中国科学院计算技术研究所

博士学位论文

机群文件系统服务器关键技术研究

姓名：吴思宁

申请学位级别：博士

专业：计算机系统结构

指导教师：徐志伟

20040101

摘要

机群文件系统作为缓解机群系统 I/O 瓶颈问题的手段，需要为机群系统的各类应用提供高性能、可扩展的文件服务，因此对机群文件系统的研究是高性能计算机体系结构研究的重要内容。本文结合曙光机群文件系统 DCFS 的设计和实现，对机群文件系统设计的关键问题进行了讨论，并针对机群文件系统服务器设计的几个问题进行了研究。本文的主要工作如下：

1. 本文对机群文件系统的体系结构进行了总结，提出了多文件系统卷的结构，该结构具有可扩展、易管理、灵活的特点；本文对多文件系统卷中存储服务器的网络存储分组的组织形式进行分析，提出了网络存储分组模型，并讨论了影响存储分组读写性能的因素；对元数据服务器的组织和元数据的分布与映射策略进行了讨论，给出了可调粒度的元数据分布策略，使得用户可以根据应用程序的模式灵活选择文件系统卷的元数据分布粒度。
2. 作者对目录操作中的两个问题进行了研究：（1）元数据目录缓存管理；（2）大目录优化。独立的元数据服务器使设计者可以根据目录缓存的特点设计合理的管理方法，作者通过研究发现，客户端目录缓存和元数据服务器上的 LOOKUP 目录缓存和 READDIR 缓存构成了一个多级的目录缓存结构，元数据服务器上的 LOOKUP 缓存和 READDIR 缓存表现出了不同的访问特性，作者根据 LOOKUP 缓存和 READDIR 目录缓存的特性提出了目录缓存的管理方法，试验表明该方法较采用 LRU、LFU 和 FBR 替换算法的缓存管理方法具有更高的缓存命中率。作者和本研究小组成员合作对大目录优化进行了研究，提出了 LMEH 动态 HASH 的目录管理算法，在 DCFS 上的试验表明，对于大目录下的元数据吞吐率性能，该方法较线性的目录管理算法平均提高了 1.97 倍。
3. 作者结合 DCFS 元数据分布策略和元数据缓存管理设计了元数据一致性协议，该协议保证了元数据一致性，分析表明其开销是可以接收的。
4. 在曙光 4000L 上设计并实现曙光机群文件系统 DCFS，给出了机群文件系统性能评价的方法，定义了读写带宽性能和元数据吞吐率的可扩展性度量。在曙光 4000L 上的测试表明，DCFS 与类似结构的 PVFS 文件系统相比，在读写性能上，DCFS 除了在小文件最高读带宽性能上比 PVFS 差 19%，在其余情况下 DCFS 的最高聚合读写性能优于 PVFS，平均高 44.4%；DCFS 元数据吞吐率的性能平均比 PVFS 高 6.391 倍；DCFS 在综合负载测试中表现出比 PVFS 更好的性能，全局响应时间为 PVFS 的 18.2%。

关键词 机群系统 机群文件系统 机群文件系统结构 目录操作 元数据一致性 性能评价

ABSTRACT

As a key component of cluster system, cluster file systems must provide high performance and scalable file service to solve the I/O bottleneck of cluster system. Research in cluster file system is important to high performance computing. With the design and implementation of DCFS (Dawning Cluster File System), this dissertation deeply discusses the key technologies of a cluster file system and studies problems in designing the servers of a cluster file system. The contributions of this dissertation include:

1. This dissertation firstly compares architectures of cluster file systems and presents a new architecture called multi file system volumes which is scalable, manageable and flexible. This dissertation analyses the structure of storage servers, presents a model of a network disk stripe group and discusses the factors that impacts the read/write bandwidth of a network disk stripe group. A metadata placement policy is proposed for user to select suitable granularity.
2. This dissertation studies two key issues of directory operations: directory cache and large directory optimization. The research showed that LOOKUP and READDIR directory cache on DCFS metadata servers have different access characteristics with that on client nodes. A new directory cache management method is presented according to these access characteristics. Experiments showed that this method has higher directory cache hit ratio than those which exploit LRU, LFU and FBR cache replacement algorithms. This dissertation introduces a large directory optimization algorithm LMEH (Limited Multi-level Extendible Hash) which is presented by author and a member of our research group. The experiments indicated that the throughput of LMEH for large directory is 1.97 times as Linear algorithm on average.
3. This dissertation designs a metadata consistency protocol according to the metadata placement policy and cache management in DCFS. This protocol ensures the consistency of metadata in a cluster file system and has acceptable cost.
4. Design and implement DCFS on the Dawning 4000L Cluster. Present methods to evaluate a cluster file system and define scalability of read/write and metadata operations. Compared with PVFS, DCFS exhibits higher aggregate bandwidth except read bandwidth for small files. The read bandwidth of DCFS for small files is 19% less than PVFS and the bandwidth of DCFS is 44.4% higher than PVFS in other experiments on average. The metadata throughput of DCFS is 7.391 times as PVFS on average. And the overall response time of DCFS is 18.2% as PVFS in the synthesis workload experiment.

Keywords: Cluster System, Cluster File System, Architecture, Directory Operation, Metadata Consistency, Performance Evaluation

声　　明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。就我所知，除了文中特别加以标注和致谢的地方外，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：吴忠宁　　日期：2004年4月8日

关于论文使用授权的说明

中国科学院计算技术研究所有权处理、保留送交论文的复印件，允许论文被查阅和借阅；并可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存该论文。

作者签名：吴忠宁　　导师签名：[illegible]　　日期：2004年4月8日

第一章 引 言

可扩展、高性能机群系统已成为高性能计算机的主流体系结构，2003 年公布的第 22 届全球排名前 10 名的高性能计算机中，机群系统占了 7 台[Top03]，基于机群体系结构的并行处理系统在高性能计算和事务处理中的应用越来越广泛。I/O 系统成为制约计算机系统包括机群系统性能提高的瓶颈，近年来，研究人员通过优化存储设备、开发新型存储体系结构和各种结构的文件系统，试图解决或缓解 I/O 瓶颈问题。机群文件系统作为缓解机群系统 I/O 瓶颈问题的手段，许多研究机构对其进行了研究，开发了各种机群文件系统，而机群文件系统的现状不能令人满意，在 2003 年的 IEEE International Conference on Cluster Computer 上，与会者关注的一个挑战问题就是：为什么还没有一个好的机群文件系统？在本章中，作者首先简单地介绍了机群及机群文件系统，分析了应用对机群文件系统的需求，简要地论述了影响机群文件系统设计的技术的发展。另外，作者介绍了本论文的课题背景和意义，同时对论文的工作进行了总结。

1.1 机群和机群文件系统

机群系统是由一组计算机系统（节点）通过高性能网络或局域网互连而形成的具有单一系统映像(Single System Image，SSI)的高可用、高性能、高可扩展性的计算机集群系统。它的每个节点都是一个完整的计算机系统，如 SMP 服务器、工作站或 PC 服务器，可以独立工作。图 1.1 示出了机群系统的体系结构。机群的三个系统结构特点如下[Hwa98]：

- ✧ **机群节点**：每个节点是一台完整计算机。这就意味着每个节点有自己的处理器、高速缓存、磁盘以及某些 I/O 适配器。此外在每个节点上驻留有完整、标准的操作系统。一个节点可以拥有多个处理器，但是只有一份操作系统映象拷贝。
- ✧ **结点间互联**：机群中的节点，通常用商品化网络，如以太网、FDDI、光通道以及 ATM 开关进行连接。
- ✧ **单一系统映象**：一个机群是一个单一计算机资源。与此相反的是分布式系统(例如一个计算机局域网)，在那里将节点作为单独资源。机群借助一些单一系统映象技术，实现了单资源概念。

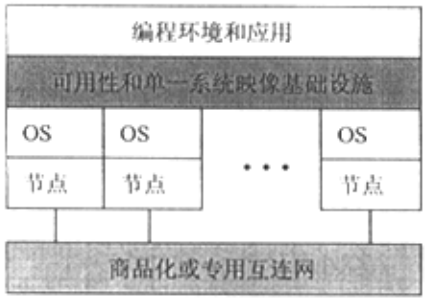


图 1.1 机群系统的典型体系结构

我们将机群系统的特征概括：

- ✧ **好用性** 由于机群系统中每个节点都是传统平台，用户能在熟悉的成熟环境中开发和运行他们的应用程序。平台提供了功能很强的工作站编程环境工具，能允许大部分现有的串行应用程序无需修改便可运行。
- ✧ **可用性** 传统的单一系统，如主机和容错系统依靠昂贵的定制设计来获取高可用性，而机群不使用定制组件，而是用廉价的商品化部件以提供含有大量冗余的较高可用性。
- ✧ **可扩展性** 一个机群的计算能力随节点增多而增加。其次，机群的可扩展性是群体可扩展性。因为是松散耦合，机群能扩展至几百个节点。
- ✧ **性能价格比** 机群能成本有效地获取上述优点。它采用大量商品化部件，其性能和价格遵循摩尔定律，从而使机群的性能价格比增长速率快于其他超级计算机系统。

机群系统的以上特点使其应用范围逐步扩大，人们不仅可以利用机群系统的高性能来进行传统上只有 MPP 才能处理的科学和工程计算问题，而且可以利用它们来进行各种事务处理。这些应用要求机群系统提供足够的 I/O 处理能力，缓解系统的 I/O 瓶颈问题。机群文件系统是是目前机群系统解决 I/O 问题的一种重要方式，是实现整个机群信息共享的重要手段。

由于机群系统应用于各种不同的环境中，不同的应用程序具有不同的特性，对机群文件系统提出了不同的要求。另外，从机群系统的结构中可以看到，机群文件系统是构建在互连网络、存储设备和节点之上的，互连网络、存储设备和节点技术的提高也推动了机群文件系统的发展。在下面，作者首先描述了机群系统一些典型应用对机群文件系统的需求，然后对影响机群文件系统的技术进行了简要的介绍。

1.1.1 数据密集型应用对机群及机群文件系统需求

传统的数据密集型科学计算应用需要更高的 I/O 处理能力。美国 DOE National Nuclear Security Administration and the DOD National Security Agency 在[DOE01]中指出，对于一台性能为 n teraflops 的高性能计算机，需要的内存容量为 $n^{3/4}$～2/3n TB，IO 聚合带宽为 $n^{3/4}$/600～n/500 GB/s，表 1.1 给出了该实验室中并行程序对高性能计算机性能的需求。从中国科学院计算技术研究所承担的雷达地形数据处理项目[Han01]中，我们可以看到典型图像数据应用对存储系统容量与带宽的需求。在此项目中，平均每天要产生 0.5 TB 的新数据，持续的传输带宽需求为 34MB/s。在这种典型的数据密集科学计算应用中，高性能存储 I/O 的性能已经成为决定应用是否可以正常运行的关键因素。

表 1.1 并行程序对高性能计算机的需求[DOE01]

Aggregate Bandwidth Rates for One Parallel Job Simulation & Physics Model Aggregate FS Requirements				
	1999	2002	2005	2008
Teraflops	3.9	30	100	400
Memory Size (TB)	2.6	13-20	32-67	44-167
I/O Rates (GB/s)	4-8	20-60	50-200	80-500

[Jia01]中指出，作为网络的驱动因素，信息数据正在成为网络的核心，数据的安全、高效存储和管理作为网络发展的基础，日益受到人们的重视。未来几年内，存储技术将成为令人瞩目的一个市场。基于 Internet 的应用，如电子商务、电子邮件和客户关系管理（CRM）等将成为存储服务的主要市场，这些应用都需要对海量数据进行快速访问。它们不但需要存储系统容量能持续扩大，而且对数据的有效管理提出了更高的要求。

不同的应用表现出不同的负载模式。对于科学计算类程序，[DOE01]中给出了该实验室三种典型并行计算程序的负载特性，从中可以看到，读写操作占了很大比例，并且很多程序是基于消息传递的 MPI 程序。而一些应用则表现为以元数据操作为主。研究人员在为网络文件系统 NFS 构造一个新的 SPEC 基准（SFS 2.0）前，对 750 台运行 NFS v2（版本 2）的 Auspex 服务器进行专门的负载研究，希望能从收集的负载数据中获得一些文件负载共性。这些研究人员发现，根据收集的数据，负载基本上可以分为两大类，如表 1.2 所示[Rob99]。在表 1.2 的两组负载模式中，元数据相关的存取操作比例分别达到了 84%与 49%。可见在某些应用模式中，元数据操作的数量要远远超过普通文件数据读写操作的数量。这种频繁发生的元数据服务请求是否能及时得到响应，在很多应用场合决定了系统的整体性能。

