分类号_____ 学号 M202073158

学校代码 10487 密级_____

华中科技大学

硕士学位论文

（学术型☑ 专业型☐）

基于 ZNS SSD 的云块存储系统写放大优化

学位申请人：罗斯琦

学 科 专 业：计算机系统结构

指 导 教 师：王桦 教授

答 辩 日 期：2023 年 05 月 18 日

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Master Degree in Engineering

ZNS SSD based Write Amplification Optimization for Cloud Block Storage Systems

Candidate : LUO Si Qi

Major : Computer Architecture

Supervisor : Prof. WANG Hua

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

May, 2023

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：罗其倚

日期：2023 年 5 月 22 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留、使用学位论文的规定，即：学校有权保留并向国家有关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。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，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。

本论文属于 保　密□，在_____年解密后适用本授权书。

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：罗其倚　　指导教师签名：王木华

日期：2023 年 5 月 22 日　　日期：2023 年 5 月 22 日

摘 要

固态硬盘（Solid State Drives, SSDs）内部以日志结构的方式管理数据，在垃圾回收（Garbage Collection, GC）时需要重新写入有效数据，造成了写放大问题，损害 SSD 的性能和寿命，究其本质是传统 SSD 无法感知主机端数据的热度信息，很容易将冷热数据混合放置，从而频繁触发 GC 操作。ZNS（Zone Namespace）SSD 允许用户在主机端自定义数据放置策略，将冷热数据分区放置，为改善写放大问题提供了良好的契机。然而，面对云块存储系统海量且频繁变化的 I/O 负载，如何在 ZNS SSD 中合理地放置数据以减轻写放大是一个严峻挑战。

首先，通过对云块存储系统负载进行深入地分析，发现数据块写入频率和块寿命整体分布类似，均遵循 Zipf 定律。接着提出了适用于 ZNS SSD 的理论最优放置策略 ZB-OPT（ZNS-Based OPT），基于未来块寿命信息，按照指数型阈值区间进行块寿命分类，可以实现最小化写放大的效果，为真实场景中的写放大优化提供理论上限。然后，针对现有研究通过预测未来块寿命进行冷热数据分类的方案无法兼顾准确性和额外开销的问题，参考负载分析的结论，提出了基于块热度的数据放置策略（Block-Heat-Based Data Placement, BHB-DP），采用一种轻量级、两阶段的数据热度分类方案，根据写入频率和块寿命，通过计算加策略的方式获取块热度值，低开销动态识别冷热数据；再根据块热度值进行数据分类放置，减少 GC 过程中的数据迁移，降低整体写放大。最后，为了解决 GC 过程中的误判问题和避免负收益现象，还设计了 CBE（Cost-Benefit-Enhanced）算法用于挑选待回收区域，并与 BHB-DP 相结合，进一步降低 GC 造成的负面影响。

在真实场景负载下的测试结果表明，BHB-DP 相较目前最新的数据放置策略，写放大平均降低了 19%，垃圾回收写入量平均降低了 49.8%，垃圾回收次数平均减少了 21.1%。

关键词： 固态硬盘；分区命名；云块存储系统；数据放置策略；写放大

Abstract

Solid State Drives (SSDs) adopt a log-structured style internally to manage data. During Garbage Collection (GC), the valid data needs to be rewritten, leading to write amplification issues that damage the performance and lifespan of SSDs. Fundamentally, traditional SSDs cannot get the temperature information of data at the host side, resulting in the mixture of hot and cold data placements, thereby frequently triggering GC operations. Zone Namespace (ZNS) SSD allows users to customize data placement strategy at the host side and separate hot and cold data, offering a good opportunity to improve write amplification issues. However, facing the massive and frequently changing I/O load of cloud block storage system, how to place data appropriately in ZNS SSD to reduce write amplification is a severe challenge.

Firstly, through an in-depth analysis of the workload of cloud block storage system, it was discovered that the overall distribution of data block write frequency and block invalidation time follows Zipf's law. Then, a theoretical optimal data placement strategy for ZNS SSDs, called ZB-OPT (ZNS-Based OPT), was proposed. This strategy utilizes future information to classify block invalidation time into exponential threshold intervals and can achieve minimal write amplification, providing a theoretical upper limit for optimizing write amplification in real-world scenarios. Subsequently, regarding the issue that existing researches on hot-cold data classification based on predicting block invalidation time cannot balance accuracy and additional overhead, a block-heat-based data placement strategy (BHB-DP) was proposed based on the conclusions of the workload analysis. BHB-DP adopts a lightweight, two-stage data heat classification approach that calculates block heat values based on write frequency and block invalidation time using a policy-based method, which can dynamically identify hot-cold data at low cost. Then, data classification and placement are performed based on the block heat values to reduce data migration during GC and degrade overall write amplification. Finally, in order to address the problem of misjudgment during GC and avoid negative yield, the Cost-Benefit-Enhanced (CBE) algorithm was designed to select areas to be recycled. CBE is combined with BHB-DP to further reduce the negative impact caused by GC.

The test results under real-world workload show that, compared with the latest data

placement strategy, BHB-DP reduces write amplification by an average of 19%, reduces the total amount of garbage collection data written by an average of 49.8%, and reduces garbage collection frequency by an average of 21.1%.

Key words: SSD, Zone Namespace, Cloud Block Storage, Data Placement Strategy, Write Amplification

目 录

1 绪论

1.1 研究背景与意义

本课题来源于华中科技大学-腾讯联合实验室合作项目。近年来，数据中心和云存储服务提供商为了满足用户低延迟访问的需求，广泛采用了固态硬盘（Solid State Drives, SSDs）作为后端存储的硬件选择[1]，相较机械硬盘（Hard Disk Drives，HDD）能提供更低的延迟以及更高的带宽。为了简化后端持久化层对不同介质存储设备的管理，云存储服务提供商通常会使用块接口，将存储设备的存储空间视为一个由固定大小逻辑数据块组成的一维数组，可以以任意顺序读取、写入。

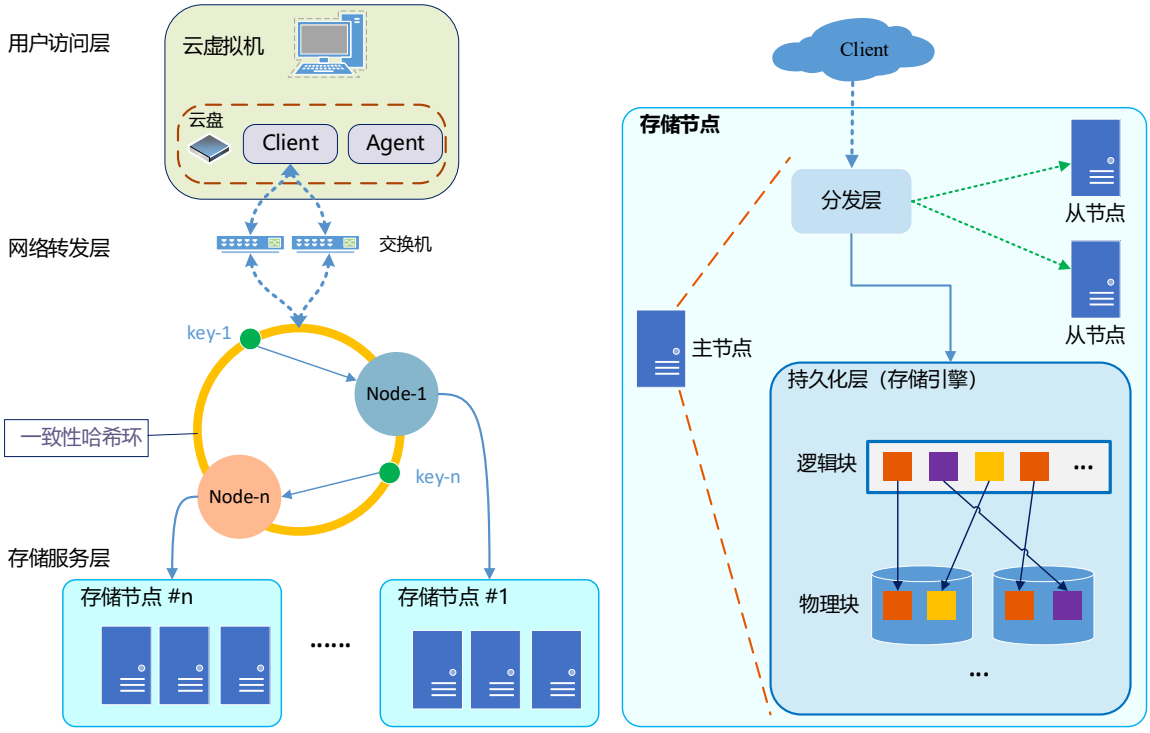
不同于 HDD，SSD 因其本身的物理特性，和块接口的需求之间存在显著的不匹配性（详见 1.1.2 所述）。尽管存储厂商在其内部隐藏了一个复杂的 FTL（Flash Translation Layer）模块，将块接口和 SSD 之间的差异抽象为逻辑块和物理页之间的映射，表面上掩盖了两者间的不匹配性，但 SSD 内部的 FTL 因无法感知主机端的数据热度信息，很容易将冷热数据放置于同一区域，需要频繁触发垃圾回收操作来获取可用空间以写入新数据，会造成严重的写放大问题，加速 SSD 磨损，降低其使用寿命，影响性能。与此同时，云存储服务提供商开始大力推行“降本增效”措施，尽管新一代基于 TLC、QLC 闪存颗粒的 SSD 成本有所下降，但其理论寿命会大幅缩短[2]。在这些低成本 SSD 逐渐普及的当前，解决严重危害 SSD 寿命的相关问题（例如写放大等）迫在眉睫。

ZNS 是 NVME 规范中最新的一套命令集，全称为 Zoned Namespace Command Set，支持 ZNS 的 SSD 被称为 ZNS SSD，允许主机端自定义传统 FTL 的核心功能（例如数据映射、数据放置策略和垃圾回收等）。ZNS SSD 的出现为改善传统 SSD 的写放大问题提供了良好的契机。

本课题旨在研究适用于云块存储系统持久层场景下 ZNS SSD 的数据放置策略，根据负载特征以及 ZNS SSD 的特性进行优化和改进，降低写放大并提高系统性能。

1.1.1 云块存储系统

云块存储系统[3]是一种基于云计算技术构建的分布式存储系统，主要面向需要高性能、高可用性、高容错性的应用场景，各主流云服务提供商均有云块存储的产品布局（例如 Tencent CBS[4]，Amazon EBS[1]，Alibaba EBS[5]等）。云块存储系统通常采用分布式的存储节点，将用户数据切割成众多逻辑块（通常为 4KB），并在多个存储节点之间打散分布存放，以提高存储的可靠性和性能。



(a) 整体架构　　(b) 存储节点架构

图 1.1 腾讯云块存储系统多层架构

图 1.1 (a)介绍了腾讯云块存储系统的整体架构，主要包括用户访问层、网络转发层和存储服务层。

（1） 用户访问层负责和用户进行交互。Client 负责接收用户的请求，是用户与云块存储系统进行交互的接口。Client 通过标准块存储协议（例如 iSCSI）向存储系统转发用户的读写请求，经网络转发层和存储节点进行交互。Agent 负责运行管理，

[1] https://aws.amazon.com/cn/ebs/

监控客户端的运行状态，维护其正常运行以及提供安全保护。

（2） 网络转发层负责数据传输和网络管理。为保证满足严苛的 SLA（Service Level Agreement）标准，当前的数据传输通常基于数据中心内部的专用交换机以及高速光纤网络进行，同时网络转发层也会负责管理网络连接和传输协议，确保数据的可靠传输。

（3） 一致性哈希的作用是尽可能地实现用户侧的逻辑数据均匀放置于各存储节点中，达到良好的负载均衡效果。用户请求在到达存储服务层之前会根据一致性哈希算法进行逻辑地址到存储节点内实际存储地址的转换。

（4） 存储服务层提供存储设备管理和数据存储服务，包含多个存储节点。存储服务层负责将数据存储到物理介质上，如 HDD 或 SSD，并实现数据的访问和管理。

图 1.1 (b)为存储节点内的详细视图，主要包括分发层和持久化层。

（1） 分发层主要负责接收和处理经过了一致性哈希后的用户 I/O 请求，并返回数据落盘后的结果。为了保证数据的可靠性和可用性，存储服务层的存储节点通常三个为一组，其中一个主节点、两个从节点，组成三副本。若当前节点为主节点，还会将用户 IO 转发到从节点，并接收从节点的反馈。

（2） 持久化层负责维护本地存储引擎，保证逻辑块和物理块之间相互转换的速度和可靠性。

1.1.2 传统 SSD 的写放大问题

持久化层内部利用底层存储引擎实现标准化接口（例如块设备接口）对用户的逻辑数据块进行存取。最初引入块接口是为了隐藏硬盘介质特性和简化主机软件，它在许多代存储设备上都能很好地工作，但和 SSD 的物理特性之间存在显著的不匹配性。具体表现为，对于由闪存介质构成的 SSD 而言，尽管单个数据块（通常为 4KB）可以被写入空白区域，任意读取，但无法实现原地更新，需要以一个更大的“擦除块”粒度（例如 2MB）先将芯片重置后才能写入新的数据，而往往擦除块中仍然存在有效数据块，即需要先进行数据迁移，然后才能执行擦除操作。存储厂商为了解决上述“不匹配性”所造成的问题，使 SSD 能兼容块接口，在其内部实现了一个复杂的 FTL，且对主机端不可见。

基于 NAND 闪存芯片的传统 SSD 架构如图 1.2 所示，主要包括 FTL、控制器、板载缓存、存储芯片等。其中 FTL 为核心组件，功能模块包括数据管理、垃圾回收、磨损均衡、异常处理等。数据管理模块通常负责维护记录着逻辑块到物理块映射关系的映射表，并将新逻辑数据按固定策略写入空白地址中。垃圾回收负责在对擦除块进行重置操作前，将有效数据迁移到另一区域，然后重置该擦除块用以写入新数据。磨损均衡负责平衡 SSD 内全部闪存芯片的擦除次数。异常处理模块负责处理 SSD 内部的各种故障，例如芯片故障、写入错误等。

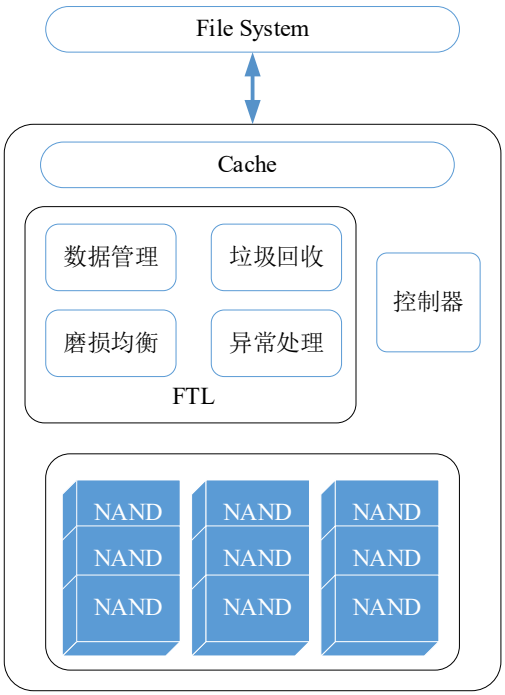


图 1.2 传统 SSD 架构

尽管内部 FTL 使得传统 SSD 能够兼容块接口，但仍存在诸多问题，包括写放大问题、容量过度配置问题、读写性能影响问题等，严重危害 SSD 的使用寿命以及性能表现。这些问题的本质原因均在于 SSD 内部数据放置不合理。如果能够区分冷热数据块，将其分区域放置，使得他们在相同或者接近的时间内失效，减少甚至避免闪存芯片擦除之前的数据迁移操作，即可很大程度上规避上述问题。而现实情况是 SSD 内部的 FTL 无法感知主机端数据的热度信息，很难实现良好的冷热数据分类放置效果。

学术界和工业界为了改变这一现状做出了诸多尝试。Multi-stream SSD[6]是三星

公司最早提出的概念，它通过提供一个存储接口，使主机系统能够告知 SSD 设备有关被写入数据的期望生命周期，从而缩小主机系统与 SSD 设备之间的语义差距[7]。但多流 SSD 并不能完全同步主机端和 SSD 内部的垃圾回收操作，对于解决传统 SSD 的相关问题而言治标不治本。

OC（Open-Channel）SSD[8]将 SSD 内部的 FTL 完全移交至主机端，迁移的功能有数据放置、数据映射、垃圾回收、IO 调度、异常处理等等[9]，理论上允许主机端根据自身业务和应用的特点进行针对性优化。但不同厂商的 SSD 产品往往具有不同的特性，行业内并没有完整统一的规范和标准，使得适配与维护成本太高。

1.1.3 ZNS 和 ZNS SSD

Zoned Namespaces 是 NVME 规范中最新的一套命令集，该命令集暴露了一套接口（zoned block storage interface）给主机端，该接口将 SSD 内部存储空间划分为多个分区进行管理，在每个分区内允许随机读，但只允许顺序写。从硬件上支持 ZNS 的 SSD 被称为 ZNS SSD。

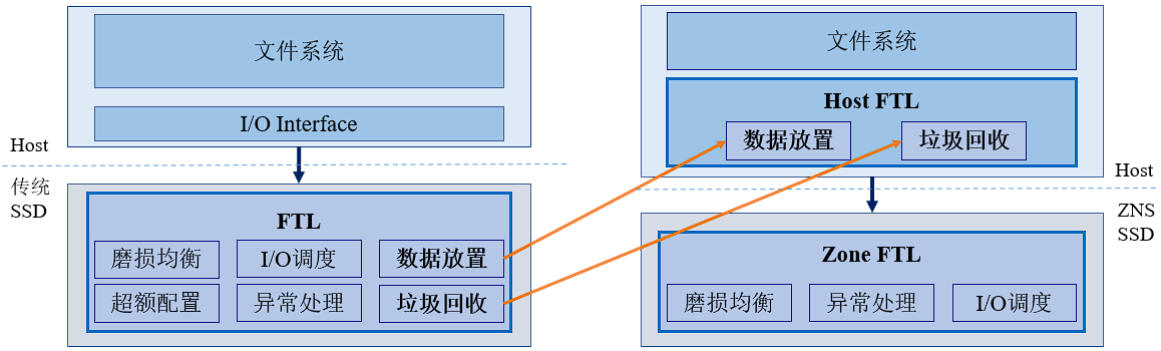


图 1.3 传统 SSD 和 ZNS SSD 的架构差异

图 1.3 展示了传统 SSD 和 ZNS SSD 在架构上的差异，ZNS SSD 允许主机端自定义 Host FTL 进行数据管理，根据不同类型的负载实现相对应的数据放置策略和垃圾回收算法，理论上能够很好改善传统 SSD 的写放大问题。和 OC SSD 完全将 FTL 移交至主机端不同，ZNS SSD 在其内部保留了一部分难以制定统一规范的 FTL 功能（即 Zone FTL），例如异常处理，由厂商根据各自的闪存介质特性单独定制。故 ZNS SSD 无需掌握专业且复杂的闪存介质相关知识，大幅降低了适配和维护成本。

Matias Bjørling 等人[10]在理论测试中指出，传统 SSD 的写速率会随写入量的增

加而陡然下降，究其原因是其内部将冷热数据混合放置，导致写入量达到 SSD 容量上限后需要频繁进行 GC 操作释放空间，搬移有效数据，占用写带宽，影响写速率。ZNS SSD 若能够分区放置冷热数据，则无需进行有效数据迁移，不存在该现象。

1.2 国内外研究现状

本节将从以下两个方面介绍国内外关于本课题已有的研究，首先介绍目前已提出的关于写放大优化的研究工作，主要包括上层存储系统优化、SSD 内的数据放置策略优化以及垃圾回收算法优化，然后介绍有关 ZNS SSD 应用的最新进展。

1.2.1 现有写放大优化研究

因传统 SSD 内部很难获取主机端数据块的热度信息，目前的写放大优化工作主要围绕上层存储系统、SSD 内部的 FTL 优化以及垃圾回收算法优化这三方面进行。其中存储系统通常指日志结构文件系统（Log-structured File System，LFS[11]）及其衍生项目，例如 F2FS[12]、BetrFS[13]等，内部采用追加写的方式写入数据，无法原地对数据进行修改，故需要通过垃圾回收操作先迁移有效数据然后释放整块区域以获得可用空间，存在写放大问题。

