计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220333
61(1): 43−65, 2024

面向多应用混部的性能保障方法综述

郭 静 胡存琛 包云岗
（中国科学院计算技术研究所先进计算机系统研究中心 北京 100190）
（中国科学院大学 北京 100190）
（guojing171@mails.ucas.ac.cn）

Survey on Guaranteeing the Performance of Co-Located Applications

Guo Jing, Hu Cunchen, and Bao Yungang
（*Research Center for Advanced Computer System*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190）
（*University of Chinese Academy of Sciences*, *Beijing* 100190）

Abstract The huge cost of investment and low resource utilization in the datacenter has long been a great concern to cloud providers. To address this issue, a straightforward way is co-locating more applications on the same hardware to improve resource efficiency. However, the shared resource contention caused by co-located applications leads to performance interference, affecting the application's performance, quality of service (QoS) and user satisfaction. Therefore, how to guarantee the performance of the co-located application has been a key issue in the colocation scenario. We introduce the researches of guaranteeing the performance of co-located applications, including the background of co-location, challenges, and key technologies. The related work is summarized from four aspects: application and cluster characterization (basic), interference detection (premise), server-level resource allocation (micro-level policy), and cluster-level job scheduling (macro-level policy). In addition, due to the diverse characteristics of co-located applications and clusters, the research of guaranteeing the performance faces different challenges and problem complexity in the different co-located scenarios. For example, the number of co-located applications deployed on a unit resource will directly affect the time cost of searching resource space, and the running mode of applications will affect the competition intensity of shared resources. Therefore, from the perspective of problem complexity, we discuss and analyze the challenges of research work from three dimensions, cluster and application characteristics, resource interference dimension, and the number of co-located applications. At the end of this paper, we discuss the future research directions and the challenges in the high deployment density scenario. We conclude that the software/hardware co-designed full-stack approach is the trend to guarantee the performance in high deployment density clusters, and this approach can help to provide predictable performance and high resource efficiency in the datacenter.

Key words co-location; performance guarantee; quality of service (QoS); resource sharing; resource isolation; interference detection; resource management; job scheduling

摘 要 数据中心的高投入和低资源利用率一直是云服务提供商关注的问题. 面对这个难题, 直接的解决方案是在同等资源上混合部署更多的应用以提高资源使用效率. 然而, 由于混部应用对共享资源的竞争导致了应用间的性能干扰, 从而影响了应用的性能、服务质量（quality of service, QoS）和用户满意度, 因

收稿日期: 2022-04-25; **修回日期**: 2023-02-09
基金项目: 广东重点研发计划项目(2020B010164003)
This work was supported by the Key Research and Development Program of Guangdong Province (2020B010164003).
通信作者: 包云岗(baoyg@ict.ac.cn)

此如何保障应用的性能已成为混部场景下的关键问题. 着重从应用和集群特征分析(基础)、干扰检测(前提)、单节点资源分配(微观层面策略)和集群作业调度(宏观层面策略)4个方面阐述多应用混部性能保障的相关背景、挑战和关键技术. 在不同的混部场景下, 由于应用和集群特征等不同, 性能保障工作所面临的挑战和问题复杂度也各异, 例如单位资源上混合部署的应用数量会直接影响到搜索资源空间的时间开销, 应用的运行方式会影响到共享资源的竞争强度. 因此, 从问题复杂度角度出发, 从应用和集群特征、资源干扰维度和混部应用个数3个维度对相关研究工作面临的挑战进行讨论和分析. 探讨了面向高密度混部场景应用性能保障方法的发展方向和挑战, 认为全栈式的软硬件协同方法是保障高密度混部下应用性能的趋势, 该方法有助于全面地提升应用性能的可靠性和数据中心的资源利用率.

关键词 混部; 性能保障; 服务质量; 资源共享; 资源隔离; 干扰检测; 资源管理; 作业调度

中图法分类号 TP316

随着移动互联网和交互式应用的大规模使用, 数据中心已然成为重要的信息基础设施, 它支持着我们的日常生活和工作, 例如即时通信、移动支付和视频会议等应用的稳定运行均离不开数据中心的支持. 由于交互式应用直接面向用户, 为保障用户体验, 应用需要能对请求进行快速响应[1-2]. 由于应用负载具有突发性特征, 为保证应用在流量峰值时也能够快速响应, 服务提供商不得不采用为其分配冗余资源的方法, 这就导致了大量资源空闲、使用效率低下[3-5]. 然而, 若不为应用预留冗余的资源, 当突发流量到来时, 陡增的负载使得应用的资源压力增大、应用性能下降, 进而影响到应用的服务质量、用户体验和用户满意度, 最终影响公司的收益. 因此数据中心面临着资源使用效率和应用服务质量的矛盾.

为兼顾资源使用效率和应用服务质量, 大量研究工作聚焦于保障交互式应用性能的同时提升资源使用效率. 由此, 数据中心的资源使用效率开始稳步提升. 2012年, 谷歌数据中心平均CPU和内存资源使用效率只有20%和40%[6]; 2013年, 谷歌的在线应用集群和离线应用集群的CPU利用率分别为30%和75%[2], 不过这种将在线应用和离线应用分开部署的方式, 会导致集群出现资源使用效率不平衡的问题. 为进一步提升数据中心的资源使用效率, 最直接的方法就是将在线应用和离线应用混合起来运行, 让离线应用充分利用在线应用闲置的资源. 通过将在线和离线应用混部运行, 2015年, 谷歌混部集群的平均CPU资源使用效率达到了40%左右[4]; 2018年, 阿里巴巴混部集群的平均CPU和内存资源使用效率分别达到了38.2%和88.2%[5]; 至2019年, 谷歌数据中心的CPU和内存资源利用率分别达到了50%和55%[7].

数据中心资源使用效率稳步提升的关键在于应用混部. 混部, 即将多个应用混合部署在同一物理资源上, 充分利用空闲资源以降低成本. 这种混合式的应用放置方法虽然提升了数据中心CPU和内存资源的使用效率, 但是混合部署会造成应用间相互干扰, 从而增加了应用性能的不可预测性[8-13]. 因此, 利用应用混部提升资源使用效率的关键在于: 如何在混部场景下保障应用的性能?

混部产生性能干扰的主要原因在于应用之间对共享资源的无序竞争. 共享资源是指从体系结构层次的CPU核、末级缓存(last level cache, LLC)、内存带宽、网络带宽等, 到系统软件层次的调度队列、同步锁、系统缓冲区等. 多个层次的共享资源竞争导致了应用的性能波动. 研究数据表明, 同一台服务器上同时运行2个不同类型应用时, 因资源无序竞争造成的性能突变可达3倍之多[14]. 因此, 在混部场景下保障应用性能的前提在于要能及时发现或预测应用在哪些资源维度上发生了资源竞争, 同时能量化或评估出资源竞争对应用性能的影响程度, 以此来指导采用何种性能保障策略.

这种因资源竞争导致的性能干扰不仅会影响到对延迟敏感(latency sensitive, LS)的交互式应用, 还会影响到对延迟不敏感的尽最大努力(best effort, BE)完成的离线应用. 对于LS应用, 性能干扰会带来3个问题: 1)影响用户体验. 对于交互式应用, 响应时间是量化应用性能的关键指标, 因性能干扰而造成的响应时间增加会影响用户体验. 2)影响公司经济效益. 谷歌、微软和亚马逊均做过相关的实验并发现应用的性能变化会直接影响到公司的收益. 当谷歌搜索结果的返回时间由0.4 s增至0.9 s后, 其广告收入下降20%[15], 微软旗下必应搜索引擎的响应时间增加到2 s时, 每个用户带给企业的收益下降了4.3%[16], 亚马逊主页加载时间每增加0.1 s, 其销售额下降1%[17]. 3)威胁用户安全. 有些应用如自动驾驶对实时性要求很高, 快速响应对于实时识别交通状况并作

出反应至关重要. 对于延迟敏感度不高的 BE 应用，为优先保障 LS 应用的性能，集群管理系统往往将其视为“二等公民”对待[5]，很难保障 BE 应用的资源公平性. 例如，当 LS 应用发生性能突变时，集群管理系统通常会暂停 BE 应用，甚至将其驱逐出该节点并重新调度其至其他节点上运行[3]. 在阿里巴巴的混部集群中，为保障 LS 应用的性能，一些 BE 应用不得不遭受反复的重调度[5]，这些运行一半的应用遭受重调度后需要重新运行，这不仅带来了极大的资源浪费，同时也会影响到 BE 应用的正常运行.

从混部的初衷出发，混部的主要目的是在保障应用性能的前提下充分利用闲置的资源以提升资源使用效率. 因此，如何保障混部应用的性能成为混部研究的关键. 对于 LS 应用，需要保障其响应延迟；对于 BE 应用，需要保障其运行时间和吞吐率，以充分利用物理资源处理更多的任务.

针对上述问题和挑战，本文从“干扰发生—干扰消除”角度对应用性能保障问题进行拆分，并从应用和集群特征分析、干扰检测、单节点资源分配和集群作业调度 4 个方面进行阐述. 具体内容包括：1)特征分析是性能保障研究工作的基础，通过分析应用的资源使用偏好、在不同干扰下的性能表现、应用在时间维度上的运行模式、集群的资源类型、应用运行方式和资源使用效率等特征，形成一套完整的应用和集群画像，加强干扰检测、资源分配和作业调度系统对应用和集群特征的感知；2)干扰检测是应用性能保障的前提，只有能及时发现或预测到应用在哪些资源维度上发生了性能干扰，才能快速响应来保障应用性能；3)单节点资源分配是从单节点的资源管理层面来保障应用性能，通过为应用分配隔离的资源从根本上降低应用之间对共享资源的竞争，从而降低应用间干扰、保障应用性能；4)集群作业调度可视为宏观层面的应用性能保障方法，即从集群的全局视角出发，在为应用选择合适的节点时就将降低性能干扰作为调度约束的条件之一，避免将易相互干扰的应用调度到同一节点混合部署，从作业调度角度保障应用的运行时性能，同时通过紧凑的作业放置来最大化集群资源利用率. 这 4 部分相关工作之间的关系如图 1 所示，特征分析形成的集群和应用画像信息为集群调度和单节点资源分配提供了指导；集群作业调度器根据集群的全局信息和应用特征对用户提交的应用进行调度(宏观)；待应用调度到节点后对其进行单节点资源分配，在应用运行过程中，干扰检测模块会持续进行干扰检测，并根据应用受到干扰的情况决定是否触发资源再分配(微观)；节点的状态信息会定期上报集群，由中心节点更新全局信息. 这 4 部分研究内容层层递进，从应用和集群的特征分析切入，再从微观和宏观 2 个层面共同保障混部应用性能. 此外，由于混部应用和集群特征等不同，性能保障工作所面临的挑战和问题复杂度也各异，例如单位资源上混合部署的应用数量和应用的运行方式会直接影响到共享资源的竞争强度和搜索资源空间的时间开销. 因此，在对每部分研究内容进行讨论和分析时，本文从问题复杂度的角度对性能保障问题进行分类，分别探讨应用和集群特征、资源干扰维度和混部应用个数等特征对制定应用性能保障策略的影响，以此对相关研究工作的适应场景进行分类和总结.

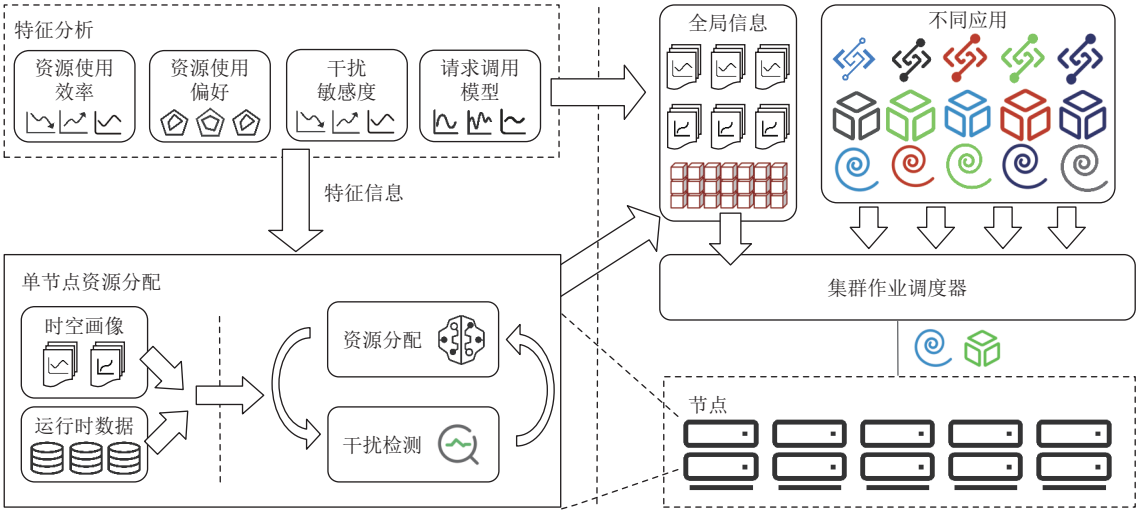


Fig. 1 Research framework of guaranteeing the performance of co-located applications

图 1 面向多应用混部的性能保障研究框架

本文的主要贡献包括4个方面:

1) 对操作系统级别和硬件级别的资源隔离技术进行总结. 指出虽然这些资源隔离技术给予了更细粒度的资源操控方式, 但如何根据应用的具体需求设计高效的软硬件协同的应用性能保障方法依旧是尚未定论的开放性问题.

