

兰州大学

硕士学位论文

对象文件系统中元数据管理技术研究

姓名：姜成龙

申请学位级别：硕士

专业：计算数学

指导教师：徐德启

20051101

摘 要

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

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

1. 分析研究基于对象存储系统的体系结构,设计并实现了一个小型的对象存储文件系统原型 OCFS。

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

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

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

关键词: 基于对象存储, 元数据管理, 负载均衡, 可用性

ABSTRACT

With the progress of information technology and the large scale application of Internet, the amount of data have been substantial increase, it bring a large chance to the development of network storage contemporary. It's a significant task of how to build up a storage system which has high performance, good expansibility and good usability and is easy to manage and safe.

Object-based Storage Technology is a new technology in storage field and gives new storage architecture. Data object is the kernel of this architecture, encapsulating metadata and data which are managed by respective system. The large scale distributed file system established in Object-based Storage Architecture can provide good scalability, high performance and strong data process capability in parallel. Managing metadata by cluster is one trend of metadata management. Our project focuses on metadata management of object-based storage architecture. the following contents are studied in this thesis:

1. On the basic of research to Object-based architecture, design and realize a prototype system to manages metadata.

2. Adopt a novel architecture OCFS II to realize MMC (Metadata Manage Cluster) which manages metadata in hiberarchy. In MMC two clusters of DPISs and MDSs manage directory metadata and file metadata respectively. And there is a research to the key techniques of it.

3. On the basic of research to load balancing, we design and realize the scheme of load-balancing which combines the static load distributing and dynamic redistributing. Metadata load is diffluent or with distributing metadata by static metadata partition algorithm. Servers feedback the load information dynamically and the load that is not balancing is redistributed. MMC is load balancing with the method and even as access "hotspot" data.

4. Design and realize the scheme of the usability of MMC. DPISs cluster adopts sharing and tolerating array of disks and the mechanism of detecting failed node to resolve fault of the sharing data and nodes. MDSs cluster can take over the task of metadata service and rebuild backup data by using backup servers and spare server. MMC can tolerate failure and recovery from single failure and multi-failure in special conditions. So the system has high reliability and good usability.

Keyword: Object-based Storage, Meta-data Manage, Load Balancing, Usability

原创性声明

本人郑重声明：本人所呈交的学位论文，是在导师的指导下独立进行研究所取得的成果。学位论文中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。

本声明的法律责任由本人承担。

论文作者签名： 姜成龙 日 期： 2005.11

关于学位论文使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用学位论文的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州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，可以采用任何复制手段保存和汇编本学位论文。本人离校后发表、使用学位论文或与该论文直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

保密论文在解密后应遵守此规定。

论文作者签名： 姜成松 导师签名： _____ 日 期： _____

第一章 绪论

随着信息社会的发展,越来越多的信息被数据化,尤其是伴随着 Internet 的发展,由此产生的各种数据呈几何级数爆炸式增长,促使数据存储容量以每年 3 到 5 倍的速度急剧增加。存储技术产品受到业界及专家们的高度重视。对大多数依靠计算机运行关键商务程序的现代企业来说,信息已经成为一种超越其竞争对手的战略资产,数据的有效使用和管理越来越成为企业 IT 部门的首要任务。企业在存储设备和存储管理软件上的开销占据了整个网络成本越来越大的比重,并将逐步超过 50%。大量数据密集型的应用,如数字图书馆、数据仓库、数据挖掘、气象数据处理、医药视频图像数据处理、生命科学研究、多媒体点播、在线数据处理等,对存储系统的性能(I/O 带宽、吞吐率、响应时间等)提出更高的要求,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存储领域所面临的一个重要课题^[3,4]。

§ 1.1 课题的研究背景

随着高性能计算由传统的主机方式向网络化集群演变,传统的基于主机的存储架构已逐渐向网络化存储发展,计算和存储分离的趋势越来越明显。针对网络化存储中 SAN 和 NAS 的不足,国际上已开展针对 Linux 集群的新型文件系统——对象存储文件系统的研究,基于对象的存储系统已经成为存储领域研究的一个新热点。

1.1.1 传统的存储技术

传统的存储技术包括基于磁盘或 RAID (Redundant Array of Inexpensive Disks, 磁盘阵列) 系统等存储设备直接与主机相连的存储技术。主机与存储系统通过 SCSI (Small Computer Systems Interface, 小型计算机系统接口) 总线连接起来,同时又通过网络接口连接到网络中,这种结构称为 DAS (Direct attached storage)。这种结构中,数据的流动路径为:当写入数据时,数据从网络经过网络接口通过 DMA (Direct Memory Access, 直接存储访问) 操作到达主机内存,然后从内存又通过 DMA 操作送到存储系统;当读取数据时,则是从存储系统到主机内存,再从内存通过网络接口送到网络上^[30,31]。

这种结构具有两个缺点:

①可扩展性差。可扩展性是指当系统进行扩展时,系统的主要性能指标是否受限。DAS 结构在当客户机数和服务器的磁盘数同时增加时,由于主机的内存限制,整个系统的吞吐量不能同步增加。

②系统的持续带宽较低,不能满足高持续带宽传输的应用要求,如多媒体数据传输等应用。

主要有三个原因限制了用户所得到的带宽和系统可扩展性。第一个原因是主机的带宽限制。由于主机是用户与存储子系统之间数据交换的必经路径,因此主机的带宽是整个系统带宽的一个重要因素,虽然计算机技术的发展使主机总线带宽大为增加,但是仍不能满足多媒体等数据服务对带宽的要求,主机带宽成为存储系统与用户之间的瓶颈。主机内存的两次 DMA 操作进一步降低了持续带宽。第二个原因是主机的内存容量限制。由于主机的内存容量有限,当有连续的大量数据访问请求时,主机的内存容量将很快达到饱和,而不能处理剩下的数据传输请求。第三个原因,文件管理系统的开销也会增加数据访问时间。以上三个方面的原因都与主机有关。由于第三方与存储子系统之间的联系是间接的,要经过主机这个桥梁,主机成为系统的瓶颈,从而导致了上述的性能缺陷。

1.1.2 网络存储技术

随着网络技术的发展,网络存储技术逐渐在存储领域占据了主流,成为继个人计算机(PC)和互联网络(Internet)后第三次以数据存储为中心的 IT 浪潮的引导者。网络存储建立在客户/服务器计算的基础上,它将管理存储和文件系统的负担分摊在计算机系统和存储设备之间,计算机负责数据的处理,而存储设备或子系统负责数据的存储。网络存储将网络技术与新兴的存储领域结合起来,充分利用网络技术的特点,彻底解决传统存储方案的弱势,其主要特征体现在超大存储容量、高速数据传输以及高的系统可用性。网络存储非常适合数据量增长迅速、拥有潜在升级数据的企业网络数据,可以实现不同数据的远距离、安全的集中管理,实现网络上的数据集中访问,实现不同主机类型的数据访问和保护等等企业面临的复杂存储问题。

目前网络存储主流技术主要有两种,一个是 NAS,一个是 SAN,两种技术适用于不同的应用领域,但目前也呈现出融合的趋势。

(1) 网络附接存储

网络附接存储(NAS, Network Attached Storage)是直接连接到网络(如以太网)的一种存储器,用类似 NFS(网络文件系统)或者 CIFS(公用 Internet 文件系统)等标准化的协议提供文件级的数据访问。典型的 NAS 提供带有预先配置好磁盘容量、集成的系统和磁盘管理软件,构成一个完备的存储解决方案。

这种方案中存储设备在功能上完全独立于网络上的主服务器,建立了存储子系统到客户机的直接连接,减少了数据传输中主机的干预,能够实现高持续带宽和好的可扩展性。专用的硬件和软件构造的专用服务器,与其他资源独立,不会占用网络主服务器的系统资源,不需要在服务器上安装任何软件,不用关闭网络上的主服务器,就可以为网络增加存储设备。由于不再是为文件服务器增加磁盘,服务器则从原先的 I/O 负载中解脱出来,摆脱瓶颈的尴尬。另外, NAS 具有较好的协议独立性,支持 Unix、NetWare、Windows95 NT、OS/2 或 Intranet Web 的数据访问。客户端不需要任何专用的软件,安装简易,甚至可以充当其他机器的网络驱动器,可以方便地利用现有的管理工具进行管

理（如支持 Java 的浏览器，Telnet 等），可以仿真 Netware 和 NFS 服务器，在 NetWare 环境中，其以一个 NetWare 服务器的身份出现，充当一个 NOVELL 卷，客户机可以把它映射为网络驱动器。NAS 为那些访问和共享大量文件系统数据的企业提供了一个高效、性能价格比优异的解决方案。NAS 适用于那些需要通过网络将文件数据传送到多台客户机上的用户。NAS 支持若干客户端之间的文件共享，所以用户可以使用 NAS 作为日常办公中需要经常交换的文件的存储子系统，比如存储网页等。

与传统的通用服务器不同，存储专用服务器能在不增加复杂度、管理开销、降低可靠性的基础上，使网络的存储容量增加，具有较好的可扩展性。因为不需要在通用服务器上添加更多的硬件，服务器的可靠性大大提高。由于专门为文件服务进行了优化，I/O 性能大大提高，能够充分利用可得到的 10MB~100MB 网络带宽，有较大的数据吞吐量。

网络附接存储具有以下特性：

①可扩展性。由于可以提供存储设备与客户机之间的直接数据传送，而不需服务器进行存储转发，所以当客户机增多时，网络主服务器负载不会按比例增长。另外随着网络存储规模的增长，系统管理开销和硬件开销也不会按比例增长，而是始终限制在一个可接受的范围内，从而提高了系统的可扩展性。

②可访问性。网络附接存储能很好支持使用不同操作系统（Unix、NetWare、Windows95、NT、OS/2 或 Intranet Web）的用户对网络存储设备的数据访问，具有较好的协议独立性。

③高性能。由于数据可以在客户机与服务器之间直接传送，而不需要服务器进行存储转发，消除了潜在的 I/O 瓶颈。另外，其软硬件体系结构也为存储设备的单一任务——数据服务进行了相应的优化，充分利用诸如内存、CPU、总线周期等硬件资源，从而大大提高了存储设备的性能。

④易管理。安装简单，便于管理，具有较好的操作优势。例如，允许客户机利用支持 Java 的浏览器通过网络对网络存储设备进行基于 Web 的管理。

⑤廉价。网络附接存储减少了管理开销，消除了不必要的停工时间，免去了购置昂贵的多功能服务器的费用，从整体上降低了系统成本。

网络附接存储存在以下局限性：

①受限的数据库支持。到目前为止，大多数 NAS 产品都是文件服务器产品，采用的是文件协议，而不是块协议或数据库访问协议。这对于文件服务是有利的，但不适合数据库处理或其他不使用文件协议的应用。而且很多数据库产品，如 ORACLE 并不支持将数据保存在 NAS 设备上。

②产品缺乏灵活性。NAS 产品是专用的，不能通过通用的、流行的软硬件来升级或改善产品的性能。NAS 产品一旦设计好之后，就很难改变。作为个体，NAS 产品是缺乏灵活性的。

③备份与恢复问题。NAS 装置在网络备份和恢复方面存在如下的两个问题：

·NAS 装置保存有大量的数据，进行数据备份与恢复需要耗用大量时间，会过分占

用网络资源。

NAS 采用的是专用操作系统，很难与现有的备份工具集成。需要采用另外的备份和恢复方案。

(2) 存储区域网络

存储区域网络 (SAN, Storage Area Network)^[5] 是通过光纤通道协议 (FCP, Fibre Channel Protocol) 使用 SCSI 命令向磁盘存取数据的存储专用网络。SAN 以数据存储为中心，采用可伸缩的网络拓扑结构，通过具有高传输速率的光纤通道的直接连接方式，提供 SAN 内部任意节点之间的多路可选择的数据交换，并且将数据存储管理集中在相对独立的存储区域网内。在多种光通道传输协议逐渐走向标准化并且跨平台群集文件系统投入使用后，SAN 最终将实现在多种操作系统下，最大限度的数据共享和数据优化管理，以及系统的无缝扩充。

SAN 推出了新的概念：共享存储设备。共享存储设备是指对于连接到 SAN 上的任何计算机而言，连接到 SAN 上的原始存储设备（比如磁盘、光盘或磁带驱动器）就如同本地附接的设备一样。SAN 中光纤通道协议的应用最为广泛，但 SAN 并非局限于一种网络技术。SAN 的目标是通过将存储系统与计算系统相隔离，使得计算系统可以更方便地共享存储设备，从而提高存储系统的可用性和性能。常用的 SAN 接口方式除了光纤通道之外，还包括 ESCON、SCSI、SSA、HIPPI (High-Performance Parallel Interface) 和以太网。SAN 网络可以包括 Extender、Multiplexor、Hub Router、Gateway 和 Switch 等各种网络设备。具体到光纤通道，它提供三种网络拓扑结构：点到点连接、环连接和交换连接。在 SAN 的环境中，点到点的连接方式并不合适。在采用环连接的形式时，光纤通道的一个环路可以最多连接 127 个设备。光纤通道交换机 (Fibre Channel Switch, FCS) 则提供了各个端口之间的全交换功能，确保更高的网络带宽。

SAN 将高速的数据传输方案同本地 SCSI 协议有机地结合在一起，消除服务器和磁盘之间的专用连接，扩展以数据为中心的服务领域。SAN 能够解决与网络存储备份有关的问题，它可以提供 100MB/s 的高性能数据管道和共享的集中管理的存储设备，极大地提高了数据备份和恢复操作的可靠性和可扩展性。SAN 进行块数据传输、极易扩展，且管理设备有效，用户可以使用 SAN 作为关键应用的存储子系统，比如数据库、关键数据备份等，进行数据的集中存取与管理。SAN 推动了新的存储模式的发展，如服务器集群和存储集群等。与传统的直接附接存储和并行 SCSI 存储相比，SAN 具有高带宽、大容量，可扩展性好的优点。目前基于 Fibre Channel 的 SAN 应用方案最多，成熟的产品也最多。除此之外，基于 iSCSI 协议和 InfiniBand 的 SAN 也开始在存储市场占有一席之地。

SAN 系统的优点是在很大程度上解决了集中存储、存储管理和存储空间共享的问题，特别是在数据的可用性、系统容量和系统性能的动态可扩展性方面明显优于 NAS 系统。

SAN 系统的缺点在于系统使用复杂,数据共享的颗粒度过大,难于直接支持文件级的数据共享。SAN 目前还是以光纤通道技术为主,代价比较昂贵。

(3) 几种存储技术的比较

传统的 DAS 最适宜用于单独的服务器和要求低初始成本的场合,但它是一种存储资源不可共享的解决方案,管理性和灵活性差。NAS 的优势是易于管理和文件共享,提供各种应用领域的文件共享和文件服务功能,包括内容传送和分发、统一的存储管理、科学计算、Web 服务等,允许在不使服务器停机的前提下进行扩展。由于它使用的互连网络是低成本的以太网,因此它的购买成本和运行成本都要比 SAN 低,十分经济。SAN 的优势是高性能(高存储吞吐量)、可扩展性好(通过单一控制点管理多个磁带和磁盘设备)和高可靠性(专用备份工具可以减少对服务器和 LAN 的使用)。三种技术的详细比较见表 1-1。

表 1.1 DAS、NAS 和 SAN 三种存储技术的比较

	DAS	NAS	SAN
安装	简单	即插即用	复杂
管理	不易	容易	集中化的存储管理
兼容性	较好	可跨平台使用, 好	没有统一标准, 差
可扩展性	差	较好	无限扩展能力, 好
维护成本	高	低	较高
容错性	差	中等	好
连接错误	中等	低	高
通用性	中等	好	差
价格	低	较高	高

1.1.3 基于对象的存储技术

新的存储需求将会推动存储技术的发展。高性能计算(HPC, high-performance computing)推动新的计算结构的出现,这促进了存储体系结构的发展,逐渐形成了基于对象存储的体系结构^[1,2,7]。自美国航天署(NASA)Goddard 航天中心的科学家们开始实施 Beowulf 项目后,集群系统以其性价比优势成为高性能计算体系结构的一种趋势,随之对存储系统提出了更高的要求^[6]。

集群是一种并行和分布处理的系统,它由一组互相连接的多个独立计算机集合而成,并作为一个整体的集成计算资源工作。这些计算机可以是单机或多处理器系统(PC、工作站或 SMP),每个节点都有自己的存储器、I/O 设备和操作系统,可以是同构的或者是异构的。集群对用户和应用来说是一个单一的系统,它可以提供低价高效的高性能环境和快速可靠的服务。集群系统不但能够充分利用现有的计算资源,而且能够通过较低的软、硬件代价实现较高性能的计算机系统。随着微处理器技术和高性能网络技术的飞速发展,集群计算逐渐成为一种有成本效益的并行/分布式计算资源,其具有可伸缩、

SAN 系统的缺点在于系统使用复杂, 数据共享的颗粒度过大, 难于直接支持文件级的数据共享。SAN 目前还是以光纤通道技术为主, 代价比较昂贵。

(3) 几种存储技术的比较

传统的 DAS 最适宜用于单独的服务器和要求低初始成本的场合, 但它是一种存储资源不可共享的解决方案, 管理性和灵活性差。NAS 的优势是易于管理和文件共享, 提供各种应用领域的文件共享和文件服务功能, 包括内容传送和分发、统一的存储管理、科学计算、Web 服务等, 允许在不使服务器停机的前提下进行扩展。由于它使用的互连网络是低成本的以太网, 因此它的购买成本和运行成本都要比 SAN 低, 十分经济。SAN 的优势是高性能(高存储吞吐量)、可扩展性好(通过单一控制点管理多个磁带和磁盘设备)和高可靠性(专用备份工具可以减少对服务器和 LAN 的使用)。三种技术的详细比较见表 1-1。

表 1.1 DAS、NAS 和 SAN 三种存储技术的比较

	DAS	NAS	SAN
安装	简单	即插即用	复杂
管理	不易	容易	集中化的存储管理
兼容性	较好	可跨平台使用, 好	没有统一标准, 差
可扩展性	差	较好	无限扩展能力, 好
维护成本	高	低	较高
容错性	差	中等	好
连接错误	中等	低	高
通用性	中等	好	差
价格	低	较高	高

1.1.3 基于对象的存储技术

新的存储需求将会推动存储技术的发展。高性能计算(HPC, high-performance computing)推动新的计算结构的出现, 这促进了存储体系结构的发展, 逐渐形成了基于对象存储的体系结构^[1,2,7]。自美国航天署(NASA)Goddard 航天中心的科学家们开始实施 Beowulf 项目后, 集群系统以其性价比优势成为高性能计算体系结构的一种趋势, 随之对存储系统提出了更高的要求^[6]。

集群是一种并行和分布处理的系统, 它由一组互相连接的多个独立计算机集合而成, 并作为一个整体的集成计算资源工作。这些计算机可以是单机或多处理器系统(PC、工作站或 SMP), 每个节点都有自己的存储器、I/O 设备和操作系统, 可以是同构的或者是异构的。集群对用户和应用来说是一个单一的系统, 它可以提供低价高效的高性能环境和快速可靠的服务。集群系统不但能够充分利用现有的计算资源, 而且能够通过较低的软、硬件代价实现较高性能的计算机系统。随着微处理器技术和高性能网络技术的飞速发展, 集群计算逐渐成为一种有成本效益的并行/分布式计算资源, 其具有可伸缩、飞速发展, 集群计算逐渐成为一种有成本效益的并行/分布式计算资源, 其具有可伸缩、

高可用、高性能、易管理和高性价比的优势,集群系统在大规模计算机的应用中成为一种发展趋势。

适合集群使用的存储系统需要解决两个重要的问题:数据共享以及高性能的 I/O 速率、数据吞吐率。首先,必须能提供对数据的共享访问,保证共享数据对计算集群上的所有进程都可用。这样可以简化应用系统的开发和维护,使应用程序容易进行读写,存储系统更容易平衡计算请求。确保集群系统得到高效使用的必备条件是,它可以对 TB (1TB=1000GB, 1GB=1000MB) 量级的共享数据进行快速访问。没有这一点,集群系统的性能将会大幅降低。随着集群系统的规模越来越大、节点越来越多,为实现各个节点对共享数据的高效访问,对存储系统的要求也越来越高,传统的、基于网络的存储系统已经不能提供满足这种共享访问所必需的性能。其次,存储系统必须提供高性能的 I/O 速率和数据吞吐率。高性能计算的问题规模越来越大,参与的节点数越来越多,要同时满足几十个,几百个甚至有时几千个服务器总的访问请求,对存储系统的 I/O 速率和数据吞吐率提出了越来越高的要求。基于对象的存储技术正作为构建大规模存储系统的基础而悄然兴起,成为存储领域研究的一个新热点。它利用现有的处理技术、网络技术和存储组件,可以通过一种简单便利的方式来获得前所未有的可扩展性和高吞吐量。

为便于数据的管理,除数据本身外还有关于数据的数据,称之为元数据。传统的存储系统中,元数据和数据本身由同一个系统管理,保存在同一台存储设备上,并且为提高访问效率,元数据与其描述的数据在物理上尽可能靠近。元数据在系统整个的存储容量中占很小的比例,研究表明,在随机的文件访问工作负载中,甚至在小文件的访问中,元数据的访问负载仅占 10%,而其余 90%都是在对文件数据的访问^[7];然而文件系统访问中有 50%—80%的只访问元数据^[10],元数据的访问极其频繁,将元数据和文件数据绑定在一起访问,会极大地影响文件访问的并行性。

基于对象的存储系统是存储领域研究的一个新热点。这种体系结构的核心是对象,对象是容纳了应用数据和一个可扩展的存储属性的基本容器。传统的文件被分解为一系列存储对象,并分发到一个或多个“智能磁盘”上,这种磁盘被称为对象存储设备(OSD, Object-based Storage Device)。每一个对象存储设备具备本地处理功能、用于数据和属性缓存的本地内存和本地的网络连接。对象存储设备构成了分布式存储结构的核心,它将许多传统的存储分配行为从文件系统层转移,从而解决了当前存储系统的一个瓶颈问题。

对象属性包括安全信息和使用状况统计信息,这些信息被用于基于安全认证的访问、服务质量控制,以及实现对象存储设备间负载均衡所需的数据动态分配。对象存储技术采用了和集群计算系统类似的可扩展结构,当存储容量增加时,它提供的均衡模型能够保证网络带宽和处理能力也同步增长,从而确保系统的可扩展性。

存储网络工业协会(SNIA)和 T10(InterNational Committee on Information Technology Standards)标准技术委员会中的联合技术小组正在制定一个关于 OSD 的标准。标准包括了一个针对 iSCSI 协议的命令集,它在原有的 SCSI 命令集中增添了对象

扩展功能。同时,对象规范和命令集的制定促使了一种新的智能存储设备的出现,这种智能存储设备可以集成到基于 IP 的、高性能、大规模并行存储环境中去。目前多个业内领先的存储设备公司都参与了这项工作,其中包括 EMC、惠普、IBM、Intel、希捷及 Veritas 软件公司等。

对象存储结构提供了新一代网络存储系统的基础。在新兴的应用中,它和一种可扩展的、为应用程序提供文件系统接口的元数据管理层结合在一起。这一层负责管理诸如目录隶属关系和文件所有权这样的信息,它还负责将跨多个对象存储设备的存储对象(每个存储对象是文件的一部分)联接成一个文件,以确保数据的可靠和可用。集群节点向元数据管理层提出数据请求,例如打开或关闭文件,通过认证后,接受它能够访问对象存储设备所必需的信息,此后集群节点可以直接对文件进行读写操作,而和元数据管理层无关。

对象存储结构作为可扩展集群文件系统的一部分被实现后,就能够为数以百计的客户端提供高容量的总带宽。简而言之,对象存储技术可以为高性能 Linux 集群系统提供高性价比的共享存储。对象存储结构在解决存储系统可扩展性、I/O 性能、可用性等方面有结构的优势,适合高性能计算(HPC, high-performance computing)对存储的需求。

§ 1.2 课题相关领域的研究现状

观察目前存储技术的发展,NAS 产品正走向成熟,SAN 目前还是以光纤通道技术为主。作为发展中的技术,目前网络存储主流技术存在着局限性,由于 SAN 通常是基于光纤的解决方案,需要专用的交换机和管理软件,成本较高;相对于 SAN,NAS 的可扩展性较小,但它可以适合中小级别的存储需求。

