https://crad.ict.ac.cn
62(1): 58−74, 2025

移动设备日志结构文件系统综述

杨梨花[1] 董 勇[1] 邬会军[1] 谭支鹏[2] 王 芳[2] 卢 凯[1]
[1](国防科技大学计算机学院 长沙 410073)
[2](信息存储系统教育部重点实验室(华中科技大学) 武汉 430074)
(yanglihua@nudt.edu.cn)

Survey of Log-Structured File Systems in Mobile Devices

Yang Lihua[1], Dong Yong[1], Wu Huijun[1], Tan Zhipeng[2], Wang Fang[2], and Lu Kai[1]
[1](*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410073)
[2](*Key Laboratory of Information Storage System* (*Huazhong University of Science and Technology*), *Ministry of Education*, *Wuhan* 430074)

Abstract NAND flash is widely utilized in mobile devices due to its excellent characteristics, including large capacity, light weight, and shock resistance. The flash friendly file system (F2FS), designed for flash features, is a typical log-structured file system (LFS). It employs a log-structured write mechanism to enhance random write performance, utilizes roll-forward recovery technology for fast consistency protection, and is commonly used as a file system for mobile devices. However, the performance of file system is impacted by fragmentation and segment cleaning. The out-of-place update mechanism of LFS and the small write mode of high-concurrency and random synchronization of mobile applications exacerbate fragmentation, leading to sluggish I/O request responses and device operation freezes. We initially introduce the relevant concepts and content of log-structured file systems in mobile devices. We then primarily outline the research status of fragmentation and segment cleaning of LFS. Firstly, we analyze the generation and impact of fragmentation, summarize the research work on reducing fragments from the perspectives of preventing fragments and reorganizing fragments. Secondly, we examine the impact of the mixed storage of hot and cold data on segment cleaning. Additionally, we summarize the research status of distinguishing hot and cold data from static and dynamic classification, and segment cleaning from the perspectives of managing data distribution and adjusting the timing, frequency, and objects of segment cleaning. Finally, we outline the main challenges and future research prospects of log-structured file systems in mobile devices.

Key words log-structured file system; mobile storage; file fragmentation; free space fragmentation; segment cleaning; hot and cold data distinction

摘 要 NAND 闪存(NAND flash)因为其大容量、轻便、抗震等优异特性,被广泛使用于移动设备.面向闪存特性设计的闪存友好型文件系统(flash friendly file system, F2FS)是典型的日志结构文件系统(log-structured file system, LFS),它采用日志结构写机制提升了随机写性能,使用前滚恢复技术实现快速的一致性保护,经常被用作移动设备的文件系统.文件系统因碎片化和段清理问题导致性能下降,而日志结构

收稿日期: 2023-10-10; **修回日期**: 2024-06-19
基金项目: 全国重点实验室基金项目(2023-KJWPDL-11); 湖南省自然科学基金项目 (2024JJ6471, 2023RC3021); 国家自然科学基金项目(62306328, U22B2005); 高性能计算国家重点实验室基金项目(202101-03)
This work was supported by the National Key Laboratory Foundation of China (2023-KJWPDL-11), the Natural Science Foundation of Hunan Province of China (2024JJ6471, 2023RC3021), the National Natural Science Foundation of China (62306328, U22B2005), and the Foundation of State Key Laboratory of High Performance Computing (202101-03).
通信作者: 董勇(yongdong@nudt.edu.cn)

文件系统的异地更新机制和移动应用的高并发随机同步小写模式进一步加剧了碎片化，导致I/O请求响应变慢、设备运行卡顿. 首先介绍了移动设备日志结构文件系统的相关概念和内容，随后总结了日志结构文件系统碎片化和段清理问题的研究现状. 一方面分析了碎片产生的原因与影响，从预防碎片产生和重整碎片2个角度总结了减少碎片的研究工作. 另一方面分析了冷热数据混合对段清理的影响，从静态分类和动态分类2方面总结了冷热数据区分技术的研究现状，从管理数据分布和调整段清理时机、频率、对象2个角度总结了段清理的研究现状. 最后展望了移动设备日志结构文件系统研究的主要挑战和未来研究工作.

关键词 日志结构文件系统；移动存储；文件碎片；空闲空间碎片；段清理；冷热数据区分

中图法分类号 TP302.1

DOI: 10.7544/issn1000-1239.202330789 **CSTR:** 32373.14.issn1000-1239.202330789

移动设备，包括智能手机、平板、可穿戴设备、智能汽车等，在日常生活中不可或缺. 移动设备广泛使用轻便且容量大的NAND闪存(NAND flash)作为存储器. 闪存友好型文件系统(flash friendly file system, F2FS[1])针对闪存特性设计，是一种典型的日志结构文件系统(log-structured file system, LFS[2]). F2FS采用日志结构方式的写入和闪存物理单元友好的数据布局方法，提高了随机小写性能，通过前滚恢复和检查点操作实现低成本且高效的一致性保护，保障了移动设备多支付场景的功能和性能，因此在移动设备中广泛使用. 其性能优化具有重要的经济收益，因而近年来受到学术界和工业界的高度关注. 使用NAND闪存的移动设备在近十年高速发展，其中智能手机的需求量最大、使用最为普遍，目前全球智能手机用户数量已超过60亿.

表1列举了使用F2FS作为文件系统的商用移动设备，从2020年9月发布的移动终端操作系统Android 11开始，安卓开源项目(Android open source project, AOSP)的动态系统更新机制要求data分区使用F2FS或第4代扩展文件系统(fourth extended file system, ext4[3]).

Table 1 Commercial Mobile Devices Using the F2FS

表1 使用F2FS的商用移动设备

生产年份	商用移动设备
2012—2015	摩托罗拉MSM8960 JBBL设备/ Droid/ G/ X系列
2014—2016	谷歌Nexus 9、摩托罗拉E LTE/Z、一加3T、华为Honor 8/ V8/ P9/ Mate 9
2018—2019	谷歌Pixel 3/3 XL、中兴Axon 10 Pro、三星Galaxy Note 10/Tab S6
2020—	基于Android 11的谷歌Pixel系列

无论是机械硬盘还是固态硬盘，都更擅长处理各种顺序读写请求，日志结构技术因将小的随机写请求组合成一个较大的顺序写入请求来更好地发挥存储硬件设备性能. 传统的日志结构文件系统性能依赖于连续的空闲空间，维护这些空间需要开销非常大的段清理操作. 而文件碎片和空闲空间碎片使得连续的文件数据和空闲空间不再连续，造成文件系统性能下降. 因此，本文围绕移动设备日志结构文件系统的碎片化和段清理问题，主要讨论产生原因、研究现状、挑战和展望.

日志结构文件系统在提供良好I/O性能的同时也被文件系统碎片化[4−5]问题所困扰. 随着频繁地创建、更新和删除文件，文件系统产生严重的碎片化问题，通常分为文件碎片和空闲空间碎片. 文件碎片是单个文件离散分布的数据片段，空闲空间碎片由未回收的无效空间和离散分布的空闲空间构成，日志结构文件系统的空闲空间碎片主要由未回收的无效空间组成. 文件碎片会分割I/O请求，使I/O请求变小，从而增加访问随机性，导致文件系统性能下降. 重整文件碎片可以减少碎片，但需要复制文件数据，并将其写到新地址，增加文件系统读写开销，缩短闪存寿命. 为减轻上述隐患，只在必要的时候才重整文件碎片. 空闲空间碎片导致新到来的文件碎片化地写入，造成文件写性能的直接下降和读性能的间接下降. 严重的空闲空间碎片加剧文件系统段清理开销[6−7]. 目前日志结构文件系统使用段清理(segment cleaning, SC)整理空闲空间碎片，然而段清理增加了读写开销和能量消耗. 针对上述问题，需要调整段清理的回收对象和频率，提高空闲空间碎片缩减的效率. 文件碎片和空闲空间碎片之间互相影响，更热的文件块更早失效导致空闲空间碎片增加，而空闲空间碎片导致新写入文件的文件碎片增加.

由于日志结构文件系统异地更新数据，需要回收失效的旧版本数据，其性能依赖于连续空闲空间，

极大地受限于段清理开销. 日志结构文件系统使用2种方法管理空闲空间: 线程(threading)写入和复制(copying)数据. 线程写入方法把文件系统中的无效块当作空闲块, 直接覆盖写. 复制数据方法先选择一个段作为清理对象, 识别该段中的有效块, 并将有效块复制到新段, 回收清理对象作为空闲段用于后续数据写入, 即段清理方法. 通常, 日志结构文件系统使用段清理方法管理空闲空间, 回收离散的无效块, 为接下来的日志结构写数据提供空闲段. 日志结构文件系统中有2种段清理方式, 前台段清理(foreground SC, FSC)和后台段清理(background SC, BSC), 只有当写请求没有足够空闲空间时才触发前台段清理, 文件系统定期唤醒内核线程触发后台段清理. 在选择清理对象时, 前台段清理使用贪心算法, 后台段清理使用成本-收益(cost-benefit, CB)算法. 触发段清理时会阻塞日志结构文件系统的正常读写请求, 增加I/O响应延迟. 在保证文件系统性能良好的前提下, 优化段清理策略, 提高段清理减少空闲空间碎片的效率非常重要.

本文主要介绍了移动设备日志结构文件系统的碎片化问题和段清理问题. 首先阐述了文件碎片和空闲空间碎片产生的原因与影响, 详细分析了现有的减少碎片的研究工作. 随后说明了冷热数据区分与段清理的关系, 总结了区分冷热数据和减少段清理的研究现状. 主要贡献包括3个方面:

1)介绍了移动设备日志结构文件系统的相关概念、性能问题和能耗问题;

2)详细分析了移动设备日志结构文件系统的碎片化和段清理问题;

3)总结了移动设备日志结构文件系统研究的主要挑战和未来研究工作.

1 移动设备存储系统

在本节中, 我们主要介绍移动设备的概念、移动设备存储系统的分层结构和相关内容, 并简要阐述移动设备存储系统的性能问题和能耗问题.

1.1 移动设备的概念

随着移动设备的普遍使用, 智能手机、平板、穿戴设备和使用安卓系统或苹果公司CarPlay的智能汽车发展很快. 智能手机的更新换代非常快, 用户体量也非常大. 2022年全球智能手机移动网络用户数量接近64亿, 预计到2028年将超过77亿; 平板电脑或者简称“平板”, 是一种小型的、可携带式的个人电脑移动设备, 允许用户通过手指头或触控笔来进行活动. 目前, 以教育类为代表的行业用平板和可拆分平板电脑的需求较大; 智能可穿戴设备是指可直接穿戴在身上或整合到衣服、配件中, 且可以通过软件支持和云端进行数据交互的设备, 当前可穿戴终端多以手机辅助设备出现, 以智能手环、智能手表和智能眼镜最为常见; 智能汽车集环境感知、规划决策等多种辅助驾驶技术于一体, 运用计算机、现代传感、通信、人工智能以及自动控制等多种高新技术. 目前, 多家品牌汽车的车载系统基于安卓或Linux内核开发, 车企或科技公司致力于建设智能汽车操作系统生态.

