第34卷 第14期 计 算 机 工 程 2008年7月
Vol.34 No.14 Computer Engineering July 2008
·开发研究与设计技术· 文章编号：1000—3428(2008)14—0283—03 文献标识码：A 中图分类号：N945.12

一种闪存文件系统的数据恢复机制

张延园，焦 磊
(西北工业大学计算机学院，西安 710072)

摘 要：基于面向大容量NAND闪存的嵌入式文件系统CFFS，结合其对芯片上数据的索引方式，在CFFS中引入引用节点和页位图等数据结构和相应算法，提出一种数据恢复机制。该机制在数据遭到破坏时将闪存文件系统恢复到一个一致的历史状态，保证芯片上数据的一致性和可用性。

关键词：嵌入式系统；NAND闪存；文件系统；数据恢复

Data Recovery Mechanism for Flash File System

ZHANG Yan-yuan, JIAO Lei
(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072)

【Abstract】 Based on the embedded file system CFFS oriented towards high capacity NAND flash chips, and according to the approach that it indexes its data on the chip, this paper proposes a data recovery mechanism by introducing new data structures, such as indexing node and page bit map, and their corresponding algorithms to CFFS. In this mechanism, the flash file system can recover to a consistent historical state when its data is damaged so that the consistency and availability of the data are effectively guaranteed.

【Key words】 embedded system; NAND flash; file system; data recovery

1 概述

通常，对数据的威胁大致可以划分为两种：

(1)设备失败，包括：存储介质损坏，设备故障，或者在数据写入设备时发生掉电或由其他灾难而引起的写操作异常中断。

(2)来自用户的威胁，包括：用户删除，篡改数据，病毒活动等。这些威胁破坏了数据的一致性和可用性。

对于磁盘驱动器而言，RAID方案[1]在设备块的层次中很好地防止了设备失败；在文件系统的层次中，一些文件系统，例如JFS(Journaling File System)[2]，已经支持对写操作异常中断进行快速恢复；对于由用户误操作产生的数据威胁，某些文件系统，特别是数据库系统也可以进行恢复。

但对于闪存而言，特别是单个芯片容量就在GB级别的大容量NAND芯片，虽然有JFFS2支持对写操作异常中断的恢复，但尚没有任何闪存文件系统支持对用户误操作导致的数据被破坏的恢复。为此，笔者提出一种数据恢复机制。

闪存芯片与磁盘驱动器进行写访问的区别之一就是异位更新(update-out-of-place)。异位更新使针对闪存设计的文件系统具有日志结构(log-structured)[1,3]，CFFS也是如此。NAND芯片的物理及其电气特性决定了其必须按页读写。在写一页之前必须对其进行擦除操作，除非该页已被擦除过。但擦除操作是以擦除块为单位进行的，并且由文件系统的一个后台线程负责。该线程称为垃圾回收线程，其作用是回收过时数据所占用的芯片空间，使其可以被重新利用。该线程通常与文件系统模块是异步的，因此不能知道在何时哪一个过时数据块被回收了。但是，在大容量NAND芯片上，如果人为地停止垃圾回收线程，使对当前而言的过时数据得以保留下来，这就为把数据恢复到以前的历史状态提供了可能。但遗憾的是，目前主流的闪存文件系统(如JFFS2，YAFFS等)都不提供类似的数据恢复功能。

2 CFFS的基本设计

CFFS(Compact Flash File System)是在Linux 2.4内核下设计实现的一个针对大容量NAND闪存芯片的文件系统。JFFS2是另一个重要的闪存文件系统，不过在2.4内核下它只针对NOR芯片，不能被应用于NAND芯片上。

JFFS2存在两个主要的问题[4]，在设计新的文件系统时应尽量克服或减小这些问题带来的影响：

(1)当被用在大容量的芯片上时，文件系统的挂载时间相对较长。笔者在Pentium 4 2.80 GHz的处理器、1 GB内存的PC上，用内核自带的blkmtd模块将一个SCSI硬盘的1 GB大小的分区模拟成MTD设备[5]进行测试，挂载JFFS2需要超过1 min的时间。这是因为JFFS2在挂载时对整个闪存分区进行全扫描，这样当闪存分区较大时，由于I/O次数过多而导致挂载时间过长。

(2)由JFFS2的索引结构带来的影响。该结构的主体是一个链式的哈希表结构，而且每一个代表正规文件的哈希表的表节点又被组织到一棵红黑树中。这样，每个正规文件都有一棵自己的红黑树。当大容量芯片上存储的文件较多并且文件数据较为分散时，这些树将变得比较庞大，从而耗费了内存空间。

