# Linux 文件系统写放大作用的定量分析

黄伟亮[1]，王超峰[2*]
（1. 苏州博世汽车部件汽车多媒体部，江苏 苏州 215000；2. 苏州大学应用技术学院，江苏 苏州 215325）

摘　要：文件系统从软件的角度来说可以将其视为数据库，文件系统一般包含有索引、块管理和分配、权限管理、日志等机制，文件内容（裸数据）则是通过这些机制存储在磁盘上，因此实际写入到磁盘的量将大于文件本身数据。选取了一款 Linux 平台常用的文件系统 EXT4 为例，从分析文件系统内部设计入手对文件系统的写放大作用进行定量分析。

关键词：文件系统；放大作用；EXT4 文件系统；定量分析

DOI:10.16184/j.cnki.comprg.2019.07.008

## 1　概述

文件系统是一种存储和组织计算机数据的方法，是操作系统不可缺少的一部分，承担着磁盘的管理和数据的读取和写入工作。文件系统的性能直接影响着操作系统的性能。比如说，系统启动时间的性能直接受到磁盘读取性能的影响。

在嵌入式系统领域，通常存储器件是 Flash 类型的存储，比如 NOR,NAND（raw NAND, SD,eMMC），相比磁盘这些器件的特点是寿命较短。一般 NOR 和 SLC NAND 的擦写寿命为 100,000 次，MLC 的寿命则更短，所以写损耗和写均衡是嵌入式系统中文件系统的重点，存储器的寿命直接决定了嵌入式产品的最大寿命。拟定量解析文件系统的写放大作用，为嵌入式系统写损耗的评估提供一个理论依据。

## 2　EXT4 文件系统组成

将数据组织起来成为文件系统的数据成为元数据。其中索引、块管理和权限管理的相关的数据可以称为元数据，而文件系统的读写放大作用的原因之一就是元数据。

除元数据外，由于 EXT4 属于日志型文件系统，日志需要一定比例的块来存放，因此，本文将着重量化元数据和日志对读写放大作用的影响。

和内存管理中的 Page Size 概念一样，文件系统中的最小的单位是 Block Size。Block Size 是 512 字节对齐的，根据磁盘分区的大小，Block Size（以下简称 BS）也会有所不同。

EXT4 文件系统的结构分布如图 1 所示。

图 1 中，对一个 EXT4 文件系统来说，Super Block 的个数为 1，其余的功能块（Group0..N, Journal Block）是按照一定的比例关系，根据分区的大小在格式化的时候计算得出的。对 EXT4 文件系统来说，默认有针对小分区（几百兆字节以内），中等分区（几个 GB），大分区（几百 G）和超大分区（>1T）的格式化参数。这些参数可以在格式化的时候显式的指定成其他的值。

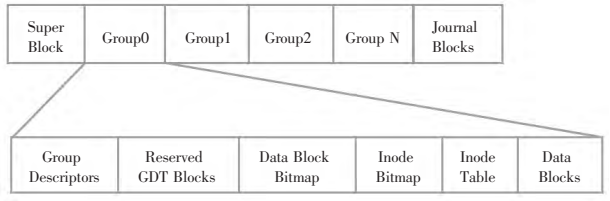


图 1　EXT4 结构图

功能块 Inode Bitmap 用来记录 Inode table 的使用情况的。当需要添加新的 Inode 描述时，文件系统会先在 Inode Bitmap 中查找空闲的可用的块，然后分配这个块。

类似的，Data Block Bitmap 是用来记录 data 块的使用情况的。当有数据需要写入到 Data Block 时，文件系统会从其中查找并分配空闲的块。唯一的区别是，数据块的个数是有需要写入的文件裸数据大小决定的，并且是按照 BS 对齐的。在查询可用的磁盘空间时，文件系统通过统计每个 Group 中空闲的数据块来得到具体的大小。

下面这张表直观的表达了不同分区大小，格式化工具所用不同参数格式化后的文件系统的参数，如表 1 所示。

作者简介：黄伟亮（1983–），男，本科，软件工程师，研究方向：嵌入式系统开发和应用；王超峰（1976–），男，通讯作者，讲师，研究方向：信息与图像处理、嵌入式系统应用。

