计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20150581
53(9):1930-1942,2016

新型非易失存储的安全与隐私问题研究综述

徐远超[1,2] 闫俊峰[1] 万 虎[1] 孙凤芸[1] 张伟功[1] 李 涛[3]
[1](首都师范大学信息工程学院 北京 100048)
[2](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[3](佛罗里达大学电子与计算机工程系 美国佛罗里达州盖恩斯维尔 32611)
(xuyuanchao@cnu.edu.cn)

A Survey on Security and Privacy of Emerging Non-Volatile Memory

Xu Yuanchao[1,2], Yan Junfeng[1], Wan Hu[1], Sun Fengyun[1], Zhang Weigong[1], and Li Tao[3]
[1](*College of Information Engineering, Capital Normal University, Beijing* 100048)
[2](*State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences), Beijing* 100190)
[3](*Department of Electrical and Computer Engineering, University of Florida, Gainesville, Florida, USA* 32611)

Abstract In recent years, emerging non-volatile memory (NVM) technologies, such as phase change memory (PCM), spin-transfer torque RAM (STT-RAM), and memristor have gained great attention of researchers. NVM has both byte-addressable and non-volatile features, thereby making it possible to replace both traditional main memory and persistent storage. Also, NVM can be used in hybrid memory and storage architecture. Due to the advantages of low latency, high density, and low power, NVM has become the promising memory technology because of the effect of alleviating memory wall problem. However, applications can access NVM directly through ordinary load/store interface, and more important, data resided in the NVM still retains after power loss, thus it imposes new challenges of security and privacy. This paper surveys several security problems about NVM and existing solutions including persistent memory leak, stray writes, metadata security, malicious wearout attacks, and non-volatile pointer. Then, privacy issues and existing studies about NVM, such as data protection and information leaks, are discussed. Finally, we explore other potential security and privacy issues related to NVM and propose several possible solutions, such as convergence of permission and protection, security of non-volatile cache, volatile NVM, and program security.

Key words memory leak; non-volatile memory (NVM); security; privacy; wearout attack; stray write

摘 要 近年来,以相变存储器(phase change memory, PCM)为代表的各种新型非易失存储(non-volatile memory, NVM)技术得到广泛关注.NVM同时具有传统内存的字节寻址特性和外存的非易失

收稿日期:2015-06-16;**修回日期**:2015-08-26
基金项目:国家自然科学基金项目(61472260,61402302);北京市自然科学基金项目(4143060);计算机体系结构国家重点实验室开放课题(CARCH201503)
This work was supported by the the National Natural Science Foundation of China (61472260, 61402302), Beijing Municipal Natural Science Foundation (4143060), and the Project Funded by the State Key Laboratory of Computer Architecture of China (CARCH201503).

特性,因而可以同时替代内存和外存,也可以用于混合存储体系结构.NVM 具有低延时、高密度、低功耗的优势,有效缓解了存储墙问题.然而,由于应用程序可以直接通过存取指令(load/store)接口访问 NVM,并且掉电后存储在 NVM 上的信息不会丢失,这给 NVM 的应用带来了一些新的安全和隐私挑战.首先讨论了持久化内存泄漏、不经意写操作、元数据安全、恶意磨损攻击、非易失指针等 NVM 应用中可能存在的安全问题以及最新的解决方案;然后讨论了数据保护、信息泄露等 NVM 应用中可能存在的隐私问题及现有的解决方案;最后探讨了 NVM 还需解决的安全和隐私问题,包括非易失缓存、程序安全等,并提出了一些解决方案,包括权限和保护机制的融合、使用易失性的 NVM 等.

关键词 内存泄漏;非易失性存储;安全;隐私;磨损攻击;不经意写

中图法分类号 TP309; TP333

信息技术近年来得到了迅猛发展,面向大数据的计算机体系结构与深度学习、神经网络等人工智能技术的结合,让人类步入了类脑计算[1]研究的时代.类脑计算是一种仿脑的大规模存储与并行处理系统.与传统的冯·诺依曼架构不同,类脑计算中的存储和计算是融合的、没有明确的界限,因此具有并行度高、鲁棒性强、能耗低等特点.可以预见,未来的体系结构发展趋势就是让计算贴近数据、减少数据的搬移[2],因此,存储成为新型计算架构的关键.近年来,各种新型非易失存储介质(non-volatile memory, NVM)的出现,如相变存储器(phase change memory, PCM)、自旋转移力矩存储器(spin-transfer torque RAM, STT-RAM)、忆阻器(memristor)等[3-5],给计算机体系结构的发展带来了福音.惠普实验室正在如火如荼开展的"The Machine"项目[6],就是采用忆阻器新型非易失存储介质,同时设计全新的操作系统 Linux++.当然,同时具有记忆和计算功能的忆阻器在技术上还不太成熟,但 PCM 等新型存储介质离产品化已经越来越近.

PCM 等非易失存储介质不仅具有字节寻址以及非易失特性,还具有存储密度高、延迟小、功耗低等优势(如表 1 所示),因此,可以替代传统的内存(如 DRAM)和外存(如机械式硬盘、基于闪存的固态硬盘等),图 1 显示了 4 种可能的存储体系架构.但是,要想完全发挥 NVM 的优势,不能简单地进行物理上的替换,还需要更新存储软件栈(software stack)[7],一方面减少软件的开销,另一方面是减少 NVM 的不足带来的负面影响,包括软件方法屏蔽存储单元的错误率(制程越小、存储密度越高,错误率也随之升高)、减少每个存储单元的写入次数(尤其是对 PCM 而言,写入次数大约为 $10^8 \sim 10^{12}$),这些不足不仅带来可靠性问题,也带来安全方面的隐患.尽管如此,可以预见,随着器件技术的成熟和支撑软件的完备,这些新型的非易失存储介质必将彻底颠覆传统的二级存储结构,给计算系统带来更大的存储容量和更低的访问延迟.

网络和信息安全设备作为一种专用的计算设备,对网络参数配置的读写速度有很高要求.比如,现有的很多路由器(如思科路由器)是将网络参数配置存放在存储容量较小但读取速度较快的非易失存储介质中,操作系统仍然存放在闪存中.随着新型存储技术的成熟,配置更大容量的内存和外存都成为可能,无疑可以进一步提升网络设备的性能.

Table1 Comparison of Properties of Several Memory and Storage Technologies[3-4]

表 1 几种存储技术属性比较[3-4]

Attribute	DRAM	SRAM	NAND Flash	STT-RAM	PCM
Cell Area/F^2	6	140	4	20	4
Access Granularity	Byte	Byte	Block	Byte	Byte
Single Cell Write Energy/($J \cdot b^{-1}$)	4.00E−15	5.00E−16	4.00E−16	2.50E−12	6.00E−12
Read Latency/ns	<10	0.2	1E5	20	12
Write/Erase Latency/ns	<10	0.2	1E6/1E5	35	100
Non-Volatile	No	No	Yes	Yes	Yes
Write Cycles	>1E16	>1E16	1E5	>1E12	1E9
In-place Update	Yes	Yes	No	Yes	Yes

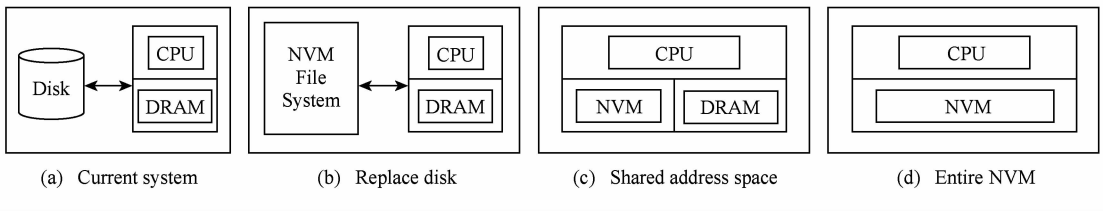

Fig. 1 System architecture options for NVM.

