# MM-CLOCK：面向 NVMe SSDs 的 CLOCK 算法优化

刘艳雪[1)] 徐军[3)] 葛颂阳[3)] 季忠铭[3)] 孟令坤[3)] 李鑫[1)] 程冠杰[1)] 袁巩生[1),2)] 陈刚[1),2)]

[1)](浙江大学软件学院 浙江 宁波 315104)

[2)] (杭州高新区（滨江）区块链与数据安全研究院 杭州 310000)

[3)](中移（苏州）软件技术有限公司 江苏 苏州 215000)

**摘 要** 近年来，存储硬件的快速发展显著提升了存储容量和数据访问速度，这些发展为页面替换算法带来了新的挑战与机遇。为应对这些变化，本文提出了一种新型 CLOCK 页面替换算法 —— MM-CLOCK，旨在利用现代硬件特性提升系统整体性能。MM-CLOCK 算法主要包括两个关键机制：页面预选机制和自适应写感知策略。其中，页面预选机制利用命中率指标提前识别出一组在后续页面替换过程中将被淘汰的候选页面，从而在替换决策阶段降低计算开销，提高替换效率。而自适应写感知策略则基于访问命中模式动态调整脏页的写回时机，实现对写负载的灵活调度，通过适当地延迟写回操作，进而减少不必要的 I/O 操作，缓解频繁写入带来的存储设备损耗问题。在这两个机制的协同作用下，MM-CLOCK 不仅加速了页面替换过程，还引入了写感知，在提高数据库整体查询性能的同时，增强了系统对存储设备的友好性。此外，我们重新设计了缓冲池架构，引入了一种阈值驱动的主动页面写回机制，它可以在页面替换阶段开始执行前主动将缓冲区中的部分脏页面刷新到存储设备，充分利用了新型硬件的并发特性，减少页面替换过程中的写操作，从而降低写回延迟，提升系统的整体吞吐量。最后，为验证 MM-CLOCK 算法的有效性，我们在 WATT 仿真框架和配备了 NVMe SSD 的 PostgreSQL 中进行了大量实验评估。实验结果表明，在 WATT 仿真框架中，相较于当前最先进的页面替换算法（如 S3-FIFO），MM-CLOCK 在执行速度上实现了高达 5.17 倍的性能提升；在实际的 PostgreSQL 数据库中，相较于广泛应用的 Clock Sweep 算法，其事务吞吐能力提高了 8%，展现了优秀的实用性。

关键词 数据库，页面替换算法，页面预选机制，自适应写感知策略，阈值驱动的主动页面写回机制

**中图法分类号** TP311 DOI 号 *投稿时不提供 DOI 号*

## MM-CLOCK: CLOCK Algorithm Optimization for NVMe SSDs

LIU Yan-Xue[1)] XU Jun[3)] GE Song-Yang[3)] JI Zhong-Ming[3)] MENG Ling-Kun[3)] LI Xin[1)]

CHENG Guan-Jie[1)] YUAN Gong-Sheng[1), 2)] CHEN Gang[1), 2)]

[1)](*School of Software Technology, Zhejiang University, Ningbo, Zhejiang* 315104)

[2)] (*Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Hangzhou* 310000)

[3)](*China Mobile (Suzhou) Software Technology Co., Ltd, Suzhou, Jiangsu* 215000)

**Abstract** In recent years, the rapid development of storage hardware has significantly improved both storage capacity and data access speed. These developments introduce new challenges and optimization opportunities for page replacement algorithms. Traditional page replacement methods such as Least Recently Used (LRU) and CLOCK, originally designed for limited-memory environments, struggle to fully utilize the performance potential of modern hardware like NVMe SSDs, which feature high I/O parallelism and asymmetric read and

本课题得到中国移动、国家重点研发计划(2023YFC3603103)、浙江省“尖兵领雁 + X”研发攻关计划(2024C01019)项目资助. **刘艳雪**，硕士，CCF 学生会员(Z3722G)，主要研究领域为数据库内存管理. **徐军(共同一作)**，博士，主要研究领域为数据中心网络. **葛颂阳**，博士，主要研究领域为分布式学习. **季忠铭**，博士，主要研究领域为数据中心网络、云计算. **孟令坤**，硕士，主要研究领域为云计算、边缘计算.**李鑫**，硕士，主要研究领域为数据库内存管理. **程冠杰**，博士，主要研究领域为数据智能服务与系统. **袁巩生(通信作者)**，博士，主要研究领域为数据库系统. **陈刚**，博士，主要研究领域为数据库、大数据管理系统和大数据智能计算.

write characteristics. To address these changes and adapt to the evolution of modern hardware, this paper proposes MM-CLOCK, a novel variant of the CLOCK page replacement algorithm, specifically designed to leverage the capabilities of modern hardware to improve overall system performance. The MM-CLOCK algorithm mainly comprises two essential mechanisms: a page pre-selection mechanism and an adaptive write-awareness strategy, which operate collaboratively to achieve high throughput and reduced write overhead. The page pre-selection mechanism leverages the buffer pool hit rate to identify a set of candidate pages that are most likely to be evicted in the subsequent page replacement process. By selecting these pages before the actual eviction phase, the algorithm reduces the computational overhead during the replacement decisions, thereby improving efficiency and scalability. The adaptive write-awareness strategy takes into account the performance degradation and shortened lifespan of storage media caused by frequent writes in scenarios of high concurrency and large-scale data updates. It dynamically adjusts the write-back timing of dirty pages based on the access hit pattern to achieve flexible scheduling of write loads. By appropriately delaying write-back operations, it reduces unnecessary I/O operations and alleviates the storage device wear problem caused by frequent writes. With the synergistic effect of these two mechanisms, MM-CLOCK not only accelerates the page replacement process but also introduces write awareness into buffer management, improving the overall query performance of the database while enhancing the system's friendliness to storage devices. Furthermore, to alleviate I/O serialization bottlenecks and adapt to modern hardware features, we redesigned the buffer pool architecture and introduced a threshold-driven proactive page writeback mechanism. This mechanism proactively flushes some dirty pages in the buffer to the storage device before the page replacement phase begins. Upon completion, these pages are marked as clean to facilitate subsequent read and replacement operations. This process is executed in parallel with the read operation thread to fully utilize the access concurrency of modern hardware devices. This reduces write operations during page replacement, thereby reducing writeback latency and improving overall system throughput. Finally, to validate the effectiveness of the MM-CLOCK algorithm, we conducted extensive experimental evaluations using the WATT simulation framework and a PostgreSQL with an NVMe SSD. These experiments covered TPC-C, TPC-E, Linkbench, and three Zipfian-distributed workloads. The experimental results demonstrated that, in the WATT simulation framework, MM-CLOCK achieved up to 5.17x faster execution compared to state-of-the-art page replacement algorithms (such as S3-FIFO), demonstrating significant performance advantages across a variety of workloads. In the real-world PostgreSQL database with NVMe SSD, MM-CLOCK also shows strong performance, achieving an 8% improvement in transaction throughput compared to the widely used Clock Sweep algorithm, demonstrating its excellent practicality.

**Key words** database; page replacement algorithm; page pre-selection mechanism; adaptive write-awareness strategy; threshold-driven proactive page writeback mechanism

## 1 引言

在数据中心网络中，服务器和计算节点往往需要运行分布式计算框架（如 Spark、Flink 等）、执行大规模 AI 训练任务，或者支持数据库系统的高效运行。这些复杂的计算任务和大规模的数据处理对页面替换算法提出了更高的要求，以确保系统对内存资源的有效利用，从而提升整体性能。此外，在远程直接内存访问（Remote Direct Memory Access，RDMA）或跨域资源共享等场景中，远程数据访问成本明显高于本地内存访问。因此，高效的页面替换算法对于减少远程数据访问和提升系统性能至关重要。

传统的页面替换算法，例如经典的 CLOCK 和最近最少使用（Least Recently Used，LRU）算法，最初是为内存容量有限、处理速度较低的旧硬件架构设计的。这些算法虽然在早期的计算环境中发挥了重要作用，但未能充分利用现代硬件的潜力。最近，数据库领域提出了几种新的页面替换算法，例如 ARC[1] 和 LeCaR[2] 等。尽管这些算法在理论上提供了更优的页面替换策略，但它们通常依赖于复杂的机制，如强化学习[3] 或自适应模型，这不可避免地增加了计算开销。此外，虽然这些算法在非计算密集型场景中可能会带来一定的性能提升，但它们的资源需求使其在高性能网络或高吞吐量、高

并发的数据库系统中的应用变得不太实际，尤其是对延迟和吞吐量有严格要求的系统。因此，如何在保证系统高性能的同时降低开销，成为了新一代页面替换算法设计中的关键挑战。

随着硬件的快速发展，由于新型存储设备（例如，NVMe SSD）能够为系统提供更快的数据访问速度和更高的性能，正逐步取代传统机械硬盘（HDD）在高性能计算和数据存储方面的应用。但是，NVMe SSD 具备两个典型特性：（1）读写不对称，写入速度明显低于读取，（2）高 I/O 并行性，支持多个 I/O 请求同时执行。而大多数现有系统并未充分考虑这些特性。例如，现有页面替换策略普遍平等地对待读写请求，忽略了写操作的高成本。尤其是在写密集型负载下，这种忽视会加剧 SSD 的写放大问题[4]，即实际写入的数据量超过主机请求的数据量，加速设备磨损。此外，主流缓冲池管理器仍然假设底层设备并不具备并发性，通常以串行方式写回脏页面。虽然这一方式能够实现最小化磁盘争用，但未能有效利用新硬件带宽，导致 CPU 和存储资源的利用率不足，系统延迟增加。

为应对上述挑战，新的页面替换算法不仅需要关注计算复杂度，还应充分考虑硬件特性，以确保在现代计算环境中实现最佳性能。在本文中，我们提出了一种新的页面置换算法 MM-CLOCK（Throughput **M**aximization and Writeback Overhead **M**inimization CLOCK-based Algorithm），作为经典 CLOCK 算法的优化变体，旨在提高数据库系统吞吐量的同时显著降低写回开销。此外，我们还重新设计了 I/O 架构，使系统能够在适当时机主动刷新脏页（即已被修改但尚未写回磁盘的页面），从而在页面置换过程中有效降低写入延迟，提升系统整体吞吐量。最后，本文在 WATT 模拟框架[5]和 PostgreSQL 数据库中实现了 MM-CLOCK，并在多类典型工作负载下进行了充分的实验，以验证算法在提升系统吞吐性能与写回开销方面的显著优势。本文的主要贡献如下：

（1）本文提出了一种新型页面替换算法 MM-CLOCK，该算法通过引入页面预选机制和自适应写感知策略，能够在不同工作负载下有效提升数据库查询的吞吐量，减少写回开销。与此同时，在一定程度上可以提高数据库的命中率。

