基于Hadoop的小文件存储优化方案

李 孟，曹 晟，秦志光

(电子科技大学计算机科学与工程学院 成都 611731)

【摘要】Hadoop作为成熟的分布式云平台，对较大的文件提供了可靠高效的存储服务，但在处理海量小文件时效率显著降低。该文提出了基于Hadoop的海量教育资源小文件的存储优化方案，利用教育资源小文件间的关联关系，将小文件进行合并成大文件以减少文件数量，并索引机制访问小文件、元数据缓存和关联小文件预取机制来提高文件的读取效率。实验结果表明，该方法提高了Hadoop文件系统存储小文件的存取效率。

关 键 词 Hadoop; 索引机制; 关联关系; 小文件存储

中图分类号 TP391.6 文献标志码 A doi:10.3969/j.issn.1001-0548.2016.01.024

Storage Optimization Method of Small Files Based on Hadoop

LI Meng, CAO Sheng, and QIN Zhi-guang

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 611731)

Abstract Hadoop distributes file system (HDFS) can process large amounts of data effectively through large clusters. However, HDFS is designed to handle large files and suffers performance penalty while dealing with large number of small files. An approach based on HDFS is proposed to improve storage efficiency of small files in HDFS. The main idea is to classify the mass small files, merge them by classes, and index the merged files aiming at reducing the amount of index items in namenodes and improving the storage efficiency. Experimental results show that the storage efficiency of small files is improved contrasting to Hadoop Archives (HAR files).

Key words Hadoop; index mechanism; relationship; storage of small files

HDFS(hadoop distributed file system)是一个具有高容错性、成本低廉性等特点的分布式文件系统。在实际应用中，小文件的数量远远多于大文件的数量。尤其是在互联网应用中，诸如网页中的图片、文档以及包括其他的数据信息xml文件等多数为小文件。 HDFS设计来对大文件进行流式存储，在处理小文件时会产生一些问题。因此，如何高效地存储和处理大量的小文件成为一个研究热点[1-3]。

HDFS对于处理海量小文件的存储有以下不足：1) 海量小文件耗费主节点内存，可能造成NAMEDODE瓶颈问题；2) 海量小文件的I/O效率低，没有一种优化机制来提高I/O性能；3) HDFS下没有明确的能够区分何为小文件的分界点；4) 没有考虑海量小文件之间的相关性等。

本文所提的教育资源小文件包括各种形式的教育资源，如word文档、pdf文档、ppt课件及文件资料等，只要和教育资源相关的内容且大小远小于Hadoop的存储块64 MB的文件都可称为教育资源小文件。这类文件具有3个特点：1) 这些文件通常在几十到几百KB，和Hadoop的存储块大小64 MB相比较小；2) 小文件之间有关联性，整理归档同类课程的小文件然后进行合并；3) 小文件的数量很多，因为网络中存在的各种学习资源量非常大，且不断地以指数级速度增长。

1 相关工作

针对小文件在HDFS中存储出现的问题，目前出现了许多的解决方案[4-10]，归纳起来主要分为：

1) 第一类都对写入的小文件进行缓冲，将缓冲的多个小文件合并为一个临时文件，将临时文件的元数据和数据对象存储至元数据服务器节点和数据服务器节点的后端存储中，从而可以有效地提高集群文件系统服务的响应时间和速度，提升数据整体的单位时间数据读写次数、吞吐量。

2) 第二类针对机群文件系统中的小文件提出新的存储和访问，步骤为：① 设置阈值，区分大小

收稿日期：2014 – 10 – 08; 修回日期：2015 – 05 – 11
基金项目：教育部-中国移动科研基金(MCM20121041)；国家自然科学基金(61133016, 61103206)；国家863计划(2011AA010706)
作者简介：李孟(1981 –)，女，博士生，主要从事计算机网络和知识工程方面的研究.

文件；② 在元数据服务器上，存储小文件的数据；③ 在元数据服务器上进行小文件创建、读写和删除。由于把小文件的数据存储在元数据服务器上，这样对于小文件的IO访问操作，如创建、读写和删除等，发起IO访问的客户端只需要与元数据服务器交互，无需与数据服务器交互，减少了小文件访问的网络延迟，提高了小文件IO的性能，从而从整体上提高了机群文件系统的IO性能。

