DOI:10.19789/j.1004-9398.2024.06.006
文献引用:高萌涓,王艺达,孙静怡,等.学习型操作系统和编译器研究综述[J].首都师范大学学报(自然科学版),2024,45(6):49-61. GAO M J,WANG Y D,SUN J Y,et al. A survey on learned operating system and compiler[J]. Journal of Capital Normal University(Natural Science Edition),2024,45(6):49-61.

学习型操作系统和编译器研究综述*

高萌涓[1],王艺达[1],孙静怡[1],桑可佳[1],王奕涵[1],徐远超[1,2**]

(1. 首都师范大学信息工程学院,北京 100048;2. 中国科学院计算技术研究所处理器芯片全国重点实验室,北京 100190)

摘要:机器学习方法具有从大量数据中提取特征并动态优化系统性能的能力,然而机器学习赋能系统软件面临众多挑战。本文重点从学习型操作系统和学习型编译器2个方面回顾了机器学习在计算机系统中的应用,探讨了机器学习赋能系统软件的挑战以及潜在研究方向,期望能为研究者提供参考。

关键词:机器学习;学习型;系统软件;操作系统;调度;编译器
中图分类号:TP31 **文献标志码**:A

A survey on learned operating system and compiler*

GAO Mengjuan[1], WANG Yida[1], SUN Jingyi[1], SANG Kejia[1], WANG Yihan[1], XU Yuanchao[1,2**]

(1. College of Information Engineering, Capital Normal University, Beijing 100048;
2. State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

Abstract: Machine learning methods have the ability to extract features from large amounts of data and dynamically optimize system performance, but there are still many challenges for machine learning to enable system software. This paper mainly surveys the research of machine learning in computer system from two aspects of learning-based operating system and learning-based compiler, discusses the challenges and potential research directions of machine learning enabling system software, and hopes to provide references for researchers.

Keywords: machine learning; learning-based; system software; operating system; scheduling; compiler
CLC: TP31 **DC**: A

0 引 言

随着数据规模的不断增长和计算能力的提升,机器学习算法不断演进,为各行业带来了巨大的机遇和挑战。为了追寻更好的人工智能(artificial intelligence,AI)体验,更复杂、更庞大的模型应运而生,但是同时模型训练的计算量呈指数型增长。为了满足人工智能应用的计算需求,计算机架构和系统的创新面临着更大的压力,同时也为系统软件的自动化、智能化和自治化带来了很多机会[1]。

首先,计算机系统是由人类专家根据直觉和启发式方法进行设计的,这需要机器学习(machine learning,ML)和系统方面的专业知识。但是这些基于启发式的设计并不具有适应性和可扩展性,启发

收稿日期:2024-05-31
*北京市自然科学基金项目(4212017);处理器芯片全国重点实验室开放课题(CLQ202410)
**通信作者:xuyuanchao@cnu.edu.cn

式方法只能在特定的场景下表现最优。然而ML能够通过不断学习和训练来适应新的数据模式和变化,并高效应用到其他场景中。当面对新的数据或情境时,ML可以根据不同情境和需求做出智能化的调整和决策,提高系统的效率和性能。这种自适应性和可扩展性使得ML系统能够持续改进和优化自身,适应不断变化的需求和挑战,为用户提供更加个性化和有效的服务。

其次,随着ML模型规模越来越大,模型对算力硬件的需求将会更高,然而面对如此大的运算量,启发式方法和基于人工设计的方法已然失效,传统的系统架构已无法支持强大的ML模型运行,需要寻求新的解决方法。而ML可以通过对数据的学习和训练,自动发现数据之间的潜在模式和规律,实现对数据的快速分类、聚类、预测等操作,从而为数据处理提供更为智能化的解决方案。而且,ML天生具有捕捉复杂属性和提取有价值信息的能力,这些信息甚至是领域专家都无法获取到的。

最后,传统的资源管理器通常需要用户手动干预和设置资源分配,对于复杂多变的系统来说,用户可能需要花费大量时间和精力来进行资源管理,增加了工作负担,并且无法根据实时需求进行动态调整,缺乏智能化的决策能力,造成资源利用率低下和性能表现不佳。而ML可以帮助系统软件更智能地管理资源。通过深度学习和强化学习(reinforcement learning,RL),系统可以根据实时需求对资源进行动态分配和优化,提高资源利用率,降低成本,并确保系统在高负载时仍能正常运行。

计算机系统设计正在向更自动化、更智能化的方向发展,ML和系统设计之间的关系正在被重新考虑。在过去的10年中,优化架构和系统以加速ML模型的执行并提高其性能。最近,出现了将ML应用于计算机系统的迹象,这不仅减少人工设计系统专家的负担,从而提高设计师的工作效率;还形成了闭合正反馈循环,即计算机系统适配ML,同时ML辅助计算机系统,形成良性循环,实现共赢。

然而,在计算机系统中应用ML的研究实践存在巨大挑战。首先,缺乏关于在计算机系统中部署ML的共同指南,例如,何时选择ML算法而不是传统的方法。到目前为止,研究工作分散在不同的研究领域,缺乏整体的观点使研究人员难以获取深入的见解或从相关领域借鉴思路。其次,如何为计算机系统中的每个不同问题选择合适的ML技术。特别是,有些问题只能通过特定的ML算法优化,而有些问题可以通过多种ML技术解决,选择最佳技术并非易事。

操作系统和编译器是计算机中2类最重要的系统软件,本文对ML在操作系统和编译器中的相关研究进行了全面综述。内容安排如下:第1节详细阐述ML在操作系统中的应用研究;第2节详细阐述ML在编译器中的应用研究;第3节讨论ML应用于计算机系统设计面临的挑战和潜在的研究方向;最后对全文进行总结。

1 ML在操作系统中的运用

操作系统管理硬件资源并为应用程序的执行提供受保护的环境,因此设计会影响在其上运行的所有应用程序。然而当今主流操作系统中的大多数功能都是为通用目的而设计的,不能很好地支持各种类型的应用程序或硬件,并且很难调优。

利用ML技术可以根据输入实现更快速更准确的预测,不需要额外的人工干预。使用ML预测操作系统的最佳配置,这样的配置可以在运行时不断适应应用程序的变化。ML还可以用于生成某些操作系统功能的策略和机制。通过设计和构建框架训练和使用操作系统的ML模型,可以避免开发操作系统的巨大工程量。此外,针对不同的应用程序,ML的方法具有更强的适应性,以产生更好的结果。

1.1 学习型操作系统的挑战

操作系统是计算机系统的核心,为了跟上现代硬件和应用程序发展的速度,操作系统开发不应该仅仅依靠人类的智慧,还应该利用人工智能和ML技术来自动“学习”如何构建和调整操作系统。文献[2]指出通用操作系统很难正确地调优,且不能很好地支持各种类型的应用程序或硬件;探讨了ML在操作系统中的机遇,包括学习配置,学习策略,学习映射机制;阐述了ML用于操作系统的挑战,包括模型构建、模型训练、模型推理、模型集成以及安全和隐私问题。

然而,在操作系统中使用ML库有3个挑战:需要内核编程技术;在内核空间中对ML库很难进行调试,并且对bug和性能开销非常敏感;操作系统的某些服务质量(quality of service,QoS)要求可能需要在内核空间中部署ML模型,以避免用户态-内核态切换所产生的额外成本。为此,文献[3]开发KMLib,针对内核空间组件的轻量级但高效的ML

