计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220561
60(3): 555−571, 2023

多租户固态盘服务质量保障技术综述

文宇鸿[1,2] 周 游[2] 吴秋霖[1] 吴 非[1] 谢长生[1]
[1](武汉光电国家研究中心(华中科技大学) 武汉 430074)
[2](华中科技大学计算机科学与技术学院 武汉 430074)
(wenyuhong@hust.edu.cn)

Quality of Service Guaranty Technology of Multi-Tenant Solid-State Drives: A Survey

Wen Yuhong[1,2], Zhou You[2], Wu Qiulin[1], Wu Fei[1], and Xie Changsheng[1]
[1] (*Wuhan National Laboratory for Optoelectronics*(*Huazhong University of Science and Technology*), *Wuhan* 430074)
[2] (*School of Computer Science and Technology*, *Huazhong University of Science and Technology*, *Wuhan* 430074)

Abstract Sharing a solid-state drive (SSD) among multiple tenants has become a common way to improve the storage resource utilization and reduce the operating cost. This is driven by the high-density flash technology development, highly parallel storage architecture inside SSDs, and the full support of interface techniques. However, multiple tenants contend for the limited storage resources including the time resource and space resource of the SSD and interfere with each other. How to guarantee the quality of service (QoS) of multi-tenant SSDs has been a hot research topic in recent years. In this paper, we analyze and summarize the current state of research about the QoS technologies of multi-tenant SSDs. Firstly, we analyze the three QoS problems of multi-tenant SSDs: performance interference among tenants, performance unfairness among tenants, and loss of overall performance. Then, we present a comprehensive classification and introduction to existing work for improving the QoS of multi-tenant SSDs from three types of objectives, i.e., performance isolation, fairness guarantee, and overall performance optimization, from two dimensions of time resource and space resource. The technical evolution line of existing work is also sorted out. Finally, we conclude the existing techniques for QoS assurance of multi-tenant SSDs and discuss potential future research directions.

Key words flash memory; solid-state drive; multi-tenant; storage quality of service; performance isolation; fairness guarantee

摘 要 得益于高密度闪存技术发展、高并行存储架构以及良好的接口技术支持,多个租户共享使用一个固态盘已经成为提高存储资源利用率和降低运营成本的常见方式.然而,多个租户竞争使用固态盘内有限的存储资源,产生相互干扰,因此如何保障多租户固态盘服务质量成为近年来的研究热点.首先,分析多租户固态盘服务质量保障面临性能干扰、性能不公平及总体性能损失三大问题;然后,从如何保障性能隔离、如何保障性能公平及如何优化总体性能3类目标,对现有工作进行全面分类介绍,并梳理它们的技术演进方向;最后,对多租户固态盘服务质量保障技术的研究现状进行总结,并展望潜在的未来研究方向.

关键词 闪存;固态盘;多租户;存储服务质量;性能隔离;公平保障

收稿日期: 2022-06-16; **修回日期**: 2022-12-27

基金项目: 国家自然科学基金项目 (U2001203, 61902137, 61821003, 61872413); 中央高校基本科研业务费专项资金 (2021XXJS108, YCJJ202202002)
This work was supported by the National Natural Science Foundation of China (U2001203, 61902137, 61821003, 61872413) and the Fundamental Research Funds for the Central Universities (2021XXJS108, YCJJ202202002).

通信作者: 周游(zhouyou2@hust.edu.cn)

中图法分类号 TP391

信息技术发展促进数据量和数据密集型应用快速增长,对存储系统的容量、成本、性能和服务质量提出巨大挑战. 国际数据公司 IDC 预测[1]: 从 2018—2025 年,全球数据量将从 32ZB 增长至 175ZB,存储器出货容量将从 1.5ZB 增加至接近 5ZB. 其中,存储器以传统机械硬盘(hard disk drive, HDD)和闪存为主,闪存占比从 2018 年的不足 15% 将在 2025 年增加到 1/3 左右,而机械硬盘的占比从 2018 年的 65% 将在 2025 年缩小到 50% 左右.

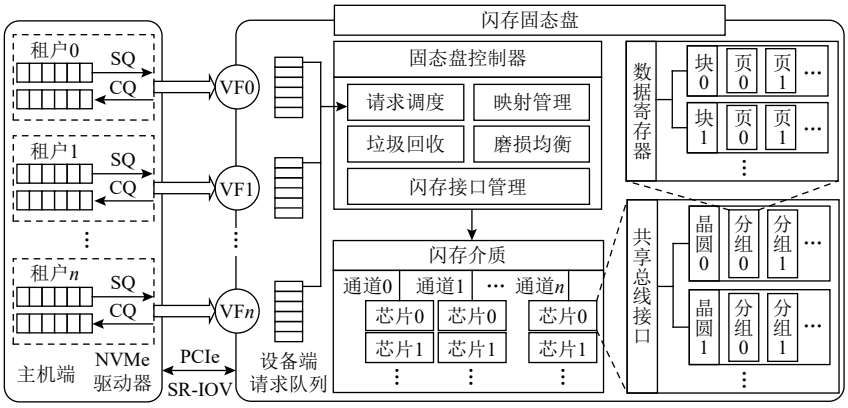
闪存的主要应用形态是固态盘(solid-state drive, SSD),相比于机械硬盘,它具有高性能、高可靠性、低能耗、小体积以及抗震动等优势. 得益于 3D 堆叠和多比特单元等闪存密度提升技术以及高度并行的内部存储架构,固态盘具有越来越高的容量、带宽和吞吐量. 例如,当前企业级固态盘的容量通常是 1~32 TB,读写带宽高达 1~7 GBps,吞吐量高达几十上百万 IOPS[2-5]. 而且,市场研究咨询公司 Wikibon[6] 于 2021 年的统计和预测显示,固态盘的单位容量存储成本近年来快速下降,将从 2026 年开始低于机械硬盘. 因此,固态盘将逐渐替代机械硬盘成为主流存储设备,在移动设备、嵌入式设备、个人电脑、服务器和云数据中心等各类存储系统中得到广泛应用[7-9].

为提高存储资源利用率和降低运营成本,多个租户(例如多个应用或虚拟机)共享使用同一个固态盘成为常见场景[10-13]. 大容量和高性能的固态盘能够满足多个租户的数据存储和访问需求. 例如,IBM 研究院针对多个企业数据中心的研究显示[14],单个虚拟机的存储容量需求一般是 100 多到几百 GB. 而且,固态盘接口技术为多租户共享提供了良好支持. 例如,PCIe(peripheral component interconnect express)接口的单根虚拟化(single root I/O virtualization, SR-IOV)特性[15]允许将一个固态盘虚拟化为多个存储设备,供主机使用;非易失性内存主机控制器接口规范(non-volatile memory express, NVMe)[16-17]支持高并行的多队列模型和多个命名空间,即能够为每个租户配置独立的 I/O 请求队列和逻辑地址空间. 另外,固态盘内部的并行存储架构也为实现多租户性能隔离提供了新契机.

1 固态盘背景介绍

固态盘主要由闪存介质、控制器以及主机接口组成,如图 1 所示.

闪存介质可划分成页(page)、块(block)、分组(plane)、晶圆(die)、芯片(chip)和通道(channel)这 6 层组织结构,这些闪存介质的组织结构形式使得闪存固态盘拥有多层级、丰富的内部并行性. 其中,闪存以页为读/写操作基本粒度,而以由大量页组织成的块结构为擦除操作基本粒度. 多个块组织成分组结构,共用一个数据寄存器,用于暂时存储写操作需要写入的页数据或者读操作读取出的页数据. 多个分组组织成晶圆结构,共用一个工作状态信号线,可通过多分组操作(multi-plane)命令[18-19]使同一晶圆内的多个分组并行执行同一闪存操作. 多个晶圆组织成芯片结构,共用一套外围电路和数据线,可通过交错操作(interleave)命令[18-19]以流水线的方式在等待一个晶圆完成访问操作的同时向同一芯片内的另一个晶圆传输数据,提高并行性. 为节省成本和空间,闪



提交队列(submission queue, SQ);完成队列(completion queue, CQ);虚拟功能(virtual function, VF)

Fig. 1 An illustration of multi-tenant SSD architecture

图 1 多租户固态盘结构图

存厂商通常将多个芯片通过通道结构集成在一起，共享一条 I/O 总线，不同通道可以并行执行读命令和写命令.

固态盘控制器负责请求调度、映射管理、垃圾回收(garbage collection, GC)、磨损均衡以及闪存接口管理等任务. 请求调度[20-21]任务负责从设备端请求队列中选择合适的排队请求进行处理；由于闪存介质存在写前擦除以及写粒度与擦除粒度不一致的特性，闪存固态盘通常采取异地更新的写入方式，因此需要映射管理[22-23]任务维护记录数据存放位置的地址映射表；异地更新会将新数据写入新页中，而将存储旧数据的旧页标记为无效状态，造成大量闪存空间存储着无效数据，因此需要垃圾回收[24-25]任务定期选择闪存块并对其进行擦除以获得空闲空间；闪存耐久远低于磁盘，闪存块擦除一定次数后会导致损坏，因此需要磨损均衡[26-27]任务将需要频繁更新的热数据写入磨损度较低的闪存块中，使得各闪存块能被均衡磨损，延长固态盘使用寿命；闪存接口管理[28]任务负责将请求调度模块选择的请求拆分成闪存页对齐的子请求，并根据闪存介质的状态将子请求发往目标芯片上执行.

闪存擦除操作必须以闪存块为粒度，在擦除一个块时，需要将其内的有效页迁移到其他块中以避免数据丢失，由此会产生额外的数据写入，造成写入放大，既影响闪存性能，又加剧闪存磨损，降低闪存使用寿命. 通常情况下，闪存固态盘内的实际物理空间要大于租户可见的逻辑空间，这些租户不可见的额外闪存物理空间称之为预留空间(over-provisioning space, OPS). 更大的预留空间可以降低垃圾回收擦除闪存块时的迁移开销(迁移更少的有效页)，缓解写入放大问题，提高固态盘性能及使用寿命[29].

