华中科技大学

博士学位论文

海量对象存储系统数据组织算法的研究

姓名：罗益辉

申请学位级别：博士

专业：计算机系统结构

指导教师：谢长生

20061101

摘 要

在网络存储系统中，数据的组织优化是提高存储系统性能的有效方法之一。合理的数据组织算法可提高存储系统的性能，使存储系统具有高可靠性、可用性和可扩展性。网络存储系统中的数据组织包括两个方面的内容：存储空间的数据组织和传输路径上的数据组织。存储空间的数据组织利用存储节点间的并行性，提高存储系统的 I/O 性能，通过不同存储节点的数据副本提高数据的可靠性和可用性；传输路径上的数据组织则是通过传输路径上的 Cache 合理配置和替换算法的选择，达到提高 I/O 性能的目的。本文根据对象存储系统的特点，研究数据组织的算法及相关问题，包括以下几方面的内容：

对象存储区别于其它网络存储系统之处在于它将文件系统的存储管理和用户管理分离，存储管理由存储节点来完成，而用户管理则通过元数据服务器来实现。这种功能的分离使得元数据服务器成为瘦服务器，便于存储系统的扩展。对象有丰富的语义，通过对象的语义，对象存储为存储系统提供安全保障，还可以为应用提供基于 QoS 的 I/O 服务。对象存储系统的体系结构使其有两种数据传输模式：NAS 模式和三方传送模式，传输模式结合 Cache 方案可达到提高存储系统性能的目的。

对象存储系统的存储空间的数据组织就是要实现数据对象在存储节点间按存储节点的容量分布，数据组织算法应尽可能减小算法的时间和空间开销，同时要保证算法适应存储系统的扩展，为此，设计了一种数据定位算法及扩展数据迁移算法。算法有以下特点：将存储节点分组，采用组间按容量分布和组内均匀分布的机制实现数据按容量分布；算法采用映射函数实现用户空间到存储空间的映射，使得算法的时间开销和空间开销最小；算法的适应性则通过记录系统的扩展信息来实现；相应的扩展迁移算法则保证系统扩展引起的数据迁移量最小。

应用要求存储系统提供基于 QoS 的 I/O 服务，而数据迁移对存储系统的服务质量有影响。通过对基于 QoS 的 I/O 服务模型的分析，定义迁移任务附加收益，并将迁移任务细分为迁移请求，在此基础上建立一个基于 QoS 的数据迁移模型，并设计

出相应的最大收益迁移调度算法。实验表明，该迁移调度算法对 I/O 服务质量的影响小于常用的迁移算法。

数据传输路径上的数据组织涉及到 Cache 替换算法，而 Cache 替换算法的好坏标准是存储系统的性能是否提高。通过对对象存储系统的加速比进行分析得出结论：Cache 对存储系统的性能改善不仅与 Cache 命中率有关，还与数据对象的设备访问时间有关。根据这一结论，设计出两种 Cache 替换算法：LAT 算法和 WLFRU 算法。LAT 算法选择 Cache 中设备访问时间短、命中率低的数据对象作为替换的对象；WLFRU 算法则通过对访问频率的加权达到同时考虑访问的局部性和数据对象的访问成本的目的。两种算法的性能都优于常用的 LRU 算法。

数据传输路径上的 Cache 设置必须结合数据的访问特点，而对象存储系统的三种实体的特点各不相同，所以，它们的 Cache 方案也各不相同。根据存储节点的读写速度慢的特点，将其 Cache 设置为预取 Buffer 和写 Buffer，前者利用预取缩短数据对象的读响应时间，后者通过延迟写，及时响应写请求。元数据服务器和客户端的 Cache 方案则和数据传送模式相联系，元数据服务器 Cache 在负载轻时缓存小数据对象，而客户端的 Cache 除了对两类数据对象都缓存外，还使用磁盘 Cache 积累采用三方传送的数据对象，达到减少网络通信量的目的。实验显示三类 Cache 的设置有助于存储系统的性能改善。

关键词：对象存储，I/O 性能，可扩展性，数据组织，数据迁移，Cache 方案，替换算法

Abstract

In the system of network storage, optimizing the data organization is one of the effective methods used to improve the system performance. A suitable algorithm to organize data can improve the performance of the storage system and provide with reliability, availability and scalability. The data organization of the network storage system includes two aspects: the data organization in the storage space and the data organization along the route to transfer data. The former utilizes the parallelism of storage nodes to improve the I/O performance of storage system, and promotes data's reliability and availability by using data copies at different storage nodes; the latter configures the suitable cache and designs replacement algorithms in order to improve the I/O performance. This paper studies the algorithms and their related aspects of the data organization according to the characteristics of the object storage system, which consists the following:

The difference between the object storage system and other network storage systems is that it makes the storage management and the user management of the file system separated. The storage management is realized at the storage nodes and the user management is implemented at the metadata server. The separating of the function makes the metadata server as a thin server, which is useful to extend the storage system. The object interface has a lot of semantics, which can provide security for storage system and QoS-based I/O services for applications. The object storage system has two modes to transfer data: the NAS mode and the third party transfer mode. Integrating the transfer mode and Cache scheme can improve the performance of the storage system.

Organizing data on the storage space of object storage system is to distribute data object among storage nodes according to the storage nodes' capacities, and the relevant algorithms must minimize the penalty of time and space; at the same time, the algorithms must also be adapted to the extending of the storage system. In order to realize those goals, one algorithm is designed to distribute data objects as users access data objects in the storage system and another algorithm is designed to migrate data objects as the storage system extends. Those algorithms have the following characteristics: the storage nodes are

divided into groups, and the data objects are distributed according to their group capacity among the storage nodes and are distributed uniformly in each group; the algorithm uses mapping function to map user space to storage space, which minimizes the overhead of time and space; the algorithm's adaptability is realized by recording the information about extending of the storage system; and the algorithm used for data migration minimizes the amount of migrated data.

The application requires the storage system providing QoS-based I/O services, however, the data migration may affect the QoS of storage system. After analyzing the model of QoS-based I/O services, we define an extra reward of migration task and divide the migration task into some small migration requests. On this basis, a data migration model based on QoS is established and a migration algorithm is designed to maximize the migration reward. The experiment indicated that this migration algorithm has less influence on I/O QoS than other frequently used migration algorithms.

Data organization at the data transfer path involves Cache replacement algorithm and the standard to evaluate Cache algorithm is whether the performance of the storage system has been improved. Some conclusions can be reached by analyzing the speedup ratio of the storage system: the factors to influence the performance of storage system include not only Cache hit ratio but also the time of data object to access storage devices. Based on these conclusions, we design two Cache replacement algorithms: LAT(Least Access Time) algorithm and WLFRU(Weighing Least Frequently Recently Used) algorithm. LAT algorithm selects the data objects with shorter time to access devices and less hit ratio as the replaced object; WLFRU algorithm weighs the access frequencies in order to take into account access localization and the penalty of data object to access devices. These two algorithms have better performance than the usual LRU algorithm.

Configuring Cache along the data transfer path must be related to the characteristics of data to access storage system, however, these three entities have different characteristics, and so their Cache schemes are different. Owing to the low speed to read/write the storage nodes, we configure pre-read Buffer and write Buffer: the former is used to shorten the response time of reading and the latter to respond fast by delaying writing onto disk. The schemes of Caches at metadata server and clients are related to the transfer mode. The Cache at metadata server is used to store some smaller data objects as there are lighter

loads; however, at clients, the memory Cache is used for data objects with two transfer modes and the disk Cache is only used to store those data objects using the third party transfer. All these are aimed to reduce network traffics. Experiment showed that these three Cache schemes can greatly improve the performance of the storage system.

Keywords: Object storage, I/O performance, Scalability, Data organization, Data migration, Cache scheme, Replacement algorithm

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到，本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并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。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保　密□，在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

1 存储系统概述

1.1 信息存储需求新特点

计算机技术的发展和普及，尤其是 Internet 的出现和普及，使得计算机无所不在，小到日用商品，大到工业生产，科学研究，离开了计算机无法进行。随着计算机应用的不断发展，计算机所处理的对象——信息，爆炸性增长。按摩尔定律，处理器的速度每 18 个月翻一番，处理器速度迅猛提高；网络的发展，特别是光纤的出现，使得信息传输速度快速提高；而存储设备由于机械速度的限制，其速度提高滞后于处理器和传输设备，因此信息存取成为计算机系统的主要瓶颈[1][2]，系统结构的研究重点也逐渐从处理、传输转向信息存储，信息存储在 IT 行业的地位越来越重要。

随着计算机应用的范围的扩展，信息量的不断增长，对信息存储提出新的要求，信息存储需求呈现出新的特点：

（1）高可靠性的信息存储。数字信息是信息的符号，其价值取决于信息的价值。随着计算机应用范围的扩展，越来越多中重要信息转变为数字信息，对于这些重要信息，存储的高可靠性至关重要。

（2）可扩展的信息存储。信息总量呈爆炸性的增长，这样对存储的需求也是无止境的。然而，考虑到技术和成本，企业的存储系统不可能一次到位，必须随着存储需求的增长而不断扩展。这就要求存储系统具高可扩展性，且其扩展在不中断存储服务的情况下进行，即进行动态可扩展。

（3）高性能的信息存储。早期计算机应用范围有限，仅用于科学计算，而 CPU 的运算速度有限，处理器成为计算机的性能瓶颈；随着处理器速度的不断提高，信息传输设备——网络成为计算机的性能瓶颈；随着网络技术的发展，特别是光纤的发展，网络速度得到很大的提高，这时计算机的性能瓶颈变成了信息存储设备。由于受机械部件的限制，磁盘数据访问时间平均每年只能提高 7~10%，数据传输率也只能以每年 20%的速度发展，而现代微处理器和内存系统正以平均每年增长 50~100%的速度发展，处理机和磁盘之间的性能差距已经越来越明显[3]。因此，提高存储性

能只能通过系统结构的方法，设计出高性能的存储系统成为系统结构的重要课题。

（4）高可用性的信息存储。在许多计算机应用中，特别是在电子商务和某些网络服务中，服务的停止意味着巨大的经济损失，这就要求计算机，当然也包括存储系统提供 24 小时×7 天的服务，所以信息存储的高可用性是信息存储研究的又一课题。

（5）自管理的信息存储。存储系统占据着整个企业业务应用系统的大部分成本。Gartner组织在它的研究报告中指出，在系统的硬件成本中，存储系统占据了大约40%～60%的比例；在系统总体成本中，存储系统占据了大约60%～80%的比例[4][5]。存储管理（包括容量部署、数据备份、负载平衡等）占据了系统管理中的大部分工作。2003年图灵奖的获得者Jim Gray曾经说过：存储系统中真正的成本是数据管理[6]。华尔街每年每TB的数据量需要花费30万美元的管理费用，并且每TB的数据量至少需要一个数据管理员。数据的备份/恢复、存档、重组织、动态扩展以及容量管理等等所带来的成本比系统的硬件成本要大得多。如果按照这样的比例计算，每PB的数据量每年就要花费3亿美元的管理费用，并且至少需要1000名数据管理员。所以，自管理的信息存储可以大大降低应用系统的成本。

（6）安全的信息存储。任何公司或机构，无论大小，都不敢置数据安全风险于不顾，毫无疑问，由安全性问题带给企业的损失可能会远远超出在安全解决方案上的投资。据统计，仅美国的公司在 2001 年便为消除计算机病毒造成的损害而花费了大约 123 亿美元；而且病毒的攻击可能会造成更高的财产损失。任何人都无法保证能免受攻击者恶意行为的伤害。 虽然许多国家和政府在法律上对恶意的攻击者定性为犯罪行为，并对抓获的攻击者进行控诉并判处相应的监禁，这种惩罚虽能起到一定的预防作用，可挽回损失而进行的工作努力通常是非常费时、昂贵，作用也非常有限，从而必须面对这样一个现实：很多种恶意攻击可对网络中的设备以及其上的数据造成伤害。如果没有对系统的安全性进行评估，也没有对重要资产进行保护，那么灾难的发生仅仅是一个时间性的问题。

总之，计算机应用要求存储系统容量更大、性能更高，信息保存更可靠、使用更方便，所有这些，推动存储系统的研究和开发不断发展。

1.2 信息存储的现状

信息存储需求推动存储系统的不断发展，存储系统的发展通过两方面实现的：存储设备的开发和存储系统结构的设计[7]。存储设备的开发通过元器件技术不断提高存储系统的性能、容量、可靠性等；系统结构则从整体出发，将多个存储设备组合成大容量、高性能、安全可靠的存储系统。

1.2.1 存储设备的现状

（1）磁盘设备

磁盘利用电磁特性将数据记录在磁片上，它是目前主流的存储设备，这主要取决于它的高性价比。常用的磁盘分两类：IDE 磁盘和 SCSI 磁盘。IDE 磁盘由于其价格便宜而在低端用户中广泛使用，SCSI 磁盘则主要用于基于光纤通道或者 SCSI-320 接口的高级服务器磁盘。最近十几年以来，磁盘容量基本上是按指数增加，这主要得益于磁盘磁密度的增加，磁盘转速的增加，以及新的技术的不断使用。

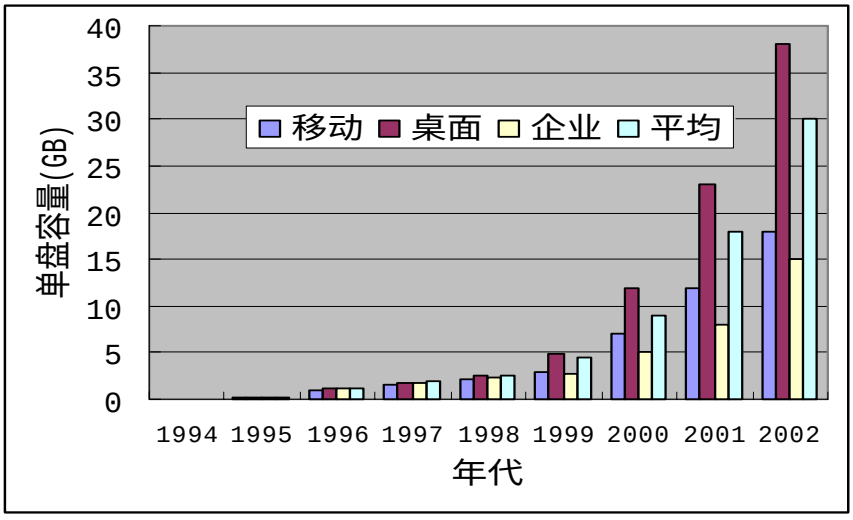


图 1.1 单盘容量的增长

目前几种公司主流的服务器磁盘性能差别不大。HITACHI 的商用硬盘 Ultrastar 系列支持 SCSI320 和 FC－AL 接口，转速可以达到 15000RPM，满道读时间为 6.7－8.9ms，平均寻道时间 4.7ms，平均延迟时间为 2ms，持续数传率 33.8MB/s 到 66.7MB/s，驱动器上的 cache 到达 8MB；而 Maxtor 公司性能最好的硬盘 atlas 转速也是 15000RPM，磁盘内部数传率最高 75MB/s，平均延迟时间 3.2ms，cache 大小也是 8MB。

Seagate公司的Cheetah 15K3的转速也是15000RPM，磁盘内部数传率最高为75MB/s。从磁盘技术的发展来看，磁盘的物理存储密度平均每年增加一倍如图1.1所示[8]，但是由于机械速度的限制，平均延迟时间和持续数传率提高的程度较小，性能的提高主要通过提高接口速度和驱动器cache大小来实现。磁盘容量性能提高的同时，磁盘其尺寸也在不断的减小，以适应消费电子类设备对大容量存储的需要。目前1英寸微磁盘设备的出现使得磁盘日益成为一种主要可移动、大容量、低价格的存储设备。

常用磁盘通过单磁头在磁介质表面旋转来读写数据，磁头的机械运动造成的延迟很难降低。一种磁盘革新技术是MEMS（Microelectromechanical systems）技术，它通过上千个固定读写磁头同时读写可移动的平面磁介质上的数据，达到并行读写的目标。该技术目前处于实验室阶段，随着它的产品化，磁盘性能将得到很大的提高[9]。

（2）光盘设备

光盘由于其价格便宜、便于携带而广泛使用，它是通过激光在盘片上刻痕来记录数据。与磁盘相比，光盘的容量相对较小，一般的光盘只能一次写，随着光盘技术的发展，可读写光盘等存储介质也被普遍使用[10]。随着超短波长的激光器的使用，如蓝色（亦称蓝紫色，波长405nm）激光器，光盘的容量将得到极大的提高。

（3）磁带设备

磁带因为其价格便宜、容量大和便携性而广泛用于数据备份系统。最常用的备份设备是磁带机或磁带库。磁带库是将多个磁带机、磁带槽及机械换带装置集成为大容量存储设备，其核心部件仍旧是磁带机，它一般需要专门的软件来管理。

目前，磁带技术与产品主要分DAT、DLT 和8mm三种。DAT和8mm均采用螺旋扫描技术，但磁带宽度不同，DAT磁带宽度约4mm。最新的DAT磁带采用了和DLT类似的高强金属带，可靠性增加。DAT技术多用于单个磁带机设备，它的数据读写速率低于最新的DLT和8mm产品。8mm是Exabyte公司的独立技术，为增加磁带强度以提高读写速度和可靠性，最新的AME磁带产品具有一定的竞争性，但由于技术不开放，使得产品的市场占有率较低。

IBM最近推出能够存储500GB未加工数据（未压缩数据）、吞吐量高达100MB/

秒的新企业磁带驱动器，并推出一种新的磁带虚拟环境，可在一个磁盘-磁带环境中支持多达 8192 个虚拟磁带盒。Exabyte 发布了 Magnum 224 LTO Ultrium 磁带库，具有备份功能并可随容量的增长而扩展，主要用于中小型企业的数据备份。各类存储系统对数据量和数据安全的要求区别很大，与之对应的价格及性能也有较大差异。随着技术的发展，新的技术将使磁带的磁密度、容量和数据传输率有很大增长。

（4）移动存储设备

在数字产品、手机等便携式电子产品和多媒体应用驱动下，移动存储市场蓬勃发展。移动存储的主要特点是便携性，同时兼有磁盘可重复读写的特点。移动存储除了容量、速度外，更注重设备的能耗，通常的移动存储卡的能耗是普通硬盘的 5%。在当今的移动存储设备的市场竞争中，低成本、低能耗安全性和高速度成为竞争焦点。

各种存储基本设备各自有自己的特点和优势，在存储市场中它们相互竞争，但不能相互取代，相反，各种存储设备可以实现优势互补，达到性能优化的目的。随着存储技术的发展和新的技术在存储中的应用，存储设备的速度和容量得到快速的提升，然而，单个存储设备的容量无法满足应用的需要，同时，由于机械速度的限制，其速度的提高远远低于处理器和传输设备的速度增长。而且存储的可靠性、可用性、安全性等的实现通过存储设备很难实现，所有这些，必须通过存储系统的体系结构的方案来实现。

1.2.2 存储系统的现状

最早的存储系统是 RAID，它通过多个廉价的存储设备按一定的规则组合在一起，达到提高性能、容量，实现安全可靠的目的。随着网络技术的发展，出现各种网络存储系统。网络存储系统有 NAS、SAN 和 iSCSI。网络存储的出现给存储的优化提供了更大的空间，也提出了新的挑战。

(1) RAID

RAID 就是将多个存储设备按照一定的形式和方案组织起来，如同使用一个硬盘一样,通过这种方式可获取比单个存储设备更高的速度、更好的稳定性、更大的存储能力。根据不同需要，RAID 有不同的级别，常用的级别有：RAID0、RAID1、RAID3、

RAID5，它们分别用来优化存储的某方面的性能。

RAID0 将多个相同磁盘组合成磁盘阵列，数据分条均匀分布到各磁盘，由于多个磁盘并发读写，所以其主要优点是提高 I/O 速度，而阵列的总容量为各磁盘之和，因此在提高 I/O 速度的同时也增大存储容量。但它没有任何容错措施，任何硬盘的损坏意味着整个阵列的损坏，且硬盘越多可靠性越低。

RAID1 用于磁盘镜像，由两个相同的磁盘构成，所有数据会被同时存放到两个磁盘上，当一个硬盘出故障时，仍可从另一个硬盘中读取数据，因此安全性得到保障，但系统的成本大大提高，因为系统的实际使用率仅有 45%。

RAID3 由 N+1 个磁盘构成，其中 N 个磁盘保存分条信息，第 N+1 个磁盘保存数据校验容错信息。当这N+1 个磁盘中的一个出现故障时，从其他N个磁盘可以恢复原始数据，这样可以达到数据恢复的目的。其最大的问题是数据校验盘可能成为性能瓶颈。

RAID5 类似 RAID3，但不同的是将数据校验码均匀分布在个磁盘，这样解决了 RAID3 中争用校验盘的问题，同一组内可进行并行写操作。但它在写入时校验运算处理上开销大，不适合输入输出数据量很大应用。

RAID 的级别选择主要考虑 3 个因素：可用性(数据冗余)、性能和成本。实际的磁盘阵列可以有某一级别的 RAID 或多个级别的 RAID 组合实现。

（2）NAS

NAS是一种以数据为中心的存储结构。按照存储网络工业协会(SNIA)的定义：NAS 是可以直接连接到网络上向用户提供文件级服务的存储设备。如图1.2所示，NAS是一种存储设备，有其自己简化的实时操作系统，它将硬件和软件有机地集成在一起，用以提供文件服务。NAS采用的协议是NFS和CIFS，其中NFS应用在UNIX环境下，而CIFS应用在NT/Windows环境下。

NAS 往往被定义为一种特殊的专用数据存储服务器，包括存储器件和内嵌系统软件，可提供跨平台文件共享功能。NAS 通常在一个LAN 上占用自己的节点，无需应用服务器的干预，允许用户在网络上存取数据。

NAS 的结构及采用的协议使得 NAS 具有以下优点[11]：⑴ 实现异构平台下的文

件共享，使得不同操作系统平台下的多个客户端可以很容易地共享 NAS 中的同一个文件。⑵ 充分利用现有的 LAN 网络结构，保护现有投资。⑶ 容易安装，使用和管理都很方便，实现即插即用。⑷适用性强。由于采用 TCP/IP 以及 NFS 和 CIFS 等标准协议，可以适应复杂的网络环境。⑸ 较低的总拥有成本。

NAS 也也有其不足，主要表现在[12][13]：⑴ 较低的 I/O 速度。NAS 采用的是 File I/O 方式，客户端的 File I/O 请求要经过 TCP/IP 协议栈封装，经过网络传输。NAS 收到 File I/O 请求后，经过 TCP/IP 协议栈解封装，获得文件系统能识别的 File I/O 请求，文件系统按 I/O 请求对存储设备进行读写；NAS 向客户端的响应采用类似的过程。所以，NAS 的网络协议开销大，其 I/O 速度比较低。正是因为这个原因，NAS 不适合在对访问速度要求高的应用场合，如数据库应用，在线事务处理。⑵ 较低的数据备份性能。这是因为数据备份需要占用 LAN 的带宽，浪费宝贵的网络资源，严重时甚至影响客户应用的顺利进行。

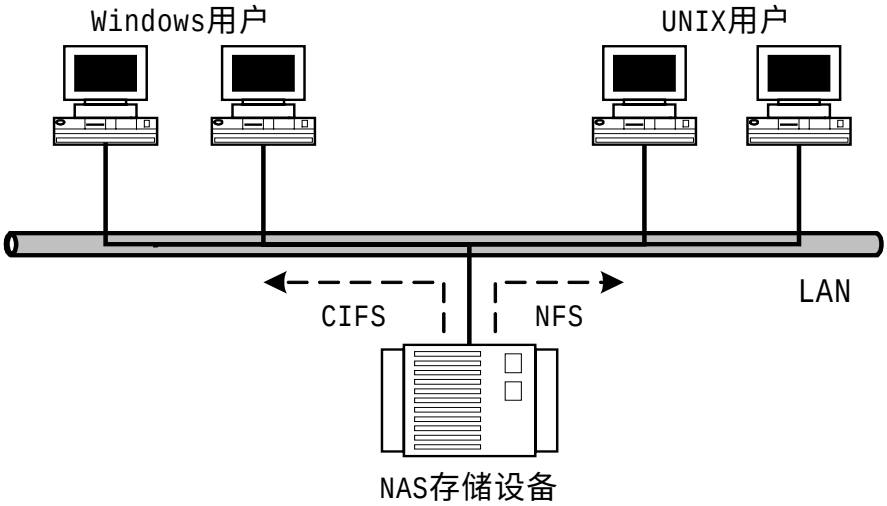


图 1.2　NAS 结构

（3）SAN

SAN 是一种以网络为中心的存储结构[14]。按照 SNIA 定义，SAN 是一种利用 Fibre Channel 等互联协议连接起来的可以在服务器和存储系统之间直接传送数据的存储网络系统。SAN 是一种体系结构，如图 1.3 所示，它是采用专用的网络协议 FC 构建、并与用户 LAN 网络分离的专用的存储网络，SAN 中存储设备和服务器之间采用 Block I/O 的方式进行数据交换。

SAN 采用独特的体系结构和网络协议，使其得具有以下优点[15]：⑴ 高性能。SAN 采用光纤通道提供 2Gbps 以上的带宽，使得数据的 I/O 速度远高于 NAS。⑵ 高可用性。在 SAN 中，不同的服务器可以访问同一存储设备，而同一存储设备可以被不同的服务器访问，也就是说，服务器与存储设备间是多对多的映射关系，所以用户可以通过不同的服务器访问存储设备，这样，当一台服务器出现故障时，其他服务器可以接管故障服务器的任务。⑶ 集中的存储管理。SAN 可以将不同的存储设备整合成统一的存储池，向用户提供服务，便于存储管理。⑷ 高可扩展性。在 SAN 中，服务器和存储设备相分离，两者的扩展可以独立进行，便于存储容量的扩展。⑸ 高性能的数据备份。由于 SAN 的存域网与用户网络分离，数据备份不占用用户网络带宽，数据备份不影响用户应用的性能。由于 SAN 具有上述优点，所以，对于数据库等 I/O 密集型的应用，SAN 有明显的优势。

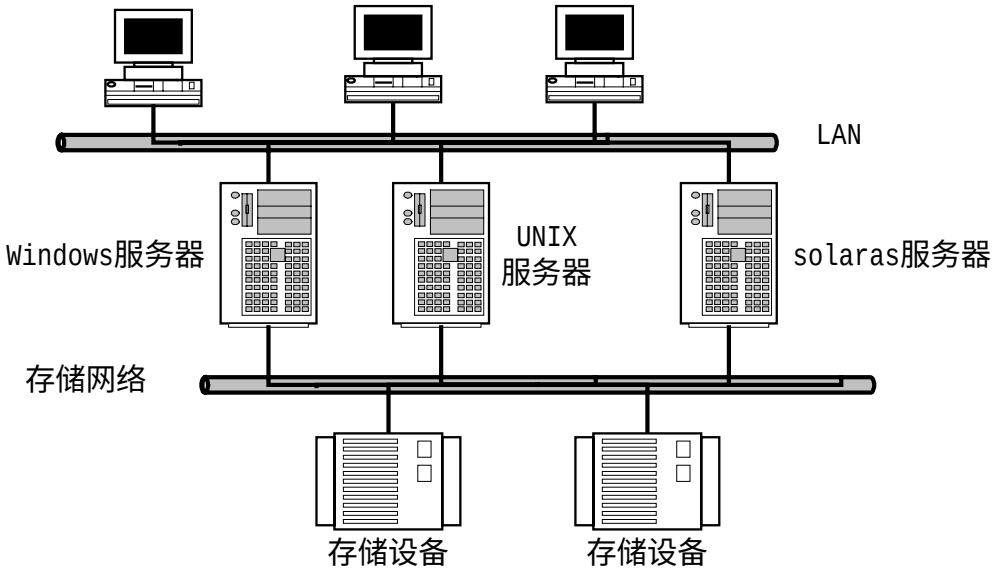


图 1.3　SAN 结构示意图

然而，SAN 也有着自身的一些不足：（1）设备的互操作性较差。目前采用最早和最多的 SAN 互联技术还是 Fibre Channel，而不同的制造商，光纤通道协议的具体实现是不同的，这在客观上造成不同厂商的产品之间难以互相操作。（2）较高的管理维护成本，构建和维护 SAN 需要有丰富经验的、并接受过专门训练的专业人员，这大大增加了构建和维护费用。（3）无法实现异构环境下的文件共享。SAN 中存储资源的共享一般指的是设备的共享，而非数据文件的共享。（4）SAN 采用的基于 FC

的网络互连设备，其成本非常昂贵。（5）连接距离限制在 10km 左右。所有这些都阻碍了 SAN 技术的普及应用和推广。

（4）IP-SAN

随着用于执行 IP 协议的计算机的速度的提高及 G 比特的以太网的出现，基于 IP 协议的存储网络实现方案成为 SAN 的更佳选择。IP-SAN 不仅成本低，而且可以解决 FC 的传播距离有限、互操作性较差等问题。

实现 IP-SAN 的协议有三种：FCIP、iFCP、iSCSI。三种协议的共同目标是封装块 I/O 数据以在 IP 网络中传送，但它们各有不同[16]：FCIP 通过网关透明地封装和解封装 FC 帧，以在 IP 网络中传送数据，主要用于通过 IP 协议实现 FC-SAN 的互连；iFCP 不仅解决 FC 的距离限制问题，还解决 FC 的路由体系结构，在 iFCP 中，每个 SAN 中用户可访问其自己的地址空间，用户通过地址变换采用虚拟地址访问远程 SAN，以解决地址空间的冲突问题；iSCSI 是最彻底的 IP 存储协议，其目标是将最流行的磁盘 I/O 协议 SCSI 封装以透明地在 IP 网络中传播。用 iSCSI 构成的 SAN 中通过 TCP/IP 协议实现存储设备和用户间的数据传送，它是最完全最彻底的 IP 存储网络。

IP-SAN 的体系结构类似于 FC-SAN，所不同的是用 iSCSI 协议代替了 FC 协议，它除了具备 FC-SAN 的特点外，还有自己的特点。在 IP-SAN 中，存域网和用户网采用相同的 TCP/IP 协议实现的，用户与存储设备间的数据传输可以直接进行而不必经过服务器转换；同时存域网和用户网在物理上是同一网络，用户与存储设备间存在着不通过服务器的直连通道。这就减轻了服务器的通信量。IP-SAN 和 NAS 都是建立在 IP 网络的基础上，这为 NAS 和 SAN 的融合提供了可能，而有些应用需要文件 I/O 和块 I/O 两种服务的并存，这样存储网络的成本将会大大地降低。此外，IP-SAN 使用 IP 协议实现数据块的传输，其连接距离远远大于 FC-SAN 的距离。

当然，在 IP-SAN 中，存域网是通过 TCP/IP 协议连接的，其网络在物理上与用户网属于同一网络，存储设备暴露在用户网络中，存储设备面临着 FC-SAN 中服务器所面临的各种攻击：恶意用户伪装用户身份、伪造用户信息、窃取数据、更改数据等。所有这些使得存储网络中的数据不再是安全的[17]。

上面给出了几种主要的存储系统，它们各有自己的优点，也有自己的不足。在实际的应用中，根据应用的不同需要，可以选择地配置某种存储系统或几种存储系统的组合。实际的存储系统中，可能对上述存储系统进行某些方面的改进，达到优化其性能的目的。

1.2.3 信息存储的研究热点

应用不断给信息存储提出新的要求，为了适应信息存储的需求，出现了一些新的存储研究方向：对象存储、存储虚拟化、存储智能化、存储安全、信息生命周期等。

（1） 对象存储技术

对象存储技术[18][19][20]是由 CMU 大学提出的，它是一个类似 NAS 存储设备的智能磁盘驱动器，但将管理、文件系统语义和存储转发相分离，仅实现基本的存储元语，由文件管理器实现文件系统的高层管理部分。它对外提供以太网、ATM 等数据通信接口与 IP 网络相连，或者通过 FC 接口连接到 SAN 上。

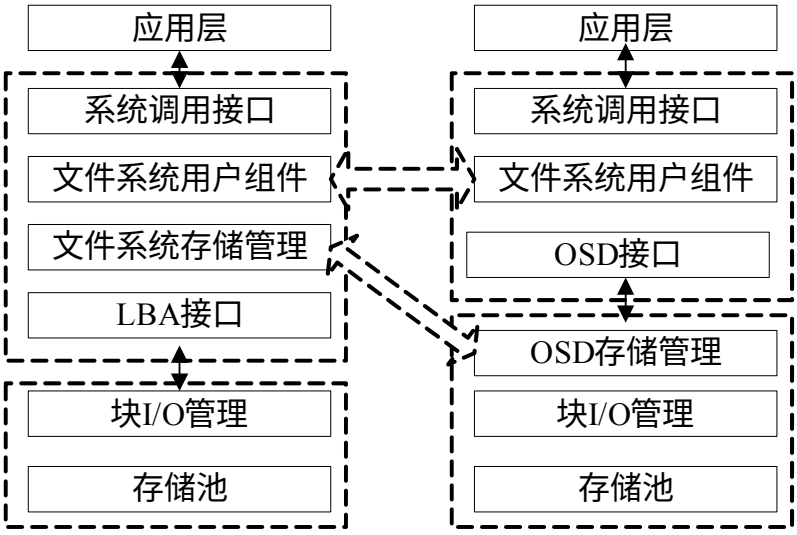


图 1.4 传统存储模型与对象存储模型的比较

对象是一个虚拟的实体，在对象存储系统中，存储设备和数据都可作为对象看待。数据对象既不同于数据块，也不同于文件，而是有大小可变的数据和元数据组成，其中，元数据用来实现数据定位和安全。对象存储设备类似于 NAS 存储设备，有简单的文件系统，能对设备中的数据对象进行有效的管理。如图 1.4 所示，与传统

