RAPID：在动态公有云环境中实现快速在线策略学习

德鲁·彭尼[a]，宾·李[b]，利忠·陈[a,*]，雅罗斯拉夫·J·西迪尔[b]，安娜·德雷韦克-奥索维茨卡[b]，拉梅什·伊利卡尔[b]，查理·泰[b]，拉维·艾耶尔[b]，安德鲁·赫德里奇[b]

[a]俄勒冈州立大学电气工程与计算机科学学院[b]英特尔公司

arXiv:2304.04797v2 [cs.LG] 4 Sep 2023

摘要

多个工作负载之间的资源共享已成为云服务提供商广泛采用的实践，其动机在于提高资源利用率并降低所有权成本。然而，由于资源争用可能对具有严格服务质量（QoS）要求的高优先级、面向用户的工作负载产生不利影响，有效的资源共享仍然是一个尚未解决的挑战。尽管近期方法已展现出有希望的结果，但由于工作负载事先未知且可能仅运行很短时间，这些工作在公有云环境中仍largely 不具备实践可行性，因为这会禁止离线学习，并严重阻碍在线学习。在本文中，我们提出了RAPID，这是一种用于高度动态运行环境中快速、完全在线的资源分配策略学习的新型框架。RAPID利用轻量级 QoS预测，通过受领域知识启发的样本效率提升和偏差降低技术实现，从而将控制与传统反馈源解耦，并以比以往工作快数个数量级的速率引导策略学习。在真实世界服务器平台上使用具有代表性的云工作负载进行评估表明，与先前最先进的方法需要数小时相比，RAPID 可在数分钟内学习稳定的资源分配策略，同时将服务质量提升9.0 倍并 将尽力而为工作负载性能提高 19-43%。

关键词：公有云、工作负载协同调度、动态资源分配、延迟关键型工作负载、性能预测、强化学习

1. 引言

在云计算范式的支持下，大规模数据中心按需部署工作负载，已发展成为满足不断增长的计算需求的高度可扩展解决方案。作为这一演进的一部分，云服务提供商持续探索提高服务器资源利用率的方法，从而降低总体拥有成本。特别是，工作负载协同调度已成为一种重要实践，即在同一台物理服务器上并发执行多个工作负载，并共享资源。然而，对这些共享资源的争用可能导致工作负载性能显著下降，并损害服务质量（QoS）。仅依靠集群级任务调度，已证明难以缓解这类细粒度资源争用行为；通常在不降低服务质量的情况下，CPU 平均利用率无法超过 50%–60% [1, 2]。随着云服务的普及，工作负载多样性进一步增加 [3]，很可能会加剧这些局限，凸显出对更细粒度资源管理的需求。

持续进行的研究致力于通过精细管理各项工作负载的资源分配，实现更高效的协同调度，从而减少资源争用。理想情况下，应为具有严格 QoS 目标的高优先级（HP）工作负载分配充足的资源，以满足当前需求；所有剩余资源则应分配给尽力而为（BE）工作负载（即没有严格性能要求的工作负载）。人们提出了多种方法来实现这种动态资源分配，并在各种场景中取得了良好成效，但由于工作负载信息有限且资源需求变化极大，这些方法在公有云环境中仍基本不具备实用性。具体而言，在公有云中，我们通常无法预先了解工作负载特征，因此资源分配策略必须在线自适应；而不支持在线策略自适应的控制器（例如基于启发式方法和控制理论的控制器）可能无法提供可接受的 QoS。云工作负载的运行时间也可能很短，通常不到 1–2 小时 [4]。与此同时，当前基于深度强化学习的最先进控制器进行初始策略学习可能需要数小时 [5–7], ，在此期间可能出现显著的服务质量下降。此外，现有的基于机器学习的方法虽然在技术上具有自适应能力，但往往适应得太慢，无法带来实际效益。要克服这些限制，就需要一种新方法，能够在不牺牲 QoS 保证的前提下快速适应新的运行条件。

本文提出资源感知预测即时分配决策（RAPID），这是一种新颖的框架，旨在实现

*通讯作者。电子邮箱地址：chenliz@oregonstate.edu

公有云环境中的快速、完全在线资源分配策略学习。RAPID 借助一种受领域知识启发的服务质量预测机制和高效的深度强化学习控制器，弥合了细粒度资源争用行为与粗粒度服务质量保证之间的差距。该服务质量预测器综合了多个时间尺度的信息；研究表明，即使在线样本有限，它也能为资源分配策略决策提供有效反馈，速度比以往工作快数个数量级。这种近乎即时的反馈进而使 RAPID 能够将资源分配策略决策与直接的服务质量测量完全解耦，并在几分钟内而非像以往工作那样耗费数小时，实现稳定的策略。在采用具有代表性的云工作负载的真实服务器平台上进行评估，结果证实 RAPID 在整体资源利用率和服务质量方面均具有显著的实际优势。具体而言，与基于控制理论和强化学习的先前最先进方法相比，RAPID 使尽力而为（BE）工作负载的平均性能提升了 19% 至 43%，同时在高度动态的运行环境中提供了强得多的服务质量保障。

2. 相关工作

资源分配优化已成为众多领域中备受关注的研究方向，涵盖从无线通信网络到智能电网等领域。特别是，随着云服务提供商需求的增长，云计算应用中的资源分配研究也得到了快速发展。

集群级调度器：一类研究专注于集群级调度器，这些调度器将工作负载分配给计算节点 [8–31]。这些调度器设计为与工作负载无关，因此通常使用节点级资源利用率和每个工作负载的资源利用率的粗粒度估计，而不考虑需求中更细粒度的波动。因此，协同调度决策必须相当保守，这使得在各种行业环境中，每个节点的平均资源利用率被限制在约 50-60% [1, 2]。因此，基于这些瞬时资源需求，进一步进行节点级资源分配优化存在很大的机会。

有限应用的节点级控制器：节点级资源管理的方法会根据硬件能力、假设和目标的不同而存在显著差异。尤其是，早期工作受到可用资源划分机制的限制，因此试图通过暂时暂停BE工作负载执行来缓解资源争用 [32–35]。各种工作还探索了仅包含BE工作负载的简化协同调度场景，因此优化总体工作负载性能、相对工作负载性能（即公平性）或二者的某种组合 [36–42]。当以严格的服务质量目标协同调度高优先级工作负载时，这些方法无法使用，而我们的工作正是针对这一问题。

通用节点级控制器：用于同时调度 HP 和 BE 工作负载的更通用方法可分为两类：基于传统方法的控制器和基于机器学习的控制器。在传统方法中，大多数基于搜索、启发式方法和控制理论。[43–52]. 这些方法提供可靠的控制，适用于静态运行环境，但在更动态的场景中表现出明显的局限性。具体而言，不断变化的资源需求可能要求基于搜索的方法进行频繁且可能存在危险的探索，这可能导致服务质量下降数个数量级 [7]。同样，基于启发式方法和控制理论的方法所采用的僵化服务质量阈值，可能导致次优的、振荡式的控制行为 [53]，同时仍需要较长的测量间隔（第 3.2 节），从而限制了其缓解细粒度（即亚秒级）资源争用的能力。通过使用服务质量代理指标（例如，已执行的指令 [46, 52] 或缓存未命中率 [43, 47]），可以部分避免这些问题，但当工作负载需求发生较大变化时，这些代理指标的阈值可能无法可靠地保持（第6节）。在这些工作中，LIBRA [52]代表了快速且一致的控制方法的最新水平，但不支持在线适应。

近期研究主要聚焦于更加通用的基于机器学习的方法，这类方法有望通过数据驱动的洞察以及持续适应新的运行环境，实现稳健的控制行为 [5–7，42，53–59]。然而，相对较高的数据需求可能导致训练时间很长，通常需要数小时 [5–7, 58]。在其他应用中，这样的训练时间或许可以接受，因为这些应用的工作负载是预先已知的，或会持续运行数天/数周；但在公有云中则完全不切实际，因为工作负载通常无法预先获知，且平均运行时间不到 1–2 小时 [4]。因此，初始策略训练期间造成的服务质量下降会大幅抵消潜在收益。即使是采用更高效网络架构的最先进方法 Twig [5]和 Twig+ [7]也仍存在这些局限性，我们将在评估中对此进行展示。同期工作 OLPart [59], 提出了一种基于多臂老虎机的在线学习方法。研究表明，OLPart 在协同调度多个 HP 和 BE 工作负载时能够提升可扩展性，但当控制间隔缩短至 2 秒以下时，性能会下降，因为该方法根本上依赖于 QoS 测量值 [59], , 从而妨碍其适应新的工作负载行为。相比之下，我们提出的方法能够以快一个数量级的控制间隔有效分配资源；根据开销分析，在嵌入式系统实现中，控制间隔可达到毫秒级。因此，我们提出 RAPID，将其作为唯一能够在高度动态的公有云环境中同时实现快速、自适应资源分配的方法。

3. 预备知识

3.1. 问题表述

物理计算节点通常具有多种资源，$R = \{R_1, ..., R_n\}$，每种资源都有有限数量的离散单元可供分配，记为 $R^* = \{R_1^*, ..., R_n^*\}$。在实践中，这些资源之间存在依赖关系，因此严格地只分配其中一部分资源，$\hat{R} = \{R_1, ..., R_m\} (m \leq n)$，可以缓解其他资源上的争用。例如，限制工作负载访问片外资源的速率。

