DOI:10.7544/issn1000-1239.2018.20180294
55(9):2029-2037,2018

一种基于微日志的持久性事务内存系统

陈 娟[1] 胡庆达[2] 陈游旻[2] 陆游游[2] 舒继武[2] 杨晓辉[1]
[1](东南大学信息科学与工程学院 南京 210093)
[2](清华大学计算机科学与技术系 北京 100084)
(chenmj09@163.com)

A Tiny-Log Based Persistent Transactional Memory System

Chen Juan[1], Hu Qingda[2], Chen Youmin[2], Lu Youyou[2], Shu Jiwu[2], and Yang Xiaohui[1]
[1](*School of Information Science and Engineering, Southeast University, Nanjing* 210093)
[2](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract In recent years, in order to exploit the performance advantage of persistent memory, researchers have designed various lightweight persistent transactional memory systems. Atomicity and consistency of transactions are mostly ensuredusing the logging mechanism. However, compared with conventional memory, memory cells of persistent memory tend to have higher write latency and limited endurance. This paper observes two problems in the existing persistent transactional memory systems: Firstly, existing systems do not distinguish between different types of write operations in the transaction. No matter whether the writes are updating existing data in memory or adding data to newly allocated memory, existing systems use the same logging mechanism to ensure consistency. Secondly, existing systems persist the address and data of every write operation to the log, even if most of them can be compressed to reduce the size. Both of the above problems lead to redundant log operations, resulting in extra write latency and write wearing. In order to solve the above problems, this paper designs and implements TLPTM, a tiny-log persistent transactional memory system. It is based on two optimization techniques: (1) AALO (allocation-aware log optimization), effectively avoids the logging overhead generated by the operations of adding data to newly allocated memory; (2) CBLO (compression-based log optimization), compresses the log before writing it to the NVM and reduces the overhead of logwriting. The experimental results show that compared with Mnemosyne, AALO improves the system performance by 15%~24%, and TLPTM using both optimizations reduces the write wearing of logging by 70%~81%.

Key words persistent memory; transaction memory system; performance; endurance; logging mechanism

摘 要 近年来,研究者们针对持久性内存良好的性能,设计了轻量级的持久性事务内存系统,它通过日志机制保证了事务的原子性和一致性.然而,相比于传统内存,持久性内存的存储单元往往具有更高的写延迟,并且存在有限的耐久性.发现现有的持久性事务内存系统存在日志机制带来过多的写操作问题:一方面,现有系统没有区分出事务中不同类型的写操作,即无论是对内存中已有数据的更新操作还

收稿日期:2018-04-16;**修回日期**:2018-06-11
基金项目:国家自然科学基金项目(61772300)
This work was supported by the National Natural Science Foundation of China (61772300).
通信作者:舒继武(shujw@tsinghua.edu.cn)

是向事务中新分配区域添加数据的写操作,现有系统都采用相同的日志机制保证它们的一致性;另一方面,现有系统将更新操作的地址和数据等字段完整地持久化到日志中,即使其中大部分数据都可以通过压缩算法减少写入量.这2方面导致了冗余的日志操作,带来了额外的写延迟和写磨损.为了解决上述问题,设计并实现了一种基于微日志的持久性事务内存系统TLPTM,主要提出2个优化技术:1)分配操作感知的日志优化策略(allocation-aware log optimization, AALO),AALO有效地避免了向事务中新分配区域添加数据的写操作产生的日志开销;2)基于压缩算法的日志优化策略(compression-based log optimization, CBLO),CBLO将日志数据压缩后再写入到日志中,减少了日志操作的写开销.测试结果表明:相比于Mnemosyne,提出的日志优化策略AALO将事务性能提高了15%~24%,基于提出的2种优化技术实现的TLPTM将日志的写入总量降低了70%~81%.

关键词 持久性内存;事务内存系统;性能;耐久性;日志机制

中图法分类号 TP391

近年来,新型非易失性存储器(non volatile memory, NVM)吸引了学术界和工业界的广泛关注.典型的非易失存储器包括PCM[1-2],STTRAM[3-4],ReRAM[5]和3D XPoint[6],其不仅具有接近传统内存材料随机访问存储器(dynamic random access memory, DRAM)的随机访问性能,而且在内存层次提供了数据持久性的保证.因此,基于NVM的持久性内存打破了传统易失性内存和持久性外存之间的界限,为设计更加高效的存储系统提供了新的机遇.

虽然持久性内存保证了数据的持久性,然而,CPU缓存依然是易失性的,并且会打乱数据写回到持久性内存的顺序,这导致持久性内存上的数据在系统突然崩溃时可能处于不一致的中间状态.为解决此问题和充分发挥出持久性内存良好的性能优势,研究者们提出了持久性事务内存系统.这些系统通过日志机制保证事务的原子性和一致性.其中,Mnemosyne[7]和NVML[8]分别是基于redo log和undo log日志机制的代表性事务内存系统.在更新原数据前,系统分别先将新/旧数据持久化到日志中,从而在系统崩溃时根据日志中存在的数据副本将其恢复到一致性的状态.这些事务系统不仅为应用程序访问持久性内存提供数据一致性的保证,而且可以通过用户态接口直接访问持久性内存上的数据,降低持久化操作的软件延迟.

