分类号 TP333 学号 GS12062384

U D C 密级 公 开

工程硕士学位论文

基于 Ceph 的分布式存储节能技术研究与实现

硕士生姓名 沈良好

学 科 领 域 软件工程

研 究 方 向 分布式存储

指 导 教 师 吴庆波 研究员

国防科学技术大学研究生院

二〇一四年十月

Research and Implementation of Distributed Storage Energy Saving Technologies Based on Ceph

Candidate：Shen Liang-hao

Advisor：Prof. Wu Qing-bo

A thesis

Submitted in partial fulfillment of the requirements

for the professional degree of Master of Engineering

in Software Engineering

Graduate School of National University of Defense Technology

Changsha，Hunan，P.R.China

Oct，2014

独 创 性 声 明

本人声明所呈交的学位论文是我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，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术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谢意。

学位论文题目：基于 Ceph 的分布式存储节能技术研究与实现

学位论文作者签名：沈良好　　日期：2014年 10月 17日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于 Ceph 的分布式存储节能技术研究与实现

学位论文作者签名：沈良好　　日期：2014年 10月 17日

作者指导教师签名：　　日期：2014年 10月 17日

目 录

表 目 录

图 目 录

摘 要

分布式存储是数据中心广泛使用的存储系统，在带来了性能与可靠性的同时也消耗了大量的能源，在数据中心能耗中分布式存储消耗的电能占了非常大的比例。因此，为了减少数据中心成本以及实现绿色计算，分布式存储的节能技术已经成为了学术界和工业界研究的热点之一。

Ceph 是近年来出现的热门的分布式存储，并因具备了高扩展性、高性能、高可靠性以及高达 PB 级的数据容量受到了越来越多的关注。然而，Ceph 同样面临着如何减少能耗的问题。一方面，Ceph 的数据放置算法使得数据副本随机分布于各数据节点，限制了系统能够达到的节能比例，在系统规模较大时，能够节省的能耗非常有限。另一方面，Ceph 存储系统中没有有效的功耗管理机制，所有节点均为全时间运行，在系统的低负载时期产生了大量不必要的能耗。因此，针对这两个问题，本文做了以下工作：

第一，针对 Ceph 基于 CRUSH 算法的数据布局，本文提出以节能为目的 PEPGO 数据布局优化算法。通过对系统节点的层级关系描述图 Crushmap 以及副本放置规则的分析，确定数据分布的故障域，进而将故障域划分为不同的功耗组。数据副本在被划分到不同故障域的同时被划分进不同的功耗组，从而在保证数据全集可用的情况下，增加了系统中可以关闭的节点的数目，提高了系统能够节能的比例，随着系统节点数目的增加，节省的能耗将非常可观。

第二，基于档位切换（Gear Shifting）模型，提出 Ceph 的多级功耗管理策略，使得系统在不同的负载状态下运行于不同的功耗级别，节省能耗。Ceph 存储系统设计的初衷是高性能和高扩展性，然而实际应用中，系统经常会处于低负载状态，此时，可以适当的关闭部分节点以节省能耗。本文提出的多级功耗管理策略能够根据负载状态调整功耗状态，从而在满足负载需求的情况下，有效地减少系统的能耗。

最后，在上述两点的基础上设计并实现了 Ceph 分布式存储的功耗管理系统。该系统利用了 Ceph 的配置和管理接口实现数据布局的优化，利用监控接口实现负载跟踪，并结合服务器硬件的电源管理机制以及网络唤醒等技术进行具体的级别切换，从而实现对 Ceph 系统的动态电源管理。本文对系统进行实验与测试，结果表明，该系统够有效的减少 Ceph 存储系统的能耗，并保证 Ceph 分布式存储的服务质量。

主题词：Ceph；分布式存储；数据布局；功耗管理；节能计算

ABSTRACT

Distributed storage systems are widely used in data centers, because of their conspicuous performance and reliability. However, the energy consumption of distributed storage systems is considerable and occupies significant part of the whole data centers' energy costs. Therefore, energy saving technologies of distributed storage system are very import for reducing the costs of data centers and green computing, and they have already become hot research points, both in academia and industry.

Ceph, invented a few yeas ago, is one of the most popular distributed storage systems nowadays. Ceph has many advantages, like high scalability, high performance, high reliability and the petabyte scale storage space. As a result, Ceph has gained increasing attention. Yet Ceph also faces the problem of reducing power consumption. For one thing, in Ceph, data blocks and their replicas are distributed to data nodes randomly, which restricts power saving proportion of the system, and when the system scales up, the power to be saved would leave munch to be desired. For another, there is no power managing mechanism in Ceph, all nodes keep running all the time, and this will cause unnecessary power consumption when system is in low load. Considered these issues, we do the following work:

Firstly, we introduce an algorithm, named PEPGO, to optimize the data layout produced by the CRUSH algorithm. Our algorithm analyses Crushmap and placement rules, which are used to describe the cluster's hierarchical relationship, and the replica placement strategy, to find the appropriate failure domains, and then split the failure domains into power groups. The data blocks' replicas will be placed in different power groups as well as in different failure domains. As a result, more nodes can be shut down, and all data sets still keep accessible . This will cause considerable energy saving when the number of nodes goes up.

Secondly, we introduce a multi-level power manage strategy, which makes system in different power level according to the system's load, based on the Gear Shifting mode. The original purposes of Ceph are high performance and high scalability, but in real workload, there are many low load periods, when some nodes can be powered off for energy efficiency. The strategy we proposed can determine the system’s power state according to the system load. As a result, the needs of system’s load are met, as well as the power consumption of system can be reduced.

Finally, we design and implement a power manage system for Ceph, based on the two former points. This system uses the existing Ceph's manage, configure interfaces to optimize the data layout, and traces the system load status by Ceph’s monitor interfaces. Besides, in this system, power manage mechanism of hardware and the network based waking up technology are used to perform the manage activities. The results of

evaluation illustrate the system can reduce the power consumption of Ceph effectively, meanwhile, the quality of service is preserved.

Key Words：Ceph, distributed storage, data layout, power manage, energy-efficient computing

第一章　绪论

1.1　课题背景

随着信息技术的发展，IT 世界的数据量呈现出爆发式的增长，根据国际数据公司 IDC 的研究报告，从 2005 年到 2020 年，全球信息总量将会增长约 300 倍，达到 40ZB（1021 B）[1]。如此海量数据的存储与分析处理，对 IT 基础设施有着非常高的要求，而数据中心则凭借自身强大的计算及存储能力为其提供了重要的平台。为此，各大企业及研究机构纷纷建立自己了的数据中心，以应对“大数据”时代的挑战。

绿色节能是数据中心建立及运行维护中需要面对的重要问题。一方面，随着基础设施规模的不断扩大，数据中心需要为日益剧增的能源消耗买单。数据中心的基础设施在生命周期内消耗的电费，很有可能超过最初购买设备的花费。而在数据中心的 TCO（总拥有成本）中，用于供电和冷却的能源花费已经是最主要的组成部分，对数据中心的拥有者来说是一项巨大的支出。另一方面，随着数量的增加与规模的扩大，数据中心所消耗的能源，在整个社会能源消耗的中所占的比重也越来越大。据统计，仅仅在 2010 年，全球的数据中心就消耗了过了 2000 亿千瓦时的电能，约占全球总用电量的 1.3%[2]，且这一数字仍呈逐年上升的趋势。在能源的重要性日益凸现，倡导节能低炭的今天，这显然是不可忽视的事实。值得注意的是，如此巨大的能耗中，真正用于服务用户请求却非常有限，只占到了 6%~12%[3]。因此，降低能耗，提高能源使用效率对数据中心有着极为的重要意义，绿色数据中心是所有数据中心的发展趋势。

用于存储系统的能源消耗是数据中心能耗的重要来源之一。部分研究人员发现数据中心的能耗组成中，存储系统消耗的电能约占 30%[4][5]，仅次于计算资源。而在有些应用场景中，这一比例会更高，例如来自 Oracle 和 HP 的研究者们发现，他们的 OLTP（在线事务处理）系统在 7 年的 TPC-C 基准测试过程中，存储系统所消耗的电能甚至占了总电能消耗的 75%以上[6]，如图 1.1 所示。此外，存储服务器巨大的用电量也导致了机房热量的大量产生，这也引起了机房冷却设备的电能消耗，从而进一步加剧了数据中心总的能源消耗。因此，减少存储系统的电能消耗对数据中心能耗有着非常重要的影响，是实现数据中心节能目的的重要手段之一。

本课题的研究工作受到了国家 863 计划项目“智能云服务与管理平台核心软件与系统”的支持。

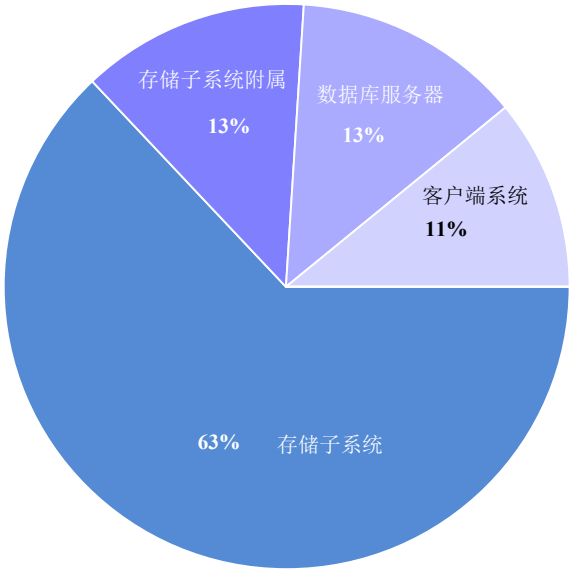

图 1.1　TPC-C 系统的能耗组成[6]

1.2　研究意义

近年来，分布式存储系统得到了迅速的发展与并在数据中心中得到了广泛的应用，与传统的单个磁盘及磁盘阵列相比，分布式存储系统能够提供更大的容量、更高的性能以更好的数据安全保证，这些优点使得分布式存储在云计算及大数据处理中起着至关重要的作用。GFS[7]和 Bigtable[8]部署在 Google 公司的数千个的普通服务器上，管理着 Google 的海量数据，并对网页搜索，谷歌地图和谷歌金融等业务提供灵活高效的服务。Dynamo[9]作为一种 Key/Value(键值)对型的分布式存储，已经成为亚马逊一系列核心服务的基础性存储技术。HBase[10] 是在 Hadoop 平台上开发的分布式数据库，在 Facebook 的海量数据处理中起着关键作用，存放着每月超过 1350 亿条的信息。但与此同时，大规模的分布式存储的能耗也是巨大的，如 Yahoo 所有的 Hadoop 集群中包含了 38000 多个节点[11]，以每个节点 400 瓦的功耗为例，则这些集群在一年的时间内，服务器消耗的电能就多达 6700 万千瓦时，这还未包括用于机房冷却的能耗。

Ceph[12]是近年出现的分布式存储系统，是一种软件定义存储（SDS[13]）的解决方案。因为具备了高性能、高可靠性及高扩展性的特点，并且支持对象存储、块级及文件系统级的统一存储，Ceph 已经成为云计算平台 OpenStack 的通用存储之一，常被构建为大规模的存储后端。此外，Ceph 项目是开放源代码的，吸引了来自全球的业界人士的关注，社区活跃着数百名的代码贡献者。随着 Redhat 公司对 Inktank（Ceph 项目的资助和维护公司）的收购，Ceph 将会在分布式存储领域

得到更多的推广与应用。因此基于 Ceph 研究分布式存储系统的节能技术，实现 Ceph 系统的节能的同时保留其可用性、可靠性等优点，有着实际的意义和广泛的应用前景。

1.3 论文的主要工作

本文分析了Ceph分布式存储在节能方面的不足并提出了相应的优化算法和策略，主要的工作包含了以下内容：

第一，对Ceph数据布局的优化，解决了如何在Ceph中节省更多能耗的问题。多副本是分布式存储中的重要技术之一，是提升性能、可靠性及可用性的重要保证，而副本的放置策略往往起着决定性的作用。但从节能的角度看，副本的放置策略也决定了系统的节能能力。因此，本文分析了 Ceph 的 CRUSH 数据布局算法在节能方面的不足，提出了面向节能的PGPEO数据布局优化算法，从而能够在保证数据集可访问性的情况下，关闭更多的节点，提升系统节能的上限。

第二，提出 Ceph 系统的多级功耗管理策略，解决了在 Ceph 中如何管理功耗以节能的问题。Ceph 系统缺乏有效的功耗管理机制，本文基于档位切换模型结合 Ceph 系统的特点，建立了 Ceph 的多级功耗管理模型，分析 Ceph 在不同状态下的功耗与能耗，并基于此模型提出了 Ceph 的多级功耗管理策略。该策略有效解决了负载感知、级别切换、数据同步等问题，能够在满足负载需求的情况下，降低系统的功耗。

第三，实现Ceph系统的多级功耗管理原型系统，并进行实验评估。基于上述的数据布局优化和多级功耗管理策略，设计并实现了 Ceph 多级功耗管理原型系统。在具体编码实现时充分利用了Ceph系统已有的管理、监控及配置接口，以及服务器硬件的电源管理功能等，在减少工作量的同时尽量减少对系统运行的影响。该系统能够优化 Ceph 的数据布局，统计 Ceph 系统的 I/O 负载情况并根据给定的策略动态地管理 Ceph 系统的功耗状态。实验表明，该系统能够有效地降低 Ceph 系统的能耗，并保证服务质量。

1.4 论文组织结构

论文共分为六章，其中第三、四、五章为本文的重点，各章节的具体内容描述如下：

第一章为绪论，主要介绍了本文研究的背景、研究意义以及对论文主要的工作进行了综述。

第二章内容为相关工作。从系统架构及技术特点等方面介绍了本文研究所基

于的平台 Ceph 分布式存储系统，并详细介绍并分析了与本文研究密切相关的Ceph 数据布局。接着对分布式存储系统常用的节能技术进行了分类。最后综述了分布式存储系统常用的功耗管理模型。

第三章内容面向节能的数据布局的优化。首先分析并提出了Ceph 数据布局在节能方面存在的问题，接着提出了 PGPEO 数据布局优化算法，最后 PGPEO 算法的效果和给系统带来的影响进行了分析与实验评测。

第四章为Ceph 的多级能耗管理策略。首先介绍了常用的分布式存储系统功耗管理的模型，并进行了分析比选。接着描述了对 Ceph 系统的功耗管理模型并进行了功耗分析。最后提出了 Ceph 的多级功耗管理策略，并对该策略的有效性和性能进行实验评测。

第五章为多级功耗管理原型系统的实现与实验评估。首先介绍了 Ceph 多级功耗管理系统的体系结构设计和工作流程，接着对各个功能模块及实现细节进行了详细的描述，最后陈述了实验评估的工作，包含了功耗管理系统对Ceph 系统功耗管理的效果及对系统性能的影响。

第六章为总结，综述了本文的研究成果，分析了存在的问题，并讨论了进一步的研究的工作。

1.5 本章小节

本章主要介绍了论文的研究背景和研究意义,为下一步进行深入的研究打下基础，并阐述了主要的研究内容和论文的组织结构。

第二章　相关研究

2.1　Ceph 系统架构及技术特点

Ceph 是近年出现的一个分布式存储系统，具备了高性能、高可靠性及高扩展性的特点，并且支对象存储、块级及文件系统级的统一存储。

2.1.1 Ceph 的系统架构

Ceph 分布式存储系统包含了四个主要的组件：对象存储设备（OSD）集群, 元数据服务器集群和客户端和监控器集群，如图 2.1 所示。其中，OSD 集群负责来自与客户端与元数据服务器的数据对象存储。元数据服务器集群负责提供元数据服务及管理文件系统的名称空间(Name Space)。监视器用于监测及管理 OSD 集群的节点状态与映射关系。客户端是执行 I/O 请求的组件，用于提交来自于应用层的 I/O 请求。

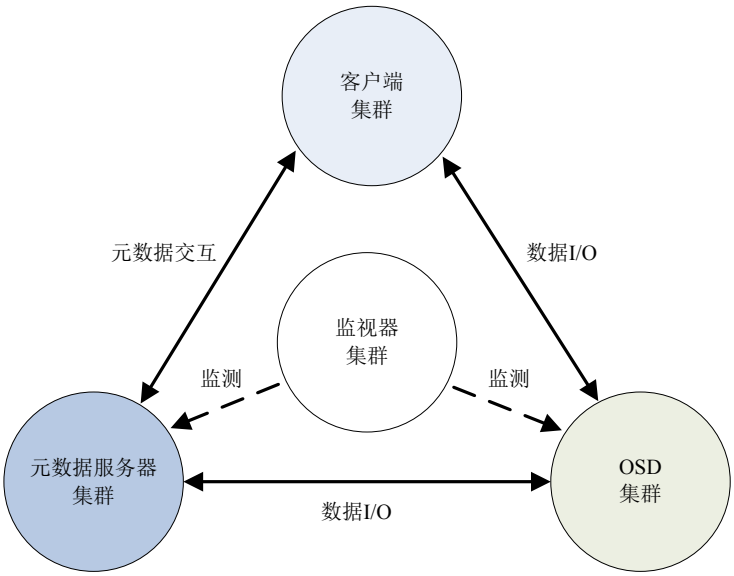

图 2.1　Ceph 的系统架构

2.1.2 Ceph 的技术特点

Ceph 与传统的分布式存储系统在设计与实现上有着很多不同之处，而这些特点也是其成功的最主要的原因。

(1) 数据与元数据分离

Ceph 最大程度上实现了数据与元数据管理的分离。元数据操作，如打开、关

闭、重命名等都是在元数据服务器集群中进行的，而数据本身的 I/O 操作，由客户端直接和 OSD 集群交互进行的，如图 2.2 所示。不同于其他传统分布式存储的是，Ceph 的元数据中并没有包含数据的存放位置，即在 Ceph 中不存在用于保存数据的逻辑编号与物理位置映射的表类结构，数据的位置完全是通过计算得来的。Ceph 的数据放置算法 CRUSH，是全局知晓的(Global Known)，所以在 Ceph 客户端可以独立的算出数据存放位置。这种分离方式简化了系统的设计，减轻了元数据集群的负载，在一些场景，如 LOSF(Lots of Small Files)应用时优势十分明显。

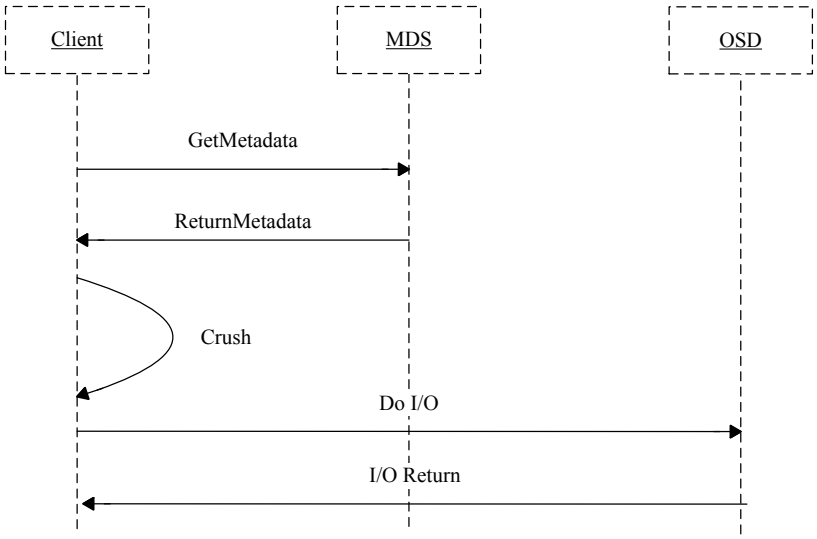

图 2.2　Ceph 的文件 I/O 顺序图

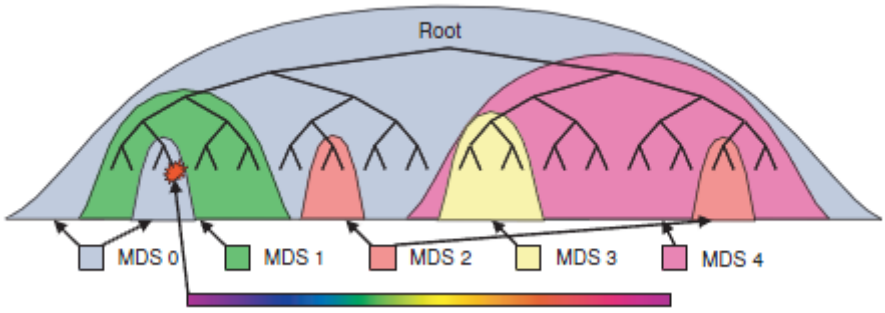

图 2.3　Ceph 的元数据管理

(2) 动态元数据管理

在文件系统的工作负载中，元数据负载占了很大部分的比例，因此，有效的元数据管理对系统的性能起着决定性的作用。Ceph 使用了一种新颖的元数据管理算法，即动态子树划分(Dynamic Subtree Partitioning)[14]算法，如图 2.3 所示。文件系统的命令空间在整个元数据服务器集群中被划分为多棵子树，并且划分是动态

