电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

硕士学位论文

MASTER THESIS

论文题目 基于CEPH分布式文件系统的云盘系统的设计与实现

学科专业 计算机应用技术

学　　号 201421060404

作者姓名 孙嘉岑

指导教师 向艳萍　　教　授

独创性声明

本人声明所呈交的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。据我所知，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作了明确的说明并表示谢意。

作者签名：孙嘉岑 日期：2017年 6月20日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技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应遵守此规定）

作者签名：孙嘉岑 导师签名：

日期：2017 年 6 月 20 日

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

基于 CEPH 分布式文件系统的云盘系统的设计与实现

孙嘉岑

指导教师 向艳萍 教 授

电子科技大学 成 都

申请学位级别 硕士 学科专业 计算机应用技术

提交论文日期 2017.04.18 论文答辩日期 2017.05.12

学位授予单位和日期 电子科技大学 2017 年 06 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

Design and Implementation of Cloud Disk System based on CEPH Distributed File system

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Computer Applied Technology**

Author: **Sun Jiacen**

Supervisor: **Xiang Yanping**

School: **School of Computer Scicence & Engineering**

摘 要

数据是当今互联网时代最核心最具价值的产物之一，个人在消费和使用互联网产品的同时产生了大量需要持久化存储的数据，个体存储设备的解决方式在这样一个存储量和存储要求日益增长的条件下不再能够满足个人用户的要求，因此越来越多的个人用户成为了云存储的受众。云计算几十年的发展使得公有云环境下的存储技术已然十分成熟，商业环境下诞生了许多优秀的云盘产品，但其核心技术是对外封闭的，而对于这种面向大量个人用户的公有云盘系统，开源社区也并没有完善可靠的整体解决方案。

本文基于上述原因将公有云盘涉及的各个关键模块和技术点进行逐个研究和设计，提出了一整套可靠的云盘系统实现架构。首先对云盘的远程文件系统操作机制的原理和技术进行了分析，研究并设计出 CFSRPC 协议，给出了协议的客户端和服务端的具体格式定义和二者的通信机制，云盘系统通过对该协议的实现，可以透明化远程文件操作的细节，方便客户端和服务端上层业务的实现。与此同时为该协议设计了事务机制以颗粒化文件传输服务，设计了 CSDA 算法在协议级别动态调节文件传输的分块大小，增加了该协议对云盘系统性能的优化。

接着对云盘系统的核心业务流程进行分析，针对云盘系统依赖的多个关键模块进行了分析和整合设计，总结出以访问层、管理层、存储层为核心的整体软件架构。基于 Ceph 分布式文件系统设计核心存储模块，研究分析 CephFS 和 MDS 的实现原理并根据云盘系统架构和 CFSRPC 协议的定义，结合多路 IO 复用机制设计实现了 Ceph 的接入模块 CCD。将云盘系统的请求分化为文件系统操作请求和管理功能操作请求，分别以 CCD 模块和管理功能模块来接受和处理，中间加入 LVS+Keepalived 的负载机制使得 CCD 节点和管理功能节点支持横向扩展。参考 Keystone 的令牌认证机制设计实现了独立认证系统。以 NW.js 为框架根据 CFSRPC 协议的定义设计实现了客户端。最后将整套云盘系统在测试环境下进行部署，完成了认证流程和云盘基本操作的测试，并对单点 CCD 进行了性能测试，确认整体系统能够达到公有云生产环境的性能要求。

关键词：云盘、挂载协议、ceph

ABSTRACT

Data is one of the most central and valuable products of the Internet at today's era, individuals produce large amounts of data that require persistent storage while consuming and using Internet products. The solution of individual storage devices is no longer able to satisfy the requirements of individual users in the conditions of ever-growing of storage capacity and storage requirement, so more and more individual users become the cloud storage audience. Decades of development of cloud computing already make the storage technology of public cloud environment very mature, there were a lot of excellent cloud products had been developed in business environment, but their core technology are closed, and for this kind of public Cloud disk system which aim to serve large number of individual users, there aren’t perfect and reliable overall solutions been proposed in open source community.

Based on the above reasons, this thesis studies and designs the key modules and technical points involved in the public cloud disk, and puts forward a set of reliable cloud disk system implementation architecture. Firstly, the principle and technology of the remote file system operating mechanism of cloud disk are analyzed, and the CFSRPC protocol is studied and designed. The specific format and the communication mechanism of the client and the server is given. Through the implementation of the agreement, you can transparent the details of remote file operations to facilitate the client and the server to achieve the upper business. At the same time, the transaction mechanism is designed to granulate the file transfer service, and the CSDA algorithm is designed to dynamically adjust the file size of the file transfer at the protocol level, which increases the performance of the cloud disk system.

Then, the core business process of cloud disk system is analyzed, and the key modules of cloud disk system are analyzed and integrated. Finally, the overall software architecture with access layer, management layer and storage layer as the core is summarized. Based on the Ceph distributed file system design core storage module, the thesis analyzes the implementation principle of CephFS and MDS and implements Ceph access module CCD according to the definition of cloud disk system architecture and CFSRPC protocol and multi-channel multiplexing mechanism. The request of the cloud

disk system is divided into the file system operation request and the management function operation request, which are accepted and processed by the CCD module and the management function module respectively. The load mechanism of LVS + Keepalived in the middle makes the CCD node and the management function node support the horizontal expansion. An independent authentication system are designed and implemented according the reference of Keystone's token authentication mechanism. The client is designed with NW.js as the framework according to the definition of CFSRPC protocol. In the end, the entire cloud disk system are deployed in the test environment, then completed the test of certification process and the basic operation of the cloud disk , and a single point of the CCD performance are tested to confirm the overall system to achieve the performance requirements of the public cloud production environment.

Keywords: cloud disk, mount protocol, ceph

目 录

第一章 绪 论

1.1 选题背景及意义

信息技术发展至今，人们的工作方式和生活方式已然产生了很大的变化，电子文档、照片、音视频文件、工具软件充斥着几乎所有人的日常生活。互联网的兴起让人们热衷于坐在电脑面前工作和娱乐，这样的社会状态下，每个人每天都会产生数据，每个人都成为了存储功能的消费者。人们将重要数据和生活产出存在自己的硬盘上，随着数据越来越重要，照片越来越珍贵，人们开始担心自己的数据是否安全，随着视频越来越高清化，人们开始担心自己的硬盘容量濒临不足。那么，解决这些问题成为了大量人的诉求，继续使用 PC 硬盘作为存储载体肯定是不可行的，单台机器的存储能力和容灾能力是有限且不可提升的。

因此，云存储[1]的概念一而再的刷新人们的期望值，云盘成为满足个人存储需求的关键。通过对存储设备的云化部署和分布式存储系统的技术升级，云盘将打破传统 PC 存储的瓶颈，极大化的扩充容量、提升可靠性。

这样的需求驱动出了一系列优秀的商业产品，百度云盘、腾讯网盘、金山快盘等等。但基于商业竞争的性质，云盘的核心技术依旧是封闭的。然而开源社区为云计算已然贡献出了很多优秀的关键技术实现，缺乏的是一个可靠、可扩展的云盘系统的整体解决方案，分布式文件系统[2]、远程文件系统操作协议、负载均衡技术[3]、认证技术[4]等云盘系统依赖技术在各自独立的技术领域有着成熟的研究成果和开源实现，如何将这些技术进行整合或二次开发，如何借鉴其设计思路以进行云盘系统整体架构和处理流程的研究设计是值得研究的。

1.2 国内外研究发展现状

云计算[5]概念被提出至今，经历了 10 年的蓬勃发展，同时伴随多方面信息技术的不断革新，云计算这个领域早已脱胎换骨。国外的信息技术产业起步早于国内，几乎所有的一流 IT 公司都早已插足云计算核心技术的研究中，他们依靠自己的研究底蕴、根据自己对新技术的一流的洞察力在云计算的市场里已然有着举足轻重的地位，同时也在影响着云计算领域的技术导向。

互联网的兴起以及传统产业的互联网化致使当今这个时代的信息数据量呈现爆发式的增长，继而产生了针对不同用户种类海量数据的存储需求。云存储作为云计算的基本组成部分[6]得到了大量研究资源的投入。在云存储的企业级市场上，国

内外各大云存储服务提供商提供着各自独特且优缺点不一的云存储服务。

1.2.1 企业级云存储

Amazon S3[7]是 Amazon AWS 产品线下的对象存储服务，为企业级用户提供了持久、可扩展、高安全性和可用性的存储解决方案来备份和归档关键数据。对于持久性，S3 首先提供了高度耐用的基础设施，秉承高达 99.99999%的耐久性为设计宗旨，将所有用户数据存储的冗余部分存储在不同的容灾区域。对于其高可扩展性，S3 的弹性扩充能力允许用户不对自身业务的存储需求进行预测，S3 任何时候都可以为用户提供存储资源的扩展和缩减服务。对于安全性，S3 对所有数据的传输进行了 SSL 加密，并支持各种权限策略使得用户定制数据的安全特性。 S3 提供高度可靠的数据传输服务，定期的流量服务能够方便用户在保证成本可接受的的前提下处理自身业务的高峰期。对于高可用性，Amazon S3 Standard 为用户保证一定期间内的超高可用性，同时用户可以通过指定 AWS zone 来优化数据传输延迟并适当节约成本。利用 Amazon S3，可以在保证以上优点的前提下很快地部署互联网应用或者进行大批量关键数据的存档。

EMC 作为一家美国信息存储咨询科技公司，提供分布式、可扩展的软件定义存储解决方案，可支持新一代应用程序。兼具公共云的低成本和、简单性及企业的可靠性。其中，Atmos 借助高可扩展性和专门构建的云存储系统，跨地理区域高效的管理和分发信息。EMC Atmos 是一种横向扩展对象存储平台，专门为存储、归档和访问非结构化数据以及在云规 模上支持无限制的应用程序和服务而构建。Atmos 提供关键的云功能，包括：全局命名空间、 分布式主动/主动体系结构、多租户、元数据驱动型策略、REST API 驱动型存储、计量和按存储容量使用计费——这些都是部署私有云、公共云或混合云存储所必需的。

华为云硬盘（Elastic Volume Service）与 AWS 的块存储服务类似，是一种允许弹性扩充的虚拟块存储服务。它基于分布式的存储资源管理方案，具有极高可靠性和容灾性，并且华为在硬件性能上面的优势使得 EVS 具有的极高的 IO 吞吐能力。EVS 以一个硬盘的形式提供服务，允许用户对其做挂载、格式化等标准的文件系统操作。

1.2.2 个人云存储

Dropbox[8]在提供个人云存储的同时是一款非常方便可靠的网络文件同步工具，它为不同操作系统实现了不同的客户端，并且拥有网页客户端，主要特性是能够为用户实现文件的自动上传和同步，同一个用户在使用不同客户端产生的存储内容

会自动进行云端的备份和其他终端设备的同步，对流媒体支持在线播放，不同用户可以进行文件的共享和分享。

Google Drive[9]是谷歌公司推出的针对个人消费者的云存储服务，支持各种类型文件的在线创建、分享以及协作。Google Drive 内置的 Google Docs，使得用户可以进行文档的在线编辑，Docs 的云化支持使得多个 Google Drive 用户可以进行协同办公。集成了 google 强大的搜索功能，支持关键字、文件类型甚至图像内涵信息的搜索，Google Drive 为用户存储了每个文件 30 天内的每一个版本，用户可以随时做文件的回滚操作。Google Drive 整合了本身的 Gmail、Google+等产品，而且开放了 API 给其他程序使用，任何内嵌了该 API 调用实现的产品都可以使用 Google Drive 的存储服务。

坚果云是一款国内的专业云盘产品，提供了 Dropbox 类似的功能的同时为用户实现了智能的文件管理，提供文件历史版本保存的功能以及强大的文件搜索，其文件同步基于增量的方式加速了文件的备份效率，全平台支持移动办公。坚果云支持其他云盘产品都没有的优质软件兼容模式，支持 WebDAV 协议[10]以及 Omnifocus[11]等优秀的第三方技术的数据同步，这使得用户可以选择甚至自研客户端实现，为专业用户提供了专业接口。

1.3 论文主要工作内容

本文首先针对云盘系统的通用操作设计了一个广域网环境下简洁高效的文件系统远程操作协议 CFSRPC，然后结合该协议的定义，基于 Ceph 分布式文件系统提出了一个公有云环境下云盘系统的整体解决方案。主要工作内容包含几下几点：

1）对 Ceph 分布式文件系统进行了详细介绍并对整体架构和各组件的运作原理进行了研究，然后对云盘的远程挂载技术进行了介绍和对比分析。

2）研究设计了一套基于远程调用模式且适用于本文所述云盘系统的文件系统远程操作协议 CFSRPC，针对云盘挂载情景为该协议设计了事务机制以及文件分块动态调节算法 CSDA。

3）对云盘系统进行整体的需求分析，设计云盘系统整体架构，对业务流程涉及的功能进行模块化设计。

4）研究 Ceph 分布式系统作为云盘系统核心存储模块的具体方案并对其涉及的问题进行分析和解决，设计负载机制。

5）对核心存储模块的接入节点 CCD 进行详细设计，这里包括对 CFSRPC 协议的服务端实现方式的设计、认证调用的设计、接入 Ceph 存储集群方式的设计。

6）基于多进程模型设计独立认证模块，设计云盘系统的客户端。

7）对 2、3、4、5、6 点提出的设计方案进行具体实现。

8）对云盘系统的整体方案进行部署，并从功能上和性能上分别进行测试，并对测试结果进行分析。

1.4 论文组织结构

本文的组织结构安排如下：

第一章，绪论，介绍论文的选题背景和现实意义，对该领域国内外研究现状进行分析，后对论文的主要工作内容和组织结构进行说明。

第二章，相关技术研究分析，对云盘系统涉及的相关技术进行研究分析。

第三章，CFSRPC 协议的研究与设计，研究设计了 CFSRPC 协议，为云盘系统提供挂载方式的技术支持。

第四章，云盘系统核心模块分析与设计，对云盘系统进行整体的需求分析、总体设计、核心存储模块设计、挂载接口 CCD 设计、认证模块设计、管理功能模块设计和客户端设计。

第五章，云盘系统细节设计和整体实现，对第四章设计的子模块进行了细节设计和实现，重点对 CCD 和 CFSRPC 协议的实现做了详细描述。

第六章，云盘系统测试与分析，对本文所设计的云盘系统进行了测试环境搭建、整体功能测试和性能测试。

第七章，总结与展望，总结了全文的主要工作，并提出了云盘系统的不足和待优化部分。

第二章 相关技术研究分析

分布式文件系统和云盘挂载技术是公有云环境下的云盘系统依赖的两个关键技术，分别为系统提供高可靠、高扩展性的存储支撑和高效、稳定的通信方式，为了保证云盘系统设计的高可用性，本章分别对 Ceph 分布式文件系统[12,13,14]和多种云盘挂载技术进行研究分析。

2.1 Ceph 分布式文件系统

分布式系统是指统筹多台计算机协同突破单台计算机受限于本地资源不可扩充而产生的在计算、存储等问题上的瓶颈。分布式存储作为分布式系统中最基本的组成部分通常分为分布式文件系统和分布式数据库。分布式文件系统主要是用于文件存储。互联网的资源，例如大量数据文件、日志文件、图片、音乐等等，最终都会以文件的形式存放在特定的物理存储设备上。存储、读取和管理这些海量的文件依靠单一的存储节点无疑是不够的，所以以集群为管理单位的分布式文件系统应运而生。

Ceph 作为一种集优秀性能、高可靠性和高扩展性于一身的分布式文件系统，能够轻松支持 PB 级别数据的存储，其高可塑性使得该文件系统能够被动优化以适应多种负载环境，本节对 Ceph 分布式文件系统进行介绍和原理研究。

2.1.1 Ceph组件

Ceph 以一个通用的底层存储系统来支撑对象存储、快存储和文件存储的功能，有着高可靠性，易于管理且开源免费，Ceph 拥有存储和管理巨量数据的能力并且提供极高的扩展性，支持上万的客户端访问 PB 级量的数据，单个 Ceph 节点（Ceph Node）以商品化硬件和智能的服务程序表示，一个 Ceph 存储集群（Ceph Storage Cluster）由大量的功能节点集合组成，它们之间相互通信交流以支持数据的动态备份和动态迁移。

Ceph 基于 RADOS（Reliable Autonomous Distributed Object Store）提供一个无限扩展的存储集群，一个最小化的 Ceph 存储集群至少包括两种类型的功能节点，Ceph Monitor 和 Ceph OSD Daemon，这里分别将其简称为 Monitor 和 OSD，一个 Monitor 维护一个集群映射表的备份，一个由多个 Monitor 节点组成 Monitor 集群能够确保在一个 Monitor 节点宕机后的可用性，Ceph 存储集群的客户端可以检索

任何一个 Monitor 来获得映射表的信息，其功能架构如图 2-1 所示。

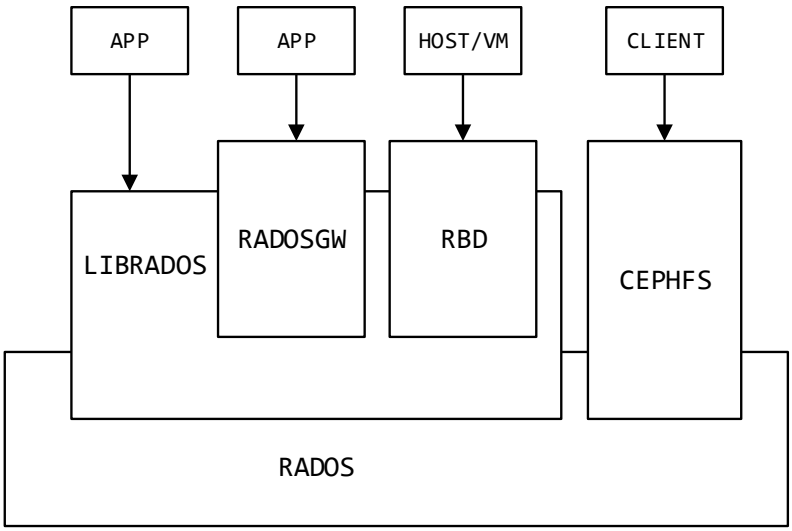

图 2-1 Ceph 功能架构

一个 OSD 会持续检查自己的状态以及集群内的其他 OSD 并将结果信息汇报给 Monitors。

所有的 Client 节点和每一个 OSD 节点都会使用 CRUSH[15]算法来计算的到有效的数据分布信息，而不是依赖于传统的查询中央管理节点的方式，Ceph 的高级特性包括通过 librados 向外提供原始接口以及大量基于其原始接口封装后的服务接口。

图 2-2 对象存储形式

Ceph 存储集群接受 Ceph Clients 的数据并将其以对象的形式存储，这里的 Ceph Clients 包括 Ceph Block Device（块设备）、Ceph Object Storage（对象存储）、Ceph Filesystem（满足 POSIX 标准的文件系统）或者是任何基于 librados 实现的定制化客户端。每个对象存在 OSD 内，其存储形式对应于文件系统里面的一个文件。每一个 OSD 都负责处理对数据对象的读写操作，其存储形式如图 2-2 所示。

OSD 将所有数据按照对象存储在一个平行的命名空间里，每个对象有自己的标识符、二进制数据以及一组以 Key-Value 形式组织的元数据集。例如，Ceph Filesystem 使用元数据来存储文件所属者、修改时间以及用户个人空间的完整目录树结构等。数据对象的组成格式如图 2-3 所示。

标识符 ID 是对象在 Ceph 存储集群全局范围内的唯一标志，二进制数据则存储了该数据对象的具体内容。

ID	Binary Data	Meta Data
1234	01010101010111001010101011001001010100101001001010010101010000111001010101000110000101010101	name1 value1 name2 value2 namen valuen

图 2-3 Ceph 数据对象组成格式

2.1.2 数据分布原理

Ceph clients 和 OSDs 的主业务流都依赖于集群的拓扑结构和数据映射信息，其数据以五个映射表的形式存储在 Monitor 组件中，本文将这五个映射表整体称为 cluster map：

1）The Monitor Map：包含集群的 fsid、位置、命名地址以及每个 Monitor 的端口，同时它还标明了两个时间戳，一个是这个 map 建立的时间，一个是其最后修改的时间。

2）The OSD Map：包含集群的 fsid、建立时间和最新修改时间、pool 的列表、备份数量、PG 号、OSD 的列表及其状态。

3）The PG Map：PG 版本、时间戳、最后 OSD 映射时间、存储占用比例、PG 的详细信息（ID、状态）、每一个 pool 的用量数据。

4）Thre CRUSH Map：存储设备列表、失效的域层次（例如：device、host、rack、row、room 等）。

5）The MDS Map：当前 MDS map 的建立时间和最后修改时间、存储元数据的 pool、元数据服务器列表。

每个 map 维护着操作状态变化的迭代历史。Monitor 集群维护着一份 cluster map 的主备份并做了集群化容灾处理，这个 cluster map 包含集群的所有节点、状态、变更以及 Ceph 存储集群的整体健康状态。

Ceph 系统支持 Pools 的概念，Pools 作为数据对象逻辑上的从属命名空间，这使得每一个对象的唯一标志以（Pool，Object）的类型来标识。一个 Pool 的大小、备份数量、CRUSH 规则集等属性决定了 Ceph 如何存储其中的数据对象。

Ceph 存储数据对象的流程主要包含两个映射过程，首先将文件对象 object 映射到数据分布簇 PG（placement group），然后将 PG 映射到 OSD 存储列表。中间层容器 PG 存在的意义有三点，其一，是为了把 object 分成组，使 PG 的数量级远远小于 object，那么在全局的层面上，Ceph 只需要追踪和维护 PG 的元数据和位置信息，大大节约了开销，其二，PG 容器的数量和规模并不完全随数据存储量的变化而变化，干预性的增加 PG 的数量可以做到均衡每个 OSD 的负载，提高整体并行能力，其三，分离故障域，将故障的数据单位上升到 PG 大大缩小了故障恢复的操作单位数量级。其映射过程如图 2-4 所示。

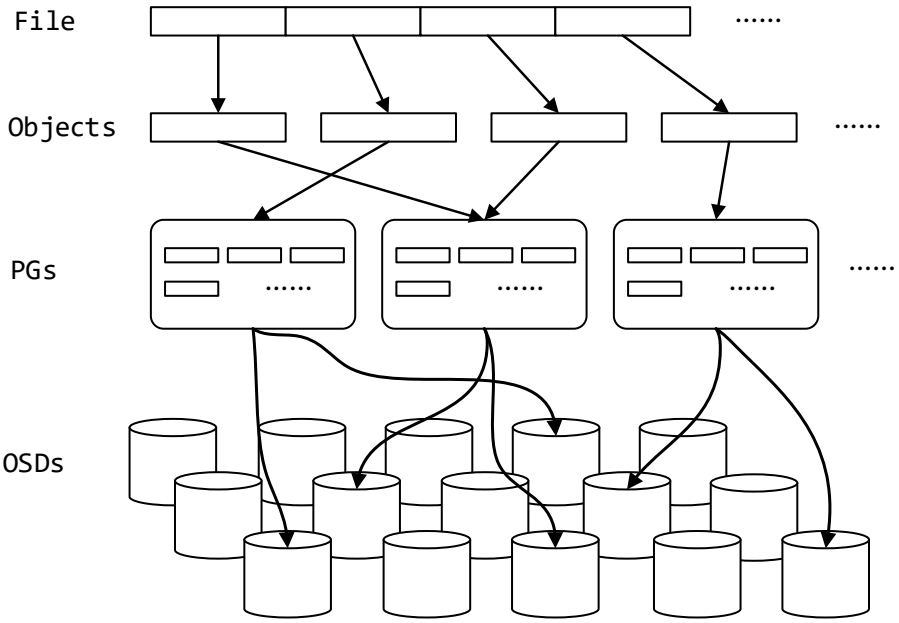

图 2-4 Ceph 数据对象映射过程

Monitor 组件拥有 cluster map 允许 clients 在读写一个数据对象时利用自身的计算资源运行 CRUSH 算法获取其位置信息。当一个 client 与一个 monitor 绑定时，client 会检索其中的最新 cluster map 以获取 monitor 集群、OSD 集群的状态信息。而最终的数据对象的位置信息需要 client 在取得前置信息后通过运行 CRUSH 算法计算得来。过程如下：

1）Client 接收到 pool ID 和 object ID
（例如：pool =“英文小说”，object=“Shakespeare”）。

2）Ceph 将 object ID 进行哈希运算。

3）将 object ID 的哈希结果与 PG 的数量进行取模运算，来得到 PG ID
（例如：PG ID = 98）。

4）Ceph 根据 pool ID 查询得到 pool ID 唯一标识符（例如：22）。

5）Ceph 得到一个对象的名字（例如：22.98）。

在本地节点计算数据对象的位置比建立一个查询会话所带来的开销要小的多。CRUSH 算法允许 client 来计算数据对象应该往哪里写和从哪里读，并直接与该数据对象主副本存储所在的 OSD 进行读写通信例程。