收稿日期：2019–04–14

表 1　实际分区可用空间比例示例

| 分区大小 | BS | 块总数 | Inode 大小 | 日志大小 | 利用率 |
|---|---|---|---|---|---|
| 280MB | 1024B | 286,719 | 128B | 8MB | 93.23% |
| 15.9GB | 4096B | 15,552,508 | 256B | 128MB | 97.34% |

### 2.1　不包含日志功能的文件创建

对于文件本身，除了文件包含的数据，还包含文件一些属性，比如：文件长度、创建时间、修改时间、读取的时间、文件大小、文件的权限等，这些信息都会占用文件系统的空间。

从文件系统元数据的角度来说，创建一个文件主要需要更新和添加如下新的块：Data Block Bitmap, Inode Bitmap, Inode Table 和 Data Block。而对于更新块这样的操作，文件系统首先是读取 BS 大小的块到内存中，然后回写到介质上，这其中最小的读写操作的颗粒细度是 BS。

因此，可以得到计算公式：

$$\begin{aligned} BN_{nj}(file\_size) = {} & super\_block + block\_group\_desc \\ & + block\_BM + inode\_BM \\ & + Inode\_Table \times 2 + (file\_size / BS \\ & + (file\_size\%BS != 0)) \times BS \end{aligned} \tag{1}$$

例如，需要创建一个包含 200 字节的文件。因为文件本身裸数据的大小是 200 字节，当前文件系统的 BS=4KB，代入公式（1）：

$$6BS + ((200/4096) + 1) * BS = 7BS = 28KB$$

### 2.2　直接块寻址和间接块寻址

从 Inode Table 块中可以直接定位某个文件的数据所在的块，即 Inode Table 直接提供数据块的地址，称为直接块寻址。

如果磁盘碎片比较多，或者要写入的文件本身很大，导致无法提供连续的逻辑块给一个新创建的文件。这时直接块寻址就没法满足描述这种复杂的情形，需要用到间接寻址。Ext4 文件系统最多可以支持 3 级的间接寻址。

图 2 描述了一级间接寻址的实现：

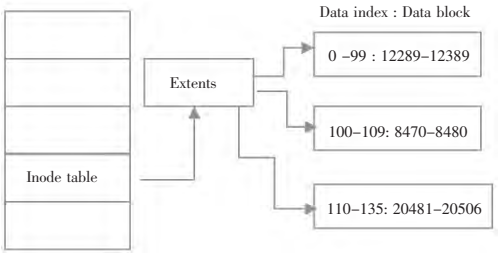


图 2　一级间接寻址

对于二级间接寻址来说，第一级 extent 指向的是第二级的 extent 而不是文件数据，而一个 extent 块可以包含 BS/3–1 个块地址描述，所以第二级 extent 将是第一级 extent 的指数倍。通常在第一级 extent 无法满足描述数据块实际分布情况时，文件系统会使用第二级 extent 描述。在实际使用中常见的是一级 extent。为了便于理解后文中的定量计算，因此在做定量计算的时候只考虑第一级 extent 的情况。

## 3　日志功能

EXT4 的日志工作原理是在创建文件过程中日志对文件系统重要的元数据甚至文件数据在更改之前在日志中进行备份，直到文件的完整数据被写入到物理介质上，日志记录一个结束的状态。

日志由日志头（header），日志描述块和日志结束（commit）块组成。一个完整的日志必须由上述 3 种块构成。格式化工具会分配一块固定大小的逻辑上连续的空间用作日志。日志的起始是日志的超级块（super），包含了日志的一些重要信息，日志的具体细节不在此文中赘述。

所以，如果要备份 N 个 BS 大小的数据。可以用如下的计算公式：

$$JBN_j = header + N_j + commit + super \tag{2}$$

其中，header 和 commit 的大小固定为一个 BS，日志的超级块需要更新，也是固定的一个 BS。

对于 N，在普通日志模式下，N 的计算公式为：

$$\begin{aligned} N_j = {} & super\_block + block\_group\_desc \\ & + data\_BM + inode\_BM \\ & + inode\_table * 2 + extent \end{aligned} \tag{3}$$

