华章IT
云计算与虚拟化技术丛书

Ceph分布式存储实战

Ceph中国社区 著

十余位专家联袂推荐，Ceph中国社区专家撰写，权威性与实战性毋庸置疑。

系统介绍Ceph设计思想、三大存储类型与实际应用、高级特性、性能测试、调优与运维。

机械工业出版社
China Machine Press

（云计算与虚拟化技术丛书）

Ceph分布式存储实战

Ceph中国社区　著

ISBN：978-7-111-55358-8

本书纸版由机械工业出版社于2016年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录

本书赞誉

序

前言

第1章　初识Ceph

1.1　Ceph概述

1.2　Ceph的功能组件

1.3　Ceph架构和设计思想

1.4　Ceph快速安装

1.4.1　Ubuntu/Debian安装

1.4.2　RHEL/CentOS安装

1.5　本章小结

第2章　存储基石RADOS

2.1　Ceph功能模块与RADOS

2.2　RADOS架构

2.2.1　Monitor介绍

2.2.2 Ceph OSD简介
2.3 RADOS与LIBRADOS
2.4 本章小结
第3章 智能分布CRUSH
3.1 引言
3.2 CRUSH基本原理
3.2.1 Object与PG
3.2.2 PG与OSD
3.2.3 PG与Pool
3.3 CRUSH关系分析
3.4 本章小结
第4章 三大存储访问类型
4.1 Ceph FS文件系统
4.1.1 Ceph FS和MDS介绍g
4.1.2 部署MDS
4.1.3 挂载Ceph FS

4.2 RBD块存储
4.2.1 RBD介绍
4.2.2 librbd介绍
4.2.3 KRBD介绍
4.2.4 RBD操作
4.2.5 RBD应用场景
4.3 Object对象存储
4.3.1 RGW介绍
4.3.2 Amazon S3简介
4.3.3 快速搭建RGW环境
4.3.4 RGW搭建过程的排错指南
4.3.5 使用S3客户端访问RGW服务
4.3.6 admin管理接口的使用
4.4 本章小结
第5章 可视化管理Calamari
5.1 认识Calamari

5.2 安装介绍
5.2.1 安装calamari-server
5.2.2 安装romana（calamari-client）
5.2.3 安装diamond
5.2.4 安装salt-minion
5.2.5 重启服务
5.3 基本操作
5.3.1 登录Calamari
5.3.2 WORKBENCH页面
5.3.3 GRAPH页面
5.3.4 MANAGE页面
5.4 本章小结
第6章 文件系统——高性能计算与大数据
6.1 Ceph FS作为高性能计算存储
6.2 Ceph FS作为大数据后端存储
6.3 本章小结

第7章　块存储——虚拟化与数据库
7.1　Ceph与KVM
7.2　Ceph与OpenStack
7.3　Ceph与CloudStack
7.4　Ceph与ZStack
7.5　Ceph提供iSCSI存储
7.6　本章小结
第8章　对象存储——云盘与RGW异地灾备
8.1　网盘方案：RGW与OwnCloud的整合
8.2　RGW的异地同步方案
8.2.1　异地同步原理与部署方案设计
8.2.2　Region异地同步部署实战
8.3　本章小结
第9章　Ceph硬件选型、性能测试与优化
9.1　需求模型与设计
9.2　硬件选型

9.3　性能调优
9.3.1　硬件优化
9.3.2　操作系统优化
9.3.3　网络层面优化
9.3.4　Ceph层面优化
9.4　Ceph测试
9.4.1　测试前提
9.4.2　存储系统模型
9.4.3　硬盘测试
9.4.4　云硬盘测试
9.4.5　利用Cosbench来测试Ceph
9.5　本章小结
第10章　自定义CRUSH
10.1　CRUSH解析
10.2　CRUSH设计：两副本实例
10.3　CRUSH设计：SSD、SATA混合实例

10.3.1　场景一：快 - 慢存储方案
10.3.2　场景二：主 - 备存储方案
10.4　模拟测试CRUSH分布
10.5　本章小结
第11章　缓冲池与纠删码
11.1　缓冲池原理
11.2　缓冲池部署
11.2.1　缓冲池的建立与管理
11.2.2　缓冲池的参数配置
11.2.3　缓冲池的关闭
11.3　纠删码原理
11.4　纠删码应用实践
11.4.1　使用Jerasure插件配置纠删码
11.4.2　ISA-L插件介绍
11.4.3　LRC插件介绍
11.4.4　其他插件介绍

11.5 本章小结
第12章 生产环境应用案例
12.1 Ceph FS应用案例
12.1.1 将Ceph FS导出成NFS使用
12.1.2 在Windows客户端使用Ceph FS
12.1.3 OpenStack Manila项目对接Ceph FS案例
12.2 RBD应用案例
12.2.1 OpenStack对接RBD典型架构
12.2.2 如何实现Cinder Multi-Backend
12.3 Object RGW应用案例：读写分离方案
12.4 基于HLS的视频点播方案
12.5 本章小结
第13章 Ceph运维与排错
13.1 Ceph集群运维
13.1.1 集群扩展

13.1.2　集群维护
13.1.3　集群监控
13.2　Ceph常见错误与解决方案
13.2.1　时间问题
13.2.2　副本数问题
13.2.3　PG问题
13.2.4　OSD问题
13.3　本章小结

本书赞誉

正如OpenStack日渐成为开源云计算的标准软件栈，Ceph也被誉为软件定义存储开源项目的领头羊。细品本书，慢嗅“基础理论讲解简明扼要，技术实战阐述深入全面”之清香。千言万语，不如动手一战。Ceph爱好者们，请启动机器，拿起本书，早日踏上Ceph专家之路。

——陈绪，博士，英特尔中国云计算战略总监，中国开源软件推进联盟常务副秘书长，2015年中日韩东北亚开源论坛最高奖项“特别贡献奖”获得者

Ceph是主流的开源分布式存储操作系统。我们看到越来越多的云服务商和企业用户开始考察Ceph，把它作为构建“统一存储”和“软件定义存储”的可信赖解决方案。Ceph的CRUSH算法引擎，聪明地解决了数

据分布效率问题，奠定了它胜任各种规模存储池集群的坚实基础。过去5年，在Red Hat、Intel等软硬件基础设施领导者的推动下，Ceph开源社区有超过10倍的增长——不仅仅具备广泛的硬件兼容性体系，大量上下游厂商添砖加瓦，也吸引了很多运营商、企业用户参与改进。XSKY很荣幸作为社区的一员，见证与实践着Ceph帮助用户进行存储基础架构革新的历程。我们欣喜地看到由Ceph中国社区撰写的本书的问世，这是一部在立意和实践方面均不输于同期几本英文书籍的作品，深入浅出，娓娓道来，凝结了作者的热情和心血。我们诚挚地向业内技术同行和Ceph潜在用户推荐此书！愿Ceph中国社区在推进开源事业的道路上取得更大的成功！

——胥昕，XSKY星辰天合（北京）数据科技有限公司CEO

在开源软件定义存储（SDS）领域，Ceph是当之无愧的王者项目。随着IaaS技术的火热发展，越来越多的用户开始在生产环境中部署SDS。伴随着基础设施开源化的趋势，很多用户希望部署开源的SDS，Ceph成为了他们的第一选择。跟大多数开源软件项目一样，Ceph具有优秀的技术特性，但也存在着部署难、运维难的问题。在使用开源Ceph发行版时，用户需要对Ceph的实现原理、部署运维最佳实践有一定了解，才能在生产环境中稳定使用这一开源技术。长期以来，中文技术社区一直没有一本对Ceph的原理、生产实践、运维实践进行剖析的好书，本书的出现填补了这一空白。该书不仅从原理上对Ceph的核心技术进行了讲解，还介绍了将Ceph部署在OpenStack、ZStack等IaaS软件上的生产环境实践，最后着重介绍了Ceph的运维和排错，是一本不可多得的Ceph百科全书，是Ceph用户、IaaS开发人员必备的一本SDS工具书。

——张鑫，前CloudStack核心初创人员，开源IaaS项目ZStack创始人

开源系统是Linux的世界，开源管理平台是OpenStack的世界，开源存储是Ceph的世界。软件定义存储（SDS）是存储发展的必然趋势，最好的开源软件定义存储方案无疑就是Ceph，我身边好多朋友已经开始在生产环境中大量部署Ceph，Ceph也表现出卓越的稳定性和性能。但是Ceph的搭建和使用门槛比较高，很高兴看到Ceph中国社区组织编写的本书的出版，为Ceph搭建学习降低了门槛，是国内Ceph爱好者的福音。Ceph中国社区为Ceph在中国的普及做了大量非常重要的工作，本书是一个里程碑，相信Ceph中国社区会继续为Ceph做出更多的贡献。

——肖力，KVM云技术社区创始人

Ceph作为分布式存储开源项目的杰出代表，在各个领域已经得到了充分验证，可以预见，在未来的几年时间内，Ceph一定会得到更广泛的应用。本书作为国内为数不多阐述Ceph的著作，从基础、原理和实践多个层面进行了详尽讲解，是一本快速了解并掌握Ceph的力作。

——孙琦（Ray），北京休伦科技有限公司CTO

软件看开源，SDS看Ceph。Ceph是目前影响力最大的开源软件定义存储解决方案，其应用范围涵盖块存储、文件存储和对象存储，广泛被业界公司所采用。

很荣幸能在第一时间读到这本书，该书从Ceph的部署开始，阐明了Ceph各个主要模块及其功能，介绍了Ceph在块存储、文件存储和对象存储不同场景下的应用方式，指明了Ceph性能调优的方案。尤其是最后的生产环境应用案例，解了使用Ceph的技术人员的燃

眉之急，给出了常见问题的解决思路，造福于整个开源云存储界。

无论是售前专家、开发架构师还是运维负责人，读一读Ceph中国社区编写的这本书，都可以细细地品一品，积极地拥抱开源、把握云存储的未来。

——楼炜，盘古数据资深云和大数据架构师

作为一名早期研究Ceph的人员，很高兴看到Ceph在近几年如火如荼的发展状态。在我刚接触Ceph时，很渴望得到系统化的介绍、培训或指导。但当时Ceph在国内还处于小众研究状态，高人难寻，深入全面的介绍资料更是没有。Ceph中国社区的朋友们出版这本介绍Ceph的书籍，为Ceph的广大研究者和爱好者做了一件很有意义的事情。相信本书一定能够成为Ceph发展的强力助推器！

——温涛，新华三集团（H3C公司）ONEStor产品研发负责人

Ceph因其先进的设计思想，良好的可靠性、可扩展性，成为存储领域的研究热点，被誉为“存储的未来”，得到广泛的部署。由Ceph中国社区组织编写的这本书是国内第一本系统介绍Ceph的书籍，全书从Ceph的历史、架构、原理到部署、运维、应用案例，讲解全面深入，可操作性强。本书非常适合想要了解Ceph、使用Ceph的读者阅读，也可供分布式存储系统设计者参考。

——汪黎，KylinCloud团队存储技术负责人，Ceph代码贡献者

从实用价值上看，本书从Ceph的基本原理、Ceph的安装部署和Ceph的应用案例等方面进行了深入浅出

的讲解，理论和实践完美结合，是难得的系统阐述Ceph的教科书，是广大Ceph爱好者的福音。

从理论价值上看，Ceph是超融合架构下首选的开源存储方案，本书详细阐述了存储相关的基本原理，不仅让你知其然，更能让你知其所以然。

——刘军卫，中国移动苏州研发中心云计算产品部技术总监

Ceph是当前最热门的分布式存储系统，在云技术领域获得了广泛的欢迎和支持。但是目前国内与此相关的书籍非常少。如果想学习Ceph，想更深入地了解Ceph，而又对密密麻麻的英文望而生畏，那么现在救星来了！本书从系统原理、基本架构、性能优化、应用实践、运维部署等各个方面对Ceph进行了全方位的介绍和分析。这是一本从入门到精通的好书，值得拥有！

——李响，博士，中兴通讯股份有限公司IaaS开源项目总监

一群开源的人用开源的方式去做一件开源的事儿，我想没有比这更合适的事情了。作为一名有着近10年的分布式存储研发和软件定义存储（SDS）产品及技术规划的先行者与践行者，很高兴看到同样已经十几岁的Ceph在众人之力和众人之智的推动下，吐故纳新，正以日新月异的速度蓬勃发展。

Ceph是每一个软件定义存储相关从业人员关注的重点，Ceph中国社区把国内广大的Ceph爱好者聚集到一起，分享“踩坑”的经验，承担了95%以上Ceph文档的本土化（翻译）工作，对Ceph在国内的发展扮演着非常重要的推动作用，非常感谢Ceph中国社区的每一位贡献者。

杉岩数据作为一家商用Ceph解决方案和服务提供商，随着Ceph商用产品越来越多地在企业级用户的生产环境中应用，一直期待能有一本适合国人阅读习惯且浅显易懂的Ceph书籍，让更多的人了解Ceph的功能特性。当我有幸阅读过此书后，我强烈建议广大SDS相关从业人员阅读此书，你一定会收获良多！

——陈坚，深圳市杉岩数据技术有限公司总经理

Ceph从2012年开始拥抱OpenStack到现在已经成为OpenStack的首选后端存储。很高兴看到Ceph中国社区把国内广大的Ceph爱好者聚集到一起，分享技术与经验，对Ceph在国内的发展起到了非常重要的推动和落地作用。也一直期待国内能有一本Ceph入门相关的书籍，看到Ceph中国社区出版的这本书，很是欣慰。国内Ceph资料从稀缺到逐渐完善，这其中离不开Ceph中国社区的贡献和努力。

——朱荣泽，上海优铭云计算有限公司存储架构师

国内第一本对Ceph进行全面剖析的书籍，并辅以大量的实战操作，内容由浅入深，特别适合希望对Ceph进行系统性学习的工程师，是国内Ceph爱好者的福音。

——田亮，北京海云捷迅科技有限公司解决方案总监

Ceph是开源分布式存储领域的一颗当红明星，随着OpenStack如火如荼的发展，Ceph也逐渐成为了OpenStack的首选后端存储。国内目前缺乏Ceph入门以及相关运维书籍，Ceph中国社区出版的这本书填补了国内Ceph的空白，是国内Ceph爱好者的福音。

——陈沙克，浙江九州云信息科技有限公司副总裁

由于其出色的系统设计，Ceph正广泛部署于各大云计算厂商的生产环境中，为用户提供对象存储、云

硬盘和文件系统存储服务。本书理论联系实际，除介绍Ceph的设计理念和原理之外，还系统介绍了Ceph的编程接口、上线部署、性能调优及应用场景，有利于读者快速掌握Ceph的运维和基于Ceph的开发。此书提供了深入理解云存储的捷径。

——吴兴义，乐视云技术经理

Ceph诞生于传统存储行业正处于巅峰之时，短短十多年间，闪存（如SSD）与软件定义存储（SDS）就联手颠覆了存储行业。作为软件定义存储领域的旗帜性项目，Ceph肩负着业界的厚望，也需要“与时俱进”，继续改进和完善，满足目标用户越来越高的要求。

众所周知，Ceph是个开源项目，成型于硬盘仍为主导的年代。如今，市场和用户需要Ceph更加产品化，同时充分利用闪存等固态存储介质带来的性能红

利。这就要求业界精简过时的代码和不必要的中间层，并为Ceph加入新的功能和特性，对此，我个人归纳为“先做减法，再做加法”。要达到上述目标，必须让更多的人关注和了解Ceph，特别是吸引有一定存储经验和积累的人或组织加入Ceph生态圈。作为一本不可多得的系统介绍Ceph的书籍，本书的出版正逢其时，定会为Ceph生态的壮大贡献更多的有生力量。

——张广彬，北京企事录技术服务公司创始人

序

Ceph是目前开源世界在存储领域的里程碑式项目，它所带来的分布式、无中心化设计是目前众多商用分布式存储模仿和学习的对象。Ceph社区经过十多年发展已经成为近几年参与度增长最快的开源社区之一，而Ceph中国社区正是背后的驱动力之一。从2015年开始，Ceph中国社区一直努力在国内普及Ceph的生态，并为广大Ceph爱好者提供了交流平台，使得众多开源爱好者能够进一步了解Ceph的魅力。在过去的10年，开源世界慢慢成为了IT创新的动力，而这10年也是国内技术爱好者受益于开源的最好时间。但是，从开源爱好者到社区的深度参与方面，尤其是在世界级开源项目上，我们还存在大缺失，而这些“沟壑”需要像Ceph中国社区这样的组织来弥补。我很欣喜地看

到Ceph中国社区能在最合适的时间成立并迅速成长，而且受到Ceph官方社区的认可。

Ceph中国社区从论坛的搭建，微信群的建立，公众号的众包翻译和文章分析，到活动的组织都体现了一个开源社区最富有活力的价值。本书正是Ceph中国社区给国内Ceph爱好者的一份正当其时的“礼物”，本书是多位Ceph实战者在Ceph集群运维和问题讨论中形成的经验和锦囊之集合。毫不夸张地说，本书是我目前看到的最棒的Ceph入门工具书，可以帮助对分布式存储或者Ceph不太熟悉的读者真正零距离地接触并使用它。

王豪迈

2016年9月8日

前言

随着信息化浪潮的到来，全球各行各业逐步借助信息技术深入发展。据悉，企业及互联网数据以每年50%的速率在增长。据权威调查机构Gartner预测，到2020年，全球数据量将达到35ZB，相当于80亿块4TB硬盘，数据结构的变化给存储系统带来了全新的挑战。那么有什么方法能够存储这些数据呢？我认为Ceph是解决未来十年数据存储需求的一个可行方案。Ceph是存储的未来！SDS是存储的未来！

为什么写这本书

目前，磁盘具备容量优势，固态硬盘具备速度优势。但能否让容量和性能不局限在一个存储器单元呢？我们很快联想到磁盘阵列技术（Redundant Array

of Independent Disk，RAID，不限于HDD）。磁盘阵列技术是一种把多块独立的硬盘按不同的方式组合起来形成一个硬盘组（Disk Group，又称Virtual Disk），从而提供比单个硬盘更高的存储性能与数据备份能力的技术。磁盘阵列技术既可提供多块硬盘读写的聚合能力，又能提供硬盘故障的容错能力。

镜像技术（Mirroring）又称为复制技术（Replication），可提供数据冗余性和高可用性；条带（Striping），可提供并行的数据吞吐能力；纠删码（Erasure Code），把数据切片并增加冗余编码而提供高可用性和高速读写能力。镜像、条带和纠删码是磁盘阵列技术经典的数据分发方式，这3种经典的磁盘技术可通过组合方式提供更加丰富的数据读写性能。

传统的磁盘阵列技术的关注点在于数据在磁盘上的分发方式，随着通用磁盘、通用服务器，以及高速网络的成本降低，使数据在磁盘上的分发扩展到在服务器节点上的分发成为可能。镜像技术、条带技术和纠删码技术基于服务器节点的粒度实现后，这些技术的特点不再局限于单个设备的性能，而是具备“横向扩展”能力。我们暂且认为这是分布式存储本质的体现。

分布式存储解决了数据体量问题，对应用程序提供标准统一的访问接入，既能提升数据安全性和可靠性，又能提高存储整体容量和性能。可以预见，分布式存储是大规模存储的一个实现方向。分布式存储广泛地应用于航天、航空、石油、科研、政务、医疗、视频等高性能计算、云计算和大数据处理领域。目前行业应用对分布式存储技术需求旺盛，其处于快速发展阶段。

Ceph是加州大学圣克鲁兹分校的Sage Weil博士论文的研究项目，是一个使用自由开源协议（LGPLv2.1）的分布式存储系统。目前Ceph已经成为整个开源存储行业最热门的软件定义存储技术（Software Defined Storage，SDS）。它为块存储、文件存储和对象存储提供了统一的软件定义解决方案。Ceph旨在提供一个扩展性强大、性能优越且无单点故障的分布式存储系统。从一开始，Ceph就被设计为能在通用商业硬件上高度扩展。

由于其开放性、可扩展性和可靠性，Ceph成为了存储行业中的翘楚。这是云计算和软件定义基础设施的时代，我们需要一个完全软件定义的存储，更重要的是它要为云做好准备。无论运行的是公有云、私有云还是混合云，Ceph都非常合适。国内外有不少的Ceph应用方案，例如美国雅虎公司使用Ceph构建对象存储系统，用于Flickr、雅虎邮箱和Tumblr（轻量博

客）的后端存储；国内不少公有云和私有云商选择Ceph作为云主机后端存储解决方案。

如今的软件系统已经非常智能，可以最大限度地利用商业硬件来运行规模庞大的基础设施。Ceph就是其中之一，它明智地采用商业硬件来提供企业级稳固可靠的存储系统。

Ceph已被不断完善，并融入以下建设性理念。

- 每个组件能够线性扩展。

- 无任何单故障点。

- 解决方案必须是基于软件的、开源的、适应性强的。

- 运行于现有商业硬件之上。

• 每个组件必须尽可能拥有自我管理和自我修复能力。

对象是Ceph的基础，它也是Ceph的构建部件，并且Ceph的对象存储很好地满足了当下及将来非结构化数据的存储需求。相比传统存储解决方案，对象储存有其独特优势：我们可以使用对象存储实现平台和硬件独立。Ceph谨慎地使用对象，通过在集群内复制对象来实现可用性；在Ceph中，对象是不依赖于物理路径的，这使其独立于物理位置。这种灵活性使Ceph能实现从PB（petabyte）级到EB（exabyte）级的线性扩展。

Ceph性能强大，具有超强扩展性及灵活性。它可以帮助用户摆脱昂贵的专有存储孤岛。Ceph是真正在商业硬件上运行的企业级存储解决方案；是一种低成本但功能丰富的存储系统。Ceph通用存储系统同时提

供块存储、文件存储和对象存储，使客户可以按需使用。

由于国内许多企业决策者逐渐认识到Ceph的优势与前景，越来越多来自系统管理和传统存储的工程师使用Ceph，并有相当数量的企业基于Ceph研发分布式存储产品，为了更好地促进Ceph在国内传播和技术交流，我们几个爱好者成立了Ceph中国社区。目前，通过网络交流群、消息内容推送和问答互动社区，向国内关注Ceph技术的同行提供信息交流和共享平台。但是，由于信息在传递过程中过于分散，偶尔编写的文档内容并不完整，导致初学者在学习和使用Ceph的过程中遇到不少疑惑。同时，由于官方文档是通过英文发布的，对英语不太熟悉的同行难于学习。鉴于此，Ceph中国社区组织技术爱好者编写本书，本书主要提供初级和中级层面的指导。根据调查反馈以及社区成员的意见，我们确定了本书内容。

本书特色

在本书中，我们将采用穿插方式讲述Ceph分布式存储的原理与实战。本书侧重实战，循序渐进地讲述Ceph的基础知识和实战操作。从第1章起，读者会了解Ceph的前生今世。随着每章推进，读者将不断学习、不断深入。我希望，到本书的结尾，读者不论在概念上还是实战上，都能够成功驾驭Ceph。每个章节在讲述完基础理论知识后会有对应的实战操作。我们建议读者在自己的电脑上按部就班地进行实战操作。这样，一来读者不会对基础理论知识感到困惑，二来可让读者通过实战操作加深对Ceph的理解。同时，如果读者在阅读过程中遇到困难，我们建议再重温已阅章节或重做实验操作，这样将会加深理解，也可以加入Ceph中国社区QQ群（239404559）进行技术讨论。

读者对象

本书适用于以下读者。

- Ceph爱好者。
- 云平台运维工程师。
- 存储系统工程师。
- 系统管理员。
- 高等院校的学生或者教师。

本书是专门对上述读者所打造的Ceph入门级实战书籍。如果你具备GNU/Linux和存储系统的基本知识，却缺乏软件定义存储解决方案及Ceph相关的经验，本书也是不错的选择。云平台运维工程师、存储系统工程师读完本书之后能够深入了解Ceph原理、部署和维

护好线上Ceph集群。同时，本书也适合大学高年级本科生和研究生作为Ceph分布式存储系统或者云计算相关课程的参考书籍，能够带领你进入一个开源的分布式存储领域，深入地了解Ceph，有助于你今后的工作。

如何阅读本书

由于Ceph是运行在GNU/Linux系统上的存储解决方案，我们假定读者掌握了存储相关知识并熟悉GNU/Linux操作系统。如果读者在这些方面知识有欠缺，可参照阅读其他书籍或专业信息网站。

本书将讲述如下的内容。

第1章　描述Ceph的起源、主要功能、核心组件逻辑、整体架构和设计思想，并通过实战的方式指导

我们快速建立Ceph运行环境。

第2章　描述Ceph的分布式本质，深入分析Ceph架构，并介绍如何使用LIBRADOS库。

第3章　描述CRUSH的本质、基本原理，以及CRUSH作用下数据与对象的映射关系。

第4章　描述Ceph FS文件系统、RBD块存储和Object对象存储的建立以及使用。

第5章　描述Calamari的安装过程和基本使用操作。

第6章　描述Ceph FS作为高性能计算和大数据计算的后端存储的内容。

第7章　描述RBD在虚拟化和数据库场景下的应用，包括OpenStack、CloudStack和ZStack与RBD的结

合。

第8章　描述基于Ceph的云盘技术方案和备份方案，描述网关的异地同步方案和多媒体转换网关设计。

第9章　描述Ceph的硬件选型、性能调优，以及性能测试方法。

第10章　描述CRUSH的结构，并给出SSD与SATA混合场景下的磁盘组织方案。

第11章　描述Ceph的缓冲池原理和部署，以及纠删码原理和纠删码库，最后描述纠删码池的部署方案。

第12章　对3种存储访问类型的生产环境案例进行分析。

第13章　描述Ceph日常运维细节，以及常见错误的处理方法。

勘误与支持

在本书的写作过程，我们也参考了Ceph中国社区往期沙龙一线工程师、专家分享的经验和Ceph官方文档。我们热切希望能够为读者呈现丰富而且权威的Ceph存储技术。由于Ceph社区不断发展，版本迭代速度快，笔者水平有限，书中难免存在技术延后和谬误，恳请读者批评指正。可将任何意见和建议发送到邮箱devin@ceph.org.cn或者star.guo@ceph.org.cn，也可以发布到Ceph中国社区问答系统 http://bbs.ceph.org.cn/ 。我们将密切跟踪Ceph分布式存储技术的发展，吸收读者宝贵意见，适时编写

本书的升级版本。Ceph中国社区订阅号为：“ceph_community”，二维码为：

欢迎读者扫描关注，“Ceph中国社区订阅号”会定期发送Ceph技术文章、新闻资讯。也欢迎读者通过这个微信订阅号进行本书勘误反馈，本书的勘误和更新也会通过订阅号发布。

致谢

首先要感谢我们社区的全体志愿者，社区的发展离不开全体志愿者们无怨无悔的奉献，正是有了你们才有了社区今日的繁荣，其次要感谢所有支持过我们的企业，是你们的慷慨解囊成就了Ceph中国社区今日的壮大，最后感谢陈晓熹的校稿以及所有为本书编写提供支持、帮助的人。未来，我们也非常欢迎有志将开源事业发扬光大的同学们积极加入我们的社区，和我们一起创造Ceph未来的辉煌。

第1章　初识Ceph

1.1　Ceph概述

1.Ceph简介

从2004年提交第一行代码开始到现在，Ceph已经是一个有着十年之久的分布式存储系统软件，目前Ceph已经发展为开源存储界的当红明星，当然这与它的设计思想以及OpenStack的推动有关。

“Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability。”这句话说出了Ceph的特性，它是可靠的、可扩展的、统一的、分布式的存储系统。Ceph可以同时提供对象存储RADOSGW（Reliable、Autonomic、Distributed、

Object Storage Gateway）、块存储RBD（Rados Block Device）、文件系统存储Ceph FS（Ceph Filesystem）3种功能，以此来满足不同的应用需求。

Ceph消除了对系统单一中心节点的依赖，从而实现了真正的无中心结构的设计思想，这也是其他分布式存储系统所不能比的。通过后续章节内容的介绍，你可以看到，Ceph几乎所有优秀特性的实现，都与其核心设计思想有关。

OpenStack是目前最为流行的开源云平台软件。Ceph的飞速发展离不开OpenStack的带动。目前而言，Ceph已经成为OpenStack的标配开源存储方案之一，其实际应用主要涉及块存储和对象存储，并且开始向文件系统领域扩展。这一部分的相关情况，在后续章节中也将进行介绍。

2.Ceph的发展

Ceph是加州大学Santa Cruz分校的Sage Weil（DreamHost的联合创始人）专为博士论文设计的新一代自由软件分布式文件系统。

2004年，Ceph项目开始，提交了第一行代码。

2006年，OSDI学术会议上，Sage发表了介绍Ceph的论文，并在该篇论文的末尾提供了Ceph项目的下载链接。

2010年，Linus Torvalds将Ceph Client合并到内核2.6.34中，使Linux与Ceph磨合度更高。

2012年，拥抱OpenStack，进入Cinder项目，成为重要的存储驱动。

2014年，Ceph正赶上OpenStack大热，受到各大厂商的“待见”，吸引来自不同厂商越来越多的开发者

加入，Intel、SanDisk等公司都参与其中，同时Inktank公司被Red Hat公司1.75亿美元收购。

2015年，Red Hat宣布成立Ceph顾问委员会，成员包括Canonical、CERN、Cisco、Fujitsu、Intel、SanDisk和SUSE。Ceph顾问委员会将负责Ceph软件定义存储项目的广泛议题，目标是使Ceph成为云存储系统。

2016年，OpenStack社区调查报告公布，Ceph仍为存储首选，这已经是Ceph第5次位居调查的首位了。

3.Ceph应用场景

Ceph可以提供对象存储、块设备存储和文件系统服务，其对象存储可以对接网盘（owncloud）应用业务等；其块设备存储可以对接（IaaS），当前主流的IaaS云平台软件，例如OpenStack、CloudStack、

Zstack、Eucalyptus等以及KVM等，本书后续章节中将介绍OpenStack、CloudStack、Zstack和KVM的对接；其文件系统文件尚不成熟，官方不建议在生产环境下使用。

4.Ceph生态系统

Ceph作为开源项目，其遵循LGPL协议，使用C++语言开发，目前Ceph已经成为最广泛的全球开源软件定义存储项目，拥有得到众多IT厂商支持的协同开发模式。目前Ceph社区有超过40个公司的上百名开发者持续贡献代码，平均每星期的代码commits超过150个，每个版本通常在2000个commits左右，代码增减行数在10万行以上。在过去的几个版本发布中，贡献者的数量和参与公司明显增加，如图1-1所示。

图1-1　部分厂商和软件

5.Ceph用户群

Ceph成为了开源存储的当红明星，国内外已经拥有众多用户群体，下面简单说一下Ceph的用户群。

（1）国外用户群

1）CERN：CERN IT部门在2013年年中开始就运行了一个单一集群超过10000个VM和100000个CPU Cores的云平台，主要用来做物理数据分析。这个集群后端

Ceph包括3PB的原始容量，在云平台中作为1000多个Cinder卷和1500多个Glance镜像的存储池。在2015年开始测试单一30 PB的块存储RBD集群。

2）DreamHost：DreamHost从2012年开始运行基于Ceph RADOSGW的大规模对象存储集群，单一集群在3PB以下，大约由不到10机房集群组成，直接为客户提供对象存储服务。

3）Yahoo Flick：Yahoo Flick自2013年开始逐渐试用Ceph对象存储替换原有的商业存储，目前大约由10机房构成，每个机房在1PB~2PB，存储了大约2500亿个对象。

4）大学用户：奥地利的因斯布鲁克大学、法国的洛林大学等。

（2）国内用户群

1）以OpenStack为核心的云厂商：例如UnitedStack、Awcloud等国内云计算厂商。

2）Ceph产品厂商：SanDisk、XSKY、H3C、杉岩数据、SUSE和Bigtera等Ceph厂商。

3）互联网企业：腾讯、京东、新浪微博、乐视、完美世界、平安科技、联想、唯品会、福彩网和魅族等国内互联网企业。

6.社区项目开发迭代

目前Ceph社区采用每半年一个版本发布的方式来进行特性和功能的开发，每个版本发布需要经历设计、开发、新功能冻结，持续若干个版本的Bug修复周期后正式发布下一个稳定版本。其发布方式跟OpenStack差不多，也是每半年发布一个新版本。

Ceph会维护多个稳定版本来保证持续的Bug修复，以此来保证用户的存储安全，同时社区会有一个发布稳定版本的团队来维护已发布的版本，每个涉及之前版本的Bug都会被该团队移植回稳定版本，并且经过完整QA测试后发布下一个稳定版本。

代码提交都需要经过单元测试，模块维护者审核，并通过QA测试子集后才能合并到主线。社区维护一个较大规模的测试集群来保证代码质量，丰富的测试案例和错误注入机制保证了项目的稳定可靠。

7.Ceph版本

Ceph正处于持续开发中并且迅速提升。2012年7月3日，Sage发布了Ceph第一个LTS版本：Argonaut。从那时起，陆续又发布了9个新版本。Ceph版本被分为LTS（长期稳定版）以及开发版本，Ceph每隔一段时间就会发布一个长期稳定版。Ceph版本具体信息见表1-

1。欲了解更多信息，请访问

https://Ceph.com/category/releases/ 。

表1-1　Ceph版本信息

Ceph 版本名称	Ceph 版本号	发布时间
Argonaut	V0.48 (LTS)	2012.6.3
Bobtail	V0.56 (LTS)	2013.1.1
Cuttlefish	V0.61	2013.5.7
Dumpling	V0.67 (LTS)	2013.8.14
Emperor	V0.72	2013.11.9
Firefly	V0.80 (LTS)	2014.3.7
Giant	V0.87.1	2015.2.26
Hammer	V0.94 (LTS)	2015.4.7
Infernalis	V9.0.0	2015.5.5
Jewel	V10.0.0	2015.11
Jewel	V10.2.0	2016.3

1.2 Ceph的功能组件

Ceph提供了RADOS、OSD、MON、Librados、RBD、RGW和Ceph FS等功能组件，但其底层仍然使用RADOS存储来支撑上层的那些组件，如图1-2所示。

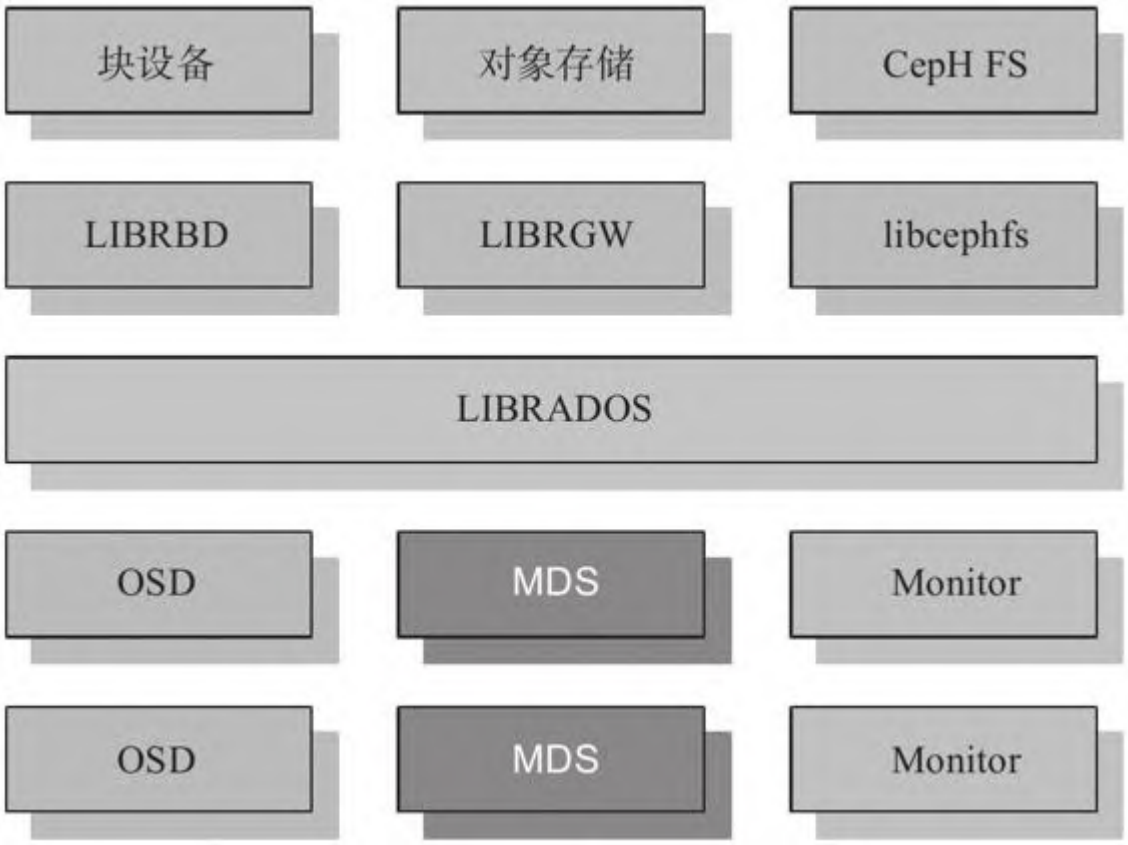

图1-2 Ceph功能组件的整体架构

下面分为两部分来讲述Ceph的功能组件。

（1）Ceph核心组件

在Ceph存储中，包含了几个重要的核心组件，分别是Ceph OSD、Ceph Monitor和Ceph MDS。一个Ceph的存储集群至少需要一个Ceph Monitor和至少两个Ceph的OSD。运行Ceph文件系统的客户端时，Ceph的元数据服务器（MDS）是必不可少的。下面来详细介绍一下各个核心组件。

• Ceph OSD：全称是Object Storage Device，主要功能包括存储数据，处理数据的复制、恢复、回补、平衡数据分布，并将一些相关数据提供给Ceph Monitor，例如Ceph OSD心跳等。一个Ceph的存储集群，至少需要两个Ceph OSD来实现active+clean健康状态和有效的保存数据的双副本（默认情况下是双副

本，可以调整）。注意：每一个Disk、分区都可以成为一个OSD。

• Ceph Monitor：Ceph的监控器，主要功能是维护整个集群健康状态，提供一致性的决策，包含了Monitor map、OSD map、PG（Placement Group）map和CRUSH map。

• Ceph MDS：全称是Ceph Metadata Server，主要保存的是Ceph文件系统（File System）的元数据（metadata）。温馨提示：Ceph的块存储和Ceph的对象存储都不需要Ceph MDS。Ceph MDS为基于POSIX文件系统的用户提供了一些基础命令，例如ls、find等命令。

（2）Ceph功能特性

Ceph可以同时提供对象存储RADOSGW（Reliable、Autonomic、Distributed、Object Storage Gateway）、块存储RBD（Rados Block Device）、文件系统存储Ceph FS（Ceph File System）3种功能，由此产生了对应的实际场景，本节简单介绍如下。

RADOSGW功能特性基于LIBRADOS之上，提供当前流行的RESTful协议的网关，并且兼容S3和Swift接口，作为对象存储，可以对接网盘类应用以及HLS流媒体应用等。

RBD（Rados Block Device）功能特性也是基于LIBRADOS之上，通过LIBRBD创建一个块设备，通过QEMU/KVM附加到VM上，作为传统的块设备来用。目前OpenStack、CloudStack等都是采用这种方式来为VM提供块设备，同时也支持快照、COW（Copy On Write）等功能。

Ceph FS（Ceph File System）功能特性是基于RADOS来实现分布式的文件系统，引入了MDS（Metadata Server），主要为兼容POSIX文件系统提供元数据。一般都是当做文件系统来挂载。

后面章节会对这几种特性以及对应的实际场景做详细的介绍和分析。

1.3　Ceph架构和设计思想

1.Ceph架构

Ceph底层核心是RADOS。Ceph架构图如图1-3所示。

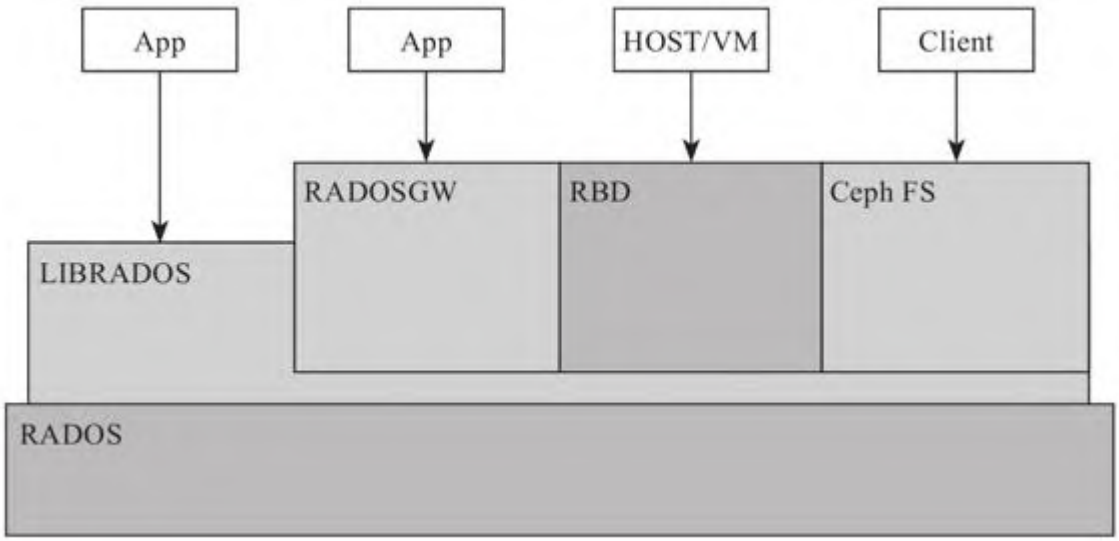

图1-3　Ceph架构图

• RADOS：RADOS具备自我修复等特性，提供了一个可靠、自动、智能的分布式存储。

• LIBRADOS：LIBRADOS库允许应用程序直接访问，支持C/C++、Java和Python等语言。

• RADOSGW：RADOSGW是一套基于当前流行的RESTful协议的网关，并且兼容S3和Swift。

• RBD：RBD通过Linux内核（Kernel）客户端和QEMU/KVM驱动，来提供一个完全分布式的块设备。

• Ceph FS：Ceph FS通过Linux内核（Kernel）客户端结合FUSE，来提供一个兼容POSIX的文件系统。

具体的RADOS细节以及RADOS的灵魂CRUSH（Controlled Replication Under Scalable Hashing，可扩展哈希算法的可控复制）算法，这两个知识点会在后面的第2、3章详细介绍和分析。

2.Ceph设计思想

Ceph是一个典型的起源于学术研究课题的开源项目。虽然学术研究生涯对于Sage而言只是其光辉事迹的短短一篇，但毕竟还是有几篇学术论文可供参考的。可以根据Sage的几篇论文分析Ceph的设计思想。

理解Ceph的设计思想，首先还是要了解Sage设计Ceph时所针对的应用场景，换句话说，Sage当初做Ceph的初衷的什么？

事实上，Ceph最初针对的应用场景，就是大规模的、分布式的存储系统。所谓“大规模”和“分布式”，至少是能够承载PB级别的数据和成千上万的存储节点组成的存储集群。

如今云计算、大数据在中国发展得如火如荼，PB容量单位早已经进入国内企业存储采购单，DT时代即将来临。Ceph项目起源于2004年，那是一个商用处理器以单核为主流，常见硬盘容量只有几十GB的年代。

当时SSD也没有大规模商用，正因如此，Ceph之前版本对SSD的支持不是很好，发挥不了SSD的性能。如今Ceph高性能面临的最大挑战正是这些历史原因，目前社区和业界正在逐步解决这些性能上的限制。

在Sage的思想中，我们首先说一下Ceph的技术特性，总体表现在集群可靠性、集群扩展性、数据安全性、接口统一性4个方面。

- 集群可靠性： 所谓“可靠性”，首先从用户角度来说数据是第一位的，要尽可能保证数据不会丢失。其次，就是数据写入过程中的可靠性，在用户将数据写入Ceph存储系统的过程中，不会因为意外情况出现而造成数据丢失。最后，就是降低不可控物理因素的可靠性，避免因为机器断电等不可控物理因素而产生的数据丢失。

• 集群可扩展性： 这里的“可扩展”概念是广义的，既包括系统规模和存储容量的可扩展，也包括随着系统节点数增加的聚合数据访问带宽的线性扩展。

• 数据安全性： 所谓“数据安全性”，首先要保证由于服务器死机或者是偶然停电等自然因素的产生，数据不会丢失，并且支持数据自动恢复，自动重平衡等。总体而言，这一特性既保证了系统的高度可靠和数据绝对安全，又保证了在系统规模扩大之后，其运维难度仍能保持在一个相对较低的水平。

• 接口统一性： 所谓“接口统一”，本书开头就说到了Ceph可以同时支持3种存储，即块存储、对象存储和文件存储。Ceph支持市面上所有流行的存储类型。

根据上述技术特性以及Sage的论文，我们来分析一下Ceph的设计思路，概述为两点：充分发挥存储本

身计算能力和去除所有的中心点。

• 充分发挥存储设备自身的计算能力： 其实就是采用廉价的设备和具有计算能力的设备（最简单的例子就是普通的服务器）作为存储系统的存储节点。Sage认为当前阶段只是将这些服务器当做功能简单的存储节点，从而产生资源过度浪费（如同虚拟化的思想一样，都是为了避免资源浪费）。而如果充分发挥节点上的计算能力，则可以实现前面提出的技术特性。这一点成为了Ceph系统设计的核心思想。

• 去除所有的中心点： 搞IT的最忌讳的就是单点故障，如果系统中出现中心点，一方面会引入单点故障，另一方面也必然面临着当系统规模扩大时的可扩展性和性能瓶颈。除此之外，如果中心点出现在数据访问的关键路径上，也必然导致数据访问的延迟增大。虽然在大多数存储软件实践中，单点故障点和性

能瓶颈的问题可以通过为中心点增加HA或备份加以缓解，但Ceph系统最终采用Crush、Hash环等方法更彻底地解决了这个问题。很显然Sage的眼光和设想还是很超前的。

1.4 Ceph快速安装

在Ceph官网上提供了两种安装方式：快速安装和手动安装。快速安装采用Ceph-Deploy工具来部署；手动安装采用官方教程一步一步来安装部署Ceph集群，过于烦琐但有助于加深印象，如同手动部署OpenStack一样。但是，建议新手和初学者采用第一种方式快速部署并且测试，下面会介绍如何使用Ceph-Deploy工具来快速部署Ceph集群。

1.4.1 Ubuntu/Debian安装

本节将介绍如何使用Ceph-Deploy工具来快速部署Ceph集群，开始之前先普及一下Ceph-Deploy工具的知识。Ceph-Deploy工具通过SSH方式连接到各节点服务器上，通过执行一系列脚本来完成Ceph集群部署。Ceph-Deploy简单易用同时也是Ceph官网推荐的默认安装工具。本节先来讲下在Ubuntu/Debian系统下如何快速安装Ceph集群。

1）配置Ceph APT源。

```
root@localhos`t:~# echo deb http://ceph.com/debian-{ceph-
stable-release}/ $(lsb_release -sc) main | sudo tee
/etc/apt/sources.list.d/ceph.list
```

2）添加APT源key。

```
root@localhost:~# wget -
```

```
q -
```

```
O -'https://ceph.com/git/?
p=ceph.git;a=blob_plain;f=keys/release.asc' | sudo apt-key
add -
```

3）更新源并且安装ceph-deploy。

```
root@localhost:~# sudo apt-get update &&sudo apt-get install
ceph-deploy -y
```

4）配置各个节点hosts文件。

```
root@localhost:~# cat /etc/hosts
192.168.1.2  node1
192.168.1.3  node2
192.168.1.4  node3
```

5）配置各节点SSH无密码登录，这就是本节开始时讲到的Ceph-Deploy工具要用过SSH方式连接到各节点服务器，来安装部署集群。输完ssh-keygen命令之后，在命令行会输出以下内容。

```
root@localhost:~# ssh-keygen
Generating public/private key pair.
Enter file in which to save the key (/ceph-
client/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ceph-
client/.ssh/id_rsa.
Your public key has been saved in /ceph-
client/.ssh/id_rsa.pub
```

6）复制key到各节点。

```
root@localhost:~# ssh-copy-idnode1
root@localhost:~# ssh-copy-idnode2
root@localhost:~# ssh-copy-idnode3
```

7）在执行ceph-deploy的过程中会生成一些配置文件，建议创建一个目录，例如my-cluster。

```
root@localhost:~# mkdir my-cluster
root@localhost:~# cd my-cluster
```

8）创建集群（Cluster），部署新的monitor节点。

```
root@localhost:~# ceph-deploy new {initial-monitor-node(s)}
```

例如：

```
root@localhost:~# ceph-deploy new node1
```

9）配置Ceph.conf配置文件，示例文件是默认的，可以根据自己情况进行相应调整和添加。具体优化情况本书后面会介绍。

```
[global]
fsid = 67d997c9-dc13-4edf-a35f-76fd693aa118
mon_initial_members = node1,
```

node2
mon_host = 192.168.1.2,

192.168.1.3
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
filestore_xattr_use_omap = true
<!-----以上部分都是

ceph-deploy默认生成的

---->
public network = {ip-address}/{netmask}
cluster network={ip-addesss}/{netmask}
<!-----以上两个网络是新增部分，默认只是添加

public network，一般生产都是定义两个网络，集群网络和数据网络分开

--------->
[osd]……

[mon]……

这里配置文件不再过多叙述。

10）安装Ceph到各节点。

```
root@localhost:~# ceph-deploy install {ceph-node}[{ceph-node} ...]
```

例如：

```
root@localhost:~# ceph-deploy install node1 node2 node3
```

11）获取密钥key，会在my-cluster目录下生成几个key。

```
root@localhost:~# ceph-deploy mon create-initial
```

12）初始化磁盘。

```
root@localhost:~# ceph-deploy disk zap {osd-server-name}:{disk-name}
```

例如：

```
root@localhost:~# ceph-deploy disk zap node1:sdb
```

13）准备OSD。

```
root@localhost:~# ceph-deploy osd prepare {node-name}:{data-disk}[:{journal-disk}]
```

例如：

```
root@localhost:~# ceph-deploy osd prepare node1:sdb1:sdc
```

14）激活OSD。

```
root@localhost:~# ceph-deploy osd activate {node-name}:
{data-disk-partition}[:{journal-disk-partition}]
```

例如：

```
root@localhost:~# ceph-deploy osd activate node1:sdb1:sdc
```

15）分发key。

```
root@localhost:~# ceph-deploy admin {admin-node} {ceph-node}
```

例如：

```
root@localhost:~# ceph-deploy admin node1 node2 node3
```

16）给admin key赋权限。

```
root@localhost:~# sudo chmod +r
/etc/ceph/ceph.client.admin.keyring
```

17）查看集群健康状态，如果是active+clean状态就是正常的。

```
root@localhost:~# ceph health
```

 提示

安装Ceph前提条件如下。

①时间要求很高，建议在部署Ceph集群的时候提前配置好NTP服务器。

②对网络要求一般，因为Ceph源在外国有时候会被屏蔽，解决办法多尝试机器或者代理。

1.4.2　RHEL/CentOS安装

本节主要讲一下在RHEL/CentOS系统下如何快速安装Ceph集群。

1）配置Ceph YUM源。

```
root@localhost:~# vim /etc/yum.repos.d/ceph.repo
[ceph-noarch]
name=Cephnoarch packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?
p=ceph.git;a=blob_plain;f=keys/release.asc
```

2）更新源并且安装ceph-deploy。

```
root@localhost:~# yum update &&yum install ceph-deploy -y
```

3）配置各个节点hosts文件。

```
root@localhost:~# cat /etc/hosts
192.168.1.2  node1
192.168.1.3  node2
192.168.1.4  node3
```

4）配置各节点SSH无密码登录，通过SSH方式连接到各节点服务器，以安装部署集群。输入ssh-keygen命令，在命令行会输出以下内容。

```
root@localhost:~# ssh-keygen
Generating public/private key pair.
Enter file in which to save the key (/ceph-
client/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ceph-
client/.ssh/id_rsa.
Your public key has been saved in /ceph-
client/.ssh/id_rsa.pub
```

5）拷贝key到各节点。

```
root@localhost:~# ssh-copy-id node1
root@localhost:~# ssh-copy-id node2
root@localhost:~# ssh-copy-id node3
```

6）在执行ceph-deploy的过程中会生成一些配置文件，建议创建一个目录，例如my-cluster。

```
root@localhost:~# mkdir my-cluster
root@localhost:~# cd my-cluster
```

7）创建集群（Cluster），部署新的monitor节点。

```
root@localhost:~# ceph-deploy new {initial-monitor-node(s)}
```

例如：

```
root@localhost:~# ceph-deploy new node1
```

8）配置Ceph.conf配置文件，示例文件是默认的，可以根据自己情况进行相应调整和添加。具体优化情况本书后面会介绍。

```
[global]
fsid = 67d997c9-dc13-4edf-a35f-76fd693aa118
mon_initial_members = node1,
```

node2
mon_host = 192.168.1.2,

192.168.1.3
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
filestore_xattr_use_omap = true
<!-----以上部分都是

ceph-deploy默认生成的

---->
public network = {ip-address}/{netmask}
cluster network={ip-addesss}/{netmask}
<!-----以上两个网络是新增部分，默认只是添加

public network，一般生产都是定义两个网络，集群网络和数据网络分开

--------->
[osd]……

[mon]……

这里配置文件不再过多叙述。

9）安装Ceph到各节点。

```
root@localhost:~# ceph-deploy install {ceph-node}[{ceph-node} ...]
```

例如：

```
root@localhost:~# ceph-deploy install node1 node2 node3
```

10）获取密钥key，会在my-cluster目录下生成几个key。

```
root@localhost:~# ceph-deploy mon create-initial
```

11）初始化磁盘。

```
root@localhost:~# ceph-deploy disk zap {osd-server-name}:
{disk-name}
```

例如：

```
root@localhost:~# ceph-deploy disk zap node1:sdb
```

12）准备OSD。

```
root@localhost:~# ceph-deploy osd prepare {node-name}:{data-disk}[:{journal-disk}]
```

例如：

```
root@localhost:~# ceph-deploy osd prepare node1:sdb1:sdc
```

13）激活OSD。

```
root@localhost:~# ceph-deploy osd activate {node-name}:{data-disk-partition}[:{journal-disk-partition}]
```

例如：

```
root@localhost:~# ceph-deploy osd activate node1:sdb1:sdc
```

14）分发key。

```
root@localhost:~# ceph-deploy admin {admin-node} {ceph-node}
```

例如：

```
root@localhost:~# ceph-deploy admin node1 node2 node3
```

15）给admin key赋权限。

```
root@localhost:~# sudo chmod +r
/etc/ceph/ceph.client.admin.keyring
```

16）查看集群健康状态，如果是active+clean状态就是正常的。

```
root@localhost:~# ceph health
```

1.5 本章小结

本章主要从Ceph的历史背景、发展事件、Ceph的架构组件、功能特性以及Ceph的设计思想方面介绍了Ceph，让大家对Ceph有一个全新的认识，以便后面更深入地了解Ceph。

第2章　存储基石RADOS

分布式对象存储系统RADOS是Ceph最为关键的技术，它是一个支持海量存储对象的分布式对象存储系统。RADOS层本身就是一个完整的对象存储系统，事实上，所有存储在Ceph系统中的用户数据最终都是由这一层来存储的。而Ceph的高可靠、高可扩展、高性能、高自动化等特性，本质上也是由这一层所提供的。因此，理解RADOS是理解Ceph的基础与关键。

Ceph的设计哲学如下。

- 每个组件必须可扩展。
- 不存在单点故障。
- 解决方案必须是基于软件的。

• 可摆脱专属硬件的束缚即可运行在常规硬件上。

• 推崇自我管理。

由第1章的讲解可以知道，Ceph包含以下组件。

• 分布式对象存储系统RADOS库，即LIBRADOS。

• 基于LIBRADOS实现的兼容Swift和S3的存储网关系统RADOSGW。

• 基于LIBRADOS实现的块设备驱动RBD。

• 兼容POSIX的分布式文件Ceph FS。

• 最底层的分布式对象存储系统RADOS。

2.1 Ceph功能模块与RADOS

Ceph中的这些组件与RADOS有什么关系呢，笔者手绘了一张简单的Ceph架构图，我们结合图2-1来分析这些组件与RADOS的关系。

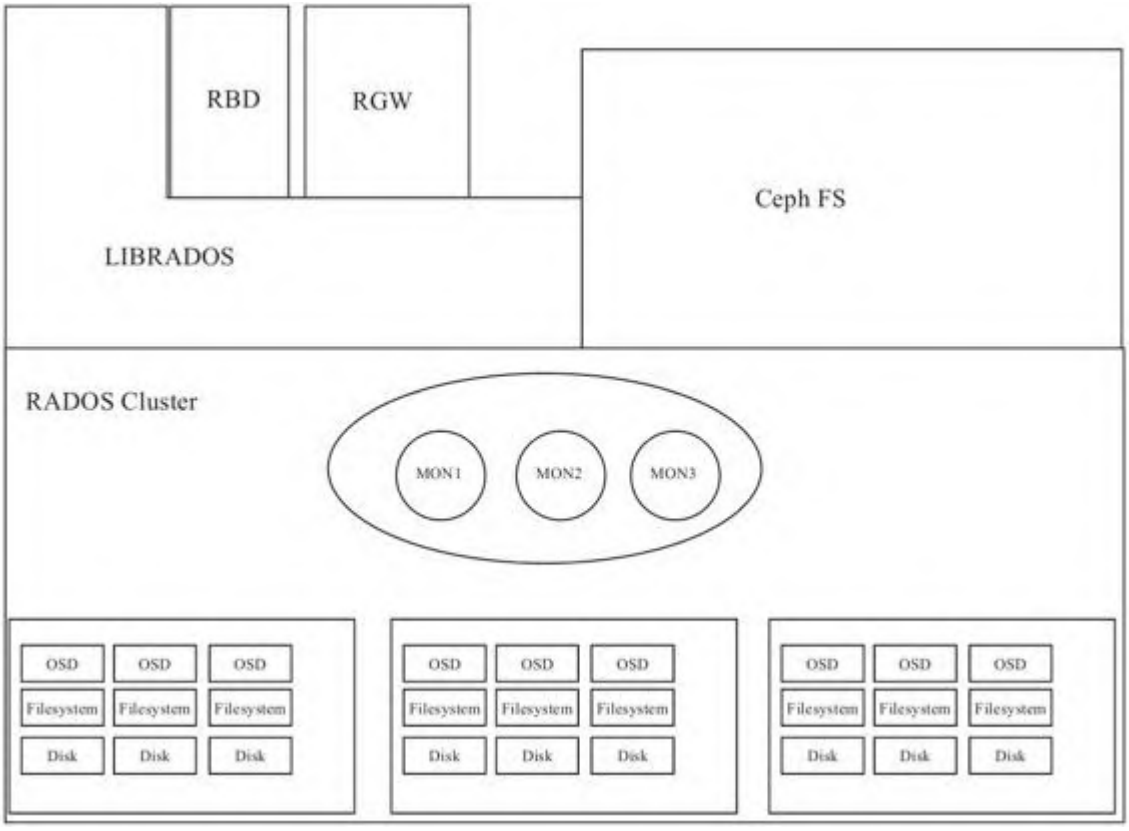

图2-1 Ceph架构图

Ceph存储系统的逻辑层次结构大致划分为4部分：基础存储系统RADOS、基于RADOS实现的Ceph FS，基于RADOS的LIBRADOS层应用接口、基于LIBRADOS的应用接口RBD、RADOSGW。Ceph架构（见图1-1）我们在第1章有过初步的了解，这里详细看一下各个模块的功能，以此了解RADOS在整个Ceph起到的作用。

（1）基础存储系统RADOS

RADOS这一层本身就是一个完整的对象存储系统，事实上，所有存储在Ceph系统中的用户数据最终都是由这一层来存储的。Ceph的很多优秀特性本质上也是借由这一层设计提供。理解RADOS是理解Ceph的基础与关键。物理上，RADOS由大量的存储设备节点组成，每个节点拥有自己的硬件资源（CPU、内存、硬盘、网络），并运行着操作系统和文件系统。本书后续章节将对RADOS进行深入介绍。

（2）基础库LIBRADOS

LIBRADOS层的功能是对RADOS进行抽象和封装，并向上层提供API，以便直接基于RADOS进行应用开发。需要指明的是，RADOS是一个对象存储系统，因此，LIBRADOS实现的API是针对对象存储功能的。RADOS采用C++开发，所提供的原生LIBRADOS API包括C和C++两种。物理上，LIBRADOS和基于其上开发的应用位于同一台机器，因而也被称为本地API。应用调用本机上的LIBRADOS API，再由后者通过socket与RADOS集群中的节点通信并完成各种操作。

（3）上层应用接口

Ceph上层应用接口涵盖了RADOSGW（RADOS Gateway）、RBD（Reliable Block Device）和Ceph FS（Ceph File System），其中，RADOSGW和RBD是在

LIBRADOS库的基础上提供抽象层次更高、更便于应用或客户端使用的上层接口。

其中，RADOSGW是一个提供与Amazon S3和Swift兼容的RESTful API的网关，以供相应的对象存储应用开发使用。RADOSGW提供的API抽象层次更高，但在类S3或Swift LIBRADOS的管理比便捷，因此，开发者应针对自己的需求选择使用。RBD则提供了一个标准的块设备接口，常用于在虚拟化的场景下为虚拟机创建volume。目前，Red Hat已经将RBD驱动集成在KVM/QEMU中，以提高虚拟机访问性能。

（4）应用层

应用层就是不同场景下对于Ceph各个应用接口的各种应用方式，例如基于LIBRADOS直接开发的对象存储应用，基于RADOSGW开发的对象存储应用，基于RBD实现的云主机硬盘等。

下面就来看看RADOS的架构。

2.2 RADOS架构

RADOS系统主要由两个部分组成，如图2-2所示。

1）OSD：由数目可变的大规模OSD（Object Storage Devices）组成的集群，负责存储所有的Objects数据。

2）Monitor：由少量Monitors组成的强耦合、小规模集群，负责管理Cluster Map。其中，Cluster Map是整个RADOS系统的关键数据结构，管理集群中的所有成员、关系和属性等信息以及数据的分发。

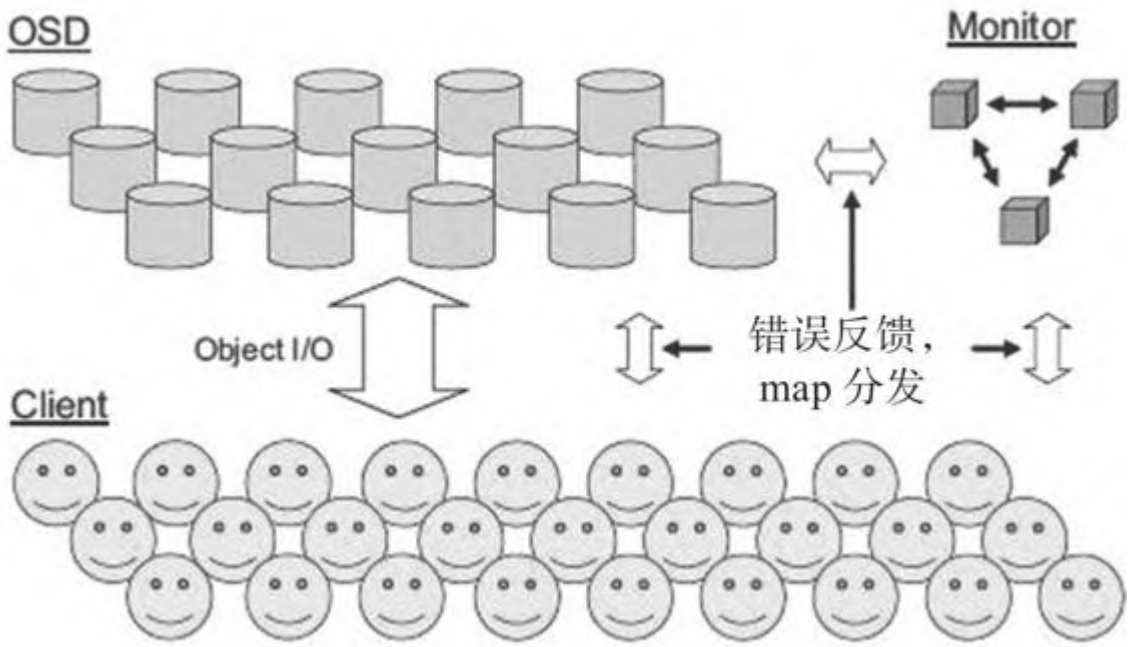

图2-2　RADOS系统架构图示

对于RADOS系统，节点组织管理和数据分发策略均由内部的Mon全权负责，因此，从Client角度设计相对比较简单，它给应用提供存储接口。

2.2.1 Monitor介绍

正如其名，Ceph Monitor是负责监视整个群集的运行状况的，这些信息都是由维护集群成员的守护程序来提供的，如各个节点之间的状态、集群配置信息。Ceph monitor map包括OSD Map、PG Map、MDS Map和CRUSH等，这些Map被统称为集群Map。

1）Monitor Map。Monitor Map包括有关monitor节点端到端的信息，其中包括Ceph集群ID，监控主机名和IP地址和端口号，它还存储了当前版本信息以及最新更改信息，可以通过以下命令查看monitor map。

```
#ceph mon dump
```

2）OSD Map。OSD Map包括一些常用的信息，如集群ID，创建OSD Map的版本信息和最后修改信息，以及

pool相关信息，pool的名字、pool的ID、类型，副本数目以及PGP，还包括OSD信息，如数量、状态、权重、最新的清洁间隔和OSD主机信息。可以通过执行以下命令查看集群的OSD Map。

```
#ceph  osd dump
```

3）PG Map。PG Map包括当前PG版本、时间戳、最新的OSD Map的版本信息、空间使用比例，以及接近占满比例信息，同时，也包括每个PG ID、对象数目、状态、OSD的状态以及深度清理的详细信息，可以通过以下命令来查看PG Map。

```
#ceph pg dump
```

4）CRUSH Map。CRUSH Map包括集群存储设备信息，故障域层次结构和存储数据时定义失败域规则信息；可以通过以下命令查看CRUSH Map。

```
```

```
#ceph osd crush dump
```

5）MDS Map。MDS Map包括存储当前MDS Map的版本信息、创建当前Map的信息、修改时间、数据和元数据POOL ID、集群MDS数目和MDS状态，可通过以下命令查看集群MDS Map信息。

```
#ceph mds dump
```

Ceph的MON服务利用Paxos的实例，把每个映射图存储为一个文件。Ceph Monitor并未为客户提供数据存储服务，而是为Ceph集群维护着各类Map，并服务更新群集映射到客户机以及其他集群节点。客户端和其他群集节点定期检查并更新于Monitor的集群Map最新的副本。

Ceph Monitor是个轻量级的守护进程，通常情况下并不需要大量的系统资源，低成本、入门级的CPU，

以及千兆网卡即可满足大多数的场景；与此同时，Monitor节点需要有足够的磁盘空间来存储集群日志，健康集群产生几MB到GB的日志；然而，如果存储的需求增加时，打开低等级的日志信息的话，可能需要几个GB的磁盘空间来存储日志。

一个典型的Ceph集群包含多个Monitor节点。一个多Monitor的Ceph的架构通过法定人数来选择leader，并在提供一致分布式决策时使用Paxos算法集群。在Ceph集群中有多个Monitor时，集群的Monitor应该是奇数；最起码的要求是一台监视器节点，这里推荐Monitor个数是3。由于Monitor工作在法定人数，一半以上的总监视器节点应该总是可用的，以应对死机等极端情况，这是Monitor节点为N（N>0）个且N为奇数的原因。所有集群Monitor节点，其中一个节点为Leader。如果Leader Monitor节点处于不可用状态，

其他显示器节点有资格成为Leader。生产群集必须至少有N/2个监控节点提供高可用性。

2.2.2 Ceph OSD简介

Ceph OSD是Ceph存储集群最重要的组件，Ceph OSD将数据以对象的形式存储到集群中每个节点的物理磁盘上，完成存储用户数据的工作绝大多数都是由OSD deamon进程来实现的。

Ceph集群一般情况都包含多个OSD，对于任何读写操作请求，Client端从Ceph Monitor获取Cluster Map之后，Client将直接与OSD进行I/O操作的交互，而不再需要Ceph Monitor干预。这使得数据读写过程更为迅速，因为这些操作过程不像其他存储系统，它没有其他额外的层级数据处理。

Ceph的核心功能特性包括高可靠、自动平衡、自动恢复和一致性。对于Ceph OSD而言，基于配置的副本数，Ceph提供通过分布在多节点上的副本来实现，

使得Ceph具有高可用性以及容错性。在OSD中的每个对象都有一个主副本，若干个从副本，这些副本默认情况下是分布在不同节点上的，这就是Ceph作为分布式存储系统的集中体现。每个OSD都可能作为某些对象的主OSD，与此同时，它也可能作为某些对象的从OSD，从OSD受到主OSD的控制，然而，从OSD在某些情况也可能成为主OSD。在磁盘故障时，Ceph OSD Deamon的智能对等机制将协同其他OSD执行恢复操作。在此期间，存储对象副本的从OSD将被提升为主OSD，与此同时，新的从副本将重新生成，这样就保证了Ceph的可靠和一致。

Ceph OSD架构实现由物理磁盘驱动器、在其之上的Linux文件系统以及Ceph OSD服务组成。对Ceph OSD Deamon而言，Linux文件系统显性地支持了其扩展属性；这些文件系统的扩展属性提供了关于对象状态、

快照、元数据内部信息；而访问Ceph OSD Deamon的ACL则有助于数据管理，如图2-3所示。

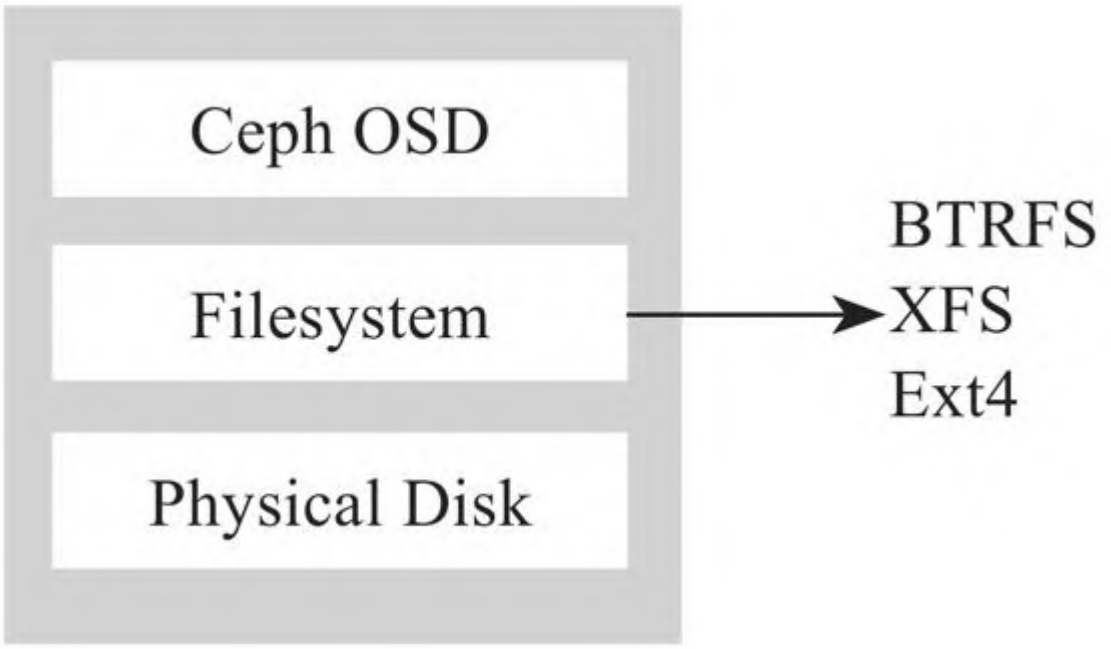

图2-3　OSD数据存储图

Ceph OSD操作必须在一个有效的Linux分区的物理磁盘驱动器上，Linux分区可以是BTRFS、XFS或者EXT4分区，文件系统是对性能基准测试的主要标准之一，下面来逐一了解。

1）BTRFS：在BTRFS文件系统的OSD相比于XFS和EXT4提供了最好的性能。BTRFS的主要优点有以下4点。

• 扩展性（scalability）：BTRFS最重要的设计目标是应对大型机器对文件系统的扩展性要求。Extent、B-Tree和动态inode创建等特性保证了BTRFS在大型机器上仍有卓越的表现，其整体性能不会随着系统容量的增加而降低。

• 数据一致性（data integrity）：当系统面临不可预料的硬件故障时，BTRFS采用COW事务技术来保证文件系统的一致性。BTRFS还支持校验和，避免了silent corrupt（未知错误）的出现。而传统文件系统无法做到这一点。

• 多设备管理相关的特性：BTRFS支持创建快照（snapshot）和克隆（clone）。BTRFS还能够方便地

管理多个物理设备，使得传统的卷管理软件变得多余。

• 结合Ceph，BTRFS中的诸多优点中的快照，Journal of Parallel（并行日志）等优势在Ceph中表现得尤为突出，不幸的是，BTRFS还未能到达生产环境要求的健壮要求。暂不推荐用于Ceph集群的生产使用。

2）XFS：一种高性能的日志文件系统，XFS特别擅长处理大文件，同时提供平滑的数据传输。目前CentOS 7也将XFS+LVM作为默认的文件系统。XFS的主要优点如下。

• 分配组：XFS文件系统内部被分为多个“分配组”，它们是文件系统中的等长线性存储区。每个分配组各自管理自己的inode和剩余空间。文件和文件夹可以跨越分配组。这一机制为XFS提供了可伸缩性和并

行特性——多个线程和进程可以同时在同一个文件系统上执行I/O操作。这种由分配组带来的内部分区机制在一个文件系统跨越多个物理设备时特别有用，使得优化对下级存储部件的吞吐量利用率成为可能。

• 条带化分配：在条带化RAID阵列上创建XFS文件系统时，可以指定一个“条带化数据单元”。这可以保证数据分配、inode分配，以及内部日志被对齐到该条带单元上，以此最大化吞吐量。

• 基于Extent的分配方式：XFS文件系统中的文件用到的块由变长Extent管理，每一个Extent描述了一个或多个连续的块。对那些把文件所有块都单独列出来的文件系统来说，Extent大幅缩短了列表。

有些文件系统用一个或多个面向块的位图管理空间分配——在XFS中，这种结构被由一对B+树组成的、面向Extent的结构替代了；每个文件系统分配组

（AG）包含这样的一个结构。其中，一个B+树用于索引未被使用的Extent的长度，另一个索引这些Extent的起始块。这种双索引策略使得文件系统在定位剩余空间中的Extent时十分高效。

• 扩展属性：XFS通过实现扩展文件属性给文件提供了多个数据流，使文件可以被附加多个名/值对。文件名是一个最大长度为256字节的、以NULL字符结尾的可打印字符串，其他的关联值则可包含多达64KB的二进制数据。这些数据被进一步分入两个名字空间中，分别为root和user。保存在root名字空间中的扩展属性只能被超级用户修改，保存在user名字空间中的可以被任何对该文件拥有写权限的用户修改。扩展属性可以被添加到任意一种XFS inode上，包括符号链接、设备节点和目录等。可以使用attr命令行程序操作这些扩展属性。xfsdump和xfsrestore工具在进行备份和

恢复时会一同操作扩展属性，而其他的大多数备份系统则会忽略扩展属性。

• XFS作为一款可靠、成熟的，并且非常稳定的文件系统，基于分配组、条带化分配、基于Extent的分配方式、扩展属性等优势非常契合Ceph OSD服务的需求。美中不足的是，XFS不能很好地处理Ceph写入过程的journal问题。

3）Ext4：第四代扩展文件系统，是Linux系统下的日志文件系统，是Ext3文件系统的后继版本。其主要特征如下。

• 大型文件系统：Ext4文件系统可支持最高1 Exbibyte的分区与最大16 Tebibyte的文件。

• Extents：Ext4引进了Extent文件存储方式，以替换Ext2/3使用的块映射（block mapping）方式。

Extent指的是一连串的连续实体块，这种方式可以增加大型文件的效率并减少分裂文件。

• 日志校验和：Ext4使用校验和特性来提高文件系统可靠性，因为日志是磁盘上被读取最频繁的部分之一。

• 快速文件系统检查：Ext4将未使用的区块标记在inode当中，这样可以使诸如e2fsck之类的工具在磁盘检查时将这些区块完全跳过，而节约大量的文件系统检查的时间。这个特性已经在2.6.24版本的Linux内核中实现。

Ceph OSD把底层文件系统的扩展属性用于表示各种形式的内部对象状态和元数据。XATTR是以key/value形式来存储xattr_name和xattr_value，并因此提供更多的标记对象元数据信息的方法。Ext4文件系统提供不足以满足XATTR，由于XATTR上存储的字

节数的限制能力，从而使Ext4文件系统不那么受欢迎。然而，BTRFS和XFS有一个比较大的限制XATTR。

Ceph使用日志文件系统，如增加了BTRFS和XFS的OSD。在提交数据到后备存储器之前，Ceph首先将数据写入称为一个单独的存储区，该区域被称为journal，这是缓冲器分区在相同或单独磁盘作为OSD，一个单独的SSD磁盘或分区，甚至一个文件文件系统。在这种机制下，Ceph任何写入首先是日志，然后是后备存储，如图2-4所示。

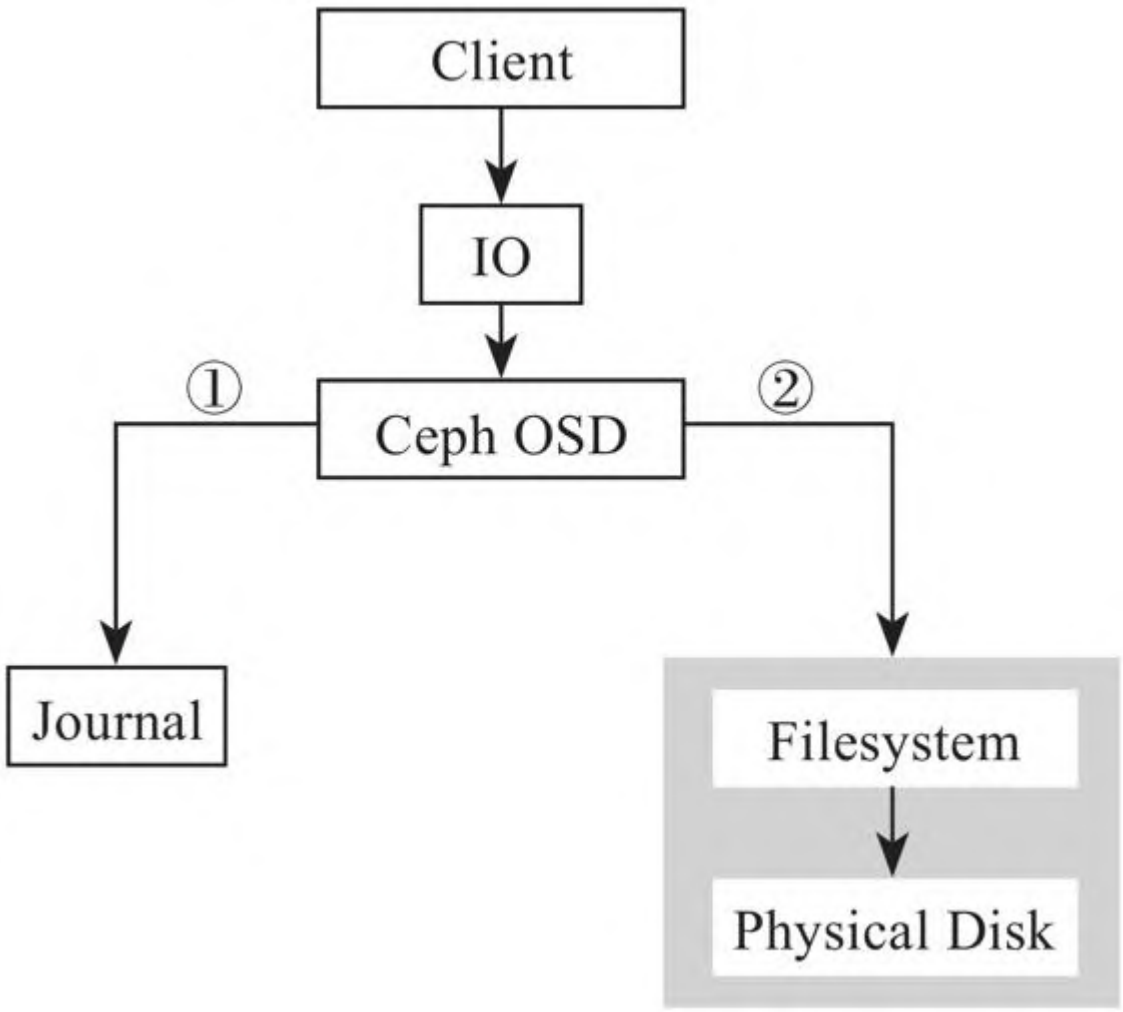

图2-4　IO流向图

journal持续到后备存储同步，每隔5s。默认情况下。10GB是该jouranl的常用的大小，但journal空间越大越好。Ceph使用journal综合考虑了存储速度和数据的一致性。journal允许Ceph OSD功能很快做小的写操作；一个随机写入首先写入在上一个连续类型的

journal，然后刷新到文件系统。这给了文件系统足够的时间来合并写入磁盘。使用SSD盘作为journal盘能获得相对较好的性能。在这种情况下，所有的客户端写操作都写入到超高速SSD日志，然后刷新到磁盘。所以，一般情况下，使用SSD作为OSD的journal可以有效缓冲突发负载。

