蓝鲸机群文件系统文件 layout 设计与实现*)

张军伟[1,2] 郭明阳[1,2] 张建刚[1] 许 鲁[1]
(中国科学院计算技术研究所 北京 100080)[1] (中国科学院研究生院 北京 100039)[2]

摘 要 维护文件逻辑位置到物理位置映射关系的文件布局是文件系统的重要组成部分。针对蓝鲸海量机群文件系统中文件的条带分布特性,分析了目前普遍采用的三级间接地址文件布局和扩展块段文件布局的优缺点。有效结合三级间接地址文件布局避免查询、直接映射的优点和扩展块段文件布局映射高效、空间占用少的优点,提出了一种新的固定长度扩展块段文件布局,并在小文件环境对其进行了优化改进,提出了幂次长度扩展块段文件布局。实验测试结果表明该文件布局可以有效适用于蓝鲸机群文件系统,达到了预期效果。该文件布局同样适用于其他采用条带机制的文件系统。

关键词 BWFS,机群文件系统,文件布局,条带

Design and Implementation of File Layout in Blue Whale Cluster File System

ZHANG Jun-wei[1,2] GUO Ming-yang[1,2] ZHANG Jian-gang[1] XU Lu[1]
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China)[1]
(Graduate School of Chinese Academy of Sciences, Beijing 100039, China)[2]

Abstract File layout, which manages the mapping relationship between logical offset of file and physical address of storage device, is important for a file system. The commonly used triple-indirect and extent-based layouts are not fit for cluster file systems that use stripe to improve performance, such like BWFS. A novel fixed-length extent-based layout was presented and improved as power-length extent-based layout especially for small-size files. The presented layout effectively absorbs the direct-mapping advantage of triple-indirect layout and efficient mapping, less layout space waste advantages of extent-based layout. Test results show that the presented layout is efficiently fit for BWFS. The layout can also be used in other file systems which use stripe technology.

Keywords BWFS, Cluster file system, File layout, Stripe

1 引言

文件布局(layout)是文件系统的重要部分,维护着文件逻辑位置到物理位置的映射关系。目前普遍采用的主要有两种 layout 布局:一种是三级间接地址(triple-indirect)布局,另一种是扩展块段(extent-based)布局。

三级间接地址布局[1]中,每一个文件逻辑位置(块号)同物理位置(块号)的映射关系都记录在文件 layout 结构中,并且记录位置根据映射算法固定确定。其地址映射过程如下:首先根据映射算法直接确定映射关系记录在 layout 结构中的位置,然后得到各级间接地址,逐级解析,最终获取目标物理位置。该布局在 ext2/ext3 等本地文件系统中使用,但不适用于海量文件系统。海量文件系统中资源通常达到 TB 甚至 PB 级别,文件的资源连续性较好,但三级间接地址布局把每个逻辑位置的映射关系都记录在文件 layout 中,因此(1)导致 layout 空间的占用较多,造成资源浪费;(2)由于 layout 空间占用较多,造成 layout 相关操作性能下降;(3)每次地址映射都需要三级间接地址的逐级遍历,造成了性能损失。

海量文件系统大多采用扩展块段文件布局[2,3]。该布局在文件 layout 中记录 extent 三元组(offset,start,len),表示从文件逻辑位置 offset 起始长度为 len 的逻辑空间,映射到物理位置 start 起始,长度为 len 的物理空间。扩展块段布局有效解决了三级间接地址布局存在的若干问题,但其也存在一些问题:(1)地址映射需要查询映射关系在文件 layout 中的记录位置,而不能采用直接计算方式,造成性能损失。一般采用 B+树组织 extent 三元组,优化查询性能。(2)在文件资源较碎,extent 三元组数量较多情况下,该方式不能达到好的效果。