CFFS为了避免全芯片扫描，引入了“引用节点”和“页位图”两种机制来减少挂载过程所花费的时间。引用节点索引了芯片上当前所有的目录项和文件数据，因而代表了文件

基金项目：西北工业大学2007年度基金资助重点项目(W002205)
作者简介：张延园(1956－)，男，教授，主研方向：网络存储，网络软件与软件工程；焦 磊，硕士研究生
收稿日期：2007-10-02 **E-mail：**crazyjiaolei@gmail.com

系统的当前状态，以及芯片上所有当前被索引的页的状态；页位图则用来记录芯片上当前所有未被索引的页的状态。引用节点在芯片上按页集中存储，一个页紧凑地存储了若干节点，每个节点都记录了其后继节点的位置，一个页中的最后一个节点的后继节点就位于另外一个页中；页位图也集中存储在芯片上的连续的若干页上。如图 1 所示，该图假设页位图占 3 页。挂载 CFFS 时，无需全芯片扫描，只需要扫描所有的引用节点，再扫描页位图，即可在内存中建立起文件系统的索引结构。因为在页中集中存储的原因，一次读操作就可以读出若干引用节点，从而有效减少了 I/O 次数。这种方式经过测试，把文件系统的挂载时间减少了约两个数量级。

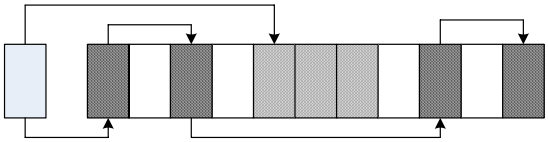

图 1　引用节点和页位图

在图 1 中，用反斜线标记的页为存储引用节点的页，用斜线标记的页为存储页位图的页。图中最前的一个有灰色阴影的页存储了一个静态超级块，它指示了页位图和第一个引用节点的位置。

CFFS 去掉了 JFFS2 的红黑树结构，只使用链式的哈希表作为文件系统在内存中的索引结构，并重新设计了目录和正规文件的表示、组织方式，从而使其占用尽可能少的内存空间。在用户使用的过程中，除了对芯片进行异位更新外，所有对索引结构的修改都只在内存中进行。这个索引结构即是挂载时采用引用节点和页位图建立的。索引结构的当前状态就是文件系统的当前状态。在卸载时，需要将这个索引结构中所包含的所有引用节点重新写回芯片，也需要重新计算各个页的状态得到新的页位图，并将其写回芯片。这些重新写回的引用节点和页位图供下次挂载时使用。如果在回写的过程中出现错误(例如掉电)，也将使 CFFS 进入一个不一致的状态，这也是需要设计实现数据恢复机制的一个原因。

3　CFFS 数据恢复机制

因为所有引用节点和页位图的位置在芯片上是任意的，要从芯片中读出引用节点和页位图，就需要知道第一个引用节点的位置，以及页位图的位置。由于每个引用节点都记录了其后继节点的位置，因此只需知道第一个引用节点的位置便可以读出所有的引用节点。CFFS 引入了静态超级块来存储第一个引用节点以及页位图在芯片上的位置，如图 1 所示。此外，静态超级块还存储文件系统的一些必要的统计信息，如整个分区的大小、擦除块的大小、页的大小等。

静态超级块只占一个页的大小。CFFS 使用芯片的前两个擦除块来存储静态超级块。格式化时将其写在第 0 块的第 0 页上。第 1 次挂载时从第 0 页读出静态超级块。然后从中读出第 1 个引用节点、页位图的位置，扫描它们从而在内存中建立 CFFS 的索引结构；在用户使用文件系统的过程中，进行文件系统操作，如创建、删除、读写目录或正规文件等，这时内存中的索引结构将发生变化，但在每一个确定的时刻它都代表了文件系统当前的状态；在卸载 CFFS 时，先回写当前内存中的索引结构所包含的所有引用节点到芯片上，再重新计算页位图并将其回写，再将它们在芯片上的位置记录到静态超级块的相应的域，最后回写静态超级块到第 0 块的第 1 页。下次就可以从第 0 块的第 1 页读出静态超级块来挂载 CFFS。以后每次卸载，静态超级块将依次被写入第 2 页、第 3 页……。如果第 0 个擦除块写满的话，就写往第 1 个擦除块，并可以擦除第 0 个擦除块；如果第 1 个擦除块写满的话，就写往第 0 个擦除块，并可以擦除第 1 个擦除块。这样 2 个擦除块就可以交替使用。

一次成功的卸载将使 CFFS 进入一个一致性状态，一个静态超级块就代表了一个一致性状态。经过多次的挂载和卸载后，芯片上会留下多个静态超级块，也就是保存了文件系统的多个一致性状态，如图 2 所示。可见，每个静态超级块都指示着自己的引用节点和页位图，这些引用节点和页位图代表了当时文件系统的状态。

