doi:10.3772/j.issn.1002-0470.2023.01.005

基于闪存固态盘的存储系统性能优化关键技术综述①

李天祥②[*,**]　蒋德钧③[*,**,***]　熊　劲[*,**]

([*]中国科学院计算技术研究所计算机体系结构国家重点实验室　北京 100190)

([**]中国科学院大学　北京 100049)

([***]鹏城实验室　深圳 518055)

摘　要　闪存固态盘(SSD)具有高并行性、复杂的内部事务以及具有计算能力等特性。现有的存储系统是针对磁盘设计的,无法充分利用闪存固态盘的性能。本文介绍闪存固态盘给现有存储系统带来的问题与挑战,并从提高系统总带宽、减少固态盘内部事务对性能的影响、性能服务质量保证与利用固态盘的计算能力 4 方面阐述其性能优化关键技术,最后讨论了基于闪存固态盘的存储系统的发展方向。本文归纳指出,软硬件协同的设计方式是优化基于闪存固态盘的存储系统的发展趋势,软硬件协同设计有助于提供可预测的性能及可控的端到端延迟。

关键词　闪存固态盘(SSD);输入输出(IO)性能;垃圾回收;服务质量保证;计算任务卸载

0　引　言

固态盘(solid state drive, SSD)已经成为主流存储设备被广泛应用。然而,传统存储系统是针对磁盘特性设计的,因此固态盘为了兼容当时的系统软件,采用了块设备接口。但是随着固态盘相关技术的发展,固态盘设备的带宽不断升高、延迟不断降低,传统存储系统没有考虑固态盘的特性,系统软件的瓶颈逐渐暴露出来,包括 Linux 块层、文件系统、应用程序等层次。(1)由于磁盘采用机械臂结构,只有柱面级的并行,因此 Linux 块层为了便于针对磁盘特性调度请求,采用了单队列块层的设计。这种单队列模式的优势是便于对请求进行集中控制,但发送与提交请求都需要加锁,限制了软件的可扩展性,无法充分利用固态盘的并行性。(2)同理,文件系统中的元数据管理与日志的集中式设计成为了瓶颈。(3)日志式文件系统与键值存储系统广泛使用的日志树(log-structured merge tree, LSM)[1]最初针对磁盘随机性能差的特点设计,将随机输入输出(input/output, IO)转换成顺序 IO,然而,这与固态盘内部的异地更新功能重复,LSM 结构合并(Compaction)带来的写放大也加速了固态盘的老化。

1　相关工作

闪存固态盘相对于传统磁盘具有速度快、延迟低的特点,二者主要有以下 3 点区别,即组成结构不同、介质特性不同和接口协议不同,下文分别从这 3 个方面介绍闪存固态盘的特点。

(1)固态盘内部全部由电路组成,包括缓存、闪存控制器、闪存颗粒及主控芯片片上系统(system on chip, SoC)等。根据闪存芯片的不同可以分为单位单元(single-level cell, SLC)、双位单元(multi-level

① 中国科学院战略性先导科技专项(XDB44030200),北京市自然科学基金-海淀原始创新联合基金(L192038),中国科学院青年创新促进会和鹏城实验室重大项目(PCL2021A08)资助。

② 男,1992 年生,博士生;研究方向:计算机系统结构;E-mail: litianxiang@ict.ac.cn。

③ 通讯作者,E-mail: jiangdejun@ict.ac.cn。

(收稿日期:2021-11-24)

cell,MLC)、三位单元(tripe-level cell,TLC)、四位单元(quad-level cell,QLC)及 3D Xpoint[2] 等,为了实现更高密度的存储容量,目前 SSD 厂商多采用 3D 堆叠技术提高单个存储颗粒的容量。与磁盘机械臂结构不同,固态盘具有数据处理的并行性,包括路(Plane)、逻辑单元(logical-unit number, LUN 也称 Die)、通道(Channel)级别的并行[3],如图 1 所示。闪存芯片(flash chip)由多个 Die 以 3D 结构封装而成。闪存芯片内的多个 Die 共享芯片使能总线(chip enable-bus),Die 共享同一地址总线。通常 Die 是固态盘内部可以独立执行指令的并行单元。通常 Die 由多个 Plane 构成,多个 Plane 内多个 Plane 的同一地址的页组成大页,作为闪存芯的最小读写单元。闪存芯片共享闪存 Channel,连接到闪存控制器上,多个 Channel 之间可以独立执行指令、数据,互不干扰。

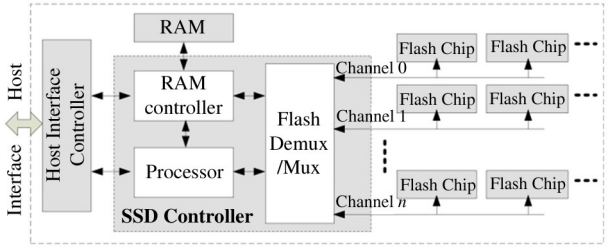

图 1　SSD 内部结构示意图[4]

(2) 闪存固态盘由闪存组成,与传统磁盘的磁性介质特性不同,具有擦后写、读写与擦除粒度非对称和可擦除次数(寿命)有限等特点。由于介质擦后写、页粒度读写与块粒度擦除的特点,固态盘采用异地更新的方式写入新数据,采用块粒度的垃圾回收(garbage collection,GC),同时为了缓解由于空闲空间不足导致的性能降低,固态盘通过预留空间(over provisioning, OP)降低垃圾回收时的性能影响。由于介质有限的寿命,固态盘要进行颗粒间的磨损均衡(wear-leveling),使固态盘不会因为部分颗粒过度老化导致可用空间的降低。因此,固态盘在固件中增加了地址转换层(flash translation layer, FTL),负责地址映射、垃圾回收以及磨损平衡等功能。

(3) 为了提高兼容性,标准固态盘通常采用块设备(block device)接口,以连续逻辑地址空间的形式暴露给主机端(Host)。标准闪存固态盘与传统磁盘的软件接口都为块设备,但物理电气接口有所不同,主要可以分为串口协议(serial advanced technology attachment,SATA)与高速外围设备协议(peripheral component interconnect express, PCIe)2 大类,SATA 由于接口物理速度限制为 6 Gb/s,多用于低端固态盘,高端企业级固态盘多采用基于 PCIe 的非易失性内存协议(non-volatile memory express, NVMe)[5]。固态盘从闪存单元、芯片封装工艺、固件以及接口协议等多个维度不断发展,逐渐形成了如表 1 所示的主流标准商品化固态盘产品,这些不同价格、性能、容量和寿命的固态盘促进了系统软件的技术变革。

表 1　数据中心固态盘规格[6]

