学校代码：10286
分类号：TP311
密　级：公开
U D C：004.9
学　号：153569

东南大学
SOUTHEAST UNIVERSITY

工程硕士学位论文

基于 Ceph 分布式系统的存储虚拟化技术的研究与实现

（学位论文形式：基础研究）

研究生姓名：张 晓 宇
导 师 姓 名：戚晓芳 副教授
吴香伟 高工

申请学位类别 工 程 硕 士　学位授予单位 东 南 大 学
工程领域名称 软件工程　论文答辩日期 2018 年 6 月 22 日
研 究 方 向 软件工程　学位授予日期 2018 年 月 日
答辩委员会主席 徐 立 臻　评 阅 人 何 洁 月
张 振 华

2018 年 月 日

东南大学

硕士学位论文

基于 Ceph 分布式系统的存储虚拟化技术的研究与实现

专 业 名 称：软件工程

研究生姓名：张晓宇

导 师 姓 名：戚晓芳

吴香伟

RESEARCH AND IMPLEMENTATION OF STORAGE VIRTUALIZATION BASED ON CEPH DISTRIBUTED SYSTEMS

A Dissertation Submitted to

Southeast University

For the Academic Degree of Master of Engineering

BY

Zhang Xiao-Yu

Supervised by

Associate Prof. Qi Xiao-Fang

Software Engineering

Southeast University

June 2018

东南大学学位论文独创性声明

本人声明所呈交的学位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的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____日期：_____

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆、《中国学术期刊（光盘版）》电子杂志社有限公司、万方数据电子出版社、北京万方数据股份有限公司有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名：_____导师签名：_____日期：_____

摘要

现代社会对大规模数据的存储需求日益增加，而传统的存储体系结构在容灾容错、扩展性和维护管理等方面存在局限，致使企业需要付出额外成本。针对该问题，本文将分布式存储系统和存储虚拟化技术充分结合起来，设计和实现了一个基于分布式系统的虚拟化存储系统，主要功能包括存储资源管理、登录安全认证和数据交互等。论文的主要工作包括以下几个方面：

（1）设计和实现了存储资源管理模块。该模块整合异构存储设备，对物理存储资源进行统一管理，为用户提供一个逻辑存储资源池，在资源池内根据用户的应用需求分配逻辑存储卷空间，通过一致性哈希算法将逻辑卷上的数据空间映射到底层对象集合。同时，系统可扩展到成百上千甚至更多的节点。当物理资源出现故障的时候，系统可迅速恢复失效数据，实现高可用性。

（2）设计和实现了登录认证模块。该模块实现了存储端对用户的身份认证，当客户端与存储端进行连接时，进行三次握手周期性的身份校验，使用 MD5 加密算法避免校验过程中传送用户的真实密码，增加系统的安全性。

（3）设计和实现了数据交互模块。该模块负责在客户端和存储端之间建立 iSCSI 会话，解析从 iSCSI 会话中传输过来的数据报文，并将解析出来的命令和数据下移到底层分布式存储系统，进行 I/O 操作。

最后，系统测试结果表明，在影响存储性能的几个因素中，文件访问模式对该存储系统吞吐量的影响最大。同时，从测试数据中可得到该存储系统吞吐量的峰值是 6,241,661KB/s。该存储虚拟化系统已在杭州华三通信技术有限公司发布，是华三面向企业所提供私有云服务的重要组成部分，帮助企业从传统存储环境向多节点的虚拟化存储体系转型，充分利用现有存储资源。

关键词：分布式；虚拟化；Ceph；iSCSI；吞吐量

Abstract

With the increasing demand for large-scale data storage in the modern society, the traditional storage architecture has limitations in terms of disaster tolerance, scalability, and maintenance management, which results in additional costs for enterprises. To solve these problems, this thesis fully combines distributed storage systems and storage virtualization technology, and designs and implements a virtualized storage system based on distributed systems. The main functions include storage resource management, logining security authentication, and data interaction. The main work of the thesis includes the following aspects:

(1) Design and implement a storage resource management module. This module integrates heterogeneous storage devices, manages physical storage resources in a unified manner, provides users with a logical storage resource pool, allocates logical storage volume space according to user application requirements in the resource pool, and map the data space on the logical volume to the underlying object collection with the consistent hashing algorithm. At the same time, the system can be expanded to hundreds or even thousands of nodes. When the physical resources fail, the system can quickly recover the failure data and achieve high availability.

(2) Design and implement a module for logining authentication. This module implements the storage side's identity authentication to the user. When the client connects to the storage side, three-way handshaking periodical identity verification is performed, and the MD5 encryption algorithm is used to avoid transmitting the user's real password during the verification process, thereby increasing the security of the system.

(3) Design and implement a module for data interaction. The module is responsible for establishing an iSCSI session between the client and the storage, parsing the data packets transmitted from the iSCSI session, and moving the parsed commands and data to the underlying distributed storage systems for I/O.

Finally, the system test results show that among several factors affecting storage performance, the file access mode has the greatest impact on the storage system throughput. At the same time, the peak value of the storage system throughput obtained from the test data is 62,461,161 KB/s. The storage virtualization system has been released by H3C Technologies Co., Limited. It is an important component of the private cloud services provided by H3C for the enterprise, which helps enterprises transform from traditional storage environments to multi-node virtual storage systems and take full advantage of existing storage resources.

Keywords: distributed systems; virtualization; Ceph; iSCSI; throughput

目录

本论文专用术语注释表

英文缩写	英文全称	中文释义
NAS	Network Attached Storage	网络附加存储
SAN	Storage Area Network	存储区域网络
NFS	Network File System	网络文件系统协议
SMB	Server Message Bloc	服务消息块协议
CIFS	Common Internet File System	通用网络文件系统协议
SCSI	Small Computer System Interface	小型计算机系统接口
FCIP	Fibre ChanneI over IP	基于 IP 的光纤通道
iFCP	Internet Fibre ChanneI ProtocoI	网络光纤通道协议
iSCSI	Internet Small Computer System Interface	互联网小型计算机系统接口

第一章 绪论

1.1 选题依据

随着互联网技术、网络多媒体和电子商务的迅速发展，存储系统作为知识和信息载体的地位变得越来越重要。据全球互联网数据中心统计，企业的数据量每隔半年就增长一倍。在传统的存储环境下，存储容量利用率仅为百分之四十左右[1]，这种低利用率导致企业在扩展存储容量的时候，需要产生大量的额外成本。同时，传统存储结构在主机内存容量和文件管理系统开销的限制下，不能满足高可扩展性和高持续性带宽的传输要求。

现代存储应用对存储设备提出了大容量、高可用性、可维护性、开放性和动态可扩展性的需求，基于主机与存储系统直接相连的存储体系结构越来越不适应目前的发展趋势。在这种情况下，存储技术的发展方向开始走向了存储虚拟化[2]。在虚拟化环境中，存储对用户是透明的，存储系统的内部功能被抽象、隐藏或隔离，进而实现应用、网络和存储设备的独立管理。存储虚拟化可简化复杂的底层存储基础，其技术思想是将资源的逻辑空间与底层的物理存储分离，将异构的物理存储设备逻辑化为具有统一接口的网络概念上的实体，从而为系统和管理员提供一个简化的虚拟资源管理环境，所有资源在网络上被统一管理，实现远距离的集中访问，满足资源共享的需求。存储虚拟化在用户和存储资源之间架设一个“桥梁”，实质上解决的是用户对存储资源的远端访问限制，并未真正摆脱对实际物理资源的依赖，它需要专业的元数据管理和资源管理设备，所以在扩展性上仍存在瓶颈。不同性能和结构的存储设备，不能充分发挥各自的优势，性能较差的部件反而会制约整个存储系统的性能，同时，用户数据在传输和存储过程中存在安全风险，在异构虚拟存储环境下进行统一的数据保护仍是一个挑战。

在存储虚拟化技术迅速发展的同时，分布式存储技术也在逐渐普及中[3]。磁盘、磁带和磁带机是标准的传统集中式数据存储设备，与这些传统的存储设备相比，分布式存储系统把数据分散在众多独立的设备上，通过可扩展的网络技术来互相连接单独存在的计算和存储单元，构建存储集群。分布式存储系统提供非结构化数据的持久性存储，这些数据被组织在由系统集群共享的分层名称空间中。分布式存储系统可以同时在相同或者不同的物理服务器节点上进行工作负载，因而可以在许多节点中构建一个共享存储系统，建立高效的超融合存储基础设施。作为 OpenStack 云计算管理平台中呼声最高的分布式存储系统，Ceph 已经实现了块存储、文件存储和对象存储这三种目前最流行的存储方式。Ceph 解除了对文件分配表的依赖，最大化分离数据和元数据的操作。同时，Ceph 动态的分布式元数据集群可以解决热点数据访问和负载分布不均的问题，提高数据读写性能和减轻机器的负载。但是，与其他分布式存储系统一样，Ceph 存储集群总是位于一个数据中心，节点之间无法跨越长距离，导致远端用户对于存储集群的访问总是受困于地域限制。

存储虚拟化技术的目标是打破距离限制，将物理资源逻辑化为虚拟设备，合理调配存储资源，但是未能解决底层物理资源的扩展问题。而分布式系统是把数据分散在多台计算机设备上，共同承担工作负载，缺点是远端用户无法访问这些分布式存储资源。针对传统存储体系的缺陷，为了充分发挥虚拟化与分布式技术各自的优势，本论文的目标是将分布式存储系统和存储虚拟化技术结合起来，实现一个基于 Ceph 分布式系统的虚拟化存储架构，统一管理和分配物理存储资源，实现存储资源的动态加入，为用户提供一个逻辑上的可扩展地址空间，克服传统存储方式的低容量、不易扩展性和受地域限制等缺点。

1.2 国内外研究现状

1.2.1 分布式存储系统的研究现状

面对不断增长的存储需求以及在系统速度、安全性和管理方面面临的诸多挑战，研究人员在研究过程中不断探索提高存储性能的方法，最后发现改变和突破传统存储体系结构是最有效的途径。最近几年以来，由于存储系统和网络技术的不断发展和相互融合，一些不同结构的分布式存储系统先后产生，在一定程度上缓解了许多应用系统对于存储容量和性能的迫切需求。

上个世纪七十年代的主机和终端模式是数据存储技术的起源，主机系统负责数据存储，它的文件系统是所有数据存储的中心[4]。自八十年代以来，客户机和服务器的模式开始成为计算机环境的主流，数据存储的聚合中心变成了文件服务器和数据库服务器。同时，客户端也负责一些零碎数据的存储。这种存储数据分散的模式开始出现在存储领域，并进一步促进了数据存储技术由传统集中式存储向分布式存储模式的转变。

分布式存储系统为共享文件提供永久存储，并通过联合分散的存储资源来构建这些共享文件的分层统一管理体系。各种各样的应用，例如概率分析、天气预报和空气动力学研究，都依赖于分布式环境来处理和分析大量数据。现在，超级计算机、集群和数据中心使用的主要存储解决方案都是分布式存储系统。目前最流行的分布式存储系统是 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS），Gluster 文件系统（Gluster File System，GlusterFS），Lustre 和 Ceph。

HDFS 是一个集中式的分布式存储系统[5]，元数据由单一的名称节点服务器进行管理，数据以块的形式被分割在几个数据节点。HDFS 使用包含元数据（例如权限、空间磁盘配额、访问时间）的索引节点在文件和目录层次结构中管理名称空间。由名称节点管理服务器管理的名称空间和元数据执行存储在各节点上的文件名和文件块之间的映射。HDFS 的单点故障和性能瓶颈源于它单一的名称节点服务器，同时它不支持数据修改，这些因素导致它并不是可靠的存储系统。

GlusterFS 没有元数据服务器，它将数据和元数据存储在连接到不同服务器的多个设备上，这组存储设备称为卷，数据块被分布和复制到卷内的多个设备上[6]。GlusterFS 使用弹性哈希算法来定位文件，以提供全局名称空间。这种算法将文件路径名转换为固定长度、统一和唯一的值，一个存储设备可以分配一定范围的值，系统基于这个值存储文件。由于缺乏元数据服务器的支持，在遍历文件目录的时候需要搜索所有的存储节点，在客户端的过度负载导致了 GlusterFS 存储的操作复杂和效率低下。

Lustre 与当前其他分布式存储系统的不同之处在于，它不提供任何数据和元数据副本[7]。相反，Lustre 选择将元数据存储在连接两个元数据服务器的共享存储设备上，支持主动和被动的故障切换。数据被拆分为多个对象，并分布在多个共享对象存储设备上，这些共享对象存储设备可以连接到多个处理 I/O 请求的对象存储服务器，并支持类似元数据管理的故障切换。Lustre 的全局名称空间由元数据服务器提供给用户，在使用元数据中索引节点和扩展属性将文件对象名称映射到其相应的对象存储设备上。

Ceph 是一个完全的分布式存储系统。与 HDFS 不同，为了确保可伸缩性，Ceph 使用元数据服务器集群提供动态分布式元数据管理，并将数据和元数据存储在对象存储设备中。元数据服务器集群管理系统的命名空间，并保证安全性和一致性，在对象存储设备执行 I/O 操作时进行元数据查询。与通过元数据服务器定位每个数据块的分布式存储系统相反，Ceph 允许客户端计算哪些存储设备应当写入所请求的数据，这个计算过程称之为 CRUSH 伪随机分布算法[8]。由于 Ceph 的数据定位是由客户端使用自己的资源来完成的，从而去除了中心查找带来的性能以及单点故障问题。除了采用 CRUSH 算法克服中央元数据服务器的瓶颈以外，Ceph 还使用了基于动态子树来分割元数据的集群架构，把对文件系统层次目录结构的管理职责智能地分配到数十个甚至数百个元数据服务器中，根据访问热点来调整各个节点上的工作负载。

Ceph 具有良好的可扩展性，而且每一个组件都是可靠和支持高可用的，可智能调整节点负载，

并能够自动检测和修复各种故障。因此，在目前开源的分布式存储系统里，Ceph 最适合做虚拟化存储系统的底层存储系统。

1.2.2 存储虚拟化技术的研究现状

在传统的存储体系中，存储设备与主机直接相连，并且存储设备之间是彼此独立的，通过简单的添加新的存储设备来扩展系统存储容量。这种体系下，完全没有发挥现有存储资源的潜在容量。此外，由于存储设备之间的容量、设备类型之间的差异，除了购买存储资源的成本以外，还要付出更多的管理成本。在这种日益增长的存储空间和随之而来的管理压力之下，为了应对大型、复杂和异构的存储环境的建立和管理，存储虚拟化技术迅速成为了存储领域的焦点。

虚拟化是指将简单文件，逻辑卷或其他存储对象（如磁盘驱动器）呈现给应用程序，从而允许存储管理员对应用程序隐藏存储的物理复杂性。存储虚拟化将各种物理存储设备合并到数据中心，创建存储环境的虚拟视图。客户端不知道他们使用的文件实际在哪里，也意识不到存储数据的介质的类型。

虚拟化技术起源于 IBM 大型机时代[9]。70 年代初，虚拟化概念开始从最初的处理器发展到存储子系统[10]，在 70 年代中期， IBM 3850 大容量存储系统使用虚拟化技术使螺旋扫描磁带盒在大型机操作系统看来像磁盘驱动器。在 1992 年为大型机系统建立第一个廉价磁盘冗余虚拟阵列时，存储虚拟化技术达到了一个新的水平，通过将数据压缩与虚拟化相结合，客户能够存储的数据超过阵列磁盘驱动器的实际容量[11]。目前通用的存储虚拟化技术主要包括网络附加存储（Network Attached Storage，NAS）、存储区域网络（Storage Area Network，SAN）和基于 IP 网络的存储[12-14]。

NAS 是以网络文件服务器为起点的共享存储方案，通过计算机网络为不同操作系统的异构客户端提供数据访问。NAS 通过硬件、软件或配置文件来提供专业的文件共享服务。 NAS 系统的服务器包含一个或多个存储驱动器，由不同的逻辑硬盘或磁盘阵列组成。NAS 使用网络文件系统（Network File System，NFS）协议、服务消息块（Server Message Block，SMB）协议和通用网络文件系统（Common Internet File System，CIFS）协议等文件共享协议来实现不同操作系统间的文件访问[15-17]，与同时提供文件服务的通用服务器相比，潜在优势是更快的数据访问、更容易的管理和更简单的配置，缺点是只能在局域网范围内使用。

SAN 模型将存储设备放置在自己的专用网络上，通过从服务器到磁盘的小型计算机系统接口（Small Computer System Interface，SCSI）总线和专用的用户网络来进行数据传输[18]。SAN 包括与局域网连接的多个服务器、交换机、集线器和大量存储设备。SAN 通过光纤通道交换技术实现服务器到存储设备的连接，并可以连接到范围更广泛的服务器和存储设备。SAN 在数据可用性降低了服务器的依赖性，缺点是需要布置光纤通道，硬件成本比较高。

基于 IP 的存储指的是块级数据在 IP 网络做到与 SAN 光纤通道类似的实现，通过千兆/万兆以太网技术可在总体性能、长距离传输、管理、成本和操作性方面优于光纤通道技术。基于 IP 的存储目前正处于广泛研究中,目前包括基于 IP 的光纤通道（Fibre ChanneI over IP，FCIP）、网络光纤通道协议（Internet Fibre ChanneI ProtocoI，iFCP）和互联网小型计算机系统接口（Internet Small Computer System Interface，iSCSI）协议等技术[19,20]。

FCIP 使用光纤通道和 IP 主干网进行组合连接，iFCP 虽然用 IP 网络取代了光纤通道管道，但同时保留了光纤通道设备连接至 IP 存储网络的功能。与这些协议相比，iSCSI 则将光纤通道完全排除在讨论范围之外。在 iSCSI 协议中，客户端的应用通过网络直接与存储设备进行数据交换，这意味着从客户端到存储设备之间的传输路径是不经过其他介质的直接 IP 链路。iSCSI 协议简而言之就是把 SCSI 协议的命令集映射到 IP 网络里，将 SCSI 命令集的原始形式和 IP 网络结合起来[21]，可以在运行 IP 协议的硬件设备上使用。

NAS 只能在局域网内使用，SAN 更依赖于硬件，iSCSI 协议可以使 SCSI 数据块在普通的 IP 网络上传输，而 SCSI 和 IP 分别是目前使用最广泛的块级数据传输标准和网络传输协议，因此 iSCSI 协议具有很强的通用性。iSCSI 协议具有高度兼容的开放性、良好的传输速度、无可比拟的安全性，

是当前最流行的网络存储技术。因而，本论文在 Ceph 分布式存储系统实现虚拟化的过程中，采用 iSCSI 作为网络存储虚拟化的技术手段，来实现远端客户对存储系统的访问。

1.3 研究内容

本论文的目标是将分布式存储系统和存储虚拟化技术结合起来，充分利用二者的优势来实现一个基于分布式系统的虚拟化存储架构，逻辑化所有的物理存储资源，为用户提供一个可扩展、高可用和安全的存储资源池，进行统一的配置和管理，具体研究内容主要包括以下几点：

（1）对存储资源进行管理

在传统主机和存储设备直接相连的结构中，用户和企业无法充分共享现有资源，而且存储设备的故障保护和负载均衡也存在问题。本文使用分布式技术将数据负载分担到存储系统底层的多个计算机节点上，提供一个逻辑存储资源池，在资源池内根据用户的应用需求分配存储空间。用户权限通过定义存储池基于主机的访问控制列表来实现，不同的用户可以通过任意一台主机访问共享的存储资源。

（2）用户登录认证

用户在向存储系统发出读写请求时，需要先进行登录安全认证。本文采用类似 TCP 三次握手式步骤进行身份校验。为了尽可能保证用户安全，避免校验过程中传送用户的真实密码，采用 MD5 加密算法来处理用户登录参数。在存储端身份校验通过之后，客户端和存储端之间可建立安全会话。

（3）数据交互

用户对存储系统的访问是不受地域和距离限制的，数据需要在网络上进行远距离传输。本文在客户端和存储端之间建立 iSCSI 会话，定义了在 iSCSI 会话中传输的数据格式，即 iSCSI 协议数据单元，并在存储端解析数据报文，将用户的读写命令和数据下移到底层分布式存储集群。同时，对底层数据分布算法做了优化，避免了由于增删存储节点带来的不必要的数据迁移。

1.4 论文结构

本文共分为六章，文章的组织结构安排如下：

第一章是整篇论文的绪论，开篇讲述了传统存储技术的局限，由此引出分布式存储系统和存储虚拟化的背景、意义和在国内外的研究现状，最后是本论文的主要研究内容和论文结构。

第二章主要是分布式和虚拟化的相关技术介绍，研究了 Ceph 分布式存储系统的基础结构和功能组件，介绍了 Ceph 系统的存储模式、读写流程以及 CRUSH 数据分布算法，同时研究了 iSCSI 协议的模型、命名机制和会话管理模式。Ceph 分布式存储系统和 iSCSI 协议是本论文所设计的存储虚拟化系统最重要的技术基础。

第三章是对存储虚拟化系统的总体设计，进行了需求分析，明确设计目标，以及介绍了系统整体工作流程和各个子模块的功能。

第四章实现了一个基于 Ceph 分布式系统的存储虚拟化系统，利用现有物理存储资源建立一个可动态扩展的虚拟资源池，描述了系统整体架构和各个子模块的实现细节。

第五章使用 IOzone 工具对上述存储系统进行测试，调整不同的参数测试系统性能，对测试数据进行全方位的分析。

第六章总结本文的主要工作，指出当前技术还存在的局限性，展望未来的发展趋势。

第二章 分布式和虚拟化的相关技术

本论文使用 Ceph 分布式存储系统把存储读写的工作负载分流在不同的节点上，同时使用 iSCSI 协议来打破存储设备的地域限制，本章将分析 Ceph 的架构组件、功能特性和数据分布策略，以及 iSCSI 协议的模型、命名机制和会话管理模式。