与传统的分布式数据存储不同，RADOS最大的特点如下。

①将文件映射到Object后，利用Cluster Map通过CRUSH计算而不是查找表方式定位文件数据到存储设备中的位置。优化了传统的文件到块的映射和BlockMap管理。

②RADOS充分利用了OSD的智能特点，将部分任务授权给OSD，最大程度地实现可扩展。

2.3 RADOS与LIBRADOS

LIBRADOS模块是客户端用来访问RADOS对象存储设备的。Ceph存储集群提供了消息传递层协议，用于客户端与Ceph Monitor与OSD交互，LIBRADOS以库形式为Ceph Client提供了这个功能，LIBRADOS就是操作RADOS对象存储的接口。所有Ceph客户端可以用LIBRADOS或LIBRADOS里封装的相同功能和对象存储交互，LIBRBD和LIBCEPHFS就利用了此功能。你可以用LIBRADOS直接和Ceph交互（如与Ceph兼容的应用程序、Ceph接口等）。下面是简单描述的步骤。

第1步：获取LIBRADOS。

第2步：配置集群句柄。

第3步：创建IO上下文。

第4步：关闭连接。

LIBRADOS架构图，如图2-5所示。

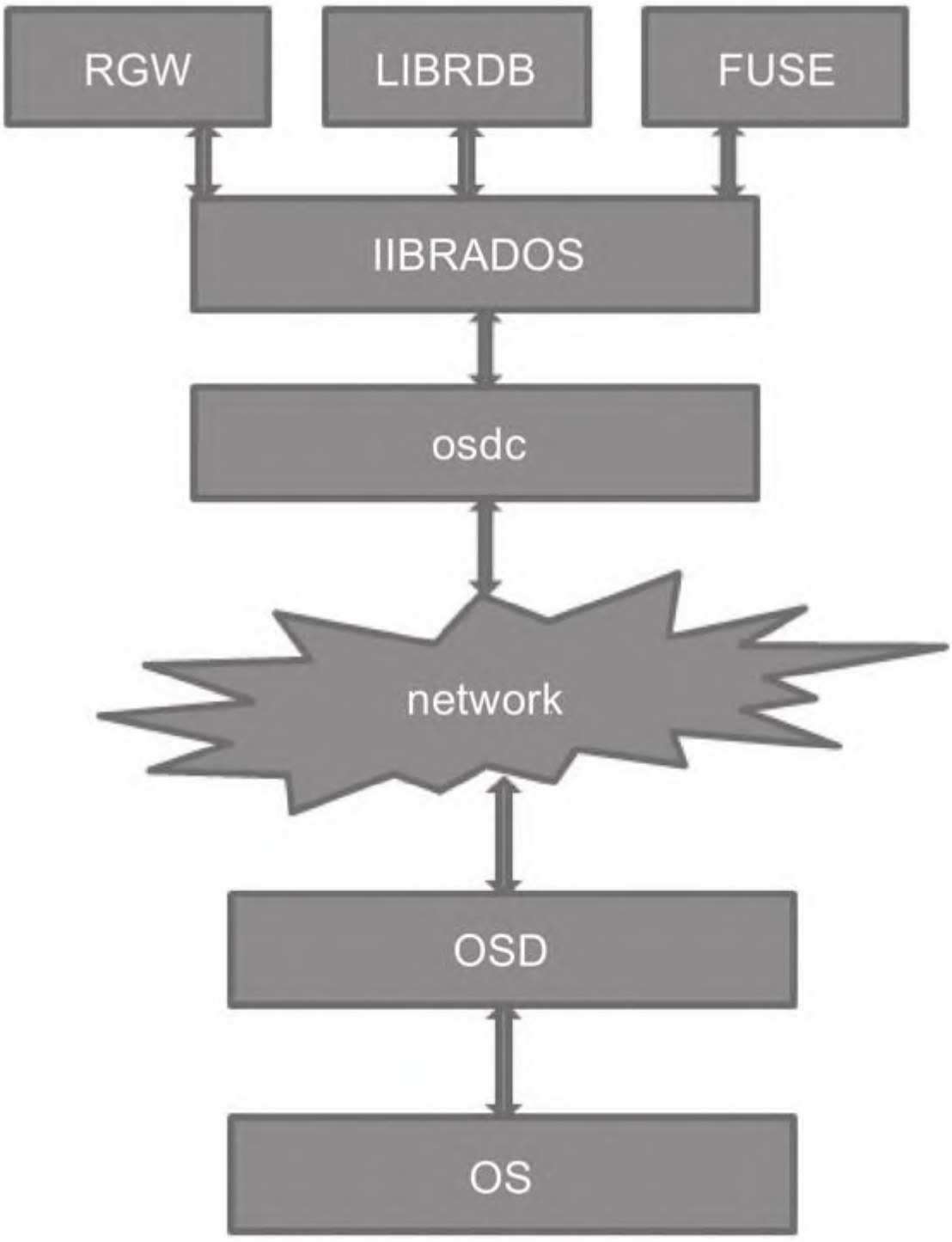

图2-5　LIBRADOS架构图

先根据配置文件调用librados创建一个rados，接下来为这个rados创建一个radosclient，radosclient包含3个主要模块（finisher、Messager、Objector）。再根据pool创建对应的ioctx，在ioctx中能够找到radosclient。再调用osdc对生成对应osd请求，与OSD进行通信响应请求。

下面分别介绍LIBRADOS的C语言、Java语言和Python语言示例。

1.LIBRADOS C语言示例

下面是LIBRADOS C语言示例。

```
#include <stdio.h>
#include <string.h>
#include <rados/librados.h>
int main (int argc, char argv**)
{
        /*声明集群句柄以及所需参数
```

```
 */
        rados_t cluster;
        char cluster_name[] = "ceph";          //集群名称

        char user_name[] = "client.admin";     //指定访问集群的用
户，这里用

admin
        uint64_t flags;
        rados_ioctx_t io;                      //rados上下文句柄

        char *poolname = "data";               //目标
```

```
pool名

        char read_res[100];
        char xattr[] = "en_US";
        /* 指定参数初始化句柄

*/
        int err;
        err = rados_create2(&cluster, cluster_name,
user_name, flags);
        if (err < 0) {
                fprintf(stderr, "%s: Couldn't create the
cluster handle! %s\n", argv[0], strerror(-err));
                exit(EXIT_FAILURE);
        } else {
                printf("\nCreated a cluster handle.\n");
        }
        /* 读取配置文件用来配置句柄

*/
        err = rados_conf_read_file(cluster,
"/etc/ceph/ceph.conf");
        if (err < 0) {
```

```
                fprintf(stderr, "%s: cannot read config
file: %s\n", argv[0], strerror(-err));
                exit(EXIT_FAILURE);
        } else {
                printf("\nRead the config file.\n");
        }
        /* 分解参数




 */
        err = rados_conf_parse_argv(cluster, argc, argv);
        if (err < 0) {
                fprintf(stderr, "%s: cannot parse command
line arguments: %s\n", argv[0], strerror(-err));
                exit(EXIT_FAILURE);
        } else {
                printf("\nRead the command line
arguments.\n");
        }
        /* 连接集群




*/
        err = rados_connect(cluster);
        if (err < 0) {
                fprintf(stderr, "%s: cannot connect to
cluster: %s\n", argv[0], strerror(-err));
                exit(EXIT_FAILURE);
        } else {
                printf("\nConnected to the cluster.\n");
        }
       //创建
```

rados句柄上下文

```
        err = rados_ioctx_create(cluster, poolname, &io);
        if (err < 0) {
                fprintf(stderr, "%s: cannot open rados pool
%s: %s\n", argv[0], poolname, strerror(-err));
                rados_shutdown(cluster);
                exit(EXIT_FAILURE);
        } else {
                printf("\nCreated I/O context.\n");
        }
        //写对象
```

```
        err = rados_write(io, "hw", "Hello World!", 12, 0);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot write object
\"hw\" to pool %s: %s\n", argv[0], poolname, strerror(-
err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
```

```
                printf("\nWrote \"Hello World\" to object
\"hw\".\n");
        }
       //设置对象属性
```

```
        err = rados_setxattr(io, "hw", "lang", xattr, 5);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot write xattr to
pool %s: %s\n", argv[0], poolname, strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nWrote \"en_US\" to xattr \"lang\"
for object \"hw\".\n");
        }
        rados_completion_t comp;
       //确认异步
```

rados句柄成功创建

```
        err = rados_aio_create_completion(NULL, NULL, NULL,
```

```
&comp);
        if (err < 0) {
                fprintf(stderr, "%s: Could not create aio
completion: %s\n", argv[0], strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nCreated AIO completion.\n");
        }
        /* Next, read data using rados_aio_read. */
        //异步读对象









        err = rados_aio_read(io, "hw", comp, read_res, 12,
0);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot read object. %s
%s\n", argv[0], poolname, strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRead object \"hw\". The contents
are:\n %s \n", read_res);
        }
       //等待对象上的操作完成









        rados_wait_for_complete(comp);
        // 释放
```

complete句柄

```
        rados_aio_release(comp);
        char xattr_res[100];
       //获取对象
```

```
        err = rados_getxattr(io, "hw", "lang", xattr_res,
5);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot read xattr. %s
%s\n", argv[0], poolname, strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRead xattr \"lang\" for object
\"hw\". The contents are:\n %s \n", xattr_res);
        }
       //移除对象属性
```

```
        err = rados_rmxattr(io, "hw", "lang");
        if (err < 0) {
                fprintf(stderr, "%s: Cannot remove xattr. %s
%s\n", argv[0], poolname, strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRemoved xattr \"lang\" for object
\"hw\".\n");
        }
       //删除对象
```

```
        err = rados_remove(io, "hw");
        if (err < 0) {
                fprintf(stderr, "%s: Cannot remove object.
%s %s\n", argv[0], poolname, strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRemoved object \"hw\".\n");
        }
    rados_ioctx_destroy(io);                    //销毁
```

```
io上下文

    rados_shutdown(cluster);                    //销毁句柄

}
```

2. LIBRADOS Java语言示例

下面是LIBRADOS Java语言示例。

```
import com.ceph.rados.Rados;
import com.ceph.rados.RadosException;
import java.io.File;
public class CephClient {
        public static void main (String args[]){
                try {
                        //获取句柄
```

```
                        Rados cluster = new Rados("admin");
                        System.out.println("Created cluster
handle.");
                        File f = new
File("/etc/ceph/ceph.conf");
                        //读取配置文件









                        cluster.confReadFile(f);
                        System.out.println("Read the
configuration file.");
                       //连接集群









                        cluster.connect();
                        System.out.println("Connected to the
cluster.");
                } catch (RadosException e) {

System.out.println(e.getMessage()+":"+e.getReturnValue());
                }
        }
}
```

3. LIBRADOS Python语言示例

下面是LIBRADOS Python语言示例。

```
#!/usr/bin/python
#encoding=utf-8
import rados, sys
cluster = rados.Rados(conffile='/etc/ceph/ceph.conf')    #获取
句柄
```

```
print "\n\nI/O Context and Object Operations"
print "================================="
print "\nCreating a context for the 'data' pool"
if not cluster.pool_exists('data'):
raise RuntimeError('No data pool exists')
ioctx = cluster.open_ioctx('data')                    #获取
```

```
pool的
```

io上下文句柄

```
print "\nWriting object 'hw' with contents 'Hello World!' to
pool 'data'."
ioctx.write("hw", "Hello World!")                    #写入对象
```

```
print "Writing XATTR 'lang' with value 'en_US' to object
'hw'"
ioctx.set_xattr("hw", "lang", "en_US")              #设置对象的属性
```

```
print "\nWriting object 'bm' with contents 'Bonjour tout le
monde!' to pool 'data'."
ioctx.write("bm", "Bonjour tout le monde!")
print "Writing XATTR 'lang' with value 'fr_FR' to object
'bm'"
ioctx.set_xattr("bm", "lang", "fr_FR")
print "\nContents of object 'hw'\n------------------------"
print ioctx.read("hw")                       #读取对象
```

```
print "\n\nGetting XATTR 'lang' from object 'hw'"
print ioctx.get_xattr("hw", "lang")         #读取对象属性
```

```
print "\nContents of object 'bm'\n------------------------"
print ioctx.read("bm")
  print "\nClosing the connection."
ioctx.close()                               #关闭
```

io上下文

```
print "Shutting down the handle."
cluster.shutdown()                          #销毁句柄
```

2.4 本章小结

本章从宏观角度剖析了Ceph架构，将Ceph架构分为基础存储系统RADOS、基于RADOS实现的CEPHFS，基于RADOS的LIBRADOS层应用接口、基于LIBRADOS的应用接口RBD、RADOSGW，其中着重讲解了RADOS的组成部分MON、OSD及其功能，最后解析LIBRADOS API的基本用法。

第3章　智能分布CRUSH

3.1　引言

数据分布是分布式存储系统的一个重要部分，数据分布算法至少要考虑以下3个因素。

1）故障域隔离。同份数据的不同副本分布在不同的故障域，降低数据损坏的风险。

2）负载均衡。数据能够均匀地分布在磁盘容量不等的存储节点，避免部分节点空闲，部分节点超载，从而影响系统性能。

3）控制节点加入离开时引起的数据迁移量。当节点离开时，最优的数据迁移是只有离线节点上的数据

被迁移到其他节点，而正常工作的节点的数据不会发生迁移。

对象存储中一致性Hash和Ceph的CRUSH算法是使用比较多的数据分布算法。在Aamzon的Dyanmo键值存储系统中采用一致性Hash算法，并且对它做了很多优化。OpenStack的Swift对象存储系统也使用了一致性Hash算法。

CRUSH（Controlled Replication Under Scalable Hashing）是一种基于伪随机控制数据分布、复制的算法。Ceph是为大规模分布式存储系统（PB级的数据和成百上千台存储设备）而设计的，在大规模的存储系统里，必须考虑数据的平衡分布和负载（提高资源利用率）、最大化系统的性能，以及系统的扩展和硬件容错等。CRUSH就是为解决以上问题而设计的。在Ceph集群里，CRUSH只需要一个简洁而层次

清晰的设备描述，包括存储集群和副本放置策略，就可以有效地把数据对象映射到存储设备上，且这个过程是完全分布式的，在集群系统中的任何一方都可以独立计算任何对象的位置；另外，大型系统存储结构是动态变化的（存储节点的扩展或者缩容、硬件故障等），CRUSH能够处理存储设备的变更（添加或删除），并最小化由于存储设备的变更而导致的数据迁移。

3.2 CRUSH基本原理

众所周知，存储设备具有吞吐量限制，它影响读写性能和可扩展性能。所以，存储系统通常都支持条带化以增加存储系统的吞吐量并提升性能，数据条带化最常见的方式是做RAID。与Ceph的条带化最相似的是RAID 0或者是“带区卷”。Ceph条带化提供了类似于RAID 0的吞吐量，N路RAID镜像的可靠性以及更快速的恢复能力。

在磁盘阵列中，数据是以条带（stripe）的方式贯穿在磁盘阵列所有硬盘中的。这种数据的分配方式可以弥补OS读取数据量跟不上的不足。

1）将条带单元（stripe unit）从阵列的第一个硬盘到最后一个硬盘收集起来，就可以称为条带

（stripe）。有的时候，条带单元也被称为交错深度。在光纤技术中，一个条带单元被叫作段。

2）数据在阵列中的硬盘上是以条带的形式分布的，条带化是指数据在阵列中所有硬盘中的存储过程。文件中的数据被分割成小块的数据段在阵列中的硬盘上顺序的存储，这个最小数据块就叫作条带单元。

决定Ceph条带化数据的3个因素。

• 对象大小：处于分布式集群中的对象拥有一个最大可配置的尺寸（例如，2MB、4MB等），对象大小应该足够大以适应大量的条带单元。

• 条带宽度：条带有一个可以配置的单元大小，Ceph Client端将数据写入对象分成相同大小的条带单元，除了最后一个条带之外；每个条带宽度，应该是

对象大小的一小部分，这样使得一个对象可以包含多个条带单元。

• 条带总量：Ceph客户端写入一系列的条带单元到一系列的对象，这就决定了条带的总量，这些对象被称为对象集，当Ceph客户端端写入的对象集合中的最后一个对象之后，它将会返回到对象集合中的第一个对象处。

3.2.1 Object与PG

Ceph条带化之后，将获得N个带有唯一oid（即object的id）。Object id是进行线性映射生成的，即由file的元数据、Ceph条带化产生的Object的序号连缀而成。此时Object需要映射到PG中，该映射包括两部分。

1）由Ceph集群指定的静态Hash函数计算Object的oid，获取到其Hash值。

2）将该Hash值与mask进行与操作，从而获得PG ID。

根据RADOS的设计，假定集群中设定的PG总数为M（M一般为2的整数幂），则mask的值为M－1。由此，Hash值计算之后，进行按位与操作是想从所有PG中近

似均匀地随机选择。基于该原理以及概率论的相关原理，当用于数量庞大的Object以及PG时，获得到的PG ID是近似均匀的。

计算PG的ID示例如下。

1）Client输入pool ID和对象ID（如pool=‘liverpool’，object-id=‘john’）。

2）CRUSH获得对象ID并对其Hash运算。

3）CRUSH计算OSD个数，Hash取模获得PG的ID（如0x58）。

4）CRUSH获得已命名pool的ID（如liverpool=4）。

5）CRUSH预先考虑到pool ID相同的PG ID（如4.0x58）。

3.2.2 PG与OSD

由PG映射到数据存储的实际单元OSD中，该映射是由CRUSH算法来确定的，将PG ID作为该算法的输入，获得到包含N个OSD的集合，集合中第一个OSD被作为主OSD，其他的OSD则依次作为从OSD。N为该PG所在POOL下的副本数目，在生产环境中N一般为3；OSD集合中的OSD将共同存储和维护该PG下的Object。需要注意的是，CRUSH算法的结果不是绝对不变的，而是受到其他因素的影响。其影响因素主要有以下两个。

一是当前系统状态 。也就是上文逻辑结构中曾经提及的Cluster Map（集群映射）。当系统中的OSD状态、数量发生变化时，Cluster Map可能发生变化，而这种变化将会影响到PG与OSD之间的映射。

二是存储策略配置 。这里的策略主要与安全相关。利用策略配置，系统管理员可以指定承载同一个PG的3个OSD分别位于数据中心的不同服务器乃至机架上，从而进一步改善存储的可靠性。

因此，只有在Cluster Map和存储策略都不发生变化的时候，PG和OSD之间的映射关系才是固定不变的。在实际使用中，策略一经配置通常不会改变。而系统状态的改变或者是因为设备损坏，或者是因为存储集群规模扩大。好在Ceph本身提供了对于这种变化的自动化支持，因而，即便PG与OSD之间的映射关系发生了变化，并不会对应用造成困扰。事实上，Ceph正是需要有目的的利用这种动态映射关系。正是利用了CRUSH的动态特性，Ceph才可以将一个PG根据需要动态迁移到不同的OSD组合上，从而自动化地实现高可靠性、数据分布re-blancing等特性。

之所以在此次映射中使用CRUSH算法，而不是其他Hash算法，原因之一是CRUSH具有上述可配置特性，可以根据管理员的配置参数决定OSD的物理位置映射策略；另一方面是因为CRUSH具有特殊的“稳定性”，也就是当系统中加入新的OSD导致系统规模增大时，大部分PG与OSD之间的映射关系不会发生改变，只有少部分PG的映射关系会发生变化并引发数据迁移。这种可配置性和稳定性都不是普通Hash算法所能提供的。因此，CRUSH算法的设计也是Ceph的核心内容之一。

3.2.3 PG与Pool

Ceph存储系统支持“池”（Pool）的概念，这是存储对象的逻辑分区。

Ceph Client端从Ceph mon端检索Cluster Map，写入对象到Pool。Pool的副本数目，Crush规则和PG数目决定了Ceph将数据存储的位置，如图3-1所示。

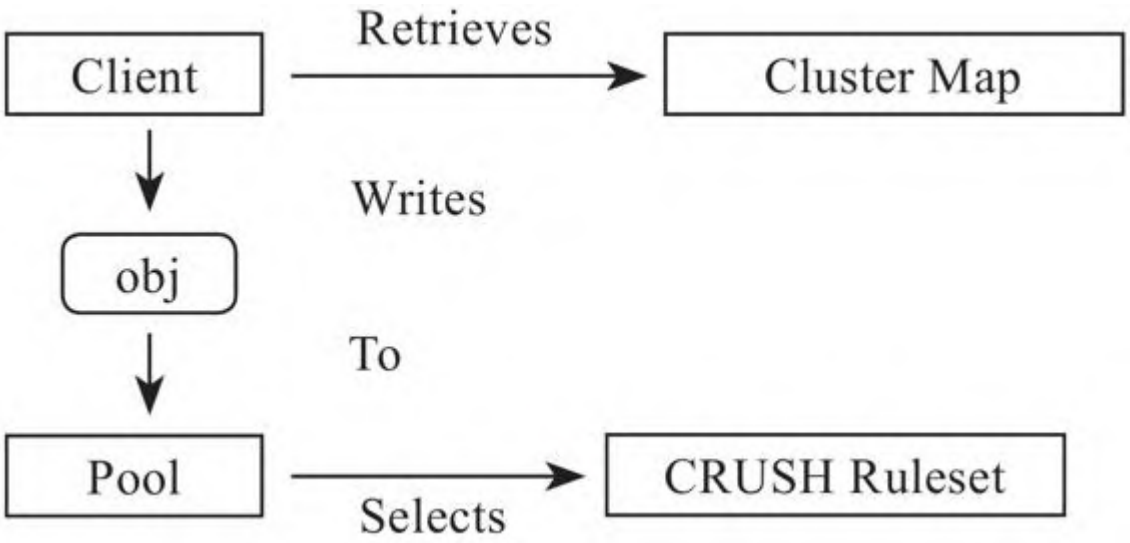

图3-1 调用关系图

Pool至少需要设定以下参数。

- 对象的所有权/访问权。
- PG数目。
- 该pool使用的crush规则。
- 对象副本的数目。

Object、PG、Pool、OSD关系图，如图3-2所示。

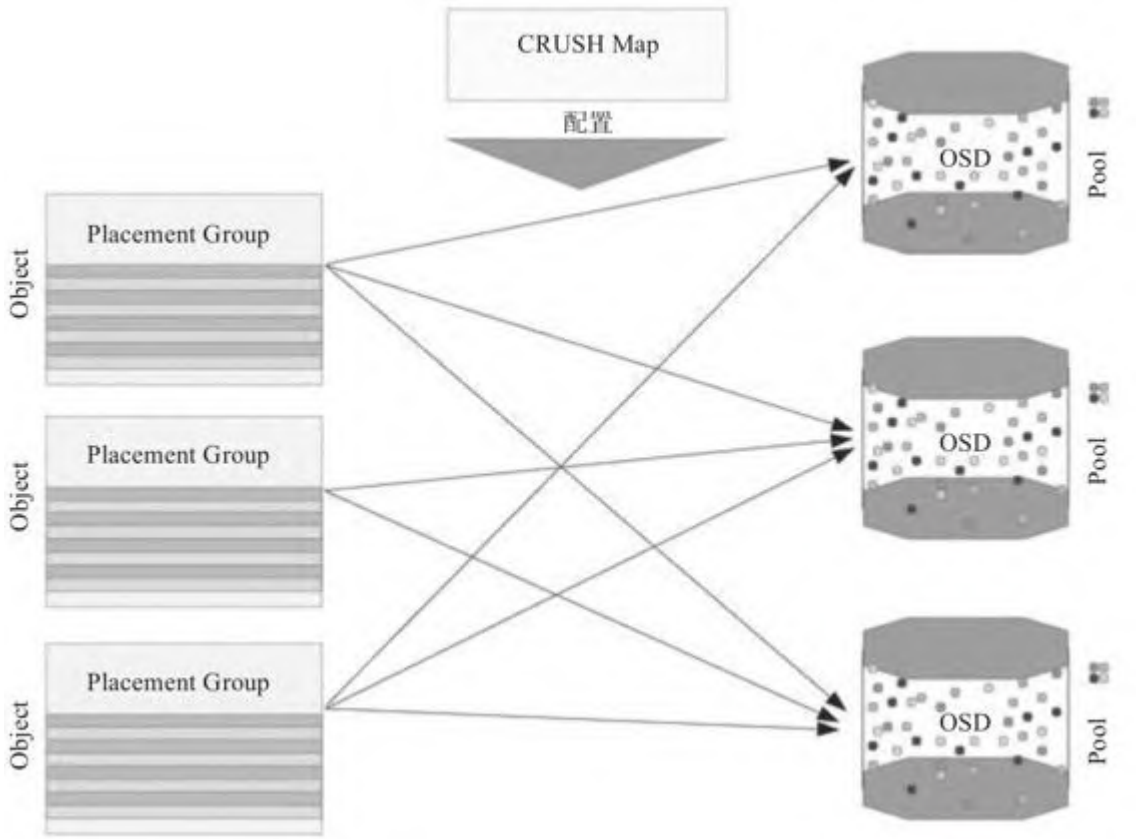

图3-2　映射关系图

Pool相关的操作如下。

1）创建pool。

```
ceph osd pool create {pool-name} {pg-num} [{pgp-num}]
[replicated] \
     [crush-ruleset-name] [expected-num-objects]
ceph osd pool create {pool-name} {pg-num}  {pgp-num}
erasure \
     [erasure-code-profile] [crush-ruleset-name]
[expected_num_objects]
```

2）配置pool配额。

```
ceph osd pool set-quota {pool-name} [max_objects {obj-
count}] [max_bytes {bytes}]
```

3）删除pool。

```
ceph osd pool delete {pool-name} [{pool-name} --yes-i-
really-really-mean-it]
```

4）重命名pool。

```
ceph osd pool rename {current-pool-name} {new-pool-name}
```

5）展示pool统计。

```
rados df
```

6）给pool做快照。

```
ceph osd pool mksnap {pool-name} {snap-name}
```

7）删除pool快照。

```
ceph osd pool rmsnap {pool-name} {snap-name}
```

8）配置pool的相关参数。

```
ceph osd pool set {pool-name} {key} {value}
```

9）获取pool参数的值。

```
ceph osd pool get {pool-name} {key}
```

10）配置对象副本数目。

```
ceph osd pool set {poolname} size {num-replicas}
```

11）获取对对象副本数目。

```
ceph osd dump | grep 'replicated size'
```

3.3 CRUSH关系分析

从本质上讲，CRUSH算法是通过存储设备的权重来计算数据对象的分布的。在计算过程中，通过Cluster Map（集群映射）、Data Distribution Policy（数据分布策略）和给出的一个随机数共同决定数据对象的最终位置。

1.Cluster Map

Cluster Map记录所有可用的存储资源及相互之间的空间层次结构（集群中有多少个机架、机架上有多少服务器、每个机器上有多少磁盘等信息）。所谓的Map，顾名思义，就是类似于我们生活中的地图。在Ceph存储里，数据的索引都是通过各种不同的Map来实现的。另一方面，Map使得Ceph集群存储设备在物理层作了一层防护。例如，在多副本（常见的三副本）的结构上，通过设置合理的Map（故障域设置为Host级），可以保证在某一服务器死机的情况下，有其他副本保留在正常的存储节点上，能够继续提供服务，实现存储的高可用。设置更高的故障域级别（如Rack、Row等）能保证整机柜或同一排机柜在掉电情况下数据的可用性和完整性。

（1）Cluster Map的分层结构

Cluster Map由Device和Bucket构成。它们都有自己的ID和权重值，并且形成一个以Device为叶子节点、Bucket为躯干的树状结构，如图3-3所示。

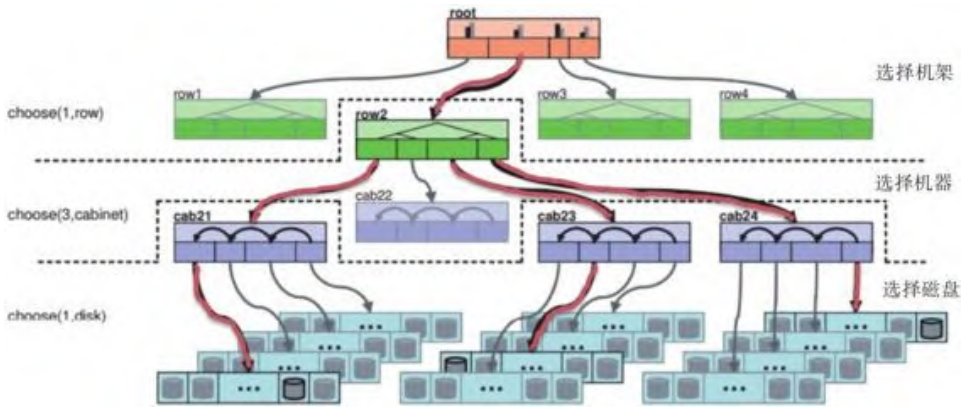

图3-3　CRUSH架构图

Bucket拥有不同的类型，如Host、Row、Rack、Room等，通常我们默认把机架类型定义为Rack，主机类型定义为Host，数据中心（IDC机房）定义为Data Center。Bucket的类型都是虚拟结构，可以根据自己的喜好设计合适的类型。Device节点的权重值代表了存储设备的容量与性能。其中，磁盘容量是权重大小的关键因素。

OSD的权重值越高，对应磁盘会被分配写入更多的数据。总体来看，数据会被均匀写入分布于群集所有磁盘，从而提高整体

性能和可靠性。无论磁盘的规格容量，总能够均匀使用。

关于OSD权重值的大小值的配比，官方默认值设置为1TB容量的硬盘，对应权重值为1。

可以在/etc/init.d/ceph原码里查看相关的内容。

…

```
363             get_conf osd_weight "" "osd crush initial weight"
364             defaultweight="$(df -P -k $osd_data/. | tail -1 | awk '{
print sprintf("%.2f",$2/1073741824) }')"  ###此处就是
```

ceph默认权重的设置值

```
            get_conf osd_keyring "$osd_data/keyring" "keyring"
366             do_cmd_okfail "timeout 30 $BINDIR/ceph -c $conf --
name=osd.$id --keyring=$osd_keyring     osd crush create-or-move -- $id
${osd_weight:-${defaultweight:-1}} $osd_location"…
```

（2）恢复与动态平衡

在默认设置下，当集群里有组件出现故障时（主要是OSD，也可能是磁盘或者网络等），Ceph会把OSD标记为down，如果在300s内未能回复，集群就会开始进行恢复状态。这个“300s”可以通过“mon osd down out interval”配置选项修改等待时间。PG（PlacementGroups）是Ceph数据管理（包括复制、修复等动作）单元。当客户端把读写请求（对象单元）推送到Ceph时，通过CRUSH提供的Hash算法把对象映射到PG。PG在CRUSH策略的影响下，最终会被映射到OSD上。

2.Data Distribution Policy

Data Distribution Policy由Placement Rules组成。Rule决定了每个数据对象有多少个副本，这些副本存储的限制条件（比如3个副本放在不同的机架中）。一个典型的rule如下所示。

```
rule replicated_ruleset {     ##rule名字

    ruleset 0          # rule的

Id
    type replicated    ## 类型为副本模式，另外一种模式为纠删码（

EC）

    min_size 1         ## 如果存储池的副本数大于这个值，此

rule不会应用
```

```
    max_size 10         ## 如要存储池的副本数大于这个值，此
```

rule不会应用

```
    step take default   ## 以
```

default root为入口

```
    step chooseleaf firstn 0 type host   ## 隔离域为
```

host级，即不同副本在不同的主机上

```
        step emit          ## 提交



    }
```

根据实际的设备环境，可以定制出符合自己需求的Rule，详见第10章。

3.CRUSH中的伪随机

先来看一下数据映射到具体OSD的函数表达形式。

```
CRUSH(x)      ->      (osd1,      osd2,      osd3.....osdn)
```

CRUSH使用了多参数的Hash函数在Hash之后，映射都是按既定规则选择的，这使得从x到OSD的集合是确定的和独立的。CRUSH只使用Cluster Map、Placement Rules、X。CRUSH是伪随

机算法，相似输入的结果之间没有相关性。关于伪随机的确认性和独立性，以下我们以一个实例来展示。

```
 [root@host-192-168-0-16 ~]# ceph osd tree
ID WEIGHT  TYPE NAME               UP/DOWN  REWEIGHT  PRIMARY-AFFINITY
-1 0.35999 root default
-3 0.09000     host host-192-168-0-17
 2 0.09000         osd.2                   up  1.00000          1.00000
-4 0.09000     host host-192-168-0-18
 3 0.09000         osd.3                   up  1.00000          1.00000
-2 0.17999     host host-192-168-0-16
 0 0.09000         osd.0                   up  1.00000          1.00000
 1 0.09000         osd.1                   up  1.00000          1.00000
```

以上是某个Ceph集群的存储的CRUSH结构。在此集群里我们手动创建一个pool，并指定为8个PG和PGP。

```
[root@host-192-168-0-16 ~]# ceph  osd pool create  crush  8   8
pool 'crush' created
```

注意

PGP是PG的逻辑承载体，是CRUSH算法不可缺少的部分。在Ceph集群里，增加PG数量，PG到OSD的映射关系就会发生变化，但此时存储在PG里的数据并不会发生迁移，只有当PGP的数量也增加时，数据迁移才会真正开始。关于PG和PGP的关系，假如把PG比作参加宴会的人，那么PGP就是人坐的椅子，如果人员增加时，人的座位排序就会发生变化，只有增加椅子时，真正的座位

排序变更才会落实。因此，人和椅子的数量一般都保持一致。所以，在Ceph里，通常把PGP和PG设置成一致的。

可以查看PG映射OSD的集合，即PG具体分配到OSD的归属。

```
 [root@host-192-168-0-16 ~]# ceph pg dump | grep ^22\. | awk '{print $1 "\t"   $17}'    ## 22 表示
```

```
 Pool id ##
dumped all in format plain
22.1     [1,2,3]
22.0       [1,2,3]
22.3      [3,1,2]
22.2      [3,2,0]
22.5     [1,3,2]
22.4      [3,1,2]
22.7      [2,1,3]
22.6      [3,0,2]
```

上面示例中，第一列是PG的编号（其中22表示的Pool的ID），共计8个，第二列是PG映射到了具体的OSD集合。如“22.1[1,2,3]”表示PG 22.1分别在OSD1、OSD2和OSD3分别存储副本。

在Ceph集群里，当有数据对象要写入集群时，需要进行两次映射。第一次从object→PG，第二次是PG→OSD set。每一次的

映射都是与其他对象无相关的。以上充分体现了CRUSH的独立性（充分分散）和确定性（可确定的存储位置）。

3.4 本章小结

本章主要围绕Ceph的核心——CRUSH展开，讲述了CRUSH的基本原理以及CRUSH的特性。读者可以了解Ceph基本要素的概念，重点应该把握Ceph的Object、PG、Pool等基本核心概念，熟悉读写流程以及CRUSH算法的组成、影响因素及选择流程。基于以上掌握的要点，为后面的深入学习打好坚实的基础。

第4章　三大存储访问类型

4.1　Ceph FS文件系统

Ceph Filesystem简称Ceph FS，是一个支持POSIX接口的文件系统存储类型。Ceph FS的发展目前比较滞后，主要是因为Ceph FS技术不成熟，另一方面或许由于云计算大潮的突起，比Ceph FS起步晚的Ceph RBD和Ceph Radosgw发展反而比较活跃，社区的发展重点也大多放在了后两者上。不过在Red Hat收购Inktank后，以及许多应用环境对Ceph FS需求量大的原因，目前Ceph FS正在越来越被重视。

Ceph官方社区于2016年4月21日发布了Jewel V10.2.0版本，Ceph FS在该版本中被提到它是第一个

公开宣称稳定的版本，至于生产环境中的使用和反馈情况还需要时间来证明。

文件系统的主要特点就是客户端可以很方便地挂载到本地使用，文件系统还可以提供资源共享的作用。Ceph FS文件系统存储类型继承了RADOS的容错性和扩展性，相比NFS或CIFS，Ceph FS可以提供副本冗余，具有数据高可靠的特性，Ceph FS的这些重要的特性也是许多应用环境中对其需求量大的原因。

Ceph FS需要使用Metadata Server（简称MDS）来管理文件系统的命名空间以及客户端如何访问到后端OSD数据存储中。MDS类似于ceph-mon，是一个服务进程，在使用Ceph FS前首先要安装和启动ceph-mds服务。

4.1.1 Ceph FS和MDS介绍

在介绍Ceph FS之前，先介绍一下什么是元数据以及它的作用。元数据主要负责记录数据的属性，如文件存储位置、文件大小和存储时间等，负责资源查找、文件记录、存储位置记录、访问授权等功能。

MDS（Metadata Server）以一个Daemon进程运行一个服务，即元数据服务器，主要负责Ceph FS集群中文件和目录的管理，确保它们的一致性，MDS也可以像MON或OSD一样多节点部署实现冗余。

Ceph文件系统主要依赖MDS守护进程提供服务，MDS提供了一个包含智能缓存层的一致性文件系统，MDS不会直接向客户端提供任何数据，所有的数据都只由后端OSD提供，这样的好处就是极大地降低了自身读写次数。除此之外，它还具备动态子树划分的特点，

MDS守护进程可以加入和退出，快速地接管故障节点。MDS守护进程可以被配置为活跃或被动状态，活跃的MDS被称为主MDS，其他的MDS进入Standby状态，当主MDS节点发生故障时，第二MDS节点将接管其工作并被提升为主节点。

Metadata信息采用在内存中缓存的方式响应外部访问请求，下面结合图4-1介绍一下MDS的交互模型。

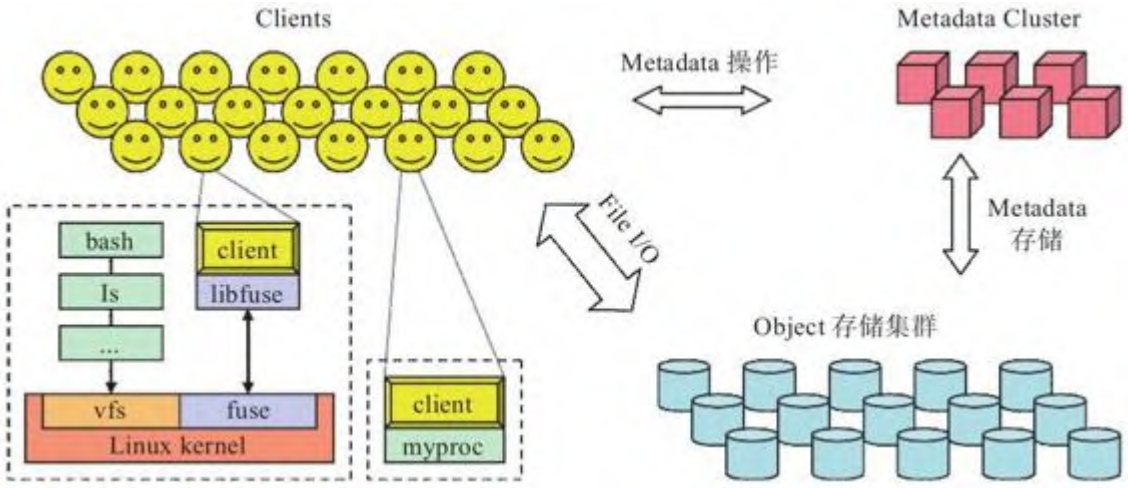

图4-1　MDS的交互模型

当一个或多个客户端打开一个文件时，客户端向MDS发送请求，实际上就是MDS向OSD定位该文件所在的

文件索引节点（File Inode），该索引节点包含一个唯一的数字、文件所有者、大小和权限等其他元数据，MDS会赋予Client读和缓存文件内容的权限，访问被授权后返回给客户端File Inode值、Layout（Layout可以定义文件内容如何被映射到Object）和文件大小，客户端根据MDS返回的信息定位到要访问的文件，然后直接与OSD执行File IO交互。

同样，当客户端对文件执行写操作时，MDS赋予Client带有缓冲区的写权限，Client对文件写操作后提交给MDS，MDS会将该新文件的信息重新写入到OSD中的Object中。

图4-2是一个MDS树状结构，MDS集群为了适应分布式缓存元数据的特点，采用了一种叫作动态子树分区（Subtree Partitioning）的策略，Subtree可以理解为横跨多个MDS节点的目录层级结构，MDS统计一个目

录中元数据信息被访问的频繁程度并计数，MDS还维护了一个称作权重的树，其记录的是最近元数据的负载情况，MDS会定期的加载并比较前后权重值的大小，然后根据权重值适当地迁移子树以实现工作流分散在每个MDS中。访问程度频繁被称为热点的目录还可以进行Hash算法后存在多个MDS上。

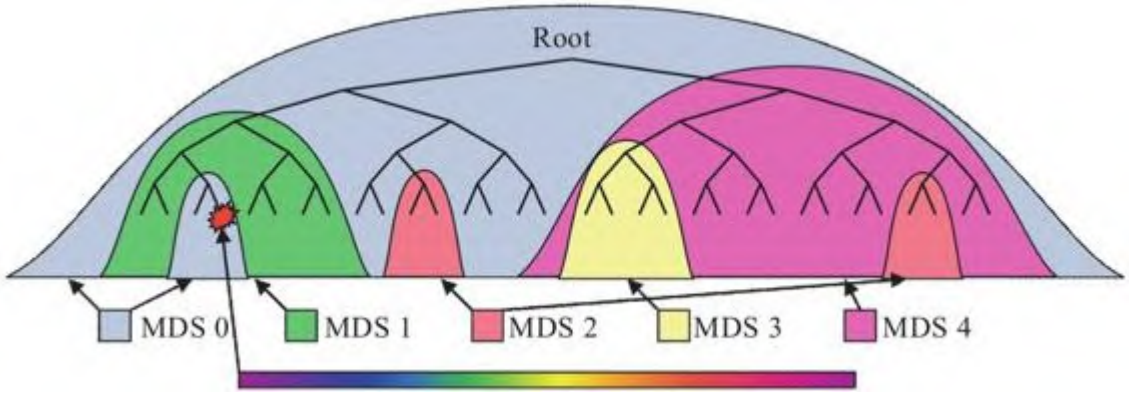

图4-2　MDS树状结构

如图4-3所示，客户端访问Ceph FS主要有分为Kernel内核驱动和Ceph FS FUSE两种方式，Linux内核从2.6.34版本开始加入了对Ceph FS的原生支持，如果客户端使用的内核版本较低或者一些应用程序依赖问

题，也可以通过FUSE（Filesystem in Userspace）客户端来挂载Ceph文件系统。

在下面的章节中会介绍如何创建Ceph FS、如何通过Kernel内核驱动或FUSE挂载Ceph文件系统。

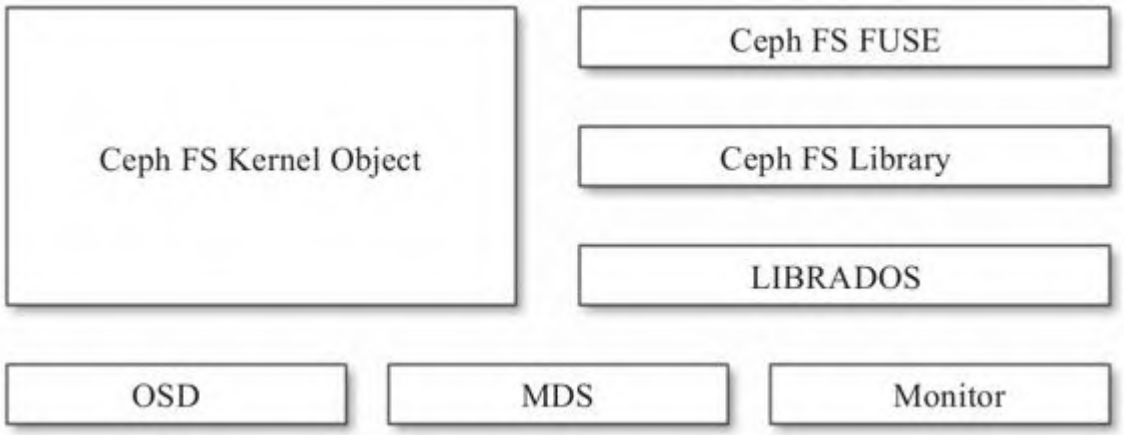

图4-3　客户端访问Ceph FS

4.1.2　部署MDS

按照前面的章节搭建好一套可以使用的Ceph分布式存储集群，下面接着部署MDS。在ceph-deploy节点上执行ceph-deploy命令为其他OSD节点创建MDS，执行结果如图4-4所示。

```
# ceph-deploy --overwrite-conf mds create ceph1
```

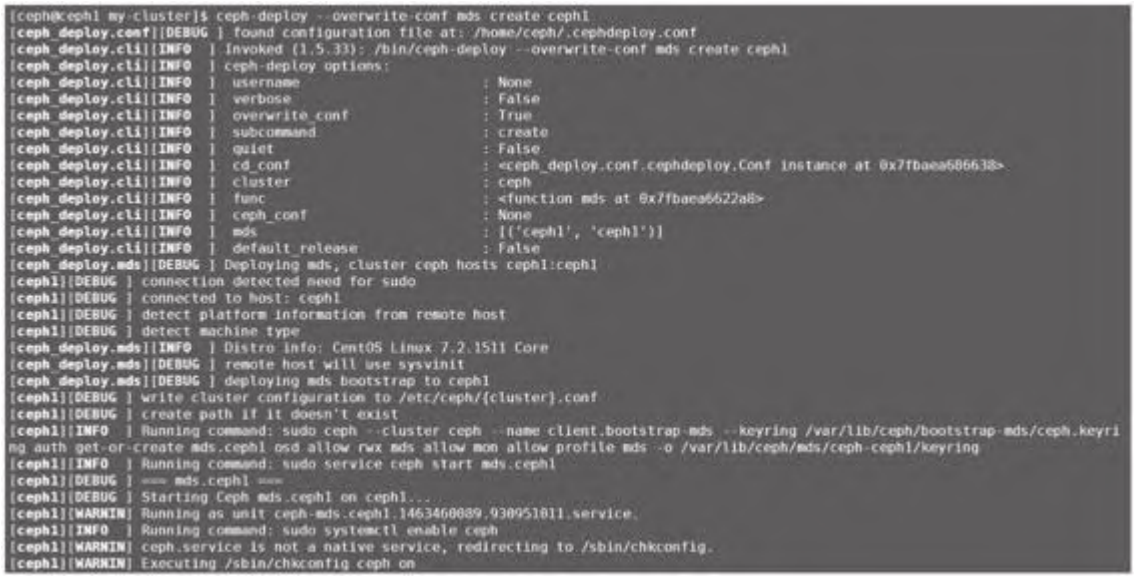

图4-4　执行结果

MDS需要使用两个Pool，一个Pool用来存储数据，一个Pool用来存储元数据。所以，创建fs_data和fs_metadata两个存储池，一个用来存储数据，另一个存储元数据。

```
# ceph osd pool create fs_data <pg_num>
# ceph osd pool create fs_metadata <pg_num>
```

创建一个Ceph FS。

命令格式：

```
ceph fs new <fs_name> <metadata> <data>
# ceph fs new cephfs fs_metadata fs_data
```

查看创建的Ceph FS。

```
# ceph fs ls
name: cephfs, metadata pool: metadata, data pools: [data ]
# ceph mds stat
e5: 1/1/1 up {0=ceph1=up:active}, 2 up:standby
```

4.1.3 挂载Ceph FS

在Linux中挂载Ceph FS有两种方式：Kernel Driver和FUSE。

关于FUSE的介绍参见：https://en.wikipedia.org/wiki/Filesystem_in_Userspace 。

（1）Kernel驱动方式

通过Kernel内核驱动挂载Ceph FS。

```
# mkdir /cephfs_test
# mount -t ceph 172.16.6.113:6789:/  /cephfs_test
```

为了实现系统启动时自动挂载，添加下面一行记录到/etc/fstab中。

```
172.16.6.113:6789://cephfs_test ceph noatime 0 2
```

（2）FUSE方式

通过挂载FUSE方式挂载Ceph FS。

```
# yum install ceph-fuse
# mkdir /fuse_test
# ceph-fuse -m 172.16.6.113:6789  /fuse_test
```

为了实现系统启动时自动挂载，添加下面一行记录到/etc/fstab中。

```
id=admin,conf=/etc/ceph/ceph.conf/fuse_test fuse.ceph
defaults 0 0
```

用df-h形式查看Ceph FS的挂载结果。

```
#df-h
```

命令输出：

```
172.16.6.113:6789:/   570G  232M  570G   1% /cephfs_test
ceph-fuse             570G  232M  570G   1% /fuse_test
```

上面输出结果中，第1条挂载记录采用Kernel Driver，第2条采用ceph-fuse方式。

4.2 RBD块存储

RBD块存储是Ceph提供的3种存储类型中使用最广泛、最稳定的存储类型。RBD块设备类似于磁盘，它可以挂载到物理机或虚拟机中，挂载的方式通常有以下两种。

- Kernel模块（简称KRBD）方式。
- 利用QEMU模拟器通过librbd方式。

本节就RBD块设备的两种使用方式以及相关操作进行简单介绍。

4.2.1 RBD介绍

RBD是RADOS Block Device的简称。RBD是Ceph分布式集群中最常用的存储类型。

块是一个有序字节，普通的一个块大小为512字节。基于块的存储是最常见的存储方式，比如常见的硬盘、软盘和CD光盘等，都是存储数据最简单快捷的设备。

Ceph块设备是一种精简置备模式，可以扩展大小且数据是以条带化的方式存储在一个集群中的多个OSD中，RBD具有快照、多副本、克隆和一致性功能。

下面是Ceph块设备通过Linux内核模块或者librbd库与OSD之间交互通信的模式。

一般在为物理主机提供块设备时才使用Kernel的RBD模块，当Ceph块设备使用基于内核模块驱动时，可以使用Linux自带的页缓存（Page Caching）来提高性能[1]；而为虚拟机（比如QEMU/KVM类型）提供块设备时，通常是利用Libvirt调用librbd库的方式提供块设备，使用基于librbd的模块时可以使用RBD缓存[2]（Cache）来提供性能。

[1] Linux page caching：https://en.wikipedia.org/wiki/Page_cache。

[2] RBD caching：http://docs.ceph.com/docs/master/rbd/rbd-config-ref/。

4.2.2 librbd介绍

librbd是一个访问rbd块存储的库，大家知道librados提供了RBD、Ceph FS和Radosgw三种存储接口，其中librbd就是利用librados与RBD进行交互的。

前面提到librbd主要用于为虚拟机提供块设备，现在最常用的是在IaaS层提供虚拟机，比如备受追捧的提供IaaS层服务的开源软件框架OpenStack，它是基于KVM/QEMU虚拟化模拟器提供虚拟机服务的，为虚拟机系统盘和数据盘提供块设备的就是大量使用Ceph提供的RBD块存储。

后面会介绍OpenStack堆栈中关于librbd的使用，这里不再进行介绍。

图4-5是QEMU通过librbd访问后端存储的模型。

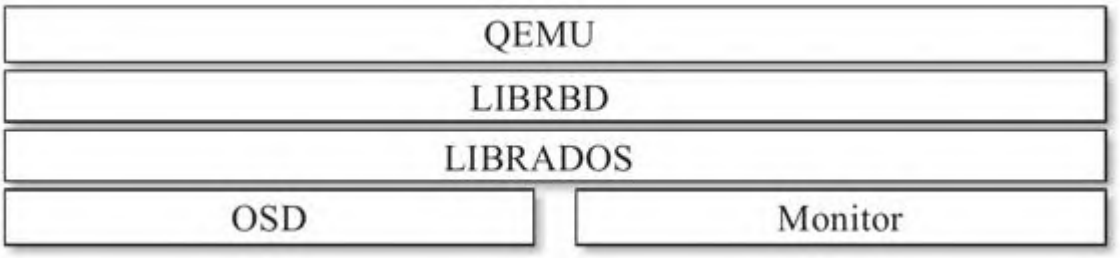

图4-5　QEMU使用RBD块存储

4.2.3 KRBD介绍

KRBD是Kernel Rados Block Device的简称，用来通过Kernel中的RBD模块实现访问后端存储。

按照前面的章节搭建好一套可以使用的Ceph分布式存储集群，在操作系统内使用modprobe命令加载RBD模块到内核（默认没有加载）。

下面是通过KRBD模块访问Ceph后端块设备的命令操作，以CentOS 7.1系统为例。

1）加载RBD内核模块。

```
# modprobe rbd
```

2）查看RBD模块信息。

命令：

```
# modinfo rbd
```

输出：

```
filename:  /lib/modules/3.10.0-
229.20.1.el7.x86_64/kernel/drivers/block/rbd.ko
license:   GPL
description: RADOS Block Device (RBD) driver……
```

3）查看RBD模块所属的软件包。

命令：

```
# rpm -


qf /lib/modules/3.10.0-
229.20.1.el7.x86_64/kernel/drivers/block/rbd.ko
```

输出：

```
kernel-3.10.0-229.20.1.el7.x86_64
```

 注意

RHEL 6.x、CentOS 6.x等低版本内核的Linux操作系统，其内核中默认没有集成RBD驱动，RHEL 7.x、CentOS 7.x、Ubuntu14.04及其他Linux衍生版本带有3.10.x内核的操作系统中已经默认集成了RBD驱动。RBD的使用依赖内核版本，为了提高性能和避免Bug，建议内核版本不小于3.10。

4）创建一个10GB大小的块设备。

命令：

```
# rbd create test_image --size 10240
```

5）查看创建的块设备。

命令：

```
# rbd list
# rbd info test_image
```

输出：

```
rbd image 'test_image':
        size 10240 MB in 2560 objects
        order 22 (4096 kB objects)
        block_name_prefix: rb.0.1041.6b8b4567
        format: 1
```

6）把test_image块设备映射到操作系统。

命令：

```
# rbd map test_image
```

输出：

```
/dev/rbd0
```

7）查看系统中已经映射的块设备。

命令：

```
# rbd showmapped
```

输出：

```
id pool image      snap device
0  rbd  test_image -    /dev/rbd0
```

命令：

```
# ll /dev/rbd0
```

输出：

```
brw-rw---- 1 root disk 253, 0 Nov 23 22:45 /dev/rbd0
```

8）取消块设备映射。

```
# rbd unmap /dev/rbd0
```

4.2.4 RBD操作

RBD的使用其实就是对后端RBD image进行使用，RBD image是由多个Object条带化存储到后端的块设备。

下面介绍对RBD的常见操作。

1.存储池Pool

1）创建一个存储池（pool）。

命令：

```
# rados mkpool pool
```

2）查看创建出的pool。

命令：

```
# rados lspools
```

输出：

```
rbd
pool
```

 注意

Ceph集群刚搭建好后，会默认创建出一个名为rbd的存储池。

2.RBD image

1）在pool存储池中创建一个大小为1GB的image。

命令：

```
# rbd create pool/image1 --size 1024 --image-format 2
```

 注意

创建image时，设置的格式为2。Format 1是原始格式，也是创建image的默认格式，Format 2支持RBD分层[1]（layering），是实现COW（Copy-On-Write）的前提。

2）查看创建出的image。

命令：

```
# rbd ls pool
```

3）查看image详细信息。

命令：

```
# rbd info pool/image1
```

输出：

```
rbd image 'image1':
        size 1024 MB in 256 objects
        order 22 (4096 kB objects)
```

```
        block_name_prefix: rbd_data.10d56b8b4567
        format: 2
        features: layering
        flags:
```

Ceph集群中一个object对象默认大小为4MB，也可以在创建image时指定object大小如下。

```
# rbd create pool/image2 --size 1024 --order 24 --image-format 2
```

注意

--order 24表示指定object大小为2^24 bytes，即16MB。若不指定--order参数，则--order默认值为22，即4MB。

4）查看image2的order和object大小。

命令：

```
# rbd info pool/image2
```

输出：

```
rbd image 'image2':
        size 1024 MB in 64 objects
        order 24 (16384 kB objects)
        block_name_prefix: rbd_data.10ee6b8b4567
        format: 2
        features: layering
        flags:
```

5）删除image。

命令：

```
# rbd rm pool/image2
```

输出：

```
Removing image: 100% complete...done.
```

3. 快照Snapshot

下面是对RBD的快照操作。

1）为image创建一个快照，快照名称为image1_snap。

```
# rbd snap create pool/image1@image1_snap
```

2）查看上面创建的快照。

命令：

```
# rbd snap list  pool/image1
```

输出：

```
SNAPID NAME SIZE
6 image1_snap 1024 MB
```

或者以长格式形式查看。

命令：

```
# rbd ls pool -l
```

输出：

```
NAME SIZE PARENT FMT PROT LOCK
image1 1024M 2
image1@image1_snap 1024M  2
```

3）查看快照更详细的信息。

命令：

```
# rbd info pool/image1@image1_snap
```

输出：

```
rbd image 'image1':
        size 1024 MB in 256 objects
        order 22 (4096 kB objects)
        block_name_prefix: rbd_data.10d56b8b4567
        format: 2
        features: layering
        flags:
        protected: False
```

4. 克隆

在克隆（Cloning）前，快照必须处于被保护（protected）的状态才能够被克隆。

命令：

```
# rbd snap protect pool/image1@image1_snap
# rbd info pool/image1@image1_snap
```

输出：

```
rbd image 'image1':
        size 1024 MB in 256 objects
        order 22 (4096 kB objects)
        block_name_prefix: rbd_data.10d56b8b4567
        format: 2
        features: layering
        flags:
        protected: True
```

克隆快照到另一个RBD Pool并成为一个新的image。

```
# rbd clone pool/image1@image1_snap rbd/image2
```

新的image2依赖父（parent）image。

命令：

```
# rbd ls rbd -l
```

输出：

```
NAME SIZE PARENT FMT PROT LOCK
image2  1024M pool/image1@image1_snap 2
```

5.依赖Children/Flatten

1）查看快照的“子”（children）。

命令：

```
# rbd children pool/image1@image1_snap
```

输出：

```
rbd/image2
```

2）把分层的RBD image变为扁平的没有层级的image。

命令：

```
# rbd flatten rbd/image2
```

输出：

```
Image flatten: 100% complete...done.
```

3）再次查看rbd/image2已经没有父（parent）image存在了，即断开了依赖关系。

命令：

```
# rbd ls rbd -
```

```
1
```

输出：

```
NAME    SIZE PARENT FMT PROT LOCK
image2 1024M          2
```

6.RBD导入导出

针对RBD image的导入和导出，如下。

1）导出RBD image。

命令：

```
# rbd export pool/image1 /tmp/image1_export
```

输入：

```
Exporting image: 100% complete...done.
```

命令：

```
# ls /tmp/image1_export
```

输入：

```
/tmp/image1_export
```

2）导入RBD image。

命令：

```
# rbd import /tmp/image1_export pool/image3 --image-format 2
```

输入：

```
Importing image: 100% complete...done.
```

命令：

```
# rbd ls pool
```

输入：

```
image1
image3
```

关于RBD image的导入和导出可以作为针对RBD块设备简单的备份与恢复。

[1] rbd layering：http://docs.ceph.com/docs/v0.93/dev/rbd-layering/。

4.2.5 RBD应用场景

前面介绍了RBD基于Kernel和librbd库的使用，大多数使用场景中是基于QEMU/KVM通过librbd的方式。提到虚拟化就自然想到目前热度很高的云计算，虚拟化是云计算的核心。云计算的IaaS层一般对外提供虚拟机资源服务，比如火热的OpenStack、CloudStack和ZStack等提供基础设施堆栈的开源软件框架。RBD块设备用于虚拟机的系统卷、数据卷，根据Ceph RBD的分层（layering）功能，还可以方便实现基于COW的克隆技术，还有基于RBD的快照、导入、导出和扩容（Resizing）等功能。

详细RBD应用案例请参考第12章的生产环境应用案例。

4.3 Object对象存储

4.3.1 RGW介绍

RGW是Ceph对象存储网关服务RADOS Gateway的简称，是一套基于Librados接口封装而实现的FastCGI服务，对外提供RESTful风格的对象存储数据访问和管理接口。RGW基于HTTP协议标准，因此非常适用于Web类的互联网应用场景，用户通过使用SDK或者其他客户端工具，能够很方便地接入RGW进行图片、视频以及各类文件的上传或下载，并设置相应的访问权限，共享给其他用户，形成最简单的网盘分享，具体过程如图4-6所示。

RGW提供的兼容接口有以下3项。

1）S3兼容接口：兼容大部分Amazon S3 API标准，详见S3接口兼容列表（见表4-1）。目前这套接口应用比较广泛，也是本书介绍的重点。

2）Swift兼容接口：兼容OpenStack Swift API。

3）Admin管理接口：实现对S3用户、Bucket和Quota等信息的统一管理。

其中，S3兼容接口提供了一套简单的Web服务接口，可随时随地通过HTTP的方式在网络上的任何位置存储和检索任意数量的数据，是一套非常实用的对象存储服务标准。很多互联网公司的产品，比如图片、视频一类资源的存储与发布等都是借助这套标准服务来实现的，节约了大量的采购、开发和部署运维成本，特别适合一次写入，多用户同时读取的场景。

图4-6为Rados Gateway服务的功能组成。

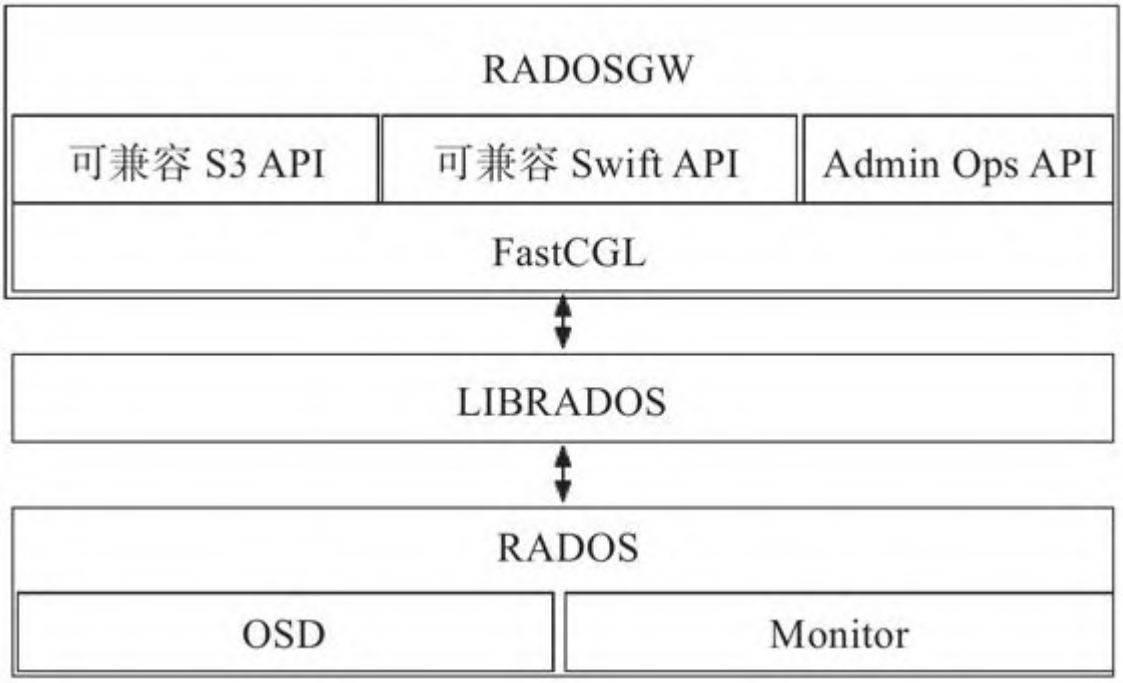

图4-6　Rados Gateway服务的功能组成

如果读者想深入了解其他接口内容请参考以下内容。

• Swift兼容列表：

http://docs.ceph.com/docs/master/radosgw/swift/

。

• admin管理接口介绍请参考

http://docs.ceph.com/docs/master/radosgw/admino

ps/ 。

4.3.2 Amazon S3简介

Amazon S3提供了一个简单的Web服务接口，可随时在Web上的任何位置存储和检索任何数量的数据。同时提供了丰富的SDK，支持Java、Python、Ruby和PHP等多种程序开发语言。由于篇幅有限，本书只介绍一些S3服务相关的基本概念，关于Amazon S3详细介绍请参考：http://docs.aws.amazon.com/zh_cn/AmazonS3/latest/dev/Introduction.html 。

1.S3服务特点

（1）安全与访问管理

S3可以针对每个对象存储桶或者对象设置独立的访问控制策略，可以允许某些S3或者访问，或者全部

对外开放等。另外，S3还支持对对象的限时访问，提供一个仅在规定时间内有效的URL。

（2）支持各种语言的SDK进行编程访问

亚马逊提供的SDK支持C++、C#、Java、PHP、Python和Ruby等常见的语言，使应用的开发能够更加灵活简单。

（3）多元化使用S3服务

现在可以通过3种方式使用S3服务。

- 通过S3 SDK编程访问。
- 通过REST API进行访问。
- 通过命令行CLI进行访问。

（4）存储对象的版本控制

对于S3存储桶中的每个对象，可以使用版本控制功能来保存各个版本，可以用于数据恢复等。

2.概念和术语

AccessKey、SecretKey：使用S3需要S3颁发的AccessKey（长度为20个字符的ASCII字符串）和SecretKey（长度为40个字符的ASCII字符串）。AccessKey用于标识客户的身份，SecretKey作为私钥形式存放于客户服务器，不在网络中传递。SecretKey通常用作计算请求签名的密钥，用以保证该请求是来自指定的客户的。使用AccessKey进行身份识别，加上SecretKey进行数字签名，即可完成应用接入与认证授权。

1）Region（区域）：创建Bucket时需要选择Region，Region一般用于标识资源存储的物理位置，比如中国区、欧洲区等。

2）Region的外网域名：比如s3.cn.ceph.work，表示中国区的S3服务对外接入地址。

3）Service（服务）：S3提供给用户的虚拟存储空间，在这个虚拟空间中，每个用户可拥有一个到多个Bucket。

4）Bucket（存储空间）：Bucket是存放Object的容器，所有的Object都必须存放在特定的Bucket中。在RGW中默认每个用户最多可以创建1000个Bucket，每个Bucket中可以存放无限多个Object。Bucket不能嵌套，每个Bucket中只能存放Object，不能再存放Bucket，Bucket下的Object是一个平级的结构。Bucket的名称全局唯一且命名规则与DNS命名规则相同。

关于Bucket的命名规范如下。

• 仅包含小写英文字母（a~z）、数字（0~9）、中线（-），即：abcdefghijklmnopqrstuvwxyz0123456789-。

• 必须由字母或数字开头。

• 长度在3~255个字符之间。

5）Object（对象，文件）：在S3中，用户操作的基本数据单元是Object。单个Object允许存储0~5TB的数据。Object包含key和data。其中，key是Object的名字；data是Object的数据。key为UTF-8编码，且编码后的长度不得超过1024个字节。

6）Key（文件名）：即Object的名字，key为UTF-8编码，且编码后的长度不得超过1024个字节。Key中可以带有斜杠，当Key中带有斜杠的时候，将会自动在控制台里组织成目录结构。

7）ACL（访问控制权限）：对Bucket和Object相关访问的控制策略，例如允许匿名用户公开访问等。目前ACL支持READ、WRITE、FULL_CONTROL三种权限。对于Bucket的拥有者，总是FULL_CONTROL。可以授予所有用户（包括匿名用户）或指定用户READ、WRITE或者FULL_CONTROL权限。

目前提供了3种预设的ACL，分别是private、public-read和public-read-write。

- private表示只有owner有READ和WRITE的权限。
- public-read表示为所有用户授予READ的权限。
- public-read-write表示为所有用户授予WRITE权限。

对于BUCKET来说，READ是指能够罗列Bucket中的Object、已经上传的分段。WRITE是指可以上传或者删

除BUCKET中Object。FULL_CONTROL则包含前面提到的针对Bucket的READ和WRITE两种操作。对于Object来说，READ是指能够查看或者下载对应的Object。WRITE是指可以写入、覆盖或删除Object。FULL_CONTROL则包含前面提到的针对Object的READ和WRITE两种操作。

8）Bucket访问控制权限：S3提供Bucket级别的权限访问控制（ACL），Bucket目前有以下3种访问权限：public-read-write、public-read和private，它们的含义如下。

• public-read-write：任何人（包括匿名访问）都可以对该Bucket中的Object进行PUT、Get和Delete操作。

• public-read：任何人（包括匿名访问）只能对该Bucket中的Object进行读操作，而不能进行写操

作。注意，对Bucket有读操作不表示对Object有读操作。

• private：只有该Bucket的创建者才可以对该Bucket及Bucket中的Object进行读写操作。

9）Obejct访问控制权限：S3提供Bucket级别的权限访问控制（ACL），Object目前有以下2种访问权限：public-read和private，它们的含义如下。

• public-read：任何人（包括匿名访问）都可以对该Object进行读操作（即下载）。

• private：只有Object的拥有者可以对该Object进行操作。

下面是RGW与Amazon S3的接口兼容列表。

表4-1　RGW与Amazon S3的接口兼容列表

特性	状态	备注
List Buckets	支持	
Delete Bucket	支持	
Create Bucket	支持	部分支持
Bucket Lifecycle	不支持	
Policy (Buckets, Objects)	不支持	支持 ACL
Bucket Website	支持	不支持
Bucket ACLs (Get, Put)	支持	
Bucket Location	支持	
Bucket Notification	不支持	
Bucket Object Versions	支持	
Get Bucket Info (HEAD)	支持	
Bucket Request Payment	不支持	
Put Object	支持	
Delete Object	支持	
Get Object	支持	
Object ACLs (Get, Put)	支持	
Get Object Info (HEAD)	支持	
POST Object	支持	
Copy Object	支持	
Multipart Uploads	支持	
CORS	支持	

注意：S3的兼容接口还在不断开发中，具体以官方分布的版本为准。

官方最新列表地址：http://docs.ceph.com/docs/master/radosgw/s3/#features-support 。

4.3.3　快速搭建RGW环境

radosgw的FastCGI可以支持多种类型的Web Server，如Apache2、Nginx等。Ceph从Hammer版本开始，在使用Ceph-deploy的情况下默认使用内置的civetweb替代旧版本的Apache2部署方式。官方文档主要介绍的是Apache2方式，但考虑到实际生产环境下的应用，本节将介绍civetweb和Nginx两种部署方式。这两种方式在功能上都没有区别，因此读者可以根据自己的实际情况进行选择。

1.civetweb方式

在使用Ceph-deploy部署好基本环境以后，在Ceph-deploy工作目录下会有一个{cluster-name}.bootstrap-rgw.keyring的文件（Hammer及以上版本才有），这个keyring包含了radosgw服务的初始

化keyring信息，下文中Ceph-deploy命令都将在这个文件所在的目录进行。

1）新建rgw。

```
ceph-deploy rgw create {gateway-node}
```

 注意

gateway-node为远程主机名，如果提示本地ceph.conf文件于远程不匹配，可以加上overwrite-conf参数覆盖远端配置。

2）修改RGW配置。

默认情况下，civetweb服务使用的是7480端口，可以通过修改ceph.conf并重启radosgw服务来改变默认端口，修改/etc/ceph/ceph.conf文件，添加如下内容。

```
[client]
rgw frontends = civetweb port=80
```

3）重启服务。

```
/etc/init.d/ceph-radosgw restart
```

有些操作系统下服务名字为radosgw，请根据实际情况进行处理。

4）检查服务是否启动。

```
root@demo# ps aux|grep radosgw
root      6336  0.1  1.5 2165224 15936 ?       Ssl  13:15
0:00 /usr/bin/radosgw -n client.rgw.demo
```

5）测试访问。

```
curl http://demo:7480
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner>
<ID>anonymous</ID><DisplayName></DisplayName></Owner>
<Buckets></Buckets></ListAllMyBucketsResult>
```

2.Nginx方式

使用civetweb方式部署比较简单，但是也存在一些问题，比如需要使用root用户启动服务，在一些安全性要求比较高的生产环境中不是很适用，因此笔者推荐使用Nginx作为前端服务。Nginx作为前端服务的好处就是Nginx配置简单灵活，第三方模块比较丰富，同时在资源消耗和性能方面也有一定优势。

（1）软件包安装

以下命令都是在需要安装radsogw的节点上运行的。

```
apt-get install nginx
apt-get intall radosgw
```

（2）新建radosgw用户和keyring

1）新建keyring。

```
sudo ceph-authtool --create-keyring
/etc/ceph/ceph.client.radosgw.keyring
sudo chmod +r /etc/ceph/ceph.client.radosgw.keyring
```

2）生成radosgw服务对应的用户和key信息，出于安全考虑，建议每个服务都可以对应一个用户和key。

```
sudo ceph-authtool /etc/ceph/ceph.client.radosgw.keyring -n
client.radosgw.gateway --gen-key
```

3）添加用户访问权限。

```
sudo ceph-authtool -n client.radosgw.gateway --cap osd
'allow rwx' --cap mon 'allow rwx'
/etc/ceph/ceph.client.radosgw.keyring
```

4）导入keyring。

```
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add
client.radosgw.gateway -i
/etc/ceph/ceph.client.radosgw.keyring
```

（3）新建radosgw对应的Pool

根据资源池列表，使用命令依次建立好radosgw所需的资源池，命令格式如下。

```
ceph osd pool create {poolname} {pg-num} {pgp-num}
```

 注意

poolname为资源池名词，pg-num和pgp-num相等，都是指对应的pool的PG数量。

资源池列表及部分资源池功能介绍如下。

- .rgw：region和zone配置信息。
- .rgw.root：region和zone配置信息。
- .rgw.control：存放notify信息。
- .rgw.gc：用于资源回收。
- .rgw.buckets：存放数据。

- .rgw.buckets.index：存放元数据信息。
- .rgw.buckets.extra：存放元数据扩展信息。
- .log：日志存放。
- .intent-log：日志存放。
- .usage：存放用户已用容量信息。
- .users：存放用户信息。
- .users.email：存放用户E-mail信息。
- .users.swift：存放swift类型的账号信息。
- .users.uid：存放用户信息。

使用命令确认资源池新建是否成功如下。

```
rados lspools
```

（4）ceph.conf配置

在/etc/ceph/ceph.conf中添加以下内容。

```
[client.radosgw.gateway]
 rgw dns name = {FQDN}
 rgw frontends = fastcgi
 host = {hostname}
 keyring = /etc/ceph/ceph.client.radosgw.keyring
 rgw socket path =  /var/run/ceph/ceph-client.radosgw.sock
 log file = {log_path}
 rgw print continue = false
 rgw content length compat = true
```

下面介绍一下ceph.conf中的一些关键参数。

• FQDN代表对外提供HTTP服务的主机域名全称，比如s3.ceph.work。

• hostname代表主机名，如demo。

• log_path代表日志存放路径。

• rgw content length compat=true，启动对http header中CONTENT字段的兼容性检查。

• rgw print continue=false，关闭100-continue特性支持。

其他配置参数请参考：http://docs.ceph.com/docs/master/radosgw/config-ref/ 。

（5）配置radosgw服务开机启动

```
mkdir -p /var/lib/ceph/radosgw/ceph-radosgw.gateway
touch /var/lib/ceph/radosgw/ceph-radosgw.gateway/done
touch /var/lib/ceph/radosgw/ceph-radosgw.gateway/sysvinit
```

（6）Nginx配置

编辑/etc/nginx/conf.d/default.conf，内容如下。

```
server {
    listen       80;
    server_name  s3.ceph.work;
    client_max_body_size 0;
    location / {
    fastcgi_pass_header Authorization;#启用对
```

HTTP header认证信息转发

```
    fastcgi_pass_request_headers on;
    fastcgi_param QUERY_STRING  $query_string; #HTTP  header
```
字段的转发

```
    fastcgi_param REQUEST_METHOD $request_method;#HTTP
```
方法字段的转发

```
    fastcgi_param CONTENT_LENGTH $content_length;#HTTP
```
内容长度字段的转发

```
    fastcgi_param CONTENT_TYPE $content_type; #HTTP 内容格式字段
的转发
```

```
    fastcgi_param HTTP_CONTENT_LENGTH $content_length; #HTTP
内容长度字段的转发
```

```
    if ($request_method = PUT) {
        rewrite ^ /PUT$request_uri; #转发
```

PUT请求

```
        }
        include fastcgi_params;
        fastcgi_pass unix:/var/run/ceph/ceph-
    client.radosgw.gateway.sock;
        }
        location /PUT/ {
        internal;#只允许内部跳转访问
```

```
        fastcgi_pass_header Authorization;
        fastcgi_pass_request_headers on;
        include fastcgi_params;
        fastcgi_param QUERY_STRING  $query_string;
        fastcgi_param REQUEST_METHOD $request_method;
        fastcgi_param CONTENT_LENGTH $content_length;
        fastcgi_param CONTENT_TYPE $content_type;
        fastcgi_param HTTP_CONTENT_LENGTH $content_length;
        fastcgi_pass unix:/var/run/ceph/ceph-
    client.radosgw.gateway.sock;
        }
    }
```

注意

如果需要配置SSL，可以参考网上的Nginx SSl配置文档资料。

（7）服务启动与检查

1）启动radosgw服务。

```
/etc/init.d/radosgw start
```

2）检查radosgw服务。

```
root@gateway-node# ps aux|grep radosgw
root   6336  0.1  1.5 2165224 15936 ?  Ssl  13:15  0:00
/usr/bin/radosgw -n client.rgw.demo
```

3）重新加载nginx配置。

```
/etc/init.d/nginx reload
```

4）使用curl命令测试服务状态，正常会看到下面的内容。

```
curl http://gateway-node
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner>
<ID>anonymous</ID><DisplayName></DisplayName></Owner>
<Buckets></Buckets></ListAllMyBucketsResult>
```

（8）使用radosgw-admin管理用户

radosgw-admin是整个radosgw服务的命令行管理工具，提供了强大的用户和Bucket等资源的管理功能。下面用这个工具新建一个S3用户，稍后会用到这个用户的信息进行访问。

1）新建S3用户。

```
root@demo:~# radosgw-admin user create --uid=demo --display-
name=demo --email=demo.ceph.com
{
    "user_id": "demo",
    "display_name": "demo",
    "email": "demo.ceph.com",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "demo",
            "access_key": "Z2OJ3XTSI7Z1DFY3FDMZ",
            "secret_key":
"C0NxHuBwFRVm0mqAttH16jDVwckPXb3JqvwTV1tA"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
```

```
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

2）新建的用户demo，记录对应的信息，之后会用到这些信息访问S3服务，下面是需要记录的内容。

```
access_key: “



Z2OJ3XTSI7Z1DFY3FDMZ”



secret_key: “
```

```
C0NxHuBwFRVm0mqAttH16jDVwckPXb3JqvwTV1tA"
```

3）删除用户。

```
radosgw-admin user rm --uid=demo
```

4）其他命令请使用帮助命令。

```
radosgw-admin --help
```

（9）DNS泛域名解析的支持

当用户访问对应的Bucket数据时，大部分情况下都需要借助域名解析，特别是很多第三方S3客户端必

须要将对应的Bucket资源关联到具体的域名上，才能正常访问S3服务，因此建议有条件的用户自己搭建一个DNS泛域名解析环境，或者使用一些第三方的域名解析服务。如果只是用于测试，可以使用修改/etc/hosts文件的方式，添加相应的Bucket解析记录。

1）修改/etc/hosts配置方法如下。

其中，bucket名称对应bucket1~bucket99，提供S3服务的主机域名为s3.ceph.work，对应IP为10.0.2.15。

