Hadoop 中处理海量小文件的方法①

李 旭, 李长云, 张清清, 胡淑新, 周玲芳

(湖南工业大学 计算机与通信学院, 株洲 412007)

摘 要: 针对 Hadoop 中提供底层存储的 HDFS 对处理海量小文件效率低下、严重影响性能的问题. 设计了一种小文件合并、索引和提取方案, 并与原始的 HDFS 以及 HAR 文件归档方案进行对比, 通过一系列实验表明, 本文的方案能有效减少 Namenode 内存占用, 提高 HDFS 的 I/O 性能.

关键词: Hadoop; HDFS; 小文件; HDFS 的 I/O 性能

Methods of Dealing With Massive Small Files in Hadoop

LI Xu, LI Chang-Yun, ZHANG Qing-Qing, HU Shu-Xin, ZHOU Ling-Fang

(Hunan University of Technology, Zhuzhou 412007, China)

Abstract: HDFS provides the underlying storage for Hadoop, however, the HDFS deals with massive small files inefficiently and decreases system performance seriously. To solve this problem, we designed a file merging, indexing and retrieval solution. Then through a series of experiments compared to the original HDFS and HAR solution, it can be shown that our scheme can effectively reduce the memory usage of Namenode and improve the I / O performance of HDFS.

Key words: Hadoop; HDFS; small files; I/O performance of HDFS

Hadoop[1]是目前十分流行的大数据分布式处理软件框架, 国内外著名的互联网公司如雅虎、百度、阿里巴巴、Facebook、Twitter 都在使用 Hadoop 做相关的大数据处理. HDFS[2]是 Hadoop 的分布式文件系统, 提供底层的数据存储服务, 对大文件的存储和高吞吐量的数据访问有着良好的性能. 然而互联网中数据增长最快的是海量的小文件, 如图片、网页和小文本等等, 随着小文件数目急剧增加, HDFS 存储和读取性能都会明显下降[3].

小文件是指那些远小于 HDFS 上默认 block 大小(64M)的文件, 这样的文件会给 Hadoop 的扩展性和性能带来严重问题. HDFS 集群以主从模式运行, 主要有两类节点: 一个 Namenode(即主节点)和多个 Datanode(即从节点) [4]. HDFS 中所有块、文件和目录都以对象的形式在 Namenode 内存中存储, 每个对象大约占 150 个字节内存空间. 假如文件数目达到上十亿, 那么 Namenode 就需占用将近 200G 的内存, 远超出目前硬件的承受能力, 严重制约了集群的扩展性. 另外, HDFS 最初是为流式访问大文件设计开发的, 如果访问大量小文件会造成大量的 Datanode 到其他 Datanode 的跳跃和搜索来取回文件, 这种文件访问方式效率低下. 同时系统对海量小文件的处理速度也远小于处理相同大小的大文件, 若每个小文件都占用一个 slot, 那么在任务的启动和释放上会花费大量时间, 明显降低系统的性能.

本文提出了一种通过文件合并在 HDFS 中存储文件的方案: 在文件写入到 HDFS 之前先对其大小进行判断: 如果是小文件就建立一个小文件队列, 通过将小文件合并成大文件来减少文件数目, 同时通过建立的相应文件索引并使用设计预读取策略, 实现小文件

① 基金项目:2013 年度科技部科技支撑计划(2013BAJ10B14－5)
收稿时间:2015-03-08;收到修改稿时间:2015-05-18

的高效访问.

1 概述

HDFS 采用主从架构, 由一个 NameNode 和多个 DataNode 组成. NameNode 是中心服务器, 负责管理文件系统的名字空间以及客户端的访问, DataNode 负责管理它所在节点上的存储. NameNode 是系统的主节点, 对系统的性能和稳定性有着至关重要的作用, 它主要负责执行文件系统的名字空间操作. 另外, HBase[5]是一个开源的基于列存储的非关系型分布式数据库(NoSQL), 它可以容错的存储海量稀疏数据, 其所有数据文件都存储在 HDFS 文件系统上, 主要包括两种文件类型: Hfile 和 HLog File. 核心组件 HRegionServer 主要负责响应用户 I/O 请求, HMaster 主要负责 Table 和 Region 的管理工作, Client 负责通过 RPC 来通信, Zookeeper 存储了各个 HRegionServer 的状态信息.