蓝鲸机群文件系统[4,5](Blue Whale File System, BWFS)支持 PB 级容量,采用文件条带(stripe)机制,把文件数据分布在多个存储设备,实现多个存储设备的并行访问,提高文件访问性能。在 BWFS 中,文件的物理资源在整体上是不连续的,而是分布在多个存储设备,因此扩展块段文件布局不能很好适用于 BWFS。本文针对采用 stripe 机制的文件系统提出了固定长度扩展块段(fixed-len-extent-based)文件布局,结合了三级间接地址和扩展块段文件布局的优点,通过计算直接确定映射关系记录在文件 layout 中位置,并降低文件 layout 空间占用,避免每次地址映射都逐级遍历,提高了地址映射性能。另外,针对固定长度扩展块段文件布局在小文件情况下空间浪费问题,对固定长度扩展块段进行了优化,提出了幂次长度扩展块段(power-len-extent-based)文件布局。

本文第 2 节详细描述了 BWFS 文件系统中固定长度扩

*)基金项目:本课题得到国家"973"计划(2004CB318205)和中科院计算所创新项目(20056420)资助。**张军伟** 博士生,主研方向为网络存储、集群文件系统;**郭明阳** 博士生,主研方向为机群文件系统;**张建刚** 副研,博士;**许 鲁** 研究员,博导。

展块段文件布局的设计与实现；第 3 节针对小文件环境，对固定长度扩展块段文件布局进行优化，提出了幂次长度扩展块段文件布局，并详细描述了其设计与实现；第 4 节介绍了理论分析和对比实验测试；最后，对本文工作进行了总结。

2 固定长度扩展块段文件布局的设计与实现

固定长度扩展块段文件布局中，文件 layout 仍以 extent 方式组织，但 extent 的长度固定，因此每个文件仅需要记录一次 extent 长度。地址映射过程中，通过计算直接获取映射关系在 layout 的记录位置，从而得到目标物理位置，避免了扩展块段的查询操作。由于映射关系记录位置固定，因此 extent 三元组(offset，start，len)结构中仅需要记录(start)，而不必记录 offset 和 len，节约了 layout 空间。

2.1 相关数据结构设计

inode 数据结构中增加 ext_len 域，记录文件的 extent 长度，仅取 2 的幂次数据块数量，并且 ext_len 仅记录指数。

文件 layout 采用与三级间接地址相同的结构，但文件 layout 中仅记录 extent 的起始物理位置，extent 的长度则通过 inode ->ext_len 计算获得。

2.2 layout 地址映射流程

为文件逻辑位置 file_offset 进行地址映射，首先根据 inode ->ext_len，计算 file_offset 所在的逻辑 extent 序号，依照该逻辑序号通过地址映射算法获取逻辑 extent 在 layout 中的记录位置，得到 extent 起始物理位置，然后根据 file_offset 进行计算，获取对应的物理位置。具体算法如图 1 所示。

```
offset_trim = file_offset >> inode ->ext_len;  // 获取 file_offset 所在逻辑 extent 序号
ind_ptr = get_layout(offset_trim);  // 获取 extent 在 layout 中的记录位置
start = *ind_ptr;  // 获取 extent 起始物理位置
blokno = start + file_offset & (( 1<< inode ->ext_len) -1);
// 获取同 file_offset 对应的物理位置
```

图 1 固定长度扩展块段 layout 地址映射流程

2.3 layout 的创建流程

写文件 file_offset 位置时，首先进行地址映射。如果获取到相应的物理位置，表示物理资源已经被分配，layout 已经创建；否则需要进行资源分配和 layout 的创建操作。根据 inode ->ext_len 确定分配资源长度：alloc_len = 2^inode ->ext_len，从资源池中分配相应长度的连续物理资源(start，alloc_len)，仅把起始物理位置 start 记录到 layout 中即可。

2.4 layout 的截短(truncate)流程

当删除文件或截短文件时，需要把文件截短部分的资源进行释放。根据要截短后文件大小 i_size，确定 i_size 所在的逻辑 extent 序号，该 extent 是需要保留的，需要从下一个 extent 开始进行资源释放，释放每个 extent 包含的资源。具体的算法如图 2 所示。

```
if(inode ->i_size == 0)
    trunc_start = 0;  // 文件被清空情况
else
    trunc_start = (inode ->i_size >> inode ->ext_len) + 1;
// 获取起始释放 extent 序号
遍历释放从 truncate_start 开始的 layout 中所有 extent 资源；
释放每一个 extent 的资源(start, 1<< inode ->ext_len);
```