```
10.0.2.15   s3.ceph.work
10.0.2.15   bucket1.s3.ceph.work
...
10.0.2.15   bucket99.s3.ceph.work
```

2）使用第三方DNS解析服务。

笔者使用dnspod的域名解析服务，添加了一条泛解析记录，用于匹配所有域名*.s3.ceph.work，关于泛域名解析记录的添加，请参考具体的服务提供商的说明文档。

4.3.4　RGW搭建过程的排错指南

在radosgw搭建过程中，如果有Web Server部署经验，特别是FastCGI相关的配置经验，会比较容易理解这套模型。

数据访问流程：Nginx收到来自Client的请求后将其转换成标准输出到FastCGI，之后由FastCGI程序通过调用librados接口完成数据的写入操作。数据读取操作则刚好相反。作者总结自身经验，绘出如下流程图，如图4-7所示。

了解数据访问的大致流程以后，可根据具体报错情况，分阶段进行排错：一般排错从最上面的Nginx入手，逐步深入底层模型。

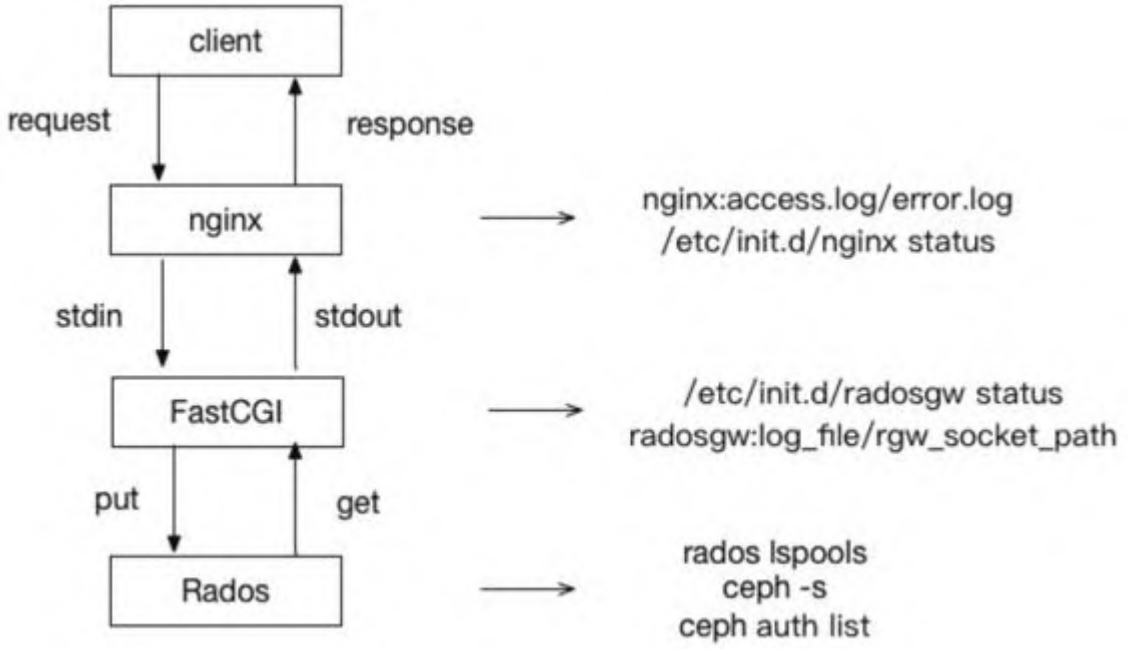

图4-7　RGW排错流程

1）Nginx层面：检查Nginx服务进程的状态，Nginx访问对应的端口是否打开，服务端口是否有防火墙设置，同时通过Nginx的access.log和error.log获取来自Client的请求处理情况。

2）FastCGI层面：检查radosgw服务是否启动，同时检查ceph.conf中的rgw_socket_path对应的sock文件是否生成，Nginx服务的用户是否有权限访问这个

sock文件，最后根据radosgw服务中配置的log，查看具体报错信息。

3）Rados层面：一般情况下radosgw服务无法启动都和这部分的配置有关，需要检查rados对应的pool是否生成，Ceph集群的健康状态，同时对应radosgw服务启动所需的用户和keyring文件是否建立等。

这里也只是笔者的一些经验分享，纯属抛砖引玉。读者在配置radsogw服务的过程中可能都会有一些环境差异，还需要结合实际情况进行相应处理。

下面是HTTP错误状态码列表，方便大家排错。

表4-2　HTTP常见错误状态码

Message	HTTP Status	描述
AccessDenied	403	拒绝访问
BadDigest	400	错误的摘要
BucketAlreadyExists	409	Bucket 已经存在
BucketAlreadyOwnedByYou	409	用户已经是 Bucket 的拥有者
BucketNotEmpty	409	Bucket 不为空

（续）

Message	HTTP Status	描述
InternalError	500	内部错误
InvalidAccessKey	403	无效的 AccessKey
InvalidACLString	400	ACL 配置无效
InvalidAuthorizationString	400	无效的验证字符串
InvalidBucketName	400	无效的 Bucket 名称
InvalidDateFormat	400	无效的日期格式
InvalidDigest	400	无效的摘要
InvalidEncryptionAlgorithm	400	无效的指定加密算法
InvalidHostHeader	400	无效的头信息
InvalidParameter	400	无效的参数
InvalidPath	400	无效的路径
InvalidQueryString	400	无效的请求字符串
InvalidRange	416	无效的 range
KeyTooLong	400	Key 太长
MetadataTooLarge	400	metadata 过大
MethodNotAllowed	405	不支持的方法
MissingDateHeader	400	头信息中缺少 data
MissingHostHeader	400	头信息中缺少 host
NoSuchBucket	404	该 Bucket 不存在
NoSuchKey	404	该 Key 不存在
NotImplemented	501	无法处理的方法
RequestTimeTooSkewed	403	发起请求的时间和服务器时间超出 15 分钟
SignatureDoesNotMatch	403	签名不匹配
TooManyBuckets	400	用户的 Bucket 数目超过限制
URLExpired	403	url 过期
BadParams	400	参数错误
ImageTypeNotSupport	400	图片类型不支持
MissingFormArgs	400	没有上传 Policy
ContentRangeError	400	Range 错误
ContentLengthOutOfRange	400	上传文件内容大于 range
PolicyError	400	Policy 错误
ExpirationError	400	Policy 中没有 expiration
FormUnmatchPolicy	400	表单中的内容和 policy 不匹配

4.3.5　使用S3客户端访问RGW服务

搭建好RGW服务以后，很多读者肯定想知道如何访问RGW服务，下面介绍3种常用的客户端工具。

1.s3cmd命令行工具的使用

s3cmd是一套基于Amazon S3接口封装的命令行管理工具，可以很方便地实现对Amazon S3资源的访问与管理。

（1）客户端安装

- pip安装方式。

pip是Python常用的包管理工具，能够非常方便地完成一些常用Python包的安装。

```
pip install s3cmd
```

• Debian/Ubuntu系列操作系统下的安装方式。

如果读者使用的是Debian/Ubuntu一类的操作系统，可以使用以下命令进行安装。

```
apt-get install s3cmd
```

其他方式请参考官方文档：http://s3tools.org/s3cmd 。

（2）客户端配置

软件包安装完成以后，需要进行一些基本设置，下面使用命令行向导进行设置。

```
root@demo:~# s3cmd --configure
Enter new values or accept defaults in brackets with Enter.
Refer to user manual for detailed description of all
options.
Access key and Secret key are your identifiers for Amazon
S3. Leave them empty for using the env variables.
Access Key: Z2OJ3XTSI7Z1DFY3FDMZ #配置
```

```
Access Key
Secret Key: C0NxHuBwFRVm0mqAttH16jDVwckPXb3JqvwTV1tA #配置
```

```
Secret Key
Default Region [US]:
Encryption password is used to protect your files from
reading
by unauthorized persons while in transfer to S3
Encryption password:
Path to GPG program [/usr/bin/gpg]:
When using secure HTTPS protocol all communication with
Amazon S3
servers is protected from 3rd party eavesdropping. This
method is
slower than plain HTTP, and can only be proxied with Python
2.7 or newer
Use HTTPS protocol [Yes]: No #根据实际情况选择是否使用
```

```
HTTPS
On some networks all internet access must go through a HTTP
proxy.
Try setting it here if you can't connect to S3 directly
HTTP Proxy server name:
New settings: #确认配置信息
```

```
  Access Key: Z2OJ3XTSI7Z1DFY3FDMZ
  Secret Key: C0NxHuBwFRVm0mqAttH16jDVwckPXb3JqvwTV1tA
  Default Region: US
  Encryption password:
  Path to GPG program: /usr/bin/gpg
  Use HTTPS protocol: False
  HTTP Proxy server name:
  HTTP Proxy server port: 0
Test access with supplied credentials? [Y/n] n
Save settings? [y/N] y
Configuration saved to '/root/.s3cfg'
```

（3）编辑上一步生成的/root/.s3cfg文件

需要调整内容如下，用户也可以跳过上一步，自己手工生成，默认是在用户的home目录下生成.s3cfg文件。

```
[default]
access_key = Z2OJ3XTSI7Z1DFY3FDMZ
check_ssl_certificate = True
check_ssl_hostname = True
default_mime_type = binary/octet-stream
enable_multipart = True
encoding = UTF-8
encrypt = False
host_base = s3.ceph.work
host_bucket = %(bucket)s.s3.ceph.work
multipart_chunk_size_mb = 15
```

```
secret_key = C0NxHuBwFRVm0mqAttH16jDVwckPXb3JqvwTV1tA
socket_timeout = 300
stop_on_error = False
use_https = False
use_mime_magic = True
verbosity = WARNING
```

（4）使用s3cmd客户端连接并管理S3资源

1）新建Bucket。

```
root@demo:~# s3cmd mb s3://s3test1
Bucket 's3://s3test1/' created
```

2）查看现有Bucket。

```
root@demo:~# s3cmd ls
2015-12-24 09:30  s3://bucket1
2015-12-24 09:30  s3://s3test1
```

3）删除Bucket。

```
root@demo:/tmp# s3cmd rb s3://s3test1
Bucket 's3://s3test1/' removed
```

4）上传Objcet。

```
root@demo:~# s3cmd put default.conf s3://bucket1
'default.conf' -> 's3://bucket1/default.conf'  [1 of 1]
 1573 of 1573   100% in    1s  1257.71 B/s  done
'default.conf' -> 's3://bucket1/default.conf'  [1 of 1]
 1573 of 1573   100% in    0s    19.89 kB/s  done
```

5）查看Objcet。

```
root@demo:~# s3cmd ls s3://bucket1
2015-12-24 09:31      1573   s3://bucket1/default.conf
```

6）下载Objcet。

```
root@demo:~# cd /tmp
root@demo:/tmp# s3cmd get s3://bucket1/default.conf
's3://bucket1/default.conf' -> './default.conf'  [1 of 1]
's3://bucket1/default.conf' -> './default.conf'  [1 of 1]
 1573 of 1573   100% in    0s    33.25 kB/s  done
```

2.GUI客户端的使用

支持S3的GUI客户端比较丰富，配置也比较简单，这里不再详细说明，只是将几个笔者经过兼容性测试觉得比较好的客户端列举一下。

• cyberduck：开源，支持Windows、Mac OS X and Linux，下载地址https://cyberduck.io/ 。

• dragondisk：支持Windows、Mac OS X and Linux，下载地址http://www.dragondisk.com/ 。

• Explorer for Amazon S3：分收费版和免费版，只支持Windows，下载地址 http://www.cloudberrylab.com/free-amazon-s3-explorer-cloudfront-IAM.aspx 。

3.Python语言的SDK用例

使用Python的boto库可以实现对S3的访问。

（1）安装boto

1）pip方式安装。

```
pip install boto
```

2）debian/ubuntu系统。

```
sudo apt-get install python-boto
```

3）CentOS系列。

```
sudo yum install python-boto
```

（2）Python用例

```
import boto
import boto.s3.connection
import sys
access_key =  'Z2OJ3XTSI7Z1DFY3FDMZ'
secret_key = 'C0NxHuBwFRVm0mqAttH16jDVwckPXb3JqvwTV1tA'
keyname='test_key'
testfile = '/tmp/test.file'
def percent_cb(complete, total):
    sys.stdout.write('.')
    sys.stdout.flush()
conn = boto.connect_s3(
        aws_access_key_id = access_key,
        aws_secret_access_key = secret_key,
        host = 's3.ceph.work',
        is_secure=False,
        calling_format =
boto.s3.connection.OrdinaryCallingFormat(),
        )
#新建
```

bucket，并列出所有创建的

```
bucket
bucket = conn.create_bucket('bucket1')
for bucket in conn.get_all_buckets():
        print "{name}\t{created}".format(
                name = bucket.name,
                created = bucket.creation_date,
                )
#新建
```

key，并设置相应

key的权限为

```
'public-read'
key = bucket.new_key(keyname)
key.set_contents_from_filename(testfile,cb=percent_cb,num_cb
=10)
print key.get_acl()
key.set_canned_acl('public-read')
print key.get_acl()
```

4.3.6 admin管理接口的使用

除了使用radsogw-admin命令行工具对RGW服务进行关联以外，RGW正提供一套RESTful接口，同样可以实现对S3用户、Bucket、quota等信息的管理。笔者在实际工作中也接触过很多基于这套接口的开发需求，下面向各位读者分享一下这套接口的一些开发经验。

1.新建admin账号

所有对admin管理接口的访问都需要通过一个S3内置账号来完成，同时还需要设置该账号的权限，具体操作如下。

```
root@demo:~# radosgw-admin user create --uid=admin --
display-name=admin
{
    "user_id": "admin",
    "display_name": "admin",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
```

```
    "subusers": [],
    "keys": [
        {
            "user": "admin",
            "access_key": "173QS7OP0FDJEW798PGM",
            "secret_key":
"MWQ3j76waSbUNSeXf6Fqvcr94SuEbWk8cM5lLFjM"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

---

2. 添加用户admin权限

设置admin账号的用户权限，允许其读取、修改users信息。

```
root@demo:~# radosgw-admin caps add --uid=admin --
caps="users=*"
{
    "user_id": "admin",
    "display_name": "admin",
```

```
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "admin",
            "access_key": "173QS7OP0FDJEW798PGM",
            "secret_key":
"MWQ3j76waSbUNSeXf6Fqvcr94SuEbWk8cM5lLFjM"
        }
    ],
    "swift_keys": [],
    "caps": [
        {
            "type": "users",
            "perm": "*"
        }
    ],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
#添加对
```

admin用户的对所有

usage信息的读写权限

```
root@demo:~# radosgw-admin caps add --uid=admin --
caps="usage=read,write"
{
    "user_id": "admin",
    "display_name": "admin",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "admin",
            "access_key": "173QS7OP0FDJEW798PGM",
            "secret_key":
"MWQ3j76waSbUNSeXf6Fqvcr94SuEbWk8cM5lLFjM"
        }
    ],
    "swift_keys": [],
    "caps": [
        {
            "type": "usage",
            "perm": "*"
        },
        {
            "type": "users",
            "perm": "*"
```

```
            }
        ],
        "op_mask": "read, write, delete",
        "default_placement": "",
        "placement_tags": [],
        "bucket_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        },
        "user_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        },
        "temp_url_keys": []
    }
```

3. 安装Python库文件

使用admin接口同样也需要通过S3的权限认证，笔者推荐使用以下Python库来实现，安装方法如下。

```
pip install request-aws
```

其他下载地址：https://pypi.python.org/pypi/requests-aws/0.1.4。

4. 新建及查询用户信息

新建s3admin.py文件，Python源码如下。

```
import requests
from awsauth import S3Auth
aws_key = '173QS7OP0FDJEW798PGM'
secret = "MWQ3j76waSbUNSeXf6Fqvcr94SuEbWk8cM5lLFjM"
server = 's3.ceph.work'
#新建用户




s3user1
url = 'http://%s/admin/user?format=json&uid=s3user1&display-
name=s3user1' % server
r = requests.put(url, auth=S3Auth(aws_key, secret, server))
print 'HTTP Status Code = %s' % r.status_code
#获取用户信息




url = 'http://%s/admin/user?format=json&uid=s3user1' %
server
r = requests.get(url, auth=S3Auth(aws_key, secret, server))
result = r.content
print result
```

5. 执行Python脚本

```
root@demo:~# python s3admin.py
HTTP Status Code = 200
{"user_id":"s3user1","display_name":"s3user1","email":"","su
spended":0,"max_buckets":1000,"subusers":[],"keys":
[{"user":"s3user1","access_key":"KO8A420OF63IJF803G1A","secr
et_key":"cMHmvSw22CaqdL6YMvlvsOuHlQVop86LumcuLij7"}],"swift_
keys":[],"caps":[]}
```

其他接口使用用例请参考：

http://docs.ceph.com/docs/master/radosgw/adminops/ 。

4.4 本章小结

本章介绍了Ceph的3大存储访问类型，以及它们的简单介绍、部署、使用与使用场景。通过本章学习，读者可以根据需求来选择和设计存储方案。

第5章　可视化管理Calamari

5.1　认识Calamari

Calamari是一个管理和监控Ceph集群的Web平台，提供了漂亮的管理和监控界面，还提供了一套RESTful API接口，目的是为了简化Ceph集群管理。起初，Calamari是InkTank公司的一款商业软件，是InkTank向其客户提供的Ceph企业版的产品。在InkTank被Red Hat收购之后，Red Hat在2014年5月30日将其开源。

Calamari Rest API文档：http://calamari.readthedocs.org/en/latest/calamari_rest/index.html 。

5.2 安装介绍

本书是以在CentOS 7.x版本的操作系统上使用源码编译、安装Calamari为例，并且假设已经搭建好了一套可用的Ceph集群。

如何在Ubuntu操作系统上安装Calamari，请参见官方链接：

```
http://calamari.readthedocs.org/en/latest/development/index.html
http://calamari.readthedocs.org/en/latest/operations/index.html
```

Ceph官方目前提供了在Ubuntu14.04操作系统上安装Calamari的deb软件包，链接如下。

```
http://download.ceph.com/calamari/
```

Calamari安装包括calamari-server、calamari-client和diamond三部分。

calamari-server提供平台管理服务，使用了SaltStack管理客户端。

calamari-client（现在改名为romana）是一个HTTP模块，为客户端使用Calamari API提供服务。

diamond是一个Python进程，用来收集Ceph存储节点上的集群数据和系统信息并发送给Graphite，每个Ceph节点都需要安装diamond。

5.2.1 安装calamari-server

1.calamari-server介绍

calamari-server是Calamari服务端，提供Calamari RESTful API，并且使用SaltStack管理Ceph节点。

calamari-server包含的组件有：Apache、salt-master、Graphite/carbon-cache、cthulhu和supervisord。

各个组件的功能如下。

1）Apache为Calamari提供Web服务。

2）Calamari服务端安装有salt-master，salt-master是salt的服务管理端，salt-minion是被管理

端。

3）Calamari服务端运行Graphite服务，Graphite是一个非常好的监控和绘图工具。Graphite后端运行一个名为carbon-cache.py的Python程序，负责处理客户端节点上的业务数据。Graphite/carbon配置文件位于/etc/graphite/carbon.conf文件中。

Calamari安装好以后，还可以直接通过URLhttp://172.16.*.*/graphite/dashboard/ [1] 访问Graphite。

Ceph Server与calamari-server之间的消息和事件通信是通过salt实现的，也就是运行在Ceph Server上的salt-minion与运行在Calamari Server上的salt-master之间的通信。

4）cthulhu是监听Ceph Server与Calamari Server之间通信的，当监听到（例如crushmap）有变化时，通过salt的通信通道向Mon发起执行一个job任务，检索crushmap信息然后同步信息。cthulhu也可以执行一些librados操作。

5）supervisord是一个允许用户监控和控制进程数量的系统程序。它可以指定一个服务如何运行，在Calamari服务端可以查看到supervisord的配置文件/etc/supervisord.conf，在配置文件最后可以看到关于calamari-server相关服务的运行方式，包含carbon-cache.py的运行方式，如下。

```
### START calamari-server ###
[program:carbon-cache]
command=/opt/calamari/venv/bin/carbon-cache.py --debug --
config /etc/graphite/carbon.conf start
[program:cthulhu]
command=/opt/calamari/venv/bin/cthulhu-manager
### END calamari-server ###
```

2.calamari-server安装

针对calamari-server的安装有些是通过vagrant结合virtualbox的方式构建rpm软件包安装的，这样略显麻烦，其实直接下载源码构建rpm即可。下面操作以从源码直接构建rpm包为例。

（1）从Github上获取Calamari源码

```
# git clone https://github.com/ceph/calamari.git
```

（2）构建rpm安装包

```
# yum install gcc gcc-c++ postgresql-libs python-virtualenv
# yum install postgresql-devel httpd checkpolicy
# yum install selinux-policy-devel selinux-policy-doc
selinux-policy-mls
```

如果系统更新过（yum update），安装包时可能会出现软件包版本问题（如图5-1所示），只需要把相关的软件包版本降低为需要的软件包版本即可，例如，

```
# yum downgrade postgresql-libs-9.2.13-1.el7_1
```

```
--> Finished Dependency Resolution
Error: Package: postgresql-devel-9.2.13-1.el7_1.x86_64 (updates)
           Requires: postgresql-libs(x86-64) = 9.2.13-1.el7_1
           Installed: postgresql-libs-9.2.14-1.el7_1.x86_64 (@updates)
               postgresql-libs(x86-64) = 9.2.14-1.el7_1
```

图5-1　安装postgresql报错

```
# cd calamari
# ./build-rpm.sh
```

构建完成的rpm安装包位于上一级目录下，如下所示。

```
# cd ../rpmbuild/RPMS/x86_64/
# yum install calamari-server-1.3.1.1-
105_g79c8df2.el7.centos.x86_64.rpm
```

（3）初始化Calamari

```
# calamari-ctl initialize
......
Username (leave blank to use 'root'): root
Email address: 123346@xx.com
Password:
Password (again):
Superuser created successfully.
[INFO] Initializing web interface...
[INFO] Starting/enabling services...
```

```
[INFO] Restarting services...
[INFO] Complete.
```

上面目录会要求输入登录Web管理平台的账户和密码，还可以修改密码，方法如下。

```
# calamari-ctl change_password --password {password} {user-
name}
```

[1] *表示隐去的部分，此部分读者可根据实际情况来替换和补充。本书后文还会出现这样的处理，不一一指出。

5.2.2 安装romana（calamari-client）

1.romana介绍

romana是由之前的calamari-client改名而来，它是一个提供Web UI的模块，主要为客户端使用Calamari API提供服务，安装在Calamari Server端。calamari-client由salt-minion和diamond组成。

romana包括dashboard、login、admin和manage四大模块，构建rpm软件包时，这些模块缺一不可。

1）dashboard模块是一个JavaScript的客户端，直接与Ceph RESTful API交互来管理Ceph。dashboard包含3个逻辑部分，分别为Dashboard、Workbench和Graphs。

• Dashboard是一个只读的视图，负责展现Ceph集群的健康状态。

• Workbench是后台OSD和Host的虚拟展现，同时最多限制展现256个OSD。

• Graphs是由负责展示图形的Graphite和负责在每个节点收集数据的diamond共同展示各种度量数据的视图。

2）login模块用于登录Web页面，模块信息位于别名为/login的URL下，实际上被重定向到后端/opt/calamari/webapp/content/login/目录。

3）admin模块是用来管理用户和Calamari信息的管理工具，采用Angular 1.x语言开发。由于admin模块功能很简单，在Calamari-server 1.2和目前的1.3

版本中admin模块已经被禁用，如果加入了user和role管理功能会重新启用该模块。

4）manage模块是用于管理Ceph集群中各种应用，如OSD管理、Pool管理、集群设置和集群日志展示等功能。目前采用Angular 1.2.x和Bootstrap3开发。

2.romana安装

（1）从Github上获取romana源码

```
# git clone https://github.com/ceph/romana
```

（2）安装依赖包

```
# yum install npm ruby  ruby-devel rubygems rpm-build
libpng-devel
```

（3）更新npm并安装相关软件包

1）把官方npm源修改为淘宝的npm源。

```
# npm config set registry https://registry.npm.taobao.org
```

2）验证npm源是否修改成功。

```
# npm config get registry
# npm install -g npm
# npm install -g bower grunt grunt-cli
# npm install -g grunt-contrib-compass@0.6.0
```

（4）使用Gem安装compass

1）把官方Ruby源修改为taobao的Ruby源。

```
# gem sources --remove https://rubygems.org/
# gem sources -a https://ruby.taobao.org/
```

2）更新gem并安装compass。

```
# gem update --system
# gem install compass
```

（5）编译romana

1）进入romana目录。

```
# cd romana
```

2）修改依赖的软件包版本。

```
# vi manage/package.json……
```

```
"grunt-connect-proxy": "~0.2.0",……
```

3）编译软件包。

```
# make build-real
# make dist
```

4）编译完成后会在上一级目录生成romana_1.2.2.tar.gz包，把该软件包复制到当前目录。

```
# cp ../romana_1.2.2.tar.gz ./
```

5）修改Makefile文件。

```
# vi Makefile
BUILD_PRODUCT_TGZ=$(SRC)/romana-build-output.tar.gz修改为：




BUILD_PRODUCT_TGZ=$(SRC)/romana_1.2.2.tar.gz
```

6）修改romana.spec文件，在“tar xfz%{tarname}”下面添加如下部分。

```
cd %{name}-%{version}
for dir in manage admin login dashboard
do
mkdir -p ../opt/calamari/webapp/content/"$dir"
```

```
cp -pr "$dir"/dist/* ../opt/calamari/webapp/content/"$dir"/
done
cd ../
rm -rf /tmp/%{name}-%{version}
mv %{name}-%{version} /tmp/
```

7）构建rpm包。

```
# make rpm……
```

```
Wrote: /root/rpmbuild/RPMS/x86_64/romana-1.2.2-
36_gc62bb5b.el7.centos.x86_64.rpm
Executing(%clean): /bin/sh -e /var/tmp/rpm-tmp.MntCPR
+ umask 022
+ cd /root/calamari/romana/../rpmbuild/BUILD
+ echo clean
clean
+ '[' /root/rpmbuild/BUILDROOT/romana-1.2.2-
36_gc62bb5b.el7.centos.x86_64 '!=' / ']'
+ rm -rf /root/rpmbuild/BUILDROOT/romana-1.2.2-
36_gc62bb5b.el7.centos.x86_64
+ exit 0
```

8）构建完成的rpm包位于上一级目录下并安装romana。

```
# cd .. /rpmbuild/RPMS/x86_64/
# yum install romana-1.2.2-36_gc62bb5b.el7.centos.x86_64.rpm
```

5.2.3 安装diamond

diamond是一个Python进程，负责收集Ceph集群节点上系统度量数据，它可以收集例如Ceph节点的CPU、内存、网络、磁盘IO、平均负载和磁盘使用量等信息，然后把收集到的信息发送给Graphite绘制图表。diamond还提供了API可以让用户自定义收集数据的方式并且支持多种数据源。

（1）从Github上获取diamond源码

```
# git clone https://github.com/ceph/Diamond
```

（2）构建rpm包

```
# cd Diamond
# git checkout origin/calamari
# make rpm
```

构建完成的rpm包位于Diamond/dist/diamond-3.4.67-0.noarch.rpm目录，把构建完成的rpm包复制到所有Ceph节点并安装。

（3）在Ceph节点安装diamond

```
# yum install diamond-3.4.67-0.noarch.rpm
```

（4）修改diamond配置文件，指向Graphite server所在的主机

```
# vi /etc/diamond/diamond.conf
  ……




  # Graphite server host
  host = ceph1
  ……
```

把host值修改为calamari-server所在的主机名，并在/etc/hosts目录中添加主机解析。

（5）重启diamond服务

```
# /etc/init.d/diamond restart
```

5.2.4　安装salt-minion

salt-minion是被管理端，所有的Ceph节点都需要安装salt-minion。

注意

如果在登录Calamari页面时报500错误，那么可能是由于salt版本太高导致不兼容，使用2014.1.13-1版本的软件包手动安装salt、salt-master、salt-minion便可解决该问题。相关版本的软件包可以在 http://rpmfind.net 上搜索并下载。

（1）安装salt-minion

```
# yum install salt-minion
```

（2）创建配置文件

```
# mkdir /etc/salt/minion.d/
# vi /etc/salt/minion.d/calamari.conf
  master: ceph1
```

ceph1是saltstack服务端主机名，saltstack服务端与calamari-server同在一台服务器上。

（3）重启salt-minion服务

```
# systemctl restart salt-minion
```

（4）在calamari-server上查看客户端认证请求并认证

```
# salt-key -




L
# salt-key -A
```

5.2.5 重启服务

配置Calamari日志权限并重启相关服务。

```
# chmod 777 -R /var/log/calamari/*
# systemctl restart supervisord
# systemctl restart httpd
```

Httpd服务启动好后，就可以登录Calamari管理平台了，登录操作请见5.3.1节的介绍。

5.3　基本操作

5.3.1　登录Calamari

Calamari管理平台默认使用80端口，直接在浏览器中输入IP地址即可。

```
http://172.16.*.*
```

用之前设置的用户名和密码登录，默认进入DASHBOARD页面，如图5-2所示。

DASHBOARD页面中由上到下展示结果说明如下。

• HEALTH集群健康状态，图中表示集群状态为HEALTH_OK。

• OSD数量及状态，图中显示集群共有6个OSD并且全部为工作状态。

• MONITORS数量及状态，图中显示集群共有3个Monitor并且全部为工作状态。

• POOLS存储池数量和状态，显示集群中共有5个存储池。

• PG（Placement Group）数量和状态。

• IOPS统计，统计了整个集群的磁盘读写情况。

• USAGE：集群资源使用状态，图中显示了集群资源总量及使用量。

• HOSTS：集群主机数量和存活状态。

图5-2　DASHBOARD页面

5.3.2　WORKBENCH页面

打开WORKBENCH页面，如图5-3所示。

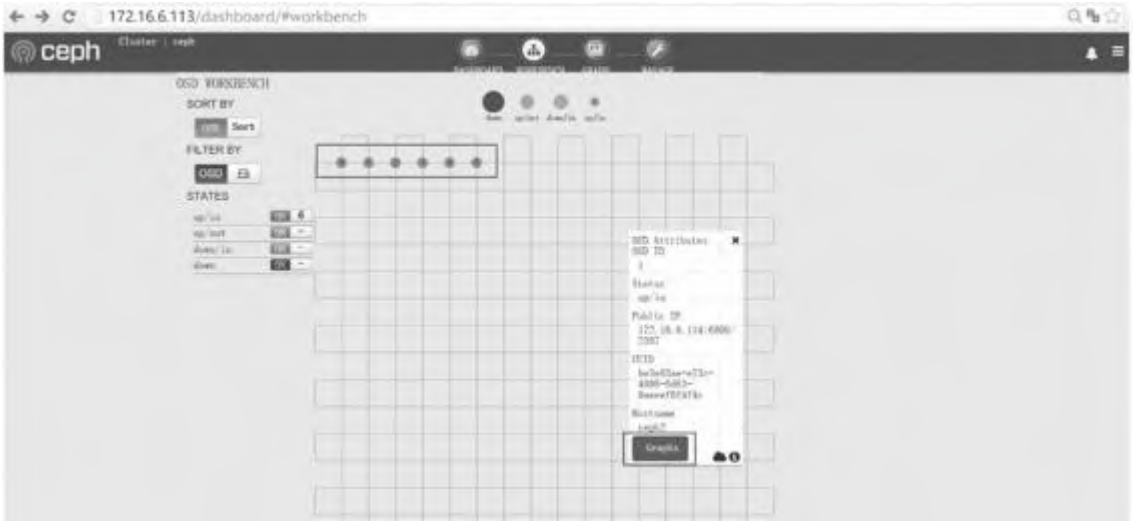

图5-3　WORKBENCH页面

Ceph集群中共有6个OSD，在图中显示为0~5，单击每个OSD可以显示出OSD的状态信息、PG数量及状态、存储池等信息。

单击图5-3中“Graphs”按钮，可以显示出OSD所在主机的CPU、平均负载和内存统计信息，如图5-4所

示。

图5-4　Ceph主机资源使用情况

5.3.3　GRAPH页面

打开GRAPH页面（如图5-5所示），这个页面中主要以展示为主，比如集群中存储池的IOPS统计情况、存储池中Disk空间的使用情况等，根据页面上的过滤条件也可以查看每台Ceph节点的CPU、平均负载和内存统计数据。

图5-5　GRAPH页面

5.3.4　MANAGE页面

打开MANAGE页面（如图5-6所示），这个页面的作用主要是对主机、集群进行配置以及对集群配置参数进行查看，还可以进行创建、配置、查看和删除存储池或存储池属性信息、查看集群日志等操作。

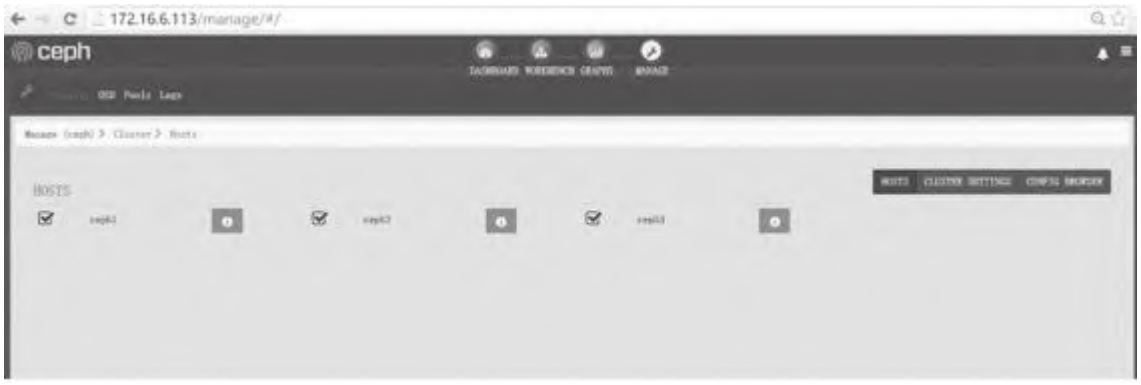

图5-6　MANAGE页面

查看已创建的存储池（Pool）信息，如图5-7所示。

图5-7　存储池信息

修改存储池的副本数，如图5-8所示。

创建存储池，如图5-9所示。

查看集群Log，如图5-10所示。

图5-8　修改存储池副本数

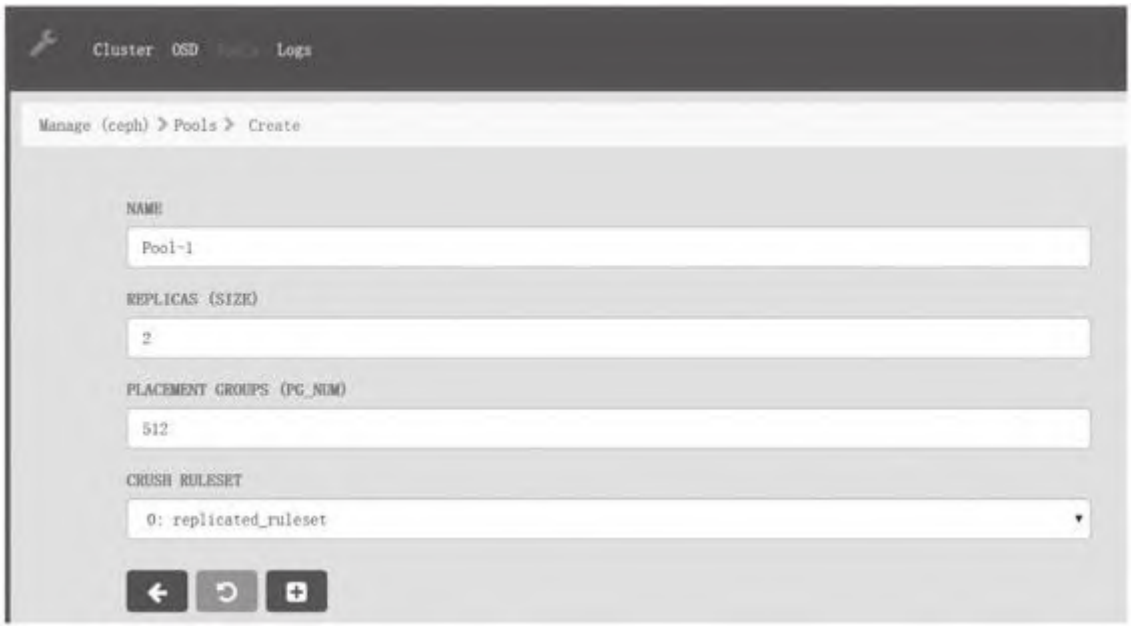

图5-9　创建存储池

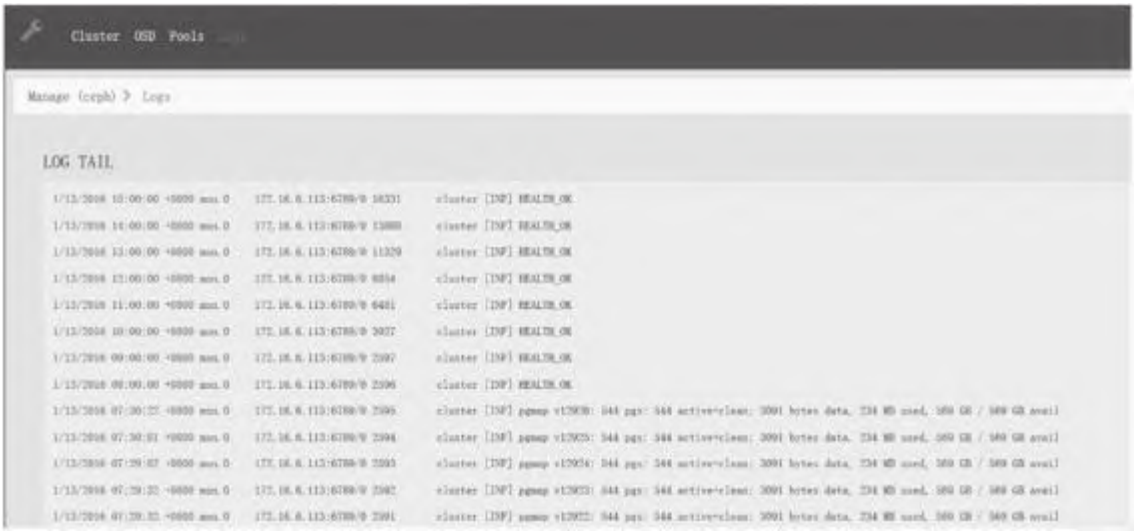

图5-10　集群Log

Calamari管理界面上对Ceph集群的操作还算丰富，基本上可以覆盖日常维护管理，监控和集群信息的统计也很直观。

5.4 本章小结

Calamari包括前端和后端两个部分，每个部分都有各自的代码库。

前端（Frontend）：这是一个基于浏览器的用户界面，主要用JavaScript实现。前端的实现利用的是Calamari的RESTful API，并且遵循模块化方法构建。因此，前端的每个组件都可以独立更新和维护。Calamari前端的代码以MIT license开源，可以从下面的链接获取它代码：https://github.com/ceph/calamari-clients 。

后端（Backend）：后端是Calamari的核心部分，用Python实现。Calamari后端利用一些组件如SaltStack、ZeroRPC、Graphite、Django-rest-framework、Django和gevent等实现。它提供了一组新

的RESTful API，用于和Ceph以及其他系统之间进行集成。Calamari对新版本进行了重新开发，利用这些新的RESTful API来完成与Ceph集群之间的交互。Calamari此前版本中使用的是Ceph的RESTful API，在其功能的实现上有一些局限性。Calamari的后端代码在LGPL2+license下开源，可以通过 https://github.com/ceph/calamari 获取其源码。

Calamari的文档（http://calamari.readthedocs.org ）很完善。无论是对于Calamari的使用者、开发者或者要利用Calamari RESTful API进行开发的人员，Calamari文档都是学习Calamari的一个好选择。像Ceph一样，Calamari也是被作为一个上游产品来开发的。你可以通过Calamari的IRC频道irc：//irc.oftc.net/ceph、注册邮件列表ceph-calamari@ceph.com或者在GitHub上发送pull request来参与Calamari的开发。

参考资料

[1]http://calamari.readthedocs.org 。

[2]https://zphj1987.gitbooks.io/calamaribook/content/index.html 。

第6章　文件系统——高性能计算与大数据

6.1　Ceph FS作为高性能计算存储

根据前面的学习，已经安装了MDS组件，测试了Ceph FS挂载。

Ceph FS有两种挂载方式：一种是通过Ceph FS内核模块（Kernel Module），另外一种是通过Ceph-Fuse用户空间（User Space），下面分别介绍这两种挂载方式。

RHEL/CentOS从7.1起，Ubuntu从14.04起，它们的内核都支持Ceph FS模块。若是RHEL/CentOS 6.x等较老的操作系统，其内核中未包含Ceph FS模块，那么需

通过FUSE方式挂载（或者编译更新的内核）。以下介绍通过内核和FUSE挂载Ceph FS的方式。

（1）通过Ceph FS内核模块挂载

通过内核模块挂载时，如果使用Ceph客户端认证方式，则提供访问授权信息。通过-o提供账户admin和认证密钥，-t提供文件系统类型：

```
[root@compute-node1 ~]# mount  -o
name=admin,secret=AQC9xRBViMV9AhAAcgxg0TId+HgpJ4haXMuFZw== -
t ceph 10.89.13.71:6789:/ /mnt/
[root@compute-node1 ~]# df -TH
Filesystem Type Size Used Avail Use% Mounted on
/dev/vda1 xfs 50G 14G 37G 27% /
devtmpfs devtmpfs 4.1G 0 4.1G 0% /dev
tmpfs tmpfs 4.2G 0 4.2G 0% /dev/shm
tmpfs tmpfs 4.2G 18M 4.1G 1% /run
tmpfs tmpfs 4.2G 0 4.2G 0% /sys/fs/cgroup
/dev/vdc1 xfs 108G 82G 26G 77% /data
/dev/vdd1 xfs 215G 70G 146G 33% /data1
tmpfs tmpfs 821M 8.2k 821M 1% /run/user/0
10.89.13.71:6789:/ ceph 1.2T 32G 1.2T 3% /mnt
```

（2）通过FUSE方式挂载

```
[root@compute-node1 ~]# mkdir -p /cephfs/
[root@compute-node1 ~]# ceph-fuse -m 10.89.13.71:6789
```

```
/cephfs/
ceph-fuse[29831]: starting ceph client
ceph-fuse[29831]: starting fuse
[root@compute-node1 ~]# df -Th
Filesystem Type Size Used Avail Use% Mounted on
/dev/vda1 xfs 46G 13G 34G 27% /
devtmpfs devtmpfs 3.9G 0 3.9G 0% /dev
tmpfs tmpfs 3.9G 0 3.9G 0% /dev/shm
tmpfs tmpfs 3.9G 17M 3.9G 1% /run
tmpfs tmpfs 3.9G 0 3.9G 0% /sys/fs/cgroup
/dev/vdc1 xfs 100G 77G 24G 77% /data
/dev/vdd1 xfs 200G 65G 136G 33% /data1
tmpfs tmpfs 783M 8.0K 783M 1% /run/user/0
ceph-fuse fuse.ceph-fuse 1.1T 29G 1.1T 3% /mnt
```

以上两个操作分别让Ceph FS通过内核模块和FUSE方式进行了挂载。

在高性能计算领域，MPI是重要的分布式计算模型。MPI是一种基于消息传递的并行编程技术，其定义了一组具有可移植性的编程接口。通过MPI编程模型，程序员能编写基于消息通信的应用程序，而这个应用程序能在不同的节点上启动并协调工作，它们需访问共享存储，而Ceph FS正好提供共享存储的访问。

各个厂商或组织遵循这些标准实现自己的MPI软件包，典型的实现包含开放源码的MPICH、OpenMPI、LAM-MPI以及商业实现Intel MPI。其中，OpenMPI实现了MPI-1.2和MPI-2.0的通信规范，并支持TCP和RDMA（Remote Direct Memory Access），是常用的MPI实现库。CentOS 7.2默认带有OpenMPI软件包，安装过程如下。

（1）下载OpenMPI

```
[root@compute-node1 ~]# cd /cephfs/
[root@compute-node1 ~]# mkdir source && mkdir software
[root@compute-node1 ~]# cd source
[root@compute-node1 ~]# wget -c http://www.open-
mpi.org/software/ompi/v1.10/downloads/openmpi-1.10.1.tar.bz2
[root@compute-node1 ~]# tar xf openmpi-1.10.1.tar.bz2
[root@compute-node1 ~]# cd openmpi-1.10.1
[root@compute-node1 ~]# ./configure --
prefix=/cephfs/software/openmpi-1.10.1
[root@compute-node1 ~]# make -j4 && make install
```

至此，OpenMPI编译安装完毕。

（2）设置OpenMPI环境变量

编辑BASH脚本/cephfs/software/openmpi-1.10.1/run.sh，填写以下内容。

```
...
export OMPI_HOME=/cephfs/software/openmpi-1.10.1/
export PATH=$OMPI_HOME/bin:$PATH
export LD_LIBRARY_PATH=$OMPI_HOME/lib:$LD_LIBRARY_PATH
...
```

编写完毕后，执行环境变量设定：source/cephfs/software/openmpi-1.10.1/run.sh。

（3）蒙特卡洛法计算PI值示例代码

```
...
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "mpi.h"
#include <stdio.h>
#include <iostream>
using namespace std;
long long int MC(long long int mynumber,int myrank) {
   int mycount=0;
   double x,y;
   int i=0;
   for(i;i<mynumber;i++){
      x=2.0*(((double)rand())/RAND_MAX)-1.0;
      y=2.0*(((double)rand())/RAND_MAX)-1.0;
      if(x*x+y*y<1.0)
      mycount++;
   }
```

```
return mycount;
}
void serial() {
   int Allnumber=50000000;
   int count=0; double x,y,pi;
   double stime,etime,atime;
   stime=clock();
   for(int i=0;i<Allnumber;i++) {
      x=2.0*(((double)rand())/RAND_MAX)-1.0;
      y=2.0*(((double)rand())/RAND_MAX)-1.0;
      if(x*x+y*y<1.0)
      count++;
   }
   pi=4.0*count/Allnumber;
   etime=clock();
   atime=(etime-stime)/1000;
   cout<<"pi="<<pi<<",time="<<atime<<endl;
   //printf("pi= %12.12f ,alltime=%.3f ",pi,atime);
}
int main(int argc,char* argv[]) {
int myrank,size;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
long long int topinnumber,myinnumber,mynumber,allnumber;
double x,y,pi,stime,etime,atime;
stime = MPI_Wtime();
allnumber = 50000000;
mynumber = allnumber/size;
if(myrank != 0) {
   myinnumber = MC(mynumber,myrank);
   MPI_Send(&myinnumber,1,MPI_LONG_LONG,0,0,MPI_COMM_WORLD);
   cout << myrank << " of " << size << " : " << myinnumber
<< endl;
   //printf("%d of %d : %d\n", myrank, size, myinnumber);
} else {
   myinnumber = MC(mynumber,myrank);
   cout << myrank << " of " << size << " : " << myinnumber
<< endl;
   //printf("%d of %d : %d\n",myrank,size,myinnumber);
   topinnumber = myinnumber;
   for(int source = 1;source < size; source++) {

MPI_Recv(&myinnumber,1,MPI_LONG_LONG,source,0,MPI_COMM_WORLD
,MPI_STATUS_IGNORE);
```

```
        topinnumber += myinnumber;
    }
    pi = 4.0 * topinnumber / allnumber;
}
etime = MPI_Wtime();
atime = etime - stime;
if(myrank == 0)
    cout << "pi=" << pi << ",time=" << atime << endl;
    //printf("pi = %12.12d ,time = &.3d\n",pi,atime);
    MPI_Finalize();
    serial();
    return 0;
}
...
```

（4）通过OpenMPI C++并行编译器编译

```
mpic++ m_pi.cpp -o m_pi
```

（5）查看程序动态链接库（查看其运行是否满足环境依赖）

```
[root@performance mpi]# ldd m_pi
linux-vdso.so.1 => (0x00007ffd92977000)
libmpi_cxx.so.1 => /cephfs/software/openmpi-
1.10.1//lib/libmpi_cxx.so.1 (0x00007fc1a8096000)
libmpi.so.12 => /cephfs/software/openmpi-
1.10.1//lib/libmpi.so.12 (0x00007fc1a7dba000)
libstdc++.so.6 => /usr/lib64/libstdc++.so.6
(0x00007fc1a7a9b000)
libm.so.6 => /usr/lib64/libm.so.6 (0x00007fc1a7799000)
libgcc_s.so.1 => /usr/lib64/libgcc_s.so.1
(0x00007fc1a7582000)
libpthread.so.0 => /usr/lib64/libpthread.so.0
```

```
(0x00007fc1a7366000)
libc.so.6 => /usr/lib64/libc.so.6 (0x00007fc1a6fa5000)
libibverbs.so.1 => /usr/lib64/libibverbs.so.1
(0x00007fc1a6d92000)
libopen-rte.so.12 => /cephfs/software/openmpi-
1.10.1/lib/libopen-rte.so.12 (0x00007fc1a6b16000)
libopen-pal.so.13 => /cephfs/software/openmpi-
1.10.1/lib/libopen-pal.so.13 (0x00007fc1a6838000)
libnuma.so.1 => /usr/lib64/libnuma.so.1 (0x00007fc1a662b000)
libpciaccess.so.0 => /usr/lib64/libpciaccess.so.0
(0x00007fc1a6421000)
libdl.so.2 => /usr/lib64/libdl.so.2 (0x00007fc1a621d000)
librt.so.1 => /usr/lib64/librt.so.1 (0x00007fc1a6014000)
libutil.so.1 => /usr/lib64/libutil.so.1 (0x00007fc1a5e11000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc1a82b1000)
libnl-route-3.so.200 => /usr/lib64/libnl-route-3.so.200
(0x00007fc1a5bc1000)
libnl-3.so.200 => /usr/lib64/libnl-3.so.200
(0x00007fc1a59a5000)
```

（6）通过mpirun执行PI计算程序

```
time mpirun --allow-run-as-root -np 4 ./m_pi
```

上述mpirun在本机执行m_pi计算程序时，将会启动4个m_pi进程，并且这4个进程通过TCP协议进行消息通信。用户可通过编写主机文件（hostfile），指定mpirun将多个并行任务派送到不同的主机运行。通过使用Ceph FS共享文件系统，OpenMPI和应用存储均存

放其中，每个计算节点立即在相同路径启动mpirun执行计算任务。

6.2 Ceph FS作为大数据后端存储

Hadoop是一个对大量数据进行分布式处理的软件框架。Hadoop软件框架核心包括HDFS和MapReduce。其中，HDFS全称为Hadoop Distributed File System，是Hadoop内置的分布式文件系统，能运行在通用服务器集群上，构建大规模的存储系统；MapReduce，其概念为“映射”和“归约”，便于编程人员编写程序并运行在分布式系统上。

Ceph FS可作为Hadoop后端数据存储池，可替代HDFS的存储方案。原因是Hadoop提供DFS存储访问协议，对于通用应用而言其仅能将数据直接存放到文件系统，而且存储过程可能涉及内容修改，Ceph FS提供POSIX兼容的完整分布式文件系统，支持更广泛的应用集中存储数据。Ceph FS作为Hadoop后端数据存储原

理：Hadoop通过hadoop-cephfs.jar插件访问Ceph FS存储，其插件的下载地址为：http://download.ceph.com/tarballs/hadoop-cephfs.jar.

以下进行Hadoop与Ceph FS的配置。实验环境的软件版本如表6-1所示。

表6-1　实验环境的软件版本

项目	版本
操作系统	CentOS 7.2 x86_64
Ceph	0.94.5
Hadoop	2.6.3

根据之前的介绍，读者需要预先部署一个Ceph集群，并配置Ceph FS共享文件服务；同时，参照Hadoop官网教程http://hadoop.apache.org/docs/stable/安装集群管理。Ceph FS替换HDFS作为后端存储时，先关闭Hadoop集群，然后修改XML配置文件，整体操作过程如下。

1）在CentOS 7.2服务器上安装Ceph FS的Java接口库。

```
yum install cephfs-java libcephfs1-devel python-cephfs
libcephfs_jni1-devel
```

2）下载Hadoop。

```
wget -c
http://mirrors.aliyun.com/apache/hadoop/common/hadoop-
2.6.3/hadoop-2.6.3.tar.gz
```

3）解压Hadoop。

```
tar xf hadoop-2.6.3.tar.gz && cd hadoop-2.6.3
```

4）配置libcephfs_jni动态链接库。

```
cd lib/native
ln -s /usr/lib64/libcephfs_jni.so .
cd ../../
```

5）下载hadoop-cephfs.jar。

```
wget -c http://ceph.com/download/hadoop-cephfs.jar
```

6）放置到系统Java库路径。

```
cp hadoop-cephfs.jar /usr/share/java/
```

7）修改Hadoop的运行环境配置文件（增加粗体行）。

```
[root@performance hadoop-2.6.3]# vim etc/hadoop/hadoop-
env.sh
...
#export JSVC_HOME=${JSVC_HOME}
export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-
"/etc/hadoop"}export
HADOOP_CLASSPATH=/usr/share/java/libcephfs.jar:/usr/share/ja
va/hadoop-cephfs.jar:$HADOOP_CLASSPATH
```

```
# Extra Java CLASSPATH elements. Automatically insert
capacity-scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
if [ "$HADOOP_CLASSPATH" ]; then
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
else
...
```

8）修改Hadoop核心配置文件。

```
[root@performance hadoop-2.6.3]# vim etc/hadoop/core-
site.xml
...
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/tmp/hadoop/</value>
</property>
<property>
  <name>fs.default.name</name>
  <value>ceph://10.89.13.71/</value>
</property>
<property>
  <name>ceph.conf.file</name>     <!--载入
```

Ceph配置文件

```
-->
  <value>/etc/ceph/ceph.conf</value>
</property>
<property>
<name>ceph.auth.id</name>     <!--设定
```

Ceph集群访问认证用户

```
-->
<value>admin</value>
</property>
<property>
  <name>ceph.auth.keyring</name>     <!--设定
```

Ceph集群

admin用户认证密钥

```
-->
  <value>/etc/ceph/ceph.client.admin.keyring</value>
</property>
<property>
  <name>ceph.data.pools</name>     <!--设定
```

Ceph集群默认存储池，

```
hadoop1-->
  <value>hadoop1</value>
</property>
<property>
  <name>fs.ceph.impl</name>     <!--设定
```

Ceph集群访问文件接口

```
-->
  <value>org.apache.hadoop.fs.ceph.CephFileSystem</value>
</property>
</configuration>
...
```

9）创建Ceph FS存储池，供Hadoop使用。

```
[root@performance hadoop-2.6.3]# ceph osd pool create
hadoop1 128
pool 'hadoop1' created
[root@performance hadoop-2.6.3]# ceph osd pool set hadoop1
size 3
set pool 26 size to 3
[root@performance hadoop-2.6.3]# ceph osd pool set hadoop1
min_size 2
set pool 26 min_size to 2
[root@performance hadoop-2.6.3]# ceph mds add_data_pool
hadoop1
added data pool 26 to mdsmap
```

上述配置完成后，即可启动Hadoop集群。通过Hadoop的dfs命令，访问Ceph FS集群文件。通过Hadoop集群命令，可检验访问Ceph FS存储能力。

10）通过Hadoop命令列出当前Ceph FS存储文件的目录内容。

```
[root@performance hadoop-2.6.3]# bin/hadoop dfs -ls /
DEPRECATED: Use of this script to execute hdfs command is
deprecated.
Instead use the hdfs command for it.
Found 0 items
```

11）通过Hadoop命令将文件导入Ceph FS存储的文件目录。

```
[root@performance hadoop-2.6.3]# ./bin/hadoop dfs -put
~/ceph/ceph-0.94.5.tar.bz2 /ceph-0.94.5.tar.bz2
DEPRECATED: Use of this script to execute hdfs command is
deprecated.
Instead use the hdfs command for it.
```

再次查看Ceph FS存储的文件目录，发现文件已经导入。

```
[root@performance hadoop-2.6.3]# ./bin/hadoop dfs -ls /
DEPRECATED: Use of this script to execute hdfs command is
deprecated.
Instead use the hdfs command for it.
Found 1 items
-rw-r--r-- 3 root 7084809 2016-01-03 16:24 /ceph-
0.94.5.tar.bz2
```

在上述过程中，Ceph FS成功作为Hadoop的后端存储使用，可以通过DFS命令对文件进行上传和下载等操

作。开发者将需要运算的数据上传到Ceph FS后，执行Hadoop计算任务，其运算过程与HDFS后端一致。

6.3 本章小结

本章介绍Ceph FS两个重要的应用场景：高性能计算和大数据处理。传统高性能计算采用通用服务器搭建Lustre并加载DAS存储方式，数据冗余通过DAS阵列解决。Ceph FS基于RADOS，可直接在通用服务器上提供副本数据冗余，是高性能计算数据存储的可选方案。Ceph FS提供Java接口，能让使用Java编写的应用程序访问数据池，而Hadoop等一批基于Java编写的大数据计算框架，可以快速并发访问Ceph FS存储空间，提高数据吞吐能力。

第7章　块存储——虚拟化与数据库

7.1　Ceph与KVM

本节只描述Ceph与KVM虚拟化的结合结果，让Ceph块存储RBD作为KVM虚拟化的后端存储。先简单介绍KVM虚拟化背景知识。

KVM虚拟化技术是当前流行的虚拟化技术之一。KVM虚拟化技术本质是利用CPU硬件虚拟化技术在Linux等操作系统内核上添加代理模块，让运行在QEMU模拟器的客户机二进制指令通过KVM模块传递给CPU硬件虚拟化，并返回运行指令。相比纯QEMU模拟器，KVM模块能让客户机不需要进行二进制翻译过程，而直接使用CPU硬件虚拟化特性，大大提高了QEMU的运行速度。

首先要检查当前操作系统自带的qemu-kvm（Fedora/RHEL/CentOS/）或者qemu-system-x86_64（Debian/Ubuntu）是否支持RBD作为虚拟磁盘后端存储。如果不做特别说明，以下提及的CentOS 7.1和Ubuntu 14.04均指64位系统。

在之前的章节，我们已经构建了基本的Ceph环境，现在读者们先通过模板CentOS 7.1克隆一台虚拟机，用来运行KVM虚拟化。

目前CentOS 7.1和Ubuntu 14.04的QEMU虚拟机程序已经实现以LIBRBD的方式访问RBD块存储。通过以下操作确认当前操作系统自带的QEMU是否支持RBD块存储。

（1）检测操作系统的KVM虚拟化是否支持RBD块存储

1）在CentOS 7.1安装KVM虚拟化以及相关管理工具。

```
[root@kvm-node1 ~]# yum install qemu-kvm qemu-img libvirt
```

2）检查qemu-kvm是否支持RBD块存储。

```
[root@kvm-node1 ~]# qemu-img --help | grep rbd
Supported formats: vvfat vpc vmdk vhdx vdi sheepdog rbd raw
host_cdrom host_floppy host_device file qed qcow2 qcow
parallels nbd iscsi gluster dmg cloop bochs blkverify
blkdebug
```

3）在Ubuntu 14.04上安装KVM虚拟化以及相关管理工具。

```
ubuntu@ubuntu:~$ sudo apt-get install qemu-system-x86 qemu-
kvm qemu-utils libvirt-bin
```

4）在Ubuntu 14.04上查看qemu-img是否支持RBD块设备。

```
root@ubuntu:~# qemu-img --help | grep rbd
Supported formats: vvfat vpc vmdk vhdx vdi sheepdog sheepdog
sheepdog rbd raw host_cdrom host_floppy host_device file qed
qcow2 qcow parallels nbd nbd nbd dmg tftp ftps ftp https
http cow cloop bochs blkverify blkdebug
```

如果出现rbd关键词，则说明当前qemu-kvm或者qemu-system-x86_64支持RBD块存储。

（2）使用块存储

接下来介绍如何让KVM虚拟化访问Ceph RBD的虚拟磁盘文件。

1）在kvm-node1安装Ceph软件。

```
[root@kvm-node1 ~]# yum install qemu-kvm qemu-img
```

①把/etc/ceph/ceph.conf复制到kvm-node1节点相同的路径位置。

②把/etc/ceph/ceph.client.admin.keyring复制到kvm-node1节点相同的路径位置。

2）此时kvm-node1的qemu-img工具使用/etc/ceph/ceph.conf作为默认配置文件，关于RBD块存储的转换操作将直接使用配置文件定义的参数。

3）为了让KVM虚拟化创建虚拟机并引导操作系统，可将已经安装操作系统的QCOW2镜像导入到RBD存储池vmpool1。下面创建RBD存储池vmpool1，并导入系统镜像。

```
[root@ceph-mon1 ~]# wget -c
http://download.zstack.org/templates/zstack-image-1.2.qcow2
[root@ceph-mon1 ~]# ceph osd pool create vmpool1 128
[root@ceph-mon1 ~]# qemu-img convert -f qcow2 -O raw zstack-
image-1.2.qcow2 rbd:vmpool1/root-image1
```

其中，上面运行qemu-img镜像导入命令时，将qcow2转化成raw格式。这是由于KVM虚拟化访问Ceph

RBD存储时将会配置raw的访问格式，故在此步骤中转换成raw格式。

4）如果安装Ceph时开启了cephx认证，则需要创建客户端访问权限。

```
[root@ceph-mon1 ~]# ceph auth get-or-create client.vmpool1
mon 'allow r' osd 'allow rwx pool=vmpool1'
[client.vmpool1]
     key = AQCaG/FUeHL/MxAAsRr9ike00H4o7rQ/NJkhhQ==
```

5）利用上面的密钥信息，创建XML配置文件。

```
[root@ceph-mon1 ~]# cat secret-vmpool1.xml
<secret ephemeral='no' private='no'>
   <usage type='ceph'>
     <name>client-vmpool1-secret</name>
   </usage>
</secret>
```

6）将上述的XML配置导入到Libvirt。

```
[root@ceph-mon1 ~]# virsh secret-define --file secret-
pool4test.xml
Secret e96b4fb6-832a-4d40-9286-a13f91548df1 created
```

7）将上述生成的UUID密码容器，设置给密钥。

```
virsh secret-set-value e96b4fb6-832a-4d40-9286-a13f91548df1
--base64 AQCaG/FUeHL/MxAAsRr9ike00H4o7rQ/NJkhhQ==
```

8）以下是KVM虚拟机描述文件（XML）的RBD块设备片段。

```
<disk type='network' device='disk'>
   <driver name='qemu' type='raw'/>
   <auth username='vmpool1'>
   <secret type='ceph' uuid='e96b4fb6-832a-4d40-9286-
a13f91548df1'/>
   </auth>
   <source protocol='rbd' name='vmpool1/root-image1'>
   <host name='ceph-mon1' port='6789'/>
   </source>
   <target dev='vdb' bus='virtio'/>
</disk>
```

其中，“ceph-mon1”是主机自定义域名解析，填写Ceph MON服务的IP地址。

以下提供了KVM虚拟机主机访问Ceph RBD块存储的完整XML配置文件。

```
<domain type='kvm'>
  <name>netapp-manager</name>
  <memory unit='GiB'>4</memory>
  <currentMemory unit='GiB'>4</currentMemory>
  <vcpu placement='static'>4</vcpu>
  <cpu mode='host-model'>
    <topology sockets='2' cores='2' threads='1'/>
  </cpu>
  <os>
    <type arch='x86_64'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='localtime'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
      <disk type='network' device='disk'>
        <driver name='qemu' type='raw'/>
        <auth username='vmpool1'>
           <secret type='ceph' uuid='e96b4fb6-832a-4d40-
9286-a13f91548df1'/>
       </auth>
      <source protocol='rbd' name='vmpool1/root-image1'>
        <host name='ceph-mon1' port='6789'/>
      </source>
      <target dev='vdb' bus='virtio'/>
    </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw' cache='none'/>
    <target dev='hdc' bus='ide'/>
    <readonly/>
  </disk>
  <interface type='bridge'>
    <source bridge='cloudbr0'/>
    <model type='virtio'/>
  </interface>
  <input type='tablet' bus='usb'>
```

```
  </input>
  <input type='mouse' bus='ps2'/>
  <graphics type='vnc' passwd='cloud123' port='-1'
autoport='yes' listen= '0.0.0.0' keymap='en-us'>
  </graphics>
    <memballoon model='none'>
    </memballoon>
  </devices>
  <seclabel type='none'/>
</domain>
```

7.2 Ceph与OpenStack

本节只描述Ceph与OpenStack的结合，让Ceph块存储RBD作为OpenStack的后端存储。

先来介绍一下OpenStack背景知识。

OpenStack是一个由NASA（美国国家航空航天局）和Rackspace合作研发并发起的开源云计算管理平台项目，由几个主要的组件组合起来完成具体工作。OpenStack支持几乎所有类型的云环境，项目目标是提供实施简单、可大规模扩展、组件丰富、标准统一的云计算管理平台。OpenStack通过各种互补的服务提供了“基础设施”即服务（IaaS）的解决方案，每个服务提供API以进行集成。

OpenStack主要有以下3个组件可以和Ceph对接。

• Glance：OpenStack的镜像管理组件，和Ceph对接之后镜像将存储在Ceph集群里面。

• Cinder：OpenStack的云硬盘组件，和Ceph对接之后云硬盘将存储在Ceph集群里面。

• Nova：Openstack的核心组件，和Ceph对接之后云主机的系统卷将会存储在Ceph集群里面。

以上3个组件和Ceph对接之后将会实现秒级创建。

温馨提示：

上传的镜像必须是RAW格式的才能支持COW。

下面说一下OpenStack如何与Ceph对接。

1）创建几个对应的池。

```
ceph osd pool create volumes 128
ceph osd pool create images 128
ceph osd pool create backups 128
ceph osd pool create vms 128
```

2）在Glance节点安装python-rbd。

```
sudo apt-get install python-rbd #for Ubuntusudo yum install python-rbd #for CentOS or RedHat
```

3）在Cinder-volume节点和Nova-compute节点安装ceph-common。

```
sudo apt-get install ceph-common #for Ubuntusudo yum install ceph #for CentOS or RedHat
```

4）把Ceph配置文件复制到Glance节点、Cinder-volume节点和Nova-compute节点。

```
ssh （

your-openstack-server-node)
```

```
sudo tee /etc/ceph/ceph.conf </etc/ceph/ceph.conf
```

5）创建CephX认证授权用户。

```
ceph auth get-or-create client.cinder mon 'allow r' osd
'allow class-read object_prefix rbd_children, allow rwx
pool=volumes, allow rwx pool=vms, allow rx pool=images'
ceph auth get-or-create client.glance mon 'allow r' osd
'allow class-read object_prefix rbd_children, allow rwx
pool=images'
ceph auth get-or-create client.cinder-backup mon 'allow r'
osd 'allow class-read object_prefix rbd_children, allow rwx
pool=backups'
```

6）把密钥复制到Glance节点、Cinder-volume节点、Nova-compute节点以及Cinder-Backup节点并且授权。

```
ceph auth get-or-create client.glance | ssh {your-glance-
api-server} sudo tee /etc/ceph/ceph.client.glance.keyring
ssh {your-glance-api-server} sudo chown glance:glance
/etc/ceph/ceph.client.glance.keyring
ceph auth get-or-create client.cinder | ssh {your-volume-
server} sudo tee /etc/ceph/ceph.client.cinder.keyring
ssh {your-cinder-volume-server} sudo chown cinder:cinder
/etc/ceph/ceph.client.cinder.keyring
ceph auth get-or-create client.cinder-backup | ssh {your-
cinder-backup-server} sudo tee /etc/ceph/ceph.client.cinder-
backup.keyring
```

```
ssh {your-cinder-backup-server} sudo chown cinder:cinder
/etc/ceph/ceph.client.cinder-backup.keyring
ceph auth get-or-create client.cinder | ssh {your-nova-
compute-server} sudo tee
/etc/ceph/ceph.client.cinder.keyring
ceph auth get-key client.cinder | ssh {your-compute-node}
tee client.cinder.key
```

7）配置Glance节点，在glance-api配置文件中修改以下内容。

```
[DEFAULT]...
default_store = rbd
show_image_direct_url = True
[glance_store]
stores = rbd
rbd_store_pool = images
rbd_store_user =  glance
ceph_conf = /etc/ceph/ceph.conf
rbd_store_chunk_size = 8
```

8）配置cinder-volume节点，添加以下内容。

```
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_flatten_volume_from_snapshot = false
rbd_max_clone_depth = 5
rbd_store_chunk_size = 4
rados_connect_timeout = -1
glance_api_version = 2
rbd_user = cinder
rbd_secret_uuid = 457eb676-33da-42ec-9a8c-9293d545c337 #这里的
```

UUID需要由

uuidgen命令生成

9）配置Cinder Backup节点，添加以下内容。

```
backup_driver = cinder.backup.drivers.ceph
backup_ceph_conf = /etc/ceph/ceph.conf
backup_ceph_user = cinder-backup
backup_ceph_chunk_size = 134217728
backup_ceph_pool =  backups
backup_ceph_stripe_unit = 0
backup_ceph_stripe_count = 0
restore_discard_excess_bytes = true
rbd_user = cinder
rbd_secret_uuid = 457eb676-33da-42ec-9a8c-9293d545c337 #这里的
```

UUID需要由

uuidgen命令生成

10）配置nova-compute节点，创建secret.xml并且注入到Libvirt里面。

```
cat > secret.xml <<EOF
<secret ephemeral='no' private='no'> <uuid>457eb676-33da-
42ec-9a8c-9293d545c337</uuid> <usage type='ceph'>
<name>client.cinder secret</name> </usage></secret>
EOF
sudo virsh secret-define --file secret.xml
sudo virsh secret-set-value --secret 457eb676-33da-42ec-
9a8c-9293d545c337 --base64 $(cat client.cinder.key) && rm
client.cinder.key secret.xml
```

11）编辑nova.conf文件，添加以下内容。

```
[libvirt]
images_type= rbd
images_rbd_pool= volumes
images_rbd_ceph_conf= /etc/ceph/ceph.conf
rbd_user= cinder
rbd_secret_uuid= 457eb676-33da-42ec-9a8c-9293d545c337
disk_cachemodes="network=writeback"
libvirt_live_migration_flag="VIR_MIGRATE_UNDEFINE_SOURCE,VIR
_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE,VIR_MIGRATE_PERSIST_DEST
"
```

12）重启glance-api、cinder-volume、cinder-backup和nova-compute服务。

```
sudo service glance-api restart #for Ubuntu
sudo service nova-compute restart #for Ubuntu
sudo service cinder-volume restart #for Ubuntu
sudo service cinder-backup restart #for Ubuntu
sudo service openstack-glance-api restart #for CentOS or Red
Hat
sudo service openstack-nova-compute restart #for CentOS or
Red Hat
sudo service openstack-cinder-volume restart #for CentOS or
Red Hat
sudo service openstack-cinder-backup restart #for CentOS or
Red Hat
```

13）验证COW云主机秒级创建。

前提条件，必须上传一个RAW格式的镜像文件，因为只有RAW格式的镜像才支持COW（Copy On Write）秒级创建。

• 在OpenStack的Horizon界面开一个云主机。

• 选择boot from volume（从镜像启动创建一个新卷）。

• 在Ceph节点上运行rbd-p volumes ls-l，出现与以下内容证明成功。

```
volume-f75d744d-8b19-4f1f-a7b1-6277d358244f 20480M
images/db500efe-3ae6-47c9-90e5-d57d4e62e291@snap
```

7.3 Ceph与CloudStack

CloudStack，项目地址 http://cloudstack.apache.org/ ，提供一个开源的“基础设施即服务”（IaaS）的解决方案，特点是高可用和高扩展的能力。CloudStack前身是Cloud.com，2011年7月被Citrix思杰收购并100%开源。2012年4月，Citrix思杰宣布把CloudStack交给Apache软件基金会管理以及开发。目前CloudStack采用Apache License v 2.0开源许可证发布源码，源码托管：https://github.com/apache/cloudstack/ 。

目前CloudStack最新版本为4.6.0，支持虚拟化包含XenServer、KVM、LXC、VMware、HyperV和OVM3，同时支持物理服务器（Baremetal）。其中，KVM虚拟化，支持RHEL/CentOS（6.x/7.x）、

Debian（7.x/8.x）/Ubuntu（12.04,14.04）和Fedora（20/21/22/23）等发行版本。CloudStack+KVM虚拟化的场景，主存储支持NFS、CLVM、RBD、共享存储和本地存储。

接下来描述CloudStack+KVM+RBD的场景部署。

目前，CloudStack+KVM的RBD块存储代码由CloudStack社区贡献者Wido den Hollander（https://github.com/wido/ ）维护。Wido设计KVM+RBD方案时，是通过Libvirt管理RBD Pool。而在Debian/Ubuntu，Libvirt默认编译开启--with-storage-rbd选项，所以使用其操作系统部署CloudStack+KVM方案时较为顺利。

而RHEL/CentOS的6.x和7.x的Libvirt默认编译设置了--without-storage-rbd选项，所以RHEL/CentOS KVM节点可以直接使用RBD。以下以CentOS 7.1为例，

重新编译Libvirt，使用Debian/Ubuntu KVM节点的场景可以忽略。当前CentOS 7.1的Libvirt版本为libvirt-1.2.8-16.el7_1.5.src.rpm。

1）安装编译版。

源码下载链接（若有更新，请读者自行下载最新版本）：

http://vault.centos.org/centos/7/updates/Source/SPackages/libvirt-1.2.8-16.el7_1.5.src.rpm 。

2）安装编译依赖包。

```
yum groupinstall "Development Tools"
yum -y install scrub dbus-devel systemtap-sdt-devel numactl-devel glusterfs-devel glusterfs-api-devel device-mapper-devel parted-devel avahi-devel python-devel libxml2-devel xhtml1-dtds readline-devel ncurses-devel libtasn1-devel gnutls-devel libattr-devel libblkid-devel augeas libpciaccess-devel yajl-devel sanlock-devel libpcap-devel libnl3-devel cyrus-sasl-devel polkit-devel libcap-ng-devel fuse-devel netcf-devel libcurl-devel audit-libs-devel ceph-devel
```

若未安装以上的软件包，编译Libvirt时也会提示依赖。

3）执行编译。

```
rpm build --rebuild libvirt-1.2.8-16.el7_1.5.src.rpm
```

当执行上述编译时，监测屏幕源码包解压结束，按Ctrl+C终止编译。

```
[root@server1 libvirt]#
...
+ cd libvirt-1.2.8
+ /usr/bin/chmod -Rf a+rX,u+w,g-w,o-w .
+ PATCHCOUNT=355
+ PATCHLIST=/tmp/lua_NaOiu0
+ git init -q
+ git config user.name rpm-build
+ git config user.email rpm-build
+ git config gc.auto 0
+ git add .
^Cerror: Error executing scriptlet /var/tmp/rpm-tmp.DqWwd0
(%prep)
...
```

4）修改文件/root/rpmbuild/SPECS/ceph.spec，修改关于rbd pool的支持选项。

```
[root@server1 ~]# vim /root/rpmbuild/SPECS/ceph.spec
...
112 %define with_storage_iscsi 0%{!?_without_storage_iscsi:%
{server_drivers}}
113 %define with_storage_disk 0%{!?_without_storage_disk:%
{server_drivers}}
114 %define with_storage_mpath 0%{!?_without_storage_mpath:%
{server_drivers}}
115 %if 0%{?fedora} >= 16 || 0%{?rhel} >= 7
```

```
116 %define with_storage_rbd 0%{!?_without_storage_rbd:%
{server_drivers}}
117 %else
118 %define with_storage_rbd 0
119 %endif
...
```

5）执行编译。

```
[root@server1 SPECS]# rpmbuild --bb libvirt.spec
```

6）查看编译的rpm包。

```
[root@server1 SPECS]# ls /root/rpmbuild/RPMS/x86_64/*1.2.8*
/root/rpmbuild/RPMS/x86_64/libvirt-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-client-1.2.8-
```

```
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-config-network-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-config-nwfilter-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-interface-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-lxc-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-network-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-nodedev-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-nwfilter-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-qemu-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-secret-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-driver-storage-
1.2.8-16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-kvm-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-daemon-lxc-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-debuginfo-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-devel-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-docs-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-lock-sanlock-1.2.8-
16.el7.centos.5.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/libvirt-login-shell-1.2.8-
16.el7.centos.5.x86_64.rpm
```

因为当前编译获得的Libvirt包版本信息和YUM源的版本信息一致，所以需要强制覆盖安装。

7）先安装YUM源版本。

```
yum -y install libvirt libvirt-client libvirt-daemon
libvirt-daemon-config-network libvirt-daemon-config-nwfilter
libvirt-daemon-driver-interface libvirt-daemon-driver-lxc
libvirt-daemon-driver-network libvirt-daemon-driver-nodedev
libvirt-daemon-driver-nwfilter libvirt-daemon-driver-qemu
libvirt-daemon-driver-secret libvirt-daemon-driver-storage
libvirt-daemon-kvm libvirt-daemon-lxc libvirt-devel libvirt-
docs libvirt-lock-sanlock libvirt-login-shell
```

8）然后强制安装编译后的Libvirt版。

```
rpm -ivh libvirt-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-
client-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-daemon-
1.2.8-16.el7.centos.5.x86_64.rpm libvirt-daemon-config-
network-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-daemon-
config-nwfilter-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-
daemon-driver-interface-1.2.8-16.el7.centos.5.x86_64.rpm
libvirt-daemon-driver-lxc-1.2.8-16.el7.centos.5.x86_64.rpm
libvirt-daemon-driver-network-1.2.8-
16.el7.centos.5.x86_64.rpm libvirt-daemon-driver-nodedev-
1.2.8-16.el7.centos.5.x86_64.rpm libvirt-daemon-driver-
nwfilter-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-daemon-
driver-qemu-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-daemon-
driver-secret-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-
daemon-driver-storage-1.2.8-16.el7.centos.5.x86_64.rpm
libvirt-daemon-kvm-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-
daemon-lxc-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-
debuginfo-1.2.8-16.el7.centos.5.x86_64.rpm libvirt-devel-
1.2.8-16.el7.centos.5.x86_64.rpm libvirt-docs-1.2.8-
16.el7.centos.5.x86_64.rpm libvirt-lock-sanlock-1.2.8-
16.el7.centos.5.x86_64.rpm libvirt-login-shell-1.2.8-
16.el7.centos.5.x86_64.rpm --force
```

9）Libvirt重新编译安装后，即可使用RBD Pool功能。

10）参考KVM和RBD的配置方法，在Ceph生成客户端认证Key。

```
# 若不存在存储池

vmpool1 则创建

[root@ceph-mon1 ~]# ceph osd pool create vmpool1 128
[root@ceph-mon1 ~]# ceph auth get-or-create client.vmpool1
mon 'allow r' osd 'allow rwx pool=vmpool1'
[client.vmpool1]
key = AQCaG/FUeHL/MxAAsRr9ike00H4o7rQ/NJkhhQ==
```

11）把这个Key信息填写到CloudStack，如图7-1所示。

①名字：管理员可以自定义。

②RADOS Monitor：填写Ceph Mon一个IP地址。

③RADOS Pool：填写Ceph存储池，即vmpool1。

④RADOS User：填写访问用户，即vmpool1。

⑤RADOS Secret：填写秘钥信息，即AQCaG/FUeHL/MxAAsRr9ike0OH4o7rQ/NJkhhQ==。

填写完毕后，在界面上单击“确定”按钮。这样，CloudStack可以通过Ceph RBD作为主存储。


Project: Default view
添加主存储
范围:
群集
* 区域:
ZONE1
* 提供点:
POD1
* 群集:
CLUSTER1
* 名称:
* 协议:
RBD
RADOS Monitor:
RADOS Pool:
RADOS User:
RADOS Secret:
存储标签:
取消
确定
HOST
For XenServer, choose NFS, iSCSI, or PreSetup. For KVM, choose NFS, SharedMountPoint, RDB, CLVM or Gluster. For vSphere, choose VMFS (iSCSI or FiberChannel) or NFS. For Hyper-V, choose SMB/CIFS. For LXC, choose NFS or SharedMountPoint. For OVM, choose NFS or ocfs2.

