计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20148358
53(2):399-415,2016

基于非易失存储器的事务存储系统综述

石 伟[1] 汪东升[1,2]
[1](清华大学计算机科学与技术系 北京 100084)
[2](清华信息科学与技术国家实验室(筹) 北京 100084)
(shiwei09@mails.tsinghua.edu.cn)

Survey on Transactional Storage Systems Based on Non-Volatile Memory

Shi Wei[1] and Wang Dongsheng[1,2]
[1](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[2](*Tsinghua National Laboratory for Information Science and Technology (TNLIST), Beijing* 100084)

Abstract With the emergence and further widespread use of non-volatile memory, the storage architecture is undergoing fundamental change. Transaction processing technologies in traditional database systems and file systems are mostly designed for rotating disks while they cannot take full advantage of new features of non-volatile memory. To take full advantage of the non-volatile memory characteristics and narrow the gap between system I/O performance and CPU processing performance, transactional storage systems and technologies designed based on non-volatile memory have gained focus and great popularity. In this paper, current status of the software layer transaction processing technologies, which are used in traditional database systems and file systems, are addressed in brief firstly. Then based on the division of non-volatile memory which includes flash and phase-change memory, the existing transactional storage systems based on non-volatile memory are discussed. Finally, the research works are summarized and the possible research directions are pointed out. Among the discussion, for the research of transactional flash-based storage systems, analysis of the optimization of transaction processing technologies using traditional host interface flash storage devices is given first, followed by analysis and comparison of the characteristics of the transaction flash storage systems using dedicated transactional interfaces. For the research of transactional PCM-based transactional storage systems, using PCM in both main memory and external storage environment for transaction processing is analyzed and compared, and the key technologies including the combination of log and cache and fine-grained logging are discussed.

Key words non-volatile memory; flash memory; phase change memory (PCM); transaction processing; transactional storage system; I/O stack

摘 要 随着非易失存储器的出现和广泛使用,存储体系结构正在发生根本改变.传统数据库系统与文件系统事务处理技术大多基于磁盘设备,设计之初并未考虑非易失存储器特性.为了充分利用非易失存储器特性,缩小计算机系统的I/O性能与CPU处理性能之间的差距,基于非易失存储器的事务存储

收稿日期:2014-12-10;**修回日期**:2015-05-18
基金项目:国家自然科学基金项目(61373025,61303002);国家"八六三"高技术研究发展计划基金项目(2012AA010905)
This work was supported by the National Natural Science Foundation of China (61373025, 61303002) and the National High Technology Research and Development Program of China (863 Program) (2012AA010905).

系统与技术成为了研究热点.首先讨论了软件层事务处理技术的现状,分别介绍了传统数据库系统与文件系统事务处理常用技术;然后依据闪存和相变内存进行划分,对现有基于非易失存储器的事务存储系统与技术进行了讨论;最后给出了基于非易失存储器的事务存储系统研究展望.在基于闪存的事务存储相关研究中,首先分析了使用传统设备接口闪存设备加速事务处理的系统设计,然后重点分析了基于专用事务接口的事务闪存存储系统研究,并对基于闪存的事务存储系统不同研究进行了比较.在基于相变内存的事务存储相关研究中,分别分析并比较了相变内存在主存环境和外存环境提供事务处理的技术,重点讨论了日志与缓存融合技术、细粒度日志技术等关键问题.

关键词 非易失性存储器;闪存;相变存储器;事务处理;事务存储系统;I/O 栈

中图法分类号 TP302.1

事务作为一个抽象概念,最早在数据库领域中被提出[1-3],表示具有原子性(atomicity)、一致性(consistency)、隔离性(isolation)和持久性(durability),也即 ACID 特性的一组数据库操作.这一抽象概念主要用于保证数据库正常工作时对共享存储资源的隔离性以及异常崩溃后的数据正确性.为了在系统或进程崩溃等异常情况下保证事务特性,Mohan 等人[4]提出了 ARIES 算法,使用了写前日志(write-ahead logging, WAL)方法将数据写入数据库.文件系统相比数据库系统对事务的概念进行了简化,只保证异常情况发生后恢复数据的一致性,采用的技术一般包括日志(journaling)[5-6]、影子页(shadow paging)[7]以及软更新(soft update)[8]技术.此外,事务概念也被扩展到计算机体系结构和编译领域以解决多核共享内存体系结构下并行程序内存访问隔离性等问题[9-10].

数据库系统和文件系统中的事务处理技术经过不断的发展,出现了很多在特定场景下的高效事务机制保证算法;另一方面,由于底层存储介质从磁带、软盘到磁盘的不断发展,事务处理技术也不断地被改进以适应底层存储设备的变化.

近年来,随着以闪存为主的非易失存储器(non-volatile memory, NVM)芯片存储密度的提升与价格的下降,具有不同设备接口、基于非易失性存储器的高性能存储设备层出不穷.除了已经量产的闪存之外,目前即将投入量产以及仍处于实验阶段的新型非易失存储介质还包括相变存储器(phase change memory, PCM)[11-15]、阻变存储器(resistive RAM, RRAM 或 ReRAM)[16-18]及磁存储器(magnetic RAM, MRAM)[19-20]等.图 1 是研究机构 Gartner 在 2013 年半导体与电子研究报告中给出的几种新型非易失存储器目前的发展状态.从图 1 可以看出,相变存储器目前即将量产,但阻变存储器和磁存储器根据其调研距离实用化还有大约 5～10 年的时间.这些非易失存储器正在从根本上改变由磁介质构成的传统存储体系结构,并进一步缩小计算机系统的 I/O 性能与 CPU 处理性能之间的差距.本文中讨论的非易失存储器主要包括目前已经量产并被广泛使用的

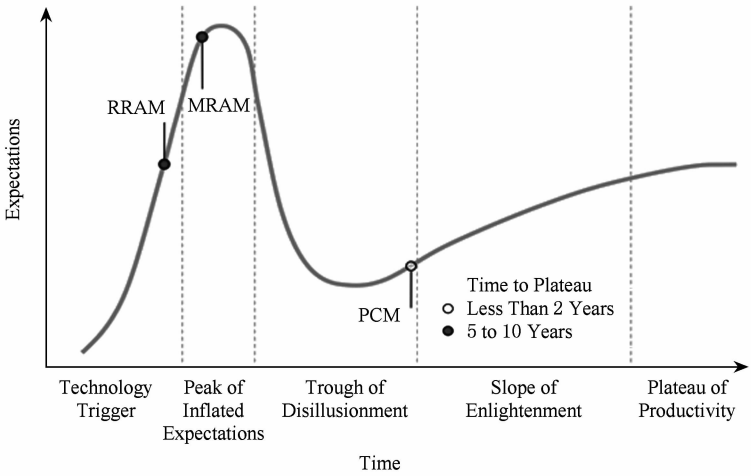


Fig. 1 Current states of development of RRAM, MRAM and PCM[21].

图 1 RRAM,MRAM,PCM 目前发展状态

闪存存储器以及即将量产的相变存储器.

对于包括闪存和相变内存在内的非易失存储介质,其自身存储特性以及构成存储设备后内部工作机制与传统存储设备存在本质差异.例如闪存具有异地更新(out-of-place update)特性、相变内存具有字节寻址(byte-addressed)特性,而现有的数据库系统与文件系统事务处理技术大部分基于磁盘设备进行设计,并没有充分利用这些特性,这导致现有的软件层事务处理技术运行在非易失存储设备上时会造成重复设计、性能较低以及写放大(write amplification)等问题.

本文首先对传统软件层事务处理技术进行了介绍;接着,依据已被广泛使用的闪存和即将量产的相变内存进行划分,对基于新型非易失存储器的事务存储系统与技术相关研究进行了讨论;最后给出了非易失存储器在事务存储系统中应用的研究展望.

1　软件层事务处理技术现状

软件层事务处理技术一直是研究的热点,由于这类技术应用在系统 I/O 栈的不同层,其技术细节不尽相同,按照应用场景的不同可以分为数据库系统事务处理技术以及文件系统事务处理技术.其中,文件系统事务处理技术一般只保证一致性和持久性,对原子性和隔离性不做强制要求.

1.1　数据库系统事务处理技术

1.1.1　ARIES

在传统的关系型数据库中,一个数据库事务通常包含若干条 SQL 语句,这些 SQL 语句经过数据库逻辑层的翻译之后,表现为对底层大量数据块的读取和写入操作.事务处理技术正是保证这些底层块设备数据块的变化,要么全部体现在数据库中,要么由于事务被中止(abort)或回滚(rollback)而在数据库中没有体现,而不存在只修改了部分数据块的中间状态.通常,数据库中的事务处理机制由存储引擎负责.

数据库存储引擎正常运行时,为上层提供事务支持并不困难.但是,当数据库系统崩溃甚至主机断电时,由于系统缓存甚至磁盘缓冲区中的数据丢失,底层存储设备上的数据信息在崩溃后的状态并不确定.要保证系统崩溃后恢复的数据仍然满足事务的 ACID 特性,需要在系统崩溃后数据恢复时进行大量的额外工作.

IBM 的研究人员 Mohan 等人[4]提出了 ARIES (algorithm for recovery and isolation exploiting semantics)恢复算法,用以在异常情况保证数据库的事务机制.ARIES 使用单独划分的一块磁盘区域作为日志区域,在将数据写入到数据库之前,首先将新数据以及旧数据一并写入到日志区域中,并使用事务标识加上标签.当日志区域大小达到某一阈值或每隔一定时间后,日志区域中的数据块被刷入到数据库中并设置检查点(checkpoint),检查点信息中包含当前系统中运行的事务状态.

如图 2 所示,当系统异常崩溃后,恢复分 3 步进行:分析(analysis)、重做(redo)和撤销(undo).首先对检查点后的日志区域进行分析,结合检查点信息,得到系统崩溃时系统中所有事务的状态;第 2 步重做操作将所有可能的在系统崩溃时还未写入数据库的数据块重新写入;最后,使用日志中存有的旧数据覆盖未提交事务的数据进行撤销操作.通过这种写前日志(WAL)的方法,ARIES 保证在异常情况发生后可以根据日志及检查点信息将数据库恢复到满足事务语义的状态.

