FlatLFS:一种面向海量小文件处理优化的轻量级文件系统*

付松龄,廖湘科,黄辰林,王 蕾,李姗姗
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:海量小文件的高效存储和访问是当前分布式文件系统面临的主要挑战之一。以GFS和HDFS为代表的分布式文件系统大多面向海量大文件的高效存储和访问设计,缺乏小文件处理的针对性优化,导致访问海量小文件时效率低下。针对分布式文件系统中海量小文件访问时的数据服务器优化问题,提出了一种采用扁平式数据存储方法的轻量级文件系统FlatLFS,取代传统文件系统对上层分布式文件系统提供数据存储和访问支持,提高了数据服务器处理小数据块时的I/O性能,从而提升了整个分布式文件系统的性能。实验表明,当数据块大小设定为1M时,FlatLFS的随机读性能分别比ext3、ext4、reiserfs高135%、112%和122%。

关键词:分布式文件系统;海量小文件;ext文件系统;云计算
中图分类号:TP333;TP316.4 **文献标志码**:A **文章编号**:1001-2486(2013)02-0120-07

FlatLFS: a lightweight file system for optimizing the performance of accessing massive small files

FU Songling, LIAO Xiangke, HUANG Chenlin, WANG Lei, LI Shanshan
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: The storage and access of massive small files are one of the challenges in the design of DFS(Distributed file system). Most of the DFSs, such as GFS and HDFS, are designed for handling massive big files. The performance of DFSs decreases greatly when accessing massive small files without special optimization for small files. This research focuses on the optimizing of the performance of data server in handling massive small files, and presents a Flat Lightweight File System called FlatLFS in which the user data are managed flat in disks. FlatLFS is supposed to substitute the traditional file system when accessing user data for upper DFSs. With the improvement of the performance of small data block processing on data servers by FlatLFS, the performance of the whole DFSs is greatly improved. The effectiveness of FlatLFS is proved with intensive experiments: when the size of data block is 1M, the performance of random read of FlatLFS is 135%, 112% and 122% higher than ext 3, ext4 and reiserfs respectively.

Key words: distributed file system; massive small files; extended file system; cloud computing

分布式文件系统是云计算平台的核心技术之一,也是当前的研究热点。业界先后涌现了很多分布式文件系统,如GFS[1]、HDFS[2]、KosmosFS[3]、MooseFS[4]、Haystack[5]、TFS[6]等。其中,HDFS是GFS的开源版,研究较为广泛,已被Yahoo、Cloudera[7],Mapr[8]等大量商用。GFS和HDFS都是为海量大文件的高效存储和访问而设计的,它们首先将用户大文件切分为若干64M的数据块,并将元数据(如用户大文件、数据块以及数据服务器之间的映射关系)存放在一个元数据服务器中,将数据块以文件的形式存放在数据服务器中。在处理小文件时,系统中的文件个数和数据块数急剧增加,带来两个问题[9-11]:(1)文件总数受限问题:元数据量急剧增加,使得文件个数和数据块数受元数据服务器内存容量限制;(2)性能问题:传统文件系统处理小文件的性能较低,造成数据服务器处理小文件的性能急剧下降。因此它们并不适合处理小文件任务。如何提高分布式文件系统处理海量小文件的能力已经成为亟待解决的问题。

目前主要有三种方法来解决分布式文件系统处理小文件时文件总数受限问题。(1)将若干小文件合并到大数据块中,如Facebook的Haystack、淘宝的TFS、Hadoop Archive[12-13]、Sequence File[14]等。前两者属于系统级解决方案,修改分布式文件系统本身,后两者属于应用级解决方案,是建立在HDFS之上的文件打包工具。这种方法在数据访问时依赖多级索引,适合处理KB级的

* **收稿日期**:2012-04-05
基金项目:国家核高基重大专项(2012zx01040001)
作者简介:付松龄(1978—),男,四川眉山人,讲师,博士研究生;E-mail: slfu@nudt.edu.cn;
廖湘科(通信作者),男,教授,博士生导师,E-mail: xkliao@263.net

超小文件，效率较低。（2）分组存储技术，如 FastDFS[15]，将集群分为多个组，同组内的存储服务器之间是互备关系。该技术的缺点是不支持大文件存储，且在组内数据服务器配置不一致的时候存在空间浪费现象。（3）减小数据块大小，如 Google 的下一代分布式文件系统 GFS2[16]，采用了分布式元数据服务器并将数据块大小从 64M 减小为 1M。但是由于传统文件系统处理小文件时性能较低[17]，导致数据服务器的性能低下，整体性能受到很大制约。所有的 GFS 开源实现版本，如 HDFS、KosmosFS、MooseFS 等，在处理小文件时均存在类似的问题。