2.1 Ceph 分布式存储系统

Ceph 是一个在单个集群内融合块设备、对象存储和文件存储等多种主流存储方式的强大和统一的分布式存储系统[22]，使用 CRUSH 算法来将各种类型的数据划分成小对象，并通过动态计算得到存储位置，实现故障域隔离和负载均衡。

2.1.1 Ceph 架构

Ceph 提供对象级别、块级别和文件级别的存储服务，如图 2.1，在系统内部，这些不同的存储服务都基于可靠、自主的分布式对象存储（Reliable, Autonomic Distributed Object Store，RADOS）底层。RADOS 层中运行着多个守护进程，每个守护进程执行特定任务。监控器（monitor，MON 进程）管理整个集群范围的节点信息，对象存储设备（object-based storage device，OSD 进程）负责检索和存储对象、数据校验恢复和心跳检测，元数据服务器（metadata server，MDS 进程）在文件级存储服务中维护每个文件的元数据[23,24]。

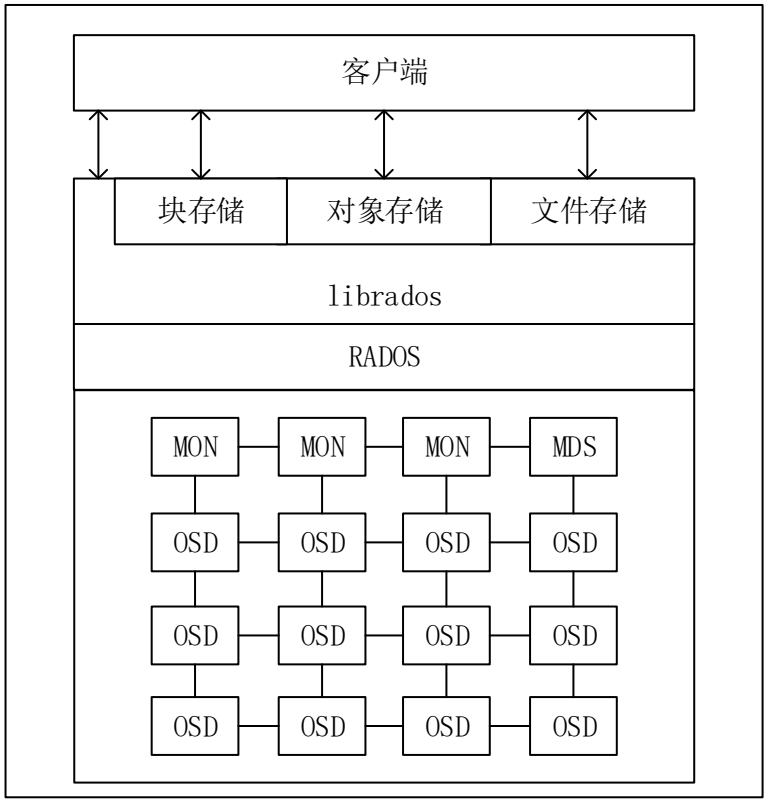

图 2.1 Ceph 架构图

libraos 是一个允许应用程序直接与 RADOS 通讯的 C 语言库[25]，可通过访问 RADOS 完成各种操作。将 librados 进行再次封装可以得到集成块存储、对象存储和文件存储的接口。

块存储是目前最流行的存储方式之一。块表示字节的连续序列，Ceph 块存储将数据块划分为一

组大小相等的对象，然后以分布式模式条带化到 RADOS 集群中。Ceph 对象存储将大文件分割为小的对象，为了方便用户执行各种操作，可兼容 Swift 和 S3 标准。Ceph 文件系统为用户提供了 POSIX 兼容的文件系统接口，与前两种存储方式不同，使用 MDS 守护进程来分离元数据和数据，可有效降低操作复杂性。

RADOS 是 Ceph 分布式系统的基石，在 Ceph 中处于最底层和最核心的位置[26]。下面分别对组成 RADOS 的三个进程的基本功能做简要的介绍。

（1）MON

monitor 负责监视整个集群的运行状况，这些集群信息都由维护集群成员的守护进程 MON 来提供[27]。monitor 通过组成监控集群来保证自己的高可用，监控集群通过 Paxos 算法[28]来实现自己数据的一致性。它提供了整个存储系统的节点信息等全局的配置信息，主要包括 monitor map，OSD map 和 MDS 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，IP 地址和端口号；OSD map 包括所有 OSD 的列表和状态；MDS map 则是所有 MDS 的列表和状态。

（2）OSD

RADOS 对象是数据存储的基本单元，默认 4MB 大小。图 2.2 是一个对象的示意图，一个对象由对象 ID、数据和元数据三个部分组成。对象 ID 唯一标识一个对象；对象的数据在本地文件系统中对应一个文件；对象的元数据，以键值对形式保存在文件对应的扩展属性中。

对象ID	数据	元数据
2834	01010101011001001000100101001001010100100101	name-value

图 2.2 对象示意图

OSD 是 Ceph 分布式文件系统的对象存储守护进程，一个 OSD 对应一个真实的硬盘[29]。OSD 是 Ceph 存储集群最重要的组件，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 守护进程来实现的。

Ceph 集群一般情况下都包含多个 OSD，对于任何读写操作请求，客户端从 monitor 获取到信息以后，将直接与 OSD 进行 I/O 操作的交互，而不再需要 monitor 干预。因为没有其他额外的层级数据处理，数据读写过程比其他存储系统更为迅速。

Ceph 提供通过分布在多节点上的副本来实现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。每个 OSD 作为某些对象的主 OSD 的同时，也可能作为某些对象的从 OSD。从 OSD 受到主 OSD 的控制，主 OSD 在磁盘故障时，OSD 守护进程将协同其他 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 的可靠和一致性。

（3）MDS

元数据保存数据的属性，如文件存储位置、文件大小和存储时间等，负责资源查找、文件记录和记录存储位置等工作。MDS 主要负责 Ceph 文件系统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[30]。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 进入备用状态。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。

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，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信息。MDS 会赋予客户端读和缓存文件内容的权限，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 执行文件交互。同样，当客户端对文件执行写操作时，MDS 赋予客户端带有缓冲区的写权限，客户端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 中的对象中。

Ceph 把形成目录层次的子树映射到 MDS，只有当单个目录成为热点时，才会根据当前工作负

载会在多个节点之间对其进行哈希处理。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的策略，即横跨多个 MDS 节点的目录层级结构。在这种这种策略下，每个 MDS 统计和记录自己的负载情况，定期根据负载情况适当地迁移子树以实现工作流分散在每个 MDS 中。

2.1.2 数据读写

pool 是一个抽象的存储池，规定了数据冗余的类型以及对应的副本分布策略[31]。目前实现了两种 pool 类型：多副本类型和纠删码类型。一个 pool 由多个放置组（placement group，PG）构成。

PG 是对象的集合，该集合里的所有对象的副本都分布在相同的 OSD 上。一个对象只能属于一个 PG，一个 PG 对应于放置在其上的 OSD 列表，一个 OSD 上可以分布多个 PG。以图 2.3 所为例，其中，PG1 和 PG2 都属于同一个 pool，都是两副本类型。PG1 和 PG2 里都包含许多对象，PG1 上的所有对象的主从副本分布在 OSD1 和 OSD2 上，PG2 上的所有对象的主从副本分布在 OSD2 和 OSD3 上。PG1 和 PG2 的从副本都分布在 OSD2 上。

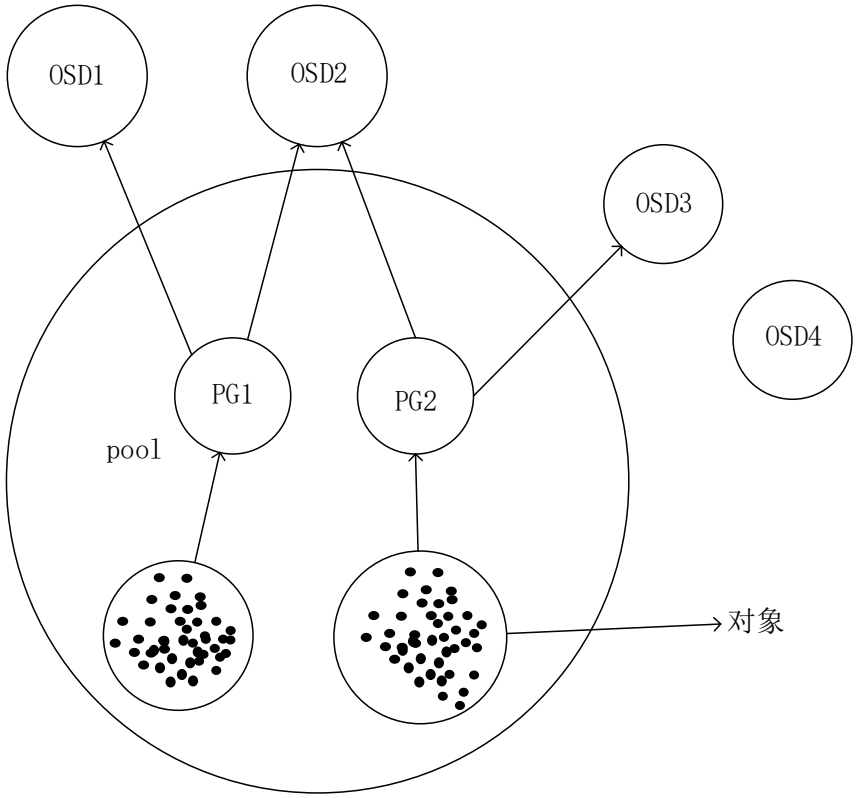

图 2.3 PG 概念图

对象寻址过程指的是查找对象在集群中分布的位置信息，过程分为两步：

（1）对象到 PG 的映射。这个过程是静态 hash 映射，通过对象 ID object_id 计算出 hash 值，用该 pool 的 PG 的总数量 pg_num 对 hash 值取模，就可以获得该对象所在的 PG 的 id 号：pg_id = hash(object_id) % pg_num。

（2）PG 到 OSD 列表映射。这个过程指的是计算 PG 上对象的副本在 OSD 上的分布策略，使用 CRUSH 机制来实现，本质上是一个伪随机分布算法。

当定位到 OSD 后，就可以进行数据读写了。数据读写操作流程如图 2.4 所示。

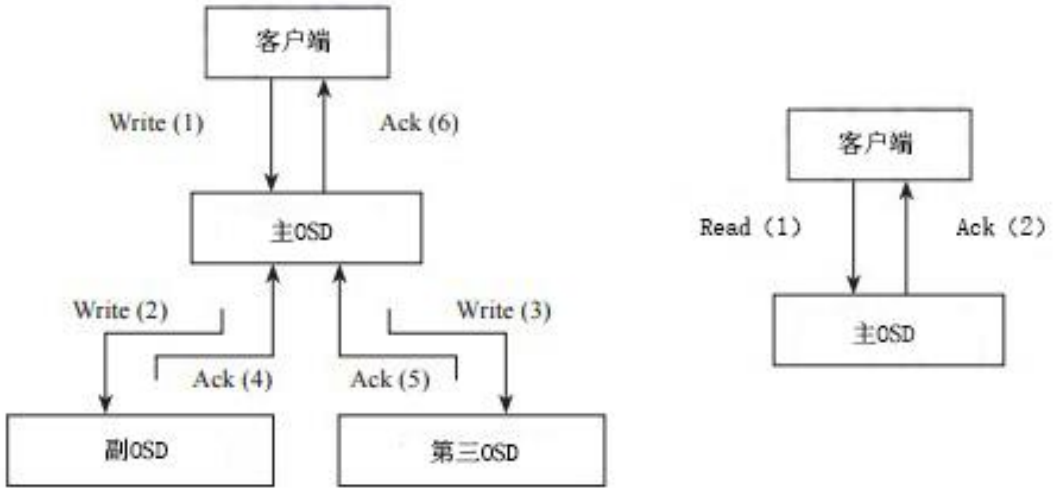

图 2.4 数据读写流程图

写操作流程如下：

（1）客户端向该 PG 所在的主 OSD 发送 write（写）请求。

（2）主 OSD 接收到写请求后，同时向两个从 OSD 发送写副本的请求，并同时写入主 OSD 的本地存储中。

（3）主 OSD 接收到两个从 OSD 发送 write 成功的 ACK 应答，同时确认自己写成功，就向客户端返回写成功的 ACK 应答。

在写操作的过程中，主 OSD 必须等待所有的从 OSD 返回正确应答，才能向客户端返回写操作成功的应答。

数据读操作只需要向主 OSD 发送读请求即可等待反馈。

2.1.3 CRUSH 算法

在过去的三十年中，所有的存储机制都涉及存储数据和保存这些数据在存储阵列位置信息的元数据。每次有新的数据被添加到当前存储系统中时，元数据最先更新，更新的内容是数据将会存放的物理位置，实际的数据存储在元数据更新之后。当存储空间比较小，数据量是从 GB 到 TB 的量级时，这个过程被证明能够很好地工作，但是如果到了 PB 或者 ZB 时，这种机制就不太适合。此外，它还造成了存储系统的一个单点故障，即在元数据失效的情况下，理论上将丢失所有的数据。虽然可以通过在单个节点上复制整个数据和元数据构造多个副本这种保证更高的容错度的方法来防止核心元数据受损坏，但是此类复杂的元数据管理机制是存储系统在可伸缩性、高可用性和性能上的瓶颈[32]。

（1）CEUSH 层级结构

cluster map 是 Ceph 集群拓扑结构的逻辑描述形式[33]。在实际应用中，Ceph 集群通常具有形如“数据中心一机架一主机一磁盘”这样的树状层级关系，所以 cluster map 可以使用树这种数据结构来实现——每个叶子节点都是真实的最小物理储设备（例如磁盘），称为 device；所有中间节点统称为 bucket，每个 bucket 可以是一些 devices 的集合，也可以是低一级的 buckets 集合；根节点称为 root，是整个集群的入口。每个节点都拥有唯一的数字 ID 和类型，以标识其在集群中所处的位置和层级，但是只有叶子节点，也就是 device 才拥有非负 ID，表明其是承载数据的最终设备。device 权重代表存储节点的性能，磁盘容量是影响权重大小的重要参数。节点的权重属性用于对 CRUSH 的选择过程进行调整，使得数据分布更加合理，上一级节点权重是其所有孩子节点的权重之和。对于所有层

级的每一个 bucket 而言，权重越高，被分配写入的数据量就越大[34-36]。

表 2.1 列举了 cluster map 中所有的节点（层级）类型。

表 2.1 cluster map 中常见的节点类型

类型 ID	类型名称	描述
0	OSD	device 设备对应的 OSD 守护进程
1	host	包含若干 OSD 设备的主机
3	chassis	刀片服务器的机箱
4	rack	包含若干主机的机架/机柜
5	row	包含若干主机的一排机柜
6	pdu	为机柜分配的电源插座
7	pod	一个大机房中的单间
8	room	包含若干机柜和主机的机房
9	data center	包含若干机房的物理数据中心
10	region	包含若干设备的可用区域
11	root	bucket 分层结构的根

这里需要注意的是，并不是每个 Ceph 集群都一定需要划分为 12 个层级，表中每种层级类型的名称也不固定，可以根据实际的应用需求来进行部署。

（2）基本选择算法 straw

在 Ceph 分布式存储系统中，不同的层级具有不同程度的灾难容忍程度，叫故障域，图 2.5 是一个典型的 Ceph 集群的层级结构。

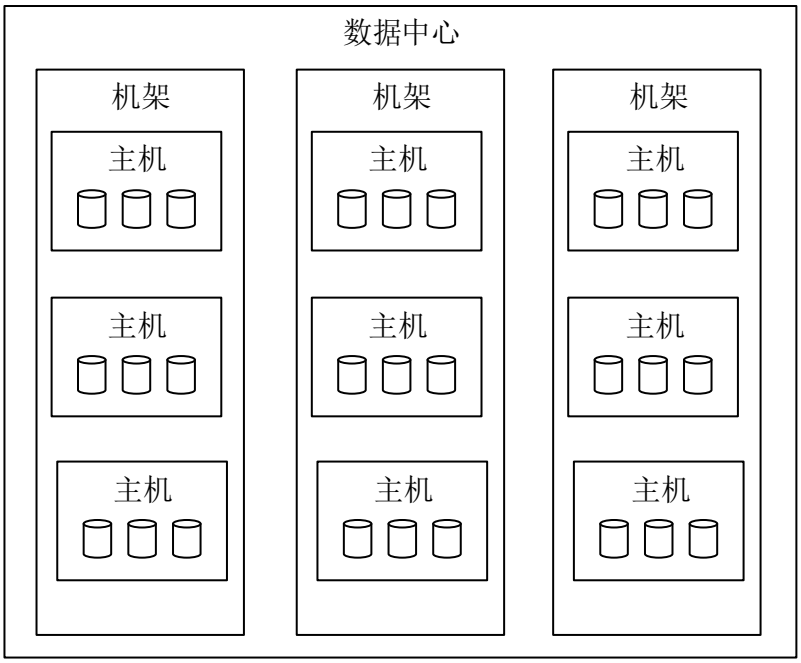

图 2.5 一个典型的 Ceph 集群层级图

在上图中，单个主机包含多个磁盘，每个机架包含多个主机并采用独立的供电和网络交换系统，从而可以将整个集群以机架为单位划分为若干故障域。为了实现高可靠性，要求数据的多个副本分布在不同机架的主机磁盘之上。因此，CRUSH 首先应该是一种基于层级的深度优先遍历算法。此外，上述层级结构中，每个层级的结构特征也存在差异，一般而言越处于顶层的结构变化的可能性越小，反之越处于底层则其结构变化越频繁，例如大多数情况下，一个 Ceph 集群自始至终只对应一个数据中心，但是主机或者磁盘数量可能一直处于变化之中。因此，从这个角度而言，CRUSH 还应该允许针对不同的层级按照其特点设置不同的选择算法，从而实现全局和动态最优。

在生产环境中需要经常添加数据，导致存储空间需求呈爆炸式增长；在大型分布式存储系统中

某些部件故障是常态，导致需要删除元素；由于愈发严苛的数据可靠性需求，导致数据副本需要存储在更高级别的故障域中，例如不同的数据中心。出于以上综合考虑，Ceph 对所有层级采用了一种叫 straw 的基本选择算法[37]。

顾名思义，straw 算法将所有元素比作吸管，针对指定输入集合，为每个元素随机计算一个长度，最后从中选择长度最长的那个元素（吸管）作为结果输出，这个过程也被形象地称为抽签，对应元素的长度称为签长。

显然 straw 算法的关键在于如何计算签长。理论上，如果所有元素构成完全一致，那么只需要将 pg_id 和 bucket_id 作为哈希输入值即可计算出对应元素的签长。因此，如果样本容量足够大，那么最终所有元素被选择的概率是相等的，从而保证数据在不同元素之间均匀分布。然而实际中存储设备随着时间也会逐渐趋于异构化，比如因为批次不同而导致的磁盘容量差异。显然，在此情况下，不应该也无法对所有设备采用同一个标准，因此需要在 CRUSH 算法中引人一个额外的参数来体现这种差异。这个参数就是上文提到的权重，如算法 2.1，Ceph 使用一个复杂的算法根据所有元素的权重计算每个元素的 scaling factor(比例因子)，比例因子大的设备分担更多的数据，比例因子小的设备分担更少的数据，使得数据在异构存储网络中也能合理地分布。

输入：对象集合 pg_id
输出：最大签长的元素
算法：

```
1   初始化元素集合，从 pg_id 里选取，记为 C={item_id1,item_id2,item_id3…};
2   for each item item_idi∈ C
3       strawi ← hash (pg_id,item_idi)*get_scaling_factor(item_idi);   //计算元素签长
4   end for
5   return 最大签长的元素
```

算法 2.1 基本选择算法

在算法 2.1 中，首先从指定 pg_id 里面初始化元素集合 C（第 1 行），然后将 pg_id 和 item_id 作为哈希输入值，再乘 scaling factor 计算出集合 C 里面每一个元素的签长（第 2-4 行），最后返回最大签长的元素（第 5 行）。

（3）CRUSH 计算规则

CRUSH 算法基于 scaling factor 将数据映射至所有存储设备之间，这个过程是受控的并且高度依赖于集群的拓扑描述——cluster map，不同的数据分布通过指定不同的归置策略实现，后者实际上是一组包括最大副本数、容灾级别等在内的自定义约束条件，通过一条归置策略将互为镜像的 3 个数据副本（这也是 Ceph 的默认数据备份策略）分别写入位于不同机架的主机磁盘之上，以避免所有副本同时掉电导致业务中断[38-40]。

针对指定输入 pg_id，CRUSH 将输出一个包含 n 个节点的集，CRUSH 的计算过程中仅仅使用 pg_id、cluster map 和归置策略作为哈希函数的输入，因此如果 cluster map 不发生变化（一般而言，归置策略不会轻易变化），那么结果就是确定的；同时因为使用的哈希函数是伪随机的，所以 CRUSH 选择每个目标存储对象的概率相对独立，从而保证数据在整个集群之间均匀分布[41,42]。

使用 cluster map 建立对应集群的拓扑结构之后，可以定义归置策略来完成数据映射[43]。

归置策略由以下步骤组成：

1）take

take 从 cluster map 选择某个被指定编号的 bucket，并以此作为后续步骤的输入，系统默认的数据分布策略是以 cluster map 中的 root 节点作为输入开始执行。

2）select

select 从输入的 bucket 当中随机选择指定类型和数量的条目（item）。Ceph 当前支持多副本和纠删码两种备份策略，相应的有两种 select 算法——firstn 和 indep。两种算法都是基于深度优先实现

