基于强化学习的泛在操作系统内存布局优化框架

沙乐天[1,2],陈 霄[2,3*],郑红美[1],潘家晔[1],董建阔[1,2],肖 甫[1,2]
(1. 南京邮电大学计算机学院、软件学院、网络空间安全学院,江苏南京 210023;2. 江苏省物联网智能感知与计算重点实验室,江苏南京 210023;3. 南京邮电大学教育科学与技术学院,江苏南京 210023)

摘 要: 随着人机物三元融合时代的到来,泛在操作系统(Ubiquitous Operating System,UOS)对资源受限且负载多变的边缘节点内存管理提出了弹性与自适应的严苛要求。然而,现有的内存管理策略大多依赖预设的静态规则,难以感知运行时的动态负载特征,导致系统在面对突发性、多模态任务时,常出现严重的内存碎片化与实时响应性能下降,成为制约泛在计算能力落地的关键瓶颈。针对上述挑战,本文提出一种基于深度强化学习的内存布局优化框架AIMO。该框架旨在通过"感知-决策-执行"的闭环机制,实现对内存资源的智能动态管理。首先,多维状态感知模块通过处理器性能监控单元(Performance Monitoring Unit,PMU)和内核插桩,实时捕捉硬件层指标(如缓存未命中率、介质损耗)、对象层行为(如访问频率、冷热度)以及系统层上下文(如内存碎片率、应用场景类型),并将其编码为高维状态向量。其次,强化学习智能决策模块将内存布局问题建模为马尔可夫决策过程(Markov Decision Process,MDP),利用近端策略优化(Proximal Policy Optimization,PPO)算法在线生成最优决策。为了兼顾嵌入式环境的资源约束,该模块采用了"离线预训练与轻量化在线微调"的策略,在保证决策确定性的同时大幅降低了计算开销。最后,策略执行模块通过在内核态设置轻量级钩子函数,实现了分配拦截、对象重定位、策略驱动的对象放置及主动碎片整理等机制,完成了对内存布局的闭环自适应控制。本文在OpenHarmony内核中实现了AIMO框架,并基于Raspberry Pi Zero 2W平台进行了实证评估。实验结果表明,与实时性能标杆TLSF相比,AIMO在维持同等水平最坏情况分配时间的同时,将大块内存分配失败率从3.4%显著降低至0.9%,最大可用连续空闲块从318 KB提升至476 KB,并使上层业务的预加载延迟减少了15.8%。此外,该框架表现出极高的轻量化特性,其决策模块的平均CPU占用率仅为1.9%,元数据存储开销相对于512 MB系统总内存占比不足0.001%。本研究证实了强化学习在操作系统内核内存管理中的有效性,为构建高效、自适应的泛在操作系统内核提供了新思路。

关键词: 人机物融合;泛在操作系统;深度强化学习;内存布局;性能优化
基金项目: 国家自然科学基金(No.62572255,No.62302238);江苏省2024前沿技术研发计划(No.BF2024071)
中图分类号: TP316.4 **文献标识码:** A **文章编号:** 0372-2112(2026)04-1612-17
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250835

A Reinforcement Learning-Based Framework for Memory Layout Optimization in Ubiquitous Operating Systems

SHA Letian[1,2], CHEN Xiao[2,3*], ZHENG Hongmei[1], PAN Jiaye[1], DONG Jiankuo[1,2], XIAO Fu[1,2]
(1. *School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu* 210023, *China*;
2. *Jiangsu Key Laboratory of Intelligent Sensing and Computing for Internet of Things, Nanjing, Jiangsu* 210023, *China*;
3. *School of Education Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu* 210023, *China*)

Abstract: With the advent of the era of human-cyber-physical ternary integration, ubiquitous operating systems (UOS) impose stringent requirements for elasticity and adaptability on memory management within resource-constrained edge nodes handling variable workloads. However, most existing memory management strategies rely on predefined static rules and struggle to perceive dynamic workload characteristics at runtime. Consequently, when facing bursty and multi-modal tasks, systems often suffer from severe memory fragmentation and degraded real-time response performance, which has become a critical bottleneck restricting the practical deployment of ubiquitous computing capabilities. To address these challenges, this paper proposes AIMO, a memory layout optimization framework based on deep reinforcement learning. This framework aims to achieve intelligent and dynamic management of memory resources through a closed-loop mechanism of "perception-decision-execution". First, a multi-dimensional state perception module captures hardware-level metrics (e.g., cache miss rate, media wear), object-level behaviors (e.g., access frequency, coldness/hotness), and system-level context (e.g., memory fragmentation rate, application scenario types) in real-time via the processor's performance monitor-

收稿日期:2025-09-24;录用日期:2026-03-05;责任编辑:覃怀银
*通信作者:陈霄

ing unit (PMU) and kernel instrumentation, encoding them into high-dimensional state vectors. Second, a reinforcement learning intelligent decision module models the memory layout problem as a markov decision process (MDP) and utilizes the proximal policy optimization (PPO) algorithm to generate optimal decisions online. To accommodate the resource constraints of embedded environments, this module adopts a strategy of "offline pre-training and lightweight online fine-tuning", which significantly reduces computational overhead while ensuring decision determinism. Finally, a policy execution module implements mechanisms such as allocation interception, object relocation, policy-driven object placement, and proactive defragmentation by establishing lightweight hook functions in the kernel space, thereby completing the closed-loop adaptive control of the memory layout. We implemented the AIMO framework in the OpenHarmony kernel and conducted an empirical evaluation based on the Raspberry Pi Zero 2W platform. Experimental results demonstrate that, compared to the real-time performance benchmark TLSF, AIMO significantly reduces the large-block memory allocation failure rate from 3.4% to 0.9% and increases the maximum available contiguous free block from 318 KB to 476 KB, while maintaining an equivalent worst-case allocation time. Furthermore, it reduces the preloading latency of upper-layer applications by 15.8%. In addition, the framework exhibits a highly lightweight nature; the average CPU utilization of its decision module is only 1.9%, and the metadata storage overhead accounts for less than 0.001% of the 512 MB total system memory. This study verifies the effectiveness of reinforcement learning in operating system kernel memory management, providing a novel approach for constructing efficient and adaptive UOS kernels.

Keywords: human-cyber-physical fusion; ubiquitous operating system; deep reinforcement learning; memory layout; performance optimization

Foundation Item(s): National Natural Science Foundation of China (No.62572255, No.62302238); The 2024 Frontier Technology Research and Development Program of Jiangsu (No.BF2024071)

0　引言

随着互联网、人工智能等技术浪潮的蓬勃发展,信息世界、物理世界和人类社会的深度融合正加速形成"人机物三元融合"的泛在计算新时代。在这一创新范式下,"应用场景即是计算机"成为软件发展的重要趋势,但也带来了前所未有的复杂挑战:如何面向多元化的人机物融合应用场景,基于海量且泛在化的软硬件异构资源,实现应用软件系统的高效、高质量开发与动态演化。为系统性应对这一复杂性,学术界与产业界共同提出了"泛在操作系统"(Ubiquitous Operating System,UOS)这一前瞻性理念[1]。UOS旨在以"系统之系统、自底向上抽象"的视角,通过构建能够汇聚、调度异构资源的"场景计算机",为上层应用提供统一且强大的支撑。其中,实现"软件定义资源系统的弹性动态管理"是UOS亟需解决的关键技术问题,要求UOS能灵活定义和抽象各类资源,并根据不断变化的场景需求,对计算、存储、网络等核心资源进行智能、高效的调度与管理。

在人机物融合的体系中,直接与物理世界深度交互、负责实时感知与精准控制的计算节点,构成了整个泛在操作系统的"神经末梢",其性能与稳定性是人机物融合成功的核心保障。这些通常以嵌入式设备或边缘节点形态存在的计算单元,是"泛在化异构资源"最为集中、也最为复杂的体现。它们往往硬件资源受限,却需要承载来自物理世界的、高度动态和不确定的任务负载[2]。因此,UOS能否真正实现对资源的"弹性动态管理",其成败的关键就在于这些计算节点。

在这些对性能和实时性要求极为严苛的边缘嵌入式节点上,内存资源是所有计算任务得以执行的基石。作为操作系统最核心的基础功能之一,内存管理的效率和稳定性直接决定了上层人机物融合应用的整体表现。传统内存管理策略的静态性、被动性以及对异构环境适应性不足等问题,使其难以有效应对此类场景下内存使用的动态性、碎片化和严格实时性需求。这种局限性不仅导致了资源利用率低下,更严重阻碍了应用性能的提升与系统的可靠运行,从而成为"场景计算机"理念落地与泛在应用部署的重大障碍。

针对泛在计算环境下内存管理挑战,学术界已从泛在操作系统理念构建与资源抽象[3-4],以及具体内存优化技术[5-7]等多个维度开展研究。然而,现有方案在面对人机物融合场景的极端动态性与资源受限性时,仍存在较大局限。大多数研究依赖静态配置或固定策略,导致缺乏对运行时负载的自适应能力。并且,跨设备协同机制薄弱,难以实现统一管理。尤其在边缘嵌入式节点上,内存请求的突发性、多模态特征,使得传统或改进的内存管理机制在碎片控制、实时响应与能效平衡方面仍然存在不足。虽然以大模型为代表的新兴人工智能技术为复杂环境下的内存资源管理与布局优化提供了新路径,但其高昂的计算开销与推理时延,使其难以直接应用于对实时性与确定性要求极高的内核级内存布局优化。

为解决上述问题,本文提出一种基于轻量化强化学习的内存布局优化框架 AIMO,旨在为泛在操作系统提供一种实现内存资源“弹性动态管理”的创新技术路径。具体来说,AIMO 通过细粒度感知硬件性能、内存对象行为及系统全局状态,构建全面的内存状态全局视图。在此基础上,将内存布局优化问题建模为马尔可夫决策过程,并使用深度强化学习(Deep Reinforcement Learning,DRL)进行智能求解,以动态生成最优的内存布局策略。最终,框架通过轻量级内核扩展与协同机制,能够实时拦截、重定向并优化内存请求,从而实现对内存布局的自主性、自适应性智能管理,为泛在操作系统在关键节点上实现高性能、高效率的内存资源管理提供强有力的技术支撑。本文的主要贡献包括以下几个方面。

(1)创新性地将深度强化学习引入泛在操作系统嵌入式计算节点的内存布局优化领域,提出了一种自主智能内存优化框架 AIMO,为泛在操作系统在动态、异构环境下实现内存资源的“弹性动态管理”提供了可行的思路。

