软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(10):3100–3119 [doi: 10.13328/j.cnki.jos.006066]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

在离线混部作业调度与资源管理技术研究综述*

王康瑾[1], 贾 统[3], 李 影[1,2]

[1](北京大学 软件与微电子学院,北京 102600)

[2](北京大学 软件工程国家工程研究中心,北京 100871)

[3](北京大学 信息科学技术学院,北京 100871)

通信作者: 李影, E-mail: li.ying@pku.edu.cn

摘 要: 数据中心是重要的信息基础设施,也是企业互联网应用的关键支撑.然而,目前数据中心的服务器资源利用率较低(仅为 10%~20%),导致大量的资源浪费,带来了极大的额外运维成本,成为制约各大企业提升计算效能的关键问题.混部(colocation),即将在线作业与离线作业混合部署,以空闲的在线集群资源满足离线作业的计算需求,作为一种重要的技术手段,混部能够有效提升数据中心资源利用率,成为当今学术界和产业界的研究热点.分析了在线作业与离线作业的特征,探讨了在离线作业间性能干扰等混部所面临的技术挑战,从性能干扰模型、作业调度、资源隔离与资源动态分配等方面就在离线混部技术进行了综述,并以业界典型混部管理系统为例探讨了在离线混部关键技术在产业界的应用及其效果,最后对未来的研究方向进行了展望.

关键词: 数据中心;资源利用率;调度算法;资源管理技术;性能干扰

中图法分类号: TP311

中文引用格式: 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.软件学报,2020,31(10):3100–3119. http://www.jos.org.cn/1000-9825/6066.htm

英文引用格式: Wang KJ, Jia T, Li Y. State-of-the-art survey of scheduling and resource management technology for colocation jobs. Ruan Jian Xue Bao/Journal of Software, 2020,31(10):3100–3119 (in Chinese). http://www.jos.org.cn/1000-9825/6066.htm

State-of-the-art Survey of Scheduling and Resource Management Technology for Colocation Jobs

WANG Kang-Jin[1], JIA Tong[3], LI Ying[1,2]

[1](School of Software and Microelectronics, Peking University, Beijing 102600, China)

[2](National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China)

[3](School of Electronics and Computer Science, Peking University, Beijing 100871, China)

Abstract: Data center is not only an important IT infrastructure, but also a key support for enterprise Internet application. However, the resource utilization of data center is pretty low (only 10%~20%), which leads to a large amount of waste of resources, brings a huge extra operation and maintenance cost, and becomes a key problem restricting enterprises to improve the computing efficiency. By colocating online services and offline tasks, colocation can effectively improve the resource utilization rate of data center, which has become a research hotspot in academia and industry. This paper analyzes the characteristics of online services and offline tasks, and discusses the technical challenges faced by the performance interference between services and jobs. This paper summarizes the key technologies from the aspects of performance interference model, job scheduling, resource isolation and dynamic resource allocation, and discusses the application and effect of colocation systems in the industry with four typical colocation system. At the end of this paper, the future research direction is presented.

Key words: Internet datacenter; resource utilization; job scheduling; resource management technology; performance interference

* 基金项目: 广东省重点领域研发计划(2020B010164003)

Foundation item: Key-area Research and Development Program of Guangdong Province, China (2020B010164003)

本文由“系统软件前沿进展”专题特约编辑武延军研究员、陈海波教授、包云岗研究员、李玲研究员推荐.

收稿时间: 2020-02-10; 修改时间: 2020-04-04; 采用时间: 2020-05-09; jos 在线出版时间: 2020-06-10

大规模数据中心是当今企业级互联网应用和云计算系统的关键支撑.为保障日益增长的互联网应用和云计算系统的计算需求,数据中心需要不断扩容,其规模和服务器总量呈现快速增长趋势.权威报告[1]指出,2020 年全球数据中心的服务器总量将达到 1 800 万台,并且正以每年 100 万台的速度增长.然而,伴随着数据中心的急速扩容,其资源利用率却始终处于较低状态.统计数据表明[1],目前全球数据中心资源利用率仅为 10%~20%,如此低的资源利用率意味着数据中心大量的资源浪费,导致目前数据中心的成本效率极低.因此,如何提升数据中心资源利用率成为一个亟需解决的关键性技术问题.

通常而言,数据中心以作业为抽象单位执行计算任务.按照计算任务的不同可将作业分为在线作业和离线作业.在线作业通常是以服务形态来处理用户请求并执行计算任务,如网页搜索服务、在线游戏服务、电商交易服务等,具备较高的实时性和稳定性需求.为保障上述需求,企业往往会对外提供明确的服务等级协议(service level agreement,简称 SLA),以明确的条款约定服务质量.违反 SLA 会给企业带来经济损失,因此数据中心往往为在线作业预留大量的服务器资源以保证其服务质量.离线作业通常是计算密集型的批处理作业,如 MapReduce 和 Spark 作业[2,3]数据分析作业/机器学习模型训练作业等.多数离线作业对于性能没有严格的要求,可以容忍较高的运行延迟并支持失败任务的重启.按照集群上所运行的作业类型,数据中心可划分为在线集群和离线集群.在线集群通常因在线作业对性能的高要求而采用过量资源供给策略,导致资源利用率较低;与之相反,随着大数据[4−7]和人工智能的发展,离线作业的规模日益扩大,离线集群对计算资源的需求日益增长,导致现有集群资源供给不足[8].因此,提升数据中心整体资源利用率的理想方法是在保障在线作业性能的前提下,将在线集群的空闲资源分配给离线作业,即:将在线作业与离线作业混合部署于同一集群,简称“混部(colocation)”.目前,在离线混部已成为数据中心提升整体资源利用率的主流方法[9].

在混部作业运行过程中,由于在离线作业竞争共享资源(如 CPU、缓存、内存带宽、网络带宽等),往往会导致作业性能下降(performance degradation),该现象也被称为性能干扰(performance interference)[10−15].性能干扰会严重影响在线作业的实时性和稳定性,降低离线作业的吞吐率,因此,混部集群管理系统必须有效控制性能干扰,并根据性能干扰快速、及时地做出调整.然而,性能干扰的程度受诸多因素影响,尤其是在超大规模的数据中心中,作业负载的动态性、资源需求的多样性、硬件架构的异构性等导致性能干扰复杂性剧增.为降低复杂的性能干扰,混部集群管理系统必须能够协调集群层的作业调度和节点层的资源管理,实现性能干扰的分层控制.集群层的作业调度,即决策作业运行的具体位置(具体某个服务器).混部作业调度是一个复杂的多目标优化问题,既需要兼顾集群的负载均衡、整体吞吐率、资源公平性等目标,又需要特别考虑性能干扰的影响,根据性能干扰的变化随时调整调度策略;节点层的资源管理,即利用资源隔离技术根据性能干扰变化动态调整运行作业的资源供给.因此,集群层的作业调度和节点层的资源管理成为降低和控制性能干扰、提升资源利用率的重要手段.

本文首先分析了在离线作业的特征,深入探讨了在离线作业在混合部署中竞争共享资源带来的性能干扰及作业调度和资源管理问题,就作业性能干扰模型、集群层面的混部作业调度以及节点层面的资源管理等关键技术进行了详细介绍和分析,并结合 4 个系统实例探讨了在离线混部关键技术在产业界的实现及其效果,最后就未来的研究方向进行了讨论和展望.

1 概 述

本节分析了在线作业和离线作业的各自特点,并探讨了在离线混部集群管理上所面临的主要问题与挑战.

1.1 在线作业与离线作业

在线作业.在线作业通常是处理用户请求的服务,典型的在线作业有网页搜索、即时通信、语音识别、流式计算、电子商务、流媒体等,通常可为企业带来直接的经济利益.在线作业具有如下特点.

(1) 运行时间长.在线作业通常以服务的形态持续运行,以请求(request)为单位触发计算任务.例如,网页搜索服务、数据库服务、社交网络服务等[16,17],运行往往持续数周甚至数月,因此在线作业也被称为长服务(long runing service)[18].

(2) 资源使用呈现动态变化.在线作业的资源使用量与用户并发请求量呈正相关,会伴随用户并发请求量发生动态变化.图 1 所示为国内某互联网公司的在线作业计算节点一个月内的 CPU 使用量变化曲线,随着时间的推进,该作业的 CPU 使用量呈现了明显的以天为周期的动态变化特征.此外,图 1 中还出现了多次 CPU 利用率突变的现象,这是因为,某些社会事件,如突发热点新闻、电商网站的促销活动等会导致用户并发请求突增[19].

(3) 对性能变化敏感.在线作业的性能通常决定了企业对外服务质量,而服务质量则直接影响企业的经济利益和用户体验.例如,Amazon 的统计数据显示,Amazon 的网页响应速度慢 0.1s 便会导致 1%的用户放弃交易[20];也有研究表明,交互式应用的响应时间只有在 100ms 以内时才能给用户流畅的使用体验[21].因此,在线作业又被称为延时敏感型作业(LC(latency critical) Job 或 LS(latency sensitive) Job)[13,15].

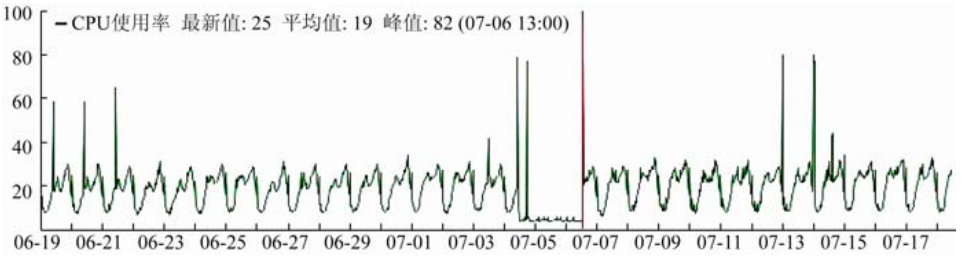

Fig.1 Static software defect prediction research framework using defect-proneness as prediction target

图 1 某在线作业所在服务器 CPU 使用率波动曲线,采样时间为 1 个月

离线作业.离线作业是指数据中心中优先级较低的、对性能要求不高的批处理作业,如 MapReduce 作业和机器学习训练作业等,离线作业具有如下特点.

(1) 运行时长较短.对 Google 和阿里巴巴数据中心中的离线作业运行时长统计后发现[22−25],离线作业运行时长大多位于数分钟到数小时这一区间.因此,与运行持续数周甚至数月的在线作业相比,离线作业的运行时长较短.

(2) 计算密集.离线作业通常需要进行大量的计算操作,数据统计显示,阿里 MapReduce 作业的 CPU:Mem 约为 1:4,即每使用 4G 内存便会消耗 1 个 CPU 的全部计算能力;深度学习模型训练作业也是典型的离线作业,此类作业通常需要进行大量的梯度计算,并且通过多次迭代直至模型收敛.

(3) 对性能变化不敏感.离线作业通常为批处理作业,对于性能没有严格的要求,可以容忍较大的运行延迟并支持任务的失败重启.

