密 级_____

桂林電子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

硕 士 学 位 论 文

题目 基于 Ceph 的海量小文件存取性能优化研究

（英文）Research on Ceph based access performance

Optimization of mass small files

研 究 生 学 号: 1502201039

研 究 生 姓 名: 陆小霞

指导教师姓名、职称: 王勇（教授）

申 请 学 位 门 类: 工学硕士

学 科、专 业: 信息与通信工程

论 文 答 辩 日 期: 2018 年 6 月

独创性（或创新性）声明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得桂林电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：陆小霞　　日期：2018.6.4

关于论文使用授权的说明

本人完全了解桂林电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属桂林电子科技大学。本人保证毕业离校后，发表论文或使用论文工作成果时署名单位仍然为桂林电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。（保密的论文在解密后遵守此规定）

本学位论文属于保密在____年解密后适用本授权书。

本人签名：陆小霞　　日期：2018.6.4

导师签名：王勇　　日期：2018.6.4

摘 要

随着云计算和大数据的迅速发展，全球数据量呈指数递增，尤其是电子商务、社交网络、科学计算等领域产生了越来越多的小文件。面对如此海量的小文件，传统的存储系统由于其设备成本和维护成本等因素已无法满足海量小文件的存储和访问需求，在这一背景下分布式存储系统得到快速发展。

相比其他分布式文件系统，开源的 Ceph 分布式文件系统因其高有效性、高扩展性、高可靠性在企业和研究领域得到了广泛的关注，但 Ceph 分布式文件系统在处理海量小文件时仍存在一些不足：由于 Ceph 集群的数据双倍写入机制，会导致海量小文件的存储效率降低；当小文件被频繁访问时，集群需要在多个存储节点之间不断查找文件，从而影响海量小文件的读取性能。因此，论文给出了一种基于 Ceph 的海量小文件存取优化方案。论文的主要工作及创新点总结如下：

(1) 针对 Ceph 系统数据双倍写入机制导致海量小文件的存储效率低这一问题，给出一种基于文件关联的小文件合并方法。该方法通过文件关联概率计算、关联集合处理将关联性比较大的小文件进行合并，同时将小文件与合并文件间的映射关系生成索引文件，提高小文件的查找效率。

(2) 针对 Ceph 系统在读取海量小文件时性能较低的问题，给出一种小文件预读取方法。该方法通过合并文件块内的利用率及相关率来衡量小文件间的相关性，然后根据其相关性实现小文件的预读取至缓存中，提高小文件的读取效率。

(3) 针对小文件预取缓存的时间局限性而导致缓存空间浪费问题，本文在 LRU 算法的基础上给出一种改进的 LRU_W 算法。该算法通过引入权重衰减因子来动态计算缓存文件的权重，当小文件长时间未被访问时，该文件的权重会随之降低。若权重小于阈值时，该文件会从缓存中移除，这样可以减少缓存空间的浪费，提高缓存文件的命中率。

实验通过对提出的改进方案进行测试，结果表明本文提出的小文件存取优化方法提高了小文件的存储和读取效率。此外，改进后的 LRU 算法通过减少缓存空间的浪费，进一步提高了小文件的读取效率。

关键词：Ceph 分布式文件系统；小文件；关联合并；预取；缓存

Research on Ceph based access performance optimization of mass small files

Abstract

The data is growing exponentially with the rapid development of cloud computing and big data analyzing, especially in e-commerce, social network, scientific computing and so on, where more and more small files are produced. The traditional storage systems have been insufficient to fulfill the storage requirements of such large number of small files because of its equipment and maintenance costs. In this context, the distributed storage systems need more improvements.

The Ceph distributed file system has been widely researched and deployed by enterprises because of high effectiveness, high scalability and high reliability. However, the Ceph distributed file system still has some disadvantages when dealing with large number of small files. The write efficiency of large number of small files will be reduced due to the double writing mechanism. When small files are frequently accessed, the cluster has to constantly locate files in storage nodes, which affects the reading performance when dealing with large number of small files. An optimization scheme for massive small file access based on Ceph is proposed in this thesis. The main work and innovations are summarized as follows:

(1) In order to improve the write efficiency of massive small files in Ceph distributed file system, a method of combining small files based on file association has been proposed. This method combines file association probability calculation and association set processing to merge small files with correlation level. Then, generates index files between small and merged files, so as to improve the efficiency of small files.

(2) To address the issue of low performance when reading large number of small files in Ceph system, a small file pre-reading method is proposed. The correlation between small files can be measured by combining the utilization rate and the correlation rate inside the file blocks. According to the relevance of files, the pre-reading process of small files is taken with cache to improve the reading efficiency of the small files.

(3) Aiming at the time consuming of pre-reading cache for small files and the following wasted buffer space. We propose an improved LRU_W algorithm based on LRU algorithm. The algorithm calculates the weight of the cache files dynamically by combining with weight attenuation factors. When the small files is inactive for a long time, the weight of the files will be reduced. If the weight is less than the threshold value, the

files will be removed from the cache in order to the reduce the waste of the cache space and increase the reading accuracy of the cache files.

Experiments show that, the small file access optimization method proposed in this thesis improves the writing and reading efficiency of small files. In addition, the improved LRU algorithm further improves the efficiency of reading small files by reducing the waste of buffer space.

Keywords: Ceph distributed file system; Small file; Merger; Prefetch; Cache

目 录

第一章 绪论

§1.1 研究背景及意义

随着云计算[1]和大数据[2]的迅速发展，全球数据量呈指数递增，传统的存储系统由于其设备成本和维护成本等因素已慢慢不能满足人们的存储需求。据统计淘宝网已存储超过 286 亿张图片文件，文件总量达到 1PB，文件的平均大小为 17.45KB 左右，其中 8KB 以下的文件达到了 61%；Facebook[3]平台已存储 600 亿张照片且每天分享约 20 亿张照片；这些互联网企业在云平台上保存的海量文件中，主要以图片、文档、日志文件为主，这些文件的大小基本都在 100KB 甚至 20KB 以下，具有海量、多样、动态等特点[4]。

根据美国西北太平洋国家实验室的研究报告可知，其文件系统中存储着 1200 万个文件，文件大小低于 64MB 的占 94%，低于 64KB 的文件占了 58%[5]，由此可见这些存储文件中大部分为小文件。随着小文件数量的逐渐增长，小文件存储和读取效率较低的问题也随之越来越突出，因此，如何进行存储、管理这些小文件也是越来越多平台研究的重点。例如 Hadoop[6]分布式文件系统提出了 Hadoop Archive[7]、Sequence file[7]和 Combine File Input Format[9]三种方案通过小文件的合并来解决小文件存储问题，但仍存在着几点不足：

(1) 在小文件进行合并时没有考虑小文件间关联关系，没有给出相应的小文件预取机制；

(2) 小文件合并时没有考虑小文件跨数据块存储问题，也即是将若干小文件合并成一个很大的文件，合并后的大文件块会被分割成不同的数据块存储，从而可能使得一个小文件分别存储在不同的数据块上。用户读取文件时，需要在不同数据块间跳跃读取，从而增加了小文件的读取时间；

(3) 没有建立相应的索引机制，在小文件访问时需要遍历所有大文件，这样会导致小文件读取效率明显降低。

针对小文件存储方面的这些不足，论文给出一种基于现有存储系统 Ceph 的小文件关联合并及小文件预读取方法。优化系统的实现不仅可以提高文件的存储效率，也可以通过小文件预读取来减少用户和集群的交互，可以较好的提高文件的访问效率。

§1.2 国内外研究现状

§1.2.1 云存储研究现状

随着全球数据量呈指数递增，传统的存储方式已经渐渐不能满足人们的存储需

求。与传统存储方式不同的是，分布式存储则是利用互联网将每台普通计算机的存储资源统一管理起来，然后以存储接口的形式对外提供存储服务。与传统的存储方式相比，分布式存储具有极强的可靠性、可用性及可扩展性。

云存储服务在国内外都受到了学者和企业的广泛关注。如国外的 IBM、Google、亚马逊等企业都拥有自己的云存储技术，并开发出了属于自己的云平台系统，这些云平台系统不仅可以为企业服务也可以为个人提供服务[10]。目前主流的分布式文件系统主要有：Google 为了解决海量数据存储问题，设计实现了分布式存储系统 BigTable[11]；Google 开发实现的可扩展的分布式文件系统 GFS[12](Google File System)；Apache Software Foundation 公司推出的 Hadoop 分布式文件系统 HDFS[13](Hadoop Distributed File System)；Sage Weil 在攻读博士期间的研究项目而开发出的 Ceph[14][15]分布式文件系统等。云存储在国内的起步相比于国外较晚一些，很多公司也不断加入到云存储的研究和开发中，如腾讯公司研发实现的 TDW 离线数据平台，该平台的集群存储容量高达 100PB[16]；2016 年 11 月 21 日阿里宣布在境外新开设的 4 个数据中心正式开始运行[17]；百度公司研究开发的百度云等，由此可见，国内企业对云存储的研究也越来越重视。

各个公司推出的平台虽然不同，但原理是相同的，这些公司均是利用分布式技术，通过相应的映射机制将平台上的文件和存储设备之间建立相应的映射关系，实现存储系统的数据管理。但不同企业研发的分布式存储系统在其数据的分布策略方面略有不同，根据系统对数据对象的管理以及组织方式可以分为：基于元数据服务器的组织结构[18]和无元数据服务器的组织结构[19]。其中 BigTable、GFS 和 Hadoop 采用的是第一种方式，而 Ceph 采用的是第二种方式，是一种无中心化的分布式文件系统，该系统目前也被企业广泛关注。

§1.2.2 小文件研究现状

随着小文件数量的不断增大，大部分分布式存储系统已经不能满足小文件的高效存储和读取的需求。例如 HDFS 的设计是为了解决几百 MB 甚至几百 TB 的大文件，并对这些大文件以 64M 一块的单位大小进行分块存储，而小文件由于文件较小而不需要进行分块存储，因此当 HDFS 存储海量小文件时，会引起 HDFS 的 NameNode[20][21]节点空间浪费等瓶颈问题。面对越来越多的小文件，如何高效存储和访问这些海量小文件是目前各个平台重点关注的问题。

针对上述问题，目前主要有两种解决方法：(1)是小文件合并，通过将若干个小文件合并为一个大文件，减少小文件元数据的数量；(2)是文件的预读取，在读取文件时将与其相邻的小文件一起读出并存入缓存中，减少用户与集群的交互以提高小文件的访问效率。

Zheng Tong[22]等人针对 HDFS 单个 NameNode 的存储架构会导致其性能严重降

低问题，提出一种小文件合并成大文件方案，同时建立小文件到大文件的映射关系存储到 Hbase 中。此外，为了提高文件读取速度，提出了一种基于 LRU 的预取方法，有效的提高了小文件的读取效率，但并未考虑小文件之间的关联关系。

You Xiaorong[23]等人针对 Hadoop 在存储海量小文件时，由于 NameNode 内存消耗高导致其存储性能较低的问题，提出一种海量教育资源的小文件存储优化方法。该方法根据教育资源间的相关性，将关联小文件合并成大文件存储至 HDFS，减少了小文件的元数据数量，缓解了 NameNode 的内存瓶颈问题；读取文件时，利用小文件预取机制，将关联教育小文件预取至缓存中，提高了文件的访问效率。

Fu Songlin[24]等人提出一种基于扁平式元数据存储方法的轻量级分布式文件系统，该系统替代传统的分布式文件系统用于小文件的存储和访问，不仅大大简化了元数据的访问过程，也提高了数据服务器在存储和读取小文件时的性能，提升了分布式文件存储系统的性能。

Li Hongqi[25]等人提出一种面向海量小文件的分布式存储系统设计方案，该系统通过使用 I/O 复用和异步调用来提高系统的并发性及扩展性，并设计实现了块存储方式以减少文件碎片，保证了系统的稳定性，但没有考虑文件之间的关联性及预读取机制。

Wang Tao[26]等人针对云存储分布式系统在访问海量小文时效率较低等问题，提出了基于用户访问任务的小文件合并及预取策略。该策略利用概率潜语义分析（Probability Latent Semantic Analysis，PLSA）模型来挖掘分析用户访问任务和访问文件间存在的关系，将属于同一个访问任务的小文件进行合并，提高了云存储系统的访问效率。

Li Linyang[27]等人针对 HDFS 处理海量 GNSS 小文件效率不高的问题，结合 GNSS 数据类型、特点以及存储过程，提出了一种新的 GNSS 小文件云存储方法。该方法分别根据观测文件和解算成果的类型进行合并，对合并后的文件生成压缩 Trie 树索引，索引切分后，根据匹配算法分布式地存储索引块，实现了海量 GNSS 小文件的高效存储。

Meng Bing[28]等人提出一种基于 TLB 映射合并机制实现小文件的合并，减少了文件元数据的数量，有效降低了 HDFS 的 NameNode 的内存消耗，并利用先验预取策略提高了文件的访问效率。

Mu Yanliang[29]等人针对 Ceph 系统的核心算法 CRUSH[30]在处理海量小文件时，存在高相关性的小文件会落入同一个数据存储节点的问题，提出了一种基于温度因子的 CRUSH 改进算法，该算法可以有效的解决由海量小文件存储导致的负载均衡问题。

§1.2.3 缓存技术研究现状

分布式系统在处理海量小文件时，通过优化小文件处理算法所带来的效果远不如系统利用缓存所带来的效果。因此，为了提高分布式文件存储系统的性能，针对不同的应用场景设计实现的存储系统，会采用不同的缓存替换方法。

目前，比较常见的几种缓存替换算法分别是最近最少使用算法 LRU[31](Least Recently Used)、最不经常使用算法 LFU[32] (Least Frequently Used)、先进先出算法 FIFO[33](First In First Out)和最近最常使用算法 MRU[34](Most Recently Used)。这些缓存算法主要是基于访问时间或者基于访问频率等实现的，其实现方式简单且算法复杂度比较低。

Perkowitz S[35]等人针对 LRU 算法的时间局部性问题，提出了一种改进的 LRU-Threshold 缓存算法； Ding Jian[36]等人针对 LFU 缓存算法的缓存污染问题，提出了一种 LFU-Aging 缓存算；Niu Dejiao[37]等人针对云存储系统中数据的存储、管理以及用户访问的特性设计了一种基于影响因子的数据缓存策略、元数据关联淘汰算法，可以实现动态调整缓存大小，淘汰影响因子比较小的数据，可以提高缓存的命中率。这些改进算法虽然都能提高缓存命中率，但其仍只是侧重缓存对象的访问历史记录。

为了进一步提高缓存性能，Huang Xueyu[38]等人在缓存替换策略的基础上引入了缓存预测模块,该模块可以根据分析缓存对象的 历史访问信息对用户可能会访问的数据集进行预测，提前将这些可能被访问的数据存储在缓存中，可以减少用户的访问时间从而提高系统的整体性能。当缓存容量不足时，根据原缓存算法实现缓存对象的移除，并且该缓存对象不在预测数据集中。基于预测的缓存替换算法主要侧重于数据预测模块，不同缓存策略的预测方式各不相同。最常用的缓存预取策略分别是：基于用户访问概率的预取策略[39]、基于数据挖掘的预取策略[40]以及基于神经网络的预取策略[41]。

§1.3 研究目标与内容

本文研究对象是基于 Ceph 的海量小文件存储系统，针对 Ceph 系统在处理海量小文件时存在的存取效率较低问题，给出一种小文件关联合并算法及小文件预读取算法，提高小文件的存取效率。围绕着研究目标，论文的工作内容主要有以下几个方面：

(1) 小文件关联合并方案：在海量小文件存储过程中，首先根据小文件访问的历史日志计算出小文件间的关联概率，然后根据文件关联概率将小文件进行关联合并，随后存储至 Ceph 集群；与此同时将小文件与合并文件间的映射关系生成索引文件并存储在客户端中，以提高小文件的查找效率。

(2) 小文件的预读取方案：在读取小文件时，可以通过合并文件块的相关率及利用率来衡量小文件间的相关性，随后将与该文件相关的小文件预读取出来并存储在缓存中，这样可以减少用户与集群的交互，提高小文件的访问效率。

(3) 小文件预取缓存优化方案：随着小文件不断预读取并存储在缓存中，由于LRU 算法的时间局限性，会出现有些文件长时间不被访问而浪费缓存空间，对此论文给出了一种改进的缓存替换算法 LRU_W。该算法在 LRU 算法的基础上加入了权重因子。若小文件长时间不被访问，则其权重随之下降，当权重小于阈值时，该文件会从缓存中移除。若权重较高则将其存储至优先级较高的二级缓存中，这样不仅可以减少缓存空间的浪费，也可以保证权重高的小文件具有较高的优先级，以达到提高缓存中文件命中率的目的。

(4) 搭建实际的 Ceph 集群环境，挂载 RBD 块设备客户端，通过 RBD 的 API 接口实现系统的批量存储和读取功能，并通过该集群分别对论文给出的方法和原型进行了实验，验证论文给出的方法对小文件存储和访问性能提高的有效性。

§1.4 论文的组织结构

本文的组织结构共分为六章，详细内容如下：

第一章：绪论。阐述了在云计算和大数据迅速发展下研究海量小文件的存储意义，对国内外小文件的研究方法进行了分析，并介绍了文件缓存技术的研究现状，明确了论文的研究目标和实现内容，最后给出了论文的主要组织结构。

第二章：相关技术。简单介绍了 Ceph 分布式文件系统的体系架构及数据存储过程，然后介绍了系统客户端存储接口的实现及其优缺点，最后阐述了小文件缓存技术。

第三章：海量小文件的存取性能优化方案。介绍 Ceph 在存储小文件时存在的问题，然后针对这些问题给出小文件关联合并方法及小文件预读取方法，最后设计了优化后 Ceph 系统和原 Ceph 系统的对比实验，证明了优化后 Ceph 系统能有效提高小文件的存储和读取效率。

第四章：小文件预取缓存算法的改进与实现。首先介绍了缓存设计思路及缓存预取策略，然后根据 LRU 算法的时间局限性给出一种改进的缓存算法。最后通过实验证明该算法能有效提高缓存的命中率及小文件的读取效率。

