分类号	TP311	密级	公开
UDC	004	学位论文编号	D-10617-308-(2019)-02069

重庆邮电大学硕士学位论文

中文题目	**Ceph 异构存储优化机制研究**
英文题目	**Research on Ceph Heterogeneous Storage Optimization Mechanism**
学　　号	**S160201071**
姓　　名	姚朋成
学位类别	工学硕士
学科专业	计算机科学与技术
指导教师	熊安萍　教授
完成日期	**2019 年 5 月 20 日**

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含他人已经发表或撰写过的研究成果，也不包含为获得 重庆邮电大学 或其他单位的学位或证书而使用过的材料。与我一同工作的人员对本文研究做出的贡献均已在论文中作了明确的说明并致以谢意。

作者签名：姚朋成　　日期：2019年6月6日

学位论文版权使用授权书

本人完全了解 重庆邮电大学 有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权 重庆邮电大学 可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

作者签名：姚朋成　　导师签名：

日期：2019年6月6日　　日期：2019年6月6日

摘要

近几十年来，随着互联网、物联网和移动互联网等应用的不断更新和发展，造成数据呈现海量式的增长。在越来越大的数据量面前，传统的单点存储已经不能够满足新时代的需求。在新的大数据背景下，分布式存储方式是解决海量数据存储的有效解决方案。Ceph 由于其支持多种存储服务，既提供文件存储服务，又提供对象存储和块存储服务，同时又能够较好地克服了单点故障问题且具有良好的扩展性，从而使其得到了广泛关注。然而，Ceph 的多副本机制使其在读写性能方面并未充分发挥集群优势，同时，在异构存储结构下，现有的存储策略限制了 Ceph 存储集群的性能。为此，论文深入研究了 Ceph 的存储机制，并提出了相应的优化策略。具体研究工作如下：

1. 研究了 Ceph 异构存储优化机制。首先针对 Ceph 采用强一致性的写入策略导致整个集群写延迟较高的问题，提出一种基于副本弱一致性的写入策略。进一步地，为有效的利用从副本节点的 I/O 性能，提出了一种基于副本弱一致性的组合存储优化策略。并对 Ceph 原生策略与优化存储策略做了实验对比，结果表明，基于副本弱一致性的组合存储优化策略在写吞吐量、随机读吞吐量、顺序读吞吐量、SSD 命中率以及降低写延迟方面都有了较大的提升。

2. 基于 SSD 和 HDD 的异构存储架构，提出一种组合存储优化策略，从而充分利用异构环境下不同介质的存储设备的性能特点。通过建立对象热度模型，提出基于热度划分模型的异构副本组合存储策略，并分别对基于热度划分模型的异构副本组合存储策略、组合存储优化策略、原生策略，从集群读效率和异构副本不同组合方式读写性能等方面进行比较，结果表明，基于热度划分模型的异构副本组合存储策略在发挥异构副本性能特点方面有巨大优势。

论文研究工作表明，在 SSD 和 HDD 的异构存储环境下，采用副本弱一致性策略以及组合存储策略，能够优化 Ceph 的存储机制，并有效提升 Ceph 集群的访问效率。

关键词：Ceph，异构存储，副本弱一致性，组合存储

Abstract

With the continuous updating and development of applications such as the Internet of Things, the mobile Internet and the Internet in recent decades, the data has grown exponentially. In the face of increasing data volume, the traditional single-point storage has not been able to meet the needs of the new era. In the context of new big data, the distributed storage is an effective solution to massive data storage. Ceph has attracted wide attention because it supports multiple storage services, not only provides file storage services, but also provides object storage and block storage services. At the same time, it overcomes the single point failure problem and has good scalability. However, Ceph's multi-replica mechanism does not give full play to its cluster advantage in read and write performance. At the same time, under heterogeneous storage structure, existing storage strategies also limit the performance of Ceph storage cluster. Therefore, the storage mechanism of Ceph is deeply studied and the corresponding optimization strategy is proposed. Specific research work is as follows:

1. The optimization mechanism of Ceph heterogeneous storage is studied. Firstly, a write strategy based on the weak consistency of replicas is proposed to solve the problem of high write latency in the whole cluster caused by Ceph's strong consistency. Furthermore, in order to effectively utilize the I/O performance of slave replica node, a composite storage optimization strategy based on the weak consistency of replicas is proposed. The experimental comparison between the Ceph native strategy and the optimized storage strategy shows that the combined storage optimization strategy based on weak consistency of replicas improves greatly in write throughput, random read throughput, sequential read throughput, SSD hit rate and write latency reduction.

2. Based on the heterogeneous storage architecture of SSD and HDD, a combined storage optimization strategy is proposed to make full use of the performance characteristics of different media storage devices in heterogeneous environments. By establishing an object heat model, a heterogeneous replica combination storage strategy based on the heat partition model is proposed. Thesis compares the cluster reading efficiency and the read-write performance of different combinations of heterogeneous replicas based on the thermal partition model. The results show that the heterogeneous replica combination storage strategy

based on the thermal partition model has great advantages in giving full play to the performance characteristics of heterogeneous replicas.

The research results show that under the heterogeneous storage environment of SSD and HDD, the storage mechanism of Ceph can be optimized and the access efficiency of Ceph cluster can be effectively improved by using the weak consistency strategy of replicas and the combined storage strategy.

Keywords: Ceph, heterogeneous storage, weak consistency, combination storage

目录

图录

表录

第 1 章 引言

1.1 研究背景和意义

近几十年来，随着互联网技术的迅速发展，网络数据呈现出快速的增长趋势[1]，尤其是在物联网、移动互联网、互联网等应用上的不断扩展，产生了海量数据，海量数据的分析应用往往会带来更大的经济价值[2]。然而，巨量数据的存储与计算是海量信息处理的基础，在越来越大的存储需求面前，传统的单点存储方式由于其容量和价格等方面的局限性，已经不能够满足新时代对存储容量的迫切需求[3]。分布式存储是解决海量数据存储的有效解决方案[4]。

分布式存储系统通过集群、网络等方式连接起来从而形成一个逻辑上的存储实体，对外提供统一的存储服务，并在存储过程中通过自身机制将文件存放在特定的一个或者一些存储资源上，从而实现存储资源的有效整合，逻辑上实现了存储设备容量上的扩充以及存储成本的低廉。

在数量众多的分布式文件系统中，大多数的文件系统在扩展性和映射效率方面均有一定的不足[5]，如传统的分布式文件系统 HDFS[6]，将文件访问信息存放在元数据服务器中，但是元数据服务器可扩展性尚不够灵活，更重要的是当小文件众多时，对于大量小文件的并发访问会造成元数据服务器瓶颈问题，从而影响文件访问效率[7]，故而在海量数据的存储需求下，必然需要一个具有无限扩展性、快速映射能力的分布式存储系统。

Ceph 是一个新兴的开源分布式存储系统[8]，是一个可靠地、自动重均衡、自动恢复的分布式存储系统[9]，由于其具有无限扩展和快速映射等突出特点已经得到了越来越多的关注和应用。 Ceph 是加利福尼亚大学的 Sage Weil 在 2007 年提出的[10]，起初只是一项关于 Ph D 存储系统的研究项目。自 2012 年 6 月第一个版本发布以来，由于其可以同时支持多种存储服务，既可以提供对象存储服务、又可以提供块存储服务和文件存储服务[11,12]，使其成为 OpenStack 的默认存储后端[13]。同时，Ceph 采用伪随机的数据映射算法[14]，取代传统分布式存储系统的元数据节点，从而使客户端可以更快速便捷地知道数据的存放位置。由于 Ceph 采用映射算法来定位文件的存放位置，使其不再依靠元数据服务器存储文件信息，从而使其具有无限扩展的能力[15]。目前，无论是在国内还是国外，

Ceph 都得到了越来越多的应用，例如雅虎公司利用 Ceph 构建用于邮箱和博客的后端对象存储系统，而国内如华为等也利用 Ceph 作为云主机的存储后端。

Ceph 在进行对象写入时，通过数据映射算法得到该对象的一组存储节点，并将第一个存储节点作为主副本节点，其他存储节点为从副本节点，然后客户端将对象发送给主副本节点进行写入，并由主副本节点将对象数据发送给从副本节点进行写入，待从副本节点中的对象写入完成后反馈给主副本节点，主副本节点收到所有从副本节点的写入磁盘完成的通知，且主副本节点也写入磁盘完成后，才响应客户端，在此期间客户端不能进行其他的操作，即采用了主、从副本的强一致性策略[16]，这将极大的增加整个集群的写延迟，进而降低写效率。同时 Ceph 在读取对象，通过数据映射算法得到该对象的所有存储节点，通过主副本节点读取对象，并没有充分利用从副本节点从而导致性能损失，进而影响整个集群的读效率。更为重要之处在于，随着不同介质的存储设备增多以及价格等因素，使得集群存储设备异构情况变得更加普遍，然而 Ceph 在设计之初没有考虑到 SSD 和 HDD 混合存储环境，从而造成在异构环境下不能有效发挥异构存储设备的性能特点[17]。

针对上述问题，本文基于 SSD 和 HDD 的异构存储环境，提出一种基于 Ceph 的数据副本弱一致性模型[18,19]，提高整个集群的写效率；通过利用 Ceph 集群副本机制，提出一种优化的读策略以提高整个集群的读取性能。针对 Ceph 异构存储环境，本文提出一种组合存储优化策略和异构副本组合方式，从而充分利用异构副本的性能特点，以达到提升整个 Ceph 集群性能的目的。

1.2 国内外研究现状

针对分布式存储系统的研究主要集中在如何提升集群读写效率和异构存储方面。下面将对传统的分布式存储系统和 Ceph 的研究现状进行分析。

1.2.1 分布式存储研究现状

分布式存储系统通过整合存储资源，对外提供统一的存储服务，具有廉价性和大容量等特点，随着大数据应用对存储需求的发展，分布式存储受到了广泛关注。

1. 读写效率方面。葛微等[20]针对 HBase 只支持主键索引，而在对非主键进行查询时的效率较低问题，提出了一种支持 HBase 非主键索引的模型和方法。通过建立持久化

的索引机制，并提出一种缓存的替换策略和数据缓存技术，从而降低访问索引数据的时间开销。 周江[21]等设计和实现了一个分布式文件系统并命名为 Clover。该系统为了解决传统分布式文件系统扩展性不足的问题，采用基于目录和一致性 Hash 相结合的名字命名空间，从而解决元数据的扩展性问题；同时为了保证元数据在集群中的一致性，系统采用了经典的两阶段提交协议；采用了全局回复机制和热备方案保证元数据信息的可靠性。Qi Mu[22]针对 HDFS 在存储大量的小文件时，用户和集群交互将变得更加频繁，从而导致消耗大量的名称节点内存，进而导致系统的性能较差的问题。作者改进存储结构中的块和建立块的二级索引机制，也就是将多个小文件合并，并且在合并小文件后建立组合索引，从而来提高访问效率。Xiao Tao[23]针对 GlusterFS 中文件元数据查找的重大瓶颈，通过将小文件合并到大文件中并重新设计元数据结构，从而减小元数据的大小，并将整个文件的元数据存储在主存储器上，达到优化小文件读写的性能。

2. 异构存储方面。杨东梅[24]等针对异构存储下 HDFS 不能发挥异构存储设备的性能特点的问题，将存储节点划分高性能节点和低性能节点。再根据其动态的性能如 CPU 使用率等，将存储节点划分为三个级别，根据文件的访问频率对存储对象进行动态的调整，从而提高数据访问效率。Kumar Rishabh[25]针对异构存储环境，通过建立 I/O 工作负载分配平台 HetStore 来实现最佳的 I/O 负载的分配，以提高整体性能，并在 HDFS、GlusterFS 等分布式系统中验证了模型的有效性。Krish K.R[26]等针对 HDFS 不考虑底层设备是否异构，不考虑底层存储设备的 I/O 特性，无法发挥异构存储设备的不同优势的问题，提出一种基于不同存储机制的分层存储系统 HatS，它是一种支持异构性的分层存储系统，它将 HDFS 重新设计为一个多层存储系统，该系统无缝地将异构存储技术集成到 Hadoop 系统中。

1.2.2 Ceph 研究现状

针对 Ceph 集群在混合存储环境下不能有效发挥异构副本性能特点，以及写延迟较高的问题，国内外都有广泛研究。针对读写效率方面的问题，主要通过改用弱一致性模型或纠删码来降低写延迟；针对异构存储问题，则结合 SSD 与 HDD 的混合的存储架构来实现存储策略的优化。

1. 读写效率方面。Ceph 在写操作时，客户端将对象数据发送给该对象对应的一组存储节点中的第一个节点也就是主副本节点后，由主副本节点将对象数据发送给所有的

从副本节点，当所有的存储节点都将该对象持久化成功后，主副本节点才会回复给客户端写入成功的通知，在此期间客户端将不能够进行其他的操作，从而造成集群的写延迟较高；在读操作时，Ceph 只从该对象的主副本节点中读取该对象，并没有利用从副本节点的 I/O 性能，最终造成整个集群的读效率较低。针对该问题，刘鑫伟[27]首先提出通过收集系统的读写操作数，通过一定的方法将系统按照读写操作数的多少划分不同的状态，然后针对不同的状态使用不同的算法，来得到当前状态下系统的同步写的副本数，从而将 Ceph 的强一致性副本策略优化为一种用户可配置的基于读写比例的动态策略，从而提高了写效率。刘莎[28]针对 Ceph 系统默认使用的 RS 纠删码算法的不足，提出 LRC 纠删码算法，即在 RS 纠删码的基础上添加额外的局部校验块来提高 RS 码的修复效率。Ke Zhan 等[29]则基于命令行和库函数，应用两种多线程的算法优化读写效率。对于小文件和大文件，分别利用不同的多线程算法。Chun-Feng Wu[30]则针对 Ceph 和 HDFS 分别适合小文件和大文件存储的特点，提出一种基于 Ceph 和 HDFS 混合分布式文件系统，将小文件放在 Ceph 集群中，大文件则放在 HDFS 中，从而达到提升写效率。上述方案要么设计过于复杂不利于实现，要么是在牺牲安全性的基础下提高写效率，而且没有特别关注读效率的问题。

2. 异构存储方面。Ceph 在设计之初并未考虑副本节点的异构存储问题，故而造成 Ceph 在异构情形下不能有效发挥异构存储设备的性能特点。针对该问题，詹玲[31]等设计了一套基于 MRAM 的内存管理模块，从而将多副本机制改为 MRAM 与磁盘结合的方式，其中 MRAM 模块用来存放对象的主副本，磁盘用来存放对象的从副本，相当于在 Ceph 原有的基础上添加了一层由 MRAM 组成的模块的异构副本存储策略，进而让 Ceph 在异构存储介质下发挥异构特性，提高集群性能。Stefan Meyer[32]等提出在异构存储环境下根据磁盘类型、位置以及底层配置对集群进行分区，并把不同的分区划分到不同的存储池中，进而创建异构池来提供不同的存储服务，从而发挥异构存储介质的性能特点，提升集群性能。其次，Veda Shankar[33]等提出采用英特尔高速缓存加速软件(Intel Cache Acceleration Software)结合高性能固态硬盘(SSD)，通过智能缓存而不是采用极端的开销的数据中心。Intel CAS 与服务器内存交互，创建一个多级缓存，并自动确定活动数据的最佳缓存级别，让应用程序执行得更快，从而发挥异构设备的特性。上述方案一定程度上解决了 Ceph 不感知异构的问题，但是方案都不够灵活，甚至有的方案还会一定程度上降低读写效率。

1.3 课题研究内容

本文针对 Ceph 的读写延迟较高以及 Ceph 不感知异构的问题，寻找一种能够有效降低 Ceph 读写延迟的方法，并有效发挥异构副本的性能特点，从而有效提高整个 Ceph 集群的读效率和写效率。本文主要做了以下研究工作：

1. 研究了 Ceph 异构存储优化机制。首先针对 Ceph 采用强一致性的写入策略导致整个集群写延迟较高的问题，提出一种基于副本弱一致性的写入策略。当集群在写入对象时，通过客户端将对象数据发送给主副本节点，当对象数据在主副本节点写入磁盘完毕后，则立即返回 Commit 通知给客户端，再由主副本节点将对象数据更新到从副本节点，从而有效降低集群的写延迟。进一步地，利用 Ceph 集群的副本机制，当集群在读取操作时，为了有效的利用从副本节点的 I/O 性能，提出了一种基于副本弱一致性的组合存储优化策略。在以上研究的基础上，采用对比实验方法，分别对 Ceph 原生策略下和本文策略下写延迟，随机读吞吐量和顺序读吞吐量进行实验对比，从而验证优化机制的有效性。

2. 针对 Ceph 在设计之初未考虑 SSD 和 HDD 混合存储架构，从而不能有效发挥异构存储设备的性能特点的问题，提出一种组合存储优化策略，有效利用 SSD 和 HDD 的不同组合方式，从而充分利用异构环境下不同介质的存储设备的性能特点。进一步地，通过建立对象的热度模型，提出基于热度划分模型的异构副本组合存储策略。最后，分别对基于热度划分模型的异构副本组合存储策略、组合存储优化策略、原生策略，从集群的读效率以及不同组合方式的读写性能等方面进行比较，从而验证本文方法的有效性。