引擎,通过降低精度训练使ML库的计算成本以及内存开销达到最低,更容易调试机器学习程序。

在操作系统内核中使用ML时,加速器在用户空间中运行,应用程序通过调用应用程序编程接口(application programming interface,API)来使用加速器。但是,这种方式无法直接访问系统的所有硬件资源,无法实现操作系统级别的资源管理和调度。虽然KMLib[3]是将ML生态系统引入操作系统的最新尝试,但与主流ML框架相比,其提供的功能有限,并且不支持加速器。为此,文献[4]提出ML辅助内核,记为lake,使内核空间应用程序能够使用加速器。如图1所示,当应用程序调用加速器API时,执行流程会切换到lakeLib模块,创建一个足够大的命令缓冲区来保存API函数标识符和所有函数参数,之后通过套接字的通道将此命令发送到lakeD,此时进入用户空间,命令将被反序列化,请求的API将在加速器上执行,执行完成后,将使用返回值构建返回命令并发回。

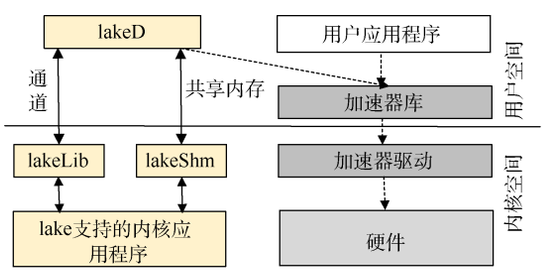


图1　Lake基本架构[3]

为了使操作系统内核可以针对特定的场景进行优化并具备强大的泛化能力,文献[5]提出了“可重配置内核数据路径”的概念,旨在使内核能够动态自我优化。其关键点是利用ML从经验数据中计算出优化策略,并通过内核虚拟机集成到内核中,虚拟机实现了可重构匹配表(reconfigurable match table,RMT)抽象,将优化策略通过RMT虚拟机集成到性能关键事件发生的内核的位置,查找当前的执行环境,执行特定的优化策略,以动态实现内核优化。

1.2　基于机器学习的调度器

操作系统管理系统的有限资源,当有多个任务要使用这些资源时,因为资源的有限性,必须按照一定的规则选择任务来占用资源,这就需要对任务进行调度,调度的目的是确保资源得到合理分配,最大程度地提高系统资源利用率、响应速度和整体性能。为了自适应和动态调整策略,深度学习和RL是实现实时调度的常用方法。本小节将从不同的应用场景探究如何将ML用于调度优化。

(1)集群调度。

传统的集群调度器忽略了作业结构(即内部依赖)和作业输入大小的高效并行度等有效信息。为此,文献[6]提出Decima,使用图神经网络(graph neural network,GNN)进行数据处理,以应对任意形状和大小的有向无环图(directed acyclic graph,DAG),并使用RL策略网络进行调度以应对随机任务到达序列,为每个作业设置高效的并行度。具体而言,如图2所示,首先对于每个作业中的每个节点,通过GNN转化为每个节点嵌入、每个作业嵌入和全局嵌入,将这3个向量输入到策略网络中,输出指定调度的结点以及结点的并行度,集群环境执行此动作并获得相应的奖励,根据奖励更新策略网络,以实现高效调度分布式计算集群上的数据处理任务。

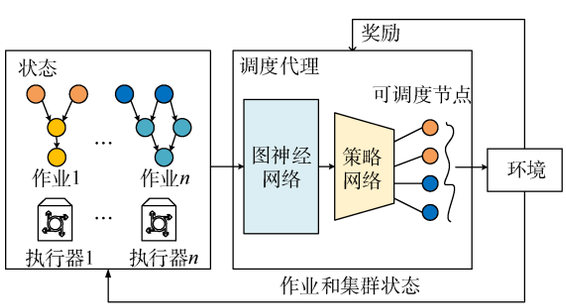


图2　Decima强化学习框架[6]

在传统的单任务学习中,模型被训练用于解决特定的任务,例如分类或回归。然而,在现实世界的许多情况下,可能面临多个相关的任务,这些任务之间可能存在一定的关联性或共享某些信息。多任务学习(multi-task learning,MTL)试图利用这些任务之间的关系或共享的特征,提高多个任务的泛化能力。传统的调度器通常采用启发式或固定策略,忽略了任务之间的关系。为此,文献[7]引入深度Q-Learning来学习MTL调度,如图3所示,该调度器使用任务不确定性直方图来监控任务的状态和共享特征,利用深度Q-Learning预测该状态下预期的动作,并使用奖励机制更新网络模型,通过试错学习最优的任务调度策略。

该领域的另外一个应用方向是优化设备放置策略。随着模型的增大,训练和推理的规模和计算需求也在增加。最常见的解决方法是使用混合硬件设备,如中央处理器(central processing unit,CPU)和图形处理器(graphics processing unit,GPU)的异构

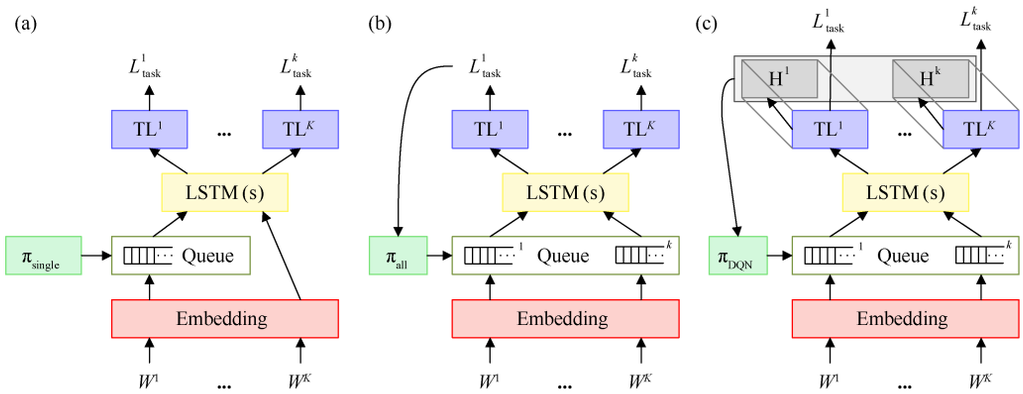


图3　多任务学习的不同调度程序[7]

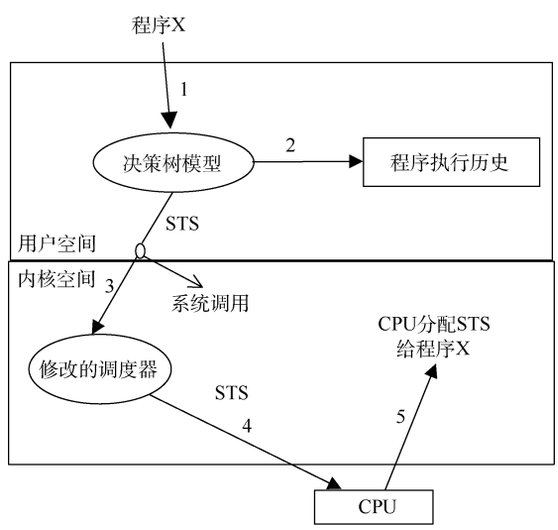
(a)单任务调度器;(b)其他研究提出的多任务调度器;(c)本文提出的多任务调度器

分布式环境,但是将部分神经模型放置在设备上的决定通常是由人类专家基于简单的启发式和直觉做出的,当网络有许多分支或者数据集很大时,人为做出决策是非常困难的。为此,文献[8]提出使用RL来优化计算图的设备放置方法,算法的关键是使用序列到序列的模型来读取有关算子的输入信息及其之间的依赖关系,为每个算子提出一个放置方案,每个方案都在硬件环境中执行,以测量执行时间,将执行时间用作奖励信号来训练模型,使其随着时间的推移给出更好的放置方案。