图 2 固定长度扩展块段 layout 的 truncate 流程

3 幂次长度扩展块段文件布局的设计与实现

固定长度扩展块段文件布局还存在若干问题，当系统中存在大量小文件时，会由于 extent 长度较大而造成空间浪费。在实际系统测试中，采用 256 块 extent 长度，拷贝总容量 115MB 的 linux 源代码到 BWFS 文件系统后，占用的空间高达约 7GB，造成约 60 多倍的空间浪费，如此高的空间浪费是不能接受的。

一种解决方式是，针对小文件设置较小的 extent 长度，则小文件的空间浪费问题可以得到解决，但由于无法准确判断哪些是小文件，哪些是大文件，对于大文件如果 extent 长度设置较小，就失去了采用 extent 的意义。

因此本文提出幂次长度扩展块段文件布局，extent 长度随文件大小变化动态调整。当文件较小时，采用较小的 extent 长度，随着文件的增大，逐渐采用较大的 extent 长度，直到 extent 长度增大到一个最大值，以同时高效满足小文件和大文件的需求。

3.1 相关数据结构设计

在 inode 结构中增加 ext_len_low，ext_len_high 域，其中 ext_len_low 记录最小 extent 长度的指数，ext_len_high 记录最大 extent 长度指数。对于小文件采用 ext_len_low 的 extent 长度，随着文件的增长，以每次扩大一倍的方式增加 extent 的长度，直到 extent 的长度增加到 ext_len_high 之后，extent 长度保持 ext_len_high 不变。

幂次长度扩展块段同固定长度扩展块段文件布局结构对比示意如图 3 所示，以 ext_len_low=1，ext_len_high=5，ext_len=5 为例。

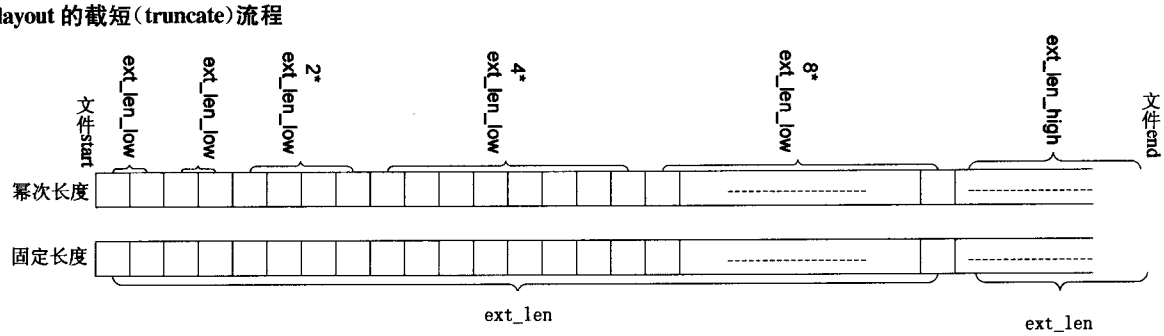

图 3 文件 layout 结构的对比

从文件逻辑位置 0 开始顺序映射，第一个 extent 长度为 2^ext_len_low；第二个 extent 长度也为 2^ext_len_low；第三个 extent 长度为 2 * 2^ext_len_low，也即 2^(ext_len_low+1)；第四个 extent 长度为 4 * 2^ext_len_low，也即 2^(ext_len_low+

2)；直到 extent 长度为 2^ext_len_high 为止。分析可知，到 2^ext_len_high 为止，已经分配的数量为：2^ext_len_low+ 2^ext_len_low+2^(ext_len_low +1) + …… +2^(ext_len_high−1) = 2^ext_len_high，因此，可以理解为把固定长度的第一个 extent 进行了细化分解，从而避免了小文件情况下的资源浪费。

