用并不关心这些，他们只关心任务的最终完成时间。

致谢

三年的研究生生活马上就要结束了，回首在东五楼的三年的时光，总结自己的收获与遗憾，在这里一并感谢所有帮助过我的人。

首先我要感谢我的导师，吴松老师。我自从大四本校保研以来，就很快进入吴老师团队参与科研工作，同时也作为我的本科毕业设计。吴老师给我的印象就是工作一丝不苟，严肃认真。学术研究上吴老师有很高的造诣，无论是大到研究方向的把控，小到论文写作一字一句的修改，都给了我极大的帮助。在论文的写作上，他尤其“苛刻”。他说写作一篇论文就是在打磨一件艺术品，是要别人来欣赏的。这句话我至今记忆犹新。记得我第一次写作时，英文逗号后应该加空格等类似的小问题，吴老师都一一指出来，这让我很感动，也让我觉得吴老师很用心的在帮我。除此之外，吴老师还组织小组成员周末打球，外出聚会等，丰富了我们的课余生活，很庆幸能遇到吴老师这样事无巨细为学生着想的导师。

其次我要感谢我的师兄们。他们是颜楚雄、王新猴、赵伟、王行军、赵新宇、甘清甜等。颜楚雄指导了我本科的毕业设计，在论文阅读和写作上都给了我很大的帮助，我研究生的研究方向也是接着他的工作继续深入研究，他完成了基础性的工作，为我后续的研究铺平了道路。王新猴在我最终的英文写作中提供了极大帮助，他帮我反复的修改了多次论文，从文章的架构，到一字一句的内容，都反复梳理，每天晚上都帮我修改论文，前后持续了一个月，对我最终论文成型帮助极大。而赵伟、王行军、赵新宇、甘清甜等师兄，则在我找工作过程中给予了很大帮助，从找实习开始，应该看什么书，有哪些面试技巧等等，事无巨细，他们都一一给了我解答，这让我后续找工作，拿 offer，都轻松了很多。有这么多热心的师兄帮助，我的整个研究生生涯少了很多坎坷，少走了很多弯路，过的轻松了很多。

其次我要感谢我实验室的同学们。他们在生活上给了我很大帮助，在繁重的科研压力下，平常一起吃饭成了最大的乐趣，我们一起讨论学术问题，一起讨论八卦，一起打球，一起出去玩，这让无聊的生活丰富了起来。其中主要有梅超、刘志毅、刘元栋、陶晟等，在日常生活中，不论学习，不论玩乐，有他们陪伴，总觉得安心。

另外，还要感谢我的父母，我的女友。他们给了我极大的支持，是我生活上的港湾，父母的鼓励是我不顺心时的动力，女友的期待是我继续努力的鞭策。有了他们的鼓舞与期待，让我决心成为更好的自己。希望在以后的生活中给予他们更多的陪伴。他们不需要我的回报，我却想回报他们。

最后，我要感谢自己，一直以来没有放纵自己，始终把学业放在首位，完成了很多困难的任务。以后我也会不忘初心，在工作中同样勤勤恳恳，任劳任怨，克服生活中不可避免的挫折与困难。相信未来不管风风雨雨，我都会充满自信地一路驰骋。

参考文献

[1] 姜誉, 方滨兴, 胡铭曾等. 大型 ISP 网络拓扑结构多点测网络拓扑结构多点测量及特征分析实例. 软件学报, 2005 , 16(5): 846 ~856 .

[2] L. A. Barroso, J. Clidaras, and U. H¨ olzle, “The datacenter as a computer: An introduction to the design of warehouse-scale machines,” Synthesis lectures on computer architecture, vol. 8, no. 3, pp. 1–154, 2013.

[3] G. Wang, S. Wang, B. Luo, W. Shi, Y. Zhu, W. Yang, D. Hu, L. Huang, X. Jin, and W. Xu, “Increasing large-scale data center capacity by statistical power control,” in proc. of EuroSys, p. 8, ACM, 2016.

[4] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi, “An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget,” in proc. of MICRO, pp. 347–358, IEEE Computer Society, 2006.

[5] 邓维, 刘方明, 金海等. 云计算数据中心的新能源应用:研究现状与趋势. 计算 机学报, 20 13, 36(3): 582 ~588 .

[6] T. Imada, M. Sato, Y. Hotta, and H. Kimura, “Power management of distributed web savers by controlling server power state and traffic prediction for qos,” in proc. of IPDPS, pp. 1–8, IEEE, 2008.

[7] X. Fan, W. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” in proc. of ISCA, vol. 35, pp. 13–23, ACM, 2007.

[8] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, “Dynamo: facebook’s datacenter-wide power management system,” in proc. of ISCA, pp. 469–480, IEEE, 2016.

[9] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller, “Ship: A scalable hierarchical power control architecture for largescale data centers,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 1, pp. 168–176, 2012.

[10] A. A. Bhattacharya, D. Culler, A. Kansal, and S. Govindan, “The need for speed and stability in data center power capping,” in proc. of IGCC, pp. 183–193, 2012.

[11] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers," in proc. of EurSys, pp. 317–330, ACM, 2009.

[12] X. Wang, M. Chen, and X. Fu, "Mimo power control for high-density servers in an enclosure," Transactions on Parallel and Distributed Systems, vol. 21, no. 10, pp. 1412–1426, 2010.

