一种基于对象存储的文件系统的设计*)

卢 萍 陈进才
(华中科技大学计算机学院 武汉 430074)

摘 要 随着存储技术的飞速发展,对象存储设备面临如何高效存储并管理TB级容量数据的问题。针对Linux通用文件系统Ext2在处理大文件和巨型目录结构时存在的局限性,采用B+树结构设计并实现了一种基于连续块的对象文件系统。对该文件系统进行的测试结果表明,该文件系统的读写效率较高,而且随着文件的增大,性能保持稳定。
关键词 对象存储,文件系统,B+树

Design and Implement of Object-based Storage File System

LU Ping CHEN Jin-cai
(College of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074, China)

Abstract Along with the rapid development of storage technologies, object-based storage device faces the problem of efficient terabit level storage. In allusion to the limitation of Ext2 file system in Linux for processing large files and huge directories, by adopting B+ tree structure, an object-based file system are designed and implemented, based on the assignment strategy of continuous blocks of data. The experiment results show that the implemented file system possesses a better data access efficiency than tranditinal file system, especially along with the capacity increasing of files, the performance of this file system can still be retained steadily.
Keywords Object-based storage, File system, B+tree

对象存储设备(Object Storage Device,OSD)是一种基于对象存储(Object-Based Storage,OBS)的智能化存储设备。OBS的基本思想是通过充分利用底层设备的处理能力将文件系统的某些功能直接集成在存储设备中,以便于存储设备自主地进行安全性管理、文件系统的底层处理和网络通信。与传统网络存储模式相比,OBS具有更好的可扩展性和自主性,从而也具有更好的系统综合性能。OBS面临的突出问题是:如何高效存储并管理TB级容量的数据。目前,一些成熟的OSD实现通常采用Linux的Ext2文件系统[1-3]进行存储管理,然而Ext2文件系统在文件的大小、目录结构的规模和性能上都有一定的局限性。本文从基本体系结构、分配策略和目录管理等角度分析了Ext2文件系统中存在的问题,通过采用B+树结构,设计并实现了一种基于连续块的OBS文件系统,使用测试工具对该系统进行了测试,并对测试结果进行了深入的分析。

1 Ext2文件系统的局限性

Ext2是基于"块"的文件系统。文件的基本组织单位是单个的数据块,组成文件的数据块通常并不是连续地分布在磁盘上,这样导致了大量的文件碎片,从而在对文件进行读写时,要多次进行磁盘寻道和扇区定位,降低了效率。磁盘空闲块的管理采用块位图结构,当系统要为文件分配磁盘空间时,使用线性搜索算法顺序扫描块位图,为文件分配一个合适的空闲块。由于块位图与数据块是一一对应的,随着磁盘容量的增大,块位图的大小也会线性增长,从而使得在进行顺序查找时,搜索时间增加,性能降低。Ext2的单块分配策略和线性搜索算法,使得创建一个大文件需要进行多次的I/O操作和空闲块的查找分配,效率很低。

在Ext2文件系统中,数据块的组织通过多级指针实现,文件的inode节点大小是固定的,最多只能包含12个直接块指针和3个间接指针。假设数据块大小为1024B,利用前12个直接指针可保存最大为12kB的文件,对这些文件的访问将非常迅速。如果文件超过了12kB,则利用一级间接指针,这一指针指向的数据块保存一组数据块指针,依次指向包含实际数据的数据块。由于每个数据块指针占用4个字节,则每个间接指针可包含256个数据块指针。这样,Ext2可保存的最大文件为17GB。由此可见,对于较小的文件,仅需利用高效的直接指针和效率稍低的一级间接指针;而对于大型文件,不得不使用二、三级间接指针,从而导致多次磁盘访问和线性查找,读写效率低且寻址空间有限。