(2)构建了一套集多维内存状态感知、深度强化学习智能决策与内存布局策略执行于一体的闭环优化机制,通过细粒度感知、DRL 策略生成与轻量级内核扩展,有效实现了内存请求的实时优化。

(3)在典型的人机物融合应用场景下进行了充分的实证评估,结果表明,框架在应对突发性、多模态内存请求时,能够显著减少内存碎片化,大幅提升内存实时响应效率,并在能效平衡方面取得显著改进,验证了框架的有效性与优越性。

1 相关工作

1.1 泛在计算环境下的内存管理挑战

随着“人机物三元融合”的泛在计算环境逐步兴起,运行于高度异构的资源环境已成为应用软件系统的新需求,这种跨场景的趋势对操作系统中以内存为代表的关键资源的调度机制等核心能力,提出了前所未有的挑战。针对 Linux 系统、微内核架构等单一系统的内存管理研究已有一定进展[8-9],但是基于静态、同构的传统操作系统,难以适应资源高度异构、动态变化且普遍受限的泛在计算环境。为此,一系列研究从推动面向人机物交互融合场景的泛在操作系统成型,到泛在计算环境下的资源调度新方案,以不同角度探讨了泛在计算环境下内存管理的新路径。梅宏等人[1]首次提出了 UOS,系统性地阐述了面向人机物三元融合的新型操作系统范式。他们从“软件定义”的视角出发,提出将物理内存资源虚拟化为逻辑资源池,支持上层应用以编程方式定义内存服务质量需求,从而实现内存资源的灵活配置与策略可编程化,为构建智能化内存管理奠定了基础[3]。这一思想与 Liu 等人[4]提出的 ServiceOS 高度呼应,后者同样将包括内存在内的系统资源解耦为可动态组合的服务单元,并引入机器学习驱动的资源调度机制,能够根据运行时负载自动调整内存分配策略,这种学习驱动的调度模式为实现泛在计算环境下的自优化内存管理提供了新思路。进一步地,范晓鹏等人[10]提出 UOS 的结构化存储机制,为泛在计算环境中异构数据的高效数据定位与内存映射提供了支持。此外,融合动态随机存储器(Dynamic Random Access Memory,DRAM)与新型非易失存储器(Non-Volatile Memory,NVM)的异构内存系统,新型异构内存控制器[5]以及混合内存页面管理机制[6]的设计,体现出当下研究针对异构内存管理挑战的思考。

此外,面向资源受限设备的内存管理优化也受到研究者的广泛关注。Cao 等人[2]设计的 XiUOS 作为面向工业物联场景的泛在操作系统,针对嵌入式设备内存紧张的特点,采用微内核架构与轻量级内存池机制,提升了系统在高并发场景下的内存稳定性。与之互补,Tripathi 等人[7]则在数据结构层面上优化内存访问效率,提出鲁棒的左-右哈希方案,在有限内存条件下降低哈希冲突与内存浪费,为泛在系统中高频内存查询操作提供了轻量化支持。Lim 等人[11]虽聚焦虚拟化集群架构,但其提出的“简化轮换过程”与“动态插入备用服务器”策略,本质上是对系统级内存调度与上下文切换开销的优化,间接缓解了因内存资源争用导致的服务延迟问题。

然而,尽管上述研究从服务化调度、数据结构优化等多个维度推进了泛在环境下的内存管理发展,现有方案仍面临诸多局限。传统机制依赖静态配置,难以自适应动态负载。跨设备内存协同不足,限制了资源池化。同时,在嵌入式终端上,轻量化与高性能难以兼顾。尤其在人机物深度交互场景中,面对突发性、多模态内存请求,传统方案在碎片控制、实时响应及能效平衡方面均显不足。因此,如何在泛在操作系统架构下,构建兼具自适应性、跨平台协同能力与低开销特性的智能化内存管理体系,仍是目前面向人机物融合场景下亟待突破的关键问题。

1.2 人工智能与大模型驱动的内存管理现状

近年来,大语言模型(Large Language Model,LLM)与大数据技术的爆发式增长,推动学术界开始探索人工智能与内存存储管理的融合,以实现自适应资源调度。相关研究[12]指出,将机器学习模块集成到相关系统中,通过预测模型实现动态资源分配与缓存主动预取,可以在不损害数据完整性的前提下提升实时分

析速度。在具体的内存分配机制创新方面,PagedAttention[13]借鉴操作系统分页思想,将KV缓存(Key-Value Cache)逻辑划分为非连续物理块,有效解决了输出序列长度不可知导致的内存浪费与碎片化问题。这一设计使相关系统的吞吐量提升数倍,凸显了内存布局优化在异构资源利用中的核心作用。此外,研究人员[14]对KV缓存管理进行了系统级分类,强调硬件感知调度对缓解长上下文内存压力的重要参考价值。

针对深度神经网络训练阶段的内存极度受限问题,学术界聚焦于张量布局与生命周期管理。例如,MEMO框架[15]针对超长上下文训练,利用双层混合整数规划优化层间内存重用,并实现Token粒度的激活值在GPU与主机内存间的动态换入换出。MegTaiChi[16]则通过实时追踪张量访问模式,协同张量分区与重计算,在受限预算下显著扩大训练批次。此外,为解决分布式环境下的最优解难题,Mist框架[17]构建了符号分析模型,利用混合整数线性规划协同优化内存足迹与并行策略,消除了跨节点通信瓶颈。

综上,基于人工智能与复杂数学建模的内存优化方案在服务器端与大模型领域取得了较多成果,但将其直接应用于泛在操作系统的边缘计算节点时,仍面临严峻的工程挑战。MEMO[15]和Mist[17]中采用的MILP或MIP求解算法属于典型的NP-hard问题,其计算复杂度随决策空间扩大呈指数级增长,难以满足泛在计算环境下微秒级的实时响应需求。其次,大规模语言模型或复杂的符号分析器本身具有庞大的内存占用与算力需求,这与边缘节点资源受限、能效敏感的特征产生了本质冲突。更为关键的是,泛在环境下的内存请求具有极强的突发性,要求管理机制必须具备极低的介入开销与极高的确定性。因此,对于需要在内核级实现细粒度、高频次反馈的内存布局优化任务,亟需寻找一种在智能化程度与计算开销之间达成更佳平衡的技术方案。这使得研究者的目光逐步从侧重宏观配置决策的大模型,转向具备更轻量化推理能力与实时自适应特征的机器学习分支。

1.3　强化学习在内存管理中的应用

与侧重于宏观配置决策的大模型不同,强化学习(Reinforcement Learning,RL)因其具备轻量化推理、实时交互优化以及对动态环境的强自适应能力,成为解决内核级细粒度资源管理问题的核心技术路径,在内存分配与调度方面展现出显著潜力[18-22]。在动态内存分配与调度方面,Lim等人[18]针对first-fit、best-fit等传统内存分配算法在动态负载下易产生碎片、适应性差的问题,提出了一种基于强化学习的动态内存分配框架,通过训练智能体与系统环境交互优化分配策略,提升内存利用率。类似地,Jia等人[19]通过Q-learning智能体动态调整Apache Spark JVM堆内存的划分,减少垃圾回收时间,从而提升应用性能。针对虚拟化环境,Garrido等人[20]提出CAVMem机制,利用连续动作强化学习实现虚拟机间的内存动态分配,在性能提升的同时降低了训练开销。而在工业物联网边缘计算场景下,Kumar等人[21]结合深度Q网络、BiLSTM和量子遗传算法优化边缘PLC系统中的内存分配,旨在降低数据丢失概率。此外,Wang等人[22]借鉴强化学习在网络交换机缓冲区管理中的应用,通过深度强化学习自动学习缓冲区管理策略,提升了共享内存缓冲区的效率。

在多层异构内存数据放置与迁移领域,强化学习也展示出巨大优势[23-29]。近年提出的ArtMem框架[23]通过强化学习动态管理分层内存系统,自适应工作负载演进以优化内存页放置和迁移。IDT系统[24]利用强化学习自动调优多层主存的数据降级策略,结合精细粒度的内存访问监控,实现了冷热数据的高效分层放置。进一步地,Karimov等人[25]聚焦于DRAM/NVM混合内存等异构内存中的页面初始分配问题,提出一种与技术无关的强化学习方法,通过奖励机制引导智能体学习最优数据放置策略,显著提升了工作负载性能和DRAM命中率。对于非易失性存储,Yoo等人[26]提出一种基于强化学习的SLC缓存管理技术,以动态调整参数来提升QLC NAND SSD的写入性能。而在混合缓存架构中,Huang等人[27]和范浩等人[28]分别利用Q-learning优化了混合缓存的性能与能耗,并通过强化学习来学习高效的数据分配策略。Xie等人[29]综合利用强化学习等机器学习算法,优化缓存替换、动态缓存大小调整及存储层选择,提高缓存命中率。

在底层内存机制与硬件协同优化方面,强化学习也提供了新的解决方案[30-34]。作为一种自适应内存控制器,Flatfish利用强化学习,基于内存访问模式生成应用感知的地址映射方案,以提升DRAM性能[30]。Yang等人[31]提出一种基于在线强化学习的缓存管理器,用于芯片系统中自适应调整LLC缓存策略,有效减少了缓存一致性失效。FAPM机制基于强化学习动态调整预取器的激活和预取程度,旨在缓解多核系统中的预取器竞争和公平性问题[32]。而在页面替换策略方面,LPR作为一种新型页面替换方案[33],通过自学习应用内存访问模式,在线选择最佳页面替换策略。McCore架构[34]则通过硬件重组内存层次结构并引入基于硬件实现的强化学习调度器,在异构多核系统中实现缓存分区和细粒度访问控制的优化。

尽管上述研究展示了强化学习在传统和新兴内存管理场景中的巨大潜力,然而,将强化学习应用于人机物融合场景下泛在操作系统的内存布局优化,仍

存在诸多挑战[35]。由于泛在计算环境，尤其是低功耗嵌入式设备上资源的高度异构性、受限性以及频繁读写操作带来的动态性，传统的集中式或高开销强化学习模型难以直接部署。并且，泛在计算环境中状态转移模式复杂多变，可能导致强化学习模型收敛困难，策略泛化能力不足，这为设计高效且可行的强化学习驱动内存优化框架带来了新的难题。