表 1.2 数据存取操作比例[Rob99]

文件数据操作	读	11%	32%
	写	5	17
元数据操作	只读	80	47
	修改	4	2
其他		0	2

1.1.2 技术发展趋势对机群文件系统的推动

存储设备、网络、网络存储构成了机群文件系统的硬件基础，它们在性能上的提高以及技术的发展不仅使机群文件系统能够获得更高性能，而且对机群文件系统的设计产生影响。在本节中，作者简要地介绍这些技术。

1.1.2.1 存储设备

磁盘是构成外存储系统的主要设备，各大厂商正在努力提高磁盘设备自身的性能来改善外存储 I/O 性能与可靠性。磁盘数据访问时间包括寻道时间、旋转延迟、介质存取时间和总线传输时间等几个部分，一般对磁盘性能的优化从以下几方面入手[Gro96]：减少寻道时间、提高磁盘驱动器马达主轴的转速以及优化磁盘控制器内部的调度算法（包括缓存和请求调度等）。

1988 年加州大学 Berkeley 分校的 David A. Patterson 等人首次提出了廉价磁盘冗余阵列（Redundant Arrays of Inexpensive Disks，RAID[Pat88][Pat89]）等级划分的概念。他们根据容错级别与数据分布方式的不同，提出了五种不同的 RAID，即 RAID1——RAID5 共 5 级，并把传统的无冗余结构的磁盘阵列结构定义为 RAID0 级。RAID 由于采取数据分块技术，即在多块磁盘上交叉存放数据，使得多个磁盘可以并行工作，从而改善 I/O 响应时间。另外，采用冗余技术，极大地提高了磁盘阵列的可靠性和可用性。在过去一些年里，又产生

了一些新的磁盘分级，如 RAID10、RAID6 与 EVENODD 等。

另外，随着半导体技术的发展与成本的下降，人们试图给磁盘设备增添更多的智能，如卡内基.梅隆大学的 Active Disk[Rie99]和 ANSI 的 T10 委员会提出的 OBD[T10-00]都尝试将更多的控制权赋予磁盘控制器，让它来更加合理地分布数据，另外还期望这种技术获得更好的可扩展能力。OBD 等存储设备的出现赋予了存储设备上的数据更多的属性，而且存储设备上具备更强计算能力，这些使得文件系统和存储设备的功能会重新划分，Lustre[Bra02]即为基于 OBD 的新型文件系统。

1.1.2.3 网络

许多研究机构和公司对机群系统的高带宽、低延迟的互连网络及网络协议进行了研究，这些网络技术为机群文件系统获得良好的性能和可扩展性提供了保证，同时，机群文件系统可以充分利用这些网络技术提供的功能设计合理的结构，如 DAFS 基于 VIA 设计了更为有效的文件协议。在本小节中，作者简要地介绍几种网络技术。

Ethernet、Fast Ethernet 与 Gigabit Ethernet

以太网是应用最广泛的局域网技术，10M 以太网、100M 快速以太网和千兆以太网目前大量应用于机群系统。千兆以太网与 10M 以太网和 100M 快速以太网标准的兼容，利用了原以太网标准规定地全部技术规范，包括 CSMA/CD 协议、以太网帧格式、全双工、流量控制以及 IEEE 802.3 标准中定义的管理对象，提供大约 1000Mbps 的带宽。

Myrinet

Myrinet[Bod03]网络的设计目标是要在局域网环境中获得系统域网络的性能， Myrinet 网络采用 MPP 系统中的数据包通信和交换技术。基于 Myrinet 网络开发了许多高速通讯协议，如 GM[Myr03]、BIP[Geo99]。由中国科学院计算技术研究所智能中心开发的 BCL[Ma01]是一种基于 Myrinet 的半用户级通信协议，它基于基于消息传递机制，通过结点间相互收发消息来完成机群间的通信、同步。

VIA

VIA（Virtual Interface Architecture）[VIA97]标准是由 Intel、Microsoft 等推出的用户级通信界面标准。VIA 旁路了 OS 和协议栈，可以通过很少的指令来完成传统的网络中需要多几个数量级的指令才能完成的通信任务。VIA 引进了基于标准的 API 来给机群系统提供低延时、高带宽的消息传递。低延时和持续高带宽是通过在发送和接受消息时减少数据拷贝和旁路 OS 来实现的。Giganet 公司（已被存储设备厂商 Emulex 收购）的 cLan（GigaNet Cluster Area Network）是基于 VIA 的机群互连产品系列。

Infiniband

Infiniband[Ifi01]是一种全新的、功能强大、设计用来支持 Internet 基础设施 I/O 互连的体系结构，被工业界的顶级公司所支持，执行委员会成员包括：Dell、Hewlett Packard、IBM、Intel、Microsoft 和 Sun，Infiniband 行业协会成员总计超过 220 个。Infiniband 是唯一既提供机箱内底板互连解决方案，又可以实现机箱间的高带宽互连，把 I/O 和系统域网络统一起来的标准。Infiniband 网络设计用来支持集群应用、存储域网络、和处理器间通信，在获

得高带宽的同时也得到 QoS 和 RAS 性能。Infiniband 网络适合应用在以资源共享为目标的高带宽、海量数据传输为主要通信特征的应用场合。

1.1.2.3 网络存储

根据共享的级别以及所依赖的底层网络类型，网络存储系统可以划分为如图 1.2 所示的类型[EMC03]：（1）图的左上角是 NAS（Network Attached Storage，网络连接存储或附网存储）系统，一般指独立连接到网络上并拥有独立的网络地址的存储设备[Hit00]；（2）SAN（Storage Area Network，存储区域网）系统[Far00]由通过高速专用网络（或者子网）连接的多种不同的数据存储设备组成；（3）IP 通信技术和块存储设备的结合形成 IP_BD（IP Block Device）系统，对于使用它的计算机主机而言是物理块设备，而实际上是一个可以解释并执行主机发来的块设备命令请求的计算机。目前这类系统有 iSCSI、FCIP、iFCP 与 NBD[Bre00]；（4）NAS+SAN 系统，有人称之为 SNAS，也有人称之为 MPFS（Multiple Path File Serving），其目的是为了结合 NAS 的文件级别的共享与 SAN 系统的高性能；（5）基于对象的访问模式与 IP 技术的结合形成内容寻址存储设备（CAS，Content Addressed Storage）。

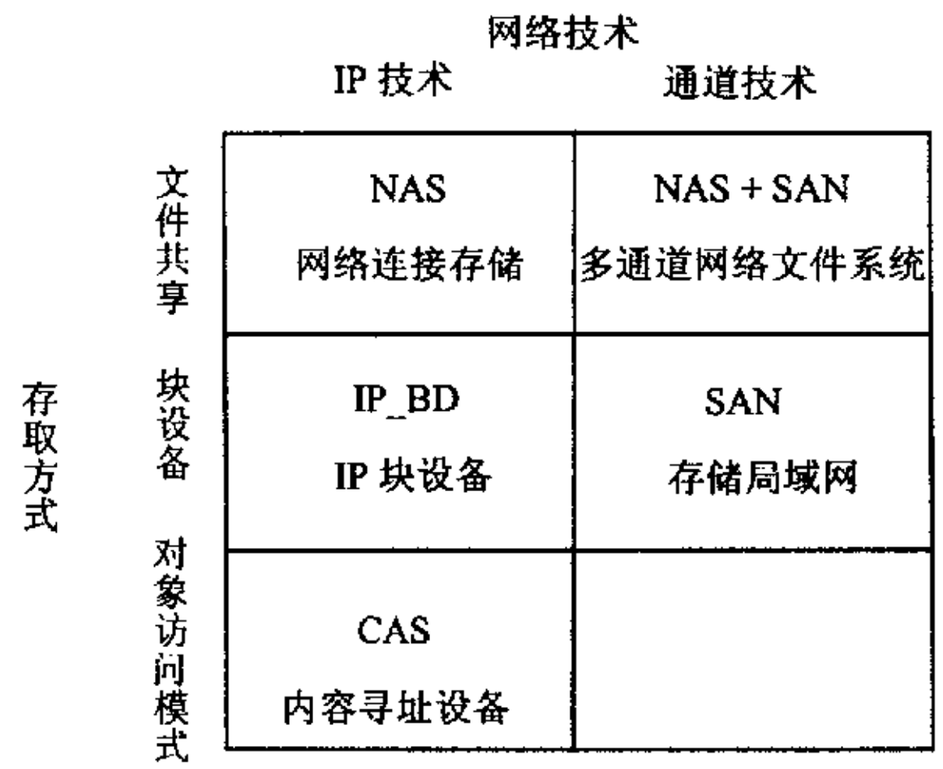


图 1.2 网络存储分类 [EMC03]

1.1.3 小结

在上面的论述中，作者首先给出了机群系统的概念及其特点，指出机群文件系统可以缓解其 I/O 瓶颈问题，从应用需求的角度分析了应用对高性能、可扩展、高可用机群文件系统的要求，并简单介绍了与机群文件系统紧密相关的网络、存储系统等技术的发展，可以看到：

- ✧ 高性能机群系统成为当前高性能计算机的主流体系结构，I/O 瓶颈问题成为制约机群系统性能的关键问题，机群文件系统旨在为机群提供全局共享、高性能、可扩展、高可用和易管理的手段。
- ✧ 各种应用需求给机群文件系统提出了新的挑战，除了传统的并行计算的应用，机群系统被广泛应用于 Web、流媒体服务，而这些应用具有同并行计算程序不同的负载模式，而这要求机群文件系统能够根据应用的特点进行调整。
- ✧ 各种技术的不断发展，特别是网络和存储设备的发展不仅为机群文件系统提供了

性能更高、更可靠的硬件，而且也促进了机群文件系统的发展。例如，VIA[VIA97]等用户级通讯协议使 DAFS[DAFS01]文件系统结构更加简洁，通过减少 I/O 路径上的开销来提高文件系统的性能；OSD[T10-00]等智能存储技术的发展使机群文件系统有可能利用存储系统的智能特性减轻服务器的负载，有代表性的系统如 LUSTRE[Bra02]；网络和存储技术的结合促进了网络存储的发展，对应与各种网络存储结构出现了各种结构的文件系统，如基于 SAN 的文件系统 GFS[Pre99]等。

1.2 课题背景和意义

本项研究工作是结合国家智能计算机研究开发中心承担的“国家 863”高性能计算机专项课题：面向网格的高性能计算机-曙光 4000 超级服务器进行的，目的是为高性能计算机中的各种应用提供高性能、可扩展、高可用的机群文件系统服务。该文件系统称为 DCFS v1.0 (Dawning Cluster File System Version 1.0)，在本文中简称为 DCFS。

曙光 4000 是国家 863 计划高性能计算机及其核心软件重大专项支持的研究项目，通过研究面向网格的高性能计算机，为网格提供计算力服务。曙光 4000 采用网格技术，体系结构以构件性(Component)、标准性(Standard)、协作性(Coordinate)为基准，采用服务化(Service)、安全化(Security)、专业化(Specialization)、智能化(Intelligence)的 3SI 技术路线。项目研究的目标包括：曙光 4000L Linux 超级服务器、曙光 4000A 高性能超级服务器、10Tflops 曙光 4000 面向网格的高性能计算机、曙光 4000H 生物信息处理高密度专用机群系统、曙光 4000T 具有自主知识产权的服务器。其中曙光 4000L 超级服务器已于 2003 年 3 月通过项目验收[Sun03]，曙光 4000L 以支持数据密集应用为主，针对应用特点对系统进行优化设计，解决关键技术问题，实现应用与系统的一体化优化。同时，能支持科学与工程计算，网络与信息服务，事务处理等多种应用。高性能、可扩展、高可用的机群文件系统可以为这些应用提供更有效的支持，因此，机群文件系统关键技术的研究对机群系统有重要的意义。

