代　号	10701	学　号	1110122659
分类号	TP311.5	密　级	公开
U D C		编　号	

题（中、英文）目 Ceph分布式文件系统的研究及性能测试

Research and Performance Testing of the Ceph Distributed File System

作 者 姓 名　李翔　学校指导教师姓名职称　李青山　教授

工 程 领 域　软件工程　企业指导教师姓名职称　魏彬　高工

论 文 类 型　技术论文　提交论文日期　二〇一四年三月

西安电子科技大学

学位论文创新性声明

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

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名：_____ 日期_____

西安电子科技大学

关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。

（保密的论文在解密后遵守此规定）

本学位论文属于保密，在___年解密后适用本授权书。

本人签名：_____ 日期_____

导师签名：_____ 日期_____

摘　要

分布式文件系统作为分布式系统的存储子系统，能够有效的解决海量数据存储的 I/O 瓶颈问题，成为了目前业界研究的热点。Ceph 分布式文件系统的第一个版本于 2012 年 6 月发布，作为一个新兴的分布式文件系统，系统的架构特点、结构特征、性能、可用性和扩展性等各个方面都亟需测试、验证与研究。

本文以 Ceph 分布式文件系统为研究对象，首先对文件系统的架构以及系统原型中的两个关键技术：CRUSH 算法和 RADOS 对象存储服务进行了详细的分析，研究了它们对系统性能和扩展性的影响；然后根据之前的分析研究结果搭建测试环境并设计系统的测试用例，对系统的性能、可用性以及扩展性进行了全面的测试。最后，根据研究测试结果提出新的 Journal 存储方案和新的集群网络拓扑结构，并对优化过的系统做进一步测试，从而验证优化是否正确。

测试结果表明，Ceph 分布式文件系统具有高可用性、高可扩展性以及优良的性能，文中所提出的两个集群部署优化方案也确实提高了系统的性能。

关键词：Ceph 分布式文件系统　　CRUSH 算法　　RADOS　　系统性能优化

Abstract

As the storage subsystem of Distributed Systems, Distributed File System that can effectively solve the I/O bottleneck problem has become a research hotspot in the corresponding industry. Ceph is an emerging Distributed File System whose first version was released in June 2012. All aspects of this Distributed File System, such as architectural and structural features, performance, availability and expansibility and so on, is in badly need of testing, verifying and studying.

The research object of this paper is Ceph Distributed File System. Firstly, we draw up a detailed analysis of two key technologies of the system's architecture and archetype: CRUSH algorithm and RADOS objective storage service. Their impact on the performance and expansibility of the Ceph system is mainly studied. In the next, we set up a test environment design system test cases according to the results of previous research and then use the cases to conduct a comprehensive test on the performance, availability and expansibility of the system. Finally, a new Journal storage solution and a new cluster network topology are proposed on the basis of the research. To verify the correctness of the optimization, a further test of the optimized system is conducted.

The test results indicate that Ceph Distributed File System possesses high availability high expansibility and excellent performance. Also, the two cluster deployment optimizations raised in this paper do improve the performance of the system.

Keyword: Ceph Distributed File System CRUSH Algorithm RADOS System Performance Optimization

目　录

第一章 绪论

根据图灵奖获得者 Jim Gray 提出的摩尔定律可知[1]，从现在起，每过 18 个月，新增加的网络数据存储量等于有史以来的数据存储量之和。近年来，随着网络应用的快速普及和云存储的推广，网络数据呈现海量的增长态势越发明显[2]。这对存储系统的容量、可扩展性、数据可用性以及 I/O 性能等方面提出了越来越高的要求。本文以 Ceph 分布式文件系统为研究对象，详细分析它的架构特点及性能特征，在此基础上，对系统的性能、可用性、扩展性等方面进行测试，并提出了关于系统性能的优化方案。下面将介绍本文的研究背景、国内外研究现状以及论文的主要工作。

1.1 选题背景及意义

1.1.1 分布式存储系统研究背景

随着信息化技术不断创新，信息化水平的不断提高，人们对数据存储能力、计算能力的需求呈现爆炸式的增长。应用程序需要存储和计算 PB 甚至 TB 级别的数据，即使增加更多的节点，更多的存储设备以及处理器，应用程序也不能提供足够快的计算能力。因此，高效的存储和计算能力成为当下必须面对的一个挑战。随着分布式文件系统对于大规模集群系统的重要性越来越大，各国政府和大型计算机公司更加重视对高性能、可伸缩性的存储系统的研究和投资。同时，以廉价的硬件以及开源的软件所组成的分布式存储系统逐渐成为主流的存储和计算平台。

分布式系统的理论出现于上个世纪的 70 年代。然而，最近十年分布式系统才得到广泛的应用，其中的一个重要的原因就是人类活动创造出的数据量远远超出了单个计算机的存储和处理能力。比如，2009 年全球互联网的网页超出了一万亿，按平均每个网页 20KB 计算，就是 20PB；又如，一个拥有 1 亿用户的电信运营商，如果每个用户每天拨打接听总共 10 个电话，每个电话按 500 字节计算，10 年后，这些用户的话费记录总量将会达到 1.75PB。除了分布式系统，人们还很难利用其他高效的手段来存储和处理这些 PB 级甚至更多的数据。

传统的分布式存储系统由高端处理器、存储设备构成，而现阶段互联网公司的分布式存储系统则是由数量多、成本低、性价比高的普通 PC 服务器通过网络连接而成。因为现阶段互联网的业务发展速度十分迅猛，这就使得文件存储系统不能依靠传统的纵向扩展的方式，即先购买小型机，当业务扩展到一定程度小型机无法满足需求时，再购买中型机，甚至大型机。互联网后端的分布式存储系统

要求支持横向的扩展方式，即通过增加普通服务器的数量来提高系统整体的存储和处理能力。普通 PC 服务器性价比高，同时发生故障的概率也高，这就需要在软件层面实现系统的自动容错，并保持数据的一致性。另外，随着服务器的不断加入，需要能够在软件层面实现自动负载均衡，使得系统的存储、处理能力得到线性扩展。

Ceph 分布式文件系统是一个新兴的分布式文件系统，它的原型是一项关于存储系统的 PhD 研究项目，由加利福尼亚大学的 Sage Weil 在 2007 年提出。2010 年三月底，可以在主线 Linux 内核（从 2.6.34 版开始）找到 Ceph 的身影。目前 Inktank 公司掌控 Ceph 文件系统的开发，Ceph 是开源的，遵循 LGPL 协议。Ceph 的第一个版本在 2012 年 6 月发布，但是就目前的情况而言 Ceph 无论从架构还是其工程实现来看都很不成熟，难以在生产环境中使用，因此对其架构进行分析，进行相关的功能和性能测试，找出系统性能瓶颈，并进行进一步的性能优化是非常有意义的。

1.1.2 分布式存储系统测试的研究背景及意义

近年来随着互联网技术的迅速发展和应用的多样化，无处不在的海量存储成为了当今研究的热点。软件系统由集中式系统逐渐转变为分布式系统。通常，分布式存储系统的结构极其复杂，由网络、数据库、应用服务器等多个部分组成，整个系统中的任何一部分皆可能成为瓶颈，从而导致整个系统性能降低，难以满足用户的需求。正是由于分布式存储系统的复杂性，使保证软件系统性能变得极为困难。软件测试是保证与确认软件质量的主要手段，软件性能测试是作为软件测试的重要组成部分，是验证软件系统是否满足用户需求、是否达到设计目标、是否能交付使用的重要途径。由于分布式软件的性能测试在国内刚刚起步，国内的一些大型软件开发企业进行了相关的性能测试实践，但是由于缺乏成熟的理论和方法对性能测试过程予以指导，因此软件性能测试的效果难以保证。

分布式文件存储系统的性能测试不同于普通的文件系统性能测试。首先，需要根据系统支持的操作确定系统的基准测试，然后，需要根据系统的设计规模对多用户并发访问系统的情形进行模拟等等。

1.2 国内外现状分析

分布式文件系统发展至今已经有 40 年的历史，第一代分布式文件系统始于上世纪 80 年代，以 NFS(Network File System)[3]和 AFS(Andrew File System)[4]为代表，它们提供标准接口，以远程文件访问为目的，更多地关注访问的性能和数据可靠性。随着网络技术的发展和普及，基于光纤通道的 SAN(Storage Area

Network)[5]、NAS(Network Access Server)[6]网络存储技术得到了广泛应用，出现了多种分布式系统体系结构，最具代表性的有 Global File System[7]，GPFS(General Parallel File System)[8]等。随着 NAS 和 SAN 技术的不断发展，研究人员考虑将两种体系结构结合起来，充分利用两者的优势，这一时期，IBM 的 StorageTank[9]、Cluster 的 Lustre[10]等文件系统是这种体系结构的代表。各种应用对存储系统提出了更多的需求，如大容量、高性能、高可用性、可扩展性、可管理性、按需服务等。

随着社交网络、移动互联网、电子商务等技术的不断发展，近些年互联网的使用者贡献了越来越大的数据。为了处理这些数据，每个互联网公司在后端都有一套成熟的分布式系统用于数据的存储、计算及其价值提取。Google 是全球最大的互联网公司，也是分布式技术上相对成熟的公司，其公布的 Google 分布式文件系统 GFS(Google File System)[11]、分布式计算系统 MapReduce[12]、分布式表格系统 Bigtable[13]都成为业界竞相模仿的对象，最近公布的全球数据库 Spanner[14]更是能够支持分布在全世界各地上百个数据中心的上百万台服务器。Google 的核心技术正是后端这些处理海量数据的分布式系统。和 Google 类似，国外的亚马逊、微软以及国内的互联网三巨头阿里巴巴、百度和腾讯的核心技术也是其后端的海量数据处理系统。

1.3 论文工作内容

本文主要对 Ceph[15][16]分布式文件系统的系统架构及关键技术进行了理论研究，详细分析研究了 Ceph 文件系统的两个存储接口即 RADOS 对象存储接口以及 Ceph RBD 块存储接口，对文件系统的性能、功能以及扩展性等方面进行了需求分析，根据需求分析设计了相关的测试用例，完成了 RADOS 对象存储接口和 Ceph RBD 块存储接口的性能测试、压力测试和扩展性测试，最后根据测试结果并综合系统结构特征对集群部署方式进行了优化，进而提高了系统的性能。在整个过程中主要完成以下工作：

1. 介绍分布式存储系统相关的基础概念以及衡量分布式存储系统的标准，介绍了包括数据分布、副本控制、集群扩展等实现分布式存储系统的关键技术。

2. 对 Ceph 分布式文件系统的体系结构和特性进行分析和研究。从系统的硬件、网络部署、集群部署、可用性、扩展性和系统优化几个方面提出了需求分析。根据需求分析主要针对 Ceph 的两个存储接口（RADOS 对象存储接口和 Ceph RBD 存储接口）进行测试的用例分析与设计。

3. 根据之前的需求分析，搭建测试系统，对 RADOS 对象存储接口和 Ceph RBD 接口进行性能测试、压力测试和扩展性测试，分析测试结果，对影响系统

性能的因子进行研究，进一步优化调整测试系统结构，提出系统性能优化方案。

1.4 论文组织结构

论文共分为六章，各章主要内容如下：

第一章：绪论。提出论文选题背景及意义、国内外研究现状分析、论文主要的工作内容和组织结构。

第二章：分布式文件系统概述。介绍了分布式文件系统的基本概念以及实现分布式文件系统的一些关键技术环节，如数据分布、副本控制、以及集群扩展性等。

第三章：Ceph 文件系统测试需求分析。分析了 Ceph 文件系统的系统架构以及关键技术，根据 Ceph 文件系统的特性提出系统硬件、集群网络配置、可用性、扩展性以及高性能方面的测试需求，为将来的测试打下理论基础。

第四章：Ceph 文件系统测试设计。根据第三章提出的测试需求，阐述系统测试流程，针对 Ceph 文件系统的两个接口（RADOS 对象存储接口和 Ceph RBD 存储接口），设计压力测试、性能测试、扩展性测试以及性能优化等方面的测试用例。

第五章：Ceph 文件系统测试优化及结果分析。介绍测试环境的部署流程，阐述系统测试流程，完成对 RADOS 对象存储接口和 Ceph RBD 块存储接口的压力测试、性能测试和扩展性测试，根据测试结果以及系统特性对测试环境进行进一步优化。最后针对测试结果进行分析并给出结论。

第六章：结束语。总结了本文的主要工作，指出工作的不足及进一步的改进方向。

第二章 分布式文件系统概述

分布式文件系统[17][18]是分布式系统的存储子系统，分布式文件系统管理的物理存储资源不一定连接在本地节点上，而是通过计算机网络实现节点的互连。

分布式存储系统面临的第一个问题就是数据分布，即将数据均匀地分布到多个存储节点。另外，为了保证可靠性和可用性，需要将数据复制多个副本，这就带来了多个副本之间的一致性问题。大规模分布式系统的重要目标就是节省成本，因而只能采用性价比比较高的PC服务器。这些服务器性能很好，但是故障率很高，这要求系统能够在软件层面实现自动容错。当存储节点出现故障时，系统能够自动检测出来，并将原有数据和服务迁移到集群中其他正常工作的节点。本章介绍分布式文件系统中的一些关键技术。

2.1 分布式文件系统基本概念

本小节介绍了分布式文件系统中一些基本概念，如系统异常、副本一致性等，接下来介绍了衡量分布式文件系统的一些常见标准。

2.1.1 异常

在大规模分布式存储系统中，往往将一台服务器或者服务器上运行的一个进程称为一个节点，节点与节点之间通过网络互连传递信息。然而，服务器节点是不可靠的，网络也是不可靠的，本节介绍存储过程中可能会遇到的各种异常。

1. 服务器异常

服务器异常是分布式存储系统中最常见的异常之一，内存错误、内存溢出、服务器断电等因素均可导致服务器异常。异常随时可能发生，当服务器发生异常时，该机器上面的节点无法正常工作，成为不可用的节点，一般需要人工介入手动重启机器。服务器重启后节点将失去所有的内存信息，因此，存储系统应该能够支持通过读取持久化设备（如机械硬盘、固态硬盘等）中的数据，来恢复异常前内存的信息，从而使系统恢复到异常前的某个一致的状态，即可用状态。

2. 网络异常

网络异常是另一种常见的异常，导致网络异常的原因可能是消息丢失、消息乱序或者网络包数据错误等。有一种特殊的网络异常称为“网络分区”，即集群的所有节点被划分为多个区域，每个区域可以正常通信，但是区域之间无法通信。例如，在多个数据中心部署某个分布式系统，由于网络调整，可能导致数据中心之间无法通信，同时，数据中心内部可以正常通信。

设计容错系统的一个基本原则是：网络永远是不可靠的，任何一个消息只有收到对方的回复后才可以认为发送成功，分布式存储系统设计时总是假设网络将会出现异常并采取相应的处理措施。

3. 磁盘异常

磁盘异常是一种发生概率很高的异常。一般，磁盘异常分为两种情况：磁盘损坏和磁盘数据错误。磁盘损坏将会丢失存储在上面的数据，因此，分布式存储系统需要将数据存储到多台服务器上，如果集群中的一台服务器磁盘出现异常，系统也能从其他服务器节点上恢复丢失的数据。对于磁盘数据错误，往往可以采用校验和机制来解决，这样的机制既可以在操作系统层面实现，又可以在上层的分布式存储系统层面实现。

4. 系统超时

由于网络异常的存在，分布式存储系统中请求结果存在“三态”的概念。在单机系统中，只要服务器没有发生异常，每个函数的执行结果是确定的，要么成功，要么失败。然而，在分布式系统中，如果某个节点向另外一个节点发起 RPC（Remote Procedure Call）调用，这个 RPC 执行的结果有三种状态：“成功”、“失败”、“超时”，也称为分布式存储系统的三态。

图 2.1 给出了 RPC 执行成功但超时的例子。服务器收到并成功处理完成客户端的请求，但是由于网络异常或者服务器宕机，客户端没有收到服务器端的回复。此时，RPC 的执行结果为超时，客户端不能简单地认为服务器处理失败。

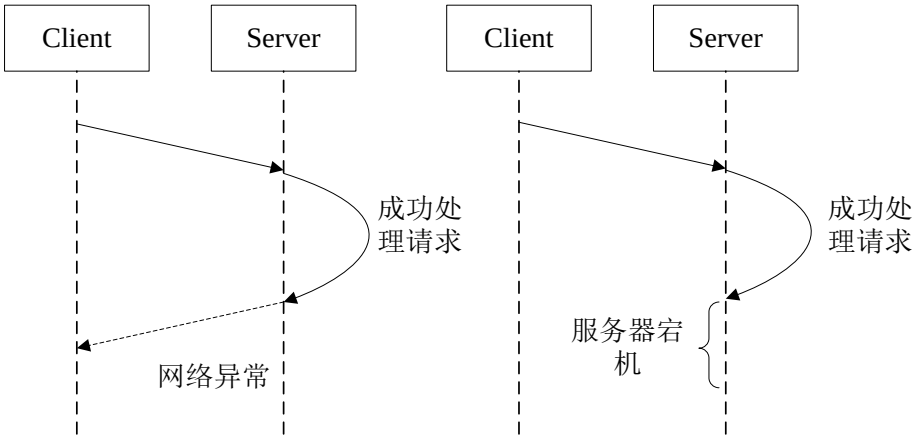

图 2.1 RPC 执行成功但超时

当出现超时状态时，只能通过不断读取之前操作的状态来验证 RPC 操作是否成功。当然，设计分布式系统时可以将操作设计为“幂等”的，也就是说，操作执行一次与执行多次的结果相同，例如，覆盖写就是一种常见的幂等操作。如果采用这种设计，当出现失败和超时时，都可以采用相同的处理方式，即一直重试直到成功。

2.1.2 副本一致性

由于服务器、网络、磁盘故障的存在，大多数分布式存储系统在设计时往往会将数据冗余备份，每一份称为一个副本，这样，当某一个节点出现故障时，可以从其他副本上读取数据。可以这么认为，副本是分布式存储系统容错技术的唯一手段。由于分布式系统中存储了数据的多个副本，因此保证副本之间的数据一致性就成了系统的理论核心。分布式系统为了提高可用性，总是不可避免的使用副本的机制，从而引发副本一致性问题。一般来说，越是强的一致性模型，用户使用起来越简单。如果系统部署在同一个数据中心，只要系统设计合理，在保证强一致性的前提下，不会对性能和可用性造成太大的影响。

可以从两个角度理解一致性：第一个角度是用户，或者说是客户端，即客户端读写操作是否符合某种特性；第二个角度是存储系统，即存储系统的多个副本之间是否一致，是否按照相同的顺序执行更新操作等等。

2.1.3 衡量分布式系统的标准

下面分别介绍评价分布式存储系统的一些常用标准。

1. 性能

常见的性能指标有系统整体的吞吐能力以及请求的响应时间。其中，系统的吞吐能力指系统在某一段时间可以处理的请求总数，通常用每秒处理的读操作数或者写操作数来衡量；系统的响应延迟是指从某个请求发出到接受返回结果消耗的时间，通常用平均延时或者 99.9%以上请求的最大延时来衡量。然而，这两个指标往往是矛盾的，追求高吞吐的系统，往往很难做到低时延，反之亦然；因此，设计系统时需要权衡这两个指标。

2. 可用性

系统的可用性是指系统在面对各种故障是能够提供正常存储服务的能力。系统的可用性可以用系统停止服务的时间与正常服务的时间的比例来衡量，也可以用某功能失败次数与成功次数的比例来衡量。可用性是分布式系统的重要指标，是系统容错能力的体现。

3. 可扩展性

系统的可扩展性指分布式存储系统通过扩展集群服务规模来提高系统存储容量、计算量和性能的能力。随着业务的发展，对底层存储系统的性能需求不断增加，比较好的方式就是通过自动增加服务器数量提高系统的能力。理想的分布式存储系统实现了“线性可扩展”，也就是说，随着集群规模的增加，系统的整体性能与服务器数量呈线性关系。

2.2 分布式文件系统关键技术

分布式系统区别于传统单机系统在于能够将数据分布到多个节点，并在多个节点间实现负载均衡。分布式系统中数据保存为多个副本，这种机制保证了分布式系统的高可靠和高可用性。同时通过数据分布，复制以及容错等机制能够将分布式系统部署到成千上万台服务器。本小节介绍分布式文件系统的一些关键技术技术，包括数据分布、副本控制、集群扩展性等。

2.2.1 数据分布

分布式存储系统与传统单机系统的区别在于分布式系统可以将数据分布到集群的多个节点中，并在多个节点之间实现一致性访问和负载均衡。数据分布的方式主要有两种，一种是哈希分布，另一种方法是顺序分布。将数据分散到多个节点后，需要尽量保证多个节点之间的负载是均衡的。衡量机器负载涉及的因素很多，如 CPU、内存、磁盘以及网络等资源使用率，读写请求量等。分布式存储系统需要能够自动识别负载高的节点，当某台机器的负载较高时，将它服务的部分数据迁移到其他机器，实现负载均衡。本节将介绍数据分布的相关内容。

1. 哈希分布

哈希分布[19]是最常见的数据分布方式，它根据数据的某一种特征值计算哈希值，并将该哈希值与集群中的服务器建立映射关系，从而将具有不同哈希值的数据分布到不同的服务器上。所谓数据特征可以是 key-value 系统中的主键（key），也可以是其他与业务逻辑相关的值。例如，将集群中的服务器按 0 到 N-1 编号（N 为服务器的数量），根据数据的主键（hash（key）%N）或者数据所属的 user_id（hash（user_id）%N）计算哈希值，来决定将数据映射到哪一个服务器节点，图 2.2 给出了哈希方式分配数据的一组例子。

