华中科技大学

硕士学位论文

对象存储系统的元数据管理

姓名：张顺达

申请学位级别：硕士

专业：计算机系统结构

指导教师：王芳

20060501

摘　　要*

随着网络技术和信息数字化的快速发展，面向海量数据的大型应用纷纷涌现，进一步对存储系统性能提出更为苛刻的要求。尽管磁存储技术仍在不断发展中，但受到块级存储访问接口制约，无法改变 I/O 性能远落后于 CPU 和内存速度的状况。对象存储系统（Object-Based Storage System）以对象为接口，将有望解决这些问题。容纳海量用户数据的对象存储系统中高效的元数据管理成为了新的挑战和研究课题。

对象存储系统由客户端、元数据服务器和各个对象存储节点三部分组成。用户数据存放在直接联网访问的智能存储节点上。元数据服务器在对象存储系统中的位置非常重要，是整个系统潜在的瓶颈。在这种具有分布式体系结构特征的对象存储系统中，文件被映射到一个或多个对象存储节点上。合理的对象分布策略对系统性能显得尤为重要。针对常用对象分布策略哈希 (Hashing) 算法和分片 (Fragment-Mapping) 算法存在的优缺点，提出一种能够结合两者优点、又尽量避免其缺点的柔性对象分布算法，同时分析了影响对象存储系统性能的主要因素。

元数据服务器的设计及元数据的组织和存储是面向对象系统中元数据管理的重要组成部分。元数据服务器使用了轻量级目录访问协议（Lightweight Directory Access Protocol，LDAP）作为存放元数据的平台，针对这个平台设计了相应的数据分配算法和数据转换模块，针对元数据访问特征，构建缓冲机制优化元数据访问性能。通过测试验证了柔性对象分布算法和元数据组织管理模式在对象系统中是行之有效的，并对系统性能的提升起到了重要作用。

关键词：网络存储，对象存储系统，元数据管理，对象分布策略，
轻量级的目录访问协议

---

*本文的研究工作受国家重点基础研究发展计划(973 计划)资助项目(2004CB318201)和中国国家自然科学基金资助项目(60303032) 资助

Abstract*

The rapid development of network technology and digital information has stimulated the emergence of mass information applications. The current storage architecture becomes the performance bottleneck. The rapid development of magnetic store technology leads to the situation that the I/O performance falls behind the speed of CUP and memory. However, the traditional block access interface can not change this situation. The Object-Based Storage (OBS) providing object-based access interface is expected to change the situation. And its metadata management becomes new challenges and research topics.

The object-based storage system contains three major components, namely are clients, Metadata Server (MDS) and object-based storage nodes. Data is stored on the nodes that can be directly accessed through the network, while metadata is managed separately by one or more specialized metadata servers. The position of the MDS in the object-based storage system is very important, and it can be a potential bottleneck of the system. In the object-based storage system files are mapped onto one or more data objects stored on the nodes. The policy for object allocation is a critical aspect affecting the overall system performance. Hashing and fragment-strip are two common techniques used for managing objects, but both have their disadvantages and advantages. We present an efficient algorithm that combines the advantages of these two approaches while avoiding their shortcomings. The key factors which can impact the performance in the objects allocation are also be discussed.

The design of MDS in object-based storage system and the organizing and management of metadata are also very important. The MDS in our system uses Lightweight Directory Access Protocol (LDAP) to store the metadata. And we design data allocation and data conversion modules especially for it. We also build buffers to optimize the performance. We test the system and prove that our object allocation algorithm is effective and the buffers optimize the performance.

Keywords: Network Storage, Object-Based Storage, Metadata Management, Object Allocation, Lightweight Directory Access Protocol

---

* The research is supported by National Basic Research Program of China (973 Program) under Grant No. 2004CB318201 and the National Science Foundation of China under Grant No.60303032.

1 绪 论

1.1 课题背景

随着信息社会的发展，越来越多的信息被数字化，尤其是伴随着 Internet 的发展，数据量呈现出爆炸式增长。因而在未来几年内，存储技术将成为令人瞩目的一个市场。1999 年，世界范围的存储服务市场为 210 亿美元；到 2003 年，已超过 400 亿美元。而在今后的几年内，存储服务市场将进入飞速发展期。基于 Internet 的应用，比如电子商务、电子邮件和企业数据信息，将成为存储服务的主要市场，这些应用都要求快速的数据访问。从存储服务的发展趋势来看，一方面，是对数据存储量的需求越来越大，另一方面，是对数据的有效管理提出了更高的要求[1]。

目前存储系统除了传统以服务器为中心的直接存储（Direct Access Storage, DAS）外，例如附网存储（Network Attached Storage, NAS）[2]、存储区域网（Storage Area Network, SAN）[3-5] 等网络存储占据主导地位。NAS 采用“ 文件 ”数据组织，通过网络接口把存储设备直接连入到网络中，是一种特制的网络文件系统“ 瘦 ”服务器，支持 NFS 和 CIFS 的网络文件协议，实现细粒度数据共享以及跨平台文件共享，具有系统易用性和可管理性，但同时存在系统扩展性差的缺陷，这包括存储容量的扩展性和性能的扩展性[6,7]。

SAN 采用“ 块 ”数据组织，通过可伸缩的高速专用存储网络互连不同类型的存储设备与服务器，提供内部任意节点间多路可选择的数据交换，方便地共享存储设备，向外界提供服务，从而提高存储系统的可用性和性能[8,9]。SAN 很大程度地解决了集中存储、存储管理和存储空间共享的问题，特别是在数据的可用性、系统容量和系统性能的动态可扩展性方面明显优于 NAS 系统，但它存在使用复杂、数据共享的颗粒度过大，以及难于直接支持文件级的数据共享。基于对象存储技术（Object-Based Storage, OBS）是采用了“ 对象 ”数据组织，克服了 NAS 与 SAN 中不足，它既有“ 块 ”接口的快速，又有“ 文件 ”接口的便于共享。对象

使文件数据和存储元数据管理进行分离，突破了 SAN 的文件共享限制和 NAS 系统中常见的数据路径瓶颈。对象由数据、属性及操作组成，在安全性、跨平台数据共享、高性能和可扩展性特性中更胜一筹。为满足文件服务、事务处理、流媒体服务等不同类型的应用需求，存储对象应是可变长的，并可以包含任何类型的数据，如文件、数据库记录、图像以及多媒体视频音频等。与基于固定的块大小访问的块存储设备不同，对象可动态地扩大和缩小，即数据的种类、属性不同，操作方法应简繁有别。对象的属性用于描述对象的特征，如多媒体数据对象的服务质量（Quality of Service, QoS）属性描述该对象的网络延迟要求；文件对象的属性描述文件的访问权限等。对象的操作类型应多种多样，既有基本的文件访问时对存储设备的操作，也有数据库、流媒体等访问时对存储设备的操作，同时操作类型还应能根据应用需求进行调整和扩充。

1.2 面向对象的存储技术简介

对象存储文件系统的核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备[10]（Object-based Storage Device，OSD）构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布，对象存储文件系统通常有以下几部分组成[11]。

1.2.1 对象

对象是系统中数据存储的基本单位，一个对象实际上就是文件的数据和一组属性的组合，这些属性可以定义基于文件的磁盘阵列（Redundant Arrays of Independent Disks, RAID）参数、数据分布和 QoS 等，而传统的存储系统中用文件或块作为基本的存储单位，在块存储系统中还需要始终追踪系统中每个块的属性，对象通过与存储系统通信维护自己的属性[12]。在存储设备中，所有对象都有一个对象标识，通过对象标识 OSD 命令访问该对象。通常有多种类型的对象，存储设备上的根对象标识存储设备和该设备的各种属性，组对象是存储设备上共享资源管理策略的对象集合等。

对象由数据、属性及相应操作代码组成，由对象 ID（OID）号标识。也就是

说，存储对象中既包含数据，也包含了数据的操作代码，是一个具有标识“接口”、“状态”和“操作”的三元组，是一个封装为一体的一般概念上的对象。将对象分类赋予不同的属性，并构成层次结构，使之服务于不同的用户，如分成根对象（Root Object），分区对象（Partition Object），集合对象（Collection Object）和用户对象（User Object），并以 Partition_ID 和 User_ID 标枳[13]。

而用户对象又由数据和两种相关的属性组成：

（1）应用数据：应用数据本质上与传统系统中的普通文件等价。它可以用类似文件的操作命令如 open, close, read 和 write 访问。

（2）存储属性：这些属性被存储设备用来管理数据。它包括对象 D;、块指针、逻辑长度等。它与传统文件系统中的节点级别的属性类似。

（3）用户属性：用户属性对存储设备而言是透明的，用来描述对象的服务质量需求、选用何种 RAID、容量配额的大小等。

1.2.2 对象存储设备

对象存储设备（OSD）是具有一定的智能设备，包括处理器、RAM 内存、网络接口、存储介质如磁盘等以及运行在其中的控制软件，目前国际上通常采用刀片式结构实现对象存储设备。OSD 提供物理视图（inode 层），将数据访问的负担分布到对象存储系统的不同部分，以避免类似 NAS 系统中元数据瓶颈问题。inode 工作分布到各个智能 OSD 设备中，因此 90%的元数据管理是分布到实际存储数据的 OSD 当中。OSD 提供四个主要功能：

（1）数据存储。OSD 管理对象数据，并将它们放置在标准的磁盘系统上，OSD 不提供块接口访问方式，Client 请求数据时用对象 ID、偏移进行数据读写。

（2）智能分布。OSD 用其自身的 CPU 和内存优化数据分布，并支持数据的预取。由于 OSD 可以智能地支持对象的预取，从而可以优化磁盘的性能。

（3）每个对象元数据的管理。OSD 管理存储在其上对象的元数据，该元数据与传统的 inode 元数据相似，通常包括对象的数据块和对象的长度。而在传统的 NAS 系统中，这些元数据是由文件服务器维护的，对象存储架构将系统中主要的元数据管理工作由 OSD 来完成，降低了主机的开销。

（4）保证安全。OSD 存储系统中的每一条命令或者传输的数据，都必须伴随对发送方和操作的认证。OSD 检查每个接入的连接是否相应的授权，拒绝非法、无效和过期的连接。

1.2.3 元数据服务器

元数据服务器（Metadata Server, MDS）协调客户机与 OSD 之间的交互，管理与上层文件系统有关的元数据，它提供下列功能：

（1）安全策略

包括身份验证、授权证书管理、访问控制等。新的 OSD 加入到网络中时，由元数据服务器对 OSD 的身份进行验证，并向新加入的 OSD 颁发证书，元数据服务器周期的更新证书，确保每个 OSD 都是合法的。同样，当客户机请求访问 OSD 时，先由元数据服务器进行身份验证，然后才向客户发送证书授权访问。

对文件的每一对 Open(Create)/Close 操作，客户只须在 Open(Create)时向元数据服务器发请求取得授权（证书），此后，客户将用该证书访问 OSD，直到文件被关闭。元数据服务器用访问控制位（Access Control Bit）描述客户的访问权限，包含在证书中返回给用户。客户机访问 OSD 时，OSD 检查访问控制位，给予客户机相应的权限。

（2）Cache 一致性维护

在 OBS 系统中，Cache 一致性是至关重要的，因为 Cache 存在于客户机，OSD 和元数据服务器中，必须保证三者的统一。而数据与元数据在系统中是分开存放的，这使一致性的维护变得复杂起来。

根据 OBS 的访问模式，可按如下方法实现 Cache 的一致性。当多个客户机访问相同文件时，每个客户机在元数据服务器注册一个“ 回调事件 ”（CallBack）。当另外的客户机修改了该共享文件时，元数据服务器将产生一个回调事件，通知所有打开该文件的客户机更新本地 Cache。客户机收到回调事件通知后，将本地 Cache 置为无效，重新向元数据服务器获取新的元数据，然后再从 OSD 读取最新数据。

（3）文件目录的元数据管理

由于与块/扇区有关的元数据管理（大约有 90%的负载）已交由 OSD 负责，元数据服务器只管理与文件目录有关的元数据（10%的负载），即将文件目录映射为对象。

采用哈希（Hashing）的方法把文件或一段文件映射为对象，但采用哈希的方法不利于系统的扩展。因为哈希函数与 OSD 的数目有关，当系统中加入新 OSD 设备时，哈希方法将不能适应这种变化。采用线性映射表可避免这个问题。元数据服务器维护文件名或一段文件与 OSD 的 IP 地址、对象 ID 的对照表。当客户机请求打开文件时，元数据服务器返回文件或文件段所在 OSD 的 IP 地址及对应对象 ID，客户机据此访问 OSD。当客户机创建新文件或对文件写入新数据时，元数据将根据 OSD 的负载情况、分配策略、是否对文件进件分条（RAID 功能）等因素找到一个合适的 OSD，并指定一个对象 ID。元数据据服务器将新的映射信息插入线性表，同时也将这些信息返回给客户机。

1.2.4 对象存储文件系统的客户端

为了有效支持 Client 支持访问 OSD 上的对象，需要在计算结点实现对象存储文件系统的 Client，通常提供 POSIX 文件系统接口，允许应用程序像执行标准的文件系统操作一样[14]。

1.3 面向对象的存储技术的历史和发展

OBS 的最早研究可追溯到 1980 的两个面向对象操作系统，一个是卡内基梅隆大学的 Hydra OS，另一个是 Intel 的 Imax-432 OS。这两个操作系统最先用可变长的对象来存储文件、进程状态等多种数据。1980 年，麻省理工大学第一次在他们的 SWALLOW 项目中实现了分布式对象存储，成为该领域的先驱。

OBS 获得的实质性进展源于卡内基梅隆大学并行数据实验室（Parallel Data Lab）的 NASD（Network-Attached Security Disks）项目[15]。NASD 着眼于开发低成本高效的存储系统，系统中采用基于对象接口的附网磁盘，运用加密技术和基于证书的认证机制加强系统的安全性。以 NASD 为基础，在 NSIC（National Storage Industry Consortium）的赞助下，存储业界几个公司合作将 NASD 发展为

Network-Attached Storage Devices（NASD），充分利用了基于对象的接口。在 NSIC/NASD 的基础上，存储网络工业协会（SNIA）成立了 OSD 工作组（OSD TWG），起草定义了 OSD 的命令集，并提交给 T10。T10 着手进行基于对象存储命令的标准化（SCSI Object-Based Storage Device Commands）[13]，并将之纳入 SCSI-3 协议框架中，目前已进行第十一次修订。

加州大学圣克鲁斯分校的存储系统研究中心也致力于 OBS 的研究工作，已开发了 OBFS（Object-Based File System）。OBFS 的目标是支持 10，000 个客户机的并发访问，100GB/sec 的吞吐量，2 P 字节的系统总容量。该系统在元数据管理方面采用了一种独到方法，该法基于 LH3（Lazy Hybrid Hashed Hierarchical）目录管理，将元数据进行分区管理[16-18]。元数据服务器组成集群方式，以支持的高效、灵活、可扩展的元数据管理。

EMC 的 Centera 是一种基于内容寻址（Content Addressed）的存储系统，也具有基于对象存储的特征。Centera 中的对象也包含数据和元数据，Centera 根据对象的内容（包括元数据）产生一个 Key 返回给用户。Key 有几个用途：在 Centera 中维护一个 Key 到对象的一维平坦映射，Key 充当 ID 号进行快速检索；由于 Key 由对象内容计算出来，用户可根据 Key 对返回的对象进行验证；通过计算 Key，Centera 可去掉重复对象。Centera 的这种方案有缺点，即对象在建立时其内容必须可得到，计算 Key 也会影响性能，因此，Centera 的这种方法只适用于定长大小数据。

IBM Haifa Research Lab 实现了一个独立的控制器形式的 OSD，称为 Antara。Antara 用自已开发的协议（Antara Protocol）进行通讯，该协议类似于 T10 制定的 SCSI OSD 命令集标准[13]。Antara 的输入输出的通讯模块是可更换的独立模块，因此 Antara 可运行于多种网络之上[19,20]，包括 IP 网络和 Fibre Channel。

