中国科学 : 信息科学 2026 年 第 56 卷 第 7 期: 1620–1634
SCIENTIA SINICA Informationis
论文

《中国科学》杂志社
SCIENCE CHINA PRESS
CrossMark click for updates

面向混部高压数据中心的应用执行追踪: 高效设计管理与智能分析诊断

王鑫凯[1], 李超[1*], 黄天浩[1], 郭进阳[1], 侯小凤[1], 徐国耀[2], 吴月敏[3], 过敏意[1]

1. 上海交通大学计算机学院, 上海 200240
2. 阿里巴巴控股集团, 杭州 310052
3. 阿里云智能, 杭州 310030
* 通信作者. E-mail: lichao@cs.sjtu.edu.cn

收稿日期: 2025–09–05; 修回日期: 2025–11–28; 接受日期: 2026–01–12; 网络出版日期: 2026–06–30

国家重点研发计划 (批准号: 2022YFB4501702) 和国家自然科学基金 (批准号: U23A6007) 资助项目

摘要 随着数据中心的快速发展, 软件复杂度与集群规模的持续增长对高效精准的应用观测能力提出了更高要求. 定位性能瓶颈和诊断性能异常需要观测应用的执行轨迹追踪来精确判断应用行为的因果关系. 本文发现在软件混合部署且硬件负载高压的复杂云数据中心环境, 现有执行追踪工具通常会使被观测应用产生 5%∼10% 的性能下降, 因此难以应用无感地部署在大规模集群中. 现代处理器引入的硬件追踪能力本应降低执行追踪的开销, 但是该能力面临的性能下降 – 空间占用 – 轨迹长度 – 管理难度四维度设计权衡未被研究人员充分探索以最小化追踪开销, 所以无法达到复杂数据中心对于执行追踪开销的高要求. 针对上述挑战, 本文提出 ETrace, 在节点和集群两个层次实现了性能开销的最优化. 本文利用硬件追踪能力的四维设计权衡, 在节点层面减少不必要的追踪控制操作以保障被追踪应用的性能, 并且基于应用实时状态动态分配有限的轨迹存储空间; 在集群层面, 通过灵活采样策略和云原生触发框架实现灵活, 高效的多节点轨迹追踪管理. 本文通过多种基准测试评估 ETrace 对于被观测应用造成的性能下降平均为 0.97%, 相比现有工作最高降低 85% 的开销. 同时本文以阿里巴巴数据中心的典型智能应用为例, 展示了 ETrace 提供的指令级追踪分析能力在异常诊断和瓶颈定位等数据中心管理任务的新发现.

关键词 执行追踪, 性能分析, 硬件追踪, 数据中心, 云计算

1 引言

在现代数据中心中, 软硬件的可观测性是高效、智能可管理性的基础. 新兴的智能软件系统存在高度的内部复杂性和广泛的外部交互, 如 AI 赋能的推荐与对话应用 [1], 高效准确的应用观测是数据采集处理实现管理闭环的基础 [2]. 面对复杂的性能异常问题如生产系统中由于硬件故障导致的组件慢性失效 [3] 或由于系统流量过载导致的亚稳态故障 [4], 实现稳定可靠的数据中心管理要求云服务提供商必须投入大量工程和人力资源, 来分析和诊断导致性能异常的因果关系并实现快速准确的故障恢复 [5,6].

引用格式: 王鑫凯, 李超, 黄天浩, 等. 面向混部高压数据中心的应用执行追踪: 高效设计管理与智能分析诊断. 中国科学: 信息科学, 2026, 56: 1620–1634, doi: 10.1360/SSI-2025-0384
Wang X K, Li C, Huang T H, et al. Intra-service tracing for shared and stressed datacenters: efficient management and intelligent analysis. Sci Sin Inform, 2026, 56: 1620–1634, doi: 10.1360/SSI-2025-0384

© 2026《中国科学》杂志社

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

分析和诊断复杂应用的性能异常具有挑战性, 主要是运行时包含时序信息的轨迹追踪的缺失. 目前实时采集的数据包括 CPU 利用率和应用吞吐等统计指标, 这种统计可观测性可以高效刻画异常发生前后的平均特征, 但是无法推断性能表现的因果关系, 针对数据中心中复杂隐蔽的性能故障经常无能为力. 相反, 通过执行追踪实现的时序可观测性, 能够可解释地揭示应用行为的因果关系 [2]. 目前服务间的分布式执行追踪可以通过捕捉远程过程调用等通信数据来定位异常应用 [7], 但是深入挖掘异常应用内部的问题根因, 则需要进一步采集异常应用的应用轨迹追踪, 通过复杂应用的函数及指令的前后执行过程实现问题诊断和瓶颈定位.

然而, 在现在复杂的软件混合部署及硬件负载高压的数据中心中实现应用轨迹追踪面临着效率的严峻挑战. 首先, 在多个应用共享资源同时运行的运行环境中 [8], 轨迹追踪需要更昂贵的内核控制操作, 造成的被观测应用性能下降比传统独占式执行环境更加严重. 其次, 在硬件资源利用率逼近上限的高压力运行环境中, 需要执行追踪进行诊断的性能异常发生频率更高, 然而执行追踪带来的性能抖动被进一步放大, 过去轨迹追踪工具产生的可容忍的 2% 开销会造成被观测应用 15% 的严重性能下降, 所以轨迹追踪需要追求极致小于 1% 的运行时开销才能满足在如此复杂的大规模集群中部署的需求. 现有轨迹追踪主要依赖内核态与用户态插桩技术, 通过植入追踪点在运行时产生轨迹 [2,9]. 尽管此类技术被应用开发者广泛用于性能调试, 但这种通过侵入应用获取追踪的方式在准确度和开销方面无法满足数据中心中多变应用的需求.

为高效获取执行追踪, 现代处理器提供了低开销的硬件追踪能力, 但现在基于该能力的系统都无法达到混部高压云环境的效率要求. 以英特尔处理器追踪为代表的硬件追踪能力可以提供不侵入目标应用执行的指令精度轨迹采集 [10,11]. 但是硬件追踪能力的封装设计需要在 4 个关键维度上进行权衡: 被追踪应用的性能下降, 追踪数据的内存开销, 单节点轨迹的覆盖长度, 以及大规模集群的管理复杂度, 这 4 个目标通常相互制约, 无法同时达到最优. 现有利用硬件追踪能力进行事后调试诊断 [5], 控制流完整性安全 [12] 和端到端离线轨迹追踪 [13] 的工作都无法达到混部高压云环境对于在线轨迹追踪工具开销的要求. 同时, 针对大规模数据中心中需要轨迹追踪的异常诊断和性能分析场景, 高效准确地触发与管理不同场景的轨迹追踪节点和配置需要考虑集群层面的开销和收益.

针对上述问题和挑战, 本文提出了 ETrace (efficient tracing facility) 轨迹追踪系统, 面向混部高压数据中心环境基于硬件追踪能力的高效灵活封装实现小于 1% 的轨迹追踪开销, 进而赋能性能异常诊断和性能瓶颈分析. ETrace 包括两个组件: 在节点层面, ETrace-Tool 实现极低轨迹追踪时间开销, 同时动态管理有限内存空间; 在集群层面, ETrace-Manager 根据不同场景灵活高效触发执行追踪工具的运行和处理, 通过云原生编排流程支持统一配置接口便捷访问. 采集的指令追踪轨迹将自动智能整合为可读可用的多维度特征, 帮助云服务提供商与应用开发者可解释的保障与优化应用. 具体而言, ETrace-Tool 通过感知并消除引发性能劣化的硬件追踪关键控制操作实现极低的轨迹追踪开销, 并且基于应用运行状态动态调整追踪的 CPU 核心与内存缓冲区设置, 充分利用有限空间存储更多有用的轨迹信息. ETrace-Manager 根据 3 种场景的需求按需准确触发对应节点的轨迹追踪, 并且通过云原生的控制流和数据流实现自动化的轨迹处理及分析.

本文在阿里巴巴生产集群中实现并部署 ETrace 系统以进行全面测试, 其在多种基准测试集上实现了平均 0.9% 的追踪开销, 相比现有工作最高降低 85% 的开销, 同时集群管理产生的额外开销也大大降低, 未产生性能影响. ETrace 已经应用于大规模集群进行异常诊断和瓶颈分析, 本文基于真实应用的实际案例分析了 ETrace 在诊断未知异常的有效性, 以及在函数和指令层面展示机器学习应用和传统计算应用的关键操作与轨迹特征.

综上所述, 本文的核心贡献总结如下.

- 本文分析了混部高压数据中心中应用轨迹追踪的必要性和更严峻的开销挑战, 并分析了新兴硬件追踪能力无法满足开销要求的四维度设计权衡.
- 本文设计了轨迹追踪系统 ETrace, 基于硬件追踪能力的高效灵活封装, 通过关键操作消除、动态空间管理和灵活触发管理, 在混部高压云数据中心环境实现了极低的观测开销.
- 本文验证了 ETrace 在真实场景中基准测试集上的极低追踪开销和管理开销, 并展示了基于该系统在大规模集群的异常诊断和瓶颈分析方面的新发现.