	傲腾 NVMe	高性能 NVMe (Intel D7)	大容量 NVMe (Intel D5)	大容量 SATA (Intel D3)
单元类型	3DXP	TLC	QLC	TLC
寿命 (P/E Cycles)	1M – 3M ~ 2M	500 – 2k ~ 1k	100 – 1k ~ 0.5k	500 – 2k ~ 1k
价格($/GB)	<1.00	<0.20	<0.10	<0.10
容量	58 GB ~ 1.5 TB	1.6 ~ 7.68 TB	7.68 ~ 30.72 TB	240 GB ~ 7.68 TB
顺序读写带宽(MB/s)	2500/2200	3500/1700	3200/1000	560/510
随机 kIOPS	550/550	400/118	427/36	97/32

由于固态盘的高性能、低延迟、以及复杂的内部事务的特点,针对传统磁盘设计的存储系统难以充分利用固态盘的性能,给基于固态盘的存储系统设计带来了以下 4 点挑战:1)高性能的固态盘对传统 Linux 内核 IO 栈的请求处理效率带来了新的挑战,如内核 IO 栈中单队列块层的可扩展性、块层的请求合并操作的开销,文件系统中集中式数据结构与日志的资源竞争,设备驱动层基于中断的请求处理模

式等。因此一些工作利用使系统软件可扩展、(部分)绕过内核及轮询(Polling)等技术降低内核的开销,另外一些工作采用用户态 IO 的方式完全绕过了内核。2)固态盘内部的垃圾回收、磨损平衡等功能对用户 IO 性能产生影响,且标准固态盘的块设备接口语义使固态盘内的数据放置不可控,促使软件与接口进行协同设计。3)存储系统有端到端延迟控制及差异化服务质量保证的需求。然而,固态盘的垃圾回收与读写干扰,使基于固态盘的存储系统难以保证服务质量。4)新的接口协议的出现使固态盘 FTL 的功能被(部分)上移至主机端,固态盘的计算能力被释放出来,如何将计算任务卸载到盘内执行是软硬件协同的固态盘系统设计的挑战。

综上所述,为解决以上这些挑战,人们做了大量工作,本综述只关注存储系统性能优化方面的工作,这些现有闪存固态盘的存储系统性能优化工作主要可分为 4 类:(1)提高系统总带宽;(2)减少固态盘内部事务对性能的影响;(3)固态盘的性能服务质量保证;(4)利用固态盘的计算能力。

2 提高系统总带宽

固态盘具有高读写带宽、低延迟的特点,针对磁盘设计的现有软件系统存在一些限制:(1)随着固态盘速度增长,软件开销占 IO 总时间的比重越来越大;(2)无法充分利用固态盘内部的并行性;(3)集中式数据结构带来竞争;(4)不充足的接口语义引起的软硬件多个层次的功能重复。

2.1 轻量级的 IO 软件栈

降低软件栈的开销主要有 2 种方式:一是采用绕过内核 IO 栈,直接在用户态与设备交互的方式处理 IO 请求;二是降低内核软件栈处理固态盘 IO 请求的开销,降低系统软件开销在 IO 总时间的比例。

绕过内核的主要技术是存储性能套件(storage performance development kit,SPDK)[7]——用户态的 NVMe 驱动,它利用用户态 IO(userspace IO, UIO)与大页内存技术,绕过内核直接在用户态向设备发送请求与处理请求完成。它采用了轮询驱动(poll mode driver,PMD)技术,相比于内核驱动基于中断的处理方式能够显著降低请求完成的处理时间。目前 SPDK 的软件生态在逐渐完善中,增加了逻辑卷管理、用户态块层、BlobFS 等功能。然而,SPDK 组件的更新迭代,更像是在用户态中重新搭建内核栈中已经完善多年的组件,追求全栈用户态技术而得到的性能收益是否符合预期值得质疑。阻碍用户态 IO 发展的一个很大的因素是缺乏通用的用户态文件系统,应用程序需要为用户态 IO 适配而改写,但代码改写与维护的人力成本较高。SPDK 中基于 BlobStore 的 BlobFS 最初是为 Ceph[8] BlueStore 及 RocksDB 而设计的,仅兼容有序表(sorted string table, SSTable)的追加写,缺乏文件系统的通用性。EvFS[9]、FSP[10]和 uFS[11]等工作基于 SPDK 构建可移植操作系统接口(portable operating system interface, POSIX)用户态文件系统,其中 EvFS 采用文件系统库的形式嵌入在应用程序中的架构,FSP 及其后续工作 uFS 采用类似分布式系统中客户端-服务器模型的架构进行设计。这种架构的主要挑战是如何降低用户态文件操作及数据传输与文件系统进程之间交互的开销。对此,FSP 及 uFS 采用了类似 NVMe 协议发送队列(submission queue, SQ)与完成队列(completion queue, CQ)的无锁环形队列设计,在应用程序进程与文件系统进程间利用基于共享内存的请求交互方式,降低数据交换开销。目前,所有用户态文件系统的权限控制都依赖于从内核中获取,但是权限操作仅在初始化中进行,不占用 IO 关键路径,对性能的损耗可以忽略不计。

另一方面,为了利用内核的便利,一些工作降低基于固态盘的内核系统开销。Shin 等人[12]针对固态盘优化了内核 IO 上下文处理流程,使处理 IO 发送与完成在同一个中央处理器(central processing unit, CPU)核中进行,同时采用硬中断与异步轮询的方式代替了小型计算机系统接口(small computer system interface,SCSI)使用的软中断,另外 NVMe[3]采用了消息信号中断扩展协议(message signaled interrupts extended, MSI-X)方式处理中断。AsyncIOS[13]使内核处理与 IO 执行异步重叠,绕过内核块层以进一步降低延迟敏感型负载在内核中的处理时间。IO _uring[14]是一种新型的内核异步 IO 系统调

用接口,通过共享设备队列到用户态,减少了因系统调用频繁陷入内核的开销,同时在内核一侧采用 Polling 技术降低高性能固态盘的内核开销。FlashShare[15]为延迟敏感型负载绕过内核块层同时应用 Polling 技术加速请求处理,保留带宽敏感型负载为原有 IO 栈逻辑。Polling 技术会使 CPU 空转消耗时钟资源,基于中断的处理方式的上下文切换与中断处理增加了请求延迟,目前最新技术趋向于两者相结合,达到优势互补的效果。

总结来说,用户态的方式绕过内核的方式去掉了内核软件栈开销,提高了 IO 处理效率,但是没有了内核的块设备与文件系统支持,应用需要进行适配性修改。因此,另外一部分工作为了利用内核提供的便利,针对高速固态盘优化了内核 IO 路径。两方面工作都是绕过或部分绕过内核,并利用大页内存或共享内存技术直通设备请求处理,同时采用 Polling 技术降低延迟敏感型负载的延迟。

2.2 利用固态盘设备内部的并行性

利用硬件的并行性工作集中在 3 个方面:(1)提高软件的并行度;(2)发大块请求以利用 SSD 的并行性;(3)向主机端暴露并行单元的固态盘,包括开放通道固态盘(open channel SSD, OCSSD)[16]和对象接口固态盘(zoned namespace SSD, ZNS)[17]等。