已有多家公司在他们的存储系统中应用了 OSD。有 IBM 的 Storage Tank，National Laboratories 和 Hewlett-Packard 的 Lustre File System，Panasas 的 Active Scale File System[21]，IBM Haifa Research Laboratories 开发的 zFS。

早期的 NASD 建立在两种流行的经改装的分布式文件系统 NFS 和 AFS 之上，

分别称为 NFS-NASD 和 AFS-NASD。在这两种系统中，文件与对象的映射采用了一种简单的方式：一个文件对应一个 NASD 对象，文件的偏移地址就是对象的偏移地址。普通文件属性相应的变为对象属性，这样上层的文件系统需要获得文件属性时可从对象中查获。由客户机发出的读写数据及读属性等常用请求由 OSD 处理，而其余不常用的请求则由元数据服务器处理。NASD 的 OSD 原型运行在 Digital UNIX 系统之上，用 UDP/IP 之上的 DCE RPC 作为接口的通信层。用修改过的 UFS 文件系统管理在磁盘上的对象。

Lustre[22]原是一个 SAN File System，为提高系统的可扩展性引入了 OSD。但 Lustre 将一组 OSD 由一个 OST（Object Storage Target）管理，OST 的对外接口与 T10 正在标准化的 OSD 命令类似。Lustre 把存储设备分为 OST 与 OSD 两个层次，使得整个系统独立于 OSD，系统引入新的 OSD 而不影响原有系统。

1.4 元数据分配算法的相关研究工作

1.4.1 现有的元数据分配算法

在元数据分配算法方面，已经有很多成熟的算法值得借鉴。原先，元数据的分配策略通常有两种。第一种，目录子树划分法，根据目录子树来划分名字空间。在目录子树划分法中，完整目录树的元数据由单独的元数据服务器来管理。但当单个的文件、目录或是目录子树的情况比较多时，这种策略会出现严重的瓶颈。而且，必须研究目录层次来决定是否允许对某个文件进行存取。有时在客户端前置 cache 的帮助下这种情况能得到一定程度的缓解，但当大量的客户同时存取同一文件或同一目录时，前置 cache 也不能起任何作用。

第二种，纯哈希算法，用哈希算法来分配元数据服务器中的名字空间。纯哈希算法根据文件标识符、文件名或是其他相关值的哈希序列将元数据分配到元数据服务器，这与目录子树划分法相比工作量分布更为均衡。但是仍然需要研究目录层次来决定是否允许对某个文件进行存取。如果哈希序列是根据完整路径名得到的，那这可能是用户拥有的关于这个文件的唯一信息，一个目录名字的改变会导致大量元数据移动。如果哈希函数使用文件名代替完整路径名（例如 Lustre），

在不同目录下，但名字相同的文件将会被映射到同一位置。从而当对不同目录下但名字相同的文件进行大量并行存取时会产生瓶颈。

上述两种算法在将元数据服务器添加或是移出群时都会遇到困难，因为会有大量的元数据需要移动。目录子树划分法中，因为子树被存储在每个元数据服务器上，因此需要重新划分整个名字空间来保持工作量的均衡。而纯哈希算法中，哈希算法本身需要改变来产生不同范围的输出，也许需要根据新的哈希函数将几乎所有的元数据移至新的服务器。

现在有种新的算法叫 LH（Lazy Hybrid）算法，结合了前面两种算法的优点并避免了他们的缺点。LH 是建立在路径名哈希映射基础上的可升级的元数据管理机制。LH 通过结合目录子树划分法和纯哈希算法的优点消除了他们的瓶颈，改善了系统性能并分散了这些潜在的昂贵操作的耗费。LH 通过消除服务器群中的热点以及将磁盘存取和服务器交流的耗费最小化实现了高性能的管理。

目录子树划分法的一个优点是可以通过联系相关的元数据服务器来存取文件的元数据，因为某个元数据服务器可以存储给定文件路径中的某些或所有目录。带前置 cache 的目录子树划分法则考虑到了同一客户反复存取同一目录情况下的相关有效元数据存取。目录子树划分法的最大缺点是元数据服务器的工作量可能不是很均衡，这会导致系统瓶颈并将第性能。当一个目录子树上的文件组被频繁存取时，存放这个目录子树的服务器会异常忙碌，从而导致相应时间变长甚至丢失请求。

哈希法消除了服务器间工作量不均衡的问题。Lustre 使用文件名尾和父目录标识符作序列的哈希函数来决定元数据的存储，但必须贯穿分层目录来找回元数据。

1.4.2 对象系统对现有策略的改进

在分布策略方面，为了改进哈希策略的可扩展性，出现了自适应哈希算法 (Self-Adaptive Hashing) [23] 。其主要思想有两方面。第一，为每个文件的元数据保持不同版本的哈希函数；第二，按需要重分配已存在文件的数据。

在面向对象的存储系统方面，当前一种非常成功的面向对象系统是 Panasas

存储集群[24]。在 Panasas 存储集群 中，其对象分布策略可描述为：如果一个文件小于 64KB，将会被以 RAID1 的形式境象地放置到前两个组件对象中。如果一个文件大于 64KB，将会被以 RAID5 的形式放置到所有组件对象中。也就是说 64KB 是区分文件大小的边界值，单个对象不大于 64KB。另一种面向对象存储系统是卡内基·梅隆大学的 Lustre 集群文件系统[22]。Lustre 集群文件系统中对象分布策略对应的是逻辑卷管理。在 Lustre 的逻辑卷管理中以 RAID 的形式组织和管理对象。文件被分割成对象，以类似 RAID 的形式分布在各设备中。

在对象文件系统方面，加州大学圣克鲁斯分校的研究人员开发了一种用于对象存储设备的文件系统——OBFS[25]。在 OBFS 中，大小文件的分界是 512KB。研究人员将一种高性能分布式文件系统：LLNL (Lawrence Livermore National Laboratory) [26] 作为大规模分布式文件系统的一个实例。通过对 LLNL 负载数据特征值的分析，得出了以上结果。OBFS 的研究表明对象文件系统也应该适应小文件频繁出现的情况。Roselli 等跟踪过类似的系统，在他们研究的系统中有很多小文件，60-70%的数据为小于 512KB 的文件[25]。

1.5 本文研究的主要内容

在基于对象存储的海量信息存储系统元数据服务器中管理与上层文件系统有关的元数据。提供一种在基于 OBS 的 PB（Peta Bytes）级环境中放置、添加、删除、修改、查找元数据的方法，完成对元数据的高效管理。主要工作包括一下几个方面：

1. 实现了用户文件到对象的映射。
2. 在 PB 级环境中维持文件目录结构并对用户呈现传统树形结构。
3. 实现了用户认证，用户管理。
4. 实现了对 OSD 的管理，保持了各 OSD 间负载均衡。
5. 在多用户高负载情况下快速响应请求。
6. 保证向用户和设备发送的命令格式符合 T10 的标准 OSD 命令。

2　元数据管理的功能设计与实现

2.1　对象存储系统的体系结构和各主要部分

基于对象存储系统 OBSS（Object-Based Storage Systems）的体系结构如图 2.1 所示。它包括元数据服务器集群（MDS cluster）、基于对象存储设备（OSD）和客户端（Client），并通过高速网络相互连接。

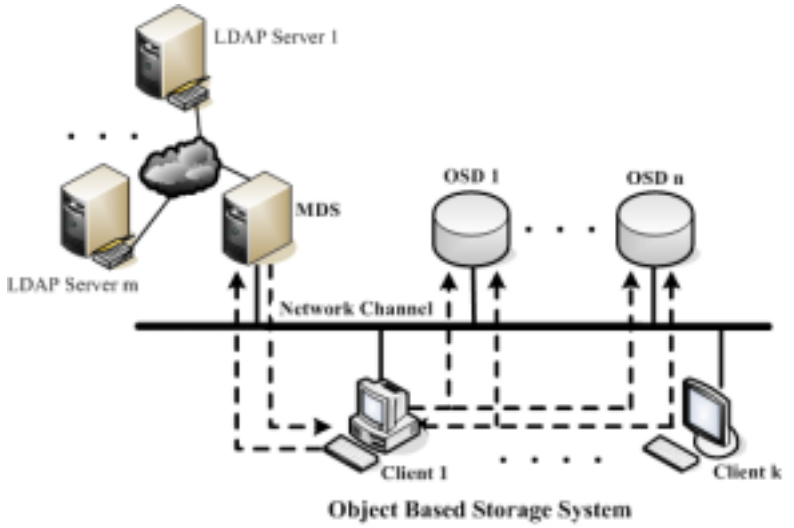


图 2.1　基于对象存储系统 OBSS 的体系结构

基于对象存储系统的模块结构如图 2.2 所示。该系统由元数据服务器 MDS、基于对象存储设备 OSD、客户端管理模块和安全管理模块组成。

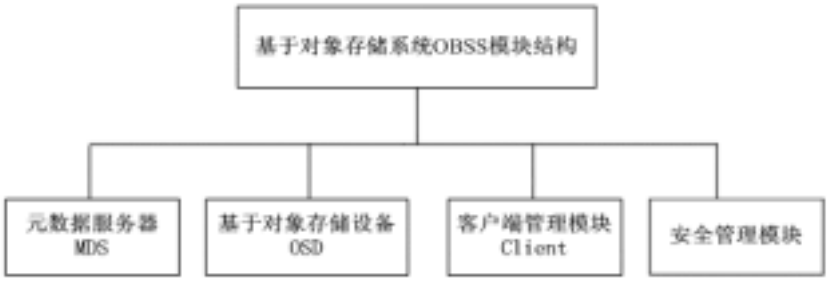


图 2.2　基于对象存储系统得模块结构

元数据服务器、基于对象存储设备和客户端之间的交互关系如图 2.3 所示。

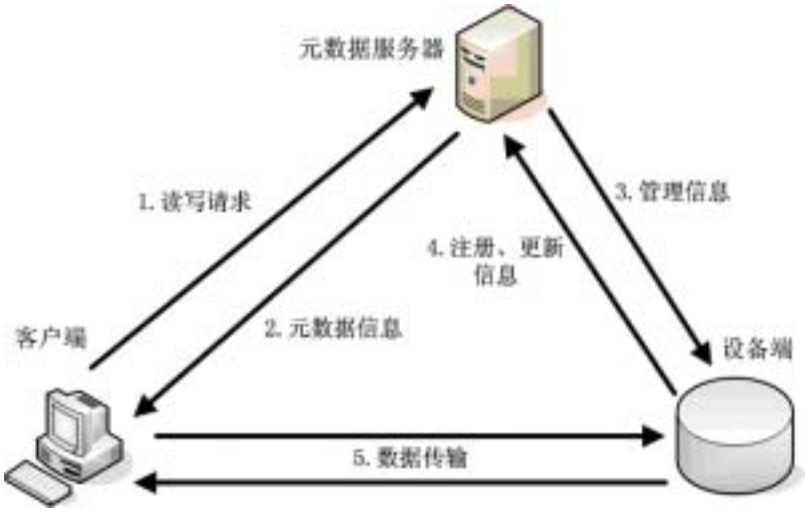


图 2.3　MDS、OSD 与客户端之间的交互关系

2.1.1　MDS 软件模块

元数据服务器 MDS 的软件模块组成和之间的关系如图 2.4 所示。

元数据服务器由文件管理模块、资源管理模块组成和轻量级目录访问协议（Lightweight Directory Access Protocol, LDAP）数据库组成。文件管理模块负责文件命名管理，将文件转换为对象。资源管理模块负责存储资源的管理。LDAP 数据库负责存储元数据及资源信息。

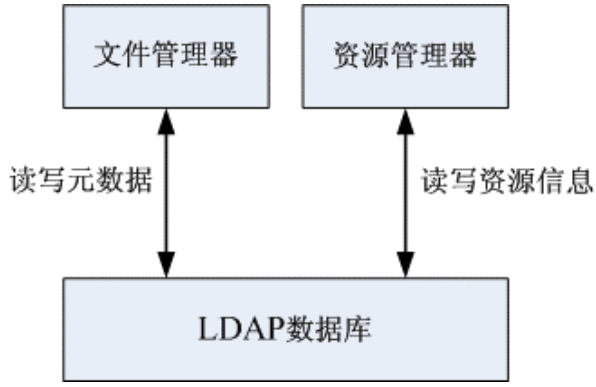


图 2.4　MDS 的软件模块结构及模块之间的关系

客户端发出的文件请求，通过网络通道传递给文件系统管理模块。在文件管理模块中将文件转换为对象并经过相应的存储策略、并发控制和负载平衡处理，然后将用户请求传递给 OSD 或客户端。

2.1.2 OSD 软件模块

基于对象存储设备 OSD 的软件模块结构和处理流程如图 2.5 所示。

OSD 由对象存储控制器、文件系统地址映射模块和块设备管理模块组成。

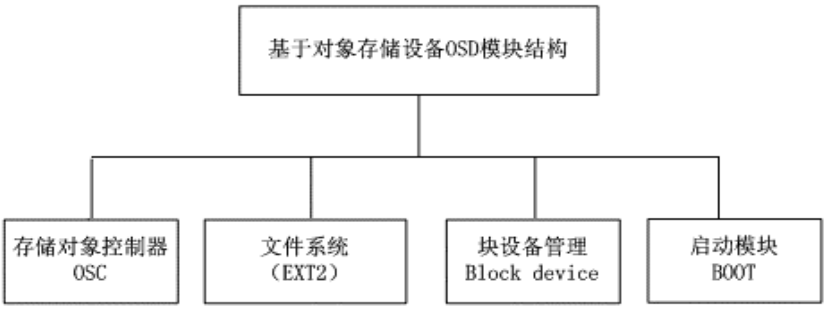


图 2.5　基于对象存储设备 OSD 的软件模块结构

启动模块负责系统初始化工作；对象存储控制器负责存储空间管理、命令管理、对象映射；文件系统地址映射模块负责访问请求的逻辑地址到物理地址的转换；块设备管理模块负责块设备的驱动。

客户端或元数据服务器发来的请求通过网络通道传至对象存储控制器。经过对象存储控制器的存储空间管理、命令处理和对象映射后，转入文件系统的地址转换处理。设备命令和地址经设备管理模块驱动设备工作。

2.1.3 CLIENT 软件模块

Client 的软件模块组成和处理流程如图 2.6 所示。该软件模块由 OSD Client、MDS Client 和本地文件系统信息处理及显示模块组成。

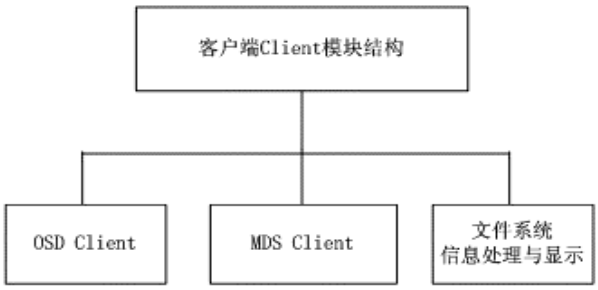


图 2.6　Client 的模块结构

OSD Client 负责与基于对象存储设备 OSD 的命令处理与封装； MDS Client 负责与元数据服务器 MDS 的命令处理；本地文件系统信息处理及显示模块负责本地文件信息的处理与显示。

客户端命令请求通过网络发给 MDS，经过 MDS 的管理与解析，返回与请求相关的 OSD 信息，再将此信息封装成为基于对象的命令，通过 OSD Client 经 iSCSI 通道与相应的基于对象的存储设备 OSD 联系。MDS Client 将请求命令处理后发给元数据服务器 MDS，然后从 MDS 处获取数据信息。OSD Client 模块通过命令处理与封装传递给 OSD。

2.2 元数据服务器软件设计

在以往的 DAS、NAS、SAN 中，由于块设备不能管理元数据，使得服务器/元数据服务器成为瓶颈。而在基于对象存储（OBS）系统中，由于与块/扇区有关的元数据管理（大约有 90%的负载）已交由 OSD 负责，元数据服务器只管理与文件目录有关的元数据（10%的负载）；使系统瓶颈得到极大的缓解。良好的元数据管理将减少不必要开销，加快访问速度，均衡负载，提高系统性能。

