计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
48(Suppl.): 175-180, 2011

Ext 文件系统检测和修复工具的研究与改进

卢雪山　戴华东　颜跃进
(国防科学技术大学计算机学院　长沙　410073)
(Shanxuel2008@163.com)

Research and Improvement on Ext Filesystem Checker

Lu Xueshan, Dai Huadong, and Yan Yuejin
(*College of Computer, National University of Defense Technology, Changsha* 410073)

Abstract　Though e2fsck can check and repair the corrupted data structure in the file system to restore it to a functional state, it still has bugs. In this paper we will analyze e2fsck' defect and its reason and present an algorithm to check and repair the node circles in the file system. At the same time we will rectify the wrong clearing links in e2fsk, and ameliorate the policy of allocating blocks for cloning dup blocks. All that above makes e2fsck more reliable. In order to make up for the loss of performance resulted from bettering function, we optimize the algorithm by incorporating the features of ext file system and e2fsck. The result of experiment shows that we not only mend the bugs of e2fsk, but also keep its performance well.

Key words　e2fsck; checker; file system

摘　要　e2fsck 能检测和修复 ext 文件系统损坏的数据结构，使文件系统恢复到可运行状态，但它还存在缺陷. 分析 e2fsck 存在的缺陷及其原因，提出一种检测与修复文件系统中节点循环的算法，同时改进了 e2fsck 错误的清除链接和克隆 *dup* 块时的块分配策略，增强了 e2fsck 的可靠性. 为了弥补功能改进带来的性能损失，结合 ext 文件系统特征和 e2fsck 的执行行为特性，优化了节点连接性检测的算法. 实验结果表明新算法改正了 e2fsck 的缺陷，并且没有降低它的性能.

关键词　e2fsck；检测与修复；文件系统

中图法分类号　TP316

随着计算机技术向各个领域不断渗透和蓬勃发展，系统中存储的数据成爆炸式增长，数据丢失的问题日益严重. 数据一旦丢失，重新取回代价昂贵，系统瘫痪和数据丢失造成公司和终端用户每年数十亿计的花费[1-2]. 文件系统在数据管理和保护上扮演中心角色，健壮的文件系统对避免数据丢失的作用举足轻重.

为了增强文件系统的健壮性，先进的文件系统技术如日志、快照等不断发展并得到应用. 日志文件系统(XFS, VxFS, Reiserfs, ext4, ZFS 等)，只要重放日志，而不必扫描整个文件系统来修复"未完成的写". 但重放日志是事件触发式的，而且日志可能意外损坏. 因此，日志不能作为文件系统从损坏中恢复的最终保障. 使用快照技术的文件系统(btrfs, ZFS, UFS2 等)，返回到一个旧的快照意味着系统在拍取这个快照之后的所有操作失效，这通常是系统不能

收稿日期：2010-12-10
基金项目：国家"八六三"高技术研究发展计划基金项目(2008AA01A203)

从损坏中恢复时才采取的最后手段. 总之,无论是日志文件系统还是使用快照的文件系统都必需配置一个检测与修复工具[3].

目前国产操作系统大多数采用开源的 ext 文件系统作为根文件系统. 这是因为 ext 文件系统具有健壮、简单、快速、易修复、使用的 cup 少,能很好地执行多线程的读和写等优点[4],同时它有配套的检测与修复工具 E2fsck. 针对 e2fsck 存在缺陷且相当复杂(3 万行 C 代码)不容易修改,Gunawi 等人提出了 SQCK[5]. 然而 SQCK 需要额外的存储空间和数据库支持,同时它的性能也不如 e2fsck. E2fsck 利用文件系统磁盘上的元数据进行修复[6],没有其他的依赖条件,运行速度快. 因此,改正 e2fsck 的缺陷是明智的.

本文分析了 e2fsck 存在的缺陷和原因,提出一种检测与修复文件系统中节点循环的算法,同时改进了 e2fsck 错误的清除链接和克隆 *dup* 块时的块分配策略,增强了 e2fsck 的可靠性. 为了弥补功能改进导致的性能损失,结合 ext 文件系统特征和 e2fsck 执行行为特性,对 e2fsck 检查节点连接性的算法进行了优化. 实验结果表明我们改正了 e2fsck 的部分缺陷,并没有降低它的性能.