的，自适应的，根据负载的变化而更改，从而达到元数据服务器的负载均衡，减少热点的目的，且在一定程度上保留元数据的局部性，减少元数据访问的开销。

(3) 基于 CRUSH 的数据放置

基于 CRUSH[15]算法的数据放置是 Ceph 的重要特点之一，是保证 Ceph 的高可靠性与高扩展性的重要措施。CRUSH 算法继承了 RUSH 算法的特性，与一致性哈希(Consistent Hash) [16]算法类似，在面对集群节点频繁的进入与退出时，CRUSH 使得需要迁移的数据量尽量减少，从而减少了数据重新映射时对系统的影响，提升扩展性。此外，CRUSH 算法还能够数据节点集群的层级关系，及用户指定的副本放置策略产生伪随机的数据分布，有着很强的实用性。关于 CRUSH 算法本文在本章第二节中有更为详细的描述。

(4) 统一存储

Ceph 提供了包括文件存储、块存储和对象存储的统一存储，如图 2.4 所示。其中，Ceph FS 可以作为通用文件系统使用（早期版本中，Ceph 利用 FUSE[17]提供文件系统的操作接口，如今 Ceph 已经被作为通用文件系统集成到 Linux 主线内核[18]），提供 POSIX 兼容的接口；Ceph RBD 可以作为通用的块设备使用，尤其在作为虚拟机环境中有着广泛的需求；RADOSGW 则可以为应用提供对象存储服务，并与亚马逊的 S3 [19]和 OpenStack 平台的 SWIFT [20]兼容。三种存储方式都是逻辑接口，作为后端存储的是 RADOS，不管是来自 Ceph FS、RBD 还是 RADOSGW 的数据，最终都是以对象的方式存储于 RADOS。RADOS 是 OSD 和监控器组成的集群，利用了 OSD 节点的计算能力实现了自动故障监测与修复、自动同步等功能，为上述的三种存储提供了可靠的存储后端。三种存储前端通过调用 librados 库与 RADOS 交互。

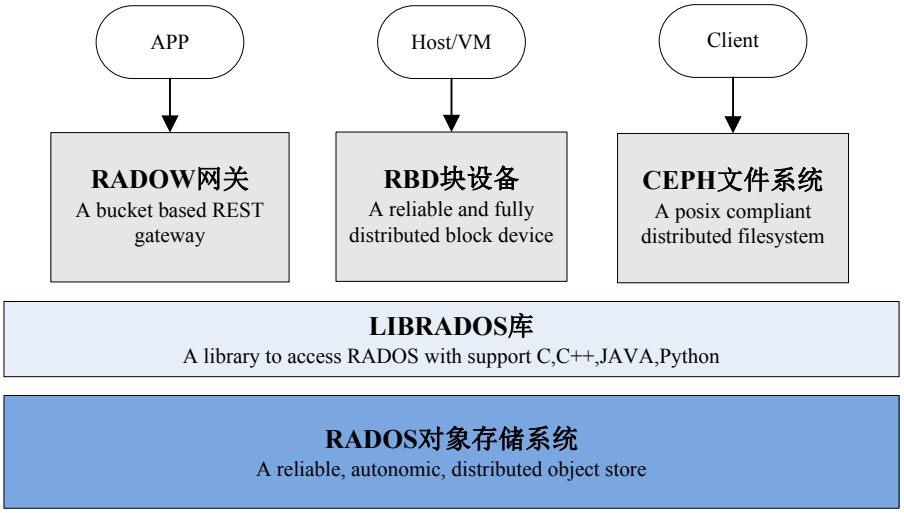

图 2.4　Ceph 的统一存储架构

2.2 Ceph 的数据布局

2.2.1 Ceph 的数据放置过程

Ceph 系统中，无论是文件、块设备还是对象数据，最终都在 RADOS 中以对象的方式存放到对象存储设备上。具体的数据放置过程为：文件和块设备首先被分割(Stripe)成多个数据对象；接着，数据对象会被映射到不同的逻辑放置组 PG（Placement Group）中，这一映射过程由简单的哈希函数映射函数完成。最后，PG 及其副本将被放置到多个 OSD 当中，此放置过程基于 CRUSH 数据放置算法。

2.2.2 CRUSH 数据放置算法

作为 Ceph 存储的的关键技术之一，CRUSH 算法由 Sage A.Weil 于 2006 年提出。CRUSH 来源于 RUSH 算法，能够基于伪随机的哈希函数产生确定的均匀的数据布局，且能够在集群出现新加节点或者故障节点的时候尽可能地减少数据的迁移。但与 RUSH 算法不同的是，CRUSH 算法是用户可控的（Controlled），能够适应不同的存储网络环境，从而更加实用。

（1）CRUSH 算法的输入与输出

CRUSH 的输入包括了对象 id、Crushmap 和数据放置规则。在实际输入时，由于数据对象的数量可能非常庞大，直接使用对象 id 输入可能带来较大的开销，所以会使用经过映射的 PG 的 id。CRUSH 的可控性体现在两个地方，一是 Crushmap，能灵活地描述出存储集群环境，二是放置规则，可以让用户自行配置适合的放置策略。CRUSH 结合输入 PG 的 id、Crushmap 和放置规则，产生一组有序的 OSD 作为输出，如图 2.5 所示。

Crushmap 用于描述存储集群的层级关系，通常用单树表示，其中包含了 bucket 和 device 类型的节点，每个节点都被指定了唯一的 ID 和权重。其中，bucket 节点通常用于描述故障域，如主机、机架、机柜等（副本分布于不同故障域，是 Ceph 保证数据安全的重要措施），可以包含其它层级的 bucket 和 device 节点，Bucket 有四种类型，对应了不同的放置算法。Device 节点用于表示对象存储设备(OSD)，只能作为叶子节点。大型的存储系统中，存储节点可能是异构的，有着不同的空间、性能，为此 Crushmap 中可以指定节点的权重，以表示节点的能力，从而影响 CRUSH 产生更合理的数据分布。这些要素使得 Crushmap 可以被灵活地配置，反映不同的存储集群环境，具有很强的实用性。

数据放置规则用于描述数据副本个数及放置策略。在 Ceph 中，存储管理员可以为不同的需求，配置适用的放置规则，如类似于 Raid 1 的两路镜像放置策略，或者将 3 个副本位于不同机架的放置规则等。放置规则的描述语言由 take、

select、emit 语句组成，其中 take 和 emit 分别用于表示规则的开始和结束，select 则用于指定具体的放置规则，如 select(3,cabinet)表示将数据的 3 个副本放置于 3 个不同的机柜中。

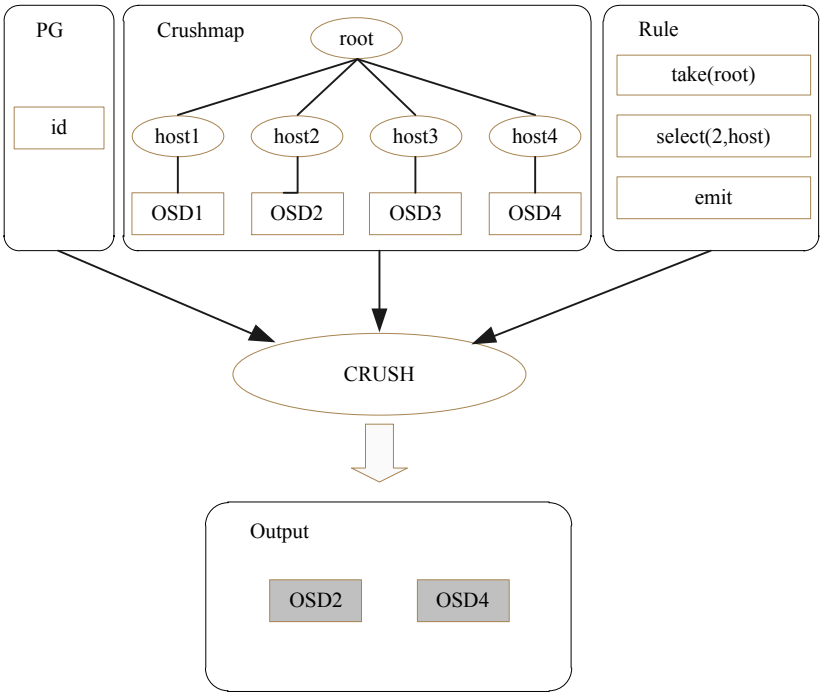

图 2.5　CRUSH 算法的输入与输出

（2）CRUSH 算法的数据放置

由于使用了 rjenkins1 哈希函数，CRUSH 的数据放置结果是确定性的，可重现的，但同时也表现出伪随机性，因为对于相似的输入来说，其输出之间没有明显的联系。在具体可以对输入 PG 的 id 进行处理时，有四种算法可以选择，并且都是依次在集群层级中迭代，直至选出合适的 OSD 为止。四种算法分别对应了 Crushmap 中的四种 bucket 类型。对于这种四种算法的优缺点比较如表 2.1 所示。

表 2.1　四种放置算法的比较[15]

类型	Uniform	List	Tree	Straw
时间复杂度	O(1)	O(n)	O(log n)	O(n)
添加节点	差	最好	好	最好
删除节点	差	差	好	最好

Uniform 类型的 bucket，在数据放置时，直接对输入 id 做哈希，直至确定合适的 OSD。这种算法速度最快，但是在存储节点发生故障退出或者新加节点的时候，会发生数据的重新映射，且数据迁移量最大。

List 类型的 bucket，在数据放置时，则考虑了 bucket 的权重。同一层级的 bucket

会被组织成一条 List，按照进入集群的早晚排序。输入 id 首先通过哈希得到一个范围属于(0,1)的值，再将这个值与每个 bucket 的累计权重比 W’（bucket 及链表中在其之前所有 bucket 的权重比之和）进行比较，进而选出一个合适的 bucket，满足该 bucket 的累计权重比是小于且最接近该哈希值的一个。选出的 bucket，将被放进工作集，进入到下一层级的迭代，直至选出合适的 OSD。List 类型 bucket 的数据放置如图 2.6 所示。图中的数据块，在新节点加入时，只有 x3 发生了迁移。事实上，List 类型 bucket 的数据放置算法，在节点增加时，需要迁移的数据是最少的。

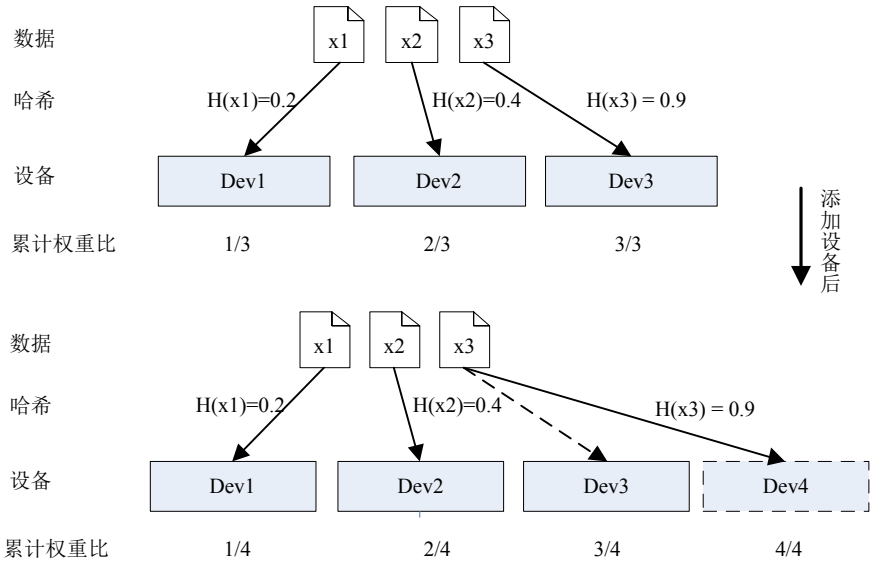

图 2.6　List 类型的放置算法

Tree 类型的 bucket，数据放置与 List 相似，只是在组织 bucket 时，使用了树结构，这使得在定位所属节点所属区域时更加迅速。虽然在增加节点时，数据迁移的表现不如 List 类型，但其在节点退出时，数据迁移量却更少。

Straw 类型的 bucket，这种类型仍是基于在 List 和 Tree 类型的思想，并进行了启发式的优化，使得数据迁移的表现最优。Straw 类型的 bucket 是目前 Ceph 系统中使用最多的类型。

CRUSH 对于同一对象的副本放置则更多处于安全考虑。在真实环境中，节点故障的发生往往具有相关性，特别是对于那些使用共同电源供电或是通过同一交换机连接的节点，往往可能同时发生故障。因此，在分布式存储领域，对于这些节点，认为他们处于同一个故障域(Failure Domain)。CRUSH 在 Crushmap 用 bucket 很好的体现了故障域划分，在放置时，副本编号 r 将被用作参数计算，类似于 c(r,x) = (placement(x)+rp)，p 为质数，这使得数据的不同副本被放置到不同的故障中，从而保证了数据安全。

在大型存储集群中，节点数目成千上万，节点的添加和故障是很常见的，而

因此产生的大面积的数据迁移必将带来灾难性的影响。Ceph 的数据放置算法可以有效地解决这个问题，进而让 Ceph 具有很强的扩展性，基于故障域的副本放置，减小了数据副本同时失效的可能，则提高了保证了系统的可靠性。

2.3 分布式存储节能技术分类

分布式存储的节能技术可以分为两大类即硬件节能技术和软件节能技术。其中，硬件节能技术分为磁盘节能技术和硬件节点节能技术，主要在硬件层面，以减少每个节点上各硬件组成部分的能耗为主要手段。软件技术分为基于节点管理的节能技术，基于数据放置的节能技术，基于数据调度的节能技术和基于缓存预取的节能技术[21]，在软件层面，通过对整个集群的配置或管理来降低能耗，提升能耗使用效率。

2.3.1 硬件技术

（1）磁盘节能

机械硬盘仍是目前使用最为广泛的存储介质，减少机械硬盘的能耗对于磁盘、磁盘阵列一级分布式存储系统的节能都是最基本、最有效的方法之一。从硬盘的组成来看，其包含了磁盘片、磁头、磁头臂、永磁铁、音圈电机、主轴马达和空气过滤片等部件。其中主轴马达是硬盘功耗的主要来源，主轴马达的转速将直接影响到磁盘的功耗状态。以磁盘的三种状态即：活动状态、空闲状态和待机状态为例，当磁盘处于活动状态（数据读取/写入）时，磁头进行寻道、定位和数据存取，主轴高速旋转，磁盘能耗最大；当磁盘处于空闲状态时，磁盘盘片仍然保持高速旋转状态，但没有磁头的定位消耗，此时磁盘的能耗稍低；而当磁盘处于待机状态时，主轴动机停止旋转且磁头静止，此状态下的磁盘能耗最低。利用以上原理，Gurumurthi 等人提出了更比传统的 TPM 更为精准的的磁盘动态转速（dynamic rotations perminute，DRPM）策略[22]，使得磁盘可以在不同的访问频率下以不同的速率旋转，从而有效地降低磁盘的能耗。Timothy Bisson 等人提出了一种 Ceph OSD(对象存储设备)的自适应的磁盘降速算法[23]。他们用在系统负载比较轻时，较多的磁盘降速，进入节能模式；系统负载较重时，则只有少部分进入节能状态。由于他们的工作针对的只是部分 OSD 上的磁盘的节能，OSD 节点本身仍处于活跃状态，因此对整个系统能耗的影响是有限的。

（2）服务器节能

分布式存储的节点通常是完整的服务器，因此，降低每台节点服务器的能耗，也是整个分布式存储系统节能的有效措施。这类技术主要以构建低功耗的节点为主，在选择节点的处理器时，放弃传统的高功耗处理器，如 Intel 志强系列等，而

选择功耗更低，或是效能更高的芯片，如基于 ARM 的 CPU 及 Intel 凌动系列等；在选取存储设备时则倾向于存取效率更高、功耗更低的固态硬盘（solid state drive，SSD)。Lim 等人采用了移动计算领域常用的低价格、低功耗的处理器并使用功耗较低的 SSD 设备作为磁盘缓存，从而提升了原数据仓库(Warehouse)计算集群的性能成本比，提高了能耗利用率[24]。与此类似，Szalay 等人在观察到分布式存储应用中，对节点的存储能力要求较高而对计算机能力要求较低的特点后，采用能耗较低、但效率也较低的 CPU 和效率更高但功耗较低的固态硬盘来构建数据中心集群[25]，实验结果表明，这种构建方式在数据密集型应用环境中，能够保证性能的同时有效的减少系统的能耗。

2.3.2 软件技术

不同于硬件节能技术，软件节能技术一般是通过对整个集群的管理降低能耗，提升能耗使用效率，从节点的工作状态角度来进行节点管理，或是从数据的角度进行数据管理。

(1) 基于节点管理的节能技术

节点管理技术的主要原理是，在不影响或者较小影响集群性能的情况下，尽可能地减少运行节点的个数和运行时间。

起转令牌(Spin-Up Tokens)是限制节点运行数目的有效机制，常用于归档系统。这种机制在集群中指定一定数目的节点为长期活跃的，而其他剩余节点则处于休眠或关闭状态，非活跃状态的节点如果需要变成活跃状态，需要获得起转令牌，集群中起转令牌的个数是有限的，各节点通过一定的规则去竞争令牌。这种机制对性能会产生一定的影响，但是带来的节能效果却是可观的，Storer 等人开发的 Pergamum [26]在采取起转令牌的情况下，增加了约 30%的 IO 开销，但却只需要让 5%的节点保持活跃。

写卸载(Write off-load)机制则能减少节点运行的时间。Narayanan 等人[27]观察到企业级负载环境中存在以下情况：企业级存储可以利用处于空闲状态的时间来节能；企业级负载中大部分的读操作是可以通过缓存来完成的。基于以上两点，他们将目标为空闲状态节点的写操作重定向到活跃的节点，从而延长那些空闲节点的节能状态持续时间，以此减少其能耗。在基于负载重现的实验评估中，写卸载技术可以节省系统 20%以上的能耗。

覆盖子集(Coverying Subset) [28]机制同样是减少集群中活跃节点的数目，不过在确定活跃节点时，考虑到了数据的可用性，即需要保证数据一个以上的副本是活跃的。在覆盖子集机制下，集群节点被划分为覆盖子集和非覆盖子集，覆盖子集中的节点处于活跃状态，其余节点处于非活跃的状态。覆盖子集的节点个数多

