

中国科学技术大学

专业硕士学位论文

(专业学位类型)



基于 ceph 的云存储运维系统的设计 和实现

作者姓名：

专业领域：

软件工程

校内导师：

企业导师：

完成时间：

二〇一七年四月二十日

University of Science and Technology of China

A dissertation for master's degree

(Professional degree type)



**Design and implementation of
cloud storage operation and
maintenace system based on
ceph**

Author:

Speciality: Software Engineering

Supervisor:

Advisor:

Finished time: Apr 20nd, 2016

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外,论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名: _____

签字日期: _____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一,学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权,即:学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版,允许论文被查阅和借阅,可以将学位论文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。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

☐公开 ☐保密 (____年)

作者签名: _____

签字日期: _____

摘 要

随着网络快速发展，数据急剧增加，采用传统存储技术存储数据，将会面临存储运维复杂、建设成本高和扩容有限等问题。基于 ceph 的云存储系统，将网络中相互独立、型号各异和价格低廉的存储设备统一组织起来，并对外提供统一存储和管理接口，形成具有价格廉价、系统稳定和易维护特性的云存储系统。

基于 ceph 的云存储系统的核心，是应用软件系统和存储设备的结合。ceph 分布式系统将网络中各式各样的存储介质组织起来，对外提供统一存储接口。云存储系统中，对存储设备的管理，由 ceph 分布式系统完成。随着数据的急剧增加，基于 ceph 的云存储系统，可以轻松实现磁盘扩容。通过系统内部实现机制，实现数据高并发存取，消除单点故障节点。云存储系统降低了对硬件的依赖，可以让不同类型和性能的磁盘协同工作，并提供统一管理，降低了存储系统建设成本。ceph 分布式系统具有故障磁盘自动检测和数据自动恢复功能，降低了存储运维成本。基于 ceph 的云存储运维系统，用于检测 ceph 云存储运行状态，及时排除系统故障设备，为用户提供稳定、高效和廉价的存储服务。

基于 ceph 的云存储运维系统，实现了对云存储系统的监控、数据的均衡操作，方便维护人员及时发现、更换故障设备，保证各个磁盘负载均衡，提高云存储系统运行性能。基于 ceph 的云存储运维系统被应用在苏州银行的 TDcube 项目中。

关键字：存储，稳定，高效，廉价

Abstraction

With the rapid development of network, the data has dramatically. Using traditional storage technology to store data, will face the problem of complex storage operation, high cost of maintaining and limit of expanding capacity. The cloud storage system based on ceph unite inexpensive storage devices which hava different models and independent of each other. On the other hand, it provides a unified external storage and management interface. The storage system has the qualities of backup, disaster recovery and easy maintenance.

The core of cloud storage system based on ceph is the combination of application software system and storage devices. Ceph distributed system organizes all kinds of storage media in the network, and provides a unified storage interface. In cloud storage system, the ceph system manages storage devices. With the rapid increase of data, cloud storage system based on ceph, can easily achieve disk expansion. Through the realization mechanism of the system, we can realize the highly concurrent access for data and eliminate the single node. The system reduces the dependence on the hardware, allows different types and performance of the disk to work together and provide unified management, reducing the cost of system construction. Ceph distributed system has the function of automatic detection and automatic data recovery, which reduces the cost of storage operation and maintenance. Based on the ceph cloud storage operation and maintenance system detects the status of ceph cloud storage, to provide users with a stable, efficient and inexpensive storage services.

Cloud storage operation and maintenance system base on ceph has achieved the monitor for the system of cloud storage, balanced operation for data. People can detection and replacement of faulty equipment and ensure that each disk load balance. Cloud storage operation and maintenance system base on ceph is applied in the TDcube project of the Bank of Suzhou.

Key Words: storage, stability, efficiency, inexpensive

目 录

第 1 章 绪 论	1
1.1 选题的依据与意义	1
1.2 国内外研究发展状况及发展趋势	2
1.2.1 云储存运维系统的发展	2
1.2.2 国内外云存储的应用和趋势	5
1.3 云存储系统设计的先进性和意义	6
1.4 本人主要工作	7
1.5 论文组织	10
第 2 章 理 论 基 础	12
2.1 云存储	12
2.1.1 云存储出现场景	12
2.1.2 云存储的关键技术	13
2.2 ceph 机制	15
2.2.1 集群框架	15
2.2.2 存储机制	17
2.3 总结	19
第 3 章 需 求 分 析	20
3.1 系统概述	20
3.2 系统总体架构	21
3.2.1 业务流程	21
3.2.2 系统构造	23
3.3 云存储运维系统功能需求分析	24
3.3.1 账户管理	24
3.3.2 功能需求	24
3.4 云存储运维系统性能需求分析	26
3.4.1 高性能需求	26
3.4.2 高可用性需求	27

3.4.3 高扩展性需求	28
3.5 总结.....	28
第4章 系统设计.....	29
4.1 系统架构设计.....	29
4.1.1 前端架构设计	29
4.1.2 后端架构设计	29
4.2 云存储底层设计.....	30
4.3 总结.....	31
第5章 系统的部署及管理	32
5.1 云存储环境搭建和部署.....	32
5.1.1 ceph 存储系统的搭建和部署	32
5.1.2 calamari 框架的搭建.....	33
5.1.3 Java 框架的搭建	34
5.1.4 前端框架的搭建	35
5.2 系统代码的管理.....	35
5.3 总结.....	37
第6章 系统实现和测试	38
6.1 开发工具及环境.....	38
6.2 云存储运维系统的实现.....	38
6.2.1 前期准备	38
6.2.2 运维系统的实现	38
6.3 云存储运维系统的功能测试.....	40
6.3.1 测试需求分析	40
6.3.2 测试方法和结果	41
6.4 总结.....	43
第7章 总 结	44
参 考 文 献.....	46
附 录.....	48

致 谢.....	51
----------	----

第1章 绪 论

通过对存储的需求，引出本次论文的选题，并对所涉及到技术的发展和趋势进行了概述和分析。本章还包含笔者在本次选题项目中的主要工作，以及对论文的整体组织结构的概述。

1.1 选题的依据与意义

个人需求：当今社会，互联网几乎已在各个国家都得到了普及。数据信息，比如照片、电影和工作文件等数据资料自然而然的出现，这就需要存储设备去保存这些信息。随着信息社会快速的发展，人们对存储的需求不断扩大，对他们的要求也不断提高。随着数据信息的体积也越来越大，U 盘、光盘和硬盘等凭借大容量、相对体积小、价格适中和存储稳定等特性轻易的战胜了传统软盘，成为当代青年日常生活必需品。随着网络磁盘的到来，很多人又将目光转移到了网盘。它相对于当代实体存储介质，有不可比拟的优点。当采用网盘存储数据时，无论在哪，只要能联网，就随时可用联网终端设备，如：电脑、手机和平板等存取、编辑和查看自己的资料。也为资料共享带来了极大的便利。很多网盘，比如 google 云端硬盘、百度网盘和 Amazon Cloud Drive 等都提供免费的网盘。对于普通用户，完全可以满足日常需求。这是云存储给个人带来的便利。云存储为人们提供了一种和物理介质存储设备一样，具有存储功能的另一种存储方式。同时，云存储也避免出现因物理设备的损坏或丢失而造成不可恢复的灾难。当代网络的普及使得云存储变得异常方便，通过手机终端即可实现数据的存取。相当于手机拥有了无限大的存储空间，极大方便用户对存储空间的需求。用户通过各个终端访问自己的云存储空间时，可以查看各个资源的使用情况，就是云存储运维系统的一个应用。

大型应用：随着云计算和物联网的兴起，智能应用成为当今一大主题。智能应用主要涉及到信息的收集、整理和应用。因涉及范围广、需求各异，会形成海量、复杂的信息数据。这些信息如果还是利用传统存储模式，将信息存储在由政府、企业提供的孤立的存储介质上，那这种存储方式的代价将会非常高^[1]。随着后期数据量和数据种类的不断增加，对磁盘管理起来会越来越复杂。在采购存储介质方面，前期面临着存储介质被闲置的浪费，后期因为设备紧张，还要有专门人员负责购买昂贵的储存介质。在扩容时，数据的安全性和可用性也是一个要面临的严峻问题，使得政府、企业前期投入特别大，后期管理比较

复杂^[2]。同时，需要配备有专门的人员去维护。存储介质的升级换代也是个很大的问题。信息的可靠性也无法得到保证。综合各种因素，当采用普通的，孤立的存储介质，将会花费高昂的代价，效果也不理想。这些种种弊端，通过云存储系统可以很好的解决。云存储系统可以按需分配磁盘空间，后期可以轻松扩容，在数据安全上也会有很好的保证，有专门人员负责管理。不会存在因一次性购买大量存储介质而造成浪费，也不用为扩容时去买新的存储介质。花费代价上远小于自己独立管理。在磁盘空间扩容时，云储存可以做到几乎不被用户感知而实现磁盘扩容。政府或企业可以通过云储存运维系统，时刻关注当前已有资源的使用情况，如可以通过观察一段时间内系统 IOPS（Input/Output Operations Per Second，每秒读写操作次数）或带宽等信息，确保系统资源满足当前用户需求。

例如阿里云协助 12306 系统示例。阿里云在 2014 年 1 月 16 日透露，阿里云为 12306 购票系统提供技术服务，承担购票时 75% 的查询流量。春运期间，峰值流量时可以超过平时流量的 10 倍之多。如果采用传统的 IT 解决方案，为春运购置大量硬件设备。在春运结束后，这些大量设备将面临被闲置的巨大浪费。如果在春运期间，峰值流量超过预期，网站将面临着瘫痪。因为大规模的服务器的采购、上架、部署和调试需要一两个月的时间，对于用户需求来说，这种方案是不可行的。利用阿里提供的云服务，避免了在购置服务器时的巨额投资。12306 购票系统可以通过云存储运维系统，时刻监控当前资源使用情况，并根据需求按量购买服务。在国庆和春运期间，会出现两次大型扩容，每次的扩容，都是在分钟级的时间内完成。结束时，又可以轻松释放掉不必要的资源，从而将成本降到最低，为 12306 购票机构节约了成本、提供了便利。

通过云储存运维系统，可以实时监控资源使用情况，并根据资源使用情况，及时采取必要的措施，让云存储系统提供更加优质的服务。

1.2 国内外研究发展状况及发展趋势

1.2.1 云储存运维系统的发展

ceph^[3]是一种具有高性能、高扩展性和高可用性的分布式存储系统。回顾历史，分布式存储系统的发展可以分为四个阶段。

第一阶段发生在 20 世纪 80 年代，随着以太网技术的快速发展，用户希望通过互联网技术，实现客户端和文件服务器间文件共享。在此期间，主要的应用包括 CMU (Carnegie Mellon University, 卡内基梅隆大学) 与 IBM (International

Bussiness Machines Corporation, 国际商业机器公司) 合作研发的 AFS (Andrew File System, Andrew 文件系统) 文件系统和 SUN (Sun Microsystems, 太阳微系统) 公司研发的 NFS (Network File System, 网络文件系统) 文件系统。

第二阶段发生在 20 世纪 90 年代, 在这个时代, 存储系统独立于计算机系统而快速发展。SAN (Storage Area Network, 存储区域网络) 在这个时代出现。当时研究的重点是实现存储系统的扩展性和文件共享。当时主要产品包括 IBM 研发的 GPFS (General Parallel File System, 通用并行文件系统) 和 Red Hat 公司的 GFS (Global File System, 全局文件系统)。

第三阶段发生在 21 世纪 00 年代, 随着计算机技术和网络的快速发展, 对存储系统容量的扩展性和存取性能, 提出了更高的要求。对象存储技术和如何提高数据并发性能是当时的研究重点^[4]。在此期间, 分布式存储系统大量出现, 包含 PVFS (Parallel Virtual File System, 并行虚拟文件系统)、Panasas、Lustre、GFS (Google File System, 谷歌文件系统) 和 ceph。分布式系统 ceph 登上历史舞台。

第四阶段发生在 21 世纪 10 年代, 随着大数据和云计算的兴起, 数据呈爆炸式增长。据研究显示, 2020 年人类存储的数据量将达到 40ZB, 相对于 2009 年的 0.8ZB, 数据增加了 50 倍。此时, 云存储应用而生。云储存具有弹性扩容、高性能、高可用、多租户和 QoS (Quality of Service, 服务质量) 等众多特性。研究重点放在了对 EB 级数据的存储^[5]。很多分布式文件系统, 将方向瞄准了云储存, 如 Nexenta、Nutaix、Gluster 和 ceph。在此阶段, 实现了云存储和云计算的再次融合, 美其名曰软件定义存储。

与开源存储软件 ceph 类似的还有 Red Hat 旗下的 Cluster。通过对比着两款系统, 来了解基于 ceph 的云储存系统的发展现状。

ceph 和 Gluster 在存储原理上, 有本质的不同, ceph 通过一个名为 RADOS (Reliable Autonomic Distributed Object Store, 可靠自治分布式对象存储) 的对象存储系统, 实现将数据以块、对象和文件的形式进行存储。Gluster 则是使用哈希算法计算出存储池对应的存储位置。所有的服务器都是用哈希算法对数据进行定位, 没有中心元数据单点, 这点和 ceph 很像, 避免出现像早起 Hadoop 上出现的访问瓶颈。在扩展性方面, 两者都避免了传统存储模式中限制扩容的单点故障, 因此可以近似线性扩展。

在数据恢复方面, ceph 具有更大的优势。相对于 Gluster, ceph 将数据存放在更多大的节点集群中。当出现故障磁盘时, 有更多的拥有故障磁盘对应数据副本的备份磁盘一起帮助数据恢复, 缩短了数据恢复时间。同时, 不会因数据恢复而使某个磁盘负载过大。

在选择性方面，两者具有较强的优势。首先，两者的源代码是开源的。其次 Inktank 和 Red Hat 公司分别为 ceph 和 Gluster 提供支持服务和管理工具包。相对于传统的存储，随着存储介质价格的不断下降，ceph 和 Gluster 的优势越来越大。

基于 ceph 的云存储系统，它在 ceph 分布式系统和云计算的基础上发展而来。通过集群的应用，将网络上不同的存储介质统一组织起来，并对外提供存储功能的一个新型存储方式^[6]。相对于传统的孤立存储，基于 ceph 的云存储系统能提供更加便携、安全、可靠和高效的性能^[7]。通过基于 ceph 的云存储系统的优势和劣势来分析基于 ceph 的云存储系统的现状。

优势：基于 ceph 的云存储系统可以按客户需求，分配对应存储空间。随着后期需求的变化，可以及时方便的进行扩容或缩容，且在对设备进行扩容或缩容时，给客户带来良好的体验，全程操作基本不会影响客户正常业务，节省了数据维护方面的开销^[8]，同时也节省升级服务器的费用；在数据安全方面，云存储会自动检测故障磁盘，当出现故障磁盘时，系统会根据预先设计的规则，将故障磁盘上的数据自动迁移到正常运行设备上^[9]，从而保证了数据的安全，并且不影响客户的正常操作。同时，也使得故障磁盘对客户的损失降到了最低；采用云存储系统，还享有专业的运维团队，负责维护存储系统的稳定和安全，随着供应商日益增多，使用费用将会不断降低。

劣势：基于 ceph 的云存储运维系统，在存储介质稳定性方面有很好的保证，但在信息安全（主要指被非法用户获取）方面存在明显不足。因为数据存储是通过网络进行存储，而非传统的本地磁盘，虽然通过加密可以降低信息泄露的风险，但在数据处理方面将变得比较复杂，开发难度大，同时使得数据处理能力下降^[10]；能耗大也是云存储面临的另一重大挑战，一方面，随着基础规模的扩大，能量消耗将会急剧增加，另一方面，随着数据量的增加，启动设备组件增加，所需能源越来越大，有统计，在 2010 年，其对电能的消耗为 2×10^{11} 千瓦时，约占全球总电量的 1.3%，且其比例还在不断增加^[11]，存储系统会在能源消耗中占很大比例，仅次于计算资源耗电量，约占 30%^[12]，因此，现在云存储的能耗也是云存储的一个重要的关注点；数据访问方面，有时候也会出现问题，这和数据存储设备、带宽有关，这个应该不成为日后的问题。

