中国科学技术大学

硕士学位论文

分布式文件系统元数据管理技术研究与实现

作者姓名: 冯幼乐

学科专业: 网络传播系统与控制

导师姓名: 朱 明 教授

完成时间: 二〇一〇年五月十日

University of Science and Technology of China

A dissertation for master's degree

Research and Implementation on Technologies of Metadata Management in Distributed File System

Author's Name: Youle Feng

Speciality: Network Comunication System & Control

Supervisor: Prof. Ming Zhu

Finished time: May 10nd, 2010

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文，是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____　　签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开　□保密（____年）

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘 要

分布式文件系统通过将多台机器的资源组织起来，对外提供统一的、大容量、高性能、高可靠，易扩展的文件服务，满足了大规模应用的要求，是目前存储领域研究的重点和难点。针对文件系统中元数据和文件数据存储和访问的不同特点，分布式文件系统通常包括数据存储系统和元数据管理系统。为应对大量的元数据操作请求，保障元数据操作的一致性和可靠性，元数据管理系统的设计与实现及其重要。本文设计并实现了分布式文件系统 LandFile 的元数据管理系统，研究了分布式文件系统元数据管理的关键问题以及解决策略，具体内容如下：

研究了面向节能的元数据服务器集群的负载均衡策略：大规模的应用中，能源问题越来越成为人们关注的焦点。本文在动态元数据管理策略的基础上，提出了一种面向节能的负载均衡策略，通过在负载较低时，主动关闭元数据服务器集中负载，达到整体节能的目的。随着负载的增加，再逐步打开元数据服务器对外提供服务。良好设计的动态元数据管理策略保证了服务器加入和退出时的平滑扩展。实验表明面向节能的负载均衡策略在负载低谷时，节能效果非常明显，对系统性能亦影响较小。

研究了元数据管理的一致性保障策略：分布式条件下元数据可能存在多个副本，许多元数据请求可能涉及多台元数据服务器，本文设计了基于主节点的元数据缓存架构，实现了多副本元数据的更新策略，针对分布式操作，设计并实现了基于两阶段提交的协议来保证其一致性。

研究了元数据管理的可靠性保障策略：本文研究了元数据服务器的节点管理、故障检测以及故障恢复机制，实现了区域自治的节点管理方法和基于日志的可靠性保障策略，保证了元数据服务器单点失效时，能够被快速替换和恢复，失效时系统仍能无间断的提供服务。

利用上述研究的负载均衡策略，一致性保障策略与可靠性保障策略，设计并实现了元数据管理系统，包括元数据请求处理和可靠性模块，改进了负载均衡模块。

所研究实现的元数据管理子系统，是 863 课题“新一代业务运行管控协同支撑环境的开发（2008AA01A317）”中分布式文件系统 LandFile 的重要组成部分。

关键词：分布式文件系统；元数据管理；节能；负载均衡；缓存一致性；基于日志的故障恢复机制

Abstract

By connecting many computers together, distributed file system can provide storage service with a uniform interface, large capacity, high performance, high-availability and high scalability, which meet the need of large scale application. Distributed file system has been a difficult and popular study point in storage fields. To address the different access features of file data and metadata, recent distributed file systems are divided into to parts: the data storage system and the metadata management system. To handle increasing metadata requests and keep the coherence and reliability of metadata operation, the design and implementation of metadata management system is very important. This article designed and implemented the metadata management system of distributed file system LandFile, and studied the key problems and solutions of metadata management system in distributed file system. This article mainly focuses on the following aspects:

The energy-saving-based load balancing strategy of metadata server cluster: in large scale application, energy cost has been a more and more important problem. Based on the dynamic metadata management, we proposal an energy-saving load balancing strategy. By merging the workload of two metadata servers into one and turning off the other metadata server when the overall load is low, the overall energy consumed is saved. The sleeping metadata server will be added into the cluster when the overall load increases. The carefully designed dynamic metadata management ensures a smooth transition when the server is added and removed. Experiment shows that the energy cost is dramatically reduced in energy-saving mode while the performance is not affected.

The coherence of metadata management: Metadata may have multiple replicas in the system. A few kinds of metadata operation may involve two metadata servers. We designed the primary-node based cache structure and implemented the update strategy of multiple replicas. We also implemented a two-phase-commit protocol to ensure the coherence of distributed metadata operation.

High reliability metadata management: We studied the nodes management, failure detection and failure recovery method of metadata server. By using the nodes management strategy based on region autonomy and log-based failure recovery, the failure node can be replaced and recovered in a short time, the system service is still

available during the failure.

Using the load balancing strategy, coherence strategy and reliable management strategy, we designed and implemented the metadata management system, which includes metadata operation server module and journal module; we also refined the load balancing module.

The metadata management system researched is an important component of 863 topic:“the Development of the New Generation Collaborative Supporting Environment of Business Operation, Management，and Control”.

Keywords: distributed file system, metadata management, load balancing, energy saving, cache coherence, log-based recovery

缩略语

缩略语	英文全称	中文全称
VOD	Video On Demand	视频点播
DAS	Direct Access Storage	直接附网存储
NFS	Network File System，	网络文件系统
VFS	Virtual File System	虚拟文件系统
OSD	Object-based Storage Devices	基于对象的存储设备
MDS	MetaData Server	元数据服务器
BS	Binding Server	绑定服务器
MAID	Massive Arrays of Idle Disks	大规模非活动磁盘阵列

目 录

图表索引

第 1 章 绪论

1.1 研究背景

随着信息技术的不断发展，越来越多的资料被数据化，全球信息存储量以每年超过 30%的速度急剧增长，存储在硬盘上的数据增长率更是高达 114%。大量数据密集型的应用，如数字图书馆、气象数据处理、医学视频图像数据处理、生命科学研究、VOD 点播、海量视频在线编辑与共享、数据挖掘、在线数据处理等，对存储系统的容量、性能(I/O 带宽、吞吐率、响应时间等)，安全性、可扩展性和可用性提出更高的要求。传统的直接存储模式（Direct Access Storage, DAS）已显得力不从心，信息存储的需求和应用的复杂性呼唤新的存储模式，如何构建一个高性能、易扩展、高可用、易管理、安全的存储系统成为目前存储领域所面临的一个重要课题。

本课题建立在 863 课题“新一代业务运行管控协同支撑环境的开发”中“海量存储服务”项目之上，项目需要研制一套可存储海量视频内容的分布式文件系统，该分布式文件系统（LandFile）主要包括元数据管理系统和分布式海量数据存储系统。本文的研究内容即为元数据管理系统。

1.2 国内外研究现状

分布式文件系统已经发展了二十多年，在理论和实践上都取得了较大进展。早期的分布式文件系统是通过局域网互联的工作站和主机的集合，一般以提供标准接口的远程文件访问为目的，由于受到网络环境、本地磁盘、处理器速度等多方面的限制的，早起的分布式文件系统更多地关注访问的性能和数据的可靠性。典型的商用分布式文件系统和研究原型包括 AFS、Coda、Locus、Sprite File System、Sun 公司的网络文件系统 NFS 等。

二十世纪九十年代以来，磁盘技术的不断进步，使得单位存储的成本在不断下降。Internet 的出现和普及，使得对实时多媒体数据的需求和应用逐渐流行起来。大规模的并行计算、数据挖掘、数字图书馆、气象数据处理和医学视频图像数据处理等应用也需要大容量高速度的分布式存储环境。早期分布式文件系统的集中式方式暴露出性能、可靠性及可扩展性等各方面的瓶颈问题，因此出现了将负载分布在多台服务器上同时保持文件操作一致性的分布式文件系统。典型的分布式文件系统和研究原型包括 NFS3 和 NFS4、Frangipani、Petal、

Global File System、PVFS、xFS、GPFS（General Parallel File System）、Storage Tank、Slice、Lustre，Google File System 等。

随着 P2P 技术的出现与发展，基于 P2P 方式的分布式文件系统也逐渐发展起来。基于 P2P 方式的分布式文件系统通过将互联网上一定范围内分散的、独立和异构的文件服务器按照一定结构组织成一个整体，共同对外提供服务。国外典型的研究成果有 OceanStore 等系统。

国内对分布式文件系统的研究从最初的机群文件系统开始，应用在曙光超级服务器集群上的机群文件系统经历了 D2K-COSMOS，D3K-COSMOS，DCFS，DCFS2 等版本的发展，在系统性能、可靠性和可用性方面都有了很大的提高。网络存储系统方面包括中国科学院计算所工程中心研制的蓝鲸网络存储系统等。国内的基于 P2P 技术的分布式文件系统主要有北京大学的燕星、清华大学的 Granary、以及中科大 MDN（Media Distribution Network）系统等。

1.2.1 NFS

网络文件系统[1] [2] [3] [4]（Network File System，NFS）最初由 Sun Microsystems 公司于 1985 年 Solaris 操作系统中实现，现已经更新到 NFSv4 版本。由于 Sun Microsystems 公司将 NFS 协议公开，NFS 成为了 UNIX/Linux 环境中最为广泛使用的分布式文件系统，并被移植到包括 Windows 在内的许多其它操作系统中,基本上成为通用分布式文件系统的标准。

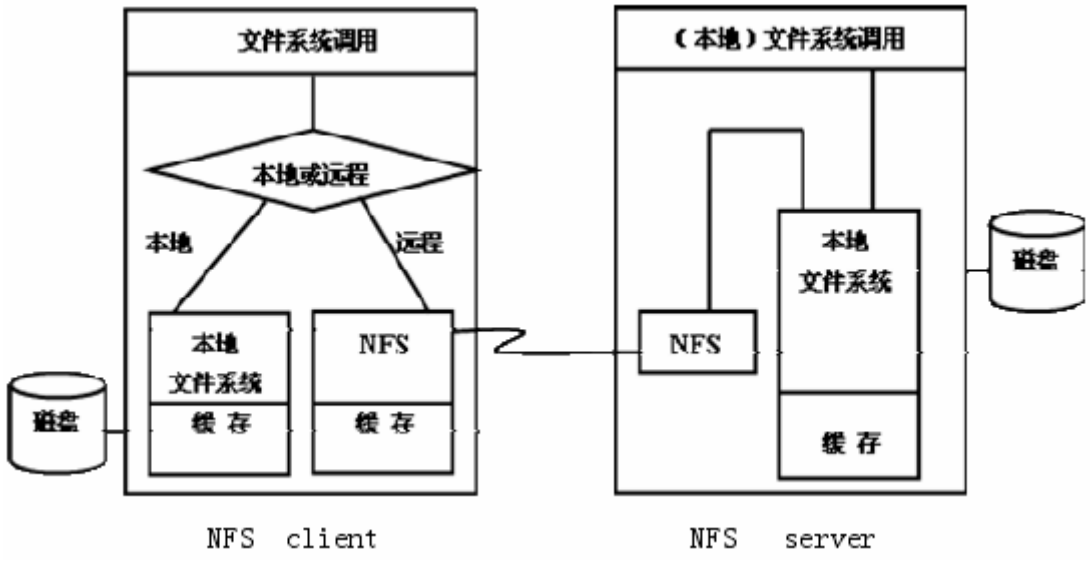

图 1.1 NFS 原理图

如图 1.1 所示，利用 Unix 系统中的虚拟文件系统（Virtual File System，VFS）机制，将客户机的文件操作请求，通过规范的文件访问协议和远程过程调用，转发到服务器端进行处理；服务器端在 VFS 之上，通过本地文件系统完成文件

的处理，实现了远程的文件共享。系统中的每台机器都可以既是客户端又是服务器。NFSv3 及其以前的版本都是采用无状态（stateless）连接和基于时间的弱同步语义，大大简化了协议和实现。但其对于客户端缓存的一致性控制较弱，不能从根本上消除 cache 一致性问题带来的风险。NFSv4 改善了客户端缓，包含了文件锁机制使其变成了有状态（stateful）的协议，引入租约（lease）机制用于文件锁的管理，使得系统的故障恢复更见简单。

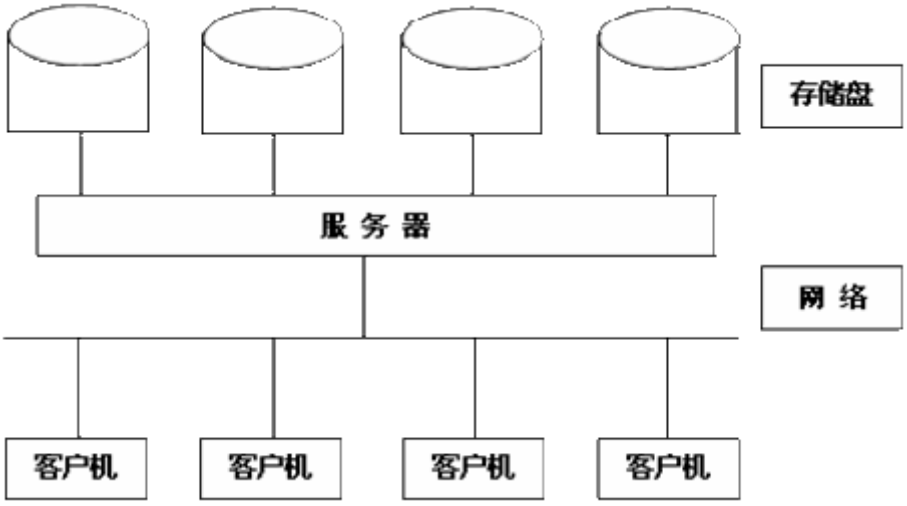

图 1.2 NFS 系统结构图

然而从文件系统的角度看，NFS 只是一个文件服务重定向器（图 1.2），类似的还有 windows 的 cifs 协议。它们本身并不参与文件系统的管理，只是将客户端的文件访问请求转发到宿主文件系统，多台服务器之间并没有统一的文件访问接口，服务器间无法进行负载均衡，在容量、性能、扩展性、可靠性和可管理性上都难有大的提高。虽然 NFS 在逐渐改善以提高单台服务器容纳的客户机数目，但架构上的缺点使得此类文件系统在高负载，大规模的应用上有较大的不足。

1.2.2 Coda

Coda[7] 文件系统是 1987 年 CMU 以 AFS-2 为原型开发出来的容错的分布式文件系统。Coda 服务器提供了透明的文件访问接口，文件数据可以在服务器间进行复制，客户端可能访问数据的任何一个副本，从而可以良好的应对热点数据的访问。Coda 的客户端会根据情况缓存服务器上的共享文件，以允许无法与服务器通讯时（网络分区，服务器故障等）使用缓存继续工作；当故障恢复后，更新过的副本将会写回服务器。出于一致性的考虑，Coda 为在分区中的更新保留了足够多的数据，网络分区恢复后，会检查所有出现写冲突的对象并尽

可能消解冲突。对于无法解决的不一致问题，会提醒人工解决冲突。Coda 系统是少有的允许并发写的文件系统。为了维护系统中缓存的一致性，Coda 采用了回叫信号机制，版本向量机制和 Resolution 例程，这些方法被多个分布式文件系统所借鉴。

1.2.3 xFS

xFS[8] 是由美国加利福尼亚大学伯克力分校设计的广域分布式文件系统。为了减少在广域网上的通讯流量，xFS 采用多层结构以利用文件系统访问的局部性，采用无效写回（invalidation-based write back）的缓存一直性策略以促使客户端主动缓存（aggressive client caching），在比较情况下，会读取临近节点的缓存以减少通讯开销。

xFS 采用无集中式服务器(Sevrerless)模型（图 1.3），它将控制处理和数据存储分布在多个服务器上从而获得系统的可扩展性、可恢复性及简化存储管理。所有的用户服务器既是存储系统的系统服务器，又是存储系统的用户服务器。

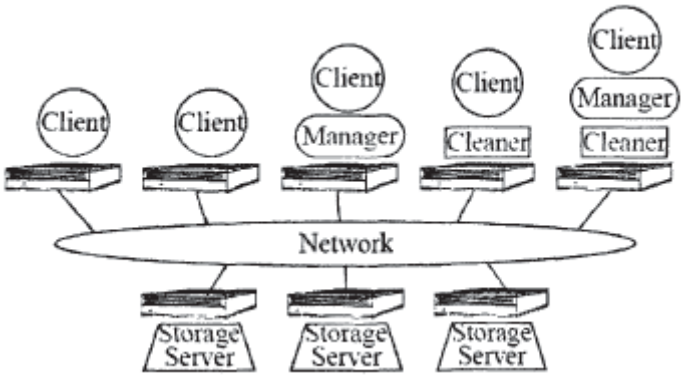

图 1.3 xFS 系统架构图

1.2.4 GPFS

General Parallel File System[9] （GPFS）是 IBM 公司开发的高性能集群文件系统。它采用共享磁盘（shared-disk）结构，客户端采用基于光纤通道或者 iSCSI 与存储设备相连，也可以通过通用网络相连。GPFS 采用 serverless 结构，通过不同粒度的分布式锁解决系统中的并发访问和数据同步问题，保证一致性；用扩展哈希（extensible hashing）技术来支持含有大量文件和子目录的大目录，提高文件的查找和检索效率。GPFS 采用日志技术对系统进行在线故障恢复，每个节点的日志都记录在共享磁盘中，单个节点失效时，系统中的其他节点可以从共享磁盘中检查失效节点的操作日志，进行元数据的恢复操作。GPFS 还支持在线动态添加、减少存储设备，然后在线重新平衡系统中的数据。这些特性在需要连续作业的高端应用中尤为重要。

1.2.5 Lustre

Lustre[10] 分布式文件系统是 Cluster File System 公司推出的面向下一代的存储的分布式文件系统。它提供 POSIX 兼容的 UNIX 系统接口，采用元数据与文件数据分开处理的系统架构（图 1.4），使用两台元数据服务器（一台作为备份）以处理元数据的操作，采用基于对象的存储设备（Object-based Storage Devices，OSD）作为整个文件系统的存储设备。Lustre 首先使用带意图的锁，明显的减少了客户端和服务器端的消息传递，缩短了操作的延迟。Lustre 使用的 OSD 存储设备，具体较高的智能特性，对外提供基于对象的读写接口，而且可以自主的进行负载均衡和故障恢复，极大的降低了元数据服务器的复杂性。

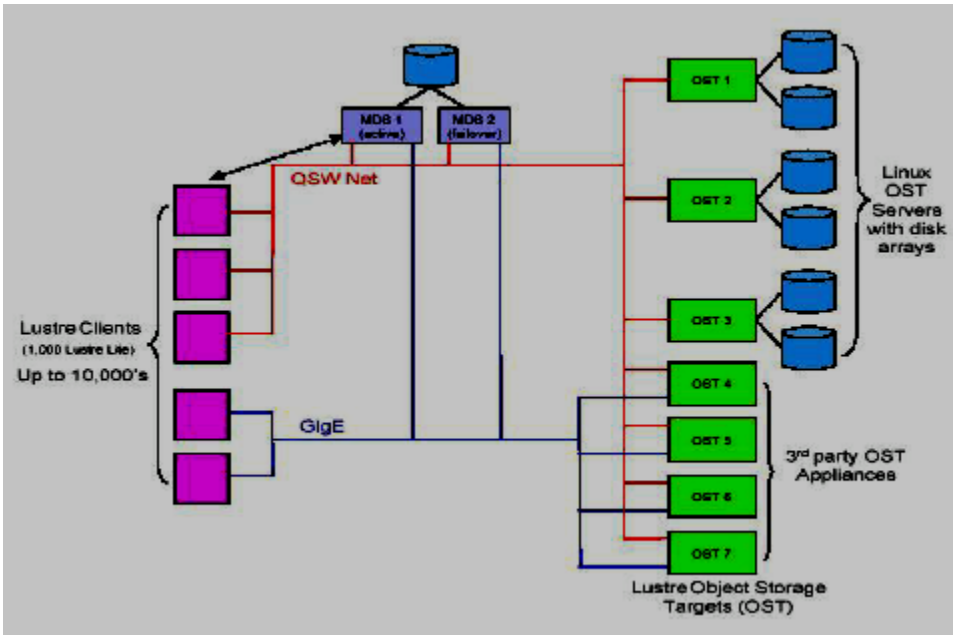

图 1.4 Lustre 系统架构图

早期的 Lustre 文件系统只有两台元数据服务器，极大的限制了 Lustre 文件系统的扩展性。现在虽然对元数据服务器进行了扩展，但是文件到这些服务器的映射采用的是哈希方法，虽然在一定程度了起到了负载均衡的作用，但是当元数据服务器数目变化时，仍需迁移大量的数据。