2.1.3 性能分析

2.1.3.1 高可靠性

在传统的存储架构中，客户端首先需要与一个集中式的组件进行交互，在该单点组件里获取到元数据后才能进一步向具体的存储节点发出服务请求。受限于单点失效问题，这种架构在性能和扩展能力方面都有一定的限制。

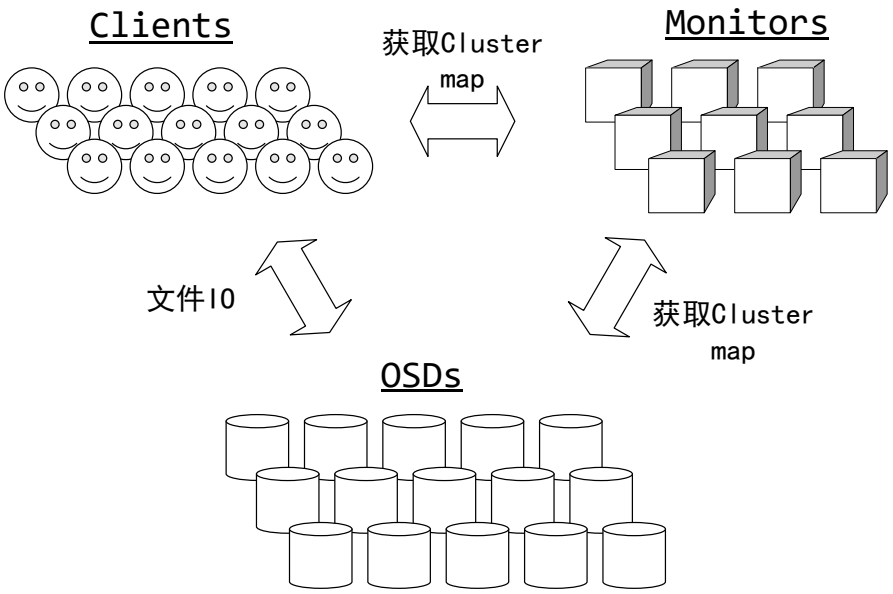

图 2-5 Ceph 组件通信架构

Ceph 放弃了这种集中式的网关，使得 Ceph Clients 能够直接与 OSD 集群进行通信，OSD 创建数据对象的副本存储于其他 OSD 上，以保证数据的安全性和高可用性。同时，Monitor 集群又可以保证管理数据和元数据的高有效性。整体架构如图 2-5 所示。

Ceph clients 在读写数据之前必须要与 Monitors 通信以获取最新的 cluster map 信息。一个 Ceph 集群允许只存在一个 Monitor，但是这将不可避免的引入单点失效问题。

为了获得可靠性和容错性，Ceph 支持 Monitor 集群化。在 Monitor 集群中，延

迟和其他故障会导致一个或多个 Monitor 的状态更新滞后。因为这个原因，Ceph 必须让多个 Monitor 节点根据集群状态达成一致。Ceph 使用大多数 Monitor 节点的状态并使用 Paxos 算法建立一个共识的集群状态作为所有 Monitor 状态更新的参考标准。

Ceph 支持数据多副本和可配置的副本策略以及故障域布局管理，支持数据的强一致性。不存在单点故障，任何组件都可以实现热替换，支持数据的自动迁移。故障的全部恢复过程不需要人工参与，且在恢复过程中能够保持集群的可用。

2.1.3.2 高性能

在许多集群化的架构中，集群成员的主要目标就是让集中化的接口能够知道每个节点的可用性。继而让集中化接口能够向 clients 提供服务。

Ceph 的 OSD 和 Ceph clients 都是集群感知的，意味着每一个 OSD 的存在对其他 OSD 都是可见的，这允许 OSD 节点能够直接与其他 OSD 节点和 Monitor 节点通信，同时，所有的 Clients 也能够直接与任何一台 OSD 节点通信。

Ceph Clients 和 OSD 都使用 CRUSH 算法来有效的计算并获得数据对象的存储位置。CRUSH 相比以往的方式能够提供一个更好的数据管理机制，并且，通过将这个计算流程分布在所有的 Ceph Clients 和 OSD 中使得其存储集群能够支撑起巨量存储数据的维护和检索。

这种各个服务节点互相通信的能力意味着 OSD 服务程序能够方便的利用 Ceph 每个节点的 CPU、内存等资源来处理庞大的任务。这种平衡资源的能力能够为节点带来以下几个主要的高性能支持和功能支持：

1） OSDs 直接为 Clients 提供服务：任何网络设备在同一个时刻能够维持网络连接数量都存在着瓶颈，因此为了打破高压环境下传统集中式系统的物理设备瓶颈，Ceph 允许 Clients 与 OSDs 直接通信以提高二者的性能和单点故障发生后整体系统的并发能力。Clients 甚至可以与一个特定 OSD 位置一个会话而不是面对一个集群进行交互。

2） OSD 成员和状态：OSD 作为一个集群节点随时向 Monitor 集群上报自身的状态，当状态机制设置为最低级别时，OSD 只上报 up 和 down 两种状态来表示该节点是否可用，但当一个 OSD 宕机时，它是无法上报自身状态的，此时，Monitor 集群会与每一个 OSD 节点维护一个心跳链接来确定其节点状态。同时，每一个 OSD 节点也有能力和权利监测和上报其他 OSD 节点的状态，这样的分担机制为 Monitor 减轻了负载。

3） 数据清洗[16]：由于文件系统或者软硬件的故障在一定几率下可能导致数据的毁坏或不完整，OSD 作为维护数据对象一致性完整性的主要组件能够提供短周期的数据清洗和长周期的深度清洗，通过比较同一个数据对象存在不同 OSD 上的备份。深度清洗将会对数据对象进行逐位的检查。

4） 冗余：同 Clients 一样，OSD 也使用 CRUSH 算法，但 Clients 是进行数据对象的定位，而 OSD 是用来计算数据对象及其备份的存储位置。一个典型情境就是对一个对象进行写操作，client 使用 CRUSH 算法计算出数据对象的存储位置，映射到 pool 和 PG，然后在 CRUSH map 中查询到该 PG 对应的主 OSD，接着向该 OSD 执行写操作。OSD 接受了完整的数据后会查询自身的 CRUSH map 找到变化的 PG 的其他备份位置，并进行数据同步。

2.1.3.3 高扩展性

当向一个 Ceph 的集群加入一个新的 OSD 后，为了均衡负载，cluster map 会被新加入的 OSD 更新，之后该 OSD 将会根据新的 cluster map 进行部分 PG 的迁移，于此同时，client 和 OSD 上执行的 CRUSH 算法的输入也相应改变。在一个健康状况良好的 Ceph 集群（意味着每个数据对象的冗余量足够）中，一个 OSD 的增加是必然不会影响任何一个 PG 的主备份，因此， 即使在 clients 和其他 OSDs 正在进行数据对象位置计算时进行 OSD 扩展，CRSUH 依然是稳定的。

OSD 的增加是 Ceph 集群存储资源扩展的最有效途径。人工参与部分仅包括硬件环境搭建、OSD 程序部署，之后便可以一键添加，所有的数据迁移和重布局都是自动完成且不影响 OSD 集群向 clients 提供业务服务。

2.2 云盘挂载技术

在个人云存储系统里，服务端提供的存储服务通常以通用文件系统的形式呈现在客户端，在客户端和服务端的通信过程中涉及到文件系统挂载、操作远程执行以及文件传输。不同的操作系统内核对文件系统的实现区别很大，要做到个人云盘与用户使用环境无缝连接，需要客户端挂载的远程文件系统能够兼容本地文件系统，这通常有两种解决方案，其一，需要客户端的宿主操作系统能够支持云盘挂载协议，对云盘系统定义的远程文件系统操作指令拥有一套兼容的实现模块。这使得个人云盘可以直接挂载到本地的文件系统上，支持以操作本地磁盘的方式操作个人云盘，其二，为客户端宿主机操作系统定制一个云盘内核模块，将用户对本地磁盘或者文件目录的操作定向到网络并发送给云盘系统所在服务器集群。以上两种方案注定使得客户端接入失去了良好的移植性，客户端对挂载技术的实现强依赖于操

作系统的文件系统模块，无论从开发难度还是跨平台角度考虑都不是一个好的选择。基于该原因，业界对云盘挂载技术的实现大多是面向协议层的，脱离了应用的业务层处理。该协议需要对文件系统操作的远程调用进行格式定义，尽可能的精简通信首部，在保证可靠性的前提下对文件系统的基本操作进行协议层定义，并对文件传输细节进行自动化的控制。

业界对云盘挂载技术有许多面向不同环境针对不同需求的实现，其中 Samba[17]、WebDAV 和 NFS[18]是广泛使用的三个技术。

Samba 是在 Unix like 系统上实现 SMB 协议的一个软件，SMB 协议（Server Messages Block）是一种在局域网内共享文件的一种应用层协议，它采用服务端/客户端的工作模式，Samba 作为一个实现了该协议的软件在 Linux 提供了服务端和客户端的软件包，在而在 Windows 系统上，只有 SMB 的客户端实现，该实现依赖于 IBM 开发的 NETBIOS[19]协议，一个典型的部署案例就是以 Linux 作为 SMB 服务端，Windows 作为 SMB 客户端，实现 Linux 上存储资源的共享。然而 SMB 协议只能在局域网下正常工作，导致公有云环境下无法方便的应用此技术。

WebDAV（Web-based Distributed Authoring and Versioning）是一种基于 HTTP 协议的通信协议，它扩展了 HTTP1.1 协议，使得客户端可以基于浏览器对 WebServer 进行文件操作，同时支持文件的加锁操作、权限控制和版本控制。基于 HTTP 使得 WebDAV 拥有极强的跨平台特性，B/S 的工作模式使得客户端的实现变得便捷而稳定，但同样因为这一点，要求服务器提供 HTTP 服务功能，即存在一个类似 Apache 的 HTTP 服务程序，在云盘系统中暴露出 80 端口增加了系统的危险性，同时 HTTP 服务程序也为服务节点增加了额外的负载。

NFS（Network File System）是一个使用 SunRPC 协议构造的网络文件系统，NFS 客户端通过向一个 NFS 服务器发出 SunRPC 请求来访问远程文件，NFS 的客户端需要操作系统级别的支持，在 Linux 下，NFS 作为一个特殊的文件系统内核模块挂载在 VFS[20]上，用户透明的对其进行文件操作，每一个操在经过 VFS 多态转换后会流向网络模块。早期版本的 NFS 只支持 UDP 作为其传输层协议，所有厂家都提供了这种实现，最新的一些实现开始支持 TCP，这使得 NFS 可以在广域网中应用，但由于其只在 Unix like 系统中普及，而大量用户依赖的 Windows 操作系统并没有 NFS 的实现，导致市面上的云盘服务提供商不使用 NFS 作为其远程文件系统操作协议。

2.3 本章小结

本章针对云盘系统的两个关键技术——Ceph 分布式文件系统和云盘挂载协议进行了研究和分析，首先介绍了 Ceph 分布式文件系统的设计优点，然后对 Ceph 组件进行详细介绍的同时对其总体架构和各个模块进行原理分析。最后介绍了云盘挂载协议在云盘系统中的角色和功能，并对其实现技术进行了简单介绍和优缺点描述。

第三章 CFSRPC 协议的研究与设计

本章对业界远程文件系统相关操作协议进行参考和对比分析，研究设计一套基于远程调用模式且适用于本文所述云盘系统的文件系统远程操作协议 CFSRPC（Ceph Filesystem Remote Procedure Call Protocol）。

3.1 远程文件系统通信协议应用场景和难点分析

云盘系统为用户提供的主要业务功能就是文件存储，公有云环境下的云盘服务需要支持大量用户对云盘文件系统操作的远程映射，用户在登陆成功后向云盘的存储接入节点发送个人云盘的挂载请求，并维持一个长连接以做到随时的远程操作。特定网络协议对用户远程操作的通信起着支撑作用，挂载操作、文件操作、目录操作等都基于该网络协议进行请求通信，为保证大量用户的体验，需要尽可能减少网络开销，这要求该协议在保证功能性和可靠性的前提下最大化精简性和目的唯一性。

3.1.1 应用场景

涉及到存储资源的云化管理，众多客户端对云端存储资源的使用必然是基于网络进行交互和 IO，通用文件操作的远程执行需要依赖于统一的调用方式和通信管道，远程文件系统通信协议为这一过程设定了统一的标准和规范，遵循这样的通信协议有助于云化资源访问方式的一致化和修改灵活性。

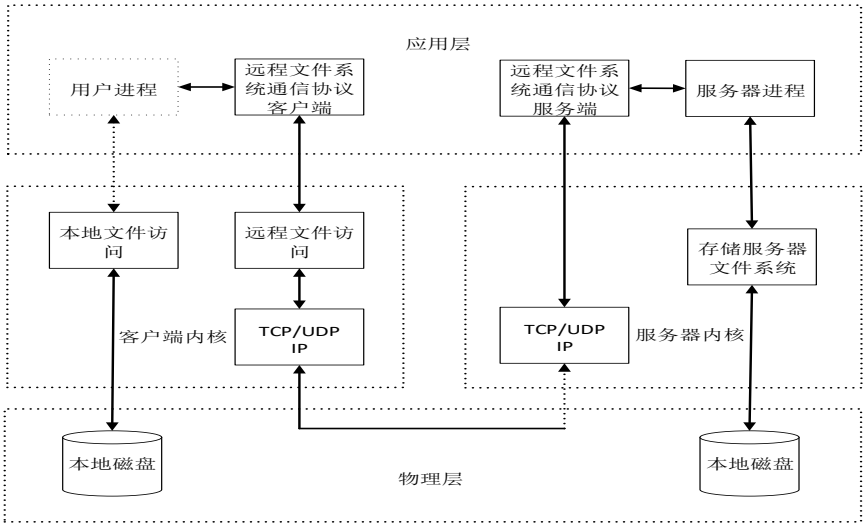

图 3-1 远程文件系统典型架构

远程文件系统[21]的典型架构如图 3-1 所示，客户端即用户进程在在访问本地文件时，经由本地操作系统的文件系统直接对本地磁盘进行 IO，当其访问远程文件系统上的资源时，远程文件系统通信协议介入工作，客户端进程包含该通信协议的客户端实现，将文件操作依据通信协议的规定进行处理和封装，然后经过本机的网络协议栈发送到公网上，服务器进程在接到这些文件操作的数据流后，会使用远程文件系统通信协议的服务端模块进行数据包的解析，解析结果对应于用户期望的具体文件操作，服务器进程根据自己的存储体系对文件进行定制化存储。返回结果也将在该通信协议的包装下进行传输。

基于网络的文件系统的数据流主要包括文件 IO 和元数据 IO，对应存储服务器的文件存储和用户空间元数据的修改。

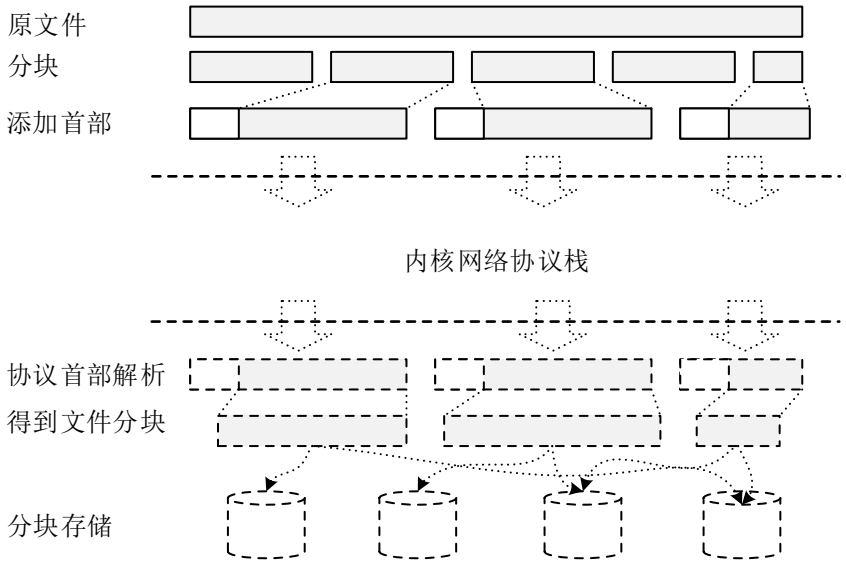

图 3-2 文件上传典型处理方式

以文件上传操作为例，其数据的传输过程如图 3-2 所示，用户将本地文件拷贝到客户端，客户端对文件进行方便传输的分块处理，并生成一个协议头部加入到每个分块前面，该头部里包含了文件上传指令以及文件大小等等操作参数，然后将每个分块写入网络套接字，数据流由此进入客户端的内核态。网络套接字的内核部分会将数据流向下送给网络协议栈，内核线程或中断例程会对数据流依次进行 TCP/UDP 分组、IP 分片[22]。链路层在接受到携带 IP 头部的数据报文后以网络环境协商好的 MMU 大小由以太网口发出，服务器的接入节点在接受到数据包后，自底向上分别经过网络层的分片重组和传输层的分组排序重组，最后用户态的服务进程以缓冲区的形式接受到数据流，应用层的客户端对数据流中的协议头进行解析，识别出一个文件的头包和尾包，根据解析结果中的文件上传操作指令，将完

整的文件数据流交给存储模块。

文件的其他操作涉及的数据流动原理跟文件上传类似，文件下载操作会带来反方向的大流量，而对于元数据的操作，除了加载巨大目录树以外，流量较小。

3.1.2 难点分析

1）远程文件系统操作的通信协议涉及到对事务的控制，任何一次操作的实施对应一个事务的开始，协议的设计需要考虑到事务的原子性、一致性、隔离性和持久性[23]。用户的操作之间的时隙往往大于一个事务完成的时间，协议需要保证单个事务处理和提交不会因为网络数据包的无序到达性而受到其他事务的影响。

2）该协议层的分块大小对用户体验以及存储服务器的性能有效和网络数据吞吐量起着关键决定作用，服务端在应用层需要在内存里为每个未完成的事务至少维护一个分块大小的缓冲区，因为当客户端与存储服务器进行一个超长调用，例如上传一个大文件时，由于公网环境的不稳定性，传输层连接极有可能中断，协议的服务端需要在此时对该调用进行小事务的拆分以保证之前的已到达数据不会因事务回滚而删除，文件分块在未接收完全时，存储节点是不会对其进行存储操作的，这就需要内存里缓冲区的支持。而分块大小同时直接影响着添加的协议头部字节数，间接影响了单位时间内网络吞吐的有效数据量。如何确定或者动态控制该分块大小是协议设计需要解决的难点，本章将在第四节设计并提出文件分块大小的动态调节机制。

3.2 CFSRPC 协议总体设计

业界内文件系统的远程使用通信协议使用比较广泛的有 NFS、SMB、WebDAV，WebDAV 由于基于 HTTP 协议需要在 CCD 上增加 HTTP 服务程序，为 CCD 节点增加了额外的开销，并且额外暴露的 web 访问端口也降低了 CCD 的安全性。SMB 协议基于 NetBIOS 导致其不支持 WAN，若使用增加 VPN 服务器的方式[22]使用 SMB 作为挂载协议增加了云盘系统的复杂性和不稳定性。NFS 基于 sunRPC 协议，其旧版本以 UDP 作为传输层协议，只适用于内网环境下的私有云存储，v3 以后的版本增加了对 TCP 协议的支持，但由于 NFS 的目的在将本地文件系统共享给客户端，且使用了大量的辅助协议，增加了个人云盘系统不需要的文件锁功能，不符合云盘接入节点的服务提供方式。

因此本节参考 NFS 的设计思路，针对本章第 1 节提出来的要点和难点，提出一个名为 CFSRPC（Ceph Filesystem Remote Procedure Call Protocol）的文件系统远程操作协议并进行总体设计。

3.2.1 总体结构定义

RPC[23]（Remote Procedure Call Protocol）远程调用协议是一种客户端通过一个本地的函数接口经由网络从远程服务器上请求服务的一种方式，一次 RPC 的调用过程分为以下几步：

1）客户端程序调用本地函数接口并发送用户态的请求参数，陷入等待。

2）函数接口对请求参数进行通信封装，由本地操作系统的内核进行网络 IO。

3）服务端接受到消息。

4）服务器守护进程对消息进行解析和处理，得到参数并进行业务操作。

5）服务器守护进程将操作结果返回客户端机器。

6）客户端进程接收到响应结果或响应数据流，结束等待。

CFSRPC 协议基于远程调用的模式和客户端单向请求服务端单向应答的模式，调用流向单一，服务端不会主动与客户端进行通信，因此协议的制定包括请求报文的头部和应答报文的头部，并基于请求应答的模式设计协议的信息交换机制。

CFSRPC 协议的通信策略基于 RPC 模型，这要求 CFSRPC 协议支持两个不同的结构：请求信息和应答信息，其协议首部分别表示为图 3-3 和图 3-4。

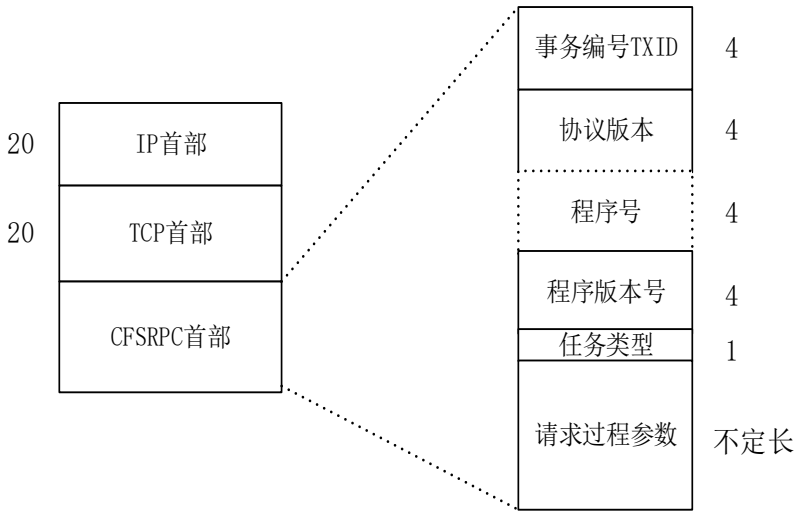

图 3-3 CFSRPC 协议请求报文首部结构

CFSRPC 协议的请求信息包含以下 4 个重要的数据字段，使得服务端程序能够对请求做出迅速的先行解析。

1）事务编号（TXID）：长度 4 个字节，作为该次远程调用事务的唯一标识，其作用和规则在下一节做详细描述。

2）协议版本（Protocol version）：表示客户端当前实现和使用的的 CFSRPC 协

议版本，长度为 4 个字节，以对每次调用过程的 CFSRPC 协议版本进行校验，不同版本的协议实现无法建立通信。

3）程序号（Program number）：表示服务端程序的唯一标识符，可以是程序名，也可以是程序型号。

4）程序版本号（Program version number）：表示服务端程序的当前版本号，与程序号一起作为服务端程序的唯一标识，服务端对二者的解析出现任何不匹配问题，将直接返回错误信息。

5）任务类型（Task type），长度 1 个字节，为了方便服务端进行在协议层区别任务类型，并对不同任务类型的请求进行资源的统筹安排和调度。

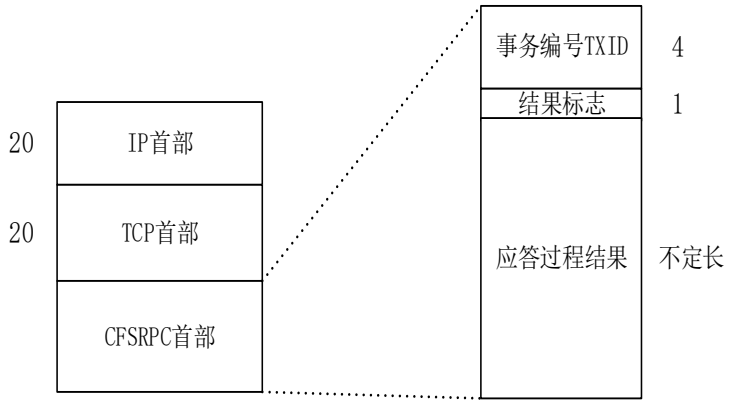

图 3-4 CFSRPC 协议应答报文首部结构

应答信息需要首先指明自身是针对哪个一个事务进行的应答，同时需要对调用过程是否成功做出回应，如果调用失败，需要指出失败原因，因此其格式至少包含结果标志（Result），用该值表示调用成功和调用失败具体原因。

CFSRPC 协议的目的是让客户端远程使用云盘服务，包括挂载、文件操作、目录操作的业务细节需要直接在协议中给出，因此请求信息和应答信息需要包含远程调用的过程参数和过程结果，以文件上传为例，过程参数需要表明过程类型（文件上传、文件下载等）、用户名、文件名、文件大小、文件路径等元数据，过程结果需要表明过程操作的结果。