的，主要区别在于纠删码要求结果是有序的，因此，如果无法得到满足指定数量（例如 5）的输出，那么 firstn 会返回形如[1，2，3，5]这样的结果，而 indep 会返回形如[1，2，3，CRUSH_ITEM_NONE，5]这样的结果，即 indep 总是返回要求数量的条目，如果对应的条目不存在（即选不出来），则使用空数据进行填充。select 执行过程中，如果选中的条目故障、过载或者与其他之前已经被选中的条目冲突，都会触发 select 重新执行，因此需要指定最大尝试次数，防止 select 陷人死循环。

3）emit

emit 向上级调用者输出最终选择结果并返回。一条数据分布策略中真正起决定性作用的是 select 操作。为了简化分布策略的配置，select 操作也支持故障域模式。以 firstn 为例，如果为故障域模式，那么 firstn 返回的指定数量的条目，保证这些设备位于不同的、指定类型的故障域之下。在故障域模式下，一条最简单的归置策略可以只包含如下 3 个操作：

take(root)

select(replicas,type)

emit(void)

上述 select 操作中的 type 为我们需要设置的故障域类型，如果设置为 rack，则 select 通过基本选择算法 straw 选择出来的所有副本需要是放置在不同机架的主机磁盘上；如果设置为 host，则 select 将所有副本放置在不同主机的磁盘上。

2.2 iSCSI 协议

本节开头阐述了 SCSI 和 iSCSI 的关系，iSCSI 协议是基于 TCP 的，实现 SCSI 协议到 TCP 网络的映射，利用 TCP/IP 网络传输 SCSI 命令和数据。请求/响应机制是 iSCSI 协议的基本实现方式，安全机制是实现 iSCSI 会话建立和数据传输的保证。通过对 iSCSI 协议的研究与分析，分析 iSCSI 系统的组成，包括 iSCSI 模型、iSCSI 命名和 iSCSI 会话结构，来深入理解 iSCSI 的本质。

2.2.1 SCSI 协议简介

SCSI 是小型计算机系统接口。与其他接口一样，SCSI 实际上是主机与设备之间相互通信的一套协议[44]。SCSI 主机利用 SCSI 主机总线适配器将存储设备或其他外部设备接入到主机的 I/O 总线，并使用 SCSI 协议管理、控制和操作这些外部设备[45]。

SCSI 命令集是 SCSI 协议的用户接口，是 SCSI 协议中最有特色的部分。用户通过 SCSI 命令请求 SCSI 设备的服务，例如，从 SCSI 设备中读数据，或向 SCSI 设备中写数据。SCSI 命令可以通过不同的传送协议和物理接口传送，例如，SCSI 总线和光纤通道等。

SCSI 标准可以支持多种不同类型的 SCSI 设备，如磁盘、磁带、打印机、扫描仪等，每类设备都有自己特定的操作方式，因而都有自己特定的命令集。然而，它们又都是 SCSI 外围设备，有着许多共同的特征，所以又有许多公用的操作方式和命令。SCSI 基本命令集定义了所有 SCSI 外围设备共有的命令。这些被定义的公共命令是所有 SCSI 设备都应该支持的。当然，一个 SCSI 目标器设备还该支持自己特定的命令集。

除了基本命令之外，SCSI 公共命令集中还定义了所有类型的 SCSI 设备都应该定义的管理参数，如诊断参数、日志参数、模式参数和重要产品参数等。应用客户通过命令描述块（Command Descriptor Block，CDB）向外围设备传递命令，有些命令还伴随着一系列的参数。

存储技术也是以市场需求为发展动力的，其主要是市场对块级别的、高速的存储访问的需求，常见的有数据库服务器等。直连式存储使用高速 SCSI 磁盘阵列，但它只能专用于某个服务器，因此它的一个阵列就是一个“数据孤岛”。为了支持互联网接口，必须完全重新组织 SCSI 标准和定义。21 世纪初成立的国际互联网工程任务组（The Internet Engineering Task Forc，IETF）对 SCSI 协议命令集与物理接口定义进行了分离[46]，并创建了新的 SCSI 体系结构模型来定义各种标准之间的交互，

以解决在 TCP / IP 基础结构上直接传输 SCSI 命令集的问题，使 SCSI 协议可以直接在互联网上传输，即 iSCSI 协议[47]。

2.2.2 iSCSI 协议模型

iSCSI 协议是一种通过 TCP /IP 网络实现块存储访问的存储网络协议，通过 IP 网络传输 SCSI 命令和存储数据，用于促进远距离数据传输和存储管理[48,49]。由于 TCP/IP 网络的普遍存在，iSCSI 可以用于通过局域网、广域网或互联网传输数据。iSCSI 协议提供了封装 SCSI 命令数据块的功能，并与千兆位和万兆位以太网传输、IP 安全性和服务质量协议相结合，为高度可扩展和安全的共享存储网络打开了新的机会。IP 存储应用程序（包括远程磁带备份和磁带保险存储、远程镜像、存储整合、内容分发以及数据中心应用程序）都受益于主流 IP 网络的普遍性、可管理性和性价比优势。

iSCSI 对 SCSI 设备的远程访问可以通过任何 TCP/ IP 网络基础结构进行，所以能够使用更廉价的组件来实现，具有很大的灵活性。操作系统将远程设备视为本地访问的块级设备，对块级请求的响应可能会遇到更多延迟，具体取决于网络流量等问题。

iSCSI 协议是从 SCSI 模型到 TCP 协议的一个映射，传输命令的两端是发起通讯以请求响应器去执行相应命令的启动器和响应通讯以执行 SCSI 发起器所发起的请求的目标器。iSCSI 协议采用客户端/服务器摸型，启动器在 iSCSI 请求中发送 SCSI 命令及参数，目标器在 iSCSI 应答中返回状态及结果。

iSCSI的工作方式是在服务器上的iSCSI启动器和存储设备上的iSCSI目标器之间传输块级别的数据[50,51]。iSCSI 协议封装 SCSI 命令，并将数据组装为 TCP / IP 层的数据包。数据包使用点对点连接通过网络发送。到达目标器后，iSCSI 协议将对数据包进行解析，分析 SCSI 命令，以便操作系统将存储视为可按常规格式进行格式化的本地 SCSI 设备。

图 2.6 显示了 iSCS1 协议层模型并描述了 SCSI 命令通过物理载体进行传送时的封装顺序。

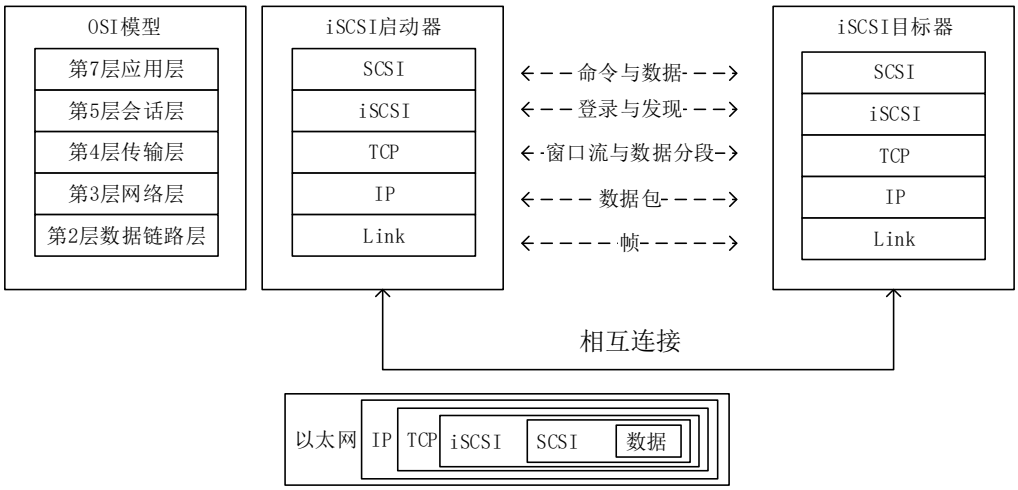

图 2.6 iSCSI 协议栈

SCSI 是工作在 OSI 模型的应用层的命令型协议。SCSI 命令描述块、数据和状态信息被封装入 TCP/IP 数据包，然后通过网络在发起方和目标方之间传输。iSCSI 是一种会话层协议，它启动了可以识别 SCSI 命令和 TCP/IP 的设备之间的可靠会话。iSCS1 会话层接口负责处理登录、验证、目标发现和会话管理。TCP 可以为 iSCSI 提供可靠的传输服务。TCP 被用来控制消息流、窗口、错误恢复和重发功能，依赖于 OSI 模型的网络层提供全局寻址和连接。此模型的数据链路层协议能够实现物理网络上节点到节点的通信。

2.2.3 iSCSI 命名

iSCSI 启动器和 iSCSI 目标器都需要一个名字来标识自己[52]。

使用名字来管理 iSCSI 存储资源的优势是忽视存储资源的具体地理位置。一个 iSCSI 节点名称是 iSCSI 的设备名称。iSCSI 名字是启动器和目标器之间相互验证的主要对象。

iSCSI 名字是与 iSCSI 节点而不是与 iSCSI 网络适配器相关联的，因而替换系统的网络适配器不需要重新配置 iSCSI 资源的分配信息。

利用 SendTargets 请求或者其他技术[53]，iSCSI 启动器可以发现与之通信的 iSCSI 目标器的名字和地址。在最终用户的操作域中，iSCSI 名字必须是唯一的。由于 IP 网络的操作域是全球范围的，因而 iSCSI 名字应该是全球唯一的。

（1）iSCSI 名字特性

每个 iSCSI 节点，不管是 iSCSI 启动器还是 iSCSI 目标器都必须有一个 iSCSI 名字，iSCSI 名字的最大长度是 223 个字节，iSCSI 启动器和 iSCSI 目标器都必须能够接收最长的 iSCSI 名字。

当一个 iSCSI 启动器与某个 iSCSI 目标器首次建立起连接之后，它在该连接上发送的第一个请求应该是登录请求，其中包含 iSCSI 启动器和 iSCSI 目标器的名字。如果正在建立的是一个恢复会话，此时 iSCSI 目标器名字可以忽略。

iSCSI 名字有下列特性：

1）iSCSI 名字是个唯一的，没有两个启动器或目标器拥有相同的名字。

2）iSCSI 名字是永久的，在一个 iSCSI 节点的生命周期中，它一直使用同一个名字。

3）iSCSI 名字不含位置或地址信息。一个 iSCSI 启动器或目标器可以移动，也可以有多个地址，地址的改变不意味着名字也随之改变。

一个 iSCSI 名字必是符合 UTF-8 编码规则的 Unicode 字符串，其编码具有下列特性：

1）不管底层协议如何，iSCSI 名字总是采用同样的编码方法。

2）iSCSI 名字的比较是相对简单的，不需要依赖于外部服务器。

3）iSCSI 名字仅由可显示的字符组成，并允许使用国际字符集，但不是大小写敏感的。iSCSI 名字中不允许出现空白字符和标点，但允许出现‘-’、‘.’、‘:’等。

4）iSCSI 名字可以利用二进制或基于 ASCII 的协议传送。

iSCSI 名字实际是为一个逻辑的软件实体所起的名字，而不是为端口或其他容易改变的硬件所起的名字。一个 iSCSI 启动器名字所表示的是 iSCSI 启动器节点，而不是个特定的网卡。相似地，一个 iSCSI 目标器名字也不应该与某个容易改变的硬件接口绑定，它所表示的是逻辑上的目标器，而不必理会其物理部分和地址。

iSCSI 名字应该遵循统一资源名（Uniform Resource Name，URN）约定。

（2）iSCSl 名字结构

一个 iSCSI 名字由两部分组成：类型指示符和名字字符串。

类型指示符有两种：iSCSI 限定名（iSCSI Qualified Name，IQN）和扩展的唯一标识符（Extended Unique Identifier，EUI），即 IEEE EUI-64 格式名[54,55]。

1）IQN 格式名

IQN 是一种基于域名的名字结构，使用者必须拥有已注册的域名。这里的域名不需要是活动的，也不需要解析成 IP 地址，仅需要保证没有其他用户在使用该域名即可。由于域名有时间限制（可能过期），因而在域名中必须加入有效期。

IQN 格式的 iSCSI 名字由下列几部分组成：串“iqn.”、日期编码（格式为 yyyy-mm）、颠倒的域名、可选的‘:’和可以包含产品类型以及厂家标识的字符串。

假如“Company Storage，Inc. ”公同拥有域名“company.con”，下面是一些合法的 iSCSI 名字：

iqn.2017-12.com.company:storage:diskarrays

iqn.2018-05.com.company:

iqn.2018-06.com.company:storage-ceph

2）EUI 格式名

IEEE 注册机构提供 IEEE 服务，用来分派全局唯一的标识符。

如果用户已经向 IEEE 注册机构注册，并已使用 EUI-64 格式名为其产品命名，就可以使用 EUI 格式的 iSCSI 名。

EUI 格式的 iSCSI，名字由两部分组成：“eui.”+EUI-64 格式的全局标识。EUI-64 格式的全局标识符是一个十六进制数，长度为 16 个字符，计 64 位。eui.03004256A534678D 是个合法 iSCSI 名字的例子。

2.2.4 iSCSI 目标器发现

iSCSI 启动器建立以后，首先需要确定它能访问的 iSCSI 目标器，然后才能与之建立会话并存取其中的数据，确定可访问 iSCSI 目标器的过程称为目标器发现。

iSCSI 启动器可以采用下列方法发现 iSCSI 目标器[56]：

（1）iSCSI 目标器的地址配置在启动器中。

（2）在 iSCSI 启动器中配置一个默认的 iSCSI 目标器地址，启动器先与该目标器建立连接，而后通过 SendTargets 正文请求和应答获取它可访问的 iSCSI 目标器列表。

（3）广播一个服务定位协议（Service location Protocol，SLP）报文，获得目标器的地址[57]。

（4）查询互联网存储名字服务（Internet Storage Name Service，iSNS）服务器，获取 iSCSI 目标器列表[58]。

SLP 是一种简单的发现协议，适用于小范围 iSCSI 目标器发现。在使用时，iSCSI 启动器广播一个 SLP 报文，其中包含要查询的目标器的名字。收到 SLP 报文的目标器检查其中的名字，如果报文中的名字与自己的名字一致，则应答。其过程类似于 ARP 协议。

iSNS 相对复杂，功能也更强大，iSNS 通过一组类似于光纤通道上的服务，改善了 iSCSI 中存储设备的配置与管理。iSNS 的管理员不再需要手工配置各存储设备的 iSCSI 启动器和目标器列表。在 iSNS 中，iSNS 服务器负责各存储设备的发现和管理，因而可将 iSNS 服务器看成是统一的配置点，通过该配置点管理工作站能够配置和管理整个存储网络，包括 iSCSI 和光纤通道设备。

iSNS 主要包含以下四个功能：

（l）名称服务（提供存储资源发现）。

（2）发现域和登录控制服务。

（3）状态变化通知服务。

（4）光纤通道和 iSCSI 设备的开放映射。

iSNS 系统主要由以下几部分组成：

（l）iSNS 协议：iSNS 协议规定了 iSNS 客户机和服务器间的通信方式。该协议适用于多种平台，包括交换机、目标器和服务器主机等。

（2）iSNS 客户机：iSNS 客户机通过 iSNS 协议启动与 iSNS 服务器的事务。iSNS 客户机用于注册设备信息，加载公共域中其他客户机的信息，以及接收公共域中的事件异步通知。

（3）iSNS 服务器：iSNS 服务器响应 iSNS 查询请求、发起 iSNS 状态变化通知，并将各存储设备在注册请求中提交的经过验征的信息存储于 iSNS 数据库中。

（4）iSNS 数据库：iSNS 数据库是为 iSNS 服务器提供的信息库，维护 iSNS 客户机的属性信息。

iSCSI 目标器是一种 iSNS 客户机。当 iSCSI 目标器启动时，它向 iSNS 服务器注册并登记自己的信息如名字、地址等。iSNS 服务器将各客户机注册的信息保存在 iSNS 数据库中。此后，当 iSCSI 启动器向 iSNS 服务器查询时，iSNS 服务器即可向其通报它可以使用的 iSCSI 目标器列表。

另外，iSCSI 协议提供了专门的 SendTargets 服务，一个 iSCSI 启动器可以通过该服务获取它可访问的 iSCSI 目标器列表，包括目标器名称、地址和端口号等。

2.2.5 iSCSI 会话

在 iSCSI 连接中，iSCSI 目标器对网络另一端的启动器来讲是透明的。iSCSI 启动器和 iSCSI 目标器之间的通讯称为会话(Session)。会话中的通信被分割成消息，也就是 iSCSI 协议数据单元（Protocol Data Unit，PDU)。iSCSl 通过 PDU 传送请求和应答[59]。一个 iSCSI PDU 由 iSCSI 头（一个基本头和多个附加头)和数据段组成，头部和数据段之后可以选择性地追加一个循环冗余码（cyclic redundancy check，CRC）校验和[60]。

出于性能方面的考虑，iSCSI 可以将命令与数据包装在一起从启动器传送到目标器，也可以将数据和应答包装在一起从目标器传送到启动器。

iSCSI 启动器和目标器之间通过一个或多个 TCP 连接相互通信，TCP 连接用于传送控制信息、SCSI 命令、参数和数据。TCP 连接由启动器生成的连接标识符（Connection ID，CID）标识[61]。

如图 2.7 所示，在 iSCSI 启动器和目标器之间的会话是由一组 TCP 连接构成的。可以动态地向一个会话中增加 TCP 连接，也可以从会话中删除 TCP 连接。

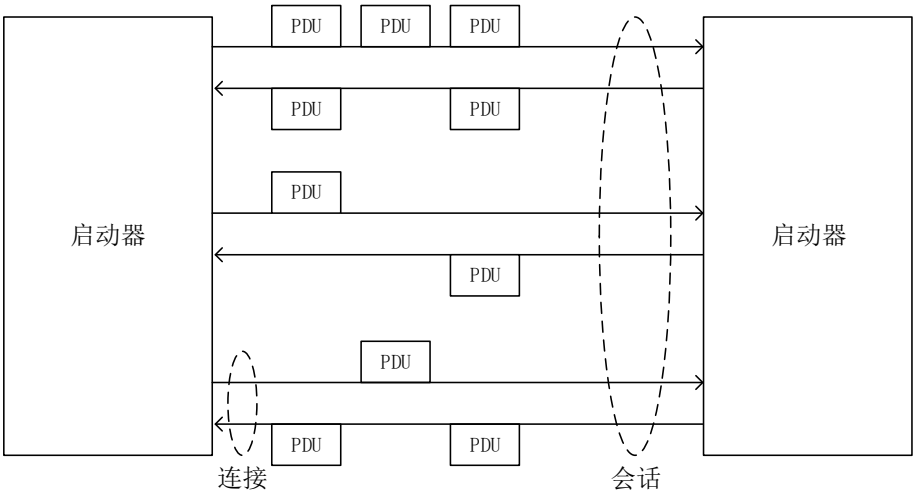

图 2.7 iSCSI 会话图

iSCSI 会话是由 iSCSI 启动器和 iSCSI 目标器使用专门的登录请求和应答 PDU 建立起来的。

iSCSI 目标器首先启动并在众所周知的 TCP 端口（或其他 TCP 端口)上监听到来的 TCP 请求。iSCSI 启动器利用自己的网络端口与 iSCSI 目标器建立 TCP 连接，而后在该连接上发出登录请求。在一个 TCP 连接上发出的第一个 PDU 是登录请求。iSCSI 目标器接到登录请求后，返回登录应答。此后，双方经过多次的请求和应答，相互认证并协商会话参数，一旦认证和协商成功，iSCSI 启动器和目标器之间的会话也就随之建立起来。

由启动器发送到目标器的数据（用户数据或命令参数）称为输出 SCSI 数据，由目标器发送到启动器的数据称为输入 SCSI 数据。

输出数据又分为征求性数据和非征求性数据[62]。征求性数据是应目标器的预备传输（Ready to Transfer，R2T）请求而传送的数据。非征求性数据是在未受目标器征求的情况下由启动器主动传送的数据，它可能被包含在 iSCSI 命令 PDU 中，也可能通过单独的 iSCSI 数据 PDU 传送。在 iSCSI 命令 PDU 中包含的数据又称为立即数据。在写操作中，允许传送立即数据可以减少协议的通信量。

立即数据总是位于要输出数据（如 SCSI Data-Out Buffer)的开始位置（偏移量为 0）。其余数据的偏移量必须在 Data-Out PDU 中显式说明。

所有后传送的数据都应该是征求性数据。也可能目标器仅允许传送立即数据而不允许再传送其他形式的非征求性数据（在单独的数据 PDU 中）。是否允许传送立即数据、是否允许传送分立的非征求性数据，以及允许传送的非征求性数据的最大长度等参数，都是在登录阶段双方协商的结果。

正常传送的输出数据应该是征求性数据：目标器发送 R2T 请求，启动器根据请求的要求传送数据。启动器必须尊重来自目标器的 R2T 请求 ，并按其请求传送数据，只要请求是合法的。

启动器不需要追踪它已经传送或接收到的数据。目标器负责追踪记录一条命令所传送的数据，并在数据输入 PDU 中提供剩余计数。数据 PDU 和命令之间的对应关系由 PDU 中的 Target Transfer Tag 域表示。另外，数据的传送必须遵守一定的顺序规则。如果使用非征求性数据，非征求性数据的顺序必须与命令的顺序一致。只要满足序号规则，命令和非征求性数据 PDU 就可以在一个连接上交叉传送，即允许在发送第 N 条命令的非征求性数据之前发送第 N+l 条命令。但第 N 条命令的非征求性数据一定要在第 N+1 条命令的非征求性数据之前发送，否则将引起会话的终止。

2.3 本章小结

通过 Ceph 分布式集群把存储工作负载分担到多个计算机节点上，在集群中，Ceph 充分利用了诸如 monitor、OSD 等组件，它们具备容错性、高度可伸缩性和高性能。Ceph 使用 CRUSH 算法来将数据存储到物理磁盘上。任何类型的数据（无论是来自 Ceph 块设备、对象存储还是文件系统）都会被分割为小对象的形式，然后存储到一个动态计算的数据存储位置上，以此来实现故障域隔离和负载均衡。这种机制使得 Ceph 脱颖而出，成为高度可伸缩的、高可靠的和高性能的分布式存储系统。iSCSI 协议是基于 TCP 协议的，实现 SCSI 协议到 TCP 网络的映射，利用 TCP/IP 网络传输 SCSI 命令和数据，可以打破地域和距离限制，为远端的用户提供可以共享访问的存储资源。

