|  |  |
| --- | --- |
| 学号 | 2017202110098 |

**海量存储结课论文**

|  |
| --- |
| **在Hadoop分布式文件系统中**  **处理小文件的新方案** |

|  |  |
| --- | --- |
| 院（系）名 称 ： | 计算机学院 |
| 专 业 名 称 ： | 信息安全 |
| 学 生 姓 名 ： | 张 尧 |

二〇一七年十二月

摘 要

基于小组讲的论文[1]，在其引用的论文中，选择了[2]进行阅读。主要对[2]的内容进行理解，并将其与[1]中相关的内容进行比较与分析。[2]主要讨论了Hadoop分布式文件系统（HDFS），并探讨了HDFS在存取小文件上的存在的问题以及解决这个问题的现有工作。之后，[2]提出了自己的解决方案，并进行了相应的实验和评估。

**关键字：**Hadoop分布式文件系统，小文件

目 录

一、 简介 1

A. Hadoop分布式文件系统 1

B. 小文件问题 2

二、 文章主要内容 2

A. 解决方案 2

(1) 文件合并 3

(2) 文件映射 3

(3) 预取 4

(4) 文件提取 5

(5) 文件访问操作 5

B. 工作总结 7

三、 比较与分析 8

参考文献 9

# 简介

## Hadoop分布式文件系统

Hadoop是一个开源项目，它开发可靠和可扩展的分布式计算软件。Hadoop框架已被广泛应用于各种集群，构建大规模，高性能的系统。Hadoop体系结构由Hadoop分布式文件系统和编程模型MapReduce组成，用于在商用计算机集群上执行数据密集型计算。Hadoop集群通过简单地添加商品机器来扩展计算容量，存储容量和I / O带宽。

Hadoop分布式文件系统（HDFS）是Hadoop的旗舰文件系统组件。灵感来自专有的Google文件系统（GFS）HDFS的设计遵循一次写入和多次读取的模式。HDFS具有主从架构，一个主节点称为NameNode，多个从节点称为DataNode。NameNode管理HDFS中的元数据和文件系统配置数据。元数据保存在NameNode的主内存中，以确保在读取/写入请求时快速访问客户端。DataNode按照NameNode的指示，在HDFS中存储和服务对文件的读/写请求。存储在HDFS中的文件按照配置被复制到任意数量的DataNode中，以确保可靠性和数据可用性。这些副本分布在群集中以确保快速计算。

Hadoop分布式文件系统提供对群集中文件的全局访问。NameNode是一个集中式的单一服务器，负责维护HDFS文件的元数据。它还维护配置数据，例如，一个名为复制因子的文件的每个块的副本计数，块的大小以及HDFS的其他此类参数。NameNode维护文件系统中文件的目录树结构。DataNodes代表客户端以块的形式存储文件。每个块都作为单独的文件存储在节点的本地文件系统中。当DataNode抽象出底层文件系统的细节时，所有节点的特征不需要相同。DataNode负责根据NameNode的请求存储，检索和删除块。HDFS中的文件被分成块，默认块大小为64 MB，每个块被复制并存储在多个DataNode中。

NameNode维护存储在HDFS中的每个文件的元数据在其主内存中。这包括存储的文件名，每个文件的相应块和容纳这些块的DataNode之间的映射。因此，客户端创建，写入，读取或删除文件的每个请求都通过NameNode。使用存储的元数据，NameNode必须将来自客户端的每个请求指向适当的DataNode集合。然后，客户端直接与DataNode进行通信以执行文件操作。

单个NameNode将元数据存储在其主内存中，成为处理大量小文件的瓶颈。小文件是任何文件，其大小明显小于HDFS的块大小。这个问题被称为小文件问题，可以防止许多潜在的应用程序使用Hadoop框架的好处。

## 小文件问题

将大量小文件存储到HDFS中会成为存储在NameNode中的元数据的内存使用量的开销。在这种情况下，当一个应用程序访问这些小文件的较大集合时，单个NameNode就成为处理元数据请求的瓶颈。此外，NameNode中的主内存大小限制了可以存储到HDFS中的文件的数量。

