|  |  |
| --- | --- |
| 分类号：TP39 | 说明: 说明: xh1 |
| 10710-2104224013 |



硕 士 学 位 论 文

面向对象存储系统负载均衡方法研究与实现

惠小红

|  |  |  |  |
| --- | --- | --- | --- |
| 导师姓名职称 | 范中磊教授 | | |
| 申请学位级别 | 专业学位硕士 | 学科专业名称 | 计算机技术 |
| 论文提交日期 |  | 论文答辩日期 | 2016.06.13 |
| 学位授予单位 | 长安大学 | | |

**Research and Implementation of a Load Balancing Method for Cloud Storage**

A Dissertation Submitted for the Degree of Master

**Candidate：Hui Xiaohong**

**Supervisor：Prof. Fan Zhonglei**

Chang’an University, Xi’an, China

论文独创性声明

本人声明：本人所呈交的学位论文是在导师的指导下,独立进行研究工作所取得的成果。除论文中已经注明引用的内容外，对论文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本论文中不包含任何未加明确注明的其他个人或集体已经公开发表的成果。

本声明的法律责任由本人承担。

论文作者签名： 年 月 日

论文知识产权权属声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属学校。学校享有以任何方式发表、复制、公开阅览、借阅以及申请专利等权利。本人离校后发表或使用学位论文或与该论文直接相关的学术论文或成果时，署名单位仍然为长安大学。

（保密的论文在解密后应遵守此规定）

论文作者签名： 年 月 日

导 师 签 名： 年 月 日

# 摘 要

基于对象的分布式存储系统(Object-Based storage,OBS)面临的第一个问题就是存储节点中数据的分布。并且数据的组织优化是提高系统性能的有效方法之一，合理的数据分布策略可提高系统对用户操作的响应时间。因此，调研和分析现有对象存储系统中数据分布策略，例如S3和Swift数据分布策略。上述两种分布策略都有一定的优势和不足，本文针对上述分布策略的不足，进行一定的扩展和改进。提出了一种扁平化的数据分布策略——FDDM（Flat Data Distribute Method）。该策略是基于OID分段式的数据分布方法。是对已有对象数据分布方法的扩展和补充。目录层次的划分粒度更加细化，结合了文件系统目录分布和对象存储扁平化分布的优点。FDDM策略可以根据用户的数量级，动态配置目录层次和每层中目录容量。策略的实现不受限于用户数据量。任何数量级下FDDM策略都能以最快的速度达到数据的均匀分布。在OBS系统中仿真实现三种数据分布策略，观察目录结构，并进行性能的测试。通过实验结果，发现该策略改善了对象存储系统中单一目录下数据量过多，数据分布不均匀的问题。实现了数据的扁平化分布。有效优化了数据的存取效率和系统的性能。

负载均衡一直都是分布式存储系统领域研究的热点问题。首先调研现有分布式存储系统负载均衡技术，分析算法优势和存在的不足。借鉴已存在的负载均衡技术。结合本系统架构进行负载均衡技术的研究和设计。本系统采用的是动静态相结合的负载均衡算法。静态算法是指根据监控系统收集和计算的各个存储节点的负载。然后使用节点负载和轮询相结合的方式实现节点分配。动态负载均衡是指当系统各节点负载倾斜时（系统扩展，节点失效，用户数据不均等），采用动态的数据迁移策略，实现数据动态迁移。完成负载的再平衡过程。上述负载均衡算法，已在本系统中得以应用和实现。

**关键词****：**对象存储，扁平化数据分布，FDDM，负载均衡，动静态结合，

负载迁移

# Abstract

The first problem is the distribution of data in the storage nodes faced in object-based storage(OBS) systems. And the organization of data is one of the effective methods to improve the system performance. Reasonable data distribution strategy can improve the system response time to user operation. Therefore, research and analysis of data distribution strategies in existing object storage systems, such as the data distribution strategies of S3 and Swift. There are not only advantages but also shortcomings between both of the distribution strategies, this paper proposed a data distribution strategy of flat data distribution strategy - FDDM (Flat Data Distribute Method) to solve the problem of the above data distribution strategy. The data distribution strategy is based on the OID segmentation, combining the advantages of the file system directory distribution and the flattened distribution of the object storage. The hierarchical granularity of the directory hierarchy is more refined. Is an extension and supplement to the existing method of data distribution. FDDM policy can dynamically configure the directory hierarchy and the directory capacity of each layer according to the number of files. FDDM strategy can fast achieve evenly distributed of the data with any order of magnitude. simulation the three kinds of data distribution strategy in the OBS system, observe the structure of directories, and performance testing. Through the results of the experimental, it is found that the strategy effectively optimizing the efficiency and system performance of data access.

Load balancing has always been a hot issue in the field of distributed storage systems. First of all, the existing distributed storage system load balancing technology, analysis of the advantages of the algorithm and the shortcomings. Using Existing Load Balancing Technology. Combined with the system architecture for load balancing technology research and design. The system uses a dynamic and dynamic combination of load balancing algorithm. A static algorithm is the load of each storage node that is collected and calculated according to the monitoring system. And then use the node load and polling combination of ways to achieve node allocation. Dynamic load balancing means that when the load of each node of the system is tilted (system expansion, node failure, user data inequality), dynamic data migration strategy is adopted to realize dynamic data migration. Complete the load rebalance process. The above load balancing algorithm has been applied and implemented in this system.

Keywords: object storage, flattened data distribution, FDDM, load balancing, the combination of dynamic and static, load migration.

**目 录**

摘 要 II

Abstract 3

第一章 绪 论 7

1.1 研究背景及目的和意义 7

1.2 研究现状 8

1.2.1对象存储研究现状 8

1.2.2负载均衡研究现状 11

1.3 研究内容 13

1.4本文结构 13

第二章 系统理论技术介绍 14

2.1 集群存储技术 14

2.2 对象存储 16

2.3 负载均衡 17

2.4 快照 18

第三章 OBS系统的架构设计 19

3.1 整体架构 19

3.2 对象存储系统各层次组件设计 21

3.2.1 总控节点 21

3.2.2 元数据服务器 22

3.2.3 文件服务器 23

3.2.4 客户端 25

3.2.4 监控系统组件 25

3.3 本章总结 27

第四章 OBS系统负载均衡和对象存储策略设计与实现 28

4.1 负载均衡算法设计 28

4.1.1 静态负载均衡算法设计 28

4.1.2 动态负载均衡算法设计 29

4.2 对象存储系统中负载均衡算法实现 32

4.2.1 迁移算法工作流程 32

4.2.2 数据迁移配置界面 33

4.3 面向存储系统扁平化数据分布策略 35

4.3.1 扁平化数据分布策略设计 35

4.3.2 扁平化数据分布策略实现 38

第五章 实验及结果分析 40

5.1 分布式存储系统仿真环境 40

5.1.1 节点配置 40

5.1.2 分布式存储系统环境配置 40

5.1负载均衡测试结果及分析 41

5.2.1静态负载均衡算法测试 41

5.2.1动态负载均衡算法测试及结果分析 42

5.3目录容量和响应时间关系测试 49

5.3.1实验仿真环境 49

5.3.2 实验过程 49

5.3.3实验结果及分析 50

5.4 FDDM策略测试结果及分析 53

5.5 性能测试结果及分析 57

5.5.1 仿真环境 57

5.5.2 实验过程及实验结果 57

总结 64

参考文献 65

致谢 67

# 绪 论

## 研究背景及目的和意义

互联网技术的不断发展和兴起（例如：AR，VR，神经网络，智能机器人等），以及移动终端的普及。用户每时每刻都有大量数据产生。如今的互联网环境，已不是单纯的互联网技术的竞争和发展，各种数据已成为各公司最宝贵的资源。然而，海量的数据量和数据量的不断增长给存储技术带来了很大的挑战，传统的存储技术已无法应对。为了应对海量数据的存储以及数据量不断增长的需求，云存储技术应运而生。云存储是指通过集群应用、网格技术或分布式文件系统功能，将网络中大量各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个服务系统[1]。国内外已有大量云存储平台应运而生。比如：Amazon的S3，Google Dirve，阿里OSS，百度BOS等。相比与传统的存储技术，云存储系统在可扩展性，利用率，成本和服务能力等方面都有很大的优势。

目前，在企业存储的大量数据中，RDBMS处理的结构化数据仅占15%，而剩下的85%大都是非结构化的数据。例如：文件，音视频，照片，信件等。针对RDBMS存储结构的研究已经相当完善。而面向非结构化的数据管理却是企业大数据管理和信息化处理的一个难题[13]。因此，本文主要针对云存储环境中非结构化数据的存储进行研究。对于非结构化数据来说，最常用的操作是存储和读取数据。传统的RDBMS擅长的是数据查询操作，在此并无用武之地，只会增加系统的复杂度。基于面向对象存储 (Object-Based storage,OBS) 系统的出现解决了上述问题，顺应了存储领域的发展趋势。OBS系统必须要解决的问题就是对象分层结构，即object目录结构。

集群存储将大量的普通PC协同起来共同对外提供存储资源。面临的第一个问题就是数据分布。另外，大规模分布式存储系统的一个重要目标是节省成本。因此，负载均衡是分布式存储系统的重要技术，在实现存储资源利用，系统动态扩展，提高响应速度方面都具有重要的意义 [2]。

最大吞吐量、最小响应时间、最快故障恢复时间是云存储系统追求的重要目标。优化资源分配，避免数据倾斜，不断寻找可以提高系统可靠性和可用性相关技术。是研究云存储平台的主要手段。数据的组织优化模型是提高系统性能的有效方法之一，合理的数据分布策略可提高系统的存储性能，提高系统对用户操作的响应时间。面对高并发量的数据访问，避免出现某些设备很忙，某些设备空闲的情况。负载均匀分配至关重要。扁平化数据分布能够大大提高系统的响应速度。优化资源分配可以有效提高资源利用率，最大化系统吞吐。动态监控手段可以实时了解系统负载状态，并且能快速进行故障定位和处理。提高系统的可靠性和可用性。提供良好的用户体验。因此，基于云存储的OBS系统中，对象存储的数据分布策略和负载均衡的研究具有重要的理论意义和实际价值。

## 研究现状

## 1.2.1对象存储研究现状

基于OBS系统的Object分层结构的研究是计算机存储领域最热门的技术之一。

很多公司都推出自己的OBS服务。对象存储节点（Object-Based Node, OBN）上目录结构合理组织非常重要。存储结构目录组织不合理，将会导致数据分布过于分散或者过于集中的问题。过于分散或者过于集中，都将响系统对数据的处理能力。合理的目录分层结构可以从软件层面提高系统的响应能力。国内外很多企业都在进行OBS系统目录组织的研究工作。

比如：Amazon提出的简单存储服务S3。数据是以object的方式存储在storage node上。从参考文献[3, 4]了解到S3的结构模型如图1.1所示。S3中主要涉及到三个概念：Object、Key和Bucket。Object是S3的基本存储单元。Key是对象的唯一标识符。Bucket是用来存储对象的容器，作用就类似用户的文件夹[4]。当存储系统中用户数据量较少时，这种OBS模型简单方便且可以满足用户对系统存取性能的要求。但是，在实际环境中，由于用户数据量爆炸式增长。这种存储环境下，使用S3存储模型，最终会导致单一目录下数据量过多。影响系统的响应性能。不利于后期的维护和系统的动态扩展。

OpenStack Object Storage (Swift)也是一种面向对象存储的数据分布模型，与Amazon的S3存储模型相比，Swift采用了一种静态的目录分布结构，在一定程度上缓解了数据分布过于集中的问题。Object存储模型如图1.2所示[5]。devs是设备路径；object directory是对象的存储目录；在object目录下存放各个partion目录，partition分布是系统在逻辑上使用一致性哈希算法将固定总数的partition映射到每个Storage node上。partition目录由若干个suffix\_path组成，suffix\_path的名称就是hash\_path名称的后三位；hash\_path是由文件所属的account、container和object ID组成。所以hash\_path 唯一。suffix\_path目录是由object的hash\_path名称构成的目录，在hash\_path目录下存放关于object的数据和元数据信息。通过上面对swift对象存储模型的详细介绍知道devs，object，partion是在对象存储中固定的三层目录结构。swift是通过suffix\_path，hash\_path两层目录扁平化对象的分布。hash\_path目录下存放对象。swift这种两层、静态对象分布模型。针对目前数据量，在一定程度上缓解了数据分布过于集中，即单一目录下数据分布过多的问题。但是，这种静态的数据分布算法适用于用户数量级固定的模型。所有数据在OBN中是否均匀分布严重依赖于用户的数据量。采用hash\_path后三位划分目录的方法，使得suffix\_path层目录容量确定。当用户数量级过大时，将导致suffix\_path层目录下子目录过多。

国内像阿里的OSS （Open Storage Service）、腾讯的COS Cloud Object Service）、新浪的SCS(Sina Cloud Storage)都是采用类似Amazon S3的对象分布模型。