对象存储模型显现出分布式存储系统的体系结构特征。对象存储系统由客户端、元数据服务器和对象设备三大部分组成，它采用三方通讯方式提供存储服务，即：客户端向元数据服务器发送读写文件请求，元数据服务器返回文件对应的对象信息和权限，客户端根据对象信息以赋予的权限访问相应的设备。传统系统中元数据和数据在同一台设备，同一个机器和同一个文件系统中[27]。基于对效率的考虑，元数据通常被单独存放在距离其描述的数据较近的物理位置上[28]。在一些分布式文件系统中，数据被存放在可以通过网络直接访问的智能设备上，而元数据由专门的元数据服务器管理[29]。不难发现元数据服务器在对象存储系统中的位置非常重要，是整个系统潜在的瓶颈。元数据服务器的中心任务就是元数据的组织和管理。

OBS 带来的一个明显好处是把存储空间分配交由 OSD 管理。传统的基于块的文件系统由两部分组成：用户相关部分和存储相关部分。用户相关部分通过逻

辑数据结构（如目录和文件）向用户提供用户调用接口；存储相关部分则将这些目录和文件映射到底层物理设备的逻辑块上[30]。在 OSD 构成的存储系统中，上层的用户相关部分不变，而下层的存储相关部分下移到 OSD 中，相应的设备接口也从基于块的接口变为基于对象接口。这样，文件系统的上层只负责把文件名等逻辑名称映射为对象 ID， 这部分工作仍由元数据服务器完成，比传统的元数据服务器，负载大为减小，对象 ID 与磁盘块的映射在各 OSD 内完成。

2.2.1 元数据服务器的功能分析

对象存储系统中元数据服务器的主要任务是：完成元数据管理功能，如文件目录结构维持，通过文件系统调用向用户提供与传统文件系统相同的文件服务。完成对象到文件的映射，管理系统内的对象分布。完成资源管理，包括用户管理、命名、权限管理，对设备进行有效管理，保证系统负载均衡及备份。主要功能可归纳为：

- 元数据的存储、访问和管理： 将元数据存放到数据库或文件系统中，方便高效地对元数据进行存储，访问和管理。由于 LDAP 针对目录访问进行了优化，在元数据管理方面具有一定优势。加之封装了较多常用功能，使开发和维护更加方便。其可扩展性和安全性也十分优异。
- 文件到对象的映射： 映射通过策略库调出一定的算法，根据 OSD 的负载情况，空闲空间的大小，文件本身的特点及 QoS 的要求等因素决定对象 ID 的分配。
- 对象在 OSD 间的分布：根据文件的大小决定如何放置到具体的 OSD 设备中。当文件大小超过一定阈值就采用分片的方式，即将文件平均分成若干份，散列在 OSD 中；否则使用哈希的方法，将文件全路径名哈希成一个整型数，根据这个数字确定放置对象的设备。分片和哈希各有自身的优势，将它们分别应用的大文件和小文件中可以最大限度的发挥算法的特点，使系统性能达到最优。
- 缓冲区管理： 针对元数据访问的特点，在系统中特制一个缓冲区，将最近访问过的元数据缓存到该缓冲区中，以便下次访问命中时直接从缓冲

区中取走数据，不必再产生磁盘 I/O。 缓冲区使用类似哈希链表加双向链表的结构，使用最近最少使用（Least Recently Used, LRU）算法淘汰过时数据，提高内存利用率。

- 向用户提供统一的文件目录结构视图：为了方便用户的使用，系统向用户呈现传统的 UNIX 目录结构，并提供 POSIX 标准文件访问接口。
- 用户管理：完成系统用户的管理，区分不同权限用户，并实现用户的扩充删减及属性的修改。
- 设备管理：对系统内所有设备进行全局管理，包括负载均衡，数据迁移，容灾备份等。

2.2.2 文件管理模块

1. 主要软件模块

元数据服务器由文件管理器，资源管理器和 LDAP 数据库三大部分组成。文件管理器主要负责与客户端的通信，对象分配，元数据管理；资源管理模块主要负责与设备通信，管理设备和用户，设备间负责均衡和调度；LDAP 数据库存放元数据信息和设备信息，也是文件管理模块和设备管理模块的公用平台。

与元数据组织和管理关系较为密切的是文件管理器，其具体的结构如图 2 所示。文件管理器各主要模块包括：

- 客户命令接受模块。负责从网络接受客户发送的读写请求。
- 对象分配、管理模块。负责从客户命令接受模块中提取信息，根据相关信息结合 LDAP 中的信息分配对象，管理元数据。在现阶段具体就是通过文件类型和大小的判断，决定文件到对象的映射。大文件使用分片算法，小文件使用哈希算法。
- 返回客户数据信息模块。将预分配好的对象信息封装成标准的 OSD 命令，通过网络传送给客户。
- 元数据到 LDAP 信息映射模块。将新建的对象信息转换成 LDAP 标准格式，并提交到后台数据库中。而具体的对象数据结构会被存放在对应记录下的属性项中去。再通过 API 函数向后台数据库提交结果。

- LDAP 信息到元数据映射模块。元数据到 LDAP 信息映射模块的逆过程。供对象分配、管理模块调用 LDAP 信息时使用。
- 缓冲区管理模块。在内存中建立一个缓冲区，将最近存入 LDAP 数据库的数据暂时缓存起来，根据读写的局部性原理提高读数据的速度。

软件的层次结构及各模块的调用关系如图 2.7 所示。

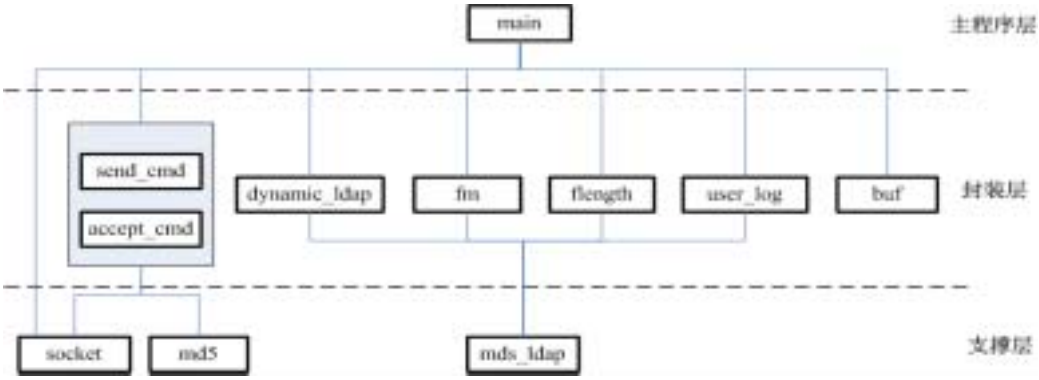


图 2.7　元数据服务器文件管理器各模块关系

在图 2.7 中，各个模块被描述为：**main**：主程序，**buf**：缓冲区管理，**fm**：文件映射到对象的算法，**flength**：文件长度管理（文件长度变化 可能会影响对象的分布），**md5**：md5 加密算法，**accept_cmd**：接受用户命令，**send_cmd**：向用户发送信息 ，**socket**：socket 相关函数，**mds_ldap**：后台数据库 ldap 相关函数，**user_log**：用户管理，**dynamic_ldap**：动态选择后台 ldap 服务器。

各模块可分为三层：主程序层，封装层和支撑层。主程序层包括 main，调用封装层完成整个程序。封装层包括 send_cmd, accept_cmd, dynamic_ldap, 文件管理模块, flength, user_log, buf。完成与客户端通信、动态后台服务器配置、文件到对象映射、文件长度管理、用户认证和缓冲区管理等功能，封装好相应函数为主程序提供调用。由于文件管理模块, flength 和 user_log 模块关系较为紧密，flength 中有对文件管理模块和 user_log 模块函数的调用，文件管理模块中也有对 user_log 模块函数的调用。支撑层则负责向封装层提供更细节的调用。支撑层通过调用相应的库及最底层实现的函数完成它们所包括的函数的功能。其中 socket 模块也直接为主函数提供初始化所需要的 SOCKET API 封装。通过三层模块，语义上完成了从抽象到具体的过程。

2. 服务流程

文件管理服务的主要流程如图 2.8 所示。

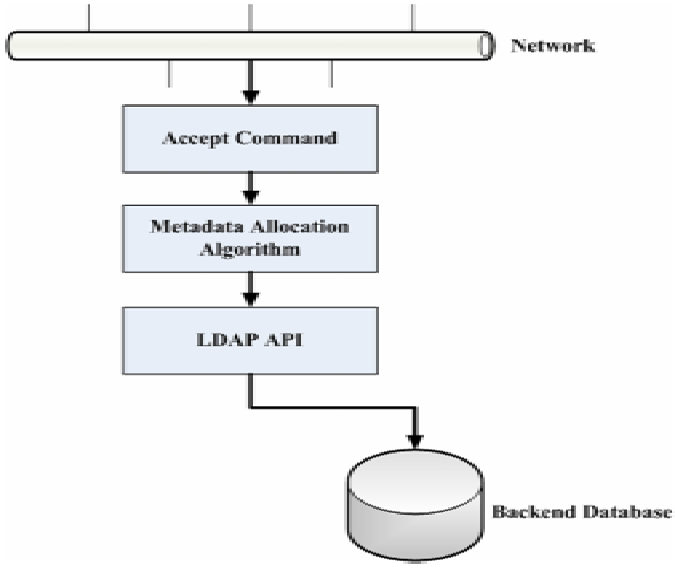


图 2.8　文件管理服务流程

文件管理模块通过网路获得用户命令，通过本地相关算法处理之后交给后台存储服务 LDAP 数据库。

文件管理模块接收到命令，创建线程。然后，根据不同的命令如：read_service（读）、write_service（写）、rm_service（删除）、mkdir_service（创建目录）、ls_service（列表）、login_service（注册）等进行处理，最后发出命令。如图 2.10 所示。这样对用户就形成了传统 UNIX 文件系统的界面，用户只需发送相应命令到服务器，进行创建读写等文件标准操作。而对具体存放的信息则是对象信息，用户和对象设备实际操作的也都是对象信息。实现了对象系统与文件系统的转换和兼容。对象分配、管理模块与元数据到 LDAP 信息映射模块和 LDAP 信息到元数据映射模块通过类似 get (struct　object *) 和 put (struct object *) 的函数交互信息。strcut object 结构包括对象 id，顺序号，对应设备的 IP 地址，对象长度，对应的文件名，对应客户 id，对应客户组 id，创建时间，修改时间，更新时间等信息。保证了文件请求预处理/元数据管理模块和 LDAP 自定义映射机制模块间的相对独立，前者不涉及任何 LDAP 信息的处理。

3. 主要数据结构

用户向元数据服务器发送命令采用下面的数据结构：

```
struct cmd    {
        long        uid;                                //客户 id
        long        gid;                                //客户组 id
        char        filename[FILE_MAX_SIZE];        //文件名
        uint64_t    filelength;                         //文件大小
        int         opcode;                             //操作指令
};
```

元数据服务器在接受到用户命令之后通过客户 id 和用户组 id 对用户进行判断，而文件名、文件大小和操作指令记录了用户一次操作的信息。

在对象存储系统中对象标识符可描述为：

```
struct OID {
        uint64_t    partition_id;       //对象分区 id
        uint64_t    user_id;            //对象用户 id
};
```

根据 OSD 标准对象由一个 128 位的唯一标识来确定，在系统中将这 128 位分割为两个 64 位 id，分别对应标准协议中的对象分区 id 和对象用户 id。

在服务器端的对象数据结构，就是服务器将要向后台数据库存放的元数据信息可描述为以下的数据结构：

```
struct object {
        struct    OID    object_id;                  //对象 id
        int       index;                              //分片顺序号
        ulong          ip;                            //设备 IP
        uint64_t    magic;                            //分片长度
        char      filename[FILE_MAX_SIZE];          //文件名
        long      uid;                                //属主 uid
        long      gid;                                //属主 gid
        unsigned    short    permission;              //读写许可
};
```

分片顺序号和长度、文件名、设备 IP 等信息结合对象标识确定了元数据在后台服务器中的存储位置，而属主和许可信息则是和权限相关的元数据信息。这些信息一起构成了对象系统的元数据信息。

服务器端向客户端发送的的对象数据结构，除了上面的对象数据结构外还增加了操作指令和链表指针元素。

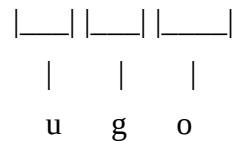
```
struct object2user {
        struct object obj;
        uint16_t opcode;                //对应 service_action
        unsigned short digest[16];      //MD5 报文摘要
        int64_t size_change;            //对象大小的变化，用于容量管理
        struct object2user *next;
};
```

由于元数据服务器与客户端的通信涉及加密认证及多个对象，所以有必要将对象结构再作一次封装，提供 MD5 加密的相关信息和组成对象链表的指针。

用户再登录和访问对象时要进行必要的认证和权限设置。对象的访问权限主要用一个短整型数来描述，这点和 UNIX 文件系统类似。具体可描述为：

unsigned short permission //文件读写许可

permission 包括 r w x r w x r w x 9 位，与 UNIX 中的文件读写许可相同。



这样每个对象都有属主，根据读取对象用户 uid 和 gid 的判断来参照文件读写许可，采取灵活的权限策略。可以实现 UNIX 文件权限的大部分内容。对 OID（对象 ID）和 opcode（操作码）进行 MD5 加密，结果放在摘要中传给客户端由客户端转发至 OSD。

4. 对象管理

对象的管理主要包括：对象在 OSD 间的分布，对象在后台数据库中的存放管理和对象长度动态管理。

对象在 OSD 间的分布可简单地描述为：大文件采用分片，小文件直接对全路径名哈希。大小文件阈值暂时定为 1MB（OBFS 为 512KB）。在第四章中将详细介绍该算法的设计实现及性能分析。

对象在后台数据库中的存放管理的主要由元数据到 LDAP 信息映射模块完成。将新建的对象信息转换成 LDAP 标准格式，并提交到 LDAP 后台数据库中。比如文件 /test/usr/src/prog.c 对应的对象会被存放到类似 dn: cn=prog.c, cn=src, cn=usr, cn=test, cn=root, dc=example, dc=com 的记录中去。在下一章中将重点介绍 LDAP 数据库的优势和在对象系统中的具体使用。

由于写操作常改变文件的大小，从而影响对应文件的大小。当文件改变超过一定大小的时候可能要增加或删除某个对象。这些都需要一个合理的算法进行管理。对象长度动态管理算法如图 2.9 所示。当小文件大小变大，超过设定的大小文件阈值而变成大文件，也就是说要分割成几个对象时，增加新的对象。而大文件增大时则只修改最后一个对象的大小，这样做是考虑到大文件本身的对象数已经较多，而且在不同设备上，再增加新的对象将会给系统增加新的负担。因为增减对象要对设备进行操作并影响系统负载调度策略，所以比较慎重。

```
从缓冲区/数据库中获取整个文件的大小
与传入参数中的文件大小进行比较
if (不变)
        跳出
else if ( 变大 )
        if (小文件变小文件)
                修改对象大小
        endif
        if (小文件变大文件)
                重新分配该文件对应的对象
                修改第一个对象大小
                向数据库追加其他新增对象
        endif
        if (大文件变大文件)
                修改最后一个对象大小
                更新缓冲区
        endif
else    // 变小
        if ( 减小的长度未超过最后一个对象大小 )
                更新最后一个对象的长度
        endif
        if ( 减小的长度恰好等于最后一个对象大小 )
                删除最后一个对象
        endif
        if ( 减小的长度大于最后一个对象大小 )
                n=(前后长度之差-最后一个对象长度)/分片长度
                删除后n + 1个对象
                修改第N - n - 1 个对象大小    // N为原来总对象数
        endif
endif
更新缓冲区
```

图 2.9　对象长度动态管理

2.2.3　资源管理模块

1. 用户管理

用户管理部分由命令行实现，直接通过控制台的输入完成用户管理的各种操作。由控制台输入之后经过程序的处理和验证提交后台数据库存储起来。

相关的用户数据结构：

```
typedef struct users{
```

```
    long    uid;                          //用户 uid
    long    gid;                          //用户 gid
    char    username[FILE_MAX_SIZE];      //用户名
    long    password;                     //Hash 后的密码
    char    homedir[FILE_MAX_SIZE];       //用户主目录
    unsigned mask;                        //用户读写权限掩码
    struct users *next;                   //下一指针
}users;
```

