面向内存文件系统的数据一致性更新机制研究

孙志龙[1] 沙行勉[1,2] 诸葛晴凤[1,2] 陈咸彰[1] 吴剀劼[1,2]
(重庆大学计算机学院 重庆 400044)[1]
(信息物理社会可信服务计算教育部重点实验室 重庆 400044)[2]

摘 要 近年来,研究界提出了多种管理新型存储级内存的内存文件系统,例如BPFS,PMFS和SIMFS。由于内存文件系统的设备访问方式和I/O路径不同于传统面向块设备的文件系统,适用于内存文件系统的数据一致性更新机制尚未得到很好的研究。为此,提出一种适用于内存文件系统的直接拷贝的数据一致性更新机制,讨论多种数据一致性更新机制在内存文件系统中的优缺点,并以内存文件系统SIMFS为基础,实现多种支持不同数据一致性更新机制的SIMFS版本。通过测试基准测试了各个SIMFS版本的性能,并分析了不同数据一致性更新机制对内存文件系统的整体性能的影响。实验结果表明,提出的直接拷贝机制在内存文件系统中取得了最好的性能。

关键词 数据一致性,内存文件系统,性能优化,日志文件系统,虚拟地址空间
中图法分类号 TP302.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.02.036

Research on Data Consistency for In-memory File Systems

SUN Zhi-long[1] Edwin H-M Sha[1,2] ZHUGE Qing-feng[1,2] CHEN Xian-zhang[1] WU Kai-jie[1,2]
(College of Computer Science, Chongqing University, Chongqing 400044, China)[1]
(Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing 400044, China)[2]

Abstract In recent years, many research works have proposed new in-memory file systems to manage storage class memory (SCM), such as BPFS, PMFS and SIMFS. Since the in-memory data access is different from traditional I/O path of block-based file systems, the data consistency mechanisms are not well studied for in-memory file systems. Thus, a new consistency mechanism called direct copying was presented for in-memory file systems. The pros and cons when different consistency strategies are used in in-memory file systems were discussed. Then, different consistency mechanisms were implemented in SIMFS to test the effectiveness of the proposed methods. Finally, experiments were conducted with standard benchmark to measure the performance of different consistency mechanisms. The experimental results show that the proposed direct copying method outperforms other strategies.

Keywords Data consistency, In-memory file system, Performance optimization, Journaling file system, Virtual address space

1 引言

随着存储级内存(Storage Class Memory, SCM)[1]技术的发展,研究界已提出多个针对SCM的新型内存文件系统,例如SCMFS[2],BPFS[3],PMFS[4],SIMFS[5]和Shortcut-JFS[6]。与面向块设备的文件系统相同,为确保用户和系统的数据在系统崩溃或断电故障后仍然保持一致,新型内存文件系统也必须使用一致性更新机制写数据。一致性更新机制的效率对文件系统的性能有至关重要的影响。

现有针对块设备文件系统的数据一致性更新机制已不适用于新型内存文件系统:1)数据更新粒度和设备访问方式不同,不同于磁盘和闪存等块设备,SCM具有可字节寻址、内存级访问速度和可用虚拟地址直接访问等特性;2)新型内存文件系统的输入输出(Input/Output, I/O)路径不同于面向块设备的文件系统,此外,相较于面向块设备的文件系统,新型内存文件系统的性能对数据一致性更新机制的效率更为敏感。因此,适用于新型内存文件系统的一致性更新机制是一个亟需探讨的问题。

到稿日期:2016-01-31 返修日期:2016-04-16 本文受"863"国家高技术研究发展计划(2015AA015304),国家自然科学基金项目(61472052),重庆市科技计划项目(cstc2014yykfB40007)资助。

孙志龙(1990—),男,硕士生,主要研究方向为面向大数据应用的新型体系结构和内存文件系统优化;沙行勉(1964—),男,博士,教授,主要研究方向为大数据存储和计算、嵌入式系统、多核系统和软件、物联网和智能计算、先进体系结构、信息安全、实时系统、云计算等;诸葛晴凤(1970—),女,博士,教授,主要研究方向为面向大数据应用的新型体系结构和系统优化、内存计算系统、物联网系统、嵌入式系统、先进存储体系结构、软/硬件协同设计等,E-mail:qfzhuge@cqu.edu.cn;陈咸彰(1989—),男,博士生,主要研究方向为面向大数据应用的新型体系结构和系统优化、内存计算系统;吴剀劼(1974—),男,博士,教授,主要研究方向为可信计算、硬件安全、容错计算、低功耗设计、实时嵌入式系统、软硬件协同设计。