[13] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, " Ensemble-level power management for dense blade servers," in proc. of ISCA, vol. 34, pp. 66–77, IEEE Computer Society, 2006.

[14] M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in proc. of ICAC, pp. 250–261, IEEE, 2005.

[15] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," in proc. of CLUSTER, vol. 11, pp. 183–195, Springer, 2008.

[16] R. J. Minerick, V. W. Freeh, and P. M. Kogge, "Dynamic power management using feedback," in proc. of COLP, Citeseer, 2002.

[17] K. Skadron, T. Abdelzaher, and M. R. Stan, "Controltheoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," in proc. of HPCA, pp. 17–28, IEEE, 2002.

[18] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," in proc. of ICS, pp. 293–302, ACM, 2005.

[19] A. Gandhi, M. Harchol-Balter, R. Das, J. O. Kephart, and C. Lefurgy, "Power capping via forced idleness," in proc. of WEED, 2009.

[20] R. Nathuji, K. Schwan, A. Somani, and Y. Joshi, "Vpm tokens: virtual machine-aware power budgeting in datacenters," in proc. of CLUSTER, vol. 12, pp. 189–203, Springer, 2009.

[21] H. Chen, C. Hankendi, M. C. Caramanis, and A. K. Coskun, “Dynamic server power capping for enabling data center participation in power markets,” in proc. of ICCAD, pp. 122–129, IEEE Press, 2013.

[22] K. Ma and X. Wang, “Pgcapping: exploiting power gating for power capping and core lifetime balancing in cmps,” in proc. of PACT, pp. 13–22, ACM, 2012.

[23] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, “Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters,” ACM SIGARCH Computer Architecture News, vol. 40, no. 1, pp. 75–86, 2012.

[24] Y. Li, D. Wang, S. Ghose, J. Liu, S. Govindan, S. James, E. Peterson, J. Siegler, R. Ausavarungnirun, and O. Mutlu, “Sizecap: Efficiently handling power surges in fuel cell powered data centers,” in proc. of HPCA, pp. 444–456, IEEE, 2016.

[25] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, “Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations,” in proc. of MICRO, pp. 248–259, ACM, 2011.

[26] L. Tang, J. Mars, and M. L. Soffa, “Compiling for niceness: Mitigating contention for qos in warehouse scale computers,” in proc. of CGO, pp. 1–12, ACM, 2012.

[27] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, “Jettison: efficient idle desktop consolidation with partial vm migration,” in proc. of the 7th ACM EurSys, pp. 211–224, ACM, 2012.

[28] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, “Dynamic right-sizing for power-proportional data centers,” IEEE/ACM Transactions on Networking, vol. 21, no. 5, pp. 1378–1391, 2013.

[29] D. Meisner, C. M. Sadler, L. A. B., W. Weber, and T. F. Wenisch, “Power management of online data-intensive services,” in proc. of ISCA, pp. 319–330, IEEE, 2011.

[30] X. Wang and M. Chen, “Cluster-level feedback power control for performance optimization,” in proc. of HPCA, pp. 101–110, IEEE, 2008.

[31] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No power struggles: Coordinated multilevel power management for the data center,” in proc. of ASPLOS, vol. 36, pp. 48–59, ACM, 2008.

[32] C. Delimitrou and C. Kozyrakis, “Paragon: Qos-aware scheduling for heterogeneous datacenters,” proc. of MICRO, vol. 34, pp. 17–30, 2014.

[33] B. Su, J. L. Greathouse, J. Gu, M. Boyer, L. Shen, and Z.Wang, “Implementing a leading loads performance predictor on commodity processors,” in proc. of ATC, pp. 205–210, USENIX Association, 2014.

[34] S. Rivoire, P. Ranganathan, and C. Kozyrakis, “A comparison of high-level full-system power models.,” HotPower, vol. 8, pp. 3–3, 2008.

[35] D. Meisner and T. F. Wenisch, “Peak power modeling for data center servers with switched-mode power supplies,” in proc. of ISLPED, pp. 319–324, ACM, 2010.

[36] 罗亮, 吴文峻, 张飞. 面向云计算数据中心的能耗建模方法.软件学报, 2014, 25(7): 1371 ~1387.

[37] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, “Optimal power allocation in server farms,” in proc. of SIGMETRICS, vol. 37, pp. 157–168, ACM, 2009.

[38] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, “Power routing: dynamic power provisioning in the data center,” in proc. of ASPLOS, vol. 45, pp. 231–242, ACM, 2010.

[39] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing, “Managing distributed ups energy for effective power capping in data centers,” in proc. of ISCA, pp. 488–499, IEEE, 2012.

[40] S. Wang, J. Chen, J. Liu, and X. Liu, “Power saving design for servers under response time constraint,” in proc. of ECRTS, pp. 123–132, IEEE, 2010.

[41] A. Li, X. Yang, S. Kandula, and M. Zhang, “Cloudcmp: comparing public cloud providers,” in proc. of IMC, pp. 1–14, ACM, 2010.

附录 1 攻读硕士期间申请的国家发明专利

[1] 一种数据中心功率管理及服务器部署方法，专利申请号 201810037874.2，专利申请日 2018 年 2 月 26 日，吴松，陈洋，王新猴，金海。

附录 2 攻读硕士期间参与的项目

[1] 国家高技术研究发展计划（863 计划），云计算关键技术与系统（三期）云端和终端资源自适应协同与调度平台，2016.7-2018.4