用户管理的流程如图 2.10 所示。

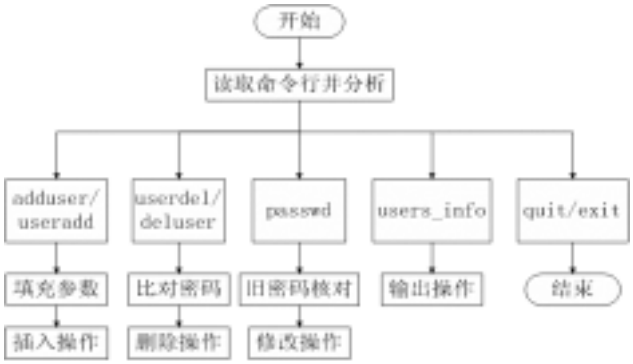


图 2.10 用户管理处理程序流程

2. 设备管理

设备管理由设备监听/探测，设备注册，设备属性管理，设备注销四个模块组成，如图 2.11 所示。

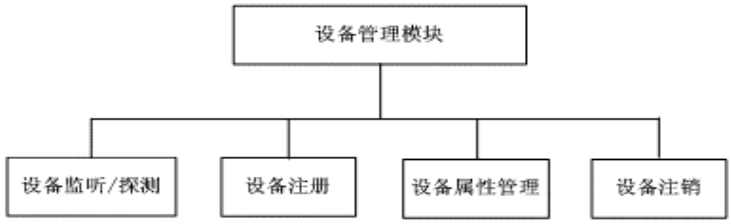


图 2.11 设备管理模块结构

服务器使用 osd_get_one_attr 发起探测，得到设备属性。设备属性管理主要是设备的容量管理。这些模块的流程相似，都是首先接收信息，然后在内存中形成链表结构，最后写入后台数据库中。

相关的数据结构：

typedef struct devinfo {

```
    ulong    ip;                //IP 地址
    int          type;          //设备类型
    int          busy;          //设备状态
    uint64_t     totalcap;      //总容量
    uint64_t     freesp;        //空闲空间
    uint64_t     partitions;    //分区数目
} devinfo;
```

为实现设备的动态注册，必须先让设备主动传送出自己的 IP 地址（socket 通信实现）。服务器循环监听，在接收设备主动向服务器发送的自身 IP 地址后开启新的进程，通过 iSCSI 提供的函数填写 g_target[]数组，然后建立 iSCSI 通道，调用 iscsi_osd_command 接口，获取 root 对象的属性值，接着填写 devinfo 结构的成员，最后通过函数 add_dev 增加设备记录到后台数据库。

设备属性管理主要对应于设备的容量管理，在对象更新（如 write）操作后中直接修改相关域进行更新。设备移除可以通过函数 del_dev 来完成。首先触发数据的备份和迁移，接着删除对应的设备记录，然后提交给后台数据库。

2.3 本章小结

本章主要介绍了元数据管理的功能设计与实现，即元数据服务器软件的设计和实现，主要包括文件管理模块和资源管理模块。着重介绍了设计思想、主要功能和模块。

本章从对象存储系统的体系结构入手，简要介绍了整个系统的组成和元数据服务器的位置。分析了元数据服务器的功能要求，包括用户界面服务的需要和元数据自身的组织管理，后者在下一章还会重点介绍。并根据功能的要求从主要软件模块、服务流程、主要数据结构、对象管理以及用户管理和设备管理对元数据服务器进行了完整的诠释。

3　元数据的组织与存储

元数据服务器文件管理器与设备管理模块是通过 LDAP 后台数据库联系在一起，两个模块的信息都存在 LDAP 的后台数据库中，可以相互调用相关信息。文件管理模块的元数据和设备管理模块中的管理信息都可以保存在 LDAP 数据库中不同的 OU 或者 O 之类的域中（类似于关系数据库的表），通过查找，统计特定域中的各记录项（Distinguished Name，DN）来完成对所需信息的调用。另外在设备管理模块中也应当由与文件管理模块中类似的 LDAP 信息到管理信息映射模块和管理信息到 LDAP 信息模块。不难发现 LDAP 数据库成为了各模块的纽带和系统的基石。

3.1　LDAP 简介

LDAP 的英文全称是 Lightweight Directory Access Protocol，一般简称为 LDAP。它基于 X.500 标准，但更加简单并且可根据需要进行定制[31]。与 X.500 不同，LDAP 支持 TCP/IP，这对访问 Internet 是必须的。LDAP 的核心规范在 RFC 中均有定义，所有与 LDAP 相关的 RFC 都可以在 LDAP man RFC 网页中找到。就像 Sybase、Oracle、Informix 或 Microsoft 的数据库管理系统（Data Base Management System, DBMS）是用于处理查询和更新关系型数据库那样，LDAP 服务器也是用来处理查询和更新 LDAP 目录的。换句话说 LDAP 目录也是一种类型的数据库，但是不是关系型数据库。不像被设计成每分钟需要处理成百上千条数据变化的数据库，例如：在电子商务中经常用到的在线交易处理（On-Line Transaction Processing, OLTP）系统，LDAP 主要是优化数据读取的性能。

LDAP 协议是跨平台的和标准的协议。实际上，LDAP 得到了业界的广泛认可，因为它是基于 Internet 标准。LDAP 服务器可以是任何一个开发源代码或商用的 LDAP 目录服务器（或者还可能是具有 LDAP 界面的关系型数据库），因为可以用同样的协议、客户端连接软件包和查询命令与 LDAP 服务器进行交互。与 LDAP 不同，如果软件产商想在软件产品中集成对 DBMS 的支持，那么通常都要对每一个数据库服务器单独定制。

LDAP 允许用户根据需要使用访问控制列表控制对数据读和写的权限。例如，设备管理员可以有权改变员工的工作地点和办公室号码，但是不允许改变记录中其它的域。访问控制列表可以根据访问数据的用户、访问数据内容、数据存放的位置以及其它对数据进行访问控制。LDAP 对于存储需要从不同的地点读取但是不需要经常更新的数据信息最为有利。例如，以下这些信息存储在 LDAP 目录中是十分有效的：公司员工的电话号码簿和组织结构图；客户的联系信；计算机管理需要的信息，包括 NIS 映射、Email 假名，软件包的配置信息，公用证书和安全秘钥等。

大多数的 LDAP 服务器都为读密集型的操作进行专门的优化。因此，当从 LDAP 服务器中读取数据的时候会比从专门为 OLTP 优化的关系型数据库中读取数据快一个数量级。也是因为专门为读的性能进行优化，大多数的 LDAP 目录服务器并不适合存储需要经常改变的数据。例如，用 LDAP 服务器来存储电话号码是一个很好的选择，但是它不能作为电子商务站点的数据库服务器。

LDAP 目录以树状的层次结构来存储数据，类似于自顶向下的 DNS 树或 UNIX 文件的目录树。就像 DNS 的主机名那样，LDAP 目录记录的标识名 DN 是用来读取单个记录，以及回溯到树的顶部。

3.2 在对象系统中使用 LDAP

Linux ext2 文件系统的最大容量为 4TB，而 LDAP 通过多台服务级联可以达到 PB 级容量[31]。对象存储系统中的元数据不包括块信息，大部分元数据读多写少，长度固定且较小，适合数据库特别是 LDAP 数据库模型。另外 LDAP 允许你根据需要使用访问控制列表控制对数据读和写的权限。在安全方面比单纯的文件系统更有优势。综合 LDAP 的夸平台、容量、性能和安全考虑，元数据存储平台使用 LDAP 而不是传统的文件系统。

在元数据到 LDAP 信息映射模块中。新建的对象信息被转换成 LDAP 标准格式，并被提交到 LDAP 后台数据库中。比如文件 /test/usr/src/prog.c 对应的对象会被存放到类似 dn: cn=prog.c, cn=src, cn=usr, cn=test, cn=root, dc=example, dc=com 的记录中去。而具体的对象数据结构会被存放在此条记录下的属性项中去，如 cn，sn 和自定义的属性等。再通过如 ldap_open()，ldap_bind()，ldap_add()等 API 函数

向后台数据库提交结果。而在 LDAP 信息到元数据映射模块中，元数据到 LDAP 信息映射模块的逆过程。对象在分配、管理模块调用 LDAP 信息时使用。

3.3 性能分析及优化

尽管理论上 LDAP 具有更高的性能，但在实际应用当中由于受到实际系统的影响，在元数据读写量不大的情况下其性能反而不如 ext2 文件系统。这也是由于 ext2 使用了 buffer cache，在 TB 级环境中对文件的读写有优化。针对当前的应用环境，在元数据服务器中加入了自己的缓冲区，使性能的到了提升。测试条件为 8K 个目录（记录），分为 2K 组每组深度为 4 层。1K 个线程并发访问。

3.3.1 缓冲区的设计

为了加快读速度，可以在系统中加一个缓冲区，将最近访问的元数据存放在内存中。当用户访问时首先到这个缓冲区中通过哈希的方法查找数据，如果有则直接从内存中取走数据，不再访问磁盘或 LDAP 服务器。根据读写的局部性原理，最近读写的文件在近一段时间内被读写的概率仍很高，这样建立缓冲区将使下次命中的读操作避免从磁盘读写数据，可以明显提高读的速度。而读操作相对于写操作更加频繁，根据“ 加快经常性事件 ”原理，系统的整体速度将的到提高。缓冲区结构如图 3.1 所示。

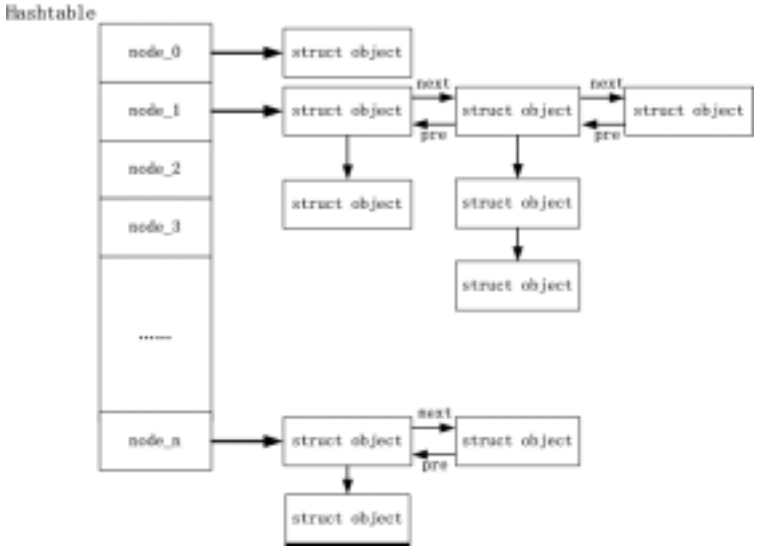


图 3.1　缓冲区结构

在这里，hashtable 是一个数组，数组元素是一个指针，指向一个存放 struct object 的空间。而 struct object 存放的就是要发送到客户端的元数据信息。为了处理冲突，多个 struct object 连成一个双向链表。

当用户发送的信息到达服务器端时，首先将所得到的文件名哈希成一个整形数，以这个数为下标找到对应的数组元素。再以这个数组元素为指针找到元数据信息。如果此时的元数据信息有效，且其中的文件名与用户发送的文件名吻合，说明就是用户要找的元数据，则可直接取走。如果元数据无效，说明该数组元素自初始化以来还没有被填充过，由系统再生成元数据然后填充到数组元素中，再写到磁盘或 LDAP 数据库中。如果元数据有效但不是用户所需数据则顺着链表向后找。若找到则取走元数据，若找不到则填充新的元数据，新的元数据对应的链表项加在链表头。此时如果链表节点数达到最大值，则将最后链表末尾项丢弃。这样就简单的实现了 LRU 算法。

3.3.2 对比测试

对使用 LDAP 数据库保存元数据及直接使用 ext2 文件系统保存元数据两种方案进行了对比，并构建了简单的缓冲区。测试条件及结果如下。测试条件为：8K 个目录（记录），分为 2K 组每组深度为 4 层。1K 个线程并发访问。哈希算法采用如下函数：

```
int hash(const char * name, int len)
{
    int    ret = 0;
    char    c;
    while( len-- ) {
        c = *name++;
        ret = (ret + (c>>4) + (c<<4))*11;
    }
    return ret & HASH_MASK;
}
index = hase( name, strlen(name));
index = index + (index>>6) + (index>>12);
```

测试结果如表 3.1。

表 3.1　LDAP 与 ext2 对比测试结果

未使用 cache 的 ext2 (ms)	使用 cache 的 ext2 (ms)	未使用 cache 的 ldap (ms)	使用 cache 的 ldap (ms)
Total：40745	**Total：40072**	**Total：88120**	**Total：44282**

对于 ext2 来说，自制的缓冲区效果并不明显。这是因为 ext2 的 buffer cache 已经十分完善。而对于 LDAP 来说性能提高很明显，并且性能已经接近了 ext2。

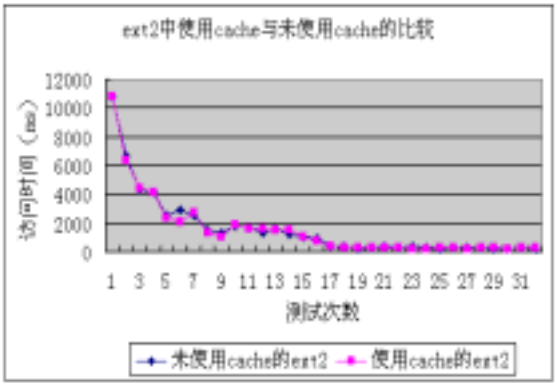


图 3.2　ext2 中使用与未使用 cache 的比较

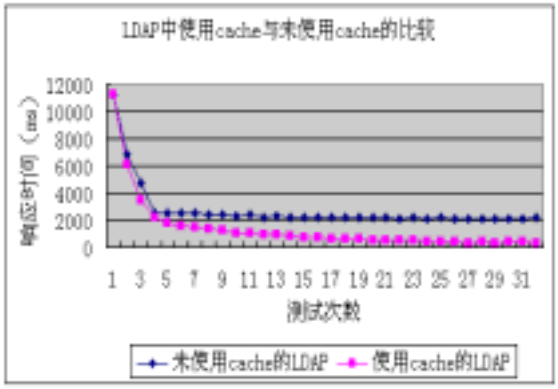


图 3.3　LDAP 中使用与未使用 cache 的比较

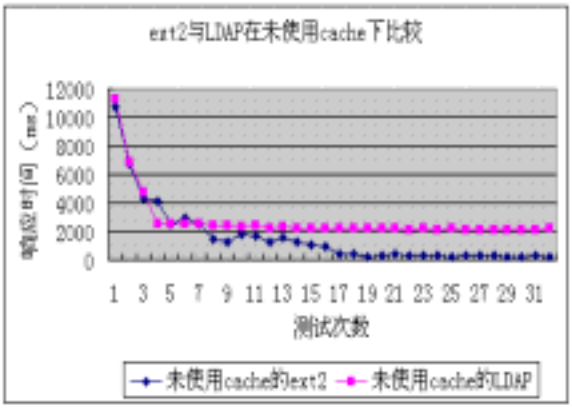


图 3.4 ext2 与 LDAP 在未使用 cache 下的比较

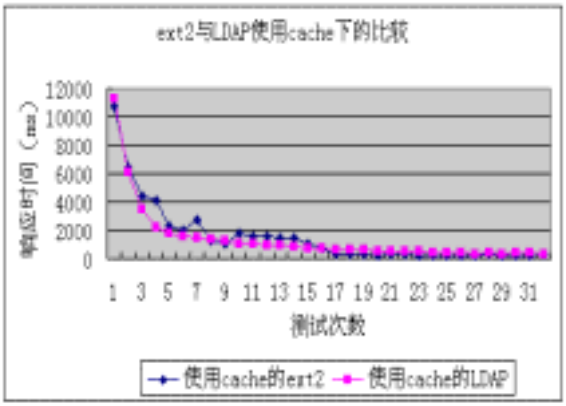


图 3.5 ext2 与 LDAP 在使用 cache 下的比较

比较结果如图 3.2 至图 3.5，。未加缓冲区时，对于 ext2：第一次 10 左右 s，第二次 5s 左右，后面的数据稳定在 0.2s 左右。对于 LDAP：第一次 10s 左右，第二次 5s 左右，后面的数据稳定在 2s 左右。加上缓冲区后 LDAP 性能接近 ext2。对于 ext2 来说，自制的缓冲区效果并不明显。这是因为 ext2 的 buffer cache 已经十分完善。而对于 LDAP 来说性能提高很明显。