2 框架设计

为实现对泛在操作系统中关键嵌入计算节点内存资源的弹性动态管理，本文提出了 AIMO 框架。AIMO 通过感知系统的多维度状态，利用决策核心生成最优策略，并通过执行接口将策略付诸实施，最终形成一个“感知-决策-执行”的自适应闭环，最大化地适应泛在操作系统的动态内存分配需求。

2.1 框架总览

AIMO 的整体架构如图 1 所示。它位于应用程序与底层内存分配器之间，作为内核中的增强模块运行。该框架由以下三大核心模块构成：(1)多维内存状态感知模块，采集硬件、对象及系统三个层次的多维信息，并编码为高维状态向量；(2)强化学习智能决策模块，接收状态向量，基于 DRL 生成最优动作，实现智能化决策；(3)内存布局策略执行模块。将抽象决策转化为对内存分配器的具体调用或直接操作。

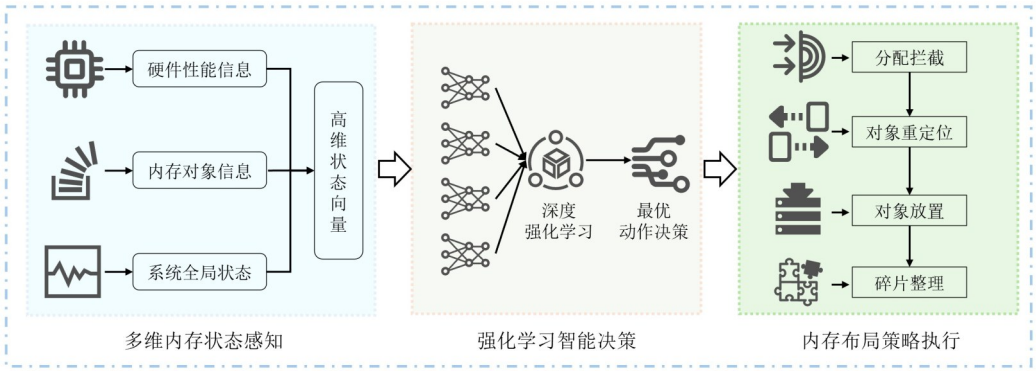


图 1 AIMO 的系统架构

Figure 1 System architecture of AIMO

AIMO 的工作流程为一个持续优化的闭环，当应用程序发起内存请求时，感知模块采集系统状态并编码；强化学习智能决策模块基于策略选择动作并输出最优策略；内存布局策略执行模块执行具体动作；执行结果再反馈给感知模块和决策模块，驱动策略的迭代与优化。

2.2 多维内存状态感知

对于内存资源的智能决策，其具体效果直接依赖于环境感知的全面性与准确性。若感知不足或存在偏差，后续的强化学习决策模块就难以生成最优的布局策略。为此，AIMO 设计了一个覆盖硬件性能、内存对象行为以及系统全局状态的多维感知模块，将即时的底层运行特征、对象级的动态行为以及全局环境与应用上下文信息逐层汇聚，最终整合为高维状态向量输入至后续强化学习决策模块。

2.2.1 硬件性能与介质状态感知

在人机物融合场景的嵌入式边缘计算节点上，内存布局优化的有效性高度依赖于对底层硬件物理状态的精确、实时获取。本模块的第一个单元，硬件性能与介质状态感知单元，目标是从物理层面捕获两类关键信息：一是反映系统与内存交互动态的硬件性能指标，二是在异构内存环境下表征不同存储介质固有属性与动态损耗的介质状态信息。

为实现低开销的实时监测，本模块利用了现代处理器内置的性能监控单元(Performance Monitoring Unit, PMU)。通过配置和读取硬件性能计数器，能够精确追踪微架构层面的事件。本模块定义系统的瞬时硬件性能向量 $\boldsymbol{S}_{\text{perf}}=[\hat{R}^{\text{LLC}}_{\text{miss}}, \widehat{\text{CPI}}, \hat{U}_{\text{bw}}, \hat{R}^{\text{TLB}}_{\text{miss}}]$，其中每个带^的分量均经过归一化处理。各分量的计算方式如下：末级缓存未命中率 $R^{\text{LLC}}_{\text{miss}}=N^{\text{LLC}}_{\text{miss}}/N^{\text{LLC}}_{\text{access}}$，每指令周期数 $\text{CPI}=N_{\text{cycles}}/N_{\text{instructions}}$，内存带宽利用率 $U_{\text{bw}}=N_{\text{bytes\_transferred}}/\left(T_{\text{sample}}\times \text{BW}_{\max}\right)$，转译后备缓冲器未命中率 $R^{\text{TLB}}_{\text{miss}}=N^{\text{TLB}}_{\text{miss}}/N^{\text{TLB}}_{\text{access}}$。其中，$N$ 代表事件的计数值，T_{sample} 是采样周期，$\text{BW}_{\max}$ 是理论最大带宽。

在人机物融合场景的边缘节点中，为了平衡成本、功耗与性能，通常会采用 SRAM、DRAM、非易失性内存等构成的异构内存体系。不同介质在访问延迟、带宽、功耗和耐久度上存在巨大差异。因此，必须对每种介质的静态属性和动态状态进行精确建模。本单元为系统中的第 i 个物理内存区域(对应一种介质)定义一个介质状态向量 $\boldsymbol{S}^{(i)}_{\text{media}}$，它包含了描述该介

质特性的多个维度。$\boldsymbol{S}^{(i)}_{\text{media}}=[T^{(i)},L^{(i)}_r,L^{(i)}_w,E^{(i)}_r,E^{(i)}_w,W^{(i)}]$,其中,$T^{(i)}$为介质类型,$L^{(i)}_r$和$L^{(i)}_w$是读/写延迟,$E^{(i)}_r$和$E^{(i)}_w$是读/写能耗,$W^{(i)}$是动态的归一化剩余耐久度(对NVM),其计算方式如式(1)所示。

$$W^{(b)}=\max\left(0,1-\frac{N^{(b)}_{\text{writes}}}{N_{\text{endurance\_max}}}\right) \tag{1}$$

其中,$N^{(b)}_{\text{writes}}$是特定物理块(Block)的累计写入次数;$N_{\text{endurance_max}}$是NVM介质的理论最大写入次数。

为了平滑上述指标的噪声,本单元使用滑动平均滤波器来计算每个指标的平均值。滑动平均滤波器会计算过去N个时间步长的平均值,其中N是一个可配置的参数。该平均值能够更好地反映硬件资源的长期趋势,避免因短时异常导致过度反应。最终得到的硬件层感知向量将作为输入特征之一,在后续模块中与对象和系统层信息进一步结合,为深度强化学习决策提供稳定的环境基础。

2.2.2　内存对象行为与热度感知

在对象层面,框架关注内存对象的生命周期与使用模式。本模块的第二个单元在分配路径中设置钩子(hook)函数与轻量化分析器,捕获对象的分配、释放、存活时长、访问频率及读写比例,并结合调用栈信息定位关键数据对象。基于这些特征,框架能够区分冷热数据,从而实现动态迁移策略。数据收集过程中,每次分配或释放都会触发hook函数,更新元数据结构并记录生命周期与访问统计。访问频率可依赖硬件性能计数器或断点机制,在访问时自动更新计数器。为控制监控开销,框架采用自适应更新算法,如算法1所示。算法通过维护对象元数据并根据访问模式动态调整采样频率,使高频对象获得精细监控、低频对象以低成本维护,在监测精度与系统负担之间取得平衡。

算法1　自适应更新算法

输入: object_id, access_event (read/write)

输出: Updated object_metadata

```
1. Procedure AdaptiveUpdate(object_id, access_event)
2.   object_metadata ← GetObjectMetadata(object_id)
3.   IF object_metadata == NULL THEN
4.    object_metadata ← CreateNewMetadata(object_id)
5.   ENDIF
6.   object_metadata.access_count ← object_metadata.access_count + 1
7.   IF access_event == read THEN
8.    object_metadata.read_count ← object_metadata.read_count + 1
9.   ELSE
10.   object_metadata.write_count ← object_metadata.write_count + 1
11.   ENDIF
12.   object_metadata.last_access_time ← CurrentTime()
13.   UpdateSamplingFrequency(object_metadata)
14.   UpdateObjectMetadata(object_id, object_metadata)
15. END Procedure
```

2.2.3　系统状态与应用上下文感知

在全局层面,感知模块需提供系统整体与应用场景的背景信息,以确保决策符合全局最优。本模块的第三个单元主要感知内存使用率、碎片化程度、空闲内存池分布,以及异构存储介质的拓扑结构与能耗特征,同时结合应用类型与阶段特征,参考性能目标,为智能体提供场景化指引。系统状态的获取依赖操作系统接口定期采集相关指标,存储介质的属性通过解析硬件描述文件获得,应用上下文则借助进程分析与性能监控工具识别类型和阶段,并提取延迟、带宽与能耗偏好等目标。与前两节的硬件层和对象层感知相结合,本单元获取的系统状态与应用上下文信息,会通过独热编码与场景标签机制转化为标准化输入。多维感知信息的融合流程如图2所示。

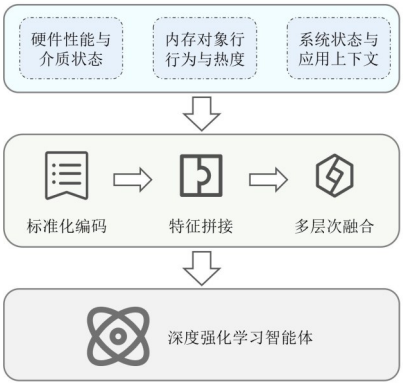


图2　多维感知信息的融合流程

Figure 2　Multi-dimensional perceptual information fusion process

最终,硬件指标、对象级特征和全局状态三类感知数据在融合模块中进行统一整合,形成高维状态向量,再输入到DRL决策单元。该融合过程确保了DRL智能体能够在全局视角下做出更加稳健的策略选择。

2.3　强化学习智能决策

DRL核心决策模块,接收由内存状态感知模块编码后的高维状态向量,通过内部的深度神经网络进行计算,最终输出一个具体的动作决策。本文将这一序列决策问题,严格地建模为一个马尔可夫决策过程(Markov Decision Process,MDP),并通过DRL算法进行求解。MDP为智能体(Agent)与环境的交互提供了一个强大的数学框架,其核心在于通过试错学习,发现能够最大化长期累积奖励的策略。