(2)CPU调度。

通常,进程调度器根据调度算法为进程分配CPU时间片,但该算法不使用进程以前的任何执行历史。如果能够识别一个程序并预测其资源需求,就可以提高程序的执行效率。为此,文献[9]提出通过描述或识别程序,了解以前的执行历史并预测其资源需求,最大限度地减少进程周转时间。具体而言,如图4所示,首先将程序作为输入给C4.5决策树,决策树对该程序进行分类并输出最小化程序执行时间所需的CPU时间片大小,称之为特殊时间片(special time slice,STS),通过系统调用将此STS信息发送给调度器,调度器根据接收到的STS引导CPU分配STS个ticks给程序。

在多处理器系统中,Linux的完全公平调度算法(completely fair scheduler,CFS)根据平均CPU利用率跟踪进程负载,以平衡处理器内核之间的工作负载。这种方法最大限度地利用了CPU资源,同时保证了进程之间的公平性,但忽略了对底层硬件资源的争用。在运行计算密集型工作负载的服务器中,对有限计算资源的不平衡需求阻碍了执行性能,为此,文献[10]提出了一种基于多层感知机的



注:CPU为中央处理器,STS为特殊时间片。

图4　基于决策树的时间片预测[9]

资源感知负载平衡器,通过动态内核跟踪收集运行时系统数据,根据收集的数据训练多层感知机模型,模仿CFS负载均衡器的决策,在内核中部署模型并进行推理。

相关研究的另外一个方向是根据用户的偏好自动分配资源。当前的静态资源划分方法往往无法将足够的资源用于用户当时最关心的应用程序,从而导致性能和用户体验度下降。文献[11]提出SmartOS,学习用户的个人偏好,优先分配资源给用户认为最重要的应用程序,收集用户的数据和反馈,并相应地调整其资源分配策略。具体而言,如图5所示:首先检测用户信息并收集资源使用情况、鼠标点击和移动、音频功能等相关数据,利用该数据离线开发一个模型预测用户满意度;随后根据状态信息预测哪种资源分配策略最能满足用户的期望并按照该策略对资源进行分配,得到下一个状态

之后,不提供手动反馈,而是让人类沮丧预测 ML 模型根据下一个状态预测用户的沮丧程度,根据奖励相应地调整资源分配策略。

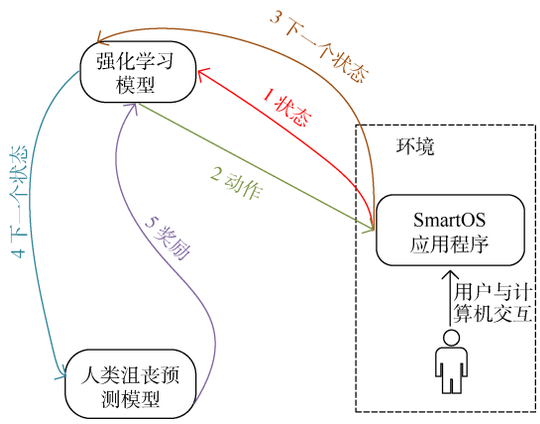


图 5　SmartOS 中的预训练强化算法[11]

(3)磁盘调度。

磁盘输入/输出(imput/output,I/O)调度器的性能受到许多因素的影响,例如工作负载、文件系统和磁盘系统。可以通过调优调度器参数来提高磁盘调度性能。调度器性能调优主要是手动完成的。为了实现自动化调优,文献[12]提出学习型磁盘调度方案,如图 6 所示,在传统的磁盘调度中嵌入一个决策模型,模型根据工作负载、磁盘使用情况、历史记录数据等信息做出相应的调度决策,并将相关数据记录在日志数据库中用于更新模型。该文献提出 4 种实现自动调度策略选择的算法,即变化感知的轮询、反馈学习、请求级学习和两层学习算法。两层学习算法学习效果最好,因其集成了工作负载级和请求级学习算法,并采用反馈学习技术来自动分析工作负载、更改调度策略和调整调度参数。

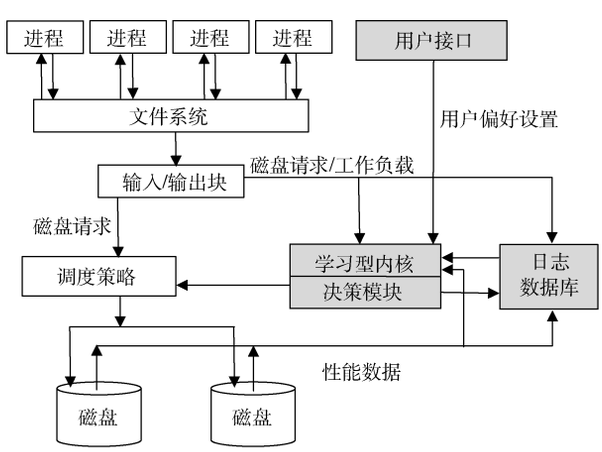


图 6　自我学习的磁盘调度架构[12]

1.3　基于机器学习的资源管理

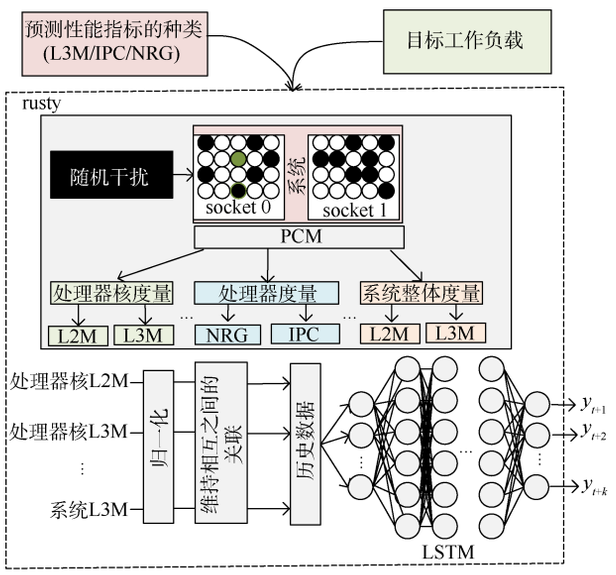
现代体系结构和系统变得如此复杂和多样化,以至于优化性能和充分利用系统资源是非常重要的。各种具有特定需求或目标的工作负载的出现使这个快速发展的领域进一步复杂化。一种解决方法是开发更有效和自动化的资源管理和任务分配方法,其中,基于 ML 的技术擅长于探索大型设计空间并同时优化多个目标,可以智能迅速地根据动态工作负载或指定的约束条件来调整策略,并在精心设计后保持更好的可扩展性和可移植性。

负载特征的数据采集对资源管理至关重要,采用 ML 技术对采集到的数据进行学习,进而预测工作负载特征,提高资源利用率。为此,文献[13]提出 resource central(RC),旨在收集大型云平台中虚拟机工作负载的遥测数据、离线学习其行为并在线提供预测以增强资源管理、提高资源利用率。RC 从虚拟机收集大量遥测数据,包括资源利用率指标;使用这些数据训练 ML 模型,以捕捉虚拟机的行为模式,预测各种工作负载特性,例如未来的 CPU 利用率和工作负载类别;经过训练的模型部署在在线环境中,并提供实时预测。通过利用这些机器学习模型,RC 可以改善云环境中的资源管理,从而提高利用率并防止资源耗尽。