2) 从微观和宏观的角度对应用性能保障的相关工作进行分析和总结. 具体包括特征分析、干扰检测、单节点资源分配和集群作业调度4个方面, 明确这4个方面的研究工作对混部应用性能保障的作用.

3) 从问题复杂度角度对干扰检测、单节点资源分配和集群作业调度的相关工作进行分类总结. 具体讨论因应用和集群特征、资源干扰维度和混部应用个数等不同对制定应用性能保障策略的影响. 指出因混部特征的不同而导致的问题复杂度各异是推动性能保障工作不断演进的助推剂.

4) 对面向高密度混部场景下应用性能保障研究的发展方向和所面临的挑战进行讨论和总结. 指出全栈式的软硬件协同方法是保障高密度混部下应用性能的趋势.

1 资源隔离技术

在混部场景下, 为应用提供性能隔离是保障应用运行时性能的关键. 性能隔离是指应用在混部场景下运行时像独占整个计算机资源一样, 无法感知其他应用的存在且不会被其他应用干扰. 为达到性能隔离的目标, 应用要能够独占分配给它的资源. 本节首先从不同资源类型的维度来介绍资源隔离技术和工具, 然后再从体系结构和软件设计的角度来介绍资源隔离方案.

1.1 资源隔离技术

根据共享资源类型, 资源隔离技术可分为功耗、末级缓存资源、内存带宽资源、CPU时间片资源、内存资源、网络带宽资源、磁盘带宽资源等7类。

1)功耗. 数据中心每年的电费开销都达到了上亿美元[18], 因此功耗是影响数据中心经济效益非常重要的指标. 在硬件层面可使用动态电压频率调节(dynamic voltage and frequency scaling, DVFS)和高级配置与电源接口(advanced configuration and power interface, ACPI)等技术来调整每个CPU核的频率[19]. DVFS通过调整CPU核的频率和电压来调整该核的性能和功耗; Linux内核中的ACPI CPUfreq驱动程序可以通过ACPI控制特定CPU的频率从而调整功耗.

2)末级缓存资源. 末级缓存是非常重要的片上资源, 研究表明, 应用对末级缓存的无序竞争会影响应用的性能和服务质量[3,20]. 当前一些工作尝试从软件层面[21-24]或硬件层面[25-28]来实现对末级缓存资源的划分, 操作系统的“页着色(page coloring)”技术即是其中之一. 页着色技术的核心思想是通过控制地址映射来实现硬件资源的管理, 从而实现末级缓存资源的划分. 随着处理器核数的不断增加, 为在硬件上提供资源管理的能力, 2016年英特尔公司在最新硬件上加入了资源调配技术(resource director technology, RDT), 该技术支持细粒度的末级缓存、内存带宽等硬件共享资源的管理, 可以给每个CPU核或进程分配隔离资源. 其中缓存分配技术(cache allocation technology, CAT)通过路划分(way-partition)为CPU核或进程分配隔离的LLC资源, 避免应用之间对末级缓存资源的竞争.

3)内存带宽资源. 混部应用之间对于内存带宽资源的竞争会直接影响到应用的访存请求延迟, 从而影响应用的性能[29], 因此内存带宽资源的隔离对上层应用性能保障至关重要. 一些研究工作尝试从软件层面对内存带宽资源进行管理, 例如MemGuard[30]和文献[31−32]通过控制CPU核的访存请求速率来间接控制内存带宽; PARTIES通过为应用分配隔离的LLC资源来间接减少内存带宽竞争[9], 当应用分配的LLC容量较小时, 应用会倾向于使用更多的内存带宽资源[33-34]. 为了在硬件上支持内存带宽资源的隔离, 英特尔在RDT中引入了内存带宽分配(memory bandwidth allocation, MBA)技术, 该技术可为每个CPU核或进程分配隔离的内存带宽资源, 减少应用之间在内存带宽层面的竞争.

4)CPU时间片资源. CPU时间片是重要的计算资源之一, 当前操作系统提供了2种分配CPU资源的方式: 第1种是以CPU核为粒度进行资源分配, 每个应用独占1个或多个CPU核资源; 第2种是以CPU时间片为粒度进行资源分配, 多个应用可共享同一个CPU的时间片. Linux的Cgroups提供的CPUset和CPU子系统与taskset命令用于支持以上2种资源分配方式. 应用独占CPU资源虽然能保证资源隔离、减少共享资源的干扰, 但由于应用往往无法充分利用独占的资源, 因此该资源分配方式会导致较低的资源使用效率. Cgroups的CPU子系统的CPU share和CPU quota机制可以限制应用在单位时间内可使用的时间片, 从而让多个应用共享同一CPU资源, 但该机制会使CPU核内共享资源(如LLC)竞争加剧.

因此，云服务提供商通常会采用超卖的方式综合使用上述 2 种资源分配方法[4-5,7,35]，对于 LS 应用以 CPU 核为粒度分配资源，对于 BE 应用，使其充分利用节点上空闲的 CPU 时间片，通过在有限的资源上超额售出资源以提升 CPU 资源使用效率.

5）内存资源. 内存是重要的计算资源之一，当前的计算机系统都采用虚拟存储技术来实现内存资源的管理. 每个进程都拥有独立的地址空间，在执行指令时，通过内存管理单元（memory management unit, MMU）将指令中的虚拟地址转换为主存上的物理地址，实现不同进程地址空间的隔离. 操作系统负责分配并管理 MMU 的地址映射，将不同进程的虚拟地址空间映射到不同的物理地址空间，从而实现内存资源的隔离. 在系统软件层面 Linux 的 Cgroups 提供的内存子系统可用于限制每个应用能使用的最大内存容量. 与 CPU 资源超卖不同的是，内存超卖或者在应用运行时降低其内存资源可能会导致应用程序因内存不足而被操作系统驱逐. 但近些年，一些工作利用页压缩[36-37]或 SWAP 设备[38-39]等进行内存资源超卖，以进一步提升内存资源使用效率. 例如，在 2019 年谷歌内存超卖数量已与 CPU 相当，而 2011 年谷歌并没有对内存资源进行超卖[7].

6）网络带宽资源. 网络资源是数据中心基础设施的核心资源之一，用于支持服务器之间数据传输、请求的接收与返回等. 例如，谷歌利用扇形结构来构建其搜索应用[1]，通过请求树来分解请求并合并结果，这需要网络资源的支持；在阿里巴巴的混部集群中，因为 BE 应用采取了存储与计算分离的架构[5]，BE 应用在运行时需要消耗网络资源从远端存储服务器拉取计算使用的数据. 应用对于网络带宽资源的竞争会引起网络包排队时间增长，进而影响 LS 应用的延迟和 BE 应用的运行时长，从而影响应用的性能和服务质量. 在软件系统层面，可使用 3 种方法对网络带宽资源进行隔离：①流量控制，Linux 的 qdisc 流量控制机制[40]可对网络带宽进行限制[9]；②基于优先级的网络包划分[41-43]，即通过为网络包划分不同优先级的方式使得高优先级的网络包可直接越过低优先级网络包的发送队列，从而降低排队时间；③网络带宽划分[44-46]，即通过为运行时应用配置合适的网络带宽来降低应用对网络带宽资源的竞争.

7）磁盘带宽资源. 磁盘带宽资源是数据中心基础设施核心资源之一，用于支持数据的读取与存储. Linux 的 Cgroups 提供的磁盘子系统可用于限制每个应用可使用的最大磁盘读带宽，减少多个进程共同读写同一块磁盘而产生的干扰.

1.2 软/硬件级别资源隔离方案

在混部场景中，应用对共享资源的竞争可能分布于整个系统栈中，为保障应用的性能隔离，需要为应用提供一个良好的资源隔离环境而不被其他应用干扰. 根据应用之间共享资源的程度不同，应用的隔离环境可分为不同类型的资源隔离抽象，如硬件分区、虚拟机、容器、函数等.

在硬件分区方面，PARD[47]提出了一种标签化的冯·诺伊曼体系结构，它通过为用户提供逻辑域来实现资源隔离，通过打标签的方式将硬件资源划分为不同的地址空间来实现性能隔离. 性能标签还可将上层应用的服务质量需求传递到底层硬件之中，共享的硬件部件能根据标签识别出来自不同应用的请求和请求处理的优先级，以实现应用之间的区分化服务，从而优先保障关键应用的性能. 此外，虚拟化技术的使用使得 1 台服务器能像 2 台或更多台独立运行的服务器一样为应用提供独立的运行环境[48]，其中虚拟机和容器被广泛应用于数据中心的混部场景中. 虚拟机是一种创建于硬件资源之上的虚拟环境，是操作系统级别的资源隔离，它通过模拟出包括 CPU、内存、网络等整套硬件资源为应用提供可靠的性能隔离. 容器是一种轻量级的虚拟化技术，与硬件分区和虚拟机所不同的是，容器之间共享同一份操作系统内核，是进程级别的资源隔离，这样的设计使得它占用更少的资源并获得更快的启动时间，并能有效地提高资源使用效率.

1.3 总 结

系统软件和硬件层面的一系列资源隔离技术为混部场景下应用运行的性能隔离提供了技术支撑和保障. 从不同资源类型的维度出发，当前软/硬件级别的资源隔离工具如表 1 所示，这些工具给予了更细粒度的资源操控方式，系统软件能够根据应用运行

Table 1 Summary of Resource Isolation Tools

表 1 资源隔离工具汇总

资源隔离机制	资源隔离工具
CPU 隔离	Linux CPUset Cgroups, taskset
内存隔离	Linux Memory Cgroups
网络隔离	Linux qdisc
磁盘带宽隔离	Linux blkio Cgroups
CPU 功耗	DVFS, ACPI
末级缓存隔离	CAT
内存带宽隔离	MBA

时的请求调用特征,通过有效地使用软/硬件提供的接口设计资源分配策略,降低混部场景下应用对共享资源的竞争,从而增强关键应用的抗干扰性.但由于数据中心应用的多样性、负载特征的多样性和共享资源的种类较多,应用对共享资源的竞争可能分布于整个系统栈中,为保障应用的性能隔离,需要根据实际的资源竞争特征制定针对性的资源调整策略.因此,如何根据应用的具体需求设计高效的软/硬件协同的应用性能保障方法依旧是尚未定论的开放性问题.

2 特征分析

混部应用和集群的特征分析是性能保障研究工作的基础,这些特征数据可用于指导制定具体的性能保障策略.通过分析应用的资源使用偏好、在不同干扰下的性能表现、应用在时间维度上的运行模式、集群的资源类型、应用运行方式和资源使用效率等特征,可形成一套完整的应用和集群画像,该画像信息可用于加强干扰检测、资源分配和作业调度系统对应用和集群特征的感知.

近些年,随着云计算的飞速发展[49-50]、数据中心机器规模的日益扩大和应用种类的不断增加,各大云服务提供商为在提升资源使用效率、作业调度、应用性能保障等方面寻求更优的解决方案,纷纷开源了一部分他们生产环境的数据集供研究使用并公开了其所面临的挑战.例如,谷歌在2011年和2019年分别开源了其集群管理系统Borg的数据集[51];阿里巴巴在2017年、2018年、2020年、2021年分别开源了其混部集群的数据集、GPU集群数据集和微服务数据集[52];微软在2017年、2019年、2020年分别开源了其Azure虚拟机和函数服务相关的数据集[53].这些数据集的开源为相关领域学者提供了更广阔的视角.基于这些开源数据集和基准测试集,针对分析的主体不同,可将特征分析工作分为2大类:针对应用的特征分析和针对集群的特征分析.

2.1 应用特征

相比于在单物理机上运行的应用,在数据中心运行的应用性能的不确定性更大.例如程序在单物理机上运行1万次,其性能波动幅度低于1%[54],而在数据中心运行的真实应用,性能波动幅度超过了600%[55],因此端到端的用户体验也遭受着极大的不确定性[56-57].多方面因素造成了在数据中心运行的应用具有更大的性能波动性,如请求的大小不同、查询结果是在缓存或磁盘、应用的上下游依赖和应用间的性能干扰等.因此,深入理解应用性能变化的原因能更有针对性地进行性能保障.

从应用的自身特征来分析,目前影响应用性能的因素涉及到应用的开发语言、业务逻辑、资源需求、干扰敏感度、请求特征和上下游依赖等.下文将着重从应用的资源需求和干扰敏感度、请求特征、上下文依赖这3个方面进行探讨.

2.1.1 资源需求和干扰敏感度

应用对资源的需求量指的是应用在某一负载强度下保证应用性能所需的资源量.若某一资源类型为应用的主要消耗资源,则可称该应用为该资源密集型应用,如CPU密集型、缓存密集型应用等.由于应用负载的动态性[6,58],应用的真实资源需求是随着负载的变化而动态变化的.因此,在对应用进行混部时,为降低应用对共享资源的竞争,应根据应用的资源需求特征避免将对同一资源需求量大的应用放置在一起运行.

应用对某一资源类型的干扰敏感度指的是在不同程度的资源竞争下应用的性能下降程度,性能下降程度高表明应用对该类干扰的敏感度高,反之则敏感度低.目前,量化应用对特定资源干扰敏感度的最常用方法是通过构建针对特定资源的基准测试集并将其与应用混部运行[9,14,59-60],即通过手动注入干扰的形式对应用进行离线分析,在各种共享资源上通过调节资源竞争强度来量化应用性能下降程度.