NVMe 是为高速固态盘设计的基于 PCIe 的接口协议,协议定义在闪存控制器中实现多设备队列,以利用固态盘的并行性。NVMe 协议也提供了命名空间(Namespace)对空间进行逻辑或物理划分,以提高并发或隔离。

AMF[18]与 RIPQ[19]采用了粗粒度分配的方式以利用并行性,按涵盖所有闪存通道的超级块粒度进行分配,目的是让同一个超级块中的数据覆盖所有通道级并行单元,以此来利用固态盘的并行性。

SDF[20]直接将 LUN 暴露给主机端,每个 LUN 作为一个单独块设备使用。传统块设备对主机呈现一个或多个逻辑块设备 sdx,而 SDF 对主机端呈现出多个物理隔离的块设备/dev/sdx{1..n}。SDF 的 FTL 采用静态块级映射,只负责磨损平衡、纠错码(error correcting code,ECC)及坏块管理,将垃圾回收交给主机端软件,因此在设备内不再为垃圾回收预留空间,释放了标准盘 OP 空间,增大了用户可利用空间。LOCS[21]是基于 SDF 构建的键值存储系统,通过多个 Compaction 线程分别对应不同物理单元的方式利用 SDF 的并行性。ParaFS[22]通过增加新的接口,向主机端暴露固态盘的硬件布局,采用大块分配与物理地址访问的方式以充分利用固态盘的各个并行单元的并行性。

总结来说,一类工作试图增大 IO 粒度来充分利用固态盘的并行性,另一类工作通过新接口暴露硬件并行性,并交给软件直接管理固态盘。前者只能影响写请求的处理,如果读请求的数据具有倾斜性,则无法充分利用硬件的并行性。软件控制的数据布局方式可以对上述缺点进行协同优化,并已经成为发展趋势。

2.3 系统软件的可扩展性

固态盘更高的吞吐性能导致系统软件中的资源竞争开销更明显,软件的可扩展性指的是系统软件性能随 CPU 核心数增长而增长的能力,针对软件可扩展性的优化工作主要包括块层与文件系统层 2 个层次的工作。块层优化的目标是充分利用裸设备的性能,文件系统的优化工作是在块层的基础上进一步减少 IO 软件栈中的集中控制。

Blk-mq[23]采用块层多队列设计代替了集中式单队列的块层架构。mq 由软件队列与硬件队列组成。软件队列是每 CPU 队列,每个 CPU 核心与固定的块层软件队列绑定,CPU 核心可以无锁地提交 IO 请求到对应的队列。硬件队列数由设备暴露,例如 NVMe 设备队列数为队列对(queue pair,Qpair/QP)[3]数目。软件队列与硬件队列采用一一映射的方式,当硬件队列数目比 CPU 核数少时,多个软件队列会被映射到同一条硬件队列,硬件队列采用先进先出的方式向设备提交请求。同时这种无锁的多队列块层给 IO 调度带了新的挑战,本文将在第 4 节中具体介绍。

SpanFS[24]将本地文件系统中日志缓存、校检点(Checkpoint)列表等集中共享的数据结构进行划分,使每个 CPU 对应自己的一部分文件系统数据结构。每个部分独立进行日志操作,各个部分间采用

两阶段提交的方式维护一致性。

总结来说,提高系统软件的 CPU 核可扩展性的主要技术手段是利用每 CPU 数据结构来减少锁的使用,并利用比较后交换(compare and swap,CAS)指令和租约(lease)等技术降低保证一致性的必需同步开销。

2.4 消除软硬件层的重复功能

由于标准固态盘的块设备接口的语义不足,使系统软件无法感知固态盘内部的异地更新与复杂的后台事务等特性,导致软件和硬件之间存在功能重复。例如,日志结构文件系统以及基于 LSM 的系统采用追加写的方式更新数据,与固态盘内部的 FTL 的以异地更新的方式写入数据在功能上是重复的。

Nameless write[25]扩充了块设备接口语义,由固态盘设备返回写入地址代替了标准块设备的传入写入地址的写入方式,以解决目前文件系统与固态盘每一层都要进行地址转换的功能重复的问题。相似地,FlashTier[26]通过增加缓存接口,减少固态盘缓存场景中数据查找的地址转换次数。

AMF[18]解决日志结构文件系统与 FTL 的多重日志(Log on Log)问题,它采用静态块映射 FTL 与限制仅异地更新的块接口相结合的技术,使逻辑块与物理块一一对应,因此只需一层软件控制的软硬件协同异地更新,从而只需保留文件系统一层的垃圾回收。

总结来说,目前文件系统与设备中都存在地址转换,如果要消除一层重复的功能方法只有 2 种:一是只在设备内进行地址转换,让软件被动接收写入地址;二是只在软件中进行地址转换。目前软硬件的发展趋势是协同设计及更细致的功能划分,以消除功能上的重复。

2.5 小 结

以上各工作利用不同技术提高基于固态盘的系统性能,包括降低软件栈开销、减少软件中的资源竞争、利用设备的并行性以及定制设备对主机端呈现的接口协议与写入粒度。这些工作将上述技术实现在 IO 路径的各个层次中,如表 2 所示。

不同层次的优化技术是可以叠加的,例如 blk-mq 与 NVMe。软件栈优化技术的核心是如何利用固态盘的并行性。高并行下集中式资源竞争是系统的瓶颈,解决资源竞争的同时也是在更好得利用设备的并行性。同时无论设备暴露给主机端的接口是开放通道或物理布局,还是标准块接口下加大写入粒度,其目的都是充分利用固态盘的并行性。

表 2 最大化固态盘性能相关工作总结

工作	降低软件开销	降低资源竞争	利用设备并行	设备接口	读写粒度	实现层次
I/O path	是	否	否	标准块设备	页	块层,驱动层
SPDK	是	否	是	标准块设备	页	用户态
NVMe	否	否	是	标准块设备	页	块层,驱动层,FTL
AMF	否	否	是	物理地址块设备	超级块	文件系统,驱动层,FTL
RIPQ	否	否	是	标准块设备	超级块	块层
SDF	否	否	是	通道	擦除块	块层,FTL
ParaFS	否	否	是	物理地址块设备	页	文件系统,FTL
blk-mq	否	是	是	标准块设备	页	块层
SpanFS	否	是	是	标准块设备	页	文件系统
Nameless	否	否	否	定制块设备	页	块层,驱动层,FTL

3 减少固态盘内部事务对性能的影响

固态盘内部有许多的后台事务,如垃圾回收、磨损平衡与数据校验等,其中最耗时的为垃圾回收与磨损均衡,本节着重介绍降低这两种内部事务的开销,以提高系统性能的工作。

3.1 减少垃圾回收对性能的影响

闪存单元具有擦后写的特性,固态盘采用异地

更新的方式隐藏擦除开销,但需要进行垃圾回收清理无效数据,并通过预留空间 OP,降低有效空间不足导致的性能降低。垃圾回收通常需要 3 个阶段,即擦除候选块选择、拷贝有效数据以及擦除。减少垃圾回收对性能影响的工作主要可以分为 2 类:(1)基于数据分类的垃圾回收;(2)软件控制的垃圾回收。