3) 第三类面向大批量文件数据存取方法，包括将所有小文件的数据合并成一个大文件；建立每个小文件的文件名及其文件编号的一一对应关系；建立每个所述文件编号与小文件的文件信息的对应关系，文件信息包括所述小文件在所述大文件中的位置。相应地，还公开一种大批量文件数据读取方法，用于读取按照本文的存放方法存放的文件数据，包括有：根据小文件的文件名来获得所述小文件的文件编号；根据所述文件编号获得所述小文件的文件信息；根据所述文件信息获得所述小文件在大文件中的位置；根据所述小文件在大文件中的位置，通过所述大文件的IO接口实现对所述小文件数据的读取。

上述现有解决小文件存储问题的专利技术方案存在以下问题：1) 已有小文件存储效率的研究主要集中在非云存储的文件系统上，而不是针对云存储环境下的分布式文件系统，即Hadoop分布式文件系统上的存储优化方法；2) 现有方法虽然提出了合并小文件的方法，但在合并时没有考虑文件之间的关联关系，且合并后读取文件时所增加的索引查询会影响文件的读取效率。

2 本文方法

本文提出了一种基于Hadoop的海量可归类小文件关联存储方法，主要解决可归类小文件的存取效率问题。针对属于某一类别的独立的小文件进行文件聚合和全局索引管理，大幅度提高了内存利用率，提高单位内存支持的最大文件数量。

2.1 文件归并

1) 将属于某个大文件的所有小文件归并为一个文件，称为merged file；

2) 对每个merged file建立一个局部索引，并在上传时将局部索引文件与文件实体一同存放在Hadoop系统的DataNode上；

3) 在读取非独立小文件时，采用元数据缓存、局部索引文件预取和关联文件预取提高文件的读取效率。

为每一个merged file建立一个局部索引文件，记录属于该merged file的所有小文件的起始位置和长度，局部索引文件位于该merged file的每一个块的起始位置，并且只为该merged file服务。

1) 局部索引文件结构

局部索引文件采用静态查找表结构，由索引头部、序列索引和文件索引3部分构成；其中索引头部由占1字节的版本号、占4字节的索引项数和占4字节的局部索引文件长度组成；序列索引由占4字节的序列名称、占4字节的文件索引的起始编号和占4字节的文件索引项数构成；文件索引项由占16字节的文件名称、占4字节的文件长度和占4字节的文件偏移构成；

2) 读文件时对局部索引文件的操作

首先根据merged file名从NameNode获取元数据；然后由merged file的元数据，从Hadoop文件系统的相应DataNode读取指定的数据块，并根据数据块内索引文件长度项读取数据块的局部索引文件；最后根据小文件名称，查找局部索引文件，获得该小文件的起始位置和长度，从而完成对小文件的读操作。当该merged file的小文件数目少于1 000 时，采用顺序查找方法；当小文件的数目超过1 000 时，顺序查找会影响读取性能，在文件索引上建立序列索引，避免查询的开销过大；如果是一级索引格式，则从局部索引文件中第一个目录项开始，与请求小文件名称逐条对比，若文件索引项的小文件名称符合，则查找成功，返回该小文件索引记录；否则，继续查找直至最后一个记录项，若没有记录项合，则返回小文件名查找失败。

如果是两级索引格式，则查找分为两个阶段：首先根据待查小文件的序列名称，在序列索引表中查找序列，如果索引表内没有记录项，则表明该序列不存在；否则根据序列指定的位置，开始顺序查找文件索引位置。

文件归并操作在Hadoop文件系统的客户端上进行，将属于同一个大文件的所有小文件合并成一个文件；具体步骤如下：

1) 计算非独立小文件总数，根据文件的数目决定采用文件索引还是采用序列索引+文件索引，由于索引每项长度固定，计算得出局部索引文件的长度，用L_{index}表示，过程如下：

① 如果采用一级索引格式，用L_{findex}表示单个索引项的长度，Number表示小文件总数，L_{head}表示

索引头部的长度，则：

$$L_{\text{index}} = L_{\text{findex}} \times \text{Number} + L_{\text{head}}$$

② 如果采用两级索引结构，假设有N个序列，每一个序列的小文件数为Number1，Number2，…，Number$N$，序列索引长L_{sindex}为：

$$L_{\text{index}} = N \times L_{\text{sindex}} + \left(\text{Number1} + \text{Number2} + \cdots + \text{Number}N\right) \times L_{\text{findex}} + L_{\text{head}}$$