一些研究发现,不同的负载强度下应用对干扰的敏感度也各异,根本原因在于当应用负载强度增强时,应用对资源的需求也随之增大,导致对共享资源的竞争也随之增强[9,14,59].因此,为量化应用在特定的资源隔离方案下的抗干扰性或验证资源分配方案的有效性,常使用应用在特定干扰或混部场景下能支持的最大负载量作为量化指标之一[9,61].与此同时,应用的资源需求特征和干扰敏感度等信息还可作为应用的画像信息之一来预测应用在不同资源分配方案或干扰场景下的性能变化[3,62-64],以此来指导应用的资源分配和作业调度,避免将对同一资源类型需求量大或敏感度高的应用放置在一起运行.

2.1.2 请求特征

根据应用的请求特征,可将应用分为3类:周期性的、持续性的和突发性的.周期性应用在请求间隔和请求频率上表现为每隔一定时间就会出现单个请求峰值[65-67];持续性应用在整个时间维度上表现为较为一致的请求频率[9,65];突发性应用表现为在不定时

间间隔下会出现不定强度的突发性请求[66,68-69].

随着应用请求特征的不同，其性能保障的复杂度也会随之变化. 对于周期性应用和持续性应用，因其请求和负载强度特征明显且单一，可通过预测应用未来的负载情况来指导资源分配和作业调度[58]. 同时，还可根据应用请求的昼夜规律，通过在夜间调度更多的 BE 应用来分时复用在线应用的空闲资源[5]. 对于突发性应用，因其请求没有规律性且难以预测，使得应用的性能保障更为复杂. 对于突发性应用需要在干扰产生时能做到及时响应和处理.

2.1.3 应用间依赖关系特征

应用在运行时可能会需要 1 个或多个上游应用的输出结果作为输入[55]，因此从全局角度看应用之间的依赖关系，可将其绘制成一个复杂的应用间调用图[70]. 其中，调用图中的节点为应用，节点间的连线代表应用之间有调用关系且线的方向为应用间的调用方向. 不同的请求内容会通过不同的应用链处理后向用户返回结果.

文献 [71] 表明，应用之间的复杂依赖会引发排队延迟效应，即应用因其上下游应用的请求排队问题所造成的性能下降不会马上显示，而可能会在几个单位时间后才出现. 在该场景下，排队延迟效应使得一些通过预测应用性能来进行资源分配或作业调度决策的工作有效性下降，原因在于这些工作主要针对应用下一时间戳的性能进行预测，而因排队延迟效应导致的性能下降却要在几个时间戳后才能显现出来. 因此，需要基于应用间的依赖关系构建更加具有前瞻性的性能预测模型.

同时，由于应用之间的依赖关系引入的反向压力（backpressure）效应和级联（cascading）效应等使得集群中的应用管理变得更加复杂[72-73]. 反向压力是指在应用依赖链中由于处于后端的应用饱和，导致前端应用也随之饱和的现象. 此时，集群管理系统可能会错误地扩展前端饱和的应用，而忽略了根本原因在于后端应用. 若应用的饱和问题没有及时得到解决，由应用饱和所造成的服务质量（quality of service，QoS）违例问题会随着反向压力效应逐级向上传递，造成 QoS 违例的级联效应. 级联效应是指在应用依赖链中由于处于后端的应用饱和而发生 QoS 违例，该性能问题通过依赖链逐级向上传递，导致其上游应用均发生 QoS 违例的现象. 例如，某应用利用同步请求向其下游应用请求数据时，由于一直在阻塞等待下游返回结果而导致其自身延迟增加，但此时发生性能问题的根本原因在于下游应用，若不考虑应用的上下游依赖而盲目地为该应用增加资源或扩展实例，不仅无法及时地改善性能还会造成资源浪费. 因此，当某一应用发生性能下降时，在采取具体措施之前需要考虑应用之间的依赖关系对端到端性能的影响.

另外一些研究发现，应用之间依赖关系的形态也会影响应用的性能，例如树的深度和宽度、节点的出度和入度等[1,70,74]. 谷歌的搜索应用之间是一种扇形（fan-out）依赖关系，该扇形由 1 个根节点和 N 个叶子节点组成[1]. 根节点的请求处理时间取决于所有的叶子节点，只有当所有的叶子节点均返回结果时根节点才能向用户返回结果. 当叶子节点的数量增加时，即便叶子节点发生 QoS 违例的概率很低，根节点应用发生 QoS 违例的概率也会随之增加. 假设 1 个叶子节点应用有 99% 的概率会在 1s 之内向根应用返回结果，而根应用需要等待 100 个叶子应用均返回结果后才能向用户返回最终结果，那么对于 fan-out 的依赖模型而言，叶子节点个数从 1 增加到 100，会导致根节点发生 QoS 违例的概率从 1% 增加到 63%（$63\%=1-0.99^{100}$）. 如若在 fan-out 模型的基础上再增加深度，根应用的 QoS 违例概率会随着树深度的增加而被逐级放大. 与此同时，对于出度或入度较大的应用节点，可将其视为请求路径上的关键应用节点，若关键节点出现性能违例，其影响的范围会随着出度和入度的节点不断向四周扩展.

2.1.4 小 结

除 2.1.1~2.1.3 节所述的特征分析工作外，一些研究工作还聚焦于应用的类型[4-7]、性能变化特征[55]、应用在不同体系结构下的性能表现[33,75] 等. 因此，虽然应用在共享环境中运行时对共享资源的竞争会产生性能干扰，并且针对应用在各个维度上的特征不同，要解决的性能保障问题的复杂度也各异，但可以根据应用的这些特征有针对性地设计资源分配和作业调度策略，以保障应用性能.

2.2 集群特征

从全局视角来观察，集群特征主要包括集群的资源类型、资源使用情况、负载均衡和作业运行方式等 4 个方面. 其中，由于一些应用对硬件资源类型很敏感[66,76]，在作业调度时应考虑到集群资源的类型不同，保证应用性能；由于不同类型的应用对 CPU 和内存资源的需求各异[5-6]，在进行资源分配和作业调度时应考虑整体资源分配的均衡性；由于不同类型应用的运行方式会影响集群的资源分配和使用情况，在制定资源分配方案时需根据应用间运行方式的不

同有针对性地设计资源分配策略. 因此, 深入理解集群的资源类型、运行的应用种类、应用的运行方式和各维度资源分配及使用情况等特征能帮助系统开发者更有针对性地设计资源调度和性能保障策略. 下文将依次从集群的资源类型、资源分配和资源利用率 2 个维度对集群特征进行探讨.

2.2.1 资源类型

许多研究工作均假设集群中的资源为同构资源, 即所有服务器的资源配置均相同. 同构资源能简化集群的作业调度和负载平衡等问题, 并且减少了集群的维护负担. 然而, 在真实生产环境中, 集群通常由多种不同类型配置的服务器组成[4,6-7]. 集群资源异构性的主要原因是随着时间的推移和技术升级, 数据中心的服务器逐步地增添和替换, 因此在数据中心中通常有 10~40 种不同配置的服务器[6,76]. 在进行集群作业调度时, 需要考虑到异构资源对应用性能的影响. 研究表明未考虑资源异构性的作业调度器会使应用的运行速度平均降低 22%, 有些甚至会慢 200%[76] . 这种因忽略了资源的异构性而导致的性能下降, 不仅会影响应用的性能, 还会造成集群的资源浪费. 因此, 为了使用户能更方便地使用异构资源, 谷歌抽象出了谷歌计算单元(Google compute units, GCUs)概念[7], 通过为每种机器类型设置 GCU 到 CPU 核的转化比例, 从而保证一个 GCU 为应用提供的计算资源是相同的. 同时, 集群中还会有一些不同类型的计算硬件, 例如 GPU、FPGA 或低功耗的 CPU 等[11,66,73,77], 因此在进行作业调度时要根据应用需求和资源情况制定相应的调度策略.

2.2.2 资源分配和资源利用率

从全局角度分析集群的资源使用特征可用于指导集群的作业类型配比和负载均衡调整. 一些研究发现集群的 CPU 资源使用率遵循很强的昼夜模式[5,38], 主要原因在于用户的行为具有昼夜特性, 因此集群可在夜间调度更多的 BE 应用来分时利用集群空闲的资源. 一些研究发现, 当前数据中心的资源分配和资源使用效率具有很强的不平衡性[3,6,58,78], 例如, 阿里巴巴混部集群的不平衡性主要体现在空间、时间和资源类型的不平衡, 空间与时间的不平衡主要是集群作业调度或负载不均衡所致[78]. 资源类型的不平衡主要是指 CPU 和内存资源使用的不平衡, 产生的原因有 2 个方面: 1)作业对资源使用的不平衡, 阿里巴巴的 BE 应用对 CPU 和内存资源的需求比为 1∶4[79], 即每使用 1 个 CPU 资源就需要消耗 4GB 的内存资源. 2)应用类型和运行方式的影响, 阿里巴巴 90% 以上的 LS 应用和部分 BE 应用为 Java 应用, 并且这些 Java 应用被封装在容器中运行. 然而, 由于缺乏有效的内存重分配技术来回收 Java 虚拟机(Java virtual machine, JVM)中的内存并将其重新分配给其他应用, 因此内存资源的占而不用现象将不断扩大[5]. 此外, 大量封装在容器中的 JVM, 使得在 JVM、容器和底层操作系统之间进行整体的内存资源管理变得更为复杂.

2.2.3 应用运行方式

除 2.2.1 节和 2.2.2 节分析的应用昼夜模式和资源使用的不平衡外, 应用在集群上的运行方式也会影响到集群资源分配策略的制定. 应用的运行方式是指, 应用是以虚拟机、容器或二进制等形式在物理机上运行. 例如, 在阿里巴巴的混部集群中, LS 应用运行在容器中, 而 BE 应用以二进制的形式直接在物理机上运行[5,80]; 在微软的 Azure 集群中, 应用均运行在虚拟机中[68]; 一些研究工作假设应用均运行在容器中[9,66,81]; 在无服务计算场景下, 有些函数应用以沙盒的形式运行在虚拟机中[82]. 应用运行方式的多样性使得集群资源管理系统需要在 JVM、容器、虚拟机和底层操作系统之间进行协调调整, 使得资源管理变得更加复杂. 因此在设计作业调度和资源管理策略时, 需要针对应用具体的资源需求和运行方式制定策略.

2.2.4 小 结

总结来说, 在制定集群的资源分配策略和应用性能保障策略时, 不仅要关注应用自身的特征, 还需要关注集群资源的异构性、资源分配策略和应用运行方式等对应用运行效率的影响, 从微观和宏观 2 个角度共同指导集群资源分配和性能保障策略的制定.

2.3 总 结

关于应用特征和集群特征分析的代表性工作如表 2 所示. 这些研究工作从真实的生产环境数据或代表应用入手, 对应用和集群的多个维度进行深入分析, 揭示了影响应用性能和集群资源使用效率的关键因素和挑战. 应用和集群在各个维度上的多样性特征打破了许多针对特定应用类型和特定环境的资源与作业调度策略, 服务器和应用因其特征的多样性需要被区别对待. 根据所要解决的问题、应用与集群的特征不同, 问题的复杂度也随之变化. 因此, 在不同的混部场景下, 应全面地从微观(应用特征)和宏观(集群特征)角度出发, 有针对性地设计性能保障策略.

3 干扰检测

为了解决混部场景下应用性能干扰问题, 首先

Table 2 Summary of Feature Analysis Research Work

表 2 特征分析研究工作总结

研究工作	应用类型	应用特征					集群特征	
		资源分配	干扰敏感度	性能特征	请求特征	依赖特征	资源类型	资源利用率
文献 [5]	LS, BE	√			√		√	√
文献 [6]	LS, BE	√			√		√	√
文献 [33]	函数应用	√	√	√	√			
文献 [55]	LS			√				
文献 [58]	数据分析类	√			√		√	√
文献 [67]	函数应用	√		√	√			
文献 [68]	黑盒应用	√						
文献 [70]	微服务			√	√	√		
文献 [80]	LS	√					√	√

注: “√”表示具备特征.

要能及时检测或预测干扰的发生. 最直接的方法是通过实时监控应用的直接性能指标来判断应用是否受到干扰. 在无法获取直接性能指标的场景下, 可采用间接性能指标来判断应用在各个资源维度受到的干扰. 然而, 随着资源干扰维度的增加, 应用在多资源维度下的行为愈加复杂, 单一的间接性能指标已无法全面地展现应用在多个资源类型维度下受到的性能干扰, 此时干扰检测的问题复杂度随着资源类型的增加而增加. 在这种场景下, 通常通过构建应用性能干扰模型来预测应用的性能变化, 根据预测出的性能来判断是否发生性能干扰. 然后, 根据干扰的具体情况采用不同的性能保障策略.

3.1 单资源维度干扰检测

直接性能指标, 如应用延迟, 最能直观体现应用的性能变化情况, 因此, 应用延迟常作为直接性能指标来判断应用是否受到干扰, 并用来量化由于干扰造成的性能下降程度[3,9,14,64,83]. 例如, Bubble-up[62] 利用统计学方法, 通过离线分析构建应用的延迟与内存压力的敏感度曲线来刻画应用对内存干扰的敏感度, 敏感度曲线量化了应用在不同的内存压力下应用性能的下降程度, 在相同的内存压力下, 应用的延迟升高的越多, 代表该应用对内存资源的干扰越敏感. 应用在运行时根据敏感度曲线能准确预测应用由于内存资源的竞争而导致的性能下降, 同时敏感度曲线还可用于指导混部应用之间的资源分配和集群层面的作业调度.