NAS 建立在现有 LAN 和文件系统协议基础之上,特别适合文件级的数据处理;SAN 则是一个独立的数据存储网,对于大容量块级数据传输具有明显的优势。NAS 和 SAN 应用互补。

尽管 NAS,SAN 之间存在着区别,但是 NAS,SAN 作为两种互为补充的存储技术正在逐渐走向融合。SAN 和 NAS 可以联合使用,通过 NAS 网关(NAS Gateway)可以组成 SAN 和 NAS 的混合存储网络,使用 SAN 作为 NAS 的后端存储可以利用 SAN 的优点为 NAS 设备提供高性能、大容量的存储,而主机则可以利用 NAS 设备的其他优点,提供良好的可扩展性和数据共享,支持多台对等客户机之间的文件共享。SAN 和 NAS 的混合使用中,SAN 提供高性能的访问速度,NAS 提供由文件处理带来的协作性,可以最大限度地利用网络化存储。

网络存储处在不断的变化和发展之中,各种新的结构和标准正在不断的制定中。例如,基于 IP 的 iSCSI 协议,实现了 SCSI 和 TCP/IP 协议的连接,使 SAN 更廉价;新的互连技术 InfiniBand 不仅可用于服务器内部的互连、服务器之间的互连、集群系统的互连,还可用于存储系统的互连,组建基于 InfiniBand 的 SAN;新的 I/O 结构如 Infini

Band、3GIO、Hyper Transport 和 Rapid IO 技术的出现,对基于传统网络连接的存储系统会产生重大的影响等这些为人们解决数据存储问题提供了新方法,相关的标准也正由 IETF, SNIA, ANSI 等组织制定中。

针对 Linux 集群对存储系统高性能和数据共享的需求,国外已开始研究全新的存储架构和新型文件系统,希望能有效结合 SAN 和 NAS 系统的优点,支持直接访问磁盘以提高性能,通过共享的文件和元数据以简化管理,目前对象存储文件系统已成为 Linux 集群系统高性能文件系统的研究热点,如 Cluster File Systems 公司的 Lustre、Panasas 公司的 ActiveScale 文件系统等。Lustre 文件系统采用基于对象存储技术,它来源于卡耐基梅隆大学的 Coda 项目研究工作,2003 年 12 月发布了 Lustre 1.0 版,预计在 2005 年将发布 2.0 版。Lustre 在美国能源部(U. S. Department of Energy: DOE)、Lawrence Livermore 国家实验室, Los Alamos 国家实验室, Sandia 国家实验室, Pacific Northwest 国家实验室的高性能计算系统中已得到了初步的应用,IBM 正在研制的 Blue Gene 系统也将采用 Lustre 文件系统实现其高性能存储。ActiveScale 文件系统技术来源于卡耐基梅隆大学的 Dr. Garth Gibson,最早是由 DARPA 支持的 NASD (Network Attached Secure Disks) 项目,目前已是业界比较有影响力的对象存储文件系统,荣获了 ComputerWorld 2004 年创新技术奖。

§ 1.3 主要的研究成果

一些领域中已经运用了对象存储的思想,但由于刚起步不久,各种研究尚待深入。特别是对于对象存储系统中的核心部分的元数据管理子系统,目前还是使用单元数据服务器管理元数据,在大型应用中将成为性能和可扩展性的瓶颈,为适应大型信息存储的应用,必须实现以集群来管理元数据。本文在实现基于对象存储思想的对象存储文件系统原型 OCFS 的基础上,主要研究元数据服务的集群化管理,包括元数据管理集群的体系结构,相应的数据分割算法,以及容错方法等,提出一种新型的使用集群管理元数据的对象存储文件系统 OCFS II,并实现了元数据管理集群的负载均衡和容错算法。主要的创新工作和贡献如下:

①实现了基于对象存储思想的对象存储文件系统原型 OCFS (Object-based Cluster File System)。OCFS 是一个小型的文件系统原型,它以对象方式管理文件,将文件分为文件元数据和文件实际数据,采用元数据服务器 (MDS, Meta-data Server) 管理文件元数据,采用对象存储目标 (OST, Object Storage Target) 存储文件实际数据,通过 CFS (Client File System) 提供与符合 POSIX 语义的标准文件系统接口。OCFS 中,元数据服务器以 SQL 数据库管理元数据,使用文件系统目录模拟对象存储目标的智能存储设备。OCFS 作为内核模块被安装,用户能够通过 VFS 与 OCFS 进行命令和数据的交互,提供 UNIX 文件共享语义,满足多个客户共同使用文件系统。

②提出了一种新型元数据管理集群的模型。本文在实现的对象存储文件系统原型

OCFS 基础上, 重点研究对象存储文件系统中元数据的管理, 对对象存储文件系统原型进行改进, 提出一种运用集群管理元数据的对象存储文件系统 OCFS II。OCFS II 通过元数据管理集群对元数据实现层次管理, 元数据管理集群包括 DPIS 和 MDS 两个集群, 分别管理目录路径元数据和文件自身元数据。该元数据管理集群既能够保证快速定位元数据, 又能够在元数据属性修改和系统扩展时保证数据迁移量小, 实现元数据管理集群的高性能和高扩展性。

③针对 OCFS II 元数据管理集群的特点, 设计和实现了静态分配和动态反馈重分配相结合的负载均衡方案。集群使用对象存储文件系统时, 文件系统的 CFS 通常与用户节点结合在一起, 由对象存储文件系统的各个 CFS 根据文件的目录路径特征按照静态 hash 算法分发元数据请求负载, 实现静态的负载均衡; DPIS 集群中的各服务器能自主地收集系统负载信息, 并动态反馈信息以实现负载重分配, 保证 DPIS 集群的负载均衡。根据文件 ID (fid) 将文件自身元数据分布存储于 MDS 集群的各节点中, DPIS 访问文件自身元数据时, 随文件自身元数据的分布存储实现负载分流, 保证 MDS 集群的负载均衡。

④研究了 OCFS II 元数据管理集群中可用性问题。DPIS 集群采用共享存储的体系结构, 其中每个节点完成相同的元数据服务功能, 采用具有容错机制的共享存储硬件实现 DPIS 集群的数据容错, 保证整个 DPIS 集群元数据服务的可用性; 出现节点故障时, 通过重新选择服务器节点实现 DPIS 集群的多节点失效容错。MDS 集群中, 采用冗余的备份服务器和空闲服务器保证服务器节点故障时的元数据服务工作的接替和数据备份的重建, 在非同伴伙伴节点的多个服务器出现故障的情况下保证 MDS 集群的可用性。

§ 1.4 论文组织结构

第一章 绪论。介绍了课题的研究背景, 各种存储技术, 包括传统存储技术中使用 SCSI 连接的 DAS、网络存储技术中的 SAN 和 NAS 以及对象存储技术的发展概况; 相关领域的研究现状; 对所做的主要工作作了简要介绍。

第二章 对象存储文件系统原型的设计。研究了对象存储系统的体系结构, 在基于对象存储思想的基础上, 提出并建立了小型的对象存储文件系统原型 OCFS, 并对各子系统的软件模块作了描述。

第三章 改进的文件存储系统 OCFS II 的设计。论述了元数据管理的目标, 元数据后端存储的管理, 以及元数据管理集群中涉及到的集群系统体系结构以及数据的分割方法等。在理论研究的基础上, 改进原型文件系统 OCFS 为 OCFS II, 采用目录路径索引服务器 DPIS 集群和元数据服务器 MDS 集群对元数据进行分层次管理, 保证了元数据服务的高性能和可扩展性。

第四章 OCFS II 元数据管理集群负载均衡的实现方案。主要研究集群系统中负载均衡实现的相关知识, 对负载均衡的分类、实现层次和方法作了阐述, 并着重论述静态负

载均衡中的负载分配调度策略和动态负载均衡中的策略和算法实现的相关内容。针对 DPIS 集群和 MDS 集群的不同结构,提出了元数据管理集群中负载均衡实现的相应方案和算法。

第五章 OCFS II 元数据管理集群可用性方案研究。研究了常用的提高系统可用性的方法,并针对 OCFS II 元数据管理集群的特点,设计实现元数据管理集群的可用性保障方案,并对其有效性进行了分析。

第六章 总结与展望。总结了所做的主要工作,并对今后的工作提出了展望。

第二章 对象存储文件系统原型 OCFS 的设计

§ 2.1 对象存储系统体系结构

对象存储系统的基础是数据对象，数据对象封装了用户数据和这些数据的属性，数据对象的使用允许对象存储系统基于每个文件（在文件的粒度上）决定数据的分布和服务的质量，提高了灵活性和可管理性。对象存储系统中存储、拣取和解释这些数据对象的设备是对象存储设备（OSD, Objected-based Storage Device），它具有独特的设计，与一些标准的存储设备有实质性的区别，比如 Fibre Channel（FC）或者 IDE，这些标准的存储设备有传统的基于块的接口。对象存储设备把低级存储函数移植到存储设备中，并通过标准的对象接口访问设备，可以进行存储层的智能空间管理，允许对象存储设备对存储介质上数据的空间分配作出决定；可以智能化地对数据进行预取和缓存。

对象存储体系结构^[1,7]，组合了当今流行的两大关键存储系统，消除了它们不适合 linux 集群配置的缺点。首先，对象存储体系结构提供允许节点直接访问存储设备和数据高性能并行访问。其次，分布系统元数据允许共享文件访问，不会出现瓶颈。对象存储体系结构在没有损害存储系统的性能和可管理性的前提下，提供了一种完整的适合 linux 集群的存储解决方案。

对象存储系统采用分布的元数据管理对文件的并行访问有重要作用。对象存储体系结构中，对节点操作的工作负载分布到智能对象存储设备上。对象存储设备管理其数据的布局 and 拣取，维持数据对象和存储介质上实际块之间的联系。如果文件在多个对象存储设备上条块化，系统元数据管理的潜在性能将获得线性增长。另外，因为元数据管理是分布的，给系统加入额外的对象存储设备时，在提高容量的同时也提高了系统潜在的并行能力。

建立在对象存储体系结构基础上的存储系统在性能和容量方面都具有极强的扩展能力，可以解决 linux 集群中相关的管理问题。通常这些集群有若干个 NAS 系统来支持工作，其中一些 NAS 系统数据分布到系统的每个节点上。通过建立计算集群中所有节点共享的单一命名空间，使得编程和系统维护的相关工作按比例减少。对象存储体系结构中分布的智能管理功能可以减少很多与管理 and 优化系统数据分布相关的工作。例如，当计算节点要求存储新数据时对象存储设备智能地做好接收组件对象的准备，新容量自动融入到存储系统中而不需要系统管理员的干预。

对象存储体系结构包括五个主要的组件：数据对象，对象存储设备，可安装文件系统，元数据服务器和网络结构。

2.1.1 数据对象

数据对象 (Data object) 简称对象, 是对象存储体系结构中重要的组件, 是对象存储系统中数据存储的基本单元。对象中封装了数据和关于数据的属性, 包含数据和其它足够的信息允许数据自治和自我管理。对象不同于作为传统存储系统中基本组件的文件和块, 对象是应用数据 (文件数据) 和定义数据各方面的存储属性 (元数据) 的组合。元数据是“关于数据的数据”, 描述了对对象的属性, 比如对象的访问许可权限, 数据分布等。对象存储系统中, 以对象 ID 标识对象, 访问对象时通过基于对象 ID 的简单接口访问对象, 对象内部包含一定长度的字节, 这些字节的开始点以及长度是很重要的, 有了它们, 可以通过三元组 ($\langle \text{object}, \text{offset}, \text{length} \rangle$) 找到对象内部的任意字节。对象存储系统有三种不同类型的对象: 根 (Root) 对象, 组 (Group) 对象和用户 (User) 对象。存储设备上的根对象标识存储设备和设备自身的属性, 包括其总的大小和可用容量; 组对象提供存储设备上对象逻辑子集的目录; 用户对象包含实际需要存储的应用数据。

传统的块存储系统中, 存储系统必须跟踪系统中每个块的全部属性, 而对象存储系统则不同, 对象维持自己的属性并和存储系统通信告诉系统如何管理这些特殊数据。通过对数据管理进行分布, 数据自身承担了数据属性管理的工作, 简化了存储系统的管理任务, 提高了存储系统的灵活性。

2.1.2 对象存储设备

对象存储设备 (OSD, Object-based Storage Device)^[9], 是目前磁盘存储中智能化的产品, 能够存储和管理服务对象而不是简单地将数据存放到磁道和扇区上。

对象存储系统中, 应用数据 (文件数据) 由对象存储设备存储与管理。对象存储设备是对象存储体系结构定义的一种新的、更智能的磁盘接口。对象存储设备是一种包含存储介质、磁盘或磁带的网络附接设备, 它有自己的 CPU、内存、网络和磁盘系统, 具有充分的管理本地存储数据的智能性, 计算节点在取得访问许可的情况下能够直接和对象存储设备通信访问存储数据, 而不必使用文件服务器作为事务的中介, 避免了文件服务器作为计算节点访问数据的中介可能出现的瓶颈。而且, 如果文件系统将数据条块化分布在一定数量的对象存储设备上, 综合的 I/O 速率和数据流通率将线性增长。例如, 单个的附接到吉比特 (Gigabit) 以太网的对象存储设备可以达到对网络的 400Mbps 的数据输送能力和 1000/秒个存储器 I/O 操作, 如果数据在 10 个对象存储设备上条块化并且能并行访问, 综合的数据率将达到 4000Mbps 和 10000 个/秒 I/O 操作。

对象存储设备的主要功能:

①实现数据的存储

任何存储设备的主要功能是从物理介质中存储和拣取 (retrieve) 数据。与传统的存储设备相同, 对象存储设备管理分布到磁盘的磁道和扇区上的数据, 对象存储设备以外

的设备不能以数据块形式访问这些数据，而只能通过对对象 ID 访问。计算节点请求特定的对象 ID、对象内部开始读或写数据的偏移值以及请求的数据块长度，通过对象存储设备来处理数据。

②实现数据的智能化分布

对象存储设备通过缓存存储器和处理器优化磁盘上数据的分布和预取数据，对象和对象协议提供数据的额外信息用于做数据分布决策。例如，对象元数据提供要写的数据的长度，允许选择连续的一系列磁道。通过使用写回 cache，缓存大量的写数据，并且以很少的跨磁盘盘片的有效遍数来写数据。类似的，对象存储设备可以智能地对对象进行预读（read-ahead）和预取（pre-fetching），将对象数据缓存在缓冲内以提供最大的访问性能。图 2.1 为对象存储设备中数据分布和数据预取示意图。

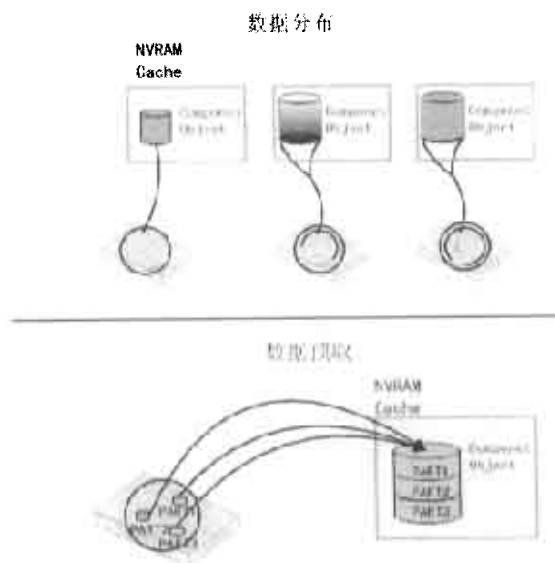


图 2.1 智能存储设备中数据分布与数据预取

③实现元数据管理

元数据类似于传统的数据，但其中包含的是与对象相关的数据块和对象的长度等信息。在传统的系统中，这种数据由文件服务器管理（对于 NAS）或者由主机操作系统管理（对于直接附接存储或 SAN）。对象存储体系结构将存储系统中大部分的元数据管理工作分布到对象存储设备上，这样降低了主机计算节点的开销。

对象存储设备可以减轻元数据服务器上元数据管理的负担。不管组件对象包含多少数据，每个对象存储设备维持自身的组件对象，元数据服务器跟踪每个对象存储设备上一个组件对象（如图 2.2）。在传统的存储系统中，元数据服务器必须跟踪每个磁盘上的每个条块化块，而在对象存储设备中不同，对象存储设备将一系列的条块单元加到初始的组件对象上，组件对象的大小是可以增长的，对于系统中的每个对象，元数据服务器

只跟踪每个对象存储设备上的一个组件对象，从而减轻了元数据服务器的负担，可以提高元数据服务器的扩展性。

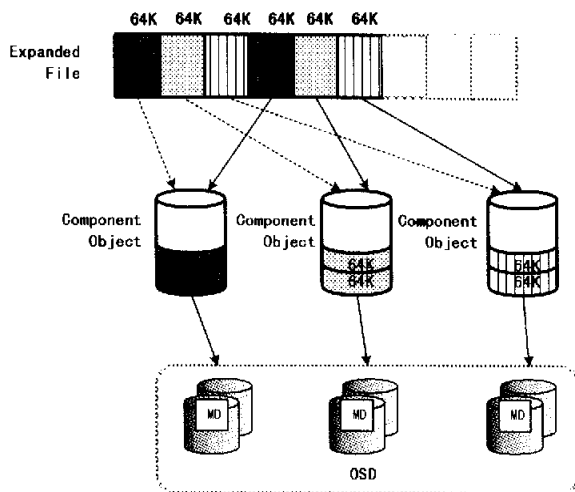


图 2.2 对象元数据管理

④保证访问的安全性

对象协议提高 SAN 或 NAS 网络存储系统上的安全性来自于两个方面的考虑。一是对象存储是采用的是一种网络协议，与网络事务相同，容易受到潜在的外来攻击。另外，其允许主机节点（类似于 SAN）到存储阵列的分布访问，这样节点可能有意或无意地（通过操作系统失效）将有害数据写到存储设备上。对象存储设备体系结构通过权能来验证客户的访问，使系统安全达到一个新的级别。每个命令或数据传输必须带有传送器（sender）和行为（action）授权的权能，这个权能是授予计算节点的安全加密令牌（token），令牌向对象存储设备描述了允许计算节点访问哪个对象，有什么特权和访问时间的长度。对象存储设备检查每个传输请求事务是否有正确的授权权能并拒绝任何没有权能的、无效的或过期的传输请求。

2.1.3 可安装文件系统

可安装文件系统（IFS, Installable File System）和用户节点相结合，可以从操作系统接受 POSIX 文件系统命令和数据，与基于对象的存储系统进行交互，使基于对象的存储系统满足用户读数据或者写数据等数据请求。

基于对象存储的可安装文件系统应该具有以下功能：能够为应用层提供 POSIX 接口，允许应用程序对底层的存储系统执行如 Open, Close, Read 和 Write 等的文件操作；计算节点中有缓存机制，缓存数据的交换；对数据的存储采用条块化管理；具有 iSCSI 协议，发送和接受对象存储设备上的数据；客户端可以从根目录下 mount 文件系统，按照其具有的权限访问相应的目录。

①POSIX 文件系统接口

分布式文件系统为应用层提供 POSIX 接口，允许应用程序对底层的存储系统执行如 Open, Close, Read 和 Write 等的文件操作。另外，支持 linux 应用程序预期的完全许可集合和访问控制，允许 Linux 应用程序可以对给定的文件独占或者共享的访问。

②缓存

一是分布式文件系统必须在计算节点中提供缓存机制，作为对象存储设备的 cache 中读入数据的补充，二是缓存写数据将多个写操作整合，有效传输和缓存 OSD 上分布的数据。三是为元数据和安全令牌维持的 cache，由于可以从缓存中得到数据访问的许可， CFS 就可以快速产生安全访问数据的访问命令。

③数据条块化

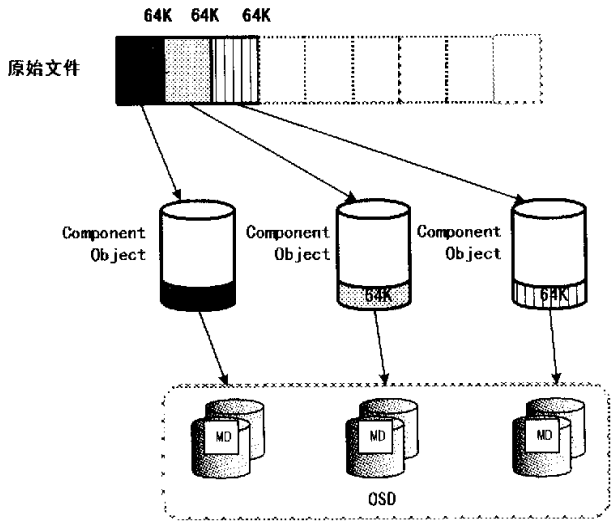


图 2.3 数据条块化

分布式文件系统可以处理跨多个对象存储设备的条块化的对象。与标准的 RAID 阵列不同，对象分布式文件系统可以对每个对象运用不同的数据分布和 RAID 级别。分布式文件系统取得一个对象并把它分解为若干组件对象，这些组件对象是发送到每个对象存储设备的对象的子集。每个组件对象的大小（条块化单元大小）和条块化宽度（对象条块化跨越的对象存储设备的数目）是组件对象的属性。由于组件对象可以进行并行的读和写，系统访问的带宽与条块化宽度有直接的关系。如果规定了 RAID 级别，则由 CFS 计算出奇偶校验单元（the parity unit）并运用到对象条块上。数据存储条块化示意图如图 2.3 所示。

④iSCSI

为了发送和接受对象存储设备上的数据，分布式文件系统必须执行 iSCSI 驱动，iSCSI 驱动中封装了 SCSI 命令集、对象对命令集的扩充以及 TCP 网络上数据的有效荷载。TCP/IP 加速器（TOE, TCP Offload Engines）能进行 iSCSI 和 TCP 协议处理，通过

这种方式卸载 (offload) 从计算节点到 TOE 适配器的 TCP 和 iSCSI 处理。

⑤Mount

所有的 CFS 在某个目录 (挂载点) 下挂载 (mount) 文件系统, 使用访问控制来决定访问文件树的不同部分, 需要诸如 Kerberos, Windows NTLM 和活动目录 (Active Directory) 的身份验证 (authentication) 机制。计算节点的身份由身份验证机制来维持, 如 Unix 系统的 UID/GIDs 和 Windows 系统的 SIDs。

⑥其他的文件系统接口

除了 POSIX 接口外, 还应具有其他的应用程序接口如 I/O 的消息传递接口 (MPI-IO, Message Passing Interface for I/O)。通过文件系统中低级的 I/O 控制接口 (low-level ioctls), MPI-IO 接口允许并行的应用程序写操作更完全地控制跨对象存储设备的数据的分布。这样可以建立宽条块以获得宽带宽, 允许集群对单个文件进行检查点检查具有更大的重启动灵活性。有两种 MPI-IO 的实现方式应用广泛, 它们是 Argonne Labs 的 MPICH/ROMIO 和 MSTI 的 MPI Pro, 均支持 MPI-2 标准。

2.1.4 元数据服务器

元数据服务器 (MDS, Meta-Data Server), 是对象存储系统中多个计算节点之间的中介, 维持所有节点上缓存数据的一致性, 保证计算节点共享数据。

对象存储系统中, 元数据服务器存储和管理元数据。目前对象存储体系结构的设计是单个的元数据服务器^[1,7], 其有两个功能: 一是向计算节点提供存储数据的逻辑视图 (虚拟文件系统 Virtual File System or VFS layer)、文件名列表和组织文件的目录结构; 二是组织物理存储介质中数据的分布 (the inode layer)。元数据服务器能够进行身份验证、文件和目录访问管理、实现 cache 一致性、进行容量管理, 并保证扩展性。

对象存储系统将存储数据的逻辑视图 (VFS layer) 与物理视图 (the inode layer) 分开, 通过对工作负载进行分布可以克服 NAS 中出现的元数据服务器瓶颈, 从而提升对象存储设备的潜在性能。元数据的 VFS 部分完成典型的 NFS 服务器中大约 10% 的元数据工作负载, 而剩下的 90% 的元数据工作负载由 inode 层存储在存储介质块上的物理分布数据做的。