优化工作的核心思想均在于分离冷热数据，尽可能将冷热程度相同的数据块放置于同一区域，使其能在相近的时间内失效，从而降低写放大，减少垃圾回收造成的开销。现有研究尝试利用诸多特征和不同方法对数据的冷热程度进行刻画。

（1） 关于日志结构存储的写放大优化

SFS[14]利用写入频率和写入时间的商作为热度值，将冷热数据分离到缓冲区缓存中的不同段，并将它们分批写入 SSD，同时，SFS 推迟了小写写入，直到一个文件段被完全填满。

F2FS[12]利用文件本身的静态语义进行冷热数据区分，文件类型主要有索引块和数据块两大类，对于索引块而言，目录的直接索引节点为热数据，普通文件的直接索引节点为暖数据，间接索引节点为冷数据。对于数据块而言，目录数据块为热数据，文件数据为暖数据，段清理数据块和多媒体数据等为冷数据。

PCStream[15]根据应用程序的上下文实现对数据块的分类。例如当应用程序写入

元数据时，它们遵循特定的执行路径，这与写入用户数据时的执行路径不同。然后，PCStream 使用 k-means 聚类算法根据每个程序上下文的数据生命周期进行冷热数据分类。

Seer-SSD[16]通过将过期逻辑块的元数据信息与 LFS 写入的数据一起传输到 SSD 中，在不影响文件系统一致性的情况下，使得过期逻辑块在 SSD 中主动或选择性地失效。解决了 LFS 和 SSD 内数据失效不同步所造成的额外写放大问题。

Gala Yadgar 等人[17]首次提出了针对 SSD 设计的 I/O 工作负载分析，利用了来自四个大型存储库的负载，详细研究它们的温度范围、对页面大小的敏感性和逻辑局部性，并将这些特征与三个 SSD 的标准性能指标——写放大、读放大和 flash 读取成本相关联，最后给出了他们的分析结论。

SepBIT[18]基于对满足类似 Zipf 分布数据的概率分析结论，将数据块分为“用户写入块”和“GC 写入块”，并利用数据块前一次的寿命推断后续寿命，通过块寿命对数据块进行冷热区分。

综上所述，现有关于日志结构存储的写放大优化工作主要利用了上层系统中丰富的文件语义信息、程序上下文信息以及负载特征来对数据块的冷热属性进行判断，将不同属性的数据块分类放置进而降低系统的整体写放大。

（2） 关于 SSD 的写放大优化

2-Level LRU[19]借鉴了缓存算法对局部性特征的利用，采用三种类型区分数据块冷热程度——“unclassified”，“candidate list”，“hot list”。该方案通过将逻辑地址从热列表移动到候选列表，再从候选列表移动到未分类状态来对逻辑地址的冷却进行建模。候选列表和热列表大小固定，一旦列表填满就会清除最近最少使用元素。

Soojun Im 等人[20]根据每个请求的大小进行分类。大写请求表示冷数据，小写请求表示热数据，除简单之外，这种策略开销极小，仅需要几个字节的 DRAM，在其他研究（如 Fabio Margaglia 等人[21]）中也有被提及。

WDAC[22]维护了一个先入先出队列存放最近写入的 N 次请求，根据权重值w_i来区分数据块冷热，其中$w_i = 2 - i * \frac{2}{N}$，$i \in [0, N]$，考虑到维护该队列所需的庞大内存开销以及 SSD 内有限的内存容量，MBF（Multiple Bloom Filter）[22][23]利用布隆过滤

器和轮询的方式对 WDAC 进行了改进，在分类过程中，计算逻辑地址匹配的布隆过滤器的加权和，如果加权和大于预定义的阈值，则将数据分类为热数据。

Bhimani J 等人[24]提出了一种基于特征的流识别方法 FIOS，将可测量的工作负载属性（如 I/O 大小、I/O 速率等）与高层工作负载特征（如频率、时序性等）联系起来，确定用于分配流 id 的工作负载特征之间的最佳组合。

J. Yang 等人[25]提出了 WARCIP，其核心思想是使用一种聚类算法来最小化数据流中数据块的重写间隔方差，然后将相同类别的数据块写入到相同的数据流中。

FADaC[26]通过在线调整分类器的参数，并在闪存页面的元数据中存储额外的信息来对任意给定的内存进行操作，从而实现更好的数据分类效果。

MLDC[27]利用 K-means 算法借助逻辑地址、请求大小、历史写入频率、新旧程度四个特征对数据块进行聚类，然后借助 CART[28]算法生成决策树作为分类器，对数据块进行分类。

Yang 等人[29]提出了一种根据预测数据块温度来放置数据的方案。应用 LSTM[30]神经网络提高温度预测的时间和空间准确性。使用 K-Means 算法进行聚类，并自动将相似的“未来温度”数据分发到相同的 NAND 块。

ML-DT[31]利用时间卷积网络 TCN，根据 I/O 大小、逻辑地址、历史块寿命等特征生成预测模型，对数据块的未来寿命进行预测，从而实现冷热数据分离。

综上所述，近年来 SSD 的写放大优化工作普遍采用了聚类算法来获取良好的数据块分类效果，同时，为了追求更高的准确性，研究者们加入了各种相关指标，如 I/O 大小、逻辑地址等，以及更加复杂的预测模型，如 LSTM、TCN 等。

（3） 垃圾回收算法优化

当 SSD 内部失效数据占比超过一定的阈值（例如 15%）后，控制器会触发垃圾回收操作。现有的垃圾回收算法旨在根据不同的指标，在候选区域中选择最合适的部分进行回收。

Greedy[32]贪婪算法和 Cost-benefit[11]成本-效益算法是两种最经典的垃圾回收算法，通过计算被写满区域的“收益值”，并选择得分最高的区域进行回收。Greedy 算法直接将已写满区域的垃圾占比作为收益值，而 Cost-benefit 算法则加入了区域年龄

进行收益值的计算。考虑到遍历全部 SSD 的开销过高，后续研究提出了众多衍生算法来降低选择过程的开销，例如 Windowed Greedy[33]，Random-Greedy[34]，d-choices[35]等。Cost-Age-Times[36]在选择回收区域的时候，同时考虑了回收开销、数据块年龄和闪存介质擦除次数。

此外，也有一些研究者提出了创新性的方案来降低甚至消除垃圾回收对系统的影响。Lifespan-based GC[37]基于固定时间片轮转的方式，对 SSD 内的区域循环进行垃圾回收，在具有良好周期性的负载场景下大幅降低了系统写放大。IPLFS[38]通过实现两个关键的技术：IPLFS（无限分区的日志结构文件系统）和区间映射（用于无限文件系统分区内空间高效的逻辑地址到物理地址映射），达成了在 SSD 寿命期间内无需进行垃圾回收的效果。

1.2.2 ZNS SSD 在存储系统中的应用

ZNS SSD 作为新硬件，近年来受到了众多研究者的重点关注，Theano Stavrinos 等人[39]建议将现有针对于传统 SSD 的研究完全转移到 ZNS SSD 上来。Devashish R. Purandare 等人[40]详细阐述了 ZNS SSD 在当前主流的日志结构数据管理领域的优势，并给出了研究方向和研究建议。

当前已有的关于应用 ZNS SSD 的研究主要围绕基于 LSM-tree 的键值存储场景展开，因为 ZNS SSD 和 LSM-tree 均遵循严格的顺序写约束而天然契合的特性，理论上 ZNS SSD 在该场景下能充分发挥其优势。Peiquan Jin 等人[41]以面向 SQL 数据库和 key-value 存储的索引结构 B+-tree 和 LSM-tree 为研究对象，探讨了修改 ZNS SSD 索引结构面临的挑战和研究机遇。Gijun Oh 等人[42]提出了一种根据 key-value 数据特点进行排序的技术，以最大化 ZNS SSD 的空间利用率和降低垃圾回收的开销。Jeeyoon Jung 等人[43]指出将 ZNS SSD 应用于 LSM-tree 场景时会遇到空间放大问题，提出了基于 SSTable 生命周期的 LL-compaction 算法，减少了 ZNS SSD 内部非必要垃圾回收操作，提高系统性能。Hee-Rock Lee 等人[44]提出了 CAZA 算法来降低现有算法 LIZA 对 ZNS SSD 内分区分配不合理所造成的额外开销，显著降低了写放大。

此外，也有一些其他关于 ZNS SSD 的优化工作。Gunhee Choi 等人[45]通过增大 ZNS SSD 垃圾回收时的 IO 粒度，平均提升了 1.9 倍的垃圾回收性能。Kyuhwa Han

等人[46]提出了基于 ZNS 的新技术，将垃圾回收过程中的数据迁移卸载到 SSD 内部进行，大幅提高了垃圾回收效率，减少了垃圾回收操作对系统的影响。

综上所述，使用 ZNS SSD 需要合适的数据放置策略与之配合。利用合理的方法完成数据的冷热区分后，再将冷热程度相近的数据写入同一个分区中，使得该分区在被回收时里面的数据几乎均已失效，则能很大程度上改善 SSD 的写放大问题。

1.3 现有研究存在的问题

对于拥有海量数据的云块存储系统而言，降低系统写放大的核心在于提高数据冷热分类的准确性，而如何设计一种低开销且有效的冷热数据分类方案，以实现良好的数据放置效果是一个严峻的挑战。本节将对现有研究存在的问题进行讨论。

1.3.1 写放大研究存在的问题

参考现有写放大研究中对于数据放置策略的优化工作，应用于云块存储系统中时，存在的问题主要可以归结为以下三类：

（1） 开销过高。现有方案[25][26][27]广泛采用了聚类算法实现对数据的分类，并训练分类器来区分冷热数据，或是利用神经网络[29][31]预测数据块寿命，将寿命接近的数据块放置于同一区域，均取得了不错的分类效果，但对于每天写入请求过亿的大型云块存储系统而言，负载频繁变化，导致数据块热度以及整体分布也在不断变化，若是不能及时更新分类器或预测模型，则冷热数据分类效果将大打折扣，但频繁聚类或是重新训练预测模型会带来庞大的计算和存储开销，尽管可以通过抽样算法来降低开销，准确性和分类效果仍无法得到保障。

（2） 适用性不佳。基于缓存思想保留热数据来实现冷热数据区分的方案[19][22]对数据的强局部性特征有较高要求，不适用于负载频繁变化的云块存储场景。Qiuping Wang 等人[18]的方案在块存储系统用户数据管理层对于单租户数据的管理能够实现较好的冷热数据分类效果，但在本文多租户数据混合的持久化层场景中表现不佳。

（3） 场景不匹配。不同于文件系统，以块为粒度进行数据管理的块存储系统无法获取到丰富的文件级别的语义信息，或是应用程序级别的上下文信息，尽管这类方案[12][14][15]获得了较好的优化效果，但无法直接应用于本文的研究场景。

对于垃圾回收算法而言，尽管常用的 greedy 及其衍生算法总能选择到局部最优（垃圾占比最高）的回收区域，但这样很可能会导致额外的非必要数据迁移，因为存在部分被回收的数据块会很快过期。Cost-benefit 算法具有较大潜力改善 greedy 算法存在的问题，但需要和数据放置策略相适配。此外，其他一些垃圾回收算法[37][38]局限性太强，不适用于本文场景。

1.3.2 ZNS SSD 应用于云块存储系统的问题

现有关于 ZNS SSD 的研究工作主要围绕键值存储场景进行，LSM-tree 的分级结构天然蕴含了冷热信息，但云块存储场景无法利用这些特征。对于在云块存储系统中应用 ZNS SSD 而言，目前还缺少两类工作：

（1） 缺少块寿命分析。块寿命[47]这一指标最早由 J. He 等人在 2017 年提出，指数据块从被写入到失效的时间。理论上若是能提前预知每个数据块的有效时间，则能实现最佳的数据放置效果，被应用于众多写放大优化研究中[25][29][31]，尽管有一些工作[48][49]对块存储系统租户的 IO 负载进行了分析，但这些分析主要为宏观层面的数据统计，例如读写比例、读写数据工作集大小、用户卷活跃程度等，并没有针对数据块的块寿命进行详细分析，无法为块存储系统存储节点内的写放大优化提供帮助。

（2） 缺少适用于 ZNS SSD 的指导性理论最优放置策略。不同于缓存算法中公认理论最优的 Belady 算法作为缓存性能的上限，可以用于评估其他缓存算法的实际效果，考虑到不同场景对于数据放置策略的诸多限制，目前并没有一种通用的最优策略作为理论上限为所有场景下的数据放置策略的优化提供指导，衡量现有数据放置策略对于冷热数据分类效果和最佳情况之间的差异。尽管现有研究[18][31]中提出了适用于传统 SSD 的理论最优放置策略，但受限于 ZNS SSD 同时开放分区数量的物理限制，无法直接应用。

1.3.3 存在问题小结

云块存储系统每天都会接收数以亿计的 I/O 请求，负载频繁变化导致数据块热度也随之不断改变，且以块为粒度管理数据无法获得文件级别丰富的语义信息，使得现有关于写放大优化的研究存在开销过高、适用性不佳、场景不匹配等问题，同时，

垃圾回收算法的选择需要和数据放置策略相匹配。而若要设计一种适用于 ZNS SSD 的数据放置策略，目前还缺少块寿命分析以及理论最优放置策略这两类工作。

1.4 本文主要内容

1.4.1 主要研究内容

从 1.3.3 的结论可知，现有关于写放大优化的研究不能很好地直接应用于基于 ZNS SSD 的云块存储场景，为此，本文首先对真实场景下的云块存储系统负载进行了详细探究，给出了分析结论。然后根据分析结论提出了新的最优策略作为理论上限，用于更好地衡量实际数据放置策略之间的差异及尚存在的改进空间。最后提出了基于块热度的数据放置策略，用于优化云块存储系统的写放大问题。

本文的主要工作可以归纳为下述 4 个方面：

（1）云块存储系统的写负载分析。本文对来自大型云块存储系统的真实负载进行了深入地特征分析，包括写流量变化、写入频率分布、块寿命分布及其分类阈值变化等，挖掘不同负载下这些关键特征的变化规律，为后续研究提供理论基础。

（2）适用于 ZNS SSD 的理论最优放置策略 ZB-OPT。现有针对于传统 SSD 的理论最优放置策略 FK 在 ZNS SSD 上表现不佳，本文通过理论分析以及实际测试，发现 ZNS SSD 有限的同时开放分区数量无法满足 FK 细粒度切分后的不同块寿命区间所需要的大量分区类别。然后依据数学分析，提出了 ZB-OPT，采用指数型增加的逻辑寿命阈值进行冷热数据块分类，能够更好地适配 ZNS SSD 的特性，最小化存储系统的写放大系数。

（3）基于块热度的数据放置策略 BHB-DP。考虑到云块存储系统每天庞大的写请求数量，频繁对数据块进行聚类或训练预测模型开销太大。写入频率能够低开销地实现冷热数据分类，但对数据块热度变化感知不够及时，块寿命具备一定的适应负载热度变化的能力，但分类效果不佳或开销太大。BHB-DP 采用一种轻量级的数据热度分类方案，根据写入频率和块寿命，通过计算加策略的方式获取块热度值，有机结合了写入频率和块寿命这两种关键指标的优势并规避了其不足。同时 BHB-DP 还结合了优化后的效益最佳算法 CBE，利用分区被写满的时间来获取分区寿命，以及在选

择待回收分区时跳过无效数据占比小于系统垃圾回收阈值的分区，可以有效避免 GC 过程中的误判问题和负收益现象。

（4）利用真实场景的数据集进行仿真实验。本文在开源日志结构存储引擎上拓展实现了所提出的数据放置策略，以及关于 ZNS SSD 的管理模块。对来自云块存储系统的海量真实负载日志进行了采样用于仿真测试，从写放大、垃圾回收写入量、垃圾回收次数三个维度对本文方案进行评估，验证其有效性。

1.4.2 本文组织结构

本文共有五个章节，具体内容组织如下：

第一章首先简要介绍了云块存储系统，本文的研究背景及研究意义，然后介绍了现有关于写放大优化的研究工作以及新硬件 ZNS SSD 的应用分析，接着指出了现有研究存在的问题以及不足，最后介绍了本文为解决这些问题所做的工作。

第二章对云块存储系统的负载信息进行了详细分析，包括基础特征分析，写入频率分析和块寿命分析，对现有针对传统 SSD 的最优放置策略 FK 表现不佳的原因进行了探究，然后构思并提出了新方案 ZB-OPT。

第三章介绍了基于块热度的数据放置策略优化方案 BHB-DP 的整体架构，详细阐述了设计思路和具体实现，包括数据放置算法、垃圾回收算法、元数据管理和 ZNS SSD 设备管理。

第四章介绍了测试环境以及 ZNS SSD 仿真参数设置，对参与对比测试的所有方案和评估指标进行了说明，然后对实验结果进行了详细分析。

第五章对本文所做的研究工作进行总结，并讨论了对未来基于 ZNS SSD 的块存储系统优化工作的展望。

2 针对写放大问题的理论最优放置策略研究

第一章指出了现有研究工作中存在的问题以及不足，本章将基于云块存储场景下的真实负载进行深入的分析，包括写流量变化、写入频率分布、块寿命分布及其分类阈值变化。然后将会探究现有理论最优策略 FK 表现不佳的原因，并提出适用于 ZNS SSD 的理论最优方案 ZB-OPT，最后总结负载分析结果和理论最优方案对数据放置策略优化工作的启示。

2.1 云块存储系统写负载分析

本节将对云块存储系统持久化层的负载进行深入分析，包括基础特征和两个核心特征——写入频率和块寿命。

2.1.1 数据来源及处理

腾讯云硬盘作为目前主流的云存储服务产品之一，数据规模已突破 EB 级，由全球多个地域的众多云块存储系统集群提供有力的存储服务支持。

本文分析的数据来源于腾讯云块存储系统某个集群为期 6 天的客户端接入模块 IO 日志记录，该存储集群正在为不同类型客户的数以万计的云盘提供块存储服务，日志中的每项记录包含了一次 I/O 的请求时间、用户卷 ID、请求偏移地址、I/O 大小、I/O 读写类型等信息。同时，为了严格保密用户隐私，数据中的用户卷 ID 已经过脱敏处理。

为了获取用户 I/O 抵达存储节点持久化层实际落盘时的负载信息，本文利用和腾讯云块存储系统内同样的一致性哈希算法以及负载均衡方法，将用户 I/O 打散至数十个存储节点中，并从中随机挑选出 10 个节点进行详细分析。

为了兼容传统 HDD 以扇区为粒度读写数据（通常为 512 字节），腾讯云块存储系统同样以扇区为最小粒度管理用户云盘的逻辑数据。为了匹配 SSD 的最小读写粒度（通常为 4KB），本文对全量 I/O 负载进行了 4KB 对齐和切分操作。

此外，受限于实验设备的内存容量，无法对切分后的负载数据进行全量分析，为了保持负载的关键特征，特别是局部性特征，本文采用了 C. A. Waldspurger 所提出的

空间哈希采样方法 shards[50]，利用 MurMurHash3 算法对负载数据进行了哈希采样，当且仅当hash$(offset + volume_{id})\ mod\ P < T$的I/O会被采样，其中$P$为模运算底数，T 为采样阈值，本文设置为$P = 10$，$T = 1$，即采样率为 1/10，以确保采样误差在可接受范围以内。

2.1.2 数据基础特征分析

（1）数据规模分析

本文旨在研究并优化持久化层的写放大问题，故负载分析仅重点关注写请求。唯一数据量用 WSS（Working Set Size）表示，通过负载信息中的$offset$和$volume_{id}$来标定每个 4KB 逻辑数据块。10 个节点全量数据和采样后的整体信息如表 2.1 所示。

表 2.1 负载信息统计

节点	全量数据				1/10 采样数据		
	总写入量 (GB)	写入量占比	写请求量	写请求占比	写入量 (GB)	WSS (GB)	用户卷数量
1	4715.64	78.26%	187163732	91.44%	483.09	62.10	255
2	4705.13	81.36%	216832359	93.00%	580.66	57.74	264
3	3942.24	67.32%	233842176	90.14%	432.82	63.29	278
4	4316.94	64.12%	257699142	89.54%	467.57	62.76	277
5	4670.37	81.56%	197498254	90.64%	556.10	62.89	293
6	3053.79	77.37%	97782554	86.69%	325.05	61.36	261
7	4529.42	78.90%	174069745	89.70%	512.78	64.24	274
8	2665.51	77.91%	95877292	86.07%	296.26	60.92	287
9	3763.34	76.37%	151040522	86.99%	406.31	63.90	279
10	2974.90	75.30%	104727883	85.93%	329.07	64.30	261