1.2 本文的贡献与内容组织

机群文件系统作为一个复杂的分布式系统包括许多值得研究的内容，本文针对机群文件系统服务器设计中的几个问题进行研究：

（1）可扩展机群文件系统结构。作者对机群文件系统结构进行了分析，针对曙光 4000L 的具体情况提出了多文件系统卷的结构，该结构具有可扩展、易管理、灵活的特点；对多文件系统卷中存储服务器的网络存储分组的组织形式进行了分析，提出了网络存储分组模型，讨论了影响存储分组读写性能的因素；对元数据服务器的组织和元数据的分布与映射策略进行了讨论，给出了可调粒度的元数据分布策略，使得用户可以根据应用程序的模式灵活选择文件系统卷的元数据分布粒度；

（2）元数据服务器目录操作。本文针对目录操作中的两个问题：元数据服务器目录缓存和大目录优化进行了讨论。作者根据 LOOKUP 缓存和 READDIR 目录缓存的特性提出了目录缓存的管理方法。试验表明该方法较采用 LRU、LFU 和 FBR 替换算法的缓存管理方法具有更高的缓存命中率。作者和本研究小组成员合作对大目录优化进行了研究，提出

了 LMEH 动态 HASH 的目录管理算法，在 DCFS 上的试验表明，该方法较线性的目录管理算法平均提高了 1.97 倍；

（3）作者结合 DCFS 元数据分布策略和元数据缓存管理设计了元数据一致性协议，该协议保证了元数据一致性，分析表明其开销是可以接收的；

（4）机群文件系统实现与评价。本研究小组在曙光 4000L 上设计并实现了 DCFS，并讨论了机群文件系统性能评价的方法。本文针对文件系统读写带宽和元数据吞吐率给出了可扩展性度量的参数，并给出了评价机群文件系统读写性能、元数据吞吐率、综合负载测试的方法。在曙光 4000L 上的测试表明，DCFS 与类似结构的 PVFS 文件系统相比，在读写性能上，DCFS 除了在小文件最高读带宽性能上比 PVFS 差 19%，在其余情况下 DCFS 的最高聚合读写性能优于 PVFS，平均高 44.4%；DCFS 元数据吞吐率的性能平均比 PVFS 高 6.391 倍；DCFS 在综合负载测试中表现出比 PVFS 更好的性能，全局响应时间为 PVFS 的 18.2%。

本文后续各章的组织如下：第二章分析了几种典型的机群/分布式文件系统，总结了机群文件系统设计的目标和关键技术；第三章分析了机群文件系统的结构，给出了 DCFS 采用的多文件系统卷的体系结构，并对存储服务器和元数据服务器的组织进行了分析和论述；第四章针对机群文件系统中的目录操作进行了研究，分析了多级的目录缓存结构，在试验的基础上给出了 DCFS 元数据服务器目录缓存管理的方法，另外，第四章还研究了大目录优化的算法，提出了 LMEH 算法；第五章描述了元数据一致性协议；在第六章中，作者给出 DCFS 文件系统在曙光 4000L 上的实现以及测试结果。

第二章 机群文件系统关键技术

机群文件系统提供了有效的管理和使用机群系统中存储子系统的手段，许多研究机构和公司提出和开发了各种提高机群文件系统性能、可扩展性、可用性和可管理性的技术。在本章中，作者从一些典型的机群/分布式文件系统出发，简要介绍了这些系统的背景、特点，然后给出了机群文件系统设计的目标，总结了机群文件系统中与 DCFS 相关的关键技术。

2.1 典型系统研究

2.1.1 NFS 和 DAFS

NFS[NFS89][NFS95][NFS03] 的最早版本于 1984 年由 SUN 微系统公司发布，它目前已被移植到几乎所有 UNIX 系统上，并成为事实上的工业标准。NFS 的设计目标是提供不同平台间透明文件访问的方法，因此，NFS 协议在设计时从可用性、互操作性和安全性等方面进行考虑。NFS 体系结构基于客户机/服务器模型，客户机和服务器之间通过 SUNRPC 进行通讯，使用 XDR 进行数据编码。早期的 NFS 协议采用无状态（Stateless）的服务流程和基于时间的客户端缓存一致性协议，简化了协议和实现。NFS 服务器在安装 NFS 文件系统和处理每个请求时进行权限检查。NFS 的缺陷在于它的单服务器结构，由于单服务器的体系结构，因此 NFS 的可扩展非常差。本来无状态的特性可以让 NFS 具有较好的容错能力，但因为它采用了单一集中式服务器的结构，当服务器出现故障时整个系统便不能提供服务，专用的 NFS 服务器如 IBM HA-NFS 采用了磁盘镜像和可选的网络备份来解决这个问题。

目前最新的 NFS 版本是 NFS v4。在 NFS v4 中加入了许多新特性，在安全性、性能方面对前面版本进行了很多改进，提高了如 Internet 上带宽低， 延迟高的环境下的性能，提供了较好的安全性、较好的跨平台互操作性以及易于协议扩展。

网络存储企业组织 SNIA 制定了基于 NFS v4 的 DAFS[DAFS01]网络文件系统协议，期望它能成为将来 NAS 系统的标准。DAFS 基于 VIA 等提供 RDMA 功能的网络协议，其基本原理是通过缩短服务器读写文件时的数据路径，来减少和重新分配 CPU 的计算任务。它提供内存到内存（memory-to-memory）的直接传输途径，使数据块的复制工作不需要经过应用服务器和文件服务器的 CPU，而是在两个物理设备的预先映射的缓冲区中直接传输。同时，由于操作系统对文件操作的介入更少了，节省下来的处理能力就被释放出来，用于其它方面的任务。

2.1.2 AFS、CODA 和 DFS

AFS[How88]是 1984 年美国卡内基-梅隆大学（CMU）研制开发的分布式文件系统。它的设计目标是支持大学校园网内数千台工作站间的文件共享。AFS 提供一致的、不依赖于

存储位置的统一的名字空间和基于会话期间的文件共享语义。AFS 的设计特点包括：良好的可扩展体系结构，以多个服务器和卷的数据复制来支持大量的用户，同时具有较好的可用性；以回调（Callback）机制维护多个缓存副本间的一致性。目前，一些开放源码组织在维护 AFS 的开发[Ope04]。

1990 年代，CMU 又在 AFS 的基础上开发了支持弱连接环境（如移动办公）的分布式文件系统 Coda[Mum94-1][Mum94-2][Mum94-3][Mum96]。Coda 的设计目标主要是在弱连接环境下提供最大的可靠性，它在许多方面都直接继承了 AFS，例如客户端缓存策略。在高可靠性方面，Coda 采用了服务器复制（Server Replication）和断连接操作（Disconnected Operation），前者由客户将对于文件的修改向多个服务器传播；后者在客户端缓存了整个的文件，在连接断开的情况下，对于这些文件的操作仍然可以进行，一旦连接恢复，就恢复到正常的情况下操作。

在 1989 年，Transac 公司接管了 AFS 的开发和产品化工作，经过他们努力，OSF 接纳了 AFS 技术，将其作为 OSF 分布式计算环境（DCE）的分布式文件系统的基础。这个新的文件系统通常叫做 DCE DFS[Vah99]，或简称 DFS。DFS 是从 AFS 中演化而来，因此在很多方面同 AFS 相似。DCE DFS 的最大特点就是对于访问共享文件提供了细粒度的严格 UNIX 文件共享语义。它通过复杂的令牌机制来做到这一点。DFS 支持四种类型的令牌：数据令牌，状态令牌，锁令牌和打开令牌，每一种令牌处理不同的文件操作。对于令牌的管理在服务器端，并由服务器启动令牌的授予和无效等动作。DFS 的体系结构很复杂，不容易实现，而且高速缓存的一致性维护机制和死锁避免机制也是高度复杂的，这使得 DFS 在目前没有获得广泛的应用。

2.1.3 xFS

早期的网络文件系统（如 NFS、Sprite）依赖于一台集中式服务器，而无集中式服务器（Serverless）的系统中所有工作站在提供文件服务方面地位是均等的。系统中的任何一台机器都能够存取、缓存及控制任意的数据块。该方法利用位置独立性并结合快速局域网，较之传统的网络文件系统，能提供更好的性能和可扩展性。另外，由于任一台工作站可以承担出现故障的部件的责任，这种无集中式服务器设计方案通过采用冗余数据存储可提供高可用性。

为了验证这种方法，加州大学伯克利分校实现了一种没有集中式服务器的文件系统原形 xFS[And95][Wan97][Wan99]，它主要完成了下列功能：并行协调式的元数据管理；协作式缓存；并行 I/O 支持；日志式存储管理。实验结果表明它的可扩展性和性能均优于 NFS 和 AFS，但 xFS 协议设计过于复杂，不易实现。

2.1.4 PVFS

PVFS[Car00][Lig99]是一种客户/服务器结构的并行文件系统，由一个元数据服务器、若干储存服务器和文件系统客户结点组成，应用通过客户节点访问 PVFS 文件系统中的数据，节点上所有的创建、删除、读写文件的请求都必须发送到服务器方进行处理。文件元数据与数据分别由不同的服务器管理。其主要特点如下：

✧ 单一元数据服务器，文件系统的元数据以本地文件系统文件的形式存在，在服务

器上按照 PVFS 名字空间的目录树结构相应地组织成树形结构。对于一些注重聚合读写带宽的并行应用程序，PVFS 单一元数据服务器的结构是合理的，但对于元数据操作性能要求较高的应用，单一的元数据服务器容易造成瓶颈。

- ✧ 文件的数据和元数据都是存储在本地文件系统之上，并且没有实现服务器端缓存，因此文件操作需要直接读写文件系统。
- ✧ PVFS 提供了多种接口。除了提供系统调用的接口外，PVFS 提供了用户库，应用程序可以使用用户库提供的接口访问文件系统，该库提供了一些灵活的策略来控制对文件的读写，例如可控制的数据分条。这个特性体现了 PVFS 并行文件系统的特点。

2.1.5 GPFS

GPFS 文件系统由 20 世纪 90 年代的 Calypso 文件系统[Moh94]以及 Tiger-Shark 文件系统[Has98]发展而来。

IBM 公司的 GPFS[Bar98][Sch02]提供给 GPFS 机群用户的是文件级共享接口。在 GPFS 服务器上通过 VSD[Att94]（Virtual Shared Disk，虚拟共享磁盘）来存储实际文件，文件数据在多个磁盘上进行分条(Stripe)。GPFS 对于访问共享文件提供了细粒度的严格 UNIX 文件共享语义，GPFS 采取了一种复杂的分布式令牌管理方法来实现这一点。GPFS 采用了动态可扩展的 Hash 技术（Extendible Hashing）对大目录的支持。采用元数据的日志技术提高文件系统的一致性。GPFS 记录所有影响文件系统一致性的元数据操作。每个结点分别记录在不同的日志文件中，并且这些日志文件都存放在 GPFS 文件系统中，这样，当一个结点失效后，其它结点就可以帮助该结点完成恢复操作。而对于元数据采用集中式管理机制，任何一个文件，动态的选择一个结点作为该文件的元数据服务器，任何对其的修改必须由该结点完成。

2.1.6 GFS

GFS[Sol97][Pre99][Pre00]最早由美国明尼苏达大学（University of Minnesota）开发，现在由 SISTINA 公司进行维护。GFS 是基于共享存储设备的对称式文件系统。主要特定描述如下：基于 Block I/O 的数据访问方式；没有元数据服务器，采用设备锁或者专用的锁服务器互斥访问文件系统的元数据；使用存储虚拟层 NSP（Network Storage Pool）管理底层的存储设备；使用扩展 Hash 技术（Extendible Hashing），提高目录操作的处理速度，这一点与 GPFS 相似；GFS 在虚拟设备（Storage Pool）上存储空间的管理类似于 Ext2 文件系统，但是它可将数据在不同的设备之间进行分布存储；提供元数据日志功能。

2.1.7 Lustre

Lustre[Bra02]是由 Cluster File System Inc. 针对传统机群/分布式文件系统的不足设计的高性能、可扩展、高可用的面向网络计算环境的分布式文件系统。2003 年 11 月，安装在 UIUC（University of Illinois at Urbana-Champaign）NCSA（National Center for Supercomputing Applications）实验室的 Linux 机群上的 Lustre 文件系统获得了 11.1GB/s 的世界最高聚合带宽。Lustre 文件系统主要采取了如下的技术：（1）采用元数据服务器负