数据（即内存带宽）会降低其争用共享缓存的能力，而共享缓存将用作这些数据的临时存储。[1] 在协同调度的工作负载中，我们考虑两个优先级级别：1）具有严格性能（即服务质量）目标的 HP 工作负载；2）没有严格性能目标的 BE 工作负载。我们将当前协同调度的工作负载表示为 $W = \{W_1^{HP}, ..., W_x^{HP}, W_1^{BE}, ..., W_y^{BE}\}$，将 HP 工作负载服务质量目标表示为 $Q^{HP} = \{Q_1^{HP}, ..., Q_x^{HP}\}$。[2] 工作负载 W_w 的测量值性能也会随时间 t 变化，记为 $P_t^{meas}(W_w)$。类似地，我们将时间 t 时工作负载 W_w 的预测性能（见第 4 节）记为 $P^{pred}\,t\,(W_w)$。在每个时间步 t，每个工作负载都会获得一定份额的各项资源，记为 $\hat{R}_{r,w,t}$。如公式 1 所示，工作负载协同调度的目标是以一种能够满足所有 HP 工作负载性能目标的方式分配有限的系统资源，同时最大化 BE 工作负载的性能。

$$\begin{aligned} \text{maximize} \quad & \sum_{w=1}^{y} P_t^{meas}(W_w^{BE}) \\ \text{subject to} \quad & P_t^{meas}(W_w^{HP}) \geq Q_w^{HP} \quad \forall w = 1, x \;\; \& \;\; \forall t = 1, T \\ & R_r^* \geq \sum_{w \in W} \hat{R}_{r,w,t} \quad \forall r = 1, m \;\; \& \;\; \forall t = 1, T \end{aligned} \tag{1}$$

请注意，满足高优先级服务质量目标所需的资源会随着工作负载行为的变化而随时间发生巨大变化。因此，不存在单一的最优解决方案。实现这些目标需要采用动态方法，根据当前需求，安全地最小化每个高优先级工-作负载的资源分配，从而最大化可供BE 工作负载使用的资源。我们提出的框架通过为每个高优先级工-作负载设置一个独立控制器来分解这项任务。因此，本文的讨论重点是单个高优先级工作负载的控制器级别设计，尽管整体方法仍适用于包含多个高优先级工作负载的环境（第 6.6节）。

3.2. 挑战 *&*洞察

由于瞬时资源需求和瞬时系统响应特性，将训练时间缩短所需的 1-2个数量级是一项具有挑战性的任务。这些因素给策略反馈带来了很大的不确定性，可能误导资源分配决策并降低服务质量。

3.2.1. 服务质量中的不确定性

资源需求的变化可能由多种原因造成，从工作负载需求的粗粒度变化（例如，每秒请求数）到工作负载执行过程中的细粒度阶段。这些变化可能相当显著；正如 Microsoft Azure 2017 数据集所显示的，平均 CPU 利用率高于 50% 的工作负载不足 5%，但近 40% 的工作负载会出现 CPU 利用率超过 80% 的峰值 [4]。与其他工作负载共同调度时，这些资源需求峰值可能会严重降低服务质量，尤其是在服务提供商采用更激进的节点超额订阅策略时，因为峰值重叠可能会使物理资源达到饱和 [2]。

我们进行了一系列实验来验证这些行为，如图1所示。每项实验（每一行）都将一个具有代表性的 HP 云工作负载与一个计算密集型尽力而为工作负载共同调度。我们使用外部流量生成器，使 HP 工作负载需求保持恒定。我们测量了每个请求的延迟，并将高于 99% 测量值的延迟（即第 99 百分位数或 p99）报告为服务质量。实验开始时，资源争用程度适中（时间步 0-100）；随后提高尽力而为资源限流，从而降低资源争用（时间步 100-200）。首先值得注意的是，以一秒为间隔获取的服务质量测量值（左栏）可能与长期平均值相差 100% 以上，尤其是在资源争用严重时。因此，单次测量可能无法恰当地反映工作负载的长期性能，进而误导控制策略学习。我们还观察到，资源限流前后的服务质量测量值分布可能依然相似，因此难以立即判断资源分配的变化是否确有益处。综上，可能需要在更长时间段内（例如 15 秒 [45]）对服务质量测量值取平均，以获得方差较低的服务质量估计；这也使得现有的大多数方法在高度动态的在线学习场景中难以实际应用。

资源分配决策也可以使用其他信息源（例如硬件性能计数器）作为服务质量的代理来制定，但现有研究中的这类方法效果有限。具体而言，非机器学习控制器通常只使用每周期指令数（IPC）作为唯一信息源。虽然 IPC（中间列）的测量值分布呈现出比 p99 延迟（左列）更多明显的阶梯，但其提供的信息往往不如其他鲜少使用的计数器（右列中的 3 个示例）丰富，而这些计数器的测量值分布呈现出最清晰的阶梯。在所有实验中，没有发现任何单个计数器能够成为最佳服务质量代理。事实上，为 Nginx 观察到的信息量最丰富的计数器，却是为 Resnet 观察到的最差计数器之一，其信息量仅略高于直接服务质量测量值。这些行为可以直观地归因于各工作负载对特定资源争用行为的敏感程度各不相同。综合这些观察结果，在测量周期相对较短的情况下，与仅使用单个计数器或直接服务质量测量值相比，结合多个计数器可以更准确地估计长期服务质量目标能否达成。尽管如此，要在数据受限的公有云环境中在线学习这些关系（而非在以往服务质量预测方法所采用的数据丰富的离线环境中）.....

[1]与大多数以往工作一样，我们假设CPU核心不会在工作负载之间共享，因为现有机制无法阻止私有的、按核心划分的资源中出现线程间争用。[2]计算节点上共同调度的工作负载集合通常会随着新作业启动和旧作业完成而随时间变化。为简化讨论，我们聚焦于工作负载固定的一个时间窗口，尽管整体方法仍适用于动态工作负载集合。

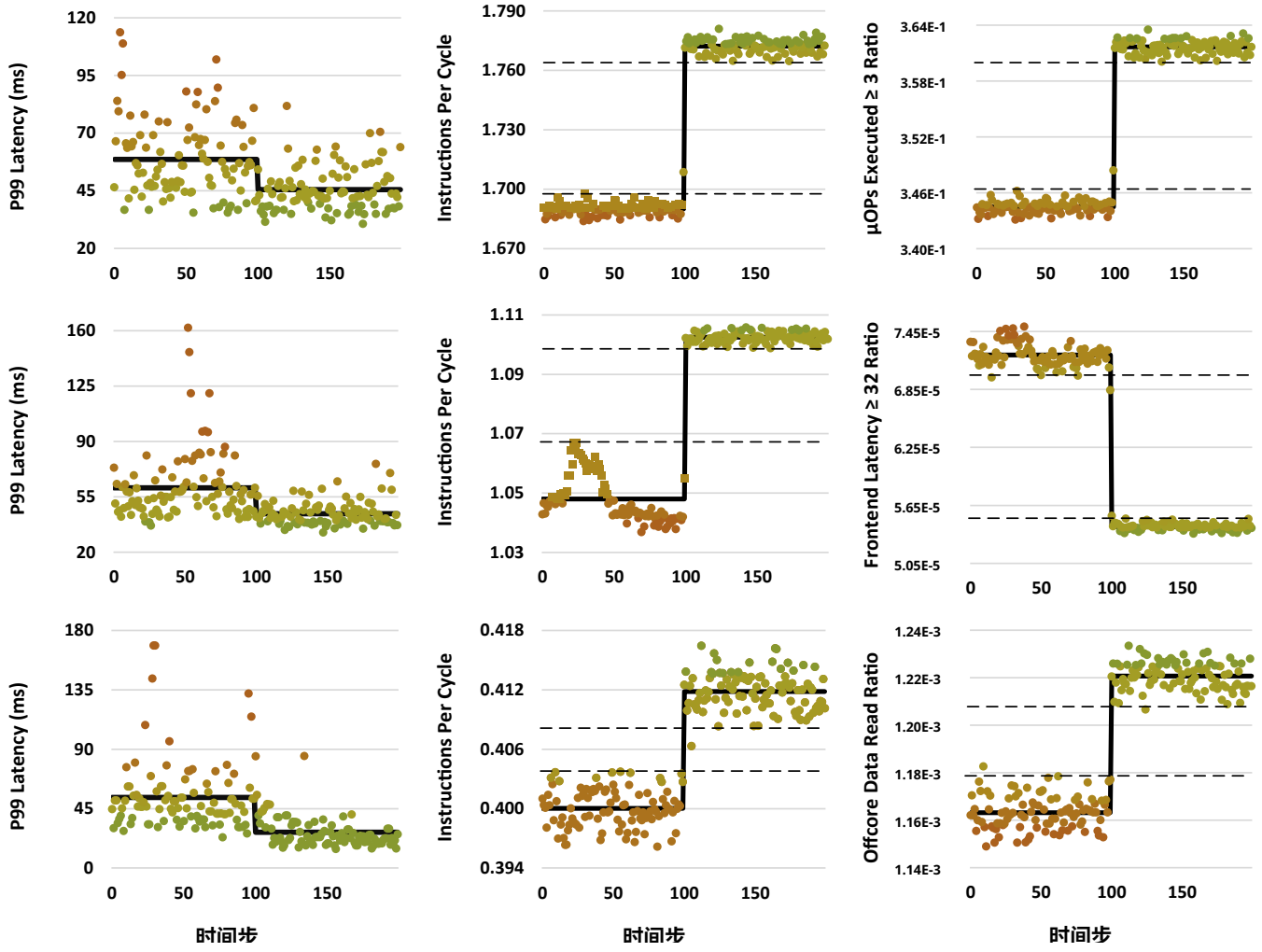


图1：协同调度 HP 和 BE 工作负载时各种测量源的一致性。顶部：HP 图像分类工作负载（ResNet）。中部：HP 推荐工作负载（DLRM）。底部：HP Web服务器工作负载（Nginx）。圆点表示每隔一秒进行的一次独立读数，实心黑线表示在应用 BE 尽力而为资源限流前后的长期平均值（时间步 100）。