的文件系统相比，对象存储将文件系统的功能分为两个分开的逻辑组件：文件管理器和存储管理器，前者由元数据服务器实现，进行层次的用户管理、命名空间管理和存取控制，后者下移到存储设备，进行实际的数据存储和数据检索。

对象存储同时具备 NAS 和 SAN 的优点：跨平台共享特性和高速直接访问，使得对象存储成为了具有高性能、高可靠性、跨平台以及安全的数据共享的存储体系结构。此外，对象存储可以通过对象属性对用户访问进行验证，从而为存储对象的访问提供了安全保障。

（2）存储虚拟化

随着存储容量的不断增大，存储管理越来越复杂，管理费用占存储预算的比例越来越高，如何管理好数据成为当务之急[22]。存储虚拟化技术是解决存储管理问题的有效手段，已经越来越受到业界的关注。据统计，目前数据存储花费占企业总体开支的比重越来越大，而其中有 91%用来购买存储管理软件，企业不得不考虑如何以有限的资源满足存储发展的需求。

存储虚拟化将物理存储设备映射为单一逻辑存储资源池。通过虚拟化化的技术方法，把各种异构的存储资源统一成对用户来说是单一视图的存储资源（storage pool），为用户或应用程序提供虚拟磁盘或虚拟卷，并为用户隐藏或屏蔽具体存储设备的各种物理特性[21]。利用存储虚拟化技术，同时配合以分条、分区和逻辑单元掩码等技术，用户可以根据自己的需求，方便地将这个大的存储池分割、分配给特定的主机或应用程序。达到对用户完全透明，实现互操作性的目的。

存储虚拟化可在不同的级别上进行：存储设备级、网络级和服务器级[21]。典型的设备级的虚拟化是 RAID，通过阵列控制器，屏蔽掉各磁盘间数据的分布。网络级存储虚拟化在路由器、交换机或专门的元数据服务器上将网络中的存储设备映射为单一的存储空间，为用户屏蔽存储设备的分布和数据的分布细节，简化用户的存储操作。服务器存储虚拟化则在应用服务器上通过虚拟化层将存储设备映射为单一的存储空间。

采用存储虚拟化技术，屏蔽具体设备的物理特性，简化系统管理员的管理操作，降低管理的复杂性；允许客户以完全透明的方式在存储设备上存储数据，动态分配

存储空间，提高存储效率；支持物理存储空间的动态扩展，提高存储系统的性价比。

（3）存储智能化

存储的智能化的目的是减少人工存储管理，实现存储自治管理，从而降低意外事故所造成的损失[23]。存储智能化包括自管理、自配置、自组织、自调节、自恢复 5 个方面，这 5 方面相互制约，相互协调，达到存储系统自管理的目的。

智能化存储结构有 5 个功能模块组成[24]：客户端、路由器、存储代理、监督员、系统管理员。客户端发出 I/O 请求；路由器将请求传送给合适的存储代理；监督员进行系统层次的规划，承担组织者的角色，动态调整代理上数据集的分配、数据集的冗余调度以及路由策略，另外，监督员也负责监控其底层系统的可靠性和性能状态，并作出预测，上层管理模块与系统管理员交互，接收高层次的目标和规范；存储代理担当存储控制器的角色，负责数据的读取、存放。管理结构层与 I/O 请求路径在逻辑上是独立的，而请求路由在逻辑上和客户端、存储代理也是分开的，自主存储的系统结构示意图如图 1.5 所示。

管理结构层赋予了整个系统的智能性，使存储包含了进化、优化的思想：预先分析错误，并给予纠正，还包括自主故障切换、故障恢复、负载平衡和自动管理软件升级的能力，从而保证关键数据能够实现 24 小时×365 天的可用性。

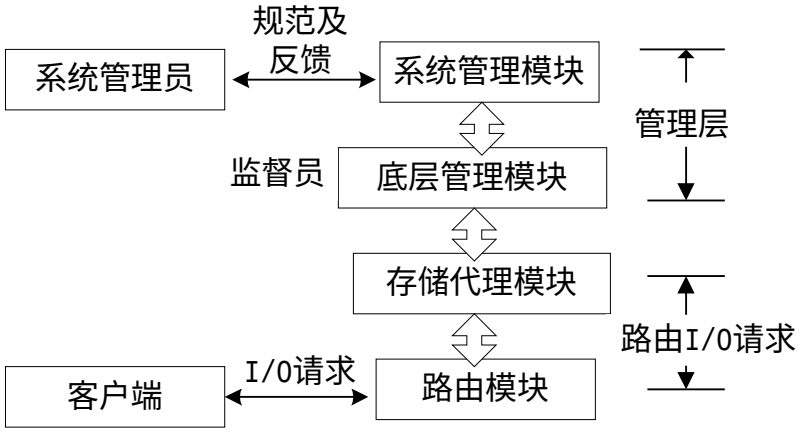


图 1. 5 智能化存储结构图

（4）存储安全

由于过去几年的几起数据安全漏洞事件闹得沸沸扬扬，存储安全最近越来越被工业界和学术界所重视。由 IBM 发起，EMC、HP、HDS 等公司参与组建一个名为

Aperi 的开放源代码存储联盟，开发管理存储设备的通用平台。学术界则提出许多方案，如卡耐基·梅隆大学的 NASD[25]，惠普实验室的 PLUTUS 系统[26]，斯坦福大学的 SiRiUs 系统[27]，这些系统尚未做成方案推入市场，只是其中的某些技术形成产品，比如 SSL 加速卡、VPN 加密加速卡。

目前，对于网络存储安全的研究主要包括以下四个方面[24]：1）增强文件服务器的安全。例如在 AFS 中，其服务器使用 Kerberos 这一标准的安全认证系统来增强系统的安全性。2）客户端加密文件系统。例如在 CFS[28]中，加密/解密的操作移到客户端的文件系统中，从而解决用户级或系统级加密的需求，这种方案中数据往往以加密状态存放到存储设备上，同时也以加密状态在网络上传输。3）客户端直接访问磁盘的认证机制。大多采用 Kerberos 认证，其中以 NASD、SNAD 系统为代表。在 NASD 中用户在访问某个文件数据时，首先由文件管理器对其进行身份验证和授权，之后通过特定的访问接口和磁盘设备进行交互，设备控制器能对用户请求进行验证，从而达到限制访问数据的目的；SNAD 中，当文件数据写到磁盘时，还同时写入每个文件数据块的数字签名，客户端使用签名来验证数据是否被修改过。 4）高度可扩展的文件系统。将多个独立的文件服务器合并成单一命名空间，并采用一定的安全措施。在 SFS 文件系统中，通过使用文件服务器的地址和公钥生成每个服务器的唯一标志符，服务器使用该公钥建立一些安全通道，然后客户端采用一些认证方案进行访问[29]。

（5）信息生命周期管理（ILM）

信息生命周期管理假定信息在不同的时间有不同的价值，因此在信息生命周期内以适当的价格、在适当的时间、把数据放在适当的设备上，以提供适当的服务[30][31]。数据生命周期管理是层次存储管理（HSM）的再生，但它与层次存储管理有本质的区别，主要表现在：1）HSM 产品没有整合网络环境下数据的智能性和可扩展性，在大型异构的客户服务器环境下增加了管理的复杂性。2）HSM 方案的源和目标卷的关系是静态的端对端的关系，迁移规则不能在多 HSM 服务器共享，对大型分布式环境不实用。3）HSM 技术缺少网络中的存储资源的全局视图，要求管理者不断地改变迁移规则以适应存储环境的改变。

数据生命周期管理要求几个组件：首先必须有数据和存储资源的视图，必须能对数据按企业的要求或相对重要性分类，最后必须有数据管理策略如何分配数据以适当的存储资源。数据必须保存为在线、半在线、离线等形式，什么时候数据必须复制迁移或删除等，智能和动态地自治这些决定和行动是数据生命周期管理的核心内容。

数据生命周期管理的最大挑战是描述数据的相关值或重要性，以及所用的存储资源。按照业务的需要，把数据放在适当的存储上，IT 能有效地在多存储资源上分配数据，可以改进存储的利用率，减少存储的预算。只有这样管理者才能把正确的数据在合适的时间放在合适的存储上。而将这些进程自动化，可以简化存储管理，降低存储成本，提高存储的性价比。

综上所述，对象存储通过定义对象的语义，简化服务器的管理，便于存储系统的扩展，同时通过语义为存储信息提供适当的安全保障；存储虚拟化和智能化从不同的方面简化存储操作，实现存储自管理，提供高可用性的存储，同时降低存储的管理成本；存储安全保证数据的安全可靠，保护信息的价值；而信息生命周期管理则通过数据的合理分布和组织，提高存储的 I/O 速度和存储的利用率，为用户提供高性价比的存储。存储的研究热点从不同的方面优化信息存储，从而推动存储技术的不断发展，以满足应用对存储的需求的增长。

1.3 存储系统的数据分布和组织

1.3.1 计算机系统层次存储模型

现代计算机虽然仍然以冯.偌依曼的“存储程序原理”为基础，但与冯.偌依曼计算机有很大的不同，其中最大的不同是：冯.偌依曼计算机以运算器为中心，而现代计算机以存储器为中心。存储器保存着处理器所执行的程序和处理器处理的数据及其运算结果，随着运算器性能的不断提高和计算机处理的数据的爆炸性增长，存储器越来越成为计算机系统的性能瓶颈。计算机系统的存储分为内存和外存储设备，内存用来缓存处理器运行所需的程序和数据，它是由高速、容量较小的存储设备组成的；数据的永久保存是通过外存储设备来实现的，外存储设备通常容量大、速度低，

价格相对于内存而言便宜。内存和外存的有机组合，为处理器提供速度快、容量大、性价比高的数据存储。

传统的存储系统从 CPU 到外分为多个层次[53]，第一层通常指处理器的内部 Cache，第二层是主板上的内存系统，第三层是外存储器，如磁盘等，第四层一般指磁带设备。各个存储器层次的数据存取时间和容量是不同的，从第一层到第四层，存储器的速度越来越低，容量越来越大，价格越来越便宜。存储网络的出现，使得计算机可以通过网络访问远端的存储设备，扩充计算机系统可以使用的存储空间，然而，它也使存储系统的数据组织更复杂。因为每个存储设备要提供相应的存储空间，就必须把这些独立的存储空间组成统一的逻辑存储空间，当存储系统中存储设备和层次的增加，就必然增加存储系统的组织和数据分配的难度。

各层存储设备的数据组织单位各不相同，Cache 和主存是以字节为单位，磁盘和磁带则以数据块为单位。Cache 和主存组成主存系统，Cache 的目的是提高主存的速度，磁盘和磁带组成外存储系统，磁带的使用则是降低成本和增大存储容量。主存和外存间数据的访问是以文件为单位，它是通过文件系统来实现的。文件系统实现存储系统的逻辑空间到物理空间的映射，通过它将文件保存在确定的物理存储设备上。

每个文件给定一个名字以区别其它文件，逻辑空间则是将文件通过文件名按一定的规则组合的命名空间，以方便用户查找和访问。目录是文件的集合的标识，一般将具有某种相关性的文件和目录存放在某一目录下。同一目录下的文件名不能相同，但不同目录下的文件名则可以相同，这是因为命名空间的文件的访问是通过目录路径来访问的。

物理空间是由存储设备按一定的规则组成的空间，物理空间可以由一序列连续的线性块号表示，每一个块号对应一个物理块，块号就是对应块的物理地址。计算机通常利用机器字长来作为寻址单位，由于机器字长有限（16 位，32 位或者 64 位），直接按机器字长寻址的空间范围有限，当存储空间很大时，存储空间的地址必须通过间接寻址来实现。间接寻址扩大了计算机的寻址空间，但也降低了数据访问速度，为了提高速度，可采用特殊的方法来实现，如直接用硬件的方法来实现。

从逻辑空间到物理空间的映射从逻辑上包含三个重要部分：一是逻辑存储空间的组成；另一方面就是在逻辑存储空间和数据内容之间建立关联，这种关联必须是可逆的，也就是通过这种关系不但可以存放并且可以取得数据，实现存储的目的；第三个方面按照上两种的规则建立一系列操作（实现方法），用于具体执行存储过程，对于可重写的存储空间还要建立回收、分配等管理。

文件系统是计算机系统用来实现存储系统的组织和数据的访问的软件模块，它实际上包括存储空间的组织和用户数据的组织两个层次。文件系统对于上层应用提供基于文件目录的树形存储空间结构，然后把每个文件目录映射到下层存储设备的物理存储空间。文件系统按照一定的数据结构将存储设备格式化为逻辑磁盘或逻辑卷，不同的文件系统有不同的数据结构。如 Windows 采用 FAT 构成文件索引结构，而 Unix 采用 Inode 索引来管理磁盘和文件的组织和访问。

逻辑磁盘是文件系统可随机访问的有固定大小的数据块的线性序列，逻辑磁盘到物理存储设备的映射是通过磁盘设备驱动程序来实现的。逻辑磁盘可以是包含在一个物理设备之内，也可能包含在几个实际的物理存储设备（RAID）。文件系统首先将文件映射到逻辑磁盘上，然而，逻辑磁盘并不是真正物理数据存放结构，数据存储的物理结构还是依赖于以磁盘为主的存储设备，因此文件系统必须完成数据逻辑组织结构到磁盘物理结构的映射，并且这种映射应该是高效和可靠的。

这样，文件系统对数据的访问由两次映射完成，一是从用户的文件命名空间到文件系统的逻辑卷，二是从逻辑卷到具体的物理存储设备。它实际上包括两方面的内容：一是通过特定的数据结构将存储设备构成一定组织结构，不同的操作系统有不同的数据结构。另一个方面是在文件系统给定的数据结构上操作，如对文件和目录的查找、删除、复制等等。

1.3.2 网络存储系统的性能模型

单个的存储设备无法满足应用的需求，存储系统是解决存储性能瓶颈的最好方法。随着网络速度的不断提高，网络存储系统成为最重要的存储系统，网络存储系统的体系机构多种多样，但可抽象为如图 1.6 所示的体系结构。网络存储将多个存储设备通过网络构成一个大的存储空间，服务器负责存储管理和数据组织；客户端和

服务器通过用户网络相连，客户对存储系统的访问通过服务器进行。尽管用户网络和存储网络在逻辑上是两个不同的网络，但是，在物理上它们既可能是两个不同网络，也可能是同一网络。因此，用户对存储设备的访问既可以是通过服务器间接访问存储设备，也可以从服务器获取元数据信息后直接访问存储设备。

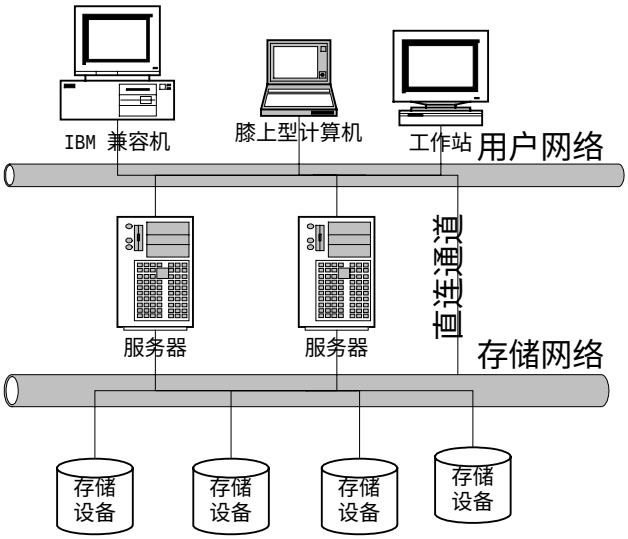


图 1.6 网络存储系统的体系结构

为了提高存储系统的 I/O 性能，在用户访问存储设备的路径上设置 Cache，通常，存储设备、服务器和客户端都设置有 Cache。这样，通过数据传输路径上的 Cache 构成一个层次存储系统。

在层次存储系统中，各级存储的性能各不相同，从客户端到存储设备的速度越来越低，容量越来越大。假定从客户端到存储设备，用户访问各级存储的速度分别为 $v_1, v_2, \ldots, v_n$，用户在各级存储访问数据的概率分别为 $p_1, p_2, \ldots, p_n$，则存储系统访问的平均速度为：

$$v = \sum_{i=1}^{n} p_i \cdot v_i \qquad (1\text{-}1)$$

从式（1-1）可以看出，提高存储系统的速度有两种方法：提高各级存储设备的速度和提高高层存储设备中数据访问的概率。

1.3.3 网络存储系统中的数据组织

存储系统的数据组织就是通过数据在存储系统中的适当分布来提高存储系统的性能。根据上述存储系统性能模型中，如果各级存储设备是单个存储设备，提高存储设备的速度只能通过元器件技术，而提高高层设备的命中率则是通过在多级存储设备中的数据组织来实现的。当某一级存储设备由多个组成时，这时可以利用多个存储设备的并行访问来提高该级存储设备的访问速度。如何利用多设备的并行性，则与数据在存储设备中的分布有关。这样存储系统中数据的分布组成一个二维空间：存储空间和传输路径，相应地，存储系统的数据组织可分为两种类型：存储空间上的数据组织和传输路径上的数据组织。

(1)存储空间上的数据组织

在存储空间上，数据组织不仅可以提高存储系统的速度，还可以提供可靠性、可用性等不同的性能。通过数据组织提高速度的主要思路是利用存储设备的并行性，按照这一思路，数据应尽可能地均匀分布在各存储设备，RAID 的设计正是基于这一思想来提高速度的。在网络存储系统中，由于存储设备不一定同构，且其提供的 I/O 接口有文件和数据块两种，这时数据分布更复杂。考虑到数据访问的概率不等，存储设备的访问速度也有差别，这时数据的分布必须考虑到数据访问的局部性，应尽可能让访问概率高的数据分布在速度快的存储设备。因此，存储系统空间上的数据组织有两种不同的标准：均匀分布和按访问概率分布。存储系统的可靠性和可用性则通过在不同的存储节点保存副本来实现，在保证可靠性和可用性的同时减少副本数量、提高存储系统的利用率正是数据组织所要做的工作。

(2)传输路径上的数据组织

在传输路径，数据分布算法的主要目的是为了提高存储系统的速度，从体系结构的观点，主要利用数据访问的局部性来实现的。由于高层存储设备通常速度快，但容量小、价格贵的，因此，数据组织试图提高高层存储的命中率，考虑到访问的局部性，高层存储的替换算法就尤其重要。常用的 LRU、LFU 等算法基于低层存储设备的访问速度相同的情况，在存储系统中，尤其是网络存储系统中，低层存储设备的 I/O 速度不一定相同，这时，替换算法将会更复杂。在存储系统的数据传输路径

上存在着多个 Cache，考虑到 Cache 的利用率，多个 Cache 如何配置，以及多 Cache 如何合作使用是数据分布要考虑的又一问题。所以，传输路径上的数据组织设计的两个标准是加速比和 Cache 的利用率。Cache 通过保存多个副本来实现提高存储系统的性能，多副本的一致性是数据组织算法必须考虑的问题。

1.3.4 数据组织的重要性

对于单个存储设备，有三种基本的组织方法，这就是连续分配，链接分配和索引分配。这三种不同组织方法形成了文件在存储设备上的不同物理结构，也就是常说的顺序文件、链接文件和索引文件[33]。连续分配方式中，每个文件在磁盘上所占的盘块是完全连续的，这样虽然有利于访问速度的提高，但不利于磁盘的充分利用，容易在磁盘上形成大量的碎片，虽然这些碎片可以通过“紧凑”技术加以整理，但非常耗时。链接方式将数据离散地分布到存储设备中，而数据的物理地址由文件分配表来记录，数据的查找必须先从文件分配表中获得数据块地址再查找数据，速度比连续方式低，但存储设备的利用率高。索引分配采用索引实现数据到存储设备的映射，索引的功能类似文件分配表，但其实现方法不同对数据读写性能的影响也不同。

在存储系统中，存储设备更多，数据组织更复杂，数据的组织对系统的性能优化影响更大，具体表现在以下几方面：

1）数据组织对数据 I/O 性能的影响。在存储系统中，有大量的存储设备，通过适当的组织算法可以达到提高数据 I/O 速度的目的。例如，在 RAID 中，通过数据的分条，将数据分布在多个磁盘中，使得数据的读写在多个磁盘中并行进行，充分利用设备的并行性，从而达到提高性能的目的。在大型存储系统中，由于数据量非常大，数据的查找要花费时间开销，适当的算法可以减少这种开销。

2）数据组织对数据可靠性的影响。如前所述，数据的可靠性是存储系统的主要目的之一，通过数据的副本或冗余可以提高数据的可靠性。副本和冗余如何在存储系统中的分布，在保证数据的可靠性的同时又提高 I/O 性能，正是数据分配算法所要考虑的问题。在 RAID5 中，将数据的冗余分条分布在所有磁盘而不是集中在单一磁盘就是避免其成为性能瓶颈。

3）数据组织对数据可用性的影响。由于设备的故障或 I/O 请求的过多，存储系

统的 I/O 服务可能中断，而某些应用要求存储系统提供不中断的 I/O 服务。通过对数据适当组织，使数据的访问尽可能分布在不同的服务器，可以达到提高可用性的目的。

4）数据组织对存储系统的扩展的影响。考虑到数据增长的速度快和成本问题，存储系统的构造不可能一步到位，存储系统必须不断地扩展。在存储系统扩展的过程中，如何保证数据合理分布，达到提高其 I/O 性能，正是数据组织要考虑的问题。同时，数据组织算法还必须考虑存储系统的扩展尽可能不降低其可用性。

实际上，数据组织算法的设计目的正是为了优化存储系统的 I/O 性能、可靠性、可用性和可扩展性等性能，当然，算法不可能使所有性能都优化，但可以综合考虑，达到系统的整体性能最优的目的。

1.4 本文的主要内容

由于存储系统的数据组织对存储系统的性能、可靠性、安全性的多方面有很大的影响，本文从提高存储系统的性能这一要求出发，讨论存储空间和数据传输路径上的数据组织方案。

第二章设计对象存储系统，它是数据组织的基础，所以的数据组织方案是根据对象存储系统的特点提出的。首先对比 NAS 和 SAN 给出对象存储的设计思路，然后分析对象存储系统的的体系结构、软件结构和数据访问模式，通过性能分析和实验表明三方传送模式适合可扩展大型存储系统。

第三章讨论数据在存储系统空间上的分布算法。首先比较两类存储空间映射算法——映射表和映射函数的优缺点，然后定义了数据分布算法最优的相关定义，在此基础上设计了存储系统中的数据分布算法和系统扩展时数据迁移算法，两类算法在保证算法的时间和空间开销小的同时，实现了数据对象在可扩展存储系统中的按性能均匀分布，达到充分利用多存储节点并行性。

第四章研究存储系统数据迁移方案。应用对存储系统的服务质量提出要求，而对象存储为存储系统服务质量的实现提供了可能。本章通过分析对象存储的特点，建立基于 QoS 的 I/O 模型，在此基础上分析和设计基于 QoS 的数据迁移模型，并讨

论其实现的相关问题，以达到数据迁移减少对存储系统 I/O 的影响，提高存储系统的可用性。

第五章研究存储系统的 Cache 替换算法。根据存储系统的特点，讨论 Cache 对存储系统的性能改善标准——加速比，找出影响加速比的主要因素。在此基础上设计出两种 Cache 替换算法：LAT 算法和 WLFRU 算法，这两类算法在考虑到命中率的同时，注重数据对象的设备访问成本对存储系统的 I/O 性能的影响。实验显示它们的性能优于常用的 Cache 替换算法。

第六章研究对象存储系统中的 Cache 方案。在数据传输路径上设置 Cache 必须考虑到数据访问的特点。结合对象存储系统中的三类存储实体的特点，讨论它们的 Cache 方案，设计中不仅考虑到单个 Cache 的实现，同时兼顾多个 Cache 间的合作使用，并结合存储系统的数据传输模式提出了混合传输模式方案，实验表明该方案对提高存储系统的性能有很大的帮助。

第七章总结全文。

1.5 课题来源

本课题来源于国家自然科学基金《统一存储网(USN)理论、结构与实验研究（60173043）》和《进化存储系统(ESS)理论与实现技术研究（60273073）》，同时也是国家 973 重大基础研究项目《下一代互联网信息存储的组织模式和核心技术研究（2004CB318203）》的研究内容之一。

2 海量对象存储系统

存储介质和通道技术的发展，使得存储设备的容量和性能不断的提升，然而存储设备性能增长的速度还是难以匹配处理和网络传输部件的性能增长，容量增长也很难满足应用的需求。网络技术和Internet的发展推动了存储需求的增长，按指数增长的存储需求迫切需要新的存储结构。传统的网络存储——附网存储（Network Attached Storage, NAS）和存储区域网（Storage Area Network, SAN），在一定范围内能满足应用的需要，但也存在这样或那样的不足。一种新的存储结构——对象存储（Object storage, OS）的出现，为存储系统的发展提供了新的方向。

2.1 对象存储思路

对象存储是一种新型的网络存储结构，它与传统的网络存储——NAS 和 SAN 有很大的不同，能满足应用对存储的新需求。本节首先从分析 NAS 和 SAN 的特点入手，讨论对象存储的特点。

2.1.1 NAS

NAS 是一种以数据为中心的存储结构。按照存储网络工业协会(SNIA)的定义：NAS 是可以直接连到网络上向用户提供文件级服务的存储设备。NAS 有其自己简化的实时操作系统，它将硬件和软件有机地集成在一起，用以提供文件服务。目前采用的协议是 NFS 和 CIFS，其中 NFS 应用在 UNIX 环境下，最早由 SUN 开发，而 CIFS 应用在 NT/Windows 环境下，是由 Microsoft 开发。

NAS 的体系结构如图 2.1，它将文件系统移植到存储设备，实际上是不同的文件系统间通过网络实现共享。NAS 具有以下优点：实现异构平台下的文件共享；充分利用现有的 LAN 网络结构，保护现有投资；容易安装，使用和管理都很方便，实现即插即用等。NAS 也有不足：NAS 采用的是 File I/O 方式，File I/O 在客户端和 NAS 端都要采用 TCP/IP 协议封装或解封，需要巨大的网络协议开销，因此 NAS 的文件访问速度相对 SAN 而言很低； NAS 只能对数据实现共享而不能实现存储空间的真

正共享；NAS 通过服务器访问存储设备，服务器成为系统的性能瓶颈；用户网络和存储网络都是 IP 网络，存储系统的安全成为问题。

2.1.2 SAN

按照 SNIA 定义，SAN 是一种利用光纤通道（FC）等互联协议连接起来的可以在服务器和存储设备之间直接传送数据的专用网络，存储设备和服务器之间采用 Block I/O 的方式进行数据交换。如图 2.2 所示，传统的 SAN 采用光纤通道（FC）作为服务器和存储设备间的网络协议，存储设备相当于服务器的外接磁盘。独特的体系结构和构建技术使得 SAN 具有如下优点：目前光纤通道可提供 2-4Gbps 的带宽，新的 10Gbps 的标准也正在制定之中，速度较快；用户可以通过多台服务器访问存储设备，可用性高；多用户可通过服务器访问同一存储设备，实现存储空间的共享；采用专用的网络整合各种不同的存储设备，形成一个统一的存储池，实现集中的存储管理；服务器和存储设备相分离，两者的扩展可以独立进行，具有高可扩展性；支持大量的设备，理论上具有 1500 万个地址；支持更远的距离，另外通过光纤通道、网卡、集线器、交换机等互联设备，用户可根据需要灵活地设置服务器和存储设备；用户网络和存储网络分离，存储系统的安全通过服务器实现；数据备份只在存储网络中进行，不占用用户 LAN 带宽等。

但 SAN 仍有自身的缺陷： SAN 实现不同平台下的存储空间的共享而非数据文件的共享；采用 FC 技术，其构建、维护、管理都非常困难； FC 的互联设备和存储设备都非常昂贵，成本高；不同厂商的 FC 设备的互操作性很难解决；连接距离有限；在 SAN 中，每个逻辑单元（LU）只被一个主机使用，也就是说，存储空间在主机间分区，主机将存储设备看作与其直接相连，共享是通过文件服务器间接访问下面的存储设备来实现的，一个服务器分配一定的逻辑单元，不同的服务器不能访问没有分配给它的逻辑单元号，不是真正意义上的共享。

由于 SAN 的设备成本高，其管理维护困难，FC—SAN 主要用在高端应用，其应用范围有限。iSCSI 的出现，使得 SAN 可利用通用的 TCP/IP 协议实现服务器和存储设备间互连，通过 iSCSI 实现的 SAN 称之为 IP-SAN，它克服了 FC-SAN 的互操作难、成本高等缺点。如图 2.2 所示，IP-SAN 用 iSCSI 协议代替 FC-SAN 中的 FC

实现服务器和存储设备间的数据块的传输，使得存储网络可以用 IP 网络实现，存储网络的成本大大降低。但是，由于存储网络和用户网络都是 IP 网络，这也导致新的安全问题。在 FC-SAN 中，由于采用 FC 构成存储网络，用户必须通过服务器才能访问存储设备，存储的安全通过服务器来实现，只要保证服务器安全可靠，就可以保证存储网络的安全。在 IP-SAN 中，用户网络和存储网络采用相同的 TCP/IP 协议，存储设备暴露在用户网络中，用户可以很方便地访问存储设备，这时仅仅通过服务器很难保证存储网络的安全。

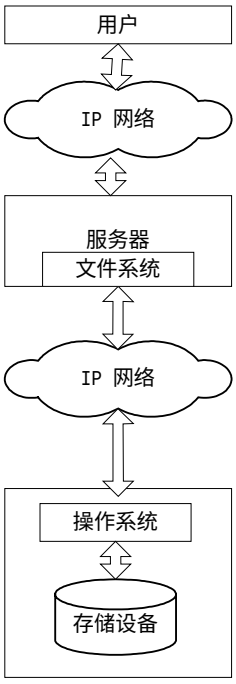


图 2.1 NAS 的体系结构

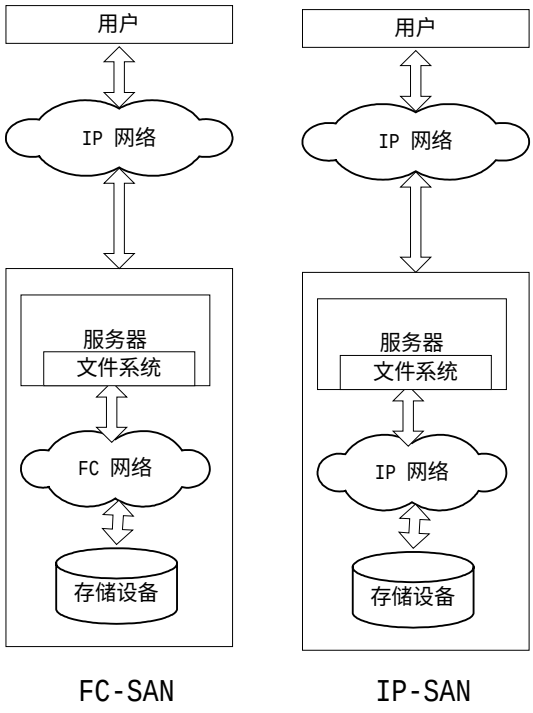


图 2.2 FC-SAN 和 IP-SAN 的体系结构

2.1.3 对象存储

如前所述，NAS 和 SAN 各有特点，它们能满足一定应用的需要。然而，应用的发展要求存储系统具有高性能、高可靠性、高可用性、高可扩展性和安全性，NAS 和 SAN 虽然能部分实现上述目的，但也存在一些问题[54]。

（1）安全问题。由于 SAN 是基于 FC 的， FC 技术使用不普遍，且其通过主机保护，现有的攻击对其影响不大。然而，在 NAS 和 IP-SAN 中，用户网络和存储网络共用一个网络，存储设备暴露在用户网络中，安全问题不可避免。在 NAS 中，通

过用户名和密码认证用户，一旦用户被认证，该用户就可以访问整个存储设备；在 SAN 中，通过主机间分区的方法，保证一定的逻辑单元只能被某一确定的服务器访问。在这两种情况下，用户要么不能访问某一存储设备（存储卷），要么可以访问整个存储设备。在这种粗粒度的安全支持下，存储的共享将导致安全问题。

（2）可扩展性。在 NAS 和 SAN 中，用户都是通过服务器访问存储设备，当用户数增加时，共享访问要求服务器间协调。在 SAN 中，可以采用专门的元数据服务器实现空间定位，这样可以取得较好的可扩展性，但是，这种协调可能导致附加成本，如元数据一致性的实现。

（3） 端到端的管理。当数据直接依附在主机时，端到端的管理很容易实现，因为任何处理都是在单个盒子中进行。然而，NAS 和 SAN 中的存储设备作为单个存储实体，单个数据单元的端到端的管理很困难。

对象存储能很好地解决上述问题。对象存储开始于 CMU 的 Garth Gibson 等提出 NASD[55]，目前 SNIA 成立了 OSD 技术工作组（OSD-TWG），在 NASD 项目的基础上对 SCSI 命令集进行扩展，这些扩展的命令集作为 OSD-2 命令集而存在，并向 T10 委员会提交了一份草案[56]。目前进入商业运作的基于对象存储的文件系统有 Panasas 公司的 PanFS[57]和 Cluster File Systems 公司的 Lustre[58]。

如图 2.3 所示，对象存储类似 NAS，但有所不同。NAS 将文件系统移植到存储设备，而对象存储则将文件系统的功能分开，将文件系统关于存储管理的部分下移到存储设备，而将文件系统中的用户组件保留在服务器中，服务器与存储设备间通过 OSD 接口通信。与 SCSI 接口不同，OSD 接口给存储对象定义了丰富的语义，语义除了提供对象属性、QoS 等外，还提供存储系统的安全保证，保证对象存储访问的安全性。对象存储的安全管理由服务器提供，对象存储设备只接受由服务器授权的 I/O 请求，而不管请求由谁发出。与 NAS 和 SAN 不同，对象存储有以下特点：