3.2 layout 地址映射流程

为文件逻辑位置 file_offset 进行地址映射，首先根据 inode ->ext_len_low 和 inode ->ext_len_high，计算 file_offset 所在的逻辑 extent 序号和 extent 长度，依照该逻辑序号通过地址映射算法获取逻辑 extent 在 layout 中的记录位置，得到 extent 起始物理位置，然后根据 file_offset 进行计算获取对应的物理位置。具体映射算法如图 4 所示。

```
if(file_offset >= 2^inode ->ext_len_high){  // file_offset 超过最大 extent 长度
    offset_trim = file_offset >> inode ->ext_len_high;  //按照最大长度进行对齐
    offset_trim += inode ->ext_len_high -inode ->ext_len_low;
  //需要补充低于最大长度 extent 的数量，确定逻辑 extent 序号；
    offset_ext_len = inode ->ext_len_high;  //记录 extent 长度；
    }else{// file_offset 低于最大 extent 长度
    for(i= inode ->ext_len_high−1; i>= inode ->ext_len_low; i−−){
        if(file_offset & (1<< i)) break;//确定 file_offset 所在 extent 的长度为 2^i
    }
      offset_trim = i −inode ->ext_len_low + 1; //根据 extent 长度 2^i 计算逻辑 extent 序号。
      offset_ext_len = (i >= inode ->ext_len_low)? i : inode ->ext_len_low; //记录 extent 长度，如果 i 低于最小 extent 长度，则按照最小 extent 长度。
}
ind_ptr = get_layout(offset_trim);  // 获取 offset_trim 所在的 layout 中的对应位置
start = *ind_ptr;  // 获取对应 extent 的起始物理位置
blokno = start + file_offset & (( 1<< offset_ext_len) −1);
// 获取同 file_offset 对应的物理位置
```

图 4 幂次长度扩展块段 layout 地址映射

3.3 layout 创建流程

在写文件 file_offset 位置时，如果地址映射不存在，则需要进行 layout 的创建。根据地址映射过程中获取的 offset_trim 和 offset_ext_len，分配物理资源（start，2^offset_ext_len），仅记录 start 到 layout 的 offset_trim 位置处。

3.4 优化后 layout 的 truncate

在 truncate 操作中，根据要截短后文件大小 i_size，按照地址映射算法，确定 i_size 所在逻辑 extent，然后从下一个 extent 开始进行资源释放，释放每个 extent 包含的资源，但每个 extent 包含的资源数量可能不同。具体的算法如图 5 所示。

```
if(inode ->i_size == 0)
    offset_trunc = 0; // 表示所有的 extent 都需要释放
else
    offset_trunc 为 inode ->i_size−1 的所在逻辑 extent 序号 + 1;
// 获取需要 trunc 的 layout 开始 extent_start。
    记录要 trunc 的 extent 的起始长度 offset_ext_len。
    遍历释放从 extent_start 开始的 layout 中所有 extent 资源（start,2^offset_ext_len），在释放过程中调整 offset_ext_len 的值，调整为下一个 extent 的长度，从而正确释放每个 extent 的资源；
```

图 5 幂次长度扩展块段 layout 的 truncate

4 分析与测试

4.1 对小文件资源浪费问题的分析和测试

采用幂次长度扩展块段文件布局设计后，小文件的空间浪费问题得到了有效的解决。

定义 文件空间浪费率为：$p=(\frac{file\_blocks * block\_size}{file\_size}-1)*100\%$，该定义不考虑文件空洞情况。

设 $f(x)=2^x * block\_size$，则幂次长度扩展块段文件布局的空间浪费率为：

$$p=\begin{cases}(\frac{f(ext\_len\_low)}{file\_size}-1)*100\%, file\_size\leqslant f(ext\_len\_low);\\(\frac{f(ext\_len\_low+1)}{file\_size}-1)*100\%, f(ext\_len\_low)<file\_size\leqslant f(ext\_len\_low+1);\\(\frac{f(ext\_len\_low+2)}{file\_size}-1)*100\%, f(ext\_len\_low+1)<file\_size\leqslant f(ext\_len\_low+2);\\\cdots\cdots\\(\frac{f(ext\_len\_high)}{file\_size}-1)*100\%, f(ext\_len\_high-1)<file\_size\leqslant f(ext\_len\_high);\\(\frac{2f(ext\_len\_high)}{file\_size}-1)*100\%, f(ext\_len\_high)<file\_size\leqslant 2f(ext\_len\_high);\\\cdots\cdots\\(\frac{nf(ext\_len\_high)}{file\_size}-1)*100\%, (n-1)f(ext\_len\_high)<file\_size\leqslant nf(ext\_len\_high);\end{cases}$$