为此，本文提出直接拷贝（Direct Copy，DC）的数据一致性更新机制，其利用连续虚拟地址空间和硬件内存管理单元（Memory Management Unit，MMU）高速定位备份数据的物理位置，以提高写操作的性能。以自主设计实现的新型内存文件系统 SIMFS 为基础，实现多种数据一致性更新机制，并使用标准测试基准 IOZone[7] 对使用不同一致性更新机制的 SIMFS 做对比测试。实验结果表明，直接拷贝的数据一致性机制取得了最好的性能。

2　相关工作

作为持久化文件系统必不可少的部分，数据一致性更新机制得到广泛研究。文件系统采用写时拷贝（Copy-on-Write，CoW）[3,8]、日志（Logging）[4,5,9,10] 或结构化日志更新[11-12] 3 种方式之一来实现写操作的原子性，从而达到维护文件系统一致性的目的。然而传统面向块设备的文件系统的一致性操作易导致写放大效应，即实际写的数据量远大于 I/O 请求的数据量。例如更新一个块中的单个字节也需要更新整个 512 字节的数据块。写放大效应会显著降低文件系统的性能。

SCM 的物理特性不同于块设备，内存文件系统可以利用 SCM 可字节寻址、中央处理器（Central Processing Unit，CPU）直接访问的特性重新设计一致性机制，尽可能地减少写放大效应，从而提升文件系统的性能[13]。例如，BPFS 提出利用 CPU 原子性写的特性来改进原有 CoW 方式对所有数据的更新以实现原子性操作[3]；PMFS 提出利用 CPU 原子性更新和细粒度日志的方式来更新元数据，采用 CoW 方式更新文件数据[4]；Shortcut-JFS 提出利用 SCM 可以按字节访问的特点来改良原来的日志实现方式，根据更新数据量的大小选择不同的更新方式[6]。本文仅讨论内核态文件系统的一致性机制，不考虑用户态文件系统[14] 的一致性。

3　内存文件系统

文件系统中有两种数据，即文件数据和文件元数据。文件元数据存有文件数据在物理介质上的存储位置的索引。内存文件系统中文件元数据的索引通常组织为多层次的结构，每层的一个节点都是一个内存页，存放着指向下一层节点的指针或文件数据。

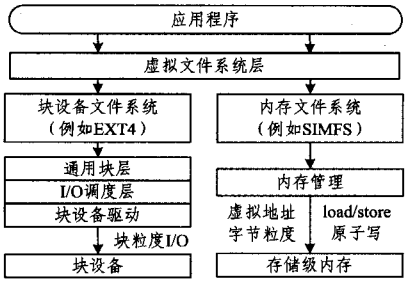

图 1　Linux 系统中的文件系统软件层次

内存文件系统所管理的 SCM，例如相变存储器（Phase Change Memory）[15] 和磁畴壁存储器（Domain Wall Memory）[16] 可以直接连接在内存总线上，并通过 CPU 的 load/store 指令进行字节粒度的访问，甚至支持 8 位、16 位和 64 位硬件原子写指令更新数据。而面向块设备的文件系统必须通过为块设备设计的块设备层等软件层次和 I/O 总线以块为粒度访问数据。内存文件系统的结构和 I/O 路径都不同于面向块设备的文件系统，如图 1 所示。因此，面向块设备文件系统的数据一致性更新机制不适用于内存文件系统。

目前，典型的内存文件系统有 SCMFS，BPFS，PMFS，SIMFS 和 Shortcut-JFS。SCMFS[2] 没有考虑数据的一致性更新机制，本文不予讨论。BPFS[3] 用树形结构组织所有的文件和文件数据，并提出用原子写指令优化后的影式分页（Shadow paging）技术对数据做一致性更新，即短路影式分页（Short-Circuit Shadow Paging，SCSP）。PMFS[4] 用 B 树组织文件数据，并采用 CoW 技术对数据做一致性更新。Shortcut-JFS[6] 自身并没有真正地实现一个内存文件系统，而是根据更新粒度的不同将自己的一致性算法实现到了 Linux 的临时文件系统 RAMFS 中。

在 SIMFS[5] 中提出了一种称为文件页表的数据结构组织文件数据。每级文件页表的结构和内存页表的结构相似，即上层页表存放下层页表的物理地址，最后一层页表存放数据页的物理地址，如图 2 所示。SIMFS 中的每个文件都有连续且独立的虚拟地址空间。当文件被打开时，SIMFS 就给该文件分配一段连续的内核虚拟地址空间，然后将该文件的文件页表插入分配得到的虚拟地址空间的内核页表。此后，SIMFS 就可以通过该文件的内核虚拟地址和 CPU 中的硬件 MMU 快速访问文件数据，而不需要搜索文件元数据，利用 MMU 和文件的连续虚拟地址空间提高性能。