第三章 存储虚拟化系统的设计

本章首先进行虚拟化系统的需求分析，明确了该系统的设计目标，然后描述了系统整体工作流程和各个子模块的功能。

3.1 需求分析

通过对 Ceph 分布式存储系统和 iSCSI 协议的分析研究，总结出功能需求、性能需求和安全性需求三个方面的内容。

3.1.1 功能需求

（1）存储资源管理

1） 目标器创建

首先采用 Ceph 分布式技术对现有物理资源进行管理，目的是把充分利用现有资源，以及把读写负载分担到众多的计算机节点上。目标器在存储虚拟化系统里指的是一组逻辑存储卷的集合。目标器创建包括建立一个虚拟存储资源池。目标器有两个标识符：数字 ID 和名称，可以通过其中一个标识符来为目标器添加逻辑存储卷。

2）目标器资源删除

目标器的删除包括对目标器的删除和对目标器内某个逻辑存储卷的删除。若删除目标器，则目标器中的所有逻辑存储卷与启动器的会话全部断开，与该目标器绑定的用户失效；若只是删除某个逻辑存储卷，用户对其他存储资源的访问不受影响。

3）用户创建

用户不是单独存在的，依赖于目标器。用户权限通过定义目标器基于主机的访问控制列表来实现，若目标器与多个客户端的启动器之间建立会话，则创建的用户可以通过任何一台启动器访问存储资源。

4）用户删除

用户可以单独删除，用户的删除不影响目标器和逻辑存储卷。

（2）登录安全认证

使用类似 TCP 三次握手的步骤进行存储端对客户端用户的身份认证。

（3）数据交互模块

存储端和客户端之间建立会话，存储端解析从会话上传输过来的数据包，得到客户端的读写命令和数据后，下移到底层 Ceph 分布式集群完成 I/O 操作。

3.1.2 性能需求

（1）系统容量需求

目标器内逻辑存储卷是可扩展的。目标器在这个存储池里，可以容纳广阔的逻辑地址空间，整

个虚拟化存储系统对目标器和逻辑存储卷的数量与容量没有任何限制，按照用户的应用需求添加大容量的存储资源。

底层物理资源同样是可扩展的。支持成百上千甚至更多的节点，充分利用硬件资源，将存储读写的工作负载都能动态地划分到各个服务器上。当动态加入新的物理存储资源时，系统需要自动智能调整数据存储负载，通过数据迁移使新加入的设备共同承担存储任务。

（2）读写性能

支持用户在远端主机上使用如表 3.1 中各种不同的读写模式来访问该存储虚拟化系统。

表 3.1 文件访问模式表

模式	描述
write	写入新文件
rewrite	从头到尾重写一个已存在的文件
read	从头到尾读取一个现有文件
reread	重新读取最近已经读过的文件
random write	在一个文件中的随机位置进行数据写入
random read	通过访问文件随机位置来读取整个文件
backwards read	从尾到头反向读取一个文件
record rewrite	写入和重写文件特定位置
stride read	按照固定大小的偏移量读取文件
fwrite	使用库函数 fwrite()来写入文件
refwrite	使用库函数 fwrite()重写一个已存在的文件
fread	使用库函数 fread()读取一个已存在的文件
refread	使用库函数 fread()再次读取一个最近读过的文件

3.1.3 安全性需求

虚拟化存储系统包括两个方面：数据的访问安全性，数据的传输和存储安全性。

（1）访问安全性

用户必须与目标器进行绑定，同时，在登录期间，目标器要认证启动器，启动器也可以认证目标器，而且每一个新加入的 iSCSI 连接都要经过认证。认证的方法是双方协商的。

（2）传输和存储安全性

由于 iSCSI 数据包是在 IP 网上传输的，因而可能出现数据错误（如校验出错），也可能出现 PDU 丢失。存储系统必须能够恢复这些错误和丢失的数据包。另外，TCP 连接本身也可能出现故障，目标器必须能够为活动的任务指派新的连接，并让它在新的连接上继续执行。

当底层的某块物理资源失效或断电以后，采用弹性的数据分布策略和物理拓扑输入实现高可用性和高持久性。

3.2 工作流程

整个系统的工作流程如图 3.1 所示。

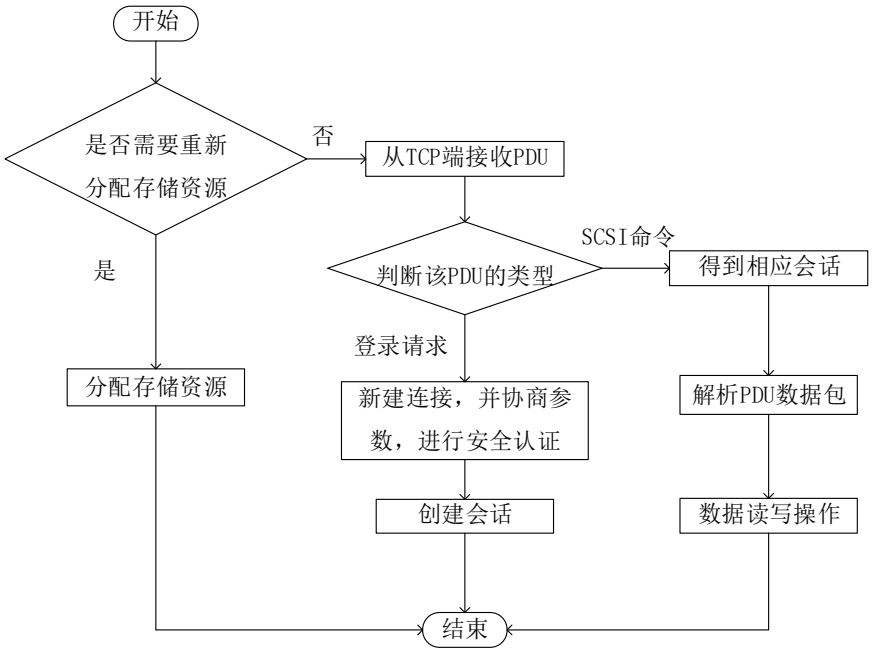

图 3.1 系统工作流程图

（1）首先根据用户需求决定是否要分配存储资源，包括目标器和附属它的逻辑存储卷。若需要，则使用存储资源模块进行存储池、存储卷和授权用户的创建。

（2）当需要进行数据读写时，根据接收到的 PDU 类型来确定是进行安全认证还是数据交互，若会话还没建立，首先要进行安全认证。安全认证是建立会话的第一阶段，由安全认证模块来完成，若通过安全认证，则开始建立会话。

（3）当接收到的 PDU 是 SCSI 命令类型，说明会话已经建立，获得相应会话，解析 PDU 数据包，然后交由数据交互模块进行各种读写操作。

3.3 总体设计

3.3.1 需求功能结构划分

整个存储系统主要包括如图 3.2 中的三个核心模块：存储资源管理模块、登录认证模块和数据交互模块。

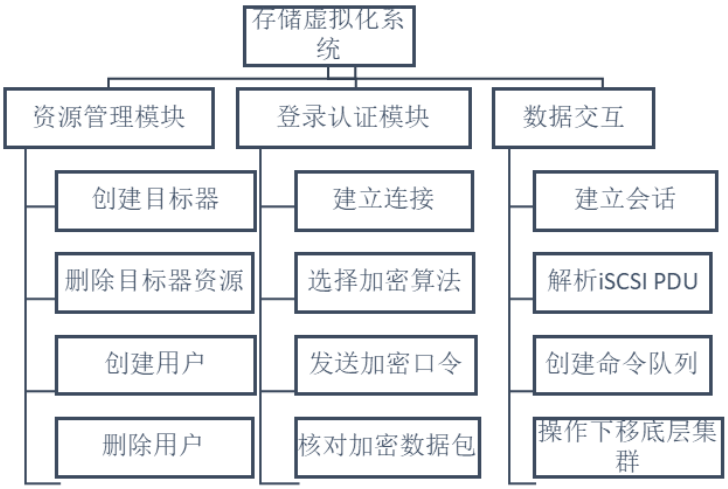

图 3.2 存储系统功能架构图

资源管理模块主要根据用户的应用访问特征和存储需求，创建目标器，分配存储资源，提供存储空间；登录认证模块在 iSCSI 启动器和目标器之间通过使用双方协商的加密算法进行相互认证，通过发送加密口令和校验加密数据包建立端到端的信任关系；数据交互模块通过解析 iSCSI 会话上传输过来的 PDU 数据包获得用户的各种读写操作请求，将这些请求下移到底层的 Ceph 分布式存储集群，完成各种 I/O 操作。

3.3.2 存储资源管理模块

系统开始运行的时候，首先要进行存储资源的配置，需要进行如图 3.3 中的一系列的操作流程。

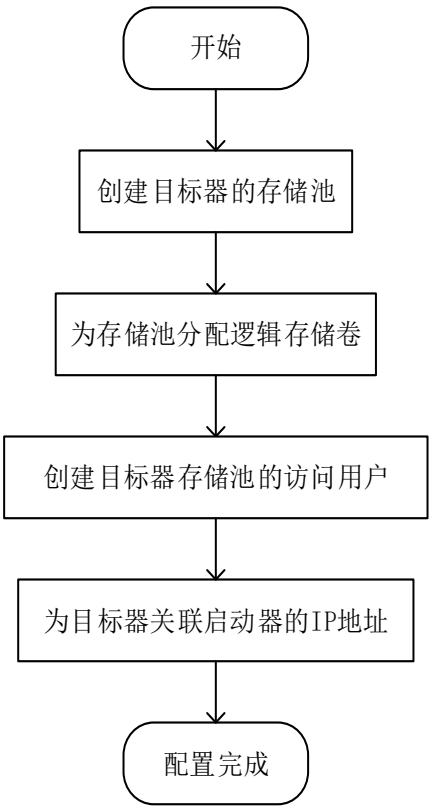

图 3.3 配置流程图

在该模块中，逻辑存储卷到 Ceph 对象集合（即第二章提到的 PG）的映射是通过一致性哈希算法实现的。在传统的哈希算法中，当 PG 数目增加时，存储卷上所有的数据空间需要重新计算映射结果，新的映射结果可能会产生比较大的变化，引起比较大的数据迁移。一致性哈希算法采用哈希环的形式，当 PG 数目增加时，存储卷的映射结果变化比传统哈希算法少，减少了数据迁移。传统哈希映射如图 3.4 所示。当增加一个 PG 时，存储卷 4、5、6 都会映射到新的 PG。替换成如图 3.5 中的一致性哈希算法后，只有存储卷 5 的映射 PG 改变了。

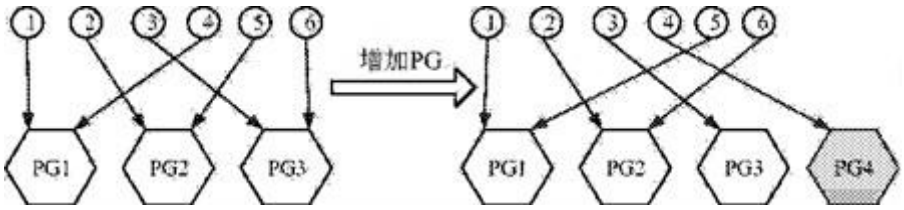

图 3.4 传统哈希处理图

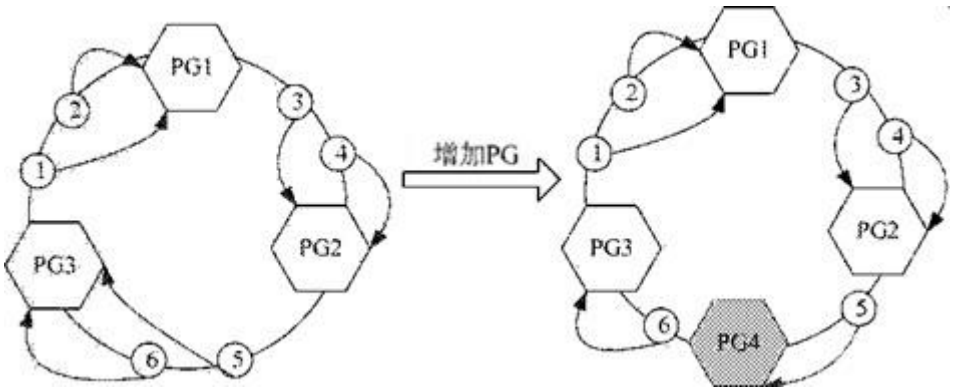

图 3.5 一致性哈希处理图

在这个模块里，我们的主要设计目标是实现客户端与服务端的传输连接和任务管理，资源管理模块的各组件如图 3.6。

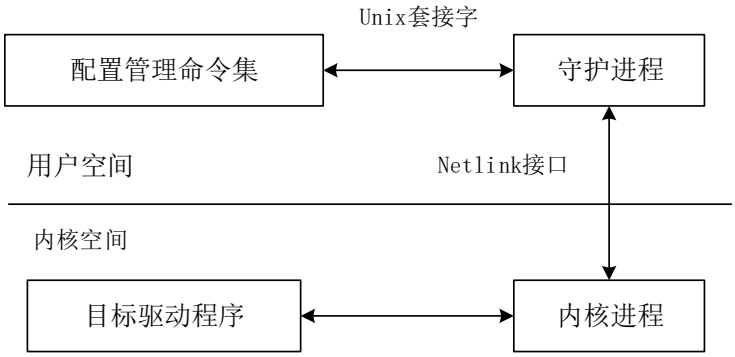

图 3.6 资源管理模块组件图

目标驱动程序的主要职责有两个：一是管理与客户端的传输连接，从传输协议数据包中获取 SCSI 命令，将其移交给数据交互模块，然后通过传输协议的数据包将响应发送回客户端；二是在 Ceph 块存储设备与内核进程之间传递任务管理请求。守护进程负责将配置管理请求异步地通过 Netlink 接口（用户空间和操作系统内核通信的方式）发送给内核进程[63]，再由目标驱动程序处理传递过来的配置管理请求。配置管理命令可以用来创建和管理目标器的逻辑单元号（存储卷），分配存储容量，同时负责与目标器相关的用户账号注册、修改和删除。

在启动该模块守护进程的时候，会初始化一个 Unix 套接字，将该套接字加到 epoll（IO 多路复用函数）中[64]，如图 3.7，开始监听 EPOLLIN 事件，当收到连接请求时，调用 mgmt_event_handler 回调函数建立 TCP 连接，并返回连接建立的响应给配置端。连接建立后，每执行一条配置命令，就会使用之前的 Unix 套接字来和守护进程通信。开始启动某一种命令时，通过 Unix 套接字发起请求，触发 epoll 的 EPOLLIN 事件，守护进程接收以后，将连接的文件描述符加到 epoll 中。当启用的命令触发 EPOLLIN 事件时，回调函数 task_recv_send_handler 开始工作，可根据不同的模式可调用不同的处理函数进行处理。若启动的是存储池类型操作命令，则使用 target_mgmt 类函数；若是存储卷模式，则使用 lun_mgmt 类函数；若要为我们创建的存储卷注册和删除用户，则使用 account_mgmt 类函数。处理完后，将事件改为 EPOLLIN|EPOLLOUT，然后发送响应给配置端。

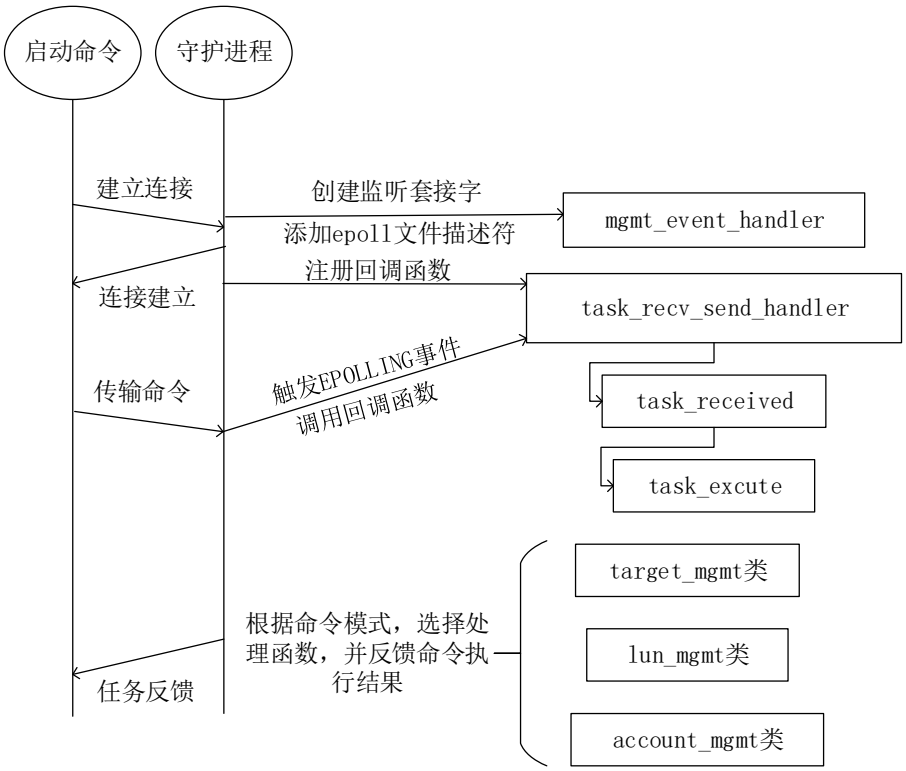

图 3.7 资源管理函数工作流程

3.3.3 登录认证模块

通过上述配置管理模块，我们已经得到了可以与客户端相互连接的逻辑存储卷，接下来，启动器要与目标器之间建立会话。启动器在登录过程中要进行安全认证。本系统采用挑战-握手验证协议（Challenge-Handshake Authentication Protocol，CHAP）实现目标器对启动器的身份认证。

CHAP 是一种加密的验证方式，能够避免建立连接时传送用户的真实密码。验证过程如图 3.8 所示，启动器向目标器发送一个 CHAP 算法列表，目标器选择一个哈希加密算法。然后，目标器向远程启动器发送一个挑战口令，其中包括会话 ID 和一个任意生成的挑战字串。远程启动器必须使用 MD5 单向哈希算法返回用户名，用户密码，加密的挑战口令和会话 ID，其中用户名以非哈希方式发送。目标器接收到这些信息以后，根据会话 ID 找到 CHAP 数据包，通过同样的哈希算法处理用户名和存在本地数据库的密码，会话 ID，以及之前的随机字串，得到一个哈希值，与接收到数据包中的哈希值进行比较，若一致，则通过认证，启动器登录成功。

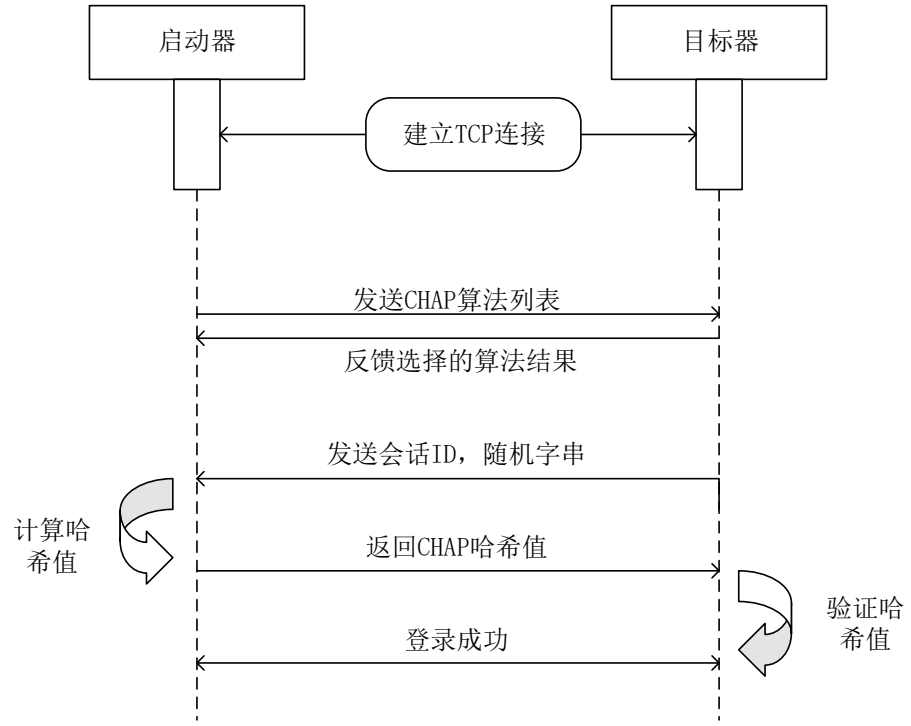

图 3.8 CHAP 验证登录流程图

登录认证的目的是建立一个会话，当启动器与目标器建立第一个 TCP 连接之后，启动器就发送登录请求，如图 3.9 中的函数工作流程，iscsi_rx_handler 函数对数据进行解析，cmnd_execute 判断解析出来的请求类型，当确认是登录请求时，cmnd_exec_login 登录操作开始执行 CHAP 操作。 接下来 cmnd_exec_auth_chap 会判断连接的认证状态，若处于认证的第一次握手阶段，此时目标器使用 chap_initiator_auth_create_challenge 函数向启动器发送一个随机字串；若处于第二次握手阶段，目标器将把启动器的用户名提取出来，并使用 account_available 来判断该用户名的权限，然后采用 MD5 加密算法处理用户名、该用户名在本地数据库的密码，以及之前发送的随机字串，得到一个哈希值，再与从启动器发送的 chap 口令进行比较，若一致，则 iSCSI 会话成功建立。