主机接口负责主机与固态盘之间的通信和数据传输，从主机端获取 I/O 请求到设备端请求队列，对其进行解析和调度后下发给控制器进行处理，待处理完成后向主机报告 I/O 请求完成状态. 目前主流使用的物理接口包括 SATA(serial advanced technology attachment)，SAS(serial attached small computer system interface)和 PCIe 等. 其中，PCIe 是连接外围设备和主机处理器的主要接口，相比于 SATA 和 SAS 接口具有更低延迟和更高带宽(例如，PCIe 4.0×4 和 PCIe 5.0×4 的理论带宽分别是 8 GBps 和 16 GBps)，而且在硬件层面支持设备虚拟化. 这些物理接口采用的逻辑接口协议一般是 AHCI(advanced host controller interface)或者 NVMe，后者能够更好地发挥出固态盘的低延迟和高带宽优势，因此得到越来越广泛的使用. NVMe 协议最多支持 64 000 条最大深度为 64 000 的请求队列，而且支持分区命名空间命令集等高级特性.

2 多租户固态盘服务质量保障面临的挑战

多租户固态盘的关键设计目标是为租户提供存储服务质量(quality of service, QoS)保障，并尽可能提高存储资源利用率和总体性能. 除存储容量需求外，租户还具有多样化的 I/O 性能要求：延迟敏感型租户要求较低的平均延迟和有限的尾延迟；吞吐量敏感型租户对存储带宽具有一定要求；在更一般的情况下，需要为租户提供稳定可预测以及公平的性能. 此外，从存储服务商角度而言，需要尽可能高效利用固态盘资源以增加收益和降低成本.

由于多租户竞争使用固态盘内有限的存储资源，多租户固态盘服务质量保障面临 3 个关键挑战，如表 1 所示. 3 个挑战分别是：1)租户之间存在性能干扰，导致不稳定和不可预测的性能[10,30]；2)租户间无序的资源竞争可能会造成不公平的性能分配[11,31]；3)不理想的资源分配可能带来较低的资源利用率和总体性能[32-33]. 下面对这位 3 个问题进行具体阐述和分析.

2.1 多租户固态盘性能干扰问题

多租户固态盘技术使得多个租户能共享使用同一固态盘，但也引发了租户间的性能干扰问题，包括由 I/O 争用带来的访问冲突和调度干扰，以及多租户

Table 1 Categories and Descriptions of Challenges in Multi-Tenant SSDs

表 1 多租户固态盘面临的挑战类别及描述

挑战	类别	描述
性能干扰	I/O 干扰	①访问冲突：多租户访问同一闪存晶圆时，闪存介质的访问冲突造成租户间的 I/O 干扰 ②调度干扰：多租户并发请求间调度策略的调度选择造成租户间的 I/O 干扰
	GC 干扰	不同租户数据混合存储在同一闪存块时，GC 将涉及多个租户的数据迁移，产生 GC 干扰
性能不公平	性能比例不公平	各租户间性能比值不符合租户权重的比值
	性能减速不公平	各租户其单独运行时的性能相比于共享运行时的性能的减速程度不同
性能损失	资源配置低效	①多租户间的资源配置不佳，导致固态盘整体性能较差 ②为部分租户预留隔离的专属资源，产生资源闲置，导致固态盘性能损失

数据混合存储带来的 GC 干扰.

当多个租户访问同一闪存晶圆时，由于闪存晶圆是执行闪存操作的最小并行单元，因此不同租户请求之间将产生访问冲突. 与此同时，调度策略对多租户并发请求的调度选择也将产生调度干扰，这都会产生租户间的 I/O 干扰. 这种 I/O 干扰源于不同租户存在不同的负载特性，包括强度、大小、访问类型等. 例如，高强度负载单位时间内发出的 I/O 请求数更多，迫使低强度负载的 I/O 请求经历更长的排队等待时间；大型 I/O 负载每个 I/O 请求的完成时间更长，会显著增加小型 I/O 负载的请求平均响应时间；由于闪存读取速度比写入速度快 10~40 倍[34]，读请求排队等待写请求的完成时会经历显著的排队延迟，众多 I/O 调度器倾向于将读请求优先于写请求响应以获得更好的读性能[21,35]，然而这会使得多租户环境下读密集型负载会不公平地优先于写密集型负载获得响应，从而干扰写密集型负载的性能.

租户间的垃圾回收干扰源于不同租户数据的混合存储. 一方面，访问顺序性强的负载通常使得更新写产生的无效页更集中，进而降低固态盘垃圾回收的迁移开销，而多个共享租户数据的混合存储会破坏这种顺序性，造成垃圾回收效率下降及固态盘性能下降，因此具有顺序访问型负载会受到随机访问型负载的性能干扰；另一方面，不同租户数据的混合存储导致每次垃圾回收涉及多个租户数据的迁移，需要经常触发垃圾回收活动的负载（如写密集型负载、随机访问型负载等）会使得低强度垃圾回收活动负载（如读密集型负载、顺序访问型负载等）承担大量本不必承担的数据迁移开销，造成性能干扰.

共享多租户间的性能干扰会使得被干扰租户的性能出现显著波动，严重时将导致其服务质量目标无法获得满足.

2.2 多租户固态盘性能不公平问题

多租户共享使用同一闪存固态盘时，由于租户间的性能干扰及不公平的共享资源管理，各租户会获得不公平的资源分配并造成性能不公平问题. 性能的度量标准包括带宽、吞吐量以及请求延迟；而性能公平则分为比例公平与减速公平 2 类.

比例公平下的性能通常以可直接量化控制的固态盘带宽或吞吐量作为度量标准. 例如，Linux 系统中可基于其资源控制框架 Cgroup 对固态盘带宽或吞吐量进行分配管理[36]. 多租户性能比例不公平表现为各租户的带宽或吞吐量比值不符合租户间的权重比值,而性能减速公平则侧重于 I/O 请求的响应延迟. 由于共享使用同一固态盘的多个租户间存在性能干扰问题，各个租户与其他租户并发运行时的 I/O 请求响应延迟通常低于该租户单独使用固态盘时的响应延迟，二者的比值称为 I/O 请求响应延迟的减速程度（*Slowdown*），如式（1）所示，$Latency_i^{\text{alone}}$和$Latency_i^{\text{shared}}$分别代表租户 i 单独运行时和与其他租户并发运行时的 I/O 请求响应延迟. 具有公平保障的多租户固态盘应当确保各租户 I/O 请求响应延迟的减速程度相同，如式（2）所示. *Fairness* 表示公平性，其值越大代表该多租户固态盘保障的减速程度越公平，*Fairness* = 1 表示多租户间减速程度的完全公平.

$$Slowdown_i=\frac{Latency_i^{\text{shared}}}{Latency_i^{\text{alone}}}, \tag{1}$$

$$Fairness=\frac{\min\{Slowdown_i\}}{\max\{Slowdown_i\}}. \tag{2}$$

租户受到的性能干扰越严重，其获得的带宽或吞吐量越低，I/O 请求响应延迟的减速程度越高，因此公平性一定程度上也反映了多租户固态盘的性能隔离程度.

2.3 多租户固态盘性能损失问题

不同的租户有着不同类型的服务需求（低延迟、高带宽、大容量等），当这些不同需求的租户共享使用固态盘时，各类资源的不合理分配会导致较低的资源利用率，造成多租户固态盘的性能损失. 例如，由于预留空间资源可以提升垃圾回收效率，降低固态盘的写入放大，提升性能. 当较少触发垃圾回收的租户（如读密集型应用）获得了大量预留空间资源分配时，这些预留空间资源过剩的租户会造成共享运行的其他预留空间资源紧张的租户（如写密集型等需要频繁触发垃圾回收的应用）的垃圾回收效率严重降低. 与此同时，垃圾回收开销的增大进一步又使得预留空间资源过剩租户的请求被频繁阻塞造成性能下降. 因此，不合理的预留空间资源分配，会使得垃圾回收需要占用更多带宽资源，造成多租户固态盘的性能损失. 此外，为避免突发请求无法满足低延迟响应需求，管理员通常会为延迟敏感型租户预留专用资源，而由于突发请求属于低频事件，过量配置的资源会长时间处于低利用率状态而无法被其他繁忙的租户所利用，这也会造成多租户固态盘的性能损失.

3 多租户固态盘服务质量保障关键技术

多租户固态盘在发展的过程中面临着性能干扰、性能不公平和性能损失 3 类挑战，与之对应发展出

性能隔离保障、性能公平保障以及性能优化3类服务质量保障技术目标及研究方向，如图2所示：

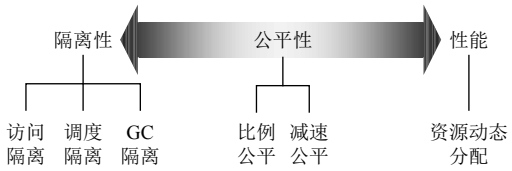


Fig. 2 Targets of QoS in multi-tenant SSDs

图2 多租户固态盘服务质量保障技术目标

性能隔离保障技术针对性能干扰问题，研究如何解决多租户对共享资源竞争带来的I/O干扰及GC干扰，实现I/O隔离（包括调度隔离及访问隔离）以及GC隔离，确保各租户负载特征发生变化时不会影响到其他租户的性能，保障各租户性能的稳定. 然而，性能隔离保障技术解决性能干扰问题的同时，意味着不合理的资源配置下，一个租户的过量配置资源或闲置资源无法被其他租户利用，必然会产生性能损失.

性能优化技术则以固态盘性能为优先考虑对象，研究如何动态调整资源配置，实现资源的高效利用，提升固态盘整体性能. 但资源的动态调整必然会引入租户间的干扰问题（被增加资源配置的租户的性能提升，被减少资源配置的租户的性能降低），且可能造成部分租户性能被严重牺牲.

性能公平保障技术则综合考虑隔离性与性能，其并不要求确保租户间性能的完全隔离，因此，允许共享资源的高效利用，但同时也并未完全放弃对租户干扰问题的管理，而是确保租户间的性能干扰对彼此造成的影响是公平的.

3.1 面向性能公平保障的服务质量保障技术

固态盘单位时间内的服务能力是有限的，从时间维度上的多租户共享竞争而言，固态盘完成一个租户的请求时间增多，必然导致完成其他租户的请求时间减少；而从空间维度上的多租户共享竞争而言，包括缓存、OPS等空间资源的分配同样影响着租户间的性能公平. 面向性能公平保障的多租户固态盘服务质量保障技术在时间维度上通过公平的I/O调度管理，包括基于预算分配以及基于队列仲裁2类策略，调整控制为各租户服务的时间；在空间维度上则通过公平的固态盘空间资源分配来影响租户的性能，最终实现多租户间的性能公平保障目标.