然而, 在生产环境中, 由于无法及时获取应用的延迟信息, 一些服务提供商利用间接指标来衡量应用的性能变化. 间接性能指标是指应用在软件系统或硬件层面的指标, 可用于量化应用在某一资源类型下受到的干扰程度或性能变化情况. 谷歌发现, 计算密集型应用的 CPI(cycles per instruction)指标可以准确地反映应用在 CPU 资源维度所受到的干扰[4]. 对于计算密集型应用, CPI 不仅与 BE 应用的吞吐率有很强的正相关性, 与 LS 应用的延迟也有很强的正相关性, 因此其干扰检测工具 CPI2[84] 利用 CPI 指标来量化应用的性能变化, 并通过应用的 CPI 概率分布来判断应用是否在 CPU 资源维度受到干扰. 为寻找干扰源, CPI2 通过计算时间窗口内受干扰应用的 CPI 与其他应用 CPU 利用率的相关系数来寻找干扰应用, 最终利用 CPU 带宽控制机制来降低干扰应用对 CPU 资源的使用, 该方法已在谷歌集群中被大规模使用并验证了其有效性. Bubble-flux[63] 发现应用的 IPC(instruction per cycles)指标与应用的平均查询延迟有很高的相关性, 因此利用 IPC 作为反映应用性能的指标, 通过运行时探测技术来构建应用的 IPC 与内存压力的敏感度曲线. 在敏感度曲线中, IPC 值下降得越多代表应用的性能下降越多, 即代表应用对内存干扰越敏感. 利用该敏感度曲线预测应用在内存资源维度受到的干扰程度, 依此来调整混部决策. Bubble-flux 通过利用应用运行时的 IPC 数据, 不仅解决了 Bubble-up 需要提前进行离线分析的局限性, 还能捕捉到应用的真实运行状态(例如负载和输入的变化).

同时, 系统软件层面的指标(如排队时间)也可作为干扰检测的间接指标. Shenango[85] 利用线程和网络包的排队时长来量化评估 CPU 资源的繁忙程度, 若应用的排队时长增加则推断应用的计算资源可能

趋于饱和,并依此判断是否需要给应用分配更多的CPU资源.然而,在混部时应用之间对于共享资源的竞争可能涵盖多个不同的资源类型,而CPI2与Shenango所用的间接性能指标仅适用于CPU资源层面的干扰检测,Bubble-up和Bubble-flux仅适用于内存资源层面的干扰检测,而忽略了其他资源维度可能产生的干扰.并且,CPI2假设干扰应用的CPU利用率和受干扰应用的CPI之间存在线性相关关系,该假设忽略了CPU内部共享资源,如LLC、内存带宽的复杂性.

总结来说,应用受制于公有云环境或大规模数据中心的黑盒特性和隐私条款,无法直接获取延迟信息作为直接性能指标,因此选取合适的间接指标来判断应用是否受到干扰并量化由于干扰造成的性能的下降程度成为关键.指标的选取应既能体现应用在干扰下性能的变化,又需要综合考虑到多维度的资源竞争情况.

3.2 多资源维度干扰检测

为了从多个资源维度综合判断混部场景下的资源竞争情况,可通过引入更多的间接性能指标对多个资源维度逐一判断.Caladan[86]在Shenango的基础上引入了处理时间、内存带宽、LLC未命中率等多指标作为判断依据,例如通过测量全局的内存带宽利用率来判断DRAM资源的饱和程度,通过测量每个核上的LLC未命中率来判断应用对于LLC资源的利用情况,从而从多个共享资源维度来综合判断CPU资源的竞争情况,并通过调整应用的CPU核数量来减少内存带宽方面的干扰,通过禁止使用CPU核的sibling来减少超线程的干扰.Alita[87]利用硬件层面的锁频率(split lock frequency)指标来判断应用间对内存总线锁的竞争情况,利用CPU的资源利用率和CPU核的温度来判断CPU功耗层面的干扰,最终对受干扰应用通过小步调整资源的方式来缓解竞争带来的干扰.DeepDive[88]利用CPI以及软/硬件层面的多个指标来综合判断应用在一级缓存(first-level cache, L1)、二级缓存(second-level cache, L2)、内存和网络等资源层面的竞争程度,并通过聚类算法进行干扰检测,利用retired指令数的变化情况来量化应用的性能下降程度.

然而,随着要管理的资源种类增加,应用的行为在多维度资源干扰下变得愈加复杂,利用间接指标分别判断各个资源维度的干扰已很难综合判断应用的受干扰情况.因此,需要通过构建函数模型来量化应用的受干扰程度.模型的输入为系统软件或体系结构层次指标,一般通过干扰注入还是实时监控获取;模型的输出为预测的应用性能指标,如应用延迟、IPC等.通过构建应用性能与多资源维度干扰下的关系函数来预测应用在不同负载强度和资源竞争情况下的性能变化,依此来判断应用是否受到干扰并量化受到干扰的程度.这些指标因便于获取且通用性强,并能解决单一指标下无法涵盖某些干扰场景的问题,被广泛用于模型构建中.

干扰注入法是构建应用性能-干扰模型最常用的方法,指利用精心设计的资源干扰程序模拟多个资源维度的竞争[9,59-60],通过改变干扰程序的强度模拟共享资源的竞争强度,从而获取目标应用的干扰-性能变化数据,利用这些数据快速构建应用的性能-干扰模型,最终通过构建好的模型来预测应用的性能变化.GAugur[89]通过对目标应用在多个资源维度上注入干扰来获得应用在多个资源维度上的性能干扰敏感度数据,并利用分类模型判断当前应用性能是否满足QoS需求,利用回归模型量化因混部而造成的性能下降程度.Amoeba[90]利用干扰注入方法获取应用的敏感度信息并构建应用延迟-干扰模型,根据应用在运行时各个资源维度上的压力来预测应用的延迟,最终通过改变应用的部署方式来消除多维度资源干扰对应用性能的影响.

然而,由于干扰注入法等同于对应用进行穷举式离线分析,需要引入额外的时间与资源开销,另一类工作利用应用的运行时数据或统计学习和神经网络等方法构建性能-干扰预测模型.文献[91]发现在给定的处理器平台上,用于体现应用性能干扰的函数模型都是相同的,只有函数模型的具体系数由应用本身决定,基于此该工作提出了一种2阶段方法,该方法利用回归分析来分别构建抽象函数模型和实例化抽象函数模型,以降低构建应用性能干扰模型的开销,最终利用该模型来预测应用在多维度干扰下的性能变化情况.MAGI[92]利用神经网络和应用的静态特征指标(如各类型指令数等)与动态特征指标(如CPU周期数、各级缓存为命中率等)拟合出应用的IPC与各维度指标的关系函数模型,通过模型预测的IPC值来判断应用是否发生性能干扰并量化性能的下降程度,然后通过敏感度分析法来寻找最大干扰源.

总结来说,利用统计学、机器学习或神经网络等方法构建的应用干扰模型能捕捉到应用的深层次特性,并能快速识别干扰的发生,然后通过利用资源分配、作业调度或改变部署方式等方法消除性能干扰.

然而，该方法有 3 点局限性：1）模型训练时间开销大，无论是利用干扰注入的离线分析方法还是实时监控来获取应用干扰数据，机器学习和神经网络等方法均需要通过不断训练模型来提升性能预测的精确度；2）模型训练需要大量的训练数据；3）机器学习和神经网络等模型因其黑盒特性导致可解释性差.

3.3 总 结

干扰检测是混部应用性能保障的关键一步. 3.1 节和 3.2 节的研究或通过监测直接性能指标或间接性能指标来判断应用在哪些资源维度发生了干扰，并量化应用因干扰所导致的性能下降程度；或通过构建应用的性能-干扰模型来预测应用的性能变化，根据预测的性能来判断应用是否受到干扰，这 2 种方法均通过量化资源竞争对应用性能的影响程度来指导后续的应用性能保障. 目前，通过利用干扰检测方法构建“性能监测—发现干扰—消除干扰”的动态反馈调节机制来保障应用性能的方法已被广泛应用到真实场景中. 该方法通过观察/预测应用性能来做到干扰的及时发现，并通过及时消除干扰来保障应用的性能. 基于干扰检测的性能保障相关工作如表 3 所示.

Table 3 Summary of Interference Detection Research Work

表 3 干扰检测研究工作总结

研究工作	性能指标	检测方法	涵盖资源干扰维度						
			CPU	内存	网络带宽	磁盘 I/O	功耗	LLC	内存带宽
Bubble-up[62]	应用延迟	统计方法		√					
Bubble-flux[63]	IPC	统计方法		√					
CPI2[84]	CPI	统计方法	√						
Shenango[85]	线程、网络包排队时延	监控	√						
Caladan[86]	排队时延迟、内存带宽利用率等	监控	√					√	√
Alita[87]	SQ.SPLIT LOCK，CPU 利用率等	统计方法					√	√	√
DeepDive[88]	CPI	聚类算法	√	√	√	√			
GAugur[89]	帧速率	机器学习	√					√	√
Amoeba[90]	应用延迟	统计方法	√	√	√	√			
MAGI[92]	IPC	神经网络	√					√	

注：“√”表示具备该维度.

4 单节点资源分配

混部场景下，应用间产生性能干扰的根本原因在于对共享资源的无序竞争，因此为应用分配隔离的资源成为保证应用性能隔离的关键. 资源分配的理想状态为资源的分配量与应用的资源需求量恰好匹配. 然而，由于应用种类的多样性、应用负载的动态性和混部场景的复杂性等因素，不同应用的资源需求是随着负载和环境的变化而动态变化的，因此在多维度资源空间下寻找单节点最优的资源分配方案成为一大研究热点. 多应用间资源分配问题类似于最优化问题，在满足节点资源总量、应用资源要求和应用性能要求等一系列约束下进行最小化资源分配，以达到在保障应用性能的前提下尽可能地混合部署更多应用以提升资源利用率的目标. 该优化问题的复杂度受资源类型数量、混部应用个数、应用性能与资源数量的函数关系是否为凸函数等因素的影响，在不同的复杂度下需采用不同的最优化算法.

4.1 基于梯度下降

在应用性能与资源数量关系是凸函数且混部应用数量不多的情况下，因资源搜索空间较小且混部场景简单，可对发生 QoS 违例的应用利用梯度下降法来寻找全局最优资源分配方案. 基于梯度下降的资源分配方法是指通过对目标应用增加或减少确定步长的资源数量，并结合应用实时性能变化来寻找最优资源分配方案的方法.

最具有代表性的研究工作是 Heracles[14] 和 PARTIES[9]. Heracles 发现谷歌的 LS 应用在 CPU 核和 LLC 资源维度下应用性能变化是凸函数，因此它利用应用延迟作为直接性能指标来进行干扰检测，在检测到应用发生 QoS 违例时通过梯度下降的方法来保障 LS 应用性能. 对于目标 LS 应用，Heracles 依次在 CPU 核与 LLC 这 2 个维度上为其逐步增加资源以寻

找最优资源分配方案；对于 BE 应用，Heracles 降低其 CPU 频率和网络带宽来减少对 LS 应用的干扰. 从节点资源分配角度看，Heracles 将节点资源分为 2 大部分：一部分供单个 LS 应用使用；另一部分供所有 BE 应用使用，即单个 LS 应用与多个 BE 应用之间相互隔离，从而保证了延迟敏感型应用的性能隔离. 然而，由于 Heracles 将节点上的资源仅分为 2 部分，因此该工作只能支持单个 LS 应用和 BE 应用的混部. 为支持多个 LS 应用和 BE 应用混部，PARTIES 在 Heracles 的基础上进行了进一步的优化，对节点上运行的每个 LS 应用都进行了资源隔离，对已经发生或可能发生 QoS 违例的 LS 应用逐个通过梯度上升方法在计算和存储 2 类资源中寻找最优资源分配方案. 当 LS 应用的性能满足 QoS 要求时，PARTIES 和 Heracles 均利用梯度下降方法对 LS 应用收回部分空闲资源供 BE 应用使用，以提升资源使用效率.

总结来说，基于梯度下降的资源分配方法简单且易于理解，通过对目标应用在每个资源维度依次增加资源的方式来逐步寻找最优资源分配方案，以保障应用的性能. 为找到全局最优解，该方法要求应用性能与资源数量为凸函数，且局限于应用数量较少且资源维度较少的情况. 当资源维度增加时，资源搜索空间增大，在一个多维的资源空间中通过依次对每个资源维度逐步增加资源这种小步慢跑方式的时间开销也随之增大，很难在有限时间内找到最优解，同时对于请求特征为突发性的应用很难做到及时响应到位. 当混部的应用个数增加时，若多个 LS 应用同时发生 QoS 违例，这种 1 次只增加 1 个 LS 应用中 1 种资源的逐个调整方式易导致对目标应用无法及时响应的问题. 因此，在资源类型多且混部应用多的场景下，基于梯度下降的资源分配方法可能会出现反应不及时或难以找到全局最优解的问题.