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

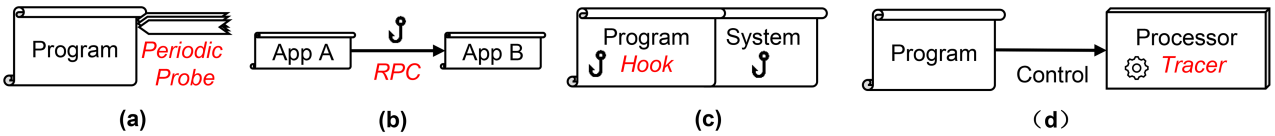


图 1　(网络版彩图) 数据中心现有轨迹追踪的原理展示和方法比较. 红色位置具体产出轨迹追踪.

Figure 1　(Color online) Comparison of current tracing tools in datacenters. Red locations produce traces. (a) Sampling; (b) distributed tracing; (c) instrumentation tracing; (d) hardware tracing.

2　研究背景与相关工作

2.1　数据中心执行追踪可观测性

新兴软件应用的可解释性管理需依赖时序化轨迹信息. 更高的应用复杂度和复杂的混合部署管理需要服务间远程调用通信与应用执行追踪来理解应用特征 [7,13,14]. 同时应用执行极容易受到异常事件的干扰, 如由于硬件故障导致的组件慢性失效 [3] 或由于系统流量过载导致的亚稳态故障 [4] 等, 此类事件会导致数据中心频繁的性能下降, 需要快速准确的根因分析与故障恢复. 因此, 云服务提供商需要提升运行应用的可观测性, 即通过外部输出推断组件内部状态的能力. 如图 1(a) 所示, 传统基于统计数据的可观测性可以通过周期性探针采集平均特征或热点定位解释部分故障 [15], 但是更加复杂, 隐蔽的软硬件故障需要时序化轨迹信息提供的因果关系作为支撑才能实现高效敏捷的诊断恢复 [5].

数据中心执行追踪可观测性系统面临着开销与准确性的权衡. 如图 1(b) 所示, 服务间的分布式追踪已经可以便捷高效地通过远程过程调用 (remote procedure call, RPC) 等通信数据获取服务调用链和定位异常应用服务 [16~18], 例如 Dapper [7] 等系统. 如图 1(c) 所示, 而深入诊断问题应用则需要应用执行追踪, 现在大部分轨迹都是基于插桩方式获取, 即在汇编级 [2]、链接库级 [9] 或系统调用级 [19] 向应用程序或操作系统中插入追踪点实现. 该类工具会侵入应用内部, 且开销与准确性依赖工具开发者的定义, 无法对于性能开销极限优化. 综上, 本文首次在混部高压数据中心环境下, 对追踪可观测性的开销影响进行了量化分析与系统优化.

2.2　处理器硬件追踪能力

为了支撑高精度, 低开销的执行追踪, 现代 CPU 处理器都逐渐引入了硬件追踪能力, 以英特尔处理器追踪技术 (intel processor trace, IPT) [10]、ARM 嵌入式追踪单元 [11] 和 RISC-V E-trace 单元 [20] 为代表, 如图 1(d) 所示. IPT 在硬件级别以数据包的形式对于应用分支跳转信息进行记录. 其对于条件分支以单比特记录分支是否跳转; 对于间接分支记录目标地址; 分支数据包依据预设的特权等级和过滤规则输出到预先定义的专属轨迹缓冲区中. 硬件追踪能力不干扰指令流水线的执行, 以极低的开销获取汇编指令的控制流轨迹.

硬件追踪能力提供的指令精度轨迹被研究人员利用实现不同功能目标. 诊断类工作利用程序崩溃前的执行追踪具体诊断根因问题, 对于单一线程的微秒级数据进行保存, 如 REPT [5] 等; 安全类工作利用实时的指令轨迹进行控制流完整性检查以保护应用不被恶意攻击, 如 Griffin [12] 等; 观测类工作对于应用端到端执行的观测精度进行优化, 如 JPortal [13] 等, 而未关注其在混部高压云环境中的运行开销. 此外, 针对混部场景下的性能观测, 现有代表性工作如 CPI2 [21] 和 Heracles [22] 已进行了深入探索, 但是他们主要依赖硬件性能计数器或统计采样来获取微架构指标, 无法还原执行流来定位具体根因, 因此无法支撑数据中心轨迹可观测性的需求. 综上, 本文首次系统研究了硬件追踪能力在数据中心执行追踪可观测性应用中的性能开销与优化方法.

3　研究动机

3.1　混部高压云环境对于执行追踪开销的新要求

在数据中心中可以大规模部署的应用执行追踪面临的核心挑战就是追踪开销, 即执行追踪必须最小化对于

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

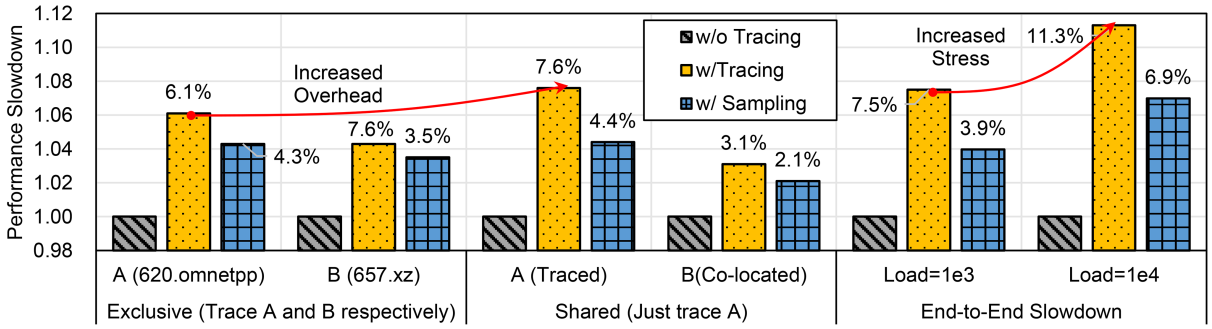


图 2　(网络版彩图) 混部高压云数据中心环境中的执行追踪给被观测应用造成严重性能下降.

Figure 2　(Color online) Tracing causes significant performance degradations in shared and stressed datacenters.

被观测应用性能的负面影响. 与现有方案允许 $\leqslant 10\%$ 数量级的追踪开销不同, 本文发现, 在真实且复杂的软件混合部署, 硬件负载高压的数据中心环境中, 追踪开销需要降低到 $\leqslant 1\%$ 的千分位量级, 具体有两方面原因.

首先, 软件混合部署场景恶化了执行追踪对被观测应用造成的性能下降. 现代数据中心为提升资源利用率, 通常会将不同软件负载部署在相同硬件上运行, 称为混部环境 [23,24]. 混部环境提高了对于应用追踪的开销要求, 因为相比过去工作研究的应用单独部署的独占环境 [15], 应用追踪会产生更高的开销. 为了构建具有代表性的混部环境, 本文选取 SPEC CPU 2017 基准测试集中两个典型应用 A (620.omnetpp) 和 B (657.xz) 混合部署在 4 个处理器核上. 应用 A 代表具有复杂控制流的内存与分支密集型负载, 而应用 B 则代表高 IPC 的计算密集型负载. 本文利用 Perf 工具对于应用 A 进行采样观测和执行追踪. 图 2 可以展示两项关键结论. 首先, 对比前两组柱状数据可以发现, 混部环境中执行追踪的开销相比独占环境中显著增加达 25%; 其次, 对比后两组柱状数据可以发现, 执行追踪开销会波及混部的未观测应用 B, 引发级联的性能下降. 在生产环境中更加激进的混合部署策略下 [8], 执行追踪的开销引发的被观测应用性能下降会进一步加剧.

其次, 硬件负载高压场景放大了执行追踪开销导致性能下降的影响. 数据中心实际运行时会频繁遇到硬件资源使用处于极限状态的情况, 例如 CPU 利用率大于 90%, 称为高压环境. 硬件资源的超额分配 [8], 评估系统性能的全链路压力测试以及 “双十一” 等极端流量峰值等因素都会造成数据中心的高压环境. 相较于负载较低的日常场景, 高压环境发生服务质量违反等性能异常问题的概率更高, 所以更迫切需要执行追踪工具进行问题定位和诊断. 本文使用开源基准程序 DeathStarBench [25] 的 SocialNetwork 应用, 使用 wrk 生成不同用户流量模拟高压环境, 并使用 Perf 工具追踪应用执行链路中的 ComposePost 服务. 如图 2 所示, 在高压环境中, 对于单个服务进行执行追踪产生的约 2% 的开销, 会导致端到端应用的请求时延高达 17% 的性能下降, 并且数据中心更加关注的 99% 长尾延迟受到的影响比 50% 平均延迟更加严重. 而类似场景下如果观测开销降低到 1%, 端到端延迟的性能下降将减少 56%~84%.

