一种基于热数据识别技术的UBIFS优化方案

马 骏[1] 童 薇[1,2] 刘景宁[1,2] 刘景超[1]
(华中科技大学光电国家实验室 武汉 430074)[1] (华中科技大学计算机科学与技术学院 武汉 430074)[2]

摘 要 NAND Flash介质具有特殊的物理性质,传统文件系统不能直接对NAND Flash进行管理,容易造成NAND Flash设备性能下降、磨损不均衡等负面影响。闪存文件系统将FTL与文件系统功能有机地结合起来,可更好地发挥NAND Flash的高性能。无序区块镜像文件系统(Unsorted Block Image File System,UBIFS)是一种被广泛使用的闪存文件系统,但其存在着写放大和垃圾回收操作频繁触发等问题。针对UBIFS中存在的问题,提出利用多哈希函数的哈希表对热数据进行识别,以降低热数据识别开销,提高热数据识别的准确率;采用多日志技术,将不同热度的日志、数据分开存放,以减少垃圾回收触发频率;采用热数据日志延迟提交技术来减少UBIFS中日志提交带来的元数据修改,进而减少了写放大产生的次数。测试与分析表明,与原UBIFS相比,优化后的UBIFS的系统内部写物理块的次数减少5%~10%,垃圾回收操作触发的次数减少了7%~13%,同时系统的IOPS提高了5%~18%,系统性能下降现象得到了有效缓解。

关键词 闪存,UBIFS文件系统,热数据,垃圾回收
中图法分类号 TP316 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.06.007

Optimization Scheme of UBIFS Based on Hot Data Identification Technology

MA Jun[1] TONG Wei[1,2] LIU Jing-ning[1,2] LIU Jing-chao[1]
(Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China)[1]
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)[2]

Abstract NAND Flash memory can't be managed by traditional file systems for its special access characteristics. Although the problem is solved by using FTL in NAND Flash devices, the structure and algorithm in traditional file systems designed for disk can easily cause severe performance degradation and uneven wear in NAND Flash devices. Flash file systems that combine some FTL functions can make better use of NAND Flash memory with further optimization. UBIFS is a widely used Flash file system in Linux, but there are still severe write-amplification and frequent garbage collection in the storage system using UBIFS. In order to solve these problems of UBIFS, this research used hash table with multi-hash functions to identify hot data in UBIFS to reduce identification overhead and improve recognition accuracy. By using multiple log, cold and hot data are separated into different storage area to reduce the frequency of garbage collection. The delay commit technology of log was also adopted to reduce the number of times write-amplification occurs caused by frequent log commit. A series of experiments were conducted to verify the performance of the proposed method. Experiments show that internal writes to physical block in optimized UBIFS are reduced by 5%~10% compared to that in the original UBIFS, and the times of garbage collection triggered in optimized UBIFS were reduced by 7%~13% compared to that in the original UBIFS. In general, IOPS of storage system is improved by 5%~18% and system performance degradation is effectively alleviated by using optimized UBIFS.

Keywords NAND flash, UBIFS, Hot data, Garbage collection

1 引言

NAND Flash作为一种新兴的存储介质,具有与磁介质迥异的物理特性。具体表现为:读写速度不对称,且延迟远大于读操作延迟;且必须以页为单位,写操作必须在空闲页或者被擦除过后的页上进行;擦除操作以块为单位,一次擦除操作存在着很大的延迟,且每一个物理块只有有限的擦除次数。根据以上性质,NAND Flash介质需要额外的闪存转换层

到稿日期:2016-11-11 返修日期:2017-02-29 本文受国家863项目计划(2015AA016701, 2015AA015301),国家自然科学基金资助项目(61402189, 61472153),计算机体系结构国家重点实验室(CARCH201505),武汉市应用基础研究计划项目(2015010101010004),信息存储系统教育部重点实验室资助。

马 骏(1992-),男,硕士,主要研究方向为闪存文件系统、非易失内存系统,E-mail:abstractmj@163.com;童 薇(1977-),女,博士,讲师,主要研究方向为大规模网络存储系统、非易失内存和I/O虚拟化;刘景宁(1957-),女,博士,教授,主要研究方向为计算机系统结构、计算机存储系统、高速接口和通道技术,E-mail:jnliu@hust.edu.cn(通信作者);刘景超(1991-),男,硕士,主要研究方向为闪存文件系统。

(Flash Translation Layer,FTL)来提供地址映射、磨损均衡和垃圾回收的功能。

传统的磁盘文件系统是针对磁介质的特性设计的文件系统,其中的数据结构和算法都针对磁盘的物理特性进行了特别的优化,若直接用于 NAND Flash 构成的固态设备(设备中自带 FTL,屏蔽底层 NAND Flash 存储介质的物理特性,向上呈现块设备接口),会造成严重的写放大和频繁的垃圾回收,导致无法充分利用 NAND Flash 器件的高性能,并大大缩短其使用寿命。Flash 文件系统将 FTL 功能与传统文件系统的设备管理功能有机地结合起来,使得文件系统能够直接管理 NAND Flash 设备的物理地址空间,并且绕过通用块层,缩短了系统的 IO 路径,所以 Flash 文件系统更适合 NAND Flash 设备。无序区块镜像文件系统(Unsorted Block Image File System,UBIFS)是一种被广泛使用的 Flash 文件系统,具有支持大容量设备、IO 性能高和内存消耗低等优点。UBIFS 作为日志结构的文件系统,解决了传统日志文件系统中存在的两次写的问题,但是在日志提交的时候,元数据的修改仍带来了写放大效应,影响了 NAND Flash 的寿命;同时,冷热数据的混合存储导致了 UBIFS 垃圾回收被频繁触发,造成了系统性能下降;如何以较小的开销解决以上问题是我们研究的重点。

