计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180772
56(1):23-34,2019

闪存存储的重构与系统构建技术

陆游游　杨　者　舒继武
(清华大学计算机科学与技术系　北京　100084)
(luyouyou@tsinghua.edu.cn)

Revisiting the Architecture and System of Flash-Based Storage

Lu Youyou, Yang Zhe, and Shu Jiwu
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract　Flash-based storage has been rapidly and widely used in different fields, from embedded systems, desktops, enterprise servers to data centers, in recent years. How to explore the potentials of flash memory is an important research direction in storage research area. Legacy storage systems have been designed for hard disks for more than 60 years, and optimizations to existing systems have limited effectiveness in exploring flash storage benefits. New flash storage architecture and systems by re-architecting flash storage show potentials, and are being adopted in industry. This paper presents the research process in this area. First, it introduces the characteristics of flash memory and solid state drive, and analyses the problems of legacy flash storage architecture. Then, it describes the architecture evolution of flash storage, including device-based FTL, host-based FTL and software managed flash. After the description, it surveys the storage systems respectively on open-channel SSDs and near-data processing, both of which are required for function relocation and cooperation in software-managed flash. Finally, it concludes the challenges and remained research problems.

Key words　open-channel SSD; software defined flash; near data processing; software and hardware co-design; open programmable SSD

摘　要　近些年来,闪存存储在嵌入式、桌面机、服务器及数据中心领域中均得到了快速而广泛的应用.如何高效发挥闪存效率是近来存储系统研究中的一个重要的问题.传统存储系统以磁盘为基础的设计维系了60余年,基于现有存储系统的优化对闪存优势的发挥效果甚微.因而,重构闪存存储成为近年来的研究热点.对该研究方向的研究现状与进展进行介绍.首先,介绍了闪存与固态盘的特性,并分析了当前闪存存储结构的问题;然后,介绍了闪存存储架构从设备内FTL、主机端FTL、软件直管闪存以及开放可编程闪存的演进;接着,从软件直管闪存的软件定义和硬件卸载2个方面,分别介绍基于开放通道SSD的存储系统和基于近数据处理的闪存存储系统;最后总结了闪存存储重构和系统构建技术的挑战与下一步研究的问题.

关键词　开放通道SSD;软件定义闪存;近数据处理;软硬件协同设计;开放可编程闪存

中图法分类号　TP391

固态盘(solid state drive, SSD)是一种完全电子式访问的存储设备.在计算机发展的历史中,磁盘自1956年起主导外存储领域约70年,然而并非完全电子式设备.在磁盘中,数据访问操作仍然需要磁头

收稿日期:2018-11-17;**修回日期**:2018-11-29
基金项目:国家自然科学基金项目(61772300,61327902)
This work was supported by the National Natural Science Foundation of China (61772300, 61327902).
通信作者:舒继武(shujw@tsinghua.edu.cn)

驱动、盘片旋转等机械式部件的操作.这些机械式部件操作限制了外存储性能的提升,也限制了计算机成为完整意义上的电子式计算机.固态盘提供了完全电子式的数据访问,使得电子计算机在计算、存储与通信三大方面均实现了电子式数据处理,因而固态盘的发展与应用在计算机发展历史中具有里程碑的意义.

当前,固态盘多采用闪存(flash memory)存储介质,其中以NAND闪存为主.NAND闪存与传统磁盘介质的特性差异较大,例如写前擦除、读写粒度与擦除粒度不匹配、读写不对称、磨损寿命有限等特性.为了兼容传统磁盘的读写接口,固态盘引入了闪存转换层(flash translation layer, FTL)将传统的读写接口转换为闪存的读写擦操作.闪存转换层的引入兼容了传统磁盘接口,但是对上层系统软件屏蔽了闪存的特性,阻碍了上层系统软件对于闪存优势的发挥,甚至恶化了闪存的寿命等缺陷问题[1].针对该问题,清华大学从2010年开始了"重构闪存存储系统"的研究,以探索闪存存储管理在软硬件上的合理划分以及跨层协作,以软硬件协同设计的方式发挥闪存存储的优势和潜力[2-3].近些年来,在闪存存储的重构与系统构建方面,学术界和工业界都取得了不少进展.本文将从传统闪存存储架构的问题、闪存存储架构的演进、开放通道SSD存储系统和闪存设备内计算等方面介绍相关进展,最后总结并展望闪存存储的重构与系统构建方面的机遇与挑战.

1 传统闪存存储架构的问题

1.1 闪存与固态盘

在大容量固态盘中,NAND闪存是主要的存储介质.NAND闪存(下文若无特殊说明,闪存是指NAND闪存)的单元以电子的方式记录数据的0/1状态.依据状态表示比特数不同,闪存分为SLC,MLC,TLC及QLC,分别表示每个闪存单元记录1,2,3和4个位数.闪存相比于传统磁盘介质具有诸多独特的特性,包括3个主要特性:

1) 写前擦除.NAND闪存单元的编程呈现单向特点,即可以从状态1编程为0,反之不行.对于闪存页的写入操作,闪存需要将数据写入已擦除的页面中.这被成为写前擦除,也被称为不可覆盖写.

2) 读写擦粒度不同.闪存以闪存页为读写粒度,例如4 KB,8 KB或16 KB等;以闪存块为擦除粒度,包含若干个页面,例如64个页面.不同的粒度也造成延迟的差异,读写操作延迟在百微妙级别,而擦除操作在毫秒级别.

3) 磨损寿命有限(耐久性).闪存单元能承受的擦写操作次数有限.经过一定数量的擦写操作后,闪存单元不能可靠地存储数据状态.这一过程被称为磨损,所经历的擦写次数用于衡量闪存的擦写寿命,也被称为耐久性(endurance).

在固态盘中,闪存颗粒通过不同的通道连接于闪存控制器.闪存控制器对闪存硬件单元进行读写擦操作.固态盘向主机端提供传统的读写接口.在主机端接口与闪存控制器之间,固态盘引入闪存转换层(FTL)将传统读写操作转换为闪存内部的读写擦操作.闪存转换层的主要功能包括3个方面:

1) 地址映射.闪存具有毫秒级的擦除延迟,为避免写前操作限制中擦除对写操作的延迟影响,FTL采用了异地更新的写入机制.异地更新将数据写入空闲物理闪存页,标记旧数据页面为过期页面,这引入了地址映射表用于记录数据的物理闪存页的地址.FTL中采用不同的映射机制,包括页级映射、块级映射或混合映射.