针对上述问题，本系统提出了一种合理的基于对象的动态可调的扁平化数据分布策略——FDDM（Flat Data Distribution Method）。FDDM策略是基于OID分段式的数据分布方法。是对已有对象数据分布方法的扩展和补充。目录层次的划分粒度更加细化，结合了文件系统目录分布和对象存储扁平化分布的优点。FDDM策略可以根据用户的数量级，动态配置目录层次和每层目录容量。策略的实现不受限于用户数据量。任何数量级下FDDM策略都能以最快的速度达到数据的扁平化分布。

bucket

object

object

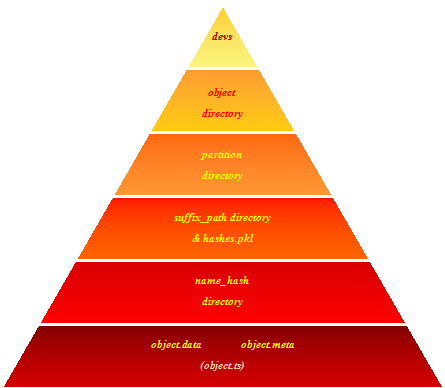
object

data

key

metadata

**图1.1 S3的基本架构图**



**图1.2 swift对象目录分布层次**

## 1.2.2负载均衡研究现状

实现负载均衡技术主要是为了实现存储资源的动态扩展，提高资源利用率，最小化响应时间，降低故障恢复和切换时间，使的无论在什么情况下，用户都能获得较好的访问质量[6]。对于存储系统而言，数据就是负载。负载均衡主要表现在数据能均匀的分布在各个存储节点。

然而，在分布式存储系统中，节点的动态扩展，节点故障，用户数据的动态添加，删除等。都会导致系统中各服务器负载的不均衡分配。为了保证集群系统中负载的均衡分配，最好的解决方法就是在集群系统中采用一种均衡分配算法。根据系统的设置，动态的实现整个系统负载的均匀分配。

负载均衡是分布式系统中关键技术之一。因此，具有很多关于负载均衡策略的研究。改进的一致性hash算法就是解决数据均衡分配的一个经典算法。

例如： Amazon Dynamo,Swift的Ring组件，Yahoo的Pnuts，Cassandra等存储架构中。Consistent hashing的基本思想就是将object和Node都映射到同一个hash数值空间中，解决集群存储中，信息与存储服务器节点之间映射关系的问题。考虑到存储服务器较少或者服务器节点的异构性，引入虚节点的概念来弥补这种不均匀的分布。算法实现如图1.3,1.4所示。当系统需要动态扩展或节点故障时，进行数据的迁移。迁移是按照映射的hash空间，静态迁移。在使用Consistent hashing算法解决负载均衡分配时存在一些问题。(1)这种静态的数据分配算法，节点和数据是否能实现均匀分配，严重依赖于hash函数的选择[21]。(2)虚拟节点在保证一致性哈希算法的负载均衡上提供了帮助，避免了单一依靠存储节点的哈希值而导致的在环状结构上分配不均的问题。但是，如何根据系统的实际情况来决定虚拟节点的数量？虚拟节点的数量过少，则无法起到平衡负载均衡的作用；虚拟节点的数量过多，则意味着需要维护大量的虚拟节点和真实节点之间的映射关系，造成较大的查询开销[21]。(3)在现有的虚节点方案中，由于缺少对节点状态的反馈机制，如果存储节点的变化牵扯到了虚节点的分布，则被影响的虚节点需要重新哈希其上关联的key 值，来确定哪些key 值需要移动，而哪些不需要，造成比较大的计算开销[21]。

除过上述的采用Consistent hashing算法实现负载均衡方法外，还有很多关于负载均衡技术的学术研究。例如：Zeng et al提出了一种在分布式环境下解决负载均衡的方案，在这篇论文中，通过数据迁移确保负载再平衡时，存在以下问题：（1）没有考虑到各副本所在的存储节点。数据迁移完成后，可能导致多副本在同一节点上。降低了系统的可靠性（2）没有明确指定在迁移过程中多少数据可被迁移。如果迁移的数据量较大，可能导致被迁移节点负载超过阈值的情况。Prabavathy.B提出了一种基于私有云存储的负载均衡算法。该算法是通过计算每个存储节点的weigh实现数据的均衡存储。当由于系统的动态扩展，或者用户动态数据删除等原因导致负载不均衡时，使用rebalancing算法迁移数据，使节点达到负载的再平衡状态。但是，仍存在以下问题（1）在进行数据迁移时，以ideal load的值作为衡量各个节点负载的标准（ideal load=（Overall load of the cluster/overall capacity of the cluster) \* Capacity）。这种计算方式没有考虑带节点本身的异构性，将所有节点同等对待。（2）在进行数据迁移时只是以ideal load的值作为衡量节点负载轻重的标准，没有考虑到系统中负载整体过重的情况。如果在系统整体负载过重的情况下，迁移数据，只会增加系统带宽的负担。

由于上述实现负载均衡时所存在的问题。以及使用范围的局限性。本文介绍一种新的面向对象存储系统的负载均衡方法——动静态相结合的负载均衡算法。使用总控节点动态实时获取系统负载进行用户数据分布。并使用合理的迁移算法保证系统负载分配不均的问题。

Node1

Node1

Object1

Object1

Node3

Object4

3

Node3

Object4

Object3

Object3

Node2

Node2

Node4

Object2

Object2

**图1.3 一致性哈希算法 图1.4 添加节点示意图**

## 1.3 研究内容

本课题主要研究一种可扩展的OBS系统的数据分布策略。并与S3和swift对象分布策略进行对比，比较性能方面的变化。并着重研究OBS系统中负载均衡技术。研究内容如下：

1. 分布式存储系统结构的研究和介绍。以及本课题使用的系统架构。研究动态的非

对称三层系统架构的优势。

1. 对系统中的负载均衡技术进行研究和设计。分别从衡量节点负载指标，负载分

布策略，迁移阈值，迁移对象选择，可用性等多角度分析和设计负载均衡算法。使的算法能够满足系统对各存储节点间负载均衡分配的要求。

1. 研究OBS节点数据分布模型，分别从目录层次和每层目录容量以及可扩展性为角度，分析适合object扁平化的数据分布策略。
2. 测试用户读写文件的响应速度与单一目录下文件数量的关系。为实现数据扁平化分布找到理论支撑。
3. 搭建测试分布式系统，对系统中数据分布策略和负载均衡过程进行测试和仿真。

结果表明基于对象的FDDM数据分布策略，能有效的实现节点数据的扁平化分布。提高系统对数据访问的响应速率。动静态结合的负载均衡算法可以使分布式系统在任何情况下保持很好的系统平衡性能。

## 1.4本文结构

本文的组织结构安排如下：

第一章：分析对象存储相关背景，以及数据组织策略和负载均衡技术在OBS系统中的重要性。介绍对象存储分层结构和负载均衡技术国内外发展现状。并分析存在的一些问题。提出本研究的目的和意义。

第二章：介绍分布式系统的一些技术。例如集群存储模型，对象存储，负载均衡，快照等。这些技术是后文算法研究和实现所需要的理论依据。

第三章：介绍OBS系统的整体架构，架构中每个组件的主要功能，以及OBS系统架构的优势。

第四章：介绍OBS系统中数据分布策略FDDM和负载均衡技术的设计和在本系统的实现。

第五章：创建仿真环境，测试和分析系统对用户操作的反应速度与目录下文件数 量的关系。创建分布式仿真环境。使用LR测试工具。模拟用户并发操作。对系统中数据分布策略和负载均衡过程进行测试和仿真。

# 系统理论技术介绍

这一章节对后面系统架构和算法设计中涉及到的分布式存储系统的理论技术做简单的介绍（集群存储模型，对象存储，负载均衡，快照），后面文章中提到不在累述，可参考本章介绍。

## 2.1 集群存储技术

集群存储技术由于在数据共享、高可扩展、高I/O性能、高可用等方面的优势而日益受到学术界和产业界的广泛重视和应用。本节主要介绍两种集群存储形式[9]。

根据不同的原则，集群存储技术可以有不同的划分方式。从系统结构的角度出发，可以将集群存储划分为对称（symmetric）和非对称（asymmetric）两种形式。

* 对称结构：如图2.1所示。对称结构是指集群存储系统中的所有节点地位均等，每个节点都了解系统的整体结构，所有节点都理解磁盘结构，提供数据和元数据的访问。为了保证系统的正确性，各个节点之间元数据的管理和一致性的保证通过节点之间的通信和分布式锁机制完成。比如RedHat的Global File System（GFS）和IBM的General Parallel File System（GPFS）[9]。



**图2.1 集群存储对称结构图**

* 非对称结构：如图2.2所示。主要特点是，将数据与元数据分开存储。提供一个或多个专有的元数据服务器来维护文件系统和相关元数据结构。数据的访问通过元数据和存储节点完成。例如GFS，CFS等都是采用这种非对称的存储架构。



**图2.2 集群存储非对称结构图**

本系统采用的是非对称存储结构，这么做的目的是实现元数据与数据的分离。避免在对称结构中所有节点之间必须要相互通信来保持数据完整性的问题。而为保持节点数据完整性和一致性而使用分布式锁机制是很复杂的过程。从系统的扩展性和数据迁移角度考虑，如果使用symmetric架构，每一次系统的扩展和数据的迁移都需要和现有系统中的所有节点进行通信，从系统的响应时间和复杂度来说都是无法容忍的。

## 对象存储

非结构化数据量的不断增长，已经无法使用传统的关系数据库的存储模型进行数据存储。主要原因是，对于一个普通用户来说，最常见的的操作就是存储和读取数据。而关系型数据最擅长的查询（索引查询）在此已无用武之地，反而会增加系统的复杂性[3]。为了使用户对文件的操作尽量简单，高效，用户大量非结构化数据使用对象存储的方式。

对象存储是用户数据在存储节点（OSN）存储基本单元。参考S3对象存储模型架构的介绍。对象存储中主要涉及三个概念：对象（Object），键（Key），桶（Bucke） [4]。

（1）Object

Object是对象存储的基本单元，也是用户进行数据操作的基本单元。主要由两部分组成：数据（data）和（Metadata）。数据可以是任意类型，而元数据是描述数据的数据（比如：last-modified，size，user等）。在本系统中，metadata信息以Mysql数据表的形式存储在专有的一个或多个元数据节点。而data存储在对应的OSN节点。

1. Key

Key是每个数据必须具有的唯一标识，就像每个人拥有一个身份证号码一样，每个对象必须拥有一个唯一的key，否则没有意义。本系统采用统一的命名方式为每个对象生成一个唯一的key值，用于对象的标识。

（3）Bucket

Bucket是用来存储对象的一个容器，类似与用户的文件夹[4]。用户的所有数据以object的方式存储在bucket中。本系统以用户为单位进行数据的存放。即一个用户的所有数据存储在一个容器中。容器是以系统生成的userid的方式进行命名。

## 负载均衡

负载均衡主要有两方面的含义：首先，是指把大量的数据流量或者并发访问均匀分布到多台的节点设备上分别处理，从而减少用户的等待时间，提高资源的利用率。其次，是指将单个重负载的运算分发到多台节点设备上进行并行处理，每个节点处理完成后，将处理结果返回给总控节点，合并后返回给客户端。这种方式可以大大提高系统的处理能力[10]。本课题研究的是面向对象的存储系统。数据对象的存储是系统的核心，系统中不存在并行处理的问题。主要目的是将各用户数据均匀分配到各个对象存储节点（OSN），实现负载的均衡分配。所以本系统中主要关心的是存储资源和数据分布。负载均衡指的是第一个含义。

存储系统负载均衡策略一般由两部分组成：首先，是数据的分布策略。数据分布策略是指按照一定的算法将数据均匀分布到各OSN分别处理。但是当系统动态扩展，节点下线，用户数据动态删除等操作引起OSN节点负载分配不均衡时。可以使用一种负载迁移的算法进行状态信息平衡。迁移算法包含四种策略。如图2.3所示：

转移策略

定位策略

选择策略

信息策略

控制信息

迁移数据

本机 远程机

**图2.3 典型的迁移算法组成**

（1）信息策略：收集系统各节点的负载信息，决定收集哪些信息，如何采集，以及这些负载信息由一个总控节点收集，还时任何节点都能够接收，这些问题在系统架构一章详细介绍[11]。

（2）转移策略：是指系统在什么情况下发起负载均衡操作，这个问题就是负载均衡实现中涉及的阈值模型和驱动策略。

1. 选择策略：根据一定的策略，决定执行哪个任务。（迁移节点，迁移数据）

（4）定位策略：决定把选择的数据转移到哪个节点上进行执行。

## 2.4 快照

快照实际上是一种摄影技术，随着存储应用的提高，用户需要在线的方式进行数据的保护， 快照技术是在线的存储设备预防数据丢失的一种有效手段。快照有三种基本形式，基于文件系统的，基于子系统的和基于卷管理器／虚拟化式的[2]。

快照技术类型：

主要有两大类型的存储快照，分别是即写即拷（copy-on-write）快照和分割镜像快照[2]。

即写拷快照是用来表现数据外观的特征。这种快照方式也常常被称作“元数据”拷贝，即并不是对所有的数据进行真正的拷贝，只是保存数据位置指针。在使用这项技术时，当已经存在快照，而有人试图修改原始数据时，快照软件会首先将原始的数据快拷贝到一个新位置。然后进行写操作[2]。即写拷贝快照的方式优势就是速度特别快。而且不需要占多少空间。本系统在进行数据迁移时就使用的是即写拷快照，主要的目的是为了在进行数据迁移时，既能够保证数据的一致性，又能够取得不停机的系统服务。