第五章：海量小文件存储系统的设计和实现。介绍系统的整体设计及各个模块的实现，并展示了系统登录、系统上传、系统下载、系统状态查询等界面。

第六章：总结与展望。首先对全文的研究工作进行了全面的总结，提出当前系统所存在的问题，并针对这些问题阐述下一步研究工作。

第二章 相关技术

本章主要介绍了 Ceph 分布式文件系统的体系架构及数据存储过程，同时介绍了系统客户端存储接口的实现原理及优缺点。最后阐述了小文件缓存技术，为海量小文件存储系统中多级缓存模块的设计与实现奠定了理论基础。

§2.1 Ceph 分布式文件系统

Ceph 是一种基于对象存储的无中心化的分布式文件系统，由于采用了对象存储的方式，其数据处理过程高度并行化，Ceph 系统能够通过添加普通服务器来扩大集群，可轻松将存储规模扩展至 PB 级。而 CRUSH 作为 Ceph 集群的核心算法可以动态的计算出各个文件的存储位置，实现了文件元数据的服务器功能，使之成为了一个即使出现单节点故障也能正常使用的存储系统。正是由于其高有效性、高可靠性及高扩展性使得 Ceph 分布式存储系统不仅在企业也在科研领域都得到了广泛关注[42]。

§2.1.1 Ceph 系统体系架构

Ceph 分布式文件存储系统的基本组件主要由监控节点 MON、对象存储节点 OSD、元数据服务器 MDS 和客户端 Clients 四部分构成[43]，各部分组件的具体功能的详细介绍如下。

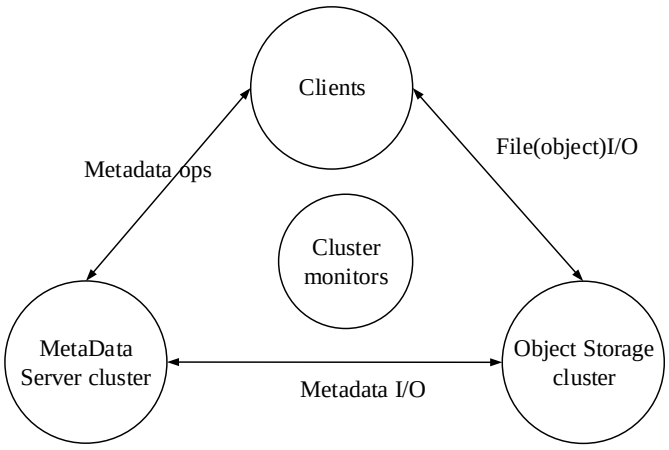

图 2.1 Ceph 系统组件

(1) OSD(Object Storage Device)节点：该节点的主要作用是存储和备份用户上传的数据，当集群出现故障或宕机后可以恢复集群丢失数据；当增加或删除 OSD 节点时，可以自动迁移数据以平衡各个节点的数据分布，同时可以实现周期性的监测自身情况并向 MON 节点发送具体信息情况。

(2) MON 节点：监控节点主要功能是监控和维护 Ceph 集群的 Mon map、OSD

map、MDS map 等各种状态信息。Ceph 分布式文件系统之所以能够实现去中心化，是因为该集群可以包含多个监控节点。MON 使用 Paxos[44]算法交互和维护 Ceph 集群信息，以达到信息的一致性。MON 节点可以根据 OSD 节点发送过来的心跳信息来调整 Ceph 集群的 cluster map 信息，可以自动调整数据分布的平衡。

(3) MDS(MetaData Server)节点：元数据服务器[45]节点的主要作用是集群用于存储上传文件的元数据，但该节点仅适用于 CEPHFS，而不能为 RBD 和 RADOSGWD 等分布式文件存储系统使用。

(4) Clients 节点：客户端节点的主要作用是当用户通过客户端上传或下载数据时，首先需要从 MON 节点获取集群状态信息，然后使用 CRUSH 算法计算出存储数据对象的主从 OSD 节点。这种原本由集群完成的计算任务，现在由客户端来完成，这样就可以减少 Ceph 集群的对计算能力的需求。

§2.1.2 数据存储过程

当数据存储在 Ceph 分布式文件系统时，不管是采用 RBD 块存储方式还是采用 RADOSGW 对象存储方式，存储的文件都会被切分成一定大小的能够使 RADOS[46] 操作的对象(Object)，但这些对象并不是直接存储到 Ceph 中，而是采用归置簇 PG 的方式，将对象映射到一个 PG 中，最后 Ceph 利用 CRUSH 算法把 PG 映射到不同的 OSD。其数据存储过程可概括为三个映射过程：(1) File -> Object；(2) Object -> PG；(3) PG -> OSD，具体存储过程如图 2.2 所示。

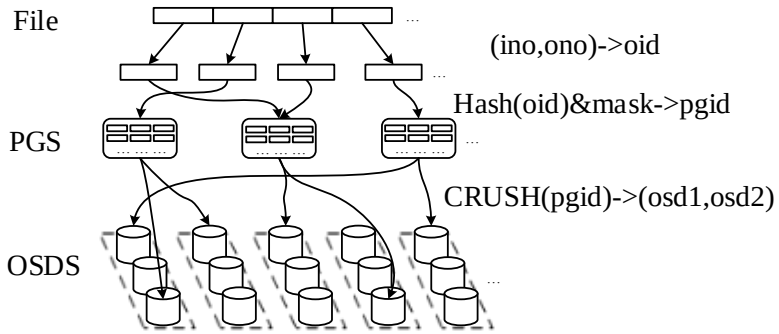

图 2.2 Ceph 数据存储流程

(1) File -> Object：Ceph 集群首先将用户上传的文件按照数据块的大小切分成能够被 RADOS 处理的对象，其中数据块的大小一般为 2M 或 4M。文件分割的优点是：一方面可以让不同大小的文件变成大小一致的数据块，提高 RADOS 的管理效率；另一方面可以使原本单一文件的串行存储变成多个对象文件之后的并行操作。

(2) Object -> PG：这一映射过程使得每个对象都会被唯一的映射到一个 PG 中去，计算对象映射到 PG 中的主要有两个步骤：1)是将对象(oid)作为 Hash 的输入值计算得到一个随机数。2)然后把这个随机数与 mask 按位进行与操作，计算出相应

PG 的值。其中 PG 的值一般为 2^N，N 为整数，而 mask 为 PG 的值减一。

(3) PG -> OSD：该映射过程将承载多个对象的 PG 映射到实际的物理存储节点 OSD 中，这一映射过程是由 CRUSH 算法完成的，其计算公式如 2-1 所示：

$$CRUSH\left(pgid, crush\_map, ruleno\right)=\left(osd_0,\ osd_1,\ osd_2 \ldots\ osd_n\right) \qquad (2\text{-}1)$$

其中，pgid 表示 Object ->PG 映射得到的 PG 编号，crush_map 是包含集群拓扑结构等信息的集群映射，ruleno 为 PG 的放置规则，osd_n 表示第 n 个物理存储设备，其数量由集群的副本数决定。但是 PG 到 OSD 的映射并不是永恒不变的，当集群宕机或者增删集群的 OSD 节点或集群的放置规则发生改变时，PG 到 OSD 的映射关系也会自动调整。

§2.2 系统客户端存储接口

随着云存储的应用越来越广，为了满足用户在不同场合的需求，Ceph 分布式存储系统提出了三种不同的存储接口：分别是 RBD、RADOSGW 和 CEPHFS。

(1) RBD：RBD 是根据最底层的 RADOS 开发和实现的，并利用 librbd 库使每一个文件块都按顺序的条带化的方式存储在集群中，这就意味着每一个块设备都可能被分布到各个 OSD 节点上。RBD 的主要作用是为用户提供高可靠性的分布式块存储磁盘，其还具有动态调整块设备的大小、可以在内存中缓存数据、用户在写入数据的同时会复制克隆备份数据等特性，这些特性让 RBD 块设备存储更加适用于企业的开发和应用，Ceph 块设备存储结构如图 2.3 所示。

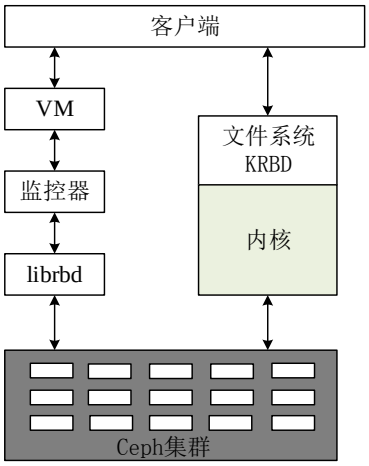

图 2.3 Ceph 块设备存储

由图 2.3 可以看出，RBD 块设备为 Ceph 分布式文件系统提供了两种存储接口，一是经过 QEMU Driver 为虚拟机提供存储客户端接口；二是操作文件系统(KRBD)的内核态中设计了一个内核模块，用户可以利用内核模块将块设备映射到相应的物理主机上，然后可以由物理主机直接进行访问。

Ceph 分布式存储系统提供的 RBD 块存储接口的优点是：RBD 用作虚拟机的硬

盘，可以面向用户通用的需求，不仅可以实现大文件的读写，而且也可以实现小文件的读写，而且块存储对延迟有比较高的要求。

(2) RADOSGW 接口：基于对象的存储系统是通过将对象网关建立在 RADOS 层上提供的存储接口，可以实现文件的上传和下载功能。RADOSGW 可以实现应用程序直接与对象存储建立连接，是因为其采用了 librados 库和 librgw。RADOSGW 不仅为 S3 提供了对象存储接口而且也为 OpenStack Swift[47]提供了相应的存储接口，Ceph 对象存储结构如图 2.4 所示。

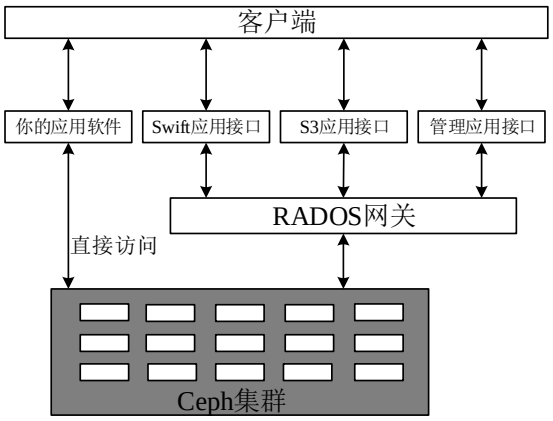

图 2.4 Ceph 对象存储结构

采用 RADOSGW 接口的存储系统有两个特点：一是 RESTful[48][49]存储接口，该接口可以为存储系统提供 get、put、del 等操作接口，对应对象存储系统中文件的上传、文件的下载以及删除等操作。二是对象存储系统利用扁平式的数据结构，这种数据结构有利于对象存储系统的无限容量的扩展。

(3) CEPHFS：CEPHFS 是一个标准的分布式文件系统，该系统利用 RADOS 实现数据的存储和读取过程，并且需要至少一个 MDS 管理系统中文件的元数据，以实现元数据和数据的分离，这样就可以大大降低文件系统实现的复杂性，也能够增加其可靠性，CEPHFS 的总体结构如图 2.5 所示。

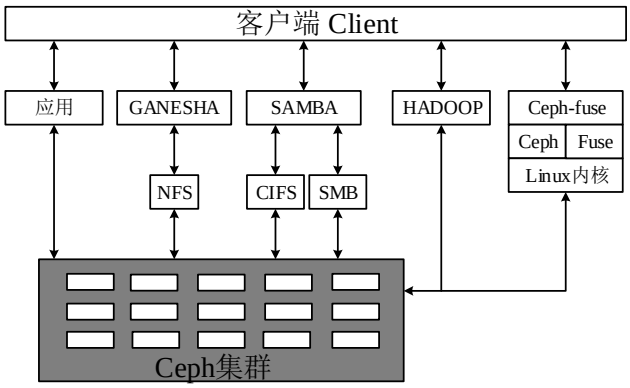

图 2.5 CEPHFS 的总体结构

§2.3 缓存技术研究概述

缓存替换算法是用于计算设备中哪些条目需要从设备中移除，提高设备的利用率。常见的几种缓存替换算法有：最近最少使用算法(LRU)、最不经常使用算法(LFU)先进先出算法(FIFO)以及最近最常使用算法(MRU)等。这些缓存算法主要是基于访问时间间隔或基于访问频率实现的。

§2.3.1 基于访问时间间隔的缓存替换算法

本节主要介绍了基于访问时间间隔的缓存替换算法的实现原理及实现过程，包括 LRU 算法及其改进的 Pitkow/Recker[50]缓存算法，并介绍了算法的优缺点。

(1) LRU 算法设计实现的核心思想是数据最近最少被访问，因此未来该数据被访问的可能性也很低，可以从缓存中移除。该缓存替换算法是根据时间局部性原理设计实现的，根据用户访问的历史行为来作为用户未来访问数据的依据。

LRU 缓存替换算法将缓存中的文件根据最近一次的访问时间进行排序，主要实现过程是：当文件新加入缓存时，记录用户访问时间，同时将其放置在缓存队列的头部；当文件再次被访问时，一方面更新缓存对象的访问时间，另一方面将被访问的文件移到缓存队列的头部；当缓存容量不足时，将长时间未被访问的文件从缓存队列的尾部淘汰。LRU 算法最常见的实现方式是利用链表结构来保存缓存数据，该算法比较适用于高局部性的数据访问场合。LRU 算法的详细实现过程如下图 2.6 所示。

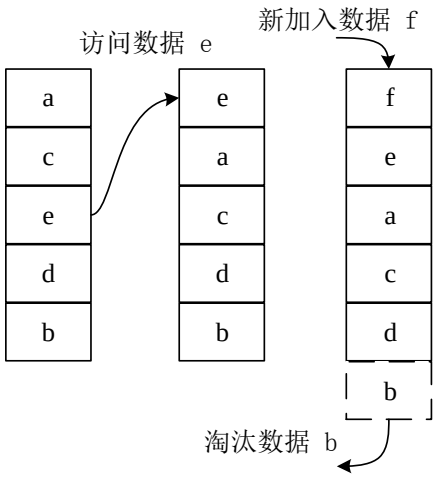

图 2.6 LRU 算法的实现过程

LRU 算法的优点是实现方式相对简单且时间复杂度为 O(1)，因此该算法被各个领域广泛应用。缺点是没有考虑缓存文件的访问频率及延迟时间。

(2) Pitkow/Recker 改进算法的核心思想是先将缓存中的缓存文件根据其访问时间进行排序，如果缓存文件的访问时间在一天之内，就根据缓存文件的大小进行替

换。如果缓存文件的访问时间均在一天前，则根据 LRU 缓存算法进行缓存替换。

§2.3.2 基于访问频率的缓存替换算法

本节主要介绍了基于访问频率的缓存替换算法的实现原理及过程，包括 LFU 算法及其改进的 LFU-Aging 缓存算法，并介绍了算法的优缺点。

(1) LFU 算法的核心思想是数据最近使用的次数最少，因此该数据将来被访问的概率也很低，可以从缓存中移除。LRU 缓存替换算法根据缓存对象的访问频率进行排序，若有新的数据加入缓存，记录该数据的访问次数并将其放置到相应的位置。当缓存空间不足时，则优先从缓存中淘汰访问次数最少的缓存文件。其详细实现算法如下图 2.7 所示。

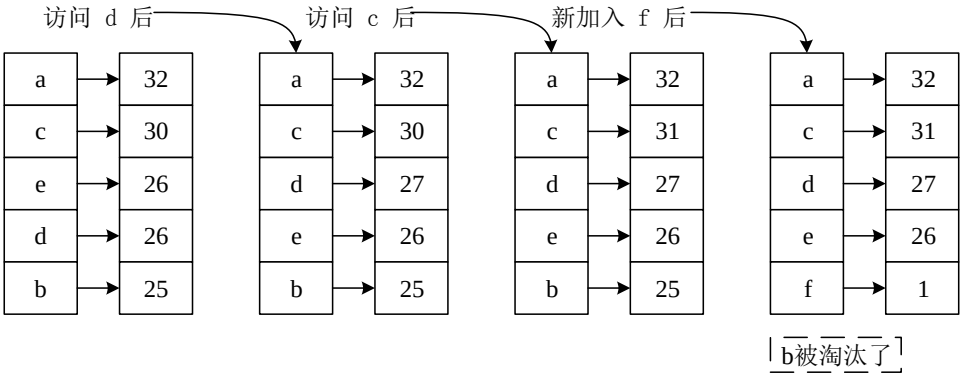

图 2.7 LFU 算法的实现过程

LFU 缓存替换算法的优点是时间复杂度比较低，比较适用于在一段时间内，缓存对象改变不大的系统。缺点是该算法并没有考虑某些数据可能只是在近期访问的频率很高，但之后再也不曾访问的情况，由于其访问频率较高，会长时间存储在缓存中，从而导致缓存空间的浪费。

(2) LFU-Aging 算法解决了 LFU 算法的缓存空间浪费问题，改进的 LFU-Aging 算法根据缓存对象的访问频率，设置了一个阈值 T，当缓存中的缓存文件的所有访问次数的平均值超过给定阈值，缓存中缓存文件的访问次数减半，因此改进后的算法可以很好的解决 LFU 算法的缓存污染问题。

§2.3.3 基于预测的缓存替换算法

LRU 算法及 LFU 算法等传统缓存算法都是根据用户访问的历史信息，决定缓存中的缓存对象。因此，会导致缓存中有些文件长时间未被访问而导致缓存空间的浪费，降低了缓存文件的命中率。

为了提高缓存的命中率，Kim H[51]等人提出了一种基于预测的缓存替换算法。该算法在现有缓存算法的基础上引入预测机制，该机制根据缓存对象的历史访问信息对用户可能会访问的数据集进行预测，可以提前将这些数据预取出来并存储在缓存中，提高了缓存文件的命中率。

基于预测的缓存替换算法的核心思想是：当有新数据加入缓存且缓存容量不足时，根据原缓存算法实现缓存对象的淘汰，并且保证该缓存对象不在预测数据集中。基于预测的缓存替换算法主要侧重于数据预测模块，不同缓存策略的预测方式各不相同。

§2.4 本章小结

本章首先介绍了 Ceph 分布式存储系统的基本组件，并详细的介绍了 Ceph 集群中各个组件的具体功能，介绍了系统数据存储过程等。然后介绍了系统客户端的三种存储接口的实现及优缺点，最后主要介绍了 LRU 算法、LFU 缓存算法以及基于预测的缓存替换算法的实现原理。

第三章 海量小文件存取性能优化方案

本章首先分析了 Ceph 分布式文件系统在存储海量小文件过程中存在的问题，然后根据需要解决的问题，给出一种小文件关联合并方法和文件预读取方法。实验证明，当 Ceph 集群处理海量小文件时，使用论文给出的方法可以较好的提高小文件的存储和读取效率。