对于大部分数据为 MB 级文件的应用场景，GFS2 的方法更合适。本文针对与 GFS2 类似的上层分布式文件系统，提出了一种适用于海量小文件访问的轻量级文件系统 FlatLFS（Flat Lightweight File System），取代操作系统中的传统文件系统为上层分布式文件系统提供数据块的存取支持，通过优化数据服务器的数据块存取性能来优化整个分布式文件系统的性能。实验表明，FlatLFS 可以显著提高数据服务器的小数据块存取性能。

1　研究动机

与 GFS2 类似的分布式文件系统解决了面向小文件处理任务的可用性问题，但是与处理大文件任务相比，性能较低[9-10]。其主要原因在于为支持小文件存取，DFS 的元数据管理复杂性增加，以及海量小文件处理时带来的频繁小数据量磁盘 I/O 操作引起的数据访问性能损耗。

1.1　分布式文件系统文件访问分析

类似于 GFS2 的分布式文件系统对文件的管理方式都是相似的[1,18-19]。它们将文件划分为若干固定大小的数据块；元数据服务器在内存中保存所有文件的元数据信息，包含文件与数据块之间以及数据块与数据服务器之间的映射关系；数据服务器以文件的形式在本地保存数据块，每个数据块对应两个文件：元数据文件和数据块文件，文件以数据块 id 命名。客户端访问文件时，先向元数据服务器申请获取文件中每个数据块的数据服务器地址，然后直接访问数据服务器上的相应数据块。

客户端访问一个文件的具体过程如下：

1. 客户端请求元数据服务器访问一个文件，用 T_1 表示；

2. 元数据服务器在内存中查询该文件所对应的数据块，以及这些数据块所在的数据服务器地址，用 T_2 表示；

3. 元数据服务器将各数据块的块号和对应的数据服务器地址返回给客户端，用 T_3 表示；

4. 客户端依次访问每一个数据块：

（a）将块号传给相应的数据服务器，请求获得该数据块，用 $T_{4.a}$ 表示；

（b）数据服务器访问相应的数据块文件和元数据文件，用 $T_{4.b}$ 表示；

（c）数据服务器将结果返回给客户端，用 $T_{4.c}$ 表示。

假设一个文件包括 n 个数据块，那么客户端读取整个文件所花费的时间可以用式（1）来表示。

$$T = T_1 + T_2 + T_3 + \sum_{1}^{n} (T_{4.a} + T_{4.b} + T_{4.c}) \quad (1)$$

网络传输时间主要决定于网络硬件环境和传输数据量，T_1、T_3、$T_{4.a}$、$T_{4.c}$ 很难优化；由于元数据服务器中的所有元数据信息均保存在内存中，其时间开销在总时间中仅占很少的比例，因此 T_2 也很难进一步得到优化。而 $T_{4.b}$ 的大小由本地文件系统的 I/O 效率决定，跟数据块的组织管理方式和读写方式相关，有进一步优化的可能。

1.2　数据服务器本地文件访问分析

本课题组针对 ext3、ext4 和 reiserfs 文件系统做了不同大小文件的随机读测试（软硬件配置见表 1），结果如图 1 所示。

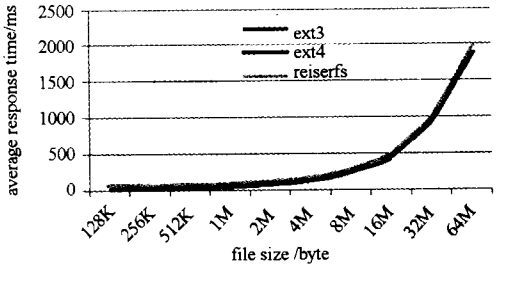


图 1　ext3、ext4、reiserfs 随机读性能

Fig. 1　Performance of random read in ext3/ext4/reiserfs

结果表明，传统文件系统中，文件随机读性能体现出两大特点：1）小文件随机访问的时间开销与文件大小关系不大；2）大文件随机访问的时间开销与文件大小成正比关系。

在传统文件系统中，文件被组织成层次式的树形结构[20]，文件数据在磁盘上分块存放，称为磁盘块。前 12 个磁盘块地址保存在文件 inode 中，其余磁盘块地址都以数据的形式保存在磁盘块（称为地址块）中。当访问文件数据时，必须先逐级读取文件绝对路径中的每一个目录，找到目标文件，然后读取目标文件的 inode 和地址块，根

