基于新型非易失存储器的混合内存架构的内存管理机制

李 琪[1,2],钟 将[2],李 雪[3],李 青[2]
(1. 淮阴师范学院计算机科学技术学院,江苏淮阴 223300;2. 重庆大学计算机学院,重庆 400044;
3. 昆士兰大学信息技术与电子工程学院,昆士兰州布里斯班 4072)

摘 要: 随着互联网和云计算技术的迅猛发展,现有动态随机存储器(Dynamic Random Access Memory,DRAM)已无法满足一些实时系统对性能、能耗的需求.新型非易失存储器(Non-Volatile Memory,NVM)的出现为计算机存储体系的发展带来了新的契机.本文针对NVM和DRAM混合内存系统架构,提出一种高效的混合内存页面管理机制.该机制针对内存介质写特性的不同,将具有不同访问特征的数据页保存在合适的内存空间中,以减少系统的迁移操作次数,从而提升系统性能.同时该机制使用一种两路链表使得NVM介质的写操作分布更加均匀,以提升使用寿命.最后,本文在Linux内核中对所提机制进行仿真实验.并与现有内存管理机制进行对比,实验结果证明了所提方法的有效性.

关键词: 非易失存储器;磨损均衡;内存计算;混合内存
中图分类号: TP333 **文献标识码:** A **文章编号:** 0372-2112 (2019)03-0664-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.03.021

Memory Management Mechanism for Hybrid Memory Architecture Based on New Non-volatile Memory

LI Qi[1,2], ZHONG Jiang[2], LI Xue[3], LI Qing[2]
(1. *School of Computer Science and Technology, Huaiyin Normal University, Huaiyin, Jiangsu* 223300, *China*;
2. *College of Computer Science, Chongqing University, Chongqing* 400044, *China*;
3. *School of Information Technology and Electrical Engineering, University of Queensland, Brisbane* 4072, *Australia*)

Abstract: With the rapid development of internet and cloud computing technology, the existing dynamic random access memory (DRAM) have been unable to meet the requirements of performance and energy consumption of some real time systems. The emergence of non-volatile memory (NVM) have shown great potential for the development of the computer storage architecture. This paper proposes a kind of efficient memory page management mechanism based on the hybrid NVM and DRAM memory architecture. The main idea of this mechanism is to save the data pages with different access characteristics in the appropriate memory space according to the characteristics of different memory media, to reduce the number of system migration operations and improve the system performance. At the same time, the mechanism uses an efficient wear-leveling algorithm to improve NVM's lifetime. Finally, the experimental results show that the proposed method is effective.

Key words: non-volatile memory; wear leveling; memory computing; hybrid memory

1 引言

目前,随着海量数据分析和挖掘方面的研究日益增多,海量数据存储对现有的主存存储技术在容量上带来了巨大的挑战,同时也对系统的性能提出了更高的要求[1].在传统的存储体系结构中,一般使用DRAM作为主存,使用磁盘或者固态硬盘作为外存.但是这种存储体系结构存在两个问题:(1)DRAM设备存储容量小[2].因为其介质密度较低,在保存数据时需要持续动态刷新,制程工艺已很难再有提升.(2)DRAM介质是

收稿日期:2017-07-27;修回日期:2018-08-24;责任编辑:梅志强
基金项目:国家重点研发计划(No. 2017YFB1402400);重庆市社会事业与民生保障科技创新专项(No. cstc2017shmsA0641);国家863计划项目(No. 2015AA015308);重庆市重点产业共性关键技术创新专项(No. cstc2017zdcy-zdyxx0047);重庆市技术创新与应用示范(产业类重点研发)项目(No. cstc2018jszx-cyzdX0086);中央高校项目(No. 2018CDYJSY0055)

易失性的,其断电以后无法持久化保存数据,因此需要外存对数据进行持久化保存.但是系统需要频繁的在主存和外存之间迁移数据,严重影响了系统性能.

