第34卷 第17期 计 算 机 工 程 2008年9月
Vol.34 No.17 Computer Engineering September 2008
·软件技术与数据库· 文章编号：1000—3428(2008)17—0044—03 文献标识码：A 中图分类号：TP311

SmallFS的研究与实现

谢 俊，刘海涛，管海兵
(上海交通大学计算机科学与工程系，上海 200240)

摘 要：当同一目录下存在大量文件时，对文件的线性检索造成传统的Linux文件系统(如ext2)性能有很大的下降。该文提出的SmallFS旨在解决该性能下降问题。SmallFS属于存储优化的一种，它对特定目录下的文件检索过程进行优化，能在较大程度上提高检索效率。此外，SmallFS基于可堆叠技术实现，具有较好的兼容性和对用户的透明性。

关键词：SmallFS系统；存储优化；可堆叠文件系统

Research and Implementation of SmallFS

XIE Jun, LIU Hai-tao, GUAN Hai-bing
(Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】Traditional Linux file system such as ext2 performs poorly if large numbers of small files exist. This is due to the linear search algorithm of the file system. This paper implements SmallFS to solve this problem. SmallFS is one kind of storage optimization. It optimizes the search algorithm to greatly improve the performance. SmallFS is based on the stackable file system technique, so it has good compatibility and is transparent to users.

【Key words】SmallFS; storage optimization; stackable file system

1 概述

在实际应用中，文件系统的某些目录下，可能会存在大量的文件，如：

(1)网络浏览器的页面缓存目录；

(2)网络流量分析工具存放抓包结果的目录；

(3)新闻组服务器存放文章和消息的目录。

在Linux环境里，目录被作为一种特殊的文件来处理。以最常用的Ext2文件系统为例，每个文件都由一个ext2_dir_entry结构来描述，同时一个目录在逻辑上是它的所有子文件的ext2_dir_entry结构的集合。因为不同文件的文件名长度可能是不同的，所以出于节省空间的目的，不同ext2_dir_entry的大小并不相同，如此使得检索目录下的文件不能采用类似数组索引的方式快速计算出文件在磁盘上的位置，只能通过线性查找的方式来定位。当一个目录下存在大量文件时，文件的检索效率会线性降低。此外根据统计，文件检索操作在所有文件操作中占了很大比例(40%~50%)[1]，所以对于开始介绍的几种应用环境，必须采用优化方法来提高系统的性能。

本文提出了SmallFS，该文件系统很好地解决了这个问题。SmallFS通过采用可堆叠技术来实现，具有良好的可扩展性和对用户的透明性。此外，SmallFS目前已在Linux下实现，但是该系统可以很容易地移植到其他Unix/类Unix操作系统下。后文的性能测试说明，对于常见的文件操作，SmallFS能够很好地改善文件系统的性能。

2 背景

2.1 相关工作

XFS[2]是Silicon Graphics为IRIX操作系统开发的文件系统，对拥有大量子文件的目录有比较好的支持。它使用B+树的形式存储文件，取代了传统文件系统中最常用的以线性结构存储文件的方式。因此，XFS在目录存在大量子文件的情况下，比IRIX操作系统原有的EFS文件系统有着更高的性能。但是，XFS只能运行在IRIX操作系统之上。

reiserfs是Linux下的一种文件系统，使用平衡树的形式来优化性能和提高磁盘空间使用率。reiserfs通过平衡文件和文件名来重新组织目录，以获得更快的访问速度。reiserfs虽然是免费的，但是稳定性不高。

usenetfs[3]是专为优化新闻组服务器的性能开发的一种文件系统。usenetfs的优化方法特别要求文件系统中的文件以数字命名，因此，并不适用于一般的包含大量子文件的目录。

2.2 可堆叠文件系统

本文提出的SmallFS采用可堆叠文件系统实现。可堆叠文件系统[4]允许开发人员以模块化的、增量开发的方式将功能添加到其他文件系统之上，为用户提供了组合多种文件系统功能的方法，很大程度上降低了开发的难度和工作量。如图1为采用可堆叠技术的SmallFS的系统结构。

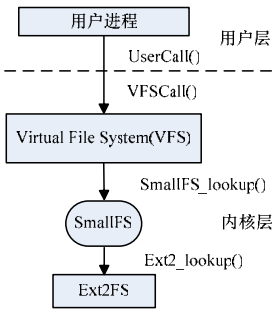


图1 SmallFS系统结构

基金项目：国家自然科学基金资助项目(60503013)
作者简介：谢 俊(1983－)，男，硕士，主研方向：文件系统；刘海涛，讲师、博士；管海兵，教授、博士生导师
收稿日期：2007-09-21 **E-mail：**xiejun.better@gmail.com