（2）本文设计了一种阈值驱动的主动脏页写回 I/O 架构，旨在有效缓解传统缓冲池中存在的串行化问题。该架构通过高效的 I/O 调度机制，充分利用新型存储设备 NVMe SSD 的并行性，减少写回延迟，显著提升了数据库系统的整体性能。

（3）本文在 WATT 模拟框架和 PostgreSQL 中实现了 MM-CLOCK，在 TPC-C、ZipfRO 等多个工作负载中进行了广泛的实验。结果表明，MM-CLOCK 在提高系统吞吐量和降低页面写回开销等方面表现优异。

## 2 相关工作

### 2.1 LRU 及其变体

LRU 页面替换算法是数据库系统内存管理中广泛应用的一种策略。如图 1 所示，LRU 通常使用一个双链表跟踪页面访问模式。每次访问一个页面时，它会被移动到双链表的前端，而尾部的页面则成为被驱逐的候选对象。通过这种方式，LRU 能够有效地确保在页面替换时移除最久未被访问的页面。

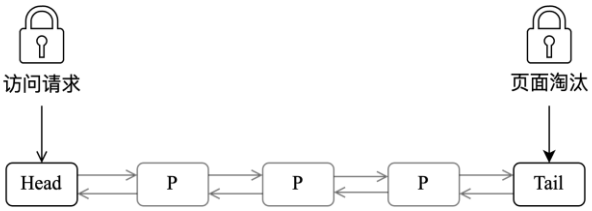



图 1 LRU 双链表示意图

在很多数据库系统中，LRU 及其变体仍然是主流的缓冲池管理策略。例如，MySQL 数据库[6]采用 LRU 来管理缓存列表，并将其列表划分为“旧子列表”和“新子列表”。页面首先被加载到旧子列表，只有在满足特定条件时，才会被转移到新子列表。该机制有效地平衡了页面访问的最近性（Recency）与访问频率（Frequency），从而优化了缓存管理。Oracle[7] 则在标准 LRU 的基础上，为了更好地支持高并发访问，采用了多个 LRU 列表以避免竞争。DB2[8] 通常依赖于标准 LRU 进行缓冲池管理，但在某些特定场景下，它也会根据当前工作负载的特点采用改进的 LRU 策略。

此外，还有一些 LRU 变体，如 2Q[9] 算法，通过引入 FIFO 列表和 LRU 列表来增强页面管理的灵活性。页面最初放置在 FIFO 列表中，再次访问时会被移动到 LRU 列表，有效地结合了 FIFO 的先进性和 LRU 的效益。自适应替换缓存（Adaptive Replacement Cache，ARC）算法则更加灵活，通过维护 LRU 和最不常用（Least Frequently Used，LFU）列表，根据实际工作负载动态调整二

者的大小。这一机制使得 ARC 能够灵活应对不同的工作负载，进一步提升系统整体性能。同样，学习型缓存替换（Learning Cache Replacement，LeCaR）和 CACHEUS[10] 等人工智能驱动的方法也引入了 LRU 列表，以实现更高效的页面管理。CLOCK 算法作为 LRU 的另一种高效近似实现，因其独特结构与丰富变体，本文将在第 2.2 小节中单独进行讨论。

尽管 LRU 在许多传统应用场景下表现良好，但其局限性在现代硬件和复杂应用环境中逐渐显现。LRU 算法假设页面访问模式遵循时间顺序，即最近访问过的页面很可能再次被访问。然而，这一假设在面对现代数据库系统的复杂访问模式和高并发环境时并不总是成立，尤其是在分布式计算和多核 CPU 环境中。由于多个处理器共享 LRU 列表头部的锁，这可能引发严重的锁争用，进而限制 LRU 算法的可伸缩性。此外，LRU 算法缺乏针对写操作的优化策略，无法有效减轻磁盘写回带来的负担，导致不必要的写回操作频繁发生，进而影响系统的整体性能。

### 2.2 CLOCK 及其变体

CLOCK 算法及其变体在许多数据库系统和操作系统中同样得到广泛应用。CLOCK 算法使用一个循环列表管理缓冲池中的页面，并通过一个受害者指针（Victim Pointer）标识进行淘汰的页面，如图 2 所示。由于 CLOCK 算法设计简单且高效，它通常应用于实际系统中。PostgreSQL[11] 数据库采用一种基于 CLOCK 算法的变体——Clock Sweep，利用访问计数器记录页面的访问频率，并根据访问计数器的数值调整页面的替换顺序。类似地，GCLOCK[12] 算法为每个页面分配一个计数器，并根据数据类型的不同赋予初始权重。当页面被访问时，计数器根据访问状态进行相应的权重更新。

此外，学术界还提出了多种基于 CLOCK 算法的改进方案，以提高自适应性和性能。例如，Dueling CLOCK[13] 根据工作负载特征在标准 CLOCK 和抗扫描变体（Scan-resistant CLOCK）之间进行动态选择。CLOCK-pro[14] 进一步扩展了 CLOCK 算法，通过结合访问频率和顺序来优化页面替换，借鉴了 LIRS（Low Inter-reference Recency Set）算法的思想。CAR（CLOCK with Adaptive Replacement）[15] 结合了 CLOCK 和 ARC 的优点，既保留了 CLOCK 的高效性，又引入了 ARC 的自适应替换策略，从而更好地适应不同的数据库工作负载。

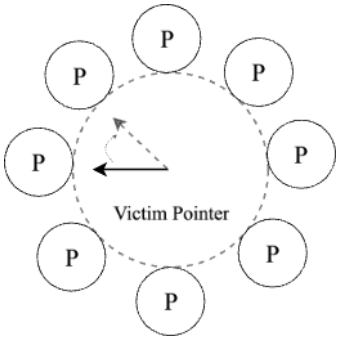



图 2 CLOCK 算法的基本结构

CLOCK 算法作为 LRU 算法的一种高效近似实现，适用于 LRU 友好型工作负载。然而，它也存在一些明显的不足，尤其是在处理大量页面时，需遍历循环列表来识别合适的页面进行替换，这使得替换选择效率较低。与 LRU 算法类似，CLOCK 算法同样缺乏写感知策略，在写密集型工作负载下会导致磁盘写入操作的增加，从而降低系统性能。

### 2.3 其它页面替换算法

除了 LRU 和 CLOCK 算法以外，新的页面替换算法应运而生，旨在解决传统算法的不足。例如，CFLRU（Clean-first LRU）[16] 改进了传统 LRU 算法，通过将 LRU 列表划分为工作区和干净优先区，优先选择干净优先区中的干净页面（即未被修改、无需写回磁盘的页面）进行驱逐，以减少不必要的写操作。LRU-WSR[17] 引入了写顺序重排序（Write Sequence Reordering，WSR）机制，优化了页面的驱逐顺序，进一步减少写操作。WATT 算法则通过调整权重控制页面值，以平衡读写负载。

在现代 Web 缓存架构中，Hyperbolic Caching[18] 利用双曲空间特性进行数据分配和访问，提升了大规模缓存系统的效率，而 SIEVE[19] 算法则采用单个 FIFO 队列和指针机制，适用于分布式缓存系统。尽管这两种算法在顺序扫描型工作负载下表现欠佳，但其局限性说明在复杂的缓存场景中，单一替换策略通常难以满足多样化性能需求。这一现象表明现代缓存系统需要多策略融合以应对复杂的工作负载，从而实现性能与资源利用率的动态平衡。

### 2.4 主动脏页刷新策略

在数据库系统中，为降低查询期间因脏页写回带来的阻塞风险，许多系统引入了后台主动脏页刷新机制。例如，PostgreSQL 通过 BgWriter 模块在后台周期性地将部分脏页写回磁盘，降低查询期间的写回延迟，从而提升系统的响应速度和并发处理

能力。openGauss 在此基础上进一步引入 PageWriter 线程组，以完成增量检查点的脏页刷写任务，并结合双写机制缓解页面断裂所带来的数据损坏风险，从而增强系统在崩溃恢复和数据页一致性保障方面的能力。

LRU-C[20] 算法则采用一种动态批量写回（Dynamic Batch Write）机制，通过维护一个指向 LRU 链表尾部首个干净页的 LRU-C 指针，动态调整每轮刷写的脏页数量。该机制不仅能够兼顾页面替换效率，还能提升写入带宽和 SSD 资源利用率，使 LRU 尾部的冷页和脏页尽早释放为可用缓存，从而为前台请求预留更多空间，增强系统在高负载下的响应能力。

尽管上述机制在一定程度上能够分散写入压力、缓解线程阻塞问题，但其本质上仍属于固定周期驱动的刷新策略，存在对工作负载变化响应迟缓的问题。若参数设置不当（如刷新周期过短或单次刷写量过大），容易导致高频更新的脏页被反复写回磁盘，引发 I/O 写放大问题，从而降低存储设备的使用效率，增加系统负担。此外，批量写回可能误将即将被访问的页面提前淘汰，造成缓存命中率下降，影响系统整体性能。

# 3 MM-CLOCK 的设计原则

## 3.1 页面预选机制

为了实现系统的最佳性能，Belady 的 MIN 算法建议从内存中淘汰最长时间未被访问的页面。从那时起，研究者提出了各种算法来对页面进行排序，以找到那些在尽可能长时间内不会被再次访问的页面，从而确保替换操作的最优性。对于单个页面的替换需求，这种方法是最优的，因为它可以保证一对一的最优解决方案。然而，在实际应用中，由于其计算复杂度较高，这类方法在大规模的页面替换请求中难以高效执行，尤其是随着计算机硬件的快速发展，特别是现代 CPU 多核架构和 SSD 技术的出现，数据库在高负载场景下可能需要每秒替换数百万个页面。因此，仅关注单个最冷页面的传统页面替换策略已难以满足现代数据库系统复杂多变的实际需求。

为了弥补这一不足，我们提出了一种**页面预选机制**。与传统的替换算法不同，这种方法不需要对所有页面根据冷热程度进行严格排序。相反，它通过快速识别那些在未来一段时间内不会被频繁访问的页面，将其标记为后续批量替换的候选页面，并统一加入空闲列表中。此机制避免了传统算法中对页面访问顺序的过度依赖，更加高效地识别出不再活跃的冷页面。同时，通过预选机制避免了在实际替换时进行复杂的页面选择，从而减少了替换决策过程中的计算开销，能够支持批量替换并有效提升数据库系统查询的运行速度。

然而，这一机制面临的主要挑战之一是如何确定预选页面的数量。如果预选的页面数量过多，尽管这些页面尚未立即驱逐，但仍需维护在空闲列表中，会增加管理开销，并可能误导后续替换阶段的调度决策；而如果预选的页面数量过少，则可能无法满足系统在高负载下对快速批量替换的需求。因此，如何动态调整每次预选页面的数量，既确保页面替换过程的高效性，又不影响系统的准确性，成为了该机制的关键问题。