2 研究背景

目前可以使用 NAND Flash 存储介质的文件系统大体上可分为 3 类,如图 1 所示。

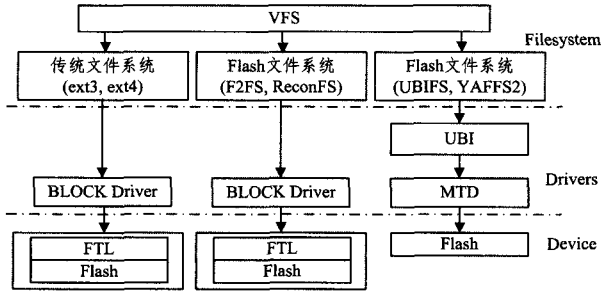

图 1 Linux 系统中不同类型的文件系统

第一类文件系统使用传统磁盘文件系统+设备端 FTL 的方式来管理 NAND Flash 设备中的文件,将 NAND Flash 设备当作磁盘来使用。传统磁盘文件系统的典型代表是 ext3/ext4 文件系统。传统文件系统的许多功能是针对磁盘介质的物理特性和机械特性来进行优化的,例如数据的预读技术、调度算法等。同时,传统文件系统中的日志机制会导致数据两次写,该机制并不适合擦除次数有限的 NAND Flash 设备。另外,为了防止元数据丢失,传统文件系统的元数据会进行频繁的同步,导致文件系统写放大效应十分严重[1,9,12]。因此使用传统文件系统管理+设备端 FTL+NAND Flash 存储介质的组合不能充分利用 NAND Flash 的性能。

第二类文件系统同样依靠设备端 FTL 来管理 NAND Flash,但是修改了传统磁盘文件系统中的某些结构和算法,使之更适合于 NAND Flash 的特性,其主要代表有 F2FS[3] 和 ReconFS[1]。相比传统磁盘文件系统,这类针对闪存特性而专门设计的文件系统在性能上有了一定的提升。例如 ReconFS 通过反向链接的形式来减少文件系统的写放大,从而延长 NAND Flash 的寿命;F2FS 使用节点地址表(Nodes Address Table,NAT)来解决 Flash 文件系统中 wandering tree 的问题,同时采用多头部日志存放热度不同的数据。但是 ReconFS 和 F2FS 不能直接访问 NAND Flash 的物理地址空间,优化力度有限,因此不能充分利用 NAND Flash 的物理特性。此外,F2FS 在解决 wandering tree 问题时加入了 NAT,使得系统中又增加了一层映射,效率并不高。虽然采用了多头部日志技术来分离冷热数据,但是固定的数据分类模式会使其冷热数据识别的准确率较低(未考虑数据温度会发生变化的事实)。

第三类文件系统可以直接管理 NAND Flash 设备,不需要设备端 FTL 的支持。此类文件系统自身包含有 FTL 功能,能够直接操作 NAND Flash 物理地址的内容。例如 UBIFS,JFFS2 和 YAFFS2[4-5],这些文件系统针对 NAND Flash 的物理特性进行优化。JFFS2 文件系统将元数据存储在内存中,从而导致文件系统不够健壮,在异常掉电的情况下无法保证数据的完整性,同时会导致文件系统内存的利用率和 NAND Flash 设备的容量成正比,使得系统的扩展性有限,因此 JFFS2 只能用于小型的 NAND Flash 设备;YAFFS2 文件系统的优势是内存占用率低,但是它的 FTL 策略过于简陋,且性能较差。UBIFS 文件系统将索引存储在 NAND Flash 设备中,因此该文件系统理论上具有良好的扩展性,各项表现在针对 NAND Flash 裸设备的文件系统中十分突出,但是它也存在着垃圾回收负载过重、冷热数据混放、写放大较为严重等问题。下面将重点分析、研究和解决上述问题。

3 问题分析

UBIFS 是一款可靠性强、性能优良的嵌入式闪存文件系统,从 Linux 2.6.32 版本开始就已经被并入 Linux 内核。但是在 UBIFS 文件系统中,由于 NAND Flash 必须异地更新,元数据更新频繁,导致其索引节点会连续溯源更新,造成写放大雪崩效应问题。除此之外,UBIFS 将热数据与冷数据统一处理,导致其垃圾回收被频繁触发,占用系统资源,影响系统性能。而现有的一些数据热度识别方法的复杂度高,开销巨大,不适用于 UBIFS 文件系统。下面将对 UBIFS 中存在的问题进行详细阐述。

3.1 写放大问题

Flash 文件系统将元数据存储在 NAND Flash 上,当数据块进行异地更新时,该数据块由于异地更新而发生地址变化,导致其直接索引节点中指向该数据块的指针需要改变,同时也导致直接索引节点需要更新。同理,该数据块的间接索引节点中的指针因为直接索引节点的异地更新而失效,导致间接索引节点需要异地更新。按照这种更新方式,上层索引结构依次递归地异地更新,这就是 wandering tree 问题[25]。

wandering tree 问题的具体发生过程如图 2 所示,图中阴影部分的节点表示更新后的数据节点。假设文件 F1 需要更新,系统将文件 F1 的数据块异地更新到阴影 F1 中,此时 F1 的直接索引指针 P1 失效,因此需要将包含新指针项的索引异地更新到带阴影的 D1 中。同理,D1 的父节点 D2 也需要进行异地更新,最终这种更新将一直递归传递到 ROOT 节点。文件系统更新一个数据块 F1,导致 F1 的父节点、祖先节

点、ROOT 节点都进行异地更新。这种由于更新一个文件的一小块数据导致多个索引块递归更新的情况被称为 wandering tree 问题。毫无疑问，wandering tree 问题会带来 Flash 文件系统严重的“写放大”。

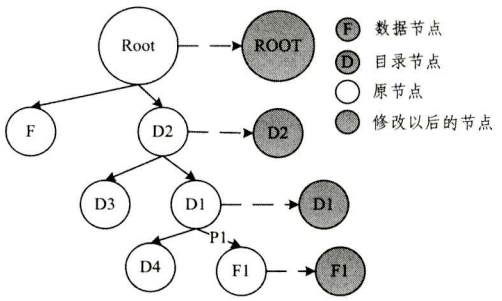

图 2　数据更新带来的 wandering tree 问题

对于 wandering tree 问题，有两种优化方法：第一种方法类似 F2FS 中采用的 NAT 表技术，但是这种技术又会带来额外的映射开销；另外一种方法是合并写操作，能减少元数据修改的次数。本文采用第二种方法，通过热数据延迟提交技术来减少整个设备内部写操作的次数。详细设计见 4.2 节。