基于 ceph 的云存储运维系统，建立在 ceph 分布式系统的基础之上。随着 ceph 分布式系统的发展，ceph 云存储运维系统实现了对下面几个方面的监控。

(1) 系统状态：记录了云储存系统健康状态，包含系统异常时对应的异常信息。

(2) 监控器状态：记录了云存储系统中，所有的监控器信息及状态。

(3) IOPS 信息: 记录了一段时间内, 在云储存系统上, 对磁盘操作的频率和对应的时间关系。通过这个信息, 可以推断出用户频繁使用云储存系统的时段。

(4) 读写速度: 记录一段时间内, 云存储系统中, 流量带宽信息。通过这个信息, 可以了解到云存储系统中, 数据的流动情况。

(5) 磁盘信息: 记录了云存储系统中, 对应集群所拥有的总的磁盘大小, 以及已使用磁盘的容量。

(6) 归置组信息: 记录了云储存系统中, 对应集群所具有的归置组个数。

(7) 数据池信息: 记录了云储存系统中数据池的个数, 以及这些数据池中容量将要被占满的数据池信息。

(8) OSD 信息: 记录了云储存系统中, 某个集群中存储节点的信息。

1.2.2 国内外云存储的应用和趋势

在国内, 阿里的阿里云 OSS (Object Storage Service, 对象存储服务) 是中国最大电商提供的云存储服务。阿里云保证 OSS 服务的可用性不低于 99.9%, 数据自动多冗余备份, OSS 还提供资源隔离、异地容灾备份, 并提供企业级多层安全防护和防 DDoS 攻击。规模可以自动扩展, 采用多 BGP (Border Gateway Protocol, 边界网关协议) 骨干网络, 无带宽限制。

百度网盘, 已经上线的产品包括: (1) 网盘: 用于提供多元化的数据存储服务, 个人用户可以免费获取 2T 容量空间; (2) 个人主页: 通过个性化的设置, 关注功能, 可以实时获得好友分型动态, 实现文件共享; (3) 相册: 用户可以通过云相册, 实现照片的存储、浏览、分享和管理, 通过照片分享和记录生活的美好; (4) 人脸识别: 百度网盘可以实现按图分类、按图去重, 还能根据图片搜索图片; (5) 通讯录备份: 百度网盘为 iPhone 和 Android 手机用户, 提供手机通讯录备份功能; (6) 记事本: 百度网盘提供笔记功能, 用户可以在线编辑文档, 支持文字、图片和语音三种记事功能。

在国外, Amazon 提供了 S3 的对象云存储。S3 具有简单的 Web 入口。通过 Amazon 的云数据迁移功能, 可以轻松的将大量数据从外界迁入到 Amazon S3 或从 Amazon S3 迁出。S3 提供了全套的 REST API 和软件开发工具包, 可以方便轻松实现和其他技术的集成^[13]。

Google 云端硬盘, 为用户提供 15GB 的免费在线存储空间, 供存储照片、录音、视频、设计稿和绘图等众多文件的存储。通过手机、平板或电脑等终端, 可以随地随时的使用文件。用户还可以轻松邀请他人查看、协作或下载用户文件, 避免了邮件发送附件的麻烦^[14]。

发展趋势：(1) 云呼叫应用。云储存是在云计算概念上延伸和发展出来的，需和云计算结合才能发挥云存储的优势。云呼叫应用就是云储存未来发展的一个很好趋势。企业无需购买任何软、硬件资源，只需要具备场地、职工等基本条件，即可按照企业自己的需求，快速搭建数据自己的呼叫中心。对于日常系统、网络 and 平台等的维护，可由服务商提供。此应用对于企业建设来说，具有周期短、部署灵活、投资少、风险低、系统容量伸缩性强和运营维护成本低等众多优势。对于客户服务中心、电话营销中心等机构，只需要根据自己的需求租用服务，即可建立一套可靠、稳定、全面和座席遍布全国的呼叫系统。

(2) 云游戏应用。云存储结合云计算，所有的游戏都在云服务器段运行，并将渲染后的运行效果，压缩后通过网络发送给用户。用户就不用购买高端处理器和显卡，只需要本地计算机具备解压能即可。用户通过普通计算机就能获得极致运行画面，这对用户的吸引是不言而喻的。同时，对于主机厂商，也不用花巨额，投资在新型主机上，他们只需要用原先很小一部分投资用在升级自己服务器即可^[15]。

(3) 云教育应用。云储存的出现，对监控、视频之类的发展，起到了极大的促进作用。可以通过云储存和云计算相结合，将视频通过云储存的方式，应用在教育行业。在信息中心搭建信息采集工作站，在各个企业或学校等机构部署录播或直播系统，并配备流媒体功能组件。可以实现将录播视频发送到流媒体平台管理中心的全局直播服务器上，也可以将录播的内容上传至信息采集中心的服务器上。方便日后的点播、评估。实现将较好的教育资源，更好的应用到全国各地。

1.3 云存储系统设计的先进性和意义

基于 ceph 的云存储系统设计的思想，是将大文件分成易于存取、相同大小的块，较均匀的分布在各个存储节点上。充分利用每个存储节点的数据处理能力，实现高效、并发的对数据进行相应的处理。同时，通过内部实现机制，避免出现单点故障。每个客户节点，通过自身计算，可以算出自己所要读取或写入文件的对应位置。彻底解决了数据存储过程中，众多请求对单个或某几个服务器造成的压力。

通过 ceph 云储存系统的存储机制，可以很好的应对大数据的存取操作，给用户带来良好的体验。在稳定性方面，基于 ceph 的云存储，通常采用三副本备份机制。用户可以根据信息的重要程度，配置对应存储机制，执行相应的存取策略，实现较稳定、安全的存储数据^[16]。云存储的内部实现机制，对出现磁盘

损坏有很好的应对策略^[17]。在后期的扩容过程中，几乎不被用户感知，就可以实现相应存储空间升级^[18]。

对服务器的投资，是耗时、耗资又耗力，且不能及时转换成经理收益。所以云服务对于那些提供服务型的初创公司，前期就不必将大量资金投资到服务器上。公司即可集中精力和人力去开发公司的服务业务。前期，规模小，根据需要购买少量存储服务。后期业务变大时，可以通过云服务，轻松扩容，从而大大降低了公司投资成本^[19]。

在 21 世纪，数据是一个巨大的财富。通过对数据分析，可以直接影响到最终的决策。数据对人们的影响是非常广泛的。小到个人，大到一个企业、一个国家甚至到国际。对于个人，比如在网上购物时，商品的评价对后来购物者决定是否购买，起着重要作用。比如个人理财，理财者会通过对比各个产品，结合产品的利润、产品风险大小、产品期限和产品曾经的收益记录等各种信息，最终决定是否购买某种理财产品。这些信息，涉及的内容比较少，通过个人的对比，即可做出决策。大到一个企业、一个国家。比如一个销售企业，试图推销某件商品时，就会涉及到投资风险评估。这就需要收集大量数据，并对数据进行整理评估分析，决定是否可以投资某些商品。当数据使用者提升到企业、国家或者更大时，他们在做出决策前需要分析大量数据，数据量远远超出了人类自身的分析能力。

当出现大量数据，并需要对大量数据进行分析时，就可以凸显云储存的使用价值。根据需要，订购合理的服务。不必担心存储介质更新换代，不必担心存储介质的损坏，也不必担心后期数据扩容给对自己业务的影响。

1.4 本人主要工作

前期阶段，参与了界面信息显示设计，并负责通过 Axure RP 7.0 画图软件，将整体页面结构和逻辑模拟出来。在中期的实现阶段，开发云存储运维系统的监控模块，负责 Server SAN 界面的开发，以及对应前端和后端的交互模块。在后期阶段，负责本模块的功能维护，还做了云存储磁盘的性能测试。所开发的运维模块，成功应用在公司中标的苏州银行的 TDcube 项目中。

在运维项目开发过程中，遇到较为有意义的问题主要有两个：第一，数据请求延时；第二，浏览器间兼容性。

1、数据请求延时

a) 问题描述

在云存储运维系统模块中，部分界面在从后台取数据时，有时需要很长的时间。数据获取时间长短和后台维护的磁盘容量成正比。当磁盘容量很大时，数据获取的时间会比较长。每当用户点击界面时，就会发送相应的请求到后台，取相应的数据。如果要读取的信息还没有返回，用户就切换到其他界面，而在另外的界面，就又会发送对应界面相应的请求。在这个界面还没返回时，又做界面切换。在连续快速的做页面切换时，就会造成后面再去做页面跳转时，即使用正常的速度切换界面，在页面数据信息显示时，有时用时也会是正常情况下的好几倍。在操作较为复杂的界面时，还会出现页面卡顿，有时还会出现刷不出界面的情况。

b) 现象分析

首先需要说明一点，JavaScript 语言的引擎是单线程的，不支持多线程操作，支持异步操作，并强制所有的异步事件排队等待执行。对界面加载缓慢这个问题，经过断点跟踪，最终发现，当采用项目框架的方法，用\$resource 模块中的 `get(function (res))` 往后发送请求时，项目中对应的 js 文件就会采用异步的方式等待后台返回结果。在发送请求的 js 代码块等待返回结果的同时，js 文件会处理后面其他程序。在结果返回前，去做页面切换。发出请求数据的 js 文件将会继续停留在内存等待数据的返回。其实，在切换界面的那一刻，前面界面对应的 js 文件，对请求返回的数据，就已经没有实际意义了。如果快速在多个界面做页面切换，就会有很多无效的 js 文件继续停留在内存，等待发送的请求返回数据。当对应的数据返回后，有相应接收函数还要去做数据返回后的相应处理。当对多个界面进行快速界面切换时，就会出现在同一个时间段，有大量的前台 js 文件并发做相应处理。而这些处理中，只有一个是有益的 js 线程。如果其他 js 线程抢占了处理器的处理权，那么，其他所有等待处理的 js 线程只能等待当前的 js 线程处理完自己的任务。这时有用的 js 线程就只能等占用处理器的 js 线程释放了处理器的使用权后，再去和其他无用的 js 线程抢占处理器使用权。通过这个过程可知，为什么在快速连续切换接界面时，界面数据信息显示会变得特别慢。

当知道这些时，对卡顿情况的理解，也变得比较容易了。当有用的 js 文件获得处理器的使用权时，去处理数据的过程中，当遇到异步操作时，处理器的使用权会立刻被其他 js 线程抢占。所以当前 js 线程即使需要一个非常短暂的异步等待，这也会使得当前 js 线程失去处理器使用权。从而使得有用 js 线程，要和其他线程，去争夺释放了的处理器的使用权。这也是在界面上，为什么会出现卡顿的原因。

还有一个现象，就是当快速对页面进行切换时，会出现界面刷新不出来。就是不论等多久，界面的数据始终无法正常显示。显然这个现象不仅仅是卡顿这么简单。刚开始对这个现象确实感到困惑，同时，这个现象还不容易复现。对于这个现象，笔者困惑了好久。后来通过不断的追踪调试，发现在固定几个界面切换时，这个现象较容易复现。笔者就试着在这几个界面，不断通过断点调试，发现在做快速多界面切换时，在当前界面被加载时，有其他界面对应的 js 文件处理无用的数据，这时有可能会和当前界面对应的属性标签进行交互。如果恰巧无用的 js 线程在做数据处理时，调用了现在界面对应的标签，做数据处理，就有极大地可能出现数据需求格式异常，造成数据处理异常中断，界面就会停止加载，出现前面所说的刷新不出来的状态。至此，可以完全知道问题描述中出现问题的根源所在。

c) 解决方法

当了解造成这些问题的具体原因后，下一步操作即为如何解决这个问题。通过查看，尝试项目中所有向后台发送请求的模块，然而却没有一个可以完美解决上述现象的模块。笔者就希望能通过新的数据传输方法框架，去实现运维界面需求的数据调用。通过不断查阅资料，发现有 Promise 机制，可以很好地解决当前所遇到的问题。

Promise 有三种状态：pending（默认）、fulfilled（完成）和 rejected（失败）。其中 pending 可以向其他两种状态进行转变。解决上述问题的方案：通过调用 XMLHttpRequest 机制，向后台发送数据。之所以使用 XMLHttpRequest 机制，是因为这个机制可以很好的控制数据请求状态，在 XMLHttpRequest 模块中，可以通过调用 XMLHttpRequest 模块中的 send 函数实现对请求的发送，通过 XMLHttpRequest 中的 onabort 函数可以实现对请求的终止。而这个特点，可以完美的解决界面中出现的上述问题。通过 Promise 模块对这个获取数据请求的机制进行封装，实现 angularjs 框架中需要的异步特性。可以很好的实现数据获取的异步特性。

在 angularjs 框架下，当点击页面跳转时，js 线程会捕获到 \$destroy 事件。通过在 \$destroy 事件中添加对应的函数调用，实现在界面跳转时做相应的工作，以实现页面跳转所需要完成的事件。利用在 XMLHttpRequest 模块中，具有立刻终止数据请求的函数的特性，结合 Promise 实现机制，可以在界面发生跳转前，利用 \$destroy 函数，使得发送数据请求的代码块，可以在界面跳转前得到相应的回应。在没有进入下一个界面前，就可以处理完所有需要等到请求返回后的处理事件。这个处理过程不会出现 js 线程竞争处理器资源的现象，即不存在异步事件排队等待处理这个现象。在跳转到新的界面时，上个界面对应的 js

线程，处理结束后就退出内存的占用，不会影响到新界面对应的 js 线程的运行。从而从根本上解决了上面提到的关于界面反应迟钝、卡顿和刷不出界面的现象。后面会在附录中列出发送请求机制。在附录中会附加上这部分实现机制。

2、浏览器间兼容性

a) 问题描述

所有功能模块开发时，使用的都是 chrome 浏览器。在基本模块开发完毕之后，在做浏览器兼容性测试时，发现运行在 IE 浏览器下，页面显示异常。云存储运维管理系统上面的数据不能正常显示，而 DTCCube 项目中其他模块的数据可以正常显示。

b) 现象分析

通过断点、定位发现，点击、刷新或切换页面时，在后台 Java 代码部分，没有收到任何关于前端代码发过来的请求。由此可以断定，问题出现在前端向后台发送数据部分。通过单步跟踪，发现在自己编写的往后台发送请求的模块，在使用 IE 浏览器时出现了问题。这也是为什么别人开发的功能模块可以正常在 IE 浏览器下运行，而运维模块却出现了页面无法正常加载。对于数据无法正常显示的情况，通过搜索相关资料，发现在 IE 浏览器上不支持 Promise 异步机制。至此，找到了问题所在，下一步就是如何解决在 IE 浏览器上，关于 Promise 的异步特性的兼容性问题。

c) 解决方法

起初，将解决问题的重点放在了找到一种代替 Promise 异步机制的功能模块，实现相应的异步请求操作。后来发现很难找到某个调用机制，能很好做到解决页面加载慢、卡顿和刷新不出来界面的问题。在思索中，受到 echarts 插件的启发，心想按照 Promise 实现机制，手动写个模块，用于支持相应的异步操作。在搜寻关于 Promise 实现机制的时候，发现有单独的 Promise 模块 promise-0.1.1.min.js 可供下载使用。笔者就下载了对应的 Promise 模块，并加载到了工程项目。在调用 Promise 模块时，不在依赖浏览器是否支持 Promise 模块，从而使得项目在不同浏览器上，在做向后台发送数据时，能更好的运行。

以上两个问题是笔者在做运维系统开发时，处理的较具有突破性的问题。通过笔者这种创新机制，可以彻底地解决延迟返回这一类问题。

1.5 论文组织

第 1 章是绪论，通过对存储的需求以及数据的急剧增加而引出此次选题的意义，然后是对所涉及到的技术发展状况和趋势的概述。第 2 章，通过对现实

需求的分析，得出云储存应用的场景和云储存所涉及到的关键技术，从而引出 **ceph** 的概念和机制。第 3 章，对系统的概念和内部架构进行了描述，根据用户需求引出对应的功能和性能需求。第 4 章讲述了系统整体设计以及云存储底层的设计的实现细节。第 5 章讲述了在实际开发的过程中，环境的搭建、部署以及系统的开发管理。第 6 章讲述了项目中，开发的功能模块以及开发过程中所进行的功能测试。第 7 章为总结，对所做的项目进行了全面的叙述，对文本进行了总结，本章还包含实习过程的心得。