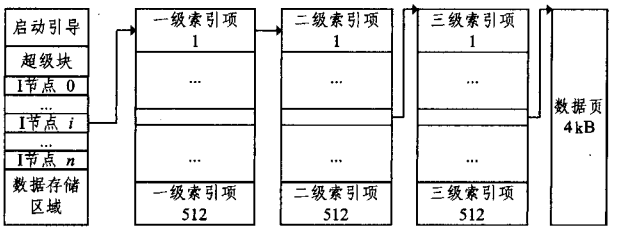

图 2　SIMFS 文件数据组织

4　数据一致性更新机制

大多数文件系统都使用以下两种技术之一来保证数据的一致性更新：写日志技术和写时拷贝的技术。两种技术都在内存文件系统中得到了不同的优化和应用。本节讨论 3 种典型的数据一致性更新机制（标准写日志（Standard Logging）技术[17]，BPFS 提出的 SCSP[3] 和 Shortcut-JFS 提出的自适应日志（Adaptive Logging，AL）技术[8]）的优缺点。图 3－图 5 以示例的形式说明并比较了它们的开销，图中每个物理块的大小为 4kB，文件结束位置如图中垂直实线所示，现要更新该文件阴影部分所示的大小为 4kB 的数据。

4.1　标准写日志

标准的写日志方式收到写请求后，日志文件系统执行如下步骤：

1）预写日志（Write-Ahead Logging，WAL）。文件系统在这个阶段将原文件中需要更新的数据写到日志区域并设置相应的标志位。

2)检查点(Checkpointing)。在这个阶段,文件系统周期性地将步骤 1)中更新后的备份数据写回原文件的相应位置。

可以看到,如果在步骤 1)出现故障,原文件的数据不受影响;如果系统在步骤 2)出现故障,在日志区域有完整的新数据,只需要在系统恢复时把日志区域的新数据更新到原文件就可以保证数据的一致性。

标准的写日志方式专为块设备设计,以块作为读写数据的最小粒度。这种方式会导致实际写的数据量远大于写请求的数据量。如图 3 所示,文件系统在 WAL 阶段把原文件更新后的数据块 B′和数据块 C′备份到日志区域,此时写了 8kB 数据。在 Checkpointing 阶段把数据块 B′和 C′分别写回原文件的数据块 B 和 C 中。所以两个步骤实际上一共写了 4kB+4kB+8kB=16kB 数据。

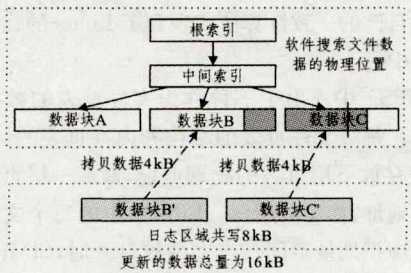

图 3 标准写日志更新

直接把标准写日志方式应用到内存中并不能很好地利用内存的虚拟地址和 CPU 中的硬件 MMU 高速访问文件数据。因为文件系统为了找到数据的物理位置,要用软件搜索文件的元数据结构,会产生较大的开销。例如在 EXT4[17] 文件系统中,需要访问的文件的数据页可能是由三级索引指向的,那么必须逐级查找,以便最终找到数据页的物理位置。

4.2 短路影子分页(SCSP)

为削弱写放大效应,BPFS[3] 基于 SCM 介质可按字节访问和原子性更新的特性提出了 SCSP。当一个写请求到达后,BPFS 执行如下步骤:

1)备份数据。找到一个可以原子性更新的点,把受到影响的数据块中不会覆盖的部分以及中间索引块中的数据拷贝到新分配的物理块中,将本次更新的数据写到新分配的物理块中的对应位置。

2)原子性更新。利用 CPU 可以原子性更新操作的特性,把新的中间索引的指针更新到根节点中对应的位置。

BPFS 利用了 CPU 原子性更新的特点,在整个更新过程中不需要更改标志位,如果在步骤 1)的备份数据阶段出现故障,原文件的数据不受影响;如果系统在步骤 2)出现故障,由于并不能判断步骤 1)是否完成,因此会抛弃本次更新。

由于 SCM 介质具有可按字节访问的特性,因此在整个操作过程中实际更新的数据量受物理页大小的影响较小。如图 4 所示,BPFS 在备份数据阶段把中间索引块的 4kB、数据块 B 的前 3kB 和需要覆盖写的 4kB 数据写到新分配的物理块中,共写 11kB 数据。在原子性更新阶段,用日志区域的中间索引块替换原中间索引块,仅需原子性地修改一个 8byte(64 位系统)大小的指针。因此两个步骤实际上一共写了 11kB 数据。