而固定长度扩展块段文件布局的空间浪费率为：

$p=(\frac{nf(ext\_len\_high)}{file\_size}-1)*100\%, (n-1)f(ext\_len\_high)<file\_size\leqslant nf(ext\_len\_high), n\geqslant 1$

文件空间浪费率随文件大小变化的曲线如图 6 所示，以 ext_len=8，ext_len_low=0，ext_len_high=8 为例，为增大显示度，file_size 仅以块大小 4k 为单位。

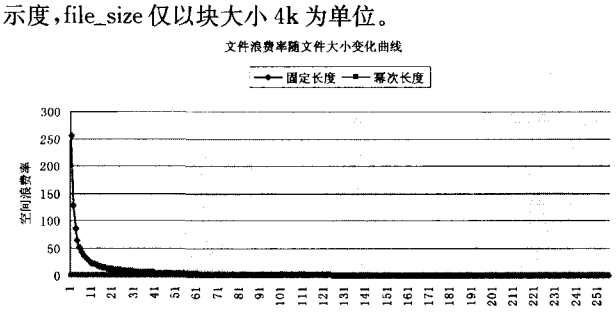

图 6 文件浪费率随文件大小变化曲线

可见在小文件情况下，固定长度扩展块段文件布局的文件浪费率严重，对于一块大小的文件空间浪费达到 255 倍，对于 2 块大小的文件空间浪费率达到 127 倍，直到对于接近 128 块大小的文件空间浪费为 1 倍。幂次长度扩展块段文件布局有效解决了小文件空间浪费的问题，在 ext_len_low 取

0，ext_len_high=8 时，对于一块大小的文件空间浪费为 1 倍，对于 2 块大小的空间浪费为 0.5 倍，对于 3～4 块的空间浪费率为 0.33 左右。直到文件大小超过 2^(ext_len－1)后，幂次长度和固定长度的 extent 映射格式空间浪费率相同，并都随着文件的增大而减小。

进行实际测试，拷贝总容量 115MB 的 linux-2.4.18-14 内核源代码到 BWFS 文件系统，固定长度扩展块段文件布局占用空间为：7546884kB，空间浪费为：63.09 倍；幂次长度扩展块段文件布局占用空间为：176756kB，空间浪费为 0.50 倍，达到了较好的效果。

幂次长度 extent 映射格式可以支持的最大文件容量约为：max_filesize = (1024^3) * (2^ext_len_high) * block_size；当 ext_len_high 取值 8，block_size 取值 4k 时，max_filesize 约为 1PB，这将在很长一段时间内都能够满足需求。

4.2 文件 layout 空间占用情况分析

相对于三级间接地址文件布局，由于幂次长度扩展块段文件布局每个间接地址代表一个特定长度的 extent，因此大大降低了 layout 的空间占用，layout 空间占用约为三级间接地址的 1/(2^inode ->ext_len_high)。以 inode ->ext_len_high 取 8 时为例，layout 空间占用约为三级间接地址 1/256。

相对于扩展块段文件布局，由于幂次长度扩展块段文件布局仅记录起始物理位置，不必记录文件逻辑位置和长度，因此也减少了 layout 的空间占用。以逻辑位置用 32 位变量，物理位置用 32 位变量，长度用 16 位变量表示为例，layout 空间占用约为 extent 映射方式 32/(32+32+16)=2/5。

4.3 性能对比测试

在 BWFS 文件系统中，对三级间接地址文件布局、扩展块段文件布局和幂次长度扩展块段文件布局进行了实现和性能对比测试。

在三级间接地址文件布局实现中，进行了优化。每次资源分配获取长度为固定 len 的连续块段而不是仅获取一块，每块的映射关系都记录到文件布局中。在地址映射过程中，三级间接地址文件布局把文件逻辑地址根据 len 对齐，仅对对齐后的逻辑地址进行地址映射。