元数据作为“关于数据的数据”, 是存储系统的中心信息, 缺少了元数据, 对象数据将成为“孤儿”数据, 变成无用的信息, 元数据的管理对于确保整个存储系统的可靠性、高性能等具有决定性作用。在大型应用中, 单个的元数据服务器可能无法承担元数据服务的工作, 从而使元数据服务成为存储系统的瓶颈, 影响存储系统的整体性能。我们可以采用可伸缩的集群方式来管理元数据, 将元数据的服务负载分布到集群中的节点上, 以集群作为一个整体来完成元数据服务工作。

2.1.5 网络结构

网络结构 (Network Fabric) 建立起计算节点到对象存储设备和元数据服务器之间的连接, 使它们之间能够互相通信。

网络是对象存储系统的关键元素。网络提供连通性的结构, 在单一的结构中绑定对象存储设备、元数据服务器和计算节点。随着廉价的吉比特以太网的应用, 可能以达到甚至超过特定的如光纤通道 (fibre channel) 存储传输速度来传输存储流量, 给对象存储系统带来了好处。以太网的日益普及降低了组件成本, 更重要的是, 建立可靠的以太网的知识非常普及, 降低了管理成本。然而, 对象存储系统只和 TCP/IP 结合在一起, 而不是以太网, 可以利用其他传输方式如 Myrinet 和 InfiniBand 对 TCP/IP 的支持建立对象存储系统。对象存储系统有三中主要的网络协议:

①iSCSI 协议。iSCSI 协议在 TCP/IP 包中封装了 SCSI, 是传输命令和数据到 OSDs 的基本传输协议。SCSI 命令数据块 (CDB, Command Data Blocks) 将读写数据命令传输到存储设备, 采取和数据本身一样的方式传送。iSCSI 协议得到扩展并能支持对象命令集合, 但仍然保持在 iSCSI 协议定义的范围内。

②RPC 命令传输。对象存储系统在计算节点和元数据服务器之间的通讯中使用了一种单独的协议, 这就是轻权的远程过程调用 (lightweight Remote Procedure Call, RPC), 使得与元数据的快速通信比较容易。

③其他服务协议。对象存储系统的建立还需要很多标准的 TCP/IP 服务。例如, 使用 NTP (Network Time Protocol, 网络时间协议) 实现计算节点和存储系统间的同步, 使用 DNS (Domain Name Server, 域名服务器) 简化地址的转换和维持, 使用各种路由协议允许计算节点和存储系统分离。这些服务全部建立在 TCP/IP 协议之上, 具有广泛的可用性和互操作性, 利用好这些协议, 对象存储系统可以得到很大的好处。

§ 2.2 对象存储文件系统原型的建立

2.2.1 对象存储文件系统原型的设计思想

基于对象存储的思想提供了一种新颖的框架, 采用这个框架可以建立适合集群系统的分布式文件系统——对象存储文件系统。对象存储文件系统能够发挥基于对象存储思想的优势, 具有高性能、高可用性和强可伸缩性。对象存储文件系统的核心是将数据通路 (数据读或写) 和控制通路 (元数据) 分离, 并且基于智能的对象存储设备构建存储系统。对象存储文件系统通常由以下几部分组成^[41]。

(1) 对象

对象是系统中数据存储的基本单位, 一个对象实际上是文件数据和一组属性的组合。其中属性可以定义基于文件的 RAID 参数、数据分布和服务质量等, 而传统的存储系统中以文件或块作为基本的存储单位, 在块存储系统中还需要始终追踪系统中每个块

的属性,而对象通过与存储系统通信维护自己的属性。在存储设备中,每个对象都有一个对象标识,OSD 命令通过对象标识访问对象。通常有多种类型的对象,存储设备上的根对象标识存储设备和该设备的各种属性,组对象是存储设备上共享资源管理策略的对象集合。

(2) 对象存储目标

对象存储目标 (OST, Object Storage Target) 管理对象存储设备。对象存储设备具有一定的智能,它有自己的 CPU、内存、网络和磁盘系统,目前国际上通常采用刀片式结构实现对象存储设备。OSD 提供数据存储、智能分布、管理对象元数据的功能。

(3) 元数据服务器

元数据服务器 (MDS, Metadata Server) 控制 Client 与 OST 的交互,主要提供以下几个功能:

①对象存储访问。MDS 构造、管理描述每个文件分布的视图,允许 Client 直接访问对象。MDS 为 Client 提供访问该文件所含对象的能力,OSD 在接收到每个请求时先将验证该能力,然后才可以访问。

②文件和目录访问管理。MDS 在存储系统上构建一个文件结构,包括限额控制、目录和文件的创建和删除、访问控制等。

③维护 Client Cache 一致性。为了提高 Client 性能,在对象存储文件系统设计时通常支持 Client 方的 Cache。由于引入 Client 方的 Cache,带来了 Cache 一致性问题,MDS 支持基于 Client 的文件 Cache,当 Cache 的文件发生改变时,将通知 Client 刷新 Cache,从而防止 Cache 不一致引发的问题。

(4) 客户端

为了有效支持用户访问存储系统中存储的数据,需要在计算节点实现对象存储文件系统的 Client 端。Client 通常向用户提供 POSIX 文件系统接口,允许应用程序像使用普通的文件系统一样,通过普通的文件系统操作和调用实现数据访问和操作。

2.2.2 对象存储文件系统原型 OCFS 的建立

在基于对象存储思想的基础上,设计实现了一个小型的文件系统原型——OCFS (Object-based Cluster File System)。OCFS 采用基于对象存储的思想,以对象方式管理文件,将用户文件分为元数据和实际数据,分别由不同的子系统进行管理。OCFS 的体系结构类似于其他基于对象存储的文件系统,包含三个子系统:客户端文件系统 (CFS, Client File System),元数据服务器 (MDS, Metadata Server) 和对象存储目标 (OST, Object Storage Target)。OCFS 的三个子系统分别有不同的功能,它们有机地结合在一起协同工作,成为一个文件系统,提供文件系统具有的功能。OST 管理基于对象存储设备,对其上存储的文件数据进行存储和管理,并向客户端提供文件数据 I/O; MDS 管理文件系统的元数据,将文件元数据和目录形成全局统一的名空间; CFS 与 MDS 进行名空间操作的交互,向应用程序提供一个全局统一名空间的文件系统,与 OST 进行文件数据

I/O 的交互，为应用程序提供文件服务。OCFS 作为内核模块被安装，提供标准的文件系统接口，能够通过 VFS 与用户进行命令和数据的交互，接受用户的命令和数据请求，与用户进行数据交换，提供用户需要的数据，按照用户的要求对自身数据进行更新。OCFS 可以在集群范围内提供 UNIX 文件共享语义，满足多个客户共同使用文件系统。图 2.4 为 OCFS 文件系统子系统之间的交互图。下面简单阐述一下 OCFS 各个子系统实现的思想及软件模块组成。

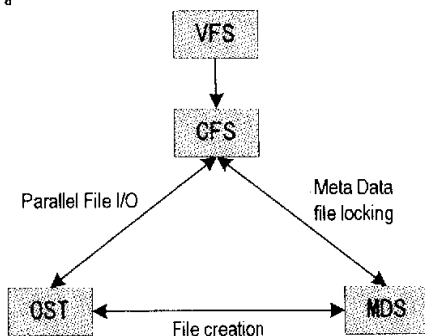


图 2.4 OCFS 文件系统子系统交互图

(1) 对象存储目标

对象存储目标管理文件数据在对象存储设备上的存储，提供文件对象数据的 I/O 服务。在 OCFS 中便于简单实现，以文件目录模拟智能的对象存储设备。对象存储目标采用如图 2.5 所示的软件模块实现。

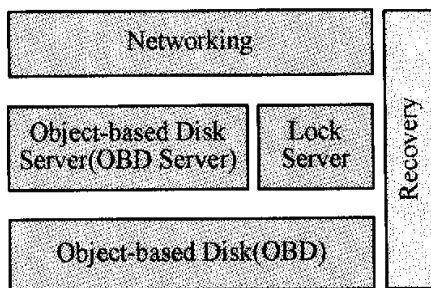


图 2.5 OST 软件模块

Networking 网络层，OCFS 中使用 Sun RPC 作为 CFS、MDS 和 OST 之间进行数据和命令传输的基础结构，使用同步的 RPC 处理命令传输，使用异步 RPC 传输数据。

OBD Server 是直接对象存储驱动程序，管理在 OBD 设备上存储的数据，包括数据的空间分配和读写操作等。在 OCFS 中，以文件系统中的某个目录模拟对象存储设备，借用文件系统来管理数据。

Lock Server 执行锁管理机制，按照锁语义锁定数据块。

(2) 客户端文件系统

客户端文件系统的作用是与 OST 和 MDS 交互。客户端文件系统根据用户的访问权限,组织用户可以访问的元数据和文件数据,向用户提供文件系统的全局视图和目录树,目录树可以分为多个文件集合,这样可以在集群范围内提供 UNIX 文件共享语义。客户端文件系统通过元数据句柄与元数据服务器交互,也就是 inodes 和目录信息的更新。客户端文件系统与对象存储目标之间的协议包含了文件 I/O 协议,其中包括块的分配、分条和安全措施。

客户端文件系统以 linux 内核模块的方式实现,当模块装入时在内核进行登记,用户进行 mount 后模块就被使用。客户端文件系统定义了超级块操作、索引节点操作、目录项操作、文件操作、地址分配操作等操作,与 VFS 交互并将相应的例程分发给负责的 MDS client 和 OBD client。

CFS 由以下几部分软件模块组成(图 2.6):

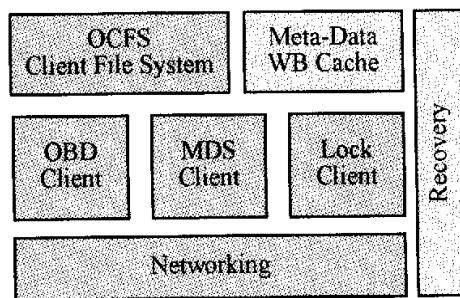


图 2.6 客户端软件模块

MDS client 包括支持 client 与 MDS 之间通信的例程, MDS client 作为一个中转站将 VFS 的请求转发给 MDS, 从元数据服务器获取元数据或者更新元数据服务器上的元数据。

OBD client 主要用于与 OST 之间进行交互, 从 OST 上读取或者向 OST 写入数据, OBD client 提供应用程序接口 API 支持 CFS 存取和更新 OST 上的文件。在 OBD client 与 OST 的事务之前, CFS 应预先从元数据服务器上取得文件的分布信息。为了利用好网络的性能, 缓解网络高峰, 起到放大网络性能的作用, OCFS 采用在 NFS 中采用的写回 cache 机制。

Lock Client 提供锁管理机制。OCFS 中, 采用简单的锁策略。策略如下: 写操作前, Lock Client 申请获取锁, 并且只允许一个用户程序拥有锁; 不需要请求文件锁可对文件进行读操作。文件锁定结束后, 应该释放文件锁。

恢复机制要求在系统失效的情况下能够对加锁的对象解锁。采用这样一个策略, 锁定文件进行写操作时, 给锁一个租用时间的超时值。需要继续使用锁时, 需要在超时之前重新租用锁, 使用这个策略时, MDS 在撤销锁之前要考虑网络的延迟。

Networking 网络层, OCFS 中使用 Sun RPC 作为 CFS、MDS 和 OST 之间进行数据和命令传输的基础结构, 使用同步的 RPC 处理命令的传输, 使用异步 RPC 来传输数据。

(3) 元数据服务器

元数据服务器管理文件系统的元数据，其软件模块如图 2.7 所示。MDS Server 提供元数据服务，根据用户的权限按照用户的要求进行元数据操作；OCFS 中元数据后端存储采用 SQL 数据库来存储和组织元数据，支持元数据的各种操作，包括元数据的插入、删除、更新等，在数据库中，包括几个表，分别管理相应的元数据信息：mds——文件的如文件类型、大小、访问时间、存储的位置等属性，dirs——文件的目录树结构，links——目录的链接，ostlist——文件存储设备列表，usedoid——管理全局统一的对象 id 的分配。

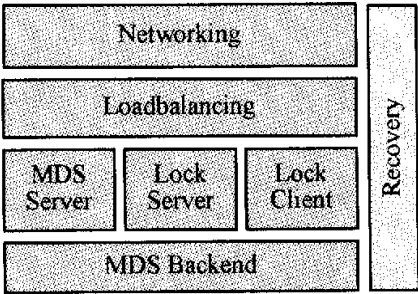


图 2.7 MDS 软件模块

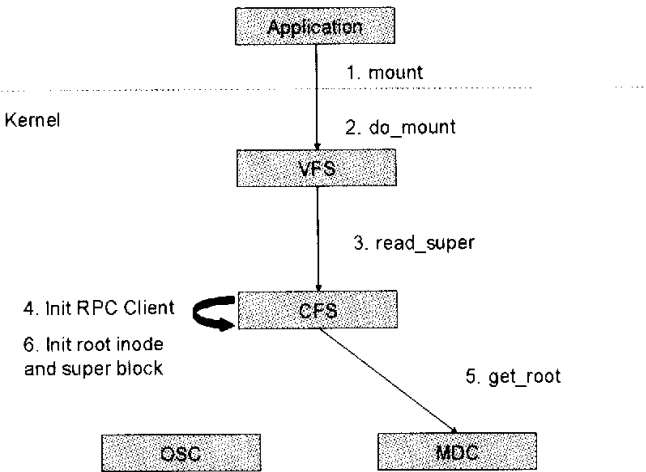


图 2.8 mount 流程

用户要使用文件系统访问其中的数据时，需要 mount 文件系统（图 2.8）。OCFS 以 Linux 内核模块的方式实现，当 mount 文件系统 OCFS 时，在内核中注册一个文件系统：

```
static DECLARE_FSTYPE(ocfs_fs_type, "ocfs", ocfs_read_super, 0);
```

VFS 调用 ocfs_read_super 来安装 OCFS 的 super block 域。

OCFS 系统实现了文件系统的 POSIX 接口，允许用户通过 VFS 执行如 Open, Close, Read 和 Write 等文件操作，它们的内部处理过程如图 2.9—2.11 所示。

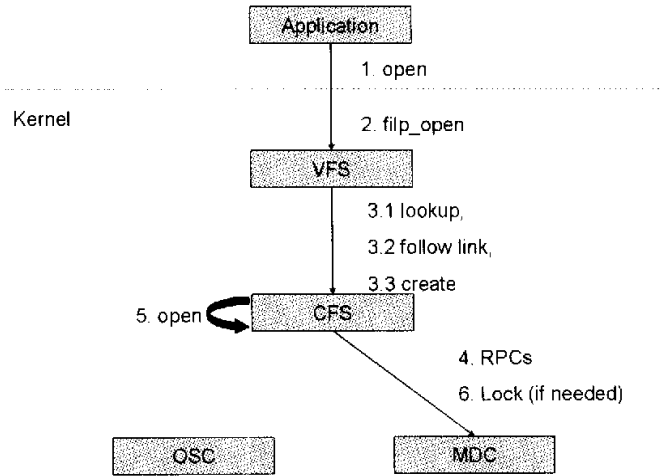


图 2. 9 Open 流程

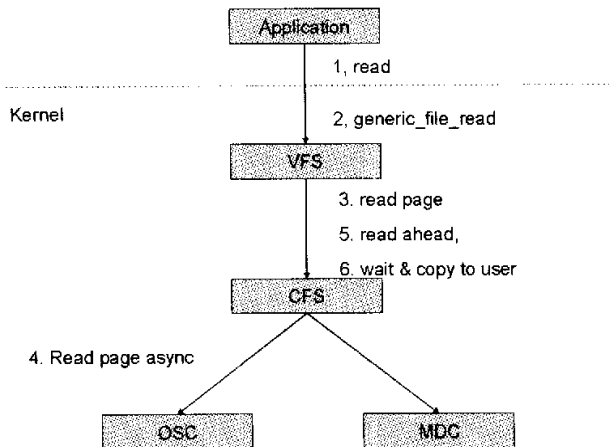


图 2. 10 Read 流程

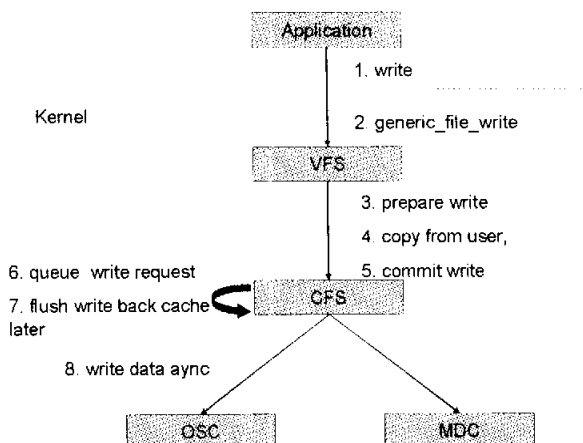


图 2. 11 Write 流程

本章主要分析对象存储系统的体系结构，提出并实现了小型的对象存储文件系统 OCFS。OCFS 采用对象存储体系结构的结构框架，包括 CFS、OST 和 MDS 三个子系统，以对象方式管理文件，提供标准的文件系统接口和 UNIX 文件共享语义，满足多个客户共同使用文件系统。但元数据服务器只能是单服务器服务，在大型应用中使用元数据服务器可能会成为系统性能和扩展的瓶颈。下一章将在本章研究内容的基础上，着重研究对象存储文件系统中的元数据服务，采用新的结构来实现以集群方式管理元数据。

第三章 改进的对象存储文件系统 OCFS II 的设计

§ 3.1 元数据管理

3.1.1 元数据管理目标

对于大型的基于对象的分布式文件系统,避免瓶颈对获得高性能和高可伸缩性是至关重要的。虽然元数据的大小在存储容量上仅占很小的比例,然而有 50%—80%的文件系统访问是访问元数据,元数据的访问就是一个潜在的瓶颈,而且元数据的管理也是对对象文件系统中一个复杂的部分,所以采取怎样的方式有效地管理好元数据是获得系统高性能和高伸缩性的前提。

系统中,通常使用元数据服务器(MDS)管理元数据,目前的设计是单个的元数据服务器^[1,7],其有两个功能:一是向计算节点提供存储数据的逻辑视图(虚拟文件系统 Virtual File System or VFS layer),文件名的列表和组织文件的目录结构;二是组织在物理存储介质中数据的分布(the inode layer)。

MDS 控制 CFS 和 OSD 上对象的交互,协调 CFS 的访问,对它们进行适当的授权,维持对相同文件用户的 cache 一致性。元数据服务器为系统提供下列服务:

①身份验证

MDS 对试图加入存储系统的对象存储设备进行识别和身份验证,MDS 提供新的存储系统成员凭证并且周期性的检查/刷新这些凭证确保成员的有效性;同样,当客户访问数据时,MDS 提供身份识别和身份验证。

②文件和目录访问管理

MDS 向 CFS 提供存储系统的文件结构。当一个节点请求对一个特定的文件执行操作时,MDS 检查文件相关的许可和访问控制权限并向请求节点提供映象(map)和权能(capability)。映象由包含请求的对象的组件的 OSD 列表和它们的 IP 地址组成。权能是提供给 CFS 的安全性和加密令牌(cryptographic token),由和每个事务相关的 OSD 检查。令牌向 OSD 描述了允许计算节点访问哪个对象,有什么样的特权和可以访问多长时间。

③cache 一致性

为了达到最高的性能,CFS 请求相关的对象并读取数据后,在本地缓存数据。如果有使用相同文件的多个节点,要采取措施保证在任何一个节点修改文件后更新本地 cache。当一个 CFS 向 MDS 要求对一个文件或文件部分的读和写权利时,MDS 登记对权利的复查。如果文件权利授予多个写操作,其他的节点修改了文件,MDS 产生一个对打开文件的所有节点的复查,让节点作废它们本地 cache 中的内容。这样,如果一个

节点对更新了的文件读访问时，它必须重新连接 OSD 更新本地缓存的数据备份，保证对同样的数据操作所有的节点的一致性。

④容量管理

MDS 必须跟踪整个系统 OSD 的容量的平衡和使用，确保整个系统对可用的磁盘资源优化使用。当计算节点要求创建对象时，MDS 在授权节点写新数据时要决定如何优化新文件的分布。因为节点在文件操作时不知道文件究竟有多大，MDS 向节点提供一个关于空间的契约 (escrow) 或者配额 (quota)。这样允许节点一个步骤创建和写数据使写操作的性能最高。在关键的写操作期间保持最高性能，关闭文件后则恢复任何过多的配额。

⑤扩展

元数据管理使系统具有能够在容量和性能上扩展的关键的结构优势。因为基于对象存储的系统将文件/目录管理和块/扇区管理分离，系统的扩展性能高于其他的基于存储结构的系统。从容量和性能的平衡方面讲，MDS 的可扩展性是允许整个基于对象存储的系统扩展的关键。

3.1.2 元数据后端存储管理

虽然元数据的尺寸很小，但在大型文件系统中，元数据也有惊人的数量。例如，包含超过十亿个文件的 petabyte 文件系统的元数据可能占用 1TB 级的空间甚至更多。这么多的数据不可能完全驻留在元数据管理集群的合作 RAM 中，同时要考虑到元数据服务器失效后能够得到一致性的恢复，元数据必须存储到持久存储器中。理想的情况是，元数据管理系统的存储器缓存能够满足大多数的读操作，但是也需要周期性到磁盘拣取请求的信息，并且所有的更新必须保存到磁盘这样的持久存储设备上。通常来说，需要有共享的元数据存储库（例如由 OSD 组成），它们应适合采用多种分割策略组织分配数据，并且在处理 MDS 失效和利用更通用的硬件方面比直接附接存储更有优势。元数据管理系统的后端在持久存储器上存储元数据，存储的对象集合的分布与文件系统类似，主要的不同之处是元数据管理系统中的文件索引节点包含的是指向其它存储对象的指针而不是文件数据。

元数据管理系统应满足这样的一些要求：首先，支持故障恢复。在系统运行过程中，系统可能出现故障，包括网络故障和节点故障，系统产生失效，正在执行的数据丢失。当故障排除后，元数据管理系统能在持久存储器中存储元数据的基础上建立与故障前一致的元数据文件系统，恢复元数据服务；其次，允许预分配 fid 创建对象。如果 fid 预分配以写回 cache 方式进行。在写回 cache 方式中，即使更新/更改会在一段时间以后才写到服务器的持久存储中，一旦本地 cache 中的事务完成，将通知 CFS 操作已经完成，这样的情况下，需要确保正在服务器执行的操作不会失败，否则就会出现元数据系统不一致的情况。可恢复的要求意味着在事务流的维护中保证所有的文件系统事务原子地执行。

文件系统和数据库是元数据后端存储管理的两种有效方式，各有其优缺点。

(1) 文件系统管理

元数据服务器中管理元数据后端存储最自然的方式是使用文件系统管理。文件系统是操作系统在计算机存储设备上存储和检索数据的逻辑方法，计算机的存储设备可以是本地驱动器，可以是在网络上使用的卷或者存储区域网络（SAN）上的共享。文件系统实现了 Unix 式操作系统需要的基本操作，可以进行文件的创建和删除、打开以文件进行读取和写入、在文件中搜索、关闭文件和创建目录存储一组文件等。

使用文件系统管理元数据的好处是便于进行数据恢复，可以采用磁盘容错的一些管理办法；方便进行文件数据的锁定；能够重用 CFS 与 OST 系统间的协议实现共享元数据文件系统；可以继承灵活的 API 存储和管理元数据，例如与安全性相关的 API。

文件系统管理元数据存在以下缺点，除非写操作有复杂的能够在多个后端存储设备之间执行事务的逻辑模块，否则单个文件集合的元数据只会存储在唯一的后端存储设备上。这样的情况下，对后端存储设备的写入数据是不能充分利用多个设备的优势，可能出现某些设备闲置，而某些设备特别繁忙的情况。在单个文件集中出现极端繁重的元数据流量时，管理元数据的文件系统可能成为潜在的瓶颈。