为了解决这一问题，我们设计了一个基于命中率的**页面驱逐函数**，该函数通过周期性地监控缓冲池的命中率来计算驱逐率（驱逐页面占总页面的比例），从而动态调整每次预选页面的数量。为了更有效地追踪页面的访问频率，我们引入了页面值（Page Value）的概念，类似于前文提到的访问计数器。每当页面被访问时，其对应的页面值便会增加，用以反映该页面的访问活跃度。页面值较低通常意味着该页面长时间未被访问，因此被认为是“冷”页面。为了有效提升内存资源的利用率，MM-CLOCK 算法倾向于优先驱逐这些冷页面。此外，MM-CLOCK 算法还引入了引脚位 R，用于标识页面当前是否正被使用。当某一页面的引脚位 R = 1 时，表示该页面处于使用状态，不允许被替换，需保留在缓冲池中。

根据驱逐率和实时页面值的分布情况，我们将预选出一组相对较冷的页面，并将其加入“free-list”空闲列表中。该列表专门用于存储待驱逐页面，确保在后续替换过程中能够迅速且准确地进行驱逐。例如，当页面驱逐函数计算出当前驱逐率为 40% 时，页面预选机制将据此筛选出一组候选驱逐页面。在该过程中，我们只需判断缓冲池中的每个页面是否属于所有页面中访问频率最低的 40%，以确定其是否应被纳入驱逐候选列表。关于这一机制的更多细节，将在第 4 节中进行详细说明。

需要注意的是，加入空闲列表中的候选页面并未实际被驱逐，而是处于一种待驱逐状态，可以理解为具备较高的驱逐优先级。这些页面的数据仍然

保留在缓冲池中，在真正被新的页面替代之前，依然可以被正常访问。因此，空闲列表中的页面在被替换之前，通常可能会经历多次复用。

### 3.2 自适应写感知策略

NVMe SSD 凭借其高 I/O 吞吐量、低功耗和低噪音等优势，已成为现代数据库系统的主流存储设备。与传统机械硬盘相比，NVMe SSD 在读写性能和响应速度方面表现尤为突出，广泛应用于高性能计算、大数据分析以及实时数据库系统中。

然而，NVMe SSD 所依赖的闪存介质存在一个关键限制：每个存储单元的擦除和写入次数是有限的，一旦超过其能承受的阈值，相关存储单元将变得不可用，进而影响整个设备的稳定性和使用寿命。这一问题在数据库系统中表现得尤为突出，因为数据库在运行过程中往往需要频繁执行写操作，尤其是在高并发和大规模数据更新的场景下，写入操作更是极为密集，这加剧了存储介质的磨损，进而导致设备性能退化与寿命缩短等问题日益严重。

因此，如何有效延长 NVMe SSD 的使用寿命，已成为数据库系统中存储架构设计时需要考虑的重要因素。一种相对直接的策略是尽可能长时间地将脏页面保留在缓冲池中，以减少频繁的写入操作，降低设备磨损。然而，若脏页面保留时间过长，可能会牺牲缓存命中率，影响系统性能。因此，在延长存储设备寿命与保障系统性能之间寻找合理的平衡，成为设计优化的关键所在。

为此，我们提出了一种**自适应写感知策略**，期望在保障缓冲池命中率的同时，有效控制写回开销。该策略通过周期性地监控缓冲池的命中率，动态调整**写感知权重**（表示允许跳过的脏页面数量），以实现对写回行为的灵活控制。具体而言，在页面预选机制生成候选页面列表的过程中，系统会优先选择缓冲池中的干净页面，以避免不必要的写操作。而对于脏页面，则根据当前的写感知权重跳过一定数量的脏页面，使其能够继续保留在缓冲池中，延后写回，从而减少对 NVMe SSD 的写入压力。实验结果表明，该策略在显著降低 NVMe SSD 写入压力的同时，依然能够维持较高的缓存命中率，兼顾了系统性能与存储介质寿命的双重需求。

### 3.3 阈值驱动的主动页面写回机制

随着硬件性能的提升，尤其是 NVMe SSD 等高性能存储设备的广泛应用，传统缓冲池架构中基于顺序访问的 I/O 模式已难以充分发挥底层硬件的并行能力，甚至可能会引发 I/O 串行化问题。在此类架构中，读写请求通常按照串行顺序排队执行，新到达的请求需等待前一操作完成后方可继续处理，导致 CPU 与存储之间存在较大的调度空隙，限制了系统的整体性能与资源利用率。这一问题在执行脏页面写回操作时尤为突出：传统策略通常在页面实际被替换出缓冲池时才触发同步写回，极易造成查询阻塞，增加事务响应时延，进而降低系统的吞吐能力。同时，在 NVMe SSD 这一类设备中存在显著的读写性能不对称性，写操作开销远高于读操作，频繁地触发同步写回还会加剧写放大问题，从而增加介质磨损，缩短设备寿命。

为缓解 I/O 串行瓶颈并适应现代硬件特性，本文提出一种**阈值驱动的主动页面写回机制**。其核心思想是在数据库查询请求执行之前，主动将部分脏页面提前写回存储设备，从而避免传统缓冲池架构中常见的 I/O 串行化问题，降低读写不对称性带来的负面影响。通过这一方法，我们能够有效减少 I/O 争用问题，提高系统的并行处理能力。具体而言，主动页面写回机制采用基于阈值的触发方式，当缓冲池空闲列表中的候选页面数量低于预设阈值时，系统将空闲列表中的脏页面主动写回，并在写回后将这些页面标记为干净页面，以便于后续的读取和替换操作，该过程与读操作线程并行执行，以充分利用新型硬件设备的高并行性。

与前文 2.4 节中所述的 BgWriter 和 LRU-C 的动态批量写回等主动脏页刷新机制相比，MM-CLOCK 的主动页面写回机制具有以下显著差异和创新优势：首先，传统方法大多基于固定周期触发脏页刷新，容易导致在脏页频繁更新期间产生重复写入，加剧写放大问题；其次，此类机制在写回后直接将页面驱逐出缓冲池，导致后续访问无法命中，造成命中率下降。相比之下，本文所采用的阈值驱动的主动页面写回机制不仅能根据运行状态动态响应缓存压力，充分利用新型存储设备的高并行性，缓解读写不对称带来的延迟问题，还允许将写回后的页面继续保留在缓冲池中，避免对命中率产生负面影响，在高负载、高并发环境下能够保持较高性能，实现性能与可靠性的双重优化。

## 4 MM-CLOCK 的实现细节

在本节中，为了清晰地阐述 MM-CLOCK 算法的工作原理，我们将深入分析其核心组成模块，

包括：（1）驱逐率计算器（Eviction Rate Calculator，ERC）；（2）写感知权重计算器（Write-awareness Weight Calculator，WWC）；（3）提前页面写回器（Early Page Writer，EPW）。如图 3 所示，这三个模块协同合作，共同保障了页面替换过程的高效性，并在系统运行中显著提升了系统的整体性能。

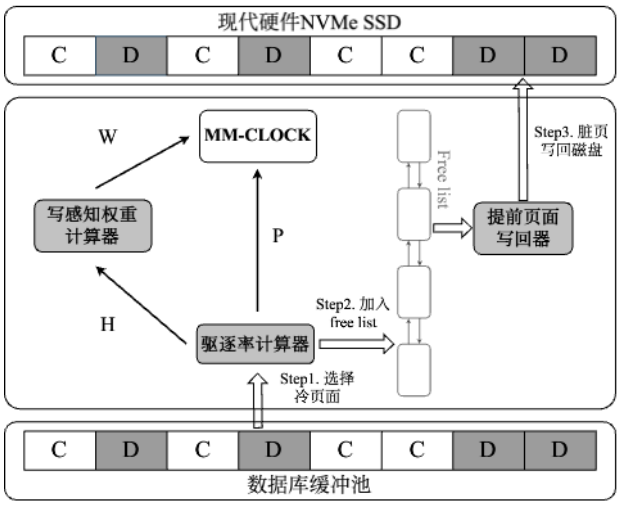



图 3 MM-CLOCK 算法框架图

### 4.1 MM-CLOCK 概述

作为一种面向现代硬件的新型页面替换策略，MM-CLOCK 算法能够根据系统运行的实时状态，灵活协调多种机制，从而在高负载、高并发的场景下发挥现代存储设备的性能优势。其核心思想是：通过命中率驱动的算法调整机制与主动写回机制相结合，在降低写入压力的同时保障缓存命中率，提高系统吞吐量。具体而言，MM-CLOCK 算法主要包含以下三个关键模块：

（1）**驱逐率计算器**：该模块用于周期性地监测缓冲池命中率 $H$，并依据页面驱逐函数计算得到对应的驱逐率 $E$。随后，根据这一驱逐率预选出一组候选页面，作为后续页面替换操作的驱逐对象。

（2）**写感知权重计算器**：该模块根据命中率 $H$ 获得相应的写感知权重 $W$，该权重用于决定在页面预选过程中应跳过多少脏页面，从而有效减少不必要的写回操作。

（3）**提前页面写回器**：该模块在数据库执行读写操作之前，主动将空闲列表中的脏页面写回存储设备。相比传统被动式写回方式，该机制能够减少写操作造成的 I/O 争用问题，有效发挥现代硬件的并行处理能力。

### 4.2 驱逐率计算器

为了实现对页面访问状态的自适应感知以及对页面驱逐行为的动态调控，MM-CLOCK 算法引入了一种**页面预选机制**。该机制以 epoch 为单位进行周期性更新，每轮 epoch 开始时，系统首先统计当前命中率 $H$，并通过驱逐函数动态计算出驱逐率，从而确定本轮页面预选的目标比例。随后，利用维护在内存中的页面值分布直方图，估算当前缓存中页面的访问热度分布，并据此推导出驱逐阈值 $E$，用于筛选出一批具有较高驱逐优先级的候选页面。最终，系统将满足阈值条件的页面作为待驱逐页面加入空闲列表，为后续替换操作提供快速可用的页框资源，在保证替换效率的同时有效控制系统开销与命中影响。