责处理元数据请求，元数据服务器采用了复制、故障切换技术增加可用性；（2）采用基于对象的存储（OBD, Object Based Device）模型，由 OST（Object Storage Target）负责实际的文件 I/O 操作；（3）通过使用设计良好的通信模块支持异构的网络硬件；（4）采用了 XML、LDAP 和 SNMP 等技术提高系统的可管理性。

2.1.8 NCIC 的分布文件系统研究

本小节简要回顾国家智能计算机研发中心（NCIC）在网络/机群文件系统领域的研究历程。COSMOS 是曙光 2000 与曙光 3000 超级服务器的机群文件系统，为了便于区分，论文将前者称之为 D2K-COSMOS[Wjy99][Zc99][Zhk99]，后者称之为 D3K-COSMOS [Fen01][He02-1][Wu02]。

D2K-COSMOS 分布文件系统

D2K-COSMOS 是国家智能计算机研发中心在 1998 年为曙光 2000 超级服务器研制的一个机群文件系统。其主要特色如下：

- ✧ Serverless 结构，没有集中式的文件服务器，所有 D2K-COSMOS 系统中的节点既可以是服务器也可以是客户机，服务器可分为存储文件元数据（包括目录数据与文件属性）的元数据服务器与存储文件数据的文件服务器两类；
- ✧ 文件系统客户端使用协作式缓存（Cooperative Cache）算法与客户节点间文件数据转发来提高 D2K-COSMOS 用户的文件读写性能；通过复杂的写无效令牌缓存协议实现了文件间共享存取的 UNIX 语义，即最近的写操作结果总被随后的读操作所见；
- ✧ 文件服务器上以普通磁盘文件来模拟磁盘块，对于单个文件，其数据以类似于 RAID-0 的方式存放在多个文件服务器上；
- ✧ 通过 vnode/vfs 层作为扩充文件系统接口，实现与 AIX 宿主操作系统的无缝连接，提供程序的二进制兼容性。

D2K-COSMOS 的主要缺陷在于使用了过于复杂的缓存协议，加上 AIX 操作系统这种封闭的体系结构，使系统的调试变得非常艰难。虽然王建勇博士[Wjy99]在理论证明了相关算法的正确性，但理论证明中的若干条件对现实系统的简化其实并不准确，也对于系统中大量的异步事件无法进行完整的理论证明。但总的来说，D2K-COSMOS 在协议构造、协议理论证明等方面进行了有益的探索，但在可用性与性能方面仍未达到生产性应用的实际需求。

D3K-COSMOS 分布文件系统

D3K-COSMOS 属于曙光 3000 超级服务器的一部分，它有两个版本，分别基于 Linux[Wu02]与 AIX 操作系统[Fen01][He02-1]。D3K-COSMOS 保留了 D2K-COSMOS 设计中的一些基本概念，如通过 vnode/vfs 层扩充文件系统方式、共享体系结构、服务器分类以及存储分组等。但是 D3K-COSMOS 在总体上更注重于系统的可靠性与稳定性，在缓存协议上进行了较多的简化工作，通过简化文件共享存取的系统语义来减小系统开发的难度；另外对于基于 Linux 开放源码操作系统的版本，系统比较容易调试。

2.1.9 计算所工程中心的蓝鲸存储系统

蓝鲸存储系统[Tia03]是由计算所工程中心研制的大型网络存储系统，该项目研究建立在 863 项目－以海量存储为核心的网络服务器系统项目上。蓝鲸网络存储系统采用如下技术：（1）蓝鲸网络存储系统的存储设备基于网络存储，使用块接口；系统服务器使用专用服务器模式，对外提高文件级的数据接口，因此，蓝鲸系统融合了 IP-SAN 和 NAS 两种结构。（2）采用了多元数据服务器的结构来提高系统元数据的处理能力，元数据到服务器间的映射采用了动态映射策略，易于实现服务器间的动态负载平衡，易于扩展和管理。

2.2 机群文件系统目标

为了更好地支持机群系统中各类应用，一个机群/分布式文件系统应当：（1）支持全局文件共享；（2）提供高性能，包括文件读写性能和元数据操作性能；（3）具有良好的可扩展性；（4）具有较高的可用性；（5）易于管理。在本小节中，作者从上述五个方面描述了机群文件系统的设计目标。

2.2.1 全局文件共享

机群文件系统负责管理系统中的存储设备，不仅向用户提供了共享的存储空间，而且通过实现全局名字空间和文件接口，向用户提供了文件级别的共享。大多数 UNIX 文件系统基于底层操作系统 VFS（Virtual File System）实现了符合 POSIX[ISO96]标准的接口,在一些文件系统[Car00][Sch02]中，为了给用户提供更灵活的操纵文件的手段，定义了一组用户级文件接口或提供了对 MPI-IO 接口的支持。

机群文件系统提供全局名字空间，使系统中任何一个文件系统的客户节点在 mount 文件系统后，看到的是一个完全一样的树型目录结构。为了实现全局名字空间，机群文件系统必须实现系统的透明性和文件共享语义。文件系统的透明性包括结构、存取、命名、复制的透明性和用户的可移动性[Sin97]。本质上讲，这些特性都与文件系统的位置透明性有关。结构透明性要求客户(client)不应知道文件服务器及存储设备的数目和位置；存取透明性保证了客户可以同样的方式来存取本地和远程文件；如果文件的名字中不含有有关文件位置的(提示)信息则称该文件系统具备命名透明性；许多机群/分布式文件系统为了提高系统的性能和可用性采用了缓存和复制技术，系统中同一个文件可能在多个结点上有拷贝副本，多个拷贝的存在及其位置等信息对客户来讲应该是被屏蔽掉的，这就是所谓的复制透明性。用户的可移动性则要求系统支持用户在不同的时间、不同的结点上工作的灵活性，并且用户感觉不到 I/O 性能有太大差异。另外，几乎所有的机群/分布式文件系统为了改善性能都引入了缓存机制来减少数据传输，这同时使文件系统共享语义的问题更为复杂，文件系统必须提供解决缓存一致性问题方法。

2.2.2 性能

一个设计良好的机群文件系统能够充分发挥系统中网络、CPU、内存和外存储设备的性能，给用户提供高性能的文件服务。应用系统的负载模式千差万别，不同的负载特性需要文件系统提供不同的文件服务。对于对读写带宽要求较高的应用程序，例如数据密集型

的科学计算程序，需要文件系统能够提供足够的读写带宽；而对于元数据密集性的应用，如 web、Email 等应用，需要文件系统具有较高的元数据处理能力。

网络和存储设备的不断发展为机群文件系统性能的提高提供了基础。VIA[VIA97]等 memory-to-memory 网络协议的出现使机群文件系统可以减少节点间数据传输的开销，同时使建立在其上的文件协议[DAFS01]更精简。磁盘、RAID 等外存储设备性能的提高使文件系统能够获的更好的性能。基于 SAN 的机群文件系统[Sol97]由于在 IO 路径上相对于传统的客户/服务器结构的系统更接近与底层的存储设备，可以获得更高的性能。

机群文件系统的结构和协议的设计对于文件系统的性能也有重要的影响。机群文件系统结构应当尽量避免出现瓶颈，并尽可能发挥系统组成部件间的并行性，在第三章中作者将对机群/分布式文件系统中各种结构进行讨论。机群/分布式文件系统协议可以从如下方面进行考虑来提高系统性能：（1）一个协议请求可能需要穿越多次网络才能完成，设计良好的文件系统请求流程应当尽可能的减少网络穿越的次数；（2）客户端缓存使客户端请求由本地完成；（3）服务器端缓存可以避免频繁的读写磁盘的操作；（4）采用异步 I/O 可以使多个 I/O 请求并行执行，使 CPU、磁盘、总线等能够以流水线方式重叠执行，减少了 CPU 及各 I/O 部件的空闲时间，提高了 I/O 吞吐率。

2.2.3 可扩展性

机群文件系统的一个重要的指标就是要提供良好的可扩展性，我们认为，机群文件系统的可扩展性包括：（1）资源的可扩展性，机群文件系统应当可以随着系统规模的增大而增大，通过往机群文件系统中增加服务器或存储设备来满足系统对性能和存储容量的需求；（2）性能的扩展性，首先，机群文件系统应该易于适应机群系统中结点和用户数目的增长，也就是说，这样的增长不应引起系统不能正常工作或 I/O 性能的急剧下降，其次，当增加系统中可用于 I/O 的资源(如磁盘、CPU 等)时，文件系统应该能够很快的利用新的资源，并且做到较好的负载平衡，其文件系统的 I/O 整体性能也随之增长。

文献[Hwa98]给出了一个可扩展分布式系统设计中的原则：平衡原理。这个原理指出，要避免不平衡系统的设计，努力最小化任何的系统瓶颈。有几种基本的方法可以用来提高机群文件系统的可扩展性(也提高了单个结点得到的性能)：（1）消除集中式服务器带来的系统瓶颈，当系统规模变大时，单一文件服务器容易成为性能瓶颈，一个解决方案就是实现分布的存储和控制[Wjy99]，采用多个服务器或者无服务器的结构。（2）采用易于扩展的体系结构，文件系统中的各种资源，如 CPU，存储设备，网络等应该彼此依赖性小，可以灵活地增加资源以满足各种应用程序的需要； (3)机群文件系统的瓶颈可能在于网络、服务器的处理能力或者服务器的磁盘，因此，采用高带宽低时延的网络、更快的服务器和存储系统可以提高系统可扩展性， RAID[Che96][Wil96]和网络磁盘分组[Wjy99]是比较成熟的技术；（4）减少网络传输，减少服务器磁盘读写，包括客户端和服务器端缓存等技术。

2.2.4 可用性

Patterson 在[Pat02]中指出，随着存储技术的发展，人们将越来越关注于存储系统的可用性和可维护性，可用性和可维护性的重要性将超过性能和可扩展性。机群文件系统作为用户数据的载体，必须提供可靠的文件数据服务。相对于采用单一文件服务器的文件系统，

采用多文件服务器、存储服务器和元数据服务器分离、网络存储分组等技术文件系统为实现高可用的文件系统提供了可能性。在分布式系统中，往往是部分节点或网络部件会出现故障，而不是整个系统的崩溃。人们通常希望一个系统中某些部件出现故障时，系统仍然能够继续运行，系统整体性能可能会有所下降，但当发生故障的部件重新变为可用时，系统应迅速恢复到正常运行状态。但机群文件系统分布式的结构使高可用协议的实现更加复杂。

对于文件系统而言，提供可靠的文件服务需要维护元数据一致性和数据一致性，并且当机群文件系统发生故障时能够恢复到某个一致性状态。机群文件系统维护统一的名字空间，名字空间记录了文件之间的关系，文件系统必须保证这种关系的正确性，用户才能访问到文件，同时，机群文件系统维护元数据间关系的一致性和元数据本身的正确性。在多个应用程序并发访问的情况下，不正确的并发控制协议可能会引起元数据的不一致；对于多元数据服务器的文件系统，一个元数据操作可能涉及多个元数据服务器，元数据服务器间需要相互协作以维护名字空间的一致性。数据一致性保证各个文件的数据的完整和正确。

2.2.5 管理

随着机群文件系统规模的扩大，如何有效地管理整个系统的资源，使用户易于使用成为一个关键的问题。一个大规模的机群系统通常配置大量的存储设备，机群文件系统及其管理工具应当能够合理的分配、添加设备，尽量减少管理员的干预。在一个大型系统中，文件系统的部署是一件极其繁琐的工作，机群文件系统管理工具应当在文件系统的安装与卸载、文件系统服务的启动与停止等方面提供有效的手段。机群文件系统的管理工具还能够监测、报告文件系统各组成部件的状态，隔离出现故障的节点。另外，机群文件系统管理工具应当具有友好的界面，使用方便。

2.2.6 典型系统总结

本节给出了机群文件系统设计的目标：全局文件共享、高性能、可扩展性、可用性和可管理性，不同的文件系统采用各种技术以实现文件系统的上述目标，表 2.1 简单的列举了 2.1 节中介绍的典型文件系统在这些方面采取的技术。表中空白的栏表示相关文献中没有给出明确的说明。