2.3.1　马尔可夫决策过程建模

一个标准的MDP可以由一个五元组(S,A,P,R,γ)来定义,其中S是状态空间,A是动作空间,P是状态

转移概率，R是奖励函数，γ是折扣因子。在本文中，AIMO的DRL智能体在每个决策时刻t观察到环境状态$S_t \in S$，执行一个动作$a_t \in A$，随后环境转移到新状态S_t+1，并返回一个即时奖励r_t。最终的目标是学习一个策略π，以最大化期望的累积折扣奖励$G_t=\sum_{k=0}^{\infty}\gamma^k r_{t+k}$。MDP的关键要素具体定义如下。

（1）状态空间。状态是对决策时刻系统环境的全面快照，其设计的优劣直接决定了DRL智能体决策的上限。为实现对人机物融合场景下关键节点的精细化管理，框架设计的状态向量S_t是一个高维度的综合体，包含了从感知模块获取的多层次信息。状态空间S的构成如表1所示。

（2）动作空间。动作是智能体在每个决策点可以执行的操作集合，是智能体策略在系统中的落地机制。为实现对内存布局的精细化控制。动作空间被设计为一个复杂的混合式空间，包含离散选择和连续参数。动作空间A的构成如表2所示。

表1 状态空间S的构成

Table 1 Composition of state space S

类别	具体特征	约束与取值
异构内存状态	各级缓存利用率与命中率、SRAM/DRAM/NVM各区域的可用容量、当前读写带宽等	所有利用率、命中率、容量比率均被量化为5个等级:{0:[0,20%],1:(20%,40%],2:(40%,60%],3:(60%,80%],4:(80%,100%]}.读写带宽根据硬件I/O总线峰值能力进行归一化,并同样离散化为5个等级
内存块信息	当前空闲/已分配内存块的地址、大小、所属进程ID、对象标签等	地址:映射至其所在的内存类型编号{0:SRAM,1:DRAM,2:NVM}. 大小:量化为4个规模等级{0:[0,4 KB],1:(4 KB,2 MB],2:(2 MB,1 GB],3:>1 GB} 进程ID/对象标签:使用哈希函数映射到固定大小的标识符空间(如0-255),非法或未映射ID统一归为特定值(如0)
关键对象特征	应用中关键数据结构(如AI模型权重、实时数据缓冲区)的ID、当前内存位置、预测的访问频率、读写比以及与其他对象的依赖关系	对象ID和内存位置为有效枚举值。访问频率和读写比为归一化至[0,1]的连续值。依赖关系由二进制掩码表示,指示依赖是否存在
应用/场景上下文	当前运行的应用类型、阶段(如AI推理阶段、数据预处理阶段)以及实时的性能目标	应用类型(如:0:Web服务,1:AI推理,2:数据库)和运行阶段(如:0:初始化,1:数据处理,2:清理)均来自一个预定义的有限类型集合。性能目标(如:0:低延迟,1:高吞吐)同样为离散枚举值

表2 动作空间A的构成

Table 2 Composition of action space A

类别	具体动作	参数与描述	约束与取值
分配时决策	分配内存	智能体需决策:1.内存类型;2.放置位置;3.对齐方式	1.类型:枚举{0:SRAM,1:DRAM,2:NVM} 2.位置:枚举{0:自动,1:近CPU,2:近对象[X]},其中X为有效对象ID 3.对齐:枚举{0:默认,1:缓存行对齐}
运行时决策	重定位	智能体需决策:1.迁移对象ID;2.目标位置暗示	1.对象ID:从当前已分配且可迁移的关键对象列表中选择有效ID 2.目标:枚举{0:DRAM,1:NVM,2:近对象[Y]},Y为有效对象ID
全局管理决策	碎片整理/内存池选择	智能体可决策是否在特定时机触发主动碎片整理,或为分配请求选择内存池	碎片整理:布尔动作{0:否,1:是},仅在系统空闲阈值以上且碎片度超阈值时可执行 内存池:枚举{0:默认池,1:大页池,2:持久化池}

（3）奖励函数。奖励函数是引导智能体学习的唯一信号，其设计至关重要。为平衡性能、效率和稳定性三大目标，并适应“场景计算机”的动态需求，本模块构建了一个场景自适应的多目标奖励函数R_t，如式（2）所示。

$$R_t=\omega_{\text{pref}}\cdot R_{\text{pref},t}+\omega_{\text{eff}}\cdot R_{\text{eff},t}+\omega_{\text{stab}}\cdot R_{\text{stab},t} \tag{2}$$

其中，ω_{pref}、ω_{eff}、ω_{stab}是根据当前应用场景上下文动态调整的权重系数，满足$\omega_{\text{pref}}+\omega_{\text{eff}}+\omega_{\text{stab}}=1$。为提升策略在不同场景下的泛化能力，避免人工设定权重的僵硬性，本框架采用基于规则的线性增益方式对权重进行小幅动态调节。具体而言，设初始权重为$\omega_{i,0}$，当满足某一场景触发条件时，对应权重按如下方式进行更新：

$$\omega_i=\text{clip}\left(\omega_{i,0}\cdot\left(1+k_i\cdot\delta_i\right),\omega_{i,\max}\right) \tag{3}$$

其中，k_i表示类别i的增益系数（通常取0.2~0.5，以限制权重变化幅度）；δ_i表示场景触发强度（如负载率、碎片率超阈值比例），已归一化至$[0,1]$；$\omega_{i,\max}$表示权重上限，设定为$\omega_{i,0}$的1.5~2倍；clip$(\cdot)$用于保证更新平稳，避免权重过大导致策略不稳定。例如：①当系统处于高负载（$\delta_{\text{pref}}=0.8$）时，性能权重更新为$\omega_{\text{pref}}=$

$\omega_{\text{pref},0}\cdot(1+0.3\times0.8)$;②当碎片率超过阈值20%($\delta_{\text{eff}}=0.2$)时,效率权重更新为$\omega_{\text{eff}}=\omega_{\text{eff},0}\cdot(1+0.5\times0.2)$;③当发生剧烈负载切换时,稳定性权重同样通过对应的δ_{stab}进行线性提升。该机制在保证权重动态可控的同时,避免了更新幅度过大导致的策略震荡,使策略的学习过程保持稳定和连续。其余各分量定义如下。

性能奖励(R_{pref}):主要由即时性能指标的变化构成,例如缓存命中率的提升(ΔCacheHitRate),并惩罚由内存操作(如重定位)引入的额外开销(OpCost),其计算如式(4)所示。

$$R_{\text{pref},t}=c_1\cdot\Delta\text{CacheHitRate}_t-c_2\cdot\text{OpCost}_t \tag{4}$$

其中,ΔCacheHitRate是通过对比当前时刻与上一时刻的全局缓存命中率差值计算得出。OpCost是一个归一化的开销估计,其计算依赖于本次决策触发的内存操作类型(如重定位、碎片整理等)及其涉及的数据量。这些开销参数在系统初始化时通过微基准测试进行标定,确保其能真实反映操作对系统性能(如延迟、带宽)的影响。c_1和c_2是用于平衡命中率收益与操作开销的权重系数。

资源效率奖励(R_{eff}):奖励内存碎片率的降低(ΔFragmentationRate),该指标反映了内存的利用效率。其计算结果如式(5)所示。

$$R_{\text{eff},t}=-c_3\cdot\Delta\text{FragmentationRate}_t \tag{5}$$

其中,ΔFragmentationRate是当前时刻与上一时刻系统内存碎片率(定义为1−(最大可用连续内存块/总可用内存))的差值,c_3是资源效率奖励的权重系数。

系统稳定性奖励(R_{stab}):在每个时间步给予一个小的生存奖励,但在发生内存分配失败或任务超时等负面事件时,给予一个较大的负惩罚($\text{Penalty}_{\text{fail}}$)。其计算如式(6)所示。

$$R_{\text{stab},t}=\text{const}_{\text{alive}}-\text{Penalty}_{\text{fail},t} \tag{6}$$

其中,$\text{const}_{\text{alive}}$是一个在每个时间步都给予的微小正常数,作为系统稳定运行的基线奖励。$\text{Penalty}_{\text{fail}}$是一个仅在特定负面事件(如内存分配失败、关键任务超时)发生时触发的较大负惩罚。

2.3.2　策略学习算法

针对上述MDP模型,特别是其高维状态空间和复杂的混合动作空间,本模块选取了近端策略优化(Proximal Policy Optimization,PPO)算法作为核心学习机制。PPO是一种先进的策略梯度方法,其通过裁剪目标函数限制每次策略更新步长,在保证学习效率的同时显著提升训练过程的稳定性,使其非常适合应用于操作系统内核环境中,尤其是资源受限且对稳定性要求极高的泛在操作系统内核。

和其他策略学习算法相比,在决策依赖性方面,PPO通过折扣因子γ优化期望累积折扣奖励$G_t=\sum_{k=0}^{\infty}\gamma^k r_{t+k}$,从而使做出的决策包含对内存布局状态的依赖,而适用于单步无状态决策问题的Bandit算法无法处理延迟奖励和决策的长期影响。在数据依赖性上,以PPO为策略算法的RL框架通过试错来自主发现新策略,而如模仿学习等算法所依赖的全面且最优的专家轨迹数据D,在最优内存管理问题中是不存在的。此外,在策略更新稳定性上,PPO引入了裁剪机制,优化的目标函数$L^{\text{CLIP}}(\theta)$如式(7)所示:

$$L^{\text{CLIP}}(\theta)=\hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\text{clip}\left(r_t(\theta),1-\epsilon,1+\epsilon\right)\hat{A}_t\right)\right] \tag{7}$$

其中,$r_t(\theta)=\dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$是新旧策略的概率比;$\hat{A}_t$是优势函数(advantage function)的估计值;ϵ是一个用于限制更新范围的超参数。式(7)通过min和clip操作,确保了策略更新不会偏离旧策略太远,防止了因学习率过大或样本噪声导致的破坏性更新,满足了部署环境的高稳定性要求。

为了将计算开销较大的PPO部署在轻量化嵌入式系统中,本框架采用了“离线预训练与安全约束下轻量在线微调”的部署架构,部署在内核中的仅为推理网络,其本质是有限层的矩阵运算,开销可控。而反向传播更新仅在系统检测到“空闲”状态时,基于回放缓冲中的数据低频触发,可以适配嵌入式的约束。

2.3.3　策略一致性与稳定性保障

为确保策略从训练环境平滑过渡到实际部署环境,并维持长期运行的可靠性,本框架采用了一套综合保障机制。该机制的核心是“离线预训练与安全约束下的轻量在线微调”范式,其设计基于前文定义的MDP模型与PPO算法特性。具体通过以下三个层面构建保障。