Table 1 Comparisons of LC jobs and BE jobs

表 1 在线作业和离线作业对比

	在线作业	离线作业
运行时长	数周甚至数月	数分钟至数小时
对性能变化的敏感程度	高	低
集群资源利用率	低	高

由于在线作业运行时的低延迟要求,为保证在线作业的性能,一般采用了在线作业独占集群的方式,避免与其他作业共享计算资源,并且为在线作业分配过量的计算资源以应对在线作业动态的资源需求.图 1 所示在线作业的峰值 CPU 需求可达 80%,为满足其在峰值负载时的计算资源需求,采用了一个在线作业实例独占一台服务器的部署方式.这种部署方式导致在线作业集群平均资源利用率较低(服务器月平均 CPU 利用率仅为 20%).与此同时,离线作业的规模日益扩大,对离线集群计算资源的需求呈现快速增长,导致离线集群资源不足[8].因此,混部成为提升数据中心资源利用率的理想方法[26],即在保障在线作业性能的前提下,将在线集群的空闲资源

分配给离线作业.混部既可以满足离线作业的资源需求,也可以提升在线集群的资源利用率,因此成为提升数据中心整体资源利用率的主流方法.

1.2 问题与挑战

由于多个作业竞争共享资源(如 CPU、缓存、内存、内存带宽、网络带宽),导致作业间出现性能干扰[13,14].这种在离线混部作业间的资源竞争及性能干扰使得混部集群作业调度与资源管理变得十分复杂.围绕如何减少和控制在离线混部作业间的性能干扰,同时提升混部集群的资源利用率,本节总结和分析了在离线混部集群管理面临的主要问题与技术挑战.

1.2.1 在离线混部作业性能干扰问题

在离线混部作业的性能受到诸多因素的影响,呈现出性能对干扰的敏感性具有动态变化、模式多样复杂等特点,尤其是在超大规模混部集群中动态变化的作业负载、多样化的资源依赖以及异构硬件架构等使得在离线混部作业的性能难以建模,在各种干扰模式下的性能预测变得极具挑战性,也增加了集群作业调度和节点资源管理的难度.

首先,如前所述,在线作业的工作负载具有动态性,同一作业的性能在不同工作负载下对于干扰的敏感性并不相同,由于高工作负载时作业所需的计算资源高于其低负载时的需求,因而高负载下的在线作业对资源有更强的竞争,也对干扰更加敏感;其次,运行在数据中心的在离线作业数量庞大且种类繁多,对关键资源的依赖性因其运行逻辑不同而各异,若混合部署于同一节点上的在线作业与离线作业具有同样或类似的关键资源依赖性,则会加剧其相互间的性能干扰;再次,节点上作业的细粒度资源共享主要依靠操作系统的资源调度机制,如分时共享或抢占式调度等,不同的资源调度机制会使得作业在运行时具有不同的抗干扰性,例如抢占式调度可减少作业在就绪进程等待队列中的排队时间,相比于分时共享的公平调度机制,抢占式调度的资源共享因其能使作业具有更好的抗干扰性而更适合于在线作业;最后,随着计算机体系结构的发展,硬件架构出现了单核、多核、异构多核等多种架构,提供了多样化的硬件级别的抗干扰机制,例如多核体系结构通过增加 CPU 的数量来减少作业间对于 CPU 的竞争,支持 Intel CAT(cache allocation technology)的 CPU 可为不同的进程划分私有的 L3 缓存空间,从而降低作业因竞争 L3 缓存资源而引起的性能干扰.这种异构硬件架构所带来的抗干扰机制的多样化更进一步增加了在离线混部作业性能预测的难度.

1.2.2 在离线混部作业调度问题

作为数据中心管理的关键技术,作业调度一直是学术界和产业界研究的热点领域,传统作业调度研究工作侧重于资源公平性[27,28]、负载均衡、提高吞吐率[29]或资源利用率等多目标.对于在离线混部作业调度而言,除了满足这些目标以外,在线作业与离线作业两种类型作业的特点及其混合部署所带来的必然的资源竞争和性能干扰使得混部集群作业调度问题更具有挑战性.

首先,如前所述,在离线混部作业的性能受到诸多因素的影响,呈现出性能敏感性动态变化、干扰模式多样复杂等特点,在动态负载和复杂干扰模式下的作业性能预测模型难以构建;其次,在离线作业调度机制需要充分考虑到在线作业与离线作业两种类型作业的运行特点和性能要求,对于离线作业而言,因其运行时间短,并行度高,需要较高的调度吞吐率和较低的调度延时;相对而言,在线作业则可接受一定程度的调度延时;最后,数据中心中在线作业与离线作业的数量庞大且种类繁多,其混合部署的组合数量将呈现指数爆炸状态,如何在巨大的解空间中快速搜索到满足多目标的在线作业与离线作业混部组合,进而进行作业调度,是一个难题.

1.2.3 在离线作业的共享资源隔离问题

导致性能干扰的一个重要原因是具有同样或类似关键资源依赖性的在线作业与离线作业对共享资源的无序竞争.资源隔离技术通过软件或软硬件协同技术控制作业对资源使用的方式来降低甚至消除对资源的无序竞争,例如 Linux CGroup 支持 CPU、内存、磁盘等资源的隔离,NUMA(non uniform memory access architecture)架构采用多内存通道技术减少内存带宽的竞争,Intel CAT 的 Cache 隔离机制通过隔离不同作业所使用的缓存空间缓解作业在竞争 L3 Cache 时发生的缓存相互替换.然而,在离线混部作业的资源隔离技术仍是一个难题.

首先,现有体系结构中的资源类型繁多,诸如现有的通用硬件在 TLB 快表、L1/2 缓存、缓存带宽、内存带

宽、总线带宽等诸多关键资源仍然缺乏对应的软硬件协同的资源隔离机制,尚无法实现应用级隔离;其次,在目前高度复杂的硬件上通过软件方法精准控制作业对于硬件资源的竞争需要同时考虑操作系统和硬件架构的具体实现,而多数商用硬件具有黑盒性,极大地增加了使用软件方法隔离硬件资源的难度,因此具有挑战性.

1.2.4 资源动态分配问题

在离线作业的资源需求随作业负载而发生动态变化[30].为保障在线作业的响应时间并提升集群资源利用率,需根据负载的变化动态分配相应的资源,优化资源配比.然而,一方面,在离线混部作业间存在的不可知且复杂的干扰会严重影响作业的性能;另一方面,作业资源配比的变化也会影响性能干扰的程度,进而产生作业的性能变化.因此,如何在资源动态分配过程中平衡资源利用率和作业性能也是一个难题,也称为近年来的研究热点[19].

首先,资源重分配带来的性能干扰变化难以建模和量化,作业性能难以预测和保障.如前所述,动态变化的作业负载、多样化的资源依赖以及异构硬件架构等均会影响作业性能,使得在各种干扰条件下的性能预测变得极具挑战性.资源重分配改变了作业的关键资源瓶颈,刷新了资源依赖关系,重置了性能对干扰的敏感度,带来了一系列不可控的连锁反应,更加剧了性能干扰的变化和不确定性.其次,资源类型多样复杂,资源间存在互补、互替代、单向依赖、多向依赖等复杂关系,使得资源分配策略搜索空间极大,限制条件极为复杂.最后,在线作业的实时负载波动和高性能需求要求资源动态分配具备精准和快速的特性.然而,由于复杂性能干扰导致作业性能难以预测,依靠有限的负载变化、混部作业组合变化等信息快速做出精准的资源分配决策极为困难.

2 相关研究工作

2.1 研究框架

针对上述问题和挑战,本文从在离线混部作业性能干扰模型、集群层面的作业调度以及节点层面的资源管理这 3 个方面进行研究,如图 2 所示.

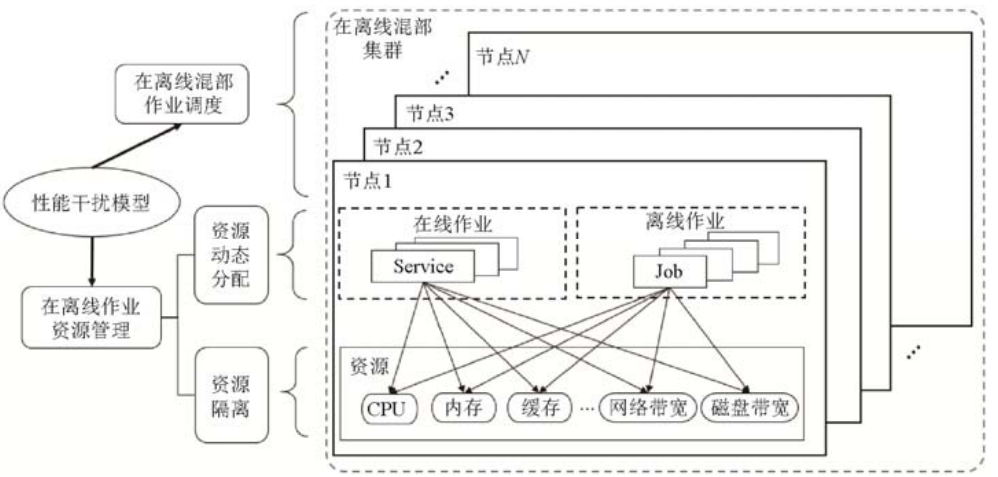

Fig.2 Research framework

图 2 研究框架

性能干扰模型.为了解决在离线混部作业的性能干扰问题,须就作业的运行环境对性能的影响进行建模,以预测作业在动态负载、资源竞争及干扰模式等条件下的性能,为集群作业调度和节点资源管理提供指导.

集群层混部作业调度.混部作业调度所研究的内容是作业的放置问题,即决定作业的运行位置.混部作业调度是一个多目标优化问题,在传统作业调度的优化目标上,还需要优化在离线作业间的性能干扰,同时满足在线作业的 SLA,兼顾作业吞吐率等要求.

节点层混部作业资源管理.混部作业资源管理须:(1) 为作业的运行控制资源供给,在满足作业资源需求的同时减少作业间的性能干扰;(2) 根据作业运行状况预测其资源需求并做出分配策略.前者涉及到操作系统和硬件架构的资源隔离技术,后者则涉及资源动态分配算法.

2.2 性能干扰模型

性能干扰模型就在离线混部作业的运行环境对作业性能的影响进行建模,以预测在动态负载、资源竞争及干扰模式等条件下的作业性能,为集群层面混部作业调度和节点层面混部资源管理提供指导.性能干扰模型的形式化描述由式(1)给出,式(1)代表作业的性能模型,性能模型的输入包含 4 部分:作业当前的负载 *load*、作业当前所面临的来自其他作业的资源竞争压力 *pressure*、作业当前的资源供给 *res*、作业运行的硬件架构 *hw*,输出则为作业 *j* 在当前运行环境下所能达到的性能 *pf*.

$$pf=Pref^{j}(load,pressure,res,hw) \tag{1}$$

性能干扰模型的输出代表了作业的性能,但在线离线作业通常具有不同的性能指标,例如,在线作业的性能评价指标通常是请求的响应时间(response time,简称 RT),离线作业的性能评价通常是作业的运行时间或作业吞吐率等指标.上述指标与作业的具体逻辑相关,可以直接反映作业真实的性能状况,因而被多个研究工作采用,如文献[13,31−34].但是获取作业级别的性能指标需要与具体应用对接,具有较差的可迁移性.并且,在公有云环境或大规模数据中心中,由于作业的黑盒性或隐私条款,获取作业级性能指标并不可行,因此不少性能干扰模型采用了系统底层的性能指标作为作业的性能指标,如描述作业在每个 CPU 时钟周期可执行的指令数量 IPC(instructions per cycle)或者 CPI(cycles per instruction)、MIPS(million instructions per second).作业间的性能干扰可引起上述指标的变化,例如一个在线作业的缓存空间被离线作业替换掉后,后续的指令在执行过程中发生了缓存缺失(cache miss),导致执行访存指令所需的处理器周期数增加,导致 IPC 降低.底层性能指标由于具有通用性强、方便获取等优点,被广泛用于性能干扰模型中,如文献[12,35−37].

