申请上海交通大学硕士学位论文

面向软件定义存储的云混合存储技术研究

论文作者 _____

学　　号 _____

导　　师 _____

专　　业 软件工程

答辩日期 2018 年 1 月 9 日

Submitted in total fulfillment of the requirements for the degree of Master
in Software Engineering

Research on Software Defined Storage on Cloud Hybrid Storage Technique

School of Software Engineering
Shanghai Jiao Tong University
Shanghai, P.R.China

Jan. 9th, 2018

面向软件定义存储的云混合存储技术研究

摘 要

随着互联网行业的发展，如今的互联网正处于一个信息爆炸的时代。IaaS（Infrastructure as a Service）云常常被用来部署大数据分析和其他各种类型的应用。虚拟机被认为构建 IaaS 云的基础，而虚拟机常常是基于虚拟机磁盘镜像（VMDI）构建的。对象存储可以让云提供商轻松地扩大存储规模，并且它通过备份数据副本的方式保证了存储的高可用性。存储介质方面，SDD 在性能、能耗等方面相较于 HDD 有较大的优势，但是由于其高昂的价格与存储空间，让 SSD 完全替代 HDD 成为企业级数据存储还是不太现实。软件定义存储，由于其通过自治的方式对存储集群的结构进行优化，它更加适合运用于将 SSD 与 HDD 混合的存储系统。

WHOBBS[1] 是本实验室之前关于混合存储的研究，它基于对象存储为虚拟机提供混合存储的块服务。它在 Ceph[2] 的基础上，构建了一套完整的混合存储系统，但是 WHOBBS 由于其单监控器架构，导致监控器成为系统的性能负担。本文设计和实现 DOBBS 系统，它在 WHOBBS 的基础上进行了扩展，DOBBS 为多监控器架构，并将整个集群分割成多个子集群。为了充分利用 SSD 和 HDD 混合存储的高效性，我们提出了局部热均衡的概念，局部热均衡截取了虚拟机的运行时信息，然后动态地计算数据流信息得出合适的对象放置策略，最后在 SSD 和 HDD 间迁移对象。在分割成子集群之后可能遇到子集群间热度分布不均衡的情况，我们提出了一个全新的叫做全局热均衡的概念。全局热均衡主要包括热度不均衡的检测和热扩散散。在热度不均衡检测部分，我们提出了衡量子集群热度的方法以及热度检测的算法。热扩散是全局热均衡的重点，为了解决大规模数据迁移，我们提出了全新的源自于物理学的概念——热扩散。我们只对元数据进行迁移，并且修改子集群的逻辑结构，而后续的数据迁移则是交给局部热均衡来完成的。

在系统设计的基础上，我们对系统加以实现，并介绍了 DOBBS 的各个模块的实现细节和主要工作流。在文章的最后，为了验证 DOBBS 的高效性，我们进行了大量的实验，首先证明了局部热均衡的高效性，其次我们证明了 DOBBS 有能力解决 WHOBBS 监控器的性能问题，以及全局热均衡的有效性。

关键词： 分布式存储　混合存储　热均衡

Research on Software Defined Storage on Cloud Hybrid Storage Technique

ABSTRACT

With the development of the Internet industry, the Internet today is in an era of information explosion. IaaS(Infrastructure as a Service) are often used to deploy big data analytics and other types of applications. Object storage allows cloud providers to easily scale up their storage, and it ensures high availability of storage by backing up data copies. In the aspect of storage media, SSD in performance, energy consumption and other aspects compared to HDD has greater advantage, because of its high price and limited capacity, it is still a long way to go before completely using SSDs for enterprise data storage. Software defined storage is more suitable for hybrid storage system, because its autonomically optimizes architecture of storage cluster.

WHOBBS[1] is our previous research on hybrid storage. WHOBBS, while effective, only supports single monitor architecture as it concentrates all the monitoring tasks on one single node. Thus, single monitor would be a serious problem for WHOBBS. In this thesis, we design and implement DOBBS, which extends WHOBBS with multi-monitor architecture and splits the cluster into subclusters. We put forward the concept of Local Heat Balancing for managing the system efficiently.Local heat balancing, actually, indicates fully exploiting the unique performance potential of SSD, and ensures high accessed data reside on SSDs. We propose a concept named Global Heat Balancing which ensures the whole cluster is in the state of heat balance. Global Heat Balancing includes two parts, heat imbalance detection and heat diffusion.

Based on system design, we implement the system and introduce the implementation details and the main workflows of each module of DOBBS. In order to verify the efficiency of DOBBS, we conducted a large number of experiments. A large number of experiments show the efficiency of DOBBS and the effectiveness of Global Heat Balancing.

KEY WORDS: Distribued Storage, Hybrid Storage, Heat Balance

目 录

第一章 绪论

1.1 研究背景

随着互联网行业的发展，如今的互联网正处于一个信息爆炸的时代。面对信息爆炸的互联网，信息的存储和处理也就产生了海量的数据。因此，传统的数据中心在性能、成本、安全和能耗上遇到了越来越多的挑战。为了应对海量数据带来的挑战，国内外一些大型 IT 公司都建立了各自的云系统和解决方案，如亚马逊的 AWS、微软的 Azure、阿里巴巴的阿里云、百度的百度云等等。

为了高可用性和节约资源，IaaS（Infrastructure as a Service）云常常被用来部署大数据分析和其他各种类型的应用。IaaS 是以体系架构为基础的服务，这些服务包括服务器、存储和网络硬件的提供[3]。可是，如果以物理机的方式提供这些计算资源，那么成本将难以估计并且会造成大量资源的浪费。取而代之，现在的云提供商选择以虚拟化技术来代替物理设备，这样充分降低了成本并且提升了可用性和可扩展性。在 IaaS 云，云提供商通过虚拟机监控程序虚拟化一台物理机，这样的虚拟化技术让一台物理机运行多台虚拟机成为可能[4]。因此，虚拟机被认为构建 IaaS 云的基础，而虚拟机常常是基于虚拟机磁盘镜像（VMDI）构建的。传统的云存储解决方案，如 NAS(Network Attached Storage) 越来越难以适应大量的云存储需求，考虑到 NAS 的扩展性低、对非结构化数据没有优化的特点，用 NAS 存储虚拟机磁盘镜像则毫无优势。相较于传统的 NAS 存储，对象存储由于其高扩展性可以更好地适应现在的大规模云存储。对象存储可以让云提供商轻松地扩大存储规模，并且它通过备份数据副本的方式保证了存储的高可用性，更加重要的是对象存储允许存储大规模的非结构化的数据。对象存储是以对象为粒度管理数据，与之相对应的就是文件系统存储和块存储。对象存储的一个重要设计原则就是抽象底层存储，使得底层存储于上层应用分离开来。对象包含了额外的描述性信息和属性，这样可以让对象可以被快速地索引和查找现在一些云服务提供商，如亚马逊（AWS S3），微软（Microsoft Azure），还有一些开源项目如，OpenStack[5]（Swift）、Ceph 和 OpenIO 等等都是基于对象存储的。

对于存储介质方面，目前的主流存储介质包括了半导体存储和磁性磁性存储两大种类。对于半导体存储，其早在 2006 年笔记本和个人 PC 厂家就开始使用的 SSD 逐渐替代更多传统的磁体存储器 HDD。与传统机械硬盘 HDD 相比，SSD 访问更加快速、可靠、低能效，但也更昂贵。在大量实验比较下，SSD 的随机访问时间一般少于 1ms，而 HDD 的随机访问时间则是在 2.9 至 12ms 之间，因此 SSD 的随机访问速度是 HDD 的 10 倍以

上[6]。纵然，SDD 在性能、能耗等方面相较于 HDD 有较大的优势，但是由于其高昂的价格与存储空间比如传统硬盘仍有较大差距。一个 SATA WCD SSD 的存储成本是 1.375 美元/GB，一个 SATA WCD HDD 的存储成本是 0.068 美元/GB。为了充分利用 SSD 的性能潜力，现在关于混合存储的研究主要是将 SSD 作为前端存储（缓存）或作为主存。如果将 SSD 作为存储缓存则必须面对两个主要问题：缓存和主存之间的一致性问题以及 SSD 的容量浪费问题。那么，如果将 SSD 作为主存，那么混合存储的关键性问题在于把哪些对象放置于 SSD。

尽管虚拟化技术帮助云计算系统解决了很多的问题，如高效利用硬件和快速虚拟机的迁移等等，但是虚拟化技术所带来副作用是存储系统会被分割成多个层级，这样就增加了端到端复杂度[7]。端到端（end-to-end）指的是从虚拟机上的应用程序开始到底层磁盘。例如考虑到不同的存储介质，虚拟机产生 IO 请求后由虚拟机管理程序通过以太网将请求发送到块存储设备，但是由于存储介质的不同，可能需要特别指定存储介质，这样则大大增加了端到端所经过的层数，并且需要系统管理员配置 IO 请求时访问的数据路径（Data Path）。软件定义存储在 2013 年被提出，用于简化云计算存储中的端到端复杂度。软件定义存储的主要特性包括自治性、虚拟化数据路径和高扩展性[8]。自治性则表示减少系统管理员对系统架构管理的开销；虚拟化数据路径对于上文所举的例子则表示虚拟机到存储设备的路径并不实际的物理存储数据路径而是由系统所控制的虚拟化路径；高扩展性是相较于传统数据中心存储而言，具备更高的扩展性。实际上，软件定义存储允许应用和数据生产商管理通过存储架构管理它们的数据而不需要额外地请求存储管理员，也不需要准备额外的指令进行操作。那么，在出现多个存储介质之后，软件定义存储则是一个合适的解决方案，这样系统可以自治地对数据的放置进行决策，而不需要系统管理员再显式地调整数据的存放位置。

实验室学长在本文开始之前已经有了关于软件定义存储的 SSD/HDD 混合存储的的研究，分别是 WHOBBS[1] 和 MOBBS[9]，它们基于对象存储为虚拟机提供混合存储的块服务。它们都在 Ceph[2] 的基础上，构建了一套完整的混合存储系统。它们最根本的思路是，将过热的数据放置在 SSD 上，反之将较冷的数据放置在 HDD 上。这两个系统都是通过对数据对象的数据流监控，数据流包括数据对象的大小、特定的访问类型等信息。它们将这些数据流信息进行处理之后得到每个对象的热度值，然后根据每个对象的热度值制定一个合适的迁移策略，使得每一个对象对象都放置在合适的存储设备上。WHOBBS 是 MOBBS 的升级版本。MOBBS 将对数据流的监控分析模块和虚拟机管理程序都放在客户端节点，因为监控分析模块会占用客户端节点的计算资源，这样势必会直接影响到虚拟机管理器和虚拟机的运行效率。WHOBBS 在 MOBBS 的基础上修正了这个问题，它把监控分析模块分离出来，放在一个独立的节点上去运行，它叫做 Monitor。

这样就是支持多个客户端节点同时运行，并且把 MOBBS 存在的性能隐患也完全消除。WHOBBS 相较于 MOBBS 虽然可以很高效，但是考虑到 WHOBBS 是单监控节点架构，那么当客户端的数量增加起来之后，监控节点的计算压力也会随之增长。这样势必还是会影响到整个系统的性能。不仅如此，因为 WHOBBS 的 Monitor 成为了整个系统的关键性节点，那么 Monitor 节点一旦宕机，整个系统都不能正确运行。

1.2 研究目标和内容

正如上文所说的，利用 SSD 和 HDD 构建混合存储可以充分利用 SSD 的高性能优势也可是有个相对经济的价格，并且由于现在 IaaS 云中存储的主要数据也是虚拟机 VMDI，那么可以利用混合存储为虚拟机 VMDI 提供块存储服务。本课题的研究目标就是构建一个高可靠、高可用性的软件定义的混合存储系统。因此，我解决了上文提到的 WHOBBS 存在的性能缺陷，设计并实现一个中心化双层系统——DOBBS。因为 DOBBS 是也是基于 Ceph 的，并且将实际的文件存储在对象存储设备（Object Storage Devices）中。通过修改 Ceph 的相关源代码，DOBBS 的底层实现对上层用户是透明的。为了解决 WHOBBS 的性能缺陷，DOBBS 将 WHOBBS 原有的 Monitor 节点的计算分摊到多个同级 Monitor 上，它们的实际功能和 WHOBBS 的 Monitor 节点的功能相似，监测虚拟机的数据流信息并制定合适的迁移策略，但是和 WHOBBS 不同的是，DOBBS 运用的是中心化双层的 Monitor 架构。基于这个架构，存储集群被分割成了多个子集群，每个子集群又由独立的 Monitor 节点所控制。DOBBS 提出了热均衡的概念，热均衡旨在将集群过热的节点数据进行迁移使得整个集群达到一个宏观上负载均衡的状态。对于 DOBBS，我们又将热均衡分为两类：局部热均衡和全局热均衡。局部热均衡指的是对于每个子集群内部的数据根据其热度数据合理地分布在 SSD/HHD 中。DOBBS 提出了全局热均衡的概念，全局热均衡的引入是考虑到可能存在某个时间段内某个子集群承受了大量的数据访问，而其他子集群仍处于闲置状态，所以需要全局热均衡来将这个过热子集群的“热量”散播出去。因此，软件定义存储在 DOBBS 被体现在两个方面，在局部热均衡中，我们通过修改 Ceph 源代码获取虚拟机动态数据流，并通过分析动态数据流，将对象在 SSD 和 HDD 间迁移，整个过程对于上层虚拟机并不可见；对于全局热均衡，DOBBS 动态检测各个子集群的热量信息，然后将过热子集群的“热量”进行扩散，这个过程实际对系统逻辑架构进行了修改，而不需要系统管理手动管理。

本课题的目标是，第一，优化现存分布式混合存储中的监控节点，创建多个监控节点，实现监控节点的分布式，从而保证分布式混合存储系统的高可用性；第二，根据用户行为，对存储内容的访问情况，动态调整存储策略，下层的混合存储实现对用户透明；第三，混合存储的关注点不再具象到 HDD/SDD 这两种存储介质，而是探索不同存储介

质的特性，动态适应混合存储系统。

1. 分布拓扑结构监控器：由于在现存的分布式混合存储系统中，都存在一个监控节点，但是由于只有一个监控节点，这样使得管理节点往往成为性能的瓶颈。为了解决这个问题，需要创建多个管理节点，也就是对管理节点分布式。
2. 分布式全局热均衡：本课题的第一个目标就是创建多个监控节点，实现监控节点的分布式。那么这将带来一个问题，每个监控节点分管不同的混合存储节点，因此监控节点可以较好得保证各自分管的混合存储节点内部的热均衡，但是这只是局部的热均衡并不是全局的。全局热均衡需要提出适当的算法、合适的策略、完善的数据迁移方法。
3. 面向多存储介质的数据放置方法：不同存储介质在对不同类型或频率的 IO 请求都会存在性能上的差异。所以要通过实验来确定存储介质的特性，并根据其性能上的特性设计数据放置方法，然后将计算出的放置方法应用到混合存储的数据迁移策略中。
4. 实验验证：通过实验验证分布式拓扑结构监控器是否可以解决 WHOBBS 的性能缺陷，以及采用中心化两层架构后系统的性能和可用使用是否得到了提高。

1.3 研究意义

由于现在 IaaS 云的流行，越来越多的企业将自己的服务部署在 IaaS 中。我们知道，IaaS 是以计算资源为服务进行商业化出售，而虚拟机则是在 IaaS 云中扮演至关重要的地位。传统的云存储解决方案，如 NAS 并不能有良好的可扩展性，并且对于云服务的高可用性也不能给予保证。相比于 NAS，对象存储的高扩展性以及对虚拟机磁盘镜像的存储可以做到十分充分的优化则让它在 IaaS 存储中崭露头角。另外，在存储介质方面，SSD 相比于 HDD 有着优异的读写性能和较低的能耗，越来越多的厂家选择渐渐用 SSD 来替代原有的 HDD。可是，SSD 又由于其高昂的价格和受限的容量，并不能完全取代 HDD。所以，将 SSD 和 HDD 混合起来构建混合存储则变得尤为重要。软件定义存储作为新兴概念，它让存储集群处于自治地状态而不需要存储管理员显式地调整存储策略。考虑到 SSD/HDD 的混合存储方式增加了虚拟机存储的端到端复杂度，因此使用软件定义存储的存储方式可以更好的解决混合存储所带来的问题。

WHOBBS 虽然已经可以高效地将数据在 SSD 和 HDD 上动态地迁移，但是 WHOBBS 的监控器只有一个，因此它就必须承担所有客户端的数据流信息，并且还将面临宕机后整个集群的性能受到影响的问题。所以解决 WHOBBS 的监控器所存在的问题则是十分紧迫的。DOBBS 在 WHOBBS 的基础上进行了扩展，我们为了解决单一监控器带来的问题，我们用多个同级的监控器来代替单一监控器，这样就可以把单一监

控器的计算任务平均分配到各个监控器上，这样就充分解决了 WHOBBS 的监控器所存在的问题。

1.4 论文组织结构

本文正文包含六章，本章为绪论，首先系统性地介绍并分析了本课题的研究背景和研究意义，在此基础上提出了本课题的研究内容和目标。其他章节的内容组织如下：

第二章介绍了和本课题有关的国内外研究现状。该章节从本课题所涉及的研究内容出发，调研并分析了与之相关的现有的国内外研究成果与不足，并提出了可以进一步研究和改进之处。

第三章介绍了本课题提出的分布式混合存储系统的体系架构设计。该章节首先从整体上介绍了分布式混合存储系统的体系架构，之后对架构中的具体模块进行了详细的分析和设计。

第四章介绍了系统实现。该章节分别介绍了分布式混合存储系统的体系架构各个模块的实现。

第五章介绍了系统实验与验证。该章节详细地介绍了为验证本课题所提出的局部热均衡和全局热均衡的有效性以及对 WHOBBS 的监控器节点性能问题解决的验证。

第六章为总结与展望。该章节总结了本课题的主要研究工作与贡献，并在此基础之上给出了今后的研究方向。

第二章 国内外研究现状

随着云计算的高速发展，每天在互联网中都会产生海量的数据。分布式存储作为云计算的基础，近些年有大量的关于分布式存储的研究。以下将通过：分布式存储及对象存储、混合存储的应用以及分布式存储负载均衡等几个方面来进行国内外研究现状的综述。