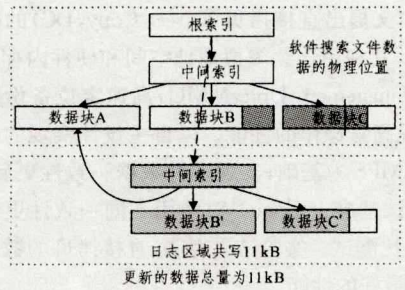

图 4 短路影子分页更新

与标准写日志方式一样,BPFS 中的 SCSP 也是通过软件逐级搜索文件元数据结构的方式查找文件数据的物理位置,也有较大的性能开销。

4.3 自适应日志

SCSP 技术在更新数据块 B 的数据时,真正需要更新的数据量只有 1kB,而实际写了 4kB 数据。为进一步减少这种额外的更新操作,Shortcut-JFS 提出了自适应日志(AL)技术。AL 技术维护一致性的思路与标准写日志文件系统基本相同,只是更好地利用了 SCM 介质按字节读写的特性。

当一个写请求到达后,如果单个数据页上更新的数据量小于页大小的一半,就在 WAL 阶段备份更新的数据,在 Checkpointing 阶段再拷贝一遍备份的数据;如果写请求大于页大小的一半,就在 WAL 阶段将原来块上不会覆盖的数据和本次更新的数据分别写到新分配的物理块,在 Checkpointing阶段用新的物理块替换原来的物理块。

如图 5 所示,在 WAL 阶段,Shortcut-JFS 将数据块 B 更新的 1kB 数据和数据块 C 更新的 3kB 数据分别写到新分配的物理块 B′和 C′中,共更新 4kB 数据;在 Checkpointing 阶段,将数据块 B′中备份的 1kB 数据写回数据块 B 中,用数据块 C′替换数据块 C。这样,总共要写的数据量大小为 4kB+1kB=5kB。

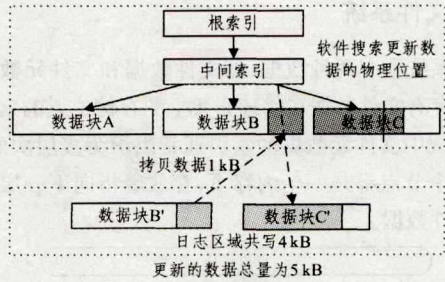

图 5 自适应日志更新

这 3 种技术都通过备份数据的方式维护文件数据的一致性。其中,SCSP 和 AL 这两种技术都面向内存文件系统,它们都利用了 SCM 介质可以按字节访问的特点,尽可能地减少需要更新的数据量。

然而,这 3 种方式都没有很好地利用内存的虚拟地址空间和 CPU 的高速硬件 MMU。实际上,本文实验表明,给文件提供连续的虚拟地址空间后,相较于用软件方式查找数据的物理位置,用硬件 MMU 和连续的虚拟地址更新数据最大可以提升 90%的性能。因此,面向内存文件系统的数据一致性更新机制要利用硬件 MMU 来实现文件数据的高速更新。

5　基于连续地址空间的直接拷贝一致性更新机制

5.1　系统结构

Linux 文件系统一般包括超级块、索引节点(inode)和文件数据 3 部分。其中，索引节点存储了每个文件的元数据信息。为支持直接拷贝(DC)的一致性数据更新，本文在文件系统中增加了临时索引节点区域和记录临时索引节点使用情况的位图，如图 6 所示。

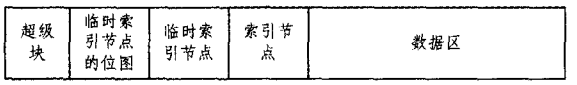

图 6　支持直接拷贝的内存文件系统结构

临时索引节点的结构在普通索引节点的结构之外增加了原文件索引节点号、起始写位置、写长度和标志位 4 个字段，分别用于记录写请求的原文件、更新数据的起始位置和长度以及写请求的完成度。

每当系统以可写方式打开一个原文件时，就为它分配一个临时索引节点来创立临时文件，并把该临时文件映射到连续的内核虚拟地址空间，用来备份数据。临时文件和 SIMFS 中的普通文件一样，也用文件页表组织文件数据。建立映射后，可以通过临时文件的虚拟地址空间和 CPU 中的硬件 MMU 定位其中的任意数据，不需要通过软件方式搜索文件数据的组织结构，从而提高了文件数据的读写性能。在文件关闭前，可以在原文件的多次写操作过程中重复利用它的临时文件；关闭原文件时回收临时索引节点并销毁映射。