3.1.1 基于预算分配的I/O请求调度策略

基于预算分配的请求调度方案如图3所示，通过周期性的“预算分配—预算消耗—调度限制”过程实现I/O请求调度. 如图4所示，在每个调度周期的开始，基于预算的请求调度方案将为各租户分配性能预算，反映了各租户本周期内预期能获得的最大性能；每当一个I/O请求被调度时，对应租户的预算将减少该I/O请求对应的开销，预算的消耗反映了租户本周期内已实际获得的性能；一旦预算被消耗完，调度器当前周期内不再调度该租户的其余I/O请求，避免其继续获得超预期的性能，直至下一周期重新分配新的预算. 基于预算的请求调度方案既可以通过稳定的预算分配及消耗来保障租户间的性能隔离，同时也能通过公平的预算分配及消耗来保障租户间的性能公平.

Linux CFQ[37]中预算以时间片的形式划分给各租户，每个租户在自己的时间片内调度I/O请求并独享固态盘性能，只有当请求队列为空或时间片到期时才切换到下一租户. 然而由于闪存读写速度差异等原因，不同读写比的租户同等长度时间片内可以实现的性能并不一定相同，公平的时间片划分并不能完全保障性能公平. Linux BFQ[38]将带宽（以扇区为单位）作为预算进行划分，预算的多少可以直接反映出性能高低，确保公平的预算分配可以保障性能公平. 然而CFQ和BFQ方案均要求租户在一个连续的时间段内调度其I/O请求，当租户请求强度低而无法

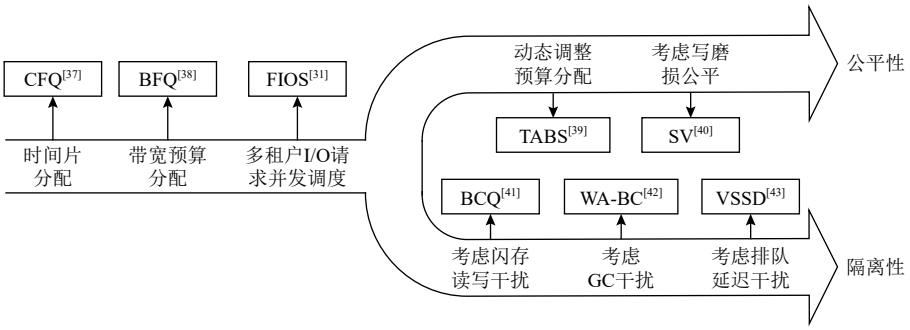


Fig. 3 Evolution of I/O request scheduling strategy based on budget allocation for multi-tenant SSDs

图3 多租户固态盘基于预算分配的I/O请求调度策略演进

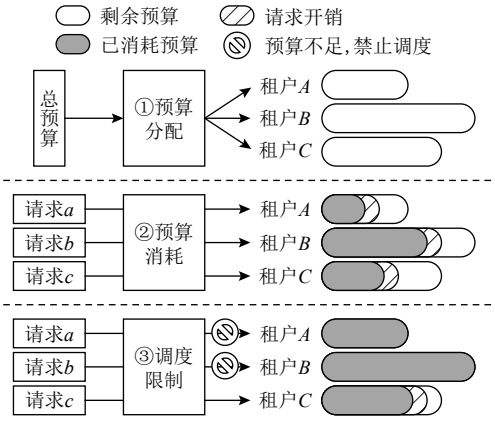


Fig. 4 I/O request scheduling based on budget allocation

图 4 基于预算分配的 I/O 请求调度

充分利用固态盘带宽时，要么会由于队列为空而提前结束其调度周期，造成租户剩余预算被不公平地没收，要么会等待至下一 I/O 请求的到来直至周期结束，造成固态盘性能的闲置浪费.

考虑到单租户可能无法充分利用固态盘的高并发性能，FIOS[31] 允许并发调度多个租户的 I/O 请求以充分利用闪存内部并行性提高性能. 然而同时调度多个租户的 I/O 请求，使得请求预算开销计算变得复杂. 例如，多个并发读请求的数据存储在同一闪存晶圆内时，由访问冲突等造成的排队延迟不应计入请求开销中，以确保预算消耗的公平性；当不同大小的读写请求并发执行时，考虑到闪存读写速度差异和请求大小等因素，其各自应当承担不同的请求开销. FIOS 通过预先对 4KB 读写请求和 128KB 读写请求的开销成本进行校准，基于请求开销与请求大小存在线性关系这一假设，建立请求开销线性模型，并在运行过程中基于该线性模型估算 I/O 请求开销.

考虑到并发读写请求之间的 I/O 干扰对性能公平性的影响，TABS[39] 在给读写请求分配公平带宽预算的同时，通过周期性根据每类请求的实际带宽和 I/O 强度来设置公平的带宽预算配置，动态调整预算分配，最终实现性能公平.

此外，SV[40] 认为在基于闪存固态盘的缓存架构下，多租户间对固态盘寿命磨损的公平与性能公平同样重要，因此将固态盘的写入次数也作为一类主要资源，以写预算的方式在租户间进行公平分配(超过预算的数据写入则由下一级存储处理)，在此基础上，采用博弈论中的沙普利值[44] 策略对共享租户垃圾回收产生的写开销进行公平分摊，实现租户间对闪存介质的公平磨损.

3.1.2 基于队列仲裁的 I/O 请求调度策略

图 5 展示了基于队列仲裁的 I/O 请求调度策略. 传统公平队列仲裁 RR(round-robbin)方案轮询调度每个租户的请求队列中的排队 I/O 请求，每调度完一个租户的 I/O 请求后就切换到下一个租户的请求队列中进行调度；WRR[45](weighted round-robbin)方案允许为各租户赋予不同权重，以实现有差别的性能，调度完的一个租户符合其权重的 I/O 请求个数后才切换到其他租户请求队列中进行调度；DRR[46](deficit round-robbin)方案则在赋予租户权重的同时，也为 I/O 请求赋予权重，即每个 I/O 请求的开销与请求大小等特征相关，调度完一个租户符合其权重的 I/O 总开销后才切换请求队列. 然而，这 3 种调度方案需要连续调度租户多个 I/O 请求直至满足其当前轮次的调度限制，当租户请求队列中的 I/O 请求个数不足时，其当前轮次无法获得足额性能，且也无法在后续轮次中被补偿，因此在 I/O 请求非匀速到达的场景下无法保障性能公平.

SFQ[47] 方案则为每个租户维护一个虚拟时间记录该租户历史性能，不同于上述轮询类调度方案只能保障每轮次的短期性能公平性，SFQ 方案可以保障各租户长期性能的公平性. 每完成一个 I/O 请求，SFQ 将该租户的虚拟时间向前推进该 I/O 请求对应的开销 L/W(L 代表 I/O 请求长度，W 代表租户权重)，通过优先调度性能最差的租户(其记录的虚拟时间最小)的 I/O 请求，最终保障各租户的性能公平. SFQ(D) [48] 则在 SFQ 方案基础上通过允许同时调度

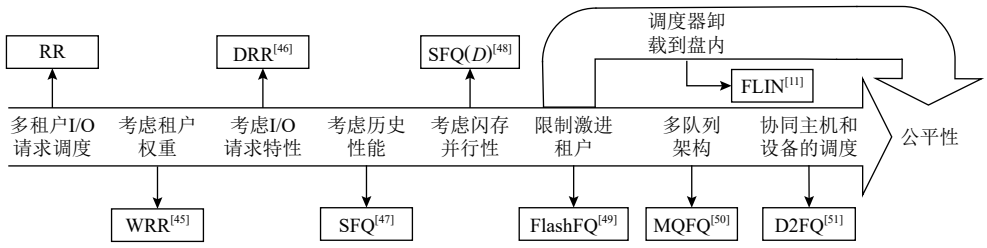


Fig. 5 Evolution of I/O request scheduling strategy based on queue arbitration for multi-tenant SSDs

图 5 多租户固态盘基于队列仲裁的 I/O 请求调度策略演进

D 个 I/O 请求，充分利用闪存固态盘内丰富的并行性能来提升固态盘性能. 然而同时调度多个租户的 I/O 请求带来高并发行的同时，也引入了租户间不公平的性能干扰，例如，随机型负载和顺序型负载并发运行时，随机型负载的高垃圾回收开销会严重干扰顺序型负载的性能等. FlashFQ[49] 在 SFQ(D)的基础上进行改进，通过阻塞性能过于超前的“激进”租户的 I/O 请求，避免其对其他租户 I/O 响应的干扰，以此提升“滞后”租户的性能，确保多租户间的性能公平.

由于现代高速固态盘技术的快速发展，存储系统的性能瓶颈逐渐从硬件性能转移到 I/O 栈软件开销. FLIN[11] 因此将 I/O 请求公平调度管理功能从主机端卸载到固态盘内实现，从而消除主机端操作系统管理 I/O 请求调度的软件栈开销. 通过“公平感知的队列插入—优先级感知的队列仲裁—排队均衡的调度选择”3 层机制，FLIN 可以在设备端实现公平请求调度. 同时由于在设备端可以确切地了解固态盘内部状态，包括空闲并行度、是否触发了垃圾回收等，FLIN 可以做出比主机端调度器更高效的调度决策，但也因此使得固态盘固件变得更为复杂，需要性能更强的盘内处理器支持.

随着多处理器等技术的进一步发展与普及，存储系统逐渐发展出包括 Linux 多队列块层[52]、NVMe 驱动器以及多队列固态盘[53] 等在内的多队列架构设计. 在多队列架构下，每个租户都拥有一个专用 I/O 队列用于直接与固态盘设备进行发送和接收 I/O 请求，以减少单队列集中管理的软件开销. 传统基于单队列集中管理的公平调度方案，由于其严格的公平性保障控制会带来严重的集中管理软件开销，已不再适用于多队列架构设计. MQFQ[50] 在多队列架构下改进 FlashFQ 方案，当激进租户的虚拟时间超过滞后租户 T 个单位时，通过可伸缩的数据结构实现激进租户请求队列的挂起及快速恢复，保障松弛的性能公平. MQFQ 方案通过配置合适的参数 T，既取得单队列集中调度请求带来的性能公平性，又获得多队列直通访问设备带来的高性能.