2.1 分布式存储及对象存储

分布式存储作为云计算的基础，谷歌在 2003 年提出的 GFS[10] 是在分布式系统领域具有划时代意义的论文，它的提出了如何采用廉价的商用计算机集群构建分布式文件系统，论文中提出的控制与业务相分离、将容错的任务交由文件系统来完成、多副本机制等等，这些概念大多都变成了之后分布式存储系统设计研究的金科玉律。在随后的 Apache 的 HDFS[11]，在一些设计上与 GFS 高度一致，它提供了一个高度容错的分布式文件系统，能够提供高吞吐量的数据访问。

对象存储的概念在上个世界 90 年代就应该在研究社区中出现，但是在当时由于云计算还没有广泛流行，所以对象存储也没有被广泛运用。对象存储，是将对象作为单位管理数据，与之对应的文件存储则是以文件系统结构为基础，而块存储则是以块为单位管理数据。[12] 介绍了对象存储发展的几个阶段，首先是由 TIO OSD 协议作为第一阶段的标准，第二阶段则是 iSCI。Ceph 作为当今非常流行并且成熟的分布式对象存储，已经集成到了 Linux 内核中。Ceph 是一个基于对象存储的 PB 级分布式文件系统，拥有极佳的可靠性和可扩展性。根据配置的不同，Ceph 可以分别提供上述的块存储、文件存储和对象存储三种服务，但是所有的服务均基于一个叫 Rados 的对象存储系统。

2.2 混合存储的应用

SSD 的读写性能、能耗等各个方面较于传统的 HDD 有较大的优势，因此近些年来 SSD 被越来越多的用于计算机存储领域。[6] 中揭示了 SSD 和 HDD 性能上的差距，并且建议用户通过价格和性能的比率来选择合适的存储设备，对于不同的数据流 SSD 和 HDD 在性能上的表现也有一定的差别。对于顺序数据流，SSD 在性能上的提升相较于其高昂的价格就并没有太多的优势，而对于随机数据流 SSD 对于读写性能的提升十分显著。因此考虑到 SSD 高昂的价格，以及其在特定数据流表现的性能差异，将 SSD 与

HDD 作为存储设备混合使用将会是一个研究的重点。

对于关于 SSD/HDD 混合存储的研究主要分为两类，一类是考虑到 SSD 相较于 HDD 具有节能、高效的特性，将其用作存储缓存；另一类是考虑到 SSD 较高的读写性能、非易失性以及拥有大容量，将 SSD 作为主要二级存储。以下将从这两类介绍 SSD 在混合存储中的应用。

在 [13] 中，作者提出了一个混合式的对象放置和移动架构，以及适应性学习算法来识别数据对象的访问热度。该论文中，混合式架构通过将数据对象在慢速的 HDD 和快速的 SSD 间相互移动来适应数据对象访问热度的动态变化，同时数据对象的放置及移动规则也可以满足用户对它们 I/O 操作的需求。基于以上的目标，论文提出了一个基于马尔科夫链的识别模型来预测数据对象在外来一段时间的访问频率，这个识别模型是以对象的访问模式为训练集训练马尔科夫链模型，一旦的得到预测结果就开始在 HDD 和 SSD 间迁移数据。为了适应用户自定义的放置规则，论文提出了一个数据放置引擎，以用户的放置需求作为输入产生合适的解决方法。[13] 的核心思想就是，作者将 SSD 和 HDD 的混合存储用到了高性能计算领域（HPC），因为是它基于预测的思想，所以可以在整个系统数据发生变化之前就将数据进行迁移，这样大大提高了整个系统的 IO 性能。

[14] 将目光放在了对 SSD/HDD 混合存储在性能、耗能和寿命问题做出了研究。[14] 设计并实现了一个多用的混合存储驱动，它实现的混合存储系统包含两个模式——分层存储模式和缓存模式，这两个模式分别将 SSD 用作存放热点数据的主存和用于存储热数据的缓存。该系统从混合虚拟设备接受数据访问请求，并将结果重定向给下层物理设备。而 [14] 在区分冷热数据的方式与 [13] 相似，也就是将热的、访问频率高的数据放在 SSD 中，反之冷的、访问频率低的数据放在 HDD 中，当数据的冷热情况发生变化，再动态地将数据进行迁移。但是它把研究的重心放在了对性能、耗能以及寿命的权衡。它得出的结论是，为了得到高性能，势必会导致能耗的增加；对于论文中，数据分块越大性能提升越大，同时会造成 SSD 寿命的快速衰减。[13] 和 [14] 都分别研究了以 SSD/HDD 为主体的混合存储，并且研究的重点都在于利用 SSD 在读写方面优异性能，将热点数据存储在 SSD 中，冷的数据存储在 HDD 中，再通过这一基本框架做进一步的研究。

而 [15] 的研究重点面向了更加特定的环境，即面向高性能计算环境下基于 SSD 突发缓存的研究。[15] 的研究不仅限于 SSD/HDD 这两种局限的存储介质类型，而是更加抽象的存储分层，它以 SSD 作为突发缓存，而底层存储是并行文件系统（PFS），突发缓存作为客户端与并行文件系统之间的缓存，常常要存储检查点（checkpoint），而检查点是需要频繁读写的，这样会造成 SSD 的寿命大幅缩短，并且由于高性能计算环境数据流往往是高速、大流量的，所以对作为突发缓存的硬件寿命面临着极大的挑战。基于这个问题，[15] 设计了一个适应性算法基于高性能计算环境下系统数据流的变化，动态

的切换检查点的放置策略，这样既保证了系统的高效运行也大大延长了突发缓存的硬件寿命。

Apple 公司在 2012 年发布了自己的混合存储设备 Fusion Drive，并将其应用到 Apple 的各个产品中。Fusion Drive 融合了 HDD 和 SDD，系统会自动管理访问最频繁的应用程序、文件、照片或者其他数据来存储在 SSD 里，而将很少访问或使用的文件留在机械硬盘上。[16] 为 Apple 开发 Fusion Drive 提供了思路。在 [16] 中，作者设计并实现了一个高性能混合存储系统 Hystor。作者在论文中提出了混合存储必须面对的三个问题，分别是 1) 高效地识别性能关键的数据快并充分利用 SSD 的存储潜力；2) 为了精确地刻画数据访问模式必须高效地保存数据访问历史；3) 系统实现的过程中避免对现存操作系统内核的改变。[16] 实现的 Hystor 充分解决了上述的三个问题，它通过监控 I/O 自动地学习数据流访问模式并识别出性能关键的数据块，只有带来最大性能收益的数据块才可以从 HDD 重映射到 SSD 中；其次，Hystor 设计了一个高效的机制即数据块表（Block Table）去存储数据块访问的历史信息，这样可以为长期的优化提供资源；它是一个 Linux 的内核模块，只对内核做了很少的修改。因此 [16] 作为一个具有里程碑意义的论文，颠覆了以前仅仅利用 SSD 优秀读写性能而将 SSD 作为缓存的传统模式，而提出了将 SSD 和 HDD 混合做个主要存的新模式。

对于 [17]，它将混合存储用到了 HDFS（Hadoop Distributed File System）中，由于原有的 HDFS 并不能用来处理多层存储，。HDFS 认为所有底层存储组件都具有相同的 I/O 特性，这样会造成对资源的浪费和不高效实用。因此，在 [17] 中，作者设计并实现了一个针对 Hadoop 的多层存储系统 hatS，它在逻辑上把相同存储类型的节点分到同一个层，这样单独地管理各个存储层，hatS 就能够捕捉分层信息并充分利用这些信息从而达到高的 I/O 性能。与 HDFS 不同的是 hatS 使用分层感知的方法在不同之间复制数据，这样使得高性能存储设备可以高效地处理更加处理大量 I/O 请求。hatS 的实际做法是利用 Hadoop 的多数据备份的规则，并结合存储分层，让尽量多的 I/O 请求作用到性能高的节点上，从而提升整个系统的吞吐能力。

[9][1] 两篇论文，将混合存储和当今非常流行的对象存储系统 Ceph 相结合，[9] 设计并实现了基于 Ceph 的混合存储系统 WHOBBS。由于原始的 Ceph 系统并没有对底层存储介质的类型给予优化，它是根据 CRUS H 算法[18] 静态地设置 osd 权值，根据权值放置数据，这个过程是静态的，并不能使用数据访问热度的动态变化。因此 WHOBBS 动态监测每一个数据对象，并且保存数据对象的历史访问记录，根据用户自定义算法来指定迁移策略，即 SSD 与 HDD 之间的迁移策略。在设计迁移策略之前，[1] 做了大量实验验证 SSD 与 HDD 在 Ceph 不同数据流下的性能，并根据实验结果指定合理的数据对象迁移策略。

综上所述，现今关于混合存储的研究大多都着眼于利用 SSD 优秀的存储性能以及低能耗等特点，但是由于 SSD 高昂的价格、有限的寿命以及在部分数据流方面表现的差异，所以现在仍用很多数据存储公司权衡这方面的利弊，选择了混合存储架构。如现在的 Apple FusionDrive、Microsoft 公司的 Ready Drive、西部数据公司的 SSHD 等等。而这些产品大多都是面向单机存储的，它们并没有充分应用到分布式存储的环境中，正如 [17] 所说，现在的大部分的分布式计算框架对待底层存储都没有根据存储的类型来进行优化。

2.3 分布式存储负载均衡

在大规模云计算环境中，负载均衡技术通过保证每一个计算资源可以获得公平的计算请求，达到了用户满意度，同时提高了资源利用率。合适的负载均衡目技术为的就是减小资源竞争、保证可扩展性、避免性能瓶颈等等。

Paiva J. 等人在 2015 年提出了 AutoPlacer[19]，它面向现今最流行的键值存储提供数据放置策略，AutoPlacer 的自调节数据放置策略可以较好地达到负载均衡目标。AutoPlacer 使用了一个轻量级的分布式优化算法，这个算法不断迭代，在每一次迭代中它以一种去中心化的方法优化前 K 个热点数据的放置。热点数据实际上产生最多远程操作的数据对象。为了识别出热点数据，并且不会有对整个系统的性能带来太大的影响，AutoPlacer 选用了一种全新流分析算法来追踪数据流（data stream）中前 K 个最常被访问数据对象。AutoPlacer 不仅支持细粒度的数据放置，还使用了全新的概率数据结构 PAA 来保证单跳路由的延迟最小。

[20] 提出了一个全新的想法，它使用倾斜的复制策略来构建节能存储集群。它提出的数据复制策略实际是基于数据访问行为的。这个策略仅仅复制访问热点数据，而这些热点数据根据 80/20 原则（80% 的数据访问往往只访问 20% 存储空间的数据）恰恰只是整个数据齐群 20% 的数据量。因此，可以将集群分为热节点集和冷节点集。热节点存储较少数量的数据副本，并且常常处于活跃状态来保证整个系统的 QoS 。冷节点则与之相反，它存储数量较大且访问频率较小的冷数据，并且这些存储节点被通过一定的方式（CPU 降频、磁盘降低转速）来进入节点模式。作者在论文中通过模式实验中验证了自己的观点，其所提出的系统不仅可以保证低能耗也可以保证整个系统的性能不受到影响。

Soni G. 等人另辟蹊径提出了全新的云数据中心的负载均衡方法[21]。这个负载均衡方法面向云数据中心的虚拟机数据流。每一个来自用户的请求都从数据中心控制器发起，数据中心控制器遍历中心负载均衡器，通过遍历结果来分配请求。中心负载均衡器会维护一个包含 id、虚拟机状态以及虚拟机优先级的数据表。中心负载均衡器分析数据表并且找到最高优先级的虚拟机，然后检查其状态，如果它的处于可用的状态则返回虚

拟机 id 给数据中心控制器。最终，数据中心控制器将会把某个数据请求分配给特定的虚拟机。作者通过实验验证了中心负载均衡方法的可行性。

Deng Y. 等人提出的负载均衡技术[22] 进行了创新，他们利用了物理学的热扩散原理来处理分布式虚拟环境（DVE）的负载均衡问题。在论文中，作者使用了一种混合式的负载均衡策略，也就是将全局负载均衡和局部负载均衡相融合。全局负载均衡指的是，存在一个中心节点来管理所有服务器的负载情况，它从各个服务器收集负载数据，并制定负载策略，这样做的优点就是可以准确地实现负载均衡，但是计算量较大，需要额外的计算资源。与全局负载均衡相反，局部负载均衡则不存在中心节点，每个服务器通过其邻接的服务器负载数据来制定自己的负载均衡策略，这样做的优点在于计算量较小，但是负载的结果不并准确。在 [22] 中，作者使用了热扩散原理，即高温区域的热量会向低温区域扩散，来处理负载均衡问题，并且将热扩散原理与全局负载均衡和局部负载均衡相结合，提出了局部扩散和全局扩散。对于热扩散在负载均衡中的应用，高度概括为，过热服务器将自己的部分数据流交给邻接较冷的服务器去处理。作者通过大量的实验和数学证明验证了基于热扩散的负载均衡技术的高效性。

综上所述，现在已经有许多在分布式系统负载均衡方面的研究，但是可以发现一点，就是这些 [19][20][22] 关于负载均衡的研究都是相对静态的，并且大多是关于数据放置如何达到负载均衡，并没有考虑计算节点过热时如何动态均衡计算任务。

2.4 本章小结

本章通过分布式存储和对象存储、混合存储和分布式存储负载均衡，这三个方面介绍了与本课题相关的国内外研究现状。分布式存储和对象存储介绍了，现在主流的分布式存储框架以及 Ceph 等对象存储系统，并叙述了这些系统在特点及应用场景。混合存储部分，从 SSD 和 HDD 的性能差距着手阐述了混合存储的必要性和意义，之后分别介绍了国外关于混合存储的研究。混合存储的主要研究还是主要面向于将 SSD 运用为主存，或是将 SSD 运用为缓存。但是现在关于混合存储的研究大部分还是面向单机存储的，分布式存储的应用相对较少。分布式负载均衡部分，介绍了多种分布式存储负载均衡的解决方案，但是可以清楚地看到，大部分解决方案都是满足静态的负载均衡，而对于运行时（动态）的负载均衡都没有涉及。

第三章 系统设计

DOBBS 是为了适应大规模 IaaS 云的底层存储，利用对象存储对非结构化数据 VMDI 进行大量优化，并且在 WHOBBS 的基础上解决了其性能缺陷。本章我将介绍 DOBBS 的系统架构设计以及各模块设计。

3.1 设计目标

对于混合存储的研究，我们已经有了 WHOBBS 和 MOBBS 两个系统。DOBBS 是在 WHOBBS 的基础上进行的改进。WHOBBS 通过动态监测虚拟机 VMDI 对象的访问数据流信息，然后客户端收集到对象的数据数据数据流信息之后定时发送给独立的监控器服务器。监控器服务器中的分析器模块则负责收集对象信息，然后根据特定的对象热度计算算法，让较“热”的对象放置于 SSD 而较“冷”的对象放置于 HDD，并且通过动态监测实现 SSD 和 HDD 上对象的动态迁移。因此，WHOBBS 系统主要包含三个部分，客户端、监控器和存储集群，它们通过网络连接。其中整个系统的核心就是监控器，监控器保存了每个对象的数据流信息，还一直在计算对象的热度和发送迁移指令。在 WHOBBS 的实现上，作者让系统仅存在一个监控器服务器，它负责收集所有接入客户端的信息并产生决策。

考虑 WHOBBS 的系统架构，我们可以看到监控器节点是整个系统的大脑和决策者，它不仅保存数据还进行大量的计算。在 [23] 一书中，作者提到基于 Web 的计算和面向服务的计算必须要保证高可靠性，用户才会信任它。但是 WHOBBS 的单一监控器架构导致了这个监控器成为系统的唯一关键节点，这就意味着一旦这个节点所在的物理机宕机、网络不能访问或者软件出现异常，对整个系统都会造成极大的影响。可以试想，如果监控器宕机，那么客户端则不能将记录的对象数据流信息发送给监控器，并且底层存储集群内的对象不能被迁移，整个系统则处于一个没有被优化的状态。

对于 WHOBBS 的架构中监控器可能存在的唯一关键节点的问题，本实验室的 RHOBBS[24] 就是用于解决这个问题。它采用了多复制的策略，架构不再采用单一监控器的模式，而是采取了多个类监控器节点，叫做 Shuffler。它们保存客户端发送过来的对象数据流信息的相同副本，并且通过 Raft 一致性协议保证了 Shuffler 间副本的一致性。并且它基于 Raft 算法，实现了 Shuffler 的错误恢复，一旦出现宕机的情况可以保证可以通过其他 Shuffler 来进行数据的热度计算等工作。

但是可以想到，由于 WHOBBS 的监控器保存的数据仅仅是对象的数据流信息而并

不是实际的对象数据，那么如果监控器宕机，虚拟机仍然可以和底层存储进行交互，只是 SSD 不会被充分利用，实质上虚拟机的运行不会受到太大影响。因此，如果从保证监控器的高可用性的角度出发去优化没有特别紧迫的重要性。我们回到 WHOBBS 的系统设计，监控器只有一个，它要接受所有接入客户端的数据流信息，然后将信息储存下来，再调用分析线程逐渐计算所有对象的热度。考虑监控器的实现，它将数据流信息放置于内存中，并且分析线程则不断地对所有内存内的数据排序、遍历和运行热度计算算法。虽然热度计算算法的算法复杂度并不高，但是对所有数据排序是会占有较多的 CPU 时间。如果当系统逐渐庞大，有越来越多的客户端接入系统，同一个监控器节点要同时接受所有客户端的网络连接。现在 WHOBBS 的实现是通过多线程处理客户端的网络连接，如果客户端数量激增，线程数必然会达到系统所能支持的上限，这样新的连接可能会被忽略。同样，如果大量客户端接入系统，监控器所保存的数据流信息也会越来越多，占用内存会不断的增加，而每次分析线程的迭代时间也会增长。这样对于高性能混合存储来说不能容忍的。