3.4 本章小结

本章介绍了对象存储系统中元数据的组织和管理。对象存储系统中元数据服务器使用了 LDAP 作为存放元数据的平台，针对这个平台设计了相应的数据分配算法和数据转换模块。简要介绍了 LDAP 的特点，并结合对象存储系统的特点分析了 LDAP 在容量、性能和安全等方面的优势。描述了 LDAP 在具体使用时，元数据在系统服务器软件和后台 LDAP 数据库间的转换过程和所设计模块的实现。还使用缓冲技术优化了性能。在本章中详细描述了缓冲区的设计和实现，分析了缓冲区对性能的影响。

4　柔性对象分布算法的研究

4.1　对象系统中的对象分布策略

面向对象的存储系统将文件视为对象的集合。这些对象分布在具有自我管理功能的智能设备 OSD 中。对象放置到不同的设备中可以使系统具有更高的容量、吞吐量、可靠性和可扩展性[25]。对于类似 Lustre [22] , GFS [32], AFS [33,34], Coda [35,36] 和 GPFS [37]的分布式文件系统的层次管理、可扩展性和可靠性的研究很多，但对于如何提高对象存储系统中对象放置策略效率的研究则相对较少。对象分布算法在通讯过程的开始就决定了系统的性能。它同时影响着设备间的负载均衡和设备间并行。

在第二章的讨论中已经得出了以下结论：对象存储模型显现出分布式存储系统的体系结构特征。元数据服务器在对象存储系统中的位置非常重要，是整个系统潜在的瓶颈。而元数据服务器的重要任务之一就是将文件映射成一个或多个对象并存放到各设备上，对象的映射与分布策略将直接影响到系统的 I/O 服务性能[38,39]。

当前理论上的对象分布策略有两种。第一种称为哈希算法(Hashing)[40]，使用哈希函数将一个文件映射为一个对象并放置到一个设备中。实际上，当一个文件很大时，如果把这个文件分布到多个设备中，则可以获得更高的并行度。

第二种策略是分片映射策略(Fragment-Mapping)[40]，将文件平均分配到各个设备中。图 4.1 和图 4.2 分别形象的描述了这两种算法。

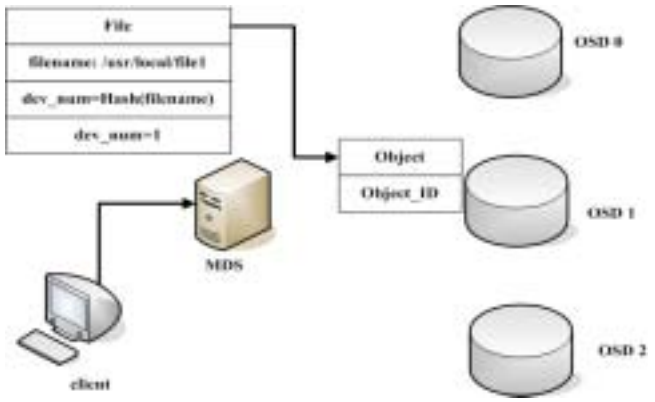


图 4.1　哈希（Hashing）算法

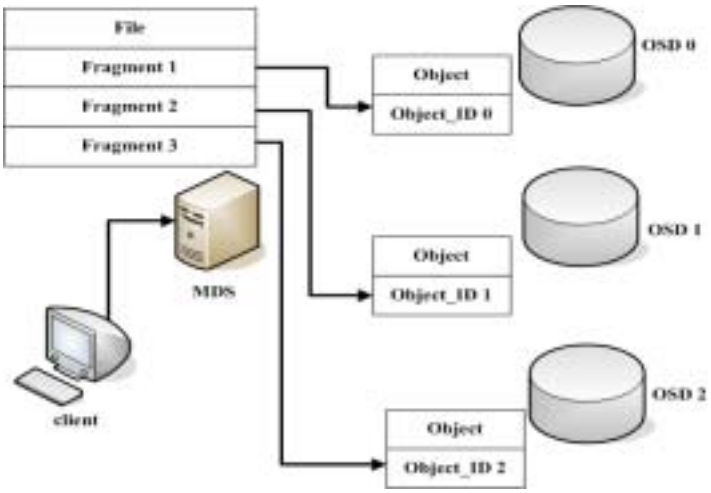


图 4.2　分片（Fragment-striping）算法

哈希和分片算法都有它们各自的缺点。哈希策略使用哈希函数将文件数据近似随机地分布在不同的设备上，从而实现负载均衡和高效的数据分布[40]。但是这种一对一的映射模型对于大文件来说不能够充分利用多设备的并行性。同时由于哈希函数的结果受设备总数的影响，通常假设存储设备数目是不变的，导致它在可扩展性上也存在问题。分片映射方法则利用了各个设备的并行性。这种映射策略有两个优点。第一，简化了客户端的操作，使客户端不需要关心服务器具体的分配策略。第二，改变设备不影响元数据服务器上现有的映射信息，具有良好的可扩展性。但是当文件较小时，向各个设备建立连接的系统开销超过了设备并行带来的好处。特别是当设备较多时，一个文件被分成很多个小对象，无论是创建对象还是建立连接都非常耗时。而且可靠性变得很差，如果一个设备不能正常工作，整个系统将无法运转。

结合以上两者的优点和对象存储系统的具体情况，提出一种柔性的映射与分布策略，小文件直接映射成一个对象并使用哈希策略映射到一个设备中；大文件被分割成多个对象，分别放置在不同的设备里。这样一方面利用了设备并行带来的性能好处，另一方面也减少了不必要的文件分割所引入的系统开销。大小文件的界限对柔性对象分布算法至关重要。选择多少设备、如何选择设备进行分片映射也是算法设计时必须要考虑的因素。这些问题在后面的章节中将依次被讨论。

4.2 柔性对象分布算法的设计

4.2.1 大小文件的边界值

通过统计分析，在 LLNL[26] 的负载中，约有 85%的对象大小是 512KB，15%的对象小于 512KB，由此在 OBFS[25] 的设计中将大小文件的边界值界定为 512KB。OBFS 的实验显示在用户态下实现的 OBFS 在两到三个方面上胜过 Linux ext2 和 ext3 文件系统。OBFS 只有 xFS[41]大小的 1/5，读性能略逊于 xFS，但 OBFS 写性能比 XFS 快 10% – 40%。基于 OBFS 的统计结果，将小于 512KB 的文件当作小对象，其余作为大对象处理。对小对象和大对象采用不同的分布处理。对象存储系统设备端使用的文件系统和 OBFS 类似。所以确定 512KB 为大小文件的边界值更有利于发挥设备端文件系统的优势。

4.2.2 将一个文件映射成对象

OSD 的数目和并行度之间的关系相当复杂。设备越多则传输通道越多，有利于并行传输。但是建立连接所花的时间对系统整体性能有着一定影响，尤其当连接数很多时，这种影响将使性能下降。数据同步也是一项费时的工作。如果很多个对象对应同一个大文件，将这些文件片断重新拼装成文件也将花费不少 CPU 时间。因此，当同一个文件对应的设备数增多时，传输并行度增加，同时也消耗了额外的系统资源。

可以用下面的公式描述这种关系：

$$\frac{T_p}{T} = \frac{n \times a + \frac{1}{n} \times b + \delta(n)}{a + b} \tag{4-1}$$

Tp: 并行传输文件的总时延

T: 串行传输文件的总时延

n: 文件对应的设备数

a: 发送方开销

b: 传输时间

δ(n): 对象间的协调时间，包括同步、校验等，是 n 的函数

公式的分子包括三个部分：n*a，b/n 和 δ(n)。n*a 表示假设连接 n 个设备的发送方开销是连接单个设备的 n 倍。而 b/n 则表明 n 个设备并行传输的带宽是单个设备的 n 倍。

根据 Sun Microsystems 的 Greg Papadopolous 的介绍[14]，互连网络类型的不同（SAN，LAN 或者 WAN 等），飞行时间和传输时间也不尽相同。

Total latency = Sender overhead + Time of flight + (Message size/Bandwidth) + Receiver overhead

SAN 的飞行时间相对于发送和接收开销来说非常短，可被忽略。而在 WAN 中飞行时间相对较长，发送方和接收方开销可被忽略。因此可以把上面的公式中的发送方开销（Sender overhead），接受方开销（receiver overhead）和飞行时间（Time of flight）简化为一个开销因子（Overhead）：

$$\text{Total latency} \approx \text{Overhead} + (\text{Message size} / \text{Bandwidth}) \tag{4-2}$$

$$a = \text{Overhead},\ b = (\text{Message size} / \text{Bandwidth}) \tag{4-3}$$

虽然 δ(n)是自变量为 n 的函数，但是由于其随 n 的变化非常小，可以近似地认为 δ(n)是一个常数。故可以用常数 c 替代 δ(n)，将前面的公式简化为：

$$\frac{T_p}{T} = \frac{n \times a + \frac{1}{n} \times b + c}{a + b} \tag{4-4}$$

进一步变形为：

$$\frac{T_p}{T} = n \times \frac{a}{a+b} + \frac{1}{n} \times \frac{b}{a+b} + \frac{c}{a+b} \tag{4-5}$$

注意到上面公式的倒数就是 T/Tp，即并行传输相对于串行传输的加速比。要是这个加速比最大等价于使 Tp/T 最小。主要关注的是将一个文件映射为多少个对象可以获得最大加速比，即 n 对 Tp/T 的影响，找到最佳的 n。故可以将 a,b 和 c 设为常数。

令 $F(n) = n \times \frac{a}{a+b} + \frac{1}{n} \times \frac{b}{a+b}$，对其求导 $\frac{dF(n)}{dn} = 0$,可以得到 $n = \sqrt{b/a}$。由于 $\frac{d^2F(n)}{dn^2} > 0$，F(n)在 $n = \sqrt{b/a}$. 处有最小值。所以：

$$\frac{T_p}{T} \geq \frac{2\sqrt{ab}}{a+b} + \frac{c}{a+b} \tag{4-6}$$

因此在 $n = \sqrt{b/a}$ 时得到传输最大加速比。如果传输时间（b）很大，或者开销

因子（a）很小时，n 都将变得很大。这也就表明如果传输时间很小，则没有必要并行传输对象。因此在的千兆网环境下，只有当文件很大时才有必要并行传输。

图 4.3 显示大文件（大于 512KB）情况下最小的 Tp/T 对应的最佳文件分片数 n 的情况。在大文件中选取了最常出现的情况：1MB，2MB，4MB，8MB，16MB，32MB, 64MB, 128MB, 256MB, 512MB 和 1GB。

发送端开销指处理器把消息放到网络的时间，包括软件和硬件所花费的时间。这里的开销因子也是类似的情况。假设这种开销不受消息长度大小的影响。事实上也只有非常大长度的消息才会有较大的开销。可以取一种典型的情况：带宽为 1000 Mbits/秒，开销因子是 80 微秒[42]。这种情况在今天的网络情况下非常普遍，可以利用这种有代表性的环境来估算有关参数。公式中的 c/(a+b) 因子对 n 并没有影响，在讨论中暂时忽略掉它。

例如对于 1MB 的文件 Message size = 16*8 Mbits, Bandwidth = 1000 Mbits, b = 1*8/1000 = 0.008s = 8000 μs, a = 80 μs, $\frac{T_p}{T} = \frac{1}{101} \times n + \frac{101}{101} \times \frac{1}{n}$。如图 4.3 所示，对于 1MB 和 2MB 的文件，当 n >= 10 时 Tp/T 的变化率已经不明显了。对于 4MB, 8MB 和 16MB 的文件，当 n >= 20 时 Tp/T 的变化不明显。同样，对大于 32MB 的文件，n >= 40 时 Tp/T 几乎不随 n 变化了。可以得到如下结论：n = 10（对应 1MB~2MB 的情况)，n=20 （对应 4MB~16MB 的情况），n= 40 （对应 32MB~1GB）。

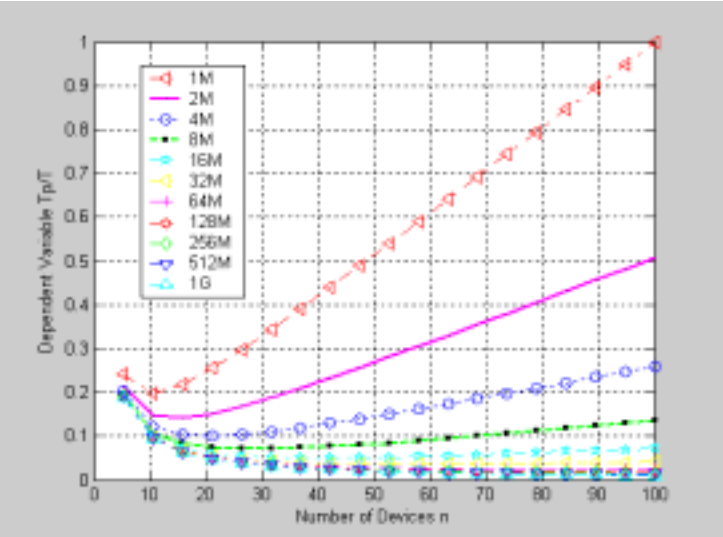


图 4.3 时间比 Tp/T 和文件最佳分片数 n 关系图

4.2.3 OSD 设备的选取

大型的对象存储系统拥有数以百计的 OSD。如何从它们当中选取合适 OSD 来并行并行传输对象呢？随机选择是一种比较好的方法，它实现起来非常简便而且能够保持系统负载均衡。但是并不能保证充分利用那些最快的设备。将设备按某些参数进行排序，例如设备速度，空闲空间，忙程度，设备类型等，然后选择前 n 个（10，20 或 40）设备。这种算法保证了当前状态最好的设备被优先使用。但是在数以百计的设备当中进行这样的排序是否会占用很长时间？是否影响系统性能？所以有必要对这类排序算法进行时间复杂度分析。

在冒泡排序算法中，在总共 N 个对象中选取前 n 个对象所需的时间复杂度为：

$$\sum_{i=1}^{n}(N-i)=\frac{2N-(n+1)}{2}\times n \tag{4-7}$$

时间复杂度：T(n) = O(n^2)，T(N) = O(N)。可以得出 n 对时间复杂度的影响比 N 更显著。因为 n 比较小（10，20 或者 40）而且元数据服务器通常都配备大容量内存和高性能 CPU，这样的排序算法不会对系统整体性能有明显的影响。

4.2.4 柔性对象分布算法

```
INITIALIZATION: Get number of devices (the total number of OSDs) and set dev[] empty.
Objects_allocation( the size of file )
INPUT: The size of file
1:      if the size of file < 512KB then                    //small file
2:              Hashing();
3:      else                                                //large file
4:              if number of devices < 20 then
5:                      N = number of devices;
6:              else if 20 <= number of devices <=40  then
7:                      N = 20;
8:              else
9:                      N = 40;
10:             endif
11:   Sort dev[1], dev[2] ... dev[number of devices] by performance;
12:             if the size of file <= N * 512KB then
13:                     Fragment_ striping (dev[1], dev[2] ... dev[the size of file/512KB]
);
14:             else
15:                     Fragment_ striping (dev[1], dev[2] ... dev[N] );
16:             endif
17:     endif

OSDs is sorted by type, busy, freesp and partitions in function Sort.
```