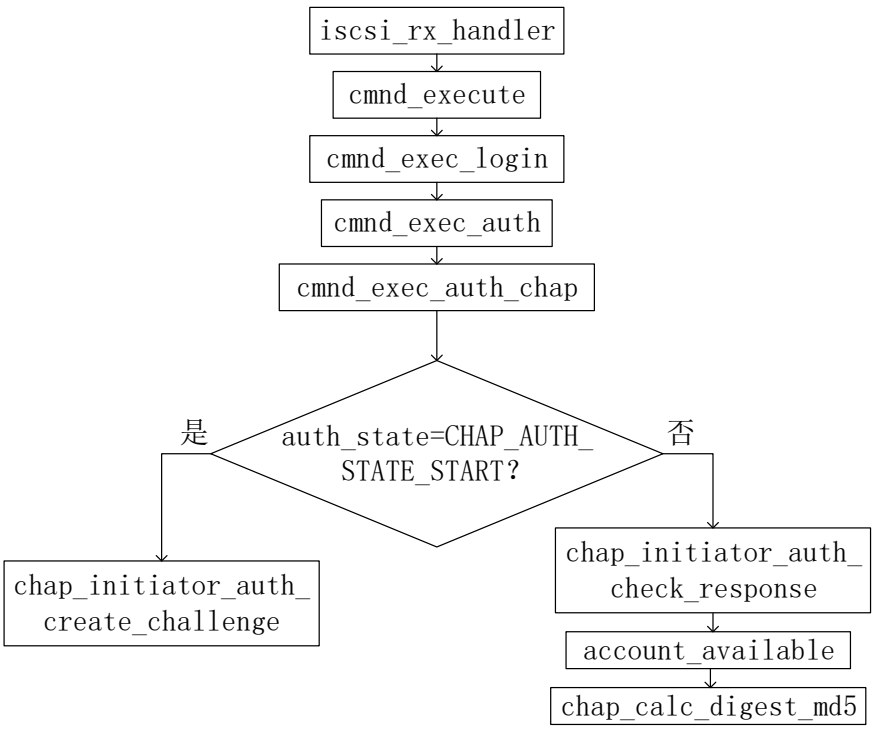

图 3.9 函数工作流程图

目前比较常用的加密算法有数据加密标准算法（Data Encryption Standard，DES）、瑞文斯特-沙米尔-阿德勒曼算法（Rivest–Shamir–Adleman，RSA）、安全哈希算法 1（Secure Hash Algorithm 1，SHA1）和消息摘要算法 5（Message-Digest Algorithm 5，MD5）。

DES 是对称密码算法，加密密钥能够从解密密钥中推算出来，反过来也成立。密钥较短，加密处理简单，加解密速度快，适用于加密大量数据。

RSA 是非对称算法，加密密钥和解密密钥是不一样的，或者说不能由其中一个密钥推导出另一个密钥。密钥尺寸大，加解密速度慢，一般用来加密少量数据。

SHA1 和 MD5 是散列算法，将任意大小的数据映射到一个较小的、固定长度的唯一值。加密性强的散列一定是不可逆的，这就意味着通过散列结果，无法推出任何部分的原始信息。任何输入信息的变化，哪怕仅一位，都将导致散列结果的明显变化。同时，这两个散列算法防冲突的，即找不出具有相同散列结果的两条信息。大量实验数据表明，在加密速率上，MD5 比 SHA1 大约快 33%。

为了尽可能保证用户安全，避免校验过程中传送用户的真实密码，以及出于加密效率的考虑，本模块采用 MD5 加密算法，步骤如图 3.10 所示。

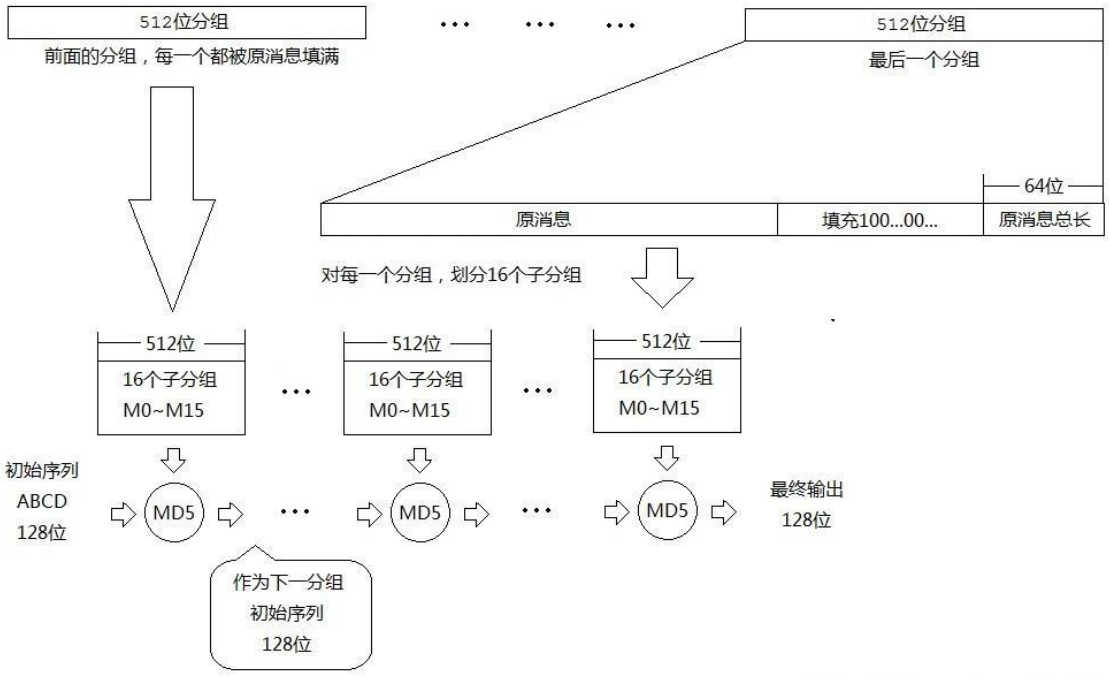

图 3.10 MD5 算法加密步骤图

（1）将要进行哈希处理的消息以 512 位为单位进行分组，其中最后的 64 位用存放原消息长度。不满 512 位要进行数据位填充，填充第一位为 1，其余为 0。每一分组在划分为 16 个 32 位子分组：M[0]-M[15]。

（2）依次赋值四个链接变量 a=0x67452301，b=0xefcdab89，ct=0x98badcfe，d=0x10325476；设置四个非线性算法，F1(X,Y,Z) =(X&Y)|((~X)&Z)，F2(X,Y,Z) =(X&Z)|(Y&(~Z))，F3(X,Y,Z) =X^Y^Z，F4(X,Y,Z)=Y^(X|(~Z))。链接变量和非线性算法用于接下来的循环计算。

（3）依次使用之前设置的非线性算法进行如图 3.11 中的四轮计算（第一轮：F=F1，第二轮：F=F2，第三轮：F=F2，第四轮：F=F4），每一轮包括 16 组线性变换。其中，s 是循环左移位数，常数 ti 是 232*abs(sM(i))的整数部分，i 取值从 1 到 64，单位是弧度。MD5STEP(F,w,x,y,z,M[j],s) = (w += F(x,y,z) + M[j]+ti, w = (w<<s | w>>(32-s)) + x)。

```
MD5STEP(F, a, b, c, d, M[0] + 0xd76aa478, 7);
MD5STEP(F, d, a, b, c, M[1] + 0xe8c7b756, 12);
MD5STEP(F, c, d, a, b, M[2] + 0x242070db, 17);
MD5STEP(F, b, c, d, a, M[3] + 0xc1bdceee, 22);
MD5STEP(F, a, b, c, d, M[4] + 0xf57c0faf, 7);
MD5STEP(F, d, a, b, c, M[5] + 0x4787c62a, 12);
MD5STEP(F, c, d, a, b, M[6] + 0xa8304613, 17);
MD5STEP(F, b, c, d, a, M[7] + 0xfd469501, 22);
MD5STEP(F, a, b, c, d, M[8] + 0x698098d8, 7);
MD5STEP(F, d, a, b, c, M[9] + 0x8b44f7af, 12);
MD5STEP(F, c, d, a, b, M[10] + 0xffff5bb1, 17);
MD5STEP(F, b, c, d, a, M[11] + 0x895cd7be, 22);
MD5STEP(F, a, b, c, d, M[12] + 0x6b901122, 7);
MD5STEP(F, d, a, b, c, M[13] + 0xfd987193, 12);
MD5STEP(F, c, d, a, b, M[14] + 0xa679438e, 17);
MD5STEP(F, b, c, d, a, M[15] + 0x49b40821, 22);
```

图 3.11 16 组线性变化图

（4）处理完所有的 512 位分组，可以得到最终的 a，b，c，d 的值，将这四个值连续输出，就得到了最终的 128 位加密结果。

3.3.4 数据交互模块

在通过登录安全认证之后，存储端和客户端之间已经建立了安全的 iSCSI 会话，接下来，客户端使用数据交互模块对逻辑存储卷进行远程读写。

（1）数据报文格式设计

客户端和存储端通过在 iSCSI 传输 PDU 数据包进行通信。一个 iSCSI PDU 由一到多个 iSCSI 头（一个基本头和多个附加头）和最多一个数据段组成，头部之后可能紧跟一个 CRC（校验和，数据段之后也可能紧跟一个校验和。

基本头（Basic Header Segment，BHS）是第一个 iSCSI 头，所有 PDU 的第一个 iSCSI 头都是基本头。基本头是定长的，占用 48 个字节。基本头后可能紧跟一到多个附加头(Additional Header Segments，AHS)。iSCSI PDU 的格式如表 3.2 所示。

表 3.2 iSCSI PDU 的格式表

基本头（48 字节）
附加头 1（可选）
附加头 2（可选）
……
附加头 n（可选）
头部校验和（可选）
数据段（可选）
数据段校验和（可选）

在 iSCSI PDU 中，每一部分的长度都是 4 的倍数，不够 4 的倍数长的部分也被填充到 4 的倍数，填充的值是 0。

基本头的格式如表 3.3 所示，请求和应答的基本头格式是一样的，区分的依据是其中的 Opcode（操作码）。

表 3.3 基本头的格式表

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	.	I	Opcode						F	Opcode特定域																						
4	TotalAHSLength								DataSegmentLength																							
8 12	LUN或Opcode特定域																															
16	Initiator Task Tag																															
20 44	Opcode特定域																															

注意：在一个字节内，第 0 位是最高有效位，该位为 1 表示 2^7 ，第 7 位是最低有效位，该位为 1 表示在 2^0。在一个 4 字节的双字内，第 0 字节是最高字节，第 3 字节是最低字节；第 0 字节的第 0 位是最高有效位，第 3 字节的第 7 位是最低有效位。

其中：I 是一个立即传送标志，1 表示立即传送。iSCSI 目标器会将它收到的带立即传送标志的命令立刻送到其上层（SCSI 层）。

F 是一个终止标志位，1 表示该 PDU 是最后一个 PDU。

TotalAHSLength 是所有附加头的总长度，单位为长字（4 字节）。无附加头时该域是 0。

DataSegmentLength 是数据段的总长度，单位为字节。无数据段时该域是 0。

LUN 是逻辑单元号。如果不需要指定逻辑单元，该域可以被忽略或做其他用处。

Initiator Task Tag 是启动器赋予的任务标签，用于标识一个任务。如果一个命令需要通过几个请求或应答 PDU 传送，那么这些 PDU 的 Initiator Task Tag 应该相同，而且其 I 标志也应该相同。

Opcode 是操作码，表示 iSCSI PDU 的类型。操作码分成两类，启动器操作码和目标器操作码。启动器操作码出现在由启动器发往目标器的 PDU 中，目标器操作码出现在由目标器发往启动器的 PDU 中。两者不能混用。

Opcode 特定域的格式与 Opcode 密切相关。

表 3.4 和表 3.5 是 iSCSI 的启动器操作码和目标器操作码。

表 3.4 iSCSI 启动器操作码

操作码	意义
0x00	Nop_Out
0x01	SCSI 命令
0x02	SCSI 任务管理请求
0x03	登录请求
0x04	数据请求
0x05	写操作
0x06	注销请求
0x10	SNACK 请求
0x1c-0x1e	自定义操作码

表 3.5 iSCSI 的目标器操作码

操作码	意义
0x20	Nop_In
0x21	SCSI 命令应答（包括 SCSI 状态等）
0x22	SCSI 任务管理操作应答
0x23	登录响应
0x24	数据应答
0x05	读操作
0x26	注销应答
0x31	准备发送请求

0x32	异步消息，目标器用于通知特殊情况
0x3c-0x3e	自定义操作码
0x3f	拒绝

所有的附加头的格式都相同，如表 3.6。

表 3.6 附加头格式表

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	AHSLength																.	.	AHSType						保留							
4x	AHS特定																															

附加头只有两种：

如果 ASHType=1，则其 ASH 特定部分是一个扩展的命令描述块。

如果 ASHType=2，则其 ASH 特定部分是一个长整数，表示期望读数据的长度。

PDU 中的长度域是很重要的，因为它界定了 iSCSI PDU 的开始和终止位置（TCP 报文中没有边界信息）。但是，iSCSI 头可能丢失或延迟到达，因而目标器不得不准备临时的缓冲区来暂存失序的报文。为了减少临时缓冲区的数量，iSCSI PDU 的长度应该较小。最大 iSCSI PDU 长度是登录阶段协商的参数之一。

（2）工作流程

无论客户端读还是写操作，目标存储器处理 I/O 流程都是相似的，只是在具体的 SCSI 报文处理过程上稍有差异，交互流程如图 3.12。

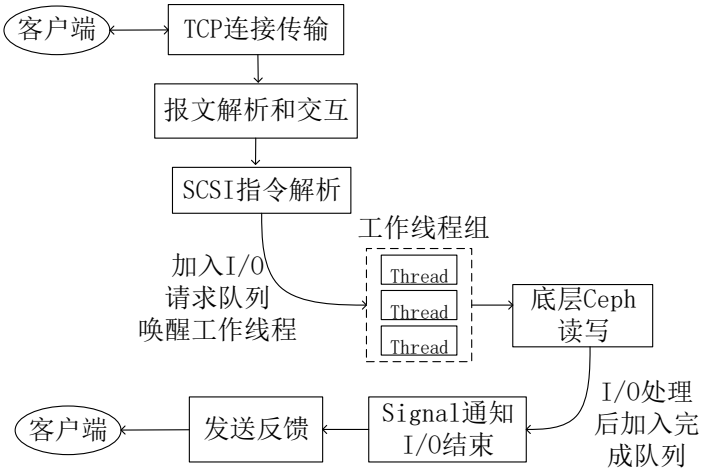

图 3.12 数据交互流程图

1） 存储端通过 IP 网络接收启动器的 TCP 消息，识别 SCSI 命令，确认 SCSI 命令操作类型，与客户端进行报文交互。

2） 下发 I/O 操作，将收到的 I/O 请求加入队列，并唤醒 I/O 工作线程组中的线程。

3） 工作线程回调响应的 Ceph 分布式存储系统数据处理接口执行 I/O 操作。

4） I/O 处理结束返回后，通知存储卷封装 I/O 完成消息，反馈给客户端 I/O 完成。

具体设计如图 3.13 中的函数工作流程：首先要对会话上传递的 iSCSI pdu 进行解析，解析操作由 iscsi_rx_handler 完成，解析得到的命令和数据按照顺序由 iscsi_task_queue 放置到任务处理队列里，当解析得到 SCSI 请求队列后，向下层传递，进而调用队列处理函数 scsi_request_fn，该函数调用 lun_peek_request 逐个取出请求，并进行处理，每次调用就从可处理的请求队列中获得一个请求。然后调用 scsi_prep_fn 进行一些请求处理前的准备工作，再使用 sd_init_command 进行 SCSI 命令数据结构的初始化，之后 sd_setup_read_write_cmd 完成 SCSI 命令的构建。接下来，当接收到的 SCSI 命令全部存放到消息队列后，scsi_dev_queue_ready 调用 lun_start_request 开始处理 SCSI 命令队列，之

后，scsi_dispatch_cmd 把 SCSI 命令分发到底层 Ceph 存储集群，bs_rbd_request 判断读写命令类型，开始数据 I/O。

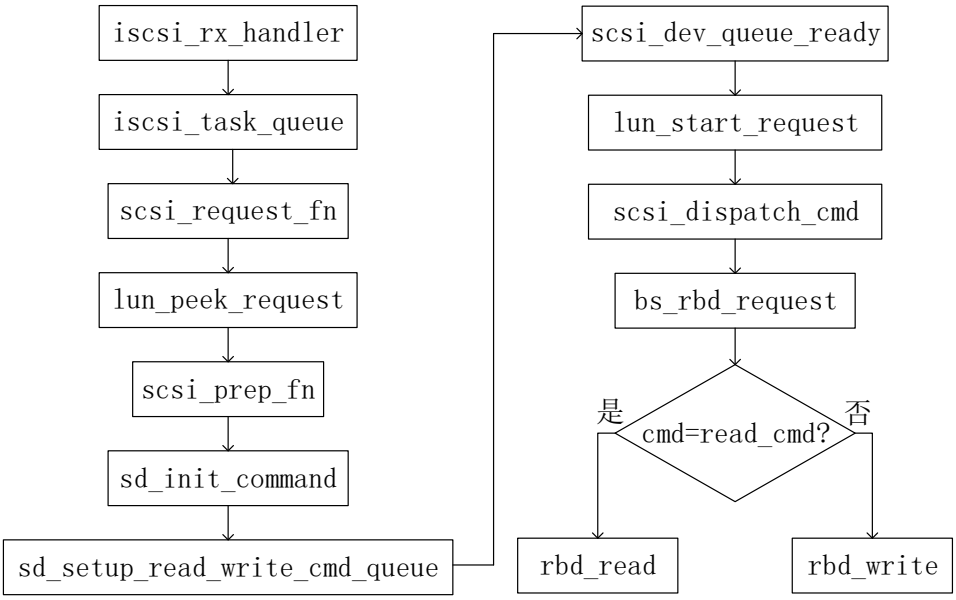

图 3.13 函数工作流程图

当写数据请求被下移到 Ceph 分布式集群时，首先将数据转换为对象，再将对象和 PG 数一起经过散列来生成其在 Ceph 存储对象池中最终存放的 PG，然后把前面计算好的 PG 经过 CRUSH 查找来确定存储数据所需特定副本数的 OSD 列表，PG 与 OSD 列表的映射关系由 MON 保存，获取 OSD 位置的具体过程如图 3.14 所示。当 Ceph 集群收到读请求时，只需向 MON 获取该数据对象的主副本 OSD 位置即可。

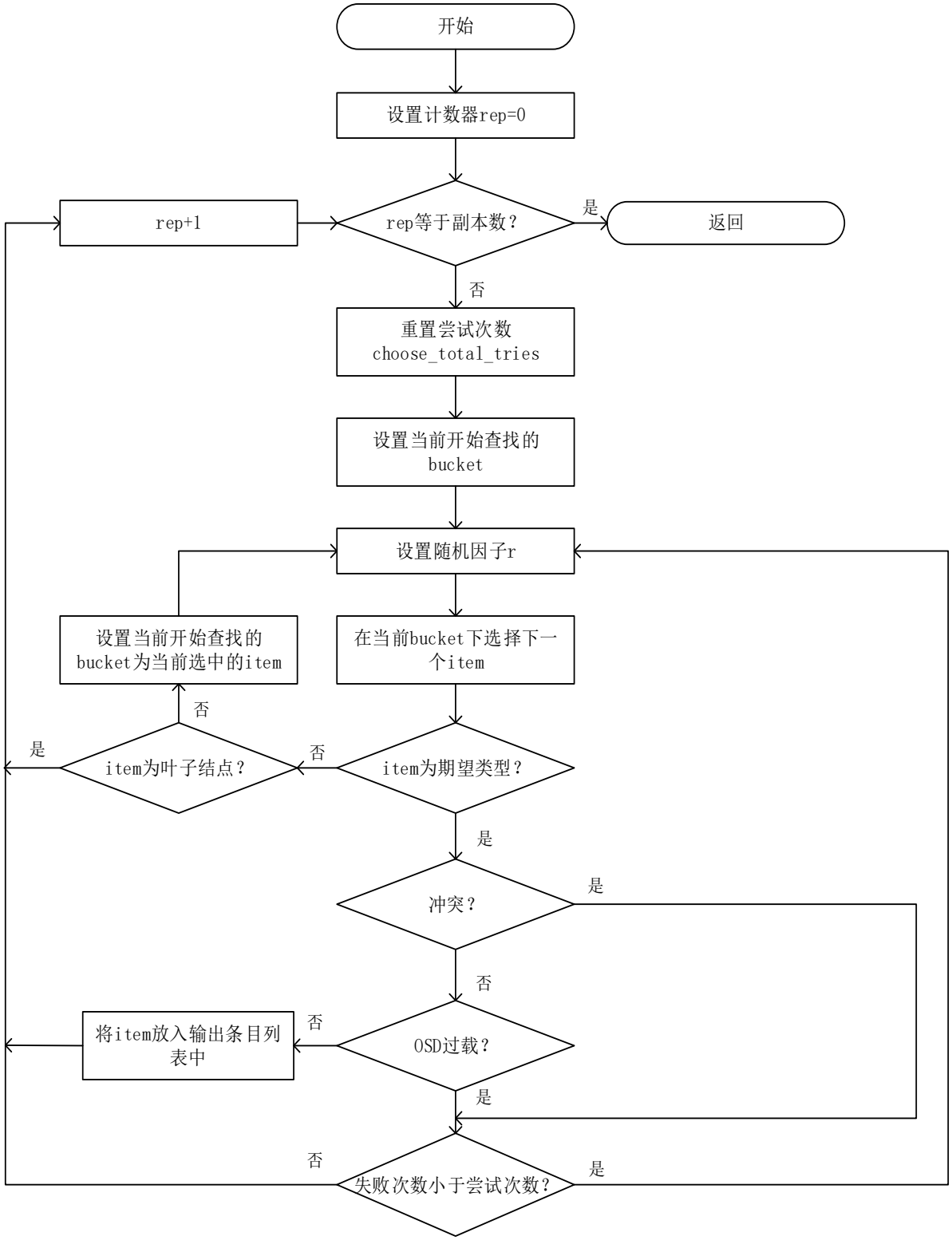

图 3.14 CRUSH 查找 OSD 过程图

下面对几个关键步骤进行说明：

1）在当前 bucket 下选择条目

当构建 cluster map 时，对每种类型的 bucket 采用 straw 算法，从对应的 bucket 中选择一个条目。因此从 bucket 选择条目的过程实际上就是执行设定的选择算法。这个选择算法的执行结果取决于两个因素：一是以输入对象的特征标识符，即 pg_id，二是随机因子 r（r 实际上是作为哈希函数的种子）。pg_id 是固定的，r 默认是条目编号，也可由待选择的副本编号和当前的尝试次数共同决定。