对于上面提到的问题，在物理资源首先的情况下应该考虑扩大系统规模即横向扩展系统 [25]。基于这样的设计原则，我们也试图横向扩展 WHOBBS 的监控器。如果单一监控器会承受所有客户端的压力，那么我们创建多个监控器，使客户端不是向一个单一的监控器发送数据流信息。我们让客户端发送数据流信息尽量平均到所有的监控器上，这样就可以减少监控器的压力。对于客户端如何接入，以及系统如何组织，在本章后半部分有详细的描述。

3.2 DOBBS 架构介绍

所谓混合存储系统，就是将 SSD 与 HDD 共同组合而成的存储系统。DOBBS 的系统设计是基于 Ceph 实现的，而 Ceph 的文件实际是存储在对象存储设备中的（Objects Storage Device）。为了提升 DOBBS 系统的高扩展性，我们借用了 Ceph 存储池（Storage Pool）的概念。存储池实际是部分对象存储设备的逻辑集合，一个存储池可以包含多个对象存储设备，而对象存储设备是挂载在物理存储介质上的。因此对于 DOBBS，我们有两种类型的对象存储设备，分别是 SSD 对象存储设备和 HDD 对象存储设备。之后，我们将部分 SSD 对象存储设备和部分 HDD 对象存储设备共同组织成一个混合存储池（Hybrid Storage Pool）。混合存储池中的对象存储设备是可以由用户配置的，通过这样的组织，DOBBS 的底层存储集群可以简单高效地扩展。一般性的，每个混合存储池都至少包括一个 SSD 对象存储设备和至少一个 HDD 对象存储设备。

如图3–1所示为 DOBBS 的系统架构图。DOBBS 是软件定义的存储系统，软件定义的存储系统将数据面与控制面进行分离，有一个中心化的控制器来负责控制和监控系

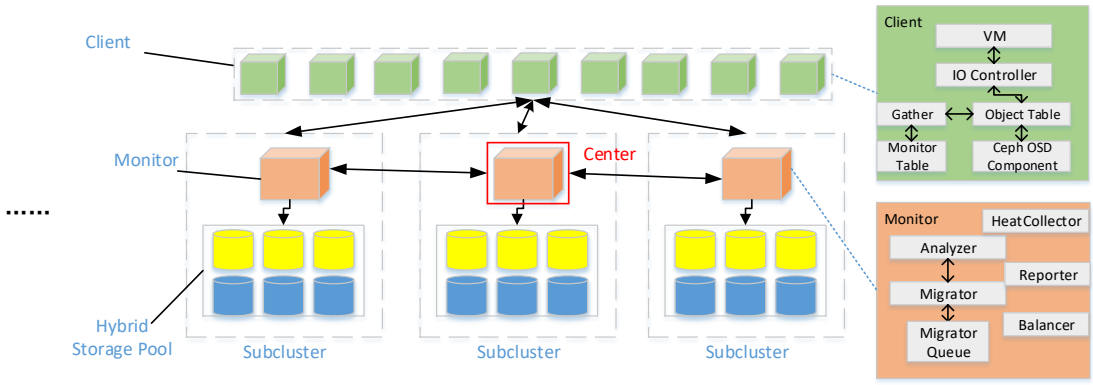

图 3–1 DOBBS 系统架构图

Fig 3–1 Architecture of DOBBS

统的数据流[26]。基于软件定义存储的基本架构，DOBBS 可以分为两层，一层是存储集群也就是混合存储池，一层是监控器（Monitor）。DOBBS 包含大量的混合存储池以及监控器。存储集群正如上文所说的，它是有大量的混合存储池组成的。为了系统的高可靠性和高性能，我们将一个监控器和一个混合存储池绑定在一起，组成一个子集群（subcluster）。之所以要划分成子集群是因为多个 Monitor 如果同时管理所有的混合存储池势必会造成管理的混乱，并大大降低系统的可扩展性。引入子集群则是可以充分提升系统的可扩展性，并使整个系统易于管理。监控器在 DOBBS 则是起到至关重要的作用，它的作用包括监控虚拟机的数据流信息并生成合适的迁移策略，还有检测其所在子集群的热度并向中心节点（Center）汇报，最终执行全局热均衡迁移指令。在 DOBBS 中，子集群仅仅只是逻辑上的概念，虚拟机 VMDI 则被均匀的分布在各个子集群的混合存储池中。监控器所监控的数据流信息也仅仅是其所在子集群对应数据的数据流信息。客户端（Client）则是虚拟机管理器（VMM）所运行的机器，它们是 DOBBS 所提供服务的对象，客户端在接入 DOBBS 之后，与底层存储系统交互并进行数据读写。中心节点则是 DOBBS 的监控器集群的大脑，它的主要作用是收集每个子集群的热度信息，并保证整个系统的热均衡。

我们将系统分为多个模块，模块间互相依赖并支持高效扩展。在客户端上，如果让虚拟机管理器直接访问 Ceph 的块存储，那么它只需要调用 Ceph 提供的 OSD Component 接口即可，但是为了获得虚拟机的数据流信息我们 VM 与 Ceph 存储设备的交互中间增加了一层 DOBBS 监控组件。因为 Ceph 是对象存储系统，所以虚拟机访问的粒度都是一个个对象。如图3–1所示，IO Controller 负责截取 VM 的访问请求，然后通过查询 Object

Table 来知道对象是在哪个 OSD 上，这是因为在数据迁移的过程中对象被频繁迁移，而 Ceph 访问对象是通过对象 ID 和 OSD 共同组合访问，因此需要一个数据结构在存储对象 ID 和 OSD ID 的映射。Gather 模块类似于一个计数器，它会记录每一个单位时间间隔内的对象的访问次数、访问类型等信息，在记录之后再定时发送给 Monitor。Monitor Table 则是用来维护 OSD ID 和子集群 ID 映射的，因为在 DOBBS 在保证整个系统热均衡的过程中需要对 OSD 上的数据进行跨子集群迁移，所以可能会导致某个 OSD 的位置发生了改变。最终，Ceph 根据 VM 请求的对象 ID 和 OSD ID 来和底层存储进行交互。

Monitor 主要有六个模块。Analyzer 模块是 Monitor 的大脑，它负责接收客户端传送过来的虚拟机数据流信息，然后根据内置算法来生成合适的对象迁移命令，命令产生后将迁移命令传递给 Migrator 模块执行。Migrator 得到迁移指令之后，进行后续的上锁等等操作，然后再将最终的指令缓存在 Migrator Queue 中，这个队列的长度是 DOBBS 所支持的最大迁移数量，这个数量当然也是可配置的。在 Monitor 上还有三个独立的模块，Reporter、HeatCollector 和 Blancer，他们都是用来负责系统热均衡的。

DOBBS 提出了两个重要的概念，局部热均衡和全局热均衡。对于局部热均衡，我们数据将对象分为两类，热对象和冷对象。所谓热对象是指的那些访问频率高，短时间内会被多次访问的对象。相反，冷对象则是指那些访问频率并不高的对象。为了充分利用 SSD 的性能优势，我们将热对象都放置于 SSD 上而将冷对象都放置于 HDD 上，然后动态地监测数据对象的冷热变化来在 SSD 和 HDD 之间迁移对象。局部热均衡只是针对每个子集群内部的局部热均衡，因此每个 Monitor 只能保证所在子集群的局部热均衡。如果将存储集群和 Monitor 分割成多个子集群，那么极有可能在某个时刻某个虚拟机产生了非常大量的 VMDI 请求，这样某个子集群会遭受大量的读写请求，数据也会频繁地在 SSD 和 HDD 间迁移，这个子集群的效率将会受到极大影响。提出全局热均衡的概念就是为了消除上述情况带来的性能波动。全局热均衡针对的粒度是 OSD 级别的，它会动态监测每个子集群的热度，然后做出迁移指令，有别于局部热均衡的迁移，全局热均衡的迁移是将 OSD 的数据进行迁移而不是在 OSD 进行数据对象的迁移。

3.3 局部热均衡

因为 SSD 相较于 HDD 在读写速度上都有非常大的优势，并且 SSD 因为并不是采用机械的方式访问数据，所以也更加节能，在现在的数据中心中是存储的主流。但是，目前的 SSD 价格还是十分昂贵，容量更是十分有限。如 [20] 所介绍的，冷数据和热数据实际也是符合 80/20 规律的，那么将数量较少的热数据放置于 SSD 数量较多的冷数据放置于 HDD 的策略是符合 SSD 容量小 HDD 容量大特点的。DOBBS 的局部热均衡则是动态地实现了数据的放置和迁移，我们从 VM 对数据的访问收集数据的信息，做出

迁移策略。局部热均衡在保证存储效率的同时，也让整个存储的实现更加经济。值得注意的是，局部热均衡是针对每个子集群内部的热均衡。局部热均衡的基本设计思路是源自于 WHOBBS 的，而我们是在 WHOBBS 的基础上做出了修改和扩展。

3.3.1 VM 数据流监测和分析

Ceph 是一个对象存储系统，在 VMDI 创建之后，Ceph 会把 VMDI 分割成大量对象存储于 OSD 上[2]。因此局部热均衡锁面向的数据流信息指的是 Ceph 中每个对象的数据流信息。VM 数据流的产生源头是位于 Client 上的虚拟机管理器，如果按照原生 Ceph 的逻辑，虚拟机管理器直接和 Ceph 的 OSD 访问模块交互，进行数据读取。但是，我们需要获取 VM 数据流信息就必须将 VM 与 Ceph 的 OSD 访问模块截断。当 VM 请求 VMDI 对象时，它会先经过 IO Controller 的访问控制。IO Controller 主要实现两个功能，一是通过对象 ID 查询到 OSD ID，二是记录这个对象的访问次数。之所以需要通过对象 ID 查询 OSD ID，是因为 Ceph 访问数据对象是通过对象 ID 和 OSD ID 共同组合访问的，但是局部热均衡会将对象在 OSD 之间频繁移动，所以需要一个数据结构记录每个对象是在哪一个 OSD 上。Object Table 就是用来记录对象位于哪个 OSD 的数据结构。IO Controller 通过对象 ID 在 Object Table 中查询 VM 请求对象所在的 OSD，然后调用 Ceph OSD Component 来后续的对象读写。IO Controller 的第二个功能是，记录这个请求对象的访问次数和访问类型。在 DOBBS 中，我们刻画对象的数据流信息通过对象 ID、访问类型（随机访问或顺序访问）以及单位时间内访问次数。IO Controller 将截取的对象数据流信息交给 Gather 模块来进行统计和整合。Gather 模块在接收到 IO Controller 传递过来的数据流信息之后，记录下数据流信息，然后将单位时间内的数据流信息发送给 Monitor，最后再清空计数器。

当 Monitor 收集到了从客户端节点发送来的数据流信息之后，它开始通过数据流信息计算每个对象的热度。这里提到的热度实际是一个具象化的数值。局部热均衡实际就是通过这个热值作为依据来进行对象的迁移。Analyzer 模块就是主要负责，对象热值的计算以及得出合适的数据放置策略。考虑到虚拟机上运行的应用程序的多变性和复杂性，用一套通用的算法来计算数据对象的热度值并不是很现实。所以，DOBBS 提供了更加开放的方式，我们提供了算法接口 API，这样用户可以直接继承这个接口来适配性地使用自己的热度值计算方法。我们还提供了一个默认的计算方法，沿用了 WHOBBS[1] 的计算方法。

Analyzer 在计算完每个对象权重之后就开始启动周期性轮询机制产生对象迁移指令，它内部会维护一个有序的队列，这个队列描述了每一个对象热度值和其所在的 OSD 编号并且是通过每个对象的热度值降序排序。DOBBS 做 SSD 和 HDD 间的数据迁移是

一种贪心的策略，我们始终保证热的对象可以占满整个 SSD。所以，Analyzer 会周期性轮询有序队列，从对头取出一定数量的对象，让它们放置于 SSD 上。在系统刚刚加载时，我们默认所有对象都位于 HDD。Ceph 默认的对象大小是 64MB，所以根据 SSD 大小来决定多少个对象放置于 SSD 上。例如，128GB 的 SSD 最多可以放置 2000 个对象，那么每次将队列头的 2000 个对象置于 SSD，如果对象原来所在的位置是 HDD，则产生 HDD 到 SSD 的迁移指令。相反，如果遍历到对象是队列 2000 个以后，并且在 SSD 上的，则产生 SSD 到 HDD 的迁移指令。

3.3.2 对象迁移

Analyzer 在产生迁移指令之后，Migrator 则负责实际的对象迁移工作。Migrator 首先会检查，当前的 SSD 是否已满，如果 SSD 已满则会终止掉当前的迁移请求。接下来，Migator 将每一条迁移指令缓存下来，并存入迁移队列 Migrate Queue 中。如果 Migrator 同时让多个对象进行迁移，那么对整个系统的性能将会是灾难性的影响。因为对象的传输将会大大占用网络带宽，并且 DOBBS 为了维护迁移过程中对象的一致性，需要频繁地对 Client 的对象上锁和解锁，那么同时大量对象迁移将会影响 VM 的性能。所以，Migrate Queue 保证了系统同时最大迁移数量，如果系统中已经有部分对象在做迁移，那么后续的迁移指令需要等待之前的迁移完成之后才可以执行。

DOBBS 为了维护迁移过程中的一致性，会使用 Remote Lock[1] 技术来对对象上锁，从而解决迁移过程中对一致性的破坏。试想这样一种情况，Analyzer 所产生的迁移请求恰好是 VM 正在访问的对象，在 DOBBS 中对象迁移的优先级是很高的，所以 Client 上的虚拟机可能在访问这个对象的时候丢失它，这样难免会造成数据的丢失，在一些数据关键型的应用中将带来难以估计的影响。Remote Lock 就是做到了由 Monitor 向 Client 发起上锁请求，Client 上锁成功后会向 Monitor 发送上锁成功请求。只有 Monitor 在收到上锁成功请求后才会进行后续的迁移。

3.4 全局热均衡

3.4.1 概念引入

之所以要提出全局热均衡的概念要回到为什么将存储集群和 Monitor 集群划归成子集群的原因。在本章第二节介绍过，将集群分割为多个子集群是为了避免多 Monitor 共同管理存储集群可能带来的混乱。但是，分割成多个子集群之后 Client 如何接入系统就会是一个问题，在 DOBBS 中我们对于全局热均衡有两个方面的考虑，一是静态的全局热均衡，二是动态的全局热均衡。

静态的全局热均衡就是 Client 在初次接入 DOBBS 系统时，Center 会分配一个子集群给这个 Client，Client 在之后与系统的交互中就是和这个子集群的 Monitor 和底层存储之间进行。Center 的分配规则实际就是一种静态的负载均衡，Center 会维护一个所有子集群热度的列表，它每次会把会“冷”的子集群分配给 Client。通过这种方式，整个集群可以达到静态的负载均衡，但是随着每个 VM 上应用的不断变化，这种短暂的负载均衡终将打破。所以，DOBBS 在静态热均衡的基础上又加入了动态热均衡的设计。

假设这样一种情况，在一段时间内一个 VM 产生了极大量的 VMDI 请求并向某个子集群的 Monitor 汇报了大量的对象数据流信息，那么这样就会导致某一个子集群突然承受非常大量的 IO 请求，而这个子集群的 Monitor 也需要处理大量的数据流信息。就如上一节所说的，Monitor 会保存一个有序队列，那么大量的数据信息将会导致 Monitor 频繁更新有序队列。这种情况会大大降低子集群上接入的其他 Client 的运行效率。为了解决这个问题，DOBBS 采用动态热均衡的策略来动态监测每个子集群的热度信息，然后做出决策找到最热的子集群再将热度扩散给其他子集群。

3.4.2 热度不均衡监测

热度不均衡出现在某个子集群在短时间内突然承受大量的 VMDI 请求，而如何刻画这一情况是全局热均衡关注的一个问题。我们观察发现子集群的热度主要有两个方面来刻画，一是存储集群的访问次数，二是 Monitor 的性能压力。因为 Ceph 的对象大小是恒定的，所以只考虑存储集群的访问次数是合理的。Monitor 上的 Reporter 模块是负责收集该子集群的热度信息。我们用单位时间内 IO 请求数来表示存储集群的热度，更加细致地来说，在 DOBBS 中我们用每秒 IOPS（IO operations Per Second）来表示存储集群的热度值。所以 Reporter 会调用内部接口实时监测存储集群所有 OSD 上的 IOPS 值，并进行汇总。而 Monitor 的热度我们用 Monitor 的 CPU 和内存利用率来表示，同样 Reporter 调用 Linux 系统调用来实时获取 Monitor 所在机器的 CPU 和内存利用率。在获得了存储集群的 IOPS 和 Monitor 的使用率之后，Repoter 将两者归一化之后组合起来，作为子集群的热度值。下面三个等式描述了子集群热度值的计算方式:

$$StorageHeat = \frac{\alpha \sum^{n} SIO + \beta \sum^{m} HIO}{m+n} \tag{3–1}$$

$$MonitorHeat = max(MemoryUsage, CPUUsage) \tag{3–2}$$

$$SubclusterHeat = StorageHeat + MonitorHeat \tag{3–3}$$

等式3–1表示存储集群的热度，其中 SIO 和 HIO 分别表示 SSD 和 HDD 的 IOPS 值，m 和 n 分别表示了子集群的存储集群内部 SSD OSD 的数量和 HDD OSD 的数量。α

和 β 是两个经验常数，在局部热均衡的过程中，我们已经将过热的对象移动到 SSD 上而冷的对象放置于 HDD 上了，所以如果直接来看 SSD 和 HDD 对 IOPS 的贡献并不准确。α 和 β 起到了归一化的作用，他们是经过大量实验调参得到的数值。等式3–2表示的是 Monitor 的热度，这里的 CPU 利用率和内存利用率都是用百分数表示的，之所以用二者的最大值是因为 Monitor 在处理大量数据流信息的时候需要存储在内存中，而计算是周期性的，所以用二者的最大值来表示 Monitor 的热度值。等式3–3则是将存储集群热度和 Monitor 热度相加来表示子集群的热度值。Repoter 定期计算所在子集群的热度信息，然后向 Center 节点汇报。

