作为在日志末尾的一个邻接段写入磁盘。一个单独的段可能会包括i节点、目录块、数据块或者都有。每一个段的开始都是该段的摘要，说明该段中都包含哪些内容。如果所有的段平均在1MB左右，那么就几乎可以利用磁盘的完整带宽。

在LFS的设计中，同样存在着i节点，且具有与UNIX中一样的结构，但是i节点分散在整个日志中，而不是放在磁盘的某一个固定位置。尽管如此，当一个i节点被定位后，定位一个块就用通常的方式来完成。当然，由于这种设计，要在磁盘中找到一个i节点就变得比较困难了，因为i节点的地址不能像UNIX中那样简单地通过计算得到。为了能够找到i节点，必须要维护一个由i节点编号索引组成的i节点图。在这个图中的表项i指向磁盘中的第i个i节点。这个图保存在磁盘上，但是也保存在高速缓存中，因此，大多数情况下这个图的最常用部分还是在内存中。

总而言之，所有的写操作最初都被缓冲在内存中，然后周期性地把所有已缓冲的写作为一个单独的段，在日志的末尾处写入磁盘。要打开一个文件，则首先需要从i节点图中找到文件的i节点。一旦i节点定位之后就可以找到相应的块的地址。所有的块都放在段中，在日志的某个位置上。

如果磁盘空间无限大，那么有了前面的讨论就足够了。但是，实际的硬盘空间是有限的，这样最终日志将会占用整个磁盘，到那个时候将不能往日志中写任何新的段。幸运的是，许多已有的段包含了很多不再需要的块，例如，如果一个文件被覆盖了，那么它的i节点就会指向新的块，但是旧的磁盘块仍然在先前写入的段中占据着空间。

为了解决这个问题，LFS有一个清理线程，该清理线程周期地扫描日志进行磁盘压缩。该线程首先读日志中的第一个段的摘要，检查有哪些i节点和文件。然后该线程查看当前i节点图，判断该i节点是否有效以及文件块是否仍在使用中。如果没有使用，则该信息被丢弃。如果仍然使用，那么i节点和块就进入内存等待写回到下一个段中。接着，原来的段被标记为空闲，以便日志可以用它来存放新的数据。用这种方法，清理线程遍历日志，从后面移走旧的段，然后将有效的数据放入内存等待写到下一个段中。由此，整个磁盘成为一个大的环形的缓冲区，写线程将新的段写到前面，而清理线程则将旧的段从后面移走。

日志的管理并不简单，因为当一个文件块被写回到一个新段的时候，该文件的i节点（在日志的某个地方）必须首先要定位、更新，然后放到内存中准备写回到下一个段中。i节点图接着必须更新以指向新的位置。尽管如此，对日志进行管理还是可行的，而且性能分析的结果表明，这种由管理而带来的复杂性是值得的。在上面所引用文章中的测试数据表明，LFS在处理大量的零碎的写操作时性能上优于UNIX，而在读和大块写操作的性能方面并不比UNIX文件系统差，甚至更好。

4.3.6 日志文件系统

虽然基于日志结构的文件系统是一个很吸引人的想法，但是由于它们和现有的文件系统不相匹配，所以还没有被广泛应用。尽管如此，它们内在的一个思想，即面对出错的鲁棒性，却可以被其他文件系统所借鉴。这里的基本想法是保存一个用于记录系统下一步将要做什么的日志。这样当系统在完成它们即将完成的任务前崩溃时，重新启动后，可以通过查看日志，获取崩溃前计划完成的任务，并完成它们。这样的文件系统被称为日志文件系统，并已经被实际应用。微软（Microsoft）的NTFS文件系统、Linux ext3和ReiserFS文件系统都使用日志。接下来，我们会对这个话题进行简短介绍。

为了看清这个问题的实质，考虑一个简单、普通并经常发生的操作：移除文件。这个操作（在UNIX中）需要三个步骤完成：

1) 在目录中删除文件；

2) 释放i节点到空闲i节点池；

3) 将所有磁盘块归还空闲磁盘块池。

在Windows中，也需要类似的步骤。不存在系统崩溃时，这些步骤执行的顺序不会带来问题；但是当存在系统崩溃时，就会带来问题。假如在第一步完成后系统崩溃。i节点和文件块将不会被任何文件获得，也不会被再分配；它们只存在于废物池中的某个地方，并因此减少了可利用的资源。如果崩溃发生在第二步后，那么只有磁盘块会丢失。

如果操作顺序被更改，并且i节点最先被释放，这样在系统重启后，i节点可以被再分配，但是旧的目录入口将继续指向它，因此指向错误文件。如果磁盘块最先被释放，这样一个在i节点被清除前的系