1) I/O 数据单位

与传统面向块的逻辑单元不同，对象存储不提供对不相关的数据块的访问，它允许通过存储对象来访问相关的数据块。存储对象是一个虚拟实体，它是一个客户认为相关的一组数据，类似于平面文件系统中的大小不限的比特流文件，但对象又

不同于文件，它只是有属性的比特序列。尽管对象既不同与数据块，也不同于文件，但可通过对象语义的定义来实现文件或数据块的 I/O，因此，通过对象实现了文件 I/O 和数据块 I/O 的统一。

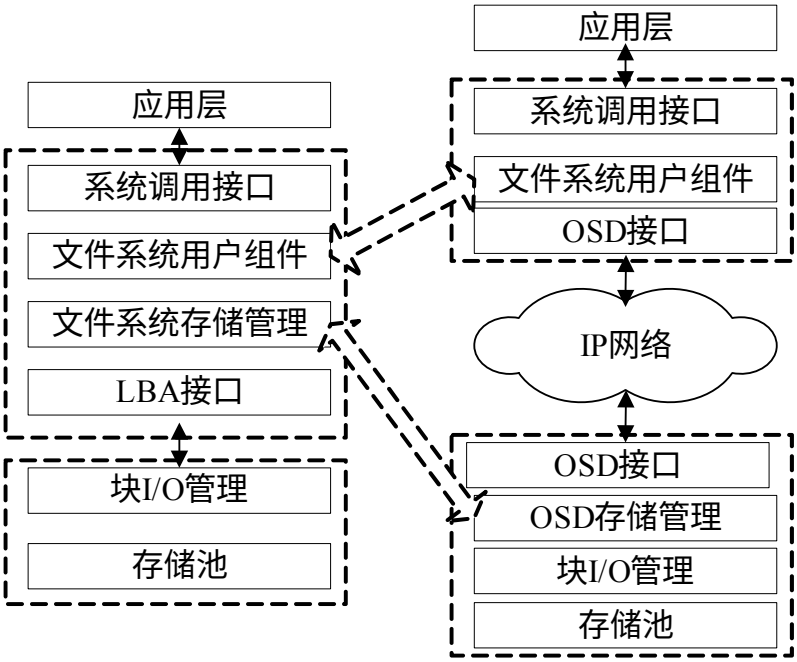


图2.3 对象存储模型与传统存储模型的比较

2) 存储管理的分离

对象存储的管理分两部分：服务器和对象存储目标端（Object Storage Target, OST）。服务器将所有存储目标端的存储空间虚拟为统一的存储空间，同时负责用户的命名空间的管理，在认证用户请求后，分配给其唯一的对象 ID，并按一定的算法实现用户命名空间到存储空间的映射。OST 负责与其相连的存储设备的管理，它没有对象命名空间，只有平面对象 ID 空间，OST 根据对象 ID 在存储设备中分配存储空间或访问存储对象。

3) 保证数据的安全

对象存储通过信用保证所有操作的安全，信用包括允许客户的操作集和完整性编码。对每个操作简单提供信用，即使信用不加密，也能提供保护（由于完整性编码），因为不可能偶然为一个操作提供一个可靠信用。为了提供安全，对信用的某些

形式的加密保护是必要的。对象存储提供安全和保护是在对象级而不是整个卷级，因此允许不可信的客户位于存储网络中，并允许共享访问存储数据而不需访问整个卷的数据。另外，由于客户不能直接访问定位元数据，可提供额外的保护，因为不可能有错误配置和蠕虫主机破坏定位元数据。

2.2 对象存储系统的体系结构

对象存储不同于 NAS 和 SAN，与其体系结构有关。下面分析对象存储系统的硬件结构和软件结构。

2.2.1 硬件体系结构

对象存储的硬件体系结构如图 2.4 所示，它有三部分组成：元数据服务器、OST 和客户，它们通过 IP 网络相连。客户的 I/O 请求经元数据服务器认证后，给其分配对象 ID 及授权，然后再访问 OST。

OST 是一种智能附网存储设备，它的存储介质可以是磁盘、磁带或者 RAID。OST 负责本地存储管理，主要有以下几种功能：(1) 数据对象的存储和访问，对首次到来的数据对象分配存储空间，对其后的 I/O 请求提供数据访问。(2) 数据管理，在传统存储系统中，数据由文件服务器或者主机操作系统管理；在对象存储系统中，存储系统的大部分数据管理工作由 OST 完成，减少了主机或服务器的开销。(3) 保证安全，OST 通过认证 I/O 请求的授权，保证合法的 I/O 请求访问对象存储设备，拒绝非法和过期 I/O 请求。

对象存储系统中，将部分存储管理的功能下移到 OST，简化了元数据服务器的管理功能，但是，对象存储的安全保证是由元数据服务器提供的，元数据服务器在管理元数据同时进行安全管理。其功能如下：(1) 存储空间的管理，将对象存储设备整合成统一存储空间，满足用户的需要；(2) 用户命名空间的管理，为用户提供能理解的命名空间，并为每个对象分配唯一的 ID；(3) 存储空间的分配和定位，按一定的算法为对象分配 OST，算法必须保证再次访问同一对象时的 OST 地址的唯一性；(4) 安全管理，元数据服务器在响应 I/O 请求前首先必须认证 I/O 请求，保证 I/O 请求由合法用户发出，在此之后，分配元数据和授权。

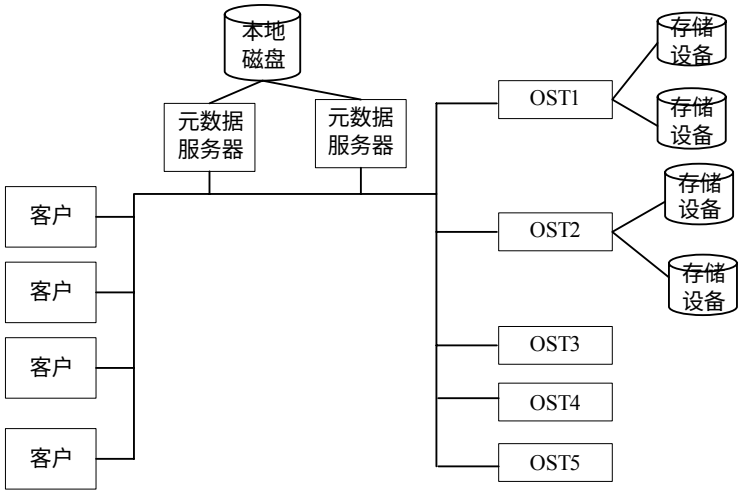


图2.4 对象存储系统体系结构

2.2.2 软件结构

对象存储的相关软件有 PanFS[10]、Centera[3]和 Lustre[82]。其中 Lustre 是由 Cluster File systems 公司开发的开源文件系统，本文在 Lustre 的基础上建立对象存储原型系统，它采用对象存储作为其基本构件，由客户、集群元数据服务器和 OST 三部分组成，三部分的软件结构如下。

在 OST，文件系统负责将对象分配到与其相连的存储设备，并在再次访问时查找和定位对象。来自客户端的 I/O 请求经 iSCSI 解封装后，通过对象认证，保证合法 I/O 请求访问存储设备。认证后的对象经解封装成为文件或数据块，再通过文件系统保存在存储设备上。

元数据服务器通过虚拟磁盘将存储设备统一为单一的存储空间，并为合法用户分配存储空间。客户端发送的 I/O 请求首先通过安全管理认证用户，并为用户分配访问存储空间的权限和信用，存储系统的安全是通过信用来保证的。虚拟存储空间和存储节点间的映射是通过一定的算法来实现的，同时，元数据服务器还可以根据需求实现对存储空间数据分布优化管理，从而达到优化存储系统性能的目的。

客户端软件主要由对象封装层和网络协议层组成部分，文件系统的文件 I/O 请求或数据块 I/O 请求经封装层转换为对象 I/O 请求，并通过网络协议封装后访问存储系统，同样，从存储系统返回的数据对象经封装层解封装，成为用户能识别的文件或

数据块。这里的网络协议层采用 iSCSI 实现，实际上，对象存储不仅仅限于使用 iSCSI。

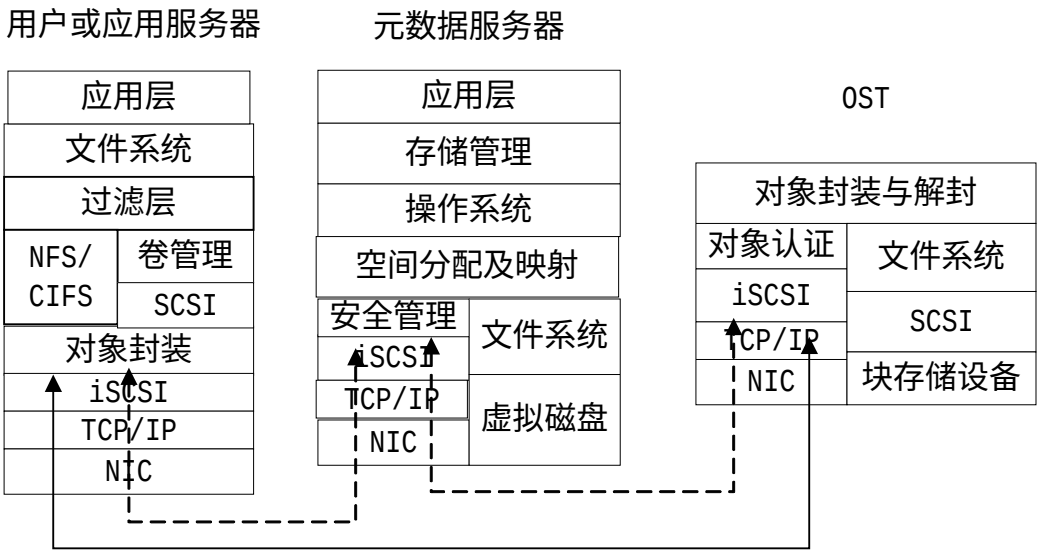


图2.5 对象存储系统的软件结构

2.3 三方传送模式和 NAS 模式

根据对象存储的体系结构，客户端和 OST 间的数据传送可采用三方传送模式和 NAS 模式，它们各有不同的特点。

2.3.1 对象存储中的数据传输模式

在对象存储系统中，客户端、元数据服务器和 OST 通过网络相连，用户对存储系统的访问可分两步：首先访问元数据服务器获取元数据，然后利用元数据访问 OST。用户对存储空间的访问可采用两种模式：NAS 模式和三方传送模式。

采用 NAS 模式时，OST 相当于元数据服务器的外接磁盘。客户端用户对存储系统的访问过程如下：(1) 用户的 I/O 请求经对象封装层封装成对象 I/O 请求送元数据服务器；(2) 元数据服务器认证用户后，查找其元数据，并根据元数据访问 OST；(3) OST 在认证对象 I/O 请求后查找数据对象，并经元数据服务器返回客户端；(4) 客户端将对象解封后响应用户 I/O 请求。

采用三方传送模式访问存储系统时，其过程如下[59]：① 用户需要访问存储系统

时，将对象 I/O 请求发给元数据服务器，以得到相应的元数据等相关信息。② 元数据服务器认证用户后，返回用户相应的元数据信息及信用。元数据描述了所访问的数据对象的属性、存储设备的 IP 地址等信息。信用用来保证数据和存储设备只被合法用户访问。③ 用户取得元数据等相关信息之后，向 OST 发出 I/O 请求（写命令时，被写的数据对象随同 I/O 请求一起发往存储设备）。④OST 认证信用和 I/O 请求后，返回应答信息：读命令时，从磁盘读取数据返回用户；写命令时，数据写进磁盘。⑤ 读写完成后信用被终止：写数据时，写完后 OST 向元数据服务器发送写盘信息，并由其终止用户写信用；读数据时用户的信用超过一定的时间后自动失效，其 I/O 权限被终止，用户再次访问存储空间时需重新申请信用。

两种访问模式各有特点，适合于不同的应用：

（1） 从性能上看，NAS 模式适用于小数据对象的传输，而三方传送模式适用于大数据对象的传输。在 NAS 模式中，元数据的传输只有从元数据服务器到 OST 一次，但数据的传输却从 OST 到元数据服务器再到客户端；而在三方传送模式中，元数据从元数据服务器到客户端再到 OST，中间多了一次元数据的解析过程，但数据直接在客户端和 OST 间传输，少了一次数据中间转发的过程。由于元数据的大小一定，而数据对象的大小变化，所以对于小数据对象而言，采用 NAS 模式减少了元数据的中间解析过程，性能好于三方传送模式，相反，对于大数据对象，尽管三方传送模式多了元数据的解析过程，但数据少了转发过程，三方传送性能好于 NAS 模式。

（2） 存储系统的可扩展性。在三方传送模式中，由于客户直接访问 OST，数据传输不经过元数据服务器，而元数据服务器对元数据处理时间短，这样，减轻了元数据服务器的负担，便于系统的扩展；相反，在 NAS 模式中，数据的传输都要通过元数据服务器，使其成为性能瓶颈，不利于系统的扩展。

（3） 数据的共享程度。存储系统中存在着多个用户共享同一数据对象的问题，采用三方传送模式，数据的共享只能局限在同一客户端的用户，不同客户端的用户之间无法实现共享；采用 NAS 模式，通过在元数据服务器上设置 Cache，可实现多用户的数据对象共享，减少访问 OST 的次数，从而提高 I/O 性能。

由此可见，两种传送模式各有优缺点，存储系统数据传送模式的选择可根据应用的需要进行，也可以采用一定的措施使两种模式混合使用，在大型存储系统中，为便于存储系统的扩展，通常采用三方传送模式，Lustre 正是按这一思路设计的。

2.3.2 两种模式的性能比较

为比较两种传输模式的性能，采用千兆以太网按图 2.4 的体系结构构成基于 iSCSI 的对象存储系统，两台服务器配置都为：Intel P4 1500 CPU，256MB RAM，千兆以太网卡，操作系统采用 RedHat7.1，其中一台作为应用服务器（客户端），另一台作为元数据服务器；OST 控制器采用 PC 机实现，配置为：P4 1500 CPU，256MB RAM，SCSI 卡为 AHA-2940UW，硬盘为 Seagate ST318437LW SCSI 磁盘，其最大传输率为 100MB/S，操作系统为 RedHat7.1。 iSCSI 采用 Intel 开发的 iSCSI-v8-intel 软件包，应用服务器上安装 iSCSI 的命令封装和解封模块，元数据服务器上安装 iSCSI 的启动器软件包，OST 控制器上安装 iSCSI 的目标器软件包。

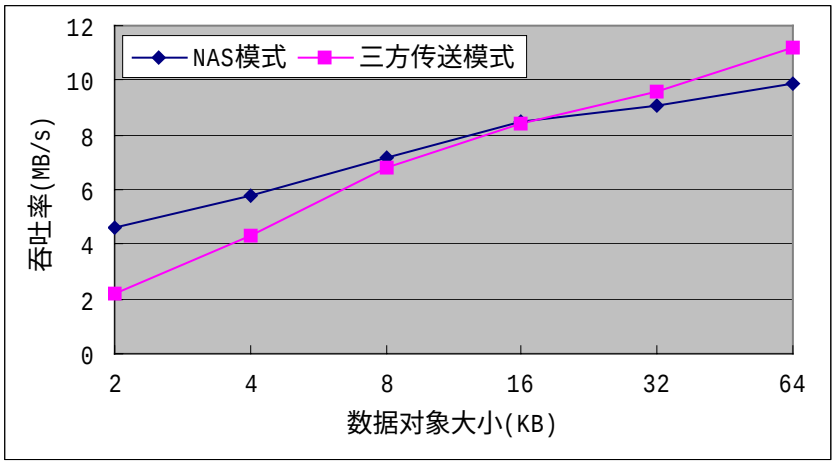


图2.6 NAS模式和三方传送模式的性能比较

NAS 模式是由应用服务器的 I/O 请求访问元数据服务器，元数据服务器获得元数据后直接访问 OST 控制器，返回的数据直接送应用服务器；为实现三方传送模式，对元数据服务器上的 iSCSI 软件进行适当修改，使其返回的是元数据而非数据对象，应用服务器则根据获得的元数据直接访问 OST。图 2.6 是在 Linux 环境下由 iSCSI-v8-intel 软件包自带的工具[60]测得的两种模式的性能比较。从图可以看出，吞吐率随着数据对象的大小增加而增大，这是因为数据对象增大，在磁盘上的数据顺

序读写增多，相应的吞吐率高；而且，对于小数据对象，NAS 模式的吞吐率大于三方传送模式，而大数据对象，三方传送模式的吞吐率大于 NAS 模式，这是因为小数据对象的共享对性能的改善大于数据中间转发对性能的影响，而大数据对象恰好相反。

实际上，实验中并没有考虑到 Cache 的作用。如果在元数据服务器设置 Cache，NAS 模式的性能有所提高；当在客户端设置 Cache 时，两种模式的性能都会提高，但三方传送模式下的性能提高更多，尤其是小数据对象的性能改善更明显。

2.4 基于三方传送的对象存储安全方案

在对象存储系统中，存储网络与用户网络在逻辑上属于不同的网络，但在物理上属于同一网络，存储设备暴露给用户，用户可以不通过服务器直接访问存储设备，为 I/O 性能提高提供了可能，但也带来了新的安全威胁：I/O 请求来自用户而非服务器，存储设备所接收的 I/O 请求不再都是合法安全的；存储网络物理上与用户属于同一网络，数据在传输过程中可能被窃听或修改，数据的保密性和完整性受到威胁；恶意用户可能通过伪造用户信息或 I/O 请求，不断地攻击存储系统，使得合法 I/O 请求无法被及时响应。

如图 2.7 所示，根据对象存储的安全需要，结合三方传送对象存储系统的特点，可以采用信用认证保证存储系统只被合法用户访问：信用是利用共享磁盘密钥生成的，采用密钥生成方案代替密钥管理方案可减少对元数据服务器的性能影响[62]；在客户端加密/解密数据、计算数据检验帧和 HMAC，可以保证数据的安全而不增加存储设备的负担。具体实现如下：

1）磁盘密钥的共享

客户端的用户申请的信用是根据磁盘密钥生成的，磁盘密钥被元数据服务器和 OST 共享。为减少元数据服务器的保存、查找密钥开销，磁盘密钥是根据磁盘信息生成的。元数据服务器有一对公共密钥和私有密钥，OST 在加入存储系统时获得元数据服务器的公共密钥。OST 将其磁盘信息用公共密钥加密后送元数据服务器，元数据服务器用其私有密钥解密后获得磁盘信息，磁盘信息包括 OST 的 IP 地址和磁盘

使用情况等信息，这样元数据服务器和 OST 拥有相同的存储视图。元数据服务器和 OST 采用相同的算法生成磁盘密钥，就实现了磁盘密钥的共享。为保证磁盘信息及时更新，OST 在其后的写 I/O 操作中必须及时通知元数据服务器更新存储视图（读 I/O 不改变磁盘信息）。

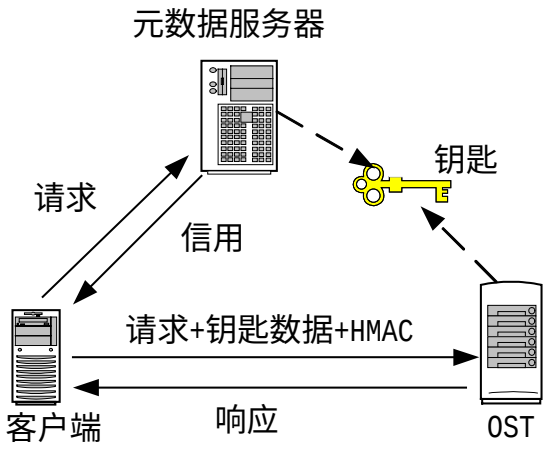


图 2.7 三方传送的对象存储系统安全模型

2）用户认证

用户访问存储系统时，首先必须被元数据服务器认证，在三方传送的对象存储系统中，可采用公共密钥算法实现用户认证。每个用户有一对公共密钥和私有密钥，用户在向元数据服务器申请存储空间时获得元数据服务器的公共密钥。为保证用户请求的唯一性，客户端为每个用户请求分配一个序列号，利用用户私有密钥对用户请求和其序列号签名后，加上用户公共密钥，把它们用元数据服务器的公共密钥加密并送元数据服务器。

元数据服务器收到用户请求后，用自己的私有密钥解密用户请求，然后用收到的用户公共密钥证实签名，验证请求序列号是否过时以及用户 ID 是否配匹。如果所有这些都正确，则用户是合法的，元数据服务器就可以为用户分配信用了。

3）信用的分配

在三方传送的对象存储系统中，采用密钥分配方案代替密钥管理方案生成信用以减少元数据服务器的开销。如前所述，元数据服务器根据磁盘信息生成磁盘密钥，信用是通过 I/O 请求和磁盘密钥生成的。信用由两部分组成：密钥数据和信用密钥。

密钥数据由用户 ID、数据 ID、元数据以及信用生存时间组成；信用密钥是用磁盘密钥对密钥数据散列而成。信用、元数据以及序列号一起通过用户公共密钥加密后返回客户端。元数据为用户提供 OST 地址及访问权限等信息；一个数据对象用一个数据密钥加密，为了减少数据密钥在客户端和元数据服务器间传送，数据密钥随同数据对象一起送 OST 并保存在存储设备；序列号是与用户请求的序列号配匹的。

4）数据的安全性

数据的安全包括两方面的内容：保密性和完整性。由于 OST 的运算能力有限，为减少 OST 的计算开销，数据的加密/解密是在客户端实现的。客户端为每个数据对象随机地分配一个对称数据密钥，写数据时每个数据对象分别用自己的数据密钥加密，一个数据对象的数据密钥随同数据一起保存在 OST 的存储设备上；读数据时用户同时从存储设备端获得数据密钥，然后用数据密钥解密对应的数据对象。

数据的完整性通过数据检验帧和 HMAC 共同实现。数据检验帧用来保证读操作时数据的完整性：加密数据的检验帧随同数据一起送 OST 并保存在那里，读数据时检验帧随数据一起返回客户端，客户端重新计算数据检验帧并与收到的检验帧比较，若相等则收到的数据是完整的。HMAC 是用来保证写操作时数据及 I/O 请求的完整性：用信用密钥对加密数据、数据检验帧、元数据以及密钥数据计算 HMAC（读数据时无加密数据及其检验帧），它们一起送 OST；OST 收到 I/O 请求后，采用元数据服务器相同算法用磁盘密钥对信用密钥数据散列生成信用密钥，用信用密钥采用客户端相同的算法计算 HMAC，若计算的 HMAC 与收到的 HMAC 相等，则用户请求及数据的完整性被验证，用户请求是合法的。若加密数据、数据检验帧、元数据以及密钥数据等中任一个在传输过程中被修改，则收到的 HMAC 和计算的 HMAC 不可能相等，这样，在保证写数据的完整性的同时认证了用户请求及信用。

2.5 对象存储应用方向

对象存储通过丰富的语义，使其兼有 NAS 和 SAN 的优点，同时具有 NAS 系统的跨平台性、安全性和 SAN 系统的性能高效性；而且还可以为用户提供基于服务质量的 I/O 服务。对象存储的应用主要表现在：

将 OST 应用到 NAS 中，就形成了提供直接访问的文件服务系统结构，称为 NAS-OSD，这是一种性能高度可扩展的文件服务系统结构，使用 OSD 后，文件服务器仍继续管理存储设备，不过无需管理数据的服务，从而只是充当文件管理器。Panasas 公司的 NAS 集群基本上是符合这一方向[24]。

将 OSD 应用到 SAN 中，就形成了一种新的 SAN 系统结构，称为 SAN-OSD，它能轻易地进行跨平台操作，从而实现基于 SAN 的分布式存储系统，如：数据库系统可以移植到异构存储环境下，这是因为与系统相关的存储管理功能都下移到设备中。

利用对象的语义传输服务质量要求，可实现基于 QoS 的 I/O[61]，这在 SAN 和 NAS 中是无法实现的。在 SAN 中，存储节点保存的无语义的数据比特，无法传输服务质量要求；而在 NAS 中，尽管传输的文件有一定的属性，但它们都是文件操作属性，同样不能传输服务质量要求。对象存储的属性集中存在未定义部分，通过对属性的修改或扩充，可以达到传输服务质量的要求，从而实现基于 QoS 的 I/O。

对象存储为存储系统的安全提供了有力保障。目前，OSD 安全工作组（OSD-SWG）已经完成了 OSD 设备端的安全模型制定工作。在安全性方面，OSD-SWG 的下一步工作是考虑存储网络各实体（如 OSD 发起端、目标端、元数据服务器）之间的安全性，如系统结构、安全接口、安全实践等；同时制定一套 SNIA 通用的存储网络安全术语和教材[58]。

2.6 本章小结

NAS 实现了异构平台的文件共享，SAN 实现数据块的共享，但它们都存在着这样或那样的缺点，难以满足现代应用对存储的需要。对象存储则通过对象实现 NAS 和 SAN 的统一，同时兼有两者的优点。对象存储中客户端、元数据服务器和 OST 处于同一网络，采用相同的网络传输协议，对象存储的这一特点使得用户访问存储系统可以采用 NAS 模式和三方传送模式，理论分析和实验表明三方传送模式更适合于可扩展的大型存储系统。

3 可扩展对象存储系统的数据分配算法

图灵奖的获得者 Jim Gray 曾经说过：存储系统中真正的成本是数据管理。存储系统的扩展是导致存储管理成本的一个主要原因，这是因为存储系统的扩展导致数据的重新组织，降低了存储系统的可用性。如何保证存储系统的可扩展性和可用性越来越被学术界和工业界所重视。本章通过对已有数据分配算法进行分析，提出一种可扩展对象存储系统的数据分配算法，该算法尽可能减少存储系统扩展时重新分配的数据量，从而提供更高的可用性。

3.1 分布式存储系统数据分配算法

随着计算机应用的发展，数字信息爆炸性增长，存储空间的需求也不断增长。单个磁盘的存储空间有限，通常采用存储系统的方法提供大的存储空间，已有的存储系统有 RAID、NAS、SAN 等。这些存储系统的构造通常是一步到位，也就是说，只有当存储系统构建好后，系统配置相对稳定，存储系统才能为用户提供 I/O 服务。由于存储需求的不断增长，考虑到成本和性能等多方面的要求，存储系统不可能一步到位，必须在使用的过程中不断扩展。存储系统在扩展的过程中，通常停止为用户提供 I/O 服务。由于数据定位算法随着配置的改变而改变，存储系统原有的数据按新的定位算法不一定能查找到。为了保证数据的可靠性，数据必须在存储系统更新时必须重新分配，这使得存储系统更新时间更长，存储系统的可用性受到影响。然而，应用要求存储系统尽可能少地减少服务中断，有些应用甚至要求存储系统提供 24x7 的服务，这就要求存储系统的扩展应尽可能少地影响其正常工作。由此可见，存储系统的可用性和可扩展性间存在着矛盾。如何保证存储系统扩展时提供高可用性，成为存储系统的设计的重要课题，造成这一矛盾的主要原因是当存储系统的配置改变后，数据分配算法也随之改变。所以如何设计出一种数据分配算法，使其适应系统扩展，即存储系统的扩展不影响原有数据的定位，成为存储系统设计者的一个重要课题。

数据分配算法就是在存储系统中给出数据所在的存储设备及存储设备上的位置，它实际上是建立和保持数据到存储设备间的映射关系。常用的数据分配算法有映射表和映射函数两大类，它们各有自己的优点和不足。

映射表采用记录数据和地址间的映射关系，表中每个记录用（数据 ID，地址）表示，保存该表的空间为 O（N），查找成本为 O（log N），其中 N 为存储系统的数据总数。这种方法简单可靠，当存储系统的增加或减少存储设备时，数据分配和查找方法不变，算法的自适应性强，而且，可以通过数据分配原则限制数据迁移，如数据只能从原有存储设备迁移到增加存储设备，不能在原有存储设备间迁移，这样可以降低扩展成本。此外，由于通过映射表可以按存储设备的存储能力分配数据，提高存储系统的可用性。映射表的最主要问题是时间和空间成本随着数据的增加而不断增加，因此，对于小型存储系统是最好的选择，但对于大型存储系统，数据查找成为了性能瓶颈。

一种改进的方法是用 Bloom 过滤器压缩映射表[62]，Bloom 过滤器用在 Summary Cache[77]和 OceanStore。该方法用密钥代替数据 ID 实现数据到存储设备的映射，每个存储设备用一个比特序列给分配到其中的数据密钥编码，编码是通过一组 Hash 函数实现的。如图 3.1 所示为一个存储设备的映射表，开始时比特序列中的所有位初始化为 0。数据密钥 x 通过一组 Hash 函数变换为一组整数，这组整数在比特序列中对应的位设置为 1，查找时用类似的方法，如果数据密钥对应的位都为 1，则该数据在该存储设备中，否则不在该存储设备中。尽管这种方法的空间要求仍然为 O(N)，但常数因子远小于映射表，所以空间开销减小。该方法的一个问题是可能有少量的出错，例如，当一个存储设备的比特序列的所有位都设置为 1 时，按上述方法某数据密钥对应的位为 1，但该数据不一定保存在该存储设备中。为了保证该算法的出错率小，动态调整比特序列，也就是说，当存储设备上的数据增多时，加长比特序列，反之则缩短比特序列，通过这种方法可以保证比特序列的利用比率在某一范围内，从而减少出错的概率。这种方法的数据查找时间为 O(H)，其中 H 是存储设备数，由于存储设备数远远小于数据，其查找成本有所降低。

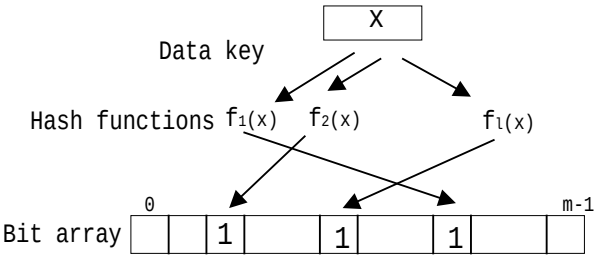


图 3.1 Bloom 过滤器

映射表的另一种改进方法是分区映射表，将存储空间分为多个区，每个区采用一个映射表实现数据及其地址间的映射[63]。这种方法可以采用一个全局的重定向表来决定数据所在的分区映射表。重定向表的大小为分区数（假设为 M），其中每个记录对应一个分区映射表。给定一个数据 ID，将它对重定向表大小取模，用此结果给出分区映射表的首地址，在该表中查找数据所在的存储设备。这样，其空间开销为 O(N/M+M)，数据查找成本为 O(logN-logM)=O(logN)，所以算法成本有所下降，其问题是当存储设备加入和退出时重定向表需要改变。

在 P2P 系统中，采用分布式 Hash 表（DHT）[79][80][81]实现数据到主机间的映射。由于该方案在 WAN 中工作，不可能保持全局的冲定向表的一致性副本，取而代之，采用距离标准来评估密钥和主机间的距离，并指派最近的主机作为该块 ID 的本地主机。每个节点保存一个有少量记录的路由表，通过记录来发送查找请求到最近的主机。路由进程保证最多经过 O(logH)网络跳到给定块 ID 的本地主机，在那里用 O(log N-log H)=O(log N) CPU 周期找到块所有者，其空间要求是 O(N/H)。DHT 的好处是能适应节点的加入和退出。

上述通过映射表实现数据到存储设备的映射的方法称为可控制定位。如果分布控制要求松散，可以直接用函数计算数据的地址，通常，映射函数通过 Hash 函数将数据分配到对应的存储设备[65]。由于这种算法不需要保存映射记录，存储空间开销小，同时数据查询通过一步计算完成，速度快，对大型存储系统有较好的性能。这种算法的最大问题是，当存储系统扩展时，存储系统的存储设备数变化，映射函数必须随之改变，已有数据必须重新分配，这样算法的扩展成本高。针对这一问题有几种改进的方法。

线性 Hash(LH)通过一对 Hash 函数 h_i 和 h_{i+1}（i=0,1,2…）将数据映射到存储设备

中[65][70]，其中 i 是存储设备所在的级。如果数据的密钥用 C 表示，N 表示初始存储设备总数，h_i 可表示为：

$$h_i：C \rightarrow C \bmod N*2^i$$

开始时，i=0， N 个存储设备依次编号为：0，1，…，N-1，数据按上式均匀地布到各存储设备。当存储系统中加入新的存储设备时，如加入 3 个存储设备，其编号为 N，N+1，N+2，这时存储设备 0，1，2 有部分数据分别送 N，N+1，N+2，相当于这三个存储设备各自分裂为两个存储设备，为了保证其映射关系不变，它们的 i=i+1，即由 0 变为 1，而没有分裂的存储设备（3，…，N-1），它们的 i 保持不变。另外，给出指针 n（这里 n=3）表示下一个分裂的起始存储设备，当存储系统中的所有存储设备都分裂后，下一次分裂的又从存储设备 0 开始。这样，在任何时刻，存储存储设备的 i 值只能为连续的两个整数，数据到存储设备间的映射可通过下式实现：

$$a \leftarrow h_i(C)$$

$$\text{if } a<n \text{ then } a \leftarrow h_{i+1}(C)$$

线性 Hash 有很好的自适应性，当存储设备加入或退出时，数据到存储存储设备的映射关系基本不变，且数据的迁移量最少。然而，其不足是数据在存储系统的分布不均匀，存储系统的并行性利用不够。