(2) 数据库管理

第二个存储元数据的方式是使用高端集群数据库进行存储和管理，比如 SQL server。这种管理方式选择高效的第三方数据库软件，实现对元数据的管理，可以从选择合适的第三方软件中受益。数据库管理方式容易处理不同形式的元数据，功能强大，管理方便，运行速度快，可以提供灵活的查询、插入和删除等操作，提供的 API 可用于各种语言，方便进行编程；能够进行访问控制，根据用户的访问权限提供数据访问，实现对数据的安全管理；具有并发控制机制，支持多用户同时使用数据。使用数据库管理元数据的缺点是需要依赖于第三方的软件。

§ 3.2 元数据管理集群

3.2.1 元数据集群管理目标

基于对象存储的分布式文件系统常用于大型的集群系统，为适应集群的需要，必须保证整个系统具有很强的可扩展性，并且能够运行在通用的硬件上，达到很高的性价比。元数据服务是整个文件系统的核心，元数据服务的快速、高效、可扩展是整个文件系统高可扩展和高性能的前提。集中的单个服务器很难满足这种需求，因为元数据服务极易成为整个系统的瓶颈，造成整个文件系统性能的下降。所以，通常应对元数据提供集群化的管理，采用元数据管理集群来管理元数据，保证系统的元数据服务的可扩展性和性能，达到提高分布式文件系统整体可扩展性和性能的目的。

实现元数据管理集群要达到以下几个目标：

①保证性能大幅度提高。各个客户对元数据的需求和操作都是独立和随机的,大量的客户会并发和同时访问,客户访问之间有很大的并行性。组织大量元数据操作的可并行操作,可以保证元数据管理集群同时服务于多个客户,从而使系统整体的元数据服务性能获得线性的增长。通常,在元数据管理集群上均匀分布元数据操作,引入RPC(remote procedure call, 远程过程调用)以避免复杂的协议。分布式文件系统设计允许 CFS 预计算(pre-compute)正确服务的位置来实现并行化,分布工作负载,提高系统整体效率。

②提供良好的负载均衡机制和资源分配工具。元数据管理集群作为一个整体完成元数据服务工作,应表现出一个很高性能的元数据服务器进行元数据服务工作的效果。元数据管理集群必须是一个负载均衡的集群,元数据服务工作负载能够在集群上均衡分布,不会因为某个服务器过载而影响集群的整体性能。负载均衡的过程也是资源分配利用的过程,通过负载均衡,合理利用集群中的资源,发挥它们的最高效能,发挥集群的高性价比优势。元数据管理集群的负载均衡机制和资源分配工具应该要便于管理,它们既要适合大型安装,还要保证在小集群的情况下,元数据服务器能访问集群中其他节点上的元数据。

③提供高可用性。元数据管理集群的服务具有高可用性,这意味着在协议方面可以对客户(CFS)掩盖服务器失效(server failure)和对其他客户掩盖客户失效(CFS failure)。元数据管理集群在单点 MDS 失效情况下能进行透明恢复,并能在失效后恢复一致的状态。基于大型集群的经验,多点同时失效是很少出现的。基于这个前提假设,系统的性能急剧提高。

④具有良好可伸缩性。元数据服务器采用集群系统的原因之一是提供高性能的元数据服务,防止元数据服务成为整个系统的瓶颈,另一个原因就是集群具有很好的可伸缩性,可以满足分布式文件系统不断扩展的元数据服务需要。

元数据管理集群要达到上述目标,从实现上讲需要解决元数据分割和集群体系结构的问题。只有在合适的集群体系结构的基础上实施有效的数据分割,实现数据在集群中节点上的分流,才能保证整个集群协同工作,提供高性能。

3.2.2 元数据的分割

随着系统的增大,系统中有越来越多的元数据,例如在包含超过十亿个文件的 PB (petabyte) 级文件系统的元数据可能占用 1TB 级的空间甚至更多^[1],最终所有的元数据必须存储在一些永久磁盘存储设备上,集群如何管理这些元数据就涉及到元数据的分割。通过元数据的分割,可以在集群中的节点之间分配元数据,使得在通常情况下能够利用好集群中的可用资源,能够负载均衡地处理元数据操作的工作负载;并且,元数据管理集群系统能够有效处理偶然出现的极端元数据操作工作负载,比如数千个 Client 同时打开相同的文件或写同一个目录,这样的元数据工作负载必须在元数据管理集群上有效分布,将这些工作负载分配给集群中的若干个服务器,而不致某个元数据服务器出现过载。元数据分割后,元数据服务器之间的缓存开销应该最小,而且元数据管理集群对

下面的元数据存储子系统掩盖 I/O 请求的能力最大, 不让元数据存储子系统感觉到 I/O 请求的压力。元数据管理集群系统能够运用失效机制增强整个系统的性能, 失效节点工作负载重新分布到其他的服务器上或者备用服务器上, 使整个系统的性能不受影响或者影响甚微。元数据的分割应该是动态的, 当存储系统的尺寸增加时, MDS 集群能够扩张到周围额外的服务器上, 并以最小的代价分布工作负载。

在现有的分布式文件系统中, 元数据分割主要有两种方法, 一种是目录子树分割 (directory subtree partitioning) 方法^[15], 另一种是纯散列 (pure hashing) 方法^[16]。这两种方法各有优缺点, 所以在两种方法之外, 应寻找综合的方法, 以发挥两种方法的优点, 避免它们的不足。

(1) 目录子树分割方法

目录子树分割方法, 按照目录子树分割元数据名空间, 每个元数据服务器管理整个目录树中的一个或多个子目录, 如 NFS、AFS、Coda、Sprite、LOCUS 等都是使用这种方法来管理元数据。静态子树分割需要系统管理员决定怎样分布文件系统, 并且人工地将目录层次的子树分配到单个的文件服务器上。因为目录层次的子树认为是独立的结构, 静态的分布简化了客户端识别负责元数据的服务器的任务, 服务器能够不和其他节点通信就能处理请求。其主要优点是, 同一目录子树中的元数据保存在同一元数据服务器中, 对该目录子树下文件的元数据获取只需要访问同一元数据服务器即可, 能够有效的利用用户端的缓存机制。这种方法允许文件系统宽度扩展, 但不能深度扩展。静态分割集群适应这样扩展的文件系统, 其增长在可用的服务器上保持均匀的分布, 或者新的数据排它地写到分配给新的层次部分的新服务器上。然而, 文件系统很难按这样规则的模式扩展, 需要人工重新分配层次适应新的数据甚至增加已有数据的 CFS 请求。其主要缺点是不能有效地平衡元数据服务器之间的工作负载, 负载重的服务器可能成为系统的瓶颈; 在可伸缩性方面, 若增加或减少元数据服务器, 不能有效地重新平衡元数据服务器之间的工作负载; 必须遍历文件路径的所有目录才能确定该文件的访问权限。

(2) 纯散列方法

按照如 inode 名称和路径名这样的文件标识执行 hash 算法将元数据分配到各元数据服务器, 解决了目录子树分割方法不能有效平衡元数据服务器之间工作负载的缺点。典型的 hash 算法是对文件全路径名进行 hash 计算来分配和定位元数据, 其优点是不需要遍历文件路径所有目录就能直接定位文件的元数据, CFS 能够直接定位和连接负责的 MDS; 由于采取负载均衡的和良好行为的 hash 函数, 请求均匀分布到集群上, 实现工作负载在集群上的均衡, 消除了层次目录结构中行为的“热点”问题。例如在单个目录中同时创建很多个文件, 因为元数据的位置与目录层次没有关系, 创建的元数据不关联到同一个元数据服务器中, 不会出现某个服务器过载的情况。缺点是修改目录时, 因为 hash 输入的改变, 该目录下所有元数据的服务器需要更改, 导致大量的元数据迁移。

为了满足 POSIX 访问语义, MDS 集群必须遍历包含请求数据的前缀 (祖先) 目录来确保目录是否给以当前用户对涉及到的数据和元数据的访问权限。因为采用 hash 算

法的分布元数据，整个目录层次的文件和目录分散和定位于每个 MDS 中，这时的遍历有很高的开销，这些开销来自于遍历分散在多个服务器上的元数据和来自于本地复制的前缀缓存。节点间的前缀 cache 有很大的开销因为父目录索引节点必须复制到正在服务于其中的一个和多个子目录的每个 MDS 上，消耗掉可以存储其他数据的存储资源。

(3) 混合方法

Brandt^[10]结合目前目录子树分割和散列方法的优点，提出了一种管理元数据的混合方法 Lazy Hybrid (LH)。这种方法基于文件路径名进行 hash 计算来快速定位文件的元数据，用双入口访问控制表存储每个文件元数据的整个路径遍历的有效访问信息，只有在访问控制需要更新时才进行文件路径的遍历，克服了必须遍历文件路径的所有目录才能确定该文件访问权限的缺点。但是，当目录属性更改时（如删除目录，更改目录名，修改目录权限）或者增加、删除元数据服务器时，需要重新计算 hash 值从而导致大量的元数据迁移。

3.2.3 元数据管理集群采用的体系结构

由于互联技术、节点的复杂度和耦合程度的不同，常用的计算机集群主要有三种体系结构——无共享体系结构，共享磁盘体系结构和共享存储器体系结构^[33]。

(1) 无共享体系结构

无共享体系结构是目前大多数集群采用的方式。每个节点都是独立的 PC 或者工作站。集群的每一个节点都是完整独立的操作系统和硬件设备集合。节点之间通过局域网或者开关阵列以松耦合的方式连接起来，彼此分享节点的部分甚至全部可用资源：CPU、内存、磁盘、IO 设备等等，以形成一个对外单一、强大的计算机系统。这类系统对 SSI（单一系统映像）的能力较弱，需要特殊的中间件或者 OS 扩展加以支持。

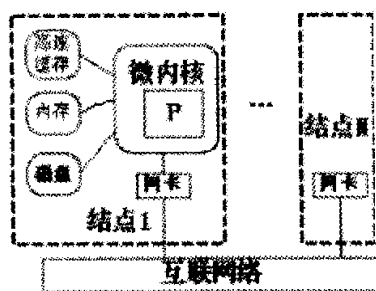


图 3.1 无共享体系结构

(2) 共享磁盘体系结构

共享磁盘体系结构的节点基本上仍是独立的计算机，没有或者不使用本地的磁盘文件系统。分布式文件系统正是这类体系结构的应用体现。常见的 NFS、AFS 或者 GFS 都属于这个范畴。而硬件上的解决常常通过共享磁盘阵列或者 SAN 来实现。该体系结

构主要能够解决区域存储空间的容量问题，通过构造单一的虚拟文件系统，提供给整个集群一个巨大的存储设备。尤其在一些高可用的场合，共享磁盘阵列常常能够解决文件系统容错和数据一致等可靠性问题。

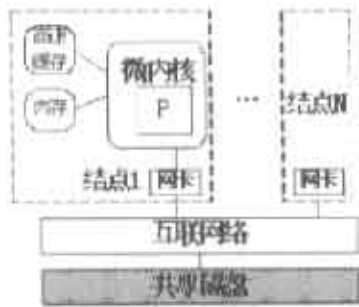


图 3.2 共享磁盘体系结构

(3) 共享存储器体系结构

共享存储器体系结构集群最不易实现，具有较强的 SSI 能力。从实现的难度上讲，不论是硬件制造的复杂性还是软件的实现难度，这种体系结构都大大超过其他几类体系结构的实现。实现这类体系结构的集群系统有 DSM（分布式共享存储集群）、NUMA、ccNUMA 等技术。在这类体系结构中，可以将多个节点的计算资源集合在一起，形成一个内存空间一致的单一系统。这样的系统具有最好的 SSI 能力。

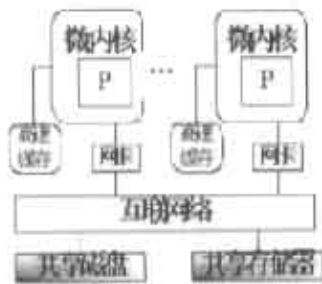


图 3.3 共享存储器体系结构

元数据的管理集群可以采用上述任何一种体系结构，或者采用使几种相混合的方式。在本文中 OCFS II 系统采用 DPIS 和 MDS 两个集群管理元数据，DPIS 采用共享磁盘体系结构，MDS 采用无共享体系结构，实际就是采用的一种混合方式。

§ 3.3 原型改进的文件系统 OCFS II

3.3.1 OCFS II 的体系结构和设计思想

为了实现元数据管理集群，在以上分析研究的基础上对基于对象存储文件系统原型

进行改进,提出一种运用集群管理元数据的对象存储文件系统 OCFS II (Object-based Cluster File System Improved version)。

(1) OCFS II 的体系结构

OCFS II 系统中包括四个部分,即客户端文件系统 (Clnet File System, CFS), 目录路径索引服务器 (Directory Path Index Server, DPIS), 元数据服务器 (Metadata Server, MDS) 和对象存储目标 (Object Storage Target, OST), 各个部分通过高速网络互连。客户端文件系统和对象存储目标的功能与其他基于对象存储的分布式文件系统上的相应部分相同,而目录路径索引服务器和元数据服务器共同完成元数据管理工作。在系统中,目录路径索引服务器管理系统的目录元数据和整个文件系统中需要一致和共享的数据,多个 DPIS 组成 DPIS 集群,采用共享存储的集群体系结构,其中每个 DPIS 都能完成相同的目录元数据服务;元数据服务器管理属性元数据等独立的元数据,多个 MDS 组成 MDS 集群,采用分布独立存储系统的体系结构,其中每个 MDS 存储相应的部分元数据,提供相应的元数据服务。DPIS 集群和 MDS 集群合称为元数据管理集群 (MMC, Metadata Manage Cluster)。其结构如图 3.4 所示。

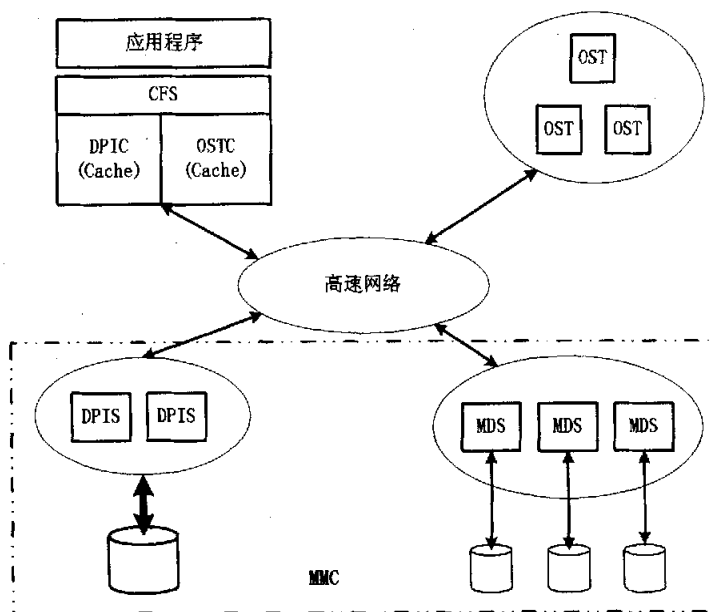


图 3.4 OCFS II 的体系结构

元数据管理集群提供整个文件系统的元数据服务。OCFS II 中,对元数据实现层次管理,分两级访问元数据,第一级是访问 DPIS 中目录路径元数据,在文件系统名空间中定位元数据;第二级是访问 MDS,对文件自身元数据进行操作。元数据管理集群中,DPIS 集群是主体,所有的元数据操作请求都是通过 DPIS 集群处理的。请求若是文件目录操作,处理请求一般在 DPIS 内部进行;若需要进行文件自身元数据操作,则根据文件对象的文件 ID,执行元数据分配和定位算法确定元数据所在的 MDS 服务器,并向

MDS 服务器发送元数据处理请求，进行元数据处理操作。为减少对物理设备和数据访问带来的网络通信开销，OCFS II 的各部分都带有缓存机制，缓存调入内存的元数据。在 DPIS 中，使用元数据高速缓存区缓存 CFS 访问过的元数据，CFS 访问元数据时，元数据请求到达 DPIS，首先查找高速缓存链表，如果有相应的数据，则直接返回数据给 CFS，否则才访问 MDS。

图 3.5 是 OCFS II 的组成和各部分功能示意图，描述了系统各部分之间的交互关系，可以看出元数据管理集群对元数据的层次管理模型。

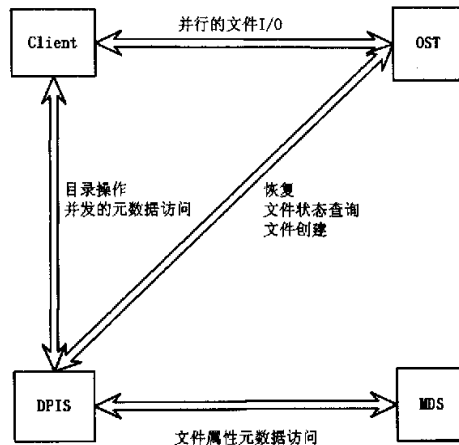


图 3.5 OCFS II 文件系统组成示意图

在 OCFS II 中，任何文件的数据分为三部分：是目录路径元数据、文件本身元数据与数据对象。目录路径元数据包括文件的文件名和文件访问控制属性，这些数据可以组成整个文件系统的名空间，保存在 DPIS 中；文件自身元数据包括文件的属性如文件大小、所有者、最近访问时间、数据对象的位置等，保存在 MDS 中；数据对象包含文件的实际数据，保存在 OST 管理的对象存储设备中。

OCFS II 中，实际的数据块分配由 OST 管理完成，文件的数据块在 OST 管理的 OSD 中可以实现条块化存储，一个文件可以包含若干个数据对象，因此，OCFS II 中的文件大小不再受限于传统设计的数据块的数量，能够支持超大规模的文件以及超大容量的目录。用户访问文件时首先通过 CFS 访问元数据管理集群中的 DPIS 与 MDS，确定对文件的访问权限和获取文件的位置信息，然后通过高速网络与 OST 直接交换数据对象，数据对象的访问绕开了服务器的中转，能够实现高性能的并行数据传输。目录路径元数据、文件本身数据以及数据对象的独立管理能够实现大规模集群存储系统的高伸缩性。

(2) 设计思想

元数据管理集群采用上述的体系结构基于两个方面的考虑：

一是实现元数据的层次管理。元数据的内容包括形成目录树结构的目录元数据和描述文件各种属性的属性元数据等。对元数据进行管理时，可以把所有的元数据放在一起管理，也可以对元数据分层次管理。所有的元数据在一起管理时，服务器类型单一，总

体结构实现比较简单,但由于各种元数据的性质不一样,访问的频率不同,而且各种元数据内容不相同极易引起元数据的存储管理不一致。如果分层次管理,每个层次的管理比较单一和简单,可以根据各自的特点实现不同的管理策略,保证管理的高效率。元数据中,目录元数据是形成整个目录树结构的中心数据,客户访问文件时,都要访问目录元数据才能访问到文件的属性元数据,当客户只需要访问目录时,不一定访问文件的属性元数据。目录元数据是形成整个目录树结构的中心数据,是可以共同访问的公用数据,CFS 连接到不同的元数据上均能正确进行文件在目录树中的定位。如果目录元数据也进行分布存储,维持数据的一致性将带来大量的通信,这无疑会大大增加系统的开销。因而,可以考虑对元数据进行分类管理,使用不同的服务器来管理不同的元数据,针对不同的元数据特性采用适合该元数据的管理方式。把目录元数据与属性元数据分开,采用集中共享存储集群管理目录元数据,可以保证对目录元数据的高频率访问,同时也便于维护目录数据的一致性。对于文件自身属性元数据,它们具有相同的大小,可以采用统一的存储方式管理,而且元数据之间是独立的,相互之间不进行通信,适合采用独立分布的存储方式。所以,在 OCFS II 系统中分别由 DPIS 集群和 MDS 集群分两个层次管理目录元数据和文件自身元数据。

二是考虑共享存储可能会成为系统的瓶颈。现在的计算机系统中,存储极有可能成为系统的瓶颈。在一个 512 节点的高性能计算集群运行时,每秒可以达到 150,000 个写数据请求^[13],而现在文件服务器中磁盘性能表现最好的 SCSI 硬盘(富士通 MAS3735, 73 GB Ultra320 SCSI, 15000 转/分) IO 性能为 366 IO/sec^[14],考虑系统中对元数据的缓存,假定缓存命中率为 99% (包括 client 端缓存和 DPIS 内存中缓存),在不考虑读请求的情况下至少需要 5 个这样的磁盘组成的阵列并行服务才能提供所需要的磁盘 IO 性能。对于大型集群使用来讲,DPIS 集群使用的共享存储极有可能成为 DPIS 集群性能的瓶颈,将会在很大程度上限制系统的扩展性。为了减轻共享存储的压力,将目录路径元数据和文件自身属性元数据分开管理,将文件自身属性元数据访问的 IO 负载分流到 MDS 集群,可以大大减轻 DPIS 共享存储 IO 服务的压力,减少共享存储成为瓶颈的可能,保证系统高性能和易扩展。

3.3.2 OCFS II 的元数据管理模型

OCFS II 中的元数据管理集群对元数据实现层次管理,它把文件的元数据分为目录路径元数据与文件自身元数据进行管理,这种设计能够避免或大大减少元数据的迁移,提高元数据的访问性能。基于 OCFS II 的体系结构以及系统独特的元数据管理方式,采用如下的元数据管理模型:

①文件 ID (fid)

在 OCFS 设计中引入目录路径索引服务器,能够为系统中所有文件(包括目录文件)分配一个全局统一的文件 ID。目录路径索引服务器采用共享存储的服务器集群,统一管理文件 ID 的分配,保证文件 ID 全局唯一。目录路径索引服务器中的索引项如下:

INDEX=<fid, dir_hash, pid, name, Ac>, 其中, fid 表示全局唯一的文件 ID, 在文件创建时分配, dir_hash 是文件全路径名散列值, pid 是文件的父目录 fid (目录文件 “/” 的父目录为 NULL), name 是文件的文件名即全路径名的最后一个分项名, Ac 表示该文件访问控制属性。路径的访问控制属性的构造同现有的遍历文件路径中的所有目录来确定该文件的访问权限的方法相同, 不同的是, 每一次创建文件时将获得的路径访问控制属性记录在当前索引项中, 在目录下创建新的文件时, 不再需要遍历前面的目录, 可以递归使用当前目录的访问控制属性和新建文件的访问控制属性构建新的访问控制属性。索引项使用双入口的访问, CFS 访问目录路径元数据时以 dir_hash 为索引项, DPIS 定位 MDS 中的元数据时以全局唯一的 fid 为索引项。这种设计使得文件路径的目录名和访问权限可以任意修改, 而 fid 始终保持不变, 所以不会因为目录名和访问权限的修改导致该目录下所有文件的元数据更新, 避免了大量的元数据迁移。特别约定根目录 “/” 的 fid 为 0。文件的访问控制属性只需要一次散列计算就可以从目录路径索引服务器中获取, 不再需要遍历全部目录, 大大提高了访问性能。

②文件的位置信息

文件的位置信息由保存文件的数据对象的 OST 所在的组编号、组内编号以及分配给文件的数据对象 ID 确定。定义文件的位置信息如下: Location=<OSTG_NO, OST_NO, OID>, 其中, OSTG_NO 表示保存文件数据对象所在的 OST 组编号, OST_NO 表示保存文件数据对象的 OST 在组中的编号, OID 表示该 OST 分配给文件的数据对象 ID。

③元数据信息项

定义所有元数据的集合为 D, 每一个文件的元数据信息项 $d \in D$, d 定义如下: Item = (fid, name, Type, Location, Other), 其中, fid 表示文件 ID, name 表示全路径名的最后一个分项名, Type 表示文件类型, Location 表示文件的位置信息, Other 表示其他的元数据信息 (如文件大小, 最近访问时间等), 文件的 fid 在文件创建时获得。

3.3.3 元数据管理集群的运行机制

基于存储的分布式文件系统应该具有以下功能: 能够为应用层提供 POSIX 接口, 允许应用程序对底层的存储系统执行如 Open, Close, Read 和 Write 等的文件操作; 计算节点中有缓存机制, 缓存数据的交换; 对数据存储采用条块化管理; 具有 iSCSI 协议, 直接发送和接受 OSD 上的数据; CFS 可以从根目录下 mount 文件系统, 按照所具有的权限访问相应的目录。