在云计算环境中,同时存在着延迟敏感型任务和批处理任务 2 种任务。对于云计算供应商来说,理想的资源管理方法能够动态调整资源的分配,将延迟敏感型任务的空闲资源回收并分配给批处理任务,从而提高总体资源利用率。然而,传统的资源回收方法或者过于保守从而导致回收率较低,或者过于激进从而导致延迟过高。为此,文献[14]提出使用多分类线性回归模型,预测定长时间窗口内延迟敏感型任务所需要的计算资源数量,从而决定回收多少空闲资源给批处理任务。通过使用这种轻量级的 ML 模型,云资源管理系统能够在线学习并适应不同延迟敏感型任务在不同场景下的负载变化特点,从而找出最优的资源回收策略。

混合工作负载是指系统中同时存在多种不同类型的工作负载,这些工作负载可能包括计算密集型、内存密集型和 I/O 密集型等任务。在云数据中心,通常采用混合工作负载作为提高资源利用率的有效机制。然而,混合工作负载正在以非常规和不可预测的方式影响资源可用性。有效的资源管理需要有持续的、预测性的系统指标运行时知识。为

此,文献[15]提出rusty,能够通过利用长短期记忆网络(long short-term memory,LSTM)的能力在运行时预测在受干扰的系统上执行的应用程序的性能指标。如图7所示,在训练过程中,rusty将目标应用程序和要预测的性能指标的种类作为输入,目标应用程序在系统上执行100次,每次都具有不同的干扰负载,在执行过程中收集有关系统性能指标的信息,并用其训练LSTM模型,以得到基于长期顺序依赖的性能预测模型。

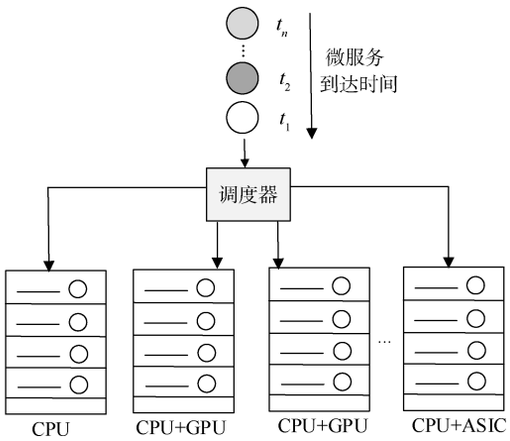


注:PCM为性能计数器监测;LSTM为长短期记忆网络。

图7　Rusty架构概述[15]

该领域的另外一个应用方向是异构集群调度。近年来,云数据中心的硬件正逐渐异构化和多样化,这种异构硬件在加速和扩展微服务能力的同时,也使微服务调度更加复杂。如图8所示,如何实现微服务异构集群调度成为一个极具挑战性的问题。研究表明,利用神经网络的数据驱动ML方法可以改善应用程序的端到端延迟和QoS违规的概率,然而这些工作都集中在同构的集群上。为此,文献[16]提出一种基于决策树的轻量级调度器Octopus。对于可以在任何设备上执行的任务,Octopus选择满足用户要求的最佳设备;对于在特定设备上执行的任务,Octopus引入一种约束机制,保留一定比例的设备资源,用于只能在该设备上执行的任务,这些资源不会被用于可以在任何设备上执行的任务。

为了节省成本,多个服务通常运行在一台服务器上。因此,资源调度成为保障服务质量QoS的关键。此外,现有的调度器无法轻易避免"资源悬崖",即在调度过程中仅略微减少一个资源就会导致QoS显著降低。为此,文献[17]提出OSML,基于ML的智能调度。OSML利用多种多层感知机模型来学习架构提示、调度方案以及QoS需求之间的关联,并通过预测QoS变化和资源边界来生成适当的资源分配方案,智能地避免资源悬崖。同时,这种算法利用深度Q网络(deep Q-network,DQN)来管理资源分配,并且可以有效地处理QoS违规和资源过度分配的情况。



注:GPU为图形处理器;ASIC为集成电路芯片。

图8　微服务调度系统概述[16]

AI与操作系统相辅相成,AI技术能够赋予操作系统更强大的智能处理能力,使其更好地理解和响应用户的需求;二者的结合,也将产生巨大的协同效应,推动计算机系统的智能化提升到更高的水平,进而更好地服务AI应用系统。

2　ML在编译器中的运用

2.1　研究概述

编译是用高级语言编写的源程序转换成计算机可以直接执行的机器语言的过程。在编译一个源程序时,编译器需要执行包括词法分析、语法分析、语义分析、中间代码生成以及代码优化和目标代码生成在内的多个中间步骤,才能将给定的源程序转化为目标程序。在这些中间步骤中,代码优化(也称为编译优化,以便和源代码级别的程序优化相区别)最为重要,也最为复杂。

编译优化的复杂性主要体现在其极为巨大的策略空间上。具体而言,编译优化作为编译的一个中间步骤,其本身也是由非常多的优化流程所构成的。因此,编译器在对任何一个源程序进行编译优化时都需要考虑以下3组问题,共同构成了一个极

为巨大的策略空间:(1)应该选择哪些优化流程;(2)如何排布这些优化流程之间的调用顺序;(3)如何决定某些优化流程内部的参数值——比如函数内联流程就需要一个阈值参数来判定不同的函数是否应该内联。此外,不同的目标体系结构、不同的程序优化目标都会对程序的最佳编译优化策略产生影响,在另一个层面上增大了编译优化的复杂性。

在应对编译优化巨大的策略空间时,相较于传统的启发式方法,ML方法往往更加有效。因为ML方法具有高效表征源程序静态、动态特征的能力,并且可以充分利用编译器的历史运行数据,学习具有不同特征的源程序与其最佳编译优化策略之间的隐藏关系,从而为未见过的源程序进行策略预测。更为关键的是,ML方法能够自适应不同的新兴体系结构或优化目标,而不需要像启发式方法一样依赖人工设计与调优。

根据研究对象的不同,ML方法在编译优化中的运用大致可以分为4个细分研究方向:(1)基于ML的编译器选项调优;(2)基于ML的优化流程;(3)基于ML的程序性能模型;(4)面向ML的编译优化框架。

将ML方法应用在编译优化中的相关研究由来已久。文献[18]总结了1990—2018年将ML应用在编译器选项调优中的研究成果。而文献[19]则从更广泛的角度介绍了截至2018年以前,ML在编译优化中的运用。然而随着深度神经网络(deep neural network,DNN)等新兴ML技术的发展,在近年来的针对编译优化的研究中也出现了许多值得关注的、使用ML方法的新工作。针对上述4个细分研究方向,精选近年来的标志性工作进行详细介绍,以期能够展示ML在编译优化中的最新应用及发展趋势,为未来的研究和创新提供启发与思考。

2.2 基于机器学习的编译器选项调优

在编译优化时,编译器通常需要选择性地执行许多不同的优化流程,才能完成对源程序的优化。然而应该选择哪些优化流程,以及如何排布这些优化流程之间的调用顺序,会对编译优化的最终效果产生不可忽视的影响。为此,编译器通常以编译器选项的形式,让编译器的调用者来指定其所要使用的优化流程及其顺序。而编译器选项调优,就是研究如何通过选择合适的编译器选项、并调整这些编译器选项的顺序,最大化编译器在某个优化目标上的优化效果。