分割镜像快照主要的作用是提供离线访问数据的能力。分割镜像也叫做原样复制。它是对某一个系统的物理拷贝。因此这个过程一般比较慢，而且每个都需要占用更多的存储空间。

本系统中使用快照主要是实现数据迁移不停机的目。

# 第三章 OBS系统的架构设计

## 3.1 整体架构

OBS系统采用与监控系统相结合的非对称分布式系统架构，使用非对称的分布式系统架构，避免了对称架构中要求各个节点之间进行数据同步而保持一致性的问题。

OBS系统划分为三层：总控节点（RootServer），元数据管理层（MetaServer），文件服务层（FileServer）。而OBS系统与监控系统的结合。就像为分布式系统安装了一双眼睛，实时了解系统的工作状态。通过有效合理的监控项配置，实时了解系统当前负载，节点工作状态。所以，OBS系统是一个真正意义上的基于节点动态反馈的分布式系统。

* RootServer

总控节点提供对整个OBS系统的节点配置、安装、以及各个节点状态的监控报警、系统负载迁移参数配置，数据和服务迁移控制、数据节点存储模型动态配置等。

* MetaServer

MetaServer是OBS结构模型层次中的元数据管理层。以数据库表格的形式存储FileServer、用户注册、用户权限、用户文件相关的元数据信息。与用户注册、权限验证、数据节点分配、文件操作、数据迁移等相关的操作都是通过元数据信息来实现管理。MetaServer使用虚拟化存储技术实现全局统一的命名空间，对文件进行统一的命名，进行集中的管理。

* FileServer

文件服务层是OBS结构模型中的数据存储层。用于存储用户的真实数据、实现用户文件的多版本存储、提供文件的多副本存储、按照目录结构分布算法实现数据在节点上扁平化分布。

根据系统需求，使用并行的，有效的数据分布策略，提高系统的响应效率，减少用户等待时间。提高系统可用性。

* Client

用户使用OBS系统与使用一般的云存储系统相同。系统提供开放的接口，以web页面的方式提供访问接口供用户使用。客户端使用系统时，先通过注册接口进行用户信息的注册，待系统通过验证后方可使用，系统提供的接口有List, Put , Post, Delete,share。

OBS系统架构图如图3.1所示：图中①表示RootServer对整个集群存储系统中的MetaServer、FileServer进行管理，配置，安装和动态监控。图中②表示各个节点向MetaServer推送节点的负载情况。如CPU，mermory，disk，fd等。③是client与MetaServer的交互，表示客户端将对文件的操作请求发送给MetaServer。④表示MetaServer将用户对文件的操作请求发送给FileServe进行数据处理。⑤表示FileServer将相关的处理结果返回给MetaServer，然后MetaSever进行元数据的处理。过程⑥为MetaServer将处理结果返回到客户端。

⑥

RootServer

④

③

①

②

①

①

⑥

MetaServer

①

⑤

④

②

②

④

⑤

FileServer

FileServer

FileServer

**图3.1 OBS系统架构图**

## 3.2 对象存储系统各层次组件设计

## 3.2.1 总控节点

RootServer功能组件图如图3.2所示：

RootServer

帐号管理

迁移控制

节点监控

服务器安装

服务器配置

**图3.2 RootServer功能组件**

RootServer的功能主要包括：集群节点配置、OBS系统监控、数据迁移控制。

RootServer控制集群中所有MetaServer、FileServer的配置和安装。主要的功能为：配置RootServer管理服务器管理员的信息、配置并且显示MetaServer和FileServer基本信息。例如：节点容量，异构性权值，节点IP。由管理员的配置完成对远程服务器软件包的安装。以及监控客户端的安装，配置和启动。RootServer与MetaServer和FileServer之间实时保持心跳，从而确保各个节点的工作状态，对下线的节点，及时切换服务换到副本服务器。并等待一段时间后如果下线的节点恢复，进行数据的补齐并将服务切换到该节点。如果下线节点未恢复则认为永久故障，删除故障节点，并进行数据的迁移。

在RootServer节点中嵌入与分布式系统相关的监控服务。安装zabbix server监控，并将监控服务器集成到RootServer。并对Zabbix Server进行二次开发，使其支持主动监控的模式。详细的与监控有关的工作见3.2.3中介绍。

RootServer节点设置数据迁移策略，配置迁移阈值，控制OBS系统数据迁移的过程。并且提供手动和自动两种迁移方式，数据迁移过程完全可控。避免访问高峰期，无用数据的迁移。以免不合理的迁移，不仅没有带来资源优化和效率上的提升，反而占用带宽，增大访问延迟。

## 3.2.2 元数据服务器

MetaServer的功能如图3.3所示：

MetaServer

迁移数据选择

文件多副本设计

用户数据节点分配

监控数据管理

用户文件管理

用户管理

**图3.3 MetaServer的功能**

MetaServer的功能主要包括：存储多个数据表、用户权限认证、创建统一的命名空间、服务器节点分配、用户数据操作。

MetaServer维护了系统的元数据信息，包括用户与主副FileServer、文件与命名空间，文件到存储路径之间的映射。其它的结构化数据信息如：用户帐号信息（包括新注册用户，通过验证可使用系统的用户），用于进行用户权限的验证。用户的管理采用授权的方式，不同的用户具有不同权限。用户文件信息（包括：文件名称，文件多版本管理，文件大小，类型，创建时间等与文件相关的元数据信息）。存储缓存文件，快速定位用户文件。与负载均衡相关的数据迁移策略，如迁移队列，迁移记录的信息。3.2.1节中监控服务器采集到的监控信息（例如：系统延迟，吞吐，网络，负载等信息）也是存储在MetaServer。

OBS系统是一个对象存储系统，用户数据是以对象的方式存储在FileServer。MetaServer会为用户的每个对象创建一个唯一的ID值，称为OID。在本系统中，OID使用md5(uniqid(rand()))方法生成。uniqid()函数：以微秒计的当前时间，生成一个ID。使用MD5方式加密，产生 32 位的独一无二字符串。保证了OID的唯一性。

服务器节点分配，OBS系统是以用户为单位存储数据，即将同一用户的所有数据存储在同一主FileServer，当用户首次上传数据时需要进行主副本存储节点分配。在MetaServer设置静态策略，详细策略见负载均衡静态策略描述。实现用户存储节点的分配，使各服务器均衡获取到用户的请求。

用户数据操作是指MetaServer接收并解析用户的请求，用户的请求包括创建目录、上传，下载，删除文件、以及音视频的在线播放。MetaServer解析请求，并将操作转发到具体的FileServer。

## 3.2.3 文件服务器

FileServer的主要功能包括：设置文件目录策略、用户文件操作、文件迁移功能、文件双副本功能。FileServer的功能如图3.4所示：

FileServer

文件多副本设计

用户文件操作

文件上传与下载

文件多版本

迁移数据选择

删除文件或目录

新建目录

**图3.4 FileServer的功能组件图**

针对现有对象存储系统中数据在存储节点上目录结构分布不均匀的问题，如对象存储中研究现状所述。OBS系统提出了基于对象存储的数据分布策略——FDDM（Flat Data Distribution Method）。FDDM策略是基于OID分段式的数据分布方法。是对已有对象数据分布方法的扩展和补充。目录层次的划分粒度更加细化，结合了文件系统目录分布和对象存储扁平化分布的优点。FDDM策略可以根据用户的数量级，动态配置目录层次和每层中目录容量。策略的实现不受限于用户数据量。任何数量级下FDDM策略都能以最快的速度达到数据的扁平化分布。FDDM目录层次结构示意图如图3.5所示。FDDMdirectory实现和分析过程见第四章介绍。

devs

object

object

userid

目录结构根据用户分

数量级动态划

FDDMdirectory

object

objectData

**图3.5 object在 FDDM策略下目录结构**

附注1：

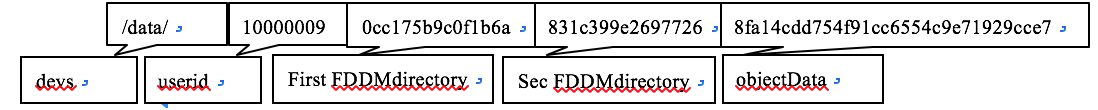
devs：设备挂载的路径

Userid：用户的文件夹，用户文件夹以userid命名

FDDMdirectory：是FDDM策略的核心。

Object：最终要保存的数据

例如：某object的存储路径如图3.6所示，假设目录层次分为两层。每层目录层次表示。

****

**图3.6 对象目录层次示例图**

用户文件操作：包括用户上传，下载，删除文件、以及音视频文件的在线播放等。文件的真实数据都是存储在FileServer。用户将请求发送的MetaServer，MetaServer解析用户的请求，并找到用户的存储节点，即FileServer后，将请求转发到FileServer进行对数据的操作。

文件迁移操作：迁移操作的策略由RootServer控制。真正数据迁移的过程在FileServer上进行。

文件复制：是指一个文件最少要有两个完整的副本来保障当一个节点故障时，仍然能保证系统服务。

## 3.2.4 客户端

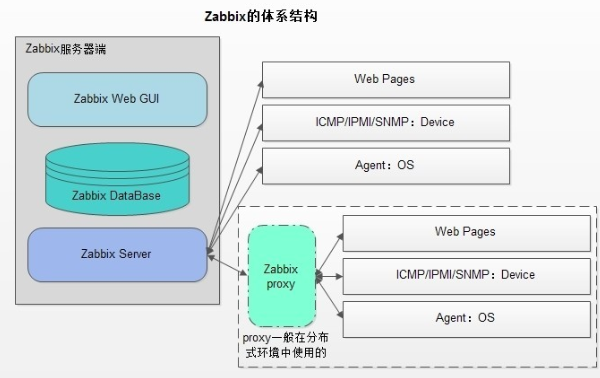
Client是OBS系统提供给用户的访问接口。客户端操作文件时，首先访问MetaServer，获取与之交互的FS信息，然后访问FS，完成对数据的相关操作。客户端不缓存文件的数据，但是可以缓存MetaServer中获取的元数据。

## 3.2.4 监控系统组件

监控系统在OBS系统的主要任务有：采集MetaServer，FileServer的负载信息（包括CPU Used/Idel，memory Used/Idel，Disk Used/Avail,Fd等）、使用心跳包方式确保OBS系统各节点可用、监控MetaServer上Mysql，web服务运行状态。配置报警规则，系统异常时进行报警、绘制监控图供管理员查看。

本系统采用的是在现有的zabbix监控系统上进行二次开发，。zabbix系统结构如图3.7所示：zabbix web和zabbix Server集成到RootServer中，web页面用于监控项的配置和系统状态的查看。Zabbix Server组件用于监控项的收集。Zabbix DataBase使用mysql数据库保存历史数据。并提供开放的查询监控数据的API，MetaServer可以通过开放API查询FileServer负载信息。本系统将Zabbix DataBase安装到MetaServer，这样做的好处是减少RootServer的工作负载，避免RootServer任务过重而成为系统瓶颈。

在RootSerever配置和安装MetaServer和FileServer时，自动安装zabbix agent，并修改agent端配置。启动服务。在zabbix Server端设置自动发现规则。Server端会主动发现agent节点，添加对节点的监控。并根据设置的策略自动添加监控项。整个过程完全自动进行，无需管理员介入。



**图3.7 zabbix系统架构图**

监控系统的工作过程如图3.8所示：其中①③表示zabbix server从FileServe和MetaServer采集监控数据。例如心跳信息，服务器负载，mysql/web服务运行状态，重要文件修改信息等。②表示agent端主动向Server端推送监控数据。④表示将采集到的数据存储到zabbix DataBase，即MetaServer节点。⑤过程表示server端通过API从Database中读取历史数据用于frontweb页面的展示，以及各个agent节点负载进行迁移控制。⑥过程为用户请求存储节点分配时，请求先到MetaServer，MetaServer根据数据库中负载信息进行资源的分配。

RootServer

Zabbix Server/

zabbix frontweb

③⑤

MetaServer

④

②

Zabbix agent/

zabbix DataBase

①

⑥

File

Zabbix agent

FileServer

FileServer

Zabbix agent

FileServe资源池

**图3.8 监控系统工作过程逻辑图**

## 3.3 本章总结

前面3.1和3.2节对系统的架构做了详细的介绍，OBS系统采用的是三层非对称的分布式系统架构。采用这种架构的优势是使用专用的元数据维护文件系统，数据的访问直接在节点之间完成。从而避免了对称结构中元数据的一致性管理问题。提高了系统的并发性。采用三层而不是两层的分级结构。使的与监控，迁移，节点数据分层控制的业务逻辑与存储分离的方式。主要目的是降低单节点的工作负载，避免由于单节点负载过重而成为系统瓶颈。OBS系统与监控系统的结合，使系统成为一个动态可控的反馈系统。使的管理员可以根据监控到系统详情，对系统进行动态的调整。在保证系统可靠性与可用性的前提下，有效资源利用，最小化响应时间。