第2章 理论基础

本章通过技术的更迭，引出云储存出现的场景，然后是对基于 ceph 的云存储技术和机制的概述，从而更加深入的了解基于 ceph 的云储存机制的实现原理。

2.1 云存储

2.1.1 云存储出现场景

随着互联网的发展、以及云计算和物联网的出现，人类产生的数据信息在以极快的速度产生。具体有多快，可通过一个名人提出的定律感受下。图灵获奖者 Jim Gray 曾提出的数据增长经验定律——网络环境下每 18 个月产生的数据量等于有史以来数据量之和。通过他提出的定律，可以更好的感受下，当前社会中数据增长的速度。随之而来的问题就是如何能安全、有效的存储这些数据。同时，保证在人们需要这些数据时，又能安全、高效的呈现给用户。

在过去的三十年中，RAID（Redundant Arrays of Independent Disks，独立冗余磁盘阵列）技术为数据的存储做出重大贡献，用于存储各种类型的数据^[20]。每种技术的出现都会有一定的局限性。当新技术出现时，不可能将将来所有可能出现的技术、需求都预测到。比如 RAID 技术，在刚兴起的时候，可以很好的满足当时需求。因为当时数据量还不算太大，在数据的存取、容灾和备份上有很好的性能。但当数据变得庞大时，RAID 的种种弊端就显得尤为突出。很多情况下，RAID 已经不能满足用户对存取业务的需求。当前，RAID 面临的问题如下^[21]：

- 1、RAID 重建困难。随着数据不断增长，单个磁盘容量也在不断扩大。很多企业单磁盘容量都是 4TB、6TB 或者更大的 10TB。以这么大的单磁盘构成磁盘阵列时，当某个磁盘出现损坏时，RAID 将会需要几小时或者几天来进行数据恢复。在恢复的过程中，如果出现第二块坏盘，那么数据恢复将会变得异常复杂。在数据恢复的过程，对业务处理产生较严重的影响。

- 2、RAID 备用磁盘增加成本。在 RAID 中会有备份磁盘，刚开始组建 RAID 时，磁盘出现故障的频率低。当系统没有故障时，备份磁盘将会被闲置。造成资源浪费，增加成本。经过一段时间，磁盘出现故障率会大大增加。当出现多个磁盘故障时，备份磁盘在恢复数据时会出现被占满的风险。这种情况，将会

面临严重的恢复问题。

3、RAID 高度依赖硬件。在 RAID 组内，对磁盘要求比较严格，需要各个磁盘在容量、转速或磁盘类型上要保持一致，不然将会严重影响存储性能。同时，随着 RAID 中磁盘阵列的不断扩大，需要更多的昂贵的 RAID 控制器，以免出现单点故障，增加了投资成本。

4、RAID 扩容有限。RAID 的扩展能力有限，当扩展到一定程度时，再继续扩容，会增大存储控制器的压力，从而会降低 RAID 性能。从而表明，RAID 的扩展能力有限，对于现在的大数据存储需求显得力不从心。

基于这些弊端，需要寻找其他存储方式，来适应当前大数据的存储。基于 ceph 的云存储系统，通过巧妙的设计，完美的解决了上述 RAID 现在的问题^[22]。在基于 ceph 的云存储系统中，当磁盘出现损坏时，会很快被系统捕获到。通过内部恢复机制，可以从故障磁盘对应数据的对等磁盘上的数据，进行恢复。因 ceph 是分布式系统，所以其要恢复的数据在系统中的各个磁盘上。所有具有损坏磁盘对应副本的磁盘，都在帮助故障磁盘恢复。在恢复的过程完全自动，且在恢复的过程中，几乎不影响任何业务处理。同时，也不需要任何备用磁盘做数据恢复的备份盘，ceph 内部恢复机制，会将被恢复的数据均匀的写到其他各个磁盘上^[23]。基于 ceph 的云存储系统中，根据磁盘的容量，每个大小的磁盘都对应一个默认权值。根据磁盘的性能和质量，可以手动修改这些权值。所以在系统中，不要求存储磁盘大小一致，从而降低了对硬件的依赖。基于 ceph 的云存储，每个客户端，只需在前期，从监控节点获取集群 map 信息，在后期可以实现独立和存储节点交互。这种设计特别灵活，且具有超强的扩展性^[24]。可以很好的满足当前和不久的将来，对存储的需求。

2.1.2 云存储的关键技术

在传统的存储系统中，当客户端需要访问数据时，需要和中间的一个模块（控制器）打交道。这个模块就是平时所说的，能引起单点故障的一个模块。同时，这个模块使得传统存储系统，在扩大到一定规模时，再继续扩容会受到限制^[25]。基于 ceph 的云存储技术，之所以能够兴起并被广泛应用，和其实现的关键技术有着重要关系。其所涉及到的关键技术如下：

（1）CRUSH 机制：基于 ceph 的云存储系统中，使用 CRUSH（Controlled Replication Under Scalable Hashing，可扩展散列的复制控制）算法，实现对数据的存取和访问。通过 CRUSH 算法，在客户端可以计算出要存储的数据位置或要读取数据的位置^[26]。当客户端需要存储或读取数据时，需要从 ceph monitor 这里获取 cluster map 信息。Cluster map 是由 CRUSH map、monitor map、MDS

(Metadata Server, 元数据服务器) map、PG (Placement Group, 归置组) map 和 OSD (Object-based Storage Device, 对象存储设备) map 组成。客户端通过 CRUSH map, 可以了解到整个系统的状态和配置, 其中包括 bucket 列表、集群设备列表、故障域分层结构和故障域规则等信息。Monitor map 用来监控集群中端点到端点的状态, 其中包括 monitor 节点名称、ceph 集群 ID、IP 地址和端口号等信息。MDS map 包含 MDS map 的创建时间、修改时间、MDS map 的版本号、数据和元数据存储池的 ID、MDS 数量和 MDS 状态等信息。PG map 包含 PG 对应的 ID、状态、up OSD sets、时间戳、PG 版本号和使用百分比等信息。OSD map 包含了集群 ID、版本号以及存储池的相关信息。通过这些信息, 每个客户端可以直接和对应的 OSD 节点直接通信了。从而避免了传统存储系统的单点故障这个缺陷, 同时也充分发挥了各个客户端的计算能力, 实现了高并发操作。

(2) monitor 机制: 基于 ceph 的云储存系统中的 monitor, 用于实时更新 cluster map 的信息, 以便客户端给他通信时, 能获得最新的 cluster map 信息。由此可知, 要使整个系统正常运行起来, 需要客户端和 monitor 进行通信, 就有可能造成单点故障的风险。在实际应用中, 每个客户端, 在刚开始加入进来时, 需要和 monitor 交互, 后面可以独立完成对数据存取。从而可以避免单点故障情况的发生。ceph 集群会有一个 monitor 集群, 且集群中 monitor 的个数为奇数个。因为 ceph 机制施行仲裁模式, 需保证有一半以上的 monitor 正常运行, 以免发生脑裂。其中一个 monitor 作为主 monitor, 负责和客户端进行交流。当这个 monitor 挂掉后, monitor 内部机制又会选取其他一个 monitor 作为主 monitor。每个客户端, 在刚开始时, 需要和 monitor 进行通信。后期在进行数据的存储和读取的时候, 可以独立完成任务。当出现读取数据和自己信息版本不一致时, 会去 monitor 中获取最新版本信息。这也使得基于 ceph 的云存储系统, 具有很强的扩展性。

(3) 身份认证: 基于 ceph 的云储存系统, 采用了 cephx 身份认证系统来实现用户身份认证。由集群的管理员, 即 client.admin, 为用户创建账户。根据用户名会生成对应的密钥, 并将密钥保存在 ceph monitor 上。同时将创建的用户信息和密钥返回给集群管理员。再有集群管理员负责将密钥发送给存储管理员, 存储管理员将密钥和用户 ID 发送给用户。Ceph monitor 除了将保存密钥并返回密钥外, 还会将密钥发给 ceph OSD、ceph MDS 共享。具体流程如图 2-1 所示。

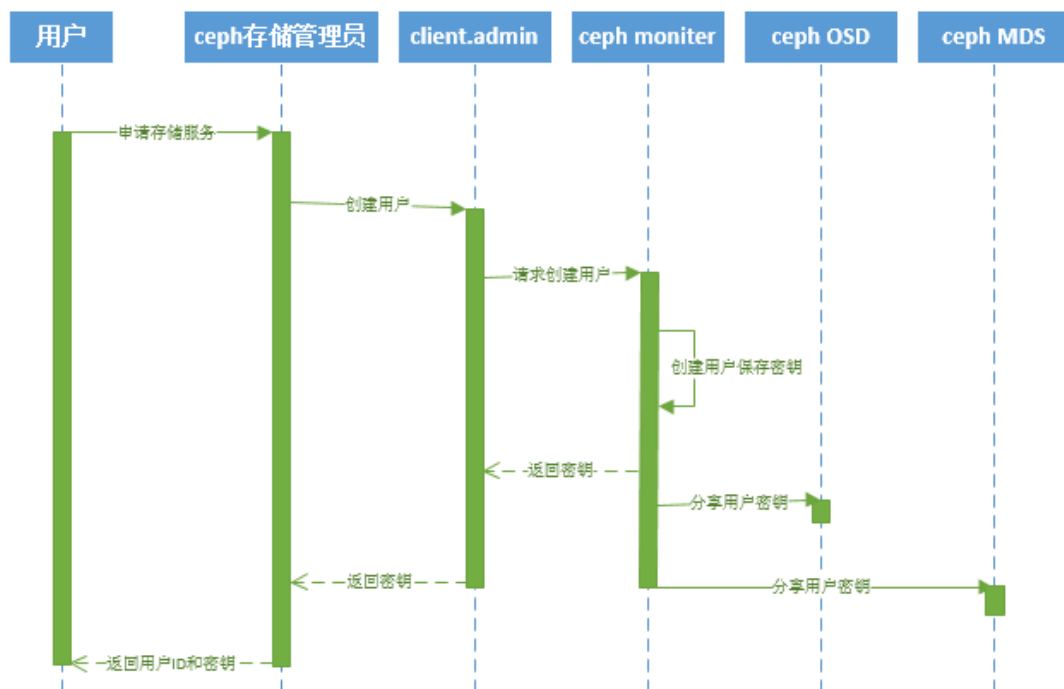


图 2-1 cephx 身份授权

2.2 ceph 机制

2.2.1 集群框架

在 ceph 的框架有以下几个部分组成：

(1) ceph monitor: ceph monitor 简称 MON。ceph monitor 负责监控、管理整个集群的健康状态。此部分包含：CRUSH map、MON map、OSD map、MON map 和 PG map。整个系统中，所有节点包括客户端、OSD 节点等，他们都会从 ceph monitor 获取集群状态信息，并将自身探测到的异常信息汇报给 ceph monitor。

(2) ceph 存储设备: ceph 存储设备简称 OSD 节点。OSD 节点是整个系统中真正存储数据的单元。当系统发出写操作时，客户端通过自身计算，就会将写对象，写入到对应 OSD 节点中。读操作，也是从对应的 OSD 节点的特定的位置，读取数据。一个 OSD 节点拥有一个守护进程。在实际的环境中，一个守护进程对应一个物理磁盘。

(3) ceph 元数据服务器: ceph 元数据服务器简称 MDS。MDS 主要为 cephFS 文件系统服务。为其提供文件层次结构，提供元数据服务。对于块存储，这部分是可有可无的。

(4) RADOS: RADOS 是 ceph 的存储基础。RADOS 负责将所有形式的数
据都以对象的形式保存。RADOS 能保证数据的一致性,其内部具有数据一致
性检测、故障检测、数据迁移和数据恢复等功能,还具有平衡磁盘数据的功能,
提供丰富、大量和全面的操作命令,供上层 librados 调用。

(5) librados: 通过 librados 库,可以实现对 RADOS 的访问。Librados 库
支持 Java、Ruby、PHP、Python、C++和 C 等众多语言。在 librados 的基础上,
又开发了 RBD(RADOS Block Devices, RADOS 块设备)、RADOS GW(RADOS
Gateway, RADOS 网关)和 cephFS(RADOS File System, RADOS 文件系统)
组件。通过这些组件,可以更高效、更具有针对性的开发。

(6) RADOS 块设备: RADOS 块设备简称 RBD,即 ceph 块设备。可以实
现对数据的分割、存储,使数据分散在多个 OSD 节点上。通过调用 librados 层,
实现和 RADOS 层的通信。RBD 实现了对 librados 的进一步封装,方便基于块
存储开发者的调用。

(7) RADOS 网关接口: RADOS 网关接口简称 RADOS GW。RADOS GW
通过 librgw(RADOS Gateway Library)和 librados 可以实现和 ceph 的通信。RGW
提供的 RESTful API 与 Amazon S3、Openstack swift 兼容。RADOS GW 是对
librados 的进一步封装。

(8) cephFS (ceph Filesystem, ceph 文件系统): 和 RBD、RADOS GW 一
样,cephFS 也是对 librados 的封装,用于存储用户数据的文件系统。

集群框架图如图 2-2 所示。

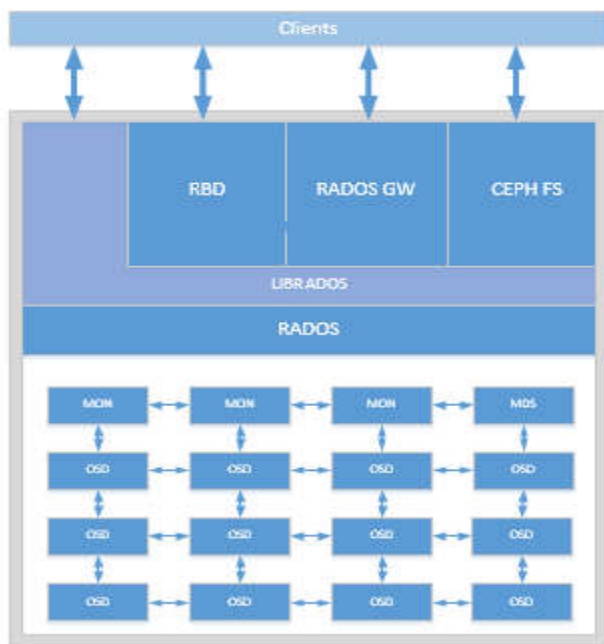


图 2-2 ceph 集群框架

2.2.2 存储机制

基于 ceph 的云存储有其独特的存取方式，正是这种存取方式，使得基于 ceph 的存储系统具有高并发的特性。同时，这种存储方式也消除单点故障瓶颈^[27]。正是因为 ceph 的独特存储机制，使得 ceph 能在众多存储系统中，脱颖而出，并占据很大的市场份额。在今后的分布式存储中，ceph 存储的应用将会越来越广泛。

图 2-3 是 ceph 分布式系统存储的宏观图。通过 ceph 分布式宏观图解，可以很清楚的看到，基于 ceph 的存储系统，是如何以分块、映射和存储的。

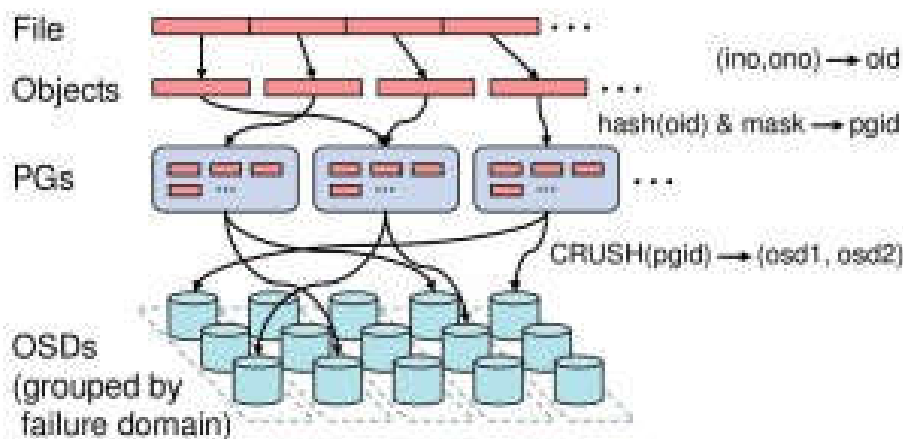


图 2-3 ceph 存储机制

通过图 2.3 可以看到，数据的存取操作的实现主要有以下几个部分。

(1) **File \rightarrow Object**: 实现方式是给每个文件一个唯一 id，然后再将源文件拆分成事先规定好的、易于在网络中传输的和大小相等的数据块。每个数据块都有自己的编号，用文件 id+块编号，可以在整个磁盘系统中为数据块指定唯一的一个编号，从而将较大文件拆分成多个易于存储的 Objects，方便后续统一管理^[28]。