虽然持久性内存具有集成度高、静态能耗低和数据持久性等优势,但其也存在一定的局限性.首先,持久性内存往往具有读写不对称性,写延迟远高于读延迟.例如PCM的读延迟与DRAM相似,而写延迟几乎是DRAM的10倍[3].此外,持久性内存还具有耐久性的问题.例如PCM只能承受住10^6~10^8次写操作[9-10],STT-RAM虽然在理论上具有较高的耐久性,但在实际测试中发现,其最多也只能承受10^{12}次写操作,无法达到理论中的期望值[9,11].因此,如何减少对持久性内存的写操作,从而减轻写操作带来的延迟和磨损是持久性事务内存系统中亟待解决的重要问题.

然而,本文发现现有持久性事务内存系统的日志机制带来了过多的写操作,主要有2个方面原因:

1) 现有系统没有区分事务中不同类型的写操作,即无论是对内存中已有数据的更新操作还是向事务中新分配区域添加数据的写操作(本文称之为分配操作),现有系统都利用相同的日志机制保证数据的一致性.然而日志操作不仅加快持久性内存设备的磨损速度,而且还需要使用CLFLUSH等指令对缓存中的数据进行强制刷新,增加事务的延迟[12],造成系统性能的下降.

本文发现redo log和undo log这2种日志机制都只是用于保证被更新的旧数据在系统崩溃时存在一个可恢复的数据副本.对于第1次写入的新数据,不需要依赖日志机制来保证数据的一致性,因此这部分日志开销可以被消除.

2) 现有系统使用相同的数据类型(通常是64 b整型)将更新操作的地址和数据等字段完整地持久化到日志中,即使这些字段存储的数据通常可以用更小的内存空间存储.这会产生大量的日志写操作,严重加快持久性内存设备的磨损速度.

本文发现日志操作中大部分数据可以通过压缩算法减少写入的日志数据总量.这种优化策略不仅可以保留原有数据的副本,而且能够减小对持久性内存的写磨损.此外,因为除了系统崩溃后的恢复过程,系统不会再次访问已经写入的日志数据,所以压

缩操作对事务性能的影响可处在一个可控的范围之内.

针对上述2个方面的问题,本文分别提出分配操作感知的日志优化策略(allocation-aware log optimization, AALO)和基于压缩算法的日志优化策略(compression-based log optimization, CBLO). AALO动态识别出事务中不同类型的写操作,并分别使用不同的日志机制保证所有写操作的一致性,避免分配操作产生的日志开销;CBLO在将新数据写入日志时,利用zigzag压缩算法[13]对日志数据进行压缩,进一步减少日志操作的写开销.

本文结合上述2种优化策略,设计并实现基于微日志的持久性事务内存(a tiny-log based persistent transactional memory, TLPTM)系统.测试结果表明:相比于Mnemosyne,TLPTM系统具有更高的耐久度,其将日志的数据总量降低70%~81%,从而更好地提升持久性内存系统的耐久性.

1　相关工作

为保证系统发生故障后能够恢复到一致性的状态,现有的持久性事务内存系统通常采用基于redo log或者undo log的日志机制.例如Mnemosyne[7]和BPPM[14-15]是基于redo log的持久性事务内存系统,而NV-heap[16],NVML[8]和DCT[17]是基于undo log的持久性事务内存系统.Wan等人在文献[18]中对undo log和redo log两种日志机制在持久性事务内存系统中的使用作了深入研究.这2种日志机制的主要区别在于:在将新数据写入持久性内存中的数据区之前,redo log先将新数据持久化到日志中,而undo log先将内存中的原数据持久化到日志中.当系统突然崩溃时,对于那些已经成功提交但未写入到数据区的事务,基于redo log的事务系统使用日志区中的数据重做已经提交的事务,而基于undo log的事务系统使用日志区中的数据执行回滚操作.无论是redo log还是undo log,两者都是为了在系统突然崩溃时,事务系统可以找到一个正确的数据版本,将数据恢复到一致性的状态.本节具体介绍近年来具有代表性的持久性事务内存系统.

NVML[8]是Intel公司开发的基于undo log的持久性事务内存系统.对于事务中每一个写操作,NVML将对应的旧数据持久化到日志中,然后再去修改原数据.然而,持久性内存会将编程上的错误(例如内存泄露或者访问野指针等)永久性地保存下来,即使在系统重启后依然可能导致系统崩溃.Coburn等人设计了NV-heaps[16],提出了一套灵活健壮的编程模型,以帮助程序员减少编程过程中可能引入的错误.

在基于undo log的事务系统中,每一次写操作需要先将旧数据持久化到日志中,然后才能修改原数据区,这将导致频繁的CLFLUSH操作,带来过高的持久化延迟.Kolli等人设计了DCT[17],该设计通过硬件的方式减少持久化操作之间的顺序依赖关系,从而充分发挥存储系统的并发处理能力,减少持久化延迟对系统性能的影响.但是,DCT系统不可避免地引入了额外的硬件开销.