图7-1　填写key信息

7.4 Ceph与ZStack

ZStack（http://zstack.org/ ），是一个全新的开源IaaS解决方案，发布于2015年4月。ZStack目标是解决数据中心自动化问题，并能通过API实现对计算、存储和网络资源的分配。相比其他IaaS技术架构，ZStack为全异步架构、微服务和一致性Hash，可承载高并发的API请求，具备稳定的架构、非常简化的部署和升级的特点，架构如图7-2所示。ZStack同样采用Apache License v2.0发布源码，源码托管：https://github.com/zstackorg/zstack 。

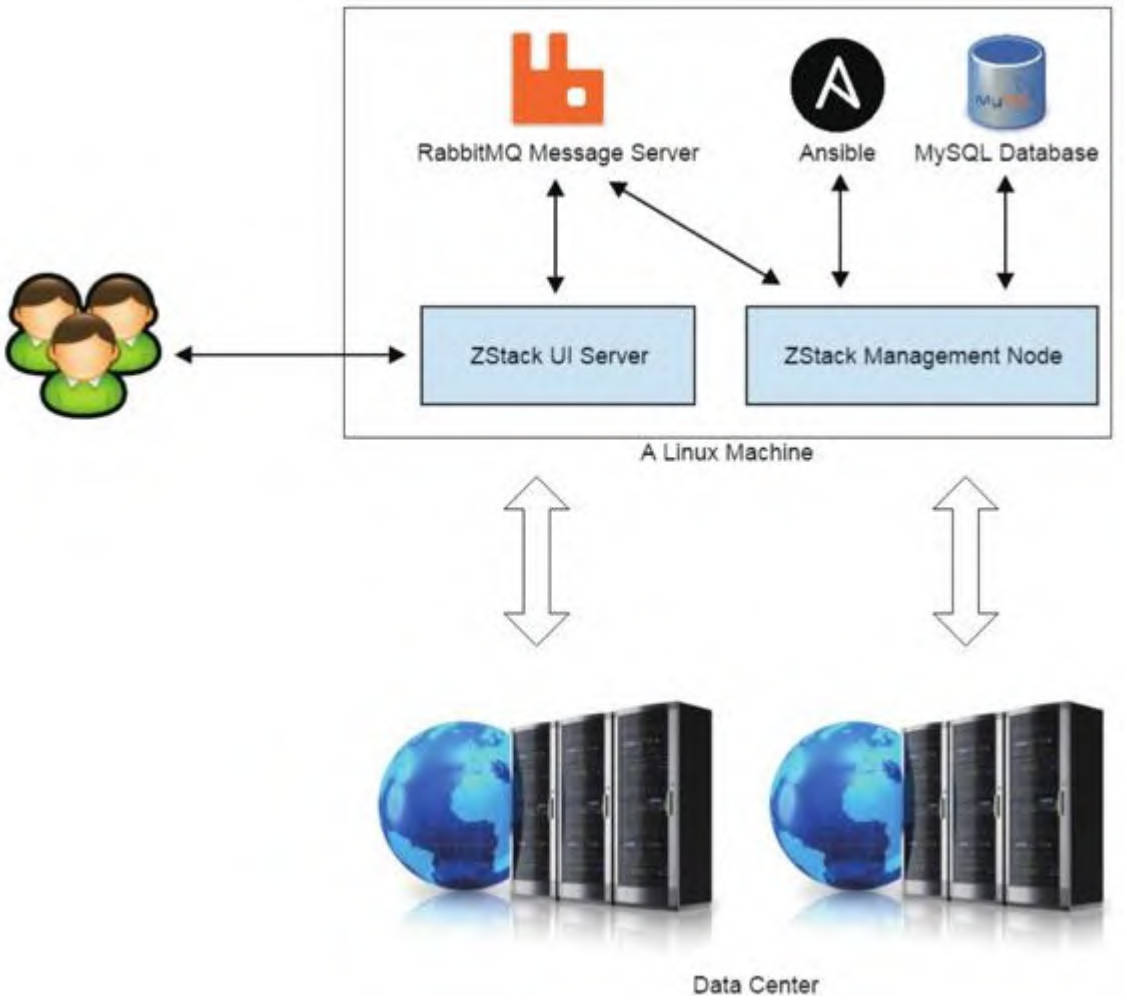

图7-2　ZStack整体架构示意图

以ZStack版本0.9为例，在CentOS 7.2 x86_64环境下单节点快速部署ZStack。

单节点模式下，请在支持CPU硬件虚拟化的服务器上部署（或者在打开嵌套虚拟化的虚拟机运行），并

至少配置CPU 4核心和内存8GB，并在根目录预留100GB硬盘空间。读者可以使用国内开源镜像配置CentOS-Base和EPEL仓库。

- 中国科学技术大学开源镜像：http://mirrors.ustc.edu.cn/ 。

- 阿里云开源镜像：http://mirrors.aliyun.com/ 。

管理节点安装过程如下。

1）下载安装脚本。

```
curl -L
http://download.zstack.org/releases/0.9/0.9.0/zstack-
install.sh -o install-zstack.sh
```

2）执行部署。

```
sudo bash install-zstack.sh -a
#国内读者可以通过国内
```

CDN下载

ZStack安装包

```
curl -L
http://download.zstack.org/releases/0.9/0.9.0/zstack-
install.sh -o install-zstack.sh
sudo bash install-zstack.sh -a -f
http://7xi3lj.com1.z0.glb.clouddn.com/releases/0.9/0.9.0/zst
ack-all-in-one-0.9.0.tgz
```

3）安装结束后，可通过网页或者命令行的方式管理ZStack。浏览器访问http://ip:5000/ 即可跳转到登录界面，界面如图7-3所示。

图7-3　ZStack登录界面

默认管理员登录账号为admin，密码为password。

4）通过系统命令行登录，方式如下。

```
[root@sa~]# zstack-cli
  ZStack command line tool
  Type "help" for more information
  Type Tab key for auto-completion
  Type "quit" or "exit" or crl-d to exit
>>>LogInByAccount accontName=admin password=password
{
    "inventory":{
       "accountUuid":"36c27e8ff05c4780bf6d2fa65700f22e"
       "createDate":"Nov 30, 2015 9:14:53 AM",
       "expiredDate":"Nov 30, 2015 11:14:53 AM",
       "userUuid":"36c27e8ff05c4780bf6d2fa65700f22e",
    },
    "success":true
}
>>>
```

ZStack支持丰富的操作命令。详细命令解释和API使用，请参看：http://zstackdoc.readthedocs.org/en/latest/ 。

如果以上都可执行，表明ZStack已经安装，但是需要进行初始化才能使用。ZStack支持5种经典的网络场景，分别是EIP、Flat Network、Three Tiered Network、Security Group和Elastic Port Forwarding。同时，ZStack不局限以上的经典网络场景，可以根据业务场景组合不同的网络服务。

ZStack详细的经典网络场景设定：http://zstack.org/tutorials/ 。其中，教程提及的主存储使用NFS，备份存储使用SFTP。读者可选择较为简单的Flat Network进行配置，配置成功后，即可启动第一台虚拟主机。

1.ZStack+Ceph场景

从0.9版本开始，ZStack正式支持Ceph作为主存储（Primary Storage）和备份存储（Backup Storage）的设备。ZStack在Ceph上实现了主存储和备份存储的统一，避免了卷和镜像的来回拷贝引来的带宽消耗。ZStack+Ceph场景的架构如图7-4所示。

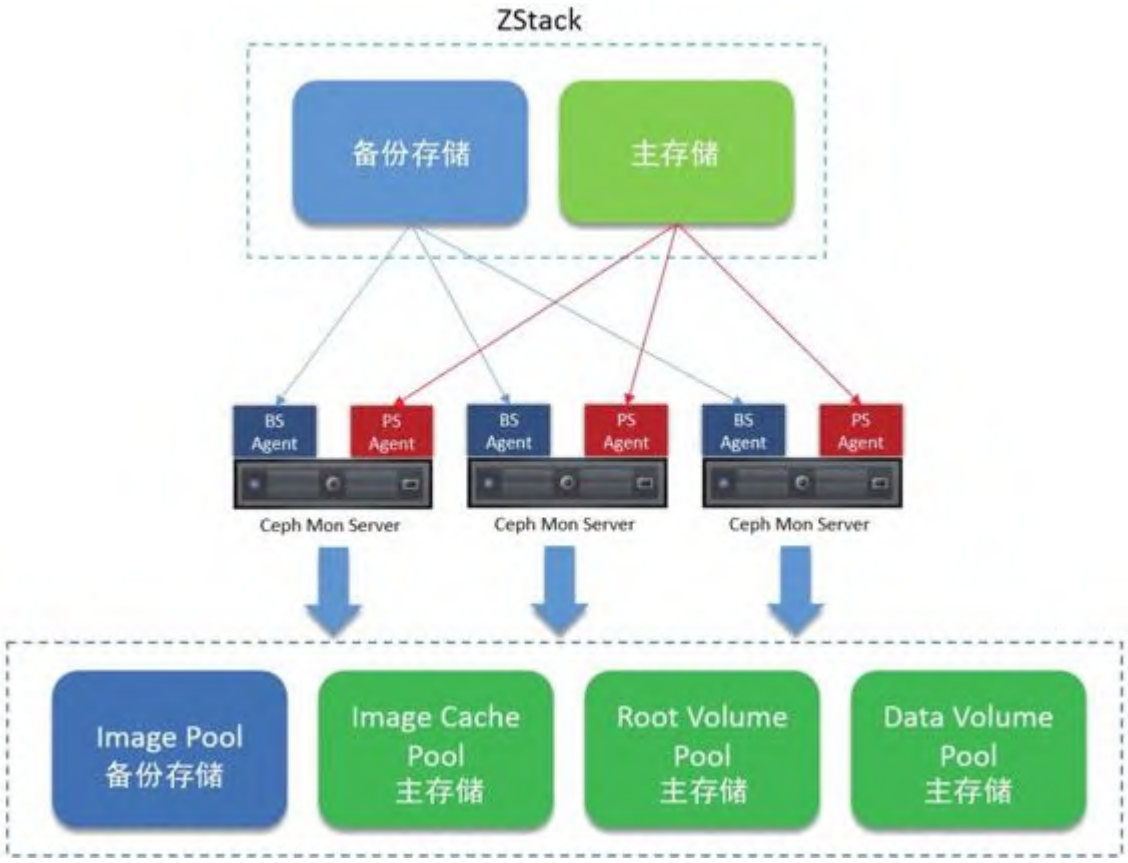

图7-4　ZStack+Ceph场景的架构

ZStack会在Ceph Mon节点同时部署主存储和备份存储的代理服务（PS Agent，BS Agent），通过心跳检测和一致性Hash算法把关于存储操作的任务调度到健康状态的代理服务器上，执行后并回调。所以，ZStack与Ceph集成的管理架构具备先天的高可用和可扩展能力。

与CloudStack和OpenStack不同的是，ZStack和Ceph之间的交互是通过部署在Ceph Mon服务器上的代理服务（Agent）来完成的。ZStack从0.9版本开始，支持Ceph RBD作为主存储被备份存储，其增加的CLI如下。

```
AddCephBackupStorage AddCephPrimaryStorage
AddMonToCephBackupStorage AddMonToCephPrimaryStorage
QueryCephBackupStorage QueryCephPrimaryStorage
RemoveMonFromCephBackupStorage
RemoveMonFromCephPrimaryStorage
```

其中：

• AddCephPrimaryStorage：添加Ceph作为主存储，主要存放镜像缓存和虚拟机运行的系统盘与数据盘，部署主存储代理。

• AddCephBackupStorage：添加Ceph作为备份存储，主要存储模板和快照，部署备份存储代理。

• AddMonToCephBackupStorage：为Ceph备份存储添加Mon管理节点，部署备份存储代理。

• AddMonToCephPrimaryStorage：为Ceph主存储添加Mon管理节点，部署主存储代理。

• QueryCephBackupStorage：查询Ceph备份存储的对象，包含模板和快照。

• QueryCephPrimaryStorage：查询Ceph主存储的对象，包含虚拟机的系统盘和数据盘。

• RemoveMonFromCephBackupStorage：从Ceph备份存储移除Mon节点（ZStack识别操作，并非对Ceph集群操作）。

• RemoveMonFromCephPrimaryStorage：从Ceph主存储移除Mon节点（ZStack识别操作，并非对Ceph集群操作）。

2.ZStack添加Ceph作为主存储

在ZStack的网页界面操作，添加Ceph集群作为主存储，如图7-5所示。

在添加主存储（Primary Storage）时，选择Ceph作为主存储类型，并填写Mon服务器域名/IP地址、用户名和对应密码。这里需要注意的是，用户可以添加单个Mon节点，也可以添加多个Mon节点。ZStack会对所添加Mon的节点部署主存储的代理服务。

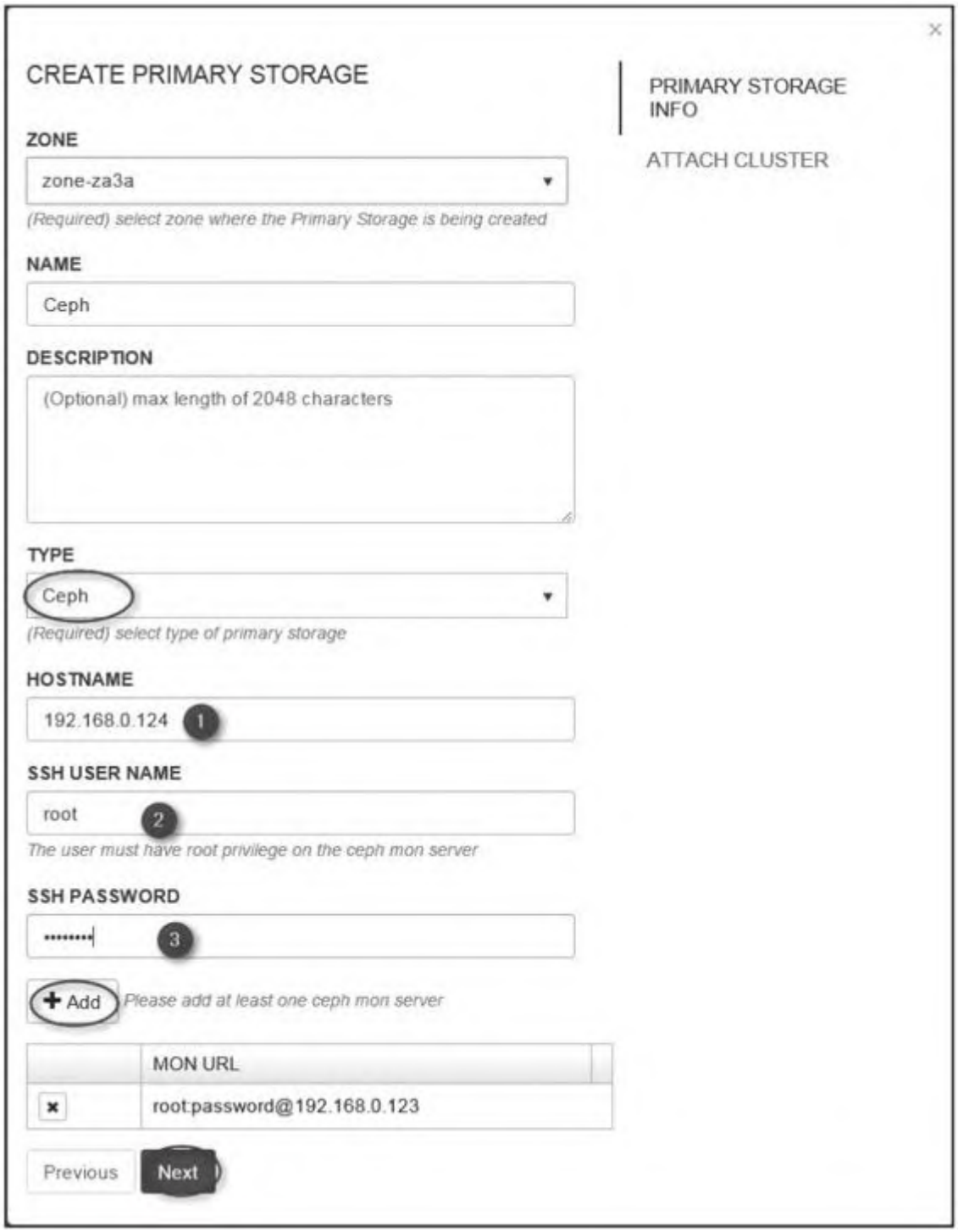

图7-5　添加Ceph集群作为主存储

ZStack支持通过命令行添加Ceph作为主存储。AddCephPrimaryStorage命令支持的参数如下。

表7-1　AddCephPrimaryStorage命令支持的参数

参数	说明
zoneUuid	主存储所属 Zone 的 Uuid
monUrls	Mon 节点的地址，支持多节点
name	定义主存储名字
dataVolumePoolName	数据盘指定目标存储池
description	主存储信息描述
imageCachePoolName	镜像缓存指定目标存储池

（续）

参数	说明
resourceUuid	指定主存储 UUID
rootVolumePoolName	系统盘指定目标存储池
systemTags	定义系统标签
timeout	定义监控检测超时时间
userTags	定义用户标签

其中，粗体标记为必需参数。

上文中在Web界面添加Ceph存储的例子，可通过以下命令行执行，效果相同。

```
>>>AddCephPrimaryStorage name=ceph
zoneUuid=d914841733fa499c9dc6d63ea339469d
monUrls=root:password@192.168.0.123,
root:password@192.168.0.124,root:password@192.168.0.125
```

在以上创建主存储过程中，ZStack会生成唯一UUID作为对应的镜像缓存、系统盘和数据盘的存储池，并在存储池名字开头标记识别用途。读者可以预先创建了特定的Ceph存储池，给ZStack提供作为镜像缓存、系统盘和数据盘使用。在Mon节点预先执行：

```
ceph osd pool create cache-disk-pool 256 256
ceph osd pool create root-disk-pool 256 256
ceph osd pool create data-disk-pool 256 256
```

在ZStack-cli命令行工具执行以下命令行：

```
>>>AddCephPrimaryStorage name=ceph
zoneUuid=d914841733fa499c9dc6d63ea339469d
monUrls=root:password@192.168.0.123,root:password@192.168.0.
124,root:passwo-rd@192.168.0.125 imageCachePoolName=cache-
disk-pool rootVolumePoolName=root-disk-pool
dataVolumePoolName=data-disk-pool
```

3.ZStack添加Ceph作为备份存储

在添加备份存储时，选择Ceph作为主存储类型，并填写Mon服务器域名/IP地址、用户名和对应密码，

如图7-6所示。

和添加主存储类似，用户可以添加单个Mon节点，也可以添加多个Mon节点。ZStack会对所添加Mon的节点部署主存储的代理服务。

ZStack支持通过命令行来添加Ceph作为备份存储。AddCephBackupStorage命令支持的参数如下。

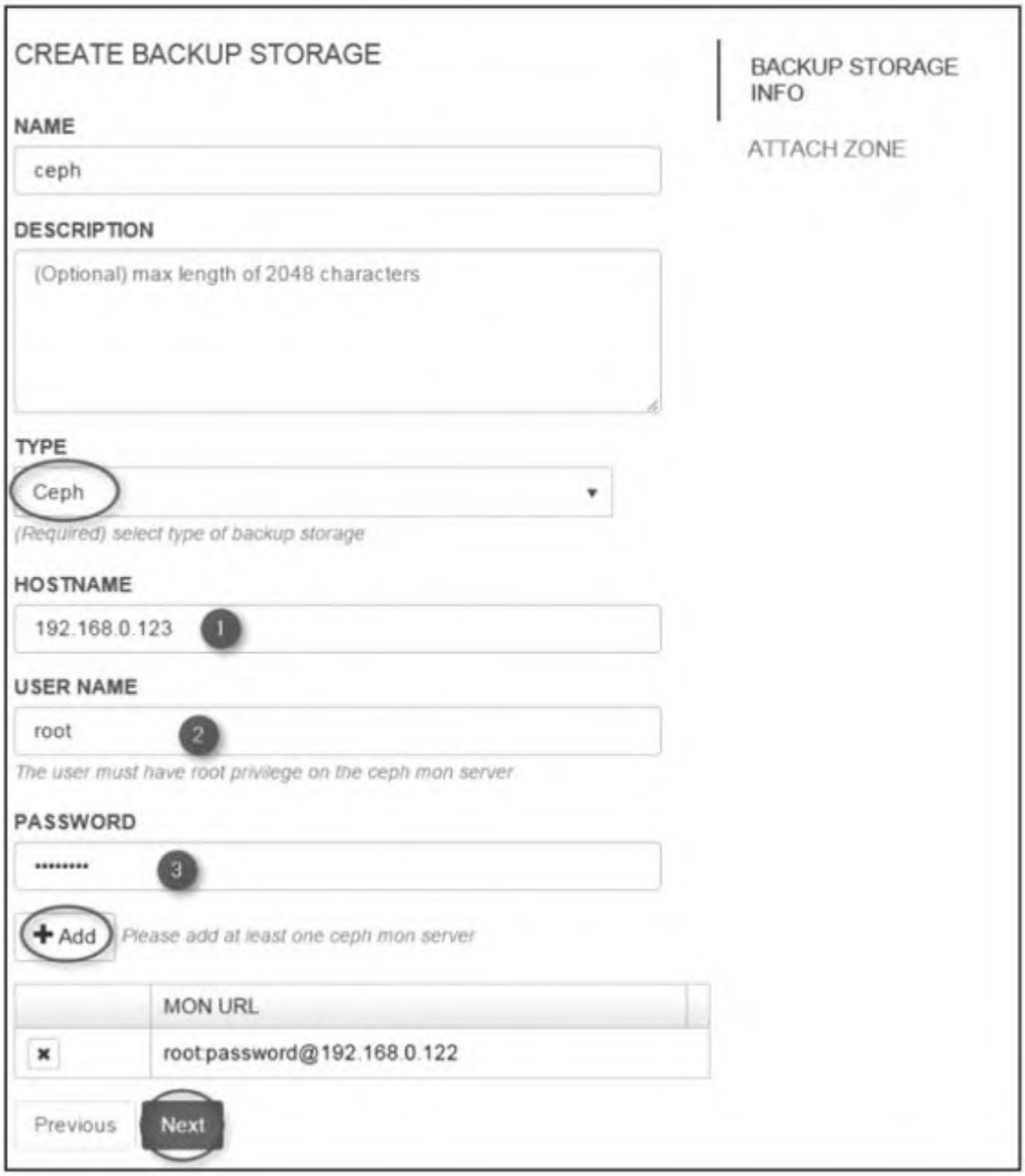

图7-6　填写相关信息

表7-2　AddCephBackupStorage命令支持的参数

name	定义备份存储名字
monUrls	Mon 节点的地址，支持多节点
description	备份存储信息描述
poolName	备份存储指定目标存储池
resourceUuid	指定备份存储 UUID
systemTags	定义系统标签
userTags	定义用户标签
timeout	定义监控检测超时时间

其中，粗体标记为必需参数。

上文中在Web界面添加Ceph存储的例子，可通过以下命令行执行，效果相同。

```
>>>AddCephBackupStorage name=ceph
monUrls=root:password@192.168.0.123,root:pa-
ssword@192.168.0.124,root:password@192.168.0.125
```

在以上创建备份存储过程中，ZStack会生成唯一UUID作为备份存储池，并在存储池名字开头标记识别用途。用户可以预先创建特定的Ceph存储池，作为ZStack的备份存储，在Mon节点预先执行：

```
ceph osd pool create backup-pool 256 256
```

在ZStack-cli命令行执行以下命令行：

```
>>>AddCephBackupStorage name=ceph
monUrls=root:password@192.168.0.123,root:password@192.168.0.
124,root:password@192.168.0.125 poolName=backup-pool
```

主存储和备份存储添加完成后，可通过Web界面或者CLI命令操作磁盘卷。其相关的CLI命令如下。

```
AttachDataVolumeToVm BackupDataVolume BackupVolumeSnapshot
ChangeVolumeState CreateDataVolume
CreateDataVolumeFromVolumeSnapshot
CreateDataVolumeFromVolumeTemplate
CreateDataVolumeTemplateFromVolume
CreateRootVolumeTemplateFromRootVolume
CreateRootVolumeTemplateFromVolumeSnapshot
CreateVolumeSnapshot DeleteDataVolume
DeleteVolumeSnapshot DeleteVolumeSnapshotFromBackupStorage
DetachDataVolumeFromVm
GetDataVolumeAttachableVm GetVmAttachableDataVolume
GetVolumeFormat
GetVolumeSnapshotTree QueryVolume QueryVolumeSnapshot
QueryVolumeSnapshotTree RevertVolumeFromSnapshot
UpdateVolume
UpdateVolumeSnapshot
```

读者查阅ZStack用户CLI手册，可以了解关于磁盘卷的操作：

http://zstackdoc.readthedocs.org/en/latest/ 。

7.5 Ceph提供iSCSI存储

1.iSCSI介绍

1998年，iSCSI由IBM公司和Cisco公司开发，允许在硬件设备、IP协议上层运行SCSI指令集（SCSI over TCP）。iSCSI最大的特点就是，可以实现在IP网络上运行SCSI协议，使其能够在100/1000/10000Mbps的以太网上进行传输。目前，iSCSI技术已经非常流行，经常被应用于高可用集群、数据库集群和虚拟化等领域。2003年，IETF（InternetEngineering Task Force，互联网工程任务组）接收iSCSI成为一项存储访问标准。

iSCSI的工作过程如下。

• iSCSI主机应用程序发出数据读写请求，操作系统会生成一个相应的SCSI命令。

• 该SCSI命令在iSCSI initiator层被封装成iSCSI消息包并通过TCP/IP传送到存储设备的以太网口。

• 存储设备的以太网口的iSCSI target层会解开iSCSI消息包，获得SCSI命令的内容，存储设备获得的SCSI命令，传送给SCSI设备执行。

• 存储设备执行SCSI命令后的响应，在经过iSCSI target层时被封装成iSCSI响应PDU，通过以太网传送给主机的iSCSI initiator层。

• iSCSI initiator会从iSCSI响应PDU里解析出SCSI响应并传送给操作系统，操作系统再响应给应用程序。

iSCSI技术优点和成本优势的主要体现包括以下几个方面。

1）硬件成本低： 构建iSCSI存储网络，除了存储设备外，交换机、线缆和接口卡都是标准的以太网配件，价格相对来说比较低廉。同时，iSCSI还可以在现有的网络上直接安装，并不需要更改企业的网络体系，这样可以最大程度地节约投入。

2）维护方便： 对iSCSI存储网络的管理，实际上就是对以太网设备的管理。当iSCSI存储网络出现故障时，问题定位及解决也会因为以太网的普及而变得容易。

3）扩展性强： 对于已经构建的iSCSI存储网络来说，增加iSCSI存储设备和服务器都将变得简单，且无需改变网络的体系结构。

4）带宽和性能： iSCSI存储网络的访问带宽依赖以太网带宽。随着千兆以太网的普及和万兆以太网的应用，iSCSI存储网络会达到甚至超过FC存储网络的带宽和性能。

5）突破距离限制： iSCSI存储网络使用的是以太网，因而在服务器和存储设备的空间布局上的限制就会少很多，甚至可以跨越国家和地区。

2.Linux下的iSCSI Initor和Target

Open-iSCSI（http://www.open-iscsi.org/ ）是比较常用的Linux iSCSI Initor工具，担当iSCSI客户端角色。类似Rados Gateway方式，通过iSCSI Gateway方法实现Ceph/RBD提供iSCSI协议存储访问，如图7-7所示。

根据前面章节内容可知，Ceph RBD提供两种场景的访问方式：内核态Kernel RBD和用户态Librbd实现读写访问。

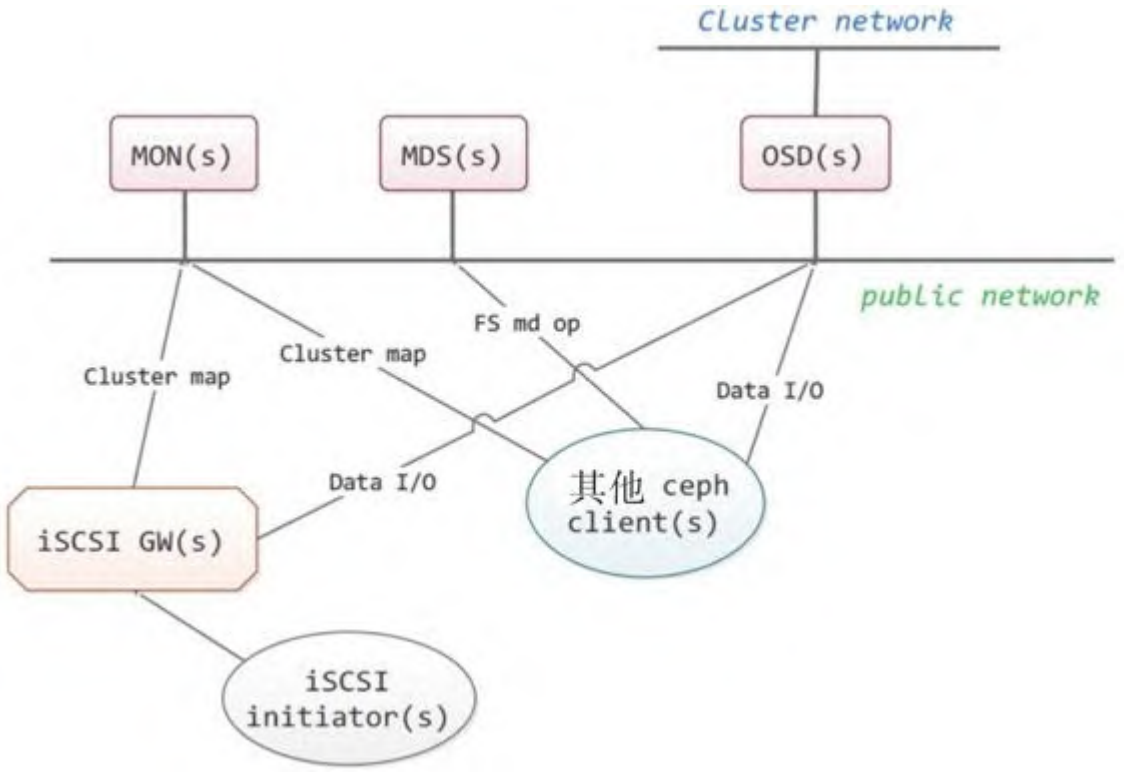

图7-7　iSCSI网关与Ceph存储通信逻辑图

基于内核态Kernel RBD实现iSCSI Target。目前GNU/Linux常见的iSCSI Target都支持块设备作为Target的后端存储，可以让Ceph的内核态RBD客户端将块设备映射到本地，然后选择合适的iSCSI Target映

射这个RBD块设备作为LUN导出。基于用户态Librbd接口实现iSCSI Target。除了iSCSI Target访问本地块设备外，另一种方案是让iSCSI Traget通过用户态Librbd接口实现数据读写功能。

在众多实现iSCSI对接Ceph/RBD的方案中，Tgt（https://github.com/fujita/tgt ）支持基于用户态Librbd实现后端存储的访问，实现过程较为简单。

3.Tgtd+Ceph部署

一般情况下，Tgt作为iSCSI Gateway需要独立部署，而且考虑高可用性，还需要两个或两个以上的iSCSI Gateway。由于服务器硬件条件的限制，可以考虑把Tgt部署在Mon节点上面，这样Mon节点服务器充当了iSCSI Gateway。以下是Tgt在CentOS 7.2系统上的部署流程。

1）准备编译环境。

```
[root@ceph-mon1 ~]# yum -y rpm-build gcc ceph librbd1-devel
```

2）获取Tgt。

```
[root@ceph-mon1 ~]# git clone
https://github.com/fujita/tgt.git
```

3）进入目录后编辑。

```
[root@ceph-mon1 tgt]# vim Makefile
...
# Export the feature switches so sub-make knows about them
export ISCSI_RDMA export CEPH_RBD = 1
```

```
export GLFS_BD
export SD_NOTIFY
...
[root@ceph-mon1 tgt]# vim scripts/tgtd.spec
...
%{_mandir}/man5/*
%{_mandir}/man8/*
%{_initrddir}/tgtd
/usr/lib/tgt/backing-store/bs_rbd.so
```

```
/etc/bash_completion.d/tgt
%attr(0600,root,root) %config(noreplace)
/etc/tgt/targets.conf
...
```

4）编译。

```
[root@ceph-mon1 tgt]# make rpm
```

5）生成安装文件。

```
[root@ceph-mon1 tgt]# ls pkg/RPMS/x86_64/
scsi-target-utils-1.0.62-v1.0.62.x86_64.rpm scsi-target-
utils-debuginfo-1.0.62-v1.0.62.x86_64.rpm
[root@ceph-mon1 tgt]#
```

6）在3个MON节点安装Tgt rpm包。

```
[root@ceph-mon1 tgt]# rpm -ivh pkg/RPMS/x86_64/scsi-target-
utils-1.0.62-v1.0.62.x86_64.rpm --force
```

7）在Ceph创建块设备。

```
[root@ceph-mon1 ~]# rbd create pool1/image1 --size 200 --
image-format 2
```

8）安装完成后，在每个Mon服务器上都添加Ceph/RBD配置文件。

```
[root@ceph-mon1 ~]# cat /etc/tgt/conf.d/ceph.conf
<target iqn.2015-12.rbd.test.com:iscsi-01>
driver iscsi
bs-type rbd
backing-store pool1/image1
</target>
[root@ceph-mon1 ~]#
```

9）启动Tgt服务。

```
[root@ceph-mon1 ~]# service tgtd start
```

10）在客户端服务器上安装iSCSI Initor程序。

```
[root@iscsi-1 ~]# yum -y install iscsi-initiator-utils
```

11）扫描iSCSI Target。

```
[root@ceph-mon1 ~]# iscsiadm -m discovery -t sendtargets -p
10.89.13.71
10.89.13.71:3260,1 iqn.2015-12.rbd.test.com:iscsi-01
[root@ceph-mon1 ~]# iscsiadm -m discovery -t sendtargets -p
10.89.13.72
10.89.13.72:3260,1 iqn.2015-12.rbd.test.com:iscsi-01
```

```
[root@ceph-mon1 ~]# iscsiadm -m discovery -t sendtargets -p
10.89.13.73
10.89.13.73:3260,1 iqn.2015-12.rbd.test.com:iscsi-01
```

12）登录iSCSI Target。

```
iscsiadm -m node -T iqn.2015-12.rbd.test.com:iscsi-01 -p
10.89.13.71 --login
iscsiadm -m node -T iqn.2015-12.rbd.test.com:iscsi-01 -p
10.89.13.72 --login
iscsiadm -m node -T iqn.2015-12.rbd.test.com:iscsi-01 -p
10.89.13.73 --login
```

13）发现本地设备。

```
 [root@iscsi-1 ~]# fdisk -l
Disk /dev/vda: 53.7 GB, 53687091200 bytes, 104857600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x00099ff6
Device Boot Start End Blocks Id System
/dev/vda1 * 2048 96258047 48128000 83 Linux
/dev/vda2 96258048 104857599 4299776 82 Linux swap / Solaris
Disk /dev/sda: 209 MB, 209715200 bytes, 409600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4194304 bytes
I/O size (minimum/optimal): 4194304 bytes / 4194304 bytes
Disk /dev/sdb: 209 MB, 209715200 bytes, 409600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4194304 bytes
I/O size (minimum/optimal): 4194304 bytes / 4194304 bytes
Disk /dev/sdc: 209 MB, 209715200 bytes, 409600 sectors
Units = sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 4194304 bytes
I/O size (minimum/optimal): 4194304 bytes / 4194304 bytes
```

14）配置多路径。

```
[root@iscsi-1 ~]# yum -y install device-mapper-multipath
[root@iscsi-1 ~]# vim /etc/multipath.conf
defaults {
user_friendly_names yes
udev_dir /dev
path_grouping_policy multibus
failback immediate
no_path_retry fail
}
[root@iscsi-1 ~]# service multipathd start
[root@iscsi-1 ~]# multipath -ll
mpatha (360000000000000000e00000000010001) dm-0 IET
,VIRTUAL-DISK
size=200M features='0' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=1 status=active
|- 2:0:0:1 sda 8:0 active ready running
|- 3:0:0:1 sdb 8:16 active ready running
`- 4:0:0:1 sdc 8:18 active ready running
```

至此，客户端主机通过Tgtd间接实现将数据存放到Ceph中。

4. 应用场景

iSCSI是较为广泛的共享块存储设备协议。Ceph通过Tgtd方式实现将RBD块存储转译为iSCSI存储访问协议，可以提供VMware ESXi、Citrix XenServer、Microsoft Hyper-V、oVirt和Oracle VM等虚拟化场景，也可以提供如Oracle RAC和Microsoft MSCS等高可用集群数据库场景。

7.6 本章小结

本章阐述了Ceph的块存储RBD服务，向读者介绍了RBD作为KVM虚拟机的云硬盘对接原理，包括OpenStack、CloudStack和ZStack通过Ceph RBD承载其云硬盘的配置过程。同时，也阐述了Ceph RBD服务通过Tgt实现iSCSI协议网关，提供iSCSI存储，在不需要修改应用软件逻辑的前提下，提供更广泛的共享块存储服务。

第8章　对象存储——云盘与RGW异地灾备

对象存储服务基于HTTP协议，在互联网以及移动互联网中有着得天独厚的优势，特别是存储多媒体数据（如图片、音频、视频等）。在实际应用方面，大家熟知的各类云盘、图片和音视频云存储服务，基本上都是利用对象存储技术，因此对象存储早已经遍布在各类互联网服务中，成为互联网不可分割的一部分。

8.1 网盘方案：RGW与OwnCloud的整合

在过去的几年里，网盘服务（例如Dropbox、Box、Google Drive等）已经变得非常流行。通过使用Ceph，能够使用任何基于S3或Swift的前端应用来部署本地的（on-premise）网盘服务。本节将会讲述Ceph S3对接OwnCloud网盘服务的方法。

为了对接OwnCloud，首先需要一个已有的Ceph集群，还需要一个能通过S3访问Ceph存储的RGW实例，OwnCloud环境架构如图8-1所示。

在本节中我们将使用名为ceph-1的radosgw实例来创建文件同步和共享服务。同时会使用本地的DNS服务，此服务配置在rgw-nodes上，支持S3子域

（subdomain）访问ceph-1 RGW实例。当然，也可以使用任何能够为ceph-1做子域解析的DNS服务器。

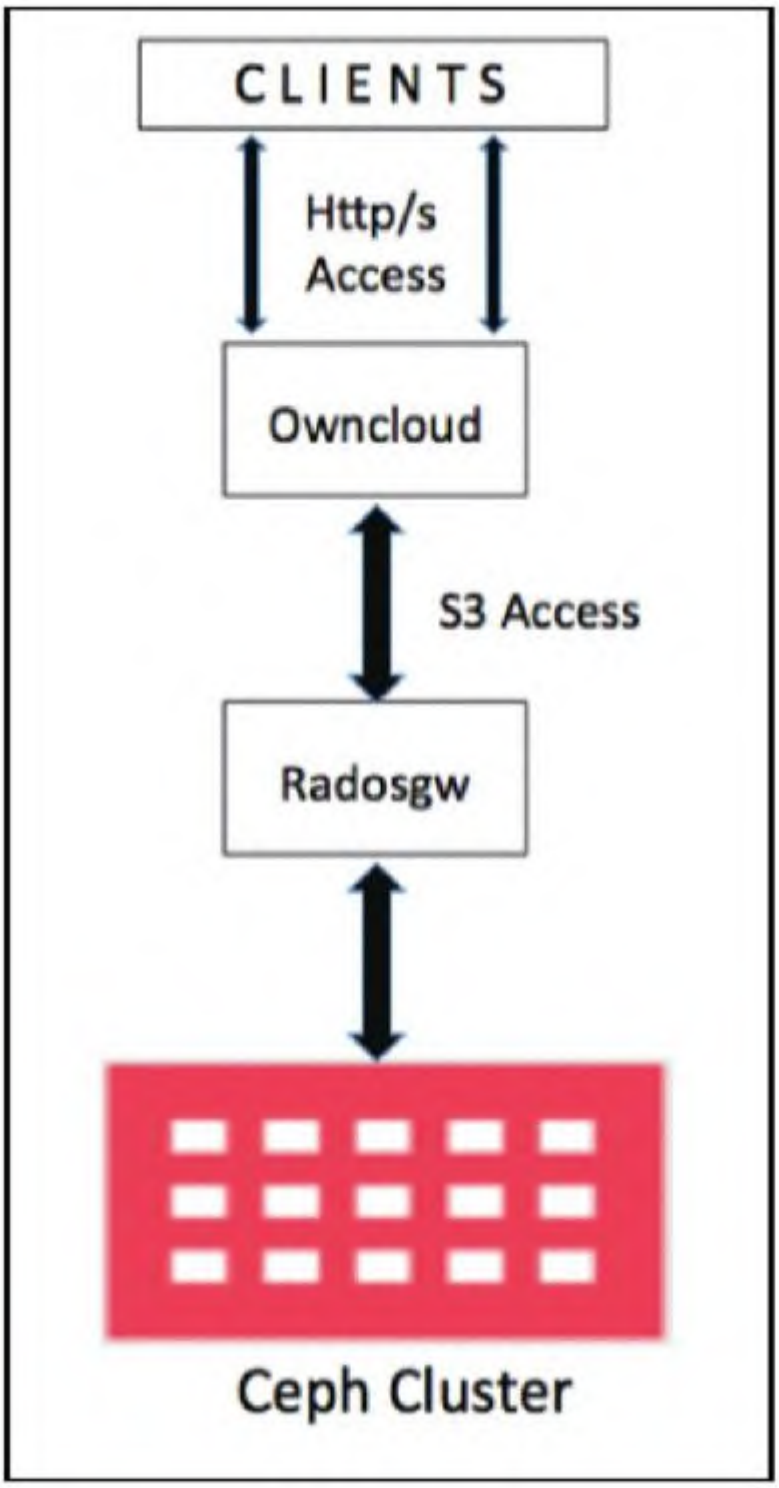

图8-1　OwnCloud环境架构

1）登录到rgw-node1，该节点也是DNS服务器，可以使用如下内容来创建/var/named/ceph-1.cephchina.com文件。

```
@ 86400 IN SOA cephchina.com. root.cephchina.com. (
20091028 ; serial yyyy-mm-dd
10800 ; refresh every 15 min
3600 ; retry every hour
3600000 ; expire after 1 month +
86400 ); min ttl of 1 day
@ 86400 IN NS cephchina.com.
@ 86400 IN A 192.168.1.107
* 86400 IN CNAME @
```

2）配置ceph-1使用DNS服务器。将rgw-node1的地址更新到/etc/resolve.conf文件，ping任意一个子域，它会被解析到ceph-1的地址。

3）确保ceph-1节点可以通过S3连接到Ceph存储集群。我们已经创建了名为us-east的用户，现在通过s3cmd命令来使用它的access_key和secret_key。

①安装s3cmd。

```
# yum install -y s3cmd
```

②配置S3cmd，提供XNKOST8WXTMWZGN29NF9作为access_key和7VJm8uAp71xKQZkjoPZmHu4sACA1SY8jTjay9dP5作为secret_key。

```
# s3cmd -




configure
```

③编辑/root.s3cmd中的host信息。

```
host_base = ceph-1.cephchina.com:7480
host_bucket = %(bucket)s.ceph-1.cephchina.com:7480
```

④测试s3cmd连接。

```
# s3cmd ls
```

⑤为OwnCloud创建用于存放对象的S3 bucket。

```
# s3cmd mb s3://owncloud
```

4）接下来，我们将安装OwnCloud，它将会为我们提供文件同步和共享服务的前端/用户界面。

①环境准备。

```
# yum install httpd mod_ssl php php-gd php-xml php-mysql
php-mbstring
```

②配置OwnCloud的系统源。

```
# cd /etc/yum.repos.d/
# wget http://download.opensuse.org/repositories/
isv:ownCloud:community/CentOS_CentOS-6/
isv:ownCloud:community.repo
# rpm -Uvh
http://download.fedoraproject.org/pub/epel/6/x86_64/epel-
release-6-8.noarch.rpm
```

③因为这是测试环境，所以禁用防火墙。

```
# systemctl disable firewalld
# systemctl stop firewalld
```

④从网页浏览器上访问OwnCloud的网页：http://192.168.*.*/owncloud/ ，并创建密码为owncloud的管理账号，如图8-2所示。

⑤第一次登录和下图界面相似，如图8-3所示。

5）配置OwnCloud来将Ceph用作一种S3外部存储。

①使用OwnCloud管理账号，依次单击Files→Apps→Not enabled→External Storage，单击Enable it，就可以激活外部存储。

②然后配置其外部存储使用Ceph。选择OwnCloud用户，选择admin，再回到左边的面板上，选择“External Storage”。

③还可以先选择Enable User External Storage，然后移动到Amazon S3 and compliant→Add Storage→Amazon S3 and Compliant来配置Amazon S3和其他兼容的存储。

图8-2　创建管理账号

图8-3　首次登录界面

6）输入Ceph radosgw用户详细信息，包括access key、secret key和hostname（见图8-4）：

• Folder name（文件夹），输入想在OwnCloud文件页面上显示的Folder name（文件夹名称）。

• Access key，输入S3 Access Key：XNKOST8WXTMWZGN29NF9。

• Secret key，输入S3 Secret key：7VJm8uAp71xKQZkjoPZmHu4sACA1SY8jTjay9dP5″。

• Bucket，输入我们在步骤3中创建的S3 Bucket名。

• Hostname（主机名），输入ceph-1.cephchina.com作为主机名。

• Port（端口），输入7480。

• Region（可选的），输入US（可选的）。

• Available for，输入owncloud（可选的）。

图8-4　输入Ceph radosgw用户详细信息

7）只要您完成了前面步骤，OwnCloud就可以通过S3连接到Ceph集群了，您将会在Folder name（文件夹名）前面看到一个绿色的圆圈，如图8-4所显示的那样。

8）接下来，通过OwnCloud网页用户界面上传你的文件。导航至Files→External Storage，单击ceph-s3来上传文件或目录。

9）切换至ceph-1节点，执行s3cmd ls s3：//owncloud命令来验证，这些文件已经被添加到Ceph存储集群中了。你将会看到从OwnCloud网页用户界面上传的文件。

8.2 RGW的异地同步方案

通过在单个Ceph集群之上搭建RGW服务，可以很轻松地实现一套基于HTTP标准的对象存储解决方案，但是对象存储服务一般都是面向互联网一类的应用，一方面互联网应用要求较高的可靠性，另一方面还需要最大可能地跨越地域限制去提供高速稳定的接入服务。

而RGW异地同步方案，刚好就是为了解决互联网服务的上述需求的。一方面在多个地理位置存放数据实现服务的高可靠和高可用，另一方面借助DNS负载均衡、CDN等成熟技术手段，提供近端访问，从而实现客户端的高速接入。

8.2.1 异地同步原理与部署方案设计

配置RGW服务之前，需要清楚以下几个基本的逻辑概念。以下概念及操作均基于Ceph 0.94版本，请读者注意。

• Region：一般用来代表逻辑上的地理区域（比如省会、国家等较大规模的地理范围），一个Region可以包含一个或多个Zone。如果一个Ceph集群隶属于多个Region，则必须指定其中一个Region为Master Region。

• Zone：指一个或多个RGW服务实例的逻辑组合，每个Region需要指定一个Master Zone来负责处理所有来自客户端的请求。也就是说，写对象操作只能在Master Zone进行（可读写），读对象操作可以在其他的Zone中进行。但是读者需要注意的是，目前RGW并未

设置任何策略来阻止除Master Zone以外的Zone进行写入操作，请务必遵循规范。

• 数据同步：实现多个Ceph集群之间的对象数据的同步（对象数据可以简单理解为Bucket内存放的object数据）。

• 元数据同步：实现多个Ceph集群之间的对象元数据信息的同步（元数据信息可以简单理解为用户Uid、Email、access-key、secret-key等一类的元数据信息）。

• RGW服务实例：这个概念相对来讲比较抽象，可以简单理解为一个RGW服务实例对应一个在操作系统上运行的RGW服务。确切来讲，一个RGW服务实例应该是对应一组Region和Zone配置信息。

• 同步日志：记录各个RGW服务实例的数据和元数据的变更情况。

• 同步代理Agent：同步代理Agent是一组同步服务，通过轮询的方式比较多个RGW服务实例之间的同步日志，从而得到需要同步的数据列表，之后根据列表调用RGW服务的相关API来实现数据和元数据的同步。

（1）异地同步原理介绍

要实现RGW异地同步，首先需要将原本孤立零散的RGW服务，按照一定逻辑组成Region和Zone，从而打破物理地域的限制，在逻辑上形成统一的命名空间。之后启动同步代理（Agent），通过轮询方式比较多个RGW服务实例之间的同步日志，最终按照Region和Zone的逻辑关系，将同步日志中的差异部分数据和元数据按照一定规则进行同步。

（2）异地同步部署方案设计

在单个Ceph集群可以运行多个RGW服务实例，而这些实例可以分属多个不同的Region和Zone，因此，部分读者在进行同步方案选型的第一步会觉得比较迷惑，以下是笔者的一些实际项目经验，供读者参考。

Region划分一般以国家或者大洲为基本单位，比如中国，简称cn。亚洲，简称Asia。

Zone划分一般以单个Ceph集群为基本单位，比如广州数据中心内有多个Ceph集群，可以按照集群分为gz-zone1、gz-zone2，另外一个数据中心分布在上海，集群分别为sh-zone1、sh-zone2。

一般来讲，考虑到Master Zone可同时进行读写，而其他Zone只能进行读取，所以一般都是将一个较大地理区域划分成一个Region，比如中国，简称cn。然

后在这个地理区域内根据业务需求、网络带宽和延迟等选定一个数据中心作为一个Master Zone，比如北京某个多线数据中心bj-zone1作为cn这个Region的Master Zone。之后考虑到异地数据的备份与近端访问，选定在上海的某个数据中心sh-zone1作为其他Zone，实现bj-zone1→sh→zone1的数据与元数据备份。

需要注意的是，无论是Region还是Zone，在RGW里面都是一组配置规则，目前版本都以json格式进行读写和存放。其中Region的配置信息，标示了该Region由哪些Zone组成，Master Zone是哪个Zone等，以下为示例。

```
{
    "name": "cn", #Region 名称
```

```
    "api_name": "cn",
    "is_master": "true",#作为
```

```
master Region
    "endpoints": [
        "http:\/\/host1.ceph.work:80\/"
    ],
    "hostnames": [],
    "master_zone": "cn-zone1",
    "zones": [
        {
            "name": "cn-zone1",
            "endpoints": [
                "http:\/\/host1.ceph.work:80\/"
            ],
            "log_meta": "true", #记录元数据日志
```

```
            "log_data": "true", #记录数据日志
```

```
            "bucket_index_max_shards": 8   #bucket 分片数量
```

```
    },
    {
        "name": "us-zone1",
        "endpoints": [
            "http:\/\/host2.ceph.work:80\/"
        ],
        "log_meta": "true",
        "log_data": "true",
        "bucket_index_max_shards": 8
    }
],
"placement_targets": [   #数据存放策略
```

```
    {
        "name": "default-placement", #策略名称
```

```
        "tags": []
    }
],
"default_placement": "default-placement" #设置默认策略
```

}

而Zone的配置，包含了指定数据和元数据存放的Pool等，以下为示例。

"domain_root": ".cn-zone1.rgw.domain",#Region和

Zone配置信息存储池

"control_pool": ".cn-zone1.rgw.control",#Region和

Zone配置信息存储池

"gc_pool": ".cn-zone1.rgw.gc",#资源回收存储池

"log_pool": ".cn-zone1.log",#日志存储池

"intent_log_pool": ".cn-zone1.intent-log",#内部同步日志存储池

"usage_log_pool": ".cn-zone1.usage",#用户用量信息存储池

```
    "user_keys_pool": ".cn-zone1.users",#用户
```

key信息存储池

```
    "user_email_pool": ".cn-zone1.users.email",#用户
```

email信息存储池

```
    "user_swift_pool": ".cn-zone1.users.swift",#swift用户信息存
储池

    "user_uid_pool": ".cn-zone1.users.uid", #用户信息存储池

    "system_key": {#同步数据所需的

key认证信息

        "access_key": "masteraccesskey",
        "secret_key": "mastersecretkey"
    },
    "placement_pools": [
```

```
        {
            "key": "default-placement",#数据存放策略名称


            "val": {
                "index_pool": ".cn-
zone1.rgw.buckets.index",#元数据存储池


                "data_pool": ".cn-zone1.rgw.buckets",#数据存储
池


                "data_extra_pool": ".cn-
zone1.rgw.buckets.extra"#附加信息存储池
```

```
                    }
                }
            ]
        }
```

8.2.2　Region异地同步部署实战

8.2..1节讲述了RGW异地同步原理以及部署方案的设计，本节将讲述进行异地部署的方法。

（1）Region和Zone的划分

本次用例由两个Ceph集群组成，每个Ceph集群组成一个Zone，Zone的名称分别为Zone Master和Zone Slave，其中Region由这两个Zone组成，名称为cn。最终效果是实现Zone Master中的数据和元数据同步到Zone Slave中。构架组成如图8-5所示。

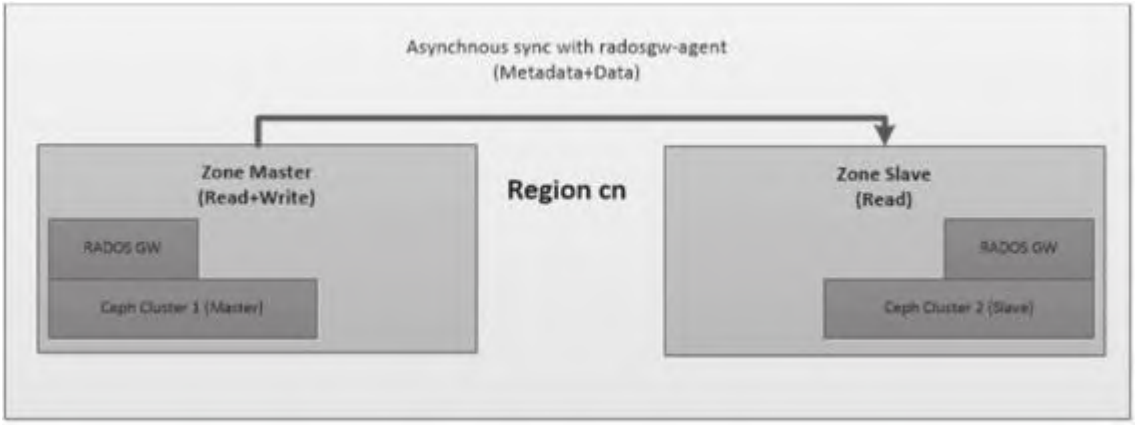

图8-5　架构组成

需要注意，所有RGW服务都依赖DNS解析，本用例的DNS配置信息如表8-1所示。

表8-1　DNS记录表

Zone name	Endpoint	IP	主机名
Master zone	host1.ceph.work	192.168.100.100	host1
Slave zone	host2.ceph.work	192.168.100.200	host2

（2）安装radosgw和radosgw-agent

注意，以下操作都在debian8 64bit系统下进行。

```
apt-get install radosgw
aptitude install python-argparse  python-setuptools python-
boto  python-yaml
apt-get install radosgw-agent #agent只需要在
```

```
master zone安装
```

（3）用户及keyring配置

以下操作在Master Zone对应的RGW服务节点上进行。

```
sudo ceph-authtool --create-keyring
/etc/ceph/ceph.client.radosgw.keyring
sudo chmod +r /etc/ceph/ceph.client.radosgw.keyring
sudo ceph-authtool /etc/ceph/ceph.client.radosgw.keyring -n
client.radosgw.master --gen-key
sudo ceph-authtool -n client.radosgw.master --cap osd 'allow
rwx' --cap mon 'allow rwx'
/etc/ceph/ceph.client.radosgw.keyring
sudo ceph-authtool /etc/ceph/ceph.client.radosgw.keyring -n
client.radosgw.slave --gen-key
sudo ceph-authtool -n client.radosgw.slave --cap osd 'allow
rwx' --cap mon 'allow rwx'
/etc/ceph/ceph.client.radosgw.keyring
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add
client.radosgw.master -i
/etc/ceph/ceph.client.radosgw.keyring
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add
client.radosgw.slave -i
/etc/ceph/ceph.client.radosgw.keyring
#完成本轮操作以后将生成的
```

/etc/ceph/ceph.client.radosgw.keyring复制到

slave相同目录

（4）新建Pool

以下操作在Master Zone对应的RGW服务节点上进行，要注意新建Pool的时候，PG数量需要根据实际情况来设置。

```
ceph osd pool create .cn.rgw.root 64 64
ceph osd pool create .master.rgw.root 64 64
ceph osd pool create .master.rgw.control 64 64
ceph osd pool create .master.rgw.domain 64 64
ceph osd pool create .master.rgw.buckets.extra 64 64
ceph osd pool create .master.intent-log 64 64
ceph osd pool create .master.usage 512 512
ceph osd pool create .master.users 64 64
ceph osd pool create .master.log 512 512
```

```
ceph osd pool create .master.users.email 64 64
ceph osd pool create .master.users.swift 64 64
ceph osd pool create .master.users.uid 64 64
ceph osd pool create .master.rgw.buckets 8192 8192
ceph osd pool create .master.rgw.buckets.index 2048 2048
ceph osd pool create .master.rgw.gc 64 64
#到
```

slave执行同样的

pool新建操作，注意对

PG容量的规划和

rule的对应。

（5）配置RGW服务

以下操作在Master Zone对应的RGW服务节点上进行，要根据实际情况进行修改。

在

/etc/ceph/ceph.conf添加以下内容

```
[client.radosgw.master]
 rgw dns name = s3.ceph.work
 rgw socket path = /home/ceph/var/run/ceph-
client.radosgw.master.sock
 host = host
 rgw region = cn
```

```
 rgw region root pool = .cn.rgw.root
 rgw zone = master
 rgw zone root pool = .master.rgw.root
 keyring = /etc/ceph/ceph.client.radosgw.keyring
log file = /home/ceph/log/radosgw.master.log
 rgw print continue = false
 rgw content length compat = true
 rgw ops log rados = false
 rgw enable usage log = true
```

（6）新建Region和Zone配置文件

以下操作在Master Zone对应的RGW服务节点上进行，要根据实际情况修改相应配置。

新建

cn.json文件，内容如下，注意

master_zone内容

```
{
    "name": "cn",
    "api_name": "cn",
    "is_master": "true",
    "endpoints": [
        "http:\/\/host1.ceph.work:80\/"
    ],
    "hostnames": [],
    "master_zone": "master",
    "zones": [
        {
            "name": "master",
            "endpoints": [
                "http:\/\/host1.ceph.work:80\/"
            ],
            "log_meta": "true",
            "log_data": "true",
            "bucket_index_max_shards": 8
        },
        {
            "name": "slave",
            "endpoints": [
                "http:\/\/host2.ceph.work:80\/"
            ],
            "log_meta": "true",
            "log_data": "true",
            "bucket_index_max_shards": 8
        }
    ],
    "placement_targets": [
        {
            "name": "default-placement",
            "tags": []
        }
    ],
    "default_placement": "default-placement"
}
#新建
```

master.json文件，内容如下，注意记录

access_key和

secret_key的预留内容

```
{
    "domain_root": ".master.rgw.domain",
    "control_pool": ".master.rgw.control",
    "gc_pool": ".master.rgw.gc",
    "log_pool": ".master.log",
    "intent_log_pool": ".master.intent-log",
    "usage_log_pool": ".master.usage",
    "user_keys_pool": ".master.users",
    "user_email_pool": ".master.users.email",
    "user_swift_pool": ".master.users.swift",
```

```
    "user_uid_pool": ".master.users.uid",
    "system_key": {
        "access_key": "masteraccesskey",
        "secret_key": "mastersecretkey"
    },
    "placement_pools": [
        {
            "key": "default-placement",
            "val": {
                "index_pool": ".master.rgw.buckets.index",
                "data_pool": ".master.rgw.buckets",
                "data_extra_pool":
".master.rgw.buckets.extra"
            }
        }
    ]
}
```

（7）导入Region和Zone的配置

以下操作在Master Zone对应的RGW服务节点上进行，要根据实际情况修改相应配置。

```
#导入
```

```
region信息
```

```
radosgw-admin region set --infile cn.json --name
client.radosgw.master
radosgw-admin region default --rgw-region=cn --name
client.radosgw.master
#导入
```

```
zone信息
```

```
radosgw-admin zone set --rgw-zone=master --infile
master.json --name client.radosgw.master
radosgw-admin regionmap update --name client.radosgw.master
```

（8）新建Agent同步账号

以下操作在Master Zone对应的RGW服务节点上进行，要根据实际情况修改相应配置。

```
radosgw-admin user create --uid="master" --display-
name="master" --name client.radosgw.master --system --
access-key=masteraccesskey --secret=mastersecretkey
radosgw-admin user create --uid="slave" --display-
name="slave" --name client.radosgw.master --system --access-
key=slaveaccesskey
```

（9）配置Nginx服务

以下操作在Master Zone对应的RGW服务节点上进行，要根据实际情况修改相应配置。配置/etc/nginx/conf.d/default.conf内容如下。

```
server {
    listen       80;
    server_name  s3.ceph.work *.s3.ceph.work;
    access_log  /var/log/nginx/log/host.access.log  main;
    #error_page  404              /404.html;
        location / {
        fastcgi_pass_header Authorization;
        fastcgi_pass_request_headers on;
        fastcgi_param QUERY_STRING  $query_string;
        fastcgi_param REQUEST_METHOD $request_method;
        fastcgi_param CONTENT_LENGTH $content_length;
        fastcgi_param CONTENT_LENGTH $content_length;
        if ($request_method = PUT) {
            rewrite ^ /PUT$request_uri;
        }
        include fastcgi_params;
        fastcgi_pass unix:/home/ceph/var/run/ceph-
client.radosgw.master.sock;
        }
        location /PUT/ {
        internal;
        fastcgi_pass_header Authorization;
```

```
        fastcgi_pass_request_headers on;
        include fastcgi_params;
        fastcgi_param QUERY_STRING  $query_string;
        fastcgi_param REQUEST_METHOD $request_method;
        fastcgi_param CONTENT_LENGTH $content_length;
        fastcgi_param  CONTENT_TYPE $content_type;
        fastcgi_pass unix:/home/ceph/var/run/ceph-
client.radosgw.master.sock;
        }
}
```

（10）启动服务与测试

以下操作在Master Zone对应的RGW服务节点上进行，要确定你使用的服务端口是可以通过防火墙的。如果没有打开，把这个端口加入到防火墙的信任列表中。

```
/etc/init.d/nginx restart
/etc/init.d/radosgw start
curl `hostname` #测试
```

s3是否正常

```
nginx -t 检查配置文件是否正常
```

（11）Slave上的用户及keyring配置

以下操作在Slave Zone对应的RGW服务节点上进行，注意keyring文件之前已经在Master Zone生成并拷贝到Slave节点上。

```
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add
client.radosgw.slave -i
/etc/ceph/ceph.client.radosgw.keyring
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add
client.radosgw.master -i
/etc/ceph/ceph.client.radosgw.keyring
```

（12）在Slave上新建Pool

具体请参考之前在Master Zone上面新建Pool的步骤（对应本小节步骤4）。

（13）在Slave上配置RGW服务

在

/etc/ceph/ceph.conf添加以下内容

```
[client.radosgw.slave]
 rgw dns name = s3.ceph.work
 rgw frontends = fastcgi
 host = host2
 rgw region = cn
 rgw region root pool = .cn.rgw.root
 rgw zone = slave
 rgw zone root pool = .master.rgw.root
 keyring = /etc/ceph/ceph.client.radosgw.keyring
 rgw socket path = /home/ceph/var/run/ceph-
client.radosgw.slave.sock
 log file = /home/ceph/log/radosgw.slave.log
 rgw print continue = false
rgw content length compat = true
```

（14）在Slave上配置Region和Zone

以下操作在Slave Zone对应的RGW服务节点上进行，要根据实际情况修改相应配置。

新建

cn.json文件，内容如下

```
{
    "name": "cn",
    "api_name": "cn",
    "is_master": "true",
    "endpoints": [
        "http:\/\/host1.ceph.work:80\/"
    ],
    "hostnames": [],
    "master_zone": "master",
    "zones": [
        {
            "name": "master",
            "endpoints": [
```

```
                "http:\/\/host1.ceph.work:80\/"
            ],
            "log_meta": "true",
            "log_data": "true",
            "bucket_index_max_shards": 8
        },
        {
            "name": "slave",
            "endpoints": [
                "http:\/\/host2.ceph.work:80\/"
            ],
            "log_meta": "true",
            "log_data": "true",
            "bucket_index_max_shards": 8
        }
    ],
    "placement_targets": [
        {
            "name": "default-placement",
            "tags": []
        }
    ],
    "default_placement": "default-placement"
}
#新建
```

slave.json文件，内容如下，注意记录

access_key和

secret_key的预留内容

```
{
    "domain_root": ".master.rgw.domain",
    "control_pool": ".master.rgw.control",
    "gc_pool": ".master.rgw.gc",
    "log_pool": ".master.log",
    "intent_log_pool": ".master.intent-log",
    "usage_log_pool": ".master.usage",
    "user_keys_pool": ".master.users",
    "user_email_pool": ".master.users.email",
    "user_swift_pool": ".master.users.swift",
    "user_uid_pool": ".master.users.uid",
    "system_key": {
        "access_key": "slaveaccesskey",
        "secret_key": "slavesecretkey"
    },
    "placement_pools": [
        {
            "key": "default-placement",
            "val": {
                "index_pool": ".master.rgw.buckets.index",
                "data_pool": ".master.rgw.buckets",
                "data_extra_pool":
".master.rgw.buckets.extra"
            }
        }
    ]
}
```

（15）在Slave上导入Region和Zone配置

#导入

region信息

```
radosgw-admin region set --infile cn.json --name
client.radosgw.slave
radosgw-admin region default --rgw-region=cn --name
client.radosgw.slave
#导入
```

zone信息

```
radosgw-admin zone set --rgw-zone=slave --infile slave.json
--name client.radosgw.slave
radosgw-admin regionmap update --name client.radosgw.slave
```

（16）在Slave上新建同步账号

```
radosgw-admin user create --uid="master" --display-
name="master" --name client.radosgw.slave --system --access-
key=masteraccesskey --secret=mastersecretkey
radosgw-admin user create --uid="slave" --display-
name="slave" --name client.radosgw.slave --system --access-
key=slaveaccesskey --secret=slavesecretkey
```

（17）在Slave上配置Nginx

具体请参考之前在Master Zone上面配置Ngnix的步骤（对应本小节步骤9）。

（18）在Slave上启动并测试服务

具体请参考之前在Master Zone上面启动服务及测试（对应本小节步骤10）。

（19）配置并启动radosgw-agent服务

以下操作在Master Zone对应的RGW服务节点上进行。

#新建

/etc/ceph/radosgw-agent/default.conf，内容如下

```
source: http://host1.ceph.work:80
src_zone: master
src_access_key: masteraccesskey
src_secret_key: mastersecretkey
dest_zone: slave
metadata_only: false #如果设置成
```

true则只同步元数据信息

```
destination: http://host2.ceph.work:80
dest_access_key: slaveaccesskey
dest_secret_key: slavesecretkey
log_file: /var/log/radosgw/radosgw-sync-cn-us.log
/etc/init.d/radosgw-agent -c default.conf start #启动
```

agent服务

（20）测试同步服务

1）测试元数据同步。

①在Master Zone的RGW服务节点上使用命令新建用户。

```
sudo radosgw-admin user create -
```

```
-uid=synctest1 --display-name=synctest1 -
```

```
email=synctest.ceph.work -
```

```
name client.radsogw.master
```

---

②在Slave Zone的RGW服务节点上使用命令查看用户信息是否同步。

```
sudo radosgw-admin user info -
```

```
-uid=synctest1-



name client.radsogw.slave
```

2）测试数据同步。

使用之前新建的synctest1用户，在Master Zone中新建Bucket和Object，之后在Slave Zone中检查相应数据是否同步，具体操作可以参考之前RGW服务章节，此处不再赘述。

8.3 本章小结

本章通过介绍RGW与OwnCloud结合的方案，搭建最简单的网盘服务，让读者可以切身体会到RGW服务在互联网应用的接入优势。然后介绍了RGW异地同步方案，结合笔者自身的实际项目经验，由浅至深地让读者在最短的时间内掌握基本的异地同步基本理论及基本设计思想，并应用到实际项目。本文涉及的RGW异地同步内容还比较基础，加上笔者水平有限，一些理论和实践经验还存在不足，对这一方面感兴趣的读者可以自行查阅官方文档及源码深入学习。

第9章　Ceph硬件选型、性能测试与优化

9.1　需求模型与设计

Ceph是一款开源的分布式存储软件，它的设计决定了它是一种灵活、高效、可靠、廉价的开源存储解决方案，它的优点在于灵活、可扩展、无单点故障、统一接口等。

Ceph的硬件选型需要根据存储需求和使用场景来制定。不同的企业有着不同的存储需求场景。Ceph是一款开源统一存储软件，也就是说，它可以从一个集群中提供文件、块和对象存储。也能够在同一个集群中，针对不同的工作负载，提供不同类型的存储池。这种设计允许企业根据自身需求调整Ceph存储。多种

方法可以定义你的存储需求，图9-1列举了3种存储场景类型。

- 高性能场景： 这种配置的类型亮点在于它在低TCO（ownership的总消耗）下每秒拥有最高的IOPS。典型的做法是使用包含了更快的SSD硬盘、PCIe SSD、NVMe做数据存储的高性能节点。通常用于块存储，也可以用在高IOPS的工作负载上。

- 通用场景： 亮点在于高吞吐量和每吞吐量的低功耗。通用的做法是拥有一个高带宽、物理隔离的双重网络，使用SSD和PCIe SSD做OSD日志盘。这种方法常用于块存储，如果你的应用场景需要高性能的对象存储和文件存储，也可以考虑使用。

- 大容量场景： 亮点在于数据中心每TB存储的低成本，以及机架单元物理空间的低成本。也被称为经济存储、廉价存储、存档/长期存储。通用的做法是使

用插满机械硬盘的密集服务器，一般是36~72台服务器，每台服务器4~6TB的物理硬盘空间。通常用于低功耗、大存储容量的对象存储和文件存储。一个好的备选方案是采用纠删码来最大化存储容量。

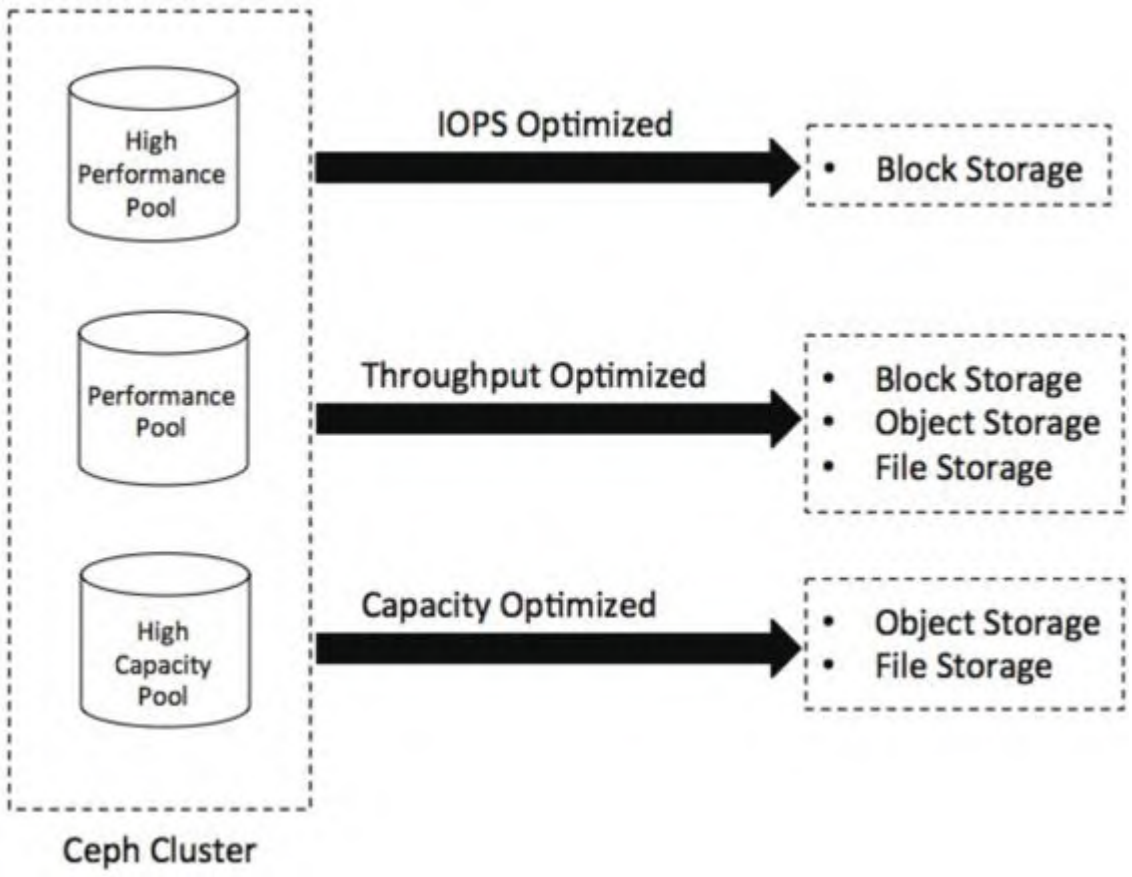

图9-1　三种存储场景类型

9.2 硬件选型

正如前面所提到的，Ceph的硬件选型需要根据环境和存储需求做出选型计划。硬件的类型、网络和集群设计，是你在Ceph集群设计前期要考虑的一些关键因素。Ceph选型没有黄金法则，因为它依赖各种因素，比如预算、性能和容量，或者两种的结合、容错性，以及使用场景。

企业可以根据预算、性能/容量需求、使用场景自由地选择任意硬件。在存储集群和底层基础设施上，他们有完全控制权。另外，Ceph的一个优势是它支持异构硬件。当创建Ceph集群时，你可以混合硬件品牌。例如，可以混合使用来自不同厂家的硬件，比如HP、Dell、Supermicro等，混用现有的硬件可以大大

降低成本。下面说一些常用的关于Ceph硬件选型的方法。

1.CPU

Ceph metadata Server会动态地重新分配负载，它是CPU敏感性的，所以Metadata Server应该有比较好的处理器性能（比如四核CPU）。

Ceph OSD运行RADOS服务，需要通过CRUSH来计算数据的存放位置，复制数据，以及维护Cluster Map的拷贝。通常建议每个OSD进程至少有一个CPU核。可以用下面的公式计算OSD的CPU需求。

```
((CPU sockets * CPU cores per socket * CPU clock speed in GHz) / No.Of OSD) >=1
```

Ceph Monitors简单地维护了Cluster Map的主干信息，所以Monitors是CPU不敏感的。

2.RAM内存

Metadata Servers以及Monitors必须能够快速地提供数据，因此必须有充足的内存（至少每个进程1GB）。OSD在日常操作时不需要过多的内存（如每进程500MB）；但是，在执行恢复操作时，就需要大量的内存（如每进程每TB数据需要约1GB内存）。通常来说，内存越多越好。

3.数据存储

规划数据存储时要考虑成本和性能的权衡。进行系统操作，同时多个后台程序对单个驱动器进行读写操作会显著降低性能。也有文件系统的限制考虑：BTRFS对于生产环境来说不是很稳定，但有能力记录journal和并行的写入数据，相对而言，XFS和EXT4会好一点。

4.网络

建议每台机器最少有两个千兆网卡，现在大多数机械硬盘都能达到100MB/s的吞吐量，网卡能处理所有OSD硬盘总吞吐量，所以推荐最少安装两个千兆网卡，分别用于公网（前端）和集群网络（后端）。集群网络（最好别连接到外网）用于处理由数据复制产生的额外负载，而且可防止DOS攻击，DOS攻击会干扰PG数据，使之在OSD数据复制时不能回到Active+Clean状态。

但是，一般生产环境会建议部署万兆网卡。下面可以算一笔账，假设通过1Gbps网络复制1TB数据需要耗时3小时，而3TB（典型配置）就需要9小时，相比之下，如果使用10Gbps复制时间可分别缩减到20分钟和1小时。所以，一般使用Ceph都会部署万兆网卡。

5.硬盘

Ceph集群的性能很大程度上取决于存储介质的有效选择。应该在选择存储介质之前了解集群的工作负载和性能需求。Ceph使用存储介质有两种方法：OSD日志盘和OSD数据盘。Ceph的每一次写操作分两步处理。当一个OSD接受请求写一个object时，它首先会把object写到acting set中的OSD对应的日志盘，然后发送一个写确认给客户端。很快，日志数据会同步到数据盘。值得注意的是，在写性能上，副本也是一个重要因素。副本因素通常要在可靠性、性能和TCO之间平衡。

6.Ceph OSD日志盘

如果工作环境是通用场景的需求，那么建议使用SSD做日志盘。使用SSD，可以减少访问时间，降低写延迟，大幅提升吞吐量。使用SSD做日志盘，可以对每个物理SSD创建多个逻辑分区，每个SSD逻辑分区（日

志），映射到一个OSD数据盘。通常10~20GB日志大小足以满足大多数场景。如果你有一个更大的SSD，不要忘记为OSD增加filestore的最大和最小的同步时间间隔。

Ceph使用中最常见的两种非易失性快速存储的类型是SATA、SAS SSD、PCIe或NVMe SSD。若想在SATA/SAS SSD之外获得高性能，SSD和OSD的比例应该是1:4。也就是说，4个OSD数据硬盘共享一个SSD。PCIe或者NVMe闪存设备的情况，取决于设备性能，SSD和OSD比例在1:12到1:18之间。也就是说，12~18个OSD数据硬盘共享一个闪存设备。

注意

这里提到的SSD和OSD的比例是非常常见的，能良好工作在大多数场景下。但是，鼓励大家针对具体的工作场景和环境测试SSD/PCIe，得到最好的效果。

使用单个SSD存储多份日志的不足之处是，一旦丢失存储多份日志的SSD，所有的关联这个SSD的OSD就出问题了，很有可能丢失数据。不过，可以通过对日志做RAID1来解决这个问题，但这会增加成本。而且，每GB的SSD的成本近乎10倍于HDD。所以，使用SSD搭建集群，会增加每GB的费用。但是，如果你希望大幅提升性能，那么把钱花在SSD做日志上是值得的。

我们已经学习了许多关于SSD日志的配置，知道这些做法能大幅提升写性能。然而，如果你不注重极度的性能，而且每GB的成本对你来说是一个决定性因素，那么你应该考虑在同一个硬盘驱动上配置日志和数据盘。这意味着，从大容量的机械硬盘中分配少量GB的空间给日志盘，其余容量用于OSD数据。这种配置可能不像SSD单独做日志盘高效，但是每GB存储空间的价格会相当少。

9.3 性能调优

对于Ceph运维人员来说最头痛的莫过于两件事：Ceph调优与Ceph运维。调优是件非常头疼的事情，笔者通过对公开的资料进行分析总结，对分布式存储系统的优化离不开以下几点。

（1）硬件层面

- 硬件规划。
- SSD选择。
- BIOS设置。

（2）操作系统层面

- Linux Kernel。

- 内存。
- Cgroup。

（3）网络层面

- 巨型帧。
- 中断亲和。
- 硬件加速。

（4）Ceph层面

- Ceph参数。
- PG Number调整。
- Ceph参数配置示例。

9.3.1 硬件优化

根据上面说到的对分布式存储系统优化的两点，本节将会从硬件的规划、SSD的选择、机器BIOS设置以及系统内存管理4个方面来介绍对硬件的优化方法。

1.硬件规划

（1）Processor

• ceph-osd进程在运行过程中会消耗CPU资源，所以一般会将每一个ceph-osd进程绑定一个CPU核上。当然如果你使用EC方式，可能需要更多的CPU资源。

• ceph-mon进程并不十分消耗CPU资源，所以不必为ceph-mon进程预留过多的CPU资源。

• ceph-msd也是非常消耗CPU资源的，所以需要提供更多的CPU资源。

（2）内存

ceph-mon和ceph-mds需要2GB内存，每个ceph-osd进程需要1GB内存，当然2GB更好。

（3）网络规划

万兆网络现在基本上是运行Ceph必备的，在网络规划上，也尽量考虑分离Cilent和Cluster网络。

2.SSD选择

硬件的选择也直接决定了Ceph集群的性能，从成本考虑，一般选择SATA SSD作为Journal，Intel SSD DC S3500 Series基本是目前的方案中的首选。400G的规格4K随机写可以达到11000 IOPS。如果在预算足够

的情况下，推荐使用PCIE SSD，这样性能会得到进一步提升，但是由于Journal在向数据盘写入数据时Block后续请求，Journal的加入并未呈现出想象中的性能提升，但是的确会对延迟有很大的改善。

如何确定你的SSD是否适合作为SSD Journal，可以参考SÉBASTIEN HAN的Ceph ： How to Test if Your SSD Is Suitable as a Journal Device?（http://www.sebastien-han.fr/blog/2014/10/10/ceph-how-to-test-if-your-ssd-is-suitable-as-a-journal-device/ ），在这里，他列出了常见的SSD的测试结果，从结果来看SATA SSD，Intel S3500性能表现最好。

3.BIOS 设置

（1）超线程

使用超线程（Hyper-Threading）技术，可以实现在一个CPU物理核心上提供两个逻辑线程并行处理任务，在拥有12个物理核心的E5 2620 v3中，配合超线程，可以达到24个逻辑线程。更多的逻辑线程可以让操作系统更好地利用CPU，让CPU尽可能处于工作状态。

基本做云平台的，VT和HT都是必须打开的，超线程技术（Hyper-Threading，HT）就是利用特殊的硬件指令，把两个逻辑内核模拟成两个物理芯片，让单个处理器都能使用线程级并行计算，进而兼容多线程操作系统和软件，减少了CPU的闲置时间，提高CPU的运行效率，见图9-2。

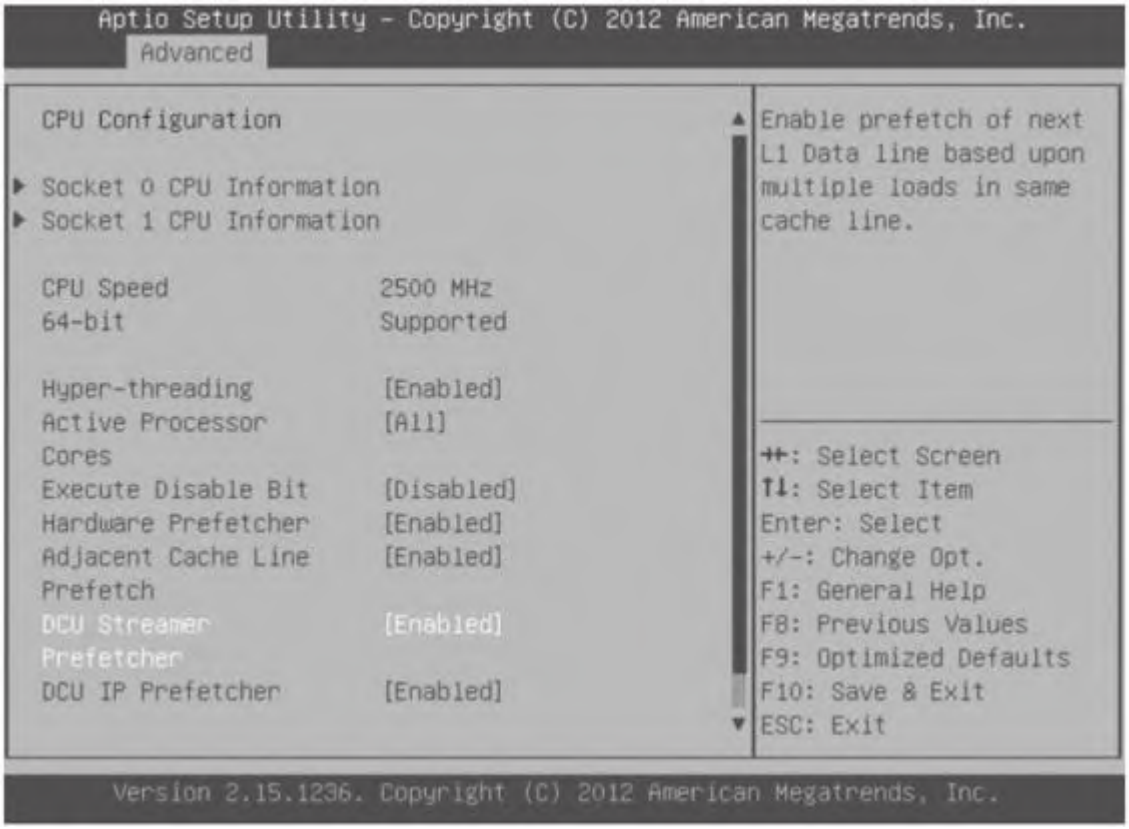

图9-2　打开超线程

（2）关闭节能

很多服务器出于能耗考虑，在出场时会在BIOS中打开节能模式，在节能模式下，CPU会根据机器负载动态调整频率。但是这个动态调频并不像想象中的那么美好，有时候会因此影响CPU的性能，在像Ceph这种需