OBS系统使用以用户为单位的数据存储模型，即每个用户的数据全部存储在用户文件夹下。这种数据存储模型由很多的优势，比如：使的数据的迁移，存储，以及用户数据的访问效率都大大提高。由于不同用户对服务质量的不同要求，可以智能化的为用户分配存储节点，真正做的按需分配。按需不止是容量上的按需，还包括服务质量上的按需分配。这种数据存储方式尤其在私有云环境中可以得到更高效的运用。这种存储方式的另一个优点是可以根据估算的用户数据量或者在RootServer节点监控到的数据分布的变化，调整某个用户数据分布参数。即上文提到的FDDMdirectory。用于实现用户数据在存储节点上的扁平化分布。FDDM策略详细内容以及实现的必要性的测试在第四和第五章节中详细介绍。

# 第四章 OBS系统负载均衡和对象存储策略设计与实现

## 4.1 负载均衡算法设计

OBS系统中由RootServer根据全局负载信息进行负载分配和调整。采用动静态相结合的负载均衡算法来解决云存储环境中负载分配和负载分配不均动态调整的过程。OBS系统是以用户为单位，进行主/副本服务器分配。因此在负载均衡算法中，数据节点的分配和数据迁移都是以用户为基本单位进行。

静态负载均衡是指当用户请求上传文件时，根据RootServer从各个节点收集到负载信息，计算出节点的weigh值。存储到MetaServer，由各节点weigh值为用户分配合适的服务器节点存储数据。也就是找当前负载最轻的服务器作为用户的存储节点。动态负载均衡是指当系统中各节点负载分配不均匀，采用动态的数据迁移策略，进行数据迁移，完成负载的再平衡过程。

## 4.1.1 静态负载均衡算法设计

静态负载均衡算法是指当用户请求上传数据时，找出系统中存储节点DNweigh值最大节点作为存储服务器。其中DNweigh值表示系统中各节点存储资源可用率。DNweigh值越大，表示节点的可用存储空间越大。DNweigh值计算方法如公式（4.1）所示：

DNweigh=Status\*A\*DNavai/DNTotal （4.1）

各存储节点定时向控制节点反馈其工作状态，确保节点正常工作。并向控制节点反馈该节点当前可用空间，总空间容量信息，以及异构性权值。可用空间，总空间容量信息用于更新节点的DNweigh值。如公式（4.1）所示。其中Status表示节点的状态，节点失效时，设置Status==0，节点正常工作，Status==1。A表示各存储节点的异构性权值。引入异构性权重的原因是存在各存储节点服务器处理能力的不同，那么允许在处理能力高的服务器上存储较多的数据。权值A由管理员在加入服务器节点时根据服务器配置动态设置，默认为1。DNavai表示各节点当前可用存储空间，DNTotal表示各节点总存储空间。当用户请求上传数据时，DNweigh值最大的节点作为存储节点，副本依次选择下一个。使用这种静态算法目的是保证负载的均衡分配。使各个服务器节点可以均衡的获得用户的请求。其实现过程如算法4.1所示。

算法4.1：

Input：A

1. FileServer定时向RootServer反馈其Status， DNavai、DNTotal的的值
2. 配置组合监控项，按照公式（4.1）计算各节点DNweigh并存储到MetaServer
3. 用户首次上传数据请求发送到MetaServer
4. 确定需要存储的副本数n
5. 按照DNweigh值降序依次返回n个服务器节点信息

## 4.1.2 动态负载均衡算法设计

静态负载策略在一定程度上保证各文件服务器都能够均衡的获得用户的请求，平衡负载的分配。但是当系统动态扩展、或者用户动态删除数据时。将会引起各节点上负载的不均衡分配。在这种状态下，需要一种再平衡算法。实现负载的动态平衡。本文设计了一种动态负载迁移算法。

RootServer对服务器的负载情况进行监控。各个存储节点的存储空间状态实时向RootServer进行反馈。通过各节点反馈的值计算系统整体的负载weigh值为DNImt，引用见式（4.2），并根据各节点负载情况计算各节点的负载weigh值为DNmt，引用见式（4.3）。管理员根据实际应用情况设定DNoverload值。DNoverload值作为衡量整个集群负载是否过重的标准参数，首先系统使用DNmt和DNImt进行比较，如果DNmt>=DNImt，说明系统整体负载较重，通知管理员请求动态添加服务器。添加服务器完成后，按照迁移算法进行数据迁移。如果DNmt<DNImt ,找出系统中负载最重的节点的负载weigh值，与迁移阈值进行比较（系统迁移阈值有两种方式，一种是系统根据负载状态计算得到的DNImt，一种管理员根据需求设置），如果符合迁移条件，则找到负载weigh值最轻的节点。计算出最大可迁移的数据量并且被迁移数据副本不能在目标服务器上。找到符合迁移条件的数据进行负载的迁移。系统整个过程对于用户来说都是透明的。

DNImt=DNTotalLoad/DNtotalcapacity  （4.2）

附注2:

DNImt系统默认迁移阈值和系统整体负载weigh

DNTotalLoad表示系统总负载

DNtotalcapacity表示系统总容量

DNmt=DNload/(DNcapacity\*A) （4.3）

附注3:

DNmt表示各存储节点的负载weigh，DNmt的值也可通过管理员手动设置得到

DNload表示各存储节点负载值

DNcapacity表示各存储节点的容量

A表示异构权值。

实施迁移算法要明确如下问题：

（1） 何时进行数据的迁移

采用阈值策略：由反馈系统计算得到或由管理员根据系统性能状态设置阈值迁移DNImt，管理员根据反馈得到的系统网络状态，带宽，qps，平响等综合因素，设置系统整体最大阈值DNoverload。是否具有符合条件的迁移数据由两个阈值DNImt和DNoverload共同确定。并且值最大DNmt节点大于DNImt，并且存在满足要求的目标节点和可迁移数据时，触发迁移算法迁移数据。当DNImt>=DNoverload时，说明系统整体负载较重。提醒管理员动态添加存储节点。

（2）目标服务器，源服务器，迁移数据选择原则

在迁移操作中有三个重要角色：SourceServer(迁移源服务器)，TargetServer(迁移目标服务器)，Migrationdata(迁移数据)。当DNImt<Noverload时，选择SourceServer==IP（Max（DNmt））（负载最重的节点的IP）。为了减少迁移次数，最大程度保证系统高效性。目标服务器选择负载最轻的服务器，即TargetServer==IP（Min（DNmt））。计算可迁移数据量，选择出可迁移数据，可迁移数据选择目标是尽量使两个节点负载量接近，又不导致源服务器的负载超过DNImt。可迁移数据选择过程如算法4.2所示。为保证系统的可靠性，迁移数据副本不能是目标服务器。

算法4.2：

Input：SourceServer、TargetServer、DNImt、Max(DNmt)、Min(DNmt)

Output：MigrationData

①AverageCount ==(Max(DNmt)+Min(DNmt))/2

②if DNImt <= AverageCount,Then AverageCount = （DNImt -Min(DNmt)-10%）

Next

③ MigrationCount == (AverageCount -Min(DNmt))\* DNcapacity\*A

④Data is sorted based on their size in descending order on the target server .Storage as collection M(a,b,c,d,e,f......).

⑤Remove the data from collection M which duplicate on the SourceServer. Storage as collection M(a,b,c,d,e,f......)

⑥If Count(M)>1,Then Next;else next.

⑦ MigrationCount <= Max(size(a+b+c+...))

⑧ MigrationData=(a,b,c......)

（3）策略进行的前提条件

为了保证算法的正确性，即使存在存储节点的负载超过迁移阈值，但存在下述情况，仍不进行数据的迁移。

* 系统整体负载较重，即DNImt>=DNoverload，不需要迁移。
* 源服务器上的任何数据都会使目标服务器达到负载上限时，默认不迁移。
* 如果源服务器上只有一个数据，即Countdata（SourceServer）==1，默认不迁移。

（4）如何保证数据迁移正确性

为保证负载迁移的正常进行，算法保证系统同一时刻只能参与一个负载迁移过程。其它迁移过程在队列中等待。当migrate==1时，表示存在节点迁移数据，migrate==0，表示系统中不存在节点迁移数据。每次只进行一个迁移过程好处一个是保证算法正确性。另一个是控制迁移的节奏。比如，一台新的服务器刚上线，负载最低，如果RootServer将大量的数据同时迁移到这台新的服务器。那么整个系统在新增机器的时候服务能力就会迅速下降。因此每一时刻只能进行一个迁移是为了使负载均衡做的比较平滑。

迁移算法整个工作流程如算法4.3所示。

算法4.3：

Input：DNImt 、DNmt、DNoverload、migrate

Output：SourceServer、TargetServer、Migrationdata

1. if migrate==0,Then,Next; Else,exit.
2. if DNImt>=DNoverload，Then，Next;else，Jump to step④.
3. send related e-mail to the administrator and then exit the program.
4. if Max（DNmt）>> DNImt，Then SourceServer=IP(Max(DNmt)) and next

step;else,exit.

1. TargetServer==IP(Min(DNmt)), And the next step.
2. if Countdata（SourceServer）==1,Then jump to step③;else，next

step.

1. Compute Migrationdata,And the step is shown as algorithm 2.
2. migrate the Migrationdata from TargetServer to SourceServer.

（5）迁移过程中系统可用性保证

数据迁移的过程中有短暂的时间需要停服务，为了尽可能的减少停机服务的时间，OBS系统预采用两次copy-on-write快照的策略。具体的操作如下：

* 原有MetaServer和FileServer对数据进行一次copy-on-write快照操作，第一次copy-on-write快照操作过程中仍然允许写操作。
* 停止写服务，对MetaServer和FileServer上的数据进行一次copy-on-write快照操作。由于第一次copy-on-write 快照过程中写入的数据一般比较少，第二次copy-on-write 快照的时间会更短。

分析：copy-on-write 快照操作只需要保存当前数据所在的指针，因此，一般速度都会很快。而且进行两次copy-on-write 快照操作，停机操作是在第二次进行copy-on-write 快照时，第一次操作的时间本来就很快，所以两次操作之间变化的数据量更少，第二次copy-on-write 快照的时间会更快。停机时间会更短。所以对于用户来说，根本感知不到停机。保证了数据在迁移时系统的可用性。

## 4.2 对象存储系统中负载均衡算法实现

## 4.2.1 迁移算法工作流程

OBS系统架构中实现动态负载均衡，整个迁移过程中RootServer,MetaServer, SourceServer、TargetServer之间的工作流程如图4.1所示：

MetaServer

SourceServer

TargetServer

文件传完后返回

manageserver

携带着验证信息

文件传输

携带着所要上传

文件的信息

RootServer

确定需要迁移的originserver，targetserver的ip,

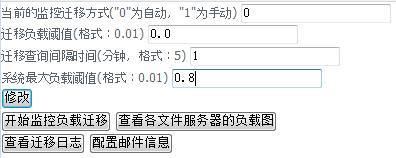
汇报迁移完成的消息

**图4.1 负载迁移工作图**

确定SourceServer上需要迁移文件的用户

## 4.2.2 数据迁移配置界面

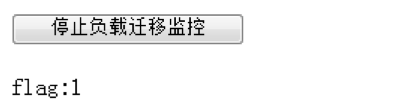
参数设置界面主要是指由管理员设置的与数据迁移相关的参数，界面如图4.2所示：



**图4.2 迁移参数设置界面**

其中：“当前监控方式”中‘自动’是指迁移整个过程自动进行。不需要管理员的参与。只是将迁移出错或者完成的信息以邮件的方式发送给管理员。迁移方式设置为 ‘手动’时，管理员点击“开始监控负载迁移页面”。点击后页面如图4.3所示。系统设置定时自动刷新图4.3所示的页面，当存在符合迁移条件的服务器，页面会弹出提示，提示管理员进行数据迁移，迁移整个过程中出现的问题都会以网页的形式，提醒管理员是否继续迁移。

设置两种方法的好处是：一般采用“自动迁移”方式，自动迁移过程中，对一些意义不大的可迁移项默认迁移数据。比如：源数据服务器上只存在一个用户，自动迁移时默认不迁移；采用‘手动迁移’，系统会提醒管理员，但是否迁移由管理员自己决定。



**图4.3 迁移监控页面**

迁移负载阈值：即迁移算法中介绍的DNImt值，当存在DMmt>DNImt时，触发迁移算法。

迁移查询间隔时间：是指检查是否执行迁移算法间隔时间。

系统最小负载阈值：即迁移策略中由管理员设置的DNoverload与计算得到的DMmt比较，确定系统是否整体阈值较重。

修改按钮：修改管理员设置的参数值。

查看各文件的负载图：观察各个节点的负载值，实验测试时会查看负载变化。

查看迁移日志：系统会将每次迁移信息保存在数据库中，如图4.4所示：



**图4.4 迁移日志页面**

## 4.3 面向存储系统扁平化数据分布策略

从第一章对象存储模型研究背景中可以看出现有的对象存储的方法，在对象存储过程中存在一定的缺陷。例如S3存储方法，用户数据量过多时会导致单一目录下数据量过多，swift在S3的存储模型上进行了一定的扩展和改进，不过仍然是一个静态的数据分布方法。根据数据量的多不同，会存在目录散乱或者过于集中的问题。从系统性能的相关文献中可以发现，当单一目录下文件过多时，对文件操作的响应时间明显变慢，本文针对系统中单一目录下文件数量是否影响系统的响应时间做了测试，详细测试和测试结果见第五章，从而证明本文提出的扁平化数据策略（FDDM）有重要的实现意义，FDDM基于动态OID分段的数据分布方法，实现数据的扁平化分布，在一定程度上加快系统的响应时间。第三章系统架构介绍中已经对FDDM策略的实现方法做了简单的介绍。本节针对FDDMdirectory的实现和分析进行详细的介绍。