为了防止陷入死循环，需要对选择每个副本过程中的尝试次数进行限制，这个限制称为全局尝试次数（choose_total_tries）。

straw 是 Ceph 数据分布机制 CRUSH 采用的基本选择算法，straw 的关键在于如何计算签长。签长取决于由所有元素权重决定的 scaling factor（比例因子）。

原 straw 算法实现中，将所有元素按其权重进行逆序排列后逐个计算每个元素的签长，,计算过程中不断累积当前元素与前后元素的权重差值，以此作为计算下一个元素的基准，因此原有实现中 straw 算法的最终选择结果不但取决于每个元素的自身权重，而且也和集合当中所有其他元素的权重强相关，改变某一个元素的权重，会影响到其他元素的权重，从而导致每次有元素加入为前集合或者从当前集合中删除时，都会引起不相关的数据迁移。

这种无谓的数据迁移使 straw 算法需要改进，我们可以通过使用独立权重对计算签长的过程进行调整，即总是倾向于让权重大的元素获得更大的签长，让权重小的元素获得更小的签长。因此，此时 straw 算法执行结果取决于三个因素： 元素集合 pg_id、元素编号 bucket_id 和元素权重，其中元素编号起的是随机种子的作用，所以针对固定的 pg_id，straw 算法实际上只受元素本身权重的影响。如果每个元素的签长只和自身权重相关，那么对于添加元素和删除元素的处理都是最优的，以添加元素为例进行论证：

① 假定当前集合中一共包含 n 个元素：（e_1，e_2，…，e_n）；

② 向集合中添加新元素 e_{n+1}；

③ 针对输入值 pg_id，加入 e_{n+1} 之前，分别计算每个元素签长并假定其中最大值为 d_{max} ：（d_1，d_2，…，d_n）；

④ 因为新元素的签长只和自身编号及权重相关，所以可以独立计算签长（其他元素的签长不受 e_{n+1} 加入的影响），假定为 d_{n+1}；

⑤ 因为 straw 算法总是选择最大的签长作为最终结果，所以如果 $d_{n+l}>d_{max}$，那么 pg 将被重新映射至新元素 e_{n+1}；反之，对 x 的已有映射结果无任何影响。

可见，添加一个元素，straw 算法会随机地将一些原有元素中的数据重新映射到新加入的元素当中。同理，删除一个元素，straw 算法会将该元素中全部数据随机地重新映射至其他元素之中。因此无论添加或者删除元素，都不会导致数据在除被添加或者删除之外的两个元素（即不相关的元素）之间进行迁移。

改进后的选择方式如算法 3.1：

输入：对象集合 pg_id

输出：最大签长的元素

算法：

```
1    初始化元素集合，从 pg_id 里选取，记为 C={item_id1,item_id2,item_id3…};
2    for each item item_idi∈ C
3        strawi ← ln(hash (pg_id,item_idi) /65536)/get_weight(item_idi);   //计算元素签长
4    end for
5    return 最大签长的元素
```

算法 3.1 改进后的基本选择算法

在算法 3.1 中，首先从指定 pg_id 里面初始化元素集合 C（第 1 行），然后将 pg_id 和 item_id 作为哈希输入值，结果落在[0,65535]之间，除以 65536 之后取自然对数，再除以自身权重(weight)后计算得到集合 C 里面每一个元素的签长（第 2-4 行），最后返回最大签长的元素（第 5 行）。

2）冲突

冲突指的是选中的条目已经存在于输出条目之中。为了降低冲突概率，可以使用当前的重试次数对递归调用时的随机因子 r 再次进行调整，这样产生递归调用时，其初始随机因子 r 将取决于待选择的副本编号和尝试次数。

3）OSD 过载

虽然 CRUSH 的能够理论上保证数据在所有磁盘之间均匀分布，但是实际上由于集群规模较小，集群整体容量有限，导致 OSD 总数有限，亦即 CRUSH 输入的样本容量不够，会导致有的 OSD 被重复计算得到。

3.4 本章小结

本章对虚拟化存储系统进行了分析和设计，包括需求分析、设计目标、系统整体工作流程和各个子模块的设计。首先从功能、性能和安全性方面进行了需求分析，明确了用户需求和系统价值，其次根据用户需求提出了设计目标，然后介绍了系统工作流程。最后，阐述了系统各个子模块的功能和具体流程设计。

第四章 存储虚拟化系统的实现

在上一章中，我们详细介绍了存储虚拟化系统的工作流程和三个子模块的具体功能，本章将阐述系统开发环境、整体架构和各个子模块内部实现细节。

4.1 系统开发环境

本论文所设计的存储虚拟化系统开发环境如表 4.1 所示。

表 4.1 系统开发环境表

开发环境	信息
操作系统	Ubuntu 14.04.3 LTS
语言	C/C++
编译器	gcc 4.8.0
开发库	STL、Linux 系统调用
第三方库	Boost 库、Librados 库和 Librbd 库

4.2 系统架构

系统整体层次如图 4.1，最底层是存储磁盘，每一个存储磁盘对应 Ceph 分布式存储系统的一个对象存储设备，Ceph 将多个对象存储设备采用精简配置，通过自身的底层机制实现可调整大小和存储数据条带化的块存储配置。一个块存储设备的真实存储空间可以分布在不同服务器上的不同磁盘上。采用 iSCSI 技术将 Ceph 块存储设备虚拟化为客户端主机能够连接和共享的一个逻辑概念上的存储卷-逻辑单元号。一组逻辑存储卷可以组成一个存储池，即 iSCSI 会话中的目标器。通过传输 iSCSI 协议数据单元，客户端主机可以在存储卷上进行数据的读写操作，例如数据的写入、复制、迁移和删除等。一个主机可以访问不同的目标器存储资源，一个目标器也可以被不同地域的主机访问。

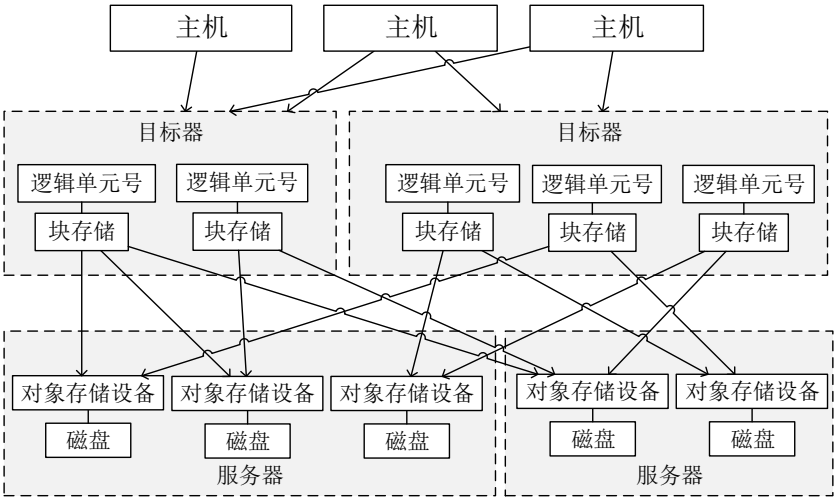

图 4.1 虚拟化存储系统层次图

该存储系统对现有的存储资源进行统一管理，将存储设备虚拟成资源池，为用户提供巨大的逻辑地址空间。Ceph 分布式存储系统本身具有包括统一存储能力、可扩展性、可靠性和自动化维护的特性，在此基础上使用 iSCSI 虚拟化技术来实现远端用户的访问。本系统通过对所有的物理存储资

源进行分布式和虚拟化，建立一个统一的存储池。系统管理员可以根据用户的应用需求来调整存储资源，从而达到共享物理存储资源的目的。该存储系统的三个核心模块所负责的功能和它们之间的相互作用如图 4.2 所示，系统管理员使用资源管理模块为用户提供存储资源，用户在远程的客户端主机通过登录认证模块与存储端之间建立安全会话，再使用数据交互模块对存储端进行读写访问。

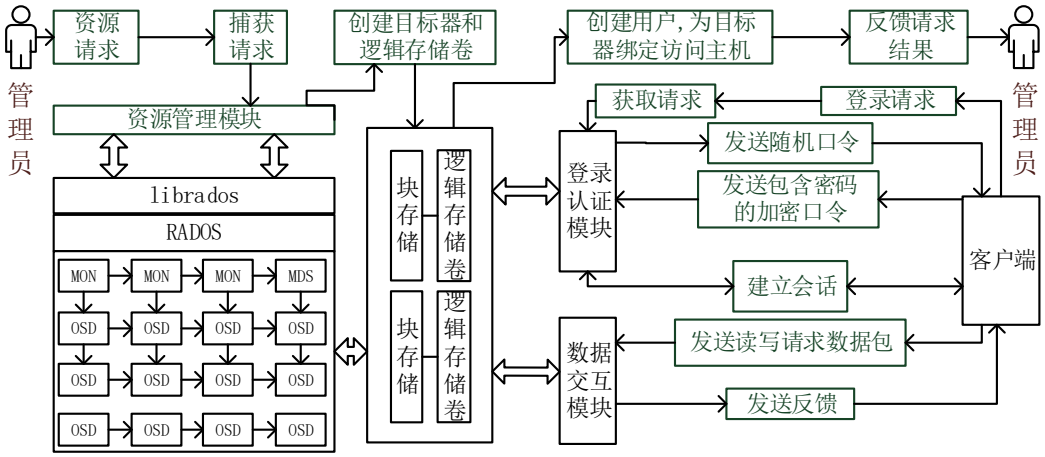

图 4.2 系统逻辑架构图

4.3 存储资源管理模块

该模块用于监视和修改与 iSCSI 目标器相关的三种对象，包括存储池、存储卷和用户，主要功能包括三个方面：（1）创建、删除和显示存储池；（2）在特定的存储池中创建和删除逻辑存储卷；（3）定义存储池基于主机的访问控制列表，赋予和删除用户对存储资源的读写权限。主要数据结构如下：

（1）目标器结构

target 数据结构包括存储池、存储卷和用户的各种参数。

```
struct target {
    char *name; //存储池名字
    int tid; //存储池 ID
    int lid; //逻辑存储卷 ID
    enum scsi_target_state {
        SCSI_TARGET_OFFLINE = 1, //离线
        SCSI_TARGET_READY, //正常运行
    } target_state; //存储池状态
    struct list_head target_siblings; //存储池链表
    struct list_head rbd_list; //Ceph 存储设备链表
    struct rbd_op *bst; //Ceph 存储设备操作类型
    struct list_head *acl_list; //用户访问控制权限
    struct account account; //用户
    struct list_head lld_siblings; //存储池内的逻辑存储卷链表
};
```

（2）命令请求和反馈结构

adm_req 表示请求命令结构，包括与上述存储池、存储卷和用户等各种操作模式下的命令类型

和参数信息，adm_req 表示命令反馈，包括命令执行结果和命令长度两个信息。

```
struct adm_req {
    enum adm_mode {
        MODE_TARGET, //存储池
        MODE_LLD, //存储卷
        MODE_ACCOUNT, //用户
    } admin_mode; //命令模式
    enum admin_op enum adm_op {
        OP_NEW, //新增
        OP_DELETE, //删除
        OP_SHOW, //列举
        OP_BIND, //绑定
        OP_UNBIND, //解除绑定
        OP_UPDATE, //更新
        OP_STATS, //会话统计信息
        } op; //命令操作类型
    char lld[LLD_NAME_LEN]; //存储卷名字数组
    uint32_t len; //请求长度
    int32_t tid; //目标器 ID
    uint64_t sid;// 用户 ID
    uint64_t lun;// 存储卷号码
    uint32_t cid;// 连接标识符
    uint32_t host_no; //主机号
    uint32_t ac_dir; //用户列表
    uint32_t force; //强制删除标志
};
struct adm_rsp {
    uint32_t err;//命令执行结果
    uint32_t len;//命令长度
};
```

（3）命令操作集合

adm_operations 数据结构里封装了所有的操作函数指针，可根据不同的操作模式和命令类型来选择不同的处理函数。

```
struct adm_operations {
    adm_err *target_create(int tid, char *args); //创建存储池
    adm_err target_destroy(int lld_no, int tid, int force); //删除存储池
    enum scsi_target_state get_target_state(int tid); //获取存储池状态
    adm_err target_update(int tid, uint64_t dev_id, char *params); //向存储池添加或删除存储卷
    adm_err acl_add(int tid, char *address); //向特定 ip 地址的主机开放存储池
    adm_err acl_del(int tid, char *address); //对主机收回存储池的访问权限
    adm_err iqn_acl_add(int tid, char *name); //添加存储池用户
    adm_err iqn_acl_del(int tid, char *name); //删除存储池用户
    adm_err target_show_all struct target *target); //获取某个存储池内所有存储卷
    adm_err account_add(char *user, char *password); //创建用户名和密码
    adm_err account_del(char *user); //删除用户
```

```
        adm_err account_ctl(int tid, int type, char *user); //赋予用户访问权限
        adm_err account_show(struct list_head *acl_list); //获取所有用户
};
```

4.4 登录认证模块

在存储资源管理模块里，我们对存储资源进行统一管理，使用户可以通过特定 IP 地址的主机来访问存储池。这时候，客户端主机是 iSCSI 启动器，而存储池是 iSCSI 目标器。要实现启动器对目标器，我们需要建立 iSCSI 会话。一个 iSCSI 会话是由多个 TCP 连接组成的。

登录认证模块的任务是完成 iSCSI 会话的登录阶段。在登录阶段，会话双方相互认证并协商会话参数。为了保护 TCP 连接，在开始登录请求之时可以先在 iSCSI 启动器和目标器之间建立安全关联。在登录阶段，TCP 连接上只能传送登录请求和应答。会话中的每个 TCP 连接都要经过登录阶段。主要数据结构如下：

（1）PDU 结构

iSCSI 启动器和目标器之间传输的信息科分割为消息块，这些消息就是我们在第二章提到的 iSCSI PDU。iSCSI PDU 数据结构由一到多个 iSCSI 头（一个基本头和多个附加头）和最多一个数据段组成。

```
struct iscsi_pdu {
        struct iscsi_hdr bhs; //基本头
        void *ahs; //附加头
        unsigned int ahssize; //附加头长度
        void *data; //数据段
        unsigned int datasize; //数据段长度
};
```

（2）TCP 连接结构

TCP 连接的内部结构，该结构保存使用连接的目标器与启动器的详细信息、连接中进行传输的各项数据操作及相关控制参数等。

```
struct iscsi_connection{
    struct iscsi_session *session; //TCP 连接使用的 iSCSI 会话
    struct iscsi_target *target; //指向连接的目标器端
    uint32_t stat_sn; //状态序列号
    uint32_t exp_stat_sn; //下一个希望得到的序号
    struct list_head *cmd_list; //连接中传送的 PDU 链表
    struct list_head *write_list; //将在连接中传送的数据 PDU 链表
    struct 1ist_head written_list; //正在被多条 TCP 连接传送的 PDU 链表
    struct timer_list *rsp_timer; //响应时间
    struct iscsi_cmnd *write_cmnd; //iSCSI 写命令
    struct iscsi_cmnd *read_cmnd; //iSCSI 读命令
    struct socket *sock; //指向目标器打开的 socket
    struct list_head *conn_list_entry; //TCP 连接链表入口
    uint16_t cid; //连接标识符
    int auth_state; //认证状态
    union {
```

```
            struct {
                int digest_alg; //加密算法种类
                int id; //chap 口令 ID
                int challenge_size; //挑战数据包长度
                unsigned char *challenge; //挑战数据包
            } chap;
        } auth;
    };
```

关键成员说明：

stat_sn 是每个 TCP 连接都有的唯一状态序列号，这些唯—的编号是在相应的 TCP 连接级别上建立的。在出现暂时或永久性通信错误时，利用状态序号可以检测丢失的状态信息并恢复之。

auth_state 表示认证状态，目标器根据认证状态来确定应该采取的认证步骤。

cid 全称是 Connection ID，是 TCP 连接的唯一标识符，由启动器生成。

chap 全称是 challenge Handshake Authentication Protocol，表示挑战握手认证协议数据包，是进行登录安全认证的关键成员。

（3）MD5 结构

安全认证是通过 MD5 加密算法实现的，我们把必要的信息封装在下面的 MD5Context 数据结构里。

```
struct MD5Context {
    unit32_t buf[4]; //算法初始参数和输出结果
    unit32_t M[16]; //消息分组
    unit32_t bytes[2]; //原消息长度
};
```

关键成员说明：

buf[4]存放 MD5 算法的四个链接变量，分别是 0x67452301，0xefcdab89，0x98badcfe 和 0x10325476，经过一系列循环计算之后，buf[4]最终存放的是将字串加密之后的哈希值。

M[16]是消息分组，作为 MD5 算法的输入值。

bytes[2]存放原消息长度，作为消息分组的一部分，在算法初始阶段赋值给 M[14]和 M[15]。

4.5 数据交互模块

在第一个 TCP 连接上通过登录安全认证之后，登录阶段结束，此时，启动器和目标器之间的会话正式建立并进入全特征阶段。一旦会话处于全特征阶段，iSCSI 启动器就可以在该会话上向 iSCSI 目标器发送命令和数据。命令和数据被封装在 iSCSI PDU 中。数据的读写操作由数据交互模块完成。主要数据结构如下：

（1）会话结构

会话结构保存会话两端的启动器和目标器、会话中建立的 TCP 连接等信息。

```
struct iscsi_session {
    int refcount; //引用计数
    struct list_head slist; //会话链表
    char *initiator; //启动器指针
    struct target *target; //目标器指针
    uint8_t isid[6]; //启动器标识符
```

```
    uint16_t tsih; //目标器句柄
    struct list_head conn_list; //连接的 TCP 链表
    int conn_cnt; //连接数目
    struct list_head cmd_list; //任务链表
    struct list_head pending_cmd_list; //待执行任务链表
    uint32_t exp_cmd_sn; //下一个期望的任务序号
    uint32_t max_queue_cmd; //会话中命令队列的最大长度
    struct param session_param[ISCSI_PARAM_MAX]; //会话参数
};
```

关键成员说明：

isid 全称是 Initiator Session ID，表示会话标识符的启动器部分，由 iSCSI 启动器指定，它由 6 个字节组成，并具有一定的结构。isid 是可配置的，而且必须保持不变。

tsih 全称是 Target Session Identifying Handle，表示目标器的会话标识句柄，是目标器为会话指定的一个标识符。在创建新会话的登录请求中，由于会话还未建立，所以 tsih 是 0；在第一个 TCP 连接成功登录之后，目标器生成 tsih 并通知启动器。在加入新连接的登录请求中，tsih 代表连接要加入的会话。

（2）SCSI 命令结构

```
struct scsi_cmd {
    struct target *c_target; //目标器
    struct list_head c_hlist; //所属的命令链表
    struct list_head qlist; //命令队列
    uint64_t dev_id; ///存储设备 ID
    struct scsi_lu *dev; //逻辑存储卷
    unsigned long state; //状态
    enum data_direction data_dir; //数据读写模式
    struct scsi_data_buffer in_sdb; //输入缓冲区
    struct scsi_data_buffer out_sdb; //输出缓冲区
    uint8_t *scb; //命令描述块
    int scb_len; //命令描述块长度
    uint8_t lun[8]; //逻辑存储卷数组
    int attribute; //任务属性
    uint64_t tag; //命令标签
    int result; //执行结果
    unsigned char sense_buffer[SCSI_SENSE_BUFFERSIZE]; //感测数据缓冲区
    int sense_len; //感测数据长度
    struct list_head bs_list; //存储设备链表
};
```

关键成员说明：

state 表示命令完成的状态。

scb 是命令描述块，表示请求存储卷要执行的命令。

out_sdb 是输出缓冲区，输出缓冲区中 包含执行命令所需要的数据与参数，这些数据和参数要传送到存储卷。在命令的执行过程中，输出缓冲区的内容保持不变。

in_sdb 是输入缓冲区，.用于保存命令执行完成后从逻辑单元中返回的信息。

attribute 是任务属性，用来使客户端赋予每个任务一个属性，比如排序规则。

sense_bufffer 是另一个缓冲区，存放从存储卷单元返回的感测数据（sense data）。结合命令状态

可根据该数据确定出现异常的原因。

4.6 本章小结

本章自底向上介绍了系统整体架构，详细描述了三个子模块——资源管理模块、登录认证模块和数据交互模块的实现细节。通过各个子模块的实现，系统为用户提供了可远程访问的存储资源，保证了用户的安全登录和高效读写操作。

第五章 系统测试

本章的目标是验证本论文所实现的存储虚拟化系统是否已经达到了需求分析阶段的要求。从功能需求、性能需求和安全性需求三个方面来检测存储虚拟化系统的实现和运行情况。通过系统测试，确保存储系统达到需求分析阶段对系统提出的要求。最后对测试结果进行分析总结。