2) 垃圾回收.异地更新机制将旧数据标记为无效,FTL需要垃圾回收功能以回收这些无效闪存页,并擦除后将空间用于新数据的分配.由于闪存以闪存块为单位擦除,垃圾回收选择待回收的闪存块,将其中有效页面拷贝到新地址,然后进行擦除.垃圾回收采用不同的算法,以减少垃圾回收的开销,并考虑闪存的磨损.

3) 磨损均衡.磨损均衡尽可能将闪存的擦写操作均匀分布于不同的闪存单元,以避免局部擦写磨损严重引发寿命问题.

闪存转换层屏蔽了很多闪存介质的特性,但从固态盘层次上仍具有一些与磁盘不同的表现.由于固态盘内部存在通道级、芯片级、Plane级等多个级别的并发,固态盘内部可以提供极高的,高于当前SATA和PCIe接口的带宽.因而,固态盘在读操作的带宽可接近于硬件接口带宽,写带宽由于闪存写延迟稍高略低于读带宽.在随机读写方面,固态盘提供了较好的随机读性能,但随机写性能较差.整体而言,由于介质的差异,固态盘与磁盘存在较大的差异.

1.2 传统固态盘存储系统的问题

在存储系统中,磁盘自1956年发明以来,长期居于外存的主导地位.当前存储系统多基于磁盘特性的假设进行设计,极少考虑其他存储介质的特性.随着闪存技术的广泛应用,如何高效利用闪存并构建适合于闪存的存储系统值得关注与深入思考.

固态盘兼容了传统磁盘的读写接口,传统系统软件可直接运行于固态盘之上.尽管固态盘提供了比磁盘更高的性能,然而闪存的优势并没有得到充分的发挥.这其中的问题主要体现在4个方面[1-5]:

1)功能冗余.存储系统软件与闪存转换层存在类似的功能,如地址映射等.这些冗余的功能既带来了不必要的开销,也增加了不同层次间操作冲突的概率.

2)维度缺失.存储软件的设计仅关注性能、可靠性等传统设计维度,而未考虑耐久性等闪存的新维度.维度的缺失造成固态盘效率低,甚至出现磨损严重等问题.

3)分工欠妥.闪存具有的新特性改变了不同功能在软硬件不同层次中的效率.例如,闪存的异地更新为存储一致性提供了高效实现的可能,因而可降低传统软件做法的开销.

4)优化错配.系统软件针对磁盘的一些优化不仅不能优化固态盘效率,反而增加固态盘的负担.例如,固态盘的读写不对称问题,即读性能优于写性能,存储系统中针对随机读的优化对于固态盘则效果甚微.

针对这些问题,固态盘试图扩展读写接口,例如引入Trim命令.Trim命令将软件中无效数据及时通知固态盘,以避免固态盘中无效操作.然而,接口的扩展对于提高固态盘效率杯水车薪.

固态盘内部的闪存转换层在不同厂商之间以及不同型号之间的算法实现差异较大,系统软件难以感知内部特性,因而难以发挥闪存的效果.在耐久性与性能方面,传统的固态盘存储结构均存在极大的挑战.

首先,在耐久性方面,闪存的寿命与密度存在矛盾.闪存要求更高的密度,单元密度要求越来越高,从SLC,MLC,TLC至QLC,制程工艺也要求越来越精细.然而这2方面的提升均导致磨损寿命越来越低,如图1所示.如果系统软件不能有效控制数据的写入量与磨损均衡,那么闪存的磨损加剧,影响寿命,进而影响数据的可靠性.

其次,在性能方面,存储系统要求闪存不但要提供更高的性能,同时要提供更稳定的性能.然而,在当前黑盒的固态盘结构上,固态盘中的垃圾回收通常会影响前端I/O.同时,内部的读写混合也造成数据延迟的抖动.因而,在软件系统上,性能抖动也是比较严重的问题,如图2所示.

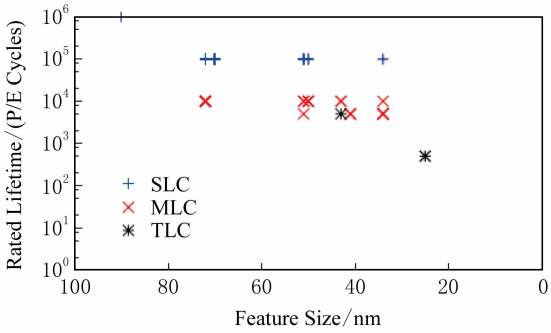
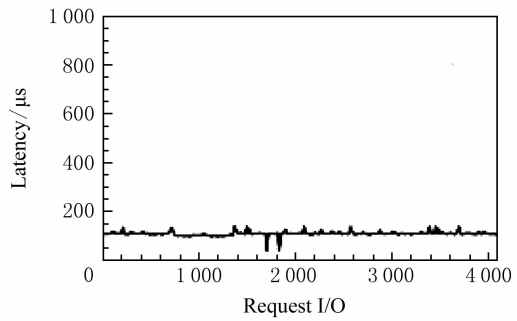
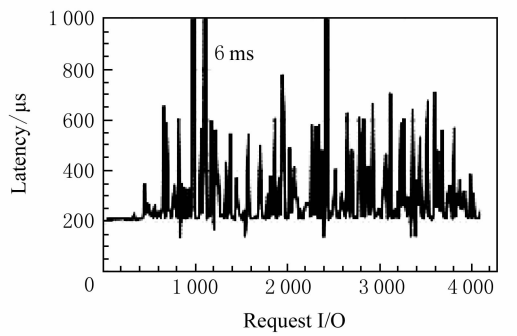


Fig. 1 Conflict between flash lifetime and density[6]

图1 闪存寿命与密度的矛盾[6]



(a) 100% Random Read



(b) 50% Random Read and 50% Random Write

Fig. 2 Interface of read and write[7]

图2 读写干扰[7]

最后,当前的固态盘存储系统仅仅将固态盘当做快速的磁盘,而丧失了闪存固态盘的很多机会,例如如何实现基于内部并发实现灵活的数据布局、如何基于读写擦特性实现更高的空间利用、如何结合盘内与系统级容错机制实现更可靠的数据保护等.

2 闪存存储架构的演进

闪存技术于20世纪80年代由日本东芝提出,其后20余年由于技术不成熟、闪存容量有限,多用于嵌入式系统.直至本世纪初,闪存技术得以发展,闪存才开始走进桌面机、服务器及数据中心等领域.

在桌面机、服务器及数据中心这些通用的计算机系统中,闪存存储架构出现了 4 种主要的形态:

2.1 设备内 FTL 闪存架构

设备内 FTL 闪存架构(device-based SSD)在设备内部通过闪存转换层向主机端提供传统读写接口,如图 3(a)所示. 这种架构是传统的闪存存储架构,在 20 世纪 90 年代便已出现,至今仍然是主流的闪存存储架构.

