软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(6):1909−1929 [doi: 10.13328/j.cnki.jos.005706]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

持久化内存文件系统的磨损攻击与防御机制*

杨朝树[1], 诸葛晴凤[2], 沙行勉[1,2], 陈咸彰[1,3], 吴 林[1], 吴 挺[1]

[1](重庆大学 计算机学院,重庆 400044)

[2](华东师范大学 计算机科学与软件工程学院,上海 200062)

[3](重庆大学 通信工程学院,重庆 400044)

通讯作者: 沙行勉, E-mail: edwinsha@gmail.com

摘 要: 近来出现诸多以非易失性存储器(non-volatile memory,简称 NVM)作为存储设备的新型持久化内存文件系统,充分发掘 NVM 的低延迟和可按字节寻址等优点,优化文件访问的 I/O 栈和一致性机制,极大提升文件系统的性能.然而,现有持久化内存文件系统都没有考虑 NVM 写耐受度低的缺陷,极易导致 NVM 被磨损穿(wear out).针对 NVM 写耐受度低的缺点,探索多种利用基本文件操作对 NVM 造成磨损攻击的方式,并在真实持久化内存文件系统 PMFS 中以实验证明磨损攻击的严重性.为有效防御针对 NVM 的磨损攻击,提出了持久化内存文件系统磨损防御机制(persistent in-memory file system wear defense technique,简称 PFWD),包括索引节点元数据虚拟化技术、超级块迁移技术、文件数据页磨损均衡技术和文件索引结构迁移技术,保护文件系统中所有可能被磨损攻击利用的数据结构.实验结果证明所提出的 PFWD 技术能有效地防御病毒发动对 NVM 的磨损攻击,提高了存储系统的稳定性.

关键词: 非易失性存储器;持久化内存文件系统;磨损攻击病毒;磨损防御机制;数据迁移

中图法分类号: TP316

中文引用格式: 杨朝树,诸葛晴凤,沙行勉,陈咸彰,吴林,吴挺.持久化内存文件系统的磨损攻击与防御机制.软件学报,2020,31(6):1909−1929. http://www.jos.org.cn/1000-9825/5706.htm

英文引用格式: Yang CS, Zhuge QF, Sha EHM, Chen XZ, Wu L, Wu T. Wear attacks and defense mechanisms for persistent in-memory file systems. Ruan Jian Xue Bao/Journal of Software, 2020,31(6):1909−1929 (in Chinese). http://www.jos.org.cn/1000-9825/5706.htm

Wear Attacks and Defense Mechanisms for Persistent In-memory File Systems

YANG Chao-Shu[1], ZHUGE Qing-Feng[2], Edwin H-M Sha[1,2], CHEN Xian-Zhang[1,3], WU Lin[1], WU Ting[1]

[1](College of Computer Science, Chongqing University, Chongqing 400044, China)

[2](School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China)

[3](College of Communication Engineering, Chongqing University, Chongqing 400044, China)

Abstract: Recently, many new persistent in-memory file systems are proposed to exploit the advantages of non-volatile memory (NVM), such as low latency and byte-addressability. As a result, the performance of the persistent in-memory file systems is greatly improved by optimizing the I/O stack and data consistency mechanisms. Nevertheless, existing persistent in-memory file systems ignores the limited write endurance of NVM, which can easily lead to the wear out of NVM. This study first explores wear attacks using the common file system operations to wear out the underlying NVM devices of persistent in-memory file systems. The effectiveness of the wear attacks is proved by experiments in PMFS, a real persistent in-memory file system. Then, in order to prevent NVM from malicious wear attacks, this study proposes a persistent in-memory file system wear defense (PFWD) strategy, which includes inode virtualization, super block

* 基金项目: 国家高技术研究发展计划(863)(2015AA015304); 国家自然科学基金(61472052); 中国博士后科学基金(2017M620412)

Foundation item: National High-Tech R&D Program of China (863) (2015AA015304); National Natural Science Foundation of China (61472052); China Postdoctoral Science Foundation (2017M620412)

收稿时间: 2018-01-09; 修改时间: 2018-05-22; 采用时间: 2018-10-26

migration, data page wear-leveling, and file index structure migration, to protect all the data structures of file systems that may be exploited by wear attacks. Experimental results show that PFWD can effectively prevent NVM from wear attacks and improve the stability of the storage system.

Key words: non-volatile memory; persistent in-memory file system; wear attack virus; wear defense mechanism; data migration

目前,处理器和存储系统之间的数据 I/O 是极为严重的性能瓶颈,导致计算机系统无法应对上层应用的强时效性和高可靠等存储服务需求.新型非易失性存储器(non-volatile memory,简称 NVM),如相变存储器(phase change memory,简称 PCM)[1−4]和 3D Xpoint[5]等,具有非易失性、低延迟、存储密度高、抗震性好、低功耗和可按字节寻址等优点.表 1 中 PCM 的存储密度是 DRAM 的 2 倍~4 倍,读写延迟比 NAND Flash 低 3 个和 2 个数量级,比 HDD 分别低 5 个和 3 个数量级.鉴于 NVM 的优良特性,取得了学术界和工业界的广泛关注,被视为潜力巨大的新一代存储设备.NVM 给现有的存储系统的发展带来了新的机遇,NVM 既可以作为内存,也可以作为外存[3,6,7].近年出现诸多利用 NVM 作为内存的新型持久化内存文件系统,例如 BPFS[8],PMFS[9],NOVA[10],SIMFS[11],SCMFS[12]和 HiNFS[13]等.这类持久化内存文件系统充分发挥 NVM 的低延迟、可按字节寻址等优点,优化文件系统的 I/O 栈,使得文件访问吞吐率达到 GB/s 级.

Table 1 Comparison among DRAM, PCM, NAND Flash, and HDD[14]

表 1 比较 DRAM,PCM,NAND Flash 和 HDD[14]

	DRAM	PCM	NAND Flash	HDD
密度(density)	1X	2~4X	4X	>4X
非易失性	否	是	是	是
按字节寻址	是	是	否	否
写延迟	20~50ns	~1μs	~500μs	~5ms
读延迟	50ns	50~100ns	25~50μs	~5ms
写耐受度(write endurance)	10^{15}	~10^{8}	<10^{4}	10^{15}

虽然 NVM 拥有诸多优点,但是却普遍具有一个重要的缺陷,即存储单元的写耐受度低[1−3,15−23].PCM 是代表性的 NVM 存储器,利用相变材料的晶态(低电阻,表示“1”)和非晶态(高电阻,表示“0”)所表现出来的导电性差异来存储数据.数据在 0 与 1 之间的改变需要向存储单元加大电流,使其在晶态与非晶态之间转变,导致存储单元的磨损.所以 PCM 存储单元的写耐受度有限,最大写次数约为 10^{8}[24−27],即 PCM 相变材料在晶体和非晶体的转变次数达到 10^{8} 时,该存储单元将变得不稳定,被视为已损坏(即磨损穿).因此,病毒主要针对 NVM 写耐受度低的缺点,通过持续修改 NVM 某个存储单元,最终损坏该存储单元,导致数据出错,进而破坏整个存储系统的可靠性.现有防御措施的主要思想是以空间换寿命,通过磨损均衡技术把针对少量存储单元的大量写操作分散到其他存储空间,使得每个存储单元的磨损度相对平均,从而避免被过度磨损.

现阶段学术界在两个层面探索磨损均衡技术:在硬件层面,典型的方法有 Start-Gap[28]和安全刷新算法(security refresh)[29],核心思想是通过在控制器动态改变逻辑地址到物理地址的映射来实现磨损均衡,但是这些磨损均衡算法抗恶意攻击的能力不强,RTA(remapping timing attack)[30]等病毒程序可以迅速探测到映射的变化规律,进而继续对固定的物理存储单元造成损伤;另一种是软件层面,主要集中在操作系统,通过动态改变进程的逻辑地址到物理地址的映射来实现磨损均衡.作为管理 NVM 存储设备的基本设施,目前的持久化内存文件系统却没有防御针对 NVM 的磨损攻击.例如:SanGuo[31]提出一种考虑磨损度的空闲页分配技术,却没有系统性地保护既有文件数据和元数据的存储单元;文献[24]提出面向 PCM 存储系统的磨损均衡机制,但是不能充分抵御针对文件索引结构等元数据和文件数据存储区的磨损攻击.病毒程序利用简单的文件操作,即可磨损穿现有持久化内存文件系统的文件数据或元数据存储区,导致文件系统的数据错误,严重破坏存储系统的可靠性.

为此,本文首次探索多种借助持久化内存文件系统的文件操作对 NVM 造成恶意磨损的攻击方式.通过分析各种攻击方式攻击的数据结构,本文提出持久化内存文件系统磨损防御技术(persistent in-memory file system wear defense technique,简称 PFWD)来保护 NVM 的存储单元.PFWD 技术包括 4 个部分,分别是索引节点元数据虚拟化技术、超级块迁移技术、文件数据页磨损均衡技术和文件索引结构迁移技术.PFWD 的核心思想:对索

引节点、超级块、文件数据、文件索引结构和日志的更新操作均匀分布到整个 NVM 存储空间.RTA 无法攻破本文提出的防御机制,因为 RTA 的有效攻击需要满足两个必要条件:一是要探测到物理空间重映射后的地址;二是获取写该地址的权限.在本文提出的基于文件系统的 PFWD 防御机制中,RTA 不具备这两个条件,因为 PFWD 把达到磨损迁移条件的物理空间直接回收到空闲链表并防止立即重分配,其地址映射信息对上层应用封闭,所以 RTA 无法查知该物理空间的地址,也不能获取其访问权限.因此,本文将探讨通过文件系统对 NVM 存储空间的磨损攻击方式,并提出对应的 PFWD 防御技术,实现 NVM 存储空间的磨损均衡.本文的主要贡献如下.

(1) 探索多种通过持久化内存文件系统对 NVM 造成磨损攻击的病毒,分析病毒恶意磨损攻击的严重性,并按攻击的数据结构分类病毒程序;

(2) 提出索引节点元数据虚拟化技术防御病毒程序对索引节点存储区的磨损攻击.该技术可在系统运行时,把更新频次高的索引节点动态迁移到磨损度低的存储区,使得索引节点的大量更新操作分散到多个物理存储区,实现索引节点存储区的磨损均衡,避免少数索引节点的存储单元被磨损穿;

(3) 提出超级块迁移技术,通过把超级块迁移到磨损度低的存储区,使得对超级块的大量更新操作分散到多个物理存储区,避免超级块的存储单元被磨损穿;

(4) 提出文件数据页磨损均衡技术,结合持久化内存文件系统必备的数据一致性机制,防御对文件数据和日志存储单元的磨损攻击.对文件进行修改,总是获取磨损度较低的空闲页作为日志和文件数据的存储区,避免少数文件数据和日志的存储单元被磨损穿;

(5) 提出文件索引结构迁移技术,根据文件索引结构物理空间的磨损程度,把更新频次高的索引项迁移到磨损度低的物理页,防止少数文件索引结构的索引项的存储单元被磨损穿;

(6) 实验表明:在病毒程序的恶意磨损攻击下,本文提出的 PFWD 技术能将 NVM 可容忍的总写次数提高 10 240 倍,而且该倍数随着 NVM 存储空间的增大而增大.