少将会影响到节能的比例和性能，在 Hadoop 集群中，一种建议的覆盖子集占总节点数比例为 10%-30%。

(2) 基于数据放置的节能技术

基于数据放置的节能技术是对分布式存储中数据放置策略的优化。在传统的分布式存储系统环境中，数据放置的策略往往是性能和扩展性为目的的。而基于数据放置的节能技术则是加入对节能的考虑，通过优化数据及其副本（或纠删码）在存储节点间的位置来获得更高的节能效果。基于数据放置的节能技术往往利用数据的冗余性，使得系统的部分磁盘节点能够在一定的时间内可以关闭或者休眠，从而达到节能的目的。比如, Sara Arbab 等人发现 HDFS 中，副本放置的策略为随机的，同时应满足一个数据有两个副本在同一个节点上，这种数据放置策略在数据块个数较大时，可以关闭节点的个数是有限的，因为很多关闭的节点很有可能由于读写请求而重新开启，不能够达到理想的节能比例。于是，他们提出了一种镜像数据块复制策略（Mirrored Data Block Replication Policy）[29]，即各节点之间是以互相镜像的方式存放数据副本的，这会使处于休眠状态的节点，在有数据访问时不需要开启，而是由活跃的镜像节点提供服务。实验结果显示,这种策略能够有效的减少 Hadoop 集群的能耗，而且在数据块越多的情况下越明显。

(3) 基于数据调度的节能技术

基于数据调度的节能技术是在系统工作过程中，根据访问模式或访问频度等特征动态调整数据存放的位置，尽可能合并读写请求到数量较少的节点上，使其它节点可以被关闭或者休眠，从而达到降低能耗的目的。文献[30]用数据活动因子来量化描述 HDFS 中数据块的访问热度，并将 RACK 划分为 Active-Zone 与 Sleep-Zone 两个区域，在确定数据位置时，热度较高的数据块尽可能地放到 Active-Zone，热度较低的放到 Sleep-Zone。由于系统在运行过程中，各数据块的热度可能是动态的，其活动因子不停地变化，使得数据块在集群中的位置会被周期性的调整，从而在保证一定的节能比例的同时，适应系统的负载规律，减少对系统的性能及服务质量的影响。相对于基于数据放置的节能技术，数据调度节能技具有动态、自适应的优点，但由于数据布局需要重组织，存储网络中可能出现较大的非目标流量。

(4) 基于缓存预取的节能技术

与磁盘相比，内存或者 SSD 有着更低的功耗，从而可以通过用内存或者 SSD 来缓存磁盘中热度较高的数据，将对磁盘的访问转移到缓存上，减少磁盘的工作时间，从而减少能耗。基于缓存预取的节能技术正是基于这一基本思想，对磁盘中的数据热度进行统计，并采使用一定的预测方法，将访问频率较高的数据预先放到缓存中，使原磁盘进入节能状态。Zhu 等人提出了一组缓存管理算法[31]，相

对于原有的离线(off-line)和在线的(online)缓存更新算法，新提出的算法是能耗敏感的。实验表明，相比于传统的 LRU 缓存更新算法，他们的算法能够节省 16%的磁盘能耗，同时，对于 OLTP 类型的工作负载，平均反应时间能优化 50%左右。 然而，基于缓存预取的节能技术也存在一些局限性，一方面，缓存技术多用于读多写少的访问模式下，对于写密集的应用场景，很难发挥出优势；另一方面，由于 SSD 或内存设备在空间价格比上于机械硬盘还是有较大差距，因此，对于需要缓存的数据量较大时，成本是必须考虑的因素。

2.4 功耗管理模型

2.4.1 覆盖子集模型

覆盖子集模型是由斯坦福大学的 J. Leverich, 和 C. Kozyrakis 提出的，被应用在多个 HDFS 节能研究中[28][29][32]。Leverish 等人通过一段时间的观察，发现 HDFS 集群中有很多节点在大部分时间内，资源的利用率是很低的，如 CPU，在接近一半的时间内利用率不到 30%。因此他们提出了一种方法只需要保留一小部分的节点提供服务，其他的则可以关闭来节省能耗。提供服务的这部分节点即称为覆盖子集 CS，其余的为非 CS，如图 2.7 所示。

覆盖子集模型中，CS 节点是始终开启的，并且为保证数据的可用性，CS 中放置了所有数据的一个副本。在系统的负载较低时，为了节能，非 CS 将全部处于关闭或者节能状态，而当系统负载升高，单靠 CS 节点已经不能满足要求，所以非 CS 节点将开启，以满足负载的需要。

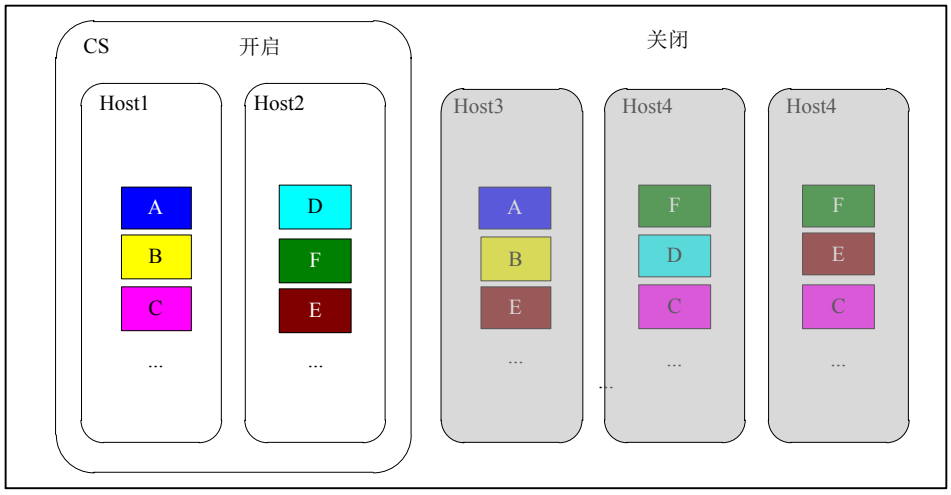

图 2.7 覆盖子集模型

2.4.2 冷热区划分模型

冷热区域划分模型（Hot Zone and a Cold Zone），是 UIUC 的 Kaushik, R.等人提出的，应用于 GreenHDFS 中[33]。该模型没有从数据冗余角度去实现功耗管理，而是基于数据冷热度的。如图 4.2 所示，根据数据的访问频率，数据被分为热数据和冷数据，而存储节点也被划分为热区（HotZone）和冷区(Cold Zone)。其中，热区存放热的数据，保持正常工作状态；冷区，存放冷的数据，保持节能状态。这种功耗管理模型中，节点的状态是保持不变的，而数据将会根据热度的变化而动态地调整所属的节点，以满足负载需要。

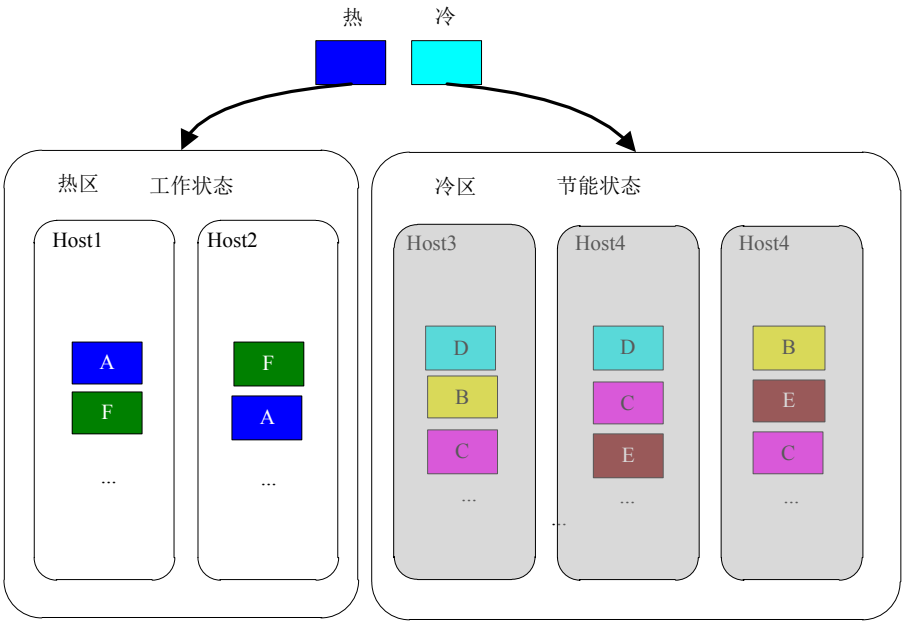

图 2.8　冷热区划分模型

2.4.3 档位切换模型

档位切换（Gear Shifting）模型由 FSU 的 Weddle, C 等人提出，被应用在 PARAID[34]，微软剑桥研究院的 Sirrea[35]，CMU 的 Rabbit[36]等存储系统中。受到汽车换挡技术的启发，在档位切换模型中，存储节点被划分为多个档位级别（Gear Level），档位级别越高则包含越多的存储节点，最高档包含了所有的节点。从数据的角度看，每个档位级别中则包含了数据的一个副本。档位切换模型如图 2.9 所示。

与汽车中的换挡常因为路况和速度的需要类似，档位切换模型中档位的切换则来自于负载状况。当负载要求系统提供更高的 I/O 并行度及带宽时，系统会切换到较高的档位级别，以提供能够满足负载要求的节点数目；而当负载较轻时，系统会切换到较低的档位级别，只保留低档位的节点开启以提供服务，其余节点则进入节能状态，从而减少整个系统的能耗。

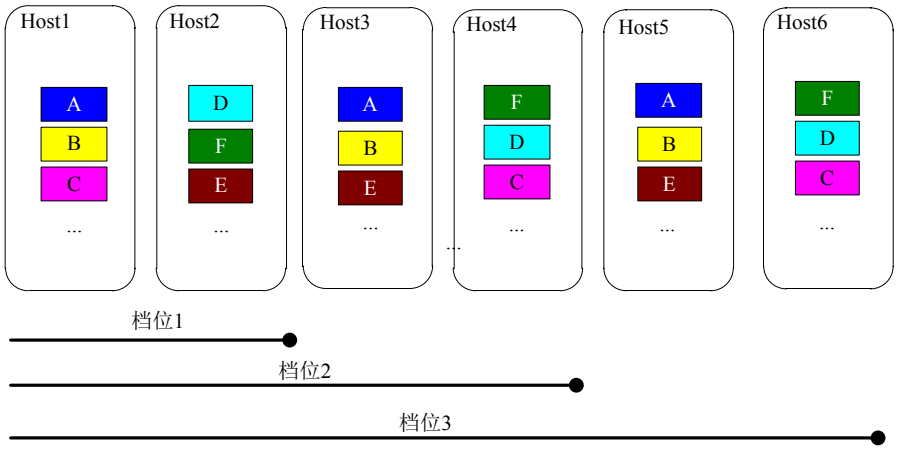

图 2.9　档位切换模型

2.5　本章小结

本章首先对 Ceph 的系统架构和技术特点做了全面的介绍，其架构设计优秀高效，技术特点也与当前的趋势十分吻合。接着介绍并分析了 Ceph 的数据布局，分析其数据及副本放置的特点，为下一章基于数据布局的节能优化提供依据。之后对分布式存储常用的节能技术进行了综述与分类，最后详细介绍了常用的功耗管理模型，为第四章的研究打下基础。

第三章　面向节能的数据布局优化

数据布局体现了数据在节点间的放置策略，是分布式存储系统设计时所需要考虑的重要因素。Ceph 基于 CRUSH 算法的数据布局，是其具备高可靠性、高性能及高扩展性的基础，然而这种数据布局同时也限制了 Ceph 系统节能的能力。针对 Ceph 现有的数据布局，进行面向节能的优化，是实现整个 Ceph 系统节能的必要工作。本文从节能比例和数据的可用性等方面分析了 CRUSH 算法数据布局的不足，并提出了 PGEO 数据布局优化算法，使得 Ceph 系统在达到高节能比例的时候，仍能保证数据的可用性。

3.1　问题分析

3.1.1 CRUSH 的副本分布

Ceph 基于 CRUSH 算法的数据布局有以下特点：数据的分布呈现出均匀的随机分布的特征，以保证存储空间的有效利用及避免访问热点的出现；同一数据的不同副本被放置到不同的故障域，以保证数据的可靠性。举例来说，数据的副本可以被放置到多个不同的机架上，益处是当其中一个机架发生故障，如网络故障、电源故障等，此机架上的所有存放数据的副本任然可以在他机架上被找到并提供服务。此外数据副本被放置到哪一层的故障域，也是可以由管理员通过规则语言指定的，例如可以指定副本放置策略为 select(2,host)即将数据的 2 个副本放置于不同的主机上，图 3.1 描述了这种副本策略的数据及副本分布。

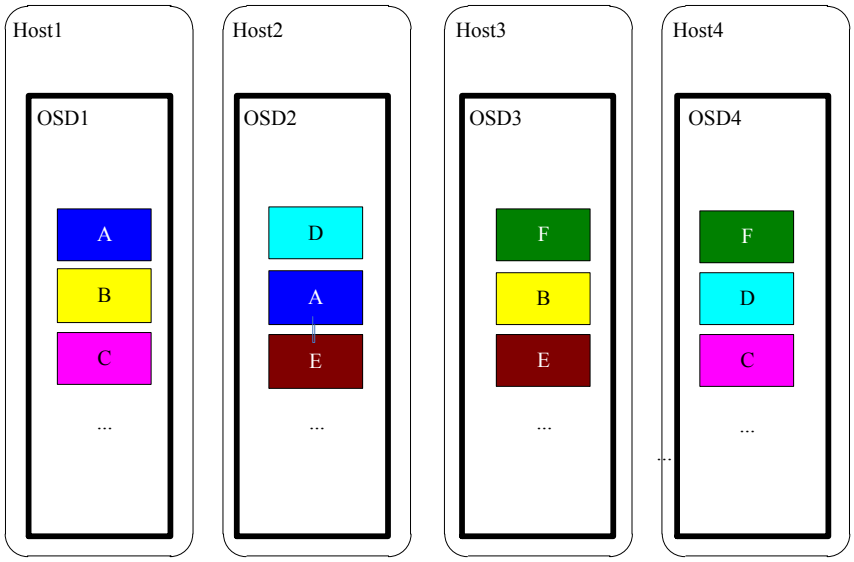

图 3.1　CRUSH 的数据及副本分布

3.1.2 问题分析

数据可用性是存储系统的重要性质之一，是进行节能优化的前提。如果发生不可用数据的访问，则一方面，对于客户端，会产生非常大的读写延迟，对于时间敏感的应用来说几乎是不可接受的；另一方面，为了保证最终访问请求的完成，已经关闭的节点会会再次启动，启动过程会产生不可忽视的额外的能耗。所以基于这个前提，分析了 Ceph 的数据布局存在的问题。

在一个 Ceph 集群中，OSD 节点数为 N，在 Crushmap 的某层被划分到 n 个故障域 fd 中，每个故障域包含的 OSD 节点个数为 N/n，副本个数为 r，放置策略为 select(r,fd)。定义一个集合 Mi 为数据 i 的副本所在故障域的集合。则在数据量足够大时，所有 Mi 的集合个数 C1 为：

$$C1 = C_n^r \tag{3.1}$$

在系统低负载时关闭其中任意的 n’个故障域以节能，则在这些关闭故障域中，包含了部分数据，其全部的 r 个副本均被关闭。n’个被关闭的故障域中，这些不可用数据的个数 C2 为：

$$C2 = C_{n'}^r \tag{3.2}$$

所以在关闭 n’故障域以节能时，对于任一数据所有副本均被关闭的概率为：

$$pro = \frac{C1}{C2} = \frac{n'(n'-1)...(n'-r+1)}{n(n-1)...(n-r+1)} \tag{3.3}$$

为保证数据的可用性，即任意数据至少有一个副本是可用的，要求 pro 为 0，此时 n’应满足：

$$n' <= r-1 \tag{3.4}$$

不等式 3.4 意味着，在 Ceph 系统中的低负载时可以任意关闭的故障域的个数不能等于或者超过数据副本的个数，否则会出现部分数据的不可用。以图 3.1 中的副本分布为例，可以关闭的主机个数为最多只有 1 台，同时关闭 2 台以上主机时，A 到 F 中至少一个数据块无法访问。在最多只能关闭 r-1 个故障域的情况下，Ceph 系统可以达到最大的节能比例为：

$$p = \frac{(r-1)*N/n}{N} = \frac{r-1}{n} \tag{3.5}$$

随着集群规模的增大，故障域的个数会越来越多，而副本个数 r 通常保持不变且相对于故障域个数较小，所以系统能够达到的节能的效果是微乎其微的。事实上，在数据副本为 3，分布于不同机架，则在集群节点的规模超过 600 个时，理论上系统能够达到的节能比例不超过 1%的。此外，系统的低负载时并不是一直存在的，实际上能够节省的能耗会更少。

3.2 PGPEO 数据布局优化算法

针对上文提出的 CRUSH 数据布局在节能方面存在的问题，提出 PGPEO(Power Group Partition based Energy-Saving oriented Optimization)数据布局优化算法。

3.2.1 算法基本思路

系统的节能比例是优化算法的主要目标，但同时还应考虑算法对系统产生的影响，所以在 PGPEO 算法设计时考虑到以下因素：

（1）节能比例与数据可用性

Ceph 的数据布局在满足数据可用性的前提下，节能比例较小的主要原因是其全局随机的副本分布。数据副本可以随机存放在任何的故障域中，使得在关闭一定数量的节点时很大可能覆盖到部分数据的所有副本，所以最直的优化思路是减小副本放置的范围。副本放置范围的限定可以通过将故障域分组，或者是通过 CD(Chained Declustering[37])策略实现。分组策略可以限定数据的某一副本只可能出现在一个组中，从而可以同时关闭不同组中的节点同时保证数据可用性。CD 放置策略，在选定第一个副本位置之后，剩余的副本则依次放置到后续的位置上，在关闭节点时，只要不同时关闭连续的个数大于副本个数的节点时都可以保证数据可用性。

（2）数据恢复并行度

数据恢复的并行度重要性主要体现在两点：一是在节点发生故障后，能够从尽量多的节点恢复数据，减少数据失效时间；二是在节能状态的节点进入工作状态时能够迅速同步数据，以尽快提供服务。通过数据布局以增加数据恢复并行度的主要思想是将副本尽量分散到多个节点，从而增加了访问的并行度。然而，由于存储集群的网络带宽的限制，数据恢复的并行度是有限的。同时，这一目标与为提高节能比例而限制副本分布范围的策略某种意义上是冲突的，所以需要折衷的考虑。在节能比例相同的情况下，分组的优化策略比 CD 策略数据恢复并行度更高。

（3）数据可靠性

数据布局对数据可靠性的影响体现在将副本放置于不同的故障域，所以在数据布局的节能优化时应尽量保留原副本策略保证的数据可靠性。Ceph 中故障域是分层次的，越高的故障域层级可以保证越高的数据可靠性，所以保证原策略的可靠性的基本思路就是优化后副本的放置范围不能低于优化前副本放置的故障域范围。举例来说，原副本分布策略是 select(3,rack)，那优化后，数据副本仍要保证至少分布在不同的机架上。

（4）可配置性

CRUSH 的灵活性体现在其副本策略是可控的，CRUSH 中用于描述集群层级关系 Crushmap 和副本策略描述的规则语言都提供了很大的配置自由度。因此，一种合理的方法是尽可能的利用 CRUSH 的配置接口，来实现优化的数据布局策略，无需直接改动 CRUSH 算法，以免影响其他特性。

基于以上分析，本文提出基于功耗组划分的 PGPEO 优化算法。一组故障域，其中所有 OSD 节点在功耗管理的过程中均处于相同的功耗状态（节能或者正常工作），则称这一组故障域处于相同的功耗组(PowerGroup)。通过划分 r(副本个数)个功耗组，并保留数据在组内的随机分布，使得此数据布局策略能兼顾节能比例与数据恢复并行度；被分组的直接对象为原副本策略指定的故障域，以保留原副本策略的语义，和因此带来的数据可靠性；通过自动分析并修改 Crushmap 在其中引入 PowerGroup 一级的 bucket，和添加新的副本策略来实现分组功能，从而充分利用 CRUSH 的可配置性。