移动设备存储系统的研究现状与发展瓶颈备受开发人员和终端用户的关注. 移动设备存储和计算资源有限, 而移动设备与用户的高互动性严格要求低延迟的性能. 研发人员需要深入钻研移动设备存储系统的工作机制, 使设备快速响应, 保证良好的用户使用体验. 对移动设备性能的改善因为实用性和商用的经济效益能很快落实到实际设备上, 为普通用户的日常生活带来便利.

1.2 移动设备存储系统架构

移动设备存储资源有限, 与用户的高度互动特性要求低时延低能耗, 本节将分析讨论移动设备存储系统架构.

移动设备存储系统分层结构如图1所示, 在这个分层结构的顶部, 用户空间进程通过系统调用访问文件, 如打开、读取、写入等. SQLite数据库是安卓系统默认的关系型数据库, 大量应用如系统邮件和脸书等, 使用SQLite持久地管理文件数据, SQLite的插入、删除、更新操作速度影响系统响应时间. C标准库(C standard library, libc)是最基本的C语言函数库, 提供C语言中使用的宏、类型的定义、字符串操作符、数学计算函数以及输入输出函数等. 作为具体文件系统之上的抽象层, 虚拟文件系统(virtual file system, VFS)将文件操作命令发送到实际文件系统. 常用的闪存文件系统还有日志闪存文件系统JFFS2(journaling flash file system version 2)[8]和无序区块映像文件系统(unsorted block image file system, UBIFS)[9].UBIFS由诺基亚公司和匈牙利塞格德大学共同研发, 使用UBI层管理闪存坏块, 通过内存技术设备(memory technology device, MTD)访问闪存芯片, 解决了JFFS存在的内存消耗大、可扩展性差、损耗均衡能力差等问题. 不同于UBIFS, F2FS不直接面向裸NAND闪存, 而是和其他通用文件系统一样面向

块设备层接口，通过闪存转换层（flash translation layer, FTL）访问 NAND 闪存.

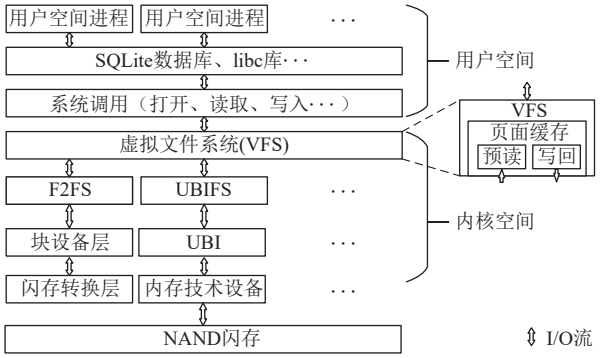


Fig. 1 Hierarchical structure of mobile device storage system

图 1 移动设备存储系统的分层结构

1.3 移动设备存储系统的性能问题

大量工作研究智能手机存储系统的瓶颈[10−11]，研究移动存储系统 I/O 特点[12] 与应用行为特征[13−14]，深入理解移动存储系统的能耗特征[15]，保证移动存储系统的快速响应[16−17] 以及解决能耗问题[18−19].

Yang 等人[20] 提出 log-on-log 问题，虽然移动应用、文件系统和 NAND 闪存的数据都是异地更新的，但因为工作负载的随机性、未对齐的段大小和不协调的多级日志垃圾回收（garbage collection, GC），反而使读写效果更差. 汉阳大学的 Jeong 等人[21] 提出 Androstep 分析安卓移动设备存储系统的性能，Androstep 包括工作负载生成器 MobiBench 和工作负载分析器 MOST（mobile storage analyzer）. Hahn 等人[22] 发现页面缓存和闪存存储设备存在 I/O 优先级倒置的问题，但现有减少 I/O 优先级倒置的技术不适用于智能手机，提出一种前台应用感知的 I/O 管理方案，通过抢占整个 I/O 栈中的后台请求以及防止前台应用的数据被从页面缓存中刷新（flushed）来加速前台请求. Han 等人[23] 观察到查找移动设备目录项时一部分前缀是一样的，通过动态跳过常见路径前缀来优化性能. Liang 等人[24−25] 发现安卓手机的内存管理问题，如回收空间太大、回收范围有限等，通过调整回收页面的内核线程 kswapd，在回收大小和整体性能间均衡. Mao 等人[26−27] 基于不同应用间的重复数据比较少的特点提出一种应用感知的数据去重技术. Ji 等人[28] 基于移动应用的文件访问模式提出双模式压缩技术以减少总写入流量. Ren 等人[29] 在页缓存中采用原子性事务机制，有效降低移动应用的响应时间和能耗.

总之，移动设备的存储和计算资源有限，终端用户对时间高度敏感，移动设备存储系统的性能追求低时延和稳定性，以保证用户流畅的使用体验.

1.4 移动设备存储系统的能耗问题

移动设备配置的电池容量有限，导致智能手机 1 天内多次充电，这非常影响用户使用体验. 智能手机的电池续航问题一直是行业痛点，移动设备各个组件活动的能耗需要尽量降低. 对于 I/O 密集型负载，移动设备存储系统的能耗与屏幕显示、网络连接的能耗相当，约占 30%[30]. 现有移动设备的能耗研究主要关注构建移动设备能耗模型、寻找能耗瓶颈和降低能耗的解决方案，缺乏面向日志结构文件系统的能耗理解和优化其能耗的策略.

移动设备日志结构文件系统优化旨在提高吞吐量、缩短响应时延、降低能耗. 移动消费电子设备由电池供电，但电池的尺寸和容量有限，管理好能耗至关重要，现有的能耗研究工作主要从理解能耗和减少能耗 2 个角度展开研究.

一些研究工作分析移动设备各组件存储活动的能耗，搭建模拟器评估系统的能耗. Mohan 等人[30] 基于差分分析的实验结果表明由随机 I/O 主导的工作负载，智能手机存储子系统会消耗大量能量（36%），与屏幕能耗相当，比网络能耗多. 他们还发现文件系统中随机 I/O 比顺序 I/O 消耗更多的能量，对于大多数工作负载，F2FS 只消耗 ext4 一半的能量. Carroll 等人[31] 分析了手机 Openmoko Neo Freerunner 的电量消耗，开发了 Freerunner 的功率模型，发现闪存芯片本身的能耗较低，但执行闪存管理层的组件，如 CPU 和 RAM 的能耗是不可忽略的. Li 等人[15] 分析了移动设备存储硬件和软件所消耗的能量，建立了一个存储能量模型（energy modeling for storage, EMOS）来估计存储活动所需能量. 一组存储密集型微基准测试显示，在安卓手机和 Windows RT 平板电脑上，存储软件比存储硬件多消耗 200 倍的能量. Yoon 等人[32] 提出了安卓能量衡量系统 AppScope，通过监视硬件组件请求的内核活动，提供关于应用能耗准确而详细的信息. Olivier 等人[33] 提出了一种 3 阶段方法来估计闪存存储系统应用 I/O 的性能和能耗，在探索阶段识别了影响存储系统性能和能耗的主要因素，在建模阶段用方程和算法对主要影响因素构建模型，最后一个阶段提出名为 OpenFlash 的模拟器并实现了原型.

一些研究工作通过钻研应用行为特征和 I/O 模式参数，提出降低能耗的方案. 由于后台应用不断消耗智能手机的电池电量，如何平衡应用启动延迟和电池寿命成为问题，Chung 等人[34] 提出了一种应用管

理框架(application management framework)终止不用的后台应用以节省能量,并预启动有益的应用以减少应用启动延迟.Nguyen 等人[35]研究缓存、设备驱动和块层中的存储参数如何影响智能手机能耗,设计并实现了 SmartStorage 系统跟踪智能手机运行时的 I/O 模式,将当前 I/O 模式与从 8 个基准中记录的已知模式匹配最优参数,如调度算法和队列深度,然后动态配置存储参数以降低能耗. Huang 等人[36]利用可穿戴设备上蓝牙和 Wi-Fi 混合的低能耗网络连接,通过使用电池供电的 RAM,设计可穿戴设备快速存储系统 WearDrive,将数据和计算从可穿戴设备传送到手机上,以低能耗成本在手机上执行大型高能耗任务,同时使用电池供电的 RAM 执行小型高能效任务以减少可穿戴设备的能耗.

一些研究工作使用新型高能效存储器件来降低能耗. Zhong 等人[37]提出 DR.Swap,采用节能非易失存储器作为交换区,利用 NVM 的字节寻址能力,允许读请求从交换区直接读取,保证只读页的零拷贝,降低智能手机能耗.Yan 等人[38]观察到智能手机应用中超过 40% 的 2 级(level 2, L2)缓存访问是操作系统的内核访问,频繁的内核访问导致 L2 缓存中用户块与内核块严重干扰,提出将 L2 缓存划分为 2 个分别只能由用户代码和内核代码访问的单独小段,同时他们发现用户段与内核段的访问行为完全不同,提出在用户段使用短保留(short-retention)自旋转移扭矩随机存取器(spin-transfer torque random access memory, STT-RAM),在内核段使用长保留 STT-RAM 的多保留 STT-RAM 使用方案,以节省缓存能耗.

综上所述,现有工作发掘移动设备存储系统能耗的模型和特点,针对能耗问题从基于读写特征和利用新型器件降低能耗展开了研究. 我们需要进一步了解日志结构文件系统的存储软件活动能耗特征,理解文件系统碎片化和段清理对能耗的影响,提出减少能耗、提高能效的有效策略.

移动设备存储系统性能与能耗优化有 2 类研究方向:第 1 类研究针对存储系统实际问题,基于应用行为特征和 I/O 访问模式解决问题[22−28];第 2 类研究在分析应用行为特征和 I/O 访问模式的基础上,还基于软硬件技术发展,采用新型存储协议或存储器件解决存储系统实际问题[36−38]. 第 1 类研究针对实际问题,需要考虑解决方案的可移植性和普适性;第 2 类研究因为在当前的移动设备中没有实际的商用产品,大多数研究是基于模拟平台展开的,目前难以实现实际生产以及推广应用.

2 移动设备日志结构文件系统

本节首先介绍日志结构文件系统的概念,然后重点介绍移动设备日志结构文件系统的碎片化问题与段清理问题. 日志结构文件系统主要存在碎片化和段清理开销大的问题,此外还有 *fsync*() 性能问题[39−42]和 *trim* 问题[43−44],这 2 个问题与日志结构文件系统的段清理问题相关.