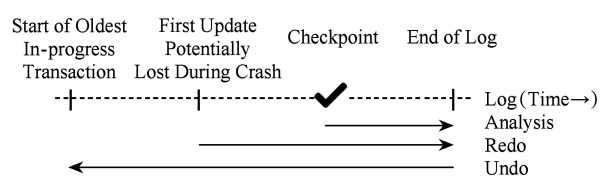


Fig. 2　The three passes of ARIES restart[22].

图 2　ARIES 重启后恢复的 3 步工作

使用 ARIES 算法保证事务语义时,数据在写入数据库之前,需先写入到日志区域中,造成了额外的写操作.在性能上,虽然 ARIES 造成了重复写入,但是由于将原先对数据库的随机写转化为了对日志区域的连续写入,同时检查点写入在后台进行.由于磁盘设备随机 I/O 性能远远低于连续 I/O 性能,所以对于磁盘设备,ARIES 虽然存在额外写但却带来了性能提升.在目前主流数据库中,MySQL InnoDB 存储引擎[23]、PostgreSQL[24]、SQL Server[25]以及 Oracle[26]均采用类似 ARIES 的事务处理机制.

1.1.2　嵌入式数据库中的事务处理技术

随着移动设备的流行,嵌入式数据库在各种移动端应用中广泛流行.相比传统关系型数据库,嵌入式数据库更加轻量,同时处于文件系统层之上.

如图 3 所示,在目前应用最为广泛的嵌入式数据库 SQLite 中,使用 2 种不同的日志模式保证数据处理的事务性.一种称为回滚模式(rollback mode),另一种称为写前日志模式(WAL mode).

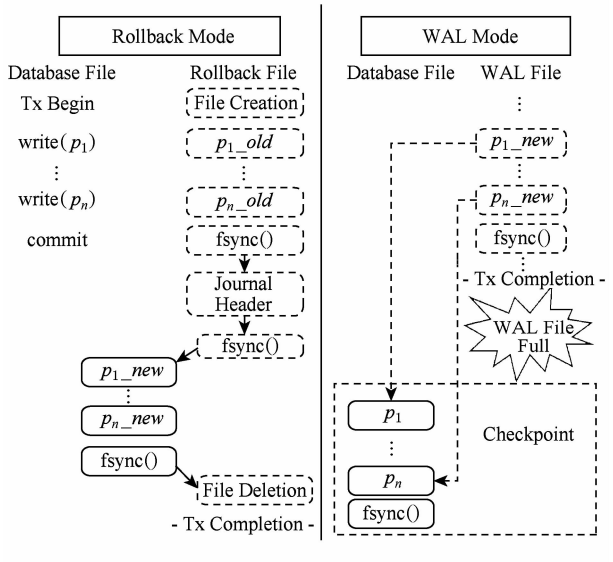


Fig. 3 SQLite journal modes[27].

图 3 SQLite 不同日志模式

由于处于文件系统层之上,这 2 种日志模式均使用文件系统提供的 fsync 调用,并假设 fsync 具有原子性. 图 3 还详细说明了 SQLite 的 2 种事务处理模式的具体流程,在回滚模式下,SQLite 对数据库文件进行修改时,首先创建一个回滚文件,然后将数据库文件被修改部分旧数据先写入回滚文件,继而对数据库文件进行修改,最后删除回滚文件. 写前日志模式则与 ARIES 非常相似,不同的是,这里写前日志是以文件的形式而不是磁盘区域的形式存在.

1.2 文件系统事务处理技术

文件系统在系统 I/O 栈中处于核心的位置,向上负责处理应用层的各种系统调用,向下负责管理底层块设备. 通常情况下,由于单个文件系统调用涉及到对许多底层数据块的操作,在系统崩溃等异常情况下操作可能会被中断. 文件系统同一时刻可能处理着大量并发的文件系统调用,所以异常情况可能会造成大量文件系统调用处于未完成的状态,这会给文件系统带来严重的一致性问题.

为了解决文件系统在系统崩溃等异常情况下的一致性问题,数据库系统事务概念被引入到文件系统中并被进行了简化. 文件系统将一个或多个系统调用封装成一个事务,但是由于文件系统并不需要强制保证原子性和隔离性,所以文件系统中事务是一种弱事务. 为了实现这种弱事务,目前主流文件系统使用的方法大致可以分为 3 类:日志、影子页和软更新.

1.2.1 日志

日志(journaling)借鉴了 ARIES 算法中的写前日志(WAL),同样也在底层块设备上使用单独的区域作为日志区域,为了避免覆盖旧数据,先将数据写入到日志区域,而后再将数据写入到文件系统中. 与写前日志不同的是,文件系统日志通常提供不同的日志级别,以 Ext3 文件系统为例,可以选择日志级别为写回(writeback)、顺序(ordered)和数据(data),由弱到强分别表示不作任何日志、只对元数据进行日志以及对元数据和数据进行日志,缺省选项为顺序. 当日志级别为顺序时,可以保证元数据一致性;而选择日志级别为数据时,可以保证数据的版本一致性. 使用日志技术的文件系统包括 Ext3/4[6], NTFS[28], ReiserFS[28], XFS[5] 等.

1.2.2 影子页

影子页是虚拟视图技术的一种实现,采用了对数据异地更新之后更新元数据的策略. 文件系统在对数据进行修改时,并不直接对该数据进行本地更新(in-place update),而是将新数据写入到别的地方,即避免数据覆盖. 除了新写入的数据之外,与新数据相关的文件系统元数据也需要修改,由于文件系统在块设备上是一个树形结构,所以由底向上一层层地异地写入新的数据,直到修改树根部分最后一个指针数据块,完成对新数据的写入. 由于在写入最后一个指针数据块之前,文件系统并没有任何的改变,而写入之后,则本次写操作完成. 使用影子页技术的文件系统包括 ZFS[29] 和 Brtfs[30] 等.

1.2.3 软更新

软更新首先分析文件系统调用涉及到的所有数据块之间的依赖关系,而后将数据块按照一定顺序写入,这个顺序可以保证异常中断后已经写入的数据或者元数据与文件系统其他元数据一致. 在具体实现中,为了保证这一点,在涉及到依赖冲突时,需要进行回滚(roll back)或者前滚(roll forward)操作. 在 Unix 的快速文件系统(FFS)中实现软更新时,可以将性能提升到内存文件系统的级别[8,31]. 但是在文件系统中实现软更新严格的顺序规则非常困难,目前使用软更新技术的有 FreeBSD 的 UFS[32] 文件系统.

1.2.4 其他技术

近年来,在文件系统事务处理研究中出现了很多新技术. 纽约大学的 Spillane 等人[33] 提出了一种事务型文件系统 Valor,向上层直接提供强事务原语. 在具体实现中,Valor 使用了类写前日志技术避免对旧数据的覆盖写,在虚拟文件系统(virtual file system, VFS)中加入了事务锁机制,保证了文件系

统调用之间的隔离性.使用 Valor 提供的接口,上层软件可以在文件系统层实现严格的 ACID 事务逻辑.此外,威斯康辛大学的 Wisdom 小组的研究人员提出了基于反向指针的文件系统 NoFS[34]以及快速恢复文件系统一致性的算法 Ffsck[35].NoFS 在传统文件系统树状结构中加入反向指针,包括目录树 i 节点(inode)之间的反向指针以及数据块与 i 节点之间的反向指针.当文件系统操作由于系统崩溃或断电等原因被中断后,NoFS 可以通过正常的文件系统数据块指针以及额外的反向指针对文件系统一致性进行验证,并恢复到一致状态.Ffsck 则通过在文件系统所管理的底层块设备上设置新的间接区域(indirect region)存放文件间接块(indirect blocks)以及目录数据块,恢复时顺序读取间接区域内的数据块进行验证,从而提高了恢复速度.

1.3　小　结

数据库系统正常运行时,事务处理机制负责事务间的数据隔离,并保证对旧数据的保护.当断电、系统崩溃或进程崩溃等异常状态发生后,事务处理机制需要中止未完成的事务,使用原先的旧数据清除没有完成的事务,从而在底层存储设备中将被故障中断的事务状态清理干净.

数据库系统主要采用基于写前日志的技术,首先将新数据写入到日志区域中,避免了对旧数据的覆盖,当事务提交后,再次写入到数据库中.这样的设计可以将随机写入转换为对日志区域的顺序写,对于磁盘友好,但由于接近 1∶1的额外的写操作,对闪存设备却会造成寿命问题.

与数据库系统相比,文件系统本身并不负责数据隔离,而是由操作系统提供的文件锁机制提供隔离性.文件系统主要保证在断电或者系统崩溃等异常状态后的数据一致性,现有的文件系统主要采用的是日志、影子页和软更新等技术.日志和影子页技术避免了对数据进行覆盖写,软更新则使用复杂的依赖性保证数据一致性.日志技术也在一定程度上造成了对底层设备的重复写入.

2　基于闪存的事务存储系统与技术

由闪存芯片组成的固态硬盘(solid-state drive,SSD)等闪存设备由于内部并行性等原因使得其 I/O 性能无论是带宽还是延迟相比磁盘都有显著地提升.为了兼容已有系统,固态硬盘大多采用传统磁盘设备接口,但是在内部工作原理上,闪存设备与磁盘设备有本质区别,传统的设备接口限制了闪存设备特性在事务处理技术上的发挥.

本节从 2 方面介绍基于闪存的事务存储系统与技术,首先介绍的是使用传统接口的闪存设备进行事务处理加速技术;接着介绍打破接口限制、扩展现有设备接口,利用软硬件协同的闪存事务处理系统.

2.1　闪存设备特性

闪存作为一种持久化的可擦除编程存储器,主要分为 NOR 闪存与 NAND 闪存.NOR 闪存存储单元使用 NOR 逻辑门,支持字节粒度的读写以及片上执行(execute on chip),程序可以在 NOR 闪存芯片内执行;NAND 闪存则使用 NAND 逻辑门作为存储单元并支持以页(page)为单位的读写.与 NOR 闪存相比,NAND 闪存具有存储密度高、成本低等优点,因而被广泛使用在存储领域.NAND 闪存根据单元存储密度不同被分为 SLC(single-level cell),MLC(multi-level cell)和 TLC(triple-level cell)闪存[36-37].下文中所讨论的闪存如无特殊说明,均为 NAND 闪存.