本文第 1 节是本文的研究动机.第 2 节探索通过文件操作攻击 NVM 的磨损攻击.第 3 节介绍本文提出的磨损攻击防御机制.第 4 节用实验验证提出的防御机制的有效性.第 5 节总结全文.

1 研究动机

持久化内存文件系统包括两类数据,即元数据和文件数据.元数据可分为超级块、索引节点和文件索引结构,对大多数文件系统而言,还包括用于支持数据一致性的日志.现有持久化内存文件系统的设计没有对文件数据和元数据实现磨损保护,病毒程序可以轻易通过文件操作造成 NVM 存储设备的损坏.以下分析现有持久化内存文件系统在磨损保护方面的设计缺陷:

(1) 超级块.保存文件系统的全局信息,如 NVM 的空闲页数和空闲索引节点数等,所以 NVM 空间的分配和释放、索引节点的申请和释放都会修改超级块,见表 2,大量文件操作都需要更新超级块,所以超级块的更新频次极高,但现有的持久化内存文件系统都把超级块保存在 NVM 固定位置,不能移动,导致超级块存储区的磨损极为严重.

Table 2 Common file operations

表 2 常用的文件操作

	write	read	mkdir	rename	link	open	symlink	unlink	create	delete
超级块	✓	×	✓	×	×	×	✓	✓	✓	✓
索引节点	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
文件索引结构	✓	×	✓	✓	×	×	×	×	✓	✓
文件数据	✓	×	✓	×	×	×	✓	✓	✓	✓
日志	✓	×	✓	✓	✓	×	✓	✓	✓	✓

(2) 索引节点.保存文件的基本信息,如文件大小、文件数据的最后访问时间(atime)、文件数据的最后修改时间(mtime)、索引节点的最后修改时间(ctime)、存储文件数据的物理页数、链接数等.文件的创建、打开、关闭、硬链接、软链接等操作都会更新索引节点.虽然现有的持久化内存文件系统采用缓存机制[32]来减少更新

NVM 中真实索引节点的写次数,但存储文件数据的物理页数、链接数、文件大小等重要信息都必须立即写回NVM,以保证数据的一致性.现有的持久化内存文件系统使用数组或者树型结构组织文件系统的索引节点.NOVA[10], SIMFS[11]和 SCMFS[12]采用数组结构管理索引节点,即在NVM划分一段连续的物理空间存储所有的索引节点; BPFS[8],PMFS[9]和 HiNFS[13]采用树形结构管理索引节点,即所有索引节点的物理存储空间按段管理,因此,索引节点分散存放在 NVM 的物理空间.无论采用那种数据结构保存索引节点,每个文件的索引节点在它的生命周期内都存放在 NVM 的固定位置,不能移动.如表 2 所示,大量常用文件操作都需要更新索引节点,所以索引节点存储区的磨损极为严重.

(3) 文件索引结构.更新文件索引结构和数据一致性的实现机制有关,文件系统的一致性主要分为元数据一致性和数据一致性.元数据一致性仅保证文件元数据能恢复到一致性状态;数据一致性同时保证文件元数据和文件数据恢复到一致性状态,因此数据一致性保证一致性的强度比文件元数据一致性高.文件系统主要采用预写日志(write-ahead logging,简称 WAL)[33−35]和写时复制(copy-on-write,简称 COW)[8,11]机制来实现数据一致性.采用预写日志机制实现数据一致性,文件覆盖写是直接修改文件数据的原存储位置,因此不涉及修改文件索引结构;采用写时复制机制实现数据一致性,文件覆盖写是把数据写到新的存储位置,然后修改文件索引结构.预写日志和写时复制机制,文件执行数据追加(append write)操作,都在文件索引结构添加索引项(备注:如果文件最后的数据页不能保存本次追加的数据).现有的持久化内存文件系统,例如 SIMFS[11]采用伪文件写(pseudo-file write,简称 PFW),BPFS[8]采用短路影子分页(short-circuit shadow paging,简称 SCSP),两种都属于写时复制机制.NOVA[10]文件数据一致性采用写时复制机制;Shortcut-JFS[16]采用自适应日志(adaptive logging,简称 AL),即预写日志结合写时复制机制.AL 的核心思想是:如果单个数据页上更新的数据量小于页大小的一半,则采用预写日志机制实现数据一致性;如果单个数据页上更新的数据量大于或等于页大小的一半,则采用写时复制机制实现数据一致性.现有的持久化内存文件系统,如果采用写时复制实现数据一致性,文件执行覆盖写操作,需修改文件索引结构,每个文件的索引结构在它的生命周期内都存放在 NVM 的固定位置,不能移动,如果持续修改文件某个区间的数据,需频繁修改文件索引结构,导致文件索引结构的存储区持续被磨损.

(4) 文件数据.更新文件数据和数据一致性实现机制有关,如果持久化内存文件系统采用预写日志机制[33−35]实现数据一致性,文件覆盖写是直接就地更新文件数据的原存储位置.现有的持久化内存文件系统,例如Shortcut-JFS[16],如果单个数据页上更新的数据量小于页大小的一半,则采用预写日志机制实现数据一致性,持续修改文件的数据量小于页大小一半时,则被修改文件数据的存储单元持续被磨损;PMFS[9]由于未实现数据一致性,仅支持元数据一致性,文件覆盖写是直接就地更新文件数据的原存储位置,持续修改文件的固定区间的数据,则被修改数据的存储单元持续被磨损.

(5) 日志.持久化内存文件系统采用日志实现数据一致性.日志的存储区可以是 NVM 的一段连续的区域,例如 PMFS[9]的日志保存在一段连续的存储空间(PMFS-Log)中,持续地执行基本的文件操作,就能写穿 PMFS-Log的存储单元;日志的存储区域也可以分散在整个NVM存储空间,例如,NOVA[10]为每个索引节点管理一个私有日志(日志结构体),以下简称索引节点私有日志,默认大小是 4KB,通过在索引节点私有日志中记录索引节点修改信息实现元数据一致性,同时,通过垃圾回收机制回收已提交的日志的存储空间.针对通过修改多个索引节点的操作,例如 rename,unlink 等,NOVA 为每个 CPU 分配一个日志(journal)区间来保存索引节点的修改信息,以保证元数据一致性,日志存储空间默认大小是 4KB,简称索引节点公有日志.NOVA 的私有日志和公有日志保存在固定的存储区间,因此,持续执行简单的文件操作,就能把 NOVA 的日志存储区间磨损穿.无论采用何种方式组织日志,现有的持久化内存文件系统都没有考虑日志存储区间的磨损均衡,如表 2 所示,大多数文件操作都需要写日志来确保文件的一致性,因此,日志的修改频次极高,所以日志存储区的磨损极为严重.

文件更新操作都会涉及到修改元数据或者文件数据,表 2 展示了常用的文件操作对元数据和文件数据的更新情况.例如文件更新操作(write)需要修改索引节点的文件数据的最后修改时间,如果文件更新操作涉及到申请新的物理页,则需要修改超级块的空闲页数和修改文件索引结构的索引项.为了保证本次修改的数据一致性,需要写日志.所以文件更新操作需要修改超级块、索引节点、文件索引结构、文件数据和日志.

由以上可知:每个文件的元数据在它的生命周期都存放在 NVM 的固定位置,都不会被移动,文件数据是否移动和持久化内存文件系统实现数据一致性机制有关,所以攻击者实现病毒程序,通过文件操作持续修改文件的元数据或者数据,使得元数据或文件数据的存储区迅速被磨损穿.例如,PCM 存储单元的最大写次数约为 10^8[24−27],PMFS 的写延迟为 300ns~500ns[9].持续修改 NVM 的某个存储单元,则可以通过下面公式估算 PCM 的使用寿命:

使用寿命(s)=存储单元最大的写次数×持久化内存文件系统写延迟 (1)

PCM 存储单元最大写次数取 10^8,PMFS 的写延迟取 500ns,则 PCM 的使用寿命为 50s,不到 1 分钟就能把 PCM 磨损穿,所以现有持久化内存文件系统的设计,攻击者利用文件操作轻易就能把 NVM 存储设备损坏.

为此,本文将基于现有持久化内存文件系统的以上设计缺陷,探索通过文件操作攻击底层 NVM 存储设备的方法,并提出相应的保护机制.

2 针对 NVM 的磨损攻击

本节探索通过文件操作对 NVM 造成恶意磨损的攻击方式.如第 1 节所述,文件系统的大量操作会对 NVM 造成磨损.本文发现,对于现有无磨损攻击防御机制的持久化内存文件系统,病毒程序不需复杂的设计,仅需执行简单的文件操作即可造成 NVM 磨损穿.为此,本文提出 5 种利用不同的基本文件操作对 NVM 造成恶意磨损的攻击方式,利用这 5 种基本攻击方式,可以任意组合成具有更大破坏力的病毒.为展示不同攻击方式的破坏性,通过列举每种攻击方式修改的数据结构,计算每种数据结构的写次数,然后分析数据结构和物理存储对应关系,最后分析 NVM 物理页的磨损度.

- 攻击方式 1:利用创建文件和删除文件操作执行磨损攻击.

创建文件需要完成 4 个操作:(1) 申请索引节点;(2) 申请物理页,创建文件索引结构;(3) 在父目录插入一条目录项;(4) 修改超级块的空闲页数和空闲索引节点数.相应地,删除文件也需要完成 4 个操作:(1) 释放文件索引结构的物理页;(2) 在父目录删除文件的目录项;(3) 释放索引节点;(4) 修改超级块的空闲页数和空闲索引节点数.此外,创建和删除文件还需要写日志确保文件的一致性.所以如表 3 所示,创建文件和删除文件操作会造成超级块、索引节点、文件索引结构、目录文件数据和日志的更新.

Table 3 Data structures of persistent in-memory file systems under wear attacks

表 3 持久化内存文件系统被磨损攻击的数据结构

	超级块	索引节点	文件索引结构	文件数据	日志
攻击方式 1	✓	✓	✓	✓	✓
攻击方式 2	×	✓	×	✓	✓
攻击方式 3	✓	✓	×	✓	✓
攻击方式 4	✓	✓	✓	✓	✓
攻击方式 5	×	✓	×	✓	✓

因此,攻击方式 1 就是在持久化内存文件系统的单个目录下反复创建文件、删除文件,例子如病毒程序 1 所示:通过一个简单的 for 循环,在一个给定的目录下执行 10^8 次创建文件和删除文件操作,对文件系统的元数据和文件数据的存储区造成大量写操作.数据结构的写次数见表 4,所有数据结构的写次数大于或等于 10^8.索引节点、文件索引结构和日志是否修改固定的存储单元,这与持久化内存文件系统的索引节点、物理页、日志管理有关.但是超级块和目录文件数据的存储单元,这两个数据结构在病毒程序 1 的操作中从不改变存储位置,而且每次创建和删除文件都会更新,所以病毒程序会在对应的存储单元执行 2×10^8 次写操作,迅速导致其磨损穿.

