新型固态盘数据放置技术研究综述

吴秋霖,袁 远*,谢徐超,谭玉娟,宋振龙,张 根,卢 凯
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 相比于传统的机械硬盘,基于闪存的固态盘(Solid-state Drives,SSD)具有高性能、低功耗、小体积以及抗震动等诸多良好特性,已经被广泛应用于嵌入式设备、个人电脑、数据中心服务器、超级计算机等多种存储系统中。固态盘内部实现了一个称为闪存转换层(Flash Translation Layer,FTL)的软件层次,屏蔽底层闪存特性并实现管理功能,使固态盘可以作为传统块接口设备使用。固态盘数据放置技术是指根据不同的存储需求,确定不同类型的写入数据在固态盘中存储位置的策略或机制,对于优化固态盘的性能、寿命和服务质量至关重要。传统固态盘在FTL中实现"冷热"数据分离算法,根据写请求的起始地址、长度、时间等信息判断数据类型从而确定数据存储位置,但是由于固态盘内写请求可用信息有限导致数据分类准确度较低。近年来,各类新型固态盘不断涌现,如多流固态盘、开放通道固态盘、分区命名空间固态盘以及灵活数据放置固态盘,它们普遍采用软硬件协同设计思想,拓宽主机与固态盘的数据语义通路,为固态盘数据放置技术提供新的设计思路。本文梳理了上述新型固态盘的数据放置技术的发展历程,详细介绍了每类技术的研究现状,重点分析它们的设计思想、实现方法、优缺点以及部署应用状态等。同时,还探讨了固态盘数据放置技术面临的挑战与未来的研究方向。

关键词: 闪存;固态盘;闪存转换层;数据放置;新型固态盘;软硬件协同设计
基金项目: 国家自然科学基金(No.62402504)
中图分类号: TP333 **文献标识码:** A **文章编号:** 0372-2112(XXXX)XX-0001-15
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20251243

Data Placement Technology in Novel Solid-State Drives: A Survey

WU Qiulin, YUAN Yuan*, XIE Xuchao, TAN Yujuan, SONG Zhenlong, ZHANG Gen, LU Kai
(*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha*, *Hunan* 410073, *China*)

Abstract: Compared to traditional hard disk drives, flash memory-based solid-state drives (SSDs) have many attractive characteristics, such as high performance, low power consumption, small size, and shock resistance, and have been widely adopted in various storage systems including embedded devices, personal computers, data center servers and supercomputers. SSDs implement a software layer called the flash translation layer (FTL), which abstracts the underlying flash memory characteristics and provides management functions, enabling SSDs to act as traditional block interface devices. Data placement technology refers to the strategies or mechanisms for determining the storage locations of different types of data within the SSD based on varying storage requirements. It is crucial for optimizing SSD performance, lifespan, and quality of service. Traditional SSDs typically implement hot-cold data separation algorithms within the FTL, identifying data types based on information such as start address, length, and time of write requests to determine storage locations. However, limited available information within SSDs leads to low data classification accuracy. In recent years, various novel SSDs have emerged, such as multi-streamed SSDs, open-channel SSDs, zoned namespace SSDs, and flexible data placement SSDs. They adopt a hardware-software co-design approach, broadening the data semantic path between the host and the SSD, and providing new design ideas for SSD data placement technology. This article reviews the evolution of data placement technology in these novel SSDs, provides a detailed introduction to the research status of each technique, and focuses on analyzing their design principles, implementation methods, advantages and disadvantages, as well as deployment status. Additionally, we discuss the challenges and future research directions of data placement technology.

Keywords: flash memory; solid-state drives; flash translation layer; data placement; novel solid-state drives; hardware-software co-design
Foundation Item(s): National Natural Science Foundation of China (No.62402504)

收稿日期:2026-02-13;录用日期:2026-03-06;责任编委:周可
*通信作者:袁远

0 引言

相比于传统的机械硬盘，基于闪存的固态盘（Solid-state Drives，SSD）具有高性能、低功耗、小体积以及抗震动等诸多良好特性。近年来，得益于制程工艺的提升以及3D堆叠、单元多比特（multi-level cell）等技术的发展，闪存固态盘的存储密度不断提升，单位容量存储价格不断下降，已经被广泛应用于嵌入式设备、个人电脑、数据中心服务器、超级计算机等多种存储系统中[1-4]。国际数据公司IDC的报告显示[5]，从2018年至2025年，全球数据在闪存设备上的存储比例从不足15%增长至接近30%，而在机械硬盘上的存储比例从65%降低至50%左右。

为了管理众多的闪存芯片、屏蔽闪存特性并将固态盘作为传统块接口设备使用，固态盘内部实现了一个称为闪存转换层（Flash Translation Layer，FTL）的软件层次[6-8]。闪存不支持覆盖写，已经写入数据的闪存区域只有被擦除之后才能继续写入。为了避免在写入过程中引入耗时的擦除操作，FTL采用异地更新的方式，将更新数据写入新的闪存位置并将旧数据标记为无效。FTL维护了逻辑地址到物理地址的映射表，将主机下发的逻辑地址转换为闪存的物理地址。当写操作将固态盘空闲页数量消耗至低于预设的阈值时，FTL还需执行垃圾回收操作以回收被无效数据占用的闪存空间。垃圾回收过程中，FTL选取一块闪存区域，将其中的有效数据迁移，然后擦除该区域。垃圾回收是导致固态盘写放大（即实际写入闪存的数据量与主机下发的数据量的比值）和性能下降的主要因素[9]。而且，闪存的耐久性有限，其在频繁的写入和擦除操作下会磨损失效，导致固态盘故障。因此，固态盘需要将写入和擦除操作均匀分散在不同的闪存单元上。

固态盘数据放置技术是指根据不同的存储需求，确定不同类型的写数据在固态盘中存储位置的策略或机制，对于优化固态盘的性能、寿命和服务质量至关重要。首先，合理的数据放置可以减少垃圾回收期间的有效数据迁移，从而提高固态盘读写带宽并降低请求时延。例如，通过将具有相似访问模式或生命周期的数据分组聚集，可以在某种类型的数据无效时擦除整个闪存区域，从而避免数据迁移。其次，合理的数据放置可以提高数据读写并行性及负载均衡。将并发访问的数据分布在并行闪存单元上，可以充分利用固态盘内部的并行性，从而提高固态盘的带宽。同时，频繁更新的数据可以放置在不同的闪存位置，以避免对闪存固定区域的频繁擦写操作，有助于实现磨损均衡以延长固态盘的使用寿命。第三，在多租户或混合工作负载环境中，来自不同应用程序或租户的读写可能会相互干扰，从而导致性能波动和QoS下降。数据隔离放置（例如为来自不同应用程序或租户的数据分配单独的存储区域）可以有效减轻这种干扰，确保稳定的I/O性能。

传统固态盘可在FTL中实现“冷热”数据分离算法，根据写请求的起始地址、长度、时间等信息判断数据类型从而确定数据存储位置，将频繁更新的数据（即热数据）与长期不更新的数据（即冷数据）分开存放，以降低固态盘垃圾回收数据迁移开销。但是，由于传统块接口的限制，在固态盘中获取的数据信息有限，数据分类准确度较低，冷热数据分离效果不佳。近年来，各类新型固态盘不断涌现，如多流固态盘（Multi-StreameD SSD），开放通道固态盘（Open-channel SSD，OCSSD），分区命名空间固态盘（Zoned NameSpace SSD，ZNS SSD）以及灵活数据放置固态盘（Flexible Data Placement SSD，FDP SSD）。这些新型固态盘普遍采用软硬件协同设计的思想，拓宽主机与固态盘的数据语义通路，允许主机软件间接指导或直接控制固态盘内的数据放置，为固态盘数据放置技术提供新的设计思路。本文对现有的新型固态盘数据放置技术进行了系统性的概述，首先梳理了新型固态盘的发展历程，然后详细介绍了每类新型固态盘数据放置技术的研究现状，重点分析它们的设计思想、实现方法、优缺点以及部署应用状态等。

本文余下部分组织如下：第一章介绍了传统固态盘的数据放置技术及挑战。第二章分析总结新型固态盘数据放置技术，包括核心原理、学术研究以及业界应用。第三章讨论了固态盘数据放置技术潜在的未来研究方向和技术趋势。第四章总结全文。

1 传统固态盘数据放置技术及挑战

1.1 传统固态盘数据放置技术

传统固态盘一般采用冷热数据分离技术，在FTL中实现冷热数据分离算法，将不同“温度”的数据分离，而将相似“温度”的数据存储到相同闪存块，减少垃圾回收过程中的数据迁移以提高垃圾回收效率，优化固态盘的性能和寿命。数据的“温度”表示数据被更新的可能性，通常热数据会在短期内被更新，而冷数据则长期保持稳定。通过冷热数据分离，无效数据会集中产生于存储热数据的闪存块中，而非分散在不同闪存块内，大幅降低回收热数据闪存块时引发的数据迁移。冷热数据分离技术的核心是数据冷热的判断，根据数据冷热的判断方式，可将冷热数据分离算法分为启发式算法、显式算法及隐式算法。

（1）启发式算法

启发式算法一般使用简单的经验规则判断写数

据的冷热,文献[10]将垃圾回收迁移的数据视为冷数据,而将主机下发的写数据视为热数据,还有研究[11-12]将写请求长度小于阈值的数据判断为热数据,反之为冷数据。