据其中记录的磁盘块物理地址读取文件数据。

假设数据服务器上某个数据块的存储路径包含 x 个父目录，数据块文件包含 y 个磁盘块，元数据文件包含 z 个磁盘块，则读取数据块的时间可以用式(2)表示。

$$T_{4.b} = T_{\text{seek}} + T_{\text{data}} \tag{2}$$

其中 T_{seek} 表示寻找并读取目标文件 inode 的时间，T_{data} 表示读取文件数据的时间，可以分别用式(3)和(4)表示。

$$T_{\text{seek}} = \sum_{1}^{x}(T_{\text{inode}} + T_{\text{item}} + T_{\text{findSubDir}}) + T_{\text{inode(targetFile)}} + T_{\text{inode(targetMetaFile)}} \tag{3}$$

$$T_{\text{data}} = \sum_{1}^{y}(T_{\text{block}}) + \sum_{1}^{z}(T_{\text{block}}) \tag{4}$$

其中 T_{inode} 表示读取目录 inode 的时间，T_{item} 表示读取目录数据的时间，$T_{\text{findSubDir}}$ 表示从父目录数据中查询子目录目录项并解析出其 inode 存储地址的时间，$T_{\text{inode(targetFile)}}$ 表示读取数据块文件 inode 的时间，$T_{\text{inode(targetMetaFile)}}$ 表示读取元数据文件 inode 的时间，T_{block} 表示读取一个磁盘块的时间。在不同的文件系统中，T_{data} 的值不尽相同。比如，ext4 引入 extent[21] 技术，减少了 I/O 次数，从而缩短了 $\sum_{1}^{y}(T_{\text{block}})$。

从式(3)和(4)可知，读取数据块文件的过程中涉及的磁盘 I/O 操作次数较多，且每次磁盘 I/O 操作的数据量均不大。

当数据块较大时，文件总数有限，文件系统可以在内核缓存所有文件的 inode 以及一部分数据，数据寻址时间在数据访问时间中所占比例不大，因此性能较高。但是，当数据块较小时，文件总数成倍增长。假设数据块大小为 1MB，总量为 10TB，采用 ext4 文件系统，则所有文件的元数据总量大约为 80G。文件系统不可能在内核缓存所有文件的元数据，磁盘 I/O 次数成倍增长，数据寻址时间在数据访问时间中所占比例较大，性能急剧下降。这也从理论上解释了图 1 的测试结果。

综上所述，从分布式文件系统的角度来看，数据服务器仅提供固定大小数据块的存取功能，而传统文件系统的层次式树形文件结构过于复杂，小文件访问性能较低，造成数据服务器的数据块存取性能低下。我们可以简化传统文件系统的功能来换取性能上的提升。

2 FlatLFS 设计与实现

FlatLFS 运行于数据服务器之上，主要目标是优化 IO 操作性能，尤其是随机读性能。

2.1 基本思想

FlatLFS 的核心思想是通过减少数据块访问中的磁盘 I/O 操作次数来提高性能。它取消了文件和目录的概念，绕过操作系统原有文件系统，直接访问磁盘，将固定大小的数据块扁平式地保存在磁盘中，向上层分布式文件系统提供存取功能。

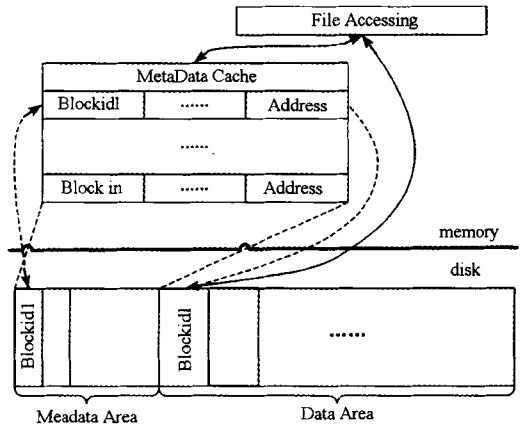


图 2 FlatLFS 数据块定位

Fig. 2 Locating data block in FlatLFS

FlatLFS 中，所有数据块的元数据(包含数据块在磁盘上的物理存储地址)集中存放，并全部缓存在内存中；数据块本身不再划分更细粒度的磁盘块，每个数据块在磁盘上的存储空间是连续的。访问数据块时，FlatLFS 首先在内存中查询获得数据块的物理存储地址，然后仅需一次磁盘 I/O操作即可完成请求。FlatLFS 的数据定位模式如图 2 所示。