在 Center 节点上，它会保存所有子集群最新的热度值，在每次所有子集群汇报之后都会更新热度值。因此，Center 节点总是保存最新的子集群的热度值。为了体现子集群热度的不均衡情况，我们用所有子集群热度的标准差来刻画子集群热度的不平衡，那么标准差越大也就说明子集群的热度越不平衡。公式3–4描述了子集群热度标准差的计算方法，其中 $SubNum$ 表示子集群数量，H_i 表示第 i 个子集群的热度值，而 μ 则表示所有子集群热度值的均值。

$$heatSD = \sqrt{\frac{1}{SubNum}\sum_{i=0}^{SubNum}(H_i - \mu)^2} \tag{3–4}$$

当然，仅仅根据子集群间的热度不均衡来触发全局热均衡必然是不充分的，我们还通过一个阈值来表示是否有子集群过热，如果出现一个子集群的热度值超过阈值，则会触发全局热均衡。我们通过算法3–1来判断是否需要启动全局热均衡，并确定应该从哪个子集群开始做全局热均衡。因为集群的大小、接入系统的 VM 所运行的应用都不尽相同，所以我们让用户可以配置热度阈值和标准差阈值。

算法3–1第 1-3 行是准备工作，获得热度值最大的子集群编号和热度值，计算实时集群热度值的标准差以及初始化当前热均衡迁移数量。第 4-22 行是算法的主循环，当存在一个子集群热度值大于阈值或者当前标准差大于标准差阈值，则进入主循环进行后续的操作，否则终止算法。第 5 行保证了，整个系统的全局热均衡迁移的数量被控制在一个范围内，这样不会让大量热均衡迁移破坏整个集群的效率，如果已经已经到达了最大数量，那么将重新获得标准差和最大热度值的子集群。第 6 行表示获取当前集群热度最低的子集群，我们将热度最低的子集群作为全局热均衡迁移的对象。第 8-16 行则表示了如果集群中热度值最低的子集群的热度仍大于最大热度阈值，将等待出现比阈值小的子集群，否则在尝试一定次数之后就中断整个监控算法，然后想用户抛出异常，通知用户所有子集群的热度值都超过了阈值，提醒系统管理员可能需要增加硬件。第 13 行的 $sleep()$ 函数是因为所有子集群可能都存在过热的情况，我们等待一段时间后看是否

有所缓解，第 17 行则表示，算法已经寻找到了源子集群和目标子集群，并启动全局热均衡迁移，同时更新当前迁移数量。

可以看出算法3–1的时间复杂度并不高，在实际情况中，Center 会一直执行这个算法，如果遇到算法退出或者终止，它将自动检测问题并重新启动检测算法。从算法可以看出，如果没有子集群的热度到达阈值并且标准差没有达到阈值，将不启动检测程序，但是我们还是需要实时检测各个子集群的运行情况，在 Center 中还会启动一个线程来专门循环执行这个算法，直至算法异常退出。

算法 3–1 子集群热度不均衡检测算法

输入： H_k：子集群 k 的热度值，HT：用户自定义的热度阈值，M：最大热均衡迁移数量，DT：用户自定义的标准差阈值，$SubNum$：子集群数量

1: $H_i \leftarrow getLargestHeatValue()$
2: $currentSD \leftarrow getCurrentStandardDeviation()$
3: $currentMigration \leftarrow 0$
4: **while** $H_i > HT$ or $currentSD > DT$ **do**
5: **if** $currentMigration < M$ **then**
6: $H_j \leftarrow getSmallestHeatValue()$
7: $tryTimes \leftarrow 0$
8: **while** $H_j > HT$ **do**
9: // All sub clusters are overheated
10: **if** $tryTimes > SubNum$ **then**
11: $abortDetection()$
12: **end if**
13: $sleep()$
14: $H_j \leftarrow getSmallestHeatValue()$
15: $tryTimes \leftarrow tryTimes + 1$
16: **end while**
17: $applyMigration(i, j)$
18: $currentMigration \leftarrow currentMigration + 1$
19: **end if**
20: $currentSD \leftarrow getCurrentStandardDeviation()$
21: $H_i \leftarrow getLargestHeatValue()$
22: **end while**

如果我们用一个单独的节点（服务器）来作为 Center，那么就必须保证这个 Center 节点的高可用性。如果 DOBBS 中的 Center 节点因为一些不可预料的因素宕机或是无法连通，那么对于整个系统来说将是致命的。Center 在整个集群中的作用是在 Client 接入系统是向 Client 分配较冷的一个子集群，还有就是收集每个子集群的热度值并做决策后发出热迁移指令。Center 如果宕机或无法响应之后，那么新 Client 将无法接入系统，并且所有子集群可能会处于一个热度极度不均衡的状态。不管是哪个问题，对 DOBBS 来说都是灾难性的。那么，如果用一个单独的计算机作为 Center，它就变成了整个系统的唯一关键节点，一个合适的解决方法就是使用高效的一致性协议，让多个节点维护该节点数据的大量备份，当这个节点宕机之后，根据一致性协议则可以快速切换 Center。如此便保证了整个系统的高可用性。在 DOBBS 中，我们不再使用一个单独的节点作为 Center，而是使用 Raft[27] 这个一致性协议，我们让集群中的某个 Monitor 作为 Raft 中的 leader，同样也是 Center，而其他 Monitor 用作备份节点，一旦这个 leader 宕机了，那么 Raft 将快速选举出一个 Monitor 作为 Center。

本小结都是对全局热均衡的准备工作，包括如何定义子集群热度值和对子集群热度值的检测。

3.4.3 热扩散

上一小节介绍了 DOBBS 全局热均衡的前半部分的工作，本小节将介绍全局热均衡的全局热均衡迁移过程，这也是全局热均衡的重点。全局热均衡迁移过程在本论文中被称作热扩散过程，因为这个过程和物理上的热扩散十分相似，所以我们就借用了热扩散的概念来概括全局热均衡迁移。物理上的热扩散过程指的是，理想情况下，热量会从热的物质向冷的物质扩散，最终两者的热量达到总量上一致。而在 DOBBS 上，我们指的是“热量”会从过热的子集群向一个相对热度值较低的子集群扩散，最终两者的热度值达到总量上的一致。过热的子集群和热度较低的子集群都是由算法3–1得到的，我们分别称它们为源子集群和目标子集群。

热扩散过程的目的是将源子集群过热的数据扩散到目标子集群上。与局部热均衡的迁移过程不同，局部热均衡是在子集群内部的不同种类的 OSD 之间迁移数据对象，而热扩散过程则是跨子集群的迁移整个 OSD 的数据。DOBBS 的策略是，源子集群在接收到 Center 发送过来的迁移请求之后，它会去查询内部的 HeatCollector，根据 HeatCollector 可以知道哪个 OSD 的 IOPS 最高，然后以这个 OSD 作为要迁移的 OSD。目标子集群也会去查询自己 HeatCollector，根据它的 HeatCollector 就可以知道哪个 OSD 的 IOPS 最低，然后将这个 OSD 作为要被迁移的 OSD。热扩散过程不能被认为是迁移的过程，它实际上是两个子集群交换 OSD 的过程。我们在介绍 DOBBS 系统架构的时候提到过，子

集群只是一个逻辑上的概念，而每个子集群之间其实在物理网络层面上是互联互通的，所以互相交换 OSD 是可行的，并且交换 OSD 只是在逻辑层面上交换，所以对整个集群的物理结构并没有任何影响。

大规模的数据传输对系统性能来说是致命的。因为热扩散的过程是将整个 OSD 的数据进行交换，这样会带来两方面的性能上的影响。局部热均衡迁移的过程中，会出现因为数据对象的迁移导致虚拟机不能正确访问数据对象的问题，这是因为对象迁移导致 Client 上的对象信息与存储集群实际存储的对象不一致，在系统实现的过程中我们使用了 Remote Lock 这一技术来解决这问题。Remote Lock 会对 Client 上的数据对象上锁，上锁之后 VM 是不能访问到该对象的。那么，如果是将整个 OSD 迁移，一个 OSD 的大小大致是 120GB，而一个 Ceph 对象的默认大小是 64MB。如果将整个 OSD 迁移的话，因为 Remote Lock 的影响，迁移过程会导致 Client 上的对象被频发上锁、解锁，于是 VM 会被暂停运行。在局部热均衡中，一次只是对一个对象迁移，所以上锁带来的影响微乎其微，但是大量对象被上锁解锁，那么对于 VM 是灾难性的。而如果 VM 上运行的实时应用程序，那更是难以想象的灾难。还有一个方面的性能问题，两个 SSD 的数据一共有 240GB，那么传输过程也会非常久，而且占用大量集群核心网络的带宽。因此，如果将两个 OSD 的全部对象进行迁移则十分不现实，也严重影响集群效率。

为了解决大规模的数据传输的问题，我们采用了一个全新的懒汉迁移方式——元数据迁移。前面我们讲过直接将整个 OSD 的数据进行迁移是不现实的，那么这种方式可以定义成是一种饿汉迁移方式。借用物理学规律，我们知道热量的传递过程也是相对缓慢的，所以我们选择了一种慢速的迁移方式。元数据迁移，是混合运用 DOBBS 系统架构和局部热均衡的迁移方式，元数据指的是局部热均衡中的 Analyzer 所维护的数据流信息。在局部热均衡中，我们介绍过数据流信息包括每个对象的热度值以及它所在 OSD 的编号等信息。在源子集群收到 Center 的热扩散请求之后，它会先检查 HeatCollector，得到当前 IOPS 最大的 OSD 编号，之后在 Analyzer 中再把属于这个最大 IOPS OSD 所有的对象数据流信息从有序队列中删除，并通过网络传输给目标子集群的 Monitor。在目标子集群收到 Center 发送过来的热扩散请求之后，它会做相似的工作：目标子集群会先检查 HeatCollector，得到当前 IOPS 最小的 OSD 编号，之后再和源子集群的做法相同，最后通过网络把 IOPS 最小 OSD 对应的对象数据流信息发送给源子集群。因为在 DOBBS 中，子集群是一个逻辑概念，所以每个子集群的 Monitor 都会本子集群所有 OSD 编号的列表。在源子集群和目标子集群互换原信息之后，它们也会修改所在子集群的 OSD 信息。

元数据迁移和饿汉模式相比，并没有做物理上的数据迁移，而仅仅从逻辑上改变了两个存储集群的中 OSD 的位置。例如，原来子集群 1 包含第 0、1、3 号 OSD，而子集

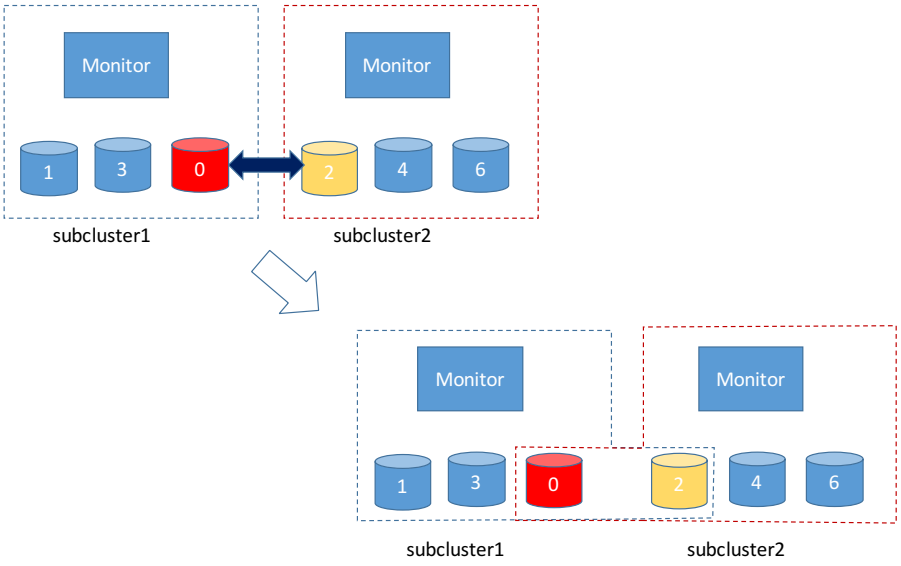

图 3–2 元数据迁移使能过程

Fig 3–2 Enable Process of Metadata Migration

群 2 包含第 2、4、6 号 OSD，现在子集群 1 作为源子集群，子集群 2 作为目标子集群，它们交换了第 0 和第 2 号 OSD，那么现在子集群 1 就包含第 2、1、3 号 OSD，而子集群 2 包含了第 0、4、6 号 OSD。图3–2所示，就是上面所举的例子的示意图。可以从图中看出元数据迁移只是修改了 OSD 的逻辑结构，而并没有在两个 OSD 上互相交换数据。在最终交换完成后，也就完成了元数据迁移的使能过程。

我们将元数据迁移分为两个过程，一个是使能过程，还有一个是懒迁移过程。使能过程是上文所提到的，两个子集群修改自身的逻辑结构，然后两个 Monitor 交换元数据的过程。当然从使能过程的结果来看，其实并没有真正的实现热量的转移，而只是修改了逻辑结构，对问题的本质并没有实质性的影响。我们称元数据迁移是一种混合的方式，懒迁移过程实际有局部热均衡参与的。我们将源子集群的属于 IOPS 最大 OSD 的元数据拷贝到了目标子集群的 Analyzer 中，从目标子集群的 Analyzer 角度看来，就像是从 Client 新汇报出大量数据对象。根据上一节我们寻找源子集群和目标子集群的策略可以知道，目标子集群一定是当前热度值最低的子集群，而源子集群一定是当前热度值最高的子集群，并且我们所交换的 OSD 也是它们彼此最“冷”和最“热”的 OSD。所以，目标子集群所收到的元数据的热度值一定是比其自身所有 OSD 上对象的热度值的最大值都要大的。根据局部热均衡的原理，目标子集群 Analyzer 在收到这些热度值很高的元数据之后，它会优先将这些元数据所对应的数据对象迁移到它的 SSD OSD 上。相

反，在源子集群接收到目标子集群的元数据之后，它会优先将这些元数据所对应的对象进行迁移。这个过程就是，元数据迁移的懒迁移过程，可以看到我们并没有让 Monitor 立即去迁移所有的数据，而是交给了局部热均衡去做迁移的工作，这相当于是一种缓慢的迁移方式。元数据迁移是有效的，因为在修改 OSD 的逻辑结构之后，局部热均衡充当了迁移者，它把过热的 OSD 上的热量辐射到目标子集群的其他 OSD 上去，做到了热量扩散。

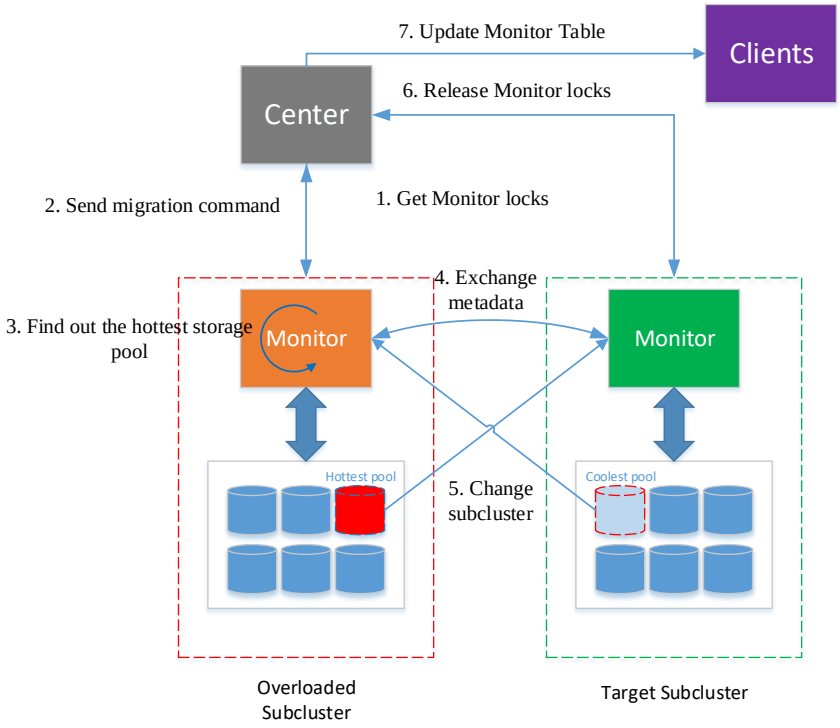

图 3–3 热扩散过程

Fig 3–3 Process of Heat Diffusion

元数据迁移过程的使能过程的迁移对象是对象的数据流信息（元数据），这个数据流信息实际上是局部热均衡用来决策的重要依据。假设有这样一个情况，在局部热均衡的过程中，某个 Monitor 正在执行对象迁移指令，它向 Client 发送了上锁指令，随即 Client 对要被迁移的对象上锁，可是在这次对象迁移过程没有执行完之前，该 Monitor 又收到了元数据迁移请求，它立即响应请求将部分元数据传送到目标子集群。在元数据迁移完成后，OSD 已经互相交换，并且 Client 上的 Monitor Table 也已经被更新。那么，Monitor 则始终不能发送 release-lock 请求对应的 Client，Client 上的这个对象可能会被永久得阻塞，这样 Client 上运行的 VM 也将受到影响。为了解决这个问题，根据 [28] 的

启发，我们提出了一个 try-lock 机制。该机制会首先尝试从源子集群和目标子集群上获得锁，但是如果其中有子集群正在执行局部热均衡的对象迁移，则返回失败，即不能获得锁。当且仅当 Center 获得了源子集群和目标子集群的两个锁之后才算上锁成功，才可以继续元数据迁移。在两个集群上的锁，一旦上锁之后，Monitor 就不能产生对象迁移请求，直到锁释放才可以进行对象迁移。

图3–3所示为 DOBBS 的热扩散过程的示意图。热扩散过程可以总结成 7 个步骤，在 Center 找到需要被扩散的源子集群和目标子集群之后，它会首先通过 try-lock 机制向两个子集群的 Monitor 获得锁。只有当同时过得两把锁之后，Center 执行第二步也就是向源子集群和目标子集群发送元数据迁移指令。两个子集群的 Monitor 接受到指令之后，它们查询各自的 HeatCollector，然后将元数据进行交换，再修改两个子集群的存储逻辑结构，两个 Monitor 更新各自的 OSD 列表。在这些过程结束之后，Center 释放两个子集群的锁，并更新 Client 上的 Monitor Table。一旦锁被释放，两个 Monitor 就开始进行局部热均衡，也就是开始懒迁移过程。