(2)显式算法

显式算法记录写入固态盘的每个数据块的更新特征信息,并以此判断数据块的冷热。数据块的更新特征信息通常包括以下两个方面:一是数据块更新的频次信息(frequency),部分研究[13-15]将更新频次高于阈值的数据判断为热数据,否则为冷数据;二是数据块更新的时间信息(recency),部分研究[16-17]将数据块上次写入时间与当前时间间隔小于阈值的数据判断为热数据,否则为冷数据;还有研究[18-22]将数据块最近两次更新的时间间隔小于阈值的数据判断为热数据,否则为冷数据。

(3)隐式算法

隐式算法使用额外的数据结构判断写入固态盘的数据块的冷热。有研究使用固定长度的LRU链表[23-24]或者固定大小的布隆过滤器[25-26]记录最近写入固态盘的数据块逻辑地址和写入次数等信息。每当有写请求到达固态盘时,FTL将每个写入数据块的逻辑地址与LRU链表或布隆过滤器上记录的地址对比,如果该数据块地址在LRU链表或布隆过滤器上,且该数据块写入次数大于预设的阈值,则该数据块被判断为热数据,否则为冷数据。

(4)小节

启发式算法数据冷热判断方式简单,对固态盘的计算和内存资源消耗可忽略不计,适用于资源严重受限的场景,如嵌入式设备等,但是相应的,其冷热数据判断准确度较低。显式算法需要记录写入固态盘的每个数据块的更新特征信息,产生的内存和计算开销较大,适用于资源较为充足且对冷热数据判断准确度要求较高的数据中心或云存储中的企业级固态盘,如文献[21]提出的SepBIT方案已经部署于阿里云以支持其日志结构块存储管理,与之前的优化方案相比,SepBIT可将固态盘写放大最多降低约20.2%,吞吐量提升约20%。隐式算法使用额外的数据结构判断数据冷热,其数据结构大小可配置,其内存和计算开销及冷热数据判断准确度适中,适用于嵌入式设备和一般消费级固态盘。

1.2 传统固态盘数据放置技术的挑战

冷热数据分离技术是最常见的固态盘数据放置技术,通常在固态盘的FTL中实现,主机软件无需任何修改,具有良好的兼容性,但是其存在以下挑战。

首先,现有的冷热数据分离算法存在一些缺陷,启发式算法使用简单的经验式规则判断数据冷热,其开销较小但是判断准确度较低。显式算法需要记录写入固态盘的每个数据块的更新特征信息,产生的内存开销较大,有研究将全部更新特征数据存储在闪存上,只缓存部分数据以降低内存开销,但是带来了额外的闪存读写开销。隐式算法使用额外的数据结构判断数据冷热,如LRU链表或布隆过滤器,其产生的内存开销较小,但是其冷热判断准确度受限于多个可调的参数,如LRU链表或布隆过滤器的大小、数据块写入计数的衰减周期等,而且一些数据结构存在固有缺陷,如链表的查询和管理开销、布隆过滤器的假阳性误判问题等。

更重要的是,冷热数据分离算法利用写请求的信息判断数据冷热,但是由于传统固态盘块接口的限制,固态盘内可获取的写请求信息有限,仅包括起始地址、长度、时间等,数据冷热判断准确度较低。而且,现有的冷热分离算法通常以数据块的逻辑地址作为数据块的唯一标识,当主机软件采用日志结构或写时复制等异地更新的写入方式时(如F2FS、BtrFS等文件系统),同一数据的逻辑地址不固定,现有的冷热判断算法会失效。

2 新型固态盘数据放置技术

2.1 新型固态盘

传统固态盘使用块接口保证了其通用性,方便快速部署与应用,但是也造成主机软件与固态盘内部存在语义鸿沟,主机软件不感知固态盘内部的闪存特征、数据布局与操作,固态盘也不感知主机写入数据的类型、寿命等信息。

近年来,各种新型固态盘不断涌现,包括多流固态盘、开放通道固态盘、分区命名空间固态盘及灵活数据放置固态盘。这些新型固态盘普遍采用软硬件协同设计的思想,拓宽了主机与固态盘之间的数据语义通路,通过主机软件向固态盘传递额外数据语义或者固态盘对外暴露内部细节和操作,主机软件可以间接指导或直接控制固态盘内的数据放置,为固态盘数据放置技术提供新的设计思路。图1展示了上述新型固态盘及数据放置技术的发展演进路线,包括每类固态盘的出现时间及相关标准是否被存储协议支持。

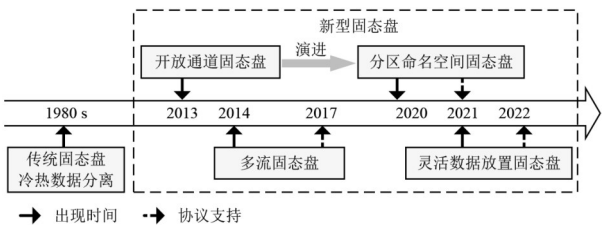


图1 新型固态盘数据放置技术发展路线

Figure 1 Development route of novel SSD data placement technology

2.2 多流固态盘

多流固态盘允许主机软件为写入固态盘的数据增加额外语义信息,指导固态盘内的数据放置。如图2所示,主机软件根据数据的类型、寿命等信息将写入固态盘的数据分为不同的“流”(stream)并附加流ID标识,然后固态盘将具有不同流ID的数据分离,将具有相同流ID的数据聚集存放,降低固态盘垃圾回收开销。图3展示了现有的多流固态盘相关研究及各项研究在I/O栈中所处的层次,多流固态盘技术的关键是数据流的划分,根据数据流划分的方式,可将多流固态盘分为静态数据分流与动态数据分流两种类型。

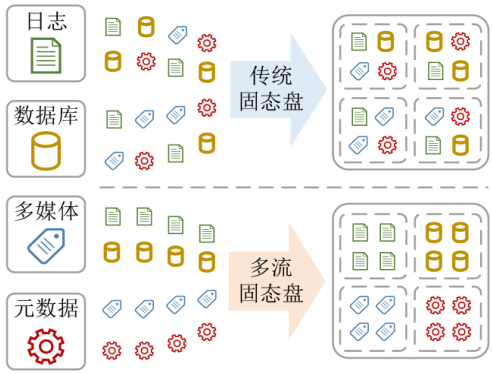


图2 多流固态盘示意图

Figure 2 Multi-streamed SSD

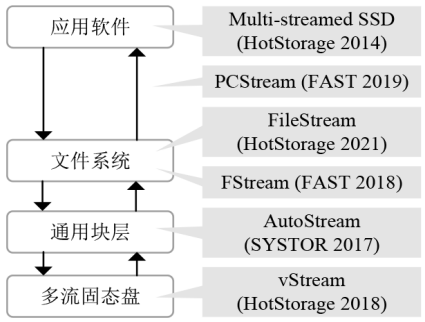


图3 多流固态盘相关研究

Figure 3 Research on multi-streamed SSD

(1)静态数据分流

静态数据分流的多流固态盘采用人为设定的固定分流策略,根据数据类型进行固定的数据流划分。文献[27]首次提出“多流”技术,在应用软件层进行数据流的划分及流ID的分配,将系统数据、应用内不同类型文件的数据(以Cassandra数据库为例,如Cassandra的日志文件、刷新文件、合并文件等)划分为不同的流,并通过系统调用将流ID传递至内核空间相关文件的管理结构上,然后随相应的写请求传递至固态盘内。实验结果表明,多流技术能将Cassandra的性能提升约56%,并将SSD寿命延长23%。但是在应用层进行数据分流存在以下问题,首先,需要同时对应用软件及系统内核进行修改,而且要求应用开发者对于应用的数据寿命、系统软件栈、流资源的限制(如设备支持的流数量等)有详细的了解,增加了应用软件开发的复杂度。其次,在多应用环境中(如虚拟机或容器),应用实例数量往往比设备支持的流数量多,导致在多个应用中分配流ID变得复杂,引入额外的管理开销。

由于文件系统可以获取其中所有文件和数据(文件系统元数据和日志等)的信息,文献[28]提出FStream,使用固定的分流策略,在文件系统层将不同类型的数据分流,并在两个常用的文件系统(Ext4和xfs)中实现了FStream。表1列出了FStream在两个文件系统中的数据流及分流策略。在文件系统层进行数据分流无需修改应用,减少应用软件的开发复杂度,并且可以支持各种不同应用,但是需要对文件系统进行大量修改,且需要针对不同的文件系统进行不同的定制化修改。

表1 Ext4与xfs中的数据分流

Table 1 Streams in Ext4 and xfs

文件系统	数据流	分流策略
Ext4	journal-stream	文件系统日志数据流
	inode-stream	inode数据流
	dir-stream	目录数据流
	misc-stream	inode/块位图、块组描述符数据流
	fname-stream	特定文件名数据流,如数据库日志文件
	extn-stream	特定文件扩展数据流,如视频文件
	data-stream	普通文件数据流
xfs	log-stream	文件系统日志数据流
	inode-stream	inode数据流
	fname-stream	特定文件名数据流
	data-stream	普通文件数据流