性能干扰模型的第 1 个参数 *load* 表示作业当前的负载,对于在线作业的负载表现为当前请求并发数,如 RPS(requests per second)和 QPS(queries per second),而离线作业的负载则与作业的输入数据规模相关.性能干扰模型的第 2 个参数 *pressure* 代表作业当前面临的资源竞争压力,即作业所遭受的性能干扰,目前关于资源竞争压力并没有统一的量化标准.文献[14,35]采用了一个对性能敏感的程序(称为 reporter 或 ruler)的运行时间并将其运行时间作为 *pressure*.文献[32,33]则使用当前的内存带宽使用量作为 *pressure*.文献[38]则引入了服务内部的队列长度作为 *pressure*.第 3 个参数 *res* 代表当前分配给作业的可用资源,由一个向量表示,可包括 CPU、内存、Cache 在内的多种共享资源.不同的性能干扰模型由于面向的场景不同,在参数的取值范围上有所不同,如面向异构集群的性能干扰模型需要在 *hw* 参数上具有多种取值,如文献[39,40],而面向同构平台的性能模型的 *hw* 参数则为定值;面向恒定作业负载场景的性能干扰模型的 *load* 为定值.

建立性能干扰模型的过程可分为两步:(1) 数据获取;(2) 模型构建.建立性能干扰模型的第 1 步是获取性能干扰数据,即获取数据集 $D=\{d_1,d_2,...,d_n\}$,$d=\{load,pressure,res,hw\}$.在数据获取方法上,存在基于干扰注入的数据获取方法和基于历史监控数据的数据获取方法两种.

基于干扰注入的数据获取方法的基本思想是在作业运行时向作业运行环境中注入性能干扰,通过不断改变干扰的强度从而获得该作业在不同干扰下的性能变化.注入干扰的方法是在作业运行环境中运行一个干扰者,干扰者通常是精心设计的一个资源密集型程序,可与作业竞争在多种共享资源上产生资源竞争.例如:作为作业运行中最主要的资源——CPU 时间片,其上的干扰可以使用大量的循环语句来产生;内存子系统是竞争最激烈的资源之一,由于内存子系统具有层次化的特点,对于不同层次的资源竞争可以通过读写不同大小的数组来产生;文献[41]总结和列举了 15 种干扰源并提出了可在多种维度上产生性能干扰的工具 iBench.使用 iBench 作为干扰注入工具的研究工作有文献[39,40].文献[14,32,33,35]专注于内存子系统上的干扰,使用了可产生内存带宽干扰的干扰者用于获取内存带宽干扰和作业性能的变化关系.文献[42]研究了超线程之间由于共享计算单元而引起的性能下降,并在性能模型中考虑了来自相邻超线程的干扰.基于干扰注入的数据获取方式理论上可以获取作业在多种性能干扰下的性能数据,但其局限性在于:(1) 造成干扰的资源类型多,采样空间大,因此获取

较为全面的性能干扰数据所需采样时间长;(2) 由于操作系统和高级程序设计语言屏蔽了底层的硬件细节,实现能够在特定资源上产生特定强度干扰的干扰者需要设计人员对体系结构和高级程序设计语言具有深入的了解,因此设计和实现有效的干扰者具有一定的难度.

基于历史监控数据的数据获取方法从作业运行产生的监控数据中提取出性能干扰模型所需要的数据集.在数据中心的运行过程中,积累了大量关于作业运行的监控数据,由于数据中心作业的动态性,作业在长期运行过程中会遭受不同程度的性能干扰,因此历史监控数据中蕴含着可用于构造性能干扰模型的数据.基于历史监控数据的数据方法具有如下优点:(1) 无需运行作业,避免了额外的资源消耗和时间开销;(2) 数据易获取;(3) 数据量大.但是该方法同样存在着如下局限性:(1) 灵活性差,只能获取历史监控数据中监控的资源,如需新增监控维度则需要重新积累历史数据;(2) 无法获取从未运行过的新作业;(3) 适用范围有限,仅适用于作业运行环境动态性高、资源竞争程度变化范围大的集群,而对于作业运行环境稳定的集群,如在线独占集群的部署策略,则由于作业运行环境缺乏足够的性能干扰而无法获得较为全面的性能干扰数据.

Table 2 Comparisons of performance interference models

表 2 性能干扰模型对比

名称	数据获取方法	性能指标	模型构建方法	*pressure* 所需数据
BubbleUp[14]	基于干扰注入	RT	基于统计方法	内存带宽
BubbleFlux[35]	基于干扰注入	IPC	基于统计方法	内存带宽,作业负载
SMiTe[42]	基于干扰注入	IPC	基于统计方法	Ruler 测量的干扰强度, 被测试应用在特定干扰下的 IPC
DeepDive[36]	基于历史监控数据	IPC	基于机器学习方法	CPU Halt,L1/L2/L3 Cache,BranchMiss
CPI^2[12]	基于历史监控数据	CPI	基于统计方法	CPU 使用量,CPI
FECBench[31]	基于干扰注入	RT	基于机器学习方法	CPI,L2 BW,L3 BW 等 12 种资源
文献[32,33]	基于干扰注入	作业运行时间	基于机器学习方法	L2_LinesIn rate,内存带宽
Spread-*n*-share[43]	基于干扰注入	IPC	基于统计方法	IPC,共享缓存,内存带宽
Mage[39]	基于干扰注入	MIPS	基于机器学习方法	内存,MBW,L1/L2/L3,TLB,网络带宽,磁盘带宽
Prophet[34]	基于干扰注入	作业运行时间	基于统计方法	PCIe 带宽,GPU 内存带宽
Kambadur[37]	基于历史监控数据	IPC	基于统计方法	CPU-cycles,CacheMiss,CPI
Seer[38]	基于历史监控数据	QoS 违反概率	基于机器学习方法	服务请求队列长度,分布式追踪数据
Paragon[40]	基于干扰注入	IPC	基于机器学习方法	内存,MBW,L1/L2/L3, TLB,网络带宽,磁盘带宽,节点性能

获取到性能干扰数据后,进入性能干扰模型构建阶段.目前性能干扰模型的构建方法主要包括基于统计方法和基于机器学习两种.基于统计的方法通过数理统计求得模型中各变量的变化关系,如文献[14,35]建立了描述资源竞争压力(使用 reporter 测量得到)和作业性能的映射关系,称为“敏感函数(sensitive function)”;文献[43]使用统计模型建立了内存带宽和作业性能的变化关系;文献[37]则通过构建作业和作业之间的混部运行性能的二维表,通过查表的方式即可实现性能干扰的预测.基于统计方法的优点是:(1) 计算量小,模型构建和更新速度快;(2) 可解释性强,便于优化.但其局限性在于:(1) 模型的设计需要先验知识,构造和优化具有一定的难度;(2) 当数据维度增多时传统的统计方法无法适用.随着机器学习方法的发展,研究人员尝试将机器学习模型应用于性能干扰模型,采用了诸如随机森林回归模型[31]、协同过滤算法[39,40]、线性回归[32,33]、聚类算法[12,36]、深度神经网络[38,44]等方法完成从作业运行时信息到作业性能的映射,基于机器学习算法建立的性能干扰模型可以学习到人类无法察觉的特征,并且具有良好的预测结果.缺点是:(1) 需要高质量的训练数据;(2) 缺乏可解释性,难以优化;(3) 在建立模型时需要较长的训练时间和计算开销.

2.3 在离线混部作业调度

作业调度是数据中心和云计算领域的研究热点,在离线混部作业向传统的作业调度提出了新的挑战.本节围绕第 1.2.2 节中总结的 3 点挑战,讨论近年来在离线混部作业调度领域的相关研究工作.

与传统作业调度相比,在离线混部作业调度需要考虑如何放置待调度作业使得作业间的性能干扰最小,尤其是对在线作业的性能干扰最小.这就需要调度器从集群的所有节点中筛选出待调度作业对其上运行的在线

作业性能干扰最小的节点.本文总结了筛选节点的两种常用方法.

(1) 基于性能模型筛选节点.该方法首先对在线作业建立性能模型,进而使用性能干扰模型来预测混部后作业所能达到的性能指标,通过搜索的方法寻找产生干扰较小的节点.例如:文献[14,34]在使用性能干扰模型预测不同作业混部运行时的性能,以预测数据为依据判断该混部作业组合是否“安全”(“安全”的混部组合是指混部组合中的每个作业都能达到其要求的性能指标);文献[28,39,40,45]使用协同过滤模型预测一个作业与其他作业混部时所能达到的性能,进而使用贪心算法搜索作业的最佳运行位置.使用预训练的性能干扰模型作为调度的依据取得了较好的效果,例如文献[14]在保证在线作业QoS(quality of service)的情况下提升了 50%~90%的集群整体资源利用率;文献[39]可以保持 52%的作业的 QoS 不被影响,而另外 33%的作业的 QoS 波动低于 10%.虽然可以有效地避免作业之间的性能干扰,但是该方法依赖了预构建的性能模型,而获取性能干扰数据和构建性能模型所带来的开销是不可忽略的问题;

(2) 基于空闲资源筛选节点.由于干扰和资源的相关性,空闲资源多的节点意味着该节点目前负载较少,因而有可能承受更多的资源竞争.文献[46,47]通过预测未来一段时间集群的空闲资源数据,进而用于作业调度;文献[48]提出一种作业偷取机制,当在线作业集群位于较低负载时可从待运行离线作业队列中调度一部分作业至在线集群运行;文献[49]在调度时首先对作业的 CPU 和 I/O 资源需求进行聚类,然后在作业间进行资源的匹配,进而完成调度.根据作业资源调度简单实用,因此适合在大规模混部系统中使用[9,50].本文在第 2.2 节中指出,作业间的性能干扰还与作业本身资源的使用特征相关,因此仅考虑将作业调度至空闲资源多的节点并不能完全保证该节点上在线作业的 QoS,因此仍存在一定的风险.

在离线混部调度不仅要减少作业间的性能干扰,还要满足传统作业调度中存在的资源公平性、作业资源需求、集群负载均衡、集群吞吐率等多个优化目标,为解决在离线混部作业调度面临的多目标优化问题,相关工作中使用了 3 类算法.