新型非易失性存储器[3](Non-Volatile Memory,NVM)的出现有利于缓解传统存储体系结构中的问题.常见的非易失性存储介质主要包括相变存储(Phase Change Memory)、石墨烯存储器(Graphene Memory)、铁电存储器(Ferroelectric RAM)等.NVM 具有低能耗、非易失性以及高的存储密度等优点.NVM 凭借其按字节存取的特性可以像 DRAM 一样被 CPU 直接访问,这成为计算设备主存的一种新的选择.但 NVM 也存在读写不对称的特性,它的写功耗高,写操作次数有限[4].近几年国内外学者对 NVM 进行了广泛的研究[5],文献[6]将 NVM 作为智能手机操作系统的交换分区来使用,当 NVM 数据页需要被读时,系统可直接进行读操作.而当 NVM 数据页需要被写时,需先将其迁移到 DRAM 中,再进行写操作,最终达到延长 NVM 使用寿命的目的.CLOCK 算法[7]将所有内存数据页保存在一个环形的数组中,并通过检查每个数据页的标记位来选择合适的数据页进行替换.但是 CLOCK 算法只对不频繁访问的冷数据页进行替换,而没有考虑读写特征.NVM 的读写不对称是其很重要的一个特性,有大量研究工作针对 NVM 写次数有限这一缺陷进行了研究,主要内容都是通过磨损均衡算法来优化 NVM 的写分布[8].本文提出了高效的混合内存管理机制(Hybrid Memory Page Management,HyMG),此策略能够有效的提升系统的性能.本文的主要贡献如下所示.

(1)本文对应用程序访问内存数据的过程进行了分析,得出了写操作不均衡这一现象.HyMG 机制就是基于这一特征来设计实现的.

(2)针对 NVM 及 DRAM 的写特性的不同,分别提出了不同的迁移策略.将具有不同访问特征的数据页保存在合适的内存介质中,同时设计了一种两路链表使得 NVM 介质的写操作分布更加均衡.

(3)最后在 Linux 系统上进行了模拟仿真,并与其他内存管理策略分别做了对比,验证了 HyMG 算法的有效性.

2　混合内存管理机制的设计与实现

本节首先介绍混合内存系统的架构,然后通过实验来描述应用程序访问内存数据页时的写不均衡现象.最后详细描述内存数据页在 DRAM 空间和 NVM 空间的迁移过程.

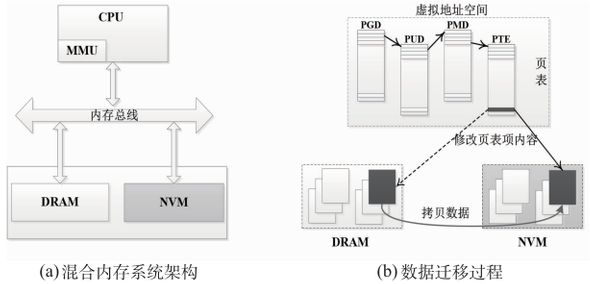
2.1　混合内存系统架构

图 1(*a*)图展示了 NVM 在混合内存中所处的位置,它和 DRAM 挂载在同一内存总线上并共享内存地址空间.图 1(*b*)图展示了用户进程虚拟地址空间到混合内存页框的映射关系,以及内存数据页从 DRAM 页框迁移到 NVM 页框的过程.本文中,我们把数据页从 DRAM 页框迁移到 NVM 页框这一过程称为"DN 迁移"操作,把数据页从 NVM 页框迁移到 DRAM 页框的过程称为"ND 迁移"操作.DN 迁移操作包括如下几个步骤:

(1)在 NVM 空间中分配一个空闲的页框;

(2)选择 DRAM 中的数据页,将其拷贝到新分配的 NVM 页框中;

(3)更新用户进程页表中该数据页对应的页表项内容,将其指向 NVM 页框,并释放原来的 DRAM 页框.



(a)混合内存系统架构　　(b)数据迁移过程

图1　系统架构

ND 迁移操作的过程与 DN 迁移操作过程相反.尽管 DRAM 和 NVM 共享同一地址空间,但是,考虑到介质特性的差异,NVM 空间的分配和回收方法却与 DRAM 空间不同,我们将在第 2.4 小节具体阐述.

2.2　内存数据的访问特征分析

本节研究应用程序访问内存数据页时的特征.实验选用 MiBench[9]中的一组基准测试,它包括了 SHA、CRC32、dijkstra、stringsearch、jpeg、gsm.同时也对第三部分实验中所用到的 fileBench[10]数据集进行了实验.它包括了 fileserver、webproxy、randomrw、videoserver 和 multi-stream-write.设置写入量分别为 5MB,10MB,15MB,20MB,25MB.