(2)动态数据分流

静态固定的分流策略不能很好地适应应用负载的变化,而且相同类型文件的数据寿命也会存在差异。因此,部分相关工作采用动态数据分流策略,根据数据特征将写数据动态划分到不同数据流。

文献[29]提出AutoStream,在块I/O层进行自动的数据流划分及流ID的分配。如图4所示,AutoStream使用分流算法,根据应用写请求的起始逻辑地址(sLBA)、请求大小(sz)等信息进行数据分流并生成流ID(sID)。AutoStream提出了两种分流算法,分别为多队列算法(Multi-Queue,MQ)和序列频率算法(Sequentiality Frequency Recency,SFR),以数据块组(包含若干个连续的数据块)为单位记录数据写入的

频次、时间、序列性等信息并进行数据分流。AutoStream无需针对不同应用和文件系统进行定制化修改,而且其动态自动分流策略能较好应对负载变化。但是AutoStream存在如下限制:首先,块I/O层获取的请求信息有限(请求的起始地址和长度),数据分流准确度有限;其次,在写时复制和日志结构等异地更新的文件系统中,同一数据块的逻辑地址不固定,基于请求地址的数据分流方法可能失效。

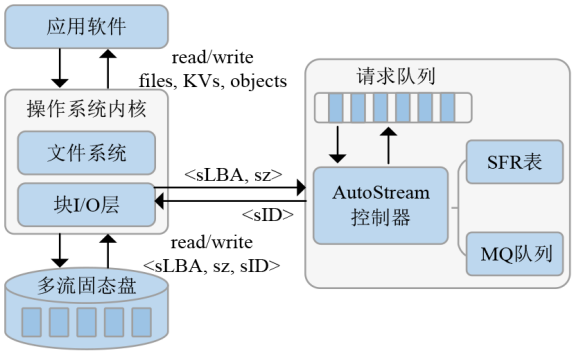


图4 AutoStream架构

Figure 4 Overview of AutoStream

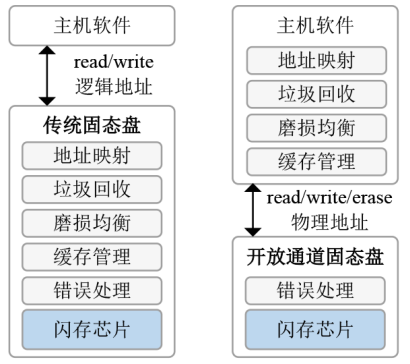
文献[30]提出FileStream,在用户空间进行数据分流并将流ID传递给内核。FileStream从某个打开文件的文件管理结构(inode)中获取该文件的7项信息,包括文件inode编号、文件名、大小、修改次数、打开时间、关闭时间、删除时间,使用机器学习的方法进行文件寿命预测并自动分配流ID,然后将流ID传递给内核中的对应文件,后续固定时间间隔内(假设为T s),此文件的数据写入均使用此流ID,T s之后,FileStream重新为该文件分配数据流。文献[31]提出基于I/O活动进行数据分流,由于程序上下文(Program Context,PC)描述了应用程序的数据从写入到write系统调用的执行路径,是I/O活动的有效标识,因此提出基于程序上下文的自动数据分流方法PCStream。PCStream获取每个数据块写路径上(write系统调用之前)各函数的PC值并相加得到PC和值,然后根据数据块的PC和值进行数据分流。FileStream及PCStream都需要同时修改用户程序与系统内核,方案部署应用较为复杂。

数据流的数量对于多流固态盘的性能至关重要,但是由于固态盘内部硬件资源的限制,一般固态盘仅支持4~16个数据流(物理数据流)。文献[32]提出虚拟数据流方案vStream,vStream支持最多65 536个虚拟数据流供软件开发者使用,然后计算每个虚拟数据流中数据的平均失效时间(数据寿命),并利用K-means聚类算法(将K配置为设备支持的物理数据流的数量)将虚拟数据流的数据寿命分类,以此将多个虚拟数据流映射至一个物理数据流。vStream还会定期更新虚拟数据流至物理数据流的映射关系,以适应负载变化。

当前,SCSI/SAS和NVMe协议均已提供对多流接口的支持[32],但是由于缺乏行业牵引,多流固态盘一直没有得到大规模部署应用。

2.3 开放通道固态盘

为了打破主机软件与固态盘的语义鸿沟,充分利用固态盘的容量与性能,开放通道技术被提出。如图5所示,开放通道技术将固态盘的介质特性、物理结构和内部操作暴露给主机软件,闪存转换层的功能被交由主机软件实现,主机软件可以直接对闪存进行读、写、擦除操作,还可以根据自身需求控制闪存数据放置、请求调度及执行垃圾回收,而固态盘内部仅实现部分硬件特性相关的功能(如数据错误处理等)。



(a) 传统固态盘 (b) 开放通道固态盘

(a) Traditional SSD (b) OCSSD

图5 传统固态盘与开放通道固态盘

Figure 5 Traditional SSD and OCSSD

OCSSD的出现引起了学术界与工业界的广泛关注,表2列举了部分OCSSD相关研究工作。按照时间可将OCSSD相关研究大致划分为三个阶段[49]。

首先,2016年之前的相关研究初步探索固态盘开放通道的优势,并逐渐形成和完善了OCSSD的概念。如图6(a)所示,OFTL[33]在内核空间中实现了FTL的所有功能,能够与上层软件进行交互,并可直接访问物理闪存。为减少写放大以延长闪存的使用寿命,OFTL对频繁的部分页更新进行缓冲和压缩,以降低更新次数,同时设计了一种基于反向指针的惰性索引技术,以减少索引元数据的持久化和日志记录开销。OFTL软硬件协同设计的思想体现了OCSSD概念雏形。SDF[34]引入一个基于FPGA的控制器并实现了主机端FTL直接访问物理闪存,可以消除固态盘内的垃圾回收预留空间,同时充分利用固态盘内闪存并行性以提升性能。NVMKV[35]利用主机端FTL实现基于哈希的键值(key-value, KV)存储,提高吞吐率并降低

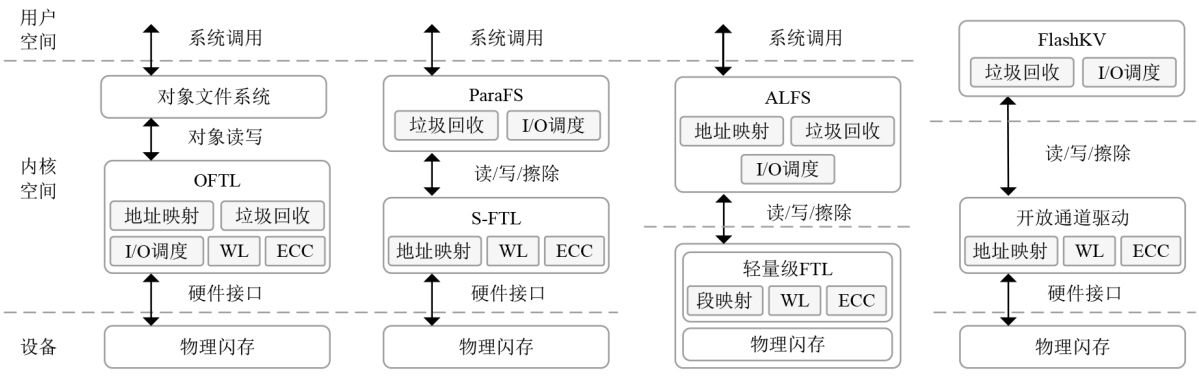
表2 开放通道固态盘部分相关研究

Table 2 Research on OCSSD

时间	相关研究
2013	OFTL[33]
2014	SDF[34]
2015	NVMKV[35]
2016	ParaFS[36]
	AMF[37]
2017	FlashKV[38]
	DIDACache[39]
	LightNVM[40]
2018	OC-Cache[41]
2019	OCStore[42]
2020	SSW[43]
2021	IODA[44]
2022	Prism-SSD[45]
2023	CFIO[46]
2024	CHEN et al[47]
2025	XHarvest[48]

写放大,同时保证可扩展性和符合ACID(原子性、一致性、隔离性和持久性)的KV操作。

随后,2016年至2018年的研究工作主要关注OCSSD的系统设计,提出部分相对成熟的OCSSD系统。如图6(b)所示,文献[36]实现了一个简单的轻量级内核FTL(S-FTL),执行磨损均衡、纠错码和块映射的功能,而在文件系统(ParaFS)中执行垃圾回收、I/O调度等功能。ParaFS使用二维(通道维度、数据冷热维度)数据放置策略,充分利用OCSSD的通道级并行性,同时减少垃圾回收期间的数据迁移开销。ParaFS采用两阶段调度策略,将读/写/擦除请求公平地分配到不同的闪存芯片,以获取稳定的性能。如图6(c)所示,AMF[37]在固态盘内仅保留一个轻量级的FTL,实现简单的段映射、磨损均衡及纠错码等功能,并对上层暴露闪存特性,然后在内核空间实现了一个基于F2FS的日志结构文件系统(ALSF),实现数据分布、垃圾回收、I/O调度等功能,通过冷热数据分离降低垃圾回收开销。如图6(d)所示,FlashKV[38]将FTL分为两个部分,一个用户空间与KV存储协同的lib库,以及一个内核空间的开放通道驱动器,两者通过ioctl接口进行交互。用户空间lib库实现了大部分FTL功能,如数据分布、垃圾回收与I/O调度,同时FlashKV将闪存垃圾回收与KV存储的合并操作协同设计,降低垃圾回收开销,而内核空间开放通道驱动实现与闪存硬件相关的简单功能。DIDACache[39]将KV缓存与用户空间FTL协同设计,使KV缓存可以直接管理底层闪存设备,消除多层冗余的映射管理,实现更优的数据分布、垃圾回收与请求调度,优化应用性能。文献[40]提出了第一个为OCSSD设计的通用子系统LightNVM,它使用物理块设备(physical block device,pblk)模块实现FTL功能,允许应用开发者直接管理闪存并行单元并根据应用负载自行设计垃圾回收与I/O调度,以优化I/O性能并提供可预测的请求时延。



