武汉理工大学毕业设计（论文）

面向数据库系统的新型存储架构研究与优化

学院（系）：计算机科学与技术学院

专业班级：计算机 1701 班

学生姓名：刘宗惠

指导教师：杜亚娟

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：

年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密□，在　　年解密后适用本授权书

2、不保密□。

（请在以上相应方框内打“√”）

作者签名：　　年　　月　　日

导师签名：　　年　　月　　日

（注：此页内容装订在论文扉页）

摘　要

非易失性存储器具有存储密度高、待机功耗低以及可字节寻址等优点，因此它被认为是一种能够代替或补充现有存储系统的器件。但是，非易失性存储器也具有读写不对称和写寿命短等缺点，因此如何避免无效写操作以延长 NVM 的使用寿命是当前研究人员亟待解决的问题。

本文研究了当前基于非易失性存储器的存储系统架构并分析了它们各自的优缺点。随后，面向 Key-Value 内存数据库，本文重点研究了 DRAM-NVM 的层次混合内存结构存在的问题。论文研究了非易失性存储器命中率低的可能原因，并针对页内热度分布不均这一原因设计了一种基于访存热度的影子 B+树索引方法。本文在 Ubuntu 18.04 环境下，使用 YCSB 对影子 B+树方法进行基准测试。研究结果表明该方法能够有效减少页面热度差异，进而提升系统性能。实验证明使用影子 B+树方法的系统命中率提升稳定在 20%。

关键词：非易失性存储；混合内存架构；访存热度；影子 B+树方法

Abstract

Non-volatile memory(NVM), with the advantages of high storage density, low standby power consumption and byte addressability, is considered to be a device that can replace or supplement traditional storage systems.

This thesis studies the current storage system architecture based on non-volatile memory and analyzes their respective advantages and disadvantages. Subsequently, facing the Key-Value in-memory database, this thesis focuses on the problems of the DRAM-NVM hierarchical hybrid memory structure. The thesis studied the possible reason for the low hit ratio of non-volatile memory, and I designed a shadow B+ tree index based on the access popularity. This method could avoid the uneven distribution of cache lines within pages. This thesis uses YCSB to benchmark the shadow B+ tree method in the Ubuntu 18.04 environment. The results show that this method can effectively reduce the difference among cache lines in pages, and it can improve the whole system performance. Experiments' results show that the hit ratio improves 20% after using the shadow B+ tree method.

Key Words： Non-volatile memory(NVM)；hierarchical hybrid memory; access popularity；shadow B+ tree index

目 录

摘 要 I
Abstract II
第 1 章 绪论 1
1.1 研究背景及意义 1
1.2 国内外研究现状 2
1.2.1 NVM 的种类及性能对比 2
1.2.2 NVM 的应用 3
1.2.3 内存数据库系统 4
1.3 本文主要内容 5
1.4 本文组织结构 5
第 2 章 基于 DRAM 和 NVM 的混合内存架构 7
2.1 NVM 用作主存 7
2.2 NVM 用作外部存储器 8
2.3 基于 NVM 的混合内存 9
2.3.1 NVRAM 的平行结构 9
2.3.2 DRAM-NVM 的垂直结构 10
2.4 面向内存数据库的混合内存 11
2.4.1 DRAM-NVM 混合内存缓冲管理 11
2.4.2 缓存行粒度的 NVM 页面 12
第三章 热度感知的性能优化方法 14
3.1 方法动机及分析 14
3.1.1 传统三层架构概述 14
3.1.2 传统架构性能性能分析 15
3.2 系统性能问题调研 17
3.2.1 NVM 命中率低的原因 17
3.2.2 系统调研 17
3.3 方法整体结构 19
3.4 热度感知的动态调整策略 20
3.5 影子 B+树的基本操作 21

3.5.1 节点的插入.....21
3.5.2 页驱逐及节点的删除.....21
3.6 复杂度和开销分析.....22
第 4 章 实验与结果分析.....24
4.1 实验环境及参数设置.....24
4.1.1 模拟平台及参数.....24
4.1.2 数据集及参数.....24
4.2 实验结果及分析.....25
第 5 章 总结与展望.....28
5.1 总结.....28
5.2 展望和未来工作.....28
参考文献.....29
致 谢.....31

第 1 章 绪论

本章首先将阐述新型存储系统研究的必要性和意义，随后将简述非易失性存储系统的国内外研究现状和内存数据库的应用现状。本章第三节将介绍文章的主要内容，第四节将展示全文的组织结构。

1.1 研究背景及意义

随着人工智能模型和大数据应用的快速发展，人类对计算机软硬件资源的需求量逐渐增加。这意味着曾经被广泛应用的计算机系统结构如今已经不再适用，因此针对体系结构的优化与革新是势在必得的。近年来，由于 CPU 计算速度的快速增长以及 CPU 与内存子系统之间的速度差异逐渐增大，存储子系统的性能逐渐成为制约整个系统性能提升的瓶颈[1]。

传统的计算机存储子系统采用动态随机存储器（Dynamic Random Access Memory，DRAM）作为主存储器，并且采用磁盘或固态硬盘（Solid State Disk, SSD）作为辅助存储器。DRAM 具有快速读写 CPU 数据的能力，但由于其单位密度较小，很难在保持芯片大小不变的条件下增大容量；同时由于 DRAM 在使用过程中逐渐掉电，因此需要每隔一段时间对其充电，否则将会失去存储在其中的全部数据，反复充电刷新的过程会导致系统能耗迅速增加[2]。SSD 等永久性存储设备具有较大的存储容量，但其于 DRAM 之间读写速度的差异使得系统具有较大的 I/O 时延。因此，具有较大容量和低能耗的非易失性存储器（Non-Volatile Memory, NVM）逐渐引起研究人员们的关注。

新型非易失性存储器（NVM）具有较高的存储密度，同样大小的 NVM 容量是 DRAM 容量的几倍到几十倍。由于其非易失的特点，因此不需要反复的充电刷新，这使得 NVM 的待机功耗趋近于零[1]。在读写速度方面，NVM 具有和 DRAM 相近的存取速度，DRAM 的读取速度仅为 NVM 的 1 倍，写入速度仅为 NVM 的几倍。同时 NVM 具有可字节寻址的特点，因此是非常有潜力的新型内存设备。NVM 也存在缺陷：读写不对称，读取速度远大于写入速度；寿命短，大约只能承受 1000 万次写操作[3]，频繁更换 NVM 将会导致较高的成本；读写速度慢，虽然相较于 SSD 等辅助存储器 NVM 具有较快的访问速度，但相较于 DRAM 的读写性能依然较差。

虽然 NVM 有望成为现有存储系统的替代器件或部分器件，但是想要广泛应用 NVM 于计算机系统，仍需要解决很多问题。例如，针对 NVM 写操作寿命短的问题，如何减少对 NVM 的写操作或如何进行读写操作均衡是急需要解决的问题；针对于 NVM 相对较慢的读写速度，如何设计新型存储子系统的结构和缓冲模式也将成为新架构考虑的重点；

针对系统故障时 NVM 中保存的错误信息与辅助存储器保存信息之间产生的数据一致性问题，如何设计 NVM 的数据组织结构达到一致性和持久化的要求也将成为 NVM 广泛使用路上必须解决的问题。

随着互联网用户规模的膨胀和数据实时访问需求量的增加，服务器数量和运维成本也随之升高，现有的辅存数据库系统已经无法满足用户的需求[4]。Key-Value 内存数据库和传统磁盘数据库的混合数据库系统逐渐成为新的解决方案。服务器将被频繁访问的热数据缓冲到内存数据库中，而将步经常访问的冷数据保存在容量更大的辅存数据库中，这样可以将减少因设备 I/O 产生时延进而提升系统整体的性能。尽管内存数据库能够减轻服务器压力同时满足对实时性要求较高的应用程序，但是它也存在物理内存容量受限和数据断电丢失的问题。因此使用 NVM 存储 Key-Value 内存数据库中的热数据也成为当下研究的重点。

1.2 国内外研究现状

本节首先将介绍非易失性存储器的常见种类并对比其性能，其次将列举出现存的使用 NVM 的存储架构并对每一种架构进行简要分析，最后将介绍内存数据库的应用场景。

1.2.1 NVM 的种类及性能对比

针对 NVM 的研究主要集中于相变存储器（Phase Change Memory, PCM）、自旋转移力矩存储器（Spin-Transfer Toque RAM, STT-RAM）、电阻式存储器（Resistive RAM, ReRAM）等几种存储材料。PCM 存储器利用硫系玻璃晶体在不同状态下表现出的不同电阻值保存信息，并通过脉冲电流实现可控的状态变化[5]； STT-RAM 使用磁隧道结(Magnetic Tunnel Junction, MTJ)保存数据,通过控制自由层和参考层的相对磁力方向改变 MTJ[6]；电阻式存储器也称为忆阻器，通过在一个含有氧化物夹层结构的强相关电子材料两端施加不同大小和方向的电压改变其电阻值[7]。

NVM 在内存方面表现较好的同时，由于其可字节寻址的特性，相比于按页粒度访问数据的 SSD，也可提供更好的访问性能。NVM 与 SSD 的特性对比如表 1.1 所示。NVM 的密度是 NAND 闪存的 4~15 倍左右，但闪存的写带宽则是 NVM 的 25~30 倍[1]。因此，将 NVM 单独作为一个存储层是不高效的，通常将其与 SSD 共同使用。

表 1 非易失性存储技术的对比

NVM 类型	密度（F^2）	时延	写带宽	寿命
NAND	1-4	25μs	5-40MB/s	10^4
PCM	4-16	48-70ns	100MB/s	10^9
其他	4-60	10-100ns	140MB/s-1GB/s	10^{12}-10^{15}

1.2.2 NVM 的应用

自从 NVM 引起研究人员的关注后，科学界在 NVM 用作计算机存储器件领域做出了几次尝试。以下几种结构是在数据库管理系统方面具有较大潜力的存储结构：

1. 直接使用 NVM 替代 DRAM 作为计算机的主存储器[8][9]；
2. 使用 NVM 替代 SSD 等持久性存储设备，形成 DRAM-NVM 两层结构[10]；
3. DRAM 和 NVM 联合形成混合内存替代传统内存，其中 DRAM 用作辅助内存而由容量更大的 NVM 充当主要内存的平行结构[11]；
4. DRAM 和 NVM 联合形成混合内存替代传统内存，NVM 用作主要内存，DRAM 充当 NVM 缓冲区的 DRAM－NVM－SSD 三层结构[12]。