为了深入探究混合部署环境中执行追踪的开销上升根因, 本文从应用执行和微架构两个维度对于观测开销进行了剖析. 本文在混合部署应用 A (620.omnetpp) 和 B (657.xz) 的基础上, 额外与应用 C (Sysbench 驱动的 MySQL) 进行混部部署以探究混部应用密度的影响. Perf 工具被用来监测应用 A 的上下文切换、处理器迁移、内核态耗时这 3 类关键软件事件, 以及分支预测错误、L1 缓存未命中、最后一级缓存 (last level cache, LLC) 未命中这 3 类关键微架构指标, 旨在深入归因图 2 中应用 A 在混部环境中性能下降更为明显的现象. 从应用执行视角分析, 如图 3(a) 所示, 混部场景中上下文切换次数激增, 这是竞争有限的计算资源和任务负载特征共同作用的结果, 如果在每次上下文切换执行追踪控制操作, 将引入巨大的观测开销. 相比之下, 应用处理器核间迁移的频率较低. 同时, 执行追踪的控制操作会导致内核态耗时增加, 且控制逻辑越复杂, 其带来的性能下降将进一步加剧. 从微架构角度来看, 如图 3(b) 所示, 混合部署环境由于竞争共享资源会显著影响分支预测准确率和缓存命中率.

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

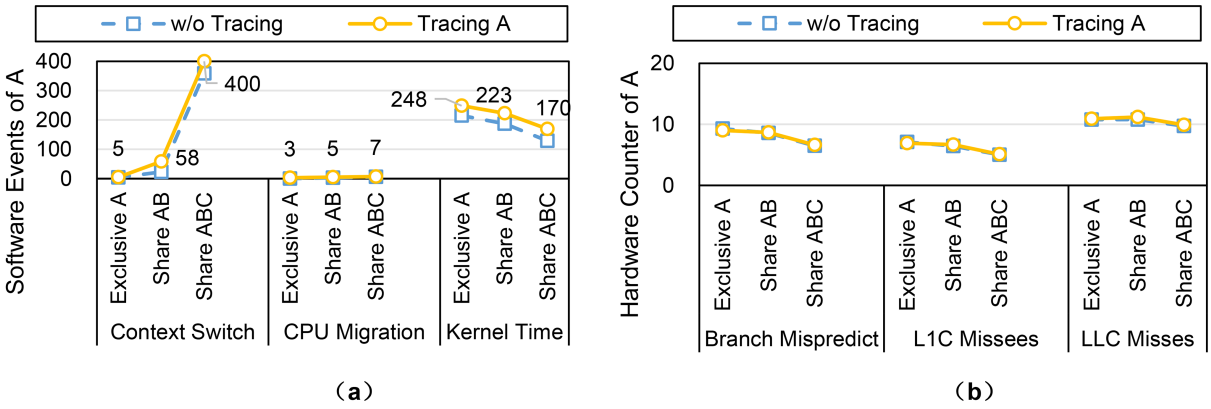


图 3　(网络版彩图) 不同混部配置下应用 A 因执行追踪导致性能下降的 (a) 软件与 (b) 硬件归因分析.

Figure 3　(Color online) Analysis of increasing tracing overheads in shared environments from (a) software execution and (b) hardware perspectives.

3.2　硬件追踪能力面临的四维设计目标权衡

鉴于上述复杂环境对于执行追踪开销的更高要求, 现代处理器引入的硬件追踪能力可以实现极低开销的执行追踪观测. 该技术支持非侵入式的指令级别追踪 [10,11], 并且研究者已经在崩溃后调试 [5]、控制流完整性 [12] 与端到端追踪 [13] 等多个非混部的专用领域应用该能力. 然而, 现有硬件追踪能力的使用远未达到混部高压数据中心可以部署的开销水平, 原因是硬件追踪能力本身存在时间开销、空间开销、轨迹长度与管理难度的四维度设计目标权衡: (1) 时间开销是指对被观测应用造成的性能下降; (2) 空间开销是指存储执行追踪占用的内存空间; (3) 轨迹长度指单个观测节点上轨迹的覆盖时间长度; (4) 管理难度是指大规模集群管理硬件追踪节点的操作复杂度. 受限于硬件设计, 四者无法同时达到最优状态, 部分维度必须作出妥协.

四维设计目标的重要性随应用场景而异, 而现有研究工作均未将时间开销作为首要优化目标. 如图 4 所示, 面向诊断的崩溃后调试方案 [5] 的空间开销重要性最高, 从而部分牺牲时间开销与轨迹覆盖. 其采用每线程 64 kB 的环形缓冲区来记录软件崩溃前数微秒的轨迹数据, 并通过每次上下文切换时的控制操作管理轨迹存储, 因此时间开销高达 5.4%~9.7%, 但是其在集群层面只需要选择诊断节点即可. 面向安全的控制流完整性方案 [12] 的空间开销与轨迹覆盖重要性最高, 导致了显著的时间开销. 其在每次上下文切换时控制硬件追踪单元, 并在轨迹装满缓冲区后频繁转移进行实时控制流对比, 因而时间开销高达 4.8%~18%, 而且其在集群层面需要选择保护节点以及转移数据进行实时比较, 集群管理操作复杂度高于只选择节点. 面向离线观测的端到端追踪方案 [13] 追求全生命周期执行追踪所以轨迹覆盖重要性最高, 从而不可避免地增加时间与空间开销, 时间开销高达 11.3%~16.5%, 同样, 其在集群层面只需要选择观测节点, 管理难度低. 综上所述, 上述场景中设计目标的重要性设定, 与复杂数据中心在线观测场景将时间开销视为最高优先级的需求存在根本性冲突.

进一步分析硬件追踪能力的极低开销未被现有方案充分发掘的原因, 主要是由于追踪控制操作带来的额外性能开销. 不加控制的开启硬件追踪对于应用的开销基本小于 1% [10], 但是在追踪过程中, 用户需操控特定的模型特定寄存器 (model-specific registers, MSR) 以配置追踪精度, 目标应用及缓冲区位置等信息. 更严重的是, MSR 的调整控制必须在关闭执行追踪时执行, 故每次控制操作均需依次执行: (1) 关闭执行追踪; (2) 修改追踪 MSR; (3) 启用执行追踪, 此过程将中断被观测应用的执行以保证轨迹数据的完整性. 此外, 若追踪配置依赖用户态信息, 控制操作需要进行用户态 – 内核态切换, 也增加了额外的性能开销. 同时, 为了尽可能降低最高重要性的时间开销, 设计者需要根据数据中心实际情况, 对于空间开销、轨迹长度及管理难度 3 个重要性较低的设计目标进行重新适配与权衡. 对于空间开销, 鉴于现代服务器普遍配备数百 GB 甚至 TB 级内存容量, 维持 kB 级别的内存占用并非刚需, 在实践中被观测节点通常允许约 1% 的内存占用临时存储应用轨迹, 基于应用需求灵活组织有限的空间以优先保障时间开销是更适合混部高压集群的选择. 对于单节点的轨迹覆盖, 上述两个维度的

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

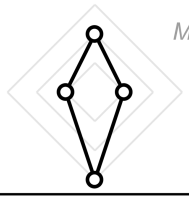
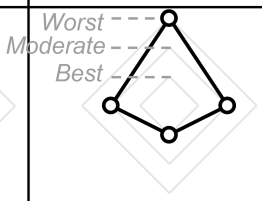
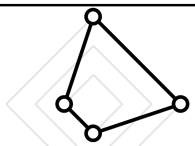
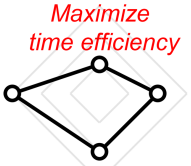
Objectives	Debugging	Security	Offline Trace	Online Trace
Implementation	Post-debugging REPT [5]	Control flow integrity Griffin [12]	End-to-end tracing JPortal [13]	On-demand tracing ETrace
4-dim Goal Time efficiency Mgmt. difficulty Space overhead Data coverage				
Time efficiency	5.4%	4.8%	11.3%	<1%
Space overhead	100KB	100MB	10GB	<1GB
Data coverage	Microsecond-level	Constant	Hour-level	Millisecond-level
Management (Mgmt.) Difficulty	Low	Medium	Low	High

图 4　(网络版彩图) 硬件追踪能力的四维设计目标权衡. 标红的对象为重要性最高的设计目标.

Figure 4　(Color online) Four-dimensional design considerations of hardware tracing capabilities. Red ones are highest priority goals.

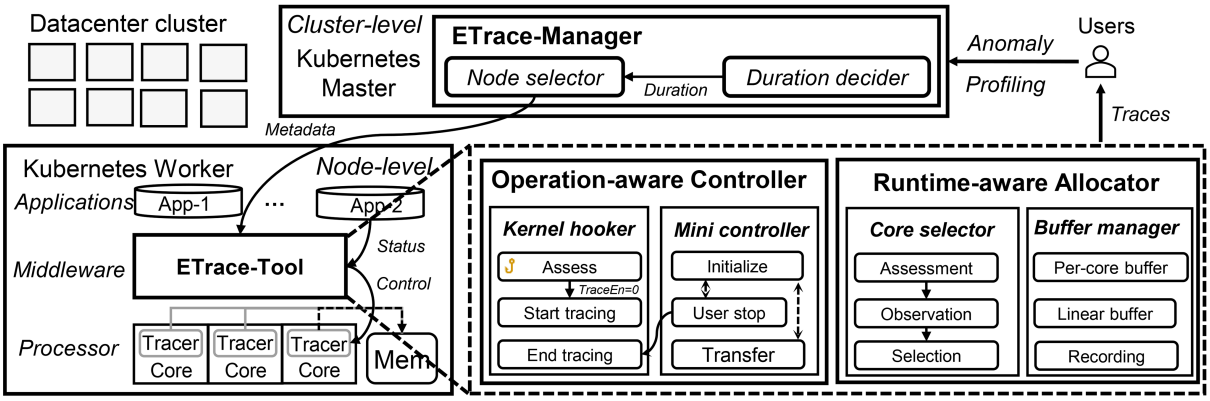


图 5　(网络版彩图) ETrace 系统的总体设计和工作流程.