**页面驱逐函数。**在关于驱逐率的建模中，MM-CLOCK 算法假设驱逐率与命中率呈负相关关系：命中率越高，说明缓冲池能够很好地满足查询请求，页面替换必要性降低，因此应适当减小驱逐率；反之，当命中率下降时，缓存命中能力开始减弱，此时应相应地提高驱逐率，以提供更多的待驱逐页面供后续页面替换使用。为此，MM-CLOCK 算法设计了一个简单的线性函数，动态计算当前的驱逐率 $E$：

$$E = \alpha * (1 - H) + \beta * e \quad (1)$$

其中，$H$ 表示当前的命中率；$e$ 为系统预设的初始驱逐率，在缺乏历史信息时为算法提供基础的驱逐保障。参数 $\alpha$ 作为命中率响应因子，较高的取值可以提高系统对工作负载变化的响应速度，增强适应性；而 $\beta$ 则是稳定性调节因子，用于平衡当前驱逐率与初始值之间的关系，有助于抑制因缓存波动引起的剧烈调整，从而保持系统整体的稳定性。参数 $\alpha$ 和 $\beta$ 的协同调节，有助于系统在响应短期波动与保持长期稳定性之间取得良好平衡。该函数使得系统可以根据缓冲池命中状态实时更新驱逐策略，提升页面替换效率与自适应能力。

进一步地，MM-CLOCK 将驱逐率与页面预选行为关联起来：假设缓冲池页面总数为 $N$，则每轮预选的待驱逐页面数量 $K$ 可由以下公式估算：

$$K = E * N \quad (2)$$

通过上述建模，系统地建立起命中率 $H$ 到页面预选数量 $K$ 的动态控制关系，有效实现了页面预选数量的周期性自适应调节。

**基于 Epoch 的驱逐率更新。**为了降低驱逐率与命中率计算带来的实时性能开销，MM-CLOCK

采用基于 epoch 的周期性更新机制。系统以一个 epoch 为单位，周期性地重新计算命中率和驱逐率，避免频繁计算造成的资源消耗。考虑到静态 epoch 长度可能无法有效应对访问模式的动态变化，MM-CLOCK 提出一种自适应 epoch 策略：系统以累计页面驱逐数量作为 epoch 更新的触发条件。具体而言，假设当前命中率为 80%，则当累计被驱逐页面数量达到缓冲池总页数的 20%（即 $1-H$）时，判定缓存状态已发生显著变化，触发新一轮 epoch 的开始，重新统计命中率并更新驱逐率。

该策略不仅降低了更新频率过高带来的系统开销，也使得驱逐策略更能匹配访问行为的实际变化，增强了算法在多样化工作负载下的适应性。

**驱逐阈值的计算。**为了提升缓存管理的效率，MM-CLOCK 算法在内存中维护了一个页面值直方图，用于描述当前 epoch 内缓存页的活跃程度。直方图通过多线程随机抽样与原子操作实现，可以在高并发环境下准确反映全局页面活跃程度的统计特征。实验结果表明，直方图的维护开销非常低。这是因为页面值的最大值通常不超过 10，且随着缓存页面数量的增加，整体维护开销趋于平稳状态，几乎可忽略不计，从而确保了内存资源的高效利用。

在完成驱逐率 $E$ 的计算后，系统基于页面值直方图估算驱逐阈值 $P$，即选取一个页面值边界数值，使得页面值小于等于 $P$ 的页面数量约占缓冲池总页数的比例为 $E$，以用于后续的空闲列表构建过程。

### 4.3 写感知权重计算器

在 MM-CLOCK 算法的设计中，系统首先计算出当前 epoch 的命中率，用于反映缓冲池中页面请求的命中情况。在此基础之上，MM-CLOCK 算法引入了一个写感知权重 $W$，用于决定在页面预选过程中跳过的脏页面数量，并根据命中率 $H$ 的变化进行动态调整。当命中率较高时，系统倾向于延迟对脏页面的驱逐，以减少不必要的写回操作；而在命中率较低的情况下，系统则适当降低写感知权重 $W$ 的取值，以提升页面替换的准确性。写感知权重计算器本质上建立了命中率 $H$ 与写感知权重 $W$ 之间的动态映射关系，具体的映射逻辑将在第 5.2 节中详细介绍。

驱逐率计算器和写感知权重计算器是组成 MM-CLOCK 空闲列表构建逻辑的核心，其具体实现方式如算法 1 所示：

**算法 1.** MM-CLOCK 空闲列表的构建.

```
// H：命中率，N：缓冲区页面数量，e：初始驱逐率
// α、β：驱逐率参数，Buffer：缓冲区页面集合
// P_v、Histogram：页面值及其直方图，D_c：脏页面计数器
输入：H，N，e，α，β，P_v、D_c，Histogram，Buffer
输出：FreeList
1.  // 计算驱逐率、驱逐阈值
2.  E = α * ( 1 - H ) + β * e
3.  TargetCount = E * N
4.  P = Threshold ( Histogram, TargetCount )
5.  // 计算写感知权重
6.  W = GetWeight ( H )
7.  // 构建 FreeList
8.  初始化 D_c = 0
9.  FOR 每个页面 page ∈ Buffer DO
10.     IF page 为干净页且 P_v( page ) ≤ P THEN
11.         AddToFreeList ( page )
12.     ELSE        // page 为脏页
13.         IF D_c ≥ W 且 P_v( page ) ≤ 0 THEN
14.             AddToFreeList ( page )
15.         ELSE
16.             D_c = D_c + 1
17. RETURN
```

在进入一个新的 epoch 时，系统首先根据上一轮 epoch 的页面访问情况统计当前的命中率 $H$，并结合页面值直方图动态计算驱逐阈值 $P$ 和写感知权重 $W$。随后，系统遍历缓冲区中所有页面：对于标记为干净的页面，若其页面值小于等于 $P$，则直接将其加入空闲列表，作为待驱逐候选；对于标记为脏的页面，系统引入了一个计数器 $D_c$，用于跟踪已跳过的脏页数量，每当系统跳过一个脏页，计数器的值加一，当 $D_c$ 达到设定的写感知权重 $W$ 后，系统在本轮 epoch 内将不再无条件跳过脏页，而是判断其页面值是否小于等于零，若成立则表明其此时已经“足够冷”，将其加入至空闲列表中。

需要指出的是，驱逐阈值 $P$ 是在假设干净页与脏页可以平等驱逐的前提下计算的，但由于脏页额外受到写感知策略的双重约束（跳过控制与页面值判断），最终进入空闲列表的页面数量通常会略小于预选目标值。这一偏差反映了算法在追求替换效率的同时，对写放大成本的显式控制，从而在性能与写放大之间实现平衡，有助于提升系统的写入性能与资源使用效率。如图 4 所示，根据算法 1 的描述，系统会将页面 p4 加入空闲列表中。

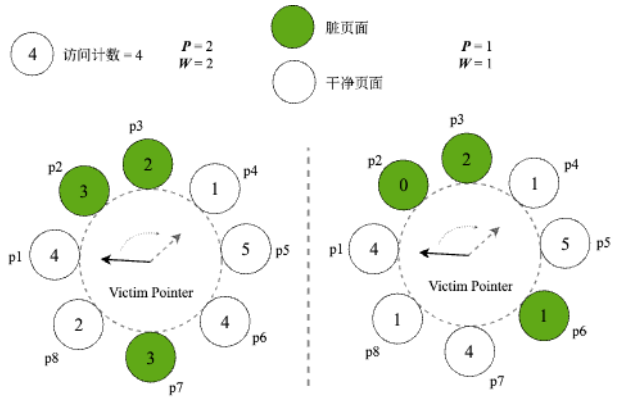



图 4 MM-CLOCK 将会驱逐 p4

算法 1 在每轮 epoch 中需遍历一次缓冲池页面集合，主循环时间复杂度为 O(*N*)，其中 *N* 为缓冲池的页面总数，尽管为线性复杂度，但该过程本质上是一种预选操作，系统可以一次性筛选出具有高驱逐优先级的候选页面，存入空闲列表，从而在后续替换阶段避免再次遍历全局页面，显著降低替换路径上的延迟与开销。空间复杂度主要由候选列表与页面值直方图构成，整体上仍为 O(*N*)。

### 4.4 提前页面写回器

当缓冲池空闲列表中的候选页面数量低于预设阈值（默认设定为缓存页总数的 10%）时，提前页面写回器将被触发。该阈值是基于大量实验结果确定的，能够在性能和资源利用率之间取得平衡。阈值驱动的主动页面写回机制的主要目标是预处理空闲列表中的脏页面：在页面被实际替换之前，主动将其从缓冲池中写回存储设备，避免在真正发生页面替换时由于脏页写回而引发延迟。这一机制能够有效利用现代硬件 NVMe SSD 的并发读写能力，提升 I/O 吞吐量，进而优化系统整体性能。

## 5 MM-CLOCK 的参数调整

尽管第 4 节中提出了驱逐率的计算方法，并定义了基于命中率动态调整的写感知权重 *W*，但相关方法的有效性尚未经过验证，参数取值的设置方式也不明确。为此，本节将通过设计多组实验，系统分析参数对性能的影响并确定合理配置。

**实验方法。**本实验使用表 1 中的三种典型工作负载类型 —— 随机分布、指数分布和高斯分布进行性能测试。在 PostgreSQL 16.1 环境下实现了 MM-CLOCK，并使用 pgbench[21] 工具生成可控命中率和读写比（R:W）的负载，以模拟不同场景下的访问模式。考虑到驱逐率和写感知权重均受到命中率的显著影响，实验首先需要确定命中率的变化区间。参考 Tsuei 等人[22]关于缓冲区大小与 TPC-C 性能关系的研究，我们将缓冲池容量设置为数据总量的 10% - 15%，以确保系统在默认配置下能够实现 80% 以上的命中率。基于上述设定，后续实验将通过调整工作负载特征及缓冲区占比，控制不同的命中率水平。具体实验涵盖以下五个命中率区间：50% - 60%、60% - 70%、70% - 80%、80% - 90%、90% - 100%。需要注意的是，在相同的条件下，缓冲池容量越大，命中率越高。因此，尽管结果图中的 x 轴表示命中率区间，实际上它反映的是是缓冲池大小。

表 1 工作负载的倾斜度比较

| 工作负载 | 倾斜度 |
|---|---|
| 随机分布 | 无 |
| 高斯分布 | 轻微 |
| 指数分布 | 中等 |
| Zipfian 分布 | 严重 |
| TPC-C | 高 |

### 5.1 页面驱逐函数参数配置与敏感性分析

在页面驱逐函数（公式 1）中，驱逐率 *E* 依赖三个关键参数：命中率响应因子 $\alpha$ 、稳定性调节因子 $\beta$ 以及初始驱逐率 *e*。尽管通过穷举所有可能的参数组合来获得最优配置是一种直观的方法，但其实验成本较高且实施困难。为了解决这一问题，本文提出了一种基于抽样策略的参数设置方法。具体而言，我们将命中率设置为 50% - 100% 范围内的 5 个具体值，驱逐率则在 20% - 80% 的范围内设置为 7 个具体值（步长为 10%），如图 5 和图 6 所示。通过在三类典型负载（随机分布、指数分布、高斯分布）下评估系统的运行速度与命中率变化，并与未引入驱逐率的情况进行对比，反向推导出 $\alpha$、$\beta$ 与 *e* 的合理取值，以确保其在不同负载下系统表现良好。