NameNode将整个元数据存储在主内存中，以便更快更高效地处理客户端请求。元数据存储在文件的每个块中。当大小大于块大小的文件被存储时，存储的元数据的量由文件大小来确定。而当一个大量的小文件，每个都小于块大小，每个文件形成一个块，因此，相应的元数据存储相当高。例如，假设文件的每个块的内存中的元数据占用大约150个字节。因此，对于分为16个64MB块的1 GB文件，将存储2.4 KB的元数据。而对于大小为100KB（共1GB）的10,500个文件，大约有1.5MB的元数据被存储。因此，大量的小文件在文件系统上占用较少的空间，但占用了大量的NameNode的主存。这导致不公平地使用可用的群集空间。此外，访问大量这些文件导致NameNode出现瓶颈。这阻止了HDFS在各种应用中的最佳使用。

# 文章主要内容

## 解决方案

[2]提出的解决方案扩展了HDFS，并被命名为Extended HDFS。虽然这个解决方案受到了Dong等人提出的解决方案的影响，但EHDFS提供了一个改进的索引机制和索引信息的预取。EHDFS有四项技术在提高HDFS处理小文件的效率方面起着重要的作用。他们是文件合并，文件映射，预取和文件提取。

### 文件合并

HDFS中小文件的主要问题是NameNode用来管理这些文件的内存量。HDFS不会区分小文件和大文件，因此无论文件大小如何，都会存储相同数量的元数据。NameNode维护HDFS中每个文件的两种类型的元数据，即文件元数据和块元数据。

文件元数据包括有关文件的信息，例如文件名称，文件在名称空间树中的位置，文件大小，修改时间，访问时间，所有权详细信息和文件权限。块元数据包含有关的保存文件数据的块列表以及这些块的位置。

文件合并技术减少了NameNode为小文件维护的文件元数据和块元数据。它利用了这样一个事实，即当用户将它们组合在一起并将其上传到HDFS时，大部分文件元数据对于所有文件保持不变。文件合并可确保NameNode仅为组合文件维护元数据，而不是为其中存在的所有小文件维护元数据。组成文件的名称及其块信息作为NameNode中特殊数据结构的一部分进行维护。文件合并过程在客户端执行，同时使用EHDFS创建文件。在创建组合文件时，客户端指定小文件的名称和与每个文件关联的数据。这些数据在客户端被缓存，直到没有超过HDFS块大小的情况下不能添加更多的文件数据。这确保了没有小文件被分割。

与文件数据一起，索引表也被放置在每个块的开始处。该表包含作为该块一部分的每个小文件的条目。每个表项都是一个（偏移量，长度）对。对于块中的第i个文件，索引表中的第i个条目指定小文件的第一个字节从块开始的偏移量和小文件的长度（以字节为单位）。索引表中的信息可用于识别指向相应文件开始和结束的偏移量。所产生的块的结构称为扩展块信息，扩展块表示组合文件的一部分。这些块与HDFS DataNode中的其他块一起存储为其他常规文件块。

### 文件映射

文件映射是将小文件名映射到包含此文件的组合文件的块的过程。这是由NameNode执行的。文件映射技术在用户想要从组合文件读取小文件时起作用。用户必须在启动读取操作时明确指定组合文件和小文件的名称。向NameNode发送一个请求以及这两个文件名，以获取所需小文件的位置。

NameNode为每个组合文件维护一个名为Constituent FileMap的数据结构。它包含一个小文件名和包含这个小文件的组合文件的逻辑块号之间的映射。除了逻辑块号和DataNode信息之外，NameNode还提供“索引条目号”。这个“索引条目号”指定存储在块开始处的索引表中的哪个条目对应于所请求的小文件，从而避免线性搜索。

文件名被散列到组成文件映射中。与文件的名称一起，维护关于块中文件排序的信息，是使用“索引条目号”字段完成的。“索引条目号”值为0表示这是给定块中的第一个文件，并且索引表中的相应条目存在于从块开始的零偏移处。索引条目号在文件合并过程中以与块中的每个文件类似的方式分配。

与Dong等人使用的映射技术相比，使用的映射技术更具可扩展性。它也不会将任何文件保存为逻辑块号映射作为文件名的一部分。这允许解决方案独立于域。

### 预取

文件合并技术只会减少NameNode中的元数据占用空间。它不会提高读取操作的性能。如前所述，HDFS被设计为“一次写入，多次读取”的模式。提高文件读取操作的速度比提高写入操作的速度更重要。