全局热均衡的热扩散过程实际也是软件定义的，在传统的数据中心一旦出现某个集群过热，则需要系统管理员手动地调度来进行负载均衡。而全局热均衡则是根据每个子集群的热度本身来重新修改各个子集群的逻辑结构，从而使得整个系统达到热均衡的状态。在全局热均衡结束之后，Center 也会通知所有的 Client 修改存储集群的逻辑结构，这样保证了全局热均衡的整个过程对上层应用是透明的。

3.5 本章小结

本章主要从 DOBBS 系统的架构、局部热均衡、全局热均衡等三个方面介绍了系统的设计。DOBBS 的架构是采用了一种两层的架构方式，整个系统主要有 4 中类型的节点，其中 Client 节点是搭载虚拟机的节点，Monitor 是每个子集群的监视器，Center 则负责全局热均衡，最后就是 OSD 节点。OSD 根据存储介质被分成了 SSD OSD 和 HDD OSD。DOBBS 还将 Monitor 和混合存储池合并成一个逻辑结构——子集群。局部热均衡是为了充分利用 SSD 的优势，并以一个经济的方式实现混合存储，我们通过动态监测对象的数据流信息计算对象热度来迁移对象到合适的 OSD 上。全局热均衡则是因为我们采用分布式多节点 Monitor 之后产生的数据访问不均衡现象，所以要通过一种动态的方式来进行负载均衡，也就是热量的传递。

DOBBS 是软件定义的存储系统，这主要体现在两个方面：局部热均衡将子集群内部的控制面和存储面进行了分离，我们动态监测虚拟机对象的数据流信息，并且根据既定的策略（policy），即热数据放置于 SSD 而冷数据放置于 HDD，对数据流进行分析，在不同的存储介质间迁移对象，整个这些过程是系统自动完成的，而不需要系统管理员

专门地指定存储介质；DOBBS 是双层结构，全局热均衡则是被用来处理上层（子集群层）的热度均衡，Center 被用来检测子集群的热度信息，并发起热扩散请求，热扩散最终会让系统的逻辑结构发生改变，而这些过程对上层 VM 的应用来说都是透明的。

第四章 系统实现

本章在上一章的基础上，介绍 DOBBS 在实现过程中所用到技术以及各个模块的具体实现情况。

4.1 实验工具和平台

4.1.1 Ceph

Ceph 是一个免费开源的存储平台，它将对象存储实现在一个分布式计算机集群内，并且为对象级、文件级和块级存储提供了接口。Ceph 主要针对完全分布式操作，没有单点故障，可以扩展到 exabyte 级别，并且有非常高的可用性。Ceph 的软件库为客户端应用程序提供了对 RADOS 基于对象的存储系统的直接访问，同时提供一些高级功能，包括 RADOS Block Device（RBD），RADOS Gateway（RGW）和 Ceph File System（Ceph FS）。

Ceph 的对象存储系统允许用户将 Ceph 安装为精简配置的块设备。当应用程序使用块设备将数据写入 Ceph 时，Ceph 会自动在整个群集中对数据进行分条和复制。Ceph 的 RADOS Block Device（RBD）还集成了基于内核的虚拟机（KVM）Ceph 的块设备可以被虚拟化，为虚拟机提供块设备服务在一些主流的虚拟化平台都使用 Ceph，例如 Apache CloudStack、Openstack 和 OpenNebula[29] 等等。

DOBBS 在 Ceph 的基础上进行修改是因为，Ceph 是对象存储系统，当虚拟机在 Ceph 存储集群创建 VMDI 之后，Ceph 会默认地把它分割成等大小的对象，而 DOBBS 需要动态监测虚拟机 VMDI 的数据流，那么如果 VMDI 已经被等大小分割那么我们就可以直接地监测 Ceph 对象而不是整个 VMDI 数据块，这样把监测和迁移的粒度放到比较小，帮助我们快速准确的迁移数据。还有一点就是，Ceph 与虚拟化的结合很充分，所以不需要我们特别地为 Ceph 进行额外的修改。

4.1.2 QEMU

QEMU[30] 是快速模拟器（Quick Emulator）的缩写，它是一个免费的开源托管型虚拟机管理程序，可执行硬件虚拟化。QEMU 是托管的虚拟机监视器：它通过动态二进制转换模拟 CPU，并提供一组设备模型，使其能够运行各种未修改的客户机操作系统。它也可以与 KVM[31] 一起使用，以接近本机的速度运行虚拟机。QEMU 还可以为用户

级进程执行 CPU 仿真，允许为一个体系结构编译的应用程序在另一个体系结构上运行。QEMU 的优势在于，它足够轻量级并且已经被合并入 Linux 内核。而在本论文中选用 QEMU 作为虚拟机管理程序主要有两个原因，一是因为 QEMU 是开源软件我们可以直接对其源代码进行侵入式修改，二是 QEMU 已经有了针对于 Ceph 的虚拟块存储服务。

4.1.3 Apache Thrift

Thrift 是一个 RPC（Remote Procedure Call）框架。Thrift 是一种接口定义语言和二进制通信协议，用于定义和创建多种语言的服务。它被用作远程过程调用（RPC）框架，并在 Facebook 上为“可扩展的跨语言服务开发”而开发。Thrift 的使用非常方便，用户只需要写一个自定义语言的文件用于定义接口和部分数据结构，之后主要输入 Thrift 的指令便可以生成对应语言的服务器/客户端文件。在 DOBBS 的实现中，我们用 Thrift 作为跨服务器远程过程调用的工具，我们只需要编写 Thrift 自己定义语言的接口语言，并可以在各个服务器上使用，十分便捷。

代码 4.1 Thrift 接口定义示意

```
struct ObjInfo {
  1:string m_eid,
  2:i32 m_osd,
  3:double m_rio,
  4:double m_wio,
}
struct MonitorMetadata{
    1:string m_eid;
    2:i32 m_osd;
    3:double m_weight;
    4:string m_ip;
}

service MonitorService {
    void finish_lock(1:string eid),
    void report_client_info(1:ObjInfo ci),
    void finish_migration(1:string eid),
    void begin_heat_diffusion(1:string to_ip),
    list<MonitorMetadata> exchange_metadata(1:i32 pool_id, 2:list<
    MonitorMetadata> monitor_extents),
    bool get_monitor_lock(),
    void release_monitor_lock(),
}
```

如代码4.1所示为 Thrift 接口定义代码，该代码表示 DOBBS 的 Monitor 模块的对外提供的接口。首先需要定义用于传输的数据结构，在 Monitor 中，我们需要收集从 Client 发送的对象数据流信息，所以接口定义中我们用 ObjInf 来表示。service 代码块表示具体接口的接口名以及参数和返回值等等。在编写完接口定义文件之后，用户需要调用 Thrift 程序即可生成指定语言的代码，然后用户将代码拷贝到需要调用接口的计算机上，即可完成 RPC 传输。

4.2 系统模块实现

如图4–1是系统模块结构图。从图中可以看到，DOBBS 主要有四个组件，分别是客户端（Client）、监控器（Monitor）、OSD 和中心控制器（Center）。而我们的各个模块分布在了这些组件上。在本章系统实现部分我们不再用模块的概念来表述，图4–1的类则对应了上一章系统架构图中的各个模块。为了直观显示 DOBBS 各个组件之间的关系，我们将 Center 独立出来作为一个组件描述，而实际场景中，Center 是位于某个 Monitor 服务器上运行的。值得注意的是，图中的 Ceph OSD Component 并不是 DOBBS 中的组件而是 Ceph 系统所提供的访问数据对象的接口。

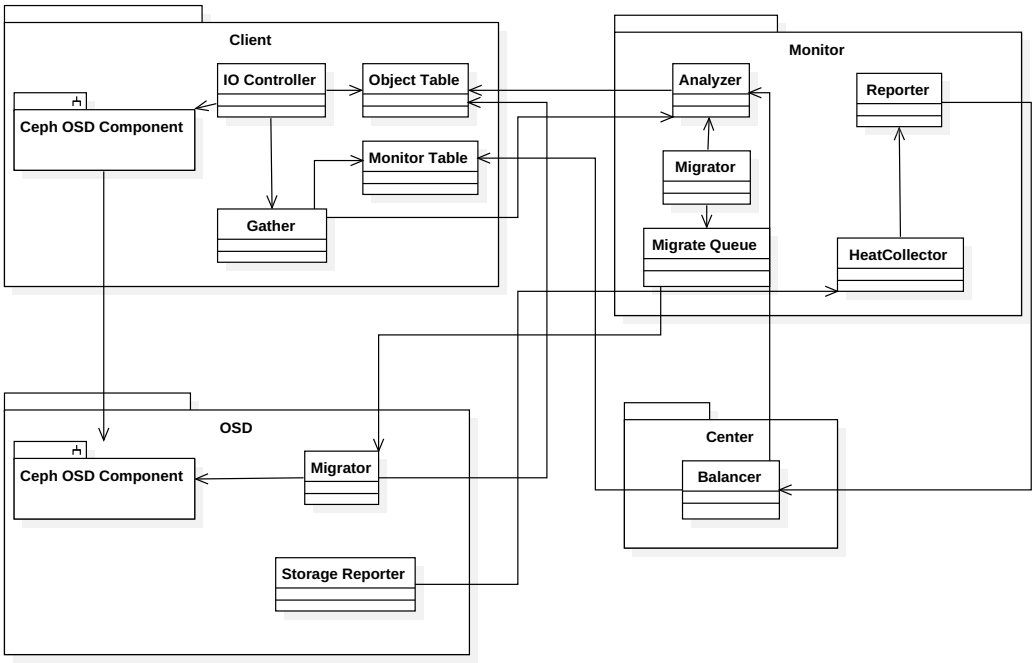

图 4–1 DOBBS 模块结构

Fig 4–1 Module Structure of DOBBS

Client 主要包含四个类，这四个类也是与系统设计中的模块一一对应的。IO Controller 用来截取 VM 的数据请求，并通过查询 Object Table 的方式来获取对象所在的 OSD 编号，再把对象 ID 和所在 OSD 编号共同传递给 Ceph OSD Component 进行数据访问。IO Controller 还有就是需要调用 Gather 的接口来记录 VM 访问对象的访问类型和 ID，因此 IO Controller 没有暴露给外部的接口，它则主要对 Ceph 和 QEMU 源代码的修改。Object Table 的主要作用是保存一个对象 ID 到 OSD ID 映射的数据结构，并暴露出可以让 IO Controller 进行查询的接口，另外在局部热均衡对象迁移结束之后 OSD 上的 Migrator 也需要调用 Object Table 提供的更新映射的接口。Gather 则用于收集对象的访问信息，并调用 Monitor 上的接口将单位时间的数据流信息汇报给 Monitor 的 Analyzer。而 Monitor Table 是用于存储 Monitor ID 与 OSD ID 的数据结构，因为全局热均衡会改变系统的逻辑结构，所以在全局热均衡结束之后会去更新 Monitor Table 提供的更新映射的接口。Gather 则用于收集对象的访问信息，并调用 Monitor 上的接口将数据汇报给 Monitor ，Gather 在汇报给 Monitor 的之前需要先通过对象的 OSD 查询 Monitor Table 得到到该 OSD 在哪个子集群，然后在将相同子集群的对象数据流信息整合起来打包发送给对象的 Monitor。

Monitor 则包含五个类。Analyzer 是用于接收 Client 的 Gather 所汇报的数据流信息，并且它内部会持续计算对象热度，在生成对象迁移策略之后调用 Migrator 发送迁移请求，Analyzer 的另一个重要功能就是接受 Center 节点的命令开始热扩散的使能过程。Migrator 则主要负责对象的迁移请求发送等。Migrator Queue 的主体是一个队列，用于存储迁移请求，另外 Migrator Queue 还会额外记录子集群当前迁移数量。Heat Collector 是在全局热均衡中所用的类，它的主要作用是调用操作系统指令获得 Monitor 的 CPU 和内存利用率，还有就是调用所在子集群 OSD 上的 Storage Reporter 接口获取存储设备的 IOPS。Reorter 则是会调取 HeatCollector 提供的接口获得实时子集群热度值，并定时将这个值发送给 Center 的 Balance。Center 上的 Balancer 的主要功能就是接受 Monitor 发送的子集群热度值，然后运行算法3–1生成热扩散请求，在热扩散结束之后再修改所有 Client 上的 Monitor Table。

4.2.1 Monitor 的模块实现

图4–2所示是 Monitor 模块的设计类图。从图中可以看到 Monitor 一共有五个主要的类，其中的 Analyzer、Migrator 和 Migrator Queue 是继承于 WHOBBS 的实现[1]。在本小结中，我们只做简单的介绍，但是我们扩展了原有 Analyzer 类的功能，如全局热均衡的使能过程。

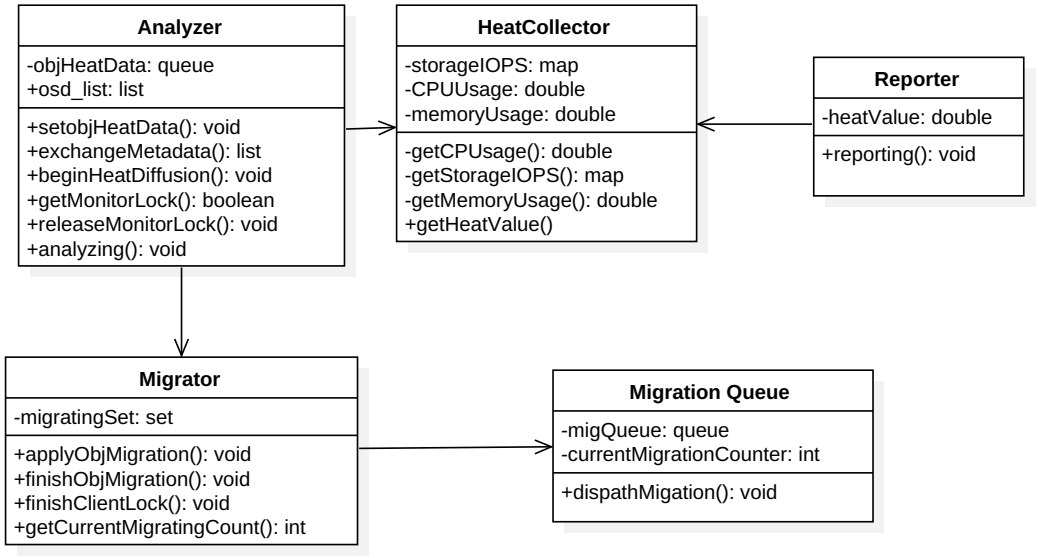

图 4–2 监视器模块类图

Fig 4–2 Class Diagram of Monitor

4.2.1.1 Analyzer

Analyzer 的主要功能就是在局部热均衡的时候保存数据对象的热度信息，然后不断分析所有数据对象，计算对象的热度，并产生对象迁移请求。objHeatData 是一个有序队列，我们实现了一个叫做 ObjectQueue<typename T> 的内置类，它内部是通过链表实现的队列结构，只不过在链表插入的时候是根据元素的热度值进行的降序排序，保证了对头的元素的热度值总是最大的。objHeatData 实际上的类型是 ObjectQueue<ObjInfo>，ObjInfo 是我们用来表示对象热度信息的结构体，这个结构体包含对象 ID（oid）、所在 OSD 编号（osd_id）、所属 Client IP（client_ip）和热度（heat_val）。因此 objHeatData 是用于存储对象热度信息的，它通过 heat_val 降序排序。Analyzer 还有个数据结构是 osd_list，它的类型是 std::list<int>，它用来存储当前子集群所包含的 OSD 编号。

analyzing() 方法是一个线程入口函数，Analyzer 在被实例化之后就会启动分析线程。分析线程工作就是不断遍历 objHeatData，产生对象迁移请求。根据上一章对 Monitor 的设计，analyzing 遍历时候的策略就是将会维护一个当前 SSD 剩余容量的计数，在 SSD 的容量没有满的时候，它会把 objHeatData 的前若干个对象放置于 SSD 上。然后在该线程内部也会有两个集合，分别表示当前在 SSD 中的对象和在 HDD 中的对象。通过前面的策略，很快 SSD 就会被填满。一旦 SSD 的容量已满，该线程则会遍历 objHeatData 的前 N 个对象，N 的大小等于 SSD 容量/单个对象大小，找到这前 N 个对象中哪些是在 SDD 上哪些是在 HDD 上。如果出现 HDD 上的对象位于前 N 的位置，则将该对象从