1 e2fsck 行为分析

1.1 extX 文件系统的磁盘布局和错误注入

extX 将整个文件系统划分为许多个"块",用"块"(若干个连续的扇区组成)作为数据存储单位,并将所有的块分为若干个块组,在没有激活"稀疏超级块"[7]特性时,每个块组的结构[8]基本相同,如图 1 所示. linux 默认激活"稀疏超级块"特性,只在某些块组中存放超级块和组描述符的备份,而并不是每个块组都有超级块和组描述符的备份. 如果一个块组含有超级块和组描述符备份,则超级块占用该块组的前两个扇区,组描述符起始于该块组的第 2 块. 否则,该块组第 1 个块被块位图块占用,第 2 个块被节点位图占用. 文件系统默认使用块组 0 的超级块和组描述符,其余保持不变. 因此,通常所说的超级块为块组 0 的超级块. 在 extX 文件系统中,0～1 号扇区为保留区,超级块总是位于文件系统的 1024 B 处,并占用 1024 B 空间.

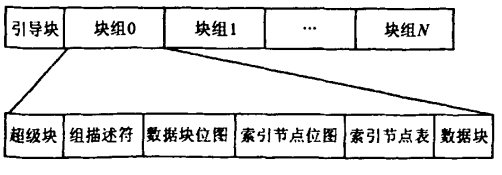


图 1 extX 文件系统磁盘布局

为了理解 e2fsck 执行中的行为,我们除了分析它的源码,还需要根据分析结果在磁盘上注入各种可能的错误,来验证 e2fsck 可能存在的缺陷. 我们假设要检验的 e2fsck 缺陷是已知的,所以要在熟知文件系统数据结构和磁盘物理结构的基础上,针对缺陷注入各种可能的错误. 在这里,我们使用 hexedit 修改磁盘上的部分元数据,注入预先设计好的错误.

1.2 e2fsck 工作过程

E2fsck 检测与修复 extX 文件系统分为 6 个阶段[6],如图 2 所示. 它通过检测与修复文件系统磁盘上的元数据,使文件系统从损坏中恢复到可运行状态. E2fsck 第 1 阶段需要扫描文件系统中所有的节点和块信息,并为后面的检测搜集信息,它占用大部分 e2fsck 时间. 第 3 阶段要检测信息存储在内存中,因此速度很快,几乎可以不考虑它的耗时.

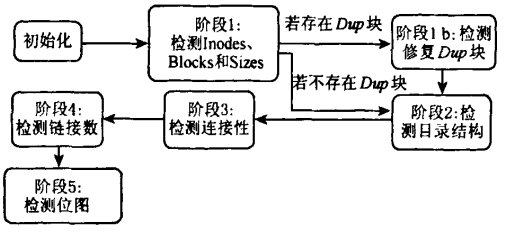


图 2 e2fsck 检测流程

1.3 e2fsck 的缺陷分析

从错误注入实验中发现,e2fsck 至少存在 4 个方面的问题:1)检测文件系统连接性算法存在错误. 它检测不到不在文件系统树上的节点循环,造成节点丢失;2)错误的清除链接,导致错误的父子关系[5];3)克隆 *dup* 块的块分配策略使用不当,可能形成文件和目录碎片,降低文件系统操作的性能;4)不安全的修复,即复制数据自由.

下面对这些问题进行详细的描述:

检测文件系统连接性算法存在错误:e2fsck 检测不到不在文件系统树上的节点循环,节点丢失. 连接性检测即检测节点是否在文件系统树上,并检测是否存在节点循环. 如图 3 所示,节点 D_1,D_2,D_3 形成的循环不在文件系统树上,e2fsck 检测不到这个

循环的存在,但是文件系统节点位图标记这些节点是已经使用的,同时它们引用的数据块也被继续占用.因此,文件系统丢失了这个节点循环上的所有节点以及它们引用的数据块.