1.4 论文的组织结构

第 1 章首先阐述了异构环境下 Ceph 存储优化机制研究的背景和意义，其次介绍了针对 Ceph 读写效率较低和不感知异构的国内外研究现状，最后介绍了本文的研究主要内容。

第 2 章首先介绍了 Ceph 的基本概念以及定义，随后阐述 Ceph 强一致性策略模型和读策略，接着详细介绍了 Ceph 异构问题，最后阐述了本文的动机和目标。

第 3 章首先分析 Ceph 存在的问题与不足，然后针对 Ceph 写效率较低的问题提出了副本弱一致性模型，随后提出了一种多副本读优化策略，并在此基础上提出基于弱一致性的异构组合存储策略，最后进行实验对比及结果分析。

第 4 章首先对异构副本组合方式进行了分析，随后给出了对象热度划分模型，接着在此基础上提出了基于热度划分模型的异构副本组合存储策略，最后进行实验对比及结果分析。

第 5 章将对本文的研究工作进行总结，并将阐述未来工作的重点和方向。

第 2 章 相关基础

Ceph 的第一个版本发布于 2012 年，是一个新兴的开源分布式存储系统，支持多种存储服务，既提供文件存储服务，又提供对象存储和块存储服务。Ceph 采用伪随机的数据映射算法从而消除了对系统单一中心节点的依赖，并且提供自治服务，是一个具有自我管理和自我修复的能力的分布式文件系统[34]。Ceph 的逻辑结构如图 2.1 所示自下而上包括四个层次：基础存储系统 RADOS、LIBRADOS、应用接口、APP 层。

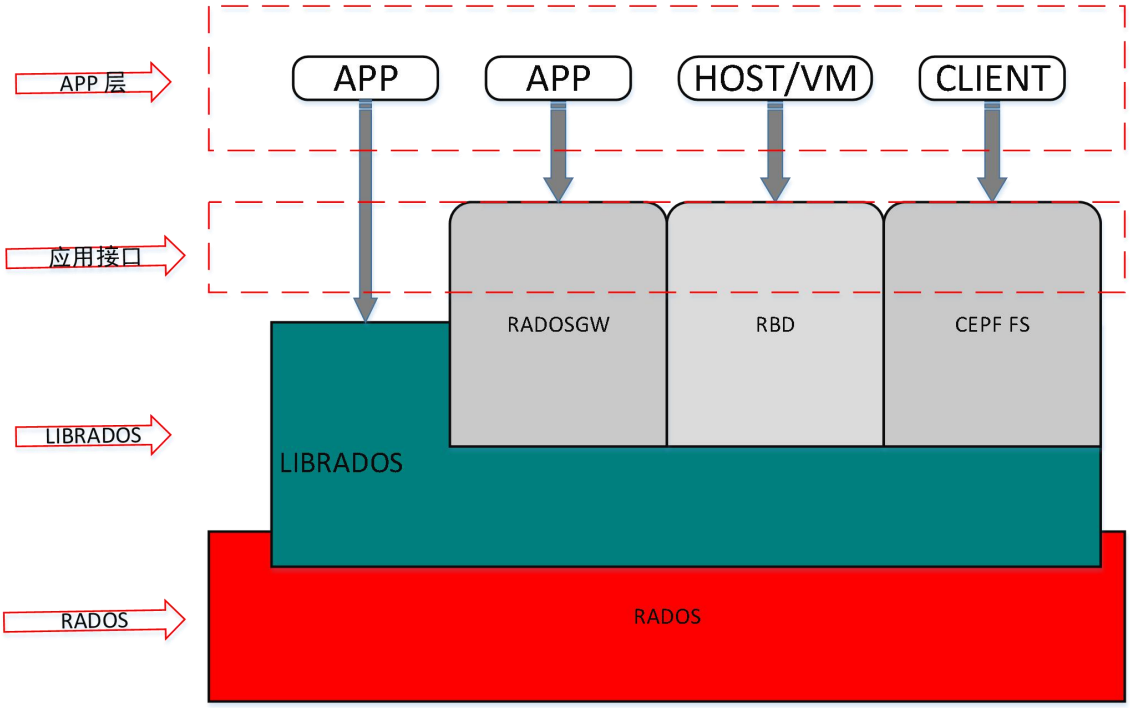

图 2.1 Ceph 逻辑架构图

(1) RADOS(Reliable, Autonomic,Distributed Object Store) [35]是 Ceph 的底层核心，Ceph 的所有存储能力都是以这一层作为支撑的。

(2) LIBRADOS[36]是对下层的 RADOS 进行封装和抽象形成基础库，向上提供支持 C/C++、Java 和 Python 等语言的 API，以便上层基于 RADOS 开发对象、文件、块或原生对象的应用，同时这一层也是允许开发人员直接调用使用的。

(3) 应用接口层[37]，这一层主要负责对下层 LIBRADOS 提供的 API 进行封装，提供对象存储接口、块存储接口和文件存储结构，从而便于客户端或者应用使用。

(4) APP 层也称客户端层，主要是对应用接口层提供的各个应用接口在不同场景下的应用，例如基于对象存储网关开发对象存储的应用，基于 Ceph FS[38,39]开发基于面向文件系统的存储应用等。

2.1 RADOS（基础存储系统）

RADOS 是 Ceph 的底层核心[40]，Ceph 的所有存储能力都是以 RADOS 作为底层支撑的[41]。RADOS 本身就是一个能够自我修复的自动、可靠、智能的对象存储系统。如图 2.2 所示，RADOS 主要由具有系统状态监测和维护功能的 Monitor 和存储节点 OSD(Object Store Device)这两种节点组成[42]。

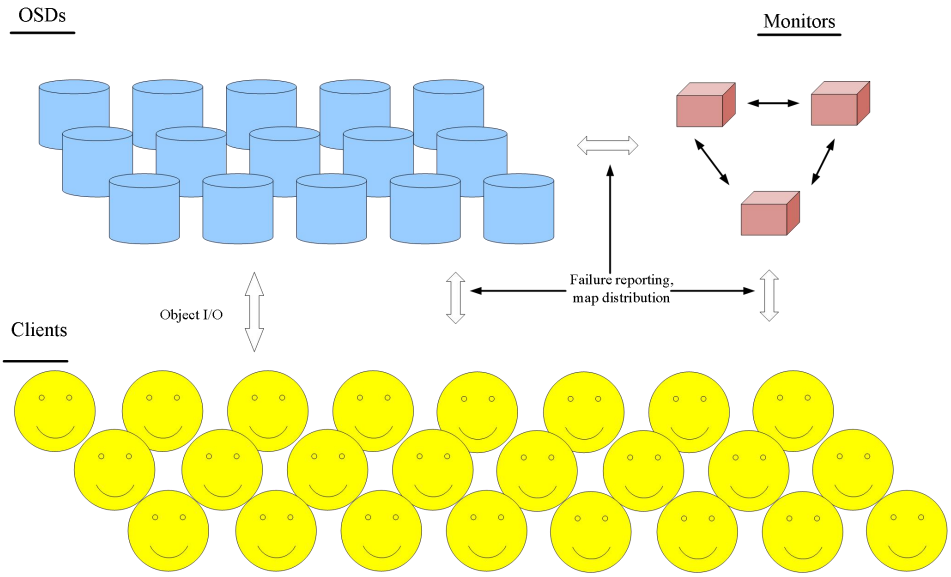

图 2.2 RADOS 结构图

OSD：每一个 OSD 拥有自己 OSD deamon 守护进程，除了具有基本的数据对象存储的功能以外，还具有数据复制、数据恢复、数据迁移等功能，其会将一些重要的信息发送给 Monitor，例如心跳、版本、地址等[43]。一个具有可靠性的 Ceph 集群需要将数据至少备份到两个 OSD 中。同时，OSD 还会和其他 OSD 进行交流，完成数据迁移或者共同维护 Cluster Map 的功能。最后，OSD 还会与 Client 进行通信，使 Client 完成数据访问的操作。

Monitor：Ceph 的监控器，主要负责维护集群的健康状态和相关元信息，其中包括用于记录数据分布和 OSD 状态的 OSD Map，记录 Monitor 状态的 Monitor Map， 记录 PG 信息的 PG Map，还有记录集群设备等信息的 CRUSH Map。

2.2 CRUSH 算法

CRUSH(Controled Replication Under Scalable Hashing)[44]算法是 RADOS 的核心算法，Ceph 正是通过 CRUSH 这个可扩展的伪随机的数据分布算法将对象映射到一组 OSD 上的。

在客户端访问 Ceph 存储节点的过程中，CRUSH 能将 PG 映射到一组存储节点 OSD 上。但 CRUSH 算法映射的结果并不是一成不变的，受到以下因素的影响：一是集群当前的状态，即 Cluster Map[45]。当集群中 OSD 相关因素改变，如 OSD 处于 down 状态或者新加入 OSD 时，Cluster Map 会进行变化，也影响了 PG 映射到一组 OSD 的结果；二是 Placement Rules，即数据放置规则[46]，通过改变此策略，可以指定 PG 所映射的一组 OSD 是存在于哪个特定的机架或者机器上。而一般情况下，存储策略很少进行改变，但因为集群中存储节点经常添加或删除，使得集群的结构也经常发生变化，从而对 CRUSH 的映射结果造成影响。然而 CRUSH 算法设计的非常巧妙，使得 CRUSH 能帮助 Ceph 灵活地处理存储节点的添加和删除等动态变化，最小化因存储节点的改变造成的数据迁移活动，不仅使得 Ceph 具有更好的可伸缩性，而且能够让数据最终非常均衡的分布到存储节点上，这种算法比普通的 Hash 算法具有更好的特性[47]。

CRUSH 在最初实现时，Sage Weil 设计了四种不同的基本选择算法[51]，如表 2.1 所示，在四种实现算法中 straw 算法的执行效率较低，但是添加和删除元素时效率是最好的，在实际应用中，都使用 straw 算法，其他三种算法基本上不会使用。straw 算法又被称为抽签算法，通过指定的输入，然后为每一个算法随机计算一个签长，最后输出签长最长的元素。CRUSH 算法为了让 Ceph 集群中不同容量的存储节点，发挥不同的作用，让容量大的存储节点存储较多的数据，而容量小的存储节点存储较少的数据，从而保证对象在整个集群中的合理分布，所以 CRUSH 引入了一个权重的概念，并通过容量来决定权重的大小，来实现数据的合理分布。straw 算法的执行结果取决于三个因素：元素编号、固定输入和元素权重，其中元素编号只起到一个随机种子的作用，所以在 straw 算法中，发挥决定性作用的是权重这个因素。故而，每一个元素的签长只与元素本身的

权重相关。但是当集群中 OSD 添加或删除后会引起不相关数据的迁移，Sage 针对 straw 算法的这个不足之处，提出了修正后的 straw2 算法。

表 2.1 CRUSH 四种基本选择算法对比

算法类型	unique	list	tree	straw
时间复杂度	O(1)	O(N)	O(log(N))	O(N)
添加元素	差	最好	好	最好
删除元素	差	差	好	最好

2.3 分层集群映射 Cluster Map

Cluster Map 是 Ceph 集群拓扑结构的逻辑描述形式，主要有设备 device 和桶 bucket 组成。

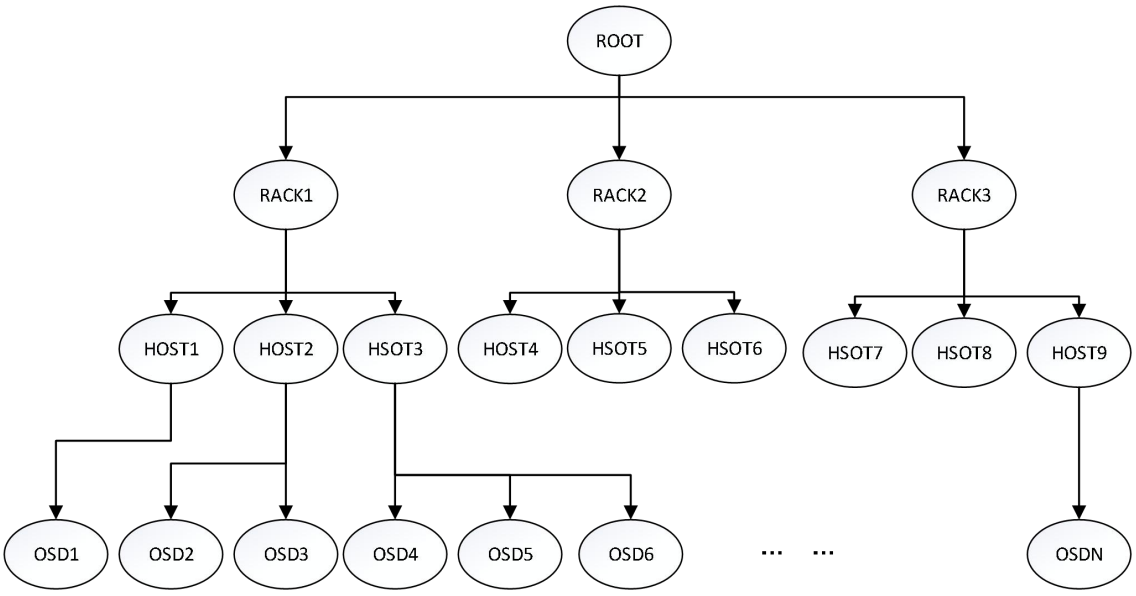

图 2.3 Cluster Map 树状结构

如图 2.3 所示，在 Cluster Map 这种树形结构中的叶子节点都是由 Ceph 中最小的物理设备 OSD 组成的，称为 device，除 device 外其他的所有节点都被称为 bucket[48]，每一个 bucket 都是由 OSD 或者低一层的 bucket 组成的，根节点 root 是整个树状结构的入口。在 Cluster Map 中每一个节点都是有权重的，其中 device 的权重默认与设备的容量成正比的，当然管理员也可以通过设置来控制 device 的权重，而 bucket 的权重则由组成它的所有孩子节点的权重之和得到，在 Ceph 集群进行数据映射时，权重值越高的节点

被选中的概率就越大，则更多数据会对其进行访问。同时在 Cluster Map 中的每一节点都有唯一的标识和类型，用来标识每一个节点在 Cluster Map 这个树状结构中的层次和位置，但是只有叶子节点，也就是 device 才拥有非负 ID。常见的节点类型如表 2.2 所示。

表 2.2 Cluster Map 常见的节点类型

类型 ID	类型名称	类型 ID	类型名称
0	osd	6	pod
1	host	7	room
2	chassis	8	datacenter
3	rack	9	region
4	row	10	
5	pdu		