图 1 应用 NVM 的 4 种体系架构

安全和隐私问题在传统的计算机体系结构中也得到了极大关注. 如针对内存保护,操作系统提供了进程地址空间、虚实地址转换、内核态和用户态、段页式管理等多种机制,确保了一个进程的执行不会影响到另一个进程,用户进程不会破坏操作系统的地址空间. 针对文件安全,也提供了文件访问许可、访问控制列表以及密码等多种机制. 除了传统的安全问题之外,NVM 的使用还带来了一些新的安全和隐私问题. 比如,系统断电后,存储在其中的数据不会在短时间内消失,程序执行的状态也是持久化的[8],增加了冷启动(cold-boot attack)[9]攻击的风险. 因此,从安全的角度看,NVM 的非易失特性增大了系统被入侵和数据被窃取的风险. NVM 既可以作为工作内存(working memory)使用,也可以作为持久化存储(persistent storage)来使用[10]. 安全风险的增大主要源于工作内存. 当 NVM 作为常规存储时,其存在的安全和隐私问题在其他持久性存储介质中也同样存在,因此不作为本文讨论的重点.

NVM 安全和隐私问题的根源主要来自 2 个方面:1)NVM 的非易失特性. 传统的操作系统是针对易失的工作内存设计的,没有考虑非易失性,因此,如果将 NVM 作为工作内存使用或构建基于 NVM 的单一存储系统,就必须考虑非易失引入的安全问题. 2)字节寻址特性. 字节寻址特性使得应用程序可以采用存取指令(load/store)接口访问 NVM 上的文件,从而带来了很高的性能,但与此同时,也带来了更高的安全风险. 传统的操作系统通过文件的权限检查决定进程是否有权访问磁盘块. 现在全部变为了通过现有的内存管理单元(memory management unit, MMU)来实施保护,因此,必须从顶层视角重新设计操作系统的安全机制,将文件权限机制与内存保护机制更好地融合[11].

针对以上问题,学术界提出了一些预防和改进措施,比如,研究加密机制对 NVM 中的数据进行加密,防止数据窃取或泄密情况的发生;采取安全的文件系统机制防止元数据被应用程序恶意或不经意修改;通过在线探测和地址动态重映射来降低磨损攻击(wearout attack)的风险等. 当然,这些措施都有一定的开销,尽可能降低开销也是相关研究重点考虑的内容.

本文针对 NVM 在使用过程中可能存在的安全和隐私问题进行了分析,包括持久化内存泄漏、不经意的写操作、元数据安全、恶意磨损攻击、非易失指针等安全问题,以及数据保护、信息泄露等隐私问题,这些问题与 NVM 的特性息息相关,与存储介质无关的安全和隐私问题没有作为本文的重点;最后给出了 NVM 在安全和隐私方面的研究展望和可能的研究方向.

1 安全问题

1.1 持久化内存泄漏

所谓内存泄漏(memory leak),是指用动态存储分配函数动态开辟的空间,在使用完毕后未释放,结果导致该内存单元一直被占用. 内存泄漏分为易失性(volatile)和持久化(persistent)的内存泄漏. 易失性内存泄漏可能导致程序崩溃,但是这种情况可以通过重启程序来恢复;而持久化内存泄漏对于程序来讲是致命的.

Mnemosyne[12]提供了 2 种机制来阻止持久化内存泄漏:1)在分配内存时,要求程序提供一个持久化的指针指向该内存,从而保证了系统崩溃时该内存不会丢失;2)通过将 NVM 存储区域转换(swap)成文件来实现 NVM 的虚拟化,进而保证一个程序中出现的内存泄漏不会影响到其他程序. 基于以上 2 种机制,即使真出现了内存泄漏,可以先分配一块新的持久化区域(persistent region),然后将活动的数据从原先的区域拷贝到新的持久化区域中,以此来恢复程序,Mnemosyne 的体系结构如图 2 所示:

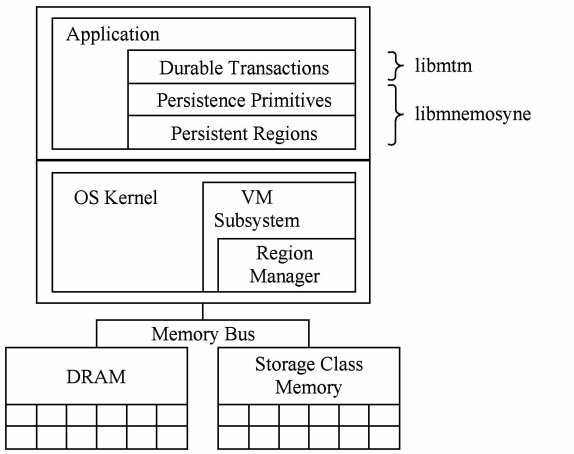

Fig. 2 Mnemosyne architecture[12].

图 2 Mnemosyne 体系结构[12]

1.2 不经意写操作

由于 NVM 的字节寻址特性,NVM 可以直接映射到进程或内核的虚拟地址空间,随后的访问可以直接通过 load/store 接口进行,不再经过冗长的文件系统路径,这大大降低了延迟,提升了性能[13-14].但是,把 NVM 直接暴露给进程或内核的虚拟地址空间,带来了极大的安全隐患,因为应用程序或驱动程序一旦存在程序缺陷(program bugs),很可能导致 NVM 的不经意写(stray writes)操作.

为了防止不经意写,PMFS[15]利用处理器中的写保护控制特征,实现了一个低开销的写保护机制.表 2 表示了写保护机制的工作原理.行名表示 NVM 映射的地址空间,列名表示不经意写的发生权限.这里不包含一个进程中多个线程间的写保护.来自用户态的不经意写操作操作内核空间时遵循特权级别(privilege levels)机制,来自用户态的不经意写操作操作用户空间时遵循分页机制.来自内核态的不经意写操作操作用户空间时遵循防止超级用户访问机制(supervisor mode access prevention, SMAP),当 SMAP 被禁时,就不允许以超级模式(supervisor mode)访问用户地址空间.来自内核态的不经意写操作操作内核空间时不会将整个 NVM 都映射到内核虚拟地址空间中,而是仅仅将其临时映射到某一个物理核的私有地址空间中.PMFS[15]利用了处理器的写保护机制(CR0.WP)来实现写保护功能.只有当 CR0.WP 没有设置时,才允许内核态操作对内核虚拟地址空间中标记为只读的页进行写操作.

Table 2 Overview of NVM Write Protection[15]

表 2 PM 写保护机制[15]

Address Space	User	Kernel
User Space	Process Isolation	SMAP
Kernel Space	Privilege Levels	Write Windows

文献[16]使用基于页表(page table)的保护来预防不经意写,通过以下 4 个步骤:1)当驱动程序被加载时,使用 *ioremap*()将 NVM 页面映射到内核虚拟地址空间中并且通过禁用读/写(R/W)位来将其初始化为只读;2)当进行读操作时,不再有额外的其他操作;3)当进行写操作时,开启 R/W 位将某一个页面置为可写,并执行写操作,当完成写操作后再将 R/W 位禁用使该页恢复为只读;4)当驱动程序卸载后,NVM 页面必须是没有映射的.