3.2.2 算法描述

PGPEO 数据布局优化算法的伪码如下：

算法 3.1 PGPEO 数据布局优化算法

输入：原系统的 crushmapbin 二进制文件

输出：优化后的 crushmapbin 二进制文件

```
1：crushMap = decompileCrushmap(crushmapbin)
2：topType = 0
4：pgs[r] = NULL
5：fdSelected = getChooseFromRule( )                /*获取被用到的故障域*/
3：addCrushmapLayer(crushMap, PowerGroup)           /*添加 PowerGrou 的层级*/
6：for fd in fdSelected do                          /*获取最高层级的 fd*/
7：    if fd.type > topType then
8：        topTpye = fd.type
9：    end if
10：end for
11：fdBuckets = getBucketsByLayer(fdTop)            /*获取层级为 fdTop 的 bucket*/
12：pgPartition(pgs,fdBuckets)                      /*将目标故障域分组* /
13：newRule = ’choose(r,PowerGroup)’                /*新的副本规则*/
14：addNewRule(newRule)                             /*添加新的规则*/
15：enableRule(newRule)                             /*启用新规则*/
```

如算法 3.1 所描述，PGPEO 的主要流程为：反编译 Crushmap 的二进制文件 crushmapbin，得到其可编辑的文本文件 Crushmap；向 Crushmap 中添加 PowerGroup 的层级，并初始化长度等于副本个数 r 的 PowerGroup 数组；匹配 Crushmap 文件中的所有用到的副本放置规则和相应的故障域；遍历用到的故障域，并选出层级为最上的故障域 fdTop；获取所有类型为 fdTop 的 bucket，作为被划分的目标故障域集合 fdBuckets；按照一定的策略将 fdBuckets 划分进 r 个 PowerGroup 中；生成新的基于功耗组的副本放置规则并添加到 Crushmap 中；重新编译 Crushmap 到二进制文件，并在 Ceph 集群中启用。

经过 PGPEO 算法的优化之后，数据副本在放置到原指定的故障域之前，将会被划分进不同的功耗组，很大程度上限制了分布的范围，这允许同时关闭不同功耗组的节点，而不影响数据可用性。仍以图 3.1 中的数据布局为例，经过优化之后，新的数据分布如图 3.2 所示，可以分别关闭 2 个功耗组中的任何一个主机同时保证所有数据是可访问的。

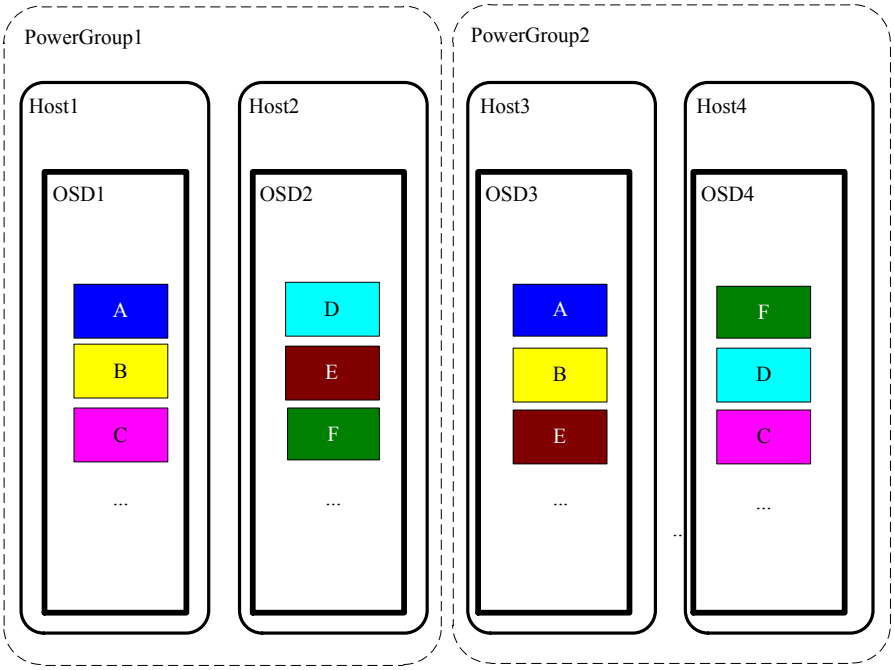

图 3.2　优化后的数据分布

3.2.3 分组算法

在 PGPEO 算法中，具体实现故障域划分进不同功耗组的是 pgPartition 过程，考虑到分组结果对负载分布、存储能力利用率等方面的影响的，本文提出了三种分组算法以满足不同的需求。

（1）轮转分组

轮转分组的思想是故障域在分组时，每隔 r 个的故障域被分到同一个功耗组

中。这种分组方法的益处是在系统节能的时候，剩余的活跃的节点仍能够均匀的分布在不同更高的故障域上，使得负载能够在其中均匀分布，同时每个故障域的散热压力也是均衡的。在 Crushmap 中，对于每个故障域及 OSD 节点都有唯一的 id 描述，且一般为顺序指定，所以在实现轮转分组算法时根据故障域 id 对副本个数 r 取模的结果划分进不同的功耗组。轮转分组算法的伪码见算法 3.2。

算法 3.2 轮转分组算法

输入： 目标故障域集合 fdBuckets
空功耗组数组 pgs[r]
输出： 包含了权重和 item 的功耗组的数组 pgs[r]

```
1：sortByID(fdBuckets)                    /*按 ID 排序*/
2：for i=0 to i<r do
3：    for every fd in fdBuckets do       /*遍历所有需划分功耗组的故障域*/
4：      if (fd.id mod r) == i then       /*根据 id 分组*/
5：         pgs[i].items.append(id);
6：         pgs[i].weigh += fd.weight;    /*功耗组权重计算*/
7：      end if
8：    end for
9：end for
10：return pgs
```

（2）子网分组

根据故障域所在的子网进行分组的原因是，一方面可以在节能的时候同时关闭子网内的交换机等网络设备，会获得更好的节能效果；另一方面，同一子网内部的节点可能同时发生故障（如交换机故障，网线断开等），所以尽量将副本划分到不同子网，保证数据安全。此外，这种划分也可以使得负载在各子网间均衡，避免某一子网内出现拥塞。子网分组的策略是根据 fd 内节点的 IP 地址，同一功耗组内的故障域，其 IP 地址是最接近的，子网分组算法的伪码见算法 3.3。

需要注意的是，在进行 IP 分配时，可能由于特殊原因位置相近（如位于同一机架或者机柜）的节点并未分配同一子网的 IP，这种情况下，子网分组算法可能不会达到真正的子网分组的效果，因为优先保证原分组策略中指定的故障域划分。

（3）权重均衡分组

权重均衡分组的目的是适应节点异构的场景。在规模的存储集群中，可能因为设备类型的不同以及购买时间的不同，导致每个节点可能具有不同的存储能力，如存储空间、IOPS 等。在 Ceph 中，用权重 weigh 来描述节点的存储能力。由于划分功耗组时，r 个功耗组均要承担同样的访问数据量，会使某些存储能力强的设

备无法充分利用，同时限制了集群的存储空间利用率。举例来说，未考虑权重均衡时，可能会出现这样的分组，第一组的所有节点均为较新购进的，空间更大，I/O 能力更强，其累计权重为 4.00，而第二组和第三组的空间较小，权重分别为 2.00 和 1.50，那在数据分布时，由于每组都会写一个副本，所以三个组写入的数据量是相同的，集群的有效存储空间则为第三组的存储空间，且数据的写入速度也收到第三组 I/O 能力的限制。为解决这一问题，本文提出了权重均衡的功耗组划分算法。理论上，将集合 S 划分进 k 个组，使得每个组和的最大值最小，被称为 k 划分问题(K-Partition Problem[38])，一般意义上是一个 NP 困难问题。考虑到，大多数集群中，存储设备的型号是非常有限的，且经常多个设备型号相同，本文采用倒序贪心算法进行各组间权重均衡，伪码见算法 3.4。后续实验证此算法是可行的。

算法 3.3 子网分组算法

输入： 目标故障域集合 fdBuckets
Ceph 配置文件 CephConf
空功耗组数组 pgs[r]

输出： 包含了权重和 item 的功耗组的数组 pgs[r]

```
1：count = 0
2：i = 0
3：total = Count(fdBuckets)                          /*fd 集合大小*/
4：for every fd in fdBuckets do                      /*生成 fd 到 IP 的映射*/
5：    hname = getHostname(CephConf,fd.items[0])     /*获取 fd 中的主机名*/
6：    addr = getIPbyHost(hname)                     /*获取 IP 地址*/
7：    appendToMap(fd,addr,map)
8：end for
9：sortByAddr(map)                                   /*根据子网 IP 排序*/
10：for every mapitem in map do                      /*遍历映射并分组*/
11：    if count>(total/r) then                      /*分到下组*/
12：        count = 0
13：        i++
14：    end if
15：    pgs[i].items.append(id)
16：    pgs[i].weigh += fd.weight                    /*权重累加*
17：    count++
18：end for
19：return pgs
```

算法 3.4 权重均衡分组算法

输入：目标故障域集合 fdBuckets
功耗组数组 pgs[r]
输出：包含了权重和 item 的功耗组的数组 pgs[r]

```
1：revSortByWeight(fdBuckets)          /*按照权重倒序排列 fd*/
2：sortByWeight(pgs)                   /*按照权重排列 power group*/
3：for every fd in fdBuckets do        /*遍历所有需划分功耗组的故障域*/
4：    pgs[0].item.append(id)          /*权重最大的 fd 分到权重最小的 pg*/
5：    pgs[0].weigh += fd.weight       /*权重累加*/
6：    sortByWeight(pgs)               /*重新排列 pgs*/
7：end for
8：return pgs
```

3.2.4 算法复杂性

PGPEO 的算法复杂性主要来自于分组过程。其中轮转分组法的时间复杂度主要在于对 fd 集合的 r*n 次遍历，r 为副本个数，一般较小，所以可以认为其复杂度为 O(n)；子网分组算法中，对 fd 集合进行了两次遍历，复杂度都是 O(n)，所以子网分组算法的复杂度同样为 O(n)；权重均衡分组算法在遍历 fd 集合的过程中，会对 r 个 pg 进行排序，r 值较小，同样可认为其复杂度是 O(n)。空间复杂度方面，三者主要的空间消耗为 fd 的集合的存储。

由于 PGPEO 算法是静态的数据布局优化算法，只有在第一次开启节能状态时被执行，所以算法本身的复杂度带来的影响是非常有限的。优化后数据布局对 Ceph 系统的影响，本文在后续的实验评测中说明。

3.3 分析及实验评测

3.3.1 对比与分析

对比本文提出的 PGPEO 优化算法和其他优化策略的优劣，并分析了 PEGPEO 能够达到的节能效果。

（1）优化策略的的对比分析

从节能比例、恢复并行度、节能节点选择难易度等方面比较了本文提出的基于 PGPEO 算法的优化策略、镜像放置策略和 CD 放置策略，结果如表 3.1 所示。

从表 3.1 可以看出，三种优化策略均可以达到理想的节能比例。但是在集群从节能状态恢复至正常工作状态或者是故障恢复时，镜像策略只能从镜像组的其他

r-1 个节点恢复数据，恢复并行度较低，速度较慢；CD 策略可以从故障节点的前 r-1 个节点，以及后续的 r-1 个节点，共 2(r-1)个节点恢复数据；基于 PEPGO 算法的优化策略则可以从集群中其他功耗组的所有节点恢复数据，在集群规模较大时，优势十分明显。在选择节能的节点时，因为镜像只要求每个镜像组都有活跃的节点即可，目标节点很容易选择，功耗组只要保留一个完整的功耗组活跃，而 CD 不能同时关掉连续的 r 个节点。在算法实现时，镜像和 CD 策略改变 CRUSH 随机的根本特性，需要做较大的改动，且影响较大，而 PGPEO 优化算法则充分利用了 CRUSH 可控的特点，可以完全通过配置完成，在实现节能优化的同时，最大程度上保留了 CRUSH 算法的优点，相对于其他两种策略优势明显。

表 3.1　不同优化策略的比较

优化策略	节能比例	恢复并行度	节点选择	可配置
镜像	r-1/r	r-1	容易	是*
CD	r-1/r	2(r-1)	一般	否
PGPEO	r-1/r	(r-1)*N/n	容易	是

（2）节能效果分析

图 3.3 显示了基于功耗组划分的算法对系统节能效果的优化。

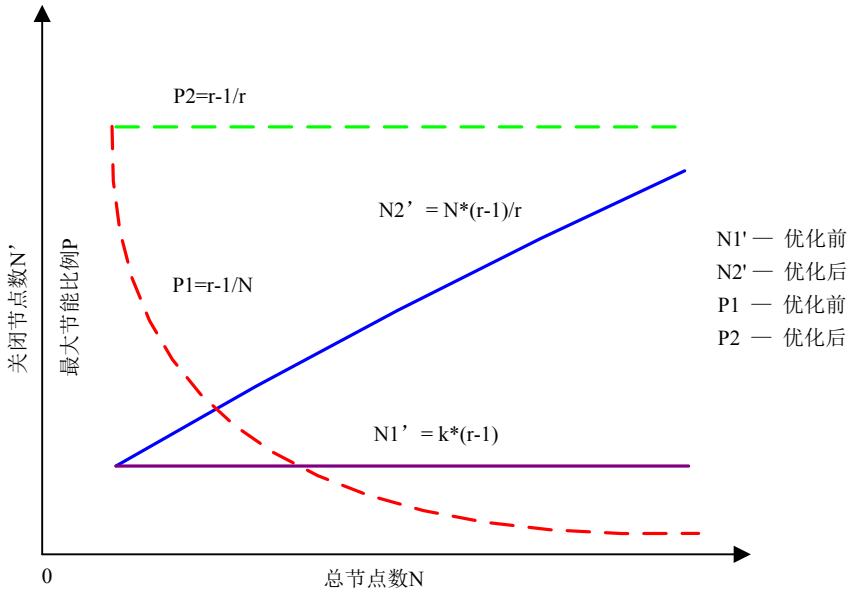

图 3.3　优化后的节能效果

经过优化后，数据块的副本分别位于 r 个不同的功耗组中，在保证数据集可用的情况下，则最多可以关闭 r-1 个功耗组，共 N*(r-1)/r 个节点，随着总节点数目的增加，可以节省的能耗数也呈线性地上升；而在优化前，集群中始终只有 r-1 个故障域的节点可以关闭，随着系统规模的增大，这一数值保持不变，节能的总量始终较低。与节能量相对应，优化后，系统能够达到的最高节能比例始终可以保持在较高的 r-1/r，而优化前，这一比例随着节点数的增加而减少，节点数目较大时，

趋近于零。因此，通过对 Ceph 的数据布局采取划分功耗组的优化后，系统的节能能力有了非常大的提升，集群规模较大时减少的能耗将非常可观。

3.3.2 实验评测

Crushtool 是 Ceph 系统中提供的用于管理和测试 CRUSH 数据分布的工具，可以模拟出 CRUSH 算法的分布效果。本文用 crushtool 测试 PEPEO 算法的优化结果及对 Ceph 的影响。

（1）Crushmap 正确性

PEPGO 算法能否正确被应用于 Ceph 集群中，很重要的一点是 Crushmap 文件优化结果的正确性。本文实现 PGPEO 优化算法后，对一个 Crushmap 文件 crushmap1.txt 进行了优化，优化的结果文件内容如下所示，其中粗体部分为优化后新加入的配置项。可以发现，powergroup 的层级已经被成功添加，并实现了对故障域（host）的划分。运行命令“crushtool –c crushmap1.txt –o map1”后无错误输出，且生成 map1 文件，且可以在 ceph 系统加载，所以证明了 PEPGO 算法优化 Crushmap 结果的正确性。

```
…
type 0 osd
type 1 host
type 2 powergroup
type 3 rack
…
powergroup pwg1 {
        id -8
        alg straw
        item host.1 weight 1.00
        item host.2 weight 1.00
}
…
Rule replicated_ruleset {
        …
        Step choose firstn 0 type powergroup
        Step chooseleaf firstn 0 type host
        …
}
```

（2）数据分布

Crushtool –test 命令可以将 1024 个 object 模拟分布到集群中。在 OSD 个数为 6 权重均为 1.00，副本策略为 select(3,host)的情况下，PEPEO 算法（分组算法为轮

转分组）优化前后 OSD 上分布的 object 数目如图 3.4 所示。从图 3.4 可以看出，优化后 Ceph 的数据布局均匀性并未发生太大的变化，数据在各 OSD 间的分布仍然比较均匀。这是因为，划分功耗组实际上并未改变 CRUSH 伪随机的特性，只是多了功耗组的选择，功耗组的权重是下层的 OSD 累积而成的，并不改变均匀性，在选择了功耗组后，下层的故障域直至 OSD 的选择仍然是伪随机的，所以仍会产生均匀的数据分布。

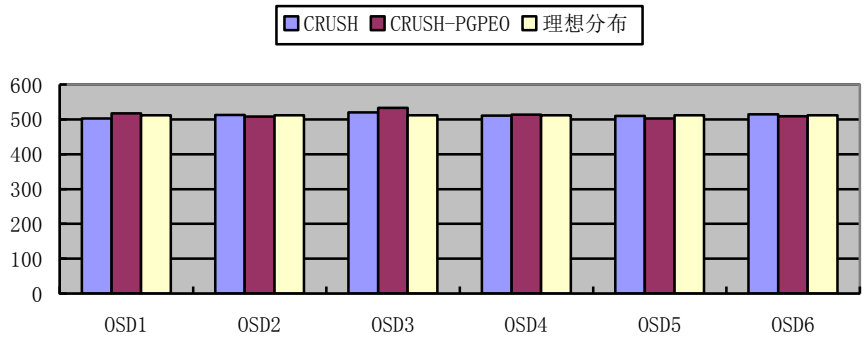

图 3.4　节点同构时分布均匀性

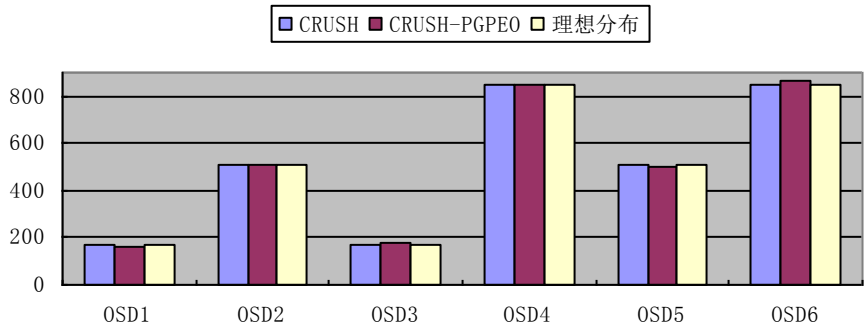

图 3.5　节点异构时分布效果

图 3.5 是 OSD 节点异构时，各 OSD 上的对象分布情况。OSD1 到 OSD6 的权重分别设置为 1.00，3.00，1.00，5.00，3.00，5.00，PGPEO 使用权重均衡分组算法进行故障域的分组。可以看出，各 OSD 上分布的对象数目大致与 OSD 节点的权重对应，从而可以充分利用节点的存储能力。这一结果体现出 PEPEO 的权重均衡分组算法的有效性。

（2）数据定位延迟

Ceph 分布式存储通过计算而非查表的方式来定位数据的设计，是提高扩展性的关键技术之一。客户端、存储节点都可以通过 CRUSH 算法直接计算出数据对象的存放位置，而不需要对象与存放地址对应的列表，因此 CRUSH 算法的执行时间对系统的性能有着重要的影响。我们通过 time 命令测量 crushtool 工具模拟 1024 次数据对象放置过程的时间消耗。优化前后，放置数据对象的总耗时如图 3.6