图 4.4　对象分布算法

所谓柔性，就是灵活变通的意思，柔性对象分布算法根据对象属性的不同情

况采取合适的分布策略。柔性对象分布算法中的设备排序是基于一些描述设备状态的参数，例如设备类型，忙程度，空闲空间，设备分区数和 IP 地址等。图 4.4 用伪码描述了这一算法。算法的基本思路是根据文件大小确定分布方式，并选择适当的 OSD 来存放对象。

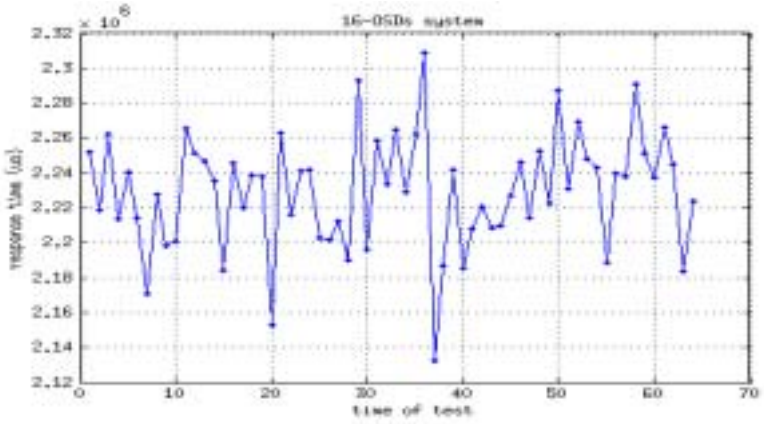
4.3 仿真结果

4.3.1 建立实验环境

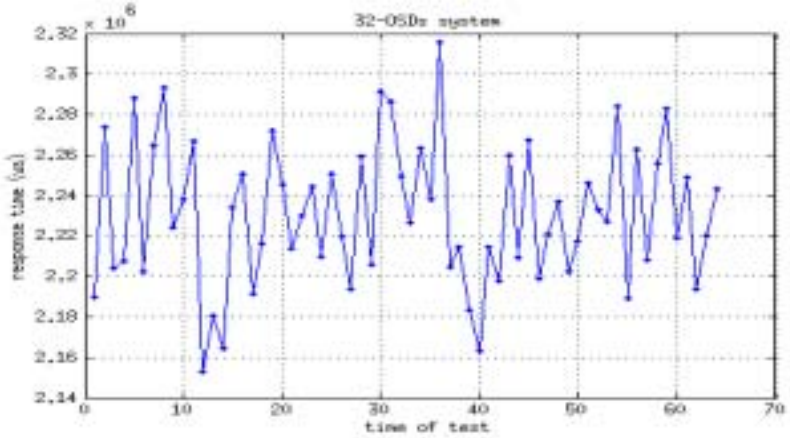
所有的实验程序都运行在一台 2.4 GHZ Intel Celeron CPU , 512 MB RAM 的 PC 机上，操作系统是 Red Hat Linux 9.0，内核版本 2.4.20。使用 Matlab 作为仿真软件。Matlab 首先依指数分布生成一个表示文件大小的随机数组。然后将柔性对象分布算法将应用于这些文件，计算出相应的传输时延。算法以 Matlab 的 M 文件的形式实现。假设 Overhead 开销因子为 80 微秒，网络带宽为 1000 Mb/s。且经过排序后选择出设备应该能比随机选择的设备更快。如果一个文件被哈希，其传输总时延为 80+文件大小*1024*8/1000μs。如果一个文件被映射到 n 个设备上，可假设其连接是近似串行建立的，则建立连接的时间为 n*80μs，其传输总时延为 n*80+文件大小*1024*8/(1000*n) μs。根据 OSD 数目的不同，将仿真的情况分为下列三类：16 个 OSD 的系统，32 个 OSD 的系统和 64 个 OSD 的系统。

4.3.2 实验结果

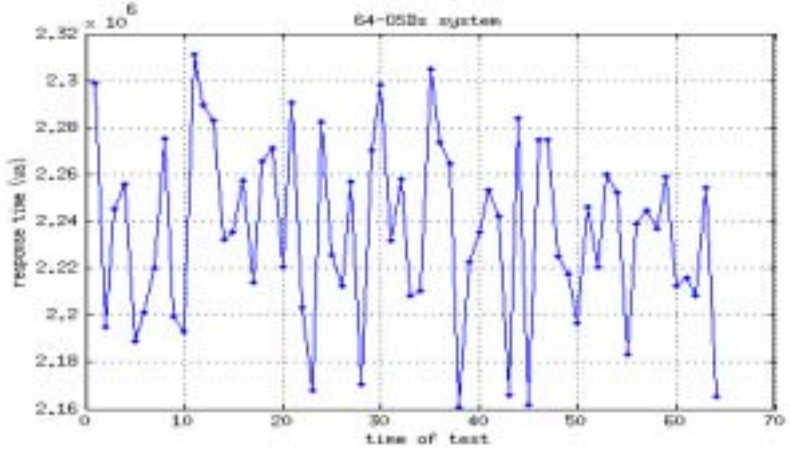
图 4.5 和 4.6 描述了 16 OSD 系统，32 OSD 系统和 64 OSD 系统的仿真结果。



(a)

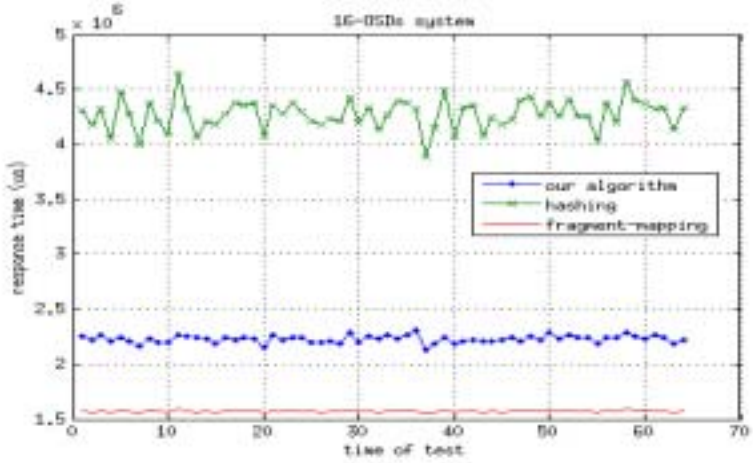


(b)

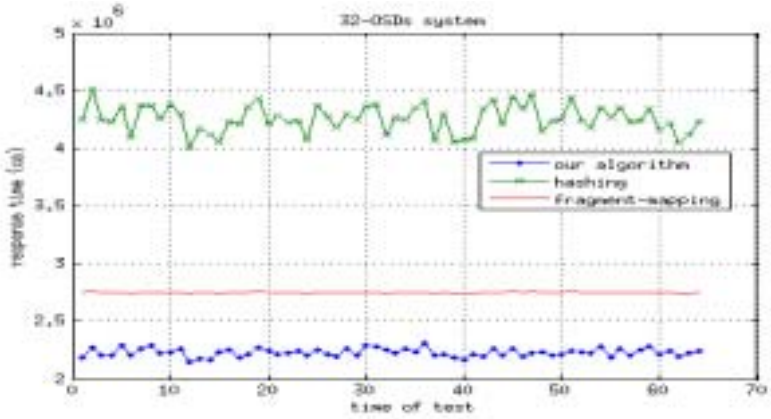


(c)

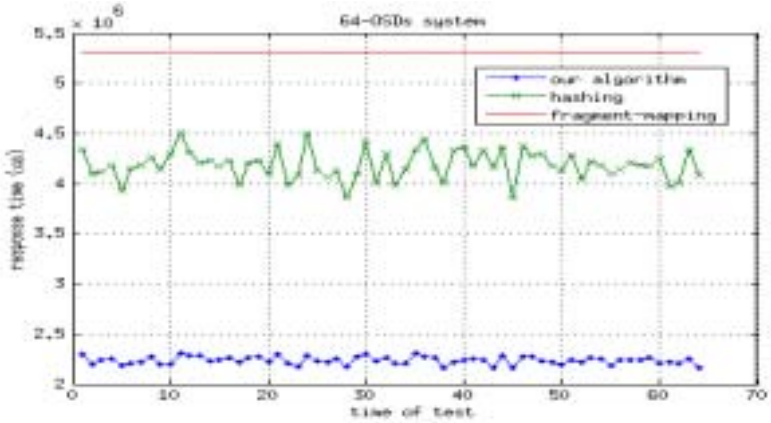
图 4.5　柔性对象分布算法的传输总时延 16-OSDs (a), 32-OSDs (b) 和 64 OSDs (c)



(a)



(b)



(c)

图 4.6　三种算法在不同规模系统 16-OSDs (a),32-OSDs (b) 和 64-OSDs (c)传输总时延

在每种虚拟系统中，都分别计算和比较了哈希算法，分片算法和柔性对象分布策略所需要的传输总时延。

表 4.1　对象分布算法的各项统计数据

设备数	均值（μs)	标准差（μs)	最大值（μs)	最小值（μs)
16	2.229474e+06	3.303443e+04	2.309076e+06	2.132241e+06
32	2.230677e+06	3.514477e+04	2.315964e+06	2.152755e+06
64	2.235837e+06	3.870051e+04	2.311283e+06	2.160869e+06

表 4.2　哈希算法的各项统计数据

设备数	均值（μs)	标准差（μs)	最大值（μs)	最小值（μs)
16	4.274640e+06	1.382437e+05	4.644530e+06	3.883163e+06
32	4.255894e+06	1.180060e+05	4.511304e+06	4.010331e+06
64	4.187441e+06	1.448126e+05	4.512065e+06	3.857762e+06

表 4.3　分片算法的各项统计数据

设备数	均值（μs)	标准差（μs)	最大值（μs)	最小值（μs)
16	1.572765e+06	8.640234e+03	1.595883e+06	1.548298e+06
32	2.751877e+06	3.687687e+03	2.759858e+06	2.744203e+06
64	5.308309e+06	2.262698e+03	5.313381e+06	5.303158e+06

4.3.3　实验结果的分析和讨论

在仿真的过程中确保了 Matlab 生成的文件大小符合 LLNL [26]所描述的负载分布。

表 4.1，4.2 和 4.3 列出了不同对象存储系统中三种算法对应的平均值，标准差和最大最小值。在 16 个 OSD 的系统中，分片算法的总时延均值是三种算法中最小的。但是当设备的数目增大时，由于发送和接收端开销增大，分片算法的性能会逐渐下降。在 32 个 OSD 的系统中分片算法的总时延均值已经超过柔性对象分布策略。而在 64 个 OSD 的系统中分片算法的总时延均值已经是三种算法中最大的了。哈希算法的总时延在 4 到 4.6 秒之间，而柔性对象分布算法则在 2 到 2.3 秒之间。它们在三个虚拟系统中的变化并不大。柔性对象分布算法之所以比哈希算法快是因为充分利用了设备间的并行性。分片算法总时延的标准差随着设备数的增加而减小。哈希算法和柔性对象分布算法总时延的标准差则基本稳定不变。而且柔性对象分布算法总时延的标准差比哈希和分片算法的都要小。从总体上看，柔性对象分布算法比其他两种算法更加稳定。并且在对象存储系统拥有大量 OSD 时表现最好。

4.4　本章小结

本章提出了一种在分布式对象存储系统中的柔性对象分布策略；它结合了哈希和分片算法两者的优点，同时又避免了它们的缺点。在柔性对象分布算法中，通过设置合理的边界值 512KB 来区分大小文件，从而将哈希和分片算法融合起来，保持了分片算法良好的可扩展性和哈希算法的高效性。讨论了算法中的两个关键因素，文件映射成为对象的最佳数目和在多大范围内选择设备进行并行传输。仿真结果证实三种算法中的柔性对象分布算法在规模扩展变化的系统中开销最小，并且其性能受设备数增加的影响较小。

5 系统测试与性能分析

5.1 测试环境

采用 IOzone 作为系统测试的工具。IOzone 是一种跨平台文件系统基准测评工具，也经常用于存储系统的测试，生成并测评各种文件操作。包括：写（write），重写（re-write），读（read），重读（re-read）等。IOzone 将评测结果保存在电子表格文件中。

系统所在环境为千兆以太网，采用的软硬件配置如表 5.1 所示。

表 5.1 测试时文件服务器和磁盘阵列的配置

	主板	CPU	内存	硬盘	操作系统	内核
元数据服务器	Intel E7520	P4 3.0G	512M	SATA Seagate 200G	RedHat AS3.1	2.4.21
设备端	Intel E7520	P4 3.0G	512M	SATA Seagate 200G	RedHat AS3.1	2.4.21
客户端	Intel E7520	P4 3.0G	512M	SATA Seagate 200G	RedHat AS3.1	2.4.21

在测试时，采用类似./iozone -i 0 -i 1 -Rab t110.xls -f /mnt/cfs/t110 -g 1G -C 的命令，生成一次最大文件为 1G 的测试。测试对象系统对并行性的利用、元数据服务器在对象系统中的功能，并验证对象分布算法的有效性。 分别进行多客户端单设备端，多客户端多设备端和单客户端多设备端测试。

5.2 测试结果及分析

多客户端单设备端，多客户端多设备端和单客户端多设备端测试结果如图 5.1 至 5.3 所示，对应详细数据在表 5.2 至 5.4 中。

表 5.2 多客户端单设备端测试数据表

客户端数	Write (KB/s)	Re-write (KB/s)	Read (KB/s)	Re-read (KB/s)
2	36856	16501	23917	26940
4	39105	19405	19227	20473
6	36432	16516	17384	17948

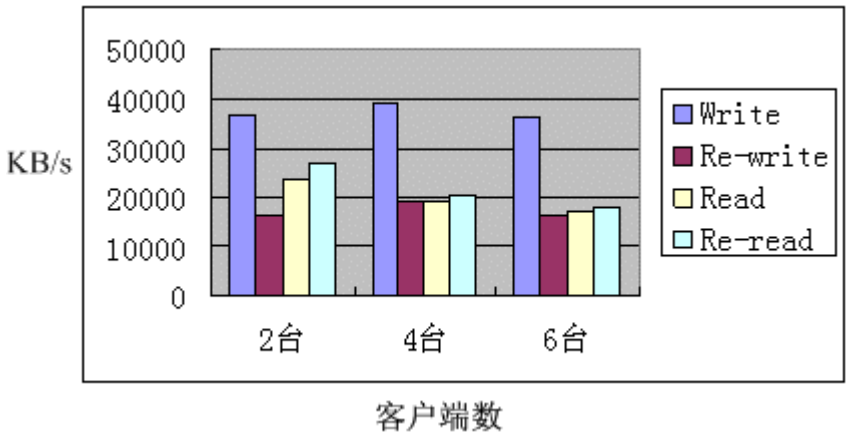


图 5.1　多客户端单设备端测试数据图

表 5.3　多客户端多设备端测试数据表

设备数	客户端数	Write (KB/s)	Re-write (KB/s)	Read (KB/s)	Re-read (KB/s)
2	2	62143	48398	54781	53441
2	4	64384	54469	41700	41970
4	2	48507	46106	122419	141606
4	3	59290	50013	68807	79081
5	2	63496	65805	173649	172386

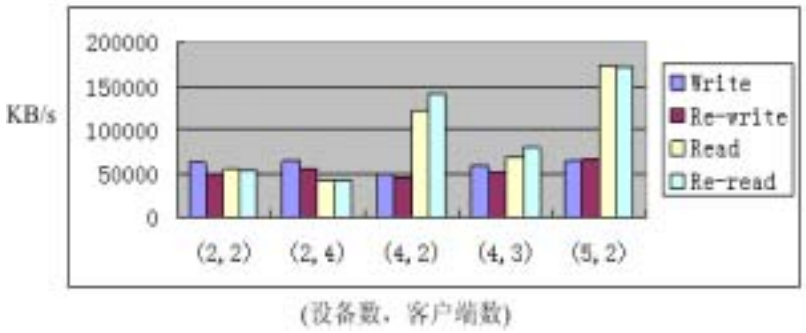


图 5.2　多客户端多设备端测试数据图

表 5.4　单客户端多设备端测试数据表

设备数	Write (KB/s)	Re-write (KB/s)	Read (KB/s)	Re-read (KB/s)
1	32745	15437	39950	42587
2	33084	23893	56614	64871
4	40876	40282	83542	83321
6	49604	48206	84137	84226