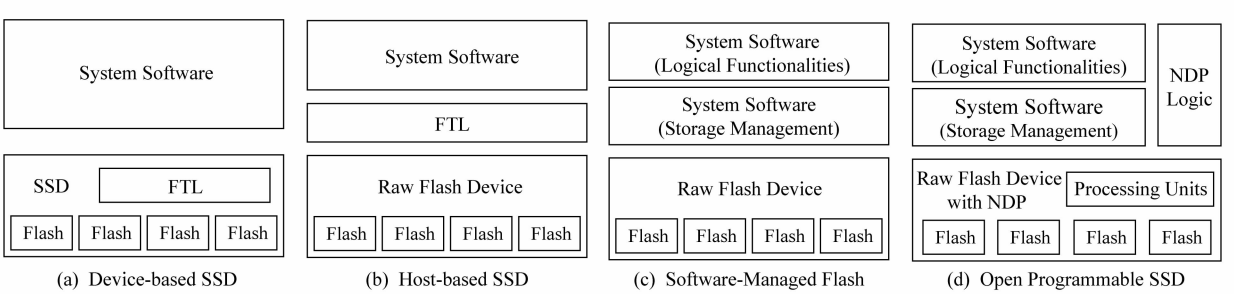


Fig. 3 Evolution of flash storage architectures (modified based on [2])

图 3 闪存存储架构的演进(基于文献[2]修改)

2.2 主机端 FTL 闪存架构

主机端 FTL 闪存架构(host-based SSD)将闪存转换层从设备内部上移到主机端进行实现,如图 3(b)所示. 这种架构约在 2008 年由美国 FusionIO 公司(FusionIO 公司后被 SanDisk 收购,再随 SanDisk 被西数公司收购)提出. 由于固态盘内部连接的闪存芯片数量越来越多,且通过不同级别提供并发,闪存转换层的地址映射表越来越大,且处理速度需求越来越高. 以页级映射为例,1 TB 闪存约需要 2 GB 映射表. 由于设备内嵌入式处理器与内存容量受限,闪存转换层的处理性能难以提升,进而影响固态盘性能. 主机端 FTL 闪存架构在主机端实现闪存转换层,可以共享主机端的 CPU 与内存能力,大幅提升了闪存转换层的处理能力,进而提升了固态盘的性能. 然而,主机端 FTL 闪存架构的缺点在于,其占用了主机端 CPU 与内存资源,导致与前台应用的资源竞争.

2.3 软件直管闪存架构

软件直管闪存架构(software-managed flash)无需闪存转换层,由系统软件直接管理闪存,对闪存物理地址进行读、写、擦操作,如图 3(c)所示. 这种架构于 2010 年由清华大学提出,在 2013 年 FAST 会议上公开[2]. 软件直管闪存架构的提出源于系统软件与闪存转换层之间的冗余与冲突(详见 1.2 节),核心在于弥合系统软件与闪存硬件之间的语义鸿沟,优化功能在软硬件层次的划分与协作,实现软硬件协同的闪存存储设计.

软件直管闪存架构移除了冗余的层次,促进了面向闪存的软件系统新设计,引入了软件管理的灵活性. 为探索验证该架构在存储系统上的优势,清华大学于 2012 年和 2013 年分别定制了 SATA 接口与 PCIe 接口的裸闪存设备,如图 4 所示. 裸闪存设备向软件系统直接导出物理地址及读写擦接口,软件系统可直接对指定闪存地址进行读写擦操作. 闪存地址包含了内部布局信息,可指定不同并发单元上的闪存页地址. 同时,裸闪存设备还导出了闪存页元数据(也称 OOB 区域)的操作. 这种裸闪存设备后来也被称作开放通道 SSD(open-channel SSD)[8],软件直管闪存架构也被成为软件定义闪存(software-defined flash)[9] 或应用管理的闪存(application-managed flash)[10]等.

Fig. 4 Raw flash devices with the SATA and the PCIe interface customized early by Tsinghua University

图 4 清华大学早期定制的 SATA 接口及 PCIe 接口裸闪存设备

主机端 FTL 闪存架构和软件直管闪存架构 2 种架构中的闪存设备类似,因而广义的开放通道 SSD 架构包含这 2 种架构.基于同样的硬件,清华大学既在主机端实现了 Jasmine PFTL[11],也设计了软件直管架构的文件系统[2,12]、键值存储[13]和分布式对象存储[14].2017 年 FAST 会议上,丹麦根本哈根 IT 大学和 CNEX Labs 在 Linux 内核 4.4 版本中提供了开放通道 SSD 架构的 LightNVM 驱动[15].LightNVM 的 pblk 也提供了主机端 FTL 的实现.

2.4 开放可编程闪存架构

开放可编程闪存架构(open programmable SSD)是清华大学在闪存架构上正在开展的研究内容,如图 3(d)所示.开放可编程闪存架构的目标是寻求功能软硬件在两者之间的合理分工与协作.这既包括将闪存特性导出给软件,由软件实现灵活管理,又包括利用 SSD 内部资源实现近数据的处理,即由软件定义与硬件卸载两者有机组成.

开放可编程闪存架构相比于软件直管闪存架构需要更多的具有处理功能的硬件卸载设计.在功能的抽象与选取以及功能重新排布后的协调均需要进一步的研究.现有的闪存存储系统主要还是在软件定义和硬件卸载 2 个方面分别研究,尚未出现两者的有机结合.第 3,4 节将分别从基于开放通道 SSD 的存储系统和基于近数据处理的闪存存储系统 2 个方面分别介绍.

3 基于开放通道 SSD 的存储系统

开放通道 SSD 向主机导出了物理地址及读写擦接口.开放通道 SSD 为主机端的存储系统提供了更多的设计空间.当前开放通道 SSD 的存储系统主要包括:文件系统、键值存储系统和分布式存储系统.

3.1 文件系统

文件系统是存储系统中的基础软件,如何适配闪存是当前文件系统研究中的重要的研究问题.当前的闪存文件系统既有基于现有闪存架构的数据组织优化的文件系统 F2FS[16],也有扩展接口对文件系统组织方式与元数据管理重新设计的 DFS[17]和 ReconFS[18],还有基于开放通道 SSD 设计的新型文件系统.