NAND 闪存的编程为单向编程,当电子被注入到存储单元后,即表示从状态“1”写为状态“0”,但不支持从状态“0”写为状态“1”.NAND 闪存在重新写入一个页面之前,需要进行擦除(erase)操作.与读、写不同的是,NAND 闪存擦除单位为块(block).通常,闪存页的大小为 4 KB 或更大,而闪存块包含 256 个或更多闪存页.每个闪存页除了数据区之外,根据页空间大小还包含 64～256 B 不等的页带外区域(out-of-band area,OOB)用于存放 ECC、逻辑页地址等信息.闪存的各个操作之间延迟相差较大,单个页读操作平均延迟为 25 μs,写操作平均延迟为 200 μs,但是擦除操作的平均延迟长达 1.5 ms[38].此外 NAND 闪存擦除操作有限,SLC 闪存可擦写 100 000 次左右,MLC 可擦写约 10 000 次,但 TLC 仅可擦写约 1 000 次[39-40].

为避免擦除操作造成的延迟并使擦除操作对上层透明从而兼容传统设备接口,闪存采用异地更新(out-of-place update)的策略对页面进行重写,并引入了一层重定向(indirection)向上层软件提供统一的逻辑地址空间.当对某个逻辑页重新写入数据时,首先分配一个空白物理页并写入数据,之后逻辑页被映射到新写入的物理页,原先所指向的物理页被标记为无效页面,并被定期地进行垃圾回收(garbage collection,GC).在闪存设备中,管理逻辑地址到物理地址映射的这一层重定向被称为闪存转换层

(flash translation layer, FTL). 除了地址映射,闪存转换层还负责垃圾回收以及磨损均衡等机制. 根据不同的设计,闪存转换层可以被实现在闪存设备固件中,也可以被实现在闪存设备驱动中.

对于事务处理,得益于闪存设备本身的高性能,使用传统接口的闪存设备可得到一定程度的性能提升,但是这并没有发挥出闪存设备全部的能力. 目前主流的数据库系统与文件系统是基于磁盘块设备进行设计并优化的,而闪存设备由于内部闪存转换层的存在,在内部机制上与磁盘设备完全不同,这使得针对磁盘设计的事务存储技术没有充分利用闪存设备的特点. 现有在基于传统接口闪存设备上构建的事务存储系统存在的不足主要体现在以下 3 方面:

1) 设计冗余

事务处理中的写前日志技术及虚拟视图技术本质上是为了保护活动事务原有数据不被覆盖,这样在事务主动撤销或者由于错误造成的回滚时可以将所修改的数据恢复到原有状态. 而闪存设备的异地更新特性与软件层事务处理机制保护旧数据的设计有异曲同工之处. 这样,异地更新这一设计同时出现在了闪存转换层以及软件层事务处理技术中. 虽然这 2 处异地更新的目地和效果均不相同,但 2 处异地更新存在设计上的冗余,增加了数据管理和存储的开销,同时也可能造成性能优化策略存在冲突.

2) 策略冲突

与设计冗余相比,策略冲突造成的问题更加严重. 以日志文件系统为例,在写文件之前,首先将写入的数据与修改的文件元数据写入存储设备日志区域中,这种基于磁盘设计的写入策略无形中增加了一倍以上的数据写入量,会对闪存设备寿命造成损害.

3) 协同缺失

传统存储设备接口相对简单,没有协同分工的能力. 闪存设备内部具有一定的处理能力,逻辑较复杂,由于传统设备接口的限制,闪存设备内部特性完全对上层系统透明. 内部透明性使得闪存设备可以替换磁盘设备并兼容遗留系统软件,但同时也完全杜绝了软硬件协同分工的可能性,将全部工作集中在软件层,没有充分利用设备性能进行协同处理.

传统接口闪存设备使用包括 SATA,SAS 等在内的现有磁盘接口,事务处理机制中简单使用传统接口闪存设备替换磁盘不能充分发挥闪存设备异地更新的特性,因而研究人员对现有闪存设备接口进行了扩展,提出了扩展接口闪存设备用于事务处理. 扩展接口闪存设备在内部将闪存异地更新特性与事务处理技术相整合,并以接口的形式对软件层开放,软件层可以直接使用由设备提供的事务处理接口进行事务处理,从而利用软硬件协同的方式提升事务处理系统的整体性能.

接下来的 2 小节中,首先将介绍相对简单的基于传统接口的闪存设备事务处理相关研究,接着重点介绍需要对设备接口进行扩充的基于专用事务接口的事务闪存存储系统.

2.2 基于传统接口的闪存设备事务处理

基于传统接口的闪存设备,包括 USB 闪存盘以及 SATA,SAS 固态硬盘相比磁盘具有较好的 I/O 性能. 在传统事务处理技术中,数据缓冲区(data buffer)在内存中缓存最近被数据库访问的块设备数据,日志缓冲区(log buffer)在内存中缓存事务产生的日志数据. 对于这 2 个缓冲区,都可以使用闪存设备进行数据持久化以加速事务处理.

2.2.1 基于 USB 闪存盘的 FlashLogging[41]

Intel 的研究人员提出的 FlashLogging 技术,使用了大量 USB 闪存盘并发处理日志 I/O 请求,提高了同步日志处理的性能,从而提高了系统的事务处理能力.

FlashLogging 使用生产者消费者模式进行设计,内存中的日志缓冲作为生产者,不同的 USB 闪存设备作为消费者. 任意时刻内存日志缓冲区需要写入日志数据时,只要在系统挂载的 USB 闪存盘中找到空闲的写入,继而再将数据写入到磁盘中. 由于多个闪存盘提高了日志写入的并发性,从而提高了事务吞吐. 但是在恢复时,FlashLogging 需要扫描所有的 USB 闪存盘,所以恢复代价也较大.

2.2.2 混合存储事务处理

虽然闪存设备具有较高性能,但是相比磁盘成本也较高,混合存储事务处理技术的研究目的在于充分发挥闪存设备的高性能与磁盘设备的大容量,做到高效融合. 滑铁卢大学的研究人员提出了使用磁盘(hard disk drive, HDD)和 SSD 的混合型存储系统对 MySQL InnoDB 存储引擎事务处理机制进行加速的设计[42],并在此基础上提出了基于读写代价的数据库缓冲区调整算法,将缓冲区数据根据读写代价分别存放于 HDD 和 SSD 中,以获得尽可能多的性能收益.

图 4 是混合存储系统事务处理的结构图,由于

数据库缓冲区中的数据被写回到 SSD 之后再读取比写回 HDD 再读取的成本要低,但是 SSD 的空间相对较小,所以将性能收益更高的数据放置在 SSD 上可以更加有效地提升整体性能. 为了衡量缓冲区中的每个页 p 存放到 SSD 上的潜在性能收益 $B(p)$,缓冲区调整算法采用了下列公式进行量化:

$$B(p)=r(p)(R_D-R_S)+w(p)(W_D-W_S), \quad (1)$$

其中,$r(p)$和 $w(p)$分别代表页 p 曾经被写和读的次数,R_D 和 R_S 分别代表硬盘和 SSD 的读该页需要的时间,而 W_D 和 W_S 分别代表硬盘和 SSD 写该页需要的时间. 由于硬盘读写时间与工作负载相关,所以采用了平均值代替.

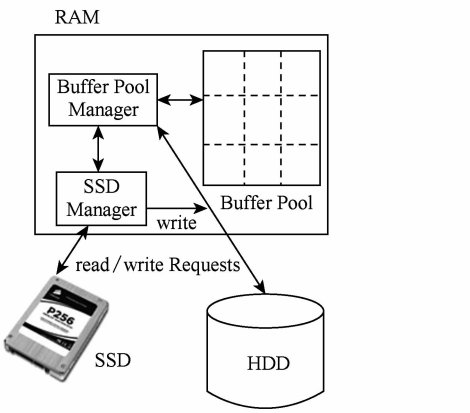


Fig. 4　Hybrid storage system for DBMS[42].

图 4　混合存储数据库系统

上述方法应用于混合存储系统可以有效地提升数据库系统的事务处理性能,同时有效利用了磁盘的大容量. 但是由于其本质上是将读写次数较多的页缓存于固态硬盘中,从一定程度上将会造成闪存设备的写磨损加剧.

2.2.3　多用户环境的固态硬盘事务处理

威斯康辛大学和 NEC 公司研究人员对多租户(multi-tenant)环境下使用固态硬盘事务处理的 3 种不同设计进行了性能测试[43]. 与磁盘设备不同的是,当使用固态硬盘作为介质时,使用虚拟化技术提供多租户的用例事务处理性能没有明显下降.

由于虚拟化技术提供多租户的用例环境下,每个虚拟机均有其各自的数据库管理系统以及操作系统,这时不管日志 I/O 还是数据 I/O 均为随机读写,在磁盘环境下,随机读写相比非虚拟化单个数据库设计以及非虚拟化多个数据库设计的日志顺序 I/O 均有较大的性能损失;固态硬盘环境下随机 I/O 与顺序 I/O 的性能相差没有磁盘那么大,因此,基于闪存固态硬盘和虚拟化技术的多租户事务处理设计可以提供良好的性能. 这也证明了在云计算环境下,基于隔离虚拟机的多租户事务存储系统使用闪存设备处理更具有优势.

2.3　扩展现有设备接口的事务闪存存储系统

使用基于传统接口的闪存设备固然可以提高事务处理的性能,但是仍然没有充分发挥闪存的优势. 为了将闪存自身的异地更新特性应用于事务处理技术中,现有闪存设备的接口必须得到扩展,从而可以将一部分的工作交给硬件,通过软硬件协同的方式实现更加高效的事务处理.

这类研究,从接口类型来看,可以分为提供数据库的 ACID 事务接口,或提供日志文件系统的一致性接口. 从提交协议上来看,可以分为链式提交协议、滑动窗口以及依据事务拓扑关系的有向无环式提交协议. 虽然具体的实现技术各异,但是所有研究都致力于提供更高效的闪存设备事务处理以及更快速的恢复.