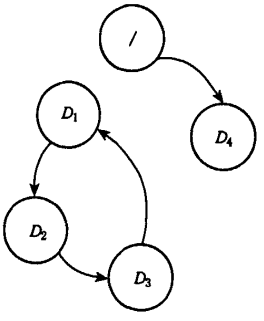


图 3 不在文件系统树上的节点循环

错误的清除链接:错误的父子关系[5]. E2fsck 在初始化父节点信息时,默认读到的第 1 个指向它的目录项节点为其父节点,如果还有其他节点指向该节点,则认为只是一个链接,可以清除.如图 4 所示,图 4(a)表示文件系统损坏之前;图 4(b)显示 D_1 损坏后指向了 D_3.检测与修复过程中,e2fsck 初始化 D_1 为 D_3 的父节点,并认为 D_2 是指向 D_3 的一个链接,应该清除,并修改 D_3 的".."目录项信息为 D_1;图 4(c)是修复后的状态.

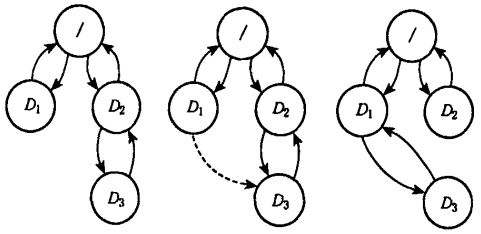


图 4 e2fsck 错误的清除链接

克隆 *dup* 块的块分配策略:两个或多个 *i*-node 指向的块称为 *dup* 块.在克隆 *dup* 块时,块分布策略与 ext 文件系统不同,可能形成文件和目录碎片,降低文件系统操作的性能.例如,同在 10 号块组的两个节点指向块组内同一个块,e2fsck 修复后,其中一个节点将指向 0 号块组的一个块.

不安全的修复:复制数据自由.如果用户运行 e2fsck 使用了"*-y*"参数,一旦发现 *dup* 块就克隆该数据块,没有考虑这种策略可能泄露私人信息.例如,一个数据块被两个索引节点共享,一个节点在/home/guest 目录,另一个在/root 目录,e2fsck 会拷贝一个新的块给/root 目录,尽管我们可能要删除 guest 的指针而保留 root 的指针.

总之,e2fsck 在功能上还存在不足,需要进一步研究与改进.

2 e2sck 的改进

2.1 节点连接性检测算法改进

当前的节点连接性检测算法存在以下缺点:1)E2fsck 的循环检测需要维持一个 inode_loop_detect 位图,开销大,尤其是大文件更为明显;2)判断是否存在循环的根据(计数器是否超过 2048)不是很精确.计数器超过 2048 不一定就存在循环,因为文件系统的一个块组中节点数可以超过 2048,倘若存在循环一定要循环 2048 次后才开始检测是否有循环;3)循环检测算法的实现过程中存在缺陷,它不能检测到不在文件系统树上的节点循环.

针对原算法不能检测系统中节点循环,我们对其进行改正,改正缺陷后的算法如算法 1 所示.由于该算法在资源利用和性能上存在不足,我们根据文件 ext 系统的特性对改正后的算法进行了优化,如算法 1 所示,采用一个长整数字段 save_node 来替代 inode_loop_detect 位图,记录可能存在循环的目录中的某个节点号,在往上遍历父节点时进行一次比较,若存在循环必回到该节点.

算法 1. 改正缺陷的节点循环检测算法.

```
while(1){(标记 ino 节点为 done)/* ino 被标
记过返回 1*/
  取 ino 节点的父节点 parent;
if(! parent||(loop_pass &&
  inode_loop_detect 位图 parent 被标记))
          连接 ino 节点到 lost+found;
          修复 dotdot 信息;
        Parent=lost+found;
          break;}
if(parent==root)break;
       ino=parent;
if(loop_pass){
    标记 inode_loop_detect 位图中的 ino;
       }
else if(parent_count++>2048){
       loop_pass=1;
```

```
if(inode_loop_detect)
    inode_loop_detect 位图清 0;
else  分配 inode_loop_detect 位图;
    }
ino=dir;}
```

算法 2. 性能优化后的节点循环检测算法.

```
while(1){  标记 ino 节点为 done;
              取 ino 节点的父节点 parent
    if(donecount==1&& 父节点存在
    && 父节点的标记为 done)
    {  if(第 1 次循环 &&ino 节点! =父
    节点)
            break;
          save_inode=父节点;
          donecount=0;
        ino=parent;
        continue;
          }
    if(parent==root)break;
    if(save_inode==parent)存在循环;
    if(! parent||存在循环){
        连接 ino 节点到 lost+found;
        修复 dotdot 信息;
      Parent=lost+found
        break;
        }
  ino=parent;
  ino=dir;}
```