病毒程序 1. 攻击方式 1 的示例代码.

输入:*src*:文件路径及名称;

输出:*NULL*.

```
1.  void LoopDelAfterCreateFile(char *src)
2.  {
```

```
3.    FILE *fd;
4.    for (int i=0; i<100000000; i++)
5.    {
6.      fd=fopen(src,"wb+");              /*创建文件*/
7.      if (0>fd)
8.      {
9.        printf("Failed to create file");
10.       return;
11.     }
12.     fclose(fd);                       /*关闭文件*/
13.     remove(src);                      /*删除文件*/
14.   }
15. }
```

Table 4 Revision counts of data structures of persistent in-memory file systems under wear attacks

表 4 持久化内存文件系统中被磨损攻击数据结构的写次数

	超级块	索引节点	文件索引结构	文件数据	日志
攻击方式 1	2×10^8	1×10^8	2×10^8	2×10^8	2×10^8
攻击方式 2	0	2×10^8	0	2×10^8	2×10^8
攻击方式 3	2×10^8	1×10^8	0	2×10^8	2×10^8
攻击方式 4	1×10^8	1×10^8	1×10^8	1×10^8	1×10^8
攻击方式 5	0	1×10^8	0	1×10^8	1×10^8

- 攻击方式 2:利用创建硬链接和删除硬链接操作执行磨损攻击.

创建硬链接需要完成两个操作:1) 在目标文件的父目录增加一条目录项;2) 源文件索引节点的链接数加一.相应地,删除硬链接也需要完成两个操作:1) 在目标文件父目录删除一条目录项;2) 源文件索引节点的链接数减一.此外,创建和删除硬链接还需写日志确保文件的一致性.如表 3 所示,创建硬链接和删除硬链接会造成索引节点、目录文件数据和日志的更新.

因此,攻击方式 2 就是在持久化内存文件系统对某个文件反复创建硬链接、删除硬链接,例子如病毒程序 2 所示:通过一个简单的 for 循环,对一个给定的文件执行 10^8 次创建硬链接和删除硬链接操作,对文件系统的元数据和文件数据的存储区造成大量写操作.数据结构的写次数见表 4,索引节点、文件数据和日志的写次数大于 10^8.日志是否修改固定的存储单元与日志管理有关,目录文件数据和索引节点的存储单元,这两个数据结构在病毒程序 2 的操作中从不改变存储位置,而且每次创建和删除硬链接都会更新,所以病毒程序会在对应的存储单元执行 2×10^8 次写操作,迅速导致其磨损穿.

病毒程序 2. 攻击方式 2 的示例代码.

输入:*src*:源文件路径及名称;*dest*:目标文件路径及名称;

输出:*NULL*.

```
1.  void LoopDelAfterLinkFile(char *src,char *dest)
2.  {
3.    FILE *fd;
4.    fd=open(src,"wb+");                 /*创建文件*/
5.    if (0>fd)
6.    {
7.      printf("Failed to open file");
8.      return;
```

```
9.      }
10.     close(fd);                                  /*关闭文件*/
11.     for (int i=0; i<100000000; i++)
12.     {
13.        link(src,dest);                          /*建立硬链接*/
14.        unlink(dest);                            /*删除硬链接*/
15.     }
16.  }
```

- 攻击方式 3:利用创建软链接和删除软链接操作执行磨损攻击.

创建软链接需要完成 4 个操作:1) 申请目标文件的索引节点;2) 在目标文件父目录增加目录项;3) 申请物理页保存源文件的位置信息;4) 修改超级块的空闲页数和空闲索引节点数.相应地,删除软链接也需要完成 4 个操作:1) 释放保存源文件位置信息的物理页;2) 在目标文件父目录删除目标文件的目录项;3) 释放目标文件的索引节点;4) 修改超级块的空闲页数和空闲索引节点数.此外,创建和删除软链接还需写日志确保文件的一致性.如表 3 所示,创建软链接和删除软链接会造成超级块、索引节点、目录文件数据和日志的更新.

因此,攻击方式 3 就是在持久化内存文件系统对某个文件反复创建软链接、删除软链接,例子如病毒程序 3 所示:通过简单的 for 循环,对给定的文件执行 10^8 次创建软链接和删除软链接操作,对文件系统的元数据和文件数据存储区造成大量写操作.数据结构的写次数见表 4,超级块、索引节点、目录文件数据和日志的写次数大于或等于 10^8.索引节点、日志是否修改固定的存储单元与索引节点和日志管理有关,超级块和目录文件数据的存储单元,这两个数据结构在病毒程序 3 的操作中从不改变存储位置,而且每次创建和删除软链接都会更新,所以病毒程序会在对应的存储单元执行 2×10^8 次写操作,迅速导致其磨损穿.

病毒程序 3. 攻击方式 3 的示例代码.

输入:*src*:源文件路径及名称;*dest*:目标文件路径及名称;

输出:*NULL*.

```
1.  void LoopDelAfterSymLinkFile(char *src,char *dest)
2.  {
3.     FILE *fd;
4.     fd=open(src,"wb+");                          /*创建文件*/
5.     if (0 >fd)
6.     {
7.        printf("Failed to open file");
8.        return;
9.     }
10.    close(fd);                                   /*关闭文件*/
11.    for (int i=0; i<100000000; i++)
12.    {
13.       symlink(src,dest);                        /*建立软链接*/
14.       unlink(dest);                             /*删除软链接*/
15.    }
16. }
```

- 攻击方式 4:利用文件覆盖写操作执行磨损攻击.

文件的覆盖写操作和文件系统实现数据一致性机制有关,如果采用预写日志机制[33−35]实现数据一致性,则文件覆盖写需要完成 3 步操作:1) 把更新内容写入日志;2) 把更新内容写入文件;3) 修改索引节点.如果采用写

时复制机制[8,11]实现数据一致性,则文件覆盖写需要完成 5 步操作:1) 把更新内容写入新申请的物理页; 2) 修改文件索引结构;3) 修改索引节点;4) 释放被替换的文件数据页;5) 修改超级块的空闲页数.此外,文件覆盖写还需写日志确保文件的一致性.如表 3 所示,文件覆盖写操作会造成超级块、索引节点、文件索引结构、文件数据和日志的更新.

因此,攻击方式 4 就是在持久化内存文件系统对某个文件反复执行覆盖写操作,例子如病毒程序 4 所示:通过一个简单的 for 循环,对一个给定的文件执行 10^8 次覆盖写操作,对文件元数据和文件数据存储区造成大量写操作.数据结构的写次数见表 4,索引节点、文件索引结构、文件数据和日志的写次数等于 10^8.每种数据结构的写次数和文件系统实现数据一致性机制息息相关,采用预写日志机制,文件执行覆盖写操作,对文件的数据的修改是直接修改原存储位置,没有涉及修改文件索引结构.因此,文件索引结构的写次数为 0,文件数据的写次数为 10^8,病毒程序 4 持续修改文件数据的固定区间 10^8 次,则文件数据页被磨损穿;采用写时复制机制,数据写到新的位置,即需要申请物理页和修改文件索引结构,则超级块、文件索引结构的写次数为 10^8.这两个数据结构在病毒程序 4 的操作中从不改变存储位置,而且每次都会更新,所以病毒程序会在对应的存储单元执行 10^8 写操作,迅速导致其磨损穿.无论是采用预写日志还是写时复制机制实现数据一致性,都需要修改索引节点和写日志,如表 4 所示,索引节点和日志的写次数都为 10^8,日志是否修改固定的存储单元与日志管理有关.索引节点在病毒 4 的操作中从不改变存储位置,所以病毒程序会在对应的存储单元执行 10^8 写操作,迅速导致其磨损穿.

病毒程序 4. 攻击方式 4 的示例代码.

输入:*src*:文件路径及名称;

输出:*NULL*.

```
1.  void LoopWriteFile(char *src,char *dest)
2.  {
3.      int fd=open(src,O_WRONLY|O_CREAT);          /*创建文件*/
4.      char buf1[256],buf2[256];
5.      memset(buf1,'a',sizeof(buf1));              /*把 a 写入 buf1*/
6.      memset(buf2,'b',sizeof(buf2));              /*把 b 写入 buf1*/
7.      for (int i=0; i<100000000; i++) {
8.          lseek(fd,0,SEEK_SET);                   /*设置文件指针位置为 0*/
9.          if (i%2){
10.             write (fd,buf1,sizeof(buf1));       /*数据写入文件*/
11.         } else {
12.             write (fd,buf2,sizeof(buf2));       /*数据写入文件*/
13.         }
14.         fsync(fd);                              /*清空缓存,写入持久化存储设备*/
15.     }
16.     close(fd);                                  /*关闭文件*/
17. }
```

- 攻击方式 5:利用文件重命名操作执行磨损攻击.

文件重命名需要完成3个操作:1) 在目标文件的父目录增加一条目录项;2) 在源文件父目录删除该文件的目录项;3) 修改索引节点.此外,文件重命名操作还需写日志确保文件的一致性.如表 3 所示,文件重命名操作会造成索引节点、目录文件数据、日志的更新.

因此,攻击方式 5 就是在持久化内存文件系统对某个文件反复执行重命名操作,例子如病毒程序 5 所示:通过一个简单的 for 循环,对一个给定的文件执行 10^8 次文件重命名操作,对文件系统的元数据和目录文件数据存储区造成大量写操作.数据结构的写次数见表 4,索引节点、文件数据和日志的写次数为 10^8,日志是否修改固定

的存储单元与日志管理有关.每执行一次文件重命名操作,需要修改源文件和目标文件的索引节点和目录文件数据.文件数据和索引节点这两个数据结构在病毒程序 5 的操作中从不改变存储位置,而且每次对文件执行重命名操作都会更新,所以病毒程序会在对应的存储单元执行 10^8 写操作,迅速导致其磨损穿.

病毒程序 5. 攻击方式 5 的示例代码.

输入:*src*:源文件路径及名称;*dest*:目标文件路径及名称;

输出:*NULL*.

```
1.  void LoopRenameFile(char *src,char *dest)
2.  {
3.      unsigned int i=0;
4.      for (int i=0; i<100000000; i++){
5.          if (0==i%2){
6.              rename(src,dest);                    /*移动文件*/
7.          } else {
8.              rename(dest,src);                    /*移动文件*/
9.          }
10.         i++;
11.     }
12. }
```

本节探索的 5 种磨损攻击方式涉及到持久化内存文件系统所有的元数据和文件数据,即超级块、索引节点、文件索引结构、文件数据和日志,每种数据都可以通过简单的文件操作进行更新.实验证明:在 PMFS[9],以上 5 种攻击方式发动对 NVM 的磨损攻击,在很短的时间内就能把 NVM 磨损穿.而现有的持久化内存文件系统,无论是 PMFS[9],还是 BPFS[8],NOVA[10],SIMFS[11]和 HiNFS[13]等,都没有考虑 NVM 磨损防御,即都可以通过简单的文件操作在很短的时间内就能把底层的 NVM 磨损穿,严重威胁到文件系统的数据稳定性.