## 4.3.1 扁平化数据分布策略设计

FDDM策略通过划分OID，组织目录层次。从划分的每一部分，按照需求，取出相应几位值。作为对应层目录。为保证存储系统安全性，取出的每一部分值经R算法（例如MD5加密）处理，作为每一层目录名称。

使用OID划分目录层次。要解决的问题如下：

* 目录层次（将OID划分为几部分）
* 每层目录最大容量（按照排列组合原理。从每一部分取出的位数决定了每层目录容量）
* 目录命名形式（取出的每一部分经过处理作为该层目录）

首先研究目录层次与用户数据量的关系。我们知道。当父目录下子目录或文件数过多过于集中时，将严重影响用户对数据的存取效率。当目录层次和目录容量过多，而用户数量级较少时。数据将会分布过于分散，浪费存储空间。因此，合理的目录层次和目录容量决定了每一层目录下子目录以及最后一层对象数据量的分布情况。

在使用FDDM策略设计对象存储层次结构时，根据文件系统的类别以及用户的数量级，合理组织目录层次和目录容量，（本系统通过监控系统查看目录分布动态配置目录结构）提高系统的访问性能。

FDDM策略实现：

设目录层次为n，从划分的OID的每一部分取出m位值。生成目录名的算法为L。目录创建过程如下：

① 取出OID，将OID平均划分为n部分；执行下一步。

②取出划分好的每一部分的前m位，保存在数组P[]中，执行下一步。

③ 取出数组p[]中的所有元素，使用L算法处理后保存到数组P[]中，执行下一 步。

④ A=pop(p[])为null？否，A作为目录名，执行下一步。是，执行⑧

⑤元素A为取出的第一个元素？是，用户文件夹作为当前目录B，执行下一步；

否，直接执行下一步。

⑥ B下是否已存在子目录A？否，B下创建子目录A。子目录A作为当前目录B。 执行下一步。是，子目录A作为当前目录B。跳转到④

⑦ 指针指向元素为空？否，取出P[]中指针指向的当前元素A，执行⑥

是，执行（8）

⑧将对象保存在当前目录B下，结束。

FDDM策略实现过程如图4.5所示（i＝0时当前目录为用户文件夹B）：

开始

否

是

否

是

当前目录B下存在子目录A

否

对象ID划分为n部分

A=L(P[i])，i++

取出每一部分中前m位值；存入数组P[]中

子目录A作为当前子目录B

是

当前目录B下存放对象

i=0

i>=n-1

当前目录B下创建子目录A，子目录A作为当前目录B

A=L(P[i])，i++

当前的目录A作为当前目录

i>=n-1

结束

**图4.5 FDDM策略流程图**

## 4.3.2 扁平化数据分布策略实现

FDDM策略已经在本系统中得以实现和应用，并且通过RootServer监控系统对目录分布的监控，对象在系统中实现了扁平化的分布。

在OBS系统中，OID使用md5(uniqid(rand()))方法生成。uniqid() 函数：以微秒计的当前时间，生成OID。使用MD5函数加密，产生 32 位的独一无二字符串。保证了OID的唯一性。管理员配置目录层次为两层，每层取出一位OID，通过Md5加密生成目录名称。第三层存放对象。

示例：用户上传数据时，先获得一个唯一的32位OID，将获得的OID值以16位为单位划分为两部分，依次取出每一部分的第一位值，保存在数组P[]中，数组P[]中的每一个元素使用MD5()进行加密。MD5加密后的这两个32位字符串作为对象存储的两层目录，存储数据时，先找到用户的存储文件夹，然后在相应层判断这两层目录是否已经创建。如果未创建，则创建。否则不做任何操作继续判断下一层目录，直到两层全部判断完成后。将对象存储在最后一层目录下。

采用上述算法实现数据分布示例：

算法实现举例：假设用户之前未上传过数据。

用户1,上传数据A和数据B，假设：

A的OID为：d5c12521a79bb79c22debca347dbbe00

B的OID为：d579091c5a880faf6fb5e6087eb1b2dc

用户文价夹为：100000001

上传数据A

1. OID划分为两部分[d5c12521a79bb79c；22debca347dbbe00]
2. 取每一部分的前一位[d，2]
3. MD5加密[md5(d),md5(2)]
4. 由于没上传文件，所以目录肯定不存在，逐层创建目录，生成的目录如下：

100000001/md5(d)/md5(2)

1. 上传数据，整个文件路径

100000001/md5(d)/md5(2)/A的OID

上传数据B，同A过程相同，取前一位，加密的目录名

[md5(d),md5(6)]

创建目录；

1. 发现要创建的第一层目录md5(d)已创建，不做任何操作，
2. 判断md5(d)目录下是否存在子目录md5(6)？否，创建该目录；子目录MD5（6） 不存在.
3. 生成的目录结构为

100000001/md5(d)/md5(6)/

1. 上传文件

## 第五章 实验及结果分析

## 5.1 分布式存储系统仿真环境

## 5.1.1 节点配置

RootServer,MetaServer,FileServer配置

OS：SUSE Linux Enterprise Server 11 SP1 (i586)

CPU：Intel(R) Xeon(R) CPU E5606 @ 2.13GHz，

DISK:9.2G

内存：512M

应用服务器：Apache/2.2.10 (Linux/SUSE)

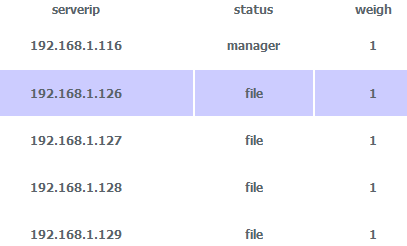
数据库：mysql 5.0.67

PHP：version 5.5.16 。

测试工具：HP LoadRunner

## 5.1.2 分布式存储系统环境配置

MetaServer和FileServer配置如图5.1所示：



**图5.1 OBS集群配置**

附注4：

serverip：节点的IP值

status：file表示为FileServer,manager为MetaServer

weigh：服务器异构性权值，由于各个节点配置相同，weigh默认都为1

## 5.1负载均衡测试结果及分析

本节将负载均衡技术应用到OBS系统，并使用HP LoadRunner测试工具， 模拟用户的并发操作，并模拟特定的负载场景。通过系统的负载图观察各个节点负载变化，测试负载均衡技术的工作状态。

## 5.2.1静态负载均衡算法测试

1. 使用单元测试脚本生成100个不同的用户，注册，验证通过后。并发请求系 统分配服务器。
2. 使用LR模拟多用户并发请求分配服务器，测试静态负载均衡算法工作的正

确性。

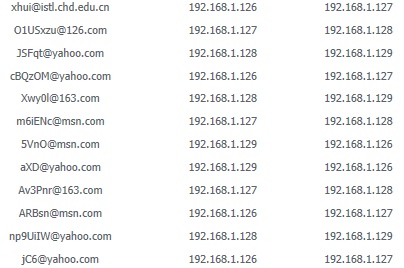
为验证算法正确性，设置存在服务器下线的情况，使用LR模拟多个用户并发请求分配服务器过程，测试过程如下：

添加新的服务器192.168.1.115。现在模拟服务器宕机情况，将ip为192.168.1.115的服务器关闭：现服务器工作状态如图5.2所示。



**图5.2 数据服务器状态图**

模拟完成后，用户分配到的主副存储节点如图5.3所示：



**图5.3 用户与主副本服务器信息**

附注5：

第一列：用户的账号

第二列：用户主服务器

第三列：用户副本服务器

（3）结果分析

由于当前系统是新配置的集群环境，各存储节点的负载均为0。此时，当用户并发请求分配服务器时。系统按照轮询算法为用户分配存储节点，从而保证了各文件服务器能够平衡的获得用户的请求。而为用户分配存储节点时，宕机的节点并未包含在内。保证了系统的可用性与可靠性。

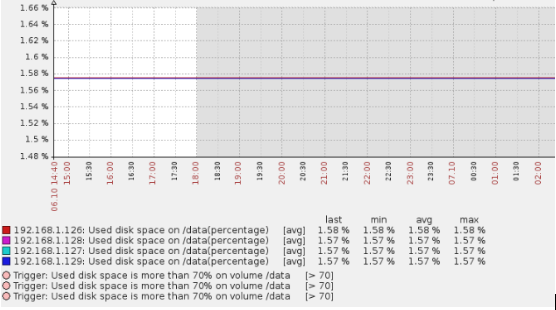
## 5.2.1动态负载均衡算法测试及结果分析

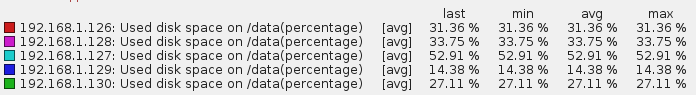
（1）手动迁移过程与结果分析

为更清楚验证迁移算法，我们可以大幅度增加某一服务器节点负载，并通过手动

设置迁移阈值的方式，控制迁移过程。通过对比迁移前后负载图观察负载变化。

系统初始状态的负载图如图5.4所示：





**图5.4 服务器初始状态负载图**

附注6：

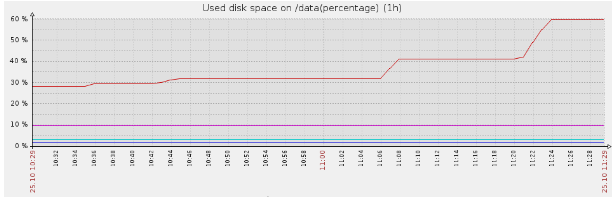
横坐标：时间

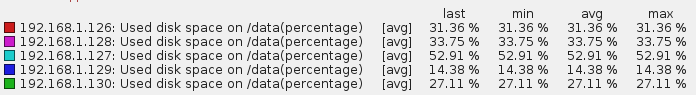
纵坐标：服务器负载百分比。各服务器负载变化由不同的颜色区分

场景一：

使用LR模拟用户上传数据，创造符合迁移条件的负载状况。

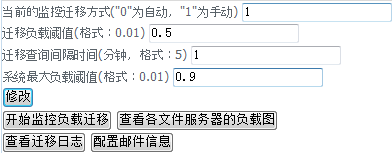
系统负载图如图5.5所示：





**图5.5 迁移前负载图**

管理员设置的迁移参数如图5.6所示：



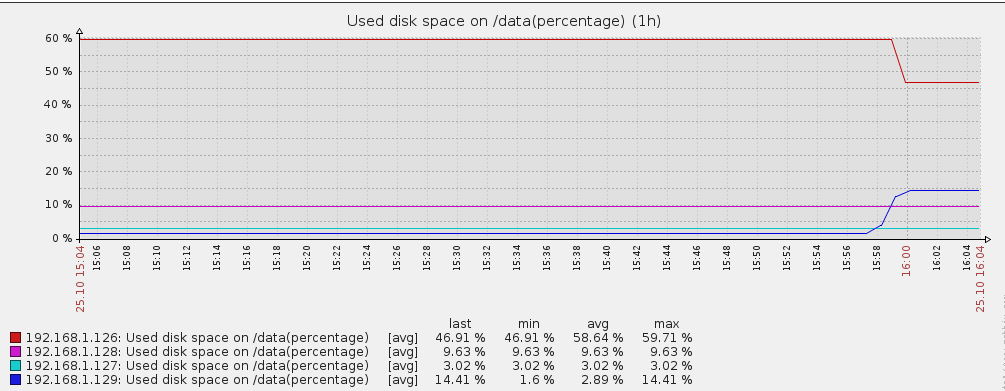
**图5.6 迁移参数页面**

点击“开始监控负载迁移”按钮，弹出如图5.7所示提示信息：



**图5.7 迁移项提示信息**

点击“确定”，进行数据迁移，迁移完成后系统的负载如图5.8所示：



**图5.8 迁移完成后负载图**

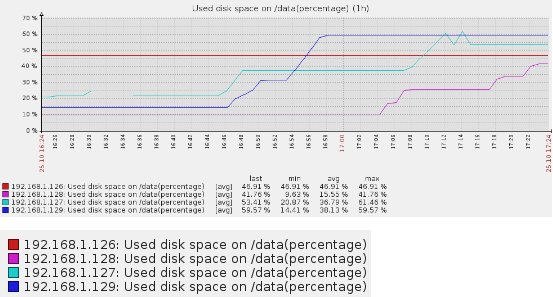
结果分析：

由图5.5和图5.8对比发现，RootServer发现负载最大存储节点的DNmt值大于DNimt，并且计算DNimt<DNoverload, RootServer节点找到负载DNmt最小的存储节点作为目标服务器。并计算出符合迁移条件的用户数据。根据设计好的负载迁移策略，完成数据的迁移。负载最重的服务器负载降低，负载最轻的服务器负载上升。并都在合理范围内。