从表 2.1 可以看到，块存储系统内大部分节点写入量和写请求占比都很高，说明优化数据放置策略可以带来很高的性能和成本收益。经过一致性哈希等负载均衡方式处理后，各个节点的 WSS 相差不大，均为 60GB 左右，每个节点的用户卷也在 270 个左右，说明用户写入的逻辑数据分布被成功打散，达到了较好的均衡效果。但写入量和写请求量相差较大，总写入量分布从 2665GB 到 4715GB 不等，写请求量分布从 9.5 亿到 25 亿不等，说明数据写热度分布十分不均。

（2）节点写流量变化

以节点 1~4 前 5 天的数据为例，节点随时间的写流量变化如图 2.1 所示。

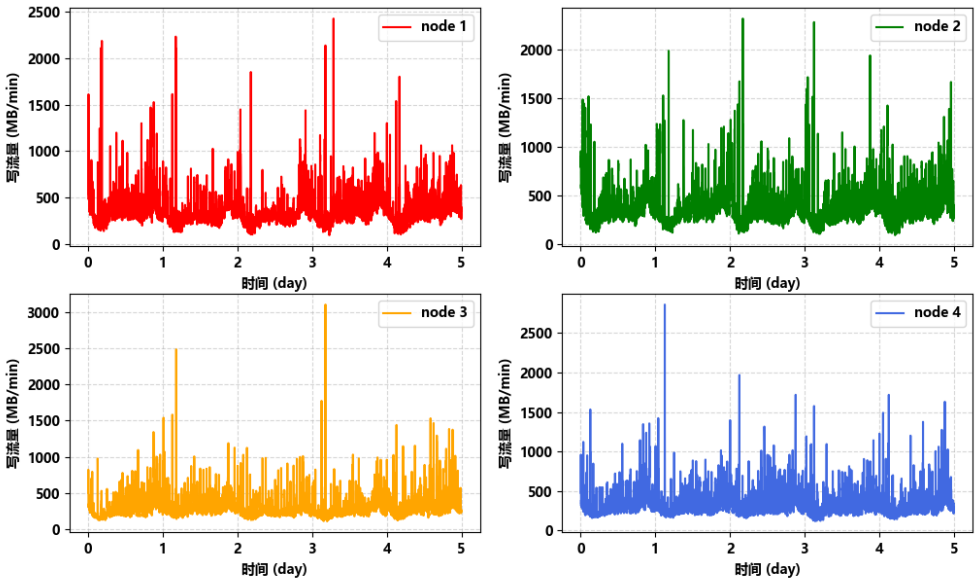


图 2.1 节点 1~4 为期 5 天的写流量变化

对于单个节点而言，整体均呈现出以天为粒度的周期性特征，其中节点 1 和 2 的周期性较好，每天刚开始数小时（0~4 点）内流量较高，峰值能达到 2000~2500MB/min，其他大部分时间在 200~1500MB/min 之间波动。而对于节点 3 和 4 而言，存在少量突发流量，例如节点 3 的第一天和第三天，以及节点 4 的第一天，超过了 2500MB/min，其他绝大部分时间均在 1500MB/min 以下。

说明对于不同节点而言，分散后的 I/O 负载模式不同。即使对于同一个节点而言，I/O 负载也在不断变化。

2.1.3 写入频率分析

写入频率这一特征被广泛应用于各种区分冷热数据的算法设计之中。例如经典缓存算法 LFU、LRU-K、ARC[51]、LIRS[52]等，以及现有的数据放置策略 MLDC[27]、FIOS[24]等。

本节以节点 1~4 为例，以天为粒度统计分析了其为期 4 天的写入次数分布。分为后 1%~100%和前 0.01%~1%两段，而前 0.01%以内的数据块写入次数极高，通常为极端突发流量，不具有参考价值，故并未在图中进行展示。

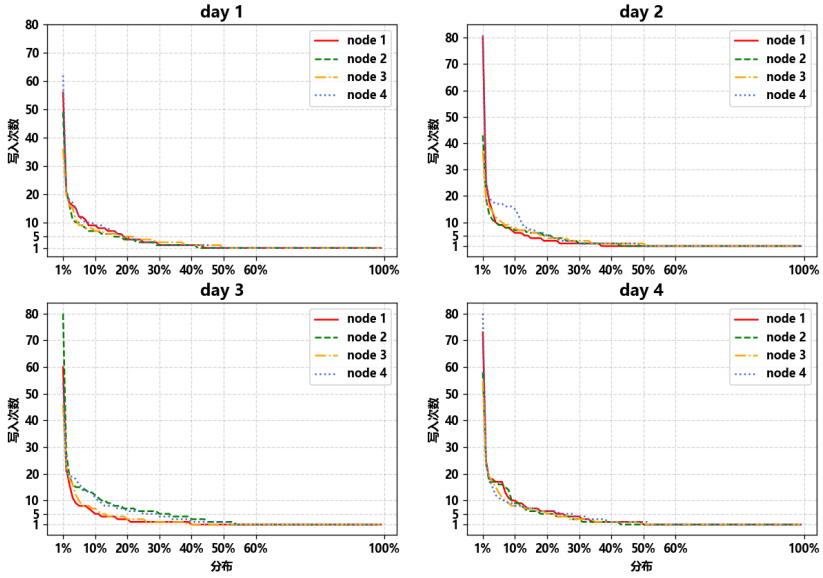


图 2.2 节点 1~4 为期 4 天的写入次数分布（后 1%~100%）

图 2.2 为节点 1~4 为期 4 天的后 1%~100%写入次数分布情况，可以看到整体符合 zipf 分布。对于节点 1~4 而言，均有超过 40%的数据块只写入了一次，仅有不到 10%的数据块写入次数在 10 次以上，而前 1%的数据块一天的写入次数也仅有不到 100 次，即存在大量数据块写入次数较少。

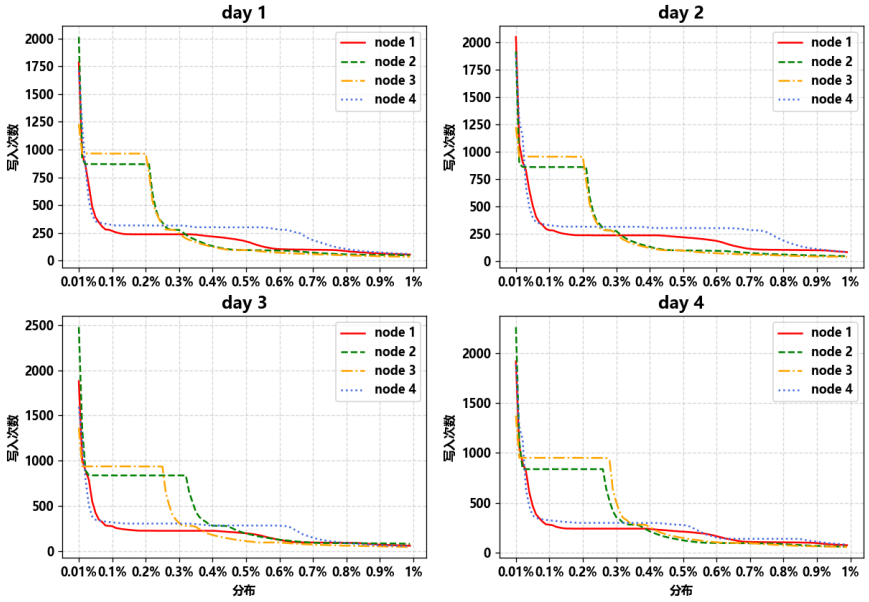


图 2.3 节点 1~4 为期 4 天的写入次数分布（前 0.01%~1%）

图 2.3 为节点 1~4 为期 4 天的前 0.01%~1%写入次数分布情况，可以看到整体同样近似符合 zipf 分布，但写入次数差异程度远大于后 1%~100%的写入次数分布。全部节点前 0.01%数据块写入次数超过了 1500 次，和前 1%数据块不到 100 的写入次数相差两个数量级以上。节点 1 和节点 4 写入次数最高的部分（超过 250 次）主要由不到 0.1%的数据块构成，而节点 2 和节点 3 则是有 0.3%，同时存在 0.2%左右的数据块写入在 1000 次左右，说明节点 1 和节点 4 的热点数据相较 2 和 3 更为集中，且进一步说明了不同节点的负载写入模式不同。

2.1.4 块寿命分析

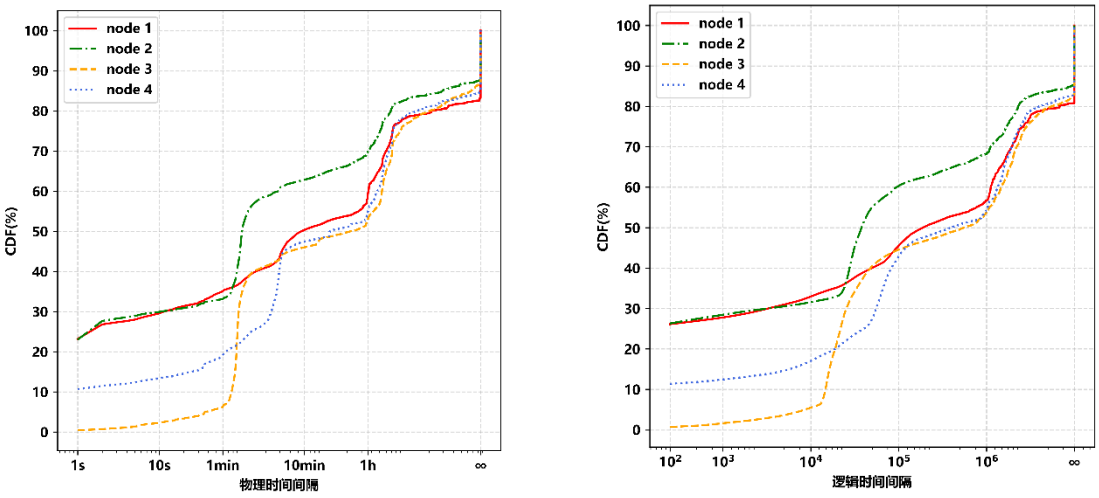
块寿命指数据块从被写入到失效的时间间隔，现有研究中的块寿命主要包含以下两种：

- Block Invalidation Physical Time（BIPT）[25][37]，块物理寿命。指同一数据块两次写入之间的物理时间间隔，通常通过系统时间戳获取。能直观地反映数据的不同生命周期，但对于负载的稳定性和周期性有较高要求。
- Block Invalidation Logical Time（BILT）[18][29][47]，块逻辑寿命。指同一数据块两次写入之间的其他数据块写入次数，通常通过全局写入计数器获取。相较物理寿命而言具有更好的通用性，更适用于理论分析以及方案设计。

本小节对块存储系统中的两种块寿命特征都进行了分析。块寿命的统计范围为采样后为期 6 天的全量数据。以节点 1~4 第一天的块寿命为例，分别统计了其 BIPT 和 BILT 的整体分布，累积分布函数如图 2.4 所示。横坐标为时间间隔（取对数），图 2.4 (a)为物理时间，图 2.4 (b)为逻辑时间。纵坐标均为为 CDF 值。分别对比各节点的 BIPT 和 BILT 分布，从中可得，整体上看趋势相同，说明物理寿命和逻辑寿命这两种指标在云块存储场景下从宏观视角上来看并没有明显区别，均近似服从 zipf 分布。所选 4 个节点块寿命分布范围都很广，物理寿命短的不到 1 秒，长的可达数小时甚至数十小时，对应逻辑寿命从小于 10^2 到大于 10^6，均存在一定占比。且 4 个节点均存在 10%~20%的块寿命为无穷大。

聚焦于单个节点而言，节点 1 和节点 2 存在超过 20%的块寿命极短，在 1s 以内，对应逻辑时间间隔小于 10^2，而节点 3 则几乎没有这种类型的负载。另一方面，节点

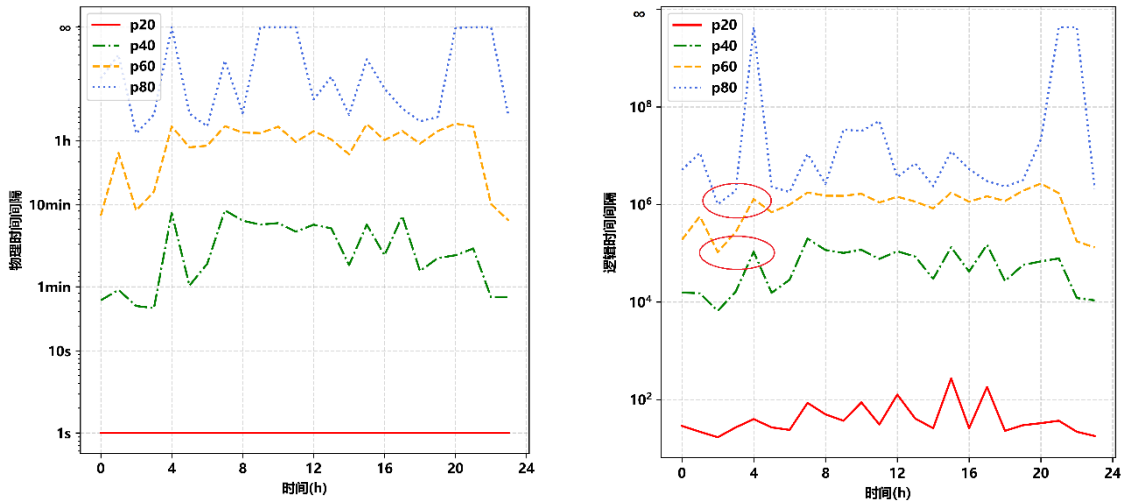
3 有接近 40%的数据块寿命位于 1 分钟到 10 分钟之间，对应逻辑间隔在 10^4~10^5 之间，而节点 1 只有不到 20%。



(a) BIPT 分布　　(b) BILT 分布

图 2.4 节点 1~4 为期 1 天的块寿命分布

另一方面，相较于图 2.3 反映出的节点写入频率特征——节点 1、4 和节点 2、3 分别在写入频率分布上更为相似，而图 2.4 则表现出节点 1、2 和节点 3、4 分别在块寿命分布上更为相似，说明块寿命和写入频率能够从不同角度对云块存储场景下数据块的冷热程度进行刻画。



(a) BIPT 阈值变化　　(b) BILT 阈值变化

图 2.5 节点 1 第 1 天的块寿命分布阈值变化

为探究现有基于块寿命的数据放置策略在云块存储场景下表现不佳的原因，以节点 1 为例，本文进一步以小时为粒度统计了全部数据块的块寿命，进行排序并获

取阈值，P20、P40、P60、P80 分别代表从小到大前 20%、40%、60%、80%的块寿命阈值。阈值变化如图 2.5 所示，其中图 2.5 (a)为物理寿命，图 2.5 (b)为逻辑寿命。可以明显看出，同一节点内负载随时间变化明显，即数据热度分布频繁变更，且存在阈值交叉的现象，如图 2.5 (b)中椭圆区域所示，其内部的曲线，在 2 点、3 点、4 点的时候，P40 和 P60、P60 和 P80 均存在很接近的情况。说明若是不能及时动态调整数据分类阈值，则会很大程度影响数据分类效果。

考虑到逻辑寿命相较物理寿命而言具有更好的通用性，以及更适用于理论分析以及方案设计，且两种块寿命指标从负载分析结果来看并无明显差异，本文后续均选择逻辑寿命作为研究对象。

2.2 针对传统 SSD 写放大问题的理论最优策略分析

本节将对现有针对于传统 SSD 的理论最优放置策略进行说明，并从理论和实际测试结果两个维度对现有策略在基于 ZNS SSD 的云块存储场景下表现不佳的原因进行分析。

2.2.1 传统 OPT 放置策略

Wang 提出了针对传统 SSD 的理想最优放置策略 FK（Future knowledge）[18]，对于给定的写入序列，先获取每个数据块的失效顺序o_i，再将其放置于第$k = \left\lceil \frac{o_i}{S} \right\rceil$个闪存擦除块中，其中 S 为擦除块的大小（是数据块的整数倍）。这样，当第 j$(1 \le j \le k)$个擦除块被写满时，里面存放着失效顺序位于$[(j-1)*S+1,\ j*S]$的数据块。且垃圾回收被触发时，必定从$k = 1$的擦除块开始依次执行，因为此时该擦除块内的数据块最先全部失效，无需进行有效数据迁移，写放大为 1。

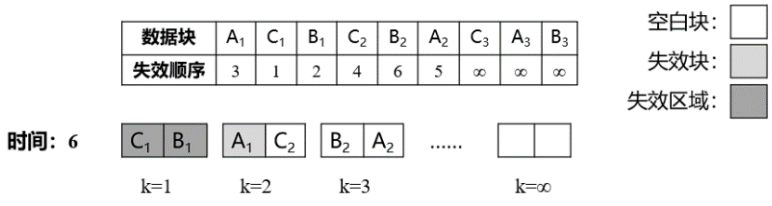


图 2.6 FK 算法示意图

以图 2.6 为例，表格中给出了写入序列“ACBC…”中各数据块的失效顺序，数

据块 A_n 代表数据块 A 第 n 次被写入，k 代表擦除块被回收的顺序。随着逻辑时间递增，数据块 A_1、A_2、B_1、B_2、C_1、C_2 依据 FK 算法被分别放入了 k 为 1、2、3 的擦除块中，当时间为 6 时，按照失效顺序，C_1、B_1、A_1 这三个数据块均已失效，即 k=1 的擦除块内数据块均已失效，此时对其进行垃圾回收则无需迁移数据，写放大为 1。

但若要直接利用 FK 算法进行理论分析，在获得全量数据写入序列的前提下，仍然无法实现，原因主要有两个：

① 获取每个数据块的失效顺序时间开销过高（请求数量 n 往往过亿，且获取块寿命加上排序的总时间复杂度为$O(n^3 logn)$）。

② ZNS SSD 能被同时写入的区域（即同时开放分区的数量）有限，无法满足海量数据按照顺序分类后所需要的大量分区类别。

2.2.2 FK 算法具体实现

为了验证 FK 算法效果，本文采用了另一种可行的分析方案，利用块寿命t_b来近似替换失效顺序，而$t_b = t_{arr}' - t_{arr}$，其中t_{arr}'代表数据块下一次写入的时间，t_{arr}代表当前到达时间，均为逻辑时间。然后，根据$type_id = \left\lceil \frac{t_b}{S_{zone}} \right\rceil$获取该数据块所属分区的类别，其中$S_{zone}$为分区大小。

为了契合 ZNS SSD 同时开放分区数量的物理限制（假设为 4），我们将$type_id$大于 4 的数据块均放置于 zone type 为 4 的区域中，案例如图 2.7 所示。

图 2.7 FK 算法应用于 ZNS SSD 示例

这是一个简化版的 ZNS SSD，同一时间能被写入的分区有且只有 4 个，每个分区内仅支持追加写，不支持原地更新，S_{zone}大小为 2。

时间 5 的时候，A_1，B_1，D_1 数据块均被写入了理想的区域，且 A_2 的写入使得前

一次写入的 A_1 失效。数据块 C_1 的理想$type_id$为 12/2=6，但因超限只能写入 4 中。此外，type 为 2 的分区因写满而转为只读状态，同时新分区被开放。

时间9的时候，type为2的只读分区内数据均已失效，我们将其称为“失效zone”，若此时垃圾回收被触发，则该分区会被优先选取，无需迁移有效数据，写放大为 1。

时间 16 的时候，type 为 1，2，3 且已写入数据的分区最终均转换成了“失效zone”，符合理想最佳的情况。但受限于同时开放分区数量的物理限制，type 4 内仍存在寿命差异较大的数据块共存于一个分区的情况，例如 C_1 和 E_1，此时 C_1 块已经失效，但 E_1 没有，若此时对该分区进行垃圾回收，依旧会产生写放大。