2.1 日志结构文件系统

20 世纪 90 年代初 Rosenblum 等人[2]提出日志结构文件系统的设计与实现,随着 NAND 闪存存储器的成本降低,日志结构文件系统的使用变得广泛. 日志结构文件系统以日志(log)方式追加写,使用段清理回收无效空间.NAND 闪存采用异地更新,以闪存页为单位进行写入,以闪存块为单位进行擦除.F2FS 专为闪存特性设计,采用日志结构写方式,以文件系统层的逻辑块为单位进行读写,以逻辑节为单位进行擦除. 基于 F2FS 的数据结构设计, Lee 等人[1]的实验结果表明 F2FS 的 I/O 性能比 ext4 的更好,合成负载 iozone 的性能优于 ext4 高达 3.1 倍, Mohan 等人[30]的实验结果表明实际安卓应用中 F2FS 比 ext4 少消耗 2 倍的能耗.

日志结构文件系统技术在不断改善, Seltzer 等人[45]分析了 Unix 快速文件系统和 LFS 的性能,发现当元数据操作成为瓶颈时,相比于 Unix 快速文件系统, LFS 的性能更好,当创建 1 KB 及更小的文件或删除 64 KB 及更小的文件时, LFS 提供了 1 个数量级的性能改进. Cheng 等人[46]提出一个基于双模相变存储器(phase change memory, PCM)的日志结构文件系统 DPLFS,在保证数据一致性的同时提高基于 PCM 的文件系统吞吐量. 非易失存储器加速(nonvolatile memory accelerated, NOVA[47])的日志结构文件系统是一个混合易失和非易失内存的文件系统,其采用传统的日志结构文件系统技术充分发挥非易失存储器(non-volatile memory, NVM)的快速随机访问特性,为每个文件提供单独的日志,实现高性能的同时保证一致性. Xu 等人[48]进一步通过合并快照、校验码等技术提高 NOVA 的容错性和可靠性,提出 NOVA-Fortis 兼顾容错性和高性能. Liao 等人[49]提出一个多核友好的日志结构文件系统 MAX,使用读者直通锁实现并发控制,引入文件单元来扩展对内存索引和缓存的访问,采用多个日志分区实现并发空间分配,解决文件系统内部的扩展性瓶颈并提升 CPU 并行性.

F2FS 是典型的日志结构文件系统，其文件数据组织布局如图 2 所示，划分为区（zone）、节（section）、段（segment）和逻辑块（logical block），以方便用户在格式化文件系统时将这些单元与闪存芯片的晶圆（die）、分组（plane）、闪存块（flash block）和闪存页（flash page）等物理单元对齐. 逻辑块是最基本的读写单位，大小为 4 KB，512 个块组成 1 个段，大小为 2 MB，1 个或多个段构成 1 个节，节是 F2FS 段清理的单位，F2FS 一般设置 1 个节由 1 个段组成，1 个或者多个节组成 1 个区，如此进行数据组织布局是为了尽量与闪存操作单元同步.F2FS 在逻辑上包含 6 个部分，分别是超级块（super block，SB）、检查点（check point pack，CP）、段信息表（segment information table，SIT）、段摘要区（segment summary area，SSA）、节点地址表（node address table，NAT）以及主区（main area）.2 个超级块存放文件系统静态元数据，互为备份，检查点记录文件系统动态元数据，用于保证文件系统一致性. 段信息表记录文件系统段中逻辑块使用状态，段摘要区记录段中逻辑块所属状态，这 2 部分信息用于段清理，节点地址表用于缓解日志结构写方式的数据更新引入的元数据写放大问题. 主区由 4 KB 大小的逻辑块组成，用于存储数据（文件数据或目录）和节点（inode 或数据块索引），主区的数据占整个系统的绝大多数. 除主区采用日志结构写入的异地更新方式以外，其他与文件系统元数据相关的区域采用就地更新（in-place update，IPU）方式.

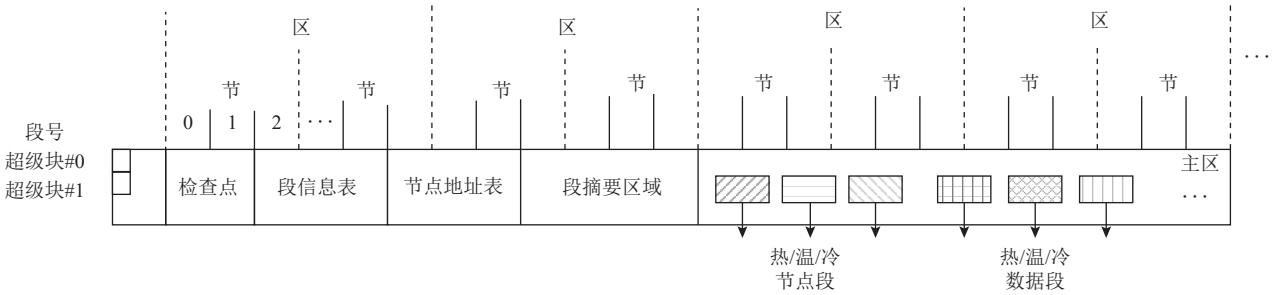


Fig. 2 File layout of F2FS

图 2 F2FS 的文件布局

分离冷热数据能实现有效数据的双峰分布[6]，更新频率较低的冷数据不容易失效，更新频率较高的热数据更容易失效. 更新后旧版本热数据失效，所在段迁移成本低，有较大的可能被选为清理对象，能减少段清理迁移冷数据造成的写放大. 主区文件块的热度根据文件系统语义，如数据功能（索引数据和文件数据），划分为 6 个大类，分别是热、温、冷的节点和相应的数据块. 主区数据热度分类如图 3 所示，这 6 类文件块在文件系统中分别从 6 个日志写入，根据系统语义，F2FS 将用户写常规文件产生的大量数据块分类为温数据，导致温数据的数据量占比最大.

自 2012 年 12 月 20 日在 Linux 内核 3.8 版本中集成 F2FS 之后，F2FS 的性能优化研究受到广泛关注. 文件碎片增加 I/O 请求数量，造成文件系统老化，导致设备响应用户请求变慢，针对该问题需要在不影响设备寿命的前提下减少文件碎片. 日志结构文件系统性能取决于连续的空闲空间，连续空闲空间不足时需要执行段清理来获得空闲段，因此在保证文件系统正常工作的前提下减少段清理次数和开销成为关键. 当空闲空间碎片数量比较多时，为回收无效空间 F2FS 会频繁触发段清理，需要有效减少空闲空间碎片以减少段清理开销.

2.2 日志结构文件系统碎片化问题

2.2.1 文件碎片与空闲空间碎片

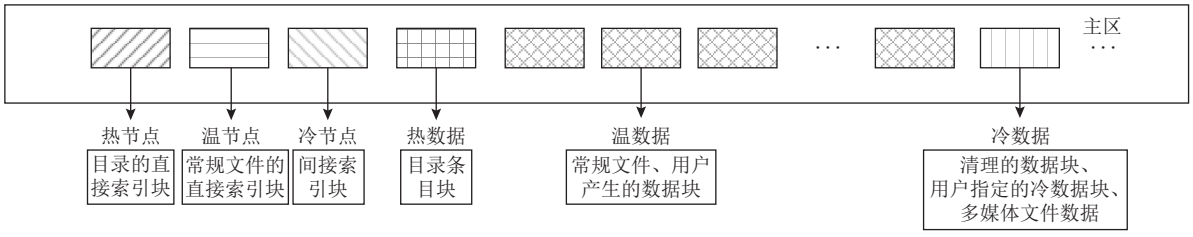


Fig. 3 Hot data and cold data classification of F2FS main area

图 3 F2FS 主区的冷热数据分类

文件碎片和空闲空间碎片造成文件系统老化, 是文件系统性能提升的瓶颈. 日志结构文件系统的异地更新机制更是加重了文件碎片化和空闲空间碎片化程度, 加剧了日志结构文件系统的段清理开销.

如图 4 所示, 在一个初始化的文件系统上, 文件 A, B 的数据连续存放, 随后创建文件 D, 更新文件 A, 追加写文件 D, 更新文件 B, 删除文件 C. 此时文件系统中的数据分布变得离散, 碎片化的文件系统状态存在文件 A, B, D 的文件碎片, 还存在空闲空间碎片. 文件碎片是单个文件离散分布的数据片段, 文件碎片增加读取文件的 I/O 次数, 导致访问随机性增大, 直接造成读性能下降. 空闲空间碎片由未回收的无效空间和离散分布的空闲空间构成, 由于 LFS 日志结构追加写, 其空闲空间碎片主要由无效数据块组成, 离散分布的空闲空间存在于日志结构文件系统所写 6 个日志头的位置.

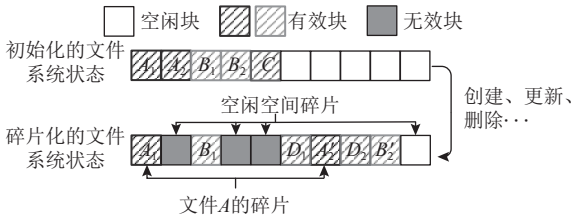


Fig. 4 Generation of file fragmentation and free space fragmentation in log-structured file systems

图 4 日志结构文件系统中文件碎片和空闲空间碎片的产生

当空闲空间不足时触发日志结构文件系统的段清理整理空闲空间碎片获得整个空闲段, 此时存在大量空闲空间碎片. F2FS 也支持以线程日志(threaded logging)的方式写“洞”(hole), 不同于追加日志写方式, 线程日志写方式不会触发段清理, 但会产生随机写, 导致写性能下降, 进而造成新的文件碎片, 降低顺序读性能. 日志结构文件系统一般通过日志以追加写形式在新地址写新数据, 这样异地更新数据导致文件碎片增加以及旧版本数据失效造成空闲空间碎片增多. 文件碎片和空闲空间碎片二者之间互相影响, 文件更新后, 存放该文件数据的文件碎片可能变成存放无效数据的空闲空间碎片, 线程日志写空闲空间碎片可能导致新的文件碎片. 此外, 如图 5 所示, 移动应用大量随机同步小写请求的 I/O 模式也会加剧碎片化. 一方面, 一些应用如 Facebook 会产生 18 个并发线程写数据到它们的数据库文件[50]; 另一方面, 一些应用会并发写入多个数据库文件, 并要求同步数据.

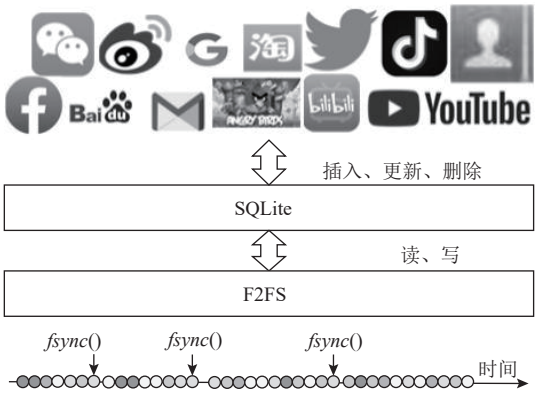


Fig. 5 F2FS multi-threaded writing of multiple files concurrently

图 5 F2FS 多线程并发写多个文件

2.2.2 冷热数据混合对碎片的影响

数据热度描述数据更新或被访问的频繁程度, 不同热度的数据混合存储和更热的数据更早失效导致空闲空间碎片增加. 由于实际 I/O 访问数据的时间和空间局部性, 数据读写间的热度倾斜普遍存在且显著. 日志结构文件系统异地更新数据的同时加剧文件碎片和空闲空间碎片化, 不同热度的数据混合存储加重了空闲空间碎片化程度.