3 磨损防御机制 PFWD

3.1 概 述

从上一节可知,病毒程序可以通过利用不同的文件操作组合出多种针对 NVM 的磨损攻击方式,它们攻击的数据涉及到持久化内存文件系统的所有数据,即超级块、索引节点、文件索引结构、文件数据和日志.为此,本文认为,应从保护被攻击数据的角度防御病毒的恶意磨损攻击.本节据此提出持久化内存文件系统磨损防御机制(persistent in-memory file system wear defense mechanism,简称 PFWD)防御病毒对 NVM 的磨损攻击.

具体而言,PFWD 包括 4 项技术:超级块迁移技术、索引节点元数据虚拟化技术、文件数据页磨损均衡技术和文件索引结构迁移技术,见表 5.其中,文件数据页磨损均衡技术用于防御针对日志和文件数据的攻击.

Table 5 PFWD defense the data under wear attack

表 5 针对被攻击数据的磨损防御机制 PFWD

数据结构	防御技术
超级块	超级块迁移技术
索引节点	索引节点元数据虚拟化技术
文件索引结构	文件索引结构迁移技术
文件数据	文件数据页磨损均衡技术
日志	文件数据页磨损均衡技术

在持久化内存文件系统实现 PFWD,NVM 的物理空间布局如图 1 所示:(1) 超级块指针,指向超级块的存储区.超级块的存储区可动态调整,当超级块存储区的磨损严重时,超级块可迁移到磨损较低的物理区间;(2) 物理页写次数表,记录 NVM 每个物理页的写次数,每次更新物理页的数据,都要在物理页写次数表增加相应的写次

数,因为 PFWD 机制能保证 NVM 物理空间的磨损均衡,所以物理页写次数表的存储区间也是磨损均衡的;(3) 索引节点映射表,记录虚拟索引节点的写次数和偏移量,实现索引节点的迁移;(4) 超级块、索引节点、文件索引结构、文件数据、日志,分散在 NVM 整个物理空间,通过超级块迁移技术、索引节点元数据虚拟化技术、文件索引结构迁移技术、文件数据页磨损均衡技术实现以上 5 种数据结构的存储区的磨损均衡.

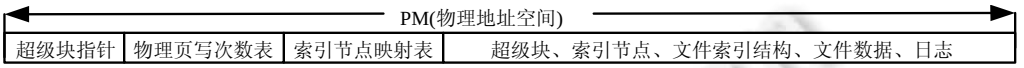

Fig.1 Layout of physical address space of NVM when PFWD is implemented in persistent in-memory file system

图 1 持久化内存文件系统实现 PFWD 时,NVM 的物理地址空间布局

持久化内存文件系统实现 PFWD,物理页写次数表和索引节点映射表的空间开销极小.例如,NVM 的存储空间大小为 10GB,物理页大小为 4KB,每 8 字节记录一个物理页的写次数,则物理页写次数表占用的存储空间是 $\frac{10\text{GB}}{4\text{K}}\times 8\text{B}=20\text{MB}$,仅占总存储空间的 0.2%.通常,所有文件索引节点总的存储空间大小设计为持久化内存文件系统存储空间的 1%,索引节点的大小为 128KB[9−11],4KB 大小的物理页能存储 32 个索引节点.索引节点映射表中每个索引节点的写次数和偏移量共占 8 字节,即分别用 4 字节记录写次数和偏移量,则索引节点映射表所占存储空间的大小是 $\frac{10\text{GB}\times 1\%}{4\text{K}}\times 32\times 8\text{B}=6.4\text{MB}$,仅占总存储空间的 0.06%.所以,物理页写次数表和索引节点映射表的存储空间开销可以忽略不计.此外,为提高检索效率,物理页写次数表和索引节点映射表都使用数组结构,并且两种数据结构采用修改 DRAM 副本的方式更新其写次数,只有在适当的时机回写 NVM,以减少两个数据结构物理存储区的写次数.因此,物理页写次数表和索引节点映射表所在存储区的磨损极低.

3.2 索引节点元数据虚拟化技术

索引节点是文件系统的重要组成部分,每个索引节点拥有唯一的编号(ino),文件系统通过 ino 来检索索引节点.本文把保存单个索引节点的 NVM 物理空间称为“索引节点槽(inode slot)”,文件系统所有的索引节点槽组成“索引节点存储区”.既有的持久化内存文件系统,ino 和 Inode slot 具有固定的映射关系,即索引节点存放在 NVM 的固定位置.例如:NOVA[10],SIMFS[11]和 SCMFS[12]采用数组结构保存所有的索引节点,即在 NVM 划分一段连续的物理空间存储所有的索引节点;BPFS[8],PMFS[9]和 HiNFS[13]虽然通过树形结构把索引节点分散存放在 NVM 的物理空间,但是一经使用就不再改变其存储位置.所以,文件系统对索引节点的所有写操作都作用于固定的存储单元.这种固定映射关系的索引节点存储区是无法抵御恶意磨损攻击,病毒程序通过简单的文件操作就能把索引节点存储单元磨损穿.为此,本文提出一种高效的索引节点元数据虚拟化技术,使得文件的索引节点在其整个生命周期内能动态迁移.该技术的核心思想:如果保存某个索引节点的 Inode slot 的磨损次数达到迁移阈值,则把该索引节点迁移到磨损较低的 Inode slot,通过动态调整索引节点的物理位置,使得对单个索引节点集中的写操作分散到整个索引节点存储区,实现索引节点存储区的磨损均衡.

为实现索引节点迁移,首先要能改变 ino 到 Inode slot 的映射关系,即改变索引节点的存储位置.为此,索引节点元数据虚拟化技术提出在文件系统初始化时,在内核预留一段连续的虚拟地址空间作为“索引节点虚拟地址空间”,同时申请多个 NVM 物理页作为“索引节点存储区”.索引节点虚拟地址空间被分割为多个大小相等的虚拟索引节点(virtual inode,简称 vInode),索引节点存储区则被划分为多个大小相等的 Inode slot.索引节点虚拟地址空间的大小等于索引节点存储区的物理空间大小,单个 vInode 所占的虚拟地址空间大小也等于单个 Inode slot 的物理空间大小,即 vInode 的总数等于 Inode slot 的总数.如图 2 所示,vInode 通过页表(page table)和 Inode slot 一一对应,该映射关系所采用的格式与进程页表的格式相同,利用 CPU 既有的硬件 MMU(memory management unit)完成虚拟地址到物理地址的转换.为了实现索引节点的迁移,索引节点元数据虚拟化技术提出在 NVM 新增数据结构“索引节点映射表”,索引节点映射表用来记录 vInode 的偏移量(offset).偏移量的总数等于文件系统索引节点的总数,即等于 vInode 和 Inode slot 的总数.偏移量的计算,具体而言,vInode 的起始地址减去

索引节点虚拟地址空间的起始地址是一个固定的差值.该差值是 vInode 大小的整数倍,用该差值对 vInode 的大小求余,结果记为 vInode 偏移量(以下简称偏移量).因此,索引节点的检索流程,例如查询索引节点 3,则从索引节点映射表中得到偏移量,用偏移量乘以 vInode 的大小,再加上索引节点虚拟地址空间的基址,得到 vInode 的虚拟地址首地址,利用 MMU 完成虚拟地址到物理地址的转换,最终得到索引节点 3 的实际数据.通过索引节点元数据虚拟化技术,实现 ino 和 Inode slot 动态解绑,使得修改索引节点映射表的偏移量就可以改变 ino 和 Inode slot 的映射关系,即可实现索引节点的迁移.

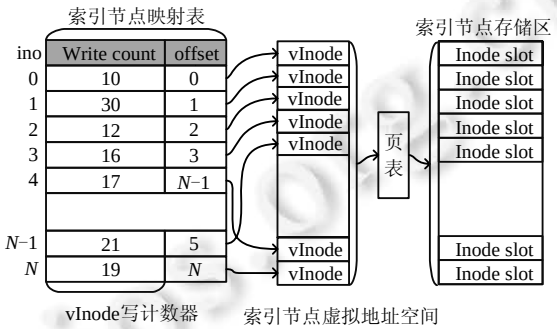

Fig.2 Inode virtualization

图 2 索引节点元数据虚拟化

为了将索引节点迁移到写次数较少的 Inode slot,则每次更新索引节点,则需要记录 Inode slot 的写次数.由于 vInode 和 Inode slot 是一一映射关系,因此,Inode slot 的写次数也是 vInode 的写次数,如图 2 所示:为每个 vInode 设置一个写计数器(write count)器(以下简称写计数器),用于记录 vInode 的写次数,写计数器保存在索引节点映射表里.ino、偏移量和写计数器在索引节点映射表是一一对应关系,具体对应关系如图 2 所示.

索引节点元数据虚拟化技术使得索引节点的更新操作均匀分布到索引节点存储区,即:每次更新索引节点,需要增加该索引节点对应的 vInode 的写计数器.如果 vInode 的写次数达到设定的迁移阈值 W_p,则查找 vInode 写计数器,把该索引节点数据拷贝到一个空闲的索引节点的 Inode slot;然后把该索引节点对应的 vInode 的计数器清零;最后,在索引节点映射表修改偏移量,实现索引节点的动态迁移.上述迁移方法可能产生一种极端情况,即索引节点在两个 Inode slot 来回迁移.例如:索引节点 *A* 对应 vInode *A*,vInode *A* 对应 Inode slot *A*,索引节点 *B* 对应 vInode *B*,vInode *B* 对应 Inode slot *B* 且索引节点 *B* 处于空闲状态,当 vInode *A* 的写次数达到迁移阈值,则索引节点 *A* 迁移到索引节点 *B* 的物理存储空间 Inode slot *B*,使得索引节点 *A* 对应 vInode *B*,索引节点 *B* 对应 vInode *A*;同时,重置 vInode *A* 的写次数为 0 和索引节点 *B* 处于空闲状态,继续写索引节点 *A*,当写次数再次达到迁移阈值,索引节点 *A* 的数据再次迁移到索引节点 *B* 的物理存储空间 Inode slot *A*.因此,持续修改索引节点 *A*,导致索引节点 *A* 在 Inode slot *A* 和 Inode slot *B* 来回迁移,迅速把这两个存储区间磨损穿.为解决上述问题并提高索引节点的管理效率,本文提出使用链表实现队列(queue)来管理空闲索引节点,以下简称索引节点空闲链表.索引节点的申请和释放规则:总是从索引节点空闲链表的尾部获取索引节点,而把释放的索引节点插入到索引节点空闲链表头部,从而避免短期内重复使用最近写过的索引节点.鉴于索引节点更新频次高,导致索引节点存储区的磨损较为严重,为达到 NVM 整个存储空间的磨损均衡,本文提出索引节点以页大小为粒度的迁移方法,即:每次更新索引节点,在物理页写次数表增加其 Inode slot 所在物理页的写次数,如果该物理页的写次数达到迁移阈值 $W_q(W_q>>W_p)$,则把该页所有的索引节点迁移到写次数较少的空闲页.