即从理论上来讲，有限的同时开放分区数量无法满足细粒度切分后的不同块寿命区间所需要的大量分区类别，需要对该方案进行改进。

2.2.3 测试与结果分析

本文选取了节点 1~4 采样后的负载进行测试，数据放置算法基于开源日志结构存储引擎和仿真 ZNS SSD 实现，测试环境详见 4.1 所述。参考最新研究[18]中将同时接受写入的区域数量设置为 6，本文在此基础上增加了一个区域用于避免“一次写入”（即永不失效）数据块的干扰，即同时开放分区数量设置为 7，前 1~6 存放对应$type_id$的数据块，同时将$type_id > 6$的数据块全部写入$type_id$为 6 的分区中，然后将寿命为∞的数据块全部写入$type_id$为 7 的分区中。垃圾回收算法选择 CBE（详见 3.4.1 所述）。写放大系数 WAF 通过用户写入量$data_{uw}$和 GC 写入量$data_{gw}$获得，计算方法如公式 2.1 所示。

$$WAF = \frac{data_{uw} + data_{gw}}{data_{uw}} \tag{2.1}$$

测试结果如表 2.2 所示：

表 2.2 FK 在节点 1~4 数据集下的测试结果

节点	用户写入量(GB)	GC 写入量(GB)	WAF
1	483	78	**1.161**
2	581	108	**1.186**
3	433	90	**1.208**
4	468	157	**1.335**

可见尽管利用了未来知识，但和最理想情况（WAF 为 1）仍有较大差距，节点 4 的写放大相较最理想的情况甚至高出了 33.5%。本文进一步获取了被回收分区的信息，GP（garbage proportion）分布如图 2.8 所示。

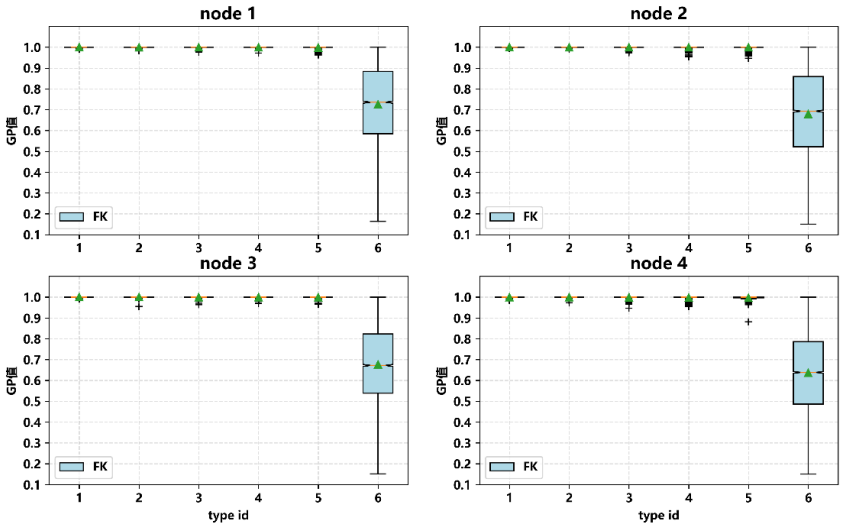


图 2.8 FK 算法回收分区的 GP 分布

从图 2.8 中可以看出，全部节点的有效数据迁移绝大部分来源于$type_id$为 6 的分区，这与理论分析一致，且其中有超过 50%回收分区的 GP 值小于 0.7，说明 FK 算法并不能很好地直接应用于 ZNS SSD。尽管利用了未来知识，但$type_id$为 6 内数据块寿命差距过大，仍会造成严重的写放大问题。迫使我们提出新的适用于 ZNS SSD 的理论最优放置策略。

2.3 适用于 ZNS SSD 的写放大理论最优策略

2.3.1 理论最优策略构思

从 2.1 的负载分析和 2.2.3 的测试结果可得，块寿命分布范围很广，ZNS SSD 有限的分区数量无法满足细粒度切分后的不同块寿命区间所需要的大量分区类别。为此我们需要探寻一种新的数据分类方式。

从宏观视角来看，块写入频率满足 zipf 分布，假设其对应关系如公式 2.2 所示：

$$f(r)=\frac{C}{r^{\alpha}}\quad 1\le r\le N,\alpha\ge 0 \tag{2.2}$$

其中C为常量，用于保证概率密度求和为1。令公式2.2中的$\alpha=1$，同时假设所有数据块都均匀分布在同一段足够长的时间段T内，对于排序为r数据块而言，其块寿命t_r为：

$$t_r=\frac{T}{f(r)}=\frac{T}{C}*r \tag{2.3}$$

从公式2.3可知，因为T和C均为常量，故块寿命和数据块的排序之间为线性相关关系。此时块寿命$t\le t_r$的数据块数量$sum_{t\le t_r}$以及数据块总量sum_N获取方式如公式2.4和2.5所述：

$$sum_{t\le t_r}=\sum_{n=1}^{r}f(n)=\sum_{n=1}^{r}\frac{C}{n}\approx \mathrm{C}*\ln(r)+\gamma\approx C*\ln(r) \tag{2.4}$$

$$sum_N=\sum_{n=1}^{N}f(n)\approx C*\ln(N) \tag{2.5}$$

其中$\gamma\approx 0.5772$为欧拉常数，当r很大时可以忽略不计。对于随机选取一个数据块i而言，其概率为：

$$p_i=\frac{1}{C*\ln(N)} \tag{2.6}$$

结合公式2.4和2.6，可得随机选择一个数据块，其寿命t≤排序为r数据块寿命t_r的概率为：

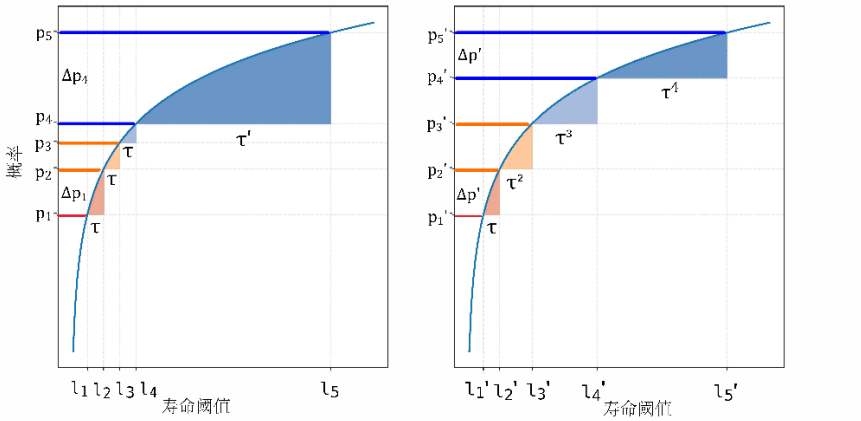
$$p_{t\le t_r}=p_i*sum_{t\le t_r}=\frac{C*\ln(r)}{C*\ln(N)}=\frac{\ln(r)}{\ln(N)} \tag{2.7}$$

由公式2.7可得，对于给定寿命阈值l_1、l_2分别为排序r_1、r_2的数据块寿命时，随机选取一个数据块的寿命t位于阈值区间的概率计算方式为：

$$p_{l_1<t\le l_2}=p_{t\le l_2}-p_{t\le l_1}=\frac{C*\ln(r_2)}{C*\ln(N)}-\frac{C*\ln(r_1)}{C*\ln(N)}=\frac{\ln(r_2/r_1)}{\ln(N)} \tag{2.8}$$

根据公式2.7和2.8可得，概率p和寿命阈值为对数关系，若想要均衡各类别分区承担的数据块写入量（即写入不同寿命区间的概率相等），避免某个类别的分区内

数据冷热差距过大导致放置效果不佳，则寿命阈值区间需要呈指数型扩充，两种分类方式对比如图 2.9 所示。其中，横坐标为寿命阈值l，纵坐标为概率p。



(a) 等阈值区间分类　　(b) 指数型扩充阈值区间分类

图 2.9 数据分类方式对比示意图

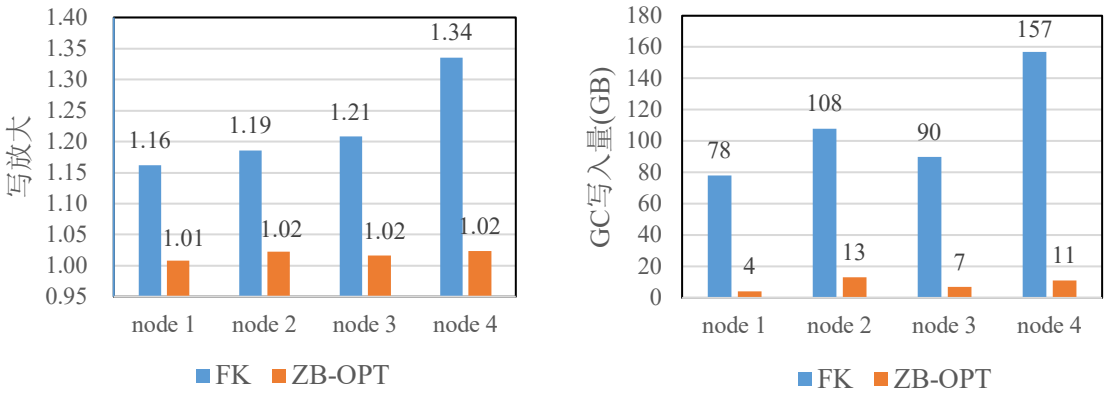
由图 2.9 (a)可得，若是参考 FK 算法，采用等距离τ的阈值区间进行分类，则会导致最后区域阈值范围τ'过大，所承担的数据块写入量（即Δp_4）远大于前面几个区域（$\Delta p_1 \sim \Delta p_3$）。若是改为采用指数级扩大的阈值区间$\tau^{\lambda}$，如图 2.9 (b)所示，则可以使各区间承担的数据块写入量相等（均为$\Delta p'$）。

2.3.2 ZB-OPT 设计与测试

基于上节的分析结论，本文提出了适用于 ZNS SSD 的理论最优放置策略 ZB-OPT，其分类方案如公式 2.9 所示，通过系数2^{λ}来实现指数型扩充寿命阈值范围的效果。

$$type\_id = \left\lfloor \frac{t_b}{S_{zone} * 2^{\lambda}} \right\rfloor, \ \lambda = 2 * (type\_id_{old} - 1) \qquad (2.9)$$

其中，$type_id_{old}$对应 FK 中的$type_id$，保证计算出的新类别总数和 FK 保持一致。同时，对于$type_{id} > 5$，即$\lambda = 8$，$t_b > 256 * S_{zone}$的数据块，全部放置于$type_id$为 6 的分区中避免超限。同样，为了避免“一次写入”数据的干扰（块寿命为∞），ZB-OPT 也将它们全部放入$type_id$为 7 的分区中。实验结果如图 2.10 所示。



(a) 写放大对比　　(b) GC 写入量对比

图 2.10 FK 和 ZB-OPT 结果对比

从图 2.10 (a)可以看到，ZB-OPT 在调整了块寿命分类阈值后，写放大有了明显降低，非常接近于 WAF 为 1 的最理想情况，相较 FK 的结果而言，从平均 1.220 下降到了 1.018，降低了 16.6%，同时，从图 2.10 (b)可得，GC 写入量从平均 108.25 GB 减少到 8.75 GB，减少了 91.1%，效果显著。

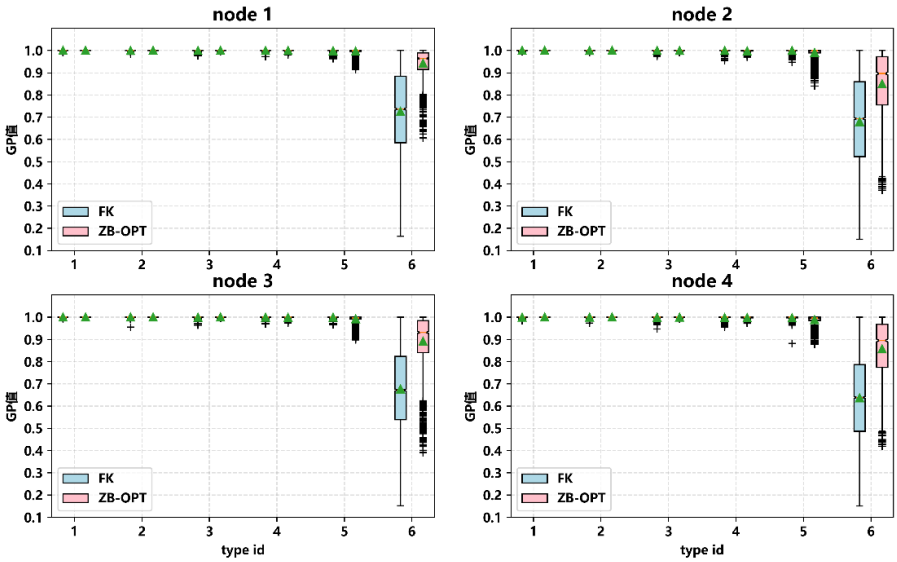


图 2.11 FK 和 ZB-OPT 回收分区的 GP 分布

同样，对每次 GC 操作进行信息统计，结果如图 2.11 所示，可以明显看到，对于测试的 4 个节点，尽管 ZB-OPT 算法因扩充了$type_id$为 2~5 的阈值范围，在垃圾回收时增加了微量来自这些分区的 GC 写，但核心源头——$type_id$为 6 的分区被回收时的 GP 平均值由 0.7 左右提升到了 0.9 左右，提高了约 30%，下四分位由 0.5 提

升到了 0.8，提高了 60%，大幅降低了整体的 GC 写入量。

2.4 对优化实际数据放置策略的启示

若想要获得理想的数据放置效果（即写放大越接近 1 越好），核心点在于尽可能的保证$t_b \leq t_{zone}$，其中t_b为块寿命，t_{zone}为分区寿命（获取方式详见 3.4.1 所述）。

从 ZB-OPT 的测试结果可得，对于块存储系统持久化层场景而言，用户写数据的块寿命整体近似服从 zipf 分布，按照指数型阶梯设置阈值能取得较好分类效果。在未来信息的帮助下，能获得准确的块寿命信息t_b，此时$t_{zone} \approx S_{zone} * 2^{\lambda}$，可以做到极大概率满足$t_b \leq t_{zone}$。

但对于实际情况而言，无法准确获取t_b，存在不可避免的误差。分类错误会增加垃圾回收次数，同时也会一定程度的增大t_{zone}。此时需要动态调整分类阈值来降低不可避免误差所带来的更多负面影响。

从 2.1 的分析结果可得，用户写的块寿命分布范围很广，且分布随时间变化明显，若想依靠历史块寿命对数据块进行冷热程度区分则需要频繁聚类以及重新训练分类器或是预测模型，开销极大。写入频率分布和块寿命类似，依靠写入频率理论上也能够实现较好的分类效果，但写入频率会受到历史惯性影响，无法很好的及时对热度改变后的数据块进行调整。

综上所述，根据负载分析结果和理论最优策略 ZB-OPT 的测试结果，所得出的关于优化实际数据放置策略的启示如下：

- 借助未来信息的帮助，利用块寿命能实现最小化写放大的数据放置效果，可以作为理论上限来衡量其他数据放置策略的表现。
- 写入频率的整体分布和块寿命类似，也能用于云块存储场景下数据块的冷热程度分类，两者均有各自的优势与不足，存在互补空间。

2.5 本章小结

本章对云块存储场景下的真实负载信息进行了分析。发现尽管通过一致性哈希等负载均衡方法，成功将用户数据的分布打散，均匀分摊到了各个存储节点，但数据

热度仍然存在明显差异，不同节点的负载访问模式不同。对写入频率的分布进行统计得出，整体近似遵循 Zipf 定律，但前 0.01%~1%和后 1%~100%数据块的写入频率差异巨大，且不同节点的数据热点分布不同。对块寿命进行分析后发现，同样近似遵循 Zipf 定律，且逻辑寿命和物理寿命在整体分布上并无明显差异，存在 10%~20%的数据块在负载统计期间内永久有效。块寿命分类阈值随时间变化明显，且存在阈值交叉现象。

测试发现针对传统 SSD 的理论最优数据放置策略 FK 在 ZNS SSD 中表现不佳，究其原因是 ZNS SSD 受到同时开放分区数量的物理限制，无法满足细粒度切分后的不同块寿命区间所需要的分区类别数量所致。然后本章提出了适用于 ZNS SSD 的理论最优放置策略 ZB-OPT，并和 FK 进行了测试对比，结果表明 ZB-OPT 显著降低了整体的垃圾回收写入量以及写放大系数。最后基于负载分析结果以及最优策略的测试结果给出了对于实际数据放置策略优化的启示。

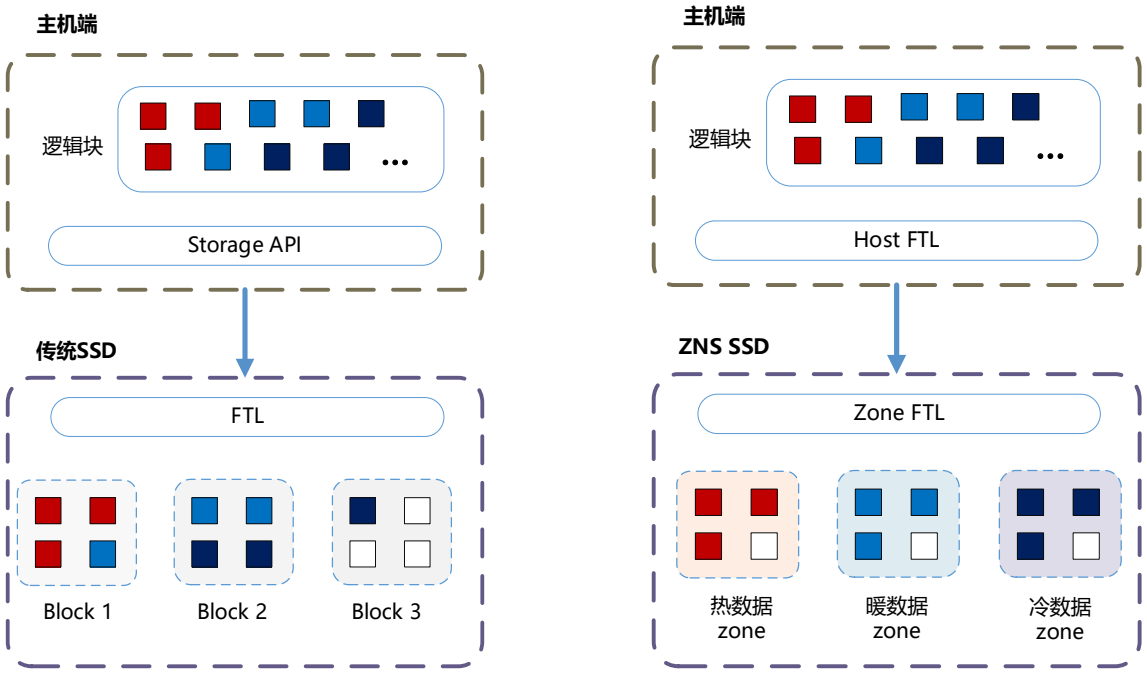
3 基于块热度的数据放置策略研究与实现

现有关于写放大优化的研究工作无法很好直接应用于基于 ZNS SSD 的云块存储系统场景，存在开销过高、适用性不佳、场景不匹配等问题，需要一种低开销且有效的冷热数据分类方案来指导数据在 ZNS SSD 内部的合理放置。本章基于第二章关于负载分析的结果，以及理论最优策略的研究与启示，提出了基于块热度的数据放置策略 BHB-DP，采用了一种轻量级的数据热度分类方案来实现冷热数据分离的放置效果，并详述了该策略的整体架构与具体实现。

3.1 总体设计

3.1.1 方案构想

传统 SSD 内部的数据放置对用户不可见，完全由 FTL 维护。但 FTL 无法获取主机端数据块的冷热程度信息，很容易将有效时间不同的数据块放置于同一区域中，频繁触发垃圾回收操作，造成写放大，影响性能。



(a) 传统 SSD 内数据放置　　(b) ZNS SSD 内数据放置

图 3.1 传统 SSD 和 ZNS SSD 的数据放置差异

如图 3.1 (a)所示，红色代表热数据块，天蓝色代表暖数据块，深蓝色代表冷数据块，传统 SSD 通常按照固定策略将逻辑块按顺序依次写入空白区域中，导致了冷热数据混合放置（如 Block 1 和 Block 2）。ZNS SSD 支持用户自定义“Host FTL”，以分区为粒度管理 SSD 的内部区域，如图 3.1 (b)所示。利用数据放置策略根据“块热度”来区分冷热数据，存放于不同的分区中，使得有效时间相近的数据块能在同一段时间内同时过期，理论上能够减少不必要的数据迁移，优化写放大问题。关于 ZNS SSD 内部分区的管理方式详见 3.3.1 所述。