D2FQ[51] 则移除主机端复杂的公平调度器，利用 NVMe 协议提供的 WRR 队列仲裁功能，通过协同主机和设备的管理，实现轻量级的 I/O 公平调度器，进一步减少主机端 I/O 栈软件开销，在存储设备并非性能瓶颈的系统中保障公平的同时实现更高的性能. 具体而言，D2FQ 采取在主机端按照租户虚拟时间的激进程度将请求插入设备端不同权重的优先级队列中，而由设备端完成 WRR 队列调度，同时主机端基于租户间性能的不公平程度调整设备端优先级队列间的权重比，最终保障租户间的性能公平. 如图 6 所示，无公平 I/O 调度的 None 方案无法保障租户间的性能公平(租户间的性能比值不符合权重比值)，而方案 D2FQ, MQFQ 以及 BFQ 则保障了租户间的性能公平(权重为 8, 6, 4 的租户 w8, w6, w4 的性能分别是权重为 2 的租户 w2 的性能的 4, 3, 2 倍)，由于 MQFQ 及 D2FQ 方案避免了集中式公平调度管理，利用了多队列架构下的直通访问性能，极大降低了 I/O 栈软件开销，因此实现了 2 倍于传统 BFQ 方案的性能.

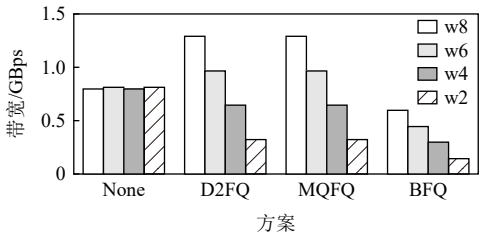


Fig. 6 Performance of different I/O request scheduling strategies for multi-tenant SSDs[51]

图 6 多租户固态盘不同 I/O 请求调度策略的性能[51]

3.1.3 空间资源公平分配策略

特别地，不同于 3.1.1 节和 3.1.2 节从 I/O 调度管理角度实现性能公平保障目标，OPS-Isolation[10] 发现由于垃圾回收需要迁移有效数据，造成写入放大，会产生额外的数据读写从而影响性能，而 OPS 大小与写入放大密切相关[29]. 因此，实现在设备端内的 OPS-Isolation 方案动态调控租户间的 OPS 资源分配从而实现对性能的控制，通过将超过公平比例性能的租户的 OPS 资源重新分配给低于公平比例性能的租户，提高后者的性能，最终实现租户间的性能公平. 方案 Justitia[54] 则通过动态调整多租户间盘内缓存资源的分配，减少数据缓存争用造成的性能不公平，预留部分缓存资源专用于低于公平比例性能的租户，由此保障共享多租户间的性能比例公平. 不同于 Justitia 通过管理盘内缓存资源分配实现块级 I/O 的性能公平，为避免主机端缓存对块级 I/O 公平性的扭曲，WaC[55] 通过管理主机缓存资源分配，根据 I/O 权重对等待获取锁的请求队列进行重排序，在缓存分配中获取锁的过程里优先考虑权重更高的应用，并在缓存回收时保障每个应用的缓存资源与 I/O 权重成正比，最终实现了应用级的 I/O 比例公平.

3.1.4 小 结

多租户固态盘通过 I/O 请求公平调度策略和空

间资源公平分配策略来保障共享多租户间的性能公平，如表 2 所示：

Table 2 Summary of QoS Technical for Multi-Tenant SSDs Oriented to Performance Fairness

表 2 面向性能公平的多租户固态盘服务质量保障技术总结

性能公平保障策略		核心思想	公平性	类别	典型方案
I/O 请求公平调度	队列仲裁	通过调整请求调度顺序，优先调度性能“滞后”租户的 I/O 请求	保障性能公平	比例公平	文献 [45-48,51]
				减速公平	文献 [11,49-50]
	预算分配	通过预算分配，限制租户每周期内的最大性能	仅保障周期内的公平性	比例公平	文献 [37-38,40]
				减速公平	文献 [31]
空间资源公平分配		通过动态调整缓存、OPS 等资源的分配间接影响租户性能	保障性能公平	比例公平	文献 [10,54-55]

I/O 请求公平调度策略包括基于队列仲裁和基于预算分配 2 类策略. 基于队列仲裁的 I/O 请求调度策略，通过控制 I/O 请求的调度顺序，优先调度性能“滞后”租户的 I/O 请求，保障调度选择的长期公平性；基于预算分配的 I/O 请求调度策略则通过预算分配的形式限制租户的最大性能，由此实现多租户固态盘性能公平保障目标. 然而由于基于预算分配的 I/O 请求调度策略仅保障周期内的公平性，上一周期内的性能不公平无法在下一周期中得到补偿和纠正，因此长期来看，其性能公平性弱于基于队列仲裁的 I/O 请求调度策略. 如图 7 所示，基于预算分配的 BFQ 方案性能公平性(即负载 aerospike 和负载 fio-4×4KB 在并发运行时性能相比于其单独运行时性能的减速程度的最大与最小的比值)优于无公平保障的 None 方案，但弱于基于队列仲裁的 MQFQ 方案.

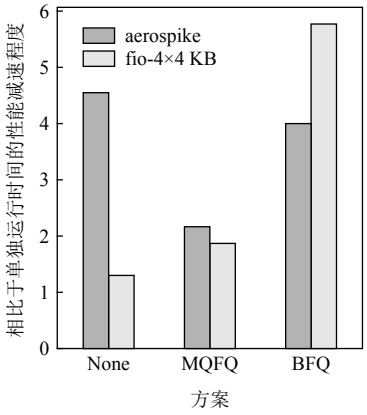


Fig. 7 Performance slowdown fairness results of different I/O request scheduling strategies for multi-tenant SSDs[50]

图 7 多租户固态盘不同 I/O 请求调度策略性能减速公平性测试结果[50]

空间资源公平分配则通过动态调整固态盘空间资源(包括缓存、OPS 资源)的分配来间接影响租户的性能表现，最终保障多租户间的性能公平. 然而，空间资源公平分配策略也存在着其局限性，例如，OPS 资源仅对写性能产生影响，当低于公平比例性能的租户为读密集型租户时，为其分配更多 OPS 资源不仅不能提升其性能，反而会降低被减少 OPS 资源的租户的写性能. 而将有限的缓存资源更多地分配给读密集型租户以提升其性能公平性的同时，也会降低用于写请求的缓存资源，由此增大了固态盘的写入放大，降低了固态盘的使用寿命及整体性能.

3.2 面向性能隔离保障的服务质量保障技术

面向性能隔离保障的多租户固态盘服务质量保障技术通过空间隔离和 I/O 隔离解决共享多租户间的性能干扰问题，最终实现性能隔离保障目标.

3.2.1 空间隔离策略

从空间维度上的共享竞争而言，多租户数据混合存储在同一闪存块内，使得固态盘执行垃圾回收时闪存块的擦除涉及到多个租户的数据迁移，由此产生租户间的 GC 干扰问题；与此同时，多租户可能同时访问同一闪存晶圆(独立执行闪存操作的最小并行单元)执行数据读取或写入，由此产生租户间的访问冲突.

多租户固态盘面向性能隔离保障的空间管理策略，根据数据隔离粒度的大小，划分成不同级别的数据隔离方案，旨在实现不同程度的性能隔离效果. 数据隔离粒度包括通道级隔离、芯片级隔离、晶圆级隔离、分组级隔离以及块级隔离，分别确保一个通道、芯片、晶圆、分组以及闪存块内只存储单个租户的数据. 其中通道级隔离、晶圆级隔离、块级隔离如图 8 所示. 由于闪存晶圆是闪存固态盘内可以独立执行闪存命令的最小并行单元，因此将晶圆级及以上级别(芯片级、通道级)的隔离称为硬隔离方案，而将分组级和块级隔离称为软隔离方案.

相比于无隔离方案，软隔离方案避免了不同租户数据被存储在同一闪存块内的情况，租户触发垃圾回收时仅选择擦除存储自身数据的闪存块，因此固态盘每次垃圾回收只涉及单个租户的数据迁移，

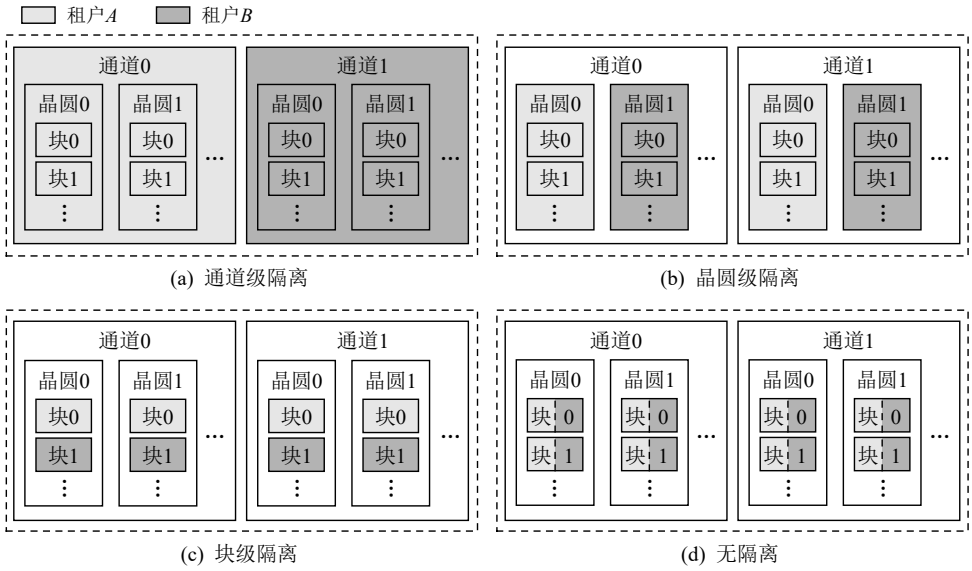


Fig. 8 Different levels of space isolation schemes for multi-tenant SSDs

图 8 多租户固态盘不同级别的空间隔离方案

实现了 GC 隔离保障. 特别地, 针对键值存储固态盘(key-value SSDs, KVSSDs)这类非传统块存储固态盘而言, 不仅数据在闪存物理介质上的混合存储会带来性能干扰, 其无隔离保障的日志结构合并树(log-structed merge-tree, LSM-tree)设计同样会造成性能干扰, 例如高强度写负载会抢占更多的低层有序字符串表(sorted string table, SSTable)空间, 使得低强度负载的数据被迫存储在更高层的 SSTable 中, 造成其糟糕的读性能. 针对该问题, Iso-KVSSD[56] 实现了一种支持多租户命名空间和性能隔离的键值存储固态盘, 为每个租户的命名空间配置专用 LSM-tree, 在第 0 层 SSTable 中混合存储多租户数据以提高缓存资源利用率, 在第 0 层 SSTable 数据往高层合并的过程中分离不同命名空间的数据并分别写入其对应的专用 LSM-tree 中, 最终保障多租户间的性能隔离.