HDD 迁移到 SSD 上。如果发现 SSD 集合中的对象不在前 N 的位置里，则将该对象迁出 SSD[1]。该分析线程发送迁移指令是通过调用 Migrator 的 applyObjMiration() 的接口来实现的。setObjData() 接口的实现相对简单，它只是用来在 Monitor 接收到 Client 汇报来的数据流对象，通过热度计算算法计算出每个对象的热度之后，如果 objHeatData 中含有这个对象则更新它的热度值，否则实例化一个 ObjInfo 对象，将它插入到 objHeaData 中。值得注意的是，objHeatData 的插入过程我们用到的是插入排序算法，从而保证每次插入都是有序的。

beginHeatDiffusion()、getMonitorLock()、releaseMonitorLock() 和 exchangeMetadata() 这四个接口是 Center 所调用的。getMonitorLock() 是用于对局部热均衡上锁，它会先去调用 Migrator 的 getCurrentMigratingCount() 接口获取现在子集群是否还有正在迁移的对象，如果接口返回值非 0，则不能上锁，接口返回 false。如果 getCurrentMigratingCount() 的返回值是 0，则会对 analyzing() 上锁并返回 true，上锁我们是通过 Linux 的互斥锁 pthread_mutex_t 来实现的。beginHeatDiffion() 接口是 Center 检测到子集群热度不均衡之后，并对 Monitor 上锁之后执行的接口。这个接口就是全局热均衡使能过程的开始。而 releaseMonitorLock() 接口是 Center 在全局热均衡使能过程之后用于释放 Monitor 锁的，它调用 pthread_mutex_unlock() 这个 Linux 线程函数来解除加载 analyzing 线程上的互斥锁。这个互斥锁是用在 Analyzer 的 objHeatData 这个数据结构上的，在上锁之后其他线程就不能对 objHeatData 进行读写操作，从而局部热均衡将被暂停。

beginHeatDiffusion() 接口传入的参数是目标子集群 Monitor 的 IP 地址。在接口开始调用的时候，它会调用 HeatCollector 的 getStorageIOPS() 接口，这个接口会返回子集群的存储集群各个 OSD 的编号和 IOPS 的映射。在获得这个映射之后，它找到最大 IOPS 的 OSD 编号。然后它会去 objHeatData 中去遍历，找到所有 osd_id 为 IOPS 最大 OSD 的对象，将它们从队列中剔除并置于一个传输 buffer 中。这个 buffer 实际上类型为 std::list<ObjInfo> 的对象。封装完成后，它通过 Thrift 调用目标子集群 Monitor 的 exchangeMetadata() 接口以参数的方式将 buffer 中的元数据传输给目标子集群。目标子集群会从接口 exchangeMetadata() 返回其最“冷”OSD 所对应的元数据，返回值类型为 std::list<ObjInfo>，然后将源子集群发送来的元数据调用 setobjHeatData() 接口插入 objHeatData 中。beginHeatDiffusion() 在接受到目标子集群的返回值之后，它把列表中的数据插入其自身的 objHeatData，在接口执行的最后它会更新 osd_list 中值，最后它调用 Center 的 finishDiffusion() 方法，通知 Center 热扩散的使能过程已经完成。exchangeMatadata() 接口则是在目标子集群的 Monitor 上被调用的接口，它的参数是源子集群最“热”OSD 对应的元数据。该接口被调用之后，它会通过查询 HeatCollector，找到 IOPS 最低的 OSD 编号，并将它所对应的元数据从 onjHeatData 中剔除，并通过返回值的方式传送

给源子集群。

4.2.1.2 Migrator 和 Migrator Queue

Migrator 和 Migrator Queue 的实现与 WHOBBS 中的实现相似, 我们这里不再详细叙述这两个模块的实现。Migrator 主要向 Analyzer 提供 applyObjMigration() 接口，Analyzer 调用该接口后，该接口的传入参数是一个表示对象迁移的数据结构，它包括被迁移对象的原 OSD 编号、目的 OSD 编号和对象 ID, ,然后它会调用 Client 上的对象上锁接口,在上锁成功后将迁移请求 (表示迁移的数据结构) 放置于 Migrate Queue。finishObjMigrating() 接口是在 OSD 结束对象传输之后调用的，调用之后会把这个对象从 migratingSet 中剔除，然后调用 Client 上的释放锁的接口[1]。与 WHOBBS 不同的是，为了支持 Monitor 锁，我们增加了 getCurrentMigratingCount() 接口，这接口在调用之后会直接返回当前 migratingSet 的大小。migratingSet 是在发送请求之后将 oid 放入其中，直到迁移完成后才会把它从 migratingSet 删除掉，因此 migratingSet 表示的是当前正在迁移对象的 ID。Migrator Queue 的实现与 WHOBBS 无异，本文则不再叙述。

4.2.1.3 HeatCollector

HeatCollector 的功能是获取子集群的所有事实资源信息的。它主要有三个成员变量，分别是 storageIOPS（存储集群 IOPS）、CPUUsage（Monitor 的 CPU 利用率）和 memoryUsage（Monitor 的内存利用率）。其中 storageIOPS 的数据类型是 std::map<int, long>，这个 map 数据结构的键为 osd_id，值为对应 OSD 的 IOPS 值。CPUUsage 和 memoryUsage 的类型都是 double，即占用百分比。

getHeatValue() 是 HeatCollector 对外提供的接口, 该接口被调用之后, 它会依次调用 getStorageIOPS()、getCPUUsage() 和 getMemoryUsage() 接口获得当前实时的子集群资源信息，并更新它的三个成员变量。对于 getStorageIOPS()，它会从在 Analyzer 的 osd_list 获取子集群所包含 OSD 的编号，然后再调用本子集群的 OSD 上的 getIOPS() 接口，最后将最新的存储集群 IOPS 值以 map 的方式返回。getCPUUsage() 接口则是调用 Linux 的系统调用查看/proc/stat 文件来获取当前 CPU 利用率。getMemoryUsage() 则调用 sysinfo() 系统调用来获得当前的内存利用率的。在 getHeatValue() 成功调用这些接口之后，它会根据公式3–1计算当前实时的热度值并返回给调用者。

4.2.1.4 Reporter

Reporter 的功能则是汇总子集群的资源利用信息，并定时发送给 Center。Reporter 包含一个成员变量，就是当前子集群的实时热度是 heatValue，它的类型是 double。re-

porting() 函数是一个线程入口函数，该线程的主要功能就是不断的调用 HeatCollector 的 getHeatValue() 接口，在每次获取之后都会调用 Center 的 reportFromMonitor() 接口把热度值传输给 Center。

4.2.2 Center 模块实现

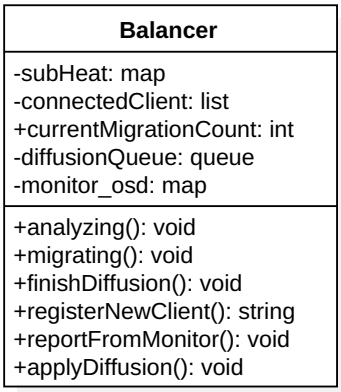

图 4–3 中心模块类图

Fig 4–3 Class Diagram of Center

如图4–3所示为 DOBBS 的 Center 模块的类图，从图中可以看到 Center 只有一个叫做 Balancer 的类。Balancer 类就实现了 Center 的所有功能，包括对各个子集群热度信息的收集、产生热迁移请求和新 Client 接入时的分配。

4.2.2.1 Balancer

Balancer 主要有 subHeat、connectedClient、currentDiffusionCount、diffusionQueue 和 monitor_osd 等五个成员变量。subHeat 的数据类型的 std::<std::string, double>，它表示各个子集群的热度值，其中的键为子集群 id，值为热度值。connectedClient，它的类型是 std::list<std::string> 表示当前接入系统的 Client 的 IP 地址。currentDiffusionCount 表示当前正在全局热均衡的请求数量，它的类型是 int。diffusionQueue 表示准备进行热扩散的队列，它的类型是 std::queue<DiffusionDetail>，DiffusionDetail 是一个用来表示热扩散请求的结构体，它包含 to_sub 和 from_sub，即源子集群和目标子集群的编号。

Balancer 的 analyzing() 函数是一个线程入口函数，这个线程主要就是用来运行子集群热度不均衡检测算法3–1，当算法找到源子集群和目的子集群之后会把原子群和目标子集群标号通过参数的方式传递给接口 applyDiffusion()。而 analyzing 线程会过一定时

间间隔执行一次热度不均衡检测算法。reportFromMonitor() 接口则是用于持续接受，各个 Monitor 所汇报的热度值信息。

applyDiffusion() 算法在接收到源子集群编号和目标子集群编号之后，会生成一个 DiffusionDetail 类型的结构体，该结构体包含 to_sub（目标子集群）、from_sub（源子集群）这两个信息，并将其放置于 diffusionQueue 中。migrating() 函数则是迁移线程的入口函数，它的工作就是不断轮训 diffusionQueue 的内容，只要队列不为空它会先检查 currentDiffusionCount 的值是否达到系统最大热扩散数量，如果已经达到则不对队列做任何处理。如果 currentDiffusionCount 小于系统最大热扩散数量，那么它会通过子集群编号调用两个子集群的 getMonitorLock() 接口，只有当两个都返回 true 时才将这个热扩散请求从 diffusionQueue 删除。在这之后，它会调用源子集群的 beginHeatDiffsion() 接口，并将目标子集群标号传递给它。

finishDiffusion() 接口是在源子集群和目标子集群结束全局热均衡的使能过程之后调用的，最终是由源子集群调用。在调用之后，Center 会首先更新 monitor_osd 的值，因为全局热均衡已经修改了子集群的逻辑结构。然后，Center 再去更新通过 connectedClient 列表去更新所有连接的 Client 的 MonitorTable。最后则是调用目标子集群和源子集群的 releaseMonitorLock() 接口对两个 Monitor 解锁。

在上一章已经讲过，当 Client 初次接入系统时，Center 会分配一个相对较“冷”的子集群给这个 Client。在 Client 初次接入时，它会调用 registerNewClient() 接口，再把自己的 IP 地址通过传参的方式发送给 Center。这个接口会遍历 subHet，找到一个热度值最低的子集群，然后将这个子集群 Monitor 的 id 返回给 Client，最后 Center 把这个 Client 的 IP 置于 connectedClient 中。

4.2.3 Client 模块实现

如图4–4为 Client 模块的类图。Client 模块是运行虚拟机和虚拟机管理器的模块，该模块的 IO Controller、Object Table 和 Gather 这三个类的实现与 WHOBBS 基本一致，所以本文不再过多赘述。但是，我们在原有实现的基础上加入了 Monitor Table 类，用于负责全局热均衡后系统的 OSD 和 Monitor 之间的变化。

因为 Client 的设计，我们要截取 VM 对对象访问的请求，这就需要我们对 Ceph 的原生代码做出修改。我们主要修改了 Ceph librbd 模块的代码，在 Ceph 调用底层对象传输之前，我们让它先调用 IO Controller 的 findOSD() 接口。IO Controller 的 findOSD() 接口是以对象 ID 作为参数，然后在 Object Table 中调用 getObjOSD() 接口进行查询对象当前所对应的 OSD 编号，然后返回给 Ceph 之后，再进行后续的对象访问。值得注意的是，因为 DOBBS 的多 Monitor 设计，以及静态全局热均衡，所以我们需要在 Client 初

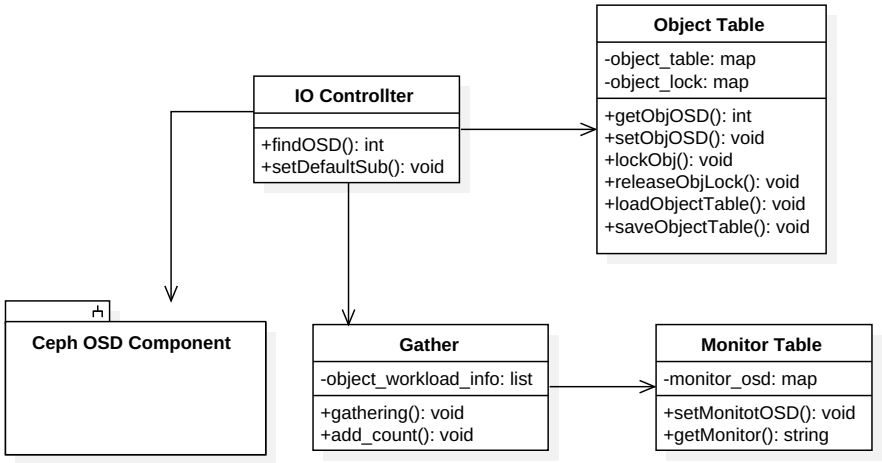

图 4–4 客户端模块类图

Fig 4–4 Class Diagram of Client

次接入系统的时候可以获得它所在的默认子集群。因此，与 WHOBBS 不同，我们在 IO Controller 加入了一个接口 setDefaultSub() 这个是用于 Center 设置当前 Client 默认子集群的接口。当然，为了在 Client 第一次接入 DOBBS 才会去向 Center 获得默认子集群编号，我们也对 Ceph 的源代码进行了修改。

Object Table 的实现与 WHOBBS 没有区别，本文仅介绍各个接口的作用，具体细节不再介绍。首先，该类包含两个成员变量，分别是 object_table 和 object_lock。它们都是 std::map 的数据结构，object_table 是用来保存对象 ID 和 OSD 编号的映射关系，而 object_lock 并不保存信息，只是保存哪些对象上了锁。而 getObjOSD() 和 setObjOSD() 则是对 object_table 进行操作，分别是通过对象 ID 获取对应的 OSD 编号，还有就是修改 object_table 的内容。lockObj() 和 releaseObj() 这两个接口都是在局部热均衡过程中保证一致性对对象上锁和解锁所用到的接口。loadObjTable() 和 saveObjTable() 则是用于在 DOBBS 系统启动和关闭后将 object_table 存盘和读取的接口。

我们在 WHOBBS 原有 Gather 类的基础上进行了扩展。gather() 是线程入口函数，它的功能就是定时向 Monitor 发送对象的数据流信息。我们定义对象的数据流信息包括，对象 ID（oid）、OSD ID(sid)、访问类型（access_type）、访问次数（access_count）。与 WHOBBS 不同，在 WHOBBS 中只有一个 Monitor 存在，所以所有 Client 都只需要向同一个 Monitor 发送自己的数据流信息。对于 DOBBS，我们的 Monitor 会有多个，而且由于全局热均衡，OSD 也会随之发生变化。在 DOBBS 的设计中，OSD 是与子集群绑定的，所以我们需要保证 Client 可以实时知道每个 OSD 都在哪个子集群中。Monitor Table 就是用来保存子集群和 OSD 的信息。gather 线程在每次向 Monitor 发送之前，都会通过

Monitor Table 查询到当前对象所在的 OSD 位于哪个子集群，然后将相同子集群的对象数据流信息组合成一个列表共同发送给对应的 Monitor。

Monitor Table 类的主要功能，首先是保存当前最新的 OSD 和子集群的信息，其次就是接受来自 Center 的更新，还有就是 Gather 会来查询。它有一个叫做 monitor_osd 的成员变量，它的类型是 std::map<std::string, std::string>，它存储了 OSD ID 到子集群 Mointor 的 IP 地址之间的映射关系。setMonitorOSD() 接口是由 Center 远程调用的，在全局热均衡的使能过程之后 Center 会通知所有连接到系统的 Client 更新各自的 Monitor Table。getMonitor() 接口则是开放给 Gather 用于查询对象所在子集群的，它传入的参数是 OSD ID，返回值为 Monitor 的 IP 地址。

4.2.4 OSD 模块实现

对于 OSD 模块的实现，我们完全沿用了 WHOBBS 的实现方案没有做更多的扩展，所以本小节只做简要的概括。可以从图4–1中看到，OSD 上的 Migrator 负责接收来自于 Monitor 的迁移请求，在接收到之后，它只需要调用 Ceph 的 OSD 组件即可完成对象的迁移。而 Storage Reporter 则是调用了操作系统指令获得当前磁盘（SSD/HDD）的 IOPS 值。Monitor 上的 Heat Collector 类会远程调用 Storage Reporter 的接口获得当前的 IOPS 值。

4.3 系统主要工作流程

本小结主要介绍 DOBBS 的主要工作流程。局部热均衡中的工作流程，如对象的迁移、对象数据流信息的获取以及虚拟机的 IO 请求这些工作流程与 WHOBBS 中的实现并没有太大的区别，所以本小节对局部热均衡的工作流程只是做一个高度的概括，而不做过多的描述。全局热均衡是一个复杂的过程，我们将它抽出三个比较主要的工作流程来进行介绍，分别是客户端接入工作流程、监控器获得并汇报子集群热度信息工作流程和全局热均衡使能过程工作流程。客户端接入过程，是静态的全局热均衡的保证并且还是让客户端可以顺利与系统交互的重要保证。监控器获得并汇报子集群热度信息的过程也就是 Center 获得各个子集群热度的过程，这对 Center 进行子集群热度不均衡检测尤为重要。全局热均衡使能过程是全局热均衡最重要的一环，本小结从 Center 发出迁移指令开始来叙述全局热均衡的使能过程。

4.3.1 局部热均衡工作流程

局部热均衡的工作流流程主要可以分为以下几个部分：虚拟机 IO 请求工作流、虚拟机汇报对象数据流信息工作流以及对象迁移工作流。虚拟机 IO 请求工作流一般是由虚拟机发起的，然后 IO Controller 先去查询 Object Table，找到该对象对应的 OSD ID，然后将 OSD ID 返回给 IO Controller，最后 IO Controller 将对象 ID 和 OSD ID 以参数的方式调用 Ceph OSD Component 接口，来进行后续的访问。在这个过程中，IO Controller 起到的是截取请求的作用。值得注意的是，因为我们要获得对象的访问信息，所有在 IO Controller 的 findDefaultOSD() 被调用的时候，它还会调用 Gather 的 add_count() 接口，把当前虚拟机所访问的对象 ID 和访问类型传递给 Gather。

虚拟机汇报对象数据流信息工作流则相对简单，Gather 的 gathering 线程会定时将储存的对象数据流信息发送给 Monitor 的 Analyzer，而每次发送之后它都会清空之前的记录。对象迁移工作流则是局部热均衡的重点。Analyzer 在得出迁移策略之后，它会调用 Migrator，Migrator 生成迁移请求，然后 Migrate Queue 把迁移请求缓存下来，如果子集群当前的迁移数量小于最大迁移数量则开始迁移。首先，调用 Client 上的 lockObj() 接口，之后向该对象对应的 OSD 发送请求，OSD 调用 Ceph OSD Component 进行对象迁移，迁移结束后 OSD 调用 Migrator 的 finishObjMigration() 接口，最后再对 Client 上的对象解锁。

4.3.2 客户端接入工作流程

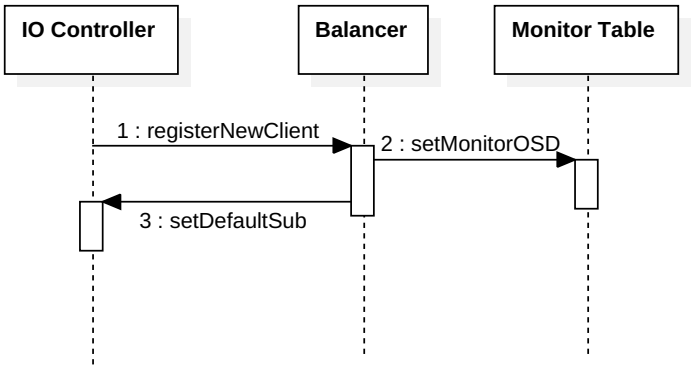

图 4–5 客户端接入的工作流程

Fig 4–5 The Work Flow of Client First Access

客户端初次接入 DOBBS 的工作流程如图4–5所示。从图中可以看到，IO Controller 会首先调用 Center 的 Balancer 的 registerNewClient() 接口，然后该接口通过查询 subHeat，