5.2　更新算法

直接拷贝把写操作分为 WAL 和 Checkpointing 两个阶段。

1)在 WAL 阶段，首先把原文件的索引号、在原文件中更新数据的起始位置和长度记录到临时索引节点中，并把标志位设为 PENDING 状态。然后把新数据和受影响的文件元数据分别写入临时文件和临时索引节点。在更新数据的过程中，可以使用连续的虚拟地址空间一次性地高速完成写操作，不需要通过软件方式查找文件数据页的元数据结构。数据写完后，把标志位设为 COMMIT 状态。这时预写日志可以保证临时索引节点中有一份本次更新的一致性数据。

2)在 Checkpointing 阶段，从原文件的写请求起始位置开始，使用原文件和临时文件的连续虚拟地址空间把临时文件中的新数据一次写入原文件。完成后把标志位设为 CHECKPOINT 状态。

一方面，直接拷贝的写数据量远少于标准写日志。以第 4.1 节中的图 3 为例，直接拷贝只需要写两次新数据，共 8kB。另一方面，直接拷贝不需要软件查找文件数据的组织结构，实际的延时开销远小于自适应日志。本文实验结果也验证了这一现象，说明基于文件虚拟地址更新数据的一致性方法优于基于单纯减少更新数据量的一致性方法。

5.3　系统恢复

在系统恢复时，通过检查索引节点中的标志位来决定恢复策略。如果标志位为 PENDING 状态，说明本次数据更新操作的备份工作尚未完成，原文件不受系统故障的影响，此时只需要重置并回收该临时索引节点。如果标志位为 COMMIT 状态，说明本次数据更新操作的备份工作已经完成，此时文件系统根据临时索引节点中存储的原文件号、更新起始位置和长度，把临时文件中的新数据重新写回原文件的相应位置，最后重置并回收该临时索引节点。如果标志位为 CHECKPOINT 状态，说明此时数据更新已经顺利完成，直接重置并回收该临时索引节点。

6　实验结果分析与优化建议

本节在 SIMFS 内存文件系统中实现各种一致性机制并测试文件系统的整体性能，通过对比文件系统的整体性能，分析各种数据一致性机制对内存文件系统的性能影响。实验结果表明，使用直接拷贝的一致性机制可以获取最高的性能。

6.1　实验环境与实验方法

本实验使用的计算机是 Dell Precision T5610 Workstation，系统配置如表 1 所列。实验划分 64 GB DRAM 用于模拟 NVM，将 SIMFS 挂载在模拟的 NVM 上进行实验。

表 1　测试系统配置

系统组件	配置情况
CPU 核心	Intel Xeon Processors (E5-2650 @2.60GHz)，16 线程
一级缓存	32kB
二级缓存	2560kB
三级缓存	20MB
内存	128GB 1866MHz
磁盘	1TB，转速为 7200rpm，SATA 接口
内核	Linux 3.11.8

本文中无一致性机制的 SIMFS 基础版本(记为 NC-SIMFS)在 Linux 3.11.8 版本的系统中实现。为比较各种一致性算法的性能，本文在 NC-SIMFS 版本的基础上分别实现支持标准写日志(记为 SL)，SCSP，AL 和 DC 4 种数据一致性更新机制的 SIMFS 版本，然后用标准的测试基准 IOZone[7] 测试以上 5 个版本的 SIMFS 的性能。

测试包含 4 种写文件方式，分别是初始化写、重写、随机写和用户缓冲写。初始化写是指分配新数据页并顺序写数据；重写是指在已有数据页上顺序写数据；随机写是指在已有数据页上乱序写数据；与初始化写操作一样，用户缓冲写也需要分配数据页并顺序写，但它是先把数据缓存起来，满足一定条件后再把数据写到文件中。为展示 4 种写方式的侧重点，本文还在 Ramdisk 上测试经典的 EXT4[17] 文件系统在 4 种写方式下的性能，并以此作为对比实验。

6.2　实验结果分析与优化建议

6.2.1　4 种写操作的异同

表 2 列出了基于 Ramdisk 的 EXT4 在 4 种写操作方式下的带宽。

表 2　基于 Ramdisk 的 EXT4 带宽/MB/s

I/O 大小/k	初始化写操作	随机写操作	重新写操作	用户缓冲写操作
1	324.9	387.1	469.3	741.0
2	553.0	612.3	708.6	823.9
3	601.8	696.2	847.6	909.8
4	884.3	969.2	1157.5	1039.2
8	1041.2	1161.2	1330.1	1213.1