2) 计算该merged file所有非独立小文件的长度和与局部索引文件长度的和Lmerge，将Lmerge与Hadoop文件系统的块大小作比较；

3) 如果Lmerge小于Hadoop文件系统的块大小，则该merged file只占用一个数据块，所有文件按默认顺序存放：首先是局部索引文件，然后是小文件序列，小文件序列按照小文件的逻辑顺序依次排列，按照小文件顺序计算每个小文件的偏移和长度，建立局部索引文件，进行步骤4)；如果Lmerge超过Hadoop文件系统的块长，归并后文件会被分成多个数据块存储，当有小文件跨数据块时，采用边界填充算法写入一段空白文件将原来的数据块填充，然后在新申请的数据块中写入该文件；

4) 根据局部索引文件中每个文件的偏移对小文件进行归并，用空白文件填充两个文件之间的空白区域。

2.2 局部索引

过程如下：

1) 依次计算每个文件的偏移，在数据块的边界处，检查是否有文件会横跨两个数据块，如果没有，转向步骤3)，否则，转向步骤2)；

2) 在这个横跨两个数据块的小文件前，建立额外的局部索引文件，该索引文件的偏移是下一个数据块的起始位置，横跨小文件的偏移是该局部索引文件的结束位置，设块长为L_{block}，局部索引文件的大小为L_{index}，新块的序列号为W，新索引文件偏移量为L_{offset}，新索引文件长度为L_{length}，横跨小文件的偏移量为L_{foffset}，则：

$$L_{\text{offset}} = (W-1) \times L_{\text{block}}$$

$$L_{\text{length}} = L_{\text{index}}$$

$$L_{\text{foffset}} = L_{\text{offset}} + L_{\text{length}}$$

3) 对下一个数据块，重复步骤1)和步骤2)

在完成边界填充后，能够确定每一个小文件在merged file内的顺序和偏移，因此可以建立局部索引文件。

2.3 缓存与预取

元数据缓存、局部索引文件预取和关联文件预取包括：

1) 元数据缓存：当小文件被读取时，将小文件映射到merged file以获取merged file的元数据，NameNode将元数据返回给客户端后，客户端根据元数据信息与相应的DataNodes交互，然后客户端将该merged file的元数据缓存，则如果该merged file的其他小文件被请求时，能够直接从缓存中读取元数据，从而减少与NameNode的交互；

2) 局部索引文件预取：根据merged file的元数据，客户端获知从哪些数据块中读取被请求文件，如果局部索引文件已经被预取，当属于该merged file的小文件被请求时，客户端根据被缓存的索引信息，直接从对应DataNode中读取；否则，局部索引文件预取操作被触发，将局部索引文件预取到客户端的缓存中，在缓存中，预取得到的局部索引文件和元数据被处理，为每一个小文件生成元数据索引信息，索引信息包括：原始小文件文件名(16字节)、DataNode ID(4字节)、块ID(4字节)、偏移(4字节) 和长度(4字节) ；

3) 关联文件预取：同一个merged file的非独立小文件有着直观的关联关系和明确的逻辑顺序，当被请求的小文件返回到客户端后，关联文件预取操作被触发，根据文件之间的逻辑顺序将该merged file下的相关小文件预取。

2.4 碎片整理

碎片整理是对DataNode的数据块中存在的空白区域的再利用。当某个小文件被删除或者其他原因造成数据块中存在空闲区域，为了利用这些空间提出了碎片整理机制。该机制通过建立一个碎片索引集合，并利用二叉树结构来定位碎片。索引集合包括各碎片所在块中的偏移和碎片的长度，按碎片长度将其依次放入索引项中，通过对碎片索引集合中索引项的查找来定位碎片，然后进行插入操作将相应大小的文件存入块中，或删除操作删除不需要的小文件，这些操作与二叉树的查找、插入和删除操作相同。

当写入某类小文件时，将小文件所属大文件相应的块上，先通过大文件的碎片索引，查看该块中是否有适合的碎片可以存入小文件，则将该碎片分为两个部分，前一部分分配给待写入小文件，后一部分碎片作为新的碎片，在碎片索引集合，删除原碎片的索引项，为新的碎片插入索引项，在小文件

索引集合中插入新写入小文件的索引项；如果没有，对碎片索引集合不进行任何改动，直接在数据块末尾的空白区分配空间给小文件存储，并在小文件索引集合插入其索引项。

当删除小文件时，首先使用待删除小文件的文件名查找索引项，判断是否存在该文件，若不存在，则删除失败；如果存在，则在碎片索引集合中插入一条新的碎片索引项；然后在碎片索引集合，判断新的碎片索引项的相邻的数据单元是否同样为碎片数据，如果存在任何一边的数据单元是空白索引，那么合并多个数据碎片成一个大的数据碎片，并更新碎片索引，当数据碎片的相邻碎片是由于数据块的分界造成时，不需要进行数据碎片的合并。