针对小文件问题存在一些针对特定的应用场景提出的解决方案, 郑丽洁[6]等根据小文本语料库的特点, 提出了 HSCS 存储策略, 对小文本的合并和检索算法进行设计, 满足海量小文本的存储需要. 赵晓永[7]等根据 MP3 文件包含的描述信息, 利用相关的归类算法并引入高效的扩展一级索引机制, 实现海量 MP3 文件高效存储. Liu[8]等根据 WebGIS 系统自身拥有的特点, 将存储着相邻地理位置信息的小文件合并为一个大文件, 然后为小文件建立相应的索引, 实现 WebGIS 系统文件高效的存取. Dong[9]等针对 Bluesky 系统中 PPT 课件存储的需求, 提出了一种将属于同一个课件的文件合并成为一个大文件的想法, 并引入一种预读取机制来满足小文件高效读取的需求, 包括索引文件的预取和数据文件的预取. 李林[10]等对海量图片存储的需求进行分析, 结合 HBase 数据库进行索引设计和图片文件名优化, 实现海量图片的高效存储和查询.

Hadoop 本身也提供了一些解决方案, 分别是: Hadoop Archive[11]、Sequence file[12]和 CombineFile InputFormat[13]. Hadoop Archive 是一个文件存档工具, 将多个小文件打包成一个 HAR 文件, 不仅可以降低 Namenode 内存的占用还可以实现对文件的透明访问, HAR 文件的结构如图 1 所示. Sequence file 由一系列的二进制键值对组成, 如果以小文件名为键, 文件内容为值, 可以将多个的小文件合并成一个大文件. CombineFileInputFormat 可以将多个文件合并成一个 split, 不过它需要设计实体类去实现.

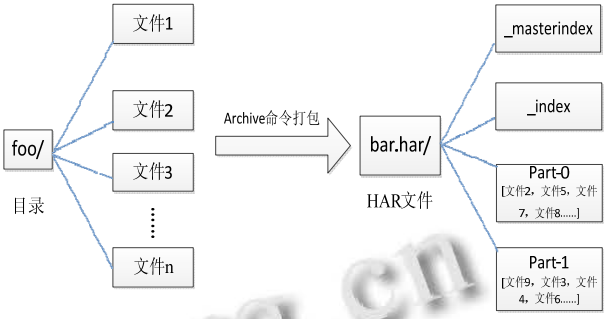

图 1 HAR 文件结构图

当前也存在一些企业级的解决方案, 如淘宝的 TFS: Taobao File System[14]和 Facebook 的 Haystack[15]都是一种类似 HDFS 的专门针对小文件的分布式存储系统, 用于公司中图片等小文件的高效存储, 公司利用自身技术实现对小文件针对性的处理并满足其网站存储和访问的需求.

2 小文件处理方案

2.1 整体文件处理方案

当有新文件请求写入 HDFS 时, 先对文件大小进行判断. 若非小文件, 就直接写入 HDFS; 否则就将文件交给小文件处理模块. 小文件处理模块主要功能是对小文件进行合并, 然后为每个合并后的文件进行唯一编号, 并将小文件和合并后文件的映射信息写入 Hbase 的文件映射表中. 接着为小文件建立相应的索引, 当合并后的文件大小达到规定的阈值就写入 HDFS, 需要读取小文件时, 可以根据 Hbase 中保存的映射信息、Namenode 中的数据块映射信息以及小文件的索引, 对小文件的信息进行预读取, 最后根据得到的信息从数据节点获取小文件的数据.

2.2 小文件合并和索引方案

HDFS 中小文件的主要问题是占用 Namenode 过多内存, 文件无论多小, 其所占用的元数据与大文件是相同的. 文件的元数据包括文件名、文件大小、文件权限、修改时间、文件在命名空间树中的位置等, 而块的元数据包括存在文件数据的块的列表以及这些块的位置. 文件合并技术可以使文件数目明显减少而 Namenode 只需要保存合并后文件的元数据.