2.2 FlatLFS 数据组织框架

FlatLFS 在磁盘上的数据组织框架如图 3 所示。

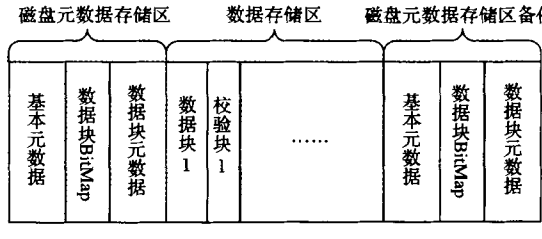


图 3 FlatLFS 中的磁盘数据组织框架

Fig. 3 Framework of data organization in FlatLFS

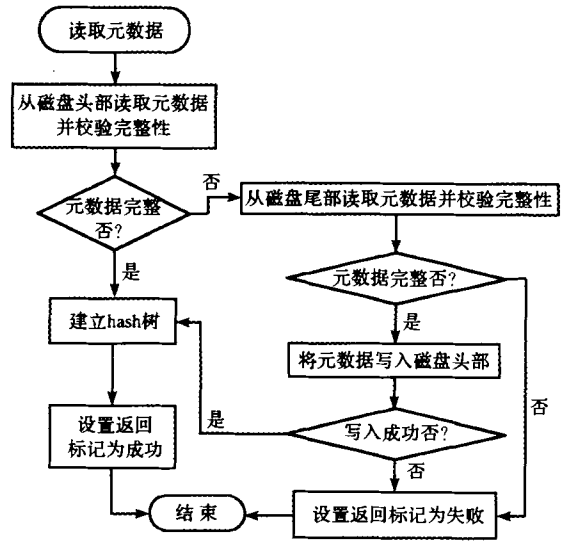
磁盘被分为两个区域：元数据存储区和数据存储区。磁盘格式化时，根据磁盘大小和数据块大小计算整个磁盘所能容纳的数据块总数，然后计算元数据存储区的大小，并在磁盘头部和尾部分别划分相应大小的存储空间作为元数据存储区，最后将元数据存储区之外的数据存储区划分为一个个相同大小的“存储块”，每个存储块的前面部分存数据块本身的内容，后面部分存数据块

的完整性校验信息。

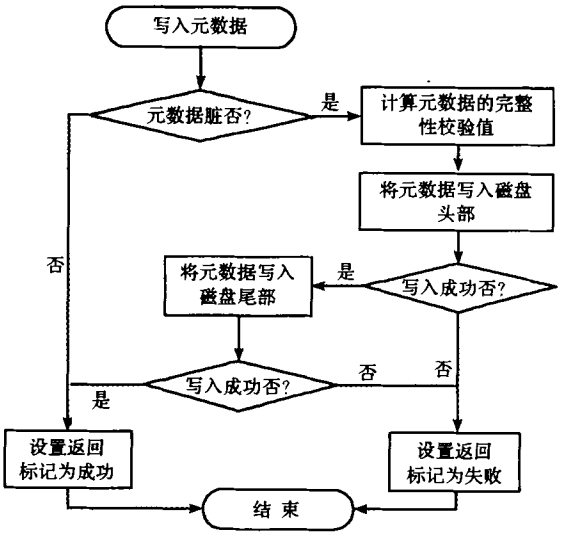
元数据存储区存储的信息包括基本元数据，数据块 bitmap，数据块元数据。其中，基本元数据存储磁盘基本信息，如数据块总数等；数据块 bitmap 表征数据块的使用情况，其中每一位对应一个数据块；数据块元数据存储所有数据块的元数据信息，其中每一个元数据项存储相应数据块的 id、类型、版本、磁盘物理存储地址等信息。

2.3　FlatLFS 数据访问操作

FlatLFS 对上层分布式文件系统提供固定大小数据块的存取功能。它在数据服务器启动时将磁盘元数据读入内存，并自动纠错。数据块访问所带来的元数据修改操作全部在内存中完成，FlatLFS 在后台周期性地将脏的元数据写入磁盘。图 4 为元数据的处理流程图。