表中 I/O 大小表示单次 I/O 请求的数据量。重写的性能优于初始化写的性能，这是因为虽然初始化写和重写都属于顺序写，但初始化写操作要写的文件并不存在，在写的过程中

需要不断地为要写的数据分配数据页，而 EXT4 的数据页分配方式是为块设备设计的，所以这种写操作方法最慢。

随机写操作不需要分配数据页，因此相对初始化写操作，其性能有了显著的提升。但由于其在写的过程中需要不断改变写指针的位置，因此其写带宽低于重写。用户缓冲写的性能高于初始化写，甚至在单次 I/O 大小较小时性能高于重写，这是因为用户缓冲写通过缓存数据减少了 I/O 次数，降低了 I/O 层次的软件开销并减少了分配数据页次数。

6.2.2 直接拷贝写一致性效果分析

表 3 展示了无一致性版本的 SIMFS(记为 NC-SIMFS)、使用直接拷贝更新一致性机制的 SIMFS(记为 DC-SIMFS)和基于 Ramdisk 的 EXT4(记为 EXT4-Ramdisk)在 4 种写操作方式下的性能。

两个版本的 SIMFS 在 4 种写方式的每个 I/O 大小上的性能都要远高于 EXT4 文件系统。这是因为 NC-SIMFS 直接使用连续虚拟地址空间来访问文件数据，DC-SIMFS 直接使用虚拟地址空间来备份文件数据，可以利用硬件 MMU 直接找到文件数据的物理地址。而 EXT4 访问文件数据时为了找到数据的物理位置，必须首先使用软件方式查找文件元数据，这会导致较大开销。与 NC-SIMFS 相比，DC-SIMFS 的性能下降大都在 5%左右，这表明内存文件系统中 I/O 的软件层次的开销对文件系统的性能影响较大。

表 3 直接拷贝写性能对比/MB/s

I/O 大小/kB	初始化写操作			随机写操作			重新写操作			用户缓冲写操作		
	NC_SIMFS	DC_SIMFS	Ext4_Ramdisk	NC_SIMFS	DC_SIMFS	Ext4_Ramdisk	NC_SIMFS	DC_SIMFS	Ext4_Ramdisk	NC_SIMFS	DC_SIMFS	Ext4_Ramdisk
1	1192.3	1197.6	324.9	1032.2	983.1	387.1	1233.9	1178.2	469.3	1161.1	1108.7	741.0
2	2059.9	2056.6	553.0	1903.7	1841.6	612.3	2232.6	2122.2	708.6	2081.8	2008.8	823.9
3	2738.6	2734.4	601.8	2707.2	2573.2	696.2	3101.6	2914.4	847.6	2891.3	2751.9	909.8
4	3292.1	3291.3	884.3	3369.3	3187.3	969.2	3750.4	3570.1	1157.5	3563.6	3365.2	1039.2
8	4667.1	4637.6	1041.2	5324.2	4807.2	1161.2	5862.2	5266.4	1330.1	5304.0	4557.0	1213.1

与 EXT4 不同，SIMFS 的初始化写、重写与用户缓冲写操作性能相差较小，它们都大于其随机写操作，这是因为 SIMFS 优化了写操作所需数据页的分配方法，这也是用户缓冲写操作相对初始化写操作在性能上没有明显提升的原因。

6.2.3 4 种一致性更新机制效果对比

图 7 所示为基于 NC-SIMFS 实现的 4 种一致性更新机制在不同写方式下的性能结果。

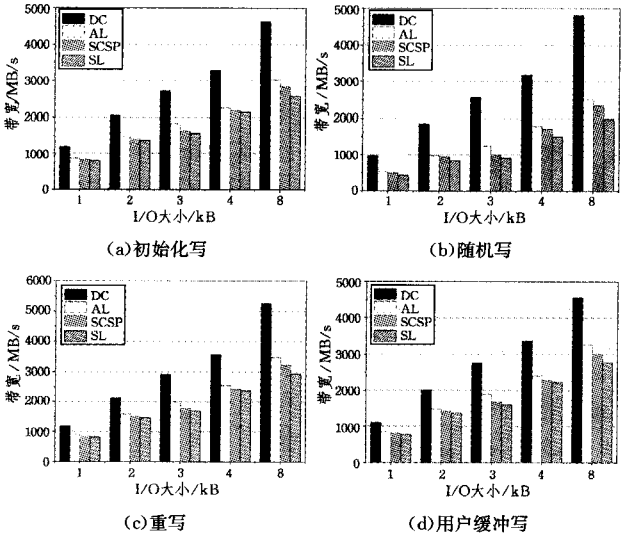

图 7 4 种写一致性机制在不同写方式中的性能

直接拷贝的 SIMFS 版本(DC)在所有测试例中的性能都优于其他 3 种一致性版本的 SIMFS 文件系统。某些情况下 DC 的带宽甚至分别达到了 AL，SCSP 和 SL 的 1.1 倍、1.6 倍和 1.9 倍，最差的情况也要比它们快 30%以上。这是因为 DC 利用了文件虚拟地址空间和硬件 MMU 直接定位到待更新数据的物理位置，避免了软件方式查找元数据结构的开销。该实验表明写操作速度达到内存级别时，写操作路径中需要经历的各种软件操作极易成为限制内存文件系统的性能瓶颈。