(1)安全可控的训练与在线更新流程。

首先,智能体的初始策略π_0在高保真仿真环境中,通过前述PPO算法进行充分训练,以最大化累积奖励G_t。此过程利用仿真环境快速生成大量覆盖广泛状态$s\in S$和动作$a\in A$的交互数据,使策略学习到一个稳健的基准。接着,本框架将离线训练后的策略部署于真实系统。在绝大多数时间,系统运行于推理模式,直接使用策略π_0根据当前状态s_t做出决策a_t,以保证实时性。同时,系统会周期性地将在线收集的真实交互数据序列(s_t,a_t,r_t,s_{t+1})存入一个缓冲区。当数据积累到一定规模,系统将在负载较低的维护窗

口,基于这些真实数据对策略进行一轮轻量的PPO优化更新。此过程直接复用式(7)的PPO目标函数$L^{\mathrm{CLIP}}(\theta)$,其裁剪机制能天然地限制策略更新步长,防止单次更新偏离已验证的有效策略过远,为核心算法的在线应用提供了内在的安全性。

(2)集成于MDP设计的策略稳定性与退化检测。

策略的稳定性保障与退化防范,其基础已内嵌于前述MDP核心要素中,本框架在在线阶段通过监控这些要素来间接实现检测。运行过程中,系统持续监控奖励函数中反映系统健康度的稳定性奖励$R_{\mathrm{stab},t}$,当系统频繁出现内存分配失败或任务超时,其持续负值即为策略退化的直接信号。同时,状态空间中关键指标(如缓存利用率、内存可用容量)若长期落入异常区间(如持续高于95%或低于10%),以及动作空间中出现极端或不合规动作选择(例如不合理地频繁迁移数据至NVM),均被判定为潜在退化行为。通过这套无需额外开销的内嵌监测机制,系统能够在策略出现异常趋势时及时捕获并进行响应。

(3)面向系统保障的安全回退机制。

当上述监测机制触发警报时,本框架将启动安全回退流程,立即切换到一套预定义的、经过验证的保守确定性规则(例如,始终优先在DRAM中分配;对于超过特定大小的请求直接使用NVM)。这些规则作为系统的安全基石,确保在DRL策略性能退化期间,系统的基本功能与稳定性不受影响。待策略在后续通过在线微调重新稳定后,系统在管理员确认或经过自动评估后切换回强化学习决策模式。

2.4 内存布局策略执行

为将DRL智能体的抽象决策转化为可在操作系统层面落地的具体操作,AIMO设计了一套精细化的内存布局操纵接口。这些接口正是2.3节中动作空间的系统实现形式,通过插桩、扩展和协同机制,对内存请求进行拦截、重定向与再优化。整个执行层既要保证动作的高效性和正确性,又要避免过高的运行时开销。因此,本模块提出了四类执行机制,分别对应于分配拦截、对象重定位、策略驱动的对象放置以及主动碎片整理。算法2给出了AIMO主控流程的伪代码,展示了策略生成与策略执行之间的关系。

2.4.1 分配拦截

在内存请求的入口,框架通过在分配器路径中嵌入轻量化hook函数,对系统调用进行拦截。拦截后的请求会先被编码成标准化的“分配请求描述”,包括请求大小、调用栈上下文、进程/线程ID等信息,再交由DRL决策层查询策略。若DRL输出了针对该请求的优化动作,例如选择目标内存介质或给出对象邻

算法2 内存布局策略的生成与执行

输入: Current high-dimensional memory state of the system (hardware/object/system)

输出: Execution result of the memory-layout action and its feedback record

```
1. WHILE system_is_running DO:
2.    state_raw = SenseMemoryState()  //多维内存状态感知模块输入
3.    state = NormalizeState(state_raw)  //基于PPO算法的智能动作决策:状态归一化
4.    action_probs = ActorNetworkForward(state) //通过Actor网络生成动作分布
5.    action_sample = SampleAction(action_probs)  //从动作分布中采样策略动作
6.    action = ValidateAndCorrect(action_sample)  //动作合法性检查
7.    value_est = CriticNetworkForward(state)  // Critic 网络估计状态价值,用于后续微调或训练
8.    SWITCH action.type:  //内存布局策略执行
9.          CASE ALLOCATE:  //分配拦截
10.              request = GetPendingAllocRequest()  //拦截待处理的分配请求
11.              params = MapActionToAllocParams(action)  //RL动作映射为分配器可识别参数
12.              result = AllocExecutor(request, params) //调用实际分配器执行优化后的分配行为
13.              RecordDecisionFeedback(state, action, result)
14.          CASE RELOCATE:  //对象重定位
15.              obj_id = action.target_object  //目标对象ID
16.              dest = action.target_location
17.              IF EstimateMigrationBenefit(obj_id, dest) > 0 THEN //判断对象迁移收益是否大于代价
18.                  result = RelocateObject(obj_id, dest) //执行迁移
19.              ELSE
20.                  result = SkipAction("migration not beneficial")
21.              ENDIF
22.              RecordDecisionFeedback(state, action, result)
23.          CASE PLACE_OBJECT:  //策略驱动的对象放置
24.              request = GetPendingAllocRequest()    //获取当前分配请求
25.              target = action.placement_hint  //RL输出邻接位置提示
26.              result = PlaceObjectWithHint(request, target)  //按提示执行智能放置
27.              RecordDecisionFeedback(state, action, result)
28.          CASE DEFRAGMENT:  //主动碎片整理
29.              IF action.level == LIGHT THEN  //根据动作选择轻量级/重量级整理方法
30.                  result = LightDefragment()  //轻量整理:快速合并空闲块
31.              ELSEIF action.level == HEAVY THEN
32.                  result = HeavyDefragment()  //重量整理:跨对象迁移,大幅降低碎片
33.              ENDIF
34.              RecordDecisionFeedback(state, action, result)
35.          DEFAULT:
36.              SkipAction("invalid or null action")
37.    END SWITCH
38.    StoreTransition(state, action, result)  //反向传播(用于在线微调或离线训练)
39. END WHILE
```

接放置建议,则由接口转换为底层分配器可识别的参数。为同时保证系统的实时性和策略优化能力,模块采用了“双队列”机制。一方面,快速队列将请求直接转发到底层分配器,以避免阻塞高频分配路径;另一方面,异步队列则收集同样的请求信息,并将其批量传递给DRL决策层,用于策略优化和模型更新。

2.4.2　对象重定位

对象重定位是本模块动态优化的核心机制。当DRL智能体检测到对象热度或访问模式发生变化时,系统通过重定位动作将其迁移至更优介质。过程包括三步:(1)锁定源对象;(2)数据拷贝或页表重映射;(3)更新元信息并恢复访问。为降低开销,模块优先采用“零拷贝”,仅在硬件不支持或数据不对齐时退化为物理拷贝。为刻画迁移代价,定义迁移代价函数如式(8)所示。综合拷贝时间、同步延迟和依赖关系,用于衡量性能扰动。仅当预期收益大于代价时,智能体才会执行迁移,该代价同时纳入奖励函数的开销项。

$$\mathrm{Cost}_{\mathrm{relocate}}\left(\mathrm{obj}_i\right)=T_{\mathrm{copy}}\left(\mathrm{size}_i,b\omega_{\mathrm{src}\to\mathrm{dst}}\right)+T_{\mathrm{sync}}\left(\mathrm{dep}_i\right)\tag{8}$$

其中,$\mathrm{Cost}_{\mathrm{relocate}}\left(\mathrm{obj}_i\right)$表示对象$\mathrm{obj}_i$在一次重定位过程中的总体代价;$T_{\mathrm{copy}}\left(\mathrm{size}_i,b\omega_{\mathrm{src}\to\mathrm{dst}}\right)$表示数据拷贝时间,取决于对象大小$\mathrm{size}_i$及源介质到目标介质的有效带宽$b\omega_{\mathrm{src}\to\mathrm{dst}}$;$T_{\mathrm{sync}}\left(\mathrm{dep}_i\right)$表示同步延迟,反映了与对象$\mathrm{obj}_i$相关的依赖数据或任务在迁移过程中必须等待的同步时间,dep_i表示与对象obj_i相关的依赖集合。

2.4.3　策略驱动的对象放置

模块支持策略驱动的智能放置功能。针对新的分配请求,DRL智能体不仅能够决定对象应当分配到哪一类存储介质,还可以给出相对位置暗示,例如与某个高频访问对象相邻,从而提升缓存局部性。与传统静态规则相比,该策略能够根据实时的系统状态和应用上下文动态调整:在延迟敏感的推理任务中,智能体更倾向于将关键对象置于高速介质并保证数据局部性;而在能耗敏感或批处理任务中,则可能优先考虑能效目标,将低频访问数据迁移至低功耗介质。对象放置问题可形式化为以下优化目标,如式(9)所示。

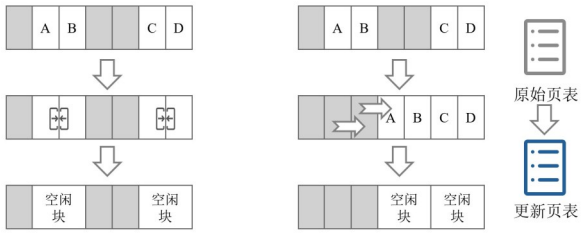
$$\mathrm{Place}\left(\mathrm{obj}_j\right)=arg\min_{m\in M}\left(L_{\mathrm{access}}\left(\mathrm{obj}_i,m\right)+\lambda\cdot D\left(\mathrm{obj}_i,N\left(\mathrm{obj}_i\right)\right)\right)\tag{9}$$

其中,M表示可选的存储介质集合;L_{access}表示对象在介质m上的平均访问延迟;$D(\cdot)$表示与邻近对象集合$N\left(\mathrm{obj}_i\right)$的距离代价;$\lambda$为权衡系数。通过这样的策略,系统能够动态实现“热数据上移、冷数据下沉”,并兼顾局部性和能耗目标。

为提升可观测性,放置结果在执行后会被记录进“决策-动作-反馈”日志,用于后续的在线微调和离线再训练。通过不断累积这些数据,系统不仅能够在运行过程中进行轻量化的在线微调,还能在离线阶段利用大规模历史数据进行再训练,从而实现策略的持续演进与优化。