（a）读取元数据流程图

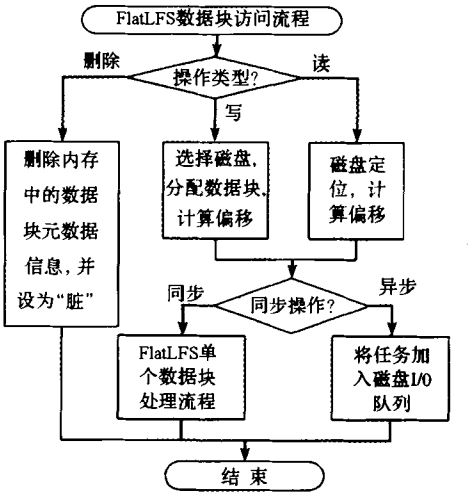


（b）写入元数据流程图

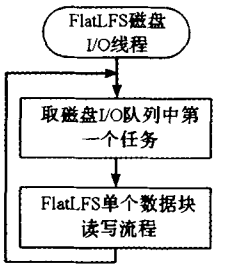
图 4　FlatLFS 元数据处理流程

Fig. 4　Metadata processing in FlatLFS

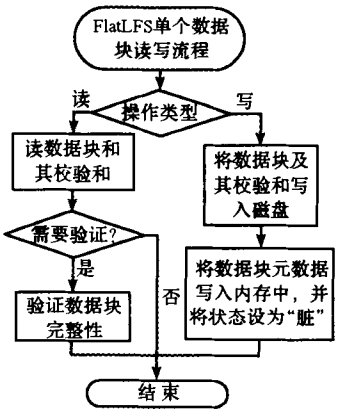
FlatLFS 删除数据块操作较为简单，只需将磁盘元数据信息中的相应数据块记录删除即可。FlatLFS 支持同步或者异步读写数据块，处理流程如图 5 所示。



（a）FlatLFS 数据块访问流程



（b）FlatLFS 数据块处理流程



（c）FlatLFS 单个数据块读写流程

图 5　FlatLFS 数据块处理流程

Fig. 5 Data block process in FlatLFS

FlatLFS 写数据块之前，首先选择负载最小的磁盘，在第一个可用数据块位置写入数据块内容和其校验和，然后再将数据块元数据写入缓存。

FlatLFS 读数据块时，首先根据数据块 id 从内存中查询数据块所在磁盘及在该磁盘内的物理存储地址，然后将数据块及其校验和读取出来，并根据需要验证数据块的完整性。

2.4 FlatLFS 实现

本课题组在 Linux 操作系统 CentOS 5.5(内核版本 2.6.32)中实现了一个 FlatLFS 原型系统。基本元数据存储区为 128 字节,格式如图 6 所示,包括:元数据校验值、元数据大小、软件版本、格式化时间、磁盘大小、数据块大小、校验数据块大小、校验块大小、数据块总量、已使用数据块数量、下一个可用块位置、最近修改时间,其中每个值占 8 个字节,其余为保留元数据区,占 32 字节,从 128 字节位置开始为数据块 bitmap 和数据块元数据。

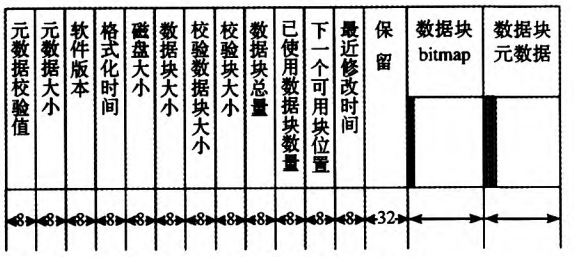


图 6 磁盘元数据存储区内部结构

Fig. 6 Structure of disk metadata storage area

元数据校验值采用最简单的奇偶校验方法进行计算;校验块大小固定为数据块大小的 1/128,每 128 字节数据用奇偶校验算法产生 1 字节的校验和。

3 FlatLFS 性能测试与结果分析

所用的硬件平台及软件环境配置如表 1 所示。

表 1 测试硬件平台和软件环境

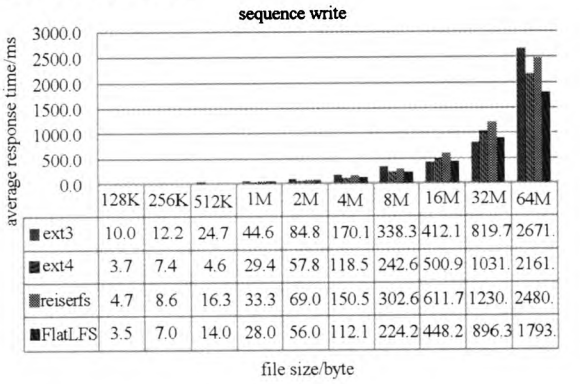
Tab. 1 Environments of hardware and software