方法 [7, 31])可能引入必须加以解决的重大误差，以确保可靠运行。

3.2.2. 系统响应的不确定性

即使拥有理想的信息源，学习有效的资源分配策略仍然是一项挑战，因为系统响应存在惯性 。这种惯性源于对有状态架构组件（如末级缓存）的固有依赖，并表现为工作负载性能的状态相关退化 [60]。因此，即便理论上合适的资源分配能够立即生效，也无法保证短期内达到可接受的服务质量，因为系统本身无法立即响应。等待观察这些影响显然会带来问题，因为后续纠正措施中的任何惯性都可能进一步延迟响应。这一现象表明，细粒度控制（即亚秒级至毫秒级控制）具有显著优势，它能够逐步纠正次优的资源分配决策，从而在性能退化发生时加以克服。然而，学习最优的纠正措施，需要以类似的频率进行策略学习。

细粒度的时间间隔，而由于上述信息来源的限制，这通常对于基于机器学习的控制器来说仍然不可行。这些洞见是我们工作的指导原则：我们利用快速的服务质量预测，从而能够以比以往工作快几个数量级的速率进行策略学习。无论如何，应用这一洞见都需要进一步考虑高效的在线学习策略，我们将在第 4节详细介绍。

4. RAPID

本节将介绍 RAPID，这是我们提出的用于细粒度资源分配的机器学习框架。RAPID 是一种实用方法，旨在高度动态的公有云运行环境中实现快速、完全在线的策略学习。我们通过以下创新方法实现这一目标：高效整合多个信息源，以最少的样本需求实现高服务质量预测准确率；将策略学习与直接的服务质量测量解耦，从而缩短训练时间；并针对机器学习资源分配方法中的各种传统问题，采用多种

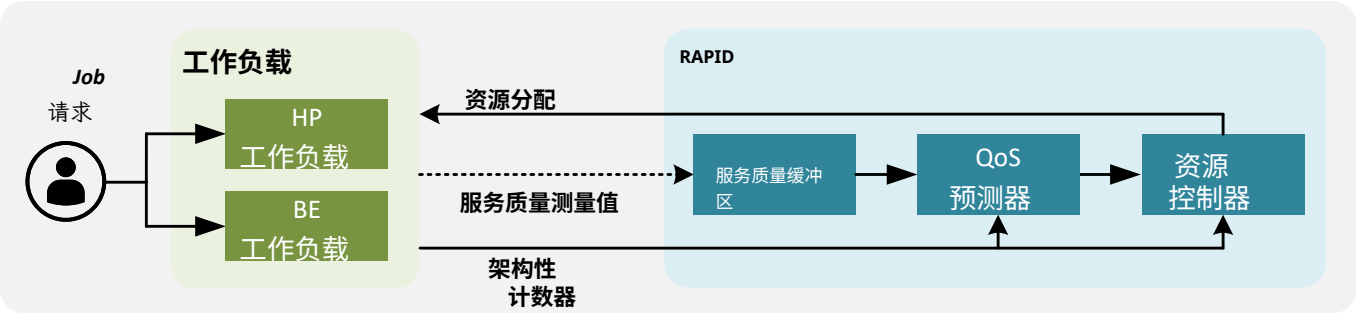


图 2：用于动态资源分配的 RAPID 框架。

受领域知识启发的解决方案。

4.1. 概述

我们提出的框架 RAPID 的高层设计如图 2所示。框架执行在接收到一个需要与其他任务共同调度到计算节点上的新HP 工作负载后开始。通常，我们假设此前并不了解该HP 工作负载，因此没有任何关于工作负载行为的先验信息。因此，RAPID 首先对一小部分资源分配选项进行采样，以收集初始训练数据，这些数据以架构计数器和 QoS 测量值对的形式给出。完成此次初始采样 / 训练后，RAPID使用 QoS预测而非 QoS测量值，从而为资源分配决策提供近乎即时的反馈。我们的资源控制器基于深度强化学习，利用这种近乎即时的反馈来安全地探索其他资源分配选项，并利用工作负载需求的瞬态波动。当可用时，通过累积长期测量平均值，QoS 预测器的准确度会持续提高；与此同时，主控制回路以精细得多的粒度执行，从而适应高度动态的运行环境。

4.2. 服务质量预测

RAPID 中快速的服务质量预测通过使用多个信息源、高效的资源子集和高影响力配置，以及一种新颖的预测器偏差降低方法，谨慎地平衡了数据需求与预测器准确性。

4.2.1. 模型

在 RAPID 中，QoS 预测被实现为一项回归任务，将短期架构计数器测量值映射到长期的逐工作负载服务质量。形式上，HP 工作负载 W_w^{HP}在时间 t 的服务质量，通过架构计数器测量值向量 $\mathbf{c}_{w,t}^{HP}$ 和函数逼近器 f进行预测，使得

$$P_t^{pred}(W_w^{HP}) = f(\mathbf{c}_{w,t}^{HP}). \tag{2}$$

近期研究表明，这种架构计数器与服务质量之间的映射虽然大致呈线性 [33, 52]，但使用非线性模型可以对其进行更准确的近似 [5, 7]。然而，由于训练样本有限，在我们的应用中，过拟合成为非线性模型一个值得特别关注的问题。我们采用一种基于径向基函数支持向量回归的相对低复杂度方法来平衡这些问题。该模型虽然是非线性的，但可以学习出相对平滑的决策边界，并且在测试中，

表 1：选定的架构事件（特征）。

	特征	**描述** [61]
计数器（*HP*）	inst_retired.any	统计已退役的指令数
	cpu_clk_unhalted.thread	统计核心未暂停时的周期数
	frontend_retired.latency_ge_32	统计未交付微操作的周期数达到 ≥ 32 个周期的时段
	uops_executed.core_cycles_ge_3	统计执行 ≥ 3 个微操作时的周期数
	保留站事件：空闲周期	统计保留站为空的周期数
	内存加载退休：二级缓存未命中	统计未命中二级缓存的已退休加载指令数
	周期活动：任意内存停顿	统计执行因等待加载而暂停的周期数
	片外请求。所有请求	统计片外内存事务
	非核心请求。需求数据_rd	统计需求读取的片外内存事务
计数器（*BE*）	inst_retired.any	统计已退役的指令数
	cpu_clk_unhalted.thread	统计核心未暂停时的周期数
	l2_rqsts.all_demand_miss	统计未命中二级缓存的需求请求数
	l2_rqsts.swpf_miss	统计未命中二级缓存的预取请求数

即使仅使用 20 个样本进行训练，其预测误差也始终低于线性模型。我们进一步通过认识到，其他工作负载引入的当前资源争用，可以通过每个单独工作负载的架构计数器间接观测到，从而最大限度地降低预测器的复杂性。换言之，即使唯一的输入是工作负载W_w^{HP}的计数器，也应当能够预测工作负载W_w^{HP}的服务质量。最后，RAPID 仅使用可用计数器中的一个小子集，即表1中列为“计数器（HP）”的计数器。这些计数器依据Penney 等人的方法，对若干具有代表性的云工作负载进行表征后选出 [7]。计数器在所需时间间隔内跨HP 工作负载的核心进行聚合，并进行对数变换以适应数值上的数量级差异，然后使用四分位距估计值进行粗略归一化。

4.2.2. 减少训练数据

RAPID 采用了多种技术来减少训练数据需求和总体训练时间。首先，RAPID 将采样/分配限制在一组高效的资源子集中，这些资源对工作负载性能和功耗的影响最大。该子集包括内存带宽，这使 RAPID 能够限制片外请求速率，并几乎消除对共享资源（例如末级缓存、内存和网络接口）的争用；还包括功耗，RAPID 利用功耗为计算密集型工作负载提升核心频率。[3]其次，RAPID 采用相对粗粒度的资源分配步长（例如，以 400 MHz 为步长调整核心频率，而非 100 MHz）。即使采用这种方法，RAPID 仍可通过在多个步长间交替，模拟细粒度的资源分配行为。最后，由于样本数量有限时均匀随机采样容易产生固有偏差，RAPID 在收集 QoS 预测数据时会避免这种做法。相反，RAPID 利用通用领域知识，将采样重点放在最有可能影响资源争用和工作负载性能的资源分配上。

4.2.3. 降低预测误差

完成服务质量预测器训练后，框架运行所需的唯一必需 测量值是简单的架构计数器测量值。然而，初始预测可能会偏向初始采样期间观察到的工作负载行为。RAPID 采用两步方案来解决这一问题：首先，通过比较过去的服务质量测量值和预测值的分布来估算偏差；其次，将该估算值用于补偿新预测中的偏差。

过去的 QoS 测量值虽然无法可靠地估计当前 QoS，但仍能为实际出现过的工作负载性能水平范围提供有用的信息。类似地，过去的 QoS 预测反映了预期会出现的工作负载性能水平范围。因此，这两个分布之间的差异可用作预测偏差的指标