3.2.2 事务机制

事务是业务执行的一个单位，在并发处理的环境和复杂的网络 IO 环境中，业务流程的事务化可以实现业务的阶段持久化功能，意味着能够为文件上传下载操作的可打断性提供协议级别的支持，且事务的隔离性使得单个客户端的云盘操作不会受到自身连接之前操作的影响，原子性使得一个用户的每一个远程调用操作

能够互不干扰的执行，以及针对 IO 型任务的序列化执行。

根据本章第一节对事务和文件分块的描述，当远程调用过程含有文件数据的远程 IO 时，文件的每一个分块的上传或下载对应单个事务的生存周期，以文件上传为例，上传的文件在应用层进行分块，客户端程序为每个分块添加一个 CFSRPC 的头部，即在合适的过程参数后添加文件分块的数据信息，然后将该添加了协议头部的分块沿着内核网络协议栈向下传输，服务端应用层在接受到该分块后，会对其进行协议解析，在程序号、程序版本号、过程号等信息验证通过后会取出所有过程参数，在过程参数里确定该次远程调用的过程编号，即指定远程调用的具体函数，接着服务器进程使用各种并发机制，将该文件 IO 任务指派给一个执行体，文件分块经由网络到达存储接入节点，当分块的数据传输完整且完毕时，服务器进程会向客户端发送一个结果报文，返回事务结束的应答信息。每一次远程调用需要包含一个事务编号以确保客户端或服务端再次接受到因为 TCP 超时重传机制导致的游离报文时做出丢弃处理。

图 3-3 和图 3-4 表示了 CFSRPC 协议的请求消息的首部和应答消息的首部，其中的第一个字段是事务编号 TXID（Transcation ID），用来实现事务的控制。该字段以 4 字节即 32 位的空间做载体，对用户的一次登录会话，最大可支持 4,294,967,296 个事务的同时触发，用户的一次远程调用操作对应一个事务编号，之后触发的操作会在上一个事务编号上做 1 的累加以做区分，服务端在解析到一个操作对应的协议首部后，会识别其事务编号，若小于上一次的编号或者并非累加的编号，则丢弃，若当前该用户正在处理一个 IO 型任务的同时再次接到拥有合法事务编号的 IO 型任务，则对该 IO 型任务进行排队处理，非 IO 型任务可以允许并发执行。文件传输会涉及到对文件的分块，每一个文件分块的传输都定义为一次新的事务，文件的第一个分块的事务被处理后，需要服务器进程将当前事务编号调整为加上该文件的分块个数后的编号，意味着该文件所有分块的到达事务都被认为合法且加入 IO 任务队列。这样的机制使得 IO 型任务和非 IO 型任务都能够得到合理的响应速度，同时维持了云盘操作事务化带来的功能性优点。

3.2.3 支撑协议

CFSRPC 协议基于远程调用的模式，为客户端程序和服务端程序提供通用的函数调用接口标准。云盘系统作为公有云服务系统，客户端与存储服务器的通信基于整个互联网环境，传输层协议的选择对 CFSRPC 的性能和可靠性有很大的影响，TCP 协议是面向连接的可靠协议，有流量控制和差错控制的功能，也正因为这些功能的存在，协议头部较大，传输效率较低。UDP 协议基于无连接的不可靠协议，

对传输的细节没有控制手段，简单的头部仅仅支持对发送报文的目的地设置，因此效率比 TCP 要高，在确定网络环境很好的前提下，UDP 可以比 TCP 提供更大的有效数据吞吐量，但对准确性要求较高的场景，TCP 在传输层提供可靠性的保证，为应用层屏蔽传输细节，让应用层可以专注自身业务的处理。CFSRPC 作为远程文件系统操作协议，需要为客户端提供可靠的文件传输服务和元数据的操作服务，公网环境的复杂性和不确定性使得丢包和数据流无序不可避免，因此为了在保证调用的可靠性的同时最大化网络带宽利用率，选择 TCP 协议作为 CFSRPC 的传输层支撑协议，以保证 CFSRPC 传输过程的正确性和合法性，其强大的流量控制和拥塞控制功能能够很好的处理互联网的不确定网路问题。

当今绝大多数操作系统的网络协议栈的实现都成熟可靠，为保证 CFSRPC 协议的简洁和功能唯一性，将其设计为应用层协议，基于 TCP 协议使得 CFSRPC 在内核态就解决了文件操作事务的无序问题和文件数据流的完整性和一致性验证问题，CFSRPC 可以专注于文件系统操作远程调用的过程控制和过程指令定义，并处理好应用层的数据分块问题。

3.3 CFSRPC 协议细节定义

3.3.1 请求过程参数格式定义

3.3.1.1 过程参数长度

用户的文件操作对应的每一个事务都会且仅产生一个 CFSRPC 头部，TCP 协议能够确保该头部所在报文完整有序的被服务端的应用层接收。因此，过程参数部只需要包括一次完整事务所需要涉及到的参数，针对每一个远程文件操作都需要有对应的唯一操作指令代码。不同的文件或目录操作存在相对统一和相对不统一的参数类型和参数长度，相对不统一的参数例如文件上传操作需要在过程参数里指定文件名、文件上传的目录、文件大小等元数据信息，目录创建操作需要在过程参数里指定目录的名字、上级目录甚至目录树的整体结构，相对统一的参数例如几乎所有远程文件操作都需要客户端在过程参数里携带用户的认证令牌，服务端需要在执行每一个事务之前对用户进行身份和权限认证。因此，请求过程参数由于过程的多样化和参数列表的不统一性，其长度无法由协议定义，需要作为协议首部的字段在交互过程中指明。

3.3.1.2 操作指令定义

云盘用户的远程操作方式集合需要涵盖一个通用文件系统的习惯性操作，CFSRPC 协议需要在过程参数部分给出操作指令的唯一表示编码和每个操作的参数结构定义，协议按照小端字节序的方向定义每个数据位，表 3-1 协议定义了不同操作指令编码方式，服务端程序按照机器内存模型的字节序[27]逆序寻找第一个置

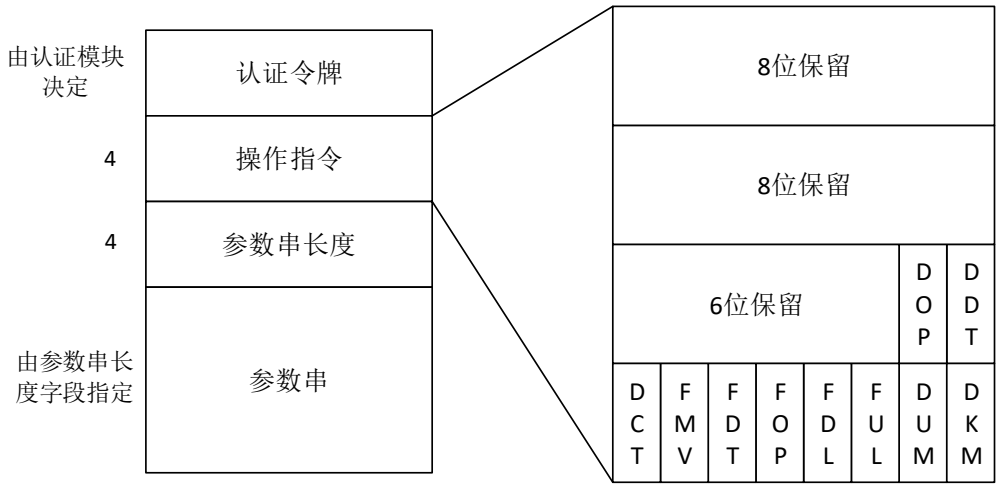

图 3-5 请求过程参数格式

1 的位置，然后按照表中定义规则进行操作指令的确定，例如客户端进行云盘挂载操作，将协议头部的操作指令部分的第一个字节的第 8 位置 1，服务端在接收到该报文后读出该操作指令的 4 个字节，按序读取每一位，读到第一位是发现被置 1，继而确定该次调用为云盘挂载操作。按照此模式，每个字节可表示 8 个操作指令，考虑到文件系统操作的有限性，4 个字节 32 位可以满足要求。存储协议首部请求过程参数的格式如图 3-5 所示。

不同操作指令的参数不同，因此将参数部分设计为参数串长度字段和参数串本身，参数串以 Key-Value 的形式做字符串化处理，具体按照标准 json[28]格式进行组织，每一对键值表示参数名和参数内容，例如以下请求调用参数：

```
1  {
2      “filename”: “music”,
3      “filesize”: “6291456”
4  }
```

表示此次调用的参数为：文件名 music，文件大小 6MB，最后对参数内容做对齐处理后存入协议首部。

CFSRPC 协议初始版本对云盘操作功能的支持以及请求参数的相关定义以表

3-1 的形式呈现，其中包括请求过程参数的功能定义和对应操作指令的编码定义。

表 3-1 请求过程操作指令定义

操作指令	编程命名（简称）	协议编码	功能定义	过程参数格式
云盘挂载	DISKMOUNT（DKM）	0x00000001	将用户个人空间挂载到本地客户端，表现为用户个人空间的根目录在客户端界面以通用文件系统的形式呈现，主要是对用户的认证，不包含文件的传输	无
云盘卸载	DISKUNMOUNT（DUM）	0x00000002	在用户在客户端主动断开云盘的操作连接并通知服务器端回收连接资源	无

上传文件	FILEUPLOAD（FUL）	0x00000004	用户新建或者上传一个新的文件，客户端对文件进行分块，然后针对每个分块顺序进行上传文件事务的调用，文件的具体数据接合在参数串的后面	{
				“filename”:
				“文件全路径名”,
				“filetype”:
				“文件类型”,
				“chunksize”:
				文件分块大小
				}

下载文件	FILEDOWNLOAD（FDL）	0x00000008	用户将文件下载到本地磁盘，需要向服务端发送下载文件指令，请求过程不涉及到文	{ “filename”: “文件全路径名”

			件传输，服务器的应答报文对文件传输进行控制。	}

打开文件	FILEOPEN（FOP）	0x00000010	用户直接通过客户端打开文件，只支持小文件的打开操作，大文件需要先行下载，客户端将文件数据读入内存中，然后传入打开文件的特定工具	{
				“filename”:
				“文件全路径名”
				}

删除文件	FILEDELETE（FDT）	0x00000020	彻底删除一个文件包含两个阶段，服务器端会对回收站的文件进行彻底删除，普通路径的文件则进行移入回收站的操作	{
				“filename”:
				“文件全路径名”
				}

移动文件	FILEMOVE（FMV）	0x00000040	移动文件只是改变一个文件的所属目录，不做任何文件数据传输	{
				“filename”:
				“文件全路径名”,
				“destiny”:
				“目的路径”
				}

新建文件夹	DIRECTORYCREATE（DCT）	0x00000080	在云盘目录下新建一个文件夹，没有具体数据的存储，目录的修改往往只是对应云盘元数据的修改	{
				“directoryname”:
				“文件夹全路径名”
				}

删除文件夹	DIRECTORYDELETE（DDT）	0x00000100	在用户个人空间中即时删除一个目录，原理跟新建文件夹一样，只是对元数据的修改	{
				“directoryname”:
				“文件夹全路径名”
				}

打开文件夹	DIRECTORYOPEN（DOP）	0x00000200	用户对服务端请求一个具体目录的下一级结构，服务器端需要查询元数据存储节点得到该文件夹下一级的所有文件系统节点	{
				“directoryname”:
				“文件夹全路径名”
				}

CFSRPC 协议允许添加新的操作指令，只需在协议的请求过程参数部分的操作指令定义部分按需找到第一个未定义的数据位，然后定义该位代表的远程操作详情即可。实现部分要求新增该操作的解析流程和处理流程，并定义其请求过程参数格式和应答过程参数格式。

3.3.2 应答过程结果格式定义

服务端对客户端的应答过程也包括 RPC 部分和过程结果部分，过程结果针对不同的请求过程会对应不同的任务类型，例如文件下载或者文件打开操作，过程结果包含反向的 IO 任务，文件分块通过服务器端向客户端发送。针对云盘挂载操作和目录打开操作，服务端会返回当前目录结构及根目录下的节点信息或打开的文件夹内的所有文件信息，由于该元数据大小与文件不是一个数量级，因此将其以数组的形式存储应答报文的的结果串中，同样以 json 格式进行存储。应答过程的结果格式与请求过程参数结构大概一致，去除认证令牌和操作指令部分，需要指明的是结果串长度和结果串本身，如图 3-6 所示。

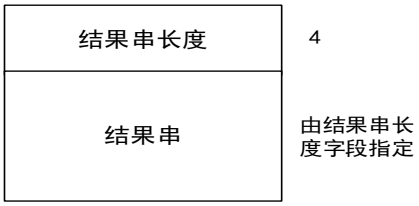

图 3-6 应答过程结果格式

由于 TCP 是全双工协议，反向的文件 IO 可以使用同一个连接进行传输，这要求服务端实现文件分块的发送流程和客户端实现文件分块的接受和处理流程，文件分块作为一次事务的数据部分跟在结果串字段后面。针对每个不同的云盘操作，结果串会有不同的格式和内容，以下对 CFSRPC 初始版本的 10 个远程云盘操作的应答过程结果的格式做出定义。

云盘挂载、打开文件夹，以上 2 种云盘操作服务端需要返回用户个人空间的某一个目录下的文件和文件夹列表包括其每个文件的类型和大小，格式一致如下：

```
{
 "file" : [
    {
     "filename" : "文件名",
     "filesize" :  文件大小(数值类型),
     "filetype" : "文件类型"
    }, 文件数组],
 "directory" : [
    {
       "directoryname":"文件夹名"
    }, 文件夹数组]
}
```

云盘卸载、移动文件、删除文件、新建文件夹、删除文件夹，以上 6 种操作指令，服务端只需要告知操作成功和操作失败的原因，格式一致如下：

```
{
    “result” : 0/1,   0 表示成功，1 表示失败
    “reason” : “失败原因”
}
```

下载文件、打开文件，以上 2 种操作指令，服务端需要向客户端返回文件分块，因此需定义分块大小，格式一致如下：

```
{  “chunksize” : ”分块大小”  }
```

上传文件，客户端需要知道该次事务是否成功，服务端需要对上传文件的文件分块大小做动态调整，因此需定义结果标志、操作失败原因和分块大小，格式如下：

```
1    {
2        “result” : 0/1,   0 表示成功，1 表示失败
3        “reason” : “失败原因”,
4        “chunksize” : “服务器端对分块大小的调整值”
5    }
```

3.4 文件分块大小动态调节机制

3.4.1 文件分块大小的影响效应

云盘系统的网络数据吞吐量主要由云盘文件的 IO 组成，文件分块越大，单位体积的文件分块数量越少，需要的协议头部个数越少，增加了有效数据占网络流量的比例，同时由于每一个文件分块的传输对应一个远程调用事务，较少的文件分块数量使得一个文件传输对应的事务执行量相对较少，减轻了服务器存储接入节点对事务排队和调度的资源开销。相反，文件分块越小，则减小了有效数据的带宽利用率以及增加了事务排队和调度开销，但由于服务器需要为每一个文件 IO 型任务维持一个该次事务的文件分块大小的内存区域，当文件分块很大且请求压力巨大时，个别用户的巨大 IO 业务会占用很大的内存资源，这会导致其他用户的饥饿，因此越小的文件分块对应着这个内存区域越小，当用户并发数量巨大时，可以做到对每个用户业务处理的资源的均衡分配。云盘的存储机制往往会对文件进行分块存储和分布，因此每次事务传输的文件分块大小要大于云盘存储机制设定的存储分块大小，以保证服务器每接受一个事务带来的文件分块数据就能够接着执行下层的存储策略。

以上分析可以看出文件的分块大小对云盘系统的多方面功能和性能有着极为重要的影响，但由于对文件分块大小的调整涉及到多方面或优或劣的变化，对文件分块大小作硬性的规定或指定无疑会在某些情景或在某些运行环境下导致云盘系统的一个或多个性能瓶颈，因此，文件分块大小的动态调节机制对于云盘系统的稳定高效运行是必不可少的。

为使得 CFSRPC 对文件分块的设定和动态调整提供协议级别的支持，本节对分块大小的动态调节算法进行分析和设计，命其名为 CSDA 算法（Chunk Size Dynamic Adjust）。

3.4.2 CSDA算法设计

研究设计 CSDA 算法的目的是为存储接入节点即 CFSRPC 协议的服务端求得合适的发送文件分块大小和接收文件分块大小，涉及到双向的文件传输，以保证避免服务端的文件发送或接收 IO 的拥塞情况，同时保证对所有用户请求事务和响应事务的均衡调度。

定义算法的输出为文件上传请求过程的文件分块大小参数和文件下载应答过程的文件分块大小参数，对于服务器端来说，前者为网络数据的接受，后者为网络数据的发送，因此设二者分别为 FCS_{in} 和 FCS_{out}。考虑到底层存储分块大小由存储机制设定，因此作为输入参数之一，以 SCS 常量表示。算法决策过程中需要考虑到机器的某些指定负载因子情况，因此引入存储接入节点的负载指标 Load，而发送文件和接收文件对应的负载指标依赖的负载因子不同，因此区分为 $Load_{in}$ 和 $Load_{out}$ 分别表示计算 FCS_{in} 和 FCS_{out} 所使用的负载指标，其计算公式见公式 3-1 和公式 3-2。

$$\text{Load}_{\text{in}} = \begin{cases} 1, & QC_{in} \le T_{max} \\ 1+QK_{in}, & QC_{in} > T_{max} \cap QK_{in} > 0.1 \end{cases} \quad (3\text{-}1)$$

$$\text{Load}_{\text{out}} = \begin{cases} 1, & QC_{out} \le T_{max} \\ 1+QK_{out}, & QC_{out} > T_{max} \cap QK_{out} > 0.1 \end{cases} \quad (3\text{-}2)$$

其中，QC_{in} 和 QC_{out} 分别表示 CFSRPC 协议的服务端所在节点对文件 IO 型事务的接受处理队列长度和发送处理队列长度，能够表示当前存储接入节点的任务负载情况，QK_{in} 和 QK_{out} 分别表示文件 IO 型事务的接受处理队列长度的增长斜率和发送处理队列长度的增长斜率，能够反映当前存储接入节点的任务处理能力和任务过载情况是否发生。T_{max} 表示当前服务节点的 CPU 支持的最大线程数量[28]，当事务等待队列的长度小于 CPU 支持的最大线程数时，说明队列里的事务可以在任何一个线程处理完当前事务后得到执行，因此将负载指标设为 1，表示负载状况良好无须对文件分块大小做出调整。单位时间内队列长度的增长斜率大于 0.1 表示其队列长度处于增长趋势，当队列长度大于 T_{max} 且处于持续增长趋势的时候，表示服务器处于高压环境，用户的业务请求不能得到及时执行，需要增加负载指标值，以减小文件分块大小，从而减小每个事务的处理时间，进而将每个用户的任务细分以方便并行处理。

CFSRPC 协议将文件分块大小的初始值定为底层存储机制的对象大小，即 SCS，且文件分块大小在动态调节过程中，最小值不得小于 SCS 指定的值，以做到协议向下的兼容性。文件分块大小的计算公式见公式 3-3 和公式 3-4。

$$FCS_{in} = \begin{cases} CUR_{in}, & Load_{in} = 1 \\ CUR_{in} \div 2, & Load_{in} > 1 \cap \ CUR_{in} \geq 2 \times SCS \\ SCS, & Load_{in} > 1 \cap \ CUR_{in} < 2 \times SCS \end{cases} \quad (3\text{-}3)$$

$$FCS_{out} = \begin{cases} CUR_{out}, & Load_{out} = 1 \\ CUR_{out} \div 2, & Load_{out} > 1 \cap \ CUR_{in} \geq 2 \times SCS \\ SCS, & Load_{out} > 1 \cap CUR_{in} < 2 \times SCS \end{cases} \quad (3\text{-}4)$$

其中 CUR_{in} 和 CUR_{out} 分别表示文件接收事务的当前文件分块大小和文件发送事务的当前文件分块大小，当负载指标为 1 时，表示当前节点的负载状态正常，所有用户的事务能够及时得到处理，文件分块大小保持一致，当负载指标大于 1 时，表示当前节点处于过载状态，因此在保证文件分块大小最小不少于 SCS 的情况下，对其进行减半操作，若减半后文件分块大小小于 SCS，则将其大小定为 SCS。该计算过程只考虑了过载情况下对文件分块大小的裁剪，而未考虑当服务器空闲时对其进行扩大操作以增加网络带宽的有效利用率，因此将该机制设计为当 Load 值在一段时间内保持为 1 且 QC 的值小于 T_{max} 的话，将 FCS 的值增加 SCS 的大小，即当负载指标变化且用户事务不存在拥挤阻塞情况时，将文件分块大小增加底层存储系统的对象大小。这使得 FCS 的大小根据 SCS 大小为标准保持数据对齐，方便后续的存储操作。

整个 CSDA 算法根据多方面参数的变化值计算得到当前最合适的文件分块大小，综合 Load 负载指标和 FCS 的计算公式以及 FCS 的增量机制，以伪代码的形式描述整体计算过程如下：

```
1   cur, qc, qk, tmax,scs
2   if (qc > tmax && qk > 0.1 )
3       load = 1 + qk
4   else load = 1
5   if (load == 1)
6       fcs = cur;
7   else if (load > 1 && cur >= 2 * scs)
8       fcs = cur / 2
9   else if (load > 1 && cur < 2 * scs)
10      fcs = scs
11  if (time(load stay 0 && qc <= tmax) > 2 sec)
12      fcs += scs
13  return fcs
```

输入参数 cur、qc、qk、tmax、scs 依次分别表示当前文件分块大小、事务队列长度、事务队列长度增加斜率、CPU 支持最大线程数、底层存储系统的存储对象大小。返回的 fcs 即为 CSRPC 协议协商的文件分块大小。

3.5 本章小结

本章对云盘系统的远程文件系统操作机制的原理和技术进行了分析，研究并设计出 CFSRPC 协议，使之能够满足远程文件系统操作的严格格式化即协议化，并给出了 CFSRPC 协议的客户端和服务端的具体格式定义和二者的通信机制，云盘系统通过对该协议的实现，可以透明化远程文件操作的细节，方便客户端和服务端上层业务的实现。本章最后为该协议设计了事务机制用于颗粒化文件传输服务，同时设计 CSDA 算法在协议级别动态调节文件传输的分块大小，增加了该协议对云盘系统性能调优的能力。

第四章 云盘系统核心模块分析与设计

信息化的大环境下，个人云盘的用户群体不断增长，个人用户数据量急速膨胀的同时，用户对数据可靠性和访问安全性意识也在不断增强。公有云环境下的云盘系统在提供用户认证、文件读写、目录管理等基础功能的同时，需要在庞大数据储量和计算量下保证每个用户数据的可靠性和一致性，在高并发访问量下保证每个用户读写个人文件和共享文件的流畅性和安全性。为了达到这样的目的，
本章通过整合云盘系统所需的几个核心模块，研究并提出云盘系统从底层到上层的整套解决方案。

4.1 云盘系统需求分析

云盘系统是一套涉及核心存储、用户认证、挂载方式、通信协议、可视化等多个技术点的复杂系统，为了保证系统的可实现和可扩展，本节对该系统从功能和性能两个方面进行需求分析。

4.1.1 功能需求分析

1）用户认证

根据使用流程，云盘系统首先需要拥有可靠安全的用户认证机制，从前端基本的用户名密码登陆机制到后台的独立认证模块，在保证认证速度的前提下，认证流程需要能够在用户安全登陆后，在整个云盘操作过程中安全持有使用资格，实现用户操作个人数据的单点认证功能，该认证模块需要集中存储和处理用户唯一标识符（加密的账号密码、指纹 Code、认证令牌[29]等），同时保证认证的速度并对用户的认证地点进行物理限制（IP、MAC 地址等）。

2）文件操作

云盘系统的客户端需要实现对用户个人文件的基本增删改操作，操作习惯基本符合主流操作系统的文件系统（例如 windows 的 explorer），在云盘内新建文件对应文件上传操作，删除文件对应将文件移至回收站，上传云盘内已有的文件，需要云盘系统能够自动识别，继而简化上传过程，加速上传速度。

3）目录管理

用户对个人数据的整理，往往是基于目录的组织结构，传统文件系统的可视化操作影响了多数用户的操作使用习惯，目录的可视化形态和实际数据结构往往是不一样的，云盘系统需要为每一个用户提供一个存储根目录，作为用户自行组织的

树状目录结构的根目录，用户可以以操作文件夹的方式来组织自己的目录结构。云盘需要能够可靠的保存并维护这样的目录树。用户通过客户端挂载云盘时，服务器端需要返回根目录下的文件结构，打开某个文件夹时，服务器端需要返回该文件夹内的文件结构，并在客户端内以文件列表的形式呈现出来。

4）数据一致

终端设备的多样化使得现今用户对多平台客户端的需求越来越大，因此，同一个用户在不同的终端上使用云盘服务的需求也越来越多，云盘系统需要满足当用户使用一台终端设备上传文件或者修改文件后，在一定的延迟时隙内，文件改变在其他终端上可见。