1.2.6 Google File System

Google File System[11] 不同于以上提到的通用分布式文件系统，它是专门针对 Google 公司内部的需求而设计的一套可扩展的分布式文件系统，用于大型的、分布式的和需对大量数据，大文件进行访问的应用。它运行在大量廉价的 PC 上，仅提供简单而有效的文件复制，一致性保证和容错机制。Google 公司的 Gmail、Video 和 Blog 等都是基于 Google File System 的。

如图 1.5 所示，一个 Google File System 集群由一个 master 和多个

chunkserver 构成，并被许多客户端访问。Google File System 采用元数据与文件数据解耦的架构，master 是文件系统的元数据服务器，chunkserver 是数据服务器。Master 维护文件系统所有的元数据，包括命名空间、访问控制信息、从文件到块的映射以及块当前的位置。同时 master 也控制系统范围内的活动，如块租赁（lease）管理，无用块（orphaned chunks）的垃圾收集，chunkserver 间的块迁移。Master 定期通过 HearBeat 消息与每一个 chunkserver 通信，给 chunkservers 传递指令并收集它们的状态。客户（client）与 master 的交换只限于对元数据（metadata）的操作，所有数据方面的通信都直接和 chunkserver 联系。另外客户和 chunkserver 都不缓存文件数据，客户端只缓存元数据。系统中已有一个 master，极大地简化了元数据管理的设计，但是也使得 master 容易成为系统的瓶颈。考虑到 Google File System 主要用于处理大文件而设计的，客户端与 master 的通讯并不频繁，这种设计是简单而有效的。

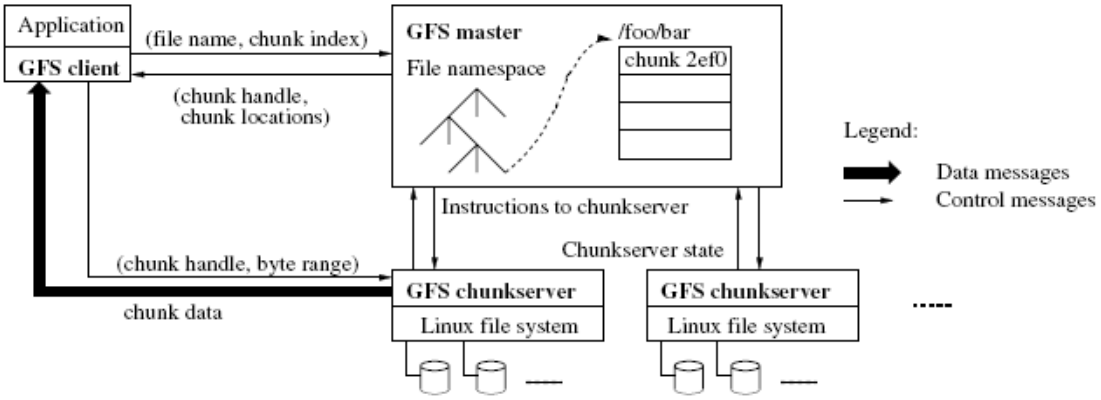

图 1.5 Google File System 结构图

1.2.7 Dawning Cluster File System (DCFS2)

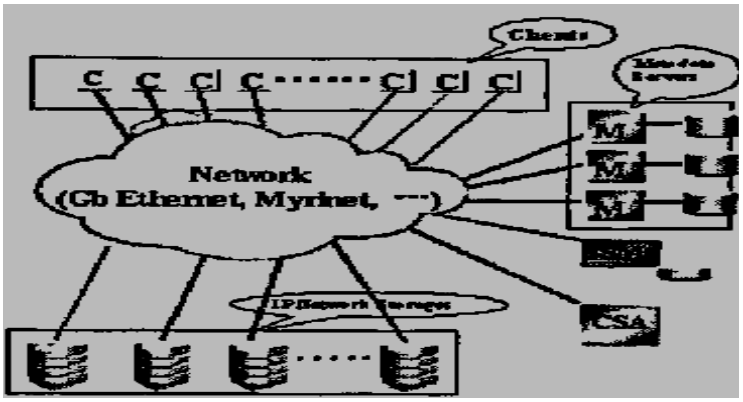

图 1.6 DCFS2 系统架构图

DCFS2[14] 机群文件系统是由国家智能中心开发研制的基于 IP-SAN 的共享存储机群文件系统，采用将文件数据与文件系统元数据分开进行处理的策略，

使用专门的元数据服务器来处理和存储元数据，高效地支持大文件系统，并且保证文件系统的高可用性。DCFS2 的结构如图 1.6 所示。

1.2.8 蓝鲸文件系统(BWFS)

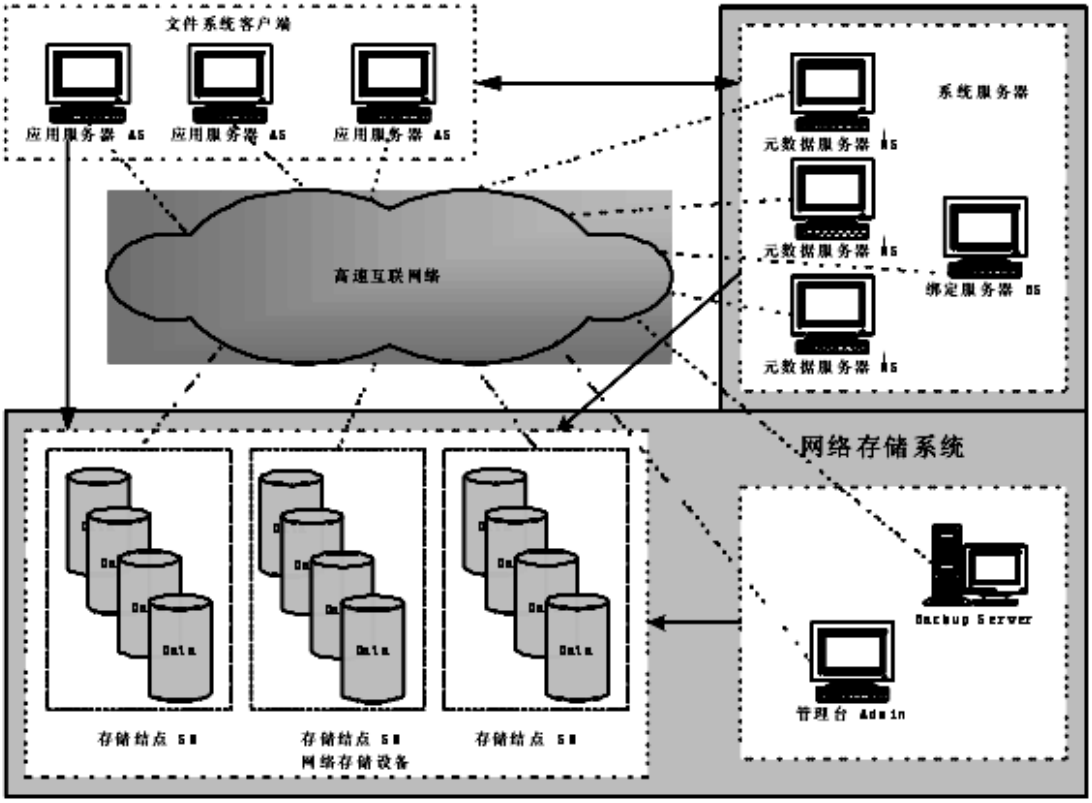

图 1.7 蓝鲸文件系统结构图

蓝鲸分布式文件系统[12] 是由中国科学院计算所工程中心研制的大型网络存储系统，它的存储设备基于网络存储，使用块接口；系统服务器使用专用服务器模式，对外提高文件级的数据接口，因此，蓝鲸系统融合了 IP-SAN 和 NAS 两种结构。它还采用了多元数据服务器的结构来提高系统元数据的处理能力，元数据到服务器间的映射采用了动态映射策略，易于实现服务器间的动态负载平衡，易于扩展和管理。图 1.7 是蓝鲸文件系统的结构图。

1.3 本文的研究内容

本文主要研究分布式文件系统元数据管理的关键问题以及解决策略，本文在对现有分布式文件系统的元数据管理技术进行调研后，针对其中的扩展性问

题、负载均衡问题、数据一致性问题和可靠性问题，设计并实现了新的元数据管理系统。

具体内容如下：

研究了面向节能的元数据服务器集群的负载均衡策略：大规模的应用中，能源问题越来越成为人们关注的焦点。本文在动态元数据管理策略的基础上，提出了一种面向节能的负载均衡策略，通过在负载较低时，主动关闭元数据服务器集中负载，达到整体节能的目的。随着负载的增加，再逐步打开元数据服务器对外提供服务。良好设计的动态元数据管理策略保证了服务器加入和退出时的平滑扩展。

研究了元数据管理的一致性保障策略：分布式条件下元数据可能存在多个副本，许多元数据请求可能涉及多台元数据服务器，本文设计了基于主节点的元数据缓存架构，实现了多副本元数据的更新策略，针对分布式操作，设计并实现了基于两阶段提交的协议来保证其一致性。

研究了元数据管理的可靠性保障策略：本文研究了元数据服务器的节点管理、故障检测以及故障恢复机制，实现了区域自治的节点管理方法和基于日志的可靠性保障策略，保证了元数据服务器单点失效时，能够被快速替换和恢复，失效时系统仍能无间断的提供服务。

利用上述研究的负载均衡策略，一致性保障策略与可靠性保障策略，设计并实现了分布式文件系统 LandFile 的元数据管理系统，包括元数据请求处理和可靠性模块，改进了负载均衡模块。

元数据管理系统的设计目标：

第一：高性能。

每一个完整的文件操作流程都是从对元数据的操作开始，又以对元数据的操作结束。元数据管理系统的性能对文件系统整体性能有着至关重要的影响。大规模的应用中，元数据管理系统面临着大量并发操作、热点数据、突发热点数据、大目录写等诸多情况，很容易使得单台元数据服务器成为系统的瓶颈。元数据管理系统必须设计好集群的架构以及相应的负载均衡算法，保证在面临这些问题时仍能有较高的性能。

第二：高可用。

元数据管理系统需要具有高可用性，作为文件系统的入口，元数据管理系统不能使用的话，存储系统也就没法发挥作用。元数据管理系统应该将单点 MDS 失效的影响减至最小，在单点 MDS 失效情况下能进行透明恢复，并能在失效后恢复一致的状态。在单点 MDS 故障恢复时，应能保证元数据服务不中断。

第三：易扩展。

元数据管理系统应能支持 MDS 的平滑扩展，在大规模的应用中，经常需要增加机器以应对不断扩展的客户需要，而故障问题，亦可能使一部分 MDS 暂时无法服务。元数据管理系统应该能够支持 MDS 的动态增加和删减，提供良好的伸缩性。

第四：透明性。

元数据管理系统应向外界提供标准的统一接口，呈现给用户的必须是单一的系统映像。分布式的元数据操作，元数据服务器的数据转发、负载迁移、故障恢复等应该细节应该向用户隐藏。

第五：易管理。

元数据管理系统应该向系统管理员提供丰富的管理接口，易使用的配置命令和方便的参数调节接口。

1.4 本文结构

第一章简述了本文研究的课题背景及相关的研究内容。第二章介绍元数据管理系统的整体架构以及要解决的问题。第三章主要研究了面向节能的负载均衡策略。第四章主要研究元数据管理的一致性保障策略和可靠性方法。第五章介绍元数据管理系统的设计与实现。第六章将对本文作综述，提出进一步的研究内容。

第 2 章 元数据管理系统

元数据是指文件系统中用来描述目录属性和文件属性等信息的数据，其中，目录属性包括目录的名称、访问控制权限、目录下的文件及子目录信息等，文件属性包括文件名、文件大小、创建时间、修改时间、最近访问时间、文件的用户、访问控制信息等。虽然文件系统中元数据占用的空间往往不到 10%，但统计表明，在对文件系统的所有访问中，对元数据的访问大约占全部访问次数的 50%到 80%[15] 。对文件系统的访问，往往都是从对元数据的访问开始，最终以对元数据的访问结束的，元数据直接关系着整个文件系统的正确性、一致性、扩展性和可用性等，对于文件系统非常重要。文件系统元数据结构的设计与实现的优劣直接影响着文件系统的整体性能，是系统评价和优化的一个重要部分。

本文的研究对象是分布式文件系统，研究的主要内容是分布式文件系统元数据管理的关键技术与实现。本章简要介绍了分布式文件系统元数据管理的关键技术及解决方案。

2.1 分布式文件系统的整体架构

分布式文件系统的系统架构经历了三个发展阶段：传统的集中式客户端/服务器架构、单元数据服务器架构和多元数据服务器架构。

在早期的分布式文件系统中，如 NFS，客户端通过将远程文件映射到本地文件系统那个来达到操作远程文件的目的，所有数据均由文件服务器负责维护。这种模式限制了文件系统的扩展性，无法满足日益正常的容量与性能方面的需求。

通过对文件数据和元数据访问特性的考察，人们发现文件数据与元数据在访问特性上有明显的不同：元数据操作的数据量一般较小，要求访问延迟低，对吞吐量的要求较低，元数据操作不容易并行处理；文件数据的数据量较大，对访问延迟不敏感，对吞吐量有较高的要求，文件数据的访问有很高的并行性。针对这种特性，出现了将文件数据与元数据解耦的系统架构（图 2.1），将元数据操作与文件 I/O 操作路径分离，采用专门的元数据服务器来处理元数据操作。在这种模式下，客户端首先访问元数据服务器，获得相应的元数据后，直接访问存储设备进行文件数据的读写。文件数据与元数据解耦的系统架构减少了元数据服务器的负载，较少了了客户操作的等待时间，对系统的性能有很大提升。

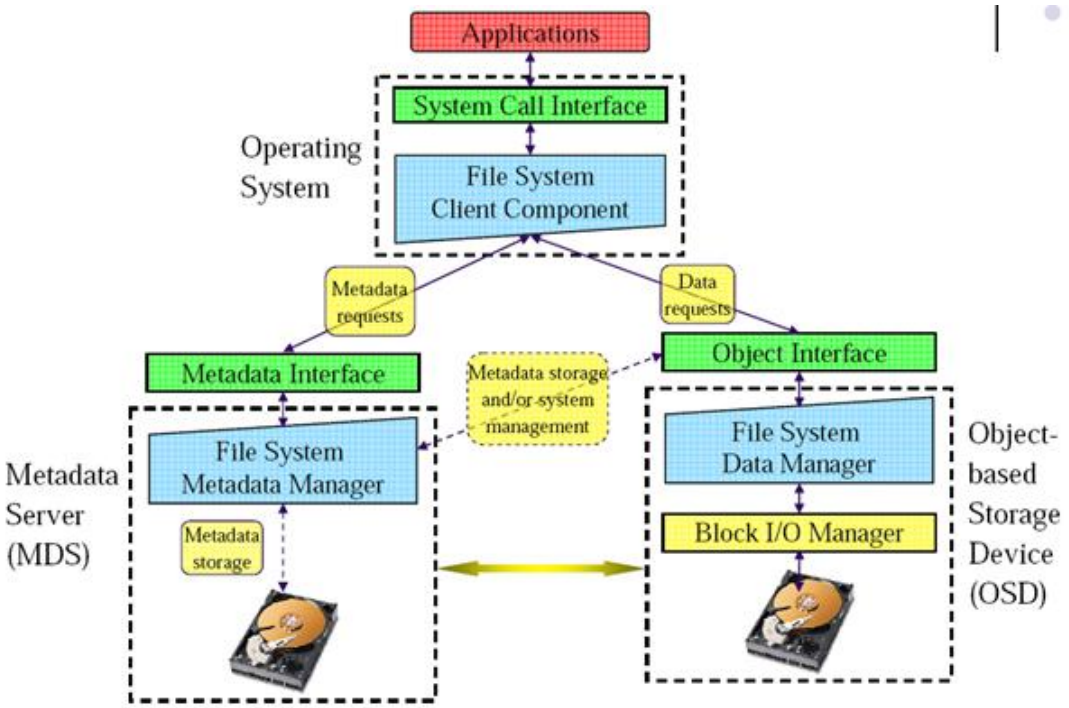

图 2.1 元数据与文件数据解耦的服务器架构

随着分布式文件系统容量和性能的飞速提升，元数据的容量也逐渐增加，PB 级别的文件系统中元数据达到了 TB 级别，单一的元数据服务器已经无法满足需求，人们采用多台元数据服务器构成的集群对元数据进行管理。元数据服务器集群解决了单元数据服务器容量和扩展性不足的问题，但多元数据服务器的存在，使得数据的一致性、负载均衡、可靠性策略都比较复杂。

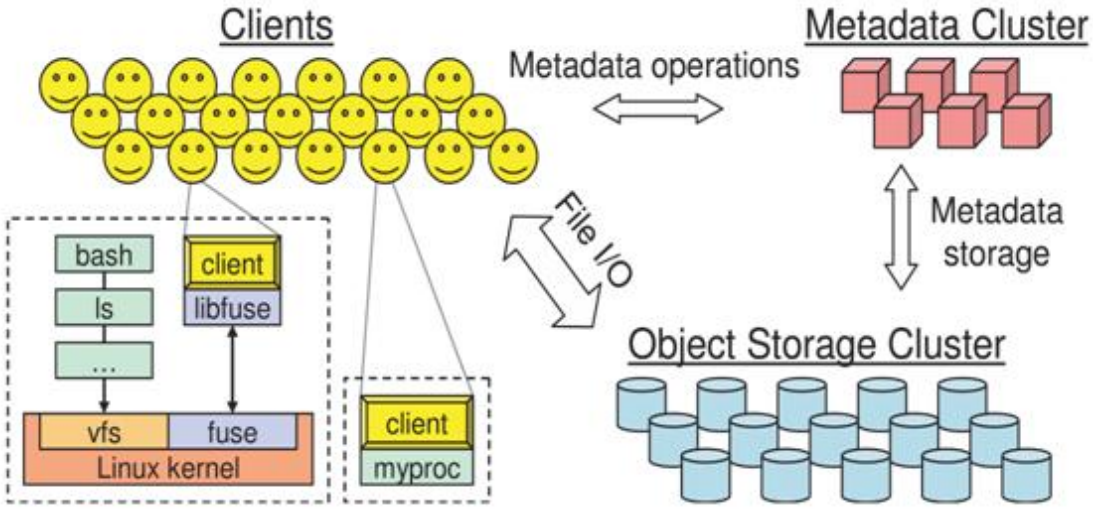

图 2.2 多元数据服务器结构

LandFile 分布式文件系统采用多元数据服务器的架构，整个系统分为三个部分，元数据管理系统，数据存储系统和客户端。三个子系统可分别运行在不

同的计算机节点,也可以多个子系统运行在同一个节点。元数据管理系统负责维持全局明名空间的一致性、文件路径到文件 id 的映射、lookup 、create 、unlink、stat 等与文件元数据相关的操作以及与文件操作中的锁服务。元数据管理系统负责自身的负载均衡与可靠性管理。数据存储系统对外提供数据读写服务，负责自身的负载均衡和故障恢复。客户端分为元数据服务器客户端和存储系统客户端，负责接受用户请求、返回处理结果以及与相应的服务器进行通讯。客户端提供 POSIX 兼容的 UNIX 系统接口。

2.2 元数据管理系统架构

对于多元数据服务器架构，元数据在这些服务器上的组织和划分是决定元数据服务性能和扩展性的重要因素。根据元数据的存储和处理方式，我们把现有的多元数据服务器的系统架构分为两种：分布式存储分布式处理和集中存储分布式处理。