如图 6 所示, 假设 1 行表示 1 个段, 文件 A 的数据热度大于文件 B 的数据热度, 文件 B 的数据热度大于文件 C 的数据热度. 在日志结构文件系统中执行以下操作: 写 A_1, B_1B_2, C_1, 追加写 A_2, B_3, C_2C_3, 异地更新 B_3(旧版 B_3 失效), 追加写 A_3A_4, C_4, 异地更新 $A_1A_2A_3A_4$.日志结构文件系统定期触发后台段清理, 基于收益最大化原则, 后台段清理倾向于选择有效块数量少且年龄大的段作为清理对象. 在图 6(a)中, 后台段清理选择第 2 个段进行清理, 将该段的有效块 C_2, C_3 复制到新段, 第 2 个段变成整段连续的空闲空间. 连续的空闲空间碎片视为 1 个空闲空间碎片, 整段连续的空闲空间是 1 个空闲段. 按照传统 F2FS 的热度分类规则, 文件 A, B, C 的数据被分类为温数据, 因此写入同一个段. 如图 6(a)所示, 文件系统中产生 3 个空闲空间碎片. 如图 6(c)所示, 在细粒度区分不同热度的数据热度划分规则中, 文件 A, B, C 的数据热度不同, 依据各自热度分别写入不同的段, 只产生 1 个空闲空间碎片, 所以不同热度的数据混合存储增加了空闲空间碎片数量. 图 6 也说明了日志结构文件系统异地更新数据导致文件碎片数量增加.

减少文件碎片和空闲空间碎片对提升文件系统性能非常重要, 如图 7 所示, 去碎片化包括减少不同文件的碎片和文件系统的空闲空间碎片数量. 图 7 所

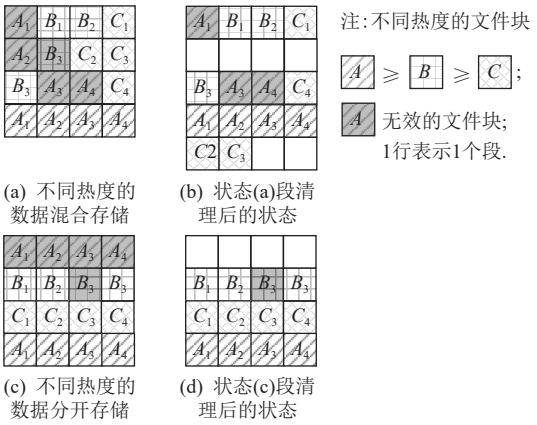


Fig. 6 Increase of free space debris caused by the mixed storage of data with different hotness

图 6 不同热度的数据混合存储导致空闲空间碎片增加

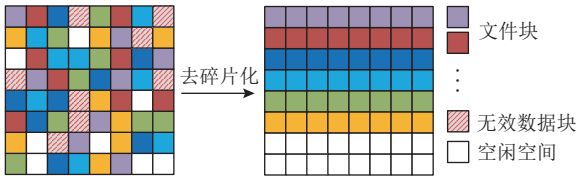


Fig. 7 Illustration of file system defragmentation

图 7 文件系统去碎片化示意图

示的去碎片化过程需要复制碎片文件数据，重整到新的连续地址以有效减少文件碎片，日志结构文件系统依赖段清理整理空闲空间碎片并回收无效空间. 目前，研究人员关注碎片化问题，主要是通过减少碎片来提升文件系统性能，很少关注通过减少碎片来减少能耗或提升能效. 接下来，我们总结日志结构文件系统碎片化问题在性能方面的研究工作.

2.2.3 碎片的产生与影响

基于不同老化工具的碎片的产生与影响的研究工作如表 2 所示，现有研究工作使用不同方法制作老化工具，通过复现各种文件系统的碎片，验证文件系统因为碎片而老化，研究碎片的产生原因，量化碎片对性能的影响，发现碎片快速产生、显著降低文件系统性能.

老化工具分为 3 类：合成负载生成器、脚本执行实际应用和 trace 重播，Kadekodi 等人[5]提出的 Geriatrix 属于合成负载生成器，Smith 等人[51]提出的老化工具以及 Liang 等人[52]使用的 Iozone 也是合成负载生成器，Ji 等人[50]提出的 AutoAge 和 Conway 等人[4]提出的 Git-Benchmark 属于脚本执行实际应用的老化工具，Jeong 等人[12]使用 MobiBench 抓取应用 trace 并重播，他们提出的 MobiBench 属于 trace 重播这一类.

Kadekodi 等人[5]提出一个配置文件驱动的文件

Table 2 Research Status of Generation and Impact of File System Fragmentation

表 2 文件系统碎片产生与影响的研究现状

老化工具类型	工具名称	主要特点
合成负载生成器	Geriatrix	提出文件系统老化工具 Geriatrix[5]
	老化负载	利用文件系统快照产生碎片[51]
	Iozone	研究 F2FS 中文件碎片对性能的影响[52]
脚本执行实际应用	AutoAge	产生、衡量和减少文件碎片的方法[50, 53]
	Git-Benchmark	证实大多数文件系统都会老化[4]
trace 重播	MobiBench	使用 MobiBench 抓取应用 trace 并重播[12]

系统老化工具 Geriatrix，用于产生目标级别的文件碎片和空闲空间碎片，老化过程分为快速老化和平稳老化 2 个阶段. Geriatrix 使用系统分区大小、空间使用率、文件大小分布、目录深度分布、相对年龄分布等老化配置参数，提供一个包含 8 个文件系统老化配置文件的存储库用于老化文件系统.

Smith 等人[51]利用文件系统快照收集信息，如 inode 号、文件类型、文件大小等，构造老化负载，通过重放与实际文件系统几个月甚至几年时间里所经历的工作负载相似的老化负载，测试文件系统产生碎片后的性能，并评估了 UNIX 快速文件系统用该方法生成的负载老化后的性能.

香港城市大学的 Liang 等人[52]评估基于移动设备的 F2FS 性能，使用 Iozone 模拟测试文件碎片对性能的影响，发现随着写操作数量的增加，F2FS 逻辑层碎片化变严重，影响预读命中率和 I/O 调度的合并操作，顺序读性能随着碎片数量的增加而降低.

Ji 等人[53]在实际移动平台上用碎片化程度(degree of fragmentation, DoF)量化了 ext4 碎片化严重程度，发现碎片导致频繁的块 I/O 请求和离散的块 I/O 模式，导致移动设备文件系统性能下降，他们还发现数据库文件是碎片化程度最严重的文件之一. Ji 等人[50]又深入研究了移动设备上产生、衡量和减少 ext4 文件碎片的方法，设计了基于脚本的文件系统老化工具 AutoAge，证实了文件碎片造成用户可察觉的延迟，并评估了现有减少文件碎片的方法.

Conway 等人[4]设计了 Git-Benchmark，通过使用 Git 连续同步 Linux 内核源代码和一个邮件服务器负载来模拟文件系统老化，用扫描时延和布局得分(layout score)来衡量文件系统老化程度. 他们发现老化的发展速度很快，导致文件系统性能大幅度下降，通过分析发现无论是基于机械硬盘(hard disk drive, HDD)，还是固态硬盘，大部分文件系统都会积累碎

片, 导致其性能下降, 如广泛使用的 ext4[54], XFS[55], BtrFS[56], ZFS[57], F2FS 都会老化. 而 Jannen 等人[58−59] 提出的 BetrFS 在实际负载测试中能有效缓解文件系统老化, Conway 等人[60] 还发现文件系统空间写满程度(fullness)与负载使用(usage)对文件系统性能的影响程度不一样, 与文件系统空间写满程度相比, 文件系统性能与负载使用、碎片的产生关系更密切, 并建议关注直接影响写性能、间接影响读性能的空闲空间碎片.

Jeong 等人[12] 提出 MobiBench 生成安卓系统工作负载, 并使用 MobiGen 记录和回放给定应用的用户行为生成的系统调用 trace. 通过回放系统调用 trace, MobiGen 可以在没有实际人为干预的情况下重现人工驱动的 I/O 活动. Jeong 等人配合使用 MobiBench 和 MobiGen 老化文件系统.

现有研究工作深入挖掘碎片的产生原因以及验证碎片对性能的影响, 之前的研究工作因为 HDD 磁盘寻道机械臂旋转的时间开销, 人们很容易接受 HDD 文件系统碎片造成性能下降, 但对于 SSD 文件系统, 人们认为 SSD 随机读写性能优于 HDD, 碎片的影响不是很大, 而去碎片化需要复制碎片数据, 考虑到 SSD 的写前擦除特性, 增加的读写操作会缩短 SSD 寿命. 但事实并非如此, 表 2 中现有研究工作表明 SSD 文件系统中碎片会快速积累, 造成文件系统反应缓慢, 导致用户体验下降. 如图 8 所示, bio 表示通用块(block)层的 1 个 I/O 结构, req 表示一个请求(request)结构, cmd 表示发送给闪存存储器的 I/O 命令(command). 文件 A 是碎片化的, 文件 B 是连续的, 进程 1 和进程 2 分别顺序读文件 A 和 B 的 4 个文件块. 同样顺序读 4 个文件块, 进程 1 需要 4 个 I/O 请求, 进程 2 仅需要 1 个 I/O 请求. 这是因为文件 A 的 4 个文件碎片在文件系统层分割 I/O 请求, 导致在块层和存储层需要多个 I/O 请求读文件 A 的数据. 文件碎片增加 I/O 次数, 使 I/O 请求变小, 增加访问随机性. 移动设备的计算存储资源和电池电量都有限, 用户要

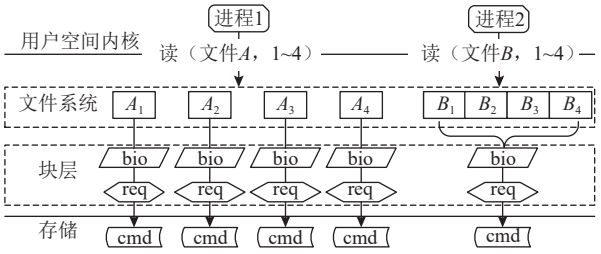


Fig. 8 Illustration of file fragmentation I/O requests[61]

图 8 文件碎片分割 I/O 请求示意图[61]

求和移动设备高度交互, 对应用响应延迟敏感. 碎片造成的设备卡顿直接导致用户不良体验, 碎片的负面影响不容轻视. 碎片产生与文件系统空间使用率、数据组织布局和应用 I/O 行为特征有关. 碎片通过影响创建、插入、合并、排序、调度等操作, 导致完成应用读写请求的 I/O 次数增加, 读写请求地址的寻找次数增加, 时间开销增大. 此外文件碎片的影响还取决于底层存储介质[62], SSD 碎片影响其内部并行程度, 碎片化程度增加导致读并行程度下降[63]. 空闲空间碎片导致新到来文件碎片化写入, 直接影响其写性能, 造成新的文件碎片, 间接影响该新写文件的读性能.