2.4 数据放置流程

数据放置规则 Placement Rules 是 Ceph 的数据放置规则，其规定了每个数据对象是如何分布到集群中去的，比如，一组副本对象是分配到特定的某个 rack 上，还是分配到不同的 host 上。每种规则包含以下三种操作：

take：选择一个 bucket 作为后续步骤的输入，默认以 root 节点作为输入的开始。

select：由于 Ceph 同时支持多副本和纠删码这两种备份策略[49]，相应的 select 支持 firstn 和 indep 这两种选择算法来选择指定数量的 items。实现两种选择算法都是使用深度优先，并无显著不同，主要区别在于纠删码要求结果是有序的，因此，如果无法满足指定数量的输出，那么 firstn 会返回形如[1,2,4]这样的结果，而 index 会返回形如[1,2,CRUSH_ITEM_NODE,4]这样的结果，即 index 总是返回要求数量的条目，如果条目不存在，则使用 CRUSH_ITEM_NODE 来填充，无论是 firstn 还是 index 都会触发 select 执行。select 以 firstn 选择算法选择指定数量 items 的流程如图 2.4 所示。

emit：输出最终选择的结果。

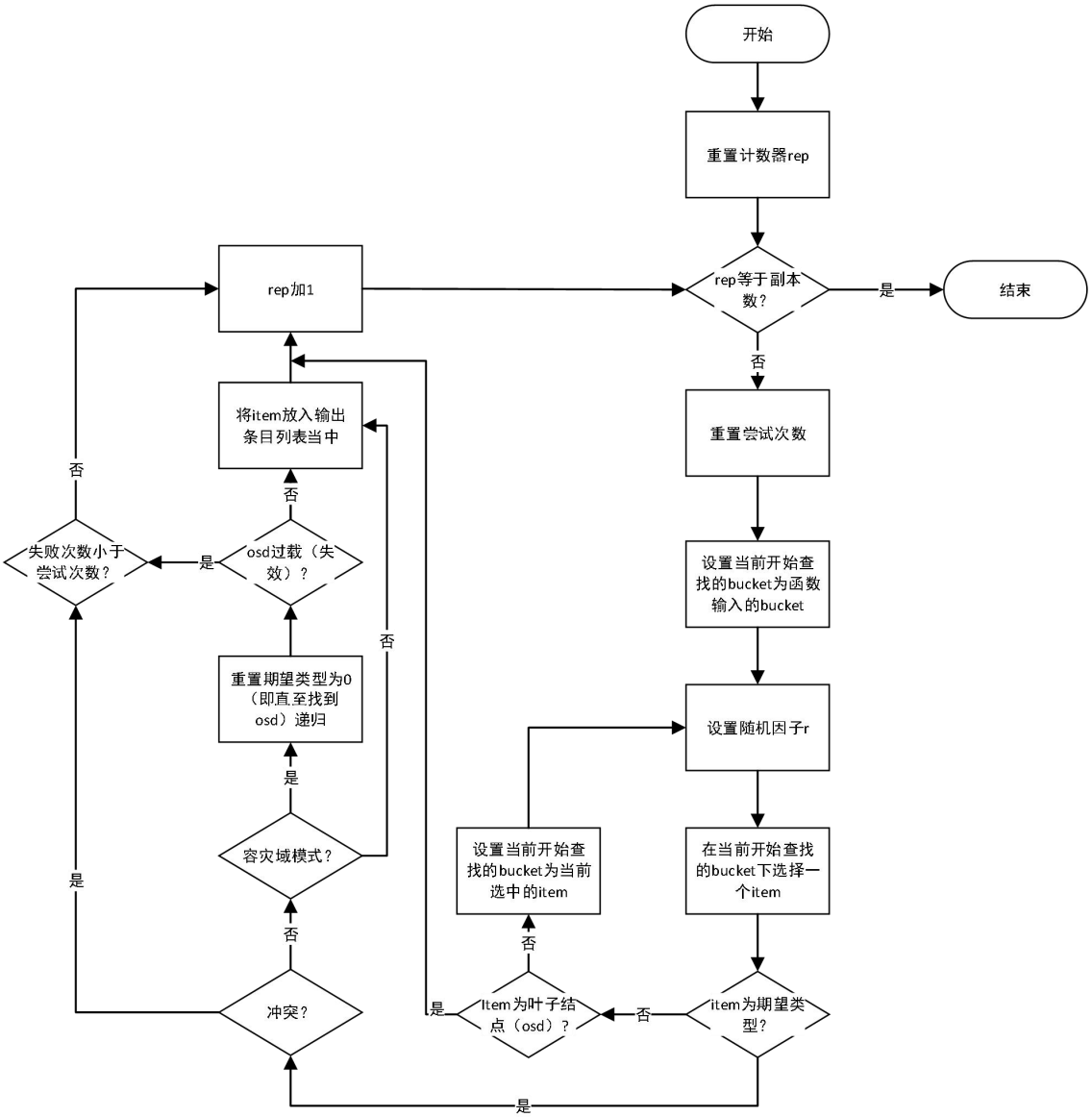

图 2.4 select 查找流程图

如表 2.3 所示，take(root)表示选择 root 作为输入的开始，并进行下面的操作。select(1,rack)表示从 root 的子节点中选择一个 rack，如果 root 的子节点不是 rack，则进行迭代，直至选择出一个 rack 并作为下一步的输入。select(3,host)以上一步的选择结果作为输入，在上一步选择的 rack 的子节点中进行迭代选择出三个不同的 host 节点，选择完成后将这一步选择的结果作为下一步的输入。select(1,disk)从上一步选择出的三个 host 节点的子节点中各选择一个 disk，并进行下一步。emit 输出上一步选择出的三个 disk。

表 2.3 挑选副本 rules 与对应结果

Action	Result
take(root)	root
select(1,rack)	row1
select(3,host)	host1 host2 host3
select(1,disk)	osd1 osd3 osd4
emit	

2.5 PG(Placement Group)

为了防止 Ceph 集群中 OSD 的变动造成整个集群的数据迁移，Ceph 引入了 PG[50] 这一概念，PG 是一个逻辑上的组，每一个 PG 都有一个唯一的标识 PGId。PG 是 Ceph 最核心和复杂的概念之一[51]，PG 位于 RADOS 的中部，负责接受和处理来自客户端的请求，PG 可以在 OSD 之间进行自由的数据迁移，这也是 Ceph 的自治和自我恢复功能实现的基础。同时 PG 也包含了完整的副本关系，比如将对象的副本分布到 PG 下的三个 OSD 设备。

PG 的常见的状态有以下几种：

Active：当前拥有最新状态数据的 PG 正在工作中，能正常处理来自客户端的读写请求。

Inactive：正在等待具有最新数据的 OSD 出现，即当前具有最新数据的 PG 不在工作中，不能正常处理来自客户端的读写请求。

Clean：PG 所包含的对象达到指定的副本数量，即对象副本数量正常。

Unclean：PG 所包含的对象没有达到指定的副本数量，比如一个 PG 没在工作，另一个 PG 在工作，对象没有复制到另外一个 PG 中。

Peering：PG 所在的 OSD 对 PG 中的对象的状态达成一个共识（维持对象一致性）。

Degraded：主 OSD 没有收到副 OSD 的写完成应答，比如某个 OSD 处于 Down 状态。

Stale：主 OSD 未在规定时间内向 Monitor 报告其 PG 状态，或者其它 OSD 向 Monitor 报告该主 OSD 无法通信。

Inconsistent：PG 中存在某些对象的各个副本的数据不一致，原因可能是数据被修改。

Repair：PG 在 scrub 过程中发现某些对象不一致，尝试自动修复。

Undersized：PG 的副本数少于 PG 所在池所指定的副本数量，一般是由于 OSD 是 Down 状态的缘故。

Scrubbing(deep + Scrubbing)：PG 对对象的一致性进行扫描。

Recovering：PG 间 peering 完成后，对 PG 中不一致的对象执行同步或修复，一般是 OSD 是 Down 状态或新加入了 OSD。

Backfilling：一般是新的 OSD 加入或移除掉了某个 OSD 后，PG 进行迁移或进行全量同步。

Down：包含必备数据的副本挂了，PG 此时处于离线状态，不能正常处理来自客户端的读写请求。

2.6 读写流程分析

Ceph 在进行文件写入时，首先通过 Ceph 的客户端调用应用接口层，再通过应用接口层调用基础库，最终完成对 RADOS 的调用。调用 RADOS 后将通过三次映射完成对象与一组存储节点的对应，具体如图 2.5 所示，首先将文件 File 按照一定的大小（默认为 4M）分割为一个个的对象，特别注意最后一块不够 4M 的也分为一个对象，并为每一个对象赋予一个唯一的 ObjectId，到此已完成第一次映射过程。接着通过对每一个对象的 ObjectId 进行哈希计算后与 PG 的数目减一，然后进行与运算，得到每一个对象的 PGId，从而达到为每一个对象对应一个 PG。这里使用哈希计算是为了保证数据在各个 PG 之间的均匀分布。对象数据在各个 PG 之间的均匀分布是为了维护集群的负载均衡。最后将通过 CRUSH 算法完成对象与 OSD 之间的对应关系，客户端将从 Monitor 拉取集群最新的 Map 图，然后通过 CRUSH 算法按照 Placement Rules 选择符合要求的指定数量的一组 OSD 节点，例如 OSD 不能是 failed、overloaded 的，这组 OSD 节点的第一个 OSD 为主副本节点，其余 OSD 为从副本节点，至此三次映射结束。

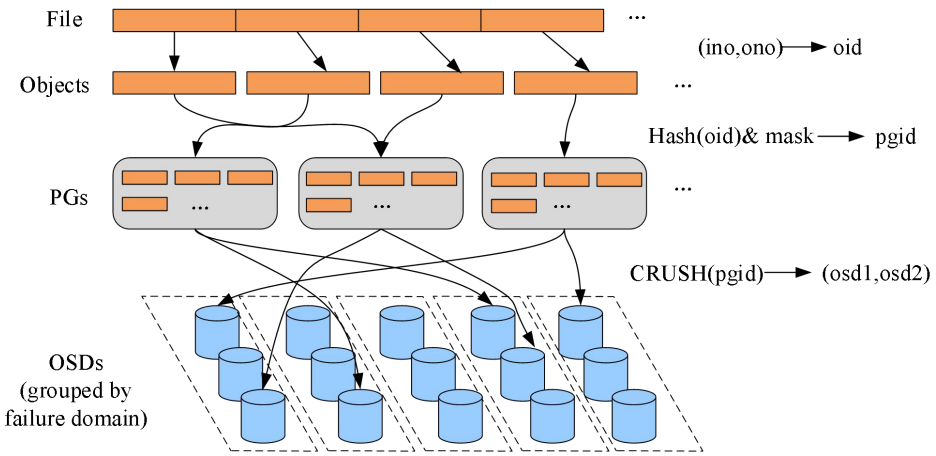

图 2.5 数据分布映射图

在完成三次映射后，找到每一个对象数据映射的一组 OSD 集合，客户端将对象发送给这组 OSD 集合中的第一个 OSD，也就是主副本节点进行写操作，并由主副本节点将对象发送给另外两个 OSD，也就是从副本节点进行写操作。当从副本节点写入完成后回复主副本节点，主副本节点在接到所有从副本节点写入完成的消息并且自身也写入完成后，则回复客户端对象写入完成。文件写入详细过程如图 2.6 所示：

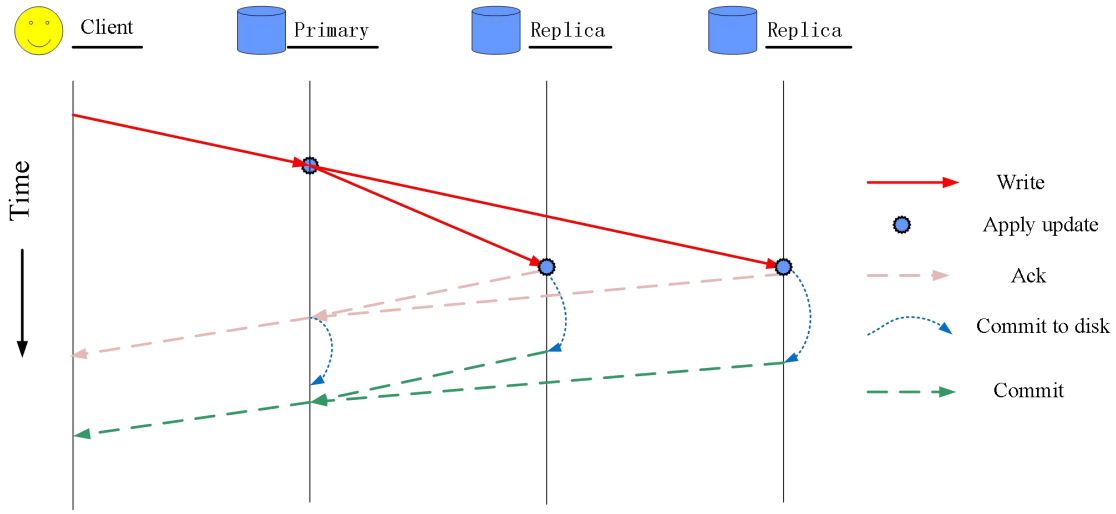

图 2.6 文件写入过程

(1). 用户调用 Ceph 的 APP 层，通过调用应用接口层的对象存储网关 RADOSG、块存储接口 RBD 或者文件存储接口 CEPHFS；

(2). 应用接口层通过下层的基础库调用基础存储系统 RADOS；

(3). 客户端通过 RADOS 中的 Monitor 获取到集群的 Map 图；

(4). 客户端调用 RADOS 将对象进行三次映射，先将对象按照特定的 size 切分为若干个对象，并为每一个切分后的对象生成一个 ObjectId，然后通过一定的计算得到每一个对象对应的 PGID，最后通过 CRUSH 算法为每一个对象得到一组 OSD 集合；

(5). 客户端将第一个 OSD 作为主副本节点，同时将对象发送给主副本节点；

(6). 主副本节点将对象数据发送给 OSD 集合中其他的 OSD 节点，当所有的 OSD 接收到写入对象后，主副本节点返回客户端响应 Ack；

(7). 当主副本节点接收到所有 OSD 写入磁盘完成的消息后，同时主副本节点中对象也写入磁盘完成时，向客户端返回写入成功 Commit 通知，当客户端接收到来自主副本节点的 Commit 通知后，将缓存的数据清除；

(8). 默认地，从 Ceph 客户端接收到来自主副本节点的 Ack 响应到收到集群的 Commit 通知期间，如果此阶段内任何一个节点未写入成功，则由客户端重新进行一次写入过程。

Ceph 在读取已经写入完成的文件时，过程与文件的写入过程极其相似，同样要经过三次映射。首先通过第一次映射拿到文件对应的每一个对象的 ObjectId，然后通过第二次的哈希计算得到每一个 ObjectId 对应的 PG，最后通过 CRUSH 算法得到每一个对象对应的 OSD 列表，然后每一个对象通过与其对应的 OSD 列表中的第一个 OSD(即主副本节点)进行通信以完成对象的读取操作，最后再将已经读取的对象拼接起来得到目标文件，至此完成了文件的读取操作。

2.7 本章小结

本章首先介绍了 Ceph 整体的逻辑架构，接着详细的介绍了 Ceph 相关技术和概念，最后详细的介绍了 Ceph 的读写机制，为本文接下来的研究提供理论基础。

第 3 章 Ceph 读写模型优化和异构感知策略

由于 Ceph 采用强一致性的写入策略，导致集群出现较高的读写延迟，且异构存储环境下，没有充分利用副本节点的 I/O 性能。本章将针对以上问题进行研究并提出优化方案，以求能够有效地提高集群的读写效率和发挥异构副本的性能特点。

3.1 问题的提出

在 SSD 和 HDD 构成的混合异构存储结构下，Ceph 现有的存取机制将不能发挥不同介质存储设备的性能特点。如图 3.1 所示，集群在对象存储过程中，并不区分 SSD 和 HDD 存储介质。

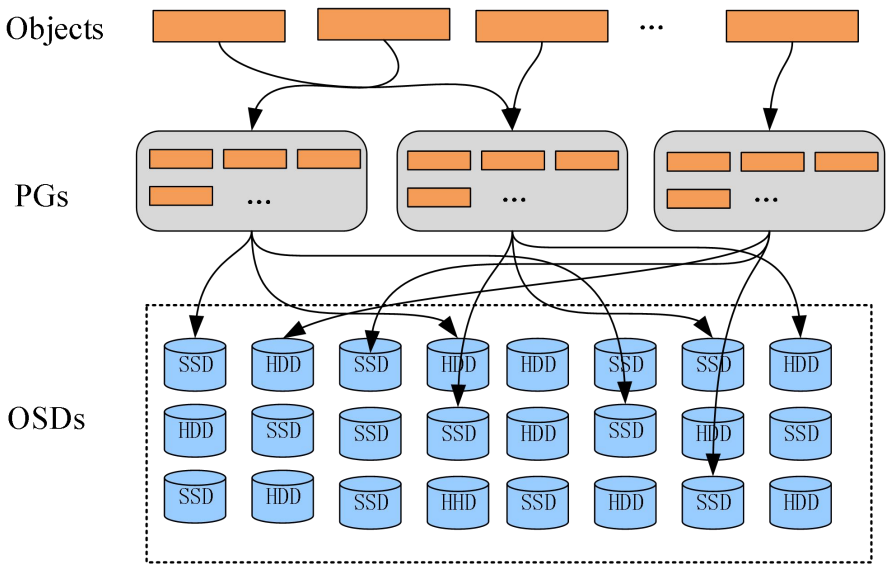

图 3.1 在 SSD 与 HDD 异构环境下对象存储过程

Ceph 在写入文件数据时，会通过三次映射过程将对象数据映射到一组 OSD。首先将文件 File 按照一定的大小（默认为 4M）分割为一个个的对象，特别注意最后一块不够 4M 的也分为一个对象，并赋给每个对象一个唯一的标识，接着通过一定的算法将这些对象映射到唯一的 PG 中去，最后通过 CRUSH 算法为每一个对象数据得到一组 OSD 集合。在找到每一个对象数据映射的一组 OSD 集合后，客户端将对象发送给这组 OSD 集合中的第一个 OSD，也就是主副本节点进行写操作，并由主副本节点将对象数据发送

给其他的 OSD，也就是从副本节点进行写操作。当从副本节点写入完成后回复主副本节点，主副本节点在接到所有从副本节点写入完成的消息并且自身也写入完成后，则回复客户端对象写入完成。通过上述对文件写入流程的分析可以发现，Ceph 采用的是副本的强一致性的模型。该策略下，用户的一次写请求，主、从副本之间采用了同步写的方式。

通过第 2.6 节的分析可知，当 Ceph 使用多副本来保证数据的安全性时，采用的是强一致性的策略来保证同一个对象的各个副本节点数据一致性。客户端首先通过三次映射为对象映射到一组 OSD 节点，这组 OSD 节点中的第一个节点为主副本节点，其余节点为从副本节点，然后客户端将对象发送给主副本节点，再由主副本节点将对象数据分发给所有的从副本节点来完成写入过程，当所有的副本节点都将对象持久化到磁盘后，才由主副本节点回复给客户端 Commit 通知，而在此期间客户端不能进行其他的操作，这极大地限制了 Ceph 的写效率从而造成较高的写延迟。