配置项	参数		说明
CPU	Intel Core2 Duo T9600,2.33GHz		
Memory	2GB DDR2		
OS	CentOS 5.5		内核版本 2.6.32
Disk	Hitachi (HTS543225 L9A300), 250GB, SATA, 5400rpm, 8MB cache, 平均寻道时间 12ms	40G	Linux 操作系统根分区
		8GB	Swap 分区
		160G	数据区

由于数据服务器上的文件管理和访问具有固定模式,因此不能使用现有的普通文件系统测试工具,如 IOZone,Bonnie ++ 等。我们自编测试程序,按顺序写、顺序读、随机读、随机写的顺序,依次对比了 FlatLFS 和传统文件系统 ext3、ext4 和 reiserfs 在数据服务器上的文件管理和访问模式下的性能。

对于传统文件系统的测试,我们参考了 HDFS 的设计,每个目录中保存 64 个数据块文件和 64 个子目录,每个数据块对应两个文件:数据块文件和元数据文件。在顺序读写测试中,按编号从小到大依次访问每个数据块文件;在随机读写测试中,首先随机产生一个块号,然后读写相应的数据块文件。测试结果如图 7 所示。

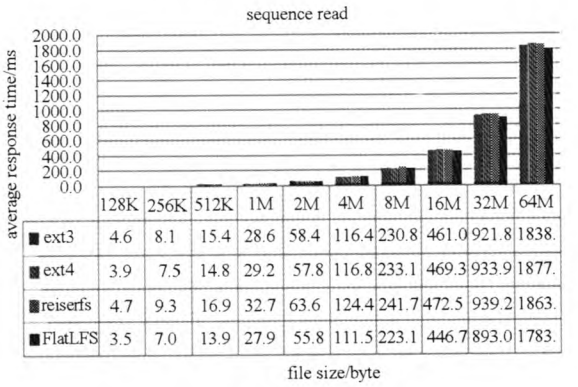
首先,从纵向比较来看,如图 7(a)、(b)所示,顺序读写的性能基本上与文件大小成正比;但是如图 7(c)、(d)所示,随机读写对小文件访问的性能影响较大,小文件随机访问性能与文件大小之间并不严格呈现线性正比关系,文件越小,相应时间降低越慢。



	128K	256K	512K	1M	2M	4M	8M	16M	32M	64M
ext3	10.0	12.2	24.7	44.6	84.8	170.1	338.3	412.1	819.7	2671.
ext4	3.7	7.4	4.6	29.4	57.8	118.5	242.6	500.9	1031.	2161.
reiserfs	4.7	8.6	16.3	33.3	69.0	150.5	302.6	611.7	1230.	2480.
FlatLFS	3.5	7.0	14.0	28.0	56.0	112.1	224.2	448.2	896.3	1793.

(a)顺序写

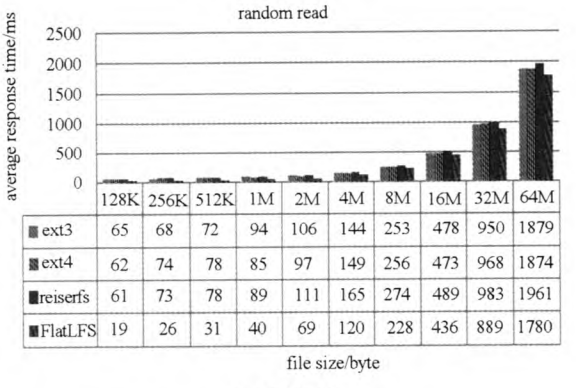
(a) Sequential write



	128K	256K	512K	1M	2M	4M	8M	16M	32M	64M
ext3	4.6	8.1	15.4	28.6	58.4	116.4	230.8	461.0	921.8	1838.
ext4	3.9	7.5	14.8	29.2	57.8	116.8	233.1	469.3	933.9	1877.
reiserfs	4.7	9.3	16.9	32.7	63.6	124.4	241.7	472.5	939.2	1863.
FlatLFS	3.5	7.0	13.9	27.9	55.8	111.5	223.1	446.7	893.0	1783.