冷热数据区分的方案设计需要依赖写入频率和块寿命这两个核心指标，两种指标的特点如下所述：

（1） 写入频率

写入频率包括用户写频率和 GC 写频率，用户写频率的增加说明近期该数据块正在变热，而 GC 写频率的增加则说明对该数据块冷热程度判断错误，该数据块正在变冷。对于云块存储场景而言，从 2.1 的负载分析结果可得，写入频率服从 zipf 分布，且写入频率高的数据块和写入频率低的数据块写入次数差异很大，理论上通过计算用户写频率和 GC 写频率的数值差异即可简单有效区分热数据和冷数据，同时还可以通过分类后的分区寿命获取块分类的寿命阈值。但写入频率会受到历史信息惯性的干扰，无法很好地适应负载变化。

（2） 块寿命

根据 2.3.2 的测试结果可得，若是能提前获取准确的块寿命信息，利用 ZB-OPT 策略借助未来信息可以实现最小化写放大的效果。但对于现实场景而言，为了降低块寿命估计错误所导致的分类错误开销，需要动态调整数据块分类阈值。尽管现有研究表明，块寿命具有一定的适应负载变化的能力，可以利用上一次块寿命来推测下一次块寿命，但从 2.1 的负载分析结果可得，不同数据块寿命差异很大，单纯依靠经验设定阈值或是根据块平均寿命来设定阈值效果不佳，无法实现合理的分类效果，且同一节点同一天内负载频繁变化，若想获得良好的分类效果则需要及时进行聚类操作更新阈值，开销极大。

综上所述，可以得出以下两点结论：

① 写入频率可以用于低开销实现冷热数据块的分类，并间接获取分类后的寿命阈值，但对于单个数据块而言，无法及时感知其热度变化。

② 块寿命具有一定的适应负载变化的能力，但直接使用块寿命获取分类阈值效果不佳或开销太大。

针对现有研究中直接通过预测块寿命来来进行冷热数据分类的方案无法兼顾准确性和额外开销的问题，本文提出一种基于“块热度”的新数据放置策略 BHB-DP，有机结合了写入频率和块寿命各自优势，并规避其不足。

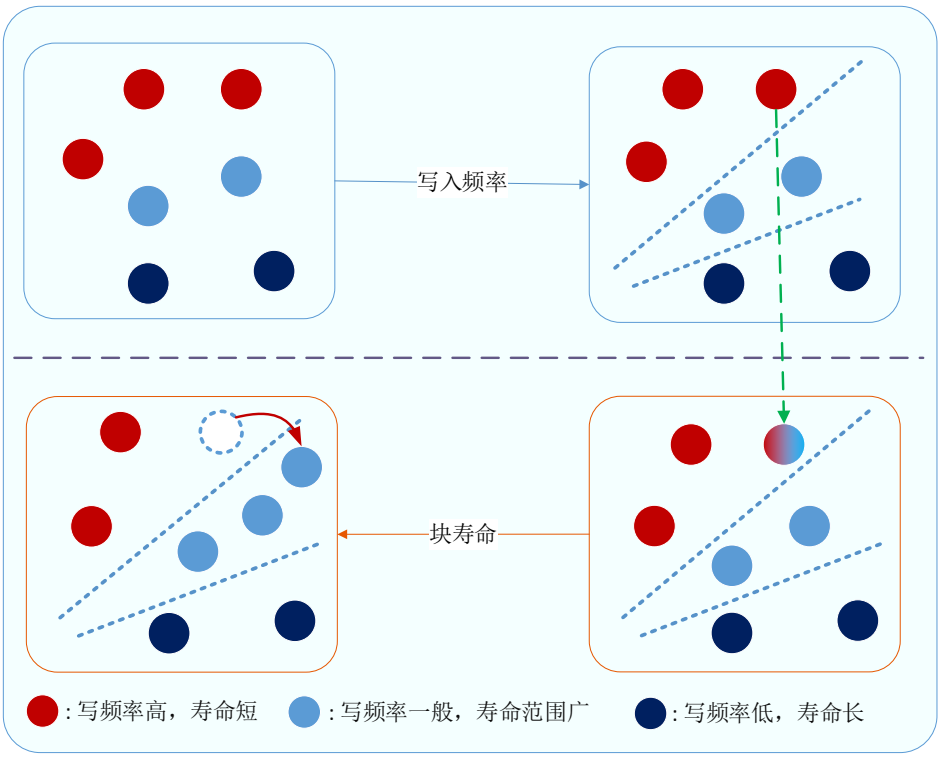


图 3.2 写入频率和块寿命的作用

如图 3.2 所示，红色代表热数据，写入频率高（用户写频率＞GC 写频率），块寿命往往也很短；天蓝色代表暖数据，写入频率一般（用户写频率和 GC 写频率相差不大），块寿命范围很广，往往是具有一定周期性，或是热数据变冷之后的数据块；深蓝色代表冷数据块，写入频率低且块寿命很长。

利用写入频率可以低开销的对这三类数据进行初步划分，但对于单独的数据块而言，其热度可能会随时发生变化，这时依靠块寿命以及判定阈值对其所属类别进行及时调整，则能够很好地弥补写入频率对数据块热度变化感知不及时的缺陷。

3.1.2 架构设计

基于块热度的数据放置策略整体方案架构如图 3.3 所示，主要包括四个部分：元数据管理模块、数据放置模块、设备管理和垃圾回收模块模块。

① **元数据管理模块**。负责记录并维护逻辑块到物理块之间的映射关系表。

② **数据放置模块**。负责利用数据放置策略 BHB-DP 获取数据块热度值bh，并根据热度值将数据块放置于对应的分区中以达到区分冷热数据的目的。

③ **设备管理模块**。负责管理和维护分区的状态信息，利用 Zoned Storage 命令集对 ZNS SSD 设备进行相关操作，例如读写数据、开启分区、重置分区等。

④ **垃圾回收模块**。负责对存储系统内的垃圾占比进行监控，一旦超过阈值则触发垃圾回收操作。首先根据垃圾回收算法选择待回收的分区，然后将其中仍然有效的数据块写入新的分区中，同时修改该数据块的映射关系以及热度信息，最后对回收完成的分区进行重置操作。

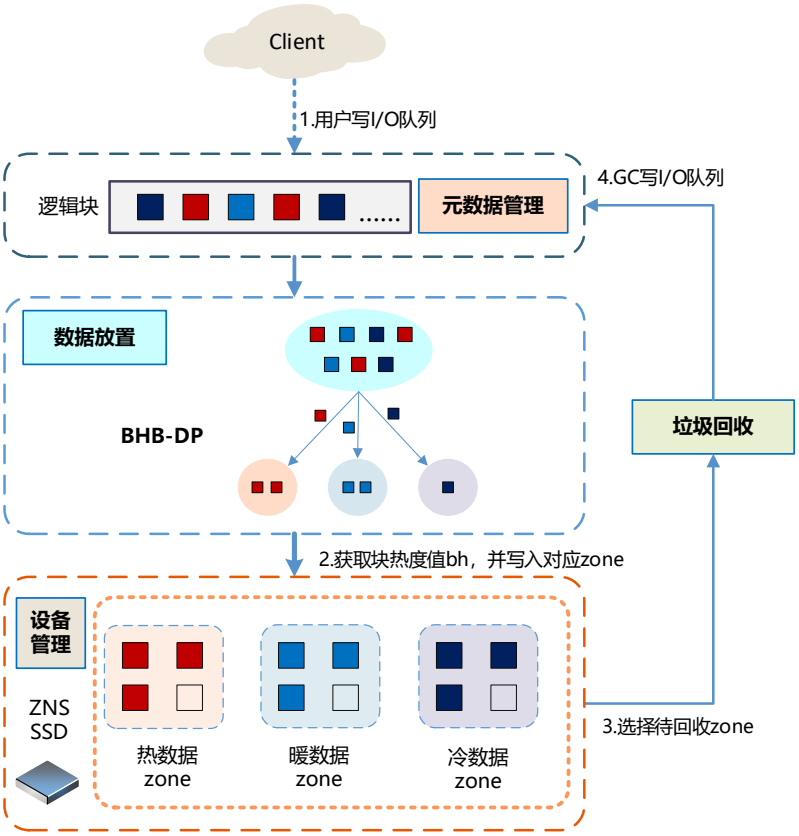


图 3.3 BHB-DP 整体架构

整体工作流程如下：

① 接收用户写 I/O 队列，获取读写请求，并进行 4KB 对齐，调用追加写接口进行数据写入，初始化或更新元数据信息。

② 通过 BHB-DP 获取块热度值bh，并写入对应的分区中。

③ 当触发垃圾回收操作时，遍历全部已写满或关闭的分区，根据垃圾回收算法从中选择待回收的分区。

④ 将回收分区中仍然有效的数据块加入 GC 写 I/O 队列中重新写入，同时更新其元数据信息。

3.2 数据放置模块设计

3.2.1 块热度获取

BHB-DP 通过两阶段（计算加策略）的方式确定一个数据块的块热度值bh（block heat）。相关属性字段说明如表 3.1 所示：

表 3.1 计算块热度值所需字段及其含义

属性名	代表含义
bh_{pre}	初始块热度值
bh	最终块热度值
$freq_{gw}$	GC 写频率
$freq_{uw}$	用户写频率
gt_{now}	当前全局逻辑时间戳
gt_{pre}	数据块上一次写入时的全局逻辑时间戳
gt_{fulled}	分区被写满时的逻辑时间戳

首先根据该数据块的用户写频率和 GC 写频率获取初始块热度值，为了对应 ZB-OPT 的冷热数据分类方式（$type_id$越小的分区存放的数据越热），本文将热度值设置为和用户写频率呈负相关，和 GC 写频率呈正相关，即块热度值越小，表示该数据越热，反之越冷。具体获取方式如公式 3.1 所述：

$$bh_{pre} = freq_{gw} - freq_{uw} + C \tag{3.1}$$

其中C为默认热度值，通常取范围中间值，以便后续调整冷热分类。为匹配 ZNS SSD 有限的同时开放分区数量（假设为 N），初始块热度值的取值范围$bh_{pre} \in [1,\ N]$。若计算所得bh_{pre}超出阈值，则用边界值 1 或 N 代替。

然后，对于$bh_{pre} < C$的数据块而言（即热数据和暖数据），需要通过块寿命进行热度确认，为了获得更好的通用性，块寿命类别选择逻辑寿命。数据块的最终块热度值bh获取方式如公式 3.2 所述：

$$bh = \begin{cases} bh_{pre}\ , BILT \le l_{type\_id} \\ bh_{pre} + 1\ , BILT > l_{type\_id} \end{cases} \tag{3.2}$$

当$BILT > l_{type_id}$时，说明该数据块已经变冷，需要增大块热度值并将其放置于更冷的分区中，否则保持不变。其中 BILT 为数据块的上一次逻辑寿命，通过公式 3.3 获取。l_{type_id}为判定阈值，其中$type_id = bh_{pre}$，代表分区的类型。判定阈值的获取方式为最近 M 个该类型分区的寿命值T_{type_id}取平均值，如公式 3.4 所述。

$$BILT = gt_{now} - gt_{pre} \tag{3.3}$$

$$l_{type\_id} = \frac{\sum_1^M T_{type\_id}}{M} \tag{3.4}$$

T_{type_id}具体指该类别分区从被写满到被回收时的时间间隔，获取方式如公式 3.5 所述：

$$T_{type\_id} = gt_{now} - gt_{fulled} \tag{3.5}$$

BHB-DP 两阶段获取块热度值的方式主要有以下三大优势：

（1）有机结合了写入频率和块寿命这两项核心指标的优势，并规避了其不足。先利用写入频率可以在短期内有效的将热数据、暖数据、冷数据分别分类放置到 N 类分区中，然后利用块寿命对其热度值进行修正，能很好地避免因为单独的数据块热度下降而导致对其分类错误的情况发生。

（2）获取方式简单，计算和存储开销都较小。块热度值的获取并不需要对数据块进行全量统计、排序、频繁聚类等操作，或是训练分类器以及预测模型，并不会对存储系统造成高额的计算和存储开销。

（3）能更好利用 ZNS SSD 的特点。根据 2.4 的启示可知，理想情况下块寿命阈值呈指数型扩大能更加契合 ZNS SSD 有限的同时开放分区的数量限制，但手动设置固定阈值不适用于负载频繁变化的云块存储场景。考虑到写入频率和块寿命相似的分布特征，利用写入频率进行分类，再根据分区寿命来反推块寿命分类阈值，则能够低开销的间接获取到相对合理的块寿命判定阈值。

3.2.2 块热度维护

云块存储系统持久化层主要为多租户混合的 I/O 请求模式，表现为 I/O 负载频繁变化，即数据热度分布并非一成不变，具体原因主要为下述两方面：

- 单租户在不同时间运行的负载可能不同，使得该用户的数据热度发生变化，进而影响系统整体的数据热度分布。
- 即使单租户运行的是稳定负载，不同租户间的 I/O 负载也会存在差异，导致云块存储系统整体的数据热度分布不断变化。

为了保证数据块冷热分离的效果，需要及时对块热度值进行维护和更新。更新操作在每次数据块被写入时进行，而写入操作包括用户写和 GC 写两类，对块热度值会造成不同影响——

（1）用户写操作。从元数据表中获取该数据块的元数据信息，若未找到，则说明数据块为首次写入，初始化其用户写和 GC 写次数，根据 BHB-DP 获取默认块热度值，并修改该数据块写入时间戳记录值。若找到，则直接将该数据块的用户写次数+1，说明该数据块正在变热，然后获取上一次块寿命以及通过 BHB-DP 获取新的块热度值，最后更新写入时间戳记录。

（2）GC 写操作。说明该数据块正在变冷。先从元数据表中获取该数据块的元数据信息，为避免历史写入频率惯性的影响，需要判定是否需要刷新——获取块寿命值$BILT$，若超过了刷新阈值R_1，则重置该数据块的历史写入频率信息，否则跳过刷新。然后将 GC 写次数+1，同时，为了更快地将“一次写入”的数据块放置于用于存

放最冷数据块的分区中，若$BILT$超过阈值R_2（$R_2 > R_1$），则在重置完写入频率信息后，直接将其 GC 写次数修改为 N-C，此时通过 BHB-DP 计算所得该数据块的热度值为$bh = N$，将会被直接放入最冷的分区中。

此外，系统中还会维护 N 个大小为 M 的 FIFO 队列，用于分别记录最近 M 个类型为 1~N 的分区寿命，进而获取其平均值l_{type_id}作为该类型分区的寿命判定阈值，其中$type_id \in [1, N]$。

3.2.3 具体实现

算法 3-1 为 BHB-DP 的具体实现，先获取块寿命，以及判断该次写操作是用户写还是 GC 写，分别执行对应函数进行数据块的信息更新（第 1~6 行），然后获取初始块热度值并避免超界（第 7~13 行），接着利用块寿命和判定阈值进行热度值确认（第 14~18 行），最后返回块热度值计算结果。

算法 3-1 基于块热度的数据放置策略 BHB-DP

```
全局变量:
    global_timestamp
输入:
    用户写频率 freq_uw, GC 写频率 freq_gw, 上一次写入时间戳 last_timestamp,
    默认热度值 C, 是否是用户写 is_uw, 同时开放分区最大数量 N
输出:
    块热度值 block_heat
1:  BILT ← global_timestamp - last_timestamp   //获取块寿命
2:  if is_uw 为真 then
3:      UserWrite()
4:  else
5:      GcRewrite()
6:  end if
7:  block_heat ← freq_gw - freq_uw + C
8:  if block_heat < 0 then
9:      block_heat ← 0
```

```
10:  end if
11:  if block_heat > N then
12:     block_heat ← N
13:  end if
14:  if block_heat < C then
15:     if BILT > 类别为 block_heat 的分区寿命平均值 then
16:        block_heat ← block_heat + 1
17:     end if
18:  end if
```

函数 1 *UserWrite()* 负责实现用户写操作时对数据块的信息更新，若数据块为首次写入，则初始化相关属性，否则直接更新写入频率，然后更新写入时间戳以及全局时间戳。

函数 1 *UserWrite():*

```
全局变量:
    global_timestamp
输入:
    逻辑地址 block_addr, 映射表 l2p_map, 用户写频率 freq_uw, GC 写频率 freq_gw,
    上一次写入时间戳 last_timestamp
输出:
    None
1:  if block_addr 在 l2p_map 内 then
2:     freq_uw ← freq_uw + 1
3:  else
4:     freq_uw ← 0
5:     freq_gw ← 0
6:  end if
7:  last_timestamp ← global_timestamp
8:  global_timestamp ← global_timestamp + 1
```

函数 2 *GcRewrite()* 负责实现 GC 写操作时对数据块的信息更新，首先根据块寿命以及阈值来判定是否需要刷新该数据块的信息，然后更新其 GC 写频率。

函数 2 *GcRewrite():*

输入:

块寿命 BILT, 用户写频率 freq_uw, GC 写频率 freq_gw, 刷新阈值 R1,

极冷数据阈值 R2, 默认热度值 C, 同时开放分区最大数量 N

输出:

None

```
1:  if BILT > R1 then
2:    freq_uw ← 0
3:    freq_gw ← 1
4:    if BILT > R2 then
5:      freq_gw ← N - C
6:    end if
7:  else
8:    freq_gw ← freq_gw + 1
9:  end if
```

3.3 设备管理模块和元数据管理模块设计

3.3.1 ZNS SSD 设备管理

不同于传统 SSD 以 page 为粒度（通常为 4KB）进行数据读取写，block 为粒度（通常包括 512 个 page)进行数据擦除操作，ZNS SSD 以分区为粒度进行数据管理。下面对分区以及分区的管理方式进行详细介绍。

（1） zone 介绍

ZNS SSD 将 LBA（logical block address）分成了很多个独立的分区，并且保证了分区和物理介质的对齐，以简化主机端和 SSD 之间的协同管理，如图 3.4 所示。每个分区内只能顺序写，可以随机读。擦除操作以整个分区为粒度。每个分区需要维护的信息主要有 4 种：

① **ZSLBA**（zone start LBA），表示每个分区的开始地址。

② **ZCAP**（zone capacity），表示每个分区的容量。

③ **WP**（write pointer），写指针，表示偏移量，用以保证分区内的顺序写操作。写指针以前的区域只能读取，不能修改。

④ **Status**，当前分区的状态。

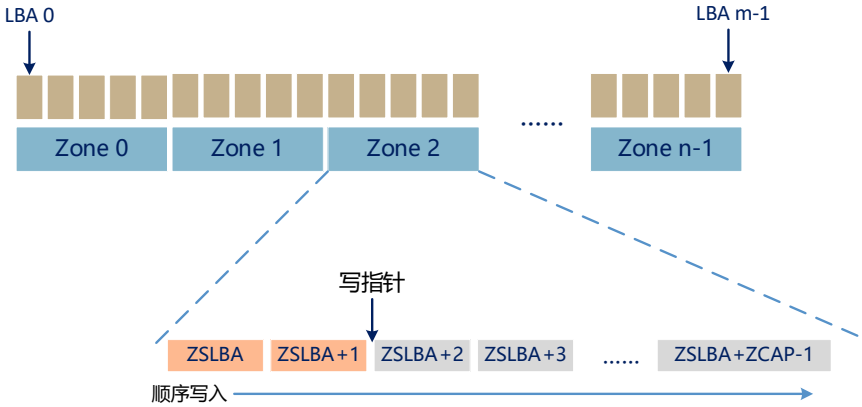


图 3.4 ZNS SSD 数据管理方式

（2） 数据写入方式

ZNS 命令集提供了两种数据写入指令，Write 和 Append。

① **Write**。负责将数据写入到指定位置，需要明确给出 LBA 地址，若写入地址不合理则会报错。此时，ZNS SSD 内为了维护顺序写约束，写入队列深度（Queue Depth，QD）只能为 1，但这种情况将会大大浪费 SSD 内部的并行性和降低系统的 CPU 效率。

② **Append**。负责批量写入，只需指定分区编号，写入数据的长度以及数据本身，然后会返回写入后的实际地址。此时 QD 最大值为分区内剩余 LBA 数量。尽管实际写入的顺序不能保证，但利用该指令写入数据能大幅提高写入效率。