(1) 启发式算法.基于专家直观经验构造算法用于寻找最优调度结果,启发式算法由于其简单直观,易于理解和修改,被多种在离线混部作业调度算法所采用,如文献[34,39,40,48,49];文献[14]通过性能干扰模型预测作业在每个机器上运行时产生的干扰,然后寻找干扰最小的节点作为作业的部署节点.文献[52]将博弈论应用于在离线混部作业调度,使用 Shapley 算法寻找作业和节点之间稳定的婚姻匹配组合(如果 A 喜欢 b 的程度比喜欢自己的配偶更高,并且 b 同样对 A 的喜好程度比自己的配偶更高,则他们很可能组成新的家庭,这样的组合是不稳定婚姻,不存在不稳定婚姻的组合被称为稳定婚姻组合),在保证作业性能的同时保证资源分配的公平性.启发式算法的局限性在于:(a) 当需要优化的目标较多时,调度问题的复杂度上升,设计兼顾多个目标的启发式算法具有一定的难度;(b) 启发式算法难以保证算法给出的解是全局最优解;

(2) 打分规则,当调度优化目标较多时,对每个目标设定量化规则,并使用这些规则对每个节点进行打分,将最终的加权分数作为节点最终得分,最后将作业调度至得分最高的节点.常见的打分规则有:根据负载打分,负载最低时节点得分为 10,满负载时得分为 0 等.节点的最终得分是所有规则的得分加权和,调度算法选择得分最高的节点作为调度决策,基于打分规则的调度算法有文献[47,52].该方法的优点是易于扩展(如果有新的优化目标只需添加新的打分规则即可),计算开销低,但其局限性在于:(a) 每个规则的权重作为调度算法的超参数,对调度结果影响较大,寻找最佳的超参数设置需要经过不断的调试;(b) 基于打分规则的调度算法同样难以保证算法给出的解是全局最优解;

(3) 整数线性规划.作业调度问题可以抽象为组合优化问题,而整数线性规划问题(integer linear program,简称 ILP)作为组合优化的一种解决方法,可用于解决作业调度问题.文献[18]通过对减少亲和性违反、最大化成功调度的作业数量、最小化资源碎片、集群负载均衡、最大化资源利用率等多种目标进行量化后,并使用加权的方式将多个优化目标统一在一个全局目标函数中,使用 CPLEX 优化器求解.基

于整数线性规划的调度算法的局限性在于:(a) 算法开销较大,文献[18]通过实验证明,在同等规模下,整数线性规划算法的求解时间开销均高于启发式算法和基于打分规则的调度算法;(b) 目标函数的构造对调度质量至关重要,对多个优化目标进行量化表示具有一定的难度.

混部作业调度算法对比见表 3.

Table 3 Comparisons of hybrid job scheduling algorithms

表 3 混部作业调度算法对比

名称	调度器架构	集群异构性	资源预测	性能干扰预测	算法类型	优化目标
Paragon[40]	集中式	异构	×	✓	机器学习+启发式算法	减少干扰,提升作业运行效率
Prophet[34]	集中式	同构	✓	✓	启发式算法	减少 AI 作业间相互干扰,提升 GPU 利用率
Mage[39]	层次化	异构	×	✓	机器学习+启发式算法	减少干扰,提升作业运行效率
Hawk[48]	集中式(在线作业)+分布式(离线作业)	同构	×	×	启发式算法	提升利用率,提升作业运行效率和稳定性
HySARC[49]	集中式	异构	×	×	启发式算法	提升资源利用效率
Spread-*n*-share[43]	集中式	同构	×	✓	启发式算法	提升集群的整体利用率,提升作业性能
文献[47]	集中式	同构	✓	×	打分规则	提升利用率,同时保证作业性能
Bubble-up[14]	集中式	同构	×	✓	启发式算法	最小化作业间的干扰,提升利用率
Medea[18]	集中式	同构	×	×	整数线性规划	最小化亲和性违反数量/资源碎片/机器使用数量,负载均衡
Cooper[51]	集中式	同构	×	✓	启发式算法	资源公平性,提升作业运行效率,减少干扰
文献[52]	集中式	同构	✓	×	打分规则	最小化干扰/提升利用率,提升作业效率

调度性能方面,当集群规模较小时使用集中式调度(centralized scheduling)即可满足作业对于调度延时的要求,集中式调度的特点是所有作业由一个中央调度器进行调度,该方法的优点是全局信息共享,可获得较好的调度质量,采用集中式调度的在离线混部作业调度算法有文献[14,18,34,40,43,47,49,51];当集群规模逐步扩大时,中央调度器成为作业调度的性能瓶颈,分布式调度(distributed schedulig)和层次化调度(hierachical scheduling)采用分治的思想,使用多个调度器共同承担作业调度的压力.分布式调度是指在一个集群中同时运行多个调度器,每个调度器只负责将作业调度至集群中的部分节点,可以实现并发作业调度.分布式调度的优点是可用性和吞吐率高,局限性在于各调度器无法感知集群的全局信息,因此可能产生次优的调度决策.层次化调度是集中式调度和分布式调度的折中方案,层次化调度通常设置两层调度器,作业首先由顶层调度器调度至底层调度器,再由底层调度器将作业调度至计算节点.每个底层调度器只管理一部分计算节点,因此缩小了调度算法的搜索空间.文献[39]采用了层次化的调度方法;文献[48]采取了集中式化调度和分布式调度相结合的调度方法,在线作业由集中式调度器调度以获得较高的调度质量,离线作业则由分布式调度器调度以获得较短的调度延时.

2.4 在离线混部作业资源管理

本节详细介绍在离线混部作业资源管理中的资源隔离技术和资源动态分配的相关研究工作.

2.4.1 资源隔离技术

混部集群单个节点通常会部署多个作业,如 Google 的混部集群有 50%的节点同时运行超过 9 个作业[12].这些作业在运行过程中依赖系统中的多种共享资源,进而产生共享资源竞争.目前解决共享资源竞争的主要方法是通过限制作业对于共享资源的使用量,进而减少作业之间在使用共享资源时发生冲突的可能性.但是,根据排队论原理,多个作业在竞争同一资源时的排队时间是影响作业性能的重要因素,例如多个作业同时竞争 CPU、内存带宽、网络带宽等资源,在这种情况下仅对于作业做资源使用量限制并不能完全消除作业间的资源竞争,因此还需要对作业进行优先级划分以减少高优先级作业在竞争共享资源时的排队时间(通常在线作业会被分配高优先级).本文根据共享资源在系统中的层次将资源隔离技术分为操作系统层的资源隔离技术和硬件层的资源隔离技术.

操作系统层的资源隔离技术对操作系统中的共享资源进行隔离,如:

(1) CPU 时间片资源.为解决多个作业对于 CPU 的竞争,最直接的方法是利用多核体系结构的特性(如图 3

所示),将不同的作业绑定在不同的 CPU 上运行.但是作业独占 CPU 的部署方式会导致较低的 CPU 利用率,进一步提升 CPU 利用率须运行更多的作业并在多个作业间共享 CPU.Linux CGroup 提供了 CPU Share 和 CPU Quota 机制,可通过操作系统的进程调度机制限制作业单位时间内可使用的 CPU 时间片长度;CPU 上的优先级划分则表现为进程优先级(如进程的 Nice 值),文献[53−55]在进程调度中增加了抢占机制,即在线作业可抢占离线作业的 CPU 时间片,增强了在线作业与离线作业共享 CPU 时的抗干扰性.

(2) 网络带宽资源.网络带宽资源作为互联网基础设施中的核心资源,被在线和离线作业高度依赖.在线作业需要通过网络接收用户请求,处理完成后再通过网络将结果发送给用户,因此网络带宽竞争会引起在线作业发送网络包时排队时间的上升,进而影响在线作业的服务质量;离线作业在处理数据前首先需要通过网络读取大量数据,如 MapReduce 作业需要从远端 HDFS 读取数据.因此,离线作业会在网络带宽上产生资源竞争.在 MapReduce 与 Memcached 的混部实验中,MapReduce 在数据读取阶段所产生的网络带宽竞争会使 Memcached 的延迟上升 83 倍[56].网络带宽资源隔离目前存在两种方法:(a) 带宽划分,即为每个作业设定最大网络带宽限制以防止作业过度使用网络带宽而引起过度的资源竞争,采用这种方法的有文献[57−59];(b) 网络包优先级划分,网络包优先级划分方法的主要思想是高优先级作业发送的网络包可以直接越过低优先级作业的发送队列,可有效减少高优先级作业网络包的排队时长,如文献[55,60−62].

(3) 磁盘 I/O 带宽资源.Linux CGroup 提供了作业级别的磁盘 I/O 控制,可限制作业的最大磁盘 I/O 带宽使用量.

Fig.3 Classical multi-cores memory architecuture and recent Intel CPU microarchitecutre[63]

图 3 经典多核存储体系结构与 Intel CascadeLake 微架构[63]

硬件层的资源隔离技术通过软硬件协同技术从协调多个作业在硬件资源上的竞争,减缓甚至消除多个作业在硬件资源上的相互干扰.目前硬件层的资源隔离技术涉及的资源包括:

(1) 内存通道.内存通道(memory channel)是竞争激烈的共享资源之一,对作业的性能影响巨大[64−66].目前数据中心中所使用的微架构通常采用了多通道设计,如图 3(右)所示的 Intel Cascade Lake 架构采用了 6 通道设计,可同时支持 6 个 CPU 独立地访问内存.多个 CPU 在单个内存通道上的访问过程可用排队模型描述,单次内存访问请求的完成时间 $T_{mem\_req}=T_{Queue}+T_{R/W}$,其中,$T_{Queue}$ 代表请求在等待内存通道的排队时间,$T_{R/W}$ 代表内存的存取时间,通常为定值;排队时间 T_{Queue} 则取决于队列长度,即队列中位于该请求之前的请求个数.因此,当一个作业占用过多的内存带宽时,会使同一时段内其他作业的访存请求的排队时间变长,从而产生性能干扰.在冯诺依曼体系结构中,内存作为核心资源,被所有程序所依赖,因此从 CPU 到内存的内存带宽资源对程序至关重要.但对于内存带宽上的干扰隔离目前的硬件缺乏相应的机制,因此,文献[67]提出了一种基于反馈机制的内存带宽保留技术 MemGuard,将内存带宽区分为在线作业和离线作业两部分,使用中断机制控制作业访存请求的发送速率,进而间接控制内存带宽.但是该方法的局限性在于其只能控制单个 CPU 核心的带宽,对于共享 CPU 的作业则不适用.文献

[68]提出了一种动态调整作业访存优先级的方法,实现了内存带宽上的动态优先级控制;

(2) LLC(last level cache)缓存.缓存资源作为影响作业性能的重要资源,其上的干扰同样不可忽略.多个作业在共享缓存时可能出现相互替换的现象,LLC 失效时原本访存指令的执行时间将从 15ns 上升至 70ns,假设 CPU 主频为 3Ghz,则一条访存指令需要多消耗 200 多个周期才能完成[69].消除此类干扰的方法是使用资源划分技术,在物理上划分多个作业对共享资源的使用.图 3(左)所示为经典的多核体系结构,多个 CPU 共享了 LLC,运行于不同 CPU 上的作业会在 LLC 上发生竞争,图 3(右)所示的 Intel Cascade Lake 微架构,通过为每个核设置独立的 LLC 以减少核间对于 LLC 的资源竞争;但是,同一 CPU 上的多个作业在混部运行时仍然会出现缓存相互替换问题,因此需要作业级别的缓存划分技术.为了实现作业级别的缓存划分,Intel 提出了 RDT 技术[70],其中,CAT(cache allocation technology)[71]可为进程或者 CGroup 分配私有的缓存空间,避免缓存相互替换;

(3) 缓存带宽.RDT 技术中的 MBA(memory bandwidth allocation)提供了作业级别的 L2 Cache 带宽划分(即 L2 与 L3 缓存之间的带宽),RDT 技术仅在 Intel 较新型号的处理器上可以使用.