2.2.4 减少碎片的研究

缓解文件碎片化的方法分为 2 类: 一类是在文件碎片产生前预防碎片产生, 如基于文件大小增长的预分配空间大小调整方法[64]、延迟分配方法[54]. 这类方法的难点在于如何精准确定对象, 否则会延长文件系统响应时间, 浪费存储空间. 另一类是在文件碎片发生后重整碎片, 如复制碎片化的数据并重新整理到新地址[65]. 这种方法需要寻找关联性, 将离散的单个小文件或文件组的数据重新组织到一起, 此过程需要复制数据, 而复制数据会增加读写数据量, 缩短存储器件寿命.

针对 F2FS 中文件碎片造成的顺序读性能下降, Li 等人[66] 提出多级阈值同步写方法. 多级阈值同步写方法包括碎片化状态判断模块和写入模式选择模块, 前者根据无效数据块数量获得碎片化程度, 后者根据碎片化程度设置就地更新阈值. 当同步写请求大小超过阈值时使用追加写, 否则就地写入以改善顺序读性能.

F2FS 提供 5 种局部数据的就地更新策略, 分别是 F2FS_IP_FORCE、F2FS_IPU_SSR、F2FS_IPU_UTIL、F2FS_IPU_SSRUTIL 和 F2FS_IPU_FSYNC 这 5 个策略, 使用 ipu_policy 参数选择其中的 1 种就地更新策略. F2FS 默认使用 F2FS_IPU_FSYNC 策略, 当一个同步请求的数据大小小于指定阈值时就地更新.

Yang 等人[67] 提出一种自适应预留空间(adaptive reserved space, ARS)策略, 分析典型移动应用的行为特征, 选择与碎片产生强相关的文件特征构造数据集, 采用低成本、高准确率的机器学习算法在大量文件中精准选择会严重碎片化的文件, 为这些文件预留空间, 有效减少文件碎片的产生. 为了最大化空间预留减少碎片的效果, Yang 等人[68] 还提出 ARST, 追溯文件描述符的写文件对象, 选择与时间无关的碎片写特征, 挖掘历史信息少的潜在碎片文件作为预

留文件，以进一步减少文件碎片，缩短应用响应延迟.

Ahn 等人[69]提出一个去碎片化的文件系统(defragmented file system, DFS)，通过动态重新定位聚拢内存中缓存的碎片文件数据块，也聚拢同一目录下的关联小文件，使得小的碎片文件连续存储在磁盘上，以减少文件碎片和提高小文件读性能.

为了在段清理过程中减少文件碎片，Park 等人[70]在内存中设计一个有效块队列(valid block queue, VBQueue)，在迁移要清理的段上的有效数据时将所有有效块复制到 VBQueue，然后根据 inode 号重新排序有效块，使得属于同一个文件的有效数据被迁移到新段时连续. 该方法在 SD 卡上更为有效，而在 SSD 上的效果不佳.

Park 等人[71]在不增加 I/O 开销的前提下提出一个抗老化的日志结构文件系统(anti-aging log-structured file system, AALFS)，AALFS 在段清理时基于有效块的 inode 号和文件偏移量排序以减少文件碎片. Park 等人[61]继续提出一个不限文件系统的去碎片化工具 FragPicker，他们发现不同于机械硬盘，碎片导致现代存储设备性能下降的原因是请求分割，即单个 I/O 请求被分割成多个请求. FragPicker 分析应用的 I/O 活动，使用不受限于特定文件系统的函数且没有修改内核，只重整对 I/O 性能至关重要的碎片数据，以实现与完全重整每个碎片文件的传统去碎片化工具相媲美的性能改进水平，最小化因整理碎片引入的 I/O 数量.

Hahn 等人[63]提出一个面向移动设备 ext4 文件系统的碎片整理工具 janusd，包括分别整理逻辑碎片和物理碎片的 janusdL 和 janusdP.文件系统层逻辑碎片增加了系统软件堆栈中的 I/O 开销，存储介质层物理碎片降低了闪存中的 I/O 并行性.janusdL 利用闪存存储内部的逻辑地址到物理地址的映射表实现无数据拷贝的逻辑碎片重整.janusdP 重整物理层的文件碎片时需要复制数据，只有在必要时才调用. 通过自适应地选择 janusdL 和 janusdP，基于定制 SSD 的实验结果表明，janusd 可以实现与 e4defrag 相同水平的 I/O 性能提升，而不会降低闪存使用寿命.

综上所述，表 3 总结了减少文件碎片的研究现状，移动设备和服务器的应用读写特征不一样，现有研究工作基于文件碎片如何影响 I/O 性能以及应用读写特征的分析，从预防碎片产生和重整碎片 2 个角度提出减少文件碎片的策略，但是目前减少碎片的研究工作还缺乏直接减少移动设备日志结构文件系统空闲空间碎片的优化技术.

Table 3 Strategies to Reduce File Fragmentation

表 3 减少文件碎片的策略

研究角度	文件系统	主要特点
碎片产生前，预防碎片产生	XFS	预分配空间大小调整方法[64]
	ext4	将块分配从写操作时间延迟到页刷新时间[54]
	F2FS	多级阈值同步写方法[66]
	F2FS	自适应选择严重碎片化文件的预留空间方法[67−68]
碎片产生后，重整碎片	DFS	通过动态地重新定位和聚拢数据块减少碎片[69]
	F2FS	段清理过程中基于 inode 号给有效块排序[70]
	不受限	只对非常影响 I/O 性能的部分数据重整碎片[61]
	ext4	在 FTL 实现无需复制数据的重整过程[63]

2.3 日志结构文件系统段清理问题

2.3.1 冷热数据混合对段清理的影响

数据越热，访问频率越高，就越容易异地更新，导致旧版本热数据变成无效. 当无效数据较多、空闲空间不足时，需要清理无效数据回收空间，而段清理成本高，不同热度的数据混合存储增加段清理开销.

如图 6 所示，相比于图 6(c)文件 A 更新后整段数据失效，无需迁移有效数据块就能获得图 6(d)第 1 行表示的空闲段，图 6(a)获得空闲段需要迁移 2 个有效块，段清理后得到如图 6(b)第 2 行表示的空闲段，并在第 5 行表示的新段上写 2 个数据块 C_2 和 C_3. 区分数据热度存储的图 6(c)段清理后到图 6(d)的数据分布无需迁移有效块，而不同热度的数据混合存储的图 6(a)段清理后到图 6(b)需要迁移 2 个有效数据块，这说明不同热度的数据混合存储加剧段清理开销.

与碎片化问题类似，现有研究工作比较少关注区分冷热数据和减少段清理对能耗的影响，其中文献 [72−73] 讨论了通过减少段清理减少能耗理论值. 研究人员主要投入到区分冷热数据和减少段清理带来的性能方面的提升以及 SSD 寿命的延长，接下来，我们总结了区分冷热数据和段清理问题性能方面的研究工作.

2.3.2 区分冷热数据的研究

区分数据热度并分开管理不同热度的数据，可以减少空闲空间碎片和段清理次数，减少不必要的读写操作，延长闪存使用寿命.

一些工作分离文件系统中的冷热数据以减少段清理开销.WOLF 设计启发式算法，按照频率分离活跃和不活跃数据，具有相同引用计数的块，按照同一个文件或目录对块进行分组[74]. Kang 等人[75]提出一种冷热数据分离方法，利用页缓存中隐藏信息动态识

别文件系统中块的热度,以有效减少数据复制操作.

Min 等人[6]基于 Linux 日志结构文件系统提出一个新的 SSD 文件系统(file system for SSDs, SFS),SFS 将所有文件系统级的随机写操作转换为 SSD 级的顺序写操作以获得最大带宽.他们统计块的写次数和频率,迭代量化所有段的热度,获得分组标准以分离冷热数据段,使得段使用率形成明显双峰分布以减少段清理开销,但该热度识别过程不能适应 I/O 模式的热度变化.

一些工作分离磁盘或 SSD 中冷热数据[76−77]以减少垃圾回收开销和缩短响应时间[78],对于区分数据热度和减少文件系统空闲空间碎片同样具有指导作用.Wang 等人[79]提出混合日志结构(hybrid log-structured, Hylog)磁盘布局,采用日志结构方式写热页提高写性能,采用覆盖方式写冷页降低清理成本.将 1 个页写入磁盘之前,HyLog 使用一个基于写代价模型的分离算法确定该页是否是热页.

Xie 等人[80]提出了一种自适应分离感知闪存转换层(adaptive separation-aware flash translation layer, ASA-FTL),该层使用采样实现轻量级分离标准识别,设计选择性缓存机制节省 SSD 中有限的 RAM 资源,使用数据聚类低成本准确识别和分离冷热数据,最终有效减少闪存的垃圾回收开销,提升闪存性能.

Li 等人[81]提出了一种基于热度感知机器学习(hotness-aware machine learning, HAML)的 SSD 管理方法 HAML-SSD,以减少 SSD 垃圾回收开销.他们根据更新频率和平均更新时间间隔定义热度,使用 2 维的 K 均值(K-means)聚类算法动态聚类热度相似的数据并存储.

Chiang 等人[82]提出动态数据聚类(dynamic data clustering, DAC)方法,使用访问频率动态分类数据,在数据更新和段清理期间进行聚类,以减少复制数据量和擦除操作数量.该方法不需要复杂计算来确定数据是热还是冷的,还设计了一个自适应清理管理器动态调整清理策略,响应数据访问行为的变化.

Kim 等人[83]提出 PCStream,使用 K 均值聚类算法根据支持的流的数量对具有相似数据生命周期的程序上下文进行分组,以分离冷热数据、减少多流 SSD 的垃圾回收开销.一个程序上下文表示一个程序的执行路径,能有效表示主要的 I/O 活动.

Yang 等人[84]发现 F2FS 中存在不同热度的数据混合存储,这些数据失效速率的差异化导致空闲空间碎片化程度增加,段清理开销增加.为此提出一种基于数据热度的多日志延迟写策略 M2H,使用 K 均值聚类算法,基于文件块更新距离感知数据热度变化,动态地准确区分数据热度,将数据分别写入相应热度的日志,使热度相近的数据汇聚在同一块区域,改善多文件间的数据组织布局.

区分数据热度的工作如表 4 所示,不同热度的数据混合存储加剧了空闲空间碎片化程度,分离热数据与冷数据进行数据重组可以减少清理开销.现有研究工作从准确定义热度、精准区分冷热数据和妥善放置冷热数据等方面展开研究,分别使用静态分类和动态分类方法分离冷热数据,分析发现现有区分文件系统数据热度的方法不够精准或无法随 I/O 模式变化动态地识别数据热度.