在实现元数据管理集群的系统中, 采用元数据管理集群管理元数据, 通过元数据分配和定位算法实现元数据在集群的服务器之间的分割, 并正确定位元数据所在的服务器。通过数据分割, 元数据按照一定的特征分配到元数据管理集群的服务器上, 或者元数据访问的服务负载分流到集群中的各服务器, 从而实现系统的负载均衡, 保证系统整体服务的高性能和扩展性。

OCFS II 中元数据管理集群由 DPIS 集群和 MDS 集群两个集群组成。两个集群采用类似的元数据分配和定位算法实现元数据的分配和定位,算法思想为:元数据以某特征值进行分配和定位,将元数据分配到某个服务器存储或者将元数据服务负载分配给某个服务器进行处理。在系统中,维持一定数量的桶(bucket,桶是所有数据的一部分数据集合),在元数据和服务负载的分配中起中介作用。元数据和服务负载的分配和定位分为两个阶段进行,首先元数据或服务负载在多个桶中均匀分布,然后桶到服务器均匀映射,从而实现元数据和服务负载全局的均匀分配。具体实现时,系统通过服务器映射算法实现桶到服务器的映射,形成“桶—服务器”显式映射表,保存在系统中。元数据和服务负载分配通过两个步骤实现,第一步,桶分配获得元数据所在的桶号,对元数据关键值 key 执行 hash 算法(桶分配算法)将元数据均匀分配到桶,可以得到元数据所在的桶号;第二步,桶到服务器的映射,根据桶号,查找桶—服务器映射表,确定存储元数据或者处理元数据服务请求的服务器。算法思想如图 3.6 所示。

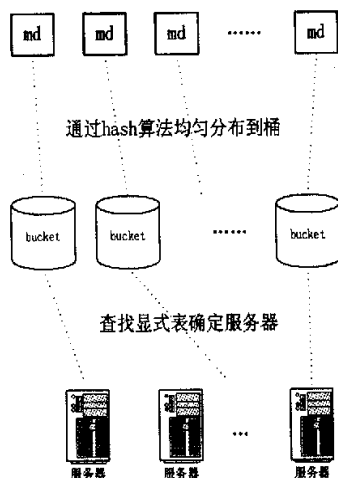


图 3.6 元数据分配和定位算法思想

OCFS II 中维持两个“桶—服务器”的显式映射表,其中,盛装目录路径元数据的桶与 DPIS 服务器的显式映射表保存在 CFS 中,盛装文件自身元数据的桶与 MDS 服务器的显式映射表分别保存在 DPIS 中,处理数据操作请求时,根据相应的 key 值按照桶分配算法确定数据或者服务所在的桶号,然后在查找相应的显式映射表定位服务器,根据定位的服务器进行连接访问。

在 CFS 中保存的目录路径桶号与 DPIS 服务器的映射表中的元素为<bucket_no, dpis>,其中, bucket_no 为服务文件对象目录路径元数据的桶号,以文件对象的目录路径的 hash 值作为 key 值执行桶号分配算法获得; dpis 为相应的 DPIS 服务器信息,由桶分配所在的 DPIS 编号和服务器地址组成,由服务器映射算法确定桶与 DPIS 服务器的对应关系。CFS 挂载文件系统时,从 DPIS 中复制映射表保存在 CFS 的内存中。当 DPIS 服务器集群进行扩展时,集群内部更新映射表,更新完毕后,在 DPIS 的后端存储中保存

显式表, 当有 CFS 要求服务时, 通知其更新它保存的显式映射表。

在 DPIS 中保存的文件自身元数据桶号与 MDS 服务器的显式映射表中的元素为 $\langle \text{bucket_no}, \text{mds} \rangle$, 其中, bucket_no 为保存文件自身元数据的桶号, 以分配给文件的 fid 作为 key 值执行桶分配算法获得; mds 为相应的 MDS 服务器信息, 由桶分配所在的 MDS 的编号和服务器地址组成, 由服务器映射算法确定桶与 MDS 服务器的对应关系。系统启动时, DPIS 从 MDS 获取该映射表并保存, 该表保存在 DPIS 共享的存储中, 每个 DPIS 的内存中有该表的数组形式的备份。当 MDS 服务器集群进行扩展时, 集群内部更新映射表, 更新完毕后, 在每个 MDS 的后端存储中保存显式表, 并通知 DPIS 集群更新它们共享的该显式表。

那么 OCFS II 系统如何实现文件系统的系统调用呢?

用户要使用文件系统访问其中的数据时, 需要 mount 文件系统, 文件系统的注册同原型系统。由于使用 DPIS 集群和 MDS 集群协同管理元数据, 安装文件系统时, 需要读取 DPIS 集群和 MDS 集群的信息, 复制目录路径元数据桶—服务器映射表到 CFS 中。具体流程示意图如图 3.7 和图 3.8。

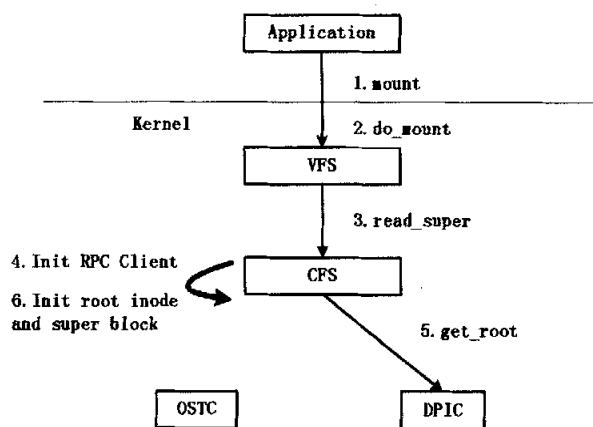


图 3.7 OCFS II 内部 mount 流程

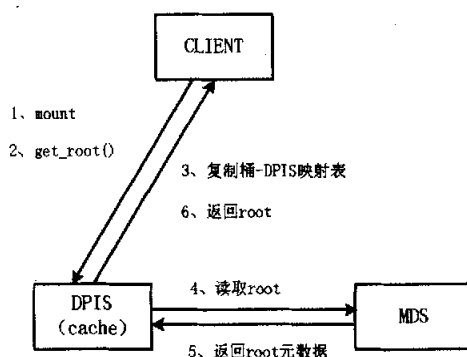


图 3.8 OCFS II mount 时子系统间流程

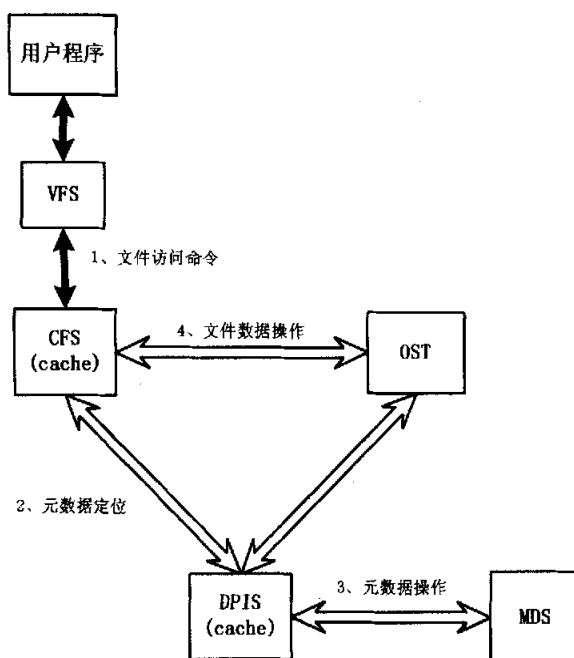


图 3.9 文件操作流程

OCFS II 能够为应用层提供 POSIX 接口，允许应用程序对底层的存储系统执行如 Open, Close, Read 和 Write 等的文件操作。文件操作流程如图 3.9 所示，具体的文件操作类似 OCFS 原型系统中的文件操作，有以下几个步骤：

①发出调用

用户需要进行数据操作时，发出文件系统系统调用，内核调用 VFS 中相应的函数，VFS 把它重定向为 OCFS II 文件系统中相应的函数调用。系统调用的入口参数中包含欲打开的文件或者需要操作的文件的文件描述符，传递给 OCFS II 相应的函数。CFS 通过与 VFS 的接口收到该系统调用，开始在 OCFS II 系统内部完成系统调用的工作。

②访问元数据

读取元数据的过程包括以下步骤：元数据定位，元数据操作。

首先，对目录路径执行目录路径的 hash 算法，得到目录路径的 hash 值，再以此作为 key 值，执行元数据分配和定位算法（对 key 执行 hash，分配到桶，再使用桶号查找“桶—DPIS 服务器”显式映射表），确定需要访问（即提供相应元数据服务）的 DPIS 服务器，发送请求到相应的 DPIS 服务器。

在 OCFS II 中，DPIS 是整个系统中 CFS 访问元数据的接口，DPIS 具有缓存机制，可以缓存 CFS 访问的元数据。DPIS 接受到请求，首先查找高速缓存，若缓存有相应的元数据则向用户返回相关信息；若缓存中没有相应的元数据，则访问目录路径元数据的后端存储，得到请求文件的 fid 值，以 fid 值作为 key 值，执行元数据分配和定位算法（对 key 执行 hash 算法，分配到桶，再使用桶号查找“桶—MDS 服务器”显式映射表），

确定需要访问的 MDS 服务器，发送请求到 MDS 服务器；

MDS 接收到请求，查找高速缓存中是否有需要的元数据，若有则返回；否则，访问后端存储器，读取相应的数据。然后，元数据按照相反的路径返回给 CFS。

③文件 I/O

CFS 取得文件的元数据后，根据文件的位置信息，访问 OST；OST 验证用户的访问权限。然后，CFS 与 OST 之间进行文件数据的交换。

本章主要研究对象存储文件系统中的元数据管理部分，分析了与实现集群管理元数据相关的数据存储和数据分割等方面的问题，提出了一个元数据管理集群模型，改进原型文件系统 OCFS 成为 OCFS II。在元数据管理集群模型中，采用目录路径索引服务器 DPIS 集群和元数据服务器 DMS 集群对元数据进行分层次管理，元数据管理集群可以保证元数据服务的高性能和可扩展性，同时能保证元数据访问能够一次定位，并且名空间信息修改时不需要迁移数据。元数据管理集群要保证元数据服务的高性能，负载均衡是必须研究和解决的重要问题，下一章在将着重研究集群系统中负载均衡实现的相关知识，针对 OCFS II 元数据管理集群中 DPIS 集群和 DMS 集群的不同特点，分别实现不同的负载均衡方案，保证元数据管理集群的高性能。

第四章 OCFS II 元数据管理集群负载均衡的实现方案

§ 4.1 负载均衡概述

集群系统的负载均衡是指计算机集群中的各个节点尽可能平均地分摊处理集群系统负载,每个节点都能有效地参与工作,这样可以充分利用系统资源,充分发挥集群系统的整体性能。负载均衡是集群系统必须具有的一项功能,集群只有实现了负载均衡才是一个适用的系统,如果不能实现负载均衡,集群系统的优势将难以体现。负载均衡是集群系统的资源管理模块,它主要合理和透明地在集群各节点之间重新分配系统负载,以达到系统的综合性能最优。根据不同的分类标准,负载均衡有多种不同的分类。但总的来说,负载均衡大致可以分为动态和静态两类^[34,35,36]。

4.1.1 负载均衡分类

按照不同的分类方法,负载均衡有多种分类,主要有以下几种:

①局部和全局

局部负载均衡指对集群内一部分节点执行负载均衡策略,进行服务负载的分配和调度;而全局负载均衡是在整个集群范围内进行服务负载的分配和调度,实现集群全局范围内的负载均衡。

②静态和动态

按照决策是否与系统的负载状态相关,负载均衡可以分为静态均衡和动态均衡。静态负载均衡中,服务负载的分配是按照预先制定的算法进行分配,平衡决策与系统状态无关,而动态负载均衡是系统在运行过程中进行的。静态方法又叫做确定性调度,而动态方法叫做负载均衡。

③最优和次优

如果根据标准,比如最小执行时间和最大系统输出,可以取得最优负载分配,那么可以认为这种负载均衡方法是最优的。一般的,负载均衡问题是 NP 完全问题。某些情况下,次优方案也是可以接受的。

④近似和启发式(在次优模型中)

在近似方法中,负载均衡算法仅搜索一个解空间的子集,当寻找到一个好的解时,终止执行。在启发方式中,调度算法使用某些特殊参数,能够近似地对真实系统建模。

⑤集中式和分散式(在动态模型)

在集中式算法中,设置了一台主机专门负责系统的负载状况的收集和迁移决策。而在分散式算法中,这些工作分配给不同的节点。

⑥协作和非协作的(对分散式)

动态负载均衡机制可以是协作的,多个决策节点间有协同操作,也可以是非协作的,由单个节点独立作出决策。

静态和动态的分类是负载均衡的基本的分类。

4.1.2 负载均衡实现方法和层次

(1) 负载均衡实现方法

服务器集群系统的负载均衡可以用硬件或者软件实现^[37]。

软件负载均衡解决方案是指在一台或多台服务器相应的操作系统上安装一个或多个附加软件来实现负载均衡,如 DNS Load Balance, CheckPoint Firewall-1 ConnectControl 等,它的优点是基于特定环境,配置简单,使用灵活,成本低廉,可以满足一般的负载均衡需求。

软件解决方案缺点也较多,因为每台服务器上安装额外的软件运行会消耗系统不定量的资源,越是功能强大的模块,消耗得越多,所以当连接请求特别大的时候,软件本身会成为服务器工作成败的一个关键;软件可扩展性并不是很好,受到操作系统的限制;由于操作系统本身的 Bug,往往会引起安全问题。

硬件负载均衡解决方案是直接服务器和外部网络间安装负载均衡设备,这种设备我们通常称之为负载均衡器。由于采用专门的设备完成专门任务,独立于操作系统,整体性能得到很大提高,加上多样化的负载均衡策略,智能化的流量管理,可达到最佳的负载均衡需求。

一般而言,硬件负载均衡在功能、性能上优于软件方式,不过成本昂贵。

(2) 负载均衡实现层次

分析服务器集群的使用特点,以客户端应用为起点纵向分析,到服务器集群,这是一个层次的结构,两个层次之间还可能有中间的层次。对于多层次的结构可以在不同的层次上实现服务器集群的负载均衡。根据负载均衡实现的层次,可以把负载均衡技术的实现分为客户端负载均衡技术、应用服务器技术、高层协议交换、网络接入协议交换等几种方式。针对网络上负载过重的不同瓶颈所在,从网络的不同层次入手,可以采用相应的负载均衡技术来解决现有问题。

4.1.3 静态负载均衡

静态负载均衡技术往往伴随着系统设计或系统初始安装的过程,具有确定的负载均衡规律,这些规律在系统投运之后,一般很少发生改变,或者改变甚微。静态负载均衡算法根据系统的先验知识作出决策,在运行前对负载进行分配。静态调度算法的目标是调度一个任务集合,使任务集中的任务能够快速高效地执行。

设计调度策略时有三个主要因素:节点互连、任务划分(粒度决策)和任务分配。

节点间的网络连接拓扑可以分为静态和动态。静态网络由点到点直接连接而成,并

在执行过程中不能改变。动态网络由交换信道实现,能够根据用户程序动态配置以满足通信需求。任务划分的粒度往往要考虑任务的并行性,任务划分的粒度太大,会降低任务的并行性,粒度太小,会增加额外的调度和切换开销等。任务分配就是向服务器集群中的节点分配任务颗粒,分配的策略非常重要。

静态负载均衡是以一种一劳永逸的方式把任务分配给处理器,而且要求预先知道任务的执行状况,这是静态负载均衡的限制。静态负载均衡中往往对例如任务执行时间或者通信延迟等运行过程中的参数给予一定的假设,但这些参数中可能包含不可预测的因素,在任务执行过程中可能超出假设范围,导致静态负载均衡方法达不到预期的效果。

4.1.4 动态负载均衡

动态负载均衡发生在系统运行过程中。由于网络通信、I/O、应用和数据处理的多变、主机服务器本身性能的变化、以及其它系统运行异常因素的影响,系统各类负载随时都可能发生大的改变,导致集群系统节点之间的负载不均衡,此时,针对负载的这些变化而作出的均衡策略,称为动态均衡技术。动态负载均衡技术很少假设运行参数的先验信息,它在运行过程中动态收集任务执行的信息并根据任务执行信息重新分配任务,能够更好地动态使用系统中的所有资源,实现提高系统性能的目标。

动态负载均衡算法按照集中程度可以分为集中式、完全分布式、层次性或它们中的一些结合算法。

在动态集中式负载均衡的系统中,全局负载信息收集在一个节点上,任何节点的负载变化信息都传给这个节点,负载均衡所有决定由它作出。这种方式的好处在于能够以较少的开销收集全局信息,挑选出最佳节点执行任务,并且可以扩充到较大的网络计算系统。

在完全分布式负载均衡系统中,每个节点保存相邻节点或系统中部分节点的负载信息,相互合作做出各自的负载分配。这种策略实现较简单,并经过一段时间后,可以选择到较合适的节点执行任务。缺点是不能获得最佳节点分配负载,很难扩展到成千上万个节点的网络计算系统。

层次性方法是根据集中式、分散式方法的优缺点结合而成的一种负载均衡算法。它将系统分成层次性的子系统,选择不同层次上特殊的节点作为负载均衡决策的控制节点,以分散其集中方式控制整个系统的负载均衡。它和网络拓扑结构有很强的相关性。

§ 4.2 OCFS II 元数据管理集群负载均衡方案的设计

4.2.1 元数据管理集群负载均衡目标

元数据管理集群负载均衡方案的设计,主要考虑以下几方面的问题:

①高性能:集群的服务负载按集群中各服务器性能均衡地分配到各服务器,服务器

集群对 client 元数据请求负载进行并行处理, 大大增强处理 clients 并发元数据请求的能力, 减少 client 的等待时间, 提高服务速度, 整个集群对 client 表现出高性能。

②可扩展性: 文件系统是不断膨胀的, 特别是分布式文件系统, 膨胀的特征更明显, 作为提供核心数据服务的元数据管理集群必须是能够扩展的。随着总服务负载的增加, 可以根据需要加入新的服务器到集群中, 集群能够自动负载均衡, 将部分服务负载分配到新加入的服务器上。

③灵活性: 负载均衡方案应能灵活地提供不同的应用需求, 满足应用需求的不断变化。集群内部的调整要尽量不影响对外服务。负载均衡过程中需要进行迁移数据时, 迁移数据量要小, 时间要尽量短。

④可靠性: 负载均衡方案应能为服务器集群提供完全的容错性和高可用性, 不能因为某个负载均衡设备失效而导致整个负载均衡方案的失败。

⑤易管理性: 有灵活、直观和安全的管理方式, 便于安装、配置、维护和监控, 提高工作效率, 最好能自动动态处理各种情况, 减少人工干预。

4.2.1 OCFS II 的元数据管理集群负载均衡分析

元数据管理集群作为大型文件系统的元数据管理单元, 它的服务负载有自己的特点, 即这些负载不是大量计算的计算负载, 而是访问元数据的数据访问负载。因而, 可以通过元数据存储的分布, 自然进行服务负载的分流, 实现负载均衡。实现时, 通过采用数据分配算法, 把元数据分布到集群中服务器节点上进行存储。访问元数据时, 通过相同的算法定位元数据所在的服务器, 可以将该元数据请求负载分配给相应的服务器节点。

对象存储文件系统中, 一般包括对象存储目标、元数据服务器和客户端三个部分。为了支持用户有效使用文件系统, 对象存储文件系统的客户端通常与计算节点结合在一起, 用户程序通过文件系统客户端进行文件系统调用和操作, 文件系统的客户端是用户与文件系统的桥梁和入口。因而, 可以在客户端分析用户数据请求的特征, 将不同的元数据请求发送给不同的元数据服务器进行处理, 实现元数据服务负载在各服务器节点上的分配, 达到负载均衡的目的。

在 OCFS II 中, 元数据分为目录路径元数据和文件自身元数据, 分别由不同的集群管理, 每一个服务器集群都必须实现负载均衡。因而, OCFS II 中元数据管理集群的负载均衡包括两个部分, 目录路径索引服务器集群的负载均衡和元数据服务器集群的负载均衡, 并且, 两个部分的负载均衡是相互独立的。目录路径索引服务器集群是采用共享存储的集群系统, 其中的每个服务器节点都能够完成相同的功能; 元数据服务器集群是每个服务器节点采用独立存储的集群系统, 每个服务器节点能够处理的元数据请求与它的存储器中存储的数据有关系。

将文件系统里的所有文件作为普通文件看待, 每个文件都有抽象的元数据。由于元数据是文件信息相同的抽象, 大小基本相同, 忽略网络通信延迟和服务器内部数据访问

时 Cache 是否命中带来的差异, 整体来说, 每个元数据请求的处理都是相似的, 具有相似的处理过程和处理复杂度。所以, 可以把每个元数据请求看成天然的元数据服务负载粒度。同时也由于元数据是文件信息相同的抽象, 大小基本相同, 元数据服务负载量可以转化为服务器上存储的数据量来表示。

总结上述分析的特点, 元数据管理的两个集群分别采用各自的方法实现负载均衡。对于 DPIS 集群, 采用基于客户端的静态负载分配和动态反馈负载重分配相结合的方法, MDS 集群采用元数据在集群节点上的分布存储而实现负载分流。静态负载均衡使用静态的 hash 算法实现负载的分配, 以负载的特征值为 key 值, 执行 hash 算法, 将负载分配给不同的服务器节点。

4.2.2 目录路径索引服务器 DPIS 集群负载均衡

假定目录路径索引服务器 DPIS 集群中有一组服务器 $S_{dpis}=\{S_0, S_1, \dots, S_{n-1}\}$, $PF(S_i)$ 表示服务器 S_i 的 cpu 主频, $Mem(S_i)$ 表示服务器 S_i 的主存大小。整个集群采用基于客户端的静态负载分配和根据服务器的动态反馈重分配相结合的方法。具体如下:

(1) DPIS 静态负载均衡 hash 算法

负载的静态分配算法基于这样的前提和假设: 文件系统中数据的访问具有随机性, 除了某些普遍使用和某些关于“热点”数据之外, 大多数文件的访问概率是相同的; 每个请求处理的复杂度是基本相同的。用户进行文件系统调用时, 调用参数中包括其访问和操作数据对象的全路径名。客户端计算该全路径名的 hash 值, 以该 hash 值作为 key 值采用 hash 算法将元数据请求分配给各服务器。DPIS 集群中, 各服务器可能是异构的, 则在负载分配时不仅要考虑服务器的负载量, 而且也必须要考虑到服务器的处理能力。这样做的好处是能让处理能力高的服务器承担更多的任务。在这里, 以权值反映服务器的处理能力。

在 DPIS 集群中, 服务器的处理能力主要体现为处理速度, 对于节点 S_i 的处理能力 $C(S_i)$, 主要从这几个指标考虑: CPU 处理能力 $C(C_i)$ 、内存容量 $C(M_i)$ 。这几个指标不能明确表示该服务器的处理能力, 所以, 我们在这里引入了参数 k , 用以说明服务对各项指标的依赖程度, 可以用一个函数 (公式 1) 来进行转换得出服务器的综合处理能力 $C(S_i)$:

$$C(S_i) = [k_1 \quad k_2] \cdot \begin{bmatrix} C(C_i) \\ C(M_i) \end{bmatrix}, i=0,1,\dots,n-1, \sum k=1 \quad (\text{公式 1})$$