如图 5 和图 6 所示，x 轴和 y 轴分别表示选取的驱逐率和命中率配置，热图值则表示在不同配置下引入驱逐率所带来的系统运行速度和命中率的提升情况。例如，在图 5 中，当命中率在 50% - 60% 区间内，驱逐率设置为 50% 时，算法的运行速度显著提升；而在命中率达到 90% - 100% 的高命中率区间时，仅需设置 20% 的驱逐率即可获得良好效果。图 6 则展示了驱逐率对命中率的影响，以指数分布负载为例，当驱逐率为 50% 时，

系统在 50% - 60% 的低命中率区间内，命中率实现提升；而在命中率达到 90% - 100% 时，驱逐率设置为 20% 同样能够带来有效的性能提升。

根据上述实验结果，结合页面驱逐函数 $E = \alpha * (1 - H) + \beta * e$（可以视为“命中率反馈调节模型”），我们可知：

（1）**命中率响应因子 $\alpha$** 用于调节负载动态变化，避免驱逐率过高（造成部分尚未“冷却”的热点页面被过早淘汰）或过低（导致对负载变化的响应不及时）。理论上，$\alpha$ 可设计为随命中率动态变化的参数，但考虑到算法复杂性以及实验观测结果，当 $\alpha$ 设置为 0.9 时，驱逐率在多种负载下能够表现出良好的性能。因此，我们将 $\alpha$ 的默认值设置为 0.9。

（2）**稳定性因子 $\beta$** 用于调节驱逐率对初始驱逐率 $e$ 的依赖程度。考虑到 $\beta$ 的值越大，系统的稳定性和抗扰动能力越强。因此，我们将 $\beta$ 设置为 1，以确保系统在不同命中率波动区间内能维持较强的鲁棒性。

（3）**初始驱逐率 $e$** 主要影响系统冷启动阶段的驱逐率，过高的取值可能导致系统在早期阶段发生误驱逐。因此我们将 $e$ 设置为 0.1，以避免系统过早地进行误驱逐。

后续的页面预选机制实验进一步验证了这一参数组合在多种典型负载下的稳定表现，体现出较强的负载适应能力与参数配置的泛化性。

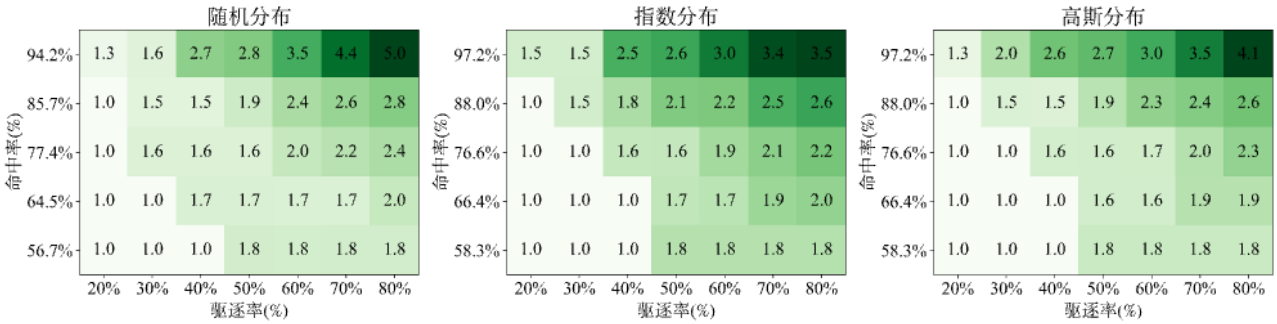



图 5 页面预选机制对 MM-CLOCK 运行速度的影响

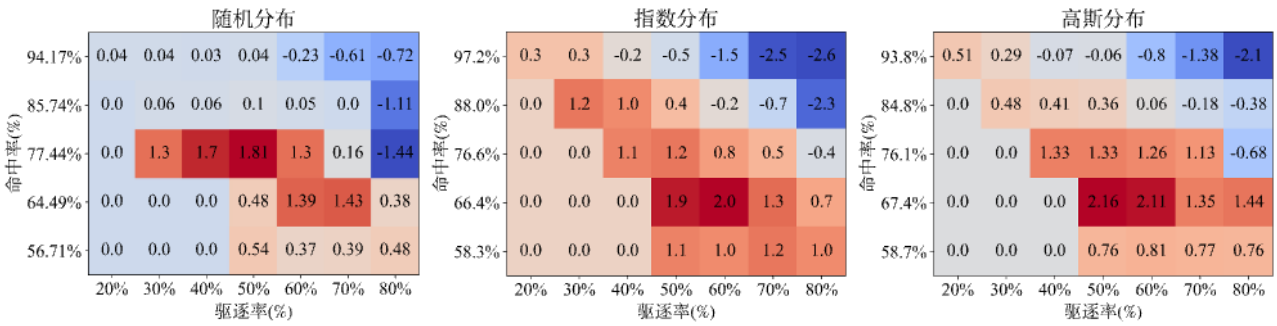



图 6 页面预选机制对 MM-CLOCK 命中率的影响

## 5.2 自适应写感知权重参数设置

**实验设计与目标。**在写感知权重 $W$ 的实验中，我们继续采用表 1 中的三类典型工作负载，并设定读写比为 90 / 10。首先，在未启用页面预选机制的条件下，运行原始 MM-CLOCK 算法，并通过调整缓冲池容量，将命中率控制在 50% - 100% 区间。随后，引入页面预选机制来降低延迟，并在不同的命中率条件下测试多个 $W$ 取值，从而评估其对命中率与写回量的影响。该实验旨在探讨写感知策略在不同缓存状态下的实际适应性，并为参数设置提供更具通用性的指导依据。

**参数设置与敏感性分析。**基于分段命中率，我们构建了写感知权重与性能变化的曲线图（图 7）。需要说明的是，图 7 最后一幅子图中的“50+”表示命中率区间 (50%, 60%]，其余标注以此类推。在分析过程中，我们固定工作负载和驱逐率参数，并在每一个命中率区间内遍历 $W \in (0, 9)$，观察其对系统命中率和写放大的影响。这一过程充分展示了写感知权重在不同工作负载下的性能表现。

从图中可见，$W$ 的值越大，页面写回延迟越高，写回量显著减少，但这同时也会降低缓存的页面替换精度，导致命中率下降，尤其是在命中率较低的场景下。例如，当命中率在 50% - 70% 区间时，$W$ 的取值大于 2 会造成命中率下降超过 4%；而当命中率高于 90% 时，系统对较大的

写延迟容忍度明显增强，命中率下降趋势减缓，说明此时可以容许较高的写感知权重以降低写放大代价。通过性能分析，我们提出了命中率区间与写感知权重 $W$ 的映射关系（表 2），从而构建了写策略的“命中率感知性”。

**基于命中率驱动的自适应策略。**为构建策略的“自适应”特性，我们提出一种轻量级写感知权重动态调节方法：系统在每轮 epoch 结束后，实时统计当前的命中率 $H$，并依据表 2 中构建的映射关系动态计算并更新当前的写感知权重 $W$。该机制无需依赖工作负载的先验信息与复杂的学习模型，可以直接利用系统的性能指标实现权重的自适应调整，具有计算成本低、部署简单、适应性强等优势，能够显著增强算法在动态工作负载下的响应能力。例如，当 $H = 92\%$ 时，系统自动设置 $W = 4$；而当命中率下降至 65% 时，系统将 $W$ 降为 1，以增强页面可替换性，避免性能的进一步恶化。

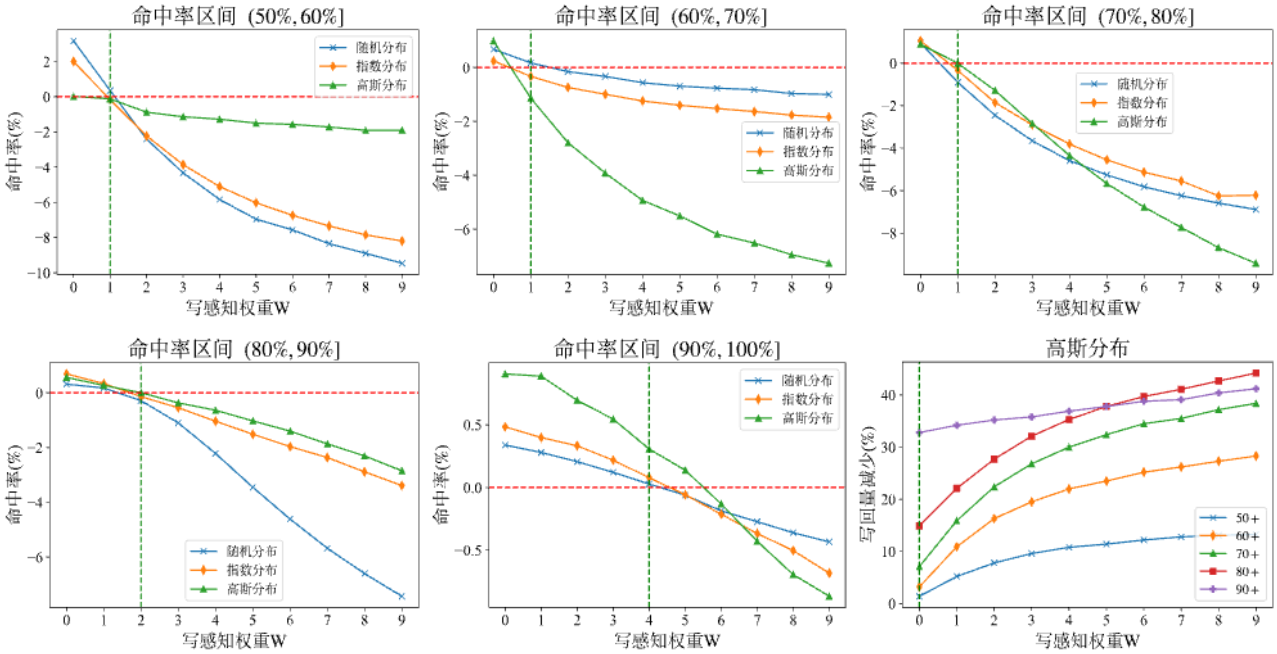



图 7 写感知权重对 MM-CLOCK 命中率和写回量的影响

表 2 基于命中率的自适应写感知权重设置

| 命中率区间 | 写感知权重 $W$ |
|---|---|
| 0-50 | 0 |
| 50-80 | 1 |
| 80-90 | 2 |
| 90-100 | 4 |