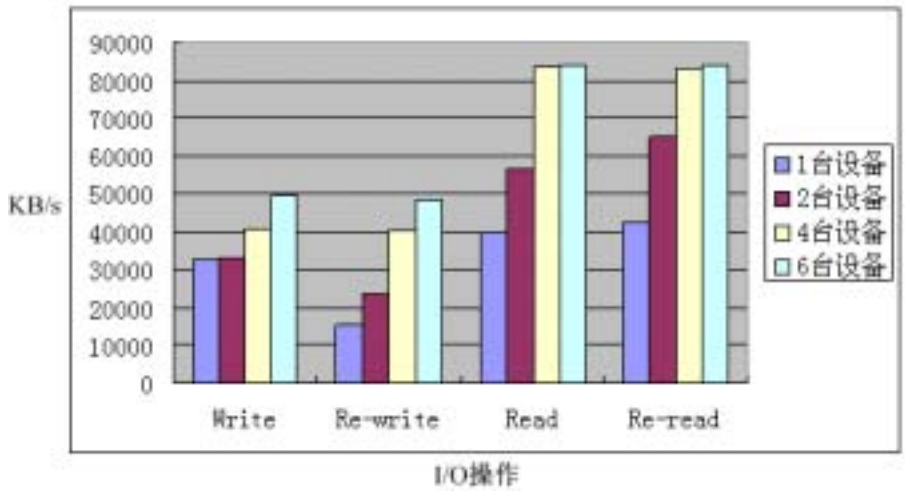


图 5.3　单客户端多设备端测试数据图

表 5.2 和图 5.1 是多客户端单设备端测试数据。在系统中只有一台设备的情况下，随着客户端的增多，系统性能不稳定并有下降的趋势。由于设备只有一个，所以元数据服务器的对象分布算法无法利用系统的并行性。而从表 5.3 和图 5.2 的多客户端多设备端测试数据可以明显看出，当设备端数目增加时候，性能能够得到提升，即使此时客户端数目也增加。这说明对象分布策略行之有效，利用了系统的并行性，将多客户端并发访问的负载合理的分配到各个设备当中。从表 5.4 和图 5.3 的单客户端多设备端测试也可以看出，当对象存储系统中设备数为多台时，读写性能比单台设备有明显提高。另外通过分析测试结果，六台设备系统和四台设备系统间磁盘 I/O 性能的提高并不明显，多设备系统中读操作被缓冲区优化的效果虽然比较明显，但是随系统内设备数增加而继续改善的速度却较缓慢。

5.3　本章小结

本章通过使用 IOzone 对面向对象的存储系统进行性能测试，结果表明对象存储系统的传输性能十分优异，并且随着系统内设备数的增加性能得到继续改善。其原因是因为采用的对象接口，使得设备具有智能性，减轻了元数据服务器的负担，使得更多的数据能够在设备间并行传输。另外对象系统的结构设计较为合理，元数据服务器对设备的管理行之有效，使得设备的扩充对性能的提高有较大帮助。测试结果还说明了对象存储系统设计的缓冲架构在数据传输中确实起到了作用，使得系统性能的到进一步提高。

6 全文总结

随着系统结构技术的发展和用户对数据访问的需求，目前的存储结构成为性能访问的瓶颈。磁存储技术的发展使得存储密度每年增 100%的同时,访问延迟的增长率仅为 10%，导致目前的基于块方式的访问接口无法改变 I/O 的访问性能相对落后于 CPU 和内存访问速度的状况。对象存储的概念应运而生。而对象存储系统中元数据管理成为了新的挑战和研究课题。

本论文在总结和分析此前相关理论与实践的基础上，针对对象存储的特点，构思并设计实现了基于对象存储的元数据服务系统。所做的主要工作和从中得出的结论归纳如下：

1. 设计实现了对象存储系统中的元数据服务器。完成了文件到对象的映射，根据 OSD 的负载情况、空闲空间的大小、文件本身的特点和 QoS 的要求等因素决定对象 ID 的分配。系统向用户呈现传统的 UNIX 目录结构视图，并提供 POSIX 标准文件访问接口。完成用户和设备等资源管理功能，区分不同权限用户，对系统内所有设备进行全局调度。在安全方面实现了一套以 MD5 为基础的加密认证机制。

2. 实现了面向对象的元数据管理和组织。通过将元数据存放到后台 LDAP 数据库中，利用 LDAP 提供的 API 库，方便高效地对元数据进行存储，访问和管理。在效率、可扩展性和安全性都达到了较好的效果。

3. 构思设计了对象管理核心算法。包括对象分布算法，对象动态长度管理算法等。特别是对象分布算法，结合了传统的哈希算法和分片算法的优点，针对对象系统的特点进行优化，使系统整体性能的到较大提升。

4. 在系统中构建了合理有效的缓冲机制。针对元数据访问的特点，将最近访问过的元数据缓存到缓冲区中，以便下次访问命中时直接从缓冲区中取走数据，不必再产生磁盘 I/O，使系统性能得到进一步优化。

致　　谢

首先我要感谢我的导师王芳副教授。感谢她在这几年中对我的悉心指导和无倦的教诲。从课程学习、论文选题、课题研究、论文审阅以及最后的定稿无不倾注了导师的心血。导师以其深厚的学术造诣、严谨的治学精神和一丝不苟的工作作风，站在学科前沿，对我的课题和论文进行前瞻性指导；以其正直无私和平易近人的高尚品德为我树立了做人的榜样。导师渊博的知识、严谨的治学态度、高瞻远瞩的学术眼光、勇于创新的学术思想、丰富的科研经验以及勤奋的工作态度都将是我今后学习和工作的楷模。

感谢冯丹教授的大力帮助，她在课题的研究过程中提出了许多宝贵的建设性意见，及时提供本领域最新的研究进展状况，关心课题的研究发展。她对学术的执着追求和出色表现将永远鼓舞激励我不断求索。

能在一个团结、民主、充满学术气氛的实验室从事研究工作，是一件庆幸的事。对实验室的庞丽萍老师、刘景宁老师、周可老师、施展老师、许蔚老师、谭志鹏老师、李春花老师、刘群博士、曾令仿博士、覃灵军博士在课题的研究过程中提供的帮助，表示谢意。在同他们的讨论和交流中，我得到了很多启迪，受益非浅。特别是刘群博士、覃灵军博士，此课题的顺利完成得益于他们的协力相助。

在这里还要特别感谢曾令仿博士在论文方面对我的指导，使我的文章能够顺利发表，论文能够顺利完成。

在课题的研究过程中，与吕松、赵水清、刘汉波、陈亮、潘磊颖、彭万丽等同学的合作使我能顺利地完成工作。另外我还得到了系统组的张羚、张葱仔、龚玮和史伟等同学的大力帮助，在此表示感谢。

最后，我要深深地感谢父亲、母亲对我的关怀。父母为了让我安心完成学业任劳任怨，他们的爱是我前进的动力。感谢我的女朋友程莹在学习、生活上的理解、关心和支持。谨以此文献给我的家人和女朋友，祝他们身体健康，工作顺利。

参考文献

[1] 张江陵，冯丹．海量信息存储．北京：科学出版社，2003．1~8

[2] Garth A.Gilbson, Rodney V.Meter. Network Attached Storage Architecture. Communications of the ACM, 2000，43(11): 37～45

[3] Marc Farley. SAN 存储区域网络.(第 2 版). 孙功星等译. 北京：机械工业出版社，2001.179～182

[4] Seigle.M. Storage area networks in video applications. SMPTE Journal, April, 2001, 110(4)：236～239

[5] Chang-Soo Kim, Gyoung-Bae Kim and Bum-Joo Shin. Volume management in SAN environment. CPADS 2001. 500～505

[6] G.A.Gibson and J.Wilkes.Strategic. Directions in Computing Research Working Group on Storage I/O Issue in Large-Scale Computing, Self_mamanging Network-attached Storage.ACM Computing, 1996, 28(4)：30～43

[7] 李长和，施亮，吴智铭．基于 Linux 的 NAS 系统的设计与研究．计算机工程，2002，28(7)：198～199

[8] Molero, X., Silla, F., Santonja, V. and Duato, J. A tool for the design and evaluation of fibre channel storage area networks. Simulation Symposium, 2001. 133～140

[9] Jander, Mary. Lauching a storage-area net. Data Communications, 1998, 27(4): 64～72

[10] P Braam. Lustre: A High Performance Distributed Linux File System. ACM Transactions on Computer Systems, 2003

[11] K.Sakar. An Analysis of Object Storage Architecture. IEEE Computer, 2003, 2(3): 23~26

[12] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit groupings: Exploiting disk bandwidth for small files. In Proceedings of the 1997 USENIX Annual Technical Conference. USENIX Association, Jan. 1997, 28(4): 30～43

[13] R. O. Weber. Information technology—SCSI object-based storage device commands (OSD). Technical Council Proposal Document T10/1355-D, Technical Committee T10, Aug. 2003

[14] Mike Mesnier, Gregory R Ganger ,Erik Riedel, Object –Based Storage. IEEE Communction, 2003, 15(4):84~90

[15] Garth A. Gibson, David F. Nagle, etc. NASD Scalable Storage Systems, Proceedings of USENIX 1999, Linux Workshop, Monterey, CA, June 1999

[16] Christos Karamanolis, Mallik Mahalingam, Dan Muntz, Zheng Zhang. DiFFS: a Scalable Distributed File System, Hewlett-Packard Company 2001. 85~91

[17] Hyeran Lim, Vikram Kapoor, Chirag Wighe and David H.-C. Du，Active Disk File System : A Distributed, Scalable File System, Proceedings of the 18 th IEEE Symposium on Mass Storage Systems and Technologies, San Diego, April 2001. 101-115

[18] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, B. Hillsberg, IBM Storage Tank—A heterogeneous scalable SAN file system, IBM SYSTEMS JOURNAL, 2003, 42(2):11~20

[19] Matt DeBergalis, Peter Corbett, etc. The Direct Access File System, Proceedings of FAST ’03, 2nd USENIX Conference on File and Storage Technologies, 2000

[20] R. Burns. Data Management in a Distributed File System for Storage Area Networks. PhD Thesis. Department of Computer Science, University of California, Santa Cruz, March 2000

[21] J.R. Moase. Panasas Storage Cluster & Object Storage Overview. www.panasas.com, October 2004

[22] Peter J. Braam. The Lustre Storage Architecture. Cluster File Systems, Inc. http://www.clusterfs.com, August 2003

[23] M. Spasojevic and M. Satyanarayanan. An Empirical Study of a Wide-Area Distributed File System. ACM Transactions on Computer Systems May 1996. 15~19

[24] 吴思宁，贺劲，熊劲，孟丹, DCFS 机群文件系统服务器组的设计与实现, 2002 全国开放式分布与并行计算学术会（DPCS2002），2002.

[25] Feng Wang, Scott A. Brandt, and Ethan L. Miller, Darrell D. E. Long. OBFS: A File System for Object-based Storage Devices. In 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST2004), College Park, MD, April 2004

[26] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In Proceedings of the 2000 USENIX Annual Technical Conference, June 2000.

[27] Lan Xue, Yong Liu. MDS Functionality Analysis. December 4, 2001. 12~13

[28] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. Andrew: A distributed personal computing environment. Communications of the ACM, Mar. 1986, 29(3):184~201

[29] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. ACM Transactions on Computer Systems, Aug. 1984, 2(3):181~197

[30] Thomas M. Ruwart. OSD: A Tutorial on Object Storage Devices. 19th IEEE Symposium on Mass Storage Systems and Technologies, University of Maryland, Maryland, USA, April 2002, 15(3): 145~151

[31] Yeong. W, Howes. T, Kille. S. Lightweight Directory Access Protocol. RFC 1778, 1995

[32] Howard, J.H. An Overview of the Andrew File System, Proceedings of the USENIX Winter Technical Conference Feb. 1988. 101~102

[33] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The Global File System. In Proceedings of the 5th NASA Goddard Conference on Mass Storage Systems and Technologies, College Park, MD, 1996. 188~203

[34] M. Satyanarayanan, J. H. Howard, D. N. Nichols, R. N. Sidebotham, A. Z. Spector and M. J. West. The ITC Distributed File System: Principles and Design, Proceedings of the loth Symposium on Operating System Principles (SOSP), Orcas Island, Washington, U.S., ACM Press, December 1985. 471~482

[35] 史小冬，孟丹，祝明发，COSMOS:一种可扩展单一映象机群文件系统，南京大学学报（自然科学）, 2001, 29(4): 259~265

[36] Braam, P. J. The Coda Distributed File System, Linux Journal, June 1998, 6(2): 46~51

[37] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File-System for Large Computing Clusters. In Proceedings of the 2002 Conference on File and Storage Technologies (FAST), USENIX, Jan. 2002. 115~116

[38] 王芳，张顺达. 对象存储系统中的柔性对象分布策略. 华中科技大学学报（自然科学版），已录用

[39] Fang Wang, Shunda Zhang, Dan Feng, Hong Jiang, Lingfang Zeng, Song Lv, A Hybrid Scheme for Object Allocation in a Distributed Object-Storage System, Accepted by the International Conference on Computational Science 2006 (ICCS 2006), UK, May, 2006

[40] Lan Xue, Yong Liu. MDS Functionality Analysis. December 4, 2001. 12~13

[41] Randolph Y. Wang and Thomas E. Anderson. xFS: A Wide Area Mass Storage

File System. In Proceedings of the Fourth Workshop on Workstation Operation Systems, October 1993

[42] John L. Hennessy, David A. Patterson. Computer Architecture A Quantitative Approach (Third Edition), 2003. 148~161

附录　（攻读学位期间发表论文目录）

[1] 王芳，张顺达. 对象存储系统中的柔性对象分布策略. 华中科技大学学报（自然科学版），已录用. (署名单位：华中科技大学计算机学院)

[2] Fang Wang, Shunda Zhang, Dan Feng, Hong Jiang, Lingfang Zeng, Song Lv, A Hybrid Scheme for Object Allocation in a Distributed Object-Storage System, Accepted by the International Conference on Computational Science 2006 (ICCS 2006), UK, May 28-31, 2006. LNCS. (署名单位：华中科技大学计算机学院)

对象存储系统的元数据管理

作者： 张顺达
学位授予单位： 华中科技大学

相似文献(10条)

1.学位论文 曾令仿 基于对象的网络存储智能处理方法研究 2006

近年来，随着存储需求的增加及存储应用日益复杂，人们需要存储系统有较高的智能性：自我管理/自我维护/甚至自我恢复等，来实现数据共享/无缝扩展/实时备份/容错/系统监控/流量控制/远程数据备份等功能。

以构造高性能智能化的存储系统为目的，对存储系统关键技术和实现方法进行了深入研究。主要内容包括：可扩展的对象存储系统结构；存储主动服务方法；策略触发机制；自适应的多并行度I、O 流水调度；层次存储智能化；基于线性时序逻辑的对象文件系统的形式化描述方法。

在继承前人工作的基础上，经过理论研究与实际开发取得了如下一些成果：

（1）实现一种可扩展的基于对象的智能存储系统，系统中的对象存储控制器OSC（Object-based Storage Controller）扩展ANSI T10 技术委员会关于对象的定义，对象除了包含属性/数据外，还扩展了包含触发器（Trigger）。对象的属性包含一定的语义，便于描述存储系统和提供动态的行为数据。

（2）研究用户数据在对象存储系统及存储设备中的分布策略与组织形式/对象负载平衡机制/对象存储系统的策略触发机制。对象的分布策略直接影响存储系统负载，提出一种混合放置（Hybrid-Layout）算法，实现静态负载平衡策略，将小文件直接映射成一个对象并通过哈希映射到一个设备中去，而对大文件则分割成多个对象，分别放置在不同的设备里。对象存储系统采用域对等（Region Peer to RegionPeer）的方法组织和管理。

（3）应用心理学等理论成果（统计模型/跟踪和预测等），提出一种基于反馈控制的预测性的存储主动服务模型，并讨论了存储主动服务系统架构。通过完善和扩展对象的定义，基于对象属性预测数据访问的变化，评估其对将来性能的影响，为“存储主动服务”提供决策依据。提出策略触发机制，并结合域对等，应用心理学模型实现动态负载平衡策略。仿真结果表明“存储主动服务”的方法是可行的。

（4）现有的网络存储，其存储协议/网络协议的复杂导致管理困难，同时I、O路径的不合理设计，增加了额外开销，降低了存储效率。分析了网络存储系统I、O路径上的软硬件开销，并应用于指导I、O 流水线设计。I、O 请求以并行度描述通过同一流水段的多个子工作量（或子任务）。提出自适应的多并行度I、O 流水调度方法，能重叠对象存储系统的I、O 处理各功能段的操作，提高对象存储系统I、O 性能。

（5）基于对象的存储，赋予备份归档新思路：将一组相关文件抽象为存储对象，方便层次存储管理自动化，使备份/归档/检索和恢复等操作变得简单而高效。

一方面，提出基于对象摘要的备份和归档系统；另一方面，设计以磁盘阵列与磁带库混合组成的/可提供海量存储空间和高性能的磁带虚拟化系统，该系统采用基于信息生命周期管理的数据迁移算法，实验测试表明，系统对用户透明，并能显著提高数据备份与恢复速度。提高了总拥有性价比。

（6）对象文件系统时序逻辑（OFSTL）是线性时序逻辑在描述文件系统应用中的一个推广。用OFSTL 描述对象文件系统的性质，用模型化的状态迁移系统表示对象文件系统的访问行为。并借助该逻辑框架研究对象文件系统访问语义和访问行为。应用该逻辑框架去形式化描述对象文件系统性质，并结合模型检测的方法验证对象文件系统设计的正确性。

2.期刊论文 宋健.刘川意.鞠大鹏.汪东升.SONG Jian.LIU Chuan-yi.JU Da-peng.WANG Dong-sheng 基于对象存储系统中多维服务质量保证的设计与实现 -计算机工程与设计2008,29(3)

存储系统的服务质量(QoS)保证对于满足上层应用的需求至关重要.基于对象存储(OBS)使用支持丰富语义的对象级接口,能够更好地实现QoS保证.在研究了基于对象存储系统中QoS交互机制的基础上,提出了一个基于对象存储设备(OSD)的多维QoS框架,设计并实现了一个应用于OSD的多维QoS算法.实验显示,该框架和算法能有效满足多个客户端的不同维度QoS要求.