2.3.1　基于闪存设备 FTL 的事务存储系统

Park 等人[44]提出的 AtomicWriteFTL 是最早使用闪存设备异地更新特性提供事务支持的研究. AtomicWriteFTL 提供了额外的设备接口:*AtomicWriteStart*()和 *AtomicWriteCommit*(),分别用于开始一个事务以及提交一个事务,并修改了 FAT 文件系统使用这 2 个接口. 当 FAT 文件系统在实现文件系统调用时需要同时对多个数据块进行修改时,就使用上述设备接口进行处理. *AtomicWriteStart*() 被调用时,生成一个随机的事务 *id*,并存放到每个页的 OOB 区域中. *AtomicWriteCommit*()被调用时,一个提交记录被写入持久化层中.

AtomicWriteFTL 由于研究时间较早,并未考虑大容量闪存设备的恢复以及对数据库事务的支持等问题. 同时该工作直接修改了 FAT 文件系统,而没有修改系统块设备缓存,这使得要使用设备提供的事务接口就必须将文件系统以同步的方式挂载,也造成了性能问题.

在 AtomicWriteFTL 的基础上,韩国成均馆大学和首尔大学的研究人员提出了一个事务型闪存转换层 X-FTL[27],并以移动设备上大量应用的嵌入式数据库软件 SQLite 作为上层应用进行接口设计. 在外部设备接口上,如表 1 所示,X-FTL 修改了 SATA 接口的读写命令,增加了一个代表事务标识的参数;此外还新增加了 commit 和 abort 命令.

Table 1 X-FTL Extended SATA Interface[27]

表 1 X-FTL 扩展 SATA 接口

Interface	Functionality
write(tid T, page p)	The write command of SATA is augmented with an id of a transaction T that writes a logical page p.
read(tid T, page p)	The read command of SATA is also augmented with an id of a transaction T that reads a logical page p.
commit(tid t)	A new command added to the SATA interface.
abort(tid t)	A new command added to the SATA interface.

如图 5 所示,与传统闪存转换层相比,X-FTL 新增了事务页映射表(transactional page mapping table, X-L2P)用于存放未提交事务中的逻辑页的映射信息,并在事务提交后将事务页映射表信息写入到闪存转换层地址映射表中. 与 AtomicWriteFTL 相比,X-FTL 没有使用闪存页 OOB 区域而是在闪存设备中开辟新的区域存放未提交的事务页映射信息.

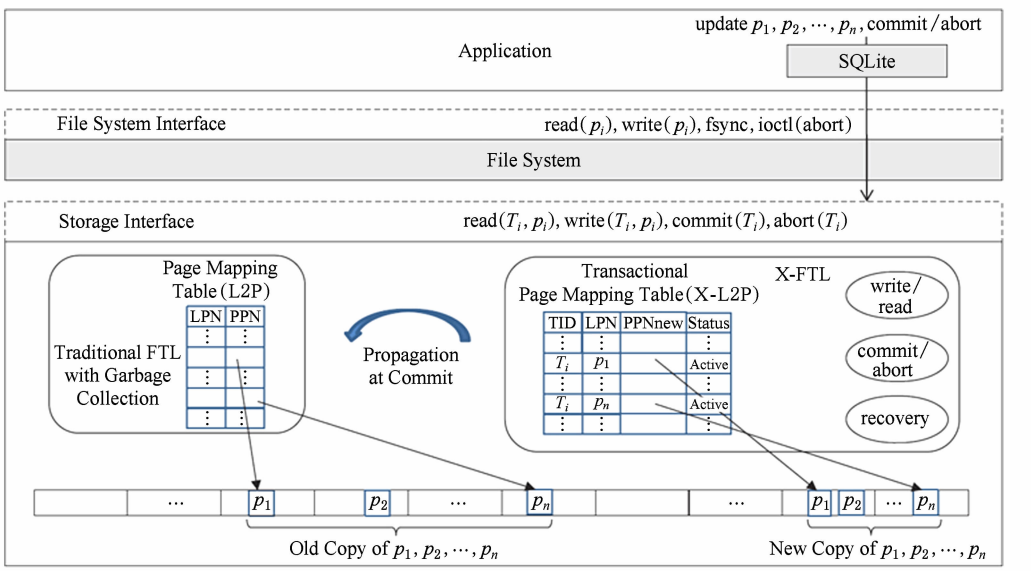


Fig. 5 X-FTL architecture: a FTL for transactional atomicity[27].

图 5 X-FTL 体系结构:一种提供事务原子性的 FTL

在 X-FTL 中,虽然数据页本身并没有被写入 2 次,但是对于 X-L2P 表中的映射信息仍然被写入了 2 次. AtomicWriteFTL 与 X-FTL 都是在现有闪存转换层基础上增加新的模块,而不需要对现有 FTL 做任何的变动.

国内中国人民大学的研究人员提出的 MixSL[45] 同样也是一种基于现有 MLC 闪存设备 FTL 的事务提交模型. 与 X-FTL 类似,MixSL 在内存中使用一个 Transaction Table 用以追踪事务状态,同时使用一个 Δ-Mapping Table 存放事务中新增的映射信息. 当故障恢复时,MixSL 扫描日志数据页 OOB 区域存放的事务信息丢弃掉未完成的数据页.

2.3.2 基于链式提交协议的 TxFlash[46]

微软的研究人员提出了基于链表的事务闪存存储系统 TxFlash 以及其提交协议简单环状提交(simple cyclic commit, SCC)和反向指针环状提交(back pointer cyclic commit, BPCC)[46]. TxFlash 利用闪存页 OOB 存放事务中下一个页的物理地址作为单向链表指针,事务的最后一页的 OOB 区域中存放事务首页的物理地址,因此一个事务中的数据页形成了环. 异常发生后,恢复时首先扫描所有的页,如果某些页根据其 OOB 中的指针构成环状结构,那么表示该事务已经提交,否则表示未提交并丢弃该事务所有的页. 由于闪存垃圾回收操作可能会破坏这种环状结构,为了区分破坏的环状结构是由垃圾回收还是未提交事务造成的,SCC 协议在写入数据页时强制擦除该数据页之前的未提交版本以及其所依赖的页,但是引入的擦除操作会影响整体写入性能,因此在 SCC 协议的基础上提出了 BPCC 协议,BPCC 中每个页除了有指向下一页的指针之外,

还包含该逻辑页上一个已提交版本的地址作为反向指针.这样如果某个页既没有反向指针指入,又处于一个被破坏的环状链表中,则这一个页属于某个未提交事务,需要被垃圾回收.

TxFlash 的 2 种提交协议 SCC 和 BPCC 对页面擦除时机存在限制,同时具有较大开销.为了解决该问题,浸会大学和南洋理工大学的研究人员在 TxFlash 的基础上提出了 Flag Commit 协议[47].Flag Commit 协议基于 SLC 闪存支持指针的原地修改,在擦除已提交事务中的页面时,原地修改 OOB 区域内的指针即可与未提交事务相区分,从而避免了 SCC 和 BPCC 由于指针依赖说带来的额外擦除开销,但 Flag Commit 协议仅限于 SLC 闪存,而无法应用于常用的 MLC 闪存中.

此外,TxFlash 的恢复基于全盘扫描,离线时间过长,印度理工学院的研究人员提出了一种基于 TxFlash 的提交协议 QRCC(quick recovery cyclic commit)[48],将每个事务的首页存放在闪存设备的一个顺序结构中.这样在恢复时,只需要依据这一结构进行事务状态检查,从而实现了 TxFlash 的恢复加速.

2.3.3　对 FTL 进行修改的事务闪存存储系统

除了基于已有 FTL 设计事务闪存存储系统之外,俄亥俄州立大学以及 FusionIO 公司的研究人员合作,对 FTL 层进行了修改,提出了 Atomic-Write[49] 的设计,直接由 FTL 层提供事务支持,并在 FusionIO 公司的产品中被广泛使用[50].与三星、Intel 等公司基于设备的(device-based)固态硬盘(SSD)等不同的是,FusionIO 面向企业提供基于主机的(host-based)固态存储系统(solid state storage, SSS),由软件层提供闪存转换层、磨损均衡和垃圾回收等功能.与 AtomicWriteFTL 和 X-FTL 相比,Atomic-Write 对上层提供的专用事务处理接口相似,但直接修改基于日志 FTL 以实现这些接口,图 6 是其内部结构图:

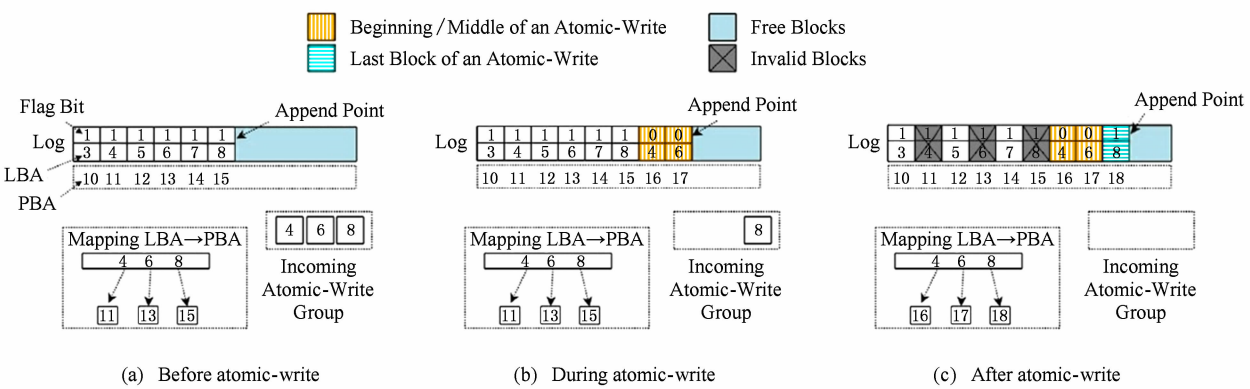


Fig. 6　Implementing Atomic-Write within a log based FTL[49].