对于全部的 4 种写方式，SL，SCSP，AL 3 种一致性更新机制的带宽在同一 I/O 大小上的性能是逐渐增加的，但增幅较小。这是因为 AL 和 SCSP 都利用了 SCM 介质可以按字节访问的特点，通过尽可能地减小实际写的数据量来提升文件系统的性能，通过第 3 节的分析可知这三者每次更新时实际写的数据量是依次递减的。

结束语 本文对比分析了 3 种典型的数据一致性更新机制，包括标准写日志技术和 2 种对内存文件系统优化后的数据一致性更新机制，即 SCSP 和 AL 技术。分析发现，这 3 种基于 SCM 介质可以按字节访问的机制在更新数据时都不能很好地利用虚拟地址空间。因此，提出了针对内存文件系统的直接拷贝的数据一致性更新机制。

本文分别实现了使用 4 种数据一致性更新机制的内存文件系统 SIMFS 版本，并且使用标准测试基准分别测试 4 个版本的性能。实验结果表明，直接拷贝的数据一致性更新机制能够带来最高的性能。分析发现，使用连续虚拟地址空间的直接拷贝机制能够充分利用硬件 MMU 来加速数据更新操作，从而达到比其他使用软件方式查找更新数据的一致性更新机制更高的性能。

参 考 文 献

[1] FREITAS R F, WILCKE W W. Storage-class memory: the next storage system technology[J]. Ibm Journal of Research & Development, 2008, 52(4): 439-447.

[2] WU X, QIU S, NARASIMHA REDDY A L. SCMFS: A File System for Storage Class Memory and its Extensions[J]. ACM Transactions on Storage (TOS), 2013, 9(3): 1-11.

[3] CONDIT J, NIGHTINGALE E B, FROST C, et al. Better I/O through byte-addressable, persistent memory[C]// Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009: 133-146.

[4] DULLORT S R, KUMAR S, KESHAVAMURTHY A, et al. System software for persistent memory[C]// Proceedings of the

Ninth European Conference on Computer Systems. ACM, 2014: 1-15.

[5] SHA H M, CHEN X, ZHUGE Q, et al. Designing an efficient persistent in-memory file system[C]// IEEE Non-Volatile Memory System and Applications Symposium (NVMSA). IEEE, 2015.

[6] LEE E, HOON YOO S, BAHN H. Design and Implementation of a Journaling File System for Phase-Change Memory[J]. IEEE Transactions on Computers, 2015, 64(5): 1349-1360.

[7] NORCOTT W D, CAPPS D. Iozone filesystem benchmark[J/OL]. http://www.iozone.org.

[8] REDEH O, BACIK J, MASON C. BTRFS: The Linux B-tree filesystem[J]. ACM Transactions on Storage (TOS), 2013, 9(3): 317-318.

[9] PRABHAKARAN V, ARPACI-DUSSEAU A C, ARPACI-DUSSEAU R H. Analysis and Evolution of Journaling File Systems [C]// USENIX Annual Technical Conference. General Track, 2005: 105-120.

[10] ZHENG L C, SUN Y L. The Implementation of Journaling File system on Embedded Memory Device [J]. Computer Science, 2002, 29(1): 72-74. (in Chinese)
郑良辰，孙玉芳. 日志文件系统在嵌入式存储设备上的实现[J]. 计算机科学，2002，29(1)：72-74.

[11] JOSEPHSON W K, BONGO L A, LI K, et al. DFS: A file system for virtualized flash storage[J]. ACM Transactions on Storage (TOS), 2010, 6(3): 37-47.

[12] ROSENBLUM M, OUSTERHOUT J K. The design and implementation of a log-structured file system[J]. ACM Transactions on Computer Systems (TOCS), 1992, 10(1): 26-52.

[13] WAN H, XU Y C, YAN J F, et al. Mitigating Log Cost through Non-Volatile Memory and Checkpoint Optimization [J]. Journal of Computer research and Development, 2015, 52(6): 1351-1361. (in Chinese)
万虎，徐远超，闫俊峰，等. 通过非易失存储和检查点优化缓解日志开销[J]. 计算机研究与发展，2015，52(6)：1351-1361.