Figure 5　(Color online) Design overview and workflow of ETrace.

要求导致只能覆盖毫秒 – 秒级别的数据, 对于性能异常场景已经足够诊断根因, 对于性能分析场景则通过集群层面多节点的重复追踪来刻画应用完整画像. 间断式追踪在集群层面需要进行节点的准确选择和轨迹的高效转移, 集群管理难度高于之前的设计方案.

4　ETrace 系统设计

基于上述分析, 本文设计了面向混部高压云环境的高效执行追踪工具 ETrace. ETrace 目标是通过硬件追踪能力的高效封装, 实现复杂场景的执行追踪采集处理, 进而赋能性能异常诊断和性能瓶颈分析. 如图 5 所示, ETrace 包括两个组件: (1) 节点层面的高效执行追踪工具 ETrace-Tool, 负责指定观测节点和应用的高效轨迹采集, 最优化时间和空间效率; (2) 集群层面的高效追踪管理工具 ETrace-Manager, 负责处理用户输入并准确触发执行追踪工具的运行与处理, 最优化集群开销与数据覆盖度.

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

4.1　ETrace-Tool 设计

ETrace 工具在节点层面的执行追踪组件 ETrace-Tool 的设计目标是, 基于处理器提供的硬件追踪能力, 最小化对被观测应用的性能影响并最大化指定内存空间的存储轨迹信息. 为了达成以上两个目标, ETrace-Tool 包含关键操作感知的采集控制器和应用运行感知的内存分配器, 前者负责追踪过程的最小化主动控制, 后者负责内存缓存区的智能分配与管理.

采集控制器通过减少昂贵的关键控制操作实现轻量化主动控制策略. 相较于频繁启停硬件追踪, ETrace-Tool 识别出影响被观测应用性能的关键寄存器切换操作, 传统方案持续将追踪数据写入每线程缓冲区, 依赖故障信号等事件终止追踪, 需在每次上下文切换时修改 MSR 以调整追踪设置, 产生大量冗余高成本操作. 在真实数据中心节点, 多数核心与线程的上下文切换间隔低于 1 ms, 意味着传统方案在秒级追踪周期中的操作开销巨大. 同时, 此类设计还会增加集群稳定性风险, 因为频繁的 MSR 修改在运行时可能引发服务器故障停机.

采集控制器的运行流程如图 5 所示, 首先, 内核挂钩单元在上下文切换跟踪点 (sched_switch) 注入钩子操控追踪器. 当目标进程调度至核心 C 时, 钩子通过设置 MSR 启用 C 的硬件追踪; 当进程被调度出时则不作操作, 自然规避重复启用产生的冗余操作. 进程过滤机制阻断目标进程调出后无关线程的无用数据采集, 该设计将高成本追踪控制操作从上下文切换次数压缩至处理核心数量, 显著降低运行时开销. 追踪控制单元接收执行追踪请求后将追踪参数下发给内核挂钩单元, 之后主动监控追踪过程, 采用高精度定时器限制追踪周期, 并根据用户请求传输追踪数据包供分析. 采集控制器全程运行于内核态, 避免用户态 – 内核态切换额外开销. 追踪结束时, 内核钩子会自动禁用所有核心追踪器, 防止无限追踪并提升系统鲁棒性.

内存分配器采用动态分配和管理轨迹缓冲区实现有限内存空间的高效利用. 硬件追踪器需在追踪开始前为 CPU 核心分配固定内存空间. 若为全部 128 个 CPU 核分配最大 128 MB 的缓冲区, 将导致高达 16 GB 的内存浪费 [13]. 鉴于数据中心节点可分配的内存空间有限, 追踪工具通常只有约 1 GB 的内存空间可使用, 均等分配至所有 CPU 核将缩减单个核心可采集的轨迹信息. 因此, 必须精准分配内存至实际执行任务的核心.

内存分配器的运行流程如图 5 所示, 追踪核心选择单元首先基于采集控制器初始化追踪时的应用元数据, 动态确定实际追踪的处理器核. 应用分为两种 CPU 资源分配模式: 绑核应用独占若干个 CPU 核; 混部应用与其他应用共享 CPU 核. 针对绑核应用, 追踪的 CPU 核心就是分配的核心, 并采用等分内存分配策略; 针对混部应用, ETrace-Tool 设计了核采样机制来动态选择追踪的 CPU 核, 包含应用当前执行的 CPU 核及随机选择的 CPU 核, 因为低利用率的 CPU 核更容易被调度所以分配更大的轨迹缓冲区. 其次, 缓冲区管理单元为每个追踪的 CPU 核而非线程分配内存缓冲区, 以降低 MSR 操作频率, 同时采用线形缓冲区替代传统环形缓冲区 [5, 12]: 一方面当缓冲区满载时丢弃额外数据, 确保记录更接近性能异常的相关追踪数据, 同时严格控制内存用量. 针对多线程应用追踪需要的线程调度信息, ETrace-Tool 在上下文切换跟踪点注入的钩子程序记录五元组上下文信息 [时间戳, CPU 核, 进程, 线程, 操作类型], 每次切换生成 24 字节数组辅助多线程追踪分析.

4.2　ETrace-Manager 设计

ETrace 工具在集群层面的追踪管理组件 ETrace-Manager 的设计目标是, 基于用户不同场景的请求, 高效触发目标节点的轨迹追踪并准确设置轨迹追踪节点的参数下发. 目前 ETrace 支持 3 种数据中心场景的按需触发: 异常指标自动触发、用户请求检测异常、集群日常性能评估. 不同场景的触发条件和节点配置都需要根据实际场景进行设置以达到集群层面的开销优化.

具体来说, 异常指标自动触发是基于节点层面的软件性能监控单元和硬件计数器监控单元对于系统状态的实时监测和上报, 由集群层面的采集器基于已有规则判断异常状态进行告警, 例如业务平均延时超过用户设置服务质量阈值等. 异常指标自动触发会按照默认观测时间长度 0.1 s 触发所有异常节点的轨迹追踪. 用户请求检测异常是基于用户主动发起的诊断请求触发性能异常应用的轨迹追踪. ETrace 配备了可通过 yaml 文件访问的统一查询接口, 应用开发者或运维工程师可以便捷提交对于指定应用, 指定时长的观测请求. ETrace 对于指定应用进行轨迹追踪后自动将解析后可读的结果返回给用户定位异常根因.

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

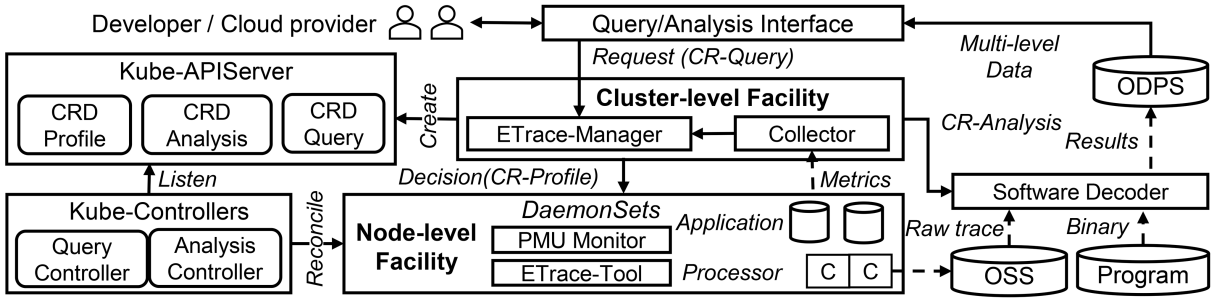


图 6　ETrace 系统在数据中心云原生管理流程中的部署方式.

Figure 6　Cloud-native deployment of ETrace in large-scale datacenters.

不同于前述异常场景, 集群日常性能评估聚焦于系统正常运行状态, 使用 ETrace 定位潜在瓶颈以辅助系统优化. 在没有性能异常的场景, 应用的执行特征高度相似 [15], 所以不加区分的观测所有节点上的目标应用将导致未产生价值的观测开销. 因此, 需要准确选择轨迹追踪的时间和节点以降低成本. 首先, 时间采样模块会基于目标应用的复杂度选择合适的观测时间, 复杂度越高需要越长时间的轨迹追踪以覆盖其执行过程, 观测时间一般设置在 0.1∼2 s 的范围内. ETrace 采用应用优先级, 二进制文件大小及历史异常数量 3 项指标加权计算得到应用复杂度从而确定轨迹追踪时间长度. 其次, 节点采样模块会选取不同数量的目标节点触发轨迹追踪. ETrace 采用应用优先级与部署机器数量两项指标加权计算追踪节点数, 观测节点数一般设置为 3∼10 台部署相同应用的机器. 集群日常性能评估的采集方式会有大量冗余执行追踪, ETrace 会合并冗余数据并综合各节点的执行追踪得到完整的应用执行行为.