Ceph 在设计之初为了让存储容量大的节点存储更多的数据，让容量小的节点存储更少的数据，引入了权重的概念，并让权重作为 CRUSH 算法的一个参数，并以权重值决定存储节点被选中的概率，从而让权重大的节点存储更多的对象，权重小的存储节点存储更少的对象，最终达到对象在整个集群的合理分布。然而，在存储节点异构情况下，以存储设备容量作为权重的方法忽视了异构存储设备的性能差异，因而在存储设备异构环境下，现有策略将不能有效地发挥异构存储设备的性能特点。

进一步地，Ceph 在读取对象时，默认只从主副本节点中读取对象，当有大量并发读时会造成主副本节点的 I/O 负载过大，而从副本节点的 I/O 性能却没有得到充分利用，进而降低整个集群的读性能。

3.2 副本弱一致性模型

针对 Ceph 在写操作时采用的强一致性策略造成集群写延迟较高的问题，本节提出了一种副本弱一致性模型，从而优化 Ceph 的写效率。

通过对 Ceph 采用的强一致性策略的写过程分析发现，Ceph 为了提高系统的安全性，只有对象在所有的副本节点中都写入完成，也就是写入磁盘成功后，主副本节点才会返回客户端 Commit 通知。但是所有副本节点中对象的写入并不是同时开始的，通常，从副本节点的对象写入开始时间要晚于主副本节点的写入， 因此主副本节点中对象的写

入完成是要早于从副本节点的写入完成。而在所有的副本节点中的对象写入完成过程中，为了防止各个副本节点因故障造成对象无法写入的问题，客户端需要等待来自主副本节点的通知，以保证各个副本节点的正常写入，也就是说，当有某一个副本节点出现写入失败时，客户端将会重新提交写操作。

实际上，客户端提出写请求，不必等待所有副本写入成功后再接收来自主副本节点的通知，而是在主副本节点对象写入过程完成后，且主副本收到所有从副本节点返回的 Ack 时，主副本节点即返回给客户端 Commit 通知，并由主副本节点来负责同步所有从副本节点的写操作。如此，对于客户端与副本节点而言，在保证数据正确提交写的同时，实现了副本间的异步写，实际上，就是采用了弱一致性模型的写策略，进而优化了整个存储集群的写效率，具体写入过程如图 3.2 所示。详细步骤如下：

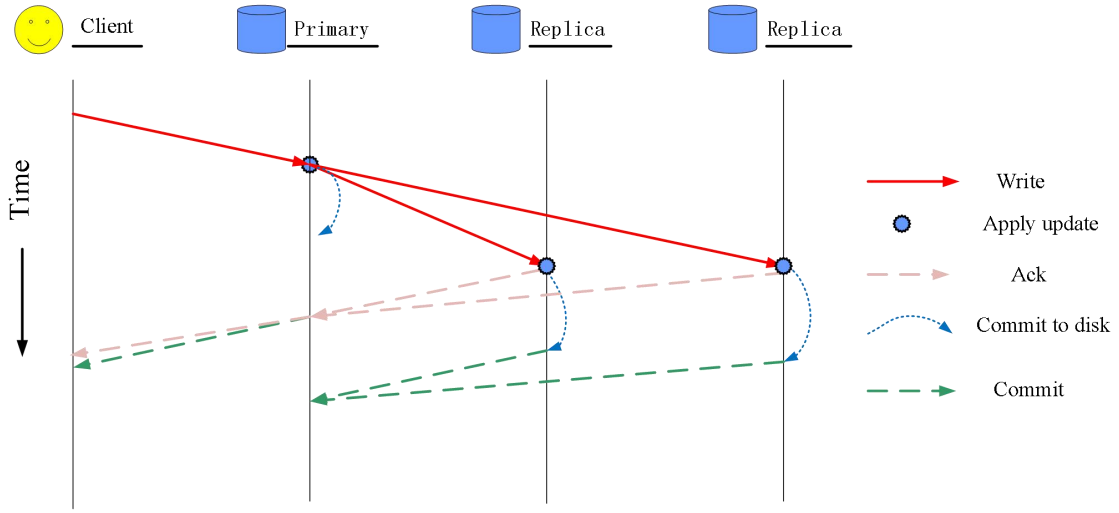

图 3.2 基于副本弱一致性模型的写策略

(1). 用户调用 Ceph 的 APP 层，通过调用应用接口层的对象存储网关 RADOSG、块存储接口 RBD 或者文件存储接口 CEPH FS；

(2). 应用接口层通过下层的基础库调用基础存储系统 RADOS；

(3). 客户端通过 RADOS 中的 Monitor 获取到集群的 Map 图；

(4). 客户端调用 RADOS 将对象通过三次映射，先将对象按照特定的 size 切分为若干个对象（Ceph 默认的 size 大小为 4M），然后通过一定的计算得到每一个切分后对象的 PGId，最后通过 CRUSH 算法为每一个对象得到一组 OSD 集合；

(5). 客户端将第一个 OSD 作为主副本节点，OSD 集合中其他的 OSD 为从副本节点，同时将对象发送给主副本节点；

(6). 主副本节点在接收到对象数据后进行写入磁盘操作，同时由主副本节点给 OSD 集合中所有从副本节点发送对象数据，并进行写入磁盘操作，当所有的从副本节点接收到写入对象后，返回给主副本节点 Ack 响应，并在副本节点中将该对象的标识写入待写入列表（用来判断对象在从副本节点中是否写入磁盘成功）；

(7). 主副本节点中对象数据写入磁盘成功，并且主副本节点接收到所有的从副本节点的 Ack 响应后，由主副本节点返回给客户端 Commit 通知，客户端在接收到来自主副本节点的 Commit 通知后，将缓存的数据清除；

(8). 当所有的从副本节点写入磁盘成功后，返回给主副本节点 Commit 通知，并删除从副本节点中待写入列表中该对象的标识；

(9). 在主副本节点返回给客户端 Commit 通知到主副本节点收到所有从副本节点的写入磁盘通知期间，如果此阶段内任何一个从副本节点未写入成功则由主副本节点重新向从副本节点发送写入过程，从而确保从副本节点中对象的写入成功；

特别地，在主副本节点返回给客户端 Commit 通知到所有从副本节点对象写入磁盘过程中，如果主副本节点崩溃，将造成一致性问题，针对此问题参考 Liao Jianwei[52]提出的恢复数据一致性的方法，提出如下处理方案。

首先，在 Monitor 添加缓存日志文件，当客户端向主副本节点写对象时，首先要写到 Monitor 日志文件中，只有日志写入成功且主副本节点中该对象写入磁盘完成后，主副本节点再返回给客户端 Commit 通知。因此，在所有从副本节点还没有完成对象写入磁盘时，如果主副本节点崩溃，存在以下两种情形：

第一种情形：主副本节点在返回给客户端 Commit 通知后崩溃，此时存在从副本节点中该对象写入磁盘成功。图 3.3 描述了这种情况，主副本节点 OSD1 在返回 Commit 通知后崩溃，而此时从副本节点 OSD2 列表显示对象 Object1 没有写入成功，而 OSD3 写入成功。当 Monitor 检测到 OSD1 崩溃后（心跳检测），将会选择一个新的主副本节点 OSD4 加入副本集群。Monitor 利用 OSD3 中写入成功的对象来完成 OSD4 中对象数据的写入，并由 OSD4 将对象同步给 OSD2，从而保证所有副本节点的数据一致性。

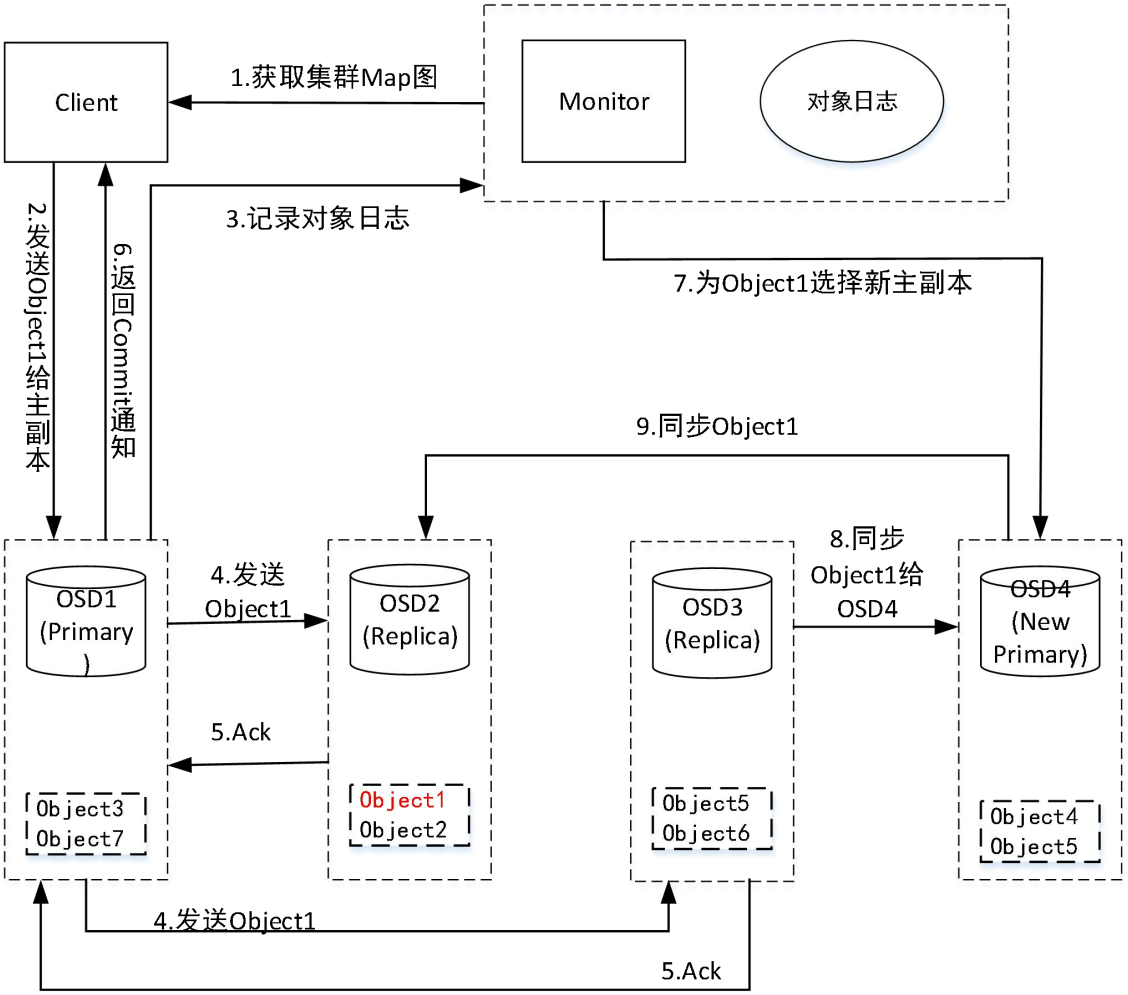

图 3.3 存在从副本节点写入成功条件下对象恢复过程

第二种情形：主副本节点在返回给客户端 Commit 通知后崩溃，此时所有从副本节点中该对象均未写入成功。如图 3.4 所示，主副本节点 OSD1 崩溃后，从副本节点 OSD2、OSD3 中对象 Object1 均未写入成功，在 Monitor 节点检测到 OSD1 崩溃后，选择一个新的主副本节点 OSD4，接着根据 Monitor 节点中日志文件记录向 OSD4 写入，再由 OSD4 同步对象 Object1 到所有从副本节点，从而保证所有副本节点数据一致性。

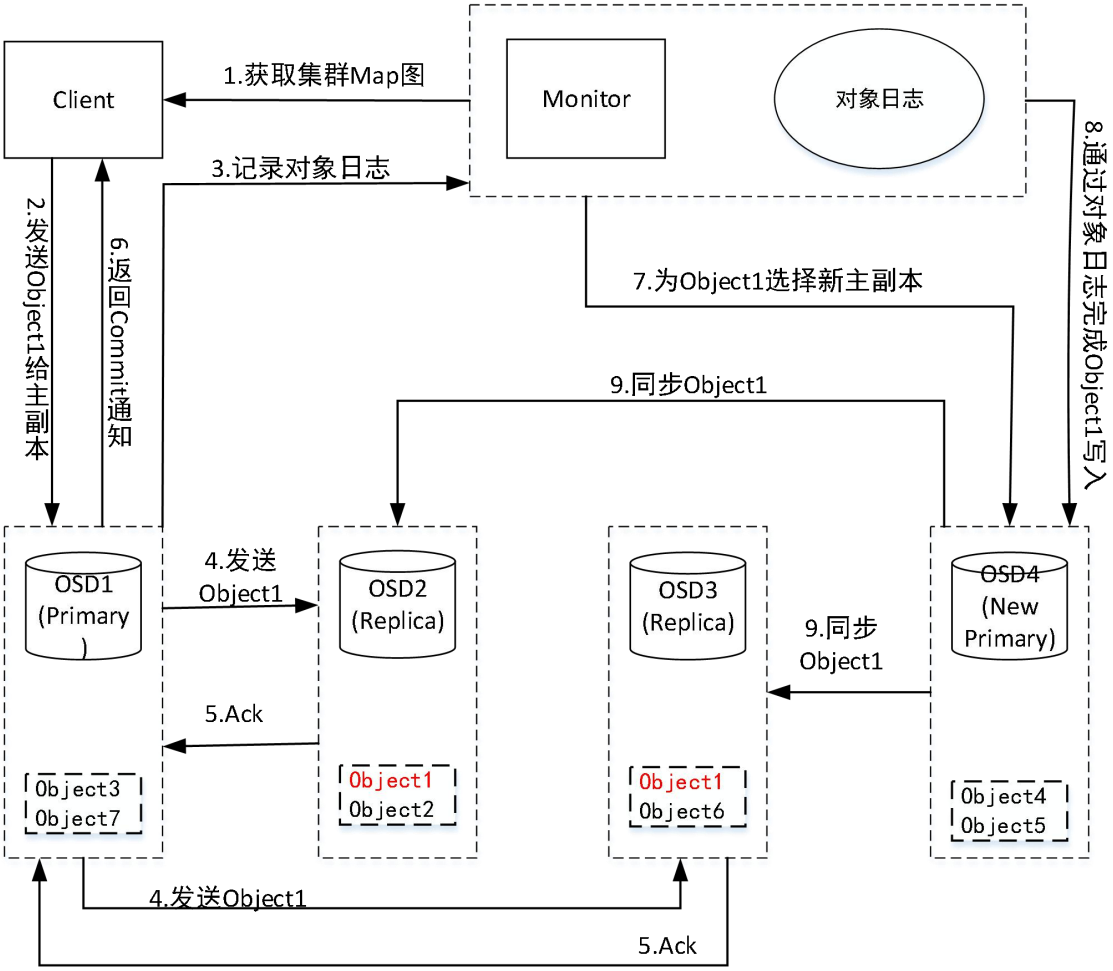

图 3.4 从副本节点均未写入成功条件下对象恢复过程

3.3 多副本读优化策略

为了充分发挥从副本节点的 I/O 性能以及降低主副本节点的 I/O 压力，同时考虑到 Ceph 在读取对象时的读效率与存储节点的 CPU、内存、硬盘以及客户端与存储节点的距离密切相关，也就是说如果一个存储节点的 CPU、内存、硬盘的性能越高，同时与客户端的距离越近，当读取该存储节点中对象时，读效率将会越高。

考虑以上，本文提出了一种多副本读优化策略，在充分考虑每一个存储节点的 CPU、内存、硬盘以及距离的基础上，通过综合权重来决定提供读取对象数据的节点，同时在考虑各副本节点性能基础之上，引入随机因子选择读取副本节点，以解决密集的并发读瓶颈问题。通过监控工具可以得到集群中每一个存储节点的 CPU、内存、硬盘性能，集群节点的拓扑距离，最后通过公式(3.1)得到每一个存储节点的综合权重。

$$P_i = w_1 \frac{Dis\tan ce_{\max} - Dis\tan ce_i}{Dis\tan ce_{\max}} + w_2 \frac{Speed_i}{Speed_{\max}} + w_3 (1\text{-}CPU_i) + w_4 \frac{1 - Mem_i}{1 - Mem_{\min}} \quad (3.1)$$