§3.1 Ceph 小文件问题

小文件是指那些在存储和访问效率方面，相比于其它大小的文件性能都大大降低，大多数分布式系统在小文件存储上都存在着很大的瓶颈问题[52]。而 Ceph 在处理小文件时面临的问题主要为如下几个方面：

(1) Ceph 集群中的本地存储接口(FileStore)为了支持事务，在 Ceph 集群中引入了日志机制，使得所有小文件的写入都需要先写入日志文件，再通过对象存储接口(ObjectStore)写入本地文件系统。在面对海量小文件存储时，由于数据的双倍写入，会导致 Ceph 集群实际磁盘的 I/O 吞吐量是其物理性能的一半，导致集群的写性能较低。其中，FileStore 的静态类图如图 3.1 所示。

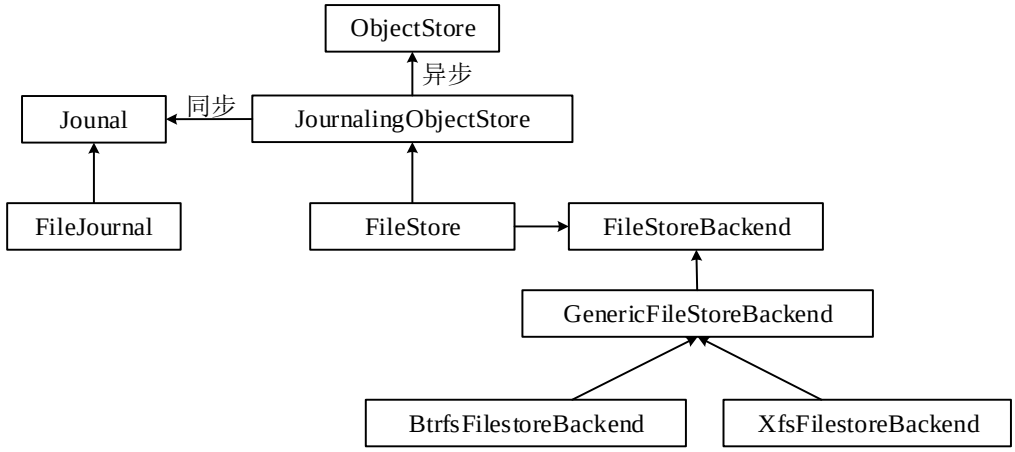

图 3.1 FileStore 的静态类图

(2) Ceph 集群具有高度的数据一致性，其对副本的一致性要求特别高。Ceph 具有不同于其它分布式文件系统的副本机制。Ceph 集群在进行文件写操作时，文件首先需要写入到集群的 Primary OSD，只有当文件完全写入主副本的 OSD 后再同时向一个或多个 Secondary OSD 写入相同的数据，当文件完全写入所有的 OSD 后，Primary OSD 立即向客户端发出响应也即是上传成功。当面对文件规模特别大的情况时，就会大大降低 Ceph 的存储效率。其副本一致性机制读写过程如图 3.2 所示。

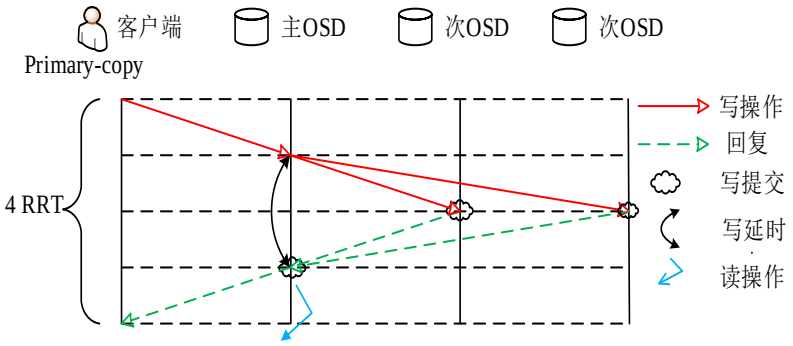

图 3.2 副本一致性机制读写过程

(3) 当数据存储至 Ceph 分布式文件系统时，主要经过三个映射过程 1) File -> Object 映射；2) Object -> PG 映射；3) PG -> OSD 映射，其对象寻址过程如图 3.3 所示。当 Ceph 集群需要存储海量小文件时，会有两方面的时间消耗。一是 Object -> PG 映射阶段，通过计算文件名映射到 PG 所带来的时间消耗；二是 PG -> OSD 映射阶段，通过计算数据存储至 OSD 结点所带来的网络传输时间消耗。当 Ceph 存储文件数量较少时，不会有太多的时间消耗，但当存储海量小文件时，这两个阶段的时间消耗是巨大的，并且由于每个小文件都是与 Ceph 集群的结点直接建立的请求，当文件数量很大的时将会给集群结点带来很大的网络通信和 I/O 压力，因此 Ceph 集群的读写性能将会大大降低。

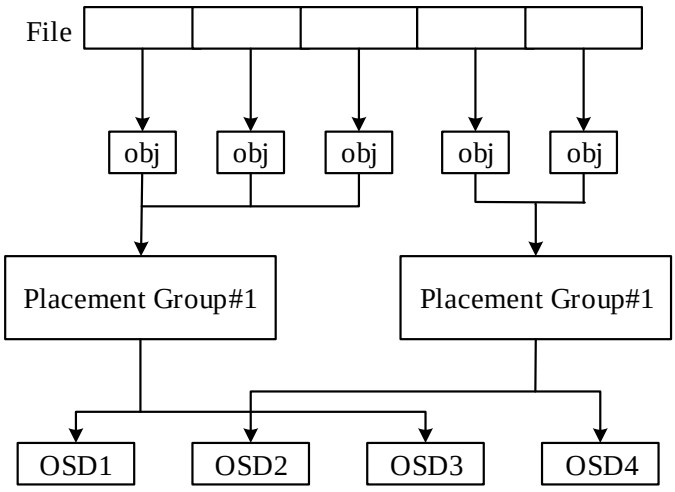

图 3.3 对象寻址过程

针对上述 Ceph 系统在处理海量小文件时存在的问题，给出一种基于文件关联的小文合并方法及文件预读取方法。

§3.2 总体设计

本文所给出的基于 Ceph 的海量小文件的存取性能优化方法的整体设计框架如图 3.4 所示。

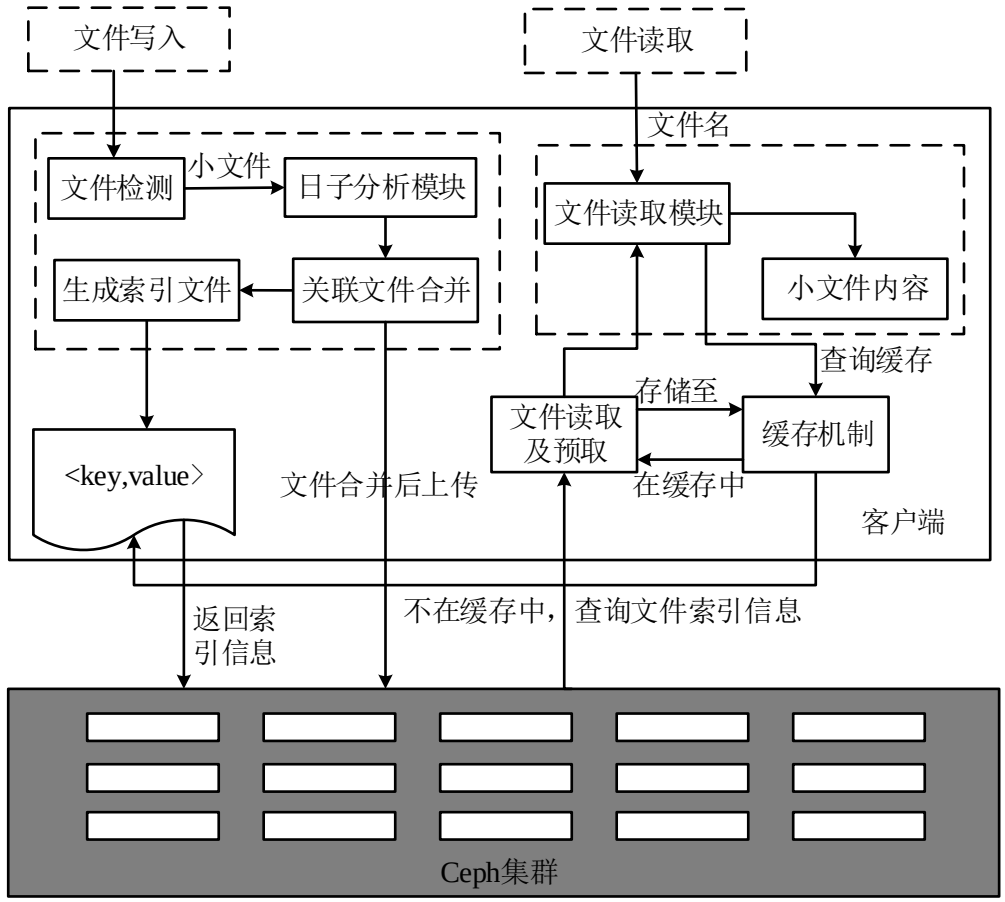

图 3.4 小文件优化整体设计框图

由图 3.4 可以看出，针对海量小文件的存取效率较低的问题，论文主要从存储和读取两方面进行优化。一方面是在文件写入时，首先利用日志分析模块分析计算出小文件间的关联概率，然后再根据文件关联概率设计出文件关联合并算法实现小文件的合并后再存储到 Ceph 集群。在小文件关联合并过程中，将小文件与合并文件间的映射关系生成索引文件并存储到客户端中，提高小文件的查找效率；另一方面是在用户读取小文件时，首先判断该文件是否在缓存中，若在缓存中，直接从缓存中读取文件并返回请求文件；否则客户端将用户请求发送给 Ceph 集群，通过索引文件获取文件在合并文件块内的位置信息，并从 Ceph 集群中读取出请求文件；然后根据小文件间的相关性实现小文件的预读取并存储到缓存中，通过减少用户与集群的交互，从而减少用户的访问时间以提高小文件的读取效率。

§3.3 小文件存储预处理

小文件存储预处理主要实现了小文件间关联概率的计算及小文件关联集合处理。 小文件关联概率计算模块可以动态的根据小文件的历史访问日志，获得小文件间的关联集合，然后利用小文件关联合并算法对文件关联集合进行合并处理。

§3.3.1 文件关联概率算法

计算小文件间的关联概率，首先需要通过分析小文件的历史访问日志，并统计出小文件 A 和 B 分别被访问的次数 N_A 和 N_B，文件 A 和 B 在一段时间内都被访问的次数 N_{AB} 及小文件历史访问日志总数 N，然后通过公式 3-1 和公式 3-2 分别计算出小文件间的关联概率。

(1) $P(B|A)$：表示用户在访问过文件 A 之后一段时间后访问文件 B 的概率，

$$P(B|A)=\frac{N_B}{N_A} \tag{3-1}$$

式 3-1 中，N_A 表示用户访问文件 A 的次数，N_B 表示用户访问文件 B 的次数。

(2) $P(AB)$：表示一段时间间隔内，用户同时访问文件 A 和文件 B 的概率，

$$P(AB)=\frac{N_{AB}}{N} \tag{3-2}$$

式 3-2 中，N 为历史文件访问日志总数，N_{AB} 为 T 时间内文件 A 和文件 B 均被访问的次数。

$P(B|A)$ 和 $P(AB)$ 同时需要满足以下条件：

$$(fileA, fileB)=\{P(B|A)\geq \min P(B|A)\&\&P(AB)\geq \min P(AB) \tag{3-3}$$

当文件 fileA 与文件 fileB 间的关联概率均大于最小阈值时，则认为 fileA 和 fileB 之间是相关的。文件关联概率统计算法的伪代码如表 3-1 所示，其中输入为小文件历史访问日志集合，输出为小文件的关联集合。

表 3-1 文件关联概率统计算法

输入：文件历史访问日志
输出：文件关联集合

```
1.  Count(Logs){
2.    dict={ }    / / 文件的访问次数
3.    for log in Logs: / /遍历日志
4.      for i→len(Logs):
5.        log_T=Logs[i]
6.        if Log_T.time-log.time<T: / / 统计 T 时间内各个小文件的访问次数
7.          name=log_T.name
8.          dict[name]++
```

```
9.                else:
10.                   break
11.               end if
12.       end for
13.   end for
14.           N_A=dict[A]
15.           N_B=dict[B]
16.           P(B|A)= N_B/N_A
17.           P(AB)=(N_B*N_A)/N^2
18.           if P(B|A)>=min P(B|A) and P(AB)>=min P(AB):
19.              file_set.add(fileA,fileB)   / /  文件关联集合
20.           else:
21.              remove fileB
22.           end if
23.   return file_set   / /返回文件关联集合
24.   }
```

§3.3.2 关联关系合并

在关联关系(file1,file2,…, file(n))中，其中，称 file1 为其关联关系的前驱，而 file(n)为其后继。

由文件关联概率算法计算得到的小文件关联集合只限于两个小文件之间，如(file1,file2)和(file2,file3)。如果只是将该关联集合内的小文件进行合并，会导致小文件多次存储而浪费存储空间。因此，在小文件关联合并前，需要将小文件关联集合中的关联关系进行合并处理，其合并过程如下：

(1) 在小文件关联集合 file_set 中选取一个在关联集合中前驱出现频率最高的关系关系，然后从这些关联关系中选择一个概率最大的小文件作为首个合并对象。

(2) 然后按依次将剩余关联集合 file_set 中前驱与该关联关系的后继相同的小文件进行合并，如果有多个小文件满足关系，那么就选择其中概率最大的关联关系的进行关联合并，每完成一次关联关系合并就将其从 file_set 中移除。

(3) 再次执行过程(2)直到没有可以合并的关联关系时，结束此次合并，然后返回过程(1)进行第二次关联关系合并。关联关系合并算法的伪代码如表 3-2 所示，其中输入为小文件关联集合，输出为处理后的关联集合。

表 3-2 关联关系合并算法

输入：文件关联集合

输出：处理后的关联集合

```
1.  merge(file,correlation){   // 文件关联关系合并
2.  if (correlation==NULL):
3.      选择首个合并对象 firstfile
4.      nextfile=firstfile.next
5.      if (firstfile.size+nextfile.size)<=4M:
6.          correlation_set.append(nextfile)      // 关联合并
7.          remove nextfile     //从关联集合中移除 nextfile
8.      else:
9.          firstfile=correlation
10.     end if
11.     if (nextfile=null):
12.  // 直到找不到适合的后继文件 nextfile，开始下一次关联合并
13.         correlation_set.append(nextfile)
14.         remove first file from file_set
15.         merge(file,correlation)   // 下一次关联合并
16.     end if
17. end if
18. return merge
19.  }   // 关联合并结束后，输出文件关联集合
```

§3.4 小文件关联合并

Ceph 集群在将上传文件提交给小文件合并模块前，首先会通过客户端检测该文件是否为小文件，若是则通过小文件的关联概率统计算法得到小文件的关联集合；然后通过关联关系合并算法对关联集合进行合并处理，得到多个小文件间的关联集合；最后根据小文件合并算法，将相关小文件进行合并后再存储。

在小文件合并过程中，为避免合并文件块的最后一个小文件被分别存储到不同的数据节点上，用户读取小文件时，需要额外的去访问其他数据节点，会大大增加文件的访问时间。因此，本文将 Ceph 集群合并文件块大小的阈值设置为 4M。当文件块小于等于 4M 时，遍历关联文件集合将每一个关联关系中的小文件进行合并，然后存储到 Ceph 集群中。当新合并文件与合并文件块的大小大于阈值 4M，原合并文件块上传至 Ceph 集群，并重新申请一个合并文件，其中小文件的合并流程如图

3.5 所示。

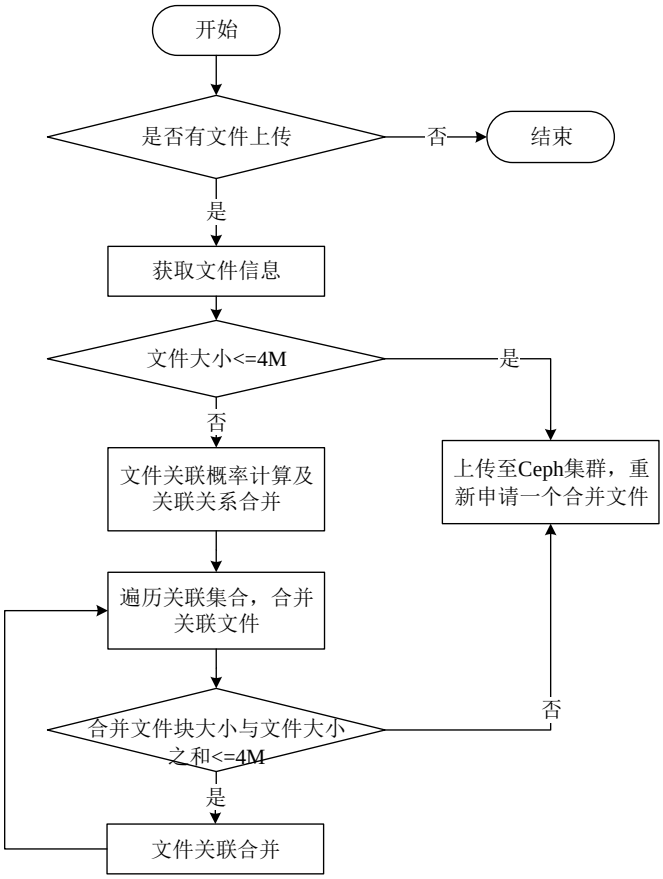

图 3.5 小文件关联合并流程图

合并文件块的结构包括合并文件块的标识符、合并文件块的名字和合并文件块的当前大小。其中小文件合并算法如表 3-3 所示。

表 3-3 小文件合并算法

输入：文件关联关系集合

输出：合并后的文件块上传到 Ceph 集群

```
1.  mergefile(correlation_set ){
2.      for file in correlation_set:
3.          if (merge1.size+file.size)<=4M:
4.              mergefile.append(nextfile)
5.          else:
6.              upload mergefile to Ceph
7.              重新创建一个大文件块 merge i (i=2,3,4….)
8.          end if
9.      end for
10. }
```