2.2.1 分布式存储分布式处理

分布式存储分布式处理的方式中，元数据按一定的方式分布到 MDS 上，每台 MDS 负责处理存储在其上的元数据请求，元数据与 MDS 的对应关系一旦确立就不会改变；这种方式实现较为简单，客户端查找元数据服务器的速度很快，但是一般比较难以进行负载均衡和扩展。常用的元数据划分方法有静态子树划分和 hash 方法。

静态子树划分方法将文件系统组织成目录树的机构，为每台元数据服务器分配一个或多个目录子树，之后除非人工干预，否则就不再改变。NFS、AFS、Coda 等文件系统都是采用这种方法来管理元数据。该方法实现简单，简化了对元数据的管理，客户端也比较容易查找到目标 MDS，由于按目录树的结构组织，可以充分利用文件系统访问的局部性。其主要缺点是：对部分元数据操作支持较差，例如 rename 操作，当涉及的目录子树在不同 MDS 上时，将会造成大量的数据迁移；难以进行负载均衡，随着文件系统的使用，不同目录子树的流行度差别较大，目录子树的收缩和扩张可能使得有的 MDS 空载而有的 MDS 成为系统瓶颈，即使手动进行干预，需要迁移的数据量也极大；伸缩性能较差，当增加或者删除 MDS 时，将会造成大量的数据迁移，增加的 MDS 需要手工指定其负责的目录子树，不易管理。

静态 hash 方法通过 hash 函数来分配和定位元数据，通常是对原件的全路径进行 hash，Lustre、zFS、Intermezzo 等文件系统采用这种方法。使用 hash 函数可以很快地定位元数据的位置，精心设计的 hash 函数可以使得元数据尽量均

匀的分布在 MDS 上。该方法的缺点是：hash 函数使得同一目录下的文件被打乱分布，难以利用文件系统访问的局部性；hash 的过程中，丢掉了对文件访问权限等信息，要确定用户对文件的操作权限，需要遍历上层目录，导致了操作效率的底下；对于 rename 等操作，需要迁移大量数据；负载均衡性能不佳，虽然元数据被均匀分布了，但是不同的元数据的访问热度不同，仍可能出现热点文件，出现负载不均时，无法通过负载/迁移等策略降低过载 MDS 的负载；系统伸缩性较差，增加或者删除 MDS 时，将会造成大量的数据迁移。上述的缺点很多是因为 hash 方法难以保证严格的文件系统访问的语义，在对文件系统语义要求不高的环境中，例如只要求读写的海量存储系统中，hash 方法仍能起到较好的作用。

华中科技大学的冯丹等人针对静态 hash 算法的缺点，提出 DOIDFH [16] (Directory Object IDentifier and Filename Hashing)算法，对目录对象标识和文件名 Hash 方法，文件元数据根据上层目录的 id 和文件名的 hash 值确定所在的 MDS。该方法还使用访问控制列表来管理文件和目录的访问权限。该方法能避免目录重命名引起的元数据迁移，其仍不能保持元数据的局部性，热点数据可能造成负载失衡也难以消除，MDS 负载均衡，当增删 MDS 时，仍会引起大量的元数据迁移。

2.2.2 集中存储分布式处理

集中存储分布式处理的方式中，元数据保存在共享的存储设备中，元数据和 MDS 的对应关系是动态划分的，每台 MDS 负责缓存一部分目录子树并处理相应的元数据操作；动态划分在出现热点数据或者 MDS 负载过高时可以很方便的进行目录子树的复制和迁移，易于扩展 MDS 和负载均衡，但实现较为复杂。常见的集中存储分布式处理的方式有动态绑定方法和动态子树划分方法。

动态绑定方法[12] 是中国科学院计算技术研究所的蓝鲸网络存储系统使用的方法，它将元数据保存在共享磁盘中，有元数据操作请求时，由绑定服务器（BS）根据约束规则和 MDS 集群的负载状况，指定一台 MDS 负责处理，BS 保证每个元数据有唯一的 MDS 负责。BS 还可以检测 MDS 的负载情况，改变绑定关系，从而达到负载均衡的目的。动态绑定方法扩展性好，负载均衡能力较强。其主要缺点是：每个元数据的请求仅有一台 MDS 来处理，难以应付热点数据，对于在大目录下大量创建文件造成的负载也难以均衡，绑定服务器容易成为系统瓶颈。

动态子树划分方法[13] 是 UCSC 的存储系统研究中心,的 CEPH 文件系统所采用的方法。动态子树划分方法将元数据保存在共享的存储系统中，MDS 对目录子树进行缓存并处理相应的元数据操作，目录子树划分到 MDS 上之后，仍

可以根据负载等情况，动态迁移。为了解决动态迁移造成的 MDS 查询时间过长的问题，客户端对已知的元数据与 MDS 的对应关系进行缓存，这样由于局部性原理，后续的操作直接联系已知的 MDS，大大增加了查询的命中率。由于元数据保存在共享的存储系统中，迁移的只是缓存的部分，大大降低了迁移的数据量，系统的伸缩性较好，可以平滑进行 MDS 的扩展。通过对热点数据的复制，可以良好应对热点数据的读操作，均衡了负载。为了应该大目录下的大量更新操作（例如在某目录下大量创建删除文件），该方法细化了子树迁移的粒度，可以将一个目录下的多个文件集合起来进行迁移。其主要缺点是：复制的存在，使得服务器的一致性保障策略实现比较复杂；动态迁移使得目标 MDS 的查找跳数较多，开销较大。虽然动态子树划分的实现较为复杂，但其良好的扩展性和负载均衡性能使其成为人们选择的热点。

2.2.3 元数据管理系统架构

LandFile 分布式文件系统的 MDS 集群采用集中存储分布式管理的系统架构，元数据保存在共享的存储系统中，各 MDS 上采用动态子树划分的方式缓存部分目录子树。在处理元数据请求时，MDS 集群中没有中心节点，各 MDS 是平等的关系。为了监测元数据的负载信息、进行负载均衡的决策以及进行节点管理，会选出一个班长节点来收集各 MDS 信息。

2.3 元数据服务器集群的负载均衡策略

网络服务中数据的访问往往是不可预测的，而且文件的访问频率随着时间在不断变化。在多 MDS 架构下，MDS 的负载不均衡现象非常容易出现，如何防止系统瓶颈、充分有效利用资源，提高用户体验，一直是人们研究的热点。MDS 集群的负载均衡策略对于文件系统的性能至关重要。

本节主要从负载均衡过程中面临的问题、常用解决方案，负载均衡的目标与评价指标介绍 MDS 集群负载均衡的关键技术。

2.3.1 MDS 集群面临的负载均衡问题

根据造成负载不均衡的原因，我们将元数据管理系统面临的负载均衡问题分为两类：元数据分布不均造成的负载不均衡，元数据访问频率不同造成的负载不均衡。

元数据分布不均造成的负载不均衡：由于不同 MDS 管理的元数据的量不同，管理元数据较多的 MDS 容易过载，管理元数据较少的 MDS 经常空载，造成系统负载不均衡。这种问题在静态子树划分的元数据管理方式中比较常见。

由于目录子树在划分到 MDS 上后随着用户的操作而收缩或扩张，一段时间后各 MDS 上的元数据量可能差别极大，从而造成 MDS 集群的负载不均衡。

元数据访问频率不同造成的负载不均衡：文件的访问频率随时间变化，不同文件的访问频率可能差别很大，在网络应用中，热点文件的访问频率可能是一般文件的上万倍，因此即使在元数据分布均匀的情况下，MDS 的负载仍可能差别很大。这种问题在静态和动态的元数据管理方式中均可能发生，因为其根本原因在于处理一个文件的元数据的 MDS 只有一台。

2.3.2 常用的解决负载均衡问题的方案

对于数据分布不均造成的负载均衡问题，有两种解决方案，一种是采用静态的 hash 方法，使得数据分布均匀，但这种对于元数据访问频率不同造成的负载不均衡效果不大；另一种是采用动态的方法，根据 MDS 的情况及时调整其管理的元数据量，动态元数据管理方式均是采用这种方法。

对于数据访问频率不同造成的负载均衡问题，根据数据访问的类型，有可以分为两种情况：由于大量读操作造成的负载不均衡和由于大量写操作造成的负载不均衡。

在大规模的网络应用中，经常出现突发的热门数据，如视频、图片和新闻等，对于这些数据，大量的操作是读，很少对其更新。因此可以采用创建副本的方式，在多个 MDS 上复制相应的元数据，以达到分流负载的目的。复制的方式虽然解决了热门数据的问题的，但各副本的数据一致性维护比较复杂。

在科学计算中，经常碰到在一个目录下反复创建和删除大量文件的操作，由于大部分文件系统并不支持并行写的操作（虽然 Coda 可以并行写，使用冲突消解机制来解决写冲突，但经常碰到无法进行冲突消解，需要人工干预的情况），这种对目录的更新操作只能集中在一台 MDS 上，从而造成负载的难以均衡。CEPH 文件系统对这种情况进行了考察，提出了目录分片的结构细化了数据迁移的力度，较好的解决了这一问题。CEPH 文件系统在目录与目录项之间，增加了一层称做目录分片的结构，目录分片是部分目录项的集合，目录内仅记录目录分片的信息，目录项的增删都是在目录分片内进行的，目录分片负责目录项增删时的权限检查。目录分片作为数据迁移基本单位，单个目录下出现大量增删操作造成负载过高时，可以迁移该目录下的部分目录分片到别的 MDS 上。当单个目录分片的内容过多是，可以分裂为多个目录分片；单个目录分片内容较少时，可以与其他目录分片进行合并。

2.3.3 负载均衡的目标与评价指标

MDS 集群的负载均衡目标从服务器的角度来说，常见的有元数据的均匀分

布，MDS 负载的均匀分布，MDS 吞吐量最大；从客户端的角度来说，常见的有使元数据操作相应时间最小，MDS 集群可服务客户数最多。根据不同的均衡目标收集合适的元数据和 MDS 信息，选择合适的 MDS 进行数据的复制和迁移。

对于负载均衡算法，主要从三个方面进行评价：均衡效果，稳定性和资源消耗。

从负载均衡的效果上说，负载均衡算法应该是有效的。负载均衡算法应该能应对各种情况，有效地消除 MDS 集群的负载不均衡情况，负载均衡后系统应能应得到提升，在工作负载较大时，应保证各节点平均分配任务。

稳定性：负载均衡算法应该是稳定的，不会出现抖动现象。在异构 MDS 中，MDS 性能可能差别较大，当负载从能力强，高负载的 MDS 迁移到能力弱，低负载的 MDS 上是，很容易造成能力弱的 MDS 过载，从而引发新一轮的迁移。这种相同的元数据的在小部分范围内的 MDS 间反复迁移的现象被称为抖动现象。要防止抖动的仿生，负载均衡算法应该考虑各 MDS 的性能，充分预测负载迁移后的稳定性，选择稳定的策略进行负载迁移。

资源消耗：负载均衡的过程中要消耗 CPU、内存，网络带宽等一系列的资源，负载均衡过程中，对一些元数据操作可能会有影响，增加其等待时间，负载均衡算法应尽量减少这种消耗。一般来说采用的原则是在效果相近的情况下，选择尽量少的元数据进行迁移，迁移时，选择尽量近的 MDS 作为目标 MDs，从而减少资源消耗，节省时间。

2.4 元数据管理的一致性保障策略

元数据管理的一致性问题为缓存的的一致性。在分布式文件系统中，元数据主要存在在三个地方：客户端缓存，MDS 缓存，存储设备。为了提高性能，通常在客户端缓存一部分元数据和服务器的信息，从而避免耗时较长的网络通讯。负载均衡过程中，为了分散对热门数据的访问，许多 MDS 会保存热门的元数据的副本以应对客户端的读操作。分布式文件系统中，MDS 处理的数据往往存储在异地的服务器。必须保证元数据在客户端缓存、MDS 缓存和存储设备上的映像的一致性，才能对外提供正确的服务。

通常有两种方法保证缓存的一致性，一种是在缓存更新时，通知其余缓存更新，由于许多缓存可能并不需要了，这种方式会导致较多的不必要的更新，代价较大；另一种方式是缓存更新时，其余缓存将会失效，其余节点在需要时重新获取数据，这种按需更新的方式可以避免不必要的更新，受到人们的欢迎。

2.5 元数据管理的可靠性策略

随着分布式文件系统容量的增加，元数据服务器的数量也在逐渐增加，元数据服务器失效的概率也越来越大。元数据管理的可靠性研究主要包括三个方面：元数据服务器的失效检测技术，元数据服务器的故障处理与恢复机制以及元数据管理可靠性的评价方法。

元数据服务器的失效检测通常使用心跳技术。元数据服务器定时向监视节点发送心跳消息，来表明自己依然正常工作。

元数据服务器的故障处理与恢复主要使用两种方法：一是备份机制，正常情况下，主备份所在的元数据服务器（主元数据服务器）处理请求，并在数据更新时同步更新各备份；当主元数据服务器失效后，由备份所在的元数据服务器（从元数据服务器）接管其工作，并对主备份进行恢复。在多元数据服务器架构中，每台服务器都有专门的服务器的话，存着着资源严重浪费的问题。一般系统中所有元数据服务器既是主元数据服务器，又是从元数据服务器，当主元数据服务器出现故障时，元数据服务器即可暂时接管其任务，保证服务的持续性。使用这种方式，不需要专用的备份服务器，极大的减少了元数据服务器的数量，但当故障发生时，对应的从元数据服务器的负载将相当于原本两台服务器的负载，负载过大，造成系统瓶颈。备份机制可以很好的保证磁盘数据的可靠性，但要保证元数据服务器缓存内容的可靠性，则需要使用日志技术。

日志技术是指元数据服务器利用日志记录元数据操作的信息，当元数据服务器出现故障时，利用日志来恢复元数据服务器。日志从使用方式上分为回滚日志和写前日志。回滚日志在操作时记录日志，当系统故障时，利用日志将系统回滚到一个一致的状态；写前日志则要求数据在写入磁盘前必须先写入日志，可以不必应用到实际系统中，而在需要时，再一次写入磁盘。使用写前日志可以极大的减少数据的通信量与等待时间，提高系统性能。日志技术在本地文件系统中得到了广泛的应用，但在多元数据服务器的架构中，一个元数据操作可能需要多个元数据服务器参与，需要使用分布式日志来解决这类问题。

元数据管理的可靠性主要由正确性，性能，故障恢复时间，存储空间等参数来评价。

2.6 本章小结

分布式文件系统的架构决定了元数据管理系统的结构，从而影响负载均衡策略、数据一致性保障策略以及可靠性策略的选择和实现。本章主要介绍了元

数据管理系统面临的问题，对元数据管理中的负载均衡策略、数据一致性保障策略以及可靠性策略的技术难点，关键技术和解决方案进行了描述，详细方案在后续章节中逐步阐述。

第 3 章 面向节能的负载均衡策略

本章提出了一种面向节能的负载均衡策略，通过在整体负载较低的时候，关闭部分元数据服务器以达到节能的目的。面向节能的负载均衡策略实际上是一种自适应的负载均衡策略，它根据服务器负载情况和用户请求的时间统计信息等来动态决定选择合适的负载均衡策略。实验表明，面向节能的负载均衡策略可以极大程度地降低能耗，对系统整体性能影响较小，节能模式的切换时间较短，切换时系统过渡平滑。

3.1 节能问题的提出

近年来，我国信息化高速发展带来的服务器能耗问题越来越受到关注。快速增长的服务器带来了巨大的电能消耗。IDC 报告显示，如果一个服务器的功耗为 300 瓦，持续运行 1 年，以每度电 0.64 元计算（工业用电），每年的电费为 1659 元。当前数据中心所需要的计算能力正在大幅增加，同时随着机架服务器、刀片式服务器的采用，设备密度极大增加，导致了功耗的增加。以 100 个插满机架式服务器的服务器机架为例，机架服务器耗电 50 千瓦，而散热所需的冷却耗电量几乎等同于计算机硬件自身的耗电量，因此冷却设备还需另耗电 50 千瓦。以每度电 0.64 元计算，一年无故障运营 100 台机架服务器将花费大约 55 万元电费。据统计，数据中心的电力消耗以每年 15%～20%的速度在增长，而与此同时能源的价格也在飞涨。数据中心每年所支付的电费，已经超出了 IT 硬件支出，或者说数据中心总拥有成本的四分之一要花在冷却方面。Google 公司的单项最大支出就是它那庞大服务器群的电费。正因为以上原因，Google 不得不把自己的处理中心部署在水电站附近，因为那里有充足的电能。微软在西伯利亚的伊尔库茨克建立，就是希望利用西伯利亚寒冷的天气提供天然的服务器冷却环境，以节约电力费用。

针对这种情况，许多服务器厂商如 INTEL，IBM，浪潮等纷纷提出了自己的节能服务器。作为服务器中的重要部分，存储领域也在关注绿色存储的问题。存储网络行业协会（SNIA）专门成立了绿色存储任务组和绿色存储技术工作组来解决存储中的能耗问题。

许多公司一直在部署大规模非活动磁盘阵列（Massive Arrays of Idle Disks，MAID）技术，以便减少数据存储能源费用。在不使用时，该技术通过关闭耗电量大的磁盘驱动器，让它们处于睡眠状态，从而达到降低能耗的目的。MAID

极大地满足了目前流行的“绿色存储”所要求的绿色、环保和节能的特点，具有 MAID 功能的存储设备可以大幅度节约对电能的消耗，特别是在大容量的数据中心或数据仓库系统中，能够减小机房供电系统的压力。然而由于 MAID 存储的工作原理是使长时间无访问请求的硬盘处于 Power OFF 状态，以达到绿色节能和延长硬盘寿命，因此 MAID 存储适用于访问频率低、数据即时可得性要求低的大容量近线存储（Nearline Storage）设备和分级存储，尤其适用在以数据恢复为目的的数据备份和数据归档等应用上。并不适用于高性能实时访问系统，或访问时间间隔较小的应用系统。

现有的节能策略大都是针对硬件设计，通过使用节能芯片或者关闭电源以减少能耗，而主要集中于软件层面的分布式文件系统，其节能策略却少有提及。

3.2 节能问题的分析

3.2.1 分布式文件系统节能遇到的问题

文件服务的特性决定了分布式文件系统难以关闭存储服务器以实现节能。web 服务器一般看重的是其计算能力，其对数据的需求往往很少从本地获取，关闭部分服务器后并不影响系统的可用性。分布式文件系统则不一样，其对外提供的存储服务要求随时可用，关闭掉任意一台存储服务器，都会使得某部分数据无法访问。

分布式存储的元数据决定了难以关闭元数据服务器以实现节能。在分布式存储分布式处理的元数据管理方法中，每台元数据负责存储和管理一部分数据。关闭任何一台元数据服务器都可能导致某部分元数据不能正常对外服务。

集中存储分布式处理的元数据管理系统架构中，元数据保存在共享的存储系统中，MDS 只负责缓存元数据，部分 MDS 的关闭对系统可用性并无影响。本章主要讨论集中存储分布式处理架构下的元数据服务器集群的节能策略。

3.2.2 MDS 集群节能的理论基础

通过对现有大规模网络应用访问特性的观察，我们发现网络服务器的能力是存在着严重浪费的。大部分的网络应用，访问的请求数目随时间变化差别极大，图 3.1 描述了某 web 服务器集群的工作特性图，可以看出高峰期的访问量是低谷期的数十倍甚至上百倍，而且这种差异随着网络应用的规模的扩大的。网络服务器集群的设计都是以高峰时期的负载为标准的，也就是说在低谷期存在着严重的服务器能力的浪费。这个时期完全可以关闭部分服务器以节省能源。MDS 集群同样存在着服务器能力浪费的问题。