对于单一使用 NVM 的存储系统，由于 NVM 具有接近 DRAM 的读写速度和可字节寻址能力，因此科学界考虑使用 NVM 完全替代容量小且数据易失的 DRAM。Arulraj 等人对直接使用 NVM 替代 DRAM 的不同设计结构进行了研究。他们将数据库系统的操作类型划分为为就地更新，日志更新和写时复制，然后基于 NVM 优化以上三种操作类型。实验结果表明，在大多数情况下，基于 NVM 的就地更新操作可同时实现最高性能和最低损耗[13]。但是在实验过程中由于 NVM 写寿命短因此系统经常因此而崩溃。

对于 DRAM－NVM 的两层结构，类似传统的 DRAM－SSD 结构，所有页面都存储在较大容量的持久层（NVM）中。而容量较小的易失性层（DRAM）充当 NVM 的缓存层。事务仅在 DRAM 中执行，并使用 fix()和 unfix()在访问页面时对页面进行锁定。Kimura H 将内存分为固定大小的页面，事务仅在 DRAM 中运行，FOEDUS 系统无需像传统磁盘存储器一样存储页面标识符，而是存储两个指针。一个指针指向存储在 NVM 中的页面副本，另一个指针指向存储在 DRAM 中的页面副本（如果该页面不为空）。当系统在 DRAM 中找不到相应的页面时，系统会将该页面加载到 DRAM 缓冲区中[14]。鉴于上述过程和 NVM 的读写不平衡特性，系统从不直接回写页面，而是通过日志的方式间接回

写页面，这个过程无形之中会减少 DRAM 和 NVM 中可利用的有效存储空间。然而 NVM 的存储密度相比较 HDD 和 SSD 更小，因此该系统能够处理的数据量不能过大，这与数据库系统的任务需求相悖。

DRAM 和 NVM 的混合内存平行架构是在上述两个架构的基础上对系统架构进行的优化尝试，研究人员尝试使用混合内存替代传统的单一内存。Lee 等人提出了 DRAM 和 NVM 的平行混合内存架构：DRAM 和 NVM 处于同一层，均可以独立存放数据。系统对内存进行数据操作过程中，通过一定的分配策略选择数据存储在 DRAM 还是 NVM 中。并且根据 DRAM 和 NVM 各自的存储特点对数据进行读取和写回操作。他们设计的系统通过较为精确的页面更新预测，将未来可能操作的数据分配到 DRAM，将不会发生操作的数据分配到 NVM 中[15]。Ju 等人在此基础上进行优化，通过设置概率参数并对该参数进行计算自适应地选择 DRAM 或 NVM[16]。该架构在实验过程中被发现非常依赖于异构内存系统的分配策略和预测算法。这意味着该架构对于访问特征突出且固定的页面会展现出较好的性能。然而在真实应用场景中，页面的访问特征具有多样性因此现有的自适应内存分配策略无法正确分辨页面特征，进而无法较好地利用 DRAM 和 NVM 各自的存储特征。同时随着平行架构而来的额外内存元数据管理会在减小混合内存有效可利用空间的同时造成 CPU 的额外负担。

基于平行架构的 DRAM－NVM 混合内存存在的问题，Dhiman 等人将混合内存分为上下两个层次，上层的 DRAM 被看作是 NVM 的缓冲层。两部分共用内存空间并且统一进行地址管理。垂直结构的混合内存能够充分利用数据的热度和 DRAM、NVM 各自的存储特点并且十分适用于大量数据的应用场景[17]。然而该架构需要动态监控数据的热度并设计恰当的热度预测算法将热的数据提前缓冲进 DRAM 中，以避免 DRAM 和 NVM 之间频繁的数据交换。因此本文采用 DRAM-NVM-SSD 的三层存储架构，并在此基础上对数据的组织方式以及基本操作等进行优化。

1.2.3 内存数据库系统

随着互联网应用的蓬勃发展，服务器的数据访问量也呈现出“井喷”式的现象，同时人工智能技术和云计算服务的发展也催生出了更多的实时系统。因此传统的基于磁盘或 SSDs 的数据库系统已经越来越不能满足应用程序的低时延和高带宽的性能需求了。

随着 DDR4 等内存成本的降低，计算系统的内存容量也不断增加，内存数据库应运而生。20 世纪 60 年代便出现了内存数据库的原型，而内存数据库技术理论在上世纪 80 年代变得成熟并且逐渐商用。内存数据库在金融、政府和电信等领域有着广泛的应用[18]。

尽管内存数据库技术已经日臻成熟，但由于其无法在系统故障后保留当前数据，即无

法实现持久性，因此使得数据的安全性和一致性无法保障。因此研究人员尝试将内存数据库与传统磁盘数据库结合使得内存数据库成为上层的缓冲系统。

本文研究的对象是开源的 Key-Value 内存数据库 Redis。Redis 可既可以单独使用又可以作为磁盘数据库的中间设备。同时它还支持 Python、C/C++、Java 在内的多种语言客户端[20]。Redis 在 memcache 的基础上提供了更丰富的数据类型，即 string、list、set、hash 和 sort set。Redis 还提供了 push/pop 和 set/get 等[20]多种原子操作以实现持久化。

Redis 数据库进行换页操作时采用 LRU 策略，而本文研究的重点是在混合存储架构上设计实现一种更加高效的数据组织方法使得 Redis 换页操作的效率更高进而提升整个数据库的性能。

1.3 本文主要内容

本文主要的内容可分为两部分：

第一部分，研究了使用 NVM 存储器件的新型存储架构，分别讨论 NVM 直接替代 DRAM、DRAM-NVM 两层结构、DRAM-NVM 平行混合内存结构和 DRAM-NVM-SSD 三层存储结构各自的工作原理、优点以及存在的问题。最终选择三层存储架构作为本次实验的体系结构。

第二部分，统计传统三层体系结构的数据库系统的命中率，并分析未命中的原因。针对该原因，设计实现了一种 B+树影子的算法使得 NVM 的优点被充分利用进而提升整个系统的性能。实验结果证明该方法在 Redis 数据库运行 YCSB-Benchmark 条件下是有效的。

1.4 本文组织结构

针对上述内容，本文可分为五个章节，如图 1.1 所示，具体内容如下：

第一章，说明本文研究的背景和意义，阐述当前国内外关于 NVM 存储架构以及 Key-Value 内存数据库的研究方向和成果。同时，说明了本文主要研究的内容和全文的章节安排。

第二章，将详细介绍使用 NVM 的新型存储架构。本章将详细地介绍四种新型存储体系结构的工作原理、优化、优点以及现存的亟需解决的问题。随后通过对比现有的四种架构，选择最合适的层次混合内存架构。最后本章将详细介绍层次架构中各层的数据存储与组织粒度以及两层之间的数据交换粒度。针对以上各种数据粒度，本章后续章节将分析一种以缓存行为粒度的页面结构，该结构驻存在 DRAM 和 NVM 中，是后续算法研究的基础。

第三章，首先将统计三层存储架构的命中率发现当前系统的命中率有待提高，对NVM 未命中的原因做出猜想。后续调研过程中发现 NVM 页面中缓存行的热度分布不均，这很可能是造成 NVM 未命中的原因。据此，本文提出了一种影子 B+树方法。该方法能以缓存行为粒度，基于缓存行的热度，对 NVM 中缓存行进行自适应调整。本章后续将详细介绍影子 B+树的结构、自适应地迁移策略以及影子 B+树节点的增加和删除等。最后，本章将会对上述影子 B+树方法进行复杂度分析。