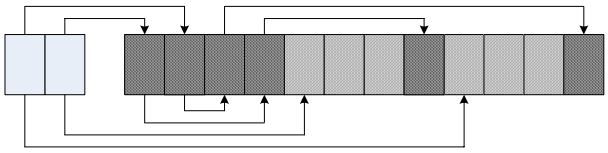

图 2　有两个静态超级块时的芯片布局

这些不同的静态超级块以及它们相应的引用节点、页位图，可以看成是一种快照(snapshot)，它们代表了不同时间点的文件系统的状态。一般情况下，在挂载时，CFFS 自动在前两个擦除块内寻找最近一次卸载时的静态超级块(见算法 2)，将其读出后，就可以接着读出当时的所有引用节点和页位图，从而建立起索引结构，这个结构就代表了最近一次卸载时文件系统的状态。但是，如果读取的不是最近一次卸载时的静态超级块，而是过去的某次卸载时的静态超级块，那么由该静态超级块将能读出当时的所有引用节点，从而在内存中建立起当时的索引结构。这时，内存中文件系统的索引结构就恢复到了当时的状态。这就是 CFFS 的把文件系统恢复到历史状态的基本思想。

在恢复过程中可以根据读出的引用节点来确定当时的所有被索引的页的状态，但不能读取相应页位图并由其确定当时的所有未被索引的页的状态，而必须进行全芯片扫描来确定这些页的状态。这是因为当时的页位图只记录了那个时刻的芯片上页的状态，而这些状态在从其后到当前为止的使用文件系统的过程中很可能已发生了变化，因此，这个页位图记录的状态对当前而言是不正确的。这时，需要像 JFFS2 那样进行全芯片扫描来确定各个未被索引的页的状态，从而建立正确的页位图，才能让文件系统恢复到当时的状态。

在实现时，分为动态数据恢复和静态数据恢复。前者是指在用户在挂载 CFFS 后的任意一个时刻，不用卸载而动态地将 CFFS 恢复到历史上的某个一致性状态；后者是指用户在没有挂载 CFFS 时，对 MTD 设备进行恢复，使得恢复完成后的第一次挂载可以得到历史上的某个一致性状态。

3.1　动态恢复的实现

对于动态恢复，笔者首先在CFFS中实现了目录的struct file_operations的ioctl()接口[6]。CFFS的挂载根目录的*i*节点号为 1，笔者有必要由此限制这个ioctl()函数只能对挂载根目录使用。恢复完成后，它的子目录的结构将发生变化。

当访问任意一个文件系统的某个目录或文件时，首先必须进行路径名解析。在这个过程中，Linux的VFS(Virtual File System)[2]会首先在它维护的dentry缓存和inode缓存中搜索路径上的每个目录或文件的dentry和inode，如果相应的缓存中

不存在要寻找的数据结构，则在缓存中创建它们。因此，如果用户挂载CFFS后访问了其上的某些目录或文件，而这些目录或文件在即将要恢复到的那个以前的状态中极有可能是不存在的，因此在恢复过程中作者需要恰当地销毁系统缓存中的这些dentry和inode。这样，动态恢复算法(算法 1)按以下 6 个步骤执行：(1)释放系统dentry缓存和inode缓存中的CFFS的除挂载根目录外的每个目录或文件的dentry和inode；(2)销毁当前内存中的索引结构，然后用指定的静态超级块及其所有的引用节点重建一个表示历史状态的索引结构；(3)对CFFS的其他数据结构重新初始化；(4)进行全芯片扫描确定各个未被索引的页的状态；(5)用重建的索引结构更新CFFS挂载根目录的inode的各个域；(6)对挂载根目录的inode重新分配一个dentry，并将其实例化到dentry缓存。

这样当用户再次从挂载根目录访问 CFFS 时，会发现其内容已经完全恢复到了历史状态。在进行上述恢复前，用户应保证所有访问 CFFS 的进程已经终止。动态恢复直接改变了文件系统的当前状态，因此，如果用户在挂载使用了一段时间后再进行动态恢复，因为恢复完成后将得到一个历史状态，所以从本次挂载到进行恢复前用户对文件系统所做的修改将全部丢失。如果想要保存这些修改，应在恢复前进行一次卸载。