4.2 基于经验规则

基于经验规则的资源分配方法是指根据已制定的规则对应用进行资源分配. 规则通常根据系统和应用的特性、具体需求或专家经验制定，用于达成特定目的，通过规则得出的资源分配方案不一定是全局最优解，但却是为达成特定目的的可行解之一. 该方法适用于对混部场景、应用特性或资源需求有较为深入了解的场景，能根据具体应用和场景特征设计合理的资源分配规则.

在公有云中，不同的资源分配方式，如按需分配和使用预留资源会直接影响到应用的性能和用户的经济效益. 因此，为了能通过灵活调整资源分配方式来保障公有云中应用的性能和高资源利用率，HCloud[83] 根据当前的资源分配方式、负载变化和应用特征提出了一种混合式的资源分配策略. HCloud 利用 Quasar[3] 来决定为应用具体分配多少资源，并根据系统的运行时信息设置了软阈值和硬阈值作为调整资源分配方式的规则线，在不同的规则线下，LS 应用和 BE 应用根据运行时性能来选择是按照应用实际需求来分配资源还是使用预留资源. 为了保障按照规则线来调整资源分配方式的有效性和灵活性，具体规则线的值可以根据系统的队列长度动态调整.

在私有云中，微软根据其混部场景中搜索应用具有突发性负载的特性以及搜索应用的 QoS 需求设计了性能隔离框架 PerfIso[69]. PerfIso 框架根据既定规则对 CPU 资源和磁盘 I/O 资源进行分配和调整，在保障搜索应用在突发性负载时满足性能约束的前提下，尽可能地让 BE 应用利用空闲资源. 对于 CPU 资源，为保障搜索应用在突发性负载下的性能，微软发现需要始终为搜索应用预留空闲的 CPU 资源来保障流量高峰时的用户体验；为提升资源利用率，PerfIso 同时允许 BE 应用在其他空闲 CPU 核上运行. 对于磁盘 I/O 资源，PerfIso 通过调整进程的优先级和权重来进行资源分配. 该工作的局限性在于，由于其始终为搜索应用预留了空闲的 CPU 资源，这些空闲的资源限制了进一步提升 CPU 资源使用效率.

在多应用混部场景下，保证各应用能公平共享资源份额是重要的约束条件之一，即要保证多应用间资源分配的公平性[93-94]. DRF[95] 提出了一种主资源公平分配策略，根据该资源分配规则可保障应用之间在多资源类型的混部场景下资源分配的公平性. 在此之前，最大最小公平分配策略是最常用的资源公平分配策略之一，同时它还被推广到了带权重的范畴，每个应用得到的资源数量与应用的权重成比例，并且能支持包括基于优先级在内的分配策略. 但因其局限于单一资源类型的分配，在多资源类型环境中需要新的机制来满足应用的多资源需求. DRF 将最大最小公平分配策略扩展到支持多种资源类型的公平分配策略，它根据规则将应用的资源需求中占总资源比重最大的视为主资源，主资源对应的资源份额称为主份额，其中，不同应用的主资源可能不同. DRF 力求最大化系统中最小的主导份额，从而实现多资源维度下的公平分配，当资源维度降为 1 时，DRF 退化为最大最小公平分配策略.

总结来说，基于规则的资源分配方法需要在非常了解集群应用和混部特征的前提下，对于应用的

主要需求能根据专家经验、应用和集群特征有针对性地制定资源分配规则.

4.3 基于启发式算法

启发式资源搜索算法是基于直观或专家经验构造的算法，在资源分配空间中，它能够通过指导搜索向最优解的方向前进从而降低了复杂性，使之能在可接受的时间开销内搜索出一个混部场景下资源分配的可行解，该可行解不一定是全局最优解. 这种探索资源分配空间的启发式方法相对简单、易于理解，并且能利用启发式知识来剪枝以减少搜索空间，从而消除组合爆炸. 该启发式的资源搜索方法与基于经验规则的资源分配方法的区别在于，后者根据既定规则和已知的资源量直接对应用进行资源分配，而前者通过在资源空间中搜索来寻找可行解.

爬山法(hill climbing)和向前搜索法(lookahead)是其中最简单的2类启发式搜索方法. 其中，爬山法要求应用的性能与资源数量之间的关系函数为凸函数，它总是沿着应用性能收益最大的方向进行搜索；而向前搜索法类似于穷举法，可适用于应用性能函数为非凸函数的场景. Whirlpool[96]利用爬山法计算应用的性能收益来决定将当前一路LLC资源分配给哪个资源组，最终对所有LLC资源逐个进行分配，该方法将平方阶的搜索复杂度降低为线性的搜索复杂度，降低了时间开销. 然而，Whirlpool通过逐个单位进行资源分配的方法对搜索空间进行剪枝，虽然降低了搜索复杂度，但是易错过某些最优资源分配方案. UCP[97]和KPart[98]利用向前搜索法对每种可能的资源分配方案都计算了性能收益，通过全局搜索来选择收益最大的方案. 该方法类似于穷举搜索法，其较高的时间复杂度限制该方法仅适用于应用数量和资源类型数量较少的场景. 此外，KPart和Whirlpool均通过合并应用的性能信息来计算不同资源分配方案的性能收益，该方法虽然利用了应用对资源使用偏好的多样性将干扰较低的应用混部在一起运行来降低干扰的影响，但当混部的应用数量增多时，这种按照概率计算性能组合的方式会随着迭代次数的增多而导致误差被放大，使得最终计算出的性能损失精确度降低，因此这2个工作都仅适用于应用数量较少的场景.

为解决穷举搜索法所带来的高额时间开销问题，Avalon[99]提出了基于二分搜索的资源分配器. 该分配器负责搜索CPU核和LLC这2个维度的资源，通过分析发现，应用的性能在CPU和LLC资源上是一个凸函数，从而保证了二分搜索能像穷举搜索一样得到最优资源分配方案. 针对目标应用，Avalon资源分配器首先将LLC资源设置为最大值，二分搜索CPU资源分配方案查找最优的CPU资源数量，待CPU资源确定后再继续二分查找最优的LLC资源数量. 该方法通过将搜索最优解的复杂度降低到log级别从而做到快速搜索最优解. CoPart[34]将资源分配问题表述为医院/居民的多对一匹配问题，通过维护LLC和内存带宽2个有限状态机来判断每个应用的状态，并向资源富裕的应用回收资源，将其重新分配以保证整体资源分配的公平性. Lin等人[100]对函数应用的上下游依赖信息进行建模，利用函数应用存在关键路径的特性提出了一种基于关键路径的算法PRCP，该算法通过贪心策略来递归优化关键路径上函数应用的内存配置，从而得到最优内存资源配置方案，该算法的目标在于解决预算约束下的应用最佳性能和性能约束下的最佳成本问题.

总结来说，虽然启发式算法利用专家经验剪枝降低了搜索资源空间的复杂度，但仅适用于资源类型较少的场景，原因有2点：1)应用在多资源维度下的性能关系函数更复杂，此时可能无法用凸优化算法来寻找可行解，并且启发式搜索算法易落入局部最优解；2)当要分配的资源类型增加时，待搜索的资源空间呈指数级增长，使得搜索最优解的复杂度增加.

4.4 基于资源分配模型

基于资源分配模型的资源分配方法是指利用数据挖掘、机器学习或神经网络等方法构建资源分配方案与应用性能的模型，模型的输入通常为应用的画像信息、节点负载情况、应用上下游依赖信息、资源类型与数量或应用运行时指标等，模型的输出为最优资源分配方案、预测的应用性能值或应用发生QoS违例的概率. 该方法适用于应用在多资源维度下应用性能与各维度资源数量的函数关系未知的场景.

针对多维度资源下应用性能与资源分配量的函数关系不是凸函数的场景，CLITE[61]和SATORI[101]利用贝叶斯优化来构建资源分配模型，从而在多资源维度下快速寻找最优解. 贝叶斯优化是一种通用的黑盒优化算法，用于求解未知目标函数的极值问题. 它首先对目标函数进行建模，即利用高斯过程回归计算每一个采样点函数值的均值和方差，然后根据均值和方差构造采集函数. 之后，利用采集函数来寻找下一个最有可能是极值的点，并将该点加入采样点集合中，重复这一搜索过程，直到迭代结束. 最后这些采样点中目标函数值最大的点即为最优解. 利

用贝叶斯优化来构建资源分配模型的优势在于其适合黑盒优化问题,无需应用的先验知识即可自动搜索出最优资源分配方案以最大化目标函数.根据构建的打分函数不同,贝叶斯优化可用于解决不同优化目标下的资源分配问题.为解决 LS 应用与 BE 应用混部时的性能保障与最大化资源使用率问题,CLITE 在运行时分别计算 LS 应用和 BE 应用的性能分数,通过构建 2 阶段打分函数来衡量当前资源分配方案的有效性.为保障多 BE 应用混部时应用间资源分配的公平性以及最大化系统吞吐率的问题,SATORI 在运行时根据节点信息动态调整打分函数中的资源公平性和吞吐率需求的权重值,以此来达到全局最优性能收益.

此外,机器学习也被广泛应用于应用性能关系函数未知的场景中.DRLPart[102] 利用深度强化学习构建了多资源分配模型,该模型通过系统运行时信息学习最优的资源分配方案.为了降低模型训练成本,通过构建性能预测模型来降低训练开销,并结合反馈调节机制解决可能出现的错误决策,以保障应用性能.相似地,Hipster[103] 使用了启发式和强化学习相结合的混合式资源分配方法.Hipster 将资源分配问题描述为马尔可夫决策过程,首先它根据应用当前所处的状态选择要执行的动作,再根据所选择的动作和当前状态估计执行该动作所能收到的奖励,来确定下一个时间间隔的资源配置;之后,它通过不断填充查找表来在线学习合适的资源配置.在模型学习阶段,Hipster 利用反馈控制循环将 LS 应用部署到对应的资源上,当应用负载较低时,为应用分配较小的 CPU 核和较低的 CPU 频率;反之则为应用分配较多的资源.在模型利用阶段,Hipster 使用查找表根据负载来选择合适的 CPU 核和频率设置,同时不断更新查找表的信息.与 Hipster 利用查找表类似,CuttleSys[104] 利用协同过滤方法来推断应用在不同资源配置下的性能和功耗,并通过动态维度搜索来寻找合适的资源配置方案,以保障 LS 应用的性能.

由于上述方法在构建资源分配模型时仅考虑了应用自身的性能变化,因此,其仅适用于无上下游依赖的单体式应用.然而,近年来云应用已经逐渐从单体式应用转变为有多层依赖关系的微服务,这种复杂的依赖关系为系统的资源管理带来了新的挑战,例如应用间复杂的拓扑使得端到端请求的关键路径不断变化[60],此外还加剧了排队延迟效应[71],应用因上下游应用的排队问题造成的性能下降可能会在几个单位时间后才表现出来,同时还引入了级联效应[72,105] 等.由此,根据应用的上下游依赖信息综合构建应用资源分配模型成为新的研究热点.

为解决因微服务之间的依赖关系导致的排队延迟效应问题,Sinan[71] 构建了一个 2 级资源分配模型:在给定资源配置下,第 1 级模型利用卷积神经网络预测应用在下一个时间戳的延迟;第 2 级模型利用卷积神经网络中提取的变量和提升树(Boosted Trees)来判断目标应用在下一个时间戳之后未来应用发生 QoS 违例的概率.Sinan 选择 2 级模型是因为若只预测应用在下一个时间戳的性能,会忽略应用因上下游依赖所导致的排队延迟效应,而排队效应所带来的性能下降需要在几个时间戳后才能显现出来;若直接预测应用在未来几个时间戳下发生 QoS 违例的概率,一种直观的方法是使用多任务学习神经网络,但该方法会严重高估应用的延迟和发生 QoS 违例的概率,预测准确率低.因此,这种 2 级资源分配模型可以用于解决复杂依赖下应用的资源分配问题.

在函数应用存在复杂依赖网络和关键路径会动态变化的场景下,为保障函数链的整体性能,FIRM[60] 提出了一个基于机器学习的 2 级资源分配模型.为减少需要调整的目标应用个数,第 1 级模型用于寻找延迟波动最大的应用并将其视为关键路径上的目标应用;第 2 级模型用于自动搜索资源分配空间并优化资源管理策略,它通过分析共享资源的竞争情况来判断是否对目标应用调整资源数量或调整部署的应用个数,并通过反馈循环来不断优化模型.

总结来说,基于资源分配模型的资源分配方法虽然能解决未知目标函数的极值问题,但需考虑的参数更多,不仅包括多维度资源还包括应用的负载变化、应用间干扰和上下游依赖信息等,同时需要大量的训练数据来训练模型以提升模型预测的准确率.因此,在模型未训练完成时,可以与 4.1~4.3 节所述的资源分配方法配合使用.