一致性 Hash 通过两个 Hash 函数 f(b)和 f(i)分别将存储存储设备和数据映射到[0，1]，数据 i 保存到存储存储设备 b 上，使得| f(b)- f(i)|或|1- f(b)- f(i)|最小，也就是说，数据 i 保存在在该区间中离 f(i)最近的存储存储设备[66][67]。当有新的存储存储设备加入时，通过 f(b)映射到[0，1]的某一点，只有该点附近的存储存储设备中的部分数据迁移到该存储存储设备，已有存储存储设备间没有数据迁移；如果某一存储设备删除，保存在该存储设备间的数据迁移到附近的存储设备就可以了。一致性 Hash 本质上和线性 Hash 一样，有很好的自适应性，但数据不能均匀分布。

为保证数据在存储系统中均匀分布，A. Brinkmann 等给出一种在 SAN 中分配数据的方案[69]。算法假定系统中有 n 个容量相等的磁盘，将[0,1]分为 n 个相等的区间，将这些区间分配给 n 个磁盘，使得每个磁盘的区间范围为 1/n。使用一个适当的伪随

机散列函数 h，将数据块映射到[0，1]中的实数，则数据块保存在该点所在的区间对应的磁盘中。算法对于磁盘的增加或减少通过下列原理实现：给定 n 个磁盘，当增加一个磁盘时，切下每个磁盘的区间[i/(n+1),i/n]，i∈{1,…,n}，将这些区间连接成连续的区间[0,….,1/(n+1)]，分配给磁盘 n+1，同时，映射到这些切片的数据块迁移到新的磁盘中；当系统中删除一个磁盘时，将该磁盘对应的区间分为 n-1 片，依次分给其它 n-1 个磁盘，对应的数据发生迁移。这种算法可以较好地解决数据均匀分布问题，然而，当系统中多次加减磁盘时，区间很多，保存区间的空间开销和在区间中查询时间会随之增加。

通过函数映射的代价是数据分布不可控制，因为 Hash 函数没考虑空间的可用性，一定的空闲空间必须预留以保证工作，这浪费空间。而且，在不可控制的数据分布中，存储系统的扩展要求数据的迁移而不是指针。在基于取模的 hash 中几乎所有的数据需要迁移。

综上所述，在分布式存储系统中，数据分配算法分为两大类：可控制的映射表和不可控制的映射函数。映射表的最大问题是其空间和时间开销较大，这在大型存储系统中是不可忍受的；映射函数的时间和空间开销较小，但其存储空间的利用率不高，存储系统的扩展开销较大。

本章通过对已有存储系统的数据分配算法进行比较分析，找出它们的问题所在，然后，建立一个集中控制的面向对象的存储存储系统模型，在此基础上设计一种数据分配算法。该算法具有以下特点：1）算法保证数据在存储设备中按性能均匀分布，充分利用存储设备的并行性，提高存储系统的性能；2）算法的快速自适应性，保证系统扩展后用户仍能访问存储系统中的所有数据，同时，在系统扩展的过程中，存储设备间数据迁移量最少； 3）算法的低时间和空间开销，即花较少的时间和较低的空间开销实现数据分配，使得系统的性能随着系统的扩展而不断提高。

3.2 分布式存储系统模型及相关定义

本节首先给出分布式存储系统模型，在此基础上，给出数据分配算法的几个相关定义。

3.2.1 分布式对象存储系统模型

数据分配算法是建立在一定的系统结构上的，分布式存储系统可分为集中控制和非集中控制两大类，它们各有优点和不足。在集中控制的存储系统中，由于采用集中的服务器对存储设备进行管理，存储设备加入和删除能及时通知给服务器，便于存储系统的管理，同时，用户通过服务器的授权才能访问存储设备，保证了存储系统的安全。然而，由于所有的 I/O 服务都要通过服务器，服务器成为存储系统的性能瓶颈，而且，一旦服务器失效，整个存储系统就崩溃。在非集中控制的存储系统中，没有集中的服务器，消除了系统的性能瓶颈，但也带来了新的问题。由于没有集中的管理服务器，所有的用户都必须有存储视图，这需要大量的空间开销，同时，由于客户的暂时离开，客户可能没有存储系统的更新视图，这样，客户按过时视图就不能访问存储系统或访问错误的数据。而且，没有集中的服务器，存储系统的安全也存在问题。

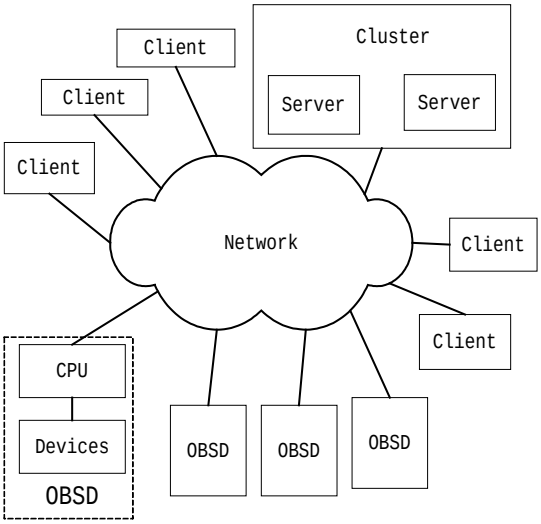


图 3.2 对象存储系统模型

如图 3.2 所示为集中控制的对象存储系统模型，由对象存储设备（OBSD）和集群元数据服务器组成，它们之间通过高速局域网相连。采用对象存储有如下优点：

1）通过对象统一两种 I/O 服务。应用通常需要存储系统提供两种 I/O 服务：文件 I/O 和块 I/O。对象存储系统中的数据——对象，是有一定语义的数据单位，通过语义的定义，它既可以是数据又可以是文件，这样通过对象就可以为应用同时提供

两种文件服务。

2）数据对象不同于数据块和文件，它不仅包含数据，还包含丰富的元数据，存储系统的数据管理是通过元数据实现的。通过元数据实现可控制的数据访问，提供存储系统的数据安全访问，拒绝非法用户的访问。

3）对象存储系统将文件系统的设备管理部分下放到对象存储设备，这样，存储设备能对该节点的数据进行分配存储空间，实现本地数据的访问和删除等操作，从而简化了元数据服务器的管理，消除 I/O 服务的性能瓶颈。

客户对对象存储设备的访问采用三方传送模式：客户首先从服务器获取对象的元数据，其中包括存储设备的 IP 地址和授权，然后直接访问存储设备。三方传送模式一方面可以减少元数据服务器的负担，另一方面可以通过一定的授权协议提供存储系统的安全保障[82]。对象存储设备是能实现自管理的存储节点，每个存储节点可以连接一个或多个磁盘，从服务器或用户的观点来看，一个节点相当于一个存储设备。存储节点有自己的 CPU 和文件系统，具有简单的计算和存储管理功能，能对用户进行安全检查和为对象分配存储空间，可以将对象保存在它能发现的任何地方。对于元数据服务器而言，存储节点功能对称，也就是说，所有的存储节点都具有相同的管理和存储功能，这样可以简化服务器的存储管理，便于存储系统的扩展。这里功能对称并不是指体系结构统一，节点可以有不同的结构、性能和存储容量。对象不同于数据块，它大小可变，还包含一定的语义，便于存储管理；同时对象也不同于文件，一个对象可能包含多个小文件，而一个大文件可能由多个对象组成。服务器为用户提供可理解的视图，将所有的存储设备虚拟为统一的存储空间——逻辑空间。服务器的数据分配方案就是实现逻辑空间到存储节点的映射。服务器采用集群的方式，可以提高其性能和避免出现单点失效。为实现对象的分配，服务器随机为每个对象分配一个密钥，不同的对象的密钥可以相同，这是因为存储系统中的每个对象必须有一个唯一的标识 ID，密钥只是用来确定对象所在的存储节点。密钥相同的对象分配到同一节点，一旦按分配算法确定了对象所在的存储节点，就在该节点中按对象 ID 查找对象。

3.2.2 存储模型的相关定义

为了设计出存储系统的数据分配算法，对存储系统模型给定相关假设。

定义 1：一个存储节点是一个连接到网络的能实现自管理的存储设备，能够响应用户的数据请求。在本模型中，存储节点对应于 OSD 磁盘，但是下述算法不仅仅适用于 OSD。

定义 2：存储节点组是一组存储节点，在系统中存储节点的加入、退出、以及重新分配权值是以组为单位进行的。

为了充分利用存储节点的并行性，提高存储系统的性能，对象应尽可能在存储节点间均匀分配。当存储节点性能相同时，对象平均分配可以提高系统性能，但是，当存储节点的性能不同时，对象平均分配可能导致性能低的存储节点负担重，而性能高的节点负担过轻。这里给出一种分配算法最优的定义。

定义 3：数据对象均匀分配是指存储系统中分配给每个存储节点的数据对象数与存储节点的权值成正比，也称为负载均衡分配。

在存储系统中，为了表示存储节点的性能不同，按存储节点性能给其分配权值，也就是说，存储节点的权值大小表示其性能的好坏，这里的性能既可以是存储容量，也可以是 I/O 速度或网络带宽，还可以是他们的综合。对象按权值分布的目的是提高存储系统的利用率和性能，当权值用来表示存储节点容量时，按权值分布意味着容量大的存储节点分配的数据对象多，保证所有存储节点充分利用；当权值用来表示存储节点的 I/O 速度时，对象按权值分布保证充分利用存储节点的性能，提高平均 I/O 速度。

由于存储系统的不断扩展，存储系统的性能不断变化，为了保证存储系统的数据实现负载均衡分配，每个存储节点分配的对象数也随着系统的扩展而变化，所以，系统扩展时存储节点间存在着对象迁移。

定义 4：数据重新组织是为了保证存储系统节点间数据的均匀分配，在存储系统扩展时发生在存储节点之间的数据对象迁移，这样，存储系统数据重新组织前后，数据在存储节点间都然均匀分布。

在进行数据的重新组织时，我们希望系统迁移的数据对象越少越好，这样可以减

少网络通信量或者使得离线重新组织的时间最小。根据这些要求，可以得到最优重新组织的定义。

定义 5：如果存储系统扩展时迁移的的数据对象最少，则称这种重新组织的方法是最优的。

根据最优重新组织的定义，可有下述引理：

引理 1：当增加一组新的存储节点而系统中原有存储节点的权值保持不变时，系统必须重新组织。在重新组织阶段，当并且仅当要迁移的数据对象是从原有的存储节点组迁移到新的存储节点组中时，重新组织是最优的。

证明：假设该重新组织是最优的，但是还存在一些数据对象需要在重新组织阶段在原有存储节点间迁移。假设存储节点 s_i 和 s_j 为原有节点，由于系统扩展时原有存储节点的权值没有发生改变，为了保持按权值分配对象，对于每个从 s_i 迁移到 s_j 的对象，我们必须满足下面的条件之一：

1、将其它的对象从 s_k 迁移到 s_i 中；

2、将其它的对象从 s_j 迁移到 s_k 中；

3、既将其它的对象从 s_k 迁移到 s_i 中，又将其它的对象从 s_j 迁移到 s_k 中。

在第一种和第三种情况下，从 s_k 迁移到 s_i 的对象中的某一个对象可以直接从 s_k 迁移到 s_j 中，因此没有必要将对象从 s_i 迁移到 s_j 中。这就和最优重新组织的假设矛盾了，我们可以减少至少一个对象的迁移并且仍然可以保持对象按权值分配。

同样地，在第二种和第三种情况下，从 s_j 迁移到 s_k 的对象中的某一个对象可以直接从 s_i 迁移到 s_k 中，因此没有必要将对象从 s_i 迁移到 s_j 中。这也和最优重新组织的假设矛盾了，

因此，如果一个重新组织是最优的并且满足引理 1 中的标准，则在重新组织阶段要迁移的所有的数据对象应该迁移到新加入的存储节点中。

下面将证明：如果重新组织满足引理 1 中的标准，并且在重新组织阶段所有要迁移的数据对象是从原有的存储节点迁移到新加入的存储节点，则这种重新组织是最优的。

假设上述的假设存在矛盾，也就是说，如果重新组织满足引理 1 中的标准，并且

在重新组织阶段所有要迁移的数据对象是从原有的存储节点迁移到新加入的存储节点的，但这种重新组织还不是最优的。

由于在重新组织阶段，所有迁移的数据对象是从原有的存储节点到新加入的存储节点，这样，某个给定的存储节点不可能同时“失去”对象和“获得”对象。换句话说，设 L 是所有的对象要移出的存储节点的集合，G 是所有的对象要移入的存储节点的集合；则 L 和 G 的交集必须为空，即 $L \cap G = \Phi$。

由于交集为空，对每个要迁移的对象而言，在存储节点 $S_L \in L$ 中的对象的个数在减少，而在存储节点 $S_G \in G$ 中的对象的个数在增加。

如果重新组织不是最优的，则一定存在某些不必要的对象迁移，去掉这些不必要的迁移后，系统按存储节点权值分配对象。如果考虑到这些不必要迁移，L 中的对象数与 G 中的对象数的比值不等于集合 L 和 G 的权值比，也就是说，存储系统中没有按节点的权值分配对象，这与重组的目的——保证对象按权值分配相矛盾。

所以，如果一个重新组织必须满足引理 1 中的标准，并且所有在重新组织阶段要迁移的数据对象是从原有的存储节点族中迁移到新加入的存储节点族中的，则这种重新组织是最优的。

存储系统的数据分配算法设计就是要保证系统中的数据均匀分布，并且在系统扩展时系统重新组织最优。

3.3 可扩展的存储系统数据分配算法

3.3.1 数据分配算法的设计原则

为了优化存储系统，在数据分配算法的设计中，必须考虑以下因素：1）数据分配原则，即按照什么原则将数据分配到存储设备，分配算法一般按照存储系统的某方面性能分配数据，达到优化存储系统的性能的目的。下面的算法按存储节点的权值分配数据对象，当存储节点性能相同时数据对象均匀分布在所有存储节点中，利用了存储设备的并行性，达到提高存储系统的 I/O 速度目的。2）算法的实现成本，包括时间和空间成本，在大型存储系统中，由于存储空间大，保存的数据多，映射信息的空间开销和数据查找时间开销不可忽略，所以，数据分配算法设计应尽可能地降低算法的时间开销和空间开销。3）分配算法的自适应性，必须保证无论存储系

统如何扩展，都能通过该算法访问保存在存储系统中的所有数据。 4）存储系统的扩展成本，存储系统扩展时，为了保证算法的自适应性，存储系统的数据必须重新组织。数据重新组织算法与数据分配算法相关，必须保证数据重新组织算法最优。

从前面所述可知，映射表可靠，数据可以均匀分布，且当存储系统扩展时成本较低，但其空间和时间开销随着系统的扩展而增大；映射函数时间和空间开销小，数据可以均匀分布，但系统扩展时数据均匀分布和数据迁移量间存在着矛盾，数据均匀分布可能导致大量的数据迁移，而减少数据迁移又可能导致数据分布不均匀。下面从一种集中控制的存储系统模型出发，根据数据分配算法的设计原则，设计出一种算法。

3.3.2 存储系统数据分配算法

考虑到存储系统的不断扩展，假定每次扩展时的存储节点为一组，每组的存储节点的性能很容易保证相同。由于存储系统的扩展在不同的时间进行，不同组的存储节点的性能不一定相同。为了适应系统扩展时的重新组织算法，假设存储系统的存储节点是以组为单位组织的，第 i 组包含 m_i 个节点，如果存储系统有 l 组存储节点，则总节点数为 $M(j)=\sum_{i=1}^{l} m_i$ ，式中 j 表示存储系统扩展的次数，j=0 表示存储系统首次构建，j=J 是存储系统的最后一次扩展。由于存储系统采用服务器集中控制，存储系统的扩展更新信息都会通知给服务器，对于对象分配算法而言，节点组数、每组包含的节点数以及存储系统的扩展信息都是确定的。

为了简化算法叙述，首先假设存储节点是同构的，即有相同的性能。由于 Hash 函数可以降低算法的时间和空间开销，算法选择一个基于对象密钥的伪随机整数函数 $z_x=f(x)$，使得 $0\le z_x<M(j)$，将对象映射到某一节点组中，然后在组中均匀分配。

```
Call(x)
{ take the j from the object key x
while(object is not mapped)
   {seed a random number generator with x
   generate a random number 0≤z<M(j)
   select the m_i which z is mapped to
   y= z mod m_i
   return the address of device y in set i}}
```

图 3.3 数据对象到存储节点的映射算法

图 3.3 是基本的数据分配算法，服务器在给对象分配存储节点前，首先为其分配一个密钥 x。由于存储系统更新时 Hash 函数要随之变化，为了记录对象分配时存储系统的更新状态，密钥中包括存储系统当时更新的次数。当用户再次访问该对象时，就从 x 中获取 j，这时 j 就不一定等于 J, 它可能是[0，1，2，…，J]中的任一数，根据更新次数可以从更新信息表中找到对象分配时的节点总数，这样数据查找采用数据分配相同的算法，就可以保证数据可靠地查找。算法获得对象的密钥后，将对象密钥作为随机数生成器的种子，使其生成在区间[0，M(j)]内的随机整数 z。将对象映射到随机整数 z 后，不是直接将对象保存在第 z 个存储节点，而是映射到该存储节点所在的组，在组内再通过 Hash 函数均匀分布，这样做的目的是，将存储节点分组管理，便于存储节点的加入和退出时数据的分布管理。实际上，如果组容量为 1，算法就直接实现在存储节点间的数据对象分布。

按照上述算法，对象在存储节点间均匀分布。存储系统构成后，在没有扩展前，总节点数为 M(0)。由于随机数生成器生成的随机整数在区间[0，M(0)]内是均匀分布的[74][75]，这样节点组 i 中分配的对象数为 $N \cdot \frac{m_i}{M(0)}$。由于组 i 中有 m_i 个存储节点，所以分配到每个节点的数据对象数为 $N \cdot \frac{m_i}{M(0)} \cdot \frac{1}{m_i} = \frac{N}{M(0)}$，这样，所有存储节点中分配的对象数相等，保证了对象在存储节点间均匀分布。后面将叙述到，通过适当的对象迁移，可以保证存储系统扩展后数据仍然均匀分布。

3.3.3 数据分配算法的时间和空间开销

由于每个数据对象有一个密钥，因此算法的空间开销是 O(N+M)，其中，O(N)表示由于数据对象的密钥引起的空间开销，O(M)表示保存存储节点 IP 地址引起的开销。密钥由存储系统更新次数和随机数组成，由于不同的存储对象可以有相同的密钥，该随机数小于存储节点的 IP 地址，同时，存储系统更新不可能频繁，更新次数不可能很多。所以， 密钥远小于 IP 地址，算法的空间开销小于映射表的空间开销 O(N)。实际上，如果将数据对象 ID 作为密钥，就可以去掉额外的密钥开销，空间开销减少为 O(M)，它远小于映射表的空间开销。

算法的时间开销由随机数生成开销、确定数据对象所在组的开销和其它算术操

作开销组成。通过种子生成随机数的时间是常数[76]，即时间开销为 O(1)。如果组中包含的节点数相等，确定数据对象所在组可以通过 Hash 函数实现。实际的组大小不可能相等，而且，考虑到新加入的存储节点的性能高于已有存储节点，即使组大小相等，其加权大小也不相等，所以不能用 Hash 函数确定对象所在的组。这时，组的确定采用折半查找法，即将组按节点序号排序，将序号分为两个子区间，确定数据对象所在的区间后，再将该区间分为两个子区间，依此类推，直到找到数据对象所在的组。按照这种方法，其时间开销为 O(log l)，其中 l 为存储系统的包含的组数。算术操作开销为 O(1)，这样，数据对象的查找时间开销为 O(1+1+log l)=O(log l)，它与映射表的时间开销 O(log N)相比，由于 l«N，所以时间开销要小得多。算法的时间开销与纯 Hash 函数的时间开销 O(1)相比，略有增加，这是由于节点分组造成的，然而分组便于存储系统的扩展管理。

3.3.4 数据重新组织算法

当存储系统中有存储节点加入或退出时，为了保证数据对象仍按性能均匀分布，存储系统中的数据必须重新组织，存储节点间必须有数据对象迁移。如引理所述，当只有数据对象从原有存储节点迁移到新增存储节点，才能保证数据迁移量最少，从而提高存储系统的可用率。当存储系统中增加存储节点时，存储系统的总节点数增加，为了保证存储系统按性能均匀分布数据对象，已有存储节点中的部分数据对象必须迁移到新增存储节点。但是，已有存储节点中的数据对象是同比减少的，它们之间没有必要进行数据重新分配，所以，为了保证存储系统的重组算法最优，不应出现已有存储节点间的数据迁移，只能是数据从已有存储节点向新增存储节点迁移。当存储系统中的某些存储节点由于寿命或其它原因必须退出时，退出前，它们中的数据应尽可能地迁移到剩余存储节点中。同理，数据对象只能从退出节点迁移到剩余节点，而不能在剩余节点间迁移。所以，数据对象只在增加（或删除）节点与其它节点间迁移，就能保证存储系统重组算法最优。图 3.4 所示的数据重组算法就是按这一思路设计的，InsertDevice(m_{l+1})用来当存储系统中增加新的存储节点组时的数据重新组织，通过该算法将已有存储节点中的部分数据迁移到新增存储节点，保证增加存储节点后数据仍均匀分布；DeleteDevice(m_k)用作存储节点组即将退出时将其中的数据对象迁移到剩余存储节点。当然，为了尽可能降低存储系统的可用性，存储系统的重组算法必须利用存储系统的空闲带宽来进行，下一章就讨论数据迁移

算法的实现。

存储系统的重组算法可以保证系统扩展后数据仍然按存储节点性能均匀分布，下面分析之。系统扩展前，每个存储节点分配的数据对象数为 N/M(J)，当存储系统中加入一组存储节点时，已有存储节点中迁移的数据对象数为 $N \cdot \frac{M(J+1)-M(J)}{M(J)M(J+1)}$，这样，数据迁移后原有存储节点剩余的对象数为：$\frac{N}{M(J)} - N \cdot \frac{M(J+1)-M(J)}{M(J)M(J+1)} = \frac{N}{M(J+1)}$。迁移的数据对象总数为：$N \cdot \frac{M(J+1)-M(J)}{M(J)M(J+1)} \cdot M(J) = N \cdot \frac{M(J+1)-M(J)}{M(J+1)}$，由于新增节点组有 m_{l+1} 个存储节点，由算法有：$M(J+1)=M(J)+m_{l+1}$,所以，分配给每个存储节点的对象数为：$N \cdot \frac{M(J+1)-M(J)}{M(J+1)} \cdot \frac{1}{m_{l+1}} = \frac{N}{M(J+1)}$。可见，系统扩展后，算法保证数据对象在系统中的所有节点间均匀分布。

```
InsertDevice(m_{l+1})
{M(J+1)=M(J)+ m_{l+1}
   For a=1 to J
      For b=1 to m_b
        For c=1 to
        Select randomly a data object and take its key x
        Replace the part of x correlating to M with M(J+1)
        Seed a random number generator with x
        Generate a random number 0≤z<M(J+1)
         y= z mod m_{l+1}
        Move the data object with x to device y in set l+1
        Next c, b, a }

DeleteDevice(m_k)
{ M(J+1)=M(J)- m_k
   For a=1 to m_k.
      While( device a is not empty)
   {Select randomly a data object and take its key x
        Replace the part of x correlating to M with M(J+1)
         seed a random number generator with x
         generate a random number 0≤z<M(J+1)
         select the m_i which z is mapped to
         y= z mod m_i
         Move the data object with x to device y in set i}
         Next a}
```

图 3.4 存储系统扩展时数据重组算法

当系统要删除一组存储节点时，必须将该组存储节点中的所有数据对象迁移到其它存储节点。由于数据迁移前各存储节点的数据对象数相等，而数据重组算法类

似数据分配算法，它将删除节点中的数据对象均匀分配到剩余存储节点，所以，数据迁移后剩余存储节点中的数据对象数也相等，也就是说，数据对象在剩余存储节点中均匀分布。

3.3.5 异构存储节点的数据分配和重组算法

存储节点的异构是指存储节点的性能不同，如速度、容量、带宽等的不同。从功能上讲，存储节点仍然是相同的，因为所有的存储节点仍然具有自管理和分配本节点的数据对象的功能。为了充分利用存储节点的并行性，数据对象按存储节点性能分配，这通过存储节点的性能加权来实现。由于存储节点是以组为单位加入或退出的，很容易保证它们的性能相同，所以节点组为单位进行性能加权。这样，存储系统在第 j 次扩展后，其总节点数为 $M(j)=\sum_{i=1}^{l} w_i \cdot m_i$ ，其中 w_i 为第 i 组的性能加权。由于存储节点必须为整数，权值的选择可通过性能比取整实现，即 $w_i=\left\lceil \frac{p_i}{p_0} \right\rceil$ ，其中，p_0 是最低的组性能，p_i 是第 i 组的性能。由于组内存储节点的性能相同，组内数据对象直接均匀分配。所以加权分配只是在确定数据对象所在的组时使用，除此之外，前面所述算法仍然适用。

3.4 本章小结

在集中控制的可扩展存储系统，由于存储系统的扩展信息及时通知给服务器，采用密钥记录数据分配时的存储状态，使得不同存储状态时保存的数据对象按照当时的 Hash 函数进行查找，保证了存储系统中的数据对象的可靠查找；算法采用数据对象密钥作为随机数生成器的参数，将数据对象均匀映射到给定区间，从而保证数据在存储节点中按性能均匀分布；通过适当的重组算法保证数据对象只能在增加（或删除）的存储节点和其它节点间迁移，使得存储系统扩展时在保证数据仍按性能均匀分布的同时，扩展成本最低；由于算法采用类似 Hash 函数实现数据对象的分配，使得算法的时间和空间开销大大减小。

4 基于 QoS 的数据迁移模型的设计

随着存储应用的发展，应用对存储系统的 I/O 性能要求越来越高，这就要求存储系统为应用提供基于 QoS 的 I/O 访问，对象存储的出现为基于 QoS 的存储系统的实现提供了可能。如前所述，存储系统在扩展和性能优化时，存在着大量的数据迁移，实际上，数据迁移还用在其它方面，如设备出错、部件升级、系统的进化、数据库重组等[83][84]。然而数据的迁移对存储客户的 I/O 性能必然产生或多或少的影响，基于 QoS 的数据迁移就是为了尽可能减少其对 I/O 性能的影响。

4.1 基于 QoS 的 I/O 模型

在传统的磁盘存储系统中，数据以块的形式保存在磁盘中，该系统只是保存文件的比特，而没有描述数据的更多属性。这样存储设备端无法知道 I/O 的 QoS 属性。随着磁盘系统越来越大，其连接的用户越来越多，这种基于块的磁盘存储系统采用尽最大努力响应 I/O 请求机制，也就无法满足应用基于服务级别需求的需要。

对象存储协议是 SCSI 命令协议的扩展, 其协议试图将数据以可变长的对象而非固定长的块保存起来，而且通过命令的扩展给对象数据定义有许多属性，改变了传统的基于块的存储中数据没有任何属性这一特点。 对象存储设备将部分或整个文件表示为有属性的数据对象，这些属性用来描述对象的各种约束。对象和文件并非一对一，一个文件可能包含多个对象，相反，也可将多个文件合成一个对象。 数据以对象形式保存，数据更多的细节，如安全属性、服务质量需求等可通过其属性在传输时同时传输。这样在数据传送给对象存储设备的同时，也提供 I/O 操作服务质量等信息。所以采用对象存储协议，就可以实现基于 QoS 的存储系统。

4.1.1 OSD 协议

在 2004.5，SNIA 发布了对象存储协议规范 V9，该协议定义了在 SCSI 启动端和目标端的通信以实现远程互操作。按照该规范，Intel 研究中心设计出其参考实现（OSDR9）并测试了 iSCSI 和 OSD，该实现包含基本 OSD 功能和 iSCSI 通信机制[87]。

在 Linux 操作系统及其文件系统下，OSDR9 提供了模拟环境来测试 OSD，该参考实现开发了一个 OSD 目标端软件，它用 Linux 文件系统例程的文件描述对象。当一个对象产生时，其文件在 Linux 文件系统下产生，该对象的相关属性通过附加文件来模拟，该文件有一个类似命名方案用来帮助组织属性查找，因此，OSD 目标端的前端采用基本级 OSD 协议，其后端将这些基本的对象转换成标准的 Linux 文件系统函数调用，如 oper()，read()。

表 4.1 OSDR9 强制执行的命令

Command Name	Added/Updated
Append	Added
Create	Updated
Create write	Added
Create Partition	Updated
Flush Object	Added
Format OSD	Added
Get Attributes	Updated
List	Added
Perform SCSI Command	Added
Perform Task MGMT Function	Added
Read	Updated
Remove	Updated
Remove Partition	Updated
Set Attributes	Updated
Set Key	Added
Set Master Key	Added
Write	Updated

表 4.1 列出了 OSDR9 中增加的 OSD 命令[88]，表中命令是 OSDR9 强制执行的命令全集。由于它们是由 OSDR9 强制执行的，所以它们都实现了，这些命令对 OSD 测试非常有用。存储对象属性，特别是用来传输 QoS 的属性，对参考实现来说是新增加的。OSDR9 提供对象的保存、处理访问等属性，所有属性有与其相联系的属性

号。有些属性与对象一起保存，它们由 OSD 规范定义，其它的属性没有给定特定的功能，保存在磁盘或主存，或以其它形式使用，这些属性可用在实现 QoS 时启动端的通信参数描述。

OSD 在运行环境下的属性对传输 QoS 属性非常有用，这些属性的保存没有意义，对基于块的磁盘也没有意义，因为它们是对实时信息的描述，但对传输 QoS 信息非常关键。通过属性机制，预定读或更改属性的操作，以获得或设置属性。获得或设置属性既可能伴随着其它命令，如产生对象、格式化 OSD 等，也可能作为一个单独命令，而不作为其它命令的附加功能。当属性获得或设置伴随着其它命令时，其次序非常重要，如获得属性在产生命令前就产生错误，这是因为对象还没有产生。GET 和 SET 可能同时执行，它们用来获得属性和设置属性。正确的顺序是获得请求的属性发生在设置命令执行前，这种操作不同于规范中的许多命令，它们首先执行 SET 操作，再执行 GET 操作。

通过设置属性和获得属性命令，可以传送数据对象的服务质量要求，这样，在存储系统的启动端和目标端间，就可以实现基于 QoS 的 I/O 访问了。

4.1.2 基于 QoS 的 I/O 响应调度模型

在存储系统中，有多个存储设备和多个客户，客户端对存储设备的访问是多对多的关系。由于网络的带宽是不断地增加的，可以认为网络带宽能够满足应用的要求，这里不予考虑。由于机械速度的限制，存储设备的 I/O 速度有限，它无法满足应用的要求，可能成为存储系统的瓶颈。尽管对象存储中采用多存储设备的并行性来提高 I/O 带宽，由于数据的分布不可能理想化，必然存在着多个用户同时访问同一存储设备的情况，而不同的用户的不同应用对 I/O 服务的质量有不同的要求。如图 4.1 所示，基于 QoS 的存储就是要解决同一存储设备如何响应不同服务质量要求的 I/O 请求问题。

QoS 要求是启动端要求的服务质量，它可以转换为对 OSD 目标端的带宽要求和缓冲大小。带宽是指请求要求分配的 I/O 带宽，缓冲大小是指一个 I/O 请求能传输的最大数据量。这两种属性，传输到 OSD 目标端，用来传递 I/O 请求的吞吐率要求。为了满足用户的 QoS 要求，在目标端和启动端必须有一定的机制来实现。现有的 OSD

协议通过命令来执行请求响应模型[88]，也就是说，用户只有在第一个请求被响应后，才发出新的请求；同样，目标端只有在响应同一用户的前一请求后才能响应其后的请求。当然，对于不同的用户的请求，目标端可以按一定的顺序响应。基于 QoS 的存储就是要解决不同用户 I/O 请求的响应顺序问题。

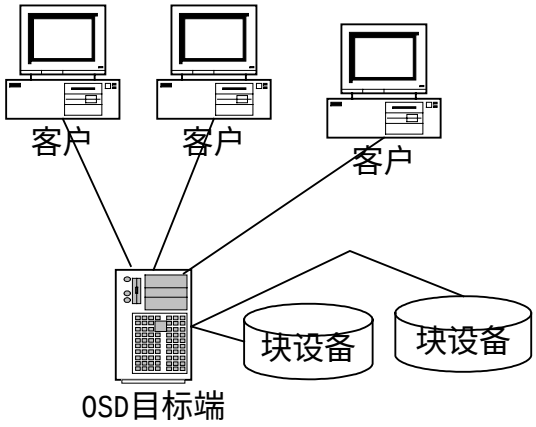


图 4.1 基于 QoS 的对象存储系统体系结构

通过 QoS 属性的传输，启动端和目标端就决定启动端隔多久发送一个请求以保持一定的带宽。一个 I/O 请求的响应延迟可通过 OSD 目标端分配的带宽和缓冲属性来决定的，满足(4-1)：

$$T = \frac{S}{C} \tag{4-1}$$

其中，S 和 C 分别表示分配给请求的缓冲区和带宽大小。例如：应用要求 20MB/s 的链路，由于缓冲大小有限，不可能在一次缓冲中读所有的数据，所以将它分为多次读。假定缓冲大小为 10000B。这意味着启动端在一秒内发送 2000 个读请求。注意，带宽和缓冲大小由客户端的应用设置决定的，所有这些属性，必须在数据传输前传输到 OSD 目标端。

在 OSD 目标端，存在着不同的用户的不同应用的 I/O 请求，它们有不同的服务质量要求。当目标端的 I/O 带宽能满足所有 I/O 请求时，就能按需求分配 I/O 带宽。然而，存储系统中存在对某一存储设备的集中访问，这时，存储设备的带宽可能不能满足用户的要求，如何分配带宽就成为问题。