Table 4 Research on Distinguishing Data Hotness

表 4 区分数据热度的研究

区分方法	方案	主要贡献
静态分类	WOLF[74]	基于频率设计启发式算法分离活跃和不活跃数据
	hint[75]	利用页缓存中隐藏信息动态识别文件系统中块的热度
	SFS[6]	利用计数块的写次数和频率,获得分组标准分离冷热数据段
	HyLog[79]	基于写代价模型的分离算法区分冷热页
动态分类	ASA-FTL[80]	在 Flash 转换层使用数据聚类分离冷热数据
	HAML-SSD[81]	基于更新频率和平均更新时间间隔,用聚类算法区分热度
	DAC[82]	根据数据的访问频率对数据进行动态分类
	PCStream[83]	基于数据生命周期使用聚类算法对程序上下文进行分组
	M2H[84]	使用 K 均值聚类算法,基于文件块更新距离感知数据热度变化

2.3.3 减少段清理的研究

日志结构文件系统性能受限于段清理开销,Seltzer 等人[45]分析了日志结构文件系统的清理开销,在事务处理环境中,当磁盘写到 48% 时,清理开销导致日志结构文件系统性能降低 34% 以上.

段清理过程包括 3 步:选择清理对象、标识及迁移有效数据、等待检查点同步并回收被清理的段.现有研究工作通过优化脏段中有效数据的分布,优化段清理选择的对象、触发时机和频率,以减少段清理触发次数,减少段清理开销.

Zhang 等人[7]观察到写流量较大时,闪存设备的内部并行性没有得到充分利用,提出了一种并行性感知的文件系统 ParaFS.该系统基于定制 FTL 的闪存设备,利用闪存通道级并行性在维持冷热数据分离的同时实现 2 维数据分配,协调文件系统级和 FTL 级的垃圾回收过程,优化读写和擦除请求的调度,提高了写密集型负载的性能.

Gwak 等人[85]提出一种 GC 记录技术以减少日志结构文件系统的段清理开销，使用日志只记录与 GC 进程相关的文件系统修改，不用引入高成本的检查点操作保证文件系统一致性. Gwak 等人[86]进一步分析检查点开销，实现段清理日志（segment cleaning journaling, SCJ）技术，在一个特殊的日志区域记录段清理的块迁移信息，使得做段清理时不用触发检查点操作，避免引入不必要的写. 考虑到等待检查点释放的无效块数量比较多时影响文件系统性能，SCJ 在元数据更新较多和检查点时间间隔较久时做检查点操作.

Park 等人[87]观察到 Android 智能手机的挂起模式与触发后台段清理操作间存在冲突，提出挂起感知的段清理（suspend-aware SC）技术，在手机屏幕关闭后启动后台段清理，在手机达到实际挂起状态时停止后台段清理. 针对 F2FS 段清理影响性能的问题，Li 等人[66]提出高检测频率的后台段清理（high frequency BSC）方法，称为 MWHFB.该方法根据空闲空间大小动态调整后台段清理的频率，提高段清理性能，降低后台段清理对闪存寿命的影响. 与 suspend-aware SC 技术不同的是，MWHFB 可以根据空闲空间大小动态调整后台段清理频率.

Wu 等人[88]提出强化学习辅助的后台段清理（reinforcement learning assisted background segment cleaning, RLBC）方法，通过学习 I/O 工作负载行为和逻辑地址空间状态，基于强化学习自适应地决定何时触发后台段清理，达到平衡系统性能和存储器寿命的目的. 为了减少移动场景的内存开销，Wu 等人[72]进一步提出了基于剪枝的多目标深度强化学习后台段清理（deep multi-objective reinforcement learning-based background segment cleaning, MOBC）方法. MOBC 考虑移动场景中的所有相关元素，自适应地做出清理决策，减少了块迁移数量和段清理触发次数. 同时提出基于稀疏多径多层感知器的结构化剪枝方法进行稀疏化处理，减少了时间和空间开销.

Yang 等人[73]发现空闲空间碎片增加了系统任务的 I/O 次数，F2FS 使用段清理整理空闲空间碎片时，后台段清理能耗较大，且减少空闲空间碎片的效果有限. 为此提出空闲空间碎片感知的垃圾回收策略 FAGC，增加空闲空间碎片化严重的筛选条件以减少触发段清理.FAGC 基于数据分析重新评估了空闲空间碎片大小的阈值，使用空闲空间碎片系数快速衡量空闲空间碎片化程度，选择空闲空间碎片化严重的段作为清理对象，提高 GC 整理空闲空间碎片的能效.

此外，Yu[89]提出的最新 F2FS 垃圾回收策略 ATGC（age-threshold based garbage collection）被集成到 Linux 5.10 及以上版本的内核. ATGC 基于年龄阈值进行垃圾回收，根据定义的年龄阈值筛选较年长的清理对象，提高了文件系统垃圾回收的效率.

综上所述，日志结构文件系统优化段清理的工作如表 5 所示，现有工作从管理数据分布和调整段清理时机、频率和对象方面展开研究. 日志结构文件系统的性能受限于段清理开销，而移动设备的存储空间和计算资源有限，需要基于日志结构文件系统的数据分布和应用 I/O 行为，提出适当频率的、精准回收无效空间的低开销段清理方法.

Table 5 Optimizing Segment Cleaning Techniques for Log-structured File Systems

表 5 日志结构文件系统优化段清理技术

设计角度	方案	主要特点
清理发生前管理数据分布	ParaFS[7]	基于闪存通道并行性设计 2 维数据分配
	SCJ[85–86]	使用日志记录段清理时文件系统的修改
调整段清理时机、频率、对象	Suspend-aware SC[87]	挂起感知的后台段清理技术
	MWHFB[66]	高检测频率的后台段清理策略
	RLBC[88]	基于强化学习自适应决定何时触发后台段清理
	MOBC[72]	基于剪枝的多目标深度强化学习的后台段清理方法
	FAGC[73]	感知空闲空间碎片化程度的垃圾回收策略

3 未来展望

面向移动设备的日志结构文件系统，目前的研究面临 3 个主要挑战，现有成果还存在比较大的优化空间，可以从 4 个方面展开未来的工作.

3.1 主要挑战

不限于日志结构文件系统，当前移动设备的存储系统研究主要面临 3 个挑战：

1）不成熟的实验环境. 不同于技术发展很快而实际产品尚未面世的新型存储器件或其他先进设备的研究可以使用模拟器展开模拟实验，移动设备的存储系统研究需要落实到实际设备，获得实际应用场景的性能验证. 而当前移动存储研究的实验环境建设尚处于初级阶段，之前的研究常使用 Google Nexus9, Google Pixel 系列手机，因为我们能修改这些设备的内核源码并重新编译，验证所设计功能的实现. 其他的移动设备，如华为系列手机，因为源码的不开放和

编译的加解密过程,我们不能直接修改其内核源码,编译并实现相应的功能,除非有华为厂商的支持.而在过于老旧的安卓和内核版本做出优化的实际意义并不大.由于苹果设备的封闭性,我们也不会使用苹果公司的智能手机和平板.所幸我们看到国内的主流手机厂商,如华为、OPPO等,逐渐与部分高校成立联合实验室,相信在老师、学生们和公司研发人员的共同努力下,移动设备的实验环境能逐步完善.

2)有限的测试数据集.移动设备的软件(应用、系统等)和硬件(各类芯片)发展得非常快,导致在之前的老旧移动设备上收集的数据集在当代移动设备上不再可用.此外,由于终端用户的隐私受保护,导致我们能收集到的样本数量非常有限,现有研究缺乏公认的普适性强的数据集.该挑战需要能收集大量志愿者的数据集,以及在技术讨论社区不断维护、更新与开源.

3)碎片化的研究工作.移动设备发展20余年,对其的研究方兴未艾.移动设备在日常生活中发挥着重要角色,终端用户数量的不断增加和经济市场的扩大,促进了研发人员对移动设备存储系统研究的重视.然而,目前的研究比较碎片化,体现在各种各样的应用场景,存储系统不同组件如内存、文件系统等的各种性能和能耗问题,同一研究层次各种小的碎片化问题等.这使得移动设备存储系统的研究工作没有成体系,各个功能模块间的优化工作可能存在冗余或冲突,开展研究寻找对比工作时存在难度.该挑战需要移动设备存储系统的研究工作不断发展,不同方向的研究工作加强沟通,同一研究层次的问题基于之前的工作不断优化.

3.2 未来研究工作

面向移动设备的日志结构文件系统,未来的研究工作可以在4个方面展开:

1)增加能耗问题的研究.我们在调研时发现随着移动设备软硬件的升级,适配现代智能设备的能耗研究工作比较少.面向智能手机的电池,厂商们不断提高智能手机的充电效率,而优化移动设备的能效比和设备的发热问题的工作存在不足.可以从2个技术角度理解移动设备的能耗,分别是拆解整个设备的功能模块能耗以及理解单个功能模块的能耗和逐渐搭建整个移动设备的能耗模型.我们能够结合特定应用场景的I/O行为特征,基于最新的硬件技术升级和软件协议更新来提高能效比或减少能耗.

2)提高成效比.移动设备的存储空间、计算资源和电量有限,移动设备日志结构文件系统的研究需要充分考虑其成本,以及对应用、内核和存储硬件的修改幅度.因为移动设备更新升级快,当改动比较大时将导致不能向前兼容,改动带来的人力和物力投入比较大,而用户的接受程度比较低.我们在开展研究时要严格控制设计方案带来的移动设备内存空间和计算开销,控制对应用和存储系统元数据的改动幅度,注意不同版本的应用软件和存储系统的兼容性.例如,我们在设计上做一些扁平的层次化设计,不需要修改应用;在内核设计优化方案时格外控制对文件系统元数据结构的修改.

3)设计方案的可移植性和普适性.因为移动设备的品牌种类众多,各种移动设备的软硬件升级快,同一品牌的终端厂商1年内可能会发布不同系列的多部智能设备.面向当前的安卓系统生态,我们在日志结构文件系统展开的研究工作组织为可拆卸模块,减少对文件系统元数据的改动,能够单独编译和拆装,以提高研究工作的可移植性和普适性.例如,减少文件系统碎片化的预留空间方案设计为可拆卸模块、单独编译和拆装;控制文件系统段清理触发频率和时机的智能方法基于输入输出的数据接口,在不同设备的不同版本的日志结构文件系统都能适用.

4)软硬件垂直整合.随着移动设备存储硬件的不断发展和软件管理技术的快速优化,我们基于应用、数据库、文件系统和存储硬件的行为特征展开垂直整合优化,充分发挥不同层级的性能优势,不同层级的去碎片化、清理过程和日志技术互相协同,使得面向不断增长的用户数据量能够智能调整,基于较大的软硬件升级能够自我迭代优化.例如,文件系统层和FTL层的垃圾回收协同管理,FTL层暴露擦除接口给文件系统,文件系统基于自身读写事务情况来安排文件系统层和FTL层的垃圾回收对象和命令触发时机.

