基于 HDFS 海量小文件读取的研究与设计

彭建烽
广东技术师范大学，广东 广州 510665

DOI: 10.61369/TACS.2025060043

摘　　要： 基于 HDFS 存储海量小文件读取效率不高的问题，对 HDFS 存储的海量小文件读取进行优化。通过引入新型并行处理框架 Spark，对系统相关的小文件进行合并，并为合并后的文件建立索引，进而提升海量小文件读取效率。

关 键 词： HDFS；海量小文件；Spark；合并；索引

Research and Design on Reading Massive Small Files Based on HDFS

Peng Jianfeng
Guangdong Polytechnic Normal University, Guangzhou, Guangdong 510665

Abstract: Based on the problem of low efficiency in reading massive small files stored in HDFS, optimize the reading of massive small files stored in HDFS. By introducing the new parallel processing framework Spark, small files related to the system are merged and indexed, thereby improving the efficiency of reading massive small files.

Keywords: HDFS; **massive small files; Spark; merge; index**

引言

随着大数据时代的到来，各行各业产生的数据量指数增长。在这些数据中，海量小文件的存储与读取问题日益严重。以电商行业为例，每一笔交易记录、每一次用户浏览行为数据等，通常以小文件形式存储。这些小文件数量庞大，如一个中等规模电商平台每天产生的交易记录小文件可能数以百万计。

Hadoop 分布式数据存储和处理框架凭借其高效、可靠、高容错等优点，渐渐成为了炙手可热的海量数据存储和处理工具 .Hadoop 分布式文件系统（HDFS）是 Hadoop 的核心部分，它所具有的两类节点以管理者 - 工作者的模式运行，即单一 NameNode（管理者）和若干个 DataNode（工作者）。NameNode 负责管理文件系统的命名空间 .DataNode 作为系统的工作节点，它们根据需要存储并检索数据块（受客户端或 NameNode 调度），并且定期向 NameNode 发送它们所存储的块的列表 [1]。 然而 HDFS 存储海量小文件面临以下两个主要问题。

NameNode 负载过高：HDFS 中的 NameNode 负责管理所有文件的元数据，包括文件名、文件大小、文件所在的 DataNode 等信息。每个小文件在 NameNode 中都会占用一定的内存空间。例如，若每个小文件的元数据平均占用 1KB 内存，100 万个小文件就会占用近 1GB 内存。同时，NameNode 还需要处理创建、删除文件、查询文件等大量元数据操作，这使得 NameNode 负荷过高，进而导致整套系统性能下降。在实际应用中，当小文件数量超过一定阈值时，NameNode 的响应时间会显著延长，严重影响系统的正常运行。

小文件读取效率低下：常规的 MapReduce 任务执行模式难以适应小文件的特性 [2]。小文件的频繁磁盘 I/O 操作是导致读取效率低下的重要原因之一。由于小文件数据量小，每次读取都需要进行磁盘寻址等操作，这会产生大量的开销。例如，对比处理 1 个 1GB 和 1000 个 100KB 的文件，处理同样大小的小文件所用时间将远多于处理单个大文件的时间 [3]。此外，任务分配不合理也使得整体处理速度受限。在传统的 MapReduce 任务分配中，无法充分考虑节点的负载情况和小文件的分布特点，导致部分节点任务过重，而部分节点资源闲置，无法满足日益增长的实时性业务需求。

一、系统整体方案设计

针对 NameNode 节点内存压力过大、读取海量小文件效率不高等问题 [4]，构建了一套紧密耦合的 Spark-Hadoop 混合架构，在数据存取等方面实现两者的深度融合。

在 Hadoop 中引入了 Spark 分布式计算平台框架，通过 Spark 中的实时计算引擎，对 HDFS 中的文件进行实时扫描，合并同类型的小文件为大数据文件，并在 MongoDB 中记录文件索引信息，降低了 NameNode 节点的内存压力的同时提高 HDFS 的读取海量小文件效率。依据文件类型进行合并，摒弃了现有仅仅依据文件大小进行小文件合并的大文件算法，而忽略了文件自身的特性，导致合并后文件在读取时效率不高的缺陷。通过文件类

型合并小文件，可以提升后续读取小文件的效率[5]。

（一）系统整体方案设计

系统架构如下图1所示：

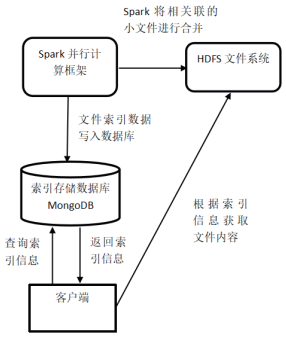

图1 系统架构图

Spark 与 Hadoop 的架构图如图1 所示。通过这种架构可以将 HDFS 上的所有同类型的小文件合并为若干个 HDFS 数据块的大小，从而减少由于文件数量过多而导致 NameNode 的内存消耗急剧增加的情况发生。HDFS 上的文件经过合并之后，重新建立了索引。新索引确保文件正确读取且读取效率快。文件经过合并后，还提供一套可靠的文件读写服务便于快速读取文件等操作。