所示。

可以看出，在使用PEPGO算法对数据布局优化后，数据定位的延迟有了一定程度的增加，但增量均摊到每一次的数据定位上来说，引入的时间开销是非常小的。事实上，由于划分了功耗组，使得Crushmap的层级结构中多了一层，从而会导致数据在放置时会多出一次选择的过程，不过，由于 CRUSH 使用了高效的rjenkins1 作为哈希函数，一次选择的时间消耗是非常有限的，对整个数据的放置过程影响更是几乎可以忽略。

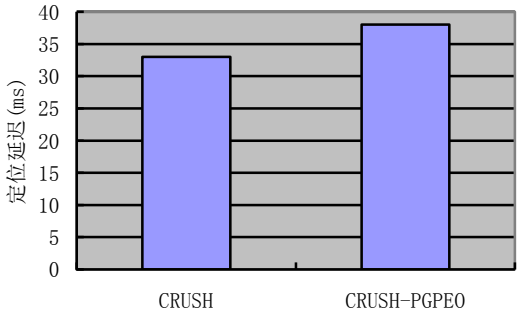

图 3.6　节能优化前后的定位延迟

3.4　本章小结

本章首先从节能的角度分析了 Ceph 分布式存储的数据布局存在的问题。Ceph 基于 CRUSH 的随机副本分布是目前较为常用的副本放置方式，不过这也导致了再节能的时候难以保证数据的可用性，数据无法访问在 Ceph 是不可接受的。接着，针对此问题提出了 PGPEO 数据布局优化算法，对 Ceph 集群进行分组，从而约束了副本分布的位置，使得 Ceph 系统在关闭大部分节点时仍能保证数据的可用性。在具体分组的实现时，提出多种分组算法以满足不同的分组需求。分析与试验评测的结果说明该 PEPEO 算法能够正确的优化 Ceph 的数据布局，提升节能能力，且对系统数据分布均匀性、放置延迟的影响微小。

第四章　多级功耗管理策略

有效的功耗管理机制是存储系统节能的前提。不同与磁盘、磁盘盘阵等存储设备，分布式存储的节点往往是整台服务器，其除了存储设备之外还包含了 CPU、内存、网卡等设备，因此对分布式存储的功耗管理有着特殊的要求。同时，节点的计算和通信能力也为功耗管理提供了更方便、高效的途径。另一方面，分布式存储系统在现实环境中所承受的负载可能是变化的，这就需要功耗管理策略具有动态性，能够在不同的负载情况下合理地调整，在节省能耗的情况下，满足工作负载的要求。

4.1　功耗管理模型选择

功耗管理模型的选取是功耗管理的重要基础，决定了系统在能耗、性能、可靠性能方面的折衷方式，也决定了需要采取的功耗管理策略。常见的用于分布式存储系统的功耗管理模型有覆盖子集（Covering Subset）模型、冷热区划分模型、档位切换（Gear Shifting）模型等，本文对此三种模型进行了分析与比较，作为后续研究的基础。

4.1.1 模型的比较与选择

覆盖子集模型中，一小部分的节点保持活跃以提供服务，被称为覆盖子集 CS，剩余节点为非 CS。CS 节点是始终开启的，并且为保证数据的可用性，CS 中放置了所有数据的一个副本。在系统的负载较低时，为了节能，非 CS 将全部处于关闭或者节能状态，而当系统负载升高，CS 节点已经不能满足要求，非 CS 节点将被重新开启，以满足负载的需要。档位切换模型中，系统的功耗被划分多档位，档位越高则包含越多的存储节点，最高档位包含了所有的节点。从数据的角度看，随着档位的升高，包含副本个数也随之增加。当负载要求系统提供更高的 I/O 并行度及带宽时，系统会切换到高功耗档位，负载要求较低时，系统会切换到低档位以节能。冷热区域划分模型模型没有从数据冗余角度去实现功耗管理，而是基于数据冷热程度。根据数据的访问频率，数据被分为热数据和冷数据，而存储节点也被划分为热区和冷区。热区存放热的数据保持正常工作状态，冷区存放冷的数据，保持节能状态。节点的状态是不变的，而数据将会根据热度的变化而动态地调整所属的节点，以满足负载需要。

从保证数据全集的可用性（数据覆盖），系统的功耗比例性，集群的高可用性以及是否支持数据分类等方面，对上述的三种功耗管理模型进行了比较，结果

如表 4.1 所示。

表 4.1　不同功耗管理模型的比较

功耗管理模型	数据覆盖	功耗比例性	高可用性	数据分类
覆盖子集	好	较好	差	无
冷热区划分	差	差	好	有
档位切换	好	好	较好	无

从表 4.1 可以看出，上述三种功耗管理模型中，在数据全集的可用性方面，覆盖子集和档位切换模型都能保证所有数据块都能立即访问的，因为这两者都是通过减少了数据冗余度的方式来节能的，而冷热区划分模型则是从另一个维度，即数据冷热度进行节点划分的，所以在该模型中，处于冷区的数据是不能立即访问的，无法保证数据全集的可用性。

功耗比例性(Power Proportionality[39]) 是功耗管理模型的重要性质之一，体现了电能的使用效率，理想功耗比例性意味着，对系统要求提供某一比例的服务能力时，系统的功耗同样处于峰值功耗的同一比例水平。覆盖子集的功耗比例性是较差的，因为其只提供了两个能耗等级，只要负载要求超过了 CS 的 I/O 能力，非 CS 就会启动，而不是按比例提供；冷热区划分模型限定了提供服务的节点集，所以是无法体现出功耗比例性；档位切换模型，因为节点被分成了多个档位，每一个档位都对应了相应的服务能力，相对于前二者则体现出较好的功耗比例性。

从集群的高可用性看，覆盖子集模型由于只开启了一个数据副本，所以只要 CS 中的节点出现了故障，则很有可能出现数据丢失或无法访问的情况；冷热区划分模型中由于并未减少数据冗余度，数据所有副本都是启用的，所以集群具有高可用性；档位切换模型可以通过设置最低档位为 2 来保证数据至少有两个副本是活跃的，所以也能较好地保证存储集群的高可用性。

数据分类则是体现出数据块的差异性，有利于通过数据迁移、负载合并以提高系统使用效率。上述三个模型中，只有冷热区划分模型从数据冷热度方面来对数据进行分类的，其他两个模型则考虑的同一数据不同副本之间的关系。然而，在 Ceph 系统中，基于 CRUSH 数据放置算法产生的均匀数据分布以及动态的元数据管理，均衡了各节点的负载，避免了访问热点的出现，数据分类技术很难在 Ceph 系统体现出优势。

通过上述的分析与比较，发现档位切换模型比其余二者有更好较好的功耗比例性且能够保证数据可用性，所以其更适用于 Ceph 系统。此外，在第三章中，本文提出的 PGPEO 优化算法也是通过优化副本分布的来提升节能能力的，与档位切换模型的型联系也更为紧密。因此，本文选择档位切换模型作为 Ceph 系统的功耗管理模型。

4.2 Ceph 的多级功耗管理

基于档位切换模型对 Ceph 的多级功耗管理进行建模，分析不同级别下 Ceph 系统的功耗，并提出相对应的功耗管理策略。

4.2.1 Ceph 的多级功耗模型

在原始的档位切换模型中，存储系统由单独的节点组成，并对节点进行档位（功耗级别）的划分。与此不同的是，Ceph 系统中存储节点集群被描述成是一个多层级的树形结构，除了作为叶子节点的对象存储设备之外，上层还存在着各个层级的故障域，此外，在第三章中，本文为了优化 Ceph 的数据布局，在集群树中还添加了功耗组 PowerGroup 的层级，在 Ceph 中的应用档位切换模型进行功耗管理时应当考虑这些差异。因此，基于档位切换模型，集合 Ceph 的特点进行了功耗级别的划分，划分方法如图 4.1 所示。

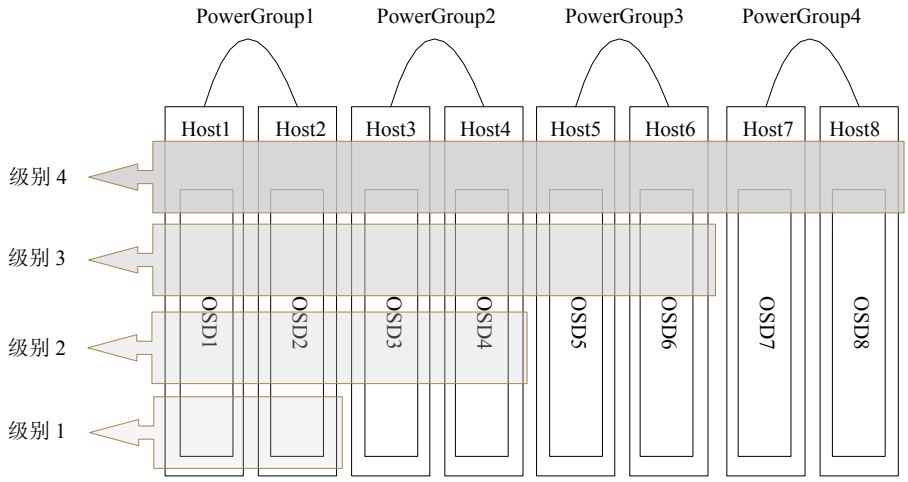

图 4.1 Ceph 的功耗级别划分

经过 PGPEO 算法优化后，数据的 r 个副本分布于不同功耗组，每个功耗组包含的其下层故障域的所有节点。所以，最直接有效的划分方式即是以功耗组为单位进行功耗级别的划分。各功耗级别的划分为：最高的功耗级别包含所有的 r（副本个数）个功耗组；功耗降级别降低时，所含的功耗组也相应减少，一个级别的降低对应 1 个功耗组的减少；最低的功耗级别则是可以设置的，设置为 1 时，可以保证获得最大的节能效果，此时系统共有 r 个功耗级别；设置为 2 时，可以保证高可用性，系统共有 r-1 个功耗级别。从数据全集可用副本个数看，功耗级别最高时，所有副本均可用，可以提供最高的性能；每个功耗级别降低均意味着可用副本数的减少，但系统的功耗也随之下降。当系统处于低功耗级别时，不属于此级别的功耗组都进入节能状态。如图 4.1 中，当 Ceph 处于功耗级别 2 时，功耗组 1 和 2 处于正常功耗状态，功耗组 3、4 则会处于节能状态。

4.2.2 Ceph 的功耗分析

在 Ceph 分布式存储中，大部分的节点为 OSD 节点，且元数据服务与集群监控均可部署在 OSD 节点上，所以本文把 OSD 集群的功耗近似看作 Ceph 系统的功耗。不同于普通磁盘的或是盘阵的是，OSD 节点是运行着操作系统和 Ceph OSD 的服务器，其包含了 CPU、磁盘、内存、网卡等硬件，因此在考虑系统功耗时候，需将这些硬件功耗全部算作节点功耗的组成部分。对一个 Ceph 集群的功耗分析，定义参数如表 4.2 所示。

表 4.2　Ceph 功耗分析参数及说明

参数	说明
N	OSD 节点个数
R	副本个数
G	系统所处的功耗级别
p_{cpu}	OSD 节点 CPU 的功耗
p_{mem}	OSD 节点内存的功耗
p_{iface}	OSD 节点网卡的功耗
p_{disk}	OSD 节点磁盘的功耗
p，p_{ps}	OSD 节点的总功耗、节能状态功耗
P_{pg}，P_{pgps}	功耗组功耗，功耗组节能状态功耗
P_{active}，$P_{inactive}$	活跃功耗组总功耗，节能功耗组总功耗
P_{sys}	整个系统的功耗

在 Ceph 集群中，相关的功耗分析如下所述。

一个 OSD 节点的功耗为：

$$p = p_{cpu} + p_{mem} + p_{iface} + p_{disk} \tag{4.1}$$

一个功耗组的功耗为：

$$P_{pg} = p*n/r \tag{4.2}$$

系统处于功耗级别 g 时，所有活跃的功耗组功耗为：

$$\begin{aligned} P_{active} &= P_{pg} * g \\ &= (n/r) * p * g \\ &= n * p * (\frac{g}{r}) \end{aligned} \tag{4.3}$$

系统处于功耗级别 g 时，所有处于节能状态的功耗组的功耗为：

$$\begin{aligned} P_{inactive} &= (n/r) * p_{ps} * (\mathrm{r}-\mathrm{g}) \\ &= n * p_{ps} * (\frac{r-g}{r}) \end{aligned} \tag{4.4}$$

系统处于功耗级别 g 时，系统的总功耗为:

$$\begin{aligned} P_{sys} &= P_{active} + P_{ps} \\ &= n * p * (\frac{g}{r}) + n * p_{ps} * (\frac{r-g}{r}) \\ &= n * \frac{p * g + p_{ps} * (r-g)}{r} \end{aligned} \tag{4.5}$$

基于功耗分析，对系统的运行一段时间的能耗进行分析，各参数及说明见表格 4.3。

表 4.3　Ceph 能耗分析参数及说明

参数	说明
t_g	系统处于功耗级别 g 的时间
e_{save}	OSD 节点一段时间内节能的能耗
e_{up}，e_{down}	OSD 节能开启能耗、进入节能模式能耗
E_{up}，E_{down}	功耗组开启能耗、进入节能模式能耗
I_{up}，I_{down}	功耗组开启、进入节能模式次数
E_{tran}	系统用于功耗级别切换的总能耗
E	系统总能耗

对于单个 OSD 节点，从正常工作状态进入节能状态，并保持时间一段时间 t，此过程节省的能耗为：

$$e_{save} = (p - p_{ps}) * t \tag{4.6}$$

由于 OSD 节点在从节能状态回到正常工作状态的过程会产生额外的 e_{up} 的能耗，所以要使节点在 t 时间段内能够实现节能，需要产生额外产生的能耗小于节省的能耗，则 t 需满足：

$$t > \frac{e_{up}}{p - p_{ps}} \tag{4.7}$$

系统总的能耗包含其处于各功耗级别时所产生的能耗以及用于功耗级别间相互切换的能耗，一段时间 t 内，系统的总能耗为：

$$\begin{aligned} E &= \sum_{g=1}^{r} P_{sys} t_g + E_{tran} \\ &= \sum_{g=1}^{r} P_{sys} t_g + \sum_{g=1}^{r} I_{up} * e_{up} * r + \sum_{g=1}^{r} I_{down} * e_{down} * r \end{aligned} \tag{4.8}$$

从等式 4.5 可以看出，只要 OSD 节点在节能模式的时候功耗较小，系统的实际功耗与功耗级别想符合，近似成线性关系；不等式 4.7 表明了系统需要有持续的低负载时期的必要性，因为短时间的节能状态减少的能耗不足以抵消用于切换的能耗。类似地，等式 4.8 同样说明在系统功耗管理时需注意到功耗级别切换带来

的额外的能耗。这些结论是本文确定管理策略的依据之一。

4.2.3 多级功耗管理策略

多级功耗管理策略的目的是在上述的 Ceph 的多级功耗管理模型基础上，动态的管理系统的功耗状态，有效地减少系统的能耗。所以，在确定功耗管理策略时面临着如何对负载感知，功耗级别确定、切换及具体的实施方法以及如何进行数据同步等问题。

（1）负载感知

多级功耗管理的需要感知系统的 I/O 负载状态。一方面，不同的功耗级下，系统具有不同的服务能力，如 I/O 带宽、并行度等，功耗管理策略需要知道负载状态以作为确定功耗级别的依据；另一方面，Ceph 基于的多级功耗模型有着较好的功耗比例性，感知并量化 I/O 负载，能够较为精确的提供服务，以做到满足工作负载要求的同时又不过度提供（Over Provision）。

Ceph 系统的 I/O 负载包含了对元数据服务器的服务和对 OSD 的负载，其中元数据的服务器的负载主要是客户端对元数据的操作，而 OSD 的负载则是目标数据的 I/O，我们主要关注 OSD 的负载。数据的 I/O 形式可以分为随机 I/O 和连续 I/O，随机 I/O 的特点为数据量较少，次数较多，如 OLTP 环境中的数据库访问等，随机 I/O 能力一般用 IOPS 度量；连续 I/O 则表现为数据量较大，次数较少，如 FileServer 服务中的文件访问等，连续 I/O 一般可以通过 I/O 吞吐率描述。为了全面地获取系统的负载状态，需要对这两种 I/O 的数据都进行收集。此外，由于连续 I/O 中的写操作需要写多个副本，而连续读操作只需要读一次，所以对于连续读和连续写也要区分。因此，需要收集的 I/O 状态数据是，在可配置的窗口时间 W 内，统计系统收到的随机访问总次数 IO_{rtotal}、连续读操作的总数据量 $IO_{rdtotal}$ 和 IO_{wtotal}，并以此计算出随机读写的 IOPS 和连续读写的吞吐率，分别如等式 4.9、4.10 和 4.11 所示。

$$IO_{ran} = IO_{rtotal} / W \tag{4.9}$$

$$IO_{wseq} = IO_{swtotal} / W \tag{4.10}$$

$$IO_{rdseq} = IO_{rdtotal} / W \tag{4.11}$$

在此基础上，可以得出系统 I/O 状态的量化描述，即 I/O 负载率 R_L：

$$R_L = \max\{\frac{IO_{ran}}{IO_{rpeak}}, \frac{IO_{wseq}}{IO_{wpeak}}, \frac{IO_{rdseq}}{IO_{rdpeak}}\} \tag{4.12}$$

其中，IO_{rpeak}，IO_{rpeak}，IO_{rpeak} 分别为预先测得的系统在负载高峰期的随机访问 IOPS、连续读以及连续写访问的吞吐率。I/O 负载率 R_L 将会被用来确定系统的功耗级别。窗口时间 W 的选择应考虑到，W 过大，可能错过最佳的节能时机，有一

定的反应延迟，W 过小，在负载变化较快时，则可能出现频繁地节点开启/关闭，产生多余的能耗。

（2）功耗级别切换

算法 4.1 Ceph 的功耗级别切换算法

输入：监测周期 period

输出：级别切换结果

```
1：for every overtime(period) do                    /*周期性检查*/
2：    gearCur = getGearLevel()                     /*获取当前功耗级别*/
3：    ioRate = getIOLoadRate()                     /*获取当前 IO 负载率*
4：    pwRate = getPowerRate()                      /*获取当前功耗负载率*
5：    gearNext = gearCur;                          /*初始化下阶段功耗级别*
6：    if ioRate < pwRate then                      /*需要降低级别*/
7：       while (gearNext >= minGear) && (ioRate <= pwRate) do
8：          gearNext = gearNext - 1
9：          pwRate = gearNext/r                   /*计算降低级别后的功耗率*/
10：      end while
11：      gearNext = gearNext + 1;                 /*优先满足负载*/
12：   end if
13：   elseif ioRate > pwRate then                 /*需要提升级别*/
14：      while (gearNext < topGear) && (ioRate >= pwRate) do
15：         gearNext = gearNext + 1
16：         pwRate = gearNext/r
17：      end while
18：   end elseif
19：   if gearNext != gearCur then
20：      switchToGear(gearNext);                  /*级别切换过程*/
21：   end if
22：end for
```

功耗级别切换发生在当前功耗级别所提供的服务能力已经高于或者低于负载的要求时。Ceph 系统中随着 OSD 节点的增减，系统的 IOPS、I/O 吞吐率和 OSD 节点之间表现出几乎线性的关系，所以，可以认为系统的服务能力同样随着活跃的功耗组个数及功耗级别的切换成比例增减。基于此，本文定义 Ceph 系统的功耗率 Rp 为：

$$R_p = g / \mathrm{r} \tag{4.13}$$

此时，Rp 同样体现了系统提供服务能力的比例，可以用来和系统 I/O 负载率 R_L 比较，确定是否需要切换功耗级别。R_L 负载率可以有两种，一种为当前负载率，即按照前一小节中所述的方式统计并计算出；另一种是下阶段负载率，通过当前统计到的数据经过预测得到。使用当前负载率进行对比是被动调整的方式，会有一定的延迟，并和级别切换检查的周期设置相关。使用下阶段负载率作为比较依据，则是主动预测的方式，级别切换的延迟较小，但负载预测是和工作负载（Work load）紧密相关的，难有通用的预测模型保证准确性。所以，目前本文使用的是基于统计数据计算出的系统当前负载。