清华大学于 2013 年提出了软件直管闪存的架构,设计并实现了对象式闪存存储系统(object flash storage system, OFSS)[2].OFSS 移除了设备内的 FTL 的实现,将原有 FTL 的功能集成于文件系统的存储管理部分,形成 Object-FTL.Object-FTL 直接管理闪存介质,并根据对象语义与闪存特性重新设计了存储机制,包括利用闪存页的 OOB 记录反向索引等额外信息以延缓索引与日志的刷写,利用闪存块/页状态转换的特性设计了空闲空间管理机制,并对不对齐的写操作采用了拼接紧凑写的机制.OFSS 的设计大幅降低了存储系统自身机制所引入的额外元数据的写入,相比于传统文件系统,显著降低了文件系统写放大,提升了系统性能,延长了闪存寿命.

麻省理工学院(Massachusetts Institute of Technology, MIT)于 2016 年在基于开放通道 SSD 的架构 AMF[10]上提出了 ALFS 文件系统.ALFS 采用日志式文件系统(log-structured file system, LFS)的基本架构,所有数据采用异地更新.基于此,ALFS 能够直接管理闪存设备的空间分配和垃圾回收,显著地减少了闪存设备的内存占用,提高文件系统的性能.

清华大学的 ParaFS[12]是以发挥 SSD 内部并发为出发点而设计的基于开放通道 SSD 的文件系统.ParaFS 采用日志式结构,将文件系统中的数据分段与闪存物理块对应,直接管理闪存设备.ParaFS 在空间分配上提出二维分配机制,综合考虑设备的并发特性和数据冷热程度,发挥闪存设备并发特性,同时保证冷热程度不同的数据相互隔离.ParaFS 利用文件系统的语义信息直接对闪存块进行垃圾回收.回收粒度与 ALFS 的"超级块"相比更小,提高了垃圾回收的效率.在 I/O 调度方面,ParaFS 在文件系统层为每个闪存通道的 I/O 请求进行优化调度.ParaFS 在重负载下能显著提升系统的整体性能,并且较好地控制了系统的性能抖动问题.

3.2 键值存储

键值存储系统在当前互联网系统中广泛使用.基于 LSM-Tree 的键值存储的数据组织方式与 SSD 内 FTL 的数据组织方式有较多相似之处.两者之间的功能冗余问题,为基于开放通道 SSD 架构直接构建键值存储系统提供机会.

北京大学和百度公司于 2014 年在开放通道 SSD 架构 SDF[9]上设计并实现了基于 LSM-Tree 的键值存储系统 LOCS[8].在 LSM-Tree 日志式更新过程中,键值存储以表的大小为粒度写入 SSD.表的大小能够设置为与闪存块大小相同或整数倍,因而无需 SSD 内部再进行垃圾回收操作.同时,由于开放通道 SSD 导出了物理地址,键值存储系统可以感知

每个通道上请求的状态,从而依据读写擦的延迟进行请求的调度,达到通道间的负载均衡.在重负载下,每个闪存块上均有大量的键值表的写入,无需页级别的并发即可发挥SSD的性能.但是采用这种方式,对同一个键值文件的请求只能发送到闪存块所在的通道处理,对于轻负载下的单个请求延迟会有所影响.

香港理工大学于2017年提出的DIDACache[19],是基于开放通道SSD架构的键值缓存系统.DIDACache在软件层直接管理闪存设备,并根据当前的缓存状态选择最合适的替换策略与垃圾回收算法,提高键值缓存系统的性能.

清华大学2017年也提出基于开放通道SSD架构的键值存储系统FlashKV[13].FlashKV在应用层直接管理闪存介质,消除LSM-Tree、文件系统和FTL的功能冗余;通过基于"超级块"的空间管理机制,降低文件索引开销,提高设备空间利用率.为了保证系统的灾后一致性,FlashKV设计了可重建的静态数据布局机制.同时,FlashKV根据负载特征,动态调整LSM-Tree的压紧(compaction)机制发起的写请求,以降低对前台请求的干扰.实验表明:与基于SSD的LevelDB相比,FlashKV的重负载性能提升了3.5倍,数据写入量减少50%.

3.3 分布式存储

对象存储在分布式存储系统中使用广泛.清华大学提出的OFSS系统在接入分布式存储系统面临着分布式存储特有的性质,例如存在分布式存储中以事务方式提交对象的读写、不同进程所提交的事务之间无依赖关系等特点.清华大学提出基于开放通道SSD构建分布式存储系统的对象存储机制OCStore[14].OCStore绕过文件系统和FTL,直接管理闪存设备,利用闪存设备的内部并发特性.同时,采用面向对象的闪存事务更新机制,为对象的数据与元数据都提供了原子性更新,显著地降低了更新开销.由于分布式系统中,存在一些相互独立的事务流,OCStore针对此设计了分组事务提交机制.OCStore根据分布式系统提交的事务语义和闪存设备的通道状态,建立事务感知的请求调度策略,将属于同一个事务的I/O请求在同一时间段进行处理,以降低事务的平均响应延迟.

此外,分布式系统采用多副本或纠删码的容错机制,闪存设备存在ECC和RAID等提高数据的可靠性,这2层存在冗余和隔离.针对此问题,OCStore提出在软件层综合管理利用分布式存储系统和闪存设备的可靠性机制,例如闪存设备读取出错恢复的同时,OCStore从其他副本读数据,从更快的一方获取正确数据.实验显示,OCStore性能提升1.5~3.0倍,延迟更低,重负载数据写入量降低的比例达到70%.

3.4 小 结

表1比较了上述的基于开放通道SSD的文件系统、键值存储和分布式存储系统.上述系统结合闪存设备的物理特性,软件与硬件协同设计,从软件层直接管理闪存设备.这种方式消除了基于传统SSD的存储系统各层之间的功能冗余,打破语义隔离,从而降低I/O延迟,减小数据写放大.

Table 1 Comparison of Storage Systems Based on Open-Channel SSD

表1 基于开放通道SSD的存储系统比较

Category	File System			Key-Value			Distributed System
	OFSS	ALFS	ParaFS	LOCS	DIDACache	FlashKV	OCStore
Feature	Object-based	Log-structured	Log-structured	External Storage	Cache	External Storage	Object Store for Distributed System
Allocation Scheme	Page	Super Block	2D[12]	Block-level	Block-level	Super Block (page-level)	2D
Parallelism Level	Medium	Medium	High	Low	Low	High	High
GC Overhead	Low	Medium	Low	Low	Low	Low	Low
Optimized I/O Scheduling	No	No	Yes	Yes	No	Yes	Yes

4 基于近数据处理的闪存存储系统

存储设备的控制器具有计算能力,内部的传输带宽通常大于主机与设备之间的传输带宽.近数据处理机制将主机的一部分计算卸载到数据所在的设备上,能够减少主机与设备之间的数据传输量,充分利用设备内的带宽资源.Active disks[20],Active Storage[21]等将近数据处理引入磁盘存储系统,加速数据库查询、外排序和图像卷积等应用.与磁盘相比,

固态盘的闪存介质具有并发特性,内部传输带宽更高.控制器由于要完成闪存控制、主机交互和 FTL,计算能力更强.闪存介质的物理特性与磁盘也有差异.近年来,一些研究工作提出利用闪存设备中的计算资源,构建基于近数据处理的闪存存储系统,提供读写以外的更丰富的接口.