选择合适的处理能力为基数 C_{base} , 通常可以选取集群中所有服务器的综合处理能力 $C(S_i)$ 的最大公约数为 C_{base} , 可以计算各服务器的权值:

$$W(S_i) = C(S_i) / C_{base} \quad (\text{公式 2})$$

静态算法中的桶数量 countb 为 DPIS 集群中所有节点权值和, 即

$$\text{countb} = \sum W(S_i) \quad (\text{公式 3})$$

静态负载均衡方案如下: DPIS 集群系统中, 维持一定数量的桶 (桶数量为 countb), 元数据请求按照全路径名的 hash 值 hash_dir 均匀分布到桶中, 每个桶通过“桶—服务器”映射算法建立起一个与服务器的映射, 这样元数据请求通过两级分配分布到服务器上。实现过程如下, 当 client 有元数据请求时, 首先以全路径名 hash 算法计算全路径名的 hash 值 hash_dir, 然后以 hash_dir 作为 key 值将 client 请求均匀分配到桶中, 返回桶号; 接着以元数据分配和定位算法查找显式表选定进行服务的服务器。

DPIS 集群桶分配算法, 根据元数据全路径名 hash 值 hash_dir 将元数据服务负载分配到相应的桶中。算法如图 4.1 所示。

```
assign_bucket(hash_dir, countb){
    key=hash_dir;
    nb=key mod countb;
    Return nb;
}
```

图 4.1 DPIS 集群桶分配算法

```
/*桶定义*/
struct bucket{
    int character; /*桶特征值*/
    int server_no; /*桶所在服务器编号*/
};
struct bucket buckets[countb];

/*服务器定义*/
struct dpissever{
    int number; /*DPIS服务器编号*/
    int pf; /*服务器主频*/
    int mem; /*服务器内存*/
    _u32 addr; /*服务器地址*/
    _u16 port; /*服务器端口*/
    int mormal; /*工作状态标志: 0,非正常工作; 1,正常工作*/
};
struct dpissever dpises[n];

/*桶服务器映射*/
assign_server()
{
    struct bucket buckets[countb];
    for(i=0;i<countb;i++){
        选出权值较大的服务器;
        将桶分配给该服务器;
        该服务器权值减1;
    }
}
```

图 4.2 DPIS 集群桶—服务器映射算法

“桶—服务器”映射算法如图 4.2 所示，采用加权轮叫算法，将桶映射到服务器，形成“桶—服务器”显式映射表。系统的客户端执行如图 4.3 所示的元数据分配和定位算法，选取提供元数据服务的 DPIS 服务器，将元数据服务负载分配给服务器。

运行过程中，客户端接收 VFS 传递的文件系统指令，CFS 调用 Load balancing 模块。Load balancing 模块对文件系统调用的数据对象元数据的全路径名执行桶分配算法，计算全路径名 hash 值，并根据 hash 值计算桶号；然后查找“桶—服务器”映射表，获得服务器号，定位元数据分配的服务器，从相关的服务器信息表中读出服务器信息，建立连接，发送请求，由相应的 DPIS 服务器执行元数据操作。

```
int select_dpisserver(hash_dir,buckets,countb)
{
    nb=assign_bucket(hash_dir,countb);
    server_no=buckets[nb].server_no;
    return server_no;
}
```

图 4.3 DPIS 集群元数据分配和定位算法

(2) 动态反馈负载重分配

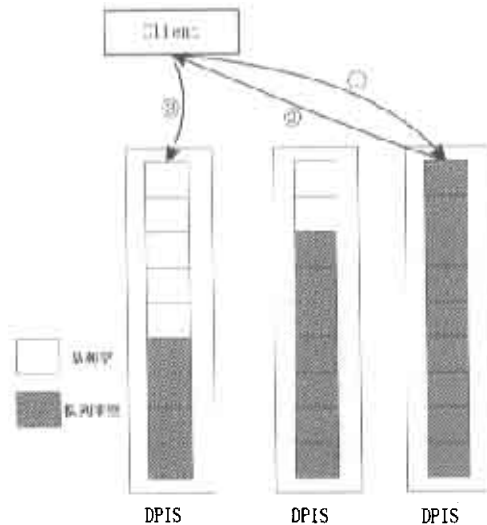


图 4.4 DPIS 动态反馈负载均衡思想

DPIS 集群中采用客户端静态分发负载没有考虑元数据请求处理过程中网络通信延迟和服务内部元数据请求处理时间的差异，因而，只是静态的分配可能会造成负载不均衡。动态反馈重分配^[38]方法为：在 DPIS 中设置请求处理队列，用以缓存客户端的元数据请求，队列的每个位置对应元数据处理线程。每个服务器监视自身的负载状况和集群其他服务器的负载状况，当该服务器的负载超过一定的阈值时，当有新的元数据服务

要求时,向客户端反馈该服务器忙信息和集群中的轻载服务器信息,客户端根据这些信息将元数据请求转发给另外的服务器,保证集群中的服务器不会出现超载运行的情况,实现负载均衡。

服务器负载信息动态查询获取服务节点上的各项信息,如 CPU 占用率 $L(C_i)$ 、内存占用率 $L(M_i)$ 、网络带宽占用率 $L(N_i)$ 、进程数量占用率 $L(P_i)$ 。这些信息并不能具体表示某一个节点的负载状况,因此,引入了参数 r ,用以说明该类服务对服务器服务性能的不同影响程度,采用函数(公式 4)将这些指标转换,得到该服务器的负载量 $load(S_i)$ 。可以通过试验调整参数 r ,使公式 4 转换的结果能正确反应服务器的负载。对于 DPIS 服务器,我们可以这样调整参数 r , $r=\{0.3,0.3,0.15,0.25\}$,用以表示 CPU 处理能力、内存容量和网络吞吐能力对于元数据服务的重要性。

$$load(S_i) = [r_1 \quad r_2 \quad r_3 \quad r_4] \cdot \begin{bmatrix} L(C_i) \\ L(M_i) \\ L(N_i) \\ L(P_i) \end{bmatrix}, i=0,1,\dots,n-1, \sum r=1 \quad (\text{公式 4})$$

参数 $load(S_i)$ 表明该服务器 S_i 的负载状况,设定服务器的状态阈值参数 $minload$ 、 $maxload$ 和 $overload$,分别代表系统负载状态为轻载状态、重载状态和超载状态。服务器中负载量 $load \leq minload$ 时,该服务器处于轻载状态;当 $minload < load \leq maxload$ 时,服务器处于适度负载状态;当 $load > maxload$ 时,服务器处于重载状态;当 $load > overload$ 时,服务器超载,不能处理新的元数据请求,如果有新的元数据请求必须由其它的服务器来处理。服务器根据一定的策略,在适当的负载状况下启动负载状况收集程序,并根据收集的信息作出负载均衡决策,找出合适的轻载服务器,以备服务器超载时转移负载。当服务器负载出现超载 $load \geq overload$ 时,启动服务器超载程序,当有新的元数据请求到达时,直接返回轻载服务器号给客户端,客户端根据服务器返回的轻载服务器号重新发送元数据请求。

请求队列长度的设置应该适中,避免长度太长超越服务器的处理能力,造成服务器不能产生足够的处理线程;也要避免队列长度太短,造成服务器处理能力未达到饱和,服务器线程太少而造成元数据处理请求的阻塞。服务器处理元数据请求时,由于请求的随机性以及请求是否在内存中有缓存数据的不同,每条请求的处理时间是不尽相同的,某些请求处理时间可能很长,会导致后面的请求必须等待前面的请求的处理,出现请求处理效率低下的情况。请求队列长度的设置应保证服务器在正常负载情况下不出现阻塞现象,后面的请求不必等待前面请求处理完毕,保证高效率、短时间处理请求。

下面分析队列请求的处理时间给出一个确定请求队列长度的度量。

在 OCFS II 系统中,处理一条元数据请求的流程如图 4.5 所示。

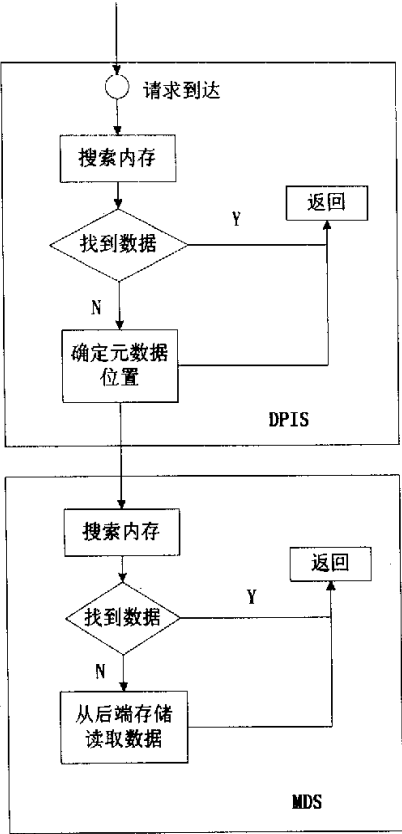


图 4.5 元数据请求处理流程

假设 DPIS 内存搜索时间为 T1，存储器访问目录时间为 T2，MDS 内存搜索时间为 T3，存储器访问元数据时间为 T4，DPIS 与 MDS 之间网络传输时间为 TT。那么，在最坏情况的情况下，元数据服务请求的处理时间 Tw 为：

$$Tw=T1+T2+T3+T4+2\times TT$$
 (公式 5)

数据在内存中的命中率除了与内存大小、内存置换策略相关外，还与访问数据的局部性有关。元数据访问请求按照全路径的 hash 映射到 DPIS 服务器上，可以解决同一目录下同时创建大量文件的局部热点问题，但也使数据的访问失去空间局部性，元数据的访问只存在时间局部性，即最近访问过的数据可能在内存中命中。假设在 DPIS 中内存中的访问元数据命中率为 P1，在 MDS 中内存中的访问元数据命中率为 P2，则元数据请求的平均处理时间 Tm 为：

$$Tm=T1\times P1+(T1+T2+T3\times P2+(T3+T4)\times (1-P2)+2\times TT)\times (1-P1)$$
 (公式 6)

当元数据在 DPIS 的内存中命中时，该请求处理时间最短为 T1。
在处理元数据请求时，处理时间有三种情况，它们的概率如表 4.1 所示。

表 4.1 元数据请求处理时间概率

缓存命中情况	DPIS 内存命中	MDS 内存命中	访问 MDS 后端
处理时间	T1	T1+T2+T3+2TT	T1+T2+T3+T4+2TT
概率	P1	(1-P1)×P2	(1-P1)×(1-P2)

元数据请求的处理不致阻塞,那么元数据请求队列长度 L_q 满足

$$L_q \geq T_w/T_m \quad (\text{公式 7})$$

同时为保证服务器的正常运行,请求队列长度不能大于服务器能够创建线程的一定比例。

动态负载均衡的实现通常包括三个步骤:负载信息收集、均衡决策和任务迁移。

目录路径索引服务器集群的动态负载均衡采用发送者启动算法,由重载节点做均衡决策将重载服务转移到轻载服务器。考虑目录路径索引服务器集群系统的特点,系统中负载状态收集可以根据收集时间的长短分为以下几种,周期收集方法,重载启动收集,超载启动收集。所谓周期收集方法,是指系统中每个节点都周期性收集集群中其它节点的负载信息,系统中每个节点都能随时了解系统的负载状况;重载启动收集是系统在正常使用中系统节点不收集系统负载信息,整个系统按照静态负载均衡的策略运行,当某个节点负载量超过一定阈值,出现重载时,启动系统负载状态收集,重载节点收集其它节点负载状况信息,便于选取轻载节点,在该重载节点负载超载时进行负载的迁移;超载启动收集是系统中某个节点出现超载时才启动系统负载状态收集,收集其它节点负载状况信息,作出负载转移决策。三种方法负载状态收集带来的开销逐渐减少,但出现超载请求等待服务的可能性增大。其中,重载启动收集方法的负载信息收集代价比较小,也能及时收集信息,因而在本系统中采用重载启动收集策略。

均衡决策根据集群负载状况,确定可以接受负载转移的轻载服务器。轻载服务器的选择可以采取多种策略,如邻居节点算法、首次轻载节点算法、全局最轻负载节点算法等。邻居节点算法,重载服务器不必收集其它服务器节点的负载状态信息,直接选取其邻居节点。首次轻载节点算法,在选取请求重发的节点时,选取按收集策略和搜索顺序出现的第一个轻载节点。全局最轻负载节点算法,收集负载的节点收集系统中所有节点负载信息,根据所有的信息选取系统中负载最轻的节点。

三种轻载节点选取的方法依顺序能够得到更优的效果,但代价逐渐增加。假定系统中 n 个服务器节点,邻居节点算法,可以直接选定元数据请求重发的服务器,不会增加重载服务器收集系统负载信息的开销,不管系统组成如何,一次决策的搜索代价为 0;但固定选择邻居节点有一定的盲目性,可能出现某个请求遇到多次繁忙的情况。首次轻载节点算法不需要搜索系统所有节点的负载信息,只需要搜索到系统中第一个轻载节点即可。当系统中有 n 个服务器节点时,决策的平均搜索代价为 $n/2$ 。其搜索出的节点是轻载节点或系统中负载量最小的节点,负载迁移是发生“抖动”的可能性比较小。全局

最轻负载节点算法需要搜索服务器集群中的所有节点,能够选取出集群中负载最低的服务器节点接收过载负载,但搜索代价最高,搜索代价恒定为 n 。

由于负载是动态变化的,请求重发处理与轻载节点的选取之间有一定的时间间隔,请求重发处理时的负载情况与轻载节点选取决策时的情况不完全一致,轻载节点选取决策时的最优效果在请求处理时不一定是最优结果。因此,为尽量使请求重发处理时的负载状况与做均衡决策时一致,需要减小负载状况信息收集的时间间隔,但这样会加大额外的负载状况收集负担。另外,由于均衡决策时的状况与请求重发时不完全一致,即决策均衡的结果总不是完全精确的,因而需要根据系统大小和一次均衡决策的复杂度对均衡决策策略付出的代价作出权衡。

DPIS 集群的动态负载均衡由重载服务器节点和客户端协同工作实现重载服务的转移,图 4.6 为服务器与 client 协同工作的均衡决策和负载转移算法。

```
server端负载监视程序
monitor()
{
    while(1){
        检查本节点负载状态load;
        if(load<maxload)正常服务;
        if(load>=maxload){
            定期收集系统其它节点负载信息;
            根据负载信息以一定的策略选取轻载节点;
        }
        if(load>overload)
            启动动态反馈模块处理client请求;
            向client返回服务器状态忙和轻载节点信息;
        sleep(interval);
    }
}

client端程序
if(请求正常处理)请求结束;
if(服务器忙)重发请求到轻载服务器;
```

图 4.6 DPIS 集群均衡决策和负载转移算法

DPIS 的负载均衡方案是静态负载均衡与动态负载均衡相结合的方案。静态 hash 负载均衡根据元数据特征将元数据请求均匀分配到各个服务器,达到负载均衡的目的。需要一个前提假设,就是所有元数据请求的概率都是相等的。总体情况下这个假设是满足的,从整个数据访问的全过程来看,数据基本具有相同的访问率,即每个元数据请求发生概率相同;数据的分配算法是一个随机的过程,每个服务器上分配大量的元数据请求,对于数量相同的大量元数据请求的组合更满足随机的分布。因而,总体上说,在大多数情况下,通过 hash 算法将元数据请求均匀分配给集群中的各服务器处理,可以保证系统是负载均衡的。

但是,从局部来讲,在很短的一段时间内,对某些数据或者某个数据的访问可能出

现峰值,特别是可能出现数据访问的“热点(hot spot)”问题,导致局部的数据访问不均衡,使某个服务器可能出现过载的情况。出现“热点”问题可能有这样两种情况:一是在某个目录中同时创建大量文件;二是同时对某个文件的大量的读操作。

第一种情况是这样的,某个目录是大量用户同时访问的对象,由于文件系统一致性的要求,在锁语义的保证下,不可能出现多个用户同时写某个文件的情况,但可能出现大量用户到同一目录下创建文件,这样整个目录内的元数据就会成为多用户访问的“热点”数据。对于我们的系统设计来说,由于各文件的文件名是不相同的,通过对目录路径执行 hash 算法后将这些不同文件的元数据分布到不同的服务器中,对它们的访问负载仍然是分流到多个服务器中的。所以,虽然这些数据可能是多个用户同时访问的热点数据,由于对路径执行 hash 算法而避免了这样的“热点”问题的出现。

第二种情况可能出现在某个文件特别重要,或者说某个时候大量的用户都对该文件数据感兴趣,他们都同时去读取该文件数据,这样就造成对该文件元数据的大量并发访问,对该文件元数据的大量并发访问不可避免会出现“热点”问题。对于这样的情况,hash 算法本身就无法起作用了,可能出现某些服务器过载。采用的动态反馈重分配方法就是解决这个问题,将过载的元数据请求重新分配到负载比较轻的服务器上。

DPIS 采用的是共享存储的集群系统,当整个系统很大时,DPIS 集群也需要较多的节点才能完成元数据服务。前面已经讨论过,当 DPIS 集群内节点太多时,共享存储可能成为瓶颈限制系统的扩展、影响系统性能,可以采用目录子树分割法分割文件系统目录树,将根文件系统的某个或者若干子目录分离到一个新的集群上进行处理。系统扩展后,需要重建桶—服务器映射表,并更新 client 端缓存的映射表。

4.2.3 元数据服务器 MDS 集群负载均衡

元数据服务器 MDS 集群,采用独立存储的服务器集群,每个服务器节点由于存储的数据不同能完成不同的元数据服务功能。通过合适的元数据分配和定位算法将元数据分布到 MDS 集群中的各服务器上存储,能够使元数据服务请求随元数据的分布而分流,从而实现负载均衡^[18]。这有一个前提假设,就是所有的数据都具有相同的访问率,它们被访问的概率都是相等的。这个假设总体情况下是满足的,从整个数据访问的全过程来看,数据基本具有相同的访问率;而且,所有数据在各服务器中的分布是随机的,数据分布前不知道数据的内容,数据的分配算法也是一个随机的过程,在每个服务器上都有大量的数据,对于大量的数量相同的数据的组合更满足随机的分布。因而,总体上讲,大多数情况下,通过 hash 算法分布数据,数据的访问负载随之分流,整个系统是负载均衡的。

假设集群中有一组服务器 $S_{mds}=\{S_0, S_1, \dots, S_{n-1}\}$,其中包含 n 个服务器, $M(S_i)$ 表示服务器 S_i 的磁盘空间的大小,以一定的容量 C_b (如 100M) 为桶大小,服务器 S_i 中的桶数量 $num_b_all(S_i)=M(S_i)/C_b$ 。元数据分配到服务器上是以桶为粒度的, C_b 的大小可以根据情况来选择,总容量相同的情况下,当 C_b 比较大时,桶数量比较少,形成

的显式表结构比较小,易于存储,但粒度太粗对服务器负载完全均衡有一定的影响;当 C_b 比较小时,元数据在服务器上分配粒度小,均衡性好,但形成的显式表结构大,存储需要较大空间。

元数据在元数据服务器集群中的分配和定位采用动态 hash 算法^[19,20,21],以文件分配的 fid 作为 key 值将元数据均匀分配到系统中的若干个桶中,系统中的桶数量随着数据量的增加而增加,其值大小与分裂级别 sl 有关,为 $\text{countb}=2^{sl}$,桶编号为 $(0, 1, \dots, 2^{sl-1})$ 。元数据分配和定位算法确定元数据存储的 MDS 服务器,以便进行元数据存储的访问,并将元数据访问请求分流。元数据分配和定位算法通过服务器映射算法(见图 4.7)建立所在分裂级别的桶—服务器显式映射表,对元数据分配和定位时,通过 MDS 集群桶分配算法(见图 4.8)将文件元数据均匀分配到桶中,根据元数据所在的桶号查找显式表确定服务器。

```

assign_server(buckets, countb, mdses, n){
    if(初始化){
        for(i=0; i<countb; i++){
            选出 num_b_unused 最大的服务器 Sj;
            桶号为 i 的桶映射到服务器 Sj 上;
            服务器 Sj 的 num_b_unused 减 1;
        }
    }
    else { /* 扩展后桶分裂更新 */
        for(i= countb/2; i<countb; i++){
            选出 num_b_unused 最大的服务器 Sj;
            桶号为 i 的桶映射到服务器 Sj 上;
            服务器 Sj 的 num_b_unused 减 1;
        }
    }
}

```

图 4.7 MDS 集群服务器映射算法

```

int assign_bucket(fid, sl)
{
    key=fid;
    countb=2^sl;
    nb=key mod countb;
    return nb;
}

```

图 4.8 MDS 集群桶分配算法

服务器映射算法采用加权轮叫算法,将桶映射到服务器,形成显式的“桶—服务器”映射表。算法中,服务器的权值为每个服务器中还能容纳的即服务器中还未使用的桶数 num_b_unused 。系统中,用以下数据结构(图 4.9)表示元数据服务器信息和桶。

```

/*MDS服务器定义*/
struct mdssever{
    int number; /*MDS服务器编号*/
    _u32 addr; /*服务器地址*/
    _u16 port; /*服务器端口*/
    int num_b_all; /*服务器中总的桶数量*/
    int num_b_unused; /*服务器中未使用桶数量*/
};

/*MDS集群中桶定义*/
struct bucket{
    int character; /*桶特征值*/
    int server_no; /*桶所在服务器编号*/
};

```

图 4.9 MDS 集群中服务器及桶定义

MDS 集群桶分配算法中, fid 为访问元数据的文件 ID, sl 为系统分裂级别, key 为执行 hash 算法的关键值, nb 为文件元数据所在的桶号。

```

int select_mdsserver(fid,sl,buckets)
{
    nb=assign_bucket(fid,sl); /*桶分配
    server_no=buckets[nb].server_no; //查找桶—服务器映射表
    return server_no;
}

```

图 4.10 元数据分配和定位算法

元数据分配和定位算法是可扩展的。算法通过桶分裂来实现系统的扩展,具体实现采用这样的方式:设置桶容量的阈值(阈值为桶最大容量的一定百分比,到达阈值后给系统一段准备扩展服务器容量的时间,保证系统分裂前系统桶中的数据不致于溢出)控制算法的扩展,当桶容量达到阈值时,桶分裂成一对伙伴桶,原桶中的数据在伙伴桶中进行分配。由于 fid 的分配是连续的,按照上面的桶分配算法,所有的桶几乎同时达到阈值。当某个服务器的某个桶容量最先达到阈值时,由该服务器通知 0 号服务器启动 MDS 集群扩展算法。扩展算法首先检查服务器中未使用容量是否满足扩展要求,即服务器中未使用的容量是否大于等于服务器中已使用容量,若是,则启动桶分裂算法;若不是,则需要按照要求扩展服务器,新扩展的服务器的容量加上原有服务器中未使用的容量应大于已使用的容量。

扩展后,DPIS 集群更新保存 MDS 集群信息和 MDS 集群的“桶—服务器”显式表。

分裂时,桶到服务器的映射算法与上面的“桶—服务器”映射算法相同,在满足负载均衡条件的情况下,尽量使伙伴桶位于同一服务器上,以减少数据迁移带来的开销。数据迁移可以有两种方式实施:一是集中迁移,二是分散迁移。集中迁移是一次性迁移完所有数据,即在桶分裂为新桶分配服务器后,分配了新桶的服务器向其伙伴桶服务器提出数据迁移申请,将新桶的数据迁移到该服务器上存储。这种方式的优点是数据迁移时间比较短,服务器集群在较短时间内就能以新的服务器集群进行服务,但在迁移过程中,有大量的数据移动操作,将会给服务器和网络带来很大的开销,会严重影响服务器集群的性能。分散迁移是不进行集中的数据迁移,而是在服务过程中缓慢实现数据的迁移,当访问某个元数据时,检测到新分配的服务器不同于旧分配的服务器时,就将数据

迁移到新服务器中。这种方式的优点是没有明显的的数据迁移时间，MDS 集群能够持续服务，但整个数据迁移时间会很长，而且对于还未迁移的元数据服务出现首次不命中的情况，服务效率低。

```
//0号服务器
{
    检查集群剩余容量num_b_unused之和;
    while(剩余容量sum_of_num_b_unused<已用容量countb){
        提示需要扩展服务器;
        if(收到申请加入消息){
            更新集群信息;
            广播集群信息;
        }
        continue;
    }
    sl=sl+1; /*分裂级别增长*/
    为新分裂的桶分配空间;
    将新桶映射到服务器;
    广播桶—服务器显式表;
    数据迁移;
}
//其它服务器:
{
    if(收到集群信息)更新集群信息;
    if(收到桶服务器显式表)更新显式表;
    数据迁移;
}
//新服务器:
{
    Step1: 获取集群信息
    发消息给0号服务器MDS, 申请加入服务器集群;
    Step2:
    If(收到集群信息)更新集群信息;
    If(收到桶服务器显式表)更新显式表;
    数据迁移;
}
```