5.1 测试环境

在本测试环境中，包括一个客户端和三台服务器。本论文设计存储系统的测试环境包括硬件环境和软件环境。其中，硬件环境如表 5.1 中所示，包括 CPU、硬盘、内存、网卡和交换机等相关参数；软件环境是 Ubuntu 14.04.3 LTS 操作系统。

表 5.1 硬件参数表

硬件	型号
CPU	Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz
内存	Micron 18ASF1G72PZ-2G3B1 DDR4 SD DIMM 8192MB
硬盘	ATA HGST HUS726020AL T907
网卡	Intel(R) 82579V Gigabit Ethernet NIC
交换机	H3C S5820V2-52Q-LSWZ152Q-L3

5.2 测试方案

在设计存储虚拟化系统测试方案的过程中，结合了需求分析阶段的系统性能要求，一方面，要确保测试方案可以对系统功能进行有效地验证，主要是根据需求功能结构图和需求功能说明书来设计具体的测试用例。另一方面，确保系统的性能要达到需求分析当中的目标。

（1）功能测试方案

- 按需求文档和设计文档编写功能测试用例。
- 查看系统功能模块是否实现。
- 查看功能项是否符合设计规范的要求。
- 按预定动作输入，是否产生预期输出结果。
- 输出结果分析，拟定修改方案。

（2）性能测试方案

吞吐量指的是单位时间内的数据传输量，是存储设备制造商给用户提供的最常见的性能指标，单位是 KB/s。我们将使用 IOzone 工具来测试存储系统在各种访问模式下的吞吐量，改变块记录大小、文件大小和读写模式等参数，通过得到的测试数据来综合分析影响系统性能的因素。

IOzone 是一个可用于分析多个不同平台上的存储系统吞吐量性能的测试工具，包括 Linux，HP-UX，Solaris 等等。IOzone 使用本地文件系统 I/O 来生成主要读写负载，可对测试系统进行大范围的文件 I/O 请求，在测试过程中文件和块记录逐渐由小变大。同时，该工具可执行许多不同的文件访问模式，允许管理员修改系统测试参数来改变读写模式。

lOzone 常用参数如下：

-a 全而测试，块大小会自动叠加。

-i N 用来选择不同的文件访问模式，比较常用的是 0,1,2，可以指定成-i 0，–i 1，-i 2，分别表示 write/rewrite（顺序写/再写），read/reread（顺序读/再读），random read/write（随机读/写）。

-r 块记录大小，指定一次读写的块大小。
-s 文件大小，指定测试文件的大小。
-f 文件名，指定测试文件的名字，完成后会自动删除（这个文件的位置是要测试的那个硬盘）。
-F 多个文件名，指定多线程下测试的文件名。
-g 指定测试文件的最大范围。
-y 指定测试文件的最小范围。
-R 产生 Excel 到标准输出（需要指定-a 的测试才能输出）。
-b 指定输出到指定文件上．比如-Rb result.xls（同样需要指定-a 的测试才能输出）。
测试步骤如下：

- 建立存储池。
- 为存储池绑定客户端主机 IP 地址。
- 创建用户名。
- 用户在客户端使用 IOzone 对存储池内的逻辑存储卷生成读写负载。

（3）安全性测试方案

- 用户登录验证，检验合法用户和非法用户的登录结果及处理办法。
- 检验用户操作权限是否符合设定要求。
- 人为使某块物理资源断电后，数据是否会迅速恢复。

5.3 测试结果及分析

5.3.1 功能测试

功能测试主要用来验证系统的功能是否实现以及是否能够流畅的运行，并且保证系统不存在功能性 bug。在测试之前，首先要部署分布式存储集群，集群状况如图 5.1 所示。本次功能测试主要对创建目标器（存储池）资源、删除目标器资源、用户注册和删除等模块进行测试，部分测试用例如表 5.2

图 5.1 分布式存储集群状态图

表 5.2 功能测试用例表

用例名称	用例编号	模块	测试类型	优先级	概述	步骤	输入	期望输出	实际输出
目标器创建	T1-01	存储资源管理	功能	高	验证目标器是否能成功创建	步骤1	对现有的物理资源构建存储集群	得到一个Ceph分布式存储集群	通过
						步骤2	输入目标器名称，创建目标器	目标器成功建立，存储系统自动为目标器分配ID	
						步骤3	根据目标器名称或ID为目标器添加逻辑存储卷	得到一个包含多个存储卷的目标器	
目标器资源删除	T1-02	存储资源管理	功能	高	验证目标器能否安全删除	步骤1	根据目标器名称或ID删除目标器内逻辑存储卷	目标器逻辑资源为空	通过
						步骤2	删除目标器	目标器删除成功	
用户注册	T1-03	存储资源管理	功能	高	验证用户是否能够成功创建	步骤1	输入用户名和密码	用户名和密码有效	通过
						步骤2	设置访问权限	用户注册成功	
用户删除	T1-04	存储资源管理	功能	高	验证用户能否安全删除	步骤1	根据目标器名称或ID删除用户列表里的用户	用户删除成功	通过

5.3.2 性能测试

在 linux 客户端执行./iozone –a –n 16m –g 4096 m –i 0 /mnt/test_iscsi_device ./iozone.xls，首先在 write/rewrite 模式下对存储系统进行测试，实验数据分别如图 5.2 和图 5.3。

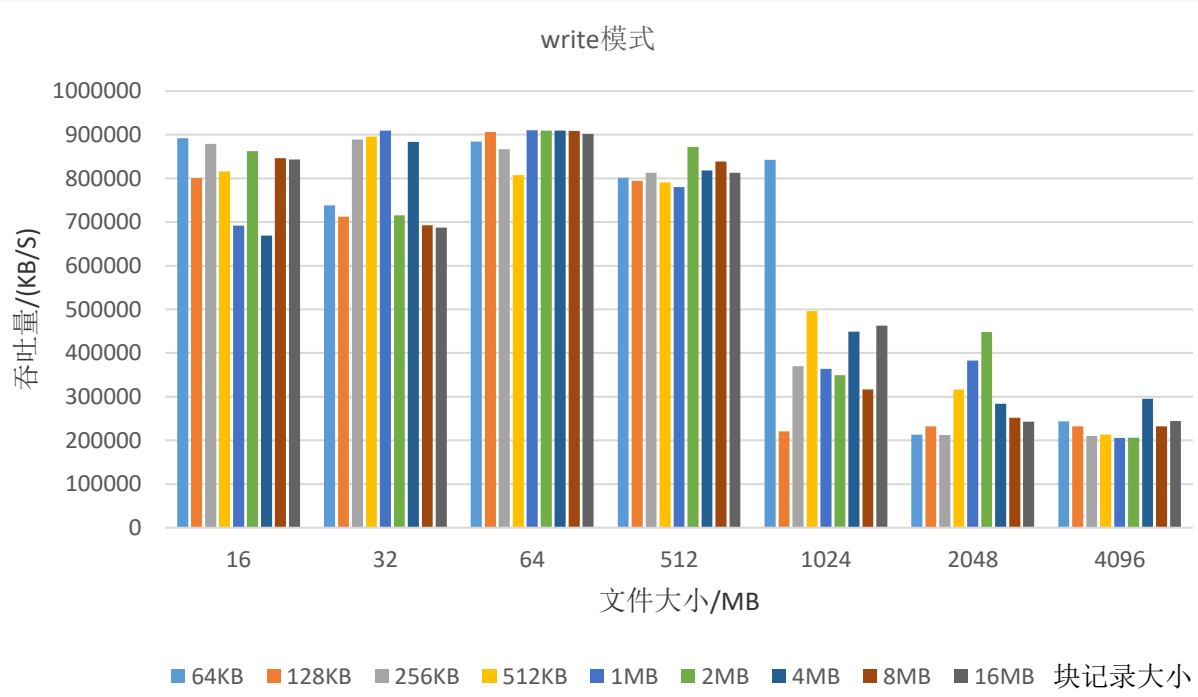

图 5.2 write 模式测试数据图

由图 5.2 可知，在第一次写入文件的时候，当文件比较小的时候，随着文件所占的空间增大，吞吐量变化不太明显。但是当文件大于 512MB 时，吞吐量突然大幅度下降，普遍降低了 37%-72%，存储块的性能衰退比较明显。这是因为文件越大，远端存储系统底层 Ceph 分布式集群需要计算的负载就越大，所以写入效率比较低。此时，吞吐量受块记录影响比较小。

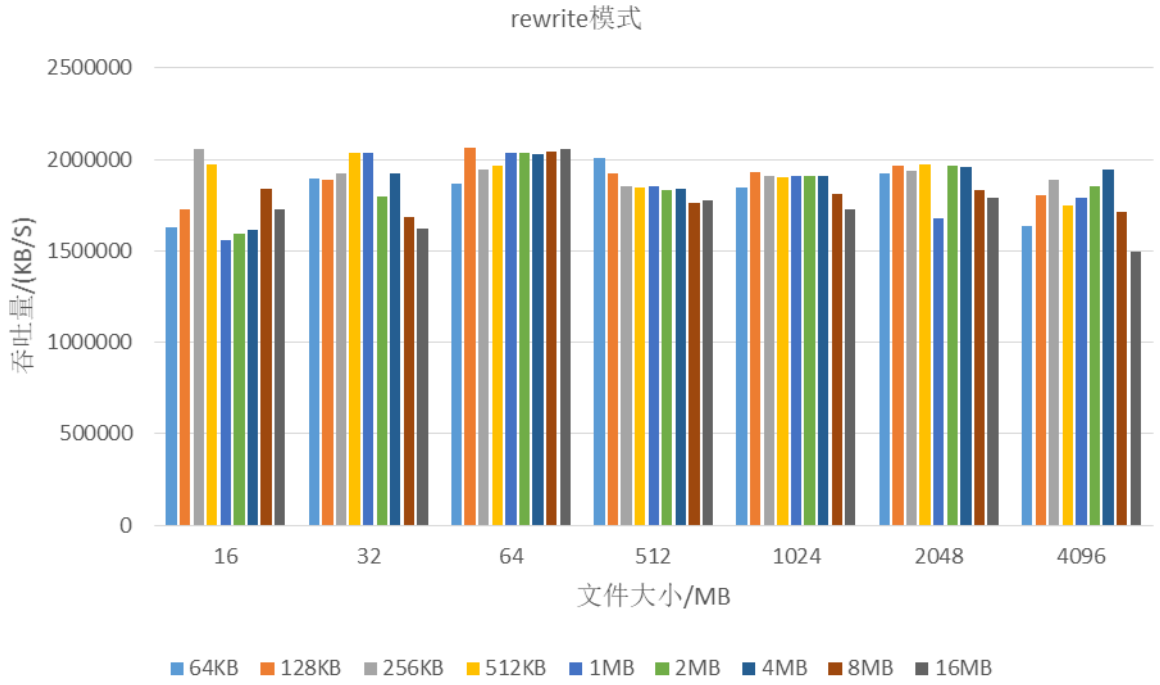

图 5.3 rewrite 模式测试数据图

由图5.3可知，rewrite即重写模式下的吞吐量大概比写模式大50%。当写入已经存在的文件时，因为元数据已经存在，所需的工作负载较少。重写性能通常高于写入新文件的性能。此时，文件和块记录大小对吞吐量的影响比较小。

执行./iozone –a –n 16m –g 4096 m –i 1 /mnt/test_iscsi_device ./iozone.xls，在 read/reread 模式下对存储系统进行测试，read 和 reread 实验数据分别如图 5.4 和图 5.5。

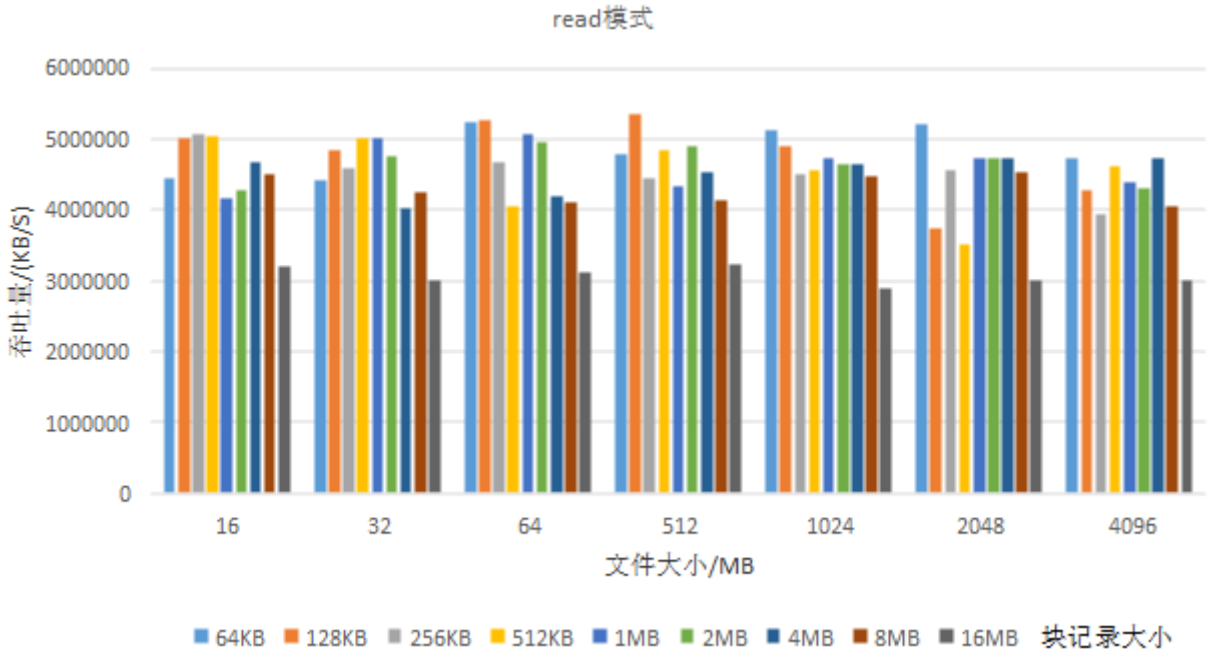

图 5.4 read 模式测试数据图

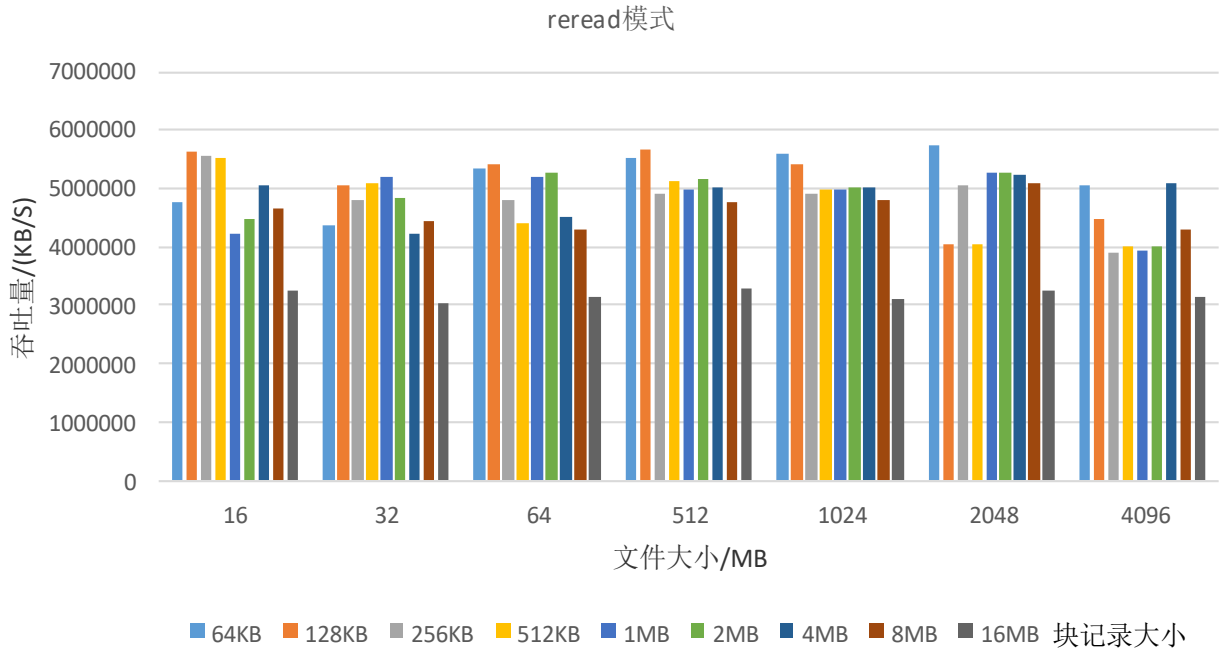

图 5.5 reread 模式测试数据图

由图 5.4 和图 5.5 数据，比较两种读取模式可知，重新读取（reread）模式比读取（read）吞吐量较大，因为操作系统通常维护最近读取的文件的数据缓存，此缓存可用于提高性能。同时，文件和块记录大小对吞吐量影响比较小，但是当块记录大到 16MB 时，吞吐量下降了 30%-50%。

执行./iozone –a –n 16m –g 4096 m –i 2 /mnt/test_iscsi_device ./iozone.xls，测试 random write/read 模式下存储系统的吞吐量，实验数据如图 5.6 和图 5.7。

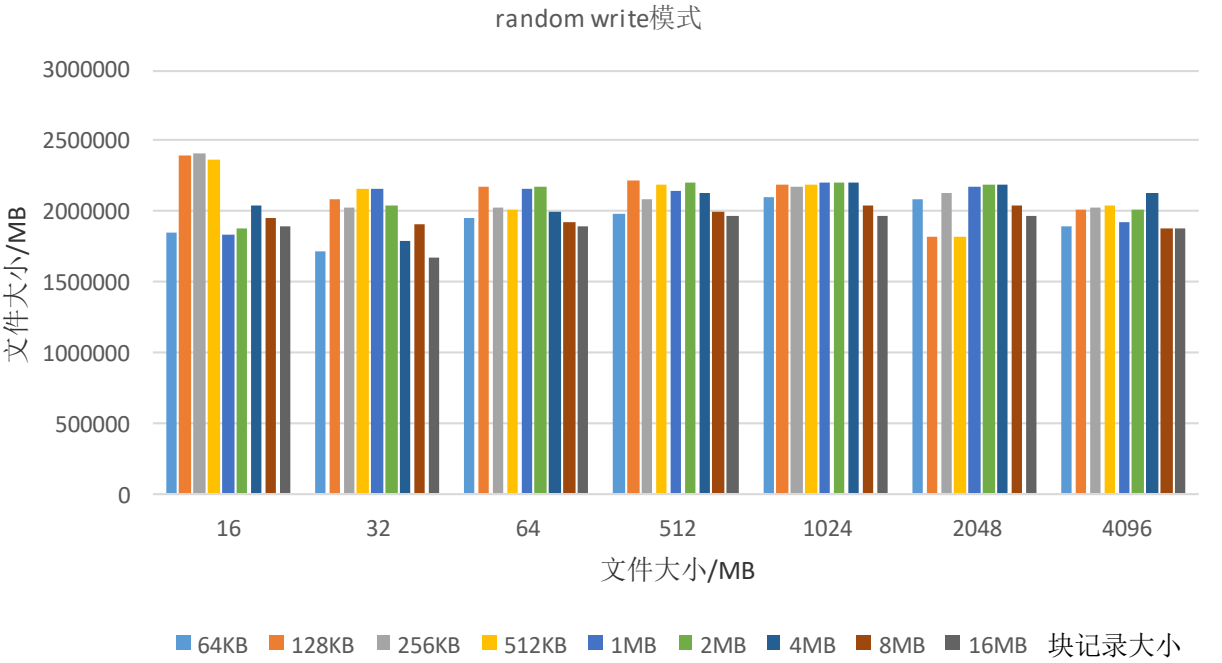

图 5.6 random write 模式测试数据图

由图 5.6 可知，随机写（random write）模式下，总体而言，文件和块记录大小对吞吐量影响比较小。

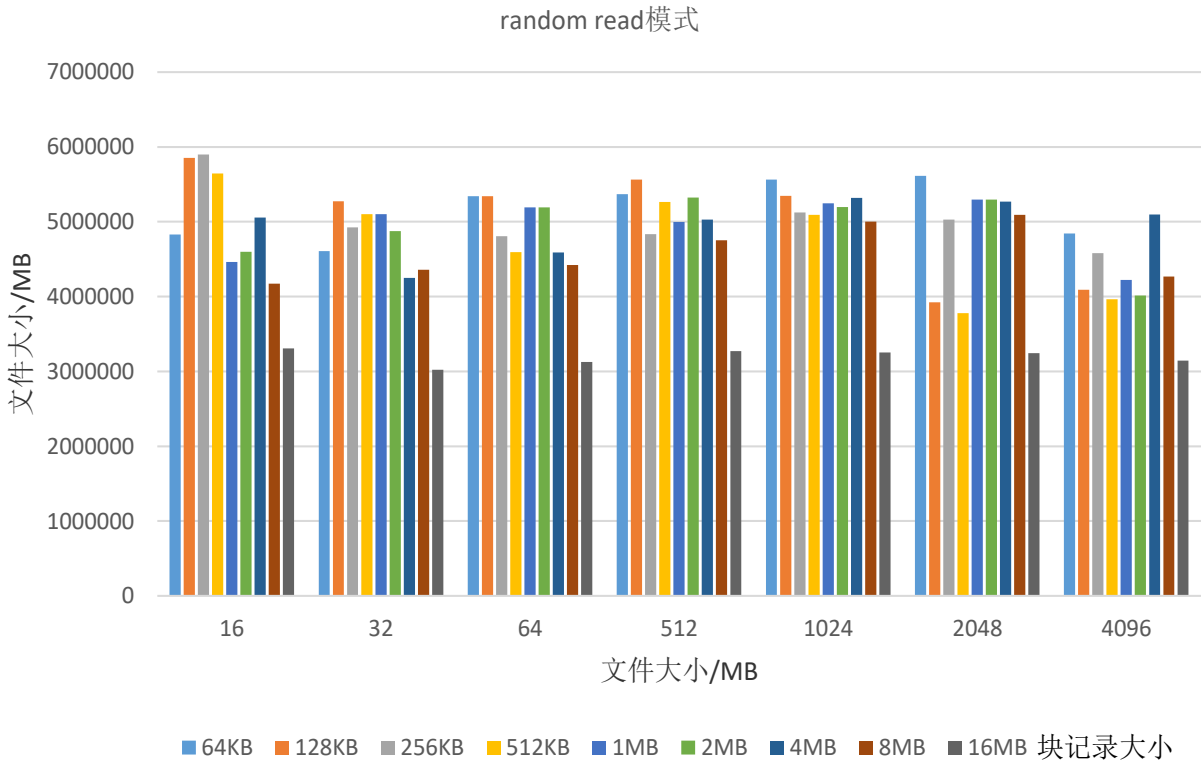

图 5.7 random read 模式测试数据图

由图 5.7 可知，随机读（random read）模式下，同之前两种读模式一样，文件和块记录大小对吞吐量影响比较小，但是当块记录大到 16MB 时，吞吐量下降了 30%-50%。

5.3.3 安全性测试

安全性测试主要对用户名的合法性、访问权限和存储安全性进行测试，部分测试用例如表 5.3 所示。

表 5.3 安全性测试用例表

用例名称	用例编号	模块	测试类型	优先级	概述	步骤	输入	期望输出	实际输出
用户登录验证	T3-01	登录认证模块	安全性	高	验证用户名的合法性	步骤1	输入正确的用户名和密码，包括是合法的字符和合法长度，直接敲击enter键或用鼠标单击登录按钮	登录成功	通过
						步骤2	输入错误的用户名，包括用户名含有非法字符、长度过长、长度过短，直接敲击enter键或用鼠标单击登录按钮	提示“用户名不存在”	
用户操作权限验证	T3-02	数据交互模块	安全性	高	验证用户权限的有效性	步骤1	使用配置读写权限的用户登录，对目标器下的存储卷进行读和写	用户可以正常读写文件	通过
						步骤2	使用只有读权限的用户登录，对目标器下的存储卷进行读和写	用户可以正常读文件，写文件时提示没有写入权限	
存储安全性验证	T3-03	存储资源管理	安全性	高	验证数据存储安全性	步骤1	登录成功后写完数据，人为拔掉某块物理硬盘	十秒钟之内恢复数据	通过

5.3.4 结果分析

由性能测试的各种统计数据所示，相比文件和块记录大小，访问模式对吞吐量的影响是巨大的。无论是对于顺序访问还是随机访问，读数据的吞吐量都比写数据普遍高 100%以上。这是因为在存

储系统的底层，Ceph 分布式系统在写数据的时候，数据最终对象位置 OSD 的计算和数据副本的写入需要消耗更多的资源。同时，从测试结果我们可以得到该存储系统吞吐量的峰值是 6,241,661KB/s.