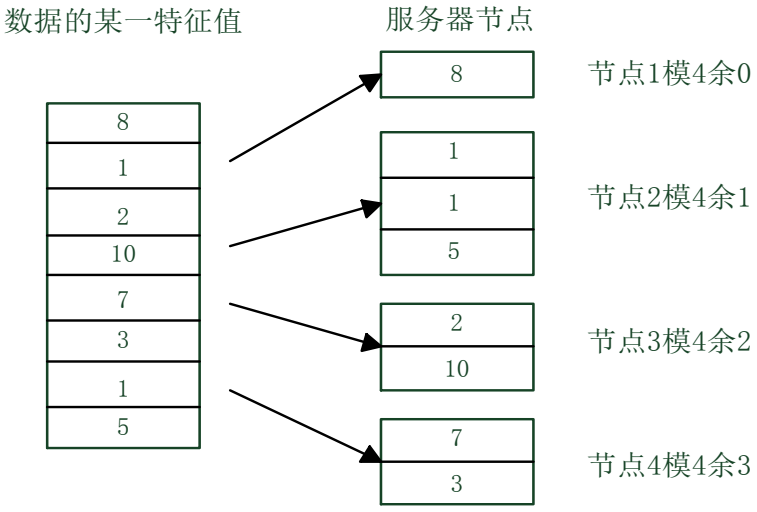

图 2.2 哈希方式分配数据

如果哈希函数的散列性能很好，哈希方程可以将数据比较均匀地分布到集群的节点中去。然而，找出一个散列特性很好的哈希函数是很难的，这是因为，如果按照主键散列，那么同一个 user_id 下的数据可能被分散到多个节点，这会使得一次操作同一个 user_id 下的多条记录变得困难；如果按照 user_id 散列，无论集群的规模有多大，某些数据量比较大的用户将会将其数据映射到相对集中的节点中去，同样会严重影响负载的平衡，如图 2.3 所示。

采用传统哈希分布算法还会遇到这样一个问题：当集群扩展或某些节点宕机，N 值发生变化，此时数据映射完全被打乱，几乎所有的数据都需要重新分布到新的集群，这将带来大量的数据迁移和重组。

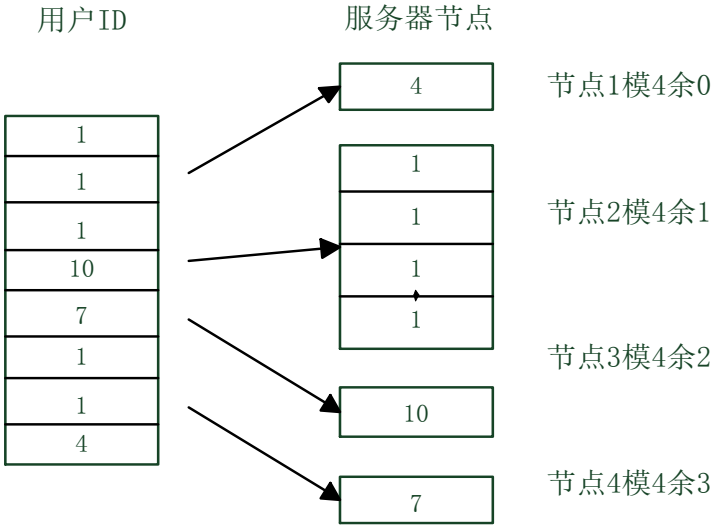

图 2.3 发生“数据倾斜”的哈希分布

2. 顺序分布

哈希散列破坏了数据的有序性，只支持随机读取操作，不能够支持顺序扫描。某些系统可以在应用层做折衷，比如互联网应用经常按照用户来进行数据拆分，并通过哈希方法进行数据分布，同一个用户的数据分布到相同的存储节点，允许对同一个用户的数据进行顺序扫描，由应用层解决跨多个用户的操作问题。另外，这种方式可能出现某些用户的数据量太大的问题，由于用户的数据限定在一个存储节点，无法发挥分布式存储系统的多机并行处理能力。

2.2.2 副本控制

为了保证分布式存储系统的高可靠性和高可用性，数据在系统中一般会存储多个副本。当某个副本所在的存储节点发生故障时，分布式存储系统能够将存储服务自动的切换到其他的副本从而保证数据的可用性。分布式存储系统通过复制协议将数据同步到多个存储节点，并确保多个副本之间的数据一致性。

同一数据的多个副本往往有一个副本被称为主副本（Primary-copy），其他副本称为备份副本（Backup-copy），由主副本负责将数据复制到备份副本。复制协议分为两种，强同步复制以及异步复制，两者的区别在于用户的写请求是否需要同步到备副本才可以返回成功。加入备份副本不止一个，复制协议还会要求写请求至少需要同步到几个备副本。当主副本出现故障时，分布式存储系统能够将服务自动切换到某个备副本，实现自动容错。

一致性和可用性是矛盾的，强同步复制协议可以保证主备副本之间的一致性，但是当备副本出现故障时，也可能阻塞存储系统的正常写服务，系统的整体可用性受到影响；异步复制协议的可用性相对较好，但是一致性得不到保障，主副本出现故障时还有数据丢失的可能。

2.2.3 集群可扩展性

通过数据分布、复制以及容错等级制，能够将分布式存储系统部署到成千上万台服务器上。可扩展性的实现手段很多，如通过增加副本个数或者缓存提高读取能力，将数据分片使得每个分片可以被分配到不同的节点以实现分布式处理，把数据复制到多个数据中心等。分布式系统大多都带有中心服务器，主流的分布式存储系统大多带有中心服务器，且能够支持成千上万台的集群规模。

1. 中心服务器

分布式存储系统中往往有一个中心服务器用于维护数据的分布信息，执行工作机管理、数据定位、故障检测和恢复、负载均衡等全局调度工作。通过引入中心服务器，可以使得系统的设计更加简单，并且更加容易做到强一致性。那么中心服务器是否会成为性能瓶颈呢？

分为两种情况：分布式文件系统的中心服务器处理执行全局调度，还需要维护文件系统目录树，内容容量可能会率先成为性能瓶颈；而其它分布式存储系统的中心服务器只需要维护数据分片的位置信息，一般不会成为瓶颈。另外，即使是分布式文件系统，只要设计合理，也能够扩展到几千台服务器。例如，Google 的分布式文件系统 GFS 能够扩展到 8000 台以上的集群，开源的 Hadoop 也能够扩展到 3000 台以上的集群。当然，设计时需要减少中心服务器的负载，比如 GFS 舍弃了对小文件的支持，并且把对数据的读写控制权下放到工作机 ChunkServer，通过客户端缓存元数据减少对中心服务器的访问等。

如果中心服务器成为瓶颈，例如需要支持超过一万台的集群规模，或者需要支持海量的小文件，那么，可以采用两级结构，如图 2.4 所示。在总控机与工作机之间增加一层元数据节点，每个元数据节点只维护一部分而不是整个分布式文件系统的元数据量。这样，总控机也只需要维护元数据节点的元数据，不可能成

为性能瓶颈。假设分布式文件系统中有 100 个元数据节点，每个元数据节点服务一亿个文件，系统总共可以服务 100 亿个文件。图 2.4 中的 DFS[20]客户端定位 DFS 工作机时，需要首先访问 DFS 总控机找到 DFS 元数据服务器，在通过元数据服务器找到 DFS 工作机。虽然看似增加了一次网络请求，但是客户端总是能够缓存 DFS 总控机上的元数据，因此并不会带来额外的开销。

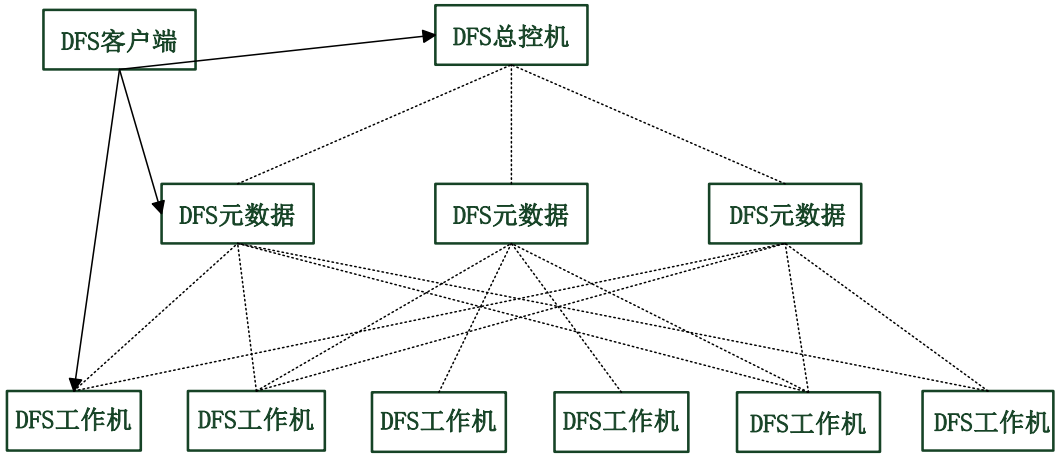

图 2.4 两级元数据架构

2. 异构系统

大规模分布式存储系统要求具有线性可扩展性，即随时加入或者删除一个或者多个存储节点，系统的处理能力与存储节点的个数成线性关系。为了实现线性可扩展性，存储系统的存储节点之间是异构的，否则，当集群规模达到一定程度后，增加节点将变得特别困难。异构系统将数据划分为很多大小接近的分片，每个分片的多个副本可以分布到集群中的任何一个存储节点。如果某个节点发生故障，原有的服务将由整个集群而不是某几个固定的节点来恢复。

如图 2.5 所示，系统中有五个分片（A，B，C，D，E），每个分片包含三个副本，如分片 A 的三个副本分别为 A1，A2，A3。假如节点 1 发生永久故障，那么可以从剩余的节点中任意选择健康的节点来增加 A，B 以及 E 的副本。由于整个集群都参与到节点 1 的故障恢复过程，故障恢复时间很短，而且集群规模越大，优势就会越明显。

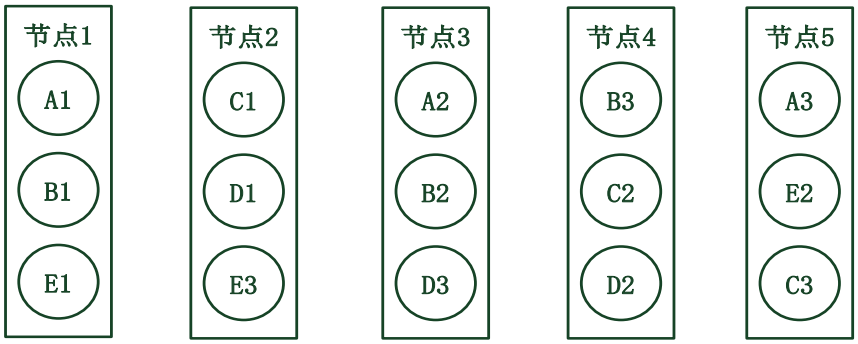

图 2.5 异构系统

2.3 Ceph 分布式文件系统介绍

Ceph 分布式文件系统主要由四个组件组成：一，客户端，每一个实例向一个主机或进程暴露一个 near-POSIX[23]文件系统接口；二，一个 OSD（Object Storage Device）集群，它们协同工作，用来存储所有数据和元数据，同时负责处理数据复制、故障恢复与负载平衡，OSD 节点还会向 Monitor 节点提供心跳信息以供其他 OSD 节点检测其状态；三，一个元数据集群（MDS，Metadata Server），用来管理名字空间（文件名和目录结构），同时协调数据的安全性、一致性和连贯性，Ceph 元数据服务器为 POSIX 文件系统用户执行像 ls、find 等基本操作提供了可行性；四，一个 Monitor 集群，它是一个轻量级结构，负责管理集群的成员状态，当对象存储设备发生故障或者新的设备添加到集群时，Monitor 负责检测和维护一个有效的集群映射。之所以说 Ceph 的接口是 near-POSIX 的，是因为它非常适合扩展接口，并且可以选择相对宽松的一致性语义，从而使应用需求和高性能之间达成更好的一致性。Ceph 文件系统架构如图 2.6 所示，客户端直接与 OSD 集群通信完成文件 IO。每一个进程可以直接同一个客户端实例直接连接，或者与一个挂载的文件系统进行交互。

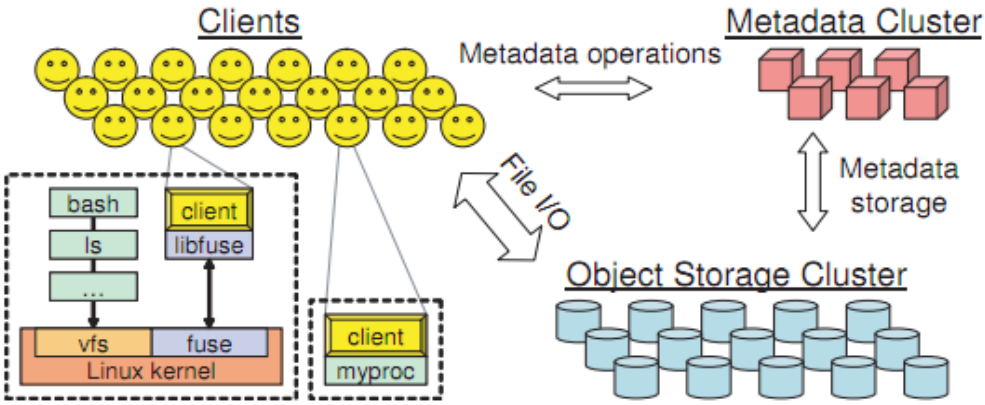

图 2.6 Ceph 分布式文件系统架构图

Ceph 架构设计的首要目标是高可扩展性（系统存储量达到数百 PB 级别以上）、高性能和高可靠性。扩展性需要考虑很多方面，需要兼顾系统整体的存储能力、吞吐量，以及客户端、独立的目录以及文件的性能表现。Ceph 文件系统目标负载包括这样十分极端的例子——数百数千个主机并行的读取同一个文件或在同一个目录下创建文件。这种情况在运行于超级计算集群上的科学应用比较常见。同时也预示着未来普遍的负载目标。更重要的是，应用对数据和元数据的访问有很大差异，而且分布式文件系统的工作负载的本质是动态的，随着时间不断变化。Ceph 同时解决了关于扩展性方面的三个问题：动态分布式的元数据管理，解耦数据和元数据，以及可靠自治的分布式对象存储。

一．动态分布式的元数据管理：因为文件系统的元数据操作通常占到了文件系统将近一半的工作量，高效的元数据管理对整个系统的性能起着重要的作用。Ceph 使用了一个新颖的基于动态子树分割的元数据集群架构，将对文件系统层次目录结构管理的职责自适应的、智能的分配到数十个甚至数百个 MDSs 中。每个 MDS 本地保存一个动态的层次结构目录分区，提高了更新和预取的效率，从而提高了系统的性能。特别地，在元数据服务器中负载的分布完全基于当前的访问模式，允许 Ceph 在任何负载下高效的利用可用的 MDS 资源，并可以根据 MDS 的数量的增长，使元数据访问的性能线性的增长。

二．通过 CRUSH 算法解耦数据和元数据访问：Ceph 文件系统最大化的将元数据管理从文件数据存储中分离，元数据操作（如 open，rename）被一个元数据集群管理，同时客户端直接和 OSDs 集群交互完成文件 IO（读和写）。基于对象的存储方案由存储设备（OSDs）来负责低级别的块分配，从而提高文件系统的扩展性。与已经存在的基于对象的文件系统相比，Ceph 将每个文件都持有的冗长的块列表替换为更短的对象列表。使用一个基于 inode number、比特范围和条块化方案的简单方程来命名文件数据所包含的的对象。同时使用 CRUSH 算法将对象分配到指定的存储设备上。这样的话，允许任何部分计算（而不是查找）包含了文件内容的对象的名字和位置，不需要维护和分发对象列表，简化了系统的设计，也减轻了元数据集群的负载。

三．可靠的自治的分布式对象存储集群（RADOS）：大型系统由数千台设备动态的构成，它们以递增的方式构建；随着新设备的添加而增长，并且随着旧设备的废除而缩减；设备故障频繁且可预期，大量的数据被创建、移动和删除。基于以上所有的因素，就需要系统高效利用资源并维持一定程度的数据复制。Ceph 授权 OSDs 集群完成数据迁、复制、故障检测和故障恢复，并负责数据的存储。从深层次的方面讲，OSDs 向客户端和元数据集群提供了一个分布的，可靠的对象存储。这个方案允许有效的平衡每个 OSD 的智能性（CPU 和内存），从而完成可靠、高可用的对象存储。

2.4 本章小结

本章首先介绍了分布式文件存储系统的基本概念，包括设备故障、副本一致性以及衡量分布式存储系统的标准。然后介绍了实现分布式文件系统的关键技术，例如数据分布、副本控制，以及集群扩展性等技术细节。最后对 Ceph 分布式文件系统进行了简单介绍。

第三章 Ceph 文件系统测试需求分析

本章首先对 Ceph 分布式文件系统中的两个关键技术进行详细的分析，一个是 CRUSH 算法，它在不依赖任何中心服务器的情况下高效的将数据映射到存储集群中，另一个是智能的分布式对象存储策略 RADOS，RADOS 促进了层次结构的动态存储集群中数据和工作量的平衡分布，并且给应用提供了一个拥有强大安全语义和一致性保证的逻辑对象存储接口。

在对 Ceph 文件系统的架构和关键技术进行详细分析后，本章提出了对 Ceph 文件系统关于硬件配置、网络配置、可用性、扩展性以及性能优化方面的测试流程以及需求分析，为下一步设计测试用例打下基础。

3.1 Ceph 文件系统分析

3.1.1 CRUSH 算法研究分析

新兴的大型分布式文件系统面临着将 PB 级别的数据分布到数百甚至数千个存储设备中去这样的任务。这样的系统必须有效的利用可用的资源，平均的分配数据和工作负载，使系统性能达到最大的同时有效的促进系统的扩展性。在 Ceph 文件系统中利用了 CRUSH 算法解决数据分布问题，它是一个可扩展的伪随机数据分布方程，其设计目标旨在为基于对象的分布式存储系统提供高效的数据映射，即不依赖任何中心目录的情况下将数据对象映射到存储设备。因为大型系统本质上是动态的，CRUSH 算法的设计在促进集群添加和移除设备的同时最大程度的减少不必要的数据移动，并根据用户定义的策略分布数据，强制性的将数据副本分配到不同的故障域中。

1. CRUSH 算法理论依据及设计原则

基于对象的存储架构是一种提供了高可扩展性、高性能、和易管理性的新兴架构[21]。与传统的基于块[22]（block-based）的存储架构不同的是，基于对象的存储设备（OSD）在其内部管理磁盘的块分布，该设备会暴露一个接口，允许系统其他部分通过该接口对已命名的大小各异的对象进行读写操作。在该系统中，每个文件的数据通常会被条块化到数目相对较小的对象中，然后被分布到存储集群，同时，对象的副本会被复制到多个设备中以避免因设备故障而造成数据的丢失。基于对象的存储系统通过将大的块列表（block lists）替换为相对较小对象列表（object lists）简化了数据分布。尽管这个做法通过减少了与文件数据定位相关的元数据并降低其复杂性，从而促进了系统的可扩展性，但是如何将数据平均

分布到上千个存储设备（这些设备具备不同的容量和性能特点）这一基本的的问题依然存在。

大多数存储系统只是简单的将新数据写到尚未被利用的设备中，这个方法的根本问题是数据只有在被写入时才会移动，而其他时间这些数据很少被移动。即使是一个完美的分布，当系统扩展时也将会失衡，因为新的设备不是被闲置就是仅仅容纳新的数据。新旧设备无法得到平衡的利用。因此，利用所有可以利用的资源来平均分布数据成为了设计的挑战。

简单的基于哈希的分布无法解决由于存储设备数量的不断变化而导致的大量数据重组这个问题。一个强健的措施是向所有可利用设备随机的的分布数据，这样可以根据概率将新数据与旧数据混合在一起，当新的设备增加进来，已存在数据的随机样本被迁移到新的设备中去从而使系统重新达到平衡。这个方法有个很重要的优势，就是所有设备会具有相同的工作负载，允许系统在任何潜在的工作负载下都能有优良的表现。此外，在一个大的存储系统中，一个大的文件会被随机的分布到一组可用的设备中去，提供了强大的并行访问和聚合带宽。

Ceph 文件系统中采用了 CRUSH（Controlled Replication Under Scalable Hashing）算法，它是一个伪随机的数据分布算法，可以高效并可靠的将对象副本分布到结构复杂的集群中去。CRUSH 用一个伪随机的、确定的方程，将一个输入值（通常是对象或对象组的 ID）映射到一组用于存储对象副本的设备中去。这不同于传统的方法，数据的定位不依赖于任何方式的文件目录或对象目录，CRUSH 仅仅需要一个当前集群层次结构的描述和副本的分布策略。这个算法有两个关键优势：一，它完全是分布式的，因此系统的任何一个部分都可以独立的计算出任何对象的位置。二，元数据的需求大多数是静态的，仅当设备增加或移除时才发生改变。

CRUSH 的设计的目的是利用现有设备最优化数据的分布，当存储设备移除或增加时高效的重组数据，当相关硬件出现故障时对对象副本定位实行灵活的限制从而提高数据的安全性。它支持多种数据安全机制，包括 n 路复制，RAID 奇偶校验，或其他形式的擦除编码。这些机制使得 CRUSH 十分适合管理超大型存储系统的对象分布。

2. CRUSH 算法分析

CRUSH 算法根据每一个设备的权重值将数据对象分布到存储设备中去，它接近于一个等概率的分布。CRUSH 利用了一个强大的多输入哈希方程，其输入仅使用 CRUSH Map、CRUSH Ruleset 和 x 就可以独立的完成到对象存储设备的映射。典型情况下，x 是一个对象名或对象组名，一个对象组内的对象会被分布

到一组相同的设备中去。CRUSH Map 是一个描述了集群中设备物理组成关系的层次结构图。CRUSH Ruleset 指定了数据的映射策略。

（1） CRUSH Map

CRUSH Map 描述了系统中了可用的存储资源以及一些逻辑元素的组成方式。CRUSH Map 由 device 和 bucket 构成，它们都有用数值表示的 ID 和与自身相关联的权重值。device 由 OSD 构成，而 bucket 将一组 device 整合到某一物理位置当中。bucket 分为四种类型，它们分别是 Uniform bucket，List bucket，Tree bucket 和 Straw bucket，每一种类型的 bucket 都代表了不同的数据重组的权衡策略。

每个 bucket 可以包含任意数量的 device 或其他类型的 bucket，允许它们在一个存储层次结构中形成内部的节点，同时 device 总是位于叶子节点。一个大型系统很可能包含容量各异，特征各异的设备，管理员可以为存储设备分配权重，权重代表了设备的存储能力。随机的数据分布将设备利用情况与工作负载相关联，这样的话设备负载将会与数据存储量成比例。