基于redo log的事务系统在事务提交时才将数据持久化到日志中,从而有效地减少了事务的持久化操作,并且不需要额外的硬件开销.Mnemosyne[7]是基于TinySTM[19]系统开发的基于redo log的持久性事务内存系统,其主要事务流程如图1所示.Mnemosyne在事务执行过程中将所有更新数据缓存在易失性的写集合中.当事务提交时,系统首先将写集合中所有数据持久化到日志中,然后再将这些数据持久化到原数据区.

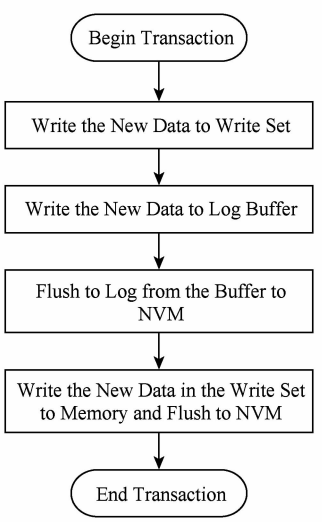


Fig. 1　The transaction mechanism process of Mnemosyne

图1　Mnemosyne的事务机制

Lu等人提出了BPPM[14-15],BPPM在完成日志操作后不需要将数据立即持久化到原数据区,只有当日志空间不足时才将数据持久化到原地址,从而有效地降低了事务的延迟.此外,BPPM还设计了模糊持久化技术,该技术能够区分出日志区中尚未提交的数据,从而避免了在CPU缓存中维护多个数据副本的额外开销.

Sun 等人提出 DP2[20],DP2 技术区分出日志写入和数据写入,在日志写入时通过持久性屏障和减少日志的保存时间来降低写操作的延迟,DP2 在一定程度上降低了事务的开销,并且提高了持久性内存设备的使用寿命.

虽然上述工作在一定程度上提高了事务的性能,但是它们所使用的日志机制不可避免地引入了引言中所提到的 2 个问题而带来了额外的写开销.

2 TLPTM 架构设计

本文提出一种基于微日志的持久性事务内存系统.图 2 描述 TLPTM 的架构图.本文使用支持 DAX[8]功能的文件系统 PMFS[21],通过内存映射文件的方式将持久性内存以用户态的形式暴露给 TLPTM,从而降低传统持久化路径的软件栈开销.上层应用通过 TLPTM 的事务接口访问和更新持久性内存上的数据.

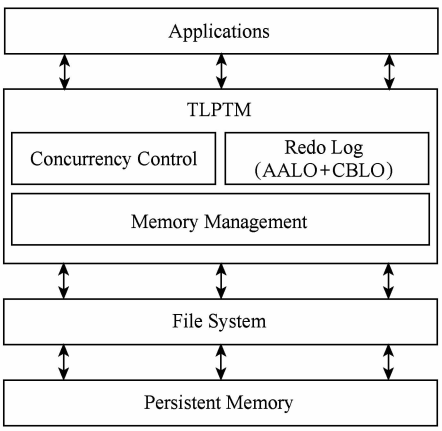


Fig. 2 Architecture of TLPTM

图 2 TLPTM 架构图

由于 TLPTM 是通过内存映射文件的方式从非易失主存中获取一块内存区间,然后在用户态保护程序在这块内存空间上的数据一致性,因此 TLPTM 只保证程序级别的一致性.而内存映射文件所涉及到的文件的一致性由文件系统保证,例如 PMFS,HiNFS[22]、支持 DAX 功能的 EXT4 文件系统等.

TLPTM 主要包含 3 个模块:

1) 日志模块.第 1 节提到,因为 undo log 导致了频繁的持久化操作,所以日志模块采用了基于 redo log 的日志机制,在事务执行过程中先将新数据维持在易失性的写集合中,只有在事务提交时会触发持久化操作.此外,TLPTM 利用 CLFLUSH 等软件指令保证持久化操作的顺序性.

2) 并发控制模块.在事务执行过程中,任何 2 个事务可能同时更新同一个数据,TLPTM 利用并发控制模块解决事务间的冲突.通过日志模块和并发控制模块,TLPTM 保证了事务的 ACID 特性.

3) 内存管理模块.TLPTM 通过文件系统的 mmap 接口,将持久性内存区域映射到进程某个固定的虚拟地址,然后利用内存管理模块管理映射出的内存区域,保证内存分配/释放操作的持久性和一致性.

为了解决引言中提到的 2 个问题,TLPTM 在日志模块中设计了分配操作感知的日志优化策略 AALO 和基于压缩算法的日志优化策略 CBLO.AALO 在事务运行过程中区分出内存更新操作和分配操作.对于分配操作,绕开了日志操作,直接将新数据写到新分配到的地址上,减少了分配操作的日志开销.CBLO 在将日志数据写入到持久性内存前,先利用压缩算法对日志数据进行压缩,减少写入日志的数据大小,从而减少对持久性内存设备的写磨损.