(2) **Objects \rightarrow PGs**: 将文件分割成诸多 objects 后，需要将这些文件映射到对应的 PGs 上去。映射规则比较简单，通过公式： $hash(oid) \& mask \rightarrow pgid$ 实现对应的映射。根据设计，pg 个数为 2 的整数次幂，mask 为 pg 总个数减一。

(3) **PGs \rightarrow OSDs**: 这个过程是将 object 的逻辑组织单元的 PG 映射到实际存储单元 OSD 上。采用 CRUSH 算法，将 pgid 带入其中，根据 ceph 对应的配置信息，可以得到有 n 个 OSD 组。其中 n 是为了实现可靠性而设置的一个值，代表副本数。在生产环境下通常为 3。

到此，基于 ceph 分布式系统下的低层，完成了从 File \rightarrow Object \rightarrow PGs

—> OSD 整个过程的映射。整个存储过程中，没有一个地方会涉及到查询全局性的表来获取存储数据的位置。存取数据到磁盘的最终位置的计算任务，由客户端完成，而不是由公用的服务器去计算。从而实现高并发特性^[29]。这也是 ceph 之所以被很多企业所接受的最重要的一个原因。

在存储系统中，对数据的存储最终要落到存储介质上。基于 ceph 的云储存系统中，各种特性最终的目的，就是让数据能够更好地落到磁盘，即 OSD 上^[30]。基于 ceph 的云储存系统，在 OSD 设计上也很有特色。在基于 ceph 的云储存系统中，OSD 包含以下状态。

Up 且 in: 该 OSD 处于运行正常状态，且至少承载了一个 PG。

Up 且 out: 该 OSD 处于运行正常状态，但并未承载任何 PG。

Down 且 in: 该 OSD 处于发生异常状态，但仍然承载至少一个 PG。

Down 且 out: 该 OSD 处于发生异常状态，且已经不再承载任何 PG。

其中 cluster map 可以收集到这些信息，并以增量形式进行扩散，其中版本号是递增的形式。当出现版本不一致时，以较新的为准。在基于 ceph 的云储存系统中，cluster map 的信息是以异步且 lazy 的形式扩散^[31]。当 OSD 节点出现故障且最终不能使用，或当系统中添加新的 OSD 节点时，这些行为都会主动或被动的被 ceph monitor 捕获到。ceph monitor 并不会立刻将最新的 cluster map 广播出去，而是等 OSD 向 ceph monitor 汇报情况时，将最新 cluster map 信息传给这个 OSD。当这个 OSD 和其他较低版本节点通信时，又会将自己的 cluster map 信息传给较低版本的节点。通过这种方式可以有效的避免了广播风暴，最终会将 cluster map 扩散到全体 OSD 和 client 端。根据 Sage 论文可知，每次版本更新，会在 $\log(n)$ 的时间复杂度内，传播到所有的 OSD 节点上。各个 OSD 会根据 cluster map 进行数据维护，而 client 在进行数据存取操作时，只需要 cluster map 的信息，即可实现自助寻址^[32]。这充分利用了各个节点的计算资源，使得基于 ceph 的云储存系统，具有高效的存取性能。

通过上述存储机制可知，一个大的文件会被分割成大小相同的、相对较小的数据单元，并被分别存储在不同的 OSD 存储节点上^[33]。同时，每个较小的数据单元都会被分配到不同的 OSD 存储节点上。同一个文件的不同 OSD 存储节点间会有通信。当 ceph 云存储系统中有磁盘出现故障，那么和故障磁盘上具有相同文件的其他 OSD 存储节点，都会检测到这个故障磁盘，当其他 OSD 存储节点检测到这个故障磁盘，就会将这个故障磁盘的信息报告给 ceph monitor。Ceph monitor 就会及时更新数据信息。与此同时，当确定故障磁盘已经无法继续使用时，ceph 云储存将会启动恢复机制，将故障磁盘所对应的信息，从副本中拷贝一份，放置在其他正常磁盘上面^[34]。从而可以做到及时发现故障磁盘，

并将数据从备份中恢复出来，保证信息的完整性。这些过程不需要人去干预而自动完成。

2.3 总结

通过前两节的讲述，可以对基于 ceph 的云存储系统的一些概念有了大致的了解。过上面对 ceph 存储机制的讲解，可知，基于 ceph 的云储存，在数据存储方面，确实有着不可比拟的优势。ceph 云存储系统，不仅实现了高可靠、高性能和高度自动化，更重要的是其解决了单点故障瓶颈问题。基于 ceph 的云存储系统，可同时支持数千个存储节点，实现了大数据对磁盘空间的需求^[35]。

第3章 需求分析

本章主要介绍了基于 ceph 的云存储运维系统中所涉及到的功能需求模块、业务处理流程和系统性能需求。

3.1 系统概述

基于 ceph 的云存储系统，可以适应现在大数据对存储的需求，且应用范围越来越广。当云存储作为一种存储服务，对外销售时，就需要具备图形化界面。基于命令行的操作，只适合开发人员、测试人员和运维人员。对于购买存储服务的用户，就需要拥有图形化界面。基于 ceph 的云存储运维系统，是为了满足客户对后台存储信息的监控而产生的。ceph 云存储运维系统，是按照客户的需求，对用户最关心的功能模块进行了图形化的封装。

基于 ceph 的云存储系统是 Server SAN 的一种应用实现^[36]。Server SAN 可以理解成，由多个带存储的服务器组成的存储网络，其内部的存储管理是由软件实现。基于这个定义，和 ceph 具有相同性质的系统，还包括 Nutanix、Nexenta、VSAN 和 Gluster 等。

通过运维系统，用户可以看到各个存储集群（cluster）。每个 cluster 又包含很多内容。在每个 cluster 中，含有以下信息。

cluster 健康状态：这个信息展现 cluster 的最近一次更新的健康时间点，在 cluster 健康状态栏中，还包含了当前 cluster 健康状态。其中 cluster 的状态包含健康、警告和失败三种状态。当 cluster 的健康状态出现警告或失败时，有对应的产生警告或失败的原因信息。

监控状态：记录对应 cluster 上含有的所有监控节点，以及监控节点所处的状态。监控节点只含有两种状态，正常和异常。

IOPS：通过折线图，可以看到最近一段时间内 cluster 的读写次数的变化情况。单位为次每秒。

读写速率：通过折线图，可以看到最近一段时间内 cluster 的读写次数的变化情况。单位为 KB 每秒。

磁盘总容量：记录对应 cluster 总的磁盘大小以及使用情况。

归置组：记录了归置组的个数和归置组的状态。其中状态包括三种状态。第一，正常状态，代表这个归置组下，拥有设定的 OSD 个数，且这些 OSD 处于正常运行状态。第二，降级状态，当对应某个归置组下运行的 OSD 个数，没

有达到预先规定的 OSD 个数，或者某个归置组下对应的 OSD 出现了故障，没能正常运行起来，此时的归置组处于降级状态。第三，当一个归置组下对应的 OSD 全部出现故障或此归置组下没有指定 OSD，此时的归置组处于异常状态。

POOL: 每个 POOL 的详细列表里面，详细介绍每个 POOL 所对应的 ID，每个 POOL 所拥有的副本数，每个 POOL 所对应的归置组个数，每个 POOL 最大容量以及已使用容量，每个 POOL 还记录了其在一段时间内，对应的 IOPS 数据。通过用户界面，用户还可以设置每个 POOL 对应的归置组个数和对应 POOL 最大的空间容量。

OSD 存储节点: OSD 存储节点也就是数据存储的单元。每个机架上会有很多台存储服务器，每台服务器上会拥有很多个 OSD 存储节点，每个 OSD 存储节点记录了本节点的 ID 编号，本节点所属的主机，本节点的状态，本节点的容量信息，本节点已用空间信息，本节点所对应的 IP 地址及对应的端口号和编号，通过运维界面，还可以控制 OSD 存储节点的状态。

OSD 存储节点是整个云储存的核心，OSD 存储节点的存储状态，直接影响到整个存储系统的性能。其中，云存储系统的负载均衡，其实就是指在各个 OSD 存储节点上的数据是否处于良好的平衡存储状态。在 OSD 界面，会有整个 cluster 的磁盘使用偏差值信息，可以依据这个信息，决定是否进行手动均衡。系统中还提供了自动均衡的功能，可以给系统设定每周的哪几天，在哪个时间段，设定最大偏差值为某个具体值，让系统自动做均衡操作。可以做到不影响用户正常操作的情况下，调整 OSD 存储中的数据，从而实现数据在 OSD 中负载均衡。

以上为云存储运维系统所主要包含的功能模块，后期随着客户需求的不断扩大，还会添加新的功能模块。其中，在项目前期设计阶段，云存储运维系统并没有手动、自动均衡模块，这个模块是后期，根据用户需求新增加的模块。

3.2 系统总体架构

3.2.1 业务流程

基于 ceph 的云存储运维系统，最底层是存储介质设备。所有的读写操作，最终都是对这层存储介质进行操作的。在存储介质上，安装了 Centos 后台服务系统，用来操纵最终的存储介质。直接和 Centos 系统交互的，是部署在 Centos 系统上的 ceph 分布式存储系统的 RADOS 层。基于 ceph 的云存储运维系统，所有的命令，最终都会转化成 RADOS 层对 Centos 系统的调用，并将结果返回到运维界面。

在 RADOS 层，具有丰富的命令。有些命令是开发人员使用的，需要对系统有很深的了解才能正确使用。被误使用后可能导致整个 ceph 系统出现严重问题。在 RADOS 层之上，提供了 LIBRADOS 层，实现了对部分命令的隐藏和限制，对 ceph 存储系统 RADOS 层的抽象和封装，并向上提供 API。

LIBRADOS 层提供了直接开发接口，供高级开发人员基于 LIBRADOS 层，进行开发。和 LIBRADOS 直接交互的是 RADOS GW、RBD 和 CEPH FS。他们是更高层次的抽象和封装，根据实际需求而选择不同封装层次和类别。基于 ceph 的云存储运维系统采用了 RBD 模块，用来和上层的 Calamari 交互。

Calamari 层主要负责处理客户端下发的命令，提取相应参数，并调用 RBD 的 API 接口，实现上层对应的操作。与 Calamari REST API 交互的是后端的 Java 层代码。

Java 层的代码，主要实现 Calamari 层和 JavaScript 层的对接。在中间对 JavaScript 层的命令进行过滤、封装并执行调用 Calamari REST API 动作，并将结果返回给 JavaScript 层。

JavaScript 部分，采用了 Angular JS 框架，能很好的实现了用户间的各种需求和数据的动态绑定。Angular JS 提供对前端用户界面的控制，实现了双向数据绑定，方便用户和系统间的交互，及时将后台数据更新到前台，为前台界面提供动态显示信息。业务的整体流程图如图 3-1 所示。

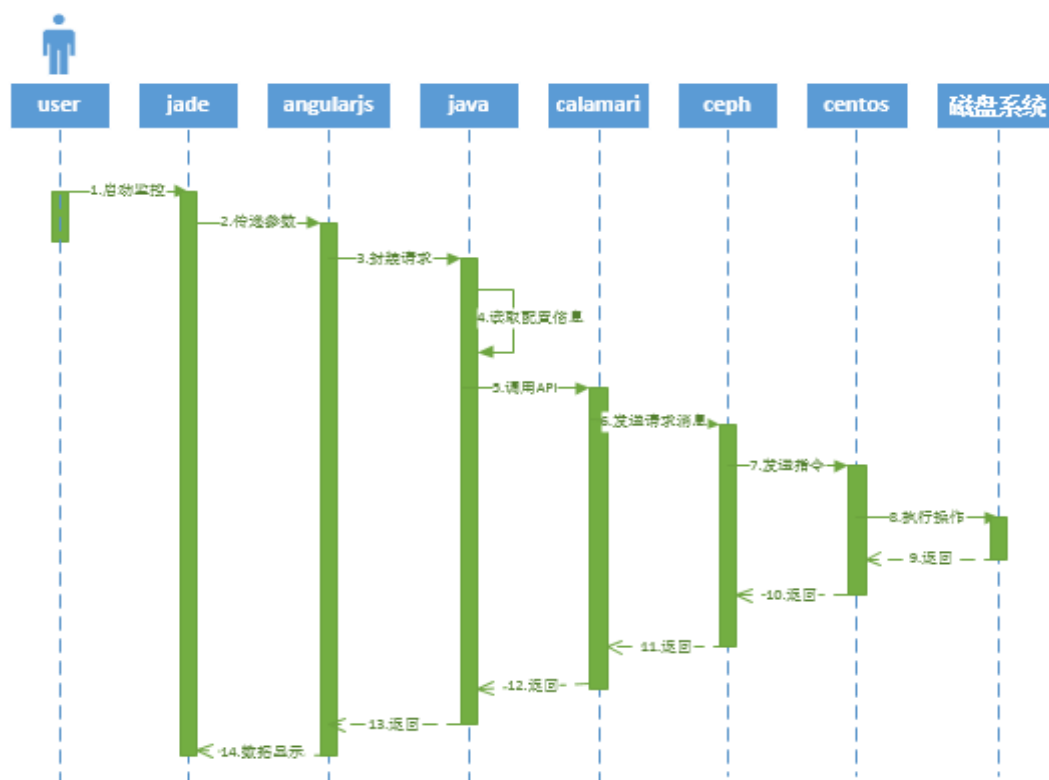


图 3-1 业务请求的总体流程

3.2.2 系统构造

基于 ceph 的云储存运维系统中，可大致分为以下几个结构。

1、数据库：用于存储数据，是 ceph 云存储中数据最终存放的地方。对应前面所讲述 OSD 节点，统一由 Centos 服务系统管理。

2、entos 系统：用于直接操纵磁盘、管理数据，为上层的 ceph 分布式系统提供服务。

3、ceph 系统：在 Centos 系统基础上部署 ceph 系统，实现软件定义存储，供用户租用。

4、Calamari 系统：通过部署 Calamari 系统，可以实现对 ceph 的 RBD 的进一步封装，为 java 模块调用提供便利。

5、springMVC：springMVC 模块，实现了用户界面和 Calamari REST API 的交互。

6、用户界面：用于监控各个集群运行状态。

其中，springMVC 和用户界面，由笔者负责开发实现。

图 3-2 为云存储运维系统模块架构图。



图 3-2 运维系统模块架构

3.3 云存储运维系统功能需求分析

3.3.1 账户管理

期初，运维系统部署完成时，只允许 super 用户登录。super 用户主要执行以下职能。

1. 修改密码：修改原始 super 用户密码。
 2. 角色管理：定义系统都是具备什么角色。例如，系统管理员、资源管理员或资源使用人等。
 3. 权限管理：定义各个角色具备什么样的权限。
 4. 用户管理：根据用户需要创建用户，并为每个用户分配对应角色。
- super 用户用例图如图 3-3 所示。