将当前热度值最低的子集群发送给 IO Controller。同时，它会把请求接入的 Client 的 IP 地址写入 connetedClient 中，然后把当前系统最新的 monitor_osd 通过调用 setMonitorSub() 发送给这个初次接入的 Client。

4.3.3 监控器获得并汇报子集群热度信息工作流程

图4–6所示，是子集群的 Monitor 向 Center 发送当前子集群热度值的工作流程，这个工作流程还包括 Monitor 的 HeatCollector 定期从子集群的 OSD 中获取 IOPS 信息，并根据公式3–1计算子集群热度，最后再把计算好的热度值通过 Thrift 提供的远程调用接口发送给 Center。

对于该工作流，值得注意的是 Reporter 发起的 getUsage 请求是由 Reporter 的 reporting 线程发送的，reporting 线程则是以固定的时间间隔发起 getUsage 请求。HeatCollector 在收到请求之后，会先去在 Analyzer 上去查询本子集群有哪些 OSD。在这之后，它会通过查询到的 OSD 列表调用每个 OSD 上的 getStorageIOPS() 接口，接口返回后，HeatCollector 将其记录并依次调用 getCPUUsage() 和 getMemoryUsage() 接口，最后通过公式计算出热度值并调用 Center 的 reportFromMonitor() 接口将它传输给 Center。

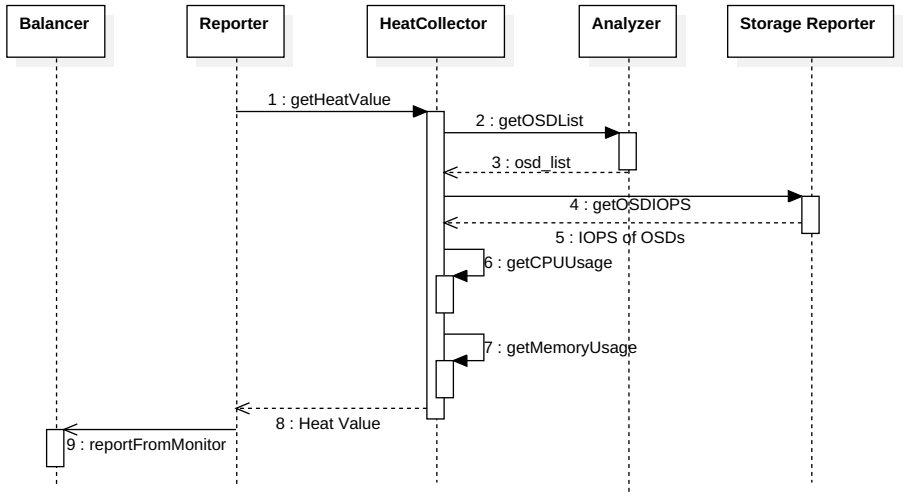

图 4–6 监控器获得并汇报子集群热度信息工作流程

Fig 4–6 The Work Flow of Monitor Reports Heat

4.3.4 全局热均衡使能过程工作流程

全局热均衡是本论文的一个核心概念，而全局热均衡的使能过程更是关键。图4–7所示为全局热均衡的使能过程的工作流程。在 Balancer 的不均衡检测算法检测出不均衡之

后，就启动了使能过程。首先就是为了解决迁移过程中的一致性问题，我们用 try-lock 机制来对 Monitor 上锁。但是图4–7中，我们假设的是成功上锁之后的工作流程。如果不能成功地从目标子集群和源子集群获得锁，则不会进行后续的使能过程。在成功获得两个

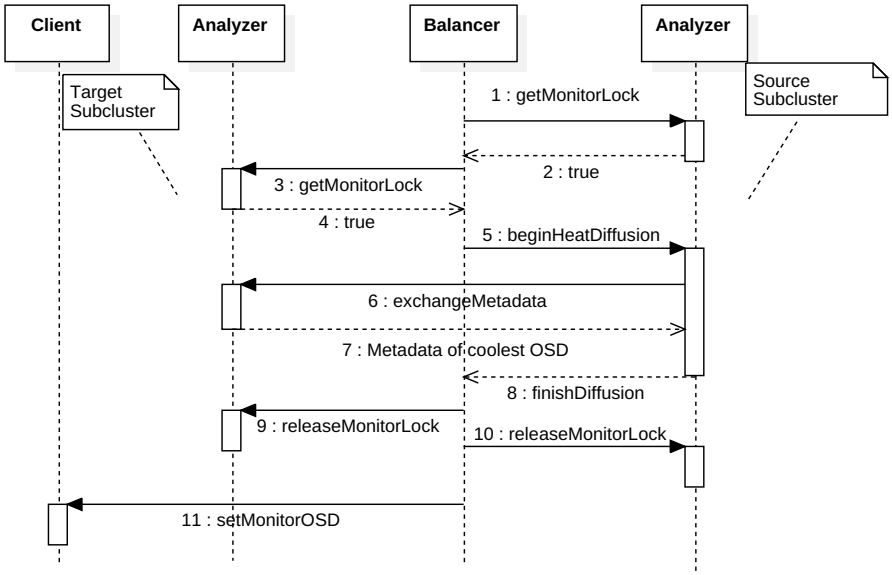

图 4–7 全局热均衡使能过程工作流程

Fig 4–7 The Work Flow of Enable Process of Global Heat Balancing

Monitor 的锁之后，Balancer 会调用源子集群 Monitor 的 Analyzer 的 beginHeatDiffusion() 接口。之后，源子集群会查询最大 IOPS 的 OSD 然后将其对应的元数据（对象热度信息）打包发送给目标子集群，通过目标子集群的 exchangeMetadata() 接口。目标子集群再去查询当前子集群 IOPS 最低的 OSD，将该 OSD 对应的元数据返回给源子集群。在所有过程完成之后，源子集群再调用 Balancer 的 finishDiffusion() 接口，完成元数据交换。随后，Balancer 对两个子集群的 Monitor 解锁。最后则是更新所有 Client 上的 Monitor Table。

4.4 本章小结

本章基于上一章设计的基础上对 DOBBS 的系统各个模块的设计进行了介绍。首先，我们介绍了 DOBBS 构建过程所用到的工具和开源项目，主要有 Ceph、QEMU 和 Apache Thrift。我们选用的主要开发语言为 C++。本章第二小节，我们首先介绍了 DOBBS 的各个模块的类图框架，然后对各个模块分别展开了叙述。因为 DOBBS 是在 WHOBBS 的基础上进行的扩展和修改，所以我们对 WHOBBS 部分的实现并没有做出详细的叙述。本章第三小节则是对系统的主要工作流程的抽象与总结。我们对全局热均衡的主要三个流程进行了详细的介绍。下一章，则是根据本章所实现的系统做出实验验证。

第五章 系统实验与验证

在第三章，我们介绍了 DOBBS 系统的架构设计和两个重要的概念。第四章，我们在系统设计的基础上介绍了系统各个模块的实现。本章则在前两章基础上，对整个系统进行了验证。

5.1 实验环境配置和搭建

在本论文的实验部分需要构造的主要有三部分，一是 Ceph 集群，二是 Monitor 集群和 Center，三则是 Client。在本实验中，我们采用了 4 台服务器作为 OSD 搭载的机器，而 Ceph 集群需要一个 Ceph 监控器，所以我们用一台专用的服务器表示 Ceph 监控器。因此，我们一共使用五台服务器构建 Ceph 集群。这五台服务器都是 HP Compaq Pro 6300 Microtower，他们的 CPU 是 Intel Core i3-3220 3.30GHz，内存是 4G DDR3 SDRAM。其上的操作系统是 Ubuntu 14.04.5 内核是 Linux kernel 的 4.4.0-31-generic 版本。其中四台服务器作为 OSD 搭载服务器，有两台搭载 SSD，两台搭载 HDD。SSD 是 120GB KINGSTON V300 SATA3 SSD，HDD 是 1TB Seagate 520S SATA HDD。一共有四块 SSD，四块 HDD，搭载 SSD 的机器各有两块 SSD，搭载 HDD 的机器各有两块 HDD，我们一共有 8 块存储设备，4 块 SSD 和 4 块 HDD。我们就在每一块存储设备上部署为 Ceph OSD，因此共有 8 个 OSD，总存储大小为 480MB+4GB。

Monitor 运行在独立的服务器上，服务器的配合是 HP Compaq Pro 6300Microtower，CPU 为 Intel Core i3-3220 3.30GHz，内存是 4G DDR3 SDRAM，操作系统为 Ubuntu 14.04.5，内核是 Linux kernel 的 4.4.0-31-generic 版本。另外的服务器用来运行 Client。所有的 Monitor、Ceph 集群和 Client 通过 100MB/s 以太网进行连接。

在 Client 上，我们用 QEMU-KVM 作为虚拟机管理器，它模拟了 1 个 vCPU 和 1GB 内存，并且 guest 操作系统为 Ubuntu server 14.04.5。在准备好硬件之后，我们将软件部署在了各个机器上，用到的 Ceph 版本是 0.94.1。在准备好硬件之后，我们将软件部署在了各个机器上，用到的 Ceph 版本是 0.94.1，QEMU 版本是 2.8.0。我们在这 8 个 OSD 上创建 Ceph 存储池。考虑到 DOBBS 把 Monitor 和存储集群分割成若干子集群，所以在本实验中，我们让一个 HDD OSD 和一个 SSD OSD 组成混合存储池，那么本实验共有 4 个子集群。这个 4 个子集群的编号为从 1 到 4。

因为 DOBBS 是 WHOBBS 的升级版本，并且解决了 WHOBBS Monitor 的性能问题，所以我们还在以上服务器的基础上构造了 WHOBBS 集群。WHOBBS 与 DOBBS 的存储

集群有个比较大的区别，WHOBBS 把所有 4 个 SSD OSD 作为了一个 SSD 存储池，相应地 4 个 HDD 作为了一个 HDD 存储池，而 DOBBS 把每个 OSD 都作为了一个独立的存储池。

5.2 性能比较

根据第三章的系统设计，整个 DOBBS 的性能比较被分为两个部分，一个是针对局部热均衡的性能验证，还有就是针对全局热均衡的有效性验证。

5.2.1 局部热均衡性能验证

考虑到 DOBBS 是 WHOBBS 改进版本，而 DOBBS 的局部热均衡部分也是借鉴 WHOBBS 来实现。WHOBBS 通过动态监测对象的数据流信息，产生合适的迁移策略保证热的对象放在 SSD 上冷的对象放在 HDD 上。所以在本论文中，我们直接简介 WHOBBS 对于系统性能验证，这样也就证明了本论文中的局部热均衡是有效的。

在 [1] 中，Shen 等将 WHOBBS 与原生 Ceph 系统做了比较。他们通过块 IO 流和文件 IO 流两种方式进行评测。结果显示，在不同的数据流下，WHOBBS 相较于原生 Ceph 在性能上取得了显著的提升。他们使用 Fio 来保证块 IO，Fio 访问数据是服从 Zipf 分布的。在块 IO 数据流下，当 zipf 的 theta 值小于 2.25 时，WHOBBS 的 IOPS 是原生 Ceph 的 2.5 倍。在 zipf 大于 2.5 后，这个时候大部分数据已经位于 SSD 上，所以 WHOBBS 的 IOPS 和 Ceph 相差并没有很大。对于三种文件 IO 数据流，分别是邮件服务器、文件服务器和 Web 服务器。在每种文件 IO 数据流下，WHOBBS 都展现出较好的性能优势，即使在文件服务器数据流下，WHOBBS 相较于原生 Ceph IOPS 都有 2.5 倍的优势。因此，可以认为 WHOBBS 是高效的。

5.2.2 WHOBBS 监控器性能比较

表 5–1 DOBBS 和 WHOBBS 的 Monitor 状态

Table 5–1 Monitor Status of DOBBS and WHOBBS

	DOBBS Monitor	WHOBBS Monitor
Migration Time	98	131
CPU Usage(%)	10.99	12.14
Bandwidth(KB/s)	3.24	4.68
Memory Usage(MB)	2.98	4.68

DOBBS 通过多 Monitor 架构来解决 WHOBBS 单一 Monitor 成为性能瓶颈的问题，并且引入了子集群的概念，但是在子集群引入之后又有可能出现子集群之间访问不均衡的情况，我们就用全局热均衡来动态监测这一情况并产生策略，进行热扩散。对于全局热均衡的验证，我们也是通过两个方面：DOBBS 的根本目的是解决 WHOBBS 单一 Monitor 性能瓶颈问题，所以应当比较 WHOBBS 的 Monitor 和 DOBBS 的 Monitor 在相同数据流下的性能区别；在子集群访问不均衡的时候，热扩散的有效性。

表格5–1表示了在相同数据流下，DOBBS 的 Monitor 和 WHOBBS 的 Monitor 在相同时间内平均迁移次数、CPU 使用率、网络带宽和内存占用率的情况。考虑到 Monitor 的实现，内存利用率主要来自于 Analyzer 对对象数据流信息的存储，而 CPU 利用率主要来自于随着对象流信息数量的增长每次迭代的计算。因此，内存使用率和 CPU 使用率都会随着对象数据流信息的增加而增大。在 DOBBS 中，我们用对 Monitor 架构分担了以前 WHOBBS 单一 Monitor 的计算压力。在 WHOBBS 中，单一 Monitor 承担了整个集群的对象信息，并且也承担了对所有 Client 信息的采集。DOBBS 与之不同，因为多个 Monitor 的参与，使得不需要再由一个 Monitor 管理所有的集群和 Client。并且，由于全局热均衡的影响，一旦 DOBBS 的子集群中某个 Monitor 过热，Center 节点感知到之后，发送热扩散请求，这样 Monitor 的相应利用率也会随之下降。如表格5–1中所示，DOBBS Monitor 的资源利用率远小于 WHOBBS。因此，可以证明 DOBBS 解决了 WHOBBS 的性能问题。

5.2.3 全局热均衡有效性验证

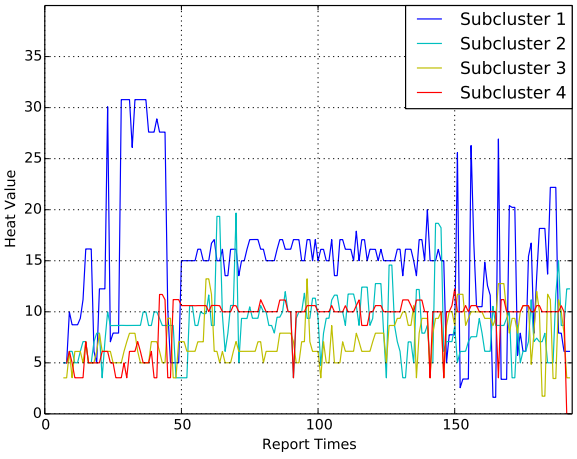

图 5–1 关闭全局热均衡的 DOBBS 子集群热度值

Fig 5–1 Heat Value of Subcluster without Global Heat Balancing

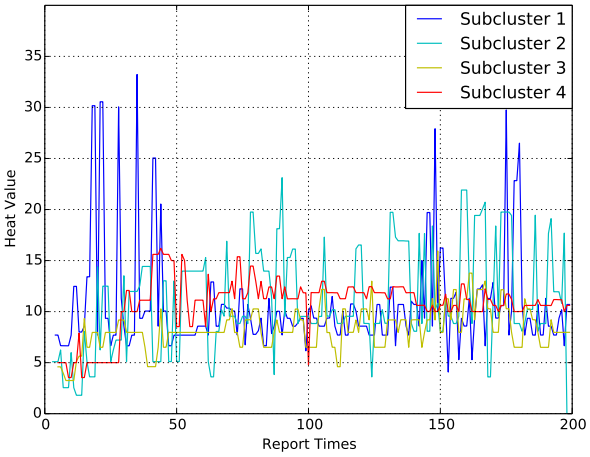

图 5–2 DOBBS 的子集群热度值

Fig 5–2 Heat Value of Subcluster with Original DOBBS

考虑到全局热均衡的设计目标和实现目标，我们在验证全局热均衡有效性的时候将原始 DOBBS（未做任何修改的 DOBBS）与关闭了全局热均衡功能的 DOBBS 进行比较。我们知道，全局热均衡的根本目的是为了解决多个子集群后热度不均衡的情况，所以我们维持了 DOBBS 的多子集群架构，但是只把全局热均衡功能关闭，来与原始 DOBBS 进行对比。图5–1和图5–2分别表示关闭全局热均衡特性的子集群和原始 DOBBS 子集群的热度值随时间的变化。DOBBS 的全局热均衡使得各个子集群的热度值相对均衡，不会出现某个子集群热度值畸高的情况，DOBBS 解决这个问题是通过将过热子集群的“热量”扩散到其他子集群来实现的。两个图中的纵坐标表示子集群的热度值，该热度值是在由公式3–3计算得出的，我们在每个 Monitor 加入了 debug 模块，专门用来追踪热度值随时间的变化情况。横坐标为汇报时间，在这两个图中我们将汇报间隔平分成数个时间槽，并且每个时间槽都有相同的长度。为了测试全局热均衡的有效性，我们实现了一个没有 Center 节点的集群，在这个集群中的 Monitor 不会收集存储集群的 IOPS 也不会计算热度值，每个子集群都是独立的结构。在测试的时候，我们设计了一个在 VM 上运行的独特的数据流。数据流的产生是通过修改一个叫做 Filebench 的基准测试工具实现的，我们将 Filebench 运行在 Client 的虚拟机上动态产生对虚拟机磁盘的访问请求。在本实验中，我们设置的热度阈值为 30。

对于这两个实验，我们采取了一个相对极端的做法，我们将若干运行了 Filebench 的 Client 都接入在 subcluster 1 上，而其他子集群只是接入平稳运行虚拟机的 Client。这个 Filebench 的数据流实际上是模仿虚拟机的加载过程在开始会有一个大量访问增加的情况，之后会趋于平稳。

表 5–2 子集群的平均热度值

Table 5–2 Average Heat Value of Subclusters

	Original DOBBS	without Heat Diffusion
subcluster 1	11.05	16.04
subcluster 2	10.69	9.0
subcluster 3	8.28	7.28
subcluster 4	10.73	8.75