3 关键技术

本节首先介绍 2 个关键日志优化策略 AALO 和 CBLO,然后介绍基于上述优化策略的 TLPTM 系统的事务机制.

3.1 分配操作感知的日志优化策略 AALO

事务的写操作主要包含 2 个类型:对持久性内存中已有数据的更新操作;向事务中新分配区域添加数据的分配操作.在现有的持久性事务内存系统中,无论是更新操作还是分配操作产生的新数据,都采用的是相同的日志机制,即将新/旧数据先写入到日志中,再写入到原数据区.

对于更新操作,系统需要利用日志机制保证其崩溃时存在可恢复的数据副本.然而对于分配操作,系统只需要将新数据直接写入到新分配的数据地址,并不需要提前写日志的操作,因为这部分数据之前在系统中并不存在,即使系统在事务提交前突然崩溃,也不会影响其一致性状态.

因此,TLPTM 设计了分配操作感知的日志优化策略 AALO.在事务执行过程中,AALO 区分出分配操作和更新操作.当应用在事务中新分配某块持久性内存区域时,TLPTM 记录下新分配的数据区域地址.当事务中有写操作发生时,TLPTM 根据事务中新分配的数据区域地址,判断出每个写操作

是否落在新分配的数据区域中,如果是,则说明当前操作是分配操作,否则就判定其为更新操作.

然后,TLPTM 根据写操作类型的不同,使用不同的日志机制来保证事务的一致性;当事务中的写操作是对内存中已有数据作修改时,TLPTM 使用 redo log 保证数据的一致性.而当事务中的写操作是向该事务新分配的区域添加数据时,TLPTM 直接在事务提交时将数据持久化到新分配的数据区.因此,TLPTM 消除了分配操作带来的日志开销.

3.2　基于压缩算法的日志优化策略 CBLO

本文观察到日志数据往往具有如下特点:对于每个写操作,日志中记录的信息主要包含 3 个字段,即该操作更新的内存地址 addr、新数据 value 以及掩码 mask,它们都是 8 B 的整型数据.现有系统将这 8 B 的数据完整地持久化到日志中,即使这些字段存储的数据通常可以用更小的数据类型表示.因此,TLPTM 设计了基于压缩算法的日志优化策略 CBLO,该策略将日志数据进行压缩以减少写入的日志数据总量,在保证原有数据副本的同时减少对持久性内存的写操作.CBLO 采用 zigzag 压缩算法[13],使用该压缩算法的原因主要有 2 个:①日志中记录的数据类型都是 64 b 的整型数据,而 zigzag 算法十分适合整型数据的压缩;②因为日志数据中 addr 和 mask 等字段在大多数情况下数值大小比较固定且数值较小,而 zigzag 算法对于小整数的压缩效果尤其明显.

zigzag 算法的基本思想是:将有符号的数字映射为无符号的数字,其以一种在正整数与负整数之间来回交错的方式进行,绝对值小的数字经 zigzag 压缩编码后,无论正负都可以只使用较少的字节数来表示.

在具体实现中,当系统执行写日志操作时,TLPTM 利用 zigzag 算法对所有需要持久化到日志中的数据进行重新编码,将其表示成比特位信息中具有更多前导 0 的另一个整数.当存储该数据的内存空间有若干个字节的比特位信息都为 0 时,zigzag 算法会尽可能将这些字节丢弃,用更小的内存空间存储剩余的字节信息,最后持久化到内存中的日志区.由此就达到了将日志数据进行压缩的目的.日志数据经压缩后,有效地减少了日志操作的数据总量,从而提高了持久性内存设备的耐久性.

值得注意的是,由于本设计在日志写入的过程中引入了压缩算法,这不可避免地引入了一定的延迟.因为在对数据进行压缩的过程中,使用的是一个循环结构,该循环结构的作用就是将数据的比特位信息中存在 1 的字节提取出来,当数据经 zigzag 进行编码后的比特位信息中具有较多的 1 时,该循环的运行时间就会越长,因此相应地会增加程序的运行时间.但由于日志数据一般只有在系统突然崩溃后需要做故障恢复的情况下才会被访问,压缩后的数据通常不需要反复的解压缩/压缩过程,所以压缩算法所带来的延迟处在一个可控的范围之内.此外,持久性内存具有读写不对称性,压缩算法能够有效降低写入数据的总量,这在一定程度上也降低了事务的平均延迟.这部分的影响会在第 4 节实验部分给出进一步的验证.

3.3　事务机制

3.3.1　事务接口

TLPTM 利用 GCC 事务编译器[23]向传统 C/C++代码加入了事务语义,通过关键字 *__tm_atomic* 和一组花括号定义事务的范围,保证这段代码的 ACID 特性.事务范围内的内存访问操作会被 GCC 事务编译器捕获,TLPTM 利用 TinySTM 替换了事务中的内存访问操作.在事务执行过程中,TLPTM 将所有更新操作记录在一个易失性的写集合中.当事务提交时,通过 redo log 日志机制,将写集合中的写操作备份到日志中,并调用持久化模块保证日志操作的持久性.最后,TLPTM 将新数据持久化到原数据区.此外,TLPTM 利用 TinySTM 的锁机制保证了事务的并发控制,从而实现了并发控制模块,这部分的细节介绍可参考之前的研究工作[19].