本文将 ETrace-Manager 集成至基于 Kubernetes 的云原生数据中心管理流程, 如图 6 所示. 在控制流方面, 用户请求 (query)、追踪配置 (profile) 和解析配置 (analysis) 被封装为 Kube-APIServer 中的定制资源 (custom resource definitions, CRD). 用户请求和轨迹采集配备独立控制器 Kube-Controller 以完成对应任务, 将用户请求与轨迹采集进行解耦, 用户请求的主调谐链路负责根据用户请求生成具体的轨迹追踪配置, 同时增加子调谐链路负责具体节点的轨迹追踪任务, 具体方法是在每个节点维护一个延迟队列来对轨迹追踪任务项进行不同任务的功能需求. 在数据流方面, 为了降低轨迹转存的内存与文件 I/O 开销, 硬件产生的轨迹追踪数据无需保存到节点硬盘, 而是直接上传至云端非结构化对象存储服务 (object storage service, OSS). 软件解码器负责从原始轨迹追踪数据中重建控制流, 解码器从 OSS 获取源数据并从软件源码仓库提取应用二进制文件辅助解码, 解码后的用户可读结果直接输出至云原生结构化数据存储 (open data processing service, ODPS) 供所有用户便捷查询分析. 同时, 用于执行流重建的软件解码器变得更加简洁高效, 其工作流程包含 3 阶段: 首先定位追踪数据中的心跳同步包 PSB; 随后同步追踪流并扫描控制流数据包; 进而执行各类型数据包的处理流程并循环至同步阶段.

5　实验分析

5.1　ETrace 实验环境

本文在 IceLake 微架构的双路 32 核 Intel 至强铂金 8369B 处理器平台测试 ETrace 系统, 配备 1 TB 内存及 Linux 4.19.91 系统, 应用的执行时间和吞吐量等指标通过日志输出获取, 系统的利用率等系统指标通过节点监控进程采集. 所有评估指标均为 10 次重复实验的平均值以降低结果偏差. 本文使用 SPEC CPU 2017 整数测试集因为其可以离线端到端运行并且能代表高负载场景下的计算密集型任务, 同时选用 3 类访存密集型的在线基准测试应用: (1) Memcached 内存缓存系统通过 Memtier 工具模拟 10 个读写比 1:1 的并发客户端; (2) Nginx 网络服务器通过 Apache 基准工具模拟 10 个并发客户端持续发送 2 万次请求且每次请求 20 字节文件; (3) MySQL 数据库通过 Sysbench 工具在 10 张百万行级数据表上模拟读写请求.

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

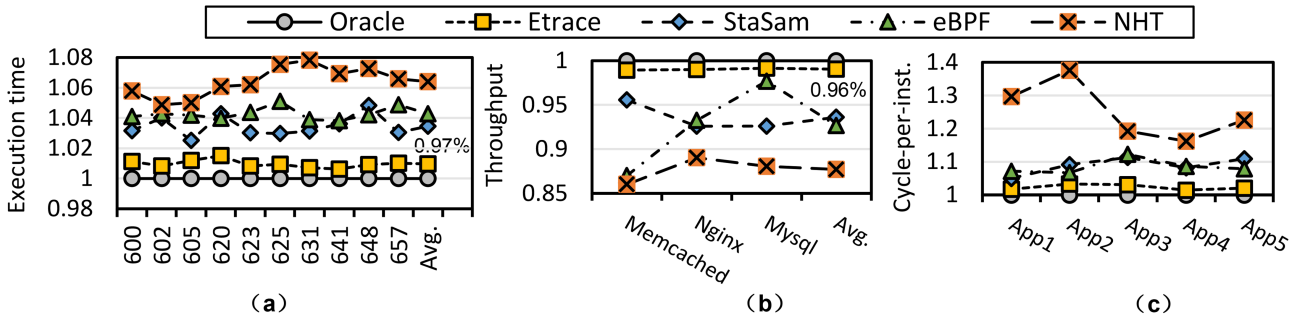


图 7　(网络版彩图) 与现有工具相比, ETrace 进行轨迹追踪对于多种类型应用在高负载、混合部署的复杂场景中造成了极低的开销, 所有结果都归一化到未进行轨迹追踪的性能.

Figure 7　(Color online) ETrace induces the lowest tracing overheads on all benchmarks under shared and stressed scenarios compared to baselines. All the results are normalized to performance without tracing. (a) Compute-intensive; (b) memory-intensive benchmarks; (c) real-world applications.

本文选择了 3 种基线方案与 ETrace 进行比较. 所有轨迹追踪工具在应用运行全程保持开启. (1) StaSam 采用非时序采样方法, 以默认 4k 频率记录统计事件 [19]; (2) eBPF 为基于 eBPF 的用户态追踪方案, 通过 bpftrace 工具记录系统调用入口事件 [26]; (3) NHT 采用当前对于硬件追踪能力的使用方案, 由开源 Perf 工具 [19] 实现; 这 3 种基线方案代表当前数据中心的主流观测方法. 由于基于硬件追踪能力进行应用的论文 [5,12,13] 与 ETrace 的设计目标存在差异且闭源难以公平复现, 故本文主要基于其论文中展示的实验结果进行对比.

5.2　ETrace 开销实验结果

本文首先测试了 ETrace 系统中的 ETrace-Tool 进行节点层面的轨迹追踪对于被观测应用造成的时间开销. SPEC CPU 应用的性能下降通过端到端执行时间下降表示, 而访存应用的性能下降通过吞吐量的下降表示. 使用各种轨迹追踪方案进行观测的性能与不进行轨迹追踪的应用性能 (Oracle) 进行归一化以方便比较. 如图 7(a) 所示, ETrace 在整个 SPEC CPU 测试集引发的性能开销为 0.4%∼1.5%, 平均性能开销为 0.97%. 相比于 StaSam, eBPF 与 NHT 基线, ETrace 分别平均减少 72%, 77% 与 85% 的轨迹追踪开销. 对于在线基准测试应用方面, 通过图 7(b) 的吞吐量指标发现 ETrace 的平均性能开销仅为 0.96%, 相比 3 类基线的追踪开销分别降低 84%, 86% 与 92%. 同时, 相较于计算密集型 SPEC CPU, 访存密集型的在线应用对轨迹追踪的开销更加敏感, 因为性能下降会引发后续查询的连锁性问题. ETrace 的平均性能开销相比于诊断类工作的 5.25% [5], 安全类工作的 4.8% [12] 和观测类工作的 11.3% [13], 均有大幅度下降. 在计算密集型的 SPEC CPU 上, ETrace 在 omnetpp 场景最差造成了 1.5% 的观测开销; 而在内存密集型的在线应用, ETrace 在 memcached 场景最差造成了 2.1% 的观测开销; 而如图 7(c) 所示的数据中心真实应用高流量场景中, ETrace 对于不同应用平均造成 1.1% 的性能下降, 最差造成 3.2% 的性能下降.

本文接下来测试了 ETrace 两个组件自身执行和管理的开销. ETrace-Tool 自身占用的 CPU 时间在工具启动阶段安装追踪内核模块时出现 0.05 核心占用的瞬时峰值, 其他执行时间的开销都小于 0.02 核心, 符合大规模集群的性能开销要求. ETrace-Tool 部署的 DaemonSet 开销. 在两次轨迹追踪的时间点, 开销会出现 0.09 核心占用的瞬时峰值, 不进行追踪时的开销约为零, 内存开销保持在 35 MB 左右. ETrace-Manager 的管理开销在异常触发轨迹追踪的时间点, 管理容器开销低于 0.003 核心, 在周期性性能评估的时间点, 管理容器开销低于 0.002 核心, 内存开销保持在 40 MB 左右. 综上, ETrace 系统能够实现大规模数据中心的轨迹追踪和触发管理的开销需求.

5.3　ETrace 使能数据中心执行追踪的真实应用实例

极低开销的 ETrace 系统可以支持大规模数据中心的透明部署进行轨迹采集. 本文将针对阿里巴巴生产集群中的典型智能应用, 从性能异常诊断和性能瓶颈定位两个关键管理任务展示使能 ETrace 后的新发现.