由于文件系统很少出现循环,因此,在检测循环的方面应尽可能减少开销. 同时要以尽量快的速度扫描完已使用的节点. 为了加快扫描文件系统树速度,根据文件系统中通常父节点号小于子节点号以及 e2fsck 顺序扫描节点的特点,我们在被检测的节点标记为 done 之后,检测其父节点是否已经检测过. 如果被检测节点的父节点也标记为 done,则可以认为被检测节点在文件系统树上,因而可以结束对该节点的检测.

改进后的算法解决了 e2fsck 不能检测节点循环的缺陷,同时减少了内存的开销.

2.2 错误的清除链接的改正

当两个或多个节点指向通一个子节点时,子节点的".."目录项包含了该节点的父节点信息. 通过比较这些父节点和子节点的".."目录项包含节点的节点号能够判断谁是正确的父节点. 若存在某个父节点节点号与子节点的".."目录项包含节点相同,则认为该节点才是子节点的父节点,而其他父节点为指向该子节点的链接. 如果这些节点中没有与子节点的".."目录项包含节点相同,则默认读到的第 1 个指向子节点的节点为父节点.

2.3 块的分配策略改进

e2fsck 在复制 *dup* 块时,分配新块的策略 ext 文件系统分配块策略不一致,它总是从文件系统中 0 号块开始搜索,找到空闲块则分配用来拷贝. 考虑到不改变块分配的程序,我们设置 e2fsck 分配块起始位置为一个动态参数,这个参数由引用这个块的节点号来决定,即引用这个块的节点所在块组的第一个块.

3 评 估

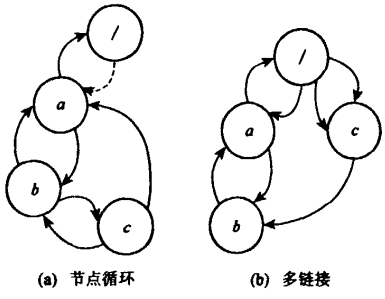
改进后的 e2fsck 的评估分为两个方面,即功能和性能. 我们针对 e2fsck 的改进分别注入各种可能的错误进行测试. 同时我们对 e2fsck 改进前后的性能进行了比较. 总而言之,改进后的 e2fsck 到达了预期目的.

3.1 功能评估

为了进行功能评估,我们使用 hexedit 直接修改文件系统磁盘元数据,注入了 3 类特定的错误:节点循环、多链接和 *dup* 块.

节点循环错误注入:修改节点的目录文件,使文件系统中节点形成圈. 如图 5(a)所示,我们先在根节点的 root 目录文件中删除目录项 a,再在节点 *c* 的目录文件中添加目录项 *a*,最后修改 *a* 目录文件的".."目录项为目录 *c*,这样形成了节点 *a*→*b*→*c*→*a* 的循环圈.

多链接注入:修改节点目录文件,使该节点的目



(a) 节点循环 (b) 多链接

图 5 文件系统错误注入

录文件包含其他目录的子目录. 如图 5(b)所示,修改节点 c 的目录文件,添加子目录 b,这样就形成了 $a \to b$ 同时也有 $c \to b$ 的链接.

Dup 块注入:修改节点的直接块或间接块指针,使其指向已被其他节点引用的块. 如图 5(b)所示,我们修改节点 c 的直接块指针,使其指向节点 a 指向的块,这样就注入了 *dup* 块.

显然,功能评估分为 3 个方面. 如表 1 所示,1)为了确保改进后的 e2fsck 能检测与修复文件系统中出现的节点循环,需要测试两种错误:在文件系统树上的节点循环和不在文件系统树上的节点循环圈. 每种错误我们都注入两个以上. 2)为了证实改进后的 e2fsck 能正确地清除链接,我们注入了 5 种损坏. 它们包括了两个或多个节点指向同一子节点的多种情况. 3)我们注入了 15 种 *dup* 块情形来验证改进后 e2fsck 分配块策略比改进前优越.