3.1.1 基于数据分类的垃圾回收

数据分类技术包括 2 类:(1)存储系统自动对数据进行冷热识别与分类;(2)软件定义的数据分类,由软件直接定义数据冷热并通过特殊接口传递给设备。

自动数据分类包括 FTL 与系统软件两类的分类技术。这类工作的主要特点是通过数据分类,降低垃圾回收中有效数据的拷贝,进而降低垃圾回收对性能的影响。FTL 区分数据冷热是常见技术,通常认为垃圾回收中拷贝的有效数据为冷数据。DAC-FTL[27]根据页面更新频次来聚合数据,替代了传统策略在垃圾回收时区分冷热数据。Delta-FTL[28]采用增量存储数据的方式,例如,一个新的写请求的数据内容一定程度上与旧版相似,Delta-FTL 只存储增量数据,而非原始数据。F2FS[29]在日志式文件系统中以擦除块对齐方式识别并根据数据的冷热程度聚合数据。但是目前标准固态盘常见的页粒度映射 FTL,使得在软件层聚合的数据又被分散到各个物理块中,打破了在文件系统层的布局,因此针对标准固态盘的主机端的数据放置优化很难达到预期的效果。RIPQ[19]与 Pannier[30]使用由覆盖所有并行单元的擦除块组成的超级块为粒度布局数据。以区分热度的超级块粒度缓存数据,并一次性写入到固态盘中,目的是让逻辑上聚合的数据在物理上还会存放在一起。但是这种设计只适用于缓存,因为超级块需要一次性写入设备,在内存中聚合成超级块时,可能会造成数据丢失,是持久化存储场景不能容忍的,同时如果聚合的数据不足覆盖所有并行单元,则不能达到预期的效果。

软件定义的数据分类通过给固态盘增加一些新的接口语义,以从主机端获取更多的信息,并将具有相同属性的数据物理上放置在一起,进而降低垃圾回收时数据的拷贝量。OFTL[31]提供对象接口,并构建了基于对象的 FTL,使固态盘能感知更多用户数据的更多信息,因此能更好地控制数据放置与处理垃圾回收。Multi-stream[32]通过特殊的块接口来向设备传递流标签,多流固态盘将具有相同标签的数据在物理上放置在一起。多流固态盘的难点是如何识别出应用中不同热度的流。FSstream[33]通过将 F2FS 的日志编号传递给多流固态盘的方式,来达到软硬件协同的数据放置效果。PCstream[34]利用程序的进程上下文识别来自不同程序的 IO 流,并将流信息标签传递给多流固态盘。由于缺乏一定的业界标准,这些软件定义的接口虽功能相近但各不相同。

总结来说,仅使用存储系统自动分类的技术可能无法达到预期的效果,使用软件定义的数据分类技术可以更好地在物理上聚合冷热数据,但通过标签传递控制数据放置的方法只能影响数据的初始放置,并不能完全控制固态盘对垃圾回收事务的处理。

3.1.2 软件控制的垃圾回收

由于设备与系统软件互不感知,设备进行垃圾回收时,回收的有效数据可能已经被软件标记为无效数据,但设备依然当做有效数据进行回收。为了这种多重日志问题,一些工作采用软件控制垃圾回收的方式,在主机端进行垃圾回收以降低多重回收对性能的影响。在主机端进行垃圾回收的前提是主机端感知固态盘的物理布局,即设备向主机端暴露出的逻辑地址能与物理块一一对应。同时,感知布局也能更好地将数据根据语义聚合并在物理上放置在一起,以降低垃圾回收有效数据的拷贝。

AMF[18]与 ParaFS[22]采用软件控制垃圾回收的方式,通过块级静态块映射 FTL 技术,使逻辑块与物理擦除块一一对应,使对逻辑块的回收等同于对物理块进行回收。这 2 个工作在文件系统层次都采用异地更新的日志式文件系统,垃圾回收为日志中的有效数据回收,消除了文件系统与设备同时执行垃圾回收的功能冗余。同时,2 个工作都针对固态盘的请求调度进行了优化,其中 ParaFS 在主机端对 IO 请求与擦除请求进行调度,AMF 在设备端调度读写与擦除请求,目的都是能够提供一致的性能表现,减少性能波动。

总结来说,这类工作的区别在于如何使用软件来协同管理设备,AMF 采用非异地更新接口,而 ParaFS 采用物理地址块设备来让软件层直接控制物理数据布局。但是软件控制的垃圾回收操作需要从设备读取有效数据到主机端,再写回到固态盘中,占用了设备通道带宽。ZNS +[35] 通过硬件手段增加了设备内拷贝有效数据的接口支持,减少了从设备到主机端的双向流量占用。

3.1.3 小 结

目前针对固态盘垃圾回收优化的研究工作的发展趋势是软硬件协同垃圾回收,目前标准固态盘 NVMe 协议在推动的 ZNS 接口,也采用了主机端软件控制垃圾回收的方式。这种方式可以更好地根据应用定义的语义信息聚合数据放置、消除软件与设备的功能冗余,同时软件控制的垃圾回收也提供了设备延迟可控的可能。同时,系统软件与固态盘硬件也在不断磨合、细致划分彼此的功能,并互相配合。

3.2 减少磨损均衡对性能的影响

磨损均衡的目的是平衡每个块的擦除计数,使闪存颗粒同时老化,来防止一些颗粒过早磨损导致整个固态盘容量减少。磨损平衡作为固态盘内部事务,同样对设备的性能产生影响。其工作主要分为两类,一类是动态磨损均衡,另一类是静态磨损均衡。

动态磨损均衡只管理分配没有任何数据迁移。文献[36]提出了一种自适应条带化数据到各个闪存通道中擦除块的算法,该算法将热的数据分配到具有最少擦除次数的闪存通道的擦除块中。

静态磨损与动态相反,具有数据迁移。Rejuvenator[37] 是一个静态磨损均衡算法,该算法将冷数据迁移到磨损更严重的块中,同时维持所有擦除块的擦除次数近似相等,以此来整体延长固态盘的寿命。FlashBlox 与 AutoSSD[38] 通过推迟磨损平衡任务的执行,降低磨损平衡对固态盘系统性能的干扰。

总结来说,磨损平衡技术与其他 IO 关键路径开销优化工作是正交的,通常磨损平衡在设备 FTL 中进行,对主机端透明。通过软硬件协同设计的方式,可以推迟磨损平衡任务的进行,以降低对性能的影响。在闪存阵列系统中,磨损均衡同样重要,常见技术为全局的滑动窗口式[39] 的数据分配。此外,文献[40]更为详尽地介绍了延长固态盘使用寿命的相关技术。

