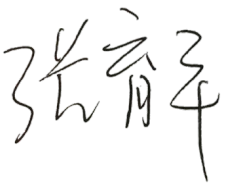
中图分类号：TP311　　论文编号：1028716 19-SZ005
学科分类号：085211

硕士学位论文

基于 HDFS 光盘库的磁盘缓存系统的设计与实现

研究生姓名　　王子炫
专 业 类 别　　工程硕士
专 业 领 域　　计算机技术
指 导 教 师　　张育平　副教授



南京航空航天大学
研究生院 计算机科学与技术学院

二〇一九年三月

Nanjing University of Aeronautics and Astronautics
The Graduate School
College of Computer Science and Technology

Design and Implementation of Disk Cache System Based on HDFS Optical Jukebox

A Thesis in

Computer technology

by

Wang Zixuan

Advised by

Prof. Zhang Yuping

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Engineering

March, 2019

承诺书

本人声明所呈交的硕士学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京航空航天大学或其他教育机构的学位或证书而使用过的材料。

本人授权南京航空航天大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本承诺书）

作者签名：王子松

日　　期：2014.3

摘 要

随着大数据时代的到来，全球每天产生的数据总量高达 PB 级，数据总量规模也越来越大。但是这些数据只有很少一部分经常使用，大部分数据在很长一段时间内都不会被访问。若是将所有的数据都存储到磁盘阵列中，将会带来高昂的存储成本和数据管理成本。随着光盘技术的发展，由于其单位存储成本低、容量大、安全性高、能耗低等优点，被应用到各种冷数据备份系统中。目前市场上出现一种基于光盘库的 Hadoop 分布式文件系统（HDFS 光盘库），HDFS 光盘库与传统的光盘库相比，在系统容量和数据传输速度方面都得到了很大的提升，但是仍然和磁盘之间存在很大差距。本课题正是为解决 HDFS 光盘库和磁盘存储设备之间的差距而进行的研究。

论文首先研究 HDFS 光盘库系统的结构特点，以及 HDFS 文件系统中小文件存储优化方案，针对 HDFS 光盘库内合并的小文件之间关联性较低的问题，本文提出了一种基于文件名的标签分类算法，并在虚拟存储模块内根据文件标签信息设计小文件合并策略。然后研究国内外缓存替换算法和预取技术，结合文件标签信息和系统内调度对象，提出了一种基于文件标签的 LB-LRU 算法（Label Based Least Recently Used），该算法有效地提高了磁盘缓存系统的缓存命中率，在 Cache 模块内设置文件预取策略提高磁盘缓存系统的缓存命中率。最后针对传统 HDFS 光盘和加入磁盘缓存系统的 HDFS 光盘在文件读写性能和 NameNode 内存消耗进行性能测试，同时对磁盘缓存系统内标签分类算法和 LB-LRU 算法的性能进行测试，测试结果表明磁盘缓存系统可有效的提高文件读写能力，降低 HDFS 光盘库的内存消耗。

关键词：小文件，标签，文件预取，缓存替换算法，关联性，光盘库

ABSTRACT

With the arrival of big data era, the total daily amount of data generated worldwide reaches to PB level accompanying with an increasing amount of total data. However, only a few data is frequently used and most will not be accessed for a long time. If all data are stored in the disk array, it will bring high storage cost and data management cost. As development of optical disk technology expands, it has been applied to various cold data backup systems, in virtue of its low storage cost, large capacity, high security and low energy consumption etc. At present, a Hadoop distributed file system (HDFS optical jukebox) based on optical jukebox appears in the market. Compared with traditional optical jukebox, HDFS optical jukebox has been greatly improved in system capacity and data transmission speed, but there still are a huge gap between HDFS and disk. Therefore, this paper aims to study how to solve the gap between HDFS optical jukebox and disk storage devices.

At first, the structure characteristics of HDFS optical jukebox system and the optimization scheme of small file storage in HDFS file system are studied. In response to the problem of low correlation between merged small files in HDFS optical jukebox, this paper comes up with a label classification algorithm based on file names and designs small files merge strategy according to files label information in the virtual storage module. Then, this paper studies the research status of cache replacement algorithm and prefetching technology at home and abroad. Combined with file label information and scheduling objects in the system, a file label-based LB-LRU (Label Based Least Current Used) algorithm is proposed to boost the cache hit rate of disk cache system, and file prefetching strategy is set in cache module to further raise the cache hit rate. Finally, the paper compares the test performance of traditional HDFS disk with HDFS disk which added to disk buffer system in file reading, writing and memory consumption of NameNode, and tests the performance of label classification algorithm and LB-LRU algorithm in disk buffer system. To sum up, according to the experimental results, the disk buffer system can effectively improve the file reading and writing ability and reduce the memory consumption of HDFS optical jukebox.

Keywords: Small Files, Label, File Prefetch, Cache Replacement Algorithms, Correlation, Optical Jukebox

目 录

图表清单

注释表

$P_{i,j}$	j 关键词内 i 标签的概率值	$n_{i,j}$	j 关键词内 i 标签的 Count 值
n_j	j 关键词内所有标签的 Count	S_i	i 标签的关键词集合
P_i	S_i 集合的标签概率值	l_j	j 关键词的长度
L	文件名长度。	P	文件名的标签概率值
Q_1	一级标签的分类准确率	N_F	一级标签分类正确的数目
N	参与分类的文件数目	Q_2	二级标签的分类准确率
N_S	二级标签分类正确的数目	N_T	训练集的文件数目

缩略词

缩略词	英文全称
IDC	Internet Data Center
HDFS	Hadoop Distributed File System
GFS	Google File System
FIFO	First In First Out
LRU	Least Recently Used
LFU	Least Frequently Used
LRFU	Least Recently Frequently Used
RR	Round Robin
HAR	HadoopArchives
LB-LRU	Label Based Least Recently Used

第一章 绪论

1.1 课题研究背景及意义

近几年来，随着人工智能、移动网络、大数据、物联网等互联网技术的快速发展，各行各业产生的数据量呈几何级增长，全球每天约产生 PB 级别的数据。据互联网数据中心（IDC, Internet Data Center）研究统计，仅 2017 年全球产生的数据量已达到 15.27 ZB，同比增长 35.7%。未来几年全球的数据增长速度将达到 25%以上，预计 2020 全球数据总量将达到 40 ZB，我国数据量也将达到 8060 EB，占数据总量的 18%。大数据时代的到来不仅带动存储行业的发展，同时也对存储提出了更高的要求。数据的增长不仅仅在数据存储设备上增加了数据中心的存储成本，同时也在数据维护和数据管理方面增加了成本。而这些数据中仅有 20%会被经常访问，我们称这种数据为热数据[1]，另外 80%的数据称之为冷数据[1]。冷数据被访问的概率很低，但是其数据量非常庞大，仍具有独特的价值。比如政府的人员档案信息、医院的病例信息、银行的历史消费记录、企业的客户信息等都需要长期保存，且在需要时能随时访问。如果冷数据和热数据均使用磁盘阵列，将带来存储资源和能耗的浪费。针对冷数据存储[2]问题，Facebook 宣布引进 10000 张蓝光光盘建立其冷数据存储中心，存储规模达到 PB 级别——甚至高达 5 PB，将冷数据和重要文件备份到其中，达到降低数据中心存储成本的目的。因此，设计一种单位存储成本低、能耗低的海量数据存储系统，将其运用到大数据存储中具有非常现实的意义。

目前市场上主流的几种存储设备中，磁盘阵列在数据访问速度方面具有绝对优势，主要被作为二级存储介质，占据了存储市场大部分份额。磁带库[3]和光盘库[4]因其单位存储成本较低等原因，主要被作为冷数据存储备份系统。随着近几年光盘技术[4]的发展，以光盘作为存储介质的集成存储设备被业界重新重视起来，光盘库数据传输速度和磁盘阵列的差距逐渐在缩短。光盘库前期购买成本高于磁盘阵列，但是光盘库的能耗仅为磁盘阵列的 3%左右。而且数据中心每隔五年需要更换一次磁盘阵列，而光盘库的使用寿命高达五十年，且对于存储环境要求较低。随着存储年限的增加，光盘库的整体存储费用将会明显低于磁盘阵列。另外光盘库内数据采用一次刻录不可更改的原则，数据安全性极高，不会被更改或病毒入侵。所以光盘库非常适合作为大数据存储系统[5]的底层存储设备，用来存储冷数据和一些重要文件的副本。

为了实现对海量数据的存储和管理，以 Hadoop 分布式文件系统（HDFS, Hadoop Distributed File System）[6]和 Google（GFS, Google File System）[7]为基础的各种分布式文件系统[8]被应用到各大互联网企业中。HDFS 文件系统，因其易扩展、高安全性、高容错性等优点，受到互联网企业青睐。HDFS 文件系统[9]非常适合大文件存储，适合读频繁的模式，HDFS 存储特性与光盘库十分相似。因此研究人员提出一种基于光盘库的 Hadoop 分布式文件系统（HDFS 光盘库）

[10]，由于 HDFS 和光盘库在面对海量小文件数据的时候效率不高[11]。一方面大量小文件会在名字节点（NameNode）下产生大量的元数据，增加 NameNode 节点的存储压力，同时也会降低数据的查询速度。另一方面由于光盘库物理结构[12]的原因，光盘库机械臂取送盘的时间相对于光驱读取小文件时间占比过高。因此 HDFS 光盘库在客户端（Client）内对小文件进行单独处理，解决 HDFS 光盘库在存储小文件时效率不高的问题，在数据节点（DataNode）内通过数据块缓存策略提高光盘库刻录速度，并通过 I/O 调度策略在读取数据的时候减少机械臂调用次数。HDFS 光盘库与传统的光盘库相比，HDFS 光盘库不仅在存储容量和数据传输速度方面有很大提升，而且实现了海量数据的管理。但是 HDFS 光盘库仅仅根据时间局部性原理对小文件进行缓存，将其合并为大于 HDFS 内数据块大小的大文件，小文件之间的关键性相对较低，同一大文件中文件连续访问的概率较低，导致数据块预取的缓存命中率降低。虽然 HDFS 光盘库设置了数据块缓存，但是缓存容量有限，只能缓存很少一部分文件。虽然 HDFS 光盘库相对光盘库性能提升很多，但是与磁盘阵列比较仍存在很大差距。

对于上述 HDFS 光盘库在存储小文件时出现的问题，可以采用在 HDFS 光盘库前端加入磁盘缓存系统。一方面磁盘缓存系统作为用户和 HDFS 光盘库之间缓冲区，通过在磁盘缓存内文件分类[13]和小文件合并[14]等处理，保证 HDFS 光盘库内存储文件均大于数据块容量，不仅减少了文件的上传时间，而且也提高 HDFS 光盘库系统的吞吐量。另一方面，缓存替换算法[15]在磁盘缓存内实时保存热数据，屏蔽了底层 HDFS 光盘库，减少用户访问数据时间。因此用加入磁盘缓存系统的方法解决 HDFS 光盘库处理小文件效率不高和用户响应时间较高等问题非常有效。

综上所述，使用 HDFS 光盘库作为大数据存储系统的底层存储设备，结合磁盘缓存系统，通过虚拟存储技术[16]设计适合大数据存储的存储系统具有非常现实的实用价值。

1.2 国内外研究现状

缓存技术最早是为了解决内存和 Cache 的速度差距较大的问题，这是因为 Cache 的读写速度明显高于内存，且 Cache 存储容量比较小，所以 CPU 缓存[17]技术非常重要。后来缓存技术已经被应用到很多不同的存储系统当中，磁盘缓存[18]主要解决磁盘相对内存速度差距较大问题，通过设计写缓存和读缓存提高数据读写速度。Web 缓存[19]是将客户机近期使用的数据存储在离客户机更近的网络服务器和客户机上，当客户机再次需要这些数据的时候，客户机和网络服务器首先查询本地数据，若查询到所需数据，则可减少使用 Web 服务器的时间，提高数据访问速度。缓存技术由缓存替换算法和预取技术两部分组成，下面介绍这两部分内容。

1.2.1 缓存替换算法

缓存替换算法是存储系统的主要研究方向，缓存命中率是评价算法优劣的重要衡量因素。

本文主要研究磁盘和 HDFS 光盘库之间的二级缓存技术，二级缓存相对于一级缓存，二级缓存在缓存容量和缓存数据访问间隔更高，突发性访问所占的比例也相对较高，二级缓存算法大多借鉴内存和 CPU 之间缓存策略的思想。比较常见的一级缓存算法有随机替换算法[20]（RR, Round Robin）、先进先出算法[21]（FIFO, First In First Out）、最近最少使用算法[22]（LRU, Least Recently Used）、最近不频繁使用算法[23]（LFU ,Least Frequently Used）、最近最少使用次数算法[24]（LRFU, Least Recently Frequently Used)等。RR 算法是在缓存容量不足时，随机选择一个数据项淘汰，RR 算法实现比较容易，但是缓存命中率较低。FIFO 算法则是将最先存入缓存的数据项淘汰，FIFO 算法相对于 RR 算法缓存命中率有所提高，但仍然无法满足需求。LRU 算法设计原理来源于时间局部原理，将上次访问时间距当前时间间隔最大的数据淘汰，LRU 算法可通过硬件或软件方式实现，实现难度不高，且命中率很高，LRU 算法及其改进算法被应用在各种存储系统、虚拟存储管理和 I/O 缓存当中。LFU 算法是将缓存中最近一段时间内访问记录最少的数据淘汰。LRFU 结合 LRU 算法和 LFU 算法优点，LRFU 算法为每个页面设计一个记录其被访问可能性权重值，页面权重值随着时间而减小，只有缓存命中时才会改变其数值，当缓存空间不足的时候，优先淘汰权重值较小的页面。除了上述几种一级缓存替换算法以外，目前针对磁带库和光盘库的二级缓存替换算法，有几种应用比较广泛的算法，如 LRU-K 算法[25]、2Q 算法[26]、MQ 算法[26]等。

LRU-K 算法的设计思想来源于 LRU 算法，其主要策略是保存缓存内所有数据项的最后 K 次的访问记录，通过数据项访问记录估算出数据项连续两次访问的期望时间间隔，每次优先将期望时间间隔最大的数据项淘汰出缓存。例如数据项 1 的期望时间间隔大于数据项 2，那么数据项 1 优先被淘汰。LUR-K 对于突发性访问较少的存储系统，取得较好的性能，但是由于 LRU-K 算法是根据数据项的近期访问记录，需要大量的存储空间且算法的复杂度较高，且对于访问频率变化较大的缓存，对系统性能提升的效果较差。

2Q 算法通过 A_1 和 A_m 两个队列对缓存内数据进行管理，A_1 根据系统预先设定的两个值 K_{in} 和 K_{out} 划分为 A_{1in} 和 A_{1out}，A_{1in} 用来存储从下一级存储设备内读取的数据，A_{1out} 则用来存储从 A_{1in} 内替换出的数据。使用 FIFO 算法对 A_1 内部数据进行管理，使用 LRU 算法对 A_m 内部数据进行管理。数据在访问过程中，首先查询 A_m 队列，若命中则将命中数据放入 A_m 队尾，若未在 A_m 队列中查询到数据则查询 A_1 队列，若在 A_1 队列中命中则将命中数据项按照 LRU 替换算法替换 A_m 队列中数据项。若 A_1 队列和 A_m 队列均未查询到数据，从下一级存储设备内读取的数据，按照 FIFO 算法方式将 A_{1in} 中数据淘汰至 A_{1out} 中，同时将 A_{1out} 中替换的算法从缓存中删除。2Q 算法的缺点在于 K_{in} 和 K_{out} 是系统事先已经设定好，A_1 队列无法根据系统运行状态进行调整。

MQ 算法中使用多个队列对缓存空间内数据进行管理，其内部全都使用 LRU 算法对的每个队列中数据项实现管理。例如 MQ 算法中有 K+1 个队列，队列名称为 Q_{out}，Q_0，Q_1，Q_2，……，

Q_i，.....Q_k，其中 i 表示队列名称。系统访问数据的时候，数据查询顺序为 Q_k，.....，Q_i，……，Q_2，Q_1，Q_0，Q_{out}。如果在 Q_i 检索到访问数据，则把数据移动至 Q_{i+1} 的末端，将 Q_{i+1} 中淘汰的数据移动到 Q_{out}，中，若 Q_{out}，队列已满，则按照 LRU 算法方式将 Q_{out}，队列中数据淘汰，若在 Q_{out} 队列中命中数据，则将数据插入 Q_0 队列尾部。如果在所有队列中都未命中，则通过下一级存储设备读取数据。

1.2.2 预取技术

预取技术是一种主动获取缓存内容的过程，根据系统内存储对象之间的关联性和存储对象的历史访问信息等，预测用户后续可能使用的数据并将其提前预取到缓存系统，降低用户的数据访问延迟。根据预取对象的不同，将预取技术分为 CPU 数据预取技术[27]、流媒体预取技术[28]和 Web 页面预取技术[29]等，虽然预取对象不同，但根据预取信息的分析方式的可以将预取技术大致分为以下几种。

(1) 基于语义信息的方式

Web 页面的缓存对象是 Web 页面内容，则可以通过分析 Web 网址的超文本标签和 URL 等信息，获取 Web 页面中图片和文字之间的关联性。Xu 等人在文献[30]中对网页的历史访问记录使用神经网络进行训练，并根据训练结果抽取 Web 页面的关键词，依据关键词信息确定预取内容，但是该方案对语义信息不丰富的 Web 页面的预测效果较差。许欢庆等人在文献[31]中则是对用户的每一次会话内容进行关键词提取，根据会话中提取的关键词信息确定会话的主题，缓存预取的内容则是根据会话主题确定。该文献中虽然使用了隐马尔可夫（Markov）模型，但论文的主要研究内容还是通过分析用户访问网页内容的语义信息获取会话的关键词和主题，同样需要 Web 网页内容具有丰富的语义。曹仰杰等人在文献[32]中将自适应剪枝技术加入 Markov 树的构造过程，达到减少状态空间的目的。班志杰等人在文献[33]中针对在线模型中运用非压缩 PPM 树的改进方案。金民锁等人在文献[[34]中将隐马尔科夫模型运用到分析用户信息过程中，例如将其用于分析用户的历史访问记录，从而获取用户兴趣并保持较高的准确率，但是仍然存在依赖高强度的语义信息且其算法复杂度高等问题。

(2) 基于数据挖掘的方式

Li.M 等人在文献[35]中记录缓存数据的访问频率并进行数据挖掘与分析，根据分析结果确定预取内容，并动态的的选择最佳的预取内容的大小。王文建等人在文献[36]中通过对 Web 页面的访问日志进行数据挖掘与分析，根据分析结果建立预测树后进行数据预取，但是该文献设计的算法时间复杂度较高，并未考虑对缓存系统内全局频率特征进行利用。徐宝文等人在文献[37]通过对用户历史访问信息进行挖掘，为每个用户构建个人兴趣，并根据个人兴趣为用户预取数据。Zhu 等人在文献[38]则是计算缓存对象的时间差和频率，根据两者大小关系确定缓存预取的内容。在文献[39]中对服务器一段时间内访问频率最高的前十文档进行统计，当接收某一用户数据访问

请求的时候，将访问频率最高的前十的文档和当前访问文档同时发送给用户。该文献设计的预取技术虽然实现较为简单，且在测试中取得不错的效率。并未考虑对每次访问特点进行利用，因此预取结果的准确率并不是很高。Chen 等人在文献[40]中通过统计文档的流行度，结合 Markov 树对系统内接下来可能访问的文档进行预测，虽然 Chen 等人设计的方案可以有效降低部分系统消耗，但是需要分析用户访问的信息，该过程会造成一定的访问延时。

(3) 基于访问概率的方式

最后一种预取技术是通过分析数据的历史访问记录，然后根据用户的历史访问概率特点获取预取内容。基于访问概率的方式中比较经典的文件预取算法是通过分析文件历史访问获取文件关联性，其中最具有代表性的有第一次后继算法(Frst Successor)、最后后继算法(Last Successor)、稳定后继算法(Stable Successor)[41]、基于概率的后继预测算法(Probability-based Successor Group Prediction)等。在 Frst Successor 算法中，若数据项 A 在首次使用之后，接下来访问的数据为数据项 B，然后数据项 A 再次被使用的时候，该算法认为数据项 B 再次被调用的概率最高。Last Successor 算法与 Frst Successor 算法相反，该算法将数据项 A 最后一次访问的后继访问数据项 B 作为数据项 A 的预取后继对象。上述两种后继算法虽然实现较为简单，但是在实际存储系统却很少被使用，主要原因是算法的预取结果的缓存命中率并不是很高。Amer 等人在 Last Successor 算法基础上又提出了 Stable Successor 算法[42]，其与 Last Successor 算法的主要区别在于在确定数据项 A 的预取对象的时候，不再将数据项 A 的最后一次访问记录作为判断标准，而是记录数据项 A 的最后的连续 N 次访问，如果在这 N 次访问记录中，数据项 A 的后继访问对象均为数据项 B，才认为 B 为 A 的预取对象。基于概率的后继预测算法则是根据文件的历史访问记录创建关系图，并按照关系图内数据之间联系确定预取文件信息。Pirollip 等人在文献[43]中设计一种 K 阶的 Markov 预测模型，预测数据的命中率随着 Markov 模型阶数的增加而增加，但是其内存开销也在不断增加。Mabroukeh 等人在文献[44]中在 Markov 模型中利用语义信息进行剪枝，提高其预测的准确度。Markov 模型[45]在对数据的访问记录的利用率和预取的准确度都非常高，但是 Markov 模型的内存消耗和系统开销同样也非常高。

1.3 本文的主要研究工作

本文在 HDFS 光盘库前端加入磁盘缓存系统降低的用户响应时间，并在磁盘缓存内针对小文件单独进行处理，在磁盘缓存系统内设计并实现一种文件管理系统。论文主要研究内容如下：

(1) 在研究预取技术的基础上，参考文件主题提取技术，设计一种基于文件名的标签分类算法，该算法主要包括：

a）对当前 HDFS 光盘库的存储文件进行分析，借鉴文件主题的提取方式，针对文件名建立关键词数据库。

b）根据关键词数据库内容与文件名进行匹配操作，提取文件名内所有关键词信息。

c）依据文件名和关键词数据库中关键词信息，设计一种标签概率的计算方法，根据文件名内标签概率值确定文件标签。

(2) 研究当前分布式存储系统面对小文件存储的解决方案，根据文件标签信息设计一种小文件存储方案。

a）为了解决 HDFS 光盘库存储小文件效率不高的问题，依据文件的标签信息在磁盘缓存系统内对小文件进行合并。

b）创建大文件内所有小文件的索引信息，并在 HDFS 光盘库内保存索引信息。

c）针对合并大文件设计小文件恢复的方法

(3) 针对磁盘缓存系统内的小文件、合并大文件和大文件，根据文件访问记录，设计一种基于文件标签的缓存替换算法实现对缓存内文件进行管理。

1.4 本文的内容安排

第一章 绪论部分介绍本文研究的背景及研究意义，介绍大数据时代存储行业的需求及挑战，当前光盘库在大数据存储中具有的优势，简单阐述 HDFS 光盘库在冷数据存储具有的优势，并提出系统中存在的问题。说明了本文提升 HDFS 光盘系统性能所做的研究工作。在对现有缓存替换算法和预取技术介绍的基础上，说明了本文的改进工作思路。

第二章 介绍本文研究所涉及的基础理论和相关技术。首先介绍底层存储 HDFS 光盘库的内部结构，然后讨论了 HDFS 处理小文件的方案。

第三章 分析 HDFS 光盘库在存储文件时遇到的问题，根据 HDFS 光盘库存在问题对本文设计的磁盘管理系统进行功能和模块的划分，并对磁盘缓存系统内标签分类模块、虚拟存储模块和 Cache 模块进行详细设计。