我们统计这些应用程序中写操作频率最高 10% 的页框所发生的写操作次数,并计算其所占总写操作次数的比例.其结果如表 1 所示,这些应用程序中大部分的写操作都发生在这 10% 的页框中.应用程序的写操作非常不均匀的落在内存数据页上.我们认为容忍 NVM 页框发生少量的写操作可以减少大量的迁移操作.但是,还必须证明一点,即直接写 1 次 NVM 数据页的开销必须小于将其换到 DRAM 然后再进行写操作的开销.这样我们保留其在 NVM 中直接进行写操作才有意义.在 ND 迁移操作中,一次内存拷贝包括一次读 NVM 数据页和一次写 DRAM 数据页.我们通过在 Linux 内核代码中设置时间戳的方式计算 ND 迁移操作的平均时间开销为 29.9μs.对于系统访问内存数据页的开

销,我们以处理器访问 DRAM 的时延来计算,处理器每次写一个数据页(4KB)需要进行 64 次(4KB/64B = 64)缓存行同步操作. 在我们的实验中,处理器写一个数据页到 DRAM 内存的时间开销为 529.6ns. 假定访问 NVM 介质的访问时延是 DRAM 的 7 倍[4],可得出处理器直接写 1 次 NVM 数据页的开销为 3.7μs. 可以计算得出,ND 迁移操作的开销是直接写 1 次 NVM 数据页开销的 8 倍左右.

表 1 写操作频率最高 10% 页框的写次数

应用程序	写次数	总写次数	比例
SHA	512931	517677	99.08%
CRC32	6846404	6852542	99.91%
Dijkstra	4605201	4634306	99.37%
stringsearch	14729	23874	61.69%
jpeg	5189	12953	40.06%
gsm	56924	65674	86.68%
multi-stream-write	62580	81920	76.39%
fileserver	144890	163840	88.43%
webproxy	194610	245760	79.19%
randomrw	221879	327680	67.71%
videoserver	268756	409600	65.61%

2.3 数据页迁移策略

ND 迁移操作:我们用阈值 T 表示写操作次数的上限. 对于 NVM 空间中的数据页,当对该 NVM 数据页进行读操作时,直接对 NVM 中的数据页进行读取. 而当对 NVM 数据页进行写操作时,首先判断该数据页的写次数是否超过阈值 T,如果没有超过阈值 T 则直接进行写操作,如果超过阈值 T 则先将其迁移到 DRAM 再进行写操作. 假设将数据页从 NVM 页框迁移到 DRAM 页框的开销为 C_{ND},直接写 NVM 数据页的开销为 C_{WN},写这个 NVM 数据页的总开销可以计算如下:

$$C_{total}=\begin{cases}C_{WN}\times i, & i\leqslant T\\ C_{WN}\times T+C_{ND}+C_{WD}(i-T), & i>T\end{cases} \tag{1}$$

其中,C_{total}表示该数据页在其生命周期中总的写开销,i 表示该数据页被写过的次数,C_{WD}表示在 DRAM 中写该数据页的开销. 接下来我们来研究上述两种情况的最优解. 如果最终情况是 $i\leqslant T$,即该数据页的写次数没有超过阈值,那么最优解应该是直接在 NVM 上写这一数据页 i 次;如果最终情况是 $i>T$,最优解应该是在第一次写该数据页时就把它迁移到 DRAM 中,以后的写操作都在 DRAM 中进行,因此,写某一数据页开销的最优解应该为:

$$OC_{cost}=\begin{cases}C_{WN}\times i, & i\leqslant T\\ C_{ND}+C_{WD}\times i, & i>T\end{cases} \tag{2}$$

根据式(1)和(2)可知,当 $i\leqslant T$ 时,ND 迁移策略能够满足最优解,而当 $i>T$ 时,ND 迁移策略所得到的解与最优解之间的比例 R 应该满足:

$$R=1+\frac{C_{WN}-C_{WD}}{C_{ND}+C_{WD}\times i}\times T \tag{3}$$

可以看出,R 是一个关于 i 的单调递减函数,也就是说,最坏的情况下 $i=T+1$. 一个数据页在其生命周期中被写了 $T+1$ 次,前面的 T 次写操作是对 NVM 页框的写操作,当发生第 $T+1$ 次写操作时,系统先将该数据页迁移到 DRAM 页框,然后对其进行写操作,并且在这之后不再进行写操作. 显然这种情况是系统开销浪费最多的. 将上一节测得的结果代入式(1)、(2),得到阈值 T 的值应为 8.