第四章，首先将介绍实验环境和相关参数。其次，将给出实验结果，分析实验结果可以得出结论：B+树影子方法对于系统性能提升有帮助。

第五章，总结本文所做的工作并且基于理论和实验过程中存在的问题对未来进一步的工作进行展望。

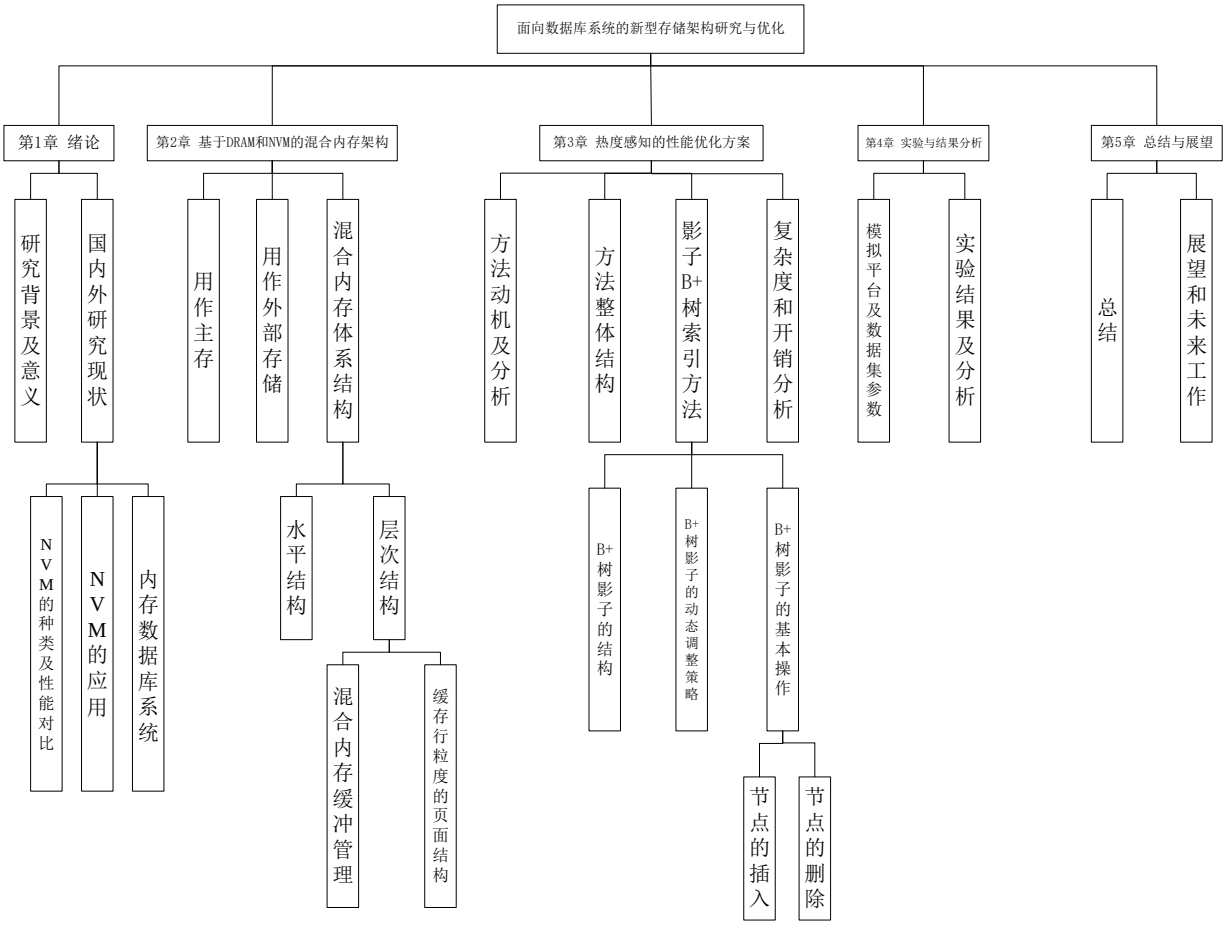


图 1.1 本文组织结构图

第 2 章 基于 DRAM 和 NVM 的混合内存架构

现有的研究成果中，研究人员已经提出了几种使用 NVM 进行优化的数据库系统结构。使用 NVM 的新型存储架构主要有四种应用形式：NVM 用作主存、NVM 用作外部存储器、DRAM-NVM 的平行结构和 DRAM-NVM 的垂直结构。本章前三节将总结回顾这些架构设计的主要内容和各自的优缺点，并且将它们与层次混合内存架构进行对比说明。本章第四节将详细介绍 DRAM-NVM 的垂直架构在数据库领域的应用，将分别从数据粒度和数据结构两方面进行说明。

2.1 NVM 用作主存

NVM 提供接近 DRAM 的读写时延和可字节寻址能力，因此它可以用作主要存储层。Arulraj 等人已经对 NVM 直接替代 DRAM 的系统进行了深入研究。他们的工作将数据库系统分为就地更新，日志记录和写时复制三部分，然后使用 NVM 分别对以上三部分进行优化[7]。实验结果表明，在大多数情况下，针对数据库的就地更新部分进行的 NVM 设计优化的性能提升最为显著，因此本文将详细介绍就地更新部分的 NVM 优化。

如图 2.1 所示，该设计将所有数据保留在 NVM 中，DRAM 仅用于存放临时数据，其地位相当于 Cache。同时日志也将保存在 NVM 中。这种结构的优点在于：1）通过最小化日志，系统崩溃后的恢复速度会提高；2）由于 CPU 可以直接访问可字节寻址的 NVM，因此读操作会变得更简单。

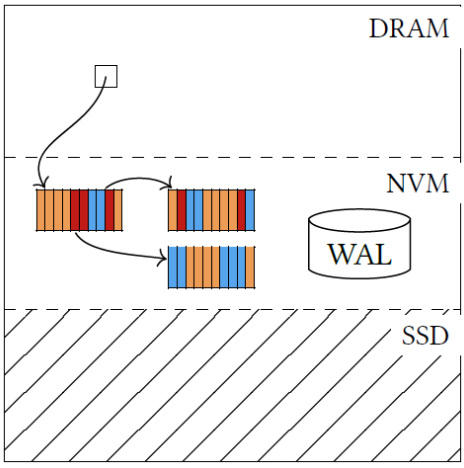


图 2.1 NVM 用作主存的架构图[12]

但是，该优化设计结构也有相应的缺点：1）由于 NVM 与 DRAM 相比具有更高的读写时延，其中 NVM 的写操作时延至少是 DRAM 的 5 倍以上，因此数据库系统想要实现较高的事务吞吐量就会变得更加困难；2）在没有 DRAM 缓冲区的情况下直接在 NVM 上工作会加速消耗 NVM 的写操作寿命，从而导致硬件系统故障；3）由于传统 CPU 和数据库系统均是针对 DRAM 进行设计的，因此直接在 NVM 上运行的数据库系统会很难编程。由于无法防止数据逐出，并且任何细微的修改都可能会被永久性地写入 NVM 中，因此就地写入 NVM 的操作都必须使对象数据处于正确状态，这对于数据库系统开发是非常困难的。

2.2 NVM 用作外部存储器

考虑到直接使用 NVM 作为内存的系统的缺点，依然使用 DRAM 作为内存似乎更具有研究意义。考虑到 NVM 相比 SSD 等外部存储设备具有更高的读写速度，因此研究人员尝试将 NVM 替代 SSD 用作 DRAM 的下层存储结构。

在该架构的缓冲管理器中，所有页面都存储在容量较大的 NVM 中，而容量较小的 DRAM 充当 NVM 的缓存。由于 NVM 相比 DRAM 拥有更高的读写时延和更短的写操作寿命，因此各项事务操作仅发生在 DRAM 中，并使用 fix()/unfix()函数将其在 DRAM 中锁定／解锁。

Kimura H .提出的 FOEDUS[19]不需要像传统的缓冲管理器一样存储页面标识符，而是存储两个指针：一个标识该页面在 NVM 中的副本；另一个标识其在 DRAM 中的页面位置（如果不为空）。当在 DRAM 中找不到页面时，该页面将从 NVM 中被加载到缓冲池中。FOEDUS 使用异步过程来组织 WAL 日志并将它们合并到 NVM 中以实现页面的持久性。该优化方案可以较好地利用 DRAM 的性能，同时利用 NVM 的持久性实现了更高效的脏页面回写[19]。

如图 2.2 所示，该系统不会直接将页面回写到 NVM 中，而是通过日志间接回写脏页面。在该系统中，NVM 主要用于存储持久性数据和冷数据，因此页面热度的判断会对其有直接重大的影响，然而当前页面热度判断和预测算法的精确度并不能满足 NVM 用作外部存储器的需求。因此 NVM 直接替代 Disk 或 SSD 的架构对于整体系统性能的提升效果不明显。

同时，直接使用 NVM 代替传统磁盘等外部存储器的系统架构使得 NVM 的可字节寻址能力并没有被体现出来，因此 NVM 的存储特性和潜力没有被充分发挥。同时 NVM 的容量也没有 SSD 等传统外部设备大，且非易失性存储存在严重的读写不平衡问题，需要研究人员设计一款写操作感知的替换算法，这会加重系统的负担，因此该系统的性能表

现并不优异。

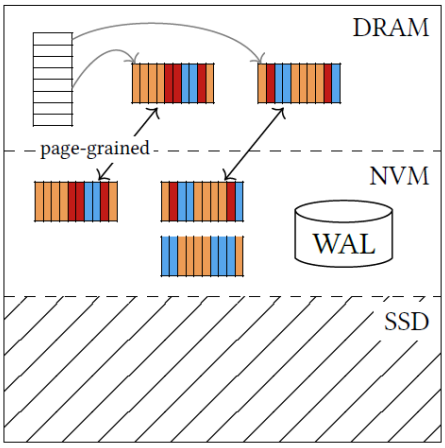


图 2.2 NVM 用作外部存储器的架构图[12]

2.3 基于 NVM 的混合内存

鉴于以上两种结构各自的缺点，研究人员提出了使用 DRAM 和 NVM 两种存储器件的混合内存作为 SSD 等持久性存储器件的上层器件。因此该设计的重点在于如何设计混合存储架构使得 DRAM 和 NVM 各自的优点能被更好地利用。以下是两种现存的混合内存设计结构，本节第一部分将详细介绍平行架构的设计特点和存在的问题，第二部分将会介绍垂直架构的特点，这是后续调研和研究的架构基础。

2.3.1 NVRAM 的平行结构

如图 2.3 所示，Lee 等人提出平行架构：传统内存 DRAM 和新型非易失性存储 NVM 是内存系统中独立的两块空间，二者互不干扰但数据可互相迁移。从上层 CPU 中被逐出的数据以及从下层存储器件缓冲的数据进入内存后，内存系统需要根据数据特点对其进行分配，此时将通过一定的管理策略，选择在 DRAM 或 NVM 空间中进行分配。其管理策略是：通过准确的热度预测，将未来更容易发生更新操作的数据分配到 DRAM 中，将短时间内发生更新操作可能性较小的数据分配到 NVM 中[11]。此后研究人员在页面预测方面进行了更进一步的探索，通过设置概率参数，量化页面可能被再次访问可能性，避免过多的异构内存系统内部的数据迁移带来的时延。同时平行架构可以避免重复数据的出现。

目前平行架构的研究中，页面迁移算法的考量因素主要是页面的局部性特征，而页面一段时间内的局部性特征并不能代表未来的访存趋势因此不具有代表性且页面的局部性特征会随着访存活动的发展而动态地发生变化。

其次，平行架构性能表现严重依赖于异构内存的管理分配策略。然而页面的访问特征是多种多样的且会随着系统运行进行动态变化，平行架构难以区分页面特性造成性能的浪费。同时，额外的内存元数据记录与管理，会对 CPU 造成过重的负担，从而影响整个系统的性能。

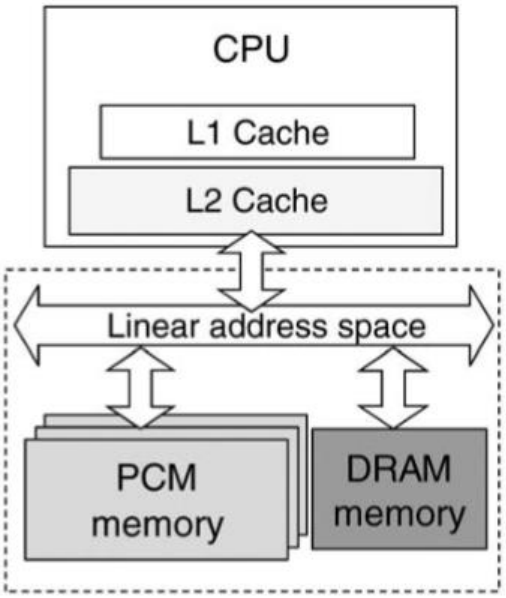


图 2.3 混合内存水平结构示意图

2.3.2 DRAM-NVM 的垂直结构

如图 2.4 所示，异构内存垂直架构中，作为缓冲区的传统内存 DRAM 和作为主存的新型非易失性存储 NVM 共同实现在一个内存条中。它们对于上层处理器和 Cache 是透明的，操作系统对内存数据的具体存储层次并不关心，且所有的访存过程均需要经过 DRAM。