## 6 实验结果与分析

在本节中，我们首先使用 WATT 框架对 MM-CLOCK 算法进行性能评估，并将其与当前最先进的算法以及数据库中广泛应用的经典替换算法进行对比。随后，我们将 MM-CLOCK 集成至 PostgreSQL 16.1，并使用表 1 所列出的工作负载进行实验，验证各个核心组件对 MM-CLOCK 系统性能的实际影响。

**实验硬件和环境配置。**所有实验均在一台配备 32 核 2.50GHz 的 Intel Xeon Platinum 8163 处理器和 256GiB 内存的机器上运行，操作系统为 Ubuntu 20.04。存储设备为一块 1788GB 的 NVMe SSD，采用 ext4 文件系统格式。

**仿真评估框架。**我们使用 WATT 框架模拟数据的页面访问过程。该框架能够模拟读写访问路径，并据此评估不同页面替换算法的运行性能。

**PostgreSQL 集成实现。**在多核服务器架构下，LRU、2Q 和 ARC 等替换算法通常容易受到锁竞争的问题影响，难以提供良好的可扩展性。为了解决这一问题，自 PostgreSQL 8.1.0 版本以来，默认页面替换策略已更换为支持无锁并发的 Clock Sweep 算法。MM-CLOCK 算法同样采用无锁设计，并使用原子操作管理时钟指针，有效提升了多核环境下的执行效率。

**工作负载概述。**我们选用了两组代表性工作负载对 MM-CLOCK 算法进行评估：

- 在 WATT 仿真环境中，如表 3 所示，我们使用了 TPC-C、TPC-E、Linkbench 以及三

类基于 Zipf 分布的负载：ZipfRO、dynZipfRO 和 dynZipfRW，以充分验证算法在多样化负载下的性能表现。

- 在 PostgreSQL 数据库中，我们则采用了第 5 节表 1 中所列的典型工作负载，涵盖不同读写特征与访问局部性场景。

表 3 WATT 仿真框架的工作负载数据统计

| 统计信息 | TPC-C | TPC-E | Linkbench | ZipfRO | dynZipfRO | dynZipfRW |
|---|---|---|---|---|---|---|
| 写数量 | 15.6% | 5.7% | 40.8% | 0% | 0% | 16.7% |
| 访问量 | 1M | 1.5M | 0.4M | 1M | 2M | 1.2M |
| 页面数 | 16,128 | 65,656 | 16,480,7 | 10,000 | 20,000 | 20,000 |

**基线算法。**在 WATT 仿真框架中，我们将 MM-CLOCK 与多种先进的页面替换算法和数据库中广泛使用的页面替换算法进行比较。在 PostgreSQL 实验中，我们选用了 Clock Sweep、CLOCK 和 Random 三种当前广泛应用于多个数据库系统中的典型替换算法作为基线。所有算法均按照原文中建议的最优参数进行配置，以确保公平和准确地进行比较。

### 6.1 WATT 框架运行效率评估实验

我们使用 WATT 仿真框架来模拟不同缓冲池大小下的所有请求，并统计各个替换算法的总执行时间，以此衡量系统的整体性能。

**实验结果。**图 8 展示了在六种工作负载下，各页面替换算法的运行时间比较。在现有的替换策略中，**S3-FIFO**[23] 作为一种先进的缓存替换算法，采用了快速降级策略。通过使用占比 10% 的小 FIFO 队列过滤冷数据，并将大多数对象存储在主 FIFO 缓存中。对于多次访问的热数据，S3-FIFO 限制页面被访问超过 3 次后，系统不再额外提高页面重要性，过于保守的策略限制了其对频繁访问数据的适应性，影响了整体缓存效率。

**ARC** 则通过使用两个 LRU 队列来分别维护最近访问的对象和频繁访问的对象。两个队列的大小会根据历史队列命中情况进行自适应调整，以优化数据存储。然而，尽管具备一定的自适应能力，但在负载较高时，即使频繁访问队列较长，队列中的热数据仍然可能被较早淘汰，限制了其性能。

**WATT** 是一种频率感知算法，它通过使用时间戳对页面进行优先级排序，并计算所有子频率的最大值作为页面的优先级，同时考虑访问的最近性和频率。然而，WATT 算法未能有效捕捉工作负载的倾斜特征，导致它难以根据不同的工作负载变化灵活调整策略。

**LIRS**[24] 通过将缓存中的数据分为最近访问过的数据和再次访问的数据，将再次访问的数据提升为 LIR 块。然而，类似于 ARC 算法，对于频繁访问的热数据也未能赋予更高的优先级，导致其在处理间歇性访问负载时表现不佳。

**LRU-2**[25] 使用两级访问历史记录保留频繁访问的页面，通过引入第二次访问时间来衡量页面重要性。与只记录最近一次访问时间的 LRU 不同，LRU-2 记录页面的最近两次访问，优先保留“至少被访问两次且第二次访问距离当前较近”的页面。该方法与 LIRS 类似，在多次访问的场景下，它也未能充分提高热数据的缓存命中率。

**Hyperbolic Caching 和 SIEVE** 是现代 Web 缓存管理中的两种重要算法。Hyperbolic Caching 通过结合随机采样与延迟评估，灵活调整缓存优先级，提升大规模缓存系统效率。然而，在高负载和动态工作负载下，计算开销较大。SIEVE 则使用简化的 FIFO 队列和指针机制，具有高吞吐量和良好的扩展性，但在顺序扫描型工作负载下表现较差，且缺乏足够的灵活性。

**LRU-C** 引入一个指向链表尾部首个干净页面的 LRU-C 指针，缺页时优先替换该指针所指向的页面，并通过动态批量写回机制，定期将尾部脏页刷回磁盘。尽管该策略可提升写效率，但周期性刷新可能导致部分即将被访问的页面被过早驱逐，从而引发命中率下降与系统性能波动。

相较于上述算法，MM-CLOCK 在所评估的多种工作负载下展现出良好的性能优势。与先进的替换算法 S3-FIFO 相比，MM-CLOCK 在运行速度上实现了约 5.17 倍的提升；相较于采用类似刷脏机制的 LRU-C 算法，其性能提升达 1.64 倍。这一提升主要归功于其两个关键机制：页面预选机制和写回优化策略。页面预选机制通过提前选择候选替换页面，避免了在页面驱逐过程中进行低效页面扫描，从而显著减少了计算开销。而写回优化策略则通过延迟写回机制，使脏页面尽可能长时间保留在缓冲区中，并主动将脏页面

提前写回存储设备，减少了 I/O 操作的序列化延迟，提高了系统的整体性能。在后续实验中，我们将在 PostgreSQL 中进一步探讨与 CLOCK 及其变体 Clock Sweep 的对比表现。

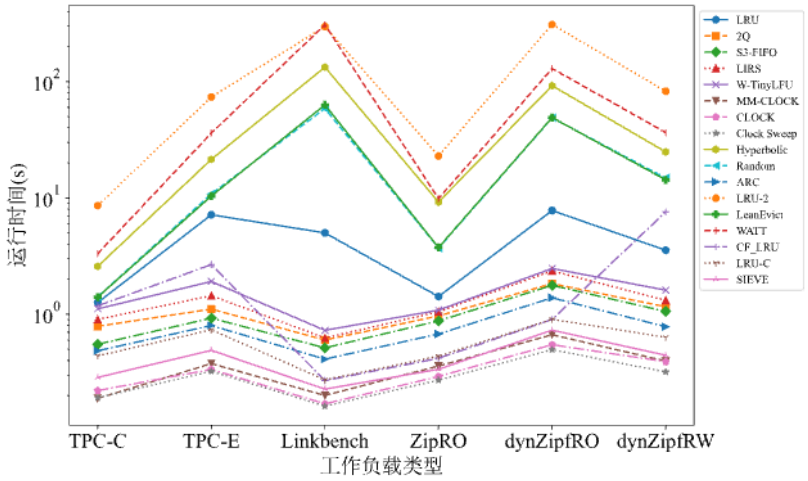


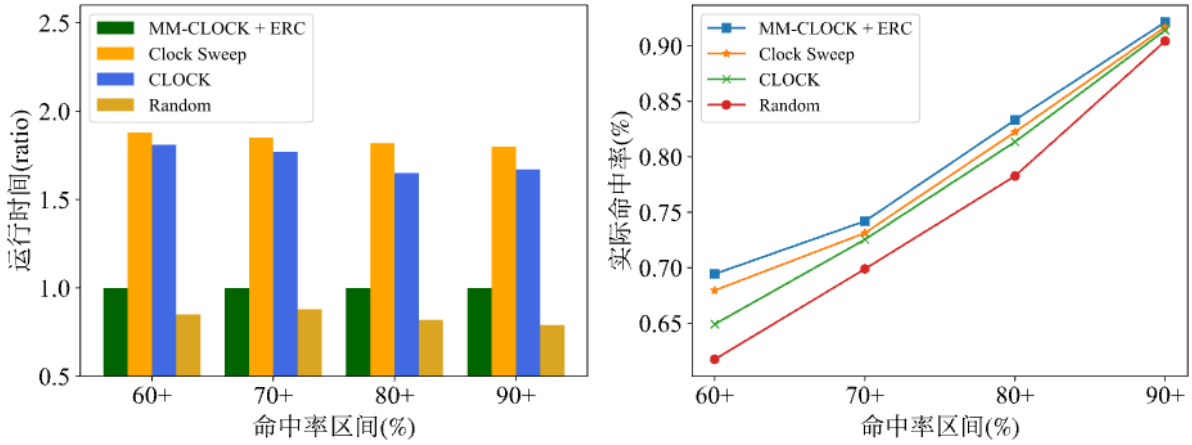

图 8 WATT 模拟框架中各个算法的运行时间对比

## 6.2 页面预选机制评估实验

为了进一步验证驱逐率计算器的影响，即页面预选机制的实际效用，我们将其与基线算法单独进行对比。在实验过程中，使用 pgbench 构建读写比为 90 / 10 的工作负载，调整缓存大小与工作负载参数，将命中率控制在 50% 至 90% 范围内。

**减少运行时间。**如图 9a 所示，在不同的命中率场景下，MM-CLOCK 算法相比 Clock Sweep 和 CLOCK 平均实现了 1.5 至 1.7 倍的加速，接近 Random 算法的性能。传统的时钟类策略在页面访问计数器为零时，需要进行第二次扫描来确认冷页，而 MM-CLOCK 算法可以利用驱逐率信息提前选择合适冷页进行淘汰，从而有效减少了扫描延迟，提高了系统的运行速度。尽管 Random 算法运行速度最快，但由于忽视了页面的冷热特性，导致其命中率较低，整体性能不如 MM-CLOCK 算法。