为了解决带宽分配问题，根据应用的服务质量要求，给每个 I/O 请求分配一个收

益函数，如式(4-2)所示：

$$R(\delta)=\begin{cases}R & t\le\delta\\ 0 & otherwise\end{cases} \tag{4-2}$$

其中 δ 表示 I/O 请求的最大响应时间，它是根据应用的响应时间要求来给定的。当 I/O 请求在 δ 内完成，存储系统获得收益，否则，收益为 0。收益 R 不仅与 I/O 请求的对象大小有关，还与其对应的应用类型有关。显然，对象越大，I/O 请求的响应对应用的影响越大，收益 R 就越大。存储系统为了比较不同的应用的 I/O 请求的收益，收益函数应该有相同的标准，如第二章中对象存储系统的体系结构所述，客户端在访问存储系统前，必须先从元数据服务器中获得元数据，这样，收益函数就在元数据服务器中集中分配，所以，所有的收益值都是按相同的标准分配的，这样，不同的应用的收益就有了可比性。

基于 QoS 的存储系统，就是要及时响应重要的 I/O 请求，适当放弃一些次要的 I/O 请求[89][90]。为此，在 OSD 的目标端，请求响应的排序必须保证满足式(4-3)：

$$\max_{Tm}\left(\sum R_i(\delta_i)\right) \tag{4-3}$$

其中，T_m 是 OSD 目标端所有 I/O 请求中最大响应时间。也就是说，在最大响应时间内，按照收益大小对 I/O 请求分配带宽和缓冲，保证收益大的请求获得所要求的带宽和缓冲，当带宽不能满足所有 I/O 请求要求时，对收益小的 I/O 请求推迟响应或拒绝响应，只有这样才能保证 OSD 目标端收益最大。由于存储节点彼此独立，当所有存储节点的收益最大，存储系统就取得尽可能高的服务质量。值得注意的是，当有新的 I/O 请求到来时，已分配的带宽可能发生改变，这是因为，如果新到的 I/O 请求的收益大于已有的 I/O 请求的收益，就可能从收益小的 I/O 请求中剥夺带宽来响应新的 I/O 请求，保证式（4-3）随着时间推移而成立。

4.2 基于 QoS 数据迁移调度算法

如前所述，存储系统中存在着大量的数据迁移，数据迁移或多或少地影响着存储系统的 I/O 性能。然而，当今 IT 环境要求系统提供 24×7 的服务，因此数据迁移应尽可能少地减少对客户 I/O 性能的影响。数据迁移有粗粒度和细粒度之分，粗粒度数

据迁移要么采用低于 I/O 请求的优先级，要么采用高于 I/O 请求的优先级而不管其对客户的访问的影响，在迁移的过程中，客户 I/O 将被拒绝[91]。当数据以细粒度迁移时，存储系统可以提供对其他数据的在线访问，这样，迁移的实现可以减少对客户访问的不利影响，提供 24*7 的服务。这里提出一种迁移方案，它平衡迁移目标和客户 I/O 性能要求，提供一种灵活的数据迁移调度方法，把管理者从数据迁移调度中解放出来，达到既不中断客户的访问，又满足迁移目标。此外，该算法还通过减少客户同时访问的影响来更有效地实现数据逻辑重组。

4.2.1 基于 QoS 的迁移思想

如前所述，基于 QoS 的存储系统中，每个 I/O 请求有一个与之相联系的收益值，该收益值是 I/O 请求在其最终期限内完成时系统自动产生的。一个迁移任务是指一批数据从一些存储节点迁移到其它存储节点。例如，一组数据的迁移开始于 T_0，它迁移 B_m 比特的数据从容量为 C 的磁盘到其它磁盘。实际上，一个迁移任务完成后，存储系统的性能得到了优化，存储系统获得附加收益。例如，在存储系统扩展时，当数据迁移完成后，数据分布到更多的存储节点，利用存储节点的并行性，存储系统的 I/O 速度得以提高。不同的迁移任务在完成后有不同的附加收益，根据迁移任务的相对重要性，系统可以分配不同的收益值。所以，迁移调度就是发现在任意时刻 t 的迁移率 $M_r(t)$和迁移完成时间 T_M，使总收益最大[93]。形式上如式(4-4)所示：

$$max_{M_r(t),T_m}[\sum R_i(\delta_i)+R_m(T_m)] \quad (4\text{-}4)$$

这里 $R_m(T_m)$是迁移任务完成的总迁移收益，其它的参数如前所述。当迁移任务有最终期限 T_{max} 时，必须解决式(4-5)所示最优问题：

$$max_{Mr(t),T_m \le T_{max}}[\sum R_i(\delta_i)+R_m(T_m)] \quad (4\text{-}5)$$

为了保证迁移不影响或尽可能少地影响存储系统的 I/O 服务，将迁移任务分为一系列的单个迁移请求，迁移任务的附加收益相应地分配给单个迁移请求，称单个迁移请求的附加收益为迁移收益。因此，迁移收益与 I/O 请求收益有相同的地位，对于存储节点，I/O 请求和迁移请求可以采用一种统一的处理方法。基于 QoS 的迁移就是在每个存储节点使用一种在线准入控制算法来控制两种请求，使存储节点获得最大

的收益，图 4.1 给出其功能模型，来自用户的 I/O 请求和来自元数据服务器的请求由元数据服务器统一分配收益，存储节点对收到的请求按收益分类并实现准入控制，保证收益大的请求优先响应，收益小的请求延迟响应或被拒绝。

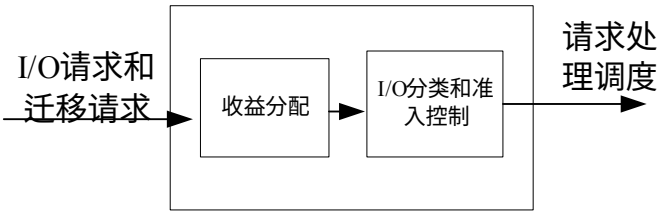


图4.2 基于QoS的迁移功能模型

4.2.2 迁移收益

迁移总收益表示为在时间 t 内完成迁移任务的效果。迁移通常发生在存储系统扩展、数据备份和系统进化等情况，对于不同的情况，迁移收益分配是不同的。它们可通过迁移效用函数来计算，迁移效用函数是用来表示迁移任务完成对系统的性能的改善效果，不同的迁移任务有不同的迁移效用函数。对于系统的扩展，迁移是为了数据分布更合理，从而提高系统的 I/O 性能，因此，随着迁移任务逐渐完成，系统的并行性逐渐增加，迁移效用函数也逐渐增大的，当迁移任务完成后，存储系统的并行性一定，所以，这时的效用函数为定值。如图 4.3(a)表示系统扩展时迁移率一定的效用函数随时间变化的曲线，相应的迁移总收益如图 4.3(b)，它是迁移效用函数对时间的积分。

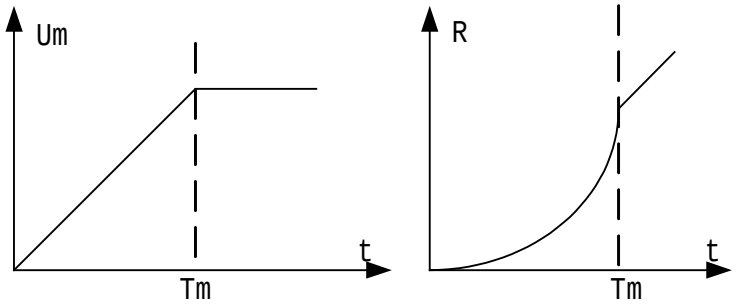


图4.3 系统扩展时的迁移收益

数据备份的效用函数与系统扩展类似，所不同的是它用来提高数据的安全性，根据系统的不同的设置，只是系数不同而已。

在系统进化时分不同的情况，如物理进化，预测磁盘失效，要求在极限时间——平均磁盘寿命内完成迁移。当迁移完成时间没有达到极限时间，业务损失被挽回，所以迁移效用函数可表示为如式(4-6)所示 [96]：

$$U_m(t)=\begin{cases}u & t \le T_{\max}\\ 0 & otherwise\end{cases} \tag{4-6}$$

T_{max} 是时间极限，u 是磁盘失效业务损失。所以迁移总收益要么为 0，要么为 u。

如前所述，为了将迁移任务变成与 I/O 请求类似的迁移请求，将迁移任务分为若干请求，单个迁移请求的收益计算非常复杂。为了简化，将迁移请求的收益表示为如式(4-7)所示：

$$r_m = R_m \cdot \frac{b}{B} \tag{4-7}$$

式中 B 表示迁移任务的大小，而 b 表示单个迁移请求的大小。将迁移任务的收益分配给单个请求，这样，迁移请求就可以象 I/O 请求一样处理从而简化 OSD 目标端的 I/O 调度算法。

4.2.3 基于 QoS 的迁移体系结构

在对象存储系统中，包含一个元数据服务器、多个客户端和多个 OSD 目标端。元数据服务器同时也是管理服务器，负责存储系统的服务质量管理和其它管理。客户和 OSD 目标端是多对多的关系，一个客户可以访问多个 OSD，而多个客户可同时访问一个 OSD。但是，考虑服务质量时，一个 OSD 目标端按其 I/O 带宽和缓冲大小为多个请求（包括 I/O 请求和迁移请求）分配带宽和缓冲，所以基于 QoS 的迁移算法的实现是在一个 OSD 目标端进行的。尽管访问一个 OSD 目标端的客户可能是多个，但客户的服务级别是通过同一元数据服务器获得的，所以，其服务质量按同一标准分配的，便于算法的实现。

实现基于 QoS 迁移的体系结构如图 4.4 所示，它在 OSD 目标端设置一个准入控制器模块，以调整 I/O 和迁移请求流。I/O 请求流来自客户端，迁移请求来自元数据服务器，客户端在发送 I/O 请求前先从元数据服务器获得服务级别，据此定义 I/O 请求的 QoS 属性（包括 I/O 收益和响应期限）。迁移请求来自元数据服务器，元数据服

务器根据迁移目标定义迁移效用函数和其它属性。元数据服务器还根据服务级别算法为客户的 I/O 请求分配服务级别，服务级别和迁移目标在同一服务器分配，便于 I/O 和迁移请求的收益按同一标准计算，OSD 目标端可以对它们统一处理。OSD 目标端采用最大收益算法，对 I/O 请求和迁移请求排序，按优先级为它们分配 I/O 带宽和缓冲，以响应 I/O 和迁移请求。客户端根据 I/O 响应时延调度 I/O 请求，元数据服务器则根据迁移请求的响应时延不断调整迁移率。通过迁移调度，使存储系统在尽可能满足客户 QoS 要求的条件下实现数据迁移。

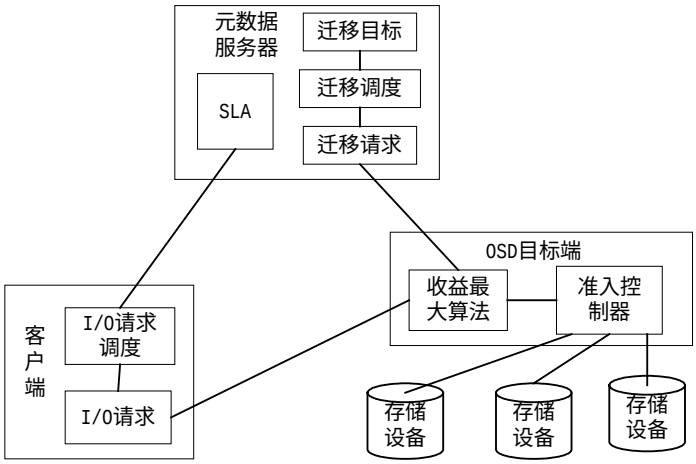


图4.4 基于QoS的迁移体系结构

4.2.4 最大收益调度算法

如前所述，将迁移任务分为迁移请求，对于 OSD 目标端而言，迁移请求和 I/O 请求采用相同的处理方法，这时，请求调度问题可表述如下：假设有 n 个请求输入，每个请求表示为 $r_i<a_i,s_i,R_i,D_i,c_i>$，a_i 为请求到达时间，s_i 是请求保持时间，R_i 是请求收益，D_i 是请求响应的最终期限，c_i 是请求需要的 I/O 带宽。当然，这种请求既可能是 I/O 请求，也可能是迁移请求。假定 C 为 OSD 目标端的总可用带宽，T 为算法考虑的总时间。在有最终期限的迁移中，T 就是最终迁移期限，如果迁移没有最终期限，T 选择一个适当长的时间。当然，即使没有迁移，准入控制器也被使用，这时，T 用适合长间隔表示。基于 QoS 的数据迁移调度就是找出请求调度方案($x_{i,t}$)，使得 OSD 目标端在此期间总收益最大。可表示为如式(4-8)所示：

$$\max \sum_{i=1}^{n} R_i \sum_{t=0}^{T} x_{i,t} \tag{4-8}$$

其中，对于任何时间 t，有：$\sum_{i=1}^{n} c_i \, p_{i,t} \le C$ ；对于任何请求有：$\sum_{i=0}^{T} x_{i,t} \le 1$ ；同时：

$$x_{i,t} = \begin{cases} 1 & r_i \text{在时间} t \text{被调度 且} (t + s_i - a_i) < D_i \\ 0 & \text{其它情况} \end{cases}$$

$$p_{i,t} = \begin{cases} 1 & r_i \text{在时间} t \text{被响应} \\ 0 & \text{其它情况} \end{cases}$$

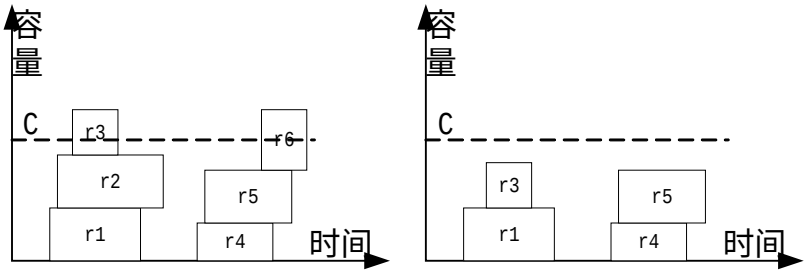
在式（4-8）中，$\sum_{t=0}^{T} x_{i,t} = 0$ 意味着请求 r_i 被拒绝。这种调度在理论上最优，但实现起来非常困难，是一个 NP 难问题[85]。

为了实现上述调度方案，文献[86]中提出一种最短保持时间优先（SRJF: Shortest Remaining Job First）算法。SRJF 算法试图找出与某一请求 r_i 冲突的请求集合，然后在它们间进行调度。

定义 1：r_i 的冲突集合就是到时间 t 还没拒绝的请求 r_j 的集合，且满足：(a) $a_i+s_i>a_j$，$a_i+s_i<a_j+s_j$ 或 (b) $a_j+s_j>a_i$，$a_i+s_i>a_j+s_j$。

由于文献[86]假设所有的请求的收益相同，所以，当找到请求 r_i 的冲突集合后，在满足容量（这里是 I/O 带宽）要求下，首先响应保持时间最短的请求。例如，有请求序列按到达先后顺序为 $r_1, r_2, r_3, r_4, r_5, r_6$，其到达时间及容量分布如图 4.5(a)所示，算法响应如图 4.5(b)所示。对于 r_1，其冲突集合为(r_2, r_3)，按最短保持时间优先算法，应为 r_3 预留容量。当为 r_3 预留容量后，仍然有足够的容量服务 r_1，所以先响应 r_1 再响应 r_3。在此之后，容量不足响应 r_2，故它被拒绝。对于 r_4，其冲突集合为(r_5, r_6)，尽管 r_6 的保持时间小于 r_5，但其保持时间迟于 r_5，故响应 r_4 后响应 r_5，所以其响应顺序如图 4.5(b)。

上述算法存在着几点问题：首先，算法假设所有请求的收益相同，在实际中这是不可能的；其次，算法假定一段时间内的请求已知，对于离线算法可行，对于在线算法无法知道某一请求之后的请求情况[98]。对于后一点，文献[86]采用一个预测器来预测短期内的请求序列，并为有潜在的高收益的请求预留容量。



(a)SRJF的请求　　(b)SRJF的响应

图4.5 SRJF算法

为了调度不同收益的请求，定义相对收益如式(4-9)所示：

$$R_i^{'} = \frac{R_i}{s_i} \tag{9}$$

相对收益综合考虑了收益和请求保持时间，算法优先响应相对收益最大的请求。当收益相同时，算法响应保持时间最短的请求，这回到了 SRJF 算法；当请求的保持时间相同，算法响应收益最大的请求，这正是式（4-8）的要求。

对于请求收益的预测，这里有所不同，存储系统的请求有 I/O 请求和迁移请求，它们有不同的特点。对于迁移请求，由于迁移率基本固定，可以用平均迁移收益表示将来的迁移分布，所以不需预测。对于 I/O 请求，考虑到 I/O 访问的局部性，其请求收益可采用式(4-10)预测：

$$R_i^{'}(t+1) = R_i^{'}(t+1)\cdot(1-\beta) + R_i^{'}(t)\cdot\beta \tag{4-10}$$

其中，$R_i^{'}(t+1)$ 表示对下一请求的相对收益的预测，$R_i^{'}(t)$ 则表示现有请求的相对收益。式中 β 表示现有请求对将来请求的影响因子，它可以采用自适应方法调整。思路如下：如果连续两次 I/O 请求的收益差值增大，β 随着增加，反之减小。

有了 I/O 请求预测和平均迁移率，最大收益算法可以根据收益分布看是否预留容量。具体算法如下：

（1）当 OSD 目标端受到请求时，计算相对收益；

（2）如果请求为迁移，按平均迁移率预测将来请求收益；如果请求为 I/O，按式（10）计算将来请求收益；

（3）如果请求收益小于将来请求（I/O 或迁移）收益，就预留容量（I/O 带宽），

在预留容量后如果容量不足，则拒绝该请求，反之，为该请求分配容量；如果请求收益大于将来请求收益，则直接分配容量。

按照该算法，当 I/O 请求的相对收益较高时，OSD 目标端就拒绝一些迁移请求。由于存储系统采用请求响应模型，也就是说，只有当前一迁移请求响应后，服务器才发送下一迁移请求，这样，当迁移请求被拒绝时，服务器就适当减小迁移率，从而减少迁移对 I/O 服务的影响。相反，当迁移请求连续被响应时，为了充分利用 OSD 目标端带宽，适当增大迁移率。通过迁移率的调整，使得存储系统既完成迁移任务，又最小化其对 I/O 服务的影响。

4.3 实验及结果分析

为了检验最大收益算法，在 DiskSim 的基础上，设计出磁盘调度算法，使其实现最大收益算法。在设计过程中，为了简化实验模型，只考虑 OSD 目标端的算法调度模型，对于客户端和元数据服务器端的 I/O 或迁移请求的调度，通过 I/O 请求或迁移请求的负载设置来实现。模型中假定系统只包含两个磁盘，其中一个是数据迁移目标盘，一个是数据迁移源盘，源盘同时也是 I/O 请求的目标盘。算法主要实现数据迁移源盘中数据的调度，这里不考虑迁移目标盘的原因是，存储系统中数据迁移的目标盘一般较少被用户访问。如前所述，数据迁移通常发生在系统扩展、设备出错、部件升级、系统的进化等。在系统扩展、设备出错、部件升级等情况中，数据迁移的目标盘通常是新增加盘，而新增加盘中用户访问的数据较少；对于系统进化或系统重组，尽管迁移目标盘中有大量的数据，但是，数据迁移是从一个源盘到多个目标盘，所以在目标盘中迁移请求较少，因而对其 I/O 性能影响较少。

实验 I/O 请求采用 DiskSim 自带的综合 I/O 负载，为了算法的实现，随机地给 I/O 请求分配收益：10、5、1。三类收益的 I/O 请求数分别为 10%、30%和 60%。迁移任务用一序列迁移请求来模拟，假定平均迁移率与单位时间平均 I/O 请求数比率为 3：8，据此分配迁移请求，同时为了简化，单个迁移的收益固定为 6。实际上对于不同的迁移任务其收益有不同的分配方法，同一种迁移的收益也可以根据磁盘带宽而变化。实际的 I/O 请求所需带宽可能变化的，这里假设 I/O 请求所需的平均带宽、迁移

所需的平均带宽以及磁盘提供的总带宽之比为 8：3：10，也就是说，磁盘提供的带宽略小于应用和迁移所需带宽。

实验主要比较最大收益算法和两种极端迁移算法 :迁移优先算法和固定平均迁移率算法[92][93]。这两种迁移算法是粗粒度的数据迁移，但它们也可以采用最大收益算法实现，只是采用不同的设置。优先迁移算法是将所有的磁盘带宽用于迁移任务，在最大收益算法中，如果将迁移请求的收益设置为高于任何 I/O 请求，就可以优先响应迁移请求，元数据服务器不断增加迁移率直到所有的磁盘带宽用于迁移，当迁移任务完成后，再响应用户 I/O 请求。平均迁移率算法基本思路是：将迁移率固定为平均迁移率，而平均迁移率是通过迁移的最终期限来计算。为了保证迁移任务的按时完成，将迁移请求的收益设置为大于任何 I/O 请求的收益，这样一来，尽管将迁移分为若干请求，但是，只要迁移一开始，迁移就会完成。采用这种调度算法，在迁移实现的过程中，仍能响应 I/O 请求，只不过是将迁移任务的剩余 I/O 带宽分配给 I/O 请求。

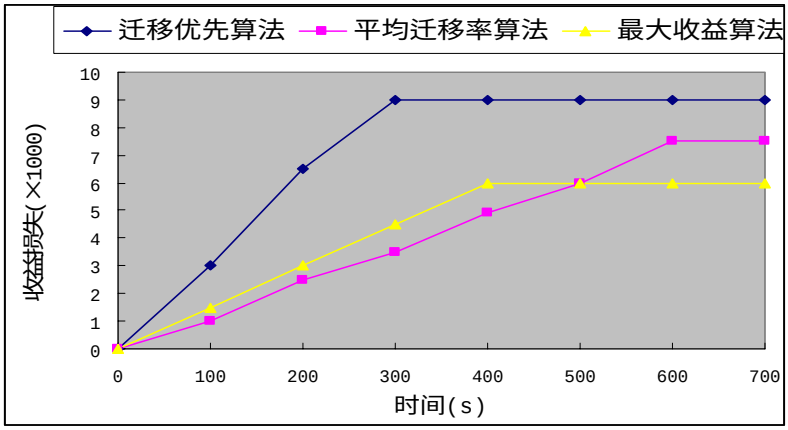


图4.6 三种算法收益损失比较

实验中主要比较两种性能：收益损失和 I/O 请求拒绝数。收益损失是指存储节点随着时间的推移完成的 I/O 请求和迁移请求的总收益的减少，由于迁移的总收益一定，它实际上反映迁移对 I/O 请求服务质量的影响。如图 4.6 所示，迁移优先算法由于在迁移的过程中拒绝所有的 I/O 请求，其收益损失最多，对于固定平均迁移率算法，由于其迁移率是根据最终期限算出，开始时其收益损失较少，但其迁移时间较长，

其最终的收益损失大于收益最大算法。 I/O 请求拒绝数从另一方面反映迁移对 I/O 性能的影响。图 4.7 假定磁盘容量和 I/O 请求分布不变，而迁移负载分别为磁盘容量的 0.1，0.3，0.6 几种情况。从图 4.7 可以看出，随着迁移负载的增加，几种算法的 I/O 请求拒绝数随之增加，但最大收益算法保证迁移对 I/O 服务的影响最小。可见，最大收益算法在保证迁移顺利完成的同时，提供用户 I/O 服务，从而实现基于 QoS 的存储系统。

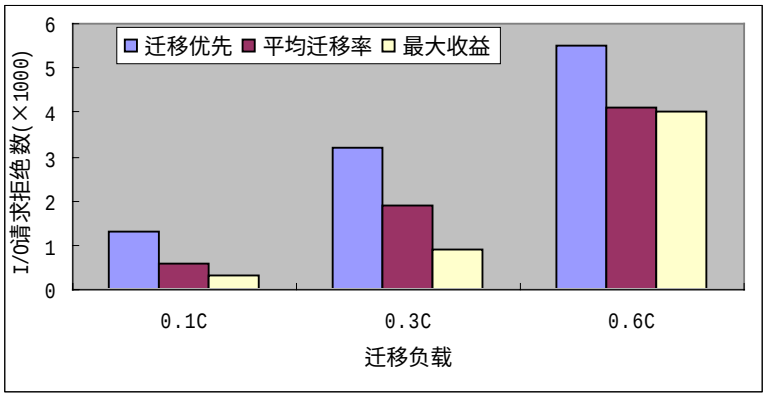


图4.7 迁移负载对I/O服务的影响

4.4 本章小结

在存储系统的管理和维护过程中存在着大量的数据迁移，数据的迁移对存储系统的 I/O 服务存在着一定的影响，而现代应用对存储服务提出了基于 QoS 的要求。通过对传统存储系统和对象存储系统的对比分析，可以看出，传统的存储系统无法提供基于 QoS 的 I/O 服务，而对象存储系统，由于在传输对象的同时随带着一定的属性，它们可用来传输 I/O 请求服务质量要求。通过对对象存储的特点进行分析，给出基于 QoS 的对象存储体系结构，并设计出基于 QoS 的对象存储目标端的 I/O 请求响应调度算法——最大收益算法，该算法是数据迁移调度算法的基础。结合数据迁移的特点，将迁移任务细分为迁移请求，迁移请求就可以象 I/O 请求一样处理。同时，根据迁移的特点，定义各种迁移的效用函数从而给出相应的迁移请求收益，以实现基于 QoS 的数据迁移。在迁移算法的设计过程中，对最短保持时间优先（SRJF）算法进行适当的改进，保证算法在在线情况下实现基于 QoS 的 I/O 请求（包括迁移请

求）的调度。通过对最大收益算法和已有数据迁移算法——迁移优先算法和平均迁移率算法的性能比较，最大收益算法在满足迁移目标的情况下，对用户 I/O 请求的服务质量影响最小。

5 存储系统的 Cache 替换算法的研究

根据摩尔定律，处理器和内存系统的性能每 18 个月翻一番；受机械部件的限制，磁盘访问速度平均每年只能提高 7~10%[3]。处理器和磁盘之间的性能差距已经越来越明显，这样存储系统逐渐成为计算机系统的性能瓶颈，存储系统的速度成了计算机系统性能提高的关键。提高存储系统的性能有多种方法，而通过设置 Cache 缓存数据是提高存储系统性能的一种有效方法。由于 Cache 的容量有限，其替换算法对存储系统的性能有很大影响。本章通过分析外存储系统的特殊性，从 I/O 性能公式入手，讨论 Cache 替换算法的设计依据，在此基础上，提出 Cache 替换算法，并通过实验模拟和测试性能。

5.1 存储系统模型及其性能公式

Cache 不仅用在外存储系统，内存和其它场合也用到，因此，Cache 替换算法不是新问题，已存在着许多 Cache 替换算法。常用的 Cache 替换策略有随机算法（RAND）、先进先出算法（FIFO）、最少使用算法（LFU）、最久没有使用算法（LRU）和将来最久没有使用算法（OPT）[100]。外存储系统中通常采用这些算法及其变体，然而，这些算法是在主存的 Cache 系统中提出的，它们假定 Cache 存储系统的性能随着 Cache 的命中率的提高而提高，对于主存－Cache 系统，由于主存的访问速度恒定，访问数据的单位固定不变，上述结论是正确的。但是，大容量复杂的外存储系统往往由性能不同的存储设备组成，存储设备间的互连也有不同的方法，这样对存储设备的访问速度不一定不同，有时速度甚至相差悬殊；即使存储设备的访问速度相同，如果访问的数据的单位大小不等，其设备访问时间也不会相同。当存储设备的访问速度不等，数据的访问单位不固定时，其访问存储设备的时间不再相同，在这样的存储系统中设置 Cache，Cache 命中率最高并不意味着存储系统的性能最好。

这里首先给出存储系统的 Cache 模型，分析其不同于主存 Cache 的特点，并分析其 I/O 性能公式，讨论影响 I/O 性能的相关因素，为存储系统 Cache 算法的设计给

出理论依据。

5.1.1 层次存储系统模型

层次存储系统模型如图 5.1 所示，它由存储设备和 Cache 组成。对于 Cache—主存系统，存储设备为主存，它们通常是速度相同的元器件，Cache 为高速缓存。在存储系统中，Cache 为主存或其它高速存储器件，而存储设备可能由不同的元器件组成，它们既可以磁盘、光盘、还可以是存储网络，因此它们的访问速度不同；不同设备与服务器的连接可以采用不同方法实现，数据传输速度也不一定相等。服务器采用高速存储设备构成集中式 Cache，服务器访问 Cache 的速度是恒定的。为使问题简单化，忽略负载平衡问题，假设设备与 Cache 间的通信延迟是恒定的，包含于设备的访问时间；同时假设 Cache 的容量足够大，能够保存足够的数据，以便替换算法的实现。

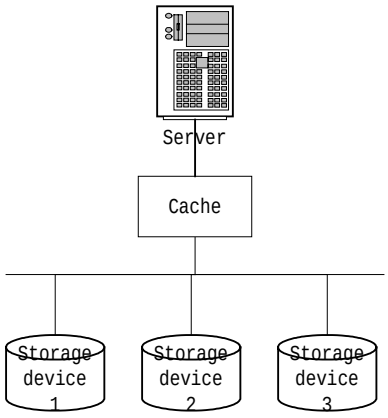


图5.1 存储系统的Cache模型

服务器对存储系统的访问通过文件系统实现，文件系统将某一数据的请求映射到 Cache。对于读操作，如果 Cache 命中，数据返回用户，否则，再访问存储设备，读取的数据送 Cache 后再由其返回用户；写数据时，数据只写 Cache，当 Cache 满或 Cache 空闲空间达到一定的阈值时再按一定的替换算法把数据写回存储设备。数据以对象为单位，对于不同的应用数据对象不同，它们既可以是文件，也可以是数据块，还可以是其它形式的数据。数据对象的大小既可能恒定也可能是可变的，例如，数据对象是数据块时，其大小是固定的，当以文件为数据对象时，其大小显然是可变的。

5.1.2 存储系统的性能公式

存储系统设置 Cache 的目的是为了提高其性能，加速比是反映采用 Cache 后存储系统性能改善程度的重要标准[102]。加速比是存储系统在完成相同任务时不采用 Cache 的访问时间与采用 Cache 的访问时间之比，如式(5-1)：

$$g = \frac{\text{无}Cache\text{时的访问时间}}{\text{有}Cache\text{时的访问时间}} \tag{5-1}$$

假设用户访问的数据对象集合为{f_1，f_2，f_3，…，f_n}，相应的数据对象大小为：{s_1，s_2，s_3，…，s_n}，每个数据对象被访问的次数分别为{m_1，m_2，m_3，…，m_n}，则存储系统总的访问次数如式(5-2)：

$$M = \sum_{i=1}^{n} m_i \tag{5-2}$$

当不采用 Cache，数据对象来自存储设备，其对应的 I/O 时间为：{t_1，t_2，t_3，…，t_n}，各数据对象的单位数据的访问时间为：{t_1'，t_2'，t_3'，…，t_n'}。由于数据对象来自不同的存储设备，数据对象的大小不一定相等，不同数据对象的 I/O 时间不一定相等，这时的总 I/O 时间 T_o 如式(5-3)所示：

$$T_o = \sum_{i=1}^{n} m_i * t_i = \sum_{i=1}^{n} m_i * s_i * t_i' \tag{5-3}$$

存储系统设置 Cache 时，除第一次必须从存储设备中读取外，以后的数据对象 I/O 可能在 Cache 中访问。当数据对象 f_i 在 Cache 中时，其 I/O 时间 t_{ci} 如式(5-4)所示：

$$t_{ci} = s_i * t_c \tag{5-4}$$

其中 t_c 为 Cache 中单位数据的访问时间，由于采用集中的 Cache，对于确定的 Cache，t_c 是恒定的。实际上，Cache 的访问时间还包括替换算法的实现时间，对于存储系统，由于数据对象一般很大，访问 Cache 时间较长，为简化问题，替换算法的实现时间可以忽略。

假设数据对象在 Cache 中的命中率分别为{p_1，p_2，p_3，…，p_n}，则 Cache 的总命中率 P 如式(5-5)所示：

$$P = \sum_{i=1}^{n} p_i \tag{5-5}$$

当设置 Cache 时，存储系统总的访问时间 T_c 为在 Cache 中访问数据对象的时间和在存储设备中访问数据对象的时间之和，如式(5-6)所示：

$$T_c = \sum_{i=1}^{n} M * p_i * s_i * t_c + \sum_{i=1}^{n} (m_i - M * p_i) * s_i * t_i' \tag{5-6}$$

这样，存储系统设置 Cache 的加速比如式(5-7)所示：

$$g = \frac{T_o}{T_c} = \frac{\sum_{i=1}^{n} m_i * t_i}{\sum_{i=1}^{n} M * p_i * s_i * t_c + \sum_{i=1}^{n} (m_i - M * p_i) * s_i * t_i'} = \frac{1}{1 - \frac{M * \sum_{i=1}^{n} p_i * s_i (t_i' - t_c)}{\sum_{i=1}^{n} m_i * t_i}} \tag{5-7}$$

根据存储系统的性能公式（5-7），可以得出以下结论：

（1）当存储设备的访问速度相同且数据对象的大小相等，也就是 t_i'、s_i、都是常量，这时式（5-7）可简化为：

$$g = \frac{1}{1 - \frac{M * s_i * (t_i' - t_c) * \sum_{i=1}^{n} p_i}{\sum_{i=1}^{n} m_i * t_i}} = \frac{1}{1 - \frac{P * (t_i' - t_c)}{t_i'}} \tag{5-8}$$