4 固态盘的性能服务质量保证

一些系统通过公平调度、隔离等技术保证公平性、最后期限,使固态盘提供不受垃圾回收等内部事务影响的稳定性能。这些系统采用对固态盘性能建模的方式推测 IO 处理是否触发垃圾回收,是基于标准固态盘服务质量(quality of service, QoS)保证的主要技术手段。另外一些系统通过对设备模型更严格的限制或者采用软硬件协同的设计方法做到让固态盘保证严格服务级别目标(service-level objective, SLO)。

4.1 公平性

比例公平是指所有应用(租户、IO 线程)成比例共享固态盘的性能,特例是,如果所有的租户具有相同的优先级则是完全公平。固态盘的读写性能有很大的差异,同时具有读写混合干扰的特点,然而,Linux 通用块层调度请求不区分对待读写,造成租户间请求处理的不公平现象。

FIOS[41] 采用基于 IO 时间片的管理策略保证公平性,并通过优先调度读请求的方式降低读写干扰的影响,同时解决针对磁盘设计的完全公平排队(complete fair queuing, CFQ)类算法对固态盘的假空闲不友好的问题。FlashFQ[42] 在 FIOS 的基础上,将基于时间片分配的公平保证策略改成了基于开始完成时间(start finish time)的算法,以满足多个任务间的公平而不是单个时间片的公平。SSD CFQ[43] 更进一步地区分了元数据与数据队列,并通过轮询的方式保证公平性。

公平调度的另外一个难点在于优先级反转问题的处理。Split-level I/O[44] 识别出 CFQ 类调度器存在优先级翻转的现象,并最终发现页面缓存及日志刷回是造成优先级翻转的原因。该工作采用系统调用、文件系统、块层协同的方式,传递优先级信息,协同调度来消除优先级翻转,最终实现了公平调度。

除此之外,设备内的优先级调度同样重要。D2FQ[45]利用 NVMe 的带权轮转调度(weighted round-robin,WRR)实现公平性,并利用基于请求虚拟时间的算法,解决了 WRR 默认只按请求数调度并执行命令而不考虑请求大小的问题。

总结来说,保证成比例共享设备性能是云存储场景的普遍需求,主要技术手段包括标签传递以及根据优先级成比例向设备发送请求。但由于固态盘的并行性,仅在块层调度做限制可能会无法达到预期的效果,还需要借助于硬件级别的公平性调度手段,如 NVMe WRR。另外,多队列块层没有了集中控制点,对块层调度器的设计带来了新的挑战。MQFQ[46]在每个 CPU 维护了一组保证公平性的队列,通过让每个核都保证公平性的方式来保证全局公平性,但增大了软件栈开销。

4.2 保证最后期限

最后期限是指保障每个请求的处理时间都不超过给定期限,也就是所有请求处理的尾延迟在给定期限内。最后期限(Deadline)最初是为解决直通(no operation,NOOP)调度会产生饥饿问题的块层 IO 调度器,优先调度即将超时的请求。最后期限所能保证的处理时间与最大尾延迟相同,比保证百分位尾延迟更严格,但保证的时间会比保证尾延迟的宽松。目前保证最后期限的工作大多源自 Linux 块层 Deadline IO 调度器。

HIOS[47]通过特殊的块接口从应用层传递延迟保障信息直到设备。设备使垃圾回收阶段化,并将每阶段的开销分摊到普通请求中,使每个请求都不超出它的最后期限。但硬件功能实现复杂,使之难以产品化。Split-level I/O[45]通过更细致的标签传递的方式,使最后期限目标不因切换异步处理线程而丢失。

总结来说,由于在软件层保证最后期限的策略保证请求处理时间的能力有限,要保证严格的最后期限需要硬件级的特殊支持。

4.3 隔 离

隔离是解决干扰的一种方法,通过用户数据的物理隔离可以消除租户间的彼此干扰。

OPS Isolation[48]发现由于垃圾回收时有效数据的拷贝会使被隔离的虚机数据再次混合,因此有时满足不了 SLO。所以它将固态盘的预留空间进行划分,预留给不同的虚机,并保持垃圾回收前后虚机的数据都在自己的隔离单元中。FWC[49]根据租户划分 Cache 空间包括数据空间与 OP 空间,多个租户彼此独立进行垃圾回收,互不干扰。此类工作根据数据语义不同提供隔离,思想大致相同。

LOCS、ParaFS 和 AMF 都通过软件控制垃圾回收,在软件中拷贝数据与向设备发送擦除命令。因此设备只包含读、写、擦除 3 种请求,不再有垃圾回收操作。同时这些工作合理调度写与擦除请求,使各并行单元的性能更一致。这类工作将数据分配与垃圾回收在功能上隔离。

更进一步地,DC-Store[50]提供 3 个层次的全栈隔离:(1)基于 NVMeSet 的物理数据隔离;(2)对设备内部的 CPU、DRAM 等资源的隔离;(3)在主机端采用 docker 与 Cgroup 结合的隔离,做到了端到端的数据隔离。

总结来说,软件隔离技术有 Cgroup 与 blk-mq 或 SPDK 结合的方式,可以实现从 CPU 绑核隔离到块层、设备驱动层的隔离,但不足是无法做到数据的通道级别的物理隔离以及设备内资源的隔离。因此如果要想做到更高级别的全栈隔离,只采用软件隔离技术是不够的,需要设备的支持。

4.4 保障 SLO

保障 SLO 是指保障租户特定的 SLO 而不是保证整体性能一致的工作。保障的 SLO 包括性能指标如带宽、IOPS、延迟、利用率等。这些工作尽可能在更多的层次间进行控制,以达到更好的效果。

第 1 类工作是动态反馈与软硬件协同。OIOS[51]在虚机监控器中加入服务级别共识(service-level agreement,SLA)管理器和 IO 请求执行反馈控制器,这些控制器感知各个虚拟机的性能需求。该工作提出了一种新颖的基于差异允许时间的请求调度算法用于 SLA 管理器对不同类型的指标如带宽、延迟、IOPS 的保障。反馈控制器动态监控固态盘的实时性能信息,传递给 SLA 管理器。该工作结合以上 2 个部件来协同满足 SLO。VSSD[52]认为排队时延与垃圾回收是导致固态盘仅通过隔离无法满

足 SLO 的原因。该工作提出了一种基于信用的 IO 公平调度器以及感知存储虚拟化的 ViSA FTL。通过软硬件的协同设计,满足每一个虚拟机的 < 性能、空间、预留空间 > 三元组构成的 SLO 目标。