目录是一种特殊的文件对象,是一组子目录入口的集合。在Ext2中,用目录inode来描述目录对象,inode中的子目录入口排成有序列表,目录入口由一个二元组(inode号,文件名)表述。文件定位时,系统解析文件名,找到第二级目录名,然后在根目录inode中,按照线性搜索算法以目录名为关键字遍历目录项列表,得到下一级目录inode号,然后读入下一级目录的inode节点,直到找到目标文件为止。对于包含成千上万个目录项的大型文件系统,线性搜索算法效率极低。

可见,Ext2文件系统不适合直接应用于OSD上,为提供海量存储支持,必须对文件组织结构进行改进。

2 OBS文件系统结构

2.1 连续块分配策略

在OBS文件系统中,文件的基本组织单位是一组连续的

*)本文研究工作得到国家973项目(编号:2004CB318201)的资助。卢 萍 副教授,研究方向为信息存储技术及系统;陈进才 副教授,研究方向为网络存储和嵌入式系统。

磁盘块，而不再是单个的磁盘块。连续块的定义用图 1 所示的连续块描述符结构(ext 结构)来表示。

Flags	OrgBlkNum	Len	Offset	Resvrd

图 1　连续块描述符结构

Flags 为标志位，用来分别表示写入时的复制、连续块分配记录、压缩信息等，OrgBlcNum 为起始块的块号，Len 为连续块长度，以块为单位，Offset 为逻辑偏移量，表示连续块在其所在文件中的相对位置，以块为单位表示，Resvrd 为保留字段。

连续块分配策略一次为文件分配多个连续的数据块，从而大大减少了查找和分配空闲块的次数[4,5]，提高 I/O 效率。在读操作时，只需读取起始块号和文件长度，就可以连续地读取数据块；在写操作时，有利于一次性写一大批数据到存储设备中，从而减少存储设备写数据的时间。

2.2　B+树索引技术

为使系统在海量存储时仍能保持快速的搜索速度，OBS 文件系统中用 B+树来组织连续块和管理目录。B+树是 B 树的一种变体，其定义为：一个阶为 m 的 B+树是树中每个内部节点都包含多达 m 个分支的树，内部节点含有 m 个子女，存储 $m-1$ 个关键字来引导检索，把每个关键字与一个指向子女节点的指针关联，叶节点存储实际记录。

用 B+树来组织连续块描述符时[6]，B+树的节点是连续块描述符的集合，B+树的根包含在该文件的 inode 中，以连续块在文件中的偏移量作为索引关键字。当对大文件进行操作时，首先通过 inode 节点找到代表该文件的 B+树的根，通过关键字来遍历该 B+树，迅速找到目标描述符，然后通过该描述符中的起始块物理块号得到实际数据块。使用 B+树的搜索时间复杂度仅为 O(log$n$)，与传统的多级指针结构相比，该结构检索速度显著加快。

用 B+树来管理目录时，目录 B+树中的节点是目录项的集合，以文件名作为索引关键字。每一个目录都含有一个由它的子目录项构成的目录 B+树，树的根节点包含在该目录的 inode 节点中。按照这种结构组织，目录 B+树能迅速地检索到目标文件的入口。

B+树结构也增加了系统的寻址空间。本文定义 B+树是一个阶为 256，最大树高为 8 的平衡树，根节点最多有 8 个子女。那么一个文件最多能容纳 $8*256^{(8-1)}$ 个连续块，一个连续块最少由一个数据块组成，大小为 1024 字节，因此，最大寻址至少为 $8*256^{(8-1)}*1024=2^{69}$ 字节，大于 64 位机的 2^{64} 字节寻址空间。对于小文件，文件数据直接内嵌在文件 inode 节点中，一旦找到该文件的 inode，就能直接获得该文件的实际内容，不需要再一次 I/O 操作。而且，由于无需单独占用一个数据块，从而减少了内部碎片的产生，节省了磁盘空间。