这时，Cache 的总命中率越高，加速比就越大。常用的 cache 替换算法 LRU 和 LFU 等正是在这样的条件下得出来的。

（2）当存储设备的访问速度不相同且数据对象的大小不相等，式（5-8）不成立，由式（5-7）可知：$\sum_{i=1}^{n} p_i * s_i (t_i' - t_c)$ 越大，加速比越大。由于命中的数据对象的大小 s_i、单位数据的存储设备访问时间 t_i'及 Cache 的访问速度 t_c 都是确定的，当所有的数据对象在 Cache 中的命中率 p_i 都达到最大时，Cache 的总命中率 P 最大，加速比才会最大。这要求所有的数据对象只有在第一次访问来自存储设备，其后的访问都来自 Cache，也就是说要求 Cache 有足够的存储空间缓存所有将来要访问的数据对象。由于服务器访问的数据是无限大，而 Cache 的容量有限，这对于存储系统的在线访问

实际上是不可实现的。

（3）实际存储系统中 Cache 的容量是有限的，不可能所有的数据对象的命中率都达到最大，因而 Cache 总命中率 P 不可能达到最大值。这时并不是总命中率 P 越大越好，因为 P 大并不意味着$\sum_{i=1}^{n} p_i * s_i(t_i'-t_c)$大，所以加速比不一定最大。加速比不仅与总命中率有关，还与数据对象命中率的分布有关。只有当设备访问时间长的数据对象在 Cache 中的命中率尽可能高时，$\sum_{i=1}^{n} p_i * s_i(t_i'-t_c)$才可能大，加速比才会最大。这就要求设备访问时间长的数据对象尽可能保存在 Cache 中，这正是外存储系统的 cache 替换算法的理论依据。

5.2 存储系统的 Cache 替换算法

根据前一节的结论，按照不同的思路，可以得出不同的 Cache 替换算法。这里给出两种不同的 Cache 替换算法：最少访问时间（LAT，Less Access Time）算法和加权 LFRU(Weighted Less Frequently/Recently Uesed)算法。

5.2.1 LAT 算法

根据存储系统的性能公式，影响存储系统性能的是 Cache 中的对象的命中率，但不是总的命中率。由式（5-8）可知，要使存储系统的加速比尽可能大，必须使$\sum_{i=1}^{n} p_i * s_i(t_i'-t_c)$尽可能大，这要求$s_i(t_i'-t_c)$大的数据对象尽可能长地保存在 Cache 中，即设备访问时间长的数据对象的命中率越高，存储系统的加速比才越大，性能越好。为此，我们为每个数据对象定义一个函数，如式(5-9)所示：

$$\phi = p_i * s_i * (t_i'-t_c) \tag{5-9}$$

式(5-9)没有考虑到 Cache 的空间利用率，如果数据对象的 φ 大，但其占的 Cache 空间也大，让它长期保存在 Cache 中，Cache 的利用率不高。由于实际 Cache 的容量有限，Cache 的替换数据对象必须比较单位数据的 φ 值。可用式(5-10)表示：

$$\phi' = \frac{\phi}{s_i} = \frac{p_i * s_i * (t_i'-t_c)}{s_i} = p_i * (t_i'-t_c) \tag{5-10}$$

由于选择替换对象时只考虑其 φ 的相对大小，为简化算法，式（5-10）中的常数 t_c 可以不考虑，这时可表示如式(5-11)所示：

$$\phi' = p_i * t_i' = p_i * \frac{t_i}{s_i} \tag{5-11}$$

据此，当存储系统需要 Cache 空间时选择 φ′最小的数据对象作为替换的对象。

式（5-11）虽然给出了选择 Cache 替换对象的标准，但它无法实现，因为对于一个存储系统，其访问的数据对象是不确定的，有无穷多个，每个数据对象的访问次数也无法确定，其概率也无法计算，这样，函数 φ′的值也就无法计算。

由于计算 φ′值的目的是选择 Cache 的数据对象作为替换对象，可以只考虑在某一刻 Cache 中的数据对象，这样数据对象数有限；如果只考虑在该时刻过去一段时间内数据访问的次数，这样数据访问的次数可测。这样函数 φ′是时间的函数，式（5-11）可表示如式(5-12)所示：

$$\phi'(t) = p_i(t) * \frac{t_i}{s_i} \tag{5-12}$$

假设各数据对象的访问是相互独立且满足指数分布，则数据对象在 Cache 中的命中率可以用泊松系数 $\lambda_i(t)$近似表示，如式(5-13)所示：

$$p_i(t) = \frac{\lambda_i(t)}{\sum_{j=1}^{n'} \lambda_j(t)} \tag{5-13}$$

式中的 n′表示在时刻 t 保存在 Cache 的数据对象数，$\lambda_i(t)$表示数据对象 f_i 在 Cache 中的访问率，$\lambda_j(t)$表示数据对象 j 在存储系统中的访问率。我们的目的是比较数据对象的相对概率，所以，式（5-13）可简化为如式(5-14)所示：

$$p_i(t) = \lambda_i(t) \tag{5-14}$$

$\lambda_i(t)$的计算可用一段时间内的访问 Cache 的次数除以时间得到，显然，时间的长短对 $\lambda_i(t)$的值有影响。我们选择数据对象从 t 开始过去的 $k_i(t)$次访问作为研究对象，显然，$k_i(t)$不能超过其在 Cache 中的命中次数；$k_i(t)$次访问所花的时间为 t 到过去第 $k_i(t)$次的时间 t_{-ki} 的时间间隔[103]。这时可表示为如式(5-15)所示：

$$p_i(t) = \frac{k_i(t)}{t - t_{-k_i}} \tag{5-15}$$

所以式（5-12）可表示为如式(5-16)所示：

$$\phi'(t) = \frac{k_i(t)}{t - t_{-k_i}} * \frac{t_i}{s_i} \tag{5-16}$$

式（5-16）表明：近期访问次数少、单位访问成本低的数据对象的函数值$\phi'(t)$也就小。根据数据的访问局部性，现在$\phi'(t)$小的数据对象将来也小，反之亦然。所以，Cache 的替换算法可按下述思路设计：将 Cache 中的所有数据对象按$\phi'(t)$排序，当需要 Cache 空间缓存新的数据对象时，选择$\phi'(t)$值最小的一个或几个数据对象作为替换对象，这样就可以保证近期访问次数多、单位访问成本高的数据尽可能长时间地保存在 Cache 中，从而使存储系统的总访问时间短，使存储系统的加速比达到最大、速度最快。由于这种算法总是选择 Cache 中设备访问时间小的数据对象作为替换的对象，所以称之为最小访问时间替换算法。

为计算Cache中的数据对象的$\phi'(t)$值，必须记录Cache中所有数据对象的使用信息，这些由数据对象链表实现。数据对象链如图2所示，每个数据对象由记录号和记录两个字段组成。在记录字段中，Ac和As分别为数据对象f_i的Cache地址和存储设备地址，t_i表示从存储设备上访问数据对象f_i所需的时间，s_i为数据对象f_i的大小。

t_{-k}表示从时刻t起到过去第k次的访问时间，它的记录可以用一个k层的先进先出栈实现，为了减少替换算法的开销，用它记录数据对象第一次进入Cache的时间，这样$k_i(t)$对应为该数据对象在Cache中访问的次数。

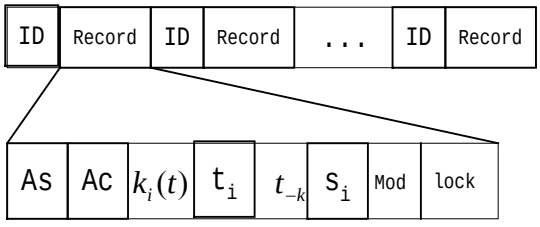


图5.2 数据对象链

lock表示数据对象的使用状态，Cache中的数据对象有两种状态：正在使用的数据对象（称之为锁定数据对象）和暂时没有使用的数据对象（称之为非锁定数据对

象）。显然，锁定数据对象是不能替换的，这样替换数据对象只能在非锁定数据对象中选择。Lock是一个整形变量，其初始值为0，每一个对该数据对象的请求使其加1，当某一请求完成后对应的lock减1，这样lock>0表示该数据对象是锁定的，lock=0时表示该对象为非锁定的。

对于非锁定的数据对象，按其 $\phi'(t)$ 值的大小构成一个替换优先级链，链中的节点由三个字段组成：数据对象的ID、$\phi'(t)$ 及其大小 s_i。优先级链按降序排列，链首为 $\phi'(t)$ 最大的数据对象，它在Cache中保存的时间最长，链尾为 $\phi'(t)$ 最小的数据对象，它是优先替换的对象。

为了加快存储系统的写速度，Cache必须预留一定的空闲空间（阈值）[104]。这样必须设置一个Cache空闲空间大小变量 S_{id}，当 S_{id} 大于阈值时写Cache并不发生数据对象替换，当 S_{id} 小于阈值时进行数据对象替换。

有了上述数据结构，Cache的替换算法就可以实现了。LAT替换算法如下：

（1）文件系统收到I/O请求后，若是读请求首先访问Cache，如果数据对象在Cache，就从Cache中读，否则从存储设备上读，同时写Cache；当用户写存储系统时直接写Cache。

（2）访问Cache时，系统更新存储对象链：首次访问时加入数据对象记录，再次命中时更新数据对象记录。访问Cache前数据对象前对应的lock加1，该请求完成后对应的lock减1。

（3）当数据对象的lock为0，数据对象由锁定状态进入非锁定状态，在替换优先级链中加上该数据对象，该数据对象可能被替换。当非锁定数据对象被再次命中时，替换优先级链中的记录被删除，该数据对象再次成为锁定数据对象。

（4）当空闲存储空间 S_{id} 小于阈值时，在替换优先级链中选择优先级高的数据对象删除。删除前查找修改位，当修改位为0时直接删除，当修改位为1时先写回数据对象再删除。数据对象删除后，其在数据对象链、替换优先级链中的对应记录被删除。

5.2.2 WLFRU 算法

LAT算法虽然反映了存储系统中设备的访问时间差别，但算法的计算开销较大，

且算法占用较大的存储空间。这种算法对网络存储中远程访问比较实用，因为在远程访问中通常采用积累方法将小数据对象组合成大数据对象。当数据对象较大时，算法的空间和时间开销相对而言可以忽略不计。这里根据上述结论，将LFU、LRU算法扩展为加权LFU和加权LRU算法，在此基础上，提出另一种算法——WLFRU算法。

5.2.2.1 WLFU 算法

对于式（5-12），如果将$\frac{t_i}{s_i}$用常数k表示，可得到式(5-17)：

$$p_i^{'} = k_i p_i \tag{5-17}$$

常数k对于不同的数据对象有所不同，所以式（5-17）是Cache中数据对象的加权命中率，把它代入式（5-7），可以得出结论：加权命中率高的数据对象在Cache中保存的时间越长，存储系统的加速比越大。这样，常用的Cache替换算法LFU就扩展为加权LFU（WLFU，Weighted LFU）算法：当Cache需要存储空间保存新数据对象时，总是选择加权命中率小的数据对象作为替换的对象。通常，为了简化算法的实现，用访问次数代替访问命中率，所以，WLFU算法选择Cache中近期最少访问的数据对象作为替换的对象。

WLFU算法的实现，关键是权值k_i的计算，对比式（5-12）和（5-17）可知，k_i实际是数据对象的单位设备访问时间，它在第一次访问数据对象时容易得到。然而，算法更关注数据对象间的相对访问时间，为此，可用数据对象的单位设备访问时间除以最小的单位设备访问时间。对于在线算法，最小单位设备访问时间并不能预先知道，可采用逐渐更新的方法保证权值总是为最小设备访问时间的相对值。算法如下：

（1） 存储系统有I/O请求时，访问Cache，如果Cache命中，数据对象对应的记录中访问次数加上对应的权值，并排列替换队列；

（2） Cache不命中时，访问存储设备，并在替换队列中加上对应的记录，同时比较单位设备访问时间与最小单位设备访问时间，若小于最小单位设备访问时间，则更新最小设备访问时间，并更新所有数据对象访问次数（第一次访问时，将其单

位设备访问时间设置为最小设备访问时间）；

（3） Cache空闲存储空间小于一定值时，替换加权访问次数最小的数据对象，直到空闲存储空间大于定值为止。

值得说明的是，在计算权值时，采用近似整数以简化计算。

5.2.2.2 WLRU 算法

WLFU算法的理论根据是Cache中数据的访问频率局部性：近期访问频繁的数据对象在将来也访问频繁。实际上，Cache中数据访问还有时间局部性：近期访问的数据对象在将来也会被访问，LRU算法就是根据这一假设提出的。在外存储系统中，考虑到数据对象的设备访问时间的差别，可将LRU算法扩展为加权LRU（WLRU，Weighted LRU）算法。

与WLFU算法不同，WLRU是对对象访问距现在的的时间间隔的加权，使设备访问时间长的对象在LRU栈中尽可能靠近栈顶，而设备访问时间短的对象远离栈顶，因此，权值可以按式(5-18)计算：

$$k_i = \left\lceil \frac{t'_{\max}}{t'_i} \right\rceil \tag{5-18}$$

式中$t'_{\max}$最大单位设备访问时间，它随着数据的访问而不断更新，而t'_i为对象i的单位设备访问时间。按式（5-18）计算，对象的权值分别为大小不同的整数，且单位设备访问时间长的对象的权值小，而单位设备访问时间短的对象的权值。

在LRU算法中，当对象被访问时，将它移到栈顶，然而，这里对象存在权值，不能简单将它移到栈顶，而是根据权值将它移到距栈顶的第k_i层，所以存在着插入的问题。为了解决这个问题，引入窗口概念，窗口设置在栈顶，其大小表示为如式(5-19)所示：

$$Window = \left\lceil \frac{t'_{\max}}{t'_{\min}} \right\rceil \tag{5-19}$$

这样，权值不可能大于窗口，所以在窗口中存在插入问题，而在它外面象LRU算法一样。同样窗口的大小是不断地更新的。算法如下：

（1） 当访问数据对象时，首先访问Cache，如果Cache命中，对象在LRU栈中

移到窗口中的k_i层，k_i层及其以下层依次下移一层，直到对象访问前所在层为止。

（2） 如果Cache没有命中，访问存储设备。如果是第一个数据对象，单位设备访问时间的最大值和最小值都设置为该对象的单位设备访问时间，在以后的访问中，如果单位设备访问时间大于最大值或小于最小值，更新最大或最小值，同时更新窗口大小。当最大值更新时，必须更新栈中所有的权值。计算数据对象的权值，并按权值插入窗口，同时移动其下面的所有对象。

（3） 当Cache的空闲空间小于一定极限时，替换栈底的若干数据对象，直到空闲空间大于极限值。

5.2.2.3 WLFRU 算法

WLFU算法考虑了数据访问的频率局部性，而WLRU考虑了数据访问的时间局部性，它们从一个侧面反映了数据访问的局部性。例如，在WLFU算法中，现在频繁访问的数据对象和过去频繁访问的数据对象没有区别对待，而在WLRU算法中，很可能刚刚访问的数据对象的访问次数很少，而频繁访问的数据对象被替换。为了解决这两种算法的不足，提出了WLFRU算法，它同时兼顾这两种局部性。

为了实现WLFRU算法，采用两个可变长的LRU栈：L_1和L_2。如图5.3所示，L_1保存最近只访问一次的数据对象，而L_2保存至少访问两次的数据对象，因此，L_1中的对象可能是较少访问的，而L_2中可能是访问频繁的对象。为了反映访问时间局部性，两个栈都采用WLRU算法，栈顶是最近访问的数据对象，而栈低是最久没有访问的数据对象。

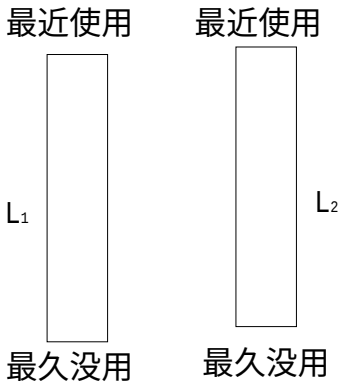


图5.3 WLFRU算法的LRU栈

假定Cache的最大空间为C，L_1的最大深度为L，L_1和L_2满足式(5-20)：

$$0 \le L_1 \le L, \quad 0 \le L_2 \le C, \quad 0 \le L_1 + L_2 \le C \tag{5-20}$$

L的设置对Cache的性能有很大影响，太小，可能引起L_1中出现颠簸现象，例如，对于某一循环程序，如果其大于L，则它们访问一次就可能被替换，实际上，它们会被多次访问。当然，L太大，可能导致L_2太小，其中的频繁数据对象被替换。为此，L的初始值设置为C/2，在以后，它随着L_1和L_2的替换频繁程度而变化。当L_1替换频繁，就增大L；当L_2替换频繁就减小L。为了保证L不频繁变化，可记录L_1和L_2的替换次数，当两者的替换次数之差大于一定值时，才增大或减少L。WLFRU算法如下：

(1)Cache命中时，数据对象的记录按WLRU算法插入L_2中，同时删除L_1中的对应记录。这时候，如果L_2=C-L，就必须替换其中的记录。

(2)如果Cache访问失效，就访问存储设备，数据对象对应的记录按WLRU算法插入L_1，这时，如果L_1=L，L_1中必须发生替换。

(3)当L_1替换的次数比L_2中的替换次数大一定值时，就增大L，反之则减小L。在L增大或减小时，同样会引起L_1或L_2的替换。

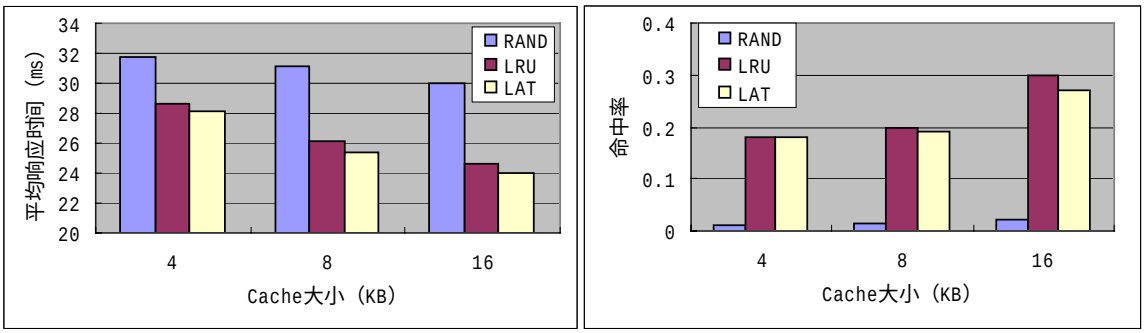
5.3 Cache 替换算法的模拟及性能分析

传统的Cache替换算法的性能评价标准是命中率，它用在Cache中访问数据对象的次数占总I/O请求数的比率来表示。命中率反映了Cache的设置导致存储设备访问的次数的减少，它在一定程度上可以反映存储系统的性能的提高。如前所述，高命中率并不意味着高性能，因此，本文采用平均响应时间来表示存储系统的性能，它较好地反映了Cache的设置导致存储系统访问速度的提高。

我们在实验室采用Disksim模拟器来仿真存储系统。Disksim是Carnegie Mellon大学开发的对存储设备和存储系统的性能进行模拟的软件[105]。为了验证LAT算法的好坏，实验中只比较了RAND、LRU和LAT替换算法，而没有考虑FIFO、LFU和OPT算法，这是由于FIFO、LFU、OPT与LRU一样，都是根据数据访问的相关性来提高Cache的命中率的，故只选LRU作为比较对象代表。Disksim自带有Cache的RAND和LRU替换算法，实验中对原代码进行适当的修改以实现LAT替换算法，请求流采用Disksim

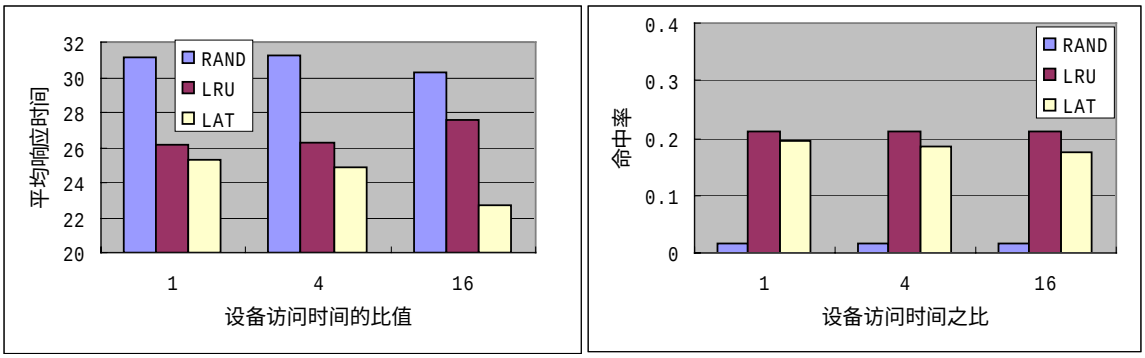
自带的合成请求流。

实验分别考虑Cache大小和设备访问时间对Cache性能的影响。首先保证存储设备的分布是固定的，测量平均响应时间和命中率随Cache大小变化的规律，如图5.4所示。从中可以看出三种替换算法的平均响应时间都随着Cache的增大而减小，但LAT算法减小得最快；它们的命中率都随着Cache的增大而增大，其中LRU算法增幅最大，可见，命中率的提高并不一定意味着平均响应时间的降低。



(a)平均响应时间与Cache大小的关系　　(b)命中率与Cache大小的关系

图5.4 Cache大小对LAT算法性能的影响



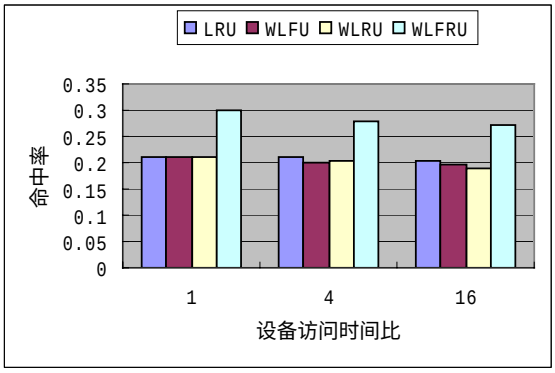
(a)平均响应时间与设备访问时间的关系　　(b)命中率与设备访问时间的关系

图5.5 设备访问时间对LAT性能的影响

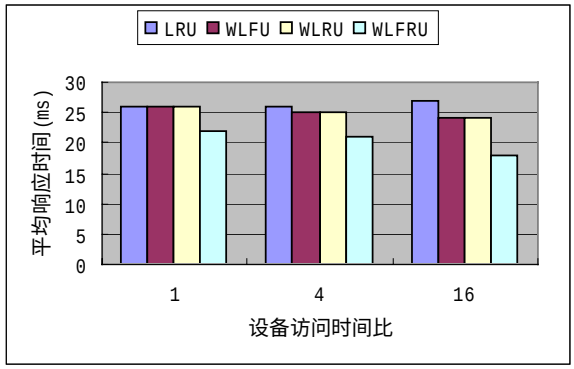
然后假设数据对象的大小不变，只考虑存储设备的速度对系统性能的影响。存储设备由两个磁盘组成，保持它们的平均访问速度一定，让两者的速度差由小到大变化，以观察三种替换算法的命中率和平均响应时间随存储设备的访问时间变化而变化的关系。如图5.5所示，LRU算法的命中率随着存储设备的访问速度之比的增大基本不变，LAT算法的命中率随着存储设备的访问速度之比的增大略有下降。但是，随着存储设备的访问速度之比的增大，LRU算法的平均响应时间增大，而LAT的平均

响应时间减少。可见，存储设备的访问速度偏离平均访问速度越大，LAT算法的性能越好。

为了评价WLFRU算法的性能，将它与LRU、WLFU及WLRU进行比较，实验通过对DiskSim的代码修改，测量几种算法的命中率和平均响应时间。Cache命中率如图5.6(a)所示，WLFU和WLRU的性能和LRU差不多，而WLFRU的命中率最高，这是由于WLFU和WLRU是通过减少设备访问时间短的数据对象的命中率来提高设备访问时间长的数据对象的命中率，但是，WLFRU同时考虑了访问的频率和时间局部性，所以其命中率最高。存储系统的平均访问时间如图5.6(b)所示，WLFU和WLRU的平均响应时间低于LRU，而WLFRU则有最低的平均响应时间，这是因为WLFU、WLRU和WLFRU不仅考虑访问的局部性，而且还考虑了设备访问时间的区别。



(a)命中率与设备访问时间比的关系



(b)平均响应时间与设备访问时间比的关系

图5.6 WLFRU的性能

5.4 本章小结

利用Cache改善存储系统的性能时替换算法的选择是重要的因素，而常用的替换算法是假设Cache的命中率高时存储系统的性能好，该结论在存储设备的访问速度接近时成立。当存储系统的存储设备的访问速度相差较大时，Cache的命中率高并不意味着系统大性能好。本章通过对存储系统的加速比的分析得出结论，只有提高单位设备访问时间长的数据对象的命中率，才能达到减小平均I/O响应时间。根据这一结论，首先设计出LAT算法，该算法并没有提高Cache命中率，但是可以减少平均I/O响应时间，然后对比LRU、LFU算法，结合上述结论，设计出WLFRU算法，该算法不仅考虑到单位设备访问时间的差别，还同时考虑到存储系统的访问频率和时间局部

性，从而达到降低平均I/O响应时间的目的。实验表明，由于考虑到的单位设备的访问时间的差别，LAT和WLFRU算法都能达到降低存储系统的平均I/O响应时间的目的。

6 对象存储系统中的 Cache 方案

对象存储系统中存在着客户端、元数据服务器和存储节点三种不同的实体，客户端的用户访问存储系统时首先访问元数据服务器，在获取元数据后再访问存储节点。对存储节点的访问有两种模式：NAS 模式和三方传送模式，它们各有不同的特点，适用于不同的数据对象。本章根据对象存储中三种实体的数据访问特点，结合数据传输模式，设计出它们的 Cache 方案及其替换算法，并通过实验验证 Cache 方案对存储系统的 I/O 性能的改善。

6.1 存储系统 Cache 方案的整体思路

随着计算机应用的发展，存储系统成逐渐成为计算机系统的性能瓶颈，提高存储系统的性能是存储界的重要课题。在数据传输路径中设置 Cache，使数据在传输路径中合理分布是提高存储系统的 I/O 性能的一种行之有效的方法。

在对象存储系统中，大量的存储节点通过元数据服务器组织成单一的存储空间，元数据服务器负责元数据的管理和其它存储管理，客户对存储设备的访问分为两步：首先访问元数据服务器，然后访问存储设备。客户访问存储系统有两种模式：NAS 模式和三方传送模式[107]。在 NAS 模式中，存储节点相当于元数据服务器的外接磁盘，客户访问元数据服务器，元数据服务器访问存储节点，当元数据服务器从存储节点获取数据或响应信息时再响应客户。三方传送模式则不同，客户首先从元数据服务器获得元数据，然后根据元数据直接访问存储节点。这两种模式各有自己的特点，适用不同的应用：三方传送模式中客户直接访问存储设备，减少了数据传输的中间转发过程，但是，它增加了从元数据服务器获取元数据过程，因此，当数据对象较小时，获取元数据的时间大于数据中间转发的时间，性能较差，相反，对于大数据对象，其数据对象中间转发时间远大于获取元数据的时间，采用三方传送性能较好。因此，三方传送模式适用于大数据对象的读写，相反，NAS 模式适用于小数据对象的读写。

无论是NAS模式还是三方传送模式，在用户访问存储系统时涉及到着三种实体：客户、元数据服务器和存储节点。由于数据访问存在着局部性，在客户端设置Cache，用来保存数据对象及其元数据，可以减少访问元数据服务器和存储节点的次数，同时，缓存的数据对象利用网络空闲带宽写存储节点，可以减少网络拥塞，提高I/O性能。由于存储系统中存在着大量的客户，不同的客户可能访问同一数据对象，数据对象保存在元数据服务器可充分利用数据访问的局部性，但是，这样一来，所有的数据访问都通过元数据服务器，元数据服务器就成为性能瓶颈。通过在客户端设置Cache以及采用三方传送模式可以解放元数据服务器。然而，三方传送模式对于小数据对象性能不好，且客户端 Cache 无法利用多用户访问同一数据的局部性，为此，同时采用 NAS 和三方传送两种模式传输数据，使小数据对象采用 NAS 模式，大数据对象采用三方传送模式。这样，在元数据服务器设置Cache，缓存元数据和小数据对象，达到改善性能的目的。客户对存储系统的访问最终要读写磁盘，由于机械速度的原因，磁盘读写成为存储系统I/O性能瓶颈，为此，存储节点也必须设置适当的Cache，用来改善磁盘的读写速度。所以，为了改善存储系统的I/O性能，在存储系统的客户端、元数据服务器和存储节点中，都应该设置Cache。

本章试图根据三种存储实体的不同特点，设计出合适的 Cache 方案及其替换算法，从而达到提高存储系统的I/O性能的目的。

6.2 存储节点的Cache方案

本节首先分析存储节点的数据访问特点，在此基础上，采用一种新型的存储设备——基于MEMS（microelectromechanical systems）存储设备作为Buffer，并根据数据访问的局部性，将Buffer分为两个：写Buffer和预取Buffer，通过它们分别提高存储节点的读写速度。

6.2.1 存储节点的数据对象访问特点

在对象存储系统中，客户端、元数据服务器和存储节点间是以对象为单位进行访问的，然而，在存储节点，数据对象最终保存在磁盘上，而磁盘的读写是以数据块为单位的，数据对象和数据块间格式的转换是通过存储节点控制器来进行的。存

储节点的数据访问有以下特点：

（1） 节点控制器访问磁盘是以块为单位的，而元数据服务器或客户是以对象为单位访问存储节点，数据对象通常包含多个数据块，因此，磁盘的访问具有空间局部性。

（2） 在三方传送模式中，不同的客户获取元数据后都可以访问存储节点，同一存储节点的数据对象可能被多个客户访问，这样，存储节点上的数据对象的访问存在着时间局部性。

（3） 由于磁盘读写速度较慢，磁盘成为存储节点的性能瓶颈，但磁盘具有掉电数据不丢失特性，这就要求存储节点的 Cache 也具有类似的性质，从而保证 Cache 在提高 I/O 性能的同时不影响数据一致性。

（4） 通常，存储节点的磁盘是同时更新的，磁盘的性能基本相同，所以 I/O 速度基本相同；由于存储节点采用统一的文件系统，磁盘的数据块大小相同。

根据磁盘访问的空间局部性，在存储节点设置一个预取 Buffer，以提高磁盘的读性能，根据数据对象访问的时间局部性，采用写 Buffer，通过写回法实现数据对象的滞后写，减少数据的写时间，同时利用访问的局部性减少 I/O 操作；为了保证 Buffer 中的数据的一致性，采用一种新型存储设备——MEMS 存储设备作为 Buffer。

6.2.2 MEMS 存储设备

MEMS（microelectromechanical systems）存储技术是一种正在开发的新型技术，它采用与磁盘不同的体系结构，但具有磁盘一样的非挥发性[108][109][110]。与磁盘相比，MEMS 存储设备寻道速度快十倍，存储密度大十倍，能量消耗小一到两个数量级。由于体系结构、微型结构和制造工艺的不同，它比磁盘更可靠[111][112][113]。但是，与磁盘相比，其成本贵 5-10 倍，由于性能和制造工艺的考虑，单片表面容量限制在 1-10GB。因此，短期内用它来代替磁盘是不现实的，磁盘将在将来一段时间在外存中仍占统治地位。由于 MEMS 存储设备有磁盘类似的特点，如非挥发性、块数据访问等，且提供比磁盘更好的性能，它可用来与磁盘构成层次存储，以提高其性价比。

MEMS 存储设备包含两类组件：移动的探针阵列和非旋转的媒体片。与磁盘通过磁片旋转进行一维定位不同，MEMS 设备通过静电力在二维空间移动探针阵列。

探针阵列是多个同时读写的探针，通过它来实现访问的并行性。由于能量和散热的限制，同时工作的探针有所限制，估计在 200 到 2000 之间。在 MEMS 设备中，媒体被分为不重叠的探针区，每个探针区只能被一个探针访问，大约为 2500×2500 比特，它根据媒体片的移动范围决定。在 MEMS 中的最小数据单位是探针扇区，每个探针扇区由(x,y,tip)决定，其中 tip 用来表示探针号，(x,y)用来表示探针在探针扇区的坐标。与磁盘类似，将可同时访问、有相同 x 值的比特集合称之为探针道，而所有有相同 x 的所有比特则称为柱面。表 6.1 给出 MEMS 存储设备的参数[113]。

表 6.1 MEMS 存储设备参数

每片媒体容量	3.2GB
平均寻道时间	0.55ms
最大寻道时间	0.81ms
最大同时操作针数	1280
最大吞吐率	89.6MB/s

MEMS 存储的研究有很多，有对 MEMS 存储的模型进行研究[114][115][116]，还有关于其参数的设计[117][118]。关于它的应用研究有多种，例如，将 MEMS 设备用做磁盘 Cache，可改善 I/O 性能 3.5 倍[112][113]；在高性能的 RAID 系统，用 MEMS 设备代替 RAID10 中的一个磁盘，可使性价比提高 4-26 倍[119]。