**提高命中率。**图 9b 展示了 MM-CLOCK 算法在不同命中率区间下始终优于其它基线算法的表现。Clock Sweep 由于减少了热页的引用计数器，容易过早驱逐热页面，导致命中率下降。而 CLOCK 算法仅使用一个访问位判断页面状态，未能充分考虑访问频率，导致其性能逊色于 Clock Sweep。尽管 Random 算法设计较为简单，但由于忽视了页面冷热程度的差异，其命中率始终较低。MM-CLOCK 算法通过优先选择冷页面进行驱逐，不仅有效降低访问延迟，还实现了最大化缓存命中率。




（a）运行时间 （b）实际命中率

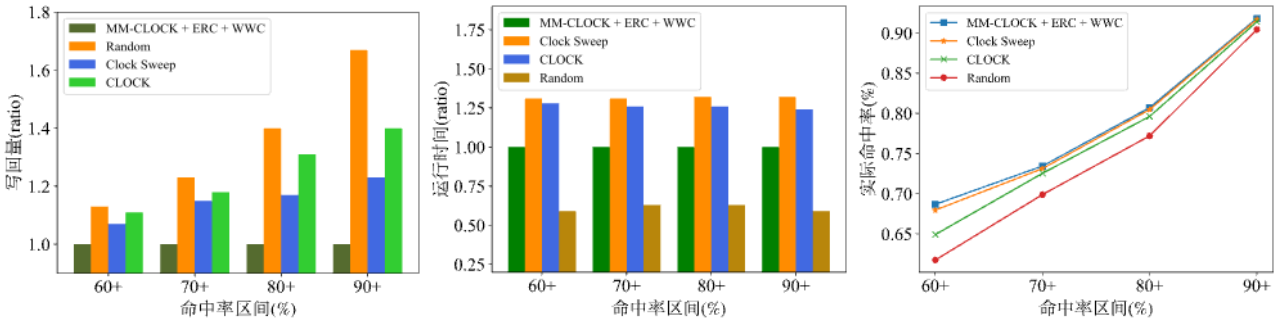
图 9 应用 ERC 的 MM-CLOCK 与其他算法的性能对比

### 6.3 自适应写感知策略评估实验

在本实验中，我们通过调整缓冲池大小和工作负载参数，将系统的命中率控制在 60% 至 99% 范围内，以实现有效地评估自适应写感知权重 $W$ 对 MM-CLOCK 算法性能所产生的影响。实验的重点是统计系统产生的总写回次数，并深入分析写感知权重对于系统运行速度和命中率造成的具体影响。

**降低写回量。**如图 10a 所示，MM-CLOCK 算法在减少写回次数方面表现出显著的优势，与其他页面替换算法相比，其写回次数的最大减少幅度可以高达 23%。在设定的 60% 至 70% 的命中率区间内，Random、Clock Sweep 和 CLOCK 三种算法的写回开销分别比 MM-CLOCK 算法高出 13%、7% 和 11%。除此之外，随着系统命中率不断提高，MM-CLOCK 算法的自适应写感知策略优势愈加明显。当命中率超过 90% 时，相比于 MM-CLOCK 算法，Random、Clock Sweep 和 CLOCK 增加的写回数量分别为 67%、23% 和 40%。这一实验结果表明，MM-CLOCK 算法通过优先驱逐干净页面，并自适应地跳过一定数量的脏页面，有效提升了其写感知能力，实现写回操作优化，减少了不必要的 I/O 开销。

**执行速度与命中率表现。**图 10b 和图 10c 显示了带有驱逐率和写感知权重的 MM-CLOCK 算法在运行速度和命中率方面的表现，在整体上优于其他页面替换算法。如图 10b 所示，在不同的命中率区间下，MM-CLOCK 算法的运行速度比 Clock Sweep 和 CLOCK 平均快 1.2 倍，但不及 Random 算法的速度。图 10c 则显示，在应用写感知权重的情况下，MM-CLOCK 算法依然能够保持较高的命中率，没有因为将脏页面长时间保留在缓冲池中而造成其命中率下降。




（a）写回量　　（b）运行时间　　（c）实际命中率

图 10　应用 ERC 和 WWC 的 MM-CLOCK 与其他算法的性能对比

### 6.4 阈值驱动的主动页面写回机制评估实验

在本节中，我们通过实验验证了阈值驱动的主动页面写回机制（即提前页面写回器）对系统性能的提升效果。为了评估该机制的影响，我们使用 pgbench 生成了不同分布（随机、高斯、指数）下的混合读写负载。实验初期，我们将读写比设置为 1 : 1，考察不同缓存比例对系统性能的影响。随后，固定缓冲池大小为数据集大小的 10%，在多个读写比下测试每种算法的每秒事务数（Transactions Per Second，TPS），以验证其事务吞吐能力。实验结果如图 11 所示。

**提高吞吐量。**如图 11a 所示，在引入阈值驱动的主动页面写回机制后，MM-CLOCK 算法的 TPS 提升了约 20% – 30%，尤其是在缓存较小的情况下，这一提升更加显著。尽管较高的缓存比和倾斜数据分布本身能够提升命中率并减少页面驱逐次数，但该机制在这些场景下依然带来约 20% 的性能增益。这表明其在多种缓存配置下均具备性能优化能力，特别是在内存资源受限的情况下，其性能提升更加明显。图 11b 进一步验证了该机制在写密集型负载下的有效性：当读写比在 25% – 45% 之间时，主动页面写回机制带来的 TPS 提升依然显著，表明主动写回机制能有效缓解写操作带来的性能瓶颈。

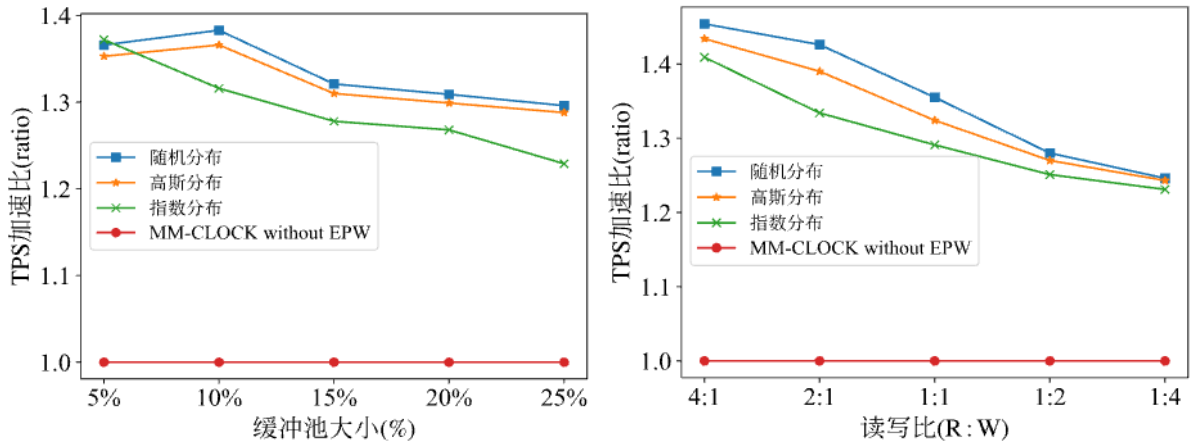
这一性能提升的根本原因在于，MM-CLOCK 所采用的阈值驱动的主动页面写回机制，不仅充分利用了 NVMe SSD 的高 I/O 并行性，还缓解了其读写速度不对称所带来的性能瓶颈。该机制通过实时监控缓冲池状态，以阈值驱动的方式动态执行脏页面的批量写回，避免了传统串行方法在读取过程中因淘汰脏页面而导致的阻塞问题。通过这一机制，显著减少了写延迟对读请求响应时间的负面影响，有效缓解了因读写不对称性带来的性能瓶颈。

此外，由于 MM-CLOCK 写回后的页面依旧保留在缓冲池中，避免了不必要的写入和误驱逐带来的 SSD 写放大问题。在高负载和高并发的环境下，MM-CLOCK 能够实现更高的吞吐量，并有效保障系统的稳定性和响应速度。

**其他优势。**表 4 展示了在不同的缓存大小、读写比为 1 : 1 以及指数分布工作负载下，阈值驱动的主动页面写回机制对命中率和写回次数的影响。实验结果表明，主动页面写回机制对命中率的影响较小，因为页面预选机制能够有效识别冷页面并将其作为候选页面在后续替换中进行淘汰，从而保持较高的命中率。即便在大量查询请求的情况下，提前写回所带来的影响也相对较小，进一步验证了该机制的有效性。

表 4 EPW 对命中率和写回量的影响

| 缓冲区大小(%) | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| 未命中率(%) | 0.001 | 0.001 | 0.005 | 0.008 | 0.001 |
| 额外写回(%) | 0.01 | 0.02 | 0.04 | 0.07 | 0.08 |




（a）不同缓冲池大小下的 TPS （b）不同读写比下的 TPS

图 11 应用 EPW 的 MM-CLOCK 性能表现

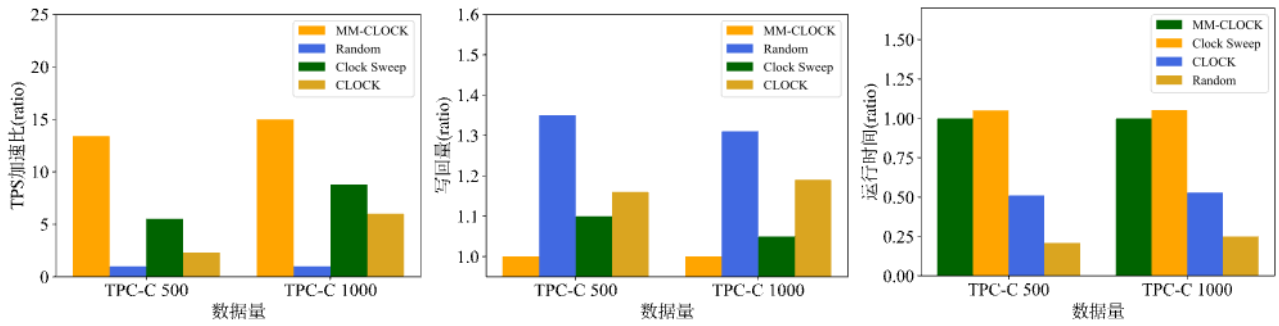
## 6.5 整体性能

数据库系统的整体性能不仅受到硬件资源的制约，还深受缓存管理设计与实现的影响。前文已详细评估了 MM-CLOCK 算法三个核心模块在不同工作负载下的表现。本节将通过 TPC-C 基准测试，进一步分析 MM-CLOCK 算法在实际数据库环境中的应用表现。在实验配置中，我们分别构建了包含 500 个（约 50GB）和 1000 个（约 100GB）数据仓库的 TPC-C 实例，统一采用 1GB 大小的缓冲池，并启用 32 个并发终端。实验围绕 TPS、命中率和写回量三项关键指标进行展开，结果如图 12 所示。

