分类号 TP3 _____

UDC _____ 编号 _____

中国科学院研究生院
博士学位论文

分布式文件系统可扩展元数据服务关键问题研究

杨德志

指导教师 许 鲁 研究员

中国科学院计算技术研究所

申请学位级别 工学博士 学科专业名称 计算机系统结构

论文提交日期 2007 年 11 月 论文答辩日期 2008 年 2 月

培养单位 中国科学院计算技术研究所

学位授予单位 中国科学院研究生院

答辩委员会主席 刘志勇 研究员

声　　明

我声明本论文是我本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，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：　　日期：

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本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、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：　　导师签名：　　日期：

摘　要

文件系统元数据描述文件系统及其管理的文件，其访问效率是文件数据访问性能的关键因素。在海量网络存储环境中，随着系统应用的多样化、应用需求规模的不断扩大，如何充分利用系统资源，提供高扩展能力的文件系统元数据服务，成为大规模分布式文件系统研究的热点问题。

多种典型应用的访问统计结果表明，文件系统元数据具有活跃性、局部性、动态性、需要更改文件系统的多元数据请求的比例很少等特点。基于此特性，本文的主要创新点是，提出动态灵活的分布式文件系统元数据服务机制和策略，支持文件系统元数据服务的动态扩展。它包括元数据存储和访问两个方面：

1) 集中共享的元数据虚拟存储模型。以虚拟存储技术支持的存储资源透明扩展、分层的存储资源管理和动态的元数据资源分配、元数据对所有请求服务器可见等为基础，元数据存储服务有效分离元数据的存储和访问，为动态灵活的元数据请求服务提供支持。
2) 动态灵活的元数据请求服务机制和策略。文件系统元数据表现出活跃性、局部性和动态性等特征。动态分布决策将文件系统名字空间结构与元数据类型相结合，由用户访问动态驱动活跃元数据的请求分布。初步的对比评估结果表明，相对于目录子树分区法，其请求处理能力将提高 30%左右。

动态灵活的元数据服务机制为服务扩展能力提供基础。为解决两阶段提交等传统协议的不足，本文提出通过动态迁移协议，集中处理跨服务器请求，有效保证请求的原子性。在最坏情况下，动态迁移协议也能够减少 25%的处理时延，且其错误恢复的影响也小得多。

针对传统和新兴应用的实验验证了元数据服务扩展能力的有效性。针对生物信息计算 BLAST 的评估表明，元数据服务器的增加将带来 20%左右的元数据服务时间降低。通过对系统实现的优化，将可能获得近线性的元数据服务扩展能力。同时，实验结果还为未来的工作方向提供了参考：1）原型系统的实现优化；2）自适应的元数据请求分布决策模型；3）系统结构的优化，以更好支持超大规模的系统扩展需求。

关键词：网络存储系统；可扩展分布式文件系统元数据服务；蓝鲸集群文件系统；蓝鲸元数据服务器集群系统

Research on Key Issues of Scalable Distributed File System Metadata Service in Large-scale Networked Storage Systems

Yang Dezhi (Computer Architecture)

Directed By Xu Lu

Efficiency of metadata access is a key factor of that of file access. In large-scale networked storage systems, with increase of file system metadata requests, doing research on key issues of scalable file system metadata service becomes a key problem of distributed file system researches.

Based on characteristics of locality, dynamicity and rare cross-server requests of active metadata, a dynamically scalable distributed file system metadata service is provided which includes:

a) A symmetric architecture of metadata servers which bases on fully shared virtual storage model and hierarchical storage resource management which supports scalable storage resource management.

b) Flexible management of file system metadata requests distribution. Supported by hierarchical distribution management, flexible metadata requests distribution which keeps metadata locality in mind decides requests distribution dynamically, which supports dynamic changes in system.

And more, there is a lightweight protocol for atomicity of cross-server metadata requests. Based on the symmetric architecture and fully shared storage model, requests which involve metadata distributed on multiple servers will be processed on single server by migrating active metadata between servers.

Evaluations on the prototype verify its scalability. And it also points out future works on architecture adjustment, adaptive metadata requests distribution, performance tuning and high-available metadata service, and so on.

Keywords: networked storage system; scalable distributed file system metadata service; BlueWhale file system; BlueWhale Multiple Metadata servers System

目　录

图目录

表目录

第一章 引言

随着服务器技术、存储技术和网络技术的发展，网络存储逐渐成为网络服务器系统的主要存储架构。网络存储系统通过集中管理存储资源，为应用提供有效的存储资源共享。大型分布式文件系统，是解决大规模集群亟需的海量集中存储、文件级共享、高并发带宽、高可扩展性的关键技术。

文件系统元数据是描述文件系统和文件的数据，文件访问包括文件元数据和数据的访问。应用规模的扩大和新应用的出现，对分布式文件系统元数据服务的性能和扩展能力提出更高的要求。在新的存储系统架构下，提供高性能、可扩展、动态平衡负载、高可用的元数据访问服务，是大型分布式文件系统研究的热点问题。

本论文主要研究大型分布式文件系统中元数据服务的可扩展性问题。期望通过深入探讨影响分布式文件系统元数据服务扩展能力的关键问题，解决分布式文件系统元数据服务的可扩展性问题，满足应用动态变化的需求，推进网络存储技术的发展。

1.1 研究的目标和意义

文件系统为应用提供数据存储服务，支持应用间的文件级数据共享。通用的 UNIX® 文件系统模型[Bovet2002][Daley1965]包括描述文件系统属性的超级块、描述文件属性的索引节点、存放目录项的目录数据块、存放数据对应的设备块号等信息的元数据块、以及存放文件数据的数据块等，除文件数据外的所有信息统称为“文件系统元数据”，简称“元数据(metadata)”。文件系统通过称为“目录”的特殊文件，组织成倒置的树形结构。从根目录出发，通过目录树遍历，应用可以定位需要访问的文件。应用能够访问到的文件名字构成文件系统名字空间（file system namespace）。

文件访问包括文件的元数据和数据访问两个步骤。在获得文件的元数据后，才能根据元数据定位文件数据的存储位置，进行数据访问。所以，文件访问时间由元数据访问时间和数据访问时间组成，即：

$$\text{Time(file)} = \text{Time(metadata)} + \text{Time(data)},$$
$$\text{Ratio(metadata)} = \text{Time(metadata)} / \text{Time(file)}$$

在大文件应用中，由于一次元数据访问可以支持大量的数据访问，Ratio(metadata)很小，元数据的访问效率对请求的影响不很明显。但是，随着系统规模的不断扩大，大文件应用的聚合元数据请求数目也将非常可观，要求有效的元数据服务。比如，ASCI[Lustre-SGSRFP2001]要求文件系统能够支持 $1.8*10^7$ 个目录、$4.5*10^8$ 到 $1.0*10^{10}$ 个文件的文件系统规模，这将导致规模非常庞大的文件系统元数据请求。

同时，在企业应用环境中，web 服务、E-Mail、在线事务处理等不断涌现的新的应用需要通过分布式文件系统管理数据的存储和共享。这些应用的主要特征是数量庞大的小文件[Williams2005]，Ratio(metadata)将变大。并且，这些应用的规模扩展同样将增加元数据服务的压力。

所以，深入探讨分布式文件系统元数据服务的关键问题，满足已有应用不断增强的扩展需求，并支持特征多样的新应用，推动网络存储技术在更宽广领域的发展，具有非常积极的理论和实际意义。

本研究主要面向局域环境的网络存储系统，目标是通过集群方式的系统服务器架构，管理元数据的存储和访问，满足应用动态变化的分布式文件系统元数据服务需求。本研究主要关注提供分布式文件系统元数据服务的服务器部分的可扩展性，不包括客户端部分相关技术的研究。本研究的分布式文件系统元数据服务的扩展性包括：

1. 支持客户端数目增加带来的系统负载的增加。系统能够支持客户端聚合元数据请求吞吐率随客户端数目的扩展。
2. 支持在既定客户端数目的前提下，客户端请求的变化带来的系统负载的动态变化。系统需要支持客户端负载变化导致的聚合元数据请求吞吐率的扩展要求。

总之，分布式文件系统需要提供可透明扩展的元数据服务，满足系统服务器规模的动态扩展和应用的元数据服务需求的动态变化，支持数据的有效存储和共享。

1.2 相关技术的发展

1.2.1 存储设备

存储设备主要包括光盘、闪存、磁带和磁盘等[Hennessy2002]。光盘主要用于移动存储。由于其相对磁介质具有更长的使用寿命，可以用来更长时间的保存归档数据。但光盘容量有限，访问速度较慢，不适合用作在线存储设备。闪存是消费电子的移动存储，其容量很小。磁带作为磁盘的后端支持设备，主要用作归档数据的存储。磁盘是主要的在线存储设备，用来存储活跃的生产数据。磁盘技术发展非常迅速，根据Seagate®[Kryder2006]的数据，2006 年 3.5 英寸硬盘的存储能力达到 500GB，带宽达到 1,000Mb/s，读寻道时间为 8 毫秒。预计到 2009 年，3.5 英寸硬盘将拥有 2,000GB 的存储能力，2,000Mb/s 的访问带宽和 7.2 毫秒的读寻道时间。到 2013 年，这些指标分别为 8,000GB，5,000Mb/s 和 6.5 毫秒。磁盘性能的增强将弱化磁带在存储系统中的作用，将取代磁带成为归档数据的主要存储介质。

1.2.2 存储系统

从存储系统结构来看，服务器的存储系统经历了直连存储（Direct-Attached Storage, DAS），附网存储(Network-Attached Storage, NAS)和存储区域网络(Storage Area Network,

SAN)等三个阶段的发展[Morris2003]。如图 1.1 所示。

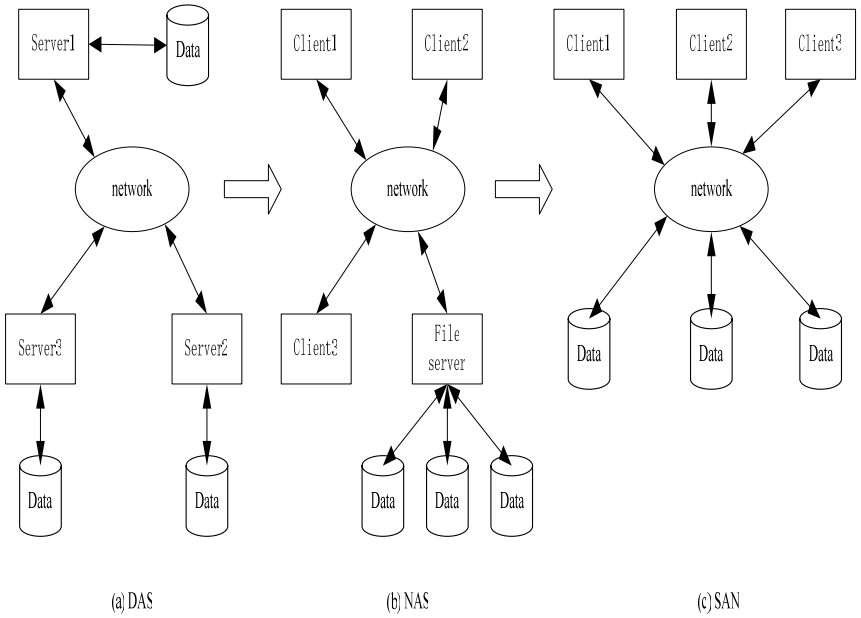

图 1.1 存储系统演进图

在 DAS 系统中，各个服务器通过 I/O 总线连接存储设备，并管理存储资源的使用。服务器不能直接使用其他服务器的存储设备，只能通过 FTP、SCP 等完成数据的共享。DAS 具有明显的缺点，包括：1）各个服务器独立管理其存储资源的使用，存储资源共享困难。2）存储设备通过 I/O 总线连接，扩展能力受限于 I/O 总线的能力。存储设备很难独立于服务器扩展，限制系统的扩展能力。3）存储资源扩展要求计算资源的同步扩展，导致系统管理成本的剧增。4）数据共享困难。

NAS 将 DAS 的存储设备集中，由专门的服务器管理，系统区分为客户端和文件服务器。各个客户端通过网络将数据读写请求提交给文件服务器，文件服务器转发客户端的数据请求。相对于 DAS 而言，NAS 能够解决存储和数据的共享问题。但文件服务器管理存储资源和转发数据读写请求的文件访问方式，将限制存储资源和数据读写性能的扩展。

SAN 使用高速网络直接连接存储设备，并通过虚拟化存储技术集中管理存储资源。客户端直接与存储设备读写数据，提高系统的存储和数据共享能力。为方便系统的管理，SAN 需要 SAN 文件系统[Menon2003]为数据共享控制提供支持。

1.2.3 分布式文件系统

分布式文件系统的研究开始于 20 世纪 80 年代，其典型代表包括 NFS[Sandberg1985][SUN1989][Callaghan1995][Shepler1999]和 CIFS[Hertel2003]等。从提供文件系统元数据服务的服务器结构看，分布式文件系统主要可以分为对称结构和非对称结构两大类[Welch2004]。

- 对称结构的分布式文件系统

在对称结构的分布式文件系统中，没有专门的服务器提供文件系统元数据服务，系统的所有客户端需要协同提供文件系统元数据服务。这类分布式文件系统主要有 DEC 的 VAXCluster[Kronenberg1986]、DEC 的 Frangipani[Thekkath1997]、RedHat 的 GFS [RedHat]、IBM 的 GPFS[Schmuck2002]、Veritas 的 GFS[Veritas]、Polyserve 的 Matrix [Polyserve]、Microsoft 的 FARSITE[Adya2002]和国家智能中心的 COSMOS[Shi2001] [Du2001]等。

对称结构的分布式文件系统要求所有客户端协同提供文件系统元数据服务。它隐含要求：1）所有的客户端同等对待。不论客户端是否需要访问文件系统，都需要参加文件系统的管理工作，在一定程度上导致计算资源的浪费；2）系统对单个客户端的依赖。客户端的性能将影响整个文件系统的性能，系统存在性能“短板”；3）数据访问的并发控制需要很多服务器参与，影响面很大；4）客户端间必须相互信赖，存在安全隐患。

- 非对称结构的分布式文件系统

非对称结构的分布式文件系统由专门的元数据服务器（Metadata Server, MS）提供分布式文件系统元数据服务，主要包括 CMU 的 Andrew File System[Morris1986] [Howard1987]、Duke 的 Slice[Anderson2000-1][Anderson2000-2]、IBM 的 StorageTank [Menon2003]、SGI 的 CXFS[Shepard2003]、ClusterFS 的 Lustre[Braam2002]、Panasas 的 ActiveScale[Panasas2003]、UCSC 的 Lazy Hybrid[Brandt2003]、HP 的 DiFFS[Zhang2001] [Karamanolis2001][Muntz2001-1][Muntz2001-2][Muntz2001-3]、国家高性能计算机工程技术研究中心的 BWFS[Yang2005]和国家智能中心的 DCFS2[Fan2004][Xiong2005]等。

系统可以由单个或多个服务器负责提供元数据服务。由于单个服务器有限的处理能力，多个元数据服务器将成为元数据服务的主流系统架构。在多个元数据服务器的集群环境中，根据服务器间的关系，存在 Active-Failover 和 Active-Active 两种结构。在正常情况下，Active-Failover 方式的系统只有 Active 的元数据服务器提供服务。当 Active 的服务器出现故障时，Failover 服务器接替其工作，继续提供服务。Active-Active 方式的系统则是多个服务器对等工作，服务器间能够做到相互冗余，提供高可用的元数据服务。

1.3 本研究的研究背景

本研究以蓝鲸网络存储系统（BlueWhale Networked Storage System）为平台，其结构如图 1.2 所示。通过蓝鲸集群文件系统（BlueWhale File System, BWFS）的协同，蓝鲸网络存储系统为应用提供文件级数据存储和共享。从图 1.2 的系统结构看，BWFS 采用带外数据（out-of-band）传输方式提供数据服务，用户文件访问的控制流和数据流分离。在获得文件的元数据后，用户直接通过网络块设备访问协议，并发地从存储设备读取数据。

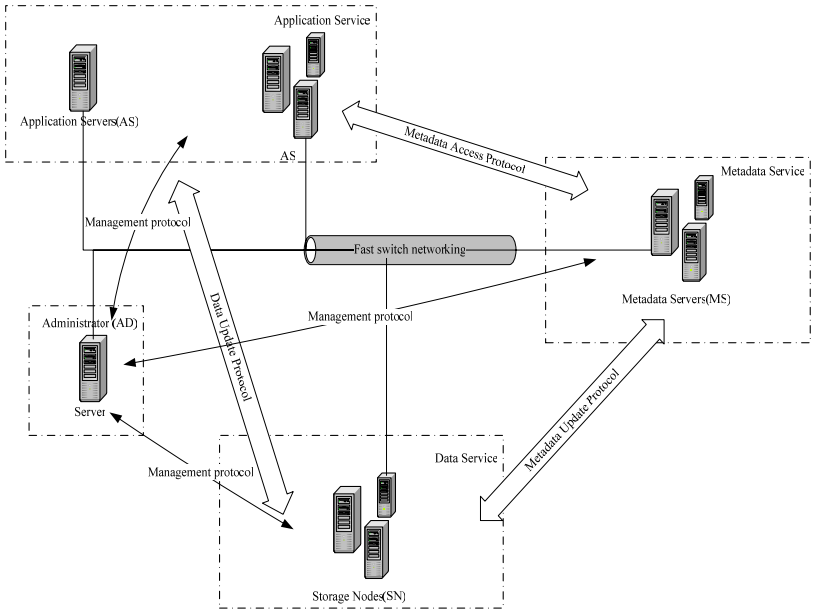

图 1.2 蓝鲸网络存储系统结构

从系统功能划分，系统由五个部分组成。第一部分是通过虚拟化存储技术完成系统存储资源的集中管理，提供存储服务的数据服务(Data Service)部分。它由完成存储资源虚拟化[Glider2003]的虚拟化控制器和多个直接连接到高速网络上的存储设备（storage node，SN）构成。第二个部分是提供文件系统元数据服务的元数据服务（Metadata Service）部分。它主要负责管理虚拟存储子系统提供的逻辑存储资源，并处理分布式文件系统的元数据请求。第三个部分是系统的管理服务器（Administrator，AD）。它通过多种方式监视和控制存储系统的状态，完成系统的单一集中管理功能。最后是应用服务器（Application Server，AS）。它支持多种硬件（x86，MIPS 等）、多种操作系统（Windows，UNIX，Linux 等）、多种应用（E-Mail 服务，文件服务，Web 服务，数据库服务，在线事务处理服务等）构成的异构环境。AS 与元数据服务、数据服务交互，完成文件的访问。这四个部分通过高速交换网络完成互联，构成统一的网络存储系统。

1.4 本研究的关键问题

本研究成果将用于图 1.2 的 BWFS 的元数据服务部分。从逻辑上，分布式文件系统元数据服务由元数据存储服务和元数据请求服务构成。所以，可扩展的分布式文件系统元数据服务需要解决的关键问题包括：

1. 存储资源的组织、管理和使用问题。存储资源的组织、管理和使用是元数据服务的基础问题。存储资源组织对象是存储文件系统元数据和数据的存储资源，其有效性要求能够以较小的代价支持存储资源动态独立地扩展。存储资源管理必需支持存储资源和资源使用者规模的扩展。存储资源的使用方式要求存储资源管理机制能够为资源用户间的共享提供有效支持。

2. 元数据请求的分布管理问题。在满足用户的元数据请求处理效率的前提下，尽可能地平衡各个服务器的负载，避免出现性能瓶颈。
3. 文件系统的一致性问题。元数据请求分布策略可能将更改文件系统请求涉及的多个元数据分布到不同服务器，请求成为跨服务器请求。尽管跨服务器请求的比例非常少[Hendricks2006]，但两阶段提交等传统的请求原子性保证协议，对系统的性能和错误恢复能力的影响都非常大，有必要探讨轻量级协议的可能性。
4. 元数据请求的并发控制问题。在新的存储系统架构下，需要探讨已有的并发控制算法的适应性。分布式文件系统需要代价较小的请求并发控制算法，满足元数据服务扩展的需要。

1.5 本研究的主要贡献

通过分析已有的研究成果，结合用户的元数据访问负载的特点，本研究获得了高扩展能力的存储资源组织、管理和使用，动态高效的元数据请求分布管理，简单有效的文件系统一致性协议，以及高效的元数据请求并发控制算法等研究成果。具体包括：

1. 基于集中虚拟存储模型的存储资源组织、分布式层次化的存储资源管理、完全共享的存储资源使用模式的文件系统元数据存储服务。虚拟存储支持存储资源的独立扩展，层次化的管理机制支持存储资源的有效利用，完全共享的存储资源使用模式解决非常困难的数据放置问题，支持动态灵活的元数据请求服务。在此基础上，元数据的存储和访问彻底分离，元数据的存储不再限制其请求的分布，元数据的存储问题得到有效解决。
2. 动态灵活的元数据请求分布管理机制和策略，支持元数据服务器规模和用户元数据访问负载的动态变化。通过仅管理活跃元数据的请求分布，大幅度缩小请求分布管理的对象集，支持请求分布管理的扩展。根据元数据活跃性管理元数据请求分布，为用户元数据访问负载的动态变化提供有效支持。采用层次化的分布管理机制，支持元数据服务器规模的动态扩展。元数据请求分布策略结合请求的动态性和元数据间客观存在的访问相关性，结合元数据请求的处理效率，尽可能地平衡服务器的负载。
3. 简单高效的更改文件系统的跨服务器请求的原子性保证协议。对称的元数据服务器系统结构、用户元数据访问的统计特征、动态灵活的元数据请求分布管理，为简单高效的元数据请求一致性协议提供足够的支持。通过动态改变活跃元数据在服务器间的分布，完成元数据在服务器间的迁移，将跨服务器请求集中化，利用相关技术保证文件系统一致性，减小协议对系统正常请求处理的影响。
4. 基于系统的请求并发情形分析，提出简单的请求同步控制算法。利用元数据请求分布信息，结合元数据请求语义，充分分析可能的请求并发，根据请求的优

先级进行判定，尽可能地并发元数据请求。

5. 根据上述结果，设计和实现了 1 个应用到 BWFS 的原型系统（蓝鲸多元数据服务器系统，BWMMS），并通过实验验证其扩展能力的有效性。

相对于传统分布式文件系统，BWMMS 的元数据服务基于集中共享的虚拟存储模型，通过动态重分布方式管理活跃元数据的请求分布，能够很好地满足元数据存储和请求服务的扩展需求，支持应用请求负载的动态变化。针对仅约为 0.006%的跨服务器元数据请求[Hendricks2006]的原子性保证，BWMMS 通过动态元数据分布管理机制的支持，改变元数据请求的分布，将元数据集中到单个服务器处理。动态迁移协议不仅在协议延迟上优于传统的协议，同时也在系统的错误恢复上占有一定的优势。

1.6 本文的组织

本章是第一章，引言部分。介绍本研究的目的和意义，相关技术的发展，研究背景和主要贡献，以及本文的组织等。

元数据服务包括元数据存储和元数据访问两个逻辑层次。第二章探讨元数据存储的关键问题，包括存储资源的组织、管理和使用。其目标是解决存储资源管理的可扩展性问题，并为元数据请求服务提供必要的支持。通过集中虚拟存储模型组织系统存储资源，层次化机制管理存储资源的使用，元数据服务器间完全共享的元数据访问方式，元数据的存储问题得到解决，为灵活的元数据请求服务提供支持。

第三章探讨有效的元数据请求分布管理问题。由于用户元数据访问表现出活跃性、动态性和局部性等特点，BWMMS 采用“后端集中决策机制”，结合服务器的负载、元数据的活跃性、文件系统的名字空间结构，动态管理元数据请求的分布。元数据访问的局部性决定元数据分布信息缓存的有效性。第四章探讨元数据分布信息缓存管理的关键问题。

更改文件系统的跨服务器请求的原子性保证是文件系统一致性的关键问题。由于传统的原子性保证协议在性能和错误恢复等方面存在不足，第五章探讨轻量级原子性保证协议的可能。通过元数据的迁移，分布式元数据请求集中到单个服务器处理，借助本地文件系统技术保证文件系统的一致性。动态迁移协议的效率和容错能力同样得到分析。

元数据迁移加剧请求同步问题。第六章介绍 BWMMS 通过分离元数据分布信息管理和文件系统元数据管理的同步机制，根据元数据访问协议分析可能的并发情形，通过判定请求的优先级，尽可能地并发元数据请求。

第七章介绍了系统的原型实现及评估。第八章是本研究工作的总结和下一步的研究建议。本文最后是参考文献、致谢和作者简历。

第二章 文件系统元数据存储服务

逻辑上，文件系统元数据服务由元数据存储和元数据访问构成。元数据存储服务是元数据请求服务的基础。如何合理有效地组织、管理和使用系统的物理存储资源是元数据存储服务的关键问题。本章讨论 BWMMS 的基于集中虚拟化存储技术的物理存储资源组织、分布式层次化的存储资源管理和完全共享的存储资源使用的元数据存储服务，有效解决存储资源的管理问题，为元数据请求管理提供坚实的基础。

2.1 元数据存储服务需求

从系统的逻辑结构来看，文件系统元数据服务包括提供文件系统元数据存储的元数据存储服务、提供文件系统元数据访问的元数据请求服务。元数据请求服务以元数据存储服务为基础，通过元数据存储服务完成元数据的存储和访问管理。文件系统元数据存储服务是文件系统元数据服务的基础问题。

存储资源的组织结构为存储资源的有效管理提供支持。只有通过合理的存储资源组织，才能有效地管理和使用系统的存储资源。在大规模系统环境中，存储资源管理和使用的参与者规模非常庞大，需要通过有效的存储资源管理机制管理，避免出现限制系统扩展的瓶颈。存储资源的使用模式是应用有效共享文件系统元数据的保证。只有通过有效的存储资源使用模式支持，存储资源用户间才能以较低的代价实现元数据的有效共享，提高元数据共享的效率。

所以，如何有效地组织异构的存储资源，加以有效的管理和使用，提供具有较强扩展能力的存储服务是文件系统元数据存储服务的关键问题。只有有效解决存储服务的关键问题，为文件系统元数据服务提供具有较强扩展能力的元数据存储服务，为有效解决元数据请求服务的关键问题提供基础，才能有效地提供文件系统元数据服务，提高文件系统元数据服务的扩展能力。

2.2 相关研究概况

已有研究在存储资源的组织、管理和使用方面进行了大量的研究。物理存储资源的组织结构主要存在分散和集中两种，如图 2.1 所示。

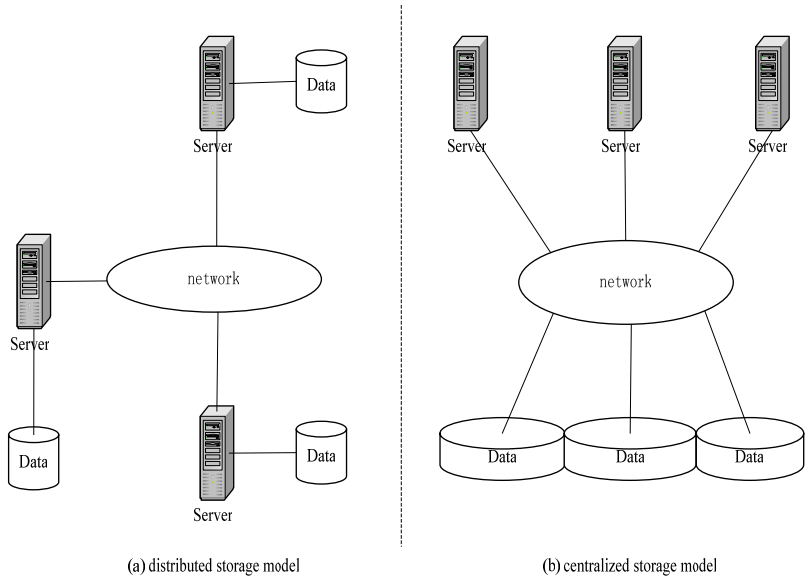

图 2.1 两种主要的存储资源组织结构

在分散的物理存储资源组织结构[Kronenberg1986][Xiong2005-1]中，存储设备和服务器紧耦合在一起，每个服务器通过其 I/O 总线连接一定数量的存储设备，多个服务器之间通过网络完成互联。这种组织结构的好处是，只要服务器支持某种方式的访问，就能很容易地加入系统，且文件系统元数据请求处理的服务器分配策略非常简单，存储位置决定请求服务器。其主要的问题包括：1）存储设备通过 I/O 总线连接到服务器，单个服务器能够支持的存储资源扩展有限；2）由于服务器和存储设备的紧耦合关系，存储设备和服务器不能彼此独立地扩展，容易导致资源的浪费；3）服务器独立管理存储资源，系统资源不容易得到充分利用；4）数据很难充分共享；5）元数据存储位置决定请求服务器，请求分布缺乏灵活性。

集中的物理存储资源组织结构集中管理系统的存储资源，并统一调度资源的使用，能够充分利用系统的存储资源。DEC Petal[Lee1996]、UCSC 的 Lazy Hybrid[Brandt2003][Weil2004]和清华大学的 Dynamic Hashing[Li2006]等，都采用这种组织结构。