资源隔离技术降低了多个作业在竞争的共享资源时的相互干扰,随着操作系统和硬件架构的不断更新和发展,出现了越来越多的软件或软硬件协同的资源隔离技术,如 GPU 隔离技术[72].但是,由于硬件结构的复杂性,在某些关键资源上仍然缺乏通用且有效的资源隔离技术,如 TLB 快表、L1/2 缓存、总线带宽等,有待进一步深入研究加以解决.

2.4.2 资源动态分配算法

现有的操作系统和虚拟化技术提供了一系列资源隔离技术,可在作业运行时动态调整该作业的可用资源.例如:Linux 提供的 CGroup 提供了包括 CPU、内存、磁盘 I/O、网络 I/O 等多种资源在内的资源弹性伸缩机制;多数商用 CPU 提供了动态调节处理器频率的功能(dynamic voltage and frequency scaling,简称 DVFS),如 Intel 系列处理器[73];缓存方面,Intel 公司推出了 CAT 技术,可为作业分配私有缓存并支持运行时修改作业可用的缓存容量.以现有资源隔离技术为基础,研究人员研究了资源动态分配算法,在作业运行时动态调整各个作业对于共享资源的使用量,进而实现控制和减少作业间性能干扰,提升作业运行效率等目标.图 4 所示为资源动态分配算法的基本工作流,作业在运行过程中所产生的监控数据被输入到资源动态分配算法,算法结合作业性能干扰模型给出资源动态调整决策(如增加资源、减少资源、迁移作业等操作),资源动态调整决策经资源隔离技术修改作业的资源分配,往复循环直至作业结束.

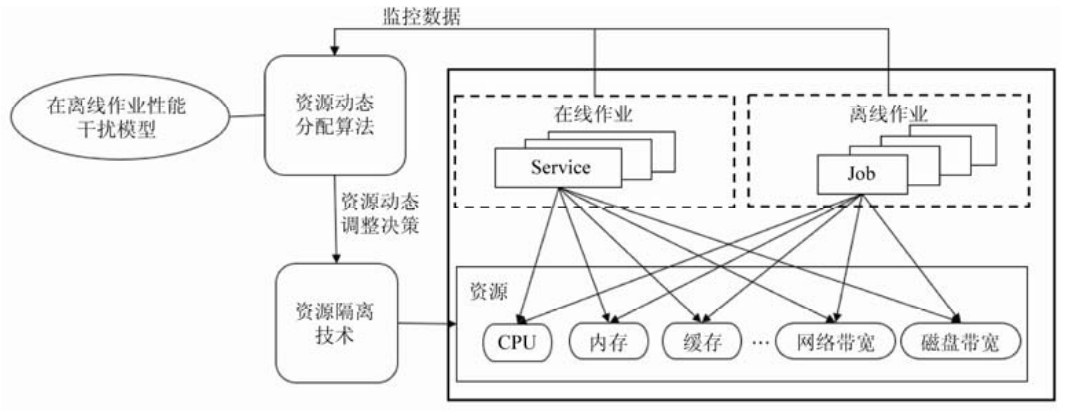

Fig.4 Taxonomy of dynamic resource management algorithms

图 4 资源动态分配算法基本工作流

从算法目标的角度,可将资源动态分配算法分为解决干扰和预防干扰两种.以解决干扰为目标的资源动态分配算法首先持续监控在线作业的性能指标并判断是否发生性能干扰,如果发生,则需要动态调整在离线作业

的资源以减少性能干扰,防止在线作业的性能持续受到影响.解决干扰的相关研究有:

文献[12]中 CPI2 通过数据分析发现了在线作业 CPI 和在线作业 RT 的相关性,因此可以使用硬件级别的指标 CPI 作为性能指标,通过监控和检测 CPI 的波动判断作业是否被干扰,检测到干扰后使用干扰模型识别干扰者,最后使用限制干扰者 CPU 或者杀死干扰者解决干扰.该方法的局限性在于:(1) 整个系统的运行效率依赖于干扰模型对于干扰者的识别准确率;(2) 仅限制干扰者的 CPU 资源而忽略了其他共享资源会导致无法有效地找到干扰源,因此无法准确、有效地处理其他资源上的性能干扰问题.

文献[36]提出了资源管理框架 DeepDive,通过监控底层的性能指标(如 CPI)预测 VM 之间的性能干扰,并识别造成干扰的 VM,随后采用迁移的方法将干扰者迁移到其他节点.实验证明了 DeepDive 中的干扰预测准确率高达 90%以上.但是 DeepDive 针对干扰采取虚拟机迁移的技术可能会使某些资源竞争较高的容器被反复迁移,考虑到迁移容器的开销相对较高,反复迁移可能会引起系统抖动.

表 4 给出了资源动态分配算法的对比情况.其中,▣表示间接控制.

Table 4 Comparisons of resource dynamic allocation algorithm

表 4 资源动态分配算法对比

名称	干扰管理方式	虚拟化技术	监控性能指标	调整的资源							
				CPU	内存	内存带宽	网络	副本数	磁盘	CPU 频率	缓存
Heracles[13]	解决干扰	容器	Latency	✓	✓	▣	✓	×	✓	✓	✓
PARTIES[10]	预防干扰	容器	Latency	✓	✓	×	×	×	✓	✓	✓
CPI^2[12]	解决干扰	容器	CPI	✓	×	×	×	×	×	×	×
PerfIso[74]	预防干扰	–	–	✓	×	×	×	×	✓	×	×
DeepDive[36]	解决和预防干扰	容器	PMC	×	×	×	×	×	×	×	×
BubbleFlux[35]	预防干扰	–	IPC	✓	×	×	×	×	×	×	×
Ginseng[75]	预防干扰	虚拟机	Latency	×	×	×	×	×	×	×	✓
Twig[76]	预防干扰	容器	Latency	✓	×	×	×	×	×	✓	×
AGILE[77]	预防干扰	虚拟机	Latency	×	×	×	×	✓	×	×	×

文献[13]提出了 Heracles,其核心思想是独立地控制多个维度的资源,在多个作业混部运行时使得每种资源的使用率不超过指定阈值.Heracles 使用 CGroups 提供的 CPU Set 隔离 CPU 资源,使用 Intel 处理器提供的 CAT 技术实现缓存动态划分,内存带宽由于没有有效的控制技术,则通过使用调整 CPU 配额的方式间接地控制内存带宽.Heracles 通过 DVFS 动态调控 CPU 的运行频率,网络带宽则使用 Linux 系统提供的 traffic control 进行作业的带宽限制.Heracles 采取了两层结构,顶层控制器基于 SLO 监控数据控制离线作业是否可以增长,底层资源控制器根据具体的资源使用量增加或者减少作业的资源配额.实验中,Heracles 提升了机器的资源利用率,虽然在线作业的性能降低了 20%~30%,但仍然有 99%的 tail latency 满足 SLO 要求.Heracles 的局限性在于每个节点只能支持一个在线作业和多个离线作业的混部,无法在同一节点的不同在线作业之间分配资源.

以解决干扰为目标的资源动态分配算法虽然可以减少或者消除作业间的性能干扰,但是性能干扰事件已经发生,在线作业的性能已经受到不可挽回的损害,可能对于某些至关重要的在线作业是无法接受的.因此研究人员提出了以预防干扰为目的的资源动态分配算法,这类算法在作业运行时根据作业的性能变化趋势进行相应的资源分配.这种方法的优势在于可最大化地保证作业的性能,但其也有过多分配资源从而导致不必要的资源浪费的可能.相关工作包括:

文献[10]提出 QoS 感知的资源管理框架 PARTIES,支持多个在线作业和多个离线作业混部,在作业运行时感知并保证所有在线作业的 QoS.PARTIES 动态调整两大类资源:(1) 计算资源,包括:CPU、LLC、CPU 频率;(2) 存储资源,内存空间、磁盘带宽.在作业运行过程中,PARITES 持续监控作业的请求响应时间 RT,然后使用预定义的状态机决定是否应该增加或减少资源,直到满足作业的 QoS.在 PARTIES 的管理下,混部节点吞吐率相比于 Heracles 提升了 61%,并且可以支持单节点运行更多的服务.但其局限性在于:(1) 由于其无法感知到作业的性能模型和资源偏好,可能会造成资源分配的不均衡,比如过度分配内存给一个计算密集型作业;(2) 需要作业级别的 RT,在公有云环境中,作业通常具有黑盒性,无法获取作业的 RT.

文献[35]提出基于反馈机制的资源管理器 BubbleFlux,在作业运行时持续监控作业的 IPC,根据在线作业

IPC 的大小动态调控在线作业和离线作业的 CPU 时间比例,以达到 LC 作业 QoS 和整体资源利用率最大化的目的.实验结果表明,BubbleFlux 管理下的资源利用率比 Bubble Up[14]提升了 2.2 倍.但是,BubbleFlux 仅能管理 CPU 资源,无法处理其他资源上的资源竞争.

文献[74]提出了 PerfIso 资源管理系统.PerfIso 管理了两类资源以消减干扰:CPU 和磁盘 I/O.在 CPU 的动态分配中,PerfIso 通过保持系统中持续存在空闲的核以处理在线作业的突发流量;在磁盘 I/O 的隔离上,PerfIso 采用的策略是使用 Defict-weighted round-robin 算法动态调整优先级,该算法的基本思想是磁盘 I/O 越多的作业优先级越低.在实验中,作者将 Bing 搜索服务和离线作业进行了混部,在 PerfIso 的管理下 CPU 使用率最多提升到 47%左右.但是该方法的局限性在于:使用隔离 CPU 和预留资源的方法会使系统中永远存在空闲状态的 CPU,进而限制了 CPU 使用率的上限.

随着强化学习的发展,也有研究人员将强化学习应用于资源动态分配算法.文献[76]使用强化学习算法动态调整作业的 CPU 配额和 CPU 的频率.强化学习的优势在于无需人为制定规则,算法会根据预先指定的 reward 函数学习出最佳的策略,结合深度神经网络有着强大的学习能力.其局限性在于:(1) 算法需要的训练时间长;(2) 神经网络的黑盒性导致算法不易调优;(3) 有限的 Action 空间难以应对复杂多变的云计算环境.

3 系统实例

提升集群利用率有利于降低企业数据中心的 TCO(total cost of ownership,总体拥有成本),大幅度提升数据中心的成本效率.因此,在离线混部集群管理系统成为工业界关注的重点领域.国外方面,Google 在其集群管理系统 Borg 中率先尝试了大规模在离线混部[9].国内方面,互联网公司百度、腾讯、阿里均在混部集群管理系统上有所实践,并开源了相应的数据集[78,79].本节就 4 个混部系统实例进行对比分析.