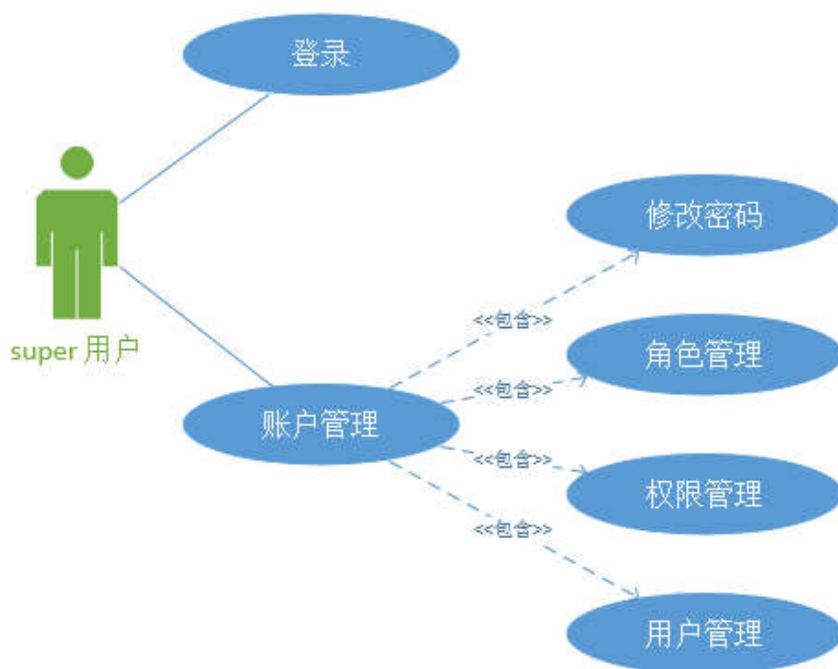


图 3-3 super 用户用例图

3.3.2 功能需求

系统要保证能时刻去检测、更新对应集群的信息，并将这些信息及时返回到用户界面。其中开发的功能模块如下。

- 1、集群列表：展示集群中所有集群及磁盘存储信息。

- 2、集群健康状况：反应当前集群健康状态，包含健康、警告和故障。
- 3、监控器状态：显示 MON 的运行状态。
- 4、IOPS 性能：用于显示整个集群中，某个时间段进行的读写频繁程度。
- 5、bandwidth：用于显示整个集群中，某个时间段内带宽使用情况。
- 6、磁盘信息：用于显示集群总的磁盘大小以及磁盘使用情况。
- 7、归置组（PG）：用于显示集群中归置组的个数和状态。
- 8、存储池（POOL）：用于显示集群中存储池的个数和状态。
- 9、OSD：用于显示 OSD 个数和状态。

基于 ceph 的云储存运维系统的用例图如图 3-4 所示。

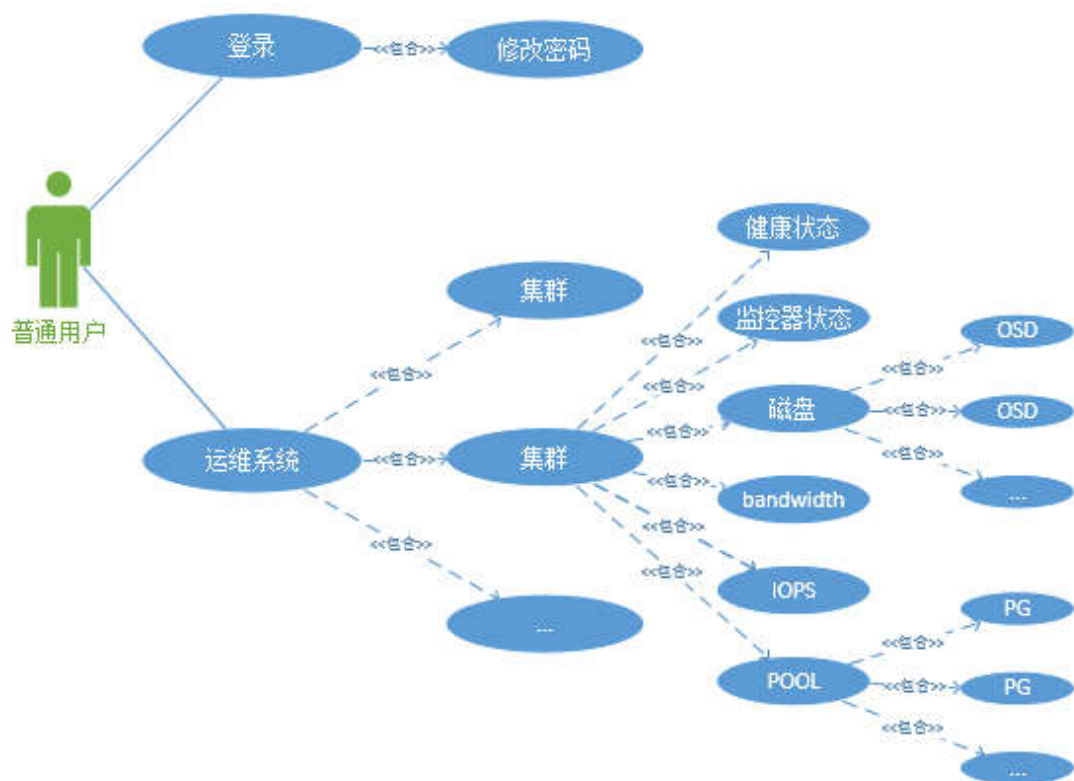


图 3-4 运维系统结构图

以上主要是讲述云存储系统的监控功能需求，其中还有一个较重要的需求，即数据的均衡操作。当磁盘数据出现分布极端化时，会严重影响到用户对数据的存取操作性能，使得系统的整体运行性能有很大下降，所以系统提供了均衡机制。因均衡机制使得数据会出现大规模迁移，占用系统资源较大，均衡期间会严重影响到用户的操作效率，所以在系统中提供了两种均衡机制：自动均衡和手动均衡。当磁盘使用偏差值超过正常范围时，通过自动均衡机制，可以按用户设定的日期和时间点去做均衡操作。从而可以做到最大限度的减小均衡时，

对用户业务的影响。手动均衡操作，是用户参照当前数据磁盘存储偏差，手动执行均衡操作。这个操作会使得系统立刻做均衡操作。

3.4 云存储运维系统性能需求分析

随着云计算和物联网的兴起，数字数据信息急剧增加。公司、企业和政府等部门将会产生极大的数据信息，他们需要有很大的存储介质来保存这些数据，并能很好的处理这些数据，提供良好的读取性能。他们对不同信息有不同要求。有些数据就是为了将数据保存起来，后期很少去使用，如监控录像等信息。这些信息只有在出现特殊情况时，才显得异常重要。这些数据在刚开始的时候，可以选择用较好的设备存储，当过了一定时间，被访问的可能性非常小时，可以将这些数据存储到廉价，处理能力较弱的存储介质上。有些数据，在存入数据库后，就可能会出现被经常查询，电信公司中，每个客户话费使用情况。这些信息，就应该存储在处理能力较强的存储设备上。针对这些不同情况，客户会有不同的性能需求。针对客户使用情况，常见的需求如下。

3.4.1 高性能需求

ceph 存储机制中，消除单点故障机制，实现了所有数据的存储和读取，不再必须通过查找某些固定的信息表就可以实现。即实现了 `client` 和 `server` 直接通信时，在其通信过程中摆脱了中间代理和相应的信息转发。

通过上述的 `ceph` 存储机制可知，一个原文件在存储前是被分割成很多大小相同的 `object`，且几乎所有的 `object` 都被分配到不同的 `OSD` 存储节点上，从而可以实现对某个文件操作时，使得所有含有本文件的 `OSD` 存储节点同时工作，实现了存储服务器的高并发的特性。同样，在读取一个文件的时候，根据文件对应的信息，所有具备这个文件 `object` 的主 `OSD` 存储节点，都会工作起来，供用户读取信息。存储数据的每个 `OSD` 存储节点，都有对应的权重值，参照权重值，充分发挥每个存储节点的存储潜能。。

保证数据不会因一个地方的磁盘损坏，而造成数据丢失的最直接方式，就是异地创建副本。创建副本需要考虑一个问题：创建副本过程中，对宽带的影响。在基于 `ceph` 的云存储系统中，`client` 只需要和标记为 `primary` 的磁盘进行数据传输就即可。再由标记为 `primary` 的 `OSD` 存储节点，通过内部存储网，创建预先规定的对应副本，从而可以避免有 `client` 去创建副本而造成的网络消耗，提高了存储性能，同时保证了数据信息的安全性。

3.4.2 高可用性需求

在 ceph 存储系统中，可以通过配置 `per-pool` 来设定每个文件分割成 `object` 对应的副本数。同时还可以设定故障域，保证数据的各个副本不在同一个地方，使得数据安全系数更高。

因基于 ceph 的存储系统可以实现 `client` 和 `server` 端的直接通信，消除单点故障隐患，所以允许系统出现很多种故障情形。这种机制还可以支持单个组件滚动升级和在线替换。通过 ceph 存储机制，可知，同一个文件被分割后存储在不同的 OSD 存储节点上，这些 OSD 存储节点间会相互通信。

当某个 OSD 存储节点出现了故障，与其通信的相邻 OSD 存储节点就会在一个通信周期内，感知到故障 OSD 存储节点，并将这种情况汇报给监控节点，使得其他 OSD 存储节点在和监控节点交互时，能及时得知系统中 OSD 存储节点状态。同时会激发 ceph 存储系统自动恢复机制，并将最终恢复后的状态反馈给监控器，再由监控器将最新状态信息更新到与其相通信的其他 OSD 存储节点上。在更新故障磁盘或新增新磁盘的过程中，客户端仍然可以正常使用数据磁盘，此时可能会出现三种情况。

1. 客户端是和已经更新过的 OSD 节点进行通信。这时因客户端 `cluster map` 信息和 OSD 节点信息不一致，OSD 会将他的最新信息更新到客户节点，客户节点可以根据最新信息去正确的地点获取数据。

2. 当访问的 OSD 节点和备份 OSD 节点没有完全更新完，此时 OSD 节点和备份节点发现版本不一致，会先更新版本，然后将最新的版本更新到客户端。客户端根据最新版本 `cluster map`，去对应地方读取信息。从而实现，当访问的 OSD 节点正在更新时，也能获取正确的数据。

3. 集群更新信息还没传到要访问的 OSD 节点，此时的 OSD 节点和客户端的 `cluster map` 信息一致，对应的存储数据也会因为更新信息，没有更新到这里，这里的数据不会有更新，所以客户端读取的信息还是正确的。

通过这三种情况可知，在整个恢复的过程中，系统中磁盘出现故障或磁盘进行扩容的过程不需要人工干预。在 OSD 存储节点恢复过程中，仍然可以通过访问故障 OSD 存储节点上，故障数据对应的备份 OSD 存储节点上的数据，实现数据正常的访问。当数据出现意外，需要恢复时，因数据是被分割成很多大小相同的 `object` 块，存储在不同的 OSD 存储节点上，当要恢复数据时，多个 OSD 存储节点可以实现同时对故障数据的恢复操作，极大的降低了数据的恢复时间，更好的提高了用户体验。

3.4.3 高扩展性需求

因为在 ceph 存储系统中，消除了中心节点这个瓶颈，将对应的任务划分到了各个存储节点，使得每个存储节点具备更多的处理能力，从而可以实现高度并行。当进行磁盘扩容时，只需要将最新集群的状态信息反馈给 monitor 节点，在一个版本更新周期内，所有的 OSD 节点和客户端节点都能学习到当前系统中整体磁盘信息。从而实现 ceph 云存储的轻松扩容，且不给服务器带来太大压力。

由于 ceph 存储系统具有在线扩展、替换和升级的功能。当存储介质发生故障时，可以实现数据的自动恢复。当系统中出现了磁盘添加或删除时，ceph 内部均衡机制算法会重新计算数据存储操作，实现数据的均匀分布，不在需要人工去干预。使得对固件的删除、增添和固件的升级不会再影响正常的业务操作。

3.5 总结

本章对基于 ceph 的云存储运维系统，进行了系统的描述，包括对系统总体架构描述，功能需求的描述，以及常见系统性能需求的描述。通过本章节的描述，可以对基于 ceph 的云存储运维系统，在功能和结构上有大致了解。

第4章 系统设计

本章主要对基于 ceph 的云储存运维系统整体系统架构的概述以及 ceph 分布式系统底层数据存储访问的概述。

4.1 系统架构设计

整个系统的设计有好几个部分组成，为了以后升级和维护的需要，在设计的过程中会考虑让每个层次之间尽可能的松耦合。后期随着用户需求不断增加或系统升级的需要，整个系统的维护只需要改动局部实现逻辑，即可达到用户需求或系统升级的需求。

4.1.1 前端架构设计

前端是采用 nodejs、npm、bower 和 gulp 搭建，这些工具为前端开发提供了极大地便利。

(1) nodejs: nodejs 其实是一个 JavaScript 运行环境，是对 V8 引擎进行了封装，提供了相应的 API 接口，方便用户进行调用。

(2) npm: 在安装 nodejs 的过程中，会附带安装上 npm。npm 是 nodejs 程序包的管理工具。根据需求，实现从网上下载所需要的对应安装包。同时还可以管理在项目中，包的依赖关系，用于管理项目中所需程序包。

(3) bower: bower 是个包管理器，用于管理项目中对 js 类库的依赖。通过命令可以将对应的依赖库下载到本地对应位置，是 web 应用的包管理器。

(4) gulp: gulp 是用于前端构建的工具，用于检测文件变化，并按设定的参数，将变化的内容写到对应文件中。Gulp 工具还用于压缩 js 文件，启动 web 服务。

这是前端所涉及到的工具，为前端开发提供了极大的便捷，通过这些工具，只要将所需要的类库写到对应文件，再执行相应命令就会自动下载，并添加到指定位置，为以后的扩展提供极大便利。

4.1.2 后端架构设计

后端包含好几个部分，通过这几个部分的整合，构成了项目的整个后端，同时，这几个部分也使得整个项目在解耦方面做得比较好。后端代码实现如下。

首先是和前台控制器交互的是 Java 代码。在 Java 代码里使用了 springMVC 框架，对前端发出的请求进一步封装后，传输给了更下层的 calamari 模块。calamari 是 ceph 的一个管理平台，相当于对 ceph 系统监控模块，做了进一步的封装，对外提供了一套具有特殊需求的 REST API 接口。当 calamari 接收到上层传输的请求后，再向 ceph 层发送相应请求，ceph 根据具体请求，获取对应数据，然后将这些数据传送到上层的 calamari 模块，再由 calamari 模块将数据传输给 Java 代码模块。在 Java 代码块，将数据做相应的封装，然后传送给前端控制器，从而实现对相应数据信息的请求。

4.2 云存储底层设计

实际开发、测试中，用三台做环境测试。在裸机情况下，可通过 PXE (preboot execute environment, 预启动执行环境) 装机。装机的过程中，通过命令脚本，实现了对 centos 系统和 ceph 系统的安装。在 ceph 的系统中，具有三个网络：业务网、管理网和存储网。

为了保证数据的安全性，采用了冗余备份的方式保存数据。开发中采用了三副本的形式。在 ceph 系统内部，需要有内部网络去完成三副本的写操作，这个网络就是存储网。每个大文件被分割成了易于存储的小的 object。系统要保证，在所有 OSD 节点中，每个 object 有三个副本，对应三个 OSD 节点。选其中一个 OSD 节点为 primary OSD，后期读取数据时，客户端只需要和这个主 OSD 节点交互，即可取得对应数据信息。其写入过程如图 4-1 所示。