根据分区挂载时日志的相关选项，可以选择将文件数据和元数据做同样的日志记录。比如说，对于 EXT4 文件系统，如果挂载选项使用了 data=journal 的选项（数据日志模式），那么文件系统的日志将记录文件内容。这个功能对文件系统的稳定性起到了极大的作用，但是显而易见的是，它也将极大地拖累文件系统的性能。针对 data=journal 这一选项的公式则为：

$$\begin{aligned} JBN_{dj}(file\_size) = {} & header + N_{dj}(file\_size) \\ & + commit + super \end{aligned} \tag{4}$$

对于 N，在数据日志（data=journal）模式下，公式为：

$$N_{dj}(file\_size)=super\_block + block\_group\_desc + data\_BM + inode\_BM + inode\_table * 2 + extent + file\_size/BS + (file\_size\%BS! =0)*1 \quad (5)$$

综上所述，对于 EXT4 文件系统的读写放大作用的定量分析，结果是不包含日志功能文件创建和日志功能的计算公式之和，计算公式如表 2 所示。

表 2　不同日志模式下的计算公式

| 日志模式 | 详细 | 实际最小写入量 |
|---|---|---|
| 默认模式 | 只元数据 | $BN_{nj}$ + $JBN_j$ |
| 数据模式 | 元数据+文件数据 | $BN_{nj}$ + $JBN_{dj}$ |

例如，在默认日志模式下， 在 BS=4KB 的文件系统上，要写入 20KB 字节的文件，实际写入的数据的量为：

$BN_{nj}$（48$KB$）+$JBN_j$（40$KB$）=88KB

而在数据日志模式下，写入同样大小的文件，实际写入的数据的量为：

$BN_{nj}$（48$KB$）+ $JBN_{dj}$（45$KB$）=93$KB$

由上面的计算示例可以看到，放大作用大约为文件本身大小的一倍左右。

由于 Linux 操作系统是多线程的并发的，并且块设备层具有电梯算法，它会将对块的读写操作请求加入队列并且对块进行就近的合并。同时 EXT4 本身还有其他特性来提升性能，比如说 delay allocate。因此，元数据的写入在大多数情况下不是串行的，而是多个文件合并过后的。例如，元数据的延时写入，可以避免创建多个文件时的重复更新同一个块上的元数据。

## 4　结语

从文件系统设计结构的角度定量对文件系统放大作用进行了分析，并提出了可以作为嵌入式产品中文件系统对存储器件损耗的评估参考计算公式。在实际使用中，由于其还受系统 cache 策略、文件系统挂载选项、磁盘可用空间大小和磨损程度等因素的影响，在使用文中公式具体评估文件系统对存储器件的损耗时，要综合这些因素的考虑后再给出最后的评估结论。

参考文献

[1] https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout.
[2] 萨瓦，等. LINUX 教程 [M] . 北京：清华大学出版社，2005.

(上接第 11 页)

后期维护工作中，需要对软件进行定期的全过程跟踪，以便预防软件出现的 bug。

千行代码缺陷率在 CMM 体系中有着不同的标准：CMM1 级 11.95‰；CMM2 级 5.52‰；CMM3 级 2.39‰；CMM4 级 0.92‰；CMM5 级 0.32‰。软件在开发的过程中，出现 bug 是一种正常现象，但如果想要提高软件可靠性，还是需要对软件进行优化，将其 CMM 级别向着 CMM5 靠近。

## 5　结语

从学生的角度并结合团队中多次项目开发的经验，阐释出对于软件质量和软件质量管理的理解，并向读者阐述“小”团队对于软件软件质量管理上的缺陷，以及结合团队成员软件开发经验给予出软件开发流程中关于软件质量管理的几点建议，希望能够帮助“小”团队可以高效地进行软件开发。

参考文献

[1] 王巍. 软件测试过程中的质量管理及风险分析 [J] . 计算机产品与流通，2017，(12)：30.
[2] 张春生，于林. 软件质量管理体系研究 [J] . 商场现代化，2008，(2)：78.
[3] 王红霞. 软件工程化过程中的质量管理 [J] . 电子技术与软件工程，2018，133 (11)：91.
[4] 许琴. 论软件工程化过程中的质量管理 [J] . 电子技术与软件工程，2017，(16)：59–60.
[5] 吴保霖. 面向开发过程的软件项目质量管理 [J] . 智能城市，2016，(3)：34–35.
[6] 王海红. 软件项目实施过程中的进度管理研究[J] . 科技经济导刊，2016，(11).