要高性能的应用场景下，建议关闭节能模式。关闭节能后，对性能还是有所提升的，见图9-3。

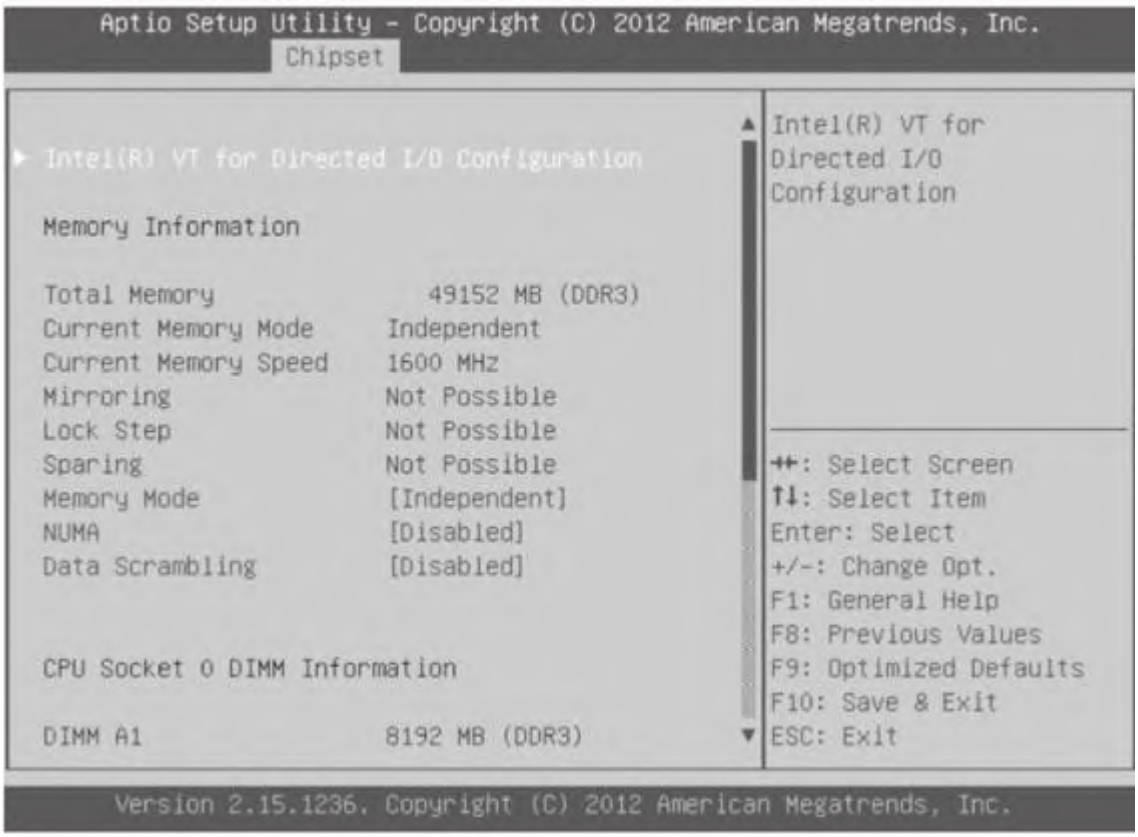

图9-3　关闭节能

当然，也可以在操作系统级别进行调整，详细的调整过程请参考（http://www.servernoobs.com/avoiding-cpu-speed-scaling-in-modern-linux-distributions-

running-cpu-at-full-speed-tips/ ），但是，不知道是否因为BIOS调整的缘故，在CentOS 6.6上并没有发现相关的设置。

```
for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do [ -f $CPUFREQ ] || continue; echo -n performance > $CPUFREQ; done
```

（3）NUMA

简单来说，NUMA思路就是将内存和CPU分割为多个区域，每个区域叫作Node，然后将Node高速互联。Node内CPU与内存访问速度快于访问其他Node的内存，NUMA可能会在某些情况下影响ceph-osd。解决的方案，一种是通过BIOS关闭NUMA，另一种就是通过cgroup将ceph-osd进程与某一个CPU Core以及同一Node下的内存进行绑定。第二种方案看起来更麻烦，所以一般部署的时候可以在系统层面关闭NUMA。在

CentOS系统下，通过修改/etc/grub.conf文件，添加numa=off来关闭NUMA，见图9-4。

```
kernel /vmlinuz-2.6.32-504.12.2.el6.x86_64 ro
root=UUID=870d47f8-0357-4a32-909f-74173a9f0633 rd_NO_LUKS
rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-
sun16 crashkernel=auto  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM
biosdevname=0 numa=off
```

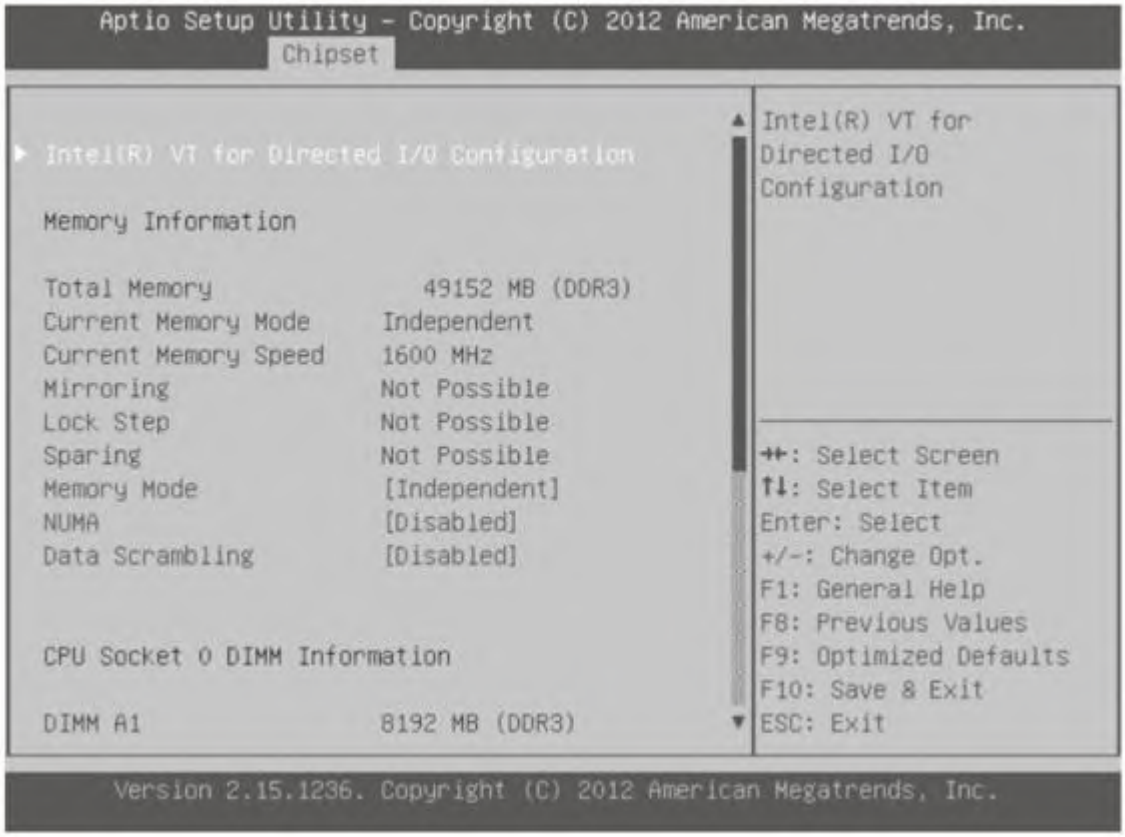

图9-4　关闭NUMA

9.3.2 操作系统优化

根据前面说到的对分布式存储系统优化的两点，在9.3.1节中讲述了从硬件的规划、SSD的选择、机器BIOS设置等几个方面，本节将从Linux系统内核级别优化、文件预读、程序进程等几个方面来讲述软件级别的优化。

1.Linux Kernel级优化

Linux操作系统在服务器市场中占据了相当大的份额，各种发行版及其衍生版在不同领域中承载着各式各样的服务。针对不同的应用环境，可以对Linux的性能进行相应的调整，性能调优涵盖了虚拟内存管理、IO子系统、进程调度系统、文件系统等众多内容，这里仅讨论对Ceph性能影响显著的部分内容。

在Linux的各种发行版中，为了保证对硬件的兼容和可靠性，很多内核参数都采用了较为保守的设置，然而这无法满足我们对于高性能计算的需求，为了Ceph能更好地利用系统资源，我们需要对部分参数进行调整。

（1）调度

Linux默认的IO调度一般针对磁盘寻址慢的特性做了专门优化，但对于SSD而言，由于访问磁盘不同逻辑扇区的时间几乎是一样的，这个优化就没有什么作用了，反而耗费了CPU时间。所以，使用Noop调度器替代内核默认的CFQ。操作如下：

```
echo noop > /sys/block/sdX/queue/scheduler
```

而机械硬盘设置为：

```
echo deadline > /sys/block/sdX/queue/scheduler
```

（2）预读

Linux默认的预读read_ahead_kb并不适合RADOS对象存储读，建议设置更大值：

```
echo "8192" > /sys/block/sdX/queue/read_ahead_kb
```

（3）进程数量

OSD进程需要消耗大量进程。关于内核PID上限，如果单服务器OSD数量多的情况下，建议设置更大值：

```
echo 4194303 > /proc/sys/kernel/pid_max
```

调整CPU频率，使其运行在最高性能下：

```
echo performance | tee
/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
>/dev/null
```

2.内存

（1）SMP和NUMA

SMP（Symmetric Multi Processor）架构中，所有的CPU共享全部资源，如总线、内存和I/O等。多个CPU之间对称工作，无主从或从属关系。每个CPU都需要通过总线访问内存，整个系统中的内存能被每个CPU平等（消耗时间相同）地访问。然而，在CPU数量不断增加后，总线的压力不断增加，最终导致CPU的处理能力大大降低。

NUMA架构体系由多个节点组成，每个节点有若干CPU和它们独立的本地内存组成，各个节点通过互联模块（CrossbarSwitch）进行访问，所以每个CPU可以访问整个系统的内存。但是，访问本地内存的速度远比访问远程内存要快，导致在进程发生调度后可能需要访问远端内存，这种情况下，程序的效率会大大降低。

Ceph目前并未对NUMA架构的内存做过多优化，在日常使用过程中，我们通常使用2~4颗CPU，这种情况下，选择SMP架构的内存在效率上还是要高一些。如果条件允许，可以通过进程绑定的方法，在保证CPU能尽可能访问自身内存的前提下，使用NUMA架构。

（2）SWAP

当系统中的物理内存不足时，就需要将一部分内存中的非活跃（inactive）内存页置换到交换分区（SWAP）中。有时候我们会发现，在物理内存还有剩余的情况下，交换分区就已经开始被使用了，这就涉及kernel中关于交换分区的使用策略，由vm.swappiness这个内核参数控制，该参数代表使用交换分区的程度：当值为0时，表示尽可能地避免换页操作；当值为100时，表示kernel会积极地换页，这会产生大量磁盘IO。因此，在内存充足的情况下，我们一

般会将该参数设置为0以保证系统性能。设定Linux操作系统参数：vm.swappiness=0，配置到/etc/sysctl.conf。

（3）内存管理

Ceph默认使用TCmalloc管理内存，在非全闪存环境下，TCmalloc的性能已经足够。在全闪存的环境中，建议增加TCmalloc的Cache大小或者使用jemalloc替换TCmalloc。

增加TCmalloc的Cache大小需要设置环境变量，建议设置为256MB大小：

```
TCMALLOC_MAX_TOTAL_THREAD_CACHE_BYTES=268435456
```

使用jemalloc需要重新编译打包Ceph，修改编译选项：

```
--with-jemalloc
--without-tcmalloc
```

3.Cgroups

Cgroups是Control groups的缩写，是Linux内核提供的一种可以限制、记录、隔离进程组（Process Groups）所使用的物理资源（如CPU、Memory、IO等）的机制。最初由Google的工程师提出，后来被整合进Linux内核。Cgroups也是LXC为实现虚拟化所使用的资源管理手段，可以说没有Cgroups就没有LXC。Cgroups内容非常丰富，展开讨论完全可以单独写一章，这里我们简单谈一下Cgroups在Ceph中的应用。

说到Cgroups，就不得不说CPU的架构，以E5-2620 v3为例（如图9-5所示）：整个CPU共享L3缓存，每个物理核心有独立的L1和L2缓存，如果开启超线程，两个逻辑CPU会共享同一块L1和L2，所以，在使用Cgroups的时候，需要考虑CPU的缓存命中的问题。

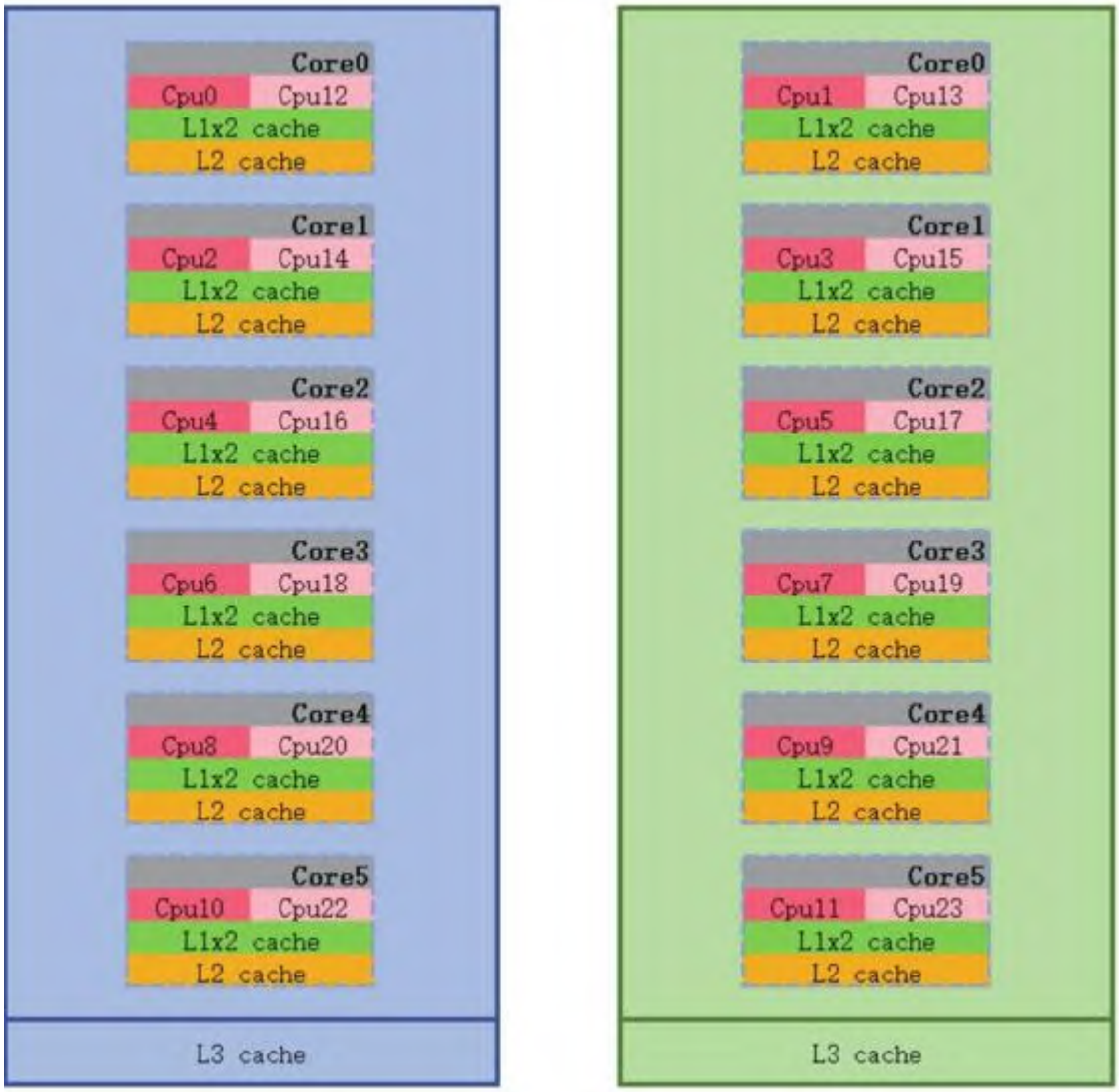

图9-5　E5 2620 v3 CPU拓扑图

可以通过hwloc工具（http://www.open-mpi.org/projects/hwloc/ ）来辨别CPU号码与真实物理核心的对应关系。例如，在配置两个Intel（R）Xeon（R）CPU E5-2680 v2的服务器，CPU拓扑如下：

```
Machine (64GB)
  NUMANode L#0 (P#0 32GB)
    Socket L#0 + L3 L#0 (25MB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) +
Core L#0
        PU L#0 (P#0)
        PU L#1 (P#20)
      L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) +
Core L#1
        PU L#2 (P#1)
        PU L#3 (P#21)
      L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) +
Core L#2
        PU L#4 (P#2)
        PU L#5 (P#22)
      L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) +
Core L#3
        PU L#6 (P#3)
        PU L#7 (P#23)
      L2 L#4 (256KB) + L1d L#4 (32KB) + L1i L#4 (32KB) +
Core L#4
        PU L#8 (P#4)
        PU L#9 (P#24)
      L2 L#5 (256KB) + L1d L#5 (32KB) + L1i L#5 (32KB) +
Core L#5
        PU L#10 (P#5)
        PU L#11 (P#25)
      L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) +
Core L#6
        PU L#12 (P#6)
        PU L#13 (P#26)
      L2 L#7 (256KB) + L1d L#7 (32KB) + L1i L#7 (32KB) +
Core L#7
        PU L#14 (P#7)
        PU L#15 (P#27)
      L2 L#8 (256KB) + L1d L#8 (32KB) + L1i L#8 (32KB) +
Core L#8
        PU L#16 (P#8)
        PU L#17 (P#28)
      L2 L#9 (256KB) + L1d L#9 (32KB) + L1i L#9 (32KB) +
Core L#9
        PU L#18 (P#9)
        PU L#19 (P#29)
    HostBridge L#0
```

```
      PCIBridge
        PCIBridge
          PCIBridge
            PCI 8086:1d6a
      PCIBridge
        PCI 8086:1521
          Net L#0 "eno1"
        PCI 8086:1521
          Net L#1 "eno2"
        PCI 8086:1521
          Net L#2 "eno3"
        PCI 8086:1521
          Net L#3 "eno4"
      PCIBridge
        PCI 1000:0079
          Block L#4 "sda"
          Block L#5 "sdb"
          Block L#6 "sdc"
          Block L#7 "sdd"
          Block L#8 "sde"
      PCIBridge
        PCI 102b:0522
          GPU L#9 "card0"
          GPU L#10 "controlD64"
      PCI 8086:1d02
        Block L#11 "sr0"
  NUMANode L#1 (P#1 32GB)
    Socket L#1 + L3 L#1 (25MB)
      L2 L#10 (256KB) + L1d L#10 (32KB) + L1i L#10 (32KB) +
Core L#10
        PU L#20 (P#10)
        PU L#21 (P#30)
      L2 L#11 (256KB) + L1d L#11 (32KB) + L1i L#11 (32KB) +
Core L#11
        PU L#22 (P#11)
        PU L#23 (P#31)
      L2 L#12 (256KB) + L1d L#12 (32KB) + L1i L#12 (32KB) +
Core L#12
        PU L#24 (P#12)
        PU L#25 (P#32)
      L2 L#13 (256KB) + L1d L#13 (32KB) + L1i L#13 (32KB) +
Core L#13
        PU L#26 (P#13)
        PU L#27 (P#33)
      L2 L#14 (256KB) + L1d L#14 (32KB) + L1i L#14 (32KB) +
```

```
Core L#14
        PU L#28 (P#14)
        PU L#29 (P#34)
      L2 L#15 (256KB) + L1d L#15 (32KB) + L1i L#15 (32KB) +
Core L#15
        PU L#30 (P#15)
        PU L#31 (P#35)
      L2 L#16 (256KB) + L1d L#16 (32KB) + L1i L#16 (32KB) +
Core L#16
        PU L#32 (P#16)
        PU L#33 (P#36)
      L2 L#17 (256KB) + L1d L#17 (32KB) + L1i L#17 (32KB) +
Core L#17
        PU L#34 (P#17)
        PU L#35 (P#37)
      L2 L#18 (256KB) + L1d L#18 (32KB) + L1i L#18 (32KB) +
Core L#18
        PU L#36 (P#18)
        PU L#37 (P#38)
      L2 L#19 (256KB) + L1d L#19 (32KB) + L1i L#19 (32KB) +
Core L#19
        PU L#38 (P#19)
        PU L#39 (P#39)
    HostBridge L#7
      PCIBridge
        PCI 8086:10fb
          Net L#12 "enp129s0f0"
        PCI 8086:10fb
          Net L#13 "enp129s0f1"
```

从上面的代码可以看出，操作系统识别的CPU号和物理核心是一一对应的关系，在对程序做CPU绑定或者使用Cgroups进行隔离时，注意不要跨CPU，以便更好地命中内存和缓存。另外，不同HostBridge对应的PCI

插槽位置不一样，管理着不同的资源，CPU中断设置时也需要考虑这个因素。

下面是CPU的Cgroup绑定脚本。

```
mkdir -p /sys/fs/cgroup/cpuset/ceph
#CPU Number: 0,1,2,3 = 0-3
echo 0,4 > /sys/fs/cgroup/cpuset/ceph/cpuset.cpus
#NUMA Node
echo 0 > /sys/fs/cgroup/cpuset/ceph/cpuset.mems
osd_pid_list=$(ps aux | grep osd | grep -v grep |awk '{print
$2}')
for osd_pid in ${osd_pid_list}
do
     echo ${osd_pid} >
/sys/fs/cgroup/cpuset/ceph/cgroup.procs
done
```

可以根据这个配置思路，将其添加到OSD或者MON的启动脚本，从pid中获取进程号。

总结一下，在使用Cgroups的过程中，我们应当注意以下问题。

- 防止进程跨CPU核心迁移，以更好利用缓存。

• 防止进程跨物理CPU迁移，以更好利用内存和缓存。

• Ceph进程和其他进程会互相抢占资源，使用Cgroups做好隔离措施。

• 为Ceph进程预留足够多的CPU和内存资源，防止影响性能或产生OOM。

 注意

在生产环境中，尤其是高性能环境中，官方推荐配置并不能完全满足Ceph进程的开销，在高性能场景（全SSD）下，每个OSD进程可能需要高达6GHz的CPU和4GB以上的内存。

9.3.3 网络层面优化

这里把网络部分的优化独立出一节来写，主要原因是网络通信在Ceph的工作流程中大量使用到。Ceph距今已经有10余年的历史，时至今日，Ceph各个组件间的通信依然使用10年前的设计：基于多线程模型的网络通信，每个终端包含读和写两个线程，负责消息的接收和发送。在默认三副本的情况下，客户端的每次写请求至少需要6次网络通信。作为Ceph的基石，在接下来的一节中，我们将讨论网络优化在Ceph中的应用。

任何时候通过一个套接字（socket）来读写数据时，都会使用一个系统调用（system call），这个调用将触发内核上下文切换（Context Switch），下面描述了一个典型的系统调用流程：

1）Ceph进程调用send()函数发送消息。

2）触发0x80中断，由用户态切换至内核态。

3）内核调用system_call()函数，进行参数检查，根据系统调用以获得对应的内核函数。

4）执行内核函数，发送数据报文。

5）内核函数执行完毕，切换回内核态。

6）Socket()调用完成。

整个数据发送/接收需要触发两次上下文切换，以及若干次内存拷贝，这些操作会消耗大量的时间，我们优化的思路就是减少这些时间损耗。在处理网络IO时需要CPU消耗大量的计算能力，因此我们希望CPU尽可能少地处理这些琐碎的IO任务，有足够的处理能力

运行Ceph集群，这一节我们主要讨论使用巨型帧、中断亲和等技术加速网络IO。

1.巨型帧

以太网诞生以来，其帧结构一直就没有发生过大的改变。默认情况下，以太网的MTU（Maximum Transmission Unit）是1500字节。默认情况下以太网帧是1522字节，包含1500字节的负载、14字节的以太网头部、4字节的CRC、4字节的VLAN Tag，超过该大小的数据包会被拆封成更多数据包。巨型帧是长度大于1522字节的以太网帧，通过调整MTU（通常我们会调整到9000）来减少网络中数据包的个数，减轻网络设备处理包头的额外开销。设置MTU需要本地设备和对端设备同时开启，开启巨型帧后，可以极大地提高性能。

2.中断亲和

前面提到了当我们要进行网络IO时，会触发系统中断。默认情况下，所有的网卡中断都交由CPU0处理，当大量网络IO出现时，处理大量网络IO中断会导致CPU0长时间处于满负载状态，以致无法处理更多IO导致网络丢包等并发问题，产生系统瓶颈。Linux 2.4内核之后引入了将特定中断绑定到指定的CPU的技术，称为中断亲和（SMP IRQ affinity）。

Linux中所有的中断情况在文件/proc/interrupt中记录，如图9-6所示。

Linux在用户空间提供了proc虚拟文件系统，让我们可以与内核内部数据结构进行交互。我们可以通过该文件系统实现中断绑定。

图9-6　中断情况记录

我们可以通过echo “$bitmask” >/proc/irq/$num/smp_affinity来改变它的值。bitmask代表CPU的掩码，以十六进制表示，每一位代表一个CPU核。

#表示将

112号中断绑定到

#10（二进制为

010000000000）号

CPU
echo 400 > /proc/irq/112/smp_affinity
#表示将

113号中断绑定到

#11（二进制为

100000000000）号

CPU
echo 800 > /proc/irq/113/smp_affinity

但是像这样计算CPU，设置中断未免太过麻烦，还好我们有irqbalance服务的帮助。该服务会定期（10秒）统计CPU的负载和系统的中断量，自动迁移中断保持负载平衡。关于irqbalance服务会影响网卡CPU效能，可通过设置/etc/sysconfig/irqbalance的IRQBALANCE_BANNED_CPUS值，指定irq不在某几个CPU上运行。例如，Linux服务器上有8个逻辑CPU，设定“1111 1100”，代表irq不运行在CPU2~CPU7上，只运行在CPU0~CPU1，其十六进制表示为“FC”。设定后重启服务：service irqbalance restart。

Irqbalane看起来很美好，在部分情况下确实能极大减少我们的工作量，但由于它的检测无法保证实时性，部分情况下甚至会加剧系统负载。建议还是根据系统规划，通过手动设置中断亲和，隔离部分CPU处理网卡中断。

3.硬件加速

在大多数情况下，CPU需要负责服务器中几乎所有的数据处理任务，事实上CPU并不如我们想象中的那样强大，在大量的数据处理中往往显得力不从心，于是便有了硬件加速技术。硬件加速能够将计算量比较大的工作分配给专门的硬件设备处理，比如常见的使用视频硬件解码加速等，在Ceph中，我们主要使用网卡完成对于网络数据处理的加速。

TCP协议处理网络流量时，需要占用大量CPU和内存资源，为了节省服务器资源的消耗，众多厂商开始在网卡中内置协处理器，将计算任务移交给协处理器完成，即TCP卸载引擎（TCP offload Engine，TOE）。TOE目前主要能协助完成以下工作。

（1）协议处理

普通网卡处理网络IO的很大一部分压力都来自于对TCP/IP协议的处理，例如对IP数据包的校验处理、对TCP数据流的可靠性和一致性处理。TOE网卡可以将这些计算工作交给网卡上的协处理器完成。

（2）中断处理

上面讲到，在通用网络IO的处理方式上，普通网卡每个数据包都要触发一次中断，TOE网卡则让每个应用程序完成一次完整的数据处理进程后才出发一次中断，显著减轻服务对中断的响应负担。

（3）减少内存拷贝

普通网卡先将接收到的数据在服务器的缓冲区中复制一份，经系统处理后分配给其中一个TCP连接，然后，系统再将这些数据与使用它的应用程序相关联，并将这些数据由系统缓冲区复制到应用程序的缓冲

区。TOE网卡在接收数据时，在网卡内进行协议处理，因此，它不必将数据复制到服务器缓冲区，而是直接复制到应用程序的缓冲区，这种数据传输方式减少了部分内存拷贝的消耗。

在Linux中，可以使用ethtool查看网卡状态或设置网卡参数，见图9-7。

```
root@compute05:~# ethtool -k em1
Features for em1:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: on
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: on
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
        tx-tcp-segmentation: off
        tx-tcp-ecn-segmentation: off
        tx-tcp6-segmentation: off
udp-fragmentation-offload: off [fixed]
```

图9-7　使用ethtool查看网卡状态或设置网卡参数

使用命令ethtool-K em1 tso on打开tcp-segmentation-offload，查看tcp-segmentation-offload已经打开（on），见图9-8。

```
root@compute05:~# ethtool -K em1 tso on
root@compute05:~# ethtool -k em1
Features for em1:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: on
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: on
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: on
        tx-tcp6-segmentation: on
udp-fragmentation-offload: off [fixed]
```

图9-8　使用ethtool打开tcp-segmentation-offload

4. RDMA

RDMA（Remote Direct Memory Access）可以在不需要操作系统干预的情况下，完成两个主机之间内存数据的传输，见图9-9。传统的套接字接口调用在每次IO过程中需要经历若干次内存拷贝和上下文切换，RDMA技术可以让应用程序在用户态直接将buffer（缓冲区）中的数据写入网卡（NIC）的内存中，以网络为载体，发送到远程网卡，直接写入应用缓存中。在RDMA的工作过程中，使用零拷贝网络技术使得应用程序可以直接与网卡互传数据，避免了用户态到内核态之间的切换和内存拷贝。RDMA的缺点是传统硬件无法直接使用RDMA，需要特殊硬件设备（网卡和交换机）支持，所以使用RDMA前需要考虑成本问题。

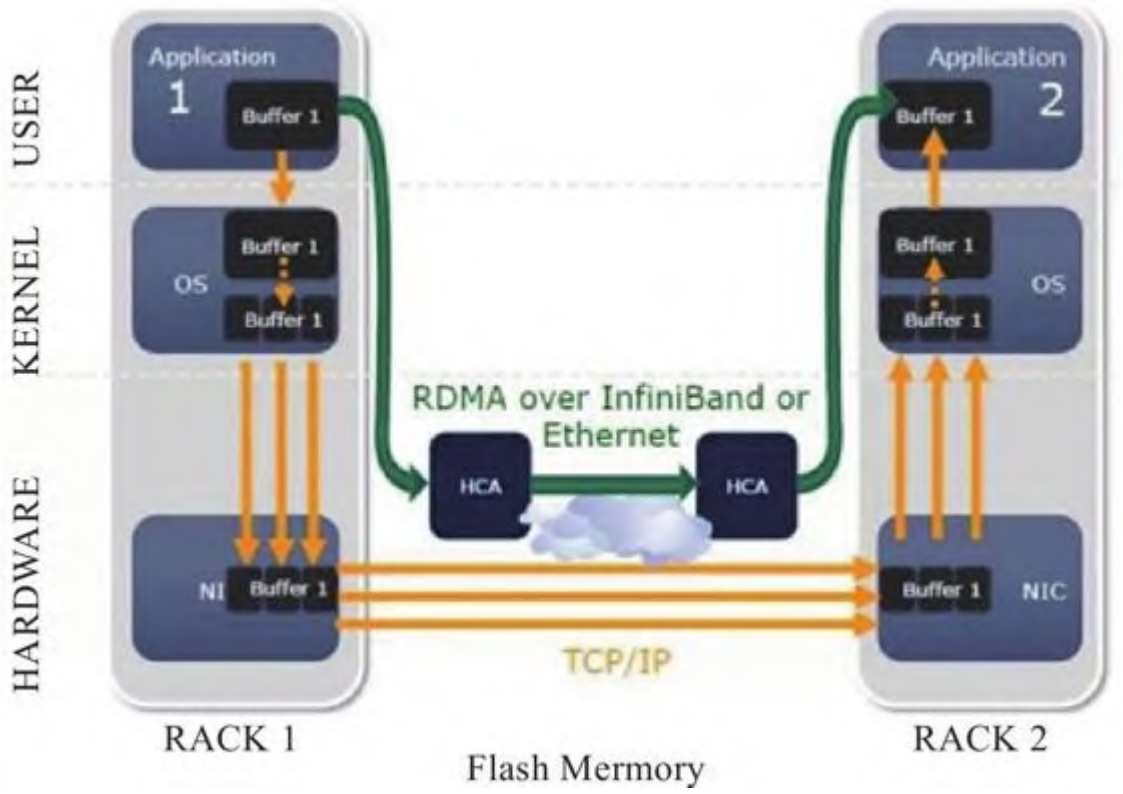

图9-9　RDMA的工作机制

目前RDMA在Ceph中的使用主要由Mellanox维护，使用accelio实现了类似SampleMessager的xio消息处理机制。Accelio是一个高性能的可靠异步消息和RPC库，accelio目前运行在RDMA上（InfiniBand或Ethernet）。下面介绍下在Ceph中如何通过accelio使用RDMA。在使用accelio之前，确定RDMA已经能在各个节点上正常运行。

1）使用xio需要accelio支持，下载Accelio源代码，构建并安装最新版本的主分支（在每台服务器上）：

```
# mkdir /tmp/xio
# cd /tmp/xio
# git clone git://github.com/accelio/accelio.git accelio.git
# cd accelio.git
# git checkout -b rec_commit
9cea8291787b72a746e42964d5de42d6d48f0e0d
# ./autogen.sh
configure.ac:13: installing './compile'
configure.ac:13: installing './config.guess'
configure.ac:13: installing './config.sub'
configure.ac:7: installing './install-sh'
configure.ac:7: installing './missing'
benchmarks/usr/xio_perftest/Makefile.am: installing
'./depcomp'
# ./configure --prefix=/opt/accelio
# make && make install
```

2）下载Ceph最新版本的主分支源代码，构建并安装（在每台服务器上）：

```
# mkdir /tmp/ceph ; cd /tmp/ceph
$ git clone --recursive https://github.com/ceph/ceph
```

```
 ceph.git
$ cd ceph.git
$ git checkout -b rec_commit
f74c60f429f26c1cd55e219642fa19cb5a803468
```

3）使用automake编译ceph：

```
$ ./autogen.sh
$ CXXFLAGS="-I/opt/accelio/include"  CFLAGS="-
I/opt/accelio/include" LDFLAGS="-L/opt/accelio/lib"
./configure --prefix=/opt/ceph --enable-xio
$ make -j32 && make install
```

4）将下面的配置复制到各个节点的/etc/ceph/ceph.conf中：

```
[global]
admin_socket = /ceph-test/var/run/ceph/ceph-$name.$id.asok
max_open_files = 131072
log_file = /ceph-test/var/log/ceph/$name.log
pid_file = /ceph-test/var/run/ceph/$name.pid
filestore_xattr_use_omap = 1
osd_pool_default_size = 1
osd_pool_default_min_size = 1
```

```
debug_auth = 0/0
debug_asok = 0/0
debug_buffer = 0/0
debug_client = 0/0
debug_context = 0/0
debug_crush = 0/0
debug_crypto = 0/0
debug_filer = 0/0
debug_filestore = 0/0
debug_finisher = 0/0
debug_heartbeatmap = 0/0
debug_journal = 0/0
debug_journaler = 0/0
debug_lockdep = 0/0
debug_monclient = 0/0
debug_mon = 0/0
debug_monc = 0/0
debug_ms = 0/0
debug_objclass = 0/0
debug_objecter = 0/0
debug_objectcacher = 0/0
debug_optracker = 0/0
debug_osd = 0/0
debug_paxos = 0/0
debug_perfcounter = 0/0
debug_rados = 0/0
debug_rbd = 0/0
debug_rgw = 0/0
debug_timer = 0/0
debug_tp = 0/0
debug_throttle = 0/0
debug_xio = 0
ms_crc_header = false
ms_crc_data = false
auth_supported = none
auth_service_required = none
auth_client_required = none
auth_cluster_required = none
rdma_local = 11.11.11.1
enable experimental unrecoverable data corrupting features =
ms-type-xio
ms_type = xio
xio_mp_max_64 = 262144
xio_mp_max_256 = 262144
xio_mp_max_1k = 262144
```

```
xio_mp_max_page = 131072
xio_portal_threads = 4
xio_max_send_inline = 8192
osd_op_threads = 2
filesore_op_threads = 4
filestore_fd_cache_size = 64
filestore_fd_cache_shards = 32
osd_op_num_threads_per_shard = 1
osd_op_num_shards = 10
throttle_perf_counter = false
ms_dispatch_throttle_bytes = 0
rbd_cache = false
[osd]
osd_client_message_size_cap = 0
osd_client_message_cap = 0
osd_enable_op_tracker = false
osd_data = /ceph-test/ceph-data/osd.$id
osd_journal = /ceph-test/ceph-data/osd.$id/journal
osd_journal_size = 256
osd_scrub_load_threshold = 2.5
osd_mkfs_type = xfs
osd_mount_options_xfs = rw,noatime
osd_mkfs_options_xfs = -Kf
osd_class_dir = /opt/ceph/lib/rados-classes
[osd.0]
host = ceph-server
devs = /dev/sdb
ms_bind_port_min = 7100
ms_bind_port_max = 7200
[mon]
mon_data = /ceph-test/ceph-data/mon.$id
[mon.0]
host = ceph-server
mon_addr = 11.11.11.1:16789
[mds]
keyring = /ceph-test/ceph-data/keyring.$name
cluster_addr = 11.11.11.1:26789
public_addr = 11.11.11.1:36789
objecter_timeout = 10
mds_reconnect_timeout = 5
mds_beacon_interval = 2
[mds.0]
 host = ceph-server
```

5）创建Mon和OSD的数据目录：

```
# LD_LIBRARY_PATH="/opt/ceph/lib;/opt/accelio/lib"
/opt/ceph/bin/mkcephfs -a -c /etc/ceph/ceph.conf -k
/etc/ceph/keyring -d /ceph-test/ceph-data  --mkfs
```

6）手动启动ceph-mon进程：

```
ceph-server #
LD_LIBRARY_PATH="/opt/ceph/lib;/opt/accelio/lib"
/opt/ceph/bin/ceph-mon -i 0 --pid-file /ceph-
test/var/run/ceph/mon.0.pid
```

7）运行ceph-client，验证ceph-mon的状态是否为up：

```
ceph-client #
LD_LIBRARY_PATH="/opt/ceph/lib;/opt/accelio/lib"
PYTHONPATH="/opt/ceph/lib/python2.7/site-packages/"
/opt/ceph/bin/ceph -s
2015-04-30 02:18:18.249382 7fb29a8e3700 -1 WARNING: the
following dangerous and experimental features are enabled:
ms-type-xio
2015-04-30 02:18:18.250454 7fb29a8e3700 -1 WARNING: the
following dangerous and experimental features are enabled:
ms-type-xio
2015-04-30 02:18:18.250672 7fb29a8e3700 -1 WARNING:
experimental feature 'ms-type-xio' is enabled
Please be aware that this feature is experimental, untested,
unsupported, and may result in data corruption, data loss,
and/or irreparable damage to your cluster.  Do not use
```

```
feature with important data.
2015-04-30 02:18:18.252814 7fb29a8e3700  0 Peer type: mon
throttle_msgs: 512 throttle_bytes: 536870912
    cluster 8eb9f961-0c37-488b-b807-61c8893ed2b1
     health HEALTH_ERR
            64 pgs stuck inactive
            64 pgs stuck unclean
            no osds
     monmap e1: 1 mons at {0=11.11.11.4:16789/0}
            election epoch 2, quorum 0 0
     osdmap e1: 0 osds: 0 up, 0 in
      pgmap v2: 64 pgs, 1 pools, 0 bytes data, 0 objects
            0 kB used, 0 kB / 0 kB avail
                  64 creating
```

8）手动启动ceph-osd进程：

```
ceph-server#
LD_LIBRARY_PATH="/opt/accelio/lib;/opt/ceph/lib"
/opt/ceph/bin/ceph-osd -i $i --pid-file /ceph-
test/var/run/ceph/osd.$i.pid
```

至此，运行在RDMA上的Ceph环境搭建完成，然后可以和通常一样使用、管理Ceph集群。

5.DPDK

DPDK（Data Plane Deveploment Kit）抛弃了传统使用CPU中断处理数据包的方式，采用了轮询方式实

现数据包处理过程：DPDK重载了网卡驱动，驱动在收到数据包后不使用中断通知CPU，而是直接使用零拷贝存入用户态内存中，使得应用程序可以通过DPDK提供的接口从内存中直接读取数据包。使用DPDK类似RDMA，避免了内存拷贝、上下文切换的时间。图9-10为DPDK层次结构图。

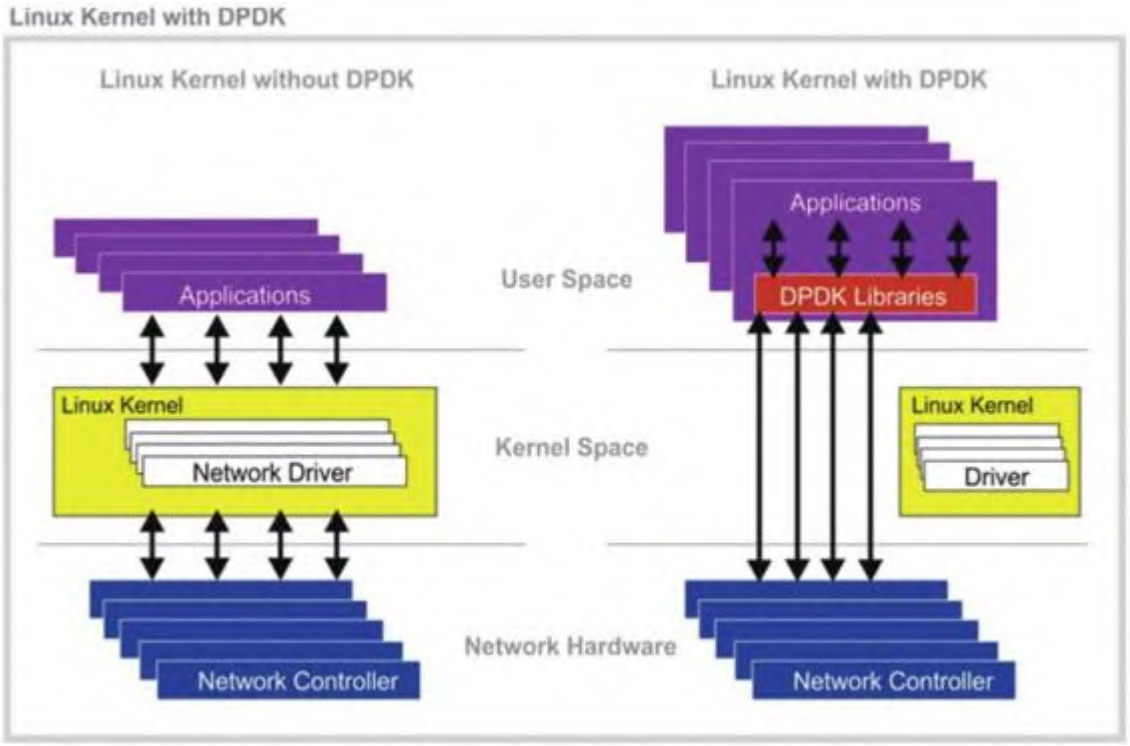

图9-10　DPDK层次结构图

由于Intel极力推广DPDK，使得Intel的大部分网卡驱动都支持DPDK，这让部分设备利旧也成为一种可能。有了硬件和系统的性能支持，开发者们就可以利用DPDK为Ceph开发更高性能的消息机制，目前社区关于DPDK的应用还在实现中，部分企业已经先社区一步应用了DPDK，我们可以根据需求决定是否投入研发DPDK的实现。

9.3.4　Ceph层面优化

1.Ceph参数

以上部分主要围绕着硬件、操作系统和网络进行优化，下面我们围绕Ceph本身的参数进行调优，Ceph将很多运行参数作为配置项保存在配置文件中，Ceph为我们提供了相当详细的配置参数供用户在不同场景下进行调整和优化。

表9-1为global全局参数以及参数描述，可以通过Linux命令sysctl来设定max open files的值fs.file-max。

表9-1　global全局参数

参数名	描述	默认值	建议值
public network	客户端访问网络		192.168.100.0/24
cluster network	集群网络		192.168.1.0/24
max open files	如果设置了该选项，Ceph 会设置系统的 max open fds	0	131 072

查看系统最大文件打开数可以使用如下命令。

```
cat /proc/sys/fs/file-max
```

表9-2为[osd]-filestore参数。

表9-2　filestore参数

参数名	描述	默认值	建议值
filestore xattr use omap	为XATTRS使用object map，EXT4文件系统时使用，XFS或者btrfs也可以使用	false	true
filestore max sync interval	从日志到数据盘最大同步间隔（seconds）	5	15
filestore min sync interval	从日志到数据盘最小同步间隔（seconds）	0.1	10
filestore queue max ops	数据盘最大接受的操作数	500	25 000
filestore queue max bytes	数据盘一次操作最大字节数（bytes）	100 << 20	10 485 760
filestore queue committing max ops	数据盘能够commit的操作数	500	5000
filestore queue committing max bytes	数据盘能够commit的最大字节数（bytes）	100 << 20	10 485 760 000
filestore op threads	并发文件系统操作数	2	32

说明：

• 调整omap的原因主要是EXT4文件系统默认仅有4K。

• filestore queue相关的参数对于性能影响很小，参数调整不会对性能优化有本质上提升。

表9-3为[osd]-journal相关参数。

表9-3　journal相关参数

参数名	描述	默认值	建议值
osd journal size	OSD 日志大小（MB）	5120	20 000
journal max write bytes	journal 一次性写入的最大字节数（bytes）	10 << 20	1 073 714 824
journal max write entries	journal 一次性写入的最大记录数	100	10 000
journal queue max ops	journal 一次性最大在队列中的操作数	500	50 000
journal queue max bytes	journal 一次性最大在队列中的字节数（bytes）	10 << 20	10 485 760 000

注意：Ceph OSD进程在往数据盘上刷数据的过程中，是停止写操作的。

表9-4为[osd]-osd config tuning参数。

表9-4　osd config tuning相关参数

参数名	描述	默认值	建议值
osd max write size	OSD一次可写入的最大值（MB）	90	512
osd client message size cap	客户端允许在内存中的最大数据（bytes）	524 288 000	2 147 483 648
osd deep scrub stride	在Deep Scrub时候允许读取的字节数（bytes）	524 288	131 072
osd op threads	OSD进程操作的线程数	2	8
osd disk threads	OSD密集型操作例如恢复和Scrubbing时的线程	1	4
osd map cache size	保留OSD Map的缓存（MB）	500	1 024
osd map cache bl size	OSD进程在内存中的OSD Map缓存（MB）	50	128
osd mount options xfs	Ceph OSD xfs Mount选项	rw，noatime，inode64	rw，noexec，nodev，noatime，nodiratime，nobarrier

表9-5为[osd]-recovery tuning参数。

表9-5　recovery tuning相关参数

参数名	描述	默认值	建议值
osd recovery op priority	恢复操作优先级，取值1-63，值越高占用资源越高	10	4
osd recovery max active	同一时间内活跃的恢复请求数	15	10
osd max backfills	一个OSD允许的最大backfills数	10	4

表9-6为[osd]-client tuning参数。

表9-6　client tuning相关参数

参数名	描述	默认值	建议值
rbd cache	RBD 缓存	true	true
rbd cache size	RBD 缓存大小（bytes）	33 554 432	268 435 456
rbd cache max dirty	缓存为 write-back 时允许的最大 dirty 字节数（bytes），如果为 0，使用 write-through	25 165 824	134 217 728
rbd cache max dirty age	在被刷新到存储盘前 dirty 数据存在缓存的时间（seconds）	1	5

2.PG数量优化

PG和PGP数量一定要根据OSD的数量进行调整，计算公式如下，但是最后算出的结果一定要接近或者等于2的指数。

```
Total PGs = (Total_number_of_OSD * 100) /
max_replication_count
```

例如，15个OSD，副本数为3的情况下，根据公式计算的结果应该为500，最接近512，所以需要设定该pool(volumes)的pg_num和pgp_num都为512。

```
ceph osd pool set volumes pg_num 512
ceph osd pool set volumes pgp_num 512
```

3. Ceph参数配置示例

下面的Ceph参数配置是在使用过程中逐渐积累的一个优化后的配置，可以参考下面的配置对Ceph集群进行参数调优。

```
 [global]
fsid = 059f27e8-a23f-4587-9033-3e3679d03b31
mon_host = 10.10.20.102, 10.10.20.101, 10.10.20.100
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
osd pool default size = 3
osd pool default min size = 1
public network = 10.10.20.0/24
cluster network = 10.10.20.0/24
max open files = 131072
[mon]
mon data = /var/lib/ceph/mon/ceph-$id
[osd]
osd data = /var/lib/ceph/osd/ceph-$id
osd journal size = 20000
osd mkfs type = xfs
osd mkfs options xfs = -f
filestore xattr use omap = true
filestore min sync interval = 10
filestore max sync interval = 15
filestore queue max ops = 25000
filestore queue max bytes = 10485760
filestore queue committing max ops = 5000
filestore queue committing max bytes = 10485760000
journal max write bytes = 1073714824
journal max write entries = 10000
journal queue max ops = 50000
journal queue max bytes = 10485760000
osd max write size = 512
```

```
osd client message size cap = 2147483648
osd deep scrub stride = 131072
osd op threads = 8
osd disk threads = 4
osd map cache size = 1024
osd map cache bl size = 128
osd mount options xfs =
"rw,noexec,nodev,noatime,nodiratime,nobarrier"
osd recovery op priority = 4
osd recovery max active = 10
osd max backfills = 4
[client]
rbd cache = true
rbd cache size = 268435456
rbd cache max dirty = 134217728
rbd cache max dirty age = 5
```

9.4 Ceph测试

在Ceph上线之前要进行测试，看看Ceph是否能够满足自己的场景需求，下面会讲到存储系统模型以及如何测试硬盘和云硬盘。

9.4.1 测试前提

在进行测试时，我们要清楚以下几点。

• 测试对象：要区分硬盘、SSD、RAID、SAN和云硬盘等，因为它们有不同的特点。

• 测试指标：IOPS和MBPS（吞吐率），下面会具体阐述。

• 测试工具：Linux下常用Fio、dd工具，Windows下常用IOMeter。

• 测试参数：IO大小、寻址空间、队列深度、读写模式和随机/顺序模式。

• 测试方法：也就是测试步骤。

测试是为了对比，所以需要定性和定量。在宣布自己的测试结果时，需要说明这次测试的工具、参数和方法，以便于比较。

9.4.2　存储系统模型

为了更好地测试，我们需要先了解存储系统。块存储系统本质是一个排队模型，我们可以拿银行为例。还记得你去银行办事时的流程吗？

1）去前台取单号。

2）等待排在你之前的人办完业务。

3）轮到你去某个柜台。

4）柜台职员帮你办完手续1。

5）柜台职员帮你办完手续2。

6）柜台职员帮你办完手续3。

7）办完业务，从柜台离开。

如何评估银行的效率呢？

- 服务时间=手1+手续2+手续3。
- 响应时间=服务时间+等待时间。
- 性能=单位时间内处理业务数量。

那银行如何提高效率呢？

- 增加柜台数。
- 降低服务时间。

因此，排队系统或存储系统的优化方法如下。

- 增加并行度。
- 降低服务时间。

9.4.3　硬盘测试

1.硬盘原理

我们应该如何测试SATA/SAS硬盘呢？首先需要了解磁盘的构造，并了解磁盘的工作方式，如图9-11所示。

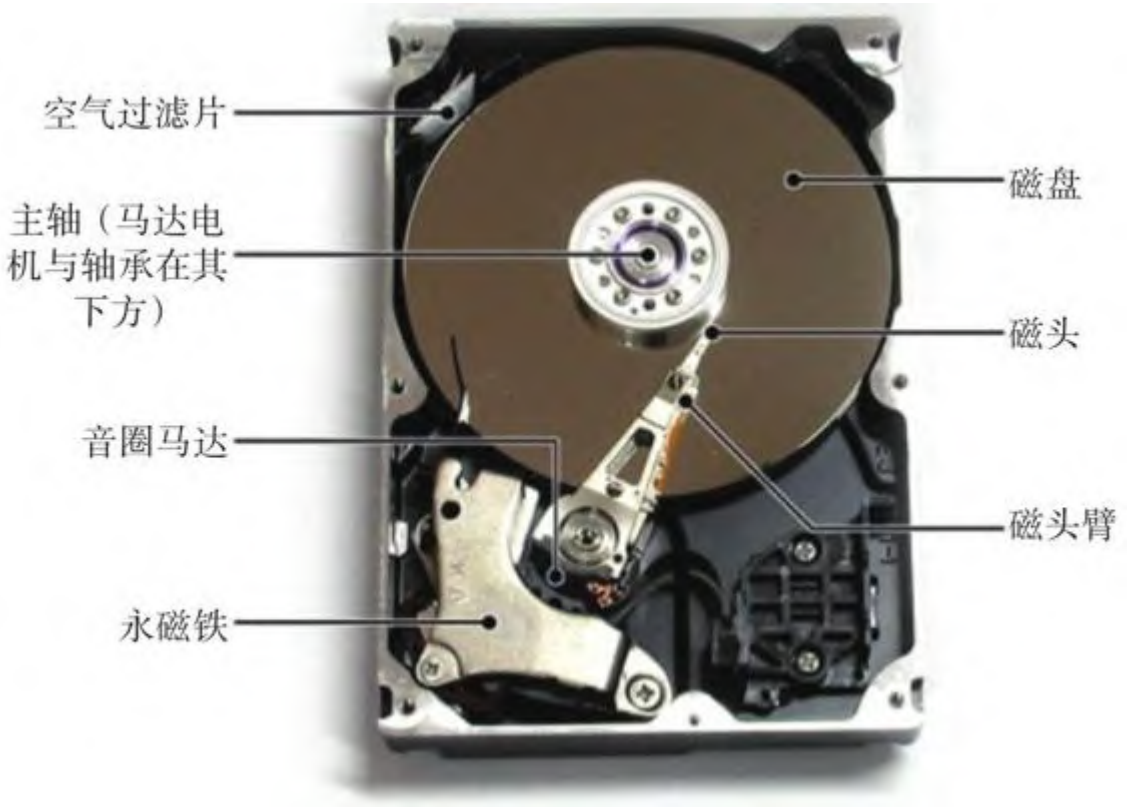

图9-11　硬盘内部构造

每个硬盘都有一个磁头（相当于银行的柜台），硬盘的工作方式如下。

1）收到IO请求，得到地址和数据大小。

2）移动磁头（寻址）。

3）找到相应的磁道（寻址）。

4）读取数据。

5）传输数据。

磁盘的随机IO服务时间：

$$服务时间 = 寻道时间 + 旋转时间 + 传输时间$$

对于10000转速的SATA硬盘来说，一般寻道时间是7 ms，旋转时间是3 ms，64KB的传输时间是0.8 ms，则SATA硬盘每秒可以进行随机IO操作是1000/（7+3+0.8）=93。所以，我们估算SATA硬盘64KB随机

写的IOPS是93。一般的硬盘厂商都会标明顺序读写的MBPS（吞吐量）。

我们在列出IOPS时，需要说明IO大小、寻址空间、读写模式、顺序/随机和队列深度。一般常用的IO大小是4KB，这是因为文件系统常用的块大小是4KB。

2.使用dd测试硬盘

虽然硬盘的性能是可以估算出来的，但是怎么才能让应用获得这些性能呢？对于测试工具来说，就是如何得到IOPS和MBPS峰值。我们先用dd测试一下SATA硬盘的MBPS（吞吐量）。

```
#dd if=/dev/zero of=/dev/sdd bs=4k count=300000 oflag=direct
记录了


300000+0 的读入
```

记录了

300000+0 的写出

1228800000字节

(1.2 GB)已复制，

17.958 秒，

```
68.4 MB/秒
```

在dd测试的同时也要对系统IO进行监控，使用iostat命令对系统IO进行查看。

```
#iostat -x sdd 5 10
...
Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s avgrq-sz avgqu-sz await svctm %util
sdd 0.00 0.00 0.00 16794.80 0.00 134358.40 8.00 0.79 0.05 0.05 78.82
...
```

为什么这块硬盘的MBPS只有68MB/s？这是因为磁盘利用率是78%，没有到达95%以上，还有部分时间是空闲的。当dd在前一个IO响应之后，在准备发起下一

个IO时，SATA硬盘是空闲的。那么，如何才能提高利用率，让磁盘不空闲呢？只有一个办法，那就是增加硬盘的队列深度。相对于CPU来说，硬盘属于慢速设备，所有操作系统会给每个硬盘分配一个专门的队列用于缓冲IO请求。

3.队列深度

什么是磁盘的队列深度？在某个时刻，有N个inflight的IO请求，包括在队列中的IO请求、磁盘正在处理的IO请求。N就是队列深度。加大硬盘队列深度就是让硬盘不断工作，减少硬盘的空闲时间。

加大队列深度→提高利用率→获得IOPS和MBPS峰值→注意响应时间在可接受的范围内。

增加队列深度的办法有很多，如下。

• 使用异步IO，同时发起多个IO请求，相当于队列中有多个IO请求。

• 多线程发起同步IO请求，相当于队列中有多个IO请求。

• 增大应用IO大小，到达底层之后，会变成多个IO请求，相当于队列中有多个IO请求队列深度增加了。

队列深度增加了，IO在队列的等待时间也会增加，导致IO响应时间变大，这需要权衡。让我们通过增加IO大小来增加dd的队列深度，看有没有效果：

```
dd if=/dev/zero of=/dev/sdd bs=2M count=1000 oflag=direct记录了


1000+0 的读入
```

记录了

1000+0 的写出

2097152000字节

(2.1 GB)已复制，

10.6663 秒，

```
197 MB/秒
```

```
Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s avgrq-sz avgqu-
sz await svctm %util
sdd 0.00 0.00 0.00 380.60 0.00 389734.40 1024.00 2.39 6.28
2.56 97.42
```

可以看到2MB的IO到达底层之后，会变成多个512KB [1] 的IO，平均队列长度为2.39，这个硬盘的利用率是97%，MBPS达到了197MB/s。也就是说，增加队列深度，是可以测试出硬盘的峰值的。

4.使用fio测试硬盘

现在，我们来测试SATA硬盘的4KB随机写的IOPS。因为环境是Linux，所以使用fio来测试。

```
$fio -ioengine=libaio -bs=4k -direct=1 -thread -rw=randwrite
-size=1000G -filename=/dev/vdb -name="EBS 4K randwrite test"
-iodepth=64 -runtime=60
```

简单介绍一下fio的参数如下。

• Ioengine：负载引擎，我们一般使用libaio，发起异步IO请求。

• bs：IO大小。

• direct：直写，绕过操作系统Cache。因为我们测试的是硬盘，而不是操作系统的Cache，所以设置为1。

• rw：读写模式，有顺序写write、顺序读read、随机写randwrite和随机读randread等。

• size：寻址空间，IO会落在[0,size）这个区间的硬盘空间上。这是一个可以影响IOPS的参数。一般设置为硬盘的大小。

• filename：测试对象。

• iodepth：队列深度，只有使用libaio时才有意义。这是一个可以影响IOPS的参数。

• runtime：测试时长。

下面我们做两次测试，分别测试iodepth=1和iodepth=4的情况。图9-12是iodepth=1的测试结果。

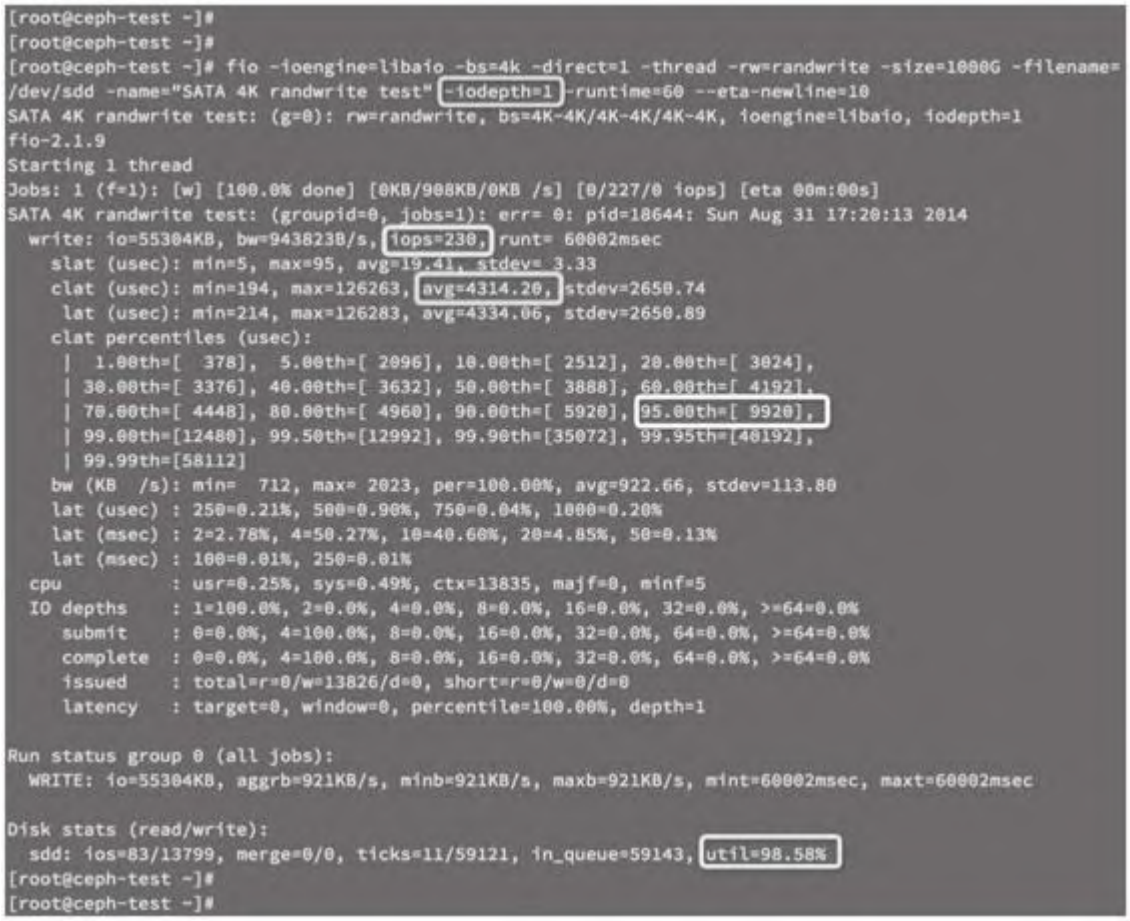

图9-12　iodepth 1测试结果

图9-12中蓝色方框里面的是测出的IOPS 230，绿色方框里面是每个IO请求的平均响应时间，大约是4.3ms。黄色方框表示95%的IO请求的响应时间是小于等于9.920 ms。橙色方框表示该硬盘的利用率已经达到了98.58%。

图9-13是iodepth=4的测试结果。

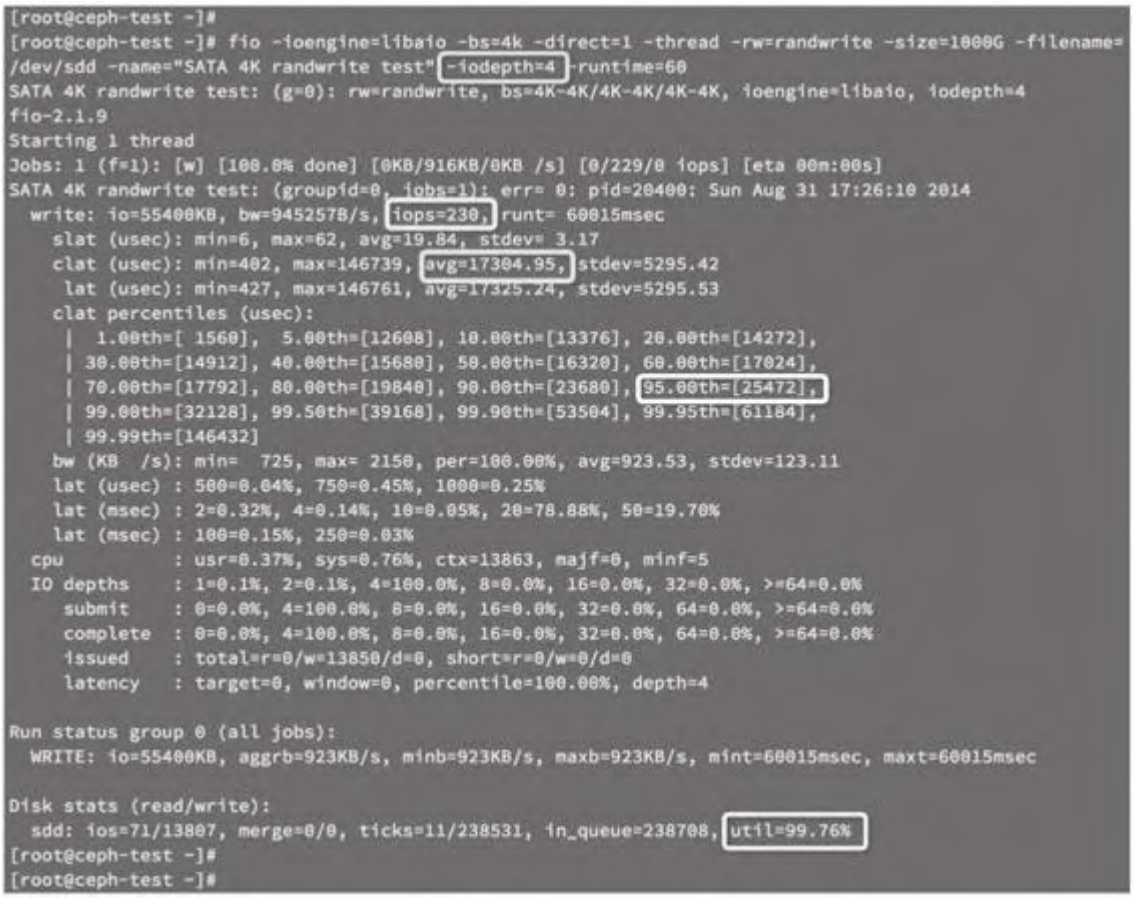

图9-13　iodepth4测试结果

我们发现这次测试的IOPS没有提高，反而IO平均响应时间变大了，是17ms。

为什么这里提高队列深度没有作用呢？原因是当队列深度为1时，硬盘的利用率已经达到了98%，说明

硬盘已经没有多少空闲时间可以压榨了。而且响应时间为4ms。对于SATA硬盘，当增加队列深度时，并不会增加IOPS，只会增加响应时间。这是因为硬盘只有一个磁头，并行度是1，所以当IO请求队列变长时，每个IO请求的等待时间都会变长，导致响应时间也变长。

表9-7是以前用IOMeter测试一块SATA硬盘的4K随机写性能，可以看到IOPS不会随着队列深度的增加而增加，反而是平均响应时间在倍增。

表9-7　用IOMeter测试一块SATA硬盘的4K随机写性能

队列深度	IOPS	平均响应时间
1	332.931 525	3.002 217
2	333.985 074	5.986 528
4	332.594 653	12.025 060
8	336.568 012	23.766 359
16	329.785 606	48.513 477
32	332.054 590	96.353 934
64	331.041 063	193.200 815
128	331.309 109	385.163 111
256	327.442 963	774.401 781

5.寻址空间对IOPS的影响

我们继续测试SATA硬盘，前面我们提到寻址空间参数也会对IOPS产生影响，下面我们测试当size=1GB时的情况，如图9-14所示。

```
[root@ceph-test ~]#
[root@ceph-test ~]# fio -ioengine=libaio -bs=4k -direct=1 -thread -rw=randwrite -size=1G -filename=/dev/sdd
 -name="SATA 4K randwrite test" -iodepth=4 -runtime=60
SATA 4K randwrite test: (g=0): rw=randwrite, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=4
fio-2.1.9
Starting 1 thread
Jobs: 1 (f=1): [w] [100.0% done] [0KB/2264KB/0KB /s] [0/566/0 iops] [eta 00m:00s]
SATA 4K randwrite test: (groupid=0, jobs=1): err= 0: pid=18045: Sun Aug 31 19:08:33 2014
  write: io=136388KB, bw=2272.1KB/s, iops=568, runt= 60006msec
    slat (usec): min=6, max=1166, avg=19.47, stdev= 8.51
    clat (usec): min=362, max=149987, avg=7014.39, stdev=2925.35
     lat (usec): min=388, max=149999, avg=7034.23, stdev=2925.34
    clat percentiles (msec):
     |  1.00th=[    4],  5.00th=[    6], 10.00th=[    6], 20.00th=[    7],
     | 30.00th=[    7], 40.00th=[    7], 50.00th=[    7], 60.00th=[    7],
     | 70.00th=[    8], 80.00th=[    8], 90.00th=[    8], 95.00th=[    9],
     | 99.00th=[   16], 99.50th=[   17], 99.90th=[   36], 99.95th=[   46],
     | 99.99th=[  143]
    bw (KB  /s): min= 1616, max= 3089, per=100.00%, avg=2277.83, stdev=135.86
    lat (usec) : 500=0.10%, 750=0.18%, 1000=0.16%
    lat (msec) : 2=0.04%, 4=0.95%, 10=95.39%, 20=2.76%, 50=0.38%
    lat (msec) : 100=0.04%, 250=0.01%
  cpu          : usr=0.66%, sys=1.57%, ctx=33608, majf=0, minf=5
  IO depths    : 1=0.1%, 2=0.1%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=0/w=34097/d=0, short=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=4

Run status group 0 (all jobs):
  WRITE: io=136388KB, aggrb=2272KB/s, minb=2272KB/s, maxb=2272KB/s, mint=60006msec, maxt=60006msec

Disk stats (read/write):
  sdd: ios=68/34044, merge=0/0, ticks=10/238240, in_queue=238389, util=99.83%
[root@ceph-test ~]#
[root@ceph-test ~]#
```

图9-14　1GB测试结果

我们发现，当设置size=1GB时，IOPS会显著提高到568，IO平均响应时间会降到7ms（队列深度为4）。

这是因为当寻址空间为1GB时，磁头需要移动的距离变小了，每次IO请求的服务时间就降低了，这就是空间局部性原理。假如我们测试的RAID卡或者是磁盘阵列（SAN），它们可能会用Cache把这1GB的数据全部缓存，极大降低了IO请求的服务时间（内存的写操作比硬盘的写操作快1000倍）。所以设置寻址空间为1GB的意义不大，因为我们是要测试硬盘的全盘性能，而不是Cache的性能。

6.硬盘优化

硬盘厂商提高硬盘性能的方法主要是降低服务时间（延迟）。

- 提高转速（降低旋转时间和传输时间）。
- 增加Cache（降低写延迟，但不会提高IOPS）。
- 提高单磁道密度（变相提高传输时间）。

[1] 为什么会变成512KB的I/O，你可以去使用Google去查一下内核参数max_sectors_kb的意义和使用方法。

9.4.4　云硬盘测试

本节我们将会讲述如何对Ceph进行测试，分别从块存储和对象存储这两种不同存储类型、不同测试工具的角度来进行测试。

我们可以通过两个方面来提高云硬盘的性能。

- 降低延迟（使用SSD，使用万兆网络，优化代码，减少瓶颈）。

- 提高并行度（数据分片，同时使用整个集群的所有SSD）。

在Linux系统下，你可以使用FIO来测试。

- 操作系统：Ubuntu 14.04。

- CPU：2。

• Memory：2GB。

• 云硬盘大小：1TB（SLA:6000 IOPS,170MB/s吞吐率）。

测试开始前，我们先安装fio，命令如下。

（1）4K随机写测试

我们首先进行4K随机写测试，具体如下。

```
#fio -ioengine=libaio -bs=4k -direct=1 -thread -rw=randwrite
-size=100G -filename=/dev/vdb \
-name="EBS 4KB randwrite test" -iodepth=32 -runtime=60
```

测试参数和测试结果如图9-15所示。

蓝色方框表示IOPS是5900，在正常的误差范围内。绿色方框表示IO请求的平均响应时间为5.42ms，黄色方框表示95%的IO请求的响应时间是小于等于6.24 ms的。

（2）4K随机读测试

我们再来进行4K随机读测试，如下。

```
#fio -ioengine=libaio -bs=4k -direct=1 -thread -rw=randread
-size=100G -filename=/dev/vdb \
-name="EBS 4KB randread test" -iodepth=8 -runtime=60
```

```
root@ustack:~#
root@ustack:~# fio -ioengine=libaio -bs=4k -direct=1 -thread -rw=randwrite -size=100G
-filename=/dev/vdb -name="EBS 4K randwrite test" -iodepth=32 -runtime=60
EBS 4K randwrite test: (g=0): rw=randwrite, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iod
epth=32
fio-2.1.3
Starting 1 thread
Jobs: 1 (f=1): [w] [100.0% done] [0KB/24419KB/0KB /s] [0/6104/0 iops] [eta 00m:00s]
EBS 4K randwrite test: (groupid=0, jobs=1): err= 0: pid=7191: Sun Aug 31 18:43:51 2014
  write: io=1383.3MB, bw=23601KB/s, iops=5900, runt= 60016msec
    slat (usec): min=2, max=260, avg= 5.14, stdev= 3.88
    clat (msec): min=1, max=206, avg= 5.42, stdev=14.69
     lat (msec): min=1, max=206, avg= 5.42, stdev=14.69
    clat percentiles (usec):
     |  1.00th=[ 1784],  5.00th=[ 2096], 10.00th=[ 2288], 20.00th=[ 2512],
     | 30.00th=[ 2672], 40.00th=[ 2800], 50.00th=[ 2896], 60.00th=[ 3024],
     | 70.00th=[ 3216], 80.00th=[ 3440], 90.00th=[ 4192], 95.00th=[ 6240],
     | 99.00th=[102912], 99.50th=[103936], 99.90th=[122368], 99.95th=[125440],
     | 99.99th=[152576]
    bw (KB  /s): min=12691, max=31392, per=100.00%, avg=23665.27, stdev=3190.72
    lat (msec) : 2=3.31%, 4=85.41%, 10=8.10%, 20=0.39%, 50=0.70%
    lat (msec) : 100=0.13%, 250=1.96%
  cpu          : usr=2.53%, sys=3.72%, ctx=52348, majf=0, minf=8
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued    : total=r=0/w=354105/d=0, short=r=0/w=0/d=0

Run status group 0 (all jobs):
  WRITE: io=1383.3MB, aggrb=23600KB/s, minb=23600KB/s, maxb=23600KB/s, mint=60016msec,
 maxt=60016msec

Disk stats (read/write):
  vdb: ios=0/354073, merge=0/0, ticks=0/2632188, in_queue=1897308, util=99.96%
root@ustack:~#
root@ustack:~#
```

图9-15　4K随机写测试结果

测试参数和测试结果如图9-16所示。

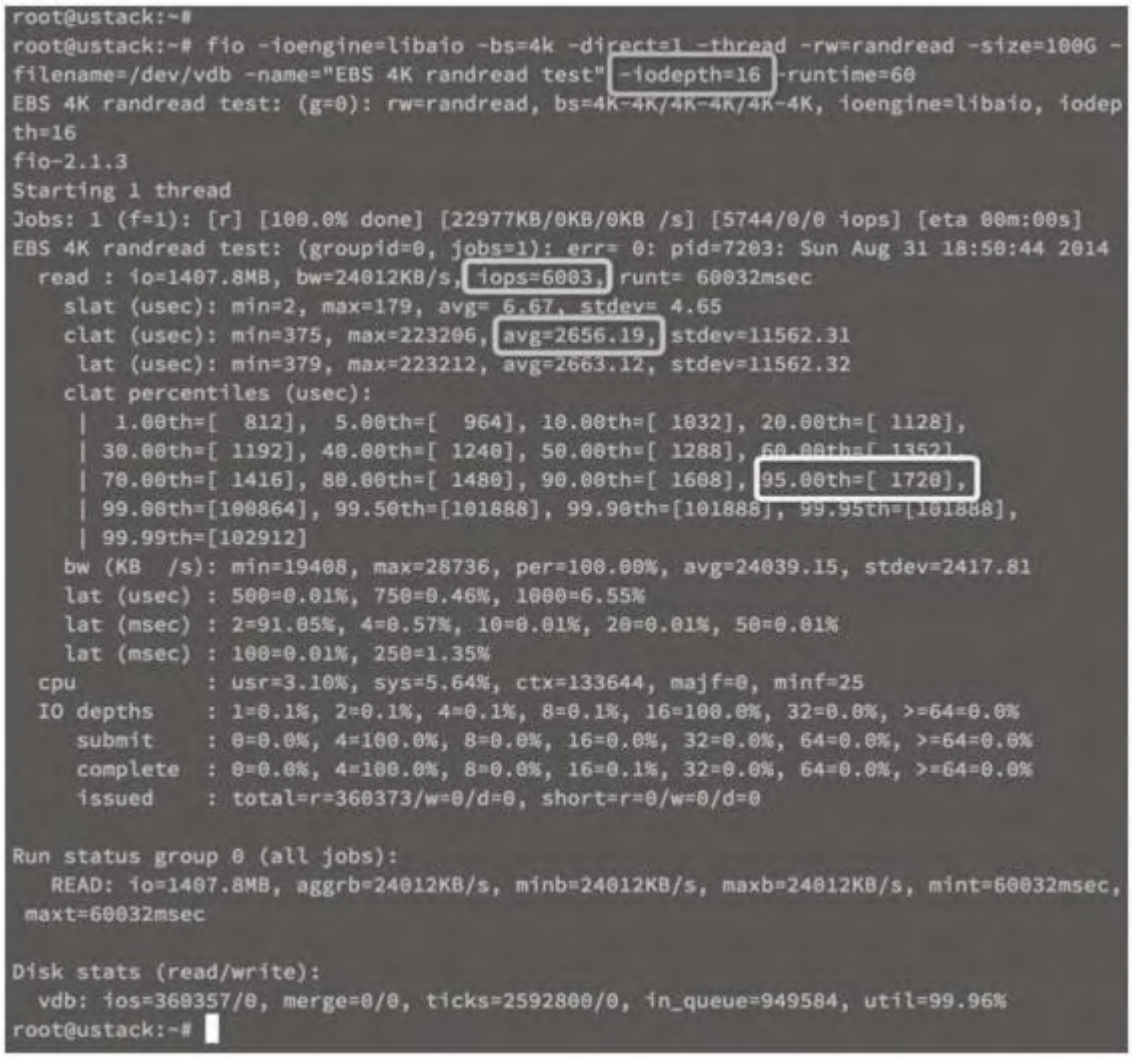

图9-16　4K随机读测试结果

（3）512KB顺序写测试

最后我们来测试512KB顺序写，看看云硬盘的最大MBPS（吞吐率）是多少，具体如下。

```
#fio -ioengine=libaio -bs=512k -direct=1 -thread -rw=write -
size=100G -filename=/dev/vdb \
-name="EBS 512KB seqwrite test" -iodepth=64 -runtime=60
```

测试参数和测试结果如图9-17所示。

蓝色方框表示MBPS为174226KB/s，约为170MB/s。

（4）dd测试吞吐率

其实使用dd命令也可以测试出170MB/s的吞吐率，不过需要设置一下内核参数，详细介绍可以在朱荣译这篇文章中查看（http://way4ever.com/?p=465#dd_170mb ）。