表 5 列举并对比了 Google(Borg)、阿里(Fuxi&Sigma)、百度(Matrix)、腾讯(YARD)的在离线混部作业管理系统.其中,“–”表示现有公开数据中无相关描述.可以看到:

在性能干扰模型方面,4 个系统实例均未将性能干扰模型应用于大规模实践.本文认为造成这一现状的原因为:上述 4 个混部系统实例均进行了大规模部署,运行了海量的在线作业和离线作业,为这些作业构建精准的性能干扰模型所需计算开销过高.虽然性能干扰模型在 Borg 的公开文档中并未提及,但其大规模作业混部环境支撑了多个性能干扰模型的研究,如文献[14,35].

Table 5 Comparisons of co-location systems

表 5 混部系统实例对比

名称	Borg(Google)[9]	Matrix(百度)[80]	Fuxi&Sigma(阿里巴巴)[50,81]	YARD(腾讯)[82]
性能干扰模型	文献[14,35]	–	–	–
在离线混部作业调度	Borg 统一调度	LC 作业:Sorlaria BE 作业:Normandy	LC 作业:Sigma BE 作业:Fuxi	YARD 统一调度
资源隔离技术	容器技术/CAT/CPU 抢占式调度	容器技术/CPU 抢占式调度	容器技术/CAT/超线程隔离/CPU 抢占式调度/网络 IO/磁盘 IO	容器技术/CPU 抢占式调度
资源动态分配	Kill 离线作业/调整离线作业资源	Kill 离线作业/调整离线作业资源	弹性内存/Kill 离线作业/调整离线作业资源	–
运行效果	集群规模压缩为原有集群的 35%	节省数万台服务器	CPU:38%[83]	CPU:45%

在在离线混部作业调度方面,Google 的 Borg(如图 5 左所示)和腾讯的 YARD 采用了统一调度的架构,即在线作业和离线作业由一个调度器统一调度,而百度 Matrix 和阿里 Fuxi&Sigma(如图 5 右所示)则采用了在线离线分离调度的方式,即在离线作业由各自的调度器调度(Sigma 和 Fuxi 分别为阿里内部原有的在线和离线作业管理系统,Sorlaria 和 Normandy 分别为百度内部原有的在线和离线作业管理系统).在离线分离调度的架构复用了原有系统的代码,减少了工程实现的复杂度,但也存在着一定的局限性:(1) 在离线作业调度器以及底层资源管理器信息不共享,需要引入额外的组件进行在离线资源的协调(如 Fuxi&Sigma 中的 Level-0)[25];(2) 系统架构较统一调度更为复杂,每个节点至少运行了 3 个 Agent,带来了额外的资源开销.从调度算法来看,Borg 和 Sigma 采用了基于打分规则的调度算法,打分规则包括:根据负载高低打分、根据资源碎片量打分、根据作业优先级

组合打分等.Fuxi、Matrix、YARD 则是对 YARN[4]调度算法的改进,在调度时考虑了节点空余资源,并使用作业画像、节点画像等预测作业的资源需求和节点未来可用资源.

从资源隔离技术来看,4 个混部系统实例均采用了多种资源隔离技术.容器技术作为轻量级的虚拟化技术被广泛使用;CPU 作为最重要的计算资源,CPU 隔离和抢占式调度成为所有混部系统的选择;CAT 技术也被较多地采用;Fuxi&Sigma 支持更多资源的隔离,如磁盘、网络、超线程上的资源隔离.

从资源动态分配算法来看,基于反馈机制的资源调整算法被广泛应用于实际系统中.其基本思想是:当在线作业的性能受到影响时杀死离线作业或动态调整离线作业的资源配额[84].基于这一思想,Borg 采用了如下策略以降低作业间的性能干扰:(1) 在线作业与物理核(physical CPU cores)绑定,并且同一个物理核只绑定一个在线作业,在线作业运行时可使用 CPU 的全部资源;离线作业被允许运行在任意核上,运行时存在 CPU 使用上限;(2) 根据在线作业资源利用率变化曲线,Borg 预测并动态调整在线作业的资源配额使得并将剩余资源分配给离线作业.基于 Borg 提供的在离线混部环境,Google 公司的研究人员还进行了多种资源动态分配算法的研究,如文献[12,13,35,36];Fuxi&Sigma 则在内存共享方面提出了弹性内存策略,通过在线作业的负载动态调整在离线作业各自的内存配额.

从运行效果来看,4 个系统实例对集群利用率均有明显提升.在 Borg 的管理下,Google 集群的资源使用效率大幅度提高,在 15 个集群上的测试表明,Borg 最多可将一个集群压缩为原有集群规模的 65%,即:使用原有集群 65%的机器便可完成原有集群的所有计算任务;Fuxi&Sigma 混部系统将原有集群的利用率从 10%提升至 40%左右,节约总体成本 30%以上;Matrix 则为百度节省了数万台服务器的成本;YARD 集群 CPU 利用率提升至 45%左右,并且在实践中将深度学习的训练作业与在线作业集群混部,为 2018 年人工智能围棋大赛冠军 PhoenixGo 的训练提供了 14w+CPU 的计算能力.

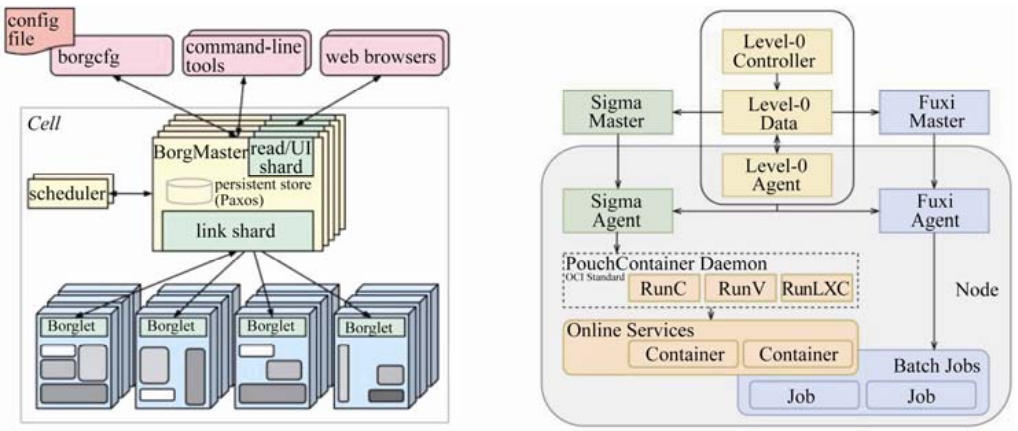

Fig.5 Architecture of Google Borg[9] (left) and Fuxi&Sigma[80] (right)

图 5 Google Borg 系统架构[9](左)与阿里 Fuxi&Sigma 系统架构[80](右)

4 未来研究方向展望

作为学术界和工业界研究的热点领域,在离线混部作业调度与资源管理技术的相关研究近来得到关注,并在实际应用中一定程度地提高了数据中心的资源利用率.但尚有许多亟待解决的问题,本文总结了 4 个未来值得研究的方向,包括:细粒度的性能干扰模型、微服务架构下的混部作业调度算法、软硬件协同的内存带宽干扰隔离技术、面向在离线混部作业的调度模拟技术.

4.1 细粒度的性能干扰模型

目前在混部作业管理系统中使用的性能干扰模型将作业执行(或请求处理)看作一个整体,进而刻画作业整体在不同干扰下性能的变化.例如,描述在下一时段内请求的平均完成时间,或者描述作业的执行时间.性能干

扰模型中的资源竞争 *pressure* 也是作业混部期间的平均压力.实际上,作业的运行过程中包括多个阶段[26],如Hadoop 作业存在 Map 阶段和 Reduce 阶段,基于深度神经网络的智能语音助手 Sirius 在处理请求的过程中存在解析语音-分析语音-生成回复等多个阶段[85].作业在不同阶段因其计算任务不同,存在着不同资源需求和性能敏感度.现有的性能干扰模型针对应用的整体性能特征进行建模,忽略了作业不同阶段的干扰敏感性变化,因此需要细粒度的性能干扰模型描述这种变化.

研究细粒度的性能模型对于混部作业调度和资源管理具有重要意义:在作业调度阶段,基于细粒度干扰模型提供的丰富信息,规划不同作业阶段与阶段之间的有序混部运行,有利于减少资源竞争,降低性能干扰;对于混部资源管理,细粒度的性能干扰模型可提供更丰富的作业运行时资源与性能的变化信息,基于这些信息可以设计更为精细的资源动态算法,合理利用作业运行期间的碎片资源,进一步提高资源利用率.

4.2 面向微服务架构的在离线混部作业调度

随着微服务架构的发展,数据中心出现了越来越多的采用微服务架构的作业[86,87].微服务提升了作业的可扩展性、容错性和可维护性,但是服务间的服务依赖(service dependency)使微服务作业的性能变化特点与传统单体作业有明显不同[38,88,89].在微服务架构中,单个微服务的性能下降会引起其他服务的级联性能下降(cascading performance degradation)[90].例如,某在线作业包含两个微服务,分别是微服务 *A* 和微服务 *B*,微服务 *A* 是用户直接访问的服务,微服务 *A* 在处理用户请求的过程中会调用微服务 *B*.如果 *B* 的性能下降,那么 *A* 等待 *B* 的返回结果的时间变长,导致*A*的对于用户请求的响应时间变长.典型的级联性能下降,如:(1) 网页搜索服务[91],其响应时间取决于最慢的搜索节点;(2) Memcached[92]和 Redis[93]等广泛使用的数据缓存系统,其性能下降会引起上层服务的性能下降.因此,在面向微服务架构的在离线混部作业调度算法不仅需要考虑作业在混部时产生的性能干扰,还需考虑性能干扰引起的级联性能下降,使得在离线混部作业调度问题更加复杂.

面向微服务架构的在离线混部作业调度面临的挑战包括:(1) 构建端到端的请求执行路径具有挑战性.端到端的请求执行路径描述了请求在多个微服务之间处理和转发的过程,是请求级服务依赖关系的表示,与作业的具体执行逻辑相关,是上层调度器的重要数据基础.在大规模微服务集群中,仅凭专家经验或者用户提供的先验知识构建端到端的服务依赖关系并不可行,并且现有的分布式追踪系统仍然存在着数据读写依赖问题和通用性问题[94],因此,如何从海量微服务运行过程中精准、高效、实时地构建端到端的请求执行路径,具有挑战性;(2) 构建面向微服务架构的性能干扰模型具有挑战性.首先,由于作业间的服务依赖关系,原本相互独立的性能干扰模型需要根据服务依赖关系进行联动;其次,需要对微服务之间的通信过程建立性能模型以反映作业性能的真实变化,但是通信性能受 RPC 协议、网络架构、集群负载等多方面影响,具有高度复杂性.因此,构建面向微服务架构的性能干扰模型具有挑战性.

随着微服务架构的广泛使用,越来越多的应用向微服务架构迁移,研究面向微服务架构的在离线混部作业调度将日益重要.