Bucket 可以以任意的方式组合，构建一个用来表示可用设备的层次结构，CRUSH Map 的表示形式如图 3.1 所示。

```
root@NEW-OSD2:~#ceph osd tree

#id    weight    typename        up/downreweight
-1     12        root default
-2     6                 rack rack1
-4     3                       host ceph1
11     1                             osd.11    up    1
12     1                             osd.12    up    1
13     1                             osd.13    up    1
-5     3                       host ceph2
21     1                             osd.21    up    1
22     1                             osd.22    up    1
23     1                             osd.23    up    1
-3     6                 rack rack2
-6     3                       host ceph3
31     1                             osd.31    up    1
32     1                             osd.32    up    1
33     1                             osd.33    up    1
-7     3                       host ceph4
41     1                             osd.41    up    1
42     1                             osd.42    up    1
43     1                             osd.43    up    1
```

图 3.1 CRUSH Map 结构

在这个存储集群层次结构中，最底层用 4 个“host” buckets 来代表若干组相同的设备，每个“host” bucket 由 3 个 devices 组成，每个 device 代表了一个 OSD 节点。然后将 2 个“host”bucket 分为一组并组合到 2 个“rack” bucket 中，最后两个“rack” bucket 组成一个“cabinet”bucket。

（2） CRUSH Ruleset

CRUSH Ruleset 定义了数据副本的分布策略，它指定了存入集群对象副本的个数，以及应用了什么样的约束条件来存放对象副本。用户可以根据自身情况制定 CRUSH Ruleset。例如，一个策略指定了一个对象的三个副本存储在位于不同服务器机架的设备中，这样他们就不用共享相同的电路设施。图 3.2 中的规则指定了所有副本都会被定位到不同的“host”bucket 中。图 3.3 中的规则副本会被分离到两个不同的“rack”bucket 中，在每个“rack”bucket 中，选择两个 OSDs，这两个 OSDs 可能属于同一个“host”bucket。

```
rule rbd
{
        ruleset 2
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}
```

图 3.2 CRUSH Ruleset1

```
rule rbd
{
        min_size 4
        max_size 4
        step take default
        step chooseleaf firstn 2 type rack
        step chooseleaf firstn 2 type osd
        step emit
}
```

图 3.3 CRUSH Ruleset 2

（3） Pools

Ceph 存储集群支持存储池（Pools）的概念，Pool 不同于 CRUSH 算法的基于位置的 bucket 的概念，它不是一个物理位置，而是为对象存储提供的逻辑分区。Pool 中有几个参数需要说明：1）Replicas，客户可以为一个对象指定副本数

量，一个典型的配置是为一个对象指定一个副本，但是客户可以通过配置调整副本的数量；2）Placement Group（PG）：每个 Pool 中包含了一定数量的 Placement Groups，一个 PG 可以将一系列的对象整合在一起。客户可以为每个 Pool 设置 PG 的数量，一个典型的配置是一个 Pool 使用 100 个 PGs；3）CRUSH Ruleset：它即为上节所提到的对象副本分布策略。

对象的映射过程如图 3.4 所示。首先客户端向 Ceph Monitor 节点获取集群当前的 CRUSH Map，然后将对象映射到存储池的某一个 PG 中。最后，CRUSH 算法可以根据该对象所在 PG 的 ID 号，集群的 CRUSH Map 以及 CRUSH Ruleset 将对象映射到一组 OSDs 中。将对象映射到 PGs 中这一过程等于是在 Ceph OSD 节点和客户端之间加了一个中间层。如果每个 Ceph 客户端知道哪个 OSD 节点拥有哪个对象的话，会大大的加大节点之间的耦合度，相反如果采用 CRUSH 算法首先将每个对象映射到一个 PG 中，然后再将每个 PG 与一个或多个 OSD 节点建立映射关系，这样的话，当新的节点添加到集群中，Ceph 可以利用这个中间层动态的重组数据。

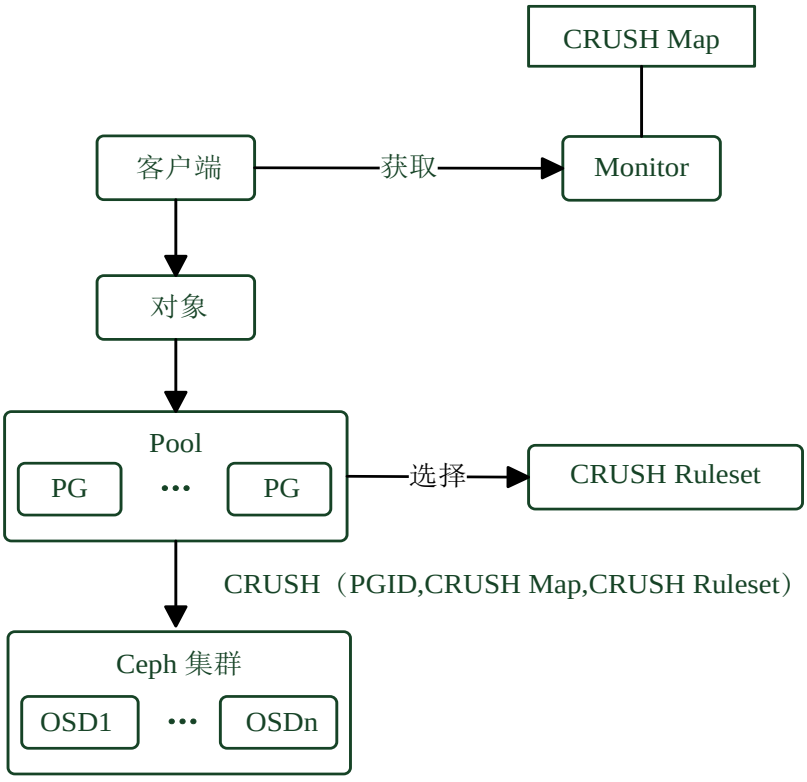

图 3.4 对象映射过程

3. 副本分配

CRUSH 在具有权重的设备中均匀的分配数据，来维持存储资源和带宽资源的平衡。在层次结构的存储设备中，副本的分配对于数据的安全性也有着至关重要的影响。根据设备的物理组织形式，CRUSH 可以通过建模从而解决设备潜在

的故障源问题。典型的故障源包括了物理位置接近的设备，共享电源的设备，以及共享网络的一组设备。把这些信息编码到 CRUSH Map 中，CRUSH 分配策略能够将对象副本分离到不同的故障域，同时保持数据合理的分布。例如，处理可能发生的并发访问的故障，如果能确保数据对象副本存储在不同的机架、使用不同电源或控制器的设备中，这样的话或许能够达到期望的效果。

因为 CRUSH 为每个复制策略或分布规则定义了 CRUSH Ruleset，允许存储系统或系统管理员指定对象副本如何放置。每一个规则包含了一些列的操作，并将这些操作应用于具有层次结构的执行环境中，take（a）操作在层次结构存储集群中选择一个 item（典型是一个 bucket），并将其赋予向量 i，然后将向量 i 作为后续操作的输入值，select（n，t）操作遍历每一个属于向量 i 中的元素，将那个元素作为根节点，并从该根节点的子树中选择 n 个不同的属于 t 类型的 item。存储设备有一个已知的，固定的类型，系统中的每一个 bucket 都有一个类型域用来区分不同类别的 bucket（某一些代表“rows”，某一些代表“cabinets”等等）结果为 n 个不同的 items 要么被追加到向量 i 中作为 select（n，t）的输入或者将其通过 emit 操作提交到结果向量中。

根据图 3.5 所示的集群层级结构。首先进行 take（root）操作，将 root 节点赋予向量 i 中，select（1，row）操作选择一个类型为“row”的 bucket（选择了 row2）。紧接着在之前所选择的 row2 节点下执行 select（3，cabinet），选择三个不同的 cabinets（选择了 cab21，cab23，cab24），最后在三个 cabinets 下迭代的执行 select（1，disk），分别在三个 cabinet 节点下选择一个 disk。最后选择的三个 disk 是在同一个 row 中，但是在三个不同 cabinet 中。这种方法可以使副本被分离的同时且被限制在同一种类型容器中（例如 rows，cabinets，shelves）。这就使得该算法很好的支持了系统的可靠性。

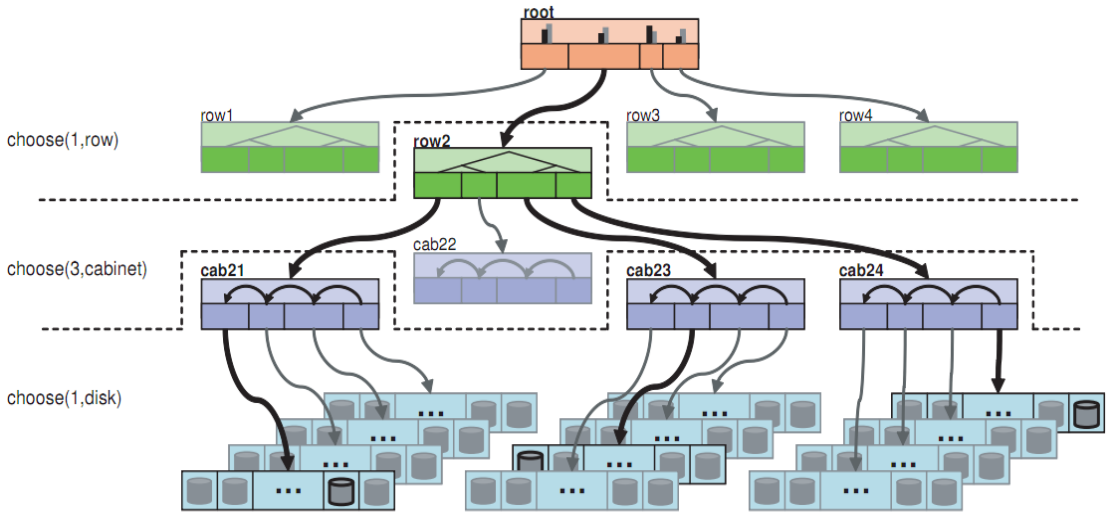

图 3.5 集群层次结构

3.1.2 可靠自治的分布式对象存储集群

本节将会介绍关于 RADOS 的设计与实现，它是一个可靠的对象存储服务，通过平衡的利用单独存储节点的智能性，可以将存储集群扩展到数千台设备。RADOS 维护了数据一致性访问并提供了强大的安全语义，同时允许节点之间通过维护小型的 CRUSH Map，进行半自治的数据管理和复制，完成故障检测和故障恢复。RADOS 的实现提供了极高的性能、可靠性和扩展性，同时向客户端提供了一个逻辑对象存储接口。

一个 PB 级别的存储系统需要动态的结构：系统以动态的方式构建，通过部署新的设备或废弃旧的设备来完成完成规模的扩展或缩减，设备故障和恢复总是持续发生，大量的数据被创建或者销毁。RADOS 通过使用具有版本的 CRUSH Map 确保数据分布的一致性和数据对象读写的一致性 。这个 map 被系统所有节点设备所共享（比如存储节点和客户端节点），并通过传播小型的增量更新来完成更新操作。

系统中所有的存储节点都知道当前数据的分布情况，设备可以通过使用类似点对点的协议半自治的管理数据副本，一致且安全的数据更新，参与故障检测并通过重新分配副本或迁移数据对象来响应设备故障。这样的话减轻了小型 Monitor 集群对于管理 CRUSH Map 的负担，通过对 CRUSH Map 的管理，能够使系统无缝的扩展到几十个甚至数千个设备。

1. RADOS 集群概述

一个 RADOS 集群包含了大量的 OSDs 和一小组 Monitors，用来负责管理 OSD 集群成员及设备状态，每一个 OSD 包含了 CPU，内存，网络接口，本地硬盘或 RAID[23]。Monitor 仅需要一块小的硬盘存储空间。RADOS 集群的结构如图 3.6 所示。

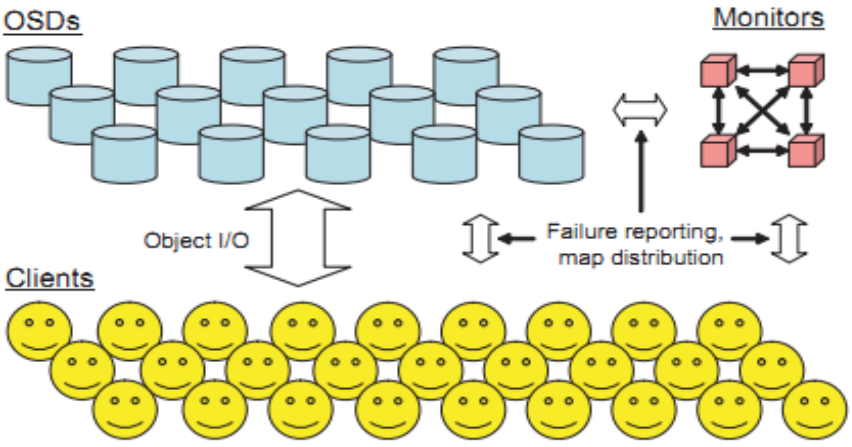

图 3.6 RADOS 结构图

RADOS 提供了三个关键的设计特征。

数据分布和集群状态是由一个可操作的紧凑的 CRUSH Map 所管理。map 包括了由 CRUSH 描述的一个紧凑的集群设备层级结构，一个全局可知的映射方程维护了对象的伪随机分布，同时保护了数据的安全。这提供给集群的所有部分（像客户端和存储设备）对数据分布的完全认知。集群的各部分可以通过计算得出数据的位置，而不需要去咨询中心化的对象目录。同时，一个小型的，紧凑的 Monitor 集群用来负责管理 map，并通过它，管理整个集群。

在更新协议中将同步性和安全性分离。促进了同步访问相同对象的效率的同时，当应用请求数据时仍然提供了强大的安全语义。

RADOS 利用设备的智能性来平衡对象副本的分布、故障的检测、故障恢复和数据的迁移。OSDs 通过观察对等点之间的 CRUSH Map 的异同和一个用来维护一致性和完成适当的数据副本分布的点对点算法来完成对分布的管理。数据更新采用了 primary-copy replication，chain replication 和一个混合策略来减小更新的延迟，同时提供了强大的一致性和安全性语义。

2. 可扩展的集群管理

RADOS 通过摈弃了大多数存储架构中所采用的控制器和网关服务器，让客户端直接访问存储设备，这使得 RADOS 有着良好的可扩展性。通过 CRUSH 算法，可以提供客户端和 OSDs 对当前数据分布的完全认知。当设备故障或集群扩展需要对数据分布做出改变时，通过 OSDs 集群之间的通信可以使它们意识到数据分布的改变。而不需要任何控制器的监控。

（1） OSD 节点状态

当 Ceph OSD 节点加入到 Ceph 集群中，会向集群报告它们的状态。一个 OSD 节点有两种状态，如图 3.7 所示：要么是 in 或者 out，标志着该 OSD 是否在集群中进行了注册，要么是 up 或者 down，标志着 OSD 节点是否在运行并且是否能够提供服务。如果一个 OSD 同时具有 down 和 in 这两个状态，说明集群不在一个健康的状态。

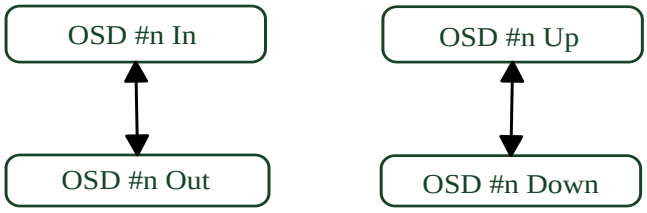

图 3.7 OSD 节点状态图

导致集群处于不健康状态有几个因素，包括了尚未启动集群服务或刚刚重启集群服务、刚刚添加或移除节点、刚刚更改了 CRUSH Map 等等。Ceph Monitor 能够定期 ping 一个 OSD 节点确保他是否在运行，然而，Ceph 同样赋予 OSD 节

点判定其相邻节点的状态是否为 down，并向 Monitor 报告的能力，然后 Monitor 更新 CRUSH Map。

（2） CRUSH Map 管理

RADOS 集群是通过操纵 CRUSH Map 来进行管理的，CRUSH Map 是一个小型的数据结构，描述了存储集群由什么样的 OSDs 组成和数据怎样映射到设备中去, 如图 3.8 所示。一个小型的，高可靠性的 Monitor 集群负责维护 CRUSH Map 和观察 OSDs 对集群改变的反应。因为 CRUSH Map 非常小，被集群共享，且完全指定了数据分配，因此客户端可以完全将存储集群（潜在的由数十到数千个节点组成）视作一个单独的逻辑对象存储节点。Monitor 集群用来操纵 CRUSH Map 从而管理整个存储集群。CRUSH Map 指定了哪些个 OSDs 包含在集群内，简单的指定了系统中数据在设备中的分布情况。它的副本会被系统中所有存储节点以及客户端所持有，进而与 RADOS 集群进行交互。因为 CRUSH Map 完全指定了数据的分布情况。客户端会暴露一个简单的接口，把整个存储集群当成一个单独的逻辑对象存储。

epoch:	map revision
up:	OSD $\mapsto$ { network address, *down* }
in:	OSD $\mapsto$ { *in*, *out* }
m:	number of placement groups $(2^k - 1)$
crush:	CRUSH hierarchy and placement rules

图 3.8 CRUSH Map

每当 CRUSH Map 因为 OSD 状态改变（设备故障）而改变时，map 的 epoch 值会随之增加。map epoch 允许所有系统组成部分商定当前的数据分配策略是什么。在一个大型系统中 OSD 发生故障和故障恢复的现象是极为普遍的，CRUSH Map 会因此而频繁地发生改变，更新信息（一个小的关于两个不同 epochs 的描述信息）随着 map epoch 的增长被分配出去。在大多数情况下，多个更新可能会被绑定在一起用来描述两个相隔较远的 map epochs，共同描述了多个设备状态的改变情况。

3. 自治的副本控制机制

RADOS 将每个对象复制到两个甚至更多的设备以确保数据的可靠性和可用性。OSD 负责序列化更新和写入复制，将客户端产生的由复制造成的网络开销转化为 OSD 集群内部的网络开销，从而可以有更高的带宽和更低的延迟。RADOS 实现了 primary-copy replication、chain replication 和 splay replication 三种对象副本分配方案，如图 3.9 所示。这三个方案都提供了强大的一致性保证。

（1） Primary-copy replication:通过第三章的介绍可以知道，对象可以通过 CRUSH 算法映射到 Placement Group（PG）中，PG 包含了一组可用的 OSD 设备，

PG 中的第一个 OSD 被称作 primary OSD，其他的 OSD 被称作 replicas OSD，客户端首先将对象写入 primary OSD 节点的 journal 中作为缓存。当写入完毕，该对象会被复制到各个 replicas OSD 节点并将对象副本写入 replicas OSD 节点的本地磁盘中，然后向 primary 节点作出响应。当所有的 replicas OSD 节点更新完毕，primary OSD 节点完成向本地磁盘的写入，并向客户端做出响应。

（2） Chain replication 将序列化更新与读过程分离。对象直接被写入 primary OSD 并完成本地磁盘写入。然后对象副本会被写入 PG 的下一个 replicas OSD 的本地磁盘中中。直到最后一个 replicas OSD 节点完成写入并向客户端作出响应。由此可见，chain replication 适合于数量相对较少的副本分配方案。

（3） Splay replication 结合了上面两种方案的关键要素，与 chain replication 类似，更新直接在头部进行，而且在尾部读取。对于需要复制多个副本这种情况，更新进行到中间 replicas OSD 时将会并行更新，从而降低了时延。

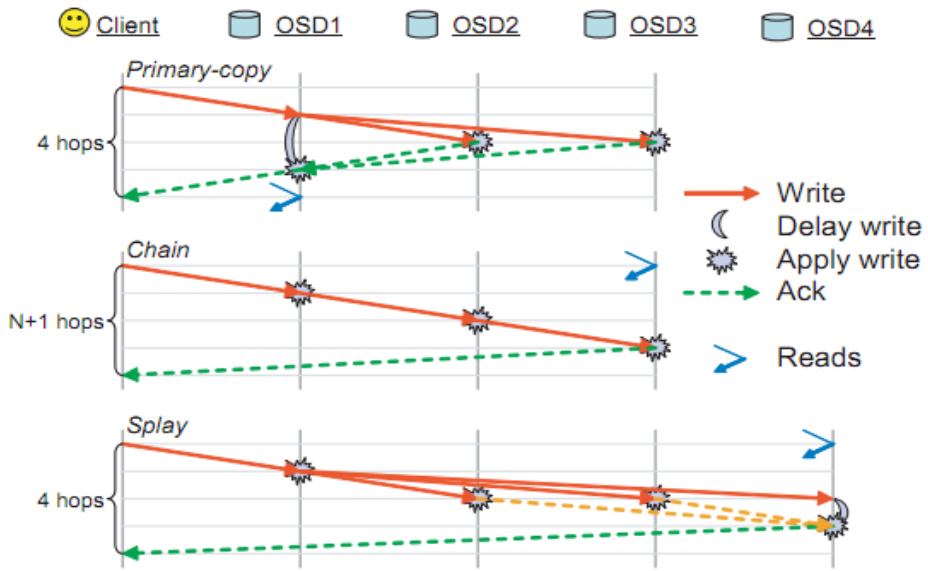

图 3.9 对象副本分配方案

4. 智能的故障检测机制

当一个集群初始化部署完毕，输入 ceph health 或 ceph –s 等命令，Ceph Monitor 节点会报告 Ceph 存储集群当前的状态。Ceph Monitor 节点通过 Ceph OSD 节点报告的状态来获取整个集群的状态。如果 OSD 节点的状态发生改变，Monitor 节点会更新 CRUSH Map 的状态。

默认情况下，每个 Ceph OSD 节点每隔 6 秒会检测相邻 OSD 节点的心跳，但可以对该时间间隔进行修改。如果相邻的 OSD 节点在 20 秒的宽限期内没有心跳反馈，该 OSD 节点会认为相邻 OSD 节点已经是 down 的状态并向 Monitor 节点进行报告，Monitor 会更新 CRUSH Map。OSD 心跳检测过程如图 3.10 所示。