在扩展块段文件布局的实现中，采用了 B+树组织 extent 三元组。

在测试中，为了突出表现文件 layout 的操作性能，提供直接操作文件布局的 ioctl 接口，屏蔽文件数据相关的读写访问操作。取三级间接地址文件布局每次资源分配的长度 len 为 2^8 个块、扩展块段文件布局每个 extent 的长度固定为 2^8 个块并且 extent 不能合并、幂次长度 extent 文件布局的 ext_len_low=ext_len_high =8 为例进行测试，分别测试在同一目录下创建、读取、删除 10 个 8GB 文件的性能。在 Intel Celeron 1GHz，内存 256MB，存储设备容量为 120GB 的节点上进行测试，测试结果如图 7 所示。

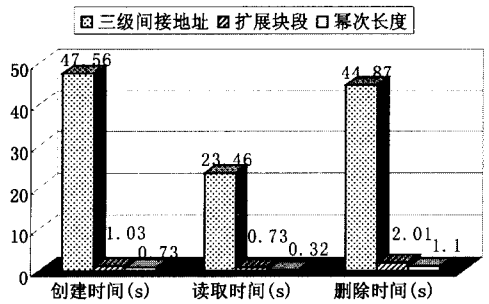

图 7 性能对比测试结果图

从图 7 可见，幂次长度扩展块段文件布局和扩展块段文件布局的操作对比三级间接地址文件布局都有较大的性能提高，幂次长度扩展块段文件布局相应操作对比扩展块段文件布局也有性能提高，这主要是因为幂次长度扩展块段的文件布局中仅记录块段首，减少了文件布局空间占用，减少了操作所涉及到的 layout 空间，提高了操作性能。

结束语 本文针对蓝鲸机群文件系统中文件资源采用 stripe 分布的特点，分析了目前普遍采用的三级间接地址的文件布局和扩展块段文件布局，并指出以上文件布局不适用于 BWFS 等采用 stripe 机制以实现并行访问、提高访问性能的海量机群文件系统。因此结合三级间接地址和扩展块段文件布局的优点，提出了固定长度 extent 的文件布局，并针对小文件情况，对该文件布局进行了优化，提出了幂次长度扩展块段文件布局。实验表明，幂次长度扩展块段文件布局能够提高文件 layout 操作性能，降低文件 layout 的空间占用。

参 考 文 献

[1] Theodore Y T, Tweedie S. Planned Extensions to the Linux Ext2/Ext3 Filesystem // Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference. Berkelsy, 2002: 235-243

[2] Mathur A, Cao M, Bhattacharya S. The new ext4 filesystem: current status and future plans // Proceedings of the Linux Symposium. Ottawa, Ontario, Canada, 2007, 6: 21-33

[3] Bryant R, Forester R, Hawkes J. Filesystem Performance and Scalabiliby in Linux 2.4.17 // Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference. Berkelsy, 2002: 259-274

[4] 黄华，张建刚，许鲁. 蓝鲸分布式文件系统的分布式分层资源管理模型. 计算机研究与发展，2005，42(6)

[5] 杨德志，黄华，张建刚，等. 大容量、高性能、高扩展能力的蓝鲸分布式文件系统. 计算机研究与发展，2005，42(6)

(上接第 7 页)

Parallel Coordinates // The 8th International Conference on Information Visualisation. London, UK, 2006

[45] Ward M O. XmdvTool: Integrating Multiple Methods for Visualizing Multivariate Data // IEEE Visualization '94. Los Alamitos, CA, 1994

[46] Buja A, Cook D, Swayne D F. XGobi: Interactive high-dimensional data visualization. Journal of Computational and Graphical Statistics, 1996, 5: 78-99

[47] Yin Hujun. ViSOM-A Novel Method for Multivariate Data Projection and Structure Visualization. IEEE Transactions on Neural Networks, 2002, 13(1): 237-243