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

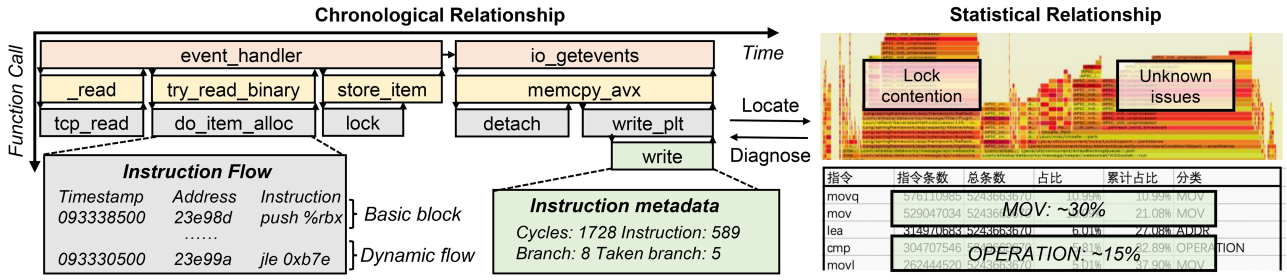


图 8　(网络版彩图) ETrace 采集的时序轨迹数据在指令层面和函数层面的展示, 以及与统计数据的补充关系.

Figure 8　(Color online) Instruction-level and function-level traces derived from ETrace raw data and its relationship with statistical samples.

5.3.1　性能异常诊断

本文首先展示 ETrace 可以获取的轨迹追踪数据内容, 如图 8 所示. 通过解析采集的分支跳转包, 得到指令层面的原始数据, 再聚合得到函数层面的轨迹数据. 指令层面可以获取目标应用的完整指令顺序流, 由静态基本块和动态控制流构成; 同时会获取到指令层面的性能指标, 包括周期、指令数、分支数等. 指令数据聚合可以得到应用函数的执行顺序, 相比于只有时间比例的统计结果, ETrace 提供的函数轨迹包含执行顺序的时序关系. 在实际使用中, 可以使用易于辨别问题的统计比例关系定位到问题, 如图中所示的锁竞争问题和指令比例; 然后使用轨迹数据对于未知问题进行诊断, 精确定位性能异常根因, 并为性能优化提供新方向.

数据中心管理者利用上述 ETrace 采集的轨迹数据可以对于异常的因果关系进行精确定位. 例如在推荐应用中, 传统监控虽能捕获执行时长, 并发线程数及磁盘 I/O 等指标的异常波动, 却无法定位异常根因. 通过 ETrace 追踪其异常执行过程可以发现某次 file_write 系统调用耗时 3.7 s, 多个系统调用因等待互斥锁陷入阻塞. 将系统调用定位至用户函数后, 溯源到同步写日志操作因磁盘 I/O 阻塞. ETrace 由此解释指标异常机制: 同步日志线程阻塞所有混部线程的日志写入操作, 导致响应时间与线程数量异常. 未来可通过隔离相似应用的磁盘资源或修改日志逻辑优化调度机制. 需特别说明的是, ETrace 侧重提升追踪效率而非诊断精度, 此类异常也可以通过高开销方案 [5, 9, 13] 诊断. 但高效的 ETrace 已成功解决诸多曾缺乏高效工具的异常案例.

5.3.2　性能瓶颈定位

基于 ETrace 进行的性能瓶颈定位主要聚焦在两个层级: (1) 函数级分析可以定位用户态与内核态的热点函数, 以供开发者洞察和优化; (2) 指令级分析可以在粗粒度的函数级分析基础上, 提供微架构和指令执行的具体细节, 用于支撑特定体系结构的探索. 图 9(a) 从集群视角展示 CPU 处理器周期的函数分布随着时间的变化, 占用约 25% 周期的具体业务函数除外. 相较于谷歌 [27] 与脸书公司 [28] 的前期研究, 本文基于 ETrace 有 3 项新发现: (1) 数据中心税类函数在阿里巴巴集群中占比较低, 包括的压缩操作 (compression, COMP)、序列化操作 (serialization, SERIA)、远程过程调用操作 (remote procedure call, RPC) 以及调度操作 [29] 总计仅占 3.7%, 远低于谷歌公司报告的 13% [27]; (2) 多线程间同步操作 (synchronization, SYNC) 占据的周期比例高达 16.6%, 远高于脸书公司报告的 2%∼10% [28]; (3) 新型内存技术相关函数分布 [30]: 远程直接内存访问 (remote direct memory access, RDMA) [31] 占 0.5%, 持久内存 (persistent memory, PM) [32] 占 0.2%. 利用高效工具观测不同集群中的关键函数并进行画像引导优化对于提升应用性能和资源利用率十分关键.

其次, 本文选取 5 类代表性应用对于 CPU 运行周期分布的差异性进行分析, 其中包括两个传统应用: (1) 基于 Havenask 引擎, CPU 密集型的搜索应用 (search, Se), (2) 基于 iGraph 系统, 内存密集型的图缓存应用 (cache, Ca); 及 3 个阿里巴巴电商的机器学习应用: (3) 基于 RTP 引擎的预测应用 (prediction, Pr); (4) 基于 BE 引擎的召回应用 (matching, Ma); (5) 基于机器视觉应用平台的推荐应用 (recommend, Re). 图 9(b) 展示了主要函数类别的分布规律, 其中包括 3 项关键发现: (1) 召回应用在内存操作与 C 语言库函数上消耗大量处理器周期, 需要对于内存分配和语言执行进行专门优化; (2) 缓存应用在互联操作 (RPC, COMP) 与新型内存操作 (RDMA,

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

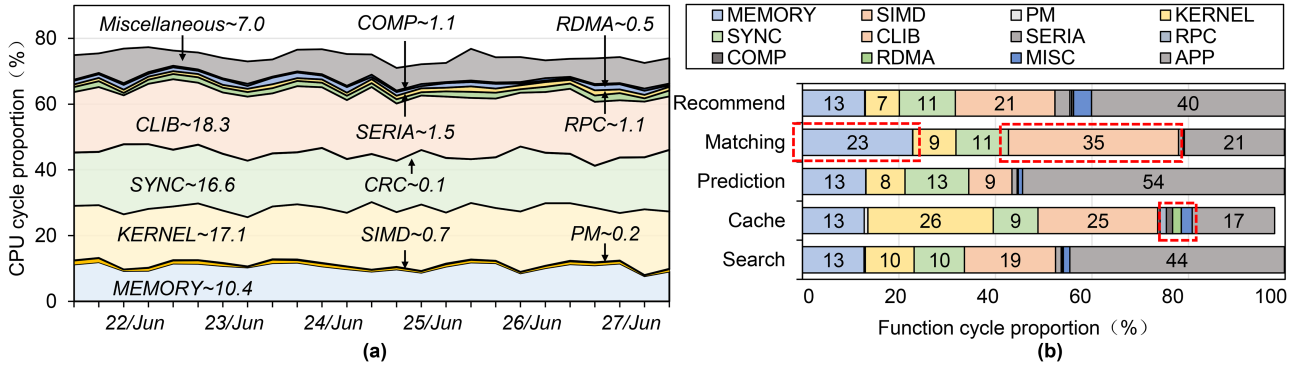


图 9　(网络版彩图) ETrace 采集数据得到的 (a) 集群维度和 (b) 应用维度的不同函数的处理器周期分布.

Figure 9　(Color online) ETrace can breakdown and analyze CPU cycle distributions at the function level from (a) cluster temporal and (b) key application perspectives.

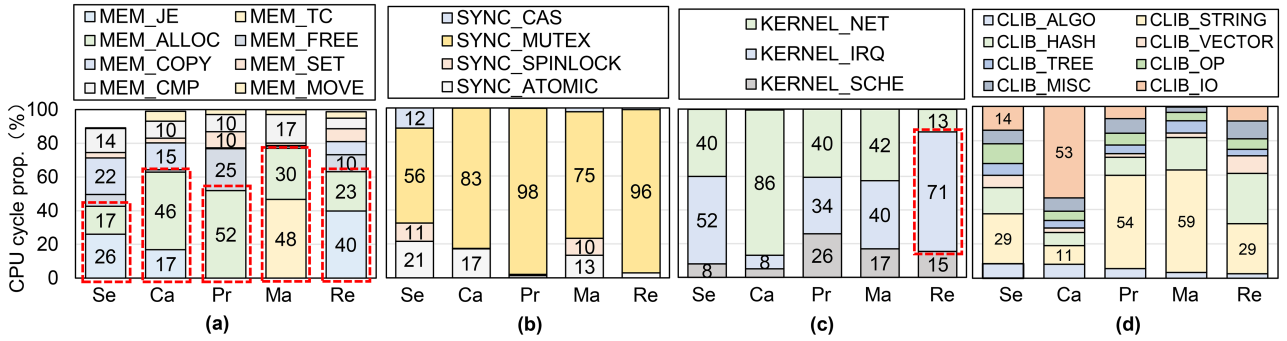


图 10　(网络版彩图) ETrace 采集数据的不同关键函数操作的瓶颈下钻分析. (a) 内存函数; (b) 同步函数; (c) 内核函数; (d) C 库函数.

Figure 10　(Color online) Function-level investigation into key categories of (a) memory operations, (b) synchronization operations, (c) kernel operations, and (d) C library operations.

PM) 上开销更大, 使用高速互联及内存技术可以有效加速其执行; (3) 多线程间同步操作由于应用并行度提升正逐步成为 5 类关键应用的共性瓶颈.