相比于软隔离方案, 硬隔离方案不仅可以保障 GC 隔离, 同时也可以通过将不同租户数据存储在不同的闪存晶圆中, 从而避免不同租户访问同一闪存晶圆, 实现租户间的调度隔离及访问隔离. 例如, VFlash[57] 为每个租户提供独立的硬件资源, 包括盘内缓存、芯片级隔离的闪存资源等, 从而完全避免租户间的性能干扰, 实现强隔离性.

硬隔离方案可以保障强隔离性, 但也存在缺陷. 首先, 单个闪存晶圆容量可达数十 GB 大小[14], 如此粗粒度的数据隔离容易造成空间浪费及低利用率. 其次, 每个租户只能访问部分并行单元, 其最大并行带宽受限[43]. 如图 9 所示, PVR 负载在晶圆级硬隔离方案 DLS 下的性能约为分组级软隔离方案 PLS 下性能的 1/3, 约为块级软隔离方案 BLS 下性能的 1/4, 且一个租户的空闲并行单元无法被其他租户访问, 会造成性能损失[32]. 此外, 为了提高盘内数据可靠性, 固态盘可能采用晶圆级独立磁盘冗余阵列(redundant arrays of independent disks, RAID)技术[58] 来避免数据丢失, 而这与硬隔离方案相冲突, 因为各租户会产生对存储校验数据的晶圆的访问冲突, 破坏硬隔离方案的隔离性. 最后, 为延长固态盘使用寿命, 避免部分闪存块过早损坏, 固态盘通常需要进行磨损均衡, 例如将热数据(擦除更频繁)写入磨损程度较低的闪存块中, 而硬隔离方案下各租户闪存空间相互隔离, 阻碍了租户间的磨损均衡, 为多租户固态盘的寿命磨损带来了新的挑战[12].

考虑到硬隔离方案的上述缺陷, FlashBlox[12] 在原有磨损均衡算法保障各租户内闪存块公平磨损的

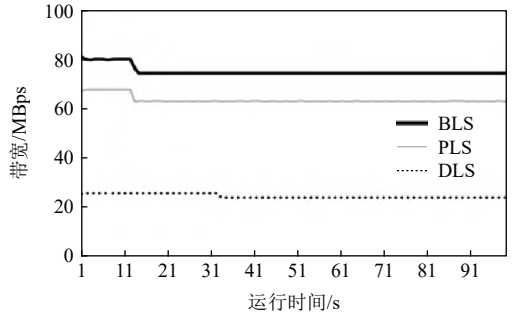


Fig. 9 Performance results for block-, plane-, and die-level isolation schemes[43]

图 9 块级、分组级和晶圆级隔离方案的性能测试结果[43]

基础上，通过补充设计租户间的磨损均衡算法，保障各租户间的公平磨损，提升多租户固态盘的使用寿命；CostPI[59]则考虑不同租户的性能需求，仅为延迟敏感型租户分配独立的闪存和缓存资源，而为其他类型租户（例如，带宽需求型租户、容量需求型租户等）分配共享闪存和缓存资源，避免为其他类型租户分配独立资源而产生资源闲置，在为延迟敏感型租户保障性能隔离的同时提升资源利用率及固态盘性能. LiveSSD[58]为解决硬隔离下盘内 RAID 奇偶校验更新引入的性能干扰问题，将具有相同偏移量的闪存页以 RAID-4 的方式构成一个条带，通过采用高速非易失性随机访问存储器（non-volatile random access memory, NVRAM）用于奇偶校验存储来避免奇偶校验更新造成的性能瓶颈，同时通过利用盘内空闲时间提前主动进行奇偶校验更新来避免其造成的 I/O 干扰，最终保障在硬隔离固态盘中实现 RAID 数据保护，并将奇偶校验更新造成的性能干扰降到最低.

3.2.2 I/O 隔离策略

多租户 I/O 请求调度策略包括基于队列仲裁的 I/O 请求调度和基于预算分配的 I/O 请求调度. 由于基于队列仲裁的 I/O 请求调度方案会优先调度性能“滞后”租户的 I/O 请求，其必然会影响性能“激进”租户的性能，即存在性能干扰问题，因此无法保障共享多租户间的性能隔离. 通常而言，面向性能隔离保障的多租户 I/O 请求调度策略采用基于预算分配的 I/O 请求调度方案（如图 3 所示），每租户的性能取决于其分配到的性能预算，而非 I/O 请求调度的先后顺序，由此实现调度隔离.

Linux CFQ 和 BFQ 采用互斥的调度策略，每次仅允许单个租户进行 I/O 调度，直至性能预算（时间片或带宽）耗尽才切换到其他租户进行 I/O 调度，因此不同租户间不存在 I/O 干扰. 随着多队列架构的发展，同时调度多个租户 I/O 请求的策略逐渐成为了充分发掘闪存固态盘内部并行性的必然选择，但也引入了多租户间的 I/O 干扰，如何感知并区分干扰的影响成为了能否保障性能隔离的关键.

FIOS 观察了读写请求间的性能干扰以及请求大小对固态盘并行性能利用率的影响，通过对 4KB 读写请求和 128KB 读写请求的开销成本进行预统计及分析，构建请求开销与请求类型和请求大小的线性关系，在共享运行过程中基于该线性模型计算 I/O 请求开销，避免多租户间的 I/O 干扰造成错误的开销计算及预算消耗.

然而请求开销不仅与 I/O 请求类型和大小相关，还与访问特性等其他因素相关. 例如，随机写负载的写入放大通常高于顺序写负载，随机读负载的并行性能通常低于顺序读负载等，因此随机读写请求的开销应高于顺序读写请求，FIOS 中仅基于 I/O 请求类型和大小的离线开销模型并不准确. BCQ[41]记录每个调度周期内完成的读写请求数，通过历史信息（如最近 20 个周期的数据）构成读写请求数与总读写开销关系的二元线性方程组，在线计算求解出读写平均开销来拟合后续 I/O 请求的预算开销.

然而，上述实现在主机端的 I/O 请求调度方案由于缺乏并发 I/O 请求间资源共享情况等设备级知识，其请求开销模型均无法解决由垃圾回收带来的性能干扰，因此其隔离性较弱. 在设备端实现的 WA-BC[42]方案在 BCQ 提出的请求开销计算模型基础上，结合空间软隔离方案，保障 GC 隔离的同时，考虑不同租户间的 GC 干扰. 其在计算得到读写请求平均开销后，基于各租户的写入放大将固态盘垃圾回收开销分摊到各租户写开销中，由此实现更强的性能隔离效果. VSSD[43]则将 I/O 请求的开销定义为所有子请求（闪存页大小）实际占用闪存时间（即完成一个读闪存页或写闪存页）之和，通过空间软隔离方案，使得各租户触发的垃圾回收（有效页迁移产生的读请求和写请求）由其自身承担开销，由此避免了租户间 I/O 冲突造成的调度干扰以及 GC 干扰，实现了多租户间的性能隔离保障.

3.2.3 小 结

面向性能隔离保障的多租户固态盘服务质量保障技术通过空间隔离及 I/O 隔离解决共享多租户间的性能干扰问题，如表 3 所示. 空间硬隔离方案通过为不同租户分配独立的硬件资源，完全避免多租户对固态盘资源的争用，由此实现调度隔离、GC 隔离以及访问隔离，表现出强性能隔离效果；空间软隔离方案允许多租户共享固态盘资源，但为不同租户分配不同闪存块，避免多租户数据混合存储在同一闪存块中，由此实现 GC 隔离；I/O 隔离方案采用基于预算分配的 I/O 请求调度器，通过稳定的预算分配及隔离的 I/O 请求开销模型，实现多租户间的调度隔离. 由于空间软隔离方案和 I/O 隔离方案各自只解决了部分性能干扰问题，因此其性能隔离效果均弱于空间硬隔离方案，但通过空间软隔离方案和 I/O 隔离方案的协同设计，其最终可实现等同于硬隔离方案的强性能隔离保障.

如图 10 所示，我们在广泛用于模拟固态盘的实验平台 SSDSim[60]中进行了不同级别的空间隔离方

Table 3 Summary of QoS Technical for Multi-Tenant SSDs Oriented to Performance Isolation

表 3 面向性能隔离的多租户固态盘服务质量保障技术总结

性能隔离保障策略			典型方案	主要措施	性能隔离效果
空间隔离	硬隔离	通道级	文献 [12,33,57,59]	为不同租户分配独立的硬件资源（缓存、闪存晶圆等）	强（调度隔离、GC 隔离、访问隔离）
		芯片级	文献 [32]		
		晶圆级	文献 [12,43,58]		
	软隔离	分组级	文献 [12,43]	多租户共享固态盘硬件资源，但将不同租户数据写入不同闪存块中，避免 GC 干扰	弱（GC 隔离）
		块级	文献 [10,42-43,56,59]		
I/O 隔离			文献 [41-43]	采用基于预算分配的 I/O 请求调度器	弱（调度隔离）

案的隔离性比较实验. 模拟的固态盘采用 4 个通道，每通道连接 4 个芯片，每芯片包含 2 560 个闪存块，每个闪存块包含 256 个 4 KB 大小闪存页，闪存读、写、擦除延迟分别为 60 μs, 800 μs, 1. 5 ms, 总线数据传输速度为 400 MBps, 预留空间占比 25%. 实验测试了 2 个租户并发运行时的性能, varmail(1), randwrite(1)代表共享租户 *A* 和 *B* 分别运行 varmail 和 randwrite 负载, 且二者权重比为 1∶1, 这 2 个负载均通过文件系统测试工具 Filebench[61] 收集而成, varmail 为写 I/O 占比 64.2% 的随机小型 I/O 负载, randwrite 为写 I/O 占比 100% 的随机 4KB I/O 负载. “硬隔离”方案采用通道级空间隔离, “软隔离”方案采用块级空间隔离, “I/O 隔离”方案采用 VSSD[43] 实现的 FACO 调度器, “无隔离”方案下多租户数据可能混合存储在同一闪存块中且多租户 I/O 请求采用无隔离保障的先进先出(first in first out, FIFO)调度策略. 从图 10 中可以发现, “软隔离+I/O 隔离”方案实现了等同于“硬隔离”方案的强隔离性, 即 varmail 负载的性能波动不会对 randwrite 负载的性能产生干扰; 而“软隔离”“I/O 隔离”和“无隔离”方案均无法保障租户间的性能隔离. 通过该 5 种隔离方案的性能表现可以发现, 保障了性能隔离的“硬隔离”和“软隔离+I/O 隔离”方案总性能更高; 而“软隔离”和“I/O 隔离”方案各自忽视了共享多租户间的 I/O 干扰或 GC 干扰, 仅保障了部分性能隔离, 最终导致其总性能弱于“硬隔离”和“软隔离+I/O 隔离”方案; “无隔离”方案则完全忽视了共享多租户间的性能干扰问题, 因此其总性能表现最差.