2.4.4　主动碎片整理

在多任务并发和频繁分配/释放的泛在计算环境中,内存碎片化会持续积累,不仅降低分配效率,还可能导致大块请求失败,影响关键任务的实时性与稳定性。为此,本模块将碎片整理纳入智能体的可控动作,使其能依据碎片率、分配失败率及负载特征主动触发整理,并结合奖励函数中的资源效率项在性能与开销间进行权衡。当系统频繁出现大块分配失败时,智能体倾向于触发重量级整理;而在碎片率适中但延迟敏感场景下,则更倾向于轻量级整理以降低运行干扰。轻量级和重量级碎片整理,如图3所示。



(a) 轻量级碎片整理　　(b) 重量级碎片整理

(a) Lightweight defragmentation　　(b) Heavyweight defragmentation

图3　轻量级和重量级碎片整理

Figure 3　Lightweight and heavyweight defragmentation

具体而言,轻量级整理仅合并相邻空闲块,成本低,适合高负载下快速回收空间;重量级整理涉及跨对象迁移与页表重映射,虽代价高但能显著降低碎片率,通常在低负载或功耗优化模式下执行。通过这一分层机制,框架可在不同场景中实现性能与长期内存利用率的自适应平衡。

3　实验评估

3.1　实验平台与环境

为全面评估AIMO框架在泛在操作系统中内存布局优化的有效性,本文构建了一个基于典型资源受限型边缘智能设备的实验平台。该平台以Raspberry Pi Zero 2W作为核心硬件,配备四核ARM Cortex-A53处理器及512 MB LPDDR2内存。此类设备以其低功耗、小尺寸、高集成度等优势,广泛应用于智能家居、环境监测及工业传感等泛在场景,是验证泛在操作系统性能的理想载体。

在软件环境方面,实验平台运行OpenHarmony

3.2 LTS轻量系统。该操作系统面向物联网与边缘设备设计,采用微内核架构,支持内核功能可裁剪和模块化部署,具备轻量化、高安全及分布式协同等显著特性。AIMO框架以源代码级形式深度集成至OpenHarmony内核,作为其内存管理子系统的智能增强模块,负责实现内核态内存资源的动态优化与自适应调控。实验硬件平台配置如表3所示。

表3 实验硬件平台配置

Table 3 Experimental hardware platform configuration

配置项	规格
型号	Raspberry Pi Zero 2W
CPU	4核 ARM Cortex-A53 @1.0 GHz
GPU	VideoCore IV
内存	512 MB LPDDR2
存储	16 GB microSD

本文设计并构建了两个典型实验场景。(1)环境主动告警场景:系统利用DHT22和MQ-135传感器实时感知物理环境参数,并经本地智能决策后,向用户终端推送告警信息,同时支持接收用户的确认反馈。此过程模拟了持续性的感知负载,涉及高频小对象内存分配操作,主要用于评估AIMO框架在保障系统实时性与稳定性方面的性能。(2)意图驱动预加载场景:当用户发出"准备拍照"等高层意图指令时,系统基于当前内存资源态势,由AIMO框架动态调整内存分配策略,预先分配图像缓冲区并加载相关处理模块。随后在模拟的10 s延迟后执行拍摄操作。该场景模拟了突发性大块内存请求,重点考察AIMO框架在内存碎片控制与资源高效回收方面的性能表现。

3.2 评价指标

本文从以下五个核心维度构建了评价指标体系。所有指标均借助内核级高精度计时器与日志系统进行采集,以确保测量精度及数据可靠性。

(1)最坏情况执行时间(Worst-Case Execution Time,WCET):采用WCET评估内存分配操作的时间确定性。在AIMO框架所增强的内存分配与释放路径上,插入高精度计时点,捕获其从入口到返回的完整执行周期。参考相关研究[36],实验利用ARM Cortex-A53处理器内建的DWT周期计数器进行采样,并将结果转换为微秒(μs)进行报告。该指标能够反映系统在高负载或内存碎片化场景下的最大响应延迟,是衡量实时系统性能的关键依据。

(2)最大可用连续内存块大小:为评估系统长期运行后的空间利用效率与内存碎片化程度,采用最大可用连续内存块大小作为核心代理指标。该值越小,表明内存被分割得越零碎,系统满足大块内存请求的能力越弱。此方法已被广泛应用于实时内存分配器的性能评估[36]。实验记录6 h压力测试期间该值的演化趋势,并在测试结束时报告最终大小(单位:KB)。

(3)关键任务延迟:该指标用于评估系统在人机物深度融合场景下的响应能力。具体而言,在环境主动告警场景中,测量从传感器数据超限到告警信息成功推送至用户终端的总耗时;在意图驱动预加载场景中,测量从用户发出"准备拍照"指令到图像处理模块资源预分配完成的响应时间。此指标直接关联用户体验与系统整体的协同效率。

(4)内存分配失败率:为检验框架在高压负载下的鲁棒性,统计大块内存请求的失败比例。实验中,周期性发起128 KB及以上大小的内存分配请求,记录总请求数与失败次数,并计算失败率。该指标模拟了设备在突发负载下遭遇内存不足的风险,是衡量框架稳定性的关键指标。

(5)AIMO框架运行时开销:该指标用于评估AIMO框架自身的资源消耗。具体测量框架内部DRL决策模块在运行期间的平均CPU占用率与内存驻留开销。此指标用于验证框架在资源受限环境下的轻量级特性,确保其对主应用任务的干扰最小化。

3.3 消融实验

本节通过构造四种具备明确功能削弱的消融配置,旨在量化AIMO框架中三大核心模块对系统性能的各自贡献。所有实验均在统一的OpenHarmony轻量系统环境下,运行相同的6 h混合负载压力测试,以确保对比的公平性与结果的可靠性。本实验关注的DRL决策模块开销仅来自运行时推理阶段,训练过程已在离线完成,不依赖嵌入式设备的额外硬件资源。该开销指预训练策略模型在内核中执行推理、依据系统状态做出决策时产生的计算与内存消耗,可视为强化学习模块的实际运行时开销。

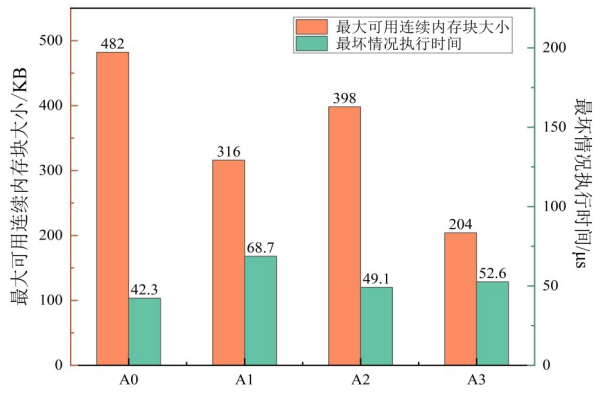
定义以下四个实验组:A0为完整AIMO框架,作为基准组,包含感知、决策与执行模块的全链路协同工作;A1在A0基础上移除感知模块。在此配置下,框架退化为静态规则驱动型策略,无法根据环境或用户意图变化进行自适应调整;A2保留感知模块,但禁用DRL决策模块。内存操作将由固定的静态策略控制,主要用于验证DRL决策带来的动态优化能力;A3保留感知与决策模块,但在执行阶段关闭对象重定位机制。此配置下,框架仅启用对象放置与主动碎片整理等其他执行策略,旨在检验对象重定位这一关键操作对内存碎片治理与空间效率的独特贡献。通过对比A0至A3在上述五个指标上的表现差异,识别各模块对系统整体性能的关键影响,从而全面验证AIMO框架设计的合理性与必要性。结果如图4所示。

图4(a)展示了不同配置下系统的实时性能与内

存可用性。在实时性方面,完整方案A0的最坏情况执行时间为42.3 μs,与移除了感知模块的A1(68.7 μs)相比,降低了38.4%。这表明环境感知能力使系统能够预见并提前为负载变化做准备,从而优化了任务的响应确定性。在内存管理方面,经过长时间运行,A0可维持的最大连续内存块达到482 KB,显著优于仅移除重定位模块的A3(204 KB)。这说明了动态对象重定位机制在主动整合内存碎片、保障大块内存分配能力方面的有效性。此外,移除了DRL决策模块的A2(398 KB)表现介于A0和A3之间,虽然其静态重定位策略优于无重定位的A3,但与A0相比仍有较大差距。这凸显了基于DRL的动态决策相较于静态策略,在长期维持内存资源可用性方面的优越性。

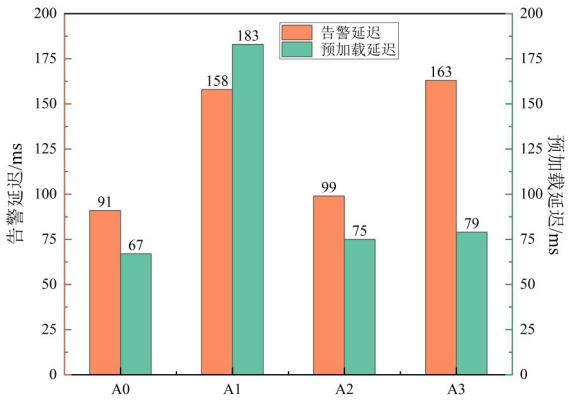
图4(b)展示了AIMO在两个典型应用场景下对端到端服务延迟的优化效果。在环境主动告警场景中,A0方案的平均响应延迟为91 ms。相比之下,缺乏环境感知能力的A1无法预测传感器数据趋势以预先分配资源,导致其延迟增加至158 ms,相较A0增加了73.6%。在意图驱动的预加载场景中,A0能够基于用户意图和当前内存状态,快速预分配所需资源,实现了67 ms的低延迟响应。而A3由于严重的内存碎片问题,其预加载过程需要额外的内存整理,延迟增至79 ms。A1则因缺乏感知能力而无法执行预加载,只能进行被动响应,其延迟高达183 ms,比A0高出173%。实验结果表明,AIMO通过"感知-决策-执行"的闭环机制,有效地将系统行为从被动资源响应转变为主动服务供给,从而显著改善了应用级的服务质量。

图4(c)从运行时开销与系统稳定性两个维度对不同方案进行了评估。在开销方面,AIMO框架(A0)