图 6　在基于日志 FTL 中实现 Atomic-Write

为了将增加事务处理对系统性能的影响控制在最小范围,Atomic-Write 在闪存转换层的映射信息中增加了一位,当这一位为“1”时,表示当前页是一个事务的最后一页.同时,每个时刻最多有一个正在写入的原子写.在系统崩溃等异常情况后,由后向前扫描映射信息日志,在遇到第一个标志位为“1”的映射信息之前,丢弃到所有的标志位为“0”的映射信息,也即丢弃掉最后一个未完成的原子写.Atomic-Write 实现基础是日志 FTL,同时为了支持事务处理仅额外写入了 1b 信息,在不涉及到并发性的前提下性能较优.但是同一时刻,Atomic-Write 最多只有一个写入中的事务,限制了底层闪存设备的并发性.FusionIO 将这一设计实现在其商业产品 Atomic Series 的 ioMemory 中,同时提供 MySQL InnoDB 存储引擎扩展以使用这些事务处理接口.

2.3.4　基于滑动窗口的 LightTx[51]

在事务处理机制中,事务的隔离性一般由软件层的锁机制负责,如果由设备层提供,设备层一般只能提供串行(serializable)一种隔离性,也即一个事务完成后处理下一个事务,但是这限制了底层存储设备的性能.为了解决这一问题,Lu 等人[51]提出了基于滑动窗口的灵活轻量事务闪存 LightTx,支持事务的 3 种不同隔离级别:严格隔离(strict isolation)、无页冲突(no-page-conflict)和串行(serializable),并由软件层决定具体的隔离级别.在内部结构上,LightTx 将闪存设备根据事务所处不同的生命周期划分为 4 个滑动窗口,分别为已完成区域(checkpointed zone)、待定窗口(pending window)、更新窗口(updating window)和空闲区域(free zone).任意时刻,正在更新的事务只存在于待定窗口与更新窗口

中. LightTx 采用页面技术的方式在恢复时判断事务是否提交,事务的最后一个页面的 OOB 区域为事务中页总数,否则为 0. 当系统崩溃恢复时,只需要对待定窗口和更新窗口区域进行扫描和验证,从而减少了存储系统离线恢复时间.

图 7 是采用 LightTx 事务恢复协议的事务闪存设备的状态快照示例,当故障发生后,扫描更新窗口和待定窗口根据页面 OOB 中的信息可以判定 Tx6, Tx9, Tx10 为未提交事务,需要被进行垃圾回收,其余事务的 FTL 层数据则要被再次写入.

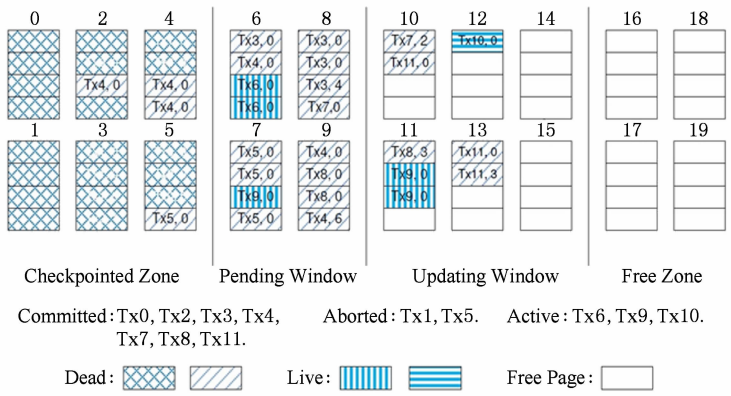


Fig. 7 Zone-based transaction state tracking[51].

图 7 基于窗口事务状态跟踪图

2.3.5 基于有向无环提交协议的 Möbius[52]

事务闪存设备除了需要写入事务数据之外,与一般的闪存设备一样,也需要保证 FTL 层的映射数据被持久化. 清华大学的研究人员基于开源硬件 OpenSSD 设计并实现了名叫 Möbius 的事务闪存设备,Möbius 将 FTL 映射数据与事务元数据相结合的方式持久化事务元数据. Möbius 将事务分为静态事务(static transaction)与动态事务(dynamic transaction),静态事务主要指事务所有待写入的页都已经在内存中,和多个页的原子写比较接近,而动态事务指事务开始时待写入的页还没有确定,需要等待计算. 通常文件系统缓存处理数据的方式是静态事务式的,但是关系型数据库中的多个 SQL 语句组成的事务是动态事务. 对于静态事务,由于事务所有的写入页已经确定,Möbius 会实现分配物理页,一个事务所分配的所有页的映射信息以及事务信息被记录到一个被称为 Atom 的闪存页中,所有的 Atom 被维护在一个称为 Atom Log Area 的区域,在恢复时使用有向无环图提交协议快速检查;而对于动态事务,Möbius 采用链表式的方式进行写入.

有向无环提交协议的关键在于当前事务的提交记录写入到后续的事务中,于是在 Atom Log Area 形成了后续 Atom 指向之前 Atom 的拓扑结构. 恢复时,对 Atom Log Area 从后向前进行扫描,如果某 Atom 被之前 Atom 所指向,这该 Atom 代表的事务已提交. 否则需要进一步进行检查,读取 Atom 内的映射信息,访问所有的物理页,如果物理页中事务 id 与 Atom 中的 id 均吻合,则表示该事务已完成,否则表示该事务未完成,需要进行垃圾回收.

2.4 小 结

基于闪存的事务存储系统按照接口技术的不同可以分成 2 类,一类系统直接使用现有接口闪存设备,包括 USB 接口或诸如 SATA,SAS 接口在内的磁盘接口闪存设备. 这类系统的设计目标主要围绕着利用这些闪存设备比传统磁盘更优的 I/O 性能加速事务处理. 另一类系统则对现有设备接口进行扩展,利用闪存异地更新特性,将写前日志的功能实现在闪存设备内部,对上层软件提供不同类型的事务处理接口,避免了重复设计造成的冗余 I/O 等问题,提升了性能.

闪存设备接口的局限性被越来越多的研究人员所关注,例如北京大学和百度公司的研究人员提出了将闪存设备通道直接暴露给软件层的设计[53],在这样的存储架构之上,直接使用闪存硬件更高效地实现了 LSM 树(log-structured merge-tree)型结构.

扩展闪存设备接口进行事务处理的技术与直接利用闪存设备 I/O 性能进行事务处理加速的技术相比更加复杂,也是目前学术界和工业界研究的主要方向,这类系统也被称为事务闪存存储系统.

事务闪存存储系统分别在系统开销、离线恢复时间、设备接口和事务隔离性等方面考虑事务处理逻辑的实现. 这些设计可以被实现在闪存设备的固件层,也可以实现在系统软件层或设备驱动层. 通过这样的设计,可以充分发挥闪存异地更新的特性,

减少了不必要的数据写入量.以日志文件系统为例,在日志模式下,系统调用的首先被写入连续的日志区域,接着被写入磁盘实际位置,数据写入量 W_T 为:

$$W_T = W_J + W_D, \quad (2)$$

其中,W_J 为磁盘日志写入量,W_D 为磁盘数据写入量,在严格一致性条件下,W_J 和 W_D 数据量大致相同,由于 W_D 是有效的数据变化量,所以写放大率达到 1:1.但是对事务闪存存储系统,W_J 的写入量可以通过闪存异地更新特性避免,闪存中的旧数据页在没有被垃圾回收之前,可以作为日志使用,从而减少了日志部分的写入.

表 2 是现有的基于闪存的事务存储系统与技术比较,其中 FlashLogging, Hybrid Buffer, Multi-tenant 直接使用现有接口的闪存设备从不同的角度对事务处理进行优化,其余则均为扩展接口的事务闪存存储系统. AtomicWriteFTL 和 X-FTL 在闪存设备已有 FTL 之上进行设计,不需要对 FTL 进行改动,但是对于故障后恢复,不仅仅需要对事务逻辑进行恢复,而且需要对 FTL 本身进行恢复. MixSL 在设计上与 X-FTL 较为接近,但是没有说明其具体使用的设备接口. Atomic-Write/FusionIO 的事务闪存设计基于日志 FTL 做了最小改动,只增加了一位代表事务的最后一页,但是这也限制了事务处理的并行性,没有充分利用底层闪存 IO 能力. TxFlash 和 Flag Commit 基于链式提交协议,但恢复时需要进行全盘扫描,离线时间过长. LightTx 基于滑动窗口,运行时代价较小,但恢复时同样需要扫描一定范围内的闪存并进行验证. Möbius 支持 2 种接口并且恢复时离线时间较短,但是对于大量小事务,由于需要额外写入 Atom 页,处理代价较大.

Table 2　Comparison of Flash-Based Transactional Storage Systems and Technologies

表 2　基于闪存的事务存储系统与技术比较

Technology	Interface	I/O Stack Redesign	Performance Overhead	Recovery Time	FTL Redesign	Parallelism	Scalability
FlashLogging[41]	USB	No	High	Medium	No	Yes	High
Hybrid Buffer[42]	SATA	No	High	Long	No	No	Low
Multi-tenant[43]	SATA/SAS	No	High	Long	No	No	Medium
AtomicWriteFTL[44]	Static	Yes	Medium	Long	No	Yes	Medium
X-FTL[27]	Static	Yes	Medium	Long	No	Yes	Medium
MixSL[45]		Yes	Medium	Long	No.	Yes	Medium
Atomic-Write/FusionIO[49]	Static	Yes	Low	Short	Yes	No	High
TxFlash[46]	Dynamic	Yes	Medium	Long		Yes	Extremely Low
Flag Commit[47]/QRCC[48]	Dynamic	Yes	Low	Long		Yes	Extremely Low
LightTx[51]	Dynamic	Yes	Low	Medium	No	Yes	High
Möbius[52]	Both	Yes	Medium	Short	Yes	Yes	High

3　基于相变存储器的事务存储系统与技术

除了已经被广泛使用的闪存之外,相变存储器作为另一种即将投入使用的存储介质,也渐渐成为了事务存储系统中研究的热点.