4.5 总 结

总结来说,第 3 节所述的干扰检测是混部场景下应用性能保障的第 1 步,最终的保障需要落实在本节讨论的资源分配上.在资源空间寻找最优资源分配方案是单节点性能保障的关键,该问题类似于最优化问题,通过构建最优化算法,根据专家经验、规则或黑盒模型在资源空间中得到满足约束条件的最优解,其代表性工作汇总如表 4 所示.此外,相较于“发生干扰—干扰消除”的事后性能保障方法,若能在应用发生性能干扰之前就未雨绸缪,通过从多个资源类型维度同时入手,利用资源隔离技术在干扰的源

Table 4 Summary of Node Resource Allocation Research Work

表 4 单节点资源分配研究工作总结

研究工作	场景	搜索方法	资源维度						
			CPU	内存	网络带宽	磁盘 I/O	功耗	LLC	内存带宽
Heracles[14]	单 LS+多 BE	梯度下降	√		√		√	√	
PARTIES[9]	多 LS+多 BE	梯度下降	√	√	√	√	√	√	
HCloud[83]	公有云	基于规则	√	√					
PerfIso[69]	单 LS+多 BE	基于规则	√			√			
DRF[95]	多应用	基于规则	√	√					
CoPart[34]	单 LS+多 BE	启发式算法						√	√
Avalon[99]	单 LS+多 BE	启发式算法	√					√	
PRCP[100]	无服务计算链	启发式算法		√					
Whirlpool[96]	多 BE	启发式算法						√	
KPart[98]	多 BE	启发式算法						√	
UCP[97]	多 BE	启发式算法						√	
CLITE[61]	多 LS+多 BE	贝叶斯优化	√					√	√
SATORI[101]	多 BE	贝叶斯优化	√					√	√
DRLPart[102]	多 BE	深度强化学习	√					√	√
Hipster[103]	单 LS+多 BE	启发式+强化学习	√				√		
CuttleSys[104]	单 LS+多 BE	数据挖掘	√					√	
Sinan[71]	微服务链	机器学习	√						
FIRM[60]	无服务计算链	机器学习	√		√	√		√	√

注：“√”表示具备该维度.

头直接保障应用性能，以此寻找最优资源配置能产生更直接的保障效果. 但这对应用性能预测工作提出了更高的要求.

5 集群作业调度

集群作业调度是用于决定将用户提交的应用放置到具体哪个节点上运行，它从宏观的角度出发通过避免将易产生干扰的应用放置在同一节点来最小化性能干扰，从而在保障应用性能的同时提升资源使用效率. 混部场景下的作业调度是一个多目标优化问题，调度结果不仅要满足应用的资源规格和资源数量需求，还需要兼顾集群整体的吞吐率、负载均衡和混部性能干扰影响等. 根据资源类型数量、应用数量、应用资源需求和最小化性能干扰等约束和目标不同，需要用不同的方法来解决.

5.1 基于经验规则

基于经验规则的集群作业调度方法是指根据专家经验、优化目标和约束条件制定规则，调度器按既定规则对新到来的应用进行调度. 该方法要求优化目标和约束条件能量化表示.

对于存在复杂依赖的大型 LS 应用，Rhythm[81] 利用子应用对整体 LS 应用性能的贡献度和子应用的干扰耐受性来指导 BE 应用的调度，在保证 LS 应用不受影响的前提下，混合部署更多的 BE 应用以充分利用闲置资源. 分析发现大型 LS 应用的各个子应用对干扰的耐受性是不同的，因此从应用的整体性能出发，通过计算单个子应用性能的均值、方差和相关系数来量化子应用对整体性能的贡献度. 之后根据贡献度信息对每个 LS 子应用推导出 2 个阈值，分别用于表示该子应用能否与 BE 应用混合部署和能给 BE 应用分配多少资源，以此来指导对 BE 应用的调度，对整体性能贡献度较小的子应用可以混合部署更多的 BE 应用. 另一类工作利用应用的上下游依赖信息来制定作业的调度策略. CARBYNE[106] 利用应用之间的上下游依赖、资源需求和持续时间等信息，在最大化资源分配公平性、最小化应用完成时间和最大化资源利用率的目标下，通过使用最短剩余时间优先（shortest remaining time first, SRTF）的作业调度算法，尽可能将更多的应用调度到集群的剩余资源中，以提升资源利用率.

此外，一些工作通过量化作业调度的约束条件

来设计打分规则,并根据这些规则对备选节点进行打分,最后调度器根据节点的分值大小来进行作业调度决策. 文献 [65] 提出利用物理机的历史资源利用率信息和应用特征来指导 BE 应用的调度方法,根据物理机的 3 种资源利用率模式和 BE 应用的依赖信息设计打分规则,对于每个 BE 应用根据最后分数来选择合适的节点,从而通过充分利用集群中空闲的计算和存储资源来提升资源利用率. 文献 [107] 提出 5 项原则来指导作业调度,根据应用的负载特征和优先级等信息对 5 项原则设置了不同的权重,每个节点的最终分值为 5 项分数的加权和,最终将应用调度至分值最大的节点.

总结来说,基于经验规则的作业调度算法其优化目标和约束条件较为简单,能通过量化来形成经验规则或打分规则,但不适用于优化目标较多或约束条件较为复杂的场景,原因有 2 点: 1)当优化目标增多时,优化问题复杂度上升,在设计打分规则时难以兼顾多个目标; 2)当约束条件较为复杂时,其形式化难度也随之上升,同时针对不同的约束条件,其分值权重的变化会直接影响到作业调度结果,因此,如何选择合适的权重值也成为新的挑战.

5.2 基于性能预测

基于性能预测的集群作业调度算法是指利用机器学习等方法来构建应用的性能预测模型,通过模型来推断应用在不同的约束条件、干扰强度或作业放置方案等场景下的性能,以此来指导调度决策.

一些工作利用推荐算法来推测新提交的、未知应用的性能,根据应用在不同干扰和资源配置下的预测性能来进行调度决策. Paragon[76], Quasar[3], Mage[108] 均利用协同过滤方法推断新提交的应用在干扰或不同资源配置下的性能,并利用应用的分类信息和贪心算法进行作业调度决策,为应用分配最匹配的服务器类型、资源数量和混部应用,以最小化应用间干扰并最大化资源利用率. 区别在于 Paragon 只负责为 BE 应用选择合适配置的物理机节点,它利用协同过滤方法推断应用在不同异构资源下的性能和应用在不同干扰下的性能来选择合适的物理机,以最小化性能干扰. 而 Quasar 在为应用选择合适配置的物理机和最小化性能干扰的同时,还利用协同过滤算法为应用寻找合适的资源分配量. Mage 利用一个 3 阶段的协同过滤模型来对未知的应用进行调度. 阶段 1 通过利用干扰注入法来获得目标应用在多个资源维度上的干扰敏感度信息; 阶段 2 利用阶段 1 的干扰敏感度信息和初始化作业调度方案来获得目标应用的性能; 阶段 3 利用阶段 2 的输出结果来推断所有作业调度方案的性能. 最终在阶段 3 的输出中选择全局性能最优的调度方案.

另一类工作利用应用之间的上下游依赖、执行重叠度或干扰敏感度等信息预测应用混部后的性能,从而向集群中空闲的资源上调度更多的 BE 应用,在保障 LS 应用性能的同时提升作业混部密度,以最大化资源使用率. 为了解决应用之间因灰度干扰导致的性能下降, Gsight[109] 基于请求在不同的执行路径上端到端性能的空间变化性与应用之间在不同程度的执行重叠下其性能的时间变化性等特征,提出了一种干扰感知的作业调度策略,该策略利用函数应用链的端到端调用路径的时空干扰画像构建增量学习模型,通过模型预测应用在不同节点上的性能以辅助作业调度决策. 最终 Gsight 利用二分搜索算法从应用之间完全重叠开始搜索调度方案,以实现将应用调度到尽可能少的节点上. Pythia[110] 通过构建一个线性回归模型来预测 LS 应用与 BE 应用混部后 LS 应用的性能,最后利用最佳适应(best fit)算法从全部可用节点中选择能满足 LS 应用性能约束且资源大小最小的节点,并将 BE 应用调度到该节点运行.

5.3 总 结

总结来说,集群作业调度可视为宏观层面的应用性能保障方法,即从集群的全局视角出发,在为应用选择合适的节点时就将降低性能干扰作为调度约束条件之一,避免将易产生干扰的应用调度到同一节点混合部署,从作业调度角度保障应用的运行时性能,同时通过紧凑的作业放置来最大化集群资源利用率,其代表性工作汇总如表 5 所示. 在调度目标和约束较为简单且可量化的场景下,可利用专家经验直接制定调度规则或打分规则来指导调度; 而当调度目标与约束条件很难用简单方法量化时,可通过构建应用的性能预测模型来预测应用在不同的约束条件或混部场景下的性能,以此来指导作业调度. 集群作业调度方法与上述的干扰消除、单节点资源分配方法是正交的,可以在混部场景中同时采用,从全局视角和单节点的局部视角 2 方面同时保障应用性能并提高资源利用率. 文献 [79] 详尽地介绍了以上 3 种方法在工业界的系统实例.

6 挑战与展望

随着编程模型的快速发展,大型应用往往被划分成一组小的应用,每个应用独立运行,应用之间通

Table 5 Summary of Cluster Job Scheduling Research Work

表 5 集群作业调度研究工作总结

研究工作	调度目标应用	集群资源异构性	方法	集群作业调度目标				
				最小化干扰	资源数量	提升应用运行效率	提升集群资源利用率	资源分配公平性
Rhythm[81]	BE	同构	基于规则				√	
CARBYNE[106]	BE	同构	SRTF			√	√	√
文献 [65]	BE	同构	基于规则				√	
文献 [107]	LS, BE	同构	基于规则	√			√	
Paragon[76]	BE	异构	协同过滤、贪心	√		√	√	
Quasar[3]	LS, BE	异构	协同过滤、贪心	√	√	√	√	
Mage[108]	LS, BE	异构	协同过滤	√		√	√	
Gsight[109]	LS, BE	同构	增量随机森林、二分	√			√	
Pythia[110]	BE	同构	最佳适应算法	√			√	

注: “√”表示具备该目标.

过松耦合的形式相互连接、协调通信并对外提供服务. 这种松耦合的架构促使将复杂应用进行细粒度的分解, 这极大增强了服务的可扩展性和容错性, 同时也便于单个应用模块的升级与维护, 如近几年备受关注的微服务[72]和无服务计算[49]. 对于无服务计算, 大型应用被分解成小应用, 以函数的形式提供简单服务, 多个函数应用之间构成复杂依赖网络对外提供复杂服务. 然而, 随着应用粒度越来越小, 应用的部署密度趋于不断增加, 应用间的依赖关系也变得愈加复杂. 部署密度是指单位资源上(如 CPU 核)运行的应用个数. 在无服务计算的场景下, 1 个 CPU 核上需要支持多个函数应用的运行[111], 这些函数应用共享同一 CPU 核资源. 这种核内共享的混部方式使得资源管理与性能保障变得更为复杂.

本文介绍的混部场景下应用性能保障研究主要聚焦于低密度混部场景, 即 1 个或多个单位资源上只运行 1 个应用. 若将其迁移到无服务计算的高密度混部场景下, 由于在单位资源上混部应用个数不断增加、在空间上应用的依赖关系愈加复杂, 使得在单节点寻找最优资源分配方案、在全局寻找最佳作业调度方案等研究的复杂度也随之增加. 因此, 当前的工作仍然存在 3 个问题和挑战.

1) 更细粒度的资源分配. 与传统应用相比, 函数应用需要更少的资源来满足性能约束. 当前, 无服务计算的服务提供商不仅限制了函数应用代码的大小, 还限制了请求和响应包的大小, 这种拆分成函数粒度的应用使得其具有更细粒度的资源需求. 在 LLC 资源方面, 传统应用至少需要 4~6 MB 缓存资源来保障性能[112], 而函数应用仅需要 1~3 MB 缓存资源就可支持应用在高负载下运行[33], 在低负载时较少的 LLC 的资源即可满足性能约束; 在 CPU 资源方面, 不同于大型应用需要独占多个 CPU 核, 多个函数应用可共享同一 CPU 核资源; 在内存资源方面, 资源分配单位不再是按照 GB 分配, 而是按 MB 进行分配, 在微软的函数集群中, 99% 的函数应用消耗的内存不超过 1 024 MB[67]. 如表 4 所示, 当前只有少数代表性工作支持内存资源的划分, 且已有工作均是以 GB 为单位进行资源分配, 这需要更细粒度的内存分配算法支持. 因此, 在高密度混部下资源分配需要满足 2 点需求: 1)性能隔离, 在软件或硬件层面需要更细粒度的资源隔离机制以保障应用的性能; 2)更合理的资源分配方案. 应用更细粒度的资源需求需要更灵活的资源分配模式, 如何更高效地将多个应用整合到 1 个资源单位中, 哪些应用可以共享资源而哪些应用必须使用隔离资源, 这些都是在高密度混部场景下需要解决的挑战.

2) 更高维度的资源空间. 在高密度共享下, 单位资源上运行的应用数量不断增加, CPU 核内资源共享使得资源的分配粒度更细, 内存资源从以 GB 为单位降低到以 MB 为单位进行分配, 进而导致资源搜索空间以指数级增长. 在更高维度的资源空间下, 上述穷举搜索、梯度下降等最优资源分配搜索方法因时间开销大、易落在局部最优解上等局限性已无法应用于该场景, 需要更高效的资源分配算法. 因此, 在更高维度资源空间下的资源分配算法应满足 2 点要求: 1)时间开销小. 能在高维资源空间和高密度混部下快速找到最优的资源分配方案. 2)质量高. 能找到全局最优解或次优解, 在保障应用性能的同时留

出更多的资源供其他应用使用,以提升资源利用率.

