一种适用于大容量 Flash 存储系统的管理方案

刘　锐，李盘林，李秉智
(重庆邮电学院 计算机学院，重庆 400065)

摘　要：针对嵌入式系统中大容量存储设备及其管理方面的需求,分析了传统的固定单元管理方式的弊端,提出了一种针对大容量 Flash 存储系统的灵活的管理方案。该方案通过实际访问量来动态地管理 Flash,提高了操作效率,同时也减少了资源占用,保证了系统的性能。

关键词：闪存；存储系统；存储管理；嵌入式系统
中图法分类号：TP343　　**文献标识码**：A　　**文章编号**：1001-3695(2006)02-0087-02

A Management Scheme for High Capacity Flash Memory Storage System

LIU Rui, LI Pan-lin, LI Bing-zhi
(School of Computer Science, Chongqing University of Posts & Telecommunications, Chongqing 400065, China)

Abstract: Aiming at the requests of high-capacity storage devices and the related management in embedded system, this paper analyzes the disadvantages of traditional fix granularity management scheme and proposes a flexible management scheme for high-capacity Flash memory storage system. The scheme adopts realistic access pattern to manage Flash dynamically, which improves the operation efficiency and reduces the resources usages, so as to ensure the system performance.
Key words: Flash Memory; Storage System; Memory Management; Embedded System

闪存(Flash Memory)是嵌入式系统中一种常用的存储介质,它是一种非易失、防震、节电的存储设备。近年来,随着容量和可靠性方面技术的新突破,Flash 存储系统更受人们青睐,因此被广泛应用于媒体存储。

嵌入式 Flash 存储技术和管理技术是目前的一个研究热点,尤其是由于 Flash 特征而引入的一些新挑战。与之相关的有逻辑地址与物理地址之间的转换、垃圾回收、Flash 文件系统的设计等。在 Flash 管理方面,通常的做法是采用静态表管理方式,即将 Flash 的扇区分成固定大小的单元(通常是 16KB),以这种单元为单位来管理使用过和未使用的 Flash 空间。这种方法对于小容量的 Flash 来说很有效,但随着 Flash 容量的增大,该方法面临着极大的挑战,其中一个突出的问题就是在系统启动及在线操作时系统性能急剧下降。

1　Flash 管理机制

通常,一个 Flash 由若干个扇区(Block)组成,每个扇区又由若干页(Page)组成。扇区是擦除操作的最小单位,而读和写都是以页为单位来进行的。在 NAND Flash 中,典型的扇区大小和页面大小分别是 16KB 和 512Byte。在每个页中有 16Byte 的空闲区域,存储内容以外的数据可以写到空闲区域。

Flash 内最新的数据被看作是有效(Live)数据,旧版本的数据被看作是无效(Dead)数据。相应地,包含新数据的页面被称作有效页(Live Page),包含旧数据的页面被称作无效页(Dead Page),空闲空间的页称作空闲页(Free Page)。在对大量的页写操作之后,系统需要回收这些分散在各个扇区中的无效页,使其变成空闲页。回收之前要先将其中的有效数据拷贝到新的空闲区域,因为擦除单位(Block)大于编程单位(Page),大量数据的拷贝可能被包含在块的回收中。

为了有效地管理 Flash 空间,常规的方法是采用静态表来处理地址转换和空间管理。因为 Flash 管理单元是一个固定参数,所以单元的大小对于内存和系统性能的均衡都非常重要。若管理单元设置小了,驻留在内存的管理表占用的空间必然增大,内存空间小了必然降低系统性能;相反,若增大管理单元,那么将会因为任何一小块数据的更新就会引起相同页面数据的整块数据重写、大量的数据备份,操作效率低,从而导致系统性能下降。为此,我们提出了可变单元动态管理 Flash 的思想。

2　基于可变单元的 Flash 管理模式

2.1　空间分配及回收

一个管理单元是一系列 Flash 页面的集合,相关数据结构驻留在内存,其状态可能是空闲(Free)、有效(Live)、清除(Clean)、废弃(Dirty)。一个空闲管理单元可以用于分配,一个有效的管理单元则意味着已被数据占用,一个废弃的管理单元表示该单元进入垃圾回收处理,而一个清除管理单元则表示还未进行垃圾处理。

管理单元采用二叉树的管理方式,每个单元被看作是树上的一个叶节点,不同层次的单元对应于不同的大小(以 2 为底的幂)。该树结构信息驻留于内存,初始化时的树结构就是一个由基于逻辑扇区地址(LBA)的空闲单元组成的层次结构。所有的内部节点都作一个标志 CLEAN_MARK,分割空闲单元和有效单元后,产生的内部节点分别标注为 CLEAN_MARK 和 DIRTY_MARK。

收稿日期：2005-01-29；修返日期：2005-03-26

在空间的分配上,我们采用伙伴系统(Buddy System) 的思想来划分。当一个新的请求到达之后,空间分配有两种情况。

(1) 存在满足请求的空闲单元。该结论显然是成立的,可以简化为背包问题来获得证明(这种空闲单元由 2^i ($i \geq 0$) 个页面组成) 。当写请求到达后,系统将为它分配一个足够大的空闲单元,如果选择的空闲单元远大于请求大小,则按照伙伴分配算法来进行分割。从树型结构的角度看,就是将父节点分割为两个子节点,再进行比较,反复进行下去,直至找到一个大小最接近于请求的空闲单元。新的数据将被写入产生的空闲单元中,然后该单元将变为有效单元。在分割的过程中,内部节点同时记录下各叶节点的状态。