垂直结构本质是将 DRAM 用作 NVM 的缓冲区，而 NVM 用作 SSD 等外部设备的缓冲设备，从而形成一个多层的结构。在垂直架构中，处于 NVM 中的数据被访问时，系统将调取该数据进入 DRAM 缓存。垂直结构能够避免平行结构对于热度预测和页面特征分析的依赖性，因此本文将采用垂直设计。关于垂直设计的具体实现细节将在下一节进一步展开。

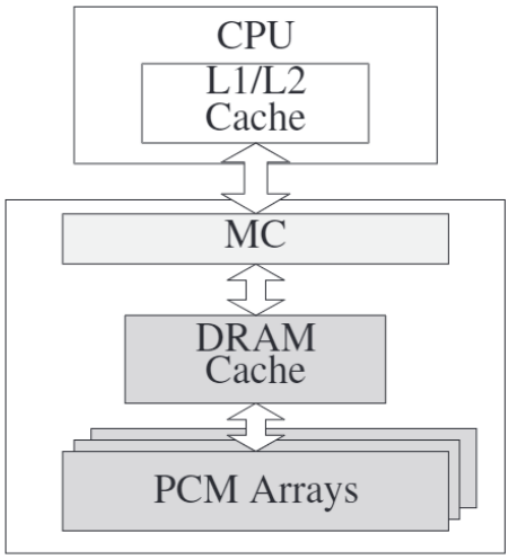


图 2.4 混合内存垂直结构示意图

2.4 面向内存数据库的混合内存

本节将详细介绍现代内存数据库系统中混合内存的应用。本节第一部分将阐述内存数据库系统的缓存过程，并结合层次架构的混合内存特点进行详细说明。本节第二部分将根据第一部分的特点介绍一种特殊的 NVM 数据结构，即缓存行粒度的 NVM 页面。本节将是后续调研和系统优化的结构基础。

2.4.1 DRAM-NVM 混合内存缓冲管理

新型存储架构研究的目的是充分利用 DRAM 和 NVM 各自的存储特点实现大规模数据的管理。为此，本文设计实现了一种 DRAM-NVM 混合内存缓冲管理器，该管理器采用混合内存的垂直结构，即 DRAM 是 NVM 的缓冲区。

CPU 进行访存操作时，若所要访问的缓存行数据位于 DRAM 中且不为已修改的脏数据则 CPU 可直接访问该数据；若索要访问的缓存行数据位于 DRAM 中，但该数据为脏数据则需要进行持久化操作使得数据具有一致性后才能进行访问。上述两种情况都不涉及 NVM 的相关操作，与传统的 DRAM 作为内存的存储系统类似。若 CPU 访存在 DRAM 中未命中，则应查找 NVM 中是否包含要访问的数据，该过程类似上述 DRAM 访问过程。在 NVM 中找到缓存行数据后系统将调取该数据进入 DRAM 中，此时数据的热度将会发生变化。若在提取过程中 DRAM 容量已满则需要将 DRAM 中短时间内不会访问的

冷的缓存行逐出 DRAM 到下层的 NVM 中。类似的缓存过程还发生在 NVM 和 SSD 等外部存储设备中，二者的区别在于 DRAM 和 NVM 之间的数据交换粒度是缓存行而 NVM 同 SSD 之间的数据交换粒度是页面，这样做的目的是减少页面页面置换操作对于系统性能的影响。

该管理器在 DRAM 和 NVM 之间以高速缓存行为粒度进行数据交换，从而通过利用 NVM 的可字节寻址能力来优化系统的带宽利用率。该管理器在 NVM 和 SSDs 之间以页面为粒度进行数据交换，这样可以延长持久性设备的写操作寿命，也可以降低系统时延。如图 2.5 所示，较热的高速缓存行将会被缓冲进速度更快的 DRAM 中，而较冷的缓存行将会因 DRAM 容量已满而被驱逐到容量更大的 NVM 中。

随着应用程序数据量的增加以及对实时性能的要求，传统的磁盘数据库已经无法满足应用需求，因此软件开发人员尝试引入 Key-Value 内存数据库，并将其作为传统外存数据库的缓存。因此本文旨在设计一种面向 Key-Value 内存数据库的新型存储策略。

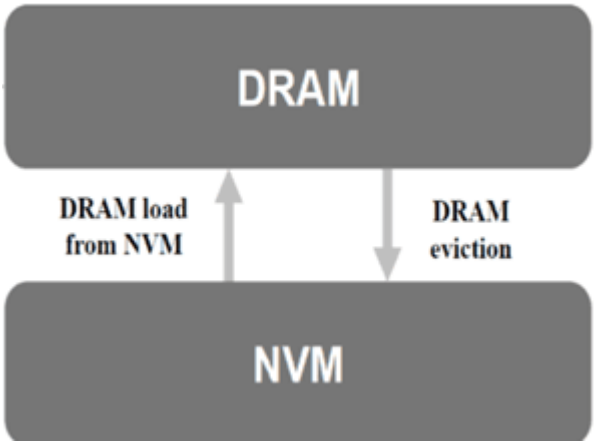


图 2.5 垂直结构混合内存的结构、数据粒度及操作

2.4.2 缓存行粒度的 NVM 页面

由于 NVM 与 DRAM 进行数据交换的粒度是缓存行而其与 SSD 数据交换的粒度仍然保持页面，因此本文实现了一种以缓存行为粒度的页面结构。上层的 DRAM 可以通过字节寻址获取 NVM 中某一个缓存行。

本文中的 NVM 缓冲管理器仅最初在 DRAM 中分配一个页面的存储空间，而没有从 NVM 中完全将该页面缓冲到 DRAM 中。当事务请求某个内存区域时，缓冲管理器先检查该缓存行是否已经被加载到了 DRAM 中，若 DRAM 中并不存在该缓存行，则缓冲管

理器会检索 NVM 页面中相应的缓存行。

如图 2.6 所示，每一个缓存行粒度的页面都拥有一个页头，页头各部分的类别和作用如下：

1. **指向 NVM 中相应页面的指针**：当 DRAM 中不存在某个缓存行时可以通过该指针将 NVM 中相应的缓存行调入 DRAM 中；
2. **页面标识符 pId**：用来区分 DRAM 中的各个页面；
3. **Resident 标志**：用来标记已经缓存进 DRAM 中的缓存行，初始状态下每一位都是 0，每当一个新的缓存行进入 DRAM 中，对应位置的二进制数值从 0 变为 1。处理器通过硬件检查 resident 确定该缓存行是否位于 DRAM 中以提升查询效率；
4. **Dirty 标志**：用来标记 DRAM 中被修改的脏缓存行，初始状态下每一位都是 0，每当一个缓存行发生修改变为脏数据并且之后需要写回到持久性设备中，其对应位置的二进制数值从 0 变为 1；
5. **r 标志**：表示该页面是否被全部缓存进 DRAM 中，若是则为 1 否则保持 0；
6. **d 标志**：表示该页面是否被完全修改，若是则为 1 否则保持 0。

在图 2.6 的例子中，该页面第一个、第三个和最后一个缓存行被加载到 DRAM 中，且第三个缓存行被修改。但是该页面并没有完全进入 DRAM 中，且页面没有完全进行修改，因此 r 和 d 都保持状态 0 不变。

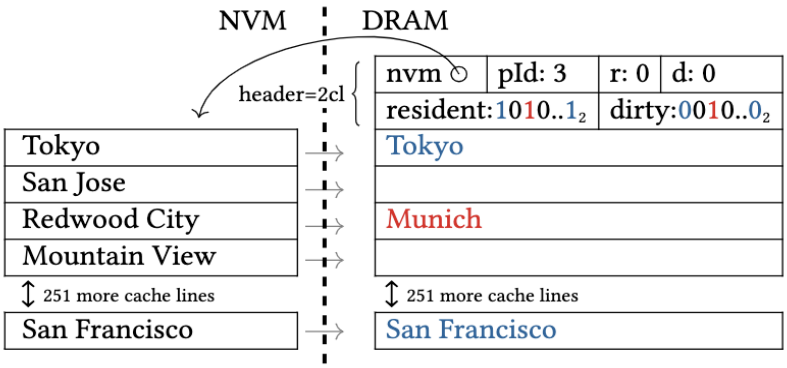


图 2.6 缓存行粒度的页面结构[12]

第三章 热度感知的性能优化方法

本章第一节将介绍热度感知的性能优化方法研究的动机，即现存的 DRAM-NVM-SSD 三层存储架构的结构和特点并对该结构进行性能统计和分析，分析现有方法的性能缺陷。本章第二节将会对性能问题出现的原因进行猜测，针对该猜测进行详细的系统调研，分析调研的结果。第三节将会根据调研结果设计具体的性能优化方案，第三节将是本章探讨的重点。最后第四节将针对该优化方法分析复杂度和开销。

3.1 方法动机及分析

本节主要研究传统 DRAM-NVM-SSDs 三层架构下的命中率并且研究未命中的原因，这部分是本次优化设计的研究动机和后续优化算法的基础。本节第一部分将介绍传统三层结构的设计细节以及各层数据的粒度和组织结构。本节第二部分将在 Redis 内存数据库上对 6 个不同的 YCSB 工作负载进行了命中率测试的相关实验。针对实验结果进行分析，猜测导致性能下降的原因，并根据该原因进行详细调研。

3.1.1 传统三层架构概述

现有的工作利用 NVM 作为内存数据库体系结构中 SSD 的上层存储层，特别是对于请求范围比较大的工作负载，能极大提高性能并降低成本。如图 3.1 所示，NVM 作为 DRAM 和 SSD 的中间层，与 SSD，DRAM 一起共同组成了一个三层存储体系结构

DRAM 中保存着从最后一级 CPU 高速缓存中驱逐出来最热的缓存行，而 NVM 作为 SSD 的上层缓冲层，可以有效避免 SSD 比较长的延迟。这种将 NVM 引入的技术，由于其具有低延迟和高吞吐量访问的特点，有效弥补了 DRAM 与 SSD 之间的性能差距。在存储系统运行过程中，当 DRAM 中的请求未命中时，将从 NVM 中检索数据。一旦 NVM 请求未命中时，请求必须访问延迟比较长的 SSD，且一旦找到所对应请求的数据，就会将其直接加载到 DRAM 中。同时，数据只会从 NVM 转移到 SSD 中，而不会从 SSD 加载到 NVM 中。