[3]此子集的选择纯粹是为了提高常见运行环境下的学习效率；只需稍作修改控制器动作规范（第4.3.3节），即可分配其他资源。

。RAPID 通过维护一个缓冲区来保存此前 b 秒内的 QoS 测量值和 QoS 预测，从而实现这种比较。可以根据较低百分位的 QoS 测量值，推导出由可接受的资源分配所带来的工作负载性能基准估计。我们使用第 10 百分位数，记为 P meas , [-] 10 t b , t (W_w^{HP})。由于平均 HP 工作负载需求在 b 秒内应相对稳定（前提是 b 较小，例如 5 秒），因此即使测量存在延迟，该估计仍然稳健。相反，可以根据较高百分位的值，推导出由较差的资源分配所导致的工作负载性能下降估计。对于预测，我们使用第 90 百分位数，记为 P pred , [-] 90 t b , t (W_w^{HP})；对于测量值，则使用 P meas , [-] 90 t b , t (W_w^{HP})。最后，可以按照公式 3 所示，对时间步 t 的新 QoS 预测 P^{pred} t (W_w^{HP}) 进行偏差校正。

$$\hat{P}_t^{pred}(W_w^{HP}) = P_{[t-b,t]}^{meas,10}(W_w^{HP}) + \frac{P_t^{pred}(W_w^{HP}) - P_{[t-b,t]}^{meas,10}(W_w^{HP})}{1+\alpha\left(\frac{P_{[t-b,t]}^{pred,90}(W_w^{HP})}{P_{[t-b,t]}^{meas,90}(W_w^{HP})} - 1\right)} \tag{3}$$

我们引入一个增益因子 α，因为在稳定系统中，最坏情况下的服务质量测量值预计会低于最坏情况下的服务质量预测值，这是因为在预测到性能下降后，可以在其实际发生之前进行纠正。根据实证结果，在控制间隔为 0.2 秒时，$0.7 \le \alpha \le 0.9$ 能够产生理想的系统行为。

4.3. 资源分配控制器

RAPID 通过轻量级模型架构和动作表示高效复用信息，有效利用 QoS 预测作为策略反馈，并借助受领域知识启发的指导，实现快速的全在线策略学习

惯性感知的决策和争用感知的探索。

4.3.1. 模型

RAPID 中的资源控制器基于 Takavoli 等人的动作分支架构及其 Dueling Double DQN 变体实现，该变体称为分支式 Dueling Q 网络（BDQ） [62]。我们的控制器架构如图3所示，为每个动作维度使用独立分支来近似

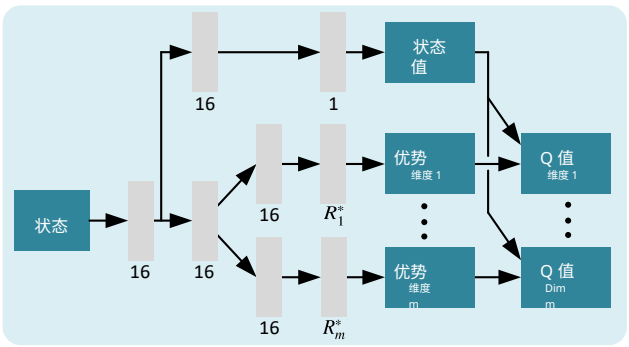


图3：动态资源分配控制器架构。灰色方框表示全连接层，每个方框下方列出了输出维度。

每个动作维度的优势函数，并结合统一的状态值预测器，以近似各动作维度的 Q 值。随后，根据每个维度中 Q 值最高的配置来选择动作。拆分这些动作维度后，网络输出数量随动作空间规模线性增长，而非组合式增长，从而简化了分配多种资源时的控制复杂度。我们按照 Takavoli 等人的建议计算时序差分 (TD) 目标和损失，其中，TD 目标取所有动作维度上的均值，损失则取所有动作维度上的均方误差。所有隐藏层均包含 16 个节点；我们发现，在训练数据有限的情况下，这能在模型容量与整体稳定性之间取得良好平衡。

4.3.2. 状态规范

RAPID 控制器的状态（公式 4）由三个要素组成。首先，RAPID 重用服务质量预测中的 HP 工作负载架构计数器 $\mathbf{c}_{w}^{HP}{}_{,t}$ 作为主要信息源，以学习资源分配对工作负载执行的影响。其次，RAPID 使用若干 BE 工作负载计数器（表1中列为“计数器（BE）”），计算这些 BE 工作负载在执行期间访问共享资源的归一化速率 λ_t^{BE}。该归一化速率由公式 5 给出（“cpu clk unhalted.thread”缩写为C，“inst retired.any”缩写为 I），可近似估算 BE 工作负载可能造成的最大潜在争用，且该估算基本不受资源影响。最后，RAPID 纳入当前服务质量（无论是预测值还是测量值），为控制器提供直接的运行余量指标。所有状态信息均按照前文对服务质量预测器的说明进行对数变换和归一化。

$$\mathbf{S} = \{\, \mathbf{c}_{w,t}^{HP},\ \lambda_t^{BE},\ \hat{P}_t(W_w^{HP}) \,\} \tag{4}$$

$$\lambda_t^{BE} = C * \frac{(l2_rqsts.all_demand_miss + l2_rqsts.swpf_miss)}{I} \tag{5}$$

4.3.3. 动作规范

公有云的动态特性要求控制器规范具有完全的灵活性。因此，我们不能像许多以往工作（例如 Twig [5]）那样，假设工作负载集合是固定的。相反，我们仅为 HP 工作负载指定动作，然后利用这些动作间接选择 BE 工作负载的资源分配。每种资源的规范如下：

- HP 工作负载的内存带宽（MBW）动作（a_{MBW}）基于限制的相对程度，可能取值范围为 0–90%。随后，这些动作将用于选择 BE 内存带宽限制值，指定为 100% − a_{MBW}。虽然我们的测试平台支持以 10% 为步长进行调整，但为了降低动作空间的维度，我们采用了前文所述的六种配置：0%、40%、60%、70%、80% 和 90%。
- 核心频率（CF）动作（a_{CF}）需要考虑系统功率限制；这些限制会根据核心数量设定频率上限。因此，限制 HP 核心频率对于最大化 BE 核心频率是必要的。为此，我们采用了更复杂的六步重叠方案，将a_{CF} 转换为 $\{2400, 3000, 3300, 3300, 3300, 3300\}$ HP 核心频率（MHz）和 $\{3300, 3000, 2400, 2000, 1400, 800\}$ BE 核心频率。

整体动作空间可表示为公式 6所示。

$$a_t = \{\{a_{MBW} \in \{0, ..., 5\}\} \times \{a_{CF} \in \{0, ..., 5\}\}\} \tag{6}$$

4.3.4. 奖励规范

奖励的设计旨在鼓励做出既能避免服务质量违规、又能最大化 BE 工作负载资源分配的决策。具体而言，对于导致服务质量违规的动作，将给予负奖励（公式 7），无论违规发生在使用服务质量测量值进行初始采样期间，还是发生在使用服务质量预测进行常规运行期间（公式 8）。随后，我们将该惩罚值截断为 $\beta = 2$。5，以减少基于状态值/优势估计的噪声。其依据是这样一种直觉：所有严重违规（例如超过目标值的 2.5 倍）都需要采取相同的（最大程度的）限流，以便快速恢复到适当的服务质量。

$$r_{w,t}^{-} = -\min\left(P_t^{ratio}(W_w^{HP}),\, \beta\right) \tag{7}$$

$$P_t^{ratio}(W_w^{HP}) = \begin{cases} \dfrac{P_t^{meas}(W_w^{HP})}{Q_w^{HP}}, & \text{if initial sampling.} \\ \dfrac{P_t^{pred}(W_w^{HP})}{Q_w^{HP}}, & \text{if regular operation.} \end{cases} \tag{8}$$

在未检测到或预测到服务质量违例时，会给出正奖励（公式 9），其依据包括两个组成部分：BE 工作负载的资源分配和服务质量裕量。对于更激进的 BE 资源分配，会给予更高奖励；当具备 BE 工作负载的先验知识时，可以使用 γ来确定各项资源的优先级，例如，当一个新的 *HP*工作负载与一个已在计算节点上运行较长时间的BE 工作负载一同调度时。我们通过实证确定，$\gamma = 0$。33是一个合适的静态值，并将动态调优留待未来工作。最后一项用于鼓励在接近服务质量目标（50%以内）时采取适当的谨慎态度，并总体上倾向于选择具有足够服务质量裕量的资源分配决策，以应对系统响应中的惯性。充裕量。

$$r_{w,t}^{+} = \gamma R_{CF,t}^{BE} + (1-\gamma) R_{MBW,t}^{BE} - \max\left(P_t^{ratio}(W_w^{HP}) - 0.5, 0\right) \tag{9}$$

4.3.5. 探索

在线学习使探索成为一项可能存在危险的任务。随机选择的资源分配可能导致严重的服务质量违规，也会延迟后续探索，因为

处理积压请求时，必须严格限制 BE 工作负载。RAPID 通过根据当前贪心动作周围的离散概率分布来选择动作，从而解决这一问题。给定探索概率 ϵ 和贪心动作索引 n，贪心动作的偏移量表示为

$$o_t = \{-\frac{\epsilon}{2n}, \ldots, -\frac{\epsilon}{2n}, 1-\epsilon, \frac{\epsilon}{2}\} \quad (10)$$

该分布会严格限制最大偏差不超过 +1，以应对服务质量客户端的存在，因为单次资源分配步骤就可能导致服务质量下降数个数量级 [7]。采用这种严格限制还使 RAPID 能够在整个执行过程中保持较高的探索概率，从而支持终身探索。

5. 方法

5.1. 平台

实验在一台配备 Intel® Xeon® Gold 6336Y CPU 的双路服务器上进行。所有工作负载都固定在第一个插槽上，而代表用户请求的流量生成器则固定在第二个插槽上。为避免私有缓存中的干扰，工作负载被固定到第一个插槽上不同的核心子集，这与大多数先前研究的做法一致。启用了超线程，因为它能提高某些工作负载可持续达到的最大请求速率。使用 Intel® Resource Director Technology [63] 限制内存带宽，并使用 Intel® Speed Select Technology [64]管理核心频率。