(2) 所有空闲单元都不满足条件。为了处理这种情况,我们采用反复“归并”可用单元的方法(处理过和未处理的空闲单元此时都可归类为可用的单元) ,直到满足请求大小的空闲单元的出现。这种情况下,我们需要先证明可行性。

定理 设空闲单元的总大小是 M 页,请求空间是 N 页($M \geq N$) ,那么空间分配算法是正确的。

证明: 若存在任何可用的单元能满足请求,那么算法只需要选择一块合适的空闲单元,然后返回(可能要涉及一些分割) ,同第(1) 种情况。否则,所有可用的空闲单元都不能满足要求,可用单元都小于请求大小 N。为了满足要求,需要合并可用单元产生新的单元以满足需求。我们采用反证法,先假定不能正确地产生空闲单元满足要求。设 $N = 2^k$, k 是非负整数。根据条件,所有可用单元都小于请求大小 N,那么可用单元的大小只能在一个集合内: $T = \{ 2^0, 2^1, \ldots, 2^{k-1} \}$。考虑归并过程,如果不存在两个大小相同的单元,那么 T 内就没有相同的两个数。换句话说,整个空闲单元的大小等于 $2^0, 2^1, \ldots, 2^{k-1}$ 之和,由于该和小于 $N = 2^k$,因此与题设矛盾,故假设不成立。所以至少存在两个大小相同的空闲单元可以归并。依此类推,这样不断归并下去,直到产生一个空闲单元满足请求,这就证明了分配算法的正确性。

综合上面两种情况,我们设计了如图 1 所示的分配空间流程。

基于管理单元的垃圾回收机制是根据 CLEAN_MARK 和 DIRTY_MARK 来进行的,原理跟上面的归并思想类似。当一个废弃单元被选中清空,那么先找到与它匹配的无效子树,若与它同一个父节点下面的另一个子节点也是无效单元,那么无效子树上所有的数据都被拷贝到其他地方。在拷贝完数据之后,这些单元都变为废弃单元,然后可以用它们取代其父节点,把这些单元逐步归并到一个大的单元就可以对这个大的单元所包含的扇区进行擦除,使之变为空闲单元,以备以后使用。

图 2 是一个垃圾回收的例子,它显示了如何归并 B(待清除) 和 C(废弃) 。首先,将 C 的伙伴 B 里面的数据拷贝到 A (空闲) 内,数据从 B 拷贝到 A 之后,A 变为有效单元,B 就变为废弃单元。B 同 C 归并产生 D,它是一个新的废弃单元,准备回收。

2.2 逻辑 - 物理地址转换

前面指出,由于 Flash 的更新,有效数据随时都在变换物理地址。传统的方法是维持一个静态表,通过给定的逻辑地址来解决相关的物理地址。静态表建立索引对应相关的物理地址。

由于我们采用了动态单元管理方法,因此需要动态链表与之相适应。这里我们采用的是哈希(Hash) 表,它驻留于内存。表中每个哈希记录是一个解决地址冲突的链,每个链节点由起始的逻辑地址、起始的物理地址和包含的页的数目组成,它代表一个由连续地址的页组成的逻辑块(LC) 。

首先将 Flash 逻辑地址空间分割成大小相同的区域,即逻辑区(LR) 。假设整个逻辑地址空间从第 0 页到第 $2^n - 1$ 页,每个逻辑区都是 2^m 页,这样,总共有 2^{n-m} 个逻辑区记录。每个记录的头部指向一个逻辑块,所有的逻辑块都被映射到哈希表,如图 3(a) 所示。哈希函数由给定的逻辑地址($n - m$) 个比特来定义。当 Flash 分配空间或垃圾回收时,哈希表中所有的逻辑块也相应地被分割或归并,以反映数据逻辑地址的变化,如图 3(b) 所示。

3 性能仿真

为了测试系统性能,验证上面提到的 Flash 管理方案,这里我们采用了 20GB 的普通硬盘替代 Flash 作仿真算法,其扇区大小和页面大小分别是 16KB 和 512Byte,其中有 18GB 的有效数据已驻留在 Flash 中,即再写入 2GB 数据后将进入垃圾回收。我们准备在内存占用和 Flash 使用效率两个方面对系统性能进行测试,并把它同传统的固定管理单元方案进行比较。为此,对于固定单元管理方案也采用两张表,一张表用于逻辑地址到物理地址的转换,另一张表用于空间管理。每个记录占用 4Byte 的 RAM,管理单元从 512Byte(1 页) 到 32 768Byte(1 Block) 。仿真过程中真实写入的数据是 25. 8GB。

固定管理单元方案仿真结果如图 4 所示,图中实线和虚线分别表示不同的管理单元下占用 RAM 和写入数据的情况分布。由图 4 可知,若管理单元设置小了,驻留 RAM 空间就大;若设置大了,那么涉及到的 Flash 页面操作就多,两种情况都将导致系统性能低下。

本方案不再受固定管理单元的影响,经仿真测试,占用 RAM 大小为 22. 6MB,写入的总数据为 26. 8GB。 (下转第 95 页)