5.4 本章小结

本章主要对存储虚拟化系统的功能、性能和安全性进行了测试和分析，从结果可以看出，本文所设计的存储系统在功能性、读写性能和安全性上完全符合我们的设计目标，能够满足企业统一管理、分配和维护存储资源的需求。

第六章 总结与展望

针对传统存储体系的性能瓶颈，本文结合分布式和虚拟化技术，设计和实现了一种全新的存储架构。基于 Ceph 分布式系统，把数据的计算和存储负载分担到多个独立的计算机设备；再通过 iSCSI 协议，使用户可以远程访问存储系统，获得一个可以理论上无限扩展的逻辑地址空间。

6.1 论文总结

本文的主要工作为以下几个方面：

（1）统一管理现有的物理资源，建立一个可扩展的虚拟存储资源池，将数据的读写和存储负载分担到多个计算机节点上，并定义了用户基于主机的访问控制列表，根据用户的应用需求分配存储容量，充分实现资源共享。

（2）使用 CHAP 方式对客户端对存储系统的登录访问进行了身份校验，通过 MD5 加密算法对用户密码进行哈希处理，确保用户对存储端所要进行的数据读写安全。

（3）为客户端和存储端之间建立了 iSCSI 会话，定义了数据报文格式，在存储端通过解析会话上传输过来的数据报文来获取客户端的读写请求和数据，并优化了底层存储集群的数据分布算法，减少了因增删存储资源导致的不必要的数据迁移。

（4）对该存储虚拟化系统做了功能测试、性能测试和安全性测试，尤其是对影响该存储系统读写性能的因素进行了重点测试研究，测试文件大小、块记录大小和各种读写模式等因素对存储系统性能的影响。

6.2 工作展望

迎合互联网数据的日益增长趋势，存储技术和体系仍在不断发展中，传统存储架构正在向云存储架构演进。分布式存储技术和存储虚拟化技术是云存储的重要实现方式，本文设计的存储系统突破了容量限制和可扩展性瓶颈，克服了数据多节点负载和用户远端访问限制，在此基础上，仍然可以开展的研究工作包括以下三个方面：

（1）解决数据访问延迟性问题

虽然本论文中的存储系统满足了数据长距离传输的需求，但由于现在互联网的带宽不够理想，IP 网络的速率和延迟都是传输数据的巨大障碍。在私有局域网中可以通过高速的交换机和路由器尽可能解决延迟问题，但是在互联网环境中，由于地域和成本限制，延迟性问题仍比较严重。不过随着广域网通信技术和网络设施的发展，我们相信访问延迟时间会逐步减少。

（2）流量控制

由于用户通过远距离进行数据的访问操作，需要传输的数据在操作系统和网络环境中内要需要通过更多的层才能从网线发出，容易出现流量堵塞问题。对于这个问题，我们可以引入适当的流量控制和数据包分发算法，最大限度的降低流量堵塞。

（3）异常恢复

由于用户在远端读写的数据是在 IP 网上传输的，因而可能出现数据错误（如校验出错），也可能出现数据包丢失。在客户端和存储系统之间的会话必须能够恢复这些错误和丢失的数据包。另外，TCP 连接本身也可能出现故障，存储系统必须能够为活动的任务指派新的连接，并让它在新的连接上继续执行。基于以上异常，我们可以定义一组异常恢复机制，允许存储系统自主从中选择合适的方法来应对不同的异常。

致谢

22 年的学生生涯即将结束，能够在东南大学度过最后三年的学生时代，是我莫大的荣耀。在这里，我很荣幸能学习到我本科阶段从未触碰过的知识和技能。

在论文完稿之际，谨对在本论文的撰写过程中，给予我帮忙的导师和同学，表示由衷的感谢，尤其要感谢我的导师戚晓芳老师，在百忙之中，她对我做了很多的辅导工作，从论文方向的选择、开题报告的撰写、参考资料的查阅，到论文最后的定稿，在各个方面都给我提出了很多建议。没有戚老师的辛苦指导，我绝对不可能顺利成文。戚老师对我的严格要求和悉心指导，将使我终身获益。

感谢研一时候一起看书学习、一起上课吃饭的实验室同学和最棒的室友杨乔虎，我很怀念那时候稚嫩的我们，为了找实习一起战斗。我永远不会忘记，在我刚跨入计算机专业的时候，是你们给了我朴素的计算机启蒙。此时此刻，我想起了我们一起做过的大数据作业和讨论过的算法，一起度过的圣诞平安夜和观赏过的拙政园。而今，我们已经或者即将分散至五湖四海，无论何时何地，希望我们都记得，曾经有个叫做三江院的温暖地方，承载了我们最后的读书年华。

感谢和我一起在杭州实习的小伙伴，那一段青涩的职场生涯至今让我记忆犹新，杭州是那么的好，只是西湖和那时候的城西银泰离我住的地方稍微有点远了。那里还有我的高中同学武晓凯、于文静以及偶尔短暂停留的我最优秀的高中同桌翟金源，在实习之余，你们仿佛就是我与远去多年高中时光的枢纽。和杭州的你们在一起的时候，我可以轻松地抛开任何学业和工作压力，做我自己。

感谢和我在研究生期间一年固定一聚的大学同学们，我怀念我们一起洗试管的升华楼和为考研奋斗的红色主教学楼，不知道翡翠湖校区外面的两家东北大酒店是不是还在营业？也不知道屯溪路校区的篮球场有没有翻新过？我记得这里的所有事情，从大一到大四，那些琐碎的和意义重大的，那些好的与坏的。虽然我已经注定和那里的许多人以后再也不会见面，但我们都曾是这所学校的一部分诶——我们心中最柔软的合工大，而这，就足够了。

感谢在新华三存储产品管理部的同事，你们给了我足够的成长空间，虽然在实习离职时候的新产品发布之际，我已无法继续我们当时未竟的存储事业，但过去，现在，以后，我永远以我是一个华三人自豪，在这里的经历，我将永远铭记于心，尤其是部门欠我的那件文化衫-_-。

感谢最后成文阶段时候实验室给过我热心建议的潘聪、李振南、靳娜娜同学和曾强师兄，你们对我的帮助远比你们想象的要多。你们都是很好很好的人，我衷心希望，当不再做学生以后，我们能够一直保持很好的朋友关系，或许很多年以后，我们可以一起举杯，共同回忆那段曾经的峥嵘岁月，另外，杯子里装的一定不是酒。还有我的初中校友兼高中同学兼研究生校友郭健，三系校友，虽然他在遥远的四牌楼。能够和你们一起做东南大学的一份子，是我的荣幸（手动欠身一礼）。

感谢我的孙朋师兄，他对我的帮助贯穿了我的整个研究生学习生涯，能认识这位米尔大哥可能是我研究生期间最幸运的事。我记得在研究生复试教室外面的第一次唠嗑和入学报到时候的不期而遇，很感谢你最后没有选择去西安交大，也很感谢你一直在我耳边那些不厌其烦的絮叨， 虽然你一直劝我改写 Java，虽然你的那些小吃远比你给我的那些学习工作经验更让我激动。

最后，要感谢我的父母和家人，感谢你们对我读研决定的支持，你们是我 25 年以来最坚强的后盾，在你们面前，我知道我可以一直做那个吃饭挑食的孩子，每当我遇到困难，就会想起你们在 30 年前决定背井离乡的那个夜晚。跟你们这么多年经历的风风雨雨相比，我一直都像是温室里被惯坏了的孩子。

再次感谢所有人，包括科比，包括库里，包括 C 罗，包括四年以前义无反顾决定跨专业考研的我自己，以及今年和我一起毕业的那些小伙伴们，不管你是在南京，大连，北京，上海，厦门，济南，西安，合肥还是纳什维尔，杭州，苏州，广州，长春，沈阳。

谨以此篇致谢，纪念那些已经逝去的少年时光，以及这篇马上就会被遗忘和风干的论文本身。

参考文献

[1] Cherubini G, Jelitto J, Venkatesan V. Cognitive storage for big data[J]. Computer, 2016, 49(4): 43-51.

[2] Xing Y, Zhan Y. Virtualization and cloud computing[M]//Future Wireless Networks and Information Systems. Springer, Berlin, Heidelberg, 2012, 24(4): 305-312.

[3] Rawat A S, Papailiopoulos D S, Dimakis A G, et al. Locality and availability in distributed storage[J]. IEEE Transactions on Information Theory, 2016, 62(8): 4481-4493.

[4] Baker M, Keeton K, Martin S. Why traditional storage systems don't help us save stuff forever[C]// Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability. 2005: 107-120.

[5] Ghazi M R, Gangodkar D. Hadoop, MapReduce and HDFS: a developers perspective[J]. Proceedings of the 2015 Computer Science, 2015, 48(12): 45-50.

[6] Xiao D, Zhang C, Li X. The Performance Analysis of GlusterFS in Virtual Storage[C]//International Conference on Advances in Mechanical Engineering and Industrial Informatics. 2015: 199-203.

[7] Schwan P. Lustre: Building a file system for 1000-node clusters[C]//Proceedings of the 2003 Linux symposium. 2003:380-386.

[8] 黄秋兰, 程耀东, 杜然, 等. 可扩展的分布式元数据管理系统设计[J]. 计算机工程, 2015, 41(5): 26-32.

[9] Chandrasekaran V V, Sivakolundu R, Fong D K F. Methods and apparatus for implementing exchange management for virtualization of storage within a storage area network: U.S. Patent 8,805,918[P]. 2014-8-12.

[10] 王国华. 存储虚拟化分配技术研究[J]. 数字技术与应用, 2015, 12(4): 103-103.

[11] 杨泳丹, 耿贞伟. 利用存储虚拟化技术实施存储优化[J]. 信息技术与信息化, 2015, 26(10): 221-222.

[12] Gibson G A, Van Meter R. Network attached storage architecture[J]. Communications of the ACM, 2000, 43(11): 37-45.

[13] Telikepalli R, Drwiega T, Yan J. Storage area network extension solutions and their performance assessment[J]. IEEE Communications Magazine, 2004, 42(4): 56-63.

[14] Lu Y, Noman F, Du D H C. Simulation Study of iSCSI-based Storage System[C]//MSST. 2004: 399-408.

[15] Leventhal A H. A file system all its own[J]. Communications of the ACM, 2013, 56(5): 64-67.

[16] Porras P, Saidi H, Yegneswaran V. An analysis of conficker's logic and rendezvous points[J]. Computer Science Laboratory, SRI International, Tech. Rep, 2009, 15(4): 36-40.

[17] Amegadzie A, Caccavale F S, Jiang X, et al. CIFS access to NFS files and directories by translating NFS file handles into pseudo-pathnames: U.S. Patent 9,110,920[P]. 2015-8-18.

[18] Wang W, Godfrey M W. A study of cloning in the Linux SCSI drivers[C]//Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on. IEEE, 2011: 95-104.

[19] Larsen S, Sarangam P, Huggahalli R, et al. Architectural breakdown of end-to-end latency in a TCP/IP network[J]. International journal of parallel programming, 2009, 37(6): 556-571.

[20] Zhang M, Liu Y, Yang Q. Cost-Effective Remote Mirroring Using the iSCSI Protocol[C]//MSST. 2004: 385-398.

[21] Lu Y, Du D H C. Performance study of iSCSI-based storage subsystems[J]. IEEE communications magazine, 2003, 41(8): 76-82.

[22] Gudu D, Hardt M, Streit A. Evaluating the performance and scalability of the Ceph distributed storage system[C]//Big Data (Big Data), 2014 IEEE International Conference on. IEEE, 2014: 177-182.

[23] Malanik D, Jaek R. The performance of the data-cluster based on the ceph platform with geographically separated nodes[C]//Mathematics and Computers in Sciences and in Industry (MCSI), 2014 International Conference on. IEEE, 2014: 299-307.

[24] 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.

[25] Zhan K, Piao A H. Optimization of Ceph Reads/Writes Based on Multi-threaded Algorithms[C]//High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on. IEEE, 2016: 719-725.

[26] Wang F, Oral H S, Fuller D, et al. Ceph parallel file system evaluation report[R]. Oak Ridge National Laboratory (ORNL); Oak Ridge Leadership Computing Facility (OLCF), 2013.

[27] Yang C T, Chen C J, Chen T Y. Implementation of ceph storage with big data for performance comparison[C]//International Conference on Information Science and Applications. Springer, Singapore, 2017: 625-633.

[28] Lamport L. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.

[29] Maltzahn C, Molina-Estolano E, Khurana A, et al. Ceph as a scalable alternative to the hadoop distributed file system[J]. login: The USENIX Magazine, 2010, 35(5): 38-49.

[30] Wang F, Nelson M, Oral S, et al. Performance and scalability evaluation of the Ceph parallel file system[C]//Proceedings of the 8th Parallel Data Storage Workshop. ACM, 2013: 14-19.

[31] Skourtis D, Watkins N, Achlioptas D, et al. Latency minimization in SSD clusters for free[R]. Tech. Rep. UCSC-SOE-13-10, UC Santa Cruz, 2013.

[32] Wei Q, Veeravalli B, Gong B, et al. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster[C]//Cluster Computing (CLUSTER), 2010 IEEE International Conference on. IEEE, 2010: 188-196.

[33] Bell T, Bompastor B, Bukowiec S, et al. Scaling the CERN OpenStack cloud[C]//Journal of Physics: Conference Series. IOP Publishing, 2015: 22-28.

[34] Andersen M P, Culler D E. BTrDB: Optimizing Storage System Design for Timeseries Processing[C]//FAST. 2016: 39-52.

[35] 杨飞, 朱志祥, 梁小江. 基于 Ceph 对象存储集群的高可用设计与实现[J]. 微电子学与计算机, 2016, 33(1): 60-64.

[36] 刘飞, 蒋德钧, 熊劲. 异构存储感知的 Ceph 存储系统数据放置方法[J]. 计算机科学, 2017, 6(2): 12-15.

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

[38] Huang M, Luo L, Li Y, et al. Research on data migration optimization of ceph[C]//Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2017 14th International Computer Conference on. IEEE, 2017: 83-88.

[39] Preslan K W, Barry A P, Brassow J E, et al. A 64-bit, shared disk file system for Linux[C]//Mass Storage Systems, 1999. 16th IEEE Symposium on. IEEE, 1999: 22-41.

[40] Yang X, Lehman T. Model driven advanced hybrid cloud services for big data: paradigm and practice[C]//Data-Intensive Computing in the Clouds (DataCloud), 2016 Seventh International Workshop on. IEEE, 2016: 32-36.

[41] 谭文贵，黄英港，王琨. 一种基于 Ceph 提供弹性块存储的研究及实现[J]. 信息通信，2017 21(5): 216-217.

[42] Mickens J W, Nightingale E B, Elson J, et al. Blizzard: Fast, Cloud-scale Block Storage for Cloud-oblivious Applications[C]//NSDI. 2014: 257-273.

[43] 贺昱洁. 负载均衡的大数据分布存储方法研究与实现[D]. 上海: 上海交通大学, 2015.

[44] Klimovic A, Kozyrakis C, Thereska E, et al. Flash storage disaggregation[C]//Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016: 29-30.

[45] Fonville M E. The virtual machine delivery network[D]. University of Twente, 2015.

[46] Wang J, Zhao Z, Xu Z, et al. I-sieve: an inline high performance deduplication system used in cloud storage[J]. Tsinghua Science and Technology, 2015, 20(1): 17-27.

[47] Chadalapaka M, Shah H, Elzur U, et al. A study of iSCSI extensions for RDMA (iSER)[C]//Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications. ACM, 2003: 209-219.

[48] Ren Y, Li T, Yu D, et al. Design, implementation, and evaluation of a numa-aware cache for iscsi storage servers[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(2): 413-422.

[49] Higa R, Matsubara K, Okamawari T, et al. Optimization of iSCSI Remote Storage Access through Multiple Layers[C]//Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on. IEEE, 2009: 612-617.

[50] Cao J, Wang S, Dai D, et al. A generic framework for testing parallel file systems[C]//Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS), 2016 1st Joint International Workshop on. IEEE, 2016: 49-54.

[51] Arroyo J P, Cors J, Dosch D L, et al. Automatic multipath iSCSI session establishment over an arbitrary network topology: U.S. Patent 9,253,256[P]. 2016-2-2.

[52] Shimano S, Nunome A, Hirata H, et al. An information propagation scheme for an autonomous distributed storage system in iscsi environment[C]//Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence (ACIT-CSI), 2015 3rd International Conference on. IEEE, 2015: 142-147.

[53] Gode N, Kashalkar R, Kale D, et al. Feature-based Comparison of iSCSI Target Implementations[J]. International Journal of Computer Applications, 2014, 85(16):34-39.

[54] Fang X, Chen J, Ye F, et al. Introduction of metadata-request queue with immediate response for i/o path optimizations on iscsi-based storage subsystem[C]//Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on. IEEE, 2015: 100-105.

[55] Chakhaiyar M G, Jibbe M K. Method and system for coupling serial attached SCSI (SAS) devices and internet small computer system internet (iSCSI) devices through single host bus adapter: U.S. Patent 8,694,723[P]. 2014-4-8.

[56] Pirahandeh M, Kim D H. Energy-aware and intelligent storage features for multimedia devices in smart classroom[J]. Multimedia Tools and Applications, 2017, 76(1): 1139-1157.

[57] Zhou R, Ai Z, Hu J, et al. Data integrity checking for iSCSI with Dm-verity[M]//Advanced Technologies, Embedded and Multimedia for Human-centric Computing. Springer, Dordrecht, 2014: 691-697.

[58] Ilham A. A, Usman S. Performance analysis of extract, transform, load (ETL) in apache Hadoop atop NAS storage using ISCSI[C]//Computer Applications and Information Processing Technology (CAIPT), 2017 4th International Conference on. IEEE, 2017: 1-5.

[59] Bahaweres R B, Budiman T R, Adriansyah A. Performance analysis of iSCSI target in wireless LAN using standard LIO[C]//Cyber and IT Service Management (CITSM), 2014 International Conference on. IEEE, 2014: 108-112.

[60] Allayear S M, Salahuddin M, Ahmed F, et al. Introducing iSCSI protocol on online based mapreduce mechanism[C]//International Conference on Computational Science and Its Applications. Springer, Cham, 2014: 691-706.

[61] Nunome A, Hirata H, Shibayama K. A distributed storage system with dynamic tiering for iscsi environment[C]//Advanced Applied Informatics (IIAIAAI), 2014 IIAI 3rd International Conference on. IEEE, 2014: 644-649.

[62] Salmani V, Shin S W. An Empirical Evaluation Methodology for iSCSI Storage Networking[C]//Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on. IEEE, 2015: 216-225.

[63] Neira-Ayuso P, Gasca R M, Lefevre L. Communicating between the kernel and user-space in Linux using Netlink sockets[J]. Software: Practice and Experience, 2010, 40(9): 797-810.

[64] Pope S L, Riddoch D J. Epoll optimisations: U.S. Patent 9,384,071[P]. 2016-7-5.