文件合并操作是在有文件请求写入 HDFS 时由客

户端完成的, 如果合并后文件大小和默认块大小一致, 那么文件数量将会大大减少, 然而实际上几乎不可能恰好大小一致. 因此需要设置空白区域, 当新文件比空白区域大的时候, 将新文件放入下一个合并队列中. 本文设计的文件合并方案是按时间对小文件进行分类处理, 将日期、首个文件写入时间、最后一个文件时间组合成输出的大文件的名字, 例如: 201410061023#201410061026. 小文件和合并后文件的结构如下:

```
Struct SmallFile{
  String SF_Name;
  DateTime SF_Time;
  String SF_Data;
  Int SF_DelFlag;}
```

SF_Name 表示小文件的名字; SF_Time 表示小文件的创建时间; SF_Data 表示小文件的数据内容; SF_DelFlag 表示小文件的删除标记. 值为 1, 表示此文件已删除; 为 0, 表示此文件未被删除; 初值为 0.

```
Struct BigFile{
  String BF_Name;
  Int BF_Size;
  String BF_Data<SmallFile>; }
```

BF_Name 表示合并之后的大文件的名字(包含了创建时间等信息); BF_Size 表示合并后大文件的大小; BF_Data 表示大文件的内容, 是由多个小文件信息组成的.

文件合并算法如下:

输入: SmallFile [n] 输出: BigFile

```
1.  DateTime t1=getTime (SmallFile[0]);
2.  Open(BigFile[0]);
3.  int i=1;
4.  while (SmallFile[i].SF_Time > t1 && BF_Size < 64M) {
5.      write(SmallFile[i]);
6.      if (!SmallFile.File_DelFlag) {
7.        Write the rest of SmallFile[i];
8.      }end if
9.      i++;
10.   }end while
11.  DateTime t2=SmallFile[i-1].SF_Time;
12.  Close(BigFile[0]);
13.  BigFile.BF_Name= t1+"#"+t2;
14.  return 1;
```

步骤 1 读取小文件创建时间并赋值给 t1; 步骤 2 打开一个大文件; 步骤 3-10 当下一个小文件准备写入而且大文件的大小小于 64M 时写入小文件; 当小文件删除标记为 0, 执行合并操作, 将小文件全部写入大文件; 步骤 11 将最后一个写入大文件的小文件创建时间赋值给 t2; 步骤 12-13 写完关闭 BigFile,并将 t1#t2 作为文件的名字; 步骤 14 返回成功信息.

在文件数据合并的同时, 需要生成一个索引表放置在合并文件的开头来记录小文件的信息, 索引表包括小文件的偏移量、字节长度和文件名. 合并后的文件结构如图 2:

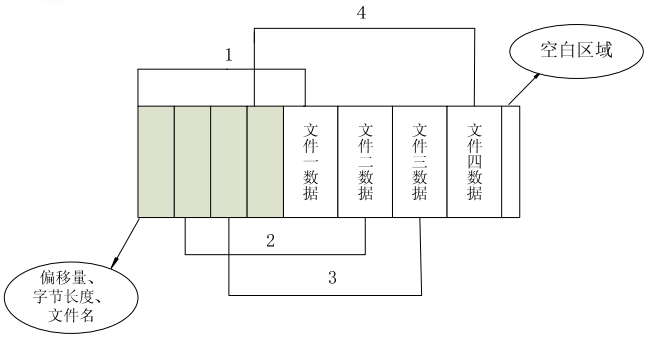

图 2 合并后的文件结构

根据小文件名称, 获取到实现所需文件及其信息的保存位置, 从而准确获取小文件数据. 除了需要 Namenode 提供的小文件数据所在的 Datanode 的位置以及块文件头的索引信息外, 小文件和合并后文件的映射信息也尤为关键.

2.3 HBase 映射表设计

本文将小文件和大文件的映射信息存储在 HBase 中, HBase 中的行是逻辑上的行, 物理模型上行是按列族(Colomn Family)分别存取的, 一个列族中的数据是通过 Colomn Qualifier 来具体描述每一列, 而 Rowkey 相当于关系型数据表中的主键.

本文设计将小文件 ID 作为表中的 Rowkey, 其值为小文件名的 MD5 值, 列族中包括三列分别为: 大文件 ID、小文件删除标记和小文件修改时间. 大文件 ID 是合并后的大文件名所对应的 MD5 值; 删除标记为 1 代表小文件已删除, 为 0 代表小文件是可以读写的; 修改时间是文件最后一次被改动的时间. HBase 的映射表结构如表 1 所示.