§3.5 B+树索引的设计与实现

由 3.2 小节和 3.3 小节的介绍可知，论文已经给出了海量小文件的预处理方法以及文件合并方案，本节与上述小文件合并方法紧密结合，以达到进一步优化小文件存储性能的目的。根据上述小文件合并方法可知，若干不同大小的小文件在合并成大文件后，会被上传到 Ceph 集群的不同数据节点上。虽然小文件合并可以减少文件数量，提高文件存储效率，但在文件读取时，需要遍历所有的文件，查找效率比较低。对此，本文在小文件关联合并的基础上引入索引机制，该机制将小文件与合并文件块间的映射关系生成索引文件并存储在客户端。

§3.5.1 B+树索引技术

1970 年，R.Bayer 等人提出一种普遍适用于外部查找的平衡多叉树，称其为 B[53]树。而 B+[54][55]树是为满足文件系统的需求而产生的一种变型树，一棵完整的 B+树，主要包括根节点、内节点和叶子节点，图 3.6 为 4 阶的 B+树的示意图。

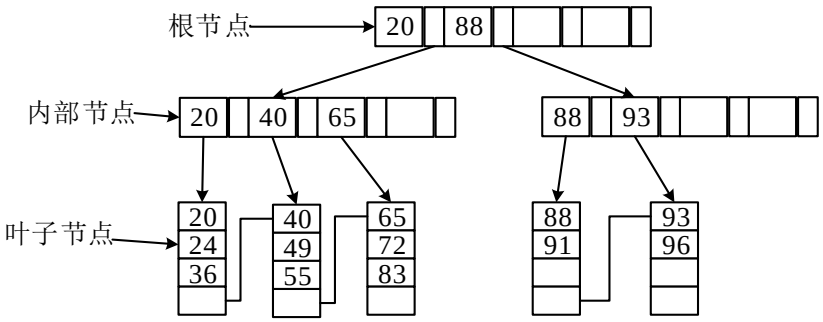

图 3.6 B+树的结构图

系统采用 B+树的这种数据结构与 B 树相比具有如下优点：

(1) B 树只有一种随机检索方法，而 B+树有两种查找方法，其不仅可以随机检索也可以顺序检索。随机检索是一种从根节点开始查找的随机查找方法；顺序检索是从最小关键字的叶子节点开始的顺序查找方法。

(2) B+树中所有关键字均放置在叶子节点中，而非叶子节点则是系统数据的索引部分。

(3) B+树比 B 树的查找效率稳定且 I/O 读写代价较低，系统文件利用 B+树索引进行数据查询时，由于其非终结点并不是直接指向文件内容的结点，仅仅是叶子结点中关键字的索引，因此索引文件中任何关键字的查找都必须走一条从根结点至叶子结点的路，因此所有关键字查询的路径长度是一样的，故每一个系统文件的查询效率都是相当的[54]。B+树的内部节点中没有像 B 树那样指向关键字的指针，因此它的内部节点相对比较小，磁盘每次可写入的关键字也比较多，所以 I/O 的读写次数也减少了很多，提高了 B+树的检索效率。

§3.5.2 索引文件的实现

在小文件访问时，目前有两种方法可以提高小文件的访问效率。Zou Zhenyu[56]等人采用文件预取机制将文件预取至缓存中，减少用户与集群的直接交互，从而提高文件的访问效率；Cheng Wenjuan[57]等人提出一种索引机制，在文件合并时生成相应的索引文件，减少小文件的查找时间，可以提高小文件的访问效率。

为了提高用户查找小文件时的查找效率，减少用户的查找时间，在小文件合并时为各个小文件建立了相应的索引文件，通过索引文件 Ceph 集群可以快速找到小文件在合并后大文件中的具体位置信息，并迅速获取小文件的相关属性。在建立索引文件时，首先为合并文件块生成索引文件，再为合并文件中的各个小文件建立局部索引。

索引文件结构由<key,value>组成，具体格式为<key，[file_offset ,file_length]>，其中使用 B+树索引中的 key 值来保存小文件的文件名及 ID 等小文件的关键信息，对应的 value 用于记录小文件在合并后大文件的起始位置偏移量 file_offset 及小文件的大小 file_length。小文件的索引结构如图 3.7 所示。

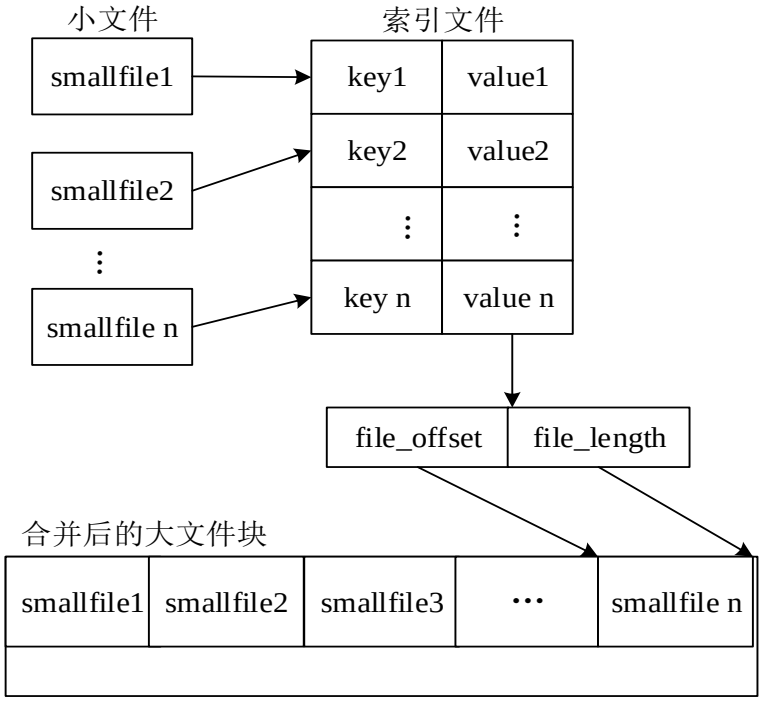

图 3.7 索引文件结构图

用户在查找文件时，首先根据要查找的小文件名在索引文件依次查找，若索引记录中找到与之相符的 key 值，说明查找成功，系统就可以根据查找到的合并后大文件的名字来查找小文件在大文件中的偏移量及文件大小，然后根据小文件在合并后大文件中的具体信息读出相应的文件内容。若索引记录没有找到与之相符的 key 值，则说明文件查找失败。

§3.6 小文件预读取与缓存机制

小文件的关联合并只能减少小文件的数量，从而减少小文件元数据的数量，但随着小文件存储数量的增加，合并文件也随之增加，小文件的访问效率也会明显降低。因此，为了进一步提高小文件的访问效率，本文设计实现了根据小文件间的关联关系进行小文件的预读取及缓存机制。

§3.6.1 小文件预读取算法

上述介绍了小文件存储时采用的小文件关联合并算法将关联的小文件进行合并后再存储的优化方法，为了证明合并之后各个文件之间的关联关系是最优解，使用文件块内的利用率和块内相关率来衡量小文件合并之后的相关性。当用户访问小文件时，不仅需要考虑文件块中小文件间的相关性，还需要考虑小文件预取时间、用户等待时间等因素，其相关定义如下。

(1) 文件块的利用率 Φ_{URFB} (Utilization Ratio of File Block)：表示一段时间内，用户访问某合并文件块中不同小文件的个数与该文件块中所有小文件总数的比值，其公式如 3-4 所示。

$$\Phi_{\mathrm{URFB}} = \frac{d\_f}{sum\_f} \tag{3-4}$$

其中，d_f 为一段时间内访问该文件块中不同小文件的个数，sum_f 为该文件块中小文件的总数。

(2) 文件块的相关率 Ψ_{CRFB} (Correlation Ratio of File Block)：用来衡量一个文件块中的小文件是否能同时传输的指标。

$$\Psi_{\mathrm{CRFB}} = \Upsilon^{i-1} * \left(1 + \frac{n * d\_f}{sum\_f^{2}}\right) \tag{3-5}$$

$$\Upsilon^{i} = \begin{cases} 1 & ,\Psi \geq 1 \\ \Psi & ,\Psi < 1 \end{cases} \tag{3-6}$$

其中，n 为该时间段内访问该文件块的总次数，Υ 表示当前文件块的相关率，对于每一个文件块，Υ 的初始值均设为 0.1，$i = 1,2,3\cdots$，由公式 3-5 和 3-6 可以看出文件块的相关率是一个不断随时间变化的过程。

若用户在系统不存在预读取时连续读取 2 个文件需要的时间为 $T_1 = 2T_{Ceph}$，有预取时读取 2 个文件的时间为 $T_2 = T_{Ceph} + T_{pre} + T_{cache}$。因为用户从缓存中读取文件时，主要是访问内存时间，故可以忽略不计。因此当 $T_2 < T_1$ 时，可以对文件进行预取存储至缓存中，并将文件关联概率 P 带入可得公式 3-7：

$$T_{Ceph} + \left(T_{pre} + T_{cache}\right) * P < \left(1 - P\right) * 2T_{Ceph} \tag{3-7}$$

其中，T_{Ceph} 表示 Ceph 集群在不存在预读取优化时，客户端从接收到用户请求到 Ceph 集群返回请求文件的时间，T_{pre} 表示 Ceph 集群预读取一个小文件至缓存中的时

间，T_{cache} 是用户从缓存中读取请求小文件的时间。

在系统预取小文件时，不仅需要考虑文件块的利用率和相关率等影响因素，而且需要考虑用户最大等待时间等。故当文件块的利用率和相关率均大于阈值 F 时，说明同时传输该文件块时无效数据较少，因此可以把该文件块中的所有文件预读取出来并存储至缓存；否则在用户最大等待时间 T_w 内，小文件的最大预取个数 *max_num* 可通过公式 3-8 计算得到。

$$\max\_num = \text{math. floor}\left(\frac{T_w - T_{Ceph}}{T_{pre}}\right) \tag{3-8}$$

小文件预取算法的伪代码如表 3-4 所示。其中，输入为要查找的小文件，输出为要查找的小文件及其相关联的文件集合。

表 3-4 文件预取算法

输入：要查找的文件 file

输出：要查找的文件及相关文件 files

```
1.  prefetch(file){
2.      if  Φ_URFB  and  Ψ_CRFB  > F:
3.          files.append(mergefile)
4.      else:
5.          max_num=math.floor(T_w-T_Ceph)/T_pre
6.          if (i >max_num):   //   i 为文件预取个数
7.            break
8.          else:
9.             for file in mergefile :
10.               P=probility(file)   //  文件关联概率
11.               if (T_Ceph+(T_pre+T_cache)*P<2T_Ceph*(1-P)):
12.               files.append(file)
13.               i=i+1
14.               end if
15.            end for
16.         end if
17.     end if
18.     return files
19.  }
```

§3.6.2 小文件缓存机制

客户端在读取小文件过程中，Ceph 集群根据小文件间的相关性把与其相关的多

个小文件读取出来并存储至缓存中。当用户从缓存中读取文件时，直接利用文件名进行索引同时读取出相关的文件内容，可以减少用户与集群的交互，从而减少了用户的访问时间以提高小文件的读取效率。

小文件缓存过程可以描述为：当用户客户端首次从 Ceph 集群中访问一个小文件时，客户端向 Ceph 系统发出读请求，系统通过索引文件得到小文件在合并文件块中的位置信息，根据索引信息 Ceph 读出相应的文件内容并返回给客户端，同时读取与其相关联的小文件并存储到缓存中。用户在访问这些小文件时，可以直接从缓存中读取，提高了小文件的读取效率。缓存机制的设计与实现在第四章有详细的介绍。

当用户访问文件时，首先判断该文件是否在缓存中，若存在缓存中，则直接读出并返回请求文件给用户，同时更新缓存信息；若不在缓存中，则向集群发出读文件请求。Ceph 集群接收请求信息，然后根据小文件的索引信息，从集群中读取请求文件及与其相关的小文件并存储在缓存中。

§3.7 实验与分析

§3.7.1 实验环境

一、Ceph 集群环境

本文的实验都是基于实际环境 Ceph 集群实现的，整个 Ceph 集群共使用了 5 台机器，1 台监控节点，3 台存储节点，1 台客户端节点，其中 1 台存储节点包含 2 个 OSD 节点。每台机器的操作系统都是 Ubuntu 14.04，Ceph 版本为 9.2.1，OSD 系统为 XFS，Python 版本为 3.4，Ceph 副本数为 3 副本，Ceph 文件块的大小设置为 4M。

Ceph 集群的物理环境的硬件配置如表 3-5 所示：

表 3-5 集群硬件配置

主机名称	硬件配置	IP 地址
Client	处理器：Intel(R) E5-2620 v2 @2.10GHz 内存：8 GB　网卡：1 GB	172.25.1.248
MON	处理器：Intel(R) E5-2620 v2 @2.10GHz 内存：8 GB　网卡：1 GB	172.25.1.110
OSD1/2	处理器：Intel(R) E5-2620 v2 @2.10GHz 内存：8 GB　网卡：1 GB	172.25.1.111
OSD3/4	处理器：Intel(R) E5-2620 v2 @2.10GHz 内存：8 GB　网卡：1 GB	172.25.1.112
OSD5/6	处理器：Intel(R) E5-2620 v2 @2.10GHz 内存：8 GB　网卡：1 GB	172.25.1.113

二、RBD 块存储接口的介绍及部署

本文在搭建完 Ceph 集群后，在 lxxclient 客户端上安装 RBD 块设备的存储接口，通过 RBD 块存储接口实现海量小文件的批量上传和批量下载功能，在节点 lxxclient 上安装 RBD 块设备的详细步骤如下：

1、在节点 lxxclient 安装 Ceph 过程

(1) 首先利用 lsb release –a 和 uname –r 来检查虚拟机操作系统的内核版本是否适合 RBD 块设备的挂载。

(2) 在 lxxclient 节点使用 apt-get 安装 Ceph。

```
lxxclient@ubuntu:~$ apt-get install ceph
```

(3) 将 Ceph 集群监控节点的配置文件 Ceph.conf 拷贝到 lxxclient 的 Ceph 的配置文件中。

2、RBD 块设备配置的具体过程

(1) 节点 lxxclient 上创建一个大小为 1024M 的块设备 cephrbd1。

```
lxxclient@ubuntu:~$ rbd create --size 1024 cephrbd1
```

(2) 在节点 lxxclient 上将 cephrbd1 映射为块设备。

```
lxxclient@ubuntu:~$ rbd map cephrbd1
```

(3) 在节点 lxxclient 上创建文件系统后，cephrbd1 块设备就可以使用了。

```
lxxclient@ubuntu:~$ sudo mkfs.ext4 -m0 /dev/rbd/cephrbd1
```

(4) 最后在节点 lxxclient 上挂载该文件系统。

```
lxxclient@ubuntu:~$ sudo mkdir /mnt/ceph-block-device
```

```
lxxclient@ubuntu:~$ sudo mount /dev/rbd/rbd/cephrbd1
 /mnt/ceph-block-device
```

在 lxxclient 节点上实现 RBD 块设备的安装后，通过命令 rbd ls 可以罗列出所有块设备的映像。

```
lxxclient@ubuntu:~$ rbd ls
bd2
cephrbd1
cephrbd2
test
```

§3.7.2 实验结果与分析

(1) 海量小文件写性能测试

该测试的性能指标是：文件平均存储时间=文件存储时间/小文件总数。具体实验过程是：分别对采用不同方案的系统上传 500，1000，1500，2000，2500，3000，4000，6000 个小文件，并比较各系统的平均存储时间。其中小文件的平均存储时间的测试结果如图 3.8 所示。

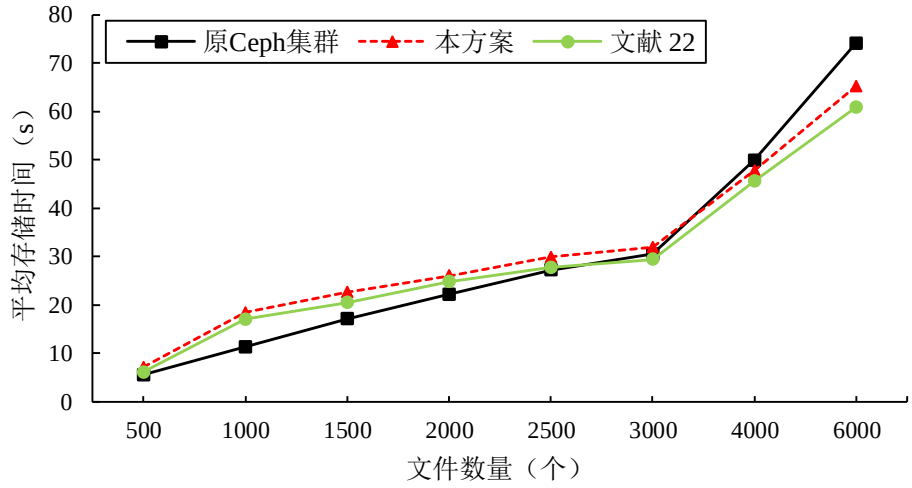

图 3.8 文件平均存储时间

由图 3.8 可以看出，在存储文件数量较少时，与原 Ceph 集群相比，本方案的平均存储时间较高，主要是因为在小文件合并存储之前需要统计分析小文件间的关联关系，增加了小文件的平均存储时间；但随着小文件存储数量的逐渐增加，原 Ceph 集群的平均存储时间明显上升，而文献[22]和本方案的平均存储时间的增长较为缓慢，是因为文献[22]和本优化方案都以不同的合并方法对小文件进行了合并处理，通过文件合并大大减少了小文件的数量，因此小文件存储时明显减少了申请存储空间的次数，从而节省了存储时间。

(2) 文件块的利用率及相关率测试

该实验的性能指标是：合并文件块的利用率和相关率。由第三章 3.6.1 小节中公式 3-4、3-5 及 3-6 可知，合并文件块的利用率和相关率都是基于一个时间段内用户的访问次数进行计算的。因此，本实验针对不同时间间隔对文件块的利用率及相关率的影响进行了测试。测试结果如图 3.9、图 3.10、图 3.11 所示。