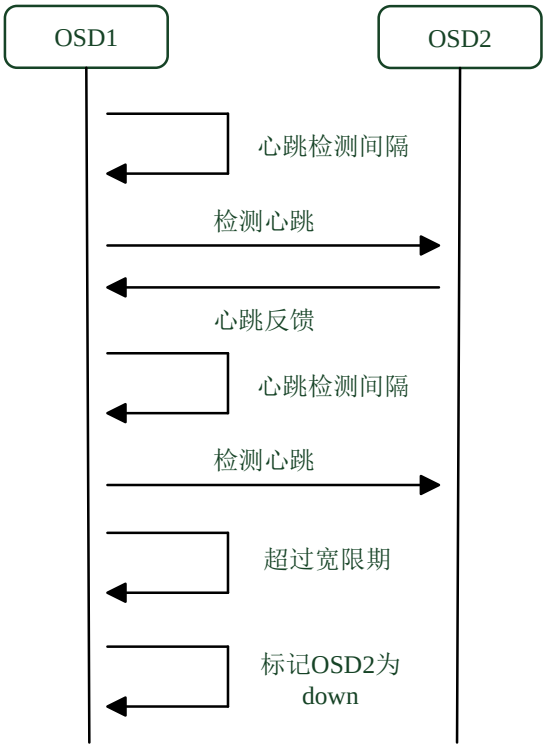

图 3.10 OSD 节点间心跳检测

默认情况下，一个 OSD 节点必须向 Monitor 节点报告三次相邻 OSD 节点的状态为 Down 时，Monitor 才会将该 OSD 节点到的状态标记为 Down，如图 3.11 所示。一般来讲用户可以指定报告的次数。

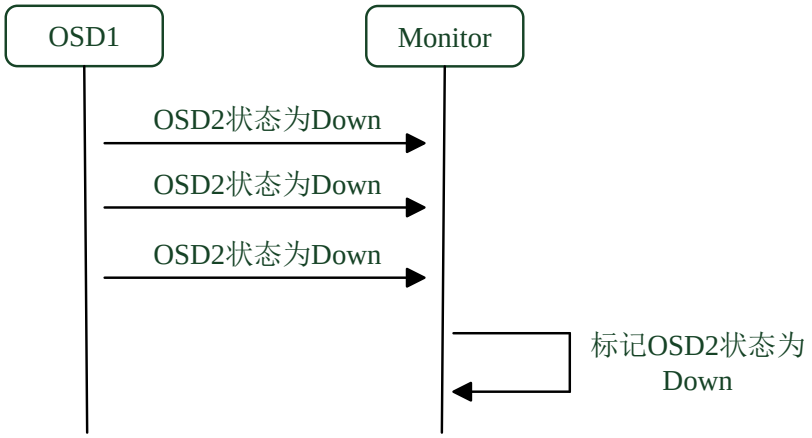

图 3.11 Monitor 心跳响应过程

3.2 测试业务流程

为了提高分布式存储系统性能测试的效率，分析分布式存储系统性能测试的目标，需要为分布式存储系统性能测试提出相关的测试业务流程模型，用业务流程模型来规范分布式存储系统性能测试的整个过程。分布式存储系统测试的业务

流程模型主要是针对分布式存储系统性能测试的过程给予指导，是性能测试成功实施的重要保障，该模型是在长期工作实践的基础上，并参考其他类似的工作模型综合而成。

Ceph 分布式文件系统是一个新兴的分布式文件系统，系统的可用性、扩展性以及性能等各方面指标需要通过测试来验证，通过测试分析这些性能指标及其变化趋势，判断与系统运行有关的性能状况，从而得出需要考察性能方面的结论，通过这些结论找到有效的方案来改善系统性能。性能测试能够暴露出分布式系统的性能瓶颈问题，收集大量的测试数据来帮助诊断和查明问题关键所在，最后起到优化性能的目的。流程按照时间顺序，对 Ceph 分布式性能测试的分为测试需求分析、测试方案制定与设计、测试执行、测试结果分析与系统调优四个阶段，如图 3.12 所示。

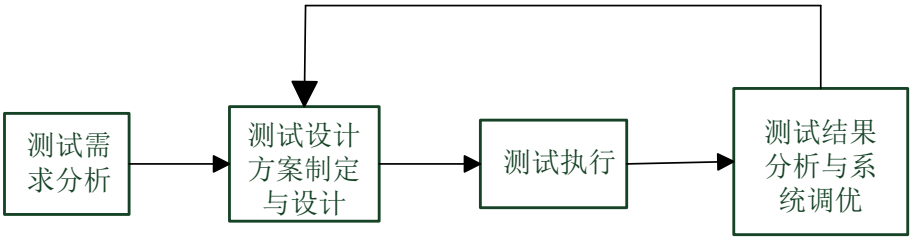

图 3.12 ceph 文件系统测试流程图

1. 测试需求分析阶段

测试需求分析即明确要“测试什么”。主要了解被测系统的体系结构、用户特征、业务特征、以及系统运行的环境，并明确性能测试的目标以及期望的系统性能指标或测试终止的标准，从而制定出详细的测试策略。

2. 测试方案制定与设计阶段

测试方案即要明确 “如何测试”。主要是根据测试需求分析的结果，形成性能测试执行方案。测试方案是对测试策略的进一步深化，主要对测试执行过程做出规定，测试方案越详细，测试就越明确。测试执行方案是后期测试工作开展的主要工作依据。据指定的测试方案设计出对应的测试场景，并根据不同的测试场景设计出相关的测试用例，准备好相关的测试脚本和测试数据等。

3. 测试执行阶段

根据制定的测试方案和设计的测试用例执行测试。主要工作包括建立测试环境、编写与调试测试脚本、部署测试场景、执行测试场景以及生成测试记录等等。

4. 测试结果分析与系统调优阶段

测试结果分析与系统调优阶段主要完成汇总、整理测试记录，分析形成的测试结果，结合测试目标，给出详尽的测试报告。在测试结果分析阶段，根据分析结果，可以对测试方案进行调整。确定系统性能参数，识别系统的瓶颈和缺陷。

通过对这些参数的调整与修改，找出系统的瓶颈和缺陷所在，优化调整系统存在的瓶颈和缺陷，提高系统性能。

3.3 测试需求分析

3.3.1 硬件需求分析

Ceph 文件系统由多种类型节点组成，其设计宗旨就是将这些节点运行在普通的商业硬件上，通过网络连接，从而构建并维护 PB 级别的数据存储集群。在计划构建自己的存储集群之前，需要综合考虑很多因素，比如需要考虑各种类型节点的硬件需求（例如节点的 CPU 规格，内存以及存储设备等），即什么样规格的硬件才能满足系统运行的要求，并将不同类型的节点运行在不同硬件类型的机器上面。

1. 节点 CPU 需求分析

Ceph 元数据集群需要动态的分配其工作负载，属于 CPU 密集型节点。所以元数据服务器应该具有强大的处理能力（比如采用四核或更高处理能力的 CPU）。由 Ceph OSD 节点构成的集群运行着 RADOS 服务，该服务利用 CRUSH 算法计算数据位置，维护并管理集群映射副本等。因此，Ceph OSD 节点应该具有比较好的处理能力（比如采用双核处理器）。Monitor 节点简单的维护集群映射的主副本，所以它们不是 CPU 密集型节点。除此之外还应该考虑到主机节点是否运行了 CPU 密集型的进程，例如，如果 Ceph 集群的某个节点上运行了用于计算的虚拟机（OpenStack Nova 等），就必须确保该节点具有强大的处理能力。一般情况下，应该将 CPU 密集型的进程运行在独立的节点上面。

2. 节点内存需求分析

Ceph 元数据服务集群和 Ceph Monitor 节点必须具有快速为其数据提供服务的能力，所以它们应该有足够大的内存（例如，每个节点 1G 内存）。对于常规的操作，Ceph OSD 节点不需要太大的内存，通常为每个节点分配 500MB 内存空间即可，但是在故障恢复过程中 Ceph OSD 节点需要更大的内存，通常 1TB 的存储容量需要分配 1GB 的内存空间，所以对 Ceph OSD 节点来讲，内存越大越好。

3. 数据存储需求分析

Ceph 存储集群从 Ceph 客户端接收数据，并以对象的方式存入集群中，每个对象相当于文件系统中的一个文件被存储到 OSD 节点中，集群中需要大量的 OSD 节点用来存储数据，因此如何权衡存储所需成本和存储性能是十分重要的。磁盘驱动受到寻道时间、访问时间、读写时间以及吞吐量等方面的限制，这些物

理方面的限制会影响到整个系统的性能，尤其是系统故障恢复期间。对某个硬盘并发的 OS 操作以及对该硬盘并发的读写请求操作都可能导致硬盘性能的降低。常见的存储设备的大致性能参数如表 3.1 所示。

表 3.1 常见存储设备性能参数表

类别	消耗的时间
千兆网络发送 1MB 数据	10ms
从内存顺序读取 1MB 数据	0.25ms
SATA 磁盘寻道	10ms
从 SATA 磁盘顺序读取 1MB 数据	20ms
固态硬盘 SSD 访问延迟	0.1~0.2ms

通常来讲，磁盘读写带宽还是不错的，15000 转的 SATA 盘的顺序读取带宽可以达到 100MB 以上，由于磁盘寻道的时间大约为 10ms，顺序读取 1MB 数据的时间为：磁盘寻道时间+数据读取时间，即 10ms+1MB/100MB/s=20ms。存储系统的性能瓶颈主要在于磁盘随机读写。设计存储系统的时候会针对磁盘的特性做很多处理，比如将随机写操作转化为顺序写，通过缓存减少磁盘随机读操作等。

固态磁盘（SSD）在最近几年得到越来越多的关注，各大互联网公司都有大量基于 SSD 的应用。SSD 的特点是随机读取延迟小，能够提供很高的 IOPS（每秒读写，Input/Output Per Second）性能。它的主要问题在于容量和价格，设计存储系统的时候一般可以用 SSD 来做缓存或者性能要求较高的关键业务。不同持久化存储介质对比如表 3.2 所示。从表中可以看出，SSD 单位成本提供的 IOPS 比传统的 SAS 或者 SATA 磁盘都要大很多，而且 SSD 功耗低、更加环保，适合小数据量并且对性能要求更高的场景。

许多导致 OSD 节点变慢的原因都是因为将操作系统、OSD 节点构建在同一个磁盘驱动上面，因此有必要为操作系统提供专有的硬盘，并使每一个 OSD 节点运行在单独的磁盘上。

表 3.2 不同持久化设备对比表

类别	每秒读写（IOPS）次数	每 GB 价格（元）	随机读取	随机写入
内存	千万级	150	友好	友好
SSD 盘	35000	20	友好	写入放大问题
SAS 磁盘	180	3	磁盘寻道	磁盘寻道
SATA 磁盘	90	0.5	磁盘寻道	磁盘寻道

3.3.2 功能测试需求分析

1. 可用性测试需求分析

系统可用性（availability）是分布式文件系统设计的首要目标。首先，系统应支持数据读写等基本功能，测试需要对提供给用户的存储接口进行测试。对 Ceph 分布式系统而言，它提供给用户三个接口，即 RADOS 对象存储接口，Ceph RBD 块设备存储接口以及 Ceph 文件系统。三种类型的接口适用于不同类型用户特征以及的使用场景，系统可用性测试应当对这三个接口分别进行压力测试，验证其性能。

2. 高可扩展性测试需求分析

系统的可扩展性（expansibility）指分布式系统通过扩展集群服务器规模来提高系统存储容量、计算量和性能的能力。基于对象的存储架构已经成为了提高存储集群扩展性的一种手段。然而，一些已经存在的系统依旧把存储节点当做被动元件，在基于本地或网络附属磁盘驱动的传统存储系统中，设备被动的响应读写操作，尽管在之前这些设备已经体现出了很高的智能性和自治性。当存储集群扩展到数千台设备，数据一致性管理、故障检测和故障恢复将会对客户端、控制器或元数据目录节点造成越来越大的负担，从而限制了系统的扩展性。

为了提供可靠的，高性能存储，系统规模的扩展成为了系统设计者当下的挑战。对于一个应用来讲，具有一个高吞吐量、低时延的文件系统是十分重要的。新兴出现的基于对象存储设备构建的集群存储架构，致力于将低级别的块分配的决定权交给智能存储设备本身，通过促进客户端直接对数据进行访问来简化数据分布并消除 IO 瓶颈。OSDs 由商业组件构成，其中集合了 CPU、网络接口、本地缓存和潜在的存储设备（磁盘或 RAID），将传统的基于块的存储接口替换为基于对象（对象有名字，且长度可变）的存储接口。

一个 PB 级别的存储系统需要动态的结构：系统以动态的方式构建，通过部署新的设备或废弃旧的设备来完成完成规模的扩展或缩减，设备故障和恢复总是持续发生，大量的数据被创建或者销毁。Ceph 文件系统的首要设计目标就是高可扩展性，以 Ceph 分布式文件系统为代表的主流分布式文件通过数据分布、复制以及容错等机制，都能够将存储集群扩展到数千台甚至数万台设备。作为云存储的基础，分布式系统的弹性扩展能力尤为重要。但是随着业务的发展，对底层存储系统的性能需求不断增加，比较好的方式就是通过自动增加服务器数量提高系统的能力。理想的分布式存储系统实现了“线性可扩展”，也就是说，随着集群规模的增加，系统的整体性能与服务器数量呈线性关系。分布式文件系统功能测试包括测试系统的弹性扩展能力，以及对系统扩展后对其性能产生的影响，验证随着存储系统集群设备的扩展系统性能是否具有线性扩展能力。

3.3.3 性能测试与系统调优需求分析

给定一个问题，往往会有多种设计方案，而方案评估的一个重要指标就是性能，性能是评估一个分布式文件系统的最为关键的因素，根据文件系统在不同场景下性能的表现判断文件系统是否适合特定的应用场景。常见的性能指标有：系统的吞吐能力以及系统的响应时间。其中，系统的吞吐能力是指在某一段时间内可以处理的请求总数，通常用每秒处理的读操作数（QPS，Query Per Second）或者写操作数（TPS，Transaction Per Second）来衡量；系统的响应延迟是指从某个请求发出到接受到返回结果消耗的时间，通常用平均延时或者 99.9%以上请求的最大延时来衡量。这两个指标往往是矛盾的，追求高吞吐的系统，往往很难做到低时延；追求低延时的系统，吞吐量也会受到限制。

分布式存储系统性能测试是收集分布式软件运行过程中各项性能指标，分析这些性能指标及其变化趋势，判断与系统运行有关的性能状况，从而得出那些需要考察性能方面的结论，通过这些结论找到有效的方案来改善系统性能。性能测试能够暴露出分布式系统的性能瓶颈问题，并收集大量的测试数据来帮助诊断和查明问题关键所在，最后起到优化系统性能的目的。分布式存储系统性能测试主要集中在以下四个方面：

1. 评估系统的能力

在真实的环境下检测系统性能，评估系统性能以及服务等级的满足情况，确认其是否可以达到用户的使用需求。

2. 识别系统的瓶颈和缺陷

受控负荷增加到一个极端水平，找出系统的瓶颈和缺陷所在，优化调整系统存在的瓶颈和缺陷，提高系统性能。

3. 确认系统的各项性能参数

在一定负荷下，确定系统的总体性能参数，包括所支持的最大并发用户数、事务处理成功率、请求响应时间、稳定运行时长等，确定在各个级别的负载及压力测试下服务器输出的具体性能参数，这些参数为系统评估与优化提供依据。

4. 系统调优

重复运行测试，验证调整系统的活动得到了预期的结果，从而改善系统性能。

3.3.4 网络部署及集群配置需求分析

1. 网络部署需求

图 3.13 为传统的数据中心网络拓扑，思科过去一直提倡这样的拓扑，分为三层，最下面是接入层（Edge），中间是汇聚层（Aggregation），上面是核心层

（core），典型的接入层交换机包含 48 个 1Gb 端口以及 4 个 10Gb 上行端口，汇聚层以及核心层的交换机包含 128 个 10Gb 的端口。

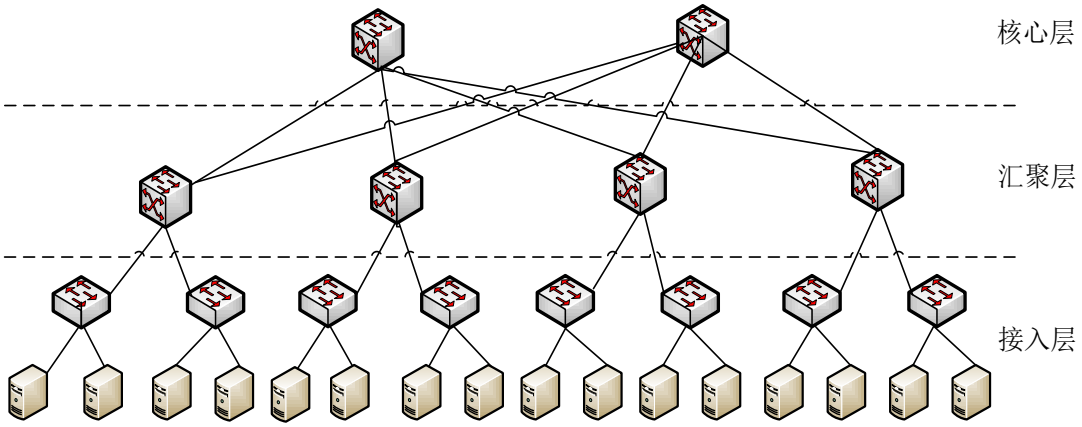

图 3.13 分布式集群三层网络拓扑图

对集群中网络的部署与规划合理与否，将会大大影响到集群的性能表现。现阶段，大多数商业硬盘都拥有 100Mb/second 的吞吐量，每个主机上的网卡应该能够处理其上 OSD 节点产生的流量。Ceph 文件系统使用 OSDs 集群与客户端通信完成数据读写，同时维护副本控制。当添加设备或设备移除时，会进行数据重组以恢复系统平衡。所有这些因素都会产生网络流量，加大网络负载。因此合理的部署集群的网络结构将变得十分重要。

2. 集群配置需求

分布式系统涉及的软硬件平台较多，这个系统中需要配置的参数也非常多，因此对如何进行配置才能使系统性能达到最优化提出了要求。Ceph 将客户数据以对象的形式存放到存储池中，首先通过 CRUSH 算法计算哪个 PG 将包含哪些对象，然后将每个 PG 映射到一组 OSD 中。因此对集群中 OSD 集群的配置，Placement Group 的配置都可能会影响到系统的性能。

3.4 本章小结

本章主要对 Ceph 文件系统进行了系统的介绍。首先介绍了 Ceph 文件系统的架构，描述了系统各个组成部分的功能与职责，对系统有一个宏观的了解。根据系统特性制定了测试业务流程，明确了针对系统要“测试什么”，“如何测试”等过程。并对 Ceph 文件系统进行了测试的需求分析，主要为硬件需求分析，功能及性能需求分析等。

第四章 Ceph 文件系统测试设计

本章根据第三章对 Ceph 文件系统可用性、扩展性以及集群部署方案等方面的需求分析，进行了测试用例设计方法的分析并设计了详细的测试用例，并根据对 Ceph 文件系统架构特性的分析，提出了集群性能优化的测试用例方案，为第五章测试过程打下基础。

4.1 测试用例设计方法

4.1.1 系统可用性用例设计方法

Ceph 存储集群为客户端提供了一组服务接口，如图 4.1 所示，它们分别是 RADOS 对象存储接口、Ceph RBD 块存储接口和 Ceph 文件系统存储接口。最底层是 RADOS 对象存储接口，其中包含了 OSD 节点和 Monitor 节点，一个 Ceph 存储集群最少包括两个 OSD 节点和一个 Monitor 节点，OSD 节点可以由硬盘、SSD、RAID 阵列等其他物理存储设备构建而成，它主要负责数据存储、处理数据复制、数据负载平衡、节点故障恢复并向 Monitor 节点提供其它 OSD 节点心跳检测的监控系统。Monitor 节点则维护了集群状态以及节点之间的映射关系，如 OSD 节点映射关系、Placement Group（PG）映射关系、CRUSH 算法映射关系等。

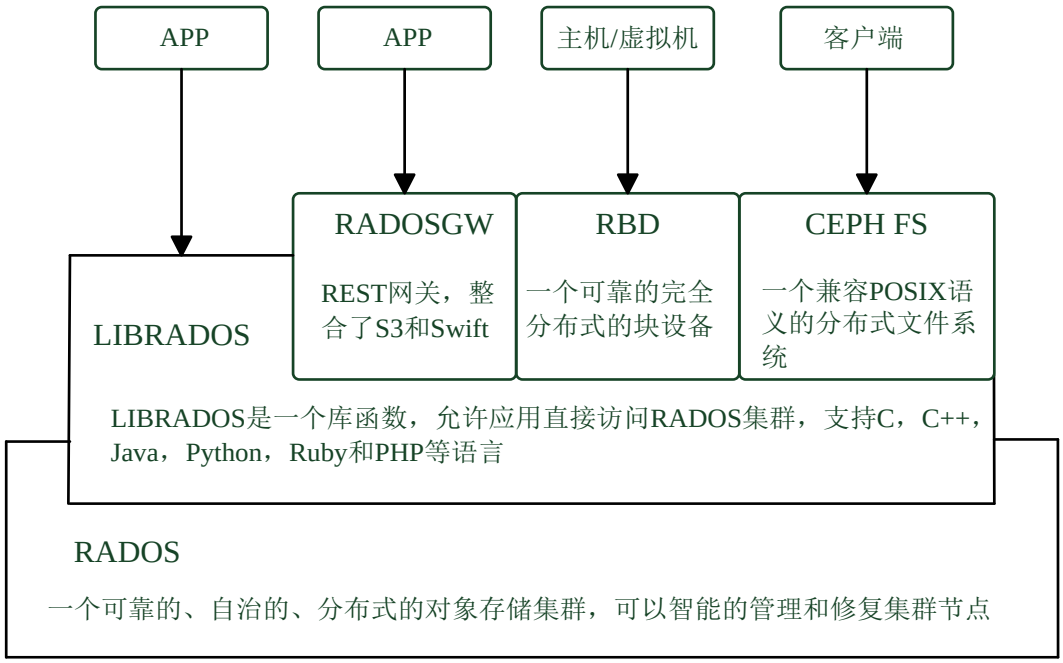

图 4.1 Ceph 客户端接口

Ceph 实现了分布式的对象存储，Ceph 可供客户端应用程序直接访问基于对

象的可靠的、自治的分布式对象存储系统（RADOS）。RADOS 是 Ceph 实现分布式存储的根本，所有的存储接口都是基于 RADOS 实现的，RADOS 本身就是一个对象存储接口，Ceph 提供的库函数可以使客户端应用直接访问 RADOS，它自身维护了集群的状态并实现了数据分发的要求，我们通常也将 RADOS 称为 Ceph 存储集群，因为其上的存储接口如 RBD，CephFS 都是基于 RADOS 的接口而实现的。