4 总 结

随着移动设备用户数据的爆炸性增长和硬件设备的快速升级,当前广泛运用于移动设备的日志结构文件系统的低时延、高能效技术面临巨大挑战.日志结构文件系统由于自身的异地更新和段清理机制,面临着文件碎片造成的系统卡顿、空闲空间碎片增加段清理频率、段清理增加读写开销等严重问题.本文对移动设备日志结构文件系统的研究现状进行了综述.首先介绍了研究背景,然后围绕文件碎片和空闲空间碎片阐述了碎片的产生与影响,详细分析了

减少碎片的研究工作. 具体包括在碎片产生前预防碎片产生和在碎片产生后重整数据 2 类，两者都需要精准地确定会严重碎片化以致严重影响文件系统性能的关键数据，使得以足够小的存储空间和读写开销实现去碎片化. 接着分析冷热数据混合与段清理的关系，详细讨论了区分冷热数据和减少段清理的研究工作. 区分冷热数据的方法可以分为静态分类和动态分类，减少段清理开销的方法可以分为段清理前管理数据分布以及调整段清理时机、频率和选择对象 2 类，好的解决方法需要能够感知 I/O 模式变化，不影响甚至能改善文件系统的前台 I/O 服务质量. 最后总结开展移动设备日志结构文件系统研究的主要挑战和未来研究工作.

作者贡献声明：杨梨花调研并撰写论文；董勇和邬会军负责调整论文框架和指导写作；谭支鹏和王芳提出指导意见并修改论文；卢凯指导论文.

参 考 文 献

[1] Lee C, Sim D, Hwang J Y, et al. F2FS: A new file system for flash storage[C]// Proc of the 13th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2015: 273−286

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

[3] Mathur A, Cao Mingming, Bhattacharya S, et al. The new ext4 filesystem: Current status and future plans[C/OL]// Proc of the Linux Symp. 2007, 2: 21- 2: 33. [2024-05-21]. https://giis.co.in/mathur-Reprint.pdf

[4] Conway A, Bakshi A, Jiao Yizheng, et al. File systems fated for senescence? Nonsense, says science![C]// Proc of the 15th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2017: 45−58

[5] Kadekodi S, Nagarajan V, Ganger G R. Geriatrix: Aging what you see and what you don't see. A file system aging approach for modern storage systems[C]// Proc of the 2018 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 691−704

[6] Min C, Kim K, Cho H, et al. SFS: Random write considered harmful in solid state drives// Proc of the 10th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2012, 12: 1−12: 16

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

[8] Woodhouse D. JFFS: The journalling flash file system[C/OL]// Proc of the Ottawa Linux Symp. 2001 [2024-05-22]. http://pficheux.free.fr/eyrolles/linux_embarque/docs_externes/jffs2.pdf

[9] Schierl A, Schellhorn G, Haneberg D, et al. Abstract specification of the UBIFS file system for flash memory[C]// Proc of the 2nd World Congress on Formal Methods. Berlin: Springer, 2009: 190−206

[10] Lee K, Won Y. Smart layers and dumb result: IO characterization of an android-based smartphone[C]// Proc of the 10th ACM Int Conf on Embedded Software. New York: ACM, 2012: 23−32

[11] Kim H, Agrawal N, Ungureanu C. Revisiting storage for smartphones[J]. ACM Transactions on Storage, 2012, 8(4): 14: 1−14: 25

[12] Jeong S, Lee K, Lee S, et al. I/O stack optimization for smartphones[C]// Proc of the 2013 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2013: 309−320

[13] Ji Cheng, Pan Riwei, Chang Lipin, et al. Inspection and characterization of app file usage in mobile devices[J]. ACM Transactions on Storage, 2020, 16(4): 25: 1−25: 25

[14] Courville J, Chen Feng. Understanding storage I/O behaviors of mobile applications[C/OL]// Proc of the 32nd Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2016 [2024-05-22]. https://ieeexplore.ieee.org/document/7897092

[15] Li Jing, Badam A, Chandra R, et al. On the energy overhead of mobile storage systems[C]// Proc of the 12th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2014: 105−118

[16] Guo Weichao, Chen Kang, Feng Huan, et al. MARS: Mobile application relaunching speed-up through flash-aware page swapping[J]. IEEE Transactions on Computers, 2015, 65(3): 916−928

[17] Zhu Yuhao, Halpern M, Reddi V J. Event-based scheduling for energy-efficient QoS (eQoS) in mobile Web applications[C]// Proc of the 21st Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2015: 137−149

[18] Cruz L, Abreu R. EMaaS: Energy measurements as a service for mobile applications[C]// Proc of the 41st IEEE/ACM Int Conf on Software Engineering: New Ideas and Emerging Results. Piscataway, NJ: IEEE, 2019: 101−104

[19] Kwon E, Han S, Park Y, et al. Reinforcement learning-based power management policy for mobile device systems[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2021, 68(10): 4156−4169

[20] Yang Jingpei, Plasson N, Gillis G, et al. Don't stack your log on my log[C/OL]// Proc of the 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads. Berkeley, CA: USENIX Association, 2014 [2024-05-22]. https://www.usenix.org/system/files/conference/inflow14/inflow14-yang.pdf

[21] Jeong S, Lee K, Hwang J, et al. Androstep: Android storage performance analysis tool[C/OL]// Proc of the Software Engineering 2013-Workshopband. 2013: 327−340 [2024-05-22]. https://oslab.kaist.ac.kr/wp-content/uploads/esos_files/publication/conferences/international/AndroStep.pdf?ckattempt=1

[22] Hahn S S, Lee S, Yee I, et al. FastTrack: Foreground app-aware I/O management for improving user experience of android smartphones[C]// Proc of the 2018 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 15−28

[23] Han Lei, Xiao Bin, Dong Xuwei, et al. DS-Cache: A refined directory

entry lookup cache with prefix-awareness for mobile devices[C]// Proc of the 2019 Design, Automation, and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2019: 1052−1057

[24] Liang Yu, Li Qiao, Xue C J. Mismatched memory management of android smartphones[C/OL]// Proc of the 11th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2019 [2024-05-22]. https://www.usenix.org/system/files/hotstorage19-paper-liang.pdf

[25] Liang Yu, Li Jinheng, Ausavarungnirun R, et al. Acclaim: Adaptive memory reclaim to improve user experience in Android systems[C]// Proc of the 2020 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2020: 897−910

[26] Mao Bo, Wu Suzhen, Jiang Hong, et al. Content-aware trace collection and I/O deduplication for smartphones[C/OL]// Proc of the 33rd Int Conf Massive Storage Systems and Technology. Piscataway, NJ: IEEE, 2017 [2024-05-22]. https://msstconference.org/MSST-history/2017/Papers/ContentAwareTraceCollection.pdf

[27] Mao Bo, Zhou Jindong, Wu Suzhen, et al. Improving flash memory performance and reliability for smartphones with I/O deduplication[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 38(6): 1017−1027

[28] Ji Cheng, Chang Lipin, Pan Riwei, et al. Pattern-guided file compression with user-experience enhancement for log-structured file system on mobile devices[C]// Proc of the 19th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2021: 127−140

[29] Ren Jinglei, Liang C J M, Wu Yongwei, et al. Memory-centric data storage for mobile systems[C]// Proc of the 2015 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2015: 599−611

[30] Mohan J, Purohith D, Halpern M, et al. Storage on your smartphone uses more energy than you think[C/OL]// Proc of the 9th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2017 [2024-05-20]. https://www.usenix.org/system/files/conference/hotstorage17/hotstorage17-paper-mohan.pdf

[31] Carroll A, Heiser G. An analysis of power consumption in a smartphone[C/OL]// Proc of the 2010 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2010 [2025-05-22]. https://www.usenix.org/legacy/events/atc10/tech/full_papers/Carroll.pdf

[32] Yoon C, Kim D, Jung W, et al. AppScope: Application energy metering framework for Android smartphone using kernel activity monitoring[C]// Proc of the 2012 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2012: 387−400

[33] Olivier P, Boukhobza J, Senn E, et al. A methodology for estimating performance and power consumption of embedded flash file systems[J]. ACM Transactions on Embedded Computing Systems, 2016, 15(4): 79: 1−79: 25

[34] Chung Y F, Lo Y T, King C T. Enhancing user experiences by exploiting energy and launch delay trade-off of mobile multimedia applications[J]. ACM Transactions on Embedded Computing Systems, 2013, 12(1s): 37: 1−37: 19

[35] Nguyen D T, Zhou Gang, Qi Xin, et al. Storage-aware smartphone energy savings[C]// Proc of the 2013 ACM Int Joint Conf on Pervasive and Ubiquitous Computing. New York: ACM, 2013: 677−686

[36] Huang J, Badam A, Chandra R, et al. WearDrive: Fast and energy-efficient storage for wearables[C]// Proc of the 2015 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2015: 613−625

[37] Zhong Kan, Zhu Xiao, Wang Tianzheng, et al. DR. Swap: Energy-efficient paging for smartphones[C]//Proc of the 2014 Int Symp on Low Power Electronics and Design. Piscataway, NJ: IEEE, 2014: 81−86

[38] Yan Kaige, Fu Xin. Energy-efficient cache design in emerging mobile platforms: The implications and optimizations[C]// Proc of the 2015 Design, Automation, and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2015: 375−380

[39] Chang Lipin, Sung Pohan, Chen Potsang, et al. Eager synching: A selective logging strategy for fast fsync() on flash-based Android devices[J]. ACM Transactions on Embedded Computing Systems, 2016, 16(2): 34: 1−34: 25

[40] Lee E, Son I, Kim J S. An efficient order-preserving recovery for F2FS with ZNS SSD[C]//Proc of the 15th ACM Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2023: 116−122

[41] Lee C G, Byun H, Noh S, et al. Write optimization of log-structured flash file system for parallel I/O on manycore servers[C]//Proc of the 12th ACM Int Conf on Systems and Storage. Berkeley, CA: USENIX Association, 2019: 21−32

[42] Yeon J, Jeong M, Lee S, et al. RFLUSH: Rethink the flush[C]// Proc of the 16th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2018: 201−210

[43] Liang Yu, Ji Cheng, Fu Chenchen, et al. iTRIM: I/O-aware TRIM for improving user experience on mobile devices[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 40(9): 1782−1795

[44] Feng Xiaolu, Chen Xianzhang, Li Ruolan, et al. CoDiscard: A revenue model based cross-layer cooperative discarding mechanism for flash memory devices[J]. Journal of Systems Architecture, 2022, 128: 102564

[45] Seltzer M I, Smith K A, Balakrishnan H, et al. File system logging versus clustering: A performance comparison[C]// Proc of the 1995 USENIX Technical Conf. Berkeley, CA: USENIX Association, 1995: 249−264

[46] Cheng Wen, Zheng Telong, Zeng Lingfang, et al. DPLFS: A dual-mode PCM-based log-structured file system[C]// Proc of the 40th Int Conf on Computer Design. Piscataway, NJ: IEEE, 2022: 324−331