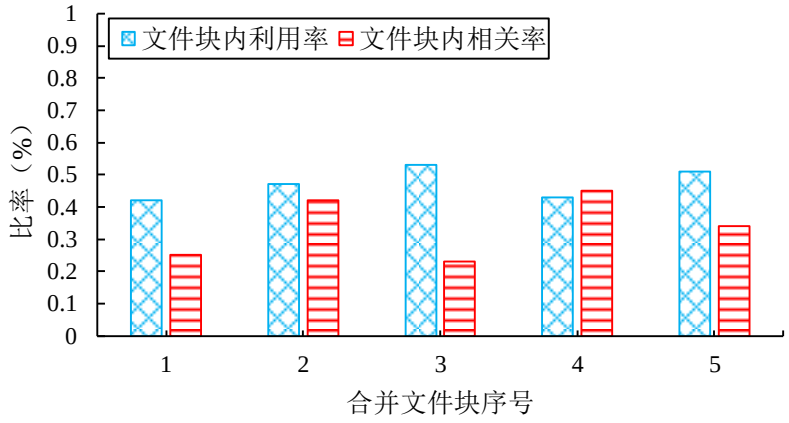

图 3.9 文件块的相关率及利用率（时间段:5min）

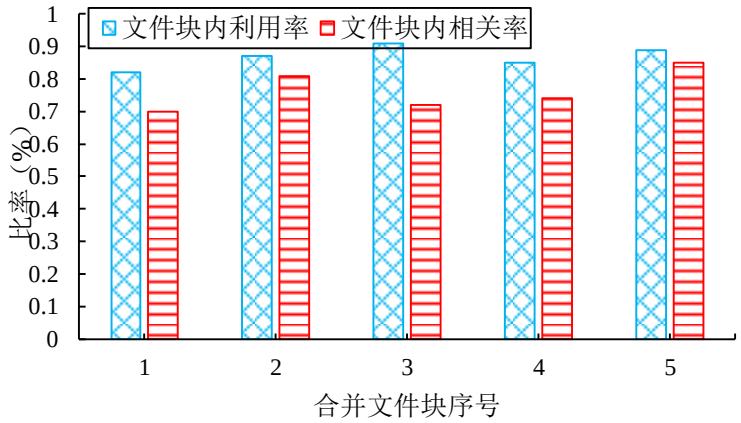

图 3.10 文件块的相关率及利用率（时间段:15min）

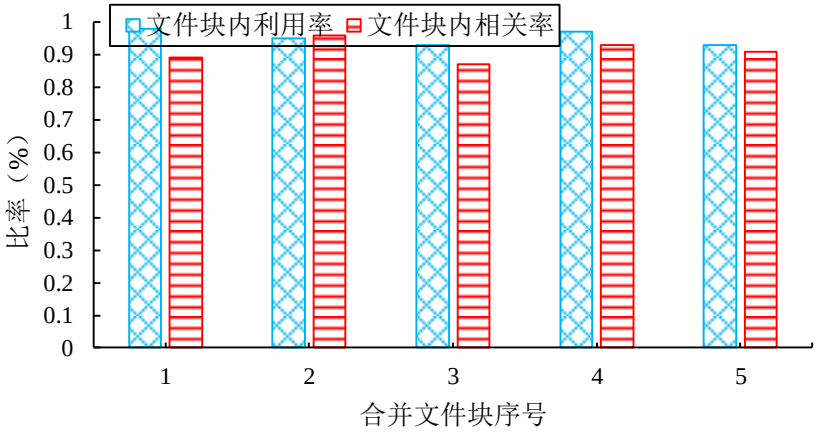

图 3.11 文件块的相关率及利用率（时间段:35min）

由图 3.9 可以看出，当时间间隔取值较小时，文件块的利用率和相关率之间的波动比较大。但是若时间间隔取值较大时，文件块的利用率和相关率都接近 100%，无法准确反映文件块中各个小文件间的相关性，如图 3.11 所示。通过大量实验发现若将时间间隔取值为 15 分钟，可以较好的反映文件块中各个小文件间的相关性，结果如图 3.10 所示，同时将文件块间利用率和相关率的关联阈值设为 0.7。如果文件

块的利用率和相关率均大于给定关联阈值，就将该文件块内的小文件一起预取出来并存储至缓存中；否则，在用户的最大等待时间内，预取 max_num 个小文件并存储至缓存中，其中 max_num 由公式 3-8 计算得出。

(3) 海量小文件读性能测试

本实验的性能指标是：平均读取时间（平均读取时间=文件总读取时间/文件总数）。该测试模块的主要作用是测试在读取同样数量小文件的情况下，采用不同方案时各个系统的平均读取时间。具体实验过程：在文件块的利用率和相关率实验的基础上，各系统分别读取 500，1000，1500，2000，2500，3000，4000，6000 个小文件时的平均读取时间。其文件平均读取时间测试结果如图 3.12 所示。

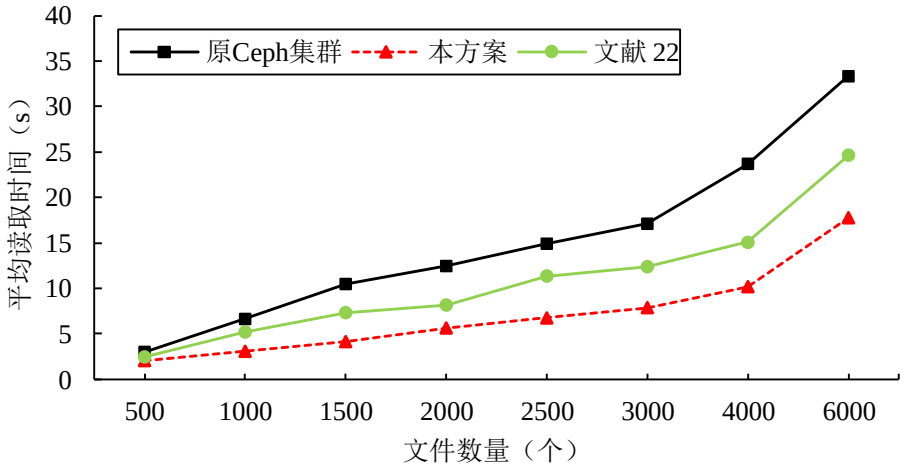

图 3.12 文件平均读取时间

由图 3.12 可以看出，随着小文件数目的不断增加，文献[22]和本文优化方案都比原 Ceph 集群节省读取时间。因为文献[22]将同一时间段上传的小文件进行合并后再存储，在读取文件时，结合 LRU 算法建立的预取机制，加快了小文件的读取速度。本文则根据小文件间的关联关系将小文件合并成大文件后再存储到系统中，在读取文件时，利用小文件预读取方法可以实现用户在读取某一个小文件的过程中，将与其相关联的多个小文件一起预读取到缓存中。这就意味着当用户从缓存中读取小文件时，可以减少用户与集群的交互，从而减少了用户的访问时间。

本文优化方案与文献[22]相比也更节省时间，主要是因为优化后 Ceph 集群中的小文件是通过文件关联合并后存储的，在读取文件时，可以将多个关联文件预取到缓存中，减少了文件读取时间。而文献[22]也有小文件合并存储过程，但其并未考虑小文件间的相关性，因此，相比本优化方案其读取效率较低。

§3.8 本章小结

本章节首先介绍了 Ceph 分布式存储系统在存储海量小文件时所出现的问题，

然后针对这些问题提出了一种基于关联关系的小文件合并方法及相应的小文件预读取方法，并设计了相关测试实验。 通过大量实验数据得到系统在存储和读取海量小文件的平均存储时间和平均读取时间的曲线图。实验表明，本章所提出的改进方法虽然在存储数量较少时，对于文件存储效率的提升并不明显，但小文件的读取效率有明显的提升。

第四章 小文件预取缓存算法的改进与实现

由第三章的介绍可知，在读取小文件时，系统利用小文件预读取算法，将与之相关联的小文件预读取出来并存储至缓存中，以提高小文件的访问效率。但随着小文件存储数量及用户请求数量的不断增大，缓存文件的数量也随之增长，这样将出现有些小文件长时间未被访问而浪费缓存空间的问题。针对这一问题，本章节提出一种基于权重因子的 LRU 改进算法，然后在改进算法的基础上设计并实现了二级缓存优化机制。实验表明，当面对海量小文件的预读取时，利用该方法可以提高缓存文件的命中率及系统的整体性能。

§4.1 小文件预取缓存设计

在本节的缓存结构设计中，缓存中的文件信息与第三章 3.5.2 小节的索引结构相似，不同的是缓存结构是直接将小文件的内容存储在缓存中，可以通过文件名直接读取小文件相关内容。其中小文件缓存信息主要包括小文件名、小文件大小、小文件内容以及小文件访问时间等，缓存中的缓存数据结构如表 4-1 所示。

表 4-1 缓存数据结构

缓存的数据结构
filename：小文件名
filesize：小文件大小
createtime：小文件创建时间
filecontent：小文件内容
access_time：小文件访问时间

当用户访问文件时，首先判断该文件是否存储在缓存中，若在缓存中，就直接从缓存中读取小文件的具体内容，同时返回给用户。这样就减少了用户与底层 Ceph 系统的交互，可以减少用户的访问时间。若该文件不在缓存中，就说明该文件尚未被预读取出来，然后向 Ceph 系统发出读请求。Ceph 系统接收请求，并通过索引文件查询该文件在合并文件块中的具体位置信息，Ceph 根据索引信息读取出请求小文件并返回给用户，同时根据小文件间的相关性将与其相关的小文件预读取至缓存中。

§4.1.1 缓存容量

分布式存储系统在处理海量小文件时，通过优化小文件处理算法所带来的效果远不如系统利用缓存带来的效果。因此，为了提高分布式存储系统的性能，针对不同应用场景设计实现的文件存储系统则会采用不同的缓存替换方法。

在确定分布式存储系统采用缓存机制可以提高系统的整体性能后，如何确定系

统的缓存容量已经成为评估系统存储性能的重要问题。通过分析存储系统的数据特征，可以确定存储系统是读请求比较集中，还是写请求比较集中。若存储系统是写请求比较集中时，可以在系统文件写入时设置一个写缓存；若该系统是读请求比较集中时，可以在系统中设置一个读缓存。系统也可以通过再次访问缓存对象文件的大小，确定系统所需要的缓存容量大小，从而可以避免因缓存文件太小而浪费缓存空间。

§4.1.2 缓存预取策略

缓存预取[58]是指用户在访问该文件之前就预先将该文件预先下载存储至缓存中，这样一方面能够较好的提升缓存对象的命中率，另一方面可以减少用户与系统的交互，从而减少用户访问时间。如何选择预取文件，如何选择缓存预取技术也是众多企业和学者研究的重点。目前最常用的缓存预取策略分别是：基于用户访问概率的预取策略、基于数据挖掘的预取策略以及基于神经网络的预取策略。

本文给出的缓存预取机制采用的是基于用户访问概率的，该机制是通过判断文件块间的相关性及利用率是否均大于等于给定阈值，若均大于阈值，则该文件块内的相关小文件可以预取出来并存储至缓存中，否则就根据小文件间的关联概率，在用户的最大等待时间内，将与其相关的小文件预取出来并存储至缓存中。

§4.2 LRU 算法的改进

LRU 算法是目前最常用的一种缓存替换算法，该算法是根据缓存文件的访问时间间隔原理设计实现的。当缓存容量不足时，将最不常被访问的文件替换出缓存。本文在设计缓存机制时，采用的是 LRU 算法来实现系统文件的预取及缓存。但在实验时发现，在面对海量文件读取及缓存时，随着文件存储数量及用户请求数量的不断增大，缓存中文件的数量也不断增长，这样将出现有些小文件长时间未被访问而浪费缓存空间的问题，导致缓存文件命中率降低。对此，论文给出一种基于访问频率及用户访问时间的 LRU 改进算法 LRU_W。该算法在文件写入缓存时，首先利用公式 4-1 为该缓存文件计算出其权重因子 R_w，并将其放置在缓存队列的头部。

$$R_w = R_{w1} * e^{-(N_t - N_r)*t} \tag{4-1}$$

式 4-1 中，R_{w1} 为文件当前的权重，N_t 为缓存中缓存容量的最大值，N_r 为某小文件在缓存中的用户访问次数。

在不考虑权重因子随着时间的增长而衰减的情况时，可以通过用户的访问次数来计算缓存文件的权重，因此在考虑到权重衰减后，当缓存文件再次被访问时，需要重新计算文件权重。若文件上一次被访问时的权重为 R_{w1}，经过一段时间 t 后，该文件再次被访问的时，通过公式 4-2 重新计算缓存文件的权重。

$$R_w = R_{w1} * e^{-(N_t - N_r)*t} + 1 \quad (4\text{-}2)$$

由公式 4-2 可以看出，缓存中小文件的权重因子会随着访问时间的增长而随之衰减。当权重小于阈值时，该文件会从缓存中移除，避免了某些文件因长时间不被访问而浪费缓存空间的情况。

§4.3 LRU_W 缓存算法的实现

LRU_W 算法是基于 LRU 算法的改进，该算法充分考虑了文件的访问时间及访问频率因素，动态计算出每一个缓存文件的权重因子。通过公式 4-2 计算缓存文件的权重，并根据缓存文件的权重来确定缓存文件的优先级。当缓存容量不足时，将权重最小的文件从缓存中移除。

§4.3.1 LRU_W 算法请求处理流程

LRU_W 缓存算法的请求处理流程如图 4.1 所示。

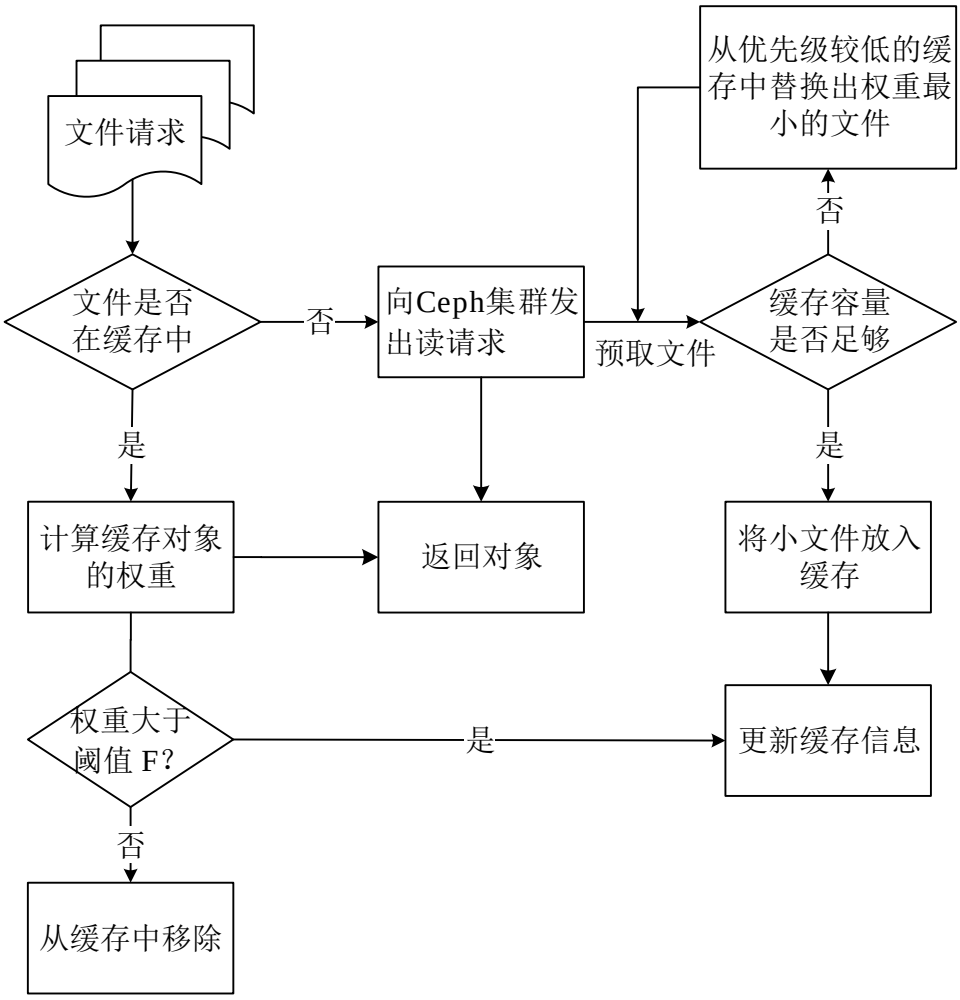

图 4.1 LRU_W 算法的文件处理流程

由图 4.1 可以看出，当用户发出访问请求时，系统需要首先缓存中是否有该文件，若存在缓存中，则从缓存中读出相应的文件内容并返回请求文件给用户，并重

新计算缓存对象的权重，若其权重值低于给定阈值，则从缓存中移除该文件，否则就更新缓存信息。若该文件不在缓存中，就说明文件尚未被读取出来，然后系统向Ceph 集群发出读请求，从 Ceph 集群中读取请求文件及与其相关的小文件至缓存中。当缓存容量不足时，优先将缓存中权重比较低的缓存对象从缓存中移除。

§4.3.2 二级缓存结构的实现

本文在 LRU_W 算法的基础上设计实现了一个二级缓存策略，该策略根据文件对应的权重因子不同，将缓存文件分别对应不同的优先级，其详细实现过程如图 4.2 所示。

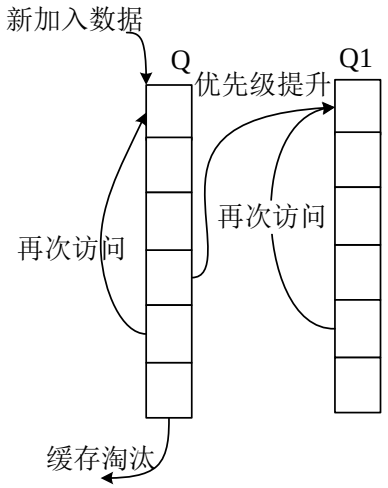

图 4.2 二级缓存数据结构

如图 4.2 所示，Q，Q1 分别代表二级缓存数据结构的一级缓存和二级缓存，其中二级缓存中的文件优先级相比一级缓存中的文件较高。该缓存策略的主要实现过程是：

1) 新加入缓存的数据放入 Q 中的首部，记录访问时间和访问次数，并给出相应的初始权重；

2) 随着时间的不断增加，文件权重的不断改变，当文件的权重大于给定的阈值时，就将优先级比较高的文件放入二级缓存 Q1 中。用户在访问文件时，首先从 Q1 中检查该文件是否存在，若不存在，再从 Q 中检查该文件是否存在。这样可以保证权重较高的文件优先级较高。