3) 全局视角下性能感知和干扰感知的作业调度. 为了降低应用的运行时干扰,一些工作[113]设计了以CPU核为粒度的作业调度器,将函数应用直接调度到核上,同时避免将多个函数应用调度到同一CPU上运行以实现性能隔离,但是其在进行调度决策时忽略了应用之间的依赖关系. 目前,LS应用间的依赖关系已经愈加复杂,与之相应的请求调用链路长度也在增加. 在阿里巴巴的混部集群中,1个LS应用请求的调用深度平均为4.27,即1个请求平均需要经过4个应用处理后才能向用户返回结果,请求链路最深能达到20[70]. 虽然阿里巴巴在集群各个层面都做了缓存,但不断增加的调用深度加剧了请求性能的不可预测性,调用链中的任何一个应用出现干扰都可能会影响整个请求的响应时间,同时该应用节点也会影响到上下游其他调用链路的性能. 因此,全局视角下的作业调度应满足2点要求:1)全局性能感知. 当单个应用出现性能干扰时能根据其所处的节点位置和上下游依赖关系做到及时预判,对可能影响到的端到端请求和级联效应做到及时调整资源或弹性伸缩. 2)全局干扰感知. 在作业调度时能根据集群中作业整体部署情况和应用的干扰敏感度有针对性地进行调度,在减少混部干扰的情况下做到高质量作业放置.

总结来说,高密度混部的应用性能保障不仅对单节点的干扰检测和资源分配提出了更高的要求,对集群层面的作业调度和宏观资源分配更是提出了新的挑战. 将上述两者相互结合,构建全栈式的软/硬件协同资源保障方法能够在混部集群全面地保障应用性能. 全局作业调度从宏观角度降低应用间的混部干扰,单节点干扰检测和软/硬件资源分配从微观角度为应用的性能保障兜底. 在高密度混部场景下,该方法有助于在满足应用性能约束的情况下提升整体资源使用率,最大化数据中心经济效益.

7 全文总结

混部作为提升数据中心资源使用效率的关键技术已被广泛应用于真实场景中. 然而,由于混部应用对共享资源的争抢,如何解决因资源争用导致的性能干扰已成为混部场景下的关键问题.

本文介绍了面向多应用混部的性能保障方法的研究工作,从特征分析、干扰检测、单节点资源分配以及集群作业调度4个方面讨论了其代表性工作,并对相应方法做出了总结. 其中集群和应用的特征分析是干扰检测和应用性能保障的基础,为应用的资源分配决策提供了依据. 然而,由于集群资源与应用特征的多样性,使得资源分配、作业调度等优化问题的复杂度也随着场景的不同而变化. 从混部场景下优化问题的复杂度来看,可将本文工作思路归纳为2类:基于专家经验和规则的性能保障方法和基于模型的性能保障方法. 对于混部应用个数较少、资源空间较小、约束条件和目标能量化表示等场景,可根据专家经验来制定启发式算法或规则对应用进行资源分配和作业调度. 然而,对于混部应用个数较多、资源空间较大、目标函数未知等场景,由于难以用简单的规则刻画应用在多因素共同影响下的复杂行为和性能变化,需要通过数据挖掘、强化学习等方法构建资源干扰/分配和应用性能变化的模型,以此来指导资源分配和作业调度. 本文最后梳理了随着混部密度不断增加,应用性能保障的挑战与展望,总结性提出了保障混部应用性能发展趋势是采用全栈式的软/硬件协同的方法,该方法有助于在高密度共享下保障应用的性能以及提高数据中心的资源使用效率.

作者贡献声明: 郭静负责搜集、整理混部场景下性能保障相关的研究工作和工作之间的逻辑联系,以及文章整体架构设计、撰写和修改;胡存琛负责干扰检测部分的相关工作整理,以及单节点资源分配相关工作分类的指导,并对全文提出指导意见和修改论文;包云岗负责文章整体思路的指导和逻辑的调整,提出指导意见并修改论文.

参 考 文 献

[1] Dean J, Barroso L A. The tail at scale[J]. Communications of the ACM, 2013, 56(2): 74−80

[2] Barroso L A, Hölzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines[J]. Synthesis Lectures on Computer Architecture, 2009, 4(1): 1−108

[3] Delimitrou C, Kozyrakis C. Quasar: Resource-efficient and QoS-aware cluster management[J]. ACM SIGPLAN Notices, 2014, 49(4): 127−144

[4] Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg [C/OL] //Proc of the 10th European Conf on Computer Systems. New York: ACM, 2015[2023-01-11].https://dl.acm.org/doi/10.1145/2741948.2741964

[5] Guo Jing, Chang Zihao, Wang Sa, et al. Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces [C/OL] //Proc of the 27th IEEE/ACM Int Symp on Quality of

Service. Piscataway, NJ: IEEE, 2019[2023-01-11].https://ieeexplore.ieee.org/document/9068614

[6] Reiss C, Tumanov A, Ganger G R, et al. Heterogeneity and dynamicity of clouds at scale: Google trace analysis [C/OL] //Proc of the 3rd ACM Symp on Cloud Computing. New York: ACM, 2012[2023-01-11].https://dl.acm.org/doi/10.1145/2391229.2391236

[7] Tirmazi M, Barker A, Deng N, et al. Borg: The next generation [C/OL] //Proc of the 15th European Conf on Computer Systems. New York: ACM, 2020[2023-01-11]. doi: 10.1145/3342195.3387517

[8] Nathuji R, Kansal A, Ghaffarkhah A. Q-clouds: Managing performance interference effects for QoS-aware clouds [C] //Proc of the 5th European Conf on Computer Systems. New York: ACM, 2010: 237−250

[9] Chen Shuang, Delimitrou C, Martínez J F. PARTIES: QoS-aware resource partitioning for multiple interactive services [C] //Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 107−120

[10] Zhuravlev S, Blagodurov S, Fedorova A. Addressing shared resource contention in multicore processors via scheduling[J]. ACM SIGPLAN Notices, 2010, 45(3): 129−142

[11] Xu Zhiwei, Li Chundian. Low-entropy cloud computing systems[J]. SCIENTIA SINICA Informationis, 2017, 47: 1149−1163 (in Chinese)
(徐志伟, 李春典. 低熵云计算系统[J]. 中国科学: 信息科学, 2017, 47: 1149−1163)

[12] Chandra D, Guo Fei, Kim S, et al. Predicting inter-thread cache contention on a chip multi-processor architecture [C] // Proc of the 11th Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2005: 340−351

[13] Tang Lingjia, Mars J, Vachharajani N, et al. The impact of memory subsystem resource sharing on datacenter applications [C] // Proc of the 38th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 2011: 283−294

[14] Lo D, Cheng Liqun, Govindaraju R, et al. Heracles: Improving resource efficiency at scale [C] //Proc of the 42nd Annual Int Symp on Computer Architecture. New York: ACM, 2015: 450−462

[15] Linden G. Marissa Mayer at Web 2.0 [EB/OL]. 2006 [2022-03-29]. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

[16] Schurman E, Brutlag J. The user and business impact of server delays, additional bytes, and http chunking in web search [EB/OL]. 2009[2023-01-11]. https://vdocuments.mx/the-user-and-business-impact-of-server-delays-additional-bytes-and-http-chunking.html

[17] Einav Y. Amazon found every 100ms of latency cost them 1% in sales[EB/OL].2019[2022-03-29].https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales

[18] Haque M E, He Yuxiong, Elnikety S, et al. Exploiting heterogeneity for tail latency and energy efficiency [C] //Proc of the 50th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2017: 625−638

[19] Zhang Lufei, Chen Zuoning. Power-efficient formal scheduling policy of VMs in virtualized clusters[J]. Computer Science, 2014, 41(8): 38−41 (in Chinese)
(张鲁飞, 陈左宁. 虚拟集群上面向功耗的形式化的VM调度策略[J]. 计算机科学, 2014, 41(8): 38−41)

[20] Leverich J, Kozyrakis C. Reconciling high server utilization and sub-millisecond quality-of-service [C/OL] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014[2023-01-11].https://dl.acm.org/doi/10.1145/2592798.2592821

[21] Lin Jiang, Lu Qingda, Ding Xiaoning, et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems [C] //Proc of the 14th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2008: 367−378

[22] Sherwood T, Calder B, Emer J. Reducing cache misses using hardware and software page placement [C] //Proc of the 13th Int Conf on Supercomputing. New York: ACM, 1999: 155−164

[23] Ye Ying, West R, Cheng Zhuoqun, et al. Coloris: A dynamic cache partitioning system using page coloring [C] // Proc of the 23rd Int Conf on Parallel Architecture and Compilation Techniques (PACT). Piscataway, NJ: IEEE, 2014: 381−392

[24] Qiu Jiefan, Hua Zonghan, Fan Jing, et al. Evolution of memory partitioning technologies: Case study through page coloring[J]. Journal of Software, 2022, 33(2): 751−769 (in Chinese)
(邱杰凡, 华宗汉, 范菁, 等. 内存体系划分技术的研究与发展[J]. 软件学报, 2022, 33(2): 751−769)

[25] Albonesi D H. Selective cache ways: On-demand cache resource allocation [C] // Proc of the 32nd Annual ACM/IEEE Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 1999: 248−259

[26] Balasubramonian R, Albonesi D, Buyuktosunoglu A, et al. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures [C] //Proc of the 33rd Annual ACM/IEEE Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2000: 245−257

[27] Chiou D, Jain P, Rudolph L, et al. Application-specific memory management for embedded systems using software-controlled caches [C] //Proc of the 37th Annual Design Automation Conf. New York: ACM, 2000: 416−419

[28] Ranganathan P, Adve S, Jouppi N P. Reconfigurable caches and their application to media processing[J]. ACM SIGARCH Computer Architecture News, 2000, 28(2): 214−224

[29] Liu Fang, Jiang Xiaowei, Solihin Y. Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance [C/OL] //Proc of the 16th Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2010[2023-01-11].https://ieeexplore.ieee.org/document/5416655/

[30] Yun H, Yao Gang, Pellizzoni R, et al. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms [C] //Proc of the 19th Real-Time and Embedded Technology and Applications Symp. Piscataway, NJ: IEEE, 2013: 55−64

[31] Iyer R, Zhao Li, Guo Fei, et al. QoS policies and architecture for cache/memory in CMP platforms[J]. ACM SIGMETRICS Performance Evaluation Review, 2007, 35(1): 25−36

[32] Herdrich A, Illikkal R, Iyer R, et al. Rate-based QoS techniques for

cache/memory in CMP platforms [C] //Proc of the 23rd Int Conf on Supercomputing. New York: ACM, 2009: 479−488

[33] Shahrad M, Balkind J, Wentzlaff D. Architectural implications of function-as-a-service computing [C] //Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2019: 1063−1075

[34] Park J, Park S, Baek W. CoPart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers [C/OL] //Proc of the 14th EuroSys Conf. New York: ACM, 2019[2023-01-11].https://dl.acm.org/doi/10.1145/3302424.3303963

[35] Bashir N, Deng Nan, Rzadca K, et al. Take it to the limit: Peak prediction-driven resource overcommitment in datacenters [C] //Proc of the 16th European Conf on Computer Systems. New York: ACM, 2021: 556−573

[36] Lagar-Cavilla A, Ahn J, Souhlal S, et al. Software-defined far memory in warehouse-scale computers [C] //Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 317−330

[37] Waldspurger C A. Memory resource management in VMware ESX server[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 181−194

[38] Gu Juncheng, Lee Y, Zhang Yiwen, et al. Efficient memory disaggregation with infiniswap [C] //Proc of the 14th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2017: 649−667

[39] Liang Shuang, Noronha R, Panda D K. Swapping to remote memory over infiniband: An approach using a high performance network block device [C/OL] //Proc of the 7th IEEE Int Conf on Cluster Computing. Piscataway, NJ: IEEE, 2005[2023-01-11].https://ieeexplore.ieee.org/document/4154093

[40] Brown M A. Traffic control howto [EB/OL]. 2006 [2022-03-29]. http://linux-ip.net/articles/Traffic-Control-HOWTO/

[41] Hong Chiyao, Caesar M, Godfrey P B. Finishing flows quickly with preemptive scheduling[J]. ACM SIGCOMM Computer Communication Review, 2012, 42(4): 127−138

[42] Hu Shuihai, Bai Wei, Chen Kai, et al. Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud[J]. IEEE Transactions on Cloud Computing, 2018, 9(2): 763−776

[43] Grosvenor M P, Schwarzkopf M, Gog I, et al. Queues don't matter when you can JUMP Them! [C/OL] //Proc of the 12th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2015[2023-01-11].https://dl.acm.org/doi/abs/10.5555/2789770.2789771

[44] Perry J, Ousterhout A, Balakrishnan H, et al. Fastpass: A centralized "zero-queue" datacenter network [C] //Proc of the 28th ACM Conf on SIGCOMM. New York: ACM, 2014: 307−318

[45] Nagaraj K, Bharadia D, Mao Hongzi, et al. Numfabric: Fast and flexible bandwidth allocation in datacenters [C] //Proc of the 30th ACM SIGCOMM Conf. New York: ACM, 2016: 188−201

[46] Wang Shuai, Gao Kaihui, Qian Kun, et al. Predictable vFabric on informative data plane [C] //Proc of the 36th ACM SIGCOMM Conf. New York: ACM, 2022: 615−632

[47] Ma Jiuyue, Sui Xiufeng, Sun Ninghui, et al. Supporting differentiated services in computers via programmable architecture for resourcing-on-demand (PARD) [C] //Proc of the 20th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2015: 131−143

[48] Nanda S, Chiueh T. A survey on virtualization technologies [EB/OL]. 2011 [2023-01-11].https://rtcl.eecs.umich.edu/papers/publications/2011/TR179.pdf

[49] Jonas E, Schleier-Smith J, Sreekanti V, et al. Cloud programming simplified: A Berkeley view on serverless computing [J]. arXiv preprint, arXiv: 1902. 03383, 2019

[50] Armbrust M, Fox A, Griffith R, et al. Above the clouds: A Berkeley view of cloud computing, UCB/EECS-2009-28[R]. Berkeley, CA: University of California, 2009

[51] Google Inc. Borg cluster workload traces [EB/OL]. 2019 [2022-03-20].https://github.com/google/cluster-data.