3 实验与分析

为了验证所述方案的优化，本文设计了一个实验方案，用于测试在优化后的HDFS中小文件的读写效率的变化以及所用内存开销。由于基于HDFS的分布式文件系统的核心功能为分布式存储，因此客户端对文件的操作集中在文件上传和文件下载，但无论是上传操作还是下载操作其本质都是文件读取。实验过程中以HDFS中的NameNode内存占用、文件的数量和大小、文件读取时间等数据作为实验参考指标，将实验结果与在原生HDFS中的实验结果对比分析。

表1 实验环境

实验环境	参数
操作系统	ubuntu 12.04 64bit
jdk版本	1.7.45
hadoop版本	0.20.3
NameNode节点数目	1
DataNode节点数目	3
处理器/GHZ	intel双核2.4
内存/GB	4
硬盘/GB	250, 5 400 r/min
网络	1.0 Gbps以太网

3.1 内存占用实验结果与分析

为了分析基于HDFS的分布式文件存储系统中NameNode的内存占用情况，设计了以下实验方法。

选取100 000个大小为1～100 KB的小文件作为实验数据集，将这些小文件依次以不同的数量分别存储在原生的HDFS系统中和经过优化过的HDFS系统中。每次启动NameNode后，分析随机读取文件时NameNode的内存占用情况。

图1为文件大小及数量分布。将这100 000个文件以10种不同的数量规模分别存储在原生HDFS系统和优化后的HDFS系统中。然后在1 h的时间内不断的随机读取小文件，观察读取小文件时HDFS中NameNode所占用系统内存的变化并取其平均值。图2为NameNode内存大小占用图。

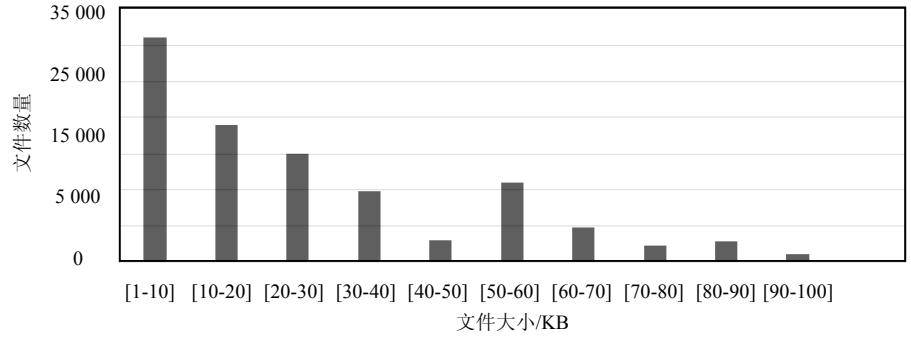

图1 文件大小及数量分布图

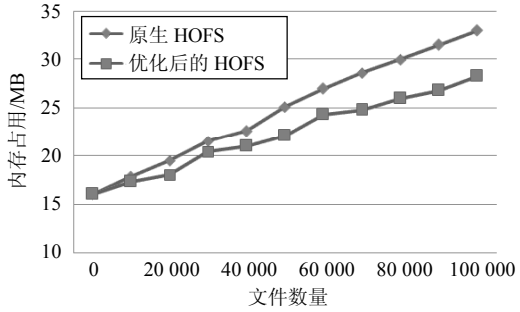

图2 NameNode内存大小占用情况

从图中可以看出当小文件数量不超过10 000时，本文设计的优化方案效果并不明显。但是随着小文件数量的不断增加，尤其是文件数量超过30 000后，优化后的HDFS中NameNode所占用的系统内存比原生HDFS节省，尤其是文件数量较大时，效果较为明显。

3.2 读写性能结果及分析

基于本文提出的优化方案，根据相应的合并规则对这些小文件进行进行合并存储。文件分类及索引处理时间为T_{sort}，存入HDFS的时间为T_{store}，文件总数量为N。则得到小文件的平均存储时间为：

$$T = (T_{\text{sort}} + T_{\text{store}}) / N$$

对于原生的HDFS系统，由于小文件在存入HDFS之前无须合并及建立索引之类的工作，因此上传到HDFS的时间即为小文件存储到HDFS中的时间$T' = T'_{\text{store}}$。