3.3.2　日志模块

当事务提交时,TLPTM 利用软件指令对日志操作和原数据写回操作进行持久化.因为日志数据只有在系统崩溃时才会被访问,所以 TLPTM 利用 MOVNT 指令绕开了 CPU 缓存,将日志数据直接持久化到持久性内存上,避免了对 CPU 缓存的干扰.此外,TLPTM 利用 MFENCE 指令,确保在执行原数据修改前将所有日志操作进行持久化.对于原数据写回操作,TLPTM 利用 CLFLUSH 指令保证所有写操作的持久性.此外,日志模块设计了 2 种优化策略 AALO 和 CBLO,解决了引言中提到的现有系统存在的 2 个问题.

3.3.3　内存管理模块

内存管理模块是基于易失性内存分配器 Hoard[24]实现的.Hoard 最初是针对多线程场景设计的,其按照超级块的粒度(例如 16KB)对传统内存进行切割.为了减少多线程间的冲突,Hoard 同时

维护了全局的超级块池和线程局部的超级块池,线程只有在局部超级块池的空间耗尽时,才会向全局超级块池申请新的超级块,从而减少了多线程间的锁冲突.对于每个超级块,Hoard将其分成固定大小的内存块,满足给定区间大小的分配操作.

然而,Hoard是基于传统内存设计的,无法保证分配操作的持久性和一致性.为了解决这个问题,TLPTM在持久性内存的固定区域预留了每个超级块的位图(bitmap).位图中每个比特位都表示了对应超级块中某个内存块的分配状态.此外,TLPTM利用事务接口,保证分配/释放操作中位图修改操作的持久性和一致性.

3.3.4 TLPTM的事务机制流程

图3表示了基于AALO和CBLO的TLPTM系统的事务机制流程图.TLPTM与基于redo log的传统系统相比,主要有2点区别:1)TLPTM增加了一个写操作判断,即判断写操作是否是分配操作;2)TLPTM在执行日志持久化操作前利用zigzag算法对其进行了压缩.

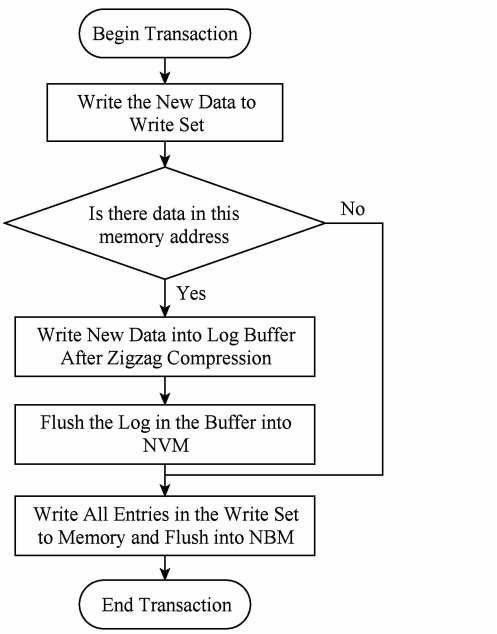


Fig. 3 The transaction mechanism process of TLPTM

图3 TLPTM的事务机制

3.3.5 系统恢复

当系统突然崩溃时,TLPTM需要将系统恢复到崩溃前的一致性状态.首先,TLPTM通过文件系统的mmap接口,将持久性内存区域映射到某个固定的虚拟地址.其次,根据超级块中的位图信息重建内存管理模块的数据结构.最后,根据日志数据中的信息,重做已经提交的事务,丢弃未完成的事务,由此完成整个系统的恢复过程.

4 实验测试与分析

为了分析评价TLPTM的性能和耐久性,本节的实验将从3个方面对TLPTM进行测试与评价:

1) 对比测试分配操作感知的日志优化策略AALO的性能和耐久性;

2) 对比测试基于压缩算法的日志优化策略CBLO的耐久性;

3) 对比测试Mnemosyne和TLPTM的性能和耐久性.

4.1 测试环境

本文所有实验均基于同一台Linux服务器,内核版本为4.4.16,处理器型号为Intel® Xeon® CPU E5-2680 v3,共包含4个物理核心.此外,服务器还配置了32 GB的内存和1 TB的硬盘.实验配置环境如表1所示:

Table 1 Experiment Configuration

表1 实验环境配置

Parameters	Configuration
OS	Linux 4.4.16
CPU	Intel® Xeon® E5-2680, 1.2 GHz
Memory/GB	32

本文使用Mnemosyne[7]中的Hash表作为实验的测试程序,并将每个事务中产生的日志数据的平均字节数和事务平均延迟作为系统在可靠性和性能的评价标准.实验中使用3种不同的负载来评估TLPTM的性能和耐久性.其中,Workload *a* 被设置为在Hash表中全部执行分配操作,Workload *b* 的场景是在Hash表中执行分配操作和更新操作的比例均为50%,Workload *c* 被设置为在Hash表中全部执行更新操作.表2中列出了这3种负载的特点.