**总体性能分析。**实验结果表明，MM-CLOCK 算法在 TPC-C 基准测试中表现出了优秀的性能。在 500 个仓库的配置下，MM-CLOCK 算法相比先进的基线算法 Clock Sweep，TPS 提升了 7%，而与 Random 算法相比提升了 13.4%。在 1000 个仓库的场景下，MM-CLOCK 的性能提升幅度进一步扩大，分别达到了 8% 和 15%。结果表明，本文提出的 MM-CLOCK 算法在 TPC-C 负载下展现出比现有算法更好的性能表现，尤其是在更大规模的数据库中。

**脏页面写回优化。**MM-CLOCK 算法通过写感知权重策略的优化实现了高效的写操作调度，相比 Clock Sweep 算法，写回量减少了约 6%。而传统的 CLOCK 算法的性能相比 Clock Sweep 更差，写回量增加了 19.7%。尽管 Random 算法的运行速度较快，但由于其随机替换特性导致其写回开销是最高的，写回量比 MM-CLOCK 算法多出 31%。这些结果表明，MM-CLOCK 算法在减少不必要的写回操作方面具有显著优势，在频繁写操作的数据库场景下表现突出。

**系统运行速度提高。**在系统运行速度方面，具有页面预选机制的 MM-CLOCK 算法相较于 Clock Sweep 算法的运行速度仅提高了 7%。这一现象的原因在于 TPC-C 工作负载中的事务具有较高的命中率，这使得根据驱逐率计算器计算得到的驱逐率较低，页面预选机制的优势未能得到充分发挥。然而，尽管 CLOCK 和 Random 这两种算法具有较快的运行速度，但由于它们在写感知上的表现较差，导致它们的整体吞吐能力低于 MM-CLOCK 算法。




（a）TPS （b）写回量 （c）运行时间

图 12 MM-CLOCK 与其他算法在 TPC-C 工作负载下的性能对比

## 7 结束语

本文提出了一种新型页面替换算法——MM-CLOCK 算法，该算法兼具无锁设计与写感知能力等特性，旨在提高数据库系统的整体性能。MM-CLOCK 算法基于动态驱逐率的设计，结合页面预选机制提前筛选出一组候选页面，从而降低页面替换时的计算开销，提高页面替换的效率。同时，MM-CLOCK 算法引入的自适应写感知策略根据系统状态动态调整脏页写回行为，有效减少不必要的脏页写回操作，降低 I/O 损耗，延长存储设备的使用寿命。此外，MM-CLOCK 算法还引入了一种阈值驱动的主动页面写回机制，该机制能够在脏页面替换阶段开始执行之前主动将其刷新到存储设备中，从而减少因脏页面写回而产生的延迟，提升了系统的运行速度和吞吐量。WATT 模拟框架中的实验结果表明，MM-CLOCK 算法在多种典型工作负载下充分展现出良好的性能优势。

为了验证 MM-CLOCK 算法在真实系统中的效果，我们进一步将其集成到 PostgreSQL 数据库系统中，并与三种主流无锁算法进行实验对比。实验结果表明，MM-CLOCK 算法在运行速度、吞吐量和命中率方面均表现出色，充分证明了其在现代硬件环境中的高效性和实用性。未来我们将继续探索在充分利用现代硬件特性的基础上，进一步提升页面替换策略的精度，尤其是在冷页面识别与驱逐优先级方面进行优化。突破仅依赖页面值进行决策的局限，探索在多个页面值相同的情况下，如何准确识别更冷的页面并优先替换，从而提升系统在复杂负载下的整体性能。

作者贡献声明 刘艳雪与徐军为共同一作.

## 参 考 文 献

[1] Megiddo N, Modha D S. ARC: A Self-Tuning, low overhead replacement cache//Proceedings of the 2nd USENIX Conference on File and Storage Technologies. San Francisco, USA, 2003: 115-130.

[2] Vietri G, Rodriguez L V, Martinez W A, et al. Driving cache replacement with ML-based LeCaR//Proceedings of the 10th USENIX Workshop on Hot Topics in Storage and File Systems. Boston, USA, 2018: 928-936.

[3] X, Wang S, Liang X, et al. Deep reinforcement learning: A survey. IEEE Transactions on Neural Networks and Learning Systems, 2022, 35(4): 5064-5078.

[4] Riggs H, Tufail S, Parvez I, et al. Survey of solid state drives, characteristics, technology, and applications// Proceedings of the 2020 SoutheastCon, Raleigh, USA, 2020: 1-6.

[5] Vöhringer D, Leis V. Write-aware timestamp tracking: Effective and efficient page replacement for modern hardware. Proceedings of the VLDB Endowment, 2023, 16(11): 3323-3334.

[6] Christudas B. MySQL//Practical Microservices Architectural Patterns: Event-Based Java Microservices with Spring Boot and Spring Cloud. Berkeley, USA, 2019: 877-884.

[7] Lahiri T, Chavan S, Colgan M, et al. Oracle database in-memory: A dual format in-memory database// Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. Seoul, Republic of Korea, 2015: 1253-1258.

[8] Tian W, Martin P, Powley W. Techniques for automatically sizing multiple buffer pools in DB2//Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research. Toronto, Canada, 2003: 294-302.

[9] Johnson T, Shasha D. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm//Proceedings of the 20th International Conference on Very Large Data Bases. San Francisco, USA, 1994: 439–450.

[10] Rodriguez L V, Yusuf F, Lyons S, et al. Learning cache replacement

with CACHEUS// Proceedings of the 19th USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2021: 341-354.

[11] Xia Z, Bu T. The implementation of flash-aware buffer replacement algorithms in PostgreSQL//Proceedings of the 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery. Zhangjiajie, China, 2015: 1215-1219.

[12] Smith A J. Sequentiality and prefetching in database systems. ACM Transactions on Database Systems, 1978, 3(3): 223-247.

[13] Khuri S, Hsu H C. Visualizing the CPU scheduler and page replacement algorithms//Proceedings of the thirtieth SIGCSE technical symposium on Computer science education. New Orleans, USA, 1999: 227-231.

[14] Jiang S, Chen F, Zhang X. CLOCK-Pro: An Effective Improvement of the CLOCK Replacement// Proceedings of the USENIX Annual Technical Conference, General Track. Anaheim, USA, 2005: 323-336.

[15] Bansal S, Modha D S. CAR: Clock with Adaptive Replacement// Proceedings of the 3rd USENIX Conference on File and Storage Technologies. San Francisco, USA, 2004: 187-200.

[16] Park S, Jung D, Kang J, et al. CFLRU: a replacement algorithm for flash memory//Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems. New York, USA, 2006: 234-241.

[17] Jung H, Shim H, Park S, et al. LRU-WSR: integration of LRU and writes sequence reordering for flash memory. IEEE Transactions on Consumer Electronics, 2008, 54(3): 1215-1223.

[18] Blankstein A, Sen S, Freedman M J. Hyperbolic caching: Flexible caching for web applications// Proceedings of the 2017 USENIX Annual Technical Conference. Santa Clara, USA, 2017: 499-511.

[19] Zhang Y, Yang J, Yue Y, et al. SIEVE is simpler than LRU: an efficient Turn-Key eviction algorithm for web caches//Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation. Santa Clara, USA, 2024: 1229-1246.

[20] Lee B, An M, Lee S W. LRU-C: Parallelizing Database I/Os for Flash SSDs. Proceedings of the VLDB Endowment, 2023, 16(9): 2364-2376.

[21] Taipalus T. On the effects of logical database design on database size, query complexity, query performance, and energy consumption. arXiv preprint arXiv:2501.07449, 2025.

[22] Tsuei T F, Packer A N, Ko K T. Database buffer size investigation for OLTP workloads. ACM SIGMOD Record, 1997, 26(2): 112-122.

[23] Yang J, Zhang Y, Qiu Z, et al. FIFO queues are all you need for cache eviction//Proceedings of the 29th Symposium on Operating Systems Principles. New York, USA, 2023: 130-149.

[24] Jiang S, Zhang X. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. ACM SIGMETRICS Performance Evaluation Review, 2002, 30(1): 31-42.

[25] O'neil E J, O'neil P E, Weikum G. The LRU-K page replacement algorithm for database disk buffering. ACM Sigmod Record, 1993, 22(2): 297-306.

**LIU Yan-Xue**, 2002, Master, her main research interest includes database memory management.

**XU Jun**, 1986, Ph.D., his main research interest includes datacenter networks.

**GE Song-Yang**, 1994, Ph.D., her main research interest includes distributed learning.

**Ji Zhong-Ming**, 1994, Ph.D., his current research interests include datacenter networks and cloud computing.

**Meng Ling-Kun**, 1990, M.E., his current research interests include cloud computing and edge computing.

**Li Xin**, 2000, Master, his main research interest includes database memory management.

**Cheng Guan-Jie**, 1996, Ph.D., his current research interests include data intelligence services and systems.

**Yuan Gong-Sheng**, 1991, Ph.D., his main research interest includes database systems.

**Chen Gang**, 1973, Ph.D., his current research interests include databases, big data management systems, big data intelligent computing.

**Background**

This research belongs to the field of memory management, especially focusing on the design and evaluation of page replacement algorithms for database buffer pools. As modern database systems increasingly rely on large memory resources to support high-throughput and low-latency data access, the efficiency of page replacement strategies has become a key factor in system performance. However, with the rapid advancement of storage hardware — such as NVMe SSDs — traditional page replacement algorithms are facing great challenges and new opportunities. Although several new page algorithms (e.g., ARC and LeCaR) have been proposed in the field of databases, many of them employ complex mechanisms such as reinforcement learning, which introduces non-trivial computational overhead. While such optimizations may provide benefits in non-compute-intensive scenarios, their resource requirements make them suboptimal for high-performance networking or database systems, where latency and throughput are critical constraints.

This work proposes MM-CLOCK, a novel variant of the

CLOCK page replacement algorithm, specifically designed to leverage the capabilities of modern hardware to improve system performance. By introducing a page pre-selection mechanism and an adaptive write-awareness strategy, MM-CLOCK effectively improves replacement efficiency, reduces unnecessary write-back operations and I/O latency, and improves the database query efficiency. Furthermore, we redesign the buffer pool architecture by introducing a threshold-driven proactive page writeback mechanism to fully utilize the concurrency characteristics of the novel hardware to improve the overall throughput of the system. This research is significant as it provides a more efficient buffer pool management solution for database systems, improves system throughput, and extends device lifespan. Experimental results demonstrate that our proposed algorithm performs effectively in practice and offers valuable insights and engineering applicability value for the system.

This research is supported by China Mobile, the National Key Research and Development Program (2023YFC3603103), and the “Pioneer” and “Leading Goose” R&D Program of Zhejiang (2024C01019).