第四章 叙述讨论磁盘缓存系统关键模块的具体实现，主要说明了系统内标签文件目录、标签分类模块、虚拟存储模块、Cache 模块的实现技术。

第五章 介绍测试中建立的 HDFS 光盘库系统和磁盘缓存系统，对磁盘缓存系统各个模块的性能测试验证工作，并根据测试结果对模块的性能给出分析和总结。

第六章 本文研究工作总结，并对下一步的研究方向给出了意见与展望。

第二章 相关理论与技术

2.1 HDFS 光盘库文件系统概述

HDFS 光盘库是一种新型基于光盘库的三级大数据存储系统，通过分布式存储技术将光盘库组成一个海量数据存储系统，HDFS 光盘库结构如图 2.1 所示。HDFS 光盘库内部结构和 HDFS 内部结构非常相似，其主要由 Client、NameNode 和 DadaNode 三部分构成。Client 负责对外的文件读写服务，并在 Client 内对小文件进行单独处理，解决 HDFS 光盘库处理小文件效率不高的问题。NameNode 负责保存传统 HDFS 内信息之外，还必须保存小文件的索引信息。DataNode 节点通过块文件管理模块实现 HDFS 和光盘库的融合。HDFS 光盘库将低于 64 MB 的数据规定为小文件，小文件在上传的时候，首先将其加入 Client 缓存，当 Client 缓存数据容量超过 64 MB 的时候，将缓存数据合并成一个大文件，并建立缓存数据的索引信息，通过 NaneNode 接口将大文件和索引信息发送给 NameNode 节点。当系统需要访问某个小文件的时候，Client 将访问请求转发至 NameNode 节点，通过 NameNode 内小文件索引 SFINode 获得对应合并大文件信息，再根据命名空间（NameSpace）内数据块元数据获取大文件所有数据块信息，将大文件下载到 Client 缓存内，并根据索引信息将小文件恢复至 Client 中。下面介绍 HDFS 光盘库内主要模块。

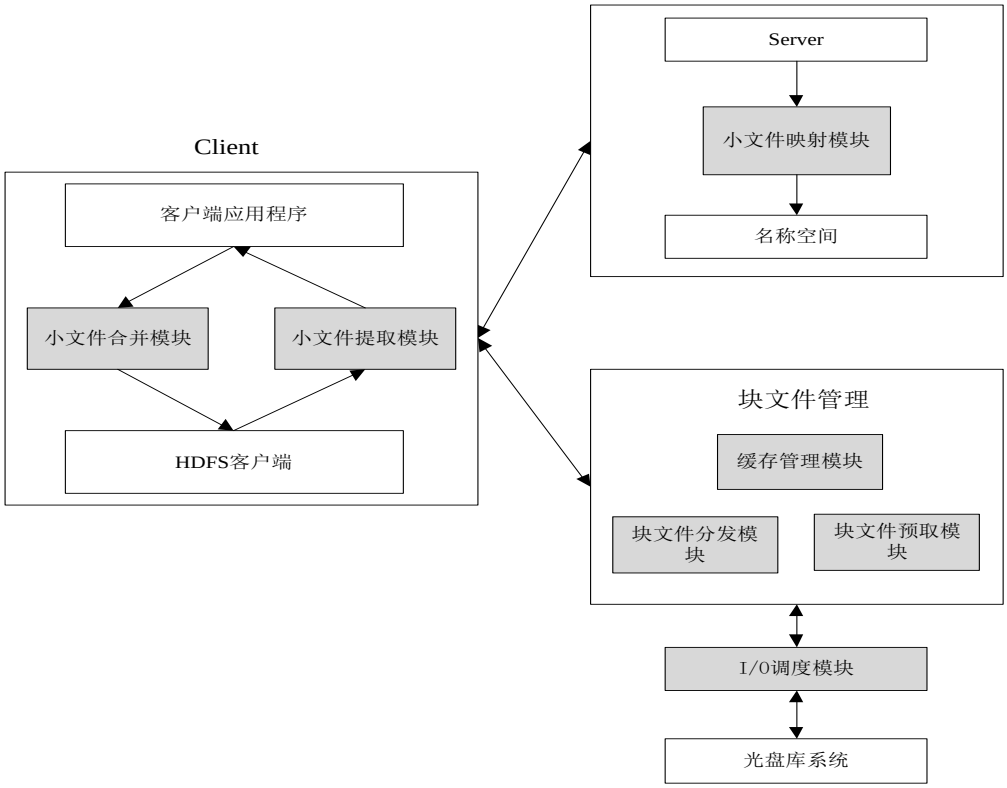


图 2.1 HDFS 光盘库系统结构图

2.1.1 小文件处理模块

由于 HDFS 和光盘库在存储小文件的效率都不高，因此 HDFS 光盘库设计小文件处理模块解决上述问题，其主要由以下三个部分构成。

小文件合并模块负责对用户新上传的小文件进行缓存，当 Client 内缓存容量达到 64 MB 的时候将缓存数据进行合并并建立索引信息。HDFS 光盘库在 Client 内设置 64 MB 的空间，主要存放 Client 的小文件数据，并建立一个索引对象用来记录当前缓存数据在大文件中的位置信息，具体结构如图 2.2 所示。索引对象主要记载以下内容，BigFilePath 记录合并大文件在 HDFS 光盘库的存储路径，samllfilepath 记录小文件在 HDFS 光盘库的存储路径，offset 表示小文件在缓存内的偏移位置，long 表示缓存中小文件的大小。smallfilepath、offset 和 long 信息是在小文件加入缓存空间的时候加入的，直到缓存小文件大小达到或者略小于 64 MB，然后将索引对象和大文件写入 HDFS 内，同时在 NameNode 节点将索引对象保存在 SFINode 内。

BigFilePath
File 1；（smallfilepath，offest，long）
...
File N；（smallfilepath，offest，long）

图 2.2 大文件索引结构图

小文件映射模块主要负责在 NameNode 节点内保存索引信息，小文件提取模块主要负责在 Client 内根据索引信息恢复小文件。系统访问小文件的时候，Client 向 NameNode 节点发送小文件 samllfilepath，NameNode 通过查询 SFINode 获取所属大文件的全路径信息，再查询 NameSpace 空间获取大文件所包含数据块的位置信息，最后建立 Client 与 DataNode 之间的通信联系，将数据块和索引信息下载到 Client 存储空间，并根据索引信息将小文件恢复至 Client。

2.1.2 块文件管理模块

块文件管理模块是 HDFS 应用到光盘库的的重要实现部分，其主要功能是管理 DataNode 节点数据块并对外提供数据块的读写服务。DataNode 节点内保存包含所有数据块的信息链表 BlockIndexTable，链表内部包含数据块 ID、数据块的大小、数据块的存储位置和时间戳等信息。根据索引表设计了缓存管理模块对 DataNode 节点内数据块进行管理，在块文件分发模块内为每个光驱设置双缓冲提高光驱的读写速度，并根据数据块之间的关联性设置块文件预取模块，提高数据的读取速度。

块文件分发模块的主要功能是将 DataNode 节点公共缓存区的数据块迁移至光驱的缓存区

内。如图 2.3 所示，光盘库的物理结构中包含多个光驱，负责多个光盘组的具体读写过程，DataNode 节点为单个光驱配置双写缓存，保证光驱在刻录数据的同时仍可以接收其他数据块的刻录任务，减少数据块从公共缓存区迁移至光驱缓存的等待时间。块文件分发模块使用两种策略提高系统的数据传输速度，第一种策略是将某一时间段内产生的数据块按时间戳进行分类，分别交由不同的光驱完成刻录，保证系统在读取这些文件的时候，通过光驱并行读取文件提高读取速度；第二是将同一文件下的所有数据块迁移至同一光盘组缓存内，保证同一文件的数据块刻录在一张光盘内，在读取数据块的时候可降低光盘库换盘次数。

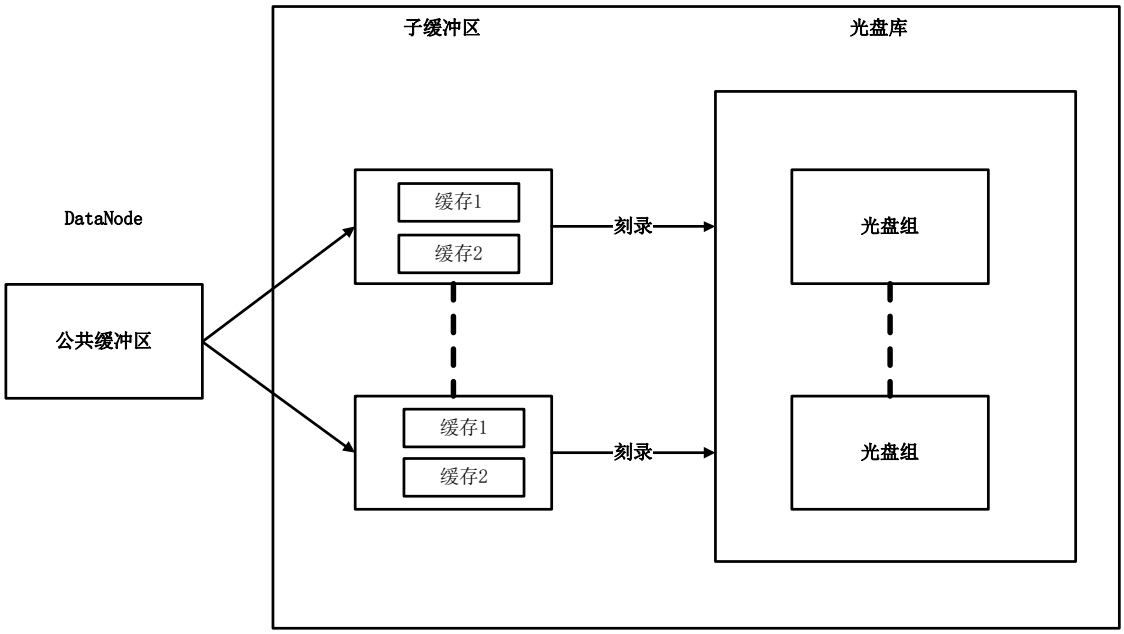


图 2.3 块文件分发结构图

缓存管理模块主要负责将系统内经常访问的热数据保存在磁盘公共缓存区内，从而可减少访问光盘库次数，提高文件访问速度。块数据缓存策略如图 2.4 所示，DataNode 节点维护了 Hot 和 Drop 两个队列，Hot 队列与 Drop 队列内都保存 DataNode 节点内使用次数较高的数据块 ID 及历史访问信息，Hot 队列保存 DataNode 节点内大部分数据块信息，主要用于读文件首次查询，Drop 队列记录刚从光盘库内读取的文件，主要负责数据块迁移替换任务，实现将数据块的查询和替换工作在两个队列内并行进行。系统在查询数据块信息的时候，先查询 Hot 队列内数据块信息，找不到再去查询 Drop 队列，若仍然未找到数据块信息，则通过 BlockIndexList 查询数据块对应的存储信息，将数据块读取至公共缓冲区。若在 Drop 队列中查询到所需数据块的信息且 Hot 队列仍有存储空间，把 Drop 队列中查询到所需数据块的信息迁移至 Hot 队列末端，如果 Hot 队列的空间不足，则将 Drop 队列中查询到所需数据块的信息与 Hot 队列末端数据项进行交换。当 Hot 队列和 Drop 队列都查询失败后，如果 Drop 队列仍有存储空间则将光盘库内读取数据块移动至 Drop 队列中，如果 Drop 队列的空间不足，则将 Drop 队列末端删除相同数目

的数据项并将光盘库内取出的数据块直接插入 Drop 队列内。

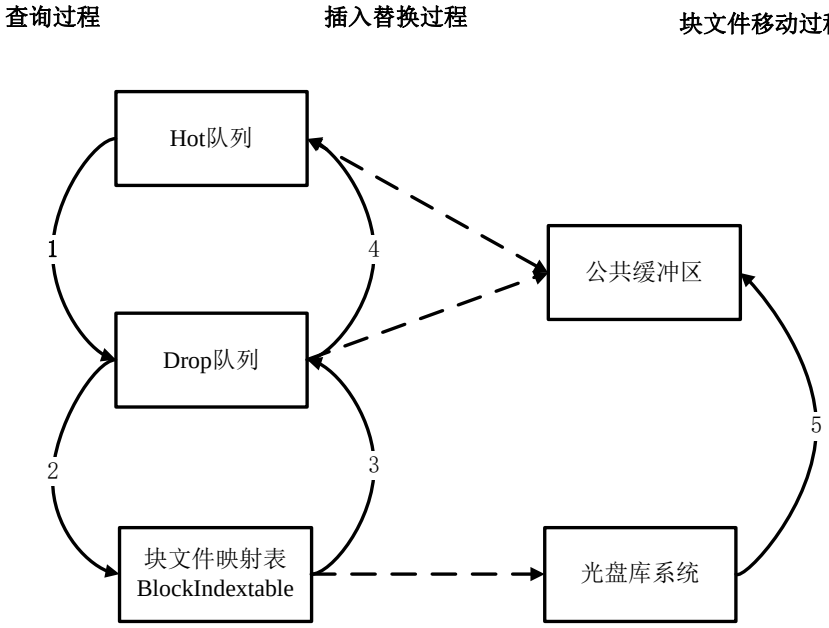


图 2.4 数据块缓存策略图

块文件预取模块则是另一种提高系统性能的重要技术，当 Hot 队列和 Drop 队列均不包含访问数据块信息的时候，系统通过查询 BlockIndexList 找出具有相同时间戳的所有数据块信息，并把其打包成批量数据块读任务交由 I/O 调度模将这些数据块备份至 DataNode 节点的磁盘空间内，并对 Hot 队列和 Drop 队列内数据块信息进行更新。通过批量数据块读请求可减少光盘库机械臂调度次数，且同一文件下的数据块具有较强关联性，在一段时间内会很大概率被访问。

2.1.3 光盘库 I/O 调度模块

I/O 调度模块负责对 DataNode 节点内数据块读任务设计调度策略，尽量减少机械臂的使用次数。由于光驱内采用双缓存机制，对于文件的写任务的压力较小，系统对于读任务的需求高于写任务，因此 I/O 调度模块主要针对读任务设计优化策略。对于访问在光盘库缓存区的数据块，直接将数据块备份至 DataNode 公共缓存中即可。HDFS 光盘库在最长队列优先算法基础上，设计一种适合光盘库 I/O 调度策略。该策略在选取 DataNode 节点内读任务的时候，筛选出读任务链表中高于系统设置的时间阀的链表项，优先对这些链表项完成读取任务；然后在没有高于系统设置的时间阀的链表项中选取读任务所需时间最长的链表项完成读取任务。

2.2 HDFS 小文件处理策略的概述

在 HDFS 处理海量小文件存储策略中，其中大多数采用 HDFS 自带的小文件处理策略改进和优化策略，还有少部分策略是根据特定的应用场景设计小文件存储方案等，下面介绍 HDFS 自带小文件处理策略和现有的其他优化策略。

2.2.1 HDFS 原生小文件处理策略

(1) HAR 文件归档技术

Hadoop 开发社区为了解决小文件带来的元数据问题提出 HadoopArchives（HAR）文件归档技术[46]，用户使用 HDFS 内提供的 archives 工具，把 MapReduce 任务中当前目录下文件转换为一个包含数据和索引的 HAR 文件，再存入 HDFS 中，HAR 文件的结构如图 2.5 所示。HAR 将待处理的小文件打包存入分片（Part）文件中，建立如图 2.5 所示的两级索引信息，实现对小文件进行定位和查找，从而达到从 Part 文件内读取数据的目的。HAR 文件归档技术是 HDFS 自身提供的小文件处理工具之一，在解决海量小文件带来的元数据问题的同时，仍然允许 MapReduce 对小文件内容直接访问，即经过 HAR 处理的小文件仍然能够当作 MapReduce 的输入。MapReduce 程序在编写的过程中可以使用“har：//URL”的访问格式对 HAR 文件内部的小文件进行操作，但是 HDFS 并未对外提供 HAR 文件内部小文件的元数据管理方法和接口，所以无法以传统管理文件的方法管理经过 HAR 处理后的小文件。

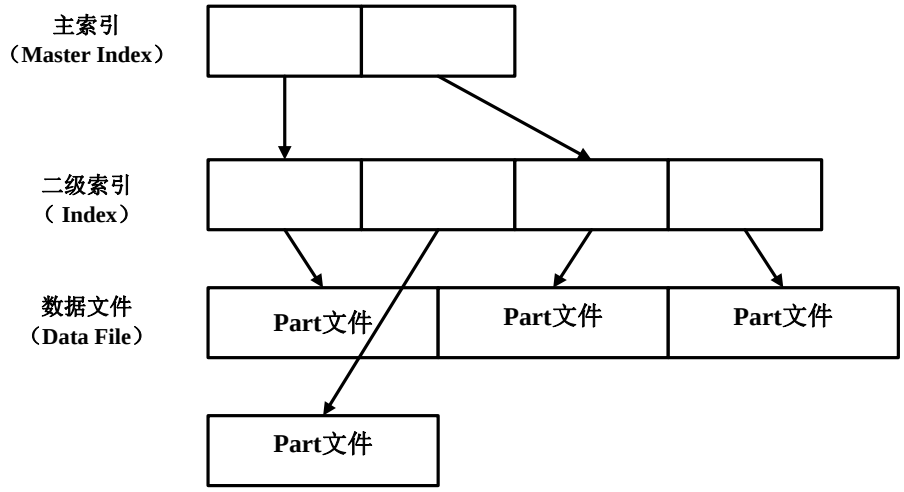


图 2.5 HAR 文件数据结构图

虽然 HAR 文件归档技术解决海量小文件带来 NameNode 节点负载过重问题，并且支持 MapReduce 输入和提供快捷的小文件访问方式，很好的解决了小文件存储问题，不过 HAR 文件归档技术也存在不足的地方。例如 HAR 在创建的过程中，需要额外的存储空间保存原始文件的副本，同时 HAR 在将小文件打包为 Part 文件的时候，为了支持小文件访问，需要维护如图 2.5 的两级索引信息，通过索引信息读取 HAR 内小文件一般情况下相比读取其他文件速度要慢。另外 HAR 文件的内容无法修改，如果 HAR 文件需要对其内部小文件实施增加和删除操作，只能通过重新创建的方式完成。

(2) SequenceFile 和 MapFile

HFDS 除了提供 HAR 以外，还提供了 SequenceFile[47]和 MapFile 两种容器实现小文件存储。SequenceFile 文件内部数据结构如图 2.6 所示，其内部数据以经过序列化处理的二进制键值对的

形式进行存储。HDFS 在存储小文件的时候，以小文件的内容作为值和名称为键值的方式存入 SequenceFile 容器中。SequeneceFile 支持压缩选项，可以减少网络数据传输流量和节约磁盘空间，不过仍然有一些缺点。SequenceFile 容器未建立索引结构，所以当需要获取 SequenceFile 容器内的小文件的时候，需要遍历 SequenceFile 容器才能查询小文件位置，另外 HDFS 同样无法对 SequenceFile 容器内部的文件以常规文件的方式进行管理

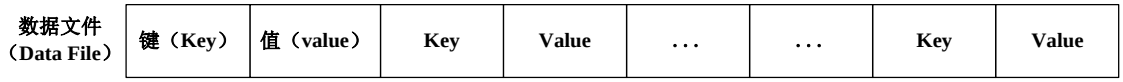


图 2.6 SequenceFile 容器结构图

MapFile 是对前文叙述 SequenceFile 的数据经过排序建立小文件索引，解决 SequenceFile 查询小文件目标位置效率低的问题。用户在使用 MapFile 构建大文件过程中，将会在索引文件中记录小文件在 SequenceFile 内的键偏移值，MapFile 容器数据结构如图 2 .7 所示。当需要访问 MapFile 容器中小文件的时候，将索引文件下载至内存，同时依据索引文件查询小文件的存储信息。MapFile 的小文件读取速度相比 SequenceFile 得到了提升，但是同 SequenceFile 一样仍然无法对 MapFile 容器内部文件以常规文件的方式进行管理。

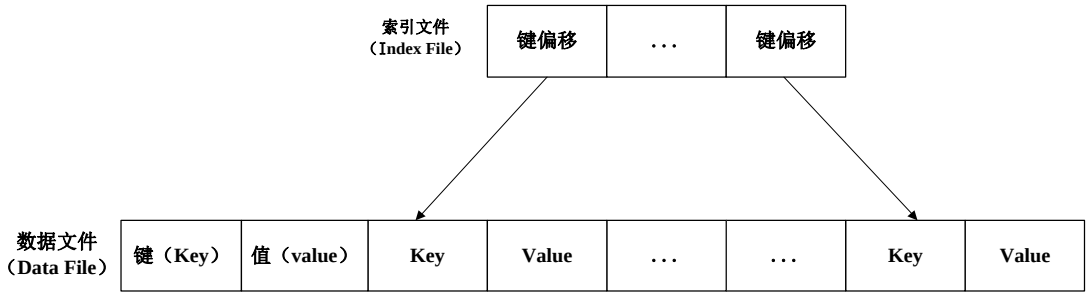


图 2.7 MapFile 容器结构图

(3) Federation 方案

HDFS 自带的第三种小文件处理策略为 Federation 方案，Federation 方案通过对 HDFS 内 NameNode 节点进行水平扩展，将海量小文件的元数据信息存储在多个 NameNode 节点减轻 NameNode 节点压力。Federation 方案结构如图 2.8 所示，Federation 方案将 HDFS 中目录划分为多个独立部分，采用多个 NameNode 节点对多个独立部分分别进行管理，每个部分的元数据信息存储在一个 NameNode 节点内。例如图 2.8 中的 NameNode1 记录 User 目录下文件和数据块的元数据信息，其它 NameNode 节点将无法对该节点内的数据进行管理，但是所有的 DataNode 节点要向每个 NameNode 节点进行注册和汇报，同时需要对来自所有 NameNode 节点的命令进行操作。

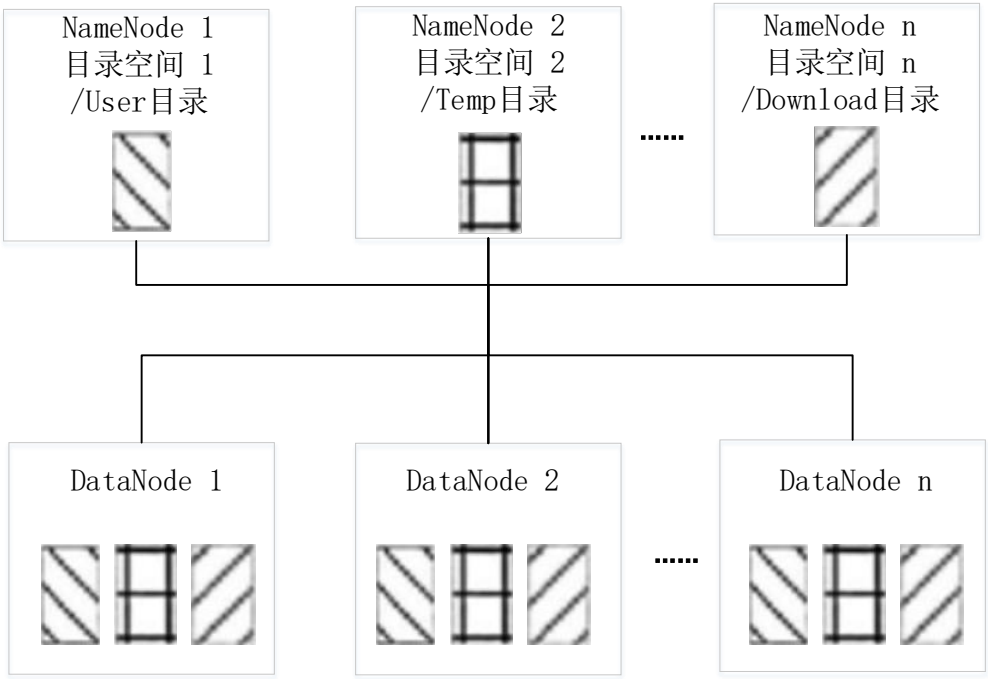


图 2.8 Federation 结构图

Federation 方案在 HDFS 内部设置多个 NameNode 节点，将海量小文件的元数据信息分别存储在多个 NameNode 节点，可以有效地解决元数据管理问题。Federation 方案相对于 HAR 和 SequenceFile 的主要优势在于能够对 HDFS 内所存储的小文件实现完整管理，可对存储小文件采用常规文件的方式进行管理，不过 Federation 方案也存在一些缺点。由表面上来看小文件元数据容量在水平上得到了扩展，如果其中某个 NameNode 节点意外退出 HDFS，则该 NameNode 节点所管理的文件将无法使用，其管理的 DataNode 的数据块同样也无法被其他 NameNode 节点管理。并且诸如小文件读写问题、高可用性配置复杂、负载均衡需手动介入等问题未能得到很好的解决。