用户进程的文件系统调用首先被转为对 VFS 的调用，其后 VFS 会相应地调用 SmallFS 的函数。在 SmallFS 中将 VFS 传递过来的 I/O 路径进行计算，并将计算出的新 I/O 路径传递给底层的 Ext2FS，Ext2FS 按照新的 I/O 路径执行 I/O 请求。

由于可堆叠文件系统是模块化的，实现不同功能的模块(如 SmallFS)是独立于其上层和下层模块的，因此可以将不同的模块自由组合，具有很好的兼容性和可移植性。此外，它对用户是透明的，所有功能操作不需要用户的参与。

3 SmallFS 的设计与实现

3.1 可能的解决方案

对于同一目录下存在大量文件导致的性能下降问题，目前的解决方案包括：

(1)改变检索过程。通过修改底层文件系统，用比线性检索更快的方式实现文件检索操作。这往往涉及到文件组织结构的改变。缺点是开发工作量大，而且因为修改的是特定的底层文件系统，所以兼容性很差。

(2)将多个小文件进行合并，以减少目录下文件的数量。由于将多个文件合并为一个文件，因此必须有一种索引机制来计算原文件在新文件中的偏移量，例如索引文件[5]。这种解决方案的缺点是开发工作量也比较大，除了文件检索操作，也要修改其他的文件操作以保证能正确地读写文件，更要保证索引和原文件的一致性。

(3)改变文件目录的组织结构，以减少单个目录下文件的数量。此解决方案的优点在于不需要对底层文件系统进行重大修改，只需要对 I/O 路径进行变换，使得原来存放在一个目录下的大量文件被分散存放在该目录下的多个子目录中，而文件读写接口保持不变。

本文提出的 SmallFS 在对以上 3 种解决方案进行深入研究后，决定采用修改文件目录结构的方法。

3.2 SmallFS 的设计思路

SmallFS 的设计目标是：高独立性，即独立于底层，以便于移植；有针对性，即可以有选择地决定进行路径变换的目录，可以仅作用于有大量子文件的目录而不影响其他目录；高性能，要求不仅能够提高系统的性能，而且要求本身的性能开销小。

SmallFS 的设计思路是将存放大量文件的目录在物理上切分成许多子目录，所有对原目录下文件的访问都会被透明地转换为对子目录下的文件的访问，如图 2 所示。

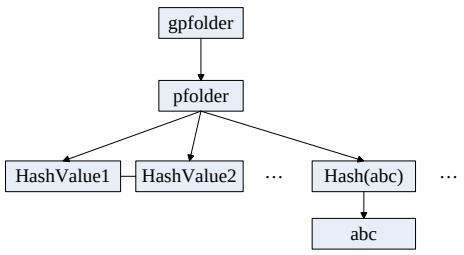


图 2 SmallFS 作用下的目录结构

SmallFS 通过对某文件访问的 I/O 路径进行 Hash 计算来得到转换后的新 I/O 路径。如原本访问的 I/O 路径是 gpfolder/pfolder/abc，Hash 函数为 y=Hash(x)，则转换后的新 I/O 路径是 gpfolder/pfolder/Hash(abc)/abc。文件名并不进行改变，而仅仅是在文件系统中的 I/O 路径进行了改变。所有涉及到目录下原文件的操作都会进行路径变换，如新建文件，会在相应的子目录中建立文件，再如检索某目录中的所有文件则会遍历该目录下的所有子目录下的文件等。

如前所示，SmallFS 是独立于底层实际文件系统的一个文件系统模块，因此用户可以有选择地将 SmallFS 动态挂载到一个特定的目录上，仅对此目录进行优化，并不会影响其他目录。因为在实际应用中，存在大量子文件的目录在整个文件系统中所占比例不大，用户可以判断出用于特定目的的目录，所以让用户手动挂载是较好的做法。

在 SmallFS 的设计中，对目录下的所有子文件和子目录都将进行 I/O 路径变化。在 Linux 系统中，目录本身作为一种特殊的文件存储，因此，SmallFS 对目录下的子文件和子目录按同样的方式处理。

值得注意的是对 I/O 路径中“…”的处理，因为用户在逻辑上并不需要知道子文件夹的存在，所以在 SmallFS 的设计中“…”将返回更上一层的目录。举例来说，如果在 gpfolder/pfolder/Hash(abc)/中发生了对“…”的检索，操作返回的是 gpfolder/而不是 gpfolder/pfolder/。