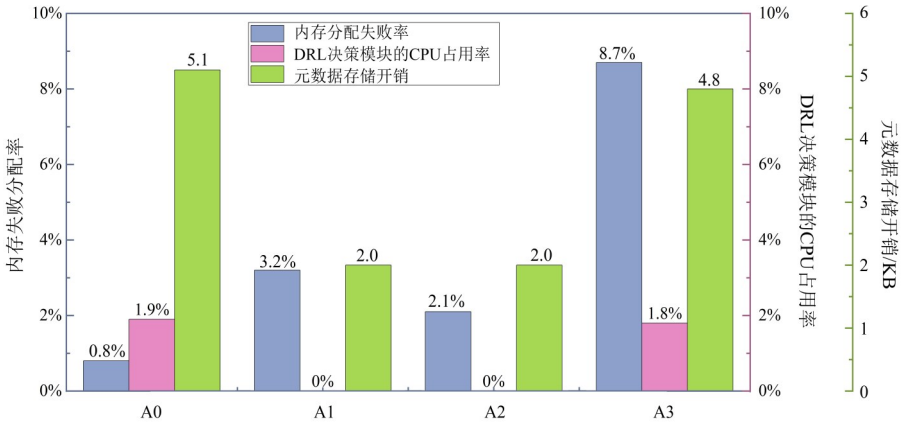
(a) 最大可用连续内存块大小与最坏情况执行时间

(a) Maximum available contiguous memory block size and worst-case execution time



(b) 告警延迟与预加载延迟

(b) Alert latency and preloading latency



(c) 内存分配失败率与运行开销

(c) Memory allocation failure rate and runtime overhead

图4　消融实验性能对比

Figure 4　Performance comparison of ablation studies

表现出轻量化的特点。其DRL决策模块的平均CPU占用率为1.9%，涵盖策略推理本身及其所需的内存状态构建、数据处理和异步调度等辅助操作的综合开销。元数据存储开销为5.1 KB，该存储开销相对于512 MB的系统总内存占比不足0.001%，表明其设计适用于资源受限的边缘环境。在稳定性方面，以大块内存（>128 KB）分配失败率作为衡量指标，A0方案的失败率仅为0.8%。作为对比，缺乏重定位能力的A3失败率高达8.7%，系统的稳定性受到严重影响。同时，A2的失败率（2.1%）虽低于A1（3.2%），但仍显著高于A0，这说明仅有环境感知而无智能决策，不足以保证系统在高动态环境下的鲁棒性。综上，集成了感知、决策与执行的完整方案是保障系统稳定运行的关键。

消融实验的结果表明，AIMO框架中感知、决策与执行三个模块的有机结合是其性能优势的根本来源。实验证明，缺少任一环节都会导致系统在实时性、内存稳定性或应用响应速度方面出现明显短板。因此，完整的AIMO框架能够在维持极低开销的同时，有效整合离散资源、预见并适应负载变化，从而为动态的人机物融合应用提供了可靠且高效的底层支撑。

此外，我们量化了AIMO框架中DRL决策模块的运行时性能开销，如表4所示。该表展现了在环境主动告警场景与意图驱动预加载场景下，单次DRL决策的平均延迟、最坏延迟以及决策触发频率等指标。在环境主动告警场景中，观测到单次DRL决策的平均延迟为23.4 μs，而最坏延迟记录为33.8 μs。在此场景下，DRL模块的决策触发频率约为12 次/秒。在意图驱动预加载场景中，决策模块表现出更低的延迟，平均延迟下降至14.7 μs，最坏延迟也保持在22.5 μs。同时，该场景下的决策触发频率略高，达到20 次/秒。这些数据表明，DRL决策模块在不同工作负载情境下，其单次推理延迟均能稳定在微秒级，这种低延迟特性，反映其能够有效融入资源受限的嵌入式系统，并有助于保障DRL决策模块在系统关键路径中引入的额外计算开销保持在可接受的范围内，从而支持系统整体的高效和响应性。

表4 DRL决策模块运行开销

Table 4 Runtime overhead of the DRL decision module

场景	单次决策平均延迟/μs	单次决策最坏延迟/μs	决策触发频率/(次/s)
环境主动告警场景	23.4	33.8	12
意图驱动预加载场景	14.7	22.5	20

3.4 对比实验

3.4.1 有效性对比

为验证AIMO框架在真实负载下的优化效果，本文对比了系统启用（AIMO On）与禁用（AIMO Off）两种模式在典型测试场景下的性能表现。实验持续运行6 h，涵盖环境主动告警与意图驱动预加载两类负载，结果如表5所示。

实验结果表明，与禁用模式相比，启用AIMO框架能为系统带来底层与应用层面的全方位性能提升。在底层，AIMO展现出对内存资源更强的确定性控制能力。一方面，它将内存分配的WCET降低了38.4%（从68.7~42.3 μs），保障了系统的实时行为；另一方面，它通过主动的碎片整理和重定位，将最大连续可用内存提升了136%（从204 KB至482 KB），并将大块内存分配失败率由8.7%降至0.8%。

表5 AIMO框架启用（On）与禁用（Off）模式的性能对比

Table 5 Performance comparison between AIMO enabled (on) and disabled (off) modes

模式	WCET/μs	最大可用连续内存块/KB	内存分配失败率/%	环境告警延迟/ms	意图驱动预加载延迟/ms
AIMO(On)	42.3	482	0.8	91	67
AIMO(Off)	68.7	204	8.7	163	183

在应用层，对于环境驱动的任务，AIMO通过预测性资源准备将响应延迟降低了44.2%（从163~91 ms）；对于用户意图驱动的任务，其主动服务能力使响应延迟降低了63.4%（从183~67 ms）。综上所述，AIMO框架在真实边缘计算场景中，不仅优化了底层内存管理的效率与稳定性，也提升了上层应用的响应性能，证实了其作为操作系统智能增强层的有效性。

3.4.2 高效性对比

本节旨在将AIMO与主流内存管理策略进行基准测试，以验证其能否在解决内存碎片这一长期挑战的同时，保持实时分配的高效率。具体来说，实验将AIMO与TLSF（Two-Level Segregated Fit）、First-Fit with Merging及RT-Thread堆管理器在相同的OpenHarmony内核环境中进行对比，结果如表6所示。

表6　不同内存管理策略在环境告警场景下的性能对比

Table 6　Performance comparison of different memory management strategies in environmental alert scenarios

方法	WCET/μs	最大可用连续空闲块/KB	内存分配失败率/%	告警推送延迟/ms
AIMO	44.1	476	0.9	93
TLSF	45.3	318	3.4	121
First-Fit with Merging	57.9	276	8.8	156
RT-Thread堆管理器	60.2	202	9.1	168

如表6所示,AIMO在维持低分配开销的同时,显著改善了内存的长期稳定性。在执行开销方面,AIMO的WCET为44.1 μs,与高性能实时分配器TLSF(45.3 μs)基本持平,表明其引入的智能决策机制未带来额外的性能负担。在内存碎片抑制方面,AIMO的优势尤为突出。长期运行后,其最大连续空闲块达到476 KB,远高于TLSF(318 KB)及两种策略(276 KB,202 KB)。此外,其大块内存分配失败率仅为0.9%,远低于其他方案的3.4%~9.1%。在应用性能方面,底层的内存优化有效降低了上层服务的延迟。AIMO将告警推送延迟控制在93 ms,相较于基线策略的121~168 ms,显著提升了响应速度和稳定性。综上,AIMO的内存布局管理机制在持续的小对象分配负载下,能有效减缓碎片累积,从而在不牺牲实时性能的前提下,保障系统的长期稳定运行与服务质量。

意图驱动预加载是对内存管理器在高负载后能否响应突发需求的关键考验。表7给出了不同内存管理策略在该场景下的性能对比。与前一场景类似,AIMO的WCET(12.4 μs)与高性能分配器TLSF(12.8 μs)几乎持平,表明其运行效率具有竞争力。然而,AIMO的核心优势在于其对内存碎片的持续抑制。当预加载任务请求大块内存时,AIMO凭借更好的内存状态(最大连续空闲块476 KB,失败率0.9%),能够快速满足请求。这导致AIMO的预加载完成延迟为69 ms,显著低于TLSF的82 ms以及其他两种策略的106 ms和119 ms。综上,在需要快速响应大块内存请求的应用场景中,AIMO通过主动的内存状态管理,有效保障了上层服务的低延迟和高成功率。

表7　不同内存管理策略在意图驱动预加载场景下的性能对比

Table 7　Performance comparison of different memory management strategies under intent-driven preloading scenarios

方法	WCET/μs	最大可用连续空闲块/KB	内存分配失败率/%	预加载完成延迟/ms
AIMO	12.4	476	0.9	69
TLSF	12.8	318	3.4	82
First-Fit with Merging	16.9	276	8.8	106
RT-Thread堆管理器	18.1	202	9.1	119

本部分实验基于Raspberry Pi Zero 2W这一典型的低算力边缘节点。其有限的内存与处理能力能更真实地反映AIMO在资源紧张环境下运行时重定位、内存碎片治理和策略推理等机制的系统开销与行为特征。实验结果初步验证了AIMO在资源受限平台上的可部署性与稳定性。AIMO的状态建模、策略推理与内核执行接口不依赖特定硬件加速器,具备向更广泛平台迁移的潜力。更高性能的边缘设备虽具备更充裕的计算与内存资源,通常不会面临同等程度的内存管理压力,其性能表现也可能不同,因此,对AIMO在这些平台上的通用性验证及其在不同场景下的性能优化研究,将在后续工作中展开。

4　结论

本文提出了一种基于强化学习的泛在操作系统内存布局优化框架AIMO,并成功在OpenHarmony内核中进行了实现与验证,旨在解决泛在操作系统中长期存在的内存碎片化挑战。与传统的被动式内存管理策略相比,该框架能够通过对应用内存行为的学习与预测,主动进行内存布局优化。框架内置的感知决策模型能够实时分析内存请求模式,并指导执行模块进行碎片整理与空间预留。实验结果表明,在模拟的长期复杂负载下,AIMO框架相较于RT-Thread堆管理器及First-Fit等经典策略,在显著提升内存空间利用率的同时,并未牺牲关键的实时性能。本研究提出的AIMO框架为泛在操作系统内核的资源管理提供了一种全新的智能化思路,为构建下一代自适应、高效率的泛在操作系统奠定了坚实的技术基础,后续工作将聚焦于其在更高性能边缘设备上的通用性验证与多场景下的性能优化研究。

参考文献

[1] 梅宏, 曹东刚, 谢涛. 泛在操作系统: 面向人机物融合泛在计算的新蓝海[J]. 中国科学院院刊, 2022, 37(1): 30-37.
Mei Hong, Cao Donggang, Xie Tao. Ubiquitous operating system: Toward the blue ocean of human-cyber-physical ternary ubiquitous computing[J]. Bulletin of Chinese Academy of Sciences, 2022, 37(1): 30-37. (in Chinese)

[2] Cao Donggang, Xue Dongliang, Ma Zhiyi, et al. XiUOS: An open-source ubiquitous operating system for industrial Internet of Things[J]. Science China Information Sciences, 2022, 65(1): 117101.