相变存储器(PCM)作为一种新型非易失存储介质,具有高密度、字节寻址、高性能读写等特点;此外与闪存相比,没有高延迟的擦除操作.由于这些优点,PCM 在存储体系结构上通常有 2 类不同的使用方式:1)与主存同级,如图 8 所示,独立构成主存或和 DRAM 构成混合型主存;2)与外存同级,构成固态硬盘,使用高性能结构作为可字节寻址的外存设备.在这 2 种不同的体系结构下,PCM 均可以被用于支持高性能的事务处理.

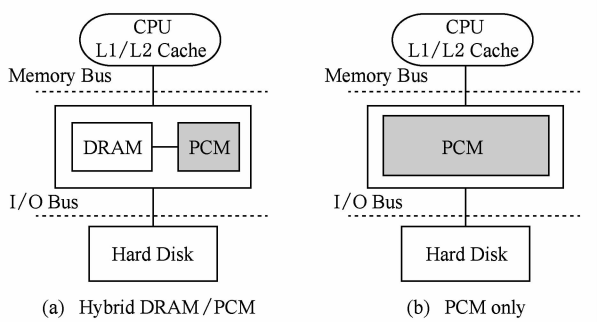


Fig. 8　PCM memory architecture alternatives[54].

图 8　PCM 作为主存的不同存储体系结构

3.1　相变存储器构建主存环境下的事务处理技术

相变存储器在构建主存环境下实现高效事务处理主要依靠于日志与缓存融合技术.缓存是存储系统高效运行所必须的,而日志则是事务处理技术带来

的额外开销. 当主存非易失时,缓存和日志之间的界限变得模糊,将日志与缓存融合可以利用系统缓存直接作为事务日志,从而更加高效.

浸会大学和南洋理工学院的研究人员提出了基于 PCM 与 DRAM 混合主存体系结构下的事务日志处理技术 PCMLogging[54]. PCMLogging 基于 PCM 与 DRAM 混合主存体系结构进行设计,其核心思想是利用 PCM 的非易失特性将数据缓冲区与日志缓冲区合二为一,这样当数据被写入到非易失的数据缓冲区后,也可以作为崩溃后的恢复日志使用.

PCMLogging 使用了映射表(mapping table, MT)、空闲页位图(free page bitmap)和反向指针(inverse mapping)3 个结构对 PCM 进行管理. 其中映射表是内存管理单元(memory management unit, MMU)中内存逻辑地址到物理地址映射的一部分,空闲页位图用于分配空闲 PCM 页,反向指针用于在系统启动时重建映射表. ActiveTxList 中存放当前活动状态的事务标识 XID,在事务将其所有脏页刷入到 PCM 中之后,ActiveTxList 中的相应 XID 会被移除. PCM 页元数据区域中 XID 字段代表最近一次修改该页的事务标识. 当系统以外崩溃后恢复时,首先扫描 ActiveTxList,丢弃未完成的事务数据,并将完成的事务数据保存并在适当时候刷新到外存中.

CMLogging 在 DRAM 和 PCM 中使用额外的数据结构记录事务的状态,并在故障恢复时依据这些状态信息丢弃未提交事务的数据. 通过将数据缓存与日志缓存相结合的方法,避免了事务处理中数据的 2 遍写入.

与 PCMLogging 相似,韩国梨花女子大学的研究人员提出了在 PCM 主存环境将文件系统日志与操作系统页缓存合二为一的文件系统事务处理方法 UBJ(union of buffer cache and journaling)[55].

如图 9 所示,通过这样的整合,首先系统更新页缓存时即完成了对日志的修改,减少了不必要的 I/O;其次,与内存总线相比,不同接口的外存设备均具有较大的延迟和较小的带宽,UBJ 将原先文件系统日志部分的写入控制在了内存这一层,而不必要和外存进行交互,从而提高了性能.

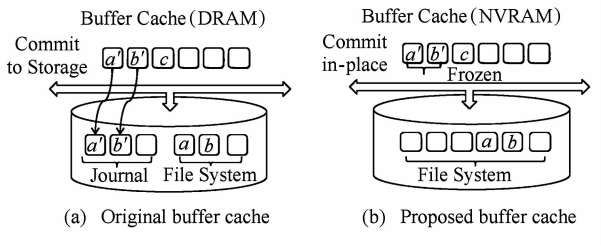


Fig. 9 Commit process comparing[55].

图 9 不同内存体系结构下的提交过程对比

如图 10 所示,匹兹堡大学、HP、IBM、AMD 和 Google 的研究人员合作提出了另外一种基于 PCM 的内存体系结构 Kiln[56],用于支持事务处理.

Kiln 在思想上与 UBJ 较为相似,与 UBJ 在内存这一层融合了页缓存和日志不同的是,Kiln 在处理器缓存中增加了一层非易失缓存,将处理器高速缓存与日志区域融合. 这种在存储体系结构中纵向的 PCM 布局有 2 个好处:1)不需要维护任何事务状态,直接利用存储体系结构中数据的纵向冗余提供日志功能,代价很小;2)将原先写日志部分的 I/O 控制在处理器内部互联中,减少了使用内存总线的 I/O 次数,从而提高了性能.

图 10 还对 Kiln 与传统块设备与混合内存体系结构下的事务处理机制进行了对比. 可以看出,Kiln 使用了纵向结构的日志布局,并将日志融入到存储

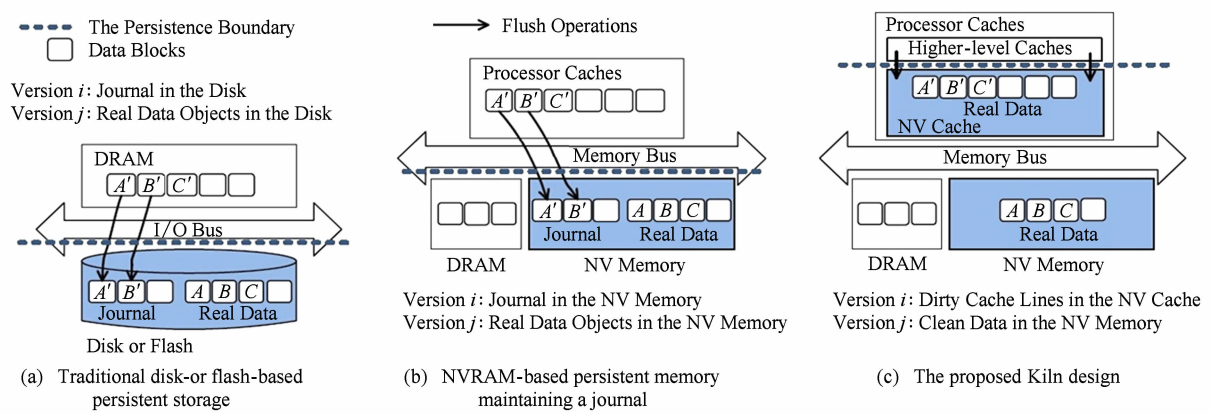


Fig. 10 Overview of Kiln persistent memory design[56].

图 10 非易失内存体系结构 Kiln

体系结构中,将持久化日志放在离处理器更近的位置,避免使用代价较高的内存总线去持久化日志数据,从而提高系统性能.但是 Kiln 对于事务的大小有一定的限制,必须要能存放于处理器的 NV Cache 中.

此外,清华大学的 Lu 等人[57]提出了应用于非易失内存环境下的 LOC(loose-ordering consistency)技术对事务处理写前日志方法的性能进行提升.LOC 中主要使用了名为 Eager Commit 和 Speculative Persistence 的 2 种技术,Eager Commit 使用事务状态表(transaction state table, TxST)对事务状态进行记录,从而在事务提交后,不需要写入额外的提交日志,从而提高性能;Speculative Persistence 指的是控制软件对于写入内存数据的可见性,消除了事物间写入的强制顺序性要求,从而提高性能.

除了应用在单节点事务处理环境下之外,PCM 也被设计应用在多核、分布式事务处理环境.与传统的易失性 DRAM 主存相比,事务处理技术中的日志方法在非易失内存环境下会被根本改变.相比传统方法,在非易失内存中,事务不需要强制进行提交前刷新(flush-before-commit).在这样的背景下,多伦多大学的研究人员提出了在非易失内存环境多核多通道(multicore and multi-socket)硬件中使用分布式日志(distributed logging)技术提升系统事务处理能力[58].多核多通道的非一致性内存访问(NUMA)体系结构下,分布式日志技术会造成处理器亲和度(processor affinity)问题,也即处理器同时访问本地和远端的内存数据页时会产生竞争.为了解决这一问题,分布式日志方法使用了事务级日志空间划分,对每一个处理器维护一个本地的日志,当事务完成之后,在这些本地的日志之间设置数据检查点(checkpoint).通过使用事务级日志空间划分内存,分布式日志技术消除了大部分处理器在进行日志时对内存页的竞争,性能得到了约 3 倍的提升.

3.2　相变存储器构建外存环境下的事务处理技术

目前相变内存芯片的 I/O 性能稍逊于 DRAM,所以也出现了一些使用 PCM 作为外存、利用其高性能和字节寻址等特性提升事务处理能力的技术,这类技术可以称为细粒度日志技术.

传统的外存块设备记录日志时,即使只有少量改动,相关块也将被整个写入到日志区域.加州大学圣地亚哥分校的研究人员使用 PCM 作为介质构造了 PCIe 接口的固态存储设备 Moneta[59],提出了基于 Moneta 的事务处理技术可编辑原子写(editable atomic write, EAW)[60].与传统的事务日志处理技术相比,由于 PCM 的字节寻址特性,使得 EAW 可以在硬件层实现对事务日志的提交前修改,这样在对某个页进行重复修改时日志中只有一份数据拷贝.EAW 从 PCM 介质特性出发,设计了一种完全不同于传统块设备外存的事务日志技术并将其实现在设备中,从而提高了存储设备的事务处理能力.

3.3　小　结

PCM 作为主存使用时,主要利用了 PCM 的非易失特性,主要有日志与缓存融合技术,使用非易失的缓存实现事务日志的功能,从而减少了日志部分的开销.

PCM 主存进行事务处理具有 2 点优势:1)将缓冲区与日志区合并,从而减少不必要的日志 I/O;2)内存层恢复,在恢复时直接利用 PCM 中的信息重建事务状态表,丢弃未完成的事务信息,而不需要和外存进行交互.