5.2. 工作负载

我们根据其在云计算中的重要性和总体代表性，选取了一组多样化的工作负载用于实验。高优先级工作负载包括：

- 来自 MLPerf Inference 基准测试套件的图像分类 [65]。该工作负载使用 ResNet-50 模型 [66] 在 ImageNet 数据集上运行图像分类[67]。
- MLPerf 推理基准套件中的推荐工作负载。该工作负载使用基于合成 Criteo Terabyte 数据集 [68] 的深度学习推荐模型（DLRM）。
- Nginx Web 服务器 [69]。Nginx 是使用最广泛的 Web 服务器，也是互联网中要求最严苛的服务提供商（包括 Netflix 和 Facebook）的首选解决方案 [70]。用户请求通过 wrk2 流量生成器进行模拟 [71]。
- Redis [72]。Redis 是使用最广泛的键值存储引擎，可为多种数据类型提供高性能的内存缓存 [73]。用户请求使用 Locust 流量生成器进行模拟 [74]。

BE 工作负载根据 Limaye 和 Adegbija [76]的建议，从 SPEC CPU2017 基准测试套件 [75] 中选取。选定的工作负载包括 502.gcc、507.cac- tuBSSN、511.povray、519.lbm、520.omnetpp、531.deepsjeng、538.imagick、548.exchange2 和 549.fotonik3d。这些工作负载涉及多个领域，包括代码编译、视频压缩、流体动力学、大气建模等。

我们测试了两种运行环境：1）静态 HP 负载搭配动态 BE 负载；2）动态 HP 负载搭配动态 BE 负载。我们通过实测确定 HP 工作负载适用的每秒请求数（RPS）速率：无需进行 BE 限流时的速率（动态最小值）；BE 需求较高期间需要严格限制 BE 时的速率（动态最大值）；以及 BE 需求较高期间需要适度限制 BE 时的速率（静态）。在我们的测试平台上，这对应于以下静态/动态 RPS 速率：图像分类为 250/100-350，推荐为 250/100-350，Nginx 为 400K/200K-500K，Redis 为 65K/55K-110K。MLPerf 工作负载使用内置流量生成器。对于 Nginx，我们将 wrk2 流量生成器配置为模拟 2K 个唯一用户访问 HTML 网页。对于 Redis，我们将 Locust 配置为以 64 个为一批，读写 2K 个唯一键值对，值的大小为 4KB。动态 BE 作业到达/完成通过循环调整活动线程数来模拟，线程数每分钟左右从最少 3 个变到最多 12 个。

5.3. 对比方法

我们将提出的框架 RAPID 与强化学习（RL）[5, 7] 和控制理论（PID） [52]领域先前最先进的方法进行比较。我们在表2中总结了各方法的特点，并在下文提供进一步的详细说明。

强化学习： 这种基于强化学习的方法采用 BDQ架构，以实现多动作维度下的高效学习。该方法最初提出时，仅支持与 HP 工作负载共同调度，并且需要数小时进行训练，

表2：评估方法的特征摘要。

方法	特征类别				
	在线策略适应	时间为稳定策略	多资源分配	多目标优化	控制间隔
RL	✓	小时	✓	✓	Long
PID	✗	分钟	✗	✗	简短
RAPID	✓	分钟	✓	✓	短

因此无法用于公有云环境。于是，我们在强化学习中加入了 所有 RAPID 中采用的基于控制器的优化，包括模型架构、状态/动作/奖励规范，以及针对 Twig 提出的所有其他优化+ [7]。因此，强化学习与 RAPID 之间的行为差异完全可以归因于本工作提出的学习效率改进。强化学习的初始训练样本从均匀分布中抽取，并在第 6.5 节进行消融实验。强化学习的策略决策/更新基于每五秒采集一次的服务质量测量值；这一间隔经确定能够适当平衡资源分配延迟与反馈一致性（第 3.2 节）。

[{"id":0,"output":"**PID：**这种控制理论方法采用比例-积分-微分（PID）控制器，能够以较低的开销提供一致的行为。控制器输入为IPC，控制器输出为内存带宽操作 [52]。由于PID无法控制多个资源，我们设置了一个安全的静态分配，以最大化HP核心频率。内存带宽决策通过将输入IPC与在初始采样期确定的目标IPC值进行比较来制定。我们发现，在五秒测量间隔下采集八个样本（即八种内存带宽配置），能够在确定目标IPC时提供适当的粒度，因此初始采样总计需要40秒。这个静态IPC阈值体现了传统控制方法的一般局限性，因为只有在工作负载需求相对静态（即与初始采样期间观察到的需求相似）时，才能保证该阈值持续有效；在动态运行环境中，它可能变得次优（第6.3节）。初始采样之后，PID使用200毫秒的控制间隔；低于该间隔时，测试基准开销（例如用于更新资源分配和收集架构计数器的系统调用）将变得不可忽略。"}]

RAPID：我们提出的框架首先在领域知识的指导下，以五秒的间隔采样少量资源分配选项（例如20 个）（即总计100 秒）。这些样本被用作服务质量预测器的初始训练数据，同时也会作为经验保存下来以供回放。随后，由于RAPID使用服务质量预测来进一步指导策略学习，采样/动作间隔可以大幅缩短。为进行公平评估，RAPID 在初始采样之后使用与 PID 相同的控制间隔（200 毫秒）。

5.4. 训练配置

强化学习：我们使用 AdaBelief 优化器[77]，搭配 β=(0.9, 0.999) 和 ϵ = 1E-8。训练从第 25 步开始，目标网络每 40 步更新一次，批量大小为 64，并采用相对较高的学习率 1E-2，配合激进的 0.5 梯度裁剪，以加速训练 [78]。折扣因子（γ）设置为 0.8，因为较差的资源分配动作通常可以在随后的几个动作内得到纠正。使用 1E-3 的权重衰减和 0.1 的 dropout，以避免对任何特定的工作负载需求过拟合 [79]。

QoS 预测： 正则化参数C = 5 通过在范围 [0.1, 50.0]内进行网格搜索确定。惩罚参数 ϵ = 0.05 用于惩罚对数尺度下超过 5% 的 QoS 测量差异（ln(1.05)≈0.05）。停止容差设为 0.05。其他设置均使用默认值在 scikit-learn 的支持向量回归实现中[80]。

5.5. 指标

近期研究 [5, 7]在两项指标的背景下比较了 HP 工作负载的服务质量：1）服务质量保证，考察服务质量违规发生的频率；2）服务质量延迟程度，考察这些服务质量违规的平均严重程度。单独来看，这两项指标都无法提供足够的信息来确定对终端用户的实际影响；即使将两者一并报告，也可能不易解读。例如，一个控制器频繁发生轻微违规，而另一个控制器则很少发生但违规程度严重。因此，我们引入加权 QoS 违规百分比这一综合性单一指标。如公式11所示，该指标通过对所有测量间隔取平均值计算得出：每个间隔要么达到 QoS 目标（权重为 0%），要么未达到 QoS 目标（权重为测量值 QoS 超出目标 QoS 的百分比）。

$$Q_{w,weighted} = 100 * \frac{1}{T} \sum_{t \in T} \max\left(\frac{P_t^{meas}(W_w)}{Q_w^{HP}}, 1\right) - 1 \qquad (11)$$

6. 评估

6.1. QoS 预测

我们首先考察 QoS 预测模型准确度。由于据我们所知，所有使用 QoS 预测的相关工作都假设进行了大量离线分析和/或掌握了工作负载行为的先验知识，因此无法与以往工作进行直接比较。相反，我们将基于 SVM 的方法与具有代表性的提升树（BT）预测器 [7]和神经网络（NN）预测器 [31]进行比较。为此，我们聚焦于一个包含 20 个随机选取样本的场景，其中 15 个样本具有可接受的 QoS，5 个样本具有不可接受的 QoS，以模拟初始采样期结束后紧接着的预期行为。我们在 25 次独立试验中预测以毫秒为单位的 p99 延迟，并报告相关结果。我们的 SVM 预测器的平均绝对误差（MAE）为 33，显著低于 NN 的 45。BT 的 MAE 相近，为 32，但其假阴性数量约为我们 SVM 预测器的 1.4 倍。我们进一步评估了对 SVM 预测器应用偏差校正所带来的益处。值得注意的是，在所有控制测试中，应用偏差校正均能降低 QoS 预测误差；Redis 的平均降幅为 22.8%，总体平均降幅为 10.1%。

6.2. 行为比较

接下来，我们考察亚秒级资源分配和在线策略适应所带来的控制行为差异。如图 4 所示，强化学习的策略决策相对不稳定，无法持续将服务质量裕量降至最低（即测量值与目标服务质量之间的差值；裕量越大，表示资源浪费越多）。相比之下，RAPID 利用快速决策间隔收集了更多训练数据，并在