（3） zone 管理指令

ZNS 命令集中提供的针对分区的操作主要包括以下四种：

① **RESET ZONE WRITE POINTER**。用于清空分区内的全部数据并重置写指针到起始位置。执行之后该分区内的数据将无法访问。

② **OPEN ZONE**。用于显式地打开一个分区，持续提供数据写入服务，直到该分区完全写满或提前使用 CLOSE ZONE 命令关闭该区域。对于 ZNS SSD

而言，同时处于 open 状态的分区存在数量上限。

③ **CLOSE ZONE。**用于显示关闭通过 OPEN 指令开放的分区，暂时释放其所占用的写入资源。

④ **FINISH ZONE。**用于将分区内的写指针移动到该分区的末尾，从而防止对该分区进行进一步的写入操作，直到该分区被重置。

（4） zone 状态转换

分区的主要状态共有 7 种，其转换机制如图 3.5 所示：

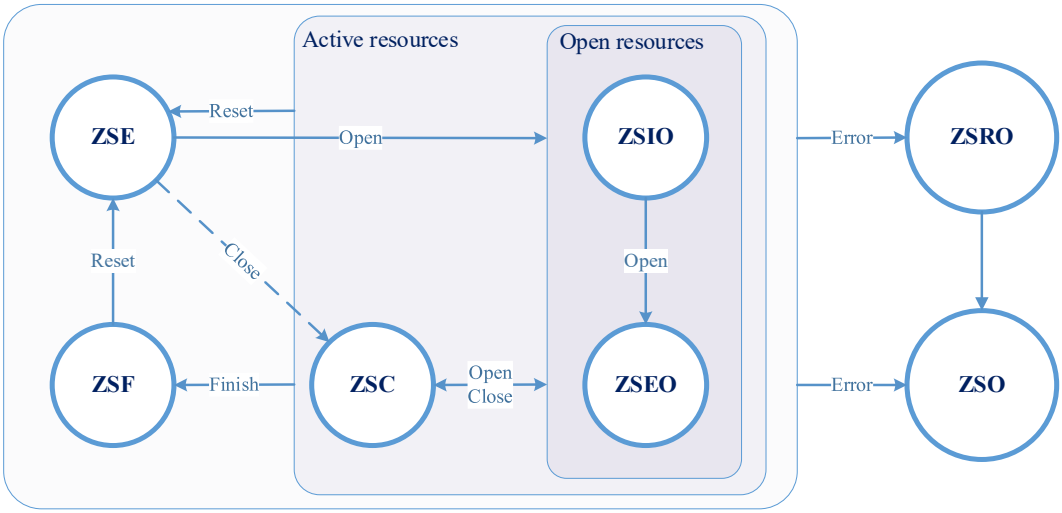


图 3.5 分区状态转换机

① **ZSE**（Zone State Empty）。表示该分区为空，可以被激活用以写入新数据。

② **ZSF**（Zone State Full）。表示该分区已写满，无法继续写入，只能提供读取服务。

③ **ZSC**（Zone State Closed）。表示该分区被关闭，无法进行读写操作。受制于 ZNS SSD 能同时开启的分区上限，若当前同时开放写入的分区达到了上限，且想要将数据写入新的分区中，则需要关闭处于开放状态的分区。

④ **ZSIO**（Zone State Implicitly Opened）。表示该分区正在写入数据，此时无法进行新的读写操作，以保证数据的准确性和可靠性。

⑤ **ZSEO**（Zone State Explicitly Opened）。表示该分区处于正常开放状态，可以写入数据。

⑥ **ZSRO**（Zone State Read Only）。表示该分区的数据被冻结，只能读取，通常

用于异常情况时的数据恢复。

⑦ **ZSO**（Zone State Offline）。表示无法读取或写入，该分区已达到擦除次数上限或出故障永远无法使用。

其中，ZSC、ZSIO、ZSEO 属于被激活的资源，正在或准备提供写服务。ZSIO 和 ZSEO 属于开放资源，正在执行或可以提供写服务。

处于 ZSE、ZSC 和 ZSIO 状态的分区可以通过 OPEN 指令显示转换为 ZSEO，开始或继续写入新数据，反之，处于 ZSIO 和 ZSEO 状态的分区也可以通过 CLOSE ZONE 指令将分区状态转换为 ZSC，释放写入资源。写入操作会将 ZSEO 状态的分区自动切换为 ZSIO 状态，写满之后则会转换为 ZSF 状态。ZSF 状态的分区可以通过 RESET 指令重置为 ZSE 状态用以重新提供新的数据写入空间。

3.3.2 设备管理具体实现

对于分区的信息管理与维护具体实现的核心数据结构如图 3.6 所示。

```
class Zone {
public:
    Zone(uint64_t id, int type_id, uint64_t timestamp, int zone_size);   //初始化
    uint64_t Append(uint64_t block_addr);   //Append 指令，返回写入后的物理地址
    void Invalidate(int i);   //将第 i 个数据块标记为无效
    void Finish(uint64_t timestamp);   //将分区状态切换为 ZSF，并记录当前时间
    bool IsFull();   //判断当前分区状态是否为 Full
    char *GetBlockData(int i);   //读取第 i 个数据块
    double GetGp() const;   //获取当前分区内无效数据占比
private:
    uint64_t zone_id = 0;
    uint64_t next_offset = 0;   //写指针
    int status = 0;   //状态机
    int zone_size = 0;
    uint64_t finished_time = 0;   //该分区被写满的时间
    };
```

图 3.6 Zone 数据结构

其中，初始化分区时需要的 id 用于标识每一个被激活的分区，全局唯一，不断递增。因为 ZNS SSD 内部严格遵循顺序写的约束，故可以直接通过 zone_id 和分区本身的容量大小计算出 ZSLBA 的值，反之，通过数据块的物理地址也能反推出其所在分区的信息以及该数据块在分区中的位置。type_id 用于表示分区的类别，其最大值对应同时开放分区的数量上限。

存储系统管理模块 Manager 实现的数据结构核心部分如图 3.7 所示。负责管理和维护整个存储引擎，主要包括维护全局逻辑时间戳（随每次用户写操作递增）、进行关于分区的操作（开启、关闭、重置）、读写数据等等。

```
class Manager {
public:
   static uint64_t global_timestamp;   //全局逻辑时间戳
   Manager(int limit_open_zones);   //初始化
   void Append(const void *buf, uint64_t addr, uint64_t bit);   //用户追加写操作
   bool GcAppend(const void *buf, uint64_t block_addr, uint64_t old_phy_addr);   //GC 写操作
   void Read(void *buf, uint64_t block_addr);   //读取数据
   void OpenNewZone(int id);   //开启新分区
   void CloseZone(int id);   //关闭分区
   void RemoveZone(int id);   //重置分区并移除该 id
   void CollectZone(int id);   //搬移分区内的有效数据
   Zone ReadZone(int id);   //读取分区信息
private:
   std::unordered_map<uint64_t, std::shared_ptr<Zone>> zone_map;   //分区管理表
   std::shared_ptr<Zone> findZone(uint64_t phy_addr);   //通过物理地址查询所在分区
   std::unique_ptr<FIFO> zone_age;   //各类别分区近期寿命
   std::unique_ptr<L2PMap> l2p_map;   //映射表
   std::unique_ptr<Placement> placement;   //数据放置算法接口
   uint64_t cur_zone_id = 0;   //单调递增的 zone id，全局唯一
   };
```

图 3.7 Manager 数据结构

3.3.3 元数据管理设计与实现

ZNS SSD 的使用需要主机端实现数据映射管理。传统 SSD 内 FTL 管理数据映射表（Logical-to-Physical，L2P）的方式主要有“页级映射”、“块级映射”和“混合映射”三种，其中页级映射将逻辑块（通常为 4KB）映射到物理块（通常也为 4KB）上，进行最细粒度的数据管理，性能最佳。块级映射将逻辑擦除块（通常为 2MB）映射到物理擦除块（通常也为 2MB），所需存储空间较页级映射更少，但对于小尺寸数据的读写表现较差，常用于 U 盘的 FTL 中。混合映射是页级映射和块级映射的一种混合方式，用于平衡性能和开销，但映射表管理方式复杂，维护开销较大。考虑到主机端丰富的内存资源，本文采用“页级映射”以获取最佳性能表现。具体实现如图 3.8 所示：

```
class L2PMap
{
public:
    L2PMap();   //初始化
    uint64_t Query(uint64_t block_addr);   //查询
    void Update(uint64_t block_addr, uint64_t phy_addr);   //更新
private:
    std::unordered_map<uint64_t, uint64_t> index_map;
};
```

图 3.8 映射表数据结构

其中，L2PMap 利用 index_map 存放每个逻辑块和物理地址的映射关系。通过 *Query*(uint64_t block_addr)函数可以查询到 block_addr 所对应的物理地址，若不存在则返回 2^{64}-1。通过 *Update*(uint64_t blockAddr, uint64_t phy_addr)函数可以添加新的映射关系，或更新 block_addr 所对应的新物理地址 phy_addr。

3.4 垃圾回收模块设计

3.4.1 垃圾回收算法设计

成本-效益算法（Cost-Benefit，CB）同时考虑了待回收分区的无效数据占比以及

存在时间，理论上比 greedy 算法能实现更好的垃圾回收效果。CB 算法通过计算全部写满状态分区的cb值并排序，然后选择cb最大的分区进行回收。cb值的计算如公式 3.6 所示：

$$cb = \frac{GP}{2 * (1 - GP)} * f(t) \tag{3.6}$$

其中，GP 为回收分区内的无效数据占比，GP 越高说明需要迁移的有效数据量越少，即收益越高。$f(t)$为关于分区寿命t_{zone}的函数。但现有的 CB 算法存在三个问题：

（1） t_{zone}的获取不合理。根据现有研究[11][26]中的实现方式，t_{zone}需要根据分区内最近一个失效的数据块来获取，这会造成明显的误判问题，例如对于一个冷数据块较多的分区，对其中任何一个有效数据块的重新写入都会重新计算该分区的寿命值，使得该分区的回收时间被大幅延后，此时反而会回收到其他具有更多即将过期的热数据的分区，从而导致误判。

（2） 存在负收益现象。对于海量数据的块存储系统而言，分区存在的寿命可能会很长，当t_{zone}很大时，即使该分区的 GP 值小于全局垃圾回收被触发的阈值，还是可能会因计算所得的cb值较高而被回收，但此时回收该分区对整体系统而言是负收益。

（3） $f(t)$的最优设置未知。现有研究并没有对$f(t)$如何设置才能达到最佳效果这一问题进行过讨论，一些研究[11][36]设置为$f(t) = t_{zone}$，但也有研究[18]中设置为$f(t) = \sqrt{t_{zone}}$，具体如何设置最佳并无定论。

为此，本文采用优化后的成本-效益算法（cost-benefit-enhanced，CBE）以解决上述问题。

首先，CBE 利用分区被写满的时间来获取t_{zone}的值。这样既可以保证热数据大概率在该分区被回收之前全部失效，又能避免上述明显的误判问题，将仅剩冷数据的分区尽早回收。

同时，CBE 加入了对于被回收分区的 GP 阈值判定，跳过 GP 值小于全局垃圾回收阈值的分区以避免负收益现象。

此外，对于问题（3），本文以节点 1 的负载数据为例进行了一组对比实验，参测放置策略为 BHB-DP 和 ZB-OPT，分别将$f(t)$设置为$f(t)=t_{zone}$，$f(t)=\sqrt{t_{zone}}$，$f(t)=\ln t_{zone}$。实验结果如表 3.2 所示：

表 3.2 不同$f(t)$设置下的结果

放置策略	垃圾回收写入量（GB）		
	$f(t)=t_{zone}$	$f(t)=\sqrt{t_{zone}}$	$f(t)=\ln t_{zone}$
BHB-DP	**123.4**	167.3	357.8
ZB-OPT	**3.7**	3.9	6.9

由结果可得，$f(t)=t_{zone}$效果最佳，其次为$f(t)=\sqrt{t_{zone}}$，而$f(t)=\ln t_{zone}$的效果则要明显差于前两者，一个合理的推测是$\ln t_{zone}$会使得分区之间寿命差异的权重大幅减小，导致存在大量即将过期数据的分区被提前回收，从而造成了更多的垃圾回收写入数据量。

故 CBE 将cb值计算公式 3.6 中的$f(t)$获取方式设置为$f(t)=t_{zone}$。

3.4.2 具体实现

CBE 算法的具体实现如算法 3-2 所示，先遍历存储系统中正在使用的全部分区信息，从中挑选出状态为 Full 的分区并获取该分区的无效数据占比 gp，若 gp 小于全局垃圾回收触发阈值 GP，则跳过该分区。然后计算回收该分区的收益值cb并加入候选列表，最后从中选择出cb值最高的分区并返回 zone_id。

算法 3-2 优化后的成本-效益算法（CBE）

输入:

全部分区信息管理表 zone_map, 待回收分区列表 CL, 全局垃圾回收阈值 GP

输出:

待回收分区的 zone_id

```
for zone_map 中的每个分区信息 do
  if 分区状态为 Full then
    获取分区无效数据占比 gp
    if gp > GP then
```

```
5:            获取分区寿命 zone_age
6:            if gp 等于 1 then
7:              cb ← 无穷大
8:            else
9:              cb ← gp / (2 * (1 - gp)) * zone_age
10:           end if
11:           获取 zone_id, 并将其和 cb 一起加入 CL
12:         end if
13:       end if
14:   end for
15:   获取 CL 中 cb 值最大的 zone_id
```

3.5 本章小结

本章通过理论分析，设计并提出了基于块热度的数据放置策略 BHB-DP，介绍了整体架构以及执行流程，包括数据放置模块、设备管理模块、元数据管理模块以及垃圾回收模块，然后详细说明了各模块的内容及其具体实现。其中，BHB-DP 根据写入频率和块寿命，通过两阶段的方式，先利用写入频率计算初始块热度值，再通过块寿命和判定阈值获取最终块热度值，然后根据块热度值对数据块进行冷热分类，将块热度值相同的数据写入到相同的分区中。BHB-DP 同时利用了写入频率和块寿命各自优势，并规避其不足，达成了低开销实现冷热数据分类的目的。此外，垃圾回收模块采用了 CBE 作为垃圾回收算法，能有效解决 GC 过程中的误判问题，并避免负收益现象。

4 实验与评估

4.1 实验环境

4.1.1 实验配置和数据选取

本文所提出的新方案 ZB-OPT 和 BHB-DP 均基于开源的日志结构存储引擎 [1]开发，同时，还在其基础之上拓展实现了第 3 章所述的元数据管理模块、设备管理模块和垃圾回收模块。此外，本文还实现了一个 I/O 生成器，用于将负载记录中的 I/O 信息转换为 I/O 结构体输入至存储引擎中。开发语言为 C++，通过 cmake 3.23.2 和 g++（GCC）8.3.1 进行编译。实验所采用的设备信息如表 4.1 所示。

表 4.1 实验环境参数

配置名称	具体参数
CPU	AMD Ryzen 5 5600X
DRAM	DDR4 32GB
SSD	NVME 2TB
OS	centOS 7
Kernel	Linux 6.2.10

测试数据集为 2.1.1 中所述的来自 10 个存储节点为期 6 天的真实场景的 IO 记录，并进行了 1/10 采样。

表 4.2 ZNS SSD 仿真参数

设置名称	参数值
Zone Size	32MB
Open Zone Limit	7
Total Size	100GB
Page Size	4KB

硬件层面采用 FEMU[53]对 ZNS SSD 进行仿真，模拟对 ZNS SSD 内部的分区进

[1] https://github.com/fallfish/sepbit

行开放、关闭、重置，以及数据读写等操作。FEMU 是一个基于 QEMU 的闪存模拟器，用于促进未来的全栈软件/硬件 SSD 研究，具有良好的扩展性。本文实验中通过 FEMU 设置选项中的“ZNS mode”来模拟 ZNS SSD 设备。参数设置如表 4.2 所示。

4.2 对比方案和评估指标

4.2.1 对比方案

本文实验所选取的全部方案具体介绍如下：

（1） Normal：不做任何冷热数据区分的常规策略，仅区分用户写和 GC 写，用于模拟传统 SSD 的顺序写入模式。其中，用户写入的数据块依照写入顺序依次放置于类别为 1~6 的分区中，GC 写入的数据块则全部放置于类别为 7 的分区中。

（2） Warcip[25]：通过 k-means 聚类算法，以最小化块寿命的方差为目标对写入数据块进行分类，其中块寿命选择了物理时间，同时借助动态分裂与合并的机制（dynamic split-and-merge，D-SAM）实现数据块分类的动态调整。考虑到该方案仅关注用户写入的数据块，实验中为其配置了类别为 1~6 的分区用于用户写数据块的类别区分，GC 写入的数据块全部放置于类别 7 的分区中。

（3） Sepbit[18]：利用上一次的块寿命推测该数据块未来可能的有效时长，按照固定的策略对数据块进行分类，其中块寿命为逻辑时间。该方案将用户写入的数据块分为冷热两类，通过最近n个数据块的平均寿命l进行区分，分别放置于类别为 1（热块）和类别为 2（冷块）的分区中。考虑到被 GC 后的冷数据有效期更长的概率更大，且具有更大的块寿命范围，对于 GC 写入块，该方案将来自于类别 1 的数据块直接放置于类别 3 中，而来自于除 1 以外其他类别分区的数据块根据分类标准分别放置于类别 4~7 的分区中，各分类标准为$[0,4l)$，$[4l,8l)$，$[8l,16l)$，$[16l,+\infty)$。

（4） BHB-DP_pre：仅利用根据用户写频率和 GC 写频率计算所得的初始块热度值来进行数据分类，分别放入类别 1~7 的分区中。其中，默认热度值 C 设置为 5。

（5） BHB-DP：同时结合了写入频率和块寿命各自的优势并规避其不足。在 BHB-DP_pre 的基础上增加了利用块寿命以及判定阈值对块热度值进行确认和刷新的操作，默认热度值 C 同样设置为 5。

（6） FK[18]：基于未来信息，将数据块分别放置于相等寿命阈值区间的类别 1~6 的分区中，对于寿命超过寿命阈值区间的数据块，也全部放置于类别为 6 的分区中。此外，为了避免“一次写入”数据块的干扰，即寿命为+∞，将其全部放置于类别为 7 的分区中。

（7） ZB-OPT：和 FK 同样基于未来信息。采用 2.3.2 所述的新数据分类策略，其他设置和 FK 相同。

Warcip 和 Sepbit 仅考虑了块寿命，分别利用了物理寿命 BIPT 和逻辑寿命 BILT，BHB-DP_pre 仅考虑了写入频率，对于现有数据放置策略工作而言，这三种方案具有很好的代表性。同时，为保证公平性，垃圾回收阈值统一设置为 15%，垃圾回收算法均采用 CBE。

4.2.2 评估指标

- 写放大系数（WAF）

  写放大系数可以直观表现出不同数据放置策略对冷热数据的分类效果，其计算方式为$WAF = \frac{data_{uw}+data_{gw}}{data_{uw}}$，其中$data_{uw}$和$data_{gw}$分别代表用户写入数据量和垃圾回收写入数据量。

- 垃圾回收写入量

  垃圾回收写入量$data_{gw}$可以很好地量化不同数据放置策略之间额外数据写入量的差异，越少的额外数据写入量则意味着更长的 SSD 使用寿命。

- 垃圾回收次数

  对于 SSD 而言，垃圾回收操作是影响 SSD 读写性能和造成长尾延时的主要因素，究其原因是擦除闪存块的耗时要比读写数据多得多。故除去因释放空间所需的必要擦除以外，额外的垃圾回收操作次数越低越好。垃圾回收次数$count_{gc}$能很好地间接反映出不同数据放置策略对 SSD 性能的影响程度。为了更直观的体现出各数据放置算法额外垃圾回收次数的差异，本文以 ZB-OPT 为基准进行了归一化处理。

4.3 结果分析

4.3.1 整体对比

整体测试结果如表 4.3 所示：

表 4.3 整体效果对比

放置策略	平均写放大	平均垃圾回收写入量（GB）	平均垃圾回收次数（归一化）
Normal	2.008	442.5	2.028
Warcip	1.639	280.6	1.729
Sepbit	1.615	270.1	1.700
BHB-DP_pre	1.493	216.5	1.557
BHB-DP	**1.309**	**135.7**	**1.341**
FK	1.206	90.3	1.222
ZB-OPT	**1.017**	**7.3**	**1**