DN 迁移操作:选择 DRAM 空间中的数据页迁移到其他存储介质一般是由替换算法来完成的. 最常用的是最近最少使用算法(Least Recently Used, LRU)[11],LRU 算法通过记录每个页最近一段时间被访问的频繁程序,将内存中的页分为冷数据和热数据,系统优先选取冷数据进行回收. LRU 算法只考虑了数据的冷热程度而没有考虑数据被访问的读写特性,因此它在混合内存架构中不再适用.

为了提出更合理的策略,我们将内存中的数据页分为:CR(Cold Read)页、CW(Cold Write)页、HR(Hot Read)页和 HW(Hot Write)页,分别表示最近很少被读的数据页、最近很少被写的数据页、最近经常被读的数据页和最近经常被写的数据页. 本文提出最近最少写(Least Recently Write, LRW)算法,该算法记录 DRAM 中每个数据页被访问的频繁程度以及读写特征,并在系统 DRAM 空间不足时选择合适的候选页迁移到 NVM 中. LRW 算法将 DRAM 中的数据页保存在三个不同的链表中:

(1)读链表. 该链表中包含 CR 页和 HR 页,这部分数据页近期都没有发生过写操作;

(2)非活动的写链表. 该链表包含 CW 页,即最近发生过少量的写操作;

(3)活动的写链表. 该链表中只包含 HW 页,即近期发生的写操作次数较多的数据页.

LRW 链表通过 PG_write 和 PG_active 这两个标志位来标记每个数据页的状态. PG_write 表示该数据页最近一次发生的操作(1 为写操作,0 为读操作). 而 PG_active = 0 表示该数据页近期只被读(或写)过一次,PG_active = 1 表示该数据页近期被读(或写)过至少两次. 图 2 描述了数据页在 LRW 的读链表、非活动的写链表、活动的写链表中的迁移情况. 在 HyMG 机制中,当 DRAM 空间不足时,LRW 算法会首先选择读链表中的数据页进行替换. 如果 DRAM 空间仍然不足,LRW 算法会对非活动的写链表中的数据页进行回收. 最后,如果回收这两部分链表中的数据页所释放的空间仍然无法满足系统的分配需求,LRW 会释放活动的写链表中

的数据页.

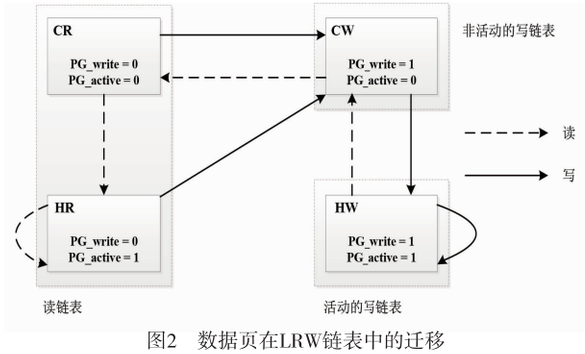


图2　数据页在LRW链表中的迁移

2.4　NVM 空闲页分配策略

ND 迁移策略在一定程度上保证了 NVM 页框在短时间内不会发生过多的写操作. 但是,在各种复杂程序运行下,仅通过 ND 迁移策略还无法保证写操作均衡地分布在它的各个页框上.

本节使用一种两路链表来管理 NVM 中的页框. 两路链表分为前半部和后半部. 在系统初始阶段,所有的空闲页框都位于前半部链表. NVM 中的每个页框都记录一个 W_C值,用于表示该页框被写过的次数. 此外,系统维护一个动态的自适应变量 W_M,它用于记录在一段时间内,NVM 所有页框中最大的 W_C值. 当回收一个 NVM 页框时,首先要判断该页框的 W_C值与系统 W_M值的大小关系. 如果 $W_C < W_M$,就将其回收到前半部的空闲链表中供应用程序调用. 如果 $W_C \geqslant W_M$,就将其回收到后半部的空闲链表中,表明写次数过多. W_M值是动态更新的,每当发生前半部链表页框不足时,算法就会扫描所有空闲页框的 W_C值用于更新 W_M值,以确保在之后的回收过程中会有页框被释放到前半部的链表中去. 同时也保证了 W_M值是这段时间内发生写操作次数最多页框的 W_C值.