5）存储资源可视化

云盘系统对内需要提供存储集群的资源监控，系统的维护者或者管理员需要能够实时的看到底层存储集群的存储资源使用量以及网络 IO 情况，对外需要为每个用户提供自己数据存储量、文件个数、目录结构等数据的展示功能。

4.1.2 性能需求分析

1）安全性

用户的个人数据的存储和传输要足够安全，云盘的认证模块需要保证在用户在认证成功后的每一步操作的安全性，认证流程需要在保证性能的前提下穿插在之后的大部分文件操作前，文件的数据传输发生在存储集群和用户终端之间，要保证其传输过程的安全性。

2）可靠性

用户的个人数据以各种形式和布局存储在云盘的核心存储集群里，为保证每一个文件的可靠性和一致性，云盘系统需要对每一个唯一的文件提供冗余存储，备份块存储的物理位置需要根据机房的容灾规划进行定制性选择。同时，当存储节点宕机时，能够进行数据的热迁移，保证数据备份数量处于长期的健康状态。

3）高并发

公有云环境下的云盘系统必然需要应对大量用户同时使用的压力，因此，云盘系统需要能够在高并发的环境下，保证每一个用户的操作体验，在网络环境稳定的条件下，需要保证同一个文件在大量用户下载时能够拥有足够多的备份访问点，用户终端和服务器的链接能够做到在多台接入机器上的负载均衡。

4）扩展性

用户数量的不断提升，用户数据的不断增加，导致云盘系统底层存储资源的不足，因此需要为存储集群提供热扩容的功能，云盘系统的核心存储模块需要能够随

时增加新的存储节点甚至访问节点，以实现云盘系统存储容量的动态扩展。

4.2 云盘系统总体设计

4.2.1 功能架构设计

基于云盘系统使用过程的时间轴，核心业务流程在客户端表现为登陆、文件/目录操作、文件传输、存储资源查看，在服务端表现为用户认证、操作授权、分布式数据存储、存储资源监控。

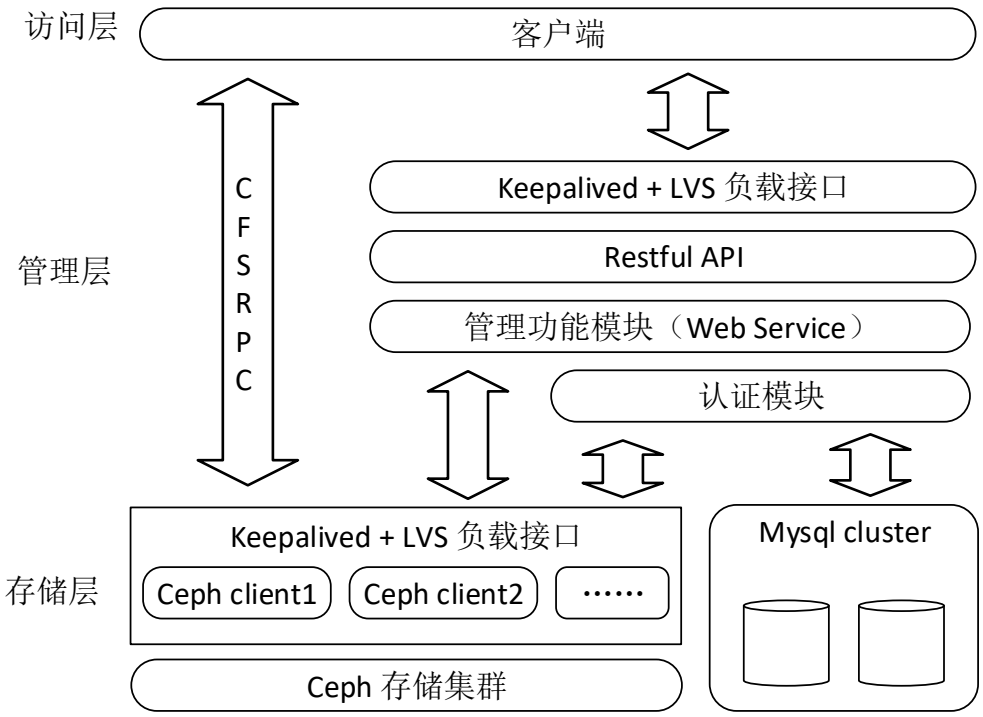

图 4-1 云盘系统架构

基于以上关键业务流程，考虑压力环境下的负载问题，以 Ceph 分布式文件系统做存储支撑，客户端通过 CFSRPC 协议与 Ceph Client 进行远程文件操作的调用，通过集中式认证模块进行用户的统一管理和权限认证，设计整体系统架构如图 4-1 所示。

系统架构分为访问层、管理层和持久化层，访问层是用户的操作接口。管理层是所有业务功能模块以及独立认证模块为访问层提供远程调用或者请求应答的服务层，用户的操作请求经由访问层的不同类型客户端以统一的 Restful API[30]的调用格式发送到管理层的业务功能模块，请求会话在经过负载机制的分发后到达业

务处理节点，每个节点提供一致的认证和服务程序。在用户的第一次登陆会话中，首先，业务功能模块接收到用户的用户名、密码等认证信息并转发向认证模块发起认证，认证通过后，业务处理模块将对用户的云盘空间进行初始化操作，包括在Ceph Client 里新建用户个人目录、配额设置、回收站空间设置等等。在用户云盘空间初始化完毕后，客户端会向 Ceph 存储集群的 client 节点请求挂载云盘，经由挂载协议完成客户端到存储层的远程挂载。业务功能模块和认证模块使用存储层的Mysql cluster[31]数据库存储用户的个人信息、权限信息以及云盘的配置信息。

该架构隔离了云盘的管理数据和存储数据，使得 Ceph 存储集群的功能性保持唯一，客户端可以通过基于 WAN 的挂载协议直接挂载云盘到本地，存储集群能够直接与客户端进行文件传输而不需要中间层会话的支持。认证模块位于管理层，向上为业务功能模块提供用户登录集中认证和用户信息的统一管理，向下为存储集群提供挂载操作、数据传输的权限认证。业务管理模块提供 Restful API 格式的服务接口，将用户操作的服务以资源的形式呈现出来，统一的调用接口支持多种客户端的接入。

4.2.2 负载架构设计

云存储是以集群技术作为其后台支撑的，为了能让集群节点的资源发挥最大效能，负载均衡技术的应用是必不可少的。将来自客户端的大量并发访问请求依据系统内部各个节点的负载情况通过一定的调度分配策略，经过代理器节点或者负载均衡接入节点，将这些请求分发到具体的计算或存储节点进行处理，以最大化利用节点资源，同时为云盘系统的服务节点的横向扩展提供了简便高效的接入方式。

公有云系统面向的受众是整个互联网的使用者，大量的操作请求汇聚在管理层的入口会给业务模块的单点宿主机带来极大的拥塞工况，从而产生系统的性能瓶颈，因此业务请求的处理服务需要负载均衡机制的支持，LVS 作为虚拟服务器系统能够在 Unix/Linux 环境下提供稳定高性能的负载均衡功能，云盘系统的线上环境保证各个节点在同一个网络内，因此使用 LVS（Linux Virtual Server）的 DR（Direct Routing）调度模式[32]，用户经由客户端发出操作请求后，请求汇入 LVS 的接入节点，LVS 可以基于 IP 层或传输层对用户的网络连接进行均衡分发，业务服务节点作为 LVS 接入服务器的代理对象会接受到转发而来的网络报文，而该调度过程作为网络层的包转发对于业务服务模块的应用程序是透明的，使得该部分的程序与负载机制没有耦合关系，纯粹面向客户端。单点的接入服务器必然会引入单点失效

问题，Keepalived 基于 VRRP 协议可以有效的避免单点故障，引入 Master 和 Backup 两个主备 LVS 接入服务器使得接入点可以在当前接入服务失效后自动进行 IP 漂移。Keepalived + LVS 的负载结构如图 4-2 所示。

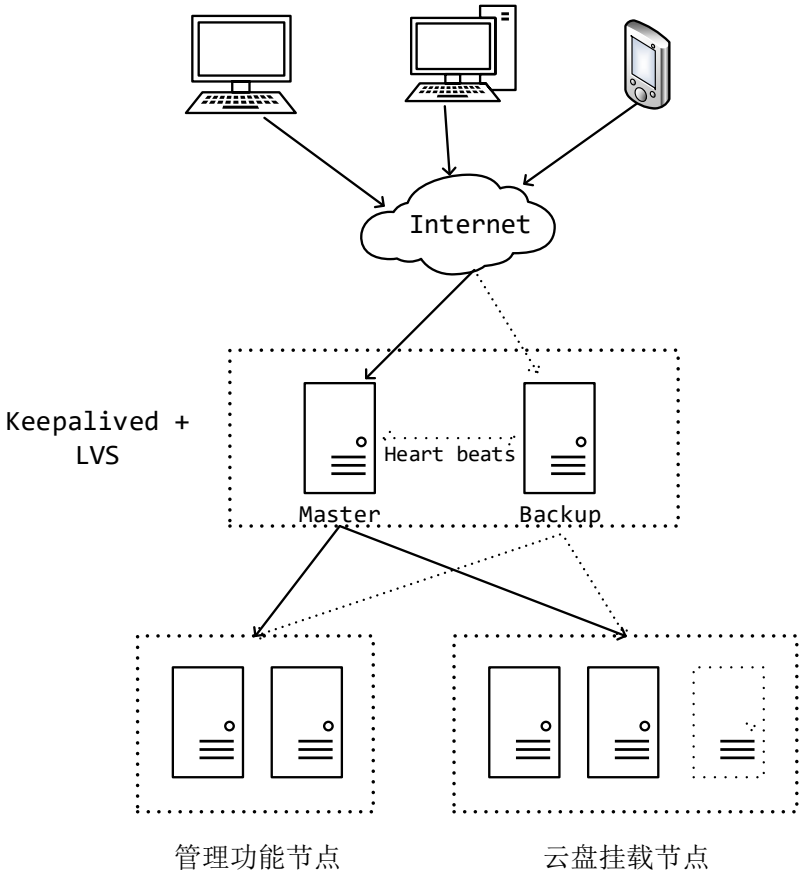

图 4-2 负载结构

在存储层，用户的挂载情景通常是客户端与 Ceph 某一个挂载节点维持一个长连接，单个挂载点同样无法满足高压环境下的用户需求，负载均衡机制必不可少。

LVS 支持在传输层对不同端口号的转发模式，因此管理服务和云盘挂载服务可以使用同一套接入服务器，接入层将 80 端口的网络流在管理功能节点内进行分发，而对于 CFSRPC 协议规定的请求则在云盘接入节点内进行分发。基于这个原因，管理功能节点是可以与云盘挂载节点合并在一个节点中提供服务的，但是由于管理业务功能的请求量与云盘操作的请求量不是一个量级，如果作为一个组件提供服务会导致功能复杂且对其进行扩展时会产生回归性能冗余。

4.3 核心存储模块设计

4.3.1 工作模式设计

云盘系统基于 Ceph 分布式文件系统实现核心存储模块，在整个系统中与其交互的角色包括访问层的客户端和管理层的业务功能模块，客户端在认证通过后使用 CFSRPC 协议将 ceph client 的个人目录挂载到本地，表现为在客户端可视化组件里呈现一个易于操作的文件系统。业务功能模块在执行用户个人空间初始化、查询存储资源或者设置云盘配额等例程时需要向 ceph client 请求对应操作。因此 ceph client 需要运行一个后台进程，监听客户端的挂载请求以及管理层的操作指令，这里称该进程为 CCD（ceph client daemon）。为实现 CCD 的效能最大化和易扩展，多个 CCD 需要能够同时运行，同时接收负载均衡接口服务器分发的网络请求。

Ceph 分布式文件系统的存储体系是基于自己的 OSD 集群向上提供对象存储服

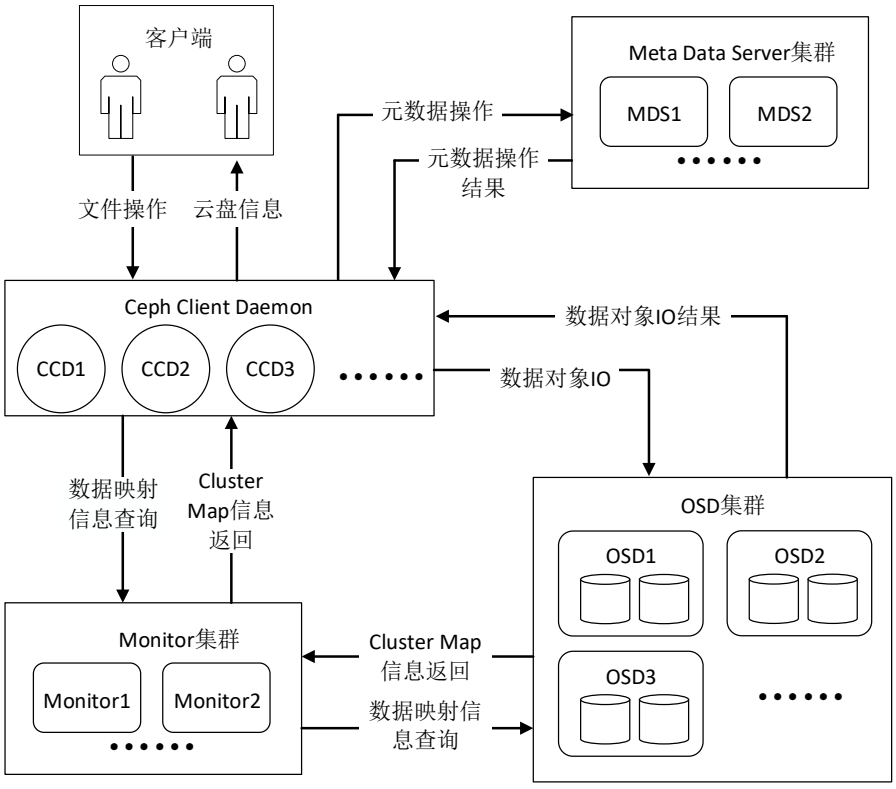

图 4-3 核心存储模块间数据流动

务，集群每个节点直接对物理存储设备进行管理，并能够做到自主修复和自主管理。Monitor 是 Ceph 用与维护集群拓扑信息和数据映射表的重要组件，Monitor 也以集群的形式保证这些数据的高可用性，CCD 与 OSD 集群在做数据操作的过程中需要

随时与 Monitor 集群进行通信，其整体数据流如图 4-3 所示。

4.3.2 存储功能设计

客户端通过 CFSRPC 协议直接与 CCD 进行通信，在管理层模块完成用户云盘空间的初始化后，客户端向 CCD 发送挂载请求，挂载成功后，客户端可以以通用的文件系统的操作方式对云盘进行查看和操作，包括对文件或目录进行创建、修改、删除，以及删除文件的恢复功能即回收站机制。

1）文件创建

客户端的文件创建操作经由 CFSRPC 协议反映到某一个 CCD 节点上，CCD 先向 Monitor 集群查询最新 cluster map 以获得最新的 OSD 集群拓扑、PG 信息、CRUSH 规则集合，接着在本地节点上对文件进行分块操作和 CRUSH 映射位置计算，得到每个分块即数据对象的 PG 信息和每个 PG 的位置信息，最后根据这些信息将每个数据对象写入对应的 OSD 节点，当 OSD 集群将所有该文件拆分的数据对象及其备份写入对应 OSD 节点的物理存储设备后，将此次存储过程的结果返回 CCD，CCD 得知文件存储成功就将文件的元数据（拥有者、权限、大小、修改时间等等）存入 MDS 集群，当 CCD 再次接受到 MDS 返回的元数据存储成功的信息后就将文件创建成功的信息返回给客户端。至此一个文件的创建和上传流程结束。

2）文件删除

CCD 接受到客户端的删除文件请求后，会向 MDS 发送一个请求，目的是将该文件的所属目录修改为回收站，相当于为用户提供了文件恢复的功能，用户在客户端的回收站界面可以看到该文件。

当用户将回收站的文件进行彻底删除后，CCD 会向 Monitor 集群请求查询该文件的的所有数据对象信息和数据对象的位置信息，在获取到每个数据对象的主备份的 OSD 节点信息后，就向这些 OSD 发出删除对应数据对象的请求，这些 OSD 会计算得到这些数据对象的其他备份位置并向这些备份所在的二级 OSD 也发送删除请求，当相关 OSD 将该文件的主备份和所有冗余部分都清除后，就向 CCD 发送删除成功的信息，CCD 继而与 MDS 通信将文件的所有元数据删除并向客户端返回结果，至此文件删除流程结束。

3）文件修改

考虑到大文件修改的可能性极低，小文件的传输和分布存储对性能的影响极低且客户端的编辑细节极度依赖于编辑器的写入模式，于是将文件修改操作的实现直接等效于一次文件删除和一次文件创建。

4）目录操作

用户在客户端可以对文件的组织结构以文件夹即目录的形式进行管理，当用户创建一个文件夹时，CCD 与 MDS 进行通信，将文件夹的属性存入 MDS 集群，MDS 以树的形式组织每一个用户的目录元数据和文件元数据，目录元数据包括该目录的拥有者、权限、修改日期以及该目录下包含的文件的集合。用户对个人云盘的目录组织的修改都会对应为 MDS 元数据树的修改。

由于云盘系统拥有集中式的认证模块，用户信息集中存储于认证模块中，因此用户的概念对存储模块是不可见的，CCD 对不同用户的服务是基于不同的目录名称，当用户挂载个人云盘成功后，CCD 根据该用户的 ID 在 MDS 集群里找到该用户空间的根目录，通过将这颗目录树发送给客户端使得客户端可以将用户的个人云盘以通用文件系统的模式呈现出来。

4.4 CCD 设计

CCD 作为客户端与存储集群直接交互的接口，向上，它接受来自客户端经由负载均衡器分发的请求，向下，它与 Ceph 的 Monitor 集群、MDS 集群和 OSD 集群进行存储数据 IO 和元数据管理。每个用户在登陆后都以长连接的形式与 CCD 保持挂载连接，因此 CCD 需要支持横向的扩展，借由上层的负载机制以支持庞大的用户数量。客户端使用 CFSRPC 协议与 CCD 完成文件系统操作的映射，每个用户在登陆成功后都会向 CCD 发送云盘挂载请求，之后所有的文件操作和目录操作都会经由 CFSRPC 协议映射为 CCD 对下层 Ceph 集群的操作，因此 CCD 和 CFSRPC 协议是密不可分的。本节按照第三章对 CFSRPC 协议的各项机制和格式定义对 CCD 的功能和架构进行总体分析和详细设计。

4.4.1 CCD工作模型

在 CCD 这个层次上，Ceph 分布式文件系统官方提供了三种 Ceph Client 组件，其中 Ceph Block Device 提供了基于存储设备的访问层次，操作系统内核通过 Ceph 提供的 RADOS 协议将其作为一个硬件设备进行处理，同时 Ceph Block Device 也支持在用户态使用 librbd 库函数进行调用操作，它要求 Ceph 存储集群至少提供 OSD 和 Monitor 服务，工作模式如图 4-4 所示。

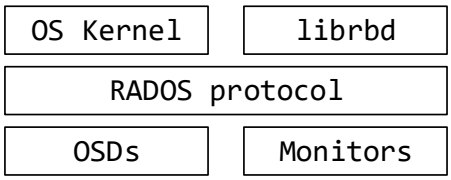

图 4-4 Ceph Block Device 工作模型

Ceph Object Gateway 基于 librados 提供了对象存储服务接口，满足 RESTful 格式的请求规范。官方为其提供了 S3 和 Openstack Swift 兼容的服务接口，该 Client 组件使用 radosgw 即对象存储网关的服务进程完成自身的对象存储功能，通过在该组件的宿主机器上 FASTCGI 接口进行 Ceph 存储集群的对象存储服务调用。该类型组件同样只需要 Ceph 集群里含有 OSD 和 Monitor 服务，其工作模型如图 4-5 所示。

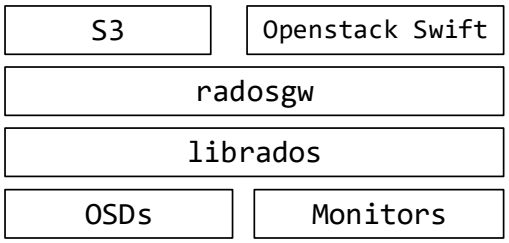

图 4-5 Ceph Object Gateway 工作模型

第三种名为 Ceph Filesystem，支持 POSIX 兼容的文件系统服务，该服务层节点的操作系统对其识别为一个对接在 VFS 模块上的特殊类型文件系统，在操作系统的文件系统操作环境下可以直接对该类型文件系统进行 POSIX 定义的操作，例如文件/目录增删改。该类型的组件需要 Ceph 集群除了 OSD 和 Monitor 服务以外提供一个名为 MDS（Metadata Service）的服务。其工作模型如图 4-6 所示。

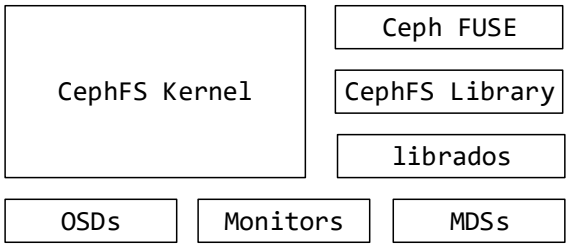

图 4-6 Ceph Filesytem 工作模型

对比三种 Ceph client 的工作原理和功能取向，Ceph Block Device 类型主要面向基础设施云服务对云化存储设备的需求，Ceph 存储资源被封装成一个存储设备提供给操作系统和指定的库函数，在其上不做任务文件存储服务和对象存储服务，甚至不提供文件系统的操作服务。Ceph Object Gateway 主要提供对象存储服务，且如果需要作为本文所述云盘系统的挂载接口，需要对其进行大量的定制化修改，令其能够支持 CFSRPC 协议，并实现协议定义的文件传输方式与对象存储调用方式的相互转换，以及添加用户个人空间的元数据存储服务功能。

Ceph 为 Ceph Filesystem 这一 client 类型设计实现了元数据服务器，为 Ceph 存储集群减轻了元数据频繁读取和修改操作的负担。且该类型的 client 组件使用 FUSE 提供了 POSIX 文件操作接口，其工作原理如图 4-7 所示。

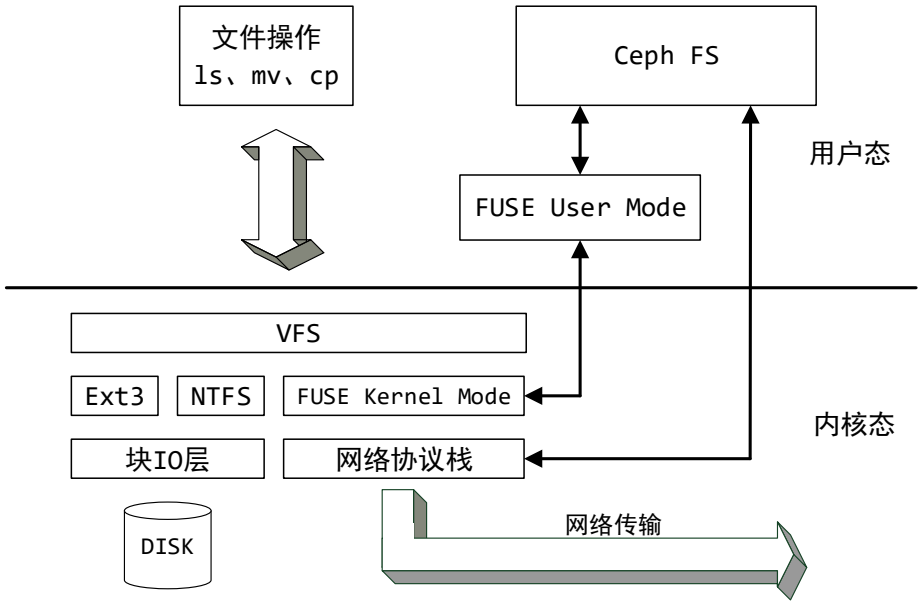

图 4-7 Ceph Filesystem 工作原理

云盘系统需要对每个用户的云盘文件目录结构和文件/目录属性进行维护，因此对元数据服务器有硬性的功能要求，为了利用 Ceph 官方的元数据服务器，使用 Ceph Filesystem 做定制化修改和设计开发，需要定制化的部分主要分为以下两点：

1）MDS 对元数据的存储是基于 CephFS 呈现出来的文件系统的目录结构组织的，云盘系统根据则需要根据用户来组织其个人空间的属性数据。CephFS 支持对用户目录的配额设置使得云盘系统可以通过为不同用户设置唯一的根目录来实现 Ceph 环境下用户元数据的隔离管理。这要求管理功能节点在用户第一次挂载云盘之前初始化这个用户的个人目录，并在客户端对元数据进行修改时能够将请求映射成 CCD 的处理流程，这当中涉及到管理功能模块与 CCD 的交互。