从结果可得，对于两种借助了未来信息的理论最优放置策略 FK 和 ZB-OPT 而言，ZB-OPT 的整体效果明显优于 FK，平均写放大不到 1.02，平均垃圾回收写入量也仅有 7.3GB，相比 FK 减少了 91.9%，同时，FK 的平均垃圾回收次数也比 ZB-OPT 多出了 22.2%，说明 ZB-OPT 更适合作为理论上限用于衡量其他实际放置策略的结果表现。

对于实际放置策略而言，BHB-DP 策略对于全部 10 个节点数据冷热分离的整体效果优于现有其他算法，成功验证了 BHB-DP 的有效性。相较于现有工作中最新的放置策略 Sepbit 而言，平均写放大降低了 19.0%，平均垃圾回收写入量减少了 49.8%，平均垃圾回收次数减少了 21.1%。

值得一提的是，仅根据写入频率区分冷热程度的放置策略 BHB-DP_pre 效果同样优于仅利用块寿命的 Sepbit 和 Warcip，说明在多租户 IO 混合的块存储系统持久化层场景下，优先利用写入频率能实现比直接使用块寿命更好的数据冷热分离效果。

此外，BHB-DP 效果整体优于 BHB-DP_pre，说明写入频率确实存在“历史影响”现象，加入块寿命对块热度值进行确认和调整后，能很大程度上解决该问题，实现更

好的数据放置效果。

4.3.2 写放大对比

写放大系数可以直观表现出不同数据放置策略对冷热数据的分类效果。10 个节点下各数据放置策略的写放大结果如图 4.1 和图 4.2 所示。清晰起见，图中对当前最新方案 Sepbit 以及本文提出的方案 BHB-DP 的结果进行了标注。

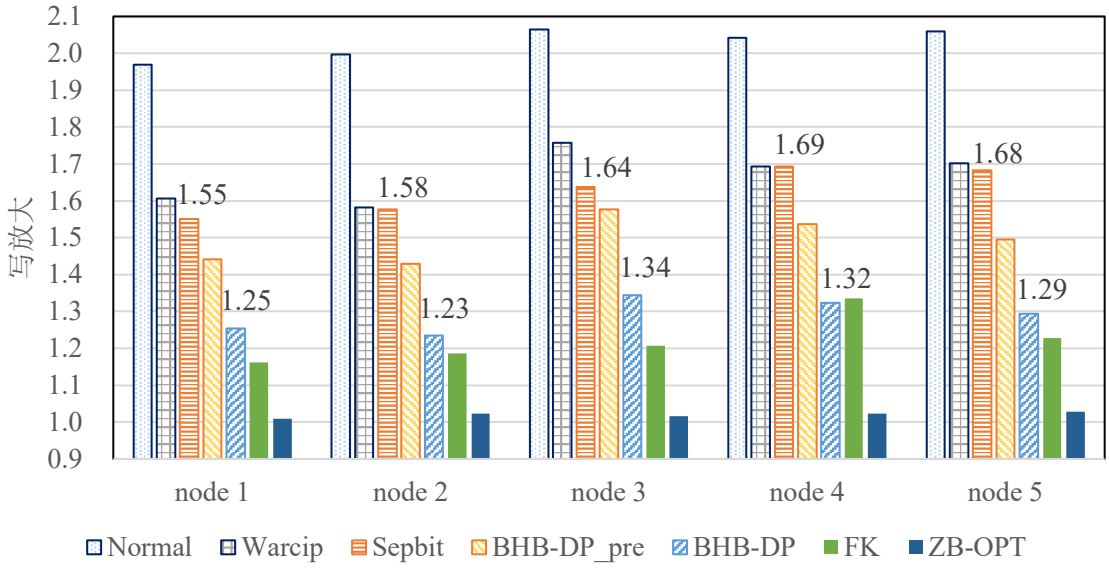


图 4.1 节点 1~5 写放大对比结果

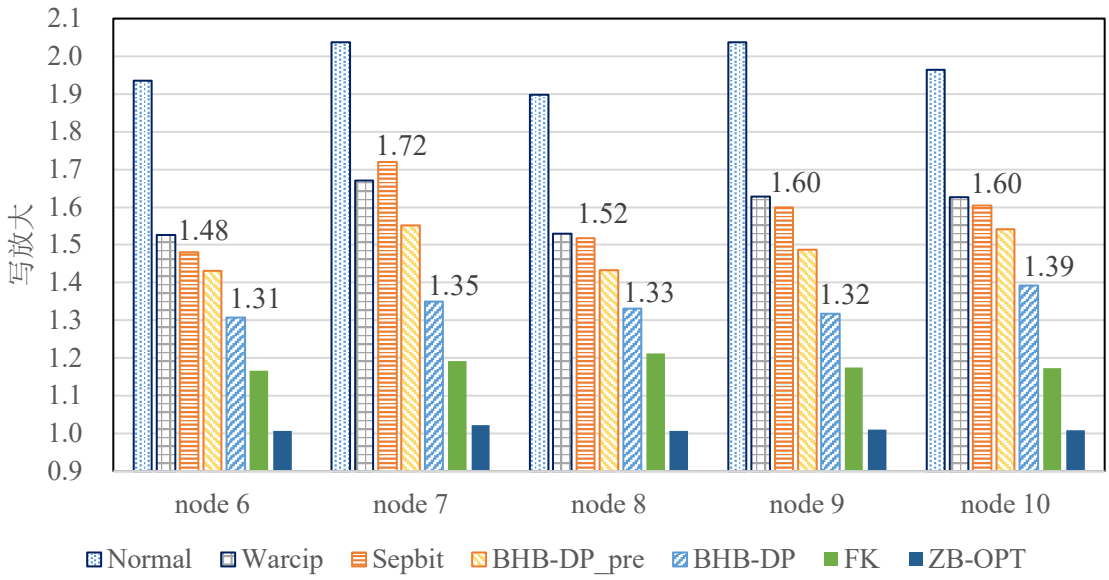


图 4.2 节点 6~10 写放大对比结果

可以看到在全部节点中，除了利用了未来信息的理论最优策略 FK 和 ZB-OPT 外，BHB-DP 的写放大在实际策略中均为最低，且明显低于现有研究中的策略 Sepbit 和 Warcip。Normal 策略因为没有进行冷热数据分离，导致写放大均为最高。BHB-DP 相较于绝大部分节点比 Warcip 表现更优的 Sepbit 而言，写放大最小降低了 11.5%，最多降低了 23.2%。说明 BHB-DP 对于多种负载场景均具有更好的适应性。

有趣的是，在节点 4 的负载场景下，BHB-DP 的写放大甚至要略低于 FK，一个合理的推测是节点 4 存在大量暖数据和冷数据，根据 FK 算法会将其全部放置于 type_id 为 6 的分区中，导致该类别分区中的数据冷热分布严重不均，进而造成了严重的写放大问题。

对于两种理论最优策略而言，ZB-OPT 在 10 个节点负载下的写放大均只有 1.01 或 1.02，最大值为节点 5，也仅有 1.03。而 FK 写放大均在 1.1 以上，最大值甚至达到了节点 4 的 1.34。说明 ZB-OPT 对于多种块存储负载场景均能实现更好的最优数据放置效果，而 FK 则因不能匹配 ZNS SSD 的特性，导致最终效果不佳。

4.3.3 垃圾回收写入量对比

垃圾回收写入量能够很好地量化比较不同数据放置策略下的额外数据写入量差异。

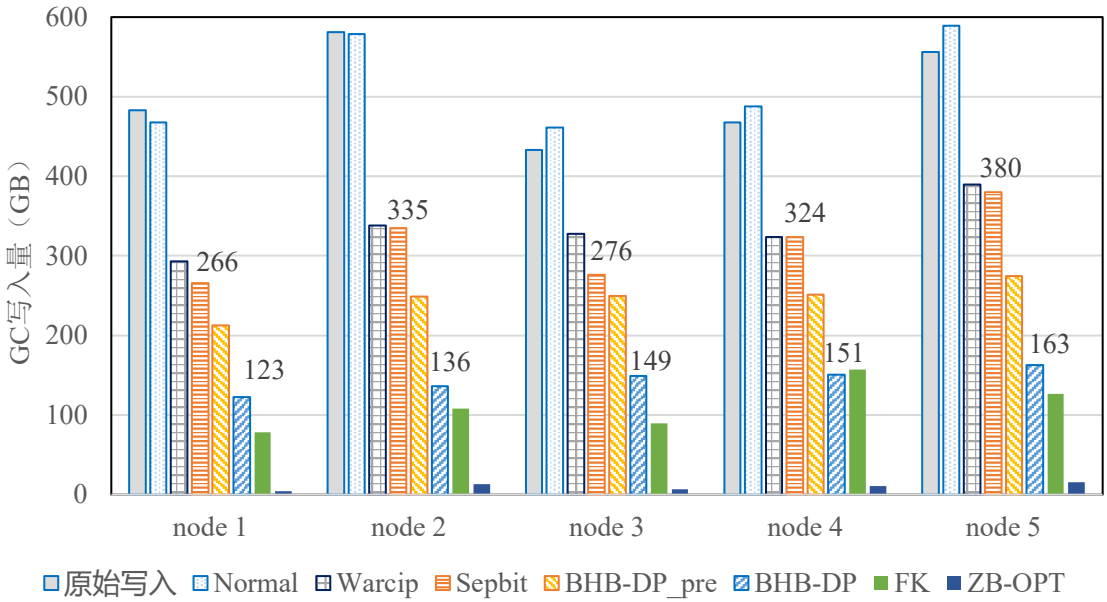


图 4.3 节点 1~5 垃圾回收写入量对比结果

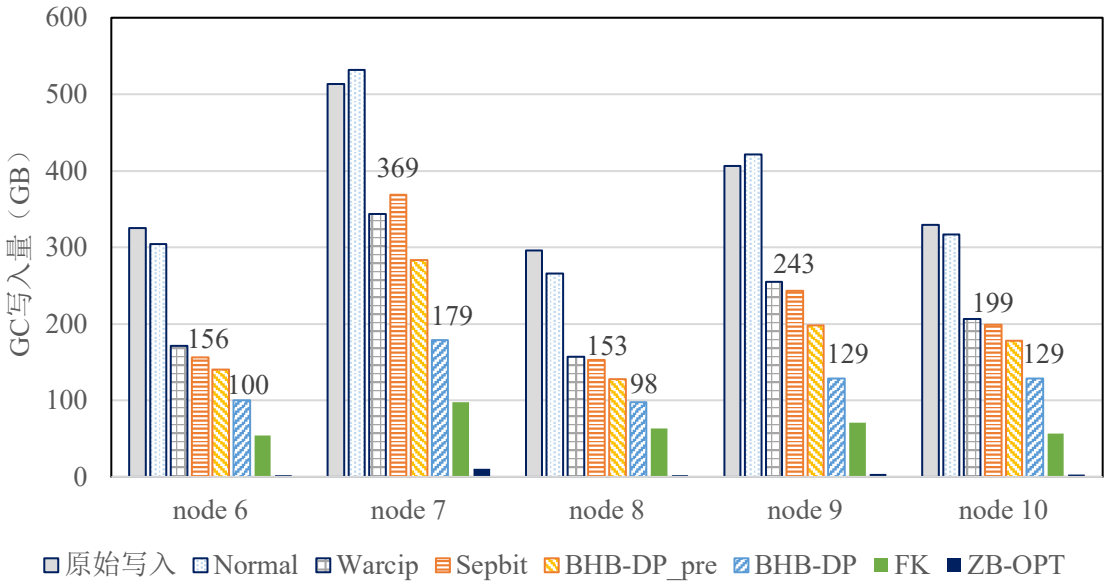


图 4.4 节点 6~10 垃圾回收写入量对比结果

10 个节点负载下的各算法垃圾回收写入量如图 4.3 和 4.4 所示。同时加入了用户原始数据写入量作为参照。图中同样对 Sepbit 和 BHB-DP 的结果进行了标注。

可以看到，10 个节点的原始数据写入量差异明显，从 296~580GB 不等。若是不引入数据放置算法（即 Normal 策略），在一些节点（如节点 3、4、5）的负载下，垃圾回收写入量甚至会超过用户数据写入量，这将严重损耗 SSD 的使用寿命。

BHB-DP 在全部节点下的垃圾回收写入量仅有 98~179GB，相较于其他实际策略所造成的大量垃圾回收写入而言均有不同程度的降低，且写入量越多，放置算法之间的差异越明显。BHB-DP 相较 Sepbit 而言最少降低了 35.9%，最多降低了 59.4%的单节点垃圾回收写入量。且对于节点 1、2、4、5 和节点 7 而言，BHB-DP 的垃圾回收写入量甚至不到 Sepbit 的一半，能显著延长 SSD 的使用时间。

对于两种理论最佳策略而言，不同于 FK 仍有数十甚至上百 GB 的垃圾回收写入，ZB-OPT 因几乎没有垃圾回收写入量而在图中几乎不可见。

4.3.4 垃圾回收次数对比

垃圾回收次数能很好地反映出不同数据放置策略对 SSD 性能的影响程度。为直观表示各放置策略额外垃圾回收次数的差异，以 ZB-OPT 的垃圾回收次数为基准，

10 个节点负载下全部数据放置策略的垃圾回收次数归一化之后的值如图 4.5 和图 4.6 所示。

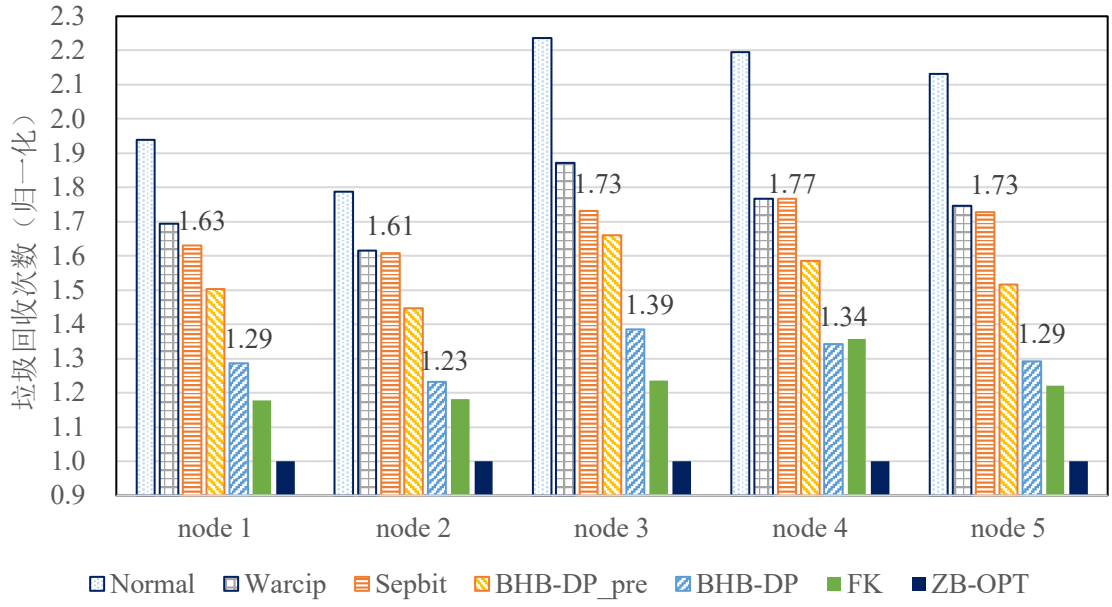


图 4.5 节点 1~5 垃圾回收写入次数（归一化）的对比结果

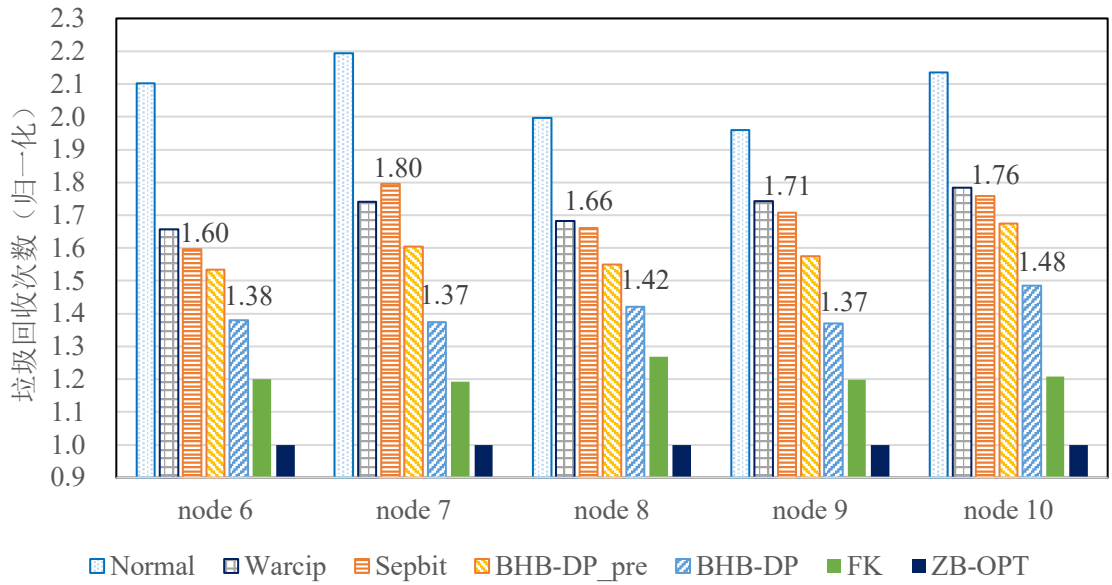


图 4.6 节点 6~10 垃圾回收写入次数（归一化）的对比结果

从图中可得，尽管理论最优策略 FK 利用了未来信息，在 10 个节点的负载场景下垃圾回收次数比 ZB-OPT 高出了 18%~36%不等，依然会对 SSD 的性能造成严重负面影响，说明 ZB-OPT 更适合作为理论性能上限衡量其他策略的表现。

对于实际数据放置策略而言，全部节点测试场景下 BHB-DP 的垃圾回收次数仅位于 1.23~1.48 之间，相较其他放置策略的垃圾回收次数均为最少。说明 BHB-DP 在现有的实际数据放置策略中对于 SSD 性能所造成的负面影响最小。具体到单节点，BHB-DP 的垃圾回收次数相比于 Sepbit 最少降低了 13.8%，最多降低了 25.4%，垃圾回收开销明显更低。

4.4 本章小结

本章利用真实场景下的负载日志记录对 BHB-DP 和 ZB-OPT 以及相关方案进行了对比测试。首先介绍了仿真测试环境以及利用 FEMU 对 ZNS SSD 进行仿真的参数设置，接着介绍了参与对比测试的几种方案，然后对评估指标进行了说明，最后介绍了对比测试结果并进行了分析。测试结果表明，对于同样利用了未来信息的理论最优放置策略而言，ZB-OPT 的写放大系数明显优于 FK，平均垃圾回收写入量也仅有 7.3GB，相比 FK 减少了 91.9%，说明 ZB-OPT 能更好的作为理论上限评估其他实际数据放置策略的表现。对于几种实际数据放置策略而言，BHB-DP 在全部节点的负载测试结果均为最优，能够实现最低的额外数据写入量和垃圾回收次数，延长 SSD 的使用时间以及优化其性能。相较目前最新的数据放置策略 Sepbit，BHB-DP 的写放大平均降低了 19.0%，垃圾回收写入量平均降低了 49.8%，垃圾回收次数平均减少了 21.1%。

5 总结与展望

当前云块存储系统普遍采用了 SSD 作为后端存储介质以获取更好的读写性能，但传统 SSD 无法感知数据块热度，将冷热数据混合放置，进而频繁触发 GC 操作带来大量额外数据写入，造成了严重的写放大问题，危害 SSD 的使用寿命及其性能表现。新硬件 ZNS SSD 支持用户端自定义数据放置策略，为改善 SSD 的写放大问题提供了良好的契机。首先为了衡量其他实际数据放置策略对于冷热数据分类的效果表现，提出了适用于 ZNS SSD 的理论最优放置策略 ZB-OPT。然后为了解决现有关于写放大优化的研究无法很好的平衡冷热数据分类效果和额外开销的问题，提出了基于块热度的数据放置策略 BHB-DP。