表 2.1 典型文件系统系统比较

文件系统	全局文件共享	性能	可扩展性	可用性	管理
NFS v2.0	单一文件服务器结构	弱一致性的客户端缓存	单一服务器结构限制了NFS的扩展性	无状态文件系统服务流程	—
AFS	多文件服务器结构，文件服务器对用户透明，系统中的卷通过 mount 协议构成统一的名字空间	弱一致性的客户端缓存	通过增加文件服务器和卷来扩展服务	服务器复制 卷复制	—

xFS	多元数据服务器，分布在各服务器上元数据构成统一名字空间	合作式缓存 数据分条 多元数据服务器结构提高元数据处理能力	Serverless 的体系结构	日志式文件系统	—
PVFS	单一元数据服务器 用户级文件访问接口	数据分条	多个存储服务器使PVFS 读写具有较好的可扩展性，单一元数据服务器的结构限制了元数据性能的扩展	——	—
GPFS	基于 VSD（Virtual Shared Disk）的共享存储结构，元数据存储在共享设备上	严格 UNIX 语义的客户端缓存 数据分条 大目录优化	VSD 的体系结构	可恢复的 VSD 元数据日志技术 数据复制 RSCT (Reliable Scalable Cluster Technology)技术	文件系统中增加配置管理器，提供支持多节点执行的管理命令和工具
GFS	基于 SAN 的结构	基于 SAN 的体系结构 客户端缓存 大目录优化	基于 SAN 的体系结构使GFS 易于扩展存储系统	日志与事务技术相结合	—
Lustre	单一元数据服务器	严格 UNIX 语义的客户端缓存 基于 OSD 的存储结构	基于 OSD 的结构使其易于扩展存储系统，	主从元数据服务器结构	XML 、LDAP 和 SNMP
COSMOS	多元数据服务器结构，分布在各服务器上元数据构成统一名字空间	合作式缓存 数据分条 多元数据服务器	多元数据服务器 网络磁盘分组	—	—
蓝鲸	多元数据服务器结构	动态元数据分布策略	基于 IP-SAN 的体系结构 多元数据服务器结构	—	—

2.3 机群文件系统关键技术

2.3.1 体系结构

机群文件系统建立在由网络和存储设备构成的硬件平台上，研究人员根据机群文件系统所基于的平台提出了各种体系结构。机群文件系统结构可以分为两类[Dev95]：（1）基于客户/服务器（Client/Server）模型的机群文件系统结构；（2）基于共享设备(shared decive approach)结构。在基于客户/服务器模型的文件系统的结构中，系统中某些节点从逻辑上作为文件服务器为其他节点服务。在基于共享设备的结构中，文件系统的客户节点可以直接存取存储设备上的文件数据，该结构包含基于共享块设备（如 SAN、IP-SAN）的结构和基于 OBD 的结构。本文在第三章中将分别对这两种结构进行分析。

基于曙光 4000L 的具体体系结构和 DCFS 设计时技术发展的条件，DCFS 采用了基于客户/服务器模型的结构，在此基础上提出了灵活、易于扩展的多文件系统卷结构以及一些优化的技术，性能测试的结果表明 DCFS 采用的结构是可行的。由于 DCFS 采用了基于客户/服务器模型的结构，因此，在本章后面的小节中描述了与客户/服务器模型的结构相关的关键技术。

2.3.2 缓存[He02-2]

机群文件系统为了提供性能普遍采用了缓存技术，而由于机群文件系统分布式的结构增加了系统缓存实现的难度，在本小节中，从文件共享语义、文件缓存实现位置等方面对缓存技术进行讨论。

文件共享语义

机群文件系统的文件缓存机制会带来缓存一致性问题。文件系统语义表征了文件访问的效果[Sin97]，而文件共享语义所解决的主要问题是当在多个进程间访问相同文件时能获得的结果。不同的文件共享语义定义了对于缓存一致性问题的不同解决方案。参考文献[Sin97]中定义了 4 类常用的 UNIX 文件共享语义：UNIX 语义、会话期间（Session）语义、不可改变的共享文件语义（Immutable Shared Semantics）以及事务处理型语义（Transaction-like Semantics）。而上述几种语义并不能适合所有实际系统中应用[Bur00]，该文针对 Storage Tank 的应用设计了合理的文件语义。

文件缓存实现

根据文件缓存所在位置，机群文件系统中的文件缓存可以划分为以下两类：客户节点缓存与服务器缓存，其中客户节点缓存既可以位于内存，也可以在本地磁盘中。文件服务器的文件缓存较容易实现；客户节点缓存需要维护各节点间缓存的一致性，实现较为复杂。客户节点内存中的文件缓存距离用户应用最近，读写性能最好，但实现强一致性语义较困难且容量有限，客户节点的宕机会给系统可靠性带来一定负面影响；客户节点物理磁盘上的文件缓存相对内存中的文件缓存具有更好的可靠性和更大的容量，但由于在物理磁盘上，存取速度较慢，常用于弱连接（即客户节点并不总保持与文件服务器的连接状态）的网络文件系统，如 Coda[Mum96]、AFS[How88]都使用了客户节点磁盘作为文件缓存。

文件写策略

文件缓存写策略决定了何时把修改后的缓存内容写回服务器端的磁盘上。在网络文件系统中，它对系统性能与可靠性有较大影响，另外文件共享语义与文件写策略也紧密相关。文件写策略主要分为立即写（Write-Through）和延迟写（Delayed-Write）两种策略，它们与高速缓存/内存子系统中的写穿（Write-Through）与写回（Write-Back）策略具有一一对应关系[Hen96]。

文件缓存有效性验证

在网络文件系统中，由于在不同的共享客户节点上可能存在多个文件数据块缓存副本，因此需要专门的客户缓存验证策略以判断客户节点上文件缓存的有效性。根据文件缓存有效性验证过程发起者的不同，有客户节点启动和文件服务器启动两种有效性验证方案[Wjy99]。客户启动的方案当有效性检查的频率较高时会带来大量的网络通讯和服务器CPU开销，服务器启动的方案可以克服这一缺点，但需要在服务器中维护大量有关缓存的信息。

文件缓存粒度

文件缓存粒度指系统可管理的文件缓存基本单元大小，选择合适的文件粒度有助于减少开销与提高性能。网络文件系统中的缓存粒度有文件级和数据块级两种，采用文件级粒度的文件缓存策略实现相对简单，但由于文件级粒度限制了对文件的写共享，在文件写共享频繁的负载中将极大的影响文件系统的性能。当采用数据块级粒度时，提高了文件系统访问的并发性，文件缓存的效率较高，但是不利之处在于实现算法比较复杂。

2.3.3 存储分布

对于客户/服务器结构的机群文件系统，当系统规模变大时，单一文件服务器容易成为性能瓶颈，一个解决方案就是实现分布式的存储和控制，快速、可扩展的网络技术使分布式磁盘存储成为可能。分布式存储还有助于提高 I/O 的并行度，多个用户发出的多个 I/O 请求可以由多个存储服务器同时执行，能够较好地克服单一存储服务器瓶颈问题。同时，机群文件系统采用了分条（Striping）等并行 I/O 技术来提高系统 I/O 带宽，一个文件存放于多个服务器上，读/写操作可以转化为对多个服务器的访问，大大提高了 I/O 的带宽。

基于共享磁盘结构的机群文件系统也采用了分条（Striping）技术将每个文件数据块分布到多个磁盘设备中，可以充分发挥多个磁盘的并行性来服务大数据量的文件读写请求。GPFS[Bar98][Sch02]选择在文件系统层中实现分条策略而不是独立的 LVM（Logical Volume Manager）层，使 GPFS 可以根据整个文件系统的配置和负载情况更灵活地选择分条策略。在 GPFS 中，文件被分成多个相同大小的块，以随机或 round-robin 方式存放到不同的磁盘中，由于 GPFS 主要应用于科学计算和流媒体服务中，块的大小通常为 256kB。

许多机群文件系统，特别是一些并行文件系统，提供了比 UNIX 文件系统标准接口更为丰富的接口[Car00][Lig99]。利用这些接口，应用程序可以控制文件在系统中的分布，使文件的分布与应用程序的文件访问模式相匹配，能够获得更高的性能。

2.3.4 元数据分布

许多文件系统采用了元数据和数据分离的结构，即元数据和数据分别由不同服务器进行处理，这是因为：（1）有些文件操作无需涉及文件的数据，而只与文件属性或者目录数据等元数据信息有关，因此可以将这两部分任务分派给两类不同的服务器 ，通过减轻服务器的负载以提高客户节点文件操作性能；（2）通过分离文件元数据与数据，使得两种类型的服务器都可以根据需要进行扩展，从而增强系统的可扩展能力；（3）在服务器的磁盘上，由于文件元数据信息与文件数据具有不同的存取特性与尺寸，比如一般元数据信息要远远小于文件数据等，因此可以在服务器方设计不同的本地存储策略，通过优化服务器性能来提高客户节点文件操作性能。

为了增强文件系统处理元数据的能力，许多机群/分布式文件系统采用了多元数据服务器的结构，多元数据服务器的结构通常可以分为两类：（1）集中存储分布处理[Tia03]；（2）分布存储分布处理[And02][Ji00][Zha01]。在（1）的结构中，元数据存储在共享的设备上，任何元数据服务器可以看到相同的元数据并处理对元数据的请求；而在（2）的结构中，元数据按照一定的分布策略分布到各个元数据服务器上，一个元数据服务器只能看到由其自身管理的元数据和处理与其管理的元数据相关的请求。在第三章中，作者将对这两种结构进行分析。

2.3.5 目录操作优化

文件系统的目录树维护着文件之间的关系，目录的操作包括 LOOKUP、READDIR、REMOVE 等操作，由于这些目录操作在文件系统的操作中占了很大的比重[Rob99]，因此，优化目录操作可以改善文件系统的性能。对机群文件系统而言，目录操作的路径包括客户端、网络和服务器，优化目录操作的方法有：（1）客户端目录缓存，可以减少目录操作穿越网络的次数；（2）优化文件系统协议，减少向服务器发送请求的次数。文[He02-2]的作者对 LOOKUP 的操作流程进行了分析，指出传统 VFS（Virtual File System）实现中路径名解析的缺陷，提出了全路径名解析的技术（Full Path Lookup Strategy）；（3）服务器端优化，包括服务器端缓存和更有效的目录组织技术。

目录缓存作为改善目录操作性能的技术被文件系统广泛采用，独立的元数据服务器使得服务器上的目录缓存管理可以采用更有效的技术，在第四章中，作者对元数据服务器上的目录缓存进行讨论。许多文件系统中目录文件按传统的线性方式组织，当目录中文件个数增多时，由于线性查找的开销增大，目录操作的性能将下降，因此，研究人员针对大目录文件提出了优化的技术，其中 B-树[Rei03][XFS03]和动态 Hash[Sch02][Pre99]是两种常用的技术。B-树是一种平衡的多路查找树，采用 B-树作为目录文件组织方式的文件系统通过维持树的平衡状态来减少搜索的时间。动态 HASH 技术希望通过文件名的 HASH 值可以直接索引到目录项，该方法需要维护一个不断变化的 HASH 表。在第四章中作者将对大目录优化进行讨论并给出动态 Hash 技术的一种改进算法。

2.3.6 可用性技术[Sxd02]

机群文件系统通常包括多个节点，文件系统中各个组成部分，包括节点、网络、存储

设备的故障可能会对文件系统中其他部分造成影响，机群文件系统的高可用性技术必须能够在出现故障时能够正确地处理。机群文件系统的可用性技术包括系统失效侦测、复制技术、日志式存储管理、元数据一致性技术。

系统失效侦测

准确及时地发现系统中的故障是实现系统高可用的重要环节。如果故障未能及时检测出来，将严重地影响系统的可用性。同时如果经常发生虚警而使系统产生不必要切换则将降低系统的效率，甚至导致关键数据的丢失。心跳[Kal97]（Heart Beat）技术和 Agent 技术是常用的两种方法。