因此，同样选取100 000个大小为1～100 KB的小文件作为实验数据集。将这些小文件以不同的数量规模存入HDFS中，对相应的存储过程重复3次，最终3次的平均时间作为小文件存储时间。然后再将这些文件全部从HDFS中读取出来，同样重复3次，3次操作的平均时间为小文件的读取时间，如图3所示。

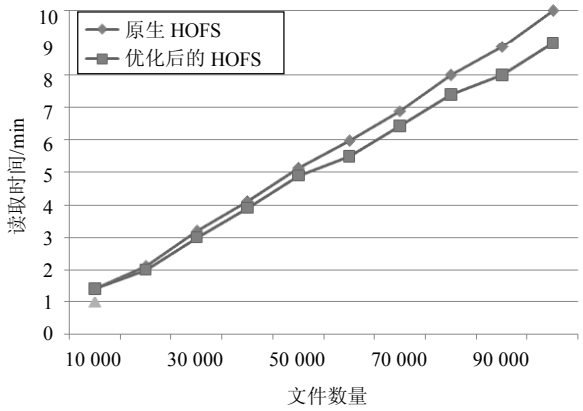

图3　小文件读取时间对比

在小文件读取方面，优化后的HDFS的小文件读取性能略有提升，其原因归结于NameNode在文件读取时查询速度的提升。但对于小文件的存储，优化后的HDFS的写入时间相对于原生HDFS有明显缩短。原因有两个方面：1) 虽然小文件文件在存入HDFS之前需要额外的索引建立与文件合并操作，但随着小文件索引的合理优化，这些操作额外消耗的时间相对于存储时间微乎其微；2) 本文提出的优化方案采用的文件合并策略，大幅度减少了NameNode中元数据的数量，因此小文件在写入HDFS时，NameNode检索HDFS空闲块的时间随着元数据的减少而减少。

4　结 束 语

本文提出的小文件归并及缓存策略将大量关联小文件合并成大文件后存于HDFS上，有效地缓解了NameNode主存的瓶颈问题；通过关联小文件及元数据的预取方案有效地提高了I/O性能，解决了HDFS上存储小文件存在的问题。对100 000个小文件进行实验，结果表明通过小文件合并及相关缓存预取策略使内存占用相对原始HDFS系统减少了19%，读取效率相对原始HDFS提高了20%。下一步将针对小文件的来源构成与分块进行更深入的研究，以进一步提高小文件在HDFS中的存取效率。

参 考 文 献

[1] LIU X, HAN J, ZHONG Y, et al. Implementing WebGIS on Hadoop: a case study of improving small file I/O performance on HDFS[C]//Cluster Computing and Workshops, 2009. CLUSTER'09. New Orleans, LA: IEEE, 2009: 1-8.

[2] MACKEY G, SEHRISH S, WANG J. Improving metadata management for small files in HDFS[C]//Cluster Computing and Workshops, 2009. CLUSTER'09. New Orleans, LA: IEEE, 2009: 1-4.

[3] JIANG L, LI B, SONG M. The optimization of HDFS based on small files[C]//Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on IEEE. Beijing: IEEE, 2010.

[4] BORTHAKUR D. The hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, 11: 21.

[5] SHAFER J, RIXNER S, COX A L. The hadoop distributed filesystem: balancing portability and performance[C]// Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium. White Plains, NY: IEEE, 2010: 122-133.

[6] DUTCH M, BOLOSKY W. A study of practical deduplication[C]//Proceedings of the 9th USENIX Conference on File and Storage Technology(FAST'11). San Jose, CA, USA: [s.n.], 2011.

[7] ATTEBURY G, BARANOVSKI A, BLOOM K, et al. Hadoop distributed file system for the grid[C]//Nuclear Science Symposium Conference Record (NSS/MIC). Orlando, FL: IEEE, 2009: 1056-1061.

[8] POTERAS C M, PETRISOR C, MOCANU M, et al. DCFMS: A chunk-based distributed file system for supporting multimedia communication[C]//2011 Federated Conference on Computer Science and Information Systems (FedCSIS). Szczecin: IEEE Press, 2011: 737-741.

[9] CHANDRASEKAR S, DAKSHINAMURTHY R, SESHAKUMAR P, et al. A novel indexing scheme for efficient handling of small files in hadoop distributed file system[C]//2013 International Conference on Computer Communication and Infromatics. Coimbatore, India: [s.n.], 2013.

[10] FU Song-ling, LIAO Xiang-ke, HE Li-gang, et al. FlatLFS: a lightweight file system for optimizing the performance of accessing massive small files[J]. Journal of National University of Defense Techonology, 2013, 35(2): 120-126.

编　辑　税　红