这种基于页表的保护方式方法虽然简单,但是却对性能带来了很大的影响,主要原因有以下 2 个:1)当写 1 个页时,必须要对页表项(page table entry, PTE)中的位进行 2 次变换(第 1 次是从只读变成可写,第 2 次是变回来),而每一次 PTE 属性的变化都会引起"PTE shootdown".2)操作系统对页表属性变换未做优化.因此,文献[16]对此进行了优化.第 1 种优化方法是使用缓冲/批处理(buffer/batching)机制进行保护.这种机制可以减少操作 PTE 标志位带来的开销,有 2 个原因:1)一个小的 DRAM 空间可以当作一个缓冲区用来缓解突发的大量写操作;2)在更新页表前,这些缓冲区用于存放等待重新组织的页.因此,只需要中止一次旁路转换缓冲(translation lookaside buffer, TLB),就可以批处理更新一系列连续页的 R/W 标志位,从而减小了单个操作带来的开销.第 2 种优化方法是使用多个缓冲区进行保护.为了提高扩展性,使用多个缓冲区,每一个缓冲区由单独的同步守护进程来管理,所有发送过来的写操作以循环的方式分布到不同的缓冲区中.

基于页表的写保护机制因为管理的粒度小、页表庞大导致占用的内存空间太大,文献[16]还提出基于临时映射(temporary mapping)的保护缓解该问题.临时映射是指只有在需要时才会动态地将 NVM 页映射到内核空间,通过这种方式来提供写保护,而不是通过控制每个页的可访问性来提供写保护.也就是说,当加载 NVM 设备时,NVM 页并不直接映射到内核虚拟地址空间中,而是当一个页被读写时才会被映射到内核地址空间中.NVM 空间对操作系统是不可见的,这就阻止了不经意写操作带来的潜在破坏性.临时映射方法的缺点在于:无法通过固定的虚拟地址来访问,使软件更加复杂;同时,共享页表项和大页映射的优势也将无法体现.

1.3 元数据安全

现在将一块 NVM 区域直接映射到应用程序的虚拟地址空间,也就是说应用程序可以直接操作 NVM 区域.传统的文件系统权限管理已经不复存在.在修改元数据前,应该通过一个内核服务(kernel service)来进行权限监控,防止元数据被随意修改.

Volos 等人[17]设计了一种灵活的文件系统体系结构——Aerie,用户程序能够感知下层 NVM 存储介质的存在,能够直接访问文件系统数据和元数据,无需与内核进行交互.然而,恶意程序也可以利用直接访问这一特性来破坏和违背文件系统不变式(file-system invariants)的约束条件,比如禁止在同一目录下插入 2 个同名的文件.从根本上来说,解决这个问题需要一个受信任实体或者要求所有的客户端验证元数据. Aerie 依靠软硬协同的方式来解决该问题,它将文件系统的实现分为不可信的用户库(libFS)和可信的文件系统服务(trusted file system, TFS),如图 3 所示. libFS 提供了访问文件的接口,包括命名和数据访问;TFS 提供了不可信程序之间的协作服务,确保了元数据的完整性和同步.因此,Aerie 中的用户库可以直接访问内存保护权限允许的全部存储介质.由于内存保护(protection)比文件权限(permission)更加严格,libFS 通过调用 TFS 来实现文件系统权限允许但被内存保护禁止的写操作,如只写的文件操作以及只能遍历的目录.

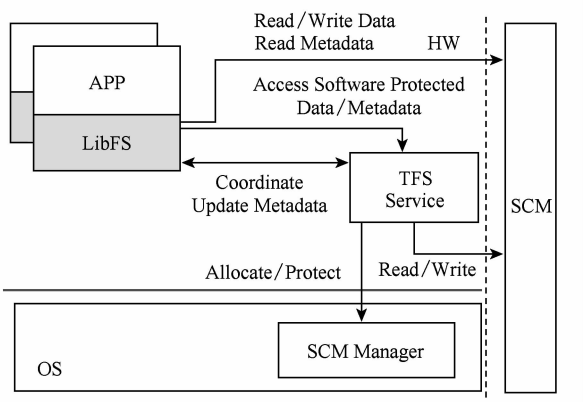

Fig. 3 Aerie architecture[17].

图 3 Aerie 体系结构[17]

1.4 恶意磨损攻击

现有的非易失存储介质都存在写入次数限制,如果反复对同一个位置进行写入操作将会造成该区域提前老化、数据出错,器件寿命也因此而缩短.正是由于有限的写入次数特性,攻击者很可能利用该特性通过恶意程序[18]对 NVM 的某些区域进行反复写入操作,造成部分存储单元快速达到耐用极限(limited endurance),在很短的时间内失效.为此,学术界提出了大量的磨损均衡算法[19-23],通过将写操作均匀分布在整个存储空间上,延长整个器件的使用寿命.

Qureshi 等人[19]提出了基于区域的起始-间隙(region based Start-Gap, RBSG)磨损均衡算法.其核心思想是将整个存储空间划分为多个区域,在写操作过程中,通过逻辑地址和物理地址重映射的方式将每一行的数据转移到相邻的地址,算法思想如图 4 所示. Start-Gap 算法的核心是地址的重映射,实现比较简单,只需要为每个区域定义 2 个独立的寄存器,分别记录起始(start)指针和间隙(gap)指针,start 跟踪存储区域中所有的行被重定位的次数,gap 跟踪存储区域中被重定位的行数.间隙行(gapline)是该存储区域中的任意一个空闲行,如果没有空闲行,需要单独给该区域一个空闲行,以便于间隙行的移动和旋转.通过这种方式便可以防止由于频繁写操作带来的行失效,从而实现了磨损均衡的效果.

Start-Gap 算法以固定的映射逻辑实现磨损均衡,抗恶意攻击能力不强.即使实现了基于随机映射的算法,但其变化规律容易被探测.另外,间隙行的移动和旋转也带来了额外开销,更重要的是,增加了空间管理的复杂度.

文献[19]还提到另外一种容忍攻击的方法,即增加对同一行写操作的时间间隔.实现的方式很多,比如,通过改变 PCM 的写队列策略,推迟写操作直到写队列达到定义的阈值.举例而言,如果 PCM 写队列有 64 个条目,可以把阈值定义为 16,只有写队列达到 16 个条目才进行一次批量写操作,如果只有攻击者的写操作,相当于延迟了 16 倍的时间,当然,这只是时间间隔上延长了,要真正减少写操作次数,还需要结合写合并等其他措施,当然,延迟写影响了实时性,因此,系统设计者需要在安全性和性能上根据实际应用场景的风险系数进行权衡.

以前的磨损均衡算法通常假设写操作是正常的,很少考虑恶意攻击等异常情况.因此,除了算法本身能否有效地实现磨损均衡外,还要从安全性方面进行设计.

安全刷新(security refresh)算法[20]通过 PCM 控制器周期性地对地址映射空间进行动态刷新,防止地址映射信息泄露.方法是:以存储体(bank)为单位,每超过一定的写入次数,就将 bank 内不同的

块地址进行重新映射,将其映射到一个新的地址.映射的过程中如图 5 所示,在开始新一轮的动态刷新前,安全刷新控制器(security refresh controller, SRC)生成一个新的随机数作为密钥,与上一个密钥共同参与运算以确定下一轮映射.因此,该算法的关键是随机数的产生、地址转换逻辑、重映射逻辑和数据交换逻辑等,这些都由 SRC 硬件完成,内嵌在每个 PCM 存储上,避免攻击者通过探测内存总线获得地址信息,从而在实现磨损均衡的同时保证了系统的安全性.