如图 3.2 所示，在 NVM 中使用了一种以缓存行为粒度的页面结构，用于适应 DRAM 和 SSDs 之间不同的粒度转换，一方面允许了 DRAM 选择性地访问缓存行数据，而不是访问整个页面；另一方面也保证了页面可以更加有效地存储于 SSDs 中。在传统的三层架构中，NVM 和 DRAM 中数据的组织结构均为 B+树结构，该结构由于由于能够自适应地平衡且查找和修改节点的时间复杂度较低，因此广泛应用于各类存储系统，例如 DRAM

和磁盘等。

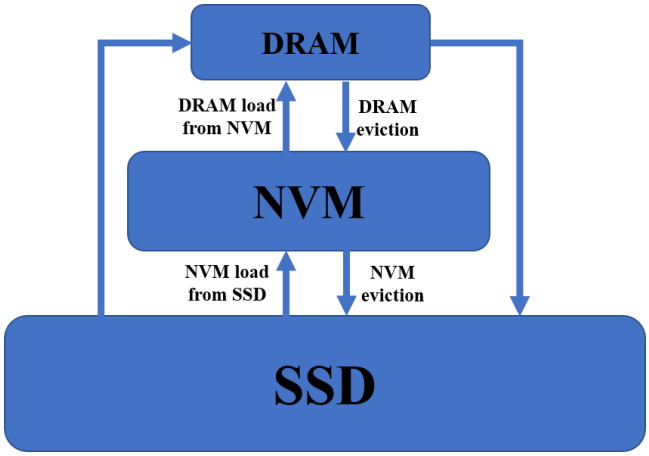


图 3.1 传统三层存储架构

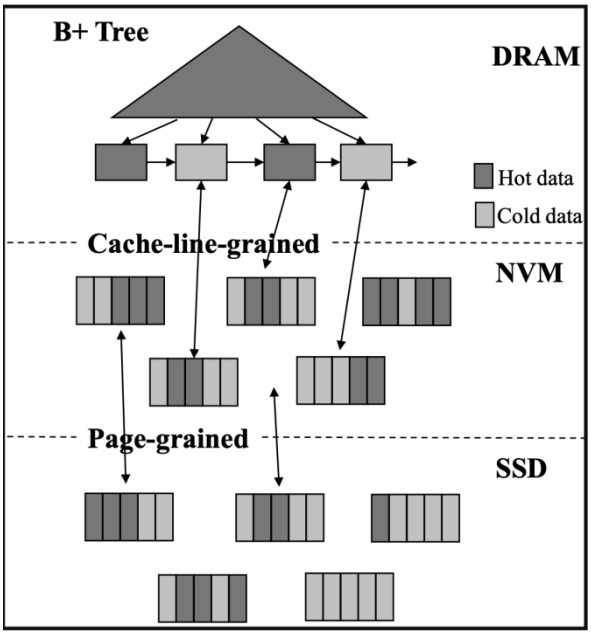
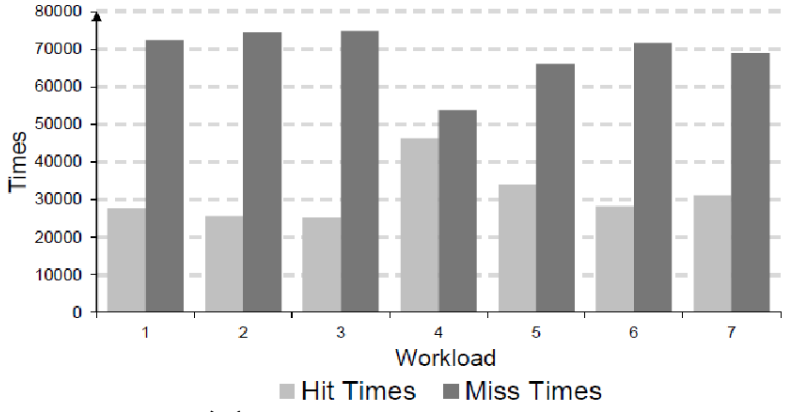


图 3.2 传统三层结构的数据组织和交换示意图

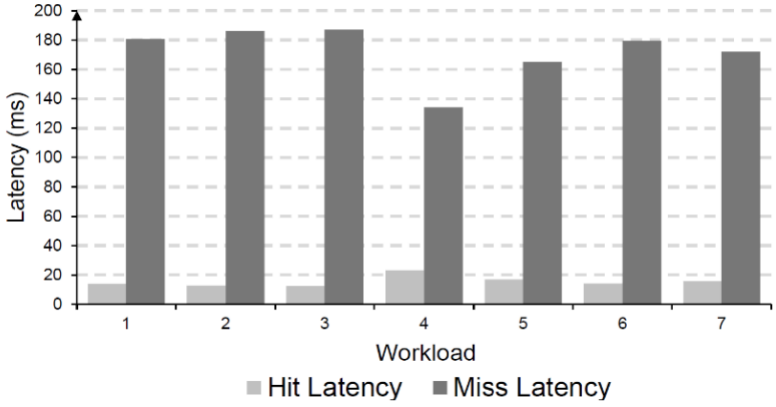
3.1.2 传统架构性能性能分析

由于大多数传统的替换算法都是为了保证 NVM 中存储较热的页面，而将短时间内不会访问的冷页面驱逐到 SSDs 等外部存储设备中，因此本实验采用了当下数据库系统中使用较多的 LRU 算法，该算法将最近访问过的且历史访问次数最多的数据保存在 DRAM 中，而将历史访问次数少的或是很久都没有再次访问的数据逐出 DRAM 到 NVM 中。如图 3.3 所示，实验收集了在 6 个不同工作负载下 NVM 的请求命中/未命中的次数，并统计

了未命中的请求次数对整体系统带来延迟和性能下降情况。



(a) NVM 中缓存行命中/未命中



(b) NVM 命中和未命中情况下的系统时延

图 3.3 YCSB 工作负载下 NVM 的命中/未命中的次数和延迟对比

图 3.3(a)的实验结果表明，全部的工作负载上命中的次数均少于未命中的次数。命中率最低的工作负载 3，其命中率仅为 25%左右；即便是命中率最高的工作负载 4，其命中率也仅为 45%左右，不足 50%。由此可以证明现存的三层存储架构的 NVM 命中率较低，大多数的访存操作都需要对 SSD 等外部设备进行访问，而这一过程不可避免地要进行系统 I/O 过程，因此会造成大量的系统时延最终导致系统性能的下降。

图 3.3(b)的实验结果表明，NVM 命中的系统时延明显低于未命中时整个系统的时延。参考(b)图结果，命中率最低的工作负载 3，未命中时延是命中时延的 18 倍左右。全部 6 个工作负载中未命中时延平均是命中时延的 16 倍左右。由此可以证明 NVM 命中率的提升对整体性能提升的重要性是巨大的。

NVM 中请求未命中数量的减少将极大提高整体系统性能。因此如何设计存储系统内

的数据组织结构和替换算法以减少 NVM 中未命中操作的数量成为系统急需解决的问题。

3.2 系统性能问题调研

本节首先对 NVM 命中率低的原因进行猜测，其次根据该猜测对传统三层存储架构进行详细的系统调研，分析原因的可能性。

3.2.1 NVM 命中率低的原因

基于 DRAM-NVM-SSDs 三层存储体系结构的设计，由于 DRAM 需要和上层的 CPU 进行数据交换，因此 DRAM 的数据粒度是缓存行（Cache Line），而 NVM 要和下层的 SSDs 等外部存储器件进行页面交换，为避免频繁读写造成的时延和写操作寿命减少因此 NVM 和 SSDs 的数据粒度都是页面。因此造成了系统中 DRAM 和 NVM 的数据访问粒度不一致。

系统默认状态下，在 NVM 中的页面组合缓存行采用的是随机选取的方式，即不考虑缓存行之间的逻辑关系仅随机实现 NVM 中页面的组合和替换等操作。因此可能会出现热度较大的某一缓存行随机和热度很冷的多个缓存行一起组成一个冷的页面。在 NVM 进行替换过程中，该页面可能会被判断为冷页面而逐出，但该热缓存行可能在不久的将来会倍 CPU 重新要求访问，此时整个页面又将会倍缓冲进上层存储结构中，如此反复进行系统 I/O 操作最终导致命中率的降低和系统时延的增加。

因此本文猜测真正导致性能下降的原因是 NVM 页面中各个缓存行之间热度差异巨大，也即 NVM 内缓存行热度分布不均。

3.2.2 系统调研

为了验证上述猜想，本文进一步统计了在一个 NVM 页面中缓存行数据的热值差异，结果如图 3.4 所示。

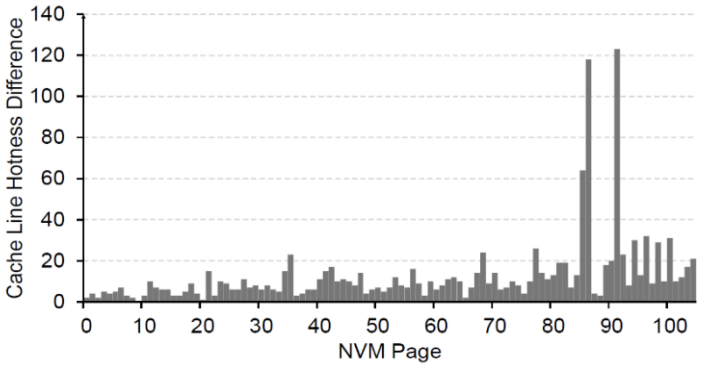


图 3.4 NVM 页面中的缓存行热度差异

图 3.4 展示的是工作负载 3 某一随机时刻 NVM 中页面内各缓存行之间热度差异的最大值。实验过程中热度采用历史访问次数替代。从上述实验结果可以看出，缓存行在 NVM 页面中热度分布很不均匀，尤其是 Pid 在 80~90 之间的页面，其热度差异最大可达到 120。这意味着一个 NVM 页面可能同时包含极热的缓存行和极冷的缓存行。这将会导致两种可能的结果：