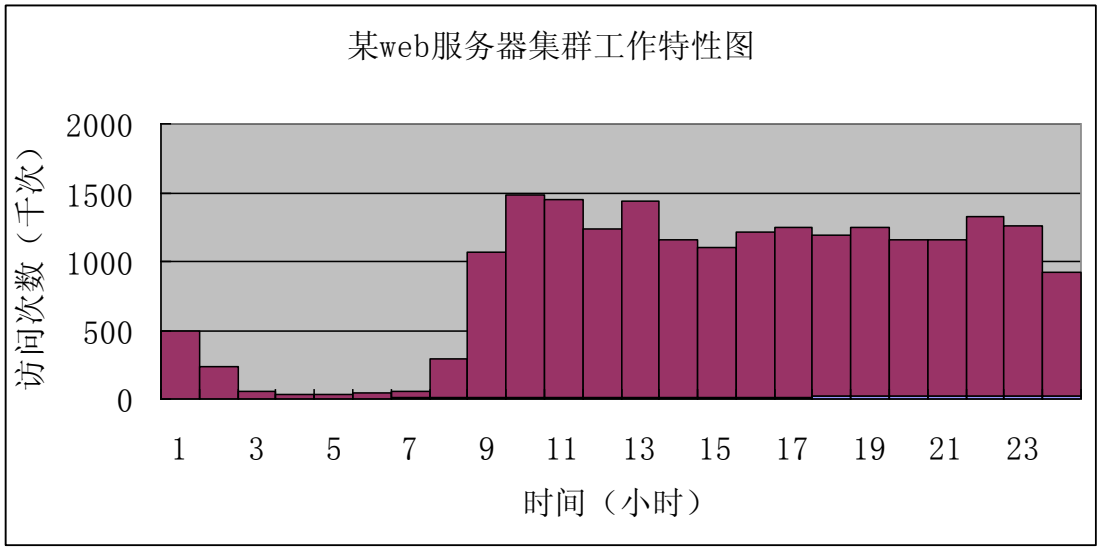

图 3.1 某 web 服务器集群工作特性图

	曙光I620r-F	曙光A620r-FX	戴尔PE 1950	联想 万全 R525	浪潮 NF285E	惠普ProLiant DL380 G5	华硕 RS160-E5	联想 万全 R525（更换CPU）
待机功率	11.2W	11W	27W	37W	26W	21.6W	5.4W	34W
空载功率	218W	203W	243W	221.2W	200W	161W	152W	201W
满载功率	253-268W	251W-270W	288-302W	273-286W	233-245W	185-192W	196-220W	250W-259W
1小时耗电	0.26KWh	0.25KWh	0.29Kwh	0.27Kwh	0.23Kwh	0.18Kwh	0.2Kwh	0.24Kwh

图 3.2 服务器能耗与负载的关系

通过对服务器负载与能耗关系的观察，我们发现服务器的能耗与负载并不是线性关系。图 3.2 描述了不同服务器在各种负载情况下的负载，其中空载功率为服务器开机后没有施加负载时的功率，此时的 CPU 占用率一般低于 5%，满载功率为服务器满负载时候的情况，此时 CPU 占用率一般超过 95%。从图中看出，在待机状况下，功耗最低；从待机转到空载状态，功耗突然增加，空载功率一般是待机功率的十余倍；从空载状态转移到满载状态，能耗只是稍微增加，增加的幅度并不大。因此将多台服务器的负载转移到一台服务器上，虽然

会造成能耗的增加，但增加的幅度并不大。而转移出负载的服务器可以进入待机状态，从而大到节能的目的。

3.2.3 MDS 集群节能策略的设计原则

MDS 集群的节能策略的设计原则是：

（1） 有效性：必须保证有效降低 MDS 集群能耗。

（2） 对性能影响小：MDS 集群的目标就是为了应对大规模的数据访问，处于节能模式时，MDS 集群应仍能正常、高效地处理客户请求，在节能模式和正常模式间切换时，应尽量平滑过渡，不影响用户体验。

（3） 简单易实现： MDS 集群的节能策略应比较容易实现。如下的一种节能策略虽然理论可行，但实现太过复杂，实不足取：为了保证元数据的可靠性，通常一份数据在多台 MDS 上保留副本，因此可以关闭一些 MDS，将对其上的元数据访问请求转移到备份所在的 MDS 上即可；然后为了保证数据副本数符合可靠性的要求，在 MDS 关闭期间，将更新记录在某个地方，在 MDS 恢复时再将更新刷新到原 MDS。

3.3 相关定义

MDS 集群采用集中存储分布式处理的系统架构，在正常模式下，MDS 动态子树划分的方法进行负载均衡。当整体较低时，采用面向节能的负载均衡策略，关闭部分 MDS，达到节能的目的。以下主要介绍负载均衡算法所用的一些参数的定义。

MDS 负载（L）：从当前研究可知，准确的表达当前服务器的负载时非常困难的。处理来自用户的请求作业需要消耗包括 CPU、内存、磁盘空间，网络带宽等各种计算机资源。如何选择合适的参数来表达节点的负载对于负载均衡算法有重要意义。良好的负载指标应当满足有效性和易获取两个要求。在元数据服务器中，需要消耗 CPU 资源来处理用户请求，消耗内存资源来缓存元数据信息，消耗网络带宽进行通信。因此我们采用 CPU，内存和带宽占用情况的加权和来定义元数据服务器 i 的负载。其中 l_{cpu} 、 l_{mem} ， l_{bw} 分别为 CPU、内存和带宽的占用情况， α 、 β ， γ 为相应的加权系数， $\alpha+\beta+\gamma=1$ ，根据应用类型的不同通过反复试验得出其具体值。容易看出 L_i 的取值范围在[0,1]之间， L_i 越大，表明 MDS 的工作负载越大， L_i 为 0 时表示 MDS 空载， L_i 为 1 是表示 MDS 满负荷。

$$L_i = \alpha * l_{cpu} + \beta * l_{mem} + \gamma * l_{bw} \qquad (3.1)$$

元数据的热度（p）：目录树中的每个节点，都有一组访问热度，根据访问类型的不同，访问热度分为读热度和写热度，读热度反映的是 read、stat 等读操作的频度，写热度反应的是 mkdir, write 等写操作的频度。元数据的总热度为读写热度之和。每当有来自用户的元数据请求时，对应节点的元数据相应类型的热度增一。考虑到不同时间内的元数据请求对热度的影响应该是不同的，元数据的热度会随时间衰减。元数据访问的时间间隔越长，对热度的影响应该越小。对某一元数据访问时，其在目录树上的祖先节点的热度亦会受到影响，即热度的更新会向上传播，传播的量逐层衰减，衰减到一定值后就不再向上传播。综合以上考虑，得到元数据节点的热度计算公式如下所示，其中 Δt 为当前时间与上次计算热度的时间差，f 为衰减函数，n 为被访问元数据与祖先节点的目录层次差。

$$\begin{aligned} p_{new} &= p_{old} * f(\Delta t) + 1 \\ p_{ancestor\_new} &= p_{ancestor\_old} * f(\Delta t) + 1/2^{n} \end{aligned} \tag{3.2}$$

MDS 的元数据总热度（P）：MDS 的元数据总热度为其缓存的所有元数据热度之和，每当与元数据热度发生变化时，更新元数据的总热度。元数据的总热度亦随时间衰减。第 i 台 MDS 的总热度 P_i 的更新公式如公式（3.2）所示，其中 Δt 为当前时间与上次计算热度的时间差，Δp 为元数据节点的热度变化。

$$P_{i\_new} = P_{i\_old} * f(\Delta t) + \Delta p \tag{3.2}$$

MDS 能力（s）：MDS 能力的评估与负载的评估一样，一般很难得到准确的表达。一般情况用其处理器能力、内存容量和网络带宽等参数的加权和来表示，但这种方法的加权系数需要反复试验才能得出，而且随着应用和 MDS 机器配置的不同，加权系数还有改变。考虑到节点能力反映的是节点处理客户端请求的能力，MDS 的元数据总热度反映了处理客户请求的数量，MDS 负载反应了处理请求消耗的资源，本文用这两个参数来定义 MDS 能力。第 i 台 MDS 的能力 C_i 为 MDS 的元数据总热度（P_i）与 MDS 的负载（L_i）的比值。C_i 的物理意义为 MDS 满负载时（$L_i = 1$）可以处理的用户请求数。容易看出处理相同数目的元数据请求，产生的负载越小，节点能力越强；相同的负载程度能处理的元数据请求数越大，节点能力越强，这是与实际状况相符的。

$$C_i = P_i / L_i \tag{3.3}$$

多台 MDS 的整体负载情况（O）：通过对多台 MDS 的负载情况的汇总，可以得到这些 MDS 的整体负载情况。当 MDS 能力相同时，整体负载情况为各 MDS 的平均值。考虑到 MDS 能力的差异性，我们使用多台 MDS 负载的加权平均值来刻画其整体负载情况，加权系数为相应 MDS 的能力值。将公式 3.3 代

入公式 3.4，我们得到公式 3.5，由于C_i的物理意义为 MDS 满负载时（$L_i=1$）可以处理的用户请求数，$\sum C_i$即为所有 MDS 满负载时可处理的请求数，$\sum P_i$为当前 MDS 处理的请求数，两者相除确实可以反映出当前 MDS 集群的负载程度。

$$O=\frac{\sum_{i=1}^{n}C_iL_i}{\sum_{i=1}^{n}C_i}$$，其中n为集群中MDS数目　　（3.4）

$$O=\frac{\sum_{i=1}^{n}C_iL_i}{\sum_{i=1}^{n}C_i}=\frac{\sum_{i=1}^{n}P_i}{\sum_{i=1}^{n}C_i}$$，其中n为集群中MDS数目　　（3.5）

传输代价（cost）： MDS 节点间传输单位数据需要花费的代价。大部分节点间传输代价的计算在两台服务器有数据交换的时候捎带完成，考虑到网络结构的稳定性，传输代价的更新并不频繁。传输代价还可以通过手动配置直接指定，对于新加入的 MDS，通过指定其到某些 MDS 的传输代价，可以加快其融入 MDS 集群的速度。

MDS 的分区：通过对传输代价进行聚类，将 MDS 分成不同的区域，区域内 MDS 间的传输代价明显小于同区域外 MDS 通信的传输代价。区域内距离聚类中心最近的 MDS 作为决策 MDS，负责收集局域内 MDS 的信息和迁移决策。

3.4 面向节能的负载均衡策略

现有的 MDS 集群负载均衡策略的目标大多为均匀分布负载，避免系统瓶颈以达到提高性能的目的。当 MDS 集群整体负载较轻，不存在过载服务器或者 MDS 间的负载基本均衡时，负载均衡过程则基本不再实施，此时的服务器计算能力存在大量的浪费，而能耗与满载时相比差别不大。

面向节能的负载均衡策略是建立在动态子树划分的元数据管理方式上，系统轻载时，通过对已有的子树迁移算法稍加改造，将轻载 MDS 的负载迁移到能力强的合适的 MDS 上，达到集中负载，节省能源的目的。

3.4.1 节能模式的触发条件

各 MDS 定期向班长节点报告自身的负载与热度信息，当班长节点发现 MDS 集群中没有过载的 MDS，而且整体的负载状况符合节能模式的触发条件时，将会启动负载汇聚操作，将轻载 MDS 的负载迁移到能力强的 MDS。

节能模式的触发条件包括两个：MDS 集群的整体负载水平和当前时间。MDS集群的整体负载水平持续一段时间低于预先设定的阈值时，可以进行MDS负载的合并。将当前时间作为节能模式的触发条件则是为了防止在预期负载较高的时段进行迁移，管理员可以手工设定禁止进入节能模式的时间，也可以自动获取。MDS 集群通过对历史负载的分析，可以得到负载随时间的分布图，从而决定在合适的时间段进入节能模式。

3.4.2 迁移 MDS 的选择

迁移 MDS 的选择包括迁出负载的 MDS 和迁入负载的 MDS。考虑到合并MDS 将会造成 MDS 整体负载水平的提升、集群负载分布不均和消耗网络资源，我们选择迁移 MDS 时有两个原则：使 MDS 集群的整体负载水平提升最小，MDS 集群负载的不均衡度增加最少和迁移的数据尽可能的少。

首先考虑第一个原则：使 MDS 集群的整体负载水平提升最小。由公式 3.5 可知，MDS 集群的整体负载水平和系统处理的总用户数（$\sum P_i$）与 MDS 集群机器的能力有关。可以认为在一定时间内，系统处理的总用户数 $\sum P_i$ 基本保持不变，则将第 j 台 MDS 移出系统后，系统的总体负载 O_j 如公式 3.6 所示，在 $\sum P_i$ 和 $\sum C_i$ 不变的情况下，可知 C_j 越小，O_j 越小。我们可以得到，将机器能力最弱的 MDS 移除集群，MDS 集群的整体负载水平提升最小。因此迁出 MDS 即为能力最弱的 MDS。

$$O_j = \frac{\sum_{i=1}^{n} P_i}{(\sum_{i=1}^{n} C_i) - C_j}，\text{其中n为集群中MDS数目} \qquad (3.6)$$

公式 3.6 在 MDS 能力相同时，将不能发挥作用。此时我们考虑第三个原则，根据一般情况下负载随元数据量的增加而增长，选择负载最低的 MDS 作为迁出 MDS。

考虑第二个原则：使 MDS 集群的负载分布尽量均衡。我们定义 MDS 集群的负载均衡度 B 如下：

$$B = \sum_{i=1}^{n} \frac{|L_i - \bar{L}|}{\bar{L}} = \sum_{i=1}^{n} \frac{|L_i - O|}{O}，\text{其中}n\text{为集群中MDS数目} \qquad (3.7)$$

可知当 MDS 的负载相同，均为平均负载时，B 为 0，此时系统处于负载均衡状态；当 MDS 集群的负载全部集中在能力最弱的 MDS 上时，系统不平衡度最高，此时 B 的取值如公式 3.8 所示。

$$B = n - 2 + \sum_{i=1}^{n} C_i, \text{其中} n \text{为集群中MDS数目} \tag{3.8}$$

将第 j 台 MDS 的负载转移到第 k 台 MDS 上后，集群的负载均衡度 B_{jk} 如公式 3.9 所示，式中的 O' 为转移后的集群平均负载。假定在一定时间内系统的总负载不变（待机 MDS 的负载会等量转换迁入 MDS 中），则由公式 3.5 可知，迁出 MDS 确定后，O' 的值即为定值，因此公式 3.9 中的前一项为固定值，可知为了使转移后的负载均衡度最小，应尽量使后一项的值最小。

$$\begin{aligned} B_{jk} &= \sum_{i=1, i\neq j,k}^{n} \frac{|L_i - O'|}{O'} + \frac{|L_k + \frac{C_j L_j}{C_k} - O'|}{O'} \\ &= \sum_{i=1, i\neq j}^{n} \frac{|L_i - O'|}{O'} + \frac{|L_k + \frac{C_j L_j}{C_k} - O'|}{O'} - \frac{|L_k - O'|}{O'} \\ &= \sum_{i=1, i\neq j}^{n} \frac{|L_i - O'|}{O'} + \frac{|L_k + \frac{C_j L_j}{C_k} - O'| - |L_k - O'|}{O'} \end{aligned} \tag{3.9}$$

（其中n为原集群中MDS数目）

定义 T_{jk} 如公式 3.10，迁入 MDS 即为使得 T_{jk} 最小的 MDS。

$$T_{jk} = |L_k + \frac{C_j L_j}{C_k} - O'| - |L_k - O'| \tag{3.10}$$

然而仅仅考虑 T_{jk} 的取值，很容易将负载能力较弱的 MDS 选为迁入 MDS，而这样的 MDS 可能在后边几次的节能过程中被选为迁出 MDS，这中元数据在同一 MDS 上迁入又很快迁出，很容易造成算法的抖动。

为了解决这一问题，我们在计算 T_{jk} 时，将会先从 MDS 集群中按照能力从弱到强的顺序剔除 x 台能力较弱的 MDS，然后在剩下的 MDS 中计算 T_{jk} 的值，选择 T_{jk} 最小的 MDS 作为迁入 MDS。为了计算 x 的取值范围，我们首先计算节能模式下 MDS 的预期规模 m，对于集群中元数据服务器的能力按照从大到小重新排列，记为 c_n，记节能模式下 MDS 集群能接受的最大整体负载为 O_{max}，则 m 为满足公式 3.12 的正整数。m 的物理意义为，在当前的访问情况下，MDS 集群能接收的整体负载水平的条件下，需要的最少的 MDS 数。

$$O_{max} * \sum_{i=1}^{m-1} c_i < \sum_{i=1}^{n} P_i <= O_{max} * \sum_{i=1}^{m+1} c_i \tag{3.12}$$

$$x = n - m \tag{3.13}$$

x 的值如公式 3.13 所示，可知修正后的策略可以完全避免抖动。

当各 MDS 能力相同时，公式 3.11 的第三个公式将无法区分多个满足条件的 MDS，此时选择满足条件的负载最低的 MDS 作为迁入 MDS。

3.4.3 负载均衡的基本流程

班长节点实时接收各 MDS 上报的信息，各 MDS 上报的信息包括负载和 MDS 的总热度，当检测到有 MDS 的负载持续一段时间超出阈值 T1（MDS 过载阈值），则启动目录子树的迁移进程；若 MDS 集群中没有过载的 MDS，而且 MDS 集群整体的负载状况小于指定阈值 T2（MDS 集群轻载），而且根据以往的统计记录，在当前以及未来的一段时间 t 内，MDS 集群都将处于轻载状态，则符合节能模式的触发条件，启动 MDS 的会聚操作，将两台 MDS 的负载集中都一台 MDS 上，然后关闭另一台 MDS。班长节点还负责定期将当前时刻 MDS 集群情况记（包括 MDS 的总热度和 MDS 集群的整体负载）记入系统，以用于预测并指导以后的负载均衡策略。

MDS 会聚的操作主要分为以下几个步骤：

（1） 准备

a) 班长 MDS 根据收集到的信息，选择节点能力最弱的 MDS 作为负载的迁出节点 SendNode。弱节点能力完全相同，则选择负载最低的 MDS 作为迁出节点 SendNode。

b) 计算当前负载下 MDS 集群的预期规模 m，以及剔除的 MDS 数 x。

c) 从候选迁入 MDS 中按能力从小到大剔除 x 台 MDS。

d) 计算关闭节点 SendNode 后预期的系统的总体负载 O'。

e) 对剩余的 MDS 计算相应的 T_{jk} 的值，取使 T_{jk} 最小的 MDS 作为负载的迁入节点 ReceiveNode。

f) 通知 SendNode，执行迁移策略后关闭。

（2） 迁移

a) SendNode 收到迁移通知后，开始执行迁移并准备关闭

b) SendNode 的上缓存的元数据包括两个部分：别的 MDS 负责的元数据在 SendNode 的副本和 SendNode 本身管理的元数据。对于前一种元数据，应通知其主节点，SendNode 此处的副本已经失效；对于后一种元数据，SendNode 首先将未写入磁盘的更新刷新到磁盘，然后将这些元数据打包发往 ReceiveNode。MDS 在关闭前，对于 MDS 正在服务的客户端，应该关闭其已有连接，不再接受在相关元数据上的更新。对于缓存中尚未写入磁盘的更新，应根据日志刷新到磁盘。

c) SendNode 首先关闭客户端的连接，根据缓存中保存的客户端信息，向客户端发送会话结束的消息，强制关闭会话。客户端会在后来的重试信息中，找到新的 MDS 处理请求。
d) 刷新日志，将日志中对元数据的更改持久化到硬盘中。
e) ReceiveNode 要在缓存中构建迁移来的目录子树，需要有根节点到目录子树根节点的元数据信息，这些需要向相关元数据的主节点请求得到。SendNode 首先向 ReceiveNode 发送要迁移目录子树的路径，ReceiveNode 根据这些路径信息，向相关的 MDS 请求数据。
f) ReceiveNode 向相关 MDS 请求到信息后，通知 SendNode 可以发送目录子树。
g) SendNode 将自己负责管理的目录子树加锁打包后，发送到 ReceiveNode。
h) ReceiveNode 接收到目录子树后，在缓存中重新构建目录子树。
i) 向上级目录子树更新迁移目录子树的信息，ReceiveNode 开始开放迁移来的目录子树，对外提供服务。
j) ReceiveNode 向班长节点更新自己的热度和负载信息，迁移完成。