通过使用两路链表并动态的更新 NVM 页框中最大的写次数 W_M,将 NVM 的页框进行了分类,发生写操作次数较多的页框保存在后半部链表中,发生写操作次数较少的页框保存在前半部链表中. 在分配时,优先使用前半部链表的页框,这使得写操作更加均衡地分布到 NVM 的每个页框中. 这种方式提升了 NVM 介质的使用寿命.

3　实验结果与分析

我们使用 Linux4.4 版本的内核进行实验,并与现有的比较经典的内存管理策略(Linux 内存管理[9]、Dr. Swap[10]、M-CLOCK[12]、CLOCK-DWF[13]、LRU-WPAM[14]和 MHR-LRU[3])做对比分析. 操作系统的内存大小为 8GB,混合内存区由 2GB 的 DRAM 和 1GB 的 NVM 组成. 使用 filebench 测试工具集中的一组测试程序,包括 multi-stream-write、fileserver、webproxy、randomrw、video-server.

3.1　迁移操作次数

本节通过具体实验来讨论阈值 T 的设定. 实验统计不同阈值下,系统发生的 ND 迁移操作和 NVM 写操作的次数. NVM 写次数包括应用程序直接对 NVM 写操作的次数以及由 DN 迁移操作引起的对 NVM 写操作的次数. 实验结果如图 3 所示.

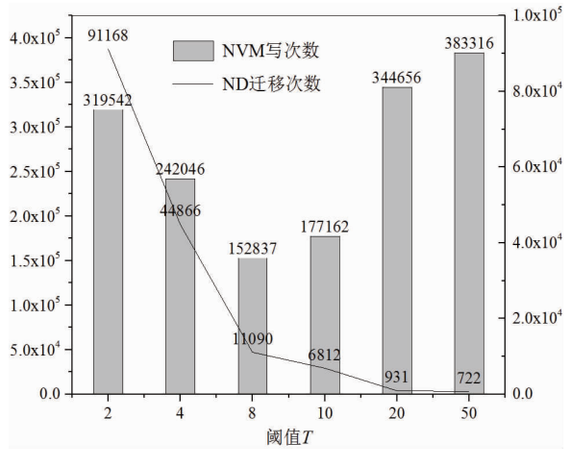


图3　不同阈值条件下的NVM写次数和ND迁移次数

在图 3 中,横轴代表了不同的阈值. 柱状图表示不同阈值下系统发生的 NVM 写操作次数(左竖轴). 而折线图表示不同阈值下系统发生的 ND 迁移操作次数(右竖轴). 当 T 较小,如 $T=2$ 时,系统中发生的 ND 迁移操作和 NVM 写次数都很多. 主要是因为大量的 NVM 数据页在写过一次后就会被迁移到 DRAM 中,引起大量的 ND 迁移操作,加剧了 DRAM 空间的消耗,DRAM 空间不足也造成了大量的数据页迁移到 NVM 中,使得 NVM 的写操作次数增加. 随着 T 的增加,ND 迁移次数急剧下降,当阈值 $T=50$ 时,系统仅发生了 722 次 ND 迁移操作,这是因为绝大部分对 NVM 数据页的写操作都直接在 NVM 中进行. 综上所述,在后续实验中,HyMG 策略中的阈值 T 都设置为 8,此时系统发生的 NVM 写操作次数最少,并且 ND 迁移次数也得到了大幅的降低.

几种测试程序在不同策略中产生的迁移操作次数如图 4 所示,这里的迁移操作次数包括 ND 迁移次数以及 DN 迁移次数. 在 Linux Swap 和 Dr. Swap 策略中,一旦应用程序尝试写一个 NVM 数据页,该数据页就会被迁移到 DRAM 页框中. M-CLOCK 仅在 DRAM 空间不足时,允许对 NVM 数据页进行一次写操作. 这三种策略都发生了大量的 ND 迁移操作. CLOCK-DWF 在处理所有新的写请求时,都会将页面移到 DRAM 中,这大大加大了 DRAM 中发生页面置换的频率,同时带来了数量巨

大的迁移次数.