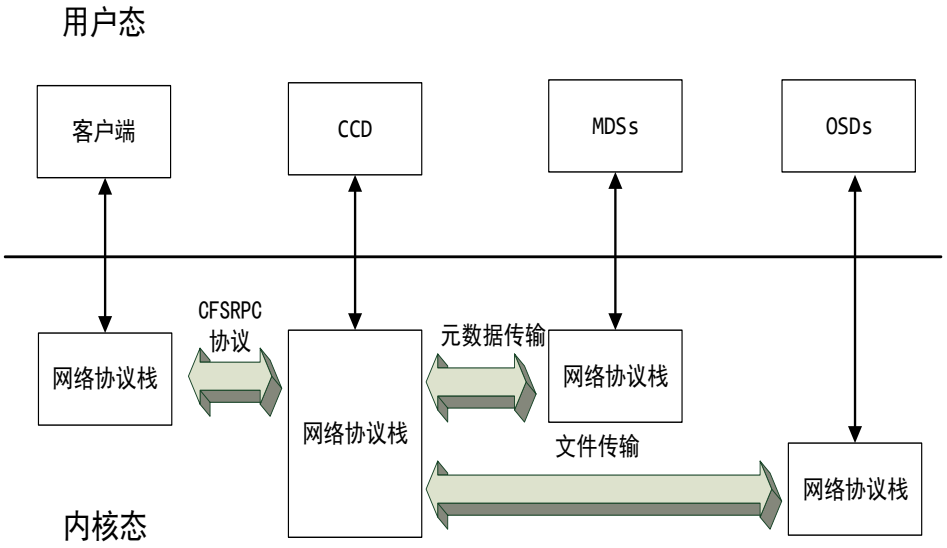

图 4-8 CCD 工作模型

2）如图 4-7 所示，可以看到 Ceph Filesytem 实现了一个 POSIX 兼容的文件系统，FUSE[33]帮助 CephFS 做文件系统操作的网络转发，反过来，CephFS 将网络上的文件传输流通过 FUSE 进行 POSIX 标准的功能映射。但是云盘系统的客户端并不需要 POSIX 标准的文件系统支持，CCD 只需要将云盘操作服务直接定向到 Ceph 存储集群和 MDS 集群即可，这个 FUSE 的中介过程可以完全去除，因此对 CephFS 程序做定制修改，使得其按照图 4-8 的架构运行。

4.4.2 CCD业务流程

图 4-9 CCD 详细业务流程图

本节对 CCD 进行业务流程分析，针对流入 CCD 的数据和流出 CCD 的数据分别进行分析，并设计其关键处理流程。

公有云环境下的云盘系统需要能够支持足够多的用户并发访问，在这样的工况下，CCD 会接受到上层负载机制不断分发的用户请求。为了方便这些业务的处理和隔离，第一步是将这些请求进行初步的分派，每一个请求事务会被分派给一个执行体进行单独的处理。在这样一个隔离的环境，CCD 的进程会首先对收到的应用层请求报文进行 CFSRPC 协议的解析操作，按照第三章对该协议的定义，获取到一个 CFSRPC 请求过程报文的认证令牌、操作指令代码、参数串长度、以及参数串本身。然后依次对操作指令代码进行比对，得到该次业务的具体请求操作，将报文携带的认证令牌发送到认证模块进行该操作的权限认证，如果不通过，直接返回失败的响应报文，对该次事务接收到的其他数据予以丢弃，反之则继续解析取得参数串长度，根据参数串长度截取参数串本身。接着根据操作指令的参数格式定义，解析得到该操所需参数，至此完成了该次操作执行的全部执行条件。CCD 对不同的操作有不同的处理机制，任何涉及到元数据操作的请求都会基于与 MDS 集群的交互，任何涉及文件 IO 的请求都会基于与 OSD 集群和 Monitor 集群的交互。

许多业务请求需要 CCD 发送反向的响应报文，文件下载操作更需要 CCD 进行反向的文件传输，在业务处理过程中出现的权限验证不通过或者业务处理细节问题也会以响应报文的形式返回给客户端，一个执行体在处理完请求业务需要反向发送响应时会按照 CFSRPC 协议定义的应答过程结果格式进行响应报文的包装和发送。CCD 在接收到请求后会在接下来做出复杂的操作和决策，详细流程如图 4-9 所示。

4.4.3 CCD处理架构

用户在通过客户端进行成功的登陆操作后，客户端进程会与 CCD 保持一个长连接，在这一次登陆会话当中，客户端通过这个连接进行云盘服务的请求，其整体处理架构如图 4-10 所示。

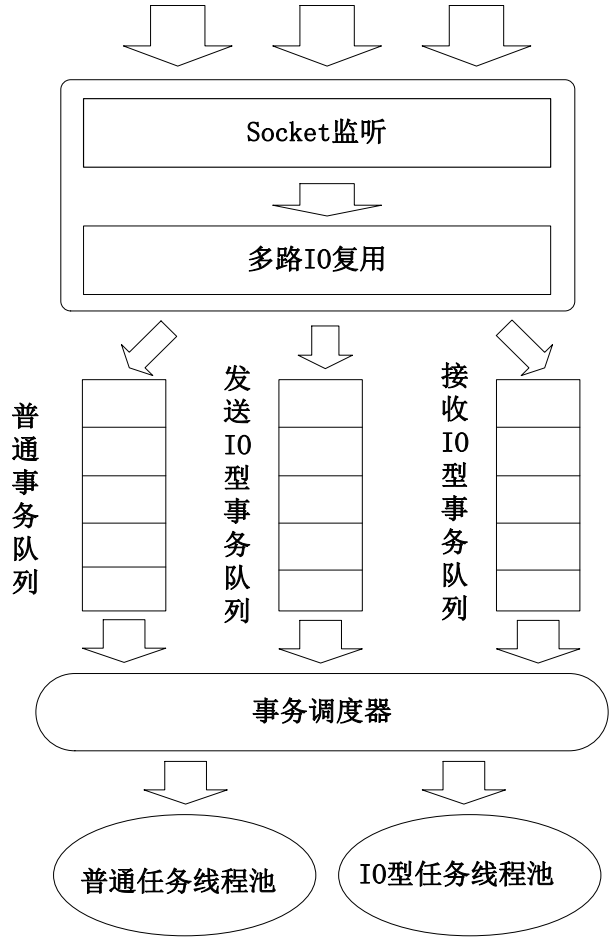

图 4-10 CCD 处理架构

4.4.3.1 监听程序

CCD 默认设计为使用 9011 端口进服务的监听，它需要监听两种类型的请求，第一种是用户登录过程完毕后产生的 TCP 建联请求，第二种是客户端在与 CCD 建联后对其发送的请求事务。因此一台 CCD 服务器往往维护了大量的网络套接字，以随时对每一条连接的请求进行处理和响应。传统的多线程方式在事务处理的隔离性方面有一定有优势，但是如果对每个连接和请求都使用新建线程的方式进行处理，即使线程开销比进程小很多，在大量用户的公有云环境下，大量的线程共存仍会造成极大的资源浪费和切换开销。

网络套接字（Socket）在 Unix 和 Unix like 系统里表现为一个文件描述符（File Descriptor）的形态[34]，CCD 监听程序的工况用 Unix 的术语描述就是监听大量的文件描述符，用户的建联请求和业务请求发送到 CCD 所在服务器后，经过内核网络模块的处理最终通过 VFS 以多态实现的方式经由文件描述符接入服务进程。传

统的多线程方式是对每个文件描述符维持一个线程的上下文环境并进行轮询操作以查看是否有新的请求到达。大量的文件描述符需要大量的线程进行轮询，在高压生产环境下不仅会导致新来的请求不能得到及时处理而且会浪费大量的计算和存储资源。

在同一个时刻往往只有很小部分的文件描述符有新的数据需要处理，多路 IO 复用机制[35]为这样的监听需求提供了一个极为合适的解决方案，大量的文件描述符在该机制下进行统一的维护，其原理如图 4-11 所示，监听进程自身可以维护大量的文件描述符，使用集合轮询或者事件触发机制来实现对文件描述符的感应，每次轮询或者被触发后就将待处理的文件描述符接入处理程序。

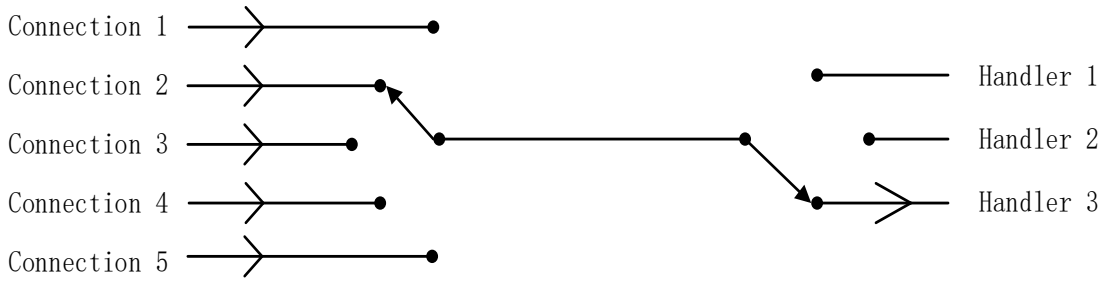

图 4-11 多路 IO 复用机制事件感应原理

Linux 提供的 epoll 机制[36]是一个很好的多路 IO 复用的实现，CCD 的监听程序使用 epoll 进行实现，以达到高效监听大量网络连接的目的。

4.4.3.2 事务调度

从图 4-10 中可以看出，CCD 在接受到请求后会先行做 CFSRPC 协议的 RPC 部分的解析，因此 CCD 的监听程序在通过多路 IO 复用机制接收到一个请求后首先对其进行 RPC 头部的解析和验证，通过后，针对不同的类型将事务存入不同的事务队列。将事务的待处理队列分成普通事务队列、发送 IO 型事务队列、接收 IO 型事务队列的原因有以下两点：

1）CCD 监听程序将事务做了先行处理后会将每一个事务分给一个线程进行后续的业务处理，事务调度器会从不同的等待队列里取出等待执行的事务，然后交给一个合适的线程。等待队列的分离方便了事务调度器实现定向调度。

2）CFSRPC 协议支持的分块大小动态调节机制需要获取发送 IO 型事务等待队列和接收 IO 型事务等待队列的实时长度作为 CSDA 算法的输入参数，这两种类型等待队列的长度变化反应了当前 CCD 的处理事务的负载情况。

事务调度器作为一个线程保持对三个等待队列的轮询操作，轮询的过程中扫描到任何队列有待处理的事务项，就将其取出并调度相关线程做后续的业务执行。

4.4.3.3 事务处理线程池

普通型事务通常只是对文件或目录的元数据和存储配置的增删改查，需要占用的处理资源较少，执行速度较快，而 IO 型事务涉及到文件传输，占用的资源和时间往往远超普通型事务，因此不同的事务的处理线程的管理方式能够影响 CCD 的整体性能。

处理线程以线程池[37]的方式进行管理，池技术的应用可以有效的减少线程创建和回收的开销。CCD 使用两个线程池分别处理普通型事务和 IO 型事务，IO 型事务的处理速度很慢，导致处理 IO 型事务需要较多的时间，其载体线程会在较长一段时间内被占用，因此对该类型任务需要更多待命线程。而普通型事务执行速度很快，线程被占用后会在很短的时间内迅速回归线程池。两种类型的事务在请求数量和处理速度上都有不小的差别，池的分化管理可以降低事务处理线程的管理难度。

4.5 认证模块设计

4.5.1 认证机制设计

本文的云盘系统拥有集中式的认证系统，用户管理和用户操作权限验证的服务统一由认证模块提供。Keystone 作为 Openstack 的核心组件能够提供简洁可靠的认证服务[38]，其 RESTful 的调用格式便于进行跨系统的移植，但是其用户管理涉及的管理角色太多，且复杂而冗长的 Token 编码方式跟 CFSRPC 协议的的简洁性不符。因此云盘系统参考 Keystone 的认证模式，采用 UUID 的 Token 生成方式，重新设计一个简单版本的认证系统，作为系统的集中认证模块，同时兼具用户管理功能。

按照图 4-1 的功能架构设计，认证模块位于系统的管理层，业务管理模块在接受到用户的登陆请求后会将用户的身份信息发送给认证模块对用户的身份信息进行验证，成功后会为本次的登陆会话计算生成一个新的 UUID 并存储到数据库的对应用户记录中，将此 UUID 作为令牌信息 Token 经由业务管理模块返回给客户端，用户在接下来的请求操作都会携带这个 Token，CFSRPC 协议在请求过程部分为认证令牌提供了协议级别的携带方式。图 4-12 展示了认证模块提供认证服务的流程。

用户在成功获取到 Token 后，在每一次请求操作的时候会经由 CFSRPC 协议的首部携带该 Token，CCD 在接受到请求后，会解析出 Token 信息和具体操作信息，然后将该 Token 发送给认证模块进行该操作的权限认证，再次通过后，CCD 就会向用户发送允许执行请求的指令，至此认证流程结束，客户端和 CCD 的业务交互开始执行。

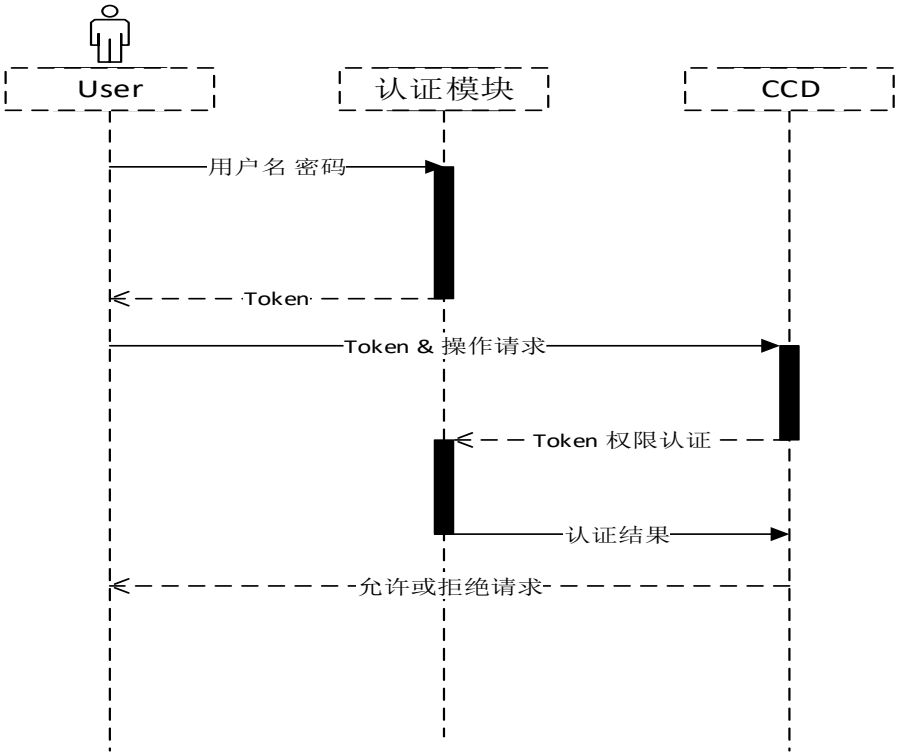

图 4-12 认证服务时序图

每一个 Token 生成后只对该次登陆会话其作用，当用户注销该次会话后，认证模块会将该用户当前的有效身份令牌清空，当用户再次登陆后重新计算一个新的令牌信息，这样的机制避免了用户在进行每一次操作钱都对用户名密码进行验证，同时增加了用户操作的安全性。

4.5.2 处理架构设计

认证模块的服务只提供给 CCD 和管理功能模块，情景如下：

1）CCD 的事务处理线程在从 CFSRPC 协议首部的请求过程部分剥离出认证令牌和操作指令后向认证模块请求权限认证。

2）访问层向管理功能模块发出的请求涉及用户和权限管理。

CCD 和管理功能模块都是拥有多个负载节点，因此认证模块的守护进程只需要维护这两种节点数量之和的连接数量即可，对一个集群来说，CCD 节点和管理

功能模块节点的数量是相对较少且可控的，对认证模块而言，内网环境下的并发量和负载程度并不会太高，因此以多进程的方式来提供服务，每个请求节点对应一TCP 个连接，每个 TCP 连接对应一个进程，每个进程对接受到的认证请求进行串行化处理，服务架构如图 4-13 所示。

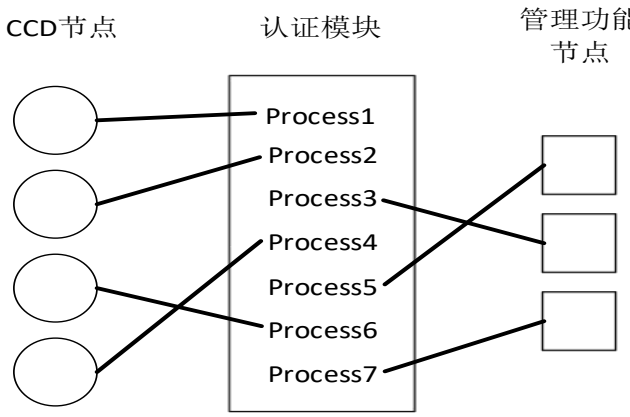

图 4-13 认证模块处理架构

4.6 管理功能模块设计

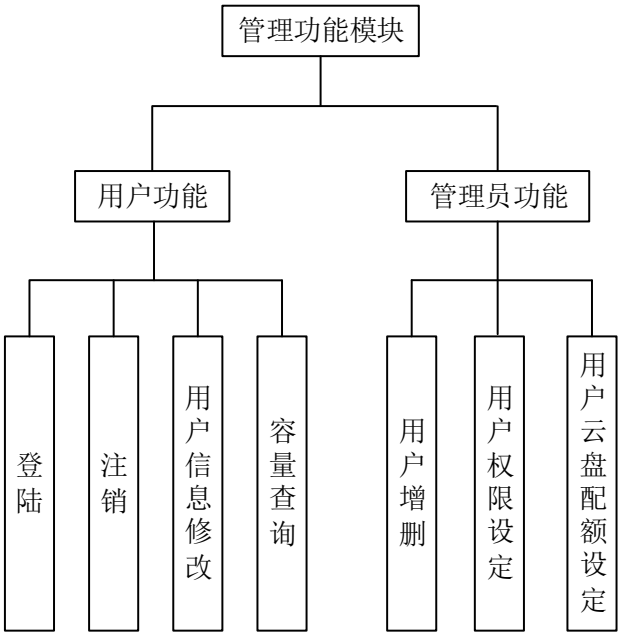

图 4-14 管理功能模块功能模型

管理功能模块与认证模块同样位于管理层，云盘系统对用户的权限信息、配额信息以及用户的个人配置信息都由管理功能模块进行维护，其功能架构如图 4-14

所示。

管理功能模块向上为用户和管理员以 RestfulAPI 的格式提供不同的管理功能，用户和管理员可以通过各自的客户端直接向该模块进行图 4-14 所示的功能请求。向下与认证模块通信，获取认证相关操作的服务。

用户的登陆、注销请求和管理员的用户权限设定请求都需要管理功能模块对认证模块的请求调用，如图 4-15 所示。而用户信息修改、容量查询以及用户增删和用户云盘配额设定只需要在数据库做对应的修改和查询。

管理功能模块以 Web Service 的形式为访问层提供服务，该模块需要接收大量的用户和管理员的请求，因此对其做均衡服务资源的负载节点设计，在每台管理功能节点上都搭载一个 Http 服务器并提供云盘的管理功能的 REST API 服务。

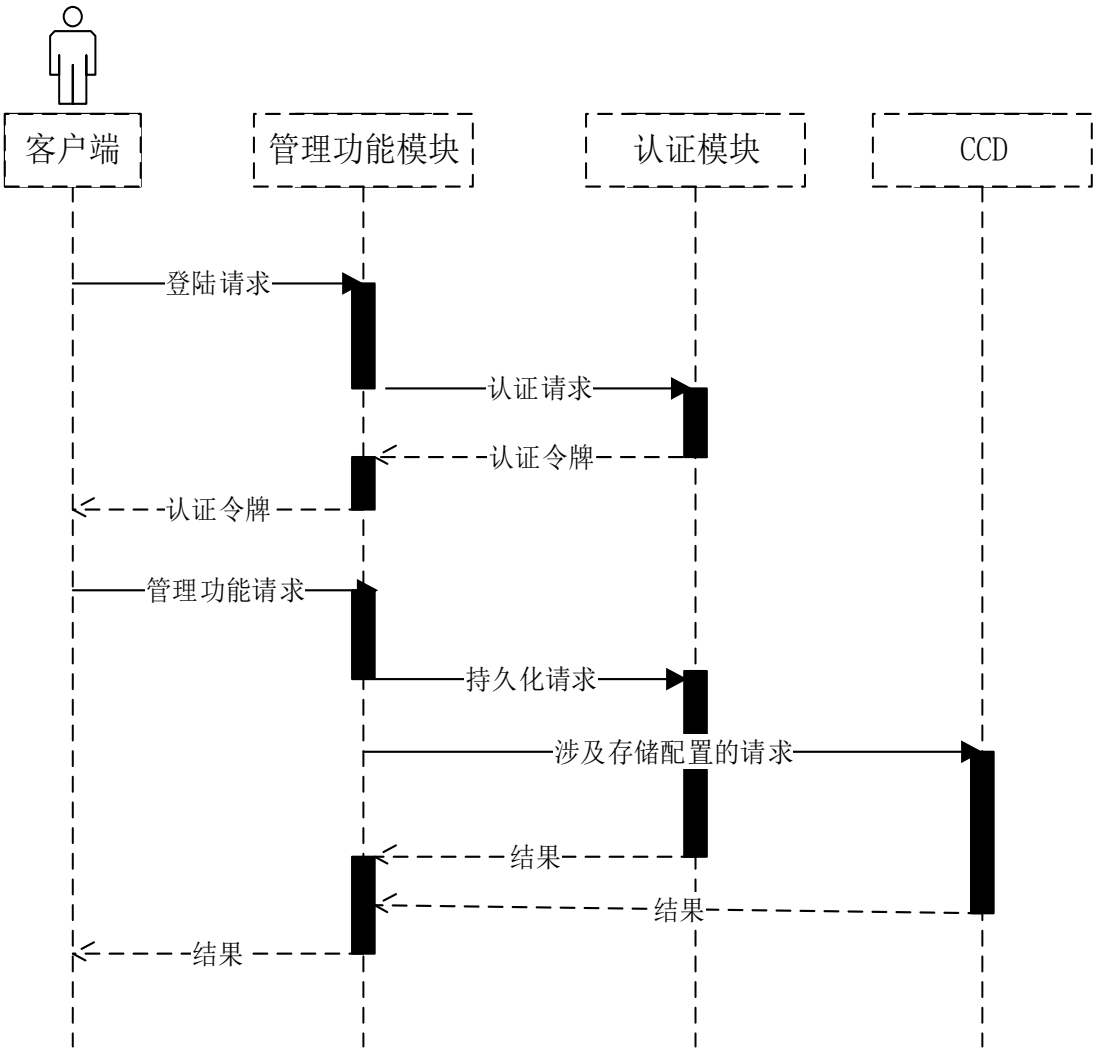

图 4-15 管理功能模块业务处理时序图

4.7 云盘客户端设计

客户端是用户访问的入口，向上需要为用户操作提供友好的 UI，向下需要通过实现 CFSRPC 协议的请求端来完成各种用户操作的请求发送和结果接收。UI 的的设计风格参考各类通用文件系统的操作界面，用户在通过登录界面成功认证后进入该操作界面，登录流程如图 4-16 所示。