根据以上分析，Ceph 的功耗级别切换算法伪码如算法 4.1 所示。其中 minGear 为允许的功耗级别的下限，为保证系统的高可用性，可以设置 minGear 为 2。系统功耗级别切换算法的流程是：检测到超时发生后，计算出系统当前的 IO 负载率与功耗率并进行比较。当系统的 I/O 负载率高于功耗率时，活跃的功耗组已经不能满足 I/O 负载的要求，系统切换到更高的级别，提供更高的服务能力；当系统的 I/O 负载率低于功耗率时，系统降低功耗级别，直至功耗率比 I/O 负载率低为止，但考虑到优先保证负载需求的满足，最后会在此基础上提升一个级别。举例来说，I/O 负载率为 40%，系统当前的功耗率为 100%，按照功耗级别的切换原则，应切换到级别 1,功耗率降为 33%，但为以满足负载为目标，系统最终会切换到功耗级别 2，此时功耗率为 67%。

（3）级别切换方式

表 4.4　不同状态下硬件设备的功耗[41]

硬件	工作状态（瓦）	空闲状态（瓦）	睡眠状态（瓦）
CPU(Xeon 5400)	80-150	12-20	3.4
内存	3.5-5	1.8-2.5	0.2
网卡	0.7	0.3	0.3
硬盘	11.16	9.29	0.99
服务器	400	132	13

级别切换方式涉及的问题是通过什么样的操作来完成功耗级别的切换。在很多研相关的工作中，研究人员都是通过关机（Power Off）和开机（Power on）的方式完成节点在节能状态和工作状态间切换的。但是，在本文提出的 Ceph 的功耗管理策略中，通过睡眠的方式使节点进入节能状态(ACPI[40]的 S3)，以唤醒的方式使节能恢复工作。主要原因如下：

一方面，睡眠/唤醒的功耗级别切换方式能够较好的满足系统节能的要求。ACPI（Advanced Configuration and Power Management Interface）是被工业界广泛认可的电源管理标准，经过多年的发展已经被普遍使用。目前，从 CPU、内存等硬件设备，到操作系统，到设备驱动，ACPI 都已经被很好的支持。ACPI 电源管

理规定了 S0 到 S5 的 6 种电源状态，其中状态 S3（Suspend to RAM）即本文使用的睡眠状态。节点在睡眠状态下，各硬件设备能够以很低的功耗运行，表 4.4 的为常见的服务器及硬件在各种工作状态下的功耗[41]。可以看出，节点处于睡眠状态时的功耗较工作状态的功耗减小很多，从约 400 瓦降低到 13 瓦，满足系统节能的需求。

另一方面，虽然睡眠状态的功耗无法与 PowerOff 时几乎为零的功耗相比，但在功耗级别切换的延迟上，睡眠/唤醒模式却要小很多。通常情况下，系统从工作状态进入睡眠状态以及从睡眠被唤醒至工作状态，都是在数秒内完成的，且唤醒之后 OSD 节点能够立即提供服务。相比之下，以 Power Off/Power On 方式切功换耗状态需要的时间则长得多，尤其是在 Power On 以进入工作状态的时间。服务器在开机启动时需要进行硬件检测，这一过程甚至会消耗一分钟以上的时间，再加上操作系统的启动和 Ceph OSD 服务的所消耗的数十秒时间，通过这种方式进行功耗级别切换的延迟是睡眠/唤醒方式的数十倍，这对于系统在短时间内提升服务能力的目标来说是无法接受的。

此外，服务器的开机过程会消耗较多电能，根据等式 4.8，多次开关机对整个系统的节能有着负面的影响，而睡眠或唤醒的过程其能耗会小得多，几乎是可以忽略的。从可操作性方面看，睡眠/唤醒的切换方式也更容易操作，均可通过远程操作完成切换，如通过网络唤醒技术（Wake on Lan[41]）实现功耗级别的提升，通过 SSH 加 Standby 命令使节点进入睡眠状态，降低功耗级别。这有利系统的自动化功耗管理。OSD 节点硬件在功耗级别切换过程中的状态变化如图 4.2 所示。。

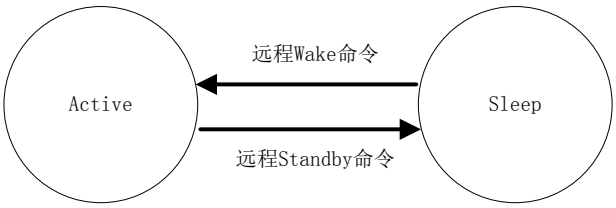

图 4.2　节点硬件状态变迁

（4）数据同步

数据同步是为了保证睡眠的节点在被唤醒后，其上的数据与活跃节点的数据是一致的，从而可靠地提供服务。对于部分数据而言，其在系统节能状态下可能被改写，但是存放数据副本的节点可能处于节能状态，而未被更新。此时，当节能状态的 OSD 节点被唤醒后，需要进行数据的同步。需要注意的是，Ceph 系统是自恢复的，即当数据的活跃的副本个数小于副本策略的设定时，系统认为此数据处于 degraded 状态，并在一段时间后自动迁移和复制，以满足副本策略的要求。所以，我们需要延后这一过程，并让其发生在存放副本的 OSD 被唤醒后。Noout

是 Ceph 系统用于维护的状态标识，这一状态下 OSD 节点可以被关闭，但不会被表示 out 状态，从而不会发生数据迁移，本文正是合理地利用了这一设定。本文首先设置 noout 状态标识，然后进行功耗级别的管理，可以在数据处于 degraded 状态时提供服务，并在数据放置的 OSD 节点被唤醒后，该数据才会被复制与同步。OSD 服务及数据对象的状态变迁分别如图 4.3、图 4.4 所示，其中数据对象的 clean 状态为正常状态，peering 状态为正在同步状态。

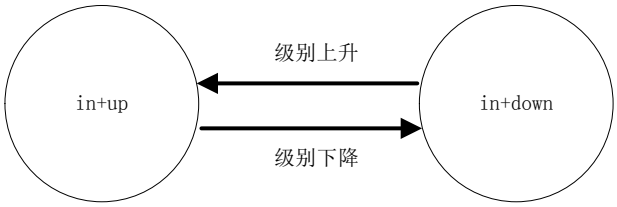

图 4.3　OSD 状态变迁

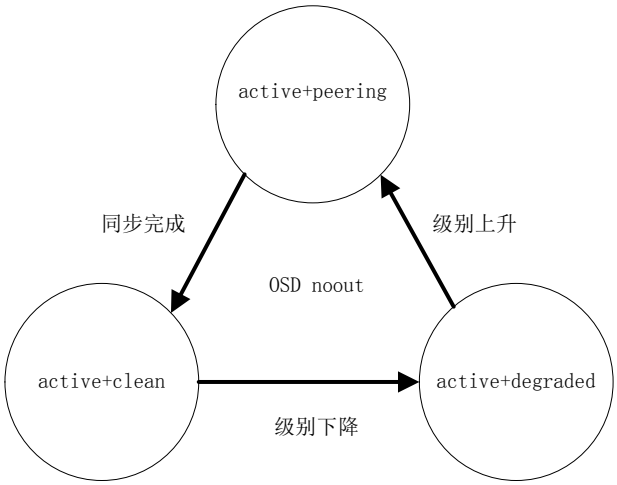

图 4.4　数据的状态变迁

数据同步的过程完全是由 Ceph 自动完成的，在每个 OSD 上都有一个日志来记录数据对象的更新状态，在节能状态的 OSD 被唤醒后，Ceph 会对比数据副本所在的各 OSD 上的日志，以确定最新的已被确认的更新，并在所有副本节点上同步。

4.3　分析及实验评测

4.3.1 功耗比例性

功耗比例性是系统节能的重要性质，体现了系统同时满足负载要求和节能目的的能力，本文提出的 Ceph 基于档位切换的功耗管理模型及多级功耗管理策略，能够根据量化的负载状态提供相应的功耗，具有较好的功耗比例性，如图 4.4 所示。图中 r 为副本个数，所以系统的功耗被分为 r 个级别，随着负载率的增上升，每增加 1/r 的步长，功耗就上升一个级别，反之亦然。另一方面，随着副本个数 r 的增加，功耗级别也相应增多，从而体现出更好的功耗比例性。

理论上，节点级的功耗管理可以提供最优的功耗比例性，但本文认为节点级的粒度并不适合与Ceph系统的功耗管理，一方面这增加了节能状态管理的复杂度，更重要的是，由于数据副本是基于功耗组分布的，当一个功耗组中只开启部分节点时，会降低系统的有效存储空间，造成新的数据对象无法写入。

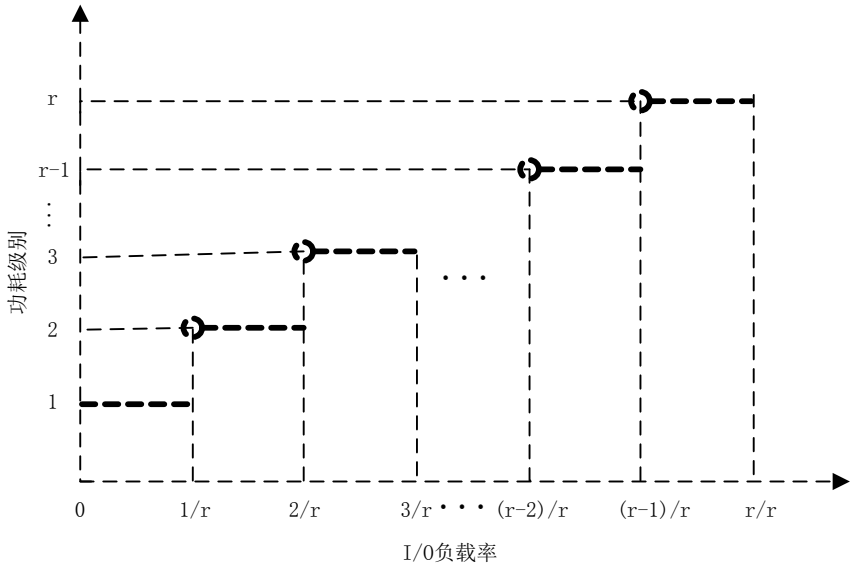

图 4.5　Ceph 多级功耗管理策略的功耗比例性

4.3.2 级别切换延迟

级别切换延迟是功耗管理实施过程中的重要指标，本文在对功耗管理策略中使用的睡眠/唤醒切换方式和传统的开机/关机方式进行了切换延迟的实验比较。切换的到高级别的延迟以节点从节能状态到集群监视器检测到OSD节点进入集群的时间为准；切换到低级别的延迟以执行节能命令到完全进入节能状态为准。级别切换延迟的测试结果如图 4.5 所示。可以发现，使用休眠/唤醒方式进行系统功耗级别的切换延迟较小，尤其是在级别上升时，虽然这些数据可能随着硬件配置的不同而变化，但休眠/唤醒的切换方式仍然具有明显的优势。

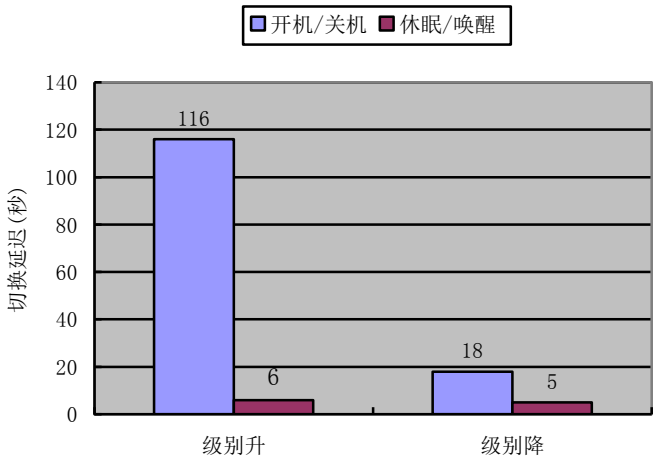

图 4.6　功耗级别切换延迟比较

4.4　本章小结

本章首先分析比较了分布式存储的功耗管理模型，其中档位切换模型关注数据可用性且有较好的功耗比例性，所以更适用于 Ceph 的功耗管理。接着，基于档位切换模型对 Ceph 系统进行了多级功耗管理的建模，并对各功耗级别的功耗及能耗进行分析。在此基础上，提出了 Ceph 分布式存储的功耗管理策略，解决了负载感知、级别切换、数据同步以及切换方式等关键问题。实验结果表明，本文提出的多级功耗管理模型有着较好的功耗比例性，功耗管理策略的级别切换延迟也很小。

第五章　原型系统与评测

在第三、四章的基础上，本章设计并实现了 Ceph 的功耗管理系统。该系统具体实现了 PGPEO 优化算法和多级功耗管理策略，并利合理的利用了 Ceph 的部分管理和配置接口。实验结果表明该功耗管理系统能够动态地管理 Ceph 分布式存储的功耗状态，有效减少 Ceph 的能耗，并能够保证其服务质量。

5.1　原型系统的设计

Ceph 的功耗管理系统采用模块化设计，不同模块承担着功耗管理过程中不同的任务；功耗管理系统以操作系统服务的方式部署于 Ceph 集群节点上，并作为单独的后台程序独立运行。

5.1.1 系统架构

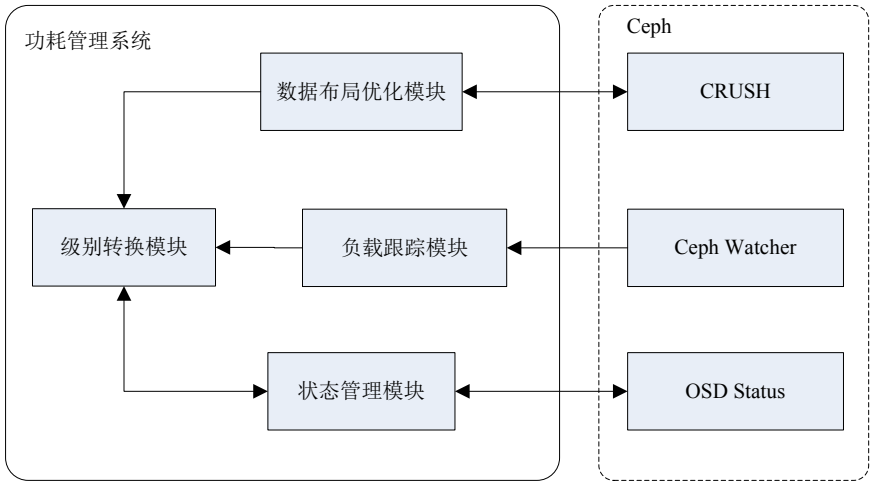

图 5.1　功耗管理系统的架构

如图 5.1，该功耗管理系统由 4 个模块组成，分别是数据布局优化模块（Layout Optimizer）、负载跟踪模块（Load Tracer）、功耗级别切换模块（Gear Shifter）和状态管理模块（Staus Manager）。各模块的主要功能及作用如下：

(1) 数据布局优化模块

该模块实现对 Ceph 的数据布局的优化算法。通过对 Ceph 集群层级描述文件 Crushmap 分析，找出副本分布的故障域和放置规则，进而进行功耗组的划分。由于 CRUSH 算法是全局知晓的，所以划分功耗组后，该模块会将划分结果反馈到整个 Ceph 集群中。

(2) 负载跟踪模块

该模块用于跟踪与统计 Ceph 集群中的 I/O 负载数据。以指定的频率采集分析 Ceph Watcher 中的 I/O 访问记录，如 I/O 次数、读和写访问的数据量等，根据窗口时间的设置分别计算出相应的指标，以便确定负载状态。

(3) 功耗级别切换模块

该模块实现了多级功耗管理策略的级别切换算法的级别确定部分。根据从负载跟踪模块得到到的 I/O 负载的指标，根据一定的策略计算系统的 I/O 负载率，并与系统当前的功耗级别做比较，确定是否需要切换功耗级别。计算负载率的策略是可以配置的，支持被动的切换和主动预测的切换。

(4) 状态管理模块

该模块负责切换算法中级别切换的执行和节点功耗状态的管理。该模块一方面利用 Ceph 的状态管理工具设置 OSD 节点在集群软件中的状态，一方面接收来自于级别切换模块的切换指令并通过远程休眠/局域网唤醒等技术控制 OSD 所在的服务器硬件的电源管理状态。

5.1.2 系统部署与运行

从部署方式看，功耗管理系统需要与 Ceph 集群的 OSD 集群、监视器以及元数据服务器集群通信，而其中 OSD 集群是有可能进入节能状态的，所以可以将该系统部署于监视器节点上，方便更快地获取到集群状态信息。功耗系统在 Ceph 集群中的部署位置如图 5.2 所示。

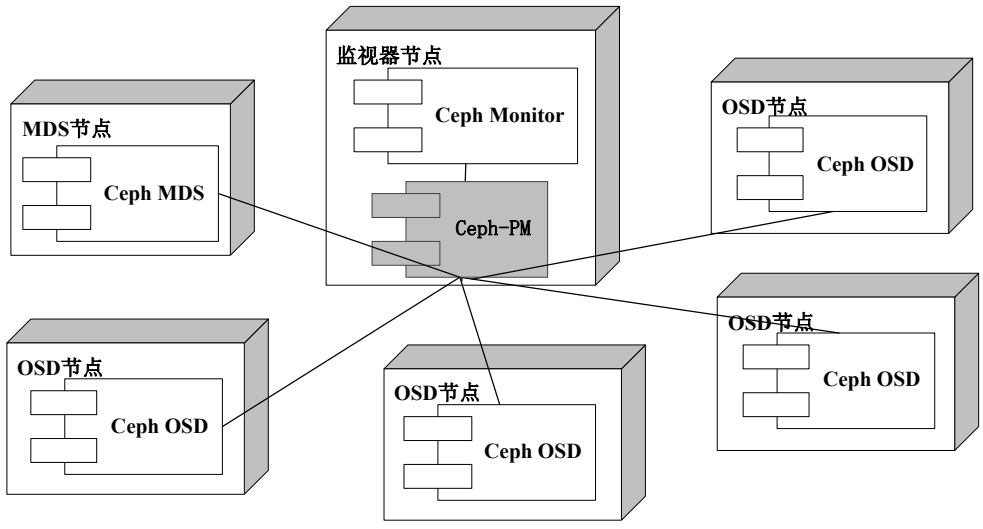

图 5.2　功耗管理系统的部署位置

从运行方式上看，功耗管理系统在集群监视器节点上作为一个操作系统的服务在后台运行，无需交互式干预。功耗管理系统需要依赖于 Ceph 服务运行，所以为该功耗管理服务编写的 LSB 启动脚本配置段（LSB Header）为：

```
### BEGIN INIT INFO
#Provides:            ceph-pm
#Default-Start:       2,3,4,5
#Defaukt-Stop         0,1,6
#Required-Start       $ceph $network
#Required-Stop        $ceph $network
#Short-Description    Start Ceph power manage system daemon at boot_time
### END INIT INFO
```