(b)顺序读

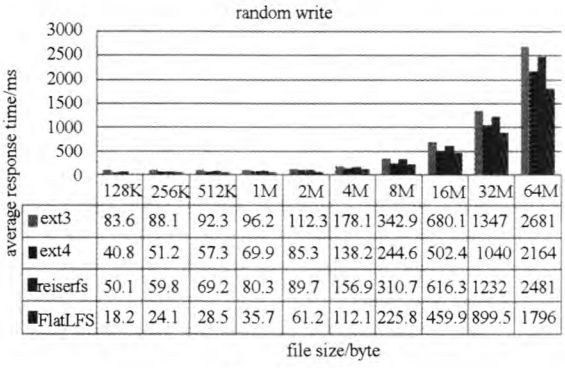
(b) Sequential read



	128K	256K	512K	1M	2M	4M	8M	16M	32M	64M
ext3	65	68	72	94	106	144	253	478	950	1879
ext4	62	74	78	85	97	149	256	473	968	1874
reiserfs	61	73	78	89	111	165	274	489	983	1961
FlatLFS	19	26	31	40	69	120	228	436	889	1780

(c)随机读

(c) Random read



	128K	256K	512K	1M	2M	4M	8M	16M	32M	64M
ext3	83.6	88.1	92.3	96.2	112.3	178.1	342.9	680.1	1347	2681
ext4	40.8	51.2	57.3	69.9	85.3	138.2	244.6	502.4	1040	2164
reiserfs	50.1	59.8	69.2	80.3	89.7	156.9	310.7	616.3	1232	2481
FlatLFS	18.2	24.1	28.5	35.7	61.2	112.1	225.8	459.9	899.5	1796

（d）随机写

（d）Random write

图 7　32GB 不同大小文件数据的性能

Fig. 7　Performance of 32GB file data with different size

然后进行横向比较，总的说来，大文件性能相差不大，但是小文件性能差别较大。如图 7（a）、（b）所示，对于小文件顺序读写的情况，FlatLFS 性能与 ext4 相似，全面领先于 ext3 和 reiserfs。如图 7（c）所示，对于小文件随机读的情况，FlatLFS 全面领先于另外三种传统文件系统；如图 7（d）所示，对于小文件随机写的情况，FlatLFS 优于 ext4 和 Reiserfs，而 ext4 和 Reiserfs 又优于 ext3。

对于分布式文件系统中的数据服务器而言，随机读访问占主导地位。从测试结果来看，假设数据块大小为 1MB，则 FlatLFS 的随机读性能分别比 ext3、ext4、reiserfs 高 135%、112%、122%，优化效果较为明显。

4　FlatLFS 分析

FlatLFS 中，每个数据块被存储在连续磁盘空间中，有且仅有一个物理存储地址。另外，FlatLFS 也大大减少了数据块元数据总量，同样以 1MB 大小的数据块和 10TB 数据总量为例，数据块总数为 1M，FlatLFS 最少情况下（元数据仅包含 8 字节数据块 ID 和 8 字节磁盘地址）仅需要大约 16MB 的元数据就可以满足要求，与原来的 80GB 相比，可以忽略不计。所有数据块的元数据集中存放在磁盘元数据存储区，在运行时全部缓存到内存中，提高了数据块查询效率。数据块寻址时仅需在内存中查询一次即可，时间用 T'_{seek} 表示。因为内存操作的速度远远大于磁盘 I/O 操作的速度，因此 T'_{seek} 远远小于 T_{seek}。

$$T'_{seek} \ll T_{seek} \tag{5}$$

另外，每个数据块读写时仅需要一次 I/O 操作，所耗时间 T'_{data} 可表示为 T^{y+z}_{block}。用一次磁盘 I/O 读写 $y+z$ 个磁盘块的时间，小于 $y+z$ 次磁盘 I/O 读写相同数据量的时间。

$$T'_{data} = T^{y+z}_{block} < \sum_{1}^{y}(T_{block}) + \sum_{1}^{z}(T_{block}) = T_{data} \tag{6}$$

结合式（2）可知，FlatLFS 读取数据块所耗时间 $T'_{4.b} < T_{4.b}$。因此，FlatLFS 与传统文件系统相比，性能得到了优化。

对传统文件系统而言，数据块越小，定位时间在总时间中所占的比例越大，性能下降越快。而对 FlatLFS 而言，无论数据块多大，均是直接读写磁盘，影响不大。当然，在处理小数据块情况下，FlatLFS 性能也有一定下降，原因是磁盘平均延时（盘片将磁道上的目标数据转到磁头下所需的时间）在总时间中所占的比例逐渐增大所造成的。