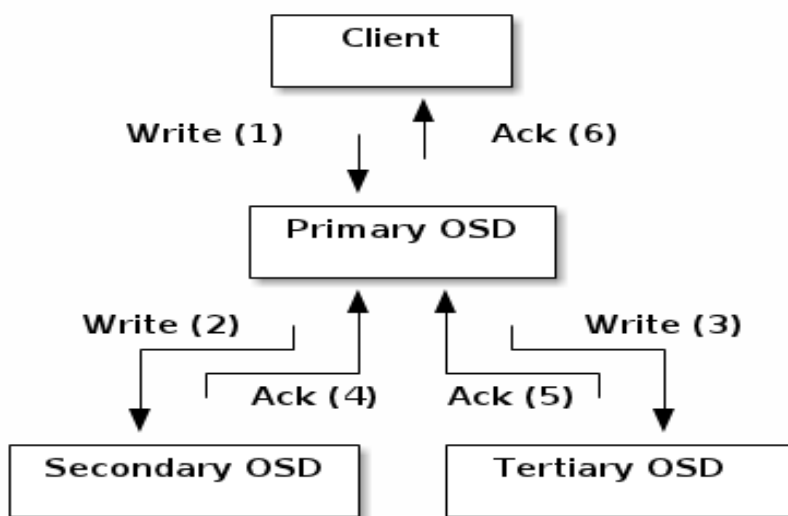


图 4-1 三副本写操作

通过图 4-1 可以看到，进行数据写操作时，确认信息是在三个副本都写好之后才进行确认的，这对于高性能的 ceph 来说是不合理的。在 ceph 内部，进行写操作时，当将数据写到内存缓冲区时，就会向 client 端发送确认信息，这时，client 可以继续往下执行。当三副本都写好后，还会向 client 发个最终确认信号，这时，client 端才可以清除这部分数据。这个过程会设计到两个网络：一个是客户端存储数据到系统内存时的业务网，一个是数据写磁盘时所用的存储网。还有一个管理网，用于对整个集群中各个节点（比如：OSD 节点、MON 节点）的管理。ceph 层结构如图 4-2 所示（每个 HDD（Hard Disk Driver，硬件驱动器）对应一个 OSD 节点，SSD（Solid State Drives，固态硬盘）用作 journal 盘，每个 SSD 盘上承载四个 HDD 盘）。

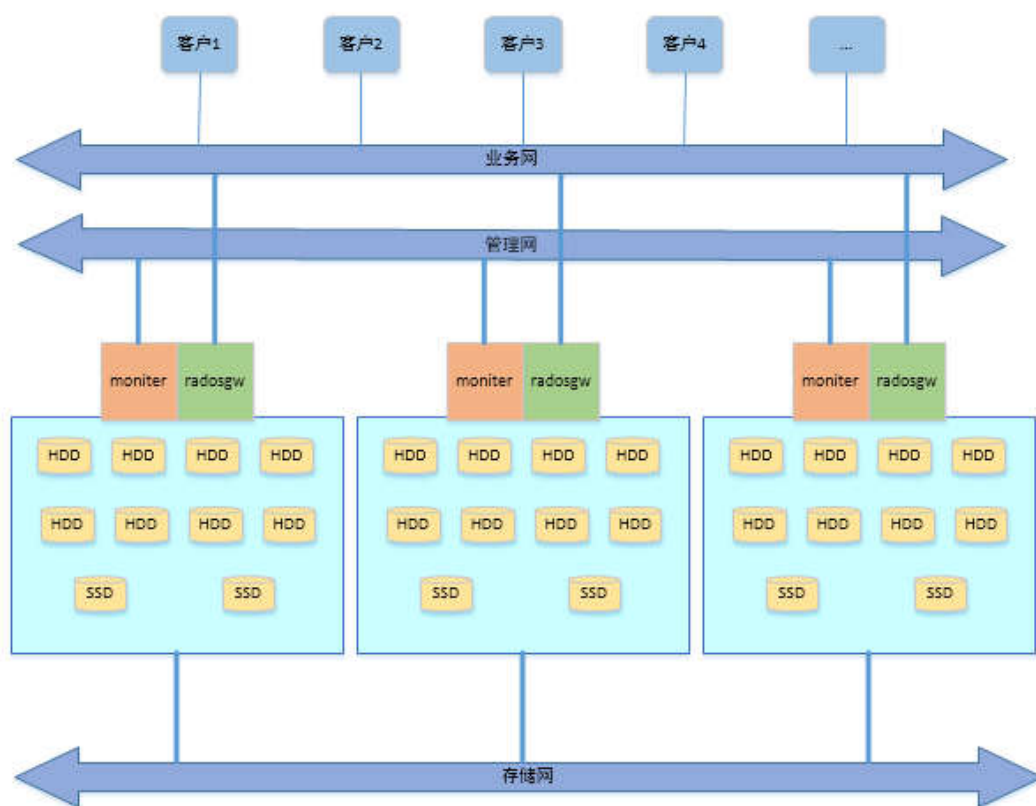


图 4-2 ceph 底层存储网络

4.3 总结

通过本章讲述，可知基于 ceph 云存储运维系统中，所涉及的一些技术和工具，以及后端框架各个交互模块。本章还包括云储存冗余备份机制，ceph 内部网络的通信机制。

第 5 章 系统的部署及管理

本章叙述了基于 ceph 的云储存运维管理系统的各个模块的搭建，以及在实际开发中，各个小组用于开发的代码管理。

5.1 云存储环境搭建和部署

5.1.1 ceph 存储系统的搭建和部署

通过上一章可以了解到，在实际工作中，使用了三台惠普服务器。通过 PXE 对三台惠普服务器进行了装机。在装机的过程中，安装了 tdcube 部署的集成脚本。在实际工作中，ceph 环境的部署过程如下。

（1）配置服务器的管理网：通过在三台服务器上分别执行命令：
`dtcube-deployer config-ip eno1 10.158.113.200/201/202 255.255.255.0 --gw 10.158.113.1` 实现对服务器的管理网的配置。其中 eno1 是各个服务器所在物理机的初始管理网的接口，10.158.113.200/201/202 是初始管理网的 IP，255.255.255.0 是初始管理网的掩码，--gw 为对应的管理网网关。

（2）格式化磁盘：通过执行下面的命令，实现对磁盘的格式化：
`dtcube-deployer format-disk "sdc sdd sde sdf sdg sdh sdi sdj sdk sdl"`。其中 "sdc sdd sde sdf sdg sdh sdi sdj" 为 HDD 盘，用作 OSD 节点，"sdk, sdl" 为 SSD 盘，用作 journal 盘。每个 SSD 盘承载 4 个 HDD 盘，当有数据写入时，用作数据记录，保证数据出现故障时，从日志中恢复。

（3）启动部署节点：选择一台服务器作为部署的虚拟机，并通过命令：
`dtcube-deployer create eno1 10.158.113.223 --gw 10.158.113.1` 实现部署虚机的设置。其中 eno1 是所在物理机的初始管理网的接口，10.158.113.223 为分配给部署虚拟机使用的部署 IP 地址，--gw 是部署虚拟机对应的网关。

（4）部署虚拟机登录：启动部署虚拟机，可以在浏览器上输入 `http://10.158.113.223` 进入部署界面，其中 `http://` 后面的 IP 就是启动部署节点所分配的 IP 地址。

（5）添加主机：在云平台界面，添加对应主机。在添加主机的对话框中，给主机起对应的名称和服务器 IP 地址。根据实际情况，填写主机所在服务器的位置并添加对应的描述信息。IP 地址为配置服务器管理网所对应每台服务器的 IP 地址。通过这个功能，将对应的三台服务器添加进来。

(6) 创建集群：在云平台界面，创建集群。在创建的对话框，会涉及到集群名称和相应的描述。在集群的选项中，添加集群所管理的主机。

(7) 部署：做完上述步骤后会进入部署界面，点击部署即可开始 **ceph** 环境的部署。

到此，**ceph** 的搭建已完成，**ceph** 的基本功能已可以正常运行，各个服务器间可进行相互通信。其网络图如图 5-1 所示。

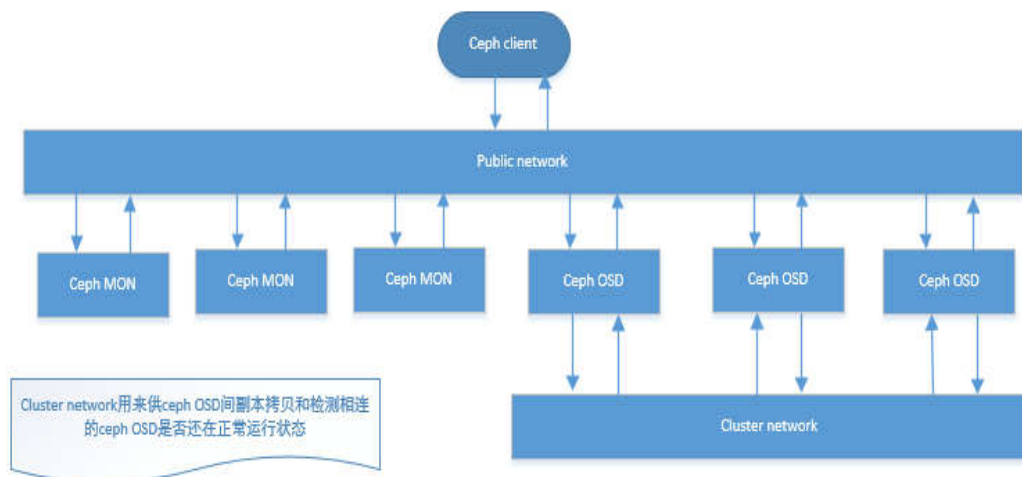


图 5-1 **ceph** 内部网络结构图

通过图 5-1，可以看到，有一个公共集群管理网，用于管理、监控服务器和 OSD 服务器间通讯，而在 OSD 存储节点间，又有另外一个内部网络，用于实现对 **primary** OSD 存储节点从外界获得的存储数据进行备份，实现了利用内部网对数据的备份。

5.1.2 calamari 框架的搭建

在 5.1.1 节，做好了对 **ceph** 的部署，在整个架构中，系统通过调用 **calamari** 对应的接口，去获取所需要的信息。**calamari** 部分的部署如下。

(1) 克隆 **calamari**：通过 **git** 工具分别将 **calamari.git**、**calamari-clients.git**、**Diamond** 代码克隆到本地库。

(2) 建立 **calamari server** 的 **rpm** 包：在 **calamri** 文件夹下，通过可执行文件 **build-rpm.sh** 建立 **calamari server** 的 **rpm** 包。

(3) **calamari server** 的安装：通过 **yum** 安装 **calamari-server***.rpm**（版本中 **calamri server** 对应的安装包）包，实现对 **calamri server** 的安装。

(4) **calamari client** 的安装：在安装 **calamari client** 前，需要安装 **calamari client** 的依赖包，通过 **yum** 安装 **npm**、**ruby**、**rubygems** 和 **ruby-devel** 工具，再

通过 npm 安装 grunt、grunt-cli、bower 和 grunt-contrib-compass 软件，通过 gem 更新安装包，并安装 compass 软件。完成前期工作后，正式进入 calamari client 的安装。进入 calamri-client 文件夹，下载对应依赖包，解压 calamari-clients 对应的压缩包。在 calamari-clients 解压包中，做相应的配置，既可以实现对 calamari client 的安装。

(5) calamari 初始化：通过 calamari-ctl 工具实现 calamari 的初始化。

(6) 在 calamari server 端配置防火墙：通过 iptables 命令实现对 4505、4506、2003 和 2004 端口的开放，从而使得 calamari server 能更好的进行通信。

(7) salt-minion 和 diamond 的安装：在 Diamond 的文件夹下通过 make 命令实现安装包的构建，再通过 yum 命令实现 salt-monion 和 diamond 的安装。

(8) 在 ceph 节点上配置 salt-monion 并启动：修改/etc/salt/minion.d/下的 calamari.conf 文件，添加 calamari 服务器对应的 ip 地址。修改/etc/salt/下的 monion 文件，添加 calamri 服务器对应的 ip 地址。通过 service 命令，重启 salt-minion 服务，通过 service 命令启动 diamon 服务。到此，实现了对 calamari 框架的搭建。

5.1.3 Java 框架的搭建

实际开发环境中，java 部分开发工具是 IntelliJ IDEA，用 IntelliJ IDEA 创建 springMVC+Maven 项目。创建过程如下。

1. 创建工程：创建工程并选择 Maven model 模式。设置支持 Spring framework。

2. 配置 pom.xml 文件：为了创建 springMVC 项目，前期需要导入 commons-logging、AOP、benns、Content、Core、Expression、Web 和 WebMVC 等核心 jar 包。通过<dependency>标签，将这些 jar 包对应的<groupId>、<artifactId>、<version>等信息写到 pom.xml 文件中。开启 IntelliJ IDEA 中自动导入 jar 包功能，实现实时加载所需 jar 包。

3. 配置 web.xml 文件：将 DispatcherServlet 用作处理所有 web 请求，可以实现对请求进行过滤。

4. 添加 SpringConfig.xml 文件：用于配置扫描目录和试图解析。

到此，整个大体框架就形成了，不同功能模块，根据不同需求，添加相应的信息即可在此架构上进行对应开发。

5.1.4 前端框架的搭建

(1) 安装 nodejs: 可以在官网: <http://www.nodejs.org/download/> 上下载和操作系统匹配的 nodejs 版本, 并安装。安装 nodejs 的过程中, npm 也会被一起安装。

(2) 安装 bower: 安装 bower 需要用到上面安装 nodejs 过程中安装的 npm 工具。执行命令: `npm install -g bower` 来安装 bower。可用 bower 工具来管理 web 应用中的软件包。

(3) 安装 gulp: 通过命令: `npm install -g gulp` 实现对 gulp 的安装。用 gulp 工具, 可以启动 web 服务, 通过配置文件可以检测 js 和 jade 文档中的语法错误, 同时能实时更新 js 和 jade 文件的修改, 并将修改后的文件及时更新到指定的目录下。

安装好这些工具后, 可以利用这些工具, 运行开发的项目。通过 gulp 工具, 可以实时的将项目中的修改更新到 Tomcat 上, 结合 chrome 浏览器, 利用其中的断点跟踪工具, 在进行前端开发时, 判断是否将修改及时更新到对应的服务器上, 从而极大的方便前端界面的开发。

到此, 系统中各个环节的搭建基本完成, 相互间的通信, 通过文件的配置, 可以实现各个模块间的通信。

5.2 系统代码的管理

一个系统一般是由一个团队开发, 里面会有很多小组, 每个小组都只是在整个项目的一个分支, 即使在一个小组里, 每个人都会分工不同。如何保证开发者之间在共同开发一个项目时, 能做到协调工作? 这就需要有一个有效的版本管理工具去管理整个项目, 控制整个项目的开发。在工作中, 使用的代码版本管理工具是 git。结合在工作中实际使用的功能, 分析工作中版本控制的应用。

(1) git 安装: 在 <https://github.com> 网站上, 根据自己系统版本下载对应的 git 并安装。

(2) 建立仓库: 通过命令 `git init` 命令初始化命令所在目录。通过 git 的 clone 命令: `git clone username@hostname: /path/to/repository` 将服务器端的项目克隆下来。

(3) 工作区: 在本地仓库中, 有三个区。第一, 工作区里是电脑里实际存放的文件; 第二, 缓存区, 工作区改动并保存的文件, 在没有提交之前都是放在这个区域的; 第三, HEAD 区, 这个区是你最后要提交代码的区域。第一个

区的文件可以通过 `git add <filename>` 命令，添加单个文件到缓存区，或通过 `git add *` 将所有的工作区中改动的文件添加到缓存区。通过 `git commit -m “提交信息说明”` 命令，将缓冲去的代码提交到 HEAD 区。

(4) 提交本地代码：当本地项目开发到一定程度，需要将本地代码提交到远程服务器时，通过 `push` 命令，将本地仓库的 HEAD 区的代码提交到远程服务器上。具体命令为：`git push origin <branch>`。通过这条命令，可以将本地的代码推送到远程代码库中。

(5) 分支的使用：在代码的开发过程中，分支最常被用在做模块功能开发。有时，在原有技术上，需要添加新特性，也会单独拉出一个分支，以防新添加的特性对模块分支的影响，导致迟迟不能将模块合入主线。通过使用分支，可以减少对代码 master 分支上的维护，使得每个小组只需要维护他们所创建的分支即可。由开发小组自己去维护自己小组的分支，相对于专门维护 master 分支效率高很多。当小组将对应的功能模块开发完成，再一次性将对应的分支合并到主线上。通过这个过程，分支实现了对应功能模块开发的全过程。通过 `git checkout -b “分支名称”` 命令实现分支的创建。在创建分支后，通过 `git checkout master` 可以将当前分支变成 master 分支，在 master 分支上时，可以通过 `git branch -d “分支名称”` 命令，将创建的分支删除。

(6) 本地代码更新：在开发的工程中，别人也在开发，同时开发一个模块就会出现冲突。最好的解决办法，就是在推送本地代码前，去拉远程代码到本地，检测是否有冲突，如果有冲突，在本地解决了冲突后，才能更好的将代码推送到远程服务器。冲突出现在有两个或两个以上的人修改同一个文件。可以使用 `git pull` 命令或者使用 `git merge <branch>` 命令实现将远程的代码自动合并到本地。不过有时会出现冲突，这时就需要手动去修改对应的冲突文件。修改后，通过命令 `git add <filename>` 命令，标记已经成功解决冲突，并将本地代码更新为最新版本。此时，可以顺利将自己本地的代码推送到远程服务器端代码库。