4.1 事务闪存

在 SSD 提供事务处理功能并向上导出事务接口,是 SSD 在近数据处理中早期应用的例子.闪存的异地更新特性,使得闪存设备天然地支持数据的多版本.利用这一特性提供事务恢复机制,可以免除软件显式维护数据版本的代价,也可以提供一致性支持[22-26].

微软研究院于 2008 年提出事务闪存 TxFlash[22],在 SSD 控制器中实现事务机制,数据和元数据存储在闪存介质上.TxFlash 设计了 SCC 和 BPCC 两个事务提交协议,均采用链表环方式,将同一事务的所有页面链接成环.故障恢复时,如果能够完整地读取该结构,事务为提交状态,否则为未提交状态.但是,由于闪存的垃圾回收破坏了环状结构,TxFlash 维持事务状态的开销极高.

针对事务状态开销的问题,清华大学的 LightTx[23] 引入了"更新窗口"事务状态跟踪机制,仅跟踪最近未确定状态的事务,同时采用计数式提交协议,避免了环状结构的维护开销.LightTx 以较低的开销在 SSD 内部提供了事务恢复机制.

进一步地,清华大学提出了 DiffTx[25],结合应用的整页写与部分页写的特征,将闪存异地更新特性与传统写前日志的方式相结合.DiffTx 对于整页写采用了闪存的异地更新方式,对部分页写采用了写前日志,避免了部分页写在异地更新中的其余数据的拷贝与写入开销.

事务闪存基于闪存异地更新特性,利用 SSD 内部的计算资源提供事务接口,是闪存存储中基于近数据处理的一个有效的例子.

4.2 对象闪存

对象闪存是基于闪存介质的对象式存储,无需经过闪存转换层、文件系统等层次.清华大学提出的软件直管闪存 OFSS 存储系统中的对象式闪存转换层 OFTL 即提供了对象闪存的功能.OFTL 当前在主机端以软件方式实现,但也可以在 SSD 内部通过硬件内固件方式实现.在 SSD 内部实现的对象闪存也是近数据处理的一种方式.

清华大学 2014 年提出的 p-OFTL[27] 是基于 OFTL 上进行的并行数据组织优化的对象闪存.p-OFTL 以对象的方式在闪存介质上进行数据的分配与索引,提供了对象的管理.p-OFTL 向上层软件系统提供了对象读、写、创建、删除、获取属性、设置属性、刷写等接口.

键值 SSD 与 p-OFTL 不同,向上层软件系统仅提供键值的 GET 和 PUT 操作接口.三星公司提出了键值 SSD(key-value SSD)[28],在 SSD 内部进行键值存储.这种方式减轻了主机端键值存储的压力,利用了 SSD 内部的资源.

不管是通用对象 SSD,还是键值 SSD,都是利用 SSD 内部资源的一种尝试.SSD 内部的内存大小等限制对于对象或键值管理都是一个挑战.如何在资源受限条件下实现高效的对象闪存仍需要进一步研究.

4.3 设备内文件系统

事务闪存和对象闪存分别向上提供了事务接口和对象接口,也有研究工作将更复杂的文件系统在设备上实现.威斯康星大学麦迪逊分校(University of Wisconsin-Madison)于 2018 年提出 DevFS[29],在存储设备内实现完整的文件系统,同时保留完整性、一致性、安全等特性.在主机端,DevFS 的用户态库支持应用通过 POSIX 接口操作文件,绕过操作系统内核.针对设备上内存的限制,DevFS 引入反向缓存(reverse-caching)机制,利用主机内存缓存文件系统的部分元数据.

设备内文件系统利用了设备内的计算和内存资源,能够避免陷入操作系统内核,减少系统调用的开销.但是,设备的硬件能力较弱,计算和内存资源有限.在多线程、元数据操作密集型等应用场景下,设备内文件系统对性能有较大的影响,快照、增量备份和去重等功能的实现也有所限制.

4.4 专用加速闪存

在应用层面上,闪存设备内部的计算资源为数据密集型的应用加速提供了机会,例如高性能计算模拟程序的数据分析、关系数据库查询.

4.4.1 Active Flash

北卡罗来纳州立大学(North Carolina State University)的 Active Flash[30],提出在超算中利用 SSD 就地分析模拟数据.

高性能计算模拟程序产生大量数据,需要对其进行分析.离线分析的方法引入大量冗余数据传输;节点分析的方法在数据产生后,加入暂存阶段,由额外的分析节点处理数据,带来额外的成本和延迟.

针对此问题,Active Flash 提出在暂存阶段,在部分模拟节点配备 SSD,与模拟程序并行执行,利用控制器空闲的计算资源就地分析数据.

Active Flash 基于 Jasmine OpenSSD 开发平台,在 FTL 固件加入了平均值、最大值、线性回归等简单的分析方法,构建原型系统. 主机通过目标为保留 LBA 的写命令,向 SSD 发送数据分析请求. SSD 控制器读取并分析数据,将结果输出到 LBA 列表指定的地址. 在控制器处理器为单核时,Active Flash 采取数据分析操作可被 I/O 操作抢占的调度策略,以降低对 SSD 性能的影响.

Active Flash 为 3 种数据分析方法,即离线分析、节点分析和固态盘就地分析,建立详细的能耗和性能模型,使用从实际应用测量的数据评估模型,研究将数据分析工作卸载到 SSD 上的可行性. Active Flash 提出的架构适用于大多数数据分析算法,而不影响到模拟程序的性能. 模拟实验和计算表明,利用固态盘就地分析数据的方法成本较低,能源利用效率更高.

4.4.2 Smart SSD

关系型数据库的查询是一种典型的数据密集型应用场景. 威斯康星大学麦迪逊分校(University of Wisconsin-Madison)提出的 Smart SSD[31],在三星商用 SSD 固件内搭建 NDP 环境,利用 SSD 的计算资源加速 Microsoft SQL Server 的查询操作.