3.2　垃圾回收开销问题

UBIFS 垃圾回收存在以下两个缺陷。

(1)在 UBIFS 垃圾回收过程中，需要首先将目标块的有效页数据读出并迁移到新的数据块中，再擦除回收该块。在读取有效页时，UBIFS 需要扫描整个逻辑块才能确认有效页的位置，费时费力。可以采用在 UBIFS 原本的逻辑块属性树中添加标志位的方法来提升 UBIFS 的垃圾回收效率，具体设计见 4.3 节。

(2)UBIFS 冷热数据混合导致垃圾回收操作中数据的大量移动和垃圾回收操作被频繁触发。如图 3 所示，其中带阴影的方块为一个无效页，图中假设 7 个页组成 1 个物理块。开始状态下，如图所示 1 号块，为冷热数据混合的数据块。此时系统中空间紧张，执行垃圾回收操作，将 1 号块中的有效数据复制到 2 号块中，同时 1 号块被擦除为空闲块。2 号块中矩形框表示后来有两个热数据页被写入。随着时间的推移，2 号块的热数据页失效，由于块中 4 个热数据页失效，只剩下 3 个有效的数据页，系统执行垃圾回收，将 2 号块中的 3 个有效页移动到 3 号块中。综上，由于冷热数据的混合，导致图 3 中 3 个冷数据的数据页一直被移动，同时冷热数据的混合导致每次垃圾回收得到的空闲页变少，进而导致垃圾回收被频繁触发。对此，可以将 UBIFS 中的冷热数据分开存放，以提高垃圾回收效率，进而提升文件系统性能。

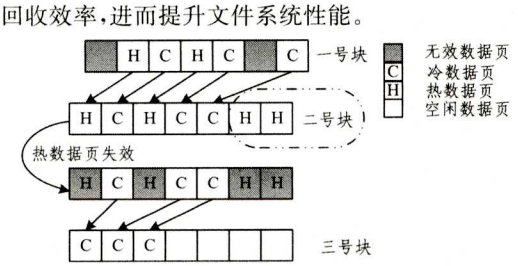

图 3　忽略数据热度对垃圾回收的影响

3.3　数据热度识别问题

常用的冷热数据识别技术有两种：基于 LRU(Least Recently Used)算法的冷热数据识别技术和使用哈希表的冷热数据识别技术。LRU 算法的识别开销大；而哈希表的实现简单且效率较高[27-28,30-31]。

LRU 算法根据数据的访问历史筛选冷热数据，其核心思想是程序的时间局部性，即如果一个数据块最近被访问过，那么该块将来被访问的几率会更高。LRU 算法使用链表存储热数据块的块号，主要操作为：当写入一个新的数据块时，该数据块的块号被插入到链表的头部；当写数据块命中时，该数据块的链表项被插入到链表的头部；当链表满时，将链表尾部的数据块号丢掉。判断一个数据块是否为热数据块的方法是遍历该链表，如果链表中有该数据块的块号则为热数据，否则为冷数据。LRU 算法的缺点在于维持 LRU 算法的空间开销巨大，识别热数据需要遍历整个链表，效率太差。

基于哈希表的冷热数据识别技术使用一个哈希表来存储数据块的热度值，当写一个数据块时，首先利用哈希函数计算出数据块号对应的哈希表项，然后将对应哈希表项中存储的热度值加 1。该算法简单且空间开销低，识别热数据的效率高。但是这种方法存在哈希冲突的问题，容易将冷数据块错误地识别为热数据块。使用多个哈希函数的哈希表的方式可以有效解决这个问题。详细分析设计见 4.1 节。

4　设计与实现

UBIFS 优化方案的总体框图如图 4 所示，中间矩形区域是 UBIFS 系统原生的功能模块，读写命令经过 UBIFS 和 UBI 层的转换变为 MTD 格式的读写命令；然后，命令经过 MTD 驱动层传输到设备端；最终，设备端将数据写入 NAND Flash 介质中。图 4 中 3 个虚线框矩形为主要优化的模块。

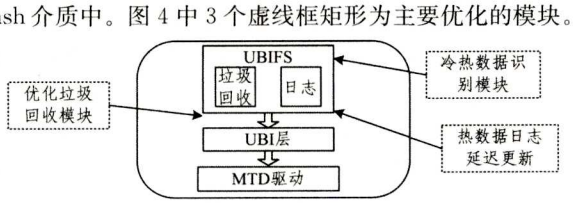

图 4　UBIFS 系统优化方案的总体设计

冷热数据识别模块：识别写入数据的冷热状态，并将数据分别存放到各自的冷、热日志中，以减少 UBIFS 写放大的次数，延长存储设备寿命。垃圾回收优化模块：在擦除块属性树(Lo-gic Erase Block Properties Tree，LPT)的头部添加表示块中页状态的元数据信息，依据元数据标识，可以减少垃圾回收过程中繁缛的扫描过程及无效数据页的读取过程。延迟提交的热数据日志模块将系统的日志重新划分为多个日志(冷元数据、热元数据、冷用户数据和热用户数据)，根据热数据容易失效的原理，延迟热数据日志的更新，减少由于日志提交而产生的写放大发生的次数。

4.1　数据热度识别模块

系统多日志优化方案和垃圾回收方案都需要使用到冷热数据识别技术，在原 UBIFS 系统中添加冷热数据识别模块，利用布隆过滤器[27]的相关原理，采用多哈希函数的哈希表来统计数据冷热程度。

在 UBIFS 冷热数据识别系统的设计中，采用类似于布隆过滤器的结构来判断一个数据块是否属于热数据块的集合。若使用一个哈希函数，则容易产生冲突，导致将不属于热数据块集合的块错误地判断为热数据块，因此需要使用多个哈希