4.3 软硬件协同的内存带宽资源隔离技术

目前,CPU 的运行速率远大于存储器的运行速率,二者性能的不匹配称为“存储墙(memory wall)”[95].虽然多级缓存和缓存预取技术在一定程度上缓解了 CPU 和内存的性能鸿沟,但是并不能从根本上解决这一问题.在目前数据中心广泛采用的多核架构中,存储墙导致内存带宽成为多核架构中竞争最激烈的资源,严重影响了混部作业的运行效率,但是目前的操作系统和硬件技术并不能提供相应的干扰隔离技术.

解决这一问题需要从软件和硬件两方面进行研究.硬件方面需要在内存控制器(memory controler)上进行改进,对每次内存访问进行标签化,进而使得内存控制器可根据标签对内存访问请求进行调度;软件层面需要研发相应的软件配置和动态调整不同作业内存访问的优先级.内存作为冯诺依曼体系结构中最重要且使用最为频繁的计算资源之一,高效隔离内存带宽上资源竞争对上层作业的性能至关重要,该方向的研究成果对进一步提高混部作业的运行效率具有重要意义.

4.4 面向在离线混部作业的调度模拟器

在作业调度算法研究中,使用模拟器验证由于具有快速、低成本等优点已成为重要的验证和评价手段.在离线混部作业调度需要模拟器对作业间的性能干扰进行模拟,但是目前主流的调度模拟器,如 CloudSim[96]以及以 CloudSim 衍生模拟器[97,98]对于作业性能的模拟仅依据作业的资源分配,无法模拟混部作业在体系结构层次竞争资源而引起的性能干扰.计算机体系结构模拟器(如 SMARTS[99]、SimGodon[100]、ZSim[101]、Gem5[102]等)虽然通过指令级模拟提供了精细再现作业的执行过程,可模拟多个作业在硬件上的资源竞争,但其缺点在于运行速度慢,模拟一个主频为 3Ghz 的 CPU 一秒钟内执行的所有指令需要数分钟甚至更长时间,高额的开销使其无法应用于大规模数据中心的模拟.文献[103]提出了微服务模拟器 uqSim,基于排队论模型模拟基于微服务架构的在线作业的运行,但其局限性在于:(1) 只能模拟在线作业,无法模拟离线作业;(2) 无法模拟作业间的性能干扰.

因此,目前仍缺乏面向在离线混部作业的调度模拟器.全方位模拟应用之间的依赖、干扰、竞争等关系,快速分析和验证混部作业调度算法在不同场景下的运行效果,降低调度算法的调试与测试难度,是未来重要的研究方向之一.

5 结束语

大规模数据中心是当今企业级互联网应用和云计算系统的关键支撑.然而,目前数据中心的服务器资源利用率较低(仅为 10%~20%),导致大量的数据中心资源的浪费.将数据中心中的在线作业和离线作业混合部署在同一节点上运行是提升数据中心资源利用率和数据中心成本效率的有效方法,具有较高的经济价值和研究价值.但是,将在线作业和离线作业混合部署面临着诸多问题与挑战,包括:在离线混部作业性能干扰问题;在离线混部作业调度问题;在离线作业的共享资源隔离问题;资源动态分配问题.

本文首先分析了上述问题与挑战,随后围绕在离线混部作业调度与资源管理技术研究框架,详细分析和总结了已有研究工作,并结合多个系统实例分析了在离线混部关键技术在实际系统中的具体应用及运行效果.最后,本文就未来的研究方向进行了展望.

References:

[1] Arman S, *et al*. United States data center energy usage report. Report, 2016.

[2] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1):10.

[3] Zaharia M, Chowdhury M, Franklin MJ, *et al*. Spark: Cluster computing with working sets. In: Proc. of the Usenix Conf. on Hot Topics in Cloud Computing. USENIX Association, 2010. 10.

[4] Vavilapalli VK, Murthy AC, Douglas C, *et al*. Apache hadoop yarn: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. 2013. 1–16.

[5] Hindman B, Konwinski A, Zaharia M, *et al*. Mesos: A platform for fine-grained resource sharing in the data center. NSDI, 2011,11 (2011):22–22.

[6] Burns B, Grant B, Oppenheimer D, *et al*. Borg, Omega, and Kubernetes. Queue, 2016,14(1):70–93.

[7] Du XY, Lu W, Zhang F. History, present, and future of big data management systems. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):127–141 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5644.htm [doi: 10.13328/j.cnki.jos.005644]

[8] Baidu large-scale strategic colocation system evolution. https://www.infoq.cn/article/aEut*ZAIffp0q4MSKDSg

[9] Verma A, Pedrosa L, Korupolu M, *et al*. Large-scale cluster management at Google with Borg. In: Proc. of the 10th European Conf. on Computer Systems. 2015. 1–17

[10] Chen S, Delimitrou C, Martínez JF. PARTIES: QoS-aware resource partitioning for multiple interactive services. 2019. [doi: 10.1145/3297858.3304005]

[11] Zhu HS, *et al*. Kelp: QoS for accelerated machine learning systems. In: Proc. of the 2019 IEEE Int'l Symp. on High Performance Computer Architecture (HPCA). IEEE, 2019.

[12] Zhang X, Tune E, Hagmann R, *et al*. CPI 2: CPU performance isolation for shared compute clusters. In: Proc. of the 8th ACM European Conf. on Computer Systems. ACM, 2013.

[13] Lo D, Cheng L, Govindaraju R, *et al*. Heracles: Improving resource efficiency at scale. ACM SIGARCH Computer Architecture News, 2015,43(3):450–462.

[14] Mars J, Tang L, Hundt R, *et al*. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In: Proc. of the 44th Annual IEEE/ACM Int'l Symp. on Microarchitecture. ACM, 2011. 248–259.

[15] Zhang SG, *et al*. Tail amplification in n-tier systems: A study of transient cross-resource contention attacks. In: Proc. of the 39th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS). IEEE, 2019

[16] Barroso LA, Dean J, Holzle U. Web search for a planet: The Google cluster architecture. IEEE Micro, 2003,23(2):22–28.

[17] Kasture H, Sanchez D. Tailbench: A benchmark suite and evaluation methodology for latency-critical applications. In: Proc. of the 2016 IEEE Int'l Symp. on Workload Characterization (IISWC). IEEE, 2016. 1–10.

[18] Garefalakis P, Karanasos K, Pietzuch P, *et al*. Medea: Scheduling of long running applications in shared production clusters. In: Proc. of the 13th EuroSys Conf. 2018. 1–13.

[19] Xu M, Buyya R. Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions. ACM Computing Surveys (CSUR), 2019,52(1):1–27.

[20] Amazon Blog. https://glinden.blogspot.jp/2006/11/marissa-mayer-at-web-20.html

[21] Card SK, Robertson GG, Mackinlay JD. The information visualizer: An information workspace. In: Proc. of the ACM SIGCHI Conf. on Human Factors in Computing Systems. New York: ACM Press, 1991. 181–188.

[22] Reiss C, Tumanov A, Ganger GR, *et al*. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proc. of the 3rd ACM Symp. on Cloud Computing. 2012. 1–13.

[23] Garg SK, Lakshmi J. Workload performance and interference on containers. In: Proc. of the 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE, 2017. 1–6.

[24] Reiss C, Tumanov A, Ganger GR, *et al*. Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical Report, Intel Science and Technology Center for Cloud Computing, 2012. 84.

[25] Cheng Y, Chai Z, Anwar A. Characterizing co-located datacenter workloads: An Alibaba case study. In: Proc. of the 9th Asia-Pacific Workshop on Systems. 2018. 1–3.

[26] Kozyrakis C. Resource efficient computing for warehouse-scale datacenters. In: Proc. of the 2013 Design, Automation & Test in Europe Conf. & Exhibition (DATE). IEEE, 2013. 1351–1356.

[27] Ghodsi A, Zaharia M, Hindman B, *et al*. Dominant resource fairness: Fair allocation of multiple resource types. NSDI, 2011,11 (2011):24–24.

[28] Isard M, Prabhakaran V, Currey J, *et al*. Quincy: Fair scheduling for distributed computing clusters. In: Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles. 2009. 261–276.

[29] Ousterhout K, Wendell P, Zaharia M, *et al*. Sparrow: Distributed, low latency scheduling. In: Proc. of the 24th ACM Symp. on Operating Systems Principles. 2013. 69–84.

[30] Liu Q, Yu Z. The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from Alibaba trace. In: Proc. of the ACM Symp. on Cloud Computing. 2018. 347–360.

[31] Barve YD, Shekhar S, Chhokra A, *et al*. FECBench: A holistic interference-aware approach for application performance modeling. In: Proc. of the 2019 IEEE Int'l Conf. on Cloud Engineering (IC2E). IEEE, 2019. 211–221.

[32] Zhao JC, Cui HM, Feng XB. Analyzing cross-core performance interference on multi-core processors based on statistical learning. Ruan Jian Xue Bao/Journal of Software, 2013,24(11):2558−2570 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4482.htm [doi: 10.3724/SP.J.1001.2013.04482]

[33] Zhao J, Cui H, Xue J, *et al*. Predicting cross-core performance interference on multicore processors with regression analysis. IEEE Trans. on Parallel and Distributed Systems, 2015,27(5):1443–1456.

[34] Chen Q, Yang H, Guo M, *et al*. Prophet: Precise QoS prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers. ACM SIGOPS Operating Systems Review, 2017,51(2):17–32.

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

[36] Novaković D, Vasić N, Novaković S, *et al*. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In: Proc. of the Presented as part of the 2013 {USENIX} Annual Technical Conf. ({USENIX}{ATC} 13). 2013. 219–230.

[37] Kambadur M, Moseley T, Hank R, *et al*. Measuring interference between live datacenter applications. In: Proc. of the Int'l Conf. on High Performance Computing, Networking, Storage and Analysis. IEEE, 2012. 1–12.

[38] Gan Y, Zhang Y, Hu K, *et al*. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In: Proc. of the 24th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2019. 19–33.

[39] Romero F, Delimitrou C. Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems. In: Proc. of the 27th Int'l Conf. on Parallel Architectures and Compilation Techniques. 2018. 1–13.

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

[41] Delimitrou C, Kozyrakis C. ibench: Quantifying interference for datacenter applications. In: Proc. of the 2013 IEEE Int'l Symp. on Workload Characterization (IISWC). IEEE, 2013. 23–33.

[42] Zhang, Y, Laurenzano MA, Mars J, Tang L. Smite: Precise QoS prediction on real-system smt processors to improve utilization in warehouse scale computers. In: Proc. of the 47th Annual IEEE/ACM Int'l Symp. on Microarchitecture. IEEE, 2014. 406–418.

[43] Tang X, Wang H, Ma X, *et al*. Spread-n-share: Improving application performance and cluster throughput with resource-aware job placement. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. 2019. 1–15.

[44] Gan Y, Pancholi M, Cheng D, *et al.* Seer: Leveraging big data to navigate the complexity of cloud debugging. In: Proc. of the 10th USENIX Conf. on Hot Topics in Cloud Computing. 2018. 13.