存储资源管理机制将影响存储资源管理的有效性。单一集中的存储资源管理机制[Kronenberg1986][Anderson1995][Callaghan1995][Xiong2005-2]集中管理存储资源的分配和释放。其优点是系统资源管理结构简单，请求处理的路径较短，单个的资源请求的处理效率较高。但是，这种结构不能很好满足大规模系统频繁的资源请求，需要合理分布资源管理功能，提高其扩展能力。

存储资源的使用模式将决定系统的数据共享方式，影响元数据请求管理策略。私有使用模式[Morris1986][Xiong2005-1]将存储资源与存储用户紧耦合，两者间形成从属关系。各个用户独立管理私有的存储资源，请求的重新分布过程，要求通过服务器转发、

或数据在存储资源的迁移等才能完成，请求在服务器间的重分布较难完成。在[Yan2004]的互斥使用模型中，存储资源划分成多个可独立装载(mount)的文件系统。同一时刻，每个文件系统只能被一个用户装载。请求重分布过程要求原用户卸载(umount)文件系统，新用户再重新装载。私有模式和互斥使用模式的服务器形成非对称的服务器结构，请求的分布策略很难灵活支持请求服务器规模的扩展。共享使用模式[Brandt2003][Weil2004][Yang2005][Li2006]通过有效的共享控制机制支持，用户的数据共享不需要数据的迁移、或者文件系统的卸载/装载过程，服务器间形成对称的结构。数据可以放置在任意存储位置，请求可以分配到任何服务器完成处理。

2.3 BWMMS 的元数据存储服务

BWMMS 的元数据存储服务，通过虚拟化存储技术集中组织系统的物理存储资源，存储设备形成对称的结构，元数据的存储位置不再影响其访问请求的分布。存储资源管理功能分散到多个部分，形成层次化的资源管理机制，适应系统的规模扩展。元数据采用 64 位的逻辑元数据资源号标识，逻辑元数据资源动态分配、并动态映射到物理存储资源，有效支持存储资源的扩展。文件系统不存在分割，请求服务器可见任意元数据，服务器间形成对称的结构，为元数据请求在服务器间的任意分布提供基础。

2.3.1 集中虚拟化的存储资源组织

已有的采用集中存储资源组织的系统，通常由各个存储资源用户直接管理物理存储资源。当系统规模非常庞大时，存储资源管理任务非常繁重，用户需要为存储资源管理做出很大的开销[Glider2003]。

为了提高系统的扩展能力、容错能力和可管理能力，增强系统各个部分的模块化，BWMMS 采用虚拟化存储技术[Morris2003][Glider2003]管理系统的物理存储资源。它在系统的存储资源提供者和存储资源使用者之间加入存储资源虚拟化管理层（Storage Virtualization Layer，SVL），物理存储资源管理任务从存储资源使用者剥离，由 SVL 独立完成系统的物理存储资源管理。系统的物理存储资源组织和管理形成独立的存储子系统。除了管理物理存储资源，SVL 还需要建立和管理物理存储资源与逻辑存储资源之间的映射关系。对存储资源用户而言，系统的物理存储资源表现为可以通过某种寻址方式访问的逻辑存储空间。

由于 SVL 集中管理系统的物理存储资源，并且物理存储资源的任何变化将被 SVL 屏蔽，系统的存储资源管理具有较强的扩展能力。并且，存储资源用户的加入和退出不需要与大规模的物理存储资源直接进行交互，用户的规模扩展同样得到支持。

通过集中虚拟化的存储资源组织，BWMMS 面对的存储资源是通过虚拟化存储技术提供的逻辑存储空间，它采用 64 位的逻辑存储资源号标识，形成线性存储地址空间。

2.3.2 分布式层次化的存储资源管理

存储资源管理机制是存储服务的重要内容。集中的存储资源管理容易导致系统瓶颈出现，不能满足大规模系统的存储资源管理需要。层次化管理结构在存储资源的提供者和使用者间形成明显的层次关系[Huang2005]，将资源管理功能分散到各个层次，消除可能的瓶颈限制。

BWMMS 通过层次化机制管理元数据存储资源，如图 2.2 所示。为避免瓶颈限制，逻辑资源管理功能分散，层次之间通过批量方式管理资源的分配和释放，层次间有效的缓存机制，降低层次化导致的资源请求时间延迟变长问题。

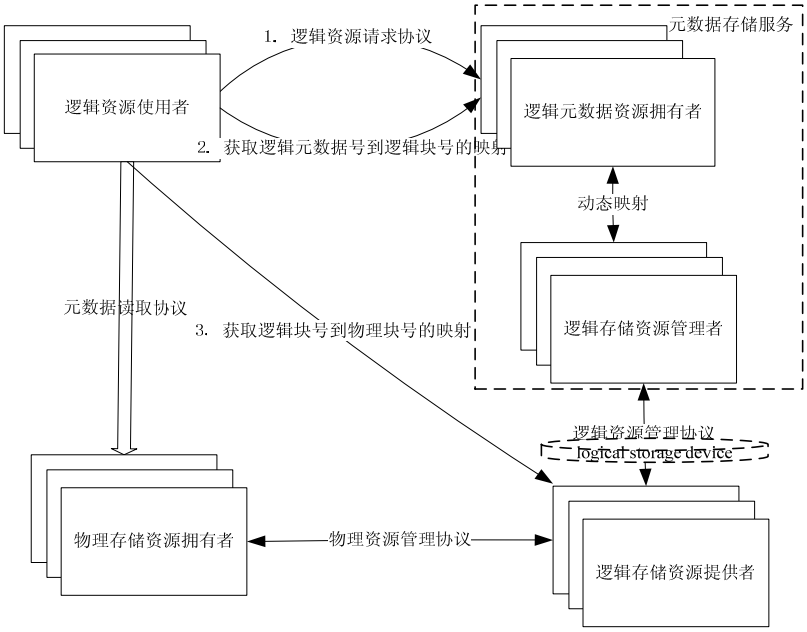

图 2.2 分布式层次化的存储资源管理

元数据存储服务需要管理的逻辑资源包括索引节点、存放元数据的间接块、目录的数据块等，逻辑元数据资源采用 64 位的逻辑元数据资源号标识，支持文件系统规模的扩展。文件系统逻辑元数据资源访问需要经过“逻辑元数据资源〈——〉逻辑存储资源〈——〉物理存储资源”的映射过程。“逻辑存储资源〈——〉物理存储资源”由存储虚拟化层完成，支持物理存储资源的扩展、逻辑存储资源与物理存储资源的动态映射等。为使用扩展的逻辑存储资源，BWMMS 通过动态分配和动态映射方式完成“逻辑元数据资源〈——〉逻辑存储资源”的映射。

逻辑存储资源管理者以批量方式从逻辑资源提供者获取可用的逻辑存储资源信息。元数据的动态分配由逻辑资源使用者驱动，逻辑资源使用者通过元数据访问协议，驱动逻辑元数据资源拥有者分配元数据。逻辑元数据资源拥有者从 64 位线性的逻辑元数据资源进行分配，并动态建立分配的逻辑元数据资源与逻辑存储资源的映射关系。元数据访问通过动态映射完成物理存储资源的定位。逻辑资源使用者首先从逻辑元数据资源拥有

者获得动态建立的逻辑元数据资源与逻辑存储资源的映射关系，然后从逻辑资源提供者获得逻辑存储资源与物理存储资源的映射关系，最后访问物理资源提供者。

逻辑元数据资源的释放同样由逻辑资源使用者动态驱动。逻辑元数据资源拥有者解除逻辑元数据资源和逻辑存储资源的映射，记录可用逻辑元数据资源，释放可用逻辑存储资源给逻辑存储资源管理者，逻辑存储资源管理者以主动或者被动的方式将可用逻辑存储资源释放给逻辑存储资源提供者。

为平衡多个逻辑元数据资源管理者间可用的逻辑元数据资源，逻辑元数据资源拥有者之间通过主动或者被动方式，交换可用逻辑元数据资源信息。所有的逻辑元数据资源拥有者的可用逻辑元数据资源交集为空。逻辑元数据资源可以从任意逻辑元数据资源拥有者分配，在任意逻辑元数据资源拥有者释放，资源管理不会出现问题。逻辑存储资源具有同样的性质。这为元数据请求的灵活分布提供基础。

2.3.3 完全共享的存储资源使用

存储资源使用模式指的是用户对存放在存储资源的数据的共享方式。私有存储将数据与请求服务器紧耦合，数据的访问请求不能灵活地分布到请求服务器。它通常只能通过显式的方式完成数据的共享，存储资源很难共享。已有的某些采用集中方式组织物理存储资源的系统，存在与私有存储相似的逻辑结构。它将逻辑存储空间划分成子空间，每个子空间对应可以独立装载的文件系统。同一时刻，一个子文件系统只能被一个用户装载，子文件系统的使用是互斥的。当用户需要访问其他子文件系统时，它首先要求装载该文件系统的用户卸载，然后自己装载该文件系统，进行请求的处理。其本质依然是通过数据的存储位置决定处理请求的服务器，同样不能很好支持请求服务器规模的扩展。

BWMMS 采用单一的逻辑元数据资源空间，由单一文件系统组织和管理统一的线性逻辑元数据资源空间。出于支持不同特征应用的需要，线性元数据资源空间可以划分成不同使用特性的区域。但是，与私有存储结构和集中互斥存储架构的存储资源使用模式的本质不同是，它没有任何请求分布限制。在逻辑上，存储资源不与请求服务器关联，所有请求服务器装载同一个文件系统。在文件系统的并发控制下，每个请求服务器可见、并能访问文件系统任意的逻辑元数据资源，用户对存储资源的访问不需要请求其他用户授予权限，这种系统称为“集中共享存储架构”。

综合文件系统元数据存储服务的关键问题，BWMMS 采用基于虚拟存储技术的集中化存储资源组织、分布式层次化的存储资源管理和完全共享的存储资源使用，元数据服务器形成对称的服务器结构，如图 2.3 所示。

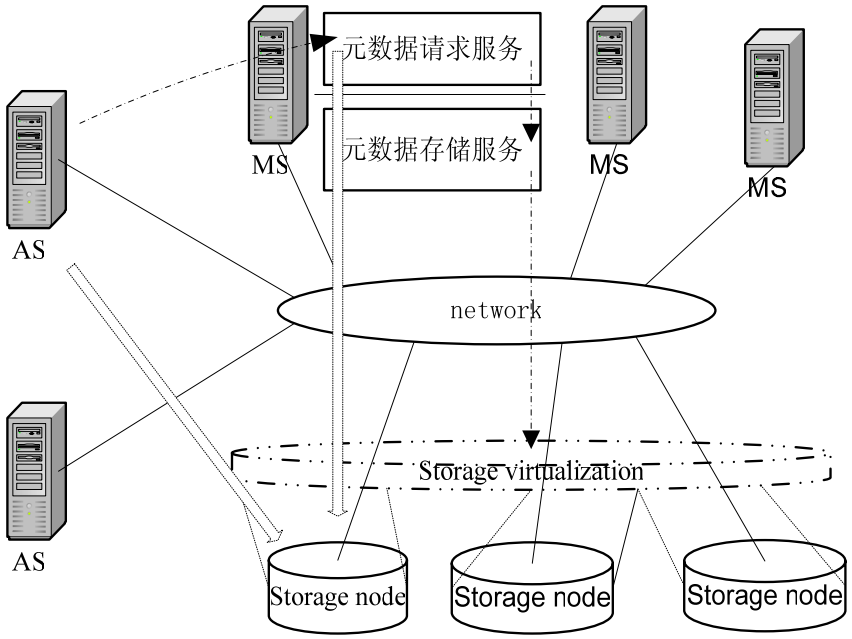

图 2.3 BWMMS 基于集中共享虚拟存储的对称服务器结构

结合图 2.2，存储节点（Storage Node，SN）是提供物理存储资源的物理资源拥有者，它将自己管理的物理存储资源提供给存储虚拟化层（Storage Virtualization Layer, SVL）管理，SVL 是图 2.2 中的逻辑资源提供者，它管理物理存储资源和逻辑存储资源的映射，提供统一的逻辑存储资源空间给逻辑存储资源管理者。元数据服务器（MS）充当逻辑存储资源管理者和逻辑元数据资源拥有者的角色，管理从 SVL 获得的逻辑存储资源，批量的、主动和被动结合的逻辑存储资源分配和释放发生在 MS 和 SVL 之间。AS 是系统的逻辑资源使用者，AS 的元数据请求语义驱动 MS 完成逻辑元数据资源、逻辑存储资源的分配和释放。

在 BWMMS 的集中共享存储架构中，元数据的存储和访问彻底分离。元数据可以存放在任何存储位置，元数据请求也可以分配给任何请求服务器。在此基础上，元数据服务器形成对称的服务器结构，元数据请求可以分布到任何服务器，消除了私有存储资源使用模式中、为平衡请求处理负载而带来的非常困难的元数据存放策略问题，也为灵活的元数据请求分布管理提供坚实的基础。

2.4 本章小结

文件系统元数据服务由元数据存储服务和元数据请求服务构成，元数据存储服务是文件系统元数据服务的基础。元数据存储服务的存储资源组织、管理和使用机制将影响元数据请求服务的有效性，本章对有效的元数据存储服务进行了探讨。

基于业界成熟的虚拟化存储理念，BWMMS 集中组织物理存储资源，并以 64 位的逻辑存储资源标识为系统存储资源扩展提供支持。通过集中化和虚拟化，存储资源能够得到有效利用，存储资源管理的扩展能力得到提高。

BWMMS 提出分布式层次化的存储资源管理机制。层次化将系统存储资源管理功能层次化、模块化，提高各个部分的独立性。分布式的存储资源管理将存储资源管理功能分散，消除存储资源管理的瓶颈。64 位的元数据逻辑资源标识为文件系统扩展提供支持，元数据的动态分配和动态映射机制，为文件系统支持逻辑存储资源的扩展提供基础。

以集中虚拟化存储和分布式层次化存储资源管理机制为基础，BWMMS 提出统一的逻辑文件系统名字空间视图，以完全共享的使用模式管理用户的存储共享。通过完全共享方式，元数据的存储与元数据的访问形成松耦合关系，元数据的存放位置不再限制处理元数据请求的服务器。元数据可以存放在存储资源的任意位置，并将其请求指定给任意服务器处理，服务器间形成对称的结构。共享的存储资源使用模式，有效地消除了私有使用模式中、为平衡请求处理负载而存在的非常困难的元数据放置问题，为灵活的元数据请求服务提供坚实的基础。

第三章 元数据请求分布管理

元数据请求服务构建在元数据存储服务基础上，响应用户的元数据请求。它需要在保证单个元数据请求处理效率的前提下，综合考虑元数据服务器的负载，避免因服务器负载热点而限制元数据服务的扩展。元数据请求服务的核心是如何在元数据服务器间有效分布元数据请求。应用的元数据请求表现出动态变化的特征，如何满足用户动态的元数据服务需求是元数据请求分布管理的关键。

本章主要讨论元数据请求分布的有效管理问题。首先介绍元数据访问协议集合，元数据访问的统计特征和相关研究的元数据请求分布管理。然后介绍 BWMMS 的动态元数据请求分布管理的机制和策略，包括请求分布的对象、请求分布的管理机制和协议、请求分布的策略等。最后通过实验评估 BWMMS 的元数据请求分布管理的有效性。

3.1 文件系统元数据访问协议

元数据请求分布管理需要明确分布管理的元数据请求类别，以确定请求分布决策时机，并根据元数据请求的类别加以区别地对待请求的分布。

文件系统元数据访问协议包括文件系统元数据的访问请求和文件元数据的访问请求两大类[Opengroup.org][SMB1987][SUN1989][Callaghan1995]。

文件系统元数据的访问请求包括查询文件系统状态，在文件系统名字空间中查找、创建、移动或删除文件等请求。

查询文件系统状态的请求主要访问文件系统超级块的内容，获取文件系统的资源使用情况，文件系统的统计信息等状态信息。

查找文件的元数据请求用来明确文件系统的名字空间是否存在指定的文件，其格式为（父目录，文件名），期望返回被查询文件名的索引节点。在本地文件系统中，查找文件的元数据请求隐含在文件名解析过程中。在分布式文件系统中，为支持异构的元数据服务器，并减少协议的通信开销，文件查找需要显式地进行，返回被查找文件的索引节点。目录内容读取请求获取目录一定范围内的目录项，其参数格式为（目录，开始位置，读取内容长度）。文件查找和目录内容读取不更改任何元数据，且其访问统计比例很高。

创建文件类元数据请求需要在文件系统名字空间加入新的文件、或者指向已经存在的文件的目录项，包括创建文件、符号连接和硬连接等。文件的创建将在目录下生成指定名字的新文件，请求格式为（父目录，文件名，文件类型）。“文件类型”是 Unix 文件系统支持的普通文件、目录和特殊设备节点等。创建符号连接的参数格式为（父目录，符号连接名，目标文件）。目标文件是被连接的文件名字，它可以是普通文件，也可以是

目录。创建文件和符号连接都需要分配新的索引节点，并在父目录中添加指向该索引节点的目录项。创建硬连接的参数格式为（父目录，硬连接名，目标文件名字）。与文件和符号连接创建不同的是，硬连接创建没有索引节点的分配，且目标文件只能是普通文件。它根据目标文件名获得索引节点号，在父目录中添加指向该索引节点的目录项，并更改目标文件的索引节点。

移动文件的元数据请求将文件（旧文件）从一个目录（源目录）移动到另一个目录（目标目录），同时可能将文件名字改成新的文件名字（新文件）。文件移动请求首先需要在目标目录中添加目录项，指向新文件。如果目标目录中存在与新文件同名的文件（同名文件），需要在添加指向旧文件的目录项之前，处理目标目录与同名文件之间的连接。在目标目录的处理完毕后，在源目录中删除指向旧文件的目录项，并更改旧文件的索引节点。文件移动操作最多将更改 4 个元数据，是最复杂的元数据请求。重命名文件请求是移动文件请求的特殊情况，它完成同一个目录下的文件移动。

删除文件的元数据请求完成目录和文件的连接的删除，其参数格式为（父目录，文件名）。它需要删除父目录中的目录项，更改被删除文件的索引节点。

针对文件的元数据访问请求主要包括读写文件的索引节点、获取文件数据所在的存储设备块号和读写权限等，这些请求仅需要对单个文件的元数据进行访问。

3.2 用户元数据访问特征

根据已有的研究结果，用户的元数据访问的主要特征可以概括为：

1. 聚合元数据请求数目所占比例非常大。

表 3.1 已有研究结果的元数据请求数目比例

	Total	Data	Metadata	Metadata%
NFS	6,680,000	1,640,000	5,040,000	75.45
AFS	1,615,500	285,500	1,330,000	82.33
Sprite	432,000	264,000	168,000	38.89
INS	8,300,000	2,470,000	5,830,000	70.24
RES	3,200,000	374,000	2,826,000	88.31
NT	3,870,000	1,501,000	2,369,000	61.21
CAMPUS	29,900,000	25,220,000	468,000	15.65
EECS	2,300,000	788,000	1,512,000	65.74

如表 3.1 所示(其中 Total，Data 和 Metadata 列的单位是次，表示请求次数)，在软件开发、Web 服务和 E-Mail 服务等应用中[Ousterhout1985][Baker1991][Dahlin1994-1][Gibson1997][Roselli2000][Ellard2003]，元数据请求数目的比例比较高，最多达到近 89%。

2. 从文件系统整体而言，文件系统元数据表现出部分活跃性[1]。

[1] 活跃文件指的是创建完成后，还会被再次访问的文件。

文件系统大部分的文件和数据不会被经常重复地访问，活跃文件和数据占整个系统的比例较少。据统计结果，从文件数目角度统计，活跃文件的比例大致为 4%-20%。从活跃的数据比例看，活跃数据的比例大致为 10%-30%[Ramakrishnan1992][Williams2005]。

3. 单个用户的元数据请求表现出局部性、突发性和动态变化的特征。

在一定时间内，单个用户的文件访问往往集中在某些目录下[Floyd1989][Agrawal2007]，不会发散到整个文件系统。单个用户的文件访问表现出随时间变化的突发性和动态性[Baker1991][Roselli2000][Vogels1999][Ellard2003]。

4. 不同的元数据请求的统计频率不同。

元数据请求存在不同的统计频率[Bozman1991][Vogels1999][Hsu2001][Ellard2004]，针对文件的元数据请求和文件查找的请求所占比例超过 95%，而需要更改多个元数据的文件创建、删除和移动等元数据请求所占比例不足 5%。

5. 元数据间在访问上存在关联性。

根据元数据访问协议，查找文件、创建文件、删除文件和移动文件等元数据请求都需要同时访问多个元数据。为保证系统元数据请求的聚合吞吐率，元数据服务器的负载平衡应该以保证访问统计频率较高的元数据请求的处理效率为前提。

所以，元数据请求分布策略应该综合考虑用户元数据访问的局部性、动态性和访问相关性，结合元数据访问协议，根据“最频繁的请求最快完成”原则，主要关注针对单个文件的元数据请求、查找文件的元数据请求的处理效率，并为其他元数据请求的正确处理提供必要的支持。

3.3 相关研究概况

现有的分布式文件系统元数据请求分布管理主要可以分为两大类：第一类是目录子树分区法，如 LOCUS[Popek1986]、AFS[Morris1986]、CODA[Satyanarayanan1990]、Sprite[Ousterhout1988]等。它依据文件系统名字空间结构，将文件系统目录树划分成目录子树，单个元数据服务器处理一个目录子树的所有元数据请求，文件的创建、删除和移动等元数据请求只能在同一目录子树的范围内进行。当各个目录子树的元数据请求差异很大时，元数据服务器负载将很难平衡。为尽可能地平衡服务器的负载，解决元数据服务随文件系统目录子树深度扩展的问题，动态目录子树分区法[Weil2004]动态收集元数据服务器的负载。当元数据服务器负载超过预定值时，将原来由该元数据服务器管理的目录子树，迁移到其他元数据服务器管理。由于以粒度非常大的目录子树为请求分布单位，其迁移过程对系统的影响较大。

第二类是哈希法。哈希法以文件名等静态的文件系统信息为哈希函数的参数，根据哈希函数计算结果决定负责元数据请求处理的元数据服务器。这类系统包括 Vesta[Corbett1996]、InterMezzo[Braam1999]、RAMA[Miller1997]、Lustre[Braam2002]和

Dynamic Hashing[Li2006]等。仅仅通过哈希函数的计算，不需要根据文件系统目录树遍历，哈希法就能够定位负责元数据请求处理的服务器，可以提高文件查找等请求的处理效率。但是，哈希法存在明显不足，包括：（1）静态设计的哈希函数很难支持系统的元数据服务器规模的动态扩展；（2)哈希函数的参数选择对元数据服务器负载的影响很大，设计不好的哈希函数可能将元数据请求集中到部分元数据服务器；（3）哈希法较难对文件访问的逻辑性提供支持。统计结果表明，单个用户在一定时间内的访问经常集中在几个目录。但是，哈希法可能将父目录和子文件分布到不同的服务器，加大分布式元数据请求的可能性；（4）当哈希参数发生变化，需要重新分布元数据请求时，需要大量的元数据请求分布信息更新操作和元数据迁移操作来适应新的元数据分布结果。

总之，目录子树分区法和哈希法主要以文件系统的静态信息为参数，完成元数据请求分布决策。它们有自己的优点，但无法很好地支持用户元数据请求的动态变化。

3.4 BWMMS 元数据请求分布管理

根据文件系统元数据的部分活跃和单个应用元数据访问的局部性特征，BWMMS 仅管理应用当前访问的活跃元数据的请求分布。同时，BWMMS 区分对待目录和常规文件的请求分布，在考虑相关性的同时，结合服务器的负载情况，动态决定元数据请求的分布。结合元数据请求的访问频率，保证频率高的请求尽可能快地完成。

本节首先定义后续论述中需要的相关概念，然后讨论请求分布管理的机制、协议和分布决策算法。

3.4.1 相关概念定义

元数据请求分布以文件索引节点为最小单位进行。当负责请求处理的元数据服务器确定后，与文件索引节点紧密相关的元数据也由该服务器负责读写。

在定义元数据请求分布管理的重要概念前，首先需要明确与管理相关的对象。

- 文件系统信息集合。集合 B 表示文件系统当前所有信息的集合，即

$$B = \{b \mid b\text{是文件系统的信息}\},$$

- 文件数据信息集合。集合 D 是普通文件的数据，则

$$D = \{d \mid d \in B, d\text{是普通文件的数据}\},$$

- 元数据信息集合。集合 M 是文件系统的元数据集合，则

$$M = \{m \in B \wedge m \notin D\},$$

集合 B，集合 D 和集合 M 间的存在的关系为：

$$M \cup D = B, M \cap D = \varnothing。$$

- 工作元数据服务器集合。集合 S 为系统当前所有提供服务的元数据服务器，即

$$S = \{s \mid s\text{是当前提供元数据服务的元数据服务器}\}\text{。}$$

根据以上集合的定义，我们将定义活跃元数据、元数据请求分布映射、元数据宿主和单一映射等概念。

定义 3.1 活跃元数据。活跃元数据（Active Metadata, AM）是文件系统当前被访问的元数据集合。任何当前被访问到的文件索引节点和其关联的元数据信息、该索引节点文件名路径前缀上的所有目录索引节点和其关联的元数据都是活跃元数据。

$$AM = \{am \mid am \in M \ \wedge \ am\text{正在被访问}\}$$

定义 3.2 元数据请求分布映射。为活跃元数据指定负责元数据请求处理的元数据服务器的过程，称为“元数据分布映射”。

$$F = \{f \mid f =< am, s >,\ am \in AM,\ s \in S\}$$

请求分布映射是活跃元数据和元数据服务器间存在的一种动态关系，根据元数据的活跃性动态建立。元数据第一次被访问而变成活跃时，将完成请求分布映射关系的建立。在元数据不活跃时，请求分布映射关系将被解除。

定义 3.3 元数据宿主。当前负责元数据请求处理的元数据服务器称为“元数据的宿主”，表示为 HOST(am)。

$$s \ = \ f(am),\ am \in AM, s \in S$$

定义 3.4 活跃元数据的单一映射。在同一时间，任意活跃元数据有且仅有一个元数据宿主。其结果是活跃元数据在元数据集群中仅有一份拷贝，可以简化元数据更新的同步问题。

$$\forall s1, s2 \in S,\ am \in AM,\ s1 \ = \ f(am),\ s2 \ = \ f(am) \ \Rightarrow s1 \ = \ s2$$

定义 3.5 服务器负载。服务器负载（WorkLoad, WL）是通过多个因素以一定的权值比例构成的，综合反映服务器当前的压力。

$$WL = \sum_{i=1}^{n} W_i P_i,\ \text{其中} \sum_{i=1}^{n} W_i = 1$$

3.4.2 分布管理机制

元数据请求分布管理机制主要包括分布式决策和集中式决策两大类[Braam2002][Anderson2000-1][Brandt2003]，如图 3.1 所示。

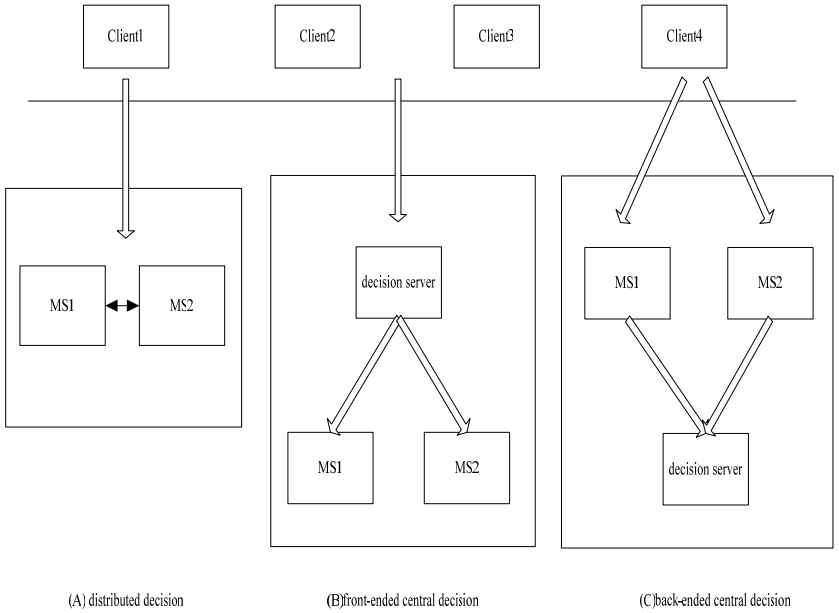

图 3.1 三种元数据请求分布管理机制

分布式决策系统的各个元数据服务器具有很大程度的自治权限。在服务器负载没有超过预先设定值时，各个服务器独立管理元数据请求的分布。当服务器负载超过设定值后，它需要与系统所有的服务器通信，协同完成元数据请求的分布决策。从结构上讲，分布式决策系统不存在单一服务器限制系统扩展的可能。但是，其采用的某些集中控制机制同样可能限制系统的扩展能力。比如，请求的分布决策要求服务器维护大量的决策信息，并公布分布决策结果。并且，由于缺乏统一的调度，服务器间的负载很难平衡。