2.3　OBS 文件系统结构设计

OBS 文件系统主要由空闲块管理器、索引结点(inode)管理器、文件管理器、目录管理器以及虚拟文件系统(VFS)接口和各类缓存组成。

VFS 接口：实现文件系统的虚拟接口，主要负责文件系统的安装和日志环境的建立。

文件管理器：对文件的 B+树组织结构进行维护和管理。负责文件的新建、寻址等。

空闲块管理器：维护整个文件系统磁盘空间的分配和回收。空闲块管理器将磁盘中的空闲连续块按照起始物理号构造成一棵 B+树，通过索引机制灵活有效地对空闲数据块进行分配，回收时合并相邻的空闲块。

目录管理器：对每个目录对象的 B+树组织进行维护和管理，负责目录的新建、插入、删除等操作。

inode 结点管理器：负责文件对象 inode 节点的分配和回收。沿用 Ext2 中的静态分配方式，索引节点按照节点号顺序存储在索引节点表中，通过索引节点位图跟踪各索引节点的使用情况，并进行分配和回收管理。

缓冲区高速缓存：为了减少对物理设备的访问次数，提高访问效率，系统维护了 3 个高速缓存，分别用来存放经常访问的文件数据块、索引节点和目录项。用哈希表对缓存进行管理。当需要访问某个数据块时，先在缓存中搜索，如果命中，访问计数加 1，直接对其进行操作；否则，数据块尚未调入缓冲区，系统必须进行磁盘 I/O 操作，将数据块调入缓存，同时分配一个空缓冲区。

3　OBS 文件系统的实现

3.1　存储逻辑结构

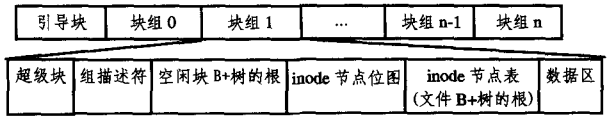

图 2　磁盘布局的逻辑映射图

OBS 文件系统的整体逻辑结构与 Ext2 类似，如图 2 所示。磁盘被分割成若干块组，每个块组由超级块、组描述符表、空闲块 B+树的根、inode 节点位图、inode 节点表和数据区六部分组成，与 Ext2 不同的是，块组的第三部分存放的不是空闲块位图结构，而是空闲块 B+树的根节点，因为空闲块是由 B+树动态管理的。超级块和组描述符分别描述整个文件系统和各块组的管理信息。inode 节点位图描述 inode 节点表中的索引节点的使用情况，inode 节点表中按节点号存放 inode 节点，但是 inode 节点的结构发生了变化，除了描述文件对象的 posix 属性外，还包含 B+树的根，inode 节点分配器通过 inode 节点位图和 inode 节点表为文件对象分配 inode 节点，inode 节点的分配方法与 Ext2 相同。数据区中存放空闲块和数据块，还有组成空闲块 B+树，文件 B+树和目录 B+树的叶子节点和内部节点。

3.2　文件管理器

与 Ext2 文件系统一样，OBS 文件系统仍采用 inode 节点来描述文件对象，但数据结构发生了变化。

一个文件系统不仅包含大文件，还有中小文件，对于这些文件的数据块如果也按 B+树的方式进行组织，不仅不能带来性能上的提升，相反还会增加系统的复杂性。考虑到 Ext2 在处理中小文件时效率较高，将 inode 结构设计成如图 3 所示形式。对于小文件(如符号链接)，将其数据内容直接保存在 inode 的后半部分中；对于中型文件(由 1 至 8 个连续块组成的文件)，其所在的数据块由连续块描述符 1 至 8(类似于原来的一级指针数组)给出；而对于大文件(由大于 8 个连续块组成)，用 B+树来组织其连续块的描述符 ext。B+树的根包含在该文件的 inode 节点中，B+树由 ext 中的偏移量作为索引。