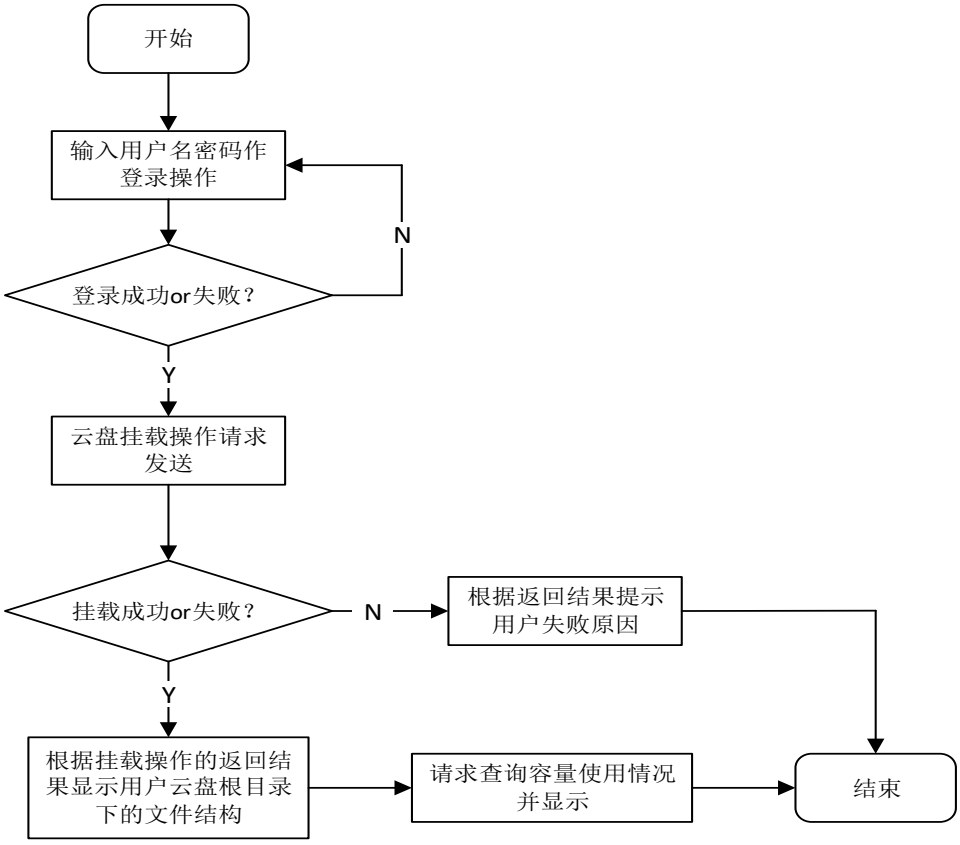

图 4-16 客户端登陆流程

登录流程完成后，客户端会与某一个 CCD 建立一个 TCP 连接，用户可以通过客户端 UI 进行其他文件操作。文件上传操作流程如图 4-17 所示，文件下载操作与其类似，也存在文件分块的接收循环，但不需要进行容量的判断。文件移动、删除操作以及所有的目录操作都是对用户数据的元数据操作，按照 CFSRPC 协议的规定进行各自操作的首部填写并通过挂载操作与 CCD 建立的链接进行一次请求的发送和一次结果的接收即可完成操作流程。

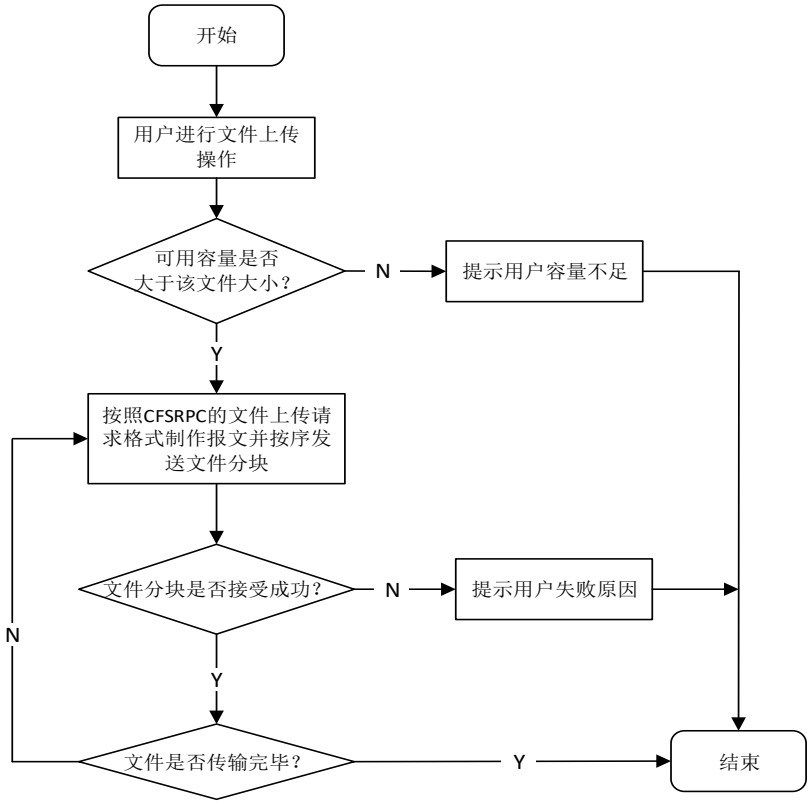

图 4-17 文件上传处理流程

4.8 本章小结

本章首先从功能和性能上对云盘系统进行了总体需求分析，并对云盘系统的核心业务流程进行分析，针对云盘系统依赖的多个关键模块进行了分析和整合设计，总结出云盘系统的整体软件架构。逐层分析了各个模块之间的依赖关系，设计其交互模式和负载结构。之后基于 Ceph 分布式文件系统设计核心存储模块，研究分析 Ceph client 和 MDS 的实现原理并根据云盘系统架构和 CFSRPC 协议设计 CCD。同时，参考 Keystone 的令牌认证机制设计了独立认证系统。最后对管理功能模块进行了总体设计并对客户端工作流程进行了分析，根据 CFSRPC 请求部分的定义对其各个功能的执行流程进行了设计。

第五章 云盘系统细节设计和整体实现

本章根据第四章对云盘系统的总体分析和各个模块的详细设计，结合第三章 CFSRPC 协议的定义，对云盘系统各个模块的实现细节进行设计并给出具体实现方案。

5.1 总体框架实现

5.1.1用户接入的负载机制实现

根据 4.2.2 小节对用户接入的负载机制的设计，使用 Keepalived+LVS 的负载均衡方案，为管理功能节点和 CCD 节点实现请求的均衡分发，管理功能节点通过 apache2 作为 HTTP 服务器，监听端口使用默认的 80 端口，而 CCD 服务程序的默认端口为 9011。根据二者端口号的不同来对传输层的报文进行转发以达到功能分流的目的。本小节通过对一个实例环境的部署来描述云盘系统负载机制的实现方式。该实例环境配置如表 5-1 所示。Master 和 Backup 作为负载层为 CCD1-2 和 Server1-2 实现功能分流和负载分流。

表 5-1 负载实例环境配置

角色	地址	描述
Master	172.20.100.2	负载主节点
Backup	172.20.100.3	负载备节点
VIP	172.20.100.1	负载入口虚拟 IP
CCD1	172.20.20.1	CCD 节点
CCD2	172.20.20.2	CCD 节点
CCD3	172.20.20.3	CCD 节点
Server1	172.20.30.1	管理功能节点
Server2	172.20.30.2	管理功能节点

首先为 Master 和 Backup 安装 LVS 和 Keepalived，安装步骤如下：

```
//安装 LVS
tar -zxvf ipvsadm-1.24.tar.gz          //解压源码。
ln -s   /usr/src/kernels/`uname -r`/   /usr/src/linux //软链指向。
make&&make install                     //编译和安装。
```

```
//安装 Keepalived
tar -zxvf keepalived-1.2.2.tar.gz //解压源码。
./configure --prefix=/usr/local/keepalived //准备编译环境。
make&&make install                          //编译和安装。
cp /usr/local/keepalived/sbin/keepalived /usr/sbin/
service keepalived start|stop       //启动或者停止
```

接着对两个节点的 Keepalived 服务做如下配置，配置文件的全路径名默认为 /etc/keepalived/Keepalived.conf。

```
global_defs {
smtp_server 127.0.0.1       // 定义 smtp 服务器
smtp_connect_timeout 15    // 定义连接超时
router_id LB_UESTC
}
vrrp_instance VI_1 {
    state MASTER    // 表示该机器为主节点, Backup 节点该处设为 BACKUP
    interface eth0          // 网卡标识
    virtual_router_id 51 // 虚拟路由标识
    priority 100             // 优先级，MASTER 要高于 BACKUP
    advert_int 1             // 主备之间同步检查的时间间隔
    authentication {       // 使用密码的验证方式
        auth_type PASS
        auth_pass uestc
    }
virtual_ipaddress {
    172.20.100.1      //虚拟 IP 地址，该值必须和 LVS 中的 VIP 一致
}
```

然后对主备负载节点都执行以下 LVS DR 转发配置：

```
ipvsadm –C // 清空原配置
ipvsadm –set 30 5 60 // 设置链接超时值
ipvsadm –A –t 172.20.100.1 –s –wwr –p 20 // 设置 VIP
// 设置将 9011 端口的包分发给三个 CCD
ipvsadm –a –t 172.20.100.1:9011 –r 172.20.20.1:9011 –g –w 1
ipvsadm –a –t 172.20.100.1:9011 –r 172.20.20.2:9011 –g –w 1
```

```
ipvsadm –a –t 172.20.100.1:9011 –r 172.20.20.3:9011 –g –w 1
// 设置将 80 端口的包分发给两个管理功能节点
ipvsadm –a –t 172.20.100.1:80 –r 172.20.30.1:80 –g –w 1
ipvsadm –a –t 172.20.100.1:80 –r 172.20.30.2:80 –g –w 1
```

完成以上配置后，先后依次重启 LVS 和 Keepalived 的服务即可实现用户接入的负载机制以及负载节点本身的故障漂移。其命令分别为：

```
service ipvsadm restart
service keepalived restart
```

5.1.2 核心存储模块实现

核心存储模块包含 Ceph 分布式文件系统中的 Monitor 节点、OSD 节点、MDS 节点以及用户操作的接口 CCD 节点，本小节根据 4.3 节的设计对 Monitor、OSD、MDS 节点的部署和配置做出描述，CCD 由于涉及对 CephFS 的二次开发，功能复杂，将其实现细节放在 5.2 节做单独描述。

Ceph 官方提供了 Ceph 集群内功能节点的简易安装方式，通过使用 python 实现的一个工具 ceph-deploy 可以很大程度上简化 ceph 集群的配置过程。在 Debian/Ubuntu 的系统环境下，其安装流程为：

1）增加发行版本的秘钥：

```
wget -q -O- 'https://download.ceph.com/keys/release.asc' | sudo apt-key add –
```

2）把 Ceph 的安装源添加到系统的 apt 仓库中。

```
echo deb https://download.ceph.com/debian-{ceph-stable-release}/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/ceph.list
```

3）更新源并安装

```
sudo apt-get update && sudo apt-get install ceph-deploy
```

在进行功能节点安装和配置前，为每一个机器都安装上 ceph-deploy 工具和 ntp 服务并关闭其 iptables 和 selinux 服务，同时在几台机器之间配置 SSH 互信。

以上工作做完后，进入需要安装 Monitor 服务的机器，使用 root 账户登录并实施以下安装流程。

1）创建工作目录，用于存放生成的配置文件和秘钥等信息

```
sudo mkdir /ceph
sudo cd /ceph
```

2）设置当前节点为 Monitor

```
ceph-deploy new 127.0.0.1
```

3）添加 Monitor 基本配置，设置每个 pool 的默认副本数为 3，最小副本数为 2，设置公共网段为 172.20.20.0/24 用于为客户端提供服务，设置集群工作网段为 172.20.40.0/24 用于集群同步数据、心跳检测等。

```
echo "osd pool default size = 3" >> ceph.conf
echo "osd_pool_default_min_size = 2" >> ceph.conf
echo "public network = 172.20.20.0/24" >> ceph.conf
echo "cluster network = 172.20.40.0/24" >> ceph.conf
```

4）激活 Monitor 节点

```
ceph-deploy mon create-initial
```

5）创建并激活 OSD 节点

```
ceph-deploy osd prepare 172.20.20.1:/dev/sdb1 172.20.20.2:/dev/sdb1 172.20.20.3:/dev/sdb1
ceph-deploy osd active 172.20.20.1:/dev/sdb1 172.20.20.2:/dev/sdb1 172.20.20.3:/dev/sdb1
```

6）将 Monitor 上的配置同步到所有 OSD 节点上

```
ceph-deploy --overwrite-conf admin 172.20.20{1..3}
```

7） 建立 MDS

```
ceph-deploy mds create 172.20.20.100
```

8）创建两个 pool，一个名为 data 用于存储云盘文件数据，另一个名为 metadata 用于存储云盘元数据信息，两个 pool 的 PG 默认值都设为 256。

```
ceph osd pool create data 256
ceph osd pool create metadata 256
```

至此除了作为 ceph client 角色的 CCD 没有部署以外，Ceph 集群就已经部署成功了，可以使用 ceph –s 命令查看 Ceph 集群的状态，如果状态显示为 active+clean，表示集群可用。

5.2 CCD 实现

4.4 节对 CCD 的设计是基于 CephFS 进行二次开发，准确的说是只保留了 CephFS 与 MDS 的交互方式和 CephFS 对文件目录操作的处理流程，重新设计了基于多路 IO 复用、事务调度和线程池的处理架构，并设计了基于 CFSRPC 协议的业务处理流程和独立的认证流程。本节对以上设计方案的实现方案作出描述。

5.2.1 CCD处理架构实现

用户在使用云盘服务之前会先行登陆操作，成功后会直接向 CCD 发送云盘挂载请求，之后会与 CCD 保持一个长连接以作为后续服务请求的通信渠道，因此 CCD 首先调用 Request_Listen 函数，该函数注册了一个端口号为 9011 的 TCP 网络监听套接字并将其设置为非阻塞模式。之后的程序流程如图 5-1 所示。

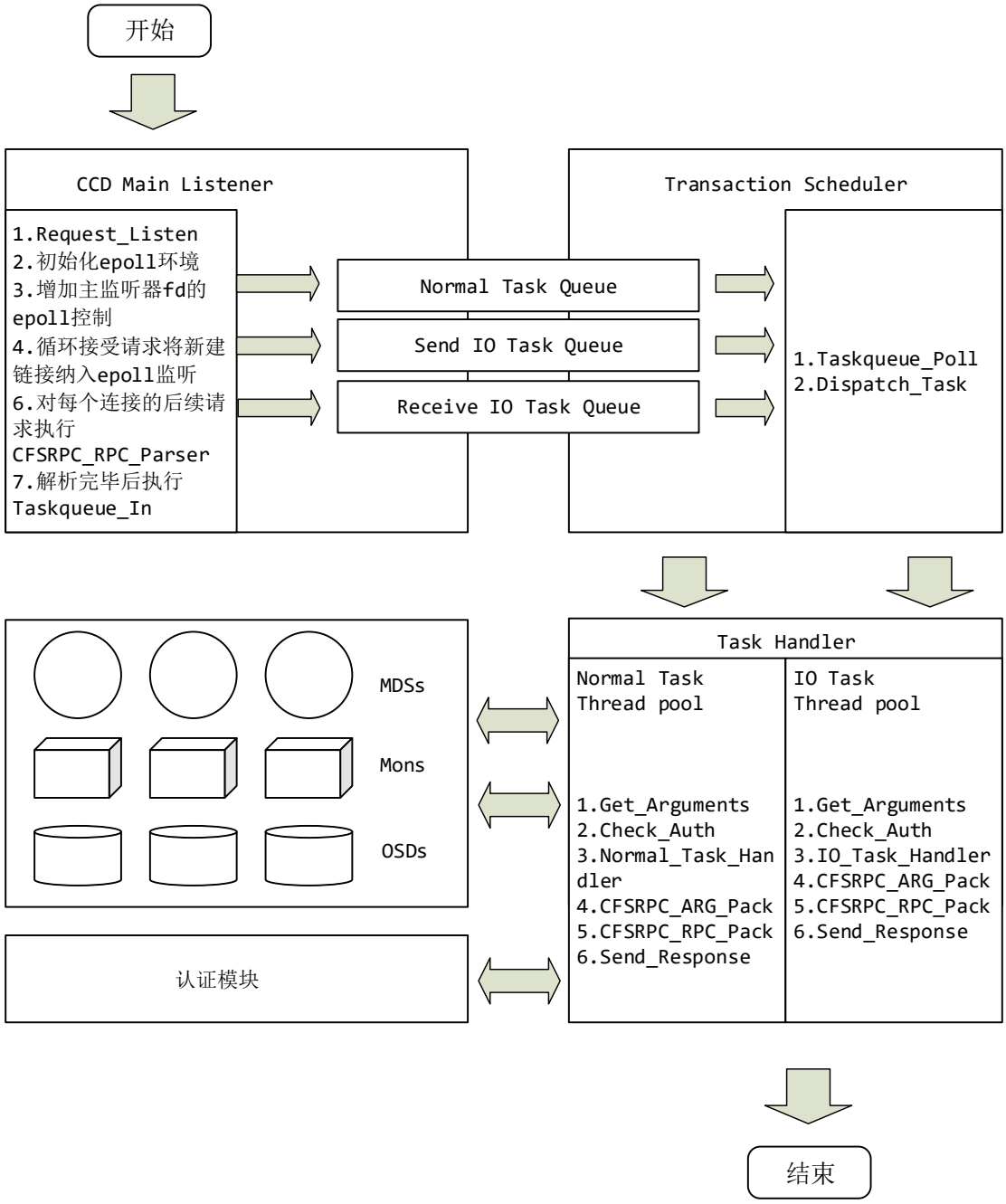

图 5-1 CCD 服务程序实现架构

CCD 的整个监听程序基于 epoll 多路 IO 复用机制，在注册完接收建联请求监听套接字以后，在 main 函数里直接进行 epoll 环境的初始化，即使用 epoll_create 创建一个 epoll 的句柄，接着创建该 9011 端口的监听事件并纳入 epoll 的触发机制，当 epoll_wait 返回，表示监听的 fd 有新的事件到达，如果该 fd 与 sfd 相等，表示新的用户连接到达，新建一个临时端口的 TCP 监听套接字并使用 epoll_ctl 加入监听管理，如果该 fd 与 sfd 不等，表示某个已建联的用户发来新的请求，这时调用 CFSRPC_RPC_Parser 对请求报文进行 RPC 部分的解析和检查，无误后根据任务类型将该用户连接的 fd 经由 Taskqueue_In 函数送入对应的待处理管道，该 fd 之后的文件传输将托管给事务处理线程，为了在托管期间的 IO 动作不会触发 epoll_wait 返回，在这里将该 fd 的监听暂时移除，但不断开连接，至此 CCD 主监听服务程序流程执行完毕，这个过程用伪代码描述如下：

```
sfd, efd, event;  // 已经建立并绑定了 9011 端口的 socket fd
listen(sfd, SOMAXCONN);  // 监听开始
efd = epoll_create1(0);          // 创建 epoll 句柄
event.data.fd = sfd;             // 将 sfd 加入 epoll 管理
event.events = EPOLLIN | EPOLLET;
epoll_ctl(efd, EPOLL_CTL_ADD, sfd, &event);
while (1) {
    n = epoll_wait(efd, events, MAXEVENTS, -1);
    for ( ; i<n; ++i) {
        if (sfd == events[i].data.fd) { // 意味着新的用户连接到来
            nfd = accept();
            epoll_ctl(efd, EPOLL_CTL_ADD, nfd …);
        } else {                        // 意味着建联用户新的请求到来
            read(nfd, buf, size)  // 从中读取 CFSRPC 报文首部的 RPC 部分
            CFSRPC_RPC_Parser(buf); // 解析
            Taskqueue_in(nfd);          // 加入待处理事务队列
            epoll_ctl(efd, EPOLL_CTL_DEL, nfd …); //暂时移除该 fd 的监听
        }
    }
}
```

CCD 主监听程序（CCD main listener）将接收到的所有请求都先行进行了 CFSRPC 协议的 RPC 部分解析和检验，然后将通过了检查的合格报文对应的通信

文件描述符 fd 按类型放入 3 个管道中，事务调度器（Transaction Scheduler）使用一个单独线程轮询三个管道，这里注意要对队列的长度进行加锁处理以避免与主线程做的 push fd 操作产生并发错误，当读出一个 fd 后，调用 Dispatch_Task 向事务处理线程池 Task Handler 请求调度一个对应类型的处理线程并将 fd 作为参数传递进去，至此事务调度器的任务完成。其过程以伪代码描述如下：

```
while(1) {
    int fd = 0;
    fd = taskq.pop();
    if (fd != 0)
        Dispatch_Task(fd);
}
```

任何一个空闲的事务处理线程在接受到 fd 的参数传入后，就会对 fd 的连接进行独占式的处理，先将剩余的 CFSRPC 协议首部读出，根据用户请求的操作指令和认证令牌向认证模块发起权限认证请求，认证通过后调用 CFSRPC_ARG_Parser 函数根据协议对请求过程参数的定义进行请求参数的解析和处理，无误后调用 Normal_Task_Handler 或者 IO_Task_Handler 执行请求服务，执行完毕后调用 CFSRPC_ARG_Pack 和 CFSRPC_RPC_Pack 对响应报文进行构造，最后调用 Send_Response 将响应报文发出，这里要注意需要将该 fd 重新加入 epoll 的监听，以便响应该用户的后续请求。这个过程的实现细节涉及对报文的解析流程和对文件目录操作的执行流程，以及对 CSDA 算法的实现，该内容放在 5.2.2 和 5.2.3 节进行详细描述。

Task Handler 以两个线程池的方式提供了事务处理载体，暴露给 Transaction Scheduler 一个定义为 TaskHandler(int fd, int type)的接口，调度器将根据任务类型来指定调用参数。两个线程池拥有一个主管理线程，通过该主管理线程对两个线程池进行管理和调用，线程池的结构和主要功能函数如下所示：

```
struct threadpool {
    int count;                                  //   总线程数
    int idle;                                   //   空闲线程数
    int max_threads;                            //   最大线程数
    void *(*real_task_handler)(int fd, int type); //   线程处理函数
};
// 线程池初始化
```

```
void threadpool_init(threadpool *pool, int threadmax);
// 直接调用 threadpool 结构体里的 real_task_handler 函数;
void task_handler(threadpool *pool, int fd, int type);
// 回收线程池
void threadpool_destroy(threadpool *pool);
```

主管理线程会在生成时构造两个线程池结构体，依次指定其 real_task_handler 函数指针指向 Normal_Task_Handler 和 IO_Task_Handler 两个实现函数，同时提供 CGI 形式的接口便于管理员调整两个线程池允许的线程最大数量。

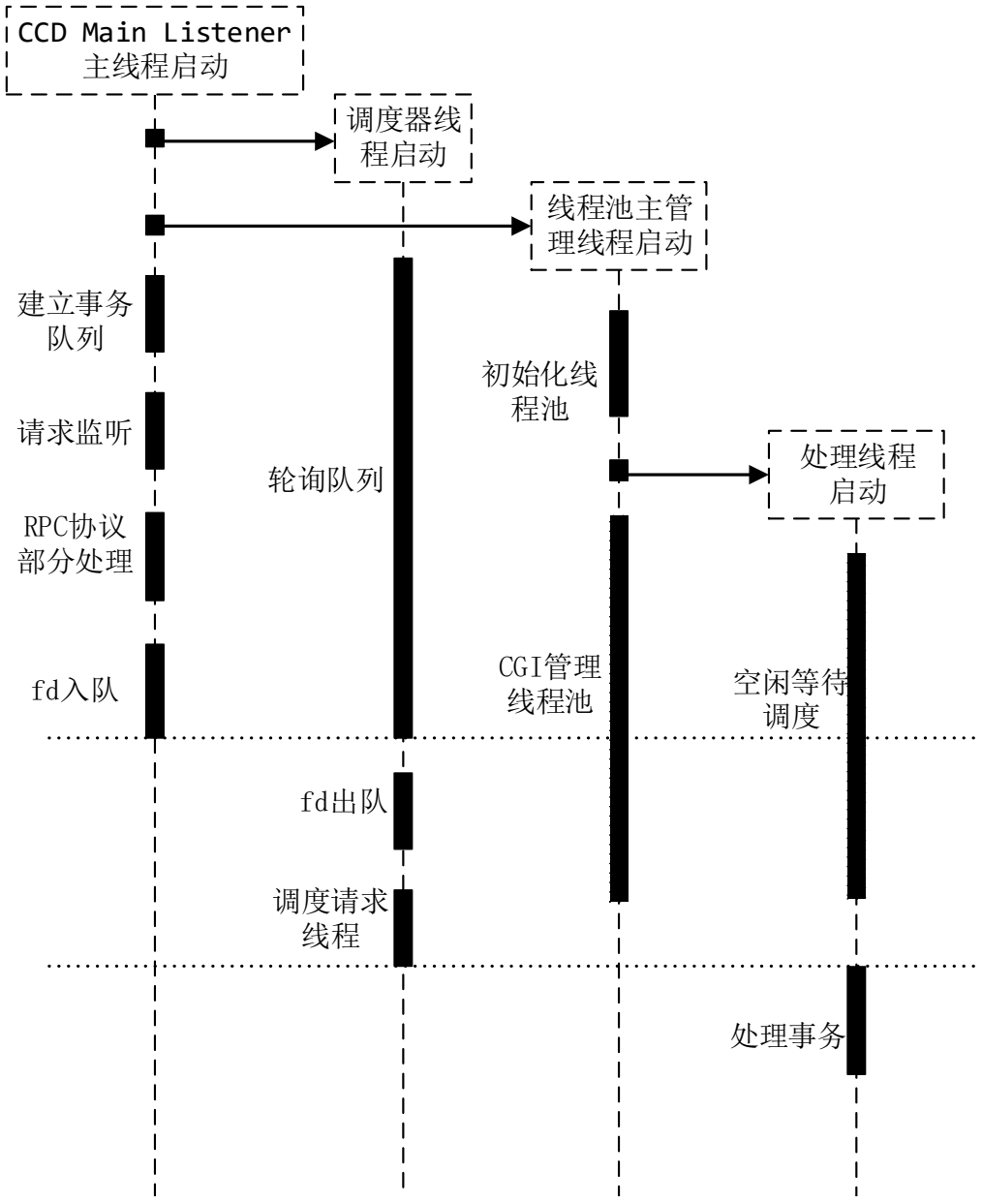

图 5-2 CCD 多线程处理过程

整个 CCD 基于一个单进程多线程的模型，每个模块都作为一个线程运行，在不同的时刻被主线程调动运行，线程启动的时序关系用图 5-2 进行描述。

CCD 服务进程启动后，主线程首先会启动一个调度器线程进行三个队列的轮询操作，再启动一个线程池管理线程按照默认的参数创建两个空的线程池，之后，主线程在完成请求监听、RPC 协议部分处理并将事务对应的网络 fd 推入待执行队列后，调度器线程的轮询会得到返回，并将 fd 出队并调度一个事务处理线程进行接下来的任务执行，这期间涉及到队列操作的并发，需要进行加锁处理，事务处理线程在完成指派事务的处理后需要将 fd 重新加入主线程的 epoll 监听，涉及到了 epoll 描述符的并发操作，因此也需要对 epoll_ctl 进行加锁处理。