2.2.2 其他小文件处理策略

目前在很多领域中都存在海量的小文件，诸如社交网络、电子商务、电子图书馆、电子学习、天文学领域、WebGIS 系统、PPT、能源、生物、气候等，而且小文件在整体数据中所占比例也在不断提高，HDFS 存储小文件的效果不佳的现状急需要解决。目前 HDFS 提供 HAR、SequenceFile 和 Federation 方式来解决海量小文件带来的问题，但是一方面都存在数据访问时间较长的问题，另一方面无法对所有场景都适用。因此国内外研究学者致力于研究 HDFS 小文件存储的优化方案，主要分为两种，一种是对 HDFS 提供的小文件存储方案进行优化，另外一种则是根据特定的应用场景设计小文件存储的方案。

现有对于 HDFS 提供的小文件存储方式的改进策略，文献[48-58]比较具有代表性。文献[48]

在 HAR 文件归档技术基础上，将小文件进行分组归档处理后，再将小文件合并大文件，并在 NameNode 节点内保存所有合并大文件的主索引，将数据块和二级索引保存在 DataNode 节点内，通过二级索引方式实现对小文件目录进行检索，从而达到降低 NameNode 节点内元数据数量过多带来的存储压力。文献[49]同样是在 HAR 文件归档技术的基础上提出了一种改进方案，Chatporn 等人根据对 HAR 的性能瓶颈的分析结果，使用一层的哈希索引替换 HAR 文件中原有的主索引和二级索引，将小文件检索过程由 HAR 文件的两次索引定位减少为一次哈希映射查找，达到减少 HAR 小文件检索时间的目的，另外该方案使用间接手段实现对 HAR 文件追加效果，提高了 HDFS 对 HAR 中小文件处理的能力。文献[50]则是对 HDFS 提供的 SequenceFile 进行优化，使用一种基于 B+树的索引结构实现小文件的读取，相对 MapFile 的索引方式，利用 B+树的高度低且树中节点有序的特点，减少小文件的检索时间。文献[51]同样是对 HDFS 提供的 SequenceFile 进行优化，对系统内海量气象小文件按照文件名进行排序，采用 SequenceFile 的方式存储海量的气象小文件，气象小文件经过排序后，能够根据二分查找等方式能够从 SequenceFile 索引中更快地实现小文件的查找和读取。文献[52]则是对 HDFS 提供的 Federation 方案进行优化，文中采用对小文件进行合并使得小文件性的访问性能得到优化，但是该文献仍然使用多个 NameNode 节点提高系统的存储规模，并且未能解决 Federation 方案存在的负载均衡等问题。上述针对 HDFS 自身提供的小文件处理策略的改进方案中，虽然在小文件的处理性能和访问性能得到了很大提升，但是仍然未设计对海量小文件的元数据有效管理策略。

除了以上优化方案以外，其他多为根据特定的应用场景设计的小文件存储方案，还有部分是选择最优的小文件的合并时机和处理时间或者选择最合适的小文件进行合并。在文献[53]中，XuHui 根据 WebGIS 地理信息系统中文件的数据特征提出了一种文件存储方案。其设计思想是根据系统小文件内容中地理位置之间的位置关系，将这些文件进行分类并进行连续存储，然后将小于系统内设定文件阀值的文件进行合并，合并方式是通过向已存在大文件采用追加的方式，同时构建所有文件的哈希索引，用来增加小文件的快速定位和灵活性。Bo Dong 等人在文献[54]内为 BlueSky 系统内产生的海量 PPT 小文件设计一种特定的存储方案，该方案根据用户对系统内文件的本地性和访问请求，首先将同一课件中的图片等内容合并为大文件，然后分析用户的历史访问信息挖掘文件的相关性，最后通过将相关性较高的文件包含的数据块及数据块索引文件全部预取至内存，从而达到提升用户小文件的浏览速度目的并且可以减少 NameNode 节点的负载问题。在文献[55]中根据小文件的结构和逻辑上的属性关系设计另一种存储优化方案。根据属性关系将小文件分为多个类型进行处理，同一类型大文件被分割成的片段文件是在结构相关的小文件，这些小文件之间具有从属关系但是文件之间又相对独立，在小文件合并的过程中，使用空白文档对于低于 HDFS 内数据块大小或者合并后发生溢出的大文件进行填充。文献[56]针对文献[55]的小文件存储方案提出了一种提高合并后小文件检索以及读取速度的方案。该方案

通过小文件合并方案减少元数据数量，同时在 NameNode 节点中建立合并大文件的哈希表结构，用以维持数据块中文件的顺序，并在哈希表中保存文件之间的位置映射。上述方案利用有序存储的特点，NameNode 节点请求的次数在小文件检索过程中得到很大程度的减少，从而减轻了 NameNode 节点的系统开销。文献[57]根据文献[55]的存储方案设计一种适合医院分析场景的小文件合并算法，该文献的小文件合并策略对文献[55]内采用的空白文件填充方式进一步优化，每次选取合并大文件中剩余空间最多的文件添加小文件，向 HDFS 中存储体积超过 95%以上的合并大文件，更加合理利用 HDFS 的数据块空间。Ankita Patel 等人在文献[58]设计的小文件存储策略不针对某一场景，提出一种通用性的小文件存储策略，主要对系统内映射文件和元数据设计预取和缓存算法，并对小文件提出一种读写策略，测试结果表明该方案 NameNode 节点内元数据存储空间下降 10%。

综上所述，虽然在解决 HDFS 小文件存储问题上取得一定成果，但是也存在一些不足之处，比如小文件的判断标准无法考证。Bo Dong 和 Xuhui 等人设计的小文件存储方案都是根据特定的场景制定，无法运用到所有的存储场景中。Ankita Patel 等人提出的通用性小文件合并策略虽然可以在一定程度上降低 NameNode 节点内存压力，但是由于未建立合并大文件的相关规则，导致缓存和预取都是一个未知数。本文则分析文件名的内容获取文件标签信息，并作为小文件合并的依据，结合大文件之间的关联性设计缓存和预取算法，提高 HDFS 光盘库的系统性能。

2.3 本章小结

本章主要介绍 HDFS 光盘库的总体结构以及其内部的子模块实现方式，并介绍了 HDFS 自带的小文件存储策略以及其他改进策略。

第三章 磁盘缓存系统的设计

3.1 系统需求分析

在上一章对 HDFS 光盘库的总体结构及实现过程进行详细介绍。由于光盘库自身物理结构的原因，光盘库在读取文件的时候，需要通过机械臂将光盘放入光驱内，这一过程达到秒级。所以光盘库在读取一些不是很大的文件，光盘库取盘时间大于读盘时间，效率非常低。HDFS 不能将低于系统内默认数据块的小文件做分割，如果 HDFS 对小文件采用和其他文件相同的方式存储，需要在 NameNode 节点内保存其元数据信息，当面临海量小文件的时候，NameNode 节点上会产生海量的元数据，导致 NameNode 节点的内存消耗加快从而限制 HDFS 系统的容量，另外在访问小文件的时候，海量元数据信息导致查询速度过慢。因此 HDFS 光盘库设计一种小文件存储方案保证 HDFS 光盘库内文件均大于数据块，同时把合并大文件的索引信息保存在 NameNode 节点内，将数据保存在 DataNode 节点内的光盘库内，从而解决小文件在 HDFS 光盘库存储遇到的问题，并在 DataNode 节点内设置块文件管理策略、块文件分发策略和 I/O 调度策略实现 HDFS 文件系统和光盘库之间的融合。HDFS 光盘库相对于传统光盘库在存储容量和数据传输速度方面已经有了很大改善，但是仍然存在以下不足之处。

1) 合并大文件内小文件之间的关联性较低。为了解决小文件存储问题，HDFS 光盘库将一段时间内的小文件在 Client 内缓存并合并为大文件，但是仅仅解决 NameNode 节点内存消耗的问题，合并大文件内和文件之间的关联性较低。

2) 小文件查询速度较慢。HDFS 光盘在小文件合并的时候建立索引信息，并将其引保存在 NameNode 内，当面对海量小文件的时候将产生海量索引信息，访问小文件的时候需要遍历索引链表，导致索引信息的查询速度较慢。

3) 读写速度相对较慢。随着光盘技术的发展，光盘库在存储容量和数据传输速度得到了很大提升，HDFS 光盘库通过分布式存储和光驱并发使得数据传输速度得到进一步的提升，但是 HDFS 光盘库的数据传输速和磁盘相比仍然存在很大差距。

4) 未对 Client 缓存内文件进行管理。HDFS 光盘库仅仅对存储的小文件的进行缓存，当 Client 内缓存空间的容量不足时，需要将缓存空间内文件淘汰，采用缓存替换算法可减少刚刚淘汰的文件在短期内再次被访问的可能性。

5) 系统内文件无法修改。由于 HDFS 光盘库内采用一次刻录不可擦除的方式刻录，因此系统内文件无法修改。

磁盘缓存系统的设计研究是为了尽量缩短 HDFS 光盘库和磁盘阵列之间的差距，因此其应具有以下特点和功能：

1. 建立文件之间的关联性。HDFS 光盘库根据时间局部性原理，仅仅对一段时间内文件进行小文件合并，但是文件之间的关联性较低，会导致连续的文件访问在不同的光盘上，为了将文件访问尽量集中在同一光盘上，减少光盘库取送盘操作，因此需要进一步提高文件之间的关联性。为了保证系统内文件的安全性，且文件类型包括文本、表格、音频，图片等，因此无法对文件内容进行分析。文件名即是区别系统内不同的文件又是对文件内容的概述，因此可以通过文件名对文件名进行分类，建立文件之间的关联性。

2. 小文件处理功能。当存储小文件的时候，由于 HDFS 文件系统和光盘库的物理结构的原因，存取性能比较低，因此 HDFS 光盘库对小文件进行单独处理，在 Client 缓存区域内对小文件进行缓存并合并为大文件。但是小文件之间的关联性较低且 Client 内缓存容量有限无法缓存大量的小文件，所以可以在磁盘缓存系统内对小文件进行处理，为此必须记录磁盘缓存内待刻录的小文件，并根据文件之间的关联性设计合并策略，保证合并大文件中小文件之间的具有较高关联性，还可以改进 HDFS 光盘库中小文件索引的构造方式，提高小文件索引的检索速度，从而进一步提高 HDFS 光盘库系统性能。

3. 文件管理目录。HDFS 光盘库的其中一大特点是容量非常大，具有良好的扩展性。因为其容量较大，所以磁盘缓存系统的容量也很大，磁盘缓存内文件数量也非常大，因此需要建立文件管理目录对磁盘缓存内文件进行管理，提高磁盘缓存系统内文件查询速度。

4. 缓存替换算法。磁盘缓存系统主要有两个功能，第一是对用户上传的文件进行文件缓存，并在磁盘缓存系统内对文件进行预处理，第二磁盘缓存系统负责与用户之间通信，并为用户直接提供文件访问的服务。因此磁盘缓存系统内的文件决定系统的整体性能，但是由于磁盘缓存系统的容量有限，不能无限制地向磁盘缓存内存入文件，当磁盘缓存的容量达到系统设置阀值的时候，需要设计一种缓存替换算法将磁盘缓存内近期不会使用的文件迁移至 HDFS 光盘库内，减少系统对 HDFS 光盘库的访问次数，进而使得系统的整体性能进一步提高。

5. 文件预取功能。文件预取是另一种提高系统性能的手段，将系统接下来可能访问的数据预取至缓存内提高缓存命中率，达到降低用户访问响应时间的目的。因此本文可根据文件之间的关联性设计一种文件预取策略，提高磁盘缓存系统的缓存命中率。

3.2 系统总体设计

根据上节的分析结果设计了如图 3.1 所示的磁盘缓存系统结构。磁盘缓存系统位于 HDFS 光盘库和用户之间。磁盘缓存系统对外提供标准 POSIX 文件读写接口，负责对用户请求及时响应，并通过标签分类算法建立文件之间关联性，通过虚拟存储模块将小文件合并为大文件，通过 Cache 模块对磁盘缓存系统内文件进行管理。HDFS 光盘库负责存储经磁盘缓存系统处理过的大文件，并向磁盘缓存系统内虚拟存储模块提供标准的读写接口。

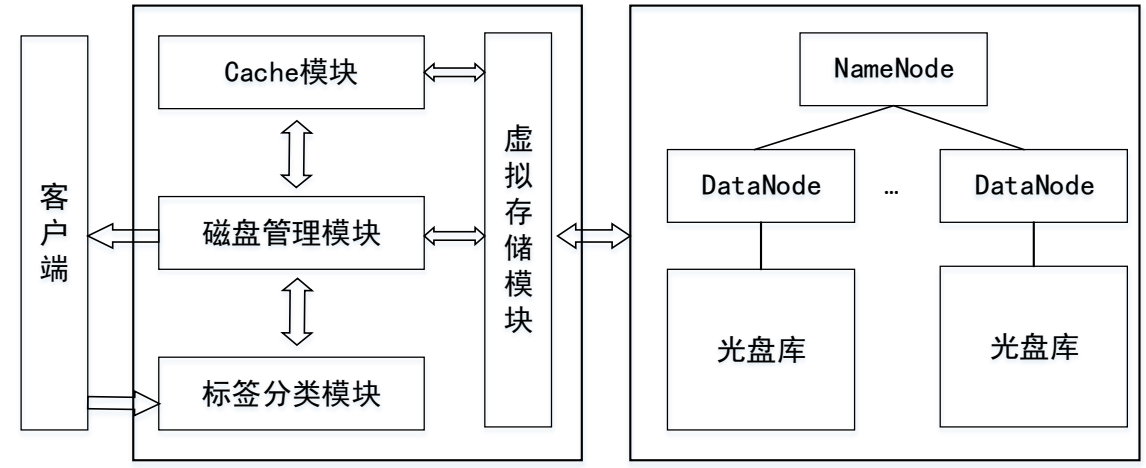


图 3.1 磁盘缓存系统结构图

磁盘缓存系统组织结构如图 3.2 所示，其内部主要包含磁盘管理模块、标签分类模块、Cache 模块和虚拟存储模块。磁盘管理模块对磁盘缓存内文件进行统一管理，并为用户提供读写服务。标签分类模块主要负责对系统新上传的文件和标签为扩展名的文件根据文件标签分类算法为每个文件贴上标签，建立文件之间的关联性，主要由关键词数据库、关键词提取模块和标签分类算法组成。虚拟存储库模块在磁盘缓存系统内模拟 HDFS 光盘库，其中小文件合并模块和小文件映射模块负责将小文件合并为适合 HDFS 光盘库存储的大文件，同时创建两者之间的索引文件并将其保存在 NameNode 节点内，小文件提取模块主要完成将 HDFS 光盘库内合并大文件恢复至磁盘缓存系统内。Cache 模块通过缓存替换算法对磁盘内文件进行动态更新，并通过文件之间关联性设计文件预取策略，进一步提高系统性能。

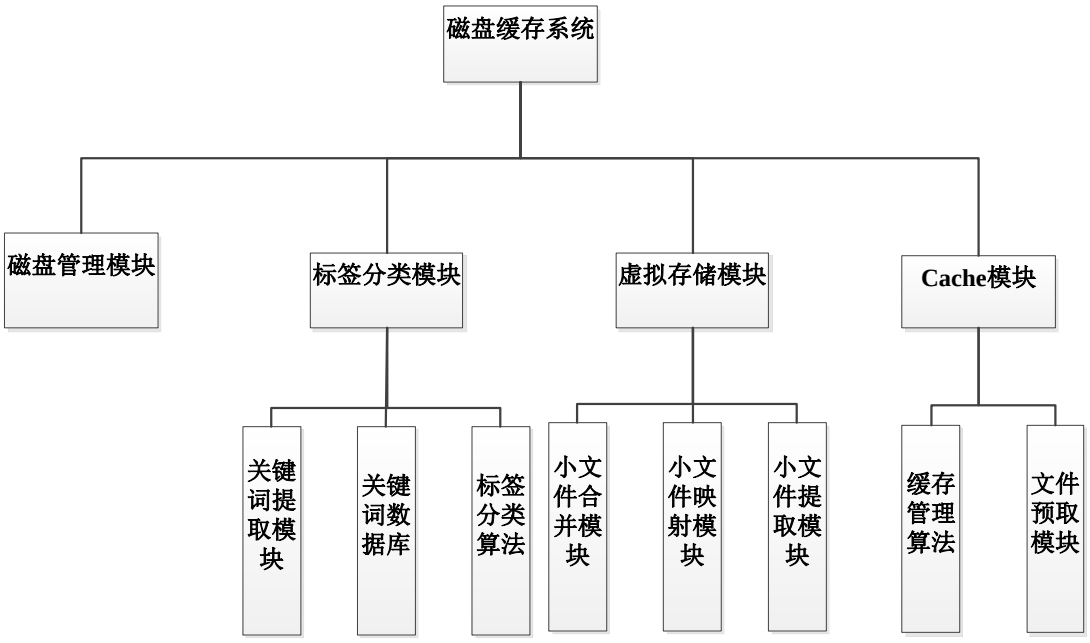


图 3.2 磁盘缓存系统组织结构图

HDFS 光盘库主要包括 Client、NameNode 节点和 DataNode 节点。NameNode 负责与磁盘缓存系统内虚拟存储模块之间通信，保存所有文件的元数据信息和索引文件信息，以及 DataNode 节点管理和任务分发。DataNode 负责维护光盘库的数据块的索引信息，通过块文件分发模块、块文件预取模块以及缓存管理模块对光盘库内数据块的进行管理。

当磁盘缓存系统接收用户请求时，流程如图 3.3 所示，磁盘缓存系统根据用户请求分为文件上传请求和文件访问请求。对于用户上传请求，系统首先通过标签分类模块为每个文件贴上标签，然后将文件在虚拟存储模块下缓存，当满足刻录条件的时候，将文件存入 HDFS 光盘库内。对于文件访问请求，通过磁盘管理模块检查磁盘是否存在备份，如果找到该文件，响应用户请求，若未找到文件，发生磁盘访问中断，由虚拟存储模块将用户访问请求转发到 HDFS 光盘库并触发文件预取。

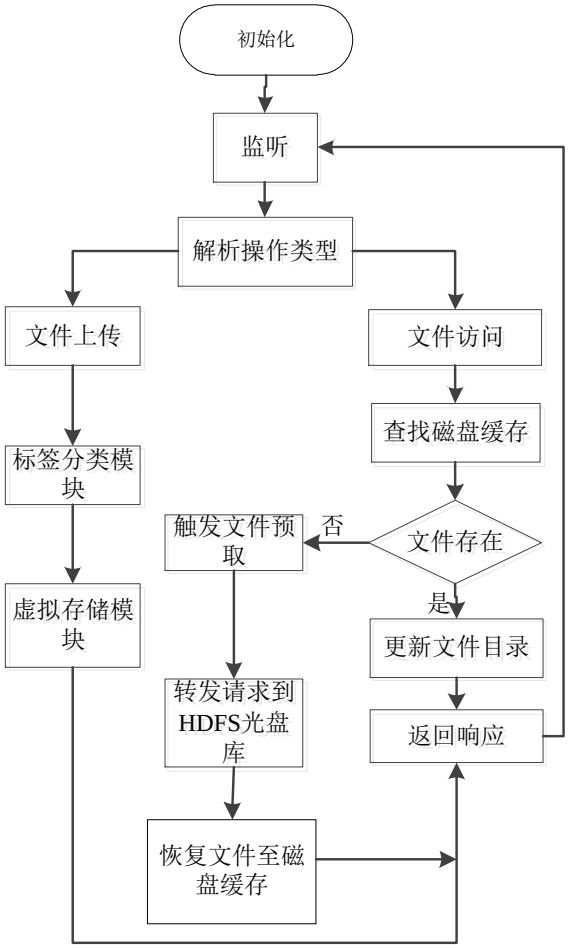


图 3.3 磁盘缓存系统流程图

3.3 标签分类模块的设计

针对 HDFS 文件系统读写小文件效率不高的缺点，目前大多采用小文件合并处理策略[59]。本文采用了一种基于文件标签的小文件存储策略解决小文件存储问题，文件标签由两种方式获

得，第一种是在用户上传文件的时候，可以主动选择文件标签类型，第二种则是通过标签分类模块获取文件标签信息。标签分类模块是磁盘缓存系统的核心内容，其主要功能是根据标签分类算法为每个文件贴上标签。由于文件种类多样，除文本文件外还包括图片、表格、音频、二进制文件等，因此无法全部通过自然语言处理技术[60]获取标签，且考虑数据的安全性，系统无法访问文件内容信息，文件名作为文件在系统内的标识也是文件内容的概述，非常适合作为标签分类模块的分析依据。为了获取文件标签信息，本文提出了一种基于文件名的标签分类算法。

3.3.1 关键词数据库的设计

关键词数据库中维护系统内全部关键词信息，主要有两个功能。第一个是在文件名分词中作为分词词典使用，第二个功能是标签分类过程中，标签分类算法依据本文设计的关键词标签概率计算文件名标签概率，然后获取文件标签。关键词的标签概率依据公式 3-1 计算获得，公式如下所示：

$$P_{i,j}=\frac{n_{i,j}}{n_j}\times 100\% \qquad (3\text{-}1)$$

其中 $P_{i,j}$ 表示 j 关键词内 i 标签概率值，$n_{i,j}$ 表示 j 关键词内 i 标签的 Count 值，n_j 表示 j 关键词内所有标签的 Count 值总和。

关键词数据库内关键词由两部分组成，第一部分是在系统创建之时，程序员根据客户存储需求和存储领域背景知识，按一级标签类型构建如表 3.1 所示的关键词表，表中记录关键词名称、关键词所属二级标签名称和关键词在标签内出现次数，由表 3.1 能够看出，关键词最多包含三个标签。关键词在首次加入关键词表的时候，加入关键词所属二级标签信息，可加入 1 到 3 个标签信息，并初始化对应 Count 值，本文设置 Count 初值为 1000，Count 值随着系统运行动态变化，且不会因为少量的错误文件导致关键词的标签概率发生较大变化。第二部分是在系统运行过程中，根据文件标签信息和关键词信息对关键词数据库内容进行更新，下面将对这一部分进行具体介绍。

表 3.1 关键词表

Name	Label1	Count1	Label2	Count2	Label3	Count3
篮球	运动	1000	商品	1000	NBA	1000
足球	运动	1000	商品	1000	世界杯	1000
口红	化妆品	1000	商品	1000		
.....	.....	.....	.....	.....	.....	.....
孙中山	人物	1000	历史	1000		