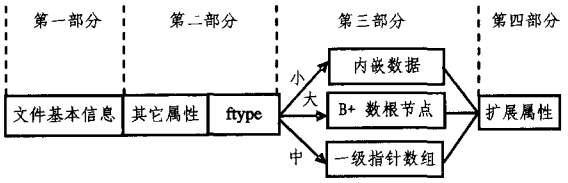

图 3　inode 节点结构

inode 节点大小是 512 字节，包含 4 部分信息。第一部分(128 字节)描述文件的基本信息：包括模式、访问权限、所有者信息、时间戳、文件长度和连接数等。第二部分(128 字节)描述对象的其它属性：支持 VFS 必需的信息、操作系统环境特定的信息。此部分最后有一个文件描述符 ftype，它描述在 inode 节点的后半部分中存储的内容。对于小文件，其数据可以直接存放在 inode 的后半部分的 256 字节中，此时 ftype 存放的是内嵌数据的描述信息，如大小和起始块号等；如果文件数据量较大或者不适合 inode 的内嵌数据空间，它将包含在连续块中，同一文件的连续块描述符组成一个 B+树，此时 ftype 中存放的是这棵 B+树根节点的头部信息。第三部分(128 字节)内嵌数据或者是代表 B+树根节点的连续块描述符 ext 数组。一个 ext 大小为 16 个字节，因此一共可以存放 8 个 ext。如果文件有 8 个或少于 8 个连续块，那么这些 ext 结构直接指向连续块(类似原来的直接块指针)。否则，作为 B+树的根，指向 B+树的叶节点或内部节点。第四部分包含扩展属性、更多内嵌数据或附加的连续块分配描述符。一旦 inode 中的 8 个 ext 结构均已填充，或内嵌数据大小超过了 128 字节就使用 inode 的最后 1/4。

B+树分为 3 层，根节点、内部节点和叶子节点。节点由头部和一组项目构成，项目就是一个 ext 结构，节点中的项目按 ext 结构中的偏移量字段排序，如图 4 所示。

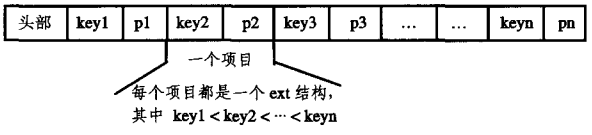

图 4　文件 B+树节点结构

空闲块 B+树的组织和文件 B+树完全相同，唯一的区别是组织顺序不一样，文件 B+树的索引关键字是连续块在文件中的偏移，而空闲块 B+树的关键字是连续块的物理地址。因此，空闲 B+树可以看作是对文件 B+树的复用，它直接使用文件 B+树中定义的叶子节点和内部节点结构。而且对 B+树的操作，也复用文件 B+树中的相关算法，只是关键字用的是物理地址而不是逻辑偏移。

3.3　目录管理器

OBS 文件系统提供了两种不同的目录组织方式：第一种用于小目录(目录项<8)，将目录项的内容直接存储到目录的 inode 节点中，而不用为其分配一块存储区域，这样减少了内部碎片，而且访问 inode 的同时可以访问到数据，不需要再进行一次 I/O 操作；第二种用于较大的目录，将目录表示为以名字为关键字的 B+树，和传统的非排序目录组织方式相比，这种方式提供了更快速的目录查找、插入和删除能力。

目录 B+树也有根节点、内部节点和叶子节点 3 种层次。根节点用 inode 节点来描述目录对象，目录 B+树的根包含在 inode 节点中。目录 inode 节点与文件 inode 节点的结构相同，也分为 4 个部分。与之对应的，在 inode 的第二部分存放 B+树根的头部，如果是按照第一种方式组织的小目录，inode 的第三部分直接用来存放目录项入口，最多可以存放 8 项；如果是按 B+树组织，第三部分存放的是 B+树的根。

无论是根节点、内部节点还是叶子节点，主体部分都由一个目录槽组成。初始化时，目录中的内容为空，称为空闲槽。目录建立过程中，空闲目录槽被分配用来存放目录 B+树的相关信息，这样的槽称为活动槽；对于叶子节点，目录槽中存放的是目录项，即子目录的 inode 号和名字；对于内部节点而言，目录槽中存放的是用于索引的名字关键字和下级节点的物理位置，称之为路由项。节点中的空闲槽和活动槽分别用空闲槽链表和活动槽数组管理和维护。