函数来保证在节省空间的情况下降低误判发生的概率，只有所有哈希函数都表明此数据块在热数据块集合中，才判断此块为热数据。如图 5 所示，以 4 个哈希函数的哈希表为例。当一个写操作从上层分发下来时，系统使用图 5 中的 4 个哈希函数 $F1(x)$，$F2(x)$，$F3(x)$，$F4(x)$将逻辑块号(图中为 25)计算成不同的哈希值(在图 5 中分别为第 1,5,3,7 块)。哈希表使用 4 位表示数据块热度，当数据块被写入时，哈希函数计算出对应的哈希表项中的热度值(每次加 1 后作为新的热度值)。例如图 5 中哈希表对数据冷热的判定，若 $F1(x)$，$F2(x)$，$F3(x)$，$F4(x)$ 4 个哈希函数计算出来的表项中的热度值的高两位全都非 0，则数据为热数据，否则数据为冷数据。图 5 表示块号 25 的写操作，$F1(25)=1$，其中第一个表项为 0101，高两位为 01；然后判断 $F2(25)=5$，其中第五个表项值为 1110，高两位为 11；判断 $F3(25)=7$，第七个表项为 0001，高两位为 00，最终 $F4(25)=3$；第三个表项为 1011，高两位为 10。然后将各项的高两位做“与”运算：01&&11&&00&&11=00。由于结果为 00，因此数据为冷数据。综上，最终判定 25 号块中的数据为冷数据。

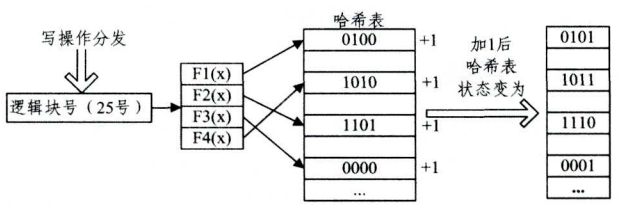

图 5 写操作分发时哈希表的状态变化

文献[13]详细阐述了哈希函数的选择，以及哈希函数个数和哈希表大小的设置。本文利用其思想来进行多函数哈希表的配置，致力于将此热度识别方法应用于 UBIFS 文件系统中，并在此基础上进行多个方面的优化。

在 UBIFS 文件系统初始化过程中，将在内存中创建并初始化数据热度的哈希表。在写操作到来时，根据上述算法判断数据热度。由于当前被频繁访问的数据在一段时间后的访问频率有可能大幅降低，因此系统每隔一定数量的写操作之后会对哈希表中的热度值进行衰减操作(热度值右移 1 位)。根据文献[13]的研究结果，在每 5000 次写操作之后进行一次热度值衰减。

4.2 优化日志模块

为了减少 UBIFS 写放大发生的次数，提出优化日志技术，该技术包括多日志技术和热数据日志延迟提交技术。

冷热日志结构如图 6 所示，系统将日志区域重新划分为 4 个区域：存储热元数据的日志、存储冷元数据的日志、存储热数据的日志、存储冷数据的日志。当上层写请求到来时，系统首先检测该数据是否为热数据，然后判断数据是否为元数据。

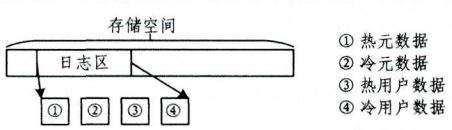

图 6 冷热日志的组织结构

热数据日志延迟写技术利用热数据容易失效的特性，通过延迟提交数据日志，将热数据在日志中存储足够长的时间。热数据在日志中容易被多次修改，系统提交热数据日志时，只需要提交最新的数据，从而减少对元数据的修改，进而减少写放大发生的次数。

图 7 示出原 UBIFS 系统日志提交过程，冷热数据混和放置，只有一个日志。其中，灰色小正方形表示冷数据块，白色小正方形表示热数据块，黑色的小正方形块表示无效的数据块，虚线圆角框表示日志区，实线圆角矩形区域表示 UBIFS 的主区(main area)(圆圈内数字表示主区状态)，第一条竖虚线为检查点 1(checkpoint1)，第二条竖虚线为检查点 2(checkpoint2)。图 7 中第一个检查点到来时，系统进行日志的提交，此时冷数据日志和热数据日志被提交到主区中(处于状态①)，所以看到此时主区中的 8 个数据块均为有效的数据块，其中为热数据的 1,2,3,4 号块中的数据有很高的概率被再次更新。随着时间的推移，系统更新主区中的热数据，热数据 1,2,4 号块变为无效，被异地更新为日志区中的 7,8,9 号块，主区状态变为②号状态。检查点 2 到来时，由于日志不为空，系统将日志区中包含热数据的 7,8,9 号块提交到主区中。最终主区变成③号状态。根据以上描述可以得知最终的结果，在检查点 1 到来时，日志一共提交了 8 个数据块，在检查点 2 到来时，日志一共提交 3 个数据块，最终需要提交的数据块数目为 11，系统需要承担 11 个数据块修改而带来的元数据的改动。

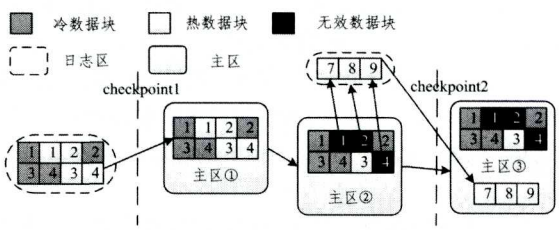

图 7 原 UBIFS 日志提交过程

优化后的日志提交方案如图 8 所示，将系统内部分为两个日志段，一个日志为冷数据日志(白色虚线框)，另一个日志为热数据日志(灰色虚线框)。假定每个日志中有 4 个数据块。当检查点 1 到来时，系统提交冷数据日志中的内容，将图 8 中冷数据的 1,2,3,4 号块提交到主区中，在检查点 1 到来时，系统延迟热数据日志的提交，只提交冷数据日志中的数据块。随着时间的推移，热数据日志中的 1,2,4 号块变为无效数据，并异地更新到热数据日志的 7,8,9 号块中。在系统的检查点 2 到来时，再进行热数据日志的提交。检查点 1 到来时，系统提交冷数据日志，并修改 1,2,3,4 号块对应的元数据；检查点 2 到来时，系统再提交热数据日志(此时图 8 中冷数据日志区中无数据，如果有数据，也需要进行提交)，并修改其 3,7,8,9 号块所对应的元数据。系统本次总共更新了 8 个数据块的元数据信息。与图 7 相比，优化后的方案减少了 3/11 的元数据更新。