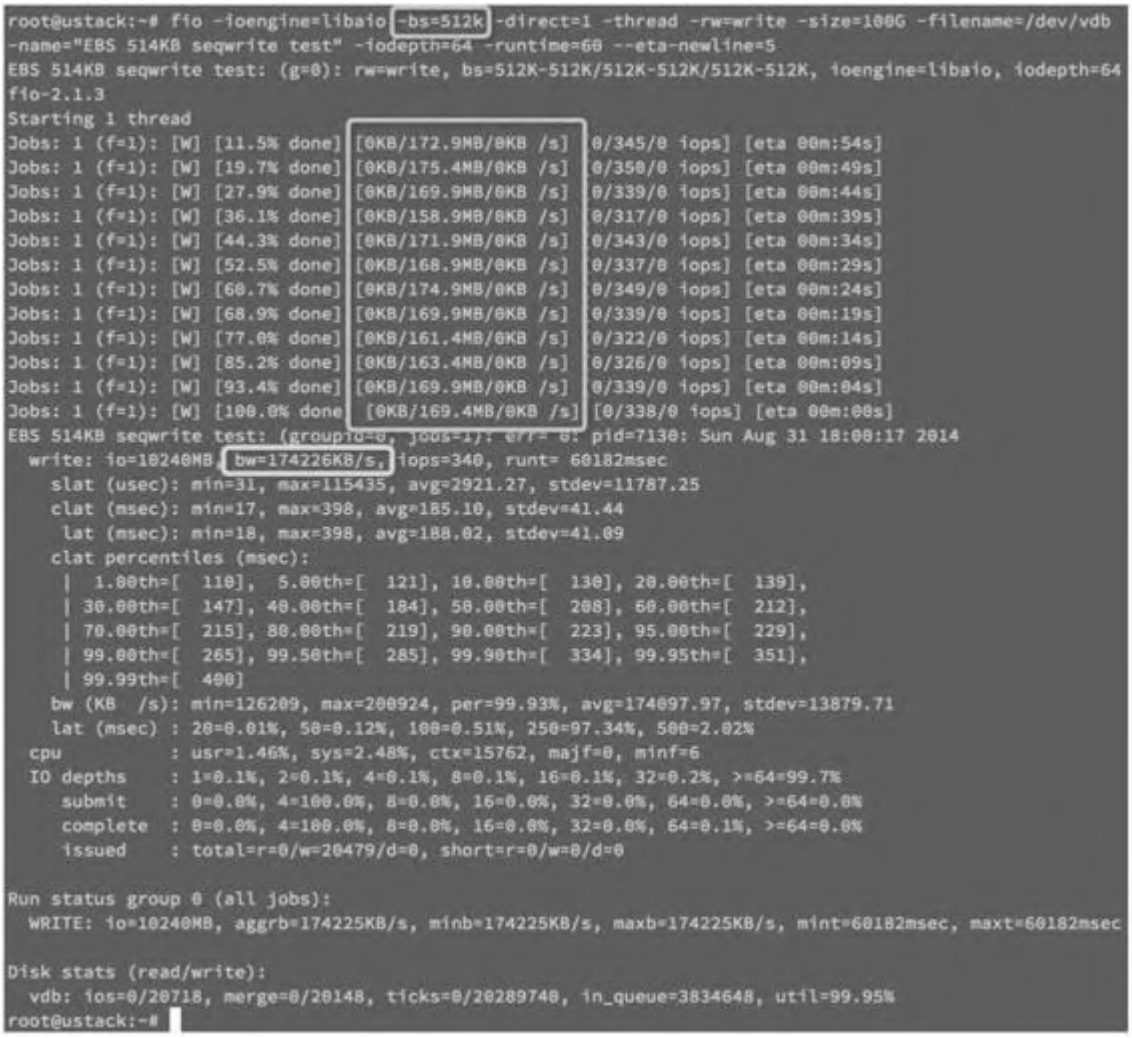

图9-17　512B随机写测试结果

9.4.5　利用Cosbench来测试Ceph

Cosbench是Intel的开源云存储性能测试软件，Cosbench目前已经广泛使用于云存储测试，并作为云存储的基准测试工具使用，Cosbench可在Windows和Linux两种系统中运行，而为了更好地发挥硬件和系统的能力，建议在使用Cosbench进行测试时，选择Linux系统。

Cosbench是一个分布式的基准测试工具，测试云对象存储系统，目前为止它支持一些云对象存储系统的测试，Cosbench也允许用户创建额外的存储系统适配器。

1.下载安装包

安装包下载地址：https://github.com/intel-cloud/cosbench/releases 。

注意，下载二进制包而非源码包，下载完成后解压对应文件到cosbench目录，以/home/demo/cosbench为例，使用的版本是v0.4.2.0 release candidate 3。本节是在Debian 8 64位系统之上进行操作的。

2.安装配置

1）controller配置。

```
#安装软件包
```

```
apt-get install openjdk-7-jre curl
cat /home/demo/cosbench/conf/controller.conf
[controller]
drivers = 10 #注意
```

driver的数量与下面的内容对应

```
log_level = INFO
log_file = log/system.log
archive_dir = archive
[driver1]  #第
```

1个

```
driver
name = driver1
url = http://10.63.48.11:18088/driver  #
...
[driver10] #第
```

```
10个
```

```
driver
name = driver10
url = http://10.63.48.20:18088/driver  #
```

2）driver配置。

```
cat /home/demo/cosbench/conf/driver.conf
[driver]
name=127.0.0.1:18088
url=http://127.0.0.1:18088/driver
```

3）其他配置。

目前版本默认启动driver和controller服务会出现服务进程卡住的bug，需要修改cosbench-start.sh，修改内容如下：

```
/home/demo/cosbench/cosbench-start.sh
35      - TOOL_PARAMS=""
35      + TOOL_PARAMS="-q 1"
```

3. 启动服务

启动服务命令如下：

```
chown +x /home/demo/cosbench/*.sh
/home/demo/cosbench/start-all.sh
```

出现如下字样，则证明安装成功：

```
Successfully started cosbench controller!
Listening on port 0.0.0.0/0.0.0.0:19089 …



Persistence bundle starting…
```

```
Persistence bundle started.
-
!!! Service will listen on web port: 19088 !!!
-
```

4.提交测试任务

测试任务的提交可以通过Web界面或者命令行，这里以命令行为例。

1）新建测试用例文件s3test.xml，注意替换其中的accesskey、secretkey、endpoint信息，具体如下：

```
<?xml version="1.0" encoding="utf-8"?>
<workload name="buckets*objects=100*100 size=4k
num_workers=100 read:write=30:70"
description="buckets*objects=100*100 size=4k num_workers=100
read:write=30:70">
  <storage type="s3" config="accesskey=
{accesskey};secretkey=
{secretkey};proxyhost=;proxyport=;endpoint=http://{endpoint}
"/>
  <!-- Small Objects, Write & Read -->
  <workflow>
    <!-- 依次新建
```

100个

bucket，名字从

gftest1~

gftest100，并发数为

8 -->
    <workstage name="create-buckets">
        <work type="init" workers="8"
config="cprefix=gftest;containers=r(1,100)"/>
    </workstage>
    <!-- 向已经建立好的每个

bucket依次写入

100个

object用于后面的读取测试，

object名称为

myobjects1~myobjects100，大小为

```
4KB -->
    <workstage name="prepare" >
        <work type="prepare" workers="100"
config="cprefix=gftest;containers=r(1,100);objects=r(1,100);
sizes=c(4)KB"/>
    </workstage>
    <!-- 等待一段时间，保障后台数据写入完毕
```

```
  -->
    <workstage name="delay" closuredelay="600" >
        <work type="delay" workers="1"/>
    </workstage>
    <!-- 测试时长为
```

3600秒，读写比例为

30%:70%，其中随机读取的数据范围是从

gftest[1-100]/
    myobjects[1-100]，以

random方式填充写入的数据并进行

Hash校验。

```
  -->
    <workstage name="main"  >
        <work name="main" workers="100" runtime="3600" >
            <operation type="read" ratio="30"
config="cprefix=gftest;containers=r(1,100);objects=u(1,100)"
/>
            <operation type="write" ratio="70"
config="cprefix=gftest;containers
=r(1,100);objects=r(1,100);content=random;hashCheck=True;siz
es=c(4)KB"/>
      </work>
    </workstage>
    <!-- 删除
```

```
 objects -->
    <workstage name="cleanup" >
        <work type="cleanup" workers="64"
config="cprefix=gftest;containers= r(1,
100);objects=r(1,100)" />
    </workstage>
    <!-- 删除
```

```
 buckets -->
    <workstage name="dispose" >
        <work type="dispose" workers="64"
config="cprefix=gftest;containers= r(1,100)" />
    </workstage>
  </workflow>
</workload>
```

2）提交任务。

```
/home/demo/cosbench/cli.sh submit s3test.xml
Accepted with ID: w179   #记录
```

```
ID
```

5. 查看任务运行情况

打开controller对应主机的管理页面，地址格式为http://{controller_IP}:19088/controller/ ，登录到管理界面，在Active Workloads可以看到对应的任务运行，如图9-18所示。

点击view details可以查看详情，如图9-19所示。

图9-18　查看运行的任务

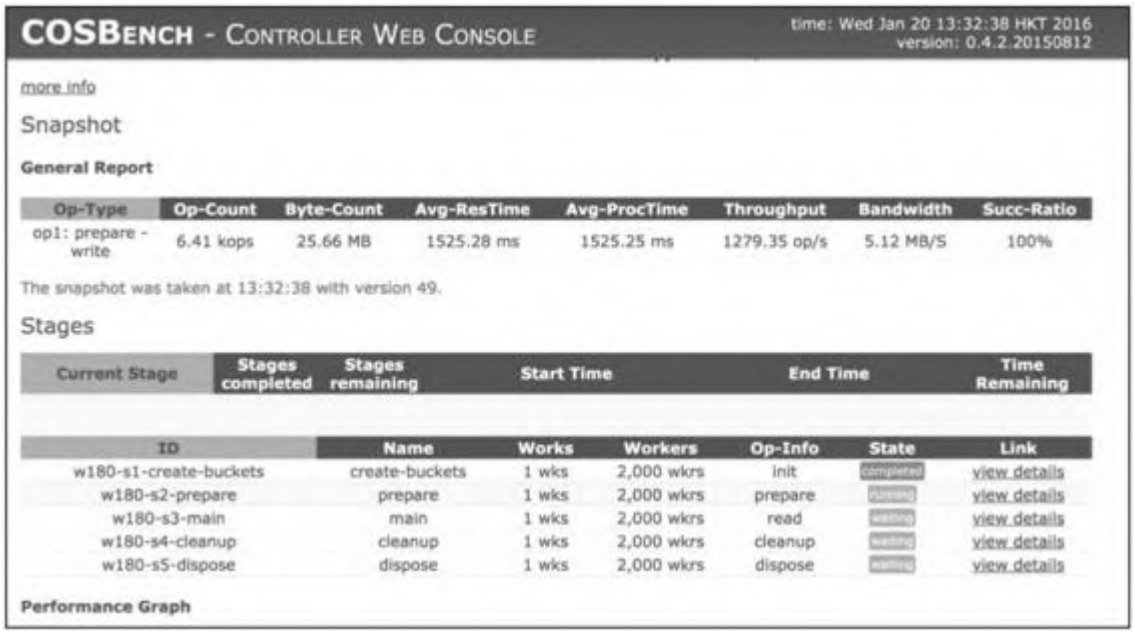

图9-19　查看详情

9.5 本章小结

本章阐述了Ceph的硬件选型、性能调优以及Ceph测试的内容，优化是一个长期迭代的过程，所有的方法都是别人的，只有在实践过程中才能发现自己的方法。本章也介绍了一些测试工具和一些观点，希望学习完本章之后能够对你有所帮助。

参考资料

[1]http://xiaoquqi.github.io/blog/2015/06/28/ceph-performance-optimization-summary/ 。

[2]https://community.mellanox.com/docs/DOC-2141 。

[3]http://way4ever.com/?p=465 。

[4]https://community.mellanox.com/docs/DOC-2141 。

第10章　自定义CRUSH

本章从定性和定量两个维度让大家了解Ceph的稳定性与可靠性，通过几个CURSH设计实例（两副本结构、SSD和SATA混合结构）让大家充分理解Ceph软件定义存储的特性。

10.1 CRUSH解析

数据的可靠性是所有存储系统的首要指标，本节通过Ceph与一致性Hash的对比以及从CRUSH层面的定量分析，展示Ceph在数据可靠性方面的特点。我们先来看一下一致性Hash的原理。

1.一致性Hash算法

一致性Hash算法根据一个叫作一致性Hash环的数据结构来实现key到存储节点的映射，如图10-1所示。

算法过程如下： 首先会构造一个长度为0~232 - 1的整数空间，然后首尾相连，形成一个封闭的环。根据存储节点的名称经过Hash处理，取得一个值（其值也在封闭环范围内），然后把存储节点按这个值映射到Hash环上（如图10-1中的node0、node1和node2）。

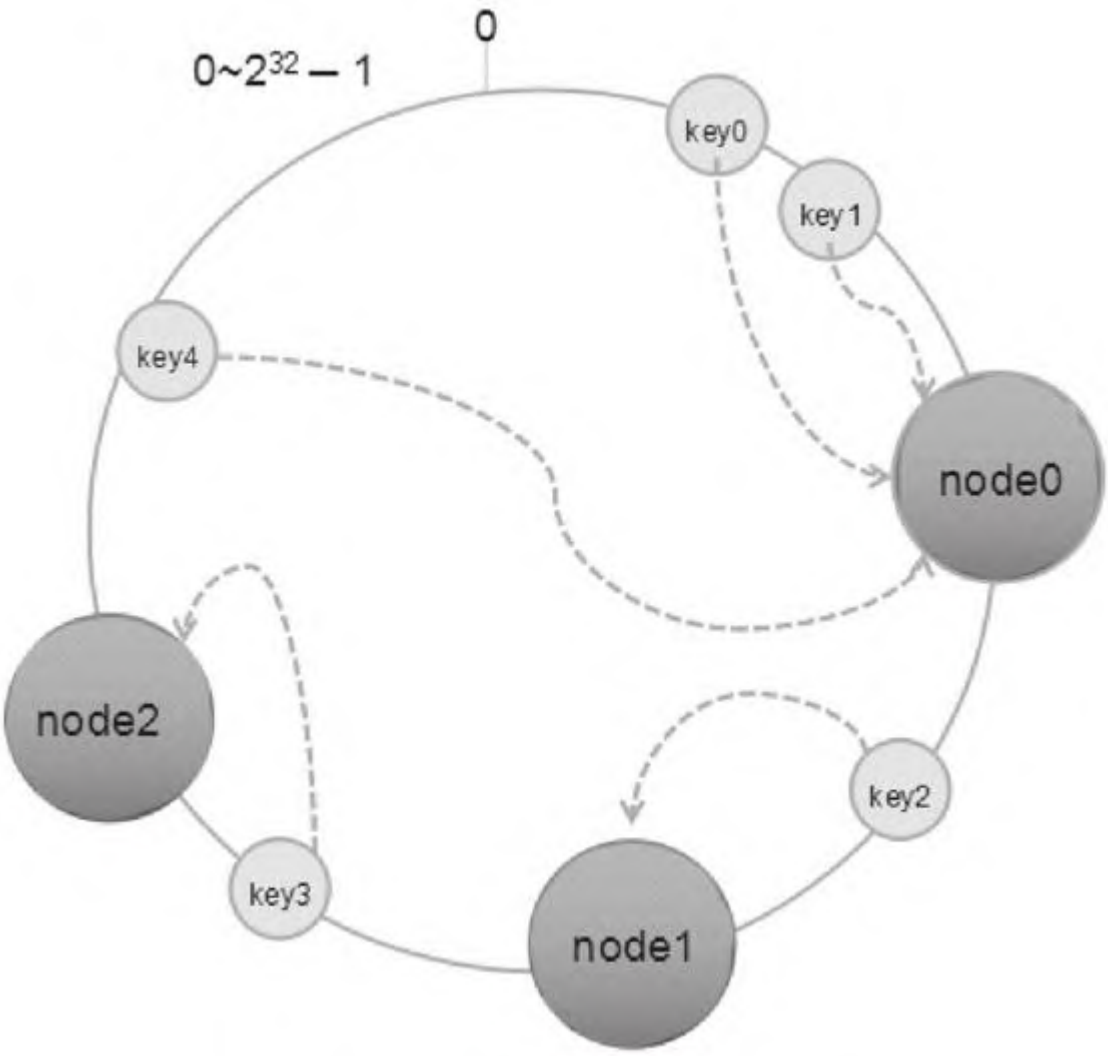

图10-1　key到存储节点的映射

假设数据为x（如图10-1中的key0~key4），存储节点数目为N（图10-1所示为3）。将数据分布到存储节点的最直接做法是，计算数据x的Hash值，Hash值落在哪个区间内（如图10-1中的key0落在[node2，

node0]，key2落在[node0，node1]），就把这个x存入相应的存储节点中。

 注意

[node2，node0]空间实际可划分为[node2，2^{32} - 1]和[0，node0]。

当有新的存储节点添加与删除时，一致性Hash算法（相较传统的取余算法）并不会引起全局数据的迁移，只会涉及部分数据转移，这对全局影响相对较小，如图10-2所示。

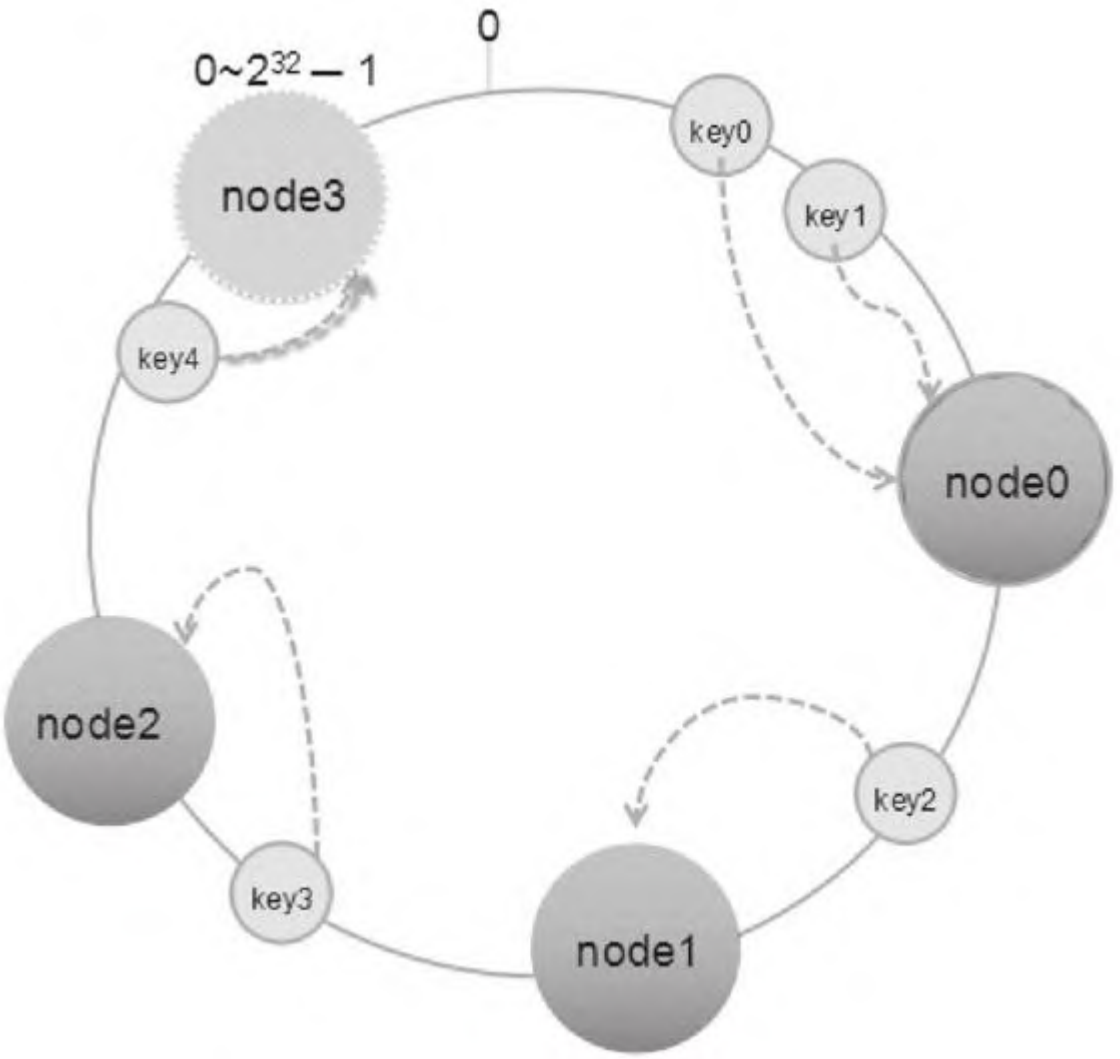

图10-2　存储节点变更引发的数据迁移

当加入新节点时（如图10-2中的node3），原本[node2，node0]的存储空间被划分成[node2，node3]、[node3，node2]两个区域，原来映射到node0的key4映射到了新的存储节点（node3）上，而其他的

数据如图10-2中的key0、key1、key2、key3并不会引起数据迁移。

当有节点被剔除时，例如图10-3所示的node2，其所存储的key4会被重新映射到[node1，node0]区域，存储到node0节点上，其他数据也不会变更。

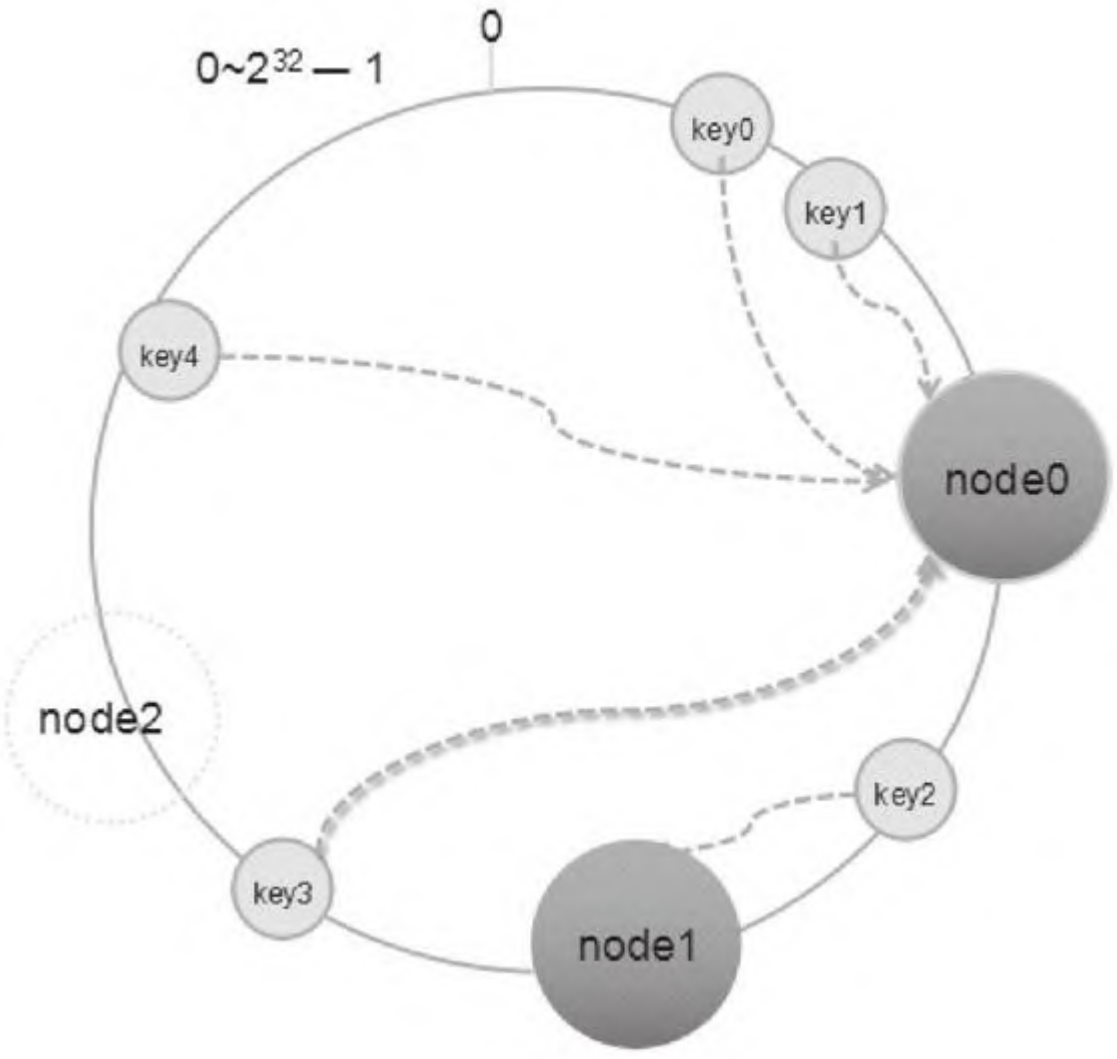

图10-3　node2被删除

但一致性Hash的一个问题是，存储节点不能将Hash空间均匀划分。如图10-3所示，[node1，node0]的空间的比[node0，node1]大得多，这容易让负责该分区的节点node0负载过重。假设两个节点的磁盘容量相等，那么当节点node0的存储空间满载时，node1的磁盘还有很大的空闲空间，但此时系统已经无法继续向[node1，node0]写入数据，从而造成资源浪费。

2. 虚拟节点

计算机界有句名言：计算机的任何问题都可以通过增加一个虚拟层来解决，计算机硬件、计算机网络、计算机软件等方面概莫如此。为解决一致性Hash算法带来的资源使用分配不均匀的问题，也可以引入虚拟化的手段。虚拟节点是相对于物理存储节点而言的，虚拟节点负责的分区上的数据最终存储到其对应

的物理节点。一台物理节点可以虚拟出一个或多个虚拟节点（如图10-4所示，每个物理节点只虚拟一个）。新的空间为[v0，v1]、[v1，v2]、[v2，v0]，分别映射到node0、node1、node2物理存储节点。如此一来，3个物理节点的数据承载趋于均匀。

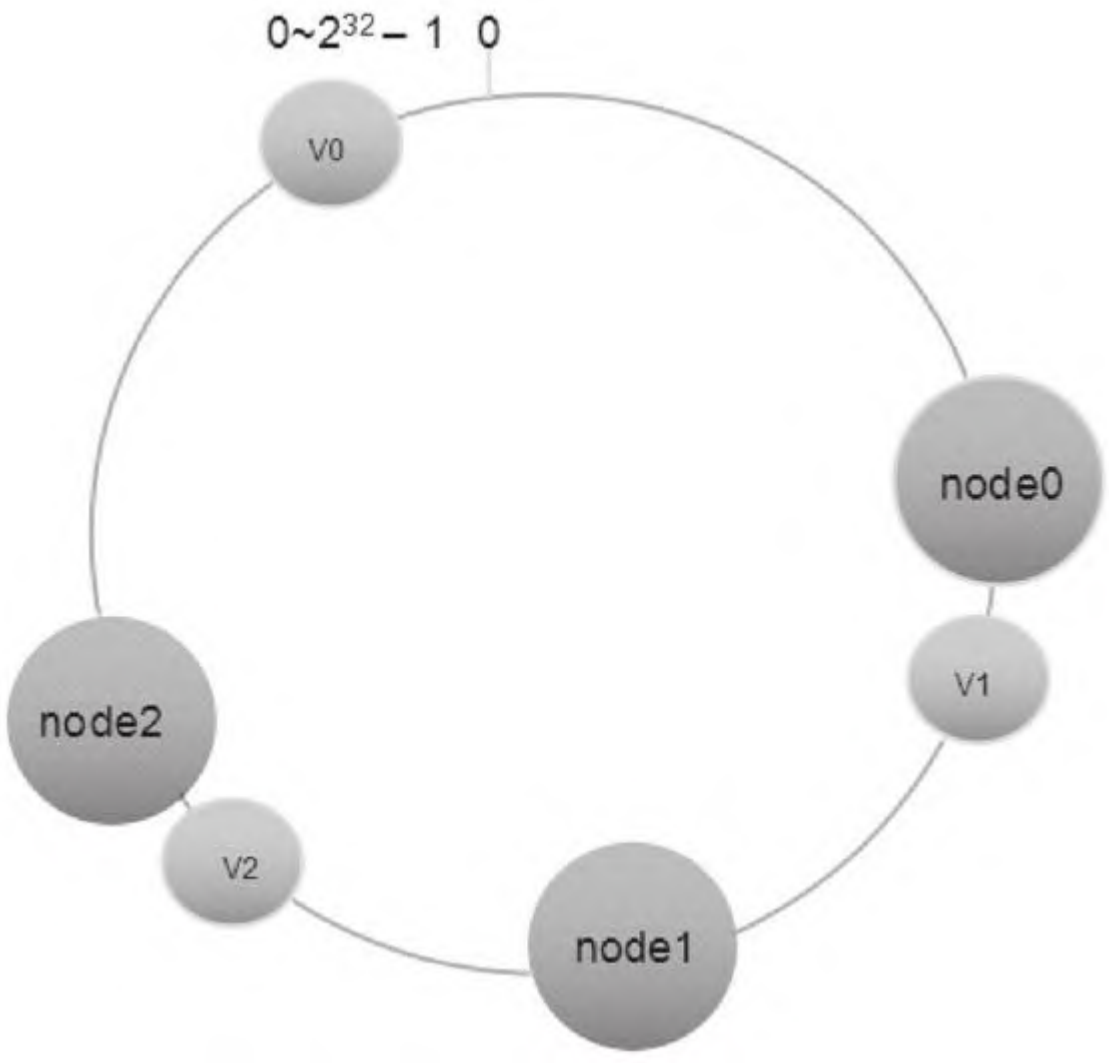

图10-4　虚拟节点

在实际应用中，可以根据物理节点的磁盘容量的大小来确定其对应的虚拟节点数目。虚拟节点数目越多，节点负责的数据区间也越大。

3. CRUSH算法

Ceph的数据分布流程如图10-5所示。

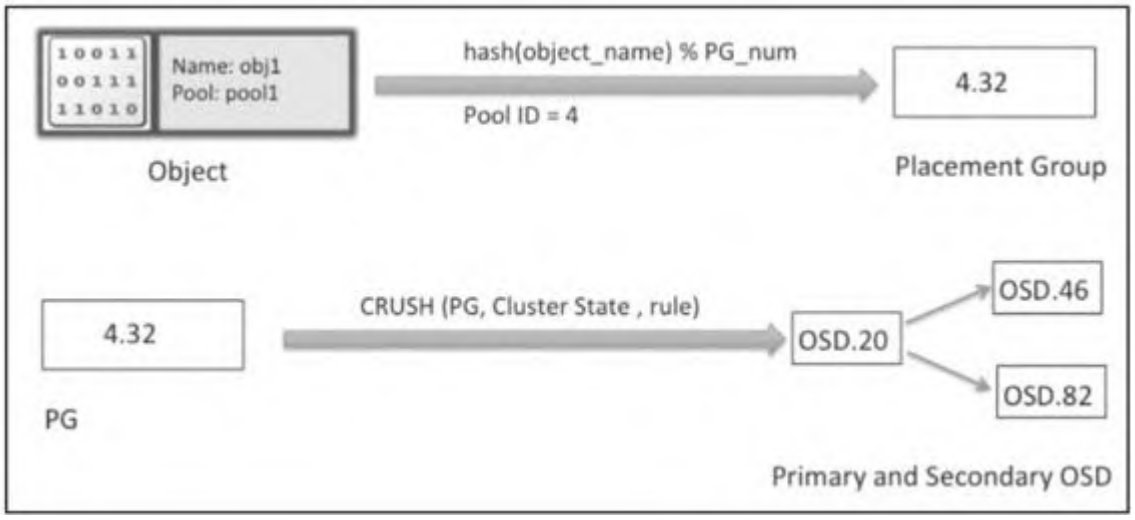

图10-5　Ceph的数据分布流程

整个流程如下。

1）取得Object的name进行Hash运算。

2）取得的Hash值与PG数取余，得到的结果与Pool ID结成PG的编号（如图10-5中的4.32）。

3）通过CRUSH算法，把PG映射到具体的OSD。

在Ceph里，PG是数据存储的管理单元，如果把PG当作一致性Hash里的存储节点，那么它就是最简单的数据分布（即取余算法）方式。不同的是，PG是抽象的存储节点，它不会随着物理节点的加入或者离开而增加或减少，因此数据到PG的映射是稳定的。

当在Ceph里创建存储池（指定PG的数量），整个Hash环就固定了，如图10-6所示，对象到PG的映射就唯一确认，即数据具体存储到哪个PG是确定的，不会随着下层OSD的增删而改变，这充分体现了Ceph在数据可靠性的特征之一。

注意

在实际环境中，切勿随便变更Pool的PG数量，这将引起数据的大规模迁移。

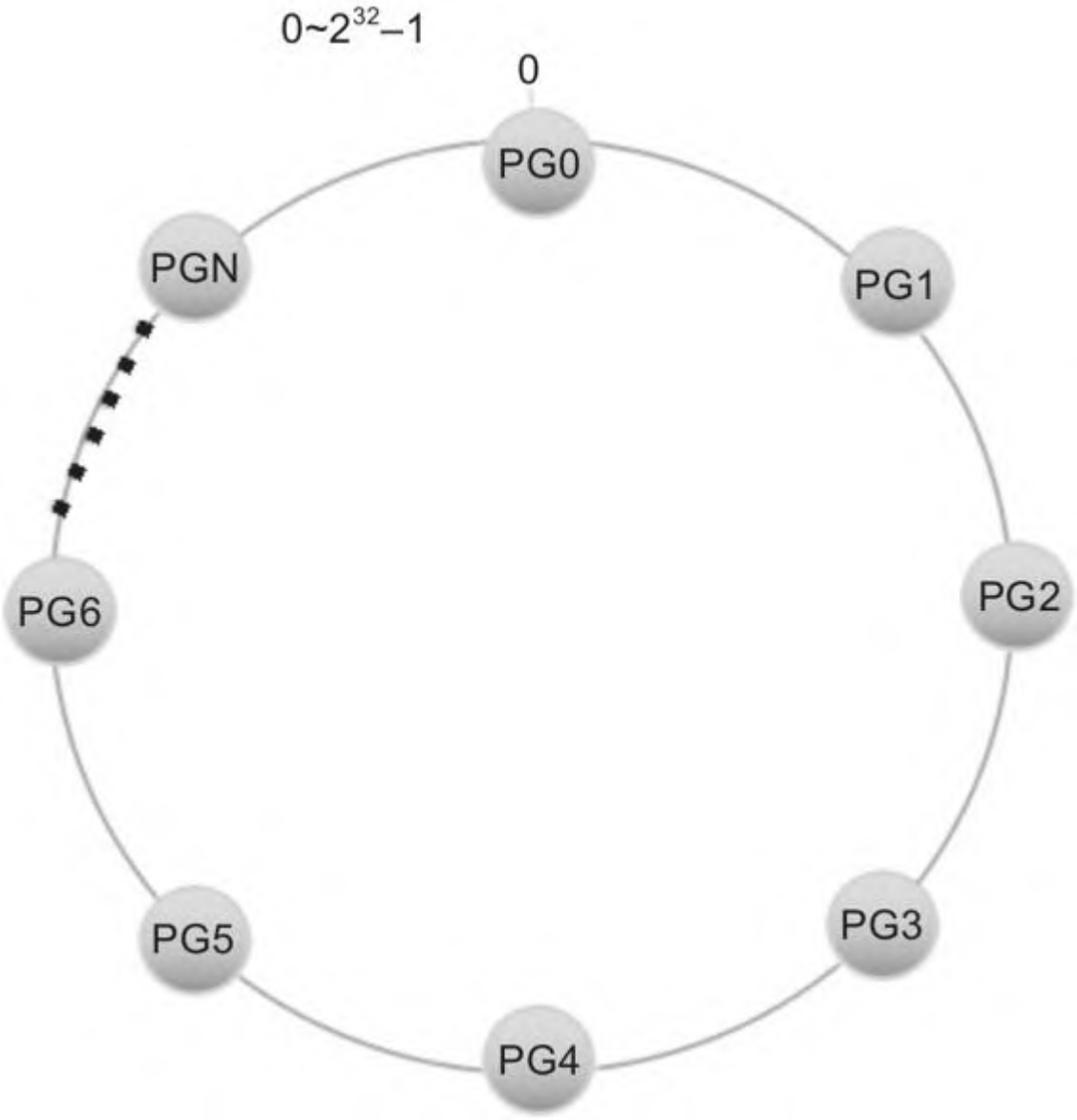

图10-6　Hash环中的PG

4.CRUSH定量分析

根据网上公开的资料，复杂推演PG的丢失率（即数据丢失的概率）过程在此不一一列出，读者只要记住最后的公式即可，如下。

丢失PG的概率的公式：P=Pr*M/C（R，N）

其中：

1）Pr表示R个（副本数）OSD故障的概率，跟它关联的因素有OSD硬盘故障概率及恢复（Recovery）期间其他（R－1）OSD的故障概率。

2）M表示Copy Set组合数，PG映射到一组OSD里表示Copy set，如pg2.3→[2，3，1]。

3）C（R，N）表示N个OSD中，任意R个OSD的组合数。

从公式看，C（R，N）对于一个既定的Ceph集群值是不变的（从侧面也证明副本数越多，集群规模越大，PG丢失的概率也越低），所以尽量缩小Pr、M的值。

从上面看出Pr值关联因素，一是缩小OSD硬盘的故障率，二是缩短恢复的时间。

缩小OSD故障率就是选择可靠的硬盘。

恢复时间的多与少取决于故障域的大小。默认的CRUSH的故障域是Host级。对于一个Host有3个OSD的主机，同时可恢复的OSD就是两个，如图10-7所示。

图10-7　故障恢复

注意

为什么故障恢复的时候只能在同一故障域内来完成呢？原因是当OSD故障时，OSDWEIGHT的值并不会更新，故障域的权重没有变化，PG映射到故障域层级没有变化，从而不会进行Rebalance，因此，故障的OSD上的PG只会转移到同一故障域内的其他OSD。

扩大故障域可以缩短恢复时间，如此即可以减小PG的丢失概率，如图10-8所示。

图10-8　故障恢复

在CRUSH中添加一个osd-region级的Bucket（osd-region是自定义的层级）。

```
root  default
rack  rack-01
   osd-region  osdrg-01    ####自定义的
```

```
osd-region bucket ###
       osd.0 up 1
       osd.1 up 1
       osd.2 up 1
       osd.3 up 1
       osd.4 up 1 …
```

```
rack  rack-02
    osd-region   osdrg-02
   …
```

```
    osd-region  osdrg-03…
```

```
# rules
rule replicated_ruleset {
    ruleset 0
    type replicated
```

```
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type osd-region  ###指定选择的故障域
```

```
     ###
        step set_choose_tries 200
        step emit
    }
```

接下来再看看copy set。

假设一个Ceph集群里有3个Rack，每个Rack有8个Host，每个Host含有3个OSD。

如果按默认的CRUSH规则（故障域为Host），R=3，M=C（24，3）×（3×3×3）=54648，如图10-9所示。

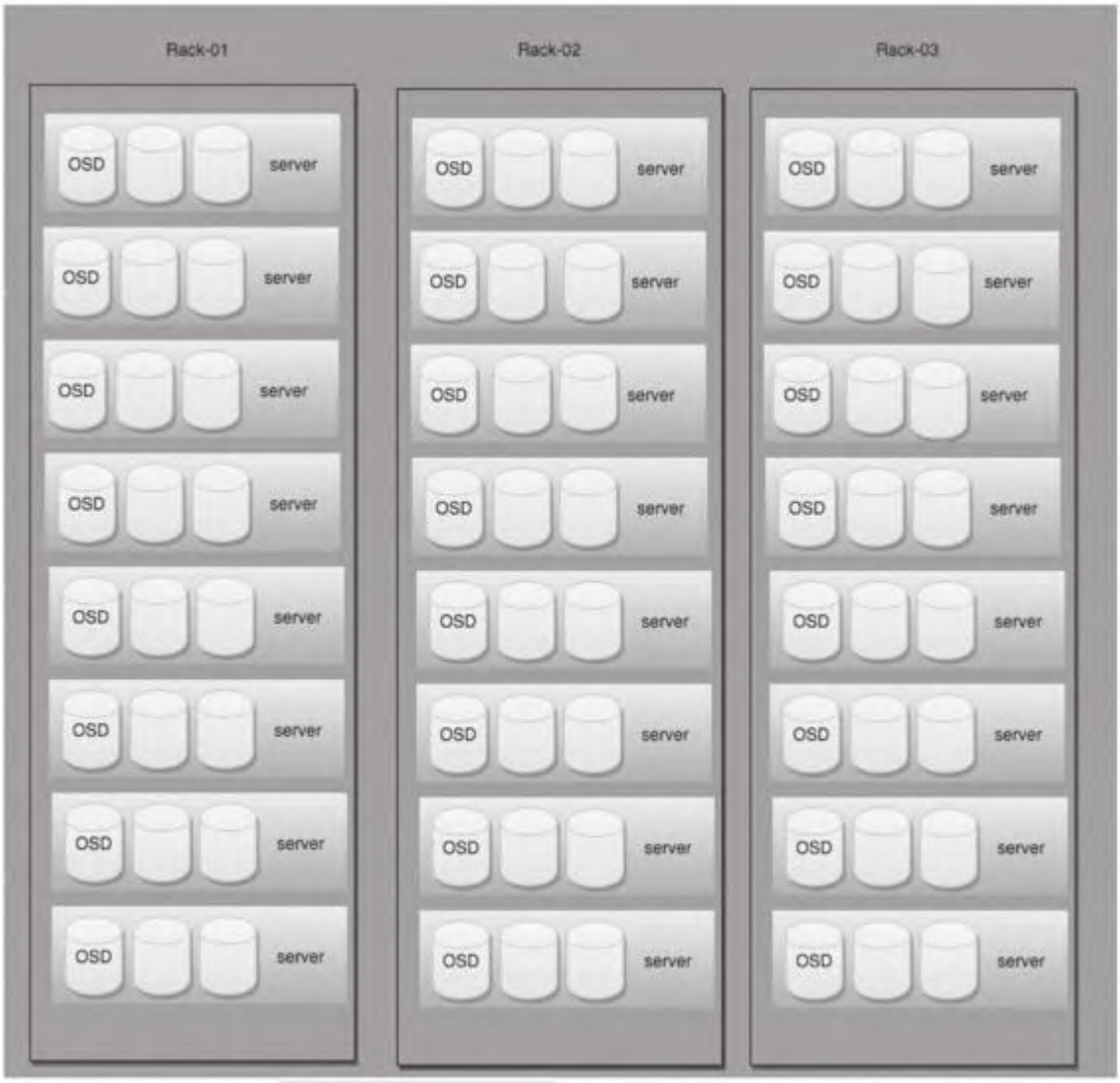

图10-9　默认故障域

如果把数据分散在不同的Rack中，如图10-10所示，此时M=24×24×24=13824。

如果把PG再次做分落限制，设置一个中间层的虚拟域replication-region，在虚拟域里设置故障域osd-region，如图10-11所示，此时copy set的组合就更小了，M=2×（12×12×12）=3456。

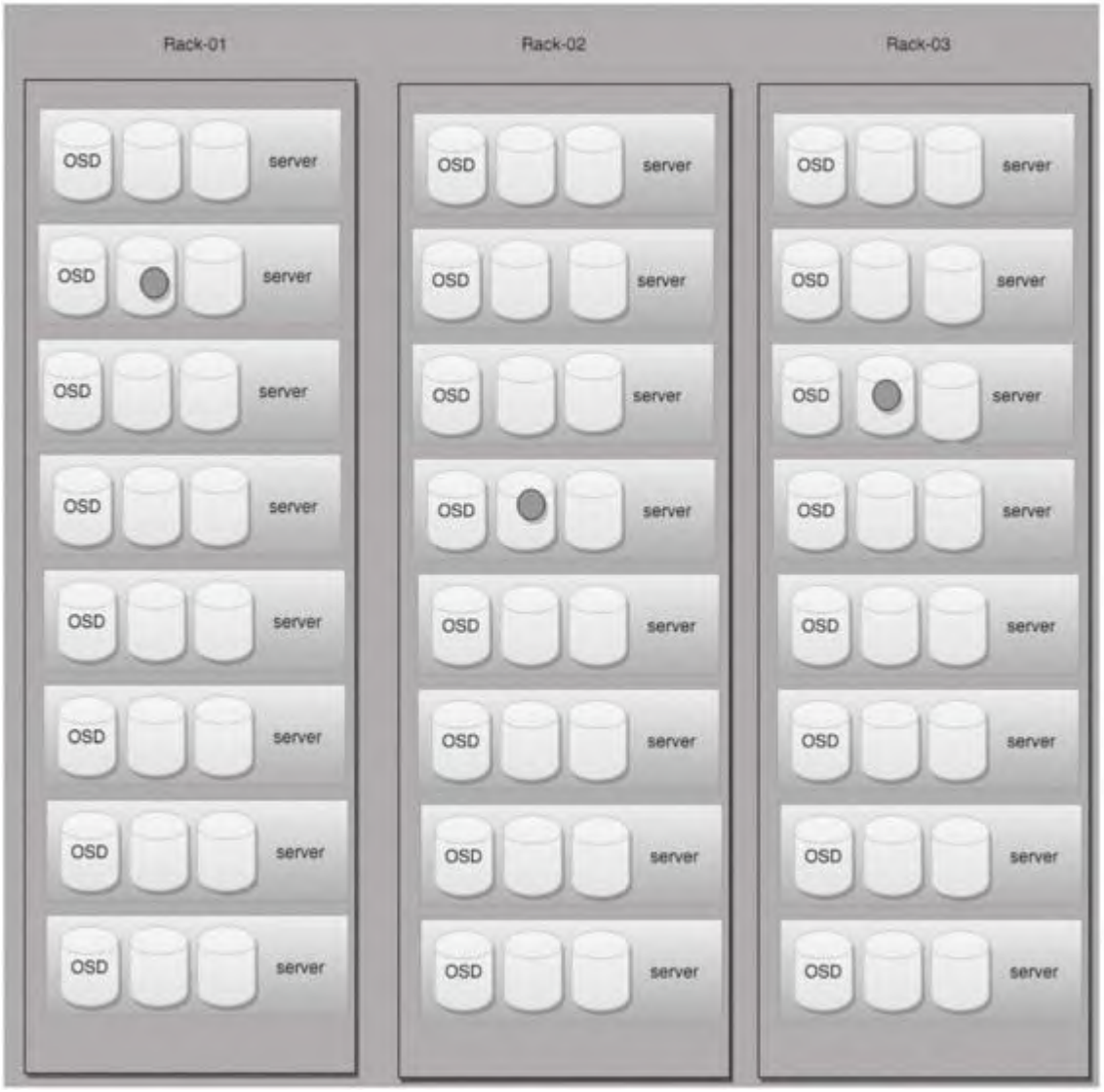

图10-10　Rack故障域

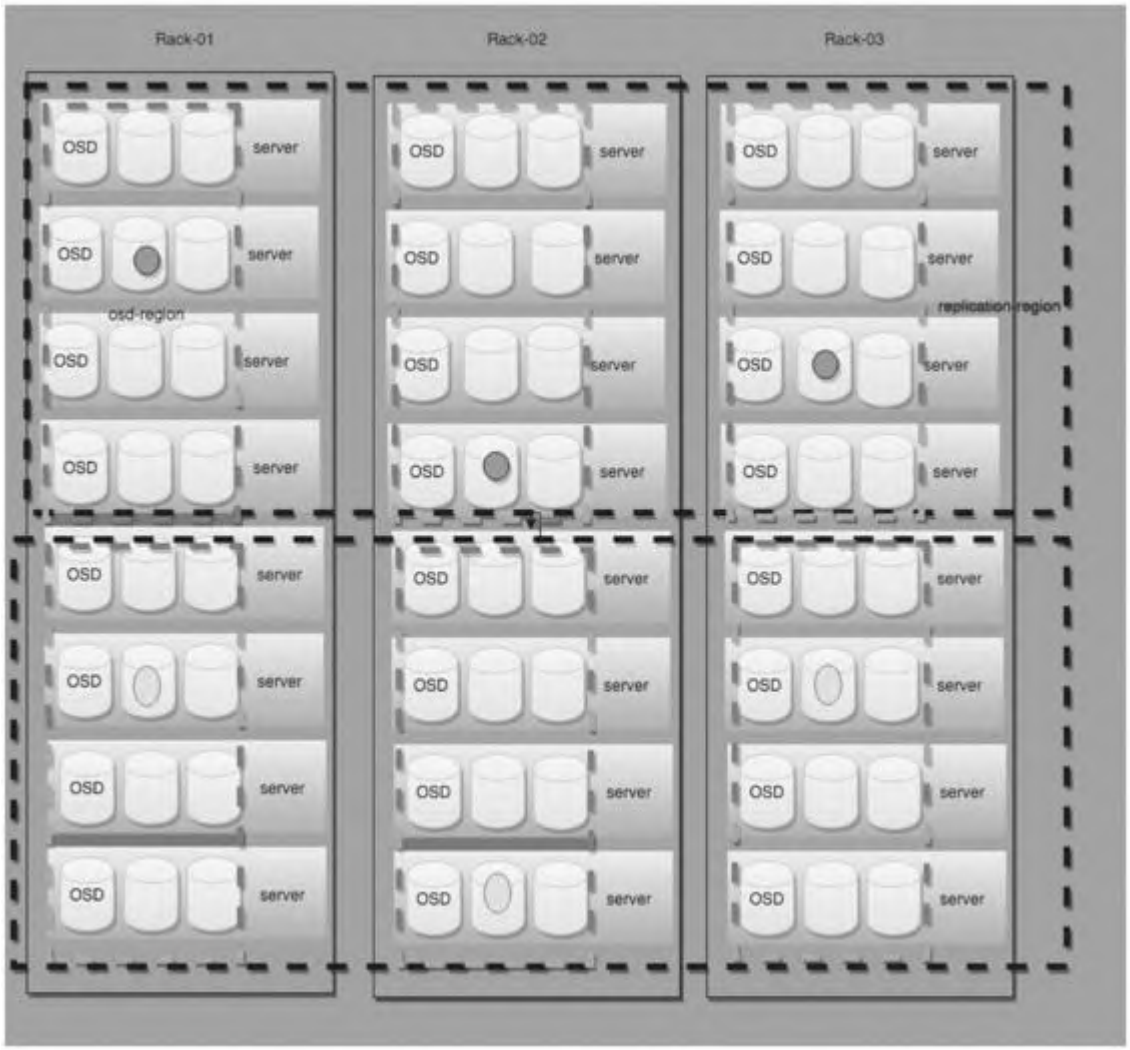

图10-11　虚拟故障域

添加了osd-region和replication-region虚拟域的CRUSH Map如下。

```
root first-zone
   replication-region  rep-01
       osd-region  osdrg-01  ### osd-region里有
```

12个

OSD ###

```
           osd.0    up    1
           …
```

```
           osd.11   up    1
        osd-region  osdrg-03
           osd.24   up    1
           …
```

```
           osd.35   up    1
        osd-region  osdrg-05
           osd.48   up    1
           …
```

```
            osd.59   up    1
    replication-region  rep-02
           osd-region  osdrg-02
           …









           osd-region  osdrg-04
           …









           osd-domain  osdrg-06
rule first-zone {
    ruleset 0
    type replicated
    min_size 1
    max_size 10
    step take  first-zone
    step choose  firstn 1  type replication-region ###选择一个
```

```
replication-region
 ###
    step chooseleaf  firstn 0  type osd-region  ###在所选
```

rep里选择

osd-region并

最终取

```
OSD ####
    step set_choose_tries 200
```

```
        step emit
    }
```

从表10-1可知，通过设置合适Crush Map可以有效地提高数据的可靠性。从P=Pr*M/C(R,N)可知，示例中仅仅通过扩大故障域、添加虚拟域的方式就比默认Host故障域在数据可靠性方面至少提高了100倍以上。

以下是基于3个柜，每柜8台主机，每主机3个OSD（假设定义为zone单元）设计的Crush结构。若规模进一步扩大，可以横向以Zone模式扩展。实际生产环境中的Crush架构要基于现有硬件规模与网络拓扑来规划，示例可供参考。

表10-1　copy set组合表

故障域	copy set	备注
Host	M=C(24,3)×(3×3×3)= 54 648	
Rack	M=24×24×24 = 13 824	
osd-region	M=2×(12×12×12) = 3456	添加 replication-region 虚拟域

10.2　CRUSH设计：两副本实例

在一般的生产环境中，为了保证数据的安全性及可靠性，常用三副本的方案来设计与实施。但有时候基于成本考虑及在业务数据的可靠性不是很高的情况下，会考虑用两副本的方案来实现。以下就以两副本来设计与实施CRUSH的方案。

1.硬件环境

表10-2是所需的硬件环境配置表。

表10-2　硬件环境配置表

存储节点主机名	OSD编号	对应的设备
storage-101	osd.0	sdb
storage-101	osd.1	sdc
storage-102	osd.2	sdb
storage-102	osd.3	sdc
storage-103	osd.4	sdb
storage-103	osd.5	sdc
storage-104	osd.6	sdb
storage-104	osd.7	sdc

我们为此添加Rack级的Bucket，分别包含两个存储节点（以Host的Bucket），然后以Rack为隔离域，保证两个副本分别落在不同的Rack上。CRUSH的逻辑架构如图10-12所示。

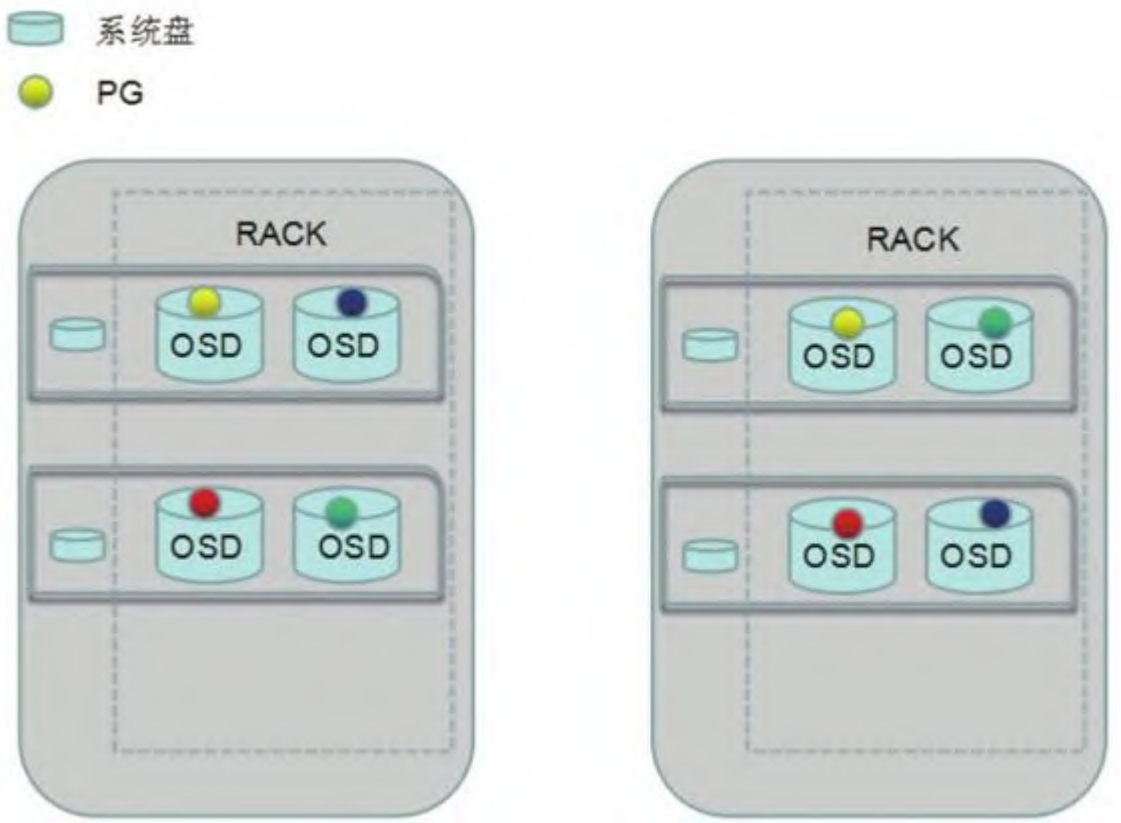

图10-12　CRUSH的逻辑架构

2. 部署

1）利用ceph-deploy工具快速部署、添加OSD。

```
[root@ceph-deploy ceph-deploy]# ceph-deploy osd --zap-disk
create
storage-101:sdb storage-101:sdc storage-102:sdb storage-
102:sdc \
storage-103:sdb storage-103:sdc storage-104:sdb storage-
104:sdc
[ceph_deploy.conf][DEBUG ] found configuration file at:
/root/.cephdeploy.conf
[ceph_deploy.cli][INFO ] Invoked (1.5.24): /usr/bin/ceph-
deploy osd --zap-disk create storage-102:sdb
[ceph_deploy.osd][DEBUG ] Preparing cluster ceph disks
storage-102:/dev/sdb: [ceph-deploy][DEBUG ] connected to
host: storage-101
[ceph-deploy][DEBUG ] detect platform information from
remote host
[ceph-deploy][DEBUG ] detect machine type
[ceph_deploy.osd][INFO ] Distro info: CentOS Linux 7.1.1503
Core
[ceph_deploy.osd][DEBUG ] Deploying osd to storage-101 ....
以下忽略!
```

>

2）成功添加OSD之后，查看一下CRUSH的默认拓扑结构。

3）再次查看集群的拓扑结构。

```
[root@ceph-deploy ceph-deploy]# ceph osd tree
ID WEIGHT TYPE NAME UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 0.72000     root default
-2 0.18000             host storage-101
0 0.09000                     osd.0 up 1.00000 1.00000
1 0.09000                     osd.1 up 1.00000 1.00000
-3 0.18000             host storage-102
2 0.09000                     osd.2 up 1.00000 1.00000
3 0.09000                     osd.3 up 1.00000 1.00000
-4 0.18000             host storage-103
4 0.09000                     osd.4 up 1.00000 1.00000
5 0.09000                     osd.5 up 1.00000 1.00000
-5 0.18000             host storage-104
6 0.09000                     osd.6 up 1.00000 1.00000
7 0.09000                     osd.7 up 1.00000 1.00000
```

4）获取当前的Crush Map。

利用Ceph工具获取当前Crush Map。

```
[root@ceph-deploy ~]#ceph osd getcrushmap -o /tmp/mycrushmap
got crush map from osdmap epoch 14
```

5）反编译Crush Map。

/tmp/mycrushmap是一个二制文件，需要通过crushtool反编译为文本文件。

```
 [root@ceph-deploy ceph-deploy] crushtool -d /tmp/mycrushmap
> /tmp/mycrushmap.txt
```

6）查看CRUSH。

直接通过文件编辑工具（如vim）查看即可。

```
# begin crush map
tunable choose_local_tries 0
tunable choose_local_fallback_tries 0
tunable choose_total_tries 50
tunable chooseleaf_descend_once 1
tunable straw_calc_version 1
# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3
device 4 osd.4
device 5 osd.5
device 6 osd.6
device 7 osd.7
# types ##默认设置的
```

Bucket层级

##

```
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
# buckets
host storage-101 { ##Host层级
```

```
##
     id -2 # do not change unnecessarily
     # weight 0.180
     alg straw
     hash 0 # rjenkins1
     item osd.0 weight 0.090
     item osd.1 weight 0.090
}
host storage-102 {
     id -3 # do not change unnecessarily
     # weight 0.180
     alg straw
     hash 0 # rjenkins1
     item osd.2 weight 0.090
     item osd.3 weight 0.090
}
host storage-103 {
     id -4 # do not change unnecessarily
     # weight 0.180
     alg straw
     hash 0 # rjenkins1
     item osd.4 weight 0.090
     item osd.5 weight 0.090
}
```

```
host storage-104 {
    id -4 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.6 weight 0.090
    item osd.7 weight 0.090
}
root default {
    id -1 # do not change unnecessarily
    # weight 0.720
    alg straw
    hash 0 # rjenkins1
    item storage-101 weight 0.180
    item storage-102 weight 0.180
    item storage-103 weight 0.180
    item storage-104 weight 0.180
}
# rules ##默认的
```

```
crush rule ##
rule replicated_ruleset {
    ruleset 0
    type replicated
    min_size 1
    max_size 10
    step take default
    step chooseleaf firstn 0 type host
    step emit
}
# end crush map
```

7）编辑CRUSH。

①添加自定义的Rack层级，依据我们的逻辑图设计，并把原来的Host层级的storage-101和storage-102划分到rack-01、storage-103和storage-104划分到rack-02。

```
rack rack-01 { ## rack 层级




 ##
    id -5 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item storage-101 weight 0.180
    item storage-102 weight 0.180
}
rack rack-02 { ## rack 层级




 ##
    id -6 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item storage-103 weight 0.180
```

```
    item storage-104 weight 0.180
}
```

---

②修改root层级，把rack-01和rack-02添加到里面。

```
root default {
    id -1 # do not change unnecessarily
    # weight 0.720
    alg straw
    hash 0 # rjenkins1
    item rack-01 weight 0.360
    item rack-02 weight 0.360
}
```

---

③把默认的Host层级故障域修改为Rack层级。

```
rule replicated_ruleset {
    ruleset 0
    type replicated
    min_size 1
    max_size 10
    step take default
     step chooseleaf firstn 0 type rack ## 就是这里
```

```
 ##
```

```
        step emit
}
```

8）编译crushmap文本文件为二进制文件。

```
 [root@ceph-deploy ceph-deploy]#crushtool -c
/tmp/mycrushmap.txt -o /tmp/mycrushmap.new
```

9）把新的crushmap应用于集群，使之生效。

```
[root@ceph-deploy ceph-deploy]#ceph osd setcrushmap -i
/tmp/mycrushmap.new
```

10）Crush Map应用之后，PG就会重新分布，此时带宽和IO就会增长。为了不影响线上业务，可以对带宽按QoS策略做处理。

 注意

• 在调整之前做好crushmap的备份，以防crush设置不当时能够及时复原。

• crush的设计应该在业务系统上线之前敲定，在线调整将面临极大的风险，必须慎之又慎。

• 副本数为2时，我们创建的pool的max_size应该修改为2，min_size为1。

• 有时候reblance未能达到完全收敛，可能需要设置tunable值为optimal，即使用ceph osd crush tunable optimal调整。

• 设置osd crush update on start为false，防止OSD重启更新crushmap。

拓展：通过命令行在线修改Crush Map

这里我们也讲一下如何通过命令行实现在线修改Crush Map，达到设计的效果。

（1）添加Rack层级：’rack-01’和’rack-02’。

```
[root@ceph-deploy ceph-deploy]#ceph osd crush add-bucket
rack-01 rack
[root@ceph-deploy ceph-deploy]#ceph osd crush add-bucket
rack-01 rack
```

（2）把rack-01和rack-02转移到root下面。

```
[root@ceph-deploy ceph-deploy]#ceph osd crush move rack-01
root=default
[root@ceph-deploy ceph-deploy]#ceph osd crush move rack-02
root=default
```

（3）把Host转移到Rack层级里。

```
[root@ceph-deploy ceph-deploy]#ceph osd crush move storage-
101 rack=rack-01 root=default
[root@ceph-deploy ceph-deploy]#ceph osd crush move storage-
102 rack=rack-01 root=default
[root@ceph-deploy ceph-deploy]#ceph osd crush move storage-
103 rack=rack-02 root=default
[root@ceph-deploy ceph-deploy]#ceph osd crush move storage-
104 rack=rack-02 root=default
```

（4）创建以Rack为隔离域的新rule。

```
[root@ceph-deploy ceph-deploy]#ceph osd crush rule create-
simple newrule default rack firstn
```

（5）为pool（假设存储池名为newpool）指定所选用的newrule（ceph osd crush rule dump可获取newrule的id）。

```
[root@ceph-deploy ceph-deploy]#ceph osd pool set newpool
crush_ruleset 1
####此处
```

1是指在

rule里

rule_id设置的值

####

10.3 CRUSH设计：SSD、SATA混合实例

随着固态硬盘的成本大幅度降低，SSD已经“飞入寻常百姓家”了。SSD的高性能已经让不少应用享受了高IO的福利，Ceph也不例外。下面将对SSD与Ceph结合的几种常见应用场景进行叙述与实战。

10.3.1　场景一：快-慢存储方案

存储节点上既有SATA盘也有SSD盘，把各节点的SSD和SATA分别整合成独立的存储池，为不同的应用供给不同性能的存储。比如常见的云环境中的虚拟机实例，对于实时数据IO性能要求高，并且都是热数据，可以把这部分的存储需求放入SSD的存储池里；而对于备份、快照等冷数据应用，相对IO性能需求比较低，因此将其可以放在普通的由SATA盘组成的存储池里。

1.硬件环境

表10-3是案例所需的硬件环境配置。

表10-3　案例所需的硬件环境配置

存储节点主机名	osd 编号	对应的设备	存储类型
storage-101	osd.0	sdb	SATA
storage-101	osd.1	sdc	SSD
storage-102	osd.2	sdb	SATA
storage-102	osd.3	sdc	SSD
storage-103	osd.4	sdb	SATA
storage-103	osd.5	sdc	SSD
storage-104	osd.6	sdb	SATA
storage-104	osd.7	sdc	SSD
storage-105	osd.8	sdb	SATA

（续）

存储节点主机名	osd 编号	对应的设备	存储类型
storage-105	osd.9	sdc	SSD
storage-106	osd.10	sdc	SATA
storage-106	osd.11	sdc	SSD

依据以上的硬件条件，我们把含有SSD的OSD聚合，并创建一个新的root层级（假如命名为ssd）并保留默认的层级关系，逻辑设计图如图10-13所示。

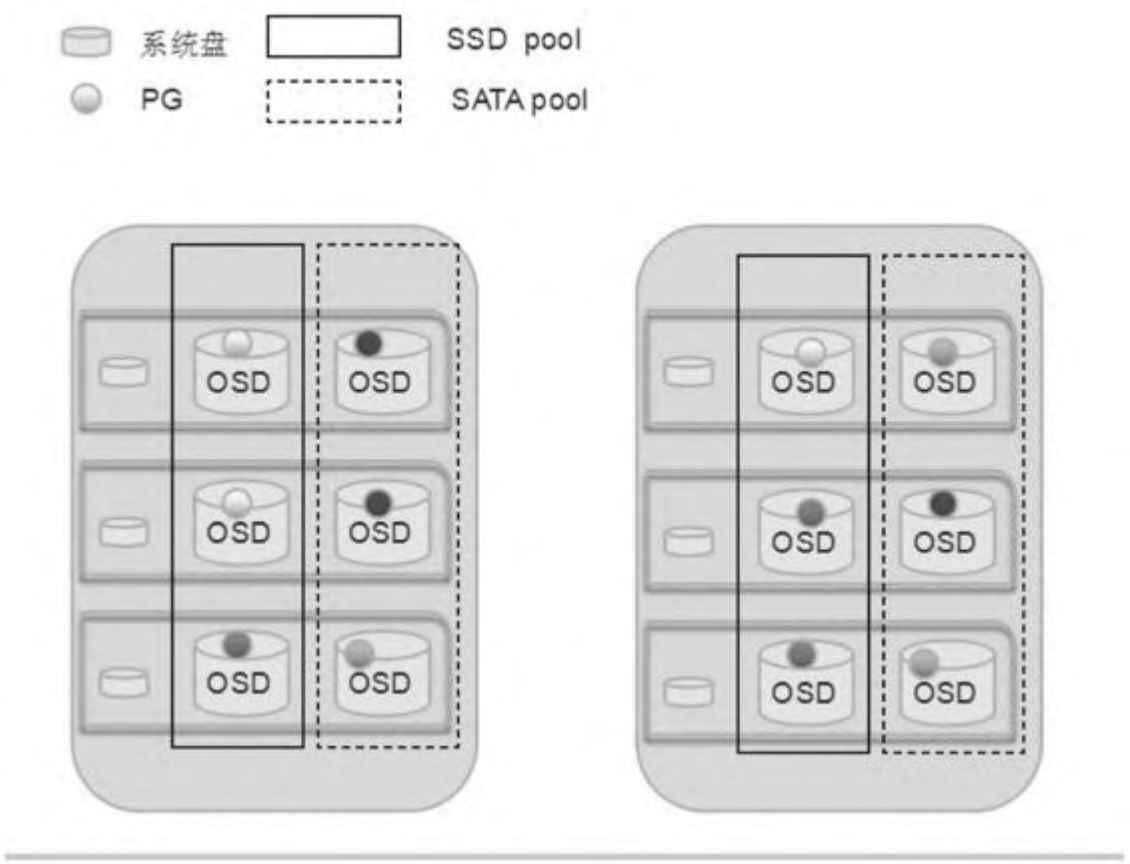

图10-13　快 - 慢存储逻辑架构图

2. 部署

1）利用ceph-deploy工具直接部署、添加OSD。

```
[root@ceph-deploy ceph-deploy]#ceph-deploy osd --zap-disk
create \
storage-101:sdb storage-101:sdc \
storage-102:sdb storage-102:sdc \
storage-103:sdb storage-103:sdc \
storage-104:sdb storage-104:sdc \
storage-105:sdb storage-105:sdc
[ceph_deploy.conf][DEBUG ] found configuration file at:
```

```
/root/.cephdeploy.conf
[ceph_deploy.cli][INFO ] Invoked (1.5.24): /usr/bin/ceph-
deploy osd --zap-disk create storage-102:sdb
[ceph_deploy.osd][DEBUG ] Preparing cluster ceph disks
storage-102:/dev/sdb:
[ceph-deploy][DEBUG ] connected to host: storage-101
[ceph-deploy][DEBUG ] detect platform information from
remote host
[ceph-deploy][DEBUG ] detect machine type
[ceph_deploy.osd][INFO ] Distro info: CentOS Linux 7.1.1503
Core
[ceph_deploy.osd][DEBUG ] Deploying osd to storage-101
....以下忽略!
```

>

2）获取当前的Crush Map。

利用Ceph工具获取当前Crush Map。

```
[root@ceph-deploy ~]#ceph osd getcrushmap -o /tmp/mycrushmap
got crush map from osdmap epoch 20
```

3）反编译Crush Map。

/tmp/mycrushmap是一个二制文件，需要通过crushtool反编译为文本文件。

```
[root@ceph-deploy ceph-deploy] crushtool -d /tmp/mycrushmap
> /tmp/mycrushmap.txt
4）编辑



Crush Map文本文件
```

4）编辑Crush Map文本文件

①设置SSD pool的Bucket入口，新建一个root层级，命名为ssd，并且把SSD设备的OSD移到里面（保留OSD所属的Host层级）。

```
host storage-101 {
    id -9 # do not change unnecessarily ## 设置唯一
```

```
ID ##
    # weight 0.180 ## 权重可以忽略
```

,会依据

osd自动计算

```
##
    alg straw
    hash 0 # rjenkins1
    item osd.1 weight 0.090
}
host storage-102 {
    id -10 # do not change unnecessarily
```

```
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.3 weight 0.090
}
host storage-103 {
    id -11 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.5 weight 0.090
}
host storage-104 {
    id -12 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.7 weight 0.090
}
host storage-105 {
    id -13 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.9 weight 0.090
}
host storage-106 {
    id -14 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.11 weight 0.090
}
root ssd { ## 新建名为
```

ssd 的

root bucket , 作为后续

SSD pool的入口

```
 ##
    id -8 # do not change unnecessarily
    # weight 0.720
    alg straw
    hash 0 # rjenkins1
    item storage-101 weight 0.090
    item storage-102 weight 0.090
    item storage-103 weight 0.090
    item storage-104 weight 0.090
    item storage-105 weight 0.090
}
```

---

②把默认的default root bucket重命名为sata，并把SATA设备的OSD转移到sata里。

```
host storage-101 {
   id -2 # do not change unnecessarily
   # weight 0.180
   alg straw
   hash 0 # rjenkins1
   item osd.0 weight 0.090
}
host storage-102 {
   id -3 # do not change unnecessarily
   # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.2 weight 0.090
}
host storage-103 {
    id -4 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.4 weight 0.090
}
host storage-104 {
    id -5 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.6 weight 0.090
}
host storage-105 {
    id -6 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.8 weight 0.090
}
host storage-106 {
    id -7 # do not change unnecessarily
    # weight 0.180
    alg straw
    hash 0 # rjenkins1
    item osd.10 weight 0.090
}
root sata { ## default重命名为
```

sata,作为

SATA pool的入口

```
##
    id -1 # do not change unnecessarily
    # weight 0.720
    alg straw
    hash 0 # rjenkins1
    item storage-101 weight 0.090
    item storage-102 weight 0.090
    item storage-103 weight 0.090
    item storage-104 weight 0.090
    item storage-105 weight 0.090
}
```

5）设置rule规则。

①为SSD pool添加rule。

```
rule ssd {
   ruleset 1
   type replicated
   min_size 1
   max_size 10
   step take ssd ## 指定入口为
```

```
 ssd bucket ##
   step choose firstn 0 type host
   step emit
}
rule ssd {
   ruleset 1
   type replicated
   min_size 1
   max_size 10
   step take ssd ## 指定入口为
```

```
 ssd bucket ##
   step choose firstn 0 type host
   step emit
}
```

②为SATA pool添加sata rule。

```
rule sata {
    ruleset 0type replicated
    min_size 1
    max_size 10
    step take sata ## 指定入口为
```

```
 sata bucket  ##
    step choose firstn 0 type host
    step emit
}
```

6）编译Crush Map文本文件为二进制文件。

```
[root@ceph-deploy ceph-deploy]#crushtool -c
/tmp/mycrushmap.txt -o /tmp/mycrushmap.new
```

7）把新的Crush Map应用于集群使之生效。

```
[root@ceph-deploy ceph-deploy]#c ceph osd setcrushmap -i
/tmp/mycrushmap.n
```

8）查看crush结构，确认新的Crush Map已经生效。

```
[root@ceph-deploy ceph-deploy]# ceph osd tree
ID WEIGHT TYPE NAME UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 0.54000   root sata
-2 0.18000          host storage-101
0 0.09000                 osd.0 up 1.00000 1.00000
-3 0.18000          host storage-102
2 0.09000                 osd.2 up 1.00000 1.00000
-4 0.18000          host storage-103
4 0.09000                 osd.4 up 1.00000 1.00000
-5 0.18000          host storage-104
6 0.09000                 osd.6 up 1.00000 1.00000
-6 0.18000          host storage-105
8 0.09000                 osd.8 up 1.00000 1.00000
-7 0.18000          host storage-106
10 0.09000                osd.10 up 1.00000 1.00000
-8 0.54000     root ssd
-9 0.09000          host storage-101
1 0.09000                 osd.1 up 1.00000 1.00000
-10 0.09000         host storage-102
3 0.09000                 osd.3 up 1.00000 1.00000
-11 0.09000         host storage-103
5 0.09000                 osd.5 up 1.00000 1.00000
-12 0.09000         host storage-104
7 0.09000                 osd.7 up 1.00000 1.00000
-13 0.09000         host storage-105
9 0.09000                 osd.9 up 1.00000 1.00000
-14 0.08000         host storage-106
11 0.09000                osd.11 up 1.00000 1.00000
```

9）创建SSD和SATA存储池。

```
[root@ceph-deploy ceph-deploy]# ceph osd pool create SSD 128
128
 pool 'SSD' created
[root@ceph-deploy ceph-deploy]# ceph osd pool create SATA
128 128
 pool 'SATA' created
```

10）为存储池指定合适的rule。

```
[root@ceph-deploy ceph-deploy]#ceph osd pool set SSD
crush_ruleset  1
[root@ceph-deploy ceph-deploy]#ceph osd pool set SATA
crush_ruleset 0
```

11）查看rule是否生效。

```
[root@ceph-deploy ceph-deploy]# ceph osd dump | grep -Ei
"ssd|sata"
pool 1 'SSD' replicated size 3 min_size 2 crush_ruleset 1
object_hash rjenkins pg_num 128 pgp_num 128 last_change 20
flags hashpspool stripe_width 0 ##Pool SSD的
```

```
rule id 为
```

```
1 ##
pool 2 'SATA' replicated size 3 min_size 2 crush_ruleset 0
object_hash rjenkins pg_num 128 pgp_num 128 last_change 22
flags hashpspool stripe_width 0  ##Pool SATA的
```

rule id 为

0 ##

12）查看PG分布是否正确。

```
[root@ceph-deploy ceph-deploy ]# ceph pg dump | grep '^1\.'
| awk 'BEGIN{print "PG_id","\t","copy_set"}{print
$1,"\t",$15}' | less
dumped all in format plain
PG_id copy_set
1.7e [1,7,9] ## SSD pool的
```

PG都分配到了

ssd rule所指向的

```
OSD ##
1.7f [1,7,11]
1.7c [5,9,11]
1.7d [1,3,11]
1.7a [5,7,9]
1.7b [11,7,5]
1.78 [1,3,11]
1.79 [5,9,11]
......
```

SATA pool的验证也是一样，在此不再展示。

以上是SSD和SATA设备组建存储池的流程，结合云环境（比如说OpenStack），可以把SSD pool给虚拟机实例使用，而SATA pool分拨给冷数据（如备份、快照等）。

10.3.2 场景二：主-备存储方案

设备配置同场景一，也是SSD和SATA设备混合，但不同于场景一组建独立的存储池分别为不同的应用提供存储服务，而是依据Ceph读写流程（如图10-14所示），把主副本放在SSD组成的Bucket里，其他副本放置在SATA设备上。如此，即可以在性能上得到一定的提升，也可以充分利用现有设备。

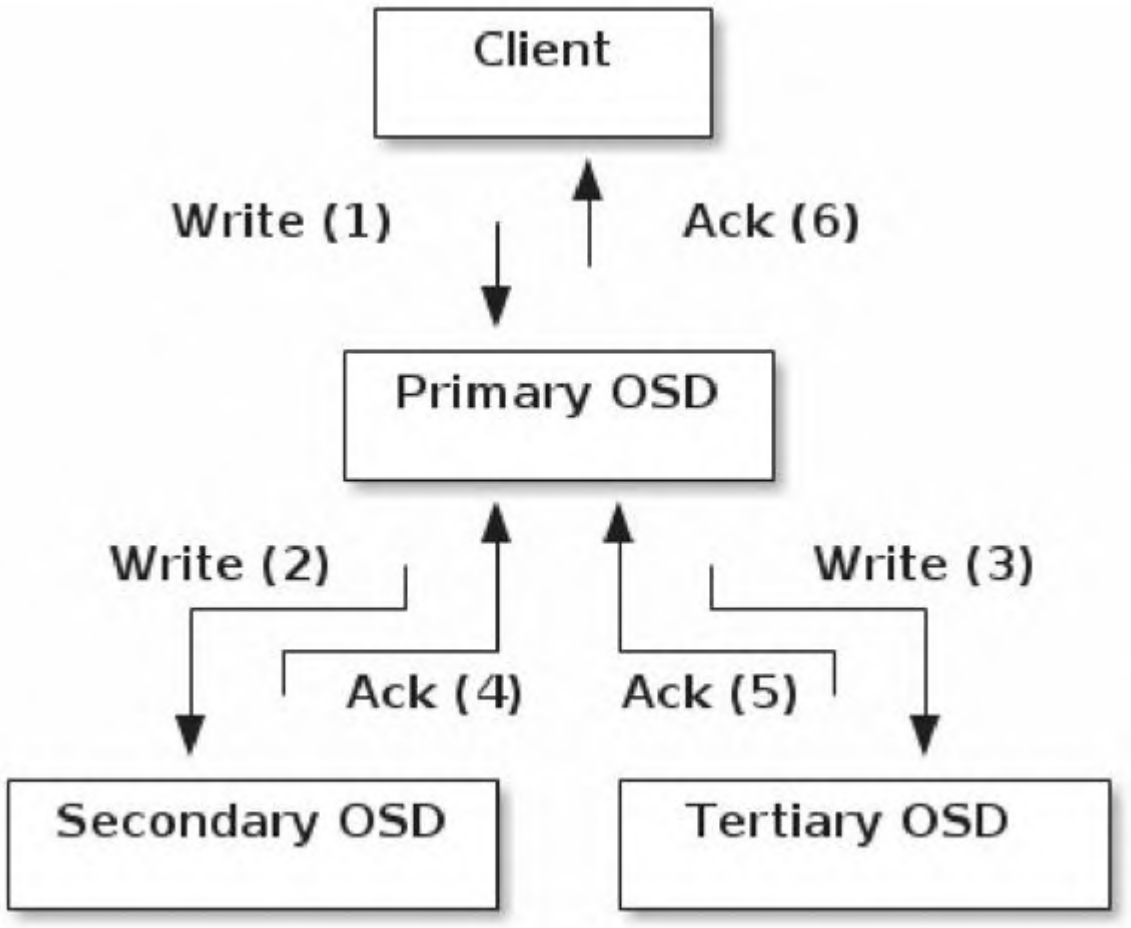

图10-14　Ceph读写流程

主－备存储方案的CRUSH设计跟场景一（快-慢存储方案）一样如图10-15所示，只是在rule上面做些修改即可。

在此，我们直接沿用场景一中的CRUSH，创建新的crush rule。

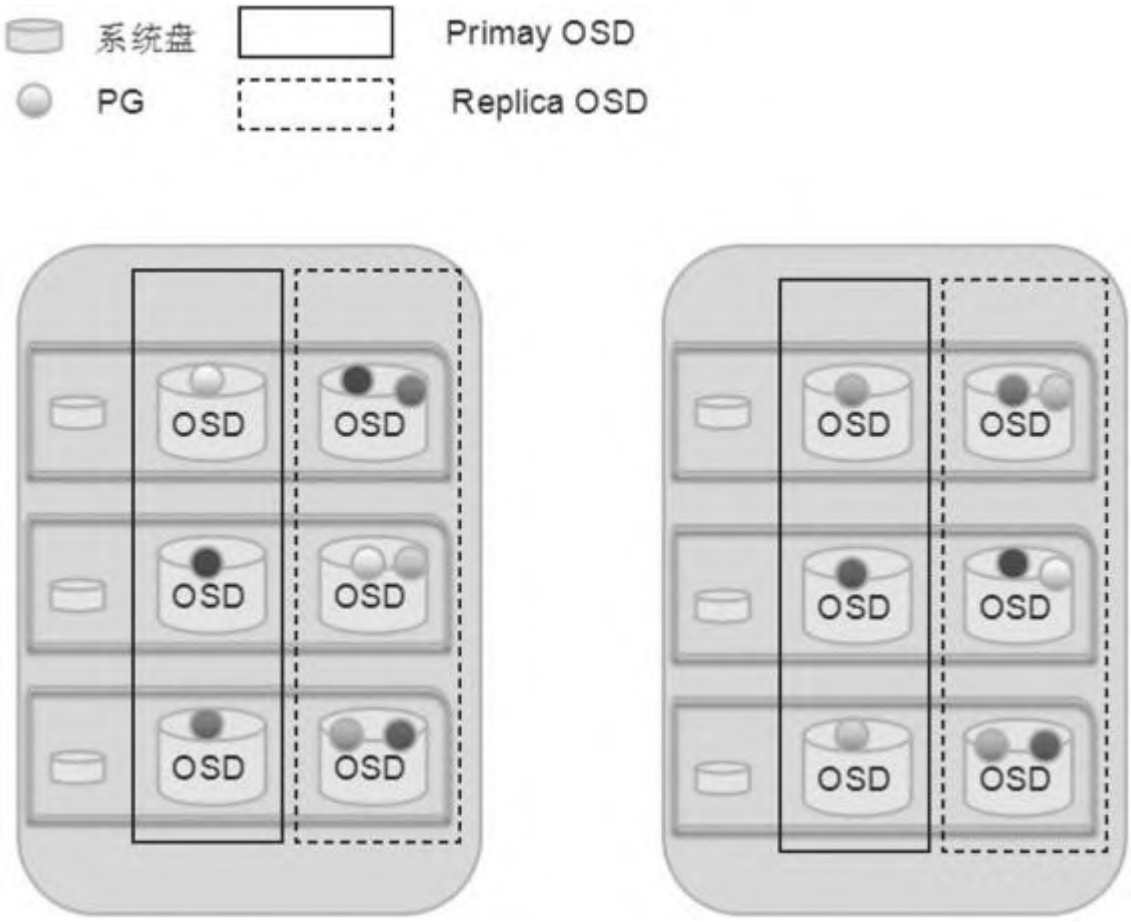

图10-15　主-备存储方案逻辑图

```
rule pg {
   ruleset 3
   type replicated
   min_size 1
   max_size 10
   step take ssd ## 指定入口为
```

```
 ssd bucket
   step choose firstn 1 type host ## 从
```

```
ssd bucket 搜索选一个合适的




 osd 存储主副本




##
   step emit
   step take sata ## 指定入口为




 sata bucket
   step choose firstn -1 type host ## 从
```

```
sata bucket 搜索其他副本所需要的




 osd 来存储，隔离域依然为




 host ##
   step emit
}
```