(a) OFTL (b) ParaFS (c) AMF (d) FlashKV

图6 典型开放通道固态盘设计

Figure 6 Typical OCSSD design

最后,2018年之后的工作主要尝试基于OCSSD系统(如LightNVM或仿真平台)解决特定领域的问题,并对其进行优化以提高性能。OC-Cache[41]使用OCSSD实现I/O缓存,通过为不同的租户分配不同的闪存通道,将不同租户的I/O、垃圾回收等操作分离,实现强性能隔离并且充分利用闪存通道并行性。OCStore[42]在设备内保留一个轻量级的FTL执行磨损均衡、纠错码等操作,而在内核中将地址映射、垃圾回收等功能与对象管理系统融合,降低多层管理的开销。SSW[43]将随机写请求与顺序写请求进行分离存放,以优化垃圾回收开销。IODA[44]利用OCSSD实现了具有可预测时延窗口的闪存阵列,IODA控制闪存

阵列中每个OCSSD的垃圾回收时间,并以循环的方式在不同OCSSD上执行垃圾回收,保证闪存阵列中同一时间只有一个OCSSD在进行垃圾回收。Prism-SSD[45]针对OCSSD提供了三种层次的抽象,分别是物理闪存层抽象、闪存功能层抽象以及用户策略层抽象,并提供了多种应用程序编程接口,使软件开发者在可用性及效率之间取得最佳权衡。CFIO[46]基于LightNVM和OCSSD实现了一种低开销的冲突感知I/O机制,利用OCSSD暴露的物理闪存地址与固态盘内部状态进行I/O调度,减少闪存上的I/O冲突以提高固态盘性能。文献[47]基于OCSSD实现了基于内容定义分块(content-defined chunking)技术的重复数据删除方法,优化指纹存储和空间管理效率,提高重复数据删除性能。XHarvest[48]则利用新兴计算链路协议(Compute Express Link,CXL)和可信执行环境来实现动态、高效和安全的OCSSD系统,避免OCSSD的主机FTL与用户软件的资源竞争。

开放通道技术将FTL功能与主机软件协同设计,允许主机软件根据自身需求决定数据放置并控制垃圾回收,降低垃圾回收开销。FTL功能的实现是OCSSD的关键,现有的OCSSD相关工作中FTL的实现大致可以分为以下三类:第一,FTL功能由单个或多个内核模块实现,如OFTL和ParaFS;第二,FTL功能由一个内核模块和一个固态盘内的轻量级FTL实现,如AMF;第三,FTL功能由应用软件与内核协同实现,如FlashKV。虽然OCSSD能实现较大的性能收益,但是其同样存在以下问题:一方面,软件开发者需要具备充足的闪存相关知识,对应用或系统软件进行大量修改以适配OCSSD;另一方面,不同厂商、不同类型的闪存硬件特性各不相同,无法对外提供统一的接口,而且,闪存厂商可能不会对外暴露某些硬件特性与细节。因此,OCSSD技术一直没有形成统一的标准与规范,也没有得到大规模普及与应用。

2.4 分区命名空间固态盘

分区命名空间固态盘(ZNS SSD)是由开放通道固态盘发展而来,可以基本消除固态盘的写放大及盘内缓存和预留空间。如图7所示,ZNS SSD对外暴露一片连续的逻辑地址空间,并被分为众多固定大小的"分区"(zone),分区通常仅支持顺序写入且在被重置(reset)之后才能重复写入。ZNS SSD对外屏蔽了闪存介质特性与实现细节,内部实现了分区与闪存并行单元的映射、磨损均衡、错误处理等硬件强相关的功能。主机软件需要控制ZNS SSD上的数据放置及执行分区清理(垃圾回收)操作,包括迁移分区内的有效数据并对分区进行重置。

文献[50]提出了ZNS的概念,其由开放通道技术

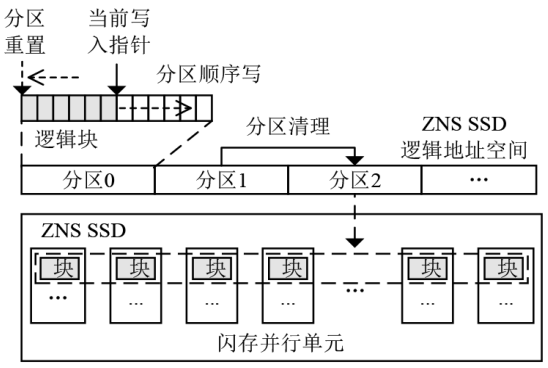


图7 分区命名空间固态盘示意图

Figure 7 Illustration of a ZNS SSD

发展而来,旨在降低设备端写放大、减少盘内缓存和预留空间、优化长尾时延和吞吐率。随后,NVMe协议于2021年增加了对ZNS的支持(NVMe 1.4 TP包)。文献[51]介绍了ZNS SSD的特点及相比于传统块接口SSD的优势,并通过修改F2FS文件系统和开发新的ZenFS文件系统实现了对ZNS SSD的支持。ZNS SSD数据放置相关工作大致可以分为两类,一类是ZNS SSD的分区清理优化,另一类是ZNS SSD与应用的协同优化。

(1)ZNS SSD分区清理优化

文献[52]认为ZNS SSD分区清理完全由主机软件执行增加了许多额外开销,如分区内有效数据迁移需要主机端请求处理、主机与设备间的数据传输、主机内存分配等,提出将ZNS SSD分区清理数据迁移卸载至固态盘内部执行,上层仅下发待清理分区的地址及数据迁移目标分区的地址,在固态盘内部完成数据迁移。文献[53]同样将分区清理的数据迁移操作卸载至固态盘内部以降低开销,同时还提出了分区与闪存块之间的动态映射方案,在待清理分区有效数据比例超过阈值时,使用地址重映射方法修改分区与闪存块的映射关系,将数据迁移目的分区映射至待迁移数据所在的闪存块,完成"逻辑"数据迁移操作,进一步降低数据迁移开销。

观察到现有的ZNS SSD产品的分区大小通常较大,可达到若干GB(如西数ZN540 SSD分区大小为2 GB,浪潮NS8600 SSD分区大小为1.44 GB),导致主机软件执行分区清理开销过大。文献[54]提出可灵活配置分区大小的ZNS SSD方案,FlexZNS,主机软件可以配置多组不同大小的分区(如256 MB、5 112 MB、1 GB等)以满足数据放置的需求并降低垃圾回收开销,每个分区映射到由多个并行闪存单元(如闪存晶圆)上相同偏移闪存块组成的闪存块组。

(2)ZNS SSD与应用协同优化

日志结构合并(Log-Structured Merge, LSM)树是

数据库应用中常用的数据结构,它将键值对数据以日志追加的方式写入数据库文件(Sorted String Table,SST)。存储设备上的众多数据库文件组织成多层级树状结构,新刷写的数据在最上层,并通过合并操作(compaction)逐渐向下层传递。同一层级(除最上层外)文件内的键值对数据按键排序且不同文件的键范围不交叠。合并操作读取相邻层级键范围交叠的文件,清理旧版本数据和文件以释放存储空间,同时生成新文件写入下层。LSM树的追加写与ZNS SSD分区的顺序写要求天然适配,但LSM树的合并操作与ZNS SSD的分区清理操作存在冗余。

有大量工作将基于LSM树的数据库应用(如RocksDB、LevelDB等)部署在ZNS SSD之上,并将LSM树的合并操作与ZNS SSD的分区清理操作结合优化,降低写放大并提升系统性能。文献[55]提出ZNSKV,在合并操作期间执行部分分区清理操作以减少数据迁移开销,同时根据存储设备的空闲容量调整合并操作的文件选择策略,在保持较高空间利用率的同时提高系统性能。由于RocksDB中的失效文件主要来自合并操作,文献[56]提出合并感知的分区分配方法CAZA,将RocksDB相邻层级中键范围交叠较多的SST文件放在相同的分区中,后续RocksDB执行合并操作时,分区中的数据更有可能同时失效,降低ZNS SSD分区清理的数据迁移开销。文献[57]提出寿命分层合并方法(LL-compaction),将LSM树不同层的文件存放在不同的分区,分区内按写入顺序选择文件执行合并操作,使分区内的文件顺序失效,分区清理时选取完全失效的分区执行清理操作。文献[58]提出LifetimeKV以降低分区清理的数据迁移开销,它使用范围合并算法,保证新生成的SST文件与上层文件具有不重叠的键范围,避免新生成的SST文件迅速参与到上层的合并中而失效,减少短寿命文件的数量。同时,LifetimeKV提出一种新的合并文件选取算法,优先选取寿命较长的SST文件参与合并。文献[59]提出同分区合并算法,尽量选取同一分区中的SST文件参与合并,最大化分区内无效数据占比,同时,新生成的SST文件被分配至预期在未来的合并操作中删除率较高的分区中,增加同分区合并的机会。文献[60]提出Prophet,综合利用刷写和合并事件来精准度量SST文件寿命,同时,提出SST文件寿命预测量化方法和SST文件分配方法,提升分区内无效SST文件比例,降低分区清理开销,最后,探索合并操作与分区清理融合的可能性,实现性能与写放大的更好权衡。