Smart SSD 在 SATA/SAS 接口上定义了基于会话(session)的 NDP 通信协议,包括 3 个命令:OPEN,GET 和 CLOSE. 主机通过 OPEN 命令,在 Smart SSD 上建立一个会话,获得会话 id. 利用 GET 命令向 SSD 轮询会话状态以及程序运行的结果;最终通过 CLOSE 命令结束会话. 此外,Smart SSD 还提供了基础的线程、内存和数据 3 种类型 API,为程序提供 CPU 和内存资源以及闪存数据的访问接口.

Smart SSD 利用 API 在 SSD 中实现简单的查询和聚合操作算子,并根据定义的通信协议修改 SQL Server 的部分组件. 与普通 SSD 搭建的系统相比,Smart SSD 将查询操作执行速度提升 2.7 倍,而能耗减少了 3.0 倍.

4.5 通用加速闪存

在专用加速以外,一些工作研究利用设备的计算资源,构建通用的应用加速闪存系统.

4.5.1 iSSD

卡内基梅隆大学(Carnegie Mello University, CMU)于 2013 年提出 iSSD[32],将 Map Reduce 模型与 SSD 内部架构结合.

SSD 内每个通道都有闪存控制器(flash memory controller, FMC),存储在闪存中的数据传输到 FMC. iSSD 提出由 FMC 执行 Map 阶段,中间结果临时存储在 SSD 的 DRAM 或闪存中. Reduce 操作则由嵌入式 CPU 完成,最后将数据写回闪存或者传输到主机.

iSSD 在数据处理策略上,基于 FMC、嵌入式 CPU 和主机 CPU 的层次结构,提出了流水线式数据处理,以及主机与 SSD 之间的负载均衡. 并建立数学模型,分析验证 iSSD 架构的性能与功耗. 分析结果表明,对于线性回归、字符串匹配等数据密集型应用,iSSD 能够提升性能 2～4 倍,降低功耗 5～27 倍.

4.5.2 Willow

Willow[33]是加州大学圣地亚哥分校在 2014 年提出的考虑安全性、基于 RPC 的近数据处理的闪存存储系统.

Willow SSD 内部包括多个存储处理器单元(storage processor unit, SPU),每个 SPU 运行小的操作系统 SPU-OS. 在主机端,Willow 内核驱动管理一组主机 RPC 终端(host RPC endpoints, HRE),操作系统和应用利用 HRE 与 SPU 发送和接收 RPC 请求,完成主机与 SSD 的交互. 在此基础上,用户以 SSD App 的形式实现近数据处理的功能模块.

Willow 在支持 NDP 功能的同时,特别研究了 SSD App 的安全问题. 为此,Willow 追踪每个 RPC 所属的主机端进程,对 SSD 存储的数据、特殊 RPC 资源和 SPU 运行的代码与数据都做了权限检查等保护机制.

4.5.3 BlueDBM

麻省理工学院(MIT)的 BlueDBM[34]构建了近数据处理的分布式闪存存储系统. 不同于之前的工作,BlueDBM 在存储卡上加入网络接口,通过网络将多个存储卡连接. 系统中的存储设备使用统一的地址空间,提供远端与本地基本一致的访问时延.

BlueDBM 存储卡通过 PCIe 与主机连接,包括闪存芯片、近数据处理引擎、网络接口和 DRAM. 存储卡仅保留了 FTL 的错误校验功能,而将地址映射、垃圾回收等移动到主机端完成,支持物理地址访问闪存介质. 此外,BlueDBM 在主机端实现了完整 FTL 功能的块设备驱动、发挥裸闪存特性的文件系统.

在调用数据处理功能时,应用程序从文件系统获取文件存储的物理地址,将物理地址和数据绕过操作系统内核一同发送给硬件,减少内核软件栈带来的 I/O 开销.

最近邻搜索算法的实验表明:与基于普通 SSD 的系统相比,BlueDBM 的执行性能提升 10 倍.与内存计算系统相比,BlueDBM 与 RAMCloud 产生 10%的内存缺失,落到 SSD 的性能相当,但成本与功耗更低.

一些工作基于 BlueDBM 平台,进行应用层面的研究.例如 BlueCache[35] 构建可扩展的键值缓存系统;GraFBoost[36] 提出 Sort-Reduce 机制,设计并实现了图分析系统.

4.6 小 结

上述工作研究基于近数据处理的闪存存储系统,各有侧重,如表 2 所示.TxFlash,LightTx 和 DiffTx 设计并实现了闪存事务接口,p-OFTL 和 Key-Value SSD 构建基于闪存的对象式存储,DevFS 在存储设备内实现了完整的文件系统.在专用加速方面,Active Flash 建立模型,分析了基于 NDP 的闪存存储系统在分析大规模模拟程序输出数据的可行性;Smart SSD 加速了 SQL 数据库简单查询操作的效率,但支持的操作有限,在 SSD 固件上编程难度较大.在通用加速方面,iSSD 将 Map-Reduce 模型与 SSD 架构结合,通过模型分析了 iSSD 在数据密集应用中的性能与功耗;Willow 基于 RPC 机制支持主机与固态盘的通信,针对 NDP 程序的安全性问题建立了保护机制;BlueDBM 在存储卡加入网络端口,构建基于近数据处理的分布式闪存存储系统.

Table 2 Comparison of NDP-Based Flash Storage System

表 2 基于近数据处理的闪存存储系统比较

Category	Transaction	Object		File System	Domain-specific	Acceleration	General-purpose Acceleration
Systems	TxFlash, LightTx, DiffTx	p-OFTL	Key-Value SSD	DevFS	Active Flash	Smart SSD	iSSD, Willow, BlueDBM
Applications	Transcation	Object Storage	Key-Value Store	File System	Scientific Data Analysis	Simple SQL Queries	Linear Regression, Atomic Write, String Match, etc.

5 总结与展望

开放通道 SSD 的存储架构与系统避免了软硬件不同层次间的冗余,弥合层次间的语义鸿沟,为软硬件协同设计提供了较大的空间,在性能、闪存寿命、可靠性等方面均有明显收益.SSD 的近数据处理在设备内增加硬件计算逻辑,提供设备内的数据预处理,降低了数据在设备与主机端的数据传输量,在利用闪存硬件特性、充分使用内部带宽等方面效果明显.闪存的效率的充分发挥要求闪存存储系统在结构上和系统构建上进行变革.

除了开放通道和近数据处理 2 个研究方面之外,闪存的研究也在探索其他的软硬件结合的方式.其中,包括了较为保守的做法,即扩展现有闪存 SSD 的接口,由软件提供优化建议(hints),使得硬件获得更多信息以进行优化,如三星公司提出的多流设备以及多流文件系统[37].另外,端到端性能与公平性保证也是闪存存储栈上的研究热点,例如端到端的顺序性保证[38]、延迟保证[39]以及公平性保证[40].