空闲槽大小为 32 字节，空闲槽链接成一个表。要插入一个目录/路由项时，从空闲槽链表中分配一个槽，填充目录/路由项内容，并将该槽号按序插入到活动槽数组中。

活动槽数组是活动目录槽号的有序数组。活动槽数组的元素对应一个活动目录槽的槽号，数组的元素按照它所指代的目录项(路由项)的名字关键字排序。使用该数组减少了目录项增删时所需要移动的次数。数组中每个元素只占用 1 位(整个活动槽大小为 128 位，占用 128/32=4 个槽)。活动槽数组放在整个目录槽数组的起始部分，目录槽增删时，只需移动数组而不用移动项。在数组中，可用二分法搜索某个目录项。

4　性能测试与分析

以配置有 Pentium4 1.7GHz 处理器、512MRAM 内存、80G IDE 硬盘和 Linux 单内核版本 2.4.6 的计算机为实验环境，以 Bonnie++和 Postmark 为测试工具，分别用 Ext2 和 OBS 文件系统进行了文件读写性能的测试和文件创建及删除等 I/O 性能的测试。

4.1　文件读写性能

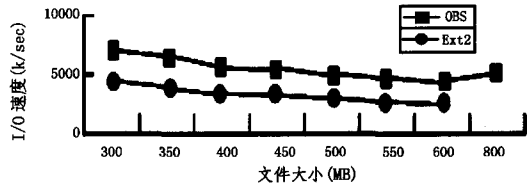

图 5　写操作 I/O 速度比较

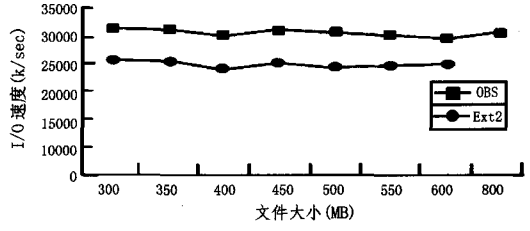

图 6　读操作 I/O 速度比较

分别用 OBS 文件系统和 Ext2 文件系统对 300MB 至 800MB 的文件执行写和读操作，测试结果如图 5 和图 6 所示。从图中可以看出，OBS 写速度是 Ext2 的 1.5 倍左右，OBS 读速度要比 Ext2 快 5000kb/s 左右，而且随着文件的增大，速度保持稳定。这表明 OBS 文件系统的 I/O 速度更快，而且 Ext2 文件系统不支持大于 800M 文件的测试，而 OBS

(下转第 172 页)

```
@1 namespace_decl name: @2  srcp: <internal>:0 dcls: @3
@2 identifier_node strg: :: lngt: 2
@3 function_decl   name: @4  type: @5  srcp: ex1.c:3
@4 identifier_node  strg: main     lngt: 4
@5 function_type    size: @8     algn: 64     retn: @9
@6 function_decl    name: @11  type: @12    chan: @13
@7 compound_stmt    line: 8    body: @14    next: @15
@8 integer_cst      type: @16  low : 64
@9 integer_type     name: @17  size: @18   algn: 32
@10     tree_list         valu: @21
```

图 3　消除冗余前的抽象语法树文本

```
@3   function_decl     name: @4   type: @5   extern          body: @7
@4   identifier_node   strg: main     lngt: 4
@5   function_type  size:@8   algn:64   retn: @9    prms: @10
@7   compound_stmt     line: 8          body: @14       next: @15
@8   integer_cst       type: @16        low : 64
@9   integer_type      name: @17        size: @18       algn: 32
@10   tree_list        valu: @21
```

图 4　冗余消除后的抽象语法树文本

结束语　实验表明，本文算法达到了很好的效果并且具有较低的时间和空间复杂度。作为改善评分系统前端的重要一步，现已用到评分系统中，取得了良好的效果。与传统的没有冗余消除解析方法相比，消除冗余后的抽象语法树具有更好的实用性。再进行进一步处理，可以非常方便地应用于对程序分析及其它领域。