（3） 结束

a) SendNode 做结束前的清理工作，主要包括通知副本失效，关闭日志记录，关闭读写接口，关闭消息连接。
b) 对于缓存在 SendNode 上的元数据副本，应通知其主 MDS 副本已失效，相关的元数据的主 MDS 会在缓存中更新副本的分布状况，以后有关该元数据的请求就不会分流到 SendNode 上来，相关元数据的更新也不会告知 SendNode。之所以不把这些副本同样打包发送到 ReceiveNode，是因为包括这些数据的话，发送的数据量就会增大许多倍，而且 ReceiveNode 上边可能已经有了部分数据的副本了，而且有些数据的副本在一定时间内未必需要，综合以上因素，对于这些副本数据的处理就是直接丢弃。
c) 将日志刷新到磁盘，关闭日志的读写接口。
d) 关闭磁盘的读写接口。
e) 向班长 MDS 更新自己状态，标记为已关闭。
f) 关闭消息读写接口，进入待机状态。

3.4.4 待机 MDS 的唤醒

当班长 MDS 发现 MDS 集群负载增加或者根据历史信息，预期 MDS 会有较大增加，现有的 MDS 集群应对较为困难时，班长 MDS 将启动待机 MDS 的唤醒流程。待机 MDS 启动后，将会从负载最重的 MDS 上迁移一部分子树以平衡系统负载。

唤醒 MDS 的选择，班长节点根据历史记录中对 MDS 能力的判断，选择待机 MDS 中能力最强的节点 WakeupNode 进行唤醒。

WakeupNode 收到唤醒信号后，启动 MDS 的初始化流程，报告班长节点启动成功。

班长节点从现有 MDS 中选择负载最重的 MDS 作为负载的 SendNode，根据 SendNode 当前的负载 $L_{SendNode}$ 与热度 $P_{SendNode}$，MDS 集群的整体负载情况 O 以及历史统计的 SendNode 的节点能力 $C_{WakeupNode}$，决定迁移的节点热度 ΔP。

$$\Delta P_{SendNode} = P_{SendNode} * \frac{L_{SendNode} - O}{L_{SendNode}} \tag{3.12}$$

$$\Delta P_{WakeupNode} = C_{WakeupNode} * O \tag{3.13}$$

$$\Delta P = \min(\Delta P_{SendNode}, \Delta P_{WakeupNode}) \tag{3.14}$$

公式 3.12 计算的是 SendNode 可以迁出的最大的热度 $\Delta P_{SendNode}$，公式 3.13 计算的是 WakeupNode 可以接受的最大热度 $\Delta P_{WakeupNode}$，显然最后的迁移热度 ΔP 为二者中较小的一个。

SendNode 根据迁移热度值 ΔP，选择相应的目录子树进行迁移。

WakeUpNode 在缓存中重建子树，开始对外服务。

3.5 负载均衡策略分析

由前述，面向节能的负载均衡策略以元数据的集中存储分布式处理为基础，这种元数据管理架构使得在不影响系统可用性的情况下关闭元数据服务器成为可能。动态负载迁移是面向节能的负载均衡策略的实现手段，动态子树划分是一种很好的表示和迁移元数据的方法，但并不是唯一。例如 2.2.2 中提高的蓝鲸文件系统，采用动态绑定的方法支持负载迁移，面向节能的负载均衡策略对其同样适用。

对于待机 MDS 的选择，直接感觉应该选择负载最低的 MDS 进行待机，感觉这样迁移的数据量将会最少，然而考虑到整体负载水平增加最少这个指标，发现应该选择节点能力最弱的 MDS 进入待机。考虑移除负载最低的 MDS，因

为该 MDS 的负载水平低于整体负载水平，其移除后集群的整体负载水平将会增加；当将负载最低的 MDS 的负载分散到别的 MDS 上，又会造成整体负载的增加，这种增加在负载最低的 MDS 同时为能力最强的 MDS 时，表现最为明显。而选择移除节点能力最弱的 MDS 进入待机状态，若其负载水平大于整体负载，则系统的整体负载是下降的；若其负载水平小于整体负载，虽然也会造成整体负载的提升，但由于其能力较弱，将其负载转移到别的 MDS 上时，对整体负载的增量是较小的。从长远的观点看，我们希望进入节电模式后，留下来工作的机器都是能力强的机器，这样才能使得尽量多的 MDS 进入待机状态，尽可能多的节省能源。选择负载较低的 MDS 进行待机，会造成能力最强的 MDS 在早起就被移出系统，最终影响节能的效率；而选择节点能力最弱的 MDS 进入待机，最后是符合这一情况的。

迁入 MDS 的选择不能仅仅考虑负载平衡度，因为为达到负载平衡常常会将负载迁入到能力弱，负载低的 MDS 上，造成抖动。根据对 MDS 集群规模的预期，在选择迁入 MDS 时，先将一定数量的能力弱的 MDS 排除在外，是可行和有效的。虽然这种选择方式会在一定程度上造成系统负载的不均衡，但我们认为在整体负载较轻的情况下，适度的不均衡是可以接受的。

待机 MDS 的唤醒非常重要，唤醒 MDS 的时机是建立在对负载情况的实时监控和对预期负载的估计上的，唤醒后的 MDS 应尽快对外服务并且平衡负载，所以在 MDS 启动后，很快就发起了一次迁移流程。

对历史数据的统计信息非常重要，进入节能模式的时机和唤醒待机 MDS 的时机都用到了对预期负载的估计，这种估计是通过对历史数据的分析得来的。负载均衡过程中的许多阈值亦可通过对历史信息分析得来。

面向节能的负载均衡策略的节能效果随着元数据服务器集群的负载分布和具体应用变化而变化。一般来说元数据服务器集群峰值和谷值的负载差异越大，节能效果越明显；元数据服务器集群处于谷值负载的时间越长，节能效果越明显。

在负载均衡过程中，做了许多假定，做出的选择可能也不是最优的，一方面许多地方没有办法准确的得出结果，例如 MDS 预期的访问请求，只能使用当前值来替代；而另一方面则是因为信息的缺乏，为了降低信息通讯量，负载均衡过程中，实际上用到的实时信息只有负载和访问热度，因此很多需要更多信息的优化的策略没有采用。

总结节能策略的，主要有以下几点：

（1） 有效性：MDS 集中负载的方法，确实能有效降低系统能耗。

（2） 性能好：节能模式是在系统计算能力过剩的时候发起的，因此对于

系统性能的影响极小。

（3） 简单性：节能策略可以通过已有的负载迁移策略稍微扩展即可实现，不需太多复杂工作。

（4） 稳定性：通过对预期 MDS 集群规模的估计，合理选择迁入 MDS，克服了可能发生的抖动现象。

3.6 实验与分析

在实验中，元数据服务系统由六台异构的元数据服务器构成，分别是 MDS0，MDS1，MDS2，MDS3，MDS4， MDS5。其中 MDS0~2 为普通的 PC 机，待机功率 25W，空载功率 150W，满载功率 200W；MDS3~4 为联想服务器，待机功率 32W，空载功率 200W，满载功率 260W MDS3~4 为联想服务器。然后分别进行两个实验，首先在未启动面向节能的负载均衡策略的前提下，提供元数据服务，并记录 MDS 集群的整体负载水平和系统的能耗，接下来启动面向节能的负载均衡策略，提供元数据服务，并记录 MDS 集群的整体负载水平和系统的能耗。通过对比，来验证上述研究的负载均衡策略的可行性与有效性。

3.6.1 实验流程

实验一：不启动面向节能的负载均衡策略。

（1） 初始化：元数据服务器集群由六台异构的元数据服务器构成，初始没有任何数据。

（2） 执行目录树创建操作，在元数据服务器集群上创建 5 层目录树，每层目录下有 10 个文件，十个文件夹。

（3） 向元数据服务器集群连续发送 24 小时元数据请求，请求根据校内某视频共享网站的访问请求生成（00：00~23:59）。

（4） 实时记录各节点的工作负载和 MDS 系统的能耗

实验二：不启动面向节能的负载均衡策略。

（1） 初始化：元数据服务器集群由六台异构的元数据服务器构成，初始没有任何数据。

（2） 执行目录树创建操作，在元数据服务器集群上创建 5 层目录树，每层目录下有 10 个文件，十个文件夹。

（3） 向元数据服务器集群连续发送 24 小时元数据请求，请求根据校内某视频共享网站的访问请求生成（00：00~23:59）。

（4） 实时记录各节点的工作负载和 MDS 系统的能耗

3.6.2 实验结果和分析

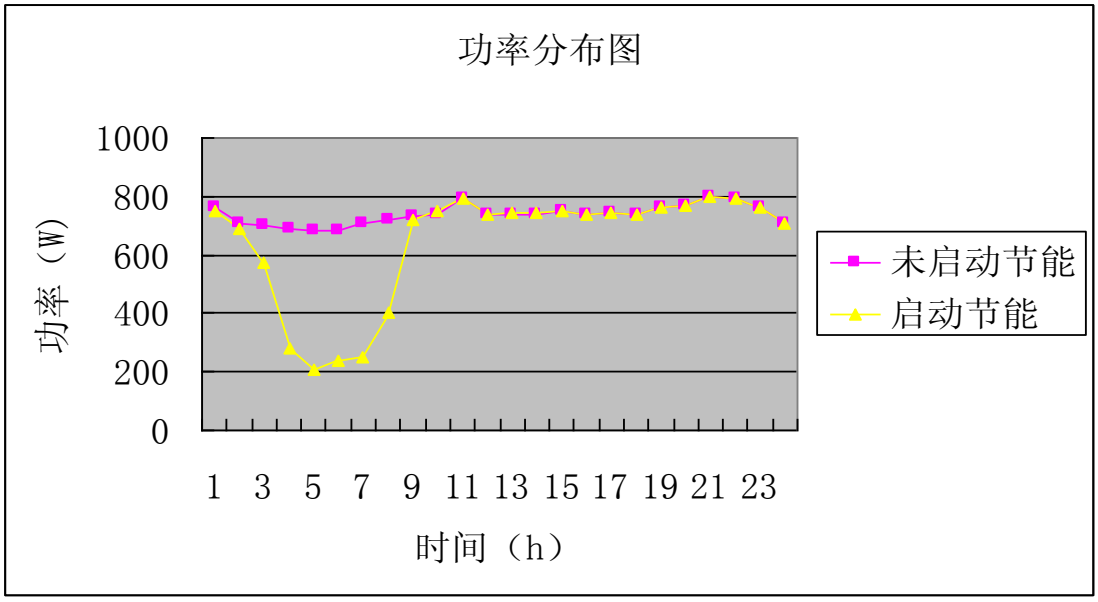

图 3.3 能耗对比

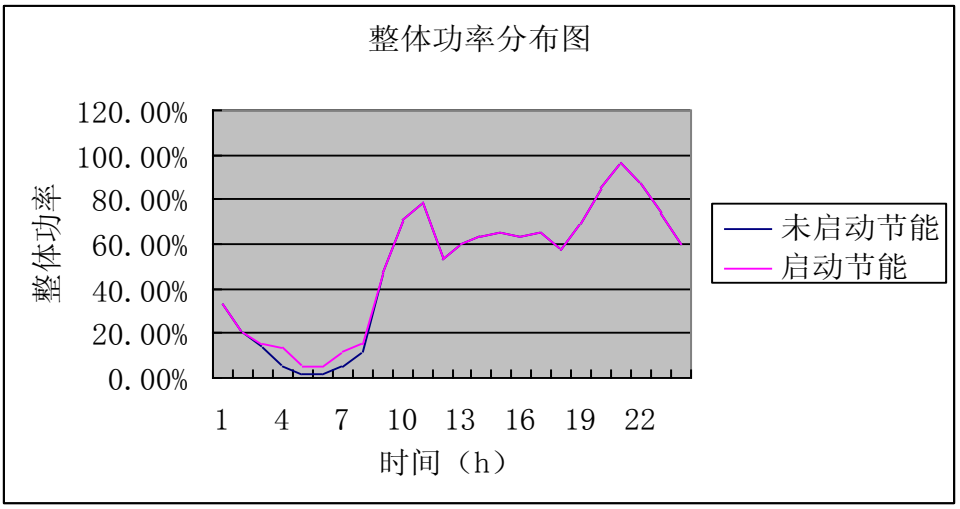

图 3.4 整体负载对比

图 3.3 为两种情况下，功耗的对比图，有图可知启用节能的策略下，在凌晨两点到 5 点间负载较低的时候，逐步关闭服务器，进入待机状态，大大节省了能耗，从凌晨七点开始，逐步唤醒服务器，开始正常工作。而未启动节能策略的，即使负载水平较低，消耗的功率却基本变化不大。

图 3.4 为两种情况下，整体负载的对比图，可以看出来在 2 点到 7 点虽然由于启动了节能模式，整体负载水平比未启动节能模式的高，但整体负载水平仍处于一个较低的水平，对处理客户请求并无影响。

3.7 本章小结

本章首先提出了分布式文件系统服务器面临的能源问题，然后分析了现有的分布式文件系统中存储服务器和分布式存储分布式处理的元数据管理方式难以解决节能问题的原因，在集中式存储分布式处理的架构下提出了面向节能的负载均衡策略：通过合并负载，关闭部分元数据服务器的方式达到节能的目的。通过给出节能策略的设计目标和基本思路，探讨了节能策略中关闭的元数据服务器和接受的元数据服务器的选择策略，以及负载合并过程的基本流程。最后通过实验和分析证明了策略的有效性和可行性。

第 4 章 元数据管理的一致性和可靠性保障策略

本章主要研究元数据管理的一致性保障策略和可靠性保障，为了保证元数据各副本的一致性，本章采用了基于主节点的元数据缓存架构，研究并实现了基于锁机制的副本更新策略以及基于两阶段提交协议的分布式操作流程；对于可靠性保障，本章研究并实现了区域自治的节点管理方法和基于日志的可靠性保障策略，保证了 MDS 的可靠性，单点失效时元数据服务不中断。

4.1 元数据管理的一致性保障策略

4.1.1 基于主节点的元数据缓存架构

在元数据管理系统中，为了应对大规模的读操作，元数据节点必须在多台元数据服务器上缓存，存在多个副本。而文件系统的特性决定了，元数据的操作必须序列化以保证元数据操作的原子性和一致性，即不能在同时有超过一个的地方对元数据的更新。

为了保证多个副本的一致性，本文采用基于主节点的元数据缓存架构。每一个元数据节点，都有且仅有一台元数据服务器可以负责对其更新，称该元数据服务器为该元数据的主 MDS，其余的副本只有读该元数据的权限。

为了保证元数据的更新能够及时被副本得知，主 MDS 在其缓存中保存元数据副本的分布信息；而为了保证副本元数据在需要新版本的内容时能够找到主元数据的位置同时为了保证 MDS 在清除缓存中的副本时候，主 MDS 能够及时更改其缓存中元数据副本的分布信息，副本元数据中同样需要保存主元数据所在的 MDS 信息。

主 MDS 的继承性：一般情况下，目录下的所有文件和文件夹与该目录有相同的主 MDS，除非特别声明。目录下所有文件和文件夹的 inode 和该目录有相同的主 MDS，目录的 inode 和目录内容可以有不同的主 MDS，目录的 inode 的主 MDS 负责对目录属性（所有者，权限等）进行更新，目录内容的主 MDS 负责对目录下的目录项进行更新（创建和删除文件）。

为了保证 MDS 在收到一个缓存中没有且不属于自己管理的元数据操作时，能够正确地找到负责的元数据，对于 MDS 缓存的每个目录子树，MDS 同样缓存该目录子树到文件系统根目录的路径上的元数据信息。这样当 MDS 收到一个元数据请求时，可以从缓存中根目录的元数据查找，若在缓存中找到，则表

明当前 MDS 缓存了该元数据，可以处理相关请求；若找不到，则 MDS 缓存的元数据与该路径至少有一个共同点（至少为根节点），设元数据 k 为缓存中元数据与目的路径最后（目录层次中的下层，根节点为第一个共同点）一个共同点，则可以将该请求转发给 k 的主 MDS，因为其主 MDS 至少比现 MDS 可以多得到下一层的所有元数据信息，依次继续必然可以查找到该路径最终负责的 MDS。

4.1.2 元数据更新的一致性保障策略

当主元数据更新时，主 MDS 会根据其缓存的副本的位置信息，向副本所在的 MDS 发送缓存失效信息，同时删除其缓存的副本位置信息（因为此时各 MDS 缓存的副本已经不是最新版本了）。其他 MDS 在收到缓存失效信息后，将相关的元数据副本状态置为失效，若后续有针对该元数据的操作，再向主 MDS 发消息，请求该元数据的最新版本。若 MDS 收到一个针对别的 MDS 负责的元数据更新操作，将会把操作转发给负责的 MDS。

为了保证元数据更新时的一致性，主元数据更新时，需要进行加锁操作，锁的状态分为以下三种：

（1） LOCK_SYNC 状态时，主副本和从副本节点都可以进行读操作。

（2） LOCK_LOCK 状态时，主副本节点有权利写操作，从副本无任何权利。

（3） LOCK_ GLOCKR 状态是 LOCK_SYNC 向 LOCK_LOCK 过渡的状态。

三种状态的转移图如图所示：

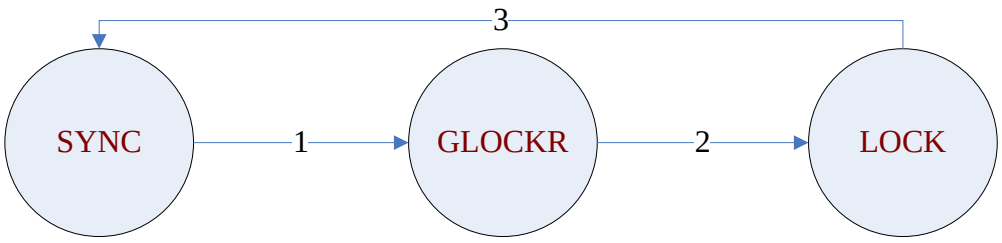

图 4.1 元数据锁状态转移图

过程 1：由于申请排他锁，需要通过两阶段协议获得，可能有些副本上有读锁操作，必须先等待，就将其负责锁的状态先设置为 LOCK_ GLOCKR，此时该状态锁阻塞主从副本上的读锁申请；

过程 2：直到所有从副本的已申请到读锁的操作都结束之后，将各自从副本上的锁都设置成 LOCK_LOCK，并回复主副本，当所有副本都回复其 ACK 之后可以将锁的状态改为 LOCK_LOCK。

过程 3：当写锁操作结束，并没有其他等待写锁操作时，将锁的 LOCK_LOCK 状态转换成 LOCK_SYNC，并通知所有从副本也将锁的 LOCK_LOCK 状态转换成 LOCK_SYNC 以及该锁负责的新修改的内容。

4.1.3 分布式操作的一致性保障策略

大部分操作可以在一台 MDS 上完成的，按有些操作可能涉及多台 MDS，例如 link、unlink 和 rename，这些操作都可能涉及到两个不在同一台 MDS 上的目录子树。若一台元数据服务器上的修改已经提交到硬盘，而另一个元数据服务器上的缓存因为故障而遗失，则会造成系统的不一致。为此我们设计了分布式日志的记录协议：