在 RADOS 层之上的是 Librados 库函数层，Librados 库函数提供了对 RADOS 的直接访问。Librados 库函数提供对 C、C++、Java、Python、Ruby 和 PHP 的支持。

系统可用性主要针对 Ceph 分布式文件系统的两个接口即 RADOS 对象存储接口以及 Ceph RBD 块存储接口进行测试。下面分别介绍。

1. RADOS 对象存储接口可用性用例分析

Ceph 的 RADOS 对象存储提供了 RESTful HTTP 应用程序接口来存储对象和元数据。它位于位于 Ceph 存储集群的最顶层，拥有自己的数据形式，维护自己的用户数据库、权限及访问控制。RADOS Gateway 使用一个统一的名字空间，这意味着用户可以使用 OpenStack Swift-compatible 应用程序接口或者 Amazon S3-compatible 应用程序接口。例如一个应用使用 S3-comptible API 做数据写入，另一个应用使用 Swift-compatible API 做数据读取。

首先对 RADOS 对象存储接口进行压力测试，客户端向 RADOS 集群做对象的读取与写入操作，如图 4.2 所示，本次测试采用 Ceph 内部构建的命令行基准测试工具“RADOS bench”，RADOS bench 工具有一定的优点，它能够明确的给出不同大小的对象的读写速度。通过 RADOS bench 指定不同的对象大小，不同的客户端并发访问的数量以及压力测试的时间，设计多组用例测试 RADOS 对象存储接口的吞吐量。

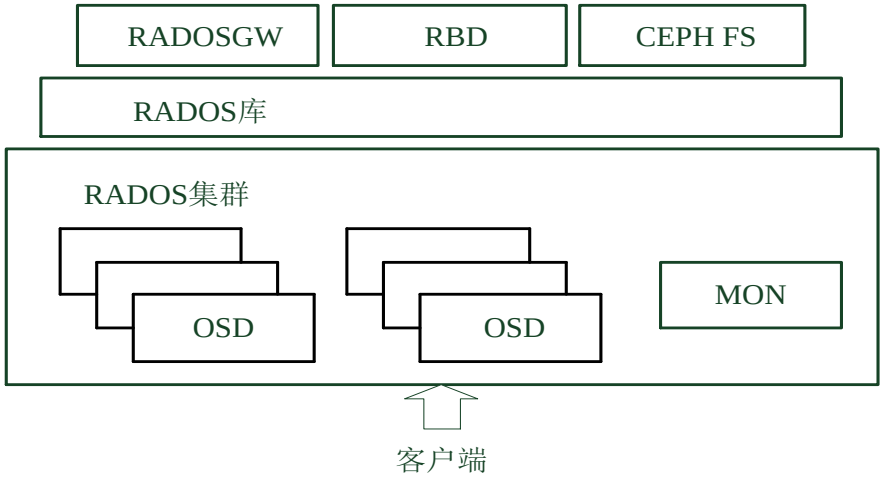

图 4.2 RADOS 集群测试示意图

从前面的分析可以得知，Ceph 存储集群从 Ceph 客户端接受数据（客户端可以是 Ceph 对象存储接口、Ceph RBD 块设备接口、Ceph 文件系统），集群以对象的方式存储数据。每个对象对应文件系统中的一个文件存储在 OSD 节点中，如图 4.3 所示，OSD 节点负责对象在存储设备上的读写操作。因此 Ceph OSD 节点非常倚重底层文件系统的稳定性和性能。目前主流的文件系统包括了 XFS[24] 文件系统、btrfs[25]文件系统、EXT4[26]文件系统等。在高可扩展的数据存储集群环境中，XFS 文件系统和 btrfs 文件系统与 EXT4 文件系统相比较而言统具有更多的优势，XFS 和 btrfs 文件系统都属于日志文件系统，这意味着当系统遇到崩溃以及断电时，它们能够提供更加健壮的恢复措施。btrfs 文件系统属于写时复制式文件系统，相对于 XFS 和 EXT4 文件系统而言它提供了新的特性，例如支持递归的快照功能（snapshots of snapshots）、支持内建磁盘阵列支援、支持子卷的概念、支持在线调整文件系统大小等。本次测试会以上述三种文件系统作为 OSD 节点的底层文件系统分别进行测试，观察采用三种不同文件系统对系统的吞吐量的影响。

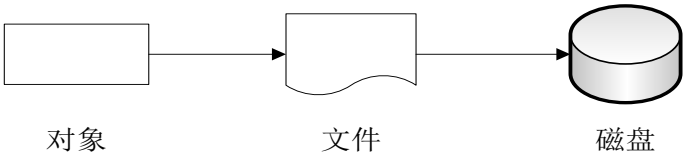

图 4.3 对象交互过程

2. Ceph RBD 块存储接口可用性用例分析

一个块是一个字节序列（例如，一个 512 字节的数据块），基于块的存储接口是最为常见的存储数据的方式。无处不在的块设备接口使得虚拟块设备成为了与大型存储系统交互的理想方式。

Ceph RADOS Block Device（又称 Ceph RBD 块存储）是 Ceph 的另一大支撑点，目前可以为虚拟机和主机提供不同路径的块存储。CephRBD 块设备存储接口配置精简，可伸缩，它利用了 RADOS 集群的一些性能特征，例如快照，副本控制及一致性。当应用通过块设备将数据写入 Ceph 集群时，Ceph 会自动的将数据条块化并复制到集群中去，Ceph 的 RADOS 块设备（RBD）同内核虚拟机（KVMs）集成在了一起，Ceph RBD 块存储接口利用内核模块或者是 librbd 库与 OSDs 进行交互。

首先测试 Ceph RBD 块设备在物理机器上的性能，客户端向 RBD 块设备接口做文件的读取与写入操作，如图 4.4 所示，分别对 Ceph RBD 裸设备以及用 EXT4、XFS 文件系统格式化后的设备进行随机读、随机写、随机读写、顺序读以及顺序写系统的 IOPS 值。并进一步更改 RBD 配置，找到最优化的解决方案。

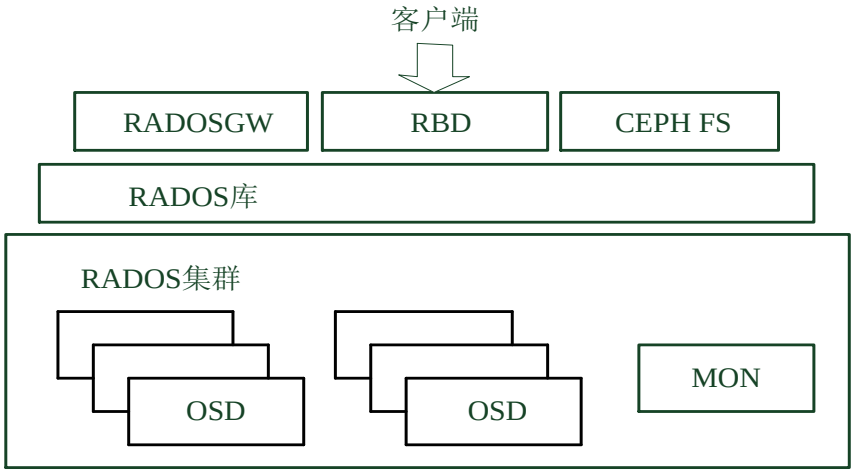

图 4.4 RBD 接口测试示意图

4.1.2 系统扩展性用例设计方法

可扩展性是指网络、系统或者进程能过通过扩充自身资源来支持更大的工作负载的能力。可扩展性可分为 4 个类别：负载可伸缩性、空间可伸缩性、时空可伸缩性和结构可伸缩性。负载可伸缩性是指系统能够平滑的在变化的工作负载下过度，通过调节资源来满足变化的负载，在调节的过程中不会产生超过规定限度的时间延迟和资源占用。空间可伸缩性是对系统调节资源的一个限制，即在负载增大的情况下，所占用的资源（内存、磁盘空间等）最多和负载呈线性增长，禁止了通过无穷无尽的增加资源来支撑负载的做法。时空可伸缩性是指在给定的运行时间要求下，通过扩展资源来满足增大的负载，在扩展的同时应用的运行时间依旧满足要求。结果可伸缩性是指实现的结构能够很容易的进行资源上的扩充和调节。

在实际应用中，可扩展性被定义为通过增加资源使服务性能产生线性增长的能力。如图 4.5 所示，有两种增加资源的方法；一种是上扩，一种是外扩。上扩是指给单个计算节点使用性能更好、速度更快的成本更高的硬件来增加资源，包括添加更多内存、添加更多更快的处理器，或者只是将应用程序迁移到功能更强大的单个计算机。外扩是增加计算节点的数目来增加资源。外扩增加了计算节点的数目，对管理体处理更大的挑战。增加资源时，外扩比上扩更容易达到最佳的可伸缩性；而且外扩这种方式可以更加灵活地增加或者减少计算节点。

Ceph 文件系统是一个高度可扩展的分布式文件系统，通常可以扩展到数千甚至上万台设备，因此系统性能能否随着集群扩展而呈线性提升的态势是测试应该关注的方面。我们分别测试 Ceph RADOS 存储集群和 Ceph RBD 接口的扩展性，通过增加单个主机上面 OSD 的个数以及主机个数测试系统的吞吐量以及 IOPS，观察系统的扩展性情况。因为实验资源有限，测试只能对系统扩展性进

行一个简单的评估。

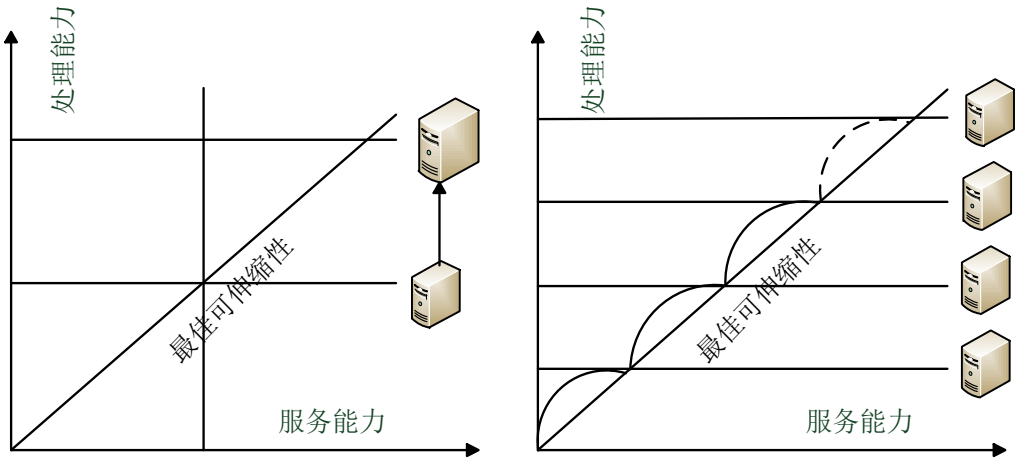

图 4.5 上扩和外扩

4.1.3 系统优化用例设计方法

1. 网络配置优化用例设计

网络设置对于构建一个高性能的 Ceph 存储集群来讲是至关重要的[27]。通过之前对 Ceph 文件系统结构的分析可知，Ceph 存储集群不会派发或调度 Ceph 客户端对集群的请求，Ceph 客户端会直接向 Ceph OSD 节点发送请求。Ceph OSD 集群会为 Ceph 客户端做数据的复制，这就意味着数据复制、故障恢复等其它因素产生的负荷会增加 Ceph 存储集群的网络负载。Ceph 系统默认采用如图 4.6 所示的网络架构，即客户端与 OSD 节点共享同一个公共网络。

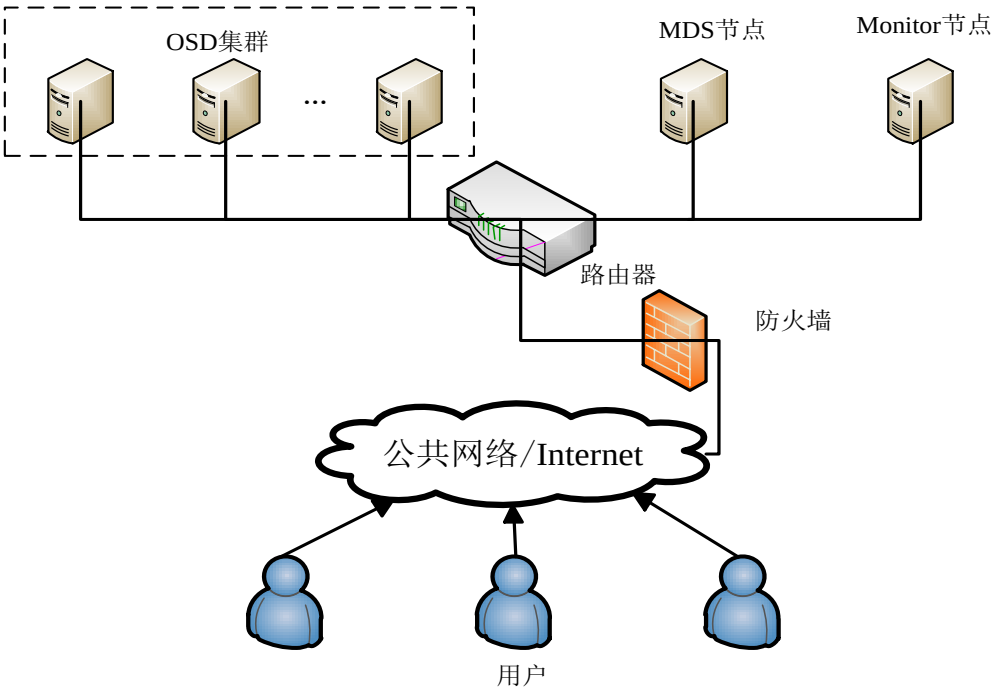

图 4.6 Ceph 集群默认网络架构

然而，采用这样的网络架构会有两个弊端：一是出于性能方面的考虑，Ceph OSD 集群负责为 Ceph 客户端数据做数据备份，当 Ceph OSD 集群处理的数据副本量超过一个，Ceph OSD 节点之间的网络负荷会大大增大 Ceph 客户端和 Ceph 存储集群之间的网络负担，这样会引入网络延迟和性能问题。OSD 节点间故障恢复、负载平衡和心跳检测同样会引入显著的问题。二是出于安全方面的考虑，如果集群受到拒绝服务攻击（Dos，Denial of Service）[28]，OSD 节点之间的通信将会被阻断，从而影响用户正常的数据读写过程。综合以上两点，有必要为 OSD 集群从新规划网络，在提高性能的同时提高系统的安全性。这里，我们提出一个新的网络架构，如图 4.7 所示。

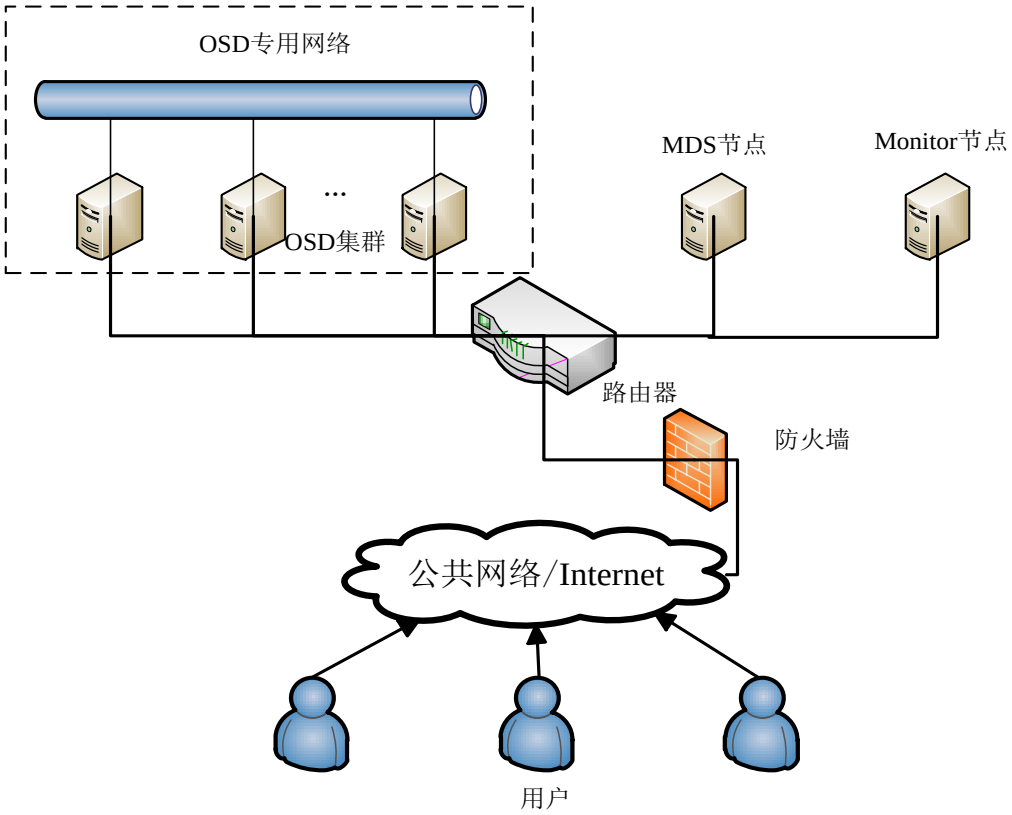

图 4.7 加入 OSD 专用网络的网络架构

即为 Ceph 存储集群配置两个网络：一个公共网络，负责客户端与 OSD 节点之间的通信，和一个集群内部网络，负责 OSD 节点之间的通信。为了支持这个设计，每个 OSD 节点应该具有两块网卡，一块用于连接客户端，另一块用于连接 OSD 节点。通过测试对比采用两种不同网络架构时系统的性能表现，验证网络优化用例设计的正确性。

2. Journal 存储优化用例设计

RADOS 将每个对象复制到两个甚至更多的 OSD 节点以确保数据的可靠性和可用性。OSD 节点分为 Journal 存储和数据存储两部分，Journal 具有以下功能：

1）确保数据和事物的一致性，它起到了传统文件系统 Journal 的作用；2）提供了原子事物，它跟踪了已经提交的事物和即将提交的事物；3）数据以顺序的方式写入 Journal；4）Journal 以先入先出的方式工作。

客户端首先将对小的，随机的 IO 顺序的写入 primary OSD 节点的 Journal 中作为缓存。当写入完毕，该对象的副本会同步的被复制到各个 replicas OSD 节点并将对象副本写入 replicas OSD 节点的磁盘中，然后向 primary 节点作出响应。当所有的 replicas OSD 节点更新完毕，primary OSD 节点完成向磁盘的写入，并向客户端做出响应。因此如果保证 Journal 高效而快速的读写性能可能会对系统性能产生一定的影响。通过改变 Journal 的存储方式对系统进行优化。

传统的方法是将 Journal 和数据对象存储在 OSD 节点普通硬盘的不同分区中，如图 4.8 所示。但是因为对象写入会分为两个过程，采用该方案会导致如图 4.9 所示的结果，很明显，采用 writehead 模式的对象写入方式占用磁盘带宽，从而影响系统的性能，导致了巨大的性能损失。

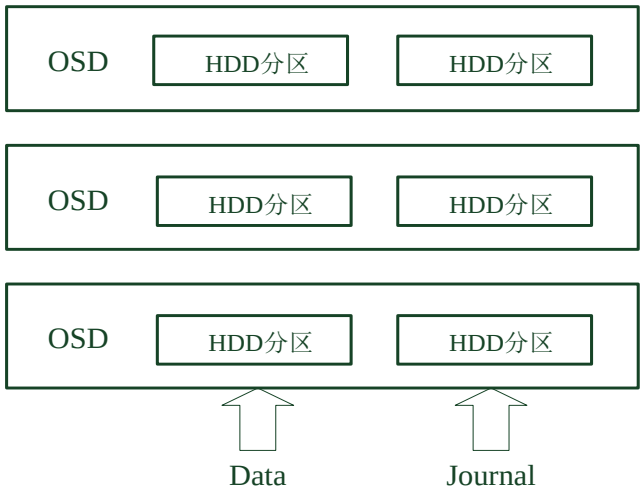

图 4.8 传统 OSD 部署方式

Device:	vMB/s
sdb1 – journal	50.11
sdb2 – osd_data	40.25

图 4.9 磁盘 IO

存储系统的性能主要包括两个维度：吞吐量以及访问延时。相对于物理机械磁盘来讲，固态硬盘 SSD 没有物理机械部件，所以它们不会遭受到与机械硬盘相同的限制，所以可以大幅度的减少随机访问时间和读写延迟，从而提升系统的吞吐量。按每 Gb 来讲，SSD 的价格通常是普通硬盘的 10 倍，但是随机访问的时间也要比普通硬盘快 100 倍。磁盘和 SSD 的访问延时差别很大，但带宽差别不大，因此，普通机械磁盘适合大块顺序访问的存储系统，而 SSD 则适合随机

访问较多或者时延比较敏感的关键系统。二者也常常组合在一起进行混和存储，热数据（访问频繁）存储到固态硬盘 SSD 中，冷数据（访问不频繁）存储到普通机械磁盘中。

在这里，我们使用 SSD 负责存储管理 Journal，使用普通 SATA 磁盘存储文件数据，如图 4.10 所示。通过测试对比两种 Journal 存储方案的性能差异，验证推论的正确性。

在选取 SSD 时有以下几点需要注意：

（1） 写密集型语义

Journal 具有写密集型语义，所以应该确保所选择的 SSD 在数据写入方面的性能要高于普通机械硬盘，一些价格低廉的 SSD 在加速访问时间的同时可能会引入写入延迟，这种现象是应该避免的。

（2） 顺序写入

当在单独的 SSD 上存储多个 OSD 的 Journal，SSD 同时处理多个 OSD Journal 的写入请求，就必须考虑 SSD 顺序写入的局限性。

（3） 磁盘分区对齐

与 SSD 性能相关的一个关键问题就是人们常常将磁盘分区作为最佳实践，但常常忽略了 SSD 合理分区对齐的重要性，这可能导致 SSD 的数据传输效率大大降低，因此，要确保合理的将 SSD 分区对齐。