与分布式决策不同的是，集中式决策采用专门的服务器，集中决策整个系统的元数据请求分布。集中式决策系统的各个元数据服务器为决策服务器提供必要的决策信息，由决策服务器集中决定元数据请求的分布。集中决策可以减轻各个服务器的分布决策开销，并能够适应服务器负载的变化，以统一简洁的方式完成请求的分布决策。各个服务器根据需要，向决策服务器查询请求分布现状，每次分布决策结果不需要广播。

按照决策服务器所处的位置，集中式决策系统可以分为“前端集中决策”和“后端集中决策”。由于元数据访问的局部性特征，后端集中决策将通过在元数据服务器缓存元数据分布结果，将必需决策服务器完成的处理变成可选，大幅度降低决策服务器的负载，提高系统的扩展能力。

BWMMS 基于“集中决策，分布处理”的原则，采用后端集中决策机制完成文件系统元数据请求的分布决策，系统的集中决策点位于元数据请求处理路径的末端。元数据服务器彼此间不需要为完成元数据请求分布决策通信，他们仅需要与集中决策服务器交互。在大多数的情况下，元数据服务器可以根据自己缓存的元数据分布信息，完成请求的处理。在必要的时候，它与集中决策服务器交互，要求决策元数据请求的分布，并根

据分布结果填充元数据分布信息缓存。通过多级的元数据分布信息缓存，BWMMS 形成层次化的元数据分布信息管理机制。其结构如图 3.2 所示。

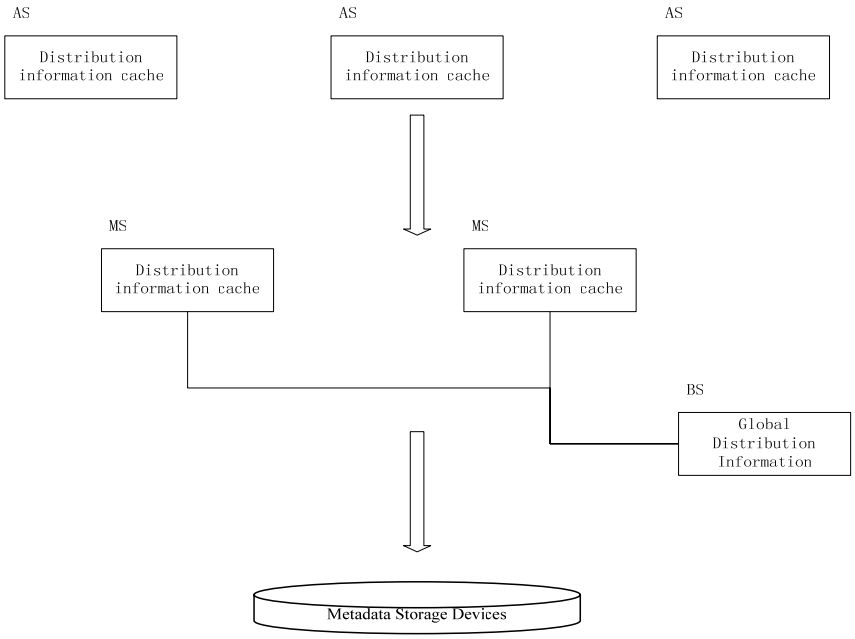

图 3.2 BWMMS 层次化元数据请求分布管理机制

图 3.2 描述的 BWMMS 层次化元数据请求分布管理机制由绑定服务器（Binding Server, BS），元数据服务器（Metadata Server, MS）和应用服务器(Application Server, AS)三个部分构成。BS 是文件系统元数据请求分布的集中决策点，主要完成活跃元数据与元数据服务器映射关系的管理工作。MS 接收 AS 的元数据请求，根据自己缓存的元数据分布信息处理请求。在必要的时候 MS 请求 BS 决策，并根据决策结果刷新元数据分布信息缓存。AS 是运行各种服务的服务器，它根据元数据请求访问结果缓存元数据的分布信息。

3.4.3 分布管理协议

元数据请求分布管理协议包括 AS 与 MS、MS 与 BS 之间的请求分布管理协议。AS 根据元数据请求的处理结果填充、或刷新其元数据分布信息缓存，其协议相对简单。MS 与 BS 间的元数据请求分布管理协议主要需要完成元数据请求分布映射的建立、查询和解除等工作。如图 3.3 所示。

活跃元数据的单一映射策略，决定活跃元数据的请求分布映射仅能由一个元数据服务器完成。当元数据分布在请求建立映射的服务器上时，这部分工作由同一个服务器完成。对应到图 3.3 的左半部分，其过程为：

根据 AS1 访问，MS1 通过 MAP 协议，请求 BS 为第一次访问变成活跃的元数据进行请求分布决策。其协议格式是(索引节点号，具有访问相关性的索引节点，元数据类型，映射模式)。BS 通过元数据请求分布算法，决定该元数据的宿主。并将结果返回给 MS1。

假定 MAP 协议返回该元数据的宿主为 MS1。MS1 将使用获得的元数据分布信息处理后续的元数据请求。当该元数据不再活跃，需要解除该元数据的请求分布管理时，MS1 主动通过 UNMAP 协议，请求 BS 解除对该元数据的请求分布管理。UNMAP 协议的格式为（索引节点号）。在获得“成功解除”的结果时，MS1 释放缓存中的信息。

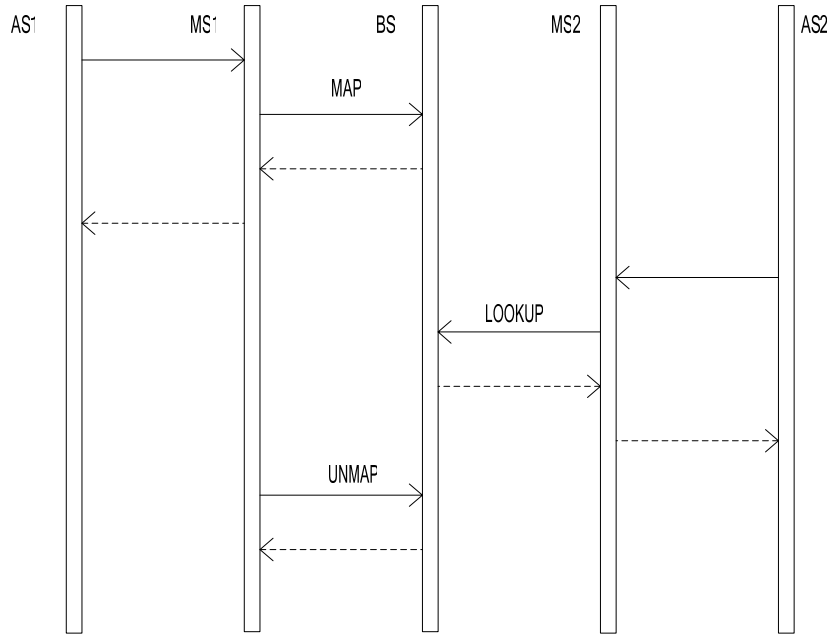

图 3.3 元数据请求分布管理协议

其他需要明确该元数据请求分布状况的元数据服务器，需要跟 BS 通信，获得分布结果。对应到图 3.3 右半部分，MS2 受 AS2 驱动，需要明确该元数据的请求分布信息。所以，它通过 LOOKUP 协议与 BS 通信，获得结果。LOOKUP 协议格式为（索引节点号），BS 返回“该元数据宿主为 MS1”的结果。

从协议的语义看，元数据请求分布的有效性在很大程度上取决于 BS 对 MAP 请求的响应，其算法的有效性将决定系统的元数据服务能力。

3.4.4 请求分布算法

元数据请求分布算法的终极目标是为活跃元数据选择最合适的元数据服务器作为其宿主，其发生在 MAP 协议的处理。BS 对 MAP 协议的处理算法将决定元数据请求处理的效率和元数据服务器的负载，需要综合考虑多种因素。

算法首先需要考虑用户在元数据服务器和文件系统两个方面是否存在特定的需求。出于性能或安全性等因素的考虑，用户可能要求其元数据请求只能由能够满足特定要求的元数据服务器处理。在这种情况下，请求分布算法应该在满足要求的元数据服务器范围选择。另一方面，用户的文件使用模式、用户间的数据共享模式，决定某些文件的元数据请求采用某种特定的分布策略更为合适。比如，每个用户在私有目录中存放文件的使用模式。由于用户间没有文件共享，并且各个用户的访问负载相差不大，目录子树分

区法能够支持有效的元数据服务。再比如，如果用户很少按照目录树遍历查找文件，哈希法能够支持元数据服务器的快速定位，提升系统的元数据服务效率。总之，元数据请求分布算法应该能够支持用户特定的需求。

算法还需要考虑元数据间存在的访问相关性。根据元数据访问协议，文件跟其父目录同时被访问的概率非常高。但它与其父目录的父目录间不一定存在访问相关性，元数据间的访问相关性仅存在有限范围中。在一般情况下，考虑与其父目录的相关性即可。

最后，算法需要考虑元数据服务器的负载情况。有效的分布决策算法需要在保证具有较高统计频率的元数据请求的有效处理的前提下，尽可能平衡服务器间的负载。如果某些服务器负载过高，可能限制元数据服务整体的扩展能力。

根据 MAP 协议，请求分布算法的输入参数为（索引节点号，父目录索引节点，元数据类型，映射模式)。其中索引节点号是被映射的元数据的标识。元数据类型表明被映射的是目录、或者普通文件。映射模式是 MS 给出的参考映射方式，包括“强制映射在本 MS”、“尽可能地映射在本 MS”、“尽可能地映射到其他 MS”和“强制映射到其他 MS”等模式。算法的正确输出是元数据宿主的标识。错误的时候输出相应的错误，由 MS 根据错误情况处理。算法的具体过程如图 3.4 所示。

算法首先明确候选元数据宿主服务器的范围。如果没有满足条件的元数据服务器，返回错误结果给 MS。在明确服务器范围后，它还需要明确该元数据是否关联特定的分布策略。如果有关联的特定分布策略，按照特定的分布策略选择候选宿主服务器。如果没有特定的分布策略绑定，则按照 BWMMS 的动态请求分布决策法（Dynamic）完成请求的分布。算法最后需要检测候选宿主的负载情况，如果不能分布新的元数据请求，则选择预测负载最小的服务器作为宿主服务器返回给 MS。

Dynamic 结合元数据的类型、映射模式和服务器负载，按照如下策略进行：

- 对于文件，首先选择父目录当前的宿主作为候选宿主。然后检查 MS 给出的参考映射模式。如果要强制映射到其他服务器，BS 选择预测负载最小的服务器作为宿主；
- 对于目录，首先以概率 α（比如 98%）在不包括父目录宿主服务器的元数据服务器集合中，选择预测负载最小的服务器作为其候选宿主。然后检查 MS 给出的参考映射模式。如果要强制映射到父目录宿主服务器，BS 选择父目录当前的宿主作为宿主。否则，选择候选宿主作为目录的宿主。

原型系统目前的实现采用元数据服务器当前分布的活跃元数据数目作为服务器的负载参数。还需要在后续工作中丰富负载参数构成，深化负载预测算法的研究。

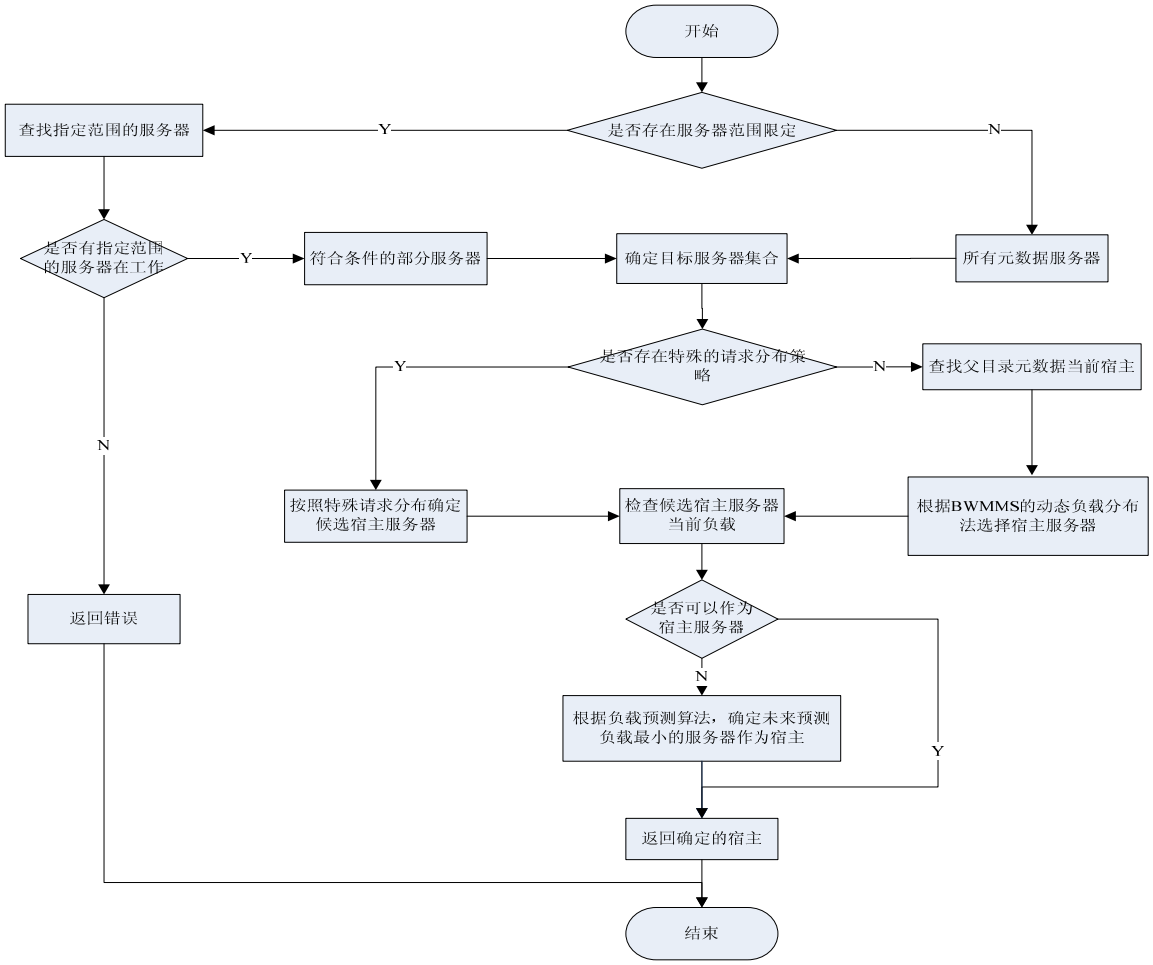

图 3.4 元数据请求分布决策算法

3.5 动态请求分布法的有效性评估

本节通过对比不同请求分布策略对系统的影响，说明 BWMMS 动态请求分布法的有效性。由于哈希函数的设计难度很大，测试仅仅对比动态请求分布法(Dynamic)和目录子树分区法(subdir)在各个客户端的请求负载不同时的表现。

评测环境包括 6 个元数据服务器和 7 个客户端，1 个采用 120GB 的 SATA 硬盘提供存储空间的存储设备，1 个 BS 和 1 个虚拟存储管理器。每个节点是 3.4GHz 的 Intel® Xeron®, 3GB 内存，RedHat® Linux® 8.0 操作系统。每个客户端在自己私有目录中运行 postmark[postmark2002]。Postmark 测试包括测试文件集生成、文件的事务[2]访问两个阶段，其主要模拟邮件服务器、文件服务器等企业应用的 I/O 访问模式。测试由不同的客户端负载构成，客户端 AS1-AS3 在单一目录下进行 10000 个事务，客户端 AS4-AS7 在 10 个子目录下总共运行 10000 个事务。

表 3.2 中的事务吞吐率是 7 个客户端聚合的每秒能够完成的事务数目，服务器工作

[2] Postmark 的事务由两个阶段构成，第一阶段是随机选择 1 个文件，读该文件全部数据，或者以追加写方式写入一定数量的数据。第二阶段是随机选择创建 1 个新文件，并写入一定数目的数据，或者删除测试集中已经存在的文件。

时间是 6 个元数据服务器的服务进程的工作时间总和。

表 3.2 两种分布策略的整体对比

	目录子树分区法	动态请求分布法	提高比例(%)
聚合事务吞吐率(ops)	622	802	28.94
聚合 MS 服务时间(s)	62.82	43.78	30.31

表 3.3 是各个客户端的事务吞吐率明细，单位为事务数目/秒。从表 3.3 可以看出，由于动态请求分布法结合服务器负载情况，动态地分布元数据请求，各个用户的元数据请求可以更合理地分布到服务器，用户的事务吞吐率相对能够得到保证。而目录子树分区法固定单个客户端的请求由既定元数据服务器处理，无法很好地利用服务器的处理能力，客户端的事务吞吐率将明显受客户端负载的影响。

表 3.3 两种分布策略的事务吞吐率明细

	AS1	AS2	AS3	AS4	AS5	AS6	AS7	总和
目录子树分区法	102	97	110	70	80	76	87	622
动态请求分布法	113	103	108	124	118	117	119	802

表 3.4 是各个服务器的具体工作时间，单位为秒。从表 3.4 可以看出，目录子树分区法的服务器负载的平衡性较差，存在服务器很少甚至没有工作的情况，服务器处理能力不能得到很好的利用。而动态请求分布法能够获得相对较好的服务器负载平衡。

表 3.4 两种分布策略的服务时间明细

	MS1	MS2	MS3	MS4	MS5	MS6	总和
目录子树分区法	14.76	16.17	15.43	16.45	0	0.00564	62.82
动态请求分布法	6.72	2.89	10.17	10.82	11.09	2.08	43.78

3.6 本章小结

元数据请求服务是文件系统元数据服务有效性的保证。元数据请求分布是元数据请求服务的核心问题，它将决定服务器的负载，影响元数据服务的扩展能力。

用户的文件访问表现出动态性和局部性的统计特征，仅仅依据静态信息进行元数据请求分布决策的目录子树分区法和哈希法等元数据请求分布策略，不能有效地支持元数据服务的动态扩展。

根据用户文件访问特征和系统扩展需要，BWMMS 提出后端集中决策机制，层次化管理元数据请求分布信息。综合考虑用户的特定需求、元数据间的访问相关性、元数据服务器的负载变化等因素，动态地完成元数据请求的分布决策，为元数据服务器规模和用户访问负载的动态变化提供支持。

请求分布策略有效性的对比评估试验结果表明，相对于目录子树分布法而言，BWMMS 的动态请求分布法能够获得近 30%的聚合事务吞吐率的提高，并能够更好地平衡元数据服务器的负载。

第四章 元数据分布信息缓存管理

集中方式的元数据请求分布管理要求有效机制的支持，以降低对集中决策服务器的依赖，弱化其对系统扩展能力的限制。用户元数据请求的局部性特征决定了活跃元数据被反复访问的概率很大，有效的元数据分布信息缓存能够缓减集中决策服务器的压力，提升其逻辑处理能力，增强系统的可扩展性。

本章结合 BWMMS 讨论集中元数据请求分布管理机制中的元数据分布信息的缓存管理，包括缓存的组织、元数据请求语义带来的缓存项的状态转换、缓存信息的替换策略等问题。最后通过实验评测元数据分布信息缓存对系统的影响。

4.1 元数据分布信息缓存的必要性

由于其简单性和有效性，集中方式的元数据请求分布管理机制具有重要的地位。尽管单个服务器因其有限的物理处理能力，而限制系统的扩展。但是，有效的请求分布管理策略可以降低集中决策服务器的负载。并且，以层叠方式组织和扩展的集中决策服务器系统结构，可以增强对前端服务器规模有效扩展的支持。

缓存技术通过有效的缓存信息缩短请求的处理路径，提高服务器的逻辑处理能力[Levy1990]。计算机系统中存在各种目的的信息缓存，如本地文件系统的目录项缓存、索引节点缓存、数据缓存[Thompson1978][Mckusick1984][Bach1986][Vahalia1996][Lions1996][Bovet2002][Tanenbaum2006]，分布式文件系统客户端的数据和元数据缓存[Dahlin1994-1][Dahlin1994-2]等。

对于文件系统元数据分布信息缓存而言，传统的缓存管理技术，如基于链表的组织结构、通过哈希法的快速查找、以及基于 LRU 的缓存替换算法等仍然适用。但是，它还需要结合元数据请求的语义，更优地组织和管理缓存。

在集中方式的元数据请求分布管理架构中，元数据分布信息缓存将起到至关重要的作用，包括：

1）降低集中决策服务器的负载，提高其逻辑处理能力。

尽管集中决策服务器的物理处理能力有限。但是，如果元数据访问负载鲜有突发、元数据请求不会分散到文件系统很广的范围时，元数据分布发生改变的概率很小，缓存的元数据分布信息将长时间有效，大多数元数据请求的处理不需要集中决策服务器的参与。这能够降低集中决策服务器的负载，提高集中决策服务器的逻辑处理能力。

2）缩短元数据请求处理路径。

在元数据服务器上，元数据请求的处理首先需要明确元数据的分布，以决定请求的

处理权限。文件系统需要根据用户可读的文件名字获得索引节点的标识，再根据索引节点标识获取文件索引节点及相关的元数据 [Bach1986]。元数据分布信息缓存能够支持元数据服务器根据请求处理权限，判断是否需要访问存储设备上的元数据。所以，元数据请求分布信息的缓存不仅可以缩短元数据分布信息的访问路径，还可以控制元数据的读取，消除不必要的存储设备访问，缩短元数据的访问路径。

4.2 元数据分布信息缓存管理的相关定义

用户的访问驱动 MS 请求元数据的分布映射。元数据分布结果将包含两种情况。一种是分布在自己的具有宿主权限的元数据分布信息，另一种是分布在其它 MS 的没有宿主权限的元数据分布信息。

只有在元数据不活跃时，元数据宿主 MS 才请求集中决策服务器解除其分布管理。并且，根据文件系统名字空间结构遍历时，将导致不具备宿主权限的 MS 缓存元数据的分布信息。所以，元数据分布信息缓存不仅需要管理当前所有的具有宿主权限的元数据分布信息，还需要缓存用户访问到、但没有宿主权限的元数据分布信息。

结合第 3.4.1 节，元数据分布信息缓存管理相关的定义为：

定义 4.1 BS 管理的系统当前所有活跃元数据分布信息集合 GMDT:

$$GMDT = \{item \mid item \in AM\}$$

GMDT 是所有活跃元数据的分布信息。所以，$GMDT = AM$ 。

定义 4.2 第 i 个元数据服务器 MSi 的元数据分布信息缓存称为“MDTi”。MDTi 中，分布在自己的具有宿主权限的元数据分布信息集合，称为“LDTi”。 LDTi 中的元数据分布信息项，称为“宿主权限项”:

$$LDTi = \{item \mid item \in MDTi \wedge MSi = HOST(item)\}$$

定义 4.3 MDTi 中，没有分布在自己的不具有宿主权限的元数据分布信息集合，称为“RDTi”。 RDTi 中的元数据分布信息项称为“非宿主权限项”:

$$RDTi = \{item \mid item \in MDTi \wedge MSi \neq HOST(item)\}$$

MDTi、LDTi 和 RDTi 的关系为：

$$MDTi = LDTi \cup RDTi,\ LDTi \cap RDTi = \varnothing$$

综合以上定义，GMDT 和各个 LDT 间存在的关系为：

$$GMDT = \bigcup_{i=1}^{n} LDT_i \ ,\ \bigcap_{i=1}^{n} LDT_i = \varnothing$$。其中n=|S|

4.3 元数据分布信息缓存的组织

元数据分布信息缓存管理首先需要确定标识缓存项的关键字。索引节点号能够唯一标识设备上的索引节点，它是关键字的一部分。在分布式文件系统中，由于文件删除操作不会广播给客户端，被删除文件仍然可能收到元数据请求。同时，已删除文件的索引节点号可能再次分配使用。为正确区别用户访问的目标索引节点，NFS[Sandberg1985]采用“索引节点号+索引节点序列号”的双关键字方式唯一标识用户访问的文件，索引节点序列号在新文件创建时递增。单位存储价格的降低[Henson2006]促进低成本的多版本文件支持。所以，为支持分布式文件系统的扩展，BWMMS 使用“索引节点号+索引节点序列号+索引节点版本号”作为元数据分布信息项关键字。

与传统的缓存管理相同，元数据分布信息缓存使用哈希链表提高分布信息项的查找速度，目标哈希链表通过 $func(ino\#, generation\#, version\#)$ 确定。缓存采用 LRU 管理缓存项的替换。在 LRU 基础上，由于用户访问特征决定了宿主权限项和非宿主权限项具有不同的访问概率，系统应该分开管理这两类信息。所以，缓存采用两个不活跃链表 entry_unused_hosted 和 entry_unused_unhosted 分别管理这两类不活跃项，并基于 LRU 决定分布信息项的替换。表 4.1 是元数据分布信息的结构。

表 4.1 元数据分布信息结构

```
struct mdt_entry {
    struct list_head me_list; /*list */
    struct list_head me_hash; /* hash list*/
    ino_t me_ino; /*  索引节点号*/
    u32 me_generation;/*索引节点序列号*/
    u32 me_version;/*索引节点版本号*/
    u32 me_host; /* MS 标示*/
    spinloct_t me_lock;
    u32 me_state; /* state */
    atomic_t me_refcnt; /* reference count，>0 为活跃元数据*/
    /*用来同步元数据请求*/
    u64 me_timestamp;
    atomic_t me_modicnt;
    atomic_t me_looupcnt;
    atomic_t me_unlinkcnt;
    atomic_t me_linkcnt;
    atomic_t me_renamecnt;
};
```

字段 me_list 和 me_hash 用来组织元数据分布信息缓存。通过这两个字段，MDT 的内存组织如图 4.1 所示。me_refcnt 记录该元数据分布信息项的活跃度。链表 entry_in_use 用来组织引用计数 me_refcnt 大于 0 的活跃元数据项。Entry_unused_hosted 和 entry_unused_unhosted 分别用来组织不活跃的宿主权限和非宿主权限项。Me_ino 记录索引节点号，me_generation 记录索引节点序列号，me_version 记录索引节点版本号。Me_host

记录该文件索引节点当前的宿主 MS 信息。其他字段主要用作元数据请求的同步控制，在第六章将具体阐述。

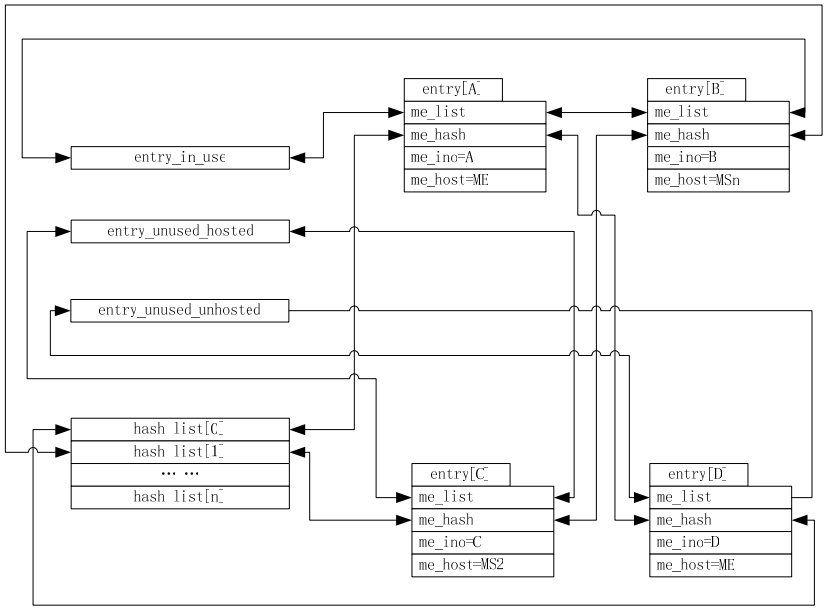

图 4.1 元数据分布信息缓存内存组织

4.4 元数据分布信息的状态转换

元数据请求语义将导致元数据分布信息的状态转换。根据元数据活跃性管理元数据请求分布的策略决定了元数据在第一次访问前和最后一次访问后没有相应的缓存项存在。并且，不同类型的元数据分布信息项将支持不同的请求处理流程。所以，在用户访问驱动下，两类元数据分布信息项具有不同的状态转换。从整体而言，元数据分布信息缓存项的生存周期从用户第一次请求时生成内存结构开始，到不活跃时释放内存结构结束。在生存周期过程中，它将随着元数据请求发生状态的转换。其整体的状态转换 Petri Net 表示如图 4.2 所示，起始状态仅位置 Pno-entry 具有 1 个标记。