Table 2 Workload Features

表2 负载的特点

Workload	Insert/%	Update/%
a	100	0
b	50	50
c	0	100

虽然持久性内存相关技术发展十分迅速,但是目前大多数研究还只是集中在设备级别,对系统级别的研究仍面临着缺少真实的NVM设备的境况.因此,大多数研究不具备在真实NVM存储环境下

验证上层软件技术有效性的条件,而构建 NVM 模拟器或模拟平台是解决上述问题的有效途径[11].本文使用与 Mnemosyne[7] 类似的持久性内存模拟方法,即将普通内存划分为常规的易失性内存和模拟的非易失内存.默认情况下,本实验使用 CPU 提供的 RDTSC 指令将模拟延迟设置为 500 ns.因为 Mnemosyne 和 TLPTM 都是基于 TinySTM[19] 和 redo log 实现的持久性事务内存系统,本文主要以 Mnemosyne 作为比较系统,从而保证了系统对比的公平性.

4.2　分配操作感知的日志优化策略的测试

本节实验将使用了 AALO 的系统与 Mnemosyne 进行对比,验证了分配操作感知的日志优化策略对持久性事务内存系统的可靠性和性能方面的影响.

4.2.1　耐久性测试

日志写入量的测试结果如图 4 所示,除负载 c 外,AALO 将每个事务日志操作产生的平均字节修改数降低了 54%～64%.负载 a 理论上更契合于本文提出的分配操作感知的优化策略,因为负载 a 连续地向 Hash 表中插入新的键值对,这会导致频繁地调用内存分配函数,由于是向内存中添加新数据,该操作不需要通过日志操作来保证一致性.但是,随着插入操作所占比例的增加,每个事务的日志操作产生的平均字节修改数也随之增加,这是因为内存分配信息也会保留在日志中,因而出现图 4 中所示的结果.

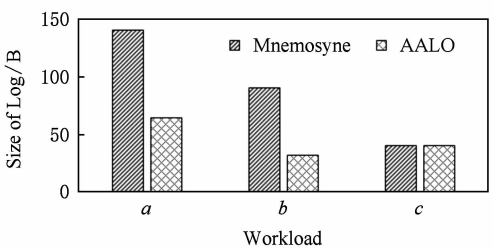


Fig. 4　Comparison of log size between Mnemosyne and AALO

图 4　Mnemosyne 和 AALO 日志写入量的对比

由于负载 c 中每次操作都是更新哈希表中已有的键值对,而本文提出的分配操作感知的优化策略只对分配操作有效,因此对耐久性的提升并不明显.

4.2.2　性能测试

在 3 种工作负载下,本节分别测试了 AALO 对持久性内存系统的性能影响,其与 Mnemosyne 的延迟对比如图 5 所示,除负载 c 外,AALO 将写入延迟降低了 15%～24%.由于负载 a 受到频繁的内存分配操作带来的影响,所以其性能要略差于其他负载.由于负载 c 全部都是更新操作,因此 2 个系统的性能基本相似.

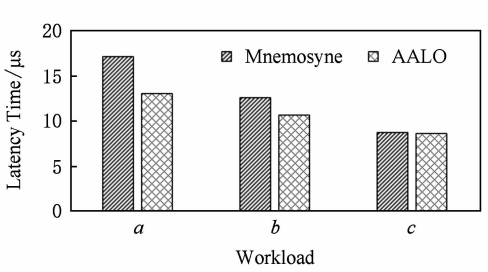


Fig. 5　Comparison of latency between transaction mechanisms

图 5　事务机制的延迟对比

由此可知,分配操作感知的日志优化策略不仅减少了每个事务的写磨损,而且也降低了日志操作引入的持久化延迟.

4.3　基于压缩算法的日志优化策略的测试

本节实验将使用 CBLO 的系统与 Mnemosyne 进行对比,验证了基于压缩算法的日志优化策略对持久性事务内存系统可靠性方面的影响.实验结果如图 6 所示,从图 6 中可以看出,使用了 CBLO 的系统中日志操作的平均字节修改数比 Mnemosyne 减少了61%～70%.磨损大大降低的原因在于,日志操作的所有字段都是 64 b 的整型数据.首先,考虑字段 addr 和 mask,当地址 addr 是 9 b 数的整型数字时,经 zigzag 压缩编码后,只需要使用 5 B 进行存储;而 mask 在多数情况下为－1,经压缩编码后,只占用 1 B.而 value 字段是由应用程序决定的,当传入的 value 较小时也能获得良好的压缩率.因此,相比于 Mnemosyne,CBLO 大大减少了持久性内存系统的日志写入量.