从图5–1很清楚的看到，在 20-45 的时间槽内出现了一个连续的高峰，而其他子集群的热量则是处于很低的水平的。这是因为我们将所有数据流在加在 subluster 1 上。而在 50-150 的时间段内，subcluster 1 都一直处于热度值的高位，此时各个子集群之间的热度值标准差已经很大。从图5–2可以看出，采用同样的数据流，虽然在 20-45 的时间槽内出现过高峰，但是高峰并不平稳，而是立刻降了下来，即使出现了新的高峰，也都可以快速地下降。观察其他子集群的热度值，可以显著得发现其他子集群的热度值相较于图5–1有了显著提升，这就说明了全局热均衡把 subcluster 1 的热量分摊到了其他子集群。综上，我们可以证明全局热均衡可以消除一个子集群过热的情况。但是，我们需要注意的是，我们定义热扩散是一个懒迁移过程，也就是说即使元数据被迁移之后，也需要等到局部热均衡来做后续的数据迁移工作。因此，我们从图5–1和图5–2看到的 subcluster 1 的热度值得到显著下降，大部分原因是元数据的迁移之后，Client 开始向新的子集群发送数据信息导致的。而热扩散则是一个长期的过程，我们通过两张图并不能完全展现出来，因为它要配合局部热均衡才能实现，所以时间会比较长。虽然没有在本文中呈现后续的实验结果，但是还是可以证明全局热均衡的有效的。

我们不仅通过图的方式说明全局热均衡的有效性。表格5–2所示，可以从数值上直观地看到各个子集群的平均热度值。明显的看到，subcluster 1 在原生 DOBBS 下的热度值明显低于没有全局热均衡的 DOBBS。而其他子集群的热度值也明显有所增加，所有热度值的标准差明显减小。通过计算可以知道，原始 DOBBS 的四个子集群的平均热度标准差为 1.28，而关闭全局热均衡的四个子集群的平均热度标准差为 3.92。

尽管 DOBBS 消除了 WHOBBS 存在的性能问题并且全局热均衡又可以做到将过热的子集群的”热量“扩散到其他子集群上，可是在 Client 上运行的 VM 可能并不能获得比 WHOBBS 更好的性能提升。我们考虑到 DOBBS 的热扩散过程，为了保证迁移过程中数据的一致性，防止热扩散的使能过程在传输元数据时和局部热均衡互相干扰，我们用 try-lock 机制来保护他们的相互影响。由 try-lock 机制我们可以知道，一旦对两个集群上锁，那么局部热均衡将无法继续进行，同理在这个时间间隔内 Client 也无法向

Monitor 汇报对象的数据流信息。这样势必会影响 Client 上 VM 的运行。但是，这个影响其实也是微乎其微的，因为在我们大量试验下发现，热扩散的使能过程的耗时基本上是小于 200ms。所以，DOBBS 相比于 WHOBBS 对 Client 上运行的 VM 并没有做到优化。

5.3 软件定义存储验证

上面的小节我们从系统性能的角度验证了与 WHOBBS 相比 DOBBS Monitor 的性能优势，以及 DOBBS 全局热均衡的有效性。本小节我们从功能的角度证明 DOBBS 是软件定义的存储系统。

在第三章的最后一节论证了 DOBBS 的局部热均衡和全局热均衡都是满足了软件定义系统的特性。首先对于局部热均衡，我们将子集群的控制面与存储面相分离，在实现过程中我们让每个子集群中的 Monitor 收集和分析对象的数据流信息并发送迁移请求，同时它也会更新 Client 上的 Object Table，这是控制面；我们通过修改 Ceph 和 QEMU 的源代码，让 VM 在 IO 请求之后先通过 IO Controller 查询到当前请求所对应的 OSD ID，再交给 Ceph OSD Component 进行后续的数据传输，由于 IO Controller 的存在，VM 上的应用程序是不清楚对象具体在哪个 OSD 上的，这是存储面。表5–3中的数据是我们从上一小节的原生 DOBBS 的子集群中获取到的，它表示了各个子集群 SSD 和 HDD 间迁移对象的数量。可以看到每个子集群的从 HDD 到 SSD 的迁移数量都比 SSD 到 HDD 的数量多，这是因为我们在第三章的迁移策略是在一开始将所有对象通过热度排序后现将所有在对头的对象移动到 SSD 上，所以这个时候会有大量的 SSD 到 HDD 的迁移，在 SSD 的数量满了之后就是根据迁移策略将对象在 SSD 和 HDD 间迁移。通过表格的数据可以证明，对象确实是在底层持续迁移的，而 VM 的应用程序仍然可以平稳正确的运行，因此局部热均衡是软件定义的。

表 5–3 局部热均衡的迁移次数

Table 5–3 Migration Count of Local Heat Balancing

	SSD to HDD	HDD to SSD
subcluster 1	22	76
subcluster 2	11	57
subcluster 3	9	54
subcluster 4	9	55

对于全局热均衡的软件定义的验证实际我们已经在上一小节进行了验证，在全局

热均衡的热扩散过程我们将 subcluster 1 的热量扩散到其他子集群的过程，实际就已经修改了子集群间的逻辑结构。在实验开始时，我们默认 subcluster 1 的 OSD 为 1 和 2，subcluer 2 的 OSD 为 3 和 4，以此类推。而实验结束之后，每个子集群的 OSD 编号都不再是系统初始化时候的编号。由此，我们也可以证明全局热均衡过程也是软件定义的。

5.4 本章小结

本章我们对 DOBBS 的系统设计和系统实现加以了实验上的论证。首先是对实现环境的搭建，我们配置了实验的硬件环境和软件环境。为了体现 DOBBS 消除了 WHOBBS 存在的性能问题，我们不仅部署了 DOBBS 在 Monitor 上，也配置了 WHOBBS 在 Monitor 上。本章第二节，我们引用了 WHOBBS 的论文结果论证了本文中的局部热均衡的高效性。其次，我们用 WHOBBS 的 Monitor 与 DOBBS 的 Monitor 做比，证明了 DOBBS 成功地消除 WHOBBS 的性能问题。之后，我们配置了一套不具有全局热均衡的 DOBBS 集群与原生 DOBBS 集群在相同数据流下作比较，实验结果可以明显的证明 DOBBS 全局热均衡的有效性。第三节则论证了 DOBBS 的局部热均衡和全局热均衡过程都是软件定义的。

第六章 总结与展望

6.1 本文总结

随着互联网行业的发展，如今的互联网正处于一个信息爆炸的时代。面对信息爆炸的互联网，对信息的存储和处理也就产生了海量的数据。为了高可用性和节约资源，IaaS 云常常被用来部署大数据分析和其他各种类型的应用。虚拟机被认为构建 IaaS 云的基础，而虚拟机常常是基于虚拟机磁盘镜像（VMDI）构建的。传统的云存储解决方案，如 NAS 越来越难以适应大量的云存储需求，对象存储由于其高扩展性可以更好地适应现在的大规模云存储。存储介质方面，SSD 由于其读写性能优势以及较低的能耗被越来越多地运用在数据中心的存储中。但是，SSD 的价格还是非常的昂贵，难以适应大规模数据中心的存储需求，所以将 SSD 与传统 HDD 构件混合存储系统则是一个主流。而近些年被提出的软件定义存储，由于其简化了虚拟机存储的端到端开销，并且通过自治的方式对存储集群的结构进行优化，使它更加适合运用于混合存储系统。

本实验室学长之前的针对虚拟机块存储所研究的软件定义的混合存储系统 MOBBS 和 WHOBBS，都是动态监测虚拟机数据对象的访问频率和访问类型然后做出合适的迁移策略，使得较“热”的对象总是分布在 SSD 上而较“冷”的对象总是分布在 HDD 上。WHOBBS 是 MOBBS 的升级版本，它对 MOBBS 的客户端性能进行了优化。而 WHOBBS 则是一个单监控器架构的混合存储系统，我们知道单监控器就必须面临单点故障或压力过大的问题。

为了解决 WHOBBS 单监控器架构所带来的性能问题，我们提出了 DOBBS，一个两层多监控架构的软件定义的混合存储系统。DOBBS 与 WHOBBS 的不同在于，我们有多个 Monitor，它们将整个集群划分成多个子集群，子集群相对独立并且由一个 Monitor 进行管理。我们提出了两个重要的概念，分别是局部热均衡和全局热均衡。局部热均衡只是针对每个子集群内部的局部热均衡，因此每个 Monitor 只能保证所在子集群的局部热均衡，它的根本目标就是让热对象放置于 SSD，让冷对象放置于 HDD。局部热均衡的设计是沿用 WHOBBS，我们只是做出了比较小的修改。全局热均衡则是在多子集群架构之后，可能会出现某个子集群遭受较大数据流而导致子集群间热度不均衡的情况，引入的概念。全局热均衡主要包括两个部分的内容，分别是热度不均衡的检测和热扩散过程。热度不均衡的检测中，我们提出了衡量子集群热度的方法以及热度检测的算法。热扩散是全局热均衡的重点，主要包括使能过程和懒迁移过程。为了解决大规模数据迁移，我们提出了全新的概念——热扩散，这一概念是源自于物理学的热扩散。我们只对

元数据进行迁移，并且修改子集群的逻辑结构，而后续的数据迁移则是交给局部热均衡来完成的。全局热均衡的过程也是软件定义的，全局热均衡根据每个子集群的热度本身来重新修改各个子集群的逻辑结构，从而使得整个系统达到热均衡的状态，这样则让系统自治地对存储集群的机构进行优化，并且整个过程对上层应用并不可见。

在介绍系统的设计之后，我们对 DOBBS 的系统实现进行了详细的介绍，为了防止避重就轻，我们将 WHOBBS 的实现部分做了很大程度的省略。在实现 DOBBS，我们选用了现在主流的 Ceph 提供对象存储，QEMU 作为虚拟机管理程序，Apache Thrift 作为模块间远程过程调用框架。我们将系统的类图做出了详细的绘制，包括每个类的设计细节和各个接口的介绍。值得注意的是，在 WHOBBS 的基础上我们增加了 Center 服务器，所以 Center 服务器是我们系统实现的重点与关键。我们还介绍了系统几个重要的工作流程，它们是通过 UML 时序图的方式绘制的，然后我们对它们也做了详细的解释。

在论文的最后，我们需要实验来验证 DOBBS 的高效性和高可用性，于是我们对实验的物理环境和软件环境进行了搭建。我们采用了 8 个 OSD 来表示存储集群，并用 4 个 Monitor 来将存储集群分割成 4 个子集群。对于局部热均衡的实验验证在 WHOBBS 的论文中已经验证过，所以本文并没有针对局部热均衡进行大量的实验验证。而对于全局热均衡，我们主要从两个角度进行的验证，分别是比较 DOBBS 与 WHOBBS 的 Monitor 在相同数据流下的资源利用差距，以及 DOBBS 的全局热均衡的有效性。对于后者，我们又实现了一个没有全局热均衡功能的 DOBBS，让它与原生 DOBBS 进行比较。大量实验证明，DOBBS 解决了 WHOBBS 的 Monitor 的性能问题，同样也验证了全局热均衡的有效性。

综上所述，我们设计并实现了一个高效的分布式混合块存储系统，它不仅可以从 SSD/HDD 的层面充分利用 SSD 的性能优势，还能做到分布式监控节点的负载均衡。在文章的最后，我们还通过实验验证了设计的高效性和实现的可行性。

6.2 未来展望

我们在第三章曾叙述过 DOBBS 的设计目标是解决 WHOBBS 的系能问题，但是我们论述了 WHOBBS 所保存的数据并不是十分重要，因此我们实际上忽略了当 Monitor 宕机之后存在的问题。所以，我们在未来应该利用一致性协议或者多版本技术来保证 Monitor 宕机之后子集群仍能进行局部热均衡，例如将它所有的 OSD 划归到其他子集群中。

在系统实现中，我们实际并没有用到 Raft 协议来作为 Monitor 之间的一致性协议，而是使用了一个单独的服务器作为 Center。因此，后续的研究和实现可以从这里出发，实现将 Monitor 与 Center 整合，并通过 Raft 解决快速错误恢复。

其次，本论文仅仅探索了 SSD 和 HDD 这两种存储介质，那么多余种类繁多的存储介质，应该有一种通用的方案来解决存储介质间迁移的策略。

参考文献

[1] LINGXUAN S, CHEN H, MA S, et al. WHOBBS: An Object-Based Distributed Hybrid Storage Providing Block Storage for Virtual Machines[C]// High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on. IEEE. [S.l.]: [s.n.], 2015: 160–165.

[2] 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. [S.l.]: [s.n.], 2006: 307–320.

[3] BHARDWAJ S, JAIN L, JAIN S. Cloud computing: A study of infrastructure as a service (IAAS)[J]. International Journal of engineering and information Technology, 2010, 2(1): 60–63.

[4] FENG Y, LI B, LI B. Price competition in an oligopoly market with multiple iaas cloud providers[J]. IEEE Transactions on Computers, 2014, 63(1): 59–73.

[5] SEFRAOUI O, AISSAOUI M, ELEULDJ M. OpenStack: toward an open-source solution for cloud computing[J]. International Journal of Computer Applications, 2012, 55(3).

[6] KASAVAJHALA V. Solid state drive vs. hard disk drive price and performance study[J]. Proc. Dell Tech. White Paper, 2011: 8–9.

[7] WU F, SUN G. Software-defined storage[J]. Report. University of Minnesota, Minneapolis, 2013.

[8] CARLSON M, YODER A, SCHOEB L, et al. Software defined storage[J]. Storage Networking Industry Assoc. working draft, Apr, 2014.

[9] MA S, CHEN H, LU H, et al. MOBBS: A Multi-tiered Block Storage System for Virtual Machines Using Object-Based Storage[C]// High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on. IEEE. [S.l.]: [s.n.], 2014: 272–275.

[10] GHEMAWAT S, GOBIOFF H, LEUNG S.-T. The Google file system[C]// ACM SIGOPS operating systems review. Vol. 37. 5. ACM. [S.l.]: [s.n.], 2003: 29–43.

[11] SHVACHKO K, KUANG H, RADIA S, et al. The hadoop distributed file system[C]// Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on. IEEE. [S.l.]: [s.n.], 2010: 1–10.

[12] FACTOR M, METH K, NAOR D, et al. Object storage: The future building block for storage systems[C]// Local to Global Data Interoperability-Challenges and Technologies, 2005. IEEE. [S.l.]: [s.n.], 2005: 119–123.

[13] WAN L, LU Z, CAO Q, et al. SSD-optimized workload placement with adaptive learning and classification in HPC environments[C]// Mass Storage Systems and Technologies (MSST), 2014 30th Symposium on. IEEE. [S.l.]: [s.n.], 2014: 1–6.

[14] LI Z, CHEN M, MUKKER A, et al. On the trade-offs among performance, energy, and endurance in a versatile hybrid drive[J]. ACM Transactions on Storage (TOS), 2015, 11(3): 13.

[15] WAN L, CAO Q, WANG F, et al. Optimizing checkpoint data placement with guaranteed burst buffer endurance in large-scale hierarchical storage systems[J]. Journal of Parallel and Distributed Computing, 2017, 100: 16–29.

[16] CHEN F, KOUFATY D A, ZHANG X. Hystor: making the best use of solid state drives in high performance storage systems[C]// Proceedings of the international conference on Supercomputing. ACM. [S.l.]: [s.n.], 2011: 22–32.

[17] KRISH K, ANWAR A, BUTT A R. Hats: A heterogeneity-aware tiered storage for hadoop[C]// Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on. IEEE. [S.l.]: [s.n.], 2014: 502–511.

[18] 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. [S.l.]: [s.n.], 2006: 122.

[19] PAIVA J, RUIVO P, ROMANO P, et al. A uto P lacer: Scalable Self-Tuning Data Placement in Distributed Key-Value Stores[J]. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2015, 9(4): 19.

[20] ZHANG L, DENG Y, ZHU W, et al. Skewly replicating hot data to construct a power-efficient storage cluster[J]. Journal of Network and Computer Applications, 2015, 50: 168–179.

[21] SONI G, KALRA M. A novel approach for load balancing in cloud data center[C]// Advance Computing Conference (IACC), 2014 IEEE International. IEEE. [S.l.]: [s.n.], 2014: 807–812.

[22] DENG Y, LAU R W. Dynamic load balancing in distributed virtual environments using heat diffusion[J]. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 2014, 10(2): 16.

[23] BARROSO L A, CLIDARAS J, HÖLZLE U. The datacenter as a computer: An introduction to the design of warehouse-scale machines[J]. Synthesis lectures on computer architecture, 2013, 8(3): 101–102.

[24] ZHAN WANG H C, HU F. RHOBBS: An Enhanced Hybrid Storage Providing Block Storage for Virtual Machines. 2016.

[25] COULOURIS G F, DOLLIMORE J, KINDBERG T. Distributed systems: concepts and design[M]. [S.l.]: pearson education, 2005: 19–20.

[26] THERESKA E, BALLANI H, O'SHEA G, et al. IOFlow: a software-defined storage architecture[C]// Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM. [S.l.]: [s.n.], 2013: 182–196.

[27] ONGARO D, OUSTERHOUT J K. In search of an understandable consensus algorithm.[C]// USENIX Annual Technical Conference. [S.l.]: [s.n.], 2014: 305–319.

[28] NASSERI M, JAMEII S M. Concurrency control methods in distributed database: A review and comparison[C]// Computer, Communications and Electronics (Comptelix), 2017 International Conference on. IEEE. [S.l.]: [s.n.], 2017: 200–205.

[29] MILOJIČIĆ D, LLORENTE I M, MONTERO R S. Opennebula: A cloud management tool[J]. IEEE Internet Computing, 2011, 15(2): 11–14.

[30] BELLARD F. QEMU, a fast and portable dynamic translator.[C]// USENIX Annual Technical Conference, FREENIX Track. [S.l.]: [s.n.], 2005: 41–46.

[31] KIVITY A, KAMAY Y, LAOR D, et al. Kvm: the Linux virtual machine monitor[C]// Proceedings of the Linux symposium. Vol. 1. [S.l.]: [s.n.], 2007: 225–230.

攻读学位期间发表的学术论文

[1] 第一作者. EI 国际会议论文, 2017.