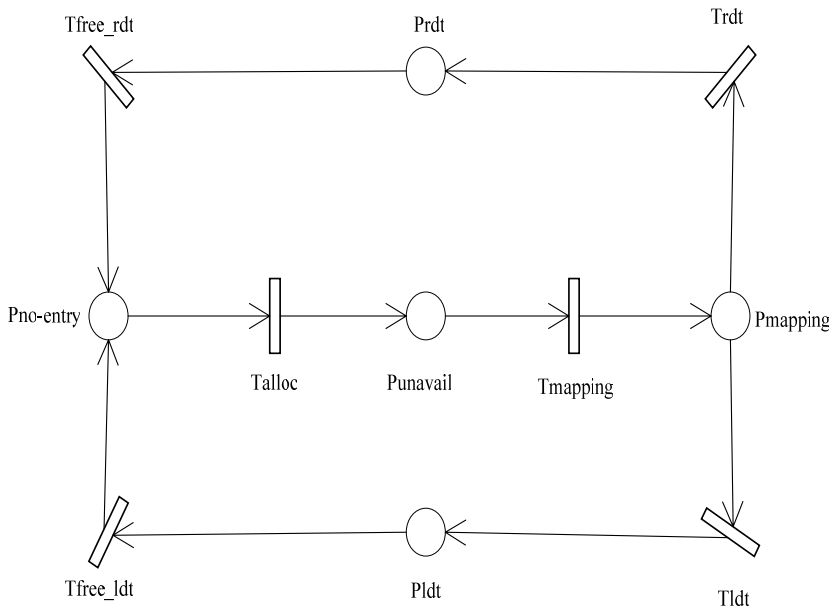

图 4.2 元数据分布信息的整体状态转换图

其中各个位置的含义为：

表 4.2 整体状态转换图的位置含义

位置名	含义
Pno-entry	没有对应缓存项存在，需要分配内存结构
Punavail	具有内存结构，但其中信息还不可用，需要请求 BS 决策其请求分布
Pmapping	正在请求 BS 决策过程中
Pldt	具有宿主权限
Prdt	没有宿主权限

各个变迁的含义为：

表 4.3 整体状态转换图的变迁含义

变迁名	含义
Talloc	为分布信息项分配内存结构
Tmapping	与 BS 通信，请求分布决策
Tldt	请求结果，具有宿主权限
Tfree_ldt	释放宿主权限的分布信息项
Trdt	请求结果，没有宿主权限
Tfree_rdt	释放非宿主权限的分布信息项

4.4.1 宿主权限项的状态转换

根据宿主权限项的请求语义，扩展图 4.2 的 Pldt 部分，得到图 4.3 的宿主权限项的状态转换的 Petri Net 描述，其起始状态是位置 Pno-entry 具有 1 个标记。

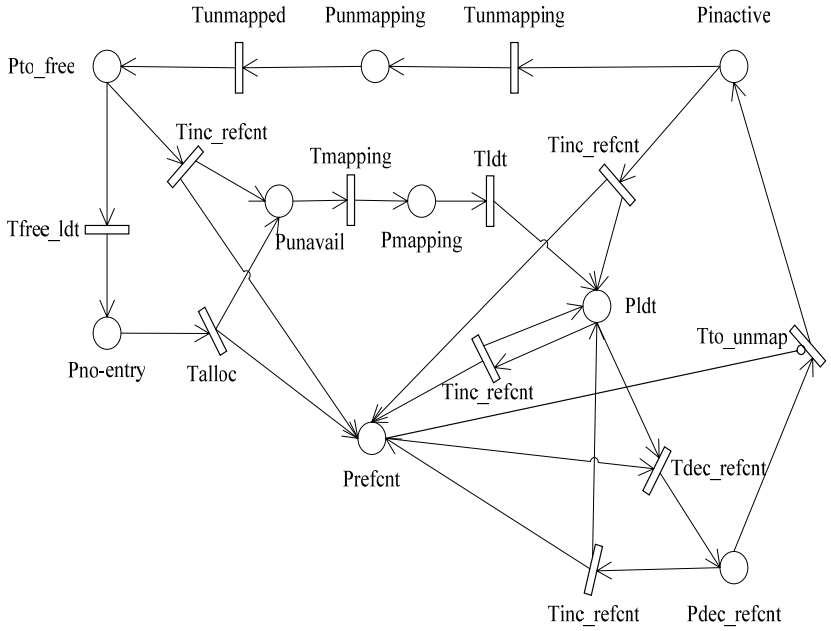

图 4.3 宿主权限项的状态转换图

图 4.3 中与图 4.2 中相同位置名的位置，具有相同的含义。除此之外的位置的含义为：

表 4.4 宿主权限项状态转换图的位置含义

位置名	含义
Prefcnt	该项的引用计数值
Pdec_refcnt	引用计数被减少
Pinactive	引用计数为 0，对应元数据不活跃
Punmapping	正在与 BS 通信，要求 BS 解除该元数据的分布管理
Pto_free	等待释放内存。此时已经从哈希链表摘下，不会与新分配内存结构冲突

同样，图 4.2 中没有包括的转换变迁的含义为：

表 4.5 宿主权限项状态转换图的变迁含义

变迁名	含义
Tdec_refcnt	请求使用完毕该项，减少引用计数
Tinc_refcnt	请求需要使用该项，增加引用计数
Tto_unmap	引用计数为 0，可以请求 BS 释放分布管理
Tunmapping	与 BS 通信，释放该项的分布管理
Tunmapped	释放请求结束，可以释放其内存结构

系统同时存在多个请求使用分布信息项，这表现在表 4.1 的 me_refcnt。Me_refcnt>0 表示有请求使用该项，元数据处于活跃状态，此时它链接在相应的哈希链表和 entry_in_use 链表。位置 Prefcnt 记录引用计数，其标记数目等同于 me_refcnt 的值。Talloc 将它初始化为 1，因为此时必然有请求正在使用该项。当分布信息项处在 Pldt 时，其他请求可以使用该项。本请求也可能使用完毕，所以增加或者减少引用计数都是可以进行的。只有在引用计数为 0（位置 Prefcnt 没有标记）时，该信息变成不活跃的，这表现在位置 Prefcnt 和变迁 Tto_unmap 之间的禁止弧。当 me_refcnt 为 0 时，它将通过其 me_list

连接到 entry_unused_hosted 链表，但此时仍通过 me_hash 链接到相应的哈希链表。在与 BS 真正开始解除不活跃元数据分布之前，可能有请求访问该元数据，其再度活跃。表现在位置 Pinactive 和位置 Pldt、Prefcnt 间的变迁 Tinc_refcnt。

当宿主权限项不活跃时，MS 主动要求 BS 释放对该项的分布管理。它首先需要通知 BS 解除对它的管理。完成后，首先将不活跃项从链表 entry_unused_hosted 摘下，然后再释放内存结构。最后，Tfree_ldt 进行时，才将分布信息项从哈希链表摘下，释放其内存结构。

4.4.2 非宿主权限项的状态转换

非宿主权限项的状态转换 Petri Net 描述如图 4.4 所示，起始状态是位置 Pno-entry 具有 1 个标记。相对于宿主权限项的状态转换图而言，变迁 Tfree_rdt 的作用和 Tfree_ldt 等同，非宿主权限项通过变迁 Trevalidate 驱动分布信息的刷新。

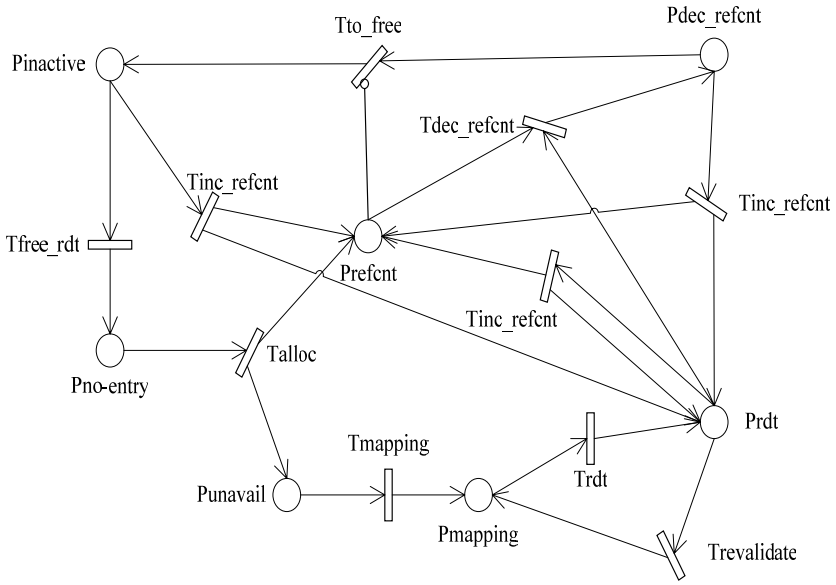

图 4.4 非宿主权限项的状态转换图

表 4.6 是图 4.2 和图 4.3 都没有的变迁的含义。

表 4.6 非宿主权限项状态转换图的变迁含义

变迁名	含义
Trevalidate	需要刷新非宿主权限项的信息

相对于宿主权限项而言，非宿主权限项的释放不需要请求 BS 解除请求分布。但是，由于元数据分布信息的动态变化，RDT 中的信息可能过时，需要向 BS 请求刷新信息，表现为位置 Prdt 和位置 Pmapping 间的变迁 Trevalidate。但是，刷新结果不会将该项转成宿主权限项，它获得的仅是新的宿主 MS 的信息。

4.5 状态转换图的活跃性分析

Petri Net 的活跃性分析可以通过求解其可达树，再合并可达树中的相同节点获得可达图完成。如果可达图中所有节点都有进和出的弧，从任意节点开始，经过一定变迁序列达到其他节点，Petri Net 是活跃的。Petri Net 可达树求解算法[Murata1989][LinChuang2001]如表 4.7 所示。

表 4.7 Petri Net 的可达树求解算法

1）根节点 r 由 M0 标注。

2）一个标注 M 的结点 x 是一个叶结点，当且仅当不存在 t∈T，t 在 M 是可实施的或者在从 r 到 x 的路径上存在一个结点 y≠x，但节点 y 也是由 M 标注的。

3）如果一个标注 M 的结点 x 不是一个叶结点，那么对于所有 t∈T 使得在 M 下可实施的 t 实施而产生一个新的结点 y，且在从 x 到 y 新产生的弧上标注 t。y 结点标注的标识 M' 可由 M'' 来计算，M'' 满足于 $M[t>M''$，即 $\forall s \in S: M''(s) = M(s) - W(s,t) + W(t,s)$。$M'$ 的计算可区别为两种情况：

①在从 r 到 y 的路径上，如果存在标注 M''' 的结点 z≠y 且 $\forall s \in S$：$M''(s) \geq M'''(s)$，那么：

$$M'(s) = \begin{cases} M'(s), \text{如果} M'(s) = M'''(s) \\ \omega, \text{其他} \end{cases}$$

②其他情况，$M' = M''$ 。

其中的 M 是所有位置的标记序列。W（x,y）是位置 x 到变迁 y，或者变迁 x 到位置 y 的弧权。M（s）表示在标记 M 中位置 s 的标记数量。

4.5.1 宿主权限项

根据图 4.3 的宿主权限项的状态转换图、表 4.7 的 Petri Net 可达树求解算法，宿主权限项的状态转化可达树如图 4.5 所示。

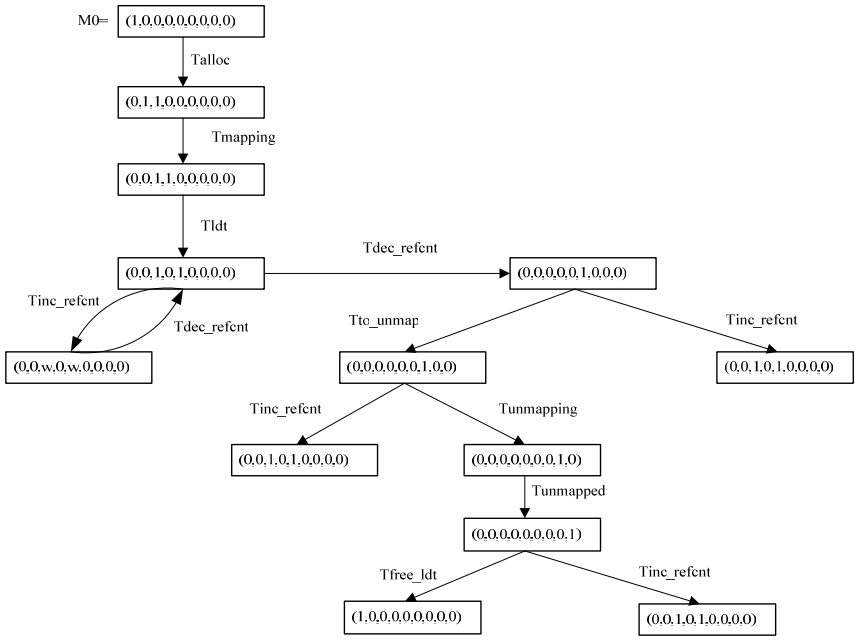

图 4.5 宿主权限项的状态转换可达树

图4.5中的节点是各个位置的标记数量的序列，其顺序为（Pno-entry, Punavail, Prefcnt, Pmapping, Pldt, Pdec_refcnt, Pinactive, Punmapping, Pto_free）。

将图4.5中相同节点合并，得到图4.6的宿主权限项的状态转换可达图。从图4.6可以看出，从宿主权限项状态转换可达图的任何节点出发，都可以通过一定变迁序列到达其它节点，状态转换图是活跃的。

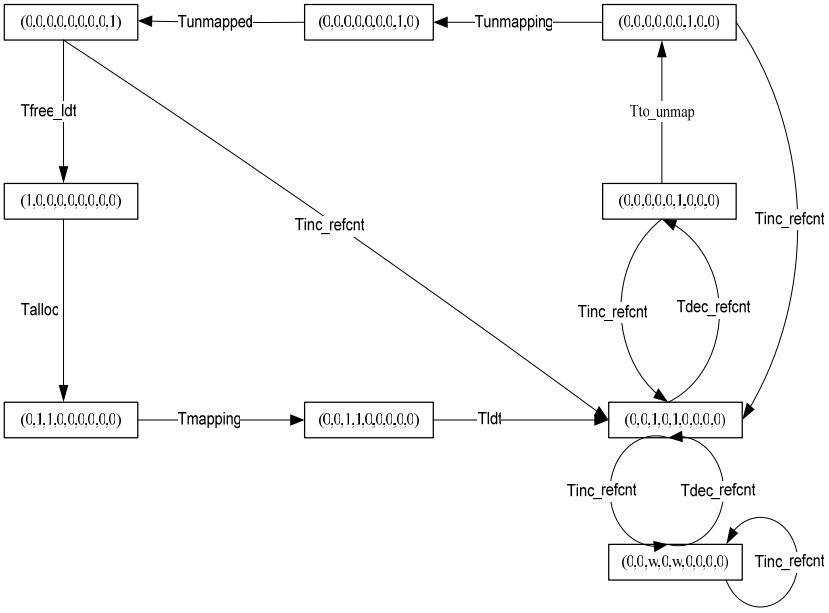

图4.6 宿主权限项的状态转换可达图

4.5.2 非宿主权限项

同样，根据图4.4和表4.7，非宿主权限项的状态转换可达树如图4.7所示。

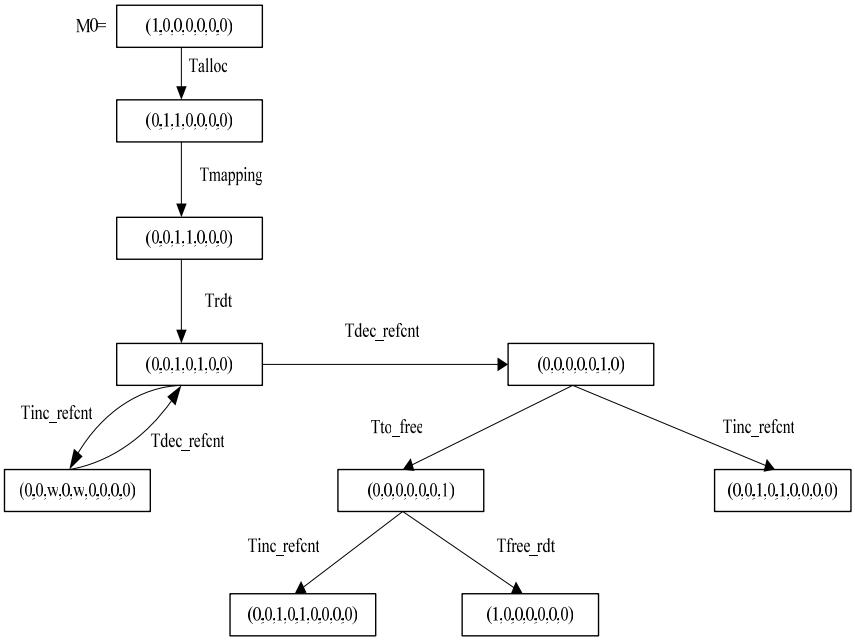

图4.7 非宿主权限项的状态转换可达树

节点包含各个位置的标记数量，其顺序为(Pno-entry, Punavail, Prefcnt, Pmapping, Prdt, Pdec_refcnt, Pinactive)。

合并图 4.7 中的相同节点，得到图 4.8 的非宿主权限项的状态转换可达图。同样，非宿主权限项的状态转换是活跃的，不会出现死锁情况。

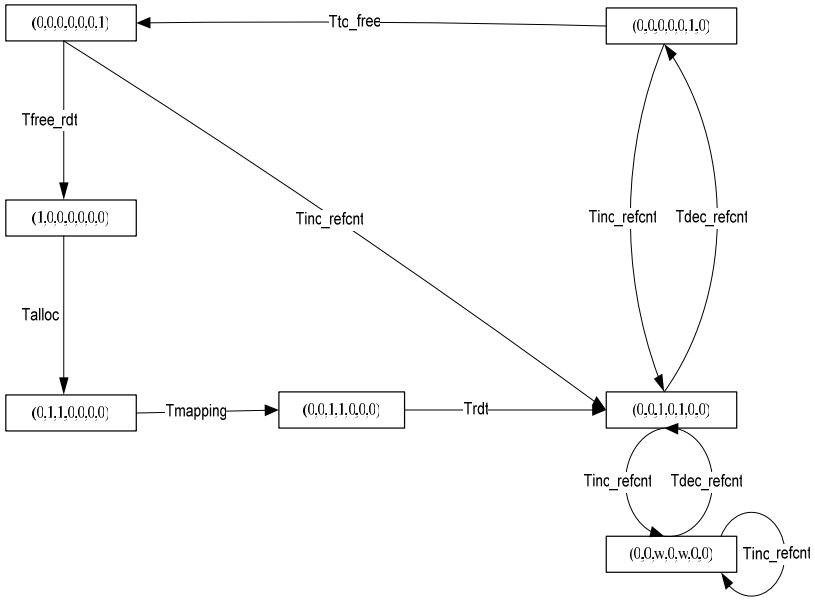

图 4.8 非宿主权限项的状态转换可达图

4.6 不活跃元数据信息的替换策略

不活跃元数据项的替换策略，将影响元数据分布信息缓存的命中率，从而影响 BS 的负载和元数据请求处理的时间延迟。

综合考虑内存开销、BS 负载等多种因素，元数据分布信息缓存中的不活跃元数据分布信息项的释放策略，由释放时间间隔、缓存的内存开销和不活跃分布信息项的数量等三个因素决定。每隔固定时间间隔，缓存总的数目、或者不活跃分布信息项的数目超过预定值时，缓存管理将根据 LRU 算法释放不活跃的分布信息项。

由于用户的元数据访问表现出很高的局部性，具有宿主权限的元数据被再次访问的概率，要远大于非宿主权限的元数据。并且，由于宿主权限的释放需要与 BS 的通信，时延较大。所以，在进行不活跃元数据项释放时，首先针对非宿主权限项进行。只有在系统内存压力非常大时，才选择宿主权限项进行释放。

结合内存压力和固定时间间隔策略，不活跃元数据的替换策略能够较好地保证系统对内存资源的消耗。非宿主权限项先于宿主权限项的释放策略，能够在保证缓存命中率的同时，降低缓存的内存开销。

4.7 缓存有效性的影响评估

为说明元数据分布信息缓存的有效性，本节从两个方面进行评估。首先评估不同缓存命中率对系统聚合吞吐率、BS 负载的影响。然后，评估典型应用模式下，应用能够获得的缓存命中率情况。

4.7.1 缓存命中率的影响

通过调整不活跃元数据分布信息的释放参数，获得 0%、50%、90%、95%、98%和 100%的缓存命中率，统计客户聚合请求吞吐率、需要与 BS 的通信数目比例。0%缓存命中率和 100%命中率分别模拟最坏和最好缓存命中情况。

为避免服务器数量庞大带来的竞争影响，测试环境由 1 个客户端、1 个元数据服务器、1 个具有 120GB 的 SATA 硬盘的存储设备和 1 个绑定服务器构成。每个服务器的 CPU 是 Intel® Xeon® 3.4GHz, 3GB 内存，操作系统是 RedHat® Linux® 8.0。

测试由 postmark 驱动。Postmark 参数是 10000 个 4KB 至 16KB 的文件，10 个子目录，50000 个事务，块读写大小为 4KB。Postmark 首先需要生成 10000 个测试文件。在创建测试文件时，MS 需要与 BS 进行通信，完成 MS 元数据分布信息缓存的填充。

图 4.9 是不同缓存命中率下，postmark 聚合事务吞吐率，单位是每秒完成的事务数量。从图中可以看出，随着缓存命中率的提高，请求吞吐率明显提高。当命中率达到 95%后，缓存对系统的事务吞吐率的影响比较弱，事务吞吐率逐渐平稳。

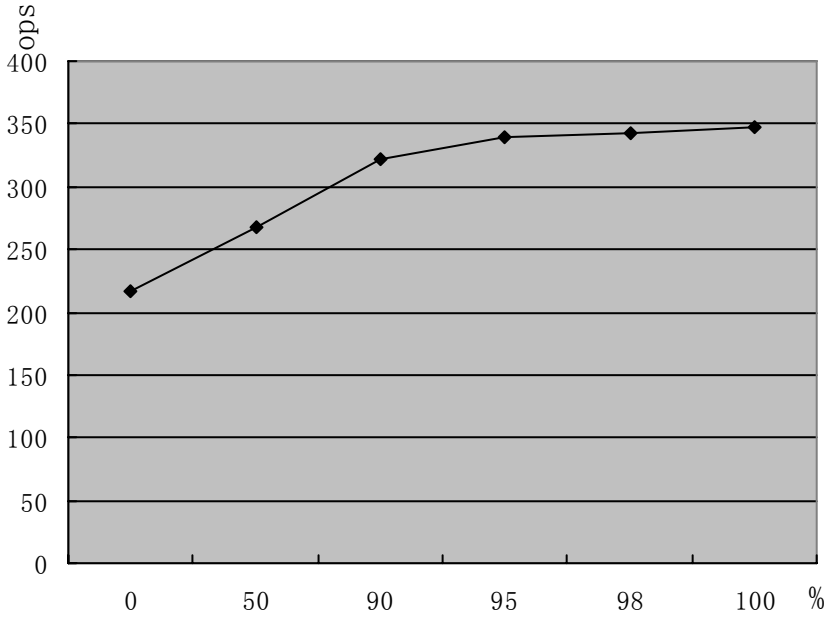

图 4.9 聚合事务吞吐率随缓存命中率变化

缓存中有效的元数据分布信息可以直接用来决定客户端请求的处理。在缓存信息无效时，通过请求 BS，刷新缓存信息。

BS 的负载压力可以通过请求比率来描述：

$$ratio = REQUEST(ms - bs) / REQUEST(as - ms)。$$

其中 REQUEST（as-ms）是 AS 发送给 MS 的请求数量，REQUEST(ms-bs)是 MS 与 BS 的通信数量。

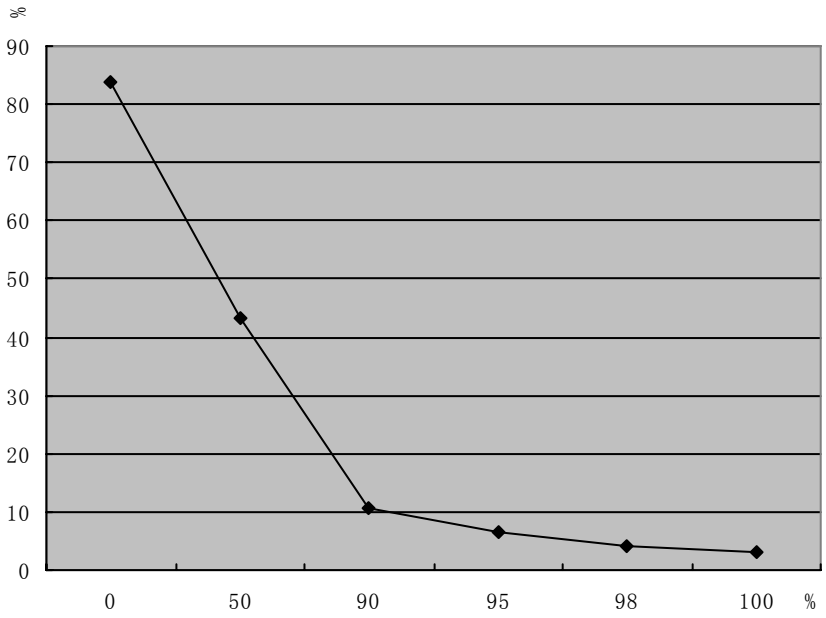

图 4.10 BS 通信负载随缓存命中率变化

图 4.10 是根据上述公式计算的各种缓存命中率下的请求比率，每个计算值包含创建测试文件集的 10000 次通信。从图可以看出，缓存命中率对 BS 的负载压力的影响非常明显。

从上述两个图还可以看出，当缓存命中率达到一定值后，进一步的提高对系统的促进作用将不太明显，为缓存管理的优化提供参考。

4.7.2 不同应用的缓存命中率

为评估不同应用可能获得的缓存命中率，本部分评估创建/删除空文件、应用以共享模式访问同一测试文件集、以及应用以私有模式访问不同测试文件集三种应用下，各个服务器的缓存命中率。测试采用 7 个客户端、6 个元数据服务器的配置，服务器的硬件配置跟上述评估相同。结果如表 4.8 所示。

表 4.8 不同应用的缓存命中率

	MS1	MS2	MS3	MS4	MS5	MS6	*平均值*
创建/删除空文件	87.49	87.49	87.52	87.49	87.49	47.06	80.76
共享模式	97.81	97.24	97.06	96.99	97.6	93.49	96.70
私有模式	96.21	96.19	96.22	96.22	96.22	96.18	96.21

由于创建空文件需要 BS 决定新创建文件的元数据请求的分布，每一个创建请求都

需要与 BS 通信。创建完成的文件并没有被反复访问，所以，缓存的作用不明显，其缓存命中率非常低，约为 81%。

共享模式是 7 个客户端同时通过“find . -exec stat {} \;”命令访问同一个目录下的文件。元数据请求以单个活跃文件为单位，并且同一个文件被反复访问的次数较多。在一个客户端驱动了请求分布并获得结果后，其他的客户端可以利用缓存信息，处理元数据请求。所以，其缓存命中率较高，达到 96.70%。

私有模式是各个客户端在各自目录下进行文件的创建和读写请求。每个元数据的请求分布结果仅对本客户端有利。所以，其缓存命中率相对于共享模式低，但由于缓存信息能够重复利用，其缓存命中率比文件创建/删除高得多。

综上所述，BWMMS 元数据分布信息缓存管理能够为具体应用提供有效的支持。

4.8 本章小结

因为其简单性和易扩展性，集中方式的元数据请求分布决策机制将成为重要的元数据请求分布管理机制。

用户元数据访问表现出的局部活跃性特征决定了缓存机制的有效性。本章结合 BWMMS 讨论集中元数据请求分布管理机制中有效的元数据分布信息缓存管理。

在元数据分布信息缓存管理中，通过哈希加速查找等传统的缓存管理技术仍然适用。在此基础上，BWMMS 还需要根据元数据的活跃性管理元数据的分布信息缓存。它需要结合元数据请求的语义，描述、验证元数据分布信息状态转换的活跃性。

BWMMS 的不活跃元数据分布信息项替换策略，结合内存开销压力、元数据的活跃性和用户访问的局部性特征，优先替换不活跃的非宿主权限信息，以支持宿主权限信息的有效访问。

元数据分布信息缓存的命中率将直接影响BS的负载和元数据请求处理的时间延迟。在本章的最后，通过调整不活跃元数据分布信息的释放参数，不同缓存命中率对系统聚合吞吐率和 BS 压力的影响得到评估。结果表明，元数据缓存管理有效性对 BS 的压力和系统元数据请求处理吞吐率的影响明显，达到一定值后，缓存命中率对系统的促进作用将不明显。同时，BWMMS 当前提供的缓存管理策略能够很好地支持应用的需要。

第五章 基于宿主改变的请求原子性保证协议

更改文件系统的元数据请求要求其结果具有原子性。在集群环境中，涉及多个元数据的元数据请求可能需要跨服务器处理，元数据请求的原子性保证问题更加突出。尽管跨服务器请求的比例极小，但两阶段提交等传统的请求原子性保证协议对系统整体性能和错误恢复能力的影响比较大。探讨轻量级协议、降低比例极小的跨服务器请求对系统的影响，对提高系统扩展能力有着重要作用。