3.3 面向性能优化的服务质量保障技术

面向性能优化的服务质量保障技术通过优化固态盘资源分配来实现最大化资源利用效率和总体性能. 固态盘资源可分为时间资源和空间资源 2 个维度, 时间资源即固态盘为租户提供 I/O 服务的总时间(表现为时间片、带宽、吞吐量等资源), 空间资源则包括缓存、OPS 等资源.

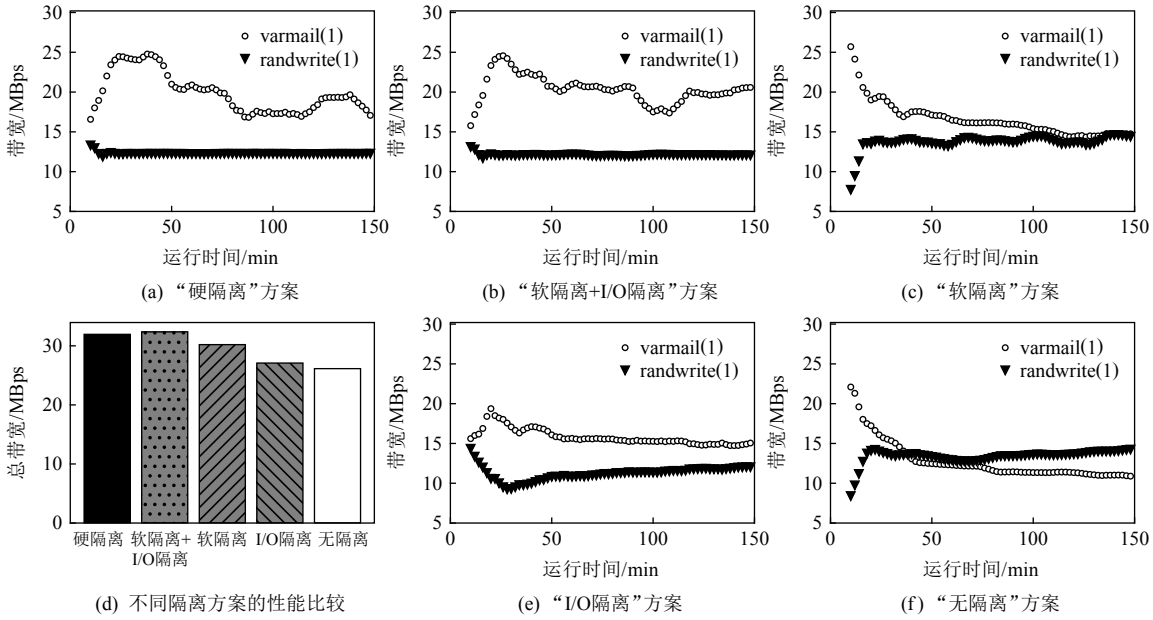


Fig. 10 Performance results of different levels of spatial isolation schemes for multi-tenant SSDs

图 10 多租户固态盘不同级别的空间隔离方案的性能结果

时间资源的分配可通过 I/O 请求调度策略实现控制，例如，在 Linux Cgroup 框架中实现的基于预算分配的 I/O 请求调度器 WDT[62]，通过跟踪各租户每周期的性能表现及时间资源(带宽预算)消耗情况，动态调整租户间的时间资源分配，以避免低强度租户分配到超过其需求的时间资源而导致的性能损失，在保障性能公平的同时提升固态盘整体性能.

空间资源的分配则通过固态盘控制器来实现调控，包括静态分区、共享竞争和动态分配 3 类，如表 4 所示，其中静态分区方案可以有效保障租户间的性能隔离，然而低效的静态分区配置和缺乏负载特性感知的共享竞争方案都无法最大化资源利用率，因此会造成性能损失. 我们依旧在 SSDSim 平台上进行测试，测试结果如图 11 所示. 图 11 中在前 150 min 内，租户 *A* 和租户 *B* 以 1∶3 的权重比分别运行 varmail 负载和 randwrite 负载；在后 150 min 内，租户 *A* 和租户 *B* 切换成以 3∶1 的权重比分别运行 randwrite 负载和 varmail 负载. “最佳分区(左)”和“最佳分区(右)”分别是通过逐一测试不同分区配置下的性能，最终分析获得在前 150 min 和后 150 min 内的最佳静态分区配置. “共享竞争”则是允许租户 *A* 和租户 *B* 自由竞争 OPS 资源. 从图 11 中，可以发现，无论是“最佳分区(左)”还是“最佳分区(右)”配置均只能实现局部最优性能，当多租户固态盘运行状态发生变化后，原本的局部最佳分区配置不再适应新的负载环境，因此空间资源的静态分区方案无法最大化固态盘资源利用率，导致低性能. 这表明静态分区策略保障性能隔离的同时也失去了优化资源分配以提升资源利用率及固态盘性能的机会，尽管可以让管理人员手动更改分区配置，但随着共享租户数量的增多及运行负载特征的复杂变化，“最佳分区”配置难以通过人工分析确定并及时手动调整；同样地，“共享竞争”方案的总体性能也远低于固态盘可达到的最佳性能. 因此，如何通过在运行时动态调整多租户间的空间资源分配以实现最大化固态盘整体性能则成为了面向性能优化的服务质量保障技术的一个重要研究问题.

Table 4 Summary of Space Resource Allocation Schemes for Multi-Tenant SSDs

表 4 多租户固态盘空间资源分配策略总结

空间资源分配策略	核心思想及优缺点
静态分区[12,43,57-59]	核心思想：将空间资源划分成固定的不同分区，每个租户分配使用其中一个分区 优点：不同租户数据分区存储，保障性能隔离 缺点：不合理的分区会造成资源利用率低，造成性能损失
共享竞争[42]	核心思想：所有租户共享固态盘的所有空间资源，租户间自由竞争空间资源的使用 优点：无需人工确定空间资源分区，可自适应负载变化 缺点：缺乏对负载特征的感知，无法最大化空间资源利用率，存在性能干扰问题
动态分配[10,32-33]	核心思想：基于机器学习、启发式算法等分配模型动态调整空间资源分配 优点：负载自适应，最大化空间资源利用率，提升固态盘整体性能 缺点：资源分配的动态调整引入了租户间的性能干扰

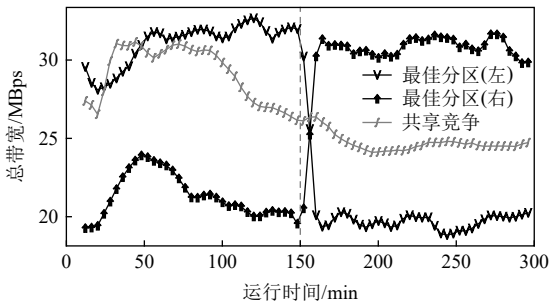


Fig. 11 Performance results of different space resource allocation schemes for multi-tenant SSDs

图 11 多租户固态盘不同空间资源分配方案的性能测试结果

SSDKeeper[33] 发现在数据通道级硬隔离方案下，多租户间不同的通道资源分配策略会产生不同的性能结果，最大性能差(平均请求延迟)可高达 10 倍，由此提出采用机器学习方法，通过人工神经网络离线训练，建立租户间负载强度比、负载读写比与不同分配策略的性能关系模型，并基于该模型在线动态调整租户间的通道资源分配，最终提升多租户固态盘的整体性能. UPI[32] 类似地采用数据芯片级硬隔离的动态分配方案，提出分配给各租户的闪存芯片集合的利用率计算模型，并基于该模型周期性动态调整空间分配，最终实现多租户固态盘空间资源整体利用率的优化. 然而，数据硬隔离方案下空间隔离粒度大(一个闪存晶圆容量可达数十 GB 大小[14])，因此动态分配开销极大，每次动态调整可能涉及 GB 数量级的数据迁移，这既会加剧固态盘的写入放大，同时也会使得动态调整速度极慢，存在滞后性，灵活性差. 基于空间软隔离方案实现低开销的空间动态调整技术以提升多租户固态盘性能尚有待进一步研究.

4 总 结

本文介绍了多租户固态盘面临的租户间性能干扰、性能不公平以及总体性能损失 3 大服务质量问题. 对应地，现有工作围绕性能隔离保障、性能公平保障以及总体性能优化 3 类目标对多租户服务质量保障技术展开研究.

具体而言，面向性能公平保障的多租户固态盘服务质量保障技术通过I/O请求公平调度策略（包括基于预算分配和基于队列仲裁2类I/O请求调度方案）以及空间资源公平分配策略保障共享多租户间的性能公平. 然而基于预算分配的I/O请求调度方案由于仅保障每周期内的性能公平，因此其公平性弱于其他方案. 此外，通过空间资源公平分配来实现性能公平也存在着局限性，例如. OPS资源无法优化读密集型租户的性能，缓存资源用于优化读密集型租户性能的同时会增大固态盘的写入放大，降低其使用寿命及性能等.