1. 分布式元数据操作的发起者更新缓存，记录日志，操作开始，然后向协作的元数据服务器（协作者）发送操作请求；

2. 若协作者无法执行操作，向发起者返回操作失败；否则完成操作，记录日志，向发起者返回操作成功；

3. 若发起者收到操作失败的消息，取消 1 所作的更改，操作结束；否则，若收到操作成功的消息，则记录日志，向协作者发送操作结束消息；

4. 协作者收到消息，记录日志，操作结束。

当元数据服务器进行故障恢复时，对于分布式元数据操作，会查询相关联的另一个元数据服务器，若该元数据服务器上的操作完成，则重做该操作，否则，即取消操作。

4.2 元数据管理的可靠性方法

本节主要及介绍元数据管理的可靠性方法，主要包括元数据服务器的节点管理和故障检测技术、基于日志的故障恢复技术。提出的元数据管理的可靠性方法可以有效检测出节点的故障，基于日志可以保证单点失效时元数据服务器可以很快恢复，恢复过程中元数据服务不中断。

4.2.1 基于区域自治的节点管理策略

元数据服务器集群采用区域自治的节点管理方式，将整个元数据服务器集群看成由数个区域构成。区域内所有节点通过自组织的方式建立联系，每个节点都有一个邻居节点列表，记录区域内节点的信息，区域内 MDS 节点间通过交换邻居节点列表来获知整个区域的情况。MDS 会定期向邻居节点报告自己的状态，亦会监视邻居节点的状态。

MDS 启动时，会指定一台 MDS 作为其邻居节点，新启动的 MDS 会复制

邻居节点的节点列表作为自己的邻居节点，然后向邻居节点列表里的 MDS 发送消息，报告自己的状态。其余 MDS 收到消息后，会将该 MDS 加入自己的邻居节点列表。

MDS 退出时，亦会向邻居节点里的 MDS 发送退出消息，在确认收到退出消息的 MDS 从邻居节点列中删除自己后，MDS 即正常退出。

MDS 异常退出时，将没有实时向邻居节点更新状态，在这种情况下，邻居节点会主动发出探测信息，若没有收到回复，则报告 MDS 故障。

另外，每个区域都会有一个班长节点，区域间的互联是通过区域的班长节点之间建立联系来完成的，当有请求的元数据在本地无法找到时，会向其他区域的班长节点发出查询请求，协助请求元数据。班长节点除了与区域外互联外，还负责负载均衡的决策和协调。

采用备份的方法保证班长节点的可靠性保证可靠性，区域内除了班长节点外，还有一个副班长节点，副班长节点是班长节点的备份，平时并不履行职责，仅在班长节点发生故障时，副班长节点才开始处理请求，同时开始新一轮班长和副班长节点的选举流程，选举出新的班长和副班长节点后，由他们接替自己的职责。一般选择区域内剩余能力最多的为班长节点，其次为副班长节点。

4.2.2 元数据服务器的故障恢复机制

当检测到故障发生时，会启动报警机制，人工决定是重启原 MDS 还是使用新的服务器替代原 MDS。由于 MDS 可以仅仅是一个进程，所以当超过一段时间后，还没有人工做出选择时，可以在已有的服务器上新启动一个进程来接替原 MDS 的工作。一般情况下，为了减少故障恢复的时间、保证系统性能，可以在 MDS 集群中配置一些备份服务器，这些服务器平时并不接收请求，只有当有 MDS 出现故障时，才会接替该 MDS 的工作，进行故障恢复和请求处理。若原 MDS 加入节点，可以作为新的备份服务器使用，也可以重新负责原有的请求处理。

在基于共享存储系统的元数据管理架构中，认为元数据保存在可靠的存储系统中，元数据存储的可靠性由存储系统来维护。由于 MDS 并不存储数据，所以我们并不需要对 MDS 之前管理的数据进行恢复。但是为了减少元数据的操作延时，MDS 采用日志技术来记录元数据的操作，许多对元数据的更新可能还未提交到磁盘。因此我们需要读取故障 MDS 的日志记录，将其中对元数据的更改更新到磁盘。

MDS 采用的是写前日志来记录元数据的操作，写前日志要求数据在写入磁盘前必须先写入日志，日志提交完整后，即可以向客户端返回操作成功，更新可以不必立即应用到实际系统中，而在需要时，再一次写入磁盘。使用写前日

志可以极大的减少数据操作的通信量与等待时间，而且多次更新一次提交，对于同一元数据的多次更新，只需要向磁盘提交最后一次的更新结果，因此日志技术可以极大提高系统性能。

MDS 的日志保存在共享的存储系统中，因此 MDS 故障时，并不受到影响。日志采用分页的结构，当一页写满后，即将该页的更新提交磁盘，然后开始新的日志页，已经提交的日志可以删除。虽然日志的记录顺序是按照 MDS 操作顺序记录的，但提交顺序并不是按此顺序，因为每条日志都已经记录了该操作完成后相关元数据的状态，所以只需要提交相关元数据的最后一次更新即可，之前的更新可以直接忽略。日志中有些操作是成对出现的，需要两者匹配才算操作完成（例如分布式的元数据操作）。

通过写前日志可以很好地保证对相关元数据的更新最终被提交到磁盘，但为了维护整个命名空间的一致性，元数据服务器中的缓存的一些信息同样需要恢复。然而为了保证日志文件的大小在一个合适的范围内，许多数据量大的状态数据（例如打开文件的状态、锁的状态，元数据副本的位置信息等）并未记录到日志中，许多涉及多台数据服务器的操作（例如 rename、link，unlink 等分布式元数据操作和子树迁移操作）也需要同别的元数据服务器进行协商才能恢复，故我们设计了如下的故障恢复协议：

（1） 恢复 MDS 读取日志文件中修改后未提交的元数据内容以及其他的相关信息，将其加入缓存。注意日志中可能有些操作没有成对出现，这种情况需要向别的 MDS 查询操作是否完成。

（2） 恢复 MDS 向其他活着的 MDS 发送查询信息，查询分布式操作的状态信息。若相关 MDS 返回操作成功，则恢复 MDS 提交该分布式操作；若相关 MDS 并未返回操作成功，则根据恢复 MDS 的日志记录来决定操作是否成功，若恢复 MDS 日志中记录操作成功，则告知相关的 MDS 操作成功；否则操作失败。在这一阶段，恢复 MDS 将明晰自己负责的主目录子树有哪些。对于其缓存中，和主目录子树无关的副本数据，则直接清除。

（3） 恢复 MDS 重连客户端，向其询问打开文件的状态信息和相应的锁的状态，恢复 MDS 根据收到的信息与客户端重新建立会话。

（4） 恢复 MDS 向其余 MDS 发送消息，声明自己对主元数据的所有权，其余 MDS 收到消息后，将相关副本的状态发送给恢复 MDS。恢复 MDS 根据收到的回复，决定主元数据上锁的状态。全部决定完成后，恢复 MDS 开始处理请求，故障恢复结束。

通过使用以上的恢复协议，可以保证 MDS 的缓存内容得到恢复。在 MDS

故障的过程中，客户端若连接不上该 MDS，会主动重试连接别的 MDS。MDS 的缓存结构设计决定了单台 MDS 故障，对其余 MDS 并无影响。而备份服务器的快速接替，也使得对故障 MDS 负责的元数据的操作可以顺利进行。

4.3 本章小结

本章首先研究了元数据服务器的缓存架构，设计并实现了多副本元数据的更新策略和基于分布式锁和两阶段提交协议的一致性保障策略。为了提高系统的性能，较少网络通讯，采用了写前日志来处理元数据操作。本章还研究并实现了基于备份服务器和日志的可靠性保证策略，保证了 MDS 的可靠性，元数据服务不中断。

第 5 章 LandFile 元数据管理系统的实现

本章主要介绍 LandFile 分布式文件系统的元数据管理系统的设计与实现方案。

5.1 LandFile 元数据管理系统的整体架构

5.1.1 LandFile 分布式文件系统的整体架构

LandFile 分布式文件系统采用元数据和文件系统份分开处理的整体架构：主要包括客户端，元数据管理系统和存储节点集群。其中客户端分为存储节点客户端和元数据客户端，分别连接存储节点集群和元数据管理系统。客户端对外不仅提供 GUI 接口，还提供 API 供系统应用调用。元数据管理系统负责存储和处理元数据，此处将元数据分开存储和处理主要为了更好的利用元数据操作的特性。存储节点集群负责向客户端提供文件数据的读取服务，采用的是基于对象的存取接口。

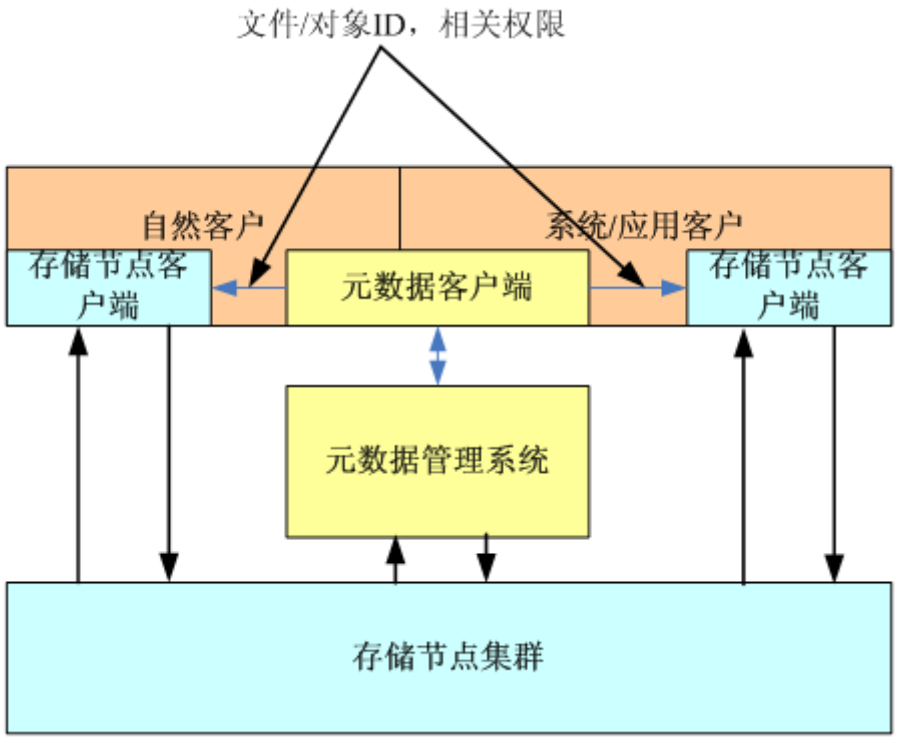

图 5.1 LandFile 分布式文件系统整体架构

5.1.2 LandFile 元数据管理系统的整体架构

元数据管理系统的结构如图 5.2 所示，元数据管理系统由元数据服务器集群和元数据存储节点集群构成，元数据服务器集群负责处理元数据请求，元数据存储节点集群负责存储元数据。元数据服务器间采用区域自治的节点管理模式，图中蓝色阴影部分即为班长节点。元数据存储节点集群复用了存储节点的系统，但是和文件数据的存储分开了，主要为了针对元数据的存取进行优化。

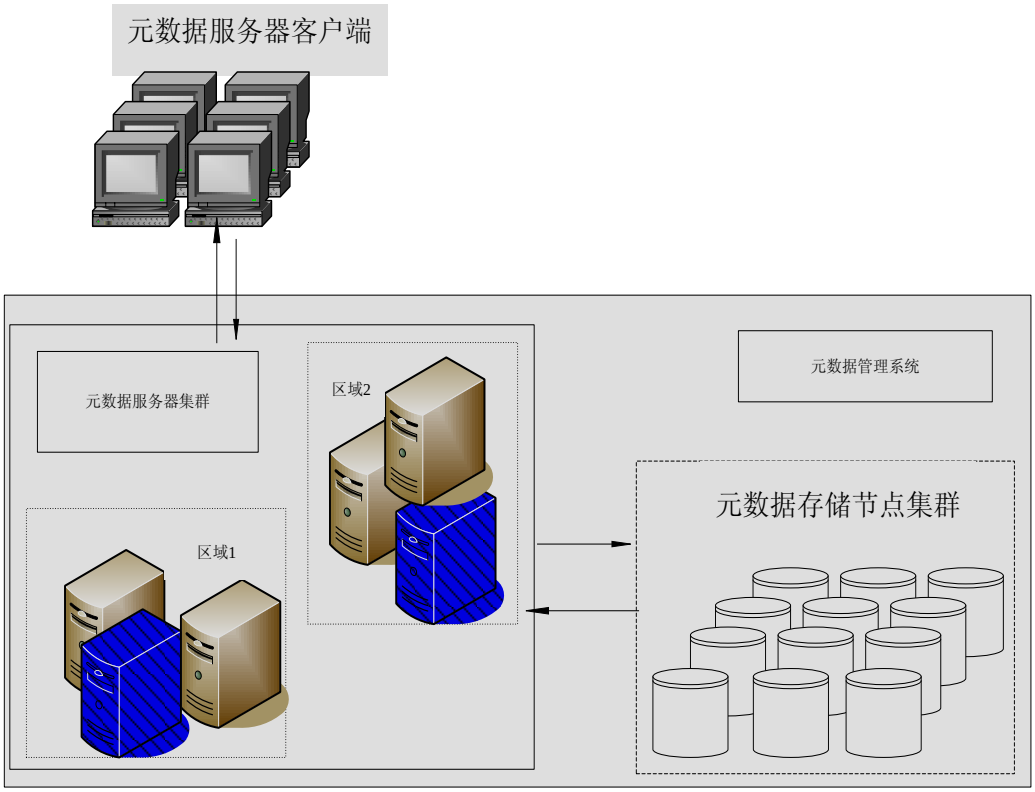

图 5.2 LandFile 元数据管理系统架构

5.1.3 LandFile 元数据管理系统的模块设计

元数据服务器的模块结构如图 5.3 所示，包括消息接口模块，节点管理模块，用户请求处理模块，可靠性模块，迁移/复制决策模块，子树迁移模块，子树复制模块，数据备份模块和事件记录模块 9 个部分，其中消息接口模块负责网络消息的传输；节点管理模块负责维护元数据服务器集群中各节点的活动情况，包括各节点的负载，心跳信息等；用户请求处理模块负责根据用户的操作请求，完成对目录子树的操作；可靠性模块负责在元数据发生更改的时候，在日志中记录对元数据进行的操作，出现故障时，根据日志进行恢复；迁移/复制决策模块负责根据元数据服务器集群的负载情况和目录对象的访问情况，来动态决定子树的划分以及元数据副本的位置；子树迁移模块和子树复制模块根据迁移/复制决策模块的决策来完成相应的操作；数据备份模块用于根据系统的要

求，定时备份元数据信息；事件记录模块负责记录元数据服务器的各种事件。

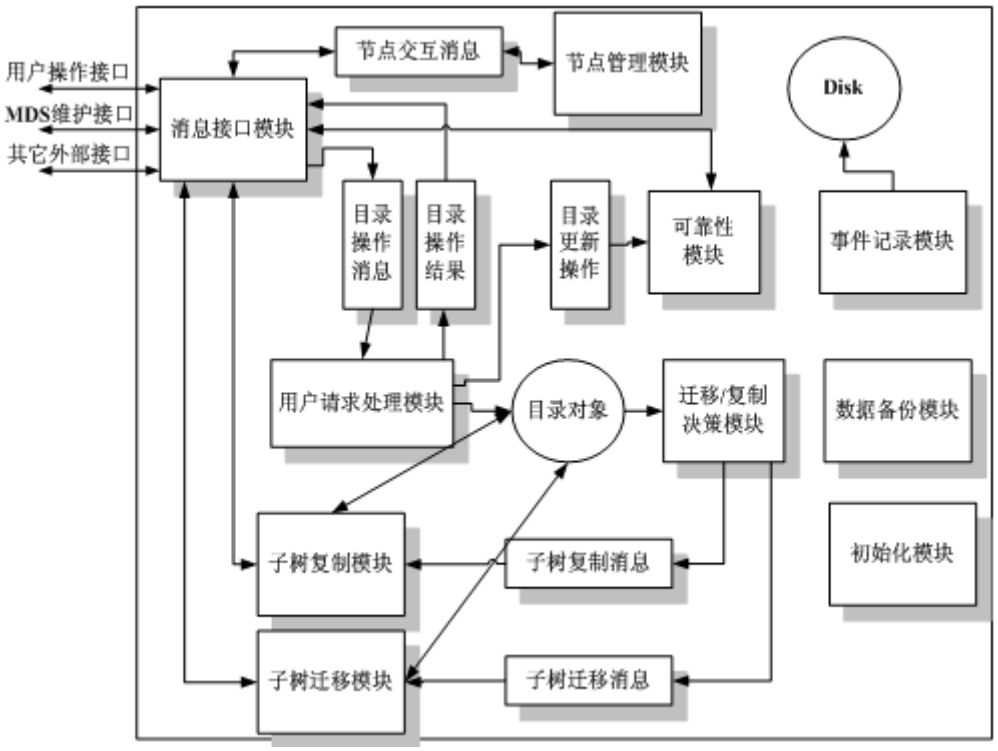

图 5.3 LandFile 元数据管理系统模块设计

5.2 迁移/复制决策模块的改进与实现

元数据管理系统的负载均衡子系统主要由三个模块组成，迁移/复制决策模块、子树迁移模块和子树复制模块。迁移/复制决策模块是负载均衡策略的决策者和发起者，子树迁移和子树复制模块负责实施具体的操作。

本文提出的面向节能的负载均衡策略，在已有 LandFile 负载均衡子系统[17]的基础上，只需对迁移/复制决策模块稍加改动即可实现。改动主要体现在一下三个方面：

（1） 对历史负载情况的统计和分析：对未来负载的准确预测，可以极大地提升面向节能的负载均衡策略的节能效果，更好的降低对系统性能的影响。虽然可以人工约定节能策略的发起时间，但考虑到系统应用的复杂多变性，采用基于历史统计信息的预期负载估计策略更加灵活可靠。对于历史负载情况的统计主要包含三个方面：MDS 集群的整体负载水平，MDS 的整体访问热度和统计负载时的时间，通过这三项指标可以较好的估计一段时间内的负载水平。

（2） 负载均衡决策函数的调整：在负载均衡决策函数中，增加了节能策

略适用条件的判断，添加了节能策略下迁移决策算法的实现，包括迁移 MDS 的选择，待机 MDS 的唤醒等。

（3） 迁移通知信息的调整：节能策略下迁移完成后 MDS 会进入待机状态，因此需要对现有的迁移通知信息就行调整，增加了待机模式的通知信息。

通过以上三个方面的调整，已有的基于复制和迁移的负载均衡子系统即可具有在负载低谷是进入节能模式的功能。面向节能的负载均衡策略复用了已有的负载迁移的功能，对大部分元数据管理系统而言，只需稍加改造即可实现。

5.3 用户请求处理模块的设计与实现

用户请求处理模块负责根据用户的操作请求，完成对目录子树的操作。用模块负责在缓存中建立并根据客户端的请求操作目录对象，客户端会话的保持，主 MDS 的查找，副本一致性的保持等。

由于目录子树的动态划分，MDS 缓存的元数据可能有多种可能，图 5.4 是一种可能的元数据分布情况，图中两台元数据服务器维护不同的目录子树而又有部分重叠。元数据服务器 1 是目录/usr/test/的主元数据服务器（图 5.4 中 MDS1 的阴影部分），其负责以目录/usr/test/为根的子树的维护。元数据服务器 0 则是其余目录子树的主元数据服务器（图 5.4 中 MDS0 的阴影部分）。在元数据服务器 1 上仍保留从根到/usr/test/中间经过的节点的部分信息，用于加快到/usr/test/的路径的解析，中间节点上与/usr/test/无关的信息都没有保存。例如元数据服务器 1 上虽保留有根目录的信息，但根目录的目录项只有/usr 一项，/tmp 则不在元数据服务器 1 的缓存中。文件/usr/test/file0 是一个热点文件，故其在两台服务器上都保留有副本，对该文件的读操作可以在两台服务器上同时进行，有效解决了热点文件的问题。