[14] LI T, LIANG H L. Design and Implement of Event-recovery File System [J]. Computer Science, 2009, 36(3): 270-272. (in Chinese)
李涛，梁洪亮. 具有事件恢复功能的文件系统的研究与实现[J]. 计算机科学，2009，36(3)：270-272.

[15] RAOUX S, BURR G W, BREITWISCH M J, et al. Phase-change random access memory: A scalable technology[J]. Ibm Journal of Research & Development, 2008, 52(4/5): 465-480.

[16] PARKIN S S P, MASAMITSU H, LUC T. Magnetic Domain-Wall Racetrack Memory[J]. Science, 2008, 320(5873): 190-194.

[17] MATHUR A, CAO M, BHATTACHARYA S, et al. The new ext4 filesystem: Current status and future plans[C]// Proceedings of the Linux Symposium. 2007.

(上接第 208 页)

[2] JIN Y M, WU Q Y, SHI Z Q, et al. RFID Lightweight Authentication Protocol Based on PRF[J]. Journal of Computer Research and Development, 2014, 51(7): 1506-1514. (in Chinese)
金永明，吴棋滢，石志强，等. 基于 PRF 的 RFID 轻量级认证协议研究[J]. 计算机研究与发展，2014，51(7)：1506-1514.

[3] LU L. Wireless Key Generation for RFID System[J]. Chinese Journal of Computers, 2015, 38(4): 822-832. (in Chinese)
鲁力. RFID 系统密钥无线生成[J]. 计算机学报，2015，38(4)：822-832.

[4] WANG L M, YI X L, LV C, et al. Security Improvement in Authentication Protocol for Gen-2 Based RFID System[J]. JCIT, 2011, 6(1): 157-169.

[5] ZHANG J S, WANG W D, MA J, et al. A Novel Authentication Protocol suitable to EPC Class 1 Generation 2 RFID system[J]. JCIT, 2012, 7(3): 259-266.

[6] AVOINE G, LAURADOUX C, MARTIN T. When Compromised Readers Meet RFID[C]// Workshop on Information Security Applications (WISA'09). 2009: 36-50.

[7] LU L, HAN J, HU L, et al. Dynamic key-updating: Privacy-preserving authentication for rfid systems[C]// Fifth Annual IEEE International Conference on Pervasive Computing and Communications. 2007: 13-22.

[8] SONG B, MITCHELL C J. RFID authentication protocol for low-cost tags[C]// Proceedings of the First ACM Conference on Wireless Network Security. 2008: 140-147.

[9] LO N W, YEH K H. An efficient mutual authentication scheme for epcglobal class-1 generation-2 rfid system[C]// Proceedings of the 2007 Conference on Emerging Direction in Embedded and Ubiquitous Computing. 2007: 43-56.

[10] ALOMAIR B, CUELLAR J, POOVENDRAN R. Scalable RFID systems: A privacy-preserving protocol with constant time identification [J]. IEEE Trans on Parallel and Distributed Systems, 2012, 23(8): 1-10.

[11] GODOR G, IMRE S. Hash-based mutual authentication protocol for low-cost RFID systems[C]// Proc of the 18th EUNICE Conf on Information and Communications Technologies. Berlin: Springer, 2012: 76-87.

[12] MAMUN M S I, MOUAKJI A, RAHMAN M S. A secure and private RFID authentication protocol under SLPN problem[C]// Proc of the 6th Int Conf on Network and System Security. Berlin: Springer, 2012: 476-489.

[13] WANG S H, LIU S J, CHEN D W. Scalable RFID Mutual Authentication Protocol with Backward Privacy [J]. Journal of computer Research and Development, 2013, 50(6): 1276-1284. (in Chinese)
王少辉，刘素娟，陈丹伟. 满足后向隐私的可扩展 RFID 双向认证方案[J]. 计算机研究与发展，2013，50(6)：1276-1284.

[14] BOLOTNYY L, ROBINS G. Physically unclonable function-based security and privacy in RFID systems[C]// Fifth Annual IEEE International Conference on Pervasive Computing and Communications. 2007: 211-220.

[15] KULSENG L, YU Z, WEI Y, et al. Lightweight mutual authentication and ownership transfer for rfid systems[C]// Proceedings of the 29th Conference on Information Communications. 2010: 251-255.

[16] KARDAS S, AKGUN M, KIRAZ M S, et al. Cryptanalysis of Lightweight Mutual Authentication and Ownership Transfer for RFID Systems[C]// Workshop on Lightweight Security & Privacy: Devices, Protocols, and Applications. 2011: 20-25.