PCM 介质构建外存设备时,利用其字节寻址的特性可以减少对相同数据块进行修改的日志 I/O,从而提高了事务处理能力.这类技术被称为细粒度日志技术,相比块设备而言,细粒度日志技术可以以字节为单位进行日志写入和读取,避免了块对齐造成的不必要的写入,从而提升系统性能.

表 3 对现有的基于 PCM 的事务存储系统和技术

Table 3　Comparison of PCM-Based Transactional Storage Class Memory Systems

表 3　基于 PCM 的事务存储系统比较

Technology	Memory Architecture	I/O Stack Redesign	PCM Layer	Recovery Time	Performance Overhead	Scalability
PCMLogging[54]	Hybrid	Yes	Memory		Medium	Low
UBJ[55]	Hybrid/Pure PCM	Yes	Memory	Medium	Low	Medium
Kiln[56]	Hybrid	Yes	L3 Cache and Memory	Medium	Low	Medium
LOC[57]	Hybrid/Pure PCM	Yes	Memory	Short	Low	High
Distributed Logging[58]	Pure PCM	Yes	Memory	Long	Medium	Extremely High
EAW[60]		Yes	Storage	Long	Medium	Medium

进行了比较,从中可以看出,PCM 事务处理主要应用在主存环境,前 5 种技术均为 PCM 在主存环境下的事务处理应用.其中 Kiln 较为特殊,还将 PCM 应用于 L3 Cache,从处理器到主存就开始保证事务性持久化.在恢复时间上,LOC 由于采用事务状态表进行查询,避免了扫描,从而缩短了恢复时间.Distributed Logging 技术采用全 PCM 主存解决了多核环境下持久化层与核之间的亲和度问题,降低了内存中的数据页竞争,提升了效率.

4 研究展望

纵观存储工业发展史,从磁带设备、软盘到光介质再到磁盘,新的数据存储介质往往会给 I/O 栈带来革命性的影响.事务存储技术作为存储领域最为重要的关键技术之一,几乎被应用于所有的数据库系统与文件系统.随着闪存等非易失存储介质的广泛应用,存储体系结构正面临着大的变革,因此在新型存储体系结构下研究事务存储技术的需求变得极为迫切.目前基于闪存的事务存储系统已经开始崭露头角,基于 PCM 的事务存储系统也已走出实验阶段.展望未来,针对目前非易失存储器应用于事务处理技术所面临的问题,以下 4 个方面仍然需要进一步探索与研究.

4.1 事务存储接口

闪存的量产使得闪存设备已经被广泛应用于传统存储体系结构的缓存层和持久化层.按照设备所处的不同层级,也拥有不同的设备接口,包括传统的 SATA,SCSI,SAS 等磁盘接口,也包括 PCIe 和 NVMe 等新型设备接口,甚至还包括内存模块 DIMM 接口.PCM 设备也根据其处在 I/O 栈不同位置使用不同的设备接口.基于非易失存储器的事务存储系统核心在于利用介质特性提升事务处理能力,因而设备接口必须将这种能力暴露给系统软件层,但是如此众多的设备接口使得非易失存储设备很难提供统一的事务处理接口,从而存在适用性问题.

对于已有的一些事务接口,也存在着许多问题.由于事务应用于不同存储系统时,对 ACID 要求并不相同,例如文件系统事务一般要弱于数据库事务,但是已有的事务接口并不能体现出这种差别;此外,对于事务中隔离性、一致性等特性,也存在着多种不同的级别,现有的事务接口对于这些不同级别也不能进行区分.因此,如何提供高适用性且支持不同特性事务的设备接口是值得研究的问题.

4.2 数据可用性

存储系统最为关键的指标莫过于数据可用性(availability),对于事务处理系统而言则尤为重要.譬如银行系统内的事务代表着金钱的流动,关系着社会稳定.数据可用性在事务处理中具体而言,可以细化为 2 类问题:1)系统正常工作时的数据可用性;2)系统遇到故障后的快速恢复.

在大数据时代,数据量成为了事务存储系统快速故障恢复(failover)的主要障碍,全盘扫描式的恢复会导致较长数据离线时间,如何高效迅速的故障恢复将会成为研究的重点.

4.3 系统可扩展性

随着数据量规模的不断增长,基于非易失存储器的事务存储系统的可扩展性问题也日益突出.在宏观上,可扩展性体现在事务处理技术高效应用于分布式环境下;在微观上,则是保证多核环境下事务处理能力可扩展.

分布式环境下如何利用非易失存储器特性提供高效的事务处理是研究的关键,目前的分布式事务处理技术主要使用 2PC(two phase commitment protocol)2 阶段提交协议[61],在该提交协议里,事务提交被分为请求阶段与提交阶段,但是分阶段提交的开销较大,如果利用非易失存储器异地更新的特性,2PC 之类的分布式事务提交协议是否可以优化可能成为研究的热点,例如在闪存设备环境中首先写入各节点,而后判断,避免等待协调所耗费的时间,同时由于闪存异地更新特性,可以方便地进行回滚.此外,多核背景下的分布式处理技术中存在处理器亲和度等问题,需要探索分配日志工作的方法.

4.4 数据可靠性

各类新兴的非易失存储器由于工艺和技术上的限制,存在着寿命有限、错误率高等问题,如何保证新介质对数据的可靠持久化,必然会成为研究的热点.这类问题的研究方向一般围绕着纠删码和软件层的数据冗余备份展开.

事务存储系统是提供高可靠性的保证,因此其本身的可靠性问题必将成为研究的热点.闪存以及 PCM 等非易失存储器相比磁盘,呈现出极为不同的可靠性特征.例如闪存中存在 FTL 层,而 FTL 中的映射数据本身的可靠性也需要保证,与磁盘设备相比,这大大增加了整体可靠性问题.目前的非易失事务存储系统的设计大多基于底层设备是可靠的这一假设,并没有考虑设备本身出现可靠性问题的情况.通过进一步分析新型非易失存储介质的可靠性特征,

并依据这些特征对事务存储系统进行改造,才能为上层软件提供更高的可靠性保证.

5 总　　结

随着闪存的广泛应用以及 PCM, MRAM, RRAM 等非易失存储器的涌现,存储体系结构正面临着巨大的变化.这些非易失存储器各有特点,例如闪存设备具有异地更新特性、PCM 设备具有字节寻址特性,如何利用这些特性对 I/O 栈进行重新设计已经成为存储系统研究的重点.

事务这一概念由图灵奖得主 James Gray 提出,目前已被广泛应用于各种存储系统中以提供原子性、一致性、隔离性和持久性.目前,各种存储系统中已有的事务处理技术大多基于传统磁盘设备,没有充分利用闪存和 PCM 等非易失存储设备的特点,造成了性能损失等问题.

本文从事务处理技术应用于新型存储体系结构的角度出发,对操作系统、数据库以及体系结构领域出现的新的事务存储系统和相关技术进行了研究和分析.这些工作分别处于存储体系结构不同层,使用的介质也主要分为已经量产的闪存和即将量产的 PCM.在事务闪存存储系统中,大多数工作利用闪存设备原生的异地更新特性避免了软件层不必要的工作,快速恢复和可扩展性问题是目前的研究热点.在 PCM 事务存储系统中,主要利用了 PCM 在体系结构中的位置,将日志与系统缓存相结合,避免了重复 I/O,此外也有工作利用 PCM 的直接寻址,提供细粒度的日志,从而提高性能.

综合现有的研究情况,目前基于闪存的事务存储系统处于起步阶段,已经出现了 FusionIO 公司的 ioDrive 等系列产品为上层提供原子写原语,但是各类事务闪存存储系统的接口较为复杂且不统一,此外随着闪存设备容量的增长,离线恢复时间和其他可扩展性问题亟需解决.而基于 PCM 的事务存储系统则还处于实验阶段,并且需要从体系结构、系统软件以及应用程序等多方面共同考虑和设计.此外,例如 MRAM 和 RRAM 等非易失存储器也即将走出实验室,如何在事务存储系统中利用这些介质自身的特点也将成为研究的热点.

参　考　文　献

[1] Gray J. Notes on Data Base Operating Systems [M]. London: Springer, 1978: 393-481

[2] Gray J. The transaction concept: Virtues and limitations [C] //Proc of the 7th Int Conf on Very Large Data Bases (VLDB). San Francisco, CA: Morgan Kaufmann, 1981: 144-154

[3] Gray J, Reuter A. Transaction Processing: Concepts and Techniques [M]. San Francisco, CA: Morgan Kaufmann, 1992: 1070

[4] Mohan C, Haderle D, Lindsay B, et al. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging [J]. ACM Trans on Database System, 1992, 17(1): 94-162

[5] Sweeney A, Doucette D, Hu W, et al. Scalability in the XFS file system [C] //Proc of the 2nd USENIX Annual Technical Conf (USENIX ATC). Berkeley, CA: USENIX Association, 1996: 167-181

[6] Tweedie S C. Journaling the Linux ext2fs filesystem [C] //Proc of the 4th Annual Linux Expo. Berkeley, CA: Linux Expo, 1998

[7] Hitz D, Lau J, Malcolm M A. File system design for an NFS file server appliance [C] //Proc of the USENIX Winter 1994 Technical Conf (WTEC). Berkeley, CA: USENIX Association, 1994: 19-19

[8] Ganger G R, Patt Y N. Metadata update performance in file systems [C] //Proc of the 1st USENIX Conf on Operating Systems Design and Implementation (OSDI). Berkeley, CA: USENIX Association, 1994: 5-14

[9] Peng Lin, Xie Lunguo, Zhang Xiaoqiang. Transactional memory system [J]. Journal of Computer Research and Development, 2009, 46(8): 1386-1398 (in Chinese)
(彭林,谢伦国,张小强. 事务存储系统[J]. 计算机研究与发展, 2009, 46(8): 1386-1398)

[10] Herlihy M, Eliot J, Moss B. Transactional memory: Architectural support for lock-free data structures [C] //Proc of the 20th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 1993: 289-300