此外因为 Hash 函数是单向的，所以不对文件名进行处理，这样就可以对一个经过 SmallFS 处理的目录进行恢复。

在 Linux 系统中，为了提高文件的检索效率，从磁盘中读入内存的 dentry 结构会被保存在缓存队列中，当下一次检索该文件时会先从缓存队列中查找，以加快速度。该缓存队列其实是一个 Hashtable，里面的每一项是一个指针，指向一个 dentry 队列。这样在查找缓存时，先通过文件名的 Hash 值找到某个队列，再在这个队列中进行查找。可以看出，Linux 检索文件时已经对文件名进行了一次 Hash 计算，所以在 SmallFS 的设计中，直接使用这个已经计算出的 Hash 值，以避免额外的 Hash 运算带来的计算开销。但是，采用这种方式会导致一个新的问题，即 Linux 所采用的 Hash 算法中 Hash 值的冲突几率很小。在实际测试中，10 000 个文件可能有 10 000 个不同的 Hash 值，这样就带来了每个子目录中只有少量文件的问题，反而降低了文件检索的性能。SmallFS 采用了另外一种方法解决了该问题，SmallFS 在取得 Hash 值后，仅取该 Hash 值的后 3 位作为子目录名。即经过 SmallFS 处理后的目录的子文件将被放入名为 000~999 的子目录的其中一个，继续以前一个例子为例，确切地说新 I/O 路径应为 gpfolder/ pfolder/Hash(abc)末 3 位/abc。

3.3 SmallFS 的实现

SmallFS 通过修改一层空白的可堆叠层来添加 I/O 路径转换功能。算法的伪代码如下：

```
hash = dentry->d_name.hash;
hash %= 1 000;
hash_dentry=GetDEntryByName(hash_value, parent_dentry, strlen
(hash_value));
MakeDir(parent_dentry->d_inode,hash_dentry,
current->fs->umask);
real_dentry = GetDEntryByName(name,hash_dentry, namelen);
```

首先获得文件的 Hash 值并取末 3 位，其后在所在目录中查找文件存放的子目录，若该子目录不存在，则用 Hash 值的末 3 位新建一个子目录，最后在子目录中查找并返回查找文件的真实 dentry。所有的文件操作实际都被转换为对真实 dentry 进行操作。

4 性能测试

本文对 SmallFS 进行了测试，测试环境的配置为 1 GHz CPU，512 MB 内存，Fedora4 2.6.12 内核，Ext2 文件系统。

因为 SmallFS 采用可堆叠技术实现，所以首先需要测试加入可堆叠层后的性能开销。测试的方法是在加载空白可堆叠层的情况下，通过编译 Am-utils 的时间来比较有无可堆叠层时的性能。Am-utils 在编译过程中将进行大量的文件操作，是一个很好的测试工具。在 2 种情况下，各编译 10 次，统计每一次编译花费的时间并计算 10 次的平均值。结果如表 1 所示。

表 1　空白可堆叠层的性能开销测试

测试条件	Am-utils 编译时间/s
无可堆叠层	358.5
有可堆叠层	362.8

测试的第 2 步是测试各种典型的文件操作在加载了 SmallFS 与没有加载 SmallFS 时的性能开销，并进行比较。

首先测试的文件操作是文件创建操作，创建操作在一些应用场景下非常频繁，如缓存服务和网络监控。该测试项目分别测试当某一目录下已经包含 1 个、5 个、10 个、50 个、……、50 000 个、100 000 个文件时，在该目录下新创建 1 000 个新文件所消耗的时间。测试结果如图 3 所示。

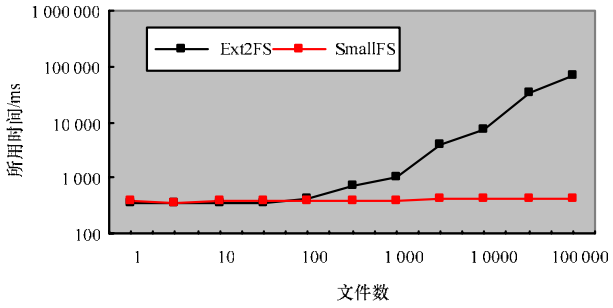


图 3　新建 1 000 个文件消耗的时间

从图 3 可以看出，在没有加载 SmallFS 时，当文件数量增大时所消耗时间也随之增大，时间增长幅度是线性的，这符合预期。而加载 SmallFS 后，对于文件数量变化所消耗的时间几乎是常数。从实现上来预测，当文件数更大时，子目录下的文件增多，在其中新建文件需要一些开销，这时 SmallFS 的消耗时间才会有小幅度增加，但是相对于没有加载 SmallFS 的性能开销来说，基本可以忽略。文件删除操作的测试结果基本与文件创建操作相同，在此不再给出。