1）该页面被判断为冷页面，但是极热的缓存行会被经常访问，导致系统 I/O 频繁发生，最终导致时延增加；

2）该页面被判断为热页面，但是极冷的缓存行可能在未来永远不会被访问，但是它会在 NVM 中占据一定容量，间接导致系统 I/O 次数的增长，最终导致时延增加。

以上两种情况将会引起 NVM 请求未命中，导致命中率的降低。当 NVM 中的驱逐操作发生时，热的 NVM 页面中的冷缓存行占用着 NVM 内部的存储空间，但这些缓存行可能在未来一段时间内不会被处理器访问。同时，被驱逐的冷的 NVM 页面中可能也包含着一些较热的缓存行。热的缓存行随着冷的 NVM 页面被驱逐出混合内存后，很有可能处理器会重新访问该缓存行，此时系统将重新将该页面缓冲进混合内存中，该过程将会导致资源的浪费和性能的降低。因此如何调整 NVM 中的缓存行，使其大致按照热度分布是本文研究的优化方向。

深入调研后发现，NVM 中未能实现根据缓存行热度进行页面组合的原因是在传统三层架构中 NVM 内页面组织是根据 B+树中各个叶节点分布进行的，而 B+树各个叶节点的排序原则是根据 Key-Value 对中 Key 值的大小进行排序。因此想要解决热度分布不均的问题，需要设计出一种新型的 NVM 中缓存行的索引结构。

3.3 方法整体结构

如图 3.1 所示，本文优化方法，即一种名为“影子 B+树的 NVM 索引结构”作用在 NVM 一层中。如图 3.5 所示，顶部是一个按键索引的 B+树，它在传统的三层架构中指向 NVM 中的数据。该树的非叶子节点是一个由 n 个键和 n+1 个子指针组成的索引结构。叶子节点包含一个统一的键，一个 PID 和一个参考 NVM 中数据地址的偏移量。

如图 3.5 底部所示，其中影子指针指向影子树的对应节点，next 指针指向它的兄弟节点，访问计数器（access counter）用来标识该缓存行的热度，每当请求该键时，其中的数值就会相应的增加。

在影子 B+树种，叶节点包含以下内容：

1）父节点；

2）指向子树的节点；

3）访问计数器，用于评估节点的热度；

4）逻辑指针，指向原始 B+树中对应的叶节点。影子树用于管理存储于 NVM 中的缓存行数据，并帮助混合存储系统进行数据替换和驱逐操作。

为方便之后根据影子 B+树中缓存行热度进行页面的重组，影子 B+树上的节点按照访问计数器内的值（热度）进行排序，即从左至右热度依次递增。这样，缓存行就可以很容易地按照热度划分为冷、凉、热三种不同类型。因此，根据原始 B+树中的数据构建出的一个影子树，将其称为逻辑树，且其中的数据称为逻辑节点。每个逻辑节点中的缓存行都被加载到影子树中，并且在第一次调整后根据热度进行排序。

相较于传统的以页面为粒度的 B+树，叶子节点数据字段的大小与一个缓存行数据的大小相同。因此 DRAM 和 NVM 之间的数据更新可以采用缓存行粒度，这样可以充分发挥 NVM 可字节寻址的特性。

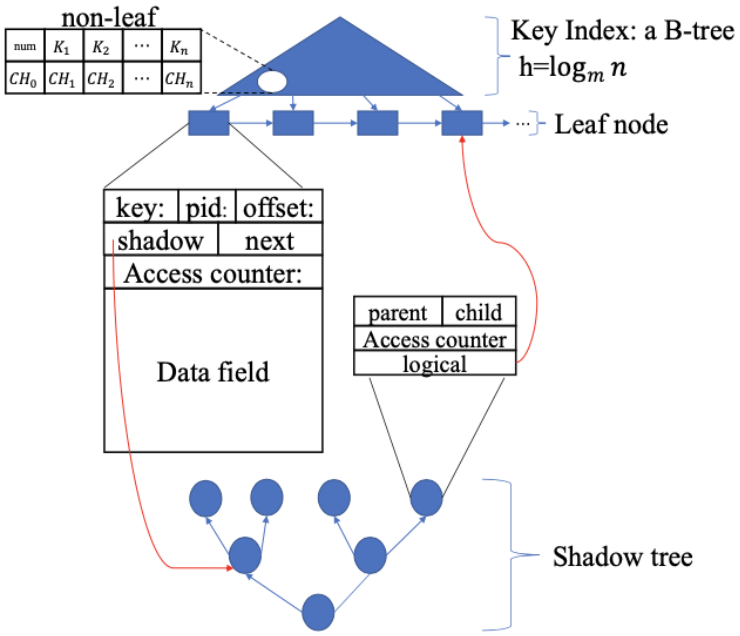


图 3.5 影子 B+树的组织结构

3.4 热度感知的动态调整策略

影子 B+树可按照热度对节点进行调整。为了减少开销避免频繁的调整节点导致时延，系统将在一定数量的请求后进行调整。首先在调整前，将对影子 B+树节点的访问计数器将进行管理。当来自 DRAM 的请求到来时，所请求的缓存行访问计数器将会增加 1 个单位。所以，尽管上一次的调整已经根据热度对节点进行排序，但在一组请求之后访问计数器的值会发生较大变化，因此需要进行影子 B+树数据调整，具体调整流程图如图 3.6 所示。

如图 3.6 所示，该方法将会对每个节点的访问计数器进行检查，不断找出访问计数器中大于预先设置的热阈值的叶子节点（热数据）。而 B+树影子方法的核心操作在于将这些节点移动到与其具有最近似访问计数器值的叶子节点附近。一旦新的位置确定了，它的逻辑节点的影子指针也相应发生改变。如果调整的空间不够，将会选择一个最冷的缓存行并驱逐其对应的页，具体驱逐操作细节将在下一节中进行详细阐述。

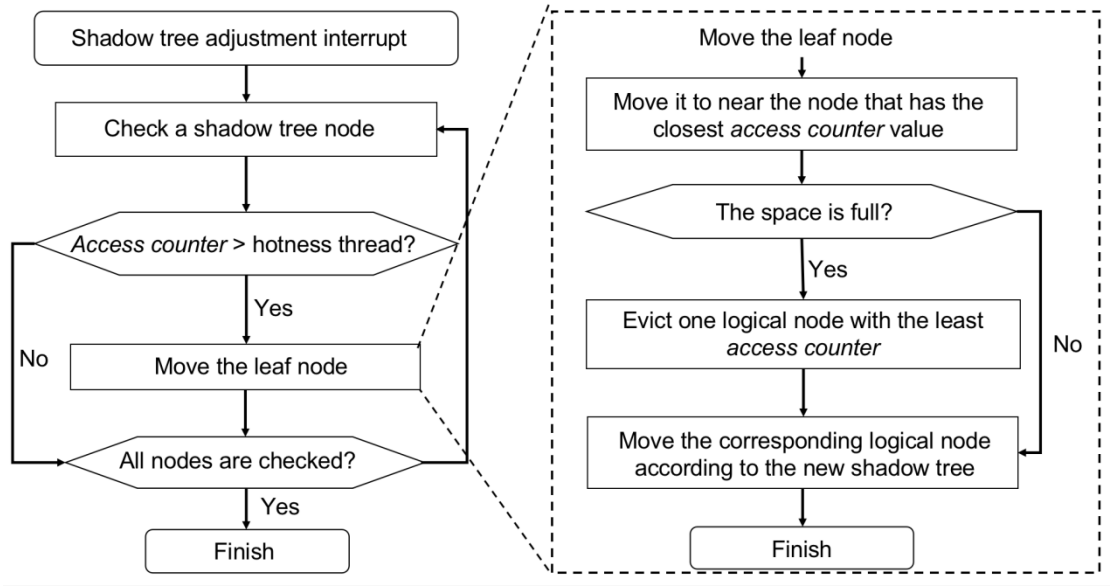


图 3.6 热度感知的动态调整策略流程图

3.5 影子 B+树的基本操作

3.5.1 节点的插入

该操作发生在 DRAM 将某一缓存行数据剔出到 NVM 中。当把从 DRAM 中驱逐的新的缓存行插入到原始树中时，影子 B ＋树也会相应生成一个新的叶子节点并插入影子 B ＋树中。换句话说，影子树中的索引数据与原始树中的索引数据相同。由于 DRAM 种剔出的数据短时间内再次访问的可能性较低，因此新的节点将会首先被插入到访问计数器中值最低的节点附近。这个新 B+树节点的逻辑指针将会指向对应的原始 B ＋树叶子节点，而这个原始树叶子节点的影子指针也将会指向新影子 B+树节点。影子 B+树节点插入的流程图如图 3.7 所示。

3.5.2 页驱逐及节点的删除

当 NVM 中的空间已满，将会有一个页被选择逐出。本文所提出的 B+树影子方法旨在驱逐那些包含最冷的缓存行数据的页面，所以将会选择在影子 B+树中访问计数器具有最小计数值的 64 个缓存行数据(假定缓存行的大小为 64B，而页大小为 4KB)并将其组成一个页驱逐到 SSD 中，同时在影子 B+树中直接删除对应的索引，从原始树中删除对应的

逻辑节点。

由于这些被驱逐的缓存行可能分布在不同的原始节点中，因此将留下的缓存行随机地重新组织到原始的逻辑页中，并将其插入到原始树中，影子 B+树的逻辑指针也将会指向相应新的原始树叶节点。该删除方法可以保证热的缓存行数据仍留在 DRAM 中，并且只驱逐那些冷的缓存行数据。这将会有效改善由于被驱逐页面中由于缓存行冷热分布不均所导致的 NVM 请求未命中的问题。驱逐过程流程图如图 3.8 所示。