在公式(3.1)中，$Speed_i$ 表示第 i 个存储节点的硬盘读取速度，CPU_i 表示第 i 个存储节点的 *CPU* 利用率，Mem_i 表示第 i 个存储节点的内存是利用率，$Mem_{\min}$ 表示整个集群所有节点中最小的内存利用率，$Speed_{\max}$ 表示集群中所有磁盘中最大的读取速度，$Dis\tan ce_i$ 为客户端到第 i 个存储节点的拓扑距离，$Dis\tan ce_{\max}$ 表示最大的拓扑距离，w_1、w_2、w_3 和 w_4 表示每一个性能指标相应的权系数，所有 w_i 相加的和为 1，其具体取值根据不同的实际需求进行设定，比如整个集群是 I/O 密集型的，可以将 w_2 适当取大，从而提高 I/O 性能在综合权重中的重要性，再比如整个集群是计算密集型的此时 CPU 的性能高低将对节点有极大的影响，此时就可以将 w_3 适当取大，从而提高 CPU 性能在综合权重中的重要性，本文这里将 w_1、w_2、w_3 和 w_4 均取 0.25。当其他性能一样时，节点的 $Dis\tan ce$ 越小将导致 $Dis\tan ce_{\max} - Dis\tan ce_i$ 将越大，然后乘以 w_1 后的综合权重值越大，同样当内存利用率越低，综合权重也将越大；当其他条件一样时，CPU 利用率越小，综合权重越大；当其他条件一样时，磁盘读取速度越高，综合权重将越大。下面举一个例子进行分析，一个集群中最大的拓扑距离 $Dis\tan ce_{\max}$ 为 3，当前存储节点与客户端的距离 $Dis\tan ce$ 也为 3，整个集群中所有节点的最大读取速率为 480M/S，当前节点磁盘读取速率为 120M/S，当前节点的 CPU 使用率为 40%，内存使用率为 20%，集群中最小的内存使用率为 0，则该节点的综合权重 $P = w_1 \frac{5-3}{5} + w_2 \frac{120}{480} + w_3 (1\text{-}40\%) + w_4 \frac{1-20\%}{1}$，其中 w_1、w_2、w_3 和 w_4 均取 0.25，所以该节点的综合权重为 0.5125。

当出现高并发的读操作时，会出现并发读操作都命中在同一个存储节点的情形，从而造成对读效率的降低。需要采用一种机制既能让综合性能高的节点命中率依然高，但是又能具有随机访问的性质，故而想到了随机数的方法。所以，进一步使用一个随机数乘以综合权重值。

$$Decision_i = P_i * Random_i \quad (3.2)$$

在公式(3.2)中，P_i 是节点 i 的综合权重值，$Random_i$ 是一个随机数，通过这两者的乘积得到节点 i 的 *Decision*，最后让该对象的所有副本节点中 *Decision* 最大的节点来提供读取服务。

由于本文把 Ceph 的写策略由强一致性模型优化为副本弱一致性模型，所以在读取对象数据时，存在主副本写入磁盘完成而从副本节点写入磁盘过程还没有完成的情况。虽然这种情况发生的概率极低，但是本文也必须考虑此种情况的发生。故而在读取对象时，会先判断对象是否在所有从副本节点中写入磁盘完成，如果对象在所有从副本中写入磁盘完成，则通过以上的模型选择综合性能最大的节点来提供读取服务；若对象只在主副本节点写入磁盘完成而在从副本节点上该对象写入磁盘过程还在继续，则由主副本节点来提供对象读取服务。具体算法如表 3.1 读取副本节点选择算法所示：

表 3.1 读取副本节点选择算法

Input: *object ID* //object ID 对象 id
Output: *osd ID*//输出读取对象节点

1: client 从 Monitor 获取最新的 *cluster map*//获取最新的集群 map
2: *osds*=CURSH(*object ID*,*cluster map*) //通过 CRUSH 计算对象存放的所有副本节点
3: **if** *writeOk* **then**//所有从副本节点中对象写入完成
4: 通过公式(3.1)式计算每一个节点的综合权重
5: 通过公式(3.2)式将每一个节点的综合权重乘以随机数
6: **return** *Decision* 最大的 osd;//返回 Decision 最大的 osd 作为读取节点
7: **else**
8: **return** 主副本 osd;//返回主副本节点
9: **end if**

(1). 客户端先通过 Monitor 获取最新的集群 Map 图；

(2). 通过 CRUSH 伪随机算法计算得到对象所存储在的一组 OSD 集合；

(3). 将 OSD 集合中第一个 OSD 作为主副本节点，其他为从副本节点，判断对象数据是否在所有从副本节点写入磁盘完成，若所有从副本节点中对象数据写入磁盘完成则进入下一步，否则跳到第(7)步；

(4). 通过收集工具得到集群中每一个节点的 CPU、内存的利用率，以及磁盘的读取速度和拓扑距离，然后通过公式(3.1)获取每一个存储节点的综合权重；

(5). 将每一个节点的综合权重值乘以一个随机数从而得到每一个节点的 *Decision*，以防止对该对象的密集访问造成整个集群的综合性能变化过快，以至于综合性能的变化出现延迟，从而造成综合性能出现误差，故而需乘以随机数；

(6). 让 *Decision* 最大的节点来通过对象读取对象服务；

(7). 如果在读取对象时，存在从副本节点中该对象数据还没有写入磁盘完成，则用主副本节点提供该对象的读取服务。

3.4 基于副本弱一致性的组合存储优化策略

表 3.2 组合优化存储算法

Input: *object ID ,N,Tries* //object ID 对象 id,N 为需要挑选的副本数,Tries 挑选每个节点允许的挑选次数

Output: *osds[]*//输出结果集

```
1: for rep=0;rep<N;rep++ do//需要挑选 N 个副本
2:     tries=0,reject=false;//tries 为本副本尝试次数，reject 为此处挑选是否符合要求
3:     repeat
4:         i=set(); //设置随机数
5:         item = crush_bucket_choose() //从当前 bucket 中挑选
6:         if item!=osd then    //判断 item 是否为 osd
7:             bucket=item; //将  bucket 设置为 item
8:         end if
9:     util(item != osd)
10:    if collide||reject then    //判断冲突和过载
11:        reject=true;    //此处 item 不符合要求
12:        tries++;    //尝试次数加一
13:    end if
14:    if rep==0&&reject==false then    //第一个副本并且没有冲突和过载
15:        if item.type==hdd then      //为 hdd 时不加入
16:            reject=true;
17:            tires++;
18         end if
19:    end if
20:    if tries==Tries then //达到本副本的最大尝试次数
21:        countine; //跳出本次循环
22:    end if
23:    if(reject==false)then    //本副本符合要求
25:        osds[]<- item;    //将 item 加入 osds
26:    end if
27: end for
28: return osds;
```

在副本弱一致性策略的基础之上，进一步考虑由 SSD 和 HDD 构成的异构存储结构，充分利用 SSD 的存储性能，本节进一步提出了基于弱一致性的副本组合存储策略。通过第 3.3 节分析可以知道，主副本节点在收到所有从副本节点的 Ack 响应，同时对象在主副本节点写入磁盘完成后，则返回给客户端 Commit 通知。也就是说对象的写延迟主要取决于主副本节点中对象的写入磁盘时间，同时，由于 SSD 的性能远远高于 HDD，

为了有效地降低主副本节点的写延迟和利用集群的异构特点，将对象的主副本节点设置为SSD，从而有效发挥SSD的性能特点以达到提高整个集群写效率的作用。

具体实现过程如下：通过修改Ceph的数据映射算法CRUSH，采用组合优化方法，当为一个对象挑选出第一个存储节点OSD时，判断该OSD的存储介质是否为SSD，若为SSD则进行第二个节点的选取，否则丢弃该节点重新选取第一个节点直至选取一个存储介质为SSD的OSD节点。在选取第二、第三个存储节点时则不需要进行存储节点的类型判断。通过上述的节点选取方式能够确保选取的主副本节点为SSD，通过本文提出的弱一致性模型可知，主节点的对象存储决定了对象的写效率，通过将主副本节点设置为SSD，以求充分降低写延迟和发挥异构副本的性能优势。修改后的算法如表3.2所示。

3.5 实验分析

本节对本章提出的读写优化模型和异构感知策略分别进行了五组实验。

第一组实验测试Ceph在原生策略和本文策略下写入不同大小对象的写延迟，从而得到本文策略在降低集群写延迟的有效性。

第二组实验测试原生策略和本文策略在读取对象不同次数的情况下，SSD的命中次数，从而得到本文策略在发挥SSD性能上的优势。

第三组实验测试在写入不同大小对象的情况下原生策略和本文策略中集群的写吞吐量。

第四组实验测试原生策略和本文策略下顺序读取不同大小对象的读吞吐量，从而得到本文策略在提高集群读吞吐量方面的作用。

第五组实验测试在原生策略和本文策略下读取不同大小对象的随机读吞吐量。

3.5.1 实验环境及测试工具

实验环境：四台虚拟机，每台虚拟机分配512M内存，20G存储空间，其中一台虚拟机装1个Monitor节点，剩下每台虚拟机装1个SSD的OSD节点，2个HDD的OSD节点。

测试工具：使用Ceph自带的rados bench，该工具的语法为：rados bench -p <pool_name> <seconds> <write|seq|rand> -b <block size> -t --no-cleanup

针对 Ceph 原生策略和本文策略，实验对读写效率和 SSD 的命中率进行测试。在写方面，将 Ceph 原生的强一致性策略与本文弱一致性策略和组合存储优化进行对比；在读方面，将 Ceph 原生的读策略与本文提出的多副本读优化策略进行对比。

3.5.2 实验结果分析

实验一 Ceph 在原生策略和本文策略下写入不同大小对象的写延迟对比

第一组实验通过测试工具 rados bench 测试 Ceph 在原生策略和本文策略下写入不同大小对象的写延迟。在不同大小对象情况下，原生 3 副本、原生单副本和本文策略下写延迟对比如下图 3.5 所示，可以发现本文方法对写延迟有大幅度的降低，当对象大小为 32M 时，本文方法与原生 3 副本相比写延迟降低十倍，与单副本相比写延迟也降低了四倍以上。

通过实验结果可以清晰的发现本文的方法相对 Ceph 原生 3 副本和单副本在写效率上均有较大的提高。这是因为 Ceph 原生 3 副本写入策略是在对象发送给主副本节点后再由主副本节点将对象发送给从副本节点，在主副本和所有从副本都写入完成后，才响应客户端，所以造成较大的写延迟。但是通过实验数据发现本文的方法相比单副本有显著的写性能优势，这是因为本文的方法不仅采用了副本弱一致性模型，也采用了基于副本弱一致性模型的组合存储策略，进一步改善了主副本节点的存储性能，从而极大地降低了写延迟，所以本文策略相比单副本也有很大的写性能优势。

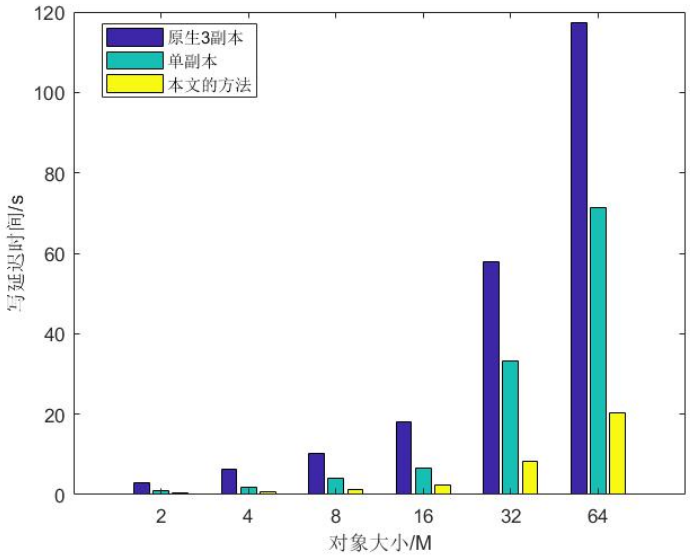

图 3.5 不同策略下不同大小对象的写延迟

实验二 测试原生策略和本文策略下读取对象不同次数时 SSD 的命中次数

第二组实验测试原生策略和本文策略下读取对象不同次数时 SSD 的命中次数。原生策略和本文策略下的 SSD 命中次数如图 3.6 所示，可以发现本文方法在读对象时，SSD 存储节点命中率相比原生的提高两倍以上。

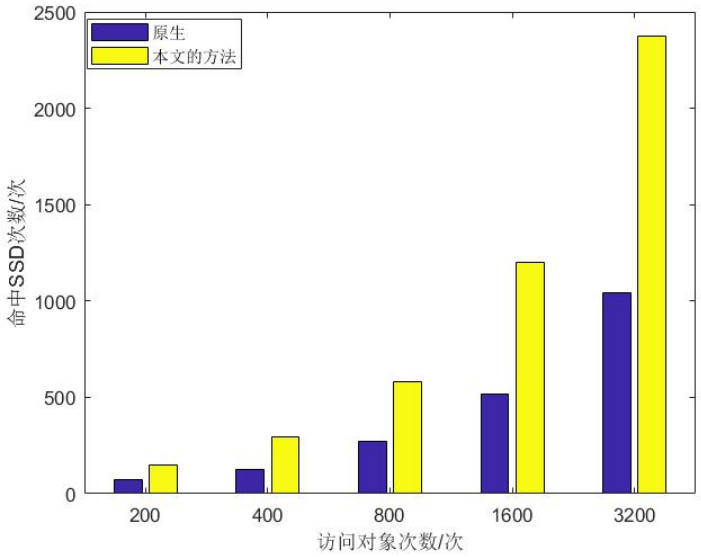

图 3.6 不同读取次数下的 SSD 的命中次数

通过对比原生策略和本文策略下读取对象不同次数时 SSD 的命中次数，可以清晰的发现本文策略显著的挺高了 SSD 存储节点的命中率。这是由于本文提出的基于副本弱一致性模型的组合存储策略将对象数据的主副本节点设置为 SSD，同时，考虑了存储节点的综合性能来分布 I/O 请求，相比 HDD 存储节点 SSD 存储节点具有更高的磁盘性能，故而 SSD 存储节点具有更高的综合性能，而在 Ceph 的原生读取策略下是不区分 SSD 和 HDD 的，只把存储空间的大小作为挑选的权重，所以在本文的方法中 SSD 的命中率显著提高。

实验三 测试在写入不同大小对象的情况下原生策略和本文策略中集群的写吞吐量

第三组实验通过 rados bench 测试工具测试在写入不同大小对象的情况下原生策略和本文策略中集群的写吞吐量。原生 3 副本、原生单副本以及本文策略下的写吞吐量如图 3.7 所示，可以发现本文方法对写吞吐量的提高明显，相比原生 3 副本当对象大小小于 16M 时写吞吐量提升 10 倍左右，相比单副本也有显著的提升，随着对象大小的增大逐渐达到了 SSD 副本节点的 I/O 瓶颈，相对性能提升就呈现下降的趋势，但在对象大小

为 64M 时，本文的方法相对原生 3 副本仍然提升 6 倍以上，相对单副本写吞吐量也提升了 4 倍以上。

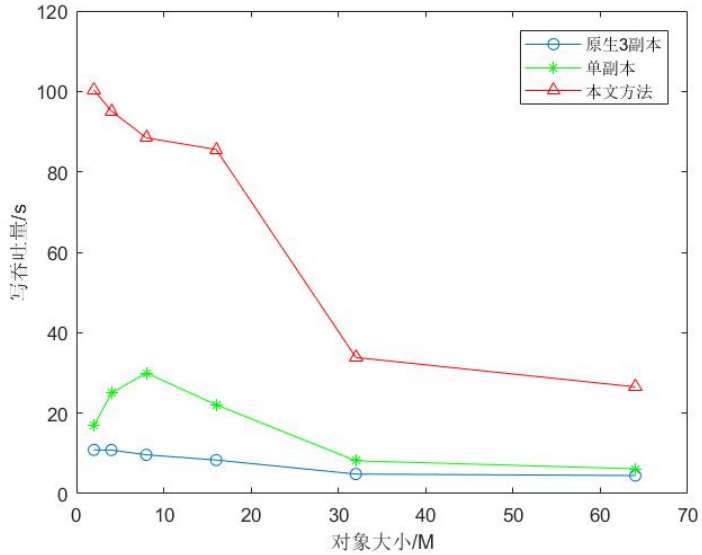

图 3.7 不同策略下不同文件大小的写吞吐量

通过以上实验结果可以清晰的发现本文的方法相比 Ceph 原生 3 副本和单副本在提升写吞吐量上具有显著的优势。这是因为本文一方面采用了副本弱一致性模型，同时又采用了主副本节点为 SSD 的组合存储策略，从而有效地提高写吞吐量，但是通过实验结果可以发现本文的方法相比原生 3 副本和单副本，在写吞吐量上的提升随着写入对象大小的增加而降低，这是因为当对象较大时，达到了 SSD 节点的 I/O 瓶颈，从而造成写吞吐量提升优势的降低。

实验四 测试原生策略和本文策略下顺序读取不同大小对象的读吞吐量

第四组实验通过 rados bench 测试工具测试原生策略和本文策略下顺序读取不同大小对象的读吞吐量。原生 3 副本、单副本和本文策略下的顺序读吞吐量如图 3.8 所示，本文方法相比原生 3 副本和单副本，在对象大小较小时，本文方法的顺序读吞吐量与原生 3 副本和单副本基本保持一致。当对象大小大于等于 16M 时，本文方法在顺序读吞吐量方面优势开始显现，在对象大小为 64M 时相比原生 3 副本和单副本都提升了 3 倍以上。