客户端请求处理几乎涉及到了元数据管理的各个方面，客户端在选择联系的目标 MDS 时，使用了客户端的缓存信息；MDS 接收到请求后，如果为更新操作，需要查找主 MDS，涉及到主 MDS 的查找流程；处理请求转发到主 MDS 上后，若相应的元数据不在 MDS 上，则启动了从远程存储系统读元数据的流程；更新元数据之前，需要启动元数据的加锁流程；对于元数据的更改，通常是先生成一个改后的版本，然后根据改后的版本，生成日志记录，更新日志，更新日志后，再将更改后的元数据加入缓存目录；元数据更改完成后，向客户端返回结果时，又启动了客户端缓存更新的流程；元数据更新后，若其存在副本，还需启动向其副本发送缓存失效的流程。

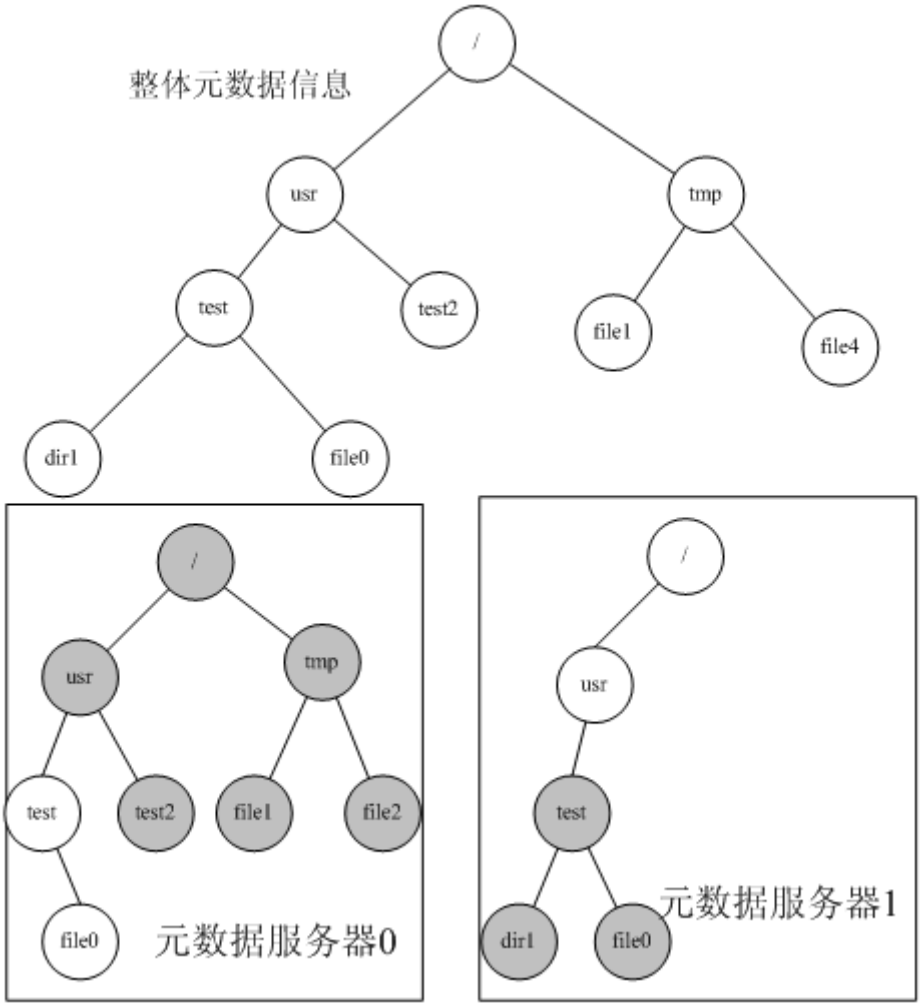

图 5.4 一种可能的元数据分布情况

下面以元数据客户端创建目录的操作为例结合图 5.4、图 5.5 和图 5.6，详细的介绍客户端请求处理的相关流程，图 5.4 为某一时刻整体缓存的元数据分布，图 5.5 为某客户端缓存的元数据分布情况，图 5.6 为客户端输入命令的执行流程。创建目录/usr/test/dir2 时的执行流程，包括以下步骤：

执行创建目录操作前缓存情况

目录名	分布位置
/	(0,1)
/usr	(0,1)

分布位置中第一项为其主元数据服务器的编号

执行创建目录操作后缓存情况

目录名	分布位置
/	(0,1)
/usr	(0,1)
/usr/test	(1,0)
/usr/test/dir2	(1)

图 5.5 客户端缓存分布

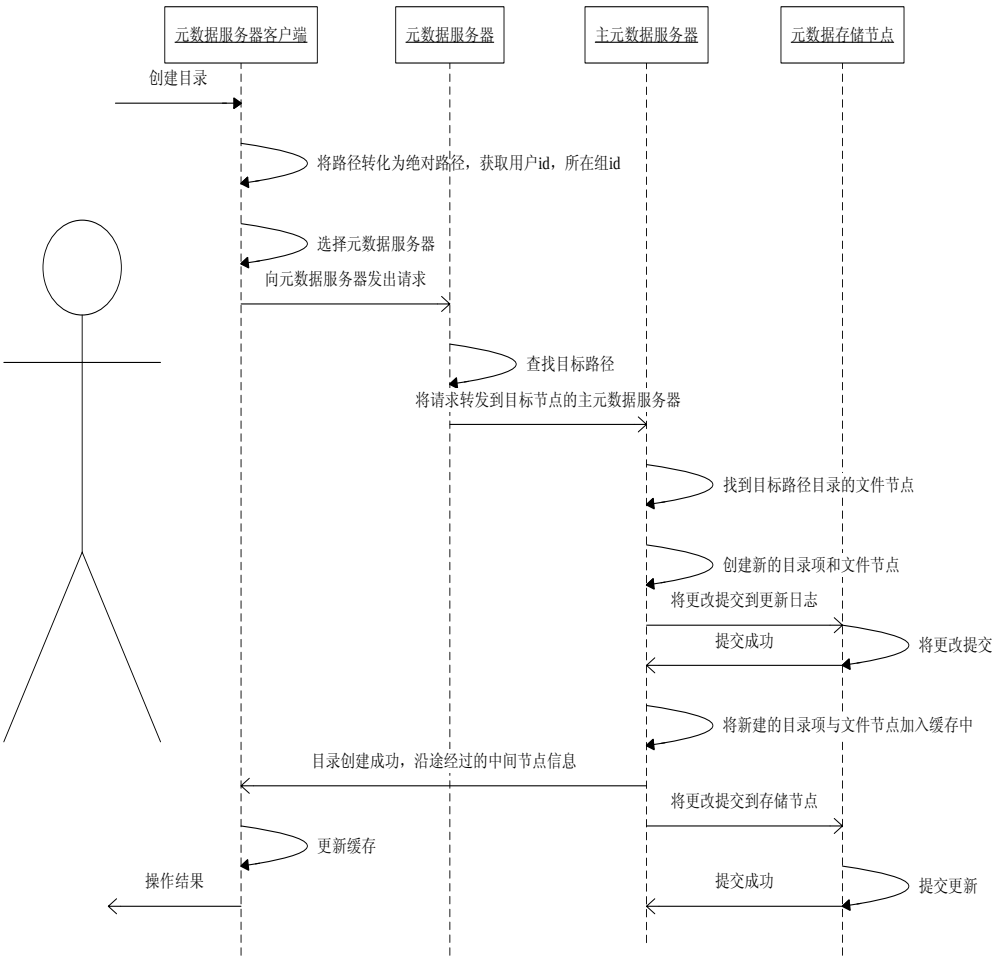

图 5.6 用户请求处理流程

（1） 元数据服务器 0 在缓存中找到/usr/test/对应的目录节点，发现本地缓存的只是其副本，/usr/test/对应的主元数据服务器为 1，故将请求转发到元数据服务器 1；

（2） 元数据服务器 1 在缓存中找到/usr/test/对应的目录节点，检查新增操作，在/usr/test/下生成新的目录项 dir2，相应的索引节点和目录节点，将更新提交到更新日志（更新日志会定期清理将脏数据提交到元数据存储节点）；

（3） 向更新日志提交成功后，元数据服务器 1 将新增的目录项，索引节点和目录节点加入到缓存的子树结构中，向客户端发送操作成功的消息，发送的消息中包含/usr/test/dir2/路径中经过的节点的信息；

（4） 客户端根据操作结果，更新缓存，返回给用户操作成功的标志。

5.4 可靠性模块的设计与实现

元数据的更新操作如图 5.7 所示，首先写入日志，然后再集中提交到存储节点。这种更新方式可以达到缩小延迟，吸收更新的目的。元数据的可靠性保证策略主要通过对日志的记录和恢复来实现。

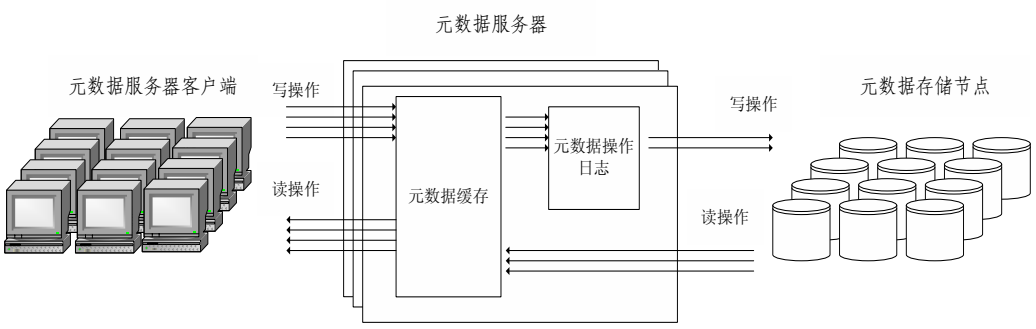

图 5.7 元数据更新示意图

5.4.1 日志结构的设计

日志的内容反映了已更改但尚未提交到磁盘的元数据内容，为了保证可以使用日志的内容对相关元数据进行恢复，所以日志中应包含恢复元数据所需要的足够信息。日志的更新时按页进行的，当日志文件写满一页时，就会另起一页保存新的日志项，同时将上一页日志内的更新内容刷新到磁盘，刷新完毕后，即可删除旧的日志文件页以节省空间以备新页被填满后保存日志项。

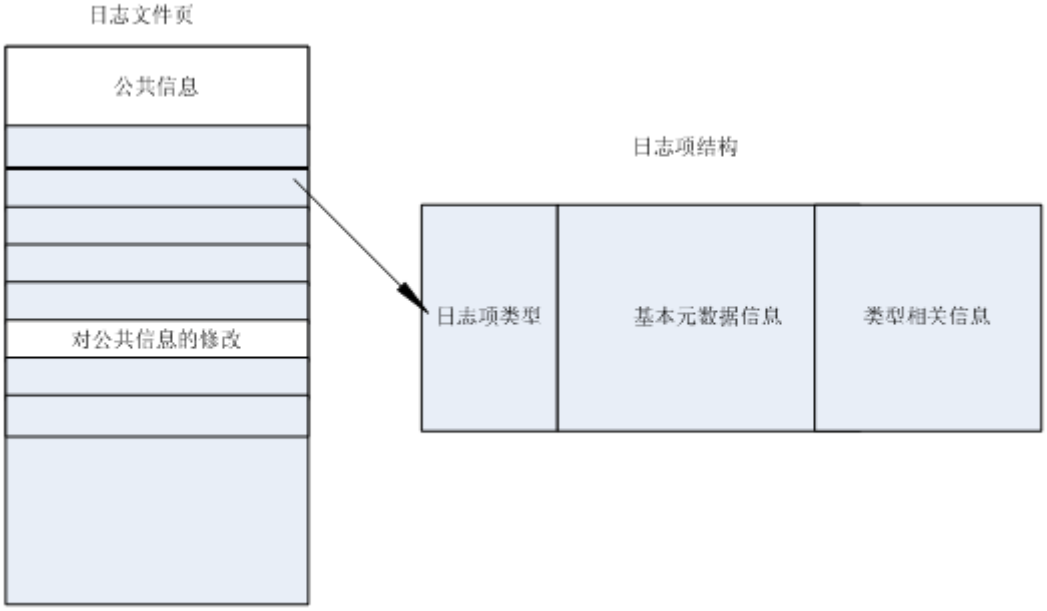

图 5.8 日志文件与日志项的结构

之所以按照页的方式来更新，是因为在页首的位置记录了一些经常会被使

用的量比较大的信息（例如从当前 MDS 的负责的目录子树到根节点的路径信息），重复记录将会浪费大量日志文件的空间，通过在页首的固定区域保存这些信息，后续的日志记录就可以在需要时查询页首的内容即可。记录的这些全局的信息也可能会被修改，但这些修改会及时的反映到日志中去，在提交的时候，将修改前的和修改后的元数据分成两个部分提交即可。

日志项有三部分组成，第一部分为日志项的类型，根据相应的日志项类型来对后边的结构进行解析。第二部分为操作相关的元数据信息，第三部分是日志项类型相关的扩展信息。日志文件页和日志项的结构如图 5.8 所示。

5.4.2 日志项的类型及定义

根据引发日志记录的操作的类型不同，可以将日志项分为四类：第一类是公共信息的更新，公共信息的更新是划分日志提交的界限，两端公共信息更新之间的数据可以一起提交，但若一段数据里包含了一次公共信息，则不能一起提交；第二类是基本的元数据更新操作的日志记录，这样的元数据操作设计到的数据简单，只需当前日志项的内容即可恢复；第三类是分布式的元数据操作的日志记录，这类元数据操作引发的日志项往往是成对出现，只有当期待的日志项出现时，才能确定操作时继续进行还是取消；第四类是迁移过程引发的日志记录，这类日志记录也是需要成对出现才能判断操作是否完成。

第一类日志项纪录的是节点管理的目录子树的结构，包含了从文件系统根节点到目录子树根节点的元数据信息，目录项类型为 EntrySubtreeMap。

```
struct ESubtreeMap{
   bufferlist data; //目录子树根节点到文件系统根节点的路径信息
   map<dirId, list<dirId> > subtrees; // 目录子树信息
}
```

引发第二类日志记录的操作包括 utime、chmod、mknod、openc，mkdir 以及部分操作的元数据在同一台 MDS 上的 link、unlink，rename 等操作。这些操作的特点是在一台 MDS 上即可完成，所以他们使用的日志记录的类型是相同的。

相关的数据结构为：

```
struct EentryUpdate{
   bufferlist data;   // 操作相关的数据
   string type; // 引发日志的元数据操作（mkdir,link 等）
   bufferlist client_map;// 仅在 rename 时使用，用以迁移在其上操作的 client

   reqid_t reqid; // 发出请求的客户端和请求的 id
```

```
    bool had_helpers; // 该操作是否需要其余 MDS 参与完成
}
```

对于第二类操作，had_helper 的值始终为 false。

引发第三类日志记录的操作为操作的对象分布在两台 MDS 上的 link、unlink 和 rename 等操作。这些操作需要两台 MDS 共同完成。每台 MDS 上都需要记录成对的日志项。在发起操作的 MDS 上，开始操作复用了上面的 EentryUpdate 结构，只是其中 had_helper 的值为 true；操作完成时同样需要记录日志项，相应的日志项类型为 EntryCommitted，结构见下。EntryCommitted 的结构非常简单，只保存了客户端和请求的 id 以匹配相应的操作。

```
struct EntryCommitted{
    reqid_t reqid; // 发出请求的客户端和请求的 id
}
```

对于协作的 MDS，其上也需要记录相应的日志项，日志项的类型为 EntryHelperUpdate，其中的 op 参数表明是处于准备阶段、提交阶段还是放弃操作。

```
struct EntryHelperUpdate{
    bufferlist commit;     // 提交需要的数据
    bufferlist rollback; // 回滚需要的数据
    string type; // 操作类型
    reqid_t reqid;// 发出请求的客户端和请求的 id
    __s32 mdsId;// 发出请求的 MDS 的 id
    __u8 op;   // 操作处于的阶段
    __u8 origop; // 原始的操作类型（link 或者 rename）
}
```

第四类日志项为目录迁移过程中记录的日志项，主要包括三类：在迁出 MDS 完成迁移时记录的日志项 EntryExport、迁入 MDS 在迁入开始时记录的日志项 EntryImportStart 和迁入 MDS 在迁入成功时记录的日志项 EntryImportFinish。主要结构如下：

```
struct EntryExport{
    bufferlist dir; // 迁出目录子树的信息
    dirId          base; // 迁出目录子树根节点 id
    set<dirId > bounds;// 目录子树的限定范围
}
```

在迁出 MDS 当 EntryExport 存在时即表明迁移完成。

```
struct EImportStart{
   dirId base;   //迁入目录子树根节点 id
   list<dirId> bounds; //迁入目录子树的限定范围
   bufferlist data;   //迁入目录子树的详细信息
   bufferlist client_map;   // 目录子树的客户信息
}
struct EImportFinish{
  protected:
   dirID base; // 迁入目录子树根的 id
   bool success;// 迁移是否成功
}
```

在迁入 MDS，EntryImportStart 与 EntryImportFinish 成对存在且 EntryImportFinish 中标记为成功才表明迁移完成。

5.4.3 根据日志项的恢复

在故障恢复节点，恢复线程读取日志进行恢复。读取日志项时，首先解析其类型，然后根据类型调用相应的函数解析日志项内容。当发现类型为 EntrySubtreeMap 的时候，开始在缓存中建立相关的子树结构，然后按照日志项的顺序，重做日志中的操作。由于日志页都是以 EntrySubtreeMap 类型的日志项开头的，所以可以保证顺利地在内存中建立起子树的结构。每类日志项都必须定义自己的恢复函数以备调用。

恢复函数的接口如下：

Int recover(MDSCache *cache)

功能：根据解析后的日志项内容，重做元数据操作

输入：服务器缓存的目录结构

输出：返回值，0 代表操作成功，-1 代表操作失败。

5.5 本章小结

本章介绍了分布式文件系统 LandFile 的元数据管理系统的设计与实现方案，首先介绍了元数据管理系统的整体架构和模块设计，重点介绍了用户请求处理模块和可靠性模块的设计与实现。

第 6 章 结束语

6.1 总结

分布式文件系统元数据管理的关键技术是本文的主要研究内容。本文主要针对其中的负载均衡策略，一致性保障策略和可靠性策略进行了研究。

本章首先从扩展性，负载均衡特性，实现复杂度等方面讨论了分布式文件系统和元数据管理系统的架构，认为元数据和数据解耦的分布式文件系统结构是以后的主流发展方向。元数据管理方面，分布式存储和集中式存储各有优缺点：分布式存储的元数据管理架构在负载均衡上的表现难以让人满意，集中式存储的管理架构则在性能和实现复杂程度上稍有不足。考虑到具体系统的需求，我们选用了集中存储分布处理的元数据管理架构。

现有元数据服务器集群都存在着过度设计的问题，其规模是以应付峰值时的访问请求而设计的，在访问低谷时，存在着严重的计算能力浪费问题，现有的文件系统对此少有涉及。本文分析了元数据服务器集群节能面临的问题，在已有的元数据服务器动态负载迁移策略的基础上，更改了负载均衡目标，将能力较低的元数据服务器的负载集中到能力高的元数据服务器上，然后关闭能力低的元数据服务器，达到了节约能耗的目的。由于是保留的都是能力强的服务器，而且此时处于访问低谷期，节能模式对系统性能影响不大。

元数据管理的一致性是元数据管理系统最重要的要求，直接关系着系统的可用性。本文设计了基于主节点的元数据缓存架构，实现了多副本元数据的更新策略，针对分布式操作，设计并实现了基于两阶段提交的协议来保证其一致性。

研究了元数据管理的可靠性保障策略：本文研究了元数据服务器的节点管理、故障检测以及故障恢复机制，实现了区域自治的节点管理方法和基于日志的可靠性保障策略，保证了元数据服务器单点失效时，能够被快速替换和恢复，失效时系统仍能无间断的提供服务。

利用上述研究的负载均衡策略，一致性保障策略与可靠性管理策略，设计并实现了分布式文件系统 LandFile 的元数据管理系统，包括元数据请求处理和可靠性模块，改进了负载均衡模块。元数据请求处理模块是整个系统的核心，几乎要与所欲的模块打交道，对于本文提到的相关问题都有涉及，元数据请求处理模块的实现需要与其他模块经常共同，协调共进。可靠性模块不仅要处理正常的更新，更要考虑故障时候的情况，日志系统的故障恢复过程是比较复杂

的。在已有的负载均衡模块的基础上，为其添加面向节能的负载均衡策略，需要做的改动并不很多。面向节能的负载均衡策略可以很容易地扩展到别的类似的系统中。

6.2 下一步工作

在大规模系统的设计中，方案的选择往往是各种需求的折中。严格的语义约束对系统性能的影响较大，下一步主要考虑在不影响系统可用性的前提下，适当降低部分语义约束，以达到较大幅度提升系统性能的目的。

参考文献

[1] Sun Microsystems, Inc. NFS: Network File System Protocol Specification, March 1989.