索引节点的迁移流程如图 3 所示.图 3(a)为迁移前的索引节点映射表,此时,索引节点 1 对应的 vInode 的写次数为 199,索引节点 3 对应的 vInode 的写次数为 0 且处于空闲状态;同时,该索引节点位于索引节点空闲链表的表尾.如果索引节点迁移的阈值此时为 200.更新索引节点 1,则做索引节点迁移.首先,把索引节点 1 的数据拷贝到索引节点 3 的 Inode slot;然后增加索引节点 3 对应的 vInode 的写计数器,索引节点 1 对应的 vInode 的写计数器清零;最后修改索引节点映射表中索引节点 1 的偏移量为 3 和索引节点 3 的偏移量为 1,即完成索引节点的

迁移.如图 3(a):ino 为 1 和 3 的偏移量分别为 1 和 3,偏移量为 1 和 3 的 vInode 的写次数分别为 199 和 0.迁移完成后的索引节点映射表如图 3(b)所示,此时,ino 为 1 和 3 的偏移量分别为 3 和 1,偏移量为 1 和 3 的 vInode 的写次数分别为 0 和 1.

ino	Write count	offset
0	150	0
1	199	1
2	30	2
3	0	3
4	35	N−1
N−1	0	4
N	30	N

(a) 迁移前

ino	Write count	offset
0	150	0
1	1	3
2	30	2
3	0	1
4	35	N−1
N−1	0	4
N	30	N

(b) 迁移后

Fig.3 Example of inode migration

图 3 索引节点迁移示例

索引节点元数据虚拟化技术使用页表来实现 vInode 到 Inode slot 的映射,直接利用 CPU 的 MMU 完成虚拟地址到物理地址的转换,最大程度保证索引节点检索的性能.索引节点的迁移对用户透明,只需把索引节点数据迁移到空闲且磨损少的 Inode slot、修改索引节点映射表的偏移量和修改 vInode 的计数器值,就能实现索引节点迁移,对其他正在使用的文件没有影响.通过索引节点元数据虚拟化技术,实现索引节点的动态迁移,使得对索引节点的写操作可以分散到不同的 Inode slot,实现索引节点存储区的磨损均衡,能有效地防御病毒对索引节点存储单元的磨损攻击.

3.3 超级块迁移技术

既有持久化内存文件系统的设计,例如 BPFS[8],PMFS[9],NOVA[10],SIMFS[11]和 HiNFS[13]等,超级块保存在 NVM 的固定位置,不能移动,大多数文件操作都需要修改超级块.例如,物理页和索引节点的分配和释放都要修改超级块的空闲页数和空闲索引节点,所以超级块存储区的磨损极为严重.而所有的持久化内存文件系统都没有对超级块存储区做磨损保护,病毒程序通过简单的文件操作就能把超级块存储单元磨损穿.所以,现有的持久化内存文件系统的设计是无法防御病毒发动对超级块存储单元的磨损攻击.

为了有效防御病毒程序发动对超级块存储单元的磨损攻击,本文提出超级块迁移技术.如图 4 所示:设置一个超级块指针,即把 NVM 物理空间的前 8 个字节作为超级块指针,用来保存超级块存储区的首地址,超级块可以迁移,当超级块存储区的写次数达到迁移阈值 W_q,则把超级块迁移到磨损低的物理页.

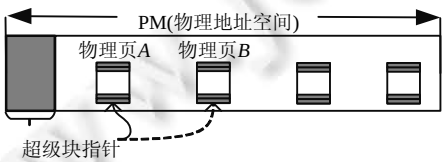

Fig.4 Example of superblock migration

图 4 超级块迁移示例

如图 4 所示:超级块保存在物理页 A,当更新超级块时,判断物理页 A 的写次数是否达到阈值 W_q.如果没有达到阈值,则修改超级块并且增加物理页 A 的写次数;否则,把超级块迁移到磨损低且空闲的物理页 B,修改超级块并且增加物理页 B 的写次数,然后修改超级块指针指向物理页 B,最后释放物理页 A,即完成超级块迁移.

超级块迁移技术实现超级块动态迁移,使得集中对超级块存储区的写操作分散到整个 NVM 存储空间,能有效地防御针对超级块存储单元的磨损攻击.

3.4 文件数据页磨损均衡技术

实现数据一致性是文件系统的基本功能,预写日志[33−35]和写时复制[8,11]是实现数据一致性的两种不同机制.文件的更新和实现数据一致性的机制息息相关,现有的持久化内存文件系统采用写时复制或者写时复制结合预写日志机制实现数据一致性.BPFS[8],NOVA[10],SIMFS[11]采用写时复制机制实现文件数据一致性.对于文件覆盖写,写时复制机制是把数据写到新的存储位置,然后修改文件索引结构,持续地修改文件数据,难以在短时间内把文件数据页磨损穿.但是,频繁地更新文件索引结构,导致文件索引结构存储区间迅速被磨损穿(见第3.5节).Shortcut-JFS[16]采用自适应日志(adaptive logging,简称AL),即预写日志结合写时复制机制实现数据一致性.持久化内存文件系统采用预写日志机制实现数据一致性或没有实现数据一致性,文件覆盖写是直接修改文件数据的原存储位置.因此,持续修改文件的数据,则迅速写穿文件数据存储空间.持久化内存文件系统无论是支持文件元数据一致性,还是支持数据一致性(备注:文件元数据一致性仅需保证文件元数据一致性状态,数据一致性需要保证文件元数据和文件数据一致性状态),都需要备份文件元数据信息.PMFS[9]仅支持元数据一致性,通过在 NVM 划分一段连续的存储区间(PMFS-log)来保存文件元数据信息,重复执行简单的文件操作,很快就能把 PMFS-log 存储空间写穿.NOVA[10]的每个索引节点都有一个私有日志,称为索引节点私有日志,同时为每个 CPU 划分一个日志区间,简称索引节点公有日志.NOVA 的索引节点私有日志和公有日志的存储区间是固定的,只有当存储空间容量不够,才会申请其他物理页来扩充容量.因此,持续对文件做简单的文件操作,在段时间内就能把索引节点的私有日志或者公有日志的存储单元磨损穿.

为了有效地防御病毒程序发动对文件数据页和日志存储单元的磨损攻击,本文提出了文件数据页磨损均衡技术.如图 5 所示,日志由日志元数据和日志文件数据组成.日志元数据主要包括文件元数据、日志状态、修改起始位置、文件修改大小和日志文件指针等,其中:文件元数据记录被修改索引节点的基本信息;日志状态包括初始化(initial)、提交(commit)、CheckPoint;日志文件指针指向日志文件.每个打开的文件都为其分配一个私有日志.如果文件操作不涉及文件数据的修改,则不使用日志文件;如果文件操作涉及到文件数据的修改,则使用日志文件.

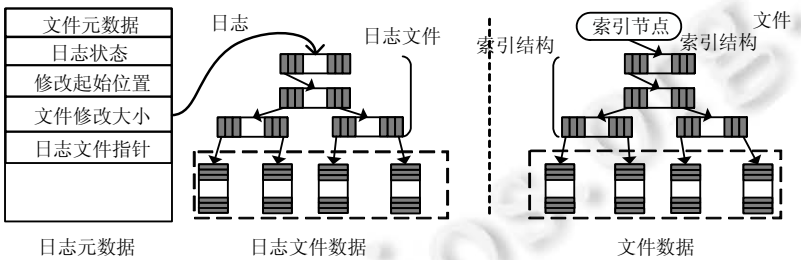

Fig.5 Wear-leveling technique of file data page

图 5 文件数据页磨损均衡技术

文件数据页磨损均衡技术采用写时复制机制实现数据一致性,即:把更新数据写入日志文件,通过交换文件数据页和日志文件数据页来实现文件的覆盖写.首先把索引节点被修改信息、文件被替换数据页的指针、修改的起始位置、修改大小等数据写入日志元数据;然后把更新数据写入日志文件;最后修改索引节点及交换日志文件数据页和文件被修改数据页,即修改日志文件索引结构和文件索引结构.对文件执行数据追加(append write)操作,不使用日志文件,首先从空间管理模块申请磨损低的物理页,写入追加数据,在物理页写次数表增加物理页的写次数;然后把本次修改的元数据信息写入日志元数据;最后修改文件索引节点,把新增物理页的指针写入文件索引结构.当开始修改文件索引节点和文件数据时,设置日志状态为提交状态;当所有的修改数据都持久化到 NVM,设置日志为 CheckPoint 状态,即修改完成.文件执行覆盖写操作,首先判断日志文件数据页的写次数是否达到迁移阈值:如果达到迁移阈值,则申请磨损低的物理页替换该日志文件数据页,然后把数据写入新申请的物理页,在物理页写次数表增加写次数;否则,把数据写入日志文件,增加物理页的写次数.日志元数据的迁

移方法同上.每次写日志元数据,需在物理页写次数表增加写次数,然后判断其写次数是否达到迁移阈值:如果达到迁移阈值,则把日志元数据迁移到磨损低的存储空间.当日志元数据迁移完成后,修改文件索引节点指向日志元数据新的存储区间,释放原存储区间.

本文提出的文件数据页磨损均衡技术,实现日志数据的动态迁移,对文件执行覆盖写操作,都是使用日志文件数据页替换文件被修改数据页.每次数据写入日志文件数据页,都判断其写次数是否达到迁移阈值,达到则做数据迁移.对文件执行数据追加操作,都是从空间管理模块申请磨损低的物理页来保存追加数据,通过文件数据页磨损均衡技术,使得对日志和文件数据的更新操作分散到整个 NVM 存储空间,实现文件数据及日志存储区的磨损均衡,能有效地防御针对文件数据和日志的存储单元的磨损攻击.

3.5 文件索引结构迁移技术

文件索引结构的修改也和文件系统实现数据一致性机制有关,本文第 3.4 节已经介绍了,现有持久化内存文件系统都采用了写时复制[8,10]机制或者写时复制结合预写日志[29−31]机制来实现数据一致性.本文提出的文件数据页磨损均衡技术也是采用写时复制机制实现数据一致性.现有持久化内存文件系统,文件被创建后,文件索引结构就不会被迁移,即:现有的持久化内存文件系统都没有对文件索引结构存储区做磨损保护,病毒程序通过简单的文件操作就能把文件索引结构的存储单元磨损穿.所以,现有的持久化内存文件系统的设计是无法防御病毒发动对文件索引结构存储单元的磨损攻击.

为了有效地防御病毒程序发动对文件索引结构存储单元的磨损攻击,本文提出了文件索引结构迁移技术,如图 6 所示,每次更新文件索引结构的索引项,都判断该索引项所在物理页的写次数是否达到迁移阈值 W_q:如果没有达到迁移阈值,则直接修改索引项;如果达到迁移阈值,则拷贝该索引项所在物理页的所有索引项到空闲且磨损低的物理页,修改索引项.然后修改上一级索引项,再判断上一级索引项所在的物理页的写次数是否达到迁移阈值 W_q:如果没有达到迁移阈值,则本次文件索引结构迁移完成;如果达到迁移阈值,则对上一级文件索引结构做迁移.按此迭代操作.