场景二：

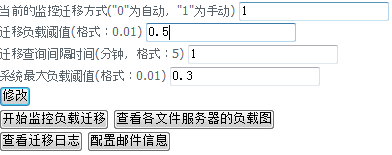
使用LR模拟多用户上传数据，使整个系统的负载上升，模拟系统负载整体过重的情况。

负载图如图5.9所示：



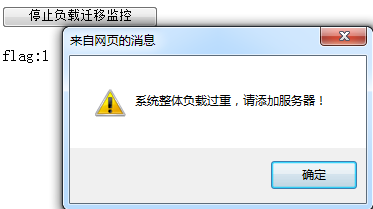
**图5.9 迁移前负载图**

为模拟系统整体负载过重情况，“迁移阈值”为系统默认计算出的DNimt值，将DNoverload值设置为小于DNimt值，如图5.10所示：



**图5.10 迁移参数页面**

点击“开始监控负载迁移”按钮，弹出如图5.11提示：



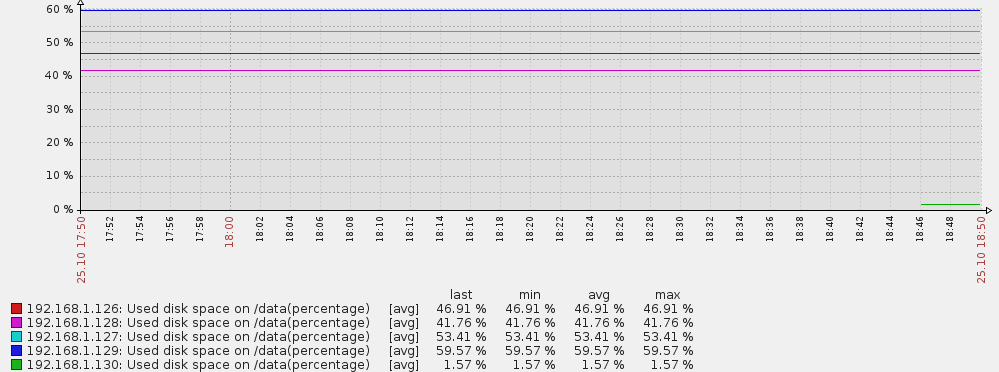
**图**5.11 **迁移提示信息**

结果分析：

由于上传的文件量有限，为模拟上述场景，将DNoverload的值设置为0.3，小于DNimt，此时系统就会根据设置的参数，判断系统整体负载过重，虽然存在存储节点的负载大于DNimt的值。系统也不会进行数据的迁移，只是提醒管理员动态扩大系统容量。实际工作中DNoverload值由系统性能，合理调整。

管理员根据提示扩展系统资源，添加服务器节点192.168.1.130；

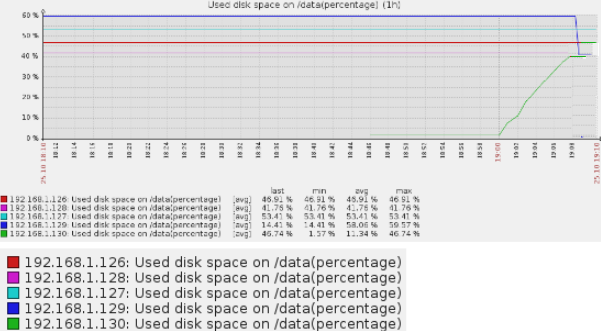
添加完成后系统的负载如图5.12所示：





**图5.12 迁移前负载图**

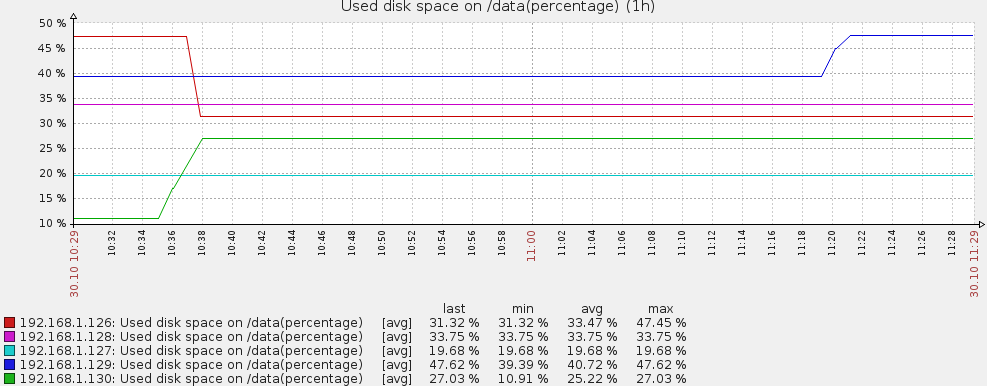
由图5.12可以看出，添加新的服务器192.168.1.130后，服务器192.168.1.130的负载为0。重新设置DNoverload参数，进行数据迁移。迁移完成后系统的负载如图5.13所示。



**图5.13 迁移后负载图**

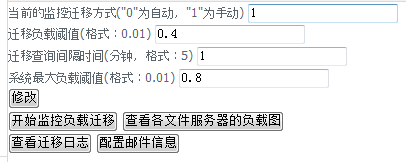
（2）自动迁移过程与结果分析

上面已经对迁移算法正确性和可用性进行了验证。符合对算法的预期设置。现在观察自动迁移时系统负载变化情况。模拟用户上传，删除数据等过程。创建负载不均衡的环境。  
 系统负载图如图5.14所示：



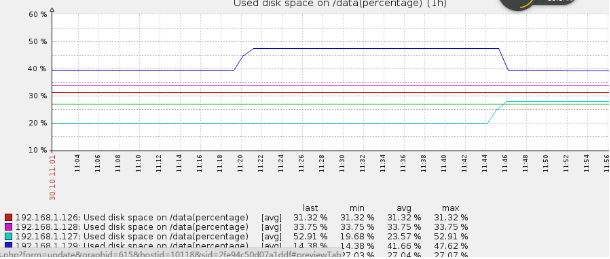
**图5.14** **自动迁移前负载图**

迁移参数如图5.15所示：



**图5.15 迁移参数页面**

一段时间后，查看系统的负载如图5.16所示：



**图5.16** **自动迁移后负载图**

结果分析：

对比分析负载图5.14和图5.16可以发现，刚开始系统中各服务器的负载差别较大。将迁移方式设置为自动迁移。几分钟后，查看系统负载值。服务器的负载都趋于“迁移负载阈值”左右分布。查看迁移日志，发现系统只进行了一次迁移。迁移算法以最少的迁移次数，最大化的均衡负载。

**5.2.3 结果分析**

通过5.2.1节和5.2.2节中对动静态相结合的负载均衡算法测试过程和结果得出，静态负载算法通过将系统的负载和轮询算法相结合的方式，将用户均衡的分配到存储节点。使的各存储节点均衡的获得用户的请求。当OBS系统进行存储节点的扩展，或者其它操作导致负载的不均衡分配时，根据设置的迁移阈值，选择迁移参数进行负载的迁移，算法在保证系统可靠性，可用性前提下，使用最少的迁移次数，最大化的平衡系统的负载。

另外，在系统工作高峰期，为避免数据迁移占用多余的带宽，管理员可以根据系统工作状态，调整参数。使的迁移工作尽量在系统空闲期进行。

## 5.3目录容量和响应时间关系测试

本节主要是测试当单一目录下文件数据量不断增长时，在该目录下进行对文件进行操作，响应时间的变化。目的是研究FDDM策略是否确实提高系统性能。

## 5.3.1实验仿真环境

影响实验结果的因素很多，因此在整个实验过程中必须保证可控变量的一致性。如硬件环境、软件环境、和文件系统等。

实验PC配置如下：

HardWare

CPU：Intel(R) Xeon(R) CPU E5606 @ 2.13GHz，

DISK：9.2G

Memory：512MB

SoftWare

OS：SUSE Linux Enterprise Server 11 SP1 (i586)

FileSystem：ext3

## 5.3.2 实验过程

在linux系统下添加一块disk，格式化为ext3文件系统，挂载到/data/，选择/data/作为存储数据的父目录。在/data/增加文件，将文件数量分别稳定在1000,10000,

100000,500000，通过脚本模拟list,read,write,add,delete文件的过程。（其中list指列出/data/下的所有文件或者统计该目录下文件数量；read/write/delete随机选取/data/下的文件，进行读/写/删除操作；add在/data/创建文件）观察当文件数量不同时，上述操作完成的时间差异和变化趋势。完成花费时间通过操作前和操作后的时间差获得。

注意细节：

1. 所有的操作在同一环境下进行，避免由于可控环境不同引起的访问性能差异

（2）当/data/下文件数量不同时。list,read,write,add,delete五个过程分别对一个文件进

行操作时，差异太小，而且可能存在随机性。因此，实验过程是对多个文件进行

多次操作。

（3）在操作中缓存会对上述过程产生影响，因此。为避免缓存对实验结果的影响，在

进行操作前清空缓存。

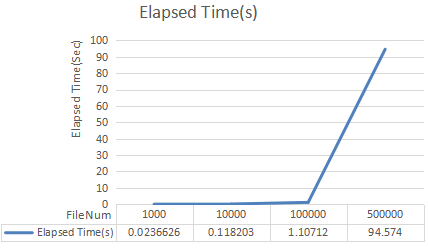
## 5.3.3实验结果及分析

实验环境中，文件系统中iNode数量有限，因此创建文件数量有限。

分别模拟单一目录下文件数量为1000,10000,100000,500000时。进行上述5种操作，观察时间变化。

（1） List操作

单一目录下文件数量分别为1000,10000,100000,500000时。List过程中文件数量和elapsed time关系如图5.17所示：



**图5.17 List files elapsed time**

附注7:

X轴：/data/文件数量

Y轴：操作花费的时间

折线：变化趋势

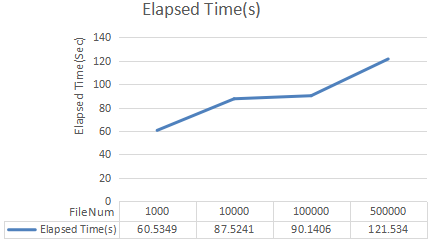
表格中Elapsed Time：操作花费具体时间，单位S

List结果分析：

由图5.17中折线部分看出，文件数量小于100000时，List操作响应时间增长缓慢。当文件数量增长到500000时，对list操作响应时间远远大于之前操作所需时间。高达1.5分钟，这对于用户来说是无法忍受。可以预测，当文件夹下目录数量到达百万时，系统中的list操作几乎是不可完成的。

（2）read操作

随机选择1000个文件，打开和关闭这1000个文件100次，当/data/下文件数量分别从1000到500000变化过程中，read文件的elapsed time如图5.18所示：



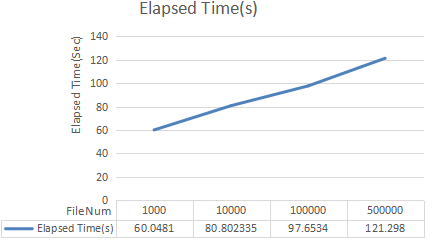
**图5.18 Read files elapsed time**

Read操作结果分析：

由图5.18 可以看出，当/data/下文件数量由1000增长到500000的过程中，系统读取文件的时间在持续增加。从1000到500000系统读文件的性能降低了66%左右。因此，可得出单一目录下文件数量越多，read操作的响应时间就会增加。性能下降的速度很快。

1. Write操作

随机选择1000个文件，向每个文件中写一次数据。当文件数量从1000到500000变化时，写文件花费的时间如图5.19所示：



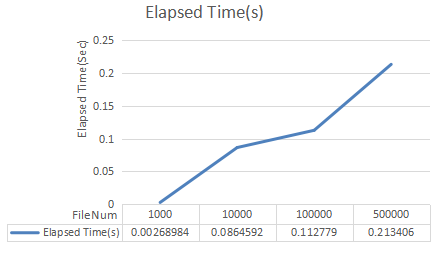
**图5.19 write files elapsed time**

Write操作结果分析：

Write操作与Read操作响应时间和数据量变化趋势基本类似，此处不在累述。

1. Add操作

当/data/下文件数量分别为1000,10000,100000,500000时。分别添加1000个文件。观察在目录下文件数量不同时，添加文件花费的时间。如图5.20所示：



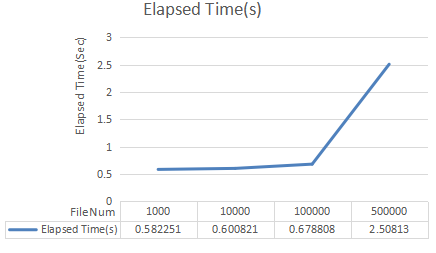
**图5.20 Add files elapsed time**

Write操作结果分析：

由图5.20折线图变化趋势得到，/data/下文件数量增多时，完成add操作所需的时间也增多。从1000到500000，虽然所需时间都很少，但从比例来看，add操作花费的时间几乎直线增加。以至于花费时间不在同一数量级。当文件数量达到百万，千万以至于十亿时，add操作所花费的时间会更大，将会降低用户体验。