3.学位论文 张伟 对象存储系统中数据通道的设计与实现 2008

随着数字信息的爆炸式增长和应用需求的不断提高，传统的网络存储系统在容量、性能、可扩展性、安全性、服务质量等方面面临着巨大挑战，对象存储技术采用全新的对象接口，被认为是下一代网络存储技术的标准。设计高效的对象存储系统，使其能充分发挥对象存储技术的优势，以满足日益增长的海量数据存储需求已成为新的研究热点。

本文对对象存储系统中数据通道的实现进行了研究。对象存储系统中客户端文件系统和对象存储设备之间的通信所经由的数据通道是影响整个系统性能的关键之一，其包括二者之间通信所采用的传输协议及其内部采用的缓冲策略。通过研究原有对象系统HOSS关键部分的运行机制，采用更细粒度的缓冲策略和开源的UNH iSCSI协议传输模块对HOSS系统中的数据通道及缓冲层进行重新设计与实现。通过与对象存储系统HOSS改进前后的读/写性能对比的结果表明，改进后的系统写性能在文件大小为512K时提高最大，提高了1.65倍；读性能在文件大小为256M时提高最大，提高了85%。通过测试还分析了包长度和OSD数量对系统读写性能的影响。文章同时提出了相关算法和思想用于解决HOSS系统存在的缓存一致性问题。

4.期刊论文 陈国志.王志华.周敬利.Chen Guozhi.Wang Zhihua.Zhou Jingli iSCSI对象存储系统的研究 -计算机工程与应用2005,41(5)

对象存储系统是一种新的网络存储体系结构,它结合了SAN的高性能和NAS的跨平台的优点.以iSCSI协议构建的IP SAN作为对象存储系统的传输网络,可以开发出实用的iSCSI对象存储系统.实验证明,该系统有较高的存储性能.

5.学位论文 龚玮 对象存储文件系统的设计与实现 2006

对象存储文件系统作为对象存储系统中的一个重要组成部分，已经成为分布式文件系统领域的研究热点。对象存储文件系统在可扩展性、安全性、性能上的诸多优势使它成为高性能计算、生命科学、能源等行业的首选。

设计并实现了一种用于对象存储系统的文件系统HUSTOBS。与其他对象存储文件系统一样，HUSTOBS文件系统由对象存储设备端、元数据服务器端和客户端三大部分组成。它们通过千兆以太网互连，三方共同协作完成一次数据操作。与传统的分布式文件系统所不同的是，HUSTOBS以对象作为基本传输单元。对象是可变长的，它继承了数据块和文件在性能和易跨平台等方面的优势。对象都具有属性，用于反映对象的某些特征。以对象作为基本传输单元使文件系统在可扩展性、安全性、易管理性和性能上有很大的提升空间。HUSTOBS完全按照对象存储命令集的协议要求设计完成，其设备端由专用的嵌入式系统充当；其客户端提供Windows和Linux环境下两种客户端，每种客户端都提供API和虚拟逻辑盘（Windows环境下）或虚拟目录（Linux环境下）两种访问方式，方便不同的用户使用。

作为分布式网络存储文件系统，针对以对象作为基本传输单元，对象存储文件系统在数据传输方面有很大的优化空间。对象存储设备通过为每个对象自定义“预取”属性页，记录用户在一段时间内的访问兴趣并自动更新，实现一种自适应的动态预取策略，提高了预取的命中率和整体性能。通过对HUSTOBS进行测试并对测试结果进行分析，针对对象存储文件系统的特点，HUSTOBS使用缓存、聚合写和预取读等方法进行了优化，取得了很好的效果。

6.学位论文 史伟 对象存储原型系统设计及相关实现 2006

数字以强大的信息表达能力以及单一的处理、传输和存储方式，融合了整个信息技术。半个世纪以来，作为数字信息载体的存储技术得到飞速发展。不断增长的存储需求和管理成本催生了基于对象的存储技术，而“对象”也有望成为下一代存储技术的标准接口。

在分析当前流行的网络存储体系结构及存储协议的基础上，对基于对象存储技术以及T10的SCSI OSD（Object-Based Storage Device Commands）标准作了深入研究。“对象”是传统块接口和文件接口的折中，基于对象存储系统在I/O性能、跨平台、可扩展性以及安全性等方面都表现不错。

实现了一个符合T10 SCSI OSD标准的对象存储原型系统，包括对象存储设备和客户端。客户端的SCSI对象设备驱动是一个Linux SCSI上层驱动，基于Linux块设备子系统实现，用来管理所有检测到的OSD设备。iSCSI启动设备是Linux SCSI协议栈的底层驱动，为客户端提供通过IP网络访问iSCSI目标设备的iSCSI通路。iSCSI目标设备实现iSCSI传输协议的Target部分。对象存储服务模块负责管理物理存储介质和处理OSD命令，以上模块均在Linux内核空间实现。

测试并分析了基于对象存储原型系统的性能，得出的结论是：通过在对象存储原型系统引入聚合读/写机制可以大大提升系统的I/O带宽。

7.期刊论文 谭支鹏.冯丹.TAN Zhi-Peng.FENG Dan 基于对象的存储系统对象迁移策略 -计算机科学2006,33(6)

对象存储系统是下一代网络存储重要组织模式,对象管理是对象存储系统的关键技术之一.本文对对象存储系统中的对象迁移策略进行了系统的研究,提出了可变阈值和阈长的动态反馈调整模型,以此来确定存储对象迁移的时机和目标对象存储设备的选取.另外,本文还利用Petri网工具对对象存储系统存储对象的迁移进行建模分析,给出了存储对象迁移控制的Petri网模型.但是随着存储系统存储节点的无限增加,Petri网模型将会无限复杂和庞大.为了减少Petri网模型控制模型的复杂度,又引入有色Petri网理论,实现了对存储对象迁移控制Petri网模型的简化.该模型同样很好地实现了对象存储系统中对象迁移控制的建模分析.这些研究对我们在建立对象存储系统时起到了很好的帮助作用.

8.学位论文 赵俊杰 面向对象存储系统安全模型的研究与实现 2006

应用系统对海量存储技术的要求越来越高，经典的基于块访问接口的SAN和基于文件访问接口的NAS系统已力不从心，而新出现的基于对象接口的网络存储技术以高性能、高可扩展性、易管理和跨平台共享等多种性能受到越来越多应用场合的欢迎。然而，当前应用对存储网络的安全性提出了更高要求，已有的基于对象接口的网络存储系统仅仅提供了对象属性访问控制的安全机制，未能从系统的角度考虑其整体的安全性。

本文在深入研究和分析了基于对象接口的网络存储技术之后，发现它面临的安全性问题及其它能够提供的安全机制，为此提出了一种新的面向对象存储系统安全模型，从系统的角度为网络存储系统的整体安全问题提供了一揽子解决方案，并在开放源码的对象存储文件系统Lustre上实现了该模型并对其相关性能进行了测试和比较。主要工作如下：

(1)通过网络存储技术产生背景、发展现状及安全需求等的分析和研究，指出了网络存储安全研究的必要性；分析了网络存储安全研究现状，指出了其面临的问题和需要进一步研究的内容。

(2)提出了面向对象存储系统安全模型，并设计了相关的安全协议。针对对象存储的体系结构特点和其所可能遭受的安全威胁，对基于该模型协议进行了安全性分析；在此基础上，对存储模型的安全性进行了定性分析；通过与相关工作的比较，说明了该模型具有较强的安全性。

(3)提出了适用于对象存储系统的轻量级多变密钥对象文件加密算法。该算法是异或运算的加解密，采用伪随机数的方法产生密钥序列，保证异或运算密钥的一次一密钥，从而

满足快速加解密和较高的安全性要求，非常适用于对象存储系统。

(4)利用开放源码的对象存储文件系统Lustre，在实验室环境中实现了基于对象接口的网络存储原型系统；在该原型系统上实现了本文提出的安全模型及相关协议，并对其相关性能进行了测试和比较分析。　　。

9.学位论文 覃灵军 基于对象的主动存储关键技术研究 2006

随着计算机和互联网的迅猛发展，网络存储应用现了一些新特点：数据总量爆炸性增长的趋势越来越快，存储管理和维护的自动化和智能化程度要求越来越高，多平台的互操作性和数据共享能力越来越重要。网络存储正发生着革命性的变化，基于对象的存储应运而生。基于对象的存储将存储接口作了根本性的改变，提出了对象接口，由此克服了块接口与文件接口的缺陷，使得对象存储系统在安全性，数据共享，可扩展性及性能等方面能做到最好的折中。对象接口访问的基本单位是对象，对象除了包含用户数据外，还包含能描述对象特征的属性。通过在用户和设备之间传递对象属性信息，对象接口比其他接口具有更为丰富的语义表达能力。

随着电子技术的发展，存储设备上已嵌入了越来越多的处理能力，上层应用的部分功能可以迁往设备（即“主动存储”的概念）。目前广泛流行的存储设备是“哑设备”，只能被动地响应用户的请求，随着设备功能越来越复杂，传统的对设备透明的管理方式已很难胜任。人们迫切需要一种设备参与在内、更简单灵活的方式来管理，而主动存储能很好地满足这些应用的需求。另外，由于硬磁盘本身机械运动的本质特征没有明显改变，在网络存储条件下网络延迟不能忽略，特别是网络共享跨越了广域网，如何提高处理节点和存储节点间的数据传输速率成为当前提高系统整体性能的关键因素。主动存储系统能够在很大程度上解决该项难题。可以说主动存储系统代表着将计算向数据移近的发展方向。但在对象接口出现之前，原有设备使用的接口是一种简单透明的接口，这妨碍了设备端和用户端的任务合作。对象接口的出现加速了主动存储的发展进程，主动存储可以利用富于表达力的对象接口将更多的信息在设备端和用户端之间传递。可以说对象存储与主动存储的结合将是未来存储界领先的技术之一。

基于对象的主动存储系统（OBASS）结合了对象接口与主动存储的优点，主动存储将建立在对象基础上，所有信息都以对象的形式出现。OBASS的基础是基于对象的存储设备（OSD），在软件层次上，OSD主要分为三层：对象层、主动服务层以及存储质量控制层。对象层统一管理所有对象，负责对象的磁盘数据组织（包括磁盘内对象数据布局和磁盘间的对象放置）。主动服务层实现了主动存储功能，上层应用把功能模块下载到OSD中后作为特殊的对象（方法对象），由对象层存储与管理，由主动服务层调度方法对象执行。存储服务质量控制层对所有层次模块的执行过程施加影响，使OSD在不同的负载状况下能满足不同用户的对象读写QoS要求。

磁盘内的对象数据组织即对象文件系统，其中一个重要的特性是性能的持久性，即对象文件系统在长期的使用的过程中，经历了频繁的对象创建、删除和写操作后，其性能仍然能够维持在较高水平。性能持久性是通过柔性空闲空间管理和分配粒度可变的渐近空间分配策略实现的。

磁盘间的对象放置策略研究对象在多个磁盘间的放置，使得整个系统性能达到最优，负载得以均衡。不同的应用放置策略是不同的，对于流媒体应用，适用的放置策略是基于阻塞概率模型的放置策略；而对于事务处理，基于响应时间模型的放置策略更合适。对象放置策略应能运用于在线环境，但在线运行时负载特点是无法事先预料的，利用OSD的智能性及对象的属性，对象放置策略可对负载的特征进行自主学习，并根据学习的结果指导对象放置。

为实现基于对象的主动存储，首先对现有的T10 OSD标准进行修改，扩充对象概念引入方法对象使其支持主动存储。方法对象的执行有两种方式：一种是外界用户的请求触发执行，另一种是条件满足时的策略触发执行。主动服务层建立了统一的框架，把这两种方法的调度机制有效结合起来，支持计算任务与管理任务向OSD迁移。方法的执行机制是主动存储的核心，针对过滤型方法和服务型方法这两类不同方法，分别提出了在Linux系统下的实现机制。为评估主动存储的效能，将这两类方法分别运用于两种不同应用：OSDFS文件系统及对象存储系统的负载均衡。在OSDFS中，两种常规操作lookup和unlink作为过滤型方法下放到OSD中，分析表明，这可以大大减少网络延迟和内存拷贝。而负载均衡算法可以作为服务型方法运行在OSD中，实验表明，启用对象复制和对象迁移的负载均衡算法能最大程度地减少平均系统响应度。

存储服务质量控制分三个层次实现QoS：上层调度器实现对象请求调度，从对象级保证QoS；中层调度器实现对象预处理，与对象文件系统及缓存结合，对于对象读，通过对象的预取，对于对象写，通过页面预分配、延迟空间分配及延迟写保障对象读写的QoS；而下层调度器与Linux I/O子系统中的块I/O调度器结合，综合考虑带优先级的实时负载以及非实时负载的调度，即考虑I/O请求的时限，磁头定位时间和优先级三个因素在内的I/O调度。将OSD的QoS调度模块（OIS）与Linux系统中其他的调度器进行了比较，结果表明，对于实时读操作，OIS引起的延时抖动比其他调度器至少小1个数量级；而对于写操作，至少小2个数量级。

10.期刊论文 刘景宁.吕满.童薇.冯丹.LIU Jing-ning.LV Man.TONG Wei.FENG Dan 对象存储系统中对象查找及标识符分配管理策略 -小型微型计算机系统2009,30(9)

对象存储系统是一种新的网络存储体系结构,它结合了存储区域网SAN的高性能和网络附加存储NAS的跨平台的优点.对象存储系统的对象查找的快慢直接影响系统的性能.针对对象存储系统的对象查找是通过对象ID找到对象Onode.分析研究了对象文件系统中的对象查找策略及对象号分配策略,提出元数据服务器与对象设备协同管理对象ID,采用连续的对象号分配策略改进对象存储系统的对象查找方法,提高了对象存储系统的对象查找速度.