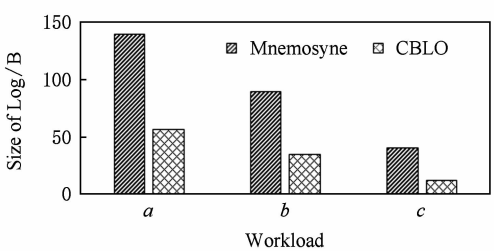


Fig. 6　Comparison of log size between Mnemosyne and CBLO

图 6　Mnemosyne 和 CBLO 日志写入量的对比

此外,压缩算法不可避免地增加了日志操作的延迟,但是因为日志数据只有在系统恢复时才会被访问到,所以压缩算法产生的开销可以接受.本文将在 4.4 节中讨论基于 2 种优化策略的 TLPTM 的性能效果.

4.4 TLPTM 与 Mnemosyne 的对比

本节实验将基于 2 种优化策略的 TLPTM 与 Mnemosyne 进行对比,由此判断 TLPTM 在性能和可靠性方面的效果.

4.4.1 耐久性测试

本实验比较了基于 AALO 与 CBLO 两种优化的 TLPTM 系统与 Mnemosyne 的耐久性. 两者产生的日志写入量对比如图 7 所示. 从图 7 中看出,在不同的工作负载中,与 Mnemosyne 相比,TLPTM 在耐久性上都有十分明显的提升,除负载 c 外,TLPTM 将日志的写入总量降低了 79%～81%. 因为负载 c 不包含分配操作,AALO 并没有对其写磨损产生优化效果,所以其耐久性的提升效果略低于其他 2 种工作负载,但其凭借 CBLO 的优化依然达到了 70%的耐久性提升.

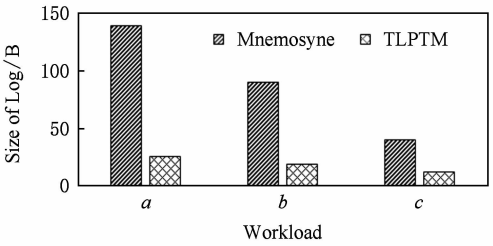


Fig. 7 Comparison of log size between Mnemosyne and TLPTM

图 7 Mnemosyne 与 TLPTM 产生日志数据的大小对比

4.4.2 性能测试

本节实验测试 TLPTM 与 Mnemosyne 的性能对比,结果如图 8 所示,相比于 4.2.2 节中的实验效果,基于 2 种优化的 TLPTM 系统的性能有所降低,这说明压缩算法在降低日志操作产生的写磨损的同时,带来了一定的性能影响. 因为在对数据进行压缩的过程中,使用的是一个循环结构,该循环结构的作用就是将数据的比特位信息中存在 1 的字节全部提取出来,当数据经 zigzag 进行编码后的比特位信息中具有较多的 1 时,该循环的运行时间就会越长,因此相应地会增加程序的运行时间,从而对系统性能造成一定影响. 使用 TLPTM 系统的用户可以根据自己的需求,选择是否使用基于压缩算法的日志优化策略 CBLO,从而获得在性能或者可靠性方面的提升.

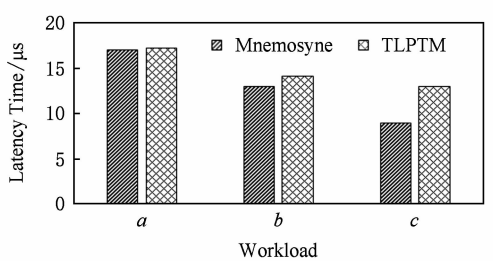


Fig. 8 Comparison of latency between TLPTM and Mnemosyne transaction mechanisms

图 8 TLPTM 与 Mnemosyne 事务机制的延迟对比

5 总 结

持久性内存因为非易失性、静态功耗低和集成度高等优势,有望代替 DRAM 成为新的主存材料. 研究人员设计了轻量级的持久性事务内存系统,用于保证持久性内存上更新操作的一致性. 然而,持久性内存同时也存在着使用寿命有限和写延迟过高的局限性. 现有的事务系统因为日志操作引入额外的写操作,带来严重的写延迟和写磨损等问题.

为了解决这些问题,本文提出了分配操作感知的日志优化策略 AALO 和基于压缩算法的日志优化策略 CBLO. 结合这 2 种优化策略,本文设计并实现了基于微日志的持久性事务内存系统 TLPTM. 实验结果表明,与现有的事务系统 Mnemosyne 相比,本文提出的分配操作感知的优化策略将事务的性能提升了 15%～24%,基于上述 2 种优化策略的 TLPTM 将日志的数据总量降低了 70%～81%.