表 1　HBase 映射表结构

Rowkey (小文件 ID)	Time Stamp (时间戳)	Colomn Family(列族)		
		大文件 ID	删除标记	修改时间
小文件 1	t3	大文件 1	0	140506 132100
	t6	大文件 3	1	140503 132100
小文件 2	t1	大文件 1	1	140406 132100
	t3	大文件 4	0	140506 132600
.....	....	....	....	....
小文件 n	t1	大文件 n	0	140506 134800

2.4 小文件预读取和提取方案

文件合并只是减少了 Namenode 的内存占用, 并未提高小文件的读取性能. HDFS 采用的是“一次写入, 多次读取”的模式, 如果需要读取的小文件数目巨大, 会产生大量和频繁的请求而造成对 Namenode 的负担.

本文设计实现了预读取文件的元数据来解决这个问题, 当客户端试着读取一个小文件的时候, 与其同在一个合并文件中的其他文件元数据会一同从 Namenode 缓存到客户端, 当元数据在缓存中存在时就不需要再建立新的 RPC 请求, 使 Namenode 的请求数明显减少从而提高系统性能(图 3).

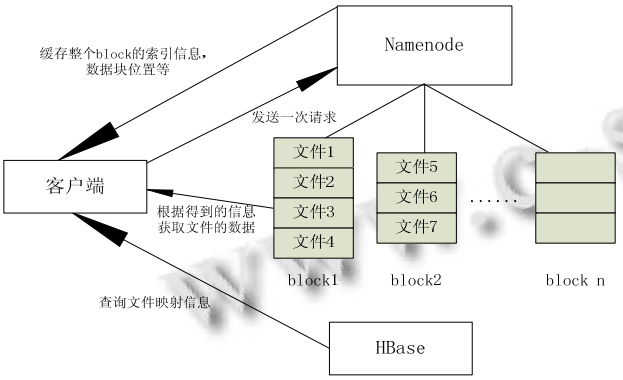

图 3　小文件信息预读取和提取示意图

当读取文件时, 根据小文件名、缓存到的信息以及 HBase 高效查询得到的映射信息, 可以快速从 Datanode 得到小文件对应的数据块的位置, 然后根据数据块上的偏移量和文件长度, 获取到小文件的数据, 最后发送给客户端. 由于没有将整个块传送到客户端, 网络的负担明显减少.

3　实验

本文的实验是由 4 台计算机搭建的 Hadoop 集群完成, 其中一台 Namenode, 四台 Datanode, 机器的配置为 Intel 酷睿双核 2.1GHz, 4G 内存, 320G 内存. 操作系统版本是 Ubuntu12.04, Hadoop 版本为 1.2.1stable, JDK 版本为 1.6. 以下所有实验都重复三次, 取平均结果.

3.1 内存占用测试

由于 Namenode 内存占用和 HDFS 性能息息相关, 本文首先测试内存的占用, 分别对原始的 HDFS、HAR 方案和本文方案三种情况进行实验. 测试的小文件分成 4 组, 每组的文件数目分别为 20000,40000, 60000, 80000, 所有的文件都小于 500KB, 包含图片、文档和网页等格式. 实验结果如图 4 所示.

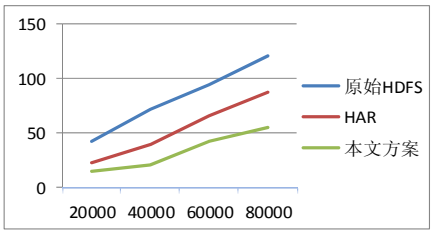

图 4　Namenode 内存占用对比

其中, 纵坐标轴表示文件占用的内存大小, 横坐标表示文件数目. 根据实验结果可以得到, 本文的方案在 Namenode 内存占用方面比原始的 HDFS 和 HAR 方案表现出了更好的性能. 与原始的 HDFS 相比, 降低了大约 49%的 Namenode 内存, 同时比 HAR 方案性能也有所提高, 使 Namenode 不再成为系统性能的瓶颈.

3.2 文件写入测试与读取测试

本次试验测试的小文件分为四组, 每组的文件数目分别为 2000,4000,6000,8000, 同样也对原始 HDFS、HAR 方案和本文方案三种情况分别进行. 其中本文方案的写入时间包括文件合并的时间以及合并后的文件写入 HDFS 的时间, 实验结果如图 5 所示. 其中纵坐标轴表示文件写入时间, 纵坐标表示文件数目. 随着小文件数的增加, 原始的 HDFS 的写入速度明显变差, 本文方案虽经过文件合并的时间, 但效率还是比较高的, 比原始的 HDFS 的文件写入速度提高了大约 38%, 比 HAR 方案提高了大约 16%, 而且随着文件数的增加,

写入速度并没有受到太大影响, 表现出了良好的性能.