复制技术

许多文件系统采用了复制技术来增加系统的可用性[How88][Mum96][Guy90]，通过保存多个副本，使系统可以在发生故障时由副本继续提供服务。复制技术在提高可用性的同时也引入了维护多个副本间一致性的开销。通常的方法包括只读复制（Read-Only Replication）、读任意副本/写所有副本协议（Read-Any-Write-All Protocol）、可用拷贝协议（Available-Copies Protocol）、主拷贝协议（Primary-Copy Protocol）和基于定额的协议（Quorum-Based Protocol）等。关于这些协议的具体解释可参考文献[Sin97]。

日志式存储管理

现在许多本地文件系统采用了日志式存储的技术，其设计思想主要是通过连续、异步的磁盘存取来提高写操作的速度[Mat97][Sta97]。它把磁盘看作被划分为若干片段的、且只可附加的日志（log），在保持了现有文件系统的快速读取的基础上，以文件缓存作为写操作的缓冲来收集对文件系统的修改，充分利用磁盘的最大带宽，以大块连续的方式将包括数据、目录和元数据（如：inode）在内的内容写回磁盘。在机群/分布式文件系统中，Zebra[Har94]和 xFS[And95]有机地结合了日志式存储(LFS)和网络 RAID 的优点，使得二者在分布式环境中都能工作得很好。

基于事务的技术

事务处理广泛应用在数据库领域中[Ozs99]，用于保证在并发访问的环境下各种操作的正确性和数据库的一致性，并能够在发生故障时将数据库恢复到一致性状态。许多文件系统借鉴了事务处理的技术来增强可用性。由于 GFS[Pre99]等对称共享存储结构的机群文件系统的节点可以直接共享存储设备，其事务处理的过程与本地文件系统（如 EXT3）类似。对于包含多个服务器的机群文件系统，特别是多元数据服务器的系统，由于一个操作可能涉及多个服务器，因此在这些系统中引入了分布式事务的技术。例如，为保证元数据一致性，Slice[And02]和 Archipelago[Ji00]在元数据操作流程种结合了事务处理中两阶段提交的协议。Diffs[Zha01]认为事务处理给正常的处理流程引入了很大的开销，提出了更为简洁的协议，第五章中作者将进行分析。

2.3.7 实现方式和位置

根据文件系统客户端的实现方式和位置，可以分为：（1）用户级文件系统，用户级文件系统提供一组库函数作为文件接口，用户通过这组接口使用文件系统提供的服务，如Galley[Nie96]文件系统；（2）核心级文件系统，如 GFS[Pre00]等文件系统，这类的文件系统基于操作系统的虚拟文件系统 VFS[74]实现，向用户提供了标准接口；（3）另一些文件系统，如 Ufo[Ale97]，文件系统的功能是通过用户级程序实现的，同时通过截获系统调用的方法向用户提供标准接口。图 2.1 给出了这三种实现方式的示意图，图中的横线下方表示操作系统的核心空间，上方表示用户空间。另外，一些文件系统采用了多种实现方式来提供多种访问接口，如 PVFS[Car00]，其在客户端不仅实现了核心模块以提供二进制兼容的 POSIX 文件访问接口，同时也实现了一组用户级的文件访问库函数，以提供更为灵活的操纵文件的接口。

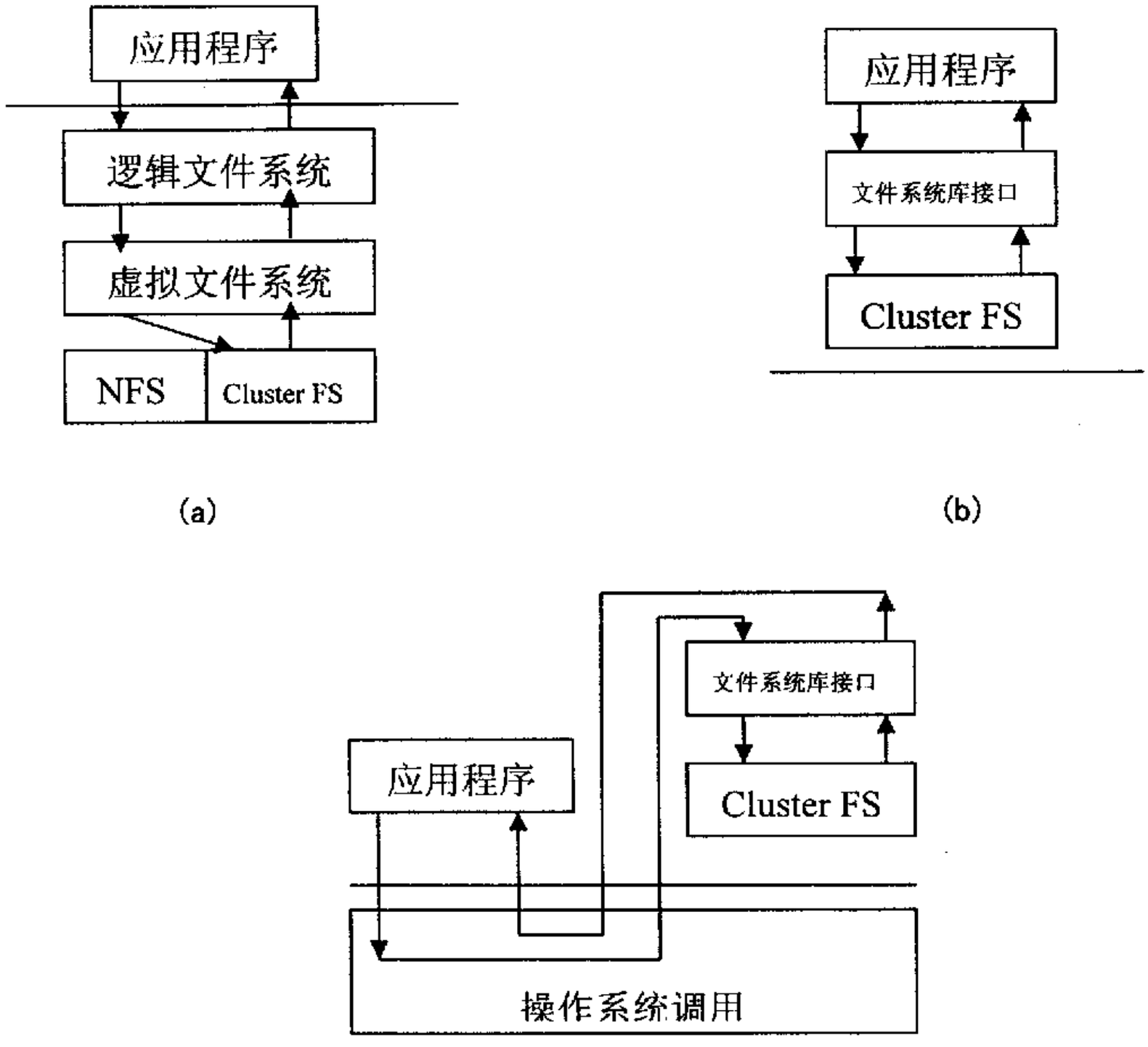


图 2.1 文件系统实现方式和位置

（a）核心级文件系统，（b）用户级文件系统，（c）截取系统调用的方法

用户级文件系统的功能在用户级实现，应用程序通过一组库函数使用文件系统，相对于核心级文件系统，用户级文件系统提供了更灵活的控制文件系统的手段，例如缓存的策略、预取的策略、数据的分布以及对文件的操作等，另外，易于编译和调试，可以利用用户级通讯机制；但用户级文件系统不能实现应用程序的二进制兼容性。核心级文件系统使用操作系统提供的 VFS 接口实现，使机群文件系统可以保持应用程序的二进制兼容性，它

保证了现有的应用程序不用重新编译和链接，就可以直接运行在机群文件系统上。同时，机群文件系统可以利用 VFS 提供的缓存和预取策略。Ufo[Ale97]等文件系统为保证应用程序的二进制兼容性采用截取系统调用的方法，而文件系统功能则完全在用户级实现的。

2.4 小结

本章首先简要介绍了典型的机群/分布式文件系统，然后给出了实现一个全局共享、高性能、可扩展、高可用以及易管理的机群文件系统采用的关键技术，包括合理的体系结构、存储和元数据的分布等，由于机群文件系统复杂，涉及的方法和技术很多，在本章中只是总结了与 DCFS 设计和实现密切相关的技术。

第三章 机群文件系统体系结构研究

机群文件系统的体系结构对文件系统的性能、可扩展性以及可管理性等方面都有着重要的影响，因此，机群文件系统结构的设计也是整个系统设计的关键。在本章中，作者首先分析了当前机群/分布式文件系统的体系结构，然后提出了多文件系统卷的结构并在DCFS 机群文件系统中实现，该结构包含网络磁盘分组和元数据服务器组织两个部分，作者分别对这两部分内容进行了分析和试验。

3.1 机群文件系统体系结构

作者在第二章中简要介绍了机群文件系统的两类结构：（1）基于客户/服务器（Client/Server）模型的机群文件系统结构；（2）基于共享设备(shared device approach)结构。从单一文件服务器结构的 NFS[NFS89][NFS95][NFS03][Cal00]到多文件服务器结构的AFS[How88]以及无集中服务器结构的 xFS[And95]文件系统都采用了第一种结构，而采用共享设备结构的系统有 GPFS[Sch02]、GFS[Pre00]和 Storage Tank[Bur00]等。在本节中，作者对这两类结构进行分析。

基于客户/服务器（Client/Server）模型的机群文件系统把文件系统的功能分布到客户和服务器上来完成这一任务，客户端的读写请求由服务器完成，客户端不能直接操纵存储设备。服务器节点负责存储和管理文件数据和元数据，客户节点利用 VFS 接口实现一个新的虚拟文件系统，并通过给服务器端发送文件系统协议请求获得服务。xFS 提出了 serverless 的结构，文件系统中所有的节点地位是平等的，但可以看到，xFS 的读写请求同样需要由存储服务器（Storage Server）完成，存储设备对客户端透明，因此，我们将 xFS 归类为客户/服务器结构。

在基于共享设备的结构中，按照共享设备的类型可以划分为：（1）基于块设备的结构，单一系统文件系统的多个被序列化的实例可以直接以存储设备块的方式存取文件数据。由于这些独立的文件系统实例可以从不同的结点上直接存取块，并且为了提高性能通常要对被存取的块进行缓存，因而必须采用相应的措施使多个实例对磁盘的存取序列化，并保证数据一致性，可以采用加锁或令牌机制来实现；（2）基于 OBD 的结构，存储系统与文件系统的接口为面向对象的接口。

3.1.1 基于客户/服务器模型的结构

图 3.1 给出了客户服务器结构的文件系统示意图，在 3.1（a）的结构中，每个服务器的功能相同，负责处理数据和元数据的请求。NFS 采用了单服务器的结构，AFS 等为了避免单服务器结构出现的性能瓶颈和单一故障点问题，更好的应用于大规模分布式环境中，将单服务器的文件系统体系结构扩展为多服务器的结构。

图 3.1（b）服务器分为元数据服务器和存储服务器两类。与 3.1（a）中的结构相比，元数据和数据处理的分离使元数据请求和数据读写请求由不同的 IO 路径完成，使服务器

的负载减轻，并且可以根据元数据和读写处理的特点分别对服务器进行优化，存储服务器和元数据服务器可以根据需要分别进行扩展，该结构使文件系统协议相对复杂，一个文件服务请求可能需要多个服务器的配合才能完成，同时增加了文件系统高可用协议设计的难度。xFS[And95]对元数据服务器和存储服务器分离的结构作了进一步的扩展，提出了 Serverless 的概念，文件系统中每个节点的地位是对等的。[Tia03]中指出，在 IDC 等应用和实际的系统中，Serverless 结构对于系统中所有节点是平等的（包括处理能力、系统访问权限和安全性等）的假设是不成立的，因此 Serverless 的结构并不适用。

如图 3.1(b)结构的文件系统通常采用将存储服务器分组以及分条（stripe）的技术提高系统的读写性能、可扩展性和可用性，如 xFS 中采用的 Stripe Group Map 技术，将存储服务器划分为多个组，资源的扩展可以通过增加存储分组实现，数据在每个组中按照分条的形式存放，使存储服务器可以并发地服务多个读写请求，每个读写请求由多个存储服务器并行处理。