[52] Alibaba Group. Alibaba cluster trace program [EB/OL]. 2021 [2022-03-29].https://github.com/alibaba/clusterdata

[53] Microsoft. Azure public dataset [EB/OL]. 2020 [2022-03-29].https://github.com/Azure/AzurePublicDataset

[54] Chen Tianshi, Guo Qi, Temam O, et al. Statistical performance comparisons of computers[J]. IEEE Transactions on Computers, 2014, 64(5): 1442−1455

[55] Krushevskaja D, Sandler M. Understanding latency variations of black box services [C] //Proc of the 22nd Int Conf on World Wide Web. New York: ACM, 2013: 703−714

[56] Ravindranath L, Padhye J, Mahajan R, et al. Timecard: Controlling user-perceived delays in server-based mobile applications [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 85−100

[57] Ravindranath L, Padhye J, Agarwal S, et al. AppInsight: Mobile App performance monitoring in the wild [C] //Proc of the 10th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2012: 107−120

[58] Amvrosiadis G, Park J W, Ganger G R, et al. On the diversity of cluster workloads and its impact on research results [C] //Proc of the 23rd USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 533−546

[59] Delimitrou C, Kozyrakis C. iBench: Quantifying interference for datacenter applications [C] //Proc of the 9th IEEE Int Symp on Workload Characterization. Piscataway, NJ: IEEE, 2013: 23−33

[60] Qiu Haoran, Banerjee S S, Jha S, et al. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices [C] //Proc of the 14th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 805−825

[61] Patel T, Tiwari D. CLITE: Efficient and QoS-aware co-location of multiple latency-critical jobs for warehouse scale computers [C] // Proc of the 26th IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2020: 193−206

[62] Mars J, Tang Lingjia, Skadron K, et al. Increasing utilization in modern warehouse-scale computers using bubble-up[J]. IEEE

Micro, 2012, 32(3): 88−99

[63] Yang Hailong, Breslow A, Mars J, et al. Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers[J]. ACM SIGARCH Computer Architecture News, 2013, 41(3): 607−618

[64] Chen Quan, Yang Hailong, Guo Minyi, et al. Prophet: Precise QoS prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers [C] //Proc of the 22nd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 17−32

[65] Zhang Yunqi, Prekas G, Fumarola G M, et al. History-based harvesting of spare cycles and storage in large-scale datacenters [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2016: 755−770

[66] Yang Yanan, Zhao Laiping, Li Yiming, et al. INFless: A native serverless system for low-latency, high-throughput inference [C] //Proc of the 27th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2022: 768−781

[67] Shahrad M, Fonseca R, Goiri Í, et al. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider [C] //Proc of the 25th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2020: 205−218

[68] Cortez E, Bonde A, Muzio A, et al. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms [C] //Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 153−167

[69] Iorgulescu C, Azimi R, Kwon Y, et al. PerfIso: Performance isolation for commercial latency-sensitive services [C] //Proc of the 23rd USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 519−532

[70] Luo Shutian, Xu Huanle, Lu Chengzhi, et al. Characterizing microservice dependency and performance: Alibaba trace analysis [C] //Proc of the 12th ACM Symp on Cloud Computing. New York: ACM, 2021: 412−426

[71] Zhang Yanqi, Hua Weizhe, Zhou Zhuangzhuang, et al. Sinan: ML-based and QoS-aware resource management for cloud microservices [C] //Proc of the 26th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2021: 167−181

[72] Gan Yu, Zhang Yanqi, Cheng Dailun, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems [C] //Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 3−18

[73] Gan Yu, Zhang Yanqi, Hu K, et al. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices [C] //Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 19−33

[74] Tian Huangshi, Zheng Yunchuan, Wang Wei. Characterizing and synthesizing task dependencies of data-parallel jobs in Alibaba cloud [C] //Proc of the 10th ACM Symp on Cloud Computing. New York: ACM, 2019: 139−151

[75] Sriraman A, Dhanotia A, Wenisch T F. Softsku: Optimizing server architectures for microservice diversity@ scale [C] //Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 513−526

[76] Delimitrou C, Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters[J]. ACM SIGPLAN Notices, 2013, 48(4): 77−88

[77] Shan Yizhou, Huang Yutong, Chen Yilun, et al. LegoOS: A disseminated, distributed OS for hardware resource disaggregation [C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 69−87

[78] Lu Chengzhi, Ye Kejiang, Xu Guoyao, et al. Imbalance in the cloud: An analysis on Alibaba cluster trace [C] //Proc of the 5th IEEE Int Conf on Big Data. Piscataway, NJ: IEEE, 2017: 2884−2892

[79] Wang Kangjin, Jia Tong, Li Ying. State-of-the-art survey of scheduling and resource management technology for colocation jobs[J]. Journal of Software, 2020, 31(10): 3100−3119 (in Chinese)
(王康瑾, 贾统, 李影. 在离线混部作业调度与资源管理技术研究综述[J]. 软件学报, 2020, 31(10): 3100−3119)

[80] Liu Qixiao, Yu Zhibin. The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from Alibaba trace [C] //Proc of the 9th ACM Symp on Cloud Computing. New York: ACM, 2018: 347−360

[81] Zhao Laiping, Yang Yanan, Zhang Kaixuan, et al. Rhythm: Component-distinguishable workload deployment in datacenters [C/OL] //Proc of the 15th European Conf on Computer Systems. New York: ACM, 2020[2023-01-11].https://dl.acm.org/doi/abs/10.1145/3342195.3387534

[82] Wang Liang, Li Mengyuan, Zhang Yinqian, et al. Peeking behind the curtains of serverless platforms [C] //Proc of the 23rd USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 133−146

[83] Delimitrou C, Kozyrakis C. HCloud: Resource-efficient provisioning in shared cloud systems [C] //Proc of the 21st Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 473−488

[84] Zhang Xiao, Tune E, Hagmann R, et al. CPI2: CPU performance isolation for shared compute clusters [C] //Proc of the 8th ACM European Conf on Computer Systems. New York: ACM, 2013: 379−391

[85] Ousterhout A, Fried J, Behrens J, et al. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads [C] //Proc of the 16th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2019: 361−378

[86] Fried J, Ruan Zhenyuan, Ousterhout A, et al. Caladan: Mitigating interference at microsecond timescales [C] //Proc of the 14th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 281−297

[87] Chen Quan, Xue Shuai, Zhao Shang, et al. Alita: Comprehensive performance isolation through bias resource management for public clouds [C/OL] //Proc of the 33rd Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2020[2023-01-11].https://ieeexplore.ieee.org/document/9355282

[88] Novaković D, Vasić N, Novaković S, et al. DeepDive: Transparently identifying and managing performance interference in virtualized environments [C] //Proc of the 18th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2013: 219−230

[89] Li Yusen, Shan Chuxu, Chen Ruobing, et al. GAugur: Quantifying performance interference of colocated games for improving resource utilization in cloud gaming [C] //Proc of the 28th Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2019: 231−242

[90] Li Zijun, Chen Quan, Xue Shuai, et al. Amoeba: QoS-awareness and reduced resource usage of microservices with serverless computing [C] //Proc of the 34th IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2020: 399−408

[91] Zhao Jiacheng, Cui Huimin, Xue Jingling, et al. Predicting cross-core performance interference on multicore processors with regression analysis[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 27(5): 1443−1456

[92] Wang Sa, Zhu Yanhai, Chen Shanpei, et al. A case for adaptive resource management in Alibaba datacenter using neural networks[J]. Journal of Computer Science and Technology, 2020, 35(1): 209−220

[93] Li Jie, Zhang Jing, Li Weidong, et al. A fair distribution strategy based on shared fair and time-varying resource demand[J]. Journal of Computer Research and Development, 2019, 56(7): 1534−1544 (in Chinese)
(李杰, 张静, 李伟东, 等. 一种基于共享公平和时变资源需求的公平分配策略[J]. 计算机研究与发展, 2019, 56(7): 1534−1544)

[94] Wang Jinhai, Huang Chuanhe, Wang Jing, et al. A heterogeneous cloud computing architecture and multi-resource-joint fairness allocation strategy[J]. Journal of Computer Research and Development, 2015, 52(6): 1288−1302 (in Chinese)
(王金海, 黄传河, 王晶, 等. 异构云计算体系结构及其多资源联合公平分配策略[J]. 计算机研究与发展, 2015, 52(6): 1288−1302)

[95] Ghodsi A, Zaharia M, Hindman B, et al. Dominant resource fairness: Fair allocation of multiple resource types [C] //Proc of the 8th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2011: 323–336

[96] Mukkara A, Beckmann N, Sanchez D. Whirlpool: Improving dynamic cache management with static data classification[J]. ACM SIGARCH Computer Architecture News, 2016, 44(2): 113−127

[97] Qureshi M K, Patt Y N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches [C] //Proc of the 39th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2006: 423−432

[98] El-Sayed N, Mukkara A, Tsai P A, et al. KPart: A hybrid cache partitioning-sharing technique for commodity multicores [C] //Proc of the 24th IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2018: 104−117

[99] Chen Quan, Wang Zhenning, Leng Jingwen, et al. Avalon: Towards QoS awareness and improved utilization through multi-resource management in datacenters [C] //Proc of the 33rd ACM Int Conf on Supercomputing. New York: ACM, 2019: 272−283

[100] Lin Changyuan, Khazaei H. Modeling and optimization of performance and cost of serverless applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(3): 615−632

[101] Roy R B, Patel T, Tiwari D. SATORI: Efficient and fair resource partitioning by sacrificing short-term benefits for long-term gains [C] //Proc of the 48th ACM/IEEE Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2021: 292−305

[102] Chen Ruobing, Wu Jinping, Shi Haosen, et al. DRLPart: A deep reinforcement learning framework for optimally efficient and robust resource partitioning on commodity servers [C] //Proc of the 30th Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2021: 175−188

[103] Nishtala R, Carpenter P, Petrucci V, et al. Hipster: Hybrid task manager for latency-critical cloud workloads [C] //Proc of the 23rd IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2017: 409−420

[104] Kulkarni N, Gonzalez-Pumariega G, Khurana A, et al. CuttleSys: Data-driven resource management for interactive services on reconfigurable multicores [C] //Proc of the 53rd Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2020: 650−664

[105] Zhou Hao, Chen Ming, Lin Qian, et al. Overload control for scaling Wechat microservices [C] //Proc of the 9th ACM Symp on Cloud Computing. New York: ACM, 2018: 149−161

[106] Grandl R, Chowdhury M, Akella A, et al. Altruistic scheduling in multi-resource clusters [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2016: 65−80

[107] Li Qing, Li Yong, Tu Bibo, et al. QoS-guaranteed dynamic resource provision in Internet data centers[J]. Chinese Journal of Computers, 2014, 37(12): 2395−2407 (in Chinese)
(李青, 李勇, 涂碧波, 等. QoS保证的数据中心动态资源供应方法[J]. 计算机学报, 2014, 37(12): 2395−2407)

[108] Romero F, Delimitrou C. Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems [C/OL] //Proc of the 27th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2018[2023-01-11].https://dl.acm.org/doi/10.1145/3243176.3243183

[109] Zhao Laiping, Yang Yanan, Li Yiming, et al. Understanding, predicting and scheduling serverless workloads under partial interference [C/OL] //Proc of the 34th Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2021[2023-01-11].https://ieeexplore.ieee.org/document/9910093

[110] Xu Ran, Mitra S, Rahman J, et al. Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads [C] //Proc of the 19th Int Middleware Conf. New York:

ACM, 2018: 146−160

[111] Agache A, Brooker M, Iordache A, et al. Firecracker: Lightweight virtualization for serverless applications [C] //Proc of the 17th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 419−434

[112] Ferdman M, Adileh A, Kocberber O, et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware[J]. ACM SIGPLAN Notices, 2012, 47(4): 37−48

[113] Kaffes K, Yadwadkar N J, Kozyrakis C. Centralized core-granular scheduling for serverless functions [C] // Proc of the 10th ACM Symp on Cloud Computing. New York: ACM, 2019: 158−164



Guo Jing, born in 1994. PhD. Student member of CCF. Her main research interests include cloud computing and resource management.

郭　静, 1994 年生. 博士. CCF 学生会员. 主要研究方向为云计算、资源管理.



Hu Cunchen, born in 1991. PhD candidate. Student member of CCF. His main research interests include cloud computing and resource management. (duihuhu@gmail.com)

胡存琛, 1991 年生. 博士研究生. CCF 学生会员. 主要研究方向为云计算、资源管理.



Bao Yungang, born in 1980. PhD, professor, PhD supervisor. Member of CCF. His main research interests include computer architecture, operating system, open-source hardware, agile design of processor chips, and datacenter architecture.

包云岗, 1980 年生. 博士, 教授, 博士生导师. CCF 会员. 主要研究方向为计算机系统结构、操作系统、开源硬件、处理器芯片敏捷设计、数据中心体系结构.