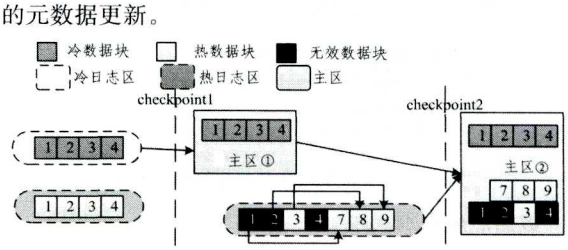

图 8 优化的 UBIFS 日志提交过程

4.3　优化垃圾回收功能

UBIFS 采用逻辑块属性树(Logic Erase Block Properties Tree,LPT)来管理逻辑块的状态,如逻辑块空闲空间、有效数据页数目等信息。LPT 采用 B+树的形式来实现。如图 9 所示,假设图中 LPT 共有 3 层,第一层全部为 LPT 的叶子节点,用来存储数据块的使用情况;第二层和第三层为 LPT 的内部节点,用来记录 LPT 的索引结构。

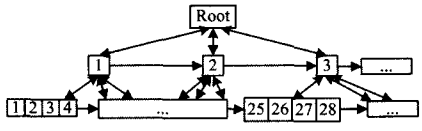

图 9　UBIFS 逻辑块属性树结构

针对 UBIFS 垃圾回收功能做了以下优化:1)在 LPT 中加入位图数据结构,假设一个块由 64 个页组成,在 LPT 中加入一个 64 位的位图,其中每一位表示一个页的有效状态。垃圾回收首先通过 LPT 选取目标块;然后读取目标块位图,根据位图判断每个页的有效状态;最后,垃圾回收只需要移动目标块中的有效页,减少了扫描块及读取无效页的操作。2)使用多日志的方式来存放冷热数据,结合系统的冷热数据识别模块,将冷数据存放在冷数据日志中,将热数据存放在热数据日志中。通过上述方式,UBIFS 文件系统能够将热数据页存放在同一个物理块中。随着时间的推移,热数据块中的数据更容易失效,垃圾回收执行一次获得的空闲块更多,从而减少了垃圾回收的次数。

5　实验结果与分析

对加入冷热数据感知等优化技术后的 UBIFS 优化方案进行评估。采用 Linux 内核自带的 Nandsim 模拟器来模拟 NAND Flash 设备,将相同配置的模拟设备分别格式化成原 UBIFS 格式和优化后的 UBIFS 格式,在多种负载条件下分别进行测试与分析。

测试基于 UBIFS 的优化方案时,需要修改 Linux 3.6.0 内核版本的 UBIFS 文件系统代码。添加冷热数据识别模块的代码,修改原有日志区的数据结构和提交策略优化了 UBIFS的垃圾回收算法。

5.1　测试环境

采用 Linux 自带的 Nandsim 模拟器来模拟 NAND Flash 设备,模拟的 NAND Flash 设备大小为 8GB,块大小为 128kB,页大小为 2kB,擦除延迟为 2ms,写延迟为 200ns,读延迟为 25ns。将优化后的 UBIFS 与优化前的 UBIFS 分别运行于相同配置的模拟设备上,进行性能测试与对比。

测试使用的机器配置是 Intel Xeon E5-2620 CPU,6 核心,频率为 2GHz,内存为 16GB DDR3,主板型号为 SuperMicro X9DRL-iF,操作系统为 CentOS 6.5。分别采用 Linux 自带的 dd 工具以及常用的文件系统性能测试工具 filebench。

表 1 列出了测试中所使用的负载和特性,包括负载描述、负载使用文件的数量、文件最大容量、负载中的工作线程数量以及负载的读写比例。使用两种测试工具进行测试,使用 dd 做单一条件的测试;而使用 filebench 工具中的 fileserver 和 varmail 这两种负载模式进行系统写性能和垃圾回收性能的测试。

表 1　测试负载设置

负载	负载描述	文件数量/大小	线程数量	读写比例
dd 命令	顺序写	1/2GB	1	0/1
fileserver	大量大文件随机读写	10000/128kB	50	1/2
varmail	大量小文件随机读写	1000/16kB	16	1/1

5.2　日志优化效果

选取 dd 命令和 filebench 的 fileserver 模式来测试系统的写入次数。由于 dd 是顺序写的负载,每个数据块仅被写入一次,在 dd 负载下系统不会有热数据,热数据日志延迟提交模块不会减少 UBIFS 内部写操作次数,因此本文的优化方法对于 dd 负载的优化效果并不明显。fileserver 负载比 dd 负载更贴近现实使用情况,系统中有热数据,热数据日志延迟提交功能正常运行,因此优化的 UBIFS 的写入次数远少于原 UBIFS 的写入次数。dd 负载测试结果如图 10 所示,横坐标为系统写入数据量,纵坐标为系统写操作的触发次数。优化的 UBIFS 写操作触发次数略多于原 UBIFS 系统的写操作次数,但在可接受范围内。产生这种结果的原因在于 dd 命令产生的是顺序写入的负载,数据具有同样的热度,导致优化的 UBIFS 中的冷热数据识别模块不但没能带来性能上的提升,而且还引发了额外的开销。但幸运的是,实际应用环境中冷热数据有明显区分,优化的 UBIFS 在通常情况下都能带来性能的提升。

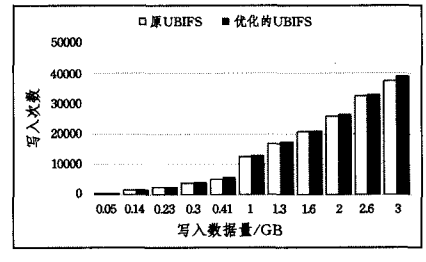

图 10　dd 负载下系统的写入次数

使用 fileserver 负载测试原 UBIFS 和优化的 UBIFS 的写入次数,测试结果如图 11 所示,横坐标为 fileserver 负载运行的时间,纵坐标为写入次数。在 fileserver 负载下,优化的 UBIFS 的写入次数远少于原 UBIFS 的写入次数。相较于原 UBIFS,优化的 UBIFS 的写入次数减少了 5%～10%。实验结果证明,在现实负载下,热数据日志延迟提交模块能够有效减少系统内部的写操作次数,从而缓解写放大情况,延长 NAND Flash 的使用寿命。