关键词数据库的更新流程如图 3.4 所示，客户在上传文件时可选择文件标签及文件关键词，

标签分类模块根据文件信息对关键词数据库进行更新。若文件包含标签和关键词信息，查询关键词数据库。若文件标签和关键词在关键词表中已存在，则更新关键词对应的标签的 Count 值；若关键词存在但标签不存在，且当前关键词的标签未满 3 个，则将标签信息加入关键词中，标签的 Count 值为关键词其他所有标签 Count 值的平均值；若关键词不存在，则向关键词表内插入关键词信息，然后将标签信息加入关键词信息内，标签的 Count 值为 1000。若用户仅选择文件标签信息，则根据关键词提取模块获取文件名内关键词信息，并更新对应的关键词的标签 Count 值。若用户仅选择文件的关键词信息，则根据关键词数据库查询标签信息，并更新对应的关键词的标签 count 值。若用户未选择文件标签和关键词信息，通过标签分类模块获取文件标签和关键词信息，并更新对应的关键词的标签 Count 值。

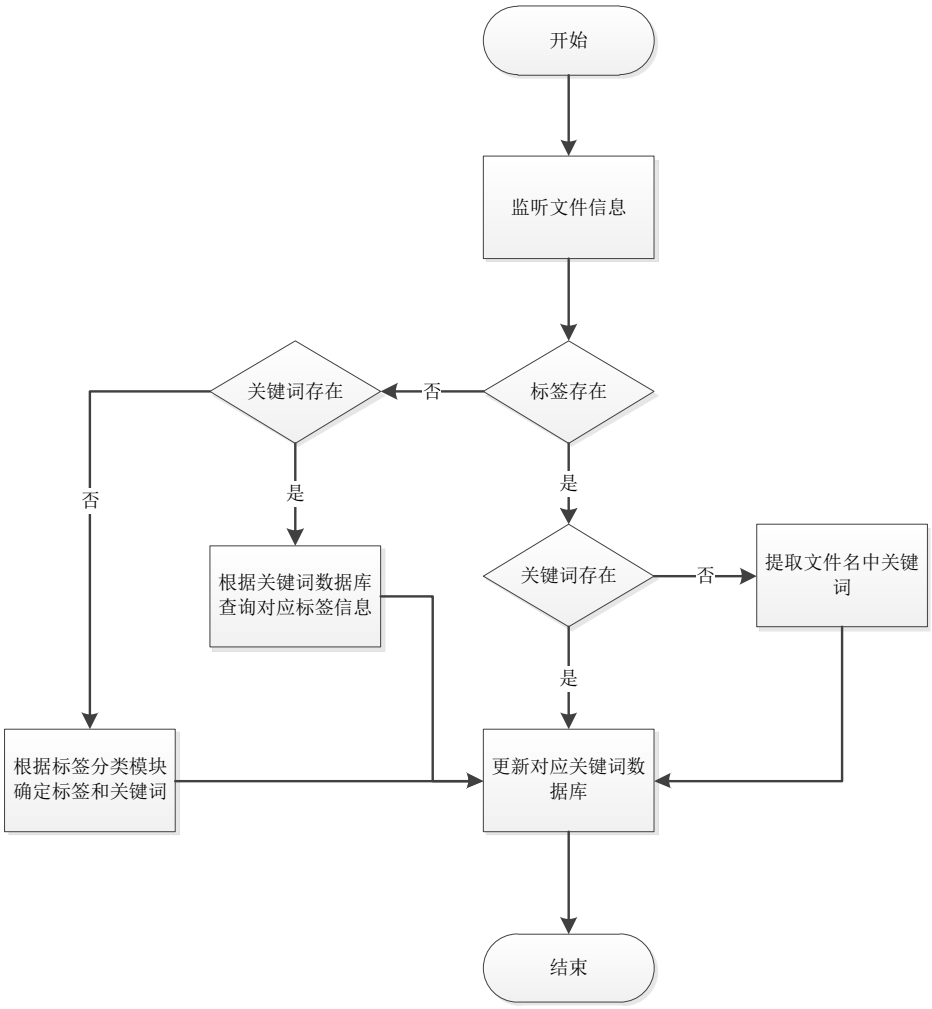


图 3.4 关键词数据库更新流程图

因此在系统创建初期，本文采用带有标签信息和关键词信息的文件集合对标签数据库进行完善，更新关键词内二级标签 Count 值比例，保证其概率值的准确性，随着系统运行，关键词数据库内关键词的数目将会不断壮大，且关键词内的二级标签的 Count 值也随着系统的运行不

断更新，关键词数据库内数据不断完善，标签概率的准确率也越来越高，因此关键词数据库的创建和训练集的选择至关重要。

3.3.2 关键词提取模块的设计

系统内文件不仅仅包含文本文件，还包含图片、表格、音频、二进制文件等，为了保证系统内文件数据的安全性，我们无法通过分析文件内容获取文件标签，因此本文设计一种基于文件名的标签分类算法。关键词提取模块通过对文件名进行分词，提取文件名中有效的关键词，然后根据关键词的标签概率的计算结果获取文件标签，所以关键词提取结果的准确性至关重要。

英文分词相对中文分词最为简单，因为英文的单位是单词，空格是英文中单词和单词之间的自然分界符，而中文词与词之间无明显界限，所以本模块主要对系统内中文分词策略做具体阐述。目前中文分词方式总体上主要有理解式分词法、机械式分词法和基于频度统计的分词法三种方式，比较常用的算法有正向最大匹配算法[61]、逆向最大匹配算法[62]、基于词频的统计方法[63]等。正向最大匹配算法的基本思想是：如果分词词典内最长词语长度为 n 个字符，首先从当前处理的文本中的最前端选择 n 个汉字字段，然后查询分词词典内是否包含该字段的词语。若分词词典内包含该字段则表示匹配成功，将该字段从文本中提取出来作为文本词语，并将文本按该字段分为两个新的文本分别进行分词处理，若分词字典内不包含该字段表示匹配失败，则将该字段的末尾字删除，然后对剩余字段重复上述操作直至字段为空。然后顺序取下一个 n 个字段再次查询分词词典，直至文本内容全部选择完。逆向最大匹配法和正向最大匹配算法不同之处是每次删除字段的第一个汉字。基于词频的统计方法则是通过计算字与字共同出现的频率来确定是否成词语，若字和字共同出现的频率大于系统设定的默认阀值，则将该字组作为一个词语处理。

据统计，单纯使用正向最大匹配算法的准确率高达 99.40%，其在大部分情况下是合理的，但是在部分情况下会歧义字段。针对上述不足之处，结合系统内文件名提取标签信息的需求和分词效率对正向最大匹配算法进行改进。首先在系统内需要创建停用字表排除停用字对关键词提取的影响，然后通过关键词提取算法提取文件名内关键词，其流程如图 3.5 所示。关键词提取算法具体操作如下：

(1) 从文件名内按字符指针顺序选择下一个汉字，如果指针指向文件名末端则关键词提取过程结束。

(2) 查询系统内停用字表，如果当前汉字是停用表内汉字，则将字符指针后移一个字符，转向步骤（1）。

(3) 从关键词数据库中筛选出以该字为词首的词语集，并按关键词的长度和关键词中二级标签的最大概率值进行降序排列。

(4) 从词语集中依次取出词语，然后从文件名中以该字开始依次截取与匹配词语长度相当的

字符串，依次同词语集中的词语相匹配。

(5) 如果当前选择的词语和文件名匹配失败，则从词语集中顺序选择下一个词语重复执行步骤（4），若词语集为空转向步骤（1）。若在某一个词语和文件名匹配过程中匹配成功，将该词语从文件名中提取出来，并将该词语加入文件名的关键词中，将文件名的剩余部分作为两个新的文件名，转向步骤(1)分别进行关键词提取。

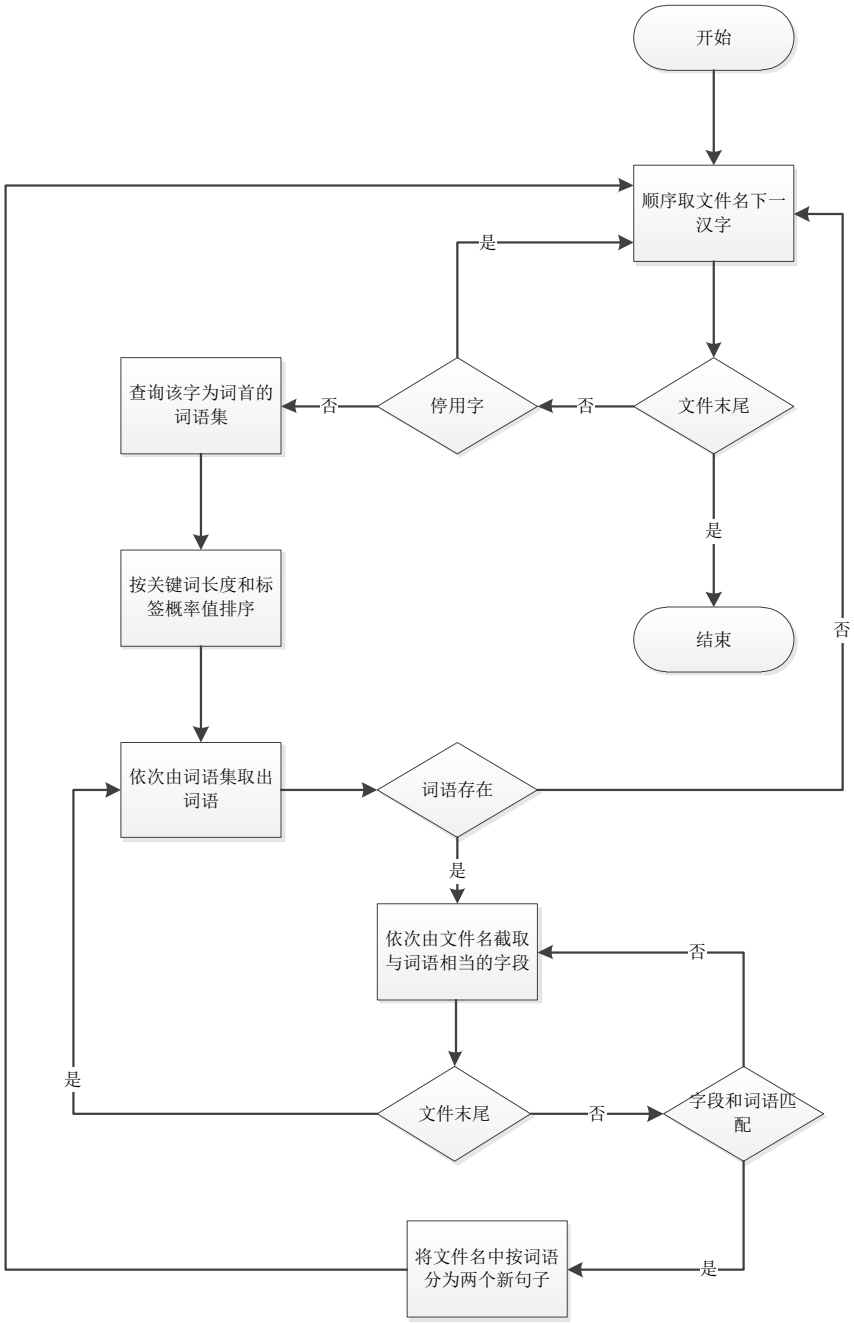


图 3.5 关键字提取流程图

3.3.3 标签分类算法的设计

标签分类算法是标签分类模块的核心内容，标签分类算法主要根据文件名的关键词、文件访问记录和文件扩展名确定文件标签。标签分类模块主要对用户新上传的文件和磁盘缓存系统内淘汰的标签为扩展名的文件进行处理。对于新上传文件，文件在上传的时候，用户可选择标签和关键词信息，也可不选择任何信息。若文件上传过程中包含标签和关键词信息，标签分类算法根据文件信息对关键词数据库进行更新。若用户未选择标签和关键词信息，则通过关键词提取模块获取关键词，然后根据关键词的标签概率获取文件标签信息，若无法获取根据上述方法获取标签信息，则将文件扩展名作为文件标签。对于标签为扩展名的文件，则通过对文件访问记录获取文件标签信息，若无法获取标签，则作为新上传文件再次处理。标签分类算法的执行流程如图 3.6 所示。标签分类算法具体操作如下：

1. 根据文件类型将文件分为两类，若文件是文件标签为扩展名的文件则执行下一步，若文件为新上传文件则执行步骤 3。

2. 若文件标签为扩展名，则根据文件的访问记录确定文件标签，例如文件 A 未能在上次标签分类处理中确定文件标签，在系统内连续三次访记录中有两次文件 A 都在文件 B 之后，则认为文件 A 的标签和文件 B 的标签相同，若文件 A 最近无访问记录或无法通过上述方法确定标签，则执行步骤 5。

3. 若文件为新上传文件，根据文件上传信息将文件分为两类，若文件上传时包含标签信息或关键词信息则执行步骤 4，若文件上传时未选择文件标签信息和关键词信息则执行步骤 5。

4. 若上传文件信息中包含文件标签，根据文件标签和关键词信息文件信息更新关键词数据库，执行步骤 9；若上传文件信息中仅包含关键词信息则执行步骤 6。

5. 通过关键词提取模块获取文件名所包含的关键词集合，表示如下

$$S=A_1，A_2，A_3，……，A_n$$

其中 A_1，A_2，A_3，……，A_n 表示文件名内所有关键词。关键词为空则执行步骤 8。

6. 将上述文件名内关键词集合按标签种类进行分类，表示如下：

$$S_1= A_1，A_3，……，A_n$$

$$S_2= A_1，A_2，……，A_{n-2}$$

……

$$S_i= A_1，A4，……，A_n$$

其中 S_i 表示文件名内包含 i 标签的关键词集合。

7. 然后依照下面公式 3-2 计算步骤 6 中各个分类词语集的标签概率值。

$$P_i = \sum(\frac{n_{i,j}}{n_j} \times \frac{l_j}{L}) \times 100\% \quad （3\text{-}2）$$

P_i 表示 S_i 分类下的标签概率值，即文件属于 i 标签的概率值，其中 $n_{i,j}$ 表示 j 关键词中 i 标签的 Count 值，n_j 表示 j 关键词在所有标签的 Count 值总和，l_j 表示 j 关键词的长度，L 表示文件名长度。

$$P = \max\{P_1, P_2, \cdots, P_i\} \tag{3-3}$$

根据公式 3-3 选择概率值最大的 S_i 的分类，i 标签即为文件标签，并更新 S_i 分类内关键词对应的 i 标签的 Count 值。执行步骤 9。

8. 若未能通过关键词提取模块获取关键词，说明无法通过上面的方法确定文件标签，则将文件扩展名作为文件标签信息。

9. 标签分类结束。

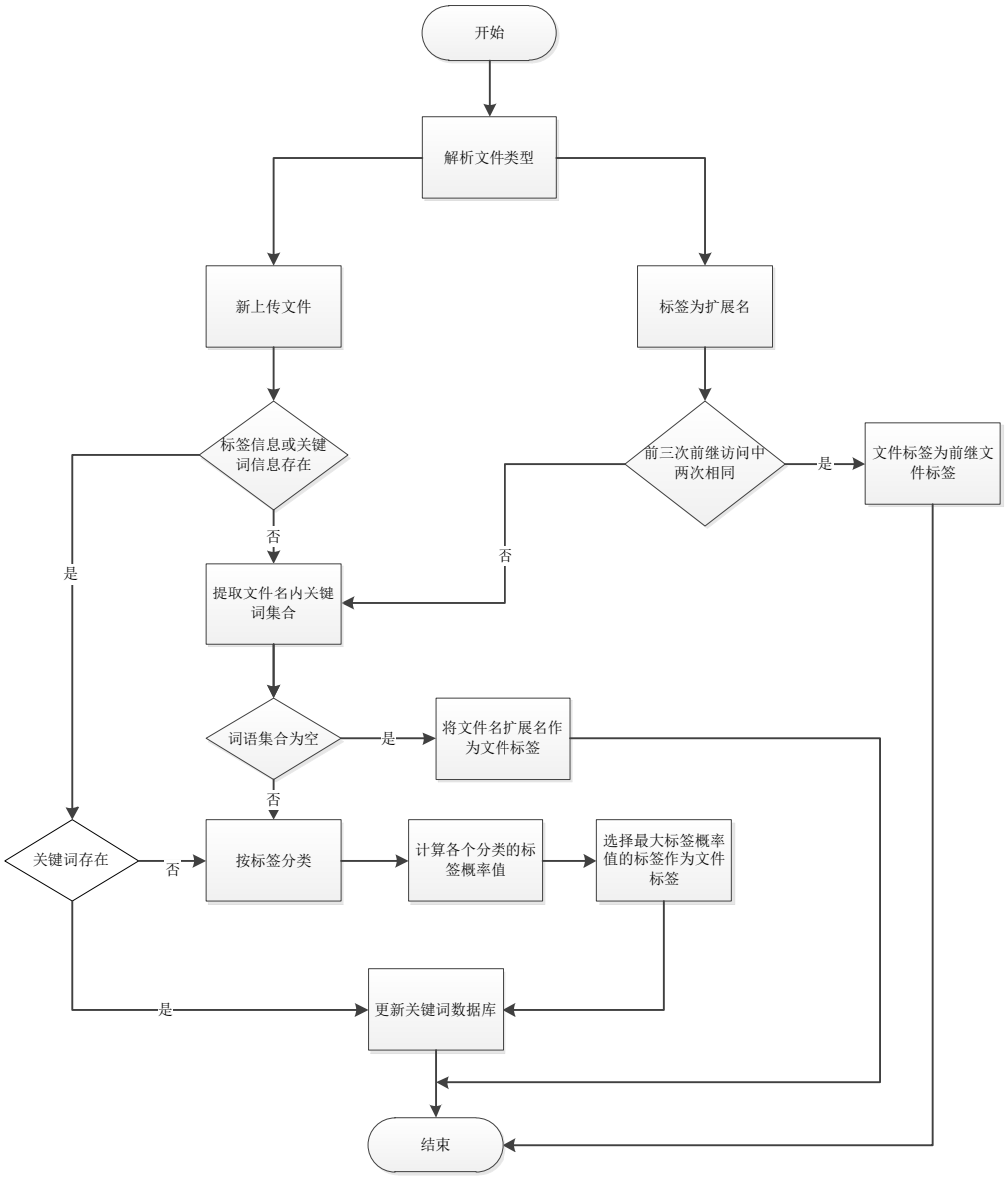


图 3.6 标签分类算法流程图

3.4 虚拟存储模块的设计

本节为了解决小文件存储带来的问题，将小文件合并和小文件提取模块转移至虚拟存储模块内，减轻 Client 的存储压力。虚拟存储模块是在磁盘缓存内模拟 HDFS 光盘库功能，对磁盘内需要刻录的文件进行缓存。虚拟存储模块主要有两个作用，一是降低文件刻录响应时间，磁盘内需要刻录的文件首先交由虚拟模块，这一过程的时间远低于实际刻录时间；二是通过小文件合并提高 HDFS 光盘库内数据传输速度，虚拟存储模块通过标签信息把一段时间具有相同标签的小文件合并为大文件，大文件更加适合 HDFS 光盘库存储。虚拟存储模块根据文件类型采用不同处理策略，将大文件作为独立存储对象写入 HDFS 光盘库，将多个小文件合并为大文件，并建立小文件与大文件之间的映射关系，然后交由 HDFS 光盘库存储，保证 HDFS 内部传输的文件均大于其内部基本块的大小。通过虚拟存储模块访问 HDFS 光盘库内文件时会触发文件预取，减少系统内部磁盘访问中断次数。下面将虚拟存储模块分为小文件合并模块、小文件映射模块、小文件提取模块分别进行详细设计。

3.4.1 小文件合并模块的设计

虚拟存储模块内维护一个如图 3.7 所示的待刻录的标签文件目录。标签文件目录中记录磁盘缓存系统内所有待刻录的小文件信息，标签文件目录包含根目录、一级标签目录和二级标签目录和文件信息。根目录中记录所有一级标签目录信息，一级标签目录中记录所有二级标签目录及目录下文件大小总和，二级目录中记录待刻录小文件信息及目录下文件大小总和，小文件信息包括文件名、文件磁盘路径、一级标签信息、二级标签信息、创建时间、上次物理访问时间、文件类型、时间戳、存储位置和修改位信息。

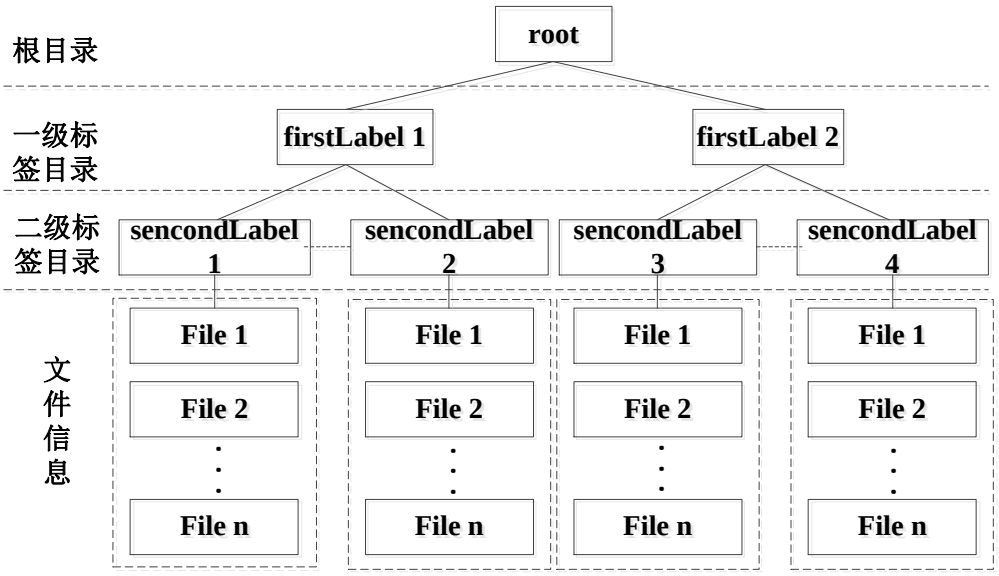


图 3.7 标签文件目录图

当磁盘内某一文件被淘汰的时候，虚拟存储模块根据文件大小将文件分为大文件和小文件，其评判标准为 HDFS 光盘库内数据块大小（128 MB），将未达到数据块容量的文件定义为小文件，反之为大文件。当大文件被淘汰的时候，虚拟存储模块直接将大文件通过 HDFS 光盘库接口将大文件上传至 HDFS 光盘库进行刻录；当小文件被淘汰的时候，根据小文件信息插入对应的文件目录下，然后检测新插入的二级标签目录下所有小文件容量总和，当容量总和超过 128 MB 的时候，将二级标签下除新插入小文件合并为一个大文件，并建立如图 3.8 所示的小文件到大文件索引文件，将合并后大文件和索引文件上传至 HDFS 光盘库完成刻录。索引文件信息包括合并大文件 MD5 值、标签信息、时间戳等。索引文件内容包括：FileMD5 表示小文件在系统内唯一标识，由文件路径、标签、文件创建时间计算获得。FileSize 表示当前小文件大小，offset 表示小文件在合并大文件内的偏移位置。

FileMD5	**FileSize**	**Offest**
小文件 1	120K	0
小文件 2	86K	120
.....	……	……
小文件 n	……	……

图 3.8 索引文件结构图

虚拟存储模块内将索引文件上传至 HDFS 光盘库 NameNode 节点，并在 NameNode 节点内维护索引文件的标签文件目录。当系统读取 HDFS 光盘内小文件的时候，首先根据索引文件目录可以快速找到对应索引文件，根据索引文件获取大文件信息，然后根据大文件信息在 NameNode 内查询数据块存储位置信息，建立磁盘缓存系统和 DataNode 之间的通信，将数据块和索引文件恢复至磁盘缓存空间内。