还有部分工作将现有的存储技术与ZNS SSD相结合。观察到同一文件或数据流压缩率相近的压缩局部性现象,文献[61]提出了Balloon-ZNS,根据数据压缩率决定闪存存储空间的管理及分区与闪存的映射,实现基于ZNS SSD的盘内压缩,进一步提升ZNS SSD存储能力。文献[62]和[63]则实现了基于ZNS SSD的全闪阵列。

ZNS SSD在OCSSD的基础上改进发展而来,闪存介质相关的特性和功能模块仍由固态盘内部处理,对外仅暴露分区逻辑地址,允许主机软件控制数据放置和垃圾回收(分区清理)的执行,优化了固态盘的性能和寿命,而且基本消除了盘内缓存和预留空间,降低了固态盘成本。学术界近年来有大量ZNS SSD相关研究,而在应用推广方面,ZNS接口已被NVMe协议支持,且西数[64]、三星[65]等厂商均推出了成熟的ZNS SSD产品。但是ZNS SSD也存在一些缺陷,首先,分区的顺序写要求限制分区的I/O请求队列深度为1,严重限制了ZNS SSD的性能优势,虽然使用append写入模式可以打破队列深度限制,但是需要对现有软件进行修改适配;其次,将垃圾回收交由主机软件执行增加了额外的请求处理、数据传输的开销,且现有ZNS SSD分区大小通常较大,可同时开启的分区数较少,不利于数据分离存放,进一步增加了主机软件执行垃圾回收的开销。

2.5 灵活数据放置固态盘

灵活数据放置(FDP)技术融合了谷歌Smart-FTL[66]与Meta直接放置方案,以优化固态盘数据放置。谷歌与Meta于2022年合作将FDP逐步整合进NVMe协议[67]。近年来,该技术在OCP、FMS、SDC等国际会议上进行了多次讨论和演进[68-69],正逐渐获得学术界与工业界的广泛关注。

FDP技术旨在降低固态盘垃圾回收开销,它既规避了ZNS SSD中显式垃圾回收带来的高昂软件成本,也避免了OCSSD所需的底层介质控制操作。FDP技术借鉴了多流技术的思想,通过为写入请求添加额外标识,使主机软件能够引导数据在固态盘内的放置。FDP技术为主机提供了将预期生命周期相近的数据分组管理的抽象机制。如图8所示,FDP技术引入以下概念以向上层系统暴露固态盘的物理架构[70]。

(1)回收单元(reclaim unit):是闪存管理的基本单位,由多个闪存块组成,其大小由固态盘制造商决定,类似于传统固态盘中跨多个闪存芯片/晶圆的相同偏移的闪存块组成的“超级块”;

(2)回收组(reclaim group):一组回收单元的集合。当设备允许主机根据主机策略隔离不同的闪存晶圆中的数据时,可以利用回收组实现;

(3)寿命组(endurance group):众多闪存块集合,这些闪存块的寿命被管理为一个单元,目的是均衡这

些闪存块磨损程度。一个典型的固态盘中通常只包含一个寿命组;

(4)回收单元句柄(reclaim unit handle):设备控制器内部的一种抽象机制,其功能类似于指针,主机无法直接操作回收单元,但是可以借助回收单元句柄引用回收单元,实现数据隔离。设备端则完全掌控句柄到物理回收单元的映射关系,因此可用回收单元句柄的数量直接决定了闪存中不同位置数据并行写入的最大并发能力;

(5)放置句柄(placement handle):索引命名空间的回收单元句柄列表,该列表在命名空间创建时定义。主机对命名空间的写操作只能使用此列表中的回收单元句柄。

(6)数据放置标识(data placement directive):写命令中的放置标识符,使用写命令中的Directive Specific(DSPEC)字段标识回收组和回收单元句柄,可将主机写入的数据放置至特定的回收单元。如果写命令中没有指定该标识,则固态盘默认使用与放置句柄0关联的回收单元句柄,并选择一个回收组来放置逻辑块;

(7)FDP配置:一项可供主机使用的可选功能,启用后可定义回收单元大小、回收组数量及回收单元句柄数量等参数。

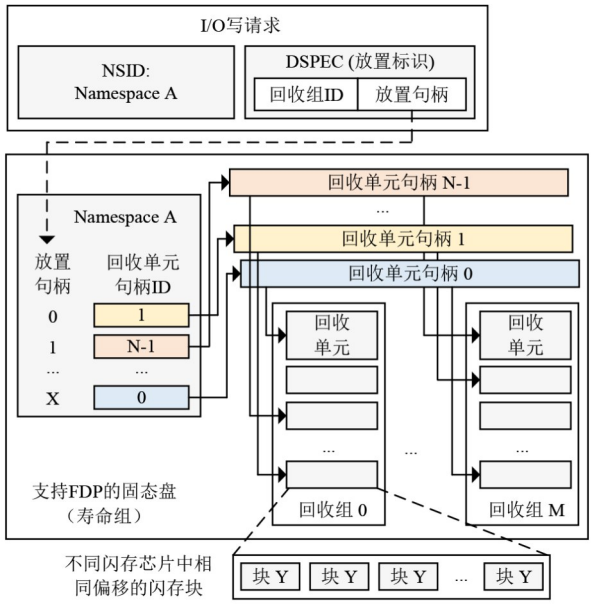


图8 灵活数据放置固态盘示意图

Figure 8 Illustration of a FDP SSD

FDP技术还增强了主机软件与固态盘的交互以优化固态盘性能。首先,主机软件可获取FDP SSD内的数据写入状态,从而帮助确定数据写入和恢复策略。例如,主机软件可以检索回收单元大小和当前回收单元剩余写入容量等参数,FDP SSD可向主机软件上报控制器事件以向主机警示有关回收单元句柄的异常情况。其次,FDP SSD可获取主机软件垃圾回收操作状态,避免固态盘与主机软件垃圾回收冲突,使固态盘只有弱垃圾回收,减少固态盘写放大与盘内的预留空间,同时优化请求时延和服务质量。最后,FDP技术具有良好的后向兼容性,可以在标准设备中激活,应用在不理解FDP的情况下也可获得收益,可以通过将不同应用或命名空间的数据分配给不同的回收单元句柄来实现。FDP技术可被用于各类场景以提升固态盘性能与使用寿命,降低存储设备总拥有成本,例如,在云计算环境中,可利用FDP技术将不同租户数据进行物理隔离,实现性能隔离并提升安全性;在数据库场景中,可利用FDP技术实现LSM树的合并操作与固态盘垃圾回收操作的协同优化,以及将数据文件、日志文件以及元数据文件分离存放以降低固态盘垃圾回收开销。

当前FDP相关软硬件生态正在逐步建立,在软件方面,Linux Kernel 5.19版本后可通过I/O Passthru[71]支持FDP、fio工具已基本支持设置回收单元和回收单元句柄并在做进一步适配、nvme-cli已支持FDP命令[72]。在硬件设备方面,三星的PM9D3a SSD[73]和大普微的Haishen5系列SSD[74]均已实现对FDP的支持。铠侠在OCP全球峰会上展示了其XD系列固态盘的FDP功能,并在RocksDB数据库进行了测试,测试结果表明,与使用传统固态盘的系统相比,启用FDP技术的系统在运行RocksDB应用时,固态盘使用寿命提高了约3倍,性能提高了约1.8倍[75]。浪潮信息推出基于FDP技术的SSD产品,突破FDP技术实现中智能流量控制、硬件资源动态隔离、应用场景兼容优化等难点,将SSD寿命提升2.5倍,每PB使用成本减少71%,可降低数据中心存储的总拥有成本[76]。开放数据中心委员会于2025年9月发布了《FDP SSD测试白皮书》,它为行业提供了一套详尽且权威的测试方法论,旨在全面评估FDP SSD的功能、性能、可靠性和兼容性,为存储系统设计者、企业用户及研究人员提供FDP SSD的技术验证依据[77]。而FDP相关学术研究还处于起步阶段,文献[78]基于NVMeVirt[79]仿真器实现了FDP SSD仿真器FDPVirt。文献[80]提出FDPFS,将FDP SSD抽象提升为文件系统目录,应用开发者可以控制数据放置,将语义相近的数据聚集存放在一起,而FDPFS负责将聚集的语义相近的数据赋予相同的放置标识。文献[70]研究利用FDP SSD来优化闪存缓存的性能并降低写放大,提出通过FDP SSD的数据放置能力,在缓存内实现大小对象分离,从而最大限度减少数据混合。