当从HDFS读取文件时，最初将请求发送到NameNode以获取与该文件关联的元数据。此元数据提供有关包含该文件的块列表以及包含这些块的DataNode的信息。这对于阅读文件的内容至关重要。与NameNode的通信通过远程过程调用（RPC）进行。因此，对于正在打开的每个文件，都必须启动一个RPC请求来获取元数据。这在NameNode出现问题时，访问大量的小文件。请求的数量和它们生成的频率，而在阅读小文件时，会给NameNode带来相当大的负担，使其成为系统中的瓶颈。

所提出的EHDFS通过提供用于预取文件元数据的框架克服了这个瓶颈。每当客户端尝试读取一个小文件（作为组合文件的一部分）时，就会从NameNode中预取相同块中的其他小文件的元数据，即所请求文件的元数据。这使得客户端可能会以类似于顺序访问的方式访问组合文件中的小文件。预取的数据然后被缓存在客户端。只要文件元数据存在于缓存中，客户端就不需要发起对NameNode的RPC请求。这确保了发送给NameNode的请求数量大大减少，从而提高了读取操作的性能。

### 文件提取

文件提取涉及从块中提取所需文件内容的过程。这个操作由DataNode执行。读取文件时，客户端指定小文件的名称和组合文件的名称。这个信息用于获取保存文件的块，保存块的DataNode和NameNode中的“索引条目号”。然后将获得的“索引条目号”发送到具有该块的数据节点。DataNode然后使用这个值并且寻找位于块开始处的索引表中的期望条目。索引表中的条目包含文件数据从块开始的偏移量以及文件数据的长度。DataNode然后寻找所需的偏移量并读取请求的文件数据并将其发送到客户端。此操作大大减少了网络上的负载，因为整个块数据不会发送回客户端。

### 文件访问操作

①写操作：只要使用EHDFS创建组合文件，就会启动写操作。用户使用客户端提供的ExtendedHDFS模块调用创建操作。这通过RPC向NameNode发送一个请求，用于创建和初始化存储组合文件所需的数据结构。NameNode创建一个特殊的inode来指示正在创建的文件是一个组合文件并将其保存到名称空间树中。NameNode初始化组成文件映射。该映射保存为表示文件的inode的一部分，并保存表示构成组合文件一部分的小文件的条目。

然后为客户端提供输出流和一些辅助方法，帮助将文件数据与组合文件中的条目相关联。写入输出的数据流最初缓存在客户端执行文件合并。缓冲区能够保存块的数据。为其数据存在于缓冲区中的所有文件构建索引表。该表包含文件合并过程中解释的每个文件的元组。缓冲区的内容被附加到索引表，然后发送到DataNode。计算表中每个条目的偏移量，使得它还包含块开始处的表的大小。

此索引表的目的是快速识别块中所需文件的开始，而不需要扫描整个块。通过在NameNode中存储与该逻辑块号一致的索引号，可以使得该表中存在与所需文件相对应的元组的索引号变得更快。

在成功将块数据写入DataNodes后，客户端向NameNode发送一个更新，指定写入最近块的文件名列表以及每个文件的索引条目编号。这由NameNode用来构造组成文件映射。重复此过程，直到客户端关闭正在创建的组合文件。

②读操作：用户通过直接访问组合文件中的小文件来启动读操作。文件系统中的文件路径通过将组合文件名称指定为包含小文件的文件夹的名称来表示。换句话说，如果“xyz”是组合文件的名称，“file1”是小文件的名称，则 用 户 应 该 将 “file1”的路径指定为“location-ofcombined-file / xyz / file1”。

客户端模块从给定路径提取组合的文件名和小文件名，并向NameNode请求与小文件对应的元数据。这是一个基于RPC的请求，NameNode通过在为指定的组合文件存储的组成文件映射中查找小文件名作出响应。逻辑块号码和索引条目号码作为响应，连同块位置（DataNode列表）一起被发送回客户端。

客户端还可以在打开小文件时将预取计数作为附加参数。该选项指示NameNode发送所请求文件周围文件的元数据以及对当前请求的响应。计数表示必须预取元数据的文件数。NameNode保存组成文件映射中块中文件的排序。因此，只要客户计划顺序访问文件进行处理，就可以指定一个较大的预取计数（受高速缓存容量的限制）。还可以使用适当的选项为当前文件之前的文件预取元数据。

这些预取的元数据条目存储在客户端缓存中。只要客户端正在打开一个小文件，就会检查该缓存，并且如果元数据已经存在于缓存中，则不会将请求发送到NameNode。这大大提高了读取操作的响应时间。它还减少了发送到NameNode的请求数量，从而减少了NameNode上的负载，同时提高了连续读取操作的速度。