为了防止部分标签下的小文件长期无法达到刻录条件而长期保存在磁盘阵列内，系统会在系统空闲时定期对虚拟 HDFS 光盘库根目录下的文件进行清理，通过对一级标签下的所有小文件大小进行统计，当文件总大小达到刻录条件时，将一级标签下的所有小文件合并，按一级标签信息存入 HDFS 光盘库内。另外为了进一步提高光盘库的数据传输速度，对块文件分发模块中数据块的分发策略进行改进，将不同时间段内具有相同标签的数据块刻录在同一张光盘上，相同标签的文件之间具有较强的关联性，因此在读取文件的时候，可以很大程度上减少机械臂的使用次数。

3.4.2 小文件映射模块的设计

HDFS 在 NameNode 节点内将文件元数据信息存储在节点的名称空间 NameSpace 中。为了实现对小文件的管理，必须将合并大文件的索引文件信息存储在 NameNode 节点内。本节在

NameNode 节点内建立如图 3.7 所示的索引文件目录 IndexFileDirectory，IndexFileDirectory 与磁盘缓存系统内文件目录有一定差距，IndexFileDirectory 是针对索引文件建立的标签文件目录，为了提高索引文件的查询速度，所在将二级标签目录下索引文件信息按时间戳大小进行排序。磁盘缓存系统在上传合并后大文件的时候，不仅需要在 NameSpace 内建立大文件和数据块之间映射关系，而且需要将大文件和小文件之间的索引文件保存在 NameNode 节点内，并将索引文件信息插入文件目录 IndexFileDirectory 内，索引文件记录组成大文件的所有小文件信息和合并后大文件信息。当用户访问的小文件不在磁盘缓存系统内，将文件访问请求转发至 HDFS 光盘库内，HDFS 光盘库首先根据文件标签信息快速定位二级标签文件目录，根据小文件时间戳信息通过二分法可在索引文件目录中快速找到对应索引文件信息，然后根据索引文件内合并大文件信息，通过 NameSpace 内大文件与数据块之间映射关系查询所有数据块存储位置信息，最后将大文件所有数据块和索引文件通过 HDFS 自身接口下载至磁盘缓存空间内。

3.4.3 小文件提取模块的设计

当系统访问小文件不在磁盘缓存系统的时候，小文件提取模块负责将 HDFS 光盘库中读取的合并大文件进行分割。在小文件合并模块中将具有相同标签内的文件合并为一大文件，再存入 HDFS 光盘库内，且合并大文件是同一时间段内淘汰的文件，因此同一大文件下内的小文件具有较强的关联性，因此当小文件发生访问中断时，本文采用将小文件所属的合并大文件预取至磁盘缓存内，通过小文件提取模块进行恢复。系统在读取 HDFS 光盘库内小文件的时候，首先磁盘缓存通过虚拟存储模块将小文件访问请求转发至 NameNode 节点，NameNode 节点在小文件索引文件目录内查找对应大文件信息，再通过 NameSpace 获取大文件存储位置信息和大文件内所有小文件索引信息，最后通过 HDFS 光盘库和虚拟存储模块之间的读写接口将合并大文件所有数据块的和索引文件下载到磁盘缓存中，然后根据索引文件将小文件从大文件中恢复至磁盘缓存系统内。虽然将大文件中所有小文件恢复至磁盘缓存内增加了小文件访问时间，但是通过文件预取可以很大程度的减少小文件的磁盘访问中断次数。

3.5 Cache 模块的设计

3.5.1 文件预取模块的设计

预取技术[64]和缓存技术是提高系统性能的两种重要方法，预取技术是预测即将访问的数据，并将这些数据批量的存入缓存空间内。在虚拟存储模块内按文件大小分为大文件和小文件，对小文件采用单独处理方式，将最近一段时间内具有相同标签的小文件合并为大文件，具有相同标签文件之间在内容上十分相似，具有较强的关联性，且合并大文件中小文件是同一时间段内淘汰的文件，在某一小文件被访问的时候，与其位于同一大文件下的小文件被访问的概率很高。若当前访问文件不在磁盘缓存系统，需要访问 HDFS 光盘库将文件恢复至磁盘缓存系统，

同时触发文件预取。文件预取流程如图 3.9 所示，文件预取按当前访问信息分为大文件访问和小文件访问，若当前访问为大文件，首先根据大文件信息在 NameSpace 查询对应的大文件所有数据块存储位置信息，在用户读取某一数据块时，将其他数据块预取至磁盘空间内，同时检索索引文件目录与该大文件时间戳差距最小的合并大文件索引信息，将合并大文件通过 HDFS 光盘库自身接口恢复至磁盘缓存系统。若当前访问为小文件访问，与该小文件处于同一合并大文件下的小文件之间拥有较强关键性，在短时间内很大概率被访问，因此将合并大文件中所有小文件全部预取至磁盘缓存系统。由上可以看出，磁盘缓存系统和 HDFS 光盘库之间仅以大文件方式进行数据传输，采用大文件作为调度单位，可以避免 HDFS 和光盘库存储小文件效率不高的问题，并且可避免多次调用 HDFS 光盘库，从而达到降低 HDFS 光盘库内 NameNode 和 DataNode 节点系统开销的目的。

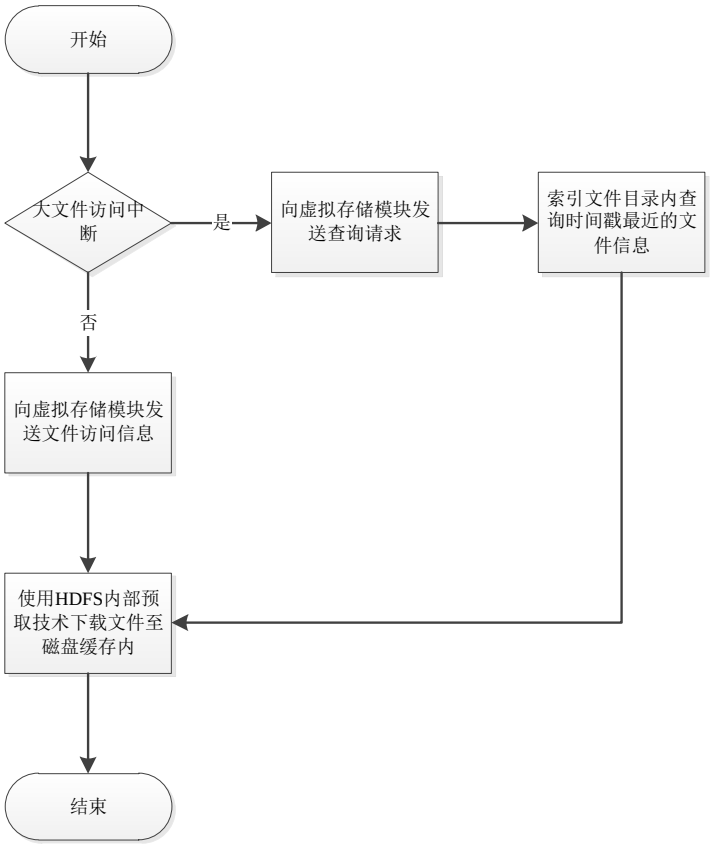


图 3.9 文件预取流程图

3.5.2 缓存替换算法的设计

磁盘缓存系统作为用户和 HDFS 光盘库之间缓存，直接为用户提供文件存储和访问服务，因此磁盘缓存系统内文件越多用户访问速度越快，但是磁盘阵列的空间有限，不能无限制的向磁盘缓存系统内存入文件，所以当磁盘缓存系统容量不足的时候，必须将部分文件替换出磁盘缓存。HDFS 光盘库的一大特点是其容量非常大，所以本文设计的磁盘缓存系统的容量也非常

大，大容量的缓存带来高额的管理成本，其缓存替换算法复杂度也会随着容量的增加而增加，因此其缓存替换算法不宜采用太复杂的算法。

目前针对二级缓存的替换算法的研究取得较好的成果，其中相对成熟的算法有 2Q 算法、LRU 算法、MQ 算法、LRFU 算法、LFU 算法等。由于磁盘缓存和 HDFS 光盘库之间采用大文件作为调度对象，调度文件量级较大。对于上述几种缓存算法的性能上相差不多，而 LRU 算法实现起来相对比较容易，且系统开销不大，所以本文在传统 LRU 算法的基础上设计基于文件标签的 LB-LRU（Label Based Least Recently Used）算法对磁盘缓存的文件进行管理。

Cache 模块内保存一个记录近期访问文件和预取文件信息的双向链表，每个链表项记录该节点的标签和文件信息，其中包含当前节点内文件类型，一级标签信息、二级标签信息、时间戳和当前节点所有文件信息的单向链表，链表结构如图 3.10 所示。Cache 缓存文件链表按节点内文件类型划分为大文件节点、小文件节点和合并大文件节点，合并大文件节点中记录合并大文件中全部小文件信息，小文件节点和大文件节点内文件信息链表内仅包含一个文件信息。

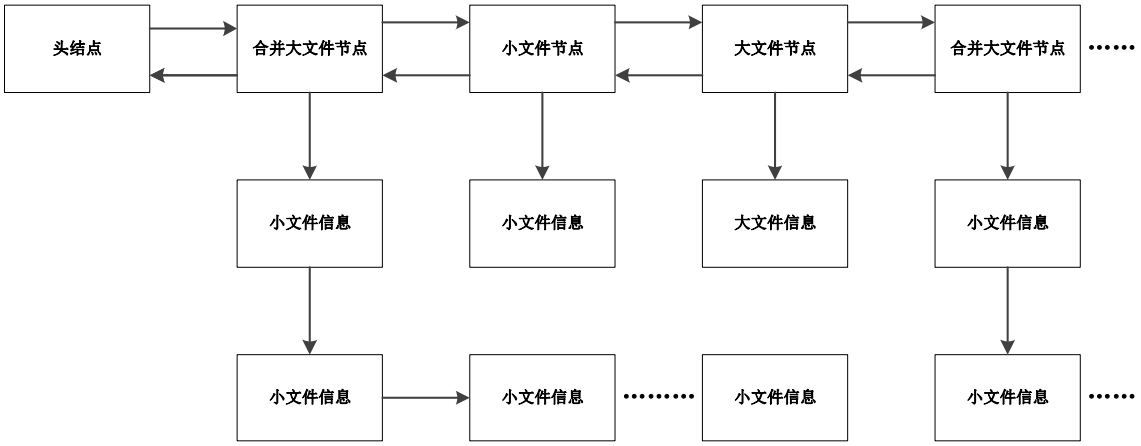


图 3.10 Cache 缓存文件链表结构图

每当系统收到一个访问请求：

Step1：查询磁盘管理模块标签文件目录查找文件信息，若通过磁盘管理模块标签文件目录未找到文件信息则执行 step2，否则响应用户请求并执行 step5。

Step2：触发 Cache 模块内文件预取模块，根据当前访问信息确定预取文件内容。

Step3：虚拟存储模块转发预取文件信息到 HDFS 光盘库。HDFS 光盘库首先通过预取文件信息找对应文件信息，然后根据 HDFS 光盘库内部数据块预取策略将预取文件备份至磁盘缓存内响应用户需求。

Step4：将合并大文件内小文件恢复到磁盘缓存内，并更新缓存根目录和对应文件信息，向 Cache 缓存链表顶端插入文件信息，执行 Step7。

Step5：检索 Cache 缓存文件链表，若找到文件节点，将该节点移动到链表顶端执行 step7。

Step6：创建一个新的链表节点，将当前文件和与该文件具有相同时间戳的文件信息加入节点内，并将该节点插入链表顶部。

Step7：检查 Cache 模块空间，若空间不足时从链表尾部开始淘汰文件，直到 Cache 模块空间达到安全值，访问结束。

随着系统的运行，最近一段时间内被访问的大文件和小文件信息都被移动到链表的前面部分，而最近一段时间内未被的访问文件将逐渐下沉到链表的末端。当磁盘缓存系统内存不足的时候，从链表的尾部开始删除文件，当被删除的文件为合并大文件的时候，通过文件信息链表查找当前节点内所有小文件进行淘汰。Cache 缓存链表内被淘汰的小文件分为三种，第一种是已经刻录且在磁盘缓存内未发生修改的文件，将文件从磁盘缓存内删除备份，更新数据库内文件信息；第二种为未完成刻录的文件，只需更新对应文件信息；最后一种是 HDFS 存在备份但是在磁盘缓存内使用过程中发生修改的文件，对于该类型文件需要作为新文件交由虚拟 HDFS 光盘库重新刻录，并对原文件信息进行覆盖。

3.6 本章小结

本章根据 HDFS 光盘库的结构的分析结果，结合 HDFS 光盘库在实际存储中可能遇到的问题，确定了磁盘缓存系统的功能需求，然后根据功能需求设计磁盘缓存系统的整体结构，将其划分为标签分类模块、虚拟存储模块和 Cache 模块，对三个模块和其子模块进行了详细设计。

第四章 磁盘缓存系统关键模块的实现

4.1 标签文件目录的实现

磁盘缓存系统内保存系统近期访问的文件和新上传的文件，磁盘缓存系统内文件数量较多，因此需要建立文件目录对磁盘内文件进行管理。本文在标签分类模块内根据文件名称为文件贴上标签信息，本文依据标签建立的标签文件目录对磁盘内文件进行管理，根据文件标签信息可快速在标签文件目录内检索到文件信息。磁盘管理模块和虚拟存储模块都建立标签文件目录，负责对磁盘缓存内和待刻录文件进行管理。下面将介绍标签文件目录的实现。

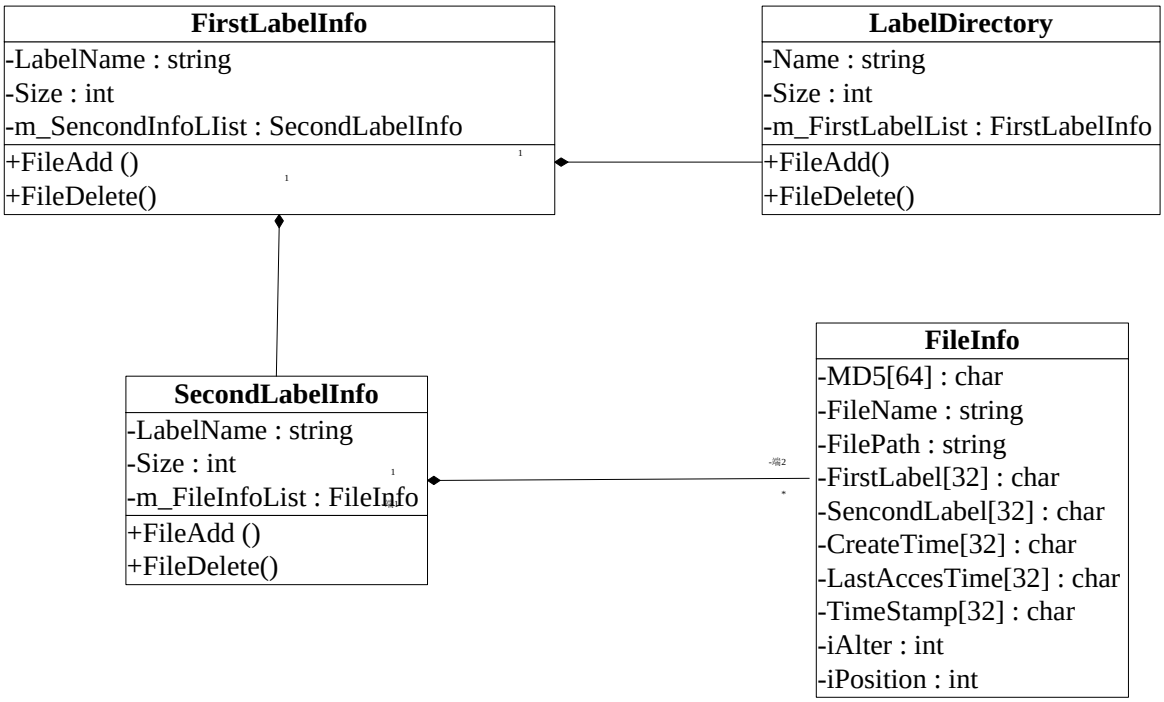


图 4.1 标签文件目录类图

如图 4.1 所示，标签目录文件类 LabelDirectory 是创建标签文件目录对象的具体实现类。FirstLabelInfoList 记录的是标签文件目录包含的所有一级标签目录信息的双向链表，FileAdd 和 FileDelete 函数实现文件的插入和删除。一级标签信息类 FirstLabelInfo 中 SecondLabelInfoList 记录一级标签目录包含的所有二级标签目录信息的双向链表。二级标签信息类 SecondLabelInfo 类中 FileInfoList 记录二级标签目录包含的所有文件信息的双向链表。其中 LabelDirectory 类、FirstLabelInfo 类和 SecondLabelInfo 类中 LabelName 表示标签名称，Size 表示目录下所有文件大小总和，FileAdd 和 FileDelete 函数实现文件的插入和删除。FileInfo 类中 FileName 表示文件名称，FilePath 表示磁盘缓存系统内文件存储唯一路径，FirstLabel 表示文件标签的一级标签信

息，SecondLabel 表示文件标签的二级标签信息， LastAccesTime 表示文件最后一次物理访问时间，CreateTime 表示文件在磁盘缓存系统内创建时间，iAlter 表示文件在磁盘缓存内使用过程中是否发生修改，iPosition 则表示文件在 HDFS 光盘库和磁盘缓存系统的存储位置信息，0 表示只在磁盘缓存系统内有备份，1 表示只在 HDFS 光盘库内有备份，2 表示在磁盘缓存系统和 HDFS 光盘库内均有备份，MD5 值由 FilePath、CreateTime 和 SecondLabel 计算获得，保证文件在系统内的唯一性，TimeStamp 记录文件时间戳，文件在未存入 HDFS 光盘库之前，TimeStamp 为文件在磁盘缓存内创建时间，存入 HDFS 光盘库以后，TimeStamp 更新为文件在 HDFS 光盘库内创建时间。磁盘管理模块和虚拟存储模块通过 LabelDirectory 类创建 DiskFileDirectory 和 VirtualStorageFileDirectory 标签文件目录。

4.2 标签分类模块的实现

标签分类模块根据文件信息为文件贴上标签和更新关键词数据库。为了实现标签分类模块的功能，我们为标签分类模块设计了如图 4.2 所示的 FileClassify 类。AntistopRecord 类记录单个关键词记录，包括关键词名称 AntistopName，关键词的标签信息链表 LabelInfoList，LabelInfoList 链表内记录关键词所属标签信息 LabelInfo，LabelInfo 记录当前二级标签名称 LabelName，一级标签名称 FirstLabelName，关键词在该标签下出现次数 Count。FileNameParticiple 类实现对文件名关键词提取，它包含一个 AntistopRecord 类链表，用于记录关键词的标签信息，通过类中 AntistopAnalyze 函数将文件名中有效的关键词提取出来，并将关键词放入文件名关键词链表 FileNameAntistopList 中。FileVisitRecord 类负责记录在当前文件访问之前的访问文件的信息。最后 FileClassify 类是标签分类模块最重要的类，它为每个文件贴上标签信息，更新关键词数据库的任务，以及将加入标签的文件信息交给磁盘管理模块，其中包含 MySQLObject 用于和关键词数据库通信，包含 FileVisitRecord 类型变量 FileVisitInfo，存储当前文件访问之前的文件访问记录，包含 FileNameParticiple 类型变量 FileNameClassity，用于分析当前文件名的关键词信息并记录。函数 StartFileNameAnalyze 负责启动标签概率计算线程，选择概率最高的标签，函数 StartFileVisitAnalyze 负责启动文件历史记录分析线程，由文件历史访问记录获取文件标签。

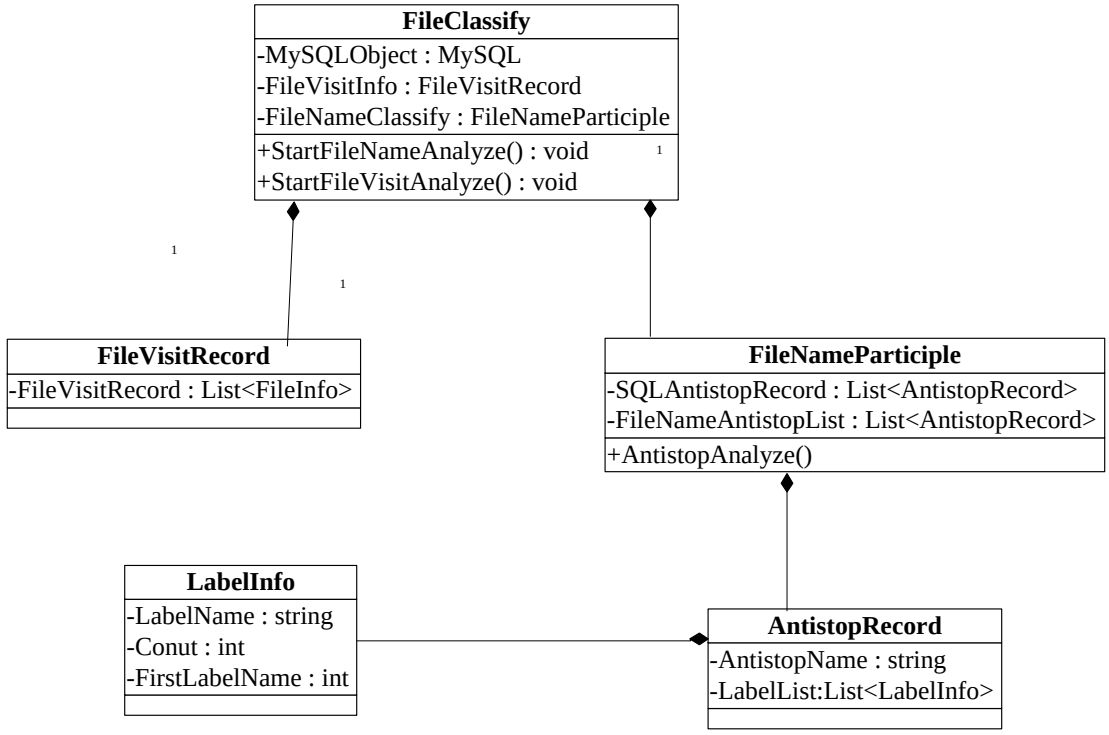


图 4.2 标签分类模块类图

标签分类模块主要对不含标签信息的文件和标签为文件扩展名的文件进行处理，标签分类模块在收到文件信息的时候，根据文件信息判断这个文件是否是新上传的文件，若不是新上传文件，则创建 FileVisitRecord 对象，并查询 FileVisitRecord 对象中 FileVisitRecord 链表内文件信息，若 FileVisitRecord 链表中三个文件有两个文件信息相同，则将该文件标签作为当前处理文件标签，若 FileVisitRecord 链表不存在两个相同文件，则将文件标签删除，作为新上传文件处理。若当前处理的文件为新上传的文件，则通过文件名获取文件标签。检查用户在上传文件时是否包含标签信息，如果包含标签信息，创建 FileNameParticiple 对象并根据标签数据库信息提取文件名内关键词加入 FileNameAntistopList 链表内，然后检查 FileNameAntistopList 链表，若为空结束标签分类任务，否则根据 FileNameAntistopList 链表内关键词信息更新对应关键词数据库内信息。如果不包含标签信息，同样创建 FileNameParticiple 对象提取文件名内关键词加入 FileNameAntistopList 链表内，检查 FileNameAntistopList 链表，若为空则将文件扩展名作为文件标签，否则将 FileNameAntistopList 链表内关键词按标签分类，分别通过调用 StartFileNameAnlyze 函数计算文件标签概率，选择其中概率最大的标签作为文件标签，然后通过 MySQLObject 更新关键词数据库内对应关键词信息。