面向性能隔离保障的多租户固态盘服务质量保障技术通过空间隔离方案（包括硬隔离和软隔离2类不同隔离级别）和基于预算分配的I/O隔离方案保障共享多租户间的性能隔离，其中硬隔离方案实现了强隔离效果，而软隔离方案和I/O隔离方案分别只实现了GC隔离和调度隔离，因此其隔离效果弱于硬隔离方案，但软隔离方案可以与I/O隔离方案协同实现强隔离性效果.

面向性能优化的多租户固态盘服务质量保障技术通过感知负载变化，动态调整共享多租户间时间资源（表现为时间片、带宽、吞吐量等资源）和空间资源（包括缓存、OPS等资源）的分配，由此最大化资源利用率，最终实现固态盘的整体性能提升.

最后，各类多租户固态盘服务质量保障技术的实现层次包含主机端和设备端. 由于在设备端内才能感知闪存并行架构，控制闪存空间分配和I/O调度，因此基于空间资源分配（包括隔离分配、公平分配及负载自适应的动态分配）方案的服务质量保障技术通常需要实现在设备端；而关于I/O请求调度方案的服务质量保障技术，设备端实现相比于主机端实现具有设计空间更大、有效性更高等优势，但会增加固态盘控制器复杂度.

5 研究展望

现有工作虽然对多租户服务质量保障技术进行了不少探索，但该领域仍存在诸多尚未解决的关键挑战，值得进一步研究. 特别是，当前固态盘技术发展趋势包括采用更高密度闪存介质（如quad-level cell, QLC）、构建基于高速持久性内存和大容量闪存的混合架构以及转向新型分区命名空间接口（zoned namespace, ZNS），这些新型介质、架构以及接口会给多租户服务质量保障带来新的挑战.

1）现有工作没有综合考虑固态盘时间和空间2个维度上的资源分配，即如何建立时间资源和空间资源分配的统一模型，结合服务质量要求和I/O负载特征，为多个租户分别选择最佳的时间和空间资源分配，从而在保证服务质量的基础上最大化资源利用效率和总体性能. 例如，在保证性能公平的基础上，为以读请求为主的租户分配更多I/O时间预算，为包含大量随机写请求的租户分配更多空间资源（以降低垃圾回收开销）.

2）现有的以总体性能优化为目标的闪存空间动态分配方案都是基于硬隔离策略，存在空间分配粒度和重分配性能开销过大、单个租户并行带宽受限、不兼容固态盘内RAID保护技术等问题. 如何基于软隔离策略在多租户之间实现细粒度和低开销的动态空间分配，以最大化存储空间利用效率和总体性能，有待进一步研究.

3）高密度闪存技术发展为固态盘的多租户支持和可靠性带来挑战. 固态盘需要为每个租户打开至少一个独立的闪存块进行数据写入，以保证性能隔离，而且为租户分配更多开放闪存块能够有效提高冷热数据分离和垃圾回收的效率. 然而，出于可靠性因素，固态盘支持同时打开的闪存块最大数量受限[63]. 3D堆叠和多比特单元等闪存技术不断发展，推动存储密度提升和成本下降的同时，也导致介质可靠性变差，这将进一步减少固态盘支持，同时打开的闪存块最大数量. 因此，开放的闪存块也成为一种稀缺资源，对固态盘能够支持的最大租户数量以及租户性能都具有关键影响. 如何优化固态盘可靠性机制以提供更多的开放闪存块资源，以及如何在多租户之间分配开放闪存块资源以保证公平性和提高总体性能，都将成为重要挑战.

4）高速持久性内存技术和高密度闪存技术不断发展，使得构建混合存储架构的固态盘成为一种趋势，例如持久性内存作为闪存的缓存，或持久性内存和闪存分别存储不同数据集. 持久性内存具有支持字节寻址和读写速度不对称等特征，其加入将增加固态盘存储资源多样性和资源分配复杂性. 如何综合考虑2类存储资源的分配，并利用它们的介质访问特征，解决性能干扰、性能不公平以及总体性能损失等服务质量问题，将是一个重要的研究方向.

5）传统块接口将存储设备抽象为一个可随机写入和就地更新的逻辑块地址空间，这与需要顺序写入和先擦后写的闪存介质特性不符，使得固态盘需要采用复杂的控制器算法来匹配块接口抽象，因此

面临性能波动大和成本较高等问题[63]. 为解决这些问题, ZNS 接口被提出, 它将存储设备抽象为连续的、固定大小的、只允许顺序写入的逻辑分区. 该接口不仅能大幅简化固态盘控制器设计, 而且允许主机控制垃圾回收操作和闪存上数据布局. 然而, ZNS 接口为多租户固态盘设计和服务质量保障带来新的问题. 例如, 逻辑分区到闪存的映射方式决定了分区的访问性能和隔离性, 面向多租户的分区映射算法需要综合考虑各租户服务质量要求和闪存并行存储架构等因素; 出于可靠性考虑, ZNS 接口固态盘只能支持有限的开放分区资源(类似于传统固态盘的开放闪存块资源), 因此需要考虑该类资源在多租户之间分配的公平性和利用效率; ZNS 接口固态盘的逻辑分区尺寸通常大且固定, 而不同租户可能需求不同大小的分区, 以支持灵活的数据布局和降低垃圾回收开销, 如何解决该矛盾将是一个研究点难点.

综上所述, 构建多租户固态盘已经成为提高存储资源利用率和降低成本的常见手段, 但在服务质量保障方面仍存在诸多重要挑战, 值得进一步深入探索.

作者贡献声明: 文宇鸿完成了相关文献的梳理、论文撰写等工作; 周游完成了方案讨论、技术支持、论文修改等工作; 吴秋霖参与了论文相关的技术讨论等工作; 吴非完成了论文框架、整体内容规划等工作; 谢长生参与了论文指导等工作.

参 考 文 献

[1] IDC. Data age 2025 [EB/OL]. [2022-05-06].https://www.import.io/wp-content/uploads/2017/04/Seagate-WP-DataAge2025-March-2017.pdf

[2] DapuStor. DapuStor enterprise NVMe SSD – Haishen3 [EB/OL]. [2022-05-06].https://en.dapustor.com/prod_view.aspx?nid=3&typeid=66&id=185

[3] Memblaze. PBlaze®6 6530 series NVMe™ SSD [EB/OL]. [2022-05-06]. http://www.memblaze.com/en/index.php?c=article&a=type&tid=111

[4] Samsung. Samsung electronics debuts industry-leading 8TB consumer SSD, the 870 QVO [EB/OL]. [2022-05-06].https://news.samsung.com/global/samsung-electronics-debuts-industry-leading-8tb-consumer-ssd-the-870-qvo

[5] Solidigm. D5-P5316 SSD [EB/OL]. [2022-05-06].https://www.solidigmtech.com.cn/content/solidigm/us/en/products/data-center/d5/p5316.html

[6] Wikibon. Flash-native architectures power next-generation real-time workloads [EB/OL]. [2022-05-06].https://wikibon.com/flash-native-drives-real-time-business-process/

[7] 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)

[8] 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)

[9] 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)
(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)

[10] Kim J, Lee D, Noh S H. Towards SLO complying SSDs through OPS isolation[C]//Proc of the 13th USENIX Conf on File and Storage Technologies (FAST'15). Berkeley, CA: USENIX Association, 2015: 183−189

[11] 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 ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2018: 397−410

[12] 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'17). Berkeley, CA: USENIX Association, 2017: 375−390

[13] Kwon M, Gouk D, Lee C, et al. DC-Store: Eliminating noisy neighbor containers using deterministic I/O performance and resource isolation[C] //Proc of the 18th USENIX Conf on File and Storage Technologies (FAST'20). Berkeley, CA: USENIX Association, 2020: 183−191

[14] Birke R, Bjoerkqvist M, Chen L Y, et al. (Big) Data in a virtualized world: Volume, velocity, and variety in cloud datacenters[C] //Proc of the 12th USENIX Conf on File and Storage Technologies (FAST'14). Berkeley, CA: USENIX Association, 2014: 177−189

[15] Ding Zhimin. I/O virtualization in enterprise SSDs [C/OL] //Proc of the Storage Developer Conf. Santa Clara, CA: Storage Networking Industry Association, 2015[2022-05-06].https://www.snia.org/sites/default/files/SDC15_presentations/virt/ZhiminDing_IO_Virtualization_eSSD.pdf

[16] NVM Express. The NVMe 2.0 specification [EB/OL]. [2022-05-06]. https://nvmexpress.org/developers/

[17] Peng Bo, Zhang Haozhong, Yao Jianguo, et al. MDev-NVMe: A NVMe storage virtualization solution with mediated pass-through[C]//Proc of the USENIX Annual Technical Conf (USENIX ATC'18). Berkeley, CA: USENIX Association, 2018: 665−676

[18] ONFI. Open NAND flash interface specification [EB/OL]. [2022-05-

06].https://media-www.micron.com/-/media/client/onfi/specs/onfi_4_2-gold.pdf

[19] Hu Yang, Jiang Hong, Feng Dan, et al. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity[C] //Proc of the Int Conf on Supercomputing. New York: ACM, 2011: 96−107

[20] Nam E H, Kim B S J, Eom H, et al. Ozone (O3): An out-of-order flash memory controller architecture[J]. IEEE Transactions on Computers, 2010, 60(5): 653−666

[21] Elyasi N, Arjomand M, Sivasubramaniam A, et al. Exploiting intra-request slack to improve SSD performance[C] //Proc of the 22nd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 375−388

[22] Gupta A, Kim Y, Urgaonkar B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings[J]. ACM SIGPLAN Notices, 2009, 44(3): 229−240

[23] Kang J, Jo H, Kim J, et al. A superblock-based flash translation layer for NAND flash memory[C] //Proc of the 6th ACM & IEEE Int Conf on Embedded Software. New York: ACM, 2006: 161−170

[24] Lin W H, Chang Lipin. Dual greedy: Adaptive garbage collection for page-mapping solid-state disks[C] //Proc of the Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2012: 117−122

[25] Debnath B, Krishnan S, Xiao Weijun, et al. Sampling-based garbage collection metadata management scheme for flash-based storage[C/OL] //Proc of the 27th IEEE Symp on Mass Storage Systems and Technologies (MSST). Piscataway, NJ: IEEE, 2011 [2022-05-06].https://ieeexplore.ieee.org/abstract/document/5937228/

[26] Chang Yuanhao, Hsieh J W, Kuo Teiwei. Improving flash wear-leveling by proactively moving static data[J]. IEEE Transactions on Computers, 2009, 59(1): 53−65