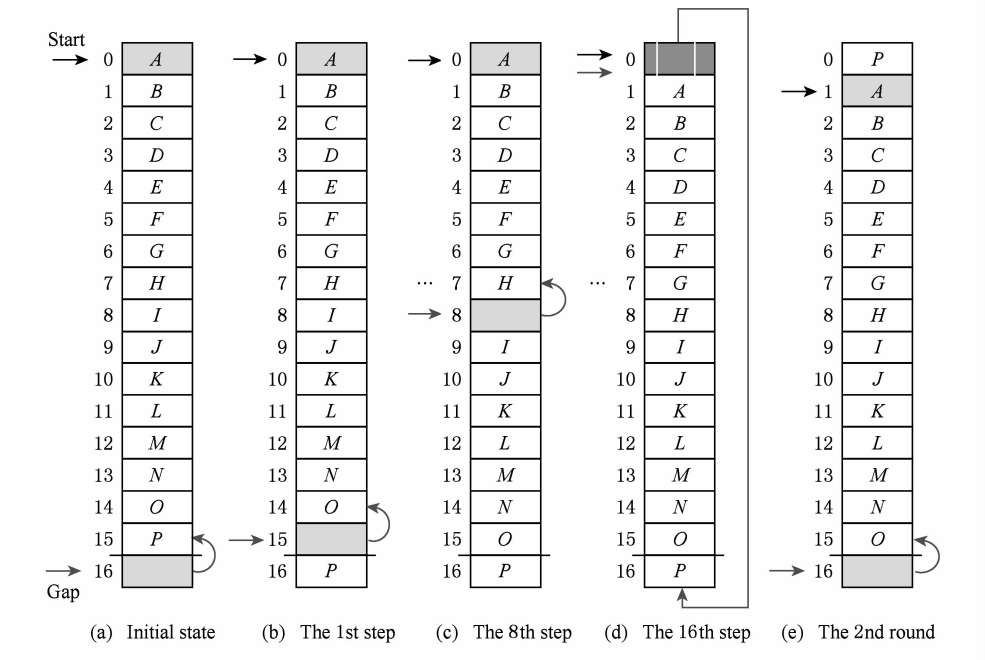

Fig. 4 Start-Gap wear leveling on a memory containing 16 lines[19].

图 4 Start-Gap 算法(16 行存储器的磨损均衡)[19]

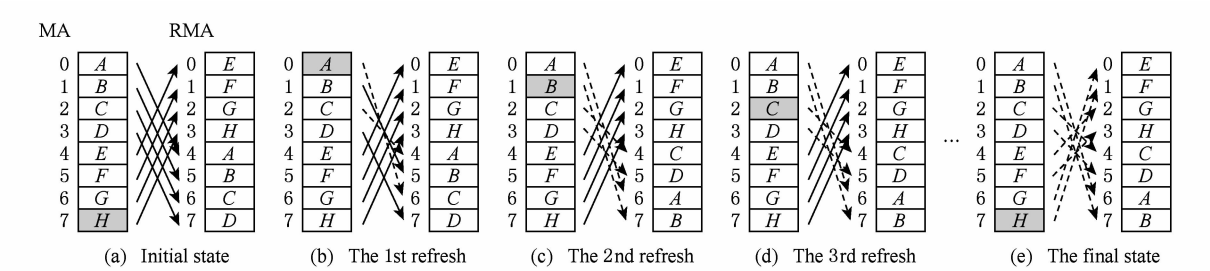

Fig. 5 An example of one complete security refresh round[20].

图 5 一轮完整的 Security Refresh 过程[20]

频繁的地址重映射会引入较大的额外写开销,有时比正常的磨损均衡算法的开销高 2~3 个数量级[21].大多数情况下,系统都处在一个安全环境下,遭受攻击的次数不是很多.因此,如果能够实时了解系统所处的安全状况、及时动态调整磨损均衡算法,有助于实现性能和安全的平衡.

在线攻击探测(online attack detection, OAD)机制[21]通过监控写数据流和地址引用来判断是否存在攻击,并通过攻击密度(attack density)来度量攻击的严重程度,以此来决定磨损均衡的地址映射频率,从而动态调整磨损均衡算法的写开销,较好地达到了安全和性能的平衡,OAD 算法的框架如图 6 所示.此文提出的自适应的磨损均衡(adaptive wear leveling, AWL)机制适合多种磨损均衡算法.实验显示,在保持算法安全性的同时,AWL 以较小的空间

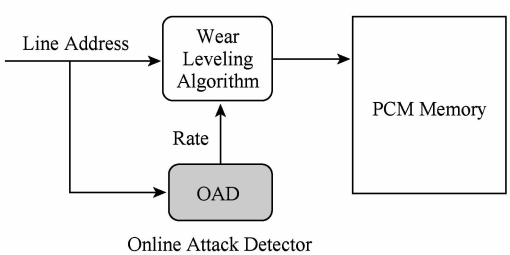

Fig. 6 Architecture of adaptive wear leveling[21].

图 6 自适应磨损均衡体系结构[21]

开销(68 B)将写开销降到了原来的 1/100～1/10,在一定程度上平衡了磨损均衡算法安全性和性能需求.

实际上,以上各种算法机制并不能从源头上制止恶意磨损攻击,因此只能被动采取防护措施.理想的做法是,从源头检测并阻止恶意程序的攻击,可以借鉴网络安全领域的入侵检测工具来阻止恶意程序.

1.5 非易失指针

NVM 具有低延迟及字节寻址特性,可以当作工作内存使用;但 NVM 也存在读写延迟不对称、写延迟和功耗高等不足,因此,采用 DRAM+NVM 的混合主存结构也是目前研究较多的一种存储结构.然而,这种混合结构增加了操作系统或内存控制器写分配数据的复杂性,同时,也由于在同一种性质的存储结构(工作内存)中同时存在易失和非易失 2 种存储介质,带来一些安全隐患.比如,存放在非易失存储区域的指针指向的是易失存储区域的地址,如果异常掉电或程序非正常结束,非易失指针[24-25]就指向了不确定区域,可能带来不安全性.

NV-Heaps[24]实现了一个可以在应用层访问的对象持久化接口,其体系架构如图 7 所示.该接口允许编程人员在分配主存空间时明确选择易失存储或者非易失存储.NVM 通过加强对指针创建规则的控制来减少因程序员误用指针而导致的安全性错误,如破坏数据结构、错误分配内存等.

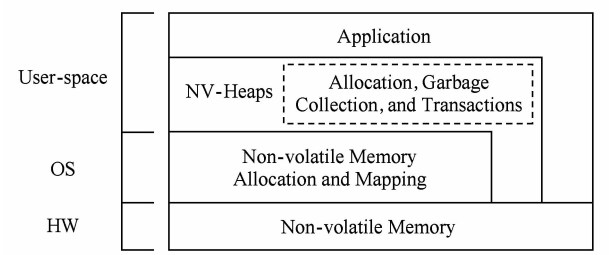

Fig. 7 NV-Heaps system stack[24].