(7) 代码回滚：在项目开发过程中，由于一时的理解偏差，或者操作不当，会出现当前 HEAD 区的代码不是你想要的代码，或者你改错了东西，使得本地 HEAD 区的项目无法正常运行起来。这时，你想回到上个 HEAD 版本，你可以通过命令 `git checkout -- <filename>` 来挽回刚才的错误操作。还有一种方法，就是从服务器上获取最新版本代码，并将本地的代码指向它。通过命令 `git fetch origin` 获取最新主线代码，通过命令 `git reset --hard origin/master`，实现将本地代码指向上一个版本的代码，从而实现代码的一次回滚。

上述内容，结合了在实际工作中用到的关于版本控制、新建分支开发新功能模块等实际需求功能，讲述了在工作中 `git` 版本控制器对应的应用。

5.3 总结

通过本章的叙述，对在项目中的开发环境的部署有了大致的了解。通过结合实际工作中的需求，叙述了分布式版本控制工具 `git` 的基本功能，运用 `git` 的基本功能，为各个开发小组提供了版本控制的基本需求。

第 6 章 系统实现和测试

本章主要讲述了实际工作中，开发基于 ceph 的云存储运维系统所涉及到的工具，以及各个模块所涉及到的内容。本章后半部分介绍了云存储运维系统，在设计时用到的功能测试。

6.1 开发工具及环境

在整个项目中，主要负责前端运维界面的开发。开发过程中，所使用到的工具和开发环境如下。

- (1) 软件环境：windows10。
- (2) 硬件环境：16G 内存 + 256G 固态硬盘。
- (3) 开发工具：IDEA 14.1.7 + JDK 1.8 + Tomcat 7.0 + chrome 浏览器 + Beyond compare。

6.2 云存储运维系统的实现

6.2.1 前期准备

(1) 项目的克隆：使用 git 的工具，克隆远程服务器端上的代码库到本地。根据各个小组开发功能的不同，创建自己小组对应的分支。在后期开发时，直接将主分支代码，切换到对应小组分支，并在自己的分支上做相应功能开发。

(2) 搭建前端环境：通过 nodejs、bower 和 gulp 等工具，将项目中引用的控件导入到实际开发环境中。系统中安装 nodejs 后会带有 npm 工具，通过 npm 命令，实现 bower 和 gulp 的安装，通过 bower 将需要的控件导入项目中，通过 gulp 加载 js 文件，并将 jade 文件转换成对应的 HTML 文件。

(3) 创建工程：使用 IDEA 集成开发工具，选择 maven 模式，将项目导入进来，这时，系统会根据 pom.xml 配置文件，去自动加载所有配置文件制定的可用的 jar 包。

6.2.2 运维系统的实现

前期，笔者主要负责运维系统界面的展现，和项目组组长讨论运维系统包

含的功能模块和界面显示风格等信息，利用 Axure RP 7.0 工具，模拟出后期想要开发的效果。运维系统整体结构的展现效果经认可后，进入运维系统开发阶段。在此阶段，笔者主要负责前端界面的开发、前端和后台的对接。

(1) 前端界面的开发：当用户进入运维管理界面，用户要能获取所有集群列表信息，可以通过每个集群的“详情”连接，可以查看对应的每个集群的信息概览。在众多信息概览里，要突出主要信息，即磁盘容量和磁盘使用量的信息。通过详情，用户可以查看每个 `cluster` 类中所包含的详细信息。

在 `cluster` 集群的首界面，会罗列所有的集群，在集群列表中展现集群最重要的信息，即存储信息。通过“详情”选项，可以查看，此集群的详细信息。针对其中比较复杂两个模块 OSD 和 POOL，又单独的开发出来两个模块，用于详细展现 OSD 和 POOL 里面的详细信息。

在 OSD 界面，展现了在每个机架上所对应的服务器和部署在服务器上面的 OSD 信息，以及这个集群所有 OSD 存储节点所在的机架位置和对应的服务器。可以通过点击对应 OSD 列表，切换到 OSD 详细信息界面，展现所有 OSD 的详细列表，还可以点击对应 OSD 查看每个 OSD 的详细信息。针对不同故障域，可以选取不同的显示方式。故障域主要包含两个：一个是 `bucket` 故障域，另一个是 `rack` 故障域。当选择按照 `bucket` 故障域显示时，同一个 `bucket` 的 OSD 节点会集中到一起。如果选择 `rack` 显示模式时，会按照实际机架去显示每台机架上 OSD 的情况。

在 OSD 详细信息界面，含有均衡选项。其中包括手动均衡和自动均衡。在手动均衡功能界面，通过填写相应的数值，点击执行，即可立刻进行均衡操作。手动均衡旁边有个自动均衡。当长时间使用存储空间时，就会出现数据在磁盘分布不均的情况。当有磁盘损坏或有新的磁盘插入时，也会出现磁盘分布不均衡。通过自动均衡选项，重新分配数据分布。

在存储池界面记录着 POOL 的详细信息、每个 POOL 所拥有的副本、所具有的 PG 个数、磁盘总容量和已经使用过的磁盘大小。在详细列表中有 IOPS 属性，用于表示 POOL 中，当进行数据读、写时所对应的速率值。通过点击“查看”对应的连接，可以查看到每个 POOL 所对应的 IOPS 在近一段时间内值的变化。IOPS 对应的信息，其单位用次每秒。当点击 POOL 列表里读写速率的查看时，可以看到对应 POOL 的读写速率的记录。还可以通过点击每个 POOL 记录后面的设置，进行对应的 POOL 的设置。可供设置的属性包括 PG 个数和磁盘最大容量。

到此，大致描述了在云储存运维系统在开发中，所开发的功能，随着后期的需求，会在相应的界面添加对应的功能模块。

(2) 后台功能的实现：在原本的框架中，前端到后台 Java 交互的是利用的 angularjs 中的 \$resource 模块，初始化一个变量 v，然后用 v 调用 \$resource 模块中的 get(function (res))，最终将结果返回给 res 参数。后面，因为云存储运维界面的特殊需求，笔者开发了，适合云存储运维系统的调用模块。通过重新编写后台调用模块，实现适合云存储运维系统自己界面的后台交互模块。通过 JavaScript 中的 Promise 模块，在 Promis 中使用 XMLHttpRequest 机制，实现了更灵活的处理异步函数的调用模块。

前台 angularjs 控制器传过来的 url 请求，会走到使用 springMVC 搭建框架中的控制器模块，即通过 @Controller 标记的类。系统会找到对应的 Java 代码中，用作控制器模块的 Java 代码，再通过 @RequestMapping 层层匹配，找到 angularjs 发过来的请求所对应的 Java 代码块。到此，实现了前台 angularjs 到后台 Java 代码的调用。

在控制器端的代码块接收到来自 angularjs 发过来的请求后，通过封装，将请求发给中间用于做数据处理的模块，即 service 模块。这个模块用于转发请求，同时处理从更底层返回的数据，实现对数据的拆分和组合，从中提取用户需要的信息，去掉用户不感兴趣的信息，并将数据封装成 json 格式的字符串（方便前端解析、识别、处理），再负责将数据转发给上一层。这一层会将上一层发送过来的请求，进行本层的相应封装，然后继续向下转发给和 calamari 部分交互的 Java 代码层，此层称为 calamariApi 层。

在 calamariApi 层，会接受来自上层 service 层发过来的请求，通过进一步的封装，调用 CloseableHttpClient 类的 execute() 方法，再通过配置文件，实现和底层 calamari 的交互，再由 calamari 调用自身的功能模块实现，封装数据，实现对 ceph 模块的调用。

以上，主要是云存储运维系统的前端部分和后台功能部分。对于更深层的系统调用，有其他小组负责维护开发。

6.3 云存储运维系统的功能测试

6.3.1 测试需求分析

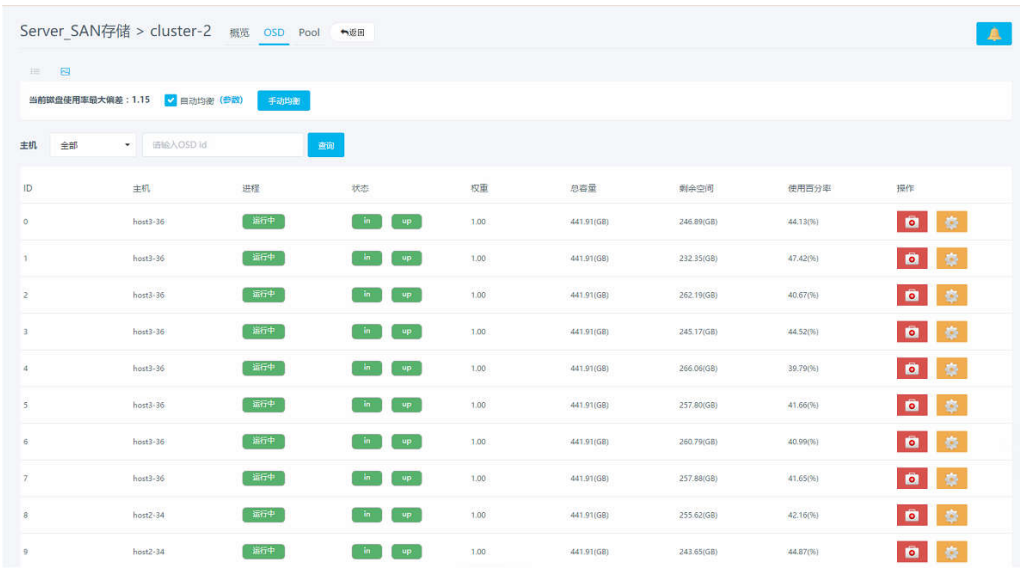
基于 ceph 的云存储运维系统，要做到能实时监控系统运行状态。当系统出现故障实时，能及时将对应信息反映到界面上，以便维护人员及时采取有效措

施，以维护系统正常、高效运行。为此，在系统设计和实现时，在对应监控界面上，要定时去后台获取状态信息数据。对于不同监控模块，获取状态数据频率会有所不同。比如，对于监控器状态、PG 个数、磁盘总容量、POOL、归置组个数和 OSD 存储节点状态等信息，可以在较短暂的时间内获取到对应信息，而对于 OSD 存储节点详细信息监控，其中包括每个 OSD 存储节点对应的主机，OSD 存储节点运行状态，OSD 存储节点磁盘使用等信息，当获取对应 OSD 存储节点详细信息时耗时较长。设计时，对于较快能获取对应状态信息的监控界面，云存储运维系统采取每十秒刷新一次，而对于获取状态时间较长的模块，实行每一分钟刷新一次。以 OSD 存储节点为例，讲述在系统开发过程中，所进行的系统性能测试。

6.3.2 测试方法和结果

测试方法：在后台对 OSD 存储节点执行相应操作，观察前端监控界面信息变化。

结果：以 ID 为 0 的 OSD 存储节点为例，进行状态信息监控。如下图 6-1 所示，此时，对应的 OSD 存储节点状态为 in 和 up。























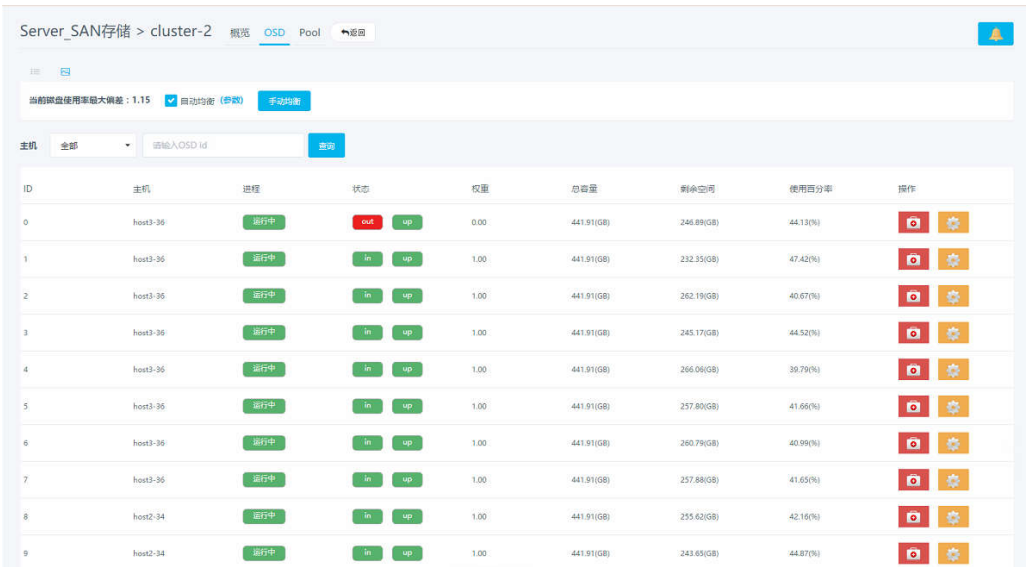
ID	主机	进程	状态	容量	总容量	剩余空间	使用百分比	操作
0	host3-36	运行中	in up	1.00	441.91GB	246.89GB	44.13%	 
1	host3-36	运行中	in up	1.00	441.91GB	232.35GB	47.42%	 
2	host3-36	运行中	in up	1.00	441.91GB	262.19GB	40.67%	 
3	host3-36	运行中	in up	1.00	441.91GB	245.17GB	44.52%	 
4	host3-36	运行中	in up	1.00	441.91GB	266.06GB	39.79%	 
5	host3-36	运行中	in up	1.00	441.91GB	257.60GB	41.66%	 
6	host3-36	运行中	in up	1.00	441.91GB	260.79GB	40.99%	 
7	host3-36	运行中	in up	1.00	441.91GB	257.88GB	41.65%	 
8	host2-34	运行中	in up	1.00	441.91GB	255.62GB	42.16%	 
9	host2-34	运行中	in up	1.00	441.91GB	243.65GB	44.87%	 

图 6-1 ID 为 0 的 OSD 状态为 in 和 up 状态图

执行命令：ceph osd out 0，此时，对应 ID 为 0 的 OSD 存储节点将会被移除 ceph 存储系统，最长等待一分钟，监控界面信息就会更新。更新后，如图 6-2 所示。



The screenshot shows the 'OSD' tab in the Ceph monitoring interface. At the top, it says 'Server_SAN存储 > cluster-2'. Below that, there are tabs for '概览', 'OSD', 'Pool', and '节点'. The 'OSD' tab is selected. A summary bar shows '当前磁盘使用率最大偏差: 1.15' and buttons for '自动均衡 (停职)' and '手动均衡'. Below this is a search bar for '主机' (Host) and 'ID' (ID). The main table lists 10 OSDs. OSD 0 is highlighted with a red 'out' status and a green 'up' button. The other OSDs are in 'in' status with green 'up' buttons.





