参 考 文 献

[1] 王甜甜. 基于语义相似度的编程题自动评分系统研究与应用. 硕士论文. 哈尔滨工业大学，2005：1-53

[2] 石峰，等. 一种解析 GCC 抽象语法树的方法[J]. 计算机应用，2004，24(3)：115-116

[3] 苏小红，陈惠鹏，孙志岗，等. C 语言大学实用教程[M]. 电子工业出版社，2004，8：231-374

[4] 刘文伟，等. 一个重建 GCC 抽象语法树的方法[J]. 计算机工程与应用，2004，8：125-128

[5] Power J F. Program annotation in XML：a parse-tree based approach[C]// Proceedings of the Ninth Working Conference on Reverse Engineering. 2002：1095-1350

[6] 王相懂，等. 基于 gcc 抽象语法树对 c++源程序结构的分析[J]. 计算机工程与应用，2006(6)：97-100

[7] Antoniol G. XML-Oriented gcc AST Analysis and Transformations[C]// Proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation. 2005，7：869-901

(上接第 133 页)

可以测试大于 1G 的文件，说明 OBS 在大文件支持方面的能力有所提高。

4.2　文件操作性能

先创建两个文件集 a 和 b，每个文件集共有 32768 个文件，10 个目录。文件集 a 的文件大小在 80B 到 120B。而文件集 b 的文件大小在 800B 到 1200B，测试创建、读和删除的 I/O 性能，结果如表 1 和表 2 示。从表中可以看出，OBS 在创建、读和删除大文件方面的性能比 Ext2 要强得多，而且，随着文件大小的增加，Ext2 的速度下降得很快，特别是顺序读操作，而 OBS 的速度则保持稳定，几乎没有太大的起伏。从 CPU 的占用率来看，随着文件系统容量的增加，OBS 的 CPU 占用率有所升高，这是因为使用了复杂的搜索算法和数据结构。

表 1　文件集 a 的文件操作性能

文件系统	文件数 最大尺寸 最小尺寸 目录数	顺序创建		读		删除		随机创建		读		删除	
		k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu
Ext2	32：120：80：10	7132	69%	10102	23%	32252	30%	7288	69%	none	none	9887	66%
OBS	32：120：80：10	6339	77%	12647	20%	8386	60%	6730	82%	558	1%	336	2%

表 2　文件集 b 的文件操作性能

文件系统	文件数 最大尺寸 最小尺寸 目录数	顺序创建		读		删除		随机创建		读		删除	
		k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu
OBS	32：1200：800：10	7647	74%	18631	25%	26135	27%	7670	72%	none	none	8939	55%
Ext2	32：1200：800：10	5045	66%	2225	7%	2386	19%	4242	57%	255	1%	171	1%

参 考 文 献

[1] Hitz D，Lau J，Malcolm M. File System Design for an NFS File Server Appliance// Anon，ed. Proceedings of USENIX 1994. San Francisco：Network Appliance，1994：103-105

[2] Schmuck F，Haskin R. GPFS：A Shared-disk File System for Large Computing Clusters// Darrell D E，ed. Proceedings of the Conference on File and Storage Technologies. Monterey：FAST02，2002：231-244

[3] Scltzer M. Analysis of extent allocation policies in file systems. Masters Report. EECS Dept.，University of California，Berkeley，Ca，1988：1-73

[4] Berchtold S，Keim D A，Kriegel H P. The X-Tree：An index structure for high-dimensional data// Anon，ed. Proceedings of the 22th International Conference on Very Large Data Bases India. 1996：28-39

[5] Johson T，Shasha D. The performance of concurrent B-tree algorithms. ACM Transaction on Database Systemns，1993，18(1)：51-101

[6] Bryant R，Forester S R，Hawkes J. Filesystem performance and scalability in linux 2. 4. 17// Anon，ed. Proceedings of the USENIX Annual Technical Confernce. CA：USENIX Association，2002：235-245