第 2 类工作是基于性能曲线推测设备实时性能。ReFlex[53]是针对计算与存储分离场景的多租户存储系统,目标是保证 95% 分位尾延迟的 SLO。该工作预先绘制不同读写比例与尾延迟、吞吐率的性能曲线,并根据系统运行状态下的读写比例所对应的能满足尾延迟需求的最大发送速率,然后通过令牌桶的方式进行流量控制,该工作巧妙地通过将延迟需求转换成流量控制的方式保障了尾延迟。该工作的主要不足是,令牌的分配以系统运行时最差读写比例下的设备稳态性能为基准进行限流,严格限制浪费了设备的性能。SSDCheck[54]通过对黑盒 SSD 建模预测 GC,当预测到 GC 触发时,缓存写入的数据,来提高 SSD 稳态的性能,同时保证租户的尾延迟不受垃圾回收影响。TimeSSD[55]通过在设备内部保存数据的访问历史,预测垃圾回收发生时的开销,根据历史信息决策数据分配,从数据分配的时机选择消除垃圾回收对请求处理的影响,并根据模型预测保证延迟。

第 3 类工作是对设备内部事件建模。AutoSSD[33]将 SSD 内部的 IO 与后台操作抽象成事务,根据用户请求的性能需求来计算每个事务处理占用的令牌数,该工作在其他工作仅考虑垃圾回收的干扰的基础上,考虑了磨损平衡、数据校验等固态盘内部事务对 IO 请求处理的影响。

总结来说,目前保证 SLO 的工作依赖设备建模或动态反馈前一时刻性能的方式,在软件层调度请求以期望满足 SLO 目标。模型对设备性能曲线刻画的精准度决定了能保证的尾延迟的能力,而对设备内部事务透明导致能力有限。

4.5 小 结

现有基于固态盘 QoS 保证的代表工作如表 3 所示,主要从解决读写干扰、垃圾回收干扰、性能隔离等多个角度保证多租户的 SLO 目标。垃圾回收是造成固态盘内部性能波动的主要原因,现有技术通过对黑盒固态盘建模的方式预测垃圾回收的发生,并通过对请求发送速率进行限制的方式来保证设备的 SLO 需求。基于定制设备的全栈 IO 隔离与具体的保证 SLO 策略相结合的方式使多租户场景下固态盘的 IO 执行变得可预测。

表 3 服务质量保证代表性工作

工作	读写干扰	垃圾回收干扰	隔离	实现层次	目标
FIOS FlashFQ	是	否	否	块层	公平性
Split-IO	是	否	否	文件系统,块层,驱动层(OS IO 栈)	公平性,最后期限
SSDCheck TimeSSD	否	是	否	块层	SLO
OPSFWC	否	是	是	OSIO 栈,FTL	隔离
ParaFS AMF	否	是	否	OS IO 栈,FTL	公平性
OIOS	是	否	否	块层	SLO
Reflex	是	否	是	用户态	SLO
DC-Store	是	是	是	全 IO 栈	SLO

5 利用固态盘的计算能力

固态盘的闪存控制器中包含一颗或多颗轻量级处理器,固态盘因而具有一定的计算能力。因此,一些计算任务被卸载在固态盘内执行,这样的固态盘被称作主动固态盘。本节主要介绍将存储系统的部分计算任务交给固态盘内部的处理器执行以提高存储系统的整体性能的工作。

第 1 类工作是计算任务卸载。Active Flash[56]在固态盘控制器中运行高性能计算负载,iSSD[57]通过专用接口将一些数据密集负载如 k-means 放在固

态盘中处理。这些工作包括如何通过块接口传输作业程序到SSD,并提出了SSD的正常操作和计算任务的调度策略,来消除计算任务对正常IO请求处理的影响。

第2类工作是硬件计算能力可编程。Biscuit[58]在SSD控制器中采用硬件进行数据过滤,设计了一个新的基于流的程序设计框架。YourSQL[59]通过专用接口将MySQL的一些查询操作放在固态盘中执行。Willow[60]是一个用户可编程的固态盘,它由可编程闪存芯片构成。每个芯片都有自己的CPU和一个特殊的操作系统,用户可以通过远程过程调用闪存芯片可编程接口,实现功能要求,如块缓存、键值存储加速等功能。A-SSD[61]采用主机端软件控制与设备可编程接口相结合的方式,为不同业务适配不同的设备内FTL以提高系统的整体性能。

另外,利用固态盘内部的CPU进行人工智能训练数据处理加速也是一个新的趋势,如Behemoth[62]和InS-DLA[63]等工作。

总结来说,上述这些工作将固态盘设备的计算能力利用起来以提高存储系统的整体性能。同时,随着固态盘技术的不断发展,标准固态盘协议也在推动新的ZNS接口,将传统的垃圾回收等功能上移至主机端,使设备内的CPU性能更加过剩。但目前标准固态盘缺少暴露设备CPU计算功能的相关接口,使利用设备计算能力变得较为困难。

6 挑战与展望

本文介绍的现有基于闪存固态盘的研究主要集中在提高系统总带宽、减少固态盘内部事务对性能的影响、性能服务质量保证和利用固态盘的计算能力4方面。从中不难发现,为了兼容传统磁盘的块设备接口而设计的系统软件,已经无法充分发挥日益发展的高速闪存固态盘的全部性能。上述系统的发展路线可以归纳总结为基于标准块设备接口的固态盘的黑盒阶段,基于扩展块设备接口与软件控制的灰盒阶段,到软硬件协同设计的白盒阶段。白盒阶段又包括完全开放物理通道的开放通道(open channel)阶段,到目前的更好地划分软硬件功能的ZNS阶段。随着闪存固态盘的发展,以及数据中心对端到端的、差异化的QoS保证需求的提升,目前的工作仍存在如下问题与挑战。

(1)基于标准固态盘的建模预测,需要对发送速率进行严格限制,导致设备性能利用率不高,同时也无法与软硬件协同技术相结合。

(2)软硬件协同设计的方式需要复杂的硬件功能支持,缺乏通用性。通用与兼容是存储系统不得不考虑的问题,标准NVMe协议推动的ZNS接口可以兼容瓦片磁盘(shingled magneting recording, SMR)的区域设备命令协议(zoned-device ATA/block commands, ZAC/ZBC)软件,同时又能进行软硬件协同设计,是目前固态盘接口协议的发展趋势。然而目前缺乏有效的基于ZNS的QoS保证工作,NVMe协议的WRR与输入输出确定性(input/output determination,IOD)功能或成为保证QoS的有效技术。

(3)用户态IO技术也是目前的发展趋势,但用户态系统需要不断完善IO生态环境(如文件系统等存储组件),提高系统的可用性。得益于用户态IO技术,IO所消耗的CPU资源显著降低,采用数据处理器(data processing unit, DPU)与用户态IO及远程直接数据存取(remote direct memory access, RDMA)网络相结合的分布式存储系统后端架构或是不错的选择。

(4)利用固态盘的计算能力需要新的标准与编程框架支持,以提供统一的接口访问。

7 结 论

本文介绍了基于闪存固态盘的存储系统关键性能优化技术方面的研究工作,分析了提高系统总带宽、减少内部事务对性能的影响、性能服务质量保证以及利用固态盘的计算能力4方面的代表性工作,并对相应技术做出了总结。最后总结并提出了对未来基于闪存固态盘的存储系统性能优化的挑战与展望,提出软硬件协同设计是基于闪存固态盘的存储系统性能优化的发展趋势,有助于提供可预测的性能以及可控的端到端延迟。

参考文献

[1] O'NEIL P, CHENG E, GAWLICK D, et al. The log-structured merge-tree (LSM-tree)[J]. Acta Informatica, 1996,33(4):351-385.