6.2.3 存储节点的写 Buffer

写 Buffer 的目的是缓存即将写磁盘的数据对象，推迟写磁盘，使用它有两个好处：由于 MEMS 存储设备的 I/O 速度远快于磁盘，通过写 Buffer 可实现快速写；写 Buffer 中的数据可被再次使用，从而减少磁盘 I/O 通信，同时，MEMS 的数据迁移采用写回法，利用磁盘的空闲 I/O 带宽写磁盘，磁盘的带宽可较好地被利用。写 Buffer 的结构如图 6.1 所示，写 Buffer 位于控制器的 Cache 和磁盘之间，当 Cache 的数据被替换是，以日志的形式添加到 MEMS 写 Buffer，当数据写入写 Buffer 时认为数据写操作完成。

写 Buffer 中的日志组成一个先进先出的循环队列，使得先写的数据先被清除。为了保证存储节点的写请求立即响应，写 Buffer 并不是等其需要空间才开始写磁盘，

而是在磁盘有空闲 I/O 带宽或 Buffer 的空闲空间小于一定值时进行。

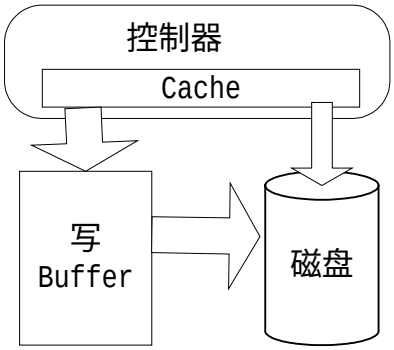


图6.1 存储节点的写Buffer

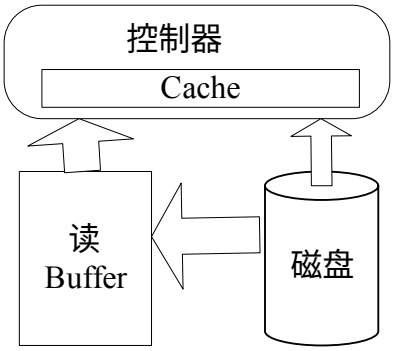


图6.2 存储节点的预取Buffer

6.2.4 存储节点的预取 Buffer

写 Buffer 在优化写负载的同时，通过推迟写对读操作也有一定的性能改善。由于数据访问存在着空间局部性，在控制器 Cache 和磁盘之间设置一个预取 Buffer，达到改善读性能的目的。在预取 Buffer 中，MEMS 存储设备以段的形式组织，一个段包含地址连续的若干块。预取 Buffer 采用命中预取算法，即当控制器要从磁盘读取某一块，就将包含该块在内的整段数据读入预取 Buffer。通过预取，存储节点可以减少读响应时间。

为了保证预取 Buffer 中的数据的利用率，当预取 Buffer 满时采用 LRU 替换算法替换过期数据。这里与写 Buffer 有所不同，当预取 Buffer 完全满才替换数据，这是为了提高预取 Buffer 的空间利用率，而在写 Buffer 中预留空间是为了加快写速度。在写 Buffer 中，采用先进先出替换算法是因为其中保存的是控制器中的过期数据，其使用概率相对小，在这里，使用 LRU 算法，是因为刚读出的数据其使用概率相对高一些。值得说明的是，预取 Buffer 替换数据时并不需要写磁盘，这是因为控制器中对数据的修改是在控制器 Cache 中进行的，而当控制器替换数据是，其修改的数据是通过写 Buffer 写回磁盘的。当然，为了保证数据的一致性，当控制器 Cache 中数据对象修改后，预取 Buffer 中的对应段应被替换。

6.2.5 存储节点控制器 Cache 替换算法

实际上，存储节点中控制器 Cache、写 Buffer（预取 Buffer）和磁盘构成一个三级存储系统。写 Buffer 和预取 Buffer 将存储节点的读写磁盘操作分离，分离的目的

是提高磁盘的读写速度，即通过预取提高磁盘的读速度，而通过写 Buffer 的推迟写提高磁盘写速度。

在存储节点中，由于相连的磁盘通常性能相同，且相同的文件系统使得它们的数据块大小相等，所以，控制器 Cache 替换算法可简单地选择为 LRU 算法。

值得说明的是，写 Buffer 采用的目的是推迟磁盘写操作，而预取是为了提前读出所需的数据，所以，写操作优先于读操作。

6.3 元数据服务器 Cache 方案

如前所述，对象存储系统中客户对存储系统的访问存在两种模式：NAS 模式和三方传送模式。它们各有优缺点，适用于不同的数据对象：NAS 模式适用于小数据对象，而三方传送模式适用于大数据对象。为了充分发挥两种模式的优点，存储系统采用混合模式，即两种访问模式同时并存，根据数据对象的大小，为其选择适当的访问模式，达到优化性能的目的。由于两种访问模式都要首先访问元数据服务器，因此，访问模式选择策略在元数据服务器上实现。

存储系统中的数据对象被多用户共享，数据访问存在着局部性，因此，在元数据服务器上设置 Cache，利用访问的局部性，可以减小 I/O 请求响应时间。元数据服务器 Cache 方案试图将 NAS 模式访问的数据缓存在 Cache 中，而 Cache 不能缓存的数据对象采用三方传送模式，所以，数据访问模式与 Cache 方案密切相关。

6.3.1 元数据服务器 Cache 的设计

元数据服务器中的 Cache 既可以用来缓存元数据，又可以用来缓存数据对象。无论采用哪种模式，I/O 请求都要访问元数据，同时，元数据来自服务器磁盘，设备访问时间相同，所以，元数据的缓存采用通常的 Cache 方案及其替换算法。数据对象的缓存则不同，采用三方传送模式数据不通过元数据服务器，也就不需 Cache，采用 NAS 模式时，数据通过元数据服务器，为了提高数据访问速度，元数据服务器的 Cache 同时缓存 NAS 模式传送的小数据对象。下面的 Cache 设计就是数据对象的 Cache 方案。

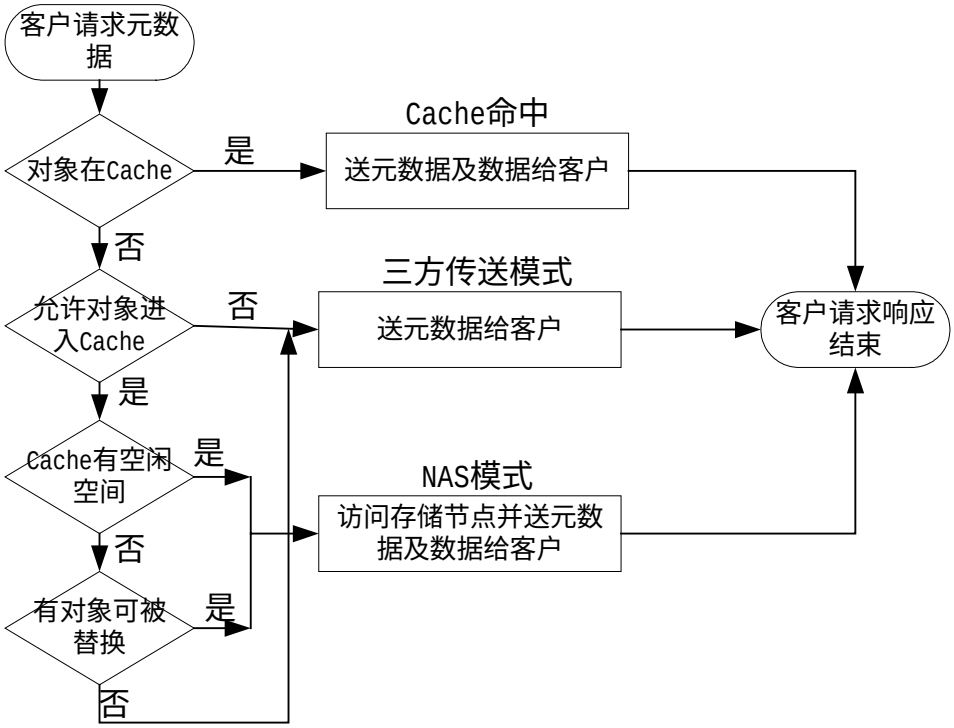


图6.3 元数据服务器Cache方案

Cache 方案由 Cache 准入策略和替换策略组成。准入策略判断数据对象是否可以缓存到 Cache 中，若能，则采用 NAS 模式访问，否则采用三方传送模式访问。替换策略则在需要空闲 Cache 空间时选择替换对象。元数据服务器的 Cache 方案如图 6.3 所示，来自客户的 I/O 请求首先访问 Cache，如果数据对象缓存在 Cache 中，就直接响应客户，响应信息包括数据对象及其元数据；否则读元数据，根据元数据判断数据对象的大小，当数据较小时，元数据服务器就直接访问存储节点，获得数据后响应客户，响应信息也包括数据对象及其元数据；当数据较大时就发送元数据给客户，由客户直接访问存储节点。为了减少 Cache 的查找时间，对元数据进行适当的修改，增加元数据对应的数据对象在 Cache 中的指针，如果数据对象不在 Cache 中，指针设置为空指针。这样查找 Cache 的开销是 O(1)。

在元数据服务器设置 Cache 的目的是减少客户 I/O 请求的响应时间，如果 Cache 命中，可以节省客户端解析元数据开销、连接存储节点开销和从存储节点访问数据开销。当然，与三方传送相比，Cache 不命中时，元数据服务器多了查找 Cache 的开销 O(1)，而且，客户在获得元数据后可以在客户端缓存元数据，再次访问时就不必

访问元数据服务器，这样，不命中导致的额外开销对 I/O 性能的影响就很小。

6.3.2 准入控制策略

Cache 方案的实现的关键是找出影响 Cache 准入的因素，而且，获得这些因素的开销不能太大。在准入控制算法的设计中，我们考虑了以下几种参数：

（1） 对象访问频率(f)：根据访问的局部性，访问越频繁的数据对象越应该缓存在 Cache 中；

（2） 对象的设备访问开销(c)：将从存储节点获取数据时间长的对象缓存在 Cache 中，就可以节省再次访问存储节点的时间，所以设备访问时间长的数据对象应尽可能保存在 Cache 中；

（3） 对象的大小(s)：由于 Cache 的空间有限，对象越大，占用的 Cache 空间越多，所以，应尽可能缓存小数据对象；

（4） I/O 负载(L)：由于所有 I/O 请求都要访问元数据服务器（三方传送模式获取元数据，NAS 模式获取数据及其元数据），I/O 负载增大，元数据服务器成为性能瓶颈，应减少 Cache 缓存数据对象，反之，I/O 负载减少则可增加 Cache 缓存数据对象。

采用第五章替换算法实现的类似方法，为每个数据对象定义一个准入函数如式(6-1)所示：

$$P_i = f_i \frac{c_i}{s_i} \tag{6-1}$$

式（6-1）表明，访问越频繁、访问成本越大、大小越小的对象，越应该缓存在 Cache 中。为此，采用下列式(6-2)所示准入控制标准：

$$P_j > \max\{\delta, \min(P_i)\} \tag{6-2}$$

式（6-2）中的 P_j 为即将访问的数据对象的函数值，P_i 为 Cache 中缓存的数据对象的函数值，δ 为准入函数极限值，它是用来避免准入函数值小的数据对象进入 Cache。例如，Cache 中有一些对象频繁访问而其它对象很少访问，这样对象的最小值会很小，如果没有 δ，准入函数值较小的新数据对象就会进入 Cache。

上述参数的计算如同第五章所述，值得说明的是，参数的计算并不是对 Cache

中的数据对象的统计，而是对所有 Cache 缓存的元数据的统计，当元数据从 Cache 中替换后，其相应参数也随之消失。这样处理的原因是，准入控制讨论的对象不仅仅是 Cache 中的数据对象，而是所有访问存储系统的对象，而所有对象的访问首先必须访问元数据，所以，可通过统计访问元数据的情况来代替数据对象的访问情况。δ 则是通过周期地计算若干个最小准入函数值的平均值，同时，它随着负载 L 的变化而变化，当负载增加，适当增大 δ，减少 Cache 缓存的数据，反之亦然。

6.3.3 Cache 替换算法

当数据对象通过了准入控制检测，就可能进入 Cache，然而，这还不足以保证数据对象进入 Cache，只有当 Cache 中有空闲空间时才能缓存数据对象。当 Cache 没有空闲空间时，就必需替换其中的某些对象，Cache 的替换则与准入控制密切联系。替换策略如下：

（1）将 Cache 中的数据对象的准入控制函数按升序排列：{1，2，…，n}；

（2）找出最小的序号 m，使得：$s_1+s_2+\ldots s_m \geq s_j$，其中 j 为即将访问的对象；

（3）如果 $P_m \leq P_j$，则数据对象 j 替换这 m 个数据对象，否则 Cache 拒绝该对象。

按照替换算法，当对象的准入函数值大于 Cache 中的一定对象的准入函数值，且这些数据对象的大小和大于待访问的数据对象，就可以缓存该对象，相应地采用 NAS 模式访问，相反，则采用三方模式访问。

6.4 客户端 Cache 方案

存储系统的客户端的特点与元数据服务器和存储节点有所不同，其 Cache 方案也有所区别。下面首先分析客户端数据对象的特点，在此基础上设计 Cache 方案。

6.4.1 客户端数据访问特点

客户端用户首先访问元数据服务器，并根据数据对象的特点采用两种模式访问存储节点，其数据访问有以下特点：

（1） 客户访问存储系统是以数据对象为单位的，对象大小是可变的；不同的对象来自不同的存储节点，它们的设备访问时间（包括网络传输时间）是不同的。

（2） 客户端访问的数据对象可能来自不同的存储节点，其空间局部性较弱，

但是，同一数据对象可能被多次访问，存在着时间局部性。

（3） 客户访问存储系统分两种模式：NAS 模式和三方传送模式。不同的访问模式适用于不同的数据对象，三方传送适合于大数据对象，而 NAS 模式适用于小数据对象。然而，选择传输模式时，数据对象的大小是相对的，因此，对于采用三方传送的数据对象最好通过积累使其传送的数据对象较大。

（4） 根据元数据服务器的 Cache 方案，采用 NAS 模式的数据对象在元数据服务器中有副本，而采用三方传送模式的数据对象则没有，为了充分利用 Cache，客户端两种模式访问的数据对象应采用不同的 Cache 方案。

6.4.2 客户端 Cache 的设计

根据客户端数据访问的特点，将 Cache 设置在文件系统与网络接口之间。如图 6.4 所示，它由 RAM Cache 和日志磁盘组成一个两层的层次 Cache 结构[120][121][122][123]。日志磁盘将数据对象及其元数据一起保存在磁盘中，采用日志形式的目的是便于数据恢复。日志磁盘既可以由硬盘实现，也可以采用 MEMS 存储设备来实现。客户访问对象存储系统时首先通过网络从元数据服务器获取数据对象的元数据。如前所述，采用 NAS 模式，数据随同元数据一起返回客户端，在三方传送模式中，返回客户端的仅仅是元数据。对元数据进行修改，使得客户端从元数据服务器中获得的元数据携带有数据传输模式，传输模式不同，意味着 Cache 方案不同。所有的数据连同其元数据缓存到 RAM Cache，但日志磁盘只保存三方传送模式访问的数据，通过日志磁盘将三方传送的小数据对象积累为大数据对象。为了减少日志磁盘开销，如同元数据服务器中对元数据的修改一样，在客户端保存的元数据中加上其在日志磁盘中的指针，这样，日志磁盘的查找只需一次计算，减少了其查找时间。对于未保存在日志磁盘中的数据对象（包括采用 NAS 模式访问的对象）其指针为空。日志磁盘之所以只保存采用三方传送模式传送的数据而不保存采用 NAS 模式传送的数据，是因为元数据服务器中有采用 NAS 模式传送的数据对象的服务本，不保存这些副本既可以节约日志磁盘的存储空间，又不影响访问局部性的利用，同时，副本的减少可减少数据不一致性的可能。值得说明的是，元数据的缓存是和数据对象分开的，这里只讨论数据对象的 Cache 方案。

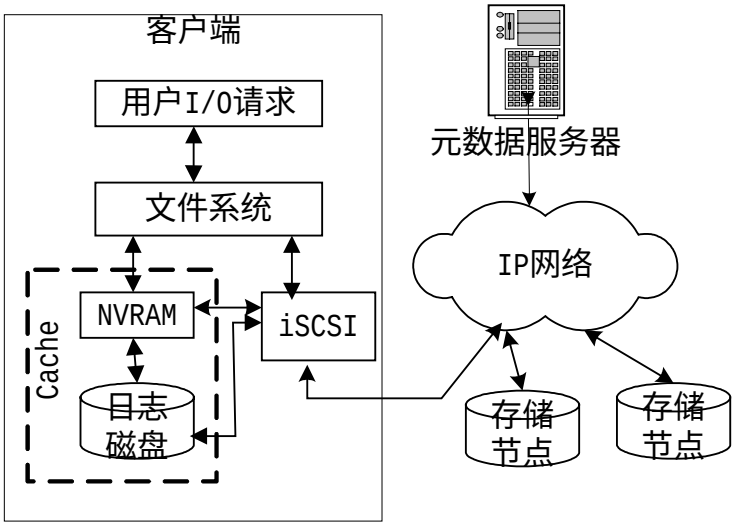


图6.4 客户端Cache体系结构

设置客户端 Cache 后，数据的访问过程如下：(1）用户的 I/O 请求通过文件系统访问对象存储系统时，首先访问 RAM Cache，如果其对应的元数据存在，则相应的数据对象也可能存在 Cache 中；(2）然后看元数据的日志指针是否为空，若为空，则在 RAM Cache 中查找数据对象，否则根据指针在日志磁盘中读数据到 RAM Cache 并响应用户；(3）若 Cache 中不存在元数据，再访问元数据服务器，这时，采用 NAS 模式返回的是数据及其元数据，将其缓存在 RAM cache 中；采用三方传送模式时返回的是元数据，根据元数据直接访问存储节点，返回的数据同样缓存在 RAM Cache 中。

RAM Cache 和日志磁盘的容量是有限的，当新的数据对象要进入时，必须采用一定的替换算法由于两者的特点不同，其替换算法有所不同。

在 RAM Cache 中，采用 NAS 模式的数据对象来自元数据服务器（元数据服务器中有 Cache)，而采用三方传送模式的数据对象则来自日志磁盘（第一次除外)，它们的设备访问时间基本固定，所以不必考虑设备访问成本，替换算法采用常用的 LRU 算法即可。

日志磁盘中的数据对象则来自不同的存储节点，它们的访问成本各不相同，其替换算法必须考虑到设备访问成本的不同，这里选择第五章所设计的 LAT 算法。

6.5 性能评价

为了简化实验，将上述 Cache 方案的性能评价分开进行，并采用不同的方法，从不同的方面比较它们的性能。

6.5.1 存储节点 Cache 性能

由于 MEMS 存储设备无法获取，采用 DiskSim 来评价存储节点性能。MEMS 作为一个自定义磁盘，其参数如表 6.1 所示。磁盘设置为：20GB 的空间，平均寻道时间为 6ms，吞吐率为 60MB/s。存储节点控制器 Cache 大小设置为 4MB，MEMS 空间大小为 100MB，为磁盘空间的 5%，其中，50MB 为写 Buffer，另 50MB 用于读 Buffer。

负载采用人工合成负载，其中读写请求各占 50%，假定 I/O 请求按照泊松分布。为了比较读写 I/O 请求特点对性能的影响，给定三种类型的负载 1、2、3，它们顺序读的请求分别占 20%、50%和 80%，负载 1 顺序读请求占 20%，表示负载中随机读写为主，即随机负载，相反，负载 3 顺序读请求占 80%，意味着负载以顺序读写为主，即顺序负载。实验首先设置数据段大小为 150MB，比较不设置 Buffer 和设置读 Buffer、写 Buffer 和读写 Buffer 的性能，如图 6.5 所示，设置 Buffer 的平均响应时间小于不设置 Buffer 的平均响应时间，这是因为：写 Buffer 通过推迟写缩短了写请求的响应时间，而读 Buffer 通过预取改善了读性能。但是，Buffer 对三种负载的性能改善则各不相同，写 Buffer 对三种负载的性能改善差不多，读 Buffer 使负载 3 的性能提高最多，而负载 1 的性能提高最少。这是由于读 Buffer 主要利用空间局部性来提高其命中率，顺序读负载具有较强的空间局部性，相反，随机读的空间局部性则很弱，所以，顺序读可以通过预取降低读磁盘时间，相反，预取使随机读的响应时间变长，而命中率很低，它不仅不能提高随机读的性能，还可能使其性能降低。

Buffer 中的数据段大小对其性能也有影响，图 6.6 为 Buffer 的性能随着数据段大小的变化而变化关系。从图中可以看出，随着 Buffer 中的数据段的增大，负载 1 和 2 的性能下降，而负载 3 的性能增加。这是因为：写 Buffer 利用磁盘 I/O 空闲时间写磁盘，数据段的大小对其影响不大，而随着数据段的增大，读 Buffer 预取数据增多，

负载 1 中随机 I/O 请求更多，命中率下降，故性能下降；相反，负载 3 中连续 I/O 请求更多，连续预取更多数据可降低 I/O 响应时间，而命中率不会下降，故性能随之提高。

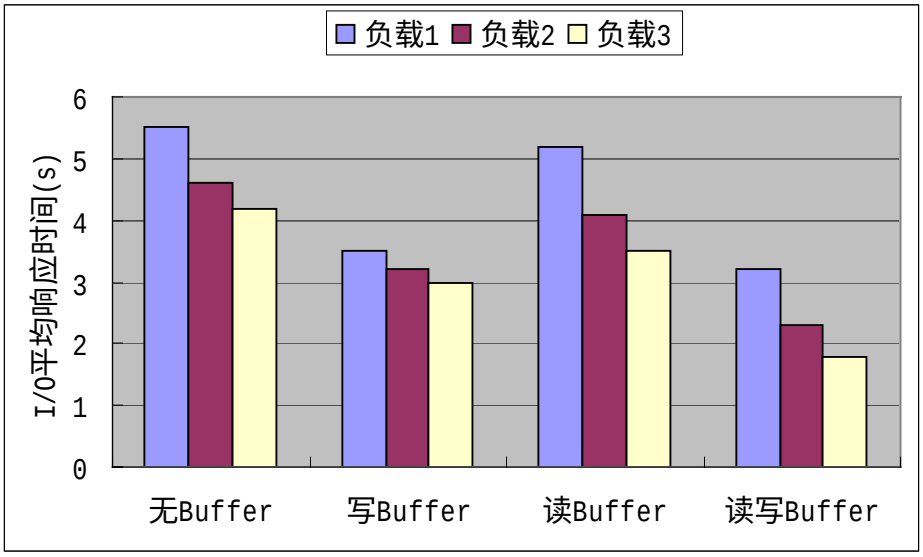


图6.5 MEMS Buffer对I/O性能的影响

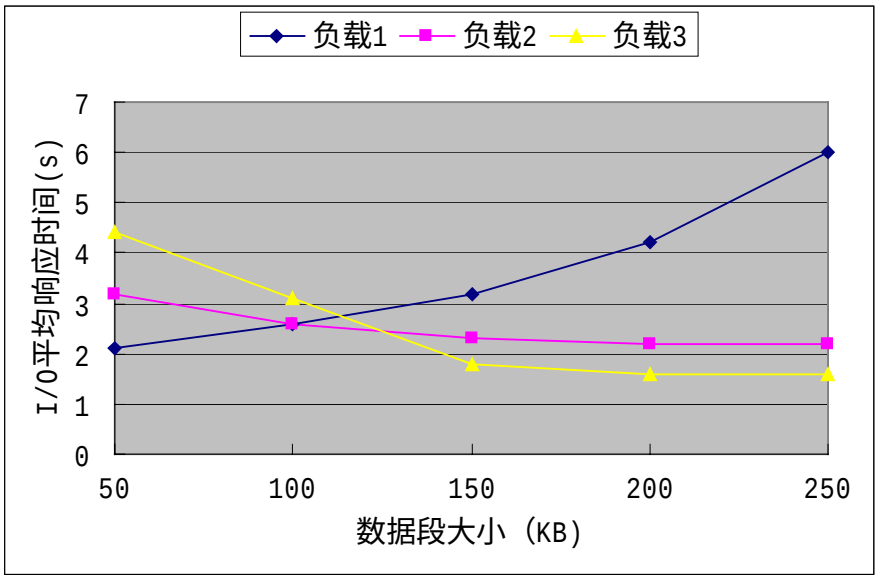


图6.6 MEMS Buffer性能与数据段的关系

6.5.2 元数据服务器及客户端 Cache 性能评价

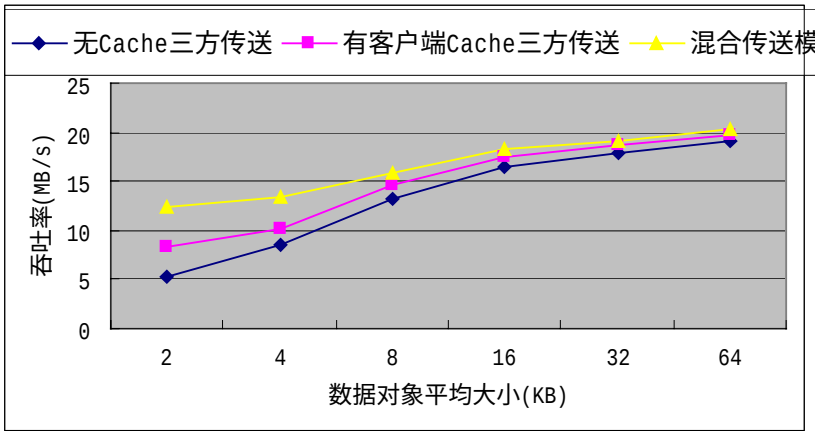
元数据服务器 Cache 方案可在三方传送的对象存储系统的基础上实现，其基本

思路是在元数据服务器上加 Cache 准入控制算法和 Cache 替换算法，同时对元数据进行修改，使其记录数据对象的元数据被访问的次数及相关信息，就可以实现元数据服务器 Cache 方案。

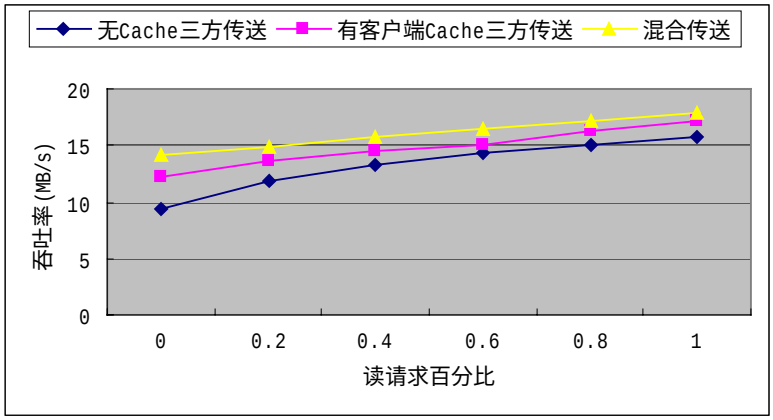
实验中采用 iSCSI 协议实现对象存储系统的数据传输，iSCSI 目标端作为存储节点，iSCSI 启动端作为元数据服务器，对启动端的 iSCSI 协议进行修改，使其在接到客户端的 I/O 请求根据数据对象大小决定数据传输模式：NAS 模式和三方传送模式，这样，既可以实现三方传送模式，也可以实现元数据服务器的 Cache 方案。客户端的 Cache 实现是在 DCD 的基础上实现的[123]，其主要思路是修改日志磁盘的替换算法。

实验中服务器、客户和存储节点的配置为： P4 1500 CPU，256MB RAM，千兆以太网卡，操作系统采用 RedHat7.1；SCSI 卡为 AHA-2940UW，硬盘为 Seagate ST318437LW SCSI 磁盘，其最大传输率为 100MB/S。

实验比较无 Cache 的三方传送模式、有客户端 Cache 的三方传送模式和混合传送模式三种情况下的 I/O 性能。图 6.7 是在客户端通过 Iometer 测试三种方案的 I/O 吞吐率。图 6.7(a)测量读写请求比例为 1：1 时吞吐率与平均数据对象大小的关系，从图可以看出，对于小数据对象，无 Cache 时性能最差，有客户端 Cache 次之，而混合模式最好；随着数据对象大小的增加，三种设置的吞吐率都提高，但无 Cache 时提高最快，而混合传送提高最慢，这是因为：客户端 Cache 将小数据对象缓存，一方面能重复使用，另一方面日志磁盘可将小数据对象积累为大数据对象，故对小数据对象而言，其性能好于无 Cache 的情况，而混合传送模式就是为了解决数据的共享的，故其对小数据对象性能最好；一个数据对象的协议开销一定，所以，随着数据对象的增大，单位数据对象的协议开销则减少，性能随平均数据对象大小的增大而提高。图 6.7(b)为固定数据对象平均大小(8KB)，吞吐率随读请求的百分比变化的关系，从图看出，三种设置的读性能好于写性能，但混合传送模式的性能最好，这是因为，写必须等存储节点数据写进磁盘才能响应，而读只需将数据读到客户端内存；有 Cache 时写性能比无 Cache 时性能提高很多，这是因为缓存数据可以利用空闲带宽进行写操作的。



(a)数据对象大小对性能的影响



(b)请求中读占百分比对性能的影响

图6.7 Cache对Ⅰ／Ｏ性能的影响

6.6 本章小结

Cache 是提高网络存储 I/O 性能的最有效方法之一，而 Cache 的设置必须考虑到存储的特点，只有根据存储的特点设置 Cache，才能保证 Cache 最好地改善存储系统的性能。对象存储系统中存在着三种存储实体：存储节点、元数据服务器和客户端，它们的特点各不相同。本章在分析三种存储实体的特点的基础上，分别设计 Cache 方案。根据存储节点中数据读写磁盘慢的特点，采用 MEMS 存储设备设置预取 Buffer 和写 Buffer，将读写 Buffer 分离，通过预取 Buffer 预取提高读速度，而通过写 Buffer 推迟写操作，由于 MEMS 的非挥发性，数据写进 MEMS 中就完成了写操作，提高了写速度。元数据服务器的主要任务是管理和组织存储系统，而用户对存储系统的

访问首先必须访问元数据服务器，从中获得元数据，在此之后，访问存储节点有两种模式：NAS 模式和三方传送模式。结合两种访问模式的特点，在元数据服务器端设置 Cache，通过 Cache 的准入策略和替换算法，使存储系统能混合使用两种模式传输数据，即小数据对象采用 NAS 模式，数据保存在元数据服务器的 Cache 中，充分利用数据访问的局部性，达到提高性能的目的；相反，考虑到 Cache 空间的利用率，大数据对象则不进入元数据服务器 Cache，而是用户获得元数据后直接访问存储节点。用户端通常远离存储节点和元数据服务器，用户端的 Cache 可保证利用网络的空闲带宽进行数据传输，结合两种传输模式特点，在用户端设置层次 Cache 对不同模式传输的数据采用不同的缓存策略，从而达到用户 Cache 和元数据服务器 Cache 合作使用，达到提高 I/O 性能目的。实验显示，三种 Cache 从不同的方面提高存储系统的 I/O 性能。

7 全文总结

计算机应用的发展对存储系统提出新的要求：容量大、速度快、可靠性高、安全性好以及可扩展性好。满足这些要求可通过多种技术来实现，其中，在存储系统中采用合适的数据组织算法是一种行之有效的手段。

数据组织必须结合存储系统的特点，才能达到性能优化的目的。本文通过对对象存储系统的体系结构和特点的分析，从提高存储系统速度出发，研究了存储系统的存储空间和数据传输路径上的数据组织算法及相关问题。

7.1 本文取得的主要研究成果

本文取得的主要研究成果和创新表现在以下几个方面：

1） 分析对象存储系统的体系结构。通过对 NAS、SAN 和对象存储的比较分析，可以看出对象存储兼有 NAS 和 SAN 的优点；对象存储的体系结构和软件结构，使得对象存储系统中有两种数据传输模式：NAS 模式和三方传送模式。理论分析和实验显示两种模式各自适用于不同的应用：NAS 模式适用于小数据对象的传输，而三方传送模式适用于大数据对象的传输。对象存储的丰富的语义使得对象存储便于实现存储安全和基于 QoS 的 I/O 服务。对象存储的这些特点正是数据组织算法的基础。

2） 提出一种海量对象存储系统中数据的组织算法和系统扩展时数据重组算法。海量存储系统中的数据组织算法充分利用多存储节点的并行性，达到提高 I/O 性能的目的。考虑到海量存储系统的空间大、数据对象多，算法的设计必须保证算法的时间和空间开销尽可能小。数据的重组算法则保证数据组织算法能适应存储系统的扩展，从而保证存储系统的扩展在增大容量的同时提高速度。

3） 提出了一种基于 QoS 的数据迁移算法。现代应用要求存储系统能提供基于 QoS 的 I/O 服务，对象存储的出现为这种要求的实现提供了可能。然而，数据的迁移或多或少地对存储系统的 I/O 服务有影响。通过对迁移的附加收益的定义，将迁移任务细分为小的迁移请求，将迁移请求和 I/O 请求同等对待，建立一个迁移模型，并分析其实现的相关问题。该迁移模型在保证迁移实现的同时，尽可能少地影响存储系

统的可用性。

4） 提出了两种 Cache 替换算法：LAT 算法和 WLFRU 算法。存储系统的数据传输路径设置 Cache 可改善 I/O 性能，而 Cache 的替换算法是存储系统性能改善的关键。通过对 Cache 存储系统的性能的分析，得出结论：Cache 存储系统的性能提高必须同时考虑到 Cache 命中率和设备的访问成本。按照这一思路，提出了 LAT 算法和 WLFRU 算法。两种算法对存储系统的性能有所提高。