图 4. 11 MDS 集群扩展算法

系统初始化时，系统中的桶数应大于集群中的服务器数量，保证系统初始化后系统的是负载均衡的。当桶容量达到阈值时，桶进行分裂，原桶中的数据在原桶和伙伴桶中进行分配，迁移桶中一半的数据到新的桶中，数据迁移量比较小。

在 OCFS II 的结构中，MDS 集群中相当于 DPIS 的后端存储，只有在 DPIS 中没有缓存数据才进行访问，若是热点访问数据，那么 DPIS 在 cache 中的命中率将大大增加，

不会引起 MDS 负载大的变化，因此，对 MDS 集群只采用静态分配算法即可。

本章主要研究集群系统负载均衡的方法及具体实现，结合 OCFS II 元数据管理集群中 DPIS 集群和 MDS 集群的不同特点，分别实现各自的负载均衡，保证整个元数据管理集群的负载均衡。DPIS 集群采用以文件全路径名 hash 值 `dir_hash` 为特征的 hash 静态负载均衡与动态反馈重分配相结合的负载均衡方案，MDS 集群中采用以元数据 `fid` 为特征值的静态数据分布存储，从而实现负载分流的负载均衡方案。下一章，将研究 OCFS II 集群系统的可用性，在可用性方面对 OCFS II 元数据管理集群做一些改进。

第五章 OCFS II 元数据管理集群可用性方案研究

§ 5.1 集群系统可用性概述

计算机系统的可用性为系统保持正常运行时间的百分比。计算机系统的可靠性用平均无故障时间 (MTTF, Mean Time To Fail) 来度量, 即计算机系统平均正常运行多长时间, 才发生一次故障。系统的可靠性越高, 平均无故障时间就越长。可维护性或者可服务性用平均维修时间 (MTTR, Mean Time To Repair) 来度量, 即系统发生故障后维修和重新恢复正常运行平均花费的时间。系统的可维护性越好, 平均维修时间越短。计算机系统的可用性定义为: $MTTF/(MTTF+MTTR) * 100\%$ ^[39]。

集群系统高可用性主要包括集群服务数据和节点的高可用性。集群服务数据的高可用性是指系统总能保证服务所需要的数据服务, 部分数据失效后, 可以有替代方法恢复数据和保证数据可用; 节点的高可用性是指部分节点的功能失效不影响系统整体的功能。

通过硬件冗余或软件的方法都可以提高系统的可用性。硬件冗余主要是通过通过在系统中维护多个冗余部件如硬盘、网线和服务器等来保证工作部件失效时可以继续使用冗余部件来提供服务; 而软件的方法是通过软件对集群中的多台机器的运行状态进行监测, 在某台机器失效时启动备用机器接管失效机器的工作来继续提供服务。通常, 采用软、硬件结合的方法能够获得更好的效率。

数据存储的容错主要有两种方法^[40]: 一是将数据分片和其冗余校验信息一起存储在一组节点上, 当某些数据片损坏不可用时, 通过其它数据片和校验信息计算获得相应的数据, 使数据恢复和可用。这种方法主要有 software RAID、RAID^[23]等。二是数据复制, 这种方法在集群的节点间相互备份数据。当节点失效时, 由于集群内有文件的多个副本存在, 集群可以继续为客户提供服务。当前存在多种数据复制技术, 主要有 mirrored disks、distorted mirroring 等。

对于服务器集群系统来说, 保证节点高可用性要解决三个问题:

①失败节点的检测。当某个服务器节点或者部分网络出现故障时, 系统本身能够感知故障的出现, 采取相应的措施。

②选举一个节点来接替失败节点的工作。确认某个节点不能正常服务后, 能够在集群中选出相应的部分接替失败节点的工作。

③选出的节点接管失败节点的工作, 集群继续向客户提供服务。

§ 5.2 OCFS II 元数据管理集群可用性方案

元数据服务的高可用性是基于存储的分布式文件系统具有高可靠性的前提,提供元数据服务的元数据管理集群应能够提供高可用性。元数据管理集群可以对客户掩盖服务器失效(server failure),在集群中某个服务器或者部分网络出现故障的情况下能够继续提供服务。在故障排除后,元数据管理集群能够进行透明恢复,并保证整个文件系统恢复一致的状态。基于大型集群的经验,多点同时失效是很少出现的,所以系统设计主要考虑单点失效。基于这个前提,元数据管理集群的可用性方案主要保证元数据管理集群中出现单点失效时能够系统容错,在此基础上,可以在某些特定情况下对多点失效进行容错。

在本文的 OCFS II 系统中,元数据的管理由目录路径索引服务器和元数据服务器两个集群系统来共同完成,分别对它们采用不同的失效恢复机制来保障整个元数据管理集群的可用性。

5.2.1 目录路径索引服务器 DPIS 集群可用性方案

OCFS II 系统的目录路径索引服务器 DPIS 集群采用共享存储的集群系统,每个服务器都能完成相同的服务,它们能够完成的服务依赖于共享存储的数据。DPIS 集群的可用性分两个方面来保障,一是保证出现节点故障的可用性;二是保证共享的目录数据的可用性^[42]。

(1) DPIS 集群的节点容错机制

①失败节点的检测。设置元数据处理请求的超时机制来检测失败节点。client 向某个 DPIS 节点请求元数据服务,当出现请求超时,可能是服务器节点繁忙,也可能是该节点出现故障,client 启动侦听程序,检查该节点是否无法响应网络,若是,将该 DPIS 记录为失败节点,并通知 DPIS 服务器集群检查失败节点状态。

②选取接替节点。由于每个 DPIS 服务器节点都能完成相同的功能,所以接替节点的选取就相对比较容易。Client 可以采用动态负载均衡中选取备用节点的相同策略,从其余服务器中选择一个负载量相对较少的服务器作为接替节点。

③接替节点接管工作。Client 将请求重新发送给新选出的服务器,完成元数据服务的请求处理。

(2) DPIS 集群中目录元数据的可用性

DPIS 集群共享数据的高可用性是实现 DPIS 集群高可靠性的关键。在 DPIS 集群中,所有 DPIS 节点完成的功能都依赖于共享存储的目录元数据,一旦目录元数据不可用,所有的 DPIS 节点将不能处理元数据请求。

DPIS 集群可以采用共享磁盘体系结构,通过采用共享的磁盘阵列可以解决共享的文件元数据容错和数据一致等可靠性问题,这种方案中通过共享的廉价磁盘阵列

(RAID) 柜或网络存储设备来实现数据共享。目前磁盘阵列技术已相当成熟, 多数产品都自含大量冗余部件和技术(如冗余控制器、冗余电源、冗余风扇、CACHE 电池保护装置、硬盘回路散热技术、硬盘防尘技术等)来保障磁盘阵列自身的可靠性, 同时磁盘阵列的核心技术——RAID 技术则可确保数据的可靠性。此外, 存储区域网络(SAN)及网络化存储设备(NAS)的发展极为迅猛, 其取代传统存储技术而成为主流的趋势已十分明显, 数据存储设施和主机系统分离的方案正越来越多地被采用。存储区域网技术解决了集群的每个结点可以直接连接/共享一个庞大的硬盘阵列, 硬件厂商也提供多种硬盘共享技术, 如光纤通道(Fiber Channel)、共享 SCSI (Shared SCSI)。InfiniBand 是一个通用的高性能 I/O 规范, 它使存储区域网可以以更小的延时传输 I/O 消息和集群通讯消息, 并提供很好的伸缩性。InfiniBand 得到绝大多数的大厂商的支持, 如 Compaq、Dell、Hewlett-Packard、IBM、Intel、Microsoft 和 SUN Microsystems 等, 它正在成为一个业界的标准。这些技术的发展使得共享存储变得容易, 在 DPIS 集群的实现时可以使用这些技术。

共享存储通常采用数据库、网络文件系统或者分布式文件系统实现对目录元数据的管理。服务器节点需要动态更新的数据一般存储在数据库系统中, 同时数据库会保证并发访问时数据的一致性。静态的数据可以存储在网络文件系统(如 NFS/CIFS)中, 但网络文件系统的伸缩能力有限, 一般来说, NFS/CIFS 服务器只能支持 3~6 个繁忙的服务器结点。对于规模较大的集群系统, 可以考虑用分布式文件系统, 如 AFS^[24]、GFS^[25,26]、Coda^[27]和 Intermezzo^[28]等。分布式文件系统可为各服务器提供共享的存储区, 它们访问分布式文件系统就像访问本地文件系统一样, 同时分布式文件系统可提供良好的伸缩性和可用性。此外, 一般情况下, 根据负载均衡策略, 访问共享存储的同一目录元数据由同一 DPIS 服务器完成, 但处于极端情况下, 很多用户同时访问某个元数据时, 可能出现不同 DPIS 服务器同时读写访问分布式文件系统上同一目录元数据时, 元数据的访问冲突需要消解才能使得资源处于一致状态。这需要一个分布式锁管理器(Distributed Lock Manager), 它可以是分布式文件系统内部提供的, 也可以是外部的。在实现 DPIS 集群时, 可以根据采用的分布式文件系统的情况, 使用分布式文件系统内部的分布式锁管理器或者编写分布式锁管理器来保证不同的 DPIS 节点并发访问的一致性。

5.2.2 元数据服务器 MDS 集群可用性方案

元数据服务器集群采用无共享的集群体系结构。集群中的每个服务器存储一部分元数据, 每个服务器上的数据是唯一的, 没有其他的备份, 每个服务器提供自己独有的服务, 各个服务器之间相互独立, 互相之间没有通信。为了实现系统的可用性, 必须采取一定的冗余结构, 对数据进行一部分冗余存储, 在某点出现失效时由冗余结构接替失效点的工作。考虑系统的扩展性和执行的效率, 元数据服务器集群中采用备份服务器来保证集群的可用性。

在实现容错的 MDS 集群中, 包括三种类型的服务器: 主服务器, 备份服务器和空

闲服务器。一对主服务器和备份服务器称为伙伴服务器，数据互为备份。伙伴服务器存储有相同的元数据，可以完成相同的元数据服务功能，当某个服务器出现故障时，其完成的服务自动切换到其伙伴服务器上，由其伙伴服务器继续完成相应的元数据服务；空闲服务器暂时不工作，当主服务器或者备份服务器出现故障时接替宕机的服务器的工作，与宕机的服务器的伙伴节点组成临时的伙伴节点，正常进行数据备份和元数据服务。

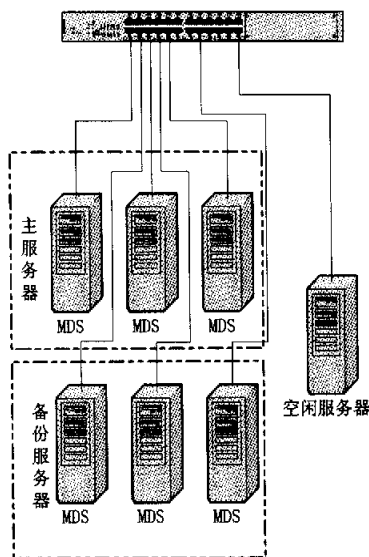


图 5.1 MDS 集群容错方案

假定系统中可能出现以下情况的失效：

①磁盘失效（disk failures）：服务器节点正常工作，但其磁盘数据出现问题，不能正确访问数据。

②临时节点失效（temporary site failures）：某个节点出现工作不正常，不能正常进行服务，经过短暂时间的修复可以重新使用；另外也可能是连接该节点的局部网络出现问题，一段时间后恢复。

③永久节点失效（permanent site failures）：也称之为灾难失效（disasters），是指某个节点出现故障，经过修复也不能够正常使用，并且所有磁盘的数据丢失。

节点的工作状态：正常状态，宕机状态，恢复状态。

容错的 MDS 集群中引入了伙伴服务器的概念，一对伙伴服务器中包括主服务器和备份服务器，每个桶的数据要同步地保存在主服务器和备份服务器上，实现数据的冗余存储，保证在一份数据失效时冗余数据能够接替工作。桶定义时，在没有容错机制时，每个桶只映射到一个服务器上，而实现容错机制后，桶映射到一对伙伴服务器上，同时在伙伴服务器的两个服务器上均有数据备份。图 5.2 为容错的 MDS 集群中服务器、伙伴服务器和桶结构的定义。


```

/*服务器定义*/
struct mdssever{
    int number; /*MDS服务器编号*/
    _u32 addr; /*服务器地址*/
    _u16 port; /*服务器端口*/
    int num_b_all; /*服务器中总的桶数量*/
    int num_b_unused; /*服务器中未使用桶数量*/
    int buddy; /*伙伴服务器编号*/
    int tmp_bud; /*临时伙伴服务器编号*/
    int mormal; /*工作状态标志: 0,宕机;1,工作正常;2,恢复状态*/
};
struct mdssever mdses[n];

/*伙伴服务器定义, 包括互为伙伴的两个服务器编号以及一个临时服务器编号*/
struct buddy_server{
    int server_no0;
    int server_no1;
    int temporary;
};
struct buddy_server buddes[nbud];

/*桶定义*/
struct bucket{
    int character; /*桶特征值*/
    struct buddy_server server; /*桶所在备份服务器编号*/
};
struct bucket buckets[countb];

```

图 5.2 容错的 MDS 集群中服务器、伙伴服务器及桶定义

元数据定位时, 元数据按照上一章的 MDS 集群桶分配算法定位到桶中, 该桶根据桶—服务器映射算法映射到一对伙伴服务器上, 当 DPIS 访问元数据时, 按桶分配算法确定元数据所在的桶。在这里桶的定义中替代某个服务器的是一对伙伴服务器, 一个主服务器和一个备份服务器, 根据元数据 fid 的特征确定哪个服务器是服务该元数据的主服务器, 哪个服务器是备份服务器。访问元数据时, 以访问主服务器为主, 元数据的访问分为读和写, 当为读访问时, 访问主服务器, 然后返回数据; 当为写访问时, 将数据写到主服务器上, 同时主服务器发送请求给备份服务器更新备份服务器数据。

执行桶分配算法后得到文件元数据所在的桶号 nb, 当前分裂级别为 sl, DPIS 选择服务器进行元数据操作算法为:

```

select_server(key, sl, buckets, mdses)
{
    nb=assign_bucket(key, sl);
    key>>sl+1;
    tmp=key&1ull;
    if(!tmp) /*选取0号伙伴特征*/
        other0:
        server_no=buckets[nb].server.server_no0;
        if(mdses[server_no].nomal){
            连接mdses[server_no]进行元数据操作;
            if(操作正常)
                return;
            else /*超时*/
                标志mdses[server_no]不正常工作;
                选取mdses[server_no]伙伴服务器操作;
                通知mdses[server_no]伙伴服务器检查mdses[server_no]状态;
        }
    }
    else
        goto other1;
}
else{
    other1:
    server_no=buckets[nb].server.server_no1;
    if(mdses[server_no].nomal)
        连接mdses[server_no]进行元数据操作;
        if(操作正常)
            return;
        else /*超时*/
            标志mdses[server_no]不正常工作;
            选取mdses[server_no]伙伴服务器操作;
            通知mdses[server_no]伙伴服务器检查mdses[server_no]状态;
    }
}
else
    goto other0;
}
标志不正常工作服务器状态;
}

```

图 5.3 容错的 MDS 集群元数据分配与定位算法

```

check_buddy(server_no)
{
    if(伙伴服务器正常 || 恢复状态)
        return;
    if(节点不正常)
        向DPIS返回伙伴节点宕机信息;
        发出警告进行人工检查;
        将空闲服务器置为临时伙伴节点;
    }
}

```

图 5.4 容错的 MDS 集群伙伴服务器状态检查算法

当服务器集群报警时,需要及时对失效的服务器进行人工检查,根据检查的结果作出相应的处理。如果失效服务器是永久节点失效,则将空闲服务器替代失效节点设置为

伙伴节点，并将服务器状态设置为恢复状态，向伙伴节点申请重建数据；当数据重建完成后设置该节点状态为正常状态，系统恢复到完全正常工作状况。如果是临时节点失效，则对该节点进行修复；修复完成后，将节点重新加入集群，该节点设置为恢复状态，然后复制空闲服务器上新数据到该节点中；数据复制完成后，将该节点重新设置为工作正常状态，并将伙伴节点的临时伙伴节点设置为 NULL。

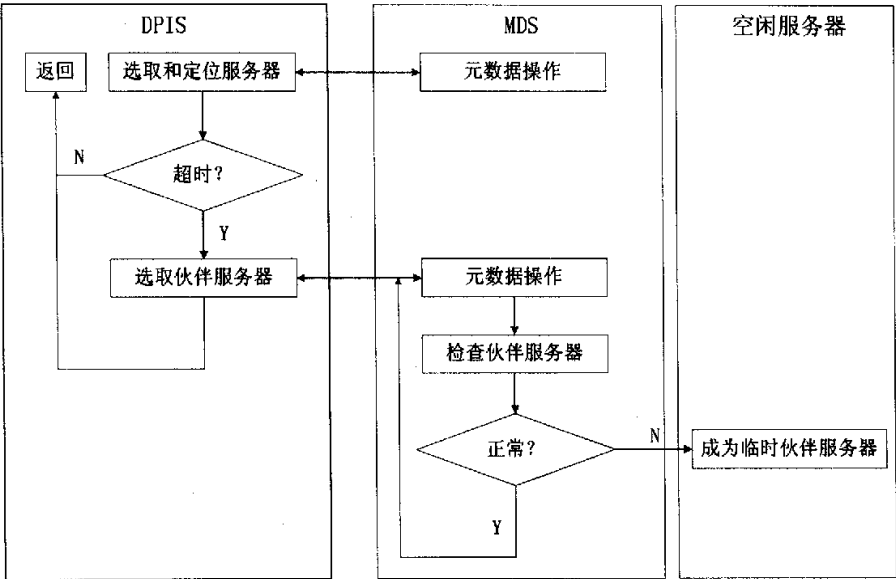


图 5.5 MDS 集群容错方案流程

按照前面的假设，系统只考虑单点失效的情况。当 MDS 集群中出现失效时，DPIS 通知要访问的服务器的伙伴服务器检查该服务器状态；伙伴服务器检查服务器状态并作出决策，结合人工检查决定恢复或者重建服务器；空闲服务器临时存储新产生的数据，必要时替代某个服务器加入集群工作。

§ 5.3 可用性方案分析

首先，对于 DPIS 集群，所获得的好处是集群的多个服务器节点失效后仍然能保证正常的元数据服务，但处理性能收到影响。当该集群正常工作时，1 次选取合适的服务器进行元数据服务，当服务器集群的某个节点失效时，自动选取另外的服务器进行元数据服务，需要两次操作。对于有 n 个服务器的 DPIS 集群，如果有 n_f 个服务器节点失效，则集群中只有 $n-n_f$ 个节点正常工作，工作效率只有正常工作时效率的 $\frac{n-n_f}{n}$ 。集群的共享存储的效率与集群采取的共享磁盘的数据存储容错方案相关，如果采用 RAID5 磁盘阵列，其效率与 RAID5 磁盘阵列的效率相同。硬件代价是在共享的磁盘阵列中必须保证相应的冗余。

其次, 对于 MDS 集群, 在有一半的服务器失效 (每一对伙伴服务器不同时失效) 的情况下仍然能保证正常服务。正常工作时, 若进行读操作, 1 次读可完成操作; 若进行写操作, 则需要两次写, 需要写主服务器并将数据同步到备份服务器。当出现服务器失效时, 需要两次选择服务器, 并进行相关的检查和恢复及重建操作。硬件代价是需要 $2n+1$ 个服务器。

从以上分析可以看出, 元数据管理集群以一定的硬件和性能代价保证系统的可用性, 使得系统能够在单点失效或者特定的多点失效情况下仍然能提供正常的元数据服务。

第六章 总结与展望

基于对象的存储技术是存储领域的新兴技术，它提出了一种新的存储体系结构，以数据对象作为数据存储和管理的基本单元。基于对象存储的文件系统在可扩展性、可靠性、易用性等方面都有显著提高，随着网络化存储技术的不断成熟，基于对象存储的文件系统将成为重要的发展方向。

元数据是对象文件系统中的核心数据，元数据服务管理对整个对象文件系统至关重要，对于大型的基于对象的分布式文件系统，其发展的趋势就是使用集群管理元数据。本文对基于对象存储文件系统中的元数据管理集群做了一些探索和研究，实现了对象存储文件系统原型 OCFS，符合基于对象存储的体系结构框架；其次，在原型 OCFS 基础上实现元数据管理集群，提出了原型改进的对象存储文件系统 OCFS II，OCFS II 系统中元数据管理集群包括 DPIS 集群和 MDS 集群，DPIS 集群管理目录元数据，MDS 集群管理文件自身元数据；第三，设计实现了 OCFS II 元数据管理集群的负载均衡方案，把静态负载分配与动态反馈负载重分配相结合，通过合适的元数据分割算法，保证元数据管理集群的负载均衡，并解决对“热点”数据访问的处理；第四，解决了 OCFS II 元数据管理集群可用性的问题，实现元数据管理集群单点失效和特定条件下多点失效的容错和恢复，保证系统的可靠性和可用性。

基于对象存储技术的研究还处在起步阶段，很多方面需要不断研究、实现和完善。本文做了一部分工作，实现了文件系统原型，但算法还未能与系统结合起来，下一步可以将算法与具体的系统结合起来作实际的测试研究；另外，管理元数据采用的元数据管理集群可以有很多灵活的体系结构，可以探索采用各种不同的体系结构并结合相应的算法，实现更好、更高性能的方案。

致 谢

在本论文完成之际，谨向在我申请硕士学位的过程中曾经指导过我的老师，关怀过我的领导，关心过我的朋友，和所有帮助过我的人们致以崇高的敬意和深深的感谢！

首先，要感谢我的导师徐德启教授。在研究生学习期间，徐老师给予了我全方位的指导、关心和帮助，在课题研究期间更是不辞辛苦地为我指点迷津，悉心指教，付出了大量的心血。徐老师严肃认真的治学态度、渊博的知识、严密的思维、踏实的作风和分析解决问题的能力、丰富的实践经验使我在学业上受益匪浅，他的高尚品质和敬业精神鞭策着我在学业上和工作中不断求实创新，开拓进取。

感谢同学党岗、贾涛在课题的研究过程中给予我的关心和帮助。

感谢兰州大学信息科学与工程学院给我提供了继续学习和深造的机会，感谢所有老师对我的关心、帮助和支持！

感谢我的家人，他们对我学业的完成给予了巨大的支持并做出了无私的奉献。

最后，向审阅本文的各位老师、各位学者表示衷心的感谢和崇高的敬意。

参考文献

- [1] Peter J. Braam(with others): The Lustre Storage Architecture, Cluster File Systems, Inc. <http://www.clusterfs.com>, 03/04/2004, HEAD
- [2] Lustre: A Scalable, High-Performance File System , Cluster File Systems, <http://www.lustre.org/docs/whitepaper.pdf>
- [3] <http://www.storagesearch.com/auspexart.html>, A Storage Architecture Guide
- [4] Network Attached Storage Architecture. Garth A. Gibson and Rodney Van Meter. COMMUNICATIONS OF THE ACM November 2000, Vol.43, No.11
- [5] Tom Clark. Designing Storage Area Networks. Addison-Wesley, 1999
- [6] Brent Welch, Garth Gibson, Managing Scalability in Object Storage Systems for HPC Linux Clusters, Clusters Brent Welch (2004)
- [7] http://www.panasas.com/docs/Object_Storage_Architecture_WP.pdf
- [8] David Nagle, Denis Serenyi, Abbie Matthews, "The Panasas ActiveScale Storage Cluster-Delivering Scalable High Bandwidth Storage", SC2004
- [9] <http://www.t10.org/ftp/t10/drafts/osd/osd-r10.pdf>
- [10] Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long and Lan Xue: Efficient Metadata Management in Large Distributed Storage Systems, In: the 17th International Parallel and Distributed Processing Symposium(IPDPS 2003), April 2003
- [11] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, and Ethan L. Miller: Dynamic Metadata Management for Petabyte-scale File Systems, University of California, Santa Cruz
- [12] Julian Satran, Eyal Henis, Pinna Vortman: Trends in Storage Infrastructure, IBM Research Laboratory in Haifa
- [13] Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, File System Workload Analysis For Large Scale Scientific Computing Applications, 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2004, College Park, Maryland, USA
- [14] Object based storage devices (OSD) , <http://www.intel.com/labs/storage/osd/tech.htm>
- [15] Levy E, Silberschatz A. ACM Computer Surveys, 1990, 22 (4) : 321
- [16] Cobett P F, Feitelson D G. ACM Transactions on Computer Systems, 1996, 14 (3) : 225