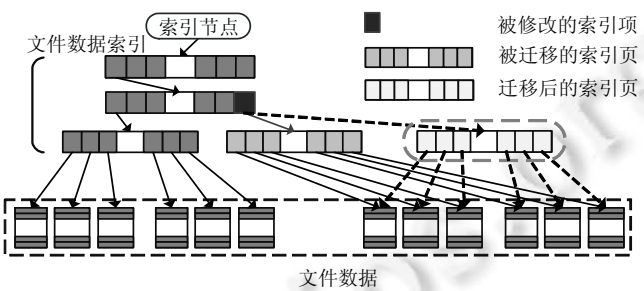

Fig.6 File index structure migration

图 6 文件索引结构迁移

本文提出的文件索引结构迁移技术同样适用于本文第 3.4 节的日志文件索引结构.通过对文件索引结构做迁移操作,使得文件索引结构更新操作分散到整个 NVM 存储空间,能有效地防御针对文件索引结构的存储单元的磨损攻击.

4 实验结果与分析

本节验证提出的 PFWD 技术对恶意磨损攻击的防御效果及性能开销.

4.1 实验配置

为进行验证,本文基于 PFWD 技术实现一个带磨损保护的持久化内存文件系统(wear protected persistent in-memory file system,简称 WPFS).对比的对象是近年来具有代表性的新型持久化内存文件系统 PMFS[9].为了

记录文件系统物理页的写次数,实验在 PMFS[9]的源代码中加入物理页的磨损计数器,计数器自身的写操作不记录在文件系统页面的写次数中.本文采用第 2 节提出的 5 种磨损攻击病毒程序分别攻击 WPFS 和 PMFS[9],分析 WPFS 和 PMFS 的 NVM 存储空间的磨损情况;对比 WPFS 和 WPFS-NoPFWD(WPFS 未实现 PFWD)的读写性能,分析持久化内存文件系统增加 PFWD 的开销.实验结果证明,PFWD 能有效地防御病毒程序发动的磨损攻击.

本文的实验平台配置有 3.20GHz Intel i5-4460 处理器,8GB 的 DRAM 内存,操作系统为 Ubuntu 15.04,Linux 内核是 4.4.30.本实验中使用 4GB 的 DRAM 内存模拟 NVM.为更好地展示 WPFS 防御磨损攻击的效果,在磨损效果实验中,WPFS 仅使用 40MB 的 NVM 存储空间,即物理页的大小为 4KB,总共 10240 个物理页.在性能开销实验中, WPFS 和 WPFS- NoPFWD 的 NVM 存储空间为 4GB 大小.

4.2 NVM磨损分析

本节对比分析 WPFS 和 PMFS[9]两种文件系统在本文提出的 5 种磨损攻击下的 NVM 磨损情况.实验结果如图 7~图 12 所示.

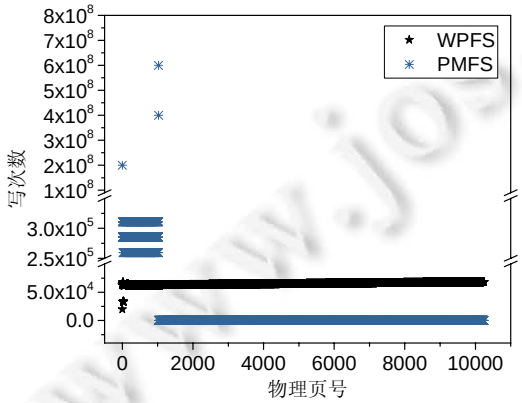

Fig.7 NVM wear under attack 1

图 7 攻击方式 1 的攻击下 NVM 的磨损情况

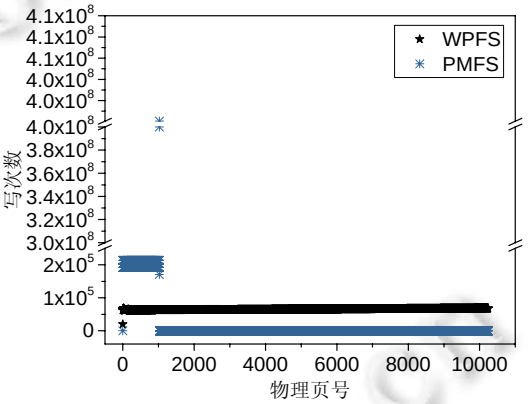

Fig.8 NVM wear under attack 2

图 8 攻击方式 2 的攻击下 NVM 的磨损情况

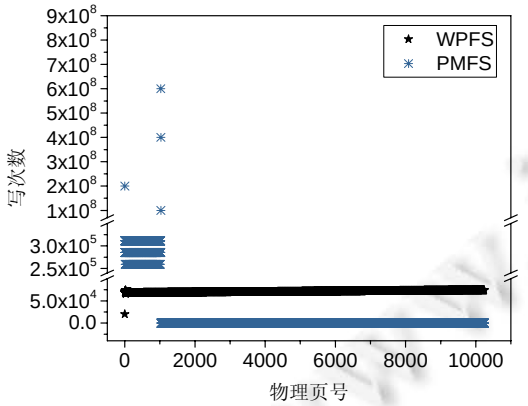

Fig.9 NVM wear under attack 3

图 9 攻击方式 3 的攻击下 NVM 的磨损情况

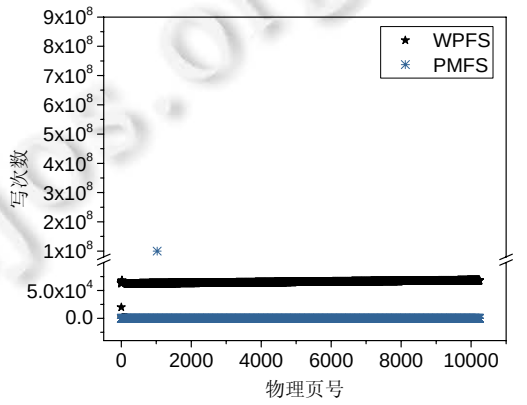

Fig. 10 NVM wear under attack 4

图 10 攻击方式 4 的攻击下 NVM 的磨损情况

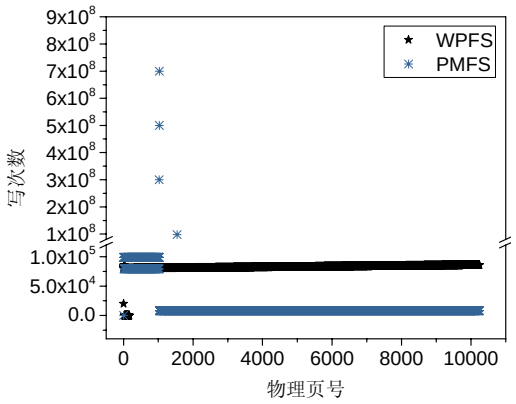

Fig.11 NVM wear under attack 5

图 11 攻击方式 5 的攻击下 NVM 的磨损情况

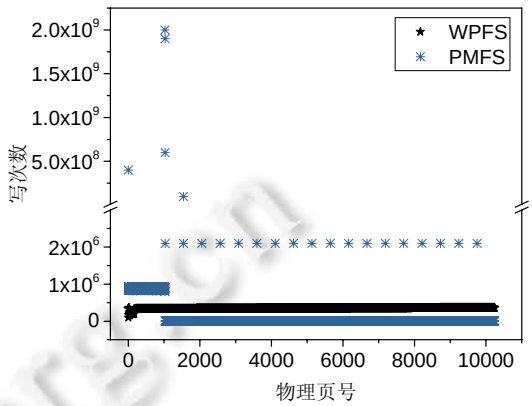

Fig.12 NVM wear under all attacks mode proposed in this paper

图 12 本文探索出的 5 种攻击方式的攻击下 NVM 的磨损情况

PMFS 在本文提出的 5 种攻击方式的磨损攻击下,耗时最长的是病毒程序 1,即通过一个简单的 for 循环,在一个给定的目录下执行 10^8 次创建文件和删除文件操作,在 13 分钟内就能把底层 NVM 存储设备磨损穿,NVM 总的写次数为 1.5×10^9,但是使用 PFWD 技术的 WPFS 文件系统却能保护 NVM 承受 21 943 倍于 PMFS 的文件操作/写操作.表 6 分析 WPFS 和 PMFS 所有物理页的最大写次数、标准偏差和变异系数.实验结果表明:PFWD 技术通过把大量集中对少数存储区的写操作分散到整个 NVM 存储空间,实现了 NVM 存储空间的磨损均衡,能有效防御病毒程序通过简单的文件操作就能把底层的 NVM 存储设备磨损穿.

Table 6 Analysis for the wear of NVM under the five proposed wear attacks

表 6 分析 NVM 在提出的 5 种磨损攻击下的磨损情况

	最大写次数			标准偏差			变异系数		
	WPFS	PMFS	PMFS/WPFS	WPFS	PMFS	PMFS/WPFS	WPFS	PMFS	PMFS/WPFS
攻击方式 1	6.8×10^4	6.0×10^8	8 824	1732.5	7 394 591.8	4 268	0.026 6	50.647 9	1 904
攻击方式 2	6.8×10^4	4.0×10^8	5 882	1732.5	5 590 350.8	3 227	0.026 6	57.241 8	2 152
攻击方式 3	7.4×10^4	6.0×10^8	8 108	1732.5	7 460 140.0	4 306	0.024 3	47.821 4	1 968
攻击方式 4	6.8×10^4	1.0×10^8	1 471	1732.5	988 211.8	570	0.026 6	101.192 9	3 804
攻击方式 5	8.6×10^4	7.0×10^8	8 140	1732.5	9 054 293.1	5 226	0.020 8	51.191 4	2 461

由图 7 可知:病毒程序 1 在 PMFS 中运行不到 13 分钟,就对 NVM 的 3 个页面造成超过 10^8 次写操作.假设底层 NVM 存储设备的写耐受度为 10^8[24−27],这 3 个 NVM 存储单元即已被磨损穿.与此同时,其他 90%以上的 NVM 页面的写次数仍然为 0.表 6 中,在 PMFS 和 WPFS 上分别运行病毒程序 1 后,其底层 NVM 物理页的最大写次数分别为 6.0×10^8 和 6.8×10^4,PMFS/WPFS 的倍数高达 8 824;其次,两者的标准偏差分别为 1 732.5 和 7 394 591.8,PMFS/WPFS 的倍数高达 4 268;变异系数分别为 0.026 6 和 50.647 9,PMFS/WPFS 的倍数高达 1 904.病毒程序 1 在目录下反复执行创建文件和删除文件,达到把超级块、索引节点、目录文件数据、文件索引结构和日志的存储区迅速磨损穿.由于 PMFS 没有对以上数据的存储区做磨损保护,尤其是日志区域,更是整个文件系统反复使用一块固定的日志 PMFS-Log[9],所以无法抵御病毒程序的恶意磨损攻击.而 WPFS 考虑到以上数据存储区的磨损保护,把大量集中对少数存储区的写操作分散到整个存储空间,实现了 NVM 的磨损均衡,能够有效防御病毒程序 1 的恶意磨损攻击.