FDP技术允许主机软件指导固态盘内的数据放置,同时增强了主机软件与固态盘的交互,缩小语义鸿沟,提升固态盘性能并降低写放大。然而,与多流固态盘类似,使用FDP需要应用开发者对应用数据寿命有较为清楚的了解,而且需要对主机软件进行部分修改以适配FDP I/O栈。同时,固态盘内仍然需要执行垃圾回收,配置盘内缓存和预留空间。

2.6 本章小结

图9对比了现有的四类新型固态盘数据放置技术[81]。多流技术扩展了块接口,允许主机软件为写入数据添加寿命标签,以便在固态盘内部实现冷热数据分离。灵活数据放置技术允许主机指定数据放置位置,并且进一步增强了主机与设备的交互。而开放通道技术则彻底打破块接口的限制,将闪存的组织结构和操作接口完全暴露给主机软件,由主机软件控制数据放置、垃圾回收、介质操作等,使得主机软件过于复杂。分区命名空间接口在开放通道接口上发展而来,将逻辑地址划分为分区,简化固态盘内设计,而且固态盘对主机屏蔽了闪存特性与操作,便于使用。多流固态盘与灵活数据放置固态盘都是传统块接口设备扩展,可后向兼容,而开放通道固态盘与分区命名空间固态盘都有别于传统块接口设备,不可后向兼容。

图9 新型固态盘数据放置技术对比

Figure 9 Comparison of novel SSD data placement technology

3 未来展望

近年来,大数据、人工智能等技术不断发展,使得应用负载特征日益复杂多变,与此同时,固态盘容量持续提升,其垃圾回收单元尺寸也不断增大,给数据放置技术提出了更高的要求和挑战。另一方面,当前已提出的部分新型固态盘数据放置技术仍存在深入研究与优化的空间,同时,更多新型固态盘(如计算存储设备(Computational Storage Drive,CSD))持续涌现,为固态盘数据放置技术提供了新的设计思路。因此,固态盘数据放置技术未来的研究工作可从以下若干方面展开。

(1)ZNS SSD的通用高性能解决方案。ZNS接口支持传统write写入模式和新型append写入模式,当前的I/O栈都是基于write模式构建,而且应用场景集中于日志结构的数据库应用(如RocksDB等)。但是由于分区顺序写要求,write模式下分区的I/O请求队列深度被限制为1,严重限制了ZNS SSD的性能。而append模式消除了分区请求队列深度限制,但是需要主机软件进行修改适配。如何设计一个通用的高性能抽象层,可以支持不同应用软件无感使用ZNS SSD,且根据应用特征自主选择使用write模式或append模式,消除I/O请求队列深度限制并最大化系统性能,是一个重要的研究方向。

(2)FDP SSD的研究与应用。当前FDP技术处于发展初期,其具体的应用场景与使用方法仍在探索之中。首先,写入数据的放置标识直接影响垃圾回收的优化效果,需要仔细设计放置标识的来源与分配策略。例如,根据应用设置回收组ID,根据应用内文件设置放置句柄,或者根据租户(多租户共享一个固态盘的情况,如虚拟机或容器)设置回收组ID,根据租户内应用设置放置句柄等。其次,FDP支持主机软件与固态盘更紧密的交互,如事件上报、垃圾回收感知等,如何利用主机软件与固态盘的信息交互,降低写放大并优化系统性能,也值得深入研究。

(3)超大容量固态盘和混合存储架构固态盘的高效数据放置方法。三维堆叠和单元多比特等技术不断发展,推动闪存存储密度提升和成本下降,当前固态盘正在向超大容量方向发展,2024年主流国际厂商已经推出了百TB级容量的固态盘。超大容量固态

盘在垃圾回收上面临更严峻的挑战,首先,高密度闪存的写性能和写寿命变差;其次,为了降低管理开销,超大容量固态盘通常采用更大尺寸的映射粒度和垃圾回收单元,增加了冷热数据分离的难度和垃圾回收的开销。设计高效的数据放置技术以平衡固态盘管理开销与垃圾回收开销,是一个值得研究的方向。另一方面,为满足容量和性能需求,当前固态盘可采用多种类型存储介质构建混合存储架构固态盘,如持久性内存与闪存混合,或者不同类型的闪存混合。盘内数据放置方案的设计,需要考虑不同类型的存储区域具有的不同的性能、寿命特征。

(4)新型数据分类/放置技术。现有的数据分类/放置技术大致可以分为两类,一是基于写入数据的逻辑地址和更新特征判断数据冷热,如盘内冷热数据分离,但是其计算开销较大且无法应对异地更新的情况;二是主机软件指导或控制的数据放置,如FDP SSD和ZNS SSD等,然而其需要主机软件生态的修改适配。观察到数据寿命与压缩率的相关性,文献[82]提出了一种新颖的数据分离方法StreamCSD,利用新型计算存储设备中的嵌入式压缩器计算数据压缩率,然后使用机器学习的方法,基于数据压缩率进行数据分类。不同于上述现有的两类技术,StreamCSD无需主机软件修改适配,同时也避免了基于数据逻辑地址的分类方法的弊端,为数据分类/放置技术的发展与演进提供了新的思路。

(5)面向大模型的固态盘数据放置方法。近年来,大模型技术不断发展,已被广泛应用于各行各业[83-84]。大模型全流程(数据预处理、训练、推理、归档等)对数据的高度依赖以及对性能的极致追求,给存储系统提出了更高的要求与挑战。面向大模型的固态盘数据放置方法的设计,需要充分考虑大模型应用与传统应用的I/O负载特征差异[85-86],以及大模型为提升性能使用的诸如模型卸载、GPU直访存储[87](GPU Direct Storage,GDS)等技术特点。在I/O负载特征方面,大模型应用的I/O操作以读取为主[85-86],数据写入集中在训练阶段的参数更新、检查点写入以及推理阶段的KV卸载等,且数据写入有较为明显的规律。例如,参数更新的时间间隔和写入地址范围较为固定;检查点数据写入模式较为固定,其数据通常为一次写入多次读取,数据生命周期一般较长;KV卸载写入数据同样为一次写入多次读取,其数据生命周期一般较短。在模型卸载、GDS等技术方面,由于模型计算以及GPU的并行度较高,数据放置需要充分利用固态盘内部的多级并行结构,提高数据访问并行性。

4 结论

数据放置技术对于降低固态盘写放大与垃圾回收开销、提升固态硬盘的性能与使用寿命至关重要,一直是固态盘相关研究领域的热点。本文对现有新型固态盘数据放置技术进行了全面总结与分析,包括多流固态硬盘、开放通道固态硬盘、分区命名空间固态硬盘及灵活数据放置固态盘。多流固态硬盘与灵活数据放置固态硬盘允许主机软件为写入请求附加数据生命周期语义或放置标识符,从而指导固态盘内部的数据放置;开放通道固态硬盘与分区命名空间固态硬盘则突破了传统块接口的限制,使主机软件能直接控制固态硬盘内部的数据布局。当前固态盘数据放置还没有标准的解决方案,相关研究领域仍然发展蓬勃,方兴未艾。

参考文献

[1] 文宇鸿, 周游, 吴秋霖, 等. 多租户固态盘服务质量保障技术综述[J]. 计算机研究与发展, 2023, 60(3): 555-571.
Wen Yuhong, Zhou You, Wu Qiulin, et al. Quality of service guaranty technology of multi-tenant solid-state drives: A survey[J]. Journal of Computer Research and Development, 2023, 60(3): 555-571. (in Chinese)

[2] 杨梨花, 董勇, 邬会军, 等. 移动设备日志结构文件系统综述[J]. 计算机研究与发展, 2025, 62(1): 58-74.
Yang Lihua, Dong Yong, Wu Huijun, et al. Survey of log-structured file systems in mobile devices[J]. Journal of Computer Research and Development, 2025, 62(1): 58-74. (in Chinese)

[3] Liao Xiangke, Xiao Liquan, Yang Canqun, et al. MilkyWay-2 supercomputer: System and application[J]. Frontiers of Computer Science, 2014, 8(3): 345-356.

[4] Xu Weixia, Lu Yutong, Li Qiong, et al. Hybrid hierarchy storage system in MilkyWay-2 supercomputer[J]. Frontiers of Computer Science, 2014, 8(3): 367-377.

[5] IDC. Data age 2025[EB/OL]. (2018-11) [2025-12-22]. https://www.seagate.com/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[6] 陆游游, 舒继武. 闪存存储系统综述[J]. 计算机研究与发展, 2013, 50(1): 49-59.
Lu Youyou, Shu Jiwu. Survey on flash-based storage systems[J]. Journal of Computer Research and Development, 2013, 50(1): 49-59. (in Chinese)

[7] 舒继武, 陆游游, 张佳程, 等. 基于非易失性存储器的存储系统技术研究进展[J]. 科技导报, 2016, 34(14): 86-94.
Shu Jiwu, Lu Youyou, Zhang Jiacheng, et al. Research progress on non-volatile memory based storage system[J]. Science & Technology Review, 2016, 34(14): 86-94. (in Chinese)

[8] 高聪明, 石亮, 刘凯, 等. 闪存固态硬盘系统结构与技

术[J]. 计算机研究与发展, 2021, 58(7): 1518-1532.
Gao Congming, Shi Liang, Liu Kai, et al. Architecture and technologies of flash memory based solid state drives[J]. Journal of Computer Research and Development, 2021, 58(7): 1518-1532. (in Chinese)

[9] Min C, Kim K, Cho H, et al. SFS: Random write considered harmful in solid state drives[C]//Proceedings of the 10th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2012.