5.1.3 系统工作流程

功耗管理系统的工作流程如图 5.3 所示。

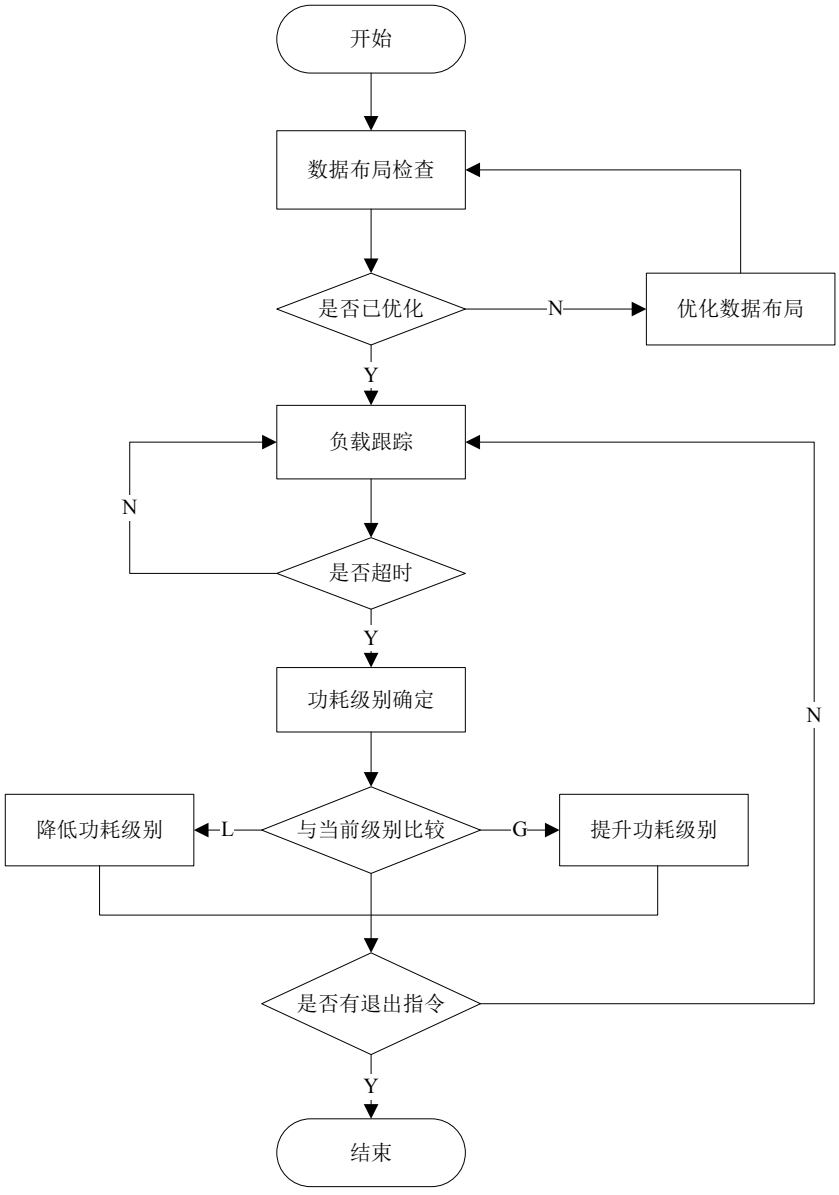

图 5.3　功耗管理系统工作流程图

5.2 功耗管理系统的实现

本文在麒麟服务器操作系统 Kylin3.2 上，使用 Python 语言编程实现 Ceph 的功耗管理系统，该统中各模块的实现如下所述。

5.2.1 数据布局优化模块

数据布局优化模块用于对 Ceph 的数据布局进行以技能为目的的优化，实现了基于功耗组划分的优化算法，主要的函数有 layout_check()，find_tg_fd()，fd_group()，enable_group()等。该模块的工作流程 5.4 如图所示。

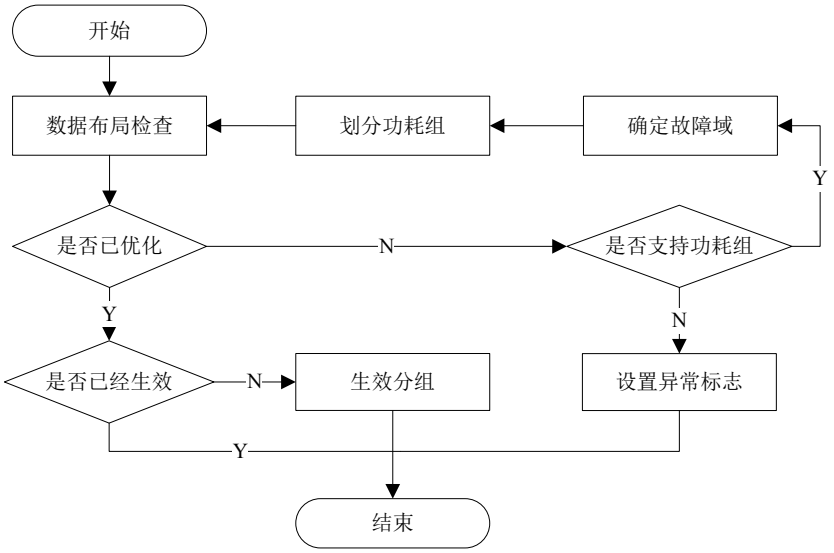

图 5.4 数据布局优化模块工作流程

主要过程及其实现函数描述为：

(1) 数据布局检查，此过程主要由 layout_check()函数完成。该函数通过 python 的模式匹配 API 检查经过反编译的集群 crushmap.txt 中是否已经添加功耗组的层级且有基于功耗组的副本放置策略。

(2) 确定故障域，此过程由 find_tg_fd()函数完成。该函数会通过模式匹配 API 匹配所有包含了副本放置策略的语句，列出其使用到的故障域，并选出 type 值最大（层级最高）的故障域作为目标故障域。

(3) 划分功耗组，此过程由 fd_group 函数完成。该函数根据指定的分组算法将目标故障域划分进 r 个不同的功耗组，每个功耗组的权重为包含的故障域的权重的总合，其关键代码片段为:

```
def fd_group(self,tgt_fds,r,group,group_flag):                /*故障域分组*/
  …
  if group_flag == rotate :                                   /*轮转分组*/
```

```
for in pg in group:
    for in fd in tgt_fds:
        if (fd.id % r) == (pg.id % r):
            pg.items.add(fd)
            pg.weight += fd.weight
```

(4) 分组生效，此过程由 enable_group()函数完成。由于 CRUSH 算法相关的改动在 Ceph 集群中是要全局知晓的，所以该函数主要功能就是先将修优化的 crushmap.txt 编译，并调用 Ceph 管理接口，是新的数据布局策略在集群中生效。此函数的实现基本都是通过直接调用系统命令和 crushtool 命令和 Ceph 管理命令完成的，所以该函数中主要调用 Python 系统包的 system 方法，主要代码片段如下：

```
def enable_group(self,mapfile):
    …
    os.system(„crushtool –c mapfile –o newmap’)
    os.system(„ceph osd crushmap –i newmap’)
    …
```

5.2.2 负载跟踪模块

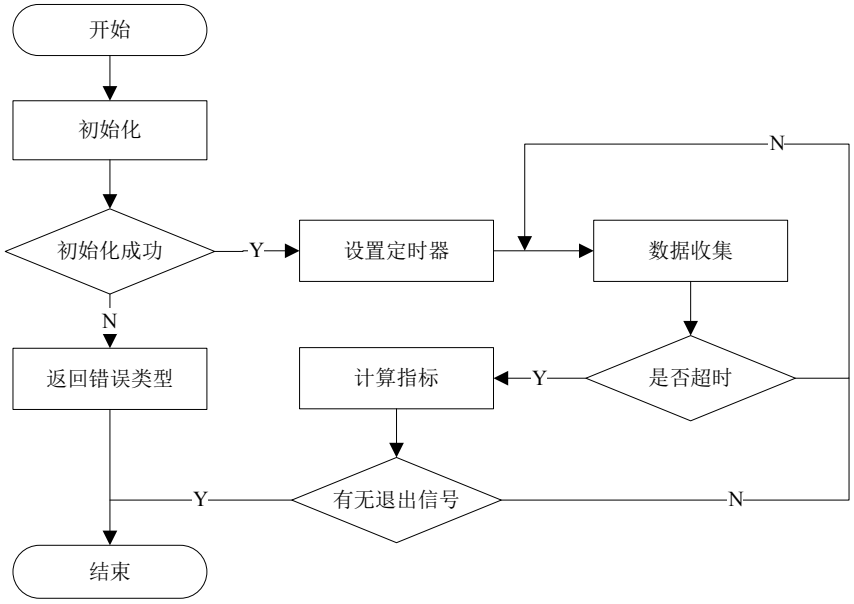

图 5.5　负载跟踪模块的工作流程

负载跟踪模块负跟踪 Ceph 集群中的 I/O 负载情况，周期性的通过 Ceph Watcher 和 Ceph Log 的获取系统的 I/O 访问记录，并根据设置的窗口时间的计算出相应的负载指标。负载跟踪模块主要函数函数有 init_load_tracer()，setup_timer()，data_stats()，compute_metrics()等。模块的工作流程如图 5.5 所示。

主要过程及其实现函数描述为：

(1) 初始化，此过程由 init_load_tracer()函数完成。该函数首先会解析用户指定的窗口时间参数，以便在后续的定时器中用到，接着以子进程的方式打开 Ceph watcher，并将其输出重定向以便后续处理。该函数的主要用到了 OptionParser 和 Subprocess 类，关键代码片段为：

```
def init_load_tracer(self):
    …
    (options,args) = parse.parse_args()
    self.wintime = options.windowtime
    …
    self.popen = subpreocess.Popen([,ceph’,’-w’],stdout = subpprocess.PIPE)
    …
```

(2) 设置定时器，此过程由 setup_timer()函数完成。该函数用到了 Threading 包中的 Timer 类，通过 timer(wintime,copute_metrics)设置定时器，其中 wintime 为上述的窗口时间，表明系统每隔 wintime 的时间就会进行一次 I/O 负载指标的计算。

(3) 数据收集，此过程由 data_stats()函数完成。该函数利用输出重定向了的 Ceph watcher 的输出，经过字符串处理后得到其中 I/O 访问的信息输出，并对采集到的 IO 次数和数据量进行累加。关键代码片段为

```
def data_stats(self):
  …
  while True:
    next_line = self.popen.stiout.readline()
    …
    self.ios += cli_ios
    self.rddata += cli_rddata
    self.wrdata += wrdata
  …
```

(4) 计算指标，该过程由 compute_metrics()函数完成。该函数在设置定时器时，被指定为超时的回调函数，所以在定时器超时后会执行，计算 I/O 负载指标。I/O 负载指标是根据 data_stats()收集的数据，除以窗口时间 wintime，得到相应的 IOPS 和 IO 吞吐率指标，并通过队列发送给功耗级别切换模块。

5.2.3 功耗级别切换模块

功耗级别切换模块主要实现了多级功耗管理策略的级别切换算法中的级别确定部分。负载跟踪模块计算出窗口时间内的 I/O 负载指标后，这些数据将会被用来计算或者预测出 I/O 负载率。I/O 负载率将被用作与系统当前的功耗级别做比较，

以确定下阶段的功耗级别。如需要切换功耗级别，将由状态管理模块来实施切换过程。功耗级别切换模块主要的函数有 init_gear_shifter()，monitor_queue()，compute_load()，compute_gear()等。模块的工作流程如图 5.6 所示。

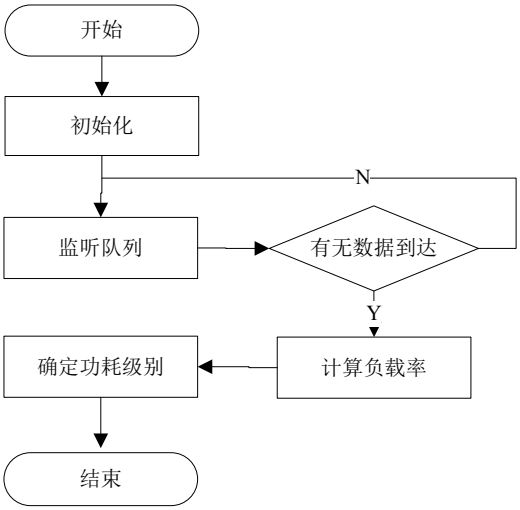

图 5.6　功耗级别切换模块工作流程

主要过程及其实现函数描述：

(1) 初始化，此过程由 init_gear_shifter()函数完成。由于负载跟踪模块是以流的方式收集负载数据的，所以，在功耗级别切换模块中需要与其实时通信，以获取最新的数据。我们使用 Queue 队列用于模块间通信，并在初始化函数中实现。此外，用于计算负载率的可配置的峰值指标也在这里导入。关键代码片段为：

```
queue = Queue.Queue()
def init_gear_shifter(self, pios,prdth,pwrth):
    …
    threading.Thread._init_(self)
    self.in_queue = queue
    self.pios = pios
    self.prdth = prdth
    self.pwrth = pwrth
    …
```

(2) 监听队列，此过程由 monitor_queue()函数完成。该函数监听已建立的队列，以 2s 的频率判断队列中是否有数据，如果有数据，则读出数据，并进入后续的计算负载率过程。

(3) 计算负载率，此过程由 compute_load()函数完成，该函数根据要求计算各指标与最高指标的比例，函数中预留了导入策略的接口，以方便添加基于负载预测的计算负载率的策略。

(4) 确定功耗级别，此过程由 compute_gear()函数完成，该函数首先会确定系统所

处的功耗级别，然后按照功耗级别切换算法描述确定是否需要切换功耗级别。关键代码片段为：

```
def compute_gear(self):
    new_gear = self.cur_gear
    new_pwrate = self.cur_pwrate
    if self.loadrate < self.cur_pwrate :                    /*需要降低级别*/
        while new_gear > self.mingear and self.loadrate <= new_pwrate :
            new_gear –
            new_pwrate = new_gear/float(self.totoal_gear)
        new_gear = new_gear+1
    if self.loadrate > self.cur_pwrate :                    /*需要降低级别*/
        …
    if not new_gear == slef.cur_gear :
        need_gear_shift = True
        return new_gear
    …
```

5.2.4 状态管理模块

状态管理模块负载能耗级切换的实施和功耗状态的管理。包含的主要函数有：init_status_manaer(),switch_gear(),update_status()。工作流程如图 5.7 所示

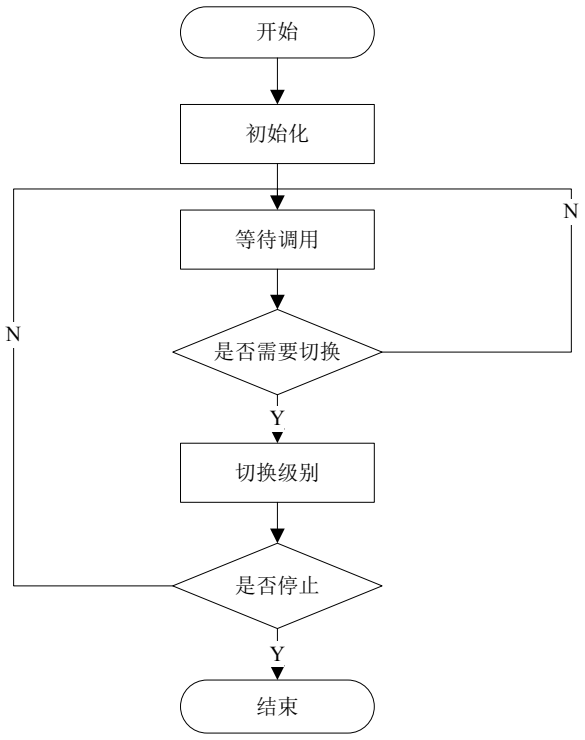

图 5.7　状态管理模块工作流程

(1) 初始化，此过程由 init_status_manaer()函数完成。该函数在模块启动时首先初始化相应的数据结构，包含了功耗级别和的功耗组的对应关系，和所有功耗组的节能状态。接着通过 Ceph 的管理命令“ceph osd set noout”将集群设置为 noout 状态，从而保证不会由于关闭节点而产生的数据迁移。该函数执行完后，执行权将被交给级别切换模块，在确定功耗级别调用状态管理模块的函数。

init_status_manager()函数的关键代码片段为：

```
def init_status_manager(self):
  …
  for osd in osds:
    mac = self.get_macaddr_by_hostname(osd)
    self.maclist.append(mac)
    self.set_osd_status_active(osd,True)
  …
  for gear_group in gear_groups:
    for power_group in power_groups:
      if power_gourp.id <= gear_group.id:
        gear_group.append(power_group)
  …
  os.system(“ceph osd set noout”)
```

(2) 级别切换，此过程由 switch_gear()函数完成。该函数首先判断级别切换标志是否被设置，如果被设置则执行切换动作。切换动作是遍历需要改变状态的功耗组，给其中的每个 OSD 都发送状态变更的命令。关键代码片段为

```
(3) def switch_gear(self,new_gear):
    …
    if need_gear_switch:
      …
      if need_up:
        for osd in osd_need_up:
          os.system(“wol %s”)%osd.mac
          set_osd_status(osd,True)
      if need_down:
        for osd in osd_need_down:
          os.system(“ssh %s echo standby > /sys/power/state”)%osd.hostname
          set_osd_status(osd,False)
  …
```

5.3 实验评测

为了评估本文实现的Ceph的多级功耗管理系统，本文做了以下实验，分别从节能效果和对系统性能的影响等方面进行对该系统进行了实验评估。

5.3.1 实验环境及部署

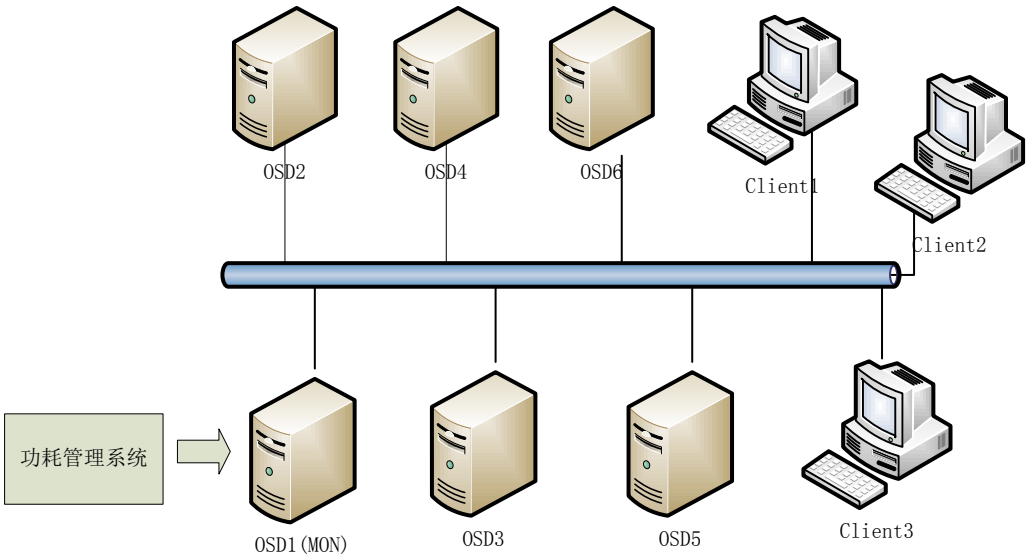

图 5.8 实验环境部署图

如图 5.8 所示，我们在一个有 6 个相同配置服务器节点的集群环境内做测试，每个服务器都作为 Ceph 存储系统的一个 OSD 节点，上面运行着 OSD 服务。同时，在其中的 OSD1 节点上还部署了 Ceph 的集群监控器 MON 服务和本文实现的功耗管理系统。Ceph 存储系统的性能未作任何优化，副本策略设置为 select(3,host)。客户端的配置为 3 台 PC 机，上面安装了 Ceph 客户端和 fio[42]测试工具。每个客户端上均创建了一个 Ceph RBD 块设备并格式化为 ext4 文件系统，挂载在本地的/mnt 路径下。所有节点和客户端上均安装的麒麟操作系统。服务器及客户端的硬件的配置如表 5.1 所示。

表 5.1 测试环境硬件配置表

硬件	OSD 节点配置	客户端配置
CPU	Xeon E5540 2.53GHz x 2	Core E5800 3.2GHz
内存	DDR3 16G	DDR3 4G
硬盘	ST Barracuda 1TB	ST Barracuda 320GB
网卡	千兆网卡	千兆网卡