3) 随着海量小文件的读取，缓存容量不足时，首先从优先级较低的一级缓存 Q 中删除权重因子较低的文件；

4) 由于缓存文件的权重因子会随着时间的增长而衰减，当其权重降低到低于阈值时，该文件将会缓存中淘汰，避免文件长时间未被访问而浪费缓存空间。

基于改进的 LRU_W 算法实现的二级缓存的核心代码如表 4-2 所示。

表 4-2 二级缓存策略实现代码

输入：若干文件的请求队列

输出：用户访问请求的文件

```
1.  cache(files){
2.      Q<small file to cache>   // 一级缓存
3.      Q1<small file to cache>   // 二级缓存
4.      文件 file 访问请求到达
5.      if file in Q1:
6.          Nr=Nr+1
7.          Rw=Rw1*e^-(Nt-Nr)*t +1   //当缓存对象再次被访问时的权重因子
8.          返回请求文件
9.      elif file in Q:
10.         Nr=Nr+1
11.         Rw=Rw1*e^-(Nt-Nr)*t+1
12.         返回请求文件
13.     else :
14.         从 Ceph 集群中读取 file
15.         Rw1=2   //初始缓存文件的权值
16.         Nr=1
17.     end if
18.     Rw=Rw1*e^-(Nt-Nr)*t   // 经过时间 t 后，缓存文件的权重因子
19.     if Rw>R:
20.         move file to Q1
21.     else:
22.         remove file from Q
23.     end if
24. }
```

§4.4 实验与分析

§4.4.1 实验环境

为了测试改进后 LRU_W 缓存算法的命中率，测试二级缓存策略对海量小文件存储系统读取性能的影响，搭建了以下测试环境：整个 Ceph 集群共使用了 5 台机器，1 台监控节点，3 台存储节点，1 台客户端节点，其中 1 台存储节点包含 2 个 OSD 节点。其中，Ceph 副本数为 3 副本，Ceph 文件块的大小设置为 4M。软硬件

测试环境如表 4-3 所示。

表 4-3 软硬件测试环境

软硬件信息	
处理器	Intel(R) I3-3230 v2 @2.60GHz
内存	8 GB
操作系统	Ubuntu 14.04
Ceph 版本	9.2.1
OSD 系统	XFS

§4.4.2 实验结果与分析

传统的 LRU 缓存替换算法仅仅考虑了缓存文件的访问时间因素，但在面对海量小文件的存储和读取时，LRU 缓存算法由于时间局限性而导致缓存文件的命中率较低。为此，本文给出了改进的 LRU_W 缓存替换算法，该算法综合考虑了缓存文件访问频率和访问时间因素，本文将 LRU_W 算法应用的系统的缓存机制中，适用于海量小文件的访问特性。实验对缓存优化机制的测试主要分为：LRU_W 改进算法的相对命中率、绝对命中率及缓存优化机制应用到系统时文件的平均读取时间测试三部分。

(1) LRU_W 算法的相对命中率

测试实验在本地缓存容量与数据总量的比值分别为 10%、20%、30%、40%、50%、55%、60%、65%时，对 LFU、LRU、LRU_W 三种算法的相对命中率进行了测试。其测试结果如图 4.3 所示。

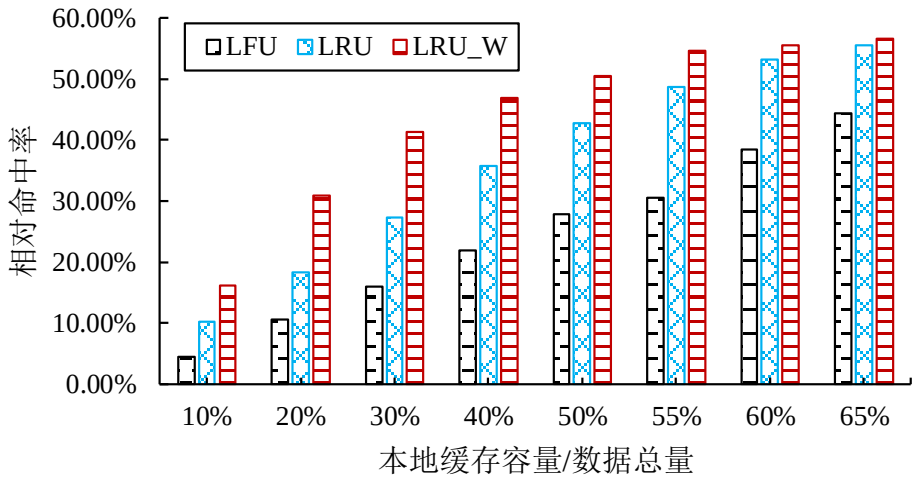

图 4.3 LRU_W、LRU、LFU 三种缓存算法的相对命中率

由图 4.3 可以看出，随着本地缓存容量与数据总量比值的不断增大，三种缓存替换算法的相对命中率也随之不断提高，但本章节给出的缓存算法的相对命中率在本地缓存容量与数据总量比值较小时一直都是最优的。

(2) LRU_W 算法的绝对命中率

该测试主要是通过不断改变本地缓存容量与数据总量的比值对缓存文件命中率的影响。具体实验过程是：本地缓存容量与数据总量的比值分别为 10%、20%、30%、40%、50%、55%、60%、65%，对 LFU、LRU、LRU_W 三种算法的绝对命中率进行测试。其测试结果如图 4.4 所示。

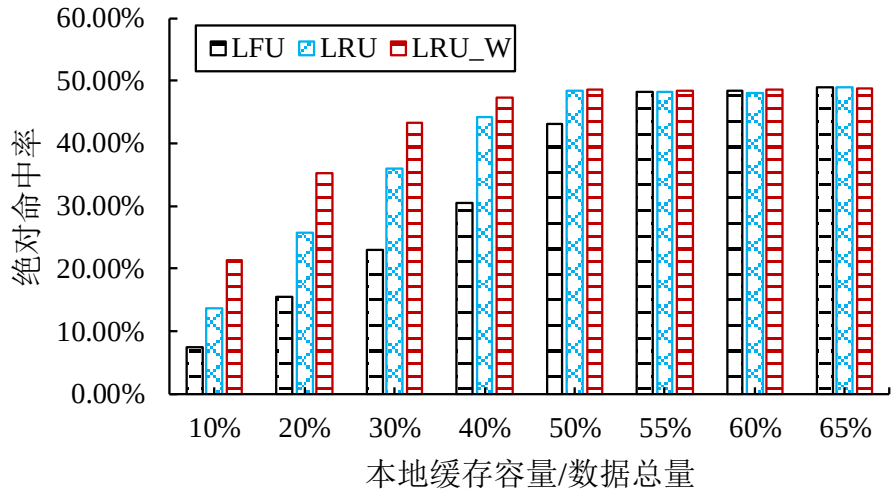

图 4.4 LRU_W、LRU、LFU 三种缓存算法的绝对命中率

由图 4.4 可以看出，随着本地缓存容量与数据总量比值的不断增大，LRU_W、LRU、LFU 三种算法的绝对命中率也随之不断提高，但明显可以看出 LRU_W 算法相比其他两种算法的绝对命中率也是相对较优的。当比值达到 55%时，可以看出三种缓存算法的绝对命中率几乎相同。

由图 4.3 和图 4.4 可以看出，LRU_W 算法和 LRU、LFU 缓存算法想比较，LRU_W 算法在本地缓存容量与数据总量的比值较小时性能较优。海量小文件存储系统中本地缓存容量是有限的，而系统数据是海量的， 因此本地缓存容量与数据总量的比值是较小的，而 LRU_W 算法在比值较小时，其缓存性能最优。

(3) 系统文件读取性能测试

该测试模块的主要作用是当系统中存在海量数据时，使用缓存优化机制的系统和无缓存优化系统对小文件查找效率的影响。具体实验过程如下：每一次测试都在上一次测试的基础上，增加相应的小文件请求。如首次测试读取 500 个小文件时的文件平均读取时间，其次是分别测试读取 1000、3000、6000、10000 个小文件时文件的平均读取时间。其中，缓存机制的缓存容量设置为 2000，小文件的平均读取时间测试结果如图 4.5 所示。

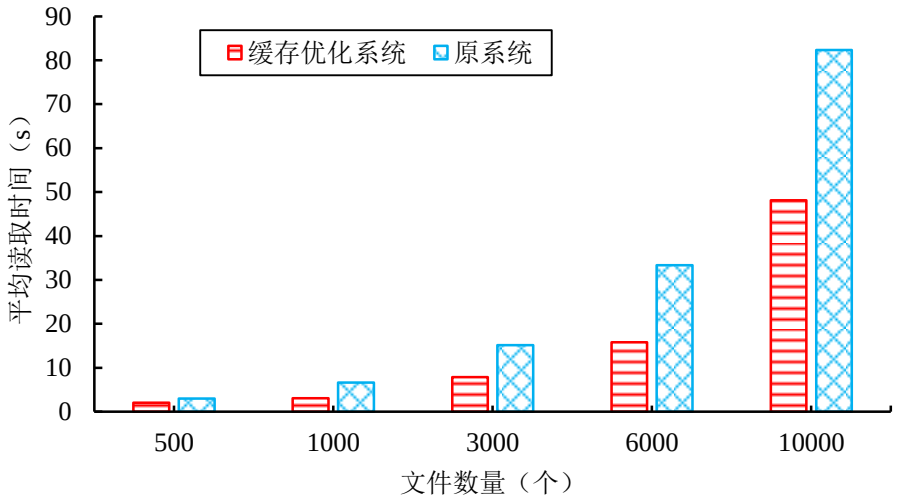

图 4.5 缓存优化系统和原系统小文件的平均读取时间

由图 4.5 可以看出，在没有缓存优化机制的系统中，随着小文件读取数量的不断增加，其平均读取时间明显增加较快；在读取同样数量的小文件时，具有缓存优化机制系统的平均读取时间显然优于原存储系统。主要是因为系统使用了缓存优化机制，可以将小文件提前存储在缓存中，用户从缓存中读取文件，可以减少用户与集群的交互，从而减少用户的读取时间，提高了系统的读取效率。同时随着读取文件数量的增加，缓存优化机制根据缓存文件的访问时间和访问频率计算出其权重衰减因子，若文件长时间未被访问，该权重随之衰减。当权重小于阈值时，该文件会从缓存中移除，可以减少缓存空间的浪费，提高缓存文件的命中率及系统的整体性能。

§4.5 本章小结

本章主要介绍缓存机制的设计，并针对 LRU 算法的时间局限性，提出了一种基于权重因子的 LRU 改进算法 LRU_W。该算法在 LRU 算法的基础上加入了文件访问频率及文件访问时间因素，计算出缓存文件的权重因子，然后由缓存文件的权重来判定小文件的优先级，决定缓存文件的淘汰顺序，设计了相应的测试实验。实验证明了本章提出的改进算法可以较好的提高缓存文件绝对缓存命中率及相对缓存命中率。此外，将基于 LRU_W 改进算法应用到海量小文件存储系统的缓存机制中，并通过实验证明了缓存优化后系统的小文件读取效率比原系统有明显的提高。

第五章 海量小文件存储系统的设计与实现

本章在第三章的小文件的存储优化方案与第四章的缓存优化机制的基础上设计并实现了海量小文件的存储系统。主要介绍海量小文件存储系统的整体设计以及系统各个模块的具体功能。该系统基于 Ceph 集群，并利用 B/S(Browser Server，浏览器/服务器模式)结构，然后通过客户端与服务器端的交互实现海量小文件的存储和读取功能。最后设计了相关测试实验，验证了系统的文件批量上传及文件批量下载、系统存储容量查询功能。

§5.1 系统整体设计

海量小文件存储系统的设计主要包括三部分：一是系统客户端系统的设计和实现；二是小文件处理模块的设计与实现；三是 Ceph 分布式文件系统的实现。系统整体设计示意图如图 5.1 所示。

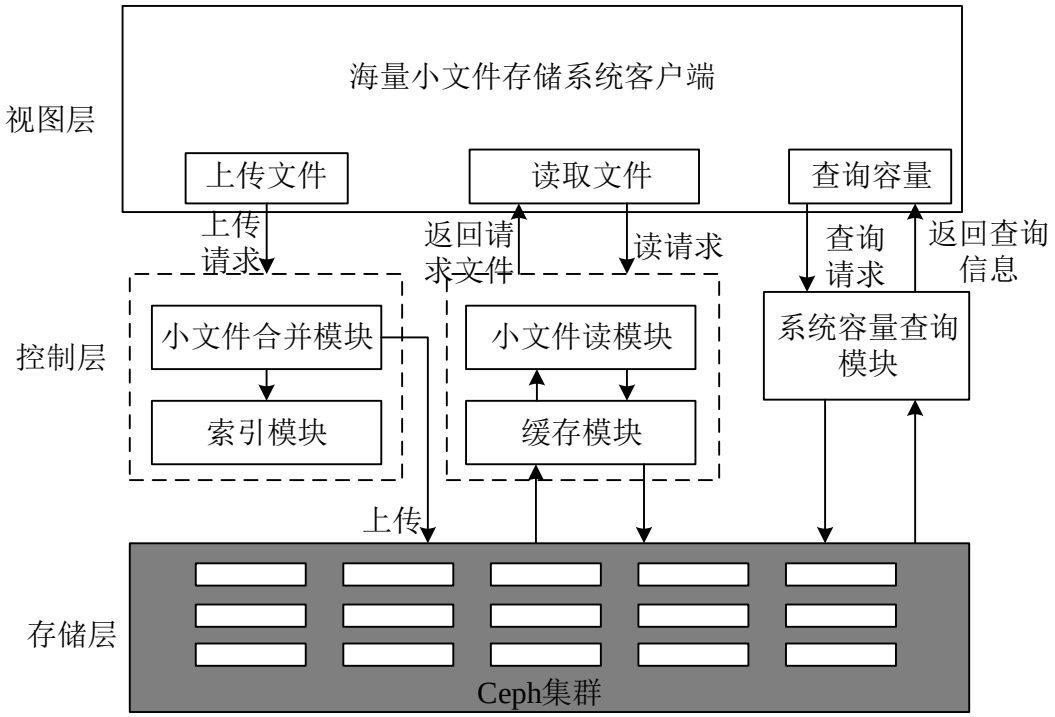

图 5.1 系统整体设计示意图

由图 5.1 可以看出，小文件存储系统主要由三大存储模块组成，下面详细介绍了各个模块的主要功能。

(1) 系统客户端：系统客户端的实现是利用 B/S[59]结构，用户可以在客户端更便捷的远程操作存储系统，实现小文件的上传、下载及系统容量查询。系统容量查询功能可以方便用户实时关注系统容量的使用及剩余情况。与传统的 Ceph 分布式文件系统相比，该系统使得用户的操作更加直观、便捷。

(2) 小文件处理模块：主要包括了小文件合并模块，索引模块，小文件预读取模块及缓存模块。各小模块分别在 5.1.1 节、5.1.2、5.1.3 和 5.1.4 节中有详细介绍。

(3) Ceph分布式文件系统：主要包括了MON监控节点，Client节点以及DataNode节点，主要功能介绍如下：

1) MON 节点的主要功能是监控和维护 Ceph 集群的 Mon map、OSD map、MDS map 等各种集群状态信息。MON 节点也可以根据 OSD 节点发送过来的心跳信息来调整 Ceph 集群的 Cluster map 信息，可以自动调整数据分布的平衡

2) Client 节点在读写数据时，首先需要连接 MON 节点，然后从 MON 节点处接收 Ceph 集群的拓扑信息，根据集群的拓扑信息，Client 节点主动计算出数据对象所需存入的主副 OSD 节点的位置，最后直接同这些 OSD 通信存放数据。通过这种将计算任务由集群主动完成转变到由 Client 节点完成的方式，可以有效减少存储集群对计算能力的要求。

3) DataNode 节点是 Ceph 集群的文件存储节点，是整个海量小文件存储系统的硬件基础部分。

海量小文件存储系统的架构采用了软件工程的分层技术，采用分层技术的优点是：

(1) 各个层次之间的耦合比较低，因此可以较好的提高代码的可管理性及可重用性。

(2) 容易实现新层对原有层替换。

(3) 整个系统以 Ceph 集群作为底层存储支撑，能够保证 Ceph 系统的兼容性。

§5.1.1 小文件合并模块

小文件合并模块是整个系统的核心部分，论文提出的小文件合并模块是在不改变原 Ceph 集群的读写操作的基础上进行的改进，在小文件上传前，先对小文件进行合并处理，其中小文件合并模块流程图如图 5.2 所示。

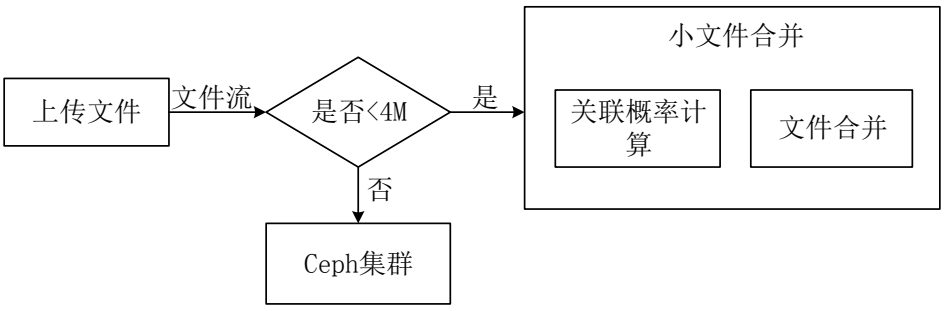

图 5.2 小文件合并模块流程图

小文件合并的具体过程是：先判断上传文件是否属于小文件，若是则通过第三章 3.3.1 小节计算出小文件的关联概率，然后根据第三章 3.3.2 小节提出的小文件关

联合并算法进行关联合并然后再存储至 Ceph 集群。若上传文件为大文件，则直接存储到 Ceph 集群。

§5.1.2 小文件索引模块

索引模块是影响小文件查找速度快慢的关键因素，随着大规模小文件的合并存储，如果没有给小文件与合并文件之间建立相应的索引文件，用户在查找文件时，需要遍历所有合并文件，降低了用户的访问效率。因此，在海量小文件存储系统设计时，加入文件索引模块，该模块在小文件合并的同时将小文件与合并文件之间的映射关系生成索引文件。其中，索引文件的数据结构如表 5-1 所示。

表 5-1 索引文件结构

索引文件结构
filename：小文件名
file_length：小文件大小
file_offset：小文件的偏移量
mergename：合并文件块名
merge_length：合并文件块的大小