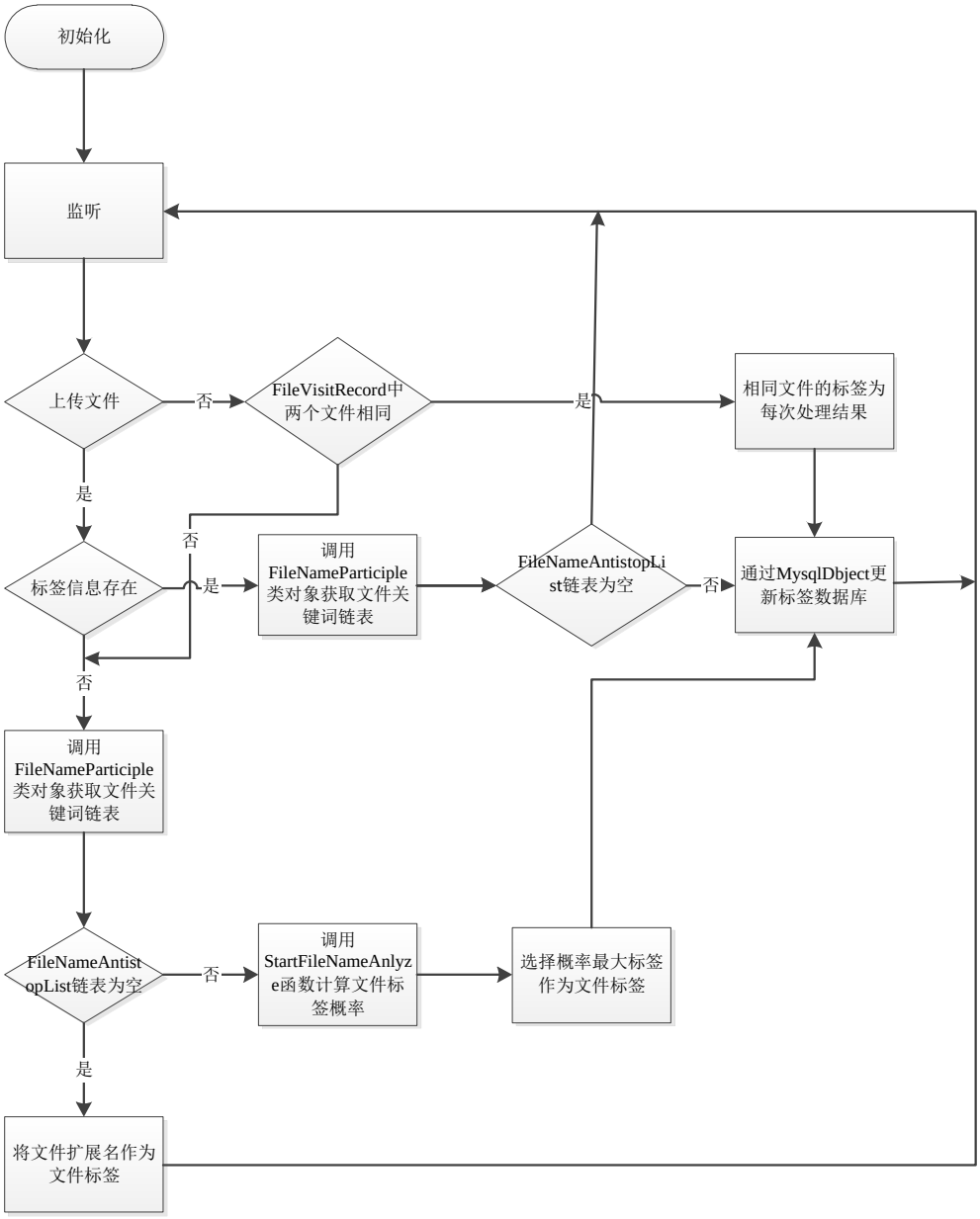


图 4.3 标签分类模块流程图

4.3 虚拟存储模块的实现

本节通过 VirtualStorage 类实现虚拟存储模块功能，类中 FileAdd 函数接收磁盘缓存系统内其他模块发送的文件信息。FileAdd 函数根据文件大小，将大文件通过 HDFS 内 VirtualStorageCommunication 类对象将其写入 HDFS 光盘库，对于小文件则交由小文件合并类 SmallFileMerge 进行处理。

4.3.1 小文件合并模块的实现

小文件合并模块是虚拟存储模块的核心内容，主要通过如图 4.4 所示的小文件合并类 SmallFileMerge 完成， IndexFileInfo 类存储合并大文件的索引文件信息，索引文件内保存组成合并大文件的所有小文件信息，包括小文件 MD5 值、小文件的偏移量 Offset、小文件大小 FileSize。IndexFileInfo 类中 MD5 表示文件在系统内唯一标识，MergeFileMD5 值表示合并大文件的 MD5 值，FileName 表示索引文件名称，FilePath 表示索引文件在磁盘缓存系统内文件存储路径，FirstLabel 表示文件的一级标签信息，SecondLabel 表示文件的二级标签信息，TimeStamp 记录文件时间戳，FirstLabel、SecondLabel 和 TimeStamp 的值与合并大文件时间戳相同。VirtualStorageCommunication 类负责建立虚拟存储模块和 NameNode 节点之间的通信，其内部封装了 HDFS 提供的 DistributedFileSystem 接口。SmallFileMerge 类是小文件合并模块的具体实现类，其负责小文件合并的过程并建立对应的索引文件。其中包含 IndexFileInfo 类型的属性变量 CurSmallFileIndexInfo，保存合并大文件的索引文件信息，包含 FileInfo 类型的属性变量 CurMergeFileInfo，记录当前合并大文件的信息，包含 VirtualStorageCommunication 类型的属性变量，负责和 NameNode 节点之间的通信。

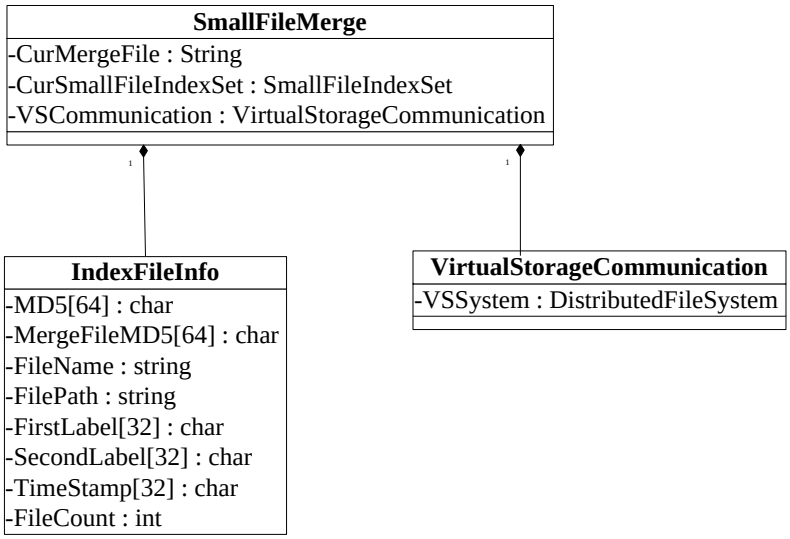


图 4.4 小文件合并类图

磁盘缓存系统通过 VirtualStorage 对象实现虚拟存储模块的功能，其内部主要流程如图 4.5 所示。VirtualStorage 对象在接收缓存系统内其他模块的文件信息的时候，根据文件信息获取文件类型。若是大文件，则通过 VirtualStorageCommunication 类对象将大文件写入 HDFS 光盘库，若是小文件，则将文件信息加入 VirtualStorageFileDirectory 目录等待刻录。VirtualStorage 对象在接收磁盘缓存内其他模块发送的小文件信息的时候，根据小文件信息查询 VirtualStorageFileDirectory 目录。若找到对应的二级标签目录，则将小文件信息插入

SecondLabelInfo 对象的 FileInfoList 中，若未找到，则通过 FirstLabelInfo 类和 SecondLabelInfo 类在 VirtualStorageFileDirectory 目录内创建 FirstLabelInfo 类和 SecondLabelInfo 类对象，然后将小文件信息插入二级标签目录下，并更新二级标签 SecondLabelInfo 对象内 Size 值。若 Size 大于 128 MB 则创建 SmallFileMerge 类创建对象，将 SecondLabelInfo 对象下除新插入小文件合并为大文件，将合并大文件信息加入 CurMergeFileInfo 变量中，将索引文件信息加入 CurSmallFileIndexInfo 变量中，通过 VirtualStorageCommunication 类创建和 HDFS 光盘库的通信。通过 VSCommunication 与 HDFS 光盘库之间通信，将大文件索引文件与大文件数据上传至 HDFS 光盘库进行刻录。

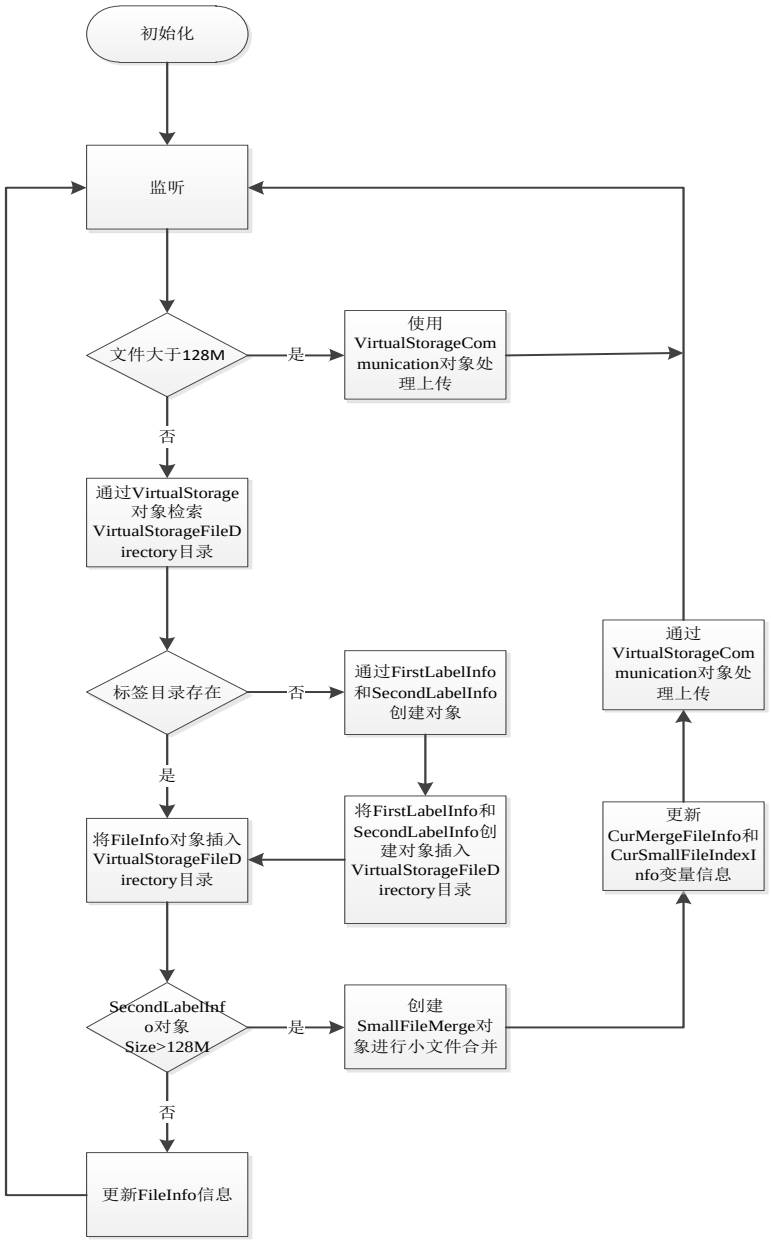


图 4.5 小文件合并流程图

4.3.2 小文件映射模块的实现

小文件映射模块的主要功能是维护 NameNode 内由图 4.6 所示的 IndexLabelDirectory 类创建的索引文件目录 IndexFileDirectory，FirstLabelInfoList 记录标签文件目录内所有一级标签目录信息的双向链表，FileAdd 和 FileDelete 函数实现索引文件的插入和删除。一级标签信息类 FirstLabelInfo 中 SecondLabelInfoList 记录一级标签目录内所有二级标签目录信息的双向链表。二级标签信息类 SecondLabelInfo 中 FileInfoList 记录二级标签目录内所有文件信息的双向链表，FileInfoList 中文件信息按时间戳大小进行排序。IndexLabelDirectory 类、FirstLabelInfo 类和 SecondLabelInfo 类中 LabelName 表示标签名称，FileAdd 和 FileDelete 函数实现文件信息的插入和删除。IndexFileInfo 类记录索引文件信息，MD5 表示索引文件的唯一标识， MergeFileMD5 表示合并大文件的 MD5 值，FileName 表示索引文名称，FilePath 表示索引文件在 NameNode 节点内的存储路径，FirstLabel 表示一级标签信息，SecondLabel 表示二级标签信息，TimeStamp 表示时间戳信息，FileCount 记录索引文件内小文件数目。其中 FirstLabel、SecondLabel、TimeStamp 的值和合并大文件内对应的值相同。

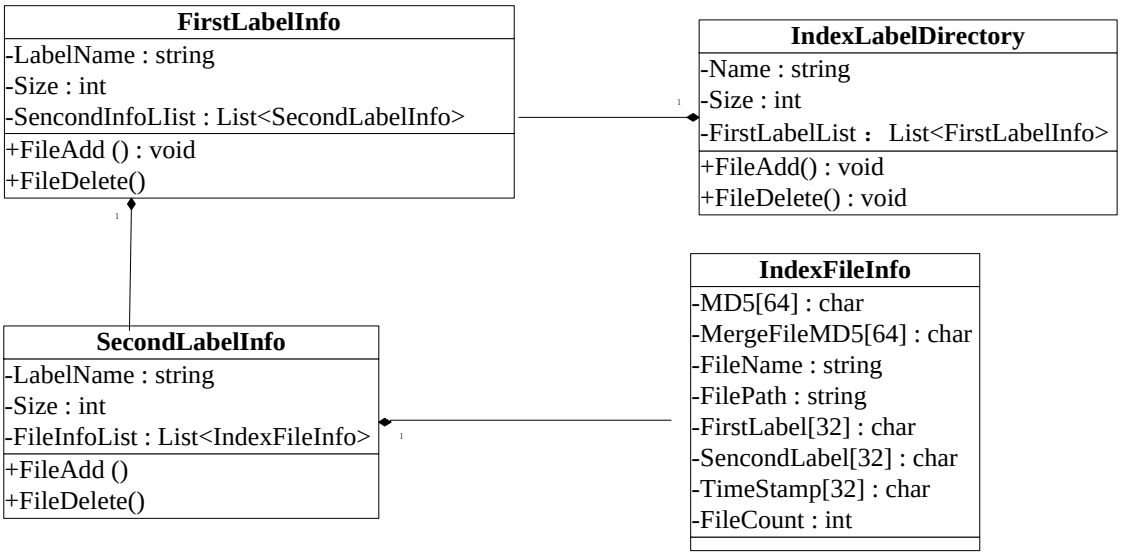


图 4.6 索引文件目录类图

在小文件合并模块创建合并大文件的时候，根据小文件信息创建对应的索引文件，虚拟存储模块通过 VirtualStorageCommunication 类对象将索引文件上传到 NameNode 节点。小文件提取模块根据索引文件信息创建 IndexFileInfo 类对象，根据 IndexFileInfo 类对象中标签信息在 IndexFileDirectory 找到对应的 SecondlabelInfo 类对象，根据 IndexFileInfo 类对象中 TimeStamp 信息，通过二分查找方式找到索引文件信息在 FileInfoList 中位置，并将 IndexFileInfo 类对象插入 FileInfoList 中。当小文件访问发生磁盘中断的时候，虚拟存储模块将文件访问请求转发至 HDFS 光盘库 NameNode 节点。NameNode 节点首先通过小文件信息的标签信息，查找对应的

的 SecondlabelInfo 类对象，查找 FileInfoList 中与小文件信息的 TimeStamp 信息相同的 IndexFileInfo 类对象，然后根据 IndexFileInfo 类对象中 MergeFileMD5、FirstLabel、SecondLabel 和 TimeStamp 等信息在 NameSpace 查询合并大文件的所有数据块存储位置，并根据 IndexFileInfo 类对象中 FilePath 获取索引文件的存储位置，最后将合并大文件和索引文件恢复至磁盘缓存系统。

4.3.3 小文件提取模块的实现

磁盘缓存系统在恢复小文件的时候，主要通过如图 4.7 所示的小文件提取类 SmallFileExtract 完成，图中列出来类的主要数据结构，CurMergeFileInfo 记录合并大文件信息，CurSmallFileIndexInfo 记录索引文件信息，VSCommunication 用于实现与 NameNode 通信，获取大文件信息及索引文件信息。

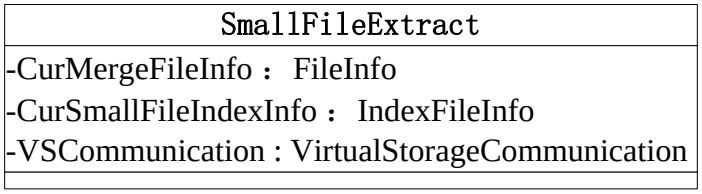


图 4.7 小文件提取类图

当系统内发生访问磁盘中断时，根据文件信息判断文件类型，如果当前处理对象为大文件，则通过虚拟存储模块内 VirtualStorageCommunication 类对象将大文件访问信息上传至 HDFS 光盘库，根据文件预取策略分析预取文件信息，然后将大文件和预取文件恢复至磁盘缓存内。如果是小文件，根据文件预取策略，合并大文件内的小文件具有较强的关联性，因此当小文件访问发生磁盘访问中断时，虚拟存储模块通过 VirtualStorageCommunication 类对象将访问请求转发至 HDFS 光盘库，NameNode 接收磁盘缓存系统的小文件访问请求后，首先查询索引文件目录 IndexFileDirectory 获取索引文件信息放入 CurSmallFileIndexInfo 变量中，然后在 NameSpace 内查找大文件信息放入 CurMergeFileInfo 变量中，通过 VSCommunication 变量将合并大文件和索引文件下载到磁盘缓存系统内，最后根据 SmallFileExtract 对象中 CurSmallFileIndexInfo 变量将小文件恢复至磁盘空间内，对于在磁盘内已经存在的文件，将恢复数据直接删除，对于磁盘中不存在的小文件，将小文件信息插入磁盘管理标签目录中，并更新小文件信息。

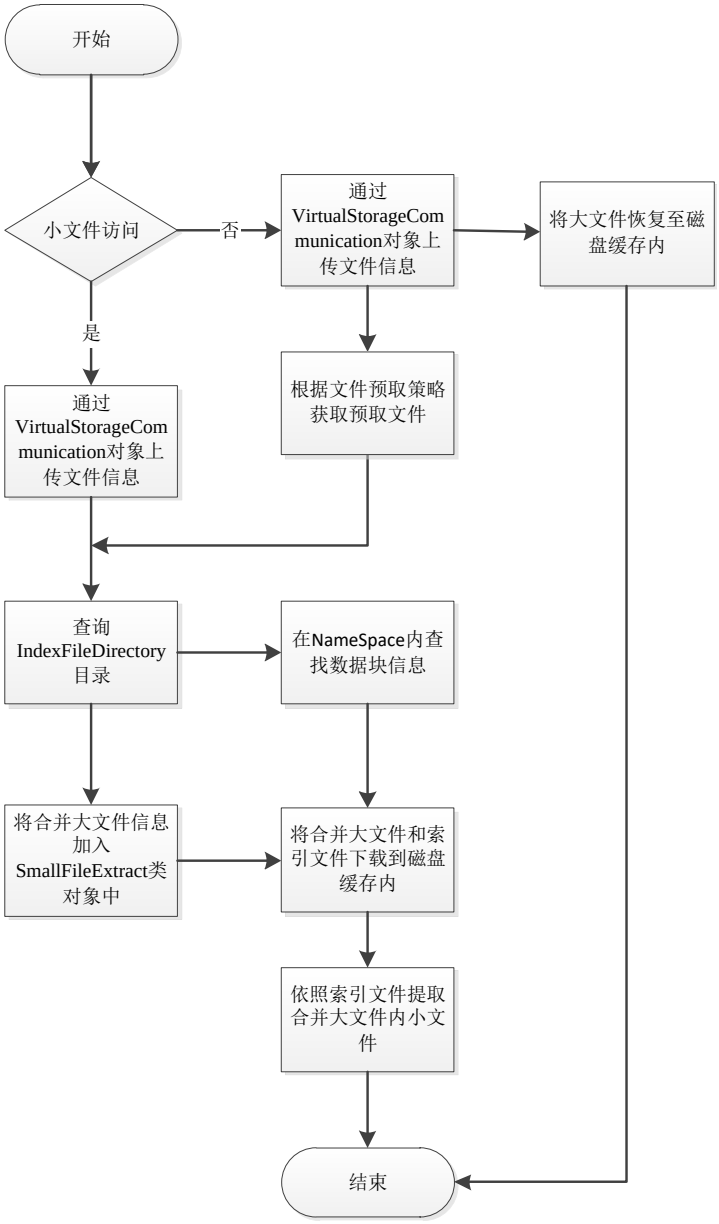


图 4.8 小文件提取流程图

4.4 Cache 模块的实现

4.4.1 文件预取模块的实现

文件预取模块主要对发生磁盘访问中断的访问请求确定其预取内容，文件预取模块其主要功能由如图 4.9 所示的 FilePrefetch 类实现。PrefectchFileList 链表记录经过 FilePrefetch 类获取的预取文件信息，FileVist 函数接收发生磁盘中断的文件信息，FileInfoQuery 函数负责在大文件发生访问中断的时候查询预取文件信息，sigFileInfo 函数负责向虚拟存储模块发送文件信息。当用户访问的文件不在磁盘缓存系统会触发文件预取模块，首先创建 FilePrefetch 类对象，通过

FileVist 函数接收当前发生访问中断的 FileInfo 类对象，并依据文件类型对采用不同的处理方式。若文件类型为小文件，将 FileInfo 类对象加入 PrefectchFileList 链表内，若文件类型为大文件，将 FileInfo 类对象加入 PrefectchFileList 链表内，并调用 FileInfoQuery 函数向虚拟存储模块内发送文件信息查询任务，FileInfoQuery 函数查询索引文件信息目录 IndexFileDirectory 内与获取与当前 FileInfo 类对象时间戳最近且标签相同的索引文件信息，并将索引文件信息插入 PrefectchFileList 链表内，调用 sigFileInfo 函数将 PrefectchFileList 链表内文件信息通过虚拟存储模块转发至 HDFS 光盘库内，HDFS 光盘库根据系统内数据块预取策略将文件快速恢复至磁盘缓存系统。

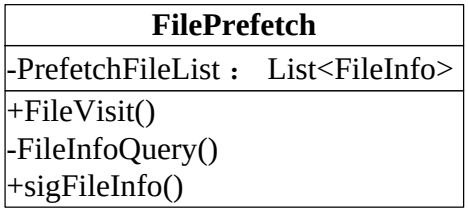


图 4.9 文件预取模块类图

4.4.2 缓存替换算法的实现

Cache 模块主要功能由如图 4.10 所示的 CacheManage 类实现。CacheManage 类内维护一个近期访问的标签信息的的双向链表 CacheFIleList，链表中的每个节点记录访问的标签信息及文件信息，FileType 表示当前节点内文件类型，Firstlabel 记录节点的一级标签信息，SecondLabel 记录节点的二级标签信息，FileInfoList 用于记录该节点所有文件信息，Timestamp 表示文件的时间戳信息。CacheManage 类中 CacheFileAdd 函数负责接收其他模块发送的文件信息，CacheFileAdd 函数通过调用类中 UpdateCacheList 对 CacheFileList 进行更新，将淘汰的文件信息通过 SigFileInfo 函数发送至虚拟存储模块。