[27] Murugan M, Du D H C. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead[C/OL] //Proc of the 27th IEEE Symp on Mass Storage Systems and Technologies (MSST). Piscataway, NJ: IEEE, 2011[2022-05-06].https://ieeexplore.ieee.org/abstract/document/5937225/

[28] Wu Guanying, He Xubin. Reducing SSD read latency via NAND flash program and erase suspension[C/OL] //Proc of the 10th USENIX Conf on File and Storage Technologies (FAST '12). Berkeley, CA: USENIX Association, 2012[2022-05-06]. https://www.usenix.org/legacy/event/fast12/tech/full_papers/Wu.pdf

[29] Hu Xiaoyu, Eleftheriou E, Haas R, et al. Write amplification analysis in flash-based solid state drives[C/OL] //Proc of the Israeli Experimental Systems Conf. New York: ACM, 2009[2022-05-06]. https://dl.acm.org/doi/abs/10.1145/1534530.1534544

[30] Kim J, Lee E, Noh S H. I/O scheduling schemes for better I/O proportionality on flash-based SSDs[C] //Proc of the 24th IEEE Int Symp on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). Piscataway, NJ: IEEE, 2016: 221−230

[31] Park S, Shen Kai. FIOS: A fair, efficient flash I/O scheduler [C/OL]//Proc of the 10th USENIX Conf on File and Storage Technologies (FAST '12). Berkeley, CA: USENIX Association, 2012[2022-05-06].https://www.usenix.org/legacy/event/fast12/tech/full_papers/Park.pdf

[32] Kim B S. Utilitarian performance isolation in shared SSDs[C/OL] //Proc of the 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '18). Berkeley, CA: USENIX Association, 2018[2022-05-06].https://www.usenix.org/system/files/conference/hotstorage18/hotstorage18-paper-kim-bryan.pdf

[33] Liu Renping, Liu Duo, Chen Xianzhang, et al. Self-adapting channel allocation for multiple tenants sharing SSD devices[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 41(2): 294−305

[34] Yang Mingchang, Chang Yuanhao, Tsao C W, et al. Utilization-aware self-tuning design for TLC flash storage devices[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016, 24(10): 3132−3144

[35] Wu Guanying, He Xubin. Reducing SSD read latency via NAND flash program and erase suspension[C/OL] //Proc of the 10th USENIX Conf on File and Storage Technologies (FAST '12). Berkeley, CA: USENIX Association, 2012[2022-05-06].https://www.usenix.org/legacy/event/fast12/tech/full_papers/Wu.pdf

[36] Paul M. CGROUPS [EB/OL]. [2022-05-06].https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html

[37] Axboe J. Linux block IO—Present and future[C/OL] //Proc of the Ottawa Linux Symp. 2004: 51−61[2022-05-06]. http://www.mnis.fr/fr/services/virtualisation/pdf/LinuxSymposium2004_V1.pdf#page=51

[38] Valente P, Avanzini A. Evolution of the BFQ storage-I/O scheduler[C] // Proc of the Mobile Systems Technologies Workshop (MST). Piscataway, NJ: IEEE, 2015: 15−20

[39] Kim J, Kim D, Won Y. Fair I/O scheduler for alleviating read/write interference by forced unit access in flash memory[C] //Proc of the 14th ACM Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2022: 86−92

[40] Choi W, Urgaonkar B, Kandemir M, et al. Fair write attribution and allocation for consolidated flash cache[C] //Proc of the 25th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 1063−1076

[41] Zhang Quan, Feng Dan, Wang Fang, et al. An efficient, QoS-aware I/O scheduler for solid state drive[C] //Proc of the 10th IEEE Int Conf on High Performance Computing and Communications & 2013 IEEE Int Conf on Embedded and Ubiquitous Computing. Piscataway, NJ: IEEE, 2013: 1408−1415

[42] Jun B, Shin D. Workload-aware budget compensation scheduling for NVMe solid state drives[C/OL] //Proc of the IEEE Non-Volatile Memory System and Applications Symp (NVMSA). Piscataway, NJ: IEEE, 2015 [2022-05-06]. https://ieeexplore.ieee.org/abstract/document/7304369

[43] Chang Dawei, Chen H H, Su Weijian. VSSD: Performance isolation in a solid-state drive[J]. ACM Transactions on Design Automation of Electronic Systems, 2015, 20(4): 1−33

[44] Nowak A S, Radzik T. The Shapley value for *n*-person games in

generalized characteristic function form[J]. Games and Economic Behavior, 1994, 6(1): 150−161

[45] Joshi K, Yadav K, Choudhary P. Enabling NVMe WRR support in Linux block layer[C/OL] //Proc of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'17). Berkeley, CA: USENIX Association, 2017[2022-05-06].https://www.usenix.org/conference/hotstorage17/program/presentation/joshi

[46] Shreedhar M, Varghese G. Efficient fair queueing using deficit round robin[C] //Proc of the Conf on Applications, Technologies, Architectures, and Protocols for Computer Communication. New York: ACM, 1995: 231−242

[47] Goyal P, Vin H M, Chen Haichen. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks[C] //Proc of the Conf on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM, 1996: 157−168

[48] Jin Wei, Chase J S, Kaur J. Interposed proportional sharing for a storage service utility[J]. ACM SIGMETRICS Performance Evaluation Review, 2004, 32(1): 37−48

[49] Shen Kai, Park S. FlashFQ: A fair queueing I/O scheduler for flash-based SSDs[C] // Proc of the USENIX Annual Technical Conf (USENIX ATC '13). Berkeley, CA: USENIX Association, 2013: 67−78

[50] Hedayati M, Shen Kai, Scott M L, et al. Multi-queue fair queuing[C] //Proc of the USENIX Annual Technical Conf (USENIX ATC'19). Berkeley, CA: USENIX Association, 2019: 301−314

[51] Woo J, Ahn M, Lee G, et al. D2FQ: Device-direct fair queueing for NVMe SSDs[C] //Proc of the 19th USENIX Conf on File and Storage Technologies (FAST'21). Berkeley, CA: USENIX Association, 2021: 403−415

[52] Bjørling M, Axboe J, Nellans D, et al. Linux block IO: Introducing multi-queue SSD access on multi-core systems[C/OL] //Proc of the 6th Int Systems and Storage Conf. New York: ACM, 2013[2022-05-06].https://dl.acm.org/doi/abs/10.1145/2485732.2485740

[53] SanDisk. Skyhawk & Skyhawk ultra NVMe PCIe SSD [EB/OL]. [2022-05-06].https://www.sandisk.com/content/dam/sandisk-main/en_us/assets/resources/data-sheets/Skyhawk-Series-NVMe-PCIe-SSD-DS.pdf

[54] Liu Renping, Tan Zhenhua, Long Linbo, et al. Improving fairness for SSD devices through DRAM over-provisioning cache management[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(10): 2444−2454

[55] Park J, Eom Y I. Weight-aware cache for application-level proportional I/O sharing[J]. IEEE Transactions on Computers, 2021, 71(10): 2395−2407

[56] Min D, Kim Y. Isolating namespace and performance in key-value SSDs for multi-tenant environments[C] //Proc of the 13th ACM Workshop on Hot Topics in Storage and File Systems. New York: ACM, 2021: 8−13

[57] Song Xiang, Yang Jian, Chen Haibo. Architecting flash-based solid-state drive for high-performance I/O virtualization[J]. IEEE Computer Architecture Letters, 2013, 13(2): 61−64

[58] Zhou You, Wu Fei, Huang Weizhou, et al. LiveSSD: A low-interference RAID scheme for hardware virtualized SSDs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 40(7): 1354−1366

[59] Liu Jiahao, Wang Fang, Feng Dan. CostPI: Cost-effective performance isolation for shared NVMe SSDs[C/OL] //Proc of the 48th Int Conf on Parallel Processing. New York: ACM, 2019[2022-05-06].https://dl.acm.org/doi/abs/10.1145/3337821.3337879

[60] Hu Yang, Jiang Hong, Feng Dan, et al. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance[J]. IEEE Transactions on Computers, 2012, 62(6): 1141−1155

[61] Github. Filebench [CP/OL]. [2022-05-06]. https://github.com/filebench/filebench/wiki.

[62] Ahn S, La K, Kim J. Improving I/O resource sharing of Linux Cgroup for NVMe SSDs on multi-core systems[C/OL] //Proc of the 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'16). Berkeley, CA: USENIX Association, 2016[2022-05-06].https://www.usenix.org/conference/hotstorage16/workshop-program/presentation/ahn

[63] Matias B, Abutalib A, Hans H, et al. ZNS: Avoiding the block interface tax for flash-based SSDs[C] //Proc of the USENIX Annual Technical Conf (USENIX ATC '21). Berkeley, CA: USENIX Association, 2021: 689–703

Wen Yuhong, born in 1999. PhD candidate. His main research interests include non-volatile memory, solid-state storage architectures, multi-tenant storage quality of service.

文宇鸿, 1999 年生. 博士研究生. 主要研究方向为非易失性存储器、固态存储体系结构以及多租户存储服务质量.

Zhou You, born in 1990. PhD, associate professor, PhD supervisor. Member of CCF. His main research interests include non-volatile memory, device and system.

周　游, 1990 年生. 博士, 副研究员, 博士生导师. CCF 会员. 主要研究方向为非易失性存储器、设备及系统.

Wu Qiulin, born in 1995. PhD. His main research interests include non-volatile memory, memory management, flash storage systems, and file systems.

吴秋霖, 1995 年生. 博士. 主要研究方向为非易失性存储器、内存管理、闪存存储系统以及文件系统.

Wu Fei, born in 1975. PhD, professor, PhD supervisor. Senior member of CCF, member of IEEE and ACM. Her main research interests include computer architecture, non-volatile memory, intelligent storage.

吴　非, 1975 年生. 博士, 研究员, 博士生导师. CCF 高级会员, IEEE 和 ACM 会员. 主要研究方向为计算机体系结构、非易失性存储器及智能存储.

Xie Changsheng, born in 1957. PhD, professor, PhD supervisor. Senior member of CCF, member of IEEE and ACM. His main research interests include emerging storage system.

谢长生, 1957 年生. 博士, 教授, 博士生导师. CCF 高级会员, IEEE 和 ACM 会员. 主要研究方向为融合存储系统.