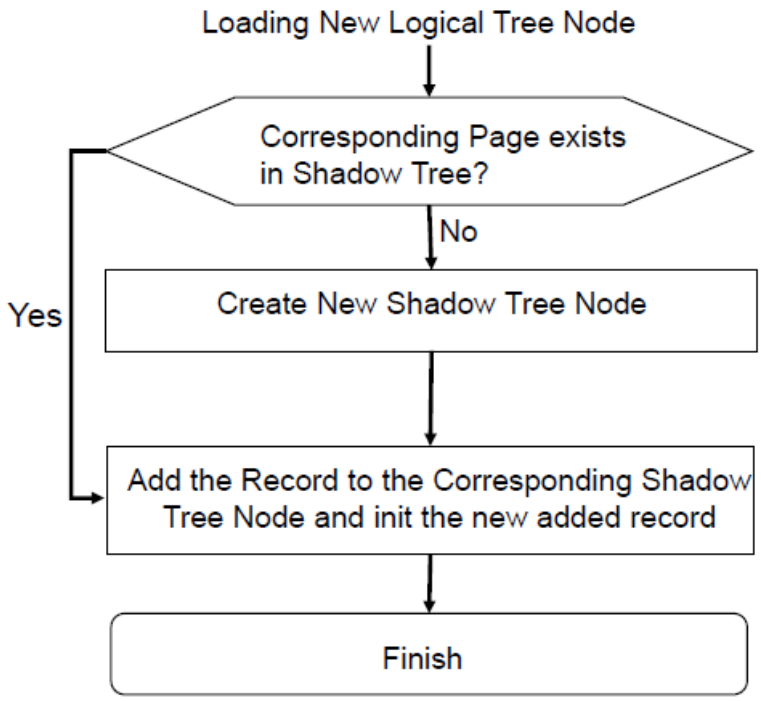


图 3.7 影子 B+树节点添加流程图

3.6 复杂度和开销分析

现在分析影子 B+树的时间复杂度及存储空间开销，将从三方面分别展开讨论其时间复杂度。

1）具有 n 个叶节点的 m 分支 B+树将会产生 $O(h)$ 的查询时间复杂度，其中 h 为 B+树的高度，等于 $\log mn$；

2)当驱逐发生时，由于树的调整，它需要搜索始终位于影子树一侧的最冷节点，因此，时间复杂度为 $\log mn$；

3）由于调整操作只在多次请求后进行调用，所以可以根据工作负载强度自适应地控制开销。

针对存储空间开销，影子 B+树占用 DRAM 的空间。假设有 128 个逻辑节点，每个逻辑

节点包含大小为 64 条大小为 64 字节的缓存行，故影子 B+树有 128*64 个节点，假设索引节点的大小为 8 字节，影子 B+树占用了 8*1024*64 位=8KB 的存储空间，这个开销对目前的计算机主存大小而言是可以忽略的。

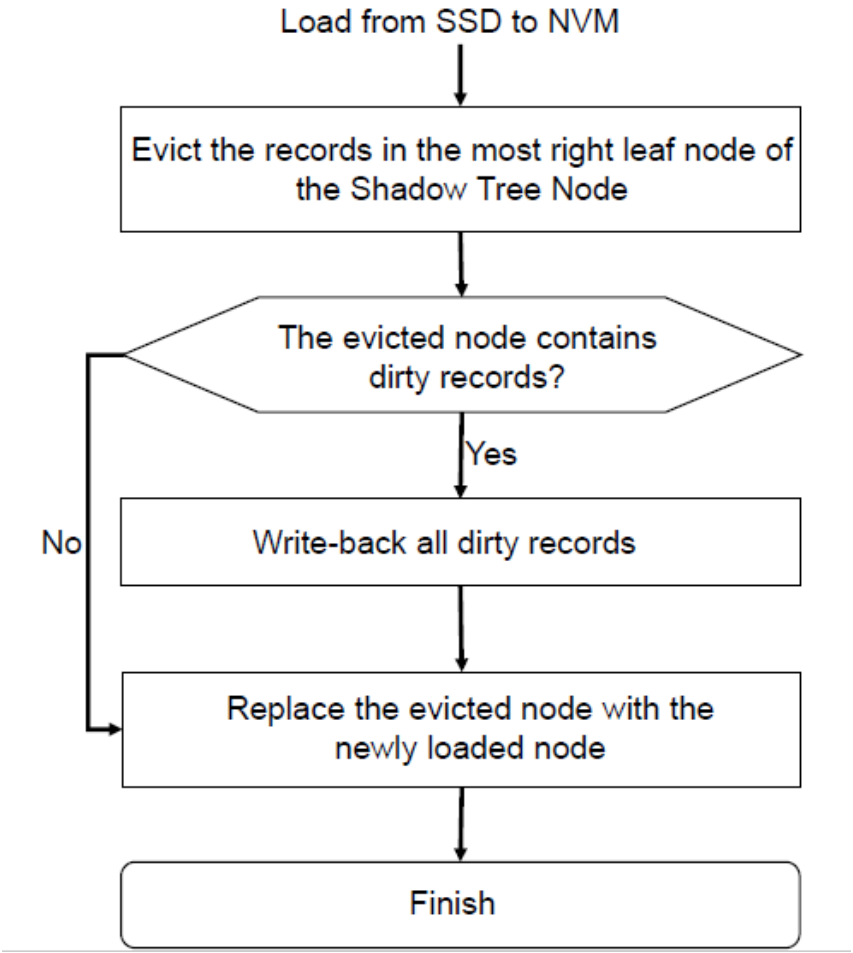


图 3.8 影子 B+树节点删除流程图

第 4 章 实验与结果分析

本章将评估 DRAM-NVM-SSDs 三层架构下的 B+树影子方法的有效性。首先，本文介绍了三层存储体系结构、数据集和对应实验平台的实验参数设置。其次，给出实验结果。最后，针对 NVM 的命中率和整体系统性能进行了分析并展开了相关讨论。

4.1 实验环境及参数设置

4.1.1 模拟平台及参数

本次实验是在配置有 Inter Core i7,12G DRAM 和 128G SSD 的 Ubuntu 18.04 服务器上进行的。由于 NVM 设备目前仍处于实验室研究阶段尚未投入商用，且 SSD 也不面向用户开放，因此很难用真实的 DRAM、NVM 和 SSDs 器件构建出三层存储体系架构。

鉴于此，本实验通过使用 mmap()分配一块内存空间来模拟一个 NVM 设备。NVM 中的所有内存区域都附加在进程地址空间的相同起始地址上，且使用 C++编程语言实现了一个影子 B+树。影子 B+树的结构采用二分查找的方式，并将相应的键值和有效负载存储在单独的数组中（按键大小排序），在启动 YCSB 工作负载前，首先，通过运行这些工作负载，跟踪 Redis（Remote Dictionary Server）数据库的索引结构获得对应所需的文件，进而利用其构建 B+影子树。实验的目的在于，对比当驱逐发生时分别采用的 B+树影子方法和传统 LRU 算法该三层存储体系结构的性能水平。具体实验过程中，设备存储大小和延迟的模拟参数如表 2 所示。

表 4.1 模拟平台参数设置

	DRAM	NVM	SSD	页面大小	缓存行大小
容量	128MB	1G	128G	4KB	64B
时延	N/A	50ns	$25\mu s$	N/A	N/A

4.1.2 数据集及参数

本次实验采用 Redis 数据库和 YCSB-Benchmark 来评估所提出 B+树影子方法的有效性。

Redis 是一种应用分布式，具有可选择持久性的内存 Key-Value 数据库，它是一种 3-

Clause BSD 许可协议开发的开源软件。而 YCSB 是一种开源的规范和编程的套件，用于评估计算机程序的检索和维护能力，它通常用于比较 NoSQL 数据库管理系统的相对性能。YCSB 提供了不同的工作负载进行评估。其中主要选择了 6 个具有不同读/写比率的工作负载，它们是预定义示例工作负载的一般化形式，具体参数如表三所示。同时，将所有的工作负载设置为 10,000 条记录和 100,000 个操作，且使用 Zipf 分布(z=1)键来建模真实世界的数据倾斜。

整个模拟的过程划分为两个阶段：1）YCSB 的加载阶段；2）运行阶段。加载阶段主要用来建立运行阶段所需要的数据，YCSB 将会按照设置的参数建立测试的数据。运行阶段将强制事务以工作负载的形式度量数据库，所使用的影子 B+树方法的索引结构是通过跟踪 Redis 运行阶段的源代码所构建的，运行阶段所获得的路径文件将被用来驱动模拟器进行相应实验。

表 4.2 YCSB-Benchmark 的工作负载设置参数

工作负载	A	B	C
比例	读操作/更新 50%/50%	读操作/更新 95%/5%	读操作/更新 100%/0
工作负载	D	E	F
比例	读操作/插入 95%/5%	扫描读/插入 95%/5%	读操作/读-写 50%/50%

4.2 实验结果及分析

图 4.1 是工作负载 A 下的 NVM 页面热度差异，对比使用 B+树影子方法前的 NVM 页面热度差异（图 3.4），采用影子 B+树方法后，NVM 中页面热度差异相比较原始页面热度差异有较明显的改进。B+树影子方法下的 NVM 页内缓存行数据热度最大差异仅为 37，而未采用 B+树影子方法的结构其 NVM 页面热度差异高达 120。相较于使用影子 B+树前的平均热度差异，NVM 各页面的平均热度差异仅在 17 左右且热度差异的方差较小。

此外，沿 x 轴方向，NVM 页内相邻缓存行数据的热度差异左侧缓慢增大，而右侧急剧增大。这是因为 B+树影子方法根据页面中的缓存行热度将其进行了重组。同时可以看到 x 轴左侧的页面差异较小而右侧的页面差异较大，对应 NVM 的影子树种左子树的热度较高且分布较平均，而右子树的热度差异较大且包含冷的缓存行的概率较高。因此在后续 NVM 的逐出操作中，系统可以优先选择右子树中热度差异大，包含冷缓存行较多的页