在从缓存或从NameNode获得所请求文件的元数据之后，在客户端和持有块的DataNode之间建立流连接。在这个阶段，被请求文件的索引条目号被发送到DataNode，DataNode读取块的索引表中指定的条目。它使用此偏移量和长度值来进行在此连接上进行的未来读取操作。

客户端可以通过指定相对于文件开头的偏移量的文件读取，即文件中的第一个字节与文件开头的偏移量为零。然后通过将该偏移量添加到从DataNode索引表获得的值中，将该偏移量转换为块中的等效偏移量。然后将读操作委托给现有的原始HDFS API，允许从指定的偏移量开始读取块。

## 工作总结

将逻辑块号保持为文件名的一部分的技术是不可缩放的，并且不能被应用于任何任意的小文件集。这可以通过维护名称节点中小文件的名称和组合文件名之间的映射来解决。在他们的文件中，Dong等人提供了这种方法的概述。

EHDFS更进一步，它不仅在NameNode中维护这样的映射，而且还维护块中文件的排序。这使得预取块中存在的下一个文件的元数据变得更容易。在按顺序访问组合文件中的文件时，可以使用此属性来减少对NameNode的元数据请求数。

EHDFS中NameNode的内存使用量明显小于不使用文件合并技术处理小文件问题的普通HDFS的NameNode内存使用率。但是，它比Dong等人提出的解决方案中的内存使用率要高。内存使用量的增加可以归因于Dong等人提出的解决方案。不在NameNode中存储任何特殊的数据结构，而新的解决方案为每个组合文件维护一个组成文件映射。内存使用量的这种增加是合理的，因为EHDFS不是域特定的，可以应用于任何一组小文件。

HDFS最初是为存储大型文件而设计的。当用于存储大量小文件时，I / O性能成为瓶颈。另外，NameNode主内存中的元数据占用空间迅速增加。所提出的EHDFS的设计和实现方式是可以将大量的小文件合并成单个合并文件，并且还提供了用于为特定数量的文件预取元数据的框架。从执行的性能分析可以得出结论，使用EHDFS可以提高访问小文件的效率并减少NameNode主内存中的元数据占用空间，对应于小文件。EHDFS通过为小文件提供更高效的元数据管理，可以更好地利用HDFS资源。对于100,000个小文件，内存使用量减少了16％。与原始HDFS相比，写入时间减少了82％，读取时间减少了28％（带预取）。

至于未来的工作，这个解决方案可以进一步改进，以提供更先进的预取框架。这个框架应该支持更好的预取文件关联分析和更好的文件合并过程，并考虑到这个文件相关信息。追加操作可以由EHDFS提供，将文件添加到现有的组合文件中。这个解决方案也可以移植到使用分布式名称空间管理的HDFS的新版本Federated HDFS。

# 比较与分析

[1]中提出的CFFS复合文件系统首先分配一个索引节点node-C，然后将各个子文件的数据部分连接起来作为复合文件的数据，各个文件的属性信息统一加到该节点的属性信息中。可以看到这与[2]中提出的文件合并、文件映射的想法是一样的。不同的地方在于，[2]中EHDFS对于文件的合并没有提出特别的算法，只是线性的堆叠。因为每个文件的使用权重不同，在预取的时候，仍然会造成较多的I/O冗余。

而CFFS中则提出了多种文件复合的方法，包括：1）基于目录的合并，将一个目录中的所有文件组合成一个复合文件，但不包含它本来的子目录；2）基于嵌入式参考的合并，根据文件中嵌入的文件引用来合并成复合文件；3）基于频率的合并，比如使用apriori算法，来确定谁和谁合并。

总的来说，CFFS文件系统是在EHDFS上发展来的，核心思想是一直的。CFFS在文件合并算法上进行了更深入的研究，使得文件系统的吞吐率有了极大提高，同时降低了系统的响应时延。

参考文献

[1] Zhang S, Catanese H, Wang A I A. The Composite-file File System: Decoupling the One-to-One Mapping of Files and Metadata for Better Performance[C]//*FAST. 2016: 15-22.*

[2] Chandrasekar S, Dakshinamurthy R, Seshakumar PG, Prabavathy B, Chitra B. A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System. *Proceedings of 2013 International Conference on Computer Communication and Information, 2013.*