[2] STOICA R, PLETKA R, IOANNOU N, et al. Understanding the design trade-offs of hybrid flash controllers [C]//IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. Rennes: IEEE, 2019:1-8.

[3] 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.

[4] MAO B, WU S, DUAN L. Improving the SSD performance by exploiting request characteristics and internal parallelism[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018,37(2): 472-484.

[5] NVMe INC. NVMexpress [EB/OL]. (2015-10-13) [2021-11-21]. https://www.nvmexpress.org.

[6] Intel, Intel SSD [EB/OL]. (2016-11-1)[2021-11-21]. https://www.intel.cn/content/www/cn/zh/products/details/memory-storage/data-center-ssds.html.

[7] Intel. SPDK[EB/OL]. (2015-12-8)[2021-11-21]. https://www.spdk.io.

[8] SA W, SA B, EL M, et al. Ceph: a scalable, high-performance distributed file system[C]//The 7th Symposium on Operating Systems design and Implementation. Berkeley: USENIX Association, 2006: 307-320.

[9] YOSHIMURA T, CHIBA T, HORII H. EvFS: user-level, event-driven file system for non-volatile memory[C]//The 11th USENIX Workshop on Hot Topics in Storage and File Systems. Renton:USENIX Association, 2019.

[10] LIU J, ARPACI-DUSSEAU A C, ARPACI-DUSSEAU R H,et al. File systems as processes[C]//The 11th USENIX Workshop on Hot Topics in Storage and File Systems. Renton: USENIX Association, 2019.

[11] LIU J, REBELLO A, DAIY, et al. Scale and performance in a file system semi-microkernel[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. Berkeley: USENIX Association, 2021: 819-835.

[12] SHIN W, CHEN Q, OH M, et al. OS I/O path optimizations for flash solid-state drives[C]//USENIX Annual Technical Conference. Philadelphia: USENIX Association, 2014, 483-488.

[13] LEE G, SHIN S, SONGW, et al. Asynchronous I/O stack: a low-latency kernel I/O stack for ultra-low latency SSDs[C]//USENIX Annual Technical Conference. Renton:USENIX Association, 2019:603-616.

[14] Linux Kernel Image. Io_uring[EB/OL]. (2019-10-22) [2021-11-21]. https://kernel.dk/io_uring-whatsnew.pdf.

[15] ZHANG J, KWON M, GOUK D, et al. Flashshare: Punching through server storage stack from kernel to firmware for ultra-low latency ssds.[C]//The 13th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad: USENIX Association, 2018,477-492.

[16] BJØRLING M, GONZALEZ J, BONNET P. LightNVM: The linux open-channel SSD subsystem[C]//The 15th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2017, 359-374.

[17] BJØRLING M. From open-channel SSDs to zoned namespaces[C/OL]. (2019-02-26)[2021-11-24]. https://www.usenix.org/conference/vault19/presentation/bjorling.

[18] LEE S, LIU M, JUN S W, et al. Application-managed flash[C]//The 14th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2016:339-353.

[19] TANG L, HUANG Q, LLOYD W, et al. RIPQ: advanced photo caching on flash for Facebook [C]//The 13th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2015:373-386.

[20] OUYANG J, LIN S, SONG J, et al. SDF: software-defined flash for web-scale Internet storage systems[C]//International Conference on Architectural Support for Programming Languages and Operating Systems. New York: Association for Computing Machinery, 2014:471-484.

[21] WANG P, SUN G, JIANG S, et al. An efficient design and implementation of LSM-tree based key-value store on open-channel SSD[C]//Proceedings of the 9th European Conference on Computer Systems. Berlin, Heidelberg: Springer-Verlag, 2014: 16-30.

[22] ZHANG J, SHU J, LU Y. ParaFS: a log-structured file system to exploit the internal parallelism of flash devices [C]//USENIX Annual Technical Conference. Denver: USENIX Association, 2016: 87-100.

[23] BJØRLING M, AXBOE J, NELLANS D, et al. Linux block IO: introducing multi-queue SSD access on multi-core systems[C]//Proceedings of the 6th International Systems and Storage Conference. New York: Association for Computing Machinery, 2013:22-31.

[24] KANG J, ZHANG B, WO T, et al. SpanFS: a scalable file system on fast storage devices[C]//USENIX Annual Technical Conference. Santa Clara: USENIX Association, 2015:249-261.

[25] ZHANG Y, ARULRAJ L P, ARPACI-DUSSEAU A C, et al. De-indirection for flash-based SSDs with nameless writes [C]//The 10th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2012: 1-15.

[26] SAXENA M, SWIFT M M, ZHANG Y. FlashTier: a lightweight, consistent and durable storage cache[C]//Proceedings of the 7th ACM European Conference On Computer Systems. New York: Association for Computing