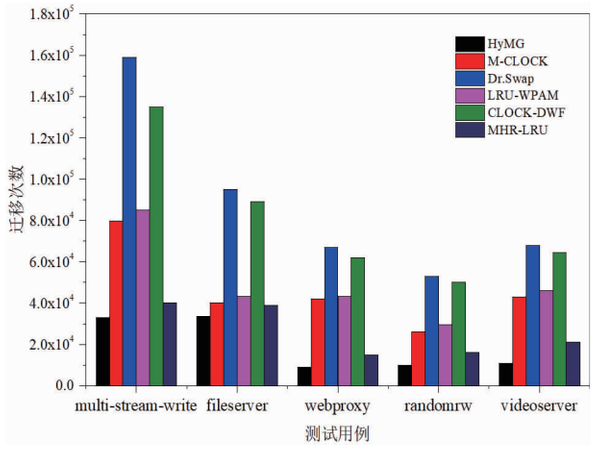


图4 不同策略下的迁移操作次数

与前面几种策略不同,HyMG 机制允许直接在 NVM 页框中访问数据页,从而减少了数据页从 NVM 页框往 DRAM 页框中迁移,同时,这还增加了 NVM 空间的使用率,从而间接减少了数据页从 DRAM 页框向 NVM 页框的迁移操作.

3.2 NVM 写操作次数

NVM 写操作次数包括应用程序直接对 NVM 数据页的写操作和由 DN 迁移操作而间接产生的对 NVM 的写操作. 实验结果如图 5 所示,Dr. Swap 策略在所有的测试程序中所产生的 NVM 写操作次数都是最少的. 因为 Dr. Swap 策略中没有直接对 NVM 数据页的写操作,它对 NVM 的写操作全是由 DN 迁移操作间接引起的. CLOCK-DWF 算法也是将直接对 NVM 数据页的写操作都迁移到 DRAM 中,但是由于 DN 迁移引起的对 NVM 写操作多于 Dr. Swap,所以对 NVM 写次数明显高于 Dr. Swap.

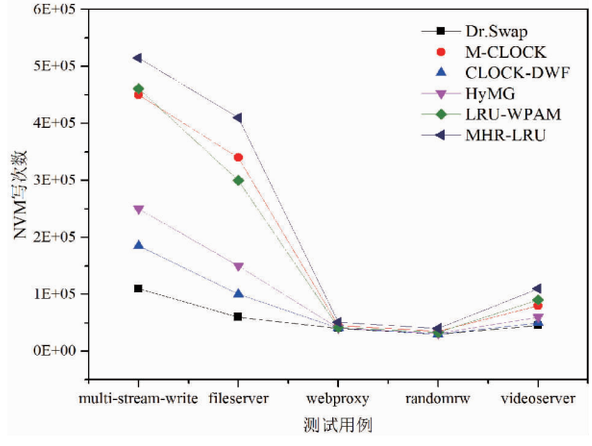


图5 NVM写操作次数

所有测试程序中,HyMG 策略发生的 NVM 写操作次数都要少于 M-CLOCK,主要因为 DRAM 空间在大多数时候都处于即将耗尽的状态,导致 M-CLOCK 策略中大部分写一次的操作都在 NVM 中进行,而且 M-CLOCK 策略中由 DN 迁移操作间接产生的 NVM 写操作次数也比 HyMG 策略要多.

3.3 系统性能

我们以程序完成所有内存数据访问的时间作为衡量系统性能的指标. 它包括读、写 DRAM 和 NVM 数据页的时间,以及进行 ND 迁移操作和 DN 迁移操作的时间. 这一时间用 t_{total} 表示如下:

$$t_{total} = (C_{DN} + C_{ND}) \times t_{mig} + C_{RD} \times t_{RD} + C_{WD} \times t_{WD} + C_{RN} \times t_{RN} + C_{WN} t_{WN} \quad (4)$$

其中,t_{mig} 表示迁移操作的时间. t_{RD}、t_{WD} 分别表示读、写 DRAM 内存数据页的时间. N_{RN}、N_{WN} 分别表示测试程序直接读、写 NVM 内存数据页的次数. t_{RN}、t_{WN} 分别表示读写 NVM 内存数据页的时间. 参数设定如下:读、写 DRAM 内存数据页的时间和读 NVM 内存数据页的时间相等,即 $t_{RD} = t_{WD} = t_{RN}$. 写 NVM 内存数据页的时间为写 DRAM 内存数据页时间的 7 倍,即 $t_{WN} = 7 * t_{WD}$. 迁移操作的时间为写 NVM 内存数据页时间的 8 倍,即 $t_{mig} = 8 * tC_{WN}$.