1. Delete操作

当/data/下文件数量分别为1000,10000,100000,500000时。分别删除1000个文件。观察在目录下文件数量不同时，删除文件花费的时间。如图5.21所示：



**图5.21 Delete files elapsed time**

Delete操作结果分析：

由图5.21得到，当文件数量小于100000时，delete操作所需时间在增长，但是变化甚微。当文件数量达到500000时，delete操作花费时间迅速增大，相比于之前操作，delete操作花费时间增加了80%左右。

**总结**：

由上面所有实验结果可以得到，单一目录下文件数量增加时。对该目录下文件进行list,read,write,add,delete任意一种操作时响应时间都会增加。因此，可以得出结论，单一目录下文件数据量过多时，在该目录下任何操作的响应时间都会增加。所以对于数据分布策略的研究很有必要性。实现数据的扁平化分布，解决单一目录下数据分布过多的问题。

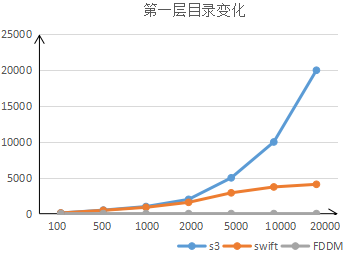
## 5.4 FDDM策略测试结果及分析

模拟Amazon S3，Swift，FDDM策略上传数据。实验中分别观察文件数量为[0-100],[100-500],[500-1000],[1000-2000],[2000-5000], [5000-10000]各层目录平均分布情况。在此实验中，通过折线图观察上传文件这一过程中目录分布与文件数量的关系，主要选几个稳定的点，停止上传文件，观察系统中目录分布状态。由试验过程总结出各用户的文件数量分别稳定在[100,500,1000,2000,5000,10000,20000]时，在S3、Swift、FDDM策略下，各层目录数量之间的关系表，如表5.1所示：

**表5.1 S3，swift，FDDM策略下各层目录数量对比表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 分布策略 | | S3 | Swift | FDDM |
| 文件数量 | 目录层次 |  |  |  |
| 100 | 第一层 | 100 | 100 | 16 |
| 第二层 | 0 | 1 | 5 |
| 第三层 | 0 | 1 | 1 |
| 500 | 第一层 | 500 | 469 | 16 |
| 第二层 | 0 | 1 | 11 |
| 第三层 | 0 | 1 | 3 |
| 1000 | 第一层 | 1000 | 886 | 16 |
| 第二层 | 0 | 1~2 | 16 |
| 第三层 | 0 | 1 | 4 |
| 2000 | 第一层 | 2000 | 1584 | 16 |
| 第二层 | 0 | 1~2 | 16 |
| 第三层 | 0 | 1 | 8 |
| 5000 | 第一层 | 5000 | 2910 | 16 |
| 第二层 | 0 | 2 | 16 |
| 第三层 | 0 | 1 | 20 |
| 10000 | 第一层 | 10000 | 3730 | 16 |
| 第二层 | 0 | 3 | 16 |
| 第三层 | 0 | 1 | 40 |
| 20000 | 第一层 | 20000 | 4096 | 16 |
| 第二层 | 0 | 5 | 16 |
| 第三层 | 0 | 1 | 77 |

由表5.1画出S3,swift,FDDM策略下各层目录对比折线图。分析各策略下目录变化状态，分别如图5.22、5.23、5.24所示。



**图5.22 各策略下第一层目录变化折线图**

附注8:

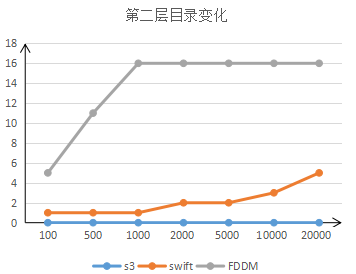
X轴：系统中对象数量

Y轴：各层目录数量。

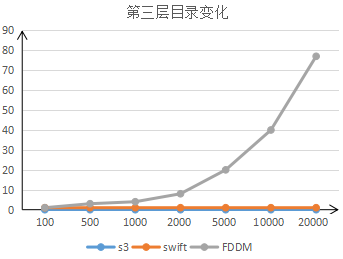
蓝色：S3策略目录分布

橘色：swift策略目录分布

灰色：FDDM策略目录分布（图b和图c相同）



**图5.23各策略下第二层目录变化折线图**



**图5.24 各策略下第一层目录变化折线图**

**结果分析：**

通过图5.22、5.23、5.24可得。S3数据分布策略下 ，所有数据都在bucket目下，第一层目录数量随着用户文件数量始终正比例增长。用户所有数据存储在单一目录下。Swift数据分布策略下：刚开始随着用户数据量的增长，第一层目录几乎直线增长。而第二层目录下只存储一个到两个文件。当文件数量超过10000个时，逐渐缓慢增长，直至该层目录达到饱和。FDDM策略下，当对象数量到达100时，第一层目录数量稳定到16。第二层目录刚开始随着对象数量变化逐渐增长，最后稳定到16目录。第三层目录随着对象数量的增长缓慢增长。通过统计上面三层目录得出。当系统中对象数量达到20000时，S3单一目录下子目录数量达到20000个。而Swift数据分布策略中存在某一目录下子目录数量达到4000多。FDDM策略下单一目录中，子目录最多只有70多个。可以预测。当系统中用户的文件数量为G级别时，S3单一目录下子目录数量也为G级别，Swift单一目录下数据量可达M级别。而FDDM策略下。可以按照用户文件数量级别，动态配置目录层次和目录容量。实现数据在存储节点上的扁平化分布。

由上述实验结果可得。S3策略并未实现数据的扁平化分布。而Swift策略在一定程度上缓解了数据分布过于集中的问题，当系统中用户数据量为K级别时，第一层目录已达到K级别，而第二层几乎都是一个目录，数据分布散乱。当系统数量达到G级别时，第一层目录已达到饱和，第二层目录容量达到M级别。使用Swift策略进行数据分布时，用户数据量过少，易导致数据分布过于散乱。而当用户数据量过多，又会导致数据分布过于集中的问题。本文提出的FDDM策略，从根本上解决了目录的分布问题。管理员可以根据系统中用户文件数量，动态配置目录层次和每层目录容量。既可以避免用户数据较少时目录分布散乱的问题。又可以解决用户数据量过大时数据分布过于集中的问题。从而实现了数据在存储节点上扁平化分布。

## 5.5 性能测试结果及分析

通过5.4节实验过程，得到了各策略下目录分布情况。通过实验结果图，可以看出FDDM策略通过分层的方式实现所有文件在数据节点上的扁平化分布，实现数据均匀分布的目的是为了提高系统的存取性能。减少用户请求等待时间。降低系统对用户的响应时间。在此模拟S3,Swift, FDDM策略分布数据。测试在相同环境中不同的分布策略下系统性能变化。

## 5.5.1 仿真环境

HardWare

CPU：Intel(R) Xeon(R) CPU E5606 @ 2.13GHz，

DISK：10G

Memory：1G

SoftWare

OS：SUSE Linux Enterprise Server 11 SP1 (i586)

FileSystem：ext3

选择配置完全相同的三台服务器，进行实验。

## 5.5.2 实验过程及实验结果

实验过程介绍：

分别模拟S3，Swift和FDDM分层过程创建文件。文件数量分别稳定到1000,10000,100000,1000000时。模拟list,read,write,add,delete文件过程。比较分别在S3,Swift和FDDM策略下完成上述操作所花费时间差异，分析FDDM分层存储结构是否有利于提高系统性能。采用多组不同数量级的object，避免由于选取一组数据或者由于数据量较少带来的实验结果判定的误差。

表5.1已经描述了当系统中文件数据量分别为1000,10000时。各策略下目录分布情况。在此列出当文件数据量为100000,1000000时。各策略下目录分布如表5.2所示。

**表5.2 各策略下目录分布**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 目录数量 | 目录层次 | S3 | swift | FDDM |
| 100000 | 第一层 | 100000 | 4096 | 16 |
| 第二层 | 0 | 24 | 16 |
| 第三层 | 0 | 1 | 390 |
| 1000000 | 第一层 | 1000000 | 4096 | 256 |
| 第二层 | 0 | 244 | 16 |
| 第三层 | 0 | 1 | 224 |

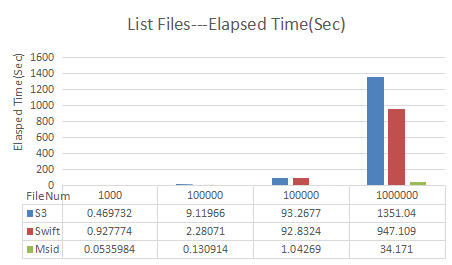
5.3节中通过实验数据证明，父目录下子目录数据量增多时，系统的响应性能将会降低。因此，此处只是测试当系统中文件数量相同时，采用不同分布策略是否对系统的响应性能有影响。为了保证实验环境和实验过程的一致性。在进行read,write,delete操作时。文件数目的选择按照S3,Swift和FDDM最后一层目录数量合理选择。避免三个策略由于操作过程不同造成结果的误差。三个策略下分别进行list,read,write,add,delete操作，并使用相同的脚本，只是修改操作的文件路径不同。当系统中object数量不同时read,write,add,delete操作时所选择object数量，操作次数和对象数量之间的关系如表5.3所示：

**表5.3 Operate Object**

|  |  |  |  |
| --- | --- | --- | --- |
| Operate(S) | Total Object Num | Operate Times | Operate Object Num |
| Read/Write | 1000 | 1000 | 4 |
| 10000 | 20 |
| 100000 | 24 |
| 1000000 | 244 |
| Add | 1000 | 1 | 1000  1000  1000  1000 |
| 10000 |
| 100000 |
| 1000000 |
| Delete | 1000 | 1 | 4 |
| 10000 | 20 |
| 100000 | 24 |
| 1000000 | 244 |

1. list操作

Object数量分别稳定到1000,10000,100000,1000000时。文件按照S3,Swift和FDDM分别在配置相同的存储节点分布完成后。使用脚本list当前文件，响应时间对比变化如图5.25所示：



**图5.25 List Performance Comparison**

附注10

柱状图

X轴：文件数量

Y轴：响应时间

蓝色，红色，绿色：S3，Swift和FDDM策略

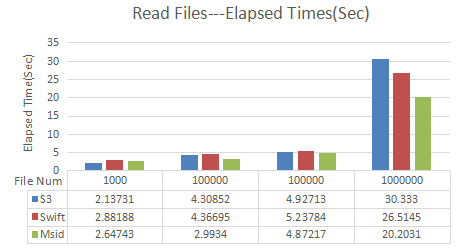
表格：第一行数据量，第二、三、四分别不同策略下list操作响应时间

**List操作结果分析：**

由图5.25可以直观的看出。使用FDDM策略进行数据的分布，list文件时响应时间最短。List文件包括统计当前文件数量或者列出当前文件夹下所有文件等。当文件数量达到1000000时，FDDM与Swift，S3策略相比，List操作的响应时间分别比它们快3.5%和2.5%左右。当系统中数据量达到百万时，S3和Swift策略下List的操作几乎是无法完成的。而使用FDDM策略分布后的数据的响应时间大概在30秒左右。

1. Read操作

S3,Swift和FDDM三种分布策略下Read操作花费时间如图5.26所示：



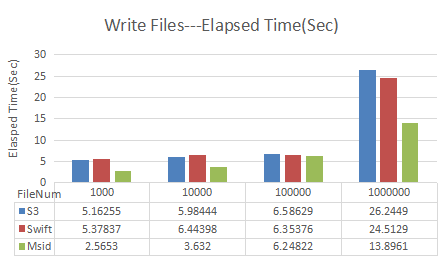
**图5.26 Read Performance Comparison**

**Read操作结果分析**：

由图5.26可以看出，当系统中文件数量少于1000时，S3分布方式读文件速度最快，FDDM次之，Swift最慢。当文件数量很少时，S3策略下bucket目录目录下子目录较少，因此查找文件速度很快，而其它两种策略下每层目录容量也很少，但是由于目录层次相对较多，所以存取时间相对较慢。但是相差不大。当存储节点下文件数量达到10000后，使用FDDM分布策略读文件的速度明显高于其它两种策略。当文件数量达到百万时，FDDM读文件的速度相比于S3提高了34%左右，相对于swift策略提高了23%左右。而且在测试过程中，通过调整FDDM的目录结构发现当文件数量不断增加时，FDDM分布策略下读取一个文件花费的时间几乎没差。可以预测，当系统中文件数据量很大时，可以使用FDDM策略动态调整目录结构，减少由于object数量增加而引起的系统性能降低的问题。