下一个频繁的操作是对文件的检索操作，读写都要涉及到检索。该测试项目分别测试了当某一目录下已经包含 1 个、5 个、10 个、50 个、……、50 000 个、100 000 个文件时，随机地进行 1 000 次检索操作所消耗的时间(测试了 10 次，取其平均值)，测试结果如图 4 所示。

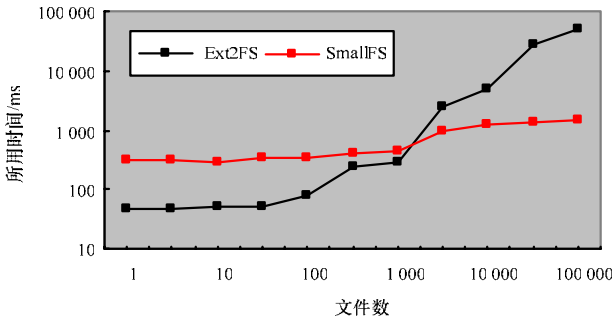


图 4　1 000 次检索操作消耗的时间

从图 4 可以看出，文件数目较少时，由于 SmallFS 需要进行多一层的检索，因此性能有一定的损失。但是随着文件数目的增加，未加载 SmallFS 时的检索时间线性上升，而加载 SmallFS 后性能损失很小。虽然测试中采用随机文件检索并不能完全符合实际的应用环境，但是在文件数目较多时，加载 SmallFS 无疑使检索性能有很大的提高。

5　结束语

本文提出的 SmallFS 采用可堆叠技术解决了文件系统中需要存储大量文件时所导致的性能下降问题。加载 SmallFS 后相比未加载 SmallFS 时各种文件操作性能均有很大提高，同时保证了可用性，此外有着较好的兼容性和可扩充性。

关于 SmallFS 的后续工作是继续优化代码，以进一步提高 SmallFS 的性能；利用可堆叠技术的特性，将 SmallFS 与安全文件系统、大文件优化文件系统等结合起来，提供应用面更广泛的功能。

参考文献

[1] Roselli D, Lorch J R, Anderson T E. A Comparison of File System Workloads[C]//Proceedings of the Annual USENIX Technical Conference. San Diego, California, USA: [s. n.], 2000: 41-54.

[2] Sweeney A, Doucette D. Scalability in the XFS File System[C]//Proceedings of the Annual USENIX Technical Conference. San Diego, California, USA: [s. n.], 1996: 1-14.

[3] Zadok E, Badulescu I. Usenetfs: A Stackable File System for Large Article Directories[R]. Computer Science Department, Columbia University, Technical Report: CUCS-022-98, 1998.

[4] Heidemann J S, Popek G J. Filesystem Development with Stackable Layers[J]. ACM Transactions on Computer Systems, 1994, 12(1): 58-89.

[5] Zadok E, Andersen J M, Badulescu I, et al. Fast Indexing: Support for Size-Changing Algorithms in Stackable File Systems[C]//Proceedings of the Annual USENIX Technical Conference. San Diego, California, USA: [s. n.], 2001: 289-304.

（上接第 43 页）

在{script}中插入多个目标字段规则表达式的集合，插入规则表达式前，需要转换变量代码，使不同字段正确对应到动态数组中。通过获取返回值中动态数组的不同数组变量就能得到经转换表达式转换后不同目标字段的值。

4　结束语

转换规则表达式的定义、编辑与执行有多种设计方案，采用本文方案能减少程序编写的复杂度和工作量，以适应较复杂的规则转换过程。本文方法提高了系统运行效率，加快了整个系统的开发进度。

参考文献

[1] 段永威, 秦　峰. 异构数据源数据转换工具的设计与实现[J]. 现代图书情报技术, 2004, (4): 59-62.

[2] Steve Teixeira Xavier Pacheco. Dephi5 开发人员指南[M]. 北京: 机械工业出版社, 2000.

[3] 姚领众. 一种基于中间库的数据库间数据转换技术[J]. 计算机系统应用, 1996, (11): 27-29.