[10] Ma Dongzhe, Feng Jianhua, Li Guoliang. LazyFTL: A page-level flash translation layer optimized for NAND flash memory[C]//Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM, 2011.

[11] Lee S, Shin D, Kim Y J, et al. LAST: Locality-aware sector translation for NAND flash memory-based storage systems[J]. ACM SIGOPS Operating Systems Review, 2008, 42(6): 36-42.

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

[13] Stoica R, Ailamaki A. Improving flash write performance by using update frequency[J]. Proceedings of the VLDB Endowment, 2013, 6(9): 733-744.

[14] Chiang M L, Lee P C H, Chang R C. Using data clustering to improve cleaning performance for flash memory[J]. Software: Practice and Experience, 1999, 29(3): 267-290.

[15] Kremer K, Brinkmann A. FADaC: A self-adapting data classifier for flash memory[C]//Proceedings of the 12th ACM International Conference on Systems and Storage. New York: ACM, 2019.

[16] Hiep N V, Hsieh J W. Timestamp-based hot/cold data identification scheme for solid state drives[C]//Proceedings of the Conference on Research in Adaptive and Convergent Systems. New York: ACM, 2018.

[17] Park H, Lee E, Kim J, et al. Lightweight data lifetime classification using migration counts to improve performance and lifetime of flash-based SSDs[C]//Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2021.

[18] Bae D H, Chang J W, Park S M, et al. An effective data clustering method based on expected update time in flash memory environment[C]//Proceedings of the 29th Annual ACM Symposium on Applied Computing. New York: ACM, 2014.

[19] Yang Jing, Pei Shuyi, Yang Qing. WARCIP: Write amplification reduction by clustering I/O pages[C]//Proceedings of the 12th ACM International Conference on Systems and Storage. New York: ACM, 2019.

[20] Xie Wei, Chen Yong, Roth P C. ASA-FTL: An adaptive separation aware flash translation layer for solid state drives[J]. Parallel Computing, 2017, 61: 3-17.

[21] Wang Qiuping, Li Jinhong, Lee P P C, et al. Separating data via block invalidation time inference for write amplification reduction in log-structured storage[C]//Proceedings of the 20th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2022.

[22] Oh S, Kim J, Han S, et al. MIDAS: Minimizing write amplification in log-structured systems through adaptive group number and size configuration[C]//Proceedings of the 22nd USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2024.

[23] Chang Lipin, Kuo Teiwei. An adaptive striping architecture for flash memory storage systems of embedded system[C]//Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium. Piscataway: IEEE, 2002.

[24] Shen Biaobiao, Li Yongkun, Xu Yinlong, et al. A lightweight hot data identification scheme via grouping-based LRU lists[C]//15th International Conference on Algorithms and Architectures for Parallel Processing. Heidelberg: Springer, 2015.

[25] Hsieh J W, Kuo Teiwei, Chang Lipin. Efficient identification of hot data for flash memory storage systems[J]. ACM Transactions on Storage, 2006, 2(1): 22-40.

[26] Park D, Du D H C. Hot data identification for flash-based storage systems using multiple bloom filters[C]//Proceedings of the 27th Symposium on Mass Storage Systems and Technologies. Piscataway: IEEE, 2011.

[27] Kang J U, Hyun J, Maeng H, et al. The multi-streamed Solid-State drive[C]//Proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley: USENIX Association, 2014.

[28] Rho E, Joshi K, Shin S U, et al. FStream: Managing flash streams in the file system[C]//Proceedings of the 16th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2018.

[29] Yang Jingpei, Pandurangan R, Choi C, et al. AutoStream: Automatic stream management for multi-streamed SSDs[C]//Proceedings of the 10th ACM International Systems and Storage Conference. New York: ACM, 2017.

[30] Zhang Yuqi, Xue Ni, Zhou Yangxu. Automatic I/O stream management based on file characteristics[C]//13th ACM Workshop on Hot Topics in Storage and File Systems. Berkeley: USENIX Association, 2021.

[31] Kim T, Hahn S S, Lee S, et al. PCStream: Automatic stream allocation using program contexts[C]//Proceedings of the 10th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley: USENIX Association, 2018.

[32] Yong H, Jeong K, Lee J, et al. vStream: Virtual stream management for multi-streamed SSDs[C]//10th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley: USENIX Association, 2018.

[33] Lu Youyou, Shu Jiwu, Zheng Weimin. Extending the lifetime of flash-based storage through reducing write amplification from file systems[C]//Proceedings of the 11th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2013.

[34] Ouyang Jian, Lin Shiding, Jiang Song, et al. SDF: Software-defined flash for web-scale internet storage systems[C]//Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2014.

[35] Marmol L, Sundararaman S, Talagala N, et al. NVMKV: A scalable, lightweight, FTL-aware key-value store[C]// Proceedings of the USENIX Annual Technical Conference. Berkeley: USENIX Association, 2015.

[36] Zhang Jiacheng, Shu Jiwu, Lu Youyou. ParaFS: A log-structured file system to exploit the internal parallelism of flash devices[C]//Proceedings of the USENIX Annual Technical Conference. Berkeley: USENIX Association, 2016.

[37] Lee S, Liu Ming, Jun S, et al. Application-managed flash[C]//Proceedings of the 14th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2016.

[38] Zhang Jiacheng, Lu Youyou, Shu Jiwu, et al. FlashKV: Accelerating KV performance with open-channel SSDs[J]. ACM Transactions on Embedded Computing Systems, 2017, 16(5s): 139.

[39] Shen Zhaoyan, Chen Feng, Jia Yichen, et al. DIDA-Cache: A deep integration of device and application for flash based key-value caching[C]//Proceedings of the 15th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2017.

[40] Bjørling M, González J, Bonnet P. LightNVM: The Linux open-channel SSD subsystem[C]//Proceedings of the 15th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2017.

[41] Wang Haitao, Li Zhanhuai, Zhang Xiao, et al. OC-cache: An open-channel SSD based cache for multi-tenant systems[C]//Proceedings of the 37th IEEE International Performance Computing and Communications Conference. Piscataway: IEEE, 2018.

[42] Lu Youyou, Zhang Jiacheng, Yang Zhe, et al. OCStore: Accelerating distributed object storage with open-channel SSDs[C]//Proceedings of the 39th IEEE International Conference on Distributed Computing Systems. Piscataway: IEEE, 2019.

[43] Du Yazhi, Gu Jihua, Xiao Zhongzhe, et al. SSW: A strictly sequential writing method for open-channel SSD[J]. Journal of Systems Architecture, 2020, 109: 101828.

[44] Li Huaicheng, Putra M L, Shi R, et al. IODA: A host/device co-design for strong predictability contract on modern flash storage[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. New York: ACM, 2021.

[45] Shen Zhaoyan, Chen Feng, Yadgar G, et al. Prism-SSD: A flexible storage interface for SSDs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(4): 882-896.

[46] Zhu Jinbin, Wang Liang, Xiao Limin, et al. CFIO: A conflict-free I/O mechanism to fully exploit internal parallelism for Open-Channel SSDs[J]. Journal of Systems Architecture, 2023, 135: 102803.

[47] Chen Chemin Shih Y C, Liu Xin, et al. Implementing content-defined chunking for deduplication in host-managed SSDs[C]//2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). Piscataway: IEEE, 2024: 159-163.

[48] Peng Li, Wu Wenbo, Yi Shushu, et al. XHarvest: Rethinking high-performance and cost-efficient SSD architecture with CXL-driven harvesting[C]//Proceedings of the 52nd Annual International Symposium on Computer Architecture. New York: ACM, 2025: 434-449.

[49] Chen Junchao, Zhang Guangyan, Wei Junyu. A survey on design and application of open-channel solid-state drives [J]. Frontiers of Information Technology & Electronic Engineering, 2023, 24(5): 637-658.

[50] Bjørling M. From open-channel SSDs to zoned namespaces[EB/OL]. (2019-01-23) [2025-12-22]. https://www.usenix. org/sites/default/files/conference/protected-files/nsdi19_slides_bjorling.pdf.

[51] Bjørling M, Aghayev A, Holmberg H, et al. ZNS: Avoiding the block interface tax for flash-based SSDs[C]//Proceedings of the USENIX Annual Technical Conference. Berkeley: USENIX Association, 2021.

[52] Han K, Gwak H, Shin D, et al. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction[C]//Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2021.

[53] Tan Zhenhua, Long Linbo, Shen Jingcheng, et al. Optimizing garbage collection for ZNS SSDs via in-storage data migration and address remapping[J]. ACM Transactions on Architecture and Code Optimization, 2024, 21(4): 77.

[54] Wang Yu, Zhou You, Lu Zhonghai, et al. FlexZNS: Building high-performance ZNS SSDs with size-flexible and parity-protected zones[C]//Proceedings of the 41st IEEE International Conference on Computer Design. Piscataway: IEEE, 2023.