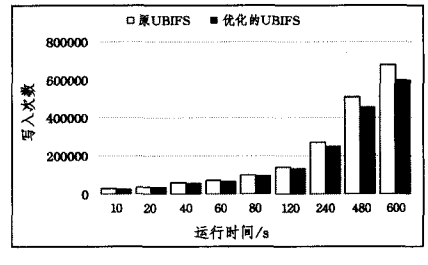

图 11　fileserver 负载下系统的写入次数

5.3 垃圾回收优化效果

垃圾回收优化效率主要考虑了两个衡量指标：垃圾回收触发次数和垃圾回收触发时导致的系统 IOPS 下降情况。

测试负载为 varmail 和 fileserver，一共测试 4 组数据：varmail 负载下垃圾回收触发次数、fileserver 负载下垃圾回收触发次数、varmail 负载下垃圾回收导致的系统 IOPS 下降情况、fileserver 负载下垃圾回收导致的系统 IOPS 下降情况。为了最大限度地减少实验误差，每组数据测试 5 次，去掉最高和最低数据，将中间 3 组数据的平均值作为最终实验结果。

首先测试系统垃圾回收的触发次数，将 MTD 存储空间作为变量，测试 MTD 设备存储空间利用率在 0%～90%时垃圾回收的触发次数。MTD 设备没有数据时，设备空间充足，不会触发垃圾回收。随着设备空间被占用比率的增加，当设备空间不足时，系统触发垃圾回收，测试结果如图 12 和图 13 所示。图 12 示出了 varmail 负载下的测试结果，开始时设备空间充足，垃圾回收不触发，当设备空间被占用比率增加到 70%时系统垃圾回收被触发，优化的 UBIFS 系统的垃圾回收触发次数明显少于原 UBIFS 系统。图 13 示出了 fileserver 负载下的测试结果，开始时设备空间充足，垃圾回收不被触发，随着设备空间被占用比率的增加，当设备空间被占用的比率达到 40%时系统垃圾回收被触发，随着设备空间被占用比率增加，系统垃圾回收触发次数急剧加大，优化的 UBIFS 依然优于原 UBIFS。通过测试结果可知，fileserver 负载下，优化的 UBIFS 系统的垃圾回收触发次数少于原 UBIFS 系统。相较于原 UBIFS，优化的 UBIFS 的垃圾回收触发次数减少了 7%～13%。

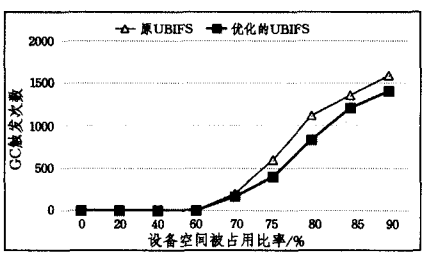

图 12 varmail 负载下触发 GC 次数

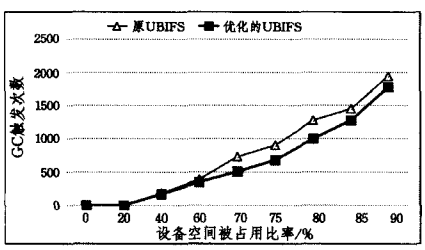

图 13 fileserver 负载下触发 GC 次数

评价系统垃圾回收效率的另一个重要的参数为垃圾回收触发时系统的 IOPS。分别在 varmail 和 fileserver 负载下测试 UBIFS 优化前后垃圾回收导致的系统 IOPS 下降情况。图 14 为 varmail 负载下的测试结果，系统未触发垃圾回收时，优化前后系统的性能在 6100 IOPS 上下波动，当设备空间被占用的比率达到 60%以上时，系统 IOPS 有明显的下降。原 UBIFS 垃圾回收导致的 IOPS 下降明显快于优化的 UBIFS 系统的 IOPS 下降。当系统占用率达到 80%后，原 UBIFS 和优化的 UBIFS 的 IOPS 下降剧烈，但是优化的 UBIFS 的 IOPS 依然高于原 UBIFS 系统。通过对测试结果进行分析可知，在 varmail 负载下，垃圾回收启动后，优化的 UBIFS 系统比原 UBIFS 系统的 IOPS 高 5%～18%。

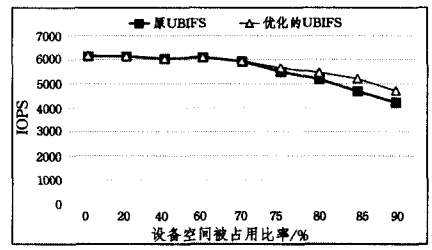

图 14 varmail 负载下的 IOPS

图 15 所示为 fileserver 下系统优化前后的 IOPS 对比。测试结果表明，在 fileserver 负载下，触发垃圾回收后，优化的 UBIFS 系统的 IOPS 明显高于原 UBIFS 系统的 IOPS。设备空间被占用比率到达 75%以上时，原 UBIFS 系统的 IOPS 下降加剧，优化的 UBIFS 系统的 IOPS 比原 UBIFS 系统的 IOPS 更高。fileserver 负载下，优化的 UBIFS 系统的 IOPS 比原 UBIFS 系统高 5%～18%。