索引文件是存储在系统客户端上的，索引文件的结构由<key, value>组成，具体格式为<filename, [file_offset, file_length]>，其中使用 B+树索引中的 key 值来保存小文件的文件名及 ID 等小文件的关键信息，对应的 value 用于记录小文件在合并后大文件的起始位置偏移量 file_offset 及小文件的大小 file_length。索引模块的具体设计与实现在第三章 3.5 小节中有详细介绍。

§5.1.3 小文件预读取模块

小文件预读取模块与系统中小文件访问效率的高低密切相关，当用户发起读请求时，用户在读取请求文件的同时将与其相关的小文件预读取出来并存储至缓存中，用户从缓存中读取文件，可以减少用户的访问时间，提高用户的访问效率。小文件的预读取流程如图 5.3 所示。

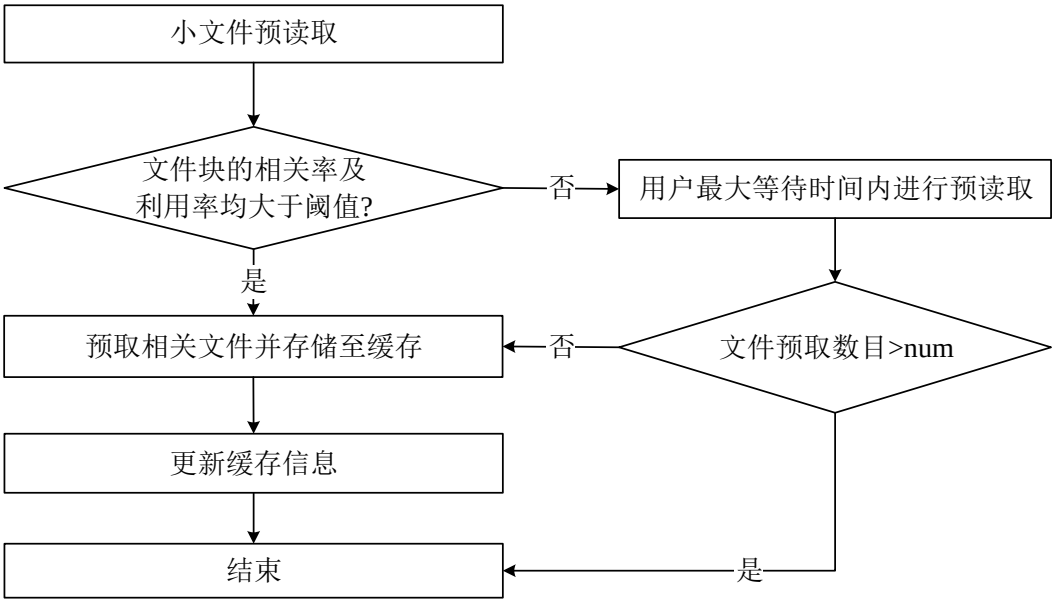

图 5.3 小文件预读取流程图

由图 5.3 可知，在预读取小文件时，首先通过第三章 3.6.1 小节中的公式 3-4 及公式 3-5 计算出合并文件块的相关率及利用率，并判断该文件块的利用率及相关率是否均大于给定阈值，若大于阈值，则将相关的小文件预取出来并存储到缓存中，若低于阈值，则在用户最大等待时间内，根据第三章 3.6.1 小节中公式 3-8 计算出最大预取个数 num，当预取个数大于 num 值时，就结束预读取，同时将请求文件和预读取文件返回给客户端。

§5.1.4 缓存模块

缓存模块主要包括小文件索引缓存和小文件预取缓存两部分，具体介绍如下：

小文件索引缓存：在小文件合并存储过程中，存储小文件与合并文件块间的映射关系及各个合并文件块的文件名及文件大小等属性信息。其索引结构与第三章 3.5.2 小节图 3.7 相同。

小文件预取缓存：在用户读取小文件过程中，根据第三章 3.6.1 小节中预读取算法，对小文件进行预读取并存储至缓存中。缓存模块中存储文件的具体格式是<key, value>，与索引文件结构不同的是缓存模块中的 value 中保存的是小文件的内容，key 中保存的依然是小文件名。缓存请求流程图与第四章 4.3.1 小节图 4.4 相同。

用户在读取小文件时，客户端首先查找缓存中是否存在请求文件，若存在，则立即从缓存中读取出该文件并返回给用户，可以减少用户与集群的交互，从而减少用户的访问时间，提高系统的访问效率。若不存在，则向系统发出读请求，并根据 5.1.2 小节索引模块生成的索引文件，获取请求文件在合并文件块中的具体位置信息，然后根据索引信息从 Ceph 集群中读取文件，同时利用 5.1.3 小节的预取机制，

将与其相关的小文件预取出来并存储到缓存中。

§5.2 系统 I/O 操作

§5.2.1 系统文件存储流程

在海量小文件存储系统中，当用户文件成功上传后，都可以在该系统上看到各个文件相应的虚拟文件，系统实现小文件存储的流程如图 5.4 所示。

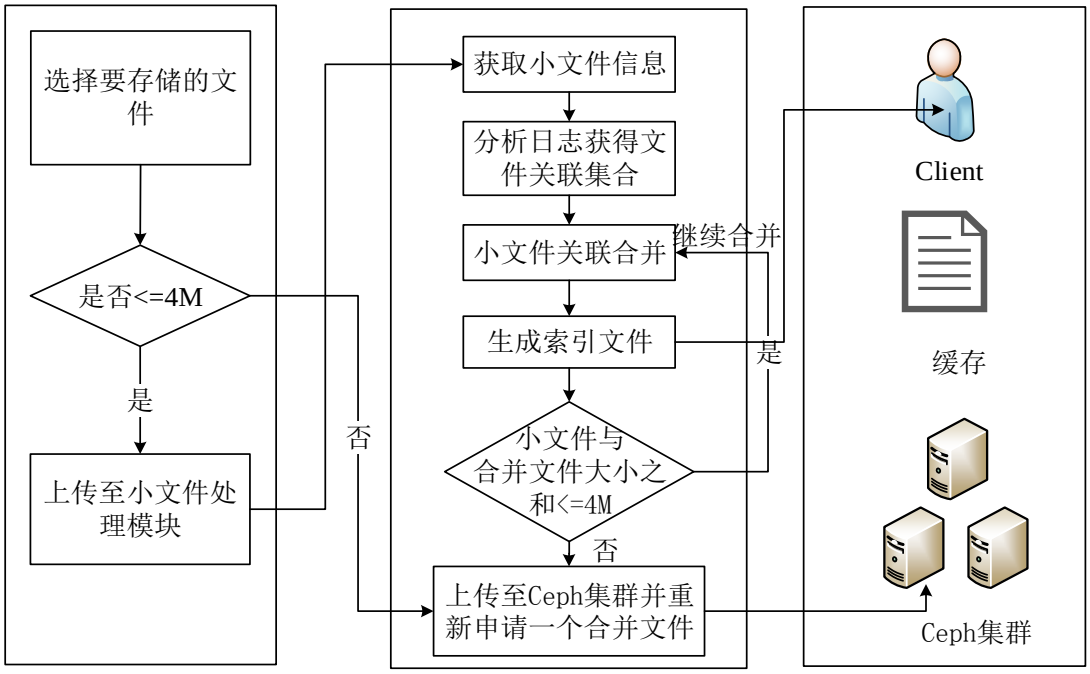

图 5.4 小文件存储流程图

该系统实现小文件存储的具体操作过程如下：

(1) 系统在存储文件时，用户需要选择需要存储哪些文件，该系统不仅可以实现单个文件的存储，而且也可以实现文件的批量存储。

(2) 在文件存储时需要先利用文件检测模块，对客户端上传的文件进行检测。系统将合并文件块的阈值设定为 4M，若文件大小大于 4M，则判定该文件为大文件，可以直接上传到 Ceph 集群；否则将该文件请求提交到小文件处理模块。

(3) 小文件处理模块的主要作用：首先获取各个小文件的属性信息，然后利用小文件关联概率计算模块。该模块通过分析统计小文件的历史访问日志，得到小文件之间的关联概率，再根据小文件间的关联概率设计的关联合并算法实现小文件的关联合并，然后将合并文件上传至 Ceph 集群。在小文件合并过程中，利用索引模块将小文件与合并文件间的映射关系生成索引文件。同时为了避免每一个文件块的最后一个小文件跨文件块存储，需要保证将要合并的小文件和文件块的大小之和小于等于阈值 4M。

(4) 当小文件完全上传后，将索引文件存储到客户端，同时将小文件的上传列

表通过系统界面显示给用户，用户可以直观的查看集群里小文件的属性信息。

§5.2.2 系统文件下载流程

图 5.5 显示了文件存储系统中小文件下载的具体流程，用户登录海量小文件存储系统后，用户在 Ceph 集群里上传的文件名、文件大小等文件属性都会直观的显示出来，用户可选择相应的文件进行下载，下载过程中会发生远程数据的传输。

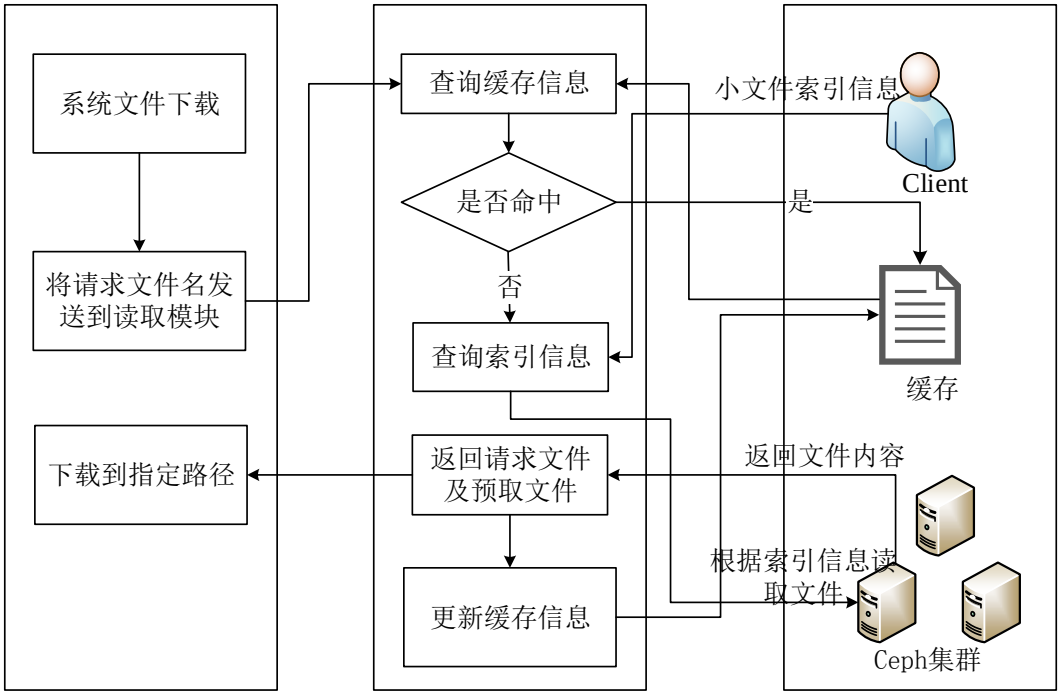

图 5.5 小文件下载流程图

由图 5.5 可知，小文件的具体下载过程是：

(1) 用户在海量小文件存储系统中选择下载功能，该功能不仅可以实现单个小文件的下载，而且也可以实现批量小文件的下载。

(2) 存储系统将用户将要下载的小文件名提交给小文件读取模块，该模块首先通过检查缓存文件，查询该文件是否在缓存中，如果在缓存中，则直接从缓存中读取该文件并返回请求文件给客户端，否则就根据该文件名查找索引信息确定该文件在合并后大文件中的具体位置，并向 Ceph 集群发出读请求。

(3) Ceph 集群给出相应的响应，返回请求文件内容给小文件处理模块，同时在小文件读取时，根据小文件间的关联关系把与请求文件相关联的小文件预读取出来并写入缓存中，同时更新缓存信息。

(4) 小文件处理模块接收到文件内容时，立即将文件内容返回存储系统，用户可以将文件下载到用户指定路径。

§5.3 基于 B/S 的小文件存储系统

Ceph 集群为用户提供的访问方式是一种类似于 Linux 命令的 Ceph Shell 命令访问方式，这种方式并不能给用户带来比较直观、比较清晰的体验，用户对文件的查找和操作也不是很方便。对此本文设计实现了以 Ceph 集群为底层支撑的海量小文件存储系统，该系统利用 B/S 架构，实现客户端与服务器端的交互。

系统采用 B/S 组织结构模式的优点是：

(1) 系统的维护及升级方法简单。目前应用软件的维护和升级是不可避免的，而采用 B/S 结构的软件的维护和升级则变得十分简单，因为基于该架构的软件只需要维护和升级服务器即可，这样可很大程度的节省人力、物力和财力。

(2) 系统开发的成本低而且操作系统的选择增多。因为 B/S 是在浏览器基础上实现的，所以其实现方法简单，主要核心功能的实现在服务器端，开发成本降低。目前，windows 操作系统已不再是唯一的选择，目前主要流行的操作系统有 Linux、Unix 等，因此，系统的服务器操作系统的选择可以有很多种。

(3) 因为 B/S 是多重架构，要求系统中各部分功能相互独立，因此系统可以实现较好的重用功能。

小文件存储系统为用户提供了一系列接口，主要包括用户登录、用户上传、用户下载、集群信息的查询、文件的读写、文件的修改和遍历文件夹等，系统中这些接口的具体描述如表 5-2 所示。

表 5-2 小文件存系统的各个接口信息

登录接口	access (user, password)
参数说明	user：用户账户 password：账户登录密码
功能描述	当用户输入账号和密码后，想要登录系统时就会触发该接口。
查询集群状态接口	bucket(pgmap, bytes_total, bytes_used, bytes_surplus)
参数说明	pgmap：Ceph 集群的所有 PG 的状态信息 bytes_total：存储集群的总容量 bytes_used：存储集群的已用容量 bytes_surplus：存储集群的剩余容量
功能描述	当系统需要查询 Ceph 集群的状态信息时就会触发该事件。
上传接口	upload(files, bpt)
参数说明	files：系统上传的文件 bpt：建立的相应的 B+树索引文件

功能描述	当系统上传文件时，就会触发该接口实现文件的上传及索引文件的建立。
下载接口	download(files, bpt)
参数说明	files：系统想要查找的文件 bpt：相关的索引文件信息
功能描述	当系统需要下载文件时，该接口就会触发。
合并接口	merge(file, position, offset,bpt)
参数说明	file：要合并的文件 position：文件的写入的位置 offset：文件的偏移量 bpt：建立 B+树索引，其中 key 存储小文件名，value 存储小文件在大文件中位置信息
功能描述	当系统实现文件上传过程中会触发文件合并接口实现小文件的合并并建立相应的索引。
文件读取接口	read(filename, buffer, position, bpt)
参数说明	filename：要读取的小文件名 buffer：查看该文件是否在缓存中 position：文件读取时的起始位置 bpt：根据文件名查询小文件在大文件中具体位置信息
功能描述	当系统从集群中读取文件时，该接口将会被触发实现文件的远程传输过程。

§5.4 系统展示

海量小文件存储系统的主要实现包括用户登录模块、用户功能选择模块、系统状态查询模块、小文件上传模块以及小文件下载模块。

(1) 用户登录模块：系统用户通过账号和密码登录存储系统。该模块根据口令来实现账户的验证方法，用户只要知道自己账户的账号和密码，就可以实现远程服务器的登录验证。

用户通过账号和密码登录存储系统，首先判断该账号是否存在，若存在，则用户可以直接登录，若不存在，则提醒用户注册账号后再登录。用户登录界面如图 5.6 所示。

图 5.6 用户登录界面

(2) 用户功能选择模块：主要作用是用户登录系统后，既可以查看系统的存储容量情况，也可以自行选择小文件的存上传或是小文件的下载功能。用户功能选择界面如图 5.7 所示。

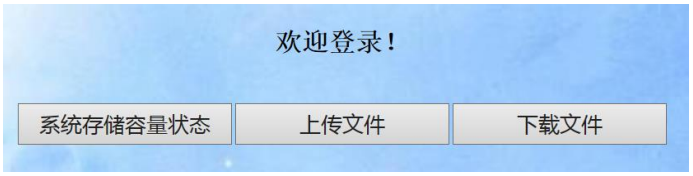

图 5.7 系统选择功能界面

(3) 系统存储容量状态模块：该模块的主要功能是方便用户查询系统存储容量使用和剩余情况，用户可以通过返回键返回用户选择界面，其中集群状态模块如图 5.8 所示。

图 5.8 集群状态信息

(4) 文件上传模块：该模块的主要功能是用于用户自主选择要上传的文件，可以实现文件的批量上传也可以实现单个文件的上传，可通过重置键重新选择要存储的文件。系统文件上传的测试结果如图 5.9 所示，文件成功上传后可通过返回键返回系统主页面。

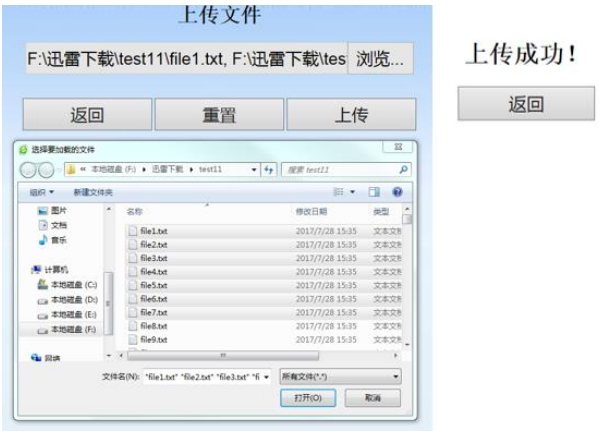

图 5.9 文件上传测试

(5) 文件下载模块：该模块的主要作用是用户可以任意选择将要下载的文件，也可以实现批量下载功能。用户在文件下载时，首先需要选择要下载的文件，然后选择小文件的保存路径，最后点击下载，实现文件的下载。系统下载文件测试过程如图 5.10(a)(b)所示。

(a)

(b)