[3] Mei Hong, Guo Yao. Toward ubiquitous operating systems: A software-defined perspective[J]. Computer, 2018, 51(1): 50-56.

[4] Liu Xuanzhe, Wang Shangguang, Ma Yun, et al. Operating systems for resource-adaptive intelligent software: Challenges and opportunities[J]. ACM Transactions on Internet Technology (TOIT), 2021, 21(2): 27.

[5] 靳晓忠, 刘海坤, 赖皓, 等. 一种可重构异构内存架构和控制器[J]. 电子学报, 2024, 52(9): 3038-3051.
Jin Xiaozhong, Liu Haikun, Lai Hao, et al. A reconfigurable heterogeneous memory architecture and memory controller[J]. Acta Electronica Sinica, 2024, 52(9): 3038-3051. (in Chinese)

[6] 李琪, 钟将, 李雪, 等. 基于新型非易失存储器的混合内存架构的内存管理机制[J]. 电子学报, 2019, 47(3): 664-670.
Li Qi, Zhong Jiang, Li Xue, et al. Memory management mechanism for hybrid memory architecture based on new non-volatile memory[J]. Acta Electronica Sinica, 2019, 47(3): 664-670. (in Chinese)

[7] Tripathi R R K, Singh P K, Singh S. Robust left-right hashing scheme for ubiquitous computing[J]. Engineering Research Express, 2024, 6(3): 035225.

[8] 钱振江, 刘永俊, 姚宇峰, 等. 微内核架构内存管理的形式化设计和验证方法研究[J]. 电子学报, 2017, 45(1): 251-256.
Qian Zhenjiang, Liu Yongjun, Yao Yufeng, et al. Research on method of formal design and verification of memory management based on microkernel architecture[J]. Acta Electronica Sinica, 2017, 45(1): 251-256. (in Chinese)

[9] 张佳辰, 胡泽瑞, 赵盛, 等. VMFS: 一种持久性内存统一管理系统[J]. 电子学报, 2021, 49(12): 2299-2306.
Zhang Jiachen, Hu Zerui, Zhao Sheng, et al. VMFS: A unified persistent memory management system[J]. Acta Electronica Sinica, 2021, 49(12): 2299-2306. (in Chinese)

[10] 范晓鹏, 阎松, 翁楚良. 面向泛在操作系统的结构化存储[J]. 中国科学: 信息科学, 2024, 54(3): 461-490.
Fan Xiaopeng, Yan Song, Weng Chuliang. Structured storage for ubiquitous operating systems[J]. Scientia Sinica Informationis, 2024, 54(3): 461-490. (in Chinese)

[11] Lim J, Song S, Lee S, et al. The design of a new virtualization-based server cluster system targeting for ubiquitous IT systems[M]//Ubiquitous Computing Application and Wireless Sensor. Dordrecht: Springer Netherlands, 2015: 361-375.

[12] Rajesh S C, Kushwaha A S. Memory optimization techniques in large-scale data management systems[J]. International Journal for Research in Management and Pharmacy, 2024, 13(11): 37.

[13] Kwon W, Li Zhuohan, Zhuang Siyuan, et al. Efficient memory management for large language model serving with PagedAttention[C]//Proceedings of the 29th Symposium on Operating Systems Principles. New York: ACM, 2023: 611-626.

[14] Li Haoyang, Li Yiming, Tian Anxin, et al. A survey on large language model acceleration based on KV cache management[J]. Transactions on Machine Learning Research, 2025, 2025.

[15] Zhao Pinxue, Zhang Hailin, Fu Fangcheng, et al. MEMO: Fine-grained tensor management for ultra-long context LLM training[J]. Proceedings of the ACM on Management of Data, 2025, 3(1): 53.

[16] Hu Zhongzhe, Xiao Junmin, Deng Zheye, et al. MegTaiChi: Dynamic tensor-based memory management optimization for DNN training[C]//Proceedings of the 36th ACM International Conference on Supercomputing. New York: ACM, 2022: 25.

[17] Zhu Zhanda, Giannoula C, Andoorveedu M, et al. Mist: Efficient distributed training of large language models via memory-parallelism co-optimization[C]//Proceedings of the Twentieth European Conference on Computer Systems. New York: ACM, 2025: 1298-1316.

[18] Lim A, Maddukuri A. Reinforcement learning for dynamic memory allocation[PP/OL]. V2. arXiv (2025-10-08)[2025-10-10]. https://arxiv.org/abs/2410.15492.

[19] Jia Danlin, Wang Li, Valencia N, et al. Learning-based dynamic memory allocation schemes for apache spark data processing[J]. IEEE Transactions on Cloud Computing, 2024, 12(1): 13-25.

[20] Garrido L A, Nishtala R, Carpenter P. Continuous-action reinforcement learning for memory allocation in virtualized servers[C]//Proceedings of the ISC High Performance 2019 International Workshops on High Performance Computing. Heidelberg: Springer, 2019: 13-24.

[21] Kumar N N, Saravana S, Balamurugan S, et al. Optimized memory allocation in edge-PLCs using deep Q-networks and bidirectional LSTM with quantum genetic algorithm[J]. e-Prime-Advances in Electrical Engineering, Electronics and Energy, 2024, 10: 100762.

[22] Wang Mowei, Huang Sijiang, Cui Yong, et al. Learning buffer management policies for shared memory switches[C]//Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications. Piscataway: IEEE, 2022: 730-739.

[23] Yi Xinyue, Du Hongchao, Wang Yu, et al. ArtMem: Adaptive migration in reinforcement learning-enabled tiered memory[C]//Proceedings of the 52nd Annual International Symposium on Computer Architecture. New York: ACM, 2025: 405-418.

[24] Chang J, Doh W, Moon Y, et al. IDT: Intelligent data placement for multi-tiered main memory with reinforcement learning[C]//Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing. New York: ACM, 2024: 69-82.

[25] Karimov E, Evenblij T, Chamazcoti S A, et al. PARL: Page allocation in hybrid main memory using reinforcement learning[J]. Journal of Systems Architecture, 2025, 159: 103310.

[26] Yoo S, Shin D. Reinforcement learning-based SLC cache technique for enhancing SSD write performance[C]//Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems. USENIX Association, 2020: 7.

[27] Huang Darong, Pahlevan A, Costero L, et al. Reinforcement learning-based joint reliability and performance optimization for hybrid-cache computing servers[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(12): 5596-5609.

[28] 范浩, 徐光平, 薛彦兵, 等. 一种基于强化学习的混合缓存能耗优化与评价[J]. 计算机研究与发展, 2020, 57(6): 1125-1139.
Fan Hao, Xu Guangping, Xue Yanbing, et al. An energy consumption optimization and evaluation for hybrid cache based on reinforcement learning[J]. Journal of Computer Research and Development, 2020, 57(6): 1125-1139. (in Chinese)

[29] Xie Qian. Application of machine learning algorithms in data cache and storage hierarchical management[C]//Proceedings of the 3rd International Conference on Data Science and Information System (ICDSIS). Piscataway: IEEE, 2025: 1-6.

[30] Li Xingchen, Yuan Zhihang, Guan Yijin, et al. Flatfish: A reinforcement learning approach for application-aware address mapping[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(11): 4758-4770.

[31] Yang Chongyi, Zhang Zhendong, Wang Xiaohang, et al. Adaptive caching policies for Chiplet systems based on reinforcement learning[C]//Proceedings of 2023 IEEE International Symposium on Circuits and Systems (ISCAS). Piscataway: IEEE, 2023: 1-5.

[32] Yang Huijing, Fang Juan. A fairness-aware prefetching mechanism based on reinforcement learning for multi-core systems[C]//Proceedings of 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys). Piscataway: IEEE, 2023: 639-646.

[33] Kim H, Yeom H Y. LPR: Learning-based page replacement scheme for scientific applications[C]//Proceedings of the 23rd International Middleware Conference Industrial Track. New York: ACM, 2022: 36-42.

[34] Jain R, Panda P R, Subramoney S. Cooperative multi-agent reinforcement learning-based co-optimization of cores, caches, and on-chip network[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2017, 14(4): 32.

[35] Li Jingchun, Zhou Fanqin, Zhang Guoyi, et al. Resource allocation for componentized multimedia service in ubiquitous computing power environment[C]//Proceedings of 2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). Piscataway: IEEE, 2021: 1-6.

[36] Masmano M, Ripoll I, Crespo A, et al. TLSF: A new dynamic memory allocator for real-time systems[C]//Proceedings of the 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004. Piscataway: IEEE, 2004: 79-88.

作者简介

沙乐天 男,1985年5月出生于江苏省徐州市。现为南京邮电大学计算机学院、软件学院、网络空间安全学院教授、博士生导师。主要研究方向为系统安全、漏洞挖掘、恶意软件分析。
E-mail: ltsha@njupt.edu.cn

陈 霄 男,1995年10月出生于江苏省宿迁市。现为南京邮电大学教育科学与技术学院讲师。主要研究方向为物联网安全、软件安全、机器学习。
E-mail: 2020070132@njupt.edu.cn

郑红美 女,2001年12月出生于四川省乐山市。现为南京邮电大学硕士研究生。主要研究方向为IoT安全。
E-mail: 1260857102@qq.com

潘家晔 男,1985年10月出生于江苏省扬州市。现为南京邮电大学计算机学院、软件学院、网络空间安全学院副教授、硕士生导师。主要研究方向为系统安全、软件安全、网络安全。
E-mail: panjy@njupt.edu.cn

董建阔 男,1992年12月出生于河北省邢台市。现为南京邮电大学计算机学院、软件学院、网络空间安全学院副教授、硕士生导师。主要研究方向为公钥密码、后量子密码、并行计算。
E-mail: djiankuo@njupt.edu.cn

肖 甫 男,1980年10月出生于湖南省邵阳市。现为南京邮电大学教授、博士生导师。主要研究方向为物联网感知与计算、数据中心网络、智能信息处理。中国电子学会会员编号:E190029628M。
E-mail: xiaof@njupt.edu.cn