图 7 NV-Heaps 系统栈结构[24]

NV-Heaps 将地址空间划分成了易失性的内存区域(如栈和易失性堆)和非易失性堆.相应地,根据指针是否存放在非易失性区域以及指针指向的区域是否为非易失性区域可以将指针划分为 4 类,如表 3 所示.这 4 类指针分别是:指针位于非易失性堆且指向同一个堆内部(intra-heap NV-to-NV)、指针位于非易失性堆且指向另外一个非易失性堆(inter-heap NV-to-NV)、指针位于易失性堆且指向非易失性堆(V-to-NV)、指针位于非易失性堆且指向易失性区域(NV-to-V).为了保证系统中指针引用的完整性,要求程序中所有的引用(如指针)必须指向有效的数据,因此,系统必须遵循 2 个约束条件:1)不能存在 NV-to-V 指针.因为一旦程序结束了,这些指针就失去了意义,而且当程序再次使用这个 NV-Heaps 时这些指针将会变得不安全;2)不能存在 inter-heap NV-to-NV 指针.因为如果 NV-Heaps 中包含了一个无法使用的对象,那么这些指针将会不安全.此外,这种类型的指针也使得垃圾回收机制更加复杂,它无法确定哪些被分配的空间已经废弃,因为可能会有其他 NV-Heaps(有可能已经不可用)中的指针指向这块空间.

Table 3 Volatile and Non-Volatile Pointers

表 3 易失性与非易失性指针

Allocated Space Attribute	Target Space Attribute	
	Volatile	Non-volatile
Volatile		V-to-NV Pointers
Non-volatile	NV-to-V Pointers	Intra-Heap/Inter-Heap NV-to-NV Pointers

2 隐私问题

2.1 数据保护

平板电脑、移动终端由于体积、功耗等原因,在未来的存储体系结构中率先采用 NVM 作为工作内存的可能性更大[26],因此可以用单一的存储介质同时替换传统的主存和外存.然而,这也给数据隐私带来了巨大隐患,因为系统崩溃或断电后存储在 NVM 上的所有敏感信息都依然存在,无论该 NVM 区域用于工作内存还是持久性存储.本文不讨论 NVM 作为持久性存储的数据保护问题,因为这与其他非易失性存储介质并没有本质差异.

相比之下,DRAM 作为工作内存要比 NVM 更加安全,因为 DRAM 是易失性的,掉电后数据将不复存在.而 NVM 是非易失性的,如果出现系统崩溃或异常掉电等意外情况,应用程序将来不及销毁存放在上面的数据[27].解决方案有以下 2 种:

1) 敏感数据特殊处理.对移动设备而言,如果数据(尤其是敏感数据)不再使用,应该将其擦除或者加密,防止冷启动攻击[9].所谓冷启动攻击,是指计算机电源关闭后所遭受的攻击.在使用 DRAM 构建的内存子系统中,计算机电源关闭后,内存中的信息实际上不会立即自动擦除,而是还能够保持 5 s 甚至更长时间,此时如果通过某种方式获取内存中的信息映像,就能够得到在内存运行时保存在其中的各种数据.PCM 等非易失性存储介质在系统断电后,上面的数据还能够保持相当长的一段时间,与易

失性内存相比,遭受冷启动攻击的风险更大,因此,必须采取相应预防措施.

当应用程序用完这些敏感数据时,安全和隐私策略必须决定操作系统什么时候如何擦除或加密这些敏感数据[28],这种策略可以由程序员来定义,让应用程序来决定哪些数据(相对较为敏感的数据)放在 DRAM 上还是放在 NVM 上.如果放在了 NVM 上,应用程序就必须妥善管理好这些数据.

2) 改进加密技术.由于加密技术对磨损均衡技术带来了很大影响,文献[29-30]针对隐私保护提出了一种基于计数器模式(counter mode)的改进之后的加密技术.文献[30]通过递增一个加密计数器以产生连续的密钥流,如图 8 所示,其中,计数器可以是任意的不会产生长时间重复输出的函数.计数器模式的加密强度来自密钥流的时间唯一性和空间唯一性.通常情况下,加密数据块的地址决定了空间唯一性,而计数器的值决定了时间唯一性.这种加密方式是针对缓存行(cache line)进行的,也就是加密的粒度通常为 64 B.这带来一个问题,如果其中只有几个字节被修改过,当写回主存时对整个缓存行进行加密的开销较大.为此,此文增加了数据块级的计数器,每一个数据块的加密通过缓存行级计数器和数据块级计数器共同完成.只有脏块对应的数据块级计数器才会自增,只加密脏块并将其写回到 NVM 中,同一个缓存行中的其他干净(未被修改过)的块不会被写回到 NVM 中.通过这种方法解决了部分写(partial write)的磨损均衡.

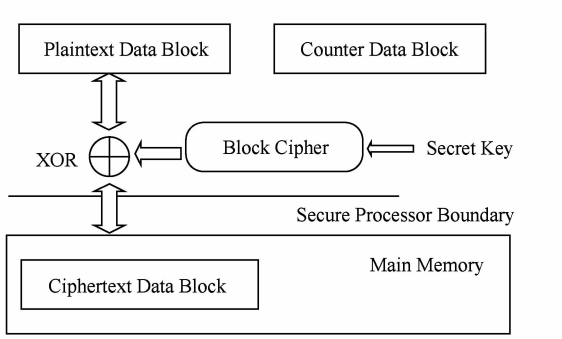

Fig. 8 Counter mode encryption for secure processors[30].

图 8 面向安全处理器的计数器模式加密[30]

对所有数据加密无疑开销很大,因此,文献[31]提出了一种针对 NVM 的增量式数据加密方式,称为 i-NVMM.i-NVMM 通过一个预测机制来判断处理器是否还会用到该数据,从而确定该数据在何时进行加密,如图 9 所示.这种增量式的数据加密方式会先对内存中暂时不用的数据进行加密,对于正在使用的数据则暂时不加密,直到达到了预测条件再对这些数据进行加密.这种加密方式主要是对主存进行加密.其思想是区分出哪些是工作集(working set),只加密除工作集之外的剩余内存数据.这种方案需要预测处理器将会使用的数据,对预测的准确度有很高的要求,因此也会带来很大的开销,加密技术也会有较大的延迟.

在基于加解密的数据保护方案中,密钥的存放是一个棘手的问题.如果把密钥等敏感数据存放在非易失存储中,当出现硬件故障时,这些敏感数据有

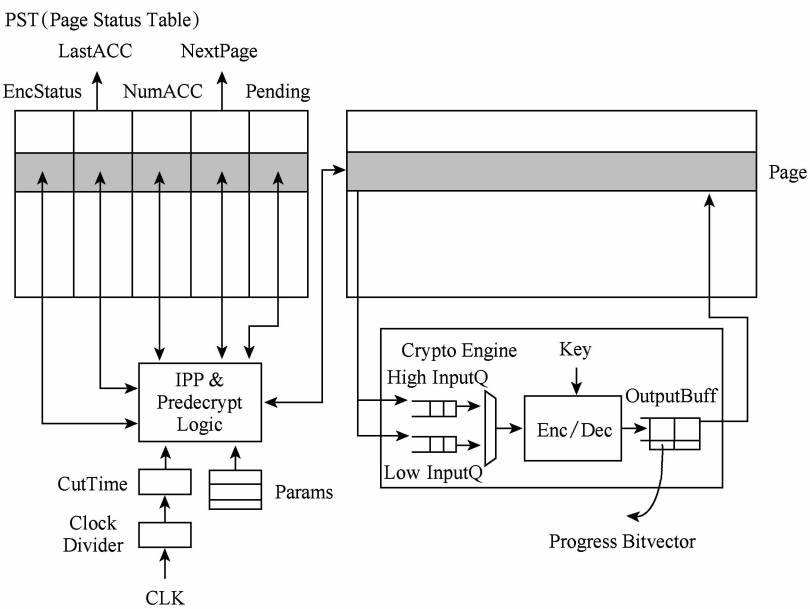

Fig. 9 High-level diagram of i-NVMM architecture support[31].