[2] B. Callaghan, B. Pawlowski, P. Staubach, Sun Microsystems, Inc. NFS Version 3 Protocol Specification, June 1995.

[3] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, Sun Microsystems, Inc. C. Beame, Hummingbird Ltd. M. Eisler, Zambeel, Inc. D. Noveck, Network Appliance, Inc. NFS Version 4 Protocol Specification, December 2000.

[4] R. Sandberg. The Sun Network File System: Design, Implementation and Experience. IN Proceedings of USENIX Summer Conference, summer 1987. University of California Press, pp. 300-313.

[5] 郭二目. Coda 分布式文件系统的缺陷及改进: [硕士学位论文]. 辽宁: 大连理工大学. 2005.

[6] 孙莉娜. 基于网络的分布式文件系统初探: [硕士学位论文]. 天津: 天津大学. 2006.

[7] Brama, P. J. The Coda Distributed File system, LINUX Jounral, June 1998.

[8] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, December 1995: 109-126.

[9] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. Proceedings of the Conference on File and Storage Technologies (FAST'02), January 2002: 231-244.

[10] Peter J. Braam, RumiZahir. Lustre Technical Project Summary, Version 2. July 29, 2001

[11] Sanjay Ghemwaat, HowardGobioff，and shun 一 TakLeung. The Google File System. Syosium on Operating Systems Prineiples(SOSP).

[12] 田颖. 分布式文件系统中的负载平衡技术研究: [硕士学位论文]. 北京: 中国科学院计算技术研究所. 2003.

[13] Sage A. Weil. Ceph: Reliable, Scalable, and High-Performance Distributed Storage. Ph.D. thesis, University of California, Santa Cruz, December, 2007

[14] 李晖. 基于日志的机群文件系统高可用关键技术研究: [硕士学位论文]. 北京: 中国科学院计算技术研究所. 2005.

[15] Drew Roselli, Jay Lorch, and Tom Anderson. A comparison of file system workloads In Proceedings of the 2000 USENIX Annual Technical Conference, pages 41–54, San Diego, CA, June 2000. USENIX Association.

[16] Dan Feng, Juan Wang, Fang Wang, Peng Xia. DOIDFH: an Effective Distributed Metadata Management Scheme. The 5th International Conference on Computational Science and Applications, 2007, pages 245-252.

[17] 肖培棕. 分布式文件系统元数据负载均衡技术研究与实现: [硕士学位论文]. 合肥: 中国科学技术大学. 2009.

致谢

三年的研究生生活即将结束，在这三年中，很多人给予了无私的帮助和指导，在此向他们表示感谢。

首先要感谢我的导师朱明教授。很荣幸，能在朱老师的指导下度过三年的研究生生活，衷心感谢朱老师三年来的教导和帮助。朱老师渊博的学识、敏锐的洞察力和严谨、勤奋的治学工作态度，为我树立了永远学习的榜样。朱老师对学生无微不至的关怀，幽默开朗的态度，灵活多变的工作方法，让我们在生活和工作各个方面都感受到春天般的温暖。在毕业论文的写作过程中，从最初选题，到关键点研究，再到最后的写作，朱老师给了我很多指导，提出很多有益的建议。

感谢实验室的刘守群、肖培棕、马林、刘大伟、易荣锋、吴彪、叶俊鹏、刘东君、薛伟、曹海滨、姚世佳、尹文科等同学，你们在我遇到问题和困难的时候及时给出建议，并协助我度过难关。

感谢 SA0710 陪我一起学习、一起生活、一起娱乐、一起找工作、一起赶论文的各位同学，是你们让我三年的研究生生活变得更加丰富多彩，充满乐趣。

感谢我的父母和家人，始终教导我诚实做人、勤勉努力的品质， 你们所给予的期望与鼓励使我一直向着做一个更好的人而努力。

感谢论文评阅老师的仔细审阅！

向所有关心和帮助过我的师长、同学和朋友致以最诚挚的谢意！

2010 年 5 月

在读期间发表的学术论文和取得的研究成果

已录用论文：

[1] 冯幼乐，朱六璋. CEPH 动态元数据管理方法分析与改进. 电子技术.

申请专利：

[1] 朱明，冯幼乐. 一种分布式文件系统动态元数据管理方法及系统 (No: 200910236456.7).

参与科研项目情况：

2008 年 7 月～2010 年 5 月 国家 863 计划课题

新一代业务运行管控协同支撑环境的开发（2008AA01A317）

项目核心成员

分布式文件系统元数据管理技术研究与实现关键技术研究的功能衰退

作者： 冯幼乐
学位授予单位： 中国科学技术大学

相似文献(10条)

1.学位论文 华清 网络环境下分布式文件系统的设计与实现 2006

今时今日，网格技术已经不再把自己局限在高性能计算的范畴中，而是通过向网络服务靠拢，建立起一套面向服务的体系架构。相应于此，网格中的数据模块也不再是一个提供资源的底层支持模块，而逐渐向一个功能独立的，相对自治的分布式文件系统发展。

本文的目的在于构造一种网格环境下的分布式文件系统。在结构上它相当于中国教育科研网格底层支撑平台的数据模块部分，为CGSP其他功能模块提供持久的数据存储功能以及稳定而高效的数据传输服务。另一方面，这个系统也可以独立运行，网格终端用户可以通过它建立自己的数据空间。

传统的分布式文件系统往往是紧耦合的、基于文件级别应用副本策略的。这些系统虽然性能出众、但往往是应对专门系统设计，通用性不强。而过去基于Web的分布式文件系统往往性能低下、存储能力弱、传输效率低。为了满足网格环境下海量数据密集型应用的需求以及面向服务的框架，我们设计并实现了一个基于分片的、松耦合的分布式文件系统，它包括底层存储资源集合、存储资源管理模块、元数据管理模块、副本管理模块、数据传输管理模块、信息监控模块、策略分析模块以及虚拟文件视图终端等部件。

本文设计并实现的系统具有以下的特点和优势：

2 稳定性：通过副本管理模块调整系统中文件合理的冗余度，对”零副本危机”进行预测并予以避免。策略分析模块会在存储资源选择时挑选健壮性最佳的数据节点进行存储；

2 高效性：在传输中使用GridFTP带状并行传输，同时提高服务器端和客户端的带宽利用率。策略分析模块会在构造传输计划时挑选最近、当前可用带宽最大的节点进行传输；

2 通用性：通过使用网络服务资源框架实现远程调用，存储资源只需向存储资源管理模块汇报就可以加入资源集，用户也可以在任何终端登入自己的用户空间；2 可扩展性：通过对策略分析模块的合理设计，开发评估函数接口。

他模块可以通过调用接口得到所需要的结果，而管理员可以通过对接口的不同实现，对配置参数的调整改变系统的运行策略。这样系统也便于重构和功能扩展。

本文所有设计、实现的结果都在一个网格环境下测试，并通过OptorSim模拟了大规模节点数、长运行时间的运行环境进行可用性测试。

2.学位论文 王建勇 可扩展的单一映象文件系统 1998

传统的分布式文件系统不能为机系统提供严格的单一映象功能,而且由于不能适应计算技术的发展趋势,无法满足应用对机群系统的I/O性能、可扩展性和可用性的需求.曙光超级服务器是典型的机群系统,他们为其研制开发了可扩展的单一映象文件系统COSMOS,并称其原型系统为S2FS.该文主要描述了S2FS的设计、实现及评价.首先,S2FS是一个全局文件系统,它通过实现位置透明性和严格的UNIX文件共享语义而保证了严格的单一系统映象.其次,为了提高S2FS系统的性能和可扩展性,该文对合作式缓存进行了研究和评价.最后,为了避免单一服务器瓶颈问题,我们为S2FS采用数据存储与元数据管理分开的策略,实现了分布式的数据存储和元数据管理功能.虽然该文在保证系统单一映象和二进制兼容性的基础上,对适合于机群文件系统的可扩展性技术进行了研究,但由于应用对I/O的需求是永无止境的,且其I/O存取特征以及计算技术的发趋势也有不断发生变化,这一切都为我们未来研制新型的分布式文件系统提出了更大的挑战.

3.期刊论文 冯幼乐.朱六璋 CEPH动态元数据管理方法分析与改进 -电子技术2010,47(9)

分布式文件系统(CEPH)的动态元数据管理方法极大地提高了元数据服务器的性能和扩展性.本文首先分析了CEPH元数据服务器集群中的负载均衡策略,针对其在异构元数据服务器和网络延迟较大时存在的问题提出了改进方案.实验证明,改进后的方法不仅提高了系统的性能,扩大了系统的使用范围.

4.学位论文 李健灿 集群存储系统架构及相关技术的研究与实现 2009

随着计算机技术的不断发展和应用，人们对数据存储的需求越来越大，对存储的容量和速度的要求越来越高。传统的存储系统因其物理组成而受到很大的限制，集群存储作为一项已被广泛使用的技术，能够提供按比例增加的服务器或存储资源的性能、容量、可靠性及可用性，突破了单机设备的种种限制。

在存储技术飞速发展的今天，集群存储技术还有着许多急需解决的问题，如随着存储的迅速增长，存储的可扩展性问题，还有存储的安全性，存储的备份和恢复等问题都还没有得到很好的解决方案。

本文研究了当前存储体系结构及相关技术，对网络连接存储、存储区域网络和集群存储文件系统的架构、特点和相应的解决方案作了较详细的分析，在此基础上，开发了protoDFS集群分布式文件系统，初步实现了文件存储、文件同步、文件访问等功能和大容量存储的特性。

本文重点讨论了集群存储的元数据分布查找方式和数据副本分发方法。元数据是最重要的系统数据，元数据的访问性能影响着并行文件系统的性能。如何合理的将系统中的元数据布局到不同的MDS上，从而使得对元数据的请求相对均匀地分散到各个MDS上是元数据分布的目标，本文尝试提出一种提高系统可扩展性和负载均衡性的解决方案。另外，数据副本的分发和访问也是系统容错性、可靠性的重要内容。本文研究了动态副本管理的技术和副本一致性策略并进行了实现。

5.会议论文 贾瑞勇.张延园 SAN文件系统元数据服务器集群体系结构及关键技术研究

SAN文件系统是一种基于存储区域网络的分布式文件系统,其设计目标是保持传统分布式文件系统的文件共享语义,同时可达到接近于本地文件系统的性能.元数据服务器集群是提高SAN文件系统性能、可扩展性和可用性的关键.本文给出了一种新颖的元数据服务器集群模型,该模型具有良好的可扩展性和容错能力,并能提供高度并发的元数据服务;深入探讨了实现该模型的两大关键技术:组通信和元数据管理,最后简单介绍了目前国际上的发展现状和研究趋势.

6.学位论文 郭威 分布式文件系统ZD-DFS的设计与实现 2006

随着互联网迅速发展，对互联网海量数据的存储和读取成为诸多网络应用系统的重要负载。当文件个数，读取需求急剧增加时容易导致后台文件系统服务器负载过大而成为整体性能的瓶颈。而传统的文件系统很难满足海量数据存储和读取的性能要求，而现有的通用分布式文件系统并不专为互联网应用背景的海量小文件存储提供良好的支持。所以需要开发对互联网环境下的海量小文件支持良好的分布式文件系统。本文设计实现的分布式文件系统ZD-DFS采用以元数据服务器管理后台存储服务器的形式向应用屏蔽资源定位、负载均衡、分布式事务、数据迁移、数据一致性等细节，并提供一个统一的编程接口。该系统通过元数据管理服务器和存储服务器节点提供满足Web应用需求的功能。元数据管理服务器负责管理分布式文件系统的全局信息，实施资源定位，数据迁移和负载均衡，管理维护后台存储节点，提供对分布式存储、分布式事务、文件资源读取等功能的全局支持。在元数据服务器的管理协调下，提供了对客户接口机对各种文件的存储，读取，更新，删除操作。ZD-DFS的分布式事务处理支持两阶段提交协议，实现了XA协议，对事务处理过程中的故障和常提供了容错机制。ZD-DFS同时通过锁机制，版本戳和心跳协议等方法保证了分布式的数据一致性。ZD-DFS分布式文件系统为海量数据提供了良好的存储、读取、更新性能，系统各部分性能均衡，不存在明显的性能瓶颈。具有较好的可扩展性，能够方便地进行存储和计算能力的扩展，为大型网络应用提供了较好的底层支持。

7.学位论文 邵强 对象存储文件系统中元数据管理集群关键技术研究与实现 2005

在信息时代，数据存储具有举足轻重的地位，存储已经开始成为关系企业生存发展的重要因素，如何构建一个高性能、高可伸缩、高可用、易管理、安全的存储系统成为目前所面临的一个重要课题。

基于对象的存储技术是存储领域的新兴技术，提出了一种新型的存储结构。对象是这种存储结构的核心，封装的元数据和文件数据分别由不同的系统管理。元数据包括文件的属性和访问权限，由元数据服务器管理；文件数据条块化存储于智能的对象存储设备；客户文件系统向用户提供存储系统的接口，可以与元数据管理系统交互和与对象存储设备直接进行数据交换。基于对象存储结构构建的大型分布式文件系统，可扩展性强、性能高，可提供较强的并发数据处理能力。

本课题主要研究对象存储结构中的元数据管理。元数据服务的扩展性和高性能对于对象存储结构至关重要，采用集群管理元数据是大型存储系统中元数据管理的一种趋势。本文采用一种新颖的结构实现层次管理元数据的元数据管理集群，分别以目录路径索引服务器集群和元数据服务器集群管理目录元数据和文件元数据，并研究其中的关键技术。

在研究集群负载均衡的基础上，设计和实现元数据管理集群静态负载分配与动态反馈重分配相结合的负载均衡方案。通过静态元数据分割算法，实现元数据服务负载分流或者元数据分布存储实现负载分流；服务器动态反馈服务器负载信息，实现不均衡负载重新分配。这样保证元数据管理集群的负载均衡，并解决“热点”数据访问问题。

另外，研究元数据管理集群中可用性问题，DPIS集群中采用共享容错磁盘阵列和节点容错机制解决共享存储数据和节点故障问题，MDS集群采用备份服务器保证服务器节点出现故障时元数据服务工作的接替和数据备份的重建，实现元数据管理集群在单点失效和特定的多点失效情况下的容错和恢复，保证系统的可靠性和可用性。

8.学位论文 娄成龙 对象文件系统中元数据管理技术研究 2005

随着信息技术的进一步发展，以及网络的大规模应用，带来了数据的爆炸性增长，也给网络存储带来了巨大的发展机会。如何构建一个扩展性强、可靠性高、易管理的高性能存储系统成为目前研究的一个重要课题。

基于对象的存储技术是存储领域的新兴技术，它提出了一种新型的存储结构，数据对象是这种存储结构的核心，数据对象封装了用户数据(文件数据)和这些数据的属性(元数据)，他们分别由不同的系统管理。以对象存储结构为基础构建的大型分布式文件系统，可扩展性强、可靠性高，能提供较强的并发数据处理能力。元数据服务管理在对象存储文件系统中尤为重要，采用集群管理元数据是大型对象存储系统中的一种趋势，本文致力于研究对象存储结构中的元数据集群管理技术，所做的主要工作如下：1.分析研究基于对象存储系统的体系结构，设计并实现了一个小型的对象存储文件系统原型OCFS。

2.研究对象存储文件系统中的元数据管理，设计原型改进的文件系统OCFSⅡ，对元数据管理集群实行层次化管理，分别以目录路径索引服务器DPIS集群和元数据服务器MDS集群管理目录元数据和文件元数据。

3.在研究集群负载均衡的基础上，设计和实现OCFSⅡ元数据管理集群静态负载分配与动态反馈重分配相结合的负载均衡方案。通过静态元数据分割算法和元数据分布存储，实现元数据服务负载分流；采用动态反馈服务器负载信息，实现不均衡负载重新分配。保证元数据管理集群的负载均衡，并解决了“热点”数据访问问题。

4.设计实现了OCFSⅡ元数据管理集群可用性保障方案。目录路径索引服务器DPIS集群中采用共享容错磁盘阵列和节点容错机制解决共享存储数据和节点故障问题；元数据服务器MDS集群采用备份服务器保证服务器节点出现故障时元数据服务工作的接替和数据备份的重建。实现了元数据管理集群在单点失效和特定的多点失效情况下的容错和恢复，保证了系统的可靠性和可用性。

9.学位论文 侯玮玮 基于内容存储关键技术研究与实现 2007

随着信息化程度的不断提高，数据对于企业的重要性凸现，存储技术在其中起到的作用日益增加，而网络技术的发展以及数据量的飞速增长，需要新的存储网络技术适应现有的网络和存储环境。面向对象存储网络这种新的存储网络体系结构，结合SAN和NAS的优点，逐渐成为学术界和工业界关注的热点。基于内容存储是面向对象存储的一个典型范例，除了具有面向对象存储的优点之外，还具有数据压缩和保持数据完整性等优点。

本文研究了基于内容存储的关键技术，包括对象和对象ID生成方法、元数据管理、存储空间管理、数据压缩技术和WORM技术。在此基础上完成了基于内容存储网络CASN的总体设计，并提出了新的安全模型和协议。同时，本文还设计并实现了一种基于内容的存储设备CASD。CASD使用根据对象内容得到的Hash值作为存储对象的标识，并以此来访问存储对象。这种方法具有一次写的功能，防止意外或者恶意的数据破坏。此外，CASD还能去除重复数据，可以节约存储空间和节省网络带宽。以上这些特性使得CASD具有很广泛的应用前景。本文介绍了基于内容存储的主要应用场景，并提出了一种基于内容存储技术在分布式文件系统中的应用。

本文在CASD的基础上，搭建了基于内容存储网络原型系统。为了进行对比测试，搭建了Intel OSDN、NFS和iSCSI三种网络存储系统，并分别比较了CASN系统和这几个存储系统的性能。实验结果表明，CASN系统和其他网络存储系统相比，性能具有一定的优势。

10.学位论文 刘安 PC机群上的分布式协作化文件系统DCFS的设计与实现 2000

由于当前的各种机群分布式文件系统中缺乏一种适合于PC机群环境的、面向高性能的实现,研究人员为DISCOS设计和实现了可扩展的单一映象文件系统-DCFS.该文描述了DCFS的设计、实现和性能分析,着重讨论了设计中的一些关键性问题.DCFS首先是一个单一映象的分布式文件系统,支持数据的透明存取和共享访文件共享语义.另外,研究从员在修改Linux操作系统核心的前提下, 通过实现一个新的基于SIO规范的I/O库,保证了与其底层平台的无缝连接,以及和Linux应用程序的完全二进制兼容.DCFS还是一个可扩展的分布式系统,在设计上采用无集中服务器Serverless结构, 将数据存储和元数据管理工作分布到整个系统中,并实现了分组的stripe存储,提高了系统的带宽和扩展性.DCFS的设计主要是面对并行应用,通过对可扩展I/O的研究,研究人员在DISCOS机群上实现了可扩展I/O底层编程界面(SIO-LLAPI)的一个子集.