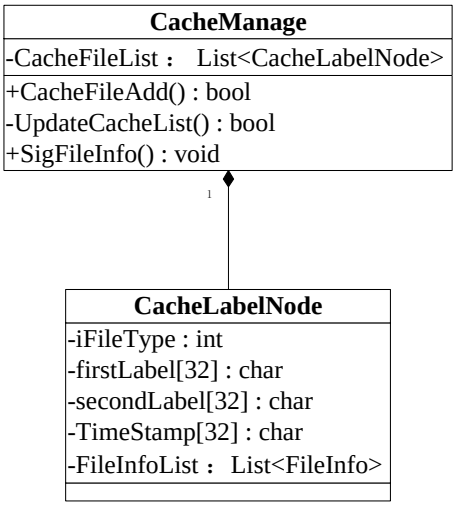


图 4.10 缓存替换算法管理类图

根据本文设计的预取策略，磁盘缓存系统和 HDFS 光盘库之间的调度单位为大文件，因此 Cache 模块主要管理对象也是大文件，根据 LB-LRU 缓存替换算法对 Cache 模块内文件进行动态管理，LB-LRU 缓存算法流程图如图 4.11 所示。当系统访问某一文件的时候，若文件不在磁盘则发生磁盘访问中断，根据 FilePrefetch 类对象确定预取文件信息，通过 VirtualStorage 类对象将访问文件和预取文件恢复至磁盘缓存系统内，并将访问文件和预取文件信息发送至 CacheManage 类对象；若在磁盘缓存系统内查询到访问文件，将文件发送给用户响应用户需求并将文件信息发送至 CacheManage 类对象。CacheManage 类对象接收其他模块内 FileInfo 类对象的时候，根据 FileInfo 对象内二级标签信息及时间戳信息检索 CacheFileList 链表。若在 CacheFileList 链表找到对应节点信息，检索节点内 FileInfoList 链表内是否包含 FileInfo 对象信息，若不存在则将 FileInfo 对象插入 FileInfoList 链表，然后将该节点移动至 CacheFileList 链表顶端。若在 CacheFileList 链表无法找到对应节点信息，则创建新的节点，将 FileInfo 对象的一级标签、二级标签、时间戳信息和文件类型信息放入节点 CacheLabelNode 类对象的 Firstlabel、SecondLabel、TimeStamp 和 FileType，检索磁盘管理模块内 DiskFileDirectory 目录，将与 FileInfo 对象具有相同时间戳的文件信息插入节点内 FileInfoList 链表内，然后将新建节点插入 CacheFileList 链表顶端。当系统内发生磁盘访问中断，需要将 HDFS 光盘库内文件恢复至磁盘缓存内而磁盘缓存空间不足时，根据 LB-LRU 缓存替换算法思想将 CacheFileList 链表尾部节点文件淘汰，当文件从 Cache 模块淘汰的时候，将文件信息中 iAlter 等于 1 的文件发送至虚拟存储模块等待刻录，将文件信息中 iAlter 等于 0 且 iposition 等于 2 的文件直接从磁盘缓存内部删除，无需进行再次刻录，将文件信息中 iAlter 等于 0 且 iposition 等于 0 的文件交由虚拟存储模块等待刻录。将二级标签信息为文件扩展名的文件交由标签分类模块再次进行文件分类。

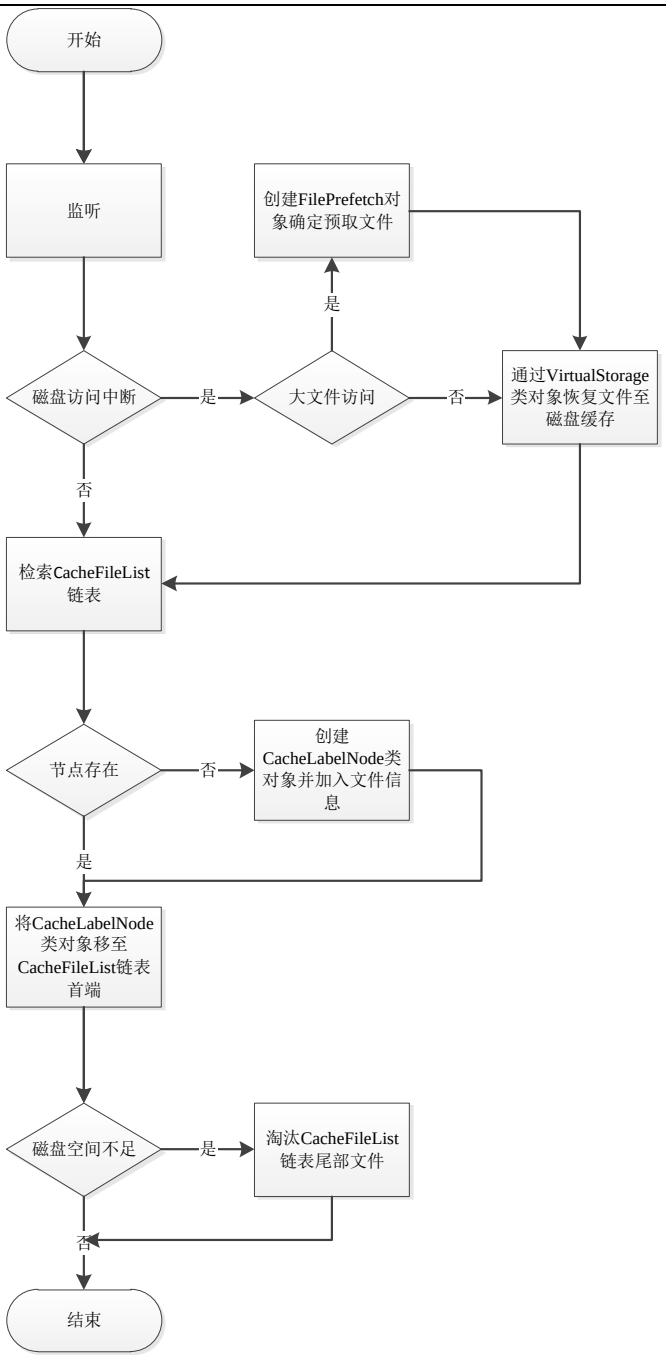


图 4.11 LB-LRU 缓存替换算法流程图

4.5 本章小结

本章首先对标签文件目录和标签分类模块的实现进行介绍，然后讨论虚拟存储模块中小文件合并模块、小文件映射模块和小文件提取模块的实现，最后描述了磁盘缓存系统内文件预取和缓存替换算法的实现过程。

第五章 系统测试与分析

5.1 测试环境

在前面两个章节中，阐述磁盘缓存系统中主要的模块的详细设计和实现，本章节对磁盘缓存系统各个模块设计具体的测试方案，通过测试证明磁盘缓存系统提升 HDFS 光盘库存储能力的可行性。本章主要对加入磁盘缓存系统的 HFFS 光盘库进行性能测试，以下将加入磁盘缓存系统的 HFFS 光盘库简称本文方案。在实际测试过程中，由于设备数量有限，我们搭建了一个简单的 HDFS 光盘库和磁盘缓存系统。测试过程中共使用 4 个服务器和 2 个光盘库，服务器采用相同的类型的电脑，配置均为 Inter Core i3-3220 CPU，3.30 GHz，操作系统为 Red Hat Enterprise Linux Server 2.6，内存 4 GB，硬盘为 SATA 500 GB， 2 个光盘库均使用 NETZON 出产的 HMS 2105 型号光盘库，该光盘库可容纳 105 张 100 GB 光盘，存储容量达 10.5 TB，同时光盘库配置三个光驱，可同时进行数据的读取和刻录，光驱的最低配置为 2 倍速，最大可调至 8 倍速。光盘库在使用过程中，为了保证数据刻录的安全性与准确性，在刻录过程中光驱使用 4 倍速，在读取数据的时候在保证数据安全的情况下采用最大倍速。磁盘缓存使用 1 台服务器对系统内文件进行缓存，HDFS 光盘库共使用 3 台服务器 2 台 HMS2015 光盘库构成，HDFS 光盘库中将 1 台服务器构建为 NameNode 节点，将 2 台服务器和光盘库构建为 2 个 DataNode 节点。市场上光盘库使用较多的是蓝光光盘，蓝光光盘有 25 GB、50 GB 和 100 GB 三种规格，在实际测试过程中为了更加接近现实，本文采用的是 25 GB 蓝光光盘，在实际测试过程中会有换盘的过程。本文研究课题来源于项目“海量数据存储系统的研究”中磁盘缓存缓存系统的研究，项目的历史版本系统主要应用于军队科研单位、医院、政府、法院、大学、银行、图书馆等，对冷数据和重要文件进行备份。本章测试所使用的数据为项目存储的历史数据，根据项目评测标准对本文设计的磁盘缓存系统的优化效果进行分析。

5.2 标签分类性能测试

5.2.1 测试方案

为了对本文设计的标签分类模块的性能进行测试，将对磁盘缓存系统内新上传的文件通过标签分类模块为文件贴上标签，并对文件标签的测试结果进行分析和评价。本节测试中所使用的数据集来源于项目存储的某单位历史数据。该数据集截取了 2017-2018 年内某单位的部分备份文件。数据集内文件名表述规范适合做标签分类测试，另外该数据集来自项目历史系统版本，文件标签均已由人工创建，有利于对标签分类结果进行分析。整个数据集共包含 8919 个文件，文件内容涵盖历史、经济、教育、体育、法律、军事、新闻、文学、工科共八类文件类型，将

文件类型在磁盘缓存系统创建初期初始化为一级标签，同时在系统创建初期根据文件类型初始化部分二级标签信息，例如一级历史标签内包含辛亥革命、解放战争、抗日战争、唐朝、宋朝等二级标签，并根据图书馆往年备份文件的文件名构建关键词数据库，如关键词邓小平属于一级标签历史的二级标签抗日战争和解放战争，同时也属于一级标签经济的二级标签改革开放。使用包含文件标签和关键词信息的训练集文件对关键词数据库中关键词的标签 Count 值进行更新，本节在关键词数据库更新阶段所使用的训练集为 2016-2017 年相同类型文件集合。为了合理的评价标签分类模块对不同类型文件的标签分类效果，本文设计一级标签准确率和二级标签正确率对文件分类效果进行评价，两种准确率计算公式如下。

$$Q_1 = \frac{N_F}{N} \times 100\% \quad (5\text{-}1)$$

$$Q_2 = \frac{N_S}{N} \times 100\% \quad (5\text{-}2)$$

公式 5-1 Q_1 表示文件分类结果的一级标签准确率，N_F 表示文件一级标签正确的数目，N 表示实际参与分类的文件数目。公式 5-2 Q_2 表示文件分类结果的二级标签准确率， N_S 表示文件二级标签正确的数目，N 表示实际参与分类的文件数目。Q_1 表示标签分类过程中一级标签分类正确的文件占测试文件的比例，Q_2 表示二级标签分类正确的文件占测试文件的比例，根据 Q_1 和 Q_2 评测标签分类模块的性能，测试采用标签分类模块代替人工分类的可能性。

5.2.2 测试结果及分析

本次测试中使用不同数量训练集更新的关键词数据库对 8919 个测试文件进行测试，并根据 Q_1 和 Q_2 评价标签分类模块的性能结果，不同数量训练集的测试结果如表 4.1 所示，其中 N_T 表示训练集中文件数目。

表 4.1 标签分类测试结果

N_T	N_F	N_S	N	Q_1	Q_2
0	6917	6682	8919	77.55%	74.91%
1000	7984	7574	8919	89.52%	84.92%
2000	8356	7916	8919	93.69%	88.75%
3000	8514	8312	8919	95.46%	93.19%
4000	8625	8547	8919	96.70%	95.82%
5000	8656	8586	8919	97.01%	96.26%

由表 4.1 可以看出，由于未使用训练集的关键词数据库内关键词的各个标签概率相同，因此对于只包含一个关键词的文件，在标签分类过程中会导致标签分类结果错误，测试结果表明二级标签的准确率仅达到 75%左右。在首次加入 1000 个训练集文件对关键词数据库更新后，文件分类的准确率明显提高，二级标签准确率达到 85%左右。随着训练集的数量增加，文件标签的准确率也不断提升，当训练集数量为 5000 个的时候，文件二级标签的准确率超过 96%，在不同数量训练集中文件的一级标签的准确率始终大于二级标签的准确率。本次测试仅对新上

传的文件进行分类处理，在实际使用过程，关键词数据库会随着系统的运行不断更新，关键词的标签准确率也会越来越高。另外对于标签为扩展名的文件，当其再次淘汰的时候，会再次通过标签分类模块进行分类，因此标签分类模块的准确率会比本次测试的结果更高。

5.3 小文件处理性能测试

5.3.1 测试方案

本节将主要测试经磁盘缓存系统处理后的文件读写能力。磁盘缓存系统根据标签之间的关联性设计小文件合并模块，对传统 HDFS 光盘库 NameNode 节点中小文件索引文件构建方式进行改进，并保存传统 HDFS 光盘库的块文件管理模块和 I/O 调度策略。在测试过程中，磁盘缓存系统缓存容量设置为 5 GB，HDFS 光盘库内部使用 Hadoop2.4.0 版本，其默认的数据块大小为 128 MB，设置光盘库的刻录阀值为 2 GB。为了对本文设计的虚拟存储模块性能进行测试，主要对本文方案和传统 HDFS 光盘库对相同的文件读写过程进行性能测试，首先测试小文件在 NameNode 节点进程的内存消耗，然后分别测试本文方案和传统 HDFS 光盘库中小文件和大文件的读写性能。

NameNode 节点进程的内存消耗测试中共使用 10000 个小文件，将其分成十组，每个小文件大小为 1 MB，先后分成 10 次上传，当完成一组文件上传后，随机选择其中 100 个小文件进行读取测试，测试文件在 NameNode 节点中所消耗的内存空间，用以评价磁盘缓存系统对小文件存储问题的改善效果。在小文件读写性能的测试中，采用将 40000 个文件分为 8 组以递增的方式逐渐增加，每一组的文件数目为 5000 个，主要包括小文件的上传响应时间和文件下载时间测试。在大文件的读写性能测试中，使用 500 个 200 MB 的大文件分成 10 组上传，测试大文件在本文方案和传统 HDFS 光盘库的大文件读写性能，主要包括大文件上传响应时间和大文件下载时间测试。上述三个测试中，均将多次测试结果的平均值作为最终测试结果。

5.3.2 测试结果及分析

在 NameNode 节点进程的内存消耗测试中，主要对本文设计的索引文件的改进方案进行测试，NameNode 节点的内存使用量的衡量工具采用了 JDK 提供的 JConsole 工具。分别对未采用小文件存储策略的 HDFS 光盘库、传统 HDFS 光盘和本文方案进行性能测试，测试结果如图 5.1 所示。图中的横轴记录测试中所使用的小文件数目，纵轴记录当前 NameNode 节点进程的内存消耗（MB）。由图 5.1 可以看出，在文件数目不超过 1000 的时候，NameNode 在三种存储方案中内存消耗差距较小，在加入多组测试数据之后，未采用小文件合并的原始 HDFS 所消耗的内存明显高于其他两种方案。对于加入小文件合并的传统 HDFS 光盘库而言在 NameNode 节点的内存消耗上相比原始 HDFS 减少了很多，但是索引信息的检索速度较慢，无形中增加了 NameNode 节点的内存消耗。本文根据标签信息建立索引文件目录，目录内部索引文件信息根

据时间戳进行排序，当检索索引文件的时候，可根据小文件标签信息快速定位二级标签文件目录，并根据时间戳信息按二分查找的方式快速查询到索引文件，达到减少索引文件检索算法复杂度的目的，从而一定程度上缓解 NameNode 节点内存压力。另外磁盘缓存系统内部保存近期访问的热数据，在访问小文件的时候，可减少访问 HDFS 光盘库的次数。因此本文方案的内存消耗相比传统 HDFS 光盘库更少。

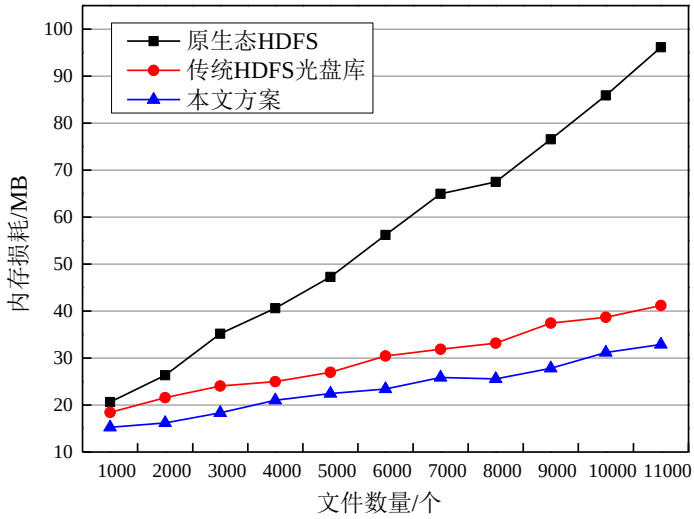


图 5.1 NameNode 内存消耗图

在小文件和大文件读写性能测试中对 1 MB 的小文件和 200 MB 的大文件在本文方案和 HDFS 光盘库内分别进行测试。WriteTime1 表示文件在本文方案中用户上传文件的响应时间，WriteTime2 表示文件在传统 HDFS 光盘库中用户上传文件的响应时间，ReadTime1 表示文件在本文方案的文件读时间，ReadTime2 表示文件在传统 HDFS 光盘库的文件读时间，测试结果如图 5.2 和 5.3 所示，纵轴记录文件读写时间，横轴记录文件个数。

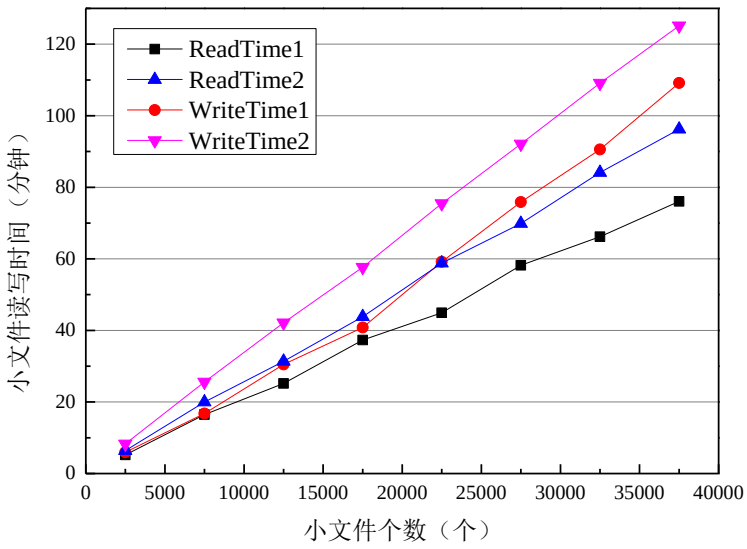


图 5.2 小文件存储性能测试图

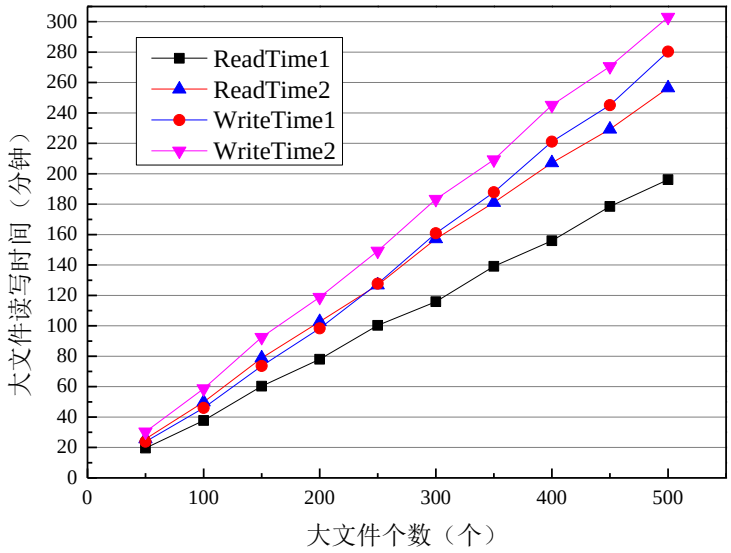


图 5.3 大文件存储性能测试图

由图 5.2 和图 5.3 分析可得，本文方案与传统 HDFS 光盘库相比，本文方案中磁盘缓存系统在响应用户访问请求的同时，可并发的在磁盘缓存系统和 HDFS 光盘库之间进行文件传输，很大程度上减少文件读写的等待时间，且磁盘与磁盘之间数据传输速度明显高于磁盘缓存系统与 HDFS 光盘库之间数据传输速度，因此在小文件和大文件读写方面优化的效果较为明显。在图 5.2 和图 5.3 中，本文方案的文件上传速度明显存在拐点，这是由于磁盘缓存系统的写入速度高于 HDFS 光盘库的刻录速度，当磁盘缓存系统容量不足时，整个系统的文件写入速度受 HDFS 光盘库写入速度限制。本文在磁盘缓存系统内对传统 HDFS 光盘库放入小文件存储方案进行优化，在磁盘缓存系统内进行小文件合并的时候，仍可持续接收其他小文件上传任务，且在 NameNode 内根据时间戳可快速查询索引文件信息，提高索引文件的查询效率，所以对小文件读写性能优化效果较为明显。

5.4 缓存替换算法性能测试

5.4.1 测试方案

命中率是评价缓存替换算法的重要参数，本文设计的 LB-LRU 算法是根据磁盘缓存系统内文件类型对 LRU 算法的改进方案。LRU 算法具有较高命中率、易实现、系统开销低等优点，在具有较多热数据访问的一级缓存中，LRU 能保持较高的命中率。二级缓存和一级缓存相比，在缓存容量和缓存数据访问间隔更高，二级缓存中突发性访问所占的比例也高于一级缓存。LRU 算法应用在二级缓存会在缓存内产生缓存污染，导致算法的性能严重下降。本文设计的 LB-LRU 算法在考虑磁盘缓存系统突发性访问的特点，结合 LRU 算法的优点和文件预取技术，根据标签之间的关联性对预取文件进行缓存，提高磁盘缓存系统命中率，减少访问 HDFS 光盘库的次数。

本节在磁盘缓存系统内使用 FIFO 算法、LRU 算法和 LB-LRU 算法对其内部文件进行管理，根据测试效果评价本文设计的 LB-LRU 算法。访问模型和缓存容量不同将导致算法的性能发生改变，因此本节测试三种算法在不同访问模型下的命中率。

5.4.2 测试结果及分析

首先测试 80/20 的访问模式。根据 80/20 原则，用户在访问文件的时候，其中 80%的访问集中在 HDFS 光盘库内 20%的文件上，因此采用缓存替换策略将系统内热数据存储在磁盘缓存系统中，可很大程度减少系统内发生磁盘中断的次数。为了测试本文设计的 LB-LRU 算法性能，本节根据 80/20 原则设计一种访问模型，模型内 80%的文件访问请求随机的分布在 HDFS 光盘库存储的 20%的数据上，测试采用 FIFO 算法、LRU 算法和 LB-LRU 算法的命中率。测试结果如图 5.4 所示，纵轴表示算法的缓存命中率，横轴表示磁盘缓存系统设置的缓存容量。