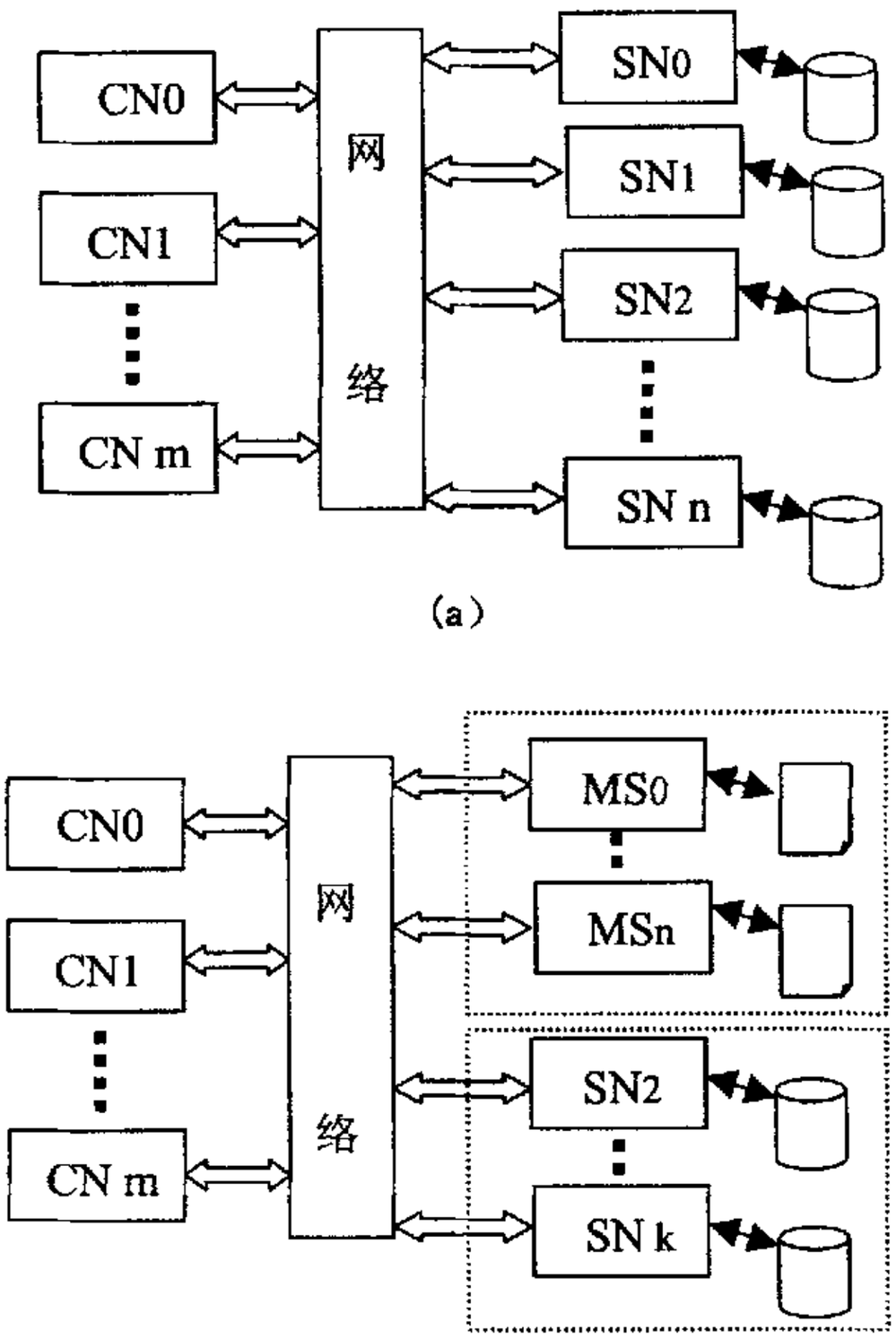


图 3.1 基于客户/服务器模型的文件系统结构，（a）中每个服务器具有相同的功能，完成元数据和数据的读写操作，（b）元数据服务器和存储服务器分离

3.1.2 基于共享设备的结构

图 3.2 给出了基于共享设备的机群文件系统结构，基于共享设备的结构可以分为对称式结构和非对称式结构，分别如图 3.2(a)和(b)所示。在图 3.2（a）所示的结构中，所有的

客户节点看到相同的存储设备，数据和元数据都存放在共享的存储设备上，数据和元数据的存储需要访问存储设备，GFS[Pre99]采用了这种体系结构。在对称式结构中，需要采取专门的设备锁或锁服务器保证对元数据以及数据的互斥访问。

图 3.2（b）给出了另外一种基于共享设备的机群文件系统结构[Bur00]，在该结构中，元数据由专门的服务器进行处理。专门的元数据服务器既简化了元数据访问的序列化问题[Moh94]，并且如 3.1（b）的结构，非对称式结构中数据存取路径和元数据路径的分离在带来性能和可扩展性提高的同时，增加了系统的复杂性和实现元数据一致性协议的难度。

基于共享设备的文件系统所采用的存储系统，除了 SAN、IP-SAN 和虚拟存储设备（VSD）外，随着 OBD 等新型存储系统的出现，基于 OBD 的文件系统[Bra02]逐渐成为研究的热点。与基于共享的块设备的机群文件系统相比，OBD 文件系统同样旁路了文件服务器，缩短了 IO 路径，同时由于将存储空间管理等任务划分由 OBD 完成，OBD 文件系统避免了基于块设备机群文件系统中对设备的并发访问的管理，具有更好的可扩展性。

基于共享设备的文件系统结构通过将存储设备分组来提供系统的可扩展性和可用性[Sch00][Bur00]。Storage Tank 将整个存储空间划分为多个存储池（Pool），通过添加存储池中的设备或增加新的存储池来实现系统资源的扩展，并且在某些存储设备出现故障时减少对其余设备的影响。

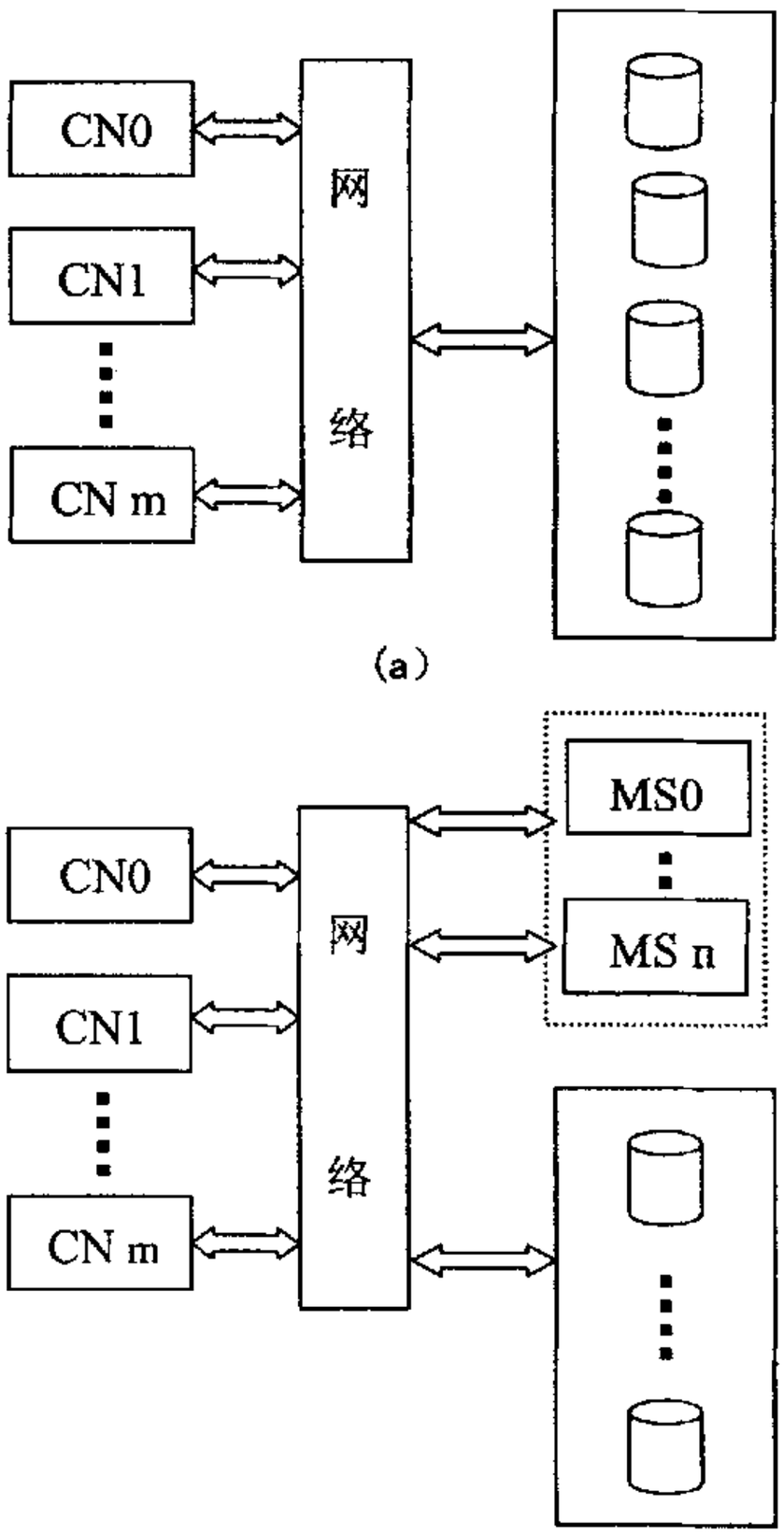


图 3.2 基于共享设备的文件系统结构，（a）对称式共享设备结构，（b）非对称式共享设备结构

3.1.3 小结

在下面作者对上述描述的机群文件系统结构进行简单的比较：（1）从性能上而言，对于元数据性能，由于现在的机群文件系统大都采用了独立的元数据服务器，客户/服务器结构和基于共享设备的结构具有相当的元数据处理能力，对于读写性能，基于共享块设备结构的文件系统 IO 路径更短，应当具有更好的读写性能，但由于客户端直接操纵存储设备，使读写过程引入了更多的额外开销，例如定位数据块位置、存储设备的并发访问等，因此，基于共享块设备所能获得的性能优势被消弱，而基于 OBD 的文件系统结合了客户/服务器结构读写过程简单和基于共享块设备结构 IO 路径短的优点，期望获得更高的性能；（2）从可扩展性而言，基于共享设备的结构消除了文件服务器可能造成的瓶颈，可以获得更好比客户/服务器结构的扩展性，但我们也注意到，对于基于块设备的结构，由于要协调存储设备的并发读写，其可扩展性会受到限制；（3）从实现的复杂程度和对设备的要求而言，基于共享设备结构的文件系统对设备要求较高，实现复杂，需要直接管理存储设备，如存储设备的虚拟化、块的分配与释放，以及对存储设备并发存取的控制等，而基于客户/服务器模型的结构对设备要求低，通用性好，相对简单，易于应用到分布式环境中。

我们对机群文件系统结构的总结以及上面的比较对我们设计机群文件系统的结构有如下启示：（1）客户/服务器结构和基于共享设备的结构各有优缺点，机群文件系统采用何种结构需要根据系统的要求和条件确定。（2）文件读写请求和元数据请求处理的分离使得文件系统可以根据读写请求和元数据请求的特点分别进行优化，分别进行扩展；（3）客户/服务器结构的文件系统和基于共享设备的文件系统都通过划分存储空间来增强可扩展性。

3.2.多文件系统卷

如 3.1 小节所述，与共享文件系统方法相比，共享设备结构对设备的要求高(高速网络和共享磁盘)，往往被用来构造高端或专用的存储设备，同时，基于共享设备的结构实现的难度大。而共享文件系统模型实现起来比较简单，它对于设备的要求不高，而且通用性好。基于曙光 4000L 的具体体系结构[Sun03]和 DCFS 设计时技术发展的条件，DCFS 采用了基于客户/服务器模型的结构。另外，我们可以看到，机群系统规模不断扩大，机群上各类应用程序表现出不同的负载特性，这些对客户/服务器结构的机群文件系统设计提出了新的挑战，需要对文件系统资源进行合理地划分，因此，我们提出并在 DCFS 中实现了多文件系统卷的结构，该结构扩展了存储空间划分的技术，将 DCFS 中的存储服务器和元数据服务器划分为多个子文件系统（称为文件系统卷）。在本节中，作者结合 DCFS 描述了多文件系统卷的结构，在 3.3 和 3.4 节中给出了多文件系统卷的结构中的两个重要的部分：存储服务器组和元数据服务器组中采用的技术并分别进行了分析。