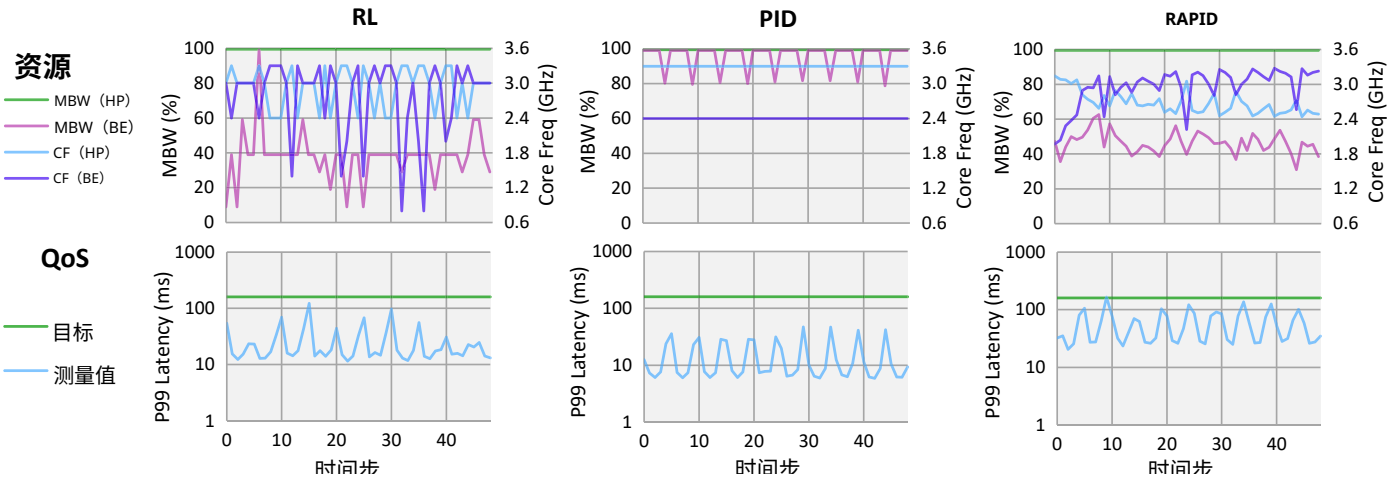


图 4：Redis（HP）与 519.lbm（BE）协同调度时的行为比较。所有图表均采用五秒平均值，以展示相同的动作间隔。RAPID 学会了稳定的资源分配策略，大幅减少了 QoS 裕量（即浪费的资源）。

约 50 秒内（图 4 中的时间步 10）收敛到稳定的资源分配策略。此外，结果表明，RAPID 学到的策略能够更稳定地降低服务质量裕量。总体而言，由于资源分配间隔不同，RAPID 能够尝试更激进的资源分配方案，因为工作负载性能出现任何下降时，都能比强化学习更快地予以纠正。PID 在大多数情况下都能提供高度一致的资源分配决策，但如图所示，它无法在所有场景中恰当地降低服务质量裕量。此外，在某些边缘场景中，服务质量可能显著下降，却不会对 IPC 造成可测量的影响。相比之下，RAPID 使用的额外架构计数器能够从不同角度反映资源争用，从而实现更稳健的行为。

6.3. 资源分配有效性

我们在各种协同调度场景下比较了 RAPID与强化学习和 PID 的优势，其中包括四种主要的 HP 工作负载和九种具有代表性的尽力而为工作负载。理想的资源分配策略应在尽量减少对服务质量影响的同时，最大化尽力而为工作负载性能。结果见表 3。首先，我们考虑尽力而为工作负载性能，以每秒指令数作为测量值。无论对于静态还是动态 HP 需求，强化学习的表现都最差，因为它需要学习适当的资源分配策略需要更多时间，并且由于其更长的决策间隔，在应对惯性时也必须更加保守。在静态高优先级需求测试中，PID相比强化学习提升了19%；但在动态 HP 需求测试中，相比强化学习仅提升了2%，因为其IPC阈值是根据执行开始时的工作负载需求确定的，随着运行条件变化，该阈值可能变得不再合适。相比之下，RAPID在所有场景中都提供了最高的尽力而为性能；由于RAPID能够快速且持续地适应变化的条件，其相较于强化学习的平均提升幅度在静态高优先级需求场景中约为43%，在动态 HP 需求场景中约为21%。

第二项考量是服务质量（QoS），我们以加权 QoS 违规百分比来报告，同时考虑违规的频率和严重程度（第 5.5 节）。理想情况下，这一数值应尽可能接近零，因为即便少量 QoS 违规也可能导致巨额罚款（可能高达数百万美元）以及更高的运营成本。即使在静态 HP 需求下，强化学习（RL）在训练初期也会出现大量 QoS 违规，尽管大多数违规仅略微超过 QoS 目标，导致服务质量下降幅度较小，仅为 0.03%。然而，在动态 HP 需求测试中，这些违规会变得更加严重；在先前未见过的状态下，不恰当的资源分配决策

表 3：服务质量与工作负载性能比较。结果显示所有 BE 工作负载的平均值。静态 HP 需求实验重复了 3 次，并根据 HP 工作负载计算置信区间。尽力而为工作负载性能以强化学习的性能为基准进行归一化。缩写：图像分类 = IC；推荐 =Rec。

标准	方法	静态 HP 需求					动态 HP 需求				
		IC	Rec	Nginx	Redis	Avg	IC	Rec	Nginx	Redis	Avg
尽力而为性能（越高=越好）	RL	1.00	1.00	1.00	1.00	1.00 ± 0.04	1.00	1.00	1.00	1.00	1.00
	PID	1.14	1.21	1.17	1.26	1.19 ± 0.01	1.00	1.11	0.93	1.04	1.02
	RAPID	**1.33**	**1.41**	**1.44**	**1.55**	**1.43 ± 0.01**	**1.15**	**1.14**	**1.24**	**1.33**	**1.21**
加权服务质量违规率（越低越好）	RL	0.097	0.036	0.000	0.000	0.033 ± 0.025	0.097	0.134	1.392	1.319	0.740
	PID	0.000	0.000	0.000	0.000	0.000 ± 0.000	0.099	0.662	0.029	0.316	0.277
	RAPID	**0.000**	**0.000**	**0.000**	**0.001**	**0.000 ± 0.000**	**0.000**	**0.000**	**0.101**	**0.022**	**0.031**

可能导致平均 QoS 出现大幅违规（0.74%）。有趣的是，PID 在某些场景中能维持接近理想的 QoS，但在其他场景中却会显著降低 QoS，这可能是因为对初始 IPC阈值的敏感度存在差异：不同工作负载因负载变化而产生的 IPC 变化幅度可能不同。最后，结果表明，RAPID 平均而言能够提供最强的服务质量保证，因为任何不恰当的策略决策都能被迅速检测并纠正。具体来说，在动态 HP 需求下，与强化学习（RL）和 PID 相比，RAPID 将服务质量下降分别降低了 95.8% 和 88.9%（提升 9.0 倍）。总体而言，RAPID 在显著提升尽力而为工作负载性能的同时，也实现了强大的服务质量保证。

6.4. 开销

RAPID 专为轻量化而设计，并支持毫秒级资源分配。具体而言，我们测得，在单个核心上进行推理的执行时间约为 0.2 ms（服务质量预测器）和 1.5 ms（资源控制器），对应的开销仅为 0.85%。尽力而为性能的提升轻松抵消了这部分开销；此外，只需根据当前预测的 QoS 裕量设置阈值，即可进一步降低开销（也就是说，仅在重新分配可能带来收益时才执行控制器）。服务质量预测器训练（5 ms）和资源控制器训练（10 ms）所需的开销通常可以忽略，因为它们不在资源分配决策的关键路径上，而且在初始学习期之后可以低频执行。因此，在嵌入式实现中，RAPID 完全可以在没有系统调用开销的情况下，以 5–10 ms 的间隔进行资源分配。

6.5. 消融实验

我们开展了多项消融测试，以验证所提组件的影响。首先，我们观察到，在初始采样期间使用领域知识指导来增强强化学习，而非使用均匀随机样本，对尽力而为性能没有造成统计显著的变化，并且仅适度提升了 HP 服务质量（加权服务质量违规仍比 RAPID 高 ≈10倍）。直观而言，这些初始样本会影响早期策略学习，但随着工作负载动态随时间变化，很快就会过时。其次，我们发现，从 RAPID 中移除除 IPC 以外的所有计数器，会使加权服务质量违规增加 5.9 倍，因为质量较低的指导会导致资源分配策略过于激进。最后，从 RAPID 中移除偏差校正会显著降低尽力而为性能（平均下降 5%，Redis 上下降 10%），这是因为出现了更多服务质量裕量预测偏低的情况，导致 RAPID 不必要地限制 BE 工作负载。

6.6. 讨论

6.6.1. 理论分析

由于我们所针对的运行环境中，工作负载行为和资源需求本身难以获知，因此精确建模工作负载性能并不现实。具体而言，我们无法保证某种资源分配绝不会导致服务质量违规。不过，我们可以，确立标准，在这些标准下，RAPID 能够保证较不严重的 QoS 违规。在此，我们假设强化学习和 RAPID 采用相同的资源分配策略。然后，我们定义 QoS 违规的严重程度，从 t_1 开始，到t_2 结束，如公式 12所示。

$$S = \int_{t_1}^{t_2} Perf(W^{HP}(t)) - Q^{HP}\, dt \tag{12}$$

RAPID 更短的控制间隔减少了对工作负载状态变化作出响应的延迟，从而能够更快地满足需求，进而降低$t_2 - t_1$。由于状态（包括服务质量）在使用服务质量预测时几乎会立即反映工作负载性能的下降，而使用服务质量测量值时则会有所滞后，因此 RAPID 作出的响应也会与资源需求更恰当地成比例。换言之，预计 RAPID 能同时缩短服务质量违规的持续时间并降低其程度。这些优势也会带来更高的尽力而为性能，因为 RAPID 能够更快、更恰当地响应 HP 资源需求较低的时段。

6.6.2. 多个工作负载与多个计算节点

相关工作表明，将基于强化学习的控制器扩展到跨多个计算节点的多个工作负载，是一项相对直接的任务 [6]。具体而言，研究发现，为每个高优先级工作负载分别设置一个控制器，其性能优于为所有高优先级工作负载设置单一控制器的方案。该方法可直接应用于我们的工作，因为每个 RAPID 控制器都试图最小化其高优先级工作负载所需的资源，从而共同最大化所有其他工作负载可用的资源。更高层级的协调仍将由集群调度器负责，该调度器通过缓解因节点资源不足而导致的高优先级工作负载之间的长期干扰问题，对 RAPID 起到补充作用。