文献[20]提出ICMC,使用ML方法提高迭代编译(iterative compilation,IC)的搜索效率,从而在较小的时间开销下获得较好的编译器选项序列。IC是编译器选项调优的基本方法之一,需要尝试使用不同的编译器选项序列对源程序进行编译优化,并测试优化后程序的性能指标,从而才能获得最佳的编译器选项序列。IC能够获得较好的编译优化效果,却伴随着巨大的时间开销。为了应对这一问题,ICMC首先对编译器选项之间进行了依赖关系分析,从而构造出备选的编译器选项子序列(简称子序列),极大程度地降低了IC的搜索空间。接着,通过对大量源程序运行IC,ICMC搜索得出这些源程序所对应的最佳子序列。将这些数据作为训练集,ICMC使用一种名为度量学习(metric learning)的ML范式,学习一种度量矩阵$\boldsymbol{M}$,将源程序的原始特征x映射为特征空间S_M中的特征向量$\boldsymbol{Mx}$。而具有相似最佳子序列的源程序,其在特征空间S_M中的特征向量之间的距离也就更近。在此基础上,当需要编译优化一个从未见过的源程序时,ICMC首先将该源程序的原始特征转换为特征空间S_M中的特征向量,然后利用K近邻算法找出S_M中与该源程序距离最近的K个源程序,最后根据这些临近源程序对应的最佳子序列,预测该源程序所对应的最佳子序列,从而极大程序地降低IC所需要的时间开销。

相比于ICMC使用ML方法改善IC的效率,文献[21]提出CNVP,直接使用ML模型预测给定的源程序所对应的最佳编译器选项,而不需要进行实际编译。与ICMC类似,如图9所示,CNVP首先构造具有优化效果的编译器选项子序列,从而降低搜索最优编译器选项的复杂度。为此,CNVP随机构造了数量庞大的编译器选项子序列,并依次进行多个源程序的编译优化,从而在这些随机构造的子序列中找到k个具有实际优化效果的子序列作为备选的核心集合。接着,CNVP将上述源程序以及所对应的最佳子序列作为训练集,训练一个深度神经网络模型,使具有根据输入的源程序直接预测该源程序对应的最佳子序列的能力。该模型由2个部分构成:其前端是对输入源程序进行分析处理的GNN,将源程序转换为紧凑且大小一致的嵌入向量;其后端是一个名为标准化值预测模型,根据前端生成的嵌入向量生成不同子序列的概率分布。

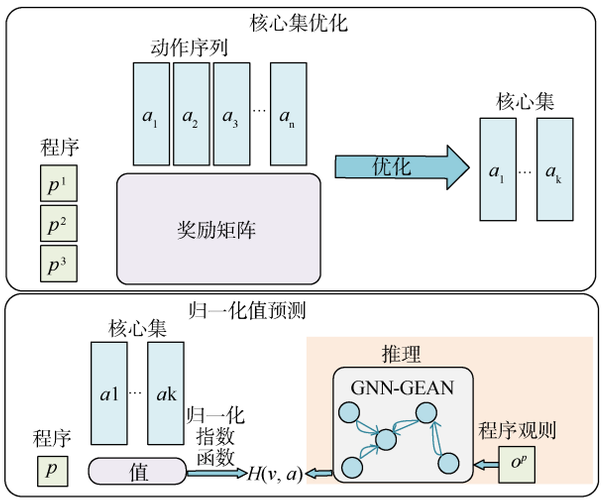


图9　CNVP主要贡献点[21]

最后,CNVP使用概率值最大的几个子序列作为最终预测出的最佳子序列。

ICMC和CNVP分别使用度量学习和GNN这2种不同的ML方法来表征源程序的特征,而文献[22]则提出LLMCO,使用大模型(large language model,LLM)根据未经优化的源程序的中间代码,预测出能够产生最好优化效果的编译器选项序列。相比于使用其他ML模型的源程序特征表示方法,LLM的优势在于可以直接以源程序的中间表示作为输入,最大化源程序信息的保留与利用。在这一思想的基础上,LLMCO使用IC方法搜索得到多个源程序的最佳编译器选项序列,然后将这些程序的中间代码和对应的最佳序列作为训练数据,有监督地训练了一个7B参数量的LLM。此外,还构造了包括预测优化后的中间代码大小、根据给定的编译器直接生成优化后的中间代码等辅助任务,从而深化了LLM对编译器内部原理的理解与拟合效果。相较于CNVP,LLMCO能够预测出优化效果更好的最佳编译器选项序列,充分展现了LLM在编译优化领域的巨大潜力。同时,作为第一个运用LLM的编译优化方法,LLMCO却也表现出了包括模型开销过大、模型输入长度受限等LLM相关的缺点。

2.3　基于机器学习的流程优化

基于ML的编译器选项调优是通过选择不同的优化流程并排布其间的调用顺序以提高编译优化的效果。而基于ML的优化流程则更进一步地使用ML方法来提高某一个优化流程其本身的编译优化效果。

文献[23]使用ML方法构建了一种端到端的循环矢量化方法,能够为给定程序中的循环生成最优的矢量化系数,从而提高编译器中循环矢量化流程的优化效果。循环矢量化流程是编译器中通过使用单指令多数据(single instruction multiple data,SIMD)指令而减少循环次数的优化流程。而循环矢量化流程优化效果主要取决于其对应的矢量化系数的取值是否合适——这一系数决定了对该循环进行矢量化操作时,每次同时处理的数据元素的数量。为了确定该系数的最佳取值,首先使用一个名为code2vec[24]的深度神经网络生成表征循环特征信息的嵌入向量,然后由一个深度RL智能体根据这个嵌入向量,对程序中的每一个循环赋予合适的矢量化系数。

ML模型也可以用来改进某些深度学习程序的优化流程。文献[25]将GNN与遗传算法相结合,改进了XLA编译器中的算子分配流程和设备内算子静态调度流程,从而最小化了神经网络计算图在硬件设备上的执行成本。具体而言,将计算图作为GNN的输入来为每个节点(也就是计算图中的算子)生成一个β分布参数,而遗传算法利用这些分布参数作为输入搜索最佳的算子分配方式和算子调度策略。通过DNN和遗传算法相结合,在较小地改动源码的前提下,取得了不错的优化效果。

2.4　基于机器学习的程序性能模型

通过使用程序性能模型,编译器可以在不进行实际编译和性能分析的前提下通过查询程序性能模型,直接判断某一个编译决策的优化效果。而相比于传统的基于人工设计的程序性能模型,基于ML的程序性能模型在预测精确度以及对不同体系结构的自适应性方面都显示出巨大的优势。

文献[26]提出使用ML方法Ithemal,根据源程序中基本块的汇编代码,预测该基本块的吞吐,从而为许多依赖于基本块吞吐的编译优化算法提供输入。基本块的吞吐是指在系统稳态循环执行某个基本块的情况下(无抢占、缓存已预热),执行该基本块所要消耗的处理器时钟周期数。基本块的吞吐和程序的总体性能息息相关,因此预测基本块的吞吐是编译器中的许多优化算法的基础。为了精确预测基本块的吞吐,Ithemal在选定的硬件平台上运行大量程序,将这些程序中的基本块的汇编代码和采集到的实际吞吐作为标注好的数据,有监督地训练了一个多层次的LSTM网络。该网络分为3个层次:第1层将输入的汇编指令中的每一个词元映

射为独热编码;第 2 层将每个指令对应的独热编码序列映射为一个单独的指令级嵌入向量;第 3 层根据基本块中的全部指令级嵌入向量预测生成该基本块的吞吐。通过使用分层 LSTM 网络,Ithemal 不但取得了相比传统方法更精确的预测效果,而且能够自适应不同的基本块大小并降低了模型推理的时间开销。