图 9 i-NVMM 体系结构支持的高级示意图[31]

被窃取的风险.为了解决该问题,文献[8]提出了内存加密控制单元(memory encryption control unit, MECU),它位于主存和二级缓存(cache)之间,用于对二级缓存和主存之间的数据传输进行加解密,用于保护主存,如图 10 所示.当从主存读取数据时,首先经过 MECU 解密,才能写入二级缓存,供 CPU 处理.同样,当处理器往主存写入数据时也要经过 MECU 进行加密.MECU 对于操作系统和 CPU 是完全透明的.用来对存储块进行加密的密钥是来自于可移动的授权令牌上的秘密信息(比如智能卡或其他类似的安全存储设备),删除这些令牌就会使主存中的明文变得不可读,从而阻止了离线攻击.

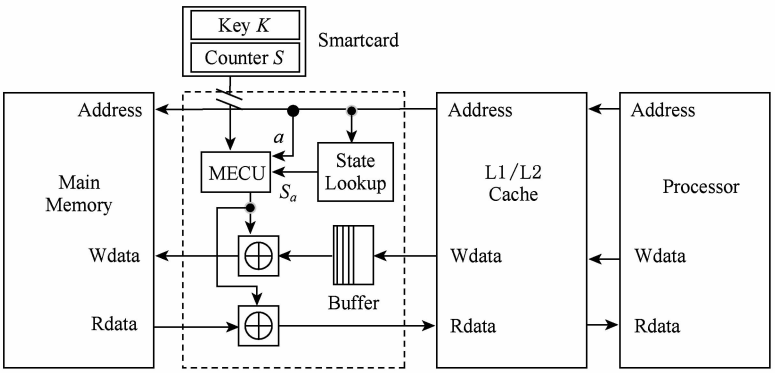

Fig. 10 A MECU-enhanced architecture[8].

图 10 一个 MECU 增强的体系结构[8]

此外,针对忆阻器的数据加密,文献[32]设计了查找表操作、移位操作、模幂与模乘等基本加密操作,并以数据加密标准算法(data encryption standard, DES)为案例进行了完整的电路设计和综合.在存储器内直接进行数据加密的方式对于提高数据存储的安全性同样具有重要的意义.

2.2 信息泄露

侧信道攻击(side-channel attack)是通过泄露的信息进行攻击的一种模式.比如,在一个加解密系统中,由于平台自身的一些物理特性导致在加解密过程中会泄露出一些物理效应信息(如时间、功耗、电磁辐射、声音、错误、缓存访问信息等),这些信息称之为侧信道信息,利用侧信道信息对密码算法和密钥进行分析的过程称为侧信道攻击.

比如,在加解密系统中针对缓存的侧信道攻击就是利用了缓存访问命中和失效时的时间和能耗差异[33].在分组密码算法中,要进行查表操作需要访问缓存,其缓存访问特征信息可通过时间或能耗特征信息泄露出来,攻击者可采集到分组密码算法运行过程中泄露的缓存访问信息,这些访问信息同查找索引、明文、密文和密钥有紧密联系,攻击者可以用来恢复密钥.学术界提出了很多防止缓存攻击的方法,尤其是如何减少由于预防攻击所引入的开销.

此外,现有的 STT-RAM,PCM 等存储介质可以在 1 个存储单元上存放 2 位或 3 位数据,称为多层单元(multi-level cell, MLC).从物理上看,这几个位并不是相邻的,而是分别存放在不同的页上,形成"页对"(paired page).这种特性不仅带来可靠性风险,也带来了安全隐患.因为在对其中的某个页进行编程时如果出现掉电,则 2 个页的数据都有可能被破坏.例如,如图 11 所示,在对 Page 11 进行编程时出现掉电,由于 Page 8 和 Page 11 是一对,Page 8 也破坏了.重新上电后,控制器通过纠错码(error correcting code, ECC)校验发现 Page 8 被破坏了,

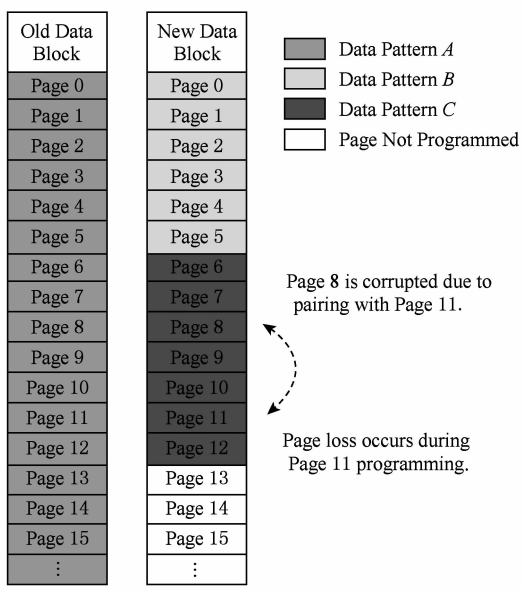

Fig. 11 Paired page corruption during programming MLC due to unexpected power loss.

图 11 MLC 编程过程中异常掉电导致的配对页的损坏

从而将 Page 8 及其后面的页全部失效.这种问题可以通过一致性技术加以解决.这种可靠性问题在单层单元(single-level cell, SLC)的 NVM 中是不存在的,因为每一个存储单元只有 1 位数据,没有"页对"的问题.

同样,攻击者也可以利用某一个页的信息,获取"页对"的相关信息进行攻击,如何预防这种攻击,还没有更多的相关研究.

3 研究展望

3.1 权限和保护机制的融合

NVM 模糊了传统内存和外存之间的界限,使用单一的存储介质可以同时完成工作内存和持久性存储的双重功能,因此,使用基于 NVM 的单一存储系统得到了极大关注[34].相对传统的二级存储架构而言,单一的存储系统缩短了 I/O 路径,但另外一方面,也带来了诸多挑战.

首先,NVM 具有字节寻址特性,应用程序可以像访问 DRAM 一样使用 load/store 接口访问 NVM.然而,另一方面,由于持久性存储介质的管理需要通过对象来完成,以及磁盘本身的信息需要记录和更新,这就导致了一些非常重要的元数据的存在.元数据在某种程度上比用户数据更重要,因此,必须保护这些数据不要被恶意修改或不经意修改.

另外,传统的操作系统使用虚拟内存管理 DRAM,使用文件系统管理磁盘.从安全的角度看,内存是通过保护机制实现的,磁盘是通过权限机制实现的.采用单一存储介质以后,为了最大化地发挥 NVM 的性能优势,势必需要使用 load/store 接口直接访问 NVM,绕过文件系统,但这也带来了很多的安全隐患,因为应用程序由于 bug 或恶意攻击等原因更容易毁坏 NVM 存储区域.

因此,需要重新设计针对单一的 NVM 存储介质的存储软件栈,包括如何更好地融合传统的权限和保护等安全机制、更好地平衡安全和性能 2 个方面需求.

3.2 非易失缓存

系统崩溃和异常掉电可能导致用户数据和元数据之间处于不一致性状态[35-40],传统的文件系统采用写前日志(write ahead logging, WAL)[41]或写时拷贝(copy on write, COW)[42]等一致性技术来保证数据的一致性,然而这些技术都带来了一定的开销,如果用于慢速存储介质,开销所占比例不大,但如果用于 NVM,则开销所占比重很大.

针对上述问题,Zhao 等人[43]提出使用非易失缓存以及相应的硬件支持可以在没有 WAL 和 COW 的情况下保证数据的一致性.此外,还有文献提出所有存储层次均为非易失存储介质的系统[14].这种系统带来了存储的可靠性,使得系统可以在掉电后得以快速启动和恢复,减少了宕机时间,然而也带来了一系列安全隐患.数据在缓存上也是非易失的,增加了敏感数据被窃取的风险;程序的执行状态也是非易失的,增加了被恶意程序攻击的风险.