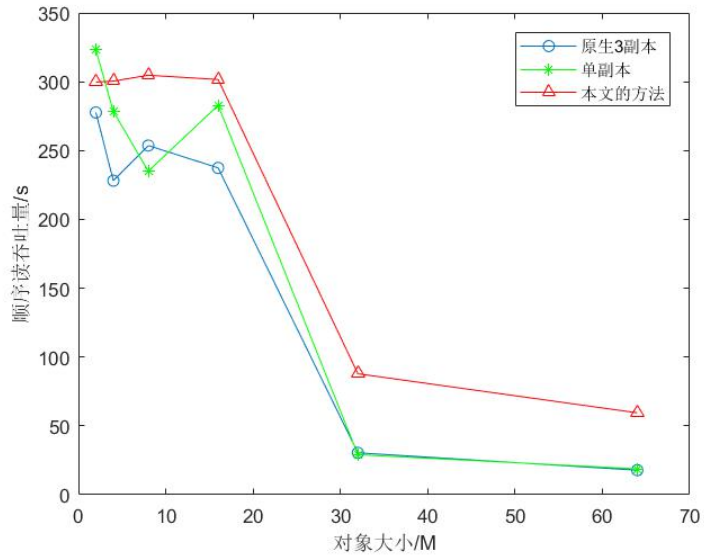

图 3.8 不同策略下不同大小对象的顺序读吞吐量

通过以上实验结果可以发现，在对象大小小于等于 16M 时本文的方法与 Ceph 原生 3 副本以及单副本的写吞吐量基本持平，这是由于当文件大小比较小时，未达到 HDD 的 I/O 瓶颈，所以本文方法与 Ceph 原生 3 副本和单副本的顺序读吞吐量基本保持持平。但随着对象大小的增大，本文的方法相比 Ceph 原生 3 副本和单副本的顺序读吞吐量优势逐渐显现，同时，Ceph 的原生 3 副本在读取对象时只读取主节点，故而在对象大小大于 16M 后，原生 3 副本和单副本的顺序读吞吐量仍然保持一致，而本文的方法采用的是多副本读优化策略，不但会读取从副本，而且还通过分析各个副本节点的综合性能，让综合性能最高的副本节点来提供读服务，故而本文的方法在对象大小大于 16M 后，在 Ceph 原生 3 副本和单副本顺序读时达到 I/O 瓶颈后，呈现出明显的吞吐量优势。

实验五 测试在原生策略和本文策略下读取不同大小文件的随机读吞吐量

第五组实验是测试在原生策略和本文策略下读取不同大小文件的随机读吞吐量，通过 rados bench 测试工具测试结果。在不同策略下的集群随机读吞吐量如图 3.9 所示，在对象大小小于 32M 时，本文的方法在随机读时的吞吐量与原生 3 副本和单副本基本保持一致，这是因为文件较小还未达到 HDD 的 I/O 瓶颈。当对象大小大于等于 32M 时本文的方法优势开始显现，相比原生 3 副本和单副本在随机读吞吐量上提升 3 倍左右。

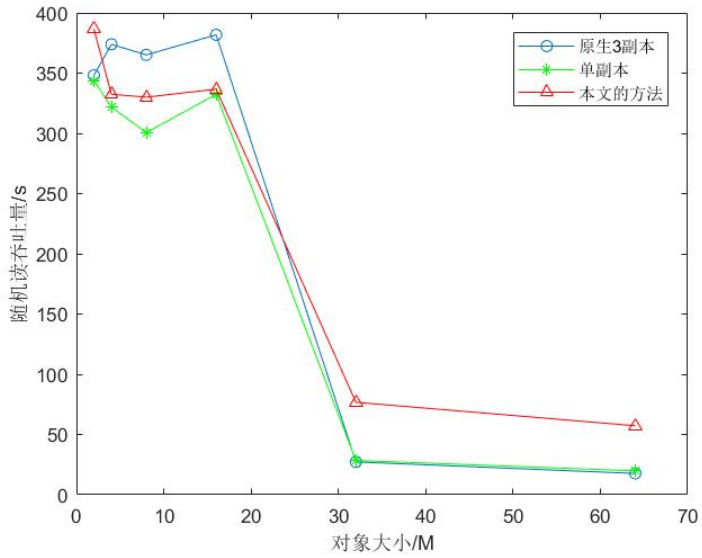

图 3.9 不同策略下不同大小对象的随机读吞吐量

通过以上实验结果可以发现，当对象大小大于等于 32M 时，本文的方法相比 Ceph 原生 3 副本和单副本在随机读吞吐量上的优势清晰的呈现出来。这是因为当对象大小大于等于 32M，随着 Ceph 原生 3 副本和单副本随机读时，提供随机读服务的副本节点达到读瓶颈，从而使随机读吞吐量出现明显的下降，而本文的方法由于采用多副本读优化策略，从而使各个副本节点均有机会提供读取服务，而且还通过分析各个副本节点的综合性能，让综合性能最高的副本节点来提供读取服务，故而本文的方法在随机读吞吐量上有明显的优势。

实验结果表明，异构存储优化机制在 SSD 命中率、写吞吐量、顺序读吞吐量、随机读吞吐量以及降低写延迟方面较原生 3 副本和单副本略而言均有较大程度的改善。

3.6 本章小结

本章针对 Ceph 读写延迟较高和不感知异构问题提出一种副本弱一致性模型和多副本读取策略，在此基础上，进一步给出了基于副本弱一致性的组合存储优化策略，通过修改 CRUSH 算法设置副本的主节点为 SSD 以求充分发挥异构副本的特性的优势。并通过实验证明本文提出方法的有效性。

第 4 章 基于热度划分模型的异构副本组合存储策略

在第三章对组合存储优化策略分析中，只是简单的考虑了主副本为 SSD 还是 HDD 的情形，并未考虑从副本的异构存储问题，所以可能会造成访问次数多的对象存储在性能较低的存储节点中，而访问次数较少的对象则存放在性能高的节点中，接下来将针对此问题进行深入研究，以求充分发挥 Ceph 在异构存储结构下的性能优势。

针对冷热数据的读写性能优化问题，He Shuibing[53]等人提出了可变数量副本机制，这极大的提高了读写效率，但这将会一定程度上降低数据的可靠性和安全性。根据 BackBlaze 发布的硬盘故障率报告发现磁盘在使用 5-6 年后将会有近 50%的损坏[54]，所以副本机制是保证数据安全性和可靠性的重要举措。本章是在确保冷热数据安全性和可用性的基础上，寻找一种优化读写方案，所以本文仍需要采用 Ceph 默认的固定副本机制，即为冷数据创建副本，以提高数据的安全性和可靠性。故而本章在 Ceph 原生三副本机制下，通过利用异构副本的不同组合方式来优化对象数据的读写性能。

4.1 异构副本组合

本节基于分布式系统通常的三副本策略，考虑在 SSD 和 HDD 的异构存储结构下的 Ceph 高效存储策略。由此，本文首先针对 SSD 和 HDD 的异构副本组合方式进行分类，并且在基于第三章中提出的弱一致性和多副本读取策略下，分析每一种异构副本存储组合方式的读写性能特点。

在 SSD 和 HDD 组成的异构存储情形下，基于分布式常用的三副本策略，有四种不同的副本组合方式，如图 4.1 所示。四种组合方式为 3HDD、1SSD+2HDD、2SSD+1HDD、3SSD, 并分别重命名为 A 类、B 类、C 类、D 类。同时考虑到本文采用了弱一致性的写策略，在 1SSD+2HDD 和 2SSD+1HDD 这两种组合方式中 SSD 为主副本节点时写效率最高，故而本文默认这两种组合方式中 SSD 节点为主副本节点。

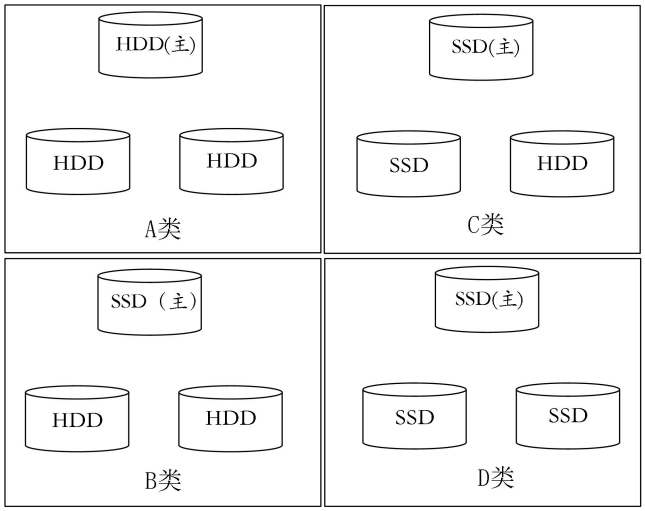

图 4.1 异构副本组合方式分类

下面将从读写效率两方面分析在异构副本情况下，副本的不同组合方式的读写性能特点。

1. 写效率。在第三章提出的弱一致性模型中，客户端将对象发送给主副本节点后，当主副本节点写入完成后，则立即返回写成功的消息给客户端，再由主副本节点将对象更新至副本节点，所以，对象的写延迟取决于对象在主副本节点的写入时间，故在弱一致性模型下，对象的写入延迟取决于主副本节点为 SSD 还是 HDD，在以上的四种副本组合方式中，主副本节点为 HDD 的是 3HDD 组合方式，主副本节点为 SSD 的是 B 类、C 类和 D 类这三种组合方式，也就是说在以 SSD 为主副本节点的三种组合方式在写延迟上是一样的。同时，SSD 的读写性能都是高于 HDD 的[55]，所以可以清楚地得知 A 类组合方式是写延迟最大的一种组合方式。

2. 读效率。在第三章中提出的多副本读取策略下，客户端在读取对象时，根据对象存储的副本节点的综合性能指标来决定从哪个副本节点读取对象，也就是说在读取对象时的读效率跟每个副本节点的性能是密切相关的。所以在 SSD 和 HDD 的异构存储结构下，当对象的所有副本节点中 SSD 节点越多时，该对象的读取效率就会越高，所以如果一个对象的所有副本节点都为 SSD 节点时，则对象的读取效率应该是最高的；相反如果一个对象的所有副本节点都是 HDD 节点，则该对象的读取效率应该是最低的。故而在以上四种副本组合方式中，就读效率而言 SSD 节点越多的组合方式读效率就越高。读效率从低到高依次为 A 类、B 类、C 类、D 类。

4.2 对象热度划分模型

通过 4.1 节的分析，在 SSD 和 HDD 异构存储结构下，将对象的写效率和读效率进行了划分，本节根据对象读写效率的划分引入读写热度，并将读写热度与异构存储的四种副本组合方式相关联，给出了对象读写热度划分模型。

基于分布式系统常用的三副本策略，将副本的组合方式分为 A 类、B 类、C 类和 D 类，并针对这四种异构情况下的副本组合方式进行写效率分析，发现 B 类、C 类和 D 类的写效率是一样的并且都高于 A 类，所以在以上四种不同的组合方式中，将写效率分为两类。在分布式系统中每一个对象都是有访问次数的，将特定周期内对象访问次数多少称为该对象的热度。而对象的访问有读写两种，因此对象的热度又分为读热度和写热度。为了提高对象的写效率，需要将写热度高的对象迁移到写性能好的副本组合方式中去，相应的为了提高对象的读效率需要将读热度高的对象迁移到读性能好的组合方式中去。同时为了防止高性能节点中存放的对象过多从而造成整个集群的负载不均衡，也需要将读写热度低的对象迁移到相应性能低的副本组合方式中去。

对于集群中对象的读写次数使用多版本布隆过滤器[56,57,58]的方法进行收集，布隆过滤器只需要占用两位就可以记录一个对象的访问，而且具有时间复杂低的优点。如图 4.2 所示，假设有 n 个版本的布隆过滤器，也就是说除去当前版本的布隆过滤器外，还有 n-1 个历史版本的布隆过滤器，对象的访问信息都是记录到当前版本中，经过一段时间或者一定数目的请求后，就会将当前版本记录的信息插入到历史版本中，同时将产生一个新的布隆过滤器，如果历史版本达到 n-1 后，将会删除最早版本的过滤器。在布隆过滤器中，每一个对象的访问信息包含两位，并且初始化状态都为 0，其中第一位用来区分读写请求，0 表示写，1 表示读；第二位用来表示是否访问过，0 表示未访问，1 表示访问过。对于读请求，会把读写标志位设为 1，对于写请求，则不可以修改读写标志位，也就是说读请求将覆盖写请求，这是因为在本文提出的异构副本组合方式中，读性能较好的组合都具有较好的写性能。

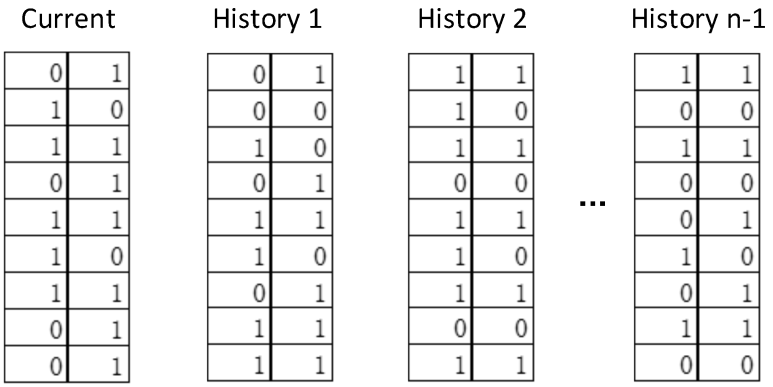

图 4.2 多版本布隆过滤器

1. 写热度。通过上一节的分析将 SSD 和 HDD 异构存储结构的四种组合方式，按照写性能分成了两类，而只有将写热度高的对象存放在写性能好的副本组合方式中才能有效提高写效率，所以本文通过对四种异构副本组合方式的写性能分析得出对象的写热度可以分为两类，分别为写冷（WT1）和写热（WT2），其中 WT1 对应 A 类写性能低的异构副本组合方式，WT2 对应 B 类、C 类、D 类写性能高的异构副本组合方式。如表 4.1 四种副本组合方式与写热度对应关系表所示：

表 4.1 四种副本组合方式与写热度对应关系表

写热度级别	写冷（WT1）	写热（WT2）
副本组合方式	A 类	B 类、C 类、D 类

为了保证各种写热度的对象在整个集群中的均匀分布，以及保证整个集群的负载均衡，同时考虑到本文提出的副本弱一致性模型，故而对象的写延迟主要取决于主副本节点，而主副本节点的写延迟与主副本节点的存储介质密切相关，而本文是在 SSD 和 HDD 的异构环境下，所以主副本节点的选择只有两种，也就是主副本节点为 SSD 或 HDD 这两种。考虑以上 ，本文提出以下写热度划分模型，将集群中 SSD 和 HDD 的比例与热度进行结合，进而保证整个集群的负载均衡。

$$W_{WT1} = \frac{H}{N}(H >= 3, N >= 6) \tag{4.1}$$

$$W_{WT2} = \frac{S}{N}(S >= 3, N >= 6) \tag{4.2}$$

在公式(4.1)和(4.2)中，H 为集群中 HDD 节点的数目，S 为集群中 SSD 节点的数目，N 为集群中所有节点的数目。其中公式(4.1)表示整个集群所有存储对象中写冷的存储对象占比，公式(4.2)表示整个集群所有存储对象中写热的存储对象占比。

写热度划分步骤如下：

(1).收集一定周期内（用户根据实际需要设定）集群中所有对象的写次数；

(2).将对象按照写次数的多少从高到低依次排列，得到对象的写频率排序表；

(3).得到集群中 SSD 节点的个数 S，HDD 节点的数目 H，以及集群中的总节点数 N，其中 S 需大于等于 3，H 需大于等于 3；

(4).通过公式(4.1)和公式(4.2)计算出W_{WT1}和W_{WT2}；

(5).对象的写频率排序表中W_{WT2}比例为写热 WT2，其余为写冷 WT1；

2. 读热度。通过对 SSD 和 HDD 异构存储结构下副本的四种组合方式在读效率方面的分析，四种组合方式的读效率按照性能高低分为了四类。在四种异构副本的组合方式中有读效率高的组合，也有读效率低的组合，而只有将读热度高的对象存放在读性能好的副本组合方式中，才能有效的提高读热度高的对象的读效率。所以本文通过对四种异构副本组合方式的读性能分析，将对象的读热度分为四种，分别为读冷(RT1)、读较热(RT2)、读热(RT3)和读高热(RT4)，其中 RT1 对应 A 类读性能最低的异构副本组合方式，RT2 对应 B 类读性能较低的异构副本组合方式，RT3 对应 C 类读性能较高的异构副本组合方式，RT4 对应 D 类读性能最高的异构副本组合方式。如表 4.2 四种副本组合方式与读热度对应关系表所示：

表 4.2 四种副本组合方式与读热度对应关系表

读热度级别	读冷(RT1)	读较热(RT2)	读热(RT3)	读高热(RT4)
副本组合方式类型	A 类	B 类	C 类	D 类

为了保证各种读热度的对象在整个集群中的均匀分布，保证整个集群的负载均衡，同时考虑到本文提出的多副本读优化策略，在此策略下对象的读效率将于每一个副本节点相关，而本文是在三副本的情况下进行分析的，所以在 SSD 和 HDD 的异构环境下，副本的组合方式有四种，而这四种组合方式每一种的读效率都不同，而且读效率的高低与副本组合方式中的 SSD 的数目密切相关。考虑以上，本文提出以下读热度划分模型，通过在 SSD 和 HDD 异构环境下，不同组合方式的占比，来决定不同热度级别对象的比例，从而保证数据的均匀分布。

$$R_{RT1} = \frac{H}{N}\left(H >= 3, N >= 6\right) \tag{4.3}$$

$$R_{RT2} = \frac{S}{N} * \frac{C_s^1}{C_s^1 + C_S^2 + C_s^3}\left(S >= 3, N >= 6\right) \tag{4.4}$$

$$R_{RT3} = \frac{S}{N} * \frac{C_s^2}{C_s^1 + C_S^2 + C_s^3}\left(S >= 3, N >= 6\right) \tag{4.5}$$

$$R_{RT4} = \frac{S}{N} * \frac{C_s^3}{C_s^1 + C_S^2 + C_s^3}\left(S >= 3, N >= 6\right) \tag{4.6}$$