3.2 静态恢复的实现

静态恢复与 CFFS 在挂载时寻找当前所需的静态超级块的算法密切相关，见算法 2(图 3)。

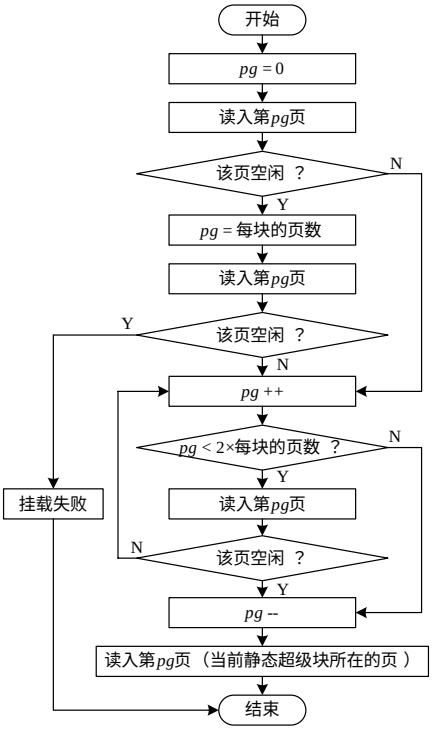

图 3　寻找当前静态超级块的算法

算法 2 中的 *pg* 是指从芯片的首页开始的页号。静态恢复就是在不用挂载文件系统的情况下，直接对设备进行恢复，使得恢复完成后的第一次挂载得到一个历史状态。由于不挂载文件系统而只针对设备进行操作，因此不会涉及 VFS。

由算法 2 可见，如果要将文件系统恢复到某个历史状态，只需要使其能在挂载的时候把过去那个历史时刻的静态超级块作为当前需要的静态超级块来读取。因此，静态恢复的算法见图 4。

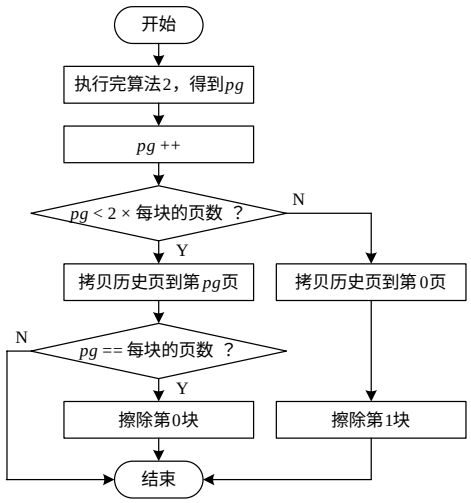

图 4　静态恢复的算法

静态恢复与动态恢复相比只是对前两个擦除块进行了扫描，并执行了一次拷贝操作，因此，速度很快。如图 5 所示，假设由 8 个页组成第 0 个擦除块。在执行拷贝操作之前，当前静态超级块位于第 2 页；现将第 0 块的第 1 页拷贝至第 3 页，这样位于第 3 页中的静态超级块将被作为当前静态超级块来读取，挂载后的状态就成了第 1 页所代表的历史状态。

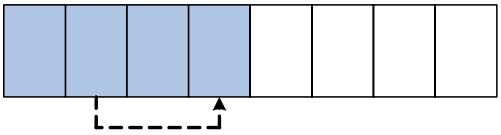

图 5　将第 0 块的第 1 页拷贝至第 3 页的示意图

4　结束语

CFFS 的数据恢复机制经过测试，效果较为满意，有效地保证了文件系统数据的一致性和可用性。

CFFS 的数据恢复与最初 CFFS 的设计紧密相关，其中很关键一点是引入了新的节点类型引用节点，它们在挂载时从芯片上读出，在卸载时又回写到芯片上，是对芯片上所有文件和目录的索引。CFFS 的数据恢复机制不论是对设备故障或是人为原因造成的数据破坏，都可以在没有利用额外存储空间的情况下就可以把闪存上的数据恢复到某个一致的历史状态，这一点与大多数的磁盘恢复策略不同，后者常需要额外的磁盘进行数据备份或组成 RAID 阵列。

参考文献

[1] Farley M. SAN 存储区域网络[M]. 孙功星, 蒋文保, 范　勇, 译. 北京: 机械工业出版社, 2002.

[2] Moshe B. Linux 文件系统[M]. 北京: 清华大学出版社, 2003.

[3] Bityutskiy A B. JFFS3 Design Issues[EB/OL]. (2005-11-02). http://www.linux-mtd.infradead.org/tech/JFFS3design.pdf.

[4] Woodhouse D. JFFS2: The Journaling Flash File System[EB/OL]. (2003-07-02). http://sourceware.org/jffs2/.

[5] Woodhouse D. Memory Technology Device Subsystemfor Linux[EB/OL]. (2005-03-02). http://www.linux-mtd.infradead.org/archive/index.html.

[6] Rubini A, Corbet J. Linux 设备驱动程序[M]. 魏永明, 骆　刚, 姜　君, 译. 北京: 中国电力出版社, 2002.