[55] Wu Denghui, Liu Biyong, Zhao Wei, et al. ZNSKV: Reducing data migration in LSMT-based KV stores on ZNS SSDs[C]//Proceedings of the 40th IEEE International Conference on Computer Design. Piscataway: IEEE, 2022.

[56] Lee H R, Lee C G, Lee S, et al. Compaction-aware zone allocation for LSM based key-value store on ZNS SSDs[C]//14th ACM Workshop on Hot Topics in Storage and File Systems. New York: ACM, 2022.

[57] Jung J, Shin D. Lifetime-leveling LSM-tree compaction for ZNS SSD[C]//14th ACM Workshop on Hot Topics in Storage and File Systems. New York: ACM, 2022.

[58] Liu Biyong, Xia Yuan, Wei Xueliang, et al. LifetimeKV: Narrowing the lifetime gap of SSTs in LSMT-based KV stores for ZNS SSDs[C]//Proceedings of the 41st IEEE International Conference on Computer Design. Piscataway: IEEE, 2023.

[59] Byeon S, Ro J, Han J Y, et al. Ensuring compaction and zone cleaning efficiency through same-zone compaction in ZNS key-value store[C]//Proceedings of the 38th International Conference on Massive Storage Systems and Technology. Piscataway: IEEE, 2024.

[60] Liu Gaoji, Yang Chongzhuo, Yu Qiaolin, et al. Prophet: Optimizing LSM-based key-value store on ZNS SSDs with file lifetime prediction and compaction compensation[C]//Proceedings of the 38th International Conference on Massive Storage Systems and Technology. Piscataway: IEEE, 2024.

[61] Wang Yu, Sun Zibin, Zhou You, et al. Balloon-ZNS: Constructing high-capacity and low-cost ZNS SSDs with built-in compression[C]//Proceedings of the 61st ACM/IEEE Design Automation Conference. New York: ACM, 2024.

[62] Kim T, Jeon J, Arora N, et al. RAIZN: Redundant array of independent zoned namespaces[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023.

[63] Yi Shushu, Sun Shaocong, Peng Li, et al. BIZA: Design of self-governing block-interface ZNS AFA for endurance and performance[C]//Proceedings of the 30th ACM SIGOPS Symposium on Operating Systems Principles. New York: ACM, 2024.

[64] Digital Western. Western Digital Ultrastar DC ZN540: Data sheet[EB/OL]. (2021-09-02) [2025-12-22]. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/data-sheet/data-sheet-ultrastar-dc-zn540.pdf.

[65] Samsung. Samsung introduces its first ZNS SSD with maximized user capacity and enhanced lifespan[EB/OL]. (2021-06-01)[2025-12-22]. https://semiconductor.samsung.com/news-events/news/samsung-introduces-its-first-zns-ssd-with-maximized-user-capacity-and-enhanced-lifespan/.

[66] Sabol C, Desai S. SmartFTL SSDs[EB/OL]. (2021-11-09-2021-11-10) [2025-12-22]. https://146a55aca6f00848c565-a7635525d40ac1c70300198708936b4e.ssl.cf1.rackcdn.com/images/c867f55eaa86f735dc82d649bd18077e9388f07f.pdf.

[67] Sabol C, Stenfort R. Flexible data placement mode (FDP)[EB/OL]. (2022-09-30)[2025-12-22]. https://nvmexpress.org/wp-content/uploads/Hyperscale-Innovation-Flexible-Data-Placement-Mode-FDP.pdf.

[68] Allison M, Rudelic J. Flexible data placement[EB/OL]. (2025-05-01) [2025-12-22]. https://snia.org/sites/default/files/2025-05/SNIA-SDC23-Allison-Rudelic-Flexible-Data-Placement.pdf.

[69] Manzanares A, Granados J, George A. Flexible data placement open source ecosystem[EB/OL]. (2025-12-04)[2025-12-22]. https://www.snia.org/sites/default/files/2025-05/SNIA-SDC23-Manzarnares-Granados-George-Flexible-Data-Placement.pdf.

[70] Allison M, George A, Gonzalez J, et al. Towards efficient flash caches with emerging NVMe flexible data placement SSDs[C]//Proceedings of the Twentieth European Conference on Computer Systems. New York: ACM, 2025: 1142-1160.

[71] Joshi K, Gupta A, González J, et al. I/O Passthru: Upstreaming a flexible and efficient I/O Path in Linux[C]//Proceedings of the 22nd Conference on File and Storage Technologies. Berkeley: USENIX Association, 2024.

[72] George A, Kumar V, Nair R, et al. Getting started with flexible data placement (FDP)[EB/OL]. (2024-02-07)[2025-12-22]. https://download.semiconductor.samsung.com/resources/white-paper/getting-started-with-fdp-v4.pdf.

[73] Samsung. PM9D3a SSD[EB/OL]. (2024-05-22)[2025-12-22]. https://semiconductor.samsung.com/news-events/tech-blog/samsung-pm9d3a-solid-state-drive/.

[74] DapuStor. Haishen5 SSD[EB/OL]. [2025-12-22]. https://www.dapustor.com/product/8.html.

[75] KIOXIA. 铠侠在2024年OCP全球峰会上展示运行RocksDB数据库、支持灵活数据放置的固态硬盘[EB/OL]. (2024-10-15)[2026-02-10]. https://www.kioxia.com.cn/zh-cn/business/news/2024/20241015-2.html.
KIOXIA. KIOXIA showcased an SSD running the RocksDB database with flexible data placement support at the 2024 OCP Global Summit [EB/OL]. (2024-10-15)[2026-02-10]. https://www.kioxia.com.cn/zh-cn/business/news/2024/20241015-2.html. (in Chinese)

[76] 浪潮信息. 寿命提升2.5倍!浪潮信息推出全新SSD, 实现FDP技术规模化应用[EB/OL]. (2025-06-23)[2026-02-10]. https://www.ieisystem.com/about/news/18948.html#.
Information Inspur. Inspur Information released a new SSD with a 2.5x lifespan extension, enabling the mass adoption of FDP technology[EB/OL]. (2025-06-23)[2026-02-10]. https://www.ieisystem.com/about/news/18948.html#. (in Chinese)

[77] 开放数据中心委员会. FDP SSD测试白皮书[EB/OL]. (2025-09) [2026-02-10]. https://www.odcc.org.cn/download/p-1970690010736570370.html.
Open Data Center Committee. FDP SSD Testing White Paper[EB/OL]. (2025-09) [2026-02-10]. https://www.odcc.org.cn/download/p-1970690010736570370.html. (in Chinese)

[78] Park J, Kim H, Ha J, et al. FDPVirt: Flexible data placement SSD emulator[C]//IEEE International Conference on Cluster Computing Workshop. Piscataway: IEEE, 2024.

[79] Kim S H, Shim J, Lee E, et al. NVMeVirt: A versatile software-defined virtual NVMe device[C]//Proceedings of the 21st USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2023.

[80] Chen Pingxiang, Seo D, Dutt N. FDPFS: Leveraging file system abstraction for FDP SSD data placement[J]. IEEE Embedded Systems Letters, 2024, 16(4): 349-352.

[81] Gonzalez J. Data placement at scale: Landscape, trade-offs, and direction[C]//Proc of Linux Plumbers Conference. 2024.

[82] Li Wenjie, Chen Xiang, Shan Yelin, et al. StreamCSD: SSD-autonomous stream management via in-storage content learning[C]//Proceedings of the 62nd Annual ACM/IEEE Design Automation Conference. New York: ACM, 2025.

[83] 康海燕, 张义钒, 王楠敏. 基于联邦大模型的网络攻击检测方法研究[J]. 电子学报, 2025, 53(6): 1792-1804.
Kang Haiyan, Zhang Yifan, Wang Nanmin. Research on network attack detection method based on federated large model[J]. Acta Electronica Sinica, 2025, 53(6): 1792-1804. (in Chinese)

[84] 黎广镕, 李广军, 尚晶, 等. 基于大模型辅助的云边协同工作流调度算法[J]. 电子学报, 2025, 53(9): 3060-3077.
Li Guangrong, Li Guangjun, Shang Jing, et al. Large language model-assisted cloud-edge collaborative workflow scheduling algorithm[J]. Acta Electronica Sinica, 2025, 53(9): 3060-3077. (in Chinese)

[85] Patel T, Byna S, Lockwood G K, et al. Revisiting I/O behavior in large-scale storage systems: The expected and the unexpected[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Piscataway: IEEE, 2019: 1-13.

[86] Lewis N, Bez J L, Byna S. I/O in machine learning applications on HPC systems: A 360-degree survey[J]. ACM Computing Surveys, 2025, 57(10): 256.

[87] Yuan Ziqi, Zhang Haoyang, Zhou Y E, et al. Cost-efficient LLM training with lifetime-aware tensor offloading via GPUDirect storage[PP/OL]. V1.arXiv (2025-06-06)[2026-02-13]. https://arxiv.org/abs/2506.06472.

作者简介

吴秋霖 男, 1995年7月出生于湖北省黄冈市。现为国防科技大学计算机学院博士后。主要研究方向为固态存储系统、文件系统。
E-mail: qiulin_wu@nudt.edu.cn

袁 远 男, 1980年11月出生于湖南省长沙市。现为国防科技大学计算机学院研究员。主要研究方向为高性能计算、智能运维和并行存储。
E-mail: yuanyuan@nudt.edu.cn