编译、应用新的Crush Map到集群里，并创建一个名为pg的存储池，且指定pg rule（步骤详见场景一），验证pg pool的PG分布。

```
 [root@ceph-deploy ceph-deploy ]# ceph pg dump | grep '^3\.'
| awk 'BEGIN{print "PG_id","\t","copy_set"}{print
$1,"\t",$15}' | less
dumped all in format plain
PG_id copy_set
3.2e [1,2,8] ## 主副都落在
```

[1,3,5,7,9,11]的

SSD的

OSD上，其他副本都在

SATA设备上

##
3.2f [1,4,6]
3.2c [3,8,10]
3.2d [3,4,8]

```
3.2a [5,6,8]
3.2b [11,2,4]
3.28 [7,8,10]
3.29 [5,4,8]
......
```

拓展：通过OSD亲和性实现主备方案

在Ceph集群环境里，并不是所有的硬盘设备性能和容量是完全一致的（如我们场景中的环境）。在高版本Ceph中（0.80以后），可以通过调整OSD的primary affinity（osd的亲和性）的值（0~1之间），实现减小OSD设备承载客户端读写的负载。（从场景一中图10-13所示，客户端的读写主要发生在primay osd上）。亲和性的调整不会引起数据的变动。

因此，实现主备存储方案便有更简便的方法，即利用osd的primary affinity值，只要把SATA设备对应OSD的primary affinity设置为0，那这些osd就不会成为主副本。读写都只落到SSD设备对应的OSD上。

1）调整primary affinity的值，首先要打开mon osd allow primary affinity的开关，默认是关闭的。

```
[root@storage-101 ~]# ceph --admin-daemon
/var/run/ceph/ceph-mon.*.asok config show|grep
'primary_affinity'"mon_osd_allow_primary_affinity":"false",#
#默认值




##
```

2）通过ceph tell实现参数在线调整。

```
[root@ceph-deploy ceph-deploy]# ceph tell mon.* injectargs \
"--mon_osd_allow_primary_affinity=1"
mon.storage-101: injectargs:mon_osd_allow_primary_affinity =
'true' mon.storage-102:
injectargs:mon_osd_allow_primary_affinity = 'true'
mon.storage-103: injectargs:mon_osd_allow_primary_affinity =
'true'
```

3）在ceph.conf的[mon]作用域添加配置。

```
mon osd allow primary affinity = true
```

4）把SATA设备对应的osd的primary affinity调整为0。

```
[root@ceph-deploy ceph-deploy ]# for i in 0 2 4 6 8 10;do
ceph osd primary-affinity osd.$i 0 ;done
set osd.0 primary-affinity to 0 (802)
set osd.2 primary-affinity to 0 (802)
set osd.4 primary-affinity to 0 (802)
set osd.6 primary-affinity to 0 (802)
set osd.8 primary-affinity to 0 (802)
set osd.10 primary-affinity to 0 (802)
```

5）验证SATA设备的osd是否还在担任primary osd。

```
[root@ceph-deploy ceph-deploy ]# for i in 0 2 4 6 8 10;do
ceph pg dump | grep active+clean | egrep "\[$i," | wc -l
;done
dumped all in format plain
0
dumped all in format plain
0
dumped all in format plain
0
dumped all in format plain
0
dumped all in format plain
0
dumped all in format plain
0
```

从检验结果看，SATA设备的osd已经没有再担任primay osd，设置有效。

10.4　模拟测试CRUSH分布

Ceph通过Crush实现数据的伪随机分布。在Ceph里，一旦你的Crush Map创建成功（无变更），你创建的所有Object对应于OSD的映射关系就已经确认了。这就是作者所理解的伪随机分布，先决条件已经确认，可以推算数据的具体分落。在现实环境中，很少有大规模的Ceph集群环境（100个节点以上）提供测试Crush Map。读者可以通过crushtool这个利器实现模拟测试CRUSH分布。方法如下。

1）通过crushrool创建crushmap文件。

```
[root@ceph-deploy ceph-deploy]#crushtool --outfn  crushmap --build --num_osds 10     host straw 2 rack straw 2 default straw 0
2015-10-09 15:00:15.194369 7f20e30fc780  1
ID  WEIGHT  TYPE NAME
-9  10.00000    default default
-6  4.00000     rack rack0
-1  2.00000         host host0
0   1.00000             osd.0
1   1.00000             osd.1
```

```
-2  2.00000         host host1
2   1.00000             osd.2
3   1.00000             osd.3
-7  4.00000     rack rack1
-3  2.00000         host host2
4   1.00000             osd.4
5   1.00000             osd.5
-4  2.00000         host host3
6   1.00000             osd.6
7   1.00000             osd.7
-8  2.00000     rack rack2
-5  2.00000         host host4
8   1.00000             osd.8
9   1.00000             osd.9
```

其中：

• --outfn crushmap表示导出的map的文件名是crushmap;

• --build表示创建一个crushmap;

• --num_osds表示此map包含10个osd，host straw 2表示每个host里包含两个osd，rack straw 2表示每个rack里包含两个host，default straw 0表示所有的rack都包含在一个root里。

注意

crush的层级可以自定义添加，比如在Rack层级上可以添加dc级，整个结构为--num_osds 10 host straw 2 rack straw 2 dc straw 1 default straw 0，生成的crush map如下所示。

```
ID  WEIGHT  TYPE NAME
-12 10.00000    default default
-9  4.00000     dc dc0
-6  4.00000         rack rack0
-1  2.00000             host host0
0   1.00000                 osd.0
1   1.00000                 osd.1
-2  2.00000             host host1
2   1.00000                 osd.2
3   1.00000                 osd.3
-10 4.00000     dc dc1
-7  4.00000         rack rack1
-3  2.00000             host host2
4   1.00000                 osd.4
5   1.00000                 osd.5
-4  2.00000             host host3
6   1.00000                 osd.6
7   1.00000                 osd.7
-11 2.00000     dc dc2
-8  2.00000         rack rack2
-5  2.00000             host host4
8   1.00000                 osd.8
9   1.00000                 osd.9
```

2）创建之后会在本地目录生成一个crushmap的二进制文件，我们可以通过crushtool工具进行反编译。

```
[root@localhost ceph ]# crushtool  -d  crushmap  -o map.txt
[root@localhost ceph ]# ll
total 8
-rw-r--r-- 1 root root  833 Oct  9 15:13 crushmap
-rw-r--r-- 1 root root 2351 Oct  9 15:20 map.txt  ##可编辑文件
```

```
##
```

3）打开本地的map.txt，我们会发现crush type部分只有我们创建的那几个层级（如下所示），不再是默认的Crush Map的10个层级。

```
# begin crush map
tunable choose_local_tries 0
tunable choose_local_fallback_tries 0
tunable choose_total_tries 50
tunable chooseleaf_descend_once 1
tunable straw_calc_version 1
# devices
device 0 device0
device 1 device1
device 2 device2
```

```
device 3 device3
device 4 device4
device 5 device5
device 6 device6
device 7 device7
device 8 device8
device 9 device9
# types
type 0 device
type 1 host
type 2 rack
type 3 dc
type 4 default     ## bucket层级只有指定创建的几个类型









 ##
# buckets
.....
```

4）为map.txt添加自定义的crush rule。

```
rule  custom {
        ruleset 1
        type replicated
        min_size 1
        max_size 10
        step take default
        step choose firstn 1 type dc
        step chooseleaf  firstn 0 type host
        step emit
}
```

rule custom简要说明。

• 在root层级里选择一个dc层级作为后续隔离域的基础。

• 在选中的dc层级里，以host为隔离域，选择osd。

5）编辑好需要的crushmap之后，再次通过crushtool进行编译。

```
[root@localhost]# crushtool  -c map.txt  -o map.bin
[root@localhost]# ll
total 12
-rw-r--r-- 1 root root  833 Oct  9 15:13 crushmap
-rw-r--r-- 1 root root 1071 Oct  9 15:36 map.bin    ##新的
```

crush map文件

```
 ##
-rw-r--r-- 1 root root 2575 Oct  9 15:31 map.txt
```

6）测试Crush Map。

```
[root@loclahost ]# crushtool -i map.bin  --test --show-
statistics --rule 1 --min-x 1 --max-x 5 --num-rep 2  --show-
mappings
rule 1 (custom), x = 1..5, numrep = 2..2
CRUSH rule 1 x 1 [9]  ##映射关系
```

,即

Object 1 映射到了

```
 OSD 9 ##
CRUSH rule 1 x 2 [7,4] ## Object 2 映射到了
```

OSD 7、

```
4 ##
CRUSH rule 1 x 3 [7,4]
CRUSH rule 1 x 4 [2,0]
CRUSH rule 1 x 5 [7,4]
rule 1 (custom) num_rep 2 result size == 1: 1/5
rule 1 (custom) num_rep 2 result size == 2: 4/5
```

其中：

- --test表示调用crushtool里的测试功能。
- --show-statistics表示显示统计结果。
- --rule 1表示使用rule 1，即我们自己创建的rule custom。

• --min-x 1--max-x 5表示创建的object的数量（如果不指定，表示创建1024）。

• --num-rep 2表示创建的副本数为2。

• --show-mappings表示显示具体的映射关系。

```
rule 1 (custom) num_rep 2 result size == 1: 1/5
rule 1 (custom) num_rep 2 result size == 2: 4/5
```

• 前者表示1/5的object成功映射到了一个osd上（即表示映射有不成功）。

• 后者表示4/5的object成功映射到了两个osd上（即表示映射成功）。

从前文的Cursh Map可知，当crush选择dc层级的dc2时，由于在此层级下只有一个Host，且隔离域刚好为Host，所以无法再映射第二副本，如图10-16所示。

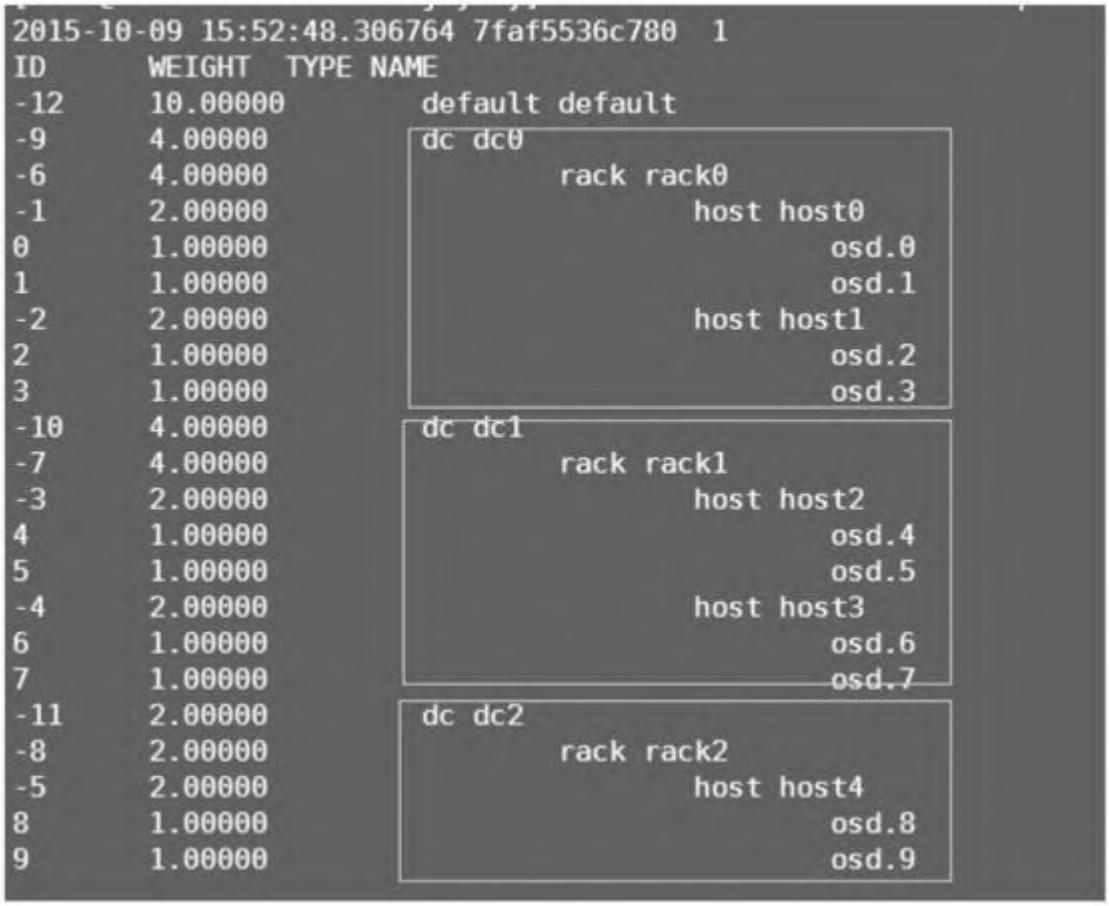

图10-16　crush map结构图

此外，还可以添加--tree显示crushmap的树状结构，--show-choose-tries显示映射时retry的次数。这个值可以在tunables部分里找到。可以通过--set-choose-total-tries强制指定。

```
[root@localhost]# crushtool -i map.bin  --test --show-
statistics --rule 1 --min-x 1 --max-x 5 --num-rep 2  --show-
mappings --set-choose-total-tries  1 -o newmap   ##强制指只制尝
```

试一次

```
 ##
rule 1 (custom), x = 1..5, numrep = 2..2
CRUSH rule 1 x 1 [9]  ##映射失败
```

```
 ##
CRUSH rule 1 x 2 [7]  ##映射失败
```

```
 ##
CRUSH rule 1 x 3 [7,4]
CRUSH rule 1 x 4 [2,0]
CRUSH rule 1 x 5 [7,4]
rule 1 (custom) num_rep 2 result size == 1: 2/5
rule 1 (custom) num_rep 2 result size == 2: 3/5
```

显然这个值会影响CRUSH的分布效果，默认值为50。

--show-utilization可以显示OSD的实际的object映射数以及目标映射值。

```
device 0:       stored : 1  expected : 0.5
device 2:       stored : 1  expected : 0.5
device 4:       stored : 2  expected : 0.5
device 7:       stored : 3  expected : 0.5
device 9:       stored : 1  expected : 0.5
```

 注意

更多的设置可以通过man crushtool获取。

10.5 本章小结

通过本章学习，读者可以从定性和定量两方面了解到Ceph的稳定性与可靠性。本章中列举了常用的CRUSH使用场景，通过灵活设计CRUSH方案，实现“指哪存哪”的目标，真正做到了软件定义存储。大家在充分理解本章内容的基础上，就可以根据实际的场景，设计出适合自己的CRUSH。

第11章　缓冲池与纠删码

11.1　缓冲池原理

缓冲（Cache，为便于读者理解，本文直接使用Cache）技术是为了协调吞吐速度相差较大的设备之间数据传送而采用的技术。

传统上，Cache为了缓和CPU和IO设备速度不匹配的矛盾，提高CPU和IO设备的并行性而存在的，在现代操作系统中，几乎所有的IO设备在与CPU交换数据时都用了Cache。所以，在计算机组成里，每个部件的设计都可以看到Cache的影子。

例如，CPU的L1 Cache、L2 Cache和L3 Cache，甚至还有L4 Cache（IBM Power8）。文件系统会设计页

面缓存（Page Cache），而硬盘内部也会设计易失性缓存，一般容量为16/32/64MB。

既然提及Cache，也需要说明一下缓冲（英文Buffer，同理，本文直接使用Buffer）。为了让大家区分Cache和Buffer概念，我们仔细分析一下。

Cache的设计原理是把读取过的数据保存起来，如果在以后的操作中重新读取时则命中（找到需要的数据），就不要去读较慢的设备，若没有命中就读较慢的设备。Cache的目标是把频繁读取的数据进行组织，并把这些内容放在最容易找到的位置，而把不再读的内容不断往后排，直至从Cache中删除。实际上，读者会想到，Cache就是优先存放“热数据”，其性能指标是“命中率”。

Buffer的设计原理是把分散的写数据内容集中进行，按照一定周期写到目标设备。典型的应用场景是

磁盘写，通过Buffer机制，写数据时候减少磁盘碎片和硬盘的反复寻道过程，从而提高系统的性能。

我们可以暂且认为Cache是针对读优化，而Buffer是针对写优化。

在本地计算机结构组成里，相比CPU和内存之间的IO差距，内存和机械硬盘的IO差距较为明显。而固态硬盘的出现，无论IOPS和带宽都胜过机械硬盘，同时内存和固态硬盘的IO差距较机械硬盘小。比较内存、固态硬盘和机械硬盘的每GB单位成本，得出的结果是内存>固态硬盘>机械硬盘。在这个成本和技术的前提下，有不少工程师提出：能否通过固态硬盘作为机械式硬盘Cache/Buffer实现数据缓存策略呢？

目前有较多的开源和商业存储领域软件和硬件提供这种分级缓存/缓冲机制。例如，在软件方面有ZFS、LVM、Bcache、Flashcache、Enhanceio和Intel

CAS；在硬件方面，Intel公司和LSI推出新形态的RAID阵列卡，这类阵列卡配备板载的SSD作为加速，外部通过minSAS连接外部硬盘。

上面介绍了缓冲技术的基础知识及背景，下面我们来介绍Ceph中缓冲技术的基本实现原理。

经过前面章节的介绍，大家了解到数据都是以object形式存放在Ceph中，这些object数据都是存储在资源池中（即pool），而组成pool的基本单位是OSD，因此充当OSD的存储介质的性能决定了整个资源池的最终性能。存储介质性能的差异，造成了资源池性能的最终差异性，因此使用SSD固态硬盘这一类的高速且价格昂贵的存储介质组成高性能资源池，使用SATA机械硬盘这一类低速廉价的介质组成低速高容量资源池，Ceph为此提出了缓冲池技术，比较完美地平衡了性能和成本的关系。要实现缓冲池技术，首先需

要将整个存储系统按照存储介质的性能分为前端高速缓冲池和后端低速存储池，使用特定算法将数据写入和读取流程进行适当的调整，将访问次数比较频繁的热数据缓存储在前端高速缓冲池中（对应热数据上浮），而访问量比较低的冷数据则存储在后端低速存储池中（冷数据下沉），从而实现提高读写性能的目的，如图11-1所示。

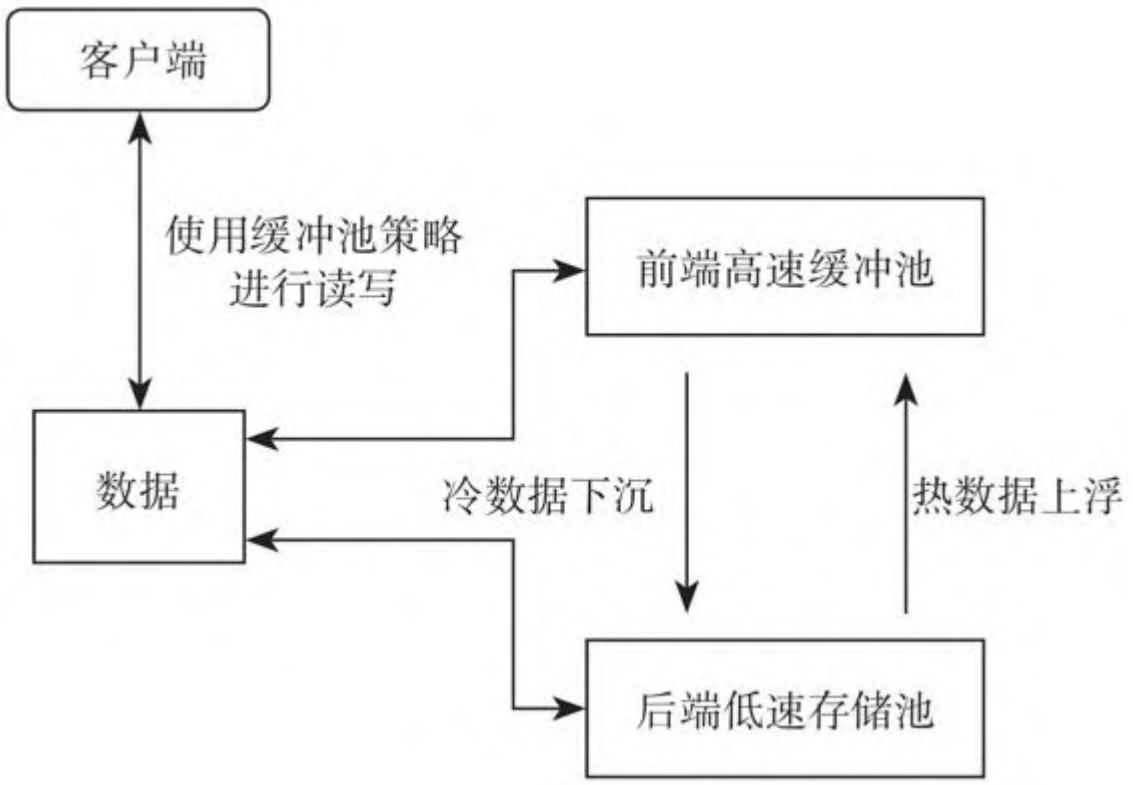

图11-1　缓冲池原理

介绍了基本的缓冲资源池原理之后，还需要向读者介绍Ceph中提供的两种不同类型的缓冲资源池策略模式。

1）写回模式：　当用户配置了写回模式，所有来自客户端的写入请求都会在抵达前端高速缓冲池后收到ACK确认，之后由高速缓冲池按照一定的策略将数据写入后端低速存储池，完成数据的最终写入。在这个模式下，前端高速缓冲池充当整个写入操作的前端缓存。当客户端需要读取数据时，读取请求先到达高速缓冲池，之后高速缓冲池从低速存储池中加载相应数据，加载完毕再由高速缓冲池返回客户端所需的数据。当客户端请求相同的数据，则直接由高速缓冲池返回之前已经加载的数据，如果数据长时间没有被访问，则会从高速缓存池清除。这个模式比较适合数据经常发生变更，又需要经常被访问的情况。

2）读取转发模式： 这个模式下读写操作都在前端高速缓冲池进行，如果客户端需要读取的数据不在高速缓冲池中，则读取请求会被转发到后端。这个模式一般用于高速缓冲池容量已经被耗尽，而又无法做扩容的情况。

11.2　缓冲池部署

自从0.80版本开始，Ceph加入缓冲池技术。在当前固态硬盘每IOPS性价比占优，而机械硬盘每GB容量更具性价比，根据业务系统数据读写的局部性，可以通过缓冲池技术把SSD存储池作为HDD存储池的前端高速缓冲池。

11.2.1　缓冲池的建立与管理

假设我们通过ceph-deploy工具部署一个标准的Ceph集群，默认所有OSD在CRUSH里都是Default Rule。本次实验，我们需要自定义Rule，对Pool指定使用某个Rule，即Pool上承载的数据均落到该Rule的硬盘上。因此，我们需要定义SATA Rule和SSD Rule。

定义SATA Rule，可以使用当前已经存在的OSD。而定义SSD Rule之前，需要为每个OSD服务器节点添加SSD。按照之前添加OSD的方法，把“SSD”OSD添加到Default Rule。然后基于“SSD”OSD创建SSD Rule，然后根据这个Rule创建SSD Pool。这样就具备SATA Pool和SSD Pool。实验只是逻辑示意，若考虑性能请选取SSD构建SSD Pool。

下面开始配置以ssd-pool作为sata-pool的前端高速缓冲池。

1）新建缓冲池，其中，ssd-pool作为sata-pool的前端高速缓冲池。

```
ceph osd tier add sata-pool ssd-pool
```

2）设定缓冲池读写策略为写回模式。

```
ceph osd tier cache-mode ssd-pool writeback
```

3）将客户端访问从sata-pool切换到ssd-pool。

```
ceph osd tier set-overlay sata-pool ssd-pool
```

11.2.2　缓冲池的参数配置

关于缓冲池的参数配置，遵循以下格式：

```
ceph osd pool set {cachepool} {key} {value}
```

（1）使用布隆过滤器以快速查找目标数据

Ceph在生产环境中会使用布隆过滤器（Bloom-Filter，1970年由Bloom中提出。它可以用于检索一个元素是否在一个集合中）实现在缓冲池中快速查找目标数据。

注意

Bloom Filter是由Bloom在1970年提出的一种多Hash函数映射的快速查找算法。通常应用在一些需要

快速判断某个元素是否属于集合，但是并不严格要求100%正确的场合。

设置缓冲池使用布隆过滤器[1]。

```
ceph osd pool set ssd-pool hit_set_type bloom
```

默认情况下缓冲池基于数据的修改时间来进行确定是否命中缓存，也可以设定热度数hit_set_count和热度周期hit_set_period，以及最大缓冲数据target_max_bytes。

```
ceph osd pool set ssd-pool hit_set_count 1
ceph osd pool set ssd-pool hit_set_period 3600
ceph osd pool set ssd-pool target_max_bytes 1073741824
```

（2）缓冲池容量的控制

在讲解缓冲池大小的问题之前，先来看看缓冲池的代理层的两大主要操作。

• 刷写 （flushing）：负责把已经被修改的对象写入到后端慢存储，但是对象依然在缓冲池。

• 驱逐 （evicting）：负责在缓冲池里销毁那些没有被修改的对象。

缓冲池代理层进行刷写和驱逐的操作，主要和缓冲池本身的容量有关。在缓冲池里，如果被修改的数据达到一个阈值（容量百分比），缓冲池代理就开始把这些数据刷写到后端慢存储。例如，当缓冲池里被修改的数据达到40%时，则触发刷写动作。

```
ceph osd pool set ssd-pool cache_target_dirty_ratio 0.4
```

当被修改的数据达到一个确定的阈值（容量百分比），刷写动作将会以高速运作。例如，当缓冲池里被修改数据达到60%时候，则高速刷写。

```
ceph osd pool set ssd-pool cache_target_dirty_high_ratio 0.6
```

当缓冲区的使用率达到一个确定的阈值（容量百分比），缓冲池的代理将会触发驱逐操作，目的是释放缓冲区空间。例如，当缓冲池里的容量使用达到80%时候，则触发驱逐操作。

```
ceph osd pool set ssd-pool cache_target_full_ratio 0.8
```

除了上面提及基于缓冲池的百分比来判断是否触发刷写和驱逐，还可以指定确定的数据对象数量或者确定的数据容量。

例如，对缓冲池设定最大的数据容量，来强制触发刷写和驱逐操作。

```
ceph osd pool set ssd-pool target_max_bytes 1073741824
```

上述数值单位为字节，1073741824B=1 GB。

同时，也可以对缓冲池设定最大的对象数量。在默认情况下，RBD的默认对象大小为4MB，1GB容量包含256个4MB的对象，则可以设定：

```
ceph osd pool set ssd-pool target_max_objects 256
```

根据前面章节内容，在RBD创建镜像时候，可以指定每个对象的大小。而在文件存储和对象存储，每个对象的大小也不一样，所以target_max_objects的数值需要根据业务场景进行设置。

注意，如果target_max_bytes和target_max_objects都设定了，会以“最近达到阈值”的机制触发刷写和驱逐动作。

（3）缓冲池的数据刷新问题

在缓冲池里，对象有最短的刷写周期。若被修改的对象在缓冲池里超过最短周期，将会被刷写到慢存

储池。例如，设定最短刷写周期为10分钟。

```
ceph osd pool set ssd-pool cache_min_flush_age 600
```

同时，也可以设定对象最短的驱逐周期。例如，设定最短驱逐周期为30分钟。

```
ceph osd pool set ssd-pool cache_min_evict_age 1800
```

[1] 关于布隆过滤器可参考：http://blog.csdn.net/hguisu/article/details/7866173 和 https://en.wikipedia.org/wiki/Bloom_filter。

11.2.3　缓冲池的关闭

1）删除缓冲池。删除一个只读缓冲池。因为只读缓冲池没有包含修改的数据内容，所以可以直接关闭并移除。

2）改变缓冲池读写模式为none，即关闭缓冲池。

```
ceph osd tier cache-mode ssd-pool none
```

3）移除缓冲池。

```
ceph osd tier remove sata-pool ssd-pool
```

删除一个读写缓冲池。因为读写缓冲池包含修改的数据内容，为了不丢失数据，应进行如下操作。

1）把缓冲池读写模式变为forward，目的是让修改过的数据刷写到慢存储。

```
ceph osd tier cache-mode ssd-pool forward
```

2）等待数分钟，确保ssd-pool的修改数据刷写到慢存储。

```
rados -p ssd-pool ls
```

3）如果ssd-pool仍然存在数据，可以强制刷写和驱逐。

```
rados -p ssd-pool cache-flush-evict-all
```

4）卸载慢存储的缓冲池。

```
ceph osd tier remove-overlay sata-pool
```

5）删除慢存储的缓冲池。

```
ceph osd tier remove sata-pool ssd-pool
```

至此，完成整个缓冲池的删除操作，读者根据实际情况选择是否删除对应的ssd-pool。

本节介绍了缓冲池技术的基本原理和基础操作，需要注意的是：目前缓冲池技术在一些应用场景下还存在性能和稳定性问题，特别是Ceph官方强烈建议不要在RBD块存储一类对读写操作频繁且性能要求也比较高的场景下使用缓冲池技术，只建议在RGW对象存储之类场景下使用该技术。

11.3 纠删码原理

纠删码（Erasure Coding，EC）是一种编码容错技术，最早是在通信行业解决部分数据在传输中的损耗问题。其基本原理就是把传输的信号分段，加入一定的校验再让各段间发生相互关联，即使在传输过程中丢失部分信号，接收端仍然能通过算法将完整的信息计算出来。在数据存储中，纠删码将数据分割成片段，把冗余数据块扩展和编码，并将其存储在不同的位置，比如磁盘、存储节点或者其他地理位置。

如果需要严格区分，实际上按照误码控制的不同功能，可分为检错、纠错和纠删3种类型。

- 检错码仅具备识别错码功能而无纠正错码功能。

• 纠错码不仅具备识别错码功能，同时具备纠正错码功能。

• 纠删码则不仅具备识别错码和纠正错码的功能，而且当错码超过纠正范围时，还可把无法纠错的信息删除。

从纠删码基本的形态看，它是k个数据块+m个校验块的结构，其中k和m值可以按照一定的规则设定，可以用公式：n=k+m来表示。变量k代表原始数据或符号的值。变量m代表故障后添加的提供保护的额外或冗余符号的值。变量n代表纠删码过程后创建的符号的总值。当小于m个存储块（数据块或校验块）损坏的情况下，整体数据块可以通过计算剩余存储块上的数据得到，整体数据不会丢失。

下面以k=2，m=1为例，为大家介绍一下如何以纠删码的形式将一个名称为cat.jpg的对象存放在Ceph

中，假定该对象的内容为ABCDEFGH。客户端在将cat.jpg上传到Ceph以后，会在主OSD中调用相应的纠删码算法对数据进行编码计算：将原来的ABCDEFGH拆分成两个分片，对应图11-2中的条带分片1（内容为ABCD）和条带分片2（内容为EFGH），之后再计算出另外一个校验条带分片3（内容为WXYZ）。按照crushmap所指定的规则，将这3个分片随机分布在3个不同的OSD上面，完成对这个对象的存储操作。整个存储过程如图11-2所示。

下面再看一下如何使用纠删码读取数据，同样还是以cat.jpg为例。客户端在发起读取cat.jpg请求以后，这个对象所在PG的主OSD会向其他关联的OSD发起读取请求，比如主OSD是图中的OSD1，当请求发送到了OSD2和OSD3，此时刚好OSD2出现故障无法回应请求，导致最终只能获取到OSD1（内容为ABCD）和OSD3（WXYZ）的条带分片，此时OSD1作为主OSD会对

OSD1和OSD3的数据分片做纠删码解码操作，计算出OSD2上面的分片内容（即EFGH），之后重新组合出新的cat.jpg内容（ABCDEFGH），最终将该结果返回给客户端。整个过程如图11-3所示。

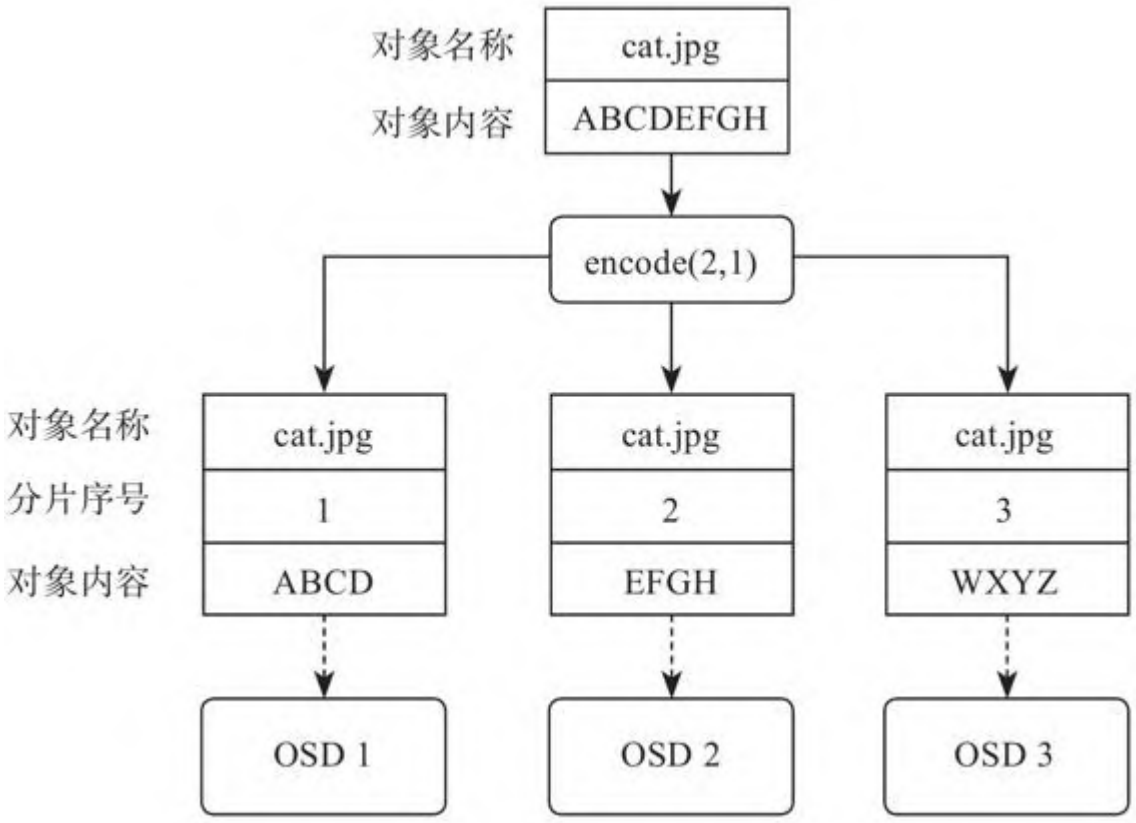

图11-2　在ceph中利用纠删码写入对象数据

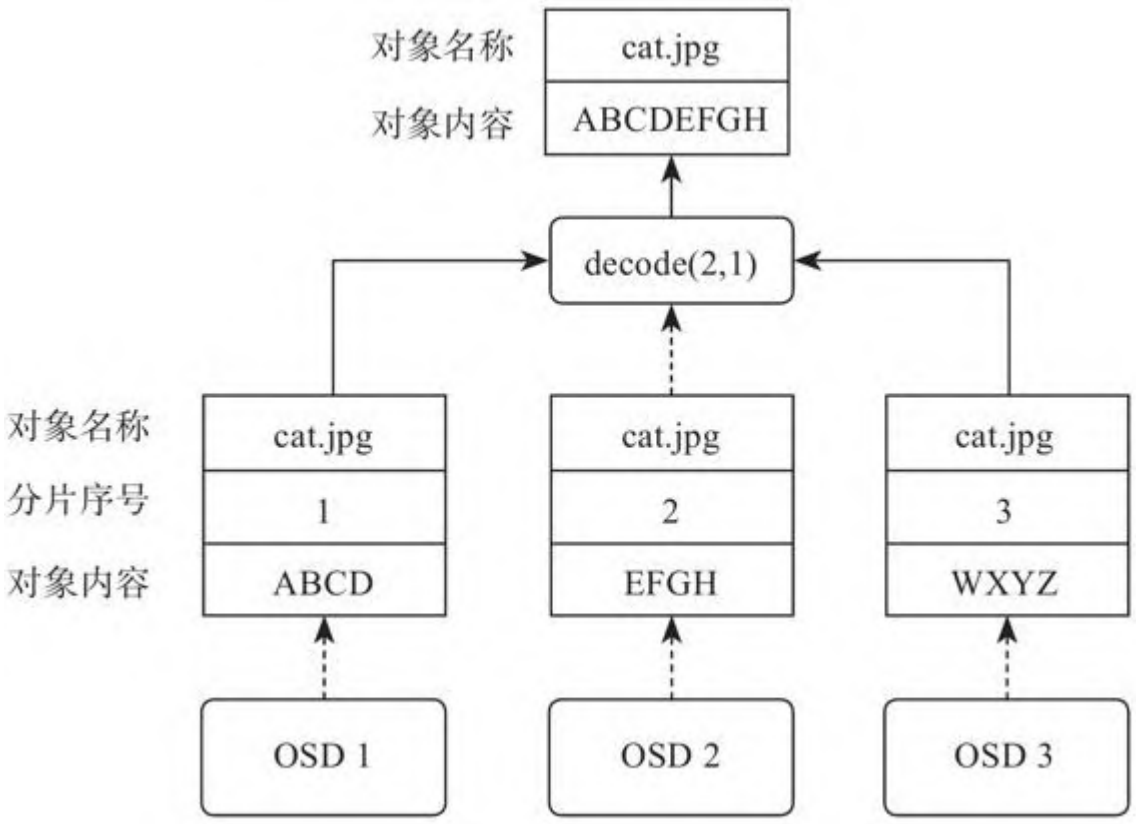

图11-3　在Ceph中利用纠删码读取对象数据

虽然纠删码能够提供和副本相近的数据可靠性，并降低冗余数据的开销，整体上能提高存储设备的可用空间。但是，纠删码所带来的额外开销主要是大量计算和网络高负载，优点同时伴随缺点。特别是在一个硬盘出现故障的情况下，重建数据非常耗费CPU资源，而且计算一个数据块时需要读出大量数据并通过网络传输。相比副本数据恢复，纠删码数据恢复时给

网络带来巨大的负担。因此，使用纠删码对硬件的设备性能是一个较大的考验，这点需要注意。另外，需要注意的是，使用纠删码所建立的存储资源池无法新建RBD块设备。

在前面章节介绍中，Ceph安装后默认有Default Rule，这个Rule默认是在Host层级进行三副本读写。副本技术带来的优点是高可靠性、优异的读写性能和快速的副本恢复。然而，副本技术带来的成本压力是较高的，特别是三副本数据情景下，每TB数据的成本是硬盘裸容量3倍以上（包括节点CPU和内存均摊开销）。纠删码具备与副本相近的高可用特性，而且降低了冗余数据的开销，同时带来了大量计算和网络高负载。以存储100GB数据为例，对比三副本和k+m=2+1两种类型所占用的存储容量，向大家简单介绍一下纠删码的容量优势。

表11-1　容量占用情况对比

类型	公式	结果
3 副本	存储容量 × 副本数 = 所需容量	100 × 3 = 300GB
K+M= 2+1	存储容量 ×(K/(K+M)) = 所需容量	100 × (2/(2+1)) ≈ 66.7GB

参考资料：

http://blog.sina.com.cn/s/blog_57f61b490102viq9.html 。

http://docs.ceph.com/docs/master/architecture/#erasure-coding 。

11.4　纠删码应用实践

11.4.1　使用Jerasure插件配置纠删码

Jerasure（http://jerasure.org/ ）是面向存储应用纠删码算法库的一个开源实现，该项目是由美国田纳西大学诺克斯维尔分校研究员James S.Plank、Scott Simmerman和Catherine D.Schuman共同开发的。Jerasure在GNU LGPL下发布。

Jerasure库设计目标是快速、灵活和模块化，支持横向模式（Horizontal mode）的纠删码。

Jerasure库是Ceph最早支持的纠删码库，在0.78版本引入，是Ceph默认的纠删码库。Ceph使用Jerasure库的RS纠错算法。Firefly是第一个支持纠删

码的长期支持版（Long Term Support，LTS）。Ceph提供的默认纠删码配置如下。

```
ceph osd erasure-code-profile get default
directory=/usr/lib64/ceph/erasure-code
k=2
m=1
plugin=jerasure
technique=reed_sol_van
```

其中，k=2代表数据块为2，m=1代表校验块为1。“k=2，m=1”相当于3块硬盘组成的RAID5。在前面提到的CRUSH里面，默认情况下数据的分布是以Host为单位，相同条带的数据块不会放置到相同的存储节点。在这个限制下，“k=2，m=1”至少需要3个存储节点。

从容量上计算，三副本的有效容量为33.33%，而“k=2，m=1”的纠删码场景下有效容量为66.66%，若配置为“k=5，m=2”则有效容量为71.43%。相比副本机制，纠删码提高了有效的存储容量。

创建测试的纠删码存储池，如下。

```
[root@ceph-mon-1 ~]# ceph osd pool create ec-pool1 64 64
erasure
pool 'ec-pool1' created
ceph osd dump
pool 22 'ec-pool1' erasure size 3 min_size 2 crush_ruleset 3
object_hash rjenkins pg_num 64 pgp_num 64 last_change 920
flags hashpspool stripe_width 4096
```

其中size=3就是k+m的值，而min_size=2则为k的值。以下测试一下纠删码存储池。

```
[root@ceph-mon-1 ~]# echo "ABCDEFGHI1234567890" | rados --
pool ec-pool1 put file1 -
[root@ceph-mon-1 ~]# rados --pool ec-pool1 get file1 -
ABCDEFGHI1234567890
[root@ceph-mon-1 ~]#
```

根据之前讲述，在创建纠删码存储池时，如果不指定纠删码配置，使用默认的设定条件。用户也可以根据实际需求定义纠删码配置文件，设定k和m的值。

```
[root@ceph-mon-1 ~]# ceph osd erasure-code-profile set my-
profile1 k=3 m=1 ruleset-failure-domain=rack
[root@ceph-mon-1 ~]# ceph osd erasure-code-profile get my-
profile1
directory=/usr/lib64/ceph/erasure-code
```

```
k=3
m=1
plugin=jerasure
ruleset-failure-domain=rack
technique=reed_sol_van
[root@ceph-mon-1 ~]# ceph osd pool create ec-pool2 64 64
erasure my-profile1
pool 'ec-pool2' created
[root@ceph-mon-1 ~]# echo "ABCDEFGHI1234567890" | rados --
pool ec-pool2 put file2 -
[root@ceph-mon-1 ~]# rados --pool ec-pool2 get file2 -
ABCDEFGHI1234567890
[root@ceph-mon-1 ~]#
```

其中，ruleset-failure-domain=rack表示相同条带下的数据不会存放到同一个rack里。

Ceph使用纠删码在进行数据存储时候，需要更多的额外资源。所以，相比副本而言，纠删码存储池缺少了一些功能特性，例如原子写操作（Partial Writes）。通过配合使用缓冲池技术，可以克服这个限制。

参考资料：

- http://docs.ceph.com/docs/master/rados/operations/erasure-code/#creating-a-sample-erasure-coded-pool 。

- http://docs.ceph.com/docs/master/rados/operations/pools/ 。

- http://docs.ceph.com/docs/master/architecture/#erasure-coding 。

11.4.2 ISA-L插件介绍

ISA-L插件是由Intel公司提供的纠删码开源解决方案（BSD授权），其核心算法部分采用的是汇编语言，同时针对Intel公司的CPU做过指令集层面的优化，相比Jerasure插件，在进行纠删码编解码时，ISA-L能够显著降低所需要的硬件资源消耗，同时有效降低编解码时长，因此大家在使用Intel公司的CPU同时对纠删码有较高的性能要求的场景下，可以考虑使用ISA-L插件。

参考资料：

- http://www.intel.com/content/www/us/en/storage/erasure-code-isa-l-solution-video.html 。

- https://software.intel.com/en-us/storage/ISA-L 。

11.4.3 LRC插件介绍

在使用Jerasure插件的时候，一个对象的数据将会以条带化的形式存储在多个OSD设备上，当有OSD出现故障的时候，底层存储需要对这个对象的数据做恢复操作的时候，需要从所有OSD中读取相应的条带信息，之后解码计算出最终的数据内容。可以试想一下，如果一个负载本来就比较高的存储系统，还需要频繁进行这类“牵一发而动全身”的数据恢复操作，对整个存储系统的性能和稳定性将会是极大的考验。因此，为了降低因为OSD故障而导致的数据恢复过程中的性能消耗，同时降低影响这类操作的范围，LRC插件特别做了这方面的优化：将原来的对象数据条带化做了进一步的细分，将原本需要分布在不同OSD的切片进行本地化分组（所谓本地化分组就是按照一定规则将一个分片组集中存储在单个OSD上），之后再以组为最

小单位均匀分布到不同的OSD上面。当OSD出现故障的时候，通过单个本地化分组（也就是单个OSD）就能恢复出对象的数据，极大减少了需要进行数据读取操作的OSD数量，从而降低了整体性能的消耗。关于LRC插件的配置与使用请参考以下链接，这里不再赘述。

- 参考资料：

http://docs.ceph.com/docs/master/rados/operations/erasure-code-lrc/ 。

11.4.4 其他插件介绍

Ceph开源社区一直都在积极推进新插件的接入与旧有插件的优化，其他插件内容可以参考以下文档，这里不再赘述。

- SHEC插件：http://docs.ceph.com/docs/master/rados/operations/erasure-code-shec/ 。

11.5 本章小结

本章重点向读者介绍了缓冲池和纠删码两种关键技术，从理论到实践尽可能让读者在短时间内了解和掌握这两大关键技术。加上作者自身的知识和技术水平有限，这两大技术在实际应用中并未有较大规模的生产系统部署，究其原因，主要是这两个技术目前成熟度不够，在一些应用过程中遇上的Bug较多，从作者的经验来看，偏向运维而研发能力较弱的技术团队不建议大规模应用到生产环境，如果必须要用的话，请务必做好生产上线前的各项测试，避免踩坑。本章介绍的两大技术标志着Ceph作为一个开源存储解决方案逐步走向成熟，在开源的力量的推动下Ceph在功能上逐步减小和同类商用产品的差距，虽然在短期内这一类的新技术或多或少会存在一些问题，但长远来看，Ceph的明天会更好。

第12章　生产环境应用案例

随着云计算OpenStack的大力发展，Ceph也得到了飞速的发展，向着云计算的方向靠拢，Ceph FS文件系统已经不再是开发重点，但仍未放弃，在Jewel版本中被正式对外宣传可以进入生产环境。

12.1 Ceph FS应用案例

Ceph FS是一个支持POSIX接口的文件系统存储类型。目前Ceph FS的发展比较滞后，主要是由于Ceph FS技术不成熟，另一方面由于云计算大潮的突起，比Ceph FS起步晚的Ceph RBD和Ceph Radosgw发展反而更加活跃，社区的发展重点也大多放在了后两者上。

由于Ceph FS的不成熟，Ceph官方也有很明显的说明，这一重要原因使其不能满足生产环境的要求，不能得到广泛应用。但是，在近期发布的Jewel版本中被正式对外宣传可以进入生产环境。

Ceph FS主要对接于传统的应用，例如直接mount到服务器作为文件系统使用，另外就是用来对接Hadoop和对接OpenStack Manila项目。

对接Hadoop在第6章中已经讲过，这里主要讲解Ceph FS如何导出成NFS来使用，Ceph FS如何在Windows环境下使用以及如何对接OpenStack Manila项目。

12.1.1　将Ceph FS导出成NFS使用

NFS（Network FileSystem）是类UNIX系统中最流行的网络共享文件系统之一。即使是不支持Ceph FS文件系统的类UNIX系统，也可以通过NFS来访问Ceph文件系统。为了能使用NFS访问Ceph文件系统，我们需要一个能够将Ceph FS重新导出为NFS的NFS服务器。NFS-Ganesha就是一个利用libcephfs库支持Ceph FS FSAL（File System Abstraction Layer，文件系统抽象层）的NFS服务器，它运行在用户态空间。

本节将展示如何将ceph-node1创建成一个NFS-Ganesha服务器，然后把Ceph FS导出为一个NFS并挂载到client-node1上。

1）在ceph-node1上安装nfs-ganesha所需要的包。

```
# yum install -y nfs-utils nfs-ganesha nfs-ganesha-fsal-ceph
```

2）在防火墙设置中，打开所需的端口（通常是2049）。因为现在配置的是测试环境，我们可以直接把防火墙关掉。

```
# systemctl stop firewalld; systemctl disable firewalld
```

3）打开NFS所需的RPC服务。

```
# systemctl start rpcbind; systemctl enable rpcbind
# systemctl start rpc-statd.service
```

4）创建NFS-ganesha的配置文件/etc/ganesha.conf，并输入如下内容。

```
EXPORT {
    Export_ID = 1;
    Path = "/";
    Pseudo = "/";
    Access_Type = RW;
    NFS_Protocols = "3";
    Squash = No_Root_Squash;
    Transport_Protocols = TCP;
    SecType = "none";
```

```
    FSAL {
         Name = CEPH;
    }
}
```

5）最后，用第4）步中创建的配置文件ganesha.conf作为参数启动ganesha nfs的守护进程。然后，可以用showmount命令来验证导出的NFS共享文件系统。

```
# ganesha.nfsd -f /etc/ganesha.conf -L /var/log/ganesha.log
-NNIV_DEBUG -d
# showmount -e
```

下面，为了在客户端机器上挂载NFS共享文件系统，只需要安装NFS客户端软件包,然后挂载之前从ceph-node1导出的共享文件系统即可。

6）在ceph-node1上安装NFS客户端软件包，并执行挂载命令。

```
# apt-get install nfs-common
# mkdir /mnt/cephfs
# mount -o rw,noatime 192.168.1.101:/ /mnt/cephfs
```

至此，就完成了Ceph FS导出NFS来使用。

12.1.2　在Windows客户端使用Ceph FS

我们已经学习了几种不同的访问Ceph文件系统的方法，如Ceph FUSE、Ceph内核驱动以及NFS Ganesha；这几种方法都只能用于Linux系统中，在Windows系统的客户端中则无法使用。

围绕着Ceph已经发展出了一个繁荣的社区，像Ceph这样的开源项目总是有着它自己的优势。Ceph-Dokan是一个Windows系统上的原生Ceph客户端，开发者是UnitedStack（有云）的前存储工程师孟圣智。除了参与OpenStack和Ceph的开发之外，他还管理着Ceph-Dokan项目。

为了能够从Windows平台上访问Ceph文件系统，Ceph-Dokan主要使用了两个组件：一个是libcephfs.dll，用于访问Ceph FS；另一个是ceph-

dokan.exe，它基于Dokan项目在Windows平台上提供类似FUSE的服务，以便能够在Windows系统中将Ceph文件系统挂载为一个本地盘。在后台，ceph-dokan.exe利用dokan.dll和libcephfs.dll实现了win32用户态文件系统。像Ceph一样，Ceph-Dokan也是一个开源项目。你可以从这个链接获取它的源码：https://github.com/ceph/ceph-dokan ，并且随时可以发送pull request来向它贡献代码。你可以选择从源码编译Ceph-Dokan；为了简化工作，你也可以直接使用从Github上克隆的ceph-cookbook项目中的ceph-dokan\ceph-dokan.exe应用程序。

1）配置一台Windows 7或者Windows 8的机器，并将其加入Ceph集群的同一个网络（192.168.1.0/24）。

2）执行下面的telnet命令来进行验证，以确保与Ceph monitor之间可达。

```
telnet 192.168.1.101 6789
```

3）确认能够访问Ceph cluster之后，就可以从https://github.com/ksingh7/ceph-cookbook/tree/master/ceph-dokan 将ceph-dokan.exe和DokanInstall_0.6.0.exe下载到Windows客户端系统中。

4）安装DokanInstall_0.6.0.exe。如果你用的是Windows 8系统，那么可能需要用兼容模式来进行安装。图12-1是Ceph-Dokan软件安装图。

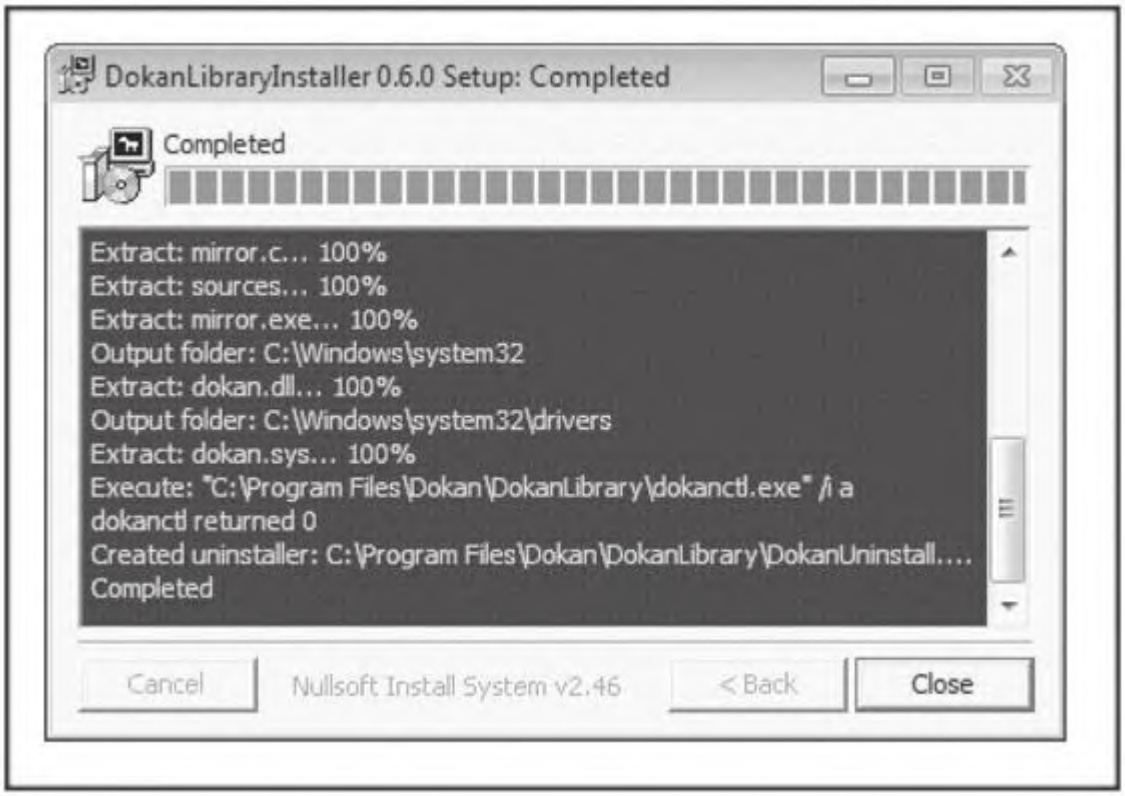

图12-1　Ceph-Dokan软件安装图

5）打开Windows命令行，并进入ceph-dokan.exe所在的目录。

6）在Windows平台上创建文件ceph.conf，该文件用于将Ceph集群监控器（Ceph Cluster Monitors）的信息告诉Ceph-Dokan。文件内容如下。

```
[global]
auth client required = none
```

```
log_file = dokan.log
mon_initial_members = ceph-node1
mon_host = 192.168.1.101
[mon]
[mon.ceph-node1]
mon addr = 192.168.1.101:6789
```

7）由于目前Ceph-Dokan不支持Cephx验证，因此要使用Ceph-Dokan，你需要在所有的Ceph集群监控机器上把Cephx禁用。

8）可以通过把/etc/ceph/ceph.conf文件中与auth相关的配置项改成none来禁用Ceph集群的cephx。

```
auth_cluster_required = none
auth_service_required = none
auth_client_required = none
```

9）修改好ceph.conf之后，重启所有监控节点上的Ceph服务。

```
# service ceph restart
```

10）确保cephx已禁用。

```
# ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config
show |grep -i auth | grep -i none
```

11）最后，在客户端Windows系统中运行ceph-dokan.exe来挂载Ceph FS。在命令行中运行如下命令。

```
ceph-dokan.exe -c ceph.conf -l m
```

这个命令会将Ceph FS挂载为Windows系统中的一个本地盘，如图12-2所示。

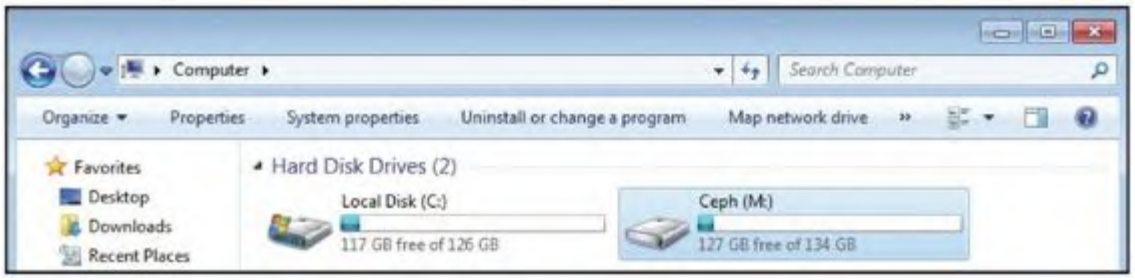

图12-2　Windows计算机显示图

目前，Ceph FS和Ceph-Dokan都需要更多的代码贡献来使其足够成熟，才能支持生产系统的负载。然

而，将它们作为验证和测试的平台，仍然是很好的选择。

提示

由于Ceph-Dokan只能正确读取UNIX格式的ceph.conf文件，因此你要用dos2unix把你的ceph.conf转换成UNIX格式。关于dos2unix程序的更多信息请参考 http://sourceforge.net/projects/dos2unix/ 。

12.1.3 OpenStack Manila项目对接Ceph FS案例

Manila项目全称是File Share Service（文件共享即服务），它是OpenStack大帐篷模式下的子项目之一，用来提供云上的文件共享，支持CIFS协议和NFS协议。Ceph FS（https://review.openstack.org/#/c/270211/ ）驱动目前已经被OpenStack合并，可以用来直接使用。不过，不建议生产使用此驱动，实际上，在云环境下是不太会允许VM业务网络能够直接访问后端的存储网络的，而在VM上直接提供对Ceph FS的访问也暴露了Ceph，因此，大概只能在内部小规模私有云中被接受，架构如图12-3所示。

还有以下一些关于Ceph对接Manila的驱动。

• Default Driver：使用RBD作为Manila Service VM的后端，在VM上启动NFS实例提供服务。

• Ganesha Driver：使用Ganesha将Ceph FS重新Reexport出去。

• Native CephFS Driver：在Guest VM上直接使用原生Ceph FS Module访问。

• VirtFS Driver：将Ceph FS挂载在Host端，VM通过VirtFS访问。

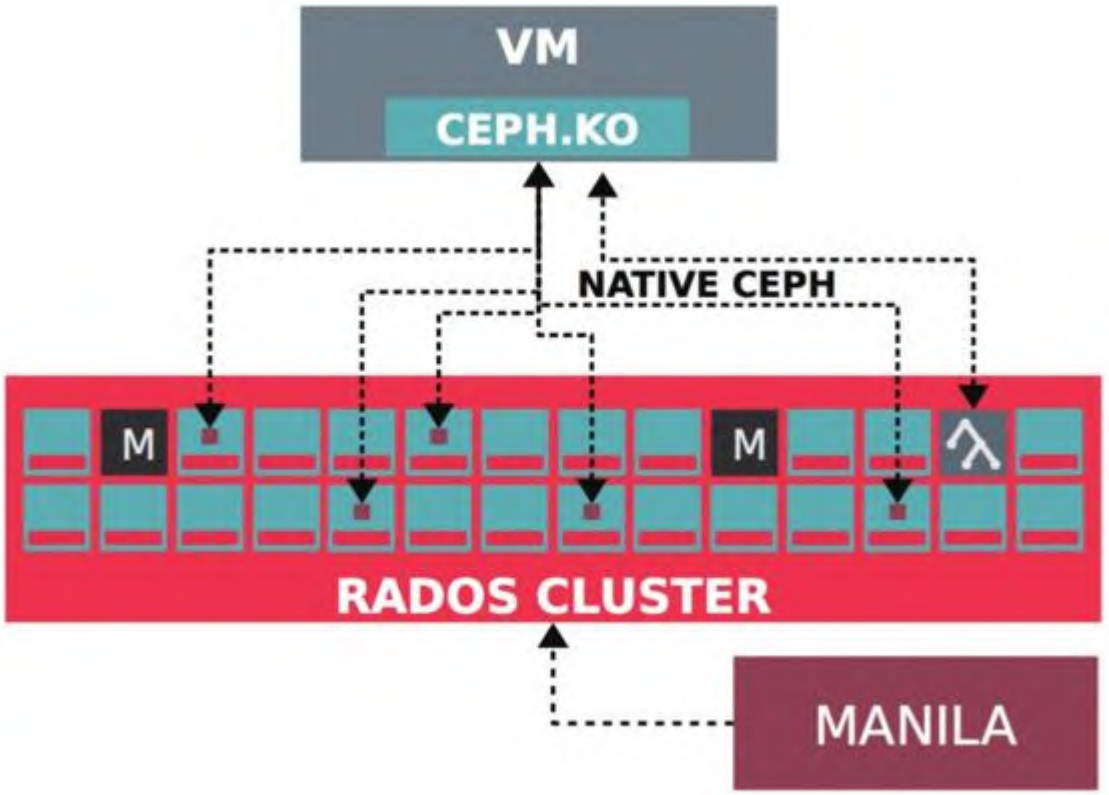

图12-3　Manila对接架构图

下面具体讲解Ceph FS如何对接Manila。

1）创建Manila Cephx认证。

```
ceph auth get-or-create client.manila mon 'allow r; allow
command "auth del" with entity prefix client.manila.; allow
command "auth caps" with entity prefix client.manila.; allow
command "auth get" with entity prefix client.manila., allow
command "auth get-or-create" with entity prefix
client.manila.' mds 'allow *' osd 'allow rw' >
keyring.manila
```

2）开启快照。

```
ceph mds set allow_new_snaps true --yes-i-really-mean-it
```

3）配置Manila，如同配置Cinder多后端一样，配置Manila使用Ceph FS后端。

```
enabled_share_protocols = NFS,CIFS,CEPHFS
[cephfs1]
driver_handles_share_servers = False
share_backend_name = CEPHFS1
share_driver =
manila.share.drivers.cephfs.cephfs_native.CephFSNativeDriver
cephfs_conf_path = /etc/ceph/ceph.conf
cephfs_auth_id = manila
enabled_share_backends = generic1, cephfs1
```

4）创建Manila Type。

```
manila type-create cephfstype false
```

5）创建Share。

```
manila create --share-type cephfstype --name cephshare1
cephfs 1
```

6）使用fuse-client挂载。

①创建一个key。

```
[client.share-4c55ad20-9c55-4a5e-9233-8ac64566b98c]
key = AQA8+ANW/4ZWNRAAOtWJMFPEihBA1unFImJczA==
```

②创建一个ceph.conf。

```
[client]
        client quota = true
 [mon.a]
        mon addr = 192.168.1.7:6789
[mon.b]
        mon addr = 192.168.1.8:6789
 [mon.c]
        mon addr = 192.168.1.9:6789
```

③使用创建好的key和ceph.conf来挂载到mnt下。

```
ceph-fuse --id=share-4c55ad20-9c55-4a5e-9233-8ac64566b98c -c
./client.conf --keyring=./client.keyring --client-
mountpoint=/volumes/share-4c55ad20-9c55-4a5e-9233-
8ac64566b98c ~/mnt
```

参考资料：

- http://docs.openstack.org/developer/manila/devref/cephfs_native_driver.html?highlight=cephfs 。

- http://www.wzxue.com/tag/manila/ 。

12.2 RBD应用案例

RBD是RADOS BLOCK DEVICE的简称。RBD是Ceph分布式存储中最常用的存储类型。

块是一个有序列字节，普通的一个块大小为512字节。基于块的存储是最常见的存储方式，比如常见的硬盘、软盘和CD光盘等，都是存储数据最简单快捷的设备。

Ceph块设备是一种精简置备模式，可以扩展大小并且数据是以条带化的方式存储在一个集群中的多个OSD中，RBD具有快照、多副本、克隆和一致性功能。

Ceph也逐渐成为了OpenStack的默认后端存储，下面主要讲解RBD对接OpenStack的案例。

12.2.1　OpenStack对接RBD典型架构

在过去的几年中，OpenStack越来越受到青睐，因为它通过软件定义了计算、网络甚至存储。当谈论起OpenStack的存储时就会注意到Ceph。根据OpenStack基金会2016年4月的用户调查报告显示，有57%的受访用户选择Ceph作为其存储后端，其中已经有39%的用户已经应用到生产环境，足以证明Ceph的稳定性和可靠性，Ceph已经成为当前应用最为广泛的软件定义存储解决方案。

Ceph提供了OpenStack一直在寻找的稳健、可靠的存储后端。它与OpenStack的Cinder、Glance和Nova等模块无缝集成，提供了一个完整的云存储后端。下面是使Ceph成为OpenStack的最佳匹配的一些主要优点。

• Ceph提供了企业级的功能丰富的存储后端，而且每GB成本非常低，这有助于保持OpenStack云部署成本较低。

• Ceph为OpenStack提供了一个统一的包含块、文件或者对象存储的解决方案，让应用程序各取所需。

• Ceph为OpenStack云提供了先进的块存储功能，包含轻松和快速地孵化实例，以及备份和克隆它们。

• 它为OpenStack实例提供了默认的持久卷，使其可以像传统的服务器那样工作，实例中的数据将不会因为重新启动虚拟机而丢失。

• Ceph支持虚机迁移，这使得OpenStack实例能做到独立于宿主机；而且扩展存储组件也不会影响虚拟机。

• 它为OpenStack卷提供的快照功能，也可以当做备份的一种手段。

• Ceph的COW克隆功能为OpenStack提供了同时孵化多个实例的能力，这有助于提供快速的虚拟机创建机制。

• Ceph支持为Swift和S3对象存储提供丰富的API接口。

Ceph和OpenStack社区已经在过去的几年里密切合作，使整合更加无缝，并充分利用它们各自的新功能。在未来，我们可以预期OpenStack和Ceph联系将更加密切。

OpenStack是一个多组件构成的模块化系统，并且每个组件都有其自身明确的任务。有多个组件需要可靠的像Ceph一样的存储后端，并且扩展集成起来，如

图12-4所示。每个组件通过自身的方式去调用Ceph来存储块设备和对象。主流的基于OpenStack和Ceph搭建的云平台使用了Cinder，Glance和Nova等模块来与Ceph集成，如图12-4所示。

OpenStack云主机服务Nova、镜像服务Glance和云硬盘服务Cinder三大服务的后端统一使用Ceph，进行高效管理，解决了虚拟机创建时间长和镜像风暴等问题，还能让虚拟机随便漂移。

OpenStack对接Ceph之后可以实现COW（Copy On Write）秒速开机，避免了Nova下载上传的一个动作，可以直接拷贝Glance存储池的镜像数据进行启动。

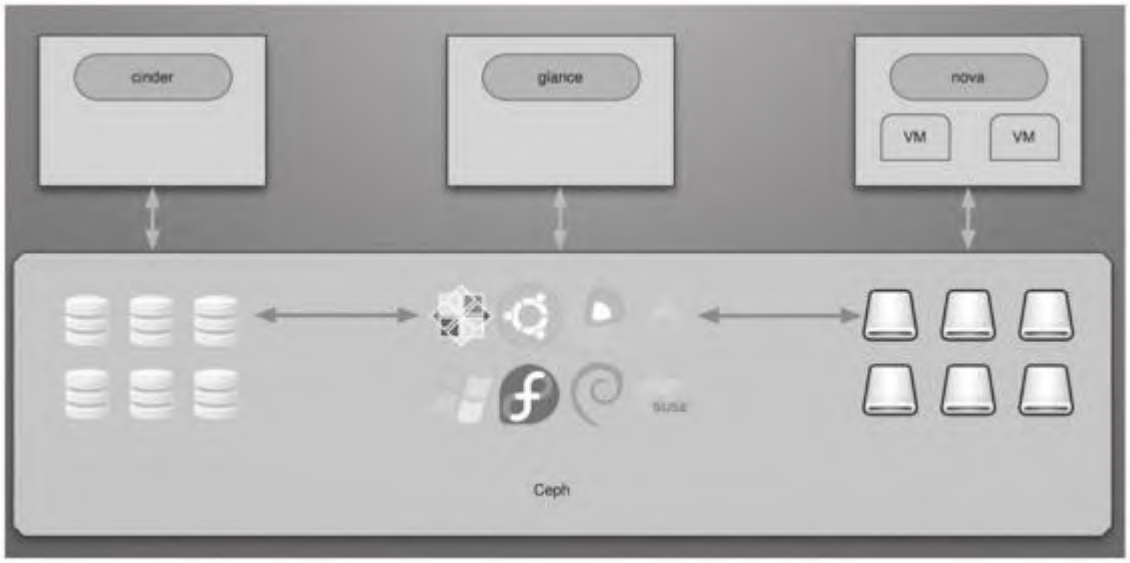

图12-4　OpenStack对接Ceph的典型架构图

12.2.2　如何实现Cinder Multi-Backend

OpenStack也可以通过Cinder Multi-Backend功能对接Ceph的SSD Pool或SATA Pool来实现性能型、容量型的云硬盘类型，用来满足数据库和大文件应用，如图12-5所示。

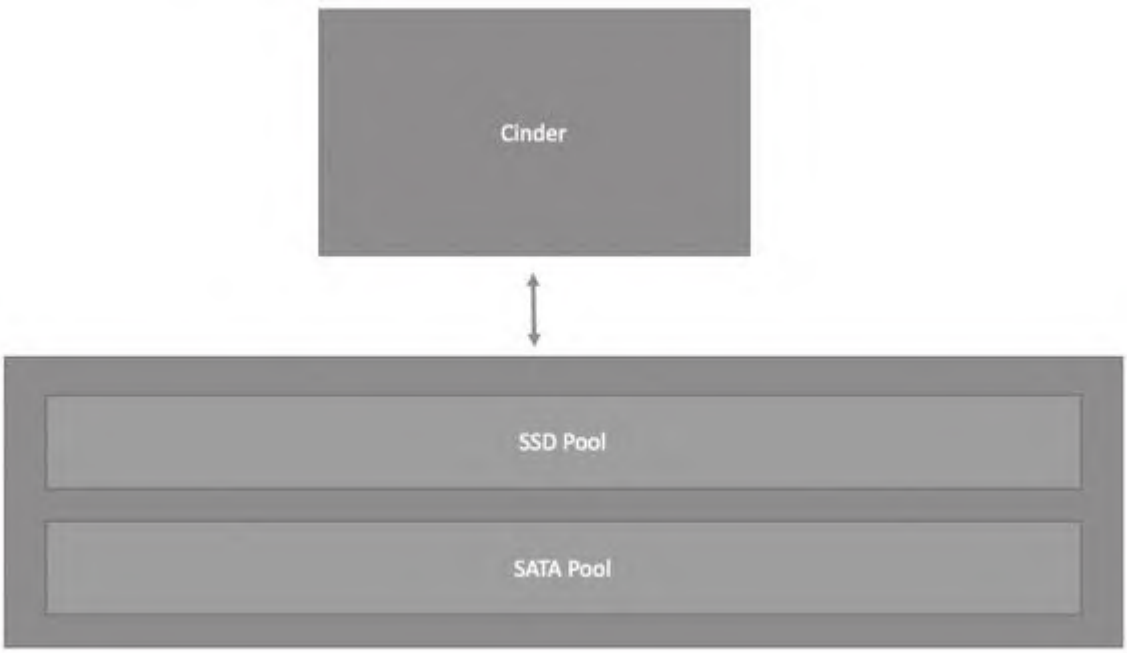

图12-5　Cinder Multi-Backend典型架构

在开启Cinder Multi-Backend功能之前需要有以下条件。

- 拥有一个Ceph集群。
- 拥有SSD池和SATA池。

下面是具体步骤。

（1）配置Cinder多后端存储

```
# Multi backend options
# Define the names of the groups for multiple volume
backends
enabled_backends=rbd-sata,rbd-ssd
# Define the groups as above
[rbd-sata]
volume_driver=cinder.volume.driver.RBDDriver
rbd_pool=cinder-sata
volume_backend_name=RBD_SATA
# if cephX is enable
#rbd_user=cinder
#rbd_secret_uuid=<None>
[rbd-ssd]
volume_driver=cinder.volume.driver.RBDDriver
rbd_pool=cinder-ssd
volume_backend_name=RBD_SSD
# if cephX is enable
#rbd_user=cinder
#rbd_secret_uuid=<None>
```

（2）关联Cinder Backend到Ceph存储池

```
$ cinder type-key ssd set volume_backend_name=RBD_SSD
$ cinder type-key sata set volume_backend_name=RBD_SATA
$ cinder extra-specs-list
+--------------------------------------+------+------------------
---------------+
|                  ID                  | Name |
extra_specs
+--------------------------------------+------+------------------
---------------+
| b1522968-e4fa-4372-8ac4-3925b7c79ee1 | ssd  |
{u'volume_backend_name': u'RBD_SSD'} |
| b50bf5a3-6044-4392-beeb-432302f6421c | sata |
{u'volume_backend_name': u'RBD_SATA'} |
+--------------------------------------+------+------------------
---------------+
```

（3）重启服务

```
sudo restart cinder-api; sudo restart cinder-scheduler ;
sudo restart cinder-volume
```

（4）创建Cinder Type

```
$ cinder type-create ssd
+--------------------------------------+------+
|                  ID                  | Name |
+--------------------------------------+------+
| b1522968-e4fa-4372-8ac4-3925b7c79ee1 | ssd  |
+--------------------------------------+------+
$ cinder type-create sata
+--------------------------------------+------+
```

```
|                  ID                  | Name |
+--------------------------------------+------+
| b50bf5a3-6044-4392-beeb-432302f6421c | sata |
+--------------------------------------+------+
```

参考资料：

- http://sangh.blog.51cto.com/6892345/1609285 。

12.3 Object RGW应用案例：读写分离方案

下面介绍一些基于RGW的对象存储应用案例，帮助读者了解对象存储在互联网的重点应用案例。

1.构架介绍

利用Nginx作为前端入口，通过简单配置，轻松实现客户端访问的读写分离：将来自客户端的PUT/POST一类请求通过proxy方式交由后端的nginx-write集群，而来自客户端的GET/HEAD一类请求则由nginx-read集群进行处理，如果12-6所示。

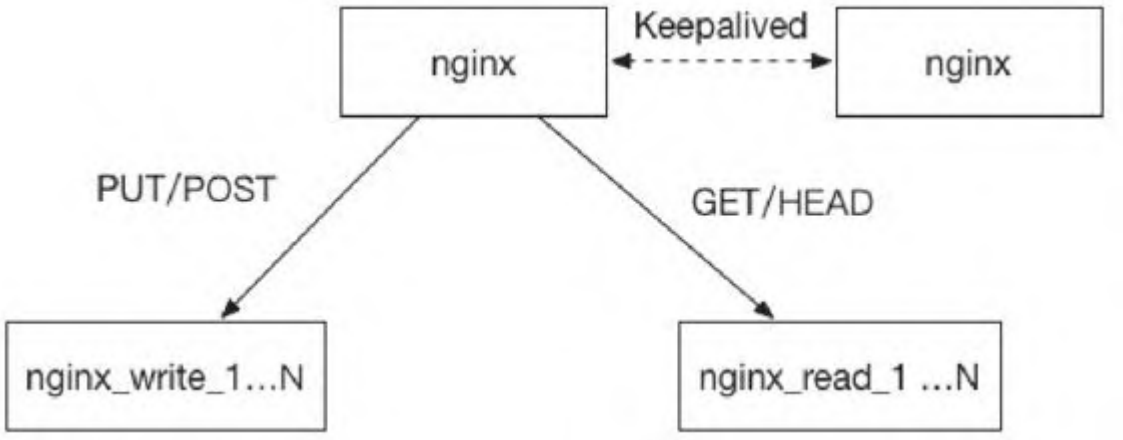

图12-6　读写分离前端架构

2.Nginx配置

下面是Nginx配置代码。

```
upstream write_zone {
zone write_zone 64k;
server 192.168.1.2:80;
server 192.168.1.3:80;
......
}
upstream read_zone {
zone read_zone 64k;
server 192.168.1.4:80;
server 192.168.1.5:80;
......
}
server {
    listen       80;
    server_name  s3.ceph.work *.s3.ceph.work;
    location / {
        if ($request_method = "(PUT|POST)"){
        proxy_pass http://write_zone;
        }
            proxy_pass http://read_zone;
```

```
                                        }
        .....其他配置

    }
```

笔者这里只是抛砖引玉，使用了最简单的配置，前端入口处的Nginx可以通过Keepalived来实现高可用，后端可以通过nginx-upstream-check-module一类的模块提高系统可用性。

相关模块下载地址：https://github.com/yaoweibin/nginx_upstream_check_module 。

12.4　基于HLS的视频点播方案

1.HLS简介

HLS [1] 是Apple公司推出的一套基于HTTP的流媒体实时传输协议，全称是HTTP Live Streaming，同时能够跨多种平台和操作系统，在视频分享、在线教育等领域得到了广泛的应用。本文不对HLS做过多介绍，只是简单介绍如何将一个普通的MP4文件转换成HLS所支持的格式，并进行在线视频播放。