- [17] http://vendor.dostor.com/arena/compare_rtg.asp
- [18] R. J. Honicky, Ethan L. Miller, An Optimal Algorithm for Online Reorganization of Replicated Data, Storage Systems Research Center Jack Baskin School of Engineering University of California, Santa Cruz Santa Cruz, CA 95064
- [19] WITOLD LITWIN, MARIE-ANNE NEIMAT, and DONOVAN A. , LH*—A Scalable, Distributed Data Structure, ACM Transactions on Database Systems, Vol. 21, No. 4, December 1996, Pages 480 - 525.
- [20] Ashok Rathi, Huizhu Lu, G.E. Hedrick, PERFORMANCE COMPARISON OF EXTENDIBLE HASHING AND LINEAR HASHING TECHNIQUES, 1990 ACM 089791-347-7/90/0003/0178
- [21] Kyoji Kawagoe, Modified Dynamic Hashing, 1985 ACM 0-89791-160-1/85/005/0201
- [22] FL J. ENBODY, H. C. DU, Dynamic Hashing Schemes, ACM Computing Surveys, Vol. 20, No. 2, June 1988
- [23] Michael Stonebraker, Gerhard A. Schloss: DISTRIBUTED RAID -- A NEW MULTIPLE COPY ALGORITHM, Walter A. Haas School of Business University of California, Berkeley, Berkeley, CA 94720
- [24] J.H. Howard. An Overview of the Andrew File System. In Proceedings of the USENIX Winter Technical Conference, Dallas, TX, USA, February 1998.
- [25] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, and Matthew T. O'Keefe. A 64-bit, Shared Disk File System for Linux. In Proceeding of 16th IEEE Mass Storage Systems Symposium, San Diego, CA, USA. March 15-18, 1999.
- [26] Global File System Website. <http://www.globalfilesystem.org/>.
- [27] Coda File System Website. <http://www.coda.cs.cmu.edu/>.
- [28] InterMezzo File System Website. <http://www.inter-mezzo.org/>.
- [29] Wensong Zhang: Linux Virtual Server for Scalable Network Services, <http://www.LinuxVirtualServer.org/>
- [30] 陈凯、白英彩, 网络存储技术及发展趋势, 电子学报, Dec 2002
- [31] 潘国腾、谢伦国, “网络存储技术研究”, 计算机研究与发展, Aug, 2003.
- [32] 刘仲、王晓东、周兴铭, 可伸缩的元数据集群系统设计, 高技术通讯, 2004 年 8 月
- [33] 林凡, 集群的可扩展性及其分布式体系结构, http://www-900.ibm.com/developerWorks/cn/linux/cluster/cluster_system/base
- [34] 李登, 分布式系统负载均衡策略研究, 中南大学硕士学位论文

- [35] 肖侖、黄金锋、卢宇彤, 网络并行计算的动态负载平衡策略, 计算机工程与科学, 第 20 卷, 第 3 期, 1998. 8
- [36] 王广芳等, 分布式计算机系统 (DCS) 负载均衡算法 20 年, 计算机工程与设计, 第 16 期, 第 5 卷
- [37] 刘爱洁, 负载均衡技术浅析, 信息产业部北京邮电设计院第七届新技术论坛
- [38] 刘健, 徐磊, 张维明, 基于动态反馈的负载均衡算法, 计算机工程与科学, 2003 年第 25 卷第 5 期
- [39] 陈国良等, 并行计算机体系结构, 高等教育出版社
- [40] 杨俊杰, 徐捷, 一种并行文件系统数据容错设计, 微计算机应用第 25 卷第 4 期
- [41] 章文嵩、王召福、刘仲, 基于对象存储的集群文件系统 CFSlight 设计与实现, 大连理工大学学报, 2003 Vol. 43 No. z1
- [42] 贾瑞勇、张延园, SAN 文件系统元数据服务器集群体系结构及关键技术研究, 第十三届全国信息存储技术学术会议论文集
- [43] 邵强、刘仲、窦勇, 对象文件系统中元数据服务器集群负载均衡研究, 第十三届全国信息存储技术会议

作者: [娄成龙](#)
学位授予单位: [兰州大学](#)

参考文献(43条)

1. [Peter J Braam](#) [The Lustre Storage Architecture](#) 2004
2. [Lustre:A Scalable,High-Performance File System,Cluster File Systems](#)
3. [A Storage Architecture Guide](#)
4. [Garth A Gibson](#), [Rodney Van Meter](#) [Network Attached Storage Architecture](#) 2000(11)
5. [Tom Clark](#) [Designing Storage Area Networks](#) 1999
6. [Brent Welch](#), [Garth Gibson](#) [Managing Scalability in Object Storage Systems for HPC Linux Clusters](#) 2004
7. [查看详情](#)
8. [David Nagle](#), [Denis Serenyi](#), [Abbie Matthews](#) [The Panasas ActiveScale Storage Cluster-Delivering Scalable High Bandwidth Storage](#)
9. [查看详情](#)
10. [Scott A Brandt](#), [Ethan L Miller](#), [Darrell D E Long](#) and [Lan Xue](#):[Efficient Metadata Management in Large Distributed Storage Systems](#) 2003
11. [Sage A Weil](#), [Kristal T Pollack](#), [Scott A Brandt](#), [Ethan L. Miller](#) [Dynamic Metadata Management for Petabyte-scale File Systems](#)
12. [Julian Satran](#), [Ealan Henis](#), [Pnina Vortman](#) [Trends in Storage Infrastructure](#)
13. [Feng Wang](#), [Qin Xin](#), [Bo Hong](#), [Scott A. Brandt](#), [Ethan L. Miller](#), [Darrell D. E. Long](#) [File System Workload Analysis For Large Scale Scientific Computing Applications](#) 2004
14. [Object based storage devices \(OSD\)](#)
15. [Levy E](#), [Silberschatz A](#) [查看详情](#) 1990(04)
16. [Cobett P F](#), [Feitelso D G](#) [查看详情](#) 1996(03)
17. [查看详情](#)
18. [R J Honicky](#), [Ethan L Miller](#) [An Optimal Algorithm for Online Reorganization of Replicated Data](#)
19. [WITOLD LITWIN](#), [MARIE-ANNE NEIMAT](#), [DONOVAN A LH*](#)-A Scalable, Distributed Data Structure 1996(04)
20. [Ashok Rathi](#), [Huizhu Lu](#), [G E Hedrick](#) [PERFORMANCE COMPARISON OF EXTENDIBLE HASHING AND LINEAR HASHING TECHNIQUES](#) 1990
21. [Kyoji Kawagoe](#) [Modified Dynamic Hashing](#) 1985
22. [FL J ENBODY](#), [H C DU](#) [Dynamic Hashing Schemes](#) 1988(02)
23. [Michael Stonebraker](#), [Gerhard A Schloss](#):[DISTRIBUTED RAID -- A NEW MULTIPLE COPY ALGORITHM](#)
24. [J H Howard](#) [An Overview of the Andrew File System](#) 1998
25. [Kenneth W Preslan](#), [Andrew P Barry](#), [Jonathan E Brassow](#), [Grant M. Erickson](#), [Erling Nygaard](#), [Christopher J. Sabol](#), [Steven R. Soltis](#), [David C. Teigland](#), and [Matthew T. O'Keefe](#) [A 64-bit, Shared Disk File System for Linux](#) 1999
26. [Global File System Website](#)

27. [Coda File System Website](#)
28. [Inter Mezzo File System Website](#)
29. [Wensong Zhang Linux Virtual Server for Scalable Network Services](#)
30. 陈凯, 白英彩 [网络存储技术及发展趋势](#)[期刊论文]-[电子学报](#) 2002(z1)
31. 潘国腾, 谢伦国 [网络存储技术研究](#) 2003
32. 刘仲, 王晓东, 周兴铭 [可伸缩的元数据集群系统设计](#) 2004
33. 林凡 [集群的可扩展性及其分布式体系结构](#)
34. 李登 [分布式系统负载均衡策略研究](#)
35. 肖依, 黄金锋, 卢宇彤 [网络并行计算的动态负载均衡策略](#) 1998(03)
36. 王广芳 [分布式计算机系统\(DCS\)负载均衡算法20年](#)
37. 刘爱洁 [负载均衡技术浅析](#)
38. 刘健, 徐磊, 张维明 [基于动态反馈的负载均衡算法](#)[期刊论文]-[计算机工程与科学](#) 2003(5)
39. 陈国良, 吴俊敏, 章锋, 章隆兵 [并行计算机体系结构](#)
40. 杨俊杰, 徐捷 [一种并行文件系统数据容错设计](#)[期刊论文]-[微计算机应用](#) 2004(4)
41. 章文嵩, 王召福, 刘仲 [基于对象存储的集群文件系统CFSlight设计与实现](#)[期刊论文]-[大连理工大学学报](#) 2003(z1)
42. 贾瑞勇, 张延园 [SAN文件系统元数据服务器集群体系结构及关键技术研究](#)[会议论文]
43. 邵强, 刘仲, 窦勇 [对象文件系统中元数据服务器集群负载均衡研究](#)[会议论文]

相似文献(10条)

1. 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均衡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。

2. 期刊论文 单颖, 姚念民, 赵建明, Shan Ying, Yao Nianmin, Zhao Jianming [基于对象存储系统体系结构的研究](#) - [计算机研究与发展](#)2009, 46(z2)

随着基于对象存储系统的快速发展, 存储系统服务性能要求越来越高. 在分析传统基于对象存储系统体系结构的基础上, 提出一种体系结构SOBSS, 通过扩展元数据服务器功能, 简化体系结构内部数据交互模式. 通过与传统基于对象存储系统体系结构的性能比较, 实验结果表明, 采用SOBSS体系结构在提高系统性能方面有明显的优势.

3. 学位论文 刘仲 [基于对象存储结构的可伸缩集群存储系统研究](#) 2005

随着处理器和网络技术的飞速发展, 大大的提高了Linux集群计算的计算能力. Linux集群计算在高性能科学计算、商业应用和海量信息服务等领域得到了广泛应用, 逐渐发展成为高性能计算中的流行方法. 而受传统存储结构的限制, 其计算能力得不到充分体现. 构建满足Linux集群计算需求的可伸缩、高性能、跨平台、安全、共享数据的存储结构对现有的存储结构提出了巨大的挑战.

新兴的对象存储结构能够利用现有的处理技术、网络技术和存储组件提供空前的可伸缩性和聚合吞吐量, 为构建新一代的大规模并行存储系统提供了基础. 本文在全面深入了解对象存储结构与现有对象存储系统的基础上, 对基于对象存储结构的大规模集群存储系统涉及的几个关键技术进行了深入研究, 提出了新颖有效的实用算法. 主要的贡献如下:

(1) 提出一种基于确定性算法分布目录对象和数据对象的可伸缩集群文件系统的框架结构, 改进了现有对象存储系统的元数据和数据对象的管理方法. 基于确定性算法自主计算数据分布的方法简化了大规模存储系统的管理, 支持元数据服务器、存储节点的动态均衡扩展.

(2) 首次提出目录路径属性与目录对象分离的元数据管理方法, 扩展了现有的对象存储结构. 该方法能够有效避免因为目录属性的修改而导致的大量元数据更新与迁移; 通过减少前缀目录的重叠缓存提高了元数据服务器Cache的利用率和命中率; 通过减少遍历目录路径的开销和充分开发目录访问的存储局部性, 减少了磁盘I/O次数; 通过元数据服务器的动态负载均衡避免单个服务器过载. 实验结果表明该方法在提高系统性能、均衡元数据分布以及减少元数据迁移等方面具有明显的优势.

- (3) 首次在研究数据对象的分布中引入MonteCarlo方法，提出一种基于动态区间映射的数据对象布局算法，支持权重分布和副本，在均衡数据分布和最少迁移数据方面都是统计意义上最优的，有效解决了动态存储系统的数据均衡分布问题，提高了系统的可扩展性。该算法的基本思想是将数据对象与随机数对应起来，将存储节点与容纳随机数的区间对应起来，将离散空间中的数据对象分布问题转化为连续空间中的区间分割问题。根据系统中存储节点的规模和权重将单位区间分割成不同长度的区间，并在区间与存储节点之间建立映射关系，通过两次映射确定数据对象的存储位置。理论分析和实验结果表明数据对象分布具有统计意义上的均衡性、自适应性和迁移最优性，定位数据对象速度快。
- (4) 提出一种可伸缩分布式节点地址计算方法。该算法使得数据对象分配地址的计算不依赖于中央节点计算或者访问一个集中式目录，所有计算节点和存储节点独立地进行地址计算，并且对数据对象的访问操作或新增节点引起的系统规模变化不需要原子更新到其他计算节点，计算节点通过视图校正算法自主学习，自动适应新的系统规模。消除了现有的集中式访问性能瓶颈，使系统具有高可伸缩性。
- (5) 分别提出基于镜像和分组的高可用数据对象布局算法，借鉴RAID的方法在算法一级上实现数据的冗余分布。在数据对象和存储节点失效时，利用冗余数据重构数据对象和存储节点，保证存储系统的高可用性。采用马尔可夫激励模型分别对基于镜像和分组的高可用数据对象布局算法的存储系统进行定量的可用性分析，计算结果表明两种方法能够有效保存存储系统的高可用性。
- (6) 对基于动态区间映射的数据对象布局算法进行扩展，提出支持节点组的数据对象副本布局算法，支持多个节点同时批量扩展，扩展后的数据对象布局算法仍然保证数据对象的分布是负载均衡的，支持权重分布和副本。理论分析和实验结果表明数据对象分布具有统计意义上的均衡性、自适应性和迁移最优性，定位数据对象速度快。
- (7) 利用上述研究成果，在Linux操作系统上设计与实现了一个基于对象存储的集群文件系统原型。

4. 学位论文 [刘群 基于可扩展对象的海量存储系统研究](#) 2006

信息存储是人类社会永恒的需求。随着计算机技术的发展和应用的普及，信息存储容量成爆炸性地增长，现有网络存储系统已无法满足人们对于存储的需要。基于对象存储（Object-Based Storage，OBS）技术适时崛起，利用现有的存储组件、处理技术和网络技术，通过简单方式来获得前所未有的高吞吐量，成为下一代网络存储的主流。它采用包含数据和属性的“对象”作为接口，既有了“块”接口的快速，又有“文件”接口的便于共享，并分离了存储数据的逻辑视图和物理视图，将存储数据的逻辑视图保留在元数据服务器中，而物理数据存放在基于对象存储设备（Object-Based Storage Device，OSD）中。同时，它将传统文件分解为系列数据对象，分发到一个或多个OSD中。虽然对象给存储系统带来了一种新的理念，但现有的与对象相关的存储系统中对象都仅定义为非定长的数据单位，束缚了“对象”这个有着丰富内涵的词汇。

基于可扩展对象的海量存储系统（Based on Scalable Object Mass StorageSystem，BSO-MSS）吸取了OBS 的优点，在“对象”现有的含义基础上扩充，使它不仅只包括用户数据，还将目录、文件、存储设备管理等纳入对象之中，形成层次结构的对象体系结构，实现对象的分布存储、层次管理的模式，并建立基于存储对象统一访问模式，将块、对象和文件三种存储接口进行融合与统一。这样不仅具有统一逻辑视图、数据共享、主动服务、并行访问、统一存储和易管理等特点，而且有着其他存储结构难以达到的高可扩展性和高性能。

通过建立系统广义随机Petri 网模型，对BSO-MSS 进行性能评价，模拟结果显示无论增加存储对象（Storage Object，SO）还是客户端，系统性能都随之增加。并采用测试工具iozone 对系统原型与Lustre 系统作对比测试，测试结果表明写性能超过Lustre，读性能略比Lustre 好，并验证了BSO-MSS 的广义随机Petri 网模型。

首次将存储系统与元胞自动机相结合，利用元胞自动机的原理，解析BSO-MSS动力演变规律。构建了一个通用框架的BSO-MSSCA 概念模型框架，并在此基础上，分析了两种具体元胞自动机模型。基于存储对象负载分配模型是将SO 解析为元胞，模拟了一个简单的负载均衡分配的动态变化，高度概括了BSO-MSS 的演变过程。

基于数据对象访问行为模型则分析数据对象的访问频率对系统的影响，结合数据对象访问的特征和主动性，通过机械学习适当调整数据对象的访问行为频率，使系统朝着稳定方向发展。通过分析基于存储对象的负载分配模型和基于数据对象的访问行为模型的演变过程，可以看出系统具有主动性、共享性、并行性、相关性等特性，是一个自组织管理的对象存储系统。

大规模分布式存储系统中，元数据高性能服务、负载均衡以及扩展性已成为一个重要的研究热点。在元数据服务器中，将元数据分解为目录对象和文件对象，目录对象为定位性元数据，提供文件所在位置和访问控制；文件对象为描述性元数据，描述文件的数据特性。每一个元数据服务器（Metadata Server，MDS）负责所有目录对象和自身的文件对象，这样充分利用MDS 中Cache，提高Cache 的命中率，减少磁盘I/O 次数，而且能够动态扩展MDS。同时，以目录对象ID 和文件名关键字的哈希值作为局部元数据查找表（Local Metadata Lookup Table，LMLT）的索引，获得相应的MDS_ID。一旦目录权限改变、更名、移动目录、修改权限等都不会造成元数据的迁移。通过Bloom Filter 算法将每个MDS 的LMLT 压缩成一个摘要，能够实现快速的元数据查找。同时采用主从备三链式结构的MDS 服务，不仅在未提高硬件成本下能够保证系统高可靠性和可用性，而且根据热点访问进行迁移，实现负载均衡。SO 是BSO-MSS 重要组成单位，它与OSD 不同之处是本身具有“接口”与“状态”标识，由数据、属性和方法组成，这样对现有的T10 OSD 标准进行了扩充。由于数据对象是通常在二维空间中命名，传统文件系统管理大量数据对象的效率是极其低，采用线性哈希查找算法，由负载因子控制分裂和合并，与传统文件系统的树结构查找相比，哈希法查找时间复杂度为O(1)。同时，针对Ext2 文件系统中数据访问至少需两次以上的磁盘操作特性，将数据的块地址和长度链接在一起，作为对象的扩展属性，连同数据对象一起存储到磁盘中，这样无论数据对象大小为多少，磁盘访问次数仅为两次。在BSO-MSS 中，负载与众多因素相关，如请求队列长度、CPU 处理能力、内存大小、网络带宽、磁盘带宽和磁盘容量等。负载柔性放置策略不仅考虑网络的影响，而且考虑SO 之间存在差异，并设置权重，权重大的SO 担负较多的负载。依据SO 属性中信息统计出负载特征，以系统响应时间为代价，自适应选择SO 数目，采用不同大小的分条进行存储，使BSO-MSS 具有更高的性能、可扩展性和自适应负载均衡能力。

5. 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。

6. 期刊论文 [谈华芳, 孙丽丽, 侯紫峰. TAN Huafang, SUN Lili, HOU Zifeng 一种基于对象存储中的元数据组织管理方法 - 计算机工程](#)2005, 31 (20)

提出了一种动态分区元数据组织管理方法.它混合了动态和静态的方法在MDS机群中分布元数据,并使用散列的技术索引元数据,利用共享存储来存放元数据.整个方法使得元数据访问可以高效地完成,机群的失败接管和扩展获得好的性能.

7. 学位论文 [韩德志 固定内容海量存储技术的研究](#) 2008

随着网络应用的普及和企业信息化的不断深入，固定内容数据的急剧增长，迫切需要大容量、高性能、高可用、易管理、易检索的海量网络存储系统。因此，针对固定内容的海量网络存储系统的研究具有重要的学术价值和实用价值。

本文针对固定内容存储与管理问题进行了深入的研究，详细阐述和分析了国内外网络存储和内容管理等方面的研究与技术发展情况，深入研究了基于内容的对象存储技术、元数据技术，探讨了将内容寻址存储和内容管理结合的方法，提出并建立了对固定内容进行描述（特征元数据）、定位和检索的元数据模型，并设计和实现了一种针对海量固定内容数据存储的原型系统，在此基础上对其性能进行分析和评估。

本文对海量固定内容存储系统的体系结构、文件系统设计、元数据管理实现等进行了详细的介绍，测试和分析系统的存储效率，并对系统的特色和具体应用进行了详细的说明。

本文的创新点如下：

1. 在研究对象存储和内容管理实现方法基础上提出内容存储的概念，并比较系统地总结内容存储的理论；
 2. 提出并实现了针对海量固定内容存储系统的多协议文件系统，满足不同用户的访问需求，该文件系统与相应安全算法结合，充分保证了整个存储网络系统中各个用户所存信息的安全性；
 3. 提出并实现了海量固定内容存储系统的特征元数据的提取方法、元数据管理模型以及元数据检索和内容数据定位方法；
 4. 通过集成创新构建了一个固定内容的归档存储原型系统，设计了多组实验测试并验证了基于对象存储的固定内容存储系统的性能。
- 本研究丰富了网络存储理论与实践，为企业面临的固定内容存储和管理问题提出了一个有效的解决方案。

8. 学位论文 [付印金](#) [PB级文件系统元数据管理关键技术的研究与实现](#) 2008

随着高性能计算技术和因特网技术的不断发展，数据资源迅猛增长，很多应用的存储需求达到PB级。为了消除存储瓶颈，有效地支持高性能计算，继DAS、NAS和SAN三种网络存储技术之后，基于对象的存储技术成为存储领域的新兴技术，并形成了一种新型的存储结构。构建在基于对象存储结构上的PB级文件系统可以有效地管理数据资源，为用户提供一个虚拟化大容量存储器的统一访问接口、高I/O带宽、以及可扩展的存储服务。对文件数据的访问需要借助于元数据，元数据管理对数据管理至关重要。PB级文件系统具有TB级的元数据，为了消除元数据访问瓶颈，必须由元数据服务器集群来管理元数据，使得其元数据管理更具有挑战性。

本课题主要研究PB级文件系统的元数据管理。首先，通过对Lustre和PVFS的I/O性能测试，比较分析了基于对象的文件系统相比于传统并行文件系统的性能优势，并分析PB级文件系统结构和各组成部分的软件模块结构及其功能。其次，提出自适应的动态目录元数据划分方法来有效地平衡元数据服务器集群的负载，同时，最少化平衡负载过程中的元数据迁移量，并通过开发目录局部性，提高元数据服务器cache的性能。再次，采用基于层次的计数型布隆过滤器数组能够提供快速的元数据查询服务，并能够节省内存开销。最后，根据元数据的特点，在结合文件访问语义和访问历史记录定义文件相关度的基础上，设计新的元数据预取策略来提高缓存命中率，降低平均响应时间。

9. 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。

10. 期刊论文 [刘仲](#) [章文嵩](#) [王召福](#) [周兴铭](#) [基于对象存储的集群存储系统设计](#) -[计算机工程与科学](#)2005, 27 (2)

集群存储是解决大规模数据存储的重要方法。本文提出一种基于对象存储的集群存储系统结构，将文件分为目录路径元数据、文件元数据与数据对象三部分并独立管理。性能比较与分析表明，该方法能够支持超大规模的文件及超大容量的目录，明显地减少网络访问消息数量，提高访问性能，并且解决了因为修改目录而导致的大量元数据迁移问题。

本文链接: http://d.g.wanfangdata.com.cn/Thesis_Y823784.aspx

授权使用: 中科院计算所(zkyjsc), 授权号: a3346d33-8ce9-4631-af0a-9e400128acca

下载时间: 2010年12月2日