因此,对于缓存中的敏感数据(包括程序执行状态),仍然需要采取加密的方法进行处理.但频繁加解密会带来很大开销,如何降低开销有待进一步研究,比如,可以考虑只在敏感数据从缓存写回到内存时进行加密.对于不敏感数据,则无需加密.

3.3 易失性的 NVM

NVM 同时具有传统 DRAM 和硬盘的双重特性,可以用一种 NVM 存储介质同时替换 DRAM 和硬盘.但是,由于程序代码静止存放和动态运行时的布局完全不同,甚至会有多个实例,因此,存储介质虽然是一样的,但还是要区分哪些作为工作内存,哪些作为持久性存储.

另外,非易失存储介质虽然具有非易失性,但它的保持时间(retention time)与写加速比(write speedup)密切相关[10].如表 4 所示.利用这个特性,可以减少 NVM 的保持时间,降低数据在 NVM 作为工作内存时被窃取的风险.通常情况下,工作内存中的数据要么是临时性数据,要么是处于非一致性状态的数据,最终的数据都要写回到持久性存储介质中,因此,作为工作内存时,其保持时间不需要太长,可以设定为几小时.如果工作内存中的数据因超过保持时间而丢失,可以从持久性存储中再次读取,但要注意数据的及时同步避免数据的不一致性.降低保持时间,不仅降低了安全风险,还加快了写入速度,降低了写功耗,可谓是一举多得.

Table 4 Write Speedup Factors with Different Retention Guarantees[10]

表 4 保持时间与写加速比之间的对应关系[10]

Non-Volatility/s	Write Speedup
10^7	1
10^6	1.2x
10^5	1.5x
10^4	1.7x
10^3	1.9x
10^2	2.1x

更进一步,也可以对持久性存储区域进行更细粒度的控制,比如,某些数据需要持久化,但保持时间不需要太长,最为典型的就是事务机制中的日志[44]以及分布式存储中的多数据副本.针对这些数据,可以选择一个合适的保持时间.

3.4 程序安全

重启是强大的恢复工具[45],重启后,DRAM 中的数据就全部丢弃了,其中也包括损坏的数据,然后从持久性存储介质中读取数据重建 DRAM.

由于程序代码鲁棒性的原因,可能因一些代码缺陷引起 DRAM 损坏,尽管这种损坏很少遇到.如果出现的话,可能导致应用程序崩溃或操作系统崩溃,通过重启可以解决该问题.如同 DRAM,NVM 同样会遭受损坏,然而由于其非易失,重启后问题仍然存在.

另外,应用程序可以直接读写 NVM,这就要求应用程序在必要时必须显式地将数据从 CPU 的高速缓存刷回到 NVM 中,如果失败的话,可能导致 NVM 上一些数据结构的损坏,而这种情况很少发生,因此,测试的难度很大.

此外,一个进程的多个线程之间也需要数据的隔离和保护,传统的处理器是通过内存一致性模型(memory consistency model)来保证程序执行的正确性,但这种模型只需要保证数据以程序的顺序写入到对所有处理器核可见的共享区域,如最后一级缓存(last level cache, LLC)、DRAM 即可.数据的持久化是由文件系统完成的,传统的文件系统(如 ext4)中只有一个单一的 I/O 队列,不存在并发问题.使用 NVM 替换 DRAM 和 HDD 之后,为了提升程序的性能,目前学术界普遍主张旁路文件系统,由应用程序直接将数据写入到持久化存储中,因此,实际上隐含着将传统的内存一致性模型延伸到持久化存储介质中.必须设计更鲁棒的编程模型和存储软件栈保证数据正确地、原子性地写入到持久性存储介质中.

4 结 论

大数据处理、类脑计算等都对存储提出了更高的要求.以相变存储器为代表的各种新型非易失存储技术的不断成熟,给新型计算机体系结构的设计带来了难得的机遇.新型存储介质同时具有字节寻址特性和非易失特性,因此,可以替代高速缓存、主存和外存,也可以与传统的存储介质混合使用,系统架构设计师可以根据应用的需求进行多种组合.

然而,NVM 的应用也带来了 3 个巨大挑战:1)针对慢速存储介质设计的传统存储软件栈需要重新设计,让应用程序可以通过 load/store 接口直接访问 NVM,绕过冗长的基于文件系统的 I/O 访问路径,以发挥出 NVM 的低延迟优势;2)由于应用程序可以直接以访问主存的方式访问 NVM 以及非易失存储介质普遍存在的写入次数有限等问题,需要采取措施防止不经意写操作、提高元数据安全以及预防磨损恶意攻击;3)由于 NVM 具有非易失特性,在作为高速缓存和工作内存时,断电后存放在上面的信息仍然存在,因此,需要采取措施防止持久化内存泄漏、非易失指针、数据保护及泄露等问题.

NVM 的安全与隐私保护是个系统工程,跨越多个层级.1)硬件层,如防止磨损攻击;2)操作系统层,要求设计灵活并且安全的文件访问接口,防止不经意的写操作,防止元数据的随意修改;3)编程模型层,需要编写鲁棒、安全的应用程序.

新型非易失存储介质还处于实验研究阶段,虽然存在读写不对称、写入次数有限等不足,但其存储密度高、功耗低、低延迟、非易失、字节寻址等特性,足以让计算机架构设计师兴奋不已.学术界和企业界都在进行紧锣密鼓的研究,新型存储介质的软件生态系统基本成熟.

可以预见,随着器件技术的成熟、支撑软件的完善、芯片成本的降低,新型存储介质必将颠覆现有的多级存储层次架构,在保证可靠性、安全隐私足够鲁棒的前提下以更高的性能服务于大数据处理和类脑计算.

参 考 文 献

[1] Sterling T, Brodowicz M, Gilmanov T. Towards brain-inspired system architectures [C] //Proc of Int Workshop on Brain-Inspired Computing. Berlin: Springer, 2013: 159-170

[2] Balasubramonian R, Chang J, Manning T, et al. Near-data processing: Insights from a MICRO-46 workshop [J]. IEEE Micro, 2014, 34(4): 36-42

[3] Zhang Hongbin, Fan Jie, Shu Jiwu, et al. Summary of storage system and technology based on phase change memory [J]. Journal of Computer Research and Development, 2014, 51(8): 1647-1662 (in Chinese)
(张鸿斌, 范捷, 舒继武, 等. 基于相变存储器的存储系统与技术综述[J]. 计算机研究与发展, 2014, 51(8): 1647-1662)

[4] Mao Wei, Liu Jingning, Tong Wei, et al. A review of storage technology research based on phase change memory [J]. Chinese Journal of Computers, 2015, 38(5): 944-960 (in Chinese)

(冒伟,刘景宁,童薇,等. 基于相变存储器的存储技术研究综述[J]. 计算机学报,2015,38(5):944-960)

[5] Strukov D B, Snider G S, Stewart D R, et al. The missing memristor found [J]. Nature, 2008, 453(7191): 80-83

[6] HP Labs. The machine: A new kind of computer [EB/OL]. [2015-06-10]. http://www.hpl.hp.com/research/systems-research/themachine

[7] Swanson S, Caulfield A M. Refactor, reduce, recycle: Restructuring the I/O stack for the future of storage [J]. Computer, 2013, 46(8): 52-59