1）客户端将mp4文件转码成m3u8格式，上传到RADOSGW中，并设置相应的访问权限，如图12-7所示。

2）完成转码后，客户端通过访问Radosgw服务，实现对HLS视频的在线播放，如图13-8所示。

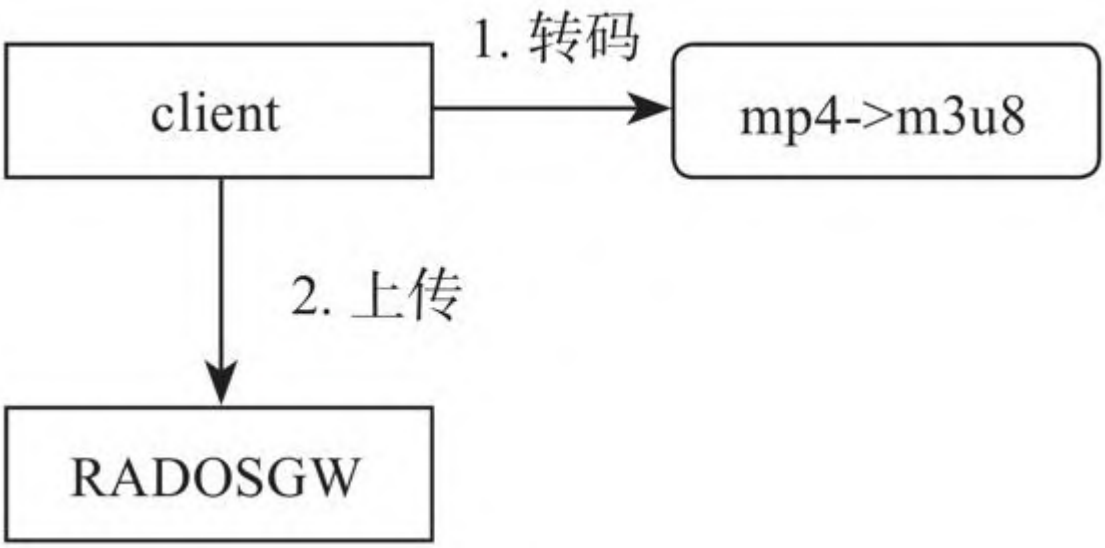

图12-7　HLS转码流程

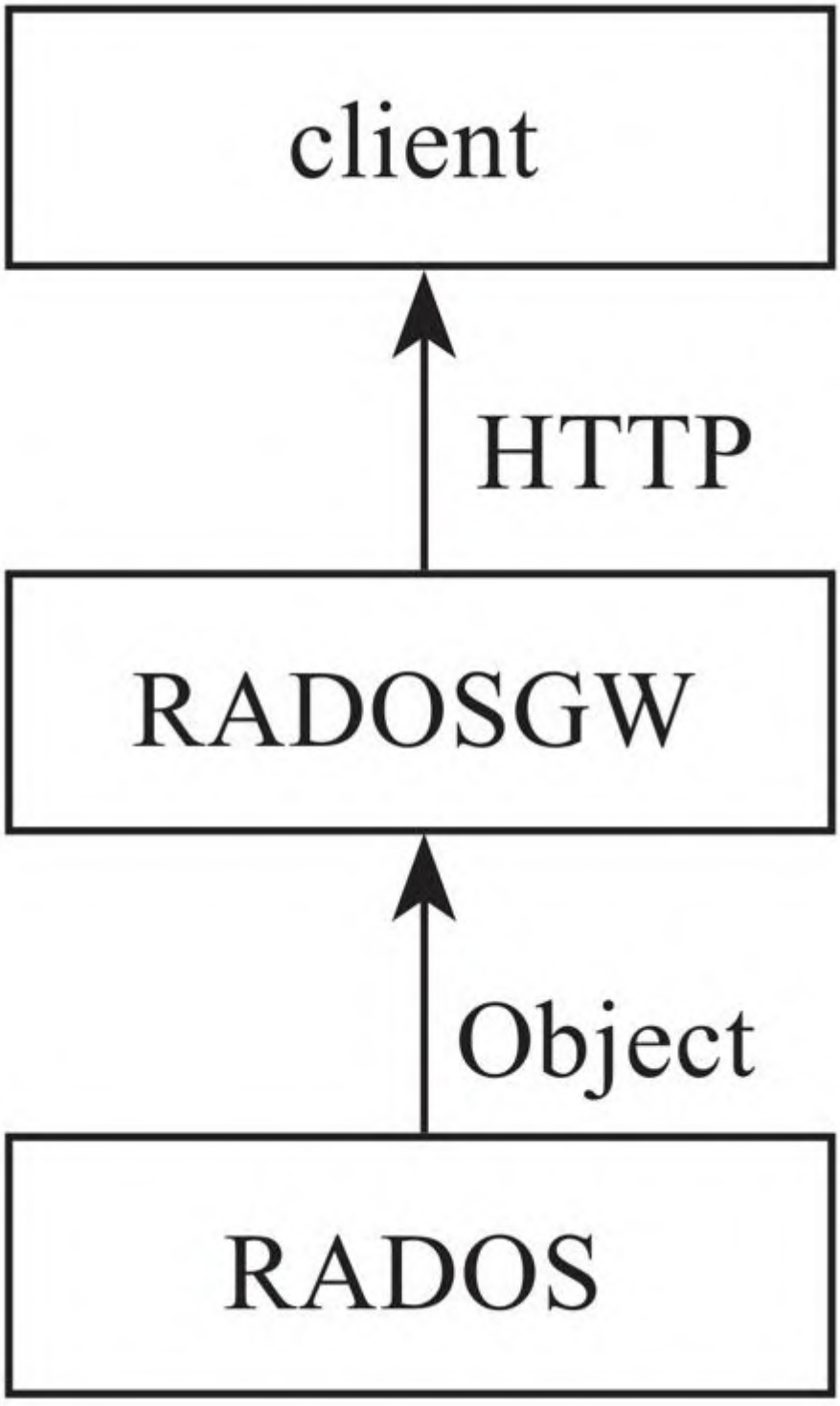

图12-8　客户端访问流程

2. 安装准备

1）client端需要安装ffmpeg工具，进行mp4→m3u8的转码，或者使用其他转码工具。ffmpeg的安装请参考网上资料。

2）client端需要有S3上传工具，比如s3cmd，或者其他上传方式。

3. 具体步骤

1）将mp4文件转码成m3u8。

```
mkdir m3u8
cd m3u8
ffmpeg -i demo.mp4 -c:v libx264 -c:a aac -strict -2  -
hls_list_size 0 -hls_time 5 -f hls demo.m3u8
```

上面命令的意思是以5s为间隔，将原来的mp4文件转码成demo.m3u8和一些ts文件，读者可以根据实际情况调整参数。

2）上传m3u8文件。

①上传本地文件到名字为hls的Bucket中。

```
s3cmd put m3u8 s3://hls --preserve
```

②设置文件访问权限。

```
s3cmd setacl s3://hls/m3u8 --acl-public --preserve
```

3）播放测试。

Window系统下使用VLC一类支持HLS的播放器打开 http://s3.ceph.work/hls/m3u8/demo.m3u8 进行播放，或者Apple MAC/IOS系统可以使用Safari浏览器直接进行播放，其他系统需要加载相关浏览器插件。只有MAC或者IOS系统的Safari浏览器原生支持HLS格式，其他环境需要添加相应插件。最后推荐几款HLS播放器及插件，如下。

- VLC：http://www.videolan.org/vlc/ 。
- jwplayer：http://www.jwplayer.com/ 。
- videojs：https://github.com/videojs/videojs-contrib-hls 。
- sewise-player：https://github.com/jackzhang1204/sewise-player 。

[1] HLS 介绍：https://developer.apple.com/streaming/。

12.5 本章小结

本章从生产环境使用案例来让读者了解到生产环境中应该如何使用Ceph。对于RGW应用，通过介绍一个简单的基于Naginx的读写分离模型，让读者了解到基于通用的HTTP协议调优手段同样适用于RGW对象存储场景，同时通过RGW结合HLS协议，让读者了解到目前互联网比较热门的点播、直播一类平台的基础应用场景。

第13章　Ceph运维与排错

13.1　Ceph集群运维

本章我们简单介绍Ceph的日常运维。Ceph运维是保证Ceph集群和整个云平台正常运行与稳定性不可或缺的一部分，在笔者看来是很重要的，国内大部分公司都是以用户的角度来使用Ceph，所以Ceph运维成为目前Ceph的几大难题之一。首先从Ceph集群的扩展、维护与监控3方面来讲解一下Ceph日常运维的一些经验。

13.1.1 集群扩展

由于原集群存储空间不足或某种原因需要扩展集群，主要分为添加OSD和MON（monitor）等。

1.添加OSD节点

1）在Ceph-admin节点添加新增OSD节点的hosts文件。

```
  root@localhost:~# cat /etc/hosts
192.168.1.2  node1
192.168.1.3  node2
192.168.1.4  node3
192.168.1.5  node4
```

2）在node4节点配置Ceph APT源/YUM源。

```
root@localhost:~# echo deb http://ceph.com/debian-{ceph-stable-
release}/ $(lsb_release -sc) main | sudo tee
/etc/apt/sources.list.d/ceph.list
root@localhost:~# vim /etc/yum.repos.d/ceph.repo
[ceph-noarch]
name=Cephnoarch packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/noarch
enabled=1
gpgcheck=1
type=rpm-md
```

```
gpgkey=https://ceph.com/git/?
p=ceph.git;a=blob_plain;f=keys/release.asc
```

3）添加APT源key。

```
root@localhost:~# wget -
```

```
q -
```

```
O -'https://ceph.com/git/?
p=ceph.git;a=blob_plain;f=keys/release.asc' | sudo apt-key add -
```

4）配置node4节点SSH无密码登录，拷贝key到node4节点。

```
root@localhost:~# ssh-copy-idnode4
```

5）在ceph-admin给node4安装Ceph。

```
root@localhost:~# ceph-deploy install {ceph-node}[{ceph-node} ...]
```

例如上面第5步，我们把相应的[ceph-node]替换成换成节点名。

```
root@localhost:~# ceph-deploy install node4
```

6）初始化node4节点磁盘。

```
root@localhost:~# ceph-deploy disk zap {osd-server-name}:{disk-name}
```

例如上面第6步，我们把相应的[osd-server-name]：[disk-name]替换成节点名：磁盘。

```
root@localhost:~# ceph-deploy disk zap node1:sdb
```

7）准备node4节点磁盘OSD。

```
root@localhost:~# ceph-deploy osd prepare {node-name}:{data-disk}[:{journal-disk}]
```

例如上面第7步，我们把相应的{node-name}:{data-disk}[:{journal-disk}]替换成节点名:分区:日志磁盘。

```
root@localhost:~# ceph-deploy osd prepare node1:sdb1:sdc
```

8）激活node4节点磁盘OSD。

```
root@localhost:~# ceph-deploy osd activate {node-name}:{data-disk-partition}[:{journal-disk-partition}]
```

例如上面第8步，我们把相应的{node-name}:{data-disk-partition}[:{journal-disk-partition}]替换成节点名:分区:日志磁盘。

```
root@localhost:~# ceph-deploy osd activate node1:sdb1:sdc
```

9）更新Crush Map信息。

```
root@localhost:~# ceph osd crush add-bucket node4 host
root@localhost:~# ceph osd crush move node4 root=default
root@localhost:~# ceph osd crush add osd.$i 1.0 host=node4
#标注此处
```

osd.$i的

$i代表

osd序号，

1.0代表

osd的权重

10）查看是否添加成功。

```
root@localhost:~# ceph osd tree
ID WEIGHT  TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 5.15991 root default
-2 1.28998     host node1
 7 0.42999         osd.1           up  1.00000          1.00000
 6 0.42999         osd.2           up  1.00000          1.00000
 8 0.42999         osd.3           up  1.00000          1.00000
-3 1.28998     host node2
 2 0.42999         osd.4           up  1.00000          1.00000
 0 0.42999         osd.5           up  1.00000          1.00000
 1 0.42999         osd.6           up  1.00000          1.00000
-4 1.28998     host node3
11 0.42999         osd.7           up  1.00000          1.00000
10 0.42999         osd.8           up  1.00000          1.00000
 9 0.42999         osd.9           up  1.00000          1.00000
-5 1.28998     host node4
 5 0.42999         osd.10          up  1.00000          1.00000
 3 0.42999         osd.11          up  1.00000          1.00000
 4 0.42999         osd.12          up  1.00000          1.00000
```

2. 删除OSD节点

Ceph-deploy不支持一键删除OSD，所以采用手动删除OSD的方式。

1）停止OSD的相关进程，删除OSD相关目录。

```
root@localhost:~# stop ceph-osd id=x        #标注此处的
```

X表示

OSD的编号

2）从Crush Map移除OSD的信息（此时会进行重构）。

```
root@locahost:~# ceph osd out osd.x                    #标注此处的
```

X表示

OSD的编号

```
root@localhost:~# ceph osd crush remove osd.x          #标注此处的
```

X表示

OSD的编号

3）删除OSD的认证信息。

```
root@locahost:~# ceph auth del osd.x                    #标注此处的
```

X表示

OSD的编号

4）删除OSD。

```
root@locahost:~# ceph osd rm x                          #标注此处的
```

X表示

OSD的编号

3.添加MON（Monitor）节点

1）在ceph-admin节点添加新增OSD节点的hosts文件。

```
  root@localhost:~# cat /etc/hosts
192.168.1.2  node1
192.168.1.3  node2
192.168.1.4  node3
192.168.1.5  node4
192.168.1.6 node5
```

2）在node4节点配置Ceph APT源/YUM源。

```
root@node4:~# echo deb http://ceph.com/debian-{ceph-stable-
release}/ $(lsb_release -sc) main | sudo tee
/etc/apt/sources.list.d/ceph.list
root@node4:~# vim /etc/yum.repos.d/ceph.repo
[ceph-noarch]
name=Cephnoarch packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?
p=ceph.git;a=blob_plain;f=keys/release.asc
```

3）添加APT源key。

```
root@node4:~# wget -

q -

O -'https://ceph.com/git/?
p=ceph.git;a=blob_plain;f=keys/release.asc' | sudo apt-key add -
```

4）配置node4节点SSH无密码登录，拷贝key到node5节点。

```
root@node4:~# ssh-copy-id node5
```

5）在ceph-admin给node4安装Ceph。

```
root@node1:~# ceph-deploy install {ceph-node}[{ceph-node} ...]
```

例如上面第5步,我们把{ceph-node}[{ceph-node}...]替换成节点名。

```
root@node1:~# ceph-deploy install node5
```

6）添加mon。

```
root@node1:~# ceph-deploy mon create {host-name [host-name]...}
```

例如上面第6步，我们把{host-name[host-name]...}替换成节点名。

```
root@node1t:~# ceph-deploy mon create node5
```

7）验证是否成功。

```
root@node1:~# ceph -
```

```
s
    cluster 67d997c9-dc13-4edf-a35f-76fd693aa118
     health HEALTH_OK
     monmap e1: 3 mons at
{node1=192.168.1.2:6789/0,node2=192.168.1.3:6789/0,
node5=192.168.1.5:6789/0}
            election epoch 28, quorum 0,1,2 node1,node2,node5
     osdmap e18223: 12 osds: 12 up, 12 in
      pgmap v3331652: 768 pgs, 3 pools, 954 GB data, 135
kobjects
            2982 GB used, 2357 GB / 5339 GB avail
                 768 active+clean
```

4.删除MON（Monitor）节点

MON节点是整个集群的协调中心，在非必要情况下，尽量减少对MON的调整。以下我们以3个MON节点的环境为示例。

以下是删除MON节点的步骤。

1）关闭需要被删除Mon节点的ceph-mon进程，观察集群是否选举出新的leader MON，待新的leader生成之后，

进入下一步骤。（以下是重新选择leader MON过程的日志显示及MON map结构）

```
[root@node1 ~]# ceph -
```

```
w …
```

```
..
2016-09-02 16:03:48.141968 mon.2 [INF] mon.node3 calling new
monitor election
2016-09-02 16:03:48.144820 mon.1 [INF] mon.node2 calling new
monitor election
2016-09-02 16:03:53.147912 mon.1 [INF] mon.node2@1 won leader
election with quorum 1,2   ### node2 成为新的
```

```
leader mon ##…
```

```
..<--!其他日志省略
```

```
-->
[root@node1 ~]# ceph mon_status | jq .      ##查看
```

MON节点的状态，

jq是一个格式化显示工具，请预先安装，下同

```
##
{
  "name": "node3",  ## mon leader  ##
  "rank": 2,    ## mon leader 的
```

```
rank id ##
  "state": "peon",
  "election_epoch": 372,   ##选举的版本数
```

```
 ##
  "quorum": [    ## 选举的范围，因为有
```

mon已关闭，此关闭的

Mon节点就被踢出选举范围

```
##
    1,
    2
  ],
  "outside_quorum": [],
  "extra_probe_peers": [],
```

```
  "sync_provider": [],
  "monmap": {  ## 关闭的
```

mon节点并未完全被删除，所以此时关闭的

Mon节点还在

Mon map里

```
##
    "epoch": 1,
    "fsid": "e4fa64c1-a6da-4f8f-be08-70544636a01b",
    "modified": "2016-05-18 12:23:32.728762",
    "created": "2016-05-18 12:23:32.728762",
    "mons": [
      {
        "rank": 0,
        "name": "node1",  ## mon node1 ##
        "addr": "192.168.122.11:6789/0"
      },
```

```
        {
          "rank": 1,
          "name": "node2",     ## mon node2 ##
          "addr": "192.168.122.12:6789/0"
        },
        {
          "rank": 2,
          "name": "node3",    ## mon node3 ##
          "addr": "192.168.122.13:6789/0"
        }
      ]
    }
}
```

2）从集群环境里删除关闭的MON节点。

```
[root@node1 ~]# ceph mon remove  node1   ## 删除关闭的
```

```
mon 节点
```

```
 ##
Error EINVAL: removing mon.node1 at 192.168.122.11:6789/0, there
will be 2 monitors    ## 提示
```

mon节点已经被删除，剩余

2个

mon ##
[root@node1 ~]# ceph mon_status | jq .   ## 查看删除之后

mon 状态

 ##
{
  "name": "node2",
  "rank": 0,   ## rank 编号重新编排

,此时

0为新的

leader的

```
rank id  ##
  "state": "leader",
  "election_epoch": 374,
  "quorum": [
    0,
    1
  ],
  "outside_quorum": [],
  "extra_probe_peers": [],
  "sync_provider": [],
  "monmap": {  ## 此时
```

mon map已经更新，

mon map版本号自增

1，删除

mon节点的信息已经不

存在

##

```
    "epoch": 2,
    "fsid": "e4fa64c1-a6da-4f8f-be08-70544636a01b",
    "modified": "2016-09-02 16:29:46.843308",
    "created": "2016-05-18 12:23:32.728762",
    "mons": [
      {
        "rank": 0,   ## rank 编号重新编排
```

```
  ##
        "name": "node2",
        "addr": "192.168.122.12:6789/0"
      },
      {
        "rank": 1,
        "name": "node3",
        "addr": "192.168.122.13:6789/0"
      }
    ]
  }
}
```

3）更新配置文件，删除踢出的mon的相关配置内容，并推送新的配置文件到所有节点。

4）使用ceph-deploy工具删除MON节点。

```
root@localhost:~# ceph-deploy mon destroy [hostname]
```

 提示

1）提前检查被添加节点的时间、防火墙。

2）对网络要求一般，因为Ceph源在外国有时候会被屏蔽，解决办法多尝试机器或者代理。

3）添加OSD的时候会造成Ceph发生remapped等状态，产生大量I/O操作。建议在部署环境的时候提前规划好Crush Map和隔离域，使对业务的影响降到最低，保证I/O操作只影响某些隔离域。

4）群集的扩容，可以参看第10章的CRUSH设计案例，根据硬件规模设计合理的CRUSH架构，为扩容打好坚实的基础。

参考资料：

- http://docs.ceph.com/docs/master/rados/deployment/ceph-deploy-osd/ 。

- http://docs.ceph.com/docs/master/rados/deployment/ceph-deploy-mon/ 。

13.1.2 集群维护

集群维护是重中之重，下面分为4个部分来说一下集群维护。

1.物理机关机离线维护

日常情况下，会遇上单个存储+计算节点需要做关机离线维护的情形，为降低对上层虚拟机服务的影响，对整个操作流程进行以下梳理。

1）在服务可正常使用的情况下，将需要维护的节点上的虚拟机做在线热迁移，清空节点的资源占用。

2）在mon或者其他有admin权限的节点上设置osd的状态锁定机制，使用如下命令。

```
root@localhost:~# for i in noout nobackfill norecover;do
ceph osd set $i;done
```

3）关闭节点进行维护。

4）维护完毕，启动节点。

5）在集群负载比较低的情况下，执行下面操作，开始底层数据同步。

```
root@localhost:~# for i in noout nobackfill norecover;do
ceph osd unset $i;done
```

6）health ok之后，将之前迁移出去的主机回迁。

2.Ceph网络配置

利用Ceph-deploy安装Ceph集群时，默认是不区分Public Network和Cluster Network的，应该单独设置Public Network和Cluster Network。其架构如图13-1所示。

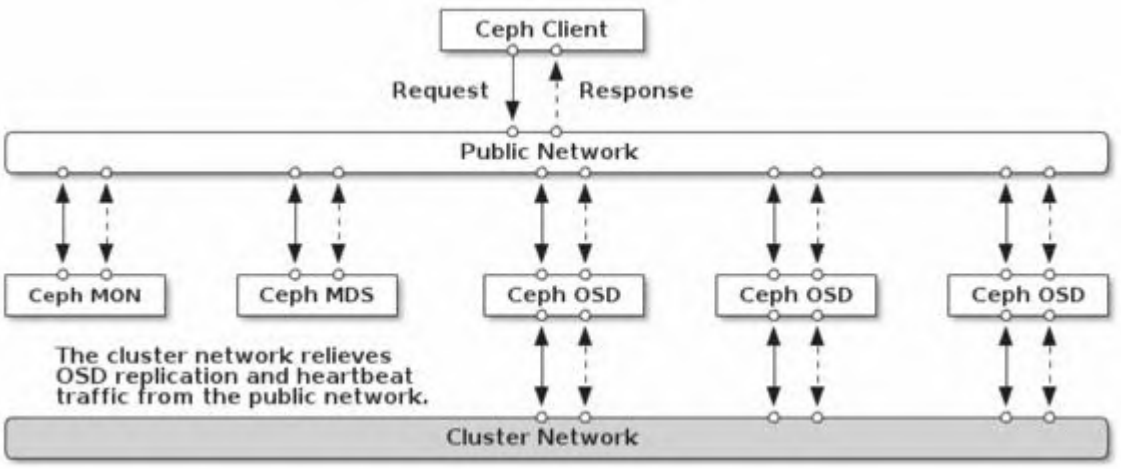

图13-1　架构

这样做带来的好处如下。

（1）性能提升

此时OSD复制PG、心跳检测、OSD故障恢复和数据Rebalancing等都通过Cluster Network，大大降低了对Public Network的依赖；Public Network可以专注于Monitor、MDS与OSD间的通信、Client请求等。

（2）安全提升

利用私有网络管理OSD间的数据与通信，屏蔽了来自Public Network的很多外部恶意攻击。

可在部署的时候添加以下配置。

```
public network = {ip-address}/{netmask}
cluster network={ip-addesss}/{netmask}
<!-----以上两个网络是新增部分，默认只是添加



public network，一般生产都是定义两个网络，集群网络和数据网络分开



-------->
```

3.利用udev增强对Ceph存储设备的有效管理

默认情况下，磁盘可以使用by-id/by-partlabel/by-parttypeuuid/by-partuuid/by-

path/by-uuid等多种形式的名称对磁盘设备进行管理，但是在Ceph中，如果磁盘数量过多，加上为了更好区别每一个OSD对应的磁盘分区用途（比如filestore或journal），同时确保物理磁盘发生变更（故障盘替换后）后对应的名称不变，对OSD对应的磁盘设备命名提出新的管理需求。

本例使用udev的方式，将磁盘按照osd[N]的方式进行命名，比如/dev/osd5_filestore_1表示osd5的第一个filestore分区，/dev/osd5_journal_5表示osd5的第一个journal分区（表示该磁盘用于osd.5的filestore），以/dev/sdd为例。

1）查看设备信息。

```
  root@loaclhost:~# udevadm info --query=all --name=/dev/sdd
P:
/devices/pci0000:00/0000:00:0d.0/ata6/host5/target5:0:0/5:0:
0:0/block/sdd
N: sdd
S: disk/by-id/ata-VBOX_HARDDISK_VB98806c01-1fe3494a
S: disk/by-id/scsi-SATA_VBOX_HARDDISK_VB98806c01-1fe3494a
S: disk/by-path/pci-0000:00:0d.0-scsi-0:0:0:0
```

```
S: osd5data
E: DEVLINKS=/dev/disk/by-id/ata-VBOX_HARDDISK_VB98806c01-
1fe3494a /dev/disk/by-id/scsi-SATA_VBOX_HARDDISK_VB98806c01-
1fe3494a /dev/disk/by-path/pci-0000:00:0d.0-scsi-0:0:0:0
/dev/osd5data
E: DEVNAME=/dev/sdd
E:DEVPATH=/devices/pci0000:00/0000:00:0d.0/ata6/host5/target
5:0:0/5:0:0:0/block/sdd #这个表示物理设备的系统




ID
```

2）选取设备的标识码，示例中用的是DEVPATH（DEVPATH=/devices/pci0000:00/0000:00:0d.0/ata6/host5/target）。

3）编写udev rules规则文件。

```
 root@localhost:~# cat /etc/udev/rules.d/20-persistent-
disk.rules
KERNEL=="sd?", SUBSYSTEM=="block",
DEVPATH=="*/devices/pci0000:00/0000:00:0d.0/ata6/host5/targe
t5:0:0/5:0:0:0*", SYMLINK+="osd5",
GOTO="END_20_PERSISTENT_DISK"
KERNEL=="sd?*", ATTR{partition}=="1", SUBSYSTEM=="block",
DEVPATH=="*/devices/pci0000:00/0000:00:0d.0/ata6/host5/targe
t5:0:0/5:0:0:0*", SYMLINK+="osd5_filestore_%n"
KERNEL=="sd?*", ATTR{partition}=="2", SUBSYSTEM=="block",
DEVPATH=="*/devices/pci0000:00/0000:00:0d.0/ata6/host5/targe
```

```
t5:0:0/5:0:0:0*", SYMLINK+="osd5_journal_%n"
LABEL="END_20_PERSISTENT_DISK"
```

4）执行以下命令，向内核发送event事件，触发udev rules的执行（类似模拟块设备的热插拔）。

```
root@localhost:~# udevadm trigger --subsystem-match=block --
action=add
```

5）检查最终效果，如图13-2所示。

图13-2　检查效果

4.Cgroups在Ceph中的应用

Cgroups是Control Groups的缩写，是Linux内核提供的一种可以限制、记录、隔离进程组（Process Groups）所使用的物理资源（如CPU、Memory和IO等）

的机制。最初由Google公司的工程师提出，后来被整合进Linux内核。Cgroups也是LXC为实现虚拟化所使用的资源管理手段，可以说没有Cgroups就没有LXC。

1）软件包安装。

```
root@localhost:~# apt-get install cgroup-bin
```

2）节点CPU和内存节点查看。

```
root@localhost:~# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Stepping:              2
CPU MHz:               2394.097
BogoMIPS:              4788.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
```

```
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15
```

3）准备配置文件。

```
root@localhost:~# vi /etc/cgconfig.conf
  ## 注意









mon和









osd部分根据实际情况进行增加









##
  ## 以下为配置文件
```

```
##
mount {
cpu = /sys/fs/cgroup/cpu;
cpuacct = /sys/fs/cgroup/cpuacct;
cpuset = /sys/fs/cgroup/cpu;
devices = /sys/fs/cgroup/devices;
memory = /sys/fs/cgroup/memory;
freezer = /sys/fs/cgroup/freezer;
}
group mon {
perm {
   task {
      uid = root;
      gid = root;
   }
   admin {
      uid = root;
      gid = root;
   }
   }
cpu {
   cpuset.cpus = 0;
   cpuset.mems = 0;
   }
memory {
   memory.limit_in_bytes = 2g;
   memory.memsw.limit_in_bytes = 3g;
   memory.soft_limit_in_bytes = 1g;
memory.swappiness = 0;
memory.oom_control = 1;
   memory.failcnt = 1;
   }
}
group osd0 {
perm {
   task {
      uid = root;
      gid = root;
   }
   admin {
      uid = root;
      gid = root;
```

```
    }
    }
cpu {
    cpuset.cpus = 1;    #绑定到指定的
```

```
cpu
    cpuset.mems = 0,1;
    }
memory {
    memory.limit_in_bytes = 2g;   #物理内存限制为
```

```
2G
    memory.memsw.limit_in_bytes = 3g;   #swap虚拟内存限制为
```

```
3G
    memory.soft_limit_in_bytes = 1g; #当物理内存成果
```

1G将触发内存回收

memory.swappiness = 0; #优先使用物理内存

memory.oom_control = 1; #当内存到达上限，不关闭进程，只将对应进程进入

sleep状态

memory.failcnt = 1; #报告当内存到达

```
    memory.limit_in_bytes次数
```

```
    }
}
```

4）修改服务配置文件。

①OSD服务配置。修改/etc/init/ceph-osd.conf配置如下。

```
instance ${cluster:-ceph}/$id
export cluster
export id
exec cgexec -
```

```
g cpu,memory:osd”
```

```
$id"
```

```
 /usr/bin/ceph-osd --cluster="${cluster:-ceph}" -i "$id" -
```

```
f
```

②Mon服务配置。修改/etc/init/ceph-mon.conf配置如下。

```
exec cgexec -
```

```
g cpu,memory:mon /usr/bin/ceph-mon --cluster="${cluster:-
ceph}" -i "$id" -f
```

```
post-stop script
    # Cleanup socket in case of segfault
    rm -f "/var/run/ceph/ceph-mon.$id.asok"
end script
```

5）Ceph服务管理。

①CentOS服务管理有两种方式：一种是/etc/init.d方式，另一种是service方式。

- /etc/init.d/方式。

```
root@localhost:~# /etc/init.d/ceph [options] [command]
[daemons]
```

例如上步操作，我们可以把[options][command][daemons]替换为start mon等。

```
root@localhost:~# /etc/init.d/ceph start mon
root@localhost:~# /etc/init.d/ceph start osd.0
root@localhost:~# /etc/init.d/ceph start
```

- service方式。

```
root@localhost:~# service ceph [command] [ daemons]
```

例如上步操作，我们可以把[command][daemons]替换为stop mon等。

```
root@localhost:~# service ceph stop mon
root@localhost:~# service ceph stop osd.0
root@localhost:~# service ceph stop
```

②Ubuntu系统服务管理。

```
root@localhost:~# [command] ceph-[Component]id=[daemons]
```

例如上步操作，我们可以把[command]ceph-[Component]id=[daemons]替换为start ceph-osd id=0等。

```
root@localhost:~# start ceph-mon id=node1
root@localhost:~# start mon
root@localhost:~# start ceph-osd id=0
root@localhost:~# stop ceph-osd id=0
root@localhost:~# status ceph-osd id=0
```

6）Ceph集群升级。

升级集群的原则是先中心再两边。即先完成集群协调中心MON节点的升级，然后再处理OSD。

①升级MON。

```
root@localhost:~# apt-get update ceph -
```

```
y
root@localhost:~# restart ceph-mon id=x
```

②升级OSD。

```
root@localhost:~# apt-get update ceph -
```

```
y
root@localhost:~# restart ceph-osd id=x
```

提示

1）网卡的选择要根据存储节点的磁盘吞吐量来计算。

2）即使配置了Public Network和Cluster Network，OSD心跳检测也依然会在两个网络之间存在。

3）Cgroup限制内存有可能会导致OOM死掉。

集群升级需谨慎。

13.1.3　集群监控

1.Ceph集群监控命令

1）检查集群健康状态。

```
root@localhost:~# ceph health
HEALTH_OK
```

2）监视集群事件。

```
root@localhost:~# ceph -
```

```
w    ##此命令只是交互监听集群改变的信息
```

```
 ##
    cluster 67d997c9-dc13-4edf-a35f-76fd693aa118
     health HEALTH_OK
     monmap e1: 2 mons at
```

```
{node1=192.168.1.2:6789/0,node2=192.168.1.3:6789/0}
            election epoch 28, quorum 0,1 node1,node2
     osdmap e18223: 12 osds: 12 up, 12 in
      pgmap v3331652: 768 pgs, 3 pools, 954 GB data, 135
kobjects
            2982 GB used, 2357 GB / 5339 GB avail
                 768 active+clean
2016-01-12 11:34:44.687218 mon.0 [INF] pgmap v3371554: 768 pgs:
768 active+clean; 954 GB data, 3043 GB used, 2296 GB / 5339 GB
avail; 0 B/s rd, 10919 B/s wr, 5 op/s
2016-01-12 11:34:45.693991 mon.0 [INF] pgmap v3371555: 768 pgs:
768 active+clean; 954 GB data, 3043 GB used, 2296 GB / 5339 GB
avail; 0 B/s rd, 27821 B/s wr, 13 op/s
```

监听集群事件详细参数，如下。

- --watch-debug：监听“调试”等级事件。
- --watch-info：监听“信息”等级事件。
- --watch-sec：监听“安全”等级事件。
- --watch-warn：监听“警告”等级事件。
- --watch-error：监听“错误”等级事件。

例如，监听“调试”等级的事件，可以使用ceph--watch-debug查看。

```
[root@node1 ~]# ceph   --watch-debug
    cluster e4fa64c1-a6da-4f8f-be08-70544636a01b
     health HEALTH_WARN
            pool default.rgw.buckets.data has many more objects
per pg than average (too few pgs?)
            mon.node2 low disk space
            mon.node3 low disk space
     monmap e2: 2 mons at
{node2=192.168.122.12:6789/0,node3=192.168.122.13:6789/0}
            election epoch 374, quorum 0,1 node2,node3
     osdmap e159: 4 osds: 4 up, 4 in
            flags sortbitwise
      pgmap v129773: 104 pgs, 13 pools, 486 MB data, 14861
objects
            1845 MB used, 378 GB / 379 GB avail
                 104 active+clean
2016-09-02 17:03:21.134427 mon.1 [DBG] from='client.?
192.168.122.12:0/3380896618' entity='client.admin' cmd=
[{"prefix": "status", "format": "json"}]: dispatch  ##此处就是现实出
```

debug的信息

##……

3）集群使用统计。

```
root@compute01:~# ceph df
GLOBAL:
    SIZE       AVAIL      RAW USED     %RAW USED
    5339G      2296G         3043G         57.00
POOLS:
    NAME         ID     USED       %USED     MAX AVAIL
OBJECTS
    volumes      10       234G      4.38          639G
41004
    vms          11     30225M      0.55          639G
4197
    images       12       690G     12.94          639G
96182
```

2.Ceph集群监控组件

（1）MON监控

Monitor负责整个Ceph集群中所有OSD状态的发现与记录，共同形成Cluster Map的Master版本，然后扩散至全体OSD以及Client。OSD使用Cluster Map进行数据的维护，而Client使用Cluster Map进行数据的寻址。

1）系统状态检测和Map维护。

OSD和Monitor之间相互传输节点状态信息，共同得出系统的总体工作状态，并形成一个全局系统状态记录数据结构，Ceph叫作Cluster Map。

```
root@localhost:~# ceph mon stat
e1: 2 mons at
{node1=192.168.1.2:6789/0,node2=192.168.1.3:6789/0}
            election epoch 28, quorum 0,1 node1,node2
```

2）MON法定票数。

一般来说，在实际运行中，Ceph MOM的个数是2n+1（n≥0）个，在线上至少3个，只要正常的节点数≥n+1，Ceph的Paxos算法能保证系统的正常运行。所以，对于3个节点，同时只能挂掉一个。一般来说，同时挂掉2个节点的概率比较小，但是万一挂掉2个呢？

如果Ceph的Monitor节点超过半数挂掉，Paxos算法就无法正常进行仲裁（quorum），此时，Ceph集群会阻塞对集群的操作，直到超过半数的Monitor节点恢复。

```
root@localhost:~# ceph quorum_status
{"election_epoch":28,"quorum":[0,1],"quorum_names":
```

```
["node1","node2"],"quorum_leader_name":"node1","monmap":
{"epoch":1,"fsid":"67d997c9-dc13-4edf-a35f-
76fd693aa118","modified":"0.000000","created":"0.000000","mons":
[{"rank":0,"name":"node1","addr":"192.168.1.2:6789\/0"},
{"rank":1,"name":"node2","addr":"192.168.1.3:6789\/0"}]}}
```

（2）OSD监控

1）监控OSD，可以显示OSD的位置、权重、状态以及权重的百分比。

```
root@localhost:~# ceph osd tree
ID WEIGHT  TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 5.15991 root default
-2 1.28998     host node1
 7 0.42999         osd.1           up  1.00000          1.00000
 6 0.42999         osd.2           up  1.00000          1.00000
 8 0.42999         osd.3           up  1.00000          1.00000
-3 1.28998     host node2
 2 0.42999         osd.4           up  1.00000          1.00000
 0 0.42999         osd.5           up  1.00000          1.00000
 1 0.42999         osd.6           up  1.00000          1.00000
-4 1.28998     host node3
11 0.42999         osd.7           up  1.00000          1.00000
10 0.42999         osd.8           up  1.00000          1.00000
 9 0.42999         osd.9           up  1.00000          1.00000
-5 1.28998     host node4
 5 0.42999         osd.10          up  1.00000          1.00000
 3 0.42999         osd.11          up  1.00000          1.00000
 4 0.42999         osd.12          up  1.00000          1.00000
```

2）OSD共有4种状态，如下。

• Up且in：说明该OSD正常运行，且已经承载至少一个PG的数据。正常状态。

• Up且out：说明该OSD正常运行，但并未承载任何PG。

• Down且in：说明该OSD发生异常，但仍然承载着至少一个PG，其中仍然存储着数据。异常状态。

• Down且out：说明该OSD已经彻底发生故障，且已经不再承载任何PG。

（3）PG监控

PG常见状态如下。

• Peering：在一个Acting Set，数据和元数据同步协商达到一致状态的过程。

• Active/Clean：Peering正常完成以后的状态，Primary PG和Primary OSD以及复制PG和OSD可以被正常读

写，集群最健康的状态。

• Degraded：OSD关闭，在这个OSD上的全部PG处于降级状态，需要尽快恢复防止数。

• Recovering：当某OSD挂了（down）时，其内容版本会落后于PG内的其他副本；当它重新加入集群时，PG内容必须更新以反映当前状态；在此期间，OSD就处在恢复状态。

• Backfilling：当集群有新的OSD加入时，会被分配到若干PG，此时数据向新PG的拷贝的状态即为回填状态。

• Remapped：PG重新映射，数据还未完成迁移所处的状态。

• Stale：PG处于一种未知状态，MON不能理获取PG的状态信息。

查看PG状态。所有PG都是active+clean状态，说明数据没有问题。

```
root@localhost:~# ceph pg stat
v3372549: 768 pgs: 768 active+clean; 954 GB data, 3043 GB used,
2296 GB / 5339 GB avail;
```

3.Ceph调试

（1）Debug日志等级

每一个子系统（mon，osd，mds等）都有输出日志、内存日志的等级。大家可以为每个子系统的日志（包括输出日志和内存日志）设置不同的等级。Ceph的日志等级设置范围为1~20，1信息量最简洁，20信息最冗杂。通常不需要把内存日志重定向到输出日志，除非出现以下的情况。

- 出现致命的错误信息。
- 源码级别的调试。

• 查看请求详情，具体查阅document on admin socket。

输出日志和内存日志可以设置成一个，如‘debugms=5’，ceph会把输出日志和内存日志都设置成为5。当然，这两者也可以单独设置，格式如下。

```
debug {subsystem} = {log-level}/{memory-level}
#for example
debug mds log = 1/20
```

总之，日志级别越高，输出的信息越多，但同时IO负担也就越大。在生产环境中，推荐使用默认的日志级别。需要排错时，可适时调节日志等级。

注意

Ceph系统默认的log级别参考文档地址为 http://ceph.com/docs/master/rados/troubleshooting/log-and-debug/#subsystem-log-and-debug-setting 。

（2）开启Debug方式

1）配置文件指定。

```
[global]
debug ms = 1/5
[mon]
debug mon = 20
debug paxos = 1/5
debug auth = 2
[osd]
debug osd = 1/5
debug filestore = 1/5
debug journal = 1
debug monc = 5/20
```

2）在线修改log level。

```
root@localhost:~# ceph tell osd.0 injectargs '

--debug-osd 0/5'
```

```
root@localhost:~# ceph --admin-daemon /var/run/ceph/ceph-
osd.0.asok config set debug_osd 0/5
```

3）查看结果。

```
root@ubuntu-ceph-06:~# ceph --admin-daemon /var/run/ceph/ceph-
osd.0.asok config show | grep -i debug_osd
```

4.Ceph集群监控软件

目前主流的Ceph开源监控软件有：Calamari、VSM、Inkscope、Ceph-Dash和Zabbix等，下面简单介绍各个开源组件。Zabbix介绍的资料非常多，这里只简单介绍前几种。

（1）Calamari

Calamari对外提供了十分漂亮的Web管理和监控界面（如图13-3所示），以及一套改进的REST API接口（不同于Ceph自身的REST API），在一定程度上简化了Ceph的管理。最初Calamari是作为Inktank公司的Ceph企业级商业产品来销售的，Red Hat公司2015年收购Inktank后，为了

更好地推动Ceph的发展，对外宣布Calamari开源，秉承开源开放精神的Red Hat着实又做了一件非常有意义的事情。

图13-3　Calamari界面

优点：

- 轻量级；
- 官方化；
- 界面友好。

缺点：

- 不易安装；
- 管理功能滞后。

（2）Virtual Storage Manager

Virtual Storage Manager（VSM）是Intel公司研发并且开源的一款Ceph集群管理和监控软件，简化了一些Ceph集群部署的步骤，可以简单地通过Web页面来操作，如图13-4所示。

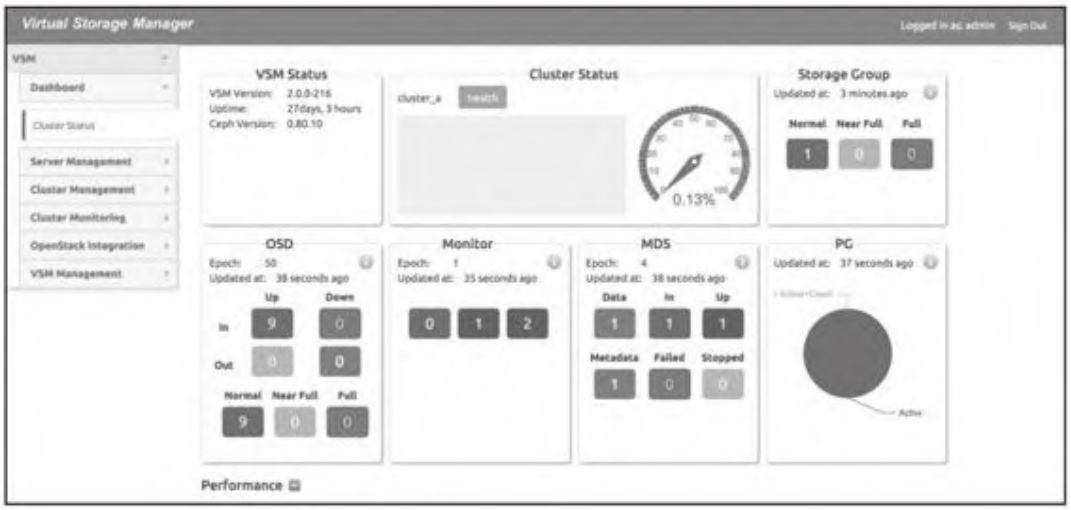

图13-4　Virtual Storage Manager界面

优点：

- 管理功能好；
- 界面友好；
- 可以利用它来部署Ceph和监控Ceph。

缺点：

- 非官方；
- 依赖OpenStack某些包。

（3）Inkscope

Inkscope是一个Ceph的管理和监控系统，依赖于Ceph提供的API，使用MongoDB来存储实时的监控数据和历史信息，如图13-5所示。

图13-5　Inkscope界面

优点：

- 易部署；
- 轻量级。

缺点：

- 监控选项少；
- 缺乏Ceph管理功能。

（4）Ceph-Dash

Ceph-Dash是用Python语言开发的一个Ceph的监控面板，用来监控Ceph的运行状态。同时提供REST API来访问状态数据，如图13-6所示。

图13-6　Ceph-Dash界面

5.OSD磁盘日常监控参考项

对Ceph OSD磁盘，一定做好定期的性能数据采集和通电时长管理，长期的数据积累对磁盘的性能与生命周期管

理会有一定帮助，同时也能确保整个集群性能的稳定。

（1）磁盘碎片管理

1）查看磁盘碎片。

```
root@localhost:~# xfs_db -c frag -r /dev/sdb1
actual 981, ideal 964, fragmentation factor 1.73%
```

2）整理碎片。

```
root@localhost:~# xfs_fsr /dev/sdb1
```

（2）OSD磁盘性能

查看当前Ceph延迟情况，展示各OSD的延迟，找出性能瓶颈。

```
 root@localhost:~# ceph osd perf
osdid fs_commit_latency(ms) fs_apply_latency(ms)
    0                   0                   5
    1                 665                   7
    2                   0                   1
## fs_commit_latency写入延迟
```

```
## fs_apply_latency读取延迟
```

可以看到，这个osd 1的commit_latency和apply_latency都增加了很多（从正常的几ms升高到几百ms），其中commit_latency是写journal完成的时间，apply_latency是写到osd的buffer cache里完成的时间。

这个osd在重启后进行恢复的时候写journal延迟升高，导致了上层写性能的下降，那就从这个点入手去进行排查。

（3）通电时长

```
root@localhost:~# smartctl -A /dev/sda
root@localhost:~# smartctl 5.41 2011-06-09 r3365 [x86_64-linux-3.8.0-44-generic] (local build)
Copyright (C) 2002-11 by Bruce Allen,
```

```
http://smartmontools.sourceforge.net
Current Drive Temperature:     27 C
Drive Trip Temperature:        65 C
Elements in grown defect list: 0
Vendor (Seagate) cache information
  Blocks sent to initiator = 0
Vendor (Seagate/Hitachi) factory information
  number of hours powered up = 53413.70
  number of minutes until next internal SMART test = 7
```

（4）Ceph集群整体IO性能

可以通过ceph -w交互打印集群状态信息，其中就有集群的容量及使用、IO读写信息。以下是截取的关键信息。

```
2016-09-02 18:32:59.664840 mon.0 [INF] pgmap v130066: 104 pgs:
104 active+clean; 608 MB data, 2229 MB used, 377 GB / 379 GB
avail; 157 kB/s rd, 1013 kB/s wr, 336 op/s  ##
```

其中，读性能为157 kB/s，写性能为1013 kB/s，IOPS为336。

Ceph内置了很多格式化数据的接口，常见的是JSON。JSON化输出的信息更详细。

```
[root@node1 dashboard]# ceph  status -f json | jq .
{
  "health": {
    "health": {
      "health_services": [
        {
          "mons": [    ## mon 节点信息
```

##

```
            {
              "name": "node2",  ## mon 名字
```

##

```
              "kb_total": 32498172,  ## Mon节点所在分区的容量
```

##

```
              "kb_used": 29814148,   ## 使用容量
```

##

```
              "kb_avail": 2684024,   ## 可用容量
```

```
##
            "avail_percent": 8,     ## 可用百分比
```

```
##
            "last_updated": "2016-09-02 18:30:36.501136",
            "store_stats": {
              "bytes_total": 25245036,
              "bytes_sst": 16987484,
              "bytes_log": 4128768,
              "bytes_misc": 4128784,
              "last_updated": "0.000000"
            },
            "health": "HEALTH_WARN",   ## 当前
```

mon的健康状态

```
 ##
            "health_detail": "low disk space" ## 详细健康信息
```

```
 ##
            },
  …

.     <!—中间忽略疗分输出—

>
    "summary": [     ## 健康状态汇总

 ##
      {
        "severity": "HEALTH_WARN",
        "summary": "pool default.rgw.buckets.data has many more
objects per pg than average (too few pgs?)"
      },
  …
```

. <!—中间忽略疗分输出—

>
  "osdmap": { ## OSD map 信息

 ##
    "osdmap": {
      "epoch": 159, ## OSD 总的版本号

 ##
      "num_osds": 4, ## 当前

OSD数量

```
##
     "num_up_osds": 4, ## UP状态
```

OSD 数量

```
##
     "num_in_osds": 4,  ## In 状态
```

OSD 数量

```
##
     "full": false,
     "nearfull": false,
     "num_remapped_pgs": 0
   }
```

```
 },
 "pgmap": {   ## PG map 信息
```

```
##
   "pgs_by_state": [
     {
       "state_name": "active+clean",
       "count": 104
     }
   ],
   "version": 130028,
   "num_pgs": 104,   ## PG总数
```

```
##
   "data_bytes": 527906065, ## 数据部分的使用量
```

```
##
   "bytes_used": 1993080832, ## 已使用的容量
```

```
##
```

```
    "bytes_avail": 405825323008, ## 可用的总容量
```

```
##
    "bytes_total": 407818403840,  ## 集群
```

总容量

```
##
    "read_bytes_sec": 229760,  ##读性能
```

，即

ceph -

w 显示（下同）

##
"write_bytes_sec": 78592, ##写性能

##
"read_op_per_sec": 205, ##读

IOPS ##
"write_op_per_sec": 280 ##写

IOPS ##

```
  },
  "fsmap": {   ## mds map信息
```

```
 ##
    "epoch": 1,
    "by_rank": []
  }
}
```

格式化的数据更利于集成到现有的监控体系。

 提示

监控Ceph要关注IOPS、吞吐率、OSD Journal延迟、读请求延迟和容量使用率等。

13.2　Ceph常见错误与解决方案

本节将对在Ceph日常维护过程中常见的错误进行简要的总结与分享。

13.2.1 时间问题

在集群环境中，时间一致性的重要性不言而喻。组件之间的通信都以保证时间同步为前提，否则，会导致数据不完整与不一致性。这对于生产系统是致命的。

在Ceph环境里，MON节点之间会进行时间误差（抖动）的检验，抖动时间超过预设的阈值，集群就会触发警告信息。

通过ceph - s获取集群的状态。

```
[root@10-10-10-231 ~]# ceph -s
    cluster 32b545b4-5662-436c-adf7-8c3d87185e4d
     health HEALTH_WARN clock skew detected on mon.10-10-10-
232, mon.10-10-10-233
     monmap e3: 3 mons at {10-10-10-
231=192.168.108.1:6789/0,10-10-10-
232=192.168.108.2:6789/0,10-10-10-233=192.168.108.3:6789/0},
election epoch 36, quorum 0,1,2 10-10-10-231,10-10-10-232,10-
10-10-233
     osdmap e244: 19 osds: 19 up, 19 in
      pgmap v22357000: 676 pgs, 6 pools, 3539 GB data, 881
kobjects
            10161 GB used, 42898 GB / 53060 GB avail
                   1 active+clean+scrubbing+deep
                 675 active+clean
```

```
client io 8146 B/s rd, 1201 kB/s wr, 167 op/s
[root@10-10-10-231 ~]# ceph health detail
mon. 10-10-10-231 addr 10.10.10.231:6789/0 clock skew
0.722957s > max 0.05s (latency 0.00400286s)
```

如上所示，health HEALTH_WARN clock skew detected on mon.10-10-10-232,mon.10-10-10-233，这表示在MON节点的10-10-10-232和10-10-10-233出现过大的时间抖动。

默认的值为0.05，可以通过admin-asok获取当前的设置值。

```
[root@10-10-10-231 ~]# ceph daemon   mon.10-10-10-231  config
show | grep mon_clock_drift_allowed
  "mon_clock_drift_allowed": "0.05",
```

解决方案：

1）如果对于时间精确度要求非常高，可以在集群内设置NTP服务，让各节点向本地时钟服务进行同步。（如何设置NTP服务在此不再详细说明）

2）精度要求级别低，可以直接在每个节点上设置，向公网时间源进行时间同步的计划任务。如下所示，设置每隔10分钟向公网进行同步（前提条件是各节点保证在同一时区）。

```
[root@10-10-10-231 registry]# crontab  -l
*/10 * * * * /usr/sbin/ntpdate   0.centos.pool.ntp.org
```

3）增加时间抖动报警的阈值。在线调整可以通过tell工具来实现，为了保存设置，最好把调整写入到配置文件里，并同步到所有节点。（即把mon clock drift allowed=0.5写入ceph.conf的default选项下）。

```
[root@10-10-10-231 ~]# ceph tell mon.\*   injectargs "--
mon_clock_drift_allowed 0.5"
mon.10-10-10-231: injectargs:mon_clock_drift_allowed = '0.5'
mon.10-10-10-232: injectargs:mon_clock_drift_allowed = '0.5'
mon.10-10-10-233: injectargs:mon_clock_drift_allowed = '0.5'
```

 注意

根据不少生产环境反映，调整抖动时间会引发一些未知的问题，在此建议设置时间强同步更为妥当。

13.2.2 副本数问题

新手在初次搭建Ceph集群环境时，限于节点数和OSD资源的限制，集群状态未能达到完全收敛状态，即所有PG保持active+clean。排除OSD故障原因，此类问题，主要还是Crush Rule耍了小把戏，只要我们看清本质，Ceph就服帖了。

默认的Crush Rule设置的隔离是Host级，即多副本（如三副本）情况下，每一副本都必须分布在不同的节点上。如此一来，当节点数不满足大于或等于副本数时，PG的状态自然就不能是active+clean，而会显示为“degraded”（降级）状态。

解决方案：

要解决这个问题，可以从两方面入手：副本数和Crush Rule。

（1）通过副本数解决

1）默认的配置多副本模式的集群的副本数是3，测试环境可以把副本数设为1。以下是查看默认创建的pool副本数。

```
[root@host-192-168-0-16 ~ ]# ceph daemon  osd.0  config  show |  grep default_size
    "osd_pool_default_size": "3",
[root@ host-192-168-0-16 ~]# ceph osd dump | grep size
pool 0 'rbd' replicated size 3 min_size 2 crush_ruleset 0 object_hash rjenkins pg_num 64 pgp_num 64 last_change 32 flags hashpspool stripe_width 0
```

2）通过tell在线修pool的副本数，并修改配置文件且同步到所有节点。保险起见，把MON和OSD关于副本数的选项都进行修改。

```
[root@host-192-168-0-18 ~]#  ceph tell  mon.\*  injectargs "--osd_pool_default_size 1"
mon.host-192-168-0-16: injectargs:osd_pool_default_size = '1'
mon.host-192-168-0-17: injectargs:osd_pool_default_size = '1'
mon.host-192-168-0-18: injectargs:osd_pool_default_size = '1'
 [root@ host-192-168-0-16 ~]#  ceph tell osd.*  injectargs "--osd_pool_default_size 1"
osd.0: osd_pool_default_size = '1'
osd.1: osd_pool_default_size = '1'
osd.2: osd_pool_default_size = '1'
```

3）此时，我们再创建Pool，然后查看Pool的副本数，并且查看PG的状态。此时创建的PG都是“active+clean”状态。

```
[root@host-192-168-0-18 ~]# ceph osd pool create   test 8 8
pool 'test' created
[root@host-192-168-0-18 ~]# ceph osd dump | grep test
pool 5 'test' replicated size 1 min_size 1 crush_ruleset 0
object_hash rjenkins pg_num 8 pgp_num 8 last_change 52 flags
hashpspool stripe_width 0
<!--- 忽略其他输入
```

```
>
      pgmap v2837: 8 pgs, 1 pools, 0 bytes data, 0 objects
            104 MB used, 284 GB / 284 GB avail
                   8 active+clean
```

注意

某些版本不能直接通过本地的mon-asok修改所有mon的配置，即ceph tellmon.*injectargs″--

osd_pool_default_size 1″是无效的，需要通过在mon节点本地执行。

（2）修改默认Crush Rule，把隔离域换成OSD

1）获取Crush Map。

```
[root@host-192-168-0-18 ~]# ceph osd getcrushmap -o /tmp/map
got crush map from osdmap epoch 53
```

2）反编译Crush Map。

```
[root@host-192-168-0-18 ~]# crushtool -d /tmp/map -o
/tmp/map.txt
```

3）编辑rule。

```
<忽略其他




>
# rules
```

```
rule replicated_ruleset {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type osd  ####把此外的
```

```
host 改为
```

```
 osd
        step emit
}
<忽略其他
```

```
>
```

4）编译Crush Map。

```
[root@host-192-168-0-18 tmp]# crushtool -c /tmp/map.txt -o
/tmp/map.new
```

```
[root@host-192-168-0-18 tmp]# ll /tmp/map*
-rw-r--r-- 1 root root  649 Jan 14 15:58 /tmp/map
-rw-r--r-- 1 root root  649 Jan 14 16:02 /tmp/map.new    ###
二进制的新
```

```
map
-rw-r--r-- 1 root root 1423 Jan 14 16:02 /tmp/map.txt
```

5）把新的Crush Map应用到群集环境中。

```
[root@host-192-168-0-18 tmp]# ceph osd setcrushmap -i
/tmp/map.new
```

6）设置Crush Map验证新的Crush Rule。

```
[root@host-192-168-0-18 tmp]# ceph osd  crush rule dump
[
    {
        "rule_id": 0,
        "rule_name": "replicated_ruleset",
        "ruleset": 0,
        "type": 1,
        "min_size": 1,
        "max_size": 10,
        "steps": [
            {
                "op": "take",
                "item": -1,
                "item_name": "default"
            },
```

```
            {
                "op": "chooseleaf_firstn",
                "num": 0,
                "type": "osd"                    ###已经修改为
osd ####
            },
            {
                "op": "emit"
            }
        ]
    }
```

7）创建Pool，查看PG的状态。此时PG正常情况下就会显示“clean+active”状态。（此过程忽略）。

 注意

当通过修改故障域来实现完全收敛时，必须保证OSD的数量大于或等于设置的副本数，如此才能保证成功。

13.2.3 PG问题

PG是Ceph里数据管理单元，PG状态反应了数据的真实状态。针对PG的常见问题，下面进行简单总结。

1.PG数量警告

在默认设置下，会有对OSD分布PG数量的警告。获取理相关警告信息，示例如下。

```
[root@host-192-168-0-18 tmp]# ceph daemon   osd.2 config
show | grep pg_warn
    "mon_pg_warn_min_per_osd": "30",      ### OSD最小警告
```

PG数

```
 ###
    "mon_pg_warn_max_per_osd": "300",     ### OSD最大警告
```

PG数

```
 ###
    "mon_pg_warn_max_object_skew": "10",
    "mon_pg_warn_min_objects": "10000",
    "mon_pg_warn_min_pool_objects": "1000",
```

当OSD分布不满足预计的阈值时，Ceph集群状态就会触发告警信息。

```
[root@host-192-168-0-18 ceph]# ceph -s
    cluster 030c5768-12fe-4645-976f-c5153a8add91
     health HEALTH_WARN
       too few PGs per OSD (2 < min 30)
     monmap e2: 3 mons at \ {host-192-168-0-
16=192.168.0.16:6789/0,host-192-168-0-
17=192.168.0.17:6789/0,host-192-168-0-
18=192.168.0.18:6789/0}
            election epoch 3860, quorum 0,1,2 host-192-168-
0-16,host-192-168-0-17,host-192-168-0-18
     osdmap e54: 3 osds: 3 up, 3 in
```

```
    pgmap v2846: 8 pgs, 1 pools, 0 bytes data, 0 objects
          104 MB used, 284 GB / 284 GB avail
                 8 active+clean
```

如上示例，由于创建的Pool只指定了8个PG，导致分布到OSD的PG数小于警告值（30）。因此，可以通过调节警告阈值或者增大Pool的PG数来消除报警。

```
[root@host-192-168-0-16 ceph]# ceph  osd pool set   test
pg_num 40
set pool 5 pg_num to 40
[root@host-192-168-0-16 ceph]# ceph  osd pool set   test
pgp_num 40
set pool 5 pgp_num to 40
```

 注意

修改PG数的时候，同时需要修改PGP的数量，默认保持一致即可。

2. PG数规划

以下对于Pool中PG数的规划进行简要说明。

（1）规划公式及说明

公式：

$$\text{PoolPGCount} = \frac{(\text{TargetPGsPerOSD})*(\text{OSDNumber})*(\text{DataPercent})}{\text{PoolSize}}$$

（2）关于计算结果取整说明

计算的最终结果应该是2的幂次方。采用2的幂次方是为了提高CRUSH算法的效率。计算出结果后，找到与这个结果相邻的两个2次幂数值，如果结果超过较小2次幂数值的25%，则选择较大的2次幂作为最终结果，反之则选择较小的那个2次幂数值。

（3）其他说明

设计计算公式的目的是确保整个集群拥有足够多的PG，从而实现数据均匀分布在各个OSD上，同时能够有效避免在Recovery和Backfill的时候因为PG/OSD比

值过高所造成的问题。如果集群中存在空Pool或者其他非活跃状态下的Pool，这些Pool并不影响现有集群的数据分布，但是这些Pool仍然会消耗集群的内存和CPU资源。

表13-1　参数介绍

名称	说明	备注
Pool PG Count	单个 Pool 的 PG 数量	
Target PGs Per OSD	每个 OSD 的 PGs 数量	① 如果未来集群的 OSD 数量基本不再增长，Target PGs per OSD =100 ② 如果未来集群的 OSD 数量可能增长到目前规模的 2 倍以内，Target PGs per OSD =200 ③ 如果未来集群的 OSD 数量增长规模大于当前 2 倍且小于 3 倍，Target PGs per OSD =300
OSD Number	集群 OSD 的总数，默认来讲是全部 OSD 的数量	如果通过 CRUSH rules 进行了 SSD 和 SATA 设备的规则拆分（比如 SSD 和 SATA 划分成两个 zone），需要单独填写对应 rule 的 OSD 数量
Data Percent	Pool 占用所在 OSD 总容量的百分比（预估值）	
Pool Size	每 个 pool 的 replicas size，默认是 3	如果使用 Erasure Coded Pools 简称 EC pool，Pool Size = K+m

（4）示例

在OpenStack融合架构下，PG规划推荐设置如图13-7所示。

OpenStack　　Add Pool　Generate Commands

Pool Name	Size	OSD #	%Data	Target PGs per OSD	Suggested PG Count
cinder-backup	3	100	25.00	200	2048
cinder-volumes	3	100	53.00	200	4096
ephemeral-vms	3	100	15.00	200	1024
glance-images	3	100	7.00	200	512
Total Data Percentage: 100.00%					PG Total Count: 7680

图13-7　PG规划推荐设置

3. PG阻塞

在OSD接受IO请求的过程中，如果网络出现抖动或者其他因素，导致IO被阻塞，可以通过Ceph health detail会得到类似的信息。

```
32 ops are blocked > 32.768 sec on osd.3
32 ops are blocked > 32.768 sec on osd.3
1 osds have slow requests
```

如上所示，IO请求都在OSD.3上面被阻塞了。通过重启对应的osd进程，一般情况下就能让osd重新接受被阻塞的IO。

```
root@localhost:~# restart ceph-osd id=x
```

4.PG修复

Ceph的scrub/deep scrub是集群自带的自我检测并修复的功能，能够实现对object进行自我修复。下面我们用手动的方式来修复。

1）找到PG所在的位置。

```
[root@host-192-168-0-16 ceph]#  ceph health detail
HEALTH_ERR 1 pgs inconsistent; 2 scrub errors
pg 17.1c1 is active+clean+inconsistent, acting [21,25,30]
2 scrub errors
```

如上所示，PG 17.1c1当前的位置在OSD 21，25，30上面。

你可以直接通过ceph pg repair 17.1c1直接检测并修复，下面我们来挖得更深入些。

2）找到问题所在。

在相应的OSD日志里，找到错误的提示，如grep-Hn'ERR'/var/log/ceph/ceph-osd.21.log。如果日志已经被切割了，用Zgrep代替。Zgrep可以对已经压缩的文件进行正则匹配。

```
log [ERR] : 17.1c1 shard 21: soid
58bcc1c1/rb.0.90213.238e1f29.00000001232d/head//17 digest 0
!= known digest 3062795895
log [ERR] : 17.1c1 shard 25: soid
58bcc1c1/rb.0.90213.238e1f29.00000001232d/head//17 digest 0
!= known digest 3062795895
```

从日志看，object的摘要（digest）本应该为3062795895，但实际却是0。

3）找到对象。

从日志里，我们已经获得如下信息。

- 问题PG是的编号是17.1c1。
- OSD的ID是21。

• object名字是

rb.0.90213.238e1f29.00000001232d。

接下来我们来找到object的具体文件所在位置。

```
[root@host-192-168-0-16 ceph]# find /var/lib/ceph/osd/ceph-
21/current/17.1c1_head/ -name
'rb.0.90213.238e1f29.00000001232d*' -ls
671193536 4096 -rw-r--r-- 1 root root 4194304 Feb 14 01:05
/var/lib/ceph/osd/ceph-
21/current/17.1c1_head/DIR_1/DIR_C/DIR_1/DIR_C/rb.0.90213.23
8e1f29.00000001232d__head_58BCC1C1__11
```

下面我们可以这样做。

• 在每个系统上监测一下这个object的大小。

• 在每个系统上校验一下这个object的MD5值。

4）修复问题。

修复步骤如下。

• 停止有错误object的OSD进程。

- 日志同步到磁盘ceph-osd-i--flush-journal。
- 移除有错误的object。
- 启动OSD进程。
- 调用ceph pg repair 17.1c1。

 注意

①以上操作对于三副本环境是很容易的，因为三副本可以通过简单的object verion比较来解决冲突。但对只有两副本的环境，只能恢复到剩下副本状态，无法保证副本是最新状态。类似选举的过程。

13.2.4　OSD问题

1.日志故障

为保证数据副本的一致性，Ceph采用了日志机制，所有写入OSD的数据会先写入到OSD指定的日志（journal）里（裸盘），然后Ceph会定期把日志刷入OSD的数据盘里。数据的读写流程如图13-8所示。

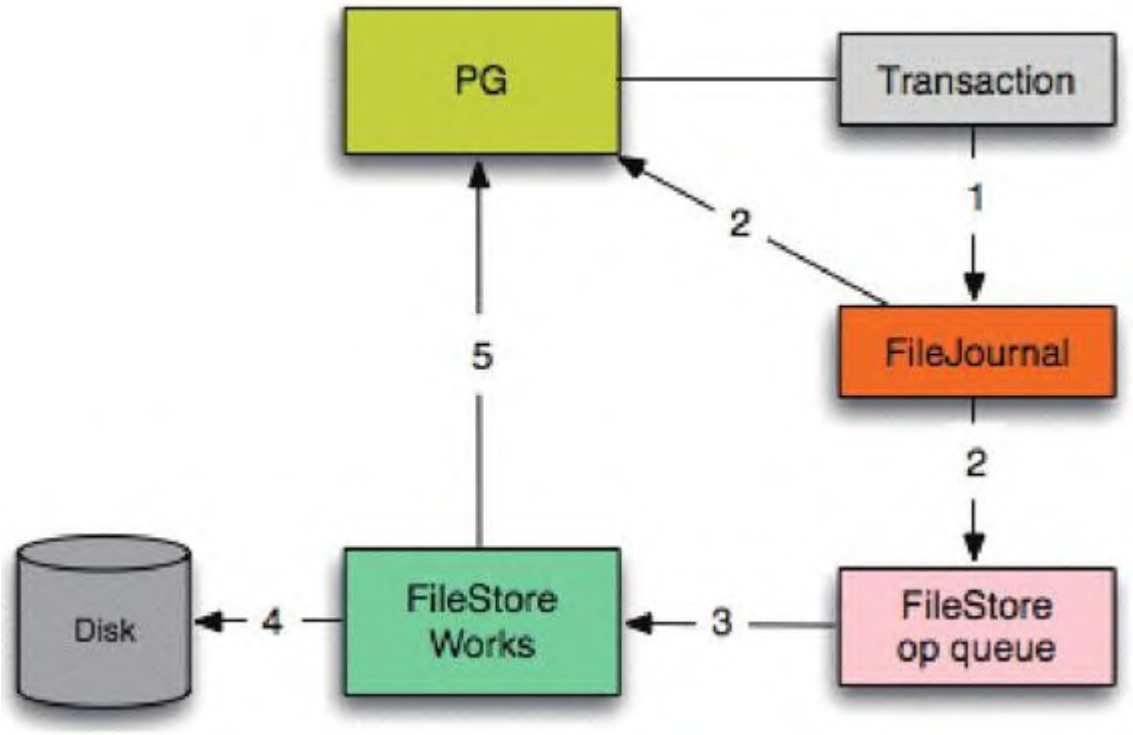

图13-8　数据读写流程

当日志出现问题时，就会引起OSD的故障。如下是常见OSD日志里的日志错误。

```
2016-01-18 13:38:13.827945 7f451a75f880 -1 journal
FileJournal::open: ondisk fsid 00000000-0000-0000-0000-
000000000000 doesn't match expected 519c4bd7-c358-4123-b901-
1ad477a1d985, invalid (someone else's?) journal
2016-01-18 13:38:13.828130 7f451a75f880 -1
filestore(/var/lib/ceph/osd/ceph-1) mount failed to open
journal /var/lib/ceph/osd/ceph-1/journal: (22) Invalid
argument
2016-01-18 13:38:13.838268 7f451a75f880 -1 osd.1 0 OSD:init:
unable to mount object store
2016-01-18 13:38:13.838320 7f451a75f880 -1  ** ERROR: osd
init failed: (22) Invalid argument
```

以上显示，OSD ID为1对应的日志出现了问题。

在OSD部署的时候，如果没有指定单独的日志盘，通常会在硬盘里划分一个分区给日志用。默认配置日志大小为5GB，可以通过“osd_journal_size”（单位为MB）在配置文件里指定日志大小。以下是故障OSD对

应的硬盘的分区情况，第一个分区（/dev/vdb1）即为日志。

```
[root@host-192-168-0-17 ceph-1]# ll /var/lib/ceph/osd/ceph-1/
total 44
-rw-r--r--  1 root root 517 Dec 25 18:04 activate.monmap
-rw-r--r--  1 root root   3 Dec 25 18:04 active
-rw-r--r--  1 root root  37 Dec 25 18:04 ceph_fsid
drwxr-xr-x 29 root root 449 Jan 18 13:42 current
-rw-r--r--  1 root root  37 Dec 25 18:04 fsid
lrwxrwxrwx  1 root root  58 Dec 25 18:04 journal -> /dev/disk/by-partuuid/debb63d8-168b-4ec2-a9c4-d1bd45157ac5
-rw-r--r--  1 root root  37 Dec 25 18:04 journal_uuid
-rw-------  1 root root  56 Dec 25 18:04 keyring
-rw-r--r--  1 root root  21 Dec 25 18:04 magic
-rw-r--r--  1 root root   6 Dec 25 18:04 ready
-rw-r--r--  1 root root   4 Dec 25 18:04 store_version
-rw-r--r--  1 root root  53 Dec 25 18:04 superblock
-rw-r--r--  1 root root   0 Jan 18 13:42 sysvinit
-rw-r--r--  1 root root   2 Dec 25 18:04 whoami
[root@host-192-168-0-17 ceph-1]# ll /dev/disk/by-partuuid/debb63d8-168b-4ec2-a9c4-d1bd45157ac5  ##日志连接文件
```

指向

```
vdb2 ##
lrwxrwxrwx 1 root root 10 Jan 18 13:39 /dev/disk/by-
partuuid/debb63d8-168b-4ec2-a9c4-d1bd45157ac5 -> ../../vdb2
[root@host-192-168-0-17 ceph-1]# parted /dev/vdb print   ##
查看
```

```
vdb的分区表
```

```
##
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 107GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:
Number  Start   End     Size    File system  Name
Flags
 2      1049kB  5369MB  5368MB               ceph journal
 1      5370MB  107GB   102GB   xfs          ceph data
```

在硬盘无硬件故障的前提下，我们可以重构OSD的日志，实现修复OSD，使之能够正常工作。

```
[root@host-192-168-0-17 ceph-1]# ceph-osd  --mkjournal -i 1
### 重构
```

OSD 1 的日志

```
 ##
 HDIO_DRIVE_CMD(identify) failed: Inappropriate ioctl for
device
2016-01-18 13:39:12.566595 7f6482eb8880 -1 journal check:
ondisk fsid 00000000-0000-0000-0000-000000000000 doesn't
match expected 519c4bd7-c358-4123-b901-1ad477a1d985, invalid
(someone else's?) journal
 HDIO_DRIVE_CMD(identify) failed: Inappropriate ioctl for
device
2016-01-18 13:39:12.632742 7f6482eb8880 -1 created new
journal /var/lib/ceph/osd/ceph-1/journal for object store
/var/lib/ceph/osd/ceph-1
```

重新启动OSD进程，观察集群的状态，一切正常故障的OSD应该能够启动。如果未能成功启动，可以查看OSD的日志（此日志为log）。

 提示

基于此，我们可以实现日志的替换。比如为某OSD创建单独的SSD日志来提高性能。

简要步骤如下。

• [root@localhost~]#for i in noout nobackfill norecover;do ceph osd set$i;done。##设置OSD的相关标记，防止出现恢复与重新平衡##

• [root@localhost~]#rm/var/lib/ceph/osd/ceph-1/journal。##删除默认的日志链接文件##

• [root@localhost~]ln-s/dev/sde1/var/lib/ceph/osd/ceph-1/journal。##创建新的日志连接文件，其中sde1是SSD的一分区##

• [root@localhost~]ceph-osd--mkjournal-i 0。##构建新的日志##

2. 更换硬盘

Ceph集群是基于廉价硬件构建的大规模分布式存储。在Ceph集群里，最常见的就是OSD存储硬盘的故障。以下我们就如何更换硬盘进行简要的说明。

因硬盘故障引起OSD进程挂掉时，集群把OSD状态标记“down”，经过超时时间（默认是300s），集群会开始进入恢复模式，数据会进行迁移。下面是PG状态变更时，ceph - s输出的群集信息。

```
2016-01-18 17:23:17.199952 mon.0 [INF] mdsmap e1: 0/0/0 up
2016-01-18 17:23:17.200516 mon.0 [INF] osdmap e153: 3 osds:
2 up, 3 in
2016-01-18 17:23:21.099109 mon.0 [INF] osd.1 out (down for
39.708968)
2016-01-18 17:23:24.495881 mon.0 [INF] osdmap e154: 3 osds:
2 up, 2 in
2016-01-18 17:23:25.672629 mon.0 [INF] pgmap v3286: 32 pgs:
25 active+undersized+degraded, 7 active+clean; 99356 kB
data, 264 MB used, 189 GB / 189 GB avail; 110/296 objects
degraded (37.162%)
2016-01-18 17:23:29.527248 mon.0 [INF] osdmap e155: 3 osds:
2 up, 2 in
2016-01-18 17:23:30.977928 mon.0 [INF] pgmap v3287: 32 pgs:
25 active+undersized+degraded, 7 active+clean; 99356 kB
data, 264 MB used, 189 GB / 189 GB avail; 110/296 objects
degraded (37.162%)
2016-01-18 17:23:34.479672 mon.0 [INF] pgmap v3288: 32 pgs:
```

```
2 active+degraded, 23 activating+degraded, 7 active+clean;
99356 kB data, 264 MB used, 189 GB / 189 GB avail; 28/296
objects degraded (9.459%)
2016-01-18 17:23:35.002622 mon.0 [WRN] reached concerning
levels of available space on local monitor storage (25%
free)
2016-01-18 17:23:35.875659 mon.0 [INF] pgmap v3289: 32 pgs:
18 active+degraded, 12 active+clean, 2
active+recovering+degraded; 99356 kB data, 273 MB used, 189
GB / 189 GB avail; 160/296 objects degraded (54.054%); 2430
kB/s, 4 objects/s recovering    ###正在进行恢复
```

```
 ##
```

1）为了尽量减小因数据迁移造成的集群性能损伤，我们先临时关闭迁移，待到新的OSD添加完成时再启动。

```
[root@host-192-168-0-17 ~]#  for i in noout nobackfill
norecover;do ceph osd unset $i;done
```

2）进入故障硬盘所在的主机，卸载OSD挂载的目录。

```
[root@host-192-168-0-17 ~]# umount /var/lib/ceph/osd/ceph-1
```

3）把故障的OSD从crush map里移除。

```
[root@host-192-168-0-17 ~]# ceph osd crush remove osd.1
```

4）删除OSD.1的认证密钥。

```
[root@host-192-168-0-17 ~]#ceph auth del osd.1
```

5）从集群里把故障OSD删除，此时通过ceph osd tree查看crush架构，此时osd.1应该已经不存在了。

```
[root@host-192-168-0-17 ~]# ceph osd rm osd.1
```

6）在主机上热插拔更换新硬盘后，查看对应的盘符（假如为vdc）；在admin节点，通过ceph-deploy添加新的OSD（带日志分区，有独立日志盘，另行指定）。

```
[root@host-192-168-0-16 ceph]# ceph-deploy --overwrite-conf
osd create --zap-disk     host-192-168-0-17:vdc
[ceph_deploy.conf][DEBUG ] found configuration file at:
/root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked (1.5.24): /usr/bin/ceph-
deploy --overwrite-conf osd create --zap-disk host-192-168-
0-17:vdc
[ceph_deploy.osd][DEBUG ] Preparing cluster ceph disks host-
192-168-0-17:/dev/vdc:
[host-192-168-0-17][DEBUG ] connected to host: host-192-168-
0-17
[host-192-168-0-17][DEBUG ] detect platform information from
remote host
[host-192-168-0-17][DEBUG ] detect machine type
[ceph_deploy.osd][INFO  ] Distro info: CentOS Linux 7.1.1503
Core
[ceph_deploy.osd][DEBUG ] Deploying osd to host-192-168-0-17
[host-192-168-0-17][DEBUG ] write cluster configuration to
/etc/ceph/{cluster}.conf
[host-192-168-0-17][INFO  ] Running command: udevadm trigger
--subsystem-match=block --action=add
[ceph_deploy.osd][DEBUG ] Preparing host host-192-168-0-17
disk /dev/vdc journal None activate True
[host-192-168-0-17][INFO  ] Running command: ceph-disk -v
prepare --zap-disk --fs-type xfs --cluster ceph -- /dev/vdc
[host-192-168-0-17][WARNIN] DEBUG:ceph-disk:Zapping
partition table on /dev/vdc
[host-192-168-0-17][WARNIN] INFO:ceph-disk:Running command:
/usr/sbin/sgdisk --zap-all --clear --mbrtogpt -- /dev/vdc
[host-192-168-0-17][WARNIN] Caution: invalid backup GPT
header, but valid main header; regenerating
[host-192-168-0-17][WARNIN] backup header from main header.
[host-192-168-0-17][WARNIN]
[host-192-168-0-17][DEBUG ]
****************************************************
***************
[host-192-168-0-17][DEBUG ] Caution: Found protective or
hybrid MBR and corrupt GPT. Using GPT, but disk
[host-192-168-0-17][DEBUG ] verification and recovery are
STRONGLY recommended.
[host-192-168-0-17][DEBUG ]
****************************************************
***************
[host-192-168-0-17][DEBUG ] GPT data structures destroyed!
You may now partition the disk using fdisk or
```

```
[host-192-168-0-17][DEBUG ] other utilities.
<!
```

--部分省略

```
>
[host-192-168-0-17][DEBUG ] The operation has completed
successfully.
[host-192-168-0-17][WARNIN] INFO:ceph-disk:calling partx on
zapped device /dev/vdc
[host-192-168-0-17][WARNIN] INFO:ceph-disk:calling partx on
prepared device /dev/vdc
[host-192-168-0-17][WARNIN] INFO:ceph-disk:re-reading known
partitions will display errors
[host-192-168-0-17][WARNIN] INFO:ceph-disk:Running command:
/usr/sbin/partx -a /dev/vdc
[host-192-168-0-17][WARNIN] partx: /dev/vdc: error adding
partitions 1-2
[host-192-168-0-17][INFO  ] Running command: udevadm trigger
--subsystem-match=block --action=add
[host-192-168-0-17][INFO  ] checking OSD status...
[host-192-168-0-17][INFO  ] Running command: ceph --
cluster=ceph osd stat --format=json
[host-192-168-0-17][WARNIN] there is 1 OSD down
[host-192-168-0-17][WARNIN] there is 1 OSD out
[ceph_deploy.osd][DEBUG ] Host host-192-168-0-17 is now
ready for osd use.  ## osd已经加入集群
```

```
##
```

7）在确认进程正常启动，并且OSD成功加入CRUSHMap后，此时再取消先前设置的OSD标记，集群则进入恢复模式。

```
[root@host-192-168-0-17 ~]#  for i in noout nobackfill
norecover;do ceph osd unset $i;done
```

 提示

• 为防止失效的OSD被标记为“out”，可把mon osd down out interval设为0，此时，集群就不会进入恢复模式。

• 确保OSD被完整删除，否则，用ceph-deploy添加的时候将会生成新的OSD ID。

• 对于非默认的CRUSH架构的Ceph集群，保证配置文件ceph.conf里包含osd crush update on start=false选项，待到OSD成功加入，通过命令把OSD转移到原来的Bucket位置；否则，新添加的OSD会默认在Host层级下面。

• 为降低I/O性能的影响，加入CRUSH Map的OSD权重初始值为0，然后逐渐添加。设置命令为ceph osd crush reweight{osd-id}0。

13.3 本章小结

通过本章学习，大家可以了解Ceph在日常运维过程的一些注意点，比如，如何更换硬盘，使线上环境受到的影响达到最小等。本章所讲只是一些普遍存在的问题与解决方法。Ceph是一个庞大的集群系统，更多经验的积累需要读者在使用过程中好好总结。