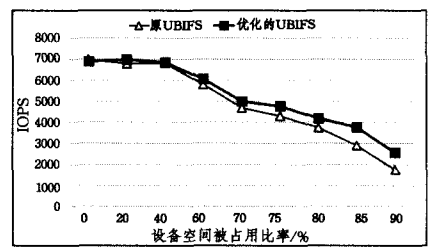

图 15 fileserver 负载下的 IOPS

结束语 NAND Flash 具有特别的物理性质，需要依靠设备端的 FTL 对 NAND Flash 设备进行管理的文件系统不能直接操作 NAND Flash 设备的物理地址空间，容易造成不必要的写放大效应，进而频繁触发垃圾回收操作。针对闪存设计的 UBIFS 虽然能够直接管理 NAND Flash 设备的物理空间，但是其不分数据冷热的统一处理导致垃圾回收效率不高；日志技术使得写放大效应频繁发生。针对 UBIFS 存在的问题，本文采用基于哈希表的热识别技术，以较小的开销准确识别热数据；采用多日志技术，将冷数据和热数据分开存放。采用热数据延迟提交技术，减少了 UBIFS 中日志提交带来的元数据修改，进而减少了写放大产生的次数，降低了垃圾回收操作对 UBIFS 读写性能的影响。采用优化的 UBIFS 方案能够提高 UBIFS 的效率。未来的工作将完善 UBIFS 内部的磨损均衡策略，进一步延长 NAND Flash 的寿命；同时提高 UBIFS 对 NAND Flash 设备访问的并行性，从而提升读写性能；也将针对特定应用的负载特性，进一步优化冷热数据识别技术。

参 考 文 献

[1] LU Y Y L, SHU J W, WANG W. ReconFS: A Reconstructable

File System on NAND Flash Storage[C]// Proceedings of the Usenix Conference on File & Storage Technologies. 2014:75-88.

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

[3] LEE C, SIM D, HWANG J Y, et al. F2FS: A New File System for NAND Flash Storage[C]// Proceedings of the Usenix Conference on File & Storage Technologies(FAST). 2015.

[4] QUICK D, ALZAABI M. Forensic Analysis of the Android File System Yaffs2[C]// 9th Australian Digital Forensics Conference. 2011.

[5] BITYUTSKIY A B. JFFS3 design issues [OL]. http://www.linux-mtd.infradead.org.

[6] OH Y, KIM E, CHOI J, et al. Optimizations of LFS with slack space recycling and lazy indirect block update[C]// Proceedings of the Annual Haifa Experimental Systems Conference. 2010.

[7] KANG D. Enhanced UBI layer for fast boot-up times of mobile consumer devices[J]. IEEE Transactions on Consumer Electronics, 2012, 58(2):450-454.

[8] LU Y, SHU J, ZHENG W. Extending the lifetime of NAND Flash-based storage through reducing write amplification from file systems[C]// Proceedings of the USENIX Conference on File and Storage Technologies (FAST). 2013:257-270.

[9] NARAYANAND, HODSONO. Whole-systempersistence[C]// Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 2012:401-410.

[10] JOO Y, NIU D, DONG X, et al. Energy-and endurance-aware design of phase change memory Caches[C]// Proceedings of the Conference on Design, Automation and Test in Europe. Leuven, Belgium, 2010:136-141.

[11] LUOJIE X, KURKOSKI B M. An improved analytic expression for write amplification in NAND Flash[C]// 2012 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2012:497-501.

[12] CAULFIELD A M, DE A, COBURN J, et al. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories[C]// Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. 2010:385-395.

[13] HSIEH J W, KUO T W, CHANG L P. Efficient identification of hot data for NAND Flash memory storage systems[J]. ACM Transactions on Storage (TOS), 2006, 2(1):22-40.

[14] DHIMAN G, AYOUB R, ROSING T. PDRAM: a hybrid PRAM and DRAM main memory system[C]// Design Automation Conference. 2009:664-669.

[15] SHI Z, CHEN X L, JI J S, et al. Space Efficient Mapping Management for Large-scale NAND Flash File System[J]. Journal of Chinese Computer Systems, 2010, 31(1):155-159. (in Chinese)
时正,陈香兰,纪金松,等. 大容量 NAND Flash 文件系统中的地址映射算法研究[J]. 小型微型计算机系统, 2010, 31(1):155-159.

[16] GUPTA A, KIM Y, URGAONKAR B. DFTL: a NAND Flash translation layer employing demand-based selective caching of page-level address mappings[J]. ACM Sigarch Computer Architecture News, 2009, 37(1):229-240.

[17] PARK D, DEBNATH B, DU D. CFTL: a convertible NAND Flash translation layer adaptive to data access patterns[J]. ACM SIGMETRICS Performance Evaluation Review, 2010, 38(1):365-366.

[18] LU Y Y, SHU J W, ZHENG W M. Extending the lifetime of NAND Flash-based storage through reducing write amplification from file systems[C]// Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13). 2013.

[19] LEE H, YUN H, LEE D. HFTL: hybrid NAND Flash translation layer based on hot data identification for NAND Flash memory[J]. IEEE Transactions on Consumer Electronics, 2009, 55(4):2005-2011.

[20] CHEN F, LUO T, ZHANG X. CAFTL: A content-aware NAND Flash translation layer enhancing the lifespan of NAND Flash memory based solid state drives[C]// Proceedings of the 9th USENIX Conference on File and Stroage Technologies. USENIX Association, 2011:6.

[21] LEE S, PARK D, CHUNG T, et al. A Log Buffer based NAND Flash Translation Layer Using Fully Associative Sector Translation[J]. IEEE Transactions on Embedded Computing Systems, 2007, 6(3):1539-9087.

[22] CHUNG T, PARK D, PARK S, et al. System Software for NAND Flash Memory: A Survey[C]// Proceedings of the International Conference on Embedded and Ubiquitous Computing. 2006:394-404.

[23] KAWAGUCHI A, NISHIOKA S, MOTODA H. A NAND Flash-Memory Based File System[C]// 1995 USENIX Technical Conference. 1995:155-164.

[24] CHIANG M L, CHANG R C. Cleaning Policies in Mobile Computers Using NAND Flash Memory[J]. Journal of System and Software, 1999, 48(3):213-231.

[25] KIM H J, LEE S G. A New NAND Flash Memory Management for NAND Flash Storage System[C]// Proceedings of the 23rd International Computer Software and Applications Conference. 1999:284.

[26] CHIANG M, CHENG C, WU C. A new FTL-based NAND Flash memory management scheme with fast cleaning mechanism[C]// International Conference on Embedded Software and Systems, 2008(ICESS'08). IEEE, 2008:205-214.

[27] PARK D, Du D H C. Hot data identification for flash - based storage systems using multiple bloom filters[C]// MASS Storage Systems and Technologies. IEEE, 2011:1-11.