（二）文件合并策略

文件打开与状态获取：在 HDFS 上通过命令将同类型文件筛选出来后，当待合并小文件总大小达到 256MB 后，即可开始对文件进行合并。通过 Spark 能够获取文件路径，文件的字节数、文件名及文件权限等状态信息。通过 Hadoop 的文件操作 API 实现文件的打开和状态获取，确保操作的安全性和稳定性。

字节数计算：可合并的字节数目是根据文件字节数量、文件路径以及 HDFS 所设置的块大小以及目前合并后文件大小之间的关联而计算出来的。合并的字节数需满足以下关系：Lhead = 22+Lpath 和 Lwrite = Min(Lleft−Lhead, Lfile+Lhead)。其中，Lpath 为文件路径的字节数，Lhead 为数据头的字节数，Lleft 是合并文件后剩余可合并的字节数，Lfile 为文件字节数，Lwrite 为可合并的字节总数。通过精确计算可合并字节数，避免了数据溢出和合并文件过大的问题，提高了合并效率和文件管理的规范性。

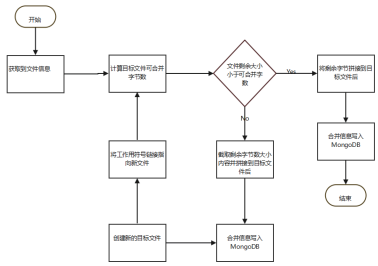

图2 文件合并流程

数据读取：得到可以合并的字节数后，从文件的当前位置读取 Lwrite 个字节。将所读取到的数据内容和数据头拼接，形成完整的文件单元。文件单元头包括2 字节的文件单元头（记录到文件路径后的总字节数）、4字节的数据长度（记录当前单元所记录的数据长度）、4 字节的数据偏移量（记录该数据在原文件的偏移量）、N 字节的文件路径（记录源文件的绝对路径）。通过对文件单元头结构的合理设计，保证了合并后的文件数据的完整性，具有可追溯性。

数据写入与循环处理：将文件单元写入到目标文件。如果源文件未完成合并，则继续执行上述步骤，直至源文件合并完成。当 Lleft<=Lhead 时，则需要创建新的合并文件来存储新的数据。通过这样的循环处理方式，可以高效地完成多个小文件的合并操作，减少文件数量，减少 NameNode 的负载。在一个合并文件中，各个文件单元是相互独立的，而合并后文件中的文件单元数量则取决于实际合并的文件数量以及 HDFS 分布式文件系统所规定的数据块大小（一般默认是256MB）。文件合并流程如下图2所示，并后的文件结构如下图3所示。

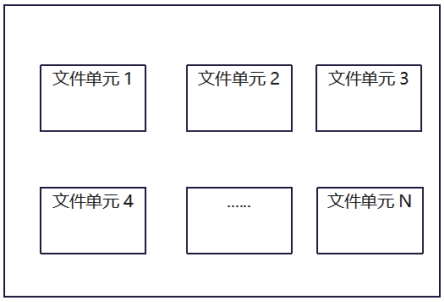

图3 合并后文件内数据结构

（三）文件索引设计

合并 HDFS 上的文件后，需要重新建立一个索引。因为文件进行合并后，源文件将会被删除，因此需要对合并后的文件建立索引。在数据库 MongoDB 中建立文件索引表，表结构包括文件路径（作为主键）、文件大小、权限属性、所有者、用户组、最后修改时间、创建时间以及合并后文件列表等字段。其中，作为唯一标识文件的主键为文件路径；文件基本属性记录文件的字段，如文件大小，权限属性，拥有者，用户群，最后修改时间，创建时间等。此外，与传统的文件索引方式相比，本索引设计增加了合并后文件列表字段，能够详细记录原文件经过合并后在不同合并文件中的分布信息。可将一份源文件可被合并成多个不同的大文件，其偏移量、数据长度等信息可通过合并后的文件列表字段准确记录在各合并文件中，便于后续读取文件。

文件索引数据结构如下表1所示。

表1 文件索引数据结构

字段名	字段属性
文件路径	主键
文件大小	
权限属性	
所有者	
用户组	
最后修改时间	
创建时间	
合并后文件列表	

（四）文件存储过程

在 HDFS 文件系统中用系统命令上传本地文件。进入主机后

台，在 HDFS 文件系统使用相关命令上传已准备好的文件。程序会实时扫描需要合并的队列，并将队列中待合并的文件合并。完成上传合并步骤后，检查文件实时合并的结果。连接 MongoDB 数据库，使用 命令行方式，查询上传的文件索引。

（五）文件读取流程

当文件在文件索引的数据库中能够检索到，则说明该文件已经经过了合并，此时需要从索引中获取到合并信息，并根据合并信息提取合并后文件中的数据后，将数据进行整合返回给调用者。具体流程为：根据用户访问的文件路径，到索引存储数据库中查询，若查询到文件信息，获取文件路径、偏移量、长度列表信息，根据列表信息提取数据，将数据按顺序拼接成一个文件，最后返回文件；若未查询到文件信息，则结束操作。