Machinery, 2012: 267-280.
[27] GUPTA A, PISOLKAR R, URGAONKAR B, et al. Leveraging value locality in optimizing NAND flash-based SSDs[C]// The 9th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2011: 91-103.
[28] WU G, HE X. Delta-FTL: improving SSD lifetime via exploiting content locality[C]// Proceedings of the 7th ACM European Conference on Computer Systems. New York: Association for Computing Machinery, 2012:253-266.
[29] LEE C, SIM D, HWANGJ Y, et al. F2FS: a new file system for flash storage[C]// The 13th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2015:273-286.
[30] LI C, SHILANE P, DOUGLIS F, et al. Pannier: a container-based flash cache for compound objects[C]// Proceedings of the 16th Annual Middleware Conference. New York: Association for Computing Machinery, 2015:50-62.
[31] LU Y, SHU J, ZHENG W. Extending the lifetime of flash-based storage through reducing write amplification from file systems[C]// The 11th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2013: 257-270.
[32] KANG J U, HYUN J, MAENGH, et al. The multi-streamed solid-state drive[C]// The 6th USENIX Workshop on Hot Topics in Storage and File Systems. Philadelphia: USENIX Association, 2014.
[33] RHO E, JOSH K, SHIN S U, et al. FStream: managing flash streams in the file system[C]// The 16th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2018: 257-264.
[34] KIM T, HONG D, HAHN S S, et al. Fully automatic stream management for multi-streamed ssds using program contexts[C]// The 17th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2019: 295-308.
[35] HAN K, GWAK H, SHIN D, et al. ZNS+: advanced zoned namespace interface for supporting in-storage zone compaction[C]// The 15th USENIX Symposium on Operating Systems Design and Implementation. Santa Clara: USENIX Association, 2021:147-162.
[36] CHANG L P, KUO T W. An adaptive striping architecture for flash memory storage systems of embedded systems[C]// Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium. Washington: IEEE, 2002:821-826.
[37] MURUGAN M, DU D H. Rejuvenator: a static wear leveling algorithm for NAND flash memory with minimized overhead[C]// 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies. Denver: IEEE, 2011:1-12.
[38] KIM B S, YANG H S, MIN S L. Auto SSD: an autonomic SSD architecture[C]// USENIX Annual Technical Conference. Boston: USENIX Association, 2018:677-690.
[39] 陈震, 刘文洁, 张晓,等. 基于磁盘和固态硬盘的混合存储系统研究综述[J]. 计算机应用, 2017, 37(5): 1217-1222.
[40] 杨松芳. 延长固态硬盘使用寿命的算法综述[J]. 中国集成电路, 2018(6):48-51.
[41] PARK S, SHEN K. FIOS: a fair, efficient flash I/O scheduler[C]// The 10th USENIX Conference on File and Storage Technologies. Santa Clara:USENIX Association, 2012:13.
[42] SHEN K, PARK S. FlashFQ: a fair queueing I/O scheduler for flash-based SSDs[C]// USENIX Annual Technical Conference. San Jose: USENIX Association, 2013: 67-78.
[43] 孟祥辉,曾学文,陈晓,等. 面向闪存存储的公平高效I/O调度器[J]. 网络新媒体技术, 2018,7(4):6.
[44] YANG S, HARTER T, AGRAWAL N, et al. Split-level I/O scheduling[C]// Proceedings of the 25th Symposium on Operating Systems Principles. New York: Association for Computing Machinery,2015: 474-489.
[45] WOO J, AHN M, LEE G, et al . D2FQ: device-direct fair queueing for NVMe SSDs[C/OL]. (2021-2-23)[2021-11-24]. https://www. usenix. org/system/files/fast21-woo. pdf.
[46] HEDAYATI M, SHEN K, SCOT M L, et al. Multi-queue fair queuing[C]// USENIX Annual Technical Conference. Renton: USENIX Association, 2019:301-314.
[47] JUNG M, CHOI W, SRIKANTAIAHS, et al. HIOS: a host interface I/O scheduler for solid state disks[C]// ACM SIGARCH Computer Architecture News. New York:Association for Computing Machinery, 2014: 289-300.
[48] KIM J, LEE D, NOH S H. Towards SLO complying SSDs through OPS isolation[C]// The 13th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association,2015:183-189.
[49] CHOI W, URGAONKAR B, KANDEMIR M, et al. Fair write attribution and allocation for consolidated flash cache[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York:Association for Computing Machinery,2020: 1063-1076.
[50] KWON M, GOUK D, LEE C, et al. DC-store: eliminating noisy neighbor containers using deterministic I/O performance and resource isolation[C]// The 18th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2020: 183-191.
[51] KIM J, LEE E, NOH S H. I/O scheduling schemes for better I/O proportionality on flash-based SSDs[C]// Modeling, Analysis and Simulation of Computer and Tele-

communication Systems. Santa Clara: USENIX Association, 2016:221-230.
[52] CHANG D W, CHEN H H, SU W J. VSSD: performance isolation in a solid-state drive. [J] ACM Transactions on Design Automation of Electronic Systems, 2015, 20(4):51-84.
[53] KLIMOVIC A, LITZ H, KOZYRAKIS C. ReFlex: remote flash ≈ local flash[J]. ACM SIGPLAN Notices, 2017,52(4):345-359.
[54] KIM J, PARK P, AHN J, et al. SSDcheck: timely and accurate prediction of irregular behaviors in black-box SSDs[C]//The 51st Annual IEEE/ACM International Symposium on Microarchitecture. Fukuoka: IEEE, 2018: 455-468.
[55] WANG X, YUAN Y, ZHOU Y, et al. Project almanac: a time-traveling solid-state drive[C]//Proceedings of the 14th EuroSys Conference. New York: Association for Computing Machinery,2019:1-16.
[56] BOBOILA S, KIM Y, VAZHKUDAI S S, et al. Active flash: out-of-core data analytics on flash storage[C]// The 28th Symposium on Mass Storage Systems and Technologies. San Diego: IEEE, 2012: 1-12.
[57] CHO S, PARK C, OH H, et al. Active disk meets flash: a case for intelligent SSDS[C]//Proceedings of the 27th International ACM Conference On International Conference On Supercomputing. New York: Association for Computing Machinery,2013: 91-102.
[58] GU B, YOON A S, BAED H, et al. Biscuit: a framework for near-data processing of big data workloads[C]// ACM/IEEE 43rd Annual International Symposium on Computer Architecture. New York: Association for Computing Machinery,2016:153-165.
[59] JO I, BAE D H, YOON A S, et al. Your SQL: a high-performance database system leveraging in-storage computing[C]//Proceedings of the VLDB Endowment. New York: Association for Computing Machinery, 2016: 924-935.
[60] SESHADRI S, GAHAGAN M, BHASKARAN M S, et al. Willow: a user-programmable SSD[C]//The 11th USENIX Symposium on Operating Systems Design and Implementation. Broomfield: USENIX Association, 2014, 67-80.
[61] 张建权. 基于软件定义的闪存存储系统构建及性能优化技术研究[D]. 武汉:华中科技大学,2018.
[62] KIM S, JIN Y, SOHN G, et al. Behemoth: a flash-centric training accelerator for extreme-scale DNNs[C]// The 19th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2021: 371-385.
[63] LIANG S W, WANG Y, LIU C, et al. InS-DLA: an In-SSD deep learning accelerator for near-data processing [C]//The 29th International Conference on Field Programmable Logic and Applications (FPL). Barcelona: IEEE, 2019:173-179.

A survey of techniques for improving performance of flash SSD based storage systems

LI Tianxiang[*,**], JIANG Dejun[*,**,***], XIONG Jin[*,**]
([*] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
([**] University of Chinese Academy of Sciences, Beijing 100049)
([***] Pengcheng Laboratory, Shenzhen 518055)

Abstract

Flash-based solid state drives (SSDs) are characterized by high parallelism, complex internal background activities and computing capability, while existing storage systems are designed for hard disk and cannot take full advantage of the performance of flash-based SSDs. This paper introduces the problems and challenges of flash SSD based storage systems, and the related techniques are described from four aspects: improving the total bandwidth of the system, reducing the influence of internal activities on performance, guarantee the quality of service of performance, and utilizing the computing capability of SSD. Finally, the unresolved technique issues of current flash SSD based storage systems are discussed. This paper concludes that the software/hardware co-design approach is a trend for SSD-based storage system, and this approach helps to provide more predictable performance and controllable end-to-end latency.

Key words: flash solid state drive (SSD), input/output (IO) performance, garbage collection, quality of service, computing task offload