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

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

[47] Li Q, Li Y, Tu BB, *et al.* QoS guarenteed dynamic resource in internet data centers. Chinese Journal of Computers, 2014,37(12): 2395̄2407 (in Chinese with English abstract).

[48] Delgado P, Dinu F, Kermarrec AM, *et al*. Hawk: Hybrid datacenter scheduling. In: Proc. of the 2015 {USENIX} Annual Technical Conf. ({USENIX}{ATC} 15). 2015. 499–510.

[49] Vasile MA, Pop F, Tutueanu RI, *et al*. HySARC 2: Hybrid scheduling algorithm based on resource clustering in cloud environments. In: Proc. of the Int'l Conf. on Algorithms and Architectures for Parallel Processing. Cham: Springer-Verlag, 2013. 416–425.

[50] Zhang Z, Li C, Tao Y, *et al*. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. Proc. of the VLDB Endowment, 2014,7(13):1393–1404.

[51] Llull Q, Fan S, Zahedi SM, *et al*. Cooper: Task colocation with cooperative games. In: Proc. of the 2017 IEEE Int'l Symp. on High Performance Computer Architecture (HPCA). IEEE, 2017. 421–432.

[52] Zhang Y, Prekas G, Fumarola GM, *et al*. History-based harvesting of spare cycles and storage in large-scale datacenters. In: Proc. of the 12th {USENIX} Symp. on Operating Systems Design and Implementation ({OSDI} 16). 2016. 755–770.

[53] Leverich J, Kozyrakis C. Reconciling high server utilization and sub-millisecond quality-of-service. In: Proc. of the 9th European Conf. on Computer Systems. 2014. 1–14.

[54] Duda KJ, Cheriton DR. Borrowed-VirtualTime (BVT) Scheduling: Supporting latency-sensitive threads in a general-purpose scheduler. In: Proc. of the SOSP. 1999.

[55] Improve CPU utilization to 90%. https://cloud.tencent.com/developer/article/1519559

[56] Grosvenor MP, Schwarzkopf M, Gog I, *et al*. Queues Don't matter when you can {JUMP} Them! In: Proc. of the 12th {USENIX} Symp. on Networked Systems Design and Implementation ({NSDI} 15). 2015. 1–14.

[57] Jeyakumar V, Alizadeh M, Mazieres D, *et al*. EyeQ: Practical network performance isolation at the edge. NSDI, 2013.

[58] Perry J, Ousterhout A, Balakrishnan H, *et al*. Fastpass: A centralized "zero-queue" datacenter network. In: Proc. of the 2014 ACM Conf. on SIGCOMM. 2014. 307–318.

[59] Vattikonda BC, Porter G, Vahdat A, *et al*. Practical TDMA for datacenter Ethernet. In: Proc. of the 7th ACM European Conf. on Computer Systems. 2012. 225–238.

[60] Vamanan B, Hasan J, Vijaykumar TN. Deadline-aware datacenter TCP (D2TCP). ACM SIGCOMM Computer Communication Review, 2012,42(4):115–126.

[61] Hong CY, Caesar M, Godfrey PB. Finishing flows quickly with preemptive scheduling. ACM SIGCOMM Computer Communication Review, 2012,42(4):127–138.

[62] Zats D, Das T, Mohan P, *et al*. DeTail: Reducing the flow completion time tail in datacenter networks. In: Proc. of the ACM SIGCOMM 2012 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication. 2012. 139–150.

[63] Cascade Lake—Microarchitectures. https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake

[64] Xiang Y, Ye C, Wang X, *et al*. EMBA: Efficient memory bandwidth allocation to improve performance on Intel commodity processor. In: Proc. of the 48th Int'l Conf. on Parallel Processing. 2019. 1–12.

[65] Park J, Park S, Han M, *et al*. Hypart: A hybrid technique for practical memory bandwidth partitioning on commodity servers. In: Proc. of the 27th Int'l Conf. on Parallel Architectures and Compilation Techniques. 2018. 1–14.

[66] Hashemi M, Swersky K, Smith JA, *et al*. Learning memory access patterns. arXiv Preprint arXiv: 1803.02329, 2018.

[67] Yun H, Yao G, Pellizzoni R, *et al*. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: Proc. of the 19th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2013. 55–64.

[68] Subramanian L, Lee D, Seshadri V, *et al*. BLISS: Balancing performance, fairness and complexity in memory access scheduling. IEEE Trans. on Parallel and Distributed Systems, 2016,27(10):3071–3087.

[69] Hennessy JL, Patterson DA. Computer Architecture: A Quantitative Approach. Elsevier, 2011.

[70] Intel CAT. https://github.com/intel/intel-cmt-cat

[71] Herdrich A, Verplanke E, Autee P, *et al*. Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family. In: Proc. of the 2016 IEEE Int'l Symp. on High Performance Computer Architecture (HPCA). IEEE, 2016. 657–668.

[72] Chen Q, Yang H, Mars J, *et al*. Baymax: QoS awareness and increased utilization for non-preemptive accelerators in warehouse scale computers. Computer Architecture News, 2016,44(2):681–696.

[73] Charles J, Jassi P, Ananth NS, Sadat A, Fedorova A. Evaluation of the Intel® core i7 turbo boost feature. In: Proc. of the IEEE Int'l Symp. on Workload Characterization. 2009.

[74] Iorgulescu C, Azimi R, Kwon Y, *et al*. Perfiso: Performance isolation for commercial latency-sensitive services. In: Proc. of the 2018 {USENIX} Annual Technical Conf. ({USENIX}{ATC} 18). 2018. 519–532.

[75] Funaro L, Ben-Yehuda OA, Schuster A. Ginseng: Market-driven {LLC} allocation. In: Proc. of the 2016 {USENIX} Annual Technical Conf. ({USENIX}{ATC} 16). 2016. 295–308.

[76] Nishtala R, Petrucci V, Carpenter P, *et al*. TWIG: Multiagent task management for colocated latency-critical cloud services. In: Proc. of the Int'l Symp. High-performance Computer Architecture. New York: ACM, 2020.

[77] Nguyen H, Shen Z, Gu X, *et al*. {AGILE}: Elastic distributed resource scaling for infrastructure-as-a-service. In: Proc. of the 10th Int'l Conf. on Autonomic Computing ({ICAC} 13). 2013. 69–82.

[78] Google Trace Data. 2015. https://github.com/google/cluster-data

[79] Alibaba Cluster Data. 2018. https://github.com/alibaba/clusterdata

[80] Matrix Warehouse Computer OS. https://myslide.cn/slides/570

[81] Alibaba Inc. Evolution of Alibaba large-scale colocation technology. 2018. https://www.alibabacloud.com/blog/evolution-of-alibaba-large-scale-colocation-technology_594172

[82] Tencent Yard. https://myslide.cn/slides/9806

[83] Guo J, Chang Z, Wang S, *et al*. Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces. In: Proc. of the Int'l Symp. on Quality of Service. 2019. 1–10.

[84] Jiang C, Han G, Lin J, *et al*. Characteristics of co-allocated online services and batch jobs in internet data centers: A case study from Alibaba cloud. IEEE Access, 2019,7:22495–22508.

[85] Hauswald JMA, Zhang LY, Li C, Rovinski A, Khurana A, Dreslinski R, Mudge T, Petrucci V, Tang L, Mars J. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In: Proc. of the 20th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York: ACM, 2015. 223–238.

[86] Dragoni N, Giallorenzo S, Lafuente AL, *et al*. Microservices: Yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering. Cham: Springer-Verlag, 2017. 195–216.

[87] Jamshidi P, Pahl C, Mendonça N C, *et al*. Microservices: The journey so far and challenges ahead. IEEE Software, 2018,35(3): 2435.

[88] Kannan RS, Subramanian L, Raju A, *et al*. Grandslam: Guaranteeing slas for jobs in microservices execution frameworks. In: Proc. of the 14th EuroSys Conf. 2019. 2019. 1–16.

[89] Sriraman A, Dhanotia A, Wenisch TF. Softsku: Optimizing server architectures for microservice diversity@ scale. In: Proc. of the 46th Int'l Symp. on Computer Architecture. 2019. 513–526.

[90] Gan Y, Zhang Y, Cheng D, *et al*. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: Proc. of the 24th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM, 2019. 3–18.

[91] Dean J, Barroso LA. The tail at scale. Communications of the ACM, 2013,56(2):74–80.

[92] Memcached. https://www.memcached.org/

[93] Redis. https://redis.io/

[94] Yang Y, Wan L, Gu J, Li Y. Transparently capturing execution path of service/job request processing. In: Proc. of the 16th Int'l Conf. on Service Oriented Computing (ICSOC 2018). 2018. 12–15.

[95] Wulf WA, McKee SA. Hitting the memory wall: Implications of the obvious. ACM SIGARCH Computer Architecture News, 1995, 23(1):20–24.

[96] Calheiros RN, Ranjan R, De Rose CAF, *et al.* CloudSim: A novel framework for modeling and simulation of cloud computing infrastructures and services. Computer Science, 2009.

[97] Chen W, Deelman E. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. In: Proc. of the IEEE Int'l Conf. on E-science. 2013.

[98] Bux M, Leser U. DynamicCloudSim: Simulating heterogeneity in computational clouds. In: Proc. of the ACM SIGMOD Workshop on Scalable Workflow Execution Engines & Technologies. 2013.

[99] Wunderlich RE, Wenisch TF, Falsafi B, *et al*. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In: Proc. of the 30th Annual Int'l Symp. on Computer Architecture. 2003. 84–97.

[100] Zhang FX, Zhang LB, Hu WW. Sim-Godson: A godson processor simulator based on SimpleScalar. Chinese Journal of Computers, 2007,30(1):68 (in Chinese with English abstract).

[101] Sanchez D, Kozyrakis C. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. ACM SIGARCH Computer Architecture News, 2013,41(3):475–486.

[102] Binkert N, Beckmann B, Black G, *et al*. The Gem5 simulator. ACM SIGARCH Computer Architecture News, 2011,39(2):1–7.

[103] Zhang Y, Gan Y, Delimitrou C. uqSim: Scalable and validated simulation of cloud microservices. arXiv Preprint arXiv: 1911.02122, 2019.

附中文参考文献:

[7] 杜小勇,卢卫,张峰.大数据管理系统的历史、现状与未来.软件学报,2019,30(1):127−141. http://www.jos.org.cn/1000-9825/5644.htm [doi: 10.13328/j.cnki.jos.005644]

[32] 赵家程,崔慧敏,冯晓兵.基于统计学习分析多核间性能干扰.软件学报,2013,24(11):2558−2570. http://www.jos.org.cn/1000-9825/4482.htm [doi: 10.3724/SP.J.1001.2013.04482]

[47] 李青,李勇,涂碧波,等.QoS 保证的数据中心动态资源供应方法.计算机学报,2014,37(12):2395−2407.

[100] 张福新,章隆兵,胡伟武.基于 SimpleScalar 的龙芯 CPU 模拟器 Sim-Godson.计算机学报,2007,30(1):70−75.

王康瑾(1993－),男,博士,主要研究领域为云计算系统,在离线混部系统.

李影(1975－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式计算,可信计算.

贾统(1993－),男,博士,主要研究领域为分布式系统,智能运维.