本章讨论基于集中共享存储结构、对称服务器结构的跨服务器元数据请求的原子性保证问题。通过改变活跃元数据的宿主，完成元数据在服务器间的迁移，跨服务器请求涉及的活跃元数据集中到单个服务器。在本地文件系统技术支持下，元数据请求由单个元数据服务器集中处理，有效解决其原子性保证问题。

5.1 问题描述

在集群环境中，元数据请求分布算法可能将访问上下文相关的元数据分布到不同的服务器，导致涉及多个元数据的请求需要跨元数据服务器协同处理。系统故障可能导致文件系统的不完整更新。如何通过保证单个请求的原子性，实现文件系统的原子更新，是分布式文件系统研究的重要问题。

为减少文件系统的不一致，出现故障的文件系统需要尽可能地修复。FSCK [McKusick1994]是本地文件系统常用的修复方法。它通过对比资源使用情况，完成文件系统元数据的修复。FSCK 需要搜索整个文件系统，其开销随文件系统规模线性增长。

针对 FSCK 的不足，日志技术[Tweedie1998][Best2002][Chinner2006][ReiserFS]使用额外的存储空间记录文件系统的更改，以缩小故障恢复的检测范围。它以单个请求的事务性保证为单位，将文件请求结果首先写到日志中，然后在更新存储设备后清除日志内容。由于日志技术需要两次设备写操作，且日志本身的内容需要同步写，其性能相对较低。为解决日志技术的性能较低问题，软更新技术[Ganger2000]通过将请求更改的元数据排序，按照请求处理的时间顺序更新设备，保证请求结果的事务性。

在分布式系统中，两阶段提交及其变种协议[Bernstein1987][Mullender1990][Samaras1993][Luckham1995]常用来保证分布式请求的原子性。尽管两阶段协议本身时延可以接受。但其协议过程的锁机制、复杂的错误恢复协议，导致它对系统整体性能和错误恢复能力的影响较大[Liu1994]。在新的存储架构中，探讨简单高效的请求原子性保证协议，显得非常必要。

5.2 两阶段提交协议

两阶段提交协议是分布式系统中广泛采用的请求原子性保证协议。本节介绍其内容，以作为 BWMMS 协议的对比参照。两阶段提交协议主体包括 1 个提交协同者(Coordinator,图中简写为 C)和若干个提交参与者(Participant,图中简写为 P)，协议过程如图 5.1 所示：

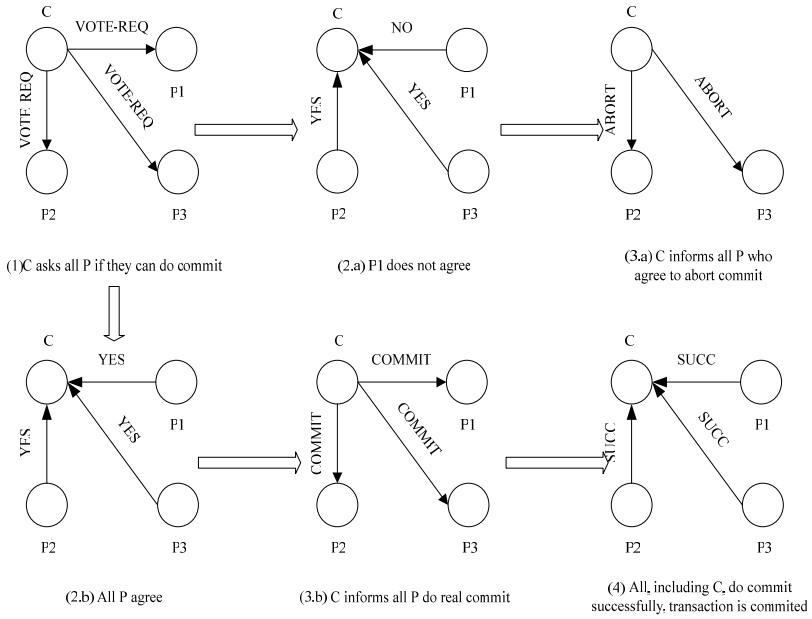

图 5.1 两阶段提交协议

如图 5.1 所示，两阶段提交协议的过程为：

1. 图 5.1（1），提交协同者向所有参与者发出 VOTE-REQ 消息，要求所有参与者投票此次更新结果是否提交。
2. 提交参与者收到 VOTE-REQ 消息后，它向提交协同者返回自己的投票结果，“YES”或“NO”（图 5.1（2.a））。“NO”表示本参与者决定放弃本次提交操作。
3. 提交协同者收集所有参与者的投票结果。如果所有参与者和自己的投票结果都是“YES”（图 5.1（2.b）），它将向所有参与者发送 COMMIT 消息（图 5.1（3.b））。否则，提交协同者做出“放弃本次提交”的决定，并向所有投出“YES”结果的参与者发送 ABORT 消息（图 5.1（3.a）），结束本次提交协议。
4. 所有投出“YES”结果的参与者等待协同者发来的“COMMIT”或者“ABORT”消息。在收到提交协同者的消息后，它根据收到的消息进行相应的处理（图 5.1（4）），然后结束本次提交协议。

从图 5.1 的两阶段提交协议过程可知，将“VOTE-REQ”和“YES/NO”，“COMMIT”和“SUCC/FAIL”分别当作 1 对消息，在涉及到 N 个（N>=2）服务器的操作中，成功提交需要 2(N-1)对消息和 N 次更新设备。不能成功提交时，其最小开销是(N-1)个都投否决票时的(N-1)对消息，最大开销是仅有 1 个提交参与者投否决票，需要(2N-3)对消息。同时，分布式日志技术要求日志内容的同步更新。

5.3 BWMMS 的跨服务器请求原子性保证协议

两阶段提交协议需要相对较多的消息通信和设备更新，还需要能够保证持久存储的

分布式日志技术提供支持。涉及到 4 个服务器(N=4)的文件移动操作，在结果成功执行的情况下，它需要 6 对消息通信和 4 次更新设备的操作。在不能成功提交时，其消息对数目范围是[3，5]。在系统规模非常大的情况下，两阶段提交协议的开销比较大。

为减少请求需要的消息数量和设备更新次数，简化系统的错误恢复过程，BWMMS 利用集中共享存储架构和对称服务器结构提供的灵活性，通过改变元数据宿主，完成元数据在服务器间的迁移，将多服务器协同处理转变成单个服务器集中处理，并借助本地文件系统技术提供请求事务性保证。

5.3.1 目标跨服务器请求

根据第 3.1 节的元数据访问协议，在存储服务与请求服务分离后，需要保证其原子性的跨服务器请求主要包括文件创建、删除和移动等。

文件创建请求向文件系统名字空间添加新的文件。它首先创建一个代表新文件的索引节点，然后在父目录中添加指向该文件的目录项。它要求新创建的文件索引节点先于父目录数据块写回设备，防止故障带来错误的索引节点引用。

文件移动请求将文件（旧文件）从一个目录移动（源目录）到另一个目录（目标目录），同时可能将文件重命名为新的文件名字（新文件）。它需要在目标目录中添加目录项指向新文件，然后从源目录中删除指向旧文件的目录项。如果目标目录中原来存在与新文件名字相同的文件(同名文件),还需要更改目标目录中原来指向同名文件的目录项，修改同名文件的索引节点。文件移动操作最多将更改 4 个元数据，是文件系统名字空间最复杂的元数据请求。

文件删除请求从父目录删除对文件的引用。删除完成后，不能再通过父目录引用文件的索引节点。它要求父目录数据块先于文件索引节点更新到存储设备。

5.3.2 元数据迁移的可行性

如前面所述，BWMMS 的集中共享虚拟存储架构、全共享的存储资源使用方式、文件系统服务的模块化和用户访问统计特征等,使得 BWMMS 通过元数据迁移保证文件系统一致性成为可能。具体包括：

1. BWMMS 基于集中共享虚拟存储模型，系统所有的元数据服务器共享线性逻辑元数据资源空间。对称的服务器结构保证元数据服务器访问同一个逻辑元数据资源的成本相同，元数据请求可以由任意元数据服务器处理，不会有明显的性能影响。
2. 在获得元数据的宿主权限后，每个元数据服务器都可以利用元数据，进行任意的元数据请求，元数据在服务器间的迁移不会影响系统的功能。
3. 分布式文件系统元数据的存储服务和请求服务分离。存储资源可以在任何服务器上释放给存储服务，不会引起存储资源管理的异常。

4. 已有研究表明，可能成为跨服务器请求的文件创建、删除和移动请求所占比例不大，不超过 5%，如表 5.1 所示[Gibson1997]。在元数据请求分布策略决策下，跨服务器请求的比例将更小[Hendricks2006]。已有的两阶段提交协议等传统的请求原子性保证协议，对系统的性能和错误恢复能力的影响极大，有必要探讨轻量级协议的可能性。

表 5.1 跨服务器请求统计特征

NFS			
Operation	Description	Percent	Quantity(*10^6)
DirReadWrite	Creation of files and directories, file renaming, links, etc.	0.7	0.21
DeleteWrite	Deletion of files and directories	0.1	0.04
AFS			
CreateFile	Create a new file in the AFS server namespace	2.1	2.0
Rename	Move a file in the AFS server namespace from one location to another location	1.9	1.7
RemoveFile	Delete a file stored on AFS server	1.5	1.4
Others	Operations includes ACL manipulations, symlinks, directory create and deletion,lock management, volume management,etc.	3.4	3.2

5.3.3 元数据迁移对象

元数据宿主改变首先需要明确迁移的对象和迁移的粒度。目录子树分区法和哈希法的迁移对象是持久存储的元数据和动态的文件锁信息等。它以构成文件系统名字空间目录子树的一组文件作为迁移对象，需要将整个目录子树迁移。相对目录子树分区法更甚的是，哈希法需要迁移的元数据范围不可控，文件系统名字空间的结构将迁移范围放大，直至目录树的叶结点。

BWMMS 的元数据迁移对象是单个索引节点及其关联的元数据块，仅迁移需要持久存储的文件系统元数据, 不迁移文件锁信息等动态的元数据。如果在迁移时元数据存在关联的动态元数据信息，它首先阻止新的动态元数据信息的产生，同时根据迁移策略进行判断，等待或者要求释放已有的动态元数据信息。当没有动态元数据信息存在时，元数据迁移才进行。

5.3.4 迁移协议数据格式

MS 在元数据迁移的数据格式中包括元数据迁移原因，以完成元数据迁移请求与其他元数据请求的同步。元数据请求的同步控制将在下一章讨论。BS 将元数据请求迁移内容转发给被迁移元数据当前的宿主 MS。元数据迁移的请求参数格式如表 5.2 所示。

表 5.2 元数据迁移协议格式

```
enum migrate_reason {
        /*硬连接的目标文件*/
         MGR_LINK = 0,
        /*符号连接的目标文件*/
```

```
        MGR_UNLINK,
        /*移动文件的目标父目录*/
        MGR_RENAME_TDIR,
        /*被移动文件*/
        MGR_RENAME_OINO,
        /*移动文件的目标文件*/
        MGR_RENAME_NINO
};
typedef enum migrate_reason migrate_reason;
struct migrate_request {
        eino_t ino;
        u32 generation;
        u32 version;
        migrate_reason reason;
};

enum migrate_result {
        MGP_OK = 0,
        MGP_DELETED,
        MGP_NOTEMPTY,
        MGP_BUSY,
        MGP_NOTSUPP
};
typedef enum migrate_result migrate_result;
struct migrate_reply {
        eino_t ino;
        u32 generation;
        u32 version;
        migrate_reason reason;
        migrate_result result;
};
```

5.3.5 迁移协议时序图

元数据迁移包括文件索引节点内容的迁移和元数据分布信息的更改。元数据迁移完成元数据从源宿主移动到目标宿主的工作，元数据分布信息改变完成相关服务器记录的元数据当前分布信息的更改。元数据迁移过程必须正确更改元数据的分布信息，避免系统出现“同一个活跃元数据具有多个宿主”的情形。

为降低元数据迁移协议的复杂度，避免复杂的错误恢复协议，元数据迁移请求通过 BS 转发、元数据内容通过 SN 中转。源 MS 将元数据内容写回到 SN，目标 MS 从 SN 读回元数据的内容。在迁移过程中，源 MS、BS 和目标 MS 根据自己的处理结果，更改元数据分布信息。元数据迁移协议过程如图 5.2 所示，假定被迁移元数据当前分布在 MS2。

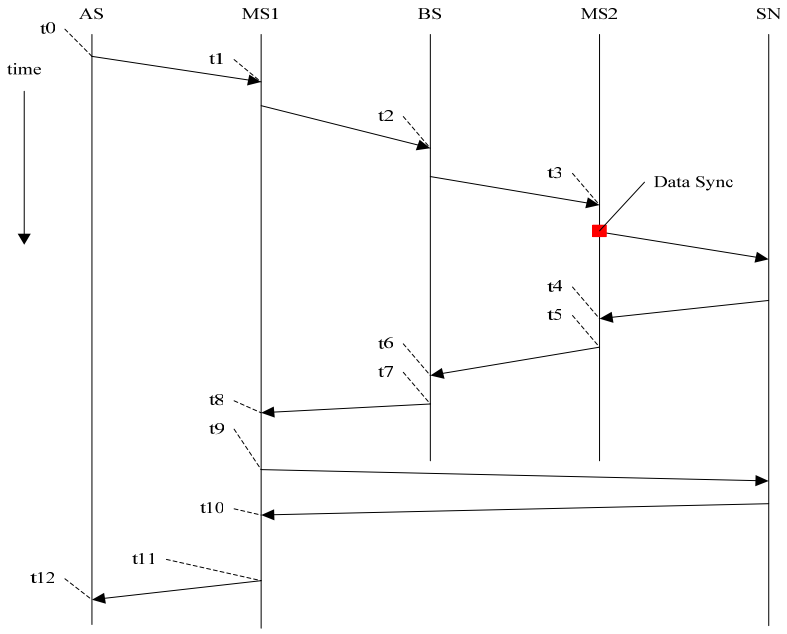

图 5.2 元数据迁移协议时序图

元数据迁移协议过程描述如下：

1. AS 向 MS1 发出硬连接、删除或者移动文件可能的分布式元数据请求;
2. MS1 通过自己的元数据分布信息缓存查找元数据分布信息，获得结果 (foo，not-on-me)。MS1 向 BS 请求获得该元数据的宿主权限；
3. BS 检查 foo 当前的宿主服务器信息，获得（foo, MS2）。BS 以 MS1 的名义要求 MS2 释放索引节点 foo 的宿主权限；
4. MS2 在能够完成权限释放的情况下，将 foo 对应的文件索引节点和元数据块更新到存储设备 SN。foo 的元数据更新完成后，MS2 更改自己维护的元数据分布记录（foo, MS2）→ (foo, MS1)，并返回结果 BS；
5. BS 更改元数据分布记录（foo, MS2）→ (foo, MS1)，赋给 MS1 宿主权限。返回结果给 MS1；
6. MS1 更改元数据分布记录（foo, not-on-me）→ (foo, onme)。然后从 SN 读取索引节点 foo。最后，MS1 按照请求的语义，完成元数据请求的处理。返回结果给 AS。

5.3.6 迁移协议的影响分析

BWMMS 的元数据迁移协议对系统的影响包括两个方面：

1）对元数据请求分布的逻辑性的影响。元数据分布策略的逻辑性考虑，将目录 bar 下的活跃文件 foo1，foo2，……，尽量与目录分布在一起，所有需要同时访问 bar 和文件的请求，如 lookup 等，将由单个元数据服务器完成。当目录 bar 从原来的服务器 MS1 迁移到新的服务器 MS2 后，可以由单个服务器完成处理的 lookup 等请求，将变成需要 MS2 和 MS1 协同才能处理。

2）对元数据服务器负载公平性的影响。在进行元数据请求分布决策时，各个元数据服务器的负载作为决策参数进行考虑。在迁移发生前，各个服务器的负载相对平衡。迁移发生后，所有原来由 MS1 处理的目录关联的元数据请求，将转移到 MS2 处理。这导致 MS1 的负载降低，而 MS2 的负载将升高，导致服务器间的负载不平衡。

尽管元数据迁移对系统有以上两个方面的负面影响。但是，元数据请求统计比例、BWMMS 元数据分布管理的动态重分布特性、以及 BWMMS 的负载决策方法，使得其影响尽可能地减到最小。

尽管元数据迁移可能导致集中请求成为跨服务器请求，影响元数据的逻辑性。但是，1）仅在文件移动请求中，目录迁移后还可能继续使用。统计结果表明，文件移动请求所占比例<0.01%[Hendricks2006]，对系统的影响甚微；2）尽管极小比例的文件移动请求将可能导致跨服务器请求的出现。但是，由于 BWMMS 元数据请求分布管理的动态重分布特性，在通过 MS2 访问 bar 下面的、分布在 MS1 的文件 foo1、foo2 之前，MS1 可能已经根据元数据活跃性，释放掉 foo1、foo2 的管理权限。这样，可以通过请求分布决策，foo1、foo2 可以重新分布到 MS2，减少 bar 被迁移带来的跨服务器请求数量。由于与具体应用密切相关，较难通过实验评估元数据请求分布管理的动态重分布特性对元数据迁移的积极影响程度。

对于公平性的影响，系统能够通过有效的负载信息分析，选择处理请求的服务器。通过 BS 全局的决策，元数据将从预期负载较重的服务器，向预期较轻的服务器迁移，避免服务器负载不平衡的加重，可能进一步促进服务器间负载的平衡。

5.3.7 迁移协议的容错分析

迁移协议容错分析基于的模型是多个进程构成的、通过异步消息进行通信的进程集合。尽管系统的可能故障包括网络因故障分割为多个子网、服务器硬件故障、或者系统软件故障等多个方面，其最终表现为在既定时间内，消息发起进程没有得到消息接收进程的答复。在没有收到答复情况下，消息发起进程可能重发请求，也可能放弃请求。协议必须能够保证系统不会因为一对进程间的消息重发、或者消息丢弃而出现不一致。根据图 5.2，协议容错分析包括 MS1-BS，BS-MS2，MS2-SN 和 MS1-SN 之间的通信过程。

1. MS2 和 SN 之间。MS2 需要将缓存中的元数据写回到设备。在成功更新设备后，更改元数据的分布信息。当在指定时间内没有收到 SN 的回应时，如果 MS2 重新写设备，此时系统仍然仅有 MS2 拥有该元数据的宿主权限，所以不存在覆盖掉其他服务器写回的有效元数据的风险；如果 MS2 放弃写设备，它认为协议处理出错，不会释放元数据的宿主权限。即使 SN 已经将元数据写回到设备，后面的写操作覆盖掉它，也不会丢失有用信息。
2. BS 和 MS2 之间。如果 BS 再次发送消息，要求 MS2 释放元数据权限。MS2 收到重发的消息，检查自己缓存的元数据分布信息后，将由于没有宿主权限而不再写设备，仅返回“正确处理”结果给 BS。如果 BS 放弃重发，它将返回“协议错误”给 MS1。如果在发送“协议错误”结果给 MS1 后，收到 MS2 返回的“协议正确处理”消息，BS 按照协议正确处理，更新元数据的分布信息，等待

MS1 收到“协议处理错误”结果后的重试。

3. MS1 和 BS 之间。如果 MS1 再次发送消息，BS 根据元数据分布信息，返回“协议正确”处理结果。如果 MS1 放弃重发消息。那么，它将返回“请求处理出错”给应用，应用根据需要可以重新发起，或者放弃。
4. MS1 和 SN 之间，由于前面的协议过程已经正确更改 MS2、BS 和 MS1 的元数据分布信息。此时，仅有 MS1 拥有元数据写权限。所以，MS1 和 SN 间任意次的读元数据请求都将获得有效的元数据信息，不会出现不一致。

总之，元数据迁移协议不仅能够保证从存储设备正确读写元数据，还能够保证“同一个活跃元数据，有且仅有一个宿主元数据服务器”。并且，由于 BS 的信息是所有 MS 记录的宿主权限项信息的并集。所以，元数据分布信息管理机制和策略能够支持 BS 故障、MS 都不能有故障，或者 BS 存活、任意 MS 故障的情况。对于 BS 和任意 MS 同时故障的支持，还需要进行深入的研究。

5.4 与两阶段提交协议对比分析

BWMMS 与两阶段提交协议的对比分析包括消息数量和更新磁盘次数，对比的请求采用跨服务器的文件创建、删除、以及涉及 4 个元数据的文件移动请求的最坏情况，对比结果如表 5.3 所示，其中的数据对格式是“(通信消息数量,设备内容更新次数)”。由于通过网络刷写大量数据到存储服务器的时间占整个协议处理时延的比例非常大，设备内容更新次数的变化，反映协议的效率变化。

表 5.3 BWMMS 动态迁移协议和两阶段提交协议对比结果

	两阶段提交协议	动态迁移协议	时间减少比例
创建文件	(2,2)	(1,1)	50%
删除文件	(2,2)	(2,1)	50%
移动文件	(6,4)	(6,3)	25%

从表中可以看出，BWMMS 动态迁移协议至少可以获得 25%的协议时延减少。并且，BWMMS 动态迁移协议不需要在协议过程中以同步写方式保存重要状态，其错误恢复也不需要多个服务器通过复杂的错误恢复协议协同，对系统的正常性能影响较小。

5.5 元数据迁移协议的影响评估

本节从不同比例的跨服务器请求对系统影响的角度，评估元数据迁移协议的影响。评估用例采用修改过事务构成的 postmark。原始 postmark 是每进行一次读文件所有内容/追加写文件内容的同时，还需要创建一个指定大小的文件/删除已有文件。这种模式中，创建/删除必定出现，称之为“100%”模式。后面的表述用“α”表示一次事务中创建/删除出现的概率,，比如α=5%表示 100 个事务中将有 5 个创建/删除的出现。本测试评估了α为 0%、3%、5%、20%和 100%共 5 种情况，其结果如图 5.3 所示。

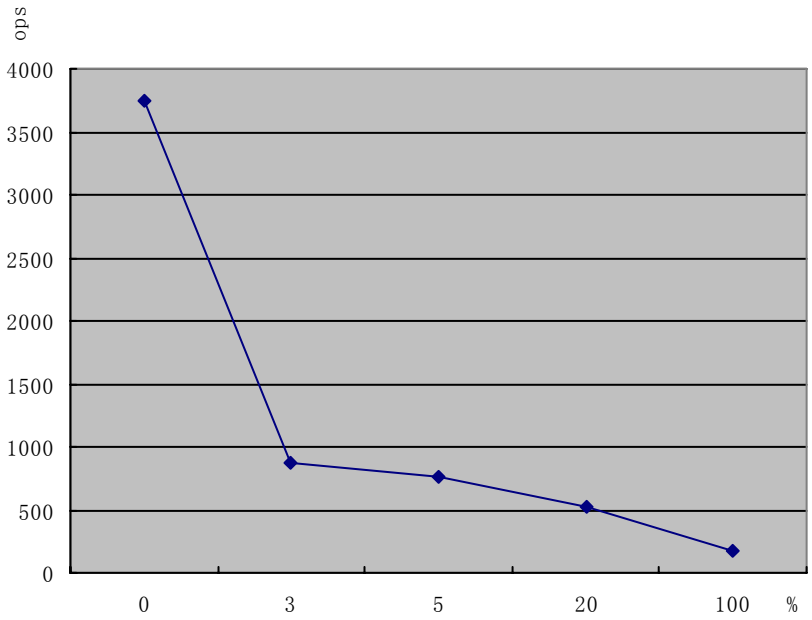

图 5.3 跨服务器请求比例的影响评估结果

表 5.4 是各种比例测试的服务器开销情况。

表 5.4 元数据迁移对服务器负载的影响

	0%	3%	5%	20%	100%
元数据迁移请求数目	0	31	48	59	65
MS 与 BS 平均通信时间(ms)	0.84	0.91	1.51	2.51	3.57
MS 聚合服务时间(s)	44.23	48.78	60.64	80.10	200.43
平均 RPC 处理速度	50,017.94	47,569.93	39,511.64	36,522.79	28,745.29

表中数据表明，随着文件创建/删除比例的增加，各个服务器需要迁移元数据的请求数量增加，导致 BS 用于元数据迁移请求处理的开销增大，影响 BS 处理 MS 的元数据分布决策请求，导致“MS 与 BS 的平均通信时间”呈增加趋势。并且，由于 MS 在处理元数据迁移请求过程中，对 AS 请求的响应不够，导致 AS 请求超时的概率增加，“平均 RPC 处理速度”呈现降低的趋势。NFS 的 trace 分析结果[Hendricks2006]表明，需要通过迁移完成处理的跨服务器请求的比例大约在 0.01%，其本身对系统的影响不会很大，但阻塞正常请求。所以，如果 BS 采用独立线程处理迁移请求，它们对正常元数据分布请求的阻塞影响将大幅度降低，元数据请求的处理能力将得到很大提高。

5.6 本章小结

在集群环境中，更改文件系统的跨服务器请求的原子性将影响文件系统的一致性。尽管其所占比例非常小，不到 1%。但是，传统的两阶段提交等协议，对系统性能和可恢复性的影响都比较大，探讨轻量级原子性保证协议显得非常必要。

以集中共享存储架构和对称元数据服务器结构为支持，BWMMS 提出“通过改变活跃元数据的宿主，完成元数据在服务器间的迁移，跨服务器请求涉及到的活跃元数据集

中到单个服务器”的动态迁移协议。在本地文件系统技术支持下，元数据请求由单个服务器集中处理。对比分析表明，相对于传统的两阶段提交协议而言，在最坏情况下，BWMMS 动态迁移协议也可以获得 25%左右的处理时延降低。并且其错误恢复更加简单，对系统的影响更加微小。

本章最后通过实验评估元数据迁移对系统的影响。对于跨服务器请求比例非常小的应用，BWMMS 采用的轻量级元数据迁移协议，对系统整体性能的影响非常微弱。结果同时表明，跨服务器请求的比例将影响系统的处理能力，要求在未来研究中进一步加强对不同特征的应用模式研究，促进系统进一步的演进。

第六章 元数据请求并发与同步控制

请求的并发和同步控制是分布式系统的重要研究内容。如何控制请求的并发，避免系统出现死锁是分布式系统必须解决的重要问题。在 BWMMS 中，通过 BS 转发元数据迁移请求的轻量级跨服务器请求原子性保证协议，加重系统的死锁可能。如何通过控制元数据请求的并发和同步，完成死锁的检测和消除是本章的主要研究内容。

BWMMS 包括元数据分布信息管理和文件系统元数据访问两个层次的并发。通过将元数据分布信息和文件系统元数据的同步控制分离，BWMMS 在元数据分布信息层进行请求的同步控制，通过分析系统可能的请求并发情况，尽可能地并发元数据分布信息请求和元数据请求，提升系统的扩展能力。

6.1 问题描述

分布式系统的死锁问题，来源于分布式计算环境中的并发进程对共享资源的竞争[Gray1981][Massey1986][Lee2001][Singhal1989][Krivokapic1999][Kshemkalyani1999]。死锁预防、死锁避免、以及死锁检测和消除[Wang1998][Terekhov1999][Gonzales1999][Ling2006]是现有解决分布式系统死锁问题的三种主要方式。分布式死锁检测和消除策略以其不需要预防分布式死锁的出现、动态地检测和消除死锁的特点，成为分布式系统死锁问题解决的主要趋势。

明确请求之间存在的时间和因果关系[Lamport1978][Schwarz1994]，有助于分布式系统的全局镜像维护、死锁问题的检测和消除。在分布式计算领域，已有大量的针对请求因果关系的研究[Fishburn1985][Dielh1992][Katz1990][Meldal1991][Schmuck1991][King2003][Zhu2003]。

投机执行可以提升系统的处理性能[Chang1999][Fraser2003][Nightingale2006][Jefferson1987][Franklin1996][Zhang1999]。它记录进程间存在的数据依赖。在进程依赖的数据不可用时，它根据预测的数据提前执行，提高进程的并行度。当依赖的数据出现最终结果后，只有在预测错误时，它才根据最终结果修正投机执行的进程。

在 BWMMS 中，分布式元数据请求通过元数据迁移，转换成集中请求处理。为简化元数据迁移协议，元数据迁移请求通过集中决策服务器转发。第 4 章已经验证 AS 的元数据请求间的并发。本章主要关注元数据迁移请求和正常元数据请求之间的并发控制问题。系统可能存在的请求并发如图 6.1 所示。