（3）Write操作

三种分布策略下Write操作花费时间如图5.27所示：



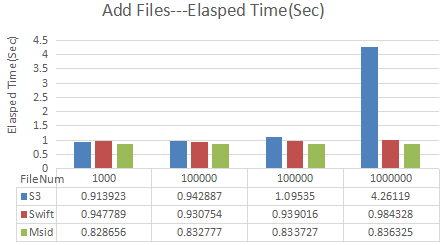
**图5.27 Read Performance Comparison**

**Read操作结果分析:**

Write操作的过程类似于Read过程。此处不再累述。

（4）Add操作

三种分布策略下每次添加1000个文件花费时间如图5.28所示：



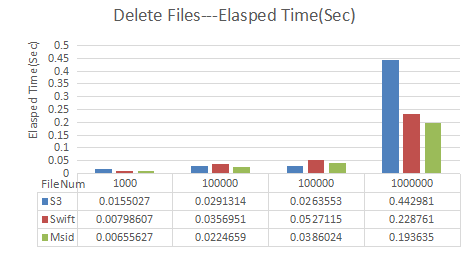
**图5.28 Add Performance Compariso**n

**Add操作结果分析:**

由5.28柱状图可以得到，当文件数量小于10000时，三种分布策略下分别add文件。FDDM与其它两种策略相比，响应时间提高了10%左右，而Swift和S3花费时间相差不大。当object数量达到100000时，FDDM相比于S3，性能提高了17%左右。而当object数量到达1000000时，明显S3分布策略下add操作花费的时间远远大于其它两种策略。横向观察可以发现FDDM策略下随着object数量增加add操作响应时间几乎没有变化。而Swift策略下add操作花费时间变化幅度较小。根据上面数据可以预测。当系统中文件数量继续增长时，S3策略下add操作花费时间将不断增长。在当前数据中，Swift策略下之所以Add操作花费时间变化幅度很小，是由于第一层目录较大，第二层文件数量较少。而已知Swift策略下第一层目录已达到饱和。因此当系统中文件数量持续增长时，第一层子目录下子目录数将会不断增加。必然会引起Add操作性能的下降。而FDDM策略可以通过动态的参数调整策略，调整每层目录的容量和目录层次，保证每层目录下子目录数量不至于过多。从而保证系统对Add操作的响应。

1. Delete操作

三种分布策略下Add object操作花费时间如图5.29所示：



**图5.29 Delete Performance Comparison**

三种策略下五种操作的实验结果总结如表5.4所示。

**表5.4 性能测试结果**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Policy  Operate(S) | FileNum | S3 | Swift | FDDM |
| List | 1000 | 0.469732 | 0.927774 | 0.0535984 |
| 10000 | 9.11966 | 2.28071 | 0.130914 |
| 100000 | 93.2677 | 92.8324 | 1.04269 |
| 1000000 | 1351.04 | 947.109 | 34.171 |
| Read | 1000 | 2.13731 | 2.88188 | 2.64743 |
| 10000 | 4.30852 | 4.36695 | 2.9934 |
| 100000 | 4.92713 | 5.23784 | 4.87217 |
| 1000000 | 30.333 | 26.5145 | 20.2031 |
| Write | 1000 | 5.16255 | 5.37837 | 2.5653 |
| 10000 | 5.98444 | 6.44398 | 3.632 |
| 100000 | 6.58629 | 6.35376 | 6.24822 |
| 1000000 | 26.2449 | 24.5129 | 13.8961 |
| Add | 1000 | 0.913923 | 0.947789 | 0.828656 |
| 10000 | 0.942887 | 0.930754 | 0.832777 |
| 100000 | 1.09535 | 0.939016 | 0.833727 |
| 1000000 | 4.26119 | 0.984328 | 0.836325 |
| Delete | 1000 | 0.0155027 | 0.00798607 | 0.00655627 |
| 10000 | 0.0291314 | 0.0356951 | 0.0224659 |
| 100000 | 0.0263553 | 0.0527115 | 0.0386024 |
| 1000000 | 0.442981 | 0.228761 | 0.193635 |

分析总结：

由上面性能测试结果可以看出，系统中使用FDDM策略进行数据在存储节点上的分布，实现了数据在存储节点上的扁平化数据分布。提高了系统的响应速度，在一定程度上解决了单一目录下数据过多，降低系统访问性能的问题。FDDM策略可根据估算用户数据量动态调整目录结构，管理员可根据性能测试结果，动态调节，选取性能最高的目录分布形式。本文提供的是一个关于数据扁平化分布的通用算法。

## 总结

本文以集群存储中对象存储为研究对象，介绍了OBS系统的架构设计。接着重点介绍了对象存储中负载均衡的研究，设计和实现。负载均衡技术包括负载的分配和负载的迁移两个过程。同时介绍了对象存储系统中节点数据分布问题，通过对S3和Swift对象存储数据模型的分析，针对数据分布模型的不足，本文提出一种动态可调的数据分布方法——FDDM。现对论文工作做如下的总结：

1: 简单介绍了云存储系统发展现状，以及云存储相比于传统存储模型所具有的优势。接着说明对非结构化数据研究的原因和必要性。从而引出对象存储。重点介绍云存储环境需要解决的问题，比如数据分布，负载均衡，如何提高可靠性、可用性、以及资源优化的问题。调研并分析了云存储环境中对象存储和负载均衡的现状，以及待解决问题。并说明了负载均衡和对象存储中数据分布方法研究的可行性和必要性。

2: 结合OBS系统，对需要了解的一些理论技术做简单的介绍（集群存储模型，对象存储，负载均衡，快照）。为后面介绍系统架构和实现算法建立理论支撑。

3: OBS系统架构设计介绍。首先介绍OBS系统的整体架构，然后详细介绍OBS架构各层次组件功能以及各个组件之间如何协作。并介绍监控系统，以及监控系统在OBS中工作流程。OBS采用的是基于动态反馈三层非对称的分布式系统架构。并说明此架构的优势。

4: 重点介绍了OBS系统中负载均衡算法和对象存储系统中数据分布算法的设计与实现。分别介绍了负载均衡算法中负载分配和负载迁移的策略设计，以及对各节点负载的存取。接着介绍了对象存储系统中如何实现数据的扁平化分布，本系统提出FDDM策略，对策略设计和在系统中实现做了详细介绍。

5: 在导师提供的集群环境下对OBS系统中负载均衡算法和对象存储系统数据分布策略进行了测试。并对文件操作系统的响应时间与目录下文件数量的关系进行测试。发现，单一目录下文件数量越多，对存取性能越低。通过对测试结果的分析，使用动静态相结合的负载均衡策略，实现了对系统中负载的均衡分配和调控，使系统能够均衡的响应用户的请求，在提高系统响应时间的同时优化了资源的利用。FDDM策略，通过性能测试发现，有效的解决了对象存储中数据的分布问题，使数据分布更加扁平化，从而带来效率上的提升。

目前，对象存储发展为下一代网络存储系统的趋势已经相当明显。综上所述，本文以对象存储系统作为研究对象，在集群存储架构、负载均衡和数据分布等方面进行了研究，自主设计并实现了一种分布式对象存储系统（OBS），对OBS中负载均衡和数据分布策略进行研究，测试结果表明，本文研究与设计的负载均衡算法，数据分布算法达到了预期的功能与目标，其中的技术和方法可以为相关对象存储系统的研究提供有益的参考，具有较好的应用和推广价值。但是OBS系统在可用性、可靠性、安全性、数据一致性等方面仍然有很多工作要做。比如：MetaServer，RootServer单点问题。以及数据节点副本问题，例如采用几个副本保证可用性，以及复制副本时采用最大保护模式还是最大性能问题，以及节点故障数据和服务恢复时数据一致性问题等。OBS系统仍需要不断进行完善，使其能够满足对海量数据的存储需求。

## 参考文献

［1］闫晓. 基于Swift的资源负载均衡策略研究[J].哈尔滨工业大学. 2015.06:4～

65

［2］杨传辉. 大规模分布式存储系统原理解析与架构实践[M]. 机械工业出版社,

2010:42～50

［3］http://aws.amazon.com/[Z/OL]. 2011.03

［4］刘鹏. 云计算[M].电子工业出版社. 2010.3:102～104

［5］<http://www.cnblogs.com/yuxc/archive/2012/07/04/2575536.html> [Z/OL].

［6］戴毅. 基于对象存储系统复杂均衡技术研究[J]. 华中科技大学. 2007.05:22~30

［7］W. Zeng, Y. Li, J. Wu. Load Rebalancing in Large-Scale

Distributed File System. IEEE 1st International Conference on Information Science

and Engineering (ICISE). 2009:265-269

［8］Prabavathy.B ,Priya.K,Chitra Babu . A Load Balancing Algorithm For Private Cloud

Storage. IEEE-31661. 2013:10~15

［9］范中磊. 网络存储技术漫谈之三：集群存储[J]. 记录媒体技术. 2010,04:2~4

［10］Li Daisong, Liu Yong, Pan Chengsheng. Study and Implementation of one kind of

Load Balancing Algorithm[J]. Transcations of Shen Yang Li Gong University. 2005.24(3):47~51

［11］Cheng Zhigang, Li Deng, Zeng Zhiwen. Reasearch on a Dynamic Load Balancing

Policy and Correspanding Model and Algorithm in the Distributrd System[J]. Mini-

Micro Systems, 2002. 23(12):1434~1437

［12］龚玮. 对象存储文件系统的设计与实现[J]. 华中科技大学. 2006.05:9~20

［13］http://solution.sequoiadb.com/cn/Detail-cat\_id-5-id-20[Z/OL]. 2011.03

［14］周 梅. 分布式锁的设计与实现[J]. 计算机工程.2008,08:80~84

［15］徐岚. 负载均衡技术的综述[J]. 光盘技术 2008,12:4

［16］负载均衡技术全攻略 [http://www.yesky.com/20010703/187822.shtml[Z/OL](http://www.yesky.com/20010703/187822.shtml%5BZ/OL)]

［17］Apache. [http://www.apache.org/[Z/OL].1999[Z/OL].20012-08-07](http://www.apache.org/%5BZ/OL%5D.1999%5BZ/OL%5D.20012-08-07)

［18］梁红洲. Web系统测试技术研究[J]. 科技广场 2007,1:23~24

［19］宫垂刚，韦一，张华丰等. PHP完全自学手册[M]. 北京：机械工业出版社，2008.9

［20］[喻占武](http://www.cnki.net/kcms/detail/search.aspx?dbcode=CJFQ&sfield=au&skey=%e5%96%bb%e5%8d%a0%e6%ad%a6&code=)，[郑胜](http://www.cnki.net/kcms/detail/search.aspx?dbcode=CJFQ&sfield=au&skey=%e9%83%91%e8%83%9c&code=)，[李忠民](http://www.cnki.net/kcms/detail/search.aspx?dbcode=CJFQ&sfield=au&skey=%e6%9d%8e%e5%bf%a0%e6%b0%91&code=). 基于对象存储的海量空间数据存储与管理学术论文[J].

2008.05

［21］巴子言，吴军，马严. 基于虚节点的一致性哈希算法的优化 软件2014 年第

35 卷 第12 期

［22］W. Zeng, Y. Li, J. Wu, Q. Zhong and Q. Zhang. ”Load Rebalancing in Large-Scale

Distributed File System.”IEEE 1st International Conference on Information Science and Engineering (ICISE), 2009, pp. 265-269.

［23］Prabavathy.B ,Priya.K,Chitra Babu "A Load Balancing Algorithm For Private

Cloud Storage" IEEE-31661

## 致谢

行文至此，感受颇多。

猛然感觉，三年的硕士生涯即将结束。而硕士生涯的点点滴滴，依然历历在目。回想这三年的校园生活。充满怀念与感激。怀念与老师，同学一起学习工作的日子，怀念自己为了实现目标而不惧辛苦，不断努力的样子。感谢，在我学习生涯中给予我帮助，支持，鼓励和关心的老师，同学们。

感谢我的导师范中磊老师。范老师在学术上高屋建瓴，拥有十年磨一剑之气魄。范老师在学术上的高瞻远瞩为我门提供了学术研究的方向，让我们能够紧跟现代互联网的发展方向，抓住机遇和挑战。在平时学习中，范老师付出自己大量时间帮助我们解决遇到的问题。耐心聆听我们对于事情的看法和态度，并给予指导和中肯意见。范老师带给我们的不止是学术上的进步，更是面对学习，工作，和生活的一种态度。文字已不足以表达我对范老师的感激。于我而言，范老师是导师，亦是朋友，更是亲人。

感谢实验室的李东海老师，感谢李老师平时对我的指导和帮助，让我在这三年的学习生涯中受益匪浅。李老师对待工作的认真负责和对科研一丝不苟的精神，都深深的影响着我。

感谢陪伴我度过三年欢乐时光的舍友们，同门师兄妹们：尤其是朱晓晨，王瑶，刘佳，王章章，赵彩云，张苗苗。感谢你们带给我的真诚的友谊和热忱的帮助。让我感受到同窗友情的温暖和弥足珍贵。我会一直带着这份温暖。并将这份温暖带给身边的人。

感谢我的父母，在我成长中给予我建议，却尊重我的每一次选择。我的每一步成长，学业上的进步都凝聚着他们的汗水和心血。他们是我最坚强的后盾力量。给予我的是无私，永恒的爱。

感谢自己，感谢那个为了目标而愿意不断努力的自己，感谢那个为了进步而愿意忍受孤单的自己，感谢那个不管经历多少都不忘初心的自己。未来的路很长，或许平庸，但是我依然会很认真，很努力的迎接每一步。

最后，感谢各位专家，教授在百忙之中参与论文的审阅和指正。

硕士生活是校园生活的结束，也是工作生涯的起点，我会迎着新的起点，更加努力。