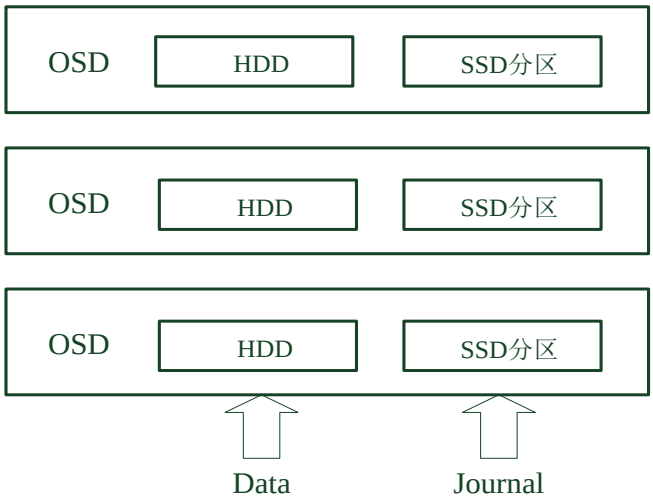

图 4.10 新的 OSD 部署方式

4.2 测试用例设计

4.2.1 RADOS 接口基准测试用例设计

首先构建 Ceph 文件系统存储集群，测试工具采用 Ceph 内部构建的命令行

基准测试工具“RADOS bench”，通过 RADOS bench 指定不同的对象大小、不同的客户端并发访问的数量、以及压力测试的时间向 RADOS 集群产生压力。同时将 OSD 格式化为不同的文件系统类型，测试不同文件系统下 RADOS 对象存储接口的性能表现。采用三个类型的测试用例，它们分别是：

1. 客户端并发访问数量

分别模拟 16 个客户端和 256 个客户端并发访问 RADOS 接口，完成对象的读写。

2. 对象块大小

分别模拟 4M，128K 和 4k 的对象块完成向 RADOS 集群的读写过程。

3. 文件系统类型

OSD 的底层文件系统分别采用 btrfs，EXT4 和 XFS 文件系统，测试不同文件系统类型对对象读写性能的影响。

具体的测试用例如表 4.1 所示。

表 4.1 RADOS 接口基准测试用例表

对象大小	客户端并发访问线程数量	OSD 底层文件系统类型	完成测试内容	集群对象副本数量
4K	16 个	XFS/EXT4/Btrfs	对象读写速率	1
4K	256 个	XFS/EXT4/Btrfs	对象读写速率	1
128K	16 个	XFS/EXT4/Btrfs	对象读写速率	1
128K	256 个	XFS/EXT4/Btrfs	对象读写速率	1
4M	16 个	XFS/EXT4/Btrfs	对象读写速率	1
4M	256 个	XFS/EXT4/Btrfs	对象读写速率	1

4.2.2 Ceph RBD 接口基准测试用例设计

首先搭建 Ceph 文件系统集群，在 Ceph 客户端创建 Ceph RBD，设备，并将 RBD 设备与客户端建立映射关系，向 RBD 设备进行文件的写入与读取，测试 Ceph 文件系统的吞吐量。分别有几个参数变量，它们分别是：

1. 文件系统类型

Ceph RBD 设备初始是没有文件系统格式化的，分别使用 EXT4 和 XFS 文件系统格式化 Ceph RBD 设备，并测试基于不同文件系统情况下系统的吞吐量。

2. 客户端 RBD 设备数量

分别在单个客户端建立一个，两个，三个和四个 RBD 设备，同时向 RBD 设备读写文件，测试 RBD 设备数量对系统吞吐量的影响。

具体的测试用例如表 4.2 所示。

表 4.2 RBD 接口测试用例表

客户端数量	单个客户端 RBD 设备量	RBD 文件系统类型	测试文件大小	单次 IO 大小	测试线程数量	完成测试内容
1	1/2/3/4	RBD 裸设备/XFS/EXT4	30G	4K	64	随机读
1	1/2/3/4	RBD 裸设备/XFS/EXT4	30G	4K	64	随机写
1	1/2/3/4	RBD 裸设备/XFS/EXT4	30G	4K	64	随机读写
2	1	RBD 裸设备/XFS/EXT4	30G	4K	64	随机读
2	1	RBD 裸设备/XFS/EXT4	30G	4K	64	随机写
2	1	RBD 裸设备/XFS/EXT4	30G	4K	64	随机读写

4.2.3 RADOS 集群优化用例分析与设计

这里采用四种不同的网络和磁盘OSD Journal用例，对RADOS集群进行优化，观察更改配置后系统的性能。四种配置分别是：

1．不使用SSD，而是使用普通SATA硬盘作为Journal的存储方案；只使用一个共有网络，而不使用OSD集群专有网络分担副本分配和数据重组的网络流量。

2．不使用SSD，而是使用普通SATA硬盘作为Journal的存储方案；使用共有网络并使用OSD集群专有网络分担副本分配和数据重组的网络流量。

3．使用SSD作为Journal的存储方案；只使用一个共有网络，而不使用OSD集群专有网络分担副本分配和数据重组的网络流量。

4. 使用SSD作为Journal的存储方案；使用共有网络并使用OSD集群专有网络分担副本分配和数据重组的网络流量。

具体测试用例如表4.3所示。

表4.3 RADOS接口优化测试用例表

对象大小	客户端并发访问线程数量	OSD 底层文件系统类型	Journal 存储方案	集群网络部署方案	完成测试内容	集群对象副本数量
4M	256	XFS	SATA 硬盘分区	仅使用公共网络	对象读写速率	1 个/3 个

表4.3 RADOS接口优化测试用例表（续）

4M	256	XFS	SATA 硬盘分区	集群公共网络，OSD 专用网络	对象读写速率	1 个/3 个
4M	256	XFS	SSD 分区	仅使用公共网络	对象读写速率	1 个/3 个
4M	256	XFS	SSD 分区	集群公共网络，OSD 专用网络	对象读写速率	1 个/3 个

4.2.4 集群扩展性用例分析与设计

在经过优化的测试环境上对 RADOS 对象存储接口进行扩展性测试，即采用 SSD 作为 Journal 存储方案并加入 OSD 专用网络，分别在集群中放置 3 个，6 个，9 个，12 个，15 个，18 个 OSD 节点，并向 OSD 集群写入 4M 块大小的对象，观察集群随着 OSD 数量变化性能的改变。具体测试用例如表 4.4 所示。

表 4.4 RADOS 对象存储接口扩展性测试用例

对象大小	客户端并发访问数量	OSD 底层文件系统类型	Journal 存储方案	网络部署方案	集群 OSD 数量	集群对象副本数量	完成测试内容
4M	256	XFS	SSD 分区	集群公共网络，OSD 专用网络	3/6/9/12/15/18	1	读写速率

在经过优化的测试环境上，即采用 SSD 作为 Journal 存储方案并加入 OSD 专用网络，对 Ceph RBD 存储接口进行扩展性测试，分别在集群中放置 3 个，6 个，9 个，12 个，15 个，18 个 OSD 节点，观察客户端随着 OSD 数量变化读写性能的改变。具体测试用例如表 4.5 所示。

表 4.5 Ceph RBD 存储接口扩展性测试用例

客户端数量	单个客户端 RBD 设备量	RBD 文件系统类型	集群 OSD 数量	测试文件大小	单次 IO 大小	测试线程数量	完成测试内容
1	1	RBD 裸设备	3/6/9/12/15/18	30G	4K	64	随机读/随机写

4.3 本章小结

本章的主要内容是为 Ceph 文件的两种客户端接口 Ceph 对象存储接口以及 Ceph RBD 块存储接口的功能及基准性能测试设计了测试用例。并设计了对 RADOS 集群进行优化的测试用例，即网络环境配置测试用例的设计以及 Journal 存储配置用例的设计。为下一步系统性能测试做准备。

第五章 Ceph 文件系统测试优化及结果分析

本章首先介绍了 Ceph 文件系统测试环境的搭建过程，对各个节点软硬件环境以及整个系统的网络环境进行构建，然后对测试环境进行了基本的验证，节点硬件基准测试和网络环境基准测试，最后根据第四章设计的测试用例对 Ceph 文件系统的性能、可用性以及扩展性进行了测试，详细的分析了测试结果，并对系统性能提出了优化方案。

5.1 测试环境准备

根据对 Ceph 分布式文件系统架构的分析，集群中应该包含四种类型的节点，它们分别是客户端节点、OSD 节点集群、Monitor 节点以及 MDS 节点，用这四种节点通过网络连接构建一个完整的 Ceph 测试集群。

5.1.1 Ceph 测试集群软硬件配置

测试环境由 9 台主机构成，其中两台主机作为客户端节点，6 台主机组成 OSD 集群，其中每个主机中有三个 OSD 节点，这样的话集群中共有 18 个 OSD 节点，Monitor 节点和 MDS 节点构建在一台主机上，测试集群的网络拓扑图如图 5.1 所示。

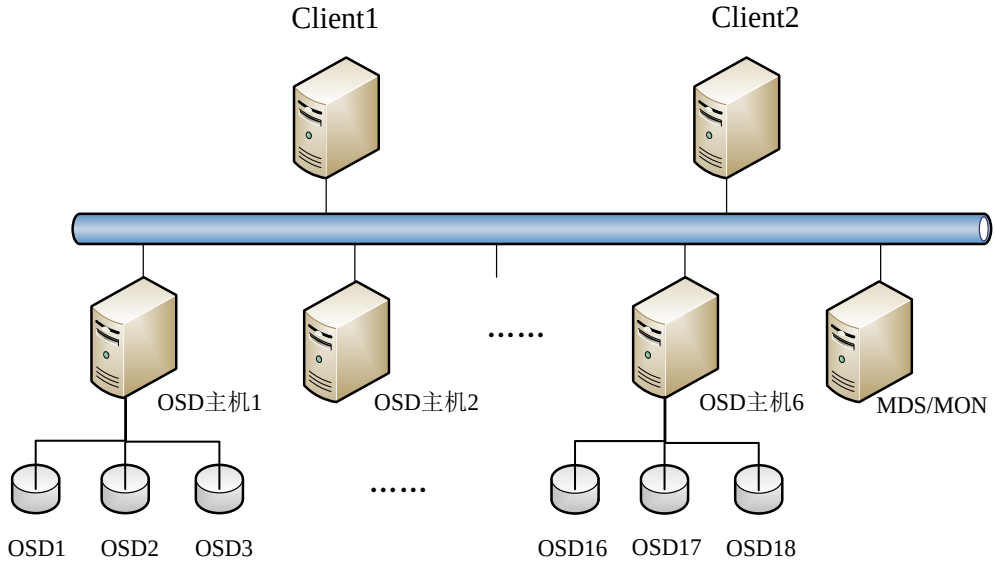

图 5.1 测试集群拓扑图

Ceph 是可以构建在普通商用硬件设施上的大型分布式文件系统，通过软件方式自动容错，保证系统的可靠性和可用性。节点硬件环境配置，包括了 CPU，内存，磁盘，网卡以及路由器等，具体的硬件型号及版本信息如表 5.1 所示。集

群中的主机只需安装操作系统并安装并部署 Ceph 文件系统即可，具体版本信息如表 5.2 所示。

表5.1 节点硬件情况

CPU	2x Xeon 5580/E5-2680
内存	12G DDR3 1600MHz
磁盘	4x 7200RPM SATA Disk/1x 300G SSD
网卡	10 GbE 82599EB Ethernet/1 GbE E1G42ET Ethernet
路由器	Cisco 10G Router Catalyst 4900M V12

表5.2 操作系统及Ceph版本情况

操作系统	Ubuntu 12.0.4 64Bit
Ceph文件系统版本	Ceph 0.56
单节点OSD数量	3

5.1.2 Ceph 测试集群网络配置

Ceph的OSD集群需要配置两个网段，一个是公共网络，一个是OSD集群私有网络，测试环境中10.0.0.X网段连接着10Geb路由器，192.168.3.X网段连接着1Geb网段，具体的IP配置信息如表5.3所示。

表5.3 节点IP部署情况

IP	主机名	组件名
10.0.0.19/192.168.3.19	NEW-OSD0	OSD
10.0.0.20/192.168.3.20	NEW-OSD1	OSD
10.0.0.6/192.168.3.6	NEW-OSD2	OSD
10.0.0.15/192.168.3.15	NEW-OSD3	OSD
10.0.0.101/192.168.3.101	NEW-OSD4	OSD
10.0.0.102/192.168.3.102	NEW-OSD5	OSD
10.0.0.22/192.168.3.22	NEW-MDS	MDS/MDS
10.0.0.7	Client1	Client
10.0.0.8	Client2	Client

5.1.3 测试集群搭建过程

1. 安装 Ceph 文件系统

在各个节点分别获取最新的 Ceph 安装包，更新系统并安装 Ceph 文件系统。

执行脚本如表 5.4 所示。

表 5.4 Ceph 文件系统安装脚本

```
wget -q -O- 'https://ceph.com/git/?p=ceph.git; f=keys/release.asc'
| sudo apt-key add -
echo deb http://ceph.com/debian-testing/ $(lsb_release -sc) main |
sudo tee /etc/apt/sources.list.d/ceph.list
sudo apt-get update && sudo apt-get install ceph
sudo apt-get update && sudo apt-get upgrade
```

2. 编写配置文件

当启动一个 Ceph 存储集群之前，每个节点都会查找 Ceph 配置文件（ceph.conf）,该配置文件提供了集群的配置信息。如果手动部署，就需要创建一个 Ceph 配置文件。配置文件指定了一些信息：集群标识符、验证设置、集群成员、主机名、主机 IP 地址、数据存储路径、Journal 存储路径以及其它一些运行时选项。OSD 节点数据默认存储路径为 var/lib/ceph/osd/$cluster-$id，Journal 默认存储路径为 var/lib/ceph/osd/$cluster-$id/journal 测试集群配置文件如表 5.5 所示。

表 5.5 Ceph 集群配置文件

```
[global]    auth cluster required = none
            auth service required = none
            auth client required = none
[osd]       osd journal size = 1000
            osd journal = /srv/ceph/journals/osd$id/journal
[mon.a]     host = NEW-MDS
[osd.0]     host = NEW-OSD0
[osd.1]     host = NEW-OSD1
[osd.2]     host = NEW-OSD2
[osd.3]     host = NEW-OSD3
[osd.4]     host = NEW-OSD4
[osd.5]     host = NEW-OSD5
…
[osd.15]    host = NEW-OSD3
[osd.16]    host = NEW-OSD4
[osd.17]    host = NEW-OSD5
[mds.a]     host = NEW-MDS
```

3. 创建数据存储路径

在 Ceph OSD 主机上为每个 OSD 节点创建数据存储目录、格式化磁盘、并将经过格式化的磁盘挂载在各个目录下，以 NEW-OSD0 主机为例，构建脚本过程如表 5.6 所示。

表 5.6 创建数据存储路径脚本

```
for i in 0 1 2; do mkdir -p /var/lib/ceph/osd/ceph-$i;done#创建数据存储目录
mkfs.xfs /dev/sdc1 –f                                    #格式化磁盘
mkfs.xfs /dev/sde1 -f                                    #格式化磁盘
mkfs.xfs /dev/sdf1 –f                                    #格式化磁盘
mount /dev/sdc1 /var/lib/ceph/osd/ceph-0                 #挂载
mount /dev/sde1 /var/lib/ceph/osd/ceph-1                 #挂载
mount /dev/sdf1 /var/lib/ceph/osd/ceph-2                 #挂载
```

4. 创建 Journal 存储路径

在 Ceph OSD 主机上为每个 OSD 节点创建 Journal 存储目录、格式化磁盘、并将经过格式化的磁盘挂载在各个目录下，以 NEW-OSD0 主机为例，构建脚本过程如表 5.7 所示。

表 5.7 创建日志存储路径脚本

```
for i in 0 1 2;do mkdir -p /srv/ceph/journals/osd$i;done #创建 Journal 存储目录
mkfs.xfs /dev/sdb1 -f                                    #格式化磁盘或分区
mkfs.xfs /dev/sdb2 -f                                    #格式化磁盘或分区
mkfs.xfs /dev/sdb3 –f                                    #格式化磁盘或分区
mount /dev/sdb1 /srv/ceph/journals/osd0                  #挂载
mount /dev/sdb2 /srv/ceph/journals/osd1                  #挂载
mount /dev/sdb3 /srv/ceph/journals/osd2                  #挂载
```

5. 编译并启动系统系统

执行如表 5.8 所示语句编译整个系统，编译成功后启动系统并查看系统状态。

表 5.8 集群编译脚本

```
cd /etc/ceph
mkcephfs -a -c /etc/ceph/ceph.conf -k ceph.keyring       #编译系统
Sudo service ceph –a start                               #启动系统
Ceph heath                                               #查看系统状态
```

5.2 集群环境测试

为了确保网络设备，磁盘等硬件设备能够正常运行，且其性能不会对系统运行造成瓶颈，所以有必要对系统执行的硬件及网络环境进行基准测试。

1. 网络环境验证

作为基准环境验证之一，该测试的目的在于验证存储节点与存储节点之间以及存储节点与客户端之间的网络性能，在该测试集群中，10.0.0.X 网段连接着 10Geb 路由器，192.168.3.X 网段连接着 1Geb 路由器，在这里我们将测试节点间的网路带宽，为之后的工作提供指导性意见。

在这里，使用 Linux 下的网络测试工具 iperf[29]。iperf 可以测试 TCP 和 UDP 带宽质量。iperf 可以测量最大 TCP 带宽，具有多种参数和 UDP 特性。iperf 可以报告带宽，延迟抖动和数据包丢失。利用 iperf 这一特性，可以用来测试一些网络设备如路由器，防火墙，交换机等的性能。

在 server 端运行：iperf -s，在 client 运行：iperf -c <server ip> -t <runtime>。由 10Geb 路由器连接的节点之间的网络带宽为 9.41Gbits/sec，操作过程如图 5.2 所示，由 1Geb 路由器连接的节点之间的网络带宽为 939Mbits/sec，操作过程如图 5.3 所示。实验证明网络环境正常。

```
root@NEW-OSD0:~# iperf -c NEW-OSD1 -t 30
------------------------------------------------------------
Client connecting to compute1, TCP port 5001
TCP window size: 23.5 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.19 port 54202 connected with 10.0.0.20 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-30.0 sec  32.8 GBytes  9.41Gbits/sec
```

图 5.2 10Geb 网络测试过程

```
root@NEW-OSD0:~# iperf -c NEW-OSD1 -t 30
------------------------------------------------------------
Client connecting to compute1, TCP port 5001
TCP window size: 23.5 KByte (default)
------------------------------------------------------------
[  3] local 192.168.3.19 port 36032 connected with 192.168.3.20 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-30.0 sec  32.8 GBytes  939Mbits/sec
```

图 5.3 1Geb 网络测试过程.

2. 存储设备性能验证

该测试目的在于验证磁盘、SSD的吞吐量及IOPS，这里通过dd命令来完成。Linux操作系统中有一种page cache（页高速缓冲存储器）机制[30]，page cache的大小为一页，通常为4K。在linux读写文件时，它用于缓存文件的逻辑内容，然后系统通过调用fsync（2）将page cache的内容永久化到硬盘上，从而提高IO性能。但是我们为了测试磁盘性能，需要跳过Linux操作系统的这一缓存机制从而得到“真实”的磁盘性能情况。通过dd的oflag=direct选项可以达到这个效果。

分别向普通磁盘（7200RPM SATA HDD）和固态硬盘（SSD Intel SSDSA2CW300G3）写入文件，文件的块大小分别设置成4K和4M，测试两种存储介质的顺序写速度。普通磁盘的测试结果如表5.9所示，固态硬盘的测试结果表5.10所示。

表5.9 HDD性能测试结果

7200RPM SATA HDD	
块大小	顺序写速度
4M	199MB/sec
4K	28MB/s

表5.10 SSD性能测试结果

SSD（Intel SSDSA2CW300G3）	
块大小	顺序写速度
4M	265MB/sec
4K	65MB/s

5.3 集群性能测试

5.3.1 RADOS 对象存储接口测试

1. RADOS集群基准测试

1) 测试环境准备

RADOS 对象存储接口测试环境的网络结构拓扑如图 5.4 所示。测试环境包括一个 Monitor 节点与 6 个 OSD 服务器，每个 OSD 服务器上面有三个 OSD 节点，它们共同构成数据存储集群。两个客户端节点向整个集群产生压力，每个 OSD 服务器中有四块 SATA 硬盘，一块 SATA 硬盘作为系统盘，另外三块 SATA 搭载 OSD 节点，负责 Journal 和数据的存储。

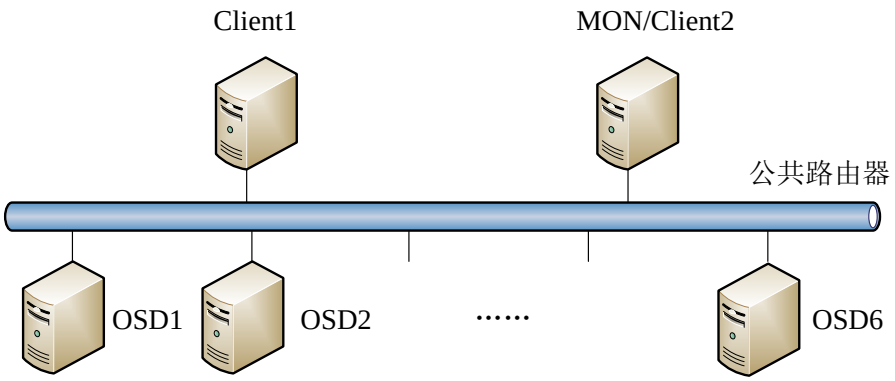

图5.4 RADOS集群基准测试环境网络结构拓扑图

2) 测试过程

该测试的目的在于模拟RADOS API读写对象的过程，本次测试采用Inktank开发的RADOS集群压力测试工具，测试RADOS集群读写4M，128K和1K大小对象时系统的吞吐量，分别模拟16个客户端和256个客户端完成读写过程，OSD节点底层文件系统分别采用Btrfs，XFS和Ext4，每次测试前文件系统必须重新格式化以确保之前测试产生的碎片不会对当前测试产出影响。RADOS集群对象副本数量设置为1。测试语句如表5.11所示。

表5.11 RADOS集群基准测试语句

```
rados bench -p 4M 1M 128K test 300 write --no-cleanup # test write
rados bench -p 4M 1M 128K test 300 seq # test read
```