基于 ML 的程序性能模型在深度学习程序的编译优化中也有很大的应用潜力。文献[27]提出 LOTP,它使用 ML 方法设计并实现了一种面向张量程序的性能模型。这个模型可以预测张量程序在不同编译策略下(比如不同的循环次序、循环分块)张量程序的性能指标。LOTP 中所设计的性能模型分为 2 个部分,其前端使用了一种名为 TreeGRU 的循环神经网络(recurrent neural network,RNN)对 tensor 程序的抽象代码树进行了编码,从而生成了大小一致且紧凑的嵌入向量。其后端使用线性网络模型根据这些嵌入向量的值预测输入张量程序的性能指标。LOTP 还设计了一个在线学习的框架,如图 10 所示,在迭代编译与实际性能分析的过程中,逐步优化模型的预测效果。此外,LOTP 还探索了迁移学习在程序性能模型中的应用。具体而言,LOTP 将性能模型划分为 2 类,一类是与特定张量程序高度相关的本地模型,另一类是与特定张量程序无关的全局模型。当预测全新且未见过的张量程序的性能指标时,可以使用程序无关的全局模型进行初步预测,而不必等待归属于该程序的局部性能模型构建完毕,从而加速了 LOTP 在线学习框架对该程序的最佳编译优化策略的探索。

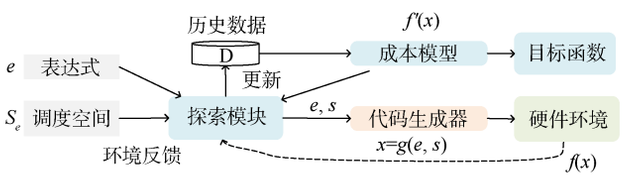


图 10　学习优化张量程序框架[27]

2.5　面向 ML 的编译优化框架

利用 ML 技术进行编译优化已经在学术界得到了广泛的研究,然而,在工业界中还未将 ML 应用于编译优化。因此文献[28]提出 MLGO 实用框架,通过在编译器中嵌入经过训练的模型,把 LLVM 中使用启发式的优化转换成模型预测。MLGO 框架将编译器的使用与训练分开,在使用时,编译器中嵌入的经过训练的模型用于做出以前由手动启发式处理的决策(内联);在训练过程中,内联器生成一个记录内联过程的日志(特征、决策等),这些日志被收集并馈送给训练算法,以生成一个新的模型。

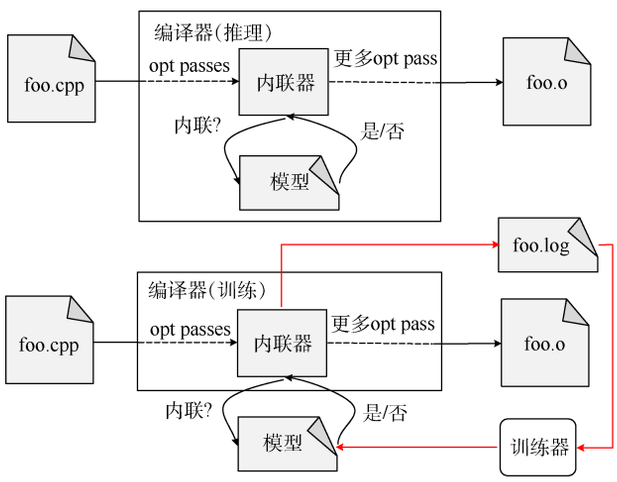


图 11　MLGO 概述[28]

人们对使用 ML 模型促进编译器优化越来越感兴趣,但是编译器和 ML 框架之间的交互仍然具有挑战性,一些优化需要紧密耦合模型和编译器内部构件。为此,文献[29]提出 ML-Compiler-Bridge,允许在传统 Python 框架内开发 ML 模型的库,该库通过提供一套通信方法(称为模型运行器)以及相关的序列化和反序列化机制,来桥接编译器和 ML 模型。如图 12 所示,模型运行器获取序列化的数据,将其写入通信通道,查询模型,并对从模型接收到的输出进行反序列化,将模型输出的数据格式转化为编译器可以处理的格式,编译器执行此决策。这些方法和机制可以适用于各种场景,并且支持进程间和进程内通信。进程间通信有效支持训练,进程内通信有效支持推理。

现代芯片优化带来了更高效的执行时间,但编译算子却需要非常长的时间,这就使得一些技术,比如自动调优,变得不太实际。因此文献[30]提出 LoopStack,一个面向 ML 的编译器栈,包括 LoopTool 和 LoopNest 2 个组件。前端 LoopTool 是一个嵌入在 Python 中的领域特定语言,用于描述神经网络工作负载的计算。LoopTool 将计算描述转化为一个由数据流图(data flow graph,DFG)组成的中间表示,DFG 描述了计算过程、内存布局和执行方式。后端 LoopNest 用来生成机器码,LoopNest 提供了一个应用程序编程接口,允许用户定义特定的循环嵌套顺序,以确定计算应该如何执行。

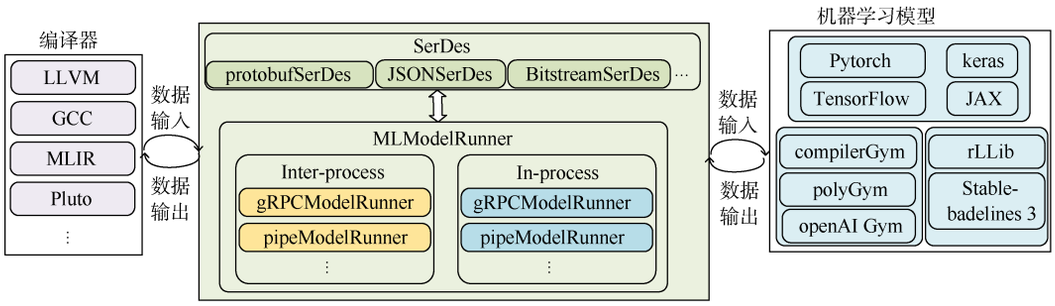


图12 机器学习与编译器连接的桥梁概述[29]

3 挑战及潜在研究方向

本节将讨论ML技术在计算机体系结构和系统中的局限性和潜力,这些技术跨越了数据、算法、实现和目标的整个开发和部署。ML技术的应用还可以成为硬件敏捷开发的推动力量。

3.1 学习型操作系统方面

ML方法为构建操作系统带来了新的机遇。为充分利用硬件资源支撑新型应用,系统的设计与实现需要不断改进与演化,以适应不同场景的需求。ML方法具有从数据中提取规律并自动优化系统性能的潜力[1]。然而,使用ML方法赋能系统软件面临一些挑战,包括设计面向系统的定制化模型、获取足量且高质量的训练数据、降低模型开销对系统性能的影响,以及消除模型误差对系统正确性的影响等。本小节将重点讨论ML技术在操作系统中的挑战和潜在方向。

(1)针对特定场景和问题开发特定的ML模型。最近的一些研究试图揭示ML系统的局限性。但这些研究都将基于ML的解决方案视为“黑匣子”,特别是在具有多个隐藏层的DNN中,无法充分解释这些复杂模型和架构在决策过程中的逻辑。此外,操作系统的功能主要由资源管理、内存管理与任务调度等组成,针对其复杂的系统架构和不同的调度目标或管理目标,不同的ML模型所优化的目标和效果也有显著差异,因此需要为具体应用场景设计定制化的ML模型,以有效地进行优化。

(2)消除模型预测误差对系统性能的影响。在一些关键性的应用和系统中,例如自动驾驶,始终保证系统行为的正确性至关重要。然而,ML模型产生的结果总是存在误差,难以提供精确度保证,如何尽可能减小误差,以及针对模型预测出现误差的结果,如何及时制定一个后备机制,避免出现重大事故灾难。