面进行逐出操作，这样有利于减少系统 I/O 的次数进而降低系统时延。

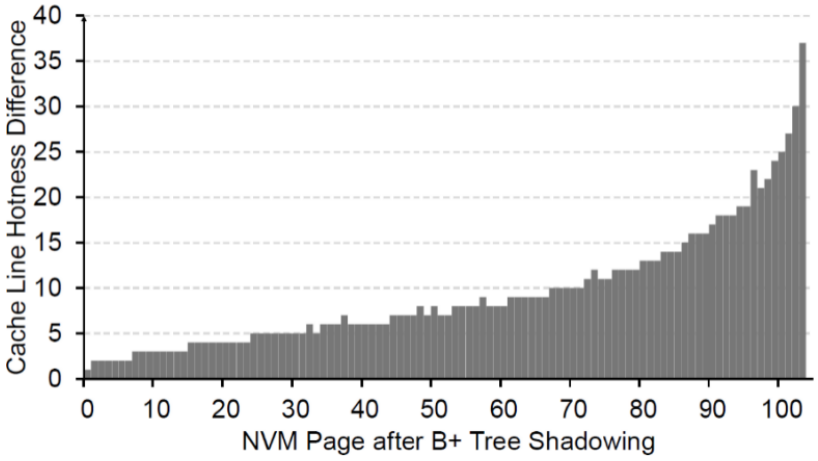


图 4.1 使用 B+树影子方法后工作负载 A 下 NVM 页内热度差异

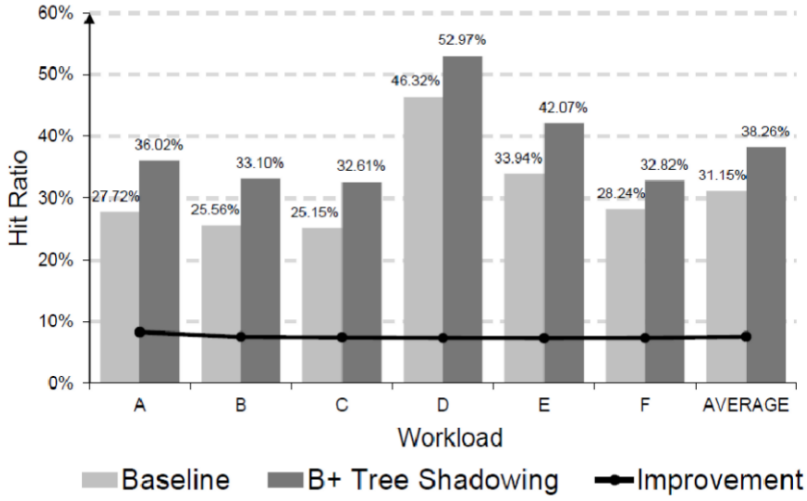
实验结果如图 4.2(a)所示，其中我们将影子 B +树方法与传统三层架构下使用 LRU 算法和 B+树索引的基础方法进行了比较。图 4.2(a)中，横轴表示要执行的工作负载类型，纵轴表示命中/未命中的百分比。黑线表示使用 B+树影子算法后相比较 LRU 算法获得的性能提升。

如图 4.2(a)所示，无论在哪种工作负载下，B+树影子方法的性能始终优于传统 B+树索引方法，且它们之间的性能差异稳定在 10％左右。在 6 中 YCSB 工作负载下，命中率最低的工作负载是第三组，B+树方法的命中率为 25.15%，而影子 B+树方法的命中率为 32.61%，命中率提升了 29.66%。命中率最高的工作负载时第四组，B+树方法的命中率为 46.32%，而影子 B+树方法的命中率为 52.97%，命中率提升了 0.14%。命中率的平均值，B+树方法为 31.15%，而影子 B+树方法为 38.26%，命中率提升了 22.83%。从此结果中，可以得出结论：使用影子 B +树方法的存储系统可以有效地提高 NVM 命中率，减少系统 I/O 的次数，进而间接减少系统时延，提高系统性能。

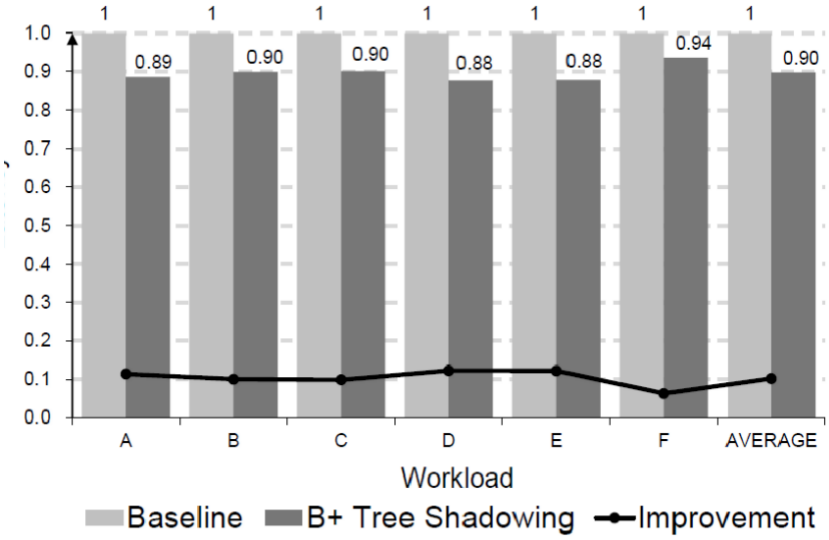
随后，本文在六个不同的工作负载条件下，在系统时延方面对传统 B+树方法和影子 B +树方法进行了对比实验，以反映系统最终性能的优化结果，如图 4.2(b)所示。图 4.2(b)中，横轴表示不同的工作负载，纵轴表示两种方法下的系统时延。

实验结果证明，采用 B+树影子方法后，系统时延明显低于传统 LRU 算法的系统时延。此外，黑线表示两种方法之间的性能差异。由此可见，B+树影子方法在各个工作负载运行条件下都有明显提升，且二者之间的差异值稳定在 10％左右。由 4.2(b)可看出，使用传统 B+树方法的 6 种 YCSB 工作负载的时延相差不大可忽略，而使用影子 B+树方法的 6 种工作负载中，第六组性能提升最少，但仍有 6%的提升，而性能提升最多的第五组

性能提升了 12%。因此，基于以上实验结果，本文可以证明影子 B +树方法能够有效地减少系统时延，进而提升系统整体的性能。



(a) 命中/未命中的比例



(b) 系统/时延性能

图 4.2 B+树影子方法与基准方法的性能结果对比

第 5 章　总结与展望

5.1 总结

本文阐述了当前计算机存储系统存在的性能问题，并着重阐述了近年来引起广泛关注的新型非易失性存储器件 NVM 的存储特性、分类等。本文研究了基于 NVM 的新型存储架构的发展、各自的工作原理、优点和存在的问题并最终选择 DRAM-NVM-SSDs 三层存储架构作为研究的重点。

在三层存储架构的基础上，本文研究了近年来具有良好研究前景的开源内存 Key-Value 数据库 Redis，研究其在传统三层存储架构下的命中率及造成命中率低的原因，即 NVM 种页内缓存行热度分布不均。

最终在基于上述原因，本文设计并实现了一种 B+树影子方法。文中详细介绍了该影子 B+树的组织结构、基本工作流程以及复杂度。在 Redis 和 YCSB-Benchmark 下对该方法进行实验，结果证明此方法对于提升命中率、降低系统实验有帮助。

5.2 展望和未来工作

本文虽然实现了主要的研究内容，但仍需要进一步研究和优化，具体总结如下：

1. 影子 B+树方法进行数据自适应地调整的过程中，用于判断数据冷热的阈值是根据经验人为设定的，可能会对整个系统性能造成影响。未来可以考虑使用更加智能的自适应算法动态获得阈值。

2. 实验所用的工作负载过少无法模拟真实情境下数据库系统的数据处理规模。

参考文献

[1] 张益铭. NVM 与 DRAM 混合存储优化策略的研究与实现[D]. 电子科技大学.

[2] 余松平. 面向大数据的混合内存管理关键技术研究[D]. 国防科学技术大学.

[3] Awad A , Blagodurov S , Solihin Y . Write-Aware Management of NVM-based Memory Extensions[C]// International Conference on Supercomputing. ACM, 2016:1-12.

[4] 秦杰杰. Redis 数据库在非易失性内存上的交换技术的研究与实现[D]. 重庆大学.

[5] Chen C , Zhang C Y . Data-intensive applications, challenges, techniques and technologies: A survey on Big Data[J]. Information Sciences: An International Journal, 2014.

[6] Deep learning applications and challenges in big data analytics[J]. Journal of Big Data, 2015, 2(1):1.

[7] Arulraj J , Pavlo A , Teja Malladi K . Multi-Tier Buffer Management and Storage System Design for Non-Volatile Memory[J]. arXiv, 2019.

[8] Zhe, Wang, Shuchang, et al. WADE: Writeback-aware dynamic cache management for NVM-based main memory system[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2013, 10(4):1-21.

[9] J Lindström, Das D , Mathiasen T , et al. NVM aware MariaDB database system[C]// Non-volatile Memory System & Applications Symposium. IEEE, 2015.

[10] Kimura H . FOEDUS: OLTP engine for a thousand cores and NVRAM[J]. ACM, 2015.

[11] Lee D H , Yoon S K , Kim C G , et al. A Space Effective DRAM Adapter for PRAM-Based Main Memory System[C]// International Conference on It Convergence & Security. IEEE, 2013.

[12] Renen A V , Leis V , Kemper A , et al. Managing Non-Volatile Memory in Database Systems[C]// the 2018 International Conference. 2018.

[13] J. Arulraj, A. Pavlo, and S. Dulloor. Let’s talk about storage & recovery methods for non-volatile memory database systems. In SIGMOD, pages 707–722, 2015.

[14] Kimura H . FOEDUS: OLTP engine for a thousand cores and NVRAM[J]. ACM, 2015.

[15] Lee S , Bahn H , Noh S H . CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures[J]. IEEE Transactions on Computers, 2014, 63(9):2187-2200.

[16] Ju G , Li Y , Xu Y , et al. Stochastic Modeling of Hybrid Cache Systems[J]. IEEE, 2016.

[17]Dhiman, Gaurav, Raid Ayoub, and Tajana Rosing. "PDRAM: A hybrid PRAM and DRAM main memory system." 2009 46th ACM/IEEE Design Automation Conference. IEEE, 2009.

[18]曹猗宣. 内存数据库的研究及其应用[D].北京邮电大学,2011.

[19]Kimura H . FOEDUS: OLTP engine for a thousand cores and NVRAM[J]. ACM, 2015.

[20]Labs, R.: Redis. https://redis.io

致　谢

我的本科生涯即将结束，在此我要感谢本科期间每一位老师。在各位老师的辛勤教育下，我才能从高中毕业后懵懂的孩子逐渐成长为能够独立解决生活中问题的人。其中我要特别感谢杜亚娟老师，是她带我步入研究的轨道。杜老师在研究中给予我专业的指导和帮助，同时她也愿意跟我分享一些个人经历。通过老师的分享和悉心指导我感到鼓舞，也意识到自己的潜能还没有开发出来。

感谢我的同学、朋友以及学长学姐，在他们的帮助下，我对计算机科学有了更深的认识，也使我成为一个更加勇敢的人。在合作的过程中，我也意识到自己的不足，这使我能够不断努力汲取更多的专业知识。

最后感谢我的父母，是他们给我营造了一个温馨平稳的学习生活环境，让我能追求更高的人生理想。