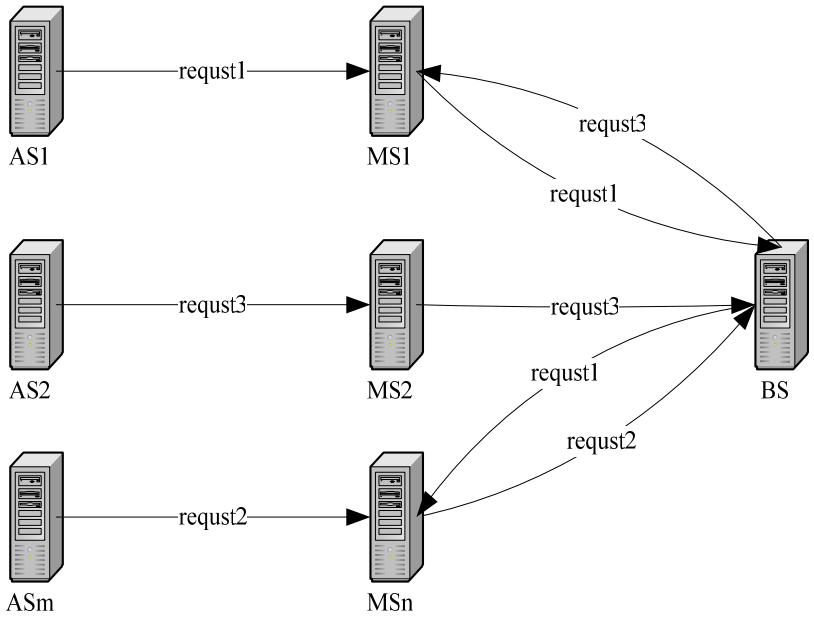

图 6.1 BWMMS 可能的请求并发

如图 6.1 所示，系统可能在两个层次出现死锁：

- 服务进程层的死锁。系统由于各个服务器没有空闲的服务进程处理新的请求而出现死锁。在图 6.1 中，BS、MS1 都可能出现这种情况。假定 MS1 在发出 request1 之后、响应 request3 之前，没有服务进程可以处理新的请求，BS 转发来的 request3 将不能得到处理。而 BS 向 MS1 转发 request3 后，它也没有服务进程处理 request1。那么，MS1 和 BS 间相互等待，导致死锁的出现。
- 文件系统层的死锁。多个元数据请求访问的元数据可能存在关联性。如果请求在拥有一个元数据的宿主权限后，还需要获得其他元数据的宿主权限，元数据请求之间就可能出现循环等待元数据宿主权限的情况。

通过通信协议的超时重发机制，服务进程层的死锁能够得到解决。本章主要分析并解决文件系统层次的死锁问题。BWMMS 特定的服务器交互模式加深了系统出现死锁的可能。在传统分布式系统的请求并发控制基础上，还需要结合文件系统元数据请求的特定语义，进一步分析可能的请求并发，检测和消除系统可能的死锁。

6.2 BWMMS 元数据请求并发控制

在 BWMMS 环境中， MS 需要处理 AS 的正常文件系统元数据请求和 BS 转发的元数据迁移请求。对 MS 而言，BS 转发的元数据迁移请求类似于计算机系统的中断。它要求元数据迁移请求能够中断 AS 的正常元数据请求流。请求中断过程如图 6.2 所示。

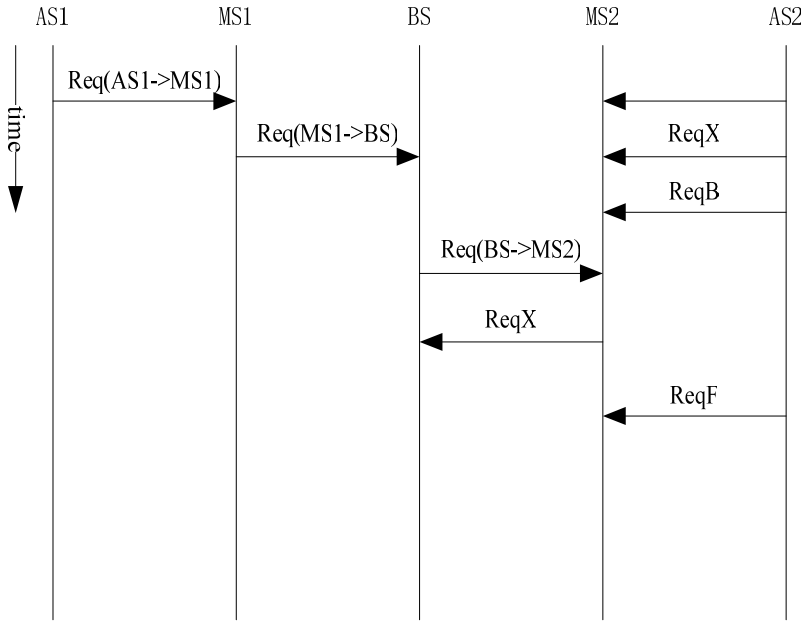

图 6.2 元数据迁移中断正常元数据请求流

假定 MS2 是 ObjX 当前的宿主。在 MS2 上，它需要处理 AS2 请求 ObjX 的正常的元数据请求流（…, ReqX, ReqB, ReqF, …）。在收到 BS 转发的 Req(BS->MS2)时，该迁移 ObjX 的请求需要中断正常的元数据请求流，迁移 ObjX 的 Req(BS->MS2)插入到 ReqB 和 ReqF 之间。迁移成功后，ObjX 的宿主为 MS1。所以，ReqB 及其前的正常元数据请求可以处理，而 ReqF 则需要等待迁移请求完成后，再返回给 AS2。AS2 重新向新的宿主 MS1 发起请求。

6.3 常规文件元数据的迁移

6.3.1 常规文件并发算法

AS 对常规文件的正常元数据请求包括文件索引节点的内容访问、文件访问权限验证、文件数据的设备块号映射、文件数据的并发访问控制信息获取等。这些请求针对单个索引节点进行，不会在持有该文件宿主权限的情况下，再向 BS 请求其他元数据的宿主权限。MS 迁移常规文件元数据的目的包括删除文件、创建到文件的硬连接、或者文件是文件移动请求的旧文件或者新文件等。所以，常规文件的请求并发仅需要控制元数据是否能被迁移，处理元数据迁移过程中收到的元数据请求等情况。依据表 4.1，常规文件元数据迁移同步控制的相关信息如表 6.1 所示。

表 6.1 常规文件并发控制的数据结构

```
struct mdt_entry {
        u32 me_state;
        u64 me_timestamp;
        atomic_t me_modicnt;
};
```

Me_modicnt 用来控制本元数据是否可以被迁移，其值表示不希望元数据被迁移的请求数目，初始值为 0。仅当 me_modicnt 为 0 时，元数据才可能被迁移。

Me_timestamp 是元数据迁移不能进行时，记录的预留时间范围，初始值为 0。在其值表示的时间内，新的正常元数据请求不能使用该元数据，需要等待迁移请求的重新到来或者超时。当超过时间范围，元数据仍然没有被迁移时，me_timestamp 被清除，正常元数据请求可以继续使用该元数据。

Me_state 记录元数据迁移请求的进行过程。在元数据迁移请求能够进行但还没有开始时，它将 me_state 赋值 ME_FLUSHING。新的元数据请求检查到 ME_FLUSHING 时，需要等待元数据迁移请求的完成。

综合上面所述，常规文件迁移同步的控制包括两个部分：

1. MS 对 AS 正常的元数据请求处理过程的同步部分的算法为：

表 6.2 常规文件正常元数据请求部分的同步算法

```
check：
  if（(me_state & ME_FLUSHING)为 0，并且本 MS 没有元数据宿主权限）{
      返回新的宿主 MS 信息给 AS；
  }
  if （me_timestamp>0 并且还没有超时，或者(me_state & ME_FLUSHING)为 1) {
      等待；
      跳到 check；
  }
  if （me_timestamp > 0，但已经超时）{
      me_timestamp = 0；
      唤醒因”me_timestamp > 0”而等待的元数据请求；
  }
  增加 me_modicnt；
  完成元数据请求处理；
  减少 me_modicnt；
  /*唤醒元数据迁移请求*/
  if (me_modicnt 为 0){
       唤醒等待”me_modicnt > 0”的进程；
  }
```

2. MS 处理常规文件迁移请求部分的算法为：

表 6.3 常规文件元数据迁移部分的同步算法

```
if (me_modicnt > 0){
      me_timestamp = 超时时间值，并返回“繁忙，再试”；
}
me_state |= ME_FLUSHING；
me_timestamp = 0；
唤醒等待 me_timestamp 的用户元数据请求；
将元数据写回到存储设备，并更改元数据分布信息；
me_state &= ~ME_FLUSHING；
唤醒等待”me_state & ME_FLUSHING 为 1”的进程；
```

6.3.2 算法活跃性验证

通过扩展图 4.4，常规文件迁移并发控制算法的 Petri Net 表示如图 6.3 所示，起始状态只有位置 Pldt 拥有 1 个标记。

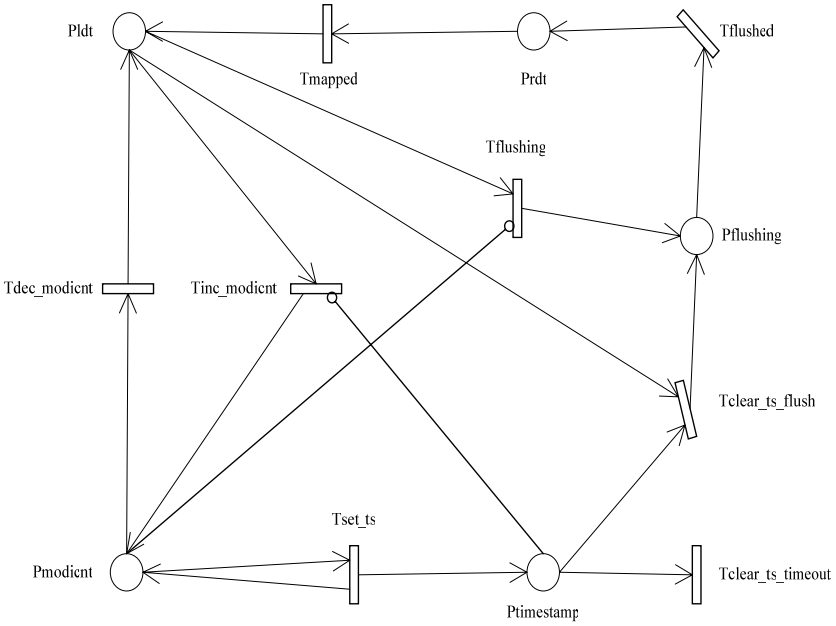

图 6.3 常规文件并发控制算法 Petri Net 表示

其中的位置含义如表所示。

表 6.4 常规文件并发控制图的位置含义

位置名	含义
Pldt	具有文件元数据的宿主权限
Prdt	不具有文件元数据的宿主权限
Pmodicnt	元数据的 me_modicnt 的数目，0 表示没有元数据请求限定“该元数据不能被迁移”
Ptimestamp	Me_timestamp 是否被设置
Pflushing	元数据迁移请求是否在进行

其中变迁的含义为：

表 6.5 常规文件并发控制图的变迁含义

变迁名	含义
Tdec_modicnt	减少 me_modicnt，允许元数据被迁移
Tinc_modicnt	增加 me_modicnt，禁止元数据被迁移
Tflushing	正在进行元数据迁移，将元数据写回到存储设备
Tclear_ts_flush	元数据迁移请求的处理清除 me_timestamp
Tclear_ts_timeout	超时，没有元数据迁移请求清除 me_timestamp，由正常的元数据请求清除。
Tflushed	完成元数据迁移
Tmapped	重新从 BS 获得宿主权限

位置 Pmodicnt 到变迁 Tflushing 间的禁止弧表示如果已经有正常使用元数据的请求

存在，元数据迁移请求不能进行。位置 Ptimestamp 到变迁 Tinc_modicnt 间的禁止弧表示元数据迁移请求阻塞后续的正常元数据请求。变迁 Tset_ts 仅在有正常请求存在的情况下进行，表明元数据迁移请求需要设置 me_timestamp，并返回。

根据表 4.2 的 Petri Net 可达树求解算法，合并可达树中的相同节点得到如图 6.4 所示的可达图，节点序列为（Pldt, Pmodicnt, Ptimestamp, Pflusing, Prdt）。图 6.4 中每个节点都可以通过一定的变迁序列到达其他节点，常规文件迁移的并发控制算法是活跃的。

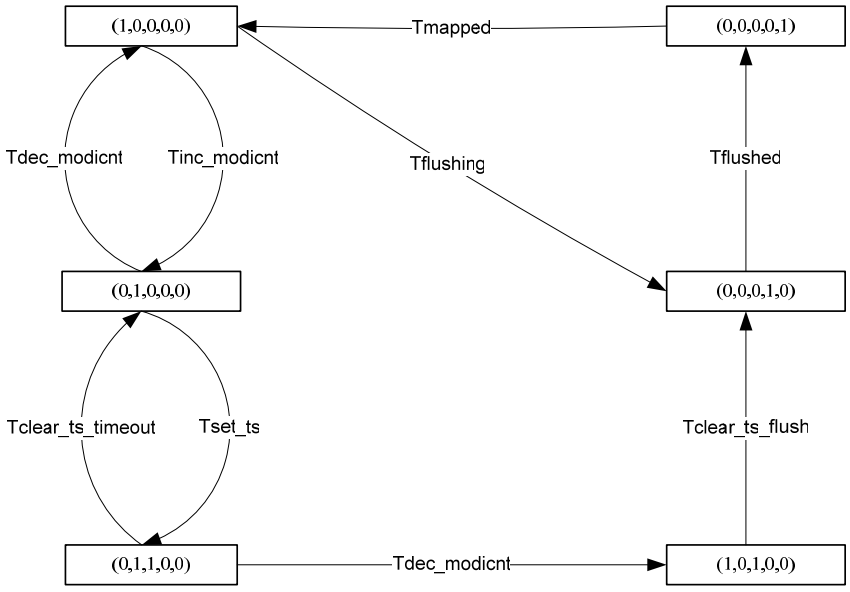

图 6.4 常规文件并发控制可达图

6.4 目录元数据的迁移

由于目录关联的元数据请求种类繁多，目录的元数据迁移带来的并发非常复杂。并且，由于元数据迁移可能导致可以集中完成的请求转换成分布式请求，将影响系统的请求处理效率。所以，目录的迁移决策需要尽可能准确地判定目录是否可以迁移。

被迁移目录可能参与的元数据请求包括：

- 创建文件操作。被迁移目录是创建文件请求的父目录。在为新创建的文件决定元数据宿主过程中，如果父目录被迁走，新创建文件可能还没有写到存储设备，其他 MS 不能使用新创建文件的最新内容。
- 查找文件操作。查找操作可能需要为被查找的文件请求元数据宿主映射。由于查找操作不会更改任何元数据，所以其父目录可以被迁走，不需要进行任何的更新设备内容的操作。
- 删除文件操作。文件的删除需要更改目录和被删除的文件。在文件删除进行前，可能需要向 BS 申请被删除文件的宿主权限。所以，父目录的迁走将导致请求的失败，AS 需要向新的父目录宿主重新请求删除文件。

- 目录是文件移动请求需要的元数据之一。被迁移目录可能是文件移动请求的参与者。

以上请求的共同点是它们在拥有目录宿主权限的前提下，与 BS 通信，查询或要求其他元数据的宿主权限。因为目录的迁移请求通过 BS 转发，所以，请求之间形成环路的可能性较大，容易导致系统死锁。

在常规文件并发控制基础上，为提高目录迁移判断的精度，需要比较目录关联的正常元数据请求与目录迁移原因之间的优先级，决定是否迁移目录的元数据。表 6.6 给出了目录迁移需要的同步结构。

表 6.6 目录并发控制的数据结构

```
struct mdt_entry {
        u32 me_state;
        u64 me_timestamp;
        atomic_t me_modicnt;
        atomic_t me_lookupcnt;
        atomic_t me_unlinkcnt;
        atomic_t me_linkcnt;
        atomic_t me_renamecnt;
};
```

在 me_state，me_timestamp 和 me_modicnt 的基础上增加了 me_lookupcnt、me_unlinkcnt、me_linkcnt 和 me_renamecnt，用来记录目录关联的文件查找、删除、创建、以及移动请求的数目。目录迁移算法将通过它们完成目录元数据请求的并发控制。

6.4.1 目录并发算法

表 6.7 目录迁移的优先级判定表

关联 \ 迁移	将被删除的目录	文件移动的目标父目录或者旧目录	文件移动的新目录
创建文件	文件创建成功将导致该目录非空，迁移请求可以提前预测“目录非空”，删除目录请求失败	可以迁移，但需要等待创建请求完成	文件创建成功将导致该目录非空。由于文件移动要求已存在目录必须为空，所以可以提前预判文件移动请求失败
删除文件	可以迁移，但需要等待文件删除请求完成。如果目录有多个孩子，可以提前预测目录非空，删除目录请求失败	可以迁移，也可以抢断文件删除请求。文件删除请求需要重新检查父目录的宿主，向新的宿主请求删除文件	可以迁移，也可以抢断文件删除请求。如果目录有多个孩子，可以提前预判文件移动请求失败
查询文件	可以迁移，可以抢断。查询文件请求不需要向新的宿主重新发起文件查询请求		
文件移动的源父目录	可以迁移，但需要等待移动请求完成。如果目录由多个孩子，可以提前预判删除目录请求失败	由于文件移动请求的串行化，这种情况的并发不会出现	
文件移动的目标父目录	文件移动成功将导致该目录非空，可以提前预判删除目录失败		
文件移动的旧目录	不可能出现这种并发，因为目录只可能有一个父目录		
文件移动的新目录	可以迁移，但需要等待文件移动请求完成		

目录是否迁移的判断策略需要考虑两个问题：1）目录迁移对请求的处理有利，尽量减少无效的迁移。比如，如果被删除的目录非空的话，删除目录的请求不能完成，其迁移将成无效迁移。2）防止目录在服务器间反复迁移，形成元数据迁移的“乒乓”现象。基于以上考虑，目录关联的元数据请求和目录迁移原因的优先级对比如表 6.7 所示。

结合表 6.6 的数据结构，根据表 6.7 的优先级判断，目录迁移的并发控制算法为：

1. MS 处理 AS 的元数据请求的并发算法为：

表 6.8 目录正常元数据请求部分的同步算法

```
check：
  if（(me_state & ME_FLUSHING)为 0，并且没有元数据宿主权限）{
        返回给 AS 新的 MS 信息；
  }
  if （me_timestamp>0 并且还没有超时，或者（me_state & ME_FLUSHING)为 1) {
        等待；
        跳到 check；
  }
  if （me_timestamp>0 但已经超时）{
        me_timestamp = 0；
        唤醒等待 me_timestamp 的用户元数据请求；
  }
  增加 me_modicnt；
  if (需要访问 BS) {
        按照请求类别增加 me_lookupcnt/me_linkcnt/me_unlinkcnt/renamecnt;
        减少 me_modicnt；
        与 BS 通信；
        按照请求类别减少 me_lookupcnt/me_linkcnt/me_unlinkcnt/renamecnt;
        跳到 check;
  }
  完成元数据请求处理；
  减少 me_modicnt；
  /*唤醒元数据迁移请求*/
  if (me_modicnt 为 0)
        唤醒等待 me_modicnt 的进程；
}
```

2. 目录元数据迁移算法为：

表 6.9 目录元数据迁移部分的同步算法

```
if (me_modicnt > 0){
    me_timestamp =  超时时间；
    返回 BUSY；
}
/*比较请求原因和关联操作的优先级。*/
if (出于移动文件的目的迁移目录){
    if (me_linkcnt > 0) {
        if (被迁移目录是移动文件请求的将被覆盖的新目录，并且目录不为空) {
            返回”目录非空”;
```

```
            } else {
                me_timestamp = timeout_second;
                返回"BUSY";
        }
        if (me_unlinkcnt > 0, 并且被迁移目录是移动文件请求的将被覆盖的新目录、且目录的孩子数
目大于 1) {
            返回 "目录非空";
        }
}
if (出于删除目录的目的迁移){
      if (me_linkcnt > 0) {
          返回 "目录非空"
      }
       if (me_renamecnt > 0 || me_unlinkcnt > 0) {
          me_timestamp = timeout_second;
          返回"BUSY";
       }
}
me_state |= ME_FLUSHING;
me_timestamp = 0;
唤醒等待 me_timestamp 的用户元数据请求;
将元数据写回到存储设备，更改元数据分布信息;
me_state &= ~ME_FLUSHING;
唤醒等待"me_state & ME_FLUSHING =1"的进程;
```

6.4.2 算法活跃性验证

由于目录并发判断涉及到的请求种类很多，本部分仅验证目录迁移请求与删除目录下文件请求的并发控制，其 Petri Net 表示如图 6.5 所示，起始状态时位置 Pldt 有 1 个标记。

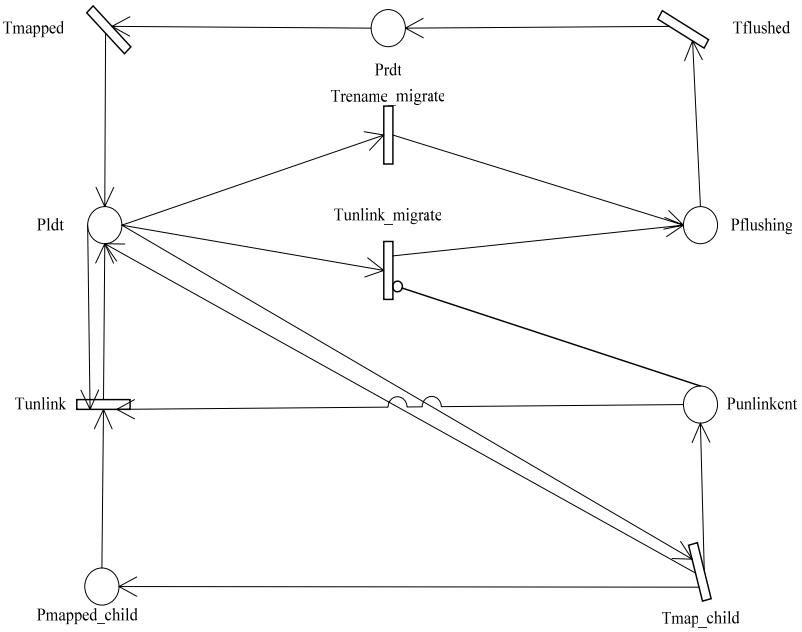

图 6.5 目录迁移与目录子文件删除的并发 Petri Net 表示

图 6.5 中位置的含义为：

表 6.10 目录并发控制图的位置含义

位置名	含义
Pldt	具有宿主权限
Prdt	没有宿主权限
Punlinkcnt	为删除文件目的，请求被删除文件的宿主权限。父目录的 unlinkcnt 记录关联的删除文件请求数目
Pmapped_child	获得被删除文件的宿主权限
Pflushing	元数据正在迁移

图 6.5 中变迁的含义为：

表 6.11 目录并发控制图的变迁含义

变迁名	含义
Tmap_child	请求被删除子文件的宿主权限
Tunlink	删除文件
Tunlink_migrate	处于删除目的请求迁移目录元数据
Trename_migrate	处于文件移动目的请求迁移目录元数据
Tflushed	文件迁移完成
Tmapped	获得元数据宿主权限

图 6.5 的下半部是删除目录下子文件的状态图。在删除文件请求 BS，获得被删除文件的宿主权限前，将目录的 me_unlink 加 1。请求完成后，me_unlinkcnt 减 1。位置 Punlinkcnt 到变迁 Tunlink_migrate 的禁止弧表明，一旦存在本目录下的文件删除操作，期望删除本目录的请求必须在目录下的删除请求完成之后才能执行，以提高迁移的精准度。

变迁 Trename_migrate 和变迁 Tunlink_migrate 的发生导致位置 Ponme 没有标记，变迁 Tunlink 不能进行。这表明如果在为删除请求迁移被删除文件过程中，删除操作的父目录被迁移走，则删除操作不能再进行，需要向新的宿主重新发起请求。

根据图 6.5，得到其可达图如图 6.6 所示，图中节点序列为（Pldt, Punlinkcnt, Pmapped_child, Pflushing, Prdt）。

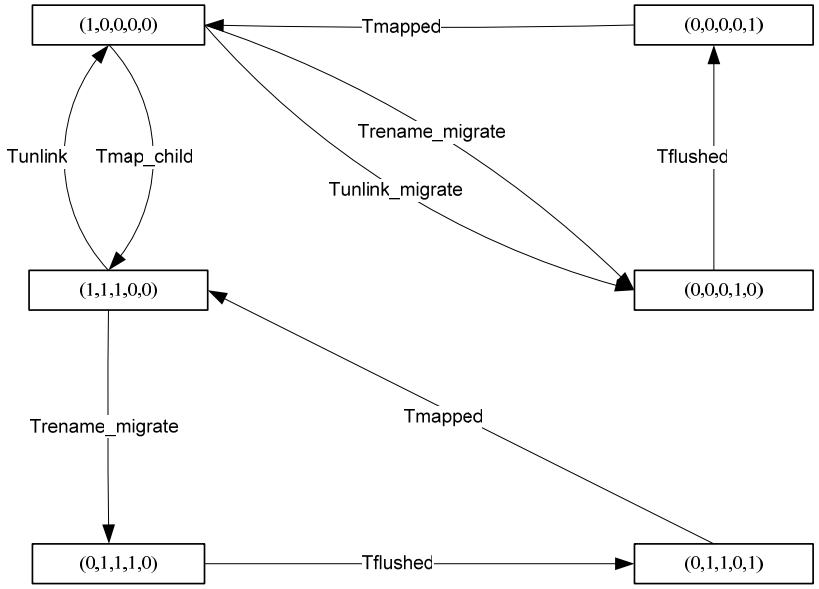

图 6.6 目录迁移与目录子文件删除并发可达图

通过类似的方法，其他的目录迁移并发控制算法的活跃性也能得到证明。综合所有的情况，目录迁移的并发控制算法是活跃有效的。

6.5 本章小结

并发进程对共享资源的竞争是导致系统死锁的根源，分布式系统的非集中结构使得系统的死锁检测和消除尤为困难。已有研究从死锁预防、死锁避免和死锁的检测与消除等角度对系统的死锁问题进行研究。请求的因果关系发现是解决系统死锁的另一种思路，通过分析并发请求之间可能存在的因果关系，为系统死锁问题解决提供支持。投机执行是增加请求并行度、提高系统处理效率的有效方法。

在这些研究成果基础上，BWMMS 在各个 MS 上以元数据请求的元数据对象为核心，结合文件系统元数据请求的语义，分析可能的请求并发。MS 需要同步的请求可以分为 AS 的正常元数据请求之间的同步、其他 MS 的元数据迁移请求与 AS 的正常元数据请求的同步两类。

AS 间的元数据请求的同步对象是元数据，它可以利用文件系统的同步机制完成请求的并发控制。而迁移请求与正常请求的同步对象是元数据的分布信息，需要在元数据分布信息层进行控制。

文件的正常元数据请求由单个元数据服务器完成，并且文件迁移目的简单。所以，通过元数据分布信息协助，采用简单的并发算法，即可完成文件迁移与正常元数据请求的并发控制。在本章中，常规文件元数据的并发控制算法通过 Petri Net 描述并验证。

目录关联的正常元数据请求种类繁多，且迁移的目的较多。并且，目录的迁移对系统的性能有明显的影响，需要提高其迁移决策的准确性。为提高迁移决策的精准度，迁

移目的和正常元数据请求被赋予一定的优先级。并发控制算法考虑元数据请求的语义，比较请求之间的优先级，保证目录元数据迁移的有效性。由于并发情况较多，在描述目录的并发控制算法基础上，本章仅采用 Petri Net 描述和验证了目录迁移请求与删除目录子文件请求的并发控制的活跃性，其他情况同样可以进行验证。

总之，BWMMS 通过比较请求之间的优先级，结合元数据请求语义，根据优先级判断请求能否被抢断，从整体上提高了系统的请求并行度，增强了系统的处理效率。

第七章 元数据服务扩展能力评估

结合元数据存储服务和元数据请求服务关键问题的解决，BWMMS 得到原型实现，并运用到 BWFS 中。本章将介绍原型系统实现中的关键问题，然后评估系统对传统应用和新兴应用的扩展支持能力。结果表明，尽管原型系统在实现上存在一些不足，BWMMS 采用的动态元数据服务机制和策略，既能够很好支持传统应用，也能够很好支持新兴应用，为 BWMMS 未来进一步发展提供了佐证。

7.1 系统原型实现

在系统服务器的物理构成上，原型系统包括若干应用服务器（AS），若干元数据服务器（MS），1 个绑定服务器（BS）和若干个存储设备（SN）。服务器的软件模块及其交互流程如图 7.1 所示。

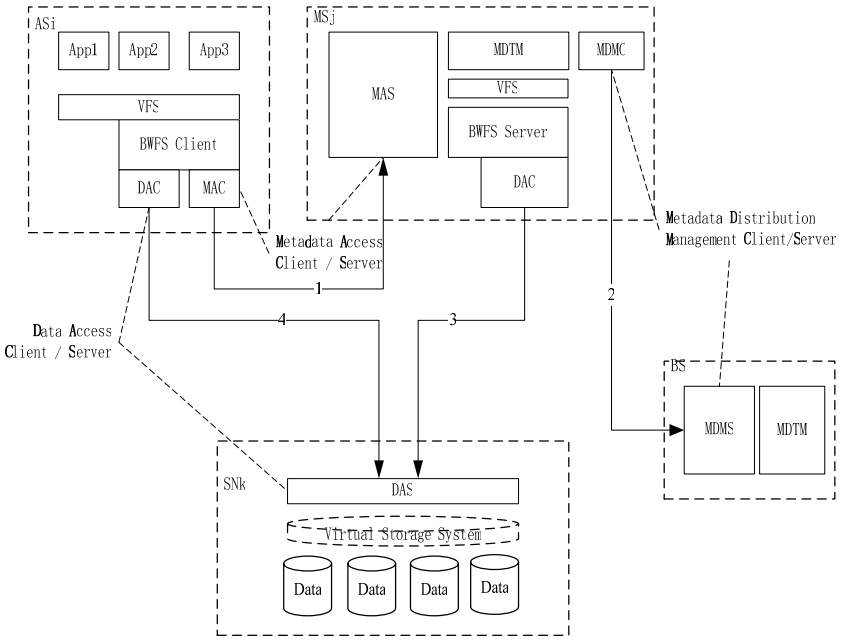

图 7.1 BWMMS 原型系统结构

AS 和 MS 使用数据访问客户端模块（Data Access Client, DAC）分别从存储设备读取文件的数据和元数据，数据访问服务器模块（Data Access Server, DAS）是存储设备上负责处理数据和元数据请求的服务。AS 通过元数据访问客户端模块（Metadata Access Client, MAC）与 MS 上的元数据访问服务器模块（ Metadata Access Server, MAS）交互，获取文件元数据信息。元数据分布信息管理客户端(MDMC)和元数据分布信息管理服务器(MDMS)是 MS 和 BS 上用来完成元数据分布信息管理的软件模块。