5　结论及展望

面向海量数据处理的分布式文件系统是当前的热点研究之一。目前，大多数分布式文件系统侧重于大文件的高效存储和访问，在海量小文件处理研究方面仅解决了可用性问题而没有解决性能问题。本文针对离散随机均匀分布的小文件访问，提出了一种通过优化数据服务器本地数据块存取效率来优化整个分布式文件系统性能的方法，设计并实现了一个采用扁平式数据存储方法的轻量级文件系统 FlatLFS，提高了系统存取数据块的性能。实验表明，当数据块大小为 1M 时，FlatLFS 的随机读性能分别比 ext3、ext4、reiserfs 高 135%、112% 和 122%。这在小数据处理需求日益增加的今天，具有很高的实用价值。

当然，FlatLFS 摒弃了传统文件系统的层次式文件管理模式，以牺牲灵活性为代价换来高效率，仅适合用于后台分布式文件系统数据服务器的数据块管理，不适合用于直接面向用户的通用文件数据管理。

下一步工作包括两方面：（1）完善 FlatLFS 的实现；（2）把 FlatLFS 集成到某种分布式文件系统中，并通过优化元数据访问性能进一步改善整个分布式文件系统的性能。

参考文献（References）

[1] Ghemawat S, Gobioff H, Leung S. The Google file system [C]//19th Symposium on Operating Systems Principles, NY: IEEE, 2003: 29 - 43.

[2] The apache hadoop project. hadoop distributed file system [EB/OL]. (2012 - 12 - 05) [2012 - 12 - 20]. http://hadoop.apache.org/.

[3] Srirams. Kosmos file system[EB/OL]. (2011 - 05) [2011 - 11 - 15]. http://code.google.com/p/kosmosfs.

[4] Moose file system[EB/OL]. (2012 - 08 - 16) [2012 - 09 -

15]. http://www.moosefs.org.

[5] Beaver D, et al. Finding a needle in Haystack: Facebook's photo storage[C] // 9th USENIX Symposium on Operating Systems Design and Implementation, October 4 - 6 Canada 2010.

[6] Taobao file system[CP/OL]. (2012-12-04)[2012-12-21]. http://code.taobao.org/p/tfs/src/.

[7] Cloudera big data solution[EB/OL]. [2012-12-22]. http://www.cloudera.com.

[8] MapR big data solution[EB/OL]. [2012-12-22]. http://www.mapr.com.

[9] McKusick M K, Quinlan S. GFS: Evolution on fast-forward [R/OL]. (2009-08-07)[2011-10-09]. http://queue.acm.org/detail.cfm? id=1594206.

[10] White T. The small files problem[R/OL]. (2009-02-02)[2011-08-23]. http://www.cloudera.com/blog/2009/02/the-small-files-problem/.

[11] Higkoo. Hadoop不适合大量小文件存储的证实[R/OL]. (2011-02-25)[2011-08-23]. http://hi.baidu.com/higkoo/blog/item/edd259129b7cf745f919b845.html.

[12] Hadoop archive guide[EB/OL]. (2010-08-17)[2011-09-21]. http://hadoop.apache.org/mapreduce/docs/r0.21.0/hadoop_archives.html.

[13] Tankel D. Hadoop archive: File compaction for HDFS[R/OL]. (2010-07-27)[2011-06-16]. http://developer.yahoo.com/blogs/hadoop/posts/2010/07/hadoop_archive_file_compaction/.

[14] Sequence File[EB/OL]. (2009-09-20)[2011-06-19]. http://wiki.apache.org/hadoop/SequenceFile.

[15] Happyfis. FastDFS distributed file system[CP/OL]. (2012-12-01)[2012-12-20]. http://code.google.com/p/fastdfs/

[16] Metz C. Google file system II: Dawn of the multiplying master nodes[R/OL]. (2009-08-12)[2011-09-21]. http://www.theregister.co.uk/2009/08/12/google_file_system_part_deux/.

[17] Vranos I. Optimising performance for many small files[R/OL]. (2011-06-02)[2011-08-09]. http://www.linux-archive.org/ubuntu-user/535087-optimising-performance-many-small-files.html.

[18] Borthaku D. HDFS architecture[R/OL]. (2009-04-09)[2011-09-03]. http://hadoop.apache.org/common/docs/r0.20.0/hdfs_design.pdf.

[19] Chuyu. TFS introduction[R/OL]. (2010-09)[2011-12-05]. http://code.taobao.org/trac/tfs/wiki/intro.

[20] Extended file system[R/OL]. (2012-06-07)[2012-08-23]. http://en.wikipedia.org/wiki/Extended_file_system.

[21] Avantika M, Cao M, et al. The new ext4 filesystem: Current status and future plans [C]//Proceedings of the Linux Symposium. Ottawa, Jan. 15, 2008.