5） 提出一种对象存储系统的 Cache 方案。分析对象存储系统的三种存储实体的特点，结合各自的特点提出相应的 Cache 方案。在存储节点，通过设置写 Buffer 达到推迟写的目的，其替换算法采用先进先出算法；通过设置预取 Buffer 减少读盘时间，相应的替换算法采用 LRU 算法。元数据服务器的 Cache 方案则与数据传输模式紧密相连，通过准入控制算法和替换算法，使得元数据服务器在负载轻时缓存小数据对象，数据访问采用 NAS 模式，而对大数据对象和负载重时的小数据对象则采用三方传送模式，相应地，元数据服务器也没有缓存问题。同样客户端的 Cache 方案也和数据传送模式有关，两种方案的数据对象都通过主存 Cache 缓存，相应的替换算法为 LRU 算法。不同的是，采用三方传送的数据对象除了主存 Cache 外还采用日志磁盘将小数据对象积累为大数据对象，以达到提高性能的目的，相应的替换算法为 LAT 算法。

7.2 进一步研究工作的考虑

不同的体系结构的数据组织算法有所不同，本文只讨论了集中控制的存储系统中的存储空间的数据组织算法，分散控制的存储系统的数据组织没有设计；同时，存储系统中的 Cache 合作和 Cache 的一致性问题没有考虑。下一步主要从以下几方面进行研究：

1）分散控制存储系统中的数据组织算法的研究；

2）数据迁移方案的实现；

3）合作 Cache 方案的设计；

4）Cache 一致性协议的设计。

致　谢

岁月如梭，几年的博士学习生活即将结束。在即将离校之际，深深感谢我的导师谢长生教授多年来对我无微不至的关怀和悉心的指导。导师渊博的知识、严谨的治学态度、开拓的学术思想、崇高的敬业精神、高瞻远瞩的眼光、诲人不倦的师德、宽厚随和的人格魅力以及睿智儒雅的风范给我留下了深刻的印象，并将影响着我的一生。在我即将完成学业走向工作岗位之际，向谢老师致以世间最美好的祝福和最真诚的谢意！

在本课题的研究过程中，得到了许多老师的大力帮助，在此表示深深的谢意。感谢裴先登教授、林安老师、黄浩老师、王海卫老师、胡迪青老师、曹强老师、万继光老师、吴非老师、谭志虎老师、刘瑞芳老师、柳少云老师以及其它许多老师的关心和帮助，感谢他们为大家营造了一个良好的工作环境以及和谐融洽的学术氛围。

在本课题的研究过程中，还得到了博士后黄建忠、蔡斌、李怀阳、博士韩德志、罗东健、易法令、刘艳、董小明、张成峰、吴伟、刘海华、赵振、任劲、钟海峰、李博等的帮助和支持，在此表示深深的谢意！还将谢意将送给实验室众多的同学，我们在一起度过了一段难忘的日子。

深深地感谢我的家人对关怀和支持，我今天的成就和他们的辛勤付出是分不开的。尤其要感谢我的妻子余丽珍，正是她对我的理解、鼓励以及在攻读博士期间默默地为我付出，才使我能安心学习，顺利地完成学业。

最后，感谢审阅论文的老师在百忙之中审阅我的论文。感谢答辩委员会的各位老师出席我的论文答辩。

参考文献

[1] R. J. T. Morris, B. J. Truskowski．The evolution of storage systems． IBM Systems Journal, 2003, Vol. 42(2): 205-217

[2] Marc Farley．Building storage networks．McGraw-Hill Osborne Media, 2002

[3] 程鹏. 磁盘阵列系统结构及性能评价. 博士学位论文，华中科技大学计算机学院，1999,7

[4] N. Allen. Don't Waste Your Storage Dollars: What You Need to Know. Research Note, Gartner Group. March 2001

[5] E. Lamb. Hardware Spending Sputters. Red Herring. June 2001. 32~33

[6] J. Gray. A Conversation with Jim Gray. Queue, June 2003, Vol.1(4):8~17

[7] 郑纬民, 汤志忠. 计算机系统结构. 清华大学出版社, 2000, 12

[8] Hans, J. Coufal．Non-Magnetic Data Storage Principles, Potential, and Problems. File and Storage technologies (FAST'02)．CA, USA ．January 28, 2002. 67-75

[9] Mustafa Uysal, Arif Merchant．Using MEMS-Based Storage in Disk Arrays．File and Storage technologies (FAST'03). San Francisco, CA. March-April 2003.130-146

[10] Tim Schooler,Plasmon．Optical Disk．19th IEEE Symposium on Mass Storage Systems, Maryland, USA, April 2002. 271-283

[11] Auspex White Paper: A Storage Architecture Guide. 2003. 3-7

[12] R. Hernandez, C. Kion, and G. Cole. IP Storage Networking: IBM NAS and iSCSI Solutions, Redbooks Publications (IBM): SG24-6240-00. 2001. 56-69

[13] G. A. Gibson and R. V. Meter. Network Attached Storage Architecture. Communications of the ACM, 2002, Vol. 43(11): 37- 45

[14] B. Phillips. Have Storage Area Networks Come of Age. Computer, 1998, Vol 31(7): 10-12

[15] S. Milanovic and Z. Petrovic. Building the Enterprise-wide Storage Area Network. In Proceeding of International Conference on EUROCON'2001. College Park. July 2001. 136 -139

[16] Anritsu Company. IP-SAN: techniques for Storage Networking over IP. http://www.us.anritsu.com/networking. October 2001, USA

[17] 谢长生，罗益辉. IP-SAN 的研究与设计. 小型微型计算机系统，2005，6，Vol.26(No.6): 912-915

[18] National Storage Industry Consortium. Network Attached Secure Disk Project. http://www.nsic.org/nasd/index.html

[19] Technical Committee of INCITS. SCSI Storage Interfaces. http://www.t10.org/

[20] A. Shafrir. OSD-Initiator. IBM Haifa LAB. http://sourceforge.net/projects/osd-initiator

[21] 刘朝斌. 存储虚拟化关键技术研究. 博士学位论文，华中科技大学计算机学院，2004

[22] Gartner. Inc. Forecast: Storage Management Software, Worldwide, 2001-2008. Market Research Report. Gartner Group, April 2004, http://www.gartner.com/

[23] G. R. Ganger, J. D. Strunk, and A. J. Klosterman. Self-* Storage: Brick-based Storage with Automated Administration. Technical Report CMU-CS-03-178. CMU, Aug. 2003

[24] 黄建忠. 三方传输模式下网络存储的安全性研究. 博士学位论文，华中科技大学计算机学院, 2005

[25] Garth A. Gibson, David F. Nagle, Khalil Amiri, et.al. File Server Scaling with Network Attached Secure Disks. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems. USA: ACM Press, 1997. 272-284

[26] M. Kallahalla, E. Riedel, R. Swaminathan, et.al. PLUTUS: Scalable Secure File Sharing on Untrusted Storage. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2003. 29~42

[27] E. Goh, H. Shacham, N. Modadugu, et.al. SiRiUS: Securing Remote Untrusted Storage. In Proceedings of the 10th Network and Distributed System Security Symposium. USA: Internet Soc, 2003. 131~145

[28] M. Blaze. A Cryptographic File System for UNIX. In Proceeding of 1st ACM Conference on Communications and Computing Security. USA: ACM Press, 1993.

9~16

[29] D. Mazieres, M. Kaminsky, M. F. Kaashoek, et.al. Separating Key Management from File System Security. In 17th ACM Symposium on Operating Systems Principles. USA: ACM Press, 1999. 124~139

[30] Todd Rief. Information Lifecycle Management—Storage Management. Computer Technology Review. 2003, 8

[31] Glenn Drhodes. Lifecycle Management for Storage Consolidation. Computer Technology Review. 2003, 10

[32] Garth Gibson and Peter Corbett. PNFS Problem Statement. IETF Internet Draft, 2004

[33] 王建军. 操作系统与硬盘的数据组织. 计算机时代, 2003,2

[34] Hong Tang, Tao Yang. An Efficient Data Location Protocol for Self-organizing Storage Clusters. In Proceedings of the 2003 ACM/IEEE conference on Supercomputing, NW Washington, DC USA: IEEE Computer society, 2003. 695-708

[35] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: Ascalable Wide-area Web Cache Sharing Protocol. IEEE/ACM Transactions on Networking, 2000, Vol.8(3): 281-93

[36] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton et al. OceanStore: An Architecture for Global-scale Persistent Storage. ACM SIGOPS Operating Systems Review, 2000,Vol.34(5): 190-201

[37] E. Thereska, M. Abd-El-Malek, J. J. Wylie, et al. Informed Data Distribution Selection in a Self-predicting Storage System. In Proceedings of IEEE International Conference on Autonomic Computing, 13-16 June 2006. 187-198

[38] V. Sundaram, T. Wood, P. Shenoy. Efficient Data Migration in Self-managing Storage Systems. In Proceedings of IEEE International Conference on Autonomic Computing, 13-16 June 2006. 297-300

[39] T. Anderson, M. Dahlin, J. Neefe, et al. Serverless Network File Systems. ACM SIGOPS Operating Systems Review 1995, Vol.29(5): 41-79

[40] E. Lee and C. Thekkath. Petal: Distributed Virtual Disks. In Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS96), Cambridge, MA, 1996. 84-92

[41] Witold Litwin, Marie-Anne Neimat and Donovan A. Schneider. LH* -A Scalable, Distributed Data Structure. ACM Transactions on Database Systems, 1996, Vol.21(4): 480-525

[42] D. Karger, E. Lehman, T. Leighton, et al. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In Proc. of the 29th ACM Symp. on Theory of Computing (STOC) , 1997. 654–663

[43] André Brinkmann, Kay Salzwedel, Christian Scheideler. Compact, Adaptive Placement Schemes for Non-uniform Requirements. SPAA 2002. 53-62

[44] Yuri Breitbart, Radek Vingralek, Gerhard Weikum. Load Control in Scalable Distributed File Structures. Distributed and Parallel Databases, 1996 Vol. 4(4): 319-354

[45] John L. Hennessy, David A. Patterson. Computer Architecture: A Quantitative Approach. Third Edition. Beijing: China Machine Press, 2002

[46] H. Jungho. Two-level Cache for Distributed System in RAID 5 Disk Controller. In Proceedings of 18th International Conference on Systems Engineering, 16-18 Aug. 2005. 64-69

[47] E. J. O'Neil, P. E. O'Neil, and G. Weikum. An Optimality Proof of the LRU-K Page Replacement Algorithm. J. ACM, 1999, Vol. 46(1): 92–112

[48] S. Zhang, I. Chen, M. Goldszmidt, et al. Ensembles of Models for Automated Diagnosis of System Performance Problems. In Proceedings of International conference on Dependable Systems and Networks (DSN), July 2005. 644-653

[49] S. Jiang and X. Zhang. LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In Proc. ACM SIGMETRICS Conf., 2002. 31-42

[50] Ekow Otoo, Frank Olken and Arie Shoshani. Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids. In Proceedings of the IEEE/ACM SC 2002 Conference, November, 2002. 1-15

[51] D. Lee, J. Choi, J.-H. Kim et al. RFU: A Spectrum of Policies That Subsumes the least recently Used and least Frequently Used Policies. IEEE Trans. Computers, 2001, Vol. 50(12): 1352–1360

[52] Y. Zhou and J. F. Philbin. The Multi-queue Replacement Algorithm for Second Level

Buffer Caches. In Proc. USENIX Annual Tech. Conf. (USENIX 2001), Boston, MA, June 2001. 91–104

[53] Kai Hwang, Zhiwei Xu. Scalable Parallel Computing Technology, Architecture, programming．McGraw-Hill 1998

[54] Alain Azagury, Vladimir Dreizin, et al. Towards an Object Store. In Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), 2003. 124-133

[55] G. Gibson, D. Nagle, K. Amiri, et al. File systems for network-attached secure disks, 1997

[56] R. O. Weber. SCSI Object-Based Storage Device Commands-2(OSD-2). Document Number: ANSI/INCITS 400-2004, Oct. 2004, http://www.t10.org/drafts.htm

[57] PANASAS, Ins. White Paper. Object Storage Architecture, October 2003, http://www.panasas.com/

[58] Cluster File Systems, Inc. Lustre File System, http://www.lustre.org/

[59] 罗益辉，谢长生．基于三方传送的 USN 的设计和实现．计算机工程，2006, Vol.32(3): 125-127

[60] Intel iSCSI Protect 2002. http://sourceforge.net/projects/intel-iscsi

[61] Kevin Klein Osowski，Tom Ruwart，David J. Lilja. Communicating Quality of Service Requirements to an Object-Based Storage Device. In Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05), 2005. 128-135

[62] Yingwu, Zhu Yiming Hu. SNARE: A Strong Security Scheme for Network-Attached Storage. In Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS'03). 2003. 250 - 259

[63] K. T. Pollack，S. M. Uttamchandani. Genesis: a Scalable Self-evolving Performance Management Framework for Storage Systems. In Proceedings of 26th IEEE International Conference on Distributed Computing Systems, 04-07 July 2006.33-33

[64] Yihui LUO, Changsheng XIE, etc. A Proxy Cache Scheme of Network Storage. In Proceedings of the Second International Conference on Embedded Software and Systems, 2005. 578-582

[65] B. Hong, S. A. Brandt, D. D. E. Long, et al. Zone-based Shortest Positioning Time First Scheduling for MEMS-based Storage Devices. In Proceedings of the 11th International Symposiumon Modeling, Analysis, and Simulation of Computer and Telecommunication Systems(MASCOTS'03), Orlando, FL, 2003. 104–113

[66] Aameek Singh, Abhishek Trivedi, Krithi Ramamritham, et al. PTC: Proxies that Transcode and Cache in Heterogeneous Web Client Environments. World WideWeb, 2004 Journal, Vol. 7(1): 7-28

[67] Shudong Jin, Azer Bestavros and Arun Iyengar. Accelerating Internet Streaming Media Delivery Using Network-aware Partial Caching. In Proceedings of the 22nd International Conference on Distributed Computing Systems(ICDCS'02), 2002. 153-160

[68] B. Awerbuch, C. Scheideler. Consistent and compact data management in distributed storage systems. In Proc.16th ACM Symposium on Parallel Algorithms and Architectures (SPAA). New York, USA: ACM Press, 2004. 44-53

[69] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, Distributed Data Placement Strategies for Storage Area Networks. In Proc. of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA), New York, USA: ACM Press, June 2000. 119-128

[70] C. Wu, R. Burns. Improving I/O Performance of Clustered Storage Systems by Adaptive Request distribution. In Proceedings of 15th IEEE International Symposium on High Performance Distributed Computing, 2006. 207-217.

[71] Hong Tang, Tao Yang. An Efficient Data Location Protocol for Self-organizing Storage Clusters. In Proceedings of the 2003 ACM/IEEE conference on Supercomputing, 2003. 53-63

[72] Ethan L. Miller R. J. Honicky. A Fast Algorithm for Online Placement and Reorganization of Replicated Data. In Proc. 17th International Parallel and Distributed Symposium (IPDPS 2003), Nice, France: IEEE CS, 2003. 57-66

[73] Hari Balakrishnan, M. Frans Kaashoek, et al. Looking up Data in P2P System. Communications of the ACM, 2003, Vol. 46(2): 43-48

[74] P. D. Coddington. Random Number Generators for Parallel Computers. NHSE Review, 1996, Vol. 1(2): 1-26

[75] Richard P. Brent. Fast and Reliable Random Number Generators for Scientific Computing. In Proceedings of the PARA'04 Workshop on the State-of-the-Art in Scientific Computing, Lyngby, Denmark, 2004. 20-23

[76] B. A. Wichmann and I. D. Hill. Algorithm AS 183: An efficient and portable pseudo-random number generator. Applied Statistics, 1982, Vol. 31(2): 188–190

[77] L. Fan, P. Cao, J. Almeida, et al. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. IEEE/ACM Transactions on Networking, 2000, Vol. 8(3): 281-293

[78] Y. Hu and Q. Yang. DCD— Disk Caching Disk: A New Approach for Boosting I/O Performance. In Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996. 169–178

[79] I. Stoica, R. Morris, D. Karger, et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. San Diego, CA: ACM Press, 2001.149-160

[80] K. Hildrum, J. Kubiatowicz, S. Rao, et al. Distributed Object Location in a Dynamic Network. In Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures(SPAA02) , August 2002. 41-52

[81] M. Castro, P. Druschel, A. Ganesh, et al. Security routing for Structured Peer-to-peer Overlay Networks. In Proceedings of the 5th Symposiumon Operating Systems Design and Implementation(OSDI02), December 2002. 299-314

[82] Yihui LUO, Changsheng XIE, Chengfeng ZHANG. A Security Scheme for United Storage Network. GCC Workshops 2004, October, 2004. 238-245

[83] M.K.Lakhamraju, R. Rastogi, S. Seshadri, and S. Sudarshan. Online Reorganization in Object Databases. In Proc. ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2000.58-69

[84] M. L. Lee, M. Kitsuregawa, B. C. Ooi, et al. Towards Self-tuning Data Placement in Parallel Database Systems. In Proc. ACM SIGMOD, New York USA: ACM Press, 2000. 225-236

[85] A. Bar-Noy, R. Bar-Yehuda, A. Freund, et al. A Unified Approach to Approximating Resource Allocation and Scheduling. In Proc. 32nd ACM Symp. on Theory of Computing, New York: ACM Press 2000.735-744

[86] A. Verma, and S. Ghosal. On Admission Control for Profit Maximization of Network Service Providers. In Proc. of the Twelfth International World Wide Web Conference, Budapest, Hungary , May 2003.128-137

[87] R. Intel Corporation. Intel iSCSI Reference Implementation. http:// www. intel. com/ technology/ computing/ storage/iscsi/ index.htm

[88] R. O. Web ster. Information Technology—SCSI Object-Based Storage Device Commands(OSD). February 2004. Rev. 9

[89] M. de Miguel, J. Ruiz, and M. Garcia. QoS-Aware Component Frameworks. In Proceedings of Tenth IEEE International Workshop on Quality of Service, May 2002. 161–169

[90] Z. Wang. Internet QoS: Architectures and Mechanisms for Quality of Service. Morgan Kaufmann Publishers, 2001

[91] J. C. Wu and S. A. Brandt. Storage Access Support for Soft Real-Time Applications. In Proceedings of 10th IEEE Real-Time and Embedded Technology and Applications Symposium(RTAS'04), Toronto, Canada, May 2004.164-171

[92] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In Proc. Of USENIX FAST, Berkeley, CA USA: USENIX Association, 2002. 21-35

[93] C. R. Lumb, A. Merchant, G. A. Alvarez. Facade: Virtual Storage Device with Performance Guarantees. In Proc. of USENIX FAST 2003, San Francisco, CA, Mar. 2003.131-144

[94] D. D. Chambliss, G. A. Alvarez, P. Pandey, et al. Performance Virtualization for Large-Scale Storage Systems. In Proc. Of 22nd Intl. Symp. On Reliable Distributed Systems, 2003.109-118

[95] E. Anderson, J. Hall, J. D. Hartline, et al. An Experimental Study of Data Migration Algorithms. In Proc. Workshop on Algorithmic Engineering, 2001. 145-158

[96] I. Dramaliev and T. Madhyastha. Optimizing Probe-based Storage. In Proceedings of the Second USENIX Conference on File and Storage Technologies(FAST), San Francisco, CA, Mar. 2003. 103–114

[97] J. L. Griffin, S. W. Schlosser, G. R. Ganger, et al. Modeling and Performance of MEMS-based Storage Devices. In Proceedings of the 2000 SIGMETRICS

Conference on Measurement and Modeling of Computer Systems, June 2000. 56–65

[98] X. Chen, P. Mohapatra, and H. Chen. An Admission Control Scheme for Predictable Server Response Time for Web Accesses. In Proc. Conf. On World Wide Web, New York USA: ACM Press, 2001. 545-554

[99] Z. Liu, M. S. Squillante, and J. L. Wolf. On Maximizing Service-Level Agreement Profits. In Proc. ACM Conf. On Electronic Commerce, Orlando, FL, 2001

[101] T. Madhyastha and K. P. Yang. Physica Modeling of Probe-based Storage. In Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies, Apr. 2001. 207–224

[102] M. Sivan-Zimet and T. M. Madhyastha. Workload Based Modeling of Probe-based Storage. In Proceedings of the 2002 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, June 2002. 256–257

[103] Ulrich Hahn, Werner Dilling, Dietmar Kallta. Improved Adaptive Replacement Algorithm for Disk Caches in HSM Systems. In Proceedings of IEEE Symposium on Mass Storage Systems, March, 1999. 128-140

[104] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Buffering. In Proc. ACM SIGMOD'93, Washington, D. C, May. 1993. 297 – 306

[105] Jung-Hoon Lee, She-Woong Jeong et al. An Intelligent Cache System with Hardware Prefetching for High Performance. IEEE Transactions on Computers, 2003, Vol.52(5): 607-616

[106] John S. Bucy. the DiskSim Simulation Environment Version three Reference Manual. School of Computer Science, Carnegie Mellon, 2003.

[107] Aameek Singh, Kaladhar Voruganti, Sandeep Gopisetty et al. A Hybrid Access Model for Storage Area Networks. In Proceedings of MSST 2005, Washington,USA: IEEE CS Press, 2005. 181-188

[108] L. Carley, J. Bain, G. Fedder, et al. Single-chip Computers with Microelectromechanical Systems-based Magnetic Memory. Journal of Applied Physics, May 2000, Vol. 87(9): 6680–6685

[109] J. W. Toigo. Avoiding a Data Crunch–A Decade away: Atomic Resolution Storage. Scientific American, May 2000, Vol. 282(5): 58–74

[110] P. Vettiger, M. Despont, U. Drechsler, et al. The “Millipede” — More than One Thousand Tips for Future AFM Data Storage. IBM Journal of Research and Development, 2000, Vol. 44(3): 323–340

[111] L. R. Carley, G. R. Ganger, and D. F. Nagle. MEMS-based Integrated-circuit Mass-storage Systems. Communications of the ACM, Nov. 2000, Vol. 43(11): 72–80

[112] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and et al. Operating System Management of MEMS-based Storage Devices. In Proceedings of the 4th Symposium on Operating Systems Design and Implementation, Oct.2000. 227–242

[113] S. W. Schlosser, J. L. Griffin, D. F. Nagle, et al. Designing Computer Systems with MEMS-based Storage. In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS), Cambridge, MA, Nov. 2000. 1–12

附录　攻读博士学位期间发表论文目录

[1] Yihui LUO, Changsheng XIE. A Security Scheme for United Storage Network.. Grid and Cooperative Computing —GCC2004 Workshops, 2004,10. LNCS Editorial.（第一署名为华中科技大学, SCIE 收录）

[2] Yihui Luo, Changsheng Xie, Xinwei Zheng, Chengfeng Zhang, Zhen Zhao. A Proxy Cache Scheme of Network Storage. Second International Conference on Embedded Software and Systems, 2005,12.IEEE Computer Society Press. （第一署名为华中科技大学, ISTP 收录）

[3] 罗益辉, 谢长生, 韩德志. IP-SAN 的安全研究与实现. 计算机研究与发展, 2004, Vol.41.(第一署名为华中科技大学)

[4] 罗益辉, 谢长生, 张成峰. 基于三方传送的 USN 的设计和实现. 计算机工程, 2006,Vol.32, No.3,（第一作者, 第一署名为华中科技大学, EI 收录）

[5] 谢长生, 罗益辉. IP-SAN 的研究与设计. 小型微型计算机系统. 2005, Vol.26, No.6.(第一署名为华中科技大学)

[6] 罗益辉, 谢长生. 统一存储网的研究和设计. 湖北大学学报（自源科学版）, 2004, Vol.26，No.4.(第二署名为华中科技大学)

[7] 罗益辉, 谢长生. 一种统一存储网的 Cache 方案. 湖北大学学报（自源科学版）, 2005, Vol.29, No.3.(第二署名为华中科技大学)

[8] 罗益辉, 谢长生, 张成峰. 存储系统的集中式 Cache 替换算法. 华中科技大学学报（自源科学版）. （第一署名为华中科技大学, 已录用）

[9] 张成峰, 谢长生, 罗益辉. 网络存储的统一和虚拟. 计算机科学. 2006, Vol.33, No.6. (第一署名为华中科技大学)

海量对象存储系统数据组织算法的研究

作者： 罗益辉
学位授予单位： 华中科技大学

相似文献(10条)

1. 期刊论文 孙丽丽.谈华芳 一种共享对象存储的并行文件系统设计 -计算机应用2005,25(z1)

高可用和动态可扩展特性是并行文件系统设计实现中的难点.利用对象存储在I/O性能、可用性和可扩展性方面的优势,提出一种共享对象存储的并行文件系统设计,在保证高性能I/O的基础上,侧重提高系统的可用性和动态可扩展性.

2. 学位论文 靳超 主动存储系统结构的研究 2003

随着计算机技术的发展以及用户对于存储需求的日益增长，主动存储系统成为热点研究。如何利用未来存储设备上的计算能力来支持高性能的计算和高效的存储访问成为问题的关键。本文首先提出了存储对象和应用对象相结合的设备访问接口结构，并提出了一种主动存储系统的可扩展模型，该模型有效地利用设备的物理特征来提高设备访问性能，在该模型的基础上提出了存储管理系统中优化服务质量的算法。在这些研究的基础上，设计并实现了一个基于主动存储设备的分布式存储系统，取得了较好的效果。

本文的贡献主要包括以下几个方面：(1)提出了一种主动存储体系结构的计算和存储访问模型，同时研究和分析了该模型相对于传统存储系统的性能优势，并探讨了适用于该计算模型的应用实例。实验结果表明基于该模型所完成的并行数据敏感性应用任务在性能方面得到了显著提高。

(2)提出了一种在主动存储设备上构建分布式存储系统的可扩展元数据访问机制。基于该机制构建的分布式存储系统能够保证系统的元数据访问具有良好的可扩展性能。实验数据表明，使用该机制的系统，随存储结点的增加吞吐率呈近似线性扩展。

(3)提出了一种利用主动存储设备上的计算能力通过理解设备的物理特征来提高I/O访问性能的机制。利用本方法可以针对不同访问模式的数据进行相应的优化处理。实验数据表明，对于同步的I/O写请求，本方法能够将性能提高5-8倍左右。

(4)提出了一种保证服务质量(QoS)的虚拟存储空间管理算法。该算法能够针对用户的不同级别和访问要求支持相应的服务质量。在充分利用整体资源的条件下，对于高优先权的用户分配较多的存储资源；同时尽可能地保证低优先权用户的权益。

(5)设计了一个主动存储设备(TOSD)。该设备采用了本文中的部分研究成果，支持高效的、事务的并发访问。并且基于该主动设备完成了一个分布式存储系统——TODS，该系统的特点是提供透明持久化的对象式访问接口，能够提供便捷的、高效的分布式数据存储功能。同时在该系统的基础上开发了一个可扩展的Email服务应用。

3. 期刊论文 聂刚.卿秀华.NIE Gang.QING Xiu-hua 基于对象存储的Lustre文件系统的研究 -信息技术2007,""(9)

随着高性能计算网络集群系统的高速发展,传统的网络存储架构—网络附加存储NAS和存储区域网SAN已越来越不能够满足系统对存储性能的要求.针对SAN和NAS的不足,新一代的网络存储技术—基于对象存储OBS成为了研究的热点.重点论述了基于对象存储的架构和特点,并针对基于对象存储的Luster文件系统进行了初步测试,通过和传统的NFS文件系统对比,分析了对象存储文件系统在可扩展性、性能、易用性和安全性等方面的优越性能.

4. 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。

5. 学位论文 鲁春怀 基于对象存储设备的文件系统及安全机制的研究 2006

基于对象存储（OBS）系统具有较好的安全性，能实现跨平台的数据共享，并具有高性能和可扩展性。基于对象存储设备（OSD）是OBS系统中智能化的网络存储节点，它能给用户提供一组基于对象的访问接口，并自主化地管理其内存储的对象。

基于对象的文件系统是OSD软件系统的核心，OSDFS是OSD中基于对象的文件系统的一种简单实现，它是在Linux的EXT2文件系统基础上实现的一种逻辑对象文件系统，通过将OSD中的对象映射为底层的EXT2文件来实现对象的存储管理。

为了优化OSD中对象的存储，提出了一种用于OSD中的智能化的基于对象的文件系统—SOBFS，它能根据对象的属性使得基于不同应用的对象采用不同的存储管理机制。SOBFS是一个文件系统容器，可以包含多种基于不同应用的文件系统，目前包含了两种文件系统：通用对象文件系统和媒体对象文件系统，分别用于存储普通文档/网页等小对象和大的媒体对象。通过与OSDFS进行理论上的比较，SOBFS显示出了更好的性能。

基于OSD中对象存储的安全性需求，提出了一种用于OSD的安全机制，它采用的基于证书的访问控制机制保证了用户对OSD中对象的合法访问以及客户和OSD之间所交换的命令和数据的完整性，而基于对象的文件系统的安全性又保证了OSD中所存放的数据的保密性，而且系统仍然具有足够的灵活性实现用户之间数据的共享。另外，实验结果表明，安全开销给系统造成的性能损失也是比较小的。

6. 会议论文 冯振乾.苏金树.张晓哲 对象存储研究 2007

信息资源爆炸式增长,给存储系统各方面带来严峻挑战。网络存储面临的剧增需求,如高性能、高可扩展性、易管理,数据共享和高安全性,促使人们思考通过融合NAs和SAN,用基于对象存储来取代传统基于块的存储方式,以寻求有效解决方案。本文首先总结目前NAS和SAN存在的主要问题,介绍了对象存储提供的解决方案;然后总结目前国内外在对象存储方面所做的研究和标准,化工作。介绍对象存储基本技术,最后对其前景进行展望。

7. 学位论文 孙丽丽 共享对象存储并行文件系统的元数据管理研究 2005

当前的高性能计算已经由传统的主机方式逐渐向机群方式演变。机群体系结构的采用一方面使得系统的计算能力大大加强，另一方面也对当前的存储系统提出了更高的要求：在保证数据共享和易管理性的前提下，要求存储系统在存储容量和I／O性能方面具有很好的可扩展性。传统的基于主机的存储架构已经远远不能满足这些要求，研究新的存储体系结构和相应的文件系统具有十分积极的意义。本文基于对象存储系统，提出一种新的共享对象存储设备的并行文件系统(命名为SOPFS)设计，其目标是为高性能计算机群提供高性能、可扩展、高可用的机群存储系统。在文中给出了SOPFS的总体描述，内容包括分布元数据管理、并行数据访问等关键技术；结合SOPFS中动态散列分区的元数据组织方法，设计实现了元数据的访问管理；针对高性能并行计算中经常出现的对同一文件／目录的高并发访问情形，提出了一种动态的元数据复制策略，通过多个存放元数据副本的MDS同时响应对同一文件元数据的并发访问请求，提高了高并发访问情形下的元数据访问效率；利用SOPFS的结构优势，即文件系统的数据通路与控制通路分离，提出了懒惰的元数据更新策略，使得文件元数据的更新独立于文件数据读写过程，进一步保证了高的I／O吞吐率；在SOPFS中设计实现了元数据的事务日志机制，以提高系统的可用性和提供系统的快速失败恢复能力。

8. 期刊论文 赵水清.冯丹.ZHAO Shui-Qing.FENG Dan 基于对象存储设备上的服务质量研究 -计算机科学2006,33(9)

基于对象存储(Object-based Storage,OBS)作为下一代互联网存储协议标准逐渐被人们所接受,基于对象存储设备(Object-based Storage Device,OSD)所具有的可扩展性和智能性能很好地支持应用程序的服务质量(Quality of Service,QoS)需求.本文分析了基于对象存储系统的QoS需求,讨论了如何把应用客户的qoS需求转化为QoS属性,扩展了OSD SCSI协议集标准,以支持QoS,接着采取量化分析方法对QoS进行了分析,详细分析应用客户和OSD之间QoS信息交互的工作过程,最后,紧密结合OSD的特性,给出OSD上的QoS三层模型Q-Model和一优化算法BRP.

9. 学位论文 吕松 对象存储结点的设计与实现 2006

随着知识经济的推进和信息时代的日益临近，同时在网络技术革新的推动下，存储行业既迎来大好的市场前景又面临巨大技术挑战。数据量的指数级增长和基于高速网络的数据应用要求的进一步提高，对数据存储技术提出了革新的要求，不仅针对网络存储的体系结构，而且对存储设备的接口也提出了新的要求。

基于对象存储提出了对象这样新的数据存储单位，增加了数据的自我管理能力，同时方便了数据的共享和访问，将对象存储系统中各个不同功能部件的职能进行了重新的分工，将对象数据的物理存储管理任务移交给存储设备，提高了存储结点的智能性。用户和存储设备结点之间直接进行数据传输，从而提高了数据访问速度以及系统的可扩展性。

基于对象的存储设备结点利用通用处理器的计算能力，实现专用OSD Controller（Object-Based Storage Device Controller）的处理功能。对象处理模块通过可装载方式插入到Linux操作系统内核，在内核态下实现，减少I/O过程中内核切换的开销。基于对象存储结点通过Iscsi通道和MDS、Client进行数据交互，顺应了现在网络存储向低成本IP存储发展的趋势。对象存储结点实现了最新T10标准中的常用OSD命令集，并对该标准进行相应的扩展。对象存储结点在Ext2文件系统基础上构建了对象存储管理，实现了对象存储接口的物理基础和外部访问接口。测试结果表明，对象接口很好地提高了设备的并行性和传输效率，同时单台对象存储结点虽然增加了元数据管理负担，但是性能仍然和文件接口的FTP相当。

10. 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。

本文链接：http://d.g.wanfangdata.com.cn/Thesis_D048303.aspx
授权使用：中科院计算所(zkyjsc)，授权号：c4092ea1-b9c2-447d-a57c-9e4001289f6c
下载时间：2010年12月2日