测试所涉及的软件配置分别如表 5.2 所示。

表 5.2　测试环境软件配置表

软件名称	配置信息
操作系统	Kylin® Linux Server 3.2 x86_64
客户端内核	Kernel-3.10.x86_64
Ceph	Ceph-0.8.0-1.x86_64
测试工具	Fio-2.1.7

5.3.2 节能效果测试

在测试节能效果时，我们通过调整活跃的客户端个数来体现负载的高低变化，并使每个负载状态持续一定的时间。在很多真实的应用场景中，负载也是持续的高、低负载时期的，并体现出一定的规律性，如邮件服务、Web 服务等通常表现出白天负载高、夜间负载低等特征，我们的实验设计正是基于这些观察。节能效果通过统计 Cpeh 系统处于不同功耗级别的次数来评估。

测试过程中，在 3 个客户端上分别用 fio 测试工具生成负载，并每隔 30 分钟，随机暂停其中的 1 个或 2 个以进入低负载状态，持续 30 分钟后恢复继续运行，共运行 8 个小时。功耗级别切换的窗口时间设置为 10 分钟。实验分为 2 组，一组为 4K 的随机访问负载，另一组为 1M 的连续访问负载，结果如图 5.9 和图 5.10 所示。

从图 5.9 可以看出，系统处于低功耗级别的次数比预计的 24 次要少，这是因为一方面，在功耗级别提升时，系统有部分数据将会同步，这将会增加存储集群间的数据流量；另一方面，功耗级别和负载的级别过于接近，导致一些小的负载波动就可能导致系统功耗级别的上升。但从总体的节能效果来看，结果还是值得认可的，节能的比例为 1-(10/3+8/2+30)/48，约为 22.3%。如果系统规模增大，节省的能源开支将会非常可观。

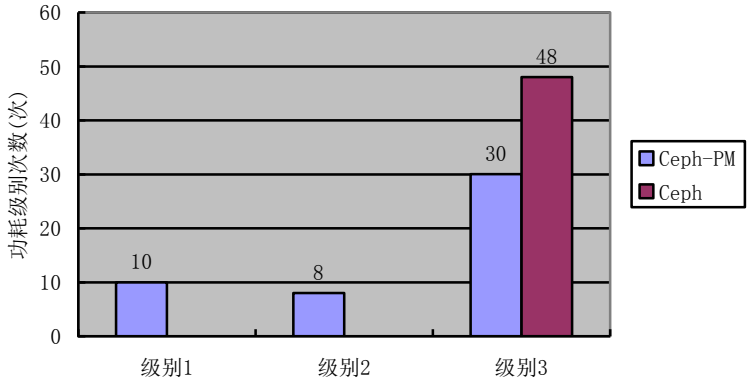

图 5.9　连续 I/O 负载下功耗级别次数

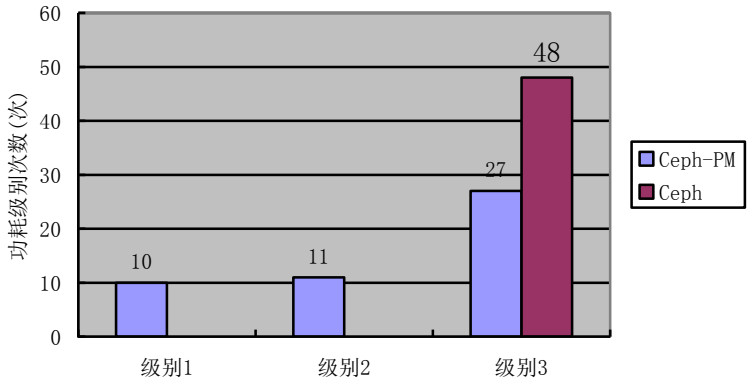

图 5.10　随机 I/O 负载下功耗级别次数

图 5.10 中的结果与连续访问负载的结果类似，但是情况略好，可能是因为随机 I/O 积累的数据不多，在功耗级别上升时用于同步的 I/O 较少。随机负载下，可以获得的节能比例约为 25.4%。需要注意的是，在真实环境中，I/O 负载的变化可能更为频繁、剧烈，所以需要更精确、复杂的负载建模，这也是本文未来的工作内容之一。

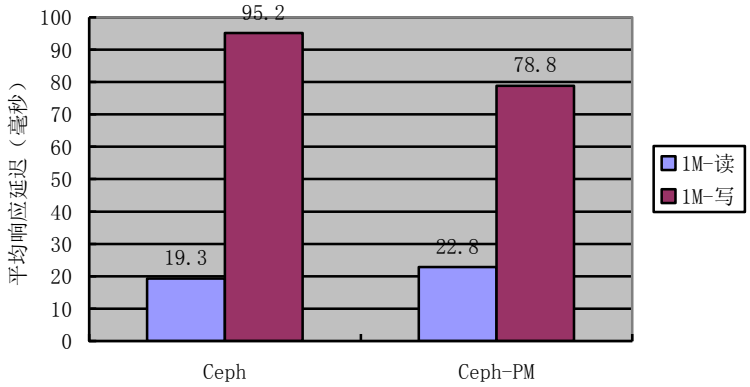

图 5.11　不同功耗级别下的连续 I/O 延迟

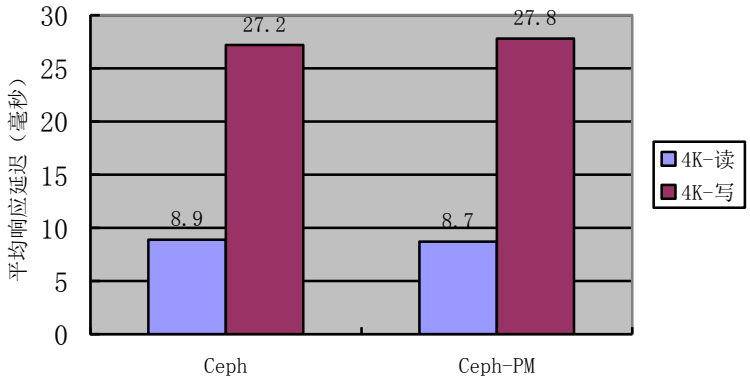

图 5.12　不同功耗级别下的随机 I/O 延迟

5.3.3 性能影响测试

客户端的读写延迟是 Ceph 系统重要性能指标。因此，我们通过在客户端用 fio 进行测试不同功耗状态下 Ceph 系统的读写延迟，结果如图 5.11、图 5.12 所示。图中 Ceph 栏为未启用功耗管理系统，即系统处于最高功耗级别时的读写延迟，Ceph-PM 是系统的功耗级别为 2 时的读写延迟。从连续 I/O 来看，低功耗下的读延迟更大，因为副本个数少了，读的并发度有所下降；但值得注意的是，低功耗状态时 Ceph 系统连续写的延迟反而更小，因为此时需要写的副本个数较高功耗级别时更少，其中用于副本复制(replicaiton)的时间更短。从随机 I/O 的结果看，不同功耗级别下，读写的延迟相当，在功耗级别降低后，结果略有变化，但是整体上影响并不大。

降低功耗级别意味 OSD 总节点的减少，会影响到系统 I/O 带宽和吞吐率，但由于降低功耗级别的前提是客户端提交的访问要求已经减小，所以这些影响，客户端并不感知。而从访问延迟上看，结果也仍能满足客户端需求。

5.4 本章小结

本章首先介绍了 Ceph 系统的多级功耗管理原型系统的设计与部署，模块化的设计使得系统架构清晰，各模块功能专一。接着描述了功耗管理系统的实现，在具体编码时，本文利用了 Python 的多个实用的类库，并合理的利用 Ceph 提供的命令行工具，减小工作量的同时，提高了系统的运行效率。最后阐述了对该系统的测试工作，实验结果表明该系统能够有效的减少 Ceph 的能耗，并能保证服务质量。

第六章 结束语

分布式存储被广泛的用于云计算与大数据处理环境中，并发挥了至关重要的作用。不过，分布式存储的能耗也随着规模的不断扩大日益剧增，对数据中心建立及维护成本的影响非常明显。减少分布式存储的能耗，提升其能源使用率，是实现绿色计算的重要措施之一。

软件定义存储 SDS 是存储系统的发展趋势，Ceph 作为 SDS 解决方案中的领跑着，得到了越来越广泛的关注。Ceph 分布式存储具有高可用、高性能、高扩展性的特点，且开放源代码，有着广阔的应用前景。然而，节能并不是 Ceph 初始设计时考虑的重点，Ceph 系统在节能能力及功耗管理上存在很多不足。

因此，本文基于 Ceph 存储进行了分布式存储节能技术的研究，分析了 Ceph 存储在节能方面的存在的问题，并提出了相应的优化策略。本文的主要完成了以下工作：

第一，从数据布局方面分析了并解决了 Ceph 系统在节能方面的不足。Ceph 基于 CRUSH 算法的数据布局策略，产生伪随机的数据及副本分布，限制了系统的节能比例。因此，本文提出面向节能的 PGPEO 数据布局优化算法。可以在保证数据全集可用的情况下，增加了系统中可以关闭的节点的数目，提高了系统能够节能的比例，随着系统节点数目的增加，节省的能耗将非常的可观。

第二，从功耗管理机制方面提出了有效的 Ceph 系统多级功耗管理策略。Ceph 存储系统设计的初衷是高性能和高扩展性，然而实际应用中，系统经常会处于低负载状态，此时，可以适当的关闭部分节点以节省能耗。因此，本文基于档位切换模型，提出 Ceph 的多级功耗管理策略，使得系统在不同的负载状态下运行于不同的功耗级别，在满足负载需求的情况下，有效地减少系统的能耗。

第三，从实际运行角度出发，设计并实现可了 Ceph 系统的多级功耗管理原型系统。该系统利用了 Ceph 已有的管理和监控接口，结合服务器硬件的电源管理机制以及网络唤醒等技术，实现了对 Ceph 系统的动态功耗管理。实验与测试的结果表明，该系统能够有效的减少 Ceph 存储系统的能耗，并保证了 Ceph 系统的服务质量。

然而，本文的研究同时也存在以下的不足之处：

一方面，本文采用的是静态的数据布局优化策略。可以在系统运行过程中，根据数据表现出的特性，结合 I/O 调度、负载整合等技术，动态地调整数据位置，以获得更好的节能效果。不过，同时也要有效解决因此带来的与 CRUSH 算法的兼容性问题。

另一方面，本文采用被动的功耗级别切换方式，因此有一定的反应延迟，虽

然可以通过设置较小的窗口时间来减小反应时间，但这也会影响节能的效率有一定的影响。而在一些工作负载中，I/O 状态的变化有很强的规律性，所以可以针对这些负载设计基于预测的切换策略，使得功耗管理更及时、准确。

此外，在进行试验评估时，用于测试的负载设计也较为简单，不一定能反应出实际应用的特点，应在条件允许的情况下尝试在真实环境中进行测试。

总之，本文的工作详细分析并解决了 Ceph 系统节能的问题，有效地提高了 Ceph 的能耗使用效率，但本文存在的一些不足也是需要我们认真思考的，将是我们未来的主要工作内容之一。

致　谢

光阴荏苒，两年半的研究生生活即将结束。在此，谨向在我攻读硕士学位过程中，曾经指导过我的老师，帮助过我的朋友，以及关怀、激励我的亲人表示崇高的敬意和深深的感谢。

首先，衷心地感谢我的导师吴庆波研究员。吴老师虽工作繁忙，但从不忘对我的关心和指导。在学习中感到困惑时，吴老师的指点常让我如醍醐灌顶，豁然开朗；在工作中遇到困难时，吴老师的和激励常让我倍感鼓舞，斗志昂扬。吴老师渊博的学识、严谨的作风、敏捷的思维，深深地影响了我，吴老师的谆谆教诲也将令我受益终生。

其次，衷心地感谢杨沙洲老师。杨老师经常抽空指导我的课题研究，和我深入讨论技术问题，并在课题选题及论文撰写方面给了我很大的帮助。虽然部门的任务繁多，但杨老师从不轻易给我安排课题之外的工作，给了我足够的空间与时间用于课题研究。此外，杨老师见识广阔，工作积极，严于律己，宽以待人，也是我学习的榜样。

感谢汪黎老师，谢谢汪老师在我课题研究中对我的指导。汪老师对 Ceph 理解深厚、见解独特，在我对课题感到迷茫时，犹如一盏明灯，指引了我前进的方向。在汪老师的带领下，我积极参与了开源社区的讨论，感受到了开源的魅力。

感谢谭郁松老师，感谢王春光、王文竹、伍复慧、陈孢孜师兄和王静师姐。谢谢大家对我学习上的指导和督促，以及生活上的无私帮助。谭老师和师兄、师姐们营造的活跃而不失严谨的学术氛围，给了我优越的学习和研讨环境，对我顺利完成学业起到了非常关键的作用。

感谢同学魏元豪，同事文云川、肖先霞、李俊良等，和他们积极而深入的讨论常让我深受启发，受益匪浅。

最后，感谢我的家人，感谢你们的陪伴与支持，你们是我坚实的后盾，是我无限动力的源泉！

参考文献

[1] Gantz J, Reinsel D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east[J]. IDC iView: IDC Analyze the Future, 2012.

[2] Koomey J. Growth in data center electricity use 2005 to 2010[EB/OL]. http://www.analyticspress.com/datacenters.html, 2011/2014-10-05.

[3] Times N Y. Power, pollution and the Internet[EB/OL].http://www. nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image. Html, 2012/2014-10-05.

[4] Kim J, Rotem D. Energy proportionality for disk storage using replication[C]//Proceedings of the 14th International Conference on Extending Database Technology. ACM, 2011: 81-92.

[5] Barroso L A, Hölzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines[J]. Synthesis lectures on computer architecture, 2009, 4(1): 1-108.

[6] Poess M, Nambiar R O. Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results[J]. Proceedings of the VLDB Endowment, 2008, 1(2): 1229-1240.

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

[8] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[C]//Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006: 205-218.

[9] DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store[C]//ACM SIGOPS Operating Systems Review. ACM, 2007, 41(6): 205-220.

[10] George L. HBase: the definitive guide[M]. " O'Reilly Media, Inc.", 2011.

[11] Konwinski A, Zaharia M, Katz R, et al. X-tracing Hadoop[C]//Hadoop Summit, 2008.

[12] 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.

[13] Software-defined_storage[EB/OL]. http://en.wikipedia.org/wiki/Software-defined_storage, 2014-09-25/2014-10-05.

[14] Weil S A, Pollack K T, Brandt S A, et al. Dynamic metadata management for petabyte-scale file systems[C]//Proceedings of the 2004 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 2004: 4-15.

[15] Weil S A, Brandt S A, Miller E L, et al. CRUSH: Controlled, scalable,

decentralized placement of replicated data[C]//Proceedings of the 2006 ACM/IEEE conference on Supercomputing. ACM, 2006: 122-133.

[16] Karger D, Lehman E, Leighton T, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web[C]//Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM, 1997: 654-663.

[17] Filesystem in Userspace[EB/OL]. http://en.wikipedia.org/wiki/Filesystem_in_Userspace, 2014-10-4/2014-10-05.

[18] Linus Torvalds. Linux kernel 2.6.34[EB/OL].https://lkml.org/lkml/2010/5/16/89, 2010-05-16/2014-10-05.

[19] Amazon S3[EB/OL].http://aws.amazon.com/cn/s3/, 2014/2014-10-05.

[20] Openstack Swift[EB/OL].https://wiki.openstack.org/wiki/Swift, 2012/2014-10-05.

[21] 王意洁, 孙伟东, 周松, 等. 云计算环境下的分布存储关键技术[J]. 软件学报, 2012, 23(4): 962-986.

[22] Gurumurthi S, Sivasubramaniam A, Kandemir M, et al. DRPM: dynamic speed control for power management in server class disks[C]//Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on. IEEE, 2003: 169-179.

[23] Bisson T, Wu J, Brandt S A. A distributed spin-down algorithm for an object-based storage device with write redirection[C]//Proceedings of the 7th Workshop on Distributed Data and Structures (WDAS'06). 2006.

[24] Lim K, Ranganathan P, Chang J, et al. Understanding and designing new server architectures for emerging warehousecomputing environments[C]//Proceedings of the 35th International Symposium on Computer Architecture (ICSA'08), Beijing, China, Jun 21-25, 2008. Piscataway, NJ, USA: IEEE, 2008: 315-326.

[25] Szalay A S, Bell G, Huang H H, et al. Low-power Amdahl-balanced blades for data intensive computing[J]. ACM SIGOPS Operating Systems Review, 2009, 44(1): 71-75.

[26] Storer M, Greenan K, Miller E, et al. Pergamum: replacing tape with energy efficient, reliable, disk-based archival storage[C]//Proceedings of the 6th USENIX Conference on File and Storage Technologies (2008), San Jose, USA, Feb 26-29, 2008. Berkeley, CA, USA: USENIX Association,2008: 1-16.

[27] Narayanan D, Donnelly A, Rowstron A. Write off-loading: Practical power management for enterprise storage[J]. ACM Transactions on Storage (TOS), 2008, 4(3): 253-267

[28] Leverich J, Kozyrakis C. On the energy (in) efficiency of hadoop clusters[J]. ACM SIGOPS Operating Systems Review, 2010, 44(1): 61-65.

[29] Yazd S A, Venkatesan S, Mittal N. Energy Efficient Hadoop Using Mirrored Data

Block Replication Policy[C]//Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems. IEEE Computer Society, 2012: 457-462.

[30] 廖彬, 于炯, 孙华, 等. 基于存储结构重配置的分布式存储系统节能算法[J]. 计算机研究与发展, 2013, 50(1): 3-18.

[31] Zhu Qingbo, David F M, Devaraj C F, et al. Reducing energy consumption of disk storage using power-aware cache management[C]//Proceedings of the 10th International Conference on High -Performance Computer Architecture (HPCA'04),Madrid, Spain, Feb 14-18, 2004. Piscataway, NJ, USA: IEEE,2004: 118-129.

[32] Lang W, Patel J M. Energy management for mapreduce clusters[J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 129-139.

[33] Kaushik R T, Bhandarkar M. GreenHDFS: towards an energyconserving, storage-efficient, hybrid Hadoop compute cluster[C]// Proceedings of the 2010 International Conference on PowerAware Computing and Systems (HotPower'10), Shanghai,China, Dec 8-10, 2010. Berkeley, CA, USA: USENIX Association,2010: 1-9.

[34] Weddle C, Oldham M, Qian J, et al. PARAID: A gear-shifting power-aware RAID[J]. ACM Transactions on Storage (TOS), 2007, 3(3): 13.

[35] Thereska E, Donnelly A, Narayanan D. Sierra: practical power-proportionality for data center storage[C]//Proceedings of the sixth conference on Computer systems. ACM, 2011: 169-182.

[36] Amur H, Cipar J, Gupta V, et al. Robust and flexible power-proportional storage[C]//Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010: 217-228.

[37] Hsiao H I, DeWitt D J. Chained declustering: A new availability strategy for multiprocessor database machines[C]//ICDE. 1990, 90: 456-465.

[38] Partition_problem[EB/OL].http://en.wikipedia.org/wiki/Partition_problem, 2014-09-24/2014-10-15

[39] Barroso L A, Hölzle U. The case for energy-proportional computing[J]. IEEE computer, 2007, 40(12): 33-37.

[40] Advanced Configuration and Power Interface[EB/OL].http://www.acpi.info/, 2014-07-23/2014-10-05.

[41] Kaushik R. GreenHDFS: data-centric and cyber-physical energy management system for big data clouds[D]. University of Illinois at Urbana-Champaign, 2013.

[42] Wake on LAN[EB//OL]. http://en.wikipedia.org/wiki/Wake-on-LAN, 2014-10-05

[43] Fio[EB/OL].http://freecode.com/projects/fio, 2014-07-10/2014-10-05.

作者在学期间取得的学术成果

[1] 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，2015.8（已录用）