对于涉及多个元数据的元数据请求，其处理过程如下：

1. 查询文件的元数据请求发送给父目录的宿主 MS。如果被查询文件当前分布在其他 MS，用户从被查询文件的宿主 MS 读取该文件的元数据信息。
2. 创建文件的元数据请求发送到父目录的宿主 MS。该元数据服务器按照文件创建语义完成请求的处理后，再向 BS 查询新创建文件的宿主 MS。根据 BS 返回结果，如果本 MS 不是新创建文件的宿主 MS，则需要将创建操作涉及到的文件（父目录和新创建文件）写回到 SN，并返回请求结果给 AS。
3. 删除文件的元数据请求发送到父目录的宿主 MS。根据需要，将请求涉及的元数据通过元数据迁移协议集中到该服务器。单个 MS 按照请求要求的语义完成请求处理。
4. 移动文件的元数据请求发送到文件移动请求的源父目录的宿主 MS，由它负责集中文件移动请求需要的元数据，并按照文件移动语义完成请求处理。为避免多个文件移动请求导致文件系统目录树结构的不一致，系统在同一时间只处理一个文件移动请求。所以，文件移动请求的处理，首先需要从 BS 申请文件移动请求处理权限，其他的文件移动请求必须在本请求完成后才能进行。

为避免复杂的系统交互过程，对于动态迁移协议，系统实现没有支持“选择完成元数据请求处理的服务器”，元数据请求由收到该请求的服务器负责处理。原型系统也没有实现“被迁移的目录重新迁到源 MS”的协议。因为 BWMMS 根据元数据活跃性管理元数据请求的分布，被迁移的元数据是否有必要迁到原来的 MS，需要在今后的工作中，根据实际应用的数据进行进一步的分析。

BWFS 的客户端模块是内核模块，没有服务进程数量的限制。MS 的 MDS 以内核线程方式存在，可以指定内核线程的数量，默认值为 8 个。为了实现的简单和减少进程调度的开销，BS 实现为基于 RPC 通信的单个服务进程结构的后台程序。元数据分布决策算法的服务器负载以各个 MS 当前分布的活跃元数据数目为参数。

在后续章节中，我们将通过实验，评估系统在传统的创建/删除文件、用户访问共享目录测试文件集和用户访问私有目录测试文件集等情况的聚合 I/O 性能的扩展情况，也将评估对新兴的生物计算应用的支持能力。所有测试均在同一测试平台进行。测试使用的服务器是 Intel® Xeon® 3.4GHz,3.0GB 的内存，运行 RedHat® Linux® 8.0。服务器通过千兆以太网络互联。文件系统的构成包括 1 个存储设备，若干元数据服务器和 1 个绑定服务器。存储设备采用 120GB 的 SATA 硬盘提供数据和元数据的存储。

7.2 元数据服务扩展能力评价指标

根据 1.1 节定义的元数据服务扩展能力，本部分元数据服务扩展能力的评估针对两种情况的系统进行：1）服务器数目固定，变化客户端的元数据请求负载，获得聚合元数据吞吐率；2）客户端负载固定，变化服务器数目，获得聚合元数据吞吐率；

假定每个客户端的元数据请求吞吐率为$Throughput(i)$，则系统聚合元数据吞吐率为：$\sum_{i=1}^{n} Throughput(i)$。为衡量系统的扩展能力，系统规模变化带来的加速比计算公式为：$\sum_{i=1}^{n} Throughput(i) / \sum_{i=1}^{1} Throughput(i)$，即 n 个评测对象系统规模的聚合元数据吞吐率与一个评测对象的元数据吞吐率的比值。

系统扩展能力的评估将不仅针对传统的 Web Service、E-Mail 等，也将针对逐渐占据重要地位的生物信息计算等新兴应用。

第 7.3 节通过能够模拟传统应用访问特征的 benchmark，测试系统在不同数据共享模式下的扩展能力。测试结果表明，BWMMS 能够提供具有较好扩展能力的元数据服务。同时，它还表明，由于原型系统实现尚存的不足，跨服务器请求比例对系统扩展能力的影响还较为明显，需要优化系统的实现。

第 7.4 节评估系统对新兴的生物信息计算应用的支持能力。评估采用生物信息计算常用的 BLAST（Basic Local Alignment Search Tool）完成。BLAST 首先将库文件初始化，然后在库文件中进行查找，明确输入序列的相似性。评估结果表明，随着服务器数目增加，元数据服务时间减少速度在 20%左右，为 BLAST 提供具有较好扩展能力的分布式文件系统元数据服务。

7.3 对传统应用的扩展支持评估

为了充分评估系统对传统应用扩展的支持能力，本节将针对共享和私有两种访问模式，评测元数据服务的扩展能力。同时，还将通过频繁的文件创建/删除，明确系统可能存在的不足。

7.3.1 共享使用模式的扩展能力评估

只读共享是用户间常见的数据共享方式。本节通过固定客户端负载，增加服务器数目，评估服务器数目增加对用户访问速度、BS 负载的影响，验证元数据请求分布策略对服务器规模的扩展支持。其期望值是，服务器数目的增加，能够将元数据请求分布到多个服务器，并获得聚合元数据吞吐率的提升。

测试首先在共享目录下生成测试文件集，然后将系统重新启动，消除系统元数据缓存和元数据分布信息缓存的影响。

7 个客户端同时使用“find . -exec stat {} \;”命令访问共享目录下的文件。测试所用文件集是 Redhat® Linux® 8.0 的 Linux 内核代码 Linux-2.4.18-14，其目录和文件总数为 7,246 个。元数据服务器数目分别是 1、2、4 和 6 个。测试使用 Linux®的 time 工具收集

“find . -exec stat {} \;”所用时间。系统聚合吞吐率如表 7.1 所示。

表 7.1 共享模式聚合吞吐率随服务器变化

服务器数目	1	2	4	6
聚合吞吐率(ops)	178.44	179.73	180.05	180.63
吞吐率加速比	1	1.0072	1.0090	1.0123

从表 7.1 的聚合吞吐率随服务器数目的变化、聚合吞吐率加速比看，元数据服务的扩展能力不太明显。表 7.2 的服务器负载和图 7.2 的 BS 元数据分布决策进程的 CPU 占用率表明，由于测试环境规模的限制，客户端的元数据请求没有给系统提供足够的压力，导致聚合元数据吞吐率没有表现出随服务器数目的显著提升。

从表 7.2 的各个服务器的负载情况来看，由于 BWMMS 的元数据分布策略有效地将元数据请求分布到多个元数据服务器，各个服务器能够并行高效地处理元数据请求，元数据服务器的聚合服务时间呈现显著降低的趋势。

表 7.2 共享模式的服务器负载

服务器数目	1	2	4	6
MS 聚合服务时间(s)	0.61	0.60	0.35	0.32
分布信息缓存命中率(%)	97.67	96.85	96.56	96.70

在共享访问模式中，元数据分布决策将由多个客户端的访问驱动。当多个用户请求访问同一个元数据时，某个客户的请求驱动元数据分布决策，其他请求可以利用该请求获得的分布信息，完成元数据请求的处理。所以，元数据分布信息缓存发挥出极大的作用，如表 7.2 所示，元数据缓存命中率在 96.5%以上，大大缓解 BS 的压力。从图 7.2 的绑定服务器的元数据分布决策进程的 CPU 占用率来看，在元数据分布信息缓存的作用下，元数据服务器数目的增加不会显著提升后端绑定服务器的压力。

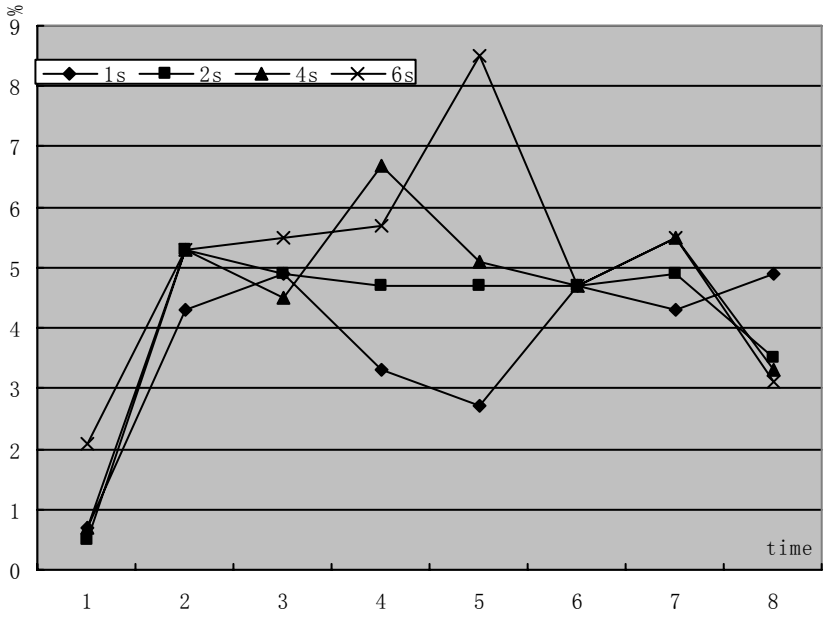

图 7.2 共享模式 BS 的 CPU 占用率

综合表 7.2 和图 7.2，BWMMS 能够很好支持，更多客户端以共享模式访问同一文件集，元数据服务表现出较强的扩展能力。

7.3.2 私有使用模式的扩展能力评估

相对共享模式而言，私有使用模式是系统更为常见的使用方式。在这种模式中，用户主要访问各自私有目录中的文件，用户间没有文件的共享。本节评估固定服务器规模下、用户负载变化带来的聚合吞吐率的变化，以及固定用户负载情况下、元数据服务器规模变化带来的聚合吞吐率变化两种情况。

7.3.2.1 负载变化的扩展能力评估

系统包括 6 个元数据服务器。测试采用 postmark[postmark2002]进行，测试文件集的大小范围为 4kB-16kB，每次文件 I/O 的块大小为 4kB。在每个客户的测试文件集生成阶段，postmark 首先创建 10 个目录，然后在这 10 个目录中总共创建 10000 个文件。在事务处理阶段，它将进行 50000 次事务[3]请求，事务处理速度通过“指定的事务数（50000）/事务阶段所用时间”计算而得。评估共测试 1、2、4 和 6 个客户端共 4 种情况，结果如表 7.3 所示。

表 7.3 私有模式聚合吞吐率随客户端变化

客户端数目	1	2	4	6
聚合吞吐率(ops)	367	522	935	1,320
吞吐率加速比	1	1.4223	2.5477	3.5967

从加速比看，在私有模式下，BWMMS 还不能支持聚合元数据吞吐率随客户端的线性扩展能力。

表 7.4 私有模式既定服务器的 MS 负载情况

客户端数目	1	2	4	6
MS 聚合服务时间(s)	0.80	3.68	15.79	40.72
平均 MS 和 BS 通信时间(ms)	6.65	7.30	7.98	8.74
平均元数据迁移时间(ms)	17.86	20.89	35.14	55.51

但是，结合表 7.4 的服务器负载情况的分析结果表明，其受限于后端绑定服务器当前的实现结构。从表 7.4 的“MS 和 BS 的平均通信时间”和“MS 聚合服务时间”看，BS 的决策结果能够将元数据请求分布到多个服务器，元数据请求负载将由多个服务器分担，服务器进程的聚合服务时间表现出增长幅度放缓的趋势。

但是，“元数据迁移的平均时间”表明，客户端数目的增加，将导致活跃元数据数目的增加，增加元数据服务器间交互的复杂度。由于 BS 仅以各个元数据服务器当前活跃元数据数目为决策参数，并且元数据缓存管理策略没有及时地将本 MS 真实活跃的元数据数目提供给 BS，元数据分布策略将关联元数据分布到不同元数据服务器的概率增大，

[3] Postmark 的事务定义见第三章分布策略对比评估部分。

系统的跨服务器请求数目出现增长趋势，需要 BS 转发的请求数目增加，导致平均元数据迁移时间的增长。

7.3.2.2 服务器变化的扩展能力评估

为验证元数据服务器增加带来的系统聚合元数据处理能力的提升，本节评测在 7 个客户端下元数据服务器数目从 1、2、4 到 6 个时的系统聚合事务吞吐率的变化趋势。7 个客户端采用 postmark 进行测试。测试文件集由位于 10 个目录下的 10000 个文件构成，每个文件的大小范围为 4kB-16kB，每次文件 I/O 的块大小为 4kB，测试完成 50000 次事务请求。系统聚合事务吞吐率如表 7.5 所示。

表 7.5 私有模式聚合吞吐率随服务器变化

服务器数目	1	2	4	6
聚合吞吐率(ops)	328	633	997	1,564
吞吐率加速比	1	1.9299	3.0396	4.7683

表 7.6 的元数据服务器负载情况，能够说明影响客户端聚合元数据吞吐率的原因。

表 7.6 私有模式既定客户端的 MS 负载情况

服务器数目	1	2	4	6
MS 聚合服务时间(s)	84.72	58.42	52.87	43.78
MS 和 BS 平均通信时间(ms)	9.32	7.43	8.03	9.07
平均元数据迁移时间(ms)	0	15.48	15.87	15.86

从表 7.6 来看，通过元数据分布策略的作用，元数据请求分布到多个服务器，使得元数据服务器的聚合服务时间呈现降低趋势。

随着服务器数量的增加，元数据服务器之间的交互复杂化。由于 BS 当前的单一进程结构实现，MS 的请求在 BS 串行执行，导致“MS 和 BS 的平均通信时间”呈现增加趋势。同时，出于平衡服务器负载目的，请求分布策略将增加跨服务器请求的数目，元数据迁移出现的概率增加，导致“元数据迁移的平均时间”呈现出增加的趋势。这将进一步阻塞 BS 的处理，导致系统性能的降低。

综合以上分析，对私有使用模式而言，尽管不能提供线性的扩展能力，BWMMS 仍然能够支持客户端和服务器数目增加带来的聚合元数据吞吐率的增加。目前其扩展能力表现不够好的关键原因在于，BS 的实现还不能很好地支持请求的并行处理。

7.3.3 创建删除空文件的扩展能力评估

尽管统计结果表明，文件创建/删除的比例非常少，仍可能存在需要大量文件创建/删除的应用。本节评估系统对密集的文件创建/删除应用的支持情况。

每个客户端在私有目录下创建 10000 个空文件，创建完成后，删掉这些空文件。测试使用 Linux® 的 time 工具收集所用时间。客户端的请求处理速率 OPER(i)=10000/TIME(i)，其中，i 表示第 i 个客户。TIME（i）为测试所耗的时间，单

位为秒。OPER（i）是通过计算得到的 I/O 速率，单位为每秒完成的操作数目(ops)。将每次测试所有客户端的请求处理速率相加，得到聚合请求处理速率：

$$\mathrm{OPER} = \sum_{i=1}^{n} \mathrm{OPER}(i), \text{其中} n \text{为客户数量}$$

系统测试 6 个元数据服务器规模下，客户数量分别为 1、2、4 和 6 个的创建和删除的聚合吞吐率，结果如表 7.7 所示。

表 7.7 创建/删除聚合吞吐率随客户端变化

客户端数目	1	2	4	6
创建聚合吞吐率(ops)	263.50	264.13	235.54	235.16
创建吞吐率加速比	1	1.002	0.894	0.892
删除聚合吞吐率(ops)	222.27	201.82	193.18	187.64
删除吞吐率加速比	1	0.908	0.869	0.844

表 7.8 是创建和删除空文件测试过程的服务器负载情况。图 7.3 是 BS 服务进程的 CPU 占用率随时间的变化曲线图。

表 7.8 创建/删除的 MS 负载情况

客户端数目	1	2	4	6
MS 聚合服务时间(s)	0.16	0.33	0.75	1.12
MS 和 BS 平均通信时间(ms)	0.570	0.601	0.752	0.741

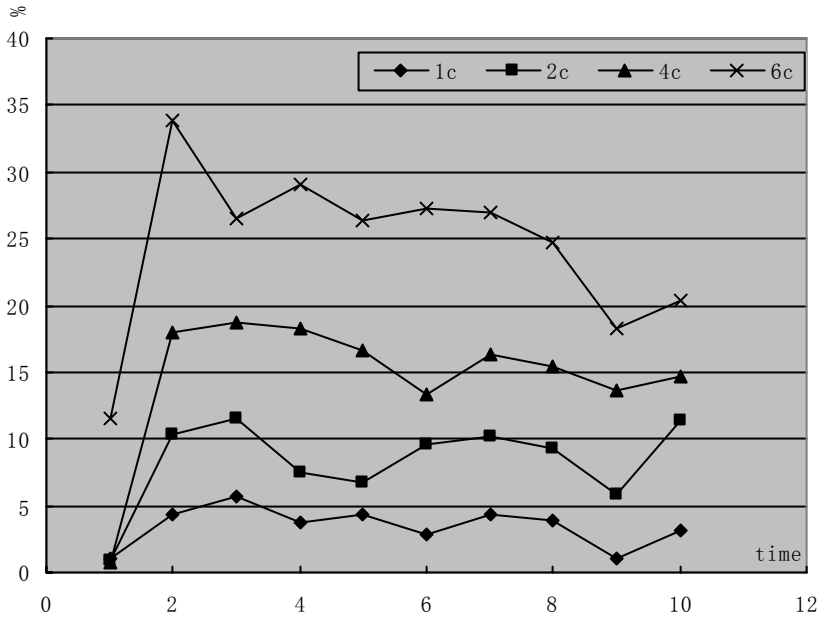

图 7.3 创建/删除 BS 的 CPU 占用率

结合表 7.8 的创建/删除文件的服务器负载和图 7.3 的 BS 服务进程的 CPU 占用率可以看出，由于文件创建过程需要为新创建的文件决定宿主映射决策，元数据缓存将不能起到提高处理效率的作用，每一个文件创建都需要 BS 参与。所以，随着应用负载的增加，BS 的负载相应地增加。BS 的处理能力和元数据创建的处理过程，限制文件创建聚

合吞吐率的提升，文件创建的聚合吞吐率随客户端负载增加而降低。

同样，由于跨服务器的文件删除请求需要通过 BS 完成元数据的迁移。尽管可能的元数据迁移请求数目很少，但它可能阻塞 BS，其他请求不能得到处理。所以，随着客户端数量的增加，系统聚合的文件删除吞吐率表现出下降的趋势。

总之，目前 BWMMS 对具有频繁的文件创建/删除的应用支持还不够,需要根据应用的实际需要进行优化。

7.4 对新兴应用的扩展支持评估

生物信息计算是近几年新兴的高性能计算应用。本节通过生物计算广泛采用的 BLAST（Basic Local Alignment Search Tool），评估 BWMMS 的元数据服务对新兴的生物计算应用的支持能力。生物处理仪获取规模庞大、文件大小较小的原始数据，经过计算处理，获得待比对的序列。BLAST 是用来比对生物序列的一级结构(如不同蛋白质的氨基酸序列或不同基因的 DNA 序列)的算法，使用待比对序列进行序列相似度的比对。BLAST 主要表现为固定数量文件的反复读请求。最坏情况下，为完成一条序列的比对，它需要读取目标数据库所有的记录。其需要的计算时间很长，长时间的文件读写请求表现出庞大的元数据请求。

本测试采用的序列库的大小为 2GB。通过 BLAST 软件包提供的实用程序 FORMATDB 的参数控制，序列库被格式化为 60 组子库，每组包括后缀分别为 phr、pin 和 psq 的 3 个文件,每组文件的大小和约为 40MB。目标库共有 180 个文件,总和为 2.4GB 左右。测试使用 4 个客户端完成 211,359 条序列的比对，总共需要 435,214,997 次元数据访问 RPC 通信。测试采用 Linux®的 time 实用程序收集比对过程需要的时间。为明确系统的扩展能力，分别测试元数据服务器数量为 1、2 和 3 个的情况。其结果如表 7.9 所示：

表 7.9 BLAST 聚合处理时间

服务器数目	1	2	3
BLAST 聚合时间(h)	260.32	230.27	208.64
聚合时间递减速度(%)	0	11.54	19.85

从 BLAST 运行总时间看，其时间随元数据服务器数目增加的降低比例不很明显，大约在 10%左右。但是，收集到的时间分布表明，BLAST 绝大部分工作在用户态进行。

通过系统态时间，扣除与元数据访问不相干的时间，得到大致的元数据访问时间，才能够更合理地表现元数据请求开销。可以从系统态时间扣除的主要时间是数据访问时间。从统计到的数据访问看，应用读取文件数据的请求数目大致在 210,000 个，每次读取 1MB 的文件。测试初期获得的 BWFS 文件数据读取速度是 37.99MB/s。根据这两个值进行计算，读取文件数据的时间大概是 1.54 小时左右。所以，从系统态时间中扣除这部分时间后，得到元数据访问的大致时间，如表 7.10 所示。

表 7.10 BLAST 元数据访问时间分析

服务器数目	1	2	3
系统态时间(h)	5.89	4.96	4.01
计算的元数据访问时间(h)	4.35	3.42	2.47
元数据时间递减速度(%)	0	21.38	31.82

元数据服务器的服务时间是各个元数据服务器处理元数据请求花费的时间。结合它来分析系统的元数据处理，能更好地说明元数据服务能力。测试的聚合服务时间如表 7.11 所示：

表 7.11 BLAST 元数据服务器聚合服务时间分析

服务器数目	1	2	3
MS 聚合工作时间(s)	20,475. 15	15,726.22	12,895.66
聚合工作时间递减速度(%)	**0**	**23.19**	**37.02**
MS 与 BS 通信时间(s)	0.138	0.264	0.369
MS 与 SN 通信时间(s)	3,269.36	6,386.92	6,801.37
MS-SN 时间所占比例(%)	**15.97**	**40.61**	**52.74**

从服务器聚合服务时间看，元数据服务器数目的增加能够带来近 20%的 MS 聚合服务时间的减少。BS 当前的单一进程结构，将导致元数据服务器的请求的串行处理，MS 与 BS 的通信时间将随服务器数目的增加而增加，这反映在“MS 与 BS 通信时间”。尽管绝对值的变化不大，但其随服务器的变化比例的增长速度非常快。同时，由于元数据访问的小读小写特征，频繁的元数据读写，将导致元数据读写时间随服务器数目增加而延长，这反映在“MS 与 SN 通信时间”。“MS-SN 时间所占比例”是“MS 与 SN 的通信时间”占整体元数据服务时间的比例，它表现出快速增加趋势。这表明，通过改善 BS 的实现结构、优化元数据服务器的元数据读写性能，系统的元数据服务效率将得到很大提升，可能获得近线性的元数据服务扩展能力。

7.5 本章小结

元数据存储服务和元数据请求服务关键问题的解决方案得到原型实现。系统实现平台是 RedHat® Linux® 8.0，由 1 个 SN、1 个 BS、多个 MS 和多个 AS 构成。

本章评估其对传统应用和新兴应用的扩展支持能力。

针对传统应用的评估表明，从结构和策略上讲，BWMMS 的对称服务器结构和动态元数据请求分布策略，能够很好地支持系统规模的扩展需要。系统能够将负载尽可能均衡地分布到各个元数据服务器，提升系统的处理能力。目前的系统原型实现存在限制扩展能力的不足，且对具有大量频繁的文件创建/删除的应用的扩展支持很差。

针对生物信息计算应用 BLAST 的评估结果表明，服务器数目增加能够带来元数据请求处理时间近 20%左右的减少。在未来工作中，通过改善 BS 的实现结构、优化元数据读写访问，系统将可能获得近线性的元数据服务扩展能力。

综上所述，BWMMS 采用的基于集中共享虚拟存储的文件系统元数据存储模型，基于文件动态访问的元数据请求分布管理机制，基于元数据动态迁移的文件系统一致性协议，基于请求优先级、投机执行请求的请求并发控制机制，能够很好地支持系统的扩展需求。同时，系统在实现上还存在不足，需要进行大量的优化工作。

第八章 结束语

存储技术、计算技术和网络技术的进步共同催生网络化存储技术。集中化网络存储有利于提高系统的存储资源利用率、存储资源管理的扩展能力和系统的可管理性，为用户提供方便的存储资源共享。

在网络存储系统基础上，分布式文件为用户提供便捷的数据共享。文件系统由元数据和数据构成，元数据访问效率是文件数据访问性能的关键因素。应用规模的不断增大和新应用的不断涌现，对分布式文件系统的元数据服务提出越来越高的要求。在新的存储系统架构下，针对分布式文件系统元数据服务的扩展能力进行深入研究，对推动网络存储技术的发展具有至关重要的意义。

8.1 本文工作总结

本文着重于分布式文件系统元数据服务的可扩展性研究，期望以多个服务器组成的服务器集群方式提供具有较强扩展能力的分布式文件系统元数据服务，以促进网络存储系统更好地支持多种应用环境，提高其适用性。

通过分析已有的研究成果，结合用户的元数据访问负载的特点，本研究获得了高扩展能力的存储资源组织、管理和使用，动态高效的元数据请求分布管理，简单有效的文件系统一致性协议，以及高效的元数据请求并发控制算法等研究成果。具体包括：

1. 基于集中虚拟存储模型的存储资源组织、分布式层次化的存储资源管理、完全共享的存储资源使用模式的文件系统元数据存储服务。虚拟存储支持存储资源的独立扩展，层次化的管理机制支持存储资源的有效利用，完全共享的存储资源使用模式解决非常困难的数据放置问题，支持动态灵活的元数据请求服务。在此基础上，元数据的存储和访问彻底分离，元数据的存储不再限制其请求的分布，元数据的存储问题得到有效解决。
2. 动态灵活的元数据请求分布管理机制和策略，支持元数据服务器规模和用户元数据访问负载的动态变化。通过仅管理活跃元数据的请求分布，大幅度缩小请求分布管理的对象集，支持请求分布管理的扩展。根据元数据活跃性管理元数据请求分布，为用户元数据访问负载的动态变化提供有效支持。采用层次化的分布管理机制，支持元数据服务器规模的动态扩展。元数据请求分布策略结合请求的动态性和元数据间客观存在的访问相关性，结合元数据请求的处理效率，尽可能地平衡服务器的负载。
3. 简单高效的更改文件系统的跨服务器请求的原子性保证协议。对称的元数据服

务器系统结构、用户元数据访问的统计特征、动态灵活的元数据请求分布管理，为简单高效的元数据请求一致性协议提供足够的支持。通过动态改变活跃元数据在服务器间的分布，完成元数据在服务器间的迁移，将跨服务器请求集中化，利用相关技术保证文件系统一致性，减小协议对系统正常请求处理的影响。

4. 基于系统的请求并发情形分析，提出简单的请求同步控制算法。利用元数据请求分布信息，结合元数据请求语义，充分分析可能的请求并发，根据请求的优先级进行判定，尽可能地并发元数据请求。
5. 根据上述结果，设计和实现了 1 个应用到 BWFS 的原型系统（蓝鲸多元数据服务器系统，BWMMS），并通过实验验证其扩展能力的有效性。

8.2 未来工作展望

通过本研究，分布式文件系统可扩展元数据服务研究的系统平台得以实现，为未来的研究提供必需的支持。初步的评估结果表明，系统采用的集中共享虚拟存储架构和基于数据共享的对称服务器结构，灵活的元数据请求分布管理，轻量级的文件系统一致性协议和有效的请求并发控制机制等，能够为系统可扩展的分布式文件系统元数据服务提供必要的支持。但是，由于本研究主要关注分布式文件系统元数据服务的扩展能力。在后续的研究中，还需要在研究内容的深化、系统适用范围的推广和新研究领域的拓展等方面进行大量的工作。

8.2.1 研究内容的深化

本研究着重于分布式文件系统元数据服务的扩展性的研究，主要集中在元数据存储服务、集群的服务器结构、元数据请求分布管理和元数据请求并发控制等关键问题。在很大程度上，系统的元数据请求处理性能和扩展能力依赖于元数据请求分布决策算法，如何提供有效的元数据请求分布决策是首先需要进一步深化的问题。除此之外，还需要在系统的实现上进行优化。