[28] PARK D, Debnath B, Nam Y, et al. HotDataTrap: a sampling—based hot data identification scheme for flash memory[C]// Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012:759-767.

[29] MTD. MemoryTechnology Device[OL]. http://www.Linux-mtd.infradead.org.

[30] WANG C, WONG W. Extending the lifetime of NAND Flash memory by salvaging bad blocks[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2012.

[31] JUNG H, YOON K, SHIM H, et al. LIRS-WSR: Integration of LIRS and Writes Sequence Reordering for Flash Memory[J]. IEEE Transactions on Consumer Electronics, 2008, 54(3): 1215-1223.

[32] JO H, KAMG J U, PARK S Y, et al. FAB: flash-aware buffer management policy for portable media players[J]. IEEE Transactions on Consumer Electronics, 2006, 52(2): 485-493.

[33] KIM H, AHN S. BPLRU: a buffer management scheme for improving random writes in NAND Flash storage[C]// USENIX Association. 2008: 1-14.

[34] OU Y, HÄRDER T, JIN P. CFDC: a NAND Flash-aware replacement policy for database buffer management[C]// ACM. 2009: 15-20.

[35] PARK S, et al. CFLRU: a replacement algorithm for NAND Flash memory[C]// ACM. 2006: 234-241.

[36] KIM J, SHIM H, PARK S Y, et al. NAND FlashLight: a lightweight NAND Flash file system for embedded systems[J]. ACM Transactions on Embedded Computing Systems (TECS), 2012, 11(1): 18.

[37] SIMMONDS C. Linux NAND Flash file systems JFFS2 vs UBIFS[C]// Embedded Systems Conference UK. 2009.

[38] HAN C X, CHEN X L, XI L I, et al. Impact of UBIFS Wear—leveling on System I/O Performance[J]. Computer Engineering, 2009, 35(6): 260-262.

[39] BROWN N. JFFS2, UBIFS, and the growth of NAND Flash storage[EB/OL]. https://lwn.net/Articles/528617.

[40] KANG E, JACKSON D. Formal modeling and analysis of a NAND Flash filesystem in Alloy[M]// Abstract state machines, B and Z. Springer Berlin Heidelberg, 2008: 294-308.

(上接第 16 页)

[30] XIAO D, YANG Y, YAO W B, et al. Multiple-File Remote Data Checking for Cloud Storage [J]. Computers & Security, 2012, 31(2): 192-205.

[31] BARSOUM A F, HASAN M A. On Verifying Dynamic Multiple Data Copies over Cloud Servers [OL]. http://cacr.uwaterloo.ca/techreports/2011/cacr2011-28.pdf.

[32] CHEN H F, LIN B G, YANG Y, et al. Public Batch Auditing for 2M-PDP Based on BLS in Cloud Storage [J]. Journal of Cryptologic Research, 2014, 1(4): 368-378.

[33] ERWAY C C, KÜPÇÜ A, PAPAMANTHOU C, et al. Dynamic Provable Data Possession [C]// Proceedings of the 16th ACM Conference on Computer and Communications Security. 2009: 213-222.

[34] WANG Q, WANG C, REN K, et al. Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(5): 847-859.

[35] LIU C, ZHANG X, CHI Y, et al. Authorized Public Auditing of Dynamic Big Data Storage on Cloud with Efficient Verifiable Fine-Grained Updates [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(9): 2234-2244.

[36] ZHU Y, AHN G J, HU H X, et al. Dynamic Audit Services for Outsourced Storage in Clouds [J]. IEEE Transactions on Services Computing, 2013, 6(2): 227-238.

[37] JIN H, JIANG H, ZHOU K. Dynamic and Public Auditing with Fair Arbitration for Cloud Data [C]// IEEE Transactions on Cloud Computing. 2016: 1.

[38] TIAN H, CHEN Y X, CHANG C C, et al. Dynamic-Hash-Table Based Public Auditing for Secure Cloud Storage [C]// IEEE Transactions on Services Computing. 2016: 1.

[39] LIU C, RAJIV R J, YANG C, et al. MuR-DPA: Top-down Leveled Multi-Replica Merkle Hash Tree Based Secure Public Auditing for Dynamic Big Data Storage on Cloud [J]. IEEE Transactions on Computers, 2015, 64(9): 2609-2622.

[40] BARSOUM A F, HASAN M A. Provable Multicopy Dynamic Data Possession in Cloud Computing Systems [J]. IEEE Transactions on Information Forensics & Security, 2015, 10(3): 485-496.

[41] WANG B Y, LI H, LI M. Privacy-Preserving Public Auditing for Shared Cloud Data Supporting Group Dynamics [C]// Proceedings of the IEEE International Conference Communication. 2013: 539-543.

[42] WANG B Y, CHOW S, LI M, et al. Storing Shared Data on the Cloud via Security-Mediator [C]// Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems. 2013: 124-133.

[43] WANG B Y, LI B C, LI H. Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud [C]// Proceedings of the 10th International Conference on Applied Cryptography and Network Security. 2012: 507-525.

[44] WANG B Y, LI B C, LI H. Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud [J]. IEEE Transactions on Cloud Computing, 2014, 2(1): 43-56.

[45] WANG B Y, LI B C, LI H. Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud [J]. IEEE Transactions on Services Computing, 2015, 8(1): 92-106.

[46] YU Y, LI Y N. Public Integrity Auditing for Dynamic Data Sharing with Multiuser Modification [J]. IEEE Transactions on Information Forensics & Security, 2015, 10(8): 1717-1726.

[47] LUO Y C. Efficient Integrity Auditing for Shared Data in the Cloud with Secure User Revocation [C]// Proceedings of the IEEE Trustcom /BigDataSE/ISPA. 2015: 434-442.

[48] JOHNSON R, MOLNAR D, SONG D, et al. Homomorphic Signature Schemes [C]// Proceedings of the Cryptographers' Track at the RSA Conference. 2002: 244-262.

[49] BONEH D, GENTRY C, LYNN B, et al. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps [C]// Proceedings of the 22nd Theory and Applications of Cryptographic Techniques. 2003: 416-423.