5.2.2 CFSRPC协议的服务端实现

CCD 在处理请求的过程中会多次使用到 CFSRPC 协议相关功能，例如在主线程接受到用户的建联请求后就会先行对 CFSRPC 报文的 RPC 部分进行解析，事务处理线程在接受到文件描述符后又会进行 CFSRPC 的请求过程参数部分解析，在返回客户端之前又需要进行响应过程结果和 RPC 部分的首部制作。因此在服务端将 CFSRPC 协议设计的操作封装为一个工具类，名为 CFSRPC_Util，其定义描述为以下代码：

```
class cfsrpc_rqt_rpc {                    // 请求报文的 RPC 部分
        int32 txid;
        int32 protocol_version;
        int32 program_code;
        int32 program_version;
        char task_type;
}
class cfsrpc_rqt_arg {                    // 请求报文的过程参数部分
        GUID token;
        int32 opeartion;
        Arguments arg;
}
class cfsrpc_rsp_rpc {                    // 应答报文的 RPC 部分
        int32 txid;
        char result;
}
```

```
class cfsrpc_rsp_arg {                    // 应答报文的过程结果部分
        Arguments arg;
}
static cfsrpc_rqt_rpc CFSRPC_RPC_Parser (string s) {}
static cfsrpc_rqt_arg CFSRPC_ARG_Parser(string s) {}
static string CFSRPC_RPC_Pack(cfsrpc_rsp_rpc) {}
static string CFSRPC_ARG_Pack(cfsrpc_rsp_arg, long chunksize = 0) {}
```

其中提供了请求报文首部和应答报文首部的类定义，4 个静态工具函数分别实现了请求/应答报文首部 RPC 部分和过程参数部分的解析和封装。

事务处理线程会在解析的过程中调用这个类的工具函数以完成涉及 CFSRPC 协议的相关工作，在请求过程参数和应答过程结果部分，有一个类型为 Argument 的类型，它被设计为一个基类，不同的操作会对应不同参数类型，因此对 CFSRPC 协议规定的操作类型需要分别实现一个 Argument 的子类以达到操作指令的协议级别处理，其类设计如图 5-3 所示。

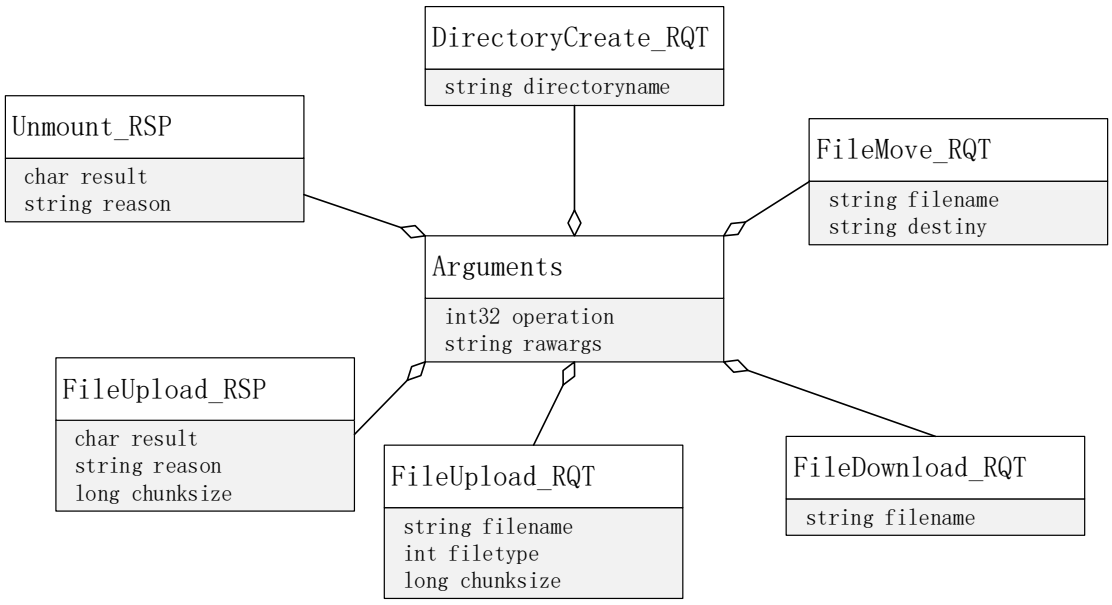

图 5-3 过程参数 Arguments 类图

图 5-3 只列出了部分操作指令涉及的过程参数类，其他操作与这部分类似。命名以_RQT 结尾表示请求过程参数，以_RSP 结尾表示应答过程参数。CFSRPC_ARG_Parser 和 CFSRPC_ARG_Pack 工具函数负责根据操作指令生产出对应的 Arguments 对象并进行参数填充。

3.4.2 节对 CSDA 算法的设计使得 CCD 能够动态计算文件传输分块的大小，CCD 在进行文件传输的操作前会查询 IO Send Queue 和 IO Receive Queue 两个队列的长度，并根据 CSDA 算法的定义进行文件分块大小的计算，作为 CFSRPC_ARG_Pack 函数的第二个参数进行该函数的调用，CFSRPC_ARG_Pack 在制作应答参数的过程中会将指定的 chunksize 装载进 chunksize 字段中。CFSRPC_Util 工具类包含了 CFSRPC 协议的服务端实现，为简化了 CCD 事务处理线程的业务流程实现。

5.2.3 CCD与原CephFS处理流程的接入实现

CCD 与原 CephFS 功能的接入主要是对 CFSRPC 协议支持的所有业务功能的 Ceph 实现例程进行封装和调用。Ceph 官方提供了一个名为 libcephfs 的库，其中包括了对 Ceph 的一系列文件操作的函数封装，CephFS 将 POSIX 定义的文件系统相关的系统调用都做了 Ceph 的实现。Ceph 如此实现是为了当 CephFS 作为一个文件系统挂载到操作系统里时，用户可以像使用本地文件系统一样对 CephFS 进行操作。

CephFS 的实现方式使得用户的所有操作都会经过 FUSE 在内核态进行一个中转，而 CCD 从网络中读取到数据后已然位于用户态，这里 CCD 将 FUSE 的用户态模块代替掉，根据文件操作的不同进行 libcephfs 的不同调用，使得 CCD 与 CephFS 的文件系统服务以及进一步 MDS 的元数据管理服务达到无缝对接。

Task_Handler 函数是事务处理线程的入口，该函数在调用 cfsrpc_rqt_arg 工具函数进行了请求过程参数的解析后，会以一个包装好的 Arguments 对象为参数调用操作指令所对应的具体执行函数。这些事务执行函数会直接调用或组合调用 libcephfs 提供的函数，相当于以一个 façade 设计模式来完成了云盘远程调用的具体处理过程，表 5-1 描述了 CFSRPC 协议定义的请求操作所使用的 libcephfs 库函数及其实现方式。

表 5-2 CCD 事务请求实现方式

操作指令	使用的库函数	实现方式

云盘挂载 DKM	int ceph_mount(string);
	
	string[] ceph_listdir(string);
	
	int ceph_stat(string path, ceph_stat stat);
		首先调用 Mount 函数让 CCD 向 MDS 发送一个挂载操作然后以个人用户的根目录为参数进行打开文件夹操作

云盘卸载 DUM	int ceph_unmount();	断开 CCD 与 MDS 的挂载连接后在断开与客户端的长连接即可

上传文件 FUL	int ceph_open(string path, int flags, mode_t mode);
	
	int ceph_write(int fd, char *buf, int64_t size, int 64_t offset);
		先使用 ceph_open 新建一个文件得到新文件的 fd，然后使用 ceph_write 传入固定大小的缓存空间将业务处理线程从客户端接受到的数据写入 OSD，同时该函数会将文件元数据修改请求发送给 MDS 集群。

下载文件 FDL	int ceph_open(string path, int flags, mode_t mode);
	
	long ceph_read(int fd, byte[] buf, long size, long offset);
		首先根据文件全路径调用 ceph_open 获取到该文件的 fd，然后以文件分块大小作为缓冲区调用 ceph_read 从 OSD 中读取文件数据，读满一个缓冲区后，包装成一个响应报文发送给客户端

打开文件 FOP	int ceph_open(string path, int flags, mode_t mode);
	
	long ceph_read(int fd, byte[] buf, long size, long offset);
		先行进行文件下载操作，然后以下载好的文件的句柄作为参数运行具体的文件打开工具

删除文件 FDT	int ceph_unlink(string path);	以文件删除请求的内的文件全路径为参数调用 ceph_unlink
移动文件 FMV	int ceph_rename(string from, string to);	以文件的原全路径名作为第一个参数，目的全路径名作为第二个参数调用 ceph_rename
新建文件夹 DCT	int ceph_mkdir(string path, mode_t	以要创建的文件全路径

	mode);	名为参数调用 ceph_mkdir
删除文件夹 DDT	int ceph_rmdir(string path);	以要删除的文件全路径名为参数调用 ceph_rmdir

打开文件夹 DOP	string[] ceph_listdir(string);
	
	int ceph_stat(string path, ceph_stat stat);
		以打开的文件夹全路径为参数调用 ceph_listdir，返回一个 String 的数组，存储了用户根目录下的所有文件和目录，对这个目录的每个文件项进行 ceph_stat 调用获取存储在 ceph_stat 结构里的大小、类型等属性信息，最后按照协议的定义进行响应结果的包装和发送

5.3 认证模块实现

认证模块以多进程的模型为 CCD 和管理功能模块提供认证服务和用户管理的持久化。认证服务节点为每个活跃的 CCD 节点和管理功能节点分配一个单独进程，该进程向上与服务请求节点维持一个 TCP 连接提供服务交互，向下与 Mysql Cluster 维持一个数据库连接提供权限信息和管理数据的持久化。认证模块以一个监听程序的形式提供服务，其服务总体流程如图 5-4 所示。

该程序首先建立一个 TCP 网络套接字监听 9012 端口，云盘系统在每添加一个 CCD 或者一个管理服务节点的时候会向该监听进程发送建联请求，监听程序在接受到新的建联请求后会为其创建一个新的临时 TCP 套接字与其保持长连接，然后使用 fork 系统调用新建一个进程，父进程将这个连接 fd 关闭，然后继续监听 9012 端口，这使得子进程单独持有这个连接的 fd，子进程会创建一个与 Mysql Cluster 的连接，然后对从该 fd 发送来的认证服务请求进行串行处理。

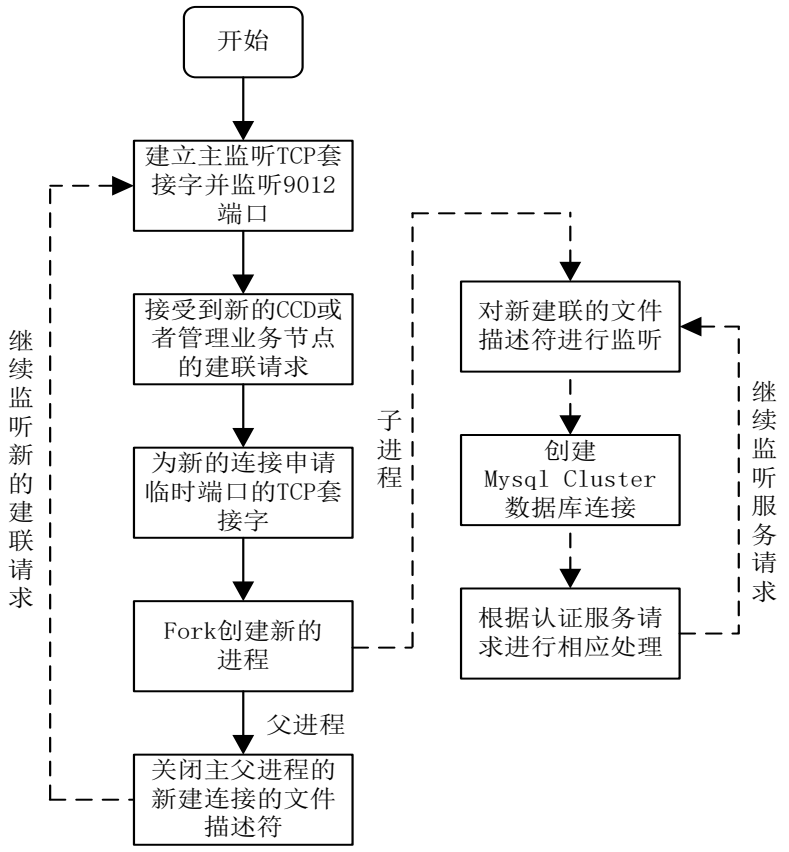

图 5-4 认证模块服务总体流程

CCD 在接受到客户端的远程文件操作服务时，首先就会向认证模块发起该操作的权限认证，这也是 CCD 向认证模块请求的唯一类型服务，请求参数包含请求类型、一个 16 字节的 Token 以及操作代码，格式如下：

```
{
    “rqttype” : “操作权限认证”
    “token” : “16 个字节的 UUID”,
    “operation” : “4 个字节的操作代码”
}
```

图 5-4 中标注的子进程在接受到一个请求后，会根据 rqttype 判断该请求的类型，对于 CCD 发送的操作权限认证请求，直接根据 Token 串从数据库中查询得到该令牌是否存在以及其对应的操作权限是否包含 operation 指定的操作代码，据此判断是否对该操作授权。响应格式如下：

```
{ “authresult” : “0 表示不授权，1 表示不授权” }
```

管理功能模块为管理员提供用户管理功能，为用户提供登陆注销功能，这些都需要依赖认证模块的服务，其通信和处理过程与其对 CCD 的服务流程类似。其中

用户管理功能只涉及到单表的增删改查，业务简单，这里不做赘述。 用户在登陆时，认证模块调用 TokenFactory 工厂类生成一个新的 UUID 作为该用户本次登录会话的认证令牌，返回给用户的同时存入数据库，用户在整个登陆会话中进行的请求都会使用 CFSRPC 协议携带上该认证令牌，当用户注销时，认证模块会将该用户的认证令牌清空。

5.4 管理功能模块实现

管理功能模块为客户端提供了登陆注销、用户管理等功能的 REST API，并基于 Apache2 提供服务。在负载机制下，多个服务节点共存，接受负载层分发的 80 端口的数据包。该模块中 REST API 对象操作的主要命令定义如表 5-3 所示：

表 5-3 管理功能模块 REST API 定义

操作角色	操作	HTTP 方法	URI
用户	登陆	POST	/api/user/login
	注销	POST	/api/user/logout
	用户信息修改	PUT	/api/user/info/*
	容量查询	GET	/api/user/quota
管理员	用户增删	PUT/DELETE	/api/admin/userid
	用户权限设定	PUT	/api/admin/userid/auth
	云盘配额设定	PUT	/api/admin/userid/quota

其中用户的第一次登陆和后续容量查询以及管理员的云盘配额设定都依赖于对 Ceph 集群中 MDS 服务的相应处理，因此，每个管理功能节点都作为一个 CCD 的客户端与其建立一个长连接，且该连接对应的用户的权限被初始化为能够修改所有用户的个人空间及其元数据。

用户第一次登陆时，管理功能节点向 CCD 发送一个新建文件夹的远程操作请求，为该用户建立其根目录，并设置其默认最大容量。

用户请求容量查询时，管理功能节点向 CCD 发送一个配额查询的请求，文件夹路径参数指定为该用户的根目录。

管理员进行云盘配额设定时，管理功能节点向 CCD 发送一个配额设定请求，文件夹路径参数指定为该用户的根目录，配额大小参数指定为管理员请求的配额大小。

5.5 客户端实现

对于云盘系统的服务端而言，客户端会通过两种通信渠道请求服务，其一是通过 HTTP 请求向管理功能模块 CFSRPC 协议无关操作，其二是通过 TCP 长连接向 CCD 节点请求 CFSRPC 协议相关操作。单一的浏览器作为客户端只能满足第一点，第二点则需要 C/S 架构的支持。因此客户端使用 webkit 作为浏览器引擎，同时使用 TCP 客户端来实现远程文件系统操作的请求发送和结果接收。

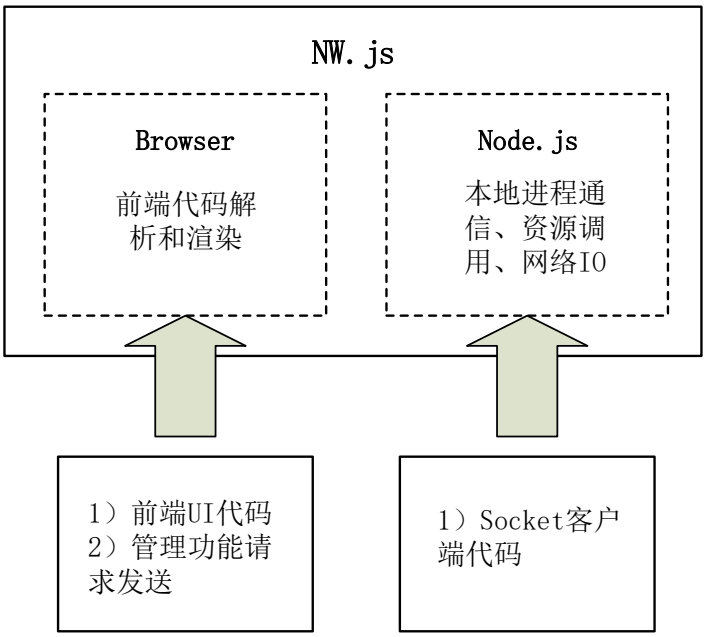

图 5-5 客户端实现架构

客户端使用 NW.js（node-webkit）[39]作为客户端桌面应用程序的实现框架，NW.js 是是 Github 上一个热门开源项目，基于著名浏览器开源项目 Chromium 和 node.js 实现的一个应用程序运行时环境，它能够将 Web 前端的技术置于客户端本地运行，同时 node.js 的集成使得该环境拥有访问本地操作系统资源、库以及网络通信资源的能力。

客户端使用 javascript 语言进行业务逻辑的实现，可视化部分包含 UI 的界面代码，并由 webkit 内核进行渲染，管理功能请求也由前端代码实现。客户端与 CCD 进行的 Socket 连接由 node.js 的 net 模块来实现，其代码描述如下所示：

```
var net = require('net');
var client = net.connect({ port: 9011 }, function() {
    var request = CFSRPC_Util.cfsrpc_rqt_pack()
    client.write(rquest); //向服务器端发送包装好的请求报文
```

```
});
client.on('data', function(data) {
    CFSRPC_Util.cfsrpc_rps_parser() //接受到服务器端发送来的报文并进行
                                    解析
});
```

CFSRPC_Util 工具类为 CFSRPC 协议相关处理的 JS 实现，为请求报文包装和结果报文解析提供函数接口。

5.6 本章小结

本章根据第四章对云盘系统各个模块的分析和总体设计，针对功能和依赖技术对每个模块的详细处理流程和程序架构进行了详细设计和实现。首先对云盘系统的负载机制进行技术选择并针对云盘的服务模型给出了详细配置方案，接着给出了核心存储模块的详细部署流程和功能节点搭配方案，然后对 CCD 进行了详细的设计和实现，包括 CCD 本身的处理架构、CFSRPC 服务端的实现、认证模块接入等等对 CephFS 的二次开发细节，最后由于实现难度较低，分别对认证模块、管理功能模块和客户端模块的详细设计和实现进行了简要的描述。

第六章 云盘系统测试与分析

6.1 测试目的

本章对本文设计实现的基于 Ceph 分布式文件系统的云盘系统进行测试与分析，从功能上测试其业务流程是否完善且可用，性能上测试云盘系统是否能够支撑大量用户的并发操作。

6.2 测试环境

6.2.1 测试环境配置

考虑到有限的测试资源以及云盘系统的功能节点种类较多，同时尽可能模拟云盘系统的生产环境，服务器测试环境使用多台虚拟机进行搭建，其中包括一台负载接入节点 LB Master、2 台 CCD 节点，Ceph 存储集群包含 3 台 OSD 节点、2 台 Monitor 节点以及 1 台 MDS 节点 MDS，认证模块使用单台机器作为服务节点并在该机器以单机模式上搭建 Mysql Cluster，管理功能节点使用单台机器作为服务节点。客户端测试环境使用一台物理 PC，测试机器使用具有多网口宿主机的虚拟机，根据测试需要进行新的测试虚拟机添加和删除，的具体配置信息如表 6-1 所示：

表 6-1 测试环境配置

角色	IP 地址	配置
LB Master	172.20.10.2	CPU：Intel Xeon E5-2620 内存：2GB 硬盘：SATA 100GB 网卡：VMare Virtual Ethernet 操作系统：Ubuntu 14.04 Server
CCD-1	172.20.20.1	
CCD-2	172.20.20.2	
MON-1	172.20.30.1	
MON-2	172.20.30.2	
MDS	172.20.40.1	
AUTH	172.20.50.1	
MGR	172.20.60.1	
TEST	172.20.100.1-4	
Client	192.168.1.6	CPU：Intel Core i3-4150 内存：4GB 硬盘：SATA 200GB

		网卡：千兆以太网卡 操作系统：Window10
OSD-1	172.20.70.1	CPU：Intel Xeon E5-2620 内存：2GB 硬盘：SATA 500GB 网卡：VMare Virtual Ethernet 操作系统：Ubuntu 14.04 Server
OSD-2	172.20.70.2	
OSD-3	172.20.70.3	

云盘系统服务集群所在网络段为 172.20.0.0/16，客户端所在网络段为 192.168.1.0/24，客户端通过负载节点接入，整体部署情况如图 6-1 所示：

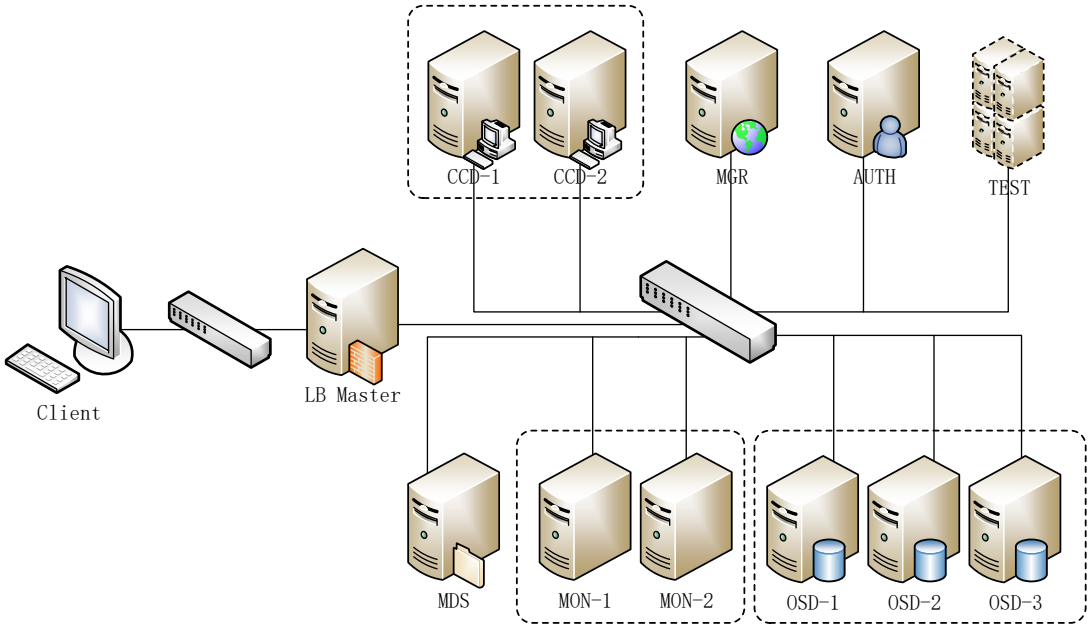

图 6-1 测试集群拓扑

6.2.2 测试环境部署

1）首先为 10 台虚拟服务器安装 Ubuntu14.04 Server 系统，以 172.20.0.0/16 网段按照表 6-1 对其进行网络配置，所有的宿主机拥有千兆网卡且连在一个内部千兆交换机上。为 Client 安装 Window10 系统，按照表 6-1 以 192.168.1.0/24 网段配置 Client，将 LB Master 所在的宿主机的第二个网卡设置为 192.168.1.0/24 网段，然后将为 LB Master 虚拟化一个新的网卡桥接在这个物理网卡上。