1. 元数据负载预测模型的扩展和评估。动态的元数据请求分布管理的核心是元数据服务器负载的预测。在保证元数据请求的处理效率的前提下，尽可能地将用户的负载分布到多个服务器。元数据服务器负载的自学习模型、元数据负载参数的丰富、元数据分布策略的优化，都是需要进一步深入的研究内容。
2. 集中决策服务器结构的改进。从系统的实验评估结果看，单一的集中决策机制和元数据分布信息缓存，能够很好支持元数据活跃性变化不频繁的应用的扩展需求。在未来工作中，为了支持更大规模的系统，还需要进一步考虑集中决策服务器结构的扩展。通过服务器集群或层叠方式组织多个决策服务器，扩展集中决策服务器结构，进一步提高系统的扩展能力。

3. 针对应用特性的元数据请求分布管理策略。BWMMS 根据的应用特征统计结果中，文件创建非常稀少，其处理过程时延较大。但是，可能存在如频繁的文件创建/删除、临时文件等应用，其元数据活跃性变化频繁，元数据分布信息缓存的作用将非常有限，导致集中决策服务器的压力非常大。为很好地支持更多类型的应用，元数据分布策略将扩展其考虑的因素范围。不仅需要考虑元数据服务器的负载、单个请求涉及的元数据之间的相关性，还需要抽取更多应用的访问特征，从单个用户角度考虑元数据请求的分布。
4. 大规模系统的验证及优化。本研究在相对较小系统规模下，对系统的扩展能力进行了初步的验证。在未来工作中，还需要从理论验证和实际环境使用两个方面，验证在更大规模系统中的扩展能力，并根据结果对系统进行优化。
5. 元数据读写的优化。由于元数据表现出频繁的小读小写特征，为降低请求服务器与存储服务器间的通信开销，需要结合应用的特征和系统的管理策略，优化请求服务器的元数据读写性能。

8.2.2 系统适应范围的扩展

本研究依赖于已有研究结果的结论：文件系统的活跃数据比例不大，活跃文件的数目比例在 5%-20%左右。需要更改文件系统的分布式元数据请求所占比例非常小，在请求数量比例上不超过 5%。基于这些结论，BWMMS 以动态灵活的机制决定元数据请求的分布，通过元数据动态迁移、集中化处理跨服务器请求等方式保证文件系统的一致性。

但是，已有研究针对的应用范围毕竟有限，如何拓展 BWMMS 的适用范围将成为下一步研究的重点。目前学术界已有的研究结果主要针对软件开发、web 服务、E-Mail 和其他商业应用环境，提取用户的数据访问特征，且研究成果的发布时间已经比较早。对于新兴的应用，如遥感信息处理、生物计算等，还没有显著的成果发表。为更好地支持网络存储系统在多种环境的应用，还需要通过与相关研究的合作，提取应用的 I/O 特征，改进和完善文件系统的相关问题。通过对更多应用特征的综合，推广系统的适用范围，推动网络存储技术的进一步发展。

8.2.3 研究领域的扩展

本研究主要关注的是分布式文件系统元数据服务的性能和扩展能力，没有过多的关注元数据服务的高可用问题。

分布式系统的高可用性是系统广泛使用的必备条件。必须得到充分的研究，系统才能很好地满足更多应用的需求。分布式系统的高可用性要求通过系统服务器间以某种方式进行互补，以性能的降低或者部分功能的不能使用为代价，提供服务的可用性。系统的故障只会带来性能的降低、或者系统功能的部分缺陷，整个系统依然能够继续为系统用户提供服务。

本研究仅仅通过各个服务器维护的元数据分布信息实现相互冗余，部分地支持元数据分布管理服务的高可用功能，保证同一个元数据不会同时具有多个宿主服务器。目前仅仅能够做到：1）通过 MS 缓存的分布信息合并，能够恢复出现故障的 BS 的元数据缓存信息；或者：2）各个 MS 出现故障但 BS 正常工作时，各个 MS 恢复后，可以通过请求驱动的方式重新缓存元数据分布信息。系统还不能支持 MS 和 BS 任意故障的情况，BWMMS 的高可用问题还需要深入的研究。

总之，本研究通过可扩展分布式文件系统元数据服务关键问题的解决，完成分布式文件系统可扩展元数据服务的平台构建工作，还需要通过未来的工作提升系统的可用性，并加强系统在高可用性等领域的研究，推动网络存储系统在更大应用范围的适用性。

参考文献

[Adya2002] A. Adya, W. J. Bolosky, M. Castro, G. Cermak. FARSITE: federated, available, and reliable storage for an incompletely trusted environment. In Symp. On Operating Systems Design and Implementation (OSDI 2002), USENIX Association, 2002:1-15

[Agrawal2007] N. Agrawal, W. J. Bolosky, J. R. Douceur, J. R. Lorch. A Five-Year Study of File System Metadata. 5th USENIX Conference on File and Storage Technologies (USENIX FAST'07), 2007: 31–45

[Anderson1995]T. E. Anderson, M. D. Dalhin, J. M. Neefe, D. A. Patterson. Serverless Network File Systems. in Proc. of 15th Symp. On Operating System Principles (SOSP 1995), 1995:109-126

[Anderson2000-1] D. C. Anderson, J. S. Chase, A. Vahdat. Interposed request routing for scalable network storage. In 4th Symp. On Operating Systems Design and Implementation (OSDI 2000), 2000

http://citeseer.ist.psu.edu/article/anderson00interposed.html

[Anderson2000-2] D. C. Anderson, J. S. Chase. Failure-atomic file access in an interposed network storage system. In Proceedings of the 9th IEEE International Symp. On High Performance Distributed Computing (HPDC 2000), 2000:157-164

http://citeseer.ist.psu.edu/anderson00failureatomic.html

[Bach1987] M. J. Bach, the Design of the UNIX Operating System, Upper Saddle River, NJ: Prentice Hall, 1987

[Baker1991] M. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, J. Ousterhout. Measurements of a Distributed File System. In Proceedings of the 13th Symp. On Operating Systems Principles (SOSP 1991), 1991:198-211

http://citeseer.ist.psu.edu/ article/baker91measurements.html

[Best2002] S. Best. JFS Log – How the Journaled File System Performs Logging, www.free-soft.org/FSM/english/issue03/sbest.pdf

[Bovet2002] D. P. Bovet, M. Cesati. Understanding the Linux Kernel, 2nd Edition. O'Reilly, Dec. 2002, ISBN: 0-596-00213-0

[Bozman1991] G. P. Bozman, H. H. Ghannad, E. D. Weinberger. A Trace-driven study of CMS file references. IBM Journal of Research and Development, Vol. 35 NO. 5/6, Sep./Nov. 1991:815-828

[Braam1999] P. J. Braam, M. Callahan, P. Schwan. The InterMezzo File System. In Proc. of the 3rd of the Perl Conference, O'Reilly Open Source Convention, 1999

[Braam2002] P. J. Braam. The Lustre Storage Architecture. 2002. http://www.lustre.org

[Brandt2003] S. A. Brandt, E. L. Miller, D. D. E. Long, Lan Xue. Efficient Metadata Management in Large Distributed Storage Systems. Proceedings of the 20th IEEE/ 11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST), 2003:290-298

[Callaghan1995] B. Callaghan, B. Pawlowski, P. Staubach. NFS Version 3 Protocol Specification, Network Working Group, RFC1813, June 1995

[Chang1999] F. Chang, G. Gibson. Automatic I/O hint generation through speculative execution. In Proceedings of the 3rd Symp. On Operating Systems Design and Implementation (OSDI 1999). 1999:1-14

[Chinner2006] D. Chinner, J. Higdon. Exploring High Bandwidth Filesystems on Large Systems. Linux Symposium 2006
http://oss.sgi.com/projects/xfs/papers/ols2006/ols-2006-paper.pdf

[Corbett1996] P. F. Corbett, D. G. Feitelson. The Vesta parallel file system. ACM Transactions on Computer Systems (TOCS), Vol. 14 NO. 3, August 1996:225-264
http://citeseer.ist.psu.edu/ corbett96vesta.html

[Dahlin1994-1] M. Dahlin, C. Mather, R. Wang, T. Anderson, D. Patterson. A Quantitative Analysis of Cache Policies for Scalable Network File Systems. In Proc. of 1994 SIGMETRICS, 1994:150-160
http://citeseer.ist.psu.edu/dahlin94quantitative.html

[Dahlin1994-2] M. Dahlin, R. Wang, T. Anderson, D. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In Proc. of the 1st Symp. On Operating Systems Design and Implementation (OSDI 1994), 1994:267-280
http://citeseer.ist.psu.edu/dahlin94cooperative.html

[Daley1965] R. C. Daley, P. G. Neumann. A General-Purpose File System for Secondary Storage. 1965 Fall Joint Computer Conference, 1965
http://www.multicians.org/fjcc4.html

[Dielh1992] C. Diehl, C. Jard. Interval Approximations and Message Causality in Distributed Systems. Proc. of the 9th Annual Symposium on Theoretical Aspects of Computer Science STACS '92, LNCS 577, 1992:363-374

[Du2001] 杜聪, 徐志伟. COSMOS 文件系统的性能分析. 计算机学报, 2001 年第 7 期, 2001:702-709

[Ellard2003] D. Ellard, J. Ledlie, P. Malkani, M. Seltzer. Passive NFS Tracing of Email and Research Workloads. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03), 2003: 203-216
http://citeseer.ist.psu.edu/ellard03passive.html

[Ellard2004] D. Ellard. Trace-Based Analyses and Optimizations for Network Storage Servers. Harvard Computer Science Technical Report TR-11-04, 2004

[Fan2004] Zhihua Fan, Jin Xiong, Jie Ma. A Failure Recovery Mechanism for Distributed Metadata Servers in DCFS2. Proceedings of the High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference on (HPCAsia'04), 2004:2-8

[Fishburn1985] P. C. Fishburn. Interval Orders and Interval Graphs. Wiley, New York, 1985

[Floyd1989] R. A. Floyd, C. S. Ellis. Directory Reference Patterns in Hierarchical File Systems. IEEE Transactions on Knowledge and Data Engineering, Vol.1 NO.2, 1989:238-247

[Franklin1996] M. Franklin, G. Sohi. ARB: A hardware mechanism for dynamic reordering of memory references. IEEE Trans. On Computers, 45(5), 1996: 552-571

[Fraser2003] K. Fraser, F. Chang. Operating system I/O speculation: How two invocations are faster than one. In Proceedings of the 2003 USENIX Technical Conference. 2003:325-338

[Ganger2000] G. Ganger, M. McKusick, C. Soules. Soft updates: A solution to the metadata update problem in file systems. Transactions on Computer Systems (TOCS), Vol. 18 No.2, 2000: 127-153

[Gibson1997] G. Gibson. File server scaling with network-attached secure disks. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (Sigmetrics '97), 1997:272 - 284
http://citeseer.ist.psu.edu/gibson97file.html

[Glider2003] J. S. Glider, C. F. Fuente, W. J. Scales. The Software Architecture of a SAN Storage Control System. IBM Systems Journal, Vol. 42 NO. 2, 2003:232-249

[Gray1981] J. N. Gray, P. Homan, H. F. Korth, R. L. Obermarck, A Straw Man Analysis of the Probability of Waiting and Deadlock in Database Systems Technical Report RJ 3066, IBM Research Laboratory, San Jose, Calif., 1981

[Gonzales1999] J. R. Gonzales de Mendivil, F. Farina, J. R. Garitagoitia, C. F. Alastruey, J. M. Bernabeu-Auban. A Distributed Deadlock Resolution Algorithm for the AND Model. IEEE Trans. Parallel and Distributed Systems, Vol. 10 NO. 5, 1999:433-447

[Hammond1998] L. Hammond, M. Willey, K. Olukotun. Data speculation support for a chip multiprocessor. In Proc. of the 8th International ACM Conference on Architecture Support for

Programming Languages and Operating Systems (ASPLOS 1998). 1998:58-69

[Hendricks2006] J. Hendricks, S. Sinnamohideen, R. R. Sambasivan, G. R. Ganger. Eliminating Cross-server operations in scalable file systems. Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-06-105, 2006

[Hennessy2002] J. L. Hennessey, D. A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Pub., 2002. ISBN: 1558605967

[Henson2006] V. Henson. KHB: A Filesystems reading list. http://lwn.net/Articles/196292/

[Hertel2003] C. Hertel. Implementing CIFS-The Common Internet File System. Prentice Hall. 2003. ISBN 0-13-047116-X

[Howard1987] J. Howard, M. Kazar, S. Menees, D. Nichols, M. West. Scale and Performance in a Distributed File System. Proceedings of the 11th ACM Symposium on Operating systems principles (SOSP 1987), 1987:1-2

[Hsu2001] W. W. Hsu, A. J. Smith, H. C. Young. I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks---an Analysis at the Logical Level. ACM Transactions on Database Systems (TODS 2001), Vol. 26 NO.1, 2001:96-143

[Huang2005] 黄华. 蓝鲸分布式文件系统的资源管理. 中国科学院研究生院博士学位论文，2005

[Jefferson1987] D. Jefferson, B. Beckman, F. Wieland, L. Blume. Time Warp Operating System. In Proc. of the 11th ACM Symp.On Operating Systems Principles (SOSP 1987), 1987:77-93

[Karamanolis2001] C. Karamanolis, L. Liu, M. Mahalingam, D. Muntz, Z. Zhang. Architecture for Scalable and Manageable File Services. HPL-2001-173, 2001

[Katz1990] S. Katz, D. Peled. Interleaving Set Temporal Logic. Theoretical Computer Science 75, 1990:263-287

[King2003] S. T. King, P. M. Chen. Backtracking Intrusions. In Proc. of the 19th ACM Symp. On Operating Systems Principles (SOSP 2003), 2003:223-236

[Krivokapic1999] N. Krivokapic, A. Kemper, E. Gudes. Deadlock Detection in Distributed Database Systems: A New Algorithm and a Comparative Performance Analysis. J. Very Large Data Bases (VLDB), Vol. 8 NO. 2, 1999:79-100

[Kronenberg1986] N. P. Kronenberg, H. M. Levy, W. D. Strecker. VAXclusters: A Closely-Coupled Distributed System. ACM Transactions on Computer Systems (TOCS 1986), Vol. 4, No. 2, 1986:130-146

[Kryder2006] M. H. Kryder. Future Storage Technologies: a Look beyond the Horizon. A

presentation at the Storage Networking World
http://www.snwusa.com/documents/presentations-s06/MarkKryder.pdf

[Kshemkalyani1999] A. D. Kshemkalyani, M. Singhal. A One-Phase Algorithm to Detect Distributed Deadlocks in Replicated Databases. IEEE Trans. Knowledge and Data Eng., Vol. 11 NO. 6, 1999:880-895

[Lamport1978] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, Vol. 21 No. 7, 1978:558-565

[Lee1996] E. K. Lee, C. A. Thekkath. Petal: distributed virtual disks. In Proceedings of the Seventh international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII), 1996:84-92
DOI= http://doi.acm.org/10.1145/237090.237157

[Lee2001] S. Lee, J. L. Kim. Performance Analysis of Distributed Deadlock Dectection Algorithms. IEEE Trans. Knowledge and Data Eng., Vol. 13 NO. 3, 2001:623-636

[Levy1990] E. Levy, A. Silberschatz. Distributed file systems: concepts and examples. ACM Computing Surveys (CSUR) Volume 22 Issue 4, 1990:321-374

[Li2006] Weijia Li, Wei Xue, Jiwu Shu, Weimin Zheng. Dynamic Hashing: Adaptive Metadata Management for Petabyte-Scale File Systems. 14th NASA Goddard, 23rd IEEE Conference on Mass Storage Systems and Technologies (MSST 2006), 2006

[Ling2006] Yibei Ling, Shigang Chen, Cho-Yu Jason Chiang. On Optimal Deadlock Detection Sheduling. IEEE Transactions on Computers, Vol. 55 NO. 9, 2006:1178-1187

[Lions1996] J. Lions. Lions' Commentary on UNIX® 6th Edition, with Source Code, San Jose, CA: Peer-to-Peer Communications, 1996

[Liu1994] M. Liu, D. Agrawal, A. El Abbadi. The Performance of Two-Phase Commit Protocols in the Presence of Site Failures. Proc. of 24th Intl. Symp. On Fault-Tolerant Computing , 1994:234-243
http://citeseer.ist.psu.edu/article/liu94performance.html

[Luckham1995] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, W. Mann. Specification and Analysis of System Architecture Using Rapide. IEEE Transactions on Software Engineering, 21(4), 1995:336-355
http://citeseer.ist.psu.edu/article/luckham95specification.html

[Lustre-SGSRFP2001] Lustre.org. Statement of Work: SGS File System, 2001
http://www.lustre.org/docs/SGSRFP.pdf

[Massey1986] W. A. Massey. A Probabilistic Analysis of a Database System. ACM

SIGMETRICS Performance Evaluation Rev., Vol. 14 NO. 1, 1986:141-146

[Mckusick1984] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry. A Fast File System for UNIX. ACM Transactions on Computer Systems (TOCS) Vol. 2 NO. 3 , 1984:181-197

http://citeseer.ist.psu.edu/mckusick84fast.html

[Meldal1991] S. Meldal, S. Sankar, J. Vera. Exploiting Locality in Maintaining Potential Causality. Proc. 10th Annual ACM Symposium on Principles of Distributed Computing, 1991:231-239

[Menon2003] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, B. Hillsberg. IBM Storage Tank-A heterogeneous scalable SAN file system. IBM Systems Journal, Vol. 42 NO. 2, 2003:250-267

[Miller1997] E. L. Miller, R. H. Katz. RAMA: An easy-to-use, high-performance parallel file system. Parallel Computing, Vol. 23 NO. 4, 1997:419-446

http://citeseer.ist.psu.edu/miller97rama.html

[Morris1986] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard. Andrew: A Distributed Personal Computing Environment. Communications of the ACM, Vol. 29 No. 3, 1986:184-201

[Morris2003] R. J. T. Morris, B. J. Truskowski. The Evolution of Storage Systems. IBM Systems Journal, Vol. 42 NO. 2, 2003:205-217

[Moyer1994] S. A. Moyer, V. S. Sunderam. PIOUS: a scalable parallel I/O system for distributed computing environments. In Proceedings of the Scalable High-Performance Computing Conference, 1994:71-78

http://citeseer.ist.psu.edu/moyer94pious.html

[Mullender1990] S. J. Mullender, G. Van Rossum, A. S. Tanenbaum, R. Van Renesse, H. Van Staveren. Amoeba: A distributed operating system for the 1990s. IEEE Computer 23(5), 1990:365-368

http://citeseer.ist.psu.edu/article/mullender90amoeba.html

[Muntz2001-1] D. Muntz. Separating Directory Structures from Physical File Systems. HPL-2001-174, 2001

[Muntz2001-2] D. Muntz. My Permanent Address: Finding A File After It Has Moved. HPL-2001-175, 2001

[Muntz2001-3] D. Muntz. Building a Single Distributed File System from Many NFS Servers. HPL-2001-176, 2001

[Murata1989] Tadao Murata. Petri Nets: Properties, Analysis and Applications. Proc. of the IEEE, Vol.77 No.4, 1989:541-580

[Opengroup.org] Opengroup.org. The Open Group Base Specifications Issue 6. IEEE Std 1003.1, 2004 Edition

http://www.unix.org/single_unix_ specification/

[Ousterhout1985]J. K. Ousterhout. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. Proceedings of the 10th ACM symposium on Operating systems principles (SOSP 1985), 1985:15-24

http://citeseer.ist.psu.edu/45549.html

[Ousterhout1988] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, B. B. Welch. The Sprite network operating system. IEEE Computer, 21(2), 1988:23-26

[Panasas2003] Panasas Corp. Inc.. Object Storage Architecture: Defining a new generation of Storage System Built on Distributed, Intelligent Storage Device. Panasas Technology White Paper, 2003

[Poleserve] Polyserve Inc.. PolyServe Cluster File System Architecture.

http://www.polyserve.com/requestinfo_formq1.php?pdf=2.

[Popek1986] G. J. Popek, B. J. Walker. The LOCUS Distributed System Architecture. Massachusetts Institute of Technology. 1986

[Ramakrishnan1992] K. K. Ramakrishnan，P. Biswas, R. Karedla. Analysis of File I/O Traces in Commercial Computing Environment. In proc. of the 1992 ACM SIGMETRICS joint international conference on Measurement and Modeling of Computer Systems, 1992:78-90

[RedHat] RedHat Inc.. Red Hat Globall File System. RedHat Whitepaper

http://www.redhat.com/whitepapers/rha/gfs/GFS_INS0032US.pdf

[ReiserFS] H. Reiser. Three reasons why ReiserFS is great for you.

http://www.namesys.com/

[Roselli2000] D. Roselli, J. R. Lorch, T. E. Anderson. A Comparison of File System Workloads. USENIX Conf. Proc., 2000:41-54

http://citeseer.ist.psu.edu/roselli00comparison.html

[Samaras1993] G. Samaras, K. Britton, A. Citron, C. Mohan. Two-phase Commit Optimizations and Tradeoffs in the Commercial Environment. In Proc. of the 9th IEEE International Conference on Data Engineering, 1993:520-529

[Sandberg1985] R. Sandberg, D. Goldeberg, S. Kleiman, D. Walsh, B. Lyon. Design and Implementation of the Sun Network Filesystem. In Summer Usenix Conference Proceedings,

1985:119-130

http://citeseer.ist.psu.edu/sandberg85design.html

[Satyanarayanan1990] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki. Coda: a highly available File System for a Distributed Workstation Environment, IEEE Trans. on Computers 39(4), 1990:447-459

http://citeseer.ist.psu.edu/satyanarayanan90coda.html

[Schmuck1991] F. Schmuck, J. Wylie. Experience with transactions in QuickSilver. In Proc. of the 13th ACM Symp. On Operating Systems Principles (SOSP 1991), 1991:239-253

[Schmuck2002] F. Schmuck, R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. Proceedings of the USENIX FAST' 2002 Conference on File and Storage Technologies, 2002:231-244

[Schwarz1994] R. Schwarz, F. Mattern. Detecting causal relationships in distributed computations: In search of the Holy Grail. Distributed Computing 7(3), 1994:149-174

http://citeseer.ist.psu.edu/article/schwarz94detecting.html

[Shepard2003] L. Shepard, E. Eppe. SGI® InfiniteStorage Shared Filesystem CXFS™: a High-Performance, Multi-OS Filesystem from SGI, White Paper. Silicon Graphics Inc., Mountain View, CA, 2003

http://www.sgi.com/

[Shepler1999] S. Shepler. NFS Version 4 Design Considerations. Network Working Group, RFC2624, June 1999

[Shi2001] 史小冬, 孟丹, 祝明发. COSMOS: 一种可扩展单一映像机群文件系统. 南京大学学报（自然科学） Vol.10, 2001

[Singhal1989] M. Singhal. Deadlock Detection in Distributed Systems. Computer, vol. 40 NO. 8, 1989:37-48

[SMB1987] Network Working Group. Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods. RFC 1001, 1987

http://www.faqs.org/rfcs/rfc1001.html

[SUN1989] SUN Microsystems Inc.. NFS: Network File System Protocol Specification. Network Working Group, RFC1094, 1989

http://www.faqs.org/rfcs/rfc1094.html

[Tanenbaum2006] A. S. Tanenbaum. Operating Systems Design and Implementation, 3rd Edition. Prentice Hall, 2006. ISBN: 0-13-142938-8

[Terekhov1999] I. Terekhov, T. Camp. Time Efficient Deadlock Resolution Algorithms.

Information Processing Letters, vol. 69, 1999:149-154

[Thekkath1997] C. A. Thekkath, T. Mann, E. K. Lee. Frangipani: a scalable distributed file system. In Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP '97), 1997:224-237

[Thompson1978] K. Thompson. UNIX Implementation. Bell System Technical Journal, Vol. 57, 1978:1931-1946

[Vahalia1996] U.Vahalia. UNIX Internals: The New Frontiers, 2nd Edition. Upper Saddle River, NJ: Prentice Hall, 1996

[Veritas] Veritas Inc.. Veritas Storage Foundation™ Cluster File System by Symantec - current access to shared data. http://eval.symantec.com/mktginfo/enterprise/fact_sheets/storage_foundation_cfs_datasheet.pdf

[Vogels1999] W. Vogels. File system usage in Windows NT 4.0. In Proc. Of the 17th ACM Symp. On Operating Systems Principles (SOSP '99), 1999:93-109

[Wang1998] Y. M. Wang, M. Merritt, A. B. Romanovsky. Guaranteed deadlock recovery: Deadlock resolution with rollback propagation. In Proc. Pacific Rim International Symposium on Fault-Tolerant Systems, 1995:92-97

[Weil2004] S. A. Weil, K. T. Pollack, S. A. Brandt, E. L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. Proc. of the ACM/IEEE SC2004 Conference, 2004:4-4

[Welch2004] B. Welch. What is a Cluster Filesystem?. 2004
http://www.beedub.com/clusterfs.html

[Williams2005] A. Williams, M. Arlitt, C. Williamson, K. Barker. Web Workload Characterization: Ten Years Later. 2005
http://pages.cpsc.ucalgary.ca/~carey/papers/2005/WebWorkload.pdf

[Xiong2005-1] 熊劲, 范志华, 马捷, 唐荣峰等. DCFS2 的元数据一致性策略. 计算机研究与发展，第 42 卷第 6 期，2005:1019-1027

[Xiong2005-2] 熊劲. 大规模机群文件系统的关键技术研究. 中国科学院研究生院博士学位论文，2005

[Yan2004] Jie Yan, Yao-Long Zhu, Hui Xiong, Renuga Kanagavelu. A Design of Metadata Server Cluster in Large Distributed Object-based Storage. 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies, 2004:100-106

[Yang2005] 杨德志, 黄华, 张建刚, 许鲁. 大容量、高性能、高扩展能力的蓝鲸分布式文件系统. 计算机研究与发展, 第 42 卷第 6 期, 2005:1028-1033

[Zhang1999] Y. Zhang, L. Rauchwerger, J. Torrellas. Hardware for speculative parallelization of partially-parallel loops in DSM multiprocessors. In Proc. of the 5th International Symp. on High Performance Computer Architecture, 1999:135-139

[Zhang2001] Zh. Zhang, C. Karamanolis, M. Mahalingam, D. Muntz. Cross-Partition Protocols in a Distributed File Service. HPL-2001-129, 2001

[Zhu2003] N. Zhu, T. Chiueh. Design, Implementation and Evaluation of the Repairable File Service. In Proc. of the International Conference on Dependable Systems and Networks. 2003:217-226

致　谢

感谢我的导师许鲁研究员。许老师严谨、求实的学术风格，是我今生的学习榜样。多年后，我依然还会想起老师的教诲：“每个人不见得比别人聪明!能否取得一定的成绩，全取决于认真与否!”。谢谢老师的教诲!

感谢张建刚博士，谢谢您这些年的指导。

感谢黄华、田颖、范勇、王敏、刘振军、尹扬等同学。

感谢蓝鲸文件系统组的所有同事。特别感谢秦平在本论文过程中提供的无私帮助，感谢宋震、贾瑞勇等同事对本论文提出的宝贵意见，感谢支持组的相关同事在本文原型系统评测过程中提供的帮助。感谢工程中心的同事们，谢谢你们提供的帮助。

感谢研究生部的老师们提供的支持。

感谢评委老师们对本论文的评阅工作!

感谢我的父母、我的亲朋好友。父母含辛茹苦这么多年，为了我们辛勤操劳，实属不易！在论文期间，父亲身患重病，希望能够顺利康复!

感谢我的岳父母这么多年来在物质和精神上的支持和鼓励。

感谢我的妻子邹锐。相识已逾十载，一起度过美好的青春年华。博士多年，是你在一直默默支持，独自承受着家庭生活的重担。谢谢！感谢我们的两个小宝贝邹喻宁、杨喻安。你们给了我们压力，也给了我们动力。

感谢所有给予我关心和帮助的人们!

作者简历

姓名：杨德志　　性别：男　　出生日期：1977.10　　籍贯：四川省岳池县

【教育经历】

2002.9 – 现在　　中国科学院计算技术研究所，
计算机系统结构，
博士研究生。

1999.9 – 2002.7　　华中理工大学计算机学院，
计算机应用技术专业，
攻读硕士学位，获工学硕士学位。

1995.9 -- 1999.7　　华中理工大学管理学院，
税务管理专业，
攻读学士学位，获经济学学士学位。

【攻读博士学位期间发表的论文】

[1]杨德志，黄华，张建刚，许鲁，大容量、高性能、高扩展能力的蓝鲸分布式文件系统，计算机研究与发展，第 42 卷，第 6 期，2005 年。

[2]杨德志，许鲁，张建刚，蓝鲸分布式文件系统元数据服务研究，计算机工程，已录用。

[3]杨德志，许鲁，张建刚，BWMMS 元数据分布信息缓存管理，计算机科学，已录用。

【攻读博士学位期间参加的科研项目】

[1] 国家“八六三”高技术研究发展计划基金项目（2002AA112010）

[2] 中国科学院计算技术研究所创新项目“蓝鲸网络存储产品化”

[3] 中国科学院百人计划项目“以网络存储为核心的服务系统”