在粗粒度函数类别的基础上, 本文进一步分解洞悉关键函数类别的分布情况. 图 10 对于占比较高的内存操作 (memory)、同步操作 (sync) 和内核相关操作 (kernel) 进行了解构分析, 发现机器学习应用与传统应用有较大差异. 对于内存操作, 内存分配器和内存分配操作的开销占比表明, 高性能的内存分配器 jemalloc 与 tcmalloc 在除预测应用外的各个应用中广泛使用, 应用根据对内存碎片和分配速度的需求差异化使用, 但是内存分配仍是关键优化瓶颈, 需要持续开发更高效的内存分配器; 对于同步操作, 多线程同步操作的增加表明互斥锁 (mutex) 的开销增加, 而原子操作 (atomic) 与自旋锁 (spinlock) 的变化较小; 对于内核操作, 推荐应用的中断操作 (irq) 显著高于其他应用, 是由于其重度多线程设计导致重调度终端频发, 进而触发互斥同步和中断周期上升, 这为设计专用加速器及系统扩展以优化推荐类应用提供新机遇.

基于 ETrace 在函数层级的分析数据揭示了运行新兴应用集群的优化机会, 但是 ETrace 支持的指令层级的分析可进一步补充粗粒度函数级分析的不足, 实现硬件视角的深度性能分析. 基于 XED 指令功能分类将数百类汇编指令归并为 36 个类别, 图 11(a) 展示主要 8 类指令的统计数量. 包括 3 项关键发现: (1) 5 个关键应用的主要指令均为数据访问类 (datatransfer) 比如 movx 指令, 但是缓存应用中占比最低而搜索应用中占比最高; (2) 机器学习应用的逻辑判断和指令跳转指令显著高于传统应用, 表明了应用更复杂的执行逻辑和分支预测的优化空间; (3) 缓存应用的 widenop 指令比例显著高于其他应用, 是由于更频繁的内存/流水线对齐操作导致, 可以进一步优化. 占比最高的数据访问类指令包含内存和寄存器的操作, 根据不同指令访问数据位宽的不同, 在图 11(b)

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

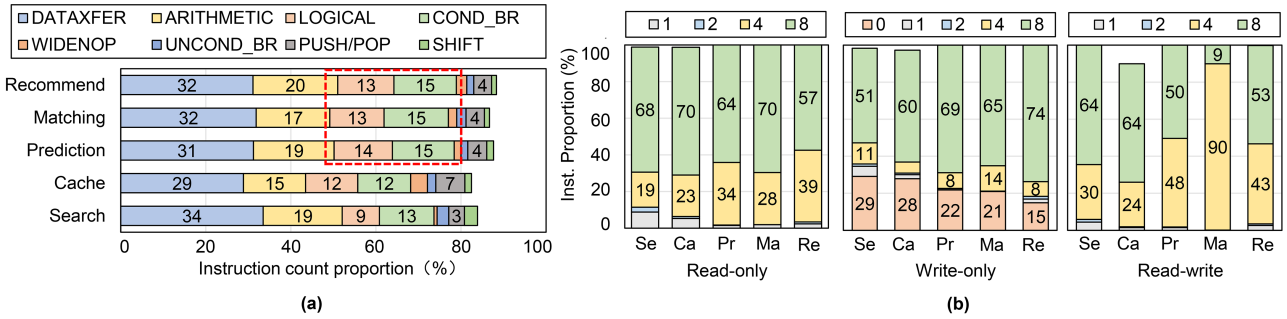


图 11　(网络版彩图) ETrace 支持针对 5 个关键应用在指令层面的 (a) 指令种类分布和 (b) 内存指令宽度分析.

Figure 11　(Color online) Instruction-level analysis of five key applications from (a) instruction category distributions and (b) memory access bandwidth differences.

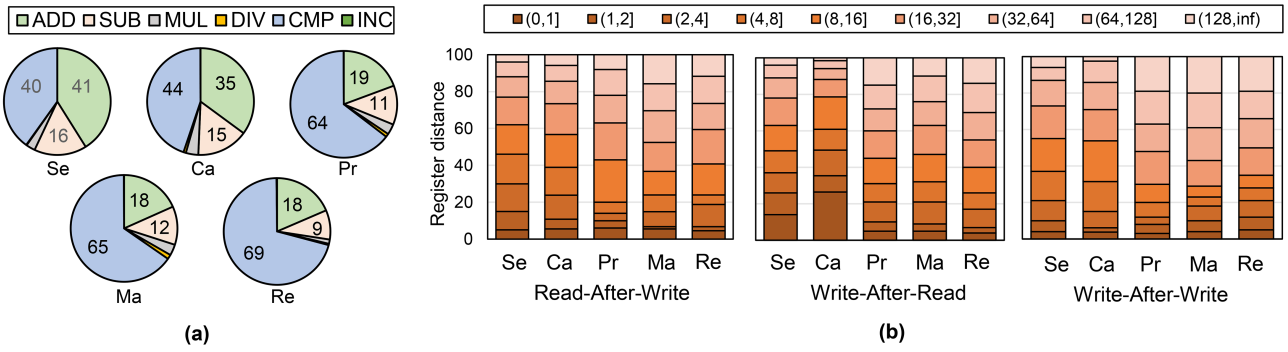


图 12　(网络版彩图) ETrace 支持针对 5 个关键应用在指令层面的 (a) 算术指令分布和 (b) 寄存器依赖距离分布分析.

Figure 12　(Color online) Instruction-level analysis of five key applications from (a) arithmetic instruction distributions and (b) register dependency distance of three cases.

中展示了 3 种访问模式操作的位宽分布, 包括读取指令、写入指令和读写指令. 3 个机器学习应用的四比特位宽内存访问显著高于传统应用, 该现象源于高吞吐推理服务中的量化数据精度妥协. 另外, 机器学习应用中计算内存地址的 lea 指令, 即写入位宽为零的比例低于传统应用.

本文进一步基于 ETrace 采集的指令轨迹分析了不同应用的计算操作 (ARITHMETIC) 分布和寄存器依赖距离情况, 如图 12 所示. 机器学习应用的比较指令 (CMP) 比例显著高于传统应用, 原因可能是大量的特征值比较操作和条件逻辑分支, 可以设置单独的编译优化路径对于该类应用的关键指令进行优化. 图 12(b) 展示了不同应用的写后读 (read-after-write, RAW)、读后写 (write-after-read, WAR) 和写后写 (write-after-write, WAW) 寄存器依赖距离, 该指标通过程序执行过程中同一寄存器两次使用之间的指令数计算得到, 可以用于指导编译器优化和微架构设计 [33,34]. 相比于传统应用, 机器学习应用的寄存器依赖距离更长, 因为其对于数据的连续操作远少于传统应用, 尤其是写后写类型的操作, 此类新的数据访问模式可以指导增加指令并行度和更激进的循环优化, 以及调整乱序执行窗口和寄存器重命名器的大小.

6　讨论与展望

6.1　硬件追踪四维设计目标的复杂性

本文中对于硬件追踪能力四维设计目标的重要性判断取决于应用场景, 但被观测应用的特征也决定了四维参数的量化表现. 具体而言, 时间开销主要由资源竞争程度决定 —— 内存密集型应用因频繁的上下文切换而导致更高的追踪介入损耗; 空间开销则受控于控制流密度 —— 高频跳转的应用会生成海量追踪包, 从而剧增内存占用; 这也直接导致轨迹长度受限, 因为在固定内存空间内, 轨迹长度与空间开销呈严格的反比关系. 相比之

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

下, 管理难度更多地受制于集群规模与使用模式, 对单体应用类型的敏感度较低. 此外, 在时间开销维度之外, ETrace 也可以通过灵活的参数化配置适配不同的优化指标. 例如, 为了实现更好的轨迹覆盖, ETrace 可以增大每核缓冲区来追踪更久的应用轨迹, 在这种情况下, ETrace 的主动追踪控制仍然通过对于昂贵控制操作的识别与消除, 实现了更好的时间开销.

6.2　多架构服务器的普适性

针对支持 ARM ETM 等硬件追踪能力的异构服务器, ETrace 的设计仍然具有普适性. 这主要归因于以下两个方面. (1) 硬件机制的相似性: ARM ETM 与 Intel PT 在底层机制上高度类似, 两者均采用基于跳转包的控制流执行追踪, 且在运行时都需要频繁交互硬件配置寄存器以动态调整追踪状态. (2) 设计方法的通用性: ETrace-Tool 提出的主动采集控制器和动态内存分配器是独立于指令架构的系统级设计, 其核心在于对于关键控制操作的识别消除和有限内存空间的高效利用, 这些优化策略在 ARM 平台上同样有效. 在具体移植路径上, ETrace-Tool 需要根据 ARM 指令集适配底层的驱动接口和寄存器操作单元, 并将 Intel 平台的 ToPA 内存管理机制迁移到 ARM 平台的 ETR (embedded trace router) 内存组织模式.