由图 8 可知,病毒程序 2 在 PMFS 中运行不到 9 分钟,就对 NVM 的两个页面造成超过 10^8 次写操作.假设底层 NVM 存储设备的写耐受度为 10^8[24−27],这两个 NVM 磨损单元即已被磨损穿.与此同时,其他 90%以上的 NVM 页面的写次数仍然为 0.表 6 中,病毒程序 2 分别在 PMFS 和 WPFS 运行,NVM 物理页的最大写次数分别为

4.0×10^8 和 6.8×10^4,PMFS/WPFS 的倍数高达 5 882;其次,两者的标准偏差分别为 1 732.5 和 5 590 350.8,PMFS/WPFS 的倍数高达 3 227;变异系数分别为 0.026 6 和 57.241 8,PMFS/WPFS 的倍数高达 2 152.病毒程序 2 对文件反复执行创建硬链接和删除硬链接,达到迅速把索引节点、目录文件数据和日志的存储区磨损穿.由于 PMFS 没有对以上数据的存储区做磨损保护,所以无法抵御病毒程序的恶意磨损攻击.而 WPFS 考虑到以上数据存储区的磨损保护,把大量集中对少数存储区的写操作分散到整个存储空间,实现了 NVM 的磨损均衡,能够有效防御病毒程序 2 的恶意磨损攻击.

由图 9 可知:病毒程序 3 在 PMFS 中运行不到 12 分钟,就对 NVM 的 4 个页面造成超过 10^8 次写操作.假设底层 NVM 存储设备的写耐受度为 $10^{8[24-27]}$,这 4 个 NVM 磨损单元即已被磨损穿.与此同时,其他 90%以上的 NVM 页面的写次数仍然为 0.如表 6 所示:病毒程序 3 分别在 PMFS 和 WPFS 运行,NVM 物理页的最大写次数分别为 6.0×10^8 和 7.4×10^4,PMFS/WPFS 的倍数高达 8 108;其次,两者的标准偏差分别为 1 732.5 和 7 460 140.0,PMFS/WPFS 的倍数高达 4 306;变异系数分别为 0.024 3 和 47.821 4,PMFS/WPFS 的倍数高达 1 968.病毒程序 3 对文件反复执行创建软链接和删除软链接,达到迅速把超级块、索引节点、目录文件数据和日志的存储区磨损穿.由于 PMFS 没有对以上数据存储区做磨损保护,所以无法抵御病毒程序的恶意磨损攻击.而 WPFS 考虑到以上数据存储区的磨损保护,把大量集中对少数存储区的写操作分散到整个存储空间,实现了 NVM 的磨损均衡,能够有效防御病毒程序 3 的恶意磨损攻击.

由图 10 可知:病毒程序 4 在 PMFS 中运行不到 2 分钟,就对 NVM 的一个页面造成超过 10^8 次写操作.假设底层 NVM 存储设备的写耐受度为 $10^{8[24-27]}$,这一个 NVM 磨损单元即已被磨损穿.与此同时,其他 99.99%以上的 NVM 页面的写次数仍然为 0.如表 6 所示:病毒程序 4 分别在 PMFS 和 WPFS 运行,NVM 物理页的最大写次数分别为 1.0×10^8 和 6.8×10^4,PMFS/WPFS 的倍数高达 1 471;其次,两者的标准偏差分别为 1 732.5 和 988 211.8,PMFS/WPFS 的倍数高达 570;变异系数分别为 0.026 6 和 101.192 9,PMFS/WPFS 的倍数高达 3 804.病毒程序 4 反复对文件数据的某个区间执行覆盖写操作,达到迅速把超级块、文件数据、文件索引结构的存储区磨损穿.由于 PMFS 没有对以上数据的存储区做磨损保护,所以无法抵御病毒程序的恶意磨损攻击.而 WPFS 考虑到以上数据存储区的磨损保护,把大量集中对少数存储区的写操作分散到整个存储空间,实现了 NVM 的磨损均衡,能够有效防御病毒程序 4 的恶意磨损攻击.

由图 11 可知:病毒程序 5 在 PMFS 中运行不到 2 分钟,就对 NVM 的 4 个页面造成超过 10^8 次写操作.假设底层 NVM 存储设备的写耐受度为 $10^{8[24-27]}$,这 4 个 NVM 磨损单元即已被磨损穿.与此同时,其他 90%以上的 NVM 页面的写次数仍然为 0.如表 6 所示:病毒程序 5 分别在 PMFS 和 WPFS 运行,NVM 物理页的最大写次数分别为 7.0×10^8 和 8.6×10^4,PMFS/WPFS 的倍数高达 8 140;其次,两者的标准偏差分别为 1 732.5 和 9 054 293.1,PMFS/WPFS 的倍数高达 5 226;变异系数分别为 0.020 8 和 51.191 4,PMFS/WPFS 的倍数高达 2 461.病毒程序 5 对文件反复执行重命名操作,达到迅速把索引节点、目录文件数据和日志的存储区磨损穿.由于 PMFS 没有对以上存储区做磨损保护,所以无法抵御病毒程序的恶意磨损攻击.而 WPFS 考虑到以上数据存储区的磨损保护,把大量集中对少数存储区的写操作分散到整个存储空间,实现了 NVM 的磨损均衡,能够有效防御病毒程序 5 的恶意磨损攻击.

由图 12 可知:在本文探索出的 5 种病毒程序共同发动对 NVM 的磨损攻击,在 PMFS 运行,就对 NVM 的 5 个页面造成超过 10^8 次写操作.假设底层 NVM 存储设备的写耐受度为 $10^{8[24-27]}$,这 5 个 NVM 磨损单元即已被磨损穿;还造成 NVM 的 18 个页面超过 8.6×10^6,接近 NVM 存储设备的写耐受度.与此同时,其他 90%以上的 NVM 页面的写次数仍然为 0.但是在 WPFS 运行,NVM 物理页的最大写次数为 364 095.由于 PMFS[9]没有对文件系统元数据和文件数据的存储区做磨损保护,所以无法抵御病毒程序的恶意磨损攻击.而 WPFS 考虑到 NVM 的磨损保护,把大量集中对少数存储区的写操作分散到整个存储空间,实现了 NVM 的磨损均衡,能够有效防御病毒程序的恶意磨损攻击.

每次对物理页数据和索引节点的更新,写次数并不是立即写回物理页写次数表和索引节点映射表,而是通过更新 DRAM 副本的方式写回,因此,这两个数据结构的存储区的写次数较少.所以表 6 中,WPFS 没有统计这些

物理页的写次数;因为攻击方式 4 和攻击方式 5 没有攻击索引节点表存储区,所以这两种攻击方式 WPFS 也没有统计索引节点表的物理页的写次数.由表 6 可知:由于 PMFS[9]并没有对元数据和文件数据进行磨损保护,本文探索的 5 种病毒程序都是通过简单的文件操作,在 PMFS[9]文件系统,大量写操作集中在少数的存储区,导致 NVM 很快就被磨损穿;但是 WPFS 考虑了 NVM 的磨损保护,把大量集中对少数存储区的写操作分散到 NVM 的整个存储空间,实现 NVM 的磨损均衡,能有效防御病毒程序发动对 NVM 的磨损攻击.

图 13 表示在病毒程序的磨损攻击下,持久化内存文件系统是否采用本文提出的 PFWD 技术时 NVM 的磨损对比,横坐标表示 NVM 总的写次数,纵坐标表示 NVM 以页为粒度的最大写次数.如果持久化内存文件系统未使用 PFWD 技术,NVM 很快就会被磨损穿;如果持久化内存文件系统使用 PFWD 技术,则 NVM 所有物理页的最大写次数仅仅为 9 766,远远未达到底层 NVM 存储设备的写耐受度 10^8[24–27].在 WPFS 文件系统,病毒程序要使得 NVM 被磨损穿,NVM 总的写次数几乎要达到 10240×10^8 次,NVM 物理页的写次数都接近 10^8,使得 NVM 总的写次数提高了 10 240 倍,而且该倍数随着 NVM 存储空间的增大而增大.

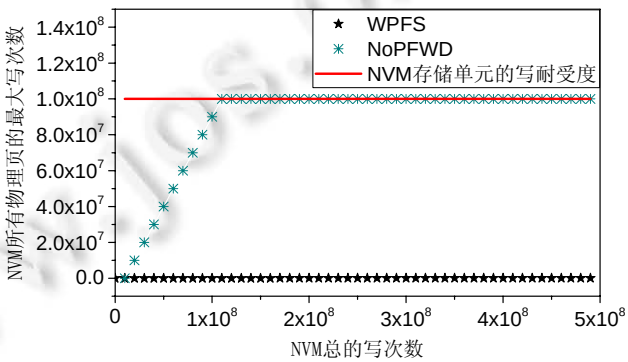

Fig.13 Comparison of NVM wear in WPFS and persistent in-memory file system when PFWD is not used under wear attacks

图 13 对比 WPFS 和持久化内存文件系统未采用 PFWD 技术时,NVM 在磨损攻击下的磨损情况

4.3 性能实验对比

为了评估 PFWD 机制的性能开销,实验中使用 FIO[36]测试工具对比测试 WPFS-NoPFWD 和 WPFS 在不同块大小的读写性能,测试结果如图 14~图 17 所示,图中横坐标为读写块大小分别 1K,2K,4K,8K,16K,32K,64K,128K,256K,512K,纵坐标表示文件读写吞吐率.图 14、图 15 分别表示随机读写的性能对比,图 16、图 17 分别表示顺序读写的性能对比.

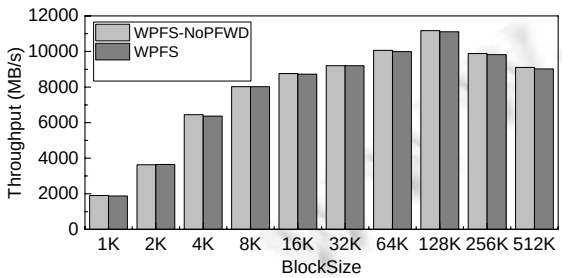

Fig.14 Random read

图 14 随机读

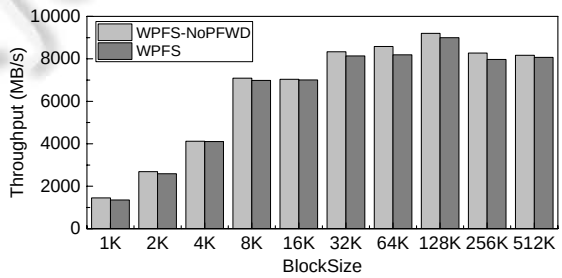

Fig.15 Random write

图 15 随机写

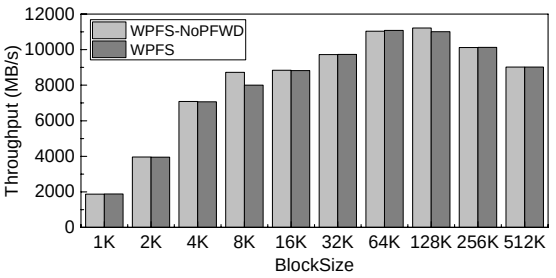

Fig.16 Sequential read

图 16 顺序读

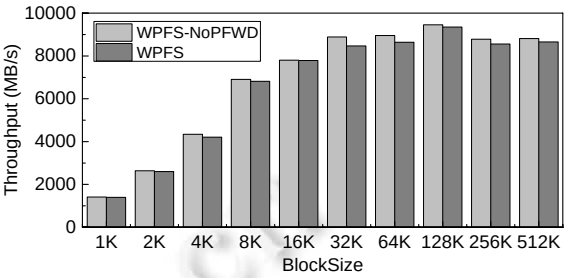

Fig.17 Sequential write

图 17 顺序写

以上性能实验结果显示:对于顺序读和随机读操作,WPFS 没有明显的性能下降.这是因为读操作不会触发磨损均衡操作.对于顺序写和随机写操作,采用 PFWD 技术的 WPFS 会有 5%左右的性能损失.原因是 WPFS 需要在写流程中更新物理页写次数表和索引节点映射表,并实施超级块、文件数据、文件索引结构和索引节点的迁移操作.总体而言,PFWD 的开销极小.