但同时,闪存存储的重构与系统构建仍然面临着新的挑战:

1) 开放可编程闪存架构

开放通道 SSD 的设计以软件方式实现部分硬件功能,而近数据处理以硬件方式实现部分软件功能.尽管当前这两者是较为独立的研究方向,然而这两者本质并不冲突.闪存存储重构的新架构的设计初衷是寻求软硬件功能的合理划分和协作,目标是开放可编程闪存架构,即包括开放通道 SSD 与近数据处理的结合.开放可编程闪存架构要求软硬件协同的设计,在软硬件功能划分中,首先需要抽取合理大小的存储功能,并对相应功能进行抽象,以实现功能在软硬件实现中的优化配置;其次需要探寻合理的主机与 SSD 设备的 API 接口,在读写接口之外研究更灵活的数据交互机制;最后还需要研究软件读写擦操作与近数据操作之间的冲突处理,尤其近数据操作对外部读写擦操作的延迟影响.

2) 数据中心与嵌入式等领域的问题

闪存存储的新结构在"大规模"和"小型化"2 个方面面临新的挑战.在"大规模"数据中心领域,由于

多租户应用的存在,需要研究应用之间的隔离及公平,以提供更好的 QoS 保证[41-43];如何在混合负载下实现高效的调度或负载感知的管理算法以提供稳定可控的性能,也是提高云数据中心中虚拟机密度的研究方向之一.同样,在"小型化"的嵌入式系统领域,数据的实时性要求比较高,因而需要研究如何利用闪存新结构以减少内部垃圾回收等操作对前端性能的影响;嵌入式领域对于能耗要求严格,因而也需要研究软件的读写擦操作以及并发操作与能耗之间的关系.

3) 应用软件的协同设计

闪存的新架构为软件系统与 SSD 设备的协同设计提供了新的机遇.新的应用软件也呈现了新的特点,例如图计算、机器学习系统等新型应用.如何实现应用软件、系统软件以及 SSD 设备的垂直方向的协同设计,如何依据应用特征提取并抽象相应的功能以实现软硬件的优化配置,如何依据应用需求设计闪存的分配与调度策略以实现端到端的性能需求保证等问题都有待进一步的研究.

从磁盘到闪存固态盘是存储硬件领域的一次重大的变革,也是计算机存储子系统中的里程碑节点.自磁盘 20 世纪 50 年代出现以来,计算机存储系统多围绕磁盘而设计,在不同的层次中设计不可避免地受到磁盘特性的影响.闪存近年来从嵌入式至数据中心等不同领域中均得到了快速而广泛的应用,如何构建面向闪存的存储系统以发挥闪存效率是亟需研究的.这不仅是对现有系统的优化,更是对闪存存储新结构以及软件系统新设计的探索.清华大学在"重构闪存存储"上开展了深入的研究,取得了明显的效果.近年来学术界和业界也开始关注闪存新结构与新系统的研究与应用.然而,这个方向仍面临诸多挑战,许多问题值得大家进一步探索.

致谢 感谢清华大学计算机科学与技术系存储实验室曾经参与"重构闪存存储"研究工作的郭晓林、王维、秦雄军等硕士研究生和张佳程、廖晓坚等博士研究生,还有来自哥本哈根信息技术大学(IT University of Copenhagen)的访问博士生 Ivan Picoli,以及来自三星公司(Samsung)的访问学者 Wonchori Zoo 和 Hongsuk Choi 对本项目作出的贡献!

参 考 文 献

[1] Lu Youyou, Shu Jiwu. Survey on flash-based storage systems [J]. Journal of Computer Research and Development, 2013, 50(1): 49-59 (in Chinese)
(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)

[2] Lu Youyou, Shu Jiwu, Zheng Weimin. Extending the lifetime of flash-based storage through reducing write amplification from file systems [C] //Proc of the 11th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2013: 257-270

[3] Lu Youyou. Research on key technologies for flash-based file systems [D]. Beijing: Tsinghua University, 2009 (in Chinese)
(陆游游. 闪存文件系统关键技术研究[D]. 北京: 清华大学, 2009)

[4] Lu Youyou, Zhang Jiacheng, Shu Jiwu. Rethinking the file system design on flash-based storage [J]. Journal of the Korean Information Science and Engineers, 2015, 33(2): 42-51

[5] 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)
(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)

[6] Luara M G, John D D, Steven S. The bleak future of NAND flash memory [C] //Proc of the 10th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2012: 17-24

[7] Matias B, Javier G, Philippe B. LightNVM slides [OL]. [2018-11-01]. https://www.usenix.org/sites/default/files/conference/protected-files/fast17_slides_bjorling.pdf

[8] Wang Peng, Sun Guangyu, Jiang Song, et al. An efficient design and implementation of LSM-tree based key-value store on open-channel SSD [C] //Proc of the 9th ACM European Conf on Computer Systems. New York: ACM, 2014: No. 16

[9] Ouyang Jian, Lin Shiding, Jiang Song, et al. SDF: Software-defined flash for Web-scale Internet storage systems [C] //Proc of the 19th Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York: ACM, 2014: 471-484

[10] Lee S, Liu Ming, Jun S, et al. Application-managed flash [C] //Proc of the 14th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2016: 339-353

[11] Guo Xiaolin. Design and implementation of flash translation layer in solid state device and taint tracking in key-value system [D]. Beijing: Tsinghua University, 2012 (in Chinese)
(郭晓林. 固态存储系统转换层与键值系统错误污染检测的设计与实现[D]. 北京: 清华大学, 2012)

[12] Zhang Jiacheng, Shu Jiwu, Lu Youyou. ParaFS: A log-structured file system to exploit the internal parallelism of flash devices [C] //Proc of 2016 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2016: 87-100

[13] 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(5): No. 139

[14] Zhang Jiacheng. Research on key technologies for building open-channel SSD-based storage systems [D]. Beijing: Tsinghua University, 2018 (in Chinese)
(张佳程. 基于开放通道闪存架构的存储系统构建技术研究[D]. 北京:清华大学, 2018)

[15] Matias B, Javier G, Philippe B. LightNVM: The Linux open-channel SSD subsystem [C] //Proc of the 15th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2017: 359-374

[16] Lee C, Sim D, Hwang J, et al. F2FS: A new file system for flash storage [C] //Proc of the 13th USENIX Conf on File and Storage Technologies (FAST). Berkeyley, CA: USENIX Association, 2015: 273-286