如果在文件索引数据库中能够检索到文件，这表明文件已经完成了合并。此时，需要从索引中提取合并的相关信息，并根据这些信息从合并后的文件中提取数据，将数据整合后返回给调用者。具体流程如下：首先，根据用户访问的文件路径在索引存储数据库中进行查询。如果查询到文件信息，接着获取文件路径、偏移量和长度的列表信息，然后根据这些列表信息提取所需的数据。将数据依次拼接成一个文件，并最终返回该文件。如果未找到文件信息，则终止操作。

二、实验与分析

（一）实验环境

本文通过部署 Hadoop 集群，对海量小文件的存取能力进行测试，以验证优化方案。Hadoop 集群共设置 9 个节点，每个节点的配置为：四核 Intel CoreTM CPU，主频3.6GHz，内存8GB，1TB 硬盘空间 . 其中一台机器作为 NameNode，其余八台作为 DataNode。每台节点安装的操作系统为 Ubuntu ubuntu16.04 LTS，Hadoop 版本为 Hadoop-3.3.2。在系统中存储20万个小文件，数据类型为 txt 格式文档、jpg 格式图片、mp3音频文件。

Spark 集群安装与配置：在已有 Hadoop 集群基础上，安装并配置 Spark 集群。确保 Spark 集群与 Hadoop 集群之间的通信链路畅通，通过配置网络参数和通信协议，实现数据交互顺畅。

MongoDB 数据库集群的安装与配置：该集群用于存储 HDFS 中文件的相关信息，包括文件的原始路径、文件的原始大小以及文件的权限信息。还有合并后的文件路径、文件大小以及文件在合并块中的偏移量等相关信息。用户在提取文件时，可通过 MongoDB 数据库中记录的文件信息找到待提取的文件的基本信息。

（二）实验分析

实验1中对系统中的小文件进行了4 次随机读取。实验结果如图4所示，优化后的 HDFS 对比原 HDFS 读取效率有了很大的提升，原因是对合并后的文件建立索引，随机访问的检索速度变快。

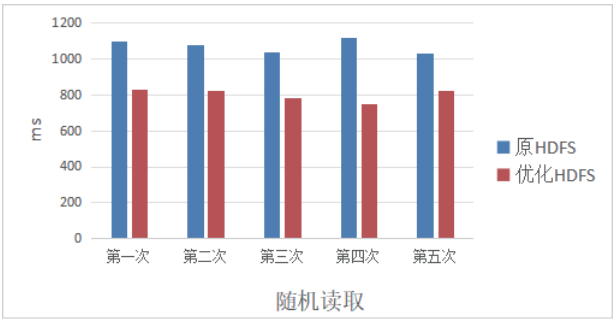

图4 随机读取

实验2分别顺序读取 5个、10个、15个、20个相关联的小文件，实验结果如下图5所示。优化的 HDFS 读取文件时间明显比原 HDFS 少。根据数据空间局部性（如果一个存储单元被访问，那么其附近的存储单元很可能在不久的将来也会被访问），合并后的相关联的小文件存储的位置相邻，那么读取合并后的文件集合的速度是快于原先分散存储读取文件的速度。

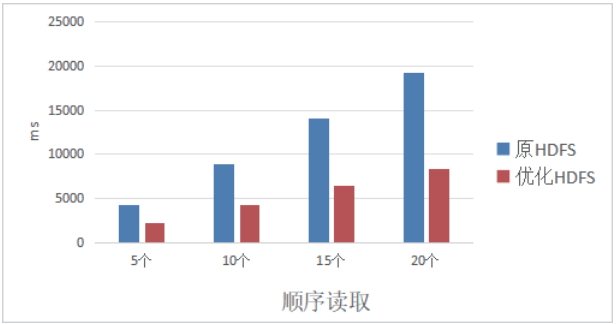

图5 顺序读取

三、结语

本文基于 Hadoop 的海量小文件的存取问题，引入新型并行处理框架 Spark，对系统相关的小文件进行合并，并为合并后的文件设计索引。该方案有效解决了 NameNode 内存压力过大和小文件的读取效率慢的问题，进而为海量小文件在 Hadoop 上的存储提供一些有价值的参考。

参考文献

[1] 刘超 . 基于云环境的海事局船检平台的设计与实现 [D]. 桂林理工大学 ,2020.DOI:10.27050/d.cnki.gglgc.2020.000547.

[2] 李文武，张建锋，王景林 . 基于 EHDFS 的海量小文件存储与检索方法 [J]. 计算机工程与设计，2022(002):043.

[3] 田峰 . 基于 HDFS 的海量小文件存储系统的研究与实现 [D]. 西安电子科技大学 ,2021.

[4] 张祥俊，伍卫国 . 基于 FastDFS 的数字媒体系统设计与实现技术研究 [J]. 计算机技术与发展，2019，29(5):6.

[5] 高朝艳，鹿虹，黄娟，等 . 基于 HDFS 的小文件存储技术研究 [J]. 电信技术研究，2020(3):10-15.