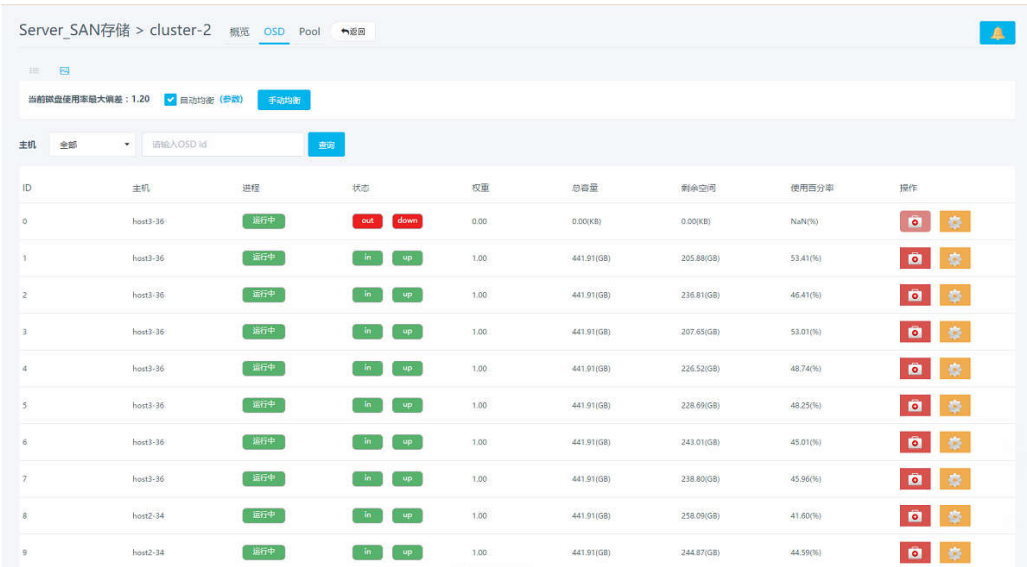
ID	主机	进程	状态	权重	总容量	剩余空间	使用百分率	操作
0	host3-36	运行中	out up	0.00	441.91(GB)	246.89(GB)	44.13(%)	 
1	host3-36	运行中	in up	1.00	441.91(GB)	232.35(GB)	47.42(%)	 
2	host3-36	运行中	in up	1.00	441.91(GB)	262.19(GB)	40.67(%)	 
3	host3-36	运行中	in up	1.00	441.91(GB)	345.17(GB)	44.52(%)	 
4	host3-36	运行中	in up	1.00	441.91(GB)	266.06(GB)	39.79(%)	 
5	host3-36	运行中	in up	1.00	441.91(GB)	257.80(GB)	41.66(%)	 
6	host3-36	运行中	in up	1.00	441.91(GB)	260.79(GB)	40.99(%)	 
7	host3-36	运行中	in up	1.00	441.91(GB)	257.88(GB)	41.65(%)	 
8	host2-34	运行中	in up	1.00	441.91(GB)	255.62(GB)	42.16(%)	 
9	host2-34	运行中	in up	1.00	441.91(GB)	243.65(GB)	44.87(%)	 

图 6-2 ID 为 0 的 OSD 状态为 out 和 up 状态图

执行命令：`ceph osd down 0`，此时，对应 ID 为 0 的 OSD 将会被 down 掉，最长等待一分钟，监控界面信息会更新。更新后，如图 6-3 所示。



The screenshot shows the 'OSD' tab in the Ceph monitoring interface after the command 'ceph osd down 0' was executed. OSD 0 is now in a 'down' state, indicated by a red 'down' button next to the 'out' status. The '使用百分率' (Usage percentage) is now 'NaN(%)'. The other OSDs remain in 'in' status with green 'up' buttons.





















ID	主机	进程	状态	权重	总容量	剩余空间	使用百分率	操作
0	host3-36	运行中	out down	0.00	0.00(KB)	0.00(KB)	NaN(%)	 
1	host3-36	运行中	in up	1.00	441.91(GB)	205.88(GB)	53.41(%)	 
2	host3-36	运行中	in up	1.00	441.91(GB)	236.81(GB)	46.41(%)	 
3	host3-36	运行中	in up	1.00	441.91(GB)	207.69(GB)	53.01(%)	 
4	host3-36	运行中	in up	1.00	441.91(GB)	226.52(GB)	48.74(%)	 
5	host3-36	运行中	in up	1.00	441.91(GB)	228.69(GB)	48.25(%)	 
6	host3-36	运行中	in up	1.00	441.91(GB)	243.01(GB)	45.01(%)	 
7	host3-36	运行中	in up	1.00	441.91(GB)	238.80(GB)	45.96(%)	 
8	host2-34	运行中	in up	1.00	441.91(GB)	258.09(GB)	41.60(%)	 
9	host2-34	运行中	in up	1.00	441.91(GB)	244.87(GB)	44.59(%)	 

图 6-3 ID 为 0 的 OSD 状态为 out 和 down 状态图

通过上面的测试，可知，当云存储系统出现故障时，运维工作人员就会很快会发现问题，从而采取必要措施，弥补系统出现的问题。其他模块，比如 `moniter` 状态、集群健康状态等的测试类似，通过后台执行命令对应命令，等待一定时间，各个模块的状态可以被前端捕获到并显示出来。

上面所述主要是功能上面的测试，也是开发过程中做的主要测试，实际开发中还会涉及到界面排版问题、边界值是否显示异常和极端数据对界面显示的影响等测试。当项目基本功能完成之后，需要在各个浏览器间进行测试。在这

次项目开发完成后，在浏览器测试就出现了问题。当时开发是在 **chrome** 浏览器上进行开发。后期在 **IE** 浏览器上运行，打开运维界面时，监控信息不能正常显示。最后跟踪原因，是因为 **IE** 浏览器不支持 **Promise** 机制。

6.4 总结

通过本章节，可以大致了解到，云存储运维系统中，所涉及到的功能模块，以及各个模块的交互逻辑。通过后面的测试可知，当系统出现某个组件出现故障时，通过云存储运维系统，可以实时监测到故障点，从而为维护人员保证系统健康稳定运行，提供了及时有效的信息。

第 7 章 总 结

基于 ceph 的云存储运维系统，是公司项目 DTCube 中的一个运维模块。TDCube 是基于混合云框架理念设计的，是一种可以横向扩展的计算和存储超融合项目。不仅提供计算虚拟化、对象存储、分布式块存储、VPC (Virtual Private Cloud, 虚拟私有云) 虚拟网络、云运维管控服务和实现云服务器服务 (ECS)、对象存储服务 (OSS) 资源的一体化，还提供了混合云备份等混合云功能。笔者在项目中主要的职责就是负责云存储运维模块中，用户界面和与用户界面交互的 Java 后台部分的开发。实现了对后台云储存系统的实时监控，并将故障信息尽快的反馈给运维人员，以便运维人员及时、有效的采取相应措施，避免以后数据灾难的发生，提高了系统的稳定性、安全性。所开发的运维模块，成功应用在公司中标的苏州银行的 TDcube 项目中。

本文主要工作有以下几个方面：

(1) 在本文第 1 章，叙述了当今社会对存储的需求。通过传统存储和云存储的对比，凸显出云储存的优越性。在此基础之上，引出了对基于 ceph 的云存储运维系统的需求。

(2) 在本文第 2 章，对云储存系统应用场景进行了叙述，可使读者能更直观的感受云存储存在的意思。本章还叙述了 ceph 分布式系统的实现机制，使读者能更加深入的了解，基于 ceph 的云存储系统的实现机制。

(3) 在本文第 3 章，对基于 ceph 的云存储运维系统的整体架构、业务流程、功能需求和性能需求进行了叙述，让读者对云存储运维系统有更直观、详尽的认识。

(4) 本文的第 4 章和第 5 章，对基于 ceph 的云存储运维系统的系统设计、系统部署和代码管理进行了叙述。通过这两章的叙述，可以让读者对云存储运维系统整体框架的搭建有更清晰的认识。

(5) 本文的第 6 章，叙述了开发系统所需要的工具，系统实现的功能以及在系统实现过程中，所涉及到的系统功能测试。通过本章，读者可以了解到云存储运维系统所实现的功能模块，以及在开发过程中所涉及到的功能测试方式。

在整个实习的过程中，有辛酸，有坎坷也有喜悦。在实习的过程中，感知到了基础知识的重要。通过自己的努力，使笔者更加坚信，只要你去努力付出，总会获得应有的汇报。这才是事业的起点，以后的路还很长，只有脚踏实地的去走，才能获得理想的效果。下面是笔者在实习过程中的一点心得。

实习期间，有段时间做界面开发，其中就用到了 HTML5、JavaScript、CSS 等语言，当时的感觉是既然在用这些语言，就应该尽快将这些语言中所有的知识都去掌握。于是就去买对应的书籍，准备去系统的学习。在学习的过程中，又会出现新的事物，在看这些书籍的过程中有用到了 nodejs、openstack 等相关的知识，根据刚开始的思想，毫无疑问，又去买对应的书籍去系统的学习。刚开始还好，涉及的东西不是太多，实习工作量不大，但后期又有很多新的知识出现。当时的感觉，时间完全不够用，毕竟每天还要去做小组分给自己的任务。看书只能抽时间。因为看书是通读，又因为时间紧，有好多东西根本不能很快消化掉。时间久了，以前看到的内容，因长时间不用，有好多都会忘记。感觉这种方式对待新出现的事物，效率特别低。同时，有好多知识，在实际开发中几乎从来未曾出现过。因为将自己好多时间用在通读书籍上，所以感觉在实际工作中因投入的时间、精力较少，而效率较低。当时笔者就怀疑，这种面对新事物的方法是否可取。通过以前的学习经验，给笔者的认知是，既然遇到新知识了，就应该系统的，认真的去将这门语言所涉及到的东西认真去看一遍。在有工作的情况下，感觉这种方法对笔者来说不适应。毕竟时间没有在校期间那么充裕，同时，通读因为没有侧重点，使得吸收效率差，还有可能会遇到一些需要有一定工作经验的人去研读的东西，这些东西很容易使初始读者对新的技术产生恐惧和排斥，对新的事物失去研究的兴趣。后来通过和别人交流，结合自己的实际情况，感觉遇到新事物，应从实际需要出发，有针对性的去学习当时需要的知识模块，尽可能多的先看和自己实际工作有关的知识。如果时间够，再以当前学的为基点，向外不断延伸，从而实现有针对性，有明确目的的去学习新的事物，利用有限的时间，尽可能快的掌握一种新技术。当掌握一定程度后，感觉新技术需要自己有个全面的掌握，那时再去系统的学习。这种情况下，有一定实战经验，再加上有针对性，学习效率自然而然的就会提高上来。

参 考 文 献

- [1] 赵铁柱.分布式文件系统性能建模及应用研究[D].广州:华南理工大学,2011.
- [2] 钱宏蕊.云存储技术发展及应用[J].电信工程技术与标准化,2012,25(4):15-20.
- [3] Wei S A. Ceph: Reliable, scalable, and high-performance distributed storage[D].UNIVERSITY OF CALIFORNIA,2007.
- [4] Mesnier,Mike R.Ganger,and Erik Riedel.Object-based storage[J].Communications Magazine,IEEE,2003,41(8):84-90.
- [5] Riedel Erik,Faloutsos Christos,Gibson G A.et al.Active disks for large-scale data processing[J]. IEEE Computer,2001,34(6):68-74.
- [6] Yan CR, Shen JY, Peng QK, et al. A throughput-driven scheduling algorithm of differentiated service for web cluster.Wuhan University Journal of Natural Sciences, 2006, 11(1):88-92.
- [7] 龚高晟,通用分布式文件系统的研究与改进[D].广州:华南理工大学,2010.
- [8] 李翔.Ceph 分布式文件系统的研究及性能测试[D].西安:西安电子科技大学,2014.
- [9] 许敏,分布式文件系统容错机制的研究与实现[D].西安:西安电子科技大学,2012.
- [10] Weil S A,Brandt S A,Miller E L,et al.Ceph:A scalable,high-performance distributed file system [C] Berlin:Proceddings of the 7 th Symposium on Operating Systems Design and Implementation(OSDI),2006:307-320.
- [11] 廖彬,于炯,孙华等.等基于云存储结构重配置的分布式存储系统节能算法[J].计算机研究与发展,2013:3-18.
- [12] 沈良好,吴庆波,杨沙洲.基于 ceph 的分布式处处节能技术研究[J].先进计算与数据处理,2015,41(8):13-17.
- [13] 王敬轩,分布式文件系统存储效率优化研究[D].华中科技大学,2013.
- [14] 廖舒恬.安全对象分布式文件系统的设计与实现[D].华中科技大学,2013.
- [15] 段剑弓.存储系统 NAS 和 SAN 的差异和统一[J].计算机应用研究,2004,21(12):94-97.
- [16] 郑传建.Ceph 对象文件系统添加任务迁移特性的研究[D].武汉理工大学,2014.
- [17] Holmquist L E,Redstrom J,Ljungstrnd P.Token-based access to digital information proceeding [M].Berlin:SpringerVerlag,2000.
- [18] 杨飞,朱志祥,梁小江.基于 ceph 对象存储的云网盘设计与实现[J].图像·编码与软件,2015,28(10):96-99.
- [19] 蔡官明.开放式云存储服务平台设计及移动云盘应用开发[D].广州:华南理工大学,2013
- [20] Chang F,Dean J,GheMawat S, et al. Bigtable: A distributed storage system for structured data[J].ACM Transations on Computer Systems (TOCS),2008,26(2):4.

- [21] 聂瑞华,张科伦,梁军.一种改进的云存储系统容错机制[J].计算机应用研究,2013,30(12):3724-3728.
- [22] Corbett J C,Dean J,Epstein M,et al.Spanner:Google's globally-distributed database[C].Proceedings of OSDI. 2012,1
- [23] 王芳,陈亮.对象存储系统中基于负载均衡的设备选择算法[J].华中科技大学学报:自然科学版,2007,35(10):46-49.
- [24] Dean J,Ghemawat S.MapReduce:simplified data processing on large cluster[J].Communications of the ACM,2008,51(1):107-113.
- [25] Zeng,Ling-Fang,Dan,and Ling-jun Qin.SOSS:smart object-based storage system[J].Machine Learning and Cybernetics,2004:26-29.
- [26] Wut,LeeW,Lin Y, etal. Dynamic load balancing mechanism base done loudstorage [C]. Computing, Communication sand Application conference(Com-Com Ap). HongKong, IEEE, 2012: 102-106.
- [27] 符永康. 云存储中数据安全关键技术研究及系统实现[D].北京:北京邮电大学,2013.
- [28] 张敏.基于对象存储文件系统研究[D].成都:电子科技大学,2012.
- [29] 李青山,魏彬.Ceph 分布式文件系统的研究性能测试[J].西安电子科技大学,2014, 29(5) :1-15.
- [30] 谢雨来,冯丹,王芳.主动存储技术及其在对象存储中的实现[J].中国计算机学会通讯,2008,4(11):27~33.
- [31] Hsiao H C,Chung H Y,shen H, etal.Load rebalancing for distributed file system,IEEE Transactionson,2013,24(5):951-962.
- [32] 袁艳丽.存储系统主动队形实现机制研究[D].武汉:华中科技大学,2011.
- [33] Hyeran Lim,Vikram Kapoor,Chirag Wighe.Active Disk File System:A Distributed,Scalable File System[J].In:Proceedings of the Eighteenth IEEE Symposium,Mass Storage Systems and Technologies,2001,32(8):101-114.
- [34] 方圆.基于对象存储元数据管理策略的研究与实现[D].郑州:将防局信息工程大学,2012.
- [35] Mesier,Mike,Gregory R.Ganger,and Erik Riedel.Object-based storage[J].Communications Magazine,IEEE,2003,41(8):84-90.
- [36] Sacks,David .Demystifying Storage Networking DAS,SAN,NAS,NAS Gateways,Fibre Channel, and iSCSI[J].IBM Storage Networking,2001,23(7):3-11.

附录

附件 A

Promise 异步机制的实现

```
/**
 * Created by sun on 1/11 0011.
 */
App.factory('PageHandle', ['$q',
  function ($q) {
    'use strict';
    var pageHandle = {};
    var requestMap = {};
    pageHandle.getData = function (url) {
      var req = new XMLHttpRequest();
      var promise = new Promise(function (resolve, reject) {
        req.open('GET', url);
        req.send();
        req.onload = function () {
          if (req.status === 200) {
            resolve(JSON.parse(req.responseText));
          } else {
            resolve('request error');
          }
        }
      });
      req.onerror = function () {
        resolve('request error');
      }
    };
  }
]);
```



```
    };

    req.onabort = function () {
        reject('abort handle');
    };

    req.ontimeout = function () {
        resolve('timeout')
    };

    req.onreadystatechange = function () {
        if (req.readyState === XMLHttpRequest.DONE) {
            delete requestMap[url];
        }
    }
});

requestMap[url] = {
    promise: promise,
    request: req
};

return promise;
};

pageHandle.abort = function () {
    for (var url in requestMap) {
        var req = requestMap[url].request;
        if (req.readyState !== XMLHttpRequest.UNSENT &&
req.readyState !== XMLHttpRequest.DONE) {
            req.abort();
        }
    }
}
```

```
};  
    return pageHandle;  
});
```

致 谢