在公式(4.3)、(4.4)、(4.5)、(4.6)中，H 为集群中 HDD 节点的数目，S 为集群中 SSD 节点的数目，N 为集群中所有节点的数目，C_n^m 是从 n 个不同的元素中，任取 m 个元素的所有组合的个数。其中(4.3)式表明整个集群中有占比 R_{RT1} 的对象的读热度划分为 RT1，(4.4)式表明整个集群中有占比 R_{RT2} 的对象的读热度划分为 RT2，其中(4.5)式表明整个集群中有占比 R_{RT3} 的对象的读热度划分为 RT3，其中(4.6)式表明整个集群中有占比 R_{RT4} 的对象的读热度划分为 RT4。

读热度划分过程如下：

(1). 收集一定周期内（用户根据实际需要设定）集群中对象的读次数；

(2). 将对象按照读次数的多少从高到低依次排列，得到集群中所有对象的读频率的排序表；

(3). 得到集群中 SSD 节点的个数 S，HDD 节点的数目 H，以及集群中的总节点数 N，其中 S 需大于等于 3，H 需大于等于 3；

(4). 通过读热度划分模型计算出 R_{RT1}、R_{RT2}、R_{RT3} 和 R_{RT4}；

(5). 在所有对象的读频率排序表中，从低到高百分比小于等于 R_{RT1} 的读热度级别划分为 RT1，对于大于 R_{RT1} 小于等于 R_{RT2} 的读热度划分为 RT2，对于大于 R_{RT2} 小于等于 R_{RT3} 的读热度划分为 RT3，对于大于 R_{RT3} 的读热度划分为 RT4。

4.3 基于热度划分模型的异构副本组合存储策略

对象进行读写热度的划分后，只有将不同读写热度的对象存储到相应的组合方式中，才能有效地提高整个集群的读写效率，故而本节将对异构存储结构下的四种副本组合方式与读写热度进行有效的结合，以求能够最大效率的发挥不同组合方式的优势。同时通

过上面四种异构副本在读写热度的对应关系，得到读写热度与组合类型的最佳匹配对应关系表，如表 4.3 读写热度与四种副本组合方式最佳匹配对应关系表所示：

表 4.3 读写热度与四种副本组合方式最佳匹配对应关系表

	读冷（RT1）	读较热（RT2）	读热（RT3）	读高热（RT4）
写冷（WT1）	A 类	B 类	C 类	D 类
写热（WT2）	B 类	B 类	C 类	D 类

表 4.4 加入相应副本组合类型判断的 CRSUH 算法

Input: *object ID ,N,Tries* //object ID 对象 id,N 为需要挑选的副本数,Tries 挑选每个节点允许的挑选次数,CombinationType 挑选副本组合类型,为空时不需要挑选特点类型
Output: *osds[]*//输出结果集

```
1: for rep=0;rep<N;rep++ do//需要挑选 N 个副本
2:     tries=0,reject=false;//tries 为本副本尝试次数，reject 为此处挑选是否符合要求
3:     repeat
4:         i=set(); //设置随机数
5:         item = crush_bucket_choose() //从当前 bucket 中挑选
6:         if item!=osd then    //判断 item 是否为 osd
7:             bucket=item; //将  bucket 设置为 item
8:         end if
9:     util(item != osd)
10:    if collide||reject then    //判断冲突和过载
11:        reject=true;    //此处 item 不符合要求
12:        tries++;    //尝试次数加一
13:    end if
14:    if rep==0&&reject==false then    //第一个副本并且没有冲突和过载
15:        if combinationType!=null&&item.type==combinationType.replication(rep) then//挑选的节点是否符合特定组合类型
16:            reject=true;
17:            tires++;
18         end if
19:    end if
20:    if tries==Tries then //达到本副本的最大尝试次数
21:        countine; //跳出本次循环
22:    end if
23:    if(reject==false)then    //本副本符合要求
25:        osds[]<- item;    //将 item 加入 osds
26:    end if
27:end for
28:return osds;
```

对象进行读写热度分析后，根据读写热度划分模型划分对象的读写热度级别，然后根据表 4.3 读写热度与四种副本组合方式进行最佳匹配，然后通过修改 CRUSH 算法计算得到一组符合特定类型的存储节点，最后将对象从原节点迁移到相应热度对应的存储节点中。修改后的 CRUSH 算法如表 4.4 所示。

结合各个副本组合类型的读写性能与存储对象的读写热度，设置面向副本组合类型的数据存储策略，具体步骤如图 4.3 所示：

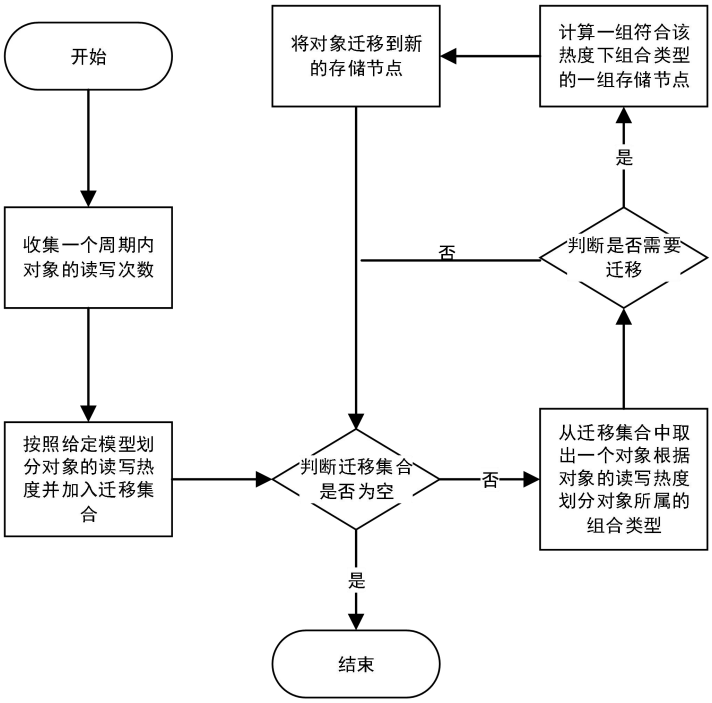

图 4.3 基于热度划分模型的异构副本存储流程图

(1). 收集一个周期内整个 Ceph 集群中存储所有对象的读写频率，并分别按照访问频率将存储对象排序；

(2). 根据存储对象的读写频率，按照热度模型划分存储对象的读写热度；

(3). 将存储对象及其读写热度加入数据迁移集合中；

(4). 根据存储对象读写热度及副本组合类别的读写性能，建立不同读写热度与组合类型的最佳匹配对应关系表，即读热度高的存储对象与读性能好的组合类型对应，写热度高的存储对象与写性能好的组合类型对应，以提升 Ceph 的系统性能；

(5). 判断数据迁移集合是否为空，若为空则直接结束，否则进入下一步；

(6). 从数据迁移集合中取出一个存储对象，并根据其读写热度在表 4.3 中查找与之匹配的组合类型（SType）；

(7). 判断对象目前所属的存储节点（COSDS）是否符合查找出来的组合类型 SType，若符合则跳过本次循环，进入步骤(5)；否则进入下一步；

(8).通过改进 Ceph 原生的 CRUSH 算法，计算出一组符合组合类型 SType 的一组存储节点（SOSDS）；

(9).将存储对象迁移从 COSDS 迁移到 SOSDS 中，跳到步骤(5)。

4.4 实验环境及结果

本节对本章提出的数据存储策略分别进行了三组实验。

第一组实验对异构副本的四种组合方式分别进行写入测试，测试每种组合方式的写延迟。

第二组实验对异构副本的四种组合方式，分别进行读延迟的测试。

第三组实验测试基于热度划分模型的异构副本组合存储策略与 Ceph 原生的策略下集群的读效率。

4.4.1 实验环境

实验环境：四台虚拟机，每台虚拟机分配 512M 内存，20G 存储空间，其中一台虚拟机装 Monitor 节点，剩下每台虚拟机装 1 个 SSD 的 OSD 节点，2 个 HDD 的 OSD 节点。

针对异构副本的四种组合类型，测试每种组合类型的写延迟以及读延迟，最后测试基于热度划分模型的异构副本组合存储策略数据迁移后的整个集群的读效率。

4.4.2 实验结果及分析

实验一 对异构副本的四种组合方式分别进行写入测试，测试每种组合方式的写延迟

第一组实验对异构副本的四种组合方式分别进行写入测试，测试每种组合方式的写延迟。通过图 4.4 所示的实验结果可以清晰的发现，A 类异构副本组合方式的写延迟是最高的，也就是说 A 类副本组合方式的写效率是最低的，同时也可以看到异构副本组合方式中的 B 类、C 类、D 类的写延迟在各个对象大小写入测试中基本保持持平。

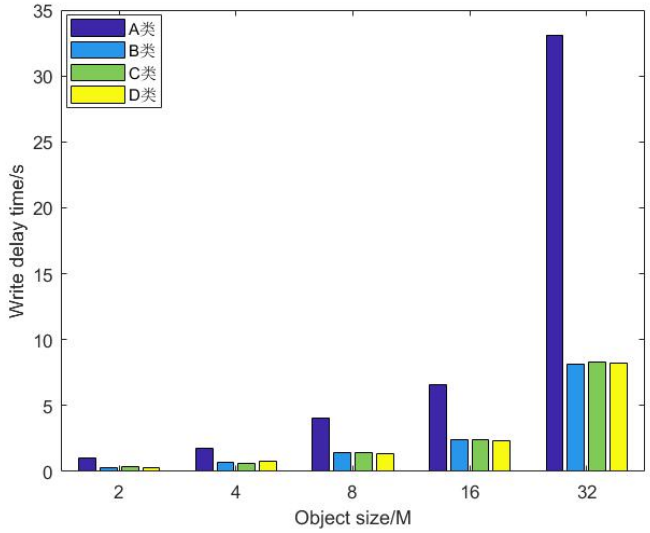

图 4.4 异构副本四种组合方式下的写延迟

通过对实验结果的分析，可知，A 类副本组合方式的写效率最低，B 类、C 类和 D 类的写效率一致而且都高于 A 类，这是由于在上文提出的副本弱一致性模型中，对象数据只需要在主副本写入后就可以响应客户端，同时由主副本节点来进行从副本节点的对象数据写入。所以对象的写效率跟主副本节点的节点类型相关，而在异构副本的四种组合方式中，A 类组合方式的主副本节点是 HDD，B 类、C 类和 D 类副本组合方式的主副本节点都是 SSD，而 SSD 的性能远远高于 HDD，故而在四种异构副本的组合方式中，A 类组合方式的写效率最低，B 类、C 类和 D 类的写效率一致而且都高于 A 类。同时通过实验结果也可以发现，当对象大小小于 16M 时，A 类组合方式跟 B 类、C 类和 D 类的写效率差距还不是很大，当对象大小大于 16M 以后，写效率的差距显著拉大。这是由于当对象大小较小时，HDD 与 SSD 的 I/O 差距还不是很明显，还没有达到 HDD 节点的 I/O 瓶颈，当对象大小为 16M 后逐渐达到 HDD 节点的 I/O 瓶颈，同时由于 SSD 的 I/O 性能高于 HDD 的 I/O 性能，所以当 HDD 逐渐达到 I/O 瓶颈后与 SSD 的效率差距逐渐拉大。故而造成 A 类副本组合方式与 B 类、C 类和 D 类的写效率差距随着对象大小的增长而增大。

实验二 对异构副本的四种组合方式，分别进行读延迟的测试

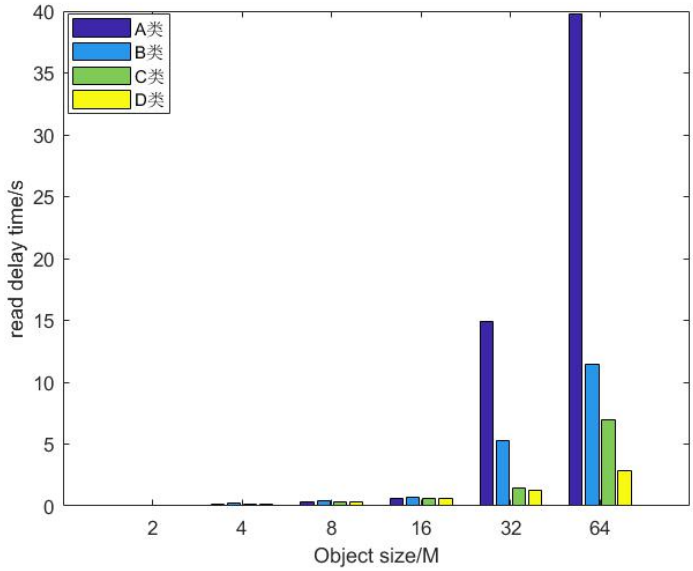

图 4.5 异构副本四种组合方式下的读延迟

第二组实验通过对异构副本的四种组合方式，分别进行读延迟的测试，从而得到每种副本组合方式的读性能。实验结果如下图 4.5 所示：在对象大小小于等于 16M 时，A 类、B 类、C 类和 D 类四种异构副本组合方式的读延迟是基本一致的，当对象大小大于等于 32M 时，可以清晰的发现四种异构副本组合方式的读延迟从高到低依次为 A 类、B 类、C 类、D 类，且这种差距随着对象大小的增大逐渐增大。这是因为在多副本读优化策略下，通过分析每个副本节点的综合性能，然后让综合性能最高的数据节点来提供读取服务。在以上四种异构副本组合方式中，四种副本组合方式的读效率是基本一样的，随着读取对象的增大，逐渐达到 I/O 瓶颈，四种副本组合方式的读效率差距逐渐明显，所以当对象数据大小大于等于 32M 后读效率从高到低依次为 D 类、C 类、B 类、A 类。

实验三 测试基于热度划分模型的异构副本组合存储策略，Ceph 原生策略和上文策略（第三章提出的基于副本弱一致性模型的组合存储策略）在集群读效率方面的对比。

第三组实验是测试基于热度划分模型的异构副本组合存储策略对整个集群读效率的提升。通过实验一和实验二可以得到四种异构副本组合方式的差距主要是在读效率方面，而基于热度划分模型的异构副本组合存储策略是通过对一段时间内数据的热度进行分析，然后将相应热度的数据迁移到对象的副本组合方式中，从而达到将热度高的数据迁移到性能高的组合方式中，从而提高整个集群的性能。将基于热度划分模型的异构副本组合存储策略，Ceph 原生的策略、以及上文策略（第三章提出的基于副本弱一致性模型的组合存储策略）做实验对比，实验结果如图 4.6 所示。

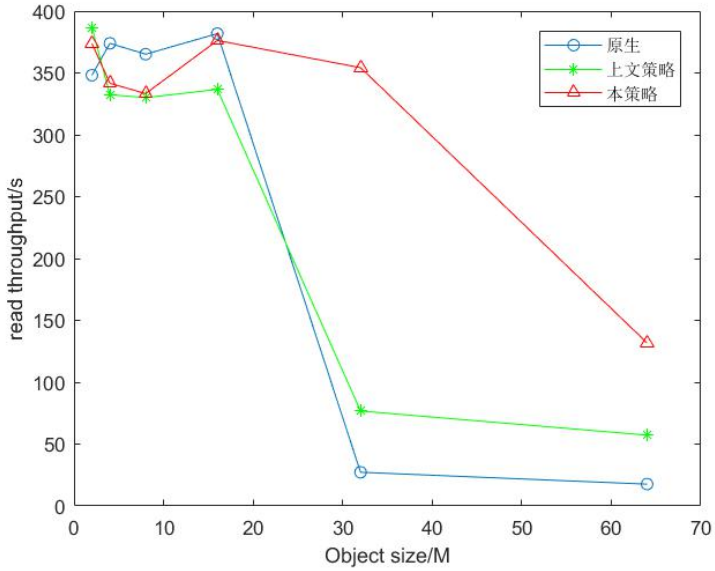

图 4.6 不同策略写的读吞吐量对比

通过实验结果可以发现，基于热度划分模型的异构副本组合存储策略的优势随着对象大小的增大与 Ceph 原生策略和基于副本弱一致性模型的组合存储策略的性能优势逐渐显现，同时，当对象大小较小时，三种策略的读效率基本持平，这是因为当文件较小时，SSD 的性能优势还没有显现出来，因此，随着对象大小的增大基于热度划分模型的异构副本组合存储策略的优势逐渐显现，表现出比较高的读效率。通过实验结果可以发现基于热度划分模型的异构副本组合存储策略对整个 Ceph 集群读效率的提升。