5.1 本文主要内容及结论

本文的主要内容及结论总结如下：

（1） 对云块存储系统的写负载进行了深入分析。首先，对来自真实生产环境负载日志中的写请求进行了数据统计，结果表明尽管通过负载均衡使得用户数据分布被成功打散，均匀分布到了各个存储节点，但数据热度分布不均，不同节点的负载访问模式不同。然后对衡量数据冷热程度的两种关键特征——写入频率和块寿命分别进行了分析。从写入频率的分布可得，整体近似服从 zipf 分布，具有很明显的偏斜特性，即高写入频率数据块和低写入频率数据块之间存在明显差异，且存在一定比例的一次写入数据。从块寿命的分布可得，整体同样近似服从 zipf 分布，且物理寿命和逻辑寿命这两种块寿命指标并无明显差异，同时，从块寿命分类阈值随时间的变化可得，数据热度分布频繁变化，且存在阈值交叉现象，说明需要及时对数据块热度以及分类阈值进行调整。

（2）提出了适用于 ZNS SSD 的理论最优放置策略 ZB-OPT。利用未来知识，提前预知数据块的寿命理论上可以实现最小化写放大的数据放置效果，但通过理论分析和实际测试发现，ZNS SSD 有限的同时开放分区数量无法满足细粒度切分后的不同块寿命区间所需要的大量分区类别。ZB-OPT 采用了新的分类策略来均衡各类别分

区承担的数据写入量，以解决同一个分区内冷热数据差异过大的问题。测试结果表明，ZB-OPT 在全部存储节点负载下的平均写放大系数仅有 1.017，相较于针对传统 SSD 的理论最优放置策略的垃圾回收写入量平均减少了 91.9%，能够更好的作为理论上限来衡量其他实际数据放置策略的结果表现。

（3）提出了基于块热度的数据放置策略 BHB-DP。针对现有研究方案中存在的问题，采用了一种轻量级的数据热度分类方案，根据写入频率和块寿命，通过两阶段的方式获取块热度值，然后根据块热度值对数据块进行冷热分类，将块热度值相同的数据写入到相同的分区中。为了适应数据热度分布的频繁变化，BHB-DP 会及时更新数据块的热度值来调整其所属类别。此外，BHB-DP 利用 CBE 算法挑选待回收分区，以避免 GC 过程中的误判问题和负收益现象。最后设计并实现了 BHB-DP 的整体框架。测试结果表明，BHB-DP 相较其他现有研究中的实际数据放置策略，有效降低了写放大系数，减少了垃圾回收写入量以及垃圾回收次数。

5.2 本文的主要创新点

本文的创新点主要可以归纳为下述两方面：

（1） 提出了适用于 ZNS SSD 的理论最优放置策略。针对传统 SSD 设计的理论最优放置策略无法适用于存在同时开放分区数量物理限制的 ZNS SSD，本文根据负载分析结论，通过数学推理，指出采用指数级扩大的阈值区间能够均衡各类别分区承担的数据写入量，从而解决单一分区内数据冷热程度差异过大的问题，并据此提出了采用新分类策略的理论最优放置策略 ZB-OPT，用于实现最小化 ZNS SSD 写放大的理论最佳效果。

（2） 提出了一种轻量级的数据热度分类方案。该方案充分考虑了两种用于冷热数据分类的关键指标——写入频率和块寿命各自的特点，根据这两种指标通过计算加策略的方式获取块热度值用于冷热数据分类，再将块热度值相同的数据放置于同一类分区中，能够低开销实现良好的数据放置效果，降低垃圾回收过程中的有效数据迁移量，解决现有研究中直接通过预测块寿命来来进行冷热数据分类的方案无法兼顾准确性和额外开销的问题。

5.3 展望

本文提出了一种新的数据放置策略用于优化基于 ZNS SSD 的云块存储系统内的写放大问题，并提出了适用于 ZNS SSD 的理论最优放置策略用于衡量实际放置策略的效果表现。尽管本文的优化工作取得了较好效果，但仍存在一些问题值得进一步探究和讨论：

（1） 适用于 ZNS SSD 的索引优化研究。ZNS SSD 支持在主机端自定义 Host FTL 用于数据管理，索引结构部分存在大量理论优化空间。考虑到未来 SSD 容量进一步增加，索引数据量也会随之增大，如何设计一种低空间开销且高性能的索引结构是一个重要且有挑战性的研究课题。

（2） 和其他降低数据冗余度技术相结合的研究。现有云块存储系统正在尝试诸多方案来降低数据冗余度以实现更低的存储成本，例如纠删码技术、重复数据删除技术、数据压缩技术等，这些技术会对数据热度分布造成怎样的影响目前尚无定论，需要根据新的写负载特征进一步优化数据放置策略以获取更好的数据放置效果。

致 谢

不积跬步，无以至千里，不积小流，无以成江海。三年的光阴转瞬即逝，但三年来不懈奋斗的汗水早已全部融入了我未来人生大厦坚实的地基中。在迈入下一个全新的阶段以前，我想在此对研究生涯中帮助过我的人致以由衷的感谢——

感谢伟大的祖国与党，面对汹涌来袭的疫情，不惧艰险，团结群众，万民一心，最终成功打赢了这场艰苦卓绝的防疫攻坚战，守护住了全国人民稳定而幸福的生活。

感谢我的父母在这二十五个春秋冬夏期间的鼓励与付出，成功塑造了我阳光且自信的性格，让我有勇气去面对这尘世的风风雨雨，往后余生还望让我尽情回报养育之恩。感谢我的亲哥和嫂子，你们无微不至的疼爱常常让我觉得受宠若惊，希望我们在未来能共享更多欢乐。感谢世界第一探险小分队的各位常委成员对我的支持与陪伴，祝愿罗诗怡和张肖天长地久，早生贵子；郑诗雨工作顺利，早日遇到如意郎君。

感谢我的硕士导师王桦老师，您才识渊博，和蔼可亲，引领我迈入学术的宏伟殿堂。回望三年前我第一次到实验室找您进行面试考核，顺利通过时的那份喜悦犹在昨天。之后和您的每一次交流讨论都让我受益匪浅，您能够耐心倾听我不成熟的想法，并给出关键且细致的指导意见，同时鼓励我勇敢向前，无私的给予了我很多帮助，让我感激不尽。祝您身体健康，工作顺心。

感谢实验室的周可老师，李春花老师，刘渝老师，你们高屋建瓴的建议为我明晰了学术突破的方向，你们严谨的治学态度和深厚的学术积累是我需要不断学习与追求的目标。感谢平易近人的程海燕老师，在背后默默管理着实验室的诸多琐事。祝愿老师们身体健康，万事如意。

感谢张煜师兄，在学习和工作上都给予了我很多的关心和帮助，你对学术孜孜不倦的追求和勇攀高峰的热情是我不断追赶的榜样。在 CBS 一起共事的时光相当充实，让我收获颇丰。你的每一次经验传授我都铭记于心。祝师兄阖家欢乐，节节高升。

感谢刘莉学姐，作为实验室的知心大姐姐，阳光开朗，乐善好施，是实验室能够长期保持融洽和谐氛围的一大重要原因。和你的交流让我如沐春风，你时刻保持的积极乐观的生活与工作态度是我需要不断学习的目标。祝学姐工作顺利，事事顺心。

感谢已经毕业的陶孟凌，张嘉伟，张光钰师兄，和你们一起在腾讯奋斗的日子充实而又愉快，深圳湾傍晚的凉风惬意而又舒爽，希望你们在各自的工作岗位上更上一层楼。

感谢即将毕业的刘科，王鹏学长，你们强大的个人能力让我敬佩，祝愿你们能找到心仪的工作。

感谢同年进入实验室的刘啸松同学，一起跑步，一起科研的时光短暂而又难忘，希望你能多点自信，享受生活。感谢蔡宝清和谢青同学，神仙眷侣，着实让人羡慕，祝福你们百年好合。感谢方浩天，夷洲，陈万成同学，和你们一起在实验室奋斗的日子充满了汗水与乐趣，祝你们前程似锦。

感谢郭潇俊博士，你强大的个人魅力让我惊讶，期待你在未来大放异彩。感谢张湛，邢广杰，谭頔凡学弟，小伙子们前途无量，希望你们再创辉煌。

感谢腾讯 CBS 团队，感谢胡健鹰导师，和你们一起共事的日子让我收益颇丰，祝愿 CBS 越来越好。

此外，特别感谢我的女朋友郑沁，让我有机会去验证我人生中最重要的一系列命题与结论，以及有动力完成这篇毕业论文，希望我们能一起去享受往后的幸福时光。

最后，感谢各位评委老师们在百忙之中抽出宝贵时间认真审阅我的论文并给出的宝贵指导和建议。

2023 年 5 月 14 日于武汉光电国家研究中心

参考文献

[1] S. Han, P. Lee, F. Xu, Y. Liu, C. He, J. Liu. An In-Depth Study of Correlated Failures in Production SSD-Based Data Centers, in: 19th USENIX Conference on File and Storage Technologies, FAST 2021, 23-25 Feb. 2021, USENIX Association, 2021: 417-429

[2] B. Kim, M. Kim. LazyRS: Improving the Performance and Reliability of High-Capacity TLC/QLC Flash-Based Storage Systems Using Lazy Reprogramming. Electronics, 2023, 12(4): 843

[3] H. Li, Y. Zhang, D. Li, Z. Zhang, S. Liu, P. Huang, et al. Ursa: Hybrid block storage for cloud-scale virtual disks, in: Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25-28 Mar. 2019, ACM, 2019: 1-17

[4] Y. Zhang, P. Huang, K. Zhou, H. Wang, J. Hu, Y. Ji, et al. OSCA: An online-model based cache allocation scheme in cloud block storage systems, in: 2020 USENIX Annual Technical Conference, USENIX ATC 2020, 15-17 Jul. 2020, USENIX Association, 2020: 785-798

[5] Q. Li, Q. Xiang, Y. Wang, R. Wen, W. Yao, Y. Dong et al. More Than Capacity: Performance-oriented Evolution of Pangu in Alibaba, in: 21st USENIX Conference on File and Storage Technologies, FAST 2023, 21-23 Feb. 2023, USENIX Association, 2023: 331-346

[6] J. Kang, J. Hyun, H. Maeng, S. Cho. The multi-streamed solid-state drive, in: 6th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage '14, Philadelphia, PA, USA, 17-18 Jun. 2014, USENIX Association, 2014

[7] 周波. 基于高性能多流 SSD 的优化研究[硕士学位论文]. 上海: 华东师范大学, 2021

[8] M. Bjørling, J. González, P. Bonnet. Lightnvm: The linux open-channel ssd subsystem, in: 15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, 27 Feb. – 2 Mar. 2017, USENIX Association, 2017: 359-374

[9] 李志涛. 基于 Open-Channel SSDs 的新型可靠存储系统研究[硕士学位论文]. 北京: 清华大学, 2019

[10] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. Moal, G. Ganger, et al. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs, in: 2021 USENIX Annual Technical Conference, USENIX ATC 2021, 14-16 Jul. 2021, USENIX Association, 2021: 689-703

[11] M. Rosenblum, J. Ousterhout. The design and implementation of a log-structured file system. ACM Transactions on Computer Systems (TOCS), 1992, 10(1): 26-52

[12] C. Lee, D. Sim, J. Hwang, S. Cho. F2FS: A new file system for flash storage, in: Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, 16-19 Feb. 2015, USENIX Association, 2015: 273~286

[13] Y. Jiao, S. Bertron, S. Patel, L. Zeller, R. Bennett, N. Mukherjee, et al. BetrFS: a compleat file system for commodity SSDs, in: EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, 5 - 8 Apr. 2022, ACM, 2022: 610-627

[14] C. Min, K. Kim, H. Cho, S. Lee, Y. Eom. SFS: random write considered harmful in solid state drives, in: Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, 14-17 Feb. 2012, USENIX Association, 2012: 1-16

[15] T. Kim, D. Hong, S. Hahn, M. Chun, S. Lee, J. Hwang, et al. Fully Automatic Stream Management for Multi-Streamed SSDs Using Program Contexts, in: 17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, 25-28 Feb. 2019, USENIX Association, 2019: 295–308

[16] Y. Zhou, K. Wang, F. Wu, C. Xie, H. Lv. Seer-SSD: Bridging semantic gap between log-structured file systems and SSDs to reduce SSD write amplification, in: 2021 IEEE 39th International Conference on Computer Design (ICCD), Storrs, CT, USA, 24-27 Oct. 2021, IEEE, 2021: 49-56

[17] G. Yadgar, M. Gabel, S. Jaffer, B. Schroeder. SSD-based workload characteristics and their performance implications. ACM Transactions on Storage (TOS), 2021, 17(1): 1-26

[18] Q. Wang, J. Li, P. Lee, T. Ouyang, C. Shi, L. Huang. Separating Data via Block Invalidation Time Inference for Write Amplification Reduction in Log-Structured

Storage, in: 20th USENIX Conference on File and Storage Technologies, FAST 2022, Santa Clara, CA, USA, 22-24 Feb. 2022, USENIX Association, 2022: 429-444

[19] L. Chang, T. Kuo. An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded System, in: Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), San Jose, CA, USA, 27-27 Sep. 2002, IEEE, 2002: 187–196

[20] S. Im, D. Shin. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. Journal of Systems Architecture, 2010, 56(12): 641-653

[21] F. Margaglia, J. Gutenberg, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster. The Devil Is in the Details: Implementing Flash Page Reuse with WOM Codes, in: 14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, 22-25 Feb. 2016, USENIX Association, 2016: 95-109

[22] D. Park, D. Du. Hot data identification for flash-based storage systems using multiple bloom filters, in: 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), Denver, Colorado, USA, 23-27 May 2011, IEEE, 2011: 1–11

[23] J. Hsieh, T. Kuo, L. Chang. Efficient identification of hot data for flash memory storage systems. ACM Transactions on Storage (TOS), 2006, 2(1): 22-40

[24] J. Bhimani, N. Mi, Z. Yang, J. Yang, R. Pandurangan, C. Choi, et al. FIOS: feature based I/O stream identification for improving endurance of multi-stream SSDs, in: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 02-07 Jul. 2018, IEEE, 2018: 17-24

[25] J. Yang, S. Pei, Q. Yang. WARCIP: Write amplification reduction by clustering I/O pages, in: 11th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2019, Renton, WA, USA, 8-9 Jul. 2019, USENIX Association, 2019: 155-166

[26] K. Kremer, A. Brinkmann. FADaC: A self-adapting data classifier for flash memory, in: 11th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2019, Renton, WA, USA, 8-9 Jul. 2019, USENIX Association, 2019: 167-178

[27] Y. Lu, C. Wu, Y. Chen. A machine-learning-based data classifier to reduce the write amplification in SSDs, in: RACS '20: International Conference on Research in

Adaptive and Convergent Systems, Gwangju, Korea, 13-16 Oct. 2020, ACM, 2020: 213-218

[28] Roman Timofeev. Classification and regression trees (CART) theory and applications[硕士学位论文]. Berlin: Humboldt University, 2004

[29] P. Yang, N. Xue, Y. Zhang, Y. Zhou, L. Sun, W. Chen, et al. Reducing Garbage Collection Overhead in SSD Based on Workload Prediction, in: 11th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2019, Renton, WA, USA, 8-9 Jul. 2019, USENIX Association, 2019

[30] S. Hochreiter, J. Schmidhuber. Long short-term memory. Neural computation, 1997, 9(8): 1735-1780

[31] C. Chakraborttii, H. Litz. Reducing write amplification in flash by death-time prediction of logical block addresses, in: HotStorage '21: 13th ACM Workshop on Hot Topics in Storage and File Systems, Virtual Event, USA, 27-28 Jul. 2021, ACM / USENIX Association, 2021: 1-12

[32] M. Wu, W. Zwaenepoel. eNVy: a non-volatile, main memory storage system. ACM SIGOPS Operating Systems Review, 1994, 28(5): 86-97

[33] X. Hu, E. Eleftheriou, R. Haas, I. Iliadis, R. Pletka. Write amplification analysis in flash-based solid state drives, in: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference 2009, Haifa, Israel, 4-6 May 2009, ACM, 2009: 1-9

[34] Y. Li, P. Lee, J. Lui. Stochastic modeling of large-scale solid-state storage systems: Analysis, design tradeoffs and optimization, in: SIGMETRICS '13: ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, Pittsburgh, PA, USA, 17-21 Jun. 2013, ACM, 2013: 179-190

[35] B. Houdt. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. ACM SIGMETRICS Performance Evaluation Review, 2013, 41(1): 191-202

[36] M. Chiang, R. Chang. Cleaning policies in mobile computers using flash memory. Journal of Systems and Software, 1999, 48(3): 213-231

[37] W. Cheng, M. Luo, L. Zeng, Y. Wang, A. Brinkmann. Lifespan-based garbage

collection to improve SSD's reliability and performance. Journal of Parallel and Distributed Computing, 2022, 164: 28-39

[38] J. Kim, M. Jang, M. Tehseen, J. Oh, Y. Won. IPLFS: Log-Structured File System without Garbage Collection, in: 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, 11-13 Jul. 2022, USENIX Association, 2022: 739-754

[39] T. Stavrinos, D. Berger, E. Katz-Bassett, W. Lloyd. Don't be a blockhead: zoned namespaces make work on conventional SSDs obsolete, in: HotOS '21: Workshop on Hot Topics in Operating Systems, Ann, Arbor, Michigan, 1-3 Jun. 2021, ACM, 2021: 144-151

[40] D. Purandare, P. Wilcox, H. Litz, F. Shel. Append is near: Log-based data management on zns ssds, in: 12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, 9-12 Jan. 2022, 2022

[41] P. Jin, X. Zhuang, Y. Luo, M. Lu. Exploring index structures for zoned namespaces SSDs, in: 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15-18 Dec. 2021, IEEE, 2021: 5919-5922

[42] G. Oh, J. Yang, S. Ahn. Efficient key-value data placement for zns ssd. Applied Sciences, 2021, 11(24): 11842

[43] J. Jung, D. Shin. Lifetime-leveling LSM-tree compaction for ZNS SSD, in: HotStorage '22: 14th ACM Workshop on Hot Topics in Storage and File Systems, Virtual Event, 27-28 Jun. 2022, ACM, 2022: 100-105

[44] H. Lee, C. Lee, S. Lee, Y. Kim. Compaction-aware zone allocation for LSM based key-value store on ZNS SSDs, in: HotStorage '22: 14th ACM Workshop on Hot Topics in Storage and File Systems, Virtual Event, 27-28 Jun. 2022, ACM, 2022: 93-99

[45] G. Choi, K. Lee, M. Oh, J. Choi, J. Jhin, Y. Oh. A New LSM-style Garbage Collection Scheme for ZNS SSDs, in: 12th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2020, 13-14 Jul. 2020, USENIX Association, 2020: 1-6

[46] K. Han, H. Gwak, D. Shin, J. Hwang. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction, in: 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, 14-16 Jul. 2021, USENIX Association, 2021: 147-162

[47] J. He, S. Kannan, A. Arpaci-Dusseau, R. Arpaci-Dusseau. The unwritten contract of solid state drives, in: Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, 23-26 Apr. 2017, ACM, 2017: 127-144

[48] J. Li, Q. Wang, P. Lee, C. Shi. An in-depth comparative analysis of cloud block storage workloads: Findings and implications. ACM Transactions on Storage, 2023, 19(2): 1-32

[49] Li J, Wang Q, Lee P P C, et al. An in-depth analysis of cloud block storage workloads in large-scale production, in: 2020 IEEE International Symposium on Workload Characterization (IISWC), Beijing, China, 27-30 Oct. 2020, IEEE, 2020: 37-47

[50] C. Waldspurger, N. Park, A. Garthwaite, I. Ahmad. Efficient MRC Construction with SHARDS, in: Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, 16-19 Feb. 2015, USENIX Association, 2015: 95-110

[51] N. Megiddo, D. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache, in: Proceedings of the FAST '03 Conference on File and Storage Technologies, San Francisco, California, USA, 31 Mar. – 2 Apr. 2003, USENIX, 2003: 115-130

[52] S. Jiang, X. Zhang. 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

[53] H. Li, M. Hao, M. Tong, S. Sundararaman, M. Bjørling, H. Gunawi. The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator, in: 16th USENIX Conference on File and Storage Technologies, FAST 2018, Oakland, CA, USA, 12-15 Feb. 2018, USENIX Association, 2018: 83-90