(3)解决训练开销大的问题。除了构建模型之外,训练时间和相关成本开销也是一大挑战。操作系统在需要处理实时性要求和并发度极高的应用场景中,对模型执行的效率和响应时间的要求显得尤为严格。这意味着系统必须能够迅速且有效地响应外部事件,同时支持多个任务或进程的高效并发执行,以确保系统的整体性能和实时响应能力。然而,由于神经网络等复杂的ML模型在执行过程中通常需要大量的计算资源和时间来执行,这会影响整个系统的性能[1]。根据《AI指数》(https://policycommons.net/artifacts/12089781/hai_ai-index-report-2024/12983534/)的估算,最新一代AI模型的训练成本已经达到前所未有的水平。例如,OpenAI的GPT-4预计使用了价值7 800万美元的计算资源进行训练,而谷歌的Gemini Ultra则耗费了1.91亿美元的计算资源成本。模型执行开销甚至可能大于使用ML方法带来的提升。另外,训练一个模型可能需要数10 d,能否开发某种技术减少模型训练时间以及成本开销?如何应用迁移学习,在类似结构但细节不同的场景中重复使用DNN?

(4)在操作系统中高效部署轻量级模型。在实时系统中需要模型快速推理,比如在自动驾驶的情况下,会使用硬件加速器,使系统能够做出关键安全决策。特别是在DNN中,模型的推理速度慢,可能会显著影响性能提升,这可能导致整体性能与传统解决方案相当。由于传统解决方案的简单性,研究者可能更倾向于传统解决方案。因此,如何在系统关键路径中部署轻量级ML模型,以尽量减少对其他操作系统组件的干扰,仍是一个值得深入探究的问题。

操作系统面向通用场景设计和实现,用户态存在大量配置项和参数,如网络配置等,内核中存在大量的启发式策略,如任务调度,缓存预取等,面向特定应用和硬件环境,性能无法实现最优,出现问

题时难以定界定位。ChatGPT及基于大模型的Agent目前已支持根据用户需求完成特定复杂任务,具有相当的智能化水平。如何利用新智能技术实现操作系统的智能化,包括性能自动调节、故障自主诊断修复等,是可以尝试的方向。

ML技术的进步以及大数据和计算资源的可用性,使得ML能够在许多领域取代过去依赖人力的工作,操作系统亦是其中领域之一。然而,将ML用于操作系统等领域仍会面临许多挑战,不仅需要同时具备系统和ML知识,还需要不断调整系统设计和ML算法,以实现最优的系统设计。但学习型操作系统是一个值得深入探索的方向,随着技术的不断进步和应用场景的不断扩展,学习型操作系统将在实际场景中发挥越来越大的作用。

3.2 学习型编译器方面

学界和工业界一致认为,新一代编译器将会是一种学习型编译器。通过使用ML技术,学习型编译器将具备有效应对计算机体系结构的日新月异变化。潜在的研究方向包括利用大语言模型驱动学习型编译器,使其能够快速支持异构算力架构的开发并自动优化。此外,学习型编译器需要具备自动解析新硬件的领域特定语言和特有指令集,促进硬件与编译的协同进化的能力。同时,AI模型将替代传统编译器在性能分析中的角色,指导如循环转换、内联、寄存器分配和指令调度等优化策略,开启编译技术的新篇章。

(1)DNN的应用。相较于传统ML方法,DNN具有出色的表征和拟合能力。因此,近年来RNN、GNN及自编码器等多种DNN在编译优化相关研究中取得了出色的表现。这与文献[19]中总结的"(2018年以前)编译优化中DNN的应用十分稀少"的情况截然不同。由此可见,在编译优化中使用DNN及其相关的模型与思想,正在成为一种明显的研究趋势。

(2)多模型协同。在尝试解决一个编译优化领域的问题时,往往可以将其划分为多个子问题。比如较为复杂的编译器选项调优问题,就可以划分成3个子问题:(1)给定一个源程序,如何根据该源程序的程序特征,将全部备选的编译器选项中挑选出可能具有优化效果的选项集合S;(2)给定一组编译器选项x,如何在不实际编译运行该源程序的前提下预测这组编译器选项的优化效果Perf_x;(3)如何利用Perf_x,高效地在S中的全部可能x的集合中搜索得到最优的编译器选项序列。这些子问题通常并不复杂,使用较为简单的模型就能充分拟合。由此可见,对于许多编译优化问题,使用多模型协同的模型架构相比于使用单一模型来说更具有优势,其不但有助于降低ML模型的总体复杂度,还能减少模型训练所需的数据量大小并提高模型的可解释性。可以说,多模型协同是当前编译优化领域ML模型架构研究的重要趋势之一。

(3)在线学习方法的使用。模型的复杂度、模型的泛化能力、模型对于特定场景下的专精能力构成了一个ML模型的不可能三角。一个ML模型,如果想要在保持其泛化能力的前提下,兼具对不同特定场景的专精能力,就需要极大地提升其模型复杂度——在DNN方面往往表现在网络的深度。而诸如强化学习这样的在线学习方法虽然牺牲了模型的泛化能力,却能够专精于特定场景下的调优效果,并且降低了对模型复杂度的要求,从而消耗更少的计算资源,并且需要更少的训练数据。在编译优化中,经常存在着一些针对某个特定场景的持续在线优化问题,针对这样的优化问题,在线学习方法的使用是当前的研究趋势之一。

(4)程序特征的表征。ML模型运用在编译优化中时,往往要基于源程序的静态、动态特征的原始特征信息生成一个程序特征的紧凑表征,即程序的嵌入向量,然后再基于这组嵌入向量训练一个执行分类或者预测任务的神经网络。因此,如何高效处理程序的原始特征以生成程序的紧凑表征就成为了影响ML模型效果的关键。ICMC使用度量学习的方法,学习一种度量矩阵$\boldsymbol{M}$,将人为选择的程序静态、动态特征映射成$\boldsymbol{M}$所决定的特征空间中的向量。CNVP则使用GNN,将源程序的中间表示直接转化为嵌入向量。而Ithemal[26]则使用层级化的LSTM对程序的汇编代码进行处理,分别生成3个层级的嵌入向量,并且将其效果与使用GNN所得到的嵌入向量进行了比较,得出"循环神经网络因为能够更好地捕捉指令之间的先后顺序而优于图神经网络"的结论。由此可见,随着源程序原始特征信息的种类以及ML方法面向的优化领域的不同,程序的最佳表征与处理方法也有所不同。文献[31]研究了2020年以前10多种不同的程序表征方法在多个预测性编译领域中的应用,得出"面向不同编译优化领域所设计的程序表征方法很难迁移应用到其他编译优化领域之中"的结论。综上所

述,虽然在诸如自然语言处理这样的研究领域中能够充分表征自然语言的语义和语言结构特征的预训练模型已经十分成熟,在编译优化领域中设计一种类似于Bert[32]的可以预训练的程序特征表征方法依旧是一个充满挑战的研究趋势。

(5)大模型辅助编译器。大模型的出现也对编译器领域产生深远影响,不仅提升了编译器的智能化水平,使之能够更深入地理解代码逻辑和语义,以进行更有效的优化,还显著增强了编译器的跨平台能力,使其能够轻松适应日益增长的异构计算架构。此外,大模型通过提供更全面的语义和逻辑检查,进一步拓展了编译器的优化空间,极大地提升了编译器的性能。这些影响共同推动了编译器技术的进步,使其能够更好地应对当前及未来的挑战。

4 总 结

本文回顾了ML在计算机系统中的应用,特别是在学习型操作系统和学习型编译器方面的进展。这些应用展示了ML如何帮助系统从海量数据中提取有用信息,动态优化性能,并提升用户体验。然而,ML赋能系统软件的过程并非一帆风顺,面临着数据质量、模型泛化能力、计算资源消耗以及安全性等多方面的挑战。为了克服这些挑战,未来的研究需要更加深入地探索ML算法与系统软件设计的融合机制,发展出更加高效、智能的系统软件解决方案。

参 考 文 献

[1] 唐楚哲, 王肇国, 陈海波. 机器学习方法赋能系统软件:挑战、实践与展望[J]. 计算机研究与发展, 2023, 60(5): 964-973.