[47] Xu Jian, Swanson S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories[C]// Proc of the 14th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 323−338

[48] Xu Jian, Zhang Lu, Memaripour A, et al. Nova-fortis: A fault-tolerant non-volatile main memory file system[C]// Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 478−496

[49] Liao Xiaojian, Lu Youyou, Xu Erci, et al. MAX: A multicore-accelerated file system for flash storage[C]// Proc of the 2021 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2021: 877−891

[50] Ji Cheng, Chang Lipin, Hahn S S, et al. File fragmentation in mobile devices: Measurement, evaluation, and treatment[J]. IEEE Transactions on Mobile Computing, 2018, 18(9): 2062−2076

[51] Smith K A, Seltzer M I. File system aging—Increasing the relevance of file system benchmarks[C]//Proc of the 1997 ACM SIGMETRICS Int Conf on Measurement and Modeling of Computer Systems. New York: ACM, 1997: 203−213

[52] Liang Yu, Fu Chenchen, Du Yajuan, et al. An empirical study of F2FS on mobile devices[C/OL]// Proc of the 23rd Int Conf on Embedded and Real-Time Computing Systems and Applications. Piscataway, NJ: IEEE, 2017 [2024-05-22]. https://ieeexplore.ieee.org/document/8046304

[53] Ji Cheng, Chang Lipin, Shi Liang, et al. An empirical study of file-system fragmentation in mobile storage systems[C]// Proc of the 8th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2016: 76−80

[54] KV A K, Cao Mingming, Santos J R, et al. Ext4 block and inode allocator improvements[C/OL]// Proc of the Linux Symp, Vol 1. 2008: 263−274 [2024-05-22]. https://landley.net/kdocs/mirror/ols2008v1.pdf#page=263

[55] Sweeney A, Doucette D, Hu Wei, et al. Scalability in the XFS file system[C/OL]// Proc of the USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 1996 [2024-05-22]. https://www.cs.princeton.edu/courses/archive/fall11/cos518/papers/xfs.pdf

[56] Rodeh O, Bacik J, Mason C. BTRFS: The Linux B-tree filesystem[J]. ACM Transactions on Storage, 2013, 9(3): 9: 1−9: 32

[57] Bonwick J, Ahrens M, Henson V, et al. The Zettabyte file system[C/OL]// Proc of the 2nd USENIX Conf on File and Storage Technologies, Vol 215. Berkeley, CA: USENIX Association, 2003 [2024-05-22]. https://www.cs.hmc.edu/~rhodes/courses/cs134/fa20/readings/The%20Zettabyte%20File%20System.pdf

[58] Jannen W, Yuan Jun, Zhan Yang, et al. BetrFS: A right-optimized write-optimized file system[C]// Proc of the 13th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2015: 301−315

[59] Yuan Jun, Zhan Yang, Jannen W, et al. Optimizing every operation in a write-optimized file system[C]// Proc of the 14th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 1−14

[60] Conway A, Knorr E, Jiao Yizheng, et al. Filesystem aging: It's more usage than fullness[C/OL]// Proc of the 11th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2019 [2024-05-22]. https://www.usenix.org/system/files/hotstorage19-paper-conway.pdf

[61] Park J, Eom Y I. FragPicker: A new defragmentation tool for modern storage devices[C]// Proc of the 28th Symp on Operating Systems Principles. New York: ACM, 2021: 280−294

[62] Kesavan R, Curtis-Maury M, Devadas V, et al. Countering fragmentation in an enterprise storage system[J]. ACM Transactions on Storage, 2020, 15(4): 25: 1−25: 35

[63] Hahn S S, Lee S, Ji Cheng, et al. Improving file system performance of mobile storage systems using a decoupled defragmenter[C]// Proc of the 2017 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 759−771

[64] Nakamura T, Komoda N. Pre-allocation size adjusting methods depending on growing file size[C]// Proc of the 5th IEEE Int Workshop on Storage Network Architecture and Parallel I/Os. Piscataway, NJ: IEEE, 2008: 19−25

[65] Djordjevic B, Timcenko V. Ext4 file system in Linux environment: Features and performance analysis[J]. International Journal of Computers, 2012, 6(1): 37−45

[66] Li Qi, Deng Aosong, Gao Congming, et al. Optimizing fragmentation and segment cleaning for CPS based storage devices[C]// Proc of the 34th ACM/SIGAPP Symp on Applied Computing. New York: ACM, 2019: 242−249

[67] Yang Lihua, Wang Fang, Tan Zhipeng, et al. ARS: Reducing F2FS fragmentation for smartphones using decision trees[C]// Proc of the 2020 Design, Automation, and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2020: 1061−1066

[68] Yang Lihua, Tan Zhipeng, Wang Fang, et al. Improving F2FS performance in mobile devices with adaptive reserved space based on traceback[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 41(1): 169−182

[69] Ahn W H, Kim K, Choi Y, et al. DFS: A de-fragmented file system[C]// Proc of the 10th IEEE Int Symp on Modeling, Analysis and Simulation of Computer and Telecommunications Systems. Piscataway, NJ: IEEE, 2002: 71−80

[70] Park J, Kang D H, Eom Y I. File defragmentation scheme for a log-structured file system// Proc of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2016: 19: 1−19: 7

[71] Park J, Eom Y I. Anti-aging LFS: Self-defragmentation with fragmentation-aware cleaning[J]. IEEE Access, 2020, 8: 151474−151486

[72] Wu Chao, Cui Yufei, Ji Cheng, et al. Pruning deep reinforcement learning for dual user experience and storage lifetime improvement on mobile devices[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3993−4005

[73] Yang Lihua, Tan Zhipeng, Wang Fang, et al. FAGC: Free space fragmentation aware GC scheme based on observations of energy consumption[C/OL]// Proc of the 2023 Design, Automation, and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2023 [2024-05-22]. https://ieeexplore.ieee.org/document/10137040

[74] Wang Jun, Hu Yiming. WOLF—A novel reordering write buffer to boost the performance of log-structured file systems[C]// Proc of the 1st USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2002: 47−60

[75] Kang D H, Eom Y I. Dynamic hot-cold separation scheme on the log-structured file system for CE devices[C]// Proc of the 2017 IEEE Int Conf on Consumer Electronics. Piscataway, NJ: IEEE, 2017: 428−429

[76] Kang M, Choi S, Oh G, et al. 2R: Efficiently isolating cold pages in flash storages[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2004−2017

[77] Sun Penghao, You Litong, Zheng Shengan, et al. Learning-based data separation for write amplification reduction in solid state

drives[C/OL]// Proc of the 60th ACM/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2023 [2024-05-22]. https://ieeexplore.ieee.org/abstract/document/10247795

[78] Zhang Qiang, Liang Jie, Xu Yinlong, et al. Research of SSD array architecture based on workload awareness[J]. Journal of Computer Research and Development, 2019, 56(4): 755−766 (in Chinese)
(张强, 梁杰, 许胤龙, 等. 基于工作负载感知的固态硬盘阵列系统的架构设计与研究[J]. 计算机研究与发展, 2019, 56(4): 755−766)

[79] Wang Wenguang, Zhao Yanping, Bunt R. HyLog: A high performance approach to managing disk layout[C]// Proc of the 3rd USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2004: 145−158

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

[81] Li Bingzhe, Deng Chunhua, Yang Jinfeng, et al. HAML-SSD: A hardware accelerated hotness-aware machine learning based ssd management[C/OL]// Proc of the 2019 IEEE/ACM Int Conf on Computer-Aided Design. Piscataway, NJ: IEEE, 2019 [2024-05-22]. https://ieeexplore.ieee.org/document/8942140

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

[83] Kim T, Hong D, Hahn S S, et al. Fully automatic stream management for multi-streamed SSDs using program contexts[C]// Proc of the 17th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2019: 295−308

[84] Yang Lihua, Tan Zhipeng, Wang Fang, et al. M2H: Optimizing F2FS via multi-log delayed writing and modified segment cleaning based on dynamically identified hotnessn[C]// Proc of the 2021 Design, Automation, and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2021: 808−811

[85] Gwak H, Kang Y, Shin D. Reducing garbage collection overhead of log-structured file systems with GC journaling[C/OL]// Proc of the 2015 Int Symp on Consumer Electronics. Piscataway, NJ: IEEE, 2015 [2022-05-22]. https://ieeexplore.ieee.org/document/7177770

[86] Gwak H, Shin D. SCJ: Segment cleaning journaling for log-structured file systems[J]. IEEE Access, 2021, 9: 142437−142448

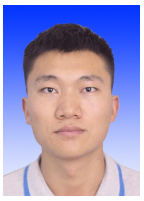
[87] Park D, Cheon S, Won Y. Suspend-aware segment cleaning in log-structured file system[C/OL]// Proc of the 7th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2015 [2024-05-22]. https://www.usenix.org/system/files/conference/hotstorage15/hotstorage15-park.pdf

[88] Wu Chao, Ji Cheng, Xue C J. Reinforcement learning based background segment cleaning for log-structured file system on mobile devices[C/OL]// Proc of the 15th IEEE Int Conf on Embedded Software and Systems. Piscataway, NJ: IEEE, 2019 [2024-05-22]. https://ieeexplore.ieee.org/document/8782508

[89] Yu Chao. Support age-threshold based garbage collection for F2FS[EB/OL]. (2020-08-04)[2024-05-22]. https://lwn.net/Articles/828027/

Yang Lihua, born in 1994. PhD. Her main research interests include computer architecture, file system, and mobile storage.
杨梨花, 1994 年生. 博士. 主要研究方向为计算机系统结构、文件系统、移动存储.

Dong Yong, born in 1980. PhD, professor. His main research interests include computer architecture, parallel I/O, and file system.
董　勇, 1980 年生. 博士, 研究员. 主要研究方向为计算机系统结构、并行 I/O、文件系统.



Wu Huijun, born in 1991. PhD, assistant professor. His main research interests include parallel file system, burst buffer file system, and machine learning.
邬会军, 1991 年生. 博士, 助理研究员. 主要研究方向为并行文件系统、加速层文件系统、机器学习.

Tan Zhipeng, born in 1973. PhD, professor, PhD supervisor. His main research interests include computer architecture, big data storage and management, and mobile storage.
谭支鹏, 1973 年生. 博士, 教授, 博士生导师. 主要研究方向为计算机系统结构、大数据存储与管理、移动存储.

Wang Fang, born in 1972. PhD, professor, PhD supervisor. Her main research interests include parallel file systems, new storage systems based on non-volatile storage devices, and network storage.
王　芳, 1972 年生. 博士, 教授, 博士生导师. 主要研究方向为并行文件系统、基于非易失存储器件的新型存储系统、网络存储.

Lu Kai, born in 1973. PhD, professor, PhD supervisor. His main research interests include computer architecture, operating system, and security research.
卢　凯, 1973 年生. 博士, 研究员, 博士生导师. 主要研究方向为计算机系统结构、操作系统、安全性研究.