参 考 文 献

[1] Raoux S, Burr G W, Breitwisch M J, et al. Phase-change random accessmemory: A scalable technology [J]. IBM Journal of Research & Development, 2010, 52(4/5): 465-479

[2] Lee B C, Zhou Ping, Yang Jun, et al. Phase-change technology and the future of main memory [J]. IEEE Micro, 2010, 30(1): 143-143

[3] Akinaga H, Shima H. Resistive random access memory (ReRAM) based on metal oxides [J]. Proceedings of the IEEE, 2010, 98(12): 2237-2251

[4] Krounbi M, Apalkov D, Driskill-Smith A, et al. Spin-transfer torque magnetic random access memory (STT-MRAM) [J]. ACM Journal on Emerging Technologies in Computing Systems, 2013, 9(2): 13: 1-13: 35

[5] Kltrsay E, Kandemir M, Sivasubramaniam A, et al. Evaluating STT-RAM as an energy-efficient main memory alternative [C] //Proc of IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2013: 256-267

[6] Intel Newsroom. Introducing Intel Optane technology-bringing 3D XPoint memory to storage and memory products [OL]. [2016-02-25]. https://newsroom. intel. com/presskits/introducing-intel-optanetechnology-bringing-3d-xpointmemory-to-storage-and-memoryproducts/

[7] Volos H, Tack A J, Swift M M. Mnemosyne: Lightweight persistent memory [C] //Proc of the 16th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011: 91-104

[8] Intel. The NVM Library [OL]. [2014-09-01]. http://pmem.io/

[9] Lu Youyou, Shu Jiwu, Sun Long, et al. Improving performance and endurance of persistent memory with loose-ordering consistency [J]. IEEE Trans on Parallel & Distributed Systems, 2017, PP(99): 1-1

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

[11] Pan Wei, Li Zhanhuai, Du Hongtao, et al. State-of-the-art survey of transaction processing in non-volatile memory environments [J]. Journal of Software, 2017, 28(1): 59-83 (in Chinese)
(潘巍, 李战怀, 杜洪涛, 等. 新型非易失存储环境下事务型数据管理技术研究[J]. 软件学报, 2017, 28(1): 59-83)

[12] Intel Company. Latency and Throughput of Intel CPUs 'clflush' instruction [OL]. [2016-09-23]. https://software.intel.com/en-us/forums/watercooler- catchall/topic/696905

[13] Google Company. Google protocol buffers [OL]. [2017-11-24]. https://developers. google. com/protocolbuffers/docs/encoding

[14] Lu Youyou, Shu Jiwu, Sun Long. Blurred persistence in transactional persistent memory [J]. IEEE Trans on Industrial Electronics, 2015, 61(1): 1-13

[15] Lu Youyou, Shu Jiwu, Sun Long. Blurred persistence: Efficient transactions in persistent memory [J]. ACM Trans on Storage, 2016, 12(1): 1-29

[16] Coburn J, Caulfield A M, Akel A, et al. NV-heaps: Making persistent objects fast and safe with next-generation, nonvolatile memories [C] //Proc of the 16th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011: 105-118

[17] Kolli A, Pelley S, Saidi A, et al. High-performance transactions for persistent memories [C] //Proc of the 21st Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 399-411

[18] Wan Hu, Lu Youyou, Xu Yuanchao, et al. Empirical study of redo and undo logging in persistent memory [C] //Proc of Non-Volatile Memory Systems and Applications Symp. Piscataway, NJ: IEEE, 2016: 1-6

[19] Felber P, Fetzer C, Riegel T. Dynamic performance tuning of word-based software transactional memory [C] //Proc of the 13th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2008: 237-246

[20] Sun Long, Lu Youyou, Shu Jiwu. DP2: Reducing transaction overhead with differential and dual persistency in persistent memory [OL]. [2018-04-01]. http://or. nsfc.gov. cn/handle/00001903-5/417160

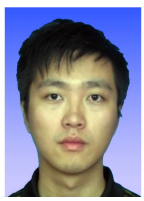
[21] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory [C] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: 15:1-15:15

[22] Chen Youmin, Shu Jiwu, Ou Jiaxin, et al. HiNFS: A persistent memory file system with both buffering and direct-access [J]. ACM Trans on Storage, 2018, 14(1): 1-30

[23] Ni Y, Welc A, Adl-Tabatabai A R, et al. Design and implementation of transactional constructs for C/C++ [J]. ACM Sigplan Notices, 2008, 43(10): 195-212

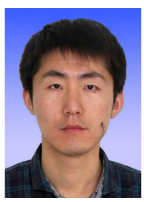
[24] Berger E D, McKinley K S, Blumofe R D, et al. Hoard: A scalable memory allocator for multithreaded applications [C] //Proc of the 9th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2000: 117-128

Chen Juan, born in 1994. Master candidate. Her main research interests include nonvolatile memories and filesystems.



Hu Qingda, born in 1990. PhD candidate. His main research interests include persistent memories and storage systems.

Chen Youmin, born in 1993. PhD candidate. His main research interests include distributed systems and storage systems.



Lu Youyou, born in 1987. Assistant professor. His main research interests include storage systems and distributed systems.

Shu Jiwu, born in 1968. Professor and PhD supervisor. His main research interests include nonvolatile memory systems and technologies, network (cloud/big data) storage systems, storage security and reliability, and parallel and distributed computing.

Yang Xiaohui, born in 1968. Associate professor. Her main research interests include intelligent information processing and information security.