2）使用 ceph-deploy 为 OSD-1、OSD-2、OSD-3 部署 Ceph OSD 服务，每台节点指定单块 500G 的机械硬盘，为 MON-1、MON-2 部署 Ceph Monitor 服务，为

MDS 部署 Ceph Metadata 服务。

3）为 CCD-1 和 CCD-2 安装 CCD 服务，为 MGR 安装 Apache2 并搭载 MGR 服务，为 AUTH 安装认证服务和单机模式 Mysql Cluster。

4）按照 5.1.1 节的部署流程为 LB Master 安装单台 LVS+Keepalived 负载接入服务。

5）为 Client 安装客户端程序。

6.3 测试及结果分析

6.3.1 功能测试

首先进行登陆测试，使用测试账号和密码进行登陆，测试云盘系统的登陆认证流程，登陆界面如图 6-2 所示：

图 6-2 登陆界面

成功登陆后可以进入云盘客户端的操作主界面，整个界面由上方的菜单栏、左边的导航栏和中央的文件操作视图组成，菜单来右边的搜索框支持用户在个人云盘内进行文件搜索操作，左边的导航栏可以对文件的分类进行筛选或者进入回收站，当前版本只包括对用户的收藏文件进行筛选，在文件操作视图可以看到 Test 用户的个人空间下的文件结构，在这里可以对文件和目录进行重命名、上传、下载、删除操作，单击新建按钮可以进行文本文件和文件夹的新建操作，具体如图 6-3 所示。

图 6-3 云盘主操作界面

单击如图 6-4 所示的按钮可以进行文件上传操作，此处以一个 5.7G 的电影文件作为测试文件。

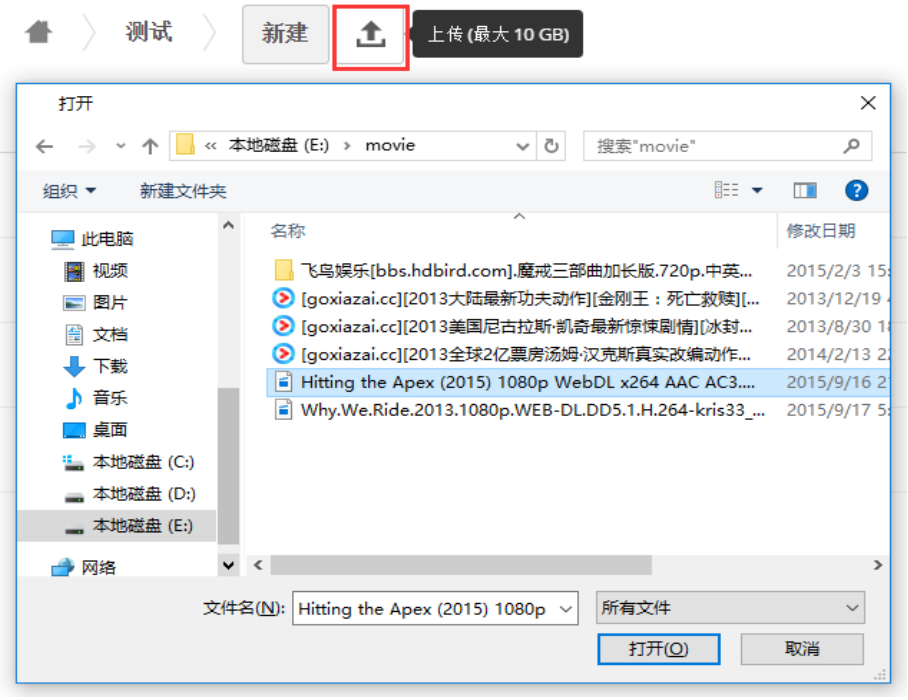

图 6-4 上传文件

上传过程中以一个随时间动态增长的矩形来呈现文件上传的进度，如图 6-4 所示：

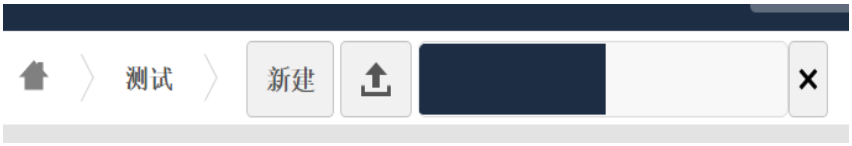

图 6-5 文件上传进度

文件上传成功后会马上显示在主视图当中，选中一个文件后可以在主视图的多处进行删除操作和下载操作，如图 6-6 所示：

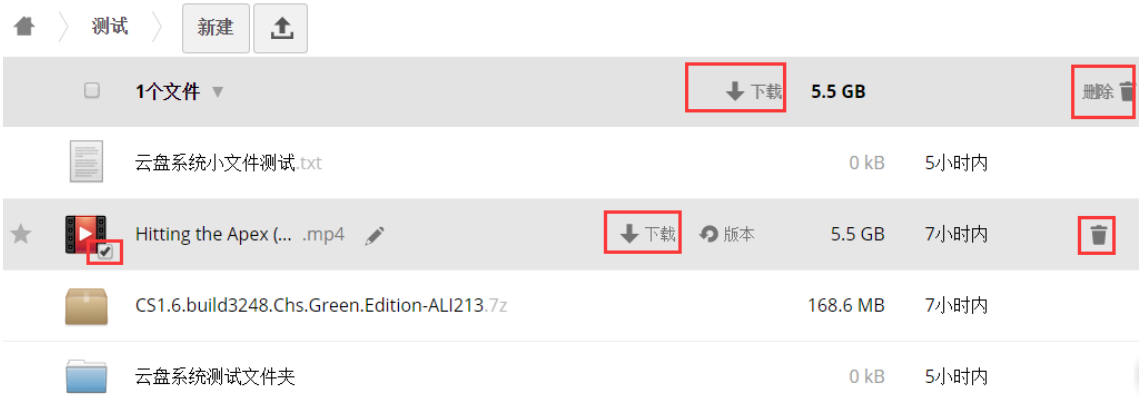

图 6-6 文件下载和删除

6.3.2 性能测试

6.3.2.1 可靠性测试

整个测试集群包含单个接入节点 LB Master、两个 CCD 节点、两个 Ceph Monitor 节点和 3 个 OSD 节点，存储集群的可靠性表现在当 Monitor、OSD 或者 CCD 出现单点失效的情况下能够保证集群的功能可用性，在客户端保持连接并进行各种操作的同时，进行三种节点的手动故障模拟，以测试集群的稳定性和容错性。测试用例及结果如表 6-2 所示。

表 6-2 可靠性测试用例及结果

用例编号	前置用例编号	测试步骤	期望	结果
1	无	客户端登陆后，新建文件 test，断开 CCD-1 的网络连接	客户端保持登陆状态，且可以正常看到文件 test，并允许对其进行修改	通过
2	1	恢复集群状	客户端保持登陆状	通过

		态，断开 OSD-1 的网络连接	态，且可以正常看到文件 test，并允许对其进行修改	
3	1	恢复集群状态，断开 MON-1 的网络连接	客户端保持登陆状态，且可以正常看到文件 test，并允许对其进行修改	通过
4	1	恢复集群状态，使用客户端上传 1G 大小的文件，在传输过程中，断开 CCD-1 的网络连接	传输过程短暂停顿后继续，直到传输成功	通过
5	2	恢复集群状态，使用客户端上传 1G 大小的文件，在传输过程中，断开 MON-1 的网络连接	传输过程无显著变化，持续到传输完成	通过
6	3	恢复集群状态，使用客户端上传 1G 大小的文件，在传输过程中，断开 OSD-1 的网络连接	传输过程短暂停顿后继续，直到传输成功	通过

测试结果表明，当允许横向扩展的 CCD、MON、OSD 服务节点出现单点失效时，存储集群能够保持功能的可用性，同时，生产环境中，大量的 OSD 节点以及数据的冗余量能够保证存储服务的超高可用性。

6.3.2.2 CCD单点压力测试

考虑到千兆的测试网络环境，生产环境下用户的网络带宽，负载均衡器对接入服务的分发，直接在内网环境下对客户端进行压力测试十分困难且不符合普遍线上用户的带宽设定，因此本节通过使用 4 台独占千兆网卡的测试虚拟机同时对云盘挂载节点 CCD 进行压力测试得到标准硬件下的单台 CCD 节点的性能上限，从而能够根据硬件的升级程度和 CCD 节点的扩展程度，再结合业界成熟的云盘系统的用户体验对云盘系统的性能瓶颈以及能够支持的用户数量级进行一个评估[40]。

表 6-3 CCD 单点文件上传性能测试结果

Test Node	2 Threads		4 Threads				8 Threads							
TEST1	8.4	8.8	4.5	5.4	4.4	5.3	3.0	3.0	3.0	2.7	2.9	2.9	2.9	2.9
TEST2	7.2	7.8	5.3	4.2	5.2	5.2	2.9	3.0	2.8	2.7	2.9	2.9	2.9	2.9
TEST3	8.8	8.4	4.6	4.6	4.5	5.4	2.8	2.7	2.6	2.7	3.0	2.8	2.7	2.6
TEST4	10.2	9.8	5.2	5.2	5.1	4.9	2.9	3.0	2.9	3.0	2.9	3.0	3.0	3.0
Total	69.4		79.0				92.8							

测试机器使用测试集群的 TEST 服务节点内的 4 个虚拟测试服务器，以一个全权限用户的方式接入 CCD1，然后以多线程的方式对该节点进行文件上传操作，文件上传测试结果如表 6-3 所示，单位统一为 MB/s。可以看到随着并行线程的增加，整体的写速率显著提高且逼近网络带宽。当只有单个实例进行文件上传操作时，速度可以达到 50M/s 左右。由于 Ceph 的文件分块存储和分块冗余的存在使得文件下载操作必然快于文件上传操作，CCD 的单点性能瓶颈不会受制于文件下载操作的速度，因此文件上传操作的测试结果可以表示为 IO 型事务的最低速度。

根据以上测试结果和分析结果，在千兆带宽的网络环境下，将单点 CCD 在不同数量实例并行操作的条件下能够为每个实例提供的 IO 事务处理速率描述为图 6-7。

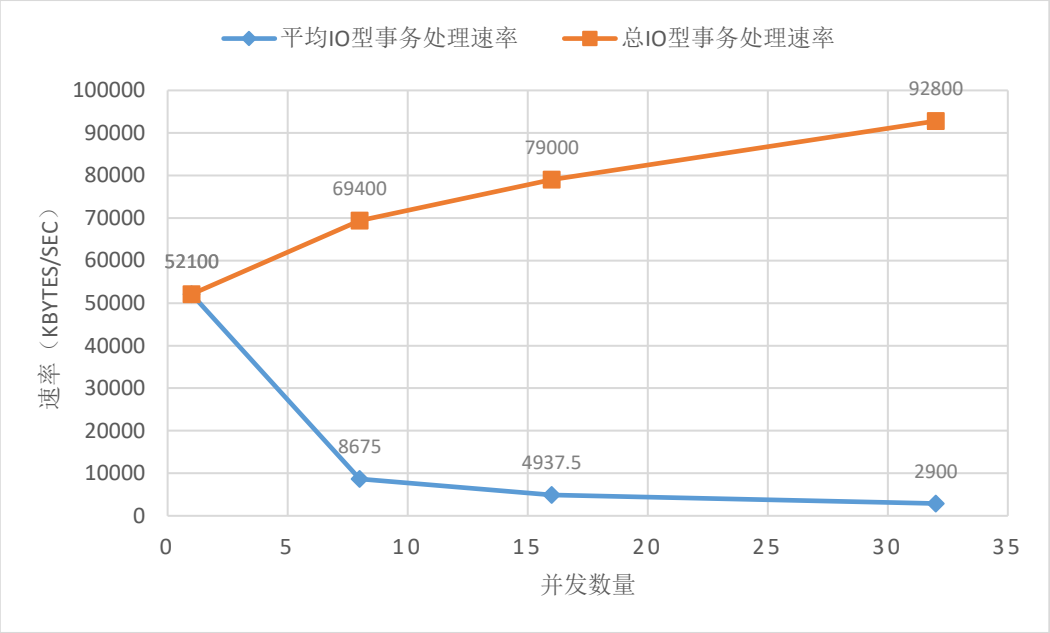

图 6-7 并发实例数量对单点 CCD 吞吐量的影响

从图中可以看出，随着并发数量的增加，CCD 总 IO 型处理速率接近网络带宽，这表示在 IO 事务密集的情景下，CCD 能够很好的发挥自身的运算资源最大化带宽利用率，平均 IO 型事务处理速率随着并发数量的增加而减少，该测试 CCD 在同时处理 32 个 IO 事务时可以保证每个事务的处理速度达到 2.9MB/s，以该图的趋势，当并发事务越多，总的处理速度越趋近带宽，假设生产环境的网络传输最大速度为 100MB/s，如果只需要满足每个用户平均能够得到 200KB/s 的云盘操作速度（市面上的绝大多数云盘产品的普通用户的体验速度），以该测试 CCD 节点的配置理论上能够支持 512 个用户同时进行文件传输操作。而生产环境下的 CCD 节点的硬件配置可以做到该测试节点配置的十倍的数量级，又因为大量保持连接的客户端之中只有很小一部分在进行 IO 事务请求，这意味着每个 CCD 节点能够维持的云盘挂载连接的数量是一个很大的量级，而 CCD 支持横向扩展，这使得云盘系统的接入模块完全能够达到公有云环境下的并发要求。

6.3.2.3 传统单机存储对比测试

由于测试集群的规模限制，为测试出传统单机存储和本文所述云盘系统的性能差异，单机的测试方法使用 ftp 服务器作为单点存储主机，分布式云盘存储使用测试集群，并分别使用一台和两台 CCD 作为接入节点，分别对 1、2、4 个测试实例的同步上传操作做测试，表 6-4 为该对比测试结果，每一项表示每个实例的上传速

度，单位为 MB/s。

表 6-4 传统单机对比测试结果

测试条件	1 instance	2 instances		4 instances			
单台 ftp 存储节点	98	40	62	19	30	18	25
测试存储集群（1 台 CCD 节点）	71	51	37	15	17	20	16
测试存储集群（2 台 CCD 节点）	69	76	69	32	36	41	34

单实例进行存储操作的时候，单台 ftp 服务器能够被一个实例完整占用网络带宽，达到 98MB/s 的上传速度。测试存储集群在单实例的情况下没有优势，当实例数量增加，单台 ftp 服务器受限于单机有限网络带宽和处理能力，每个实例能够享有的上传速度大幅度下降，当进行单台 CCD 接入节点来处理上传服务时，单机的本质并没有改变，额外的协议处理和分布式存储流程导致其速度慢于传统的 ftp 服务器，而当接入节点扩充到 2 台时，多实例的服务质量有了显著提升。

CCD 节点横向扩展能力使得在生产环境下能够支持大量用户并发进行存储操作，同时能够保证每个用户的存储服务质量。

6.4 本章小结

本章对云盘系统的所有功能模块进行了最小化的测试环境部署，描述了部署过程和测试环境的拓扑结构，并基于该环境进行了云盘系统的功能测试，然后对支持横向扩展的服务进行可靠性测试，同时通过对云盘的 CCD 节点进行单点性能测试并与传统单机存储做了对比测试，总结出云盘系统的整体性能表现，确认其足以满足公有云环境的性能要求。

第七章 总结与展望

7.1 本文研究成果

本文以公有云环境下的个人云存储作为研究课题，分析并提出了公有云盘系统所依赖的关键技术，并对这些技术进行了源码级别的研究和分析，基于云盘系统的通信需求设计了 CFSRPC 协议，以 Ceph 分布式文件系统作为核心存储模块，使用二次开发的方式设计了新的 Ceph Client（CCD），并加入 CFSRPC 协议的接入实现和独立的认证流程，继而完成了基于 Ceph 分布式文件系统的云盘系统的设计与实现。

在 CFSRPC 协议的设计过程中，首先定义了远程调用协议的标准格式，并为其加入了个人云盘操作的协议化支持，使得客户端对云盘的操作可以在协议层进行解析，然后在协议首部纳入了事务编号、认证令牌、任务类型等参数项，使得该协议具备事务管理、安全认证和任务按类型排队的功能，最后为 CFSRPC 协议设计了 CSDA 算法，使得服务端的协议实现能够根据待处理事务队列、宿主机可执行最大线程数等负载因子进行文件传输分块大小的动态调节。

云盘系统使用 Ceph 分布式文件系统提供底层存储支撑，首先对 Ceph 的各个组件及其架构进行了原理性研究，提出了云盘系统的核心存储架构，然后设计实现 CCD 作为新的 Ceph 接入模块，在 CCD 设计过程中，研究提出了它的面向协议的工作模型、业务流程和基于多路 IO 复用的处理架构，在 CCD 实现过程中，完成了其核心处理架构的实现，加入了 CFSRPC 协议的解析包装流程和云盘操作的权限认证流程，实现了 CCD 与原 CephFS 组件 POSIX 存储流程的无缝优化接入。

云盘系统的整体解决方案还包括为 CCD 模块和管理功能模块设计实现的负载接入机制、基于多进程模型的独立的认证模块、基于 NW.js（node-webkit）实现的跨平台客户端。

论文在最后对云盘系统的整套组件进行了测试环境的配置和部署，并按照文件系统操作需求对其进行了功能测试，然后对 CCD 进行单点压力测试，结合业界成熟的云盘系统的用户体验对云盘系统的性能瓶颈以及能够支持的用户数量级进行了评估。

7.2 后续工作展望

本文提出的云盘系统能够满足公有云环境下个人云存储的基本功能和性能，但对于整体系统而言仍然存在值得优化和设计的地方，未来考虑从以下几个方面对其进行研究：

1）Ceph 分布式文件系统支持使用 pool 对文件对象进行统一配置，其内容包括对象大小、PG 个数、冗余数量等等，对 CCD 进行升级使其能够根据不同的 pool 对不同大小的文件进行不同的定制，优化云盘的储性能[41]。

2）设置文件访问热度值，根据文件的访问频率来优化 CRUSH 算法对文件的分布，使得高频访问文件能够拥有更快的读取速度[42]。

3）识别用户上传的文件是否已经存在，对已存在的文件的上传操作进行引用加一的优化处理。

4）设置共享云盘空间，为用户提供文件共享功能。

致 谢

研究生三年即将结束，离开的日子已然近在咫尺，回望读研的三年时光，繁重的学习任务和紧凑的研究工作在带来辛苦的同时也带来了满囊的收获，在这里，首先我要感谢我的导师向艳萍教授，是她在我研一最迷茫的时候给我指点了方向，帮助我树立了符合我自身兴趣点的工程观和学术观。三年来我所得到的每一分收获每一分成长都离不开向老师对我的指导和关心。

我还要感谢实验室的每一位老师，在我三年的工作学习生活中，无时无刻不受他们的人格、工程能力、学术修养的影响，一路走来，不知有多少难关都是在他们的帮助和督导下度过。同时，我也要感谢实验室里这群机智可爱师兄弟，在工作上我们互相扶持、互相监督，在生活上我们互相帮助、互相爱护，是他们为我的研究生生活增加了色彩和乐趣。

最后，我要感谢我的父母，感谢他们给予我还算聪明的头脑，在我学习生涯当中为我扬帆护航。

参考文献

[1] Giampaolo D B, De Atley D, Watson M G, et al. CLOUD STORAGE: US, WO/2012/167108[P]. 2012.

[2] Richard B, Chalon D. Distributed file system: US, EP1387296[P]. 2004.

[3] 陈志刚, 李登, 曾志文. 分布式系统中动态负载均衡实现模型[J]. 中南大学学报(自然科学版), 2001, 32(6):635-639.

[4] 李春林, 王丽芳, 蒋泽军,等. 基于身份认证技术安全体系的研究[J]. 微电子学与计算机, 2005, 22(4):8-11.

[5] Hayes B. Cloud computing[J]. Communications of the Acm, 2008, 51(7):9-11.

[6] 任宇宁. 云计算时代的存储技术——云存储[J]. 科技传播, 2012(3):256-257.

[7] Amazon S3[J]. 2010.

[8] Czarnecki J. dropbox[J]. 2013.

[9] Lamont I. Google Drive & Docs in 30 minutes[J]. Good Housekeeping, 2013, 184(1):263-296.

[10] Jr E J W, Goland Y Y. WebDAV[M]// ECSCW '99. Springer Netherlands, 2002:291-310.

[11] Dalrymple J. OmniFocus[J]. Macworld, 2008.

[12] Weil S A, Brandt S A, Miller E L, et al. Ceph: a scalable, high-performance distributed file system[C] Symposium on Operating Systems Design and Implementation. USENIX Association, 2006:307-320.

[13] 刘莎. 基于对象存储的 Ceph 分布式文件系统的研究[D]. 杭州电子科技大学, 2016.

[14] 马如悦. Ceph:一个可扩展的高性能分布式文件系统[J]. 程序员, 2011(3):95-97.

[15] Liang X Y, Guan Z C. Ceph CRUSH Data Distribution Algorithms[J]. Applied Mechanics & Materials, 2014, 596:196-199.

[16] 王曰芬, 章成志, 张蓓蓓,等. 数据清洗研究综述[J]. 现代图书情报技术, 2007(12):50-56.

[17] 李杨, 李曙东, 黄亮. Samba 服务器的研究与应用[J]. 中国水运月刊, 2011, 11(6):90-91.

[18] Shepler S, Callaghan B, Robinson D, et al. NFS version 4 Protocol[J]. Ousterhout, "Caching in the Sprite Network File System," ACM Transactions on Computer Systems 6(1, 2000.

[19] Glass B. Understanding netBIOS[J]. Byte, 1989, 14(1):301-306.

[20] Shi F, Zhou Y. VFS kernel mechanism in Linux system[J]. Journal of Shaanxi Normal University, 2005, 33(1).

[21] Remote File System[J]. 2010.

[22] Wehrle K, Pahlke F, Ritter H, et al. Linux Network Architecture[C] Prentice-Hall, Inc. 2004.

[23] Jim Gray, Andreas Reuter[M].Transaction Prossessing: Concepts and Techniques. 2004.20-50

[24] Netze H. Re: SAMBA im VPN | Forum - heise online[J]. Heise Zeitschriften Verlag.

[25] Bershad B N, Anderson T E, Lazowska E D, et al. Lightweight remote procedure call[J]. Acm Transactions on Computer Systems, 1990, 8(1):37-55.

[26] 蒋振宇. 网络编程中的字节序处理[J]. 华南金融电脑, 2004, 12(2):74-75.

[27] Tene G, Wolf M A, Click C N. CPU utilization metering on sytems that include multiple hardware threads per core: US, US 20070288728 A1[P]. 2007.

[28] Gould J, Smith D, Pei M. Shared registration multi-factor authentication tokens: US, US8769655[P]. 2014.

[29] Richardson L, Ruby S. Restful web services[J]. O'reilly Media Inc, 2007, 4(Sept):199-204.

[30] Thalmann L, Ronstrom M. MySQL cluster architecture overview[J]. 2004.

[31] Zhang W. Abstract Linux Virtual Server for Scalable Network Services[J]. Ottawa Linux Symposium, 2008.

[32] 吴一民,刘伟安. 基于 Fuse 的用户态文件系统的设计[J]. 微计算机信息,2010,(06):159-160+168.

[33] Miller F P, Vandome A F, Mcbrewster J, et al. File descriptor[M]. Alphascript Publishing, 2011.

[34] 史蒂文斯.UNIX 网络编程[M].北京人民邮电出版社,2015,650-670

[35] 崔滨, 万旺根, 余小清,等. 基于 EPOLL 机制的 LINUX 网络游戏服务器实现方法[J]. 微计算机信息, 2006(7X):64-66.

[36] 李昊, 刘志镜. 线程池技术的研究[J]. 现代电子技术, 2004, 27(3):77-80.

[37] Cui B, Xi T. Security Analysis of Openstack Keystone[C] International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. IEEE, 2015.

[38] 史明昌, 孙成宝, 曹刚,等. 一种识别码的生成方法和装置以及识别码应用方法:, CN 102298716 A[P]. 2011.

[39] 王文睿. node-webkit:HTML5 桌面应用运行环境[J]. 程序员, 2014(1):127-129.

[40] 吉慧. 云存储产品性能测试方法研究[J]. 电信技术, 2013, 1(7):7-10.

[41] 张毕涛, 辛阳. 基于 Ceph 的海量小文件存储的优化方法[C] 中国通信学会学术年会. 2014.

[42] 穆彦良, 徐振明. Ceph 存储中基于温度因子的 CRUSH 算法改进[J]. 成都信息工程学院学报, 2015, 30(6):563-567.

硕士学位论文

MASTER THESIS