6.6.3. 其他控制方法

基于贝叶斯优化的控制方法因其简单性和可靠性，近年来的相关研究中越来越受欢迎。然而，这些方法通常假设运行环境相对静态，资源分配配置能够在较长时间内保持最优。在更普遍的运行环境中，例如我们的工作所针对的环境，这些假设会带来问题。实际上，要适应图 4 所示的资源需求动态波动，就需要近乎连续地重新采样，或采用非常保守的资源分配。这些缺点在相关工作中得到了强调 [7] 因此，我们在测试中不考虑贝叶斯优化。

7. 结论

由细粒度资源分配支持的工作负载协同调度，已成为云服务提供商在不损害客户需求的情况下满足日益增长需求的一种有前景的方法。然而，基于机器学习的资源分配方法在很大程度上仍然

在公有云等高度动态的运行环境中并不切实际。我们提出的方法RAPID通过使用轻量级的服务质量预测模型，将策略决策与传-统反馈来源解耦，从而解决了这些局限。这些预测被证明能够为策略决策提供有效反馈，其速率比使用服务质量测量值时可实现的速率快数个数量级，进而实现在高度动态的运行环境中进行完全在线的策略学习。评估证实，与先前工作需要数小时相比，RAPID 仅需几分钟即可学习稳定的资源分配策略。因此，RAPID 显著提升了HP服务质量和尽力而为工作负载性能，通过更有效的资源利用率，带来更高的客户满意度并降低总体成本。

致谢

我们感谢 Kamil Andrzejewski 在 PID 控制器实现方面提供的帮助，以及RipanDas在工作负载设置方面提供的帮助。这项工作部分得到了英特尔公't司学术研究经费的支持。

参考文献

[1] C. Lu, K. Ye, G. Xu, C.-Z. Xu, T. Bai，云中的不平衡：对阿里巴巴集群跟踪数据的分析，载于：2017年 IEEE 大数据国际会议（Big Data），2017年，第2884–2892页。 [2] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, J. Wilkes，Borg：下一代，载于：EuroSys'20，2020年，1 14页。–。 [3] Flexera，云状态报告，技术报告，Flexera（2022年）。 [4] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, R. Bianchini，Resource Central：了解并预测工作负载，以改进大型云平台中的资源管理，载于：第26届操作系统原理研讨会论文集，2017年，第153–167页。[5] R. Nishtala, V. Petrucci, P. Carpenter, M. Sj¨ala nder，Twig：面向共置延迟关键型云服务的多智能体任务管理，载于：高性能计算机体系结构国际研讨会（HPCA），2020年，第167–179页。[6] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, R. K. Iyer，Firm：面向服务等级目标的微服务智能细粒度资源管理框架，载于：第14届USENIX 操作系统设计与实现会议论文集，2020年，第805–825页。 [7] D. Penney, B. Li, J. J. Sydir, L. Chen, C. Tai, S. Lee, E. Walsh, T. Long，Prompt：为网络应用学习动态资源分配策略，《未来一代计算机系统》145（2023年）164–175。 [8] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, A. Goldberg，Quincy：分布式计算集群的公平调度，载于：ACM SIGOPS 第22届操作系统原理研讨会论文集，2009年，第261–276页。 [9] M. Zaharia, D. Borthakur, J. Sen Sar ma, K. Elmeleegy, S. Shenker, I. Sto- ica，延迟调度：实现集群调度局部性与公平性的简单技术，载于：第5届欧洲计算机系统会议论文集，2010年，第265–278页。 [10] R. Nathuji, A. Kansal, A. Ghaffarkhah，Q-clouds：管理面向服务质量的云中的性能干扰效应，载于：第5届欧洲计算机系统会议论文集，2010年，第237–250页。 [11] J. Mars, L. Tang, R. Hundt, K. Skadron, M. L. Soffa，Bubble-up：通过合理共置提高现代仓库规模计算机的利用率，载于：2011年 IEEE/ACM 第44届微体系结构国际研讨会（MICRO），2011年，第248–259页。 [12] C. Delimitrou, C. Kozyrakis，Paragon：异构数据中心中的服务质量感知调度，《ACM计算机系统汇刊》31（2013年）。

[13] M. Schwarzkopf、A. Konwinski、M. Abd-El-Malek、J. Wilkes，《Omega：面向大型计算集群的灵活可扩展调度器》，载于：第八届 ACM 欧洲计算机系统会议论文集，2013年，第351–364页。[14] J. Mars、L. Tang，《Whare-map：“同构”仓库规模计算机中的异构性》，载于：第40届计算机体系结构国际研讨会论文集，2013年，第619–630页。[15] C. Delimitrou、C. Kozyrakis，《Quasar：资源高效且服务质量感知的集群管理》，载于：第19届编程语言与操作系统架构支持国际会议论文集，2014年，第127–144页。[16] D. Lo、L. Cheng、R. Govindaraju、L. A. Barroso、C. Kozyrakis，《迈向大型延迟关键型工作负载的能耗比例化》，载于：2014年 ACM/IEEE 第41届计算机体系结构国际研讨会（ISCA），2014年，第301–312页。[17] Y. Zhang、M. A. Laurenzano、J. Mars、L. Tang，《Smite：在真实系统 SMT 处理器上精准预测服务质量，以提高仓库规模计算机的利用率》，载于：2014年第47届 IEEE/ACM 微体系结构国际研讨会，2014年，第406–418页。[18] S. Blagodurov、A. Fedorova、E. Vinnik、T. Dwyer、F. Hermenier，《集群中的多目标作业放置》，载于：SC '15：高性能计算、网络、存储与分析国际会议论文集，2015年，第1–12页。[19] C. Delimitrou、D. Sanchez、C. Kozyrakis，《Tarcil：协调大型共享集群中的调度速度与质量》，载于：第六届 ACM 云计算研讨会论文集，2015年，第97–110页。[20] A. Verma、L. Pedrosa、M. R. Korupolu、D. Oppenheimer、E. Tune、J. Wilkes，《Google 使用 Borg 进行大规模集群管理》，载于：欧洲计算机系统会议（EuroSys）论文集，2015年，第1–17页。[21] S. Singh、I. Chana，《Q-aware：基于服务质量的云资源配置》，《计算机与电气工程》47（2015）：138–160。[22] C. Delimitrou、C. Kozyrakis，《Hcloud：共享云系统中的资源高效配置》，载于：第二十一届编程语言与操作系统架构支持国际会议论文集，2016年，第473–488页。[23] H. Mao、M. Alizadeh、I. Menache、S. Kandula，《利用深度强化学习进行资源管理》，载于：第15届 ACM 网络热点研讨会论文集，2016年，第50–56页。[24] V. S. Marco、B. Taylor、B. Porter、Z. Wang，《通过内存感知的任务共置提高 S park 应用吞吐量：一种混合专家方法》，载于：第18届 ACM/IFIP/USENIX 中间件会议论文集，2017年，第95–108页。[25] N. Liu、Z. Li、J. Xu、Z. Xu、S. Lin、Q. Qiu、J. Tang、Y. Wang，《利用深度强化学习进行云资源分配与电源管理的分层框架》，载于：2017年 IEEE 第37届分布式计算系统国际会议（ICDCS），2017年，第372–382页。[26] L. Wei、C. H. Foh、B. He、J. Cai，《面向 IaaS 云中异构工作负载的高效资源分配》，《IEEE 云计算汇刊》6（1）（2018）：264–275。[27] K. Rzadca、P. Findeisen、J. Swiderski、P. Zych、P. Broniek、J. Kusmierek、P. Nowak、B. Strack、P. Witusowski、S. Hand 等，《Autopilot：Google 的工作负载自动扩缩》，载于：第十五届欧洲计算机系统会议论文集，2020年，第1–16页。[28] A. Khalid、Q. ul Ain、A. Qasim、Z. Aziz，《面向雾计算物联网的基于服务质量的最优资源分配与工作负载均衡》，《开放计算机科学》11（1）（2021）。[29] J. Chen、Y. Wang、T. Liu，《基于资源请求自适应预测的云计算主动资源分配方法》，《EURASIP 无线通信与网络期刊》2021（1）（2021年12月）。[30] S. Tuli、G. Casale、N. R. Jennings，《GOSH：在雾计算环境中使用深度代理模型进行任务调度》，《IEEE 并行与分布式系统汇刊》（2022）。[31] S. Tuli、S. R. Poojara、S. N. Srirama、G. Casale、N. R. Jennings，《COSCO：利用协同仿真和基于梯度的优化实现雾计算环境中的容器编排》，《IEEE 并行与分布式系统汇刊》33（1）（2022）：101–116。[32] H. Yang、A. Breslow、J. Mars、L. Tang，《Bubble-flux：精准的在线服务质量管理，提高仓库规模计算机的利用率》，《ACM SIGARCH 计算机体系结构新闻》41（3）（2013）：607–618。