[17] Josephson W K, Bongo L A, Flynn D, et al. DFS: A file system for virtualized flash storage [J]. ACM Transactions on Storage, 2010, 6(3): 14:1-14:25

[18] Lu Youyou, Shu Jiwu, Wang Wei. ReconFS: A reconstructable file system on flash storage [C] //Proc of the 12th USENIX Conf on File and Storage Technologies (FAST). Berkeyley, CA: USENIX Association, 2014: 75-88

[19] Shen Zhaoyan, Chen Feng, Jia Yichen, et al. DIDACache: A deep integration of device and application for flash based key-value caching [C] //Proc of the 15th USENIX Conf on File and Storage Technologies (FAST). Berkeyley, CA: USENIX Association, 2017: 391-405

[20] Acharya A, Uysal M, Saltz, J. Active disks: Programming model, algorithms and evaluation [C] //Proc of the 8th Int Conf on Architecture Support for Programming Languages and Operating Systems. New York: ACM, 1998: 81-91

[21] Riedel E, Gibson G A, Faloutsos C. Active storage for large-scale data mining and multimedia [C] //Proc of the 24th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 1998: 62-73

[22] Prabhakaran V, Rodeheffer T L, Zhou Lidong. Transactional flash [C] //Proc of the 8th USENIX Conf on Operating Systems Design and Implementation (OSDI). Berkeley, CA: USENIX Association, 2008: 147-160

[23] Lu Youyou, Shu Jiwu, Guo Jia, et al. LightTx: A lightweight transactional design in flash-based SSDs to support flexible transactions [C] //Proc of the 31st IEEE Int Conf on Computer Design (ICCD). Piscataway, NJ: IEEE, 2013: 115-122

[24] Lu Youyou, Shu Jiwu, Zhu Peng. TxCache: Transactional cache using byte-addressable non-volatile memories in SSDs [C] //Proc of the 3rd IEEE Nonvolatile Memory Systems and Applications Symp. Piscataway, NJ: IEEE, 2014: No. 7

[25] Lu Youyou, Shu Jiwu, Guo Jia, et al. Supporting system consistency with differential transactions in flash-based SSDs [J]. IEEE Transactions on Computers, 2016, 65(2): 627-639

[26] Ouyang Xiangyong, Nellans D, Wipfel R, et al. Beyond block I/O: Rethinking traditional storage primitives [C] // Proc of the 17th IEEE Int Symp on High Performance Computer Architecutre (HPCA). Piscataway, NJ: IEEE, 2011: 301-311

[27] Wang Wei, Lu Youyou, Shu Jiwu. p-OFTL: An object-based semantic-aware parallel flash translation layer [C] // Proc of the Conf on Design, Automation and Test in Europe (DATE). Piscataway, NJ: IEEE, 2014: No. 157

[28] Samsung Electronics Co. Ltd.. Samsung key value SSD enables high performance scaling [OL]. [2018-11-01]. https://www.samsung.com/semiconductor/global.semi.static/Samsung_Key_Value_SSD_enables_High_Performance_Scaling-0.pdf

[29] Kannan S, Arpaci-Dusseau A C, Arpaci-Dusseau R H, et al. Designing a true direct-access file system with DevFS [C] // Proc of the 16th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2018: 241-255

[30] Tiwari D, Boboila S, Vazhkudai S, et al. Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines [C] //Proc of the 11th USENIX Conf on File and Storage Technologies (FAST). Berkeyley, CA: USENIX Association, 2013: 119-132

[31] Do J, Kee Y S, Patel J M, et al. Query processing on smart SSDs: Opportunities and challenges [C] //Proc of the 2013 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 1221-1230

[32] Cho S, Park C, Oh H, et al. Active disk meets flash: A case for intelligent SSDs [C] //Proc of the 27th Int Conf on Supercomputing (ICS). New York: ACM, 2013: 91-102

[33] Seshadri S, Gahagan M, Bhaskaran S, et al. Willow: A user-programmable SSD [C] //Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 67-80

[34] Jun S W, Liu Ming, Lee S, et al. BlueDBM: An appliance for big data analytics [C] //Proc of the 42nd Annual Int Symp on Computer Architecture. New York: ACM, 2015: 1-13

[35] Xu S, Lee S J, Jun S W, et al. BlueCache: A scalable distributed flash-based key-value store [C/OL] //Proc of the 42nd Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2016 [2018-11-01]. https://dspace.mit.edu/bitstream/handle/1721.1/105947/965235977-MIT.pdf?sequence=1

[36] Jun S W, Wright A, Zang S, et al. GraFBoost: Accelerated flash storage for external graph analytics [C] //Proc of the 45th Annual Int Symp on Computer Architecture. New York: ACM, 2018: 411-424

[37] Rho E, Joshi K, Shin S, et al. FStream: Managing flash streams in the file system [C] //Proc of the 16th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2018: 257-264

[38] Won Y, Jung J, Choi G, et al. Barrier-enabled IO stack for flash storage [C] //Proc of the 16th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2018: 211-226

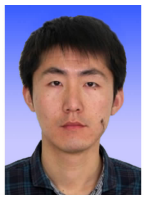
[39] Zhang Jie, Kwon M, Gouk D, et al. FLASHSHARE: Punching through server storage stack from kernel to firmware for ultra-low latency SSDs [C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementation (OSDI). Berkeley, CA: USENIX Association, 2018: 477-492

[40] Tavakkol A, Sadrosadati M, Ghose S, et al. FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives [C] //Proc of the 45th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 2018: 397-410

[41] Huang Jian, Badam A, Caulfield L, et al. FlashBlox: Achieving both performance isolation and uniform lifetime for virtualized SSDs [C] //Proc of the 15th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2017: 375-390

[42] Yan Shiqin, Li Huaicheng, Hao Mingzhe, et al. Tiny-Tail Flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs [C] //Proc of the 15th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2017: 15-28

[43] Hao Mingzhe, Li Huaicheng, Tong M H, et al. MittOS: Supporting millisecond tail tolerance with fast rejecting slo-aware OS interface [C] //Proc of the 26th ACM Symp on Operating Systems Principles (SOSP). New York: ACM, 2017: 168-183



Lu Youyou, born in 1987. PhD, assistant professor and PhD supervisor in Tsinghua University. Member of CCF. His main research interests include storage system, distributed system and computer architecture.

Yang Zhe, born in 1996. PhD candidate. His main research interests include flash-based storage systems and file systems.

Shu Jiwu, born in 1968. PhD, professor and PhD supervisor in Tsinghua University. Fellow of CCF. His main research interests include network storage and cloud storage, storage security and reliability, and parallel processing technologies.