表 1 e2fsck 缺陷改进前后修复对比

错误	节点循环检测	清除链接	块分配策略
改正前	0/5	0/5	15/15
改正后	5/5	5/5	15/15

注:后一项数据表示注入的错误数,前一项数据表示改进的数目. 如"0/5","5"为主如错误数,"0"为改进的错误数.

实验结果表明,我们改进了 e2fsck 在节点循环检测和清除链接方面的缺陷,同时我们对上述实验中给 *dup* 块分配的块进行分析,发现改进前分配的块都在第 1 个块组中,而改进后分配的块分散在指向它的索引节点所在块组. 改进了块分配的合理性.

3.2 性能评估

性能评估主要测试 e2fsck 功能改进后工作性能是否降低. 实验环境:2.0 GHz Inter Core2 Duo E2180 CPU、2 GB 内存、250 GB WDC WD2500AAKS 硬盘的计算机,Linux2.6.18,e2fsck1.41 和 hexedit1.2.12.

我们设置了 3 个大小分别为 1 GB,10 GB 和 100 GB 的文件系统,每个文件系统已使用空间都为 80%. 测试了节点连接性检测算法性能和 e2fsck 改正前后的运行时间.

节点连接性检测算法性能主要通过检测完所有目录节点时算法扫描目录节点的次数(即循环次数)来表示. 如表 2 所示,算法优化前后循环次数差别相当大. 算法优化前检测目录节点的循环次数随目录数量的增加呈指数型增长. 优化后的算法,通常情况下,循环次数接近目录节点的数目. 显然,优化后的算法性能有很大提高.

表 2 节点连接性检测算法性能优化前后比较

扫描目录数	目录数			
	50	624	978	3372
算法优化前	200	2476	4583	25113
算法优化后	54	628	984	3381

改进前后的 e2fsck 整体性能如图 6 所示,e2fsck 在改进前后运行时间基本相同:

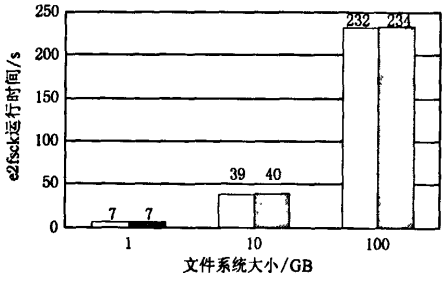


图 6 e2fsck 改进前后的运行时间

实验结果表明,改进后的 e2fsck 在性能上没有降低,完全达到预期目标.

4 结束语

本文分析了 e2fsck 的缺陷,提出了一种检测 ext 文件系统节点循环的算法,改进了错误的清除链接算法和克隆 *dup* 块的块分配策略. 实验显示,改进后 e2fsck 在性能上没有降低. 在此遗留的克隆 *dup* 块的安全问题将是我们下一步要解决的关键问题.

参 考 文 献

[1] Demsky B, Rinard M. Automatic detection and repair of errors in data structures //Proc of OOPSLA'03, 2003

[2] Bairavasundaram L N, Sundararaman S, Andrea C. Arpaci-Dusseau, et al. Torlerating file-system mistakes with EnvyFS //Proc of USENIX'09

[3] Funk R. fsck / xfs. http://lwn.net/Articles/226851/

[4] Henson V, Brown Z, Ts'o T, et al. Reducing fsck time for ext2 file system. Linux Symposium, 2006, 1: 395-407

[5] Gunawi H S, Rajimwale A, Arpaci-Dusseau A C, et al. SQCK: A declarative file system checker //Proc of the 8th Symp on Operating System Design and Implementation (OSDI'08). 2008: 131-146

[6] McKusick M, Joy W, Leffler S, et al. Fsck-The UNIX File System Check Program. Unix System Managers Manual-4. 3 BSD Virtual VAX-11 Version, 1986

[7] 马林. 数据重现——文件系统原理精解与数据恢复最佳实践. 北京: 清华大学出版社, 2009

[8] Bovet D P, Cesati M, 陈莉君, 张琼声, 张宏伟译. 深入理解linux内核. 中国电力出版社, 2008

卢雪山 男,1984年生,硕士研究生,主要研究方向为文件系统和文件系统检测与修复.

戴华东 男,1975年生,博士,副研究员,主要研究方向为系统软件、体系结构和并行计算.

颜跃进 男,1976年生,博士,助理研究员,主要研究方向为操作系统、容错计算机.