[8] Enck W, Butler K, Richardson T, et al. Defending against attacks on main memory persistence [C] //Proc of Computer Security Applications Conf. Piscataway, NJ: IEEE, 2008: 65-74

[9] Halderman J A, Schoen S D, Heninger N, et al. Lest we remember: Cold-boot attacks on encryption keys [J]. Communications of the ACM, 2009, 52(5): 91-98

[10] Liu R S, Shen D Y, Yang C L, et al. NVM duet: Unified working memory and persistent store architecture [J]. ACM SIGPLAN Notices, 2014, 49(4): 455-470

[11] Volos H, Swift M. Storage systems for storage-class memory [C] //Proc of Annual Non-Volatile Memories Workshop (NVMW'11). Piscataway, NJ: IEEE, 2011

[12] Volos H, Tack A J, Swift M M. Mnemosyne: Lightweight persistent memory [J]. ACM SIGPLAN Notices, 2011, 46(3): 91-104

[13] Moraru I, Andersen D G, Kaminsky M, et al. Consistent, durable, and safe memory management for byte-addressable non volatile main memory [C] //Proc of the 1st ACM SIGOPS Conf on Timely Results in Operating Systems. New York: ACM, 2013: 1-17

[14] Caulfield A M, Mollov T I, Eisner L A, et al. Providing safe, user space access to fast, solid state disks [J]. ACM SIGARCH Computer Architecture News, 2012, 40(1): 387-400

[15] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory [C] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: 15

[16] Chen F, Mesnier M P, Hahn S. A protected block device for persistent memory [C] //Proc of the 30th Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2014: 1-12

[17] Volos H, Nalli S, Panneerselvam S, et al. Aerie: Flexible file-system interfaces to storage-class memory [C] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: 14

[18] Chhabra S, Solihin Y. Defining anomalous behavior for phase change memory [C] //Proc of Workshop on the Use of Emerging Storage and Memory Technologies, Held in Conjunction with HPCA. Piscataway, NJ: IEEE, 2010: 1-7

[19] Qureshi M K, Karidis J, Franceschini M, et al. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling [C] //Proc of the 42nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2009: 14-23

[20] Seong N H, Woo D H, Lee H H S. Security refresh: Protecting phase-change memory against malicious wear out [J]. IEEE Micro, 2011, 31(1): 119-127

[21] Qureshi M K, Seznec A, Lastras L, et al. Practical and secure PCM systems by online detection of malicious write streams [C] //Proc of the 17th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2011: 478-489

[22] Yun J, Lee S, Yoo S. Bloom filter-based dynamic wear leveling for phase-change RAM [C] //Proc of the Conf on Design, Automation and Test in Europe, New York: ACM, 2012: 1513-1518

[23] Seznec A. A phase change memory as a secure main memory [J]. Computer Architecture Letters, 2010, 9(1): 5-8

[24] Coburn J, Caulfield A M, Akel A, et al. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories [J]. ACM SIGARCH Computer Architecture News, 2011, 39(1): 105-118

[25] SNIA Technical Position. NVM programming model (NPM) [EB/OL]. (2013-12-21) [2015-06-10]. http://snia.org/sites/default/files/NVMProgrammingModel_v1.pdf

[26] Farrow R. Interview with steve swanson [J]. ;login:, 2015, 40(1): 15-17

[27] Bailey K, Ceze L, Gribble S D, et al. Operating system implications of fast, cheap, non-volatile memory [C] //Proc of the 13th USENIX Conf on Hot Topics in Operating Systems. Berkeley, CA: USENIX Association, 2011: 2-6

[28] Badam A. How persistent memory will change software systems [J]. Computer, 2013, 46(8): 45-51

[29] Young V, Nair P J, Qureshi M K. DEUCE: Write-efficient encryption for non-volatile memories [C] //Proc of the 20th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2015: 33-44

[30] Kong J, Zhou H. Improving privacy and lifetime of PCM-based main memory [C] //Proc of IEEE/IFIP Int Conf on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2010: 333-342

[31] Chhabra S, Solihin Y. i-NVMM: A secure non-volatile main memory system with incremental encryption [C] //Proc of the 38th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2011: 177-188

[32] Zhu Xuan. Memristor-based storage encryption architecture technology [D]. Changsha: National University of Defense Technology, 2014 (in Chinese)

(朱玄. 基于忆阻器的存储加密体系结构技术[D]. 长沙: 国防科学技术大学, 2014)

[33] Kong J, Aciicmez O, Seifert J P, et al. Architecting against software cache-based side-channel attacks [J]. IEEE Trans on Computers, 2013, 62(7): 1276-1288

[34] Meza J, Luo Y, Khan S, et al. A case for efficient hardware-software cooperative management of storage and memory [C] //Proc of the 5th Workshop on Energy-Efficient Design (WEED). Piscataway, NJ: IEEE, 2013: 1-7

[35] Kadav A, Renzelmann M J, Swift M M. Tolerating hardware device failures in software [C] //Proc of the 22nd ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2009: 59-72

[36] Pillai T S, Chidambaram V, Alagappan R, et al. All file systems are not created equal: On the complexity of crafting crash-consistent applications [C] //Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 1-16

[37] Chidambaram V, Sharma T, Arpaci-Dusseau A C, et al. Consistency without ordering [C] //Proc of the 10th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2012: 9-24

[38] Jose J, Banikazemi M, Belluomini W, et al. MetaData persistence using storage class memory: Experiences with flash-backed DRAM [C] //Proc of the 1st Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads. New York: ACM, 2013: 3-9

[39] Narayanan D, Hodson O. Whole-system persistence [J]. ACM SIGARCH Computer Architecture News, 2012, 40 (1): 401-410

[40] Fryer D, Sun K, Mahmood R, et al. Recon: Verifying file system consistency at runtime [J]. ACM Trans on Storage, 2012, 8(4): 15-27

[41] Hagmann R. Reimplementing the cedar file system using logging and group commit [C] //Proc of the Symp on Operating Systems Principles. New York: ACM, 1987: 155-162

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

[43] Zhao J, Li S, Yoon D H, et al. Kiln: Closing the performance gap between systems with and without persistence support [C] //Proc of the 46th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2013: 421-432

[44] Sun L, Lu Y, Shu J. DP^2: Reducing transaction overhead with differential and dual persistency in persistent memory [C] //Proc of the 12th ACM Int Conf on Computing Frontiers. New York: ACM, 2015: 24

[45] Bridge B. NVM support for C applications [EB/OL]. (2015-01-20) [2015-06-10]. http://www.snia.org/sites/default/files/BillBridgeNVMSummit2015Slides.pdf

Xu Yuanchao, born in 1975. PhD and assistant professor since 2005, postgraduate supervisor since 2013, at the College of Information Engineering, Capital Normal University, China. Member of China Computer Federation. His main research interests include computer architecture and operating system.

Yan Junfeng, born in 1991. Master candidate. Her main research interests include system software for non-volatile memory.

Wan Hu, born in 1991. Master candidate. Student member of China Computer Federation. His main research interests include system software for non-volatile memory.

Sun Fengyun, born in 1989. Master. Student member of China Computer Federation. Her main research interests include system software for non-volatile memory.

Zhang Weigong, born in 1967. PhD, professor and PhD supervisor. Senior member of China Computer Federation. His main research interests include embedded computer architecture, fault tolerance.

Li Tao, born in 1972. Full professor in the Department of Electrical and Computer Engineering at the University of Florida. Received his PhD degree in computer engineering from the University of Texas at Austin. Member of ACM and IEEE. His main research interests include computer architecture, the impacts of emerging technologies/applications on computing.