我们将 Linux 策略运行每组程序的时间作为基准参照,图 6 中各方案运行测试程序的时间是其与 Linux 策略运行测试程序时间的比值,比值越小,说明该方案相较于 Linux 策略运行的时间就越短,相应的系统性能就越好. 与其它策略相比,使用 HyMG 的系统最高提升了 128.5% 的性能(与 Linux 相比),最少提升了 46% 的性能(与 MHR-LRU 相比). 通过分析可以知道,影响系统的性能主要因素是系统所产生的迁移操作次数,其次是 NVM 的写操作次数. 使用 Linux 策略的系统在性能方面表现最差,因为该系统发生的迁移操作次数最多. 在 fileserver 这组实验中,M-CLOCK 和 HyMG 所产生的迁移操作次数以及 NVM 写操作次数都比较接近,因此使用这两种策略的系统在性能方面差异不大.

3.4 NVM 介质寿命

影响 NVM 介质寿命的主要因素是 NVM 中页框的最大写次数. NVM 介质的使用寿命 f 表示为:

$$f = \frac{W_{max} \times t_{exe}}{W_{exe}} \quad (5)$$

其中,W_{exe} 表示运行测试程序时,NVM 介质的最大写操作次数,t_{exe} 表示程序执行的时间,W_{max} 表示 NVM 所能承受的写操作次数上限. 该实验比较两组测试用例:未考虑磨损均衡的 HyMG 机制和考虑了磨损均衡的 HyMG 机制(即使用本文提出的两路链表). 他们分别运行测试程序,产生的 NVM 最大写操作次数的结果如表 2 所示.

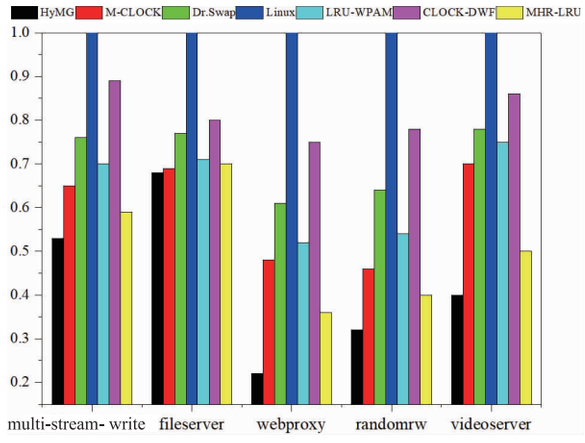


图6　归一化后的系统运行时间

表 2　NVM 的最大写次数

测试用例	未考虑磨损均衡的 HyMG	考虑磨损均衡的 HyMG
multi-stream-write	11089	1247
fileserver	6981	737
webproxy	811	104
randomrw	598	88
videoserver	769	112

使用f_{nowl}和f_{wl}分别表示未考虑磨损均衡及考虑磨损均衡时 NVM 的介质寿命. W_{nowl}和W_{wl}分别表示未考虑磨损均衡及考虑磨损均衡时 NVM 介质中的最大写操作次数. NVM 介质寿命的比值为:

$$\frac{f_{wl}}{f_{nowl}}=\frac{W_{nowl}}{W_{wl}} \tag{6}$$

最终,NVM 介质寿命比较结果如图 7 所示,相对于未考虑磨损均衡,考虑了磨损均衡的 NVM 介质平均寿命提高了 6.9 倍.

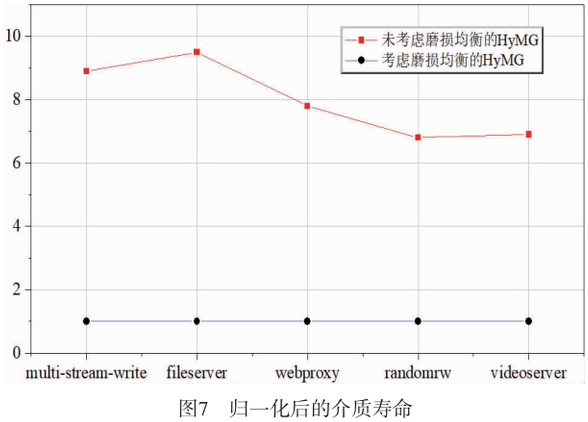


图7　归一化后的介质寿命

4　结论

本文主要研究了如何在混合内存中更好的进行数据分布. 通过两种数据页迁移策略,使得具有不同访问特征的数据页能动态地调整其在混合内存中的位置. 并通过磨损均衡策略来优化 NVM 介质的使用寿命. 最后对 HyMG 机制进行了仿真实验,并与现有的机制进行了对比,与其他策略相比性能最高提升了 128.5%,同时,HyMG 的磨损均衡策略使 NVM 介质寿命提升了 6.9 倍.