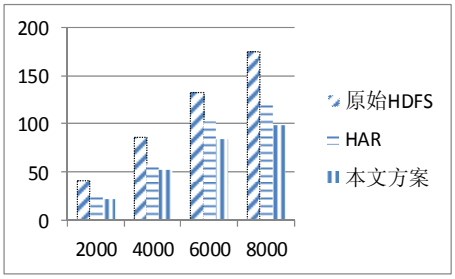

图 5 文件写入时间对比

文件读取的实验采用和文件写入测试相同的文件集, 分别从写入的四组文件中随机读取其中的 500 个文件, 然后分别记录采用三种不同方案时小文件读取的时间. 实验结果如图 6 所示. 其中纵坐标轴表示文件读取时间, 纵坐标表示文件数目. 采用本文方案可以高效的完成小文件的提取, 比其他两种情况花费的时间都要少, 比原始 HDFS 节约了大约 57%, 说明本文对文件的索引和预读取方案提高了文件的读取效率.

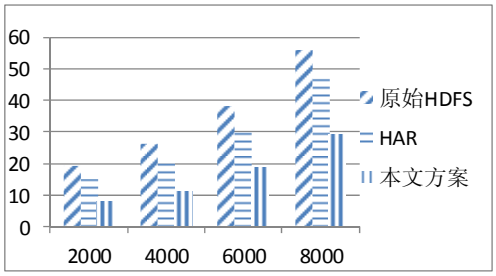

图 6 文件读取时间对比

4 结语

HDFS 的小文件问题严重影响系统的性能, 目前还没有一种系统的、统一的解决方案. 现有方案也有各种局限性, 同一种方案针对不用的应用场景表现的效果也不尽一致. 本文针对 HDFS 处理大量小文件时存在的不足, 借鉴经验提出了小文件合并方案, 明显减少了 Namenode 的内存占用和频繁的读写, 并可以通过索引方案、HBase 映射方案以及预读取策略高效的提取小文件, 从而提高 HDFS 的文件读写效率, 提高 I/O 性能. 未来将会在设计多个 Namenode 或者改变 HDFS 底层存储机制上做进一步研究.

参考文献

1 Hadoop official site.http://hadoop.apache.org, 2012.

2 HDFS official wiki. http://en.wikipedia.org/wiki/HDFS.

3 Small-Files-Problem. http://www.cloudera.com/blog/2009/02/the-small-files-problem/.

4 White T. 周敏奇,王晓玲,金澈清,钱卫宁译.Hadoop 权威指南.第 2 版.北京:清华大学出版社,2011.

5 George L. HBase: The Definitive Guide: Random Access to Your Planet-Size Data. O'Reilly, Ireland (2011).

6 郑丽洁.小文本语料库在 Hadoop 平台上的存储策略研究[硕士学位论文].武汉:华中师范大学,2014.

7 赵晓永,杨扬,孙莉莉,等.基于 Hadoop 的海量 MP3 文件存储架构研究.计算机应用,2012,32(6):1724–1726.

8 Liu XH, Han JZ, Zhong YQ, Han CD. Implementing WebGIS on Hadoop: a case study of improving small file I/O performance on HDFS. Proc. of the 2009 IEEE Conf. on Cluster Computing and Workshops. 2009. 1–8.

9 Dong B, Qiu J, Zheng QH, Zhong X, Li JW, Li Y. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files. In Proc. of IEEE SCC'2010. pp. 65–72.

10 李林.基于 hadoop 的海量图片存储模型的分析和设计[硕士学位论文].杭州:杭州电子科技大学,2011.

11 Hadoop archives. http://hadoop.apache.org/common/ docs/r0.20.2/hadoop_archive.

12 Sequence file. http://wiki.apache.org/hadoop/SequenceFile.

13 CombineFileInputFormat. http://hadoop.apoache.org/docs/stable/api/org/apache/hadoop/mapred/lib/CombineFileInputFormat.html.

14 Taobao File System, http://tfs.taobao.org/.

15 Beaver D, Kumar S, Li HC, et al. Finding a needle in haystack: facebook's photo storage. OSDI. 2010, 10: 1–8.