实验结果表明，本文基于热度划分模型的异构副本组合存储策略，由于有效的利用异构副本的不同组合方式，从而充分发挥不同存储介质的性能特点，因此在提高集群读效率等方面有较大提升。

4.5 本章小结

本章针对在现有策略下 Ceph 不能够有效利用异构副本的不同组合方式，提出基于热度划分模型的异构副本组合存储策略，通过将对象的热度与异构副本的组合方式结合，从而发挥异构环境下不同副本组合方式的特点。并通过实验证明基于热度划分模型的异构副本组合存储策略在提升集群读取效率方面的作用。

第 5 章 总结与展望

本章对本文针对 Ceph 读写延迟较高和不感知异构提出的解决方案进行总结，并指出以后的工作方向。

5.1 总结

Ceph 是第一个版发布于 2012 年的开源分布式存储系统，由于其显著的特点，使其得到越来越多的应用。但是在进行写操作时，Ceph 为了保证存储节点中对象数据的一致性，采用的是强一致性的写入策略，在写对象时，当对象在所有副本节点上都写入完成后才响应客户端，在此期间客户端不能够进行任何的操作，故而造成较高的写延迟。同时 Ceph 在读取对象数据时，只读取对象的主副本节点，从而造成从副本节点的性能浪费。更重要的是 Ceph 在设计之初并未考虑到存储节点的异构性问题，从而在异构副本的环境下不能充分发挥异构副本的性能特点，造成性能浪费。本文针对以上问题分别提出了解决方法，以求能够有效的提高 Ceph 集群的读写性能。下面对论文的主要工作和研究成果进行总结。

(1). 本文研究了 Ceph 异构存储优化机制。首先针对 Ceph 原生的写策略造成较高的写延迟，提出一种副本弱一致性模型。该模型在写操作时，客户端把对象发送给主副本节点，当对象在主副本节点写完毕，则立即返回写成功消息给客户端，再更新到副本，从而提高写效率。并由主副本节点来确保该对象在从副本节点中写入磁盘过程，从副本节点写入磁盘完成后，返回写成功消息给主副本节点。若在设定时间内主副本节点未收到从副本节点的写成功消息，则由主副本节点重新发送对象数据给从副本节点，重复写过程。

进一步地，针对 Ceph 原生的读策略造成从副本节点的性能浪费，提出了一种多副本读优化策略。在读取操作时，由于本文采用的是弱一致性的写操作，所以存在读取对象时，对象数据只在主副本节点写入完成，如果是这种情况的话则从主副本节点读取对象数据，若对象在所有副本节点都写入完成，则通过分析每个副本节点的综合性能，让综合性能最高的副本节点提高读服务，从而有效的提高读效率。

(2). 针对 Ceph 在存储节点的异构情况下，不能发挥异构副本的性能特点。首先提出了基于副本弱一致性模型的组合存储策略，通过将主副本节点设置为 SSD，从而提高写效率。通过进一步分析异构副本的组合方式，异构副本的组合方式分为 A 类、B 类、C 类和 D 类，并对每种组合方式的读写性能进行分析，最后通过建立对象的热度模型，提出基于热度划分模型的异构副本组合存储策略，相应热度的对象迁移到对应的副本组合存储方式中，从而有效提高整个集群的读效率。

5.2 展望

本文已经完成了预期的研究内容，并且通过实验验证了本文提出的读写优化模型，以及组合存储策略和数据迁移模型在提高 Ceph 集群读写效率方面的有效性。在未来的工作中，本文可以在以下方面进行更深入的研究工作。

(1). 在多副本读优化策略中，因为在写操作时采用的是副本弱一致性模型，所以在读取操作时，可能存在对象在从副本节点上还未写入完成，此时只能读取主副本节点，同时由于需要判断对象是不是在从副本节点上写入磁盘完成，从而造成对读效率的影响，因此下一步工作可以针对此种情况提出一种有效的解决方法，以提高此种情况下的读效率。

(2). 在基于热度划分模型的异构副本组合存储策略中，对象热度的判断，是通过收集一段时间内的读写次数来进行的，但是在这段时间内的读写次数可能高也可能低，甚至这段时间内整个集群中的对象读写次数都为零，所以仅通过收集一段时间内的读写次数来进行对象的热度分析还不太适合所有场景。因此下一步的工作可以选择更为合适的热度分析模型，以求在一个热度分析周期内能更准确有效的分析出对象的热度信息，提升组合存储的效率。

参考文献

[1] Liu Yingbo, Wang Feng, Deng Hui, et al. Investigation on distributed file system for scientific big data storage[J]. ICIC Express Letters Office, 2018, 6(9): 2577-2582.

[2] 涂新莉，刘波，林伟伟．大数据研究综述[J]．计算机应用研究，2014, 31(6):1612-1616.

[3] Zhang Shuo, Miao Li, Zhang Dafang, et al. A strategy to deal with mass small files in HDFS[C]// Sixth International Conference on Intelligent Human-machine Systems & Cybernetics. Zhejiang: IEEE Press, 2014: 331-334.

[4] Won H, Nguyen M C, Gil M S, et al. Moving metadata from ad hoc files to database tables for robust, highly available, and scalable HDFS[J]. The Journal of Supercomputing, 2017, 3(6): 2657-2681.

[5] 张毕涛, 辛阳. 基于 Ceph 的海量小文件存储的优化方法[J]. 第十届中国通信学会学术年会论文集, 2014.

[6] 胡博, 陈桓, 张良杰, 等. 一种跨 HDFS 集群的文件资源调度机制[J]. 计算机学报, 2017, 40(9):2093-2110.

[7] Liu Jiang, Li Bing, Song Meina. The optimization of HDFS based on small files[C]//2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT). Beijing: IEEE, 2010: 912-915.

[8] Wu Duo, Wang Yong, Feng Hao, et al. Optimization design and realization of ceph storage system based on software defined network[C]// International Conference on Computational Intelligence and Security. Hongkong: IEEE Press, 2018: 277-281.

[9] 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. Berkeley: USENIX Association Press, 2006: 307-320.

[10] Lv Wuyou. Content level access control for openstack swift storage design and implementation of an encrypted cloud storage system based on openStack swift[C]// 7th IEEE International Conference on Software Engineering and Service Science. Beijing: IEEE Press, 2017: 373-377.

[11] 余骏, 肖健, 于策, 等. 分布式文件系统本地数据访问的优化[J]. 计算机应用研究, 2014(6):1635-1638.

[12] 胡勋, 杨靖琦. 基于 Ceph 的云网盘系统设计与实现[J]. 信息系统工程, 2018 (6): 68.

[13] 彭潇, 张俊, 印钊. 基于 Ceph 的 OpenStack 存储克隆链[J]. 计算机与现代化, 2017 (9): 114-119.

[14] 陈豪钧. 浅谈 Ceph 的设计原理[J]. 信息通信, 2017 (4): 146-147.

[15] Zhang X, Gaddam S, Chronopoulos A T. Ceph distributed file system benchmarks on an openstack cloud[C]// IEEE International Conference on Cloud Computing in Emerging Markets. Bangalore: IEEE Press, 2015: 113-120.

[16] Zhang Jiayuan, Wu Yongwei, Yeh-ching Chung. PROAR: A weak consistency model for Ceph[C]//2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). Wuhan: IEEE Press, 2016: 347-353.

[17] 刘飞, 蒋德钧, 张欢, 等. 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, 44(6): 17-22.

[18] 田怡萌, 李小勇, 刘海涛. 分布式文件系统副本一致性检测研究[J]. 计算机研究与发展, 2012, 49(S1): 276-280.

[19] 贾艳燕, 娄燕飞, 杨树强, 等. 分布异构多数据库中多副本一致性维护研究与实现[C]//第二十三届中国数据库学术会议论文集 (技术报告篇). 2006.

[20] 葛微, 罗圣美, 周文辉, 等. HiBase:一种基于分层式索引的高效 HBase 查询技术与系统[J]. 计算机学报, 2016(1):140-153.

[21] 周江, 王伟平, 孟丹, 等. 面向大数据分析的分布式文件系统关键技术[J]. 计算机研究与发展, 2014, 51(2):382-394.

[22] Mu Q, Jia Y, Luo B. The optimization scheme research of small files storage based on HDFS[C]//2015 8th International Symposium on Computational Intelligence and Design (ISCID). Hangzhou: IEEE Press, 2015, 1: 431-434.

[23] Xie Tao, Liang Alei. Small file access optimization based on GlusterFS[C]//Proceedings of 2014 International Conference on Cloud Computing and Internet of Things. Guangzhou: IEEE Press, 2014: 101-104.

[24] 杨冬菊, 李青, 邓崇彬. HDFS 异构集群中的分级存储调度机制[J]. 小型微型计算机系统, 2017, 38(1): 29-34.

[25] Kumar Rishabh, Vishrut Mehta, Tushant Jha, et al. HetStore: A platform for IO workload assignment in a heterogeneous storage environment[C]//2016 IEEE International

Conference on Cloud Computing in Emerging Markets (CCEM). Bangalore: IEEE Press, 2016: 25-31.

[26] Krish K R, Anwar A, Butt A R. hats: A heterogeneity-aware tiered storage for hadoop[C]//2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Chicago: IEEE Press, 2014: 502-511.

[27] 刘鑫伟. 基于 Ceph 分布式存储系统副本一致性研究[D]. 武汉: 华中科技大学, 2016.

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

[29] Ke Zhan, Piao Ai Hua. Optimization of Ceph reads/writes based on multi-threaded algorithms[C]//2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). Sydney: IEEE Press, 2016: 719-725.

[30] Wu Chun-Feng, Tse-Chuan Hsu, Yang Hongji, et al. File placement mechanisms for improving write throughputs of cloud storage services based on Ceph and HDFS[C]//2017 International Conference on Applied System Innovation (ICASI). Sapporo: IEEE Press, 2017: 1725-1728.

[31] 詹玲, 朱承浩, 万继光. Ceph 文件系统的对象异构副本技术研究与实现[J]. 小型微型计算机系统, 2017, 38(9): 2011-2016.

[32] Meyer S, Morrison J P. Supporting heterogeneous pools in a single ceph storage cluster[C]//2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). Timisoara: IEEE Press, 2015: 352-359.

[33] Shankar V, Lin R. Performance study of ceph storage with intel cache acceleration software: Decoupling hadoop mapreduce and hdfs over ceph storage[C]//2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud). New York: IEEE Press, 2017: 10-13.

[34] Azginoğlu N, Eren M A, Çelık M, et al. Ceph-based storage server application[C]//2018 6th International Symposium on Digital Forensic and Security (ISDFS). Antalya: IEEE Press, 2018: 1-4.

[35] Weng Jia Yow, Yang Chao Tung, Chang Chih Hung. The integration of shared storages with the CephFS and Rados Gateway for big data accessing[C]//2018 IEEE 42nd Annual

Computer Software and Applications Conference (COMPSAC). Tokyo: IEEE Press, 2018, 2: 93-98.

[36] 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.

[37] Edwin H.-M. Sha, Liang Yutong, Jiang Weiwen, et al. Optimizing data placement of mapreduce on ceph-based framework under load-balancing constraint[C]//2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). Wuhan: IEEE Press, 2016: 585-592.

[38] Poat M D, Lauret J. Achieving cost/performance balance ratio using tiered storage caching techniques: A case study with CephFS[C]//Journal of Physics: Conference Series. Santiago: IOP Publishing Press, 2017, 898(6): 062022.

[39] Borges G, Crosby S, Boland L. CephFS: a new generation storage platform for Australian high energy physics[C]//Journal of Physics: Conference Series. Santiago: IOP Publishing Press, 2017, 898(6): 062015.

[40] 彭潇, 张俊, 印钊. 基于 Ceph 的 OpenStack 存储克隆链[J]. 计算机与现代化, 2017 (9): 114-119.

[41] Weil S A, Leung A W, Brandt S A, et al. Rados: a scalable, reliable storage service for petabyte-scale storage clusters[C]//Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07. Reno: ACM Press, 2007: 35-44.

[42] 王建宇. OpenStack 平台与 Ceph 统一存储的集成[J]. 中国管理信息化, 2016 (4): 168-168.

[43] Yunjung Han, Kwonyong Lee, Sungyong Park. A dynamic message-aware communication scheduler for Ceph storage system[C]//2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W). Augsburg: IEEE Press, 2016: 60-65.

[44] Weil S A, Brandt S A, Miller E L, et al. CRUSH: controlled, scalable, decentralized placement of replicated data[C]//SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. New York: IEEE Press, 2006: 31-31.

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

[46] 王勇,叶苗,等. 基于软件定义网络和多属性决策的 Ceph 存储系统节点选择方法[J]. 计算机学报, 2018(2):93-107.

[47] Liang Xiao Yang , Guan Zhang Cen . Ceph CRUSH data distribution algorithms[J]. Applied Mechanics and Materials, 2014, 596:196-199.

[48] 沈良好, 吴庆波, 杨沙洲. 基于 Ceph 的分布式存储节能技术研究[J]. 计算机工程, 2015, 41(8): 13-17.

[49] Arafa Y, Barai A, Zheng M, et al. Fault tolerance performance evaluation of large-scale distributed storage systems HDFS and Ceph Case Study[C]//2018 IEEE High Performance extreme Computing Conference (HPEC). Waltham: IEEE Press, 2018: 1-7.

[50] van der Ster D C, Lamanna M, Mascetti L, et al. Ceph-based storage services for Run2 and beyond[C]//Journal of Physics: Conference Series. Fukuoka: IOP Publishing Press, 2015, 664(4): 042054.

[51] 谢型果. Ceph 设计原理与实现（第一版）[M]. 北京：机械工业出版社. 2017: 116-120.

[52] Liao Jianwei, Li Li, Chen Huaidong, et al. Adaptive replica synchronization for distributed file systems[J]. IEEE Systems Journal, 2015, 9(3): 865-877.

[53] He Shuibing, Sun Xian He. A Cost-Effective Distribution-Aware Data Replication Scheme for Parallel I/O Systems[J]. IEEE Transactions on Computers, 2018, 67(10): 1374-1387.

[54] https://www.backblaze.com/blog/how-long-do-disk-drives-last/

[55] 聂凤. 混合存储系统中自适应存储策略的研究[D]. 武汉: 华中科技大学, 2013.

[56] 张震, 汪斌强, 陈庶樵, 等. 几何布鲁姆过滤器的设计与分析[J]. 电子学报, 2012, 40(9): 1852-1857.

[57] 谢鲲, 伍菊红. 一种面向覆盖查询的布鲁姆过滤器设计[J]. 计算机工程, 2016, 42(4):301-306.

[58] 谢鲲, 施文. 一种隐私保护的可逆布鲁姆过滤器[J]. 计算机工程与科学, 2017, 39(6): 1104-1111.

致谢

匆匆，匆匆的三年的研究生生活就要结束了，三年的时候过得如此匆匆，回忆这段时光，内心感觉很充实，很踏实，很幸福。借此论文之际，以最诚挚的心情向这三年来给予我支持和帮助的人表达我的感谢。

感谢我的导师熊安萍教授，熊老师作为我们实验室团队的核心负责人，不仅在学习上指导我们，也在生活工作上教育我们，教育我们如何成为一个对社会对祖国有用的人。教育我们善待生活工作上的每一个人，戒急戒躁，用实际行动向我展示了对待学术的认真态度，是我一生的良师，足够一生受益。在此向我的导师熊老师致以最真诚的感谢！

感谢实验室的每一个同门，特别感谢我实验室三年的同桌皮阳同学，感谢他三年来对我的鼓励和包容，感谢他对我实验上帮助，是你的帮助让我提前完成了实验，是你的帮助让我不断进步；感谢实验室的师兄师姐师弟师妹对我的关心和支持，感谢你们为我营造的积极向上的实验室氛围，感谢你们给我这个如家一般的实验室大家庭，让我度过了开心愉快同时又收获满满的研究生生活，再次感谢你们！

感谢我的父亲，是你的教导一种激励着我不断前行，是你的支持让我一直觉得自己有一个坚强的后盾；感谢我的母亲，是你的每次问候温暖着我的心，让我在苦恼和无助时感受到了巨大的力量。感谢你们给我提供一个温暖的家，感谢你们对我这么多年来的支持，祝你们永远身体健康，开开心心每一天！

感谢答辩组的每一个教授和专家，感谢你们给我的宝贵意见，让我进一步完善我的研究工作，感谢你们！

谨以此献给所以关心我的人，感谢你们，祝你们永远开心！

攻读硕士学位期间从事的科研工作及取得的成果

参与科研项目：

[1] 云南省精准扶贫大数据平台，省部级一般，2017.09~2018.10

发表及完成论文：

[1] 熊安萍,**姚朋成**,龙林波.基于Ceph的读写模型优化和异构副本组合方式,201810660726.6 [P].专利.(已受理)

[2] **Pengcheng Yao**. Read/write performance optimization based on Ceph heterogeneous storage[C]//2019 International Conference on Artificial Intelligence and Communication Engineering (AICE 2019).(已录用)