[33] X. Zhang、E. Tune、R. Hagmann、R. Jnagal、V. Gokhale、J. Wilkes，《Cpi2：共享计算集群中的 CPU 性能隔离》，载于：第八届 ACM 欧洲计算机系统会议论文集，2013年，第379–391页。[34] S. Blagodurov、D. Gmach、M. Arlitt、Y. Chen、C. Hyser、A. Fedorova，《通过基于权重的共置管理，在满足关键服务级别协议的同时最大化服务器利用率》，载于：2013年 IFIP/IEEE 国际集成网络管理研讨会（IM 2013），2013年，第277–285页。[35] H. Kasture、D. B. Bartolini、N. Beckmann、D. Sanchez，《Rubik：面向延迟关键型系统的快速分析式电源管理》，载于：2015年第48届 IEEE/ACM 微体系结构国际研讨会（MICRO），2015年，第598–610页。[36] J. Rao、X. Bu、C.-Z. Xu、L. Wang、G. Yin，《Vconf：一种面向虚拟机自动配置的强化学习方法》，载于：第六届国际自治计算会议论文集，2009年，第137–146页。[37] C.-Z. Xu、J. Rao、X. Bu，《Url：一种统一的强化学习自治云管理方法》，《并行与分布式计算杂志》72（2）（2012）。[38] L. Funaro、O. A. Ben-Yehuda、A. Schuster，《Ginseng：市场驱动的末级缓存分配》，载于：2016年 USENIX 年度技术会议论文集，2016年，第295–308页。[39] J. Park、S. Park、W. Baek，《CoPart：在商用服务器上协同划分末级缓存和内存带宽，以实现公平性感知的工作负载整合》，载于：第十四届 EuroSys 会议论文集，2019年，第1–16页。[40] Q. Chen、S. Xue、S. Zhao、S. Chen、Y. Wu、Y. Xu、Z. Song、T. Ma、Y. Yang、M. Guo，《Alita：通过偏置资源管理，为公有云提供全面的性能隔离》，载于：SC20：高性能计算、网络、存储与分析国际会议，2020年，第1–13页。[41] R. B. Roy、T. Patel、D. Tiwari，《Satori：牺牲短期收益以换取长期收益，实现高效且公平的资源划分》，载于：第48届计算机体系结构国际研讨会，2021年，第292–305页。[42] R. Chen、J. Wu、H. Shi、Y. Li、X. Liu、G. Wang，《Drlpart：一种面向商用服务器的高效稳健最优资源划分深度强化学习框架》，载于：第三十届高性能并行与分布式计算国际研讨会论文集，2021年，第175–188页。[43] H. Cook、M. Moreto、S. Bird、K. Dao、D. Patterson、K. Asanovic，《缓存划分的硬件评估：在保持响应能力的同时提高利用率和能效》，载于：计算机体系结构国际研讨会（ISCA），2013年，第308–319页。[44] V. Petrucci、M. A. Laurenzano、J. Doherty、Y. Zhang、D. Moss´e、J. Mars、L. Tang，《Octopus-man：面向仓库规模计算机中异构多核处理器的 QoS 驱动任务管理》，载于：2015年 IEEE 第21届高性能计算机体系结构国际研讨会（HPCA），2015年，第246–258页。[45] D. Lo、L. Cheng、R. Govindaraju、P. Ranganathan、C. Kozyrakis，《Heracles：大规模提升资源效率》，载于：计算机体系结构国际研讨会（ISCA），2015年，第450–462页。[46] H. Zhu、M. Erez，《Dirigent：在共享多核系统上为延迟关键型任务保障服务质量》，载于：编程语言与操作系统架构支持国际会议（ASPLOS），2016年，第33–47页。[47] C. Xu、K. Rajamani、A. Ferreira、W. Felter、J. Rubio、Y. Li，《Dcat：面向高效、性能敏感型基础设施即服务的动态缓存管理》，载于：第十三届 EuroSys 会议论文集，2018年，第1–13页。[48] S. Chen、C. Delimitrou、J. F. Mart´ınez，《PARTIES：面向多个交互式服务的 QoS 感知资源划分》，载于：编程语言与操作系统架构支持国际会议（ASPLOS），2019年，第107–120页。[49] X. Chen、J. Lin、Y. Ma、B. Lin、H. Wang、G. Huang，《基于渐进式 QoS 预测模型的云软件服务自适应资源分配》，《中国科学：信息科学》（2019年9月）。[50] T. Patel、D. Tiwari，《CLITE：为仓库规模计算机中的多个延迟关键型作业提供高效且 QoS 感知的共置》，载于：高性能计算机体系结构国际研讨会（HPCA），2020年，第193–206页。[51] Q. Li、B. Li、P. Mercati、R. Illikkal、C. Tai、M. Kishinevsky、C. Kozyrakis，Rambo：使用贝叶斯优化进行微服务资源分配，IEEE《计算机架构快报》20 (1) (2021) 46–49。[52] Y. Zhang, J. Chen, X. Jiang, Q. Liu, I. M. Steiner, A. J. Herdrich, K. Shu, R. Das, L. Cui, L. Jiang，《Libra：通过动态内存带宽管理清理云端》，载于：国际高性能计算机体系结构研讨会（HPCA），2021年，第 815–826 页。[53] R. Nishtala, P. Carpenter, V. Petrucci, X. Martorell，《Hipster：面向延迟关键型云工作负载的混合任务管理器》，载于：国际高性能计算机体系结构研讨会（HPCA），2017年，第 409–420 页。[54] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, C. Delimitrou，《Seer：利用大数据应对云微服务性能调试的复杂性》，载于：第二十四届编程语言与操作系统架构支持国际会议论文集，2019年，第 19–33 页。[55] X. Chen, F. Zhu, Z. Chen, G. Min, X. Zheng, C. Rong，《利用预测增强型反馈控制与强化学习进行云端软件服务资源分配》，IEEE《云计算汇刊》（2020年5月）。[56] B. Li, Y. Wang, R. Wang, C. Tai, R. Iyer, Z. Zhou, A. Herdrich, T. Zhang, A. Haj-Ali, I. Stoica, K. Asanovic，《RLDRM：利用深度强化学习为网络功能虚拟化进行闭环动态缓存分配》，载于：IEEE 网络软件化会议（NetSoft），2020年，第 335–343 页。[57] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, C. Delimitrou，《Sinan：基于机器学习且感知服务质量的云微服务资源管理》，载于：第二十六届 ACM 编程语言与操作系统架构支持国际会议论文集，2021年，第 167–181 页。[58] X. Chen, L. Yang, Z. Chen, G. Min, X. Zheng, C. Rong，《针对云端软件服务、结合工作负载时间窗口的资源分配：一种深度强化学习方法》，IEEE《云计算汇刊》11 (2) (2023) 1871–1885。[59] R. Chen, H. Shi, Y. Li, X. Liu, G. Wang，《OLPart：基于在线学习的资源划分，用于在商用计算机上共置多个延迟关键型作业》，载于：第十八届欧洲计算机系统会议论文集，2023年，第 347–364 页。[60] H. Kasture, D. Sanchez，《Ubik：为延迟关键型工作负载提供严格服务质量保障的高效缓存共享》，载于：编程语言与操作系统架构支持国际会议（ASPLOS），2014年，第 729–742 页。[61] Intel 公司，《Intel 64 和 IA-32 架构软件开发人员手册》，software.intel.com/content/www/us/en/develop/articles/intel-sdm.html（2022年）。[62] A. Tavakoli, F. Pardo, P. Kormushev，《深度强化学习的动作分支架构》，arXiv:1711.08946（2019年）。[63] Intel 公司，《Intel® 资源调度器技术》，intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html（2021年）。[64] Intel 公司，《Intel® Speed Select 技术》，intel.com/content/www/us/en/architecture-and-technology/speed-select-technology-article.html（2021年）。[65] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, . Y. Zhou，《MLPerf 推理基准测试》（2019 arXiv:1911.02549）。[66] K. He, X. Zhang, S. Ren, J. Sun，《用于图像识别的深度残差学习》，载于：2016年 IEEE 计算机视觉与模式识别会议（CVPR），2016年，第 770–778 页。[67] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei，《ImageNet：大型分层图像数据库》，载于：2009年 IEEE 计算机视觉与模式识别会议，2009年，第 248–255 页。[68] Criteo Labs，《太字节点击日志》，labs.criteo.com/2013/12/download-terabyte-click-logs/（2013年）。[69] F5 Networks，《Nginx》，nginx.com（2021年）。[70] Netcraft，《2021年12月 Web 服务器调查》，news.netcraft.com/archives/category/web-server-survey/（2021年）。[71] G. Tene，《wrk2：wrk 的恒定吞吐量、正确延迟记录变体》，github.com/giltene/wrk2（2019年）。[72] Redis，《Redis》，redis.io（2021年）。

[73] DB-Engines，《键值存储的 DB-Engines 排名》，db-engines.com/en/ranking/key-value+store（2021年）。[74] Locustio，Locust，locust.io/（2021年）。[75] 标准性能评估公司，SPEC 基准测试，spec.org/benchmarks.html（2017）。[76] A. Limaye，T.Adegbija，SPECCPU2017 基准测试套件的工作负载特征分析，载于：《2018年 IEEE 系统与软件性能分析国际研讨会》（ISPASS），IEEE，2018年，第149–158页。[77] J. Zhuang，T. Tang，Y. Ding，S. Tatikonda，N. Dvornek，X. Papademetris，J. Duncan，AdaBelief 优化器：通过对观测梯度的信念调整步长，神经信息处理系统会议（2020年）。[78] J.Zhang，T.He，S.Sra，A.Jadbabaie，梯度裁剪为何能加速训练：自适应性的理论依据，arXiv:1905.11881（2019年）。[79] J. Farebrother，M.C.Machado，M. Bowling，深度 Q网络中的泛化与正则化，arXiv:1810.00123（2018年）。[80] F. Pedregosa，G.Varoquaux，A.Gramfort，V. Michel，B. Thirion，O. Grisel，M. Blondel，P. Prettenhofer，R. Weiss，V. Dubourg，J. Van-derplas，A.Passos，D.Cournapeau，M. Brucher，M. Perrot，E.Duchesnay，Scikit-learn：Python 中的机器学习，《机器学习研究杂志》12（2011年）2825–2830。