参考文献

[1] 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.
Wei M,Liu J N,Wei T,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)

[2] Li T,John L K. Run-time modeling and estimation of operating system power consumption[A]. 2003 ACMSIGMETRICS International Conference on Measurement and Modeling of Computer Systems[C]. San Diego,California,USA:ACM,2003. 160 – 171.

[3] Zhang J,Liao X,Jin H,et al. Anoptimal page-level power management strategy in PCM-DRAM hybrid memory[J]. International Journal of Parallel Programming,2017,45(1):4 – 16.

[4] Burr G W,Breitwisch M J,Franceschini M,et al. Phase change memory technology[J]. Journal of Vacuum Science & Technology B Microelectronics & Nanometer Structures,2010,28(2):223 – 262.

[5] Zhang H,Chen G,Ooi B C,et al. In-memory big data management and processing:a survey[J]. IEEE Transactions on Knowledge & Data Engineering,2015,27(7):1920 – 1948.

[6] Zhong K,Liu D,Liang L,et al. Energy-efficient in-memory paging for smartphones[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,2016,35(10):1577 – 1590.

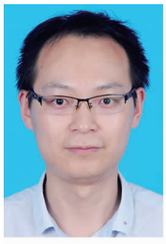
[7] Li M,Zhang H J,Wu Y J,et al. MemSC:a scan-resistant and compact cache replacement framework for memory-based key-value cache systems[J]. Journal of Computer Science and Technology,2017,32(1):55 – 67.

[8] Jiang L,Du Y,Zhang Y,et al. LLS:Cooperative integration of wear-leveling and salvaging for PCM mainmemory[A]. DSN'11 Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks[C]. Washington,DC,USA,IEEE,2011. 221 – 232.

[9] Guthaus M R,Ringenberg J S,Ernst D,et al. MiBench:A free,commercially representative embedded benchmark suite[A]. WWC'01 Proceedings of the Workload Characterization,2001. WWC-4. 2001 IEEE International Work-

shop[C]. Washington, DC, USA, IEEE, 2001. 3 – 14.
[10] Chen C H, Hsiu P C, Kuo T W, et al. Age-based PCM wear leveling with nearly zero search cost[A]. DAC'12 Proceedings of the 49th Annual Design Automation Conference[C]. San Francisco, California, IEEE, 2012. 453 – 458.
[11] 倪亚路,周晓方. 一种基于伪 LRU 的新型共享 Cache 划分机制[J]. 电子学报,2013,41(4):681 – 684.
NI Ya-lu, ZHOU Xiao-fang. A novel pseudo-LRU based shared cache partitioning mechanism[J]. Acta Electronica Sinica, 2013, 41(4): 681 – 684. (in Chinese)
[12] Lee M, Dong H K, Kim J, et al. M-CLOCK: migration-optimized page replacement algorithm for hybrid DRAM and PCM memory architecture[A]. SAC15 Proceedings of the 30th Annual ACM Symposium on Applied Computing [C]. Salamanca, Spain: ACM, 2015. 2001 – 2006.
[13] Lee S, Bahn H, Noh S H. CLOCK-DWF: a write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures[J]. IEEE Transactions on Computers, 2014, 63(9): 2187 – 2200.
[14] Tian W, Li J, Zhao Y, et al. Optimal task allocation on non-volatile memory based hybrid main memory[J]. IEEE Transactions on Very Large Scale Integration Systems, 2013, 21(7): 1271 – 1284.

作者简介



李 琪 男,1987 年生于江苏淮安. 重庆大学计算机学院博士. 主要研究方向为复杂网络、图计算、并行计算.
E-mail: liqi0713@ foxmail. com

钟 将 男,1974 年生于重庆市. 重庆大学计算机学院教授. 主要研究方向为大数据分析与挖掘、自然语言处理、云网融合、网络安全等.
E-mail: zhongjiang@ cqu. edu. cn