图 5.10 文件下载测试

§5.5 本章小结

本章主要研究介绍了海量小文件存储系统的设计与实现，首先介绍了海量小文件存储系统的整体设计，然后介绍了系统客户端模块、小文件处理模块以及 Ceph 分布式文件系统三个模块的主要功能，重点介绍了系统实现文件存储的具体流程以及系统文件读取的具体流程，并介绍了采用 B/S 设计的存储系统的优点，以及存储系统在实现过程中的部分核心接口，最后介绍了海量小文件存储系统的具体实现，并展示了该系统的各个功能模块的界面。

第六章 总结与展望

§6.1 总结

随着科学信息技术的迅速发展，各个领域中产生的数据量也在迅速增长，在这些数据中大多数为小文件。面对海量小文件的存储，传统的存储系统已无法满足小文件的高效存储及访问需求。Ceph 分布式存储系统因其高有效性、高可靠性、高扩展性在企业和科研领域得到了广泛关注。但 Ceph 分布式文件系统在存储海量小文件时，仍存在着大多数存储系统都会出现的问题，如没有考虑小文件之间的关联性，没有考虑小文件跨块存储问题等。因此本文在研究和分析了国内外给出方法的基础上，提出了一种基于 Ceph 的海量小文件的存取优化方法，解决小文件的存储和读取问题。具体工作介绍如下：

(1) 介绍了在云存储环境下海量小文件的产生，Ceph 分布式文件系统在存储海量小文件时存在的不足，研究分析了国内外关于小文件问题的最新解决方法。然后简单介绍了 Ceph 分布式存储系统的主要组件和整体架构，详细介绍了系统的数据存储过程及 Ceph 系统针对不同应用场景提出的三种对外存储接口，论文使用的是其中之一的 RBD 存储接口。

(2) 针对由于 Ceph 集群数据的双倍写入，从而导致集群在存储海量小文件时存储性能下降问题，提出了一种基于文件关联合并和预读取的优化方法。该方法主要包括了小文件关联模块、小文件合并模块、索引模块、小文件预取模块及缓存模块。

1) 小文件关联模块：主要通过分析小文件的历史访问日志，获得各个小文件之间的关联关系，然后对关联集合进行合并处理，使得原本只有两个文件的关联集合合并成多个文件的关联集合，避免文件的重复存储，浪费存储空间。

2) 小文件合并模块：在小文件上传时，首先获取小文件的属性信息，判断该文件是否属于小文件，若是则提交到小文件关联模块得到小文件间的关联关系，然后根据其关联性进行小文件合并。为了避免小文件跨块存储，需要保证每一个将要合并的小文与大文件块的大小之和小于等于给定的阈值 4M。小文件的合并后可以大大减少小文件的数量，提高了小文件的存储效率。

3) 索引模块：在小文件合并过程中，将小文件与大文件间的映射关系生成索引文件。首先为大文件建立索引，然后再为合并大文件块中的小文件建立局部索引。用户在访问文件时通过索引信息可以迅速查找到小文件的具体位置信息，解决了小文件访问时，需要遍历所有文件，降低文件读取效率的问题。

4) 小文件预读取及缓存模块：在访问小文件时，通过合并文件块之间的相关率及利用率来衡量小文件之间的关联性，并考虑用户的最大等待时间等因素，将相关

联的小文件预读取到缓存中。用户从缓存中读取文件，可以减少与系统的交互，可以有效的提高文件的访问效率。

(3) 针对 LRU 缓存替换算法的时间局限性问题，提出了一种改进的 LRU_W 算法，该算法给缓存中的每一个缓存对象定义了一个权重因子，该权重会随着缓存对象的长时间不被访问而下降。若该权重低于相应的阈值，则从缓存中移除，减少缓存空间的浪费，提高缓存命中率，并应用到了实际设计实现的海量小文件存储系统中。

在优化后的 Ceph 集群上设计实现了海量小文件存储系统，该系统主要实现了文件的上传和下载功能以及对底层系统容量的查询功能。利用系统的文件上传和下载功能对 Ceph 集群优化前后的存储和访问效率进行了测试，结果表明优化后的 Ceph 集群在小文件的访问效率方面有较好的提高。

§6.2 展望

本文针对 Ceph 系统在存储海量小文件时出现的高延迟、实际内存消耗较高等问题，提出的基于 Ceph 的海量小文件的存取优化方法，并设计实现了一个基于 Ceph 的海量小文件存储系统，虽然该系统在访问性能方面有较好的提高，但本文的研究工作仍有一些不足，需要在未来的工作中做进一步完善，主要包括以下几个方面：

(1) 该系统的设计实现的存储对象主要是文本文件，虽然也可用于 word 文件但却还不适用与视频文件及 pdf 文件等，在这方面仍需要继续完善。

(2) 本文提出的小文件存储方法中，小文件的存储效率需要进一步提高。在文件合并前需要消耗部分时间来找到小文件之间的关联性，这就在不同程度上增大了小文件的存储时间，因此需要进一步优化合并算法。

(3) 在访问文件时，仅仅考虑了文件合并后，合并文件块内小文件间的关联关系，没有考虑不同合并文件块间的小文件相关率对小文件访问效率的影响。

(4) 论文测试没有考虑异构环境对海量小文件存取性能的影响，没有考虑异构环境下，不同缓存机制性能的提升是否可以提高小文件的读取性能。

参考文献

[1] Yang Chaowei, Huang Qunying, Li Zhenlong, et al. Big Data and cloud computing: innovation opportunities and challenges[J]. International Journal of Digital Earth, 2017, 10(1):13-53.

[2] Hu Han, Wen Yonggang, Chua Tat-Seng , et al. Toward Scalable Systems for Big Data Analytics: A Technology Tutorial[J]. IEEE Access, 2017, 2(1):652-687.

[3] Wang Lujun, Long Xiang, Wu Xingbo, et al. SFFS:Low-Latency Small-File-Oriented Distributed File System[J]. Journal of Frontiers of Computer Science and Technology, 2014, 8(4):438-445.

[4] 焦嘉烽，李云．大数据下的典型机器学习平台综述[J]．计算机应用，2017, 37(11):3039-3047.

[5] Grant Mackey, Saba Sehrish, Jun Wang. Improving Metadata Management for Small Files in HDFS. Proceedings of 2009 IEEE International Conference on Cluster Computing and Workshops. 2010.

[6] Liu Xin, Zhao Dehai, Xu Liang, et al. A Distributed Video Management Cloud Platform Using Hadoop[J]. IEEE Access, 2017, 3:2637-2643.

[7] Hadoop archives[EB/OL]. http://hadoop.apache.org/common/docs/current/hadoop_archives.html.

[8] Sequence file Wiki[EB/OL]. http: //wiki. apache. org/hadoop/Sequence File.

[9] Map file[EB/OL].http: //hadoop.apache. org/common/docs/current/api/org/apache/hadoop/io/MapFile. html.

[10] Ramesh D, Patidar N, Kumar G, et al. Evolution and analysis of distributed file systems in cloud storage: Analytical survey[C]//Computing, Communicationand Automation (ICCCA), 2016 International Conference on. IEEE, 2016: 753-758.

[11] Poslad S, Middleton S E, Chaves F, et al. A Semantic IoT Early Warning System for Natural Environment Crisis Management[J]. IEEE Transactions on Emerging Topics in Computing, 2017, 3(2):246-257.

[12] Elomari A, Maizate A, Hassouni L. Data storage in big data context: A survey[C]//

International Conference on Systems of Collaboration. IEEE, 2017:1-4.

[13] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]//Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on. IEEE, 2010: 1-10.

[14] Weil S A, Brandt S A, Miller E L, et al. CEPH: a scalable, high-performance distributed file system[C]//7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA. 2006:307-320.

[15] Weil S A. CEPH: reliable, scalable, and high performance-distributed storage[J]. Santa Cruz, 2007.

[16] 翟艳堂. 腾讯大规模 Hadoop 集群实践 [EB/OL]. http://www.csdn.net/article/2014-02-19/2818473-Tencent-Hadoop. 2014/02/19

[17] 罗锦莉. 阿里云四大海外数据中心将开服[J]. 金融科技时代, 2016(12):80-80.

[18] Lin Huo, Ran Yi. Research on Metadata Management Scheme of Distributed File System[C]//Computer Science and Applications (CSA), 2015 International Conference on. IEEE, 2015:37-41.

[19] 王意洁, 孙伟东, 周松,等. 云计算环境下的分布存储关键技术[J]. 软件学报, 2012, 23(4):962-986.

[20] Zhang Lizong, Cui Yuan, Luo Guangchun, et al. Dynamic Load Balance Algorithm for Big-data Distributed Storage[J]. Computer Science, 2017, 44(5):178-183.

[21] Hua Xiayu, Wu Hao, Ren Shangping. Enhancing Throughput of Hadoop Distributed File System for Interaction-Intensive Tasks[J]. Journal of Parallel and Distributed Computing, 2014, 74(8):2770-2779.

[22] Zheng Tong, Guo Weibin, Fan Guisheng, et al. Research on Optimization Method of Merging and Prefetching for Massive Small Files in HDFS [J]. Computer Science,2017(b11):516-519.

[23] You Xiaorong, Cao Sheng. Storage Research of Small Files in Massive Education Resource[J]. Computer Science, 2015, 42(10):76-80.

[24] Fu Songlin, He Ligang, Huang Chenlin, et al. Performance Optimization for Managing Massive Numbers of Small Files in Distributed File Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(12):3433-3448.

[25] Li Hongqi, Zhu Liping, Sun Guoyu, et al. Design and implementation of distributed mass small file storage system [J].Computer Engineering and Design, 2016,37(01):86-92.

[26] Wang Tao, Yao Shihong, Xu Zhengquan, et al. An Effective Strategy for Improving Small File Problem in Distributed File System[C]// International Conference on Information Science and Control Engineering. IEEE Computer Society, 2015:122-126.

[27] Li Linyang, Lv Zhiping, Cui Yang, et al. The Optimized Cloud Storage Method of Massive GNSS Small Files[J]. Geomatics and Information Science of Wuhan University, 2017, 42(8):1068-1074.

[28] Meng Bing, Guo Weibin, Fan Guisheng, et al. A novel approach for efficient accessing of small files in HDFS: TLB-MapFile[C]// IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, NETWORKING and Parallel/distributed Computing. IEEE Computer Society, 2016:681-686.

[29] Mu Yanliang, Xu Zhenming. An Improved CRUSH Algorithm based on Temperature factor in Ceph Storage[J]. Journal of Chengdu University of Information Technology, 2015, 30(6):563-567.

[30] Weil S A, Brandt S A, Miller E L, et al. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data[C]// ACM/IEEE Conference on Supercomputing. ACM, 2006:31.

[31] Beckmann N, Sanchez D. Modeling cache performance beyond LRU[C]// IEEE International Symposium on High PERFORMANCE Computer Architecture. IEEE Computer Society, 2016:225-236.

[32] Jaleel A, Najafabadi H H, Subramaniam S, et al. CRUISE: cache replacement and utility-aware scheduling[J]. Acm Sigarch Computer Architecture News, 2012, 40(1):249-260.

[33] Jung D Y, Lee Y S.Cache Replacement Policy Based on Dynamic Counter Method[J]. Advanced Science Letters, 2012, 19(5):1530-1534.

[34] Jiang B, Nain P, Towsley D. LRU Cache under Stationary Requests[J]. Acm Sigmetrics Performance Evaluation Review, 2017, 45(2).

[35] Perkowitz S. A survey of Web cache replacement strategies[J]. Acm Computing Surveys, 2003, 35(4):374-398.

[36] Ding Jian, Wang Yongbin, Wang Shujuan, et al. Design and implementation of high efficiency acquisition mechanism for broadcast audio material[C]// IEEE/ACIS, International Conference on Computer and Information Science. IEEE, 2017:667-670.

[37] Niu Dejiao, Cai Tao, Zhan Yongzhao, et al. Metadata Caching Subsystem for Cloud Storage[J]. Applied Mechanics and Materials, 2012, 214:584-590.

[38] Huang Xueyu, Zhong Yanqing. Web Cache Replacement Algorithm Based on Multi-Markov Chains Prediction Model[J]. Microelectronics and Computer, 2014(5):123-125.

[39] 王文建. 基于访问路径挖掘的 Web 缓存与预取模型研究[D].西南交通大学,2014.

[40] García R, Verdú E, Regueras L M, et al. A neural network based intelligent system for tile prefetching in web map services[J]. Expert Systems with Applications An International Journal, 2013, 40(10):4096-4105.

[41] Jing Cong, Wang Mei, An Peng Cheng, et al. 3D model prefetching system based on neural network[J]. Computer Applications and Software, 2015, 32(7):182-185.

[42] 陈凌剑, 王勇, 俸皓. 基于网络延时的 CEPH 存储性能优化方法[J]. 微电子学与计算机, 2017, 34(6):84-88.

[43] 李翔. Ceph 分布式文件系统的研究及性能测试[D]. 西安电子科技大学, 2014.

[44] Lamport L. The part-time parliament[J]. Acm Transactions on Computer Systems, 1998, 16(2):133-169.

[45] Weil S A, Pollack K T, Brandt S A, et al. Dynamic Metadata Management for Petabyte-Scale File Systems[C]// Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference. IEEE, 2004:4-4.

[46] Weil S A, Leung A W, Brandt S A, et al. RADOS:a scalable, reliable storage service for petabyte-scale storage clusters[C]// International Petascale Data Storage Workshop. DBLP, 2007:35-44.

[47] Van der Ster D, Wiebalck A. Building an organic block storage service at CERN with Ceph[C]//Journal of Physics: Conference Series. IOP Publishing, 2014, 513(4):

042-047.

[48] Varga E, Lendak I, Gavric M, et al. Applicability of RESTful web services in control center soft ware integrations[C]// International Conference on Innovations in Information Technology. IEEE, 2011:282-286.

[49] Gamez-Diaz A, Fernandez P, Ruiz-Cortes A. An Analysis of RESTful APIs Offerings in the Industry[C]//International Conference on Service-Oriented Computing. Springer, Cham, 2017: 589-604.

[50] Mohan C, Haderle D, Lindsay B, et al. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging[J].ACM Transactions on Database Systems (TODS), 1992, 17(1):94-162.

[51] Kim H, Ryu M, Ramachandran U. What is a good buffer cache replacement scheme for mobile flash storage[J]. Acm Sigmetrics Performance Evaluation Review, 2013, 40(1):235-246.

[52] 段良涛. 基于云平台的多媒体管理技术研究[D].青岛：青岛理工大学, 2015.

[53] Shi En, Gu Daquan, Feng Jing, et al. Research and improvement of B + tree indexing mechanism[J]. Application Research of Computers, 2017, 34(6):1766-1769.

[54] Priyadarshini S, Sahoo G. A Modified and Memory Saving Approach to B+ Tree Index for Search of an Image Database based on Chain Codes[J]. International Journal of Computer Applications, 2011, 9(3):24-28.

[55] Zhou Wei, Lu Jin, Luan Zhongzhi, et al. SNB-index: a SkipNet and B+ tree based auxiliary Cloud index[J]. Cluster Computing, 2014, 17(2):453-46.

[56] Zou Zhenyu, Zheng Quan, Wang Song, et al. Optimization Scheme of Small File in Cloud Storage System Based on HDFS[J]. Computer Engineering, 2016, 42(03):34-40+46.

[57] Cheng Wenjuan, Zhou Miaomiao, Tong Bing, et al. Optimizing small file storage process of the HDFS which based on the indexing mechanism[C]// IEEE, International Conference on Cloud Computing and Big Data Analysis. IEEE, 2017:44-48.

[58] 刘柳. 面向个人云存储的缓存机制研究[D]. 浙江大学, 2015.

[59] Yu Hao. Platform Design of Sports Meeting Management System for Regular Colleges and Universities Based on B/S Structure[J]. Wireless Personal Communications, 2018:1-10.

致 谢

时间转眼即逝，我的研究生生活马上就要结束了，在这三年的学习与生活中，我经历了不知所谓时的迷茫，满心期待到最后的伤心失望等，这些经历不仅让我让一点点的成熟长大，而且也让我收获颇丰。在这里我不仅学习到了知识，而且也认识了很多良师益友，他们的陪伴和帮助是我在这最美好的回忆。

在这里我想要对曾经给予我帮助的老师们、师兄师姐、师弟师妹、班级同学以及我的舍友们表示感谢。

首先，我需要感谢我的导师王勇老师。王老师治学严谨，知识渊博，为人和善，对我们学术要求严格，跟随王老师的这几年让我受益匪浅。从最初确定研究方向到论文的选题，到最后优化系统的设计与实现，以及现在论文的撰写，王老师都给了我悉心的指导。如果没有王老师的帮助和支持，该论文也许不会这么的顺利完成。其次，我想要感谢叶苗和俸皓老师，感谢他们在我每次遇到问题时，都耐心的给予指导，不仅在生活上给予我帮助而且在科研论文的撰写过程中，他们都给了我非常有用的建议。如果没有两位老师的耐心教导和诚心帮助，该论文同样不会顺利完成，在这里再次感谢老师们的悉心教导。

我还要感谢实验室的师兄师弟们以及同级的同学，刘玉明师兄不仅在生活上而且在学习研究上都给予了我很大的帮助，柯文龙师兄也经常在学习上给我们解惑，师弟们也在我论文撰写时给予了不少帮助。在和同实验室的周慧怡、何雄森、韦海宇、谢兵兵、吴铎、刘辉勇一起学习的过程中，他们都曾给予我不少的帮助，尤其是在我刚确定研究方向时，很多不懂的地方都是他们在耐心的教我。同时我也感谢我的舍友们，感谢她们创建了一个舒适的睡眠环境，只有休息好，才有精力好好的学习。

最后，我要感谢我的家人，感谢爸爸妈妈对我的支持，感谢哥哥姐姐在生活上给予我的帮助，你们从不对我要求什么，你们一直在背后默默的为我付出，非常感谢你们对我的支持，你们也是我奋斗和努力的动力，借此表达我对你们最深的敬意。

在这里也衷心的感谢参与论文评审的各位专家，谢谢你们在百忙之中给本文提出宝贵的意见。

作者在攻读硕士期间主要研究成果

一、学术论文

[1] 陆小霞,王勇,雷晓春.Ceph 系统中海量气象小文件存取性能优化方法[J].桂林电子科技大学学报,2018,38(6).

二、专利

[1] 以第 2 作者（导师第一作者）申请发明专利《一种基于 Ceph 的海量小文件存取优化方法》,申请号 201810343960.6.

三、参与科研情况

[1] 中电科海洋信息技术研究院有限公司,云安全存储与云应用安全研究(61631019)。