[11] Xia Fei, Jiang Dejun, Xiong Jin, et al. A survey of phase change memory systems [J]. Journal of Computer Science and Technology, 2015, 30(1): 121-144

[12] 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)

[13] Qureshi M, Gurumurthi S, Rajendran B. Phase Change Memory: From Devices to Systems [M]. New York: Morgan Claypool Publishers, 2011

[14] Qureshi M, Srinivasan V, Rivers J. Scalable high performance main memory system using phase-change memory technology [C] //Proc of the 36th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 2009: 24-33

[15] Wu Zhangling, Jin Peiquan, Yue Lihua, et al. A survey on PCM-based big data storage and management [J]. Journal of Computer Research and Development, 2015, 52(2): 343-361 (in Chinese)
(吴章玲, 金培权, 岳丽华, 等. 基于PCM的大数据存储与管理研究综述[J]. 计算机研究与发展, 2015, 52(2): 343-361)

[16] Zangeneh M, Joshi A. Design and optimization of nonvolatile multibit 1T1R resistive RAM [J]. IEEE Trans on Very Large Scale Integration (VLSI) Systems, 2014, 22(8): 1815-1828

[17] Sheu S S, Cheng K H, Chang M F, et al. Fast-write resistive RAM (RRAM) for embedded applications [J]. IEEE Design & Test of Computers, 2011, 28(1): 64-71

[18] Tsunoda K, Kinoshita K, Noshiro H, et al. Low power and high speed switching of Ti-doped NiO ReRAM under the unipolar voltage source of less than 3 V [C] //Proc of IEEE Int Electron Devices Meeting (IEDM). Piscataway, NJ: IEEE, 2007: 767-770

[19] Andre T, Nahas J, Subramanian C, et al. A 4-Mb 0.18 μm 1T1MTJ toggle MRAM with balanced three input sensing scheme and locally mirrored unidirectional write drivers [J]. IEEE Journal of Solid-State Circuits, 2005, 40(1): 301-309

[20] Senni S, Torres L, Sassatelli G, et al. Exploration of magnetic RAM based memory hierarchy for multicore architecture [C] //Proc of 2014 IEEE Computer Society Annual Symp on VLSI (ISVLSI). Piscataway, NJ: IEEE, 2014: 248-251

[21] Gartner Inc. Hype cycle for semiconductors and electronics technologies [EB/OL]. [2015-04-21]. https://www.gartner.com/doc/2795317/hype-cycle-semiconductors-electronics-technologies

[22] Franklin M. Concurrency control and recovery [M] //The Computer Science and Engineering Handbook. Boca Raton, FL: CRC Press, 1997: 1058-1077

[23] Hall C, Bonnet P. Getting priorities straight: Improving Linux support for database I/O [C] //Proc of the 31st Int Conf on Very Large Data Bases (VLDB). New York: ACM, 2005: 1116-1127

[24] Herzog R. PostgreSQL—The Linux of databases [J]. Linux Journal, 1998, (46): 1-9

[25] Lomet D, Barga R, Mokbel M, et al. Immortal DB: Transaction time support for SQL server [C] //Proc of the 24th ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2005: 939-941

[26] Lahiri T, Ganesh A, Weiss R, et al. Fast-Start: Quick fault recovery in oracle [C] //Proc of the 20th ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2001: 593-598

[27] Kang W, Lee S, Moon B, et al. X-FTL: Transactional FTL for SQLite databases [C] //Proc of the 32nd ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 97-108

[28] Prabhakaran V, Arpaci-Dusseau A C, Arpaci-Dusseau R H. Analysis and evolution of journaling file systems [C] //Proc of the 11th USENIX Annual Technical Conf (USENIX ATC). Berkeley, CA: USENIX Association, 2005: 105-120

[29] Powell H. ZFS and Btrfs: A quick introduction to modern filesystems [J]. Linux Journal, 2012, (95): 218-223

[30] Rodeh O, Bacik J, Mason C. BTRFS: The Linux B-Tree filesystem [J]. ACM Trans on Storage, 2013, 9(3): 1-9

[31] Ganger G R, Mckusick M K, Soules C A N, et al. Soft updates: A solution to the metadata update problem in file systems [J]. ACM Trans on Computer System, 2000, 18(2): 127-153

[32] Stein C A, Howard J H, Seltzer M I. Unifying file system protection [C] //Proc of the 7th USENIX Annual Technical Conf (USENIX ATC). Berkeley, CA: USENIX Association, 2001: 79-90

[33] Spillane R, Gaikwad S, Chinni M, et al. Enabling transactional file access via lightweight kernel extensions [C] //Proc of the 7th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2009: 29-42

[34] 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 (FAST). Berkeley, CA: USENIX Association, 2012: 9-21

[35] Ma A, Dragga C, Arpaci-Dusseau A, et al. Ffsck: The fast file-system checker [J]. ACM Trans on Storage (TOS), 2014, 10(1): 2-14

[36] Detlev R. Flash Memories, Economic Principles of Performance, Cost and Reliability Optimization [M]. Berlin: Springer, 2014

[37] Lu Youyou, Shu Jiwu. Survey on flash-based storage systems [J]. Journal of Computer Research and Development, 2013, 50(1): 49-59 (in Chinese)
(陆游游, 舒继武. 闪存存储系统综述[J]. 计算机研究与发展, 2013, 50(1): 49-59)

[38] Agrawal N, Prabhakaran V, Wobber T, et al. Design tradeoffs for SSD performance [C] //Proc of the 14th USENIX Annual Technical Conf (USENIX ATC). Berkeley, CA: USENIX Association, 2008: 57-70

[39] Boboila S, Desnoyers P. Write endurance in flash drives: Measurements and analysis [C] //Proc of the 8th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2010: 9-21

[40] Grupp L M, Caulfield A M, Coburn J, et al. Characterizing flash memory: Anomalies, observations, and applications [C] //Proc of the 42nd Annual IEEE/ACM Int Symp on Microarchitecture Conf (MICRO). Piscataway, NJ: IEEE, 2009: 24-33

[41] Chen S. FlashLogging: Exploiting flash devices for synchronous logging performance [C] //Proc of the 28th ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2009: 73-86

[42] Liu X, Salem K. Hybrid storage management for database systems [C] //Proc of the 32nd ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 541-552

[43] Zhang N, Tatemura J, Patel J, et al. Re-evaluating designs for multi-tenant OLTP workloads on SSD-basedI/O subsystems [C] //Proc of the 33rd ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2014: 1383-1394

[44] Park S, Ji H Y, Seong Y O. Atomic write FTL for robust flash file system [C] //Proc of the 9th Int Symp on Consumer Electronics (ISCE). Piscataway, NJ: IEEE, 2005: 155-160

[45] Fan Y, Meng X. MixSL: An efficient transaction recovery model in flash-based DBMS [G] //LNCS 7923: Proc of the 14th Int Conf on Web-Age Information Management. Berlin: Springer, 2013: 393-404

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

[47] On S T, Xu J, Choi B, et al. Flag commit: Supporting efficient transaction recovery in flash-based DBMSs [J]. IEEE Trans on Knowledge and Data Engineering, 2012, 24 (9): 1624-1639

[48] Kulkarni N, Gopinath K. Quick recovery in transactional flash [C] //Proc of 2013 IEEE Int Conf on Electronics, Computing and Communication Technologies (CONECCT). Piscataway, NJ: IEEE, 2013: 1-6

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

[50] FusionIO Inc. In a battle of hardware, software innovation comes out on top [EB/OL]. FusionIO Research Center Blog, [2014-04-26]. http://www.fusionio.com/blog/in-a-battle-of-hardware-software-innovation-comes-out-on-top

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

[52] Shi Wei, Wang Dongsheng, Wang Zhanye, et al. Möbius: A high performance transactional SSD with rich primitives [C] //Proc of the 30th Symp on Mass Storage Systems and Technologies (MSST). Piscataway, NJ: IEEE, 2014: 1-11

[53] Wang Peng, Sun Guangyu, Jiang Song, et al. An efficient design and implementation of LSM-tree based key-value store on open-channel SSD [C] //Proc of the 9th European Conf on Computer Systems (EuroSys). New York: ACM, 2014: 1-14

[54] Gao S, Xu J, He B, et al. PCMLogging: Reducing transaction logging overhead with PCM [C] //Proc of the 20th ACM Int Conf on Information and Knowledge Management (CIKM). New York: ACM, 2011: 2401-2404

[55] Lee E, Bahn H, Noh S H. Unioning of the buffer cache and journaling layers with non-volatile memory [C] //Proc of the 11th USENIX Conf on File and Storage Technologies (FAST). Berkeley, CA: USENIX Association, 2013: 73-80

[56] 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 (MICRO). Piscataway, NJ: IEEE, 2013: 421-432

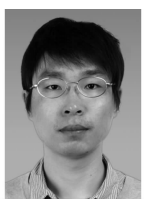
[57] Lu Youyou, Shu Jiwu, Sun Long, et al. Loose-ordering consistency for persistent memory [C] //Proc of the 32nd IEEE Int Conf on Computer Design (ICCD). Piscataway, NJ: IEEE, 2014: 216-223

[58] Wang T, Johnson R. Scalable logging through emerging non-volatile memory [C] //Proc of the 40th Int Conf on Very Large Data Bases (VLDB). New York: ACM, 2014: 865-876

[59] Caulfield A M, De A, Coburn J, et al. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories [C] //Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2010: 385-395

[60] Coburn J, Bunker T, Schwarz M, et al. From ARIES to MARS: Transaction support for next-generation, solid-state drives [C] //Proc of the 24th ACM Symp on Operating Systems Principles (SOSP). New York: ACM, 2013: 197-212

[61] Mohan C, Lindsay B. Efficient commit protocols for the tree of processes model of distributed transactions [J]. SIGOPS Operating System Review, 1985, 19(2): 40-52



Shi Wei, born in 1988. PhD. Member of China Computer Federation. His research interests include flash-based storage and file systems.

Wang Dongsheng, born in 1966. PhD, professor and PhD supervisor. Senior member of China Computer Federation, member of IEEE. His research interests include computer architecture, high performance computing, storage and file systems, and network security (wds@tsinghua.edu.cn).