5 结 论

本文首次探索多种借助持久化内存文件系统的文件操作对 NVM 造成恶意磨损的攻击方式.提出持久化内存文件系统磨损防御技术 PFWD.PFWD 技术通过把大量集中对少数存储区的写操作分散到整个 NVM 存储空间,实现了 NVM 存储空间的磨损均衡,能有效防御病毒程序通过简单的文件操作就能把底层的 NVM 存储设备磨损穿.即使 NVM 存储设备在硬件层已经实现了 NVM 的磨损均衡,PFWD 技术在持久化内存文件系统层通过避免集中的写操作,使得写操作分散到整个 NVM 存储空间,能使得硬件层的磨损均衡算法减少数据迁移的开销,提高存储系统的吞吐率.

References:

[1] Joshi M, Zhang WY, Li T. Mercury: A fast and energy-efficient multi-level cell based phase change memory system. In: Proc. of the IEEE Int'l Symp. on High Performance Computer Architecture. 2011. 345−356.

[2] Luo L, Liu Y, Qian DP. Survey on in-memory computing technology. Ruan Jian Xue Bao/Journal of Software, 2016,27(8): 2147−2167 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5103.htm [doi: 10.13328/j.cnki.jos.005103]

[3] Shen ZR, Xue W, Shu JW. Research on new non-volatile storage. Journal of Computer Research & Development, 2014,51(2): 445−453 (in Chinese with English abstract).

[4] Zhou P, Zhao B, Yang J, Zhang YT. A durable and energy efficient main memory using phase change memory technology. ACM SIGARCH Computer Architecture News, 2009,37(3):14−23.

[5] Intel and Micron. Intel and micron produce breakthrough memory technology. 2015. https://newsroom.intel.com/news-releases/

[6] Sha EHM, Chen XZ, Ma DL, Zhuge QF. Design of persistent embedded main memory databases on non-volatile memory. Ruan Jian Xue Bao/Journal of Software, 2016,27(Suppl.(2)):320−327 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16046.htm

[7] Pan W, Li ZH, Du HT, Zhou CC, Su J. State-of-the-Art survey of transaction processing in non-volatile memory environments. Ruan Jian Xue Bao/Journal of Software, 2017,28(1):59−83 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5141.htm [doi: 10.13328/j.cnki.jos.005141]

[8] Condit J, Nightingale EB, Frost C, Ipek E, Lee B, Burger D, Coetzee D. Better I/O through byte-addressable, persistent memory. In: Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles (SOSP 2009). New York: ACM, 2009. 133−146.

[9] Condit J, Nightingale EB, Frost C, Ipek E, Lee B, Burger D, Coetzee D. System software for persistent memory. In: Proc. of the European Conf. on Computer Systems. ACM, 2014. 1−15.

[10] Xu J, Swanson S. Nova: A log-structured file system for hybrid volatile/non-volatile main memories. In: Proc. of the 14th USENIX Conf. on Fileand Storage Technologies (FAST 2016). Santa Clara: USENIX Association, 2016. 323−338.

[11] Sha EHM, Chen XZ, Zhuge QF, Shi L, Jiang WW. A new design of in-memory file system based on file virtual address framework. IEEE Trans. on Computers, 2016,65(10):2959−2972.

[12] Wu XJ, Reddy ALN. SCMFS: A file system for storage class memory. In: Proc. of the High Performance Computing, Networking, Storage and Analysis. IEEE, 2011. 1−11.

[13] Ou JX, Shu JW, Lu YY. A high performance file system for non-volatile main memory. In: Proc. of the 11th European Conf. on Computer Systems (EuroSys 2016). New York: ACM, 2016. [doi: https://doi.org/10.1145/2901318.2901324]

[14] Kwon JH, Kim DG, Jo SG. Memory system and wear leveling method thereof. Patant: US8108592, 2012.

[15] Hu JT, Zhuge QF, Xue CJ, Tseng WC, Sha EHM. Software enabled wear-leveling for hybrid PCM main memory on embedded systems. In: Proc. of the Conf. on Design, Automation and Test in Europe (DATE 2013). San Jose: EDA Consortium, 2013. 599−602.

[16] Lee EJ, Yoo SH, Jang JE, Bahn HY. Shortcut-JFS: A write efficient journaling file system for phase change memory. In: Proc. of the 2012 IEEE 28th Symp. on Mass Storage Systems and Technologies (MSST). 2012. 1−6.

[17] Yun JS, Lee SG, Yoo SJ. Dynamic wear leveling for phase-change memories with endurance variations. IEEE Trans. on Very Large Scale Integration Systems, 2015,23(9):1604−1615.

[18] Hu JT, Xie MM, Pan C, Xue CJ, Zhuge QF, Sha EHM. Low overhead software wear leveling for hybrid PCM+DRAM main memory on embedded systems. IEEE Trans. on Very Large Scale Integration Systems, 2015,23(4):654−663.

[19] Yun JS, Lee SG, Yoo SJ. Bloom filter-based dynamic wear leveling for phase-change RAM. In: Proc. of the Design, Automation & Test in Europe Conf. & Exhibition. IEEE, 2012. 1513−1518.

[20] Huang FT, Feng D, Hua Y, Zhou W. A wear-leveling-aware counter mode for data encryption in non-volatile memories. In: Proc. of the Design, Automation & Test in Europe Conf. & Exhibition. IEEE, 2017. 910−913.

[21] Chen XZ, Sha EHM, Jiang WW, Zhuge QF, Chen JX, Qin JJ, Zeng YS. The design of an efficient swap mechanism for hybrid DRAM-NVM systems. In: Proc. of the 13th Int'l Conf. on Embedded Software (EMSOFT 2016). New York: ACM, 2016. 10.

[22] Zhang X, Sun GY. Toss-Up wear leveling: Protecting phase-change memories from inconsistent write patterns. In: Proc. of the 54th Annual Design Automation Conf. (DAC 2017). New York: ACM, 2017.

[23] Wong HSP, Raoux S, Kim SB, Liang J, Reifenberg JP, Rajendran B, Asheghi M, Goodson KE. Phase change memory. Proc. of IEEE, 2010,98(12):2201−2227.

[24] Chang HS, Chang YH, Hsiu PC, Kuo TW, Li HP. Marching-Based wear-leveling for PCM-based storage systems. ACM Trans. on Design Automation of Electronic Systems, 2015,20(2):1−22.

[25] Liu D, Wang TZ, Wang Y, Shao ZL, Zhuge QF, Sha EHM. Application-Specific wear leveling for extending lifetime of phase change memory in embedded systems. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2015,33(10): 1450−1462.

[26] Zhang YY, Swanson S. A study of application performance with non-volatile main memory. In: Proc. of the Symp. on Mass Storage Systems & Technologies. IEEE, 2015. 1−10.

[27] Liu D, Lin Y, Huang P, Zhu X, Liang L. Durable and energy efficient in-memory frequent-pattern mining. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2017,36(12):2003−2016.

[28] Zeng KS, Lu YY, Wan H, Shu JW. Efficient storage management for aged file systems on persistent memory. In: Proc. of the Conf. on Design, Automation & Test in Europe (DATE 2017). Belgium: European Design and Automation Association, 2017. 1773−1778.

[29] Volos H, Tack AJ, Swift MM. Mnemosyne: Lightweight persistent memory. ACM SIGARCH Computer Architecture News, 2011, 39(1):91−104.

[30] Mohan C, Haderle D, Lindsay B, Pirahesh H, Schwarz P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. on Database Systems, 1992,17(1):94−162.

[31] Coburn J, Caulfield AM, Akel A, Grupp LM, Gupta RK, Jhala R, Swanson S. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. ACM SIGARCH Computer Architecture News, 2011,39(1):105−118.

[32] Hill M D, Smith A J. Evaluating associativity in CPU caches. IEEE Trans. on Computers, 1989,38(12):1612−1630.

[33] Qureshi MK, Karidis JP, Franceschini M, Srinivasan V, Lastras L, Abali B. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In: Proc. of the IEEE/ACM Int'l Symp. on Microarchitecture. 2009. 14−23.

[34] Seong NH, Woo DH, Lee HH. Security refresh: Protecting phase-change memory against malicious wear out. IEEE Micro, 2011, 32(1):119−127.

[35] Huang FT, Feng D, Xia W, Zhou W, Zhang YC, Fu M, Jiang CT, Zhou YK. Security RBSG: Protecting phase change memory with security-level adjustable dynamic mapping. In: Proc. of the 2016 IEEE Int'l Parallel and Distributed Processing Symp. 2016. 1081−1090.

[36] Fin: Flexible I/O tester. http://freecode.com/projects/fio

附中文参考文献:

[2] 罗乐,刘轶,钱德沛.内存计算技术研究综述.软件学报,2016,27(8):2147−2167. http://www.jos.org.cn/1000-9825/5103.htm [doi: 10.13328/j.cnki.jos.005103]

[3] 沈志荣,薛巍,舒继武.新型非易失存储研究.计算机研究与发展,2014,51(2):445−453.

[6] 沙行勉,陈咸彰,马殿龙,诸葛晴凤.基于非易失性内存的持久化嵌入式内存数据库.软件学报,2016,27(Suppl.(2)):320−327. http://www.jos.org.cn/1000-9825/16046.htm

[7] 潘巍,李战怀,杜洪涛,周陈超,苏静.新型非易失存储环境下事务型数据管理技术研究.软件学报,2017,28(1):59−83. http://www.jos.org.cn/1000-9825/5141.htm [doi: 10.13328/j.cnki.jos.005141]

杨朝树(1984－),男,贵州剑河人,博士,CCF 学生会员,主要研究领域为非易失性存储,文件系统,大数据系统.

诸葛晴凤(1970－),女,博士,教授,博士生导师,CCF 专业会员,主要研究领域为大数据处理系统,嵌入式系统,物联网系统,优化算法.

沙行勉(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为大数据系统,并行计算和系统,嵌入式系统和软件.

陈咸彰(1989－),男,博士,讲师,CCF 专业会员,主要研究领域为嵌入式系统和软件,新型存储,文件系统.

吴林(1991－),男,博士,主要研究领域为新型存储系统,文件系统.

吴挺(1991－),男,博士,主要研究领域为内存文件系统,云计算,虚拟化.