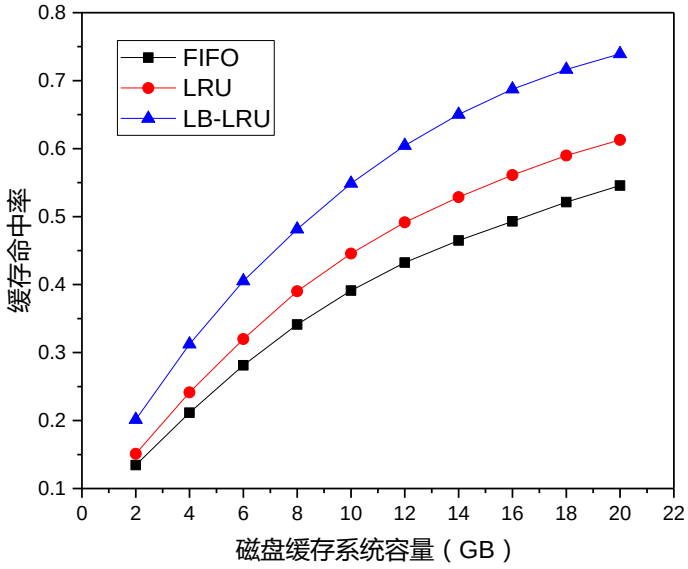


图 5.4 80/20 访问模式测试结果

由图 5.4 的测试结果可以看出，当磁盘缓存系统容量较小的时候，LB-LRU 算法和另外两种算法的性能相差不大。随着磁盘缓存系统的容量不断增加，三种算法的性能也不断提高，且 LB-LRU 算法在 80/20 访问模式中的测试效果最佳。因为 LB-LRU 算法的链表中包含合并大文件节点，节点内部小文件之间具有较强的关联性，可提高磁盘缓存系统的命中率。因此可以看出 LB-LRU 算法相对于 LRU 算法和 FIFO 算法在磁盘缓存系统中取得效果最好。

由图 5.4 的测试结果可以看出，LRU 算法在 80/20 访问模型中保持较高的命中率，但是二级缓存中存在较多的突发性性访问，因此本节增加突发性访问模型的测试。图 5.5 是在 80/20 访问模式基础上加入 10%的突发性文件访问的三种算法命中率测试结果，即增加 10%的文件随机访问作为突发性文件访问，加入突发性文件访问后，其访问模型不符合 80/20 原则。由图 5.5 测试结果可以看出，在加入突发性访问以后，LRU 算法的性能相对在 80/20 访问模式中降低了

很多。由于有大量的随机的文件访问的时候，根据 LRU 算法淘汰规则，可能会将部分热数据替换出磁盘缓存系统，因此才会导致 LRU 算法的缓存命中率降低。LB-LRU 算法中发生访问中断的时候，将预取的合并大文件加入缓存链表内，可减少突发性文件访问的中断次数，虽然冷数据访问会将部分文件替换出内存，但是合并大文件内包含小文件较多，每个小文件访问都会提升合并大文件的优先级，包含热数据的合并大文件也不容易替换出缓存，LB-LRU 算法可将突发性访问文件快速替换出内存，减少其带来的影响。

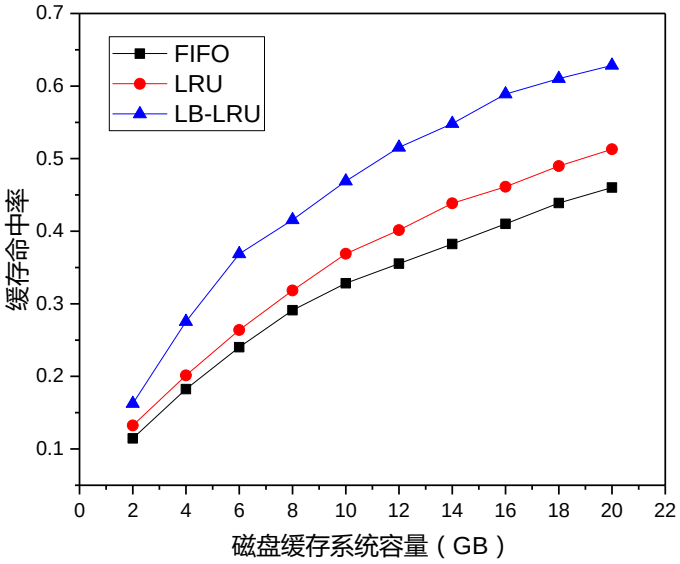


图 5.5 10%突发性访问模型测试结果

二级缓存相对一级缓存的突发性访问比例更高，因此进一步提高突发性访问比例测试三种算法的缓存命中率，在 80/20 访问模式基础上加入 20%的突发性文件访问进行测试，由图 5.6 的测试结果可以看出，FIFO 算法和 LRU 算法在磁盘缓存系统的表现性能较差，LB-LRU 算法虽然在加入突发性访问后性能有所下降，但是仍然保持较高的缓存命中率。

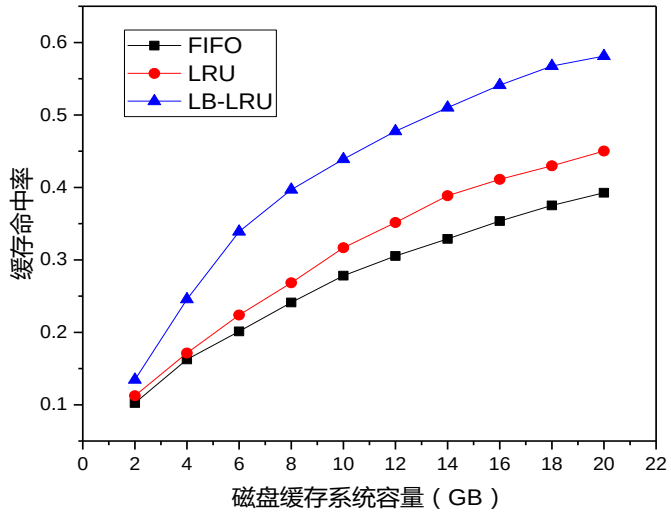


图 5.6 20%突发性访问模型测试结果

5.5 本章小结

本章首先介绍测试中使用设备和数据来源，然后对本文设计的标签分类模块、虚拟存储模块和 Cache 模块设计具体的测试方案。由测试结果可以看出，磁盘缓存系统中标签分类模块的分类结果具有较高的准确率，虚拟存储模块有效地提高文件的读写能力并降低 NameNode 节点进程的内存消耗，Cache 模块通过 LB-LRU 算法和预取策略有效地提高磁盘缓存的命中率，进一步提高文件的读写速度。

第六章 总结与展望

6.1 总结

随着大数据时代的带来，全球每天产生的数据量高达 PB 级，数据总量规模也越来越大。但是这些海量数据中，仅有少量的热数据被经常访问，另外的冷数据基本上很少被访问。如果将热数据和冷数据均采用磁盘阵列存储，将会造成存储成本的极大提高。对于冷数据存储，目前比较流行的存储系统是 HDFS 光盘库，但是 HDFS 光盘库相对于磁盘阵列之间的差距仍然较大，因此本文在 HDFS 光盘库的基础上设计一种磁盘缓存系统，期望缩小与磁盘阵列的差距。对于磁盘缓存系统的设计与实现，主要研究内容如下：

1. 分析了缓存系统的现有的集中缓存替换算法和预取策略，并对分布式存储系统处理小文件存储的策略进行了详细的分析和总结。

2. 介绍 HDFS 运用在光盘库的具体实现过程，分析 HDFS 光盘库在实际存储存在的问题。

3. 根据 HDFS 光盘库系统存在问题设计标签分类模块建立文件之前的关联性。标签分类模块内主要通过文件名和关键词数据库匹配结果，获取文件名内关键词信息，然后根据标签分类算法获取文件的标签信息。

4. 对传统 HDFS 光盘库的小文件存储方案进行优化。虚拟存储模块根据文件标签信息对磁盘缓存系统内小文件进行合并，并优化了小文件映射模块，提高索引信息的查询速度。

5. 本文根据磁盘缓存系统存储文件类型，提出一种基于文件标签的 LB-LRU 算法对系统内文件进行管理，并设计一种文件预取策略进一步提高磁盘缓存系统的缓存命中率。

6. 对所设计的磁盘缓存系统进行了编码实现，搭建简单的磁盘缓存系统和 HDFS 光盘库，并将磁盘缓存系统加入 HDFS 光盘库前端，为磁盘缓存系统内各个模块设计具体的测试方案，对加入磁盘缓存系统的 HDFS 光盘库和传统 HDFS 光盘库进行测试比较，并对测试结果进行总结和分析。

6.2 进一步研究工作

为了进一步践行研究工作，以下几个内容将是研究达成实践的关键部分:

1. 在文件标签分类过程中，依据关键词数据库中关键词信息获取文件标签，由于关键词数量有限，会出现文件名无法提取关键词情况，创建内容更加丰富的分词词典，通过自然语言处理技术对文件名进行分词的文件标签分类结果更加准确。

2. 本文文件预取对象是经过小文件合并处理的大文件，而大文件的历史访问记录未做处理，可通过分析大文件历史访问记录挖掘文件之间的关联性，保证预取的准确性，并根据大文件之

间的关联性，在系统空闲的时候预取近期访问的热数据。

3. 蓝光盘只能刻录一次，若文件发生修改，只能采用重新刻录并覆盖原文件信息，即原文件数据存储空间将无法使用，导致系统内存储容量越来越少，因此对系统内对系统内无法使用空间较多的光盘进行替换，是下一步研究工作。

4. 本文将索引文件保存在 NameNode 节点内，当索引文件的数目较多的时候，可能导致 NameNode 节点的内存不足，因此进一步研究索引文件的存储方案，将索引文件存储在 DataNode 节点内，将索引文件的索引保存在 NameNode 节点内，可有效的提高 HDFS 光盘库系统容量。

参考文献

[1]. Driscoll A O , Sleator R D . Synthetic DNA: The next generation of big data storage[J]. Bioengineered, 2013, 4(4):123-125.

[2]. 王峰,闫汇,刘圆.冷数据存储研究[J].电信技术,2017(06):22-24.

[3]. Springer U S . magnetic tape storage[J]. Computer Science & Communications Dictionary, 2000, 2(22):613 - 614.

[4]. Mcleod R R , Daiber A J , Mcdonald M E , et al. Microholographic multilayer optical disk data storage[J]. Applied Optics, 2005, 44(16):3197-207.

[5]. Ohta T, Nishiuchi K, Narumi K, et al. Overview and the Future of Phase-Change Optical Disk Technology[J]. Japanese Journal of Applied Physics, 1999, 39(39):770-774.

[6]. Ragunathan T , Battula S K , Gopisetty R , et al. Novel Read Algorithms for Improving the Performance of Big Data Storage Systems[J]. Procedia Computer Science, 2015, 50:264-269.

[7]. Ciritoglu H E , Saber T , Buda T S , et al. [IEEE 2018 IEEE International Congress on Big Data (BigData Congress) - San Francisco, CA, USA (2018.7.2-2018.7.7)] 2018 IEEE International Congress on Big Data (BigData Congress) - Towards a Better Replica Management for Hadoop Distributed File System[J]. 2018:104-111.

[8]. Kamath A, Jaiswal A, Dive K. From Idea to Reality: Google File System[J]. International Journal of Computer Applications, 2014, 103(9):8-10.

[9]. Nightingale E B , Chen P M , Flinn J . Speculative execution in a distributed file system[J]. ACM Transactions on Computer Systems, 2006, 24(4):361-392.

[10]. 魏南琛. 面向光盘库的 HDFS 文件系统应用研究与实现[D]. 武汉：华中科技大学, 2014.

[11]. Zhang W, Lu G, He H, et al. Exploring large-scale small file storage for search engines[J]. Journal of Supercomputing, 2016, 72(8):2911-2923.

[12]. 曹强,严文瑞,姚杰,谢长生.一种超大容量自动光盘库的设计与实现[J].红外与激光工程,2016,45(09):35-42.

[13]. Hui-zong LI,Xue-gang HU,Yao-jin LIN,Wei HE,Jian-han PAN.A social tag clustering method based on common co-occurrence group similarity[J].Frontiers of Information Technology & Electronic Engineering,2016,17(02):122-134.

[14]. Wang T , Yao S , Xu Z , et al. A small file merging and prefetching strategy based on access task in cloud storage[J]. Geomatics & Information Science of Wuhan University, 2013,

38(12):1504-1512.

[15]. Balamash A, Krunz M. An overview of web caching replacement algorithms[J]. Communications Surveys & Tutorials IEEE, 2004, 6(2):44-56.

[16]. Lu D X, Niu Y X, Zou L Q. Analysis of Virtual Storage Technology and its Application in the Library[J]. Advanced Materials Research, 2013, 756-759:1289-1294.

[17]. SMITH, A. J. CPU cache prefetching : Timing evaluation of hardware implementations[J]. IEEE Transactions on Computers, 2002, 47(5):509-526.

[18]. Zhuo L, Wang B, Yu W. HALO: a fast and durable disk write cache using phase change memory[J]. Cluster Computing, 2017(9):1-13.

[19]. Benhamida N, Bouallouche-Medjkoune L, Aïssani D. Simulation evaluation of a relative frequency metric for web cache replacement policies[J]. Evolving Systems, 2017, 9(C):1-10.

[20]. 吕慧, 何炎祥, 黄传河. 随机缓存可靠组播算法[J]. 武汉理工大学学报, 2009(18):24-27.

[21]. VAZQUEZCORTIZO, GARCIA, BLONDIA, et al. FIFO by sets ALOHA (FS-ALOHA): a collision resolution algorithm for the contention channel in wireless ATM systems[J]. Performance Evaluation, 1999, s 36–37(99):401-427.

[22]. Jelenkovi, Predrag R, Radovanovi, et al. Least-recently-used caching with dependent requests[J]. Theoretical Computer Science, 2002, 326(1):293-327

[23]. Selvakumar S, Sahoo S K, Venkatasubramani V. Delay sensitive least frequently used algorithm for replacement in web caches[J]. Computer Communications, 2004, 27(3):322-326.

[24]. Lee D, Choi J, Kim J H, et al. LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies[J]. Computers IEEE Transactions on, 1999, 27(1):134-143.

[25]. O'Neil E J , O'Neil P E , Weikum G . An optimality proof of the LRU-K page replacement algorithm[J]. Journal of the Acm, 1999, 46(1):92-112.

[26]. Megiddo N, Modha D S. ARC: A Self-Tuning, Low Overhead Replacement Cache[C]//FAST. 2003, 3(2003): 115-130.

[27]. Byna S , Chen Y , Sun X H . Taxonomy of Data Prefetching for Multicore Processors[J]. Journal of Computer Science and Technology, 2009, 24(3):405-417.

[28]. Wang X, Kwon T, Choi Y, et al. Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users[J]. IEEE Wireless Communications, 2013, 20(3):72-79.

[29]. Park D J , Kim H J . Prefetch policies for large objects in a Web-enabled GIS application[J]. Data & Knowledge Engineering, 2001, 37(1):65-84.

[30]. C.Xu, T. Ibrahim. A Keyword-Based Semantic Prefetching Approach in Internet News Services. IEEE Transactions on Knowledge and Data Engineering. 2004, 16(5): 601-611

[31]. 许欢庆,王永成,孙强.基于隐马尔可夫模型的 Web 网页预取[J].上海交通大学学报,2003(03):404-407.

[32]. 曹仰杰,石磊,卫琳,古志民.基于剪枝技术的自适应 PPM 预测模型[J].计算机工程与应用,2006(28):141-144+158.

[33]. 班志杰,古志民,金瑜.基于非压缩后缀树的在线 PPM 预测模型[J].计算机工程,2008(10):70-72.

[34]. 金民锁,刘红祥,王佐.基于隐马尔科夫模型的浏览路径预测[J].黑龙江科技学院学报,2005(03):167-170.

[35]. Li M, Varki E, Bhatia S, et al. Ta P: Table-based prefetching for storage caches. Proc. of the 6th USENIX Conference on File and Storage Technologies. 2008(2): 81-96

[36]. 王文建,陶宏才.一种新的基于预测的网页预取模型和缓存算法[J].工业控制计算机,2014,27(08):45-47.

[37]. 徐宝文,张卫丰.数据挖掘技术在 Web 预取中的应用研究[J].计算机学报,2001(04):430-436.

[38]. Zhu H, Chen Y, Sun X H. Timing local streams:improving timeliness in data prefetching[C]// Acm International Conference on Supercomputing. 2010.,16(9):169-178

[39]. Ossa B A D L. Key factors in web latency savings in an experimental prefetching system[J]. Journal of Intelligent Information Systems, 2012, 39(1):187-207.X.Chen, X. Zhang. A popularity-based prediction model for Web prefetching. Computer. 2003, 36(3): 63-70

[40]. Chen X, Zhang X. A popularity-based prediction model for Web prefetching[J]. Computer, 2003, 36(3):63-70.

[41]. Amer A, Long D D E. Aggregating caches: A mechanism for implicit file prefetching[C]//Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on. IEEE, 2001: 293-301.

[42]. Amer A, Long D D E. Noah: Low-cost file access prediction through pairs[C]//Performance, Computing, and Communications, 2001. IEEE International Conference on. IEEE, 2001: 27-33.

[43]. PITKOW J, PIROLLI P. Mining longest repeating subsequence to predict World Wide Web surfing[A]. Usenix, 1999: 139-150

[44]. Mabroukeh N R, Ezeife C I. Semantic-rich markov models for web prefetching[C]//Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on. IEEE, 2009: 465-470..

[45]. Singh S I, Sinha S K. A New Trust Model Based on Time Series Prediction and Markov

Model[J]. Communications in Computer & Information Science, 2010, 101:148-156.

[46]. Korat V G, Pamu K S. Reduction of Data at Namenode in HDFS using harballing Technique. International Journal of Advanced Research in Computer Engineering & Technology, 2012, 1(4): 635-642

[47]. Lai Y, Zhong Zhi S. An efficient data mining framework on hadoop using java persistence API. In: Proceedings of 12th IEEE International Conference on Computer and Information Technology. Chengdu: IEEE, 2010. 203-209

[48]. Mackey G, Sehrish S, Wang J. Improving metadata management for small files in HDFS[C]//Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on. IEEE, 2009: 1-4..

[49]. Vorapongkitipun C, Nupairoj N. Improving performance of small-file accessing in Hadoop[C]//Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on. IEEE, 2014: 200-205..

[50]. 李宇文. Hadoop 平台下文件副本存储改进及小文件合并存取优化的研究[D].武汉：武汉理工大学,2015.

[51]. Xue S J, Pan W B, Fang W. A novel approach in improving I/O performance of small meteorological files on HDFS[C]//Applied Mechanics and Materials. Trans Tech Publications, 2012, 117: 1759-1765.

[52]. Aishwarya K, Sreevatson M C, Babu C, et al. Efficient prefetching technique for storage of heterogeneous small files in Hadoop Distributed File System Federation[C]//Advanced Computing (ICoAC), 2013 Fifth International Conference on. IEEE, 2013: 523-530..

[53]. 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. IEEE International Conference on. IEEE, 2009: 1-8..

[54]. Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li, Ying Li. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files[P]. Services Computing (SCC), 2010 IEEE International Conference on,2010：65-72

[55]. Dong B, Zheng Q, Tian F, et al. An optimized approach for storing and accessing small files on cloud storage[J]. Journal of Network and Computer Applications, 2012, 35(6): 1847-1862..

[56]. Chandrasekar S, Dakshinamurthy R, Seshakumar P G, et al. A novel indexing scheme for efficient handling of small files in hadoop distributed file system[C]//Computer Communication

and Informatics (ICCCI), 2013 International Conference on. IEEE, 2013: 1-8.

[57]. He H , Du Z , Zhang W , et al. Optimization strategy of Hadoop small file storage for big data in healthcare[J]. Journal of Supercomputing, 2016, 72(10):3696-3707.

[58]. Patel A, Mehta M A. A novel approach for efficient handlin g of small files in HDFS[C]// Advance Computing Conference. IEEE, 2015:1258-1262.

[59]. Huang L, Liu J, Meng W. A Review of Various Optimization Schemes of Small Files Storage on Hadoop[C]//2018 37th Chinese Control Conference (CCC). IEEE, 2018: 4500-4506..

[60]. 唐培丽,胡明,张勇.基于中文文本主题提取的分词方法研究[J].吉林工程技术师范学院学报,2005(02):34-36..

[61]. 王惠仙,龙华.基于改进的正向最大匹配中文分词算法研究[J].贵州大学学报(自然科学版),2011,28(05):112-115+119.

[62]. 丁振国,张卓,黎靖.基于 Hash 结构的逆向最大匹配分词算法的改进[J].计算机工程与设计,2008(12):3208-3211+3265.

[63]. 罗燕,赵书良,李晓超,韩玉辉,丁亚飞.基于词频统计的文本关键词提取方法[J].计算机应用,2016,36(03):718-725.

[64]. Dasygenis M, Brockmeyer E, Durinck B, et al. A memory hierarchical layer assigning and prefetching technique to overcome the memory performance/energy bottleneck[C]//Proceedings of the conference on Design, Automation and Test in Europe-Volume 2. IEEE Computer Society, 2005: 946-947.

致　谢

时间是培育成就，获得结果的土壤，南京航空航天大学的两年半的研究生求学之路即将结束了，在这段时间的学习中，我学到了很多专业知识，在实习过程中也收获了专业知识实践运用的一手资料，这些都为以后的工作奠定了基础。此外，经过几个月的努力，完成了毕业论文撰写，我更加感受到了从开始的零散到最后的总结的完整性，如同工作中每个项目的实现，如同人生中每段经历的过程，感恩研究生的学习生涯让我收获了专业知识，收获了一些全新的思维方式，更要感谢那些真诚关心和帮助过我的老师、父母、同学、朋友。

首先，由衷的感谢我的研究生导师张育平教授。在这两年半的时间里，我从老师那里汲取的不仅是知识，更是为人处世的一些准则和对待工作的态度，从老师身上理解到著名教育家陶行知先生的名言——“学深为师，品正为范”这句话的意义。由于自己的本科基础知识较弱，在这两年半时间里张老师在对我的培养倾注了更多心血，对我学习过程中的耐心教导，还有在论文撰写过程中提供专业难题的攻克思路，推荐实践运用的机会等等。言语表达不出内心的感激与感恩，且待他日工作中学以致用老师孜孜不倦给予的专业知识，学习老师严谨的治学态度对待每个任务，学习老师真诚对待身边的每个人。再次感恩我的人生导师——张育平教授。

其次，我要感谢校外实习导师张定林老师。张老师具有丰富的项目经验和分析项目的创新性思维，在实习这段时间内，他对我实习工作的支持与指导为我的研究提供了很多研究素材。还要感谢庞红军项目经理、朱肖项目经理、王鹏飞、张佳雯等团队伙伴，在实习阶段对我的帮助，从他们那一点点了解到公司的业务流程，工作方式以及工作状态，这些都是在书本上拿不到的知识和财富，感谢他们给了我团队合作的安全感。

时间是短暂的，犹如弹指一挥间，时间也是永恒的，在这三年里的每个画面都是定格的，这段日子里一起努力成长的研究生好友：袁鹏泰、胡涵、张天奇、曹雪岳等，你们对时间的珍惜，对目标的追求，对结果的坚持不放弃精神不断影响着我，也不断鼓舞着我。感谢相互支持与相互鼓励的这段日子，更要感谢这段日子里的你们。想要表达的有很多，纸短情长，我还想用些文字感谢我的父母。感谢他们对我无条件的支持和照顾，他们用最朴实的爱支持我往前走。毕业之际也是新的开始，愿用所学回报社会，成长自己，报恩父母。

最后，感谢各位专家和教授的审评，让我所有的努力能被看见，所有的付出能被展示。你们的付出更让我有了自信去应对未来的工作，对未来的实际工作也充满了期待与热情，我将不断努力，通过自我价值的实现去创造更多的社会价值，为国家发展贡献自己的一份力量。

在学期间的研究成果及发表的学术论文

攻读硕士学位期间发表（录用）论文情况

1. 王子炫，魏力，张育平，基于磁光虚拟存储系统的文件调度算法[J].计算机与现代化（已录用）

攻读硕士学位期间申请的专利情况

1. 一种基于 HDFS 光盘库的存储方法。发明专利，申请号：201811443283.1，发明人：王子炫，张育平
2. 一种基于 HDFS 光盘库的存储系统。发明专利，申请号：201811443267.2，发明人：王子炫，张育平