3) 测试结果

在4K对象写入测试中，XFS和btrfs文件系统展现出了较好的性能，随着并发访问量的增加三种文件系统的性能都有一定程度的提升。在4K对象读取测试中，并发访问数量比较少时，三种文件系统的性能差别不大，而随着并发访问量的增加，ext4系统体现了较好的性能，甚至超过了btrfs系统的性能，这就使得ext4系统适合与专注小型文件读取的系统当中。测试结果如图5.5、5.6所示。

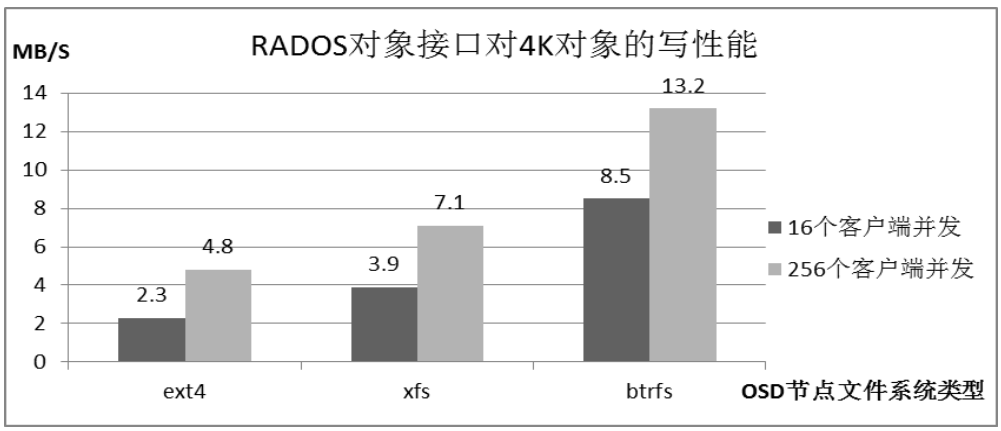

图 5.5 RADOS 集群写 4K 对象系统吞吐量测试结果

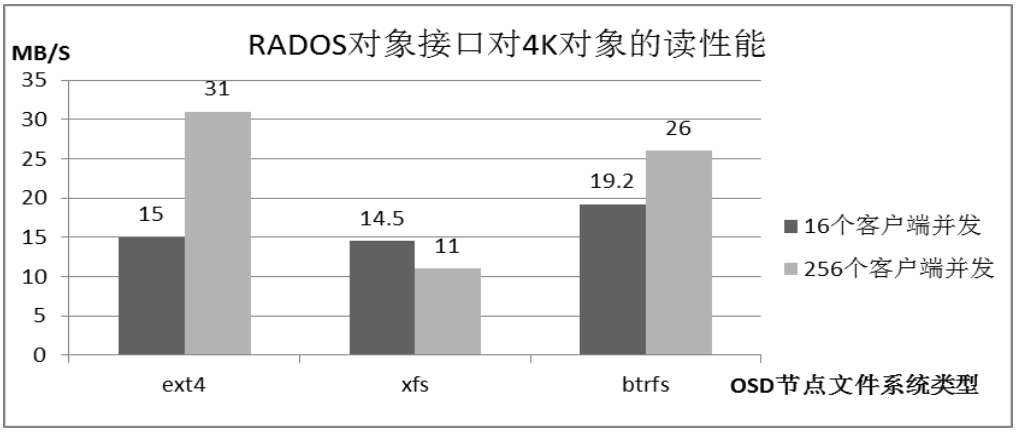

图 5.6　RADOS 集群读 4K 对象系统吞吐量测试结果

相对于 4K 对象的写入测试，ext4 系统对 128K 对象的写入测试的性能超过了 XFS 系统，但依旧远远低于 btrfs 系统，在对 128K 对象的读取测试中，随着并发访问量的增加，XFS 和 btrfs 系统的性能有所下降，而 ext4 系统略微增长。测试结果如图 5.7、5.8 所示。

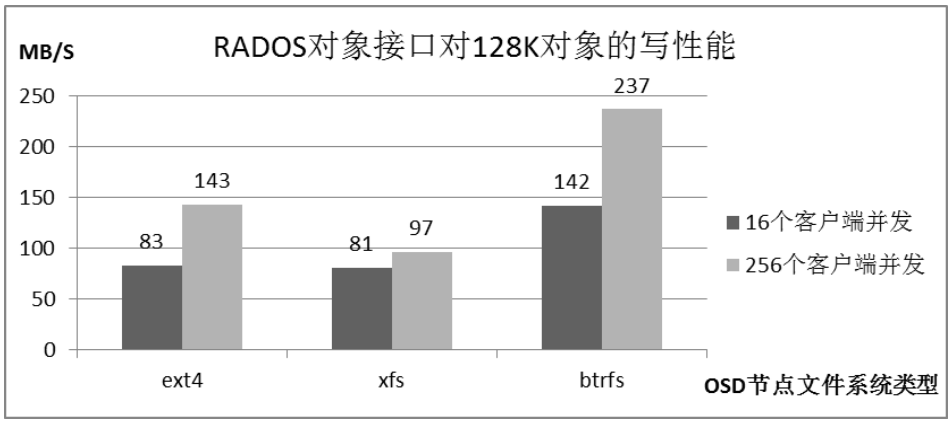

图 5.7　RADOS 集群写 128K 对象系统吞吐量测试结果

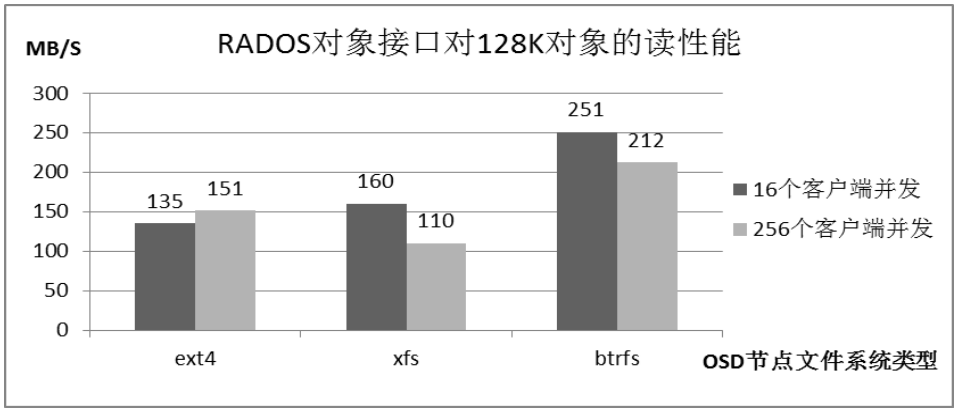

图 5.8　RADOS 集群读 128K 对象系统吞吐量测试结果

三种文件系统对于 4M 对象的读写性能相对稳定，并发访问的增长会使读写性能略微提升，ext4 系统和 XFS 系统对于 4M 对象读写体现出了相似的性能特征。测试结果如图 5.9、5.10 所示。

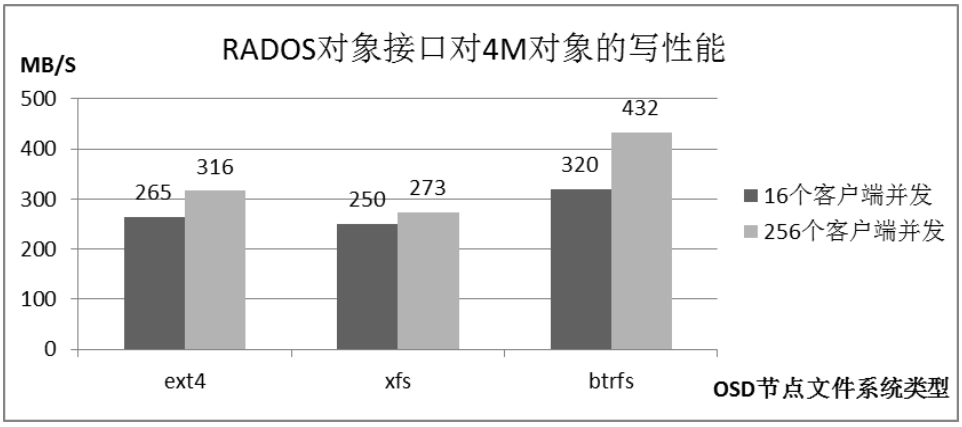

图 5.9 RADOS 集群读写 4M 对象系统吞吐量测试结果

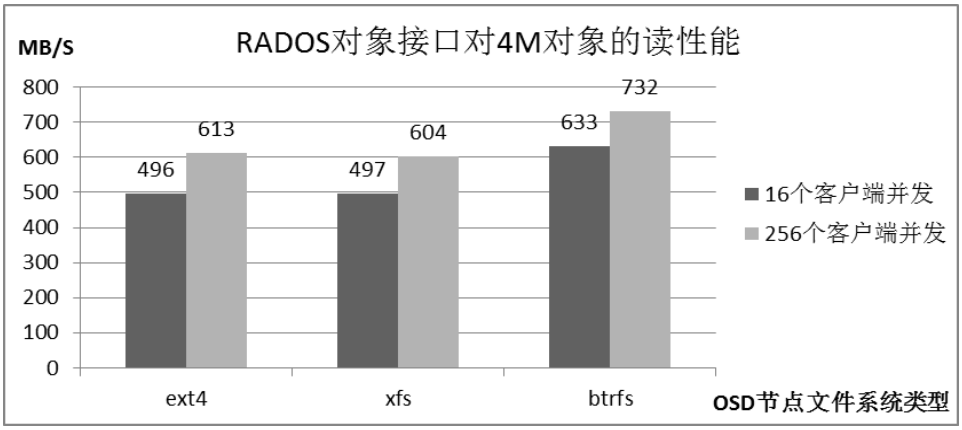

图 5.10 RADOS 集群读 4M 对象系统吞吐量测试结果

2. RADOS集群优化

1) 测试环境准备

根据对Ceph文件系统的分析，这里采用两种方式对RADOS集群的性能进行优化，首先为OSD集群构建专用的通信网络，负责承担副本分配产生的网络流量。而是采用SSD作为Journal文件的存储方案，代替原有的采用普通 SATA硬盘作为Journal文件的存储方案的方式。优化后的RADOS集群网络拓扑如图5.11所示。

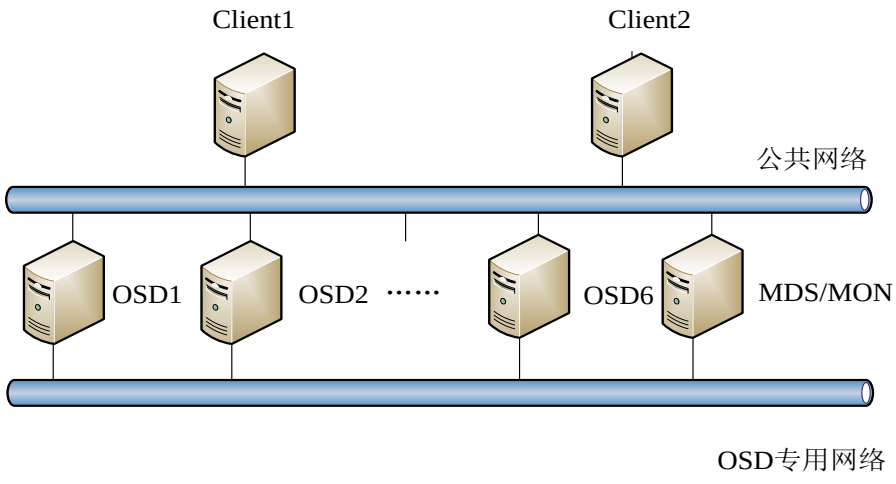

图 5.11 优化后的 RADOS 网络结构拓扑图

2) 测试过程

和基准测试一样使用Inktank开发的RADOS集群压力测试工具测试RADOS对4M对象进行读写时系统的吞吐量，模拟256个并发客户端访问集群，OSD底层文件系统采用XFS，本次测试将RADOS集群对象副本数量设置为一个和三个，分别进行测试。将测试结果与基准测试的结果比较，验证优化前后系统性能的差别。

3) 测试结果

首先用客户端向RADOS集群存储写入数据，并指定向RADIS集群存入1个对象副本，分别采用4种不同的网络配置和Journal配置方式，测试结果如图5.12所示。从测试结果可以看出，当将集群副本数量设置为1时，采用SSD作为Journal存储方案使系统的写性能有较大的提升，对系统读性能没有影响。而为OSD集群设置专有网络对系统的读写性能都没有太大的影响。

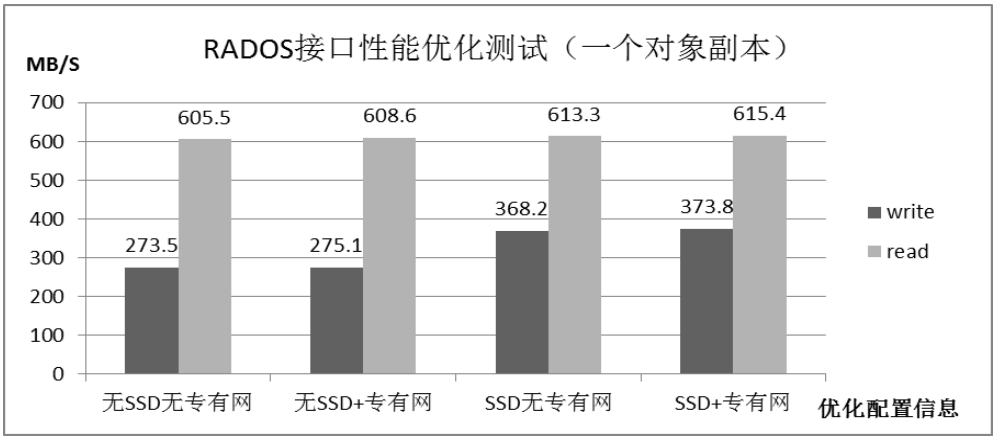

图 5.12 向 RADOS 集群写入一个对象副本时四种配置方式的系统读写性能

然后将存入向RADOS集群对象副本数量设置为三个，同样采用4种不同 的网络配置和Journal配置方式测试结果如图5.13所示。可以看出采用SSD作为Journal存储方案和为OSD集群配置专有的网络使系统的写性能均有较大的提升，但是对系统的读性能没有明显的影响。

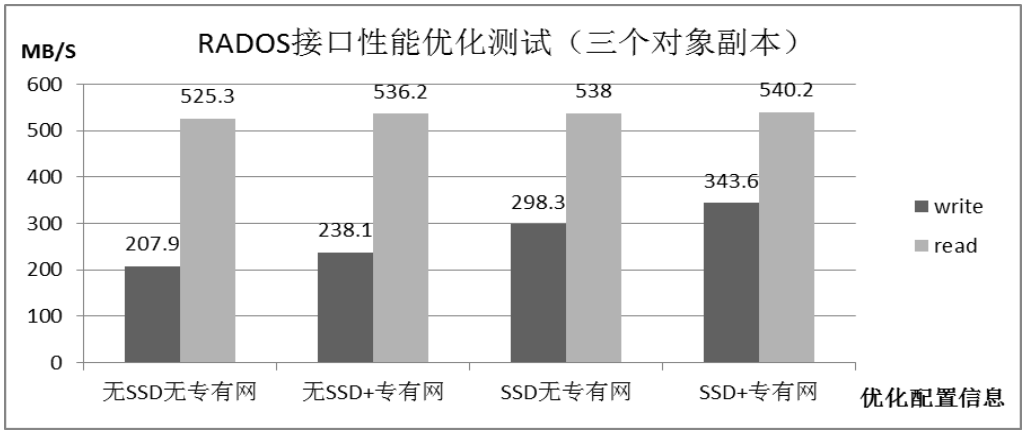

图 5.13 向 RADOS 集群写入三个对象副本时四种配置方式的系统读写性能

3. RADOS 集群扩展性测试

在经过优化的测试环境上，对RADOS对象存储接口进行扩展性测试，分别在集群中放置3个，6个，9个，12个，15个，18个OSD节点，并向OSD集群写入4M块大小的对象，观察集群随着OSD数量变化性能的改变。测试结果如图5.14所示。通过测试结果可以看出，随着OSD数量的递增RADOS的吞吐量体现出了良好的扩展性。

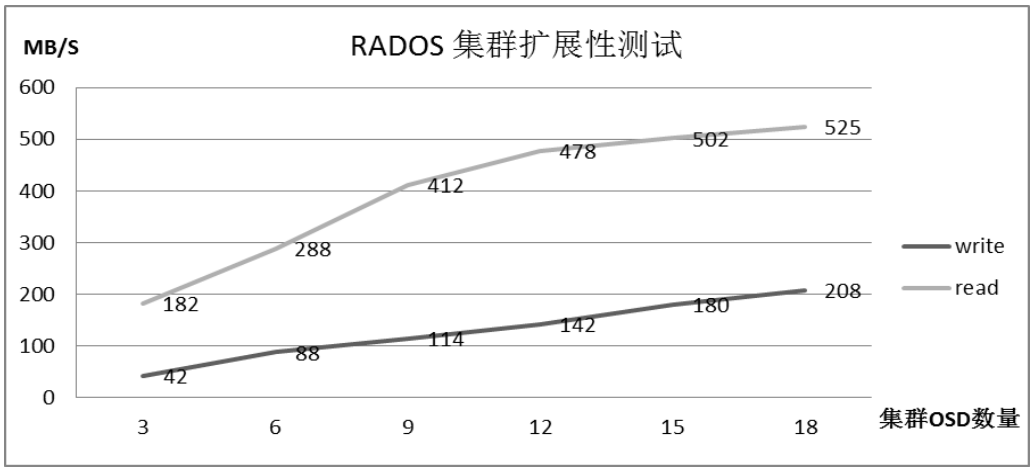

图 5.14 RADOS 集群扩展性测试结果

5.3.2 Ceph RBD 存储接口测试

1. RBD 接口基准测试

1) 测试环境

Ceph RBD 接口与 CephFS 文件系统一样具有 Ceph 对象存储系统提供的 librados 接口交互，它将块设备文件当做对象来存储。因为 RBD 构建在 librados 之上，RBD 继承了 librados 的特性，当一个应用将使用块设备将数据写入 Ceph，Ceph 会自动的将数据条块化并将数据复制到集群中去。通过将块设备数据条块化到集群，Ceph 提高了对块设备数据的读写性能。RBD 同样可以 KVMs（kernel virtual machines）整合在一起。因为 Ceph RBD 接口是构建在 RADOS 集群之上的，所以对 RBD 接口的基准测试就构建在优化过的 RADOS 集群之上，在 Ceph 客户端创建 RBD 设备，完成对 Ceph RBD 接口的测试。

2) 测试过程

首先搭建 RADOS 存储集群，然后在 Ceph 客户端创建 RBD 块设备。分 为两种情况，第一种情况是在客户端创建裸 RBD 块设备，创建过程如下：1）在 Ceph 客户端建立 RBD 块设备。2）将块设备映射与客户端建立映射关系。3）挂载。操作过程如如 5.15 所示。第二种情况是将新建立的 RBD 块设备格 式化为文件系统，这里分别格式化为 EXT4 和 XFS 文件系统。创建过程如下：1）在

Ceph 客户端建立 RBD 块设备。2）将块设备映射与客户端建立映射关系。3）创建文件系统挂载。操作过程如图 5.16 所示。为了使用 Ceph 块设备，必须访问一个正在运行中的 Ceph 集群。

```
rbd create rbd-image0 – size 32768
rbd list
rbd map rbd/rbd-image0
rbd showmapped
mount /dev/rbd0 /mnt/rbd0
```

图 5.15 RBD 裸设备创建过程

```
rbd create rbd-image0 – size 32768
rbd map rbd/rbd-image0
rbd showmapped
mkfs.ext4/dev/rbd0
mount /dev/rbd0 /mnt/rbd0
rbd create rbd-image0 – size 32768
rbd map rbd/rbd-image0
rbd showmapped
mkfs.XFS/dev/rbd0
mount /dev/rbd0 /mnt/rbd0
```

图 5.16 带文件系统的 RBD 块设备的创建过程

本次对 RBD 接口的基准测试所使用的测试工具是 fio，这个工具最大的特点是使用简单，支持的文件操作非常多，主要用来测试文件系统的 IOPS。使用 fio 测试 RBD 设备的随机读，随机写以及随机读写时系统的吞吐量，测试语句如表 5.12 所示。filename=/mnt/rbd0/tt，测试文件的名称以及目录；direct=1，该参数会使测试过程绕过机器自带的 buffer，使测试结果更加真实；rw=randrw，测试写和读的 I/O；bs=4k，单次 io 的大小为 4k；size=30G，测 试文件大小为 30G，以每次 4k 的 io 进行测试。Numjobs=64，测试的线程为 64；

表 5.12 RBD 接口基准测试语句

```
fio -filename=/mnt/rbd0/tt -direct=1 -rw=randrw -bs=4k -size=30G
-numjobs=64 -runtime=120 -group_reporting --name=test
```

3) 测试结果

裸 RBD 设备和基于 EXT4 和 XFS 文件系统的 RBD 的随机读性能基本一致，如图 5.17 所示。基于 XFS 文件系统的 RBD 的随机写性能和随机读写性能与裸 RBD 的性能基本一致，但是基于 EXT4 的 RBD 的随机写与随机读写性能非常差，如图 5.18 和图 5.19 所示。