[2] ZHANG Y Y, HUANG Y T. “Learned” operating systems[J]. ACM SIGOPS Operating Systems Review, 2019, 53(1): 40-45.

[3] AKGUN I U, AYDIN A S, ZADOK E. KMLIB: towards machine learning for operating systems[EB/OL].[2024-04-26]. https://www.fsl.cs.stonybrook.edu/~umit/files/kmllib-sysml-paper.pdf.

[4] FINGLER H, TARTE I, YU H, et al. Towards a machine learning-assisted kernel with lake[EB/OL]. [2024-04-26]. https://https://dl.acm.org/doi/pdf/10.1145/3575693.3575697.

[5] QIU Y, LIU H, ANDERSON T, et al. Toward reconfigurable kernel datapaths with learned optimizations[EB/OL]. [2024-04-26]. https://dl.acm.org/doi/10.1145/3458336.3465288

[6] MAO H Z, SCHWARZKOPF M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters[EB/OL]. [2024-04-26]. https://arxiv.org/abs/1810.01963v2.

[7] MESHGI K, MIRZAEI M S, SEKINE S. Q-learning scheduler for multi task learning through the use of histogram of task uncertainty[EB/OL]. [2024-04-26]. https://www.researchgate.net/publication/361058997.

[8] MIRHOSEINI A, PHAM H, LE Q V, et al. Device placement optimization with reinforcement learning[EB/OL].[2024-04-26]. https://arxiv.org/pdf/1706.04972.pdf.

[9] NEGI A, KUMAR P K. Applying machine learning techniques to improve linux process scheduling[EB/OL]. [2024-04-26]. https://ieeexplore. ieee. org/document/4085157.

[10] CHEN J, BANERJEE S S, KALBARCZYK Z T, et al. Machine learning for load balancing in the linux kernel [EB/OL]. [2024-04-26]. https://dl. acm. org/doi/epdf/10.1145/3409963.3410492.

[11] GOODARZY S. SMARTOS: automating allocation of operating system Resources to User Preferences via Reinforcement Learning[D]. American: University of Colorado, 2022.

[12] ZHANG Y, BHARGAVA B. Self-learning disk scheduling[J]. IEEE Transactions on Knowledge and Data Engineering, 2008, 21(1): 50-65.

[13] CORTEZ E, BONDE A, MUZIO A, et al. Resource central: understanding and predicting workloads for improved resource management in large cloud platforms [EB/OL].[2024-04-26]. https://api.semanticscholar.org/CorpusID:207600624.

[14] WANG Y, ARYA K, KOGIAS M, et al. Smartharvest: Harvesting idle cpus safely and efficiently in the cloud [EB/OL]. [2024-04-26]. https://dl. acm. org/doi/abs/10.1145/3447786.3456225.

[15] MASOUROS D, XYDIS S, SOUDRIS D. Rusty: runtime system predictability leveraging LSTM neural networks[J]. IEEE Computer Architecture Letters, 2019, 18(2): 103-106.

[16] MAHAPATRA R, AHN B H, WANG S T, et al. Exploring efficient ML-based scheduler for microservices in heterogenous clusters[EB/OL].[2024-04-26]. https://cseweb. ucsd. edu/~bhahn221/doc/paper/iscaw22-octopus.pdf.

[17] LIU L, DOU X, CHEN Y. Intelligent resource scheduling for co-located latency-critical services: a multi-model collaborative learning approach[EB/OL].

[2024-04-26]. https://www.usenix.org/conference/fast23/presentation/liu.

[18] ASHOURI A H, KILLIAN W, CAVAZOS J, et al. A survey on compiler autotuning using machine learning [J]. ACM Computing Surveys, 2018, 51(5): 1-42.

[19] WANG Z, O'BOYLE M. Machine learning in compiler optimization [J]. Proceedings of the IEEE, 2018, 106 (11): 1879-1901.

[20] LIU H, LUO J, LI Y, et al. Iterative compilation optimization based on metric learning and collaborative filtering [J]. ACM Transactions on Architecture and Code Optimization, 2021, 19(1): 1-25.

[21] LIANG Y, STONE K, SHAMELI A, et al. Learning compiler pass orders using coreset and normalized value prediction [EB/OL]. [2024-04-26]. https://dl. acm. org/doi/10.5555/3618408.3619263.

[22] CUMMINS C, SEEKER V, GRUBISIC D, et al. Large language models for compiler optimization [EB/OL]. [2024-04-26]. https://arxiv.org/abs/2309.07062.

[23] HAJ-ALI A, AHMED N K, WILLKE T, et al. Neurovectorizer: end-to-end vectorization with deep reinforcement learning [EB/OL]. [2024-04-26]. https://dl.acm.org/doi/10.1145/3368826.3377928.

[24] ALON U, ZILBERSTEIN M, LEVY O, et al. Code2vec: learning distributed representations of code [J]. Proceedings of the ACM on Programming Languages, 2019, 3: 1-29.

[25] PALIWAL A, GIMENO F, NAIR V, et al. Zero-shot learning for fast optimization of computation graphs [EB/OL]. [2024-04-26]. https://nips.cc/virtual/2019/14800.

[26] MENDIS C, RENDA A, AMARASINGHE S, et al. Ithemal: accurate, portable and fast basic block throughput estimation using deep neural networks [EB/OL]. [2024-04-26]. https://arxiv.org/pdf/1808.07412v1.

[27] CHEN T Q, ZHENG L M, YAN E, et al. Learning to optimize tensor programs [EB/OL]. [2024-04-26]. https://arxiv.org/abs/1805.08166.

[28] TROFIN M, QIAN Y, BREVDO E, et al. Mlgo: a machine learning guided compiler optimizations framework [EB/OL]. [2024-04-26]. https://arxiv. org/abs/2101.04808.

[29] VENKATAKEERTHY S, JAIN S, KALVAKUNTLA U, et al. The next 700 ML-enabled compiler optimizations [EB/OL]. [2024-04-26]. https://dl. acm. org/doi/abs/10.1145/3640537.3641580.

[30] WASTI B, GRUBISIC D, STEINER B, et al. LoopStack: ML-friendly ML compiler stack [EB/OL]. [2024-04-26]. https://nips.cc/virtual/2022/62272.

[31] DA SILVA A F, BORIN E, PEREIRA F M Q, et al. Program representations for predictive compilation: State of affairs in the early 20's [J]. Journal of Computer Languages, 2022, 73: 101171.

[32] DEVLIN J, CHANG M W, LEE K, et al. Bert: pre-training of deep bidirectional transformers for language understanding [EB/OL]. [2024-04-26]. https://arxiv. org/abs/1810.04805.

(责任编辑:马田田)

勘误声明

本刊对发表在2024年45卷4期42-47页的"拟南芥损伤诱导的钙信号传递模式的研究"论文进行勘误。勘误内容:45页右栏2.4小节第16行"L4叶片大约在1 000 s达到最大值"更正为"L4叶片大约在100 s达到最大值"。