6.3　复杂异常场景的适应性

针对复杂异常场景, ETrace 也具有较强的适应能力. 首先, 在极端突发流量的复杂场景中, 当系统面临硬件资源接近饱和且上下文切换频率剧增的挑战时, ETrace 采取的主动追踪控制有效避免了高频切换控制操作带来的追踪开销上升. 同时, ETrace 采取的每核缓冲区管理策略避免了对于内存资源的过度使用, 但是在所有处理器核心都满负载运行的极端流量下, 核采样策略可能导致部分核心的追踪数据丢失, 其次, 在多硬件故障叠加的异常场景中, 尽管 ETrace 无法直接修复硬件故障, 但它能精准捕捉硬件故障引发的软件行为差异. 此外, ETrace 独立于应用运行, 确保只要处理器核心尚能执行指令, 就能保留故障发生前的最后现场, 这对于分析多故障叠加的死锁或崩溃尤为关键.

7　全文总结

复杂的数据中心环境需要深度理解共享高压场景下的性能异常机制, 其核心挑战在于实现极低时间开销的高效应用轨迹追踪工具. 本文系统分析复杂场景下追踪开销的造成原因, 提出 ETrace 系统, 基于硬件追踪能力构建的高效应用轨迹追踪系统. ETrace 在节点层面优化控制操作与动态管理空间, 在集群层面实现灵活追踪编排. 通过真实场景中的实验验证, ETrace 在不同的基准测试集上实现平均 0.97% 的性能开销, 相比现有轨迹追踪工具提升达 6.6 倍. 展望未来, ETrace 实现的高效应用轨迹追踪可观测性, 能通过可解释的异常诊断和瓶颈定位等增强大规模数据中心的可管理性.

参考文献

1 Zhang L, Qian Y, Zhou L, et al. CoVoMix: advancing zero-shot speech generation for human-like multi-talker conversations. In: Proceedings of the 38th International Conference on Neural Information Processing Systems, 2024. 100291–100317

2 Luo Y, Rodrigues K, Li C, et al. Hubble: performance debugging with in-production, just-in-time method tracing on android. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022. 787–803

3 Gunawi H S, Suminto R O, Sears R, et al. Fail-slow at scale: evidence of hardware performance faults in large production systems. ACM Trans Storage, 2018, 14: 1–26

4 Huang L, Magnusson M, Muralikrishna A B, et al. Metastable failures in the wild. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022. 73–90

5 Cui W, Ge X, Kasikci B, et al. REPT: reverse debugging of failures in deployed software. In: Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, 2018. 17–32

6 Wang X, Hou X, Li C, et al. EXIST: enabling extremely efficient intra-service tracing observability in datacenters. In: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2025. 355–372

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

7 Sigelman B H, Barroso L A, Burrows M, et al. Dapper, a large-scale distributed systems tracing infrastructure. 2010. https://diyhpl.us/~bryan/papers2/distributed/distributed-systems/dapper.2010.pdf
8 Lu C, Xu H, Ye K, et al. Understanding and optimizing workloads for unified resource management in large cloud platforms. In: Proceedings of the 18th European Conference on Computer Systems, 2023. 416–432
9 Weng L, Huang P, Nieh J, et al. Argus: debugging performance issues in modern desktop applications with annotated causal tracing. In: Proceedings of USENIX Annual Technical Conference (USENIX ATC 21), 2021. 193–207
10 Intel. Intel® 64 and IA-32 architectures software developer's manual. System Programming Guide, 2025
11 ARM. Embedded trace macrocell architecture specification: ETMv1.0 to ETMv3.5. 2011. https://developer.arm.com/documentation/ihi0014/latest
12 Ge X, Cui W, Jaeger T. GRIFFIN: guarding control flows using intel processor trace. SIGPLAN Not, 2017, 52: 585–598
13 Zuo Z, Ji K, Wang Y, et al. JPortal: precise and efficient control-flow tracing for JVM programs with Intel processor trace. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021. 1080–1094
14 Liu D, Zhang L, Xu Y C, et al. Power synchronization: taming massive diversified serverless functions under power constraints. Sci China Inf Sci, 2024, 68: 132101
15 Ren G, Tune E, Moseley T, et al. Google-wide profiling: a continuous profiling infrastructure for data centers. IEEE Micro, 2010, 30: 65–79
16 Wang X, Li C, Zhang L, et al. Exploring efficient microservice level parallelism. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022. 223–233
17 Luo S, Xu H, Lu C, et al. Characterizing microservice dependency and performance: alibaba trace analysis. In: Proceedings of the ACM Symposium on Cloud Computing, 2021. 412–426
18 Fang H, Li C, Wang Q, et al. A method of microservice performance anomaly detection based on deep learning. J Comput Res Dev, 2024, 61: 600–613
19 Gregg B. Linux performance. 2025. https://www.brendangregg.com/linuxperf.html
20 OpenXiangShan. XiangShan. 2025. https://github.com/OpenXiangShan/XiangShan
21 Zhang X, Tune E, Hagmann R, et al. CPI2: CPU performance isolation for shared compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems, 2013. 379–391
22 Lo D, Cheng L, Govindaraju R, et al. Heracles: improving resource efficiency at scale. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA), 2015. 450–462
23 Guo J, Hu C, Bao Y. Survey on guaranteeing the performance of co-located applications. J Comput Res Dev, 2024, 61: 43–65
24 Wang X, Li C, Zhuansun Y, et al. AUM: unleashing the efficiency potential of shared processors with accelerator units for LLM serving. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA), 2026
25 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: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems, 2019. 3–18
26 eBPF. What is eBPF? 2025. https://ebpf.io/what-is-ebpf/
27 Kanev S, Darago J P, Hazelwood K, et al. Profiling a warehouse-scale computer. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015. 158–169
28 Sriraman A, Dhanotia A. Accelerometer: understanding acceleration opportunities for data center overheads at hyperscale. In: Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, 2020. 733–750
29 Wang X, He H, Li Y, et al. Not all resources are visible: exploiting fragmented shadow resources in shared-state scheduler architecture. In: Proceedings of the ACM Symposium on Cloud Computing, 2023. 109–124
30 Wang J, Zhang L, Wang P Y, et al. Memory system optimization for graph processing: a survey. Sci Sin Inform, 2019, 49: 295–313 [王靖, 张路, 王鹏宇, 等. 面向图计算的内存系统优化技术综述. 中国科学: 信息科学, 2019, 49: 55–73]
31 Shu J W, Chen Y M, Wang Q, et al. Progress on storage systems for disaggregated data centers. Sci Sin Inform, 2023, 53: 1503–1528 [舒继武, 陈游旻, 汪庆, 等. 分离式数据中心的存储系统研究进展. 中国科学: 信息科学, 2023, 53: 1503–1528]
32 Zhang B, Zheng S, Qi Z, et al. NBTree: a lock-free PM-friendly persistent B+-tree for eADR-enabled PM systems. Proc VLDB Endow, 2022, 15: 1187–1200
33 Jing N, Jiang L, Zhang T, et al. Energy-efficient eDRAM-based on-chip storage architecture for GPGPUs. IEEE Trans Comput, 2016, 65: 122–135
34 Jing N, Wang J, Fan F, et al. Cache-emulated register file: an integrated on-chip memory architecture for high performance GPGPUs. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016. 1–12

https://www.sciengine.com/doi/10.1360/SSI-2025-0384

Intra-service tracing for shared and stressed datacenters: efficient management and intelligent analysis

Xinkai WANG[1], Chao LI[1*], Tianhao HUANG[1], Jinyang GUO[1], Xiaofeng HOU[1], Guoyao XU[2], Yuemin WU[3] & Minyi GUO[1]

1. *School of Computer Science, Shanghai Jiao Tong University, Shanghai 200240, China*
2. *Alibaba Group, Hangzhou 310052, China*
3. *Alibaba Cloud, Hangzhou 310030, China*
* Corresponding author. E-mail: lichao@cs.sjtu.edu.cn

Abstract With the rapid expansion of data centers, the escalating complexity of software and cluster scales demands more efficient and precise application observability. To identify performance bottlenecks and diagnose anomalies, it is essential to leverage execution tracing to accurately determine the causal relationships within application behaviors. However, we observe that in complex cloud data center environments-characterized by workload co-location and high hardware pressure, existing execution tracing tools typically incur a 5%∼10% performance overhead, hindering their seamless deployment across large-scale clusters. Although hardware tracing capabilities in modern processors are designed to reduce overhead, the four-dimensional design trade-offs among time efficiency, space overhead, data coverage, and management complexity remain insufficiently explored. Consequently, current solutions fail to meet the stringent overhead requirements of complex data centers. To address these challenges, this paper proposes ETrace, which optimizes time efficiency at both the node and cluster levels. At the node level, we leverage the four-dimensional design trade-offs of hardware tracing to minimize unnecessary control operations, thereby ensuring application performance, and dynamically allocate limited trace storage based on the application's real-time state. At the cluster level, we implement flexible and efficient multi-node trace management through a versatile sampling strategy and a cloud-native triggering framework. Evaluations across various benchmarks demonstrate that ETrace incurs an average performance degradation of only 0.97%, representing up to 85% overhead reduction compared to state-of-the-art solutions. Furthermore, using typical intelligent applications in Alibaba data centers as case studies, Also, ETrace's instruction-level tracing capability greatly enhances datacenter manageability with newer insights on anomaly diagnosis and bottleneck localization.

Keywords tracing observability, performance analysis, processor tracing, datacenter, cloud computing

https://www.sciengine.com/doi/10.1360/SSI-2025-0384