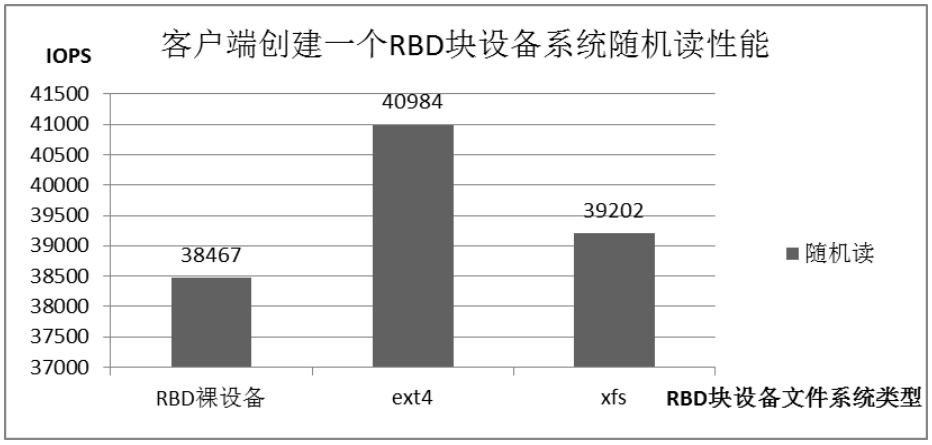

图 5.17 三种类型 RBD 设备的随机读性能

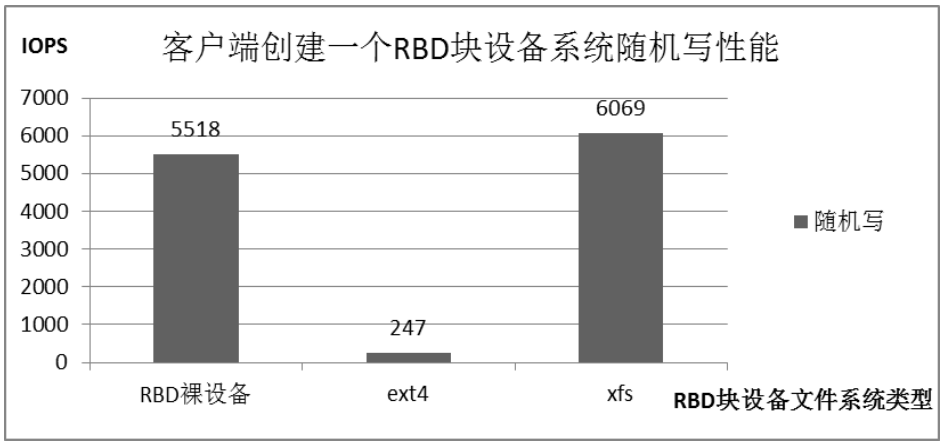

图 5.18 三种类型 RBD 设备的随机写性能

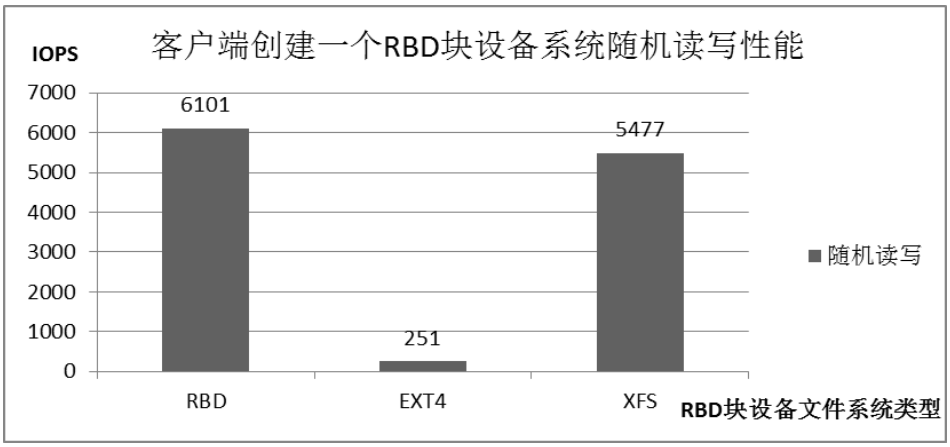

图 5.19 三种类型 RBD 设备的随机读写性能

2. RBD 接口性能调优测试

在 RBD 接口基准测试的基础上，分别在一个客户端创建一个，两个，三个，四个 RBD 设备，相对个 RBD 设备同时完成数据的读写。测试 RBD 数量对系统读写性能的影响。当在一个客户端构建 2 个 RBD 设备时，随机读性能达到最佳。基于 EXT4 文件系统单位 RBD 设备当设备数量达到 3 个，4 个时，随机读性能下降明显。如图 5.20 所示。

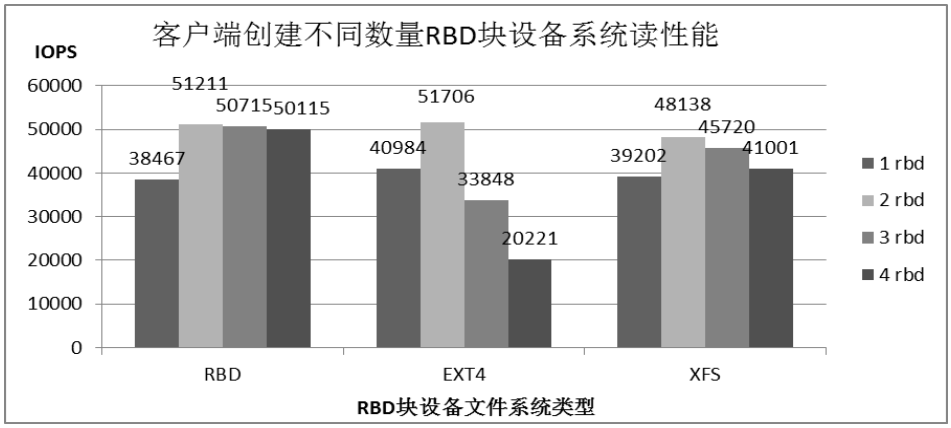

图 5.20 不同 RBD 设备数量情况下接口的随机读性能

当单客户端构建 3 个 RBD 设备时，系统的随机写性能达到最大，基于 EXT4 的 RBD 设备随着 RBD 设备量的增加性能仍有提升的空间，测试结果如图 5.21 所示。

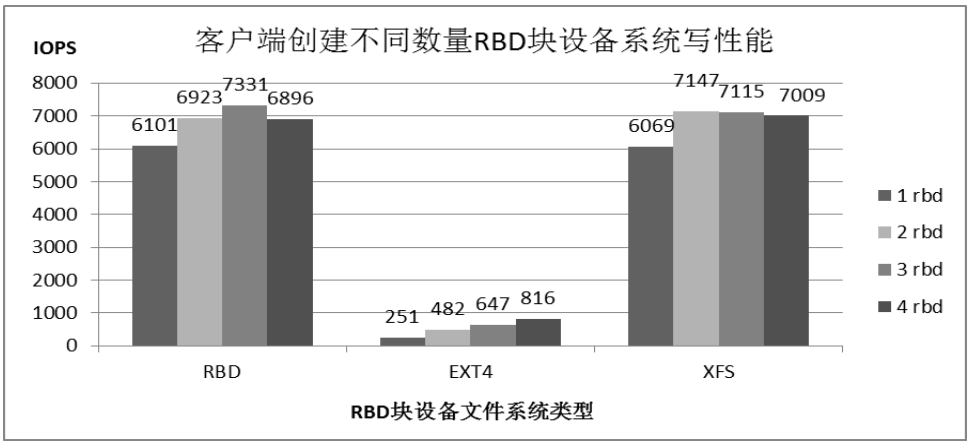

图 5.21 不同 RBD 设备数量情况下接口的随机写性能

当单客户端构建 3 个 RBD 设备时，裸 RBD 设备和基于 XFS 的 RBD 设备的随机读写性能达到最大，基于 EXT4 的 RBD 设备随着 RBD 设备量的增加性能仍有提升的空间，测试结果如图 5.22 所示。

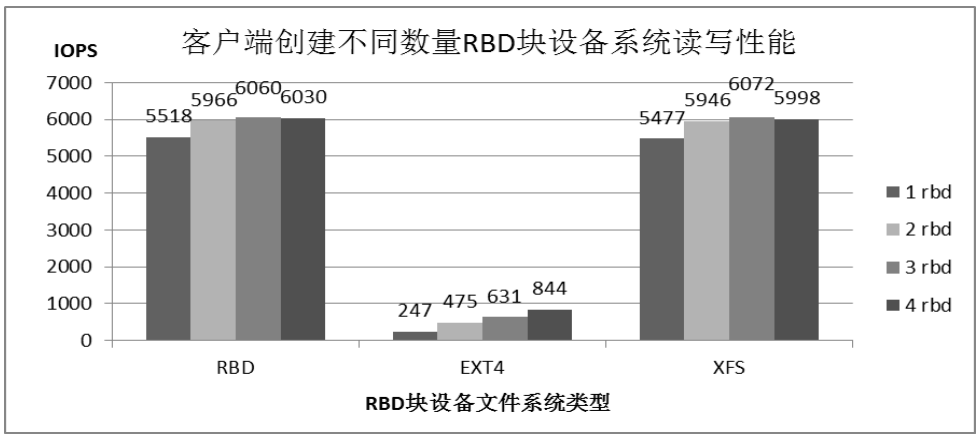

图 5.22 不同 RBD 设备数量情况下接口的随机读写性能

3. Ceph RBD 接口扩展性测试

在经过优化的测试环境上，对 Ceph RBD 存储接口进行扩展性测试，分别在集群中放置 3 个，6 个，9 个，12 个，15 个，18 个 OSD 节点，观察客户端随着 OSD 数量变化读写性能的改变。测试结果如图 5.23、5.24 所示。可以看出随着 OSD 数量的增长 RBD 接口体现出了良好的扩展性。

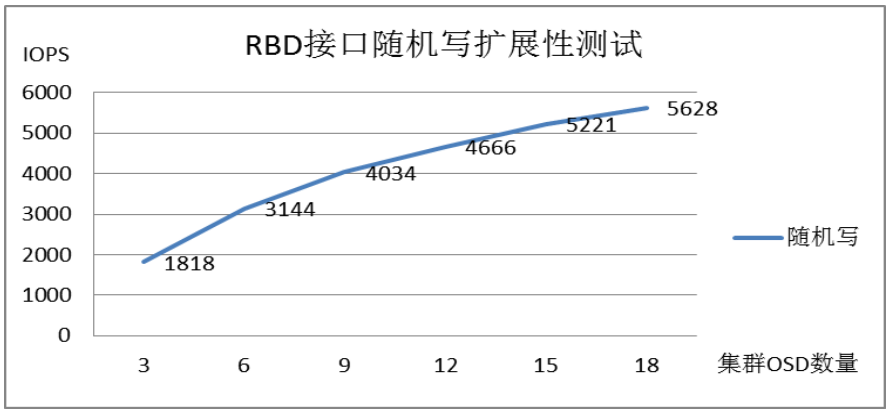

图 5.23 RBD 接口随机写扩展性测试结果

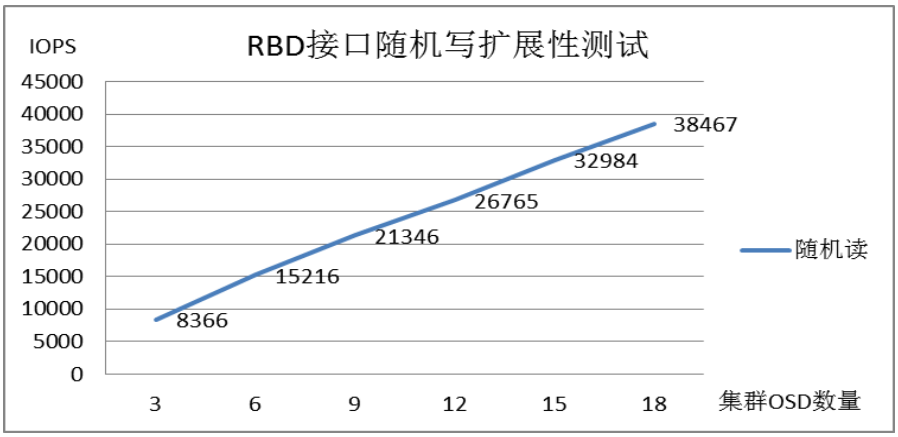

图 5.24 RBD 接口随机读扩展性测试结果

5.4 测试结果分析

根据对测试结果，我们可以得出以下结论：

第一，采用 SSD 作为 Journal 的存储方案显著的提高了系统的性能。但是由于 SSD 的价格较普通磁盘来讲要高的多，在实际当中如果部署大型的集群，可以考虑在一个主机部署一块 SSD 和 N（N 通常是 3 到 6 块）块普通 HDD，然后将 SSD 分为 N 个区，每个分区与一块普通 HDD 共同构成一个 OSD 节点，共同负责 OSD 的 Journal 和数据对象的存储。

第二，副本数量为 1 时系统的写性能明显好于副本数量为 3 时的写性能，而读性能没有太大的变化。原因是该测试集群采用的是 primary-copy 副本分配方式，如图 5.25 所示。

当对象写入集群式，首先被写到 primary OSD 节点中，当副本个数大于 1 个，假设，副本数目为两个，Primary OSD 会并行的将副本写入 Secondary OSD 和 Tertiary OSD 中，当写入完成，Secondary OSD 和 Tertiary OSD 会向 Primary OSD 做出响应，Primary OSD 收到响应后会向客户端做出响应，标志着写过程的完成。当客户端向集群读取对象则直接从 Primary OSD 读取。所以副本数量不会影响集群的读性能。

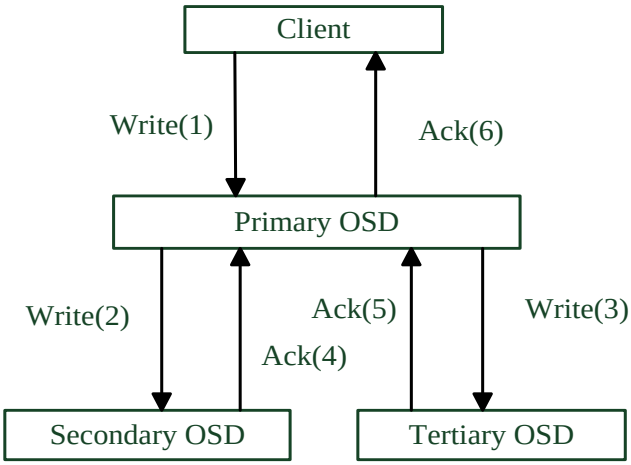

图 5.25 primary-copy 副本分配方案写过程

第三，当向集群写入单个对象副本时 ，在 OSD 集群内部使用专用网络明显提高了系统的性能，因为当 Primary OSD 向 Secondary OSD 和 Tertiary OSD 中复制副本时，利用到了 OSD 集群专用网络，这种配置方式分担了客户端与 OSD 节点之间的网络流量，提高了系统性能。在构建自己的集群时，强烈建议为 OSD 集群设置自己的专有网络，当集群 OSD 扩展时，OSD 专有网络同样会分担由数据迁移和重组造成的网络流量。

5.5 本章小结

本章首先对搭建测试集群的分析过程做了简要介绍，接下来进行了集群软硬件信息的介绍，完成了对集群网络及硬件设施的基准测试，并根据第四章提出的测试用例对系统完成了性能的测试，并对测试数据做出了分析，提出了对集群性能优化的方案。

第六章 结束语

6.1 论文工作总结

现阶段，互联网行业快速发展，由此产生了海量的网络数据，从而对存储系统的容量、可扩展性、吞吐量以及数据可用性等方面提出了越来越高的要求。而目前存储技术相对于数据处理和传输技术来讲，发展相对滞后，成为了阻碍存储系统性能的提高的主要瓶颈。分布式系统能够提供 PB 级别的海量数据存储，具有高可扩展性，高并发性，高可靠性，易管理性等使用特点，得到了各界的广泛关注。因此，分布式文件系统性能的研究是分布式文件系统研究的重点和难点。然而，分布式系统在性能因素分析及提取、性能研究、性能预测、性能优化设计等方面存在着很多问题。

针对以上情况，本文从分布式文件系统的系统入手对分布式文件系统的若干关键问题进行了研究。本文所做的工作如下：

（1）介绍分布式系统相关的基础概念。包括数据分布、副本控制、集群扩展等关键技术。介绍了衡量分布式文件系统的性能指标。

（2）重点围绕着分布式文件系统 Ceph 进行分析和研究。重点研究了 Ceph 分布式文件系统的两个关键技术：1）通过 CRUSH 算法解耦数据和元数据的存储，客户端利用该算法可以计算对象在集群中的位置。CRUSH 算法强制将对象副本分配到集群的故障域中，在提高数据安全性的同时，很好的适应了大型存储系统动态的结构特征，即设备故障、集群扩展和重组经常发生的现象。2）智能的分布式对象存储集群服务（RADOS），RADOS 利用 OSD 的智能性，管理数据复制、故障检测与恢复、低级别数据块管理、数据迁移，不涉及任何中心服务器，最大程度的实现系统的扩展性。

（3）对 Ceph 文件系统可用性、扩展性以及集群部署方案等方面进行了测试需求的分析，进行了测试用例设计方法的分析并设计了详细的测试用例，并根据对 Ceph 文件系统架构性能的分析，提出了集群性能优化的测试用例方案。根据对 Ceph 文件系统技术特点的研究搭建测试系统，对 Ceph 文件系统的两个存储接口即 RADOS 对象存储接口和 Ceph RBD 块存储接口进行了性能测试、压力测试。经过对实验结果的评估和分析，得出了 Ceph 分布式文件系统具有良好的可用性、可扩展性和高性能，同时，文中提出的集群系能优化方案确实提高了系统的存储性能。

6.2 后续工作展望

Ceph 文件系统作为一个开源系统，目前还在不断更新完善中，每一个版本都会有相应的性能优化。今后的研究工作将在以下几方面进行：

1. 在之前学习研究的基础上进一步学习 Ceph 文件系统的新特性。

2. 继续优化 RADOS 集群的配置，采用磁盘阵列的方式构建 OSD 节点（例如，SSD 用于 Journal 存储，用三个 HDD 采用 RAID0 的方式构建磁盘阵列，用于数据存储）。

3. 根据 Ceph 可无限扩展的特性，研究如何将 Ceph 集成到例如 OpenStack 等云平台中，做为云平台的后端存储解决方案。

致　谢

三年的硕士研究生学习生活就要结束了，本文将为我的学生生涯画上一个句号，三年中，很多人给予了我无私的帮助和指导，正因此，我才能顺利地完成我的学业。

在研究生期间，我有幸得到了李青山教授的指导，衷心感谢李老师三年来的教导和培育。李老师有渊博的计算机学术知识，严禁地治学态度。这令我受益匪浅，对我今后的工作有很大的帮助。另外，在毕业论文的写作过程中，从最初选题，到关键点研究，再到最后的写作，李老师给了我很多指导，提出很多有益的建议。

在研二时期，我有幸到英特尔亚太研发中心的软件服务部门实习，并有幸认识了魏彬，黄析伟，何普江，陈津等几位工程师，衷心的感谢他们在我实习期间对我无私的指导，让我在学生生涯即可体验到企业文化，并在项目中磨练自己。

我还要感谢我的舍友张红兵，刘帅，张弘驰同学，与他们的朝夕相处让我感受到了家庭的温暖与团结的力量。

最后，谨以此文献给我的父亲和母亲。我相信我的父母是天下间最伟大的父母，他们这辈子大部分的辛劳都是为了我可以过上更好的生活。他们的谆谆教导都让我坚定前进的信念，脚踏实地的做人来回报那些支持过自己的人们。

参考文献

[1] Gray J. What next? A few remaining problems in Information Technology[C] //ACM Federated Research Computer Conference. 1999.

[2] 赵铁柱. 分布式文件系统性能建模及应用研究[D]. 广州: 华南理工大学, 2011.

[3] Pawlowski B, Juszczak C, Staubach P, et al. NFS Version 3: Design and Implementation[C]//USENIX Summer. 1994: 137-152.

[4] Howard J H. An overview of the andrew file system[M]. Carnegie Mellon University, Information Technology Center, 1988.

[5] O'connor M A. Method of enabling heterogeneous platforms to utilize a universal file system in a storage area network: U.S. Patent 6,564,228[P]. 2003-5-13.

[6] Wu C T, Chao Y L. Network access server: U.S. Patent D370,470[P]. 1996-6-4.

[7] Soltis S R, Ruwart T M, O'keefe M T. The global file system[C]//NASA CONFERENCE PUBLICATION. 1996: 319-342.

[8] Schmuck F B, Haskin R L. GPFS: A Shared-Disk File System for Large Clusters[C]//FAST. 2002, 2: 19.

[9] Menon J, Pease D A, Rees R, et al. IBM Storage Tank—a heterogeneous scalable SAN file system[J]. IBM Systems Journal, 2003, 42(2): 250-267.

[10] Pilaud D, Halbwachs N, Plaice J A. LUSTRE: A declarative language for programming synchronous systems[C]//Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages (14th POPL 1987). ACM, New York, NY. 1987, 178: 188.

[11] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.

[12] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[13] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.

[14] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally-distributed database[C]//Proceedings of OSDI. 2012, 1.

[15] Weil S A. Ceph: Reliable, scalable, and high-performance distributed storage[D]. UNIVERSITY OF CALIFORNIA, 2007.

[16] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance

distributed file system[C]//Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006: 307-320.
[17] 杨传辉，分布式文件系统工程实践，2010.
[18] 刘杰，分布式系统原理
[19] 陈涛, 肖侬, 刘芳, 等. 基于聚类和一致 Hash 的数据布局算法[J].
[20] Josephson W K, Bongo L A, Li K, et al. Dfs: A file system for virtualized flash storage[J]. ACM Transactions on Storage (TOS), 2010, 6(3): 14.
[21] Azagury A, Dreizin V, Factor M, et al. Towards an object store[C]//Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on. IEEE, 2003: 165-176.
[22] McGregor A J. Block-Based Distributed File Systems[D]. The University of Waikato, 1997.
[23] Patterson D A, Gibson G, Katz R H. A case for redundant arrays of inexpensive disks (RAID)[M]. ACM, 1988.
[24] Wang R Y, Anderson T E. xFS: A wide area mass storage file system[C]// Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on. IEEE, 1993: 71-78.
[25] Mason C. The btrfs filesystem[J]. The Orcale cooperation, 2007.
[26] Cao M, Bhattacharya S, Tso T. Ext4: The Next Generation of Ext2/3 Filesystem[C]//2007 Linux Storage & Filesystem Workshop. 2007.
[27] Xin Q, Miller E L, Schwarz S, et al. Impact of failure on interconnection networks for large storage systems[C]//Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on. IEEE, 2005: 189-196.
[28] Needham R M. Denial of service[C]//Proceedings of the 1st ACM Conference on Computer and Communications Security. ACM, 1993: 151-153.
[29] Tirumala A, Qin F, Dugan J, et al. Iperf: The TCP/UDP bandwidth measurement tool[J]. htt p://dast. nlanr. net/Projects, 2005.
[30] Harty K, Cheriton D R. Application-controlled physical memory using external page-cache management[M]. ACM, 1992.