3.2.1 DCFS 总体结构

图 3.3 给出了 DCFS 文件系统的结构示意图，虚线左边的部分为 DCFS 文件系统。DCFS 采用了基于客户/服务器（Client/Server）模型的结构，它由文件系统客户节点、文件元数据服务器节点、存储服务器节点及配置/管理节点四部分构成，这几部分之间通过互连网络连接在一起。

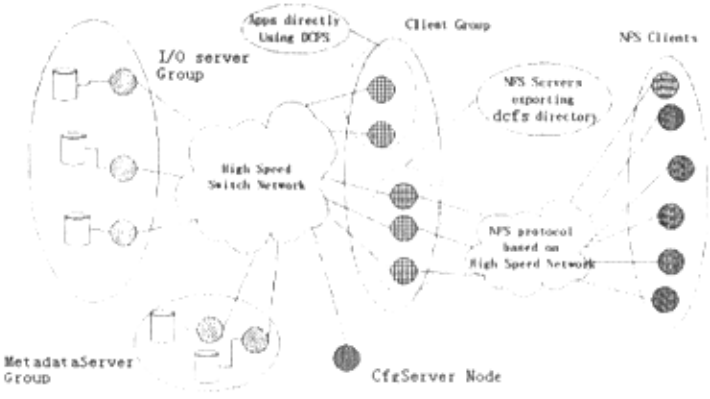


图 3.3 DCFS 组成示意图

存储服务器负责文件数据的存储、管理以及文件读写请求的处理。为了有效地管理文件系统的存储空间，同时也考虑大规模文件的带宽性能，DCFS 系统将它们划分为文件系统卷来管理，在每个 DCFS 文件系统卷内，存储服务器节点组织成多个网络磁盘分组的结构，文件的数据以类似 RAID-0 的方式来存储在某个网络磁盘分组中。在文件读写时，通过多存储服务器节点并发读写而获得比单存储节点结构更好的聚集文件 I/O 性能。

元数据服务器负责管理 DCFS 的文件数据分布信息、DCFS 目录文件及普通 DCFS 文件元数据（包括文件长度、权限、日期及其它属性信息）的存储。在提供客户端缓存与文件锁的 DCFS 中，元数据服务器还负责维护缓存与文件锁状态。每个卷的元数据服务器组中有一个超级管理器的特殊文件服务器进程，它负责管理 DCFS 文件系统卷的超级块以及其它重要信息。当存在多个 DCFS 文件系统卷时，系统中对应的会存在多个超级管理器。在系统配置时，DCFS 的配置协议将协助系统管理员把这些超级管理器进程分散在不同节点上以减少单个超级管理器节点崩溃时不会影响其它 DCFS 文件系统卷。

超级服务器的系统管理员通过配置管理器来管理 DCFS。这些配置与管理活动主要有启动与添加 DCFS 服务器、扩充服务器磁盘容量等；安装或卸载 DCFS 客户端接口以及监控所有 DCFS 节点状态等。

文件系统客户代理节点指加载了 DCFS 文件系统核心扩展与相关系统进程的机群节点。在这些节点上的管理员通过一组 DCFS 工具可以得知 DCFS 系统中目前的文件系统个数以及文件系统名称，然后他们通过建立本地的 DCFS 文件系统 mount 点就可以安装并存取 DCFS 文件系统。

3.2.2 多文件系统卷结构

与 3.1 小节中指出的存储空间的划分不同，DCFS 文件系统将服务器组（包括存储服务器和元数据服务器）划分为多个卷，每个卷可以配置成包括不同数目的存储服务器和元数据服务器以满足不同应用的需求，所有卷构成一个全局共享的名字空间，图 3.4 给出了 DCFS 配置为两个卷时的结构。DCFS 卷的存储服务器采用了存储网络分组的技术，具体的描述见 3.2.3 小节；DCFS 卷采用了多元数据服务器结构，并采用了可调粒度的元数据分布策略，作者将在 3.2.4 节中进行描述。

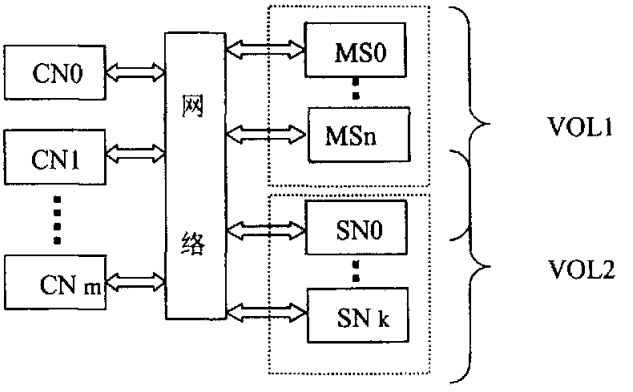


图 3.4 多文件系统卷配置举例

DCFS 多文件系统卷的结构与现有系统有如下不同：AFS[How88]中将整个名字空间划分为多个区（Cell），但 AFS 每个区是客户节点和服务器的组合，通常指系统中某个管理域，并且 AFS 每个区的服务器结构与 DCFS 不同，AFS 中的服务器的功能是相同的，同时处理文件读写请求和元数据请求，而 DCFS 的文件系统卷采用了存储服务器和元数据服务器分离的结构，每个卷中的存储服务器采用了 3.2.3 中介绍的网络存储分组的结构；xFS 中 Stripe Group Map 的技术和 Storage Tank[Bur00]等基于共享设备的文件系统只是对存储空间进行划分，而 DCFS 对文件系统资源的划分包括存储服务器和元数据服务器；LVM[LVM02][Sis02]等软件将存储设备划分为多个逻辑卷供上层软件使用，提供的是设备级的共享，而 DCFS 提供的是文件级共享。

多文件系统卷的机群文件系统结构对可扩展性、管理性、可用性、灵活性方面有如下的考虑：

(1) 扩展性。多文件系统卷的结构将 DCFS 的资源划分为三个层次：卷、服务器和存储设备。DCFS 的可扩展性表现在：（a）通过增加服务器和存储设备，并配置成一个新的卷添加到 DCFS 的全局名字空间中来增加文件系统的资源；（b）DCFS 文件系统卷采用了网络存储分组和多元数据服务器结构，可以增加服务器和存储设备到现有的卷中。

(2) 灵活性，指根据应用程序优化配置，更合理地利用系统资源。对于一个确定的机群文件系统配置，很难同时满足不同负载特性的应用程序的需求，在 DCFS 机群文件系统中，可以根据不同应用程序负载特性配置成多个子文件系统卷，图 3.4 给出了两个文件系统卷的配置，在文件系统卷 1 中，包括 n 个元数据服务器和 1 个存储服务器，用于服务于元数据密集型应用程序；而文件系统卷 2 则主要应用于科学类计算中，在这类应用中，对元数据的操作所占的比例不是很大，因此在文件系统卷 2 中，我们只配置了 1 个元数据服务器，同时配置了 m 个存储服务器。在每个客户端，用户可以根据需要 mount 任何一个文件系统卷，或同时 mount 两个文件系统卷。在每一个文件系统卷中，用户可以配置不同的参数以适应应用程序的负载特性，例如用户可以配置元数据分布粒度的参数，使元数据均匀地分布到多个元数据服务器上；用户也可以选择不同的分条单位大小来充分的利用多个存储服务器间的并行性。

(3) 高可用。不同的应用程序可以运行在不同的文件系统卷上，应用程序间互不影响，

当卷中的节点出现故障时不会影响其余卷上应用程序的运行；另外，文件系统卷中的存储服务器采用了 3.2.3 中介绍的网络分组结构，在一个存储分组中的服务器出现故障时其余分组仍可继续提供服务。

(4) 可管理性。DCFS 将整个机群系统视为一个管理域，由 DCFS 的配置管理器负责管理，包括文件系统的启动、停止和增加文件系统卷等；DCFS 可以将不同的文件系统卷分配给不同的应用程序使用，使这些应用程序不互相影响。

3.3 DCFS 网络存储分组

为了有效的管理机群文件系统的存储空间，多文件系统卷的结构将系统的存储空间划分为两个层次：文件系统卷和网络存储分组。在每个文件系统卷内划分为多个存储分组，每个存储分组可以管理多个存储设备，存储分组管理的设备可以分布在不同的节点上，文件按分条的形式存放在存储分组中的设备上。在本节中，首先介绍了网络存储分组的结构，然后对存储分组进行了数学分析，分析了影响读写性能的因素。

3.3.1 网络存储分组结构

图 3.5 表示了 DCFS 的一个卷存储分组的构成，从中可以看出，每个卷可以包含多个存储分组，DCFS 的存储分组配置单位是磁盘，而不是存储服务器，每个存储服务器都可以带多个磁盘，而每个分组(stripe)可以根据需要跨几个存储服务器。在创建文件时，元数据服务器根据轮转法（round-robin）为被创建文件选择一个分组和逻辑起始磁盘号，存储分组号和逻辑起始磁盘号决定了文件的数据从哪一个网络磁盘开始存放。DCFS 中的文件按分条单位大小（Stripe Unit）被划分为连续的数据块，DCFS 根据轮转法（round-robin）把数据块存储在每个磁盘上。

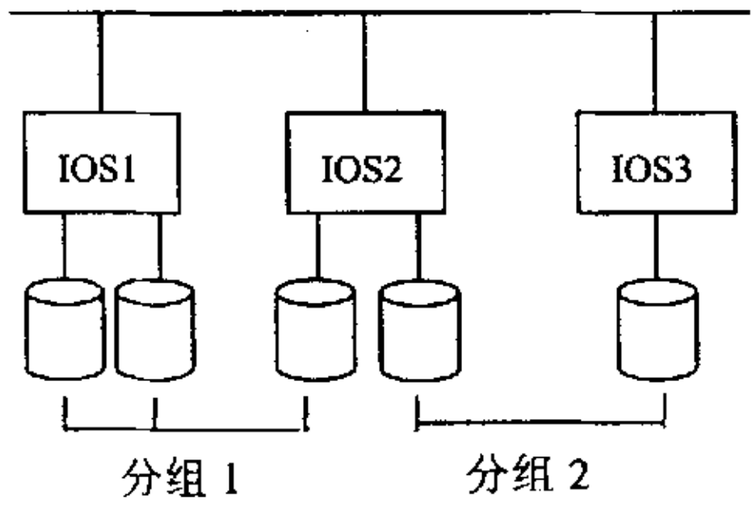


图 3.5 网络存储分组结构

利用多个磁盘的聚集性能来求得与网络性能之间的匹配是一种消除机群文件系统性能瓶颈，提高可扩展性的有效办法。磁盘分组(stripe groups)是由 P. Chen 等人为实现大规模的磁盘阵列提出来的[Che94]，我们也为 DCFS 实现了存储分组功能，并称之为网络存储分组，网络存储分组有如下的优点[Fen01]：

（1）存储分组能够使存储介质的利用更为均衡。实现网络磁盘分组功能后，一个分组中只包含系统的一部分磁盘，可以采用一定的措施保证各个分组之间工作负载的平衡，并且保证组内各磁盘的使用情况基本也是平衡的。

（2）存储分组有利于多个 I/O 操作的并发执行。在实现了存储分组的系统中，当不同

的文件落在不同的存储分组内时，多个客户对这些文件的 I/O 请求可由不同的存储分组来响应，同一分组内的多个存储服务器也提高了读写请求的并发性，具体分析见下一小节。

（3）存储分组能够使磁盘带宽与网络带宽相匹配，使网络和磁盘都能得到有效利用

（4）存储分组有利于改进可用性。当系统中部分存储器(或磁盘)崩溃后，只有包含这些出错的存储器(或磁盘)的存储分组不能提供 I/O 服务，其它的存储分组将不会受到影响。然而若没有实现存储分组功能，若系统中其中一个存储器(或磁盘)变为不可用时，某些文件服务将变为不可用。

3.3.2 DCFS 网络分组模型

DCFS 读写请求采用 fork/join 方式完成，也就是说读写请求首先被分解成多个独立的请求发送到存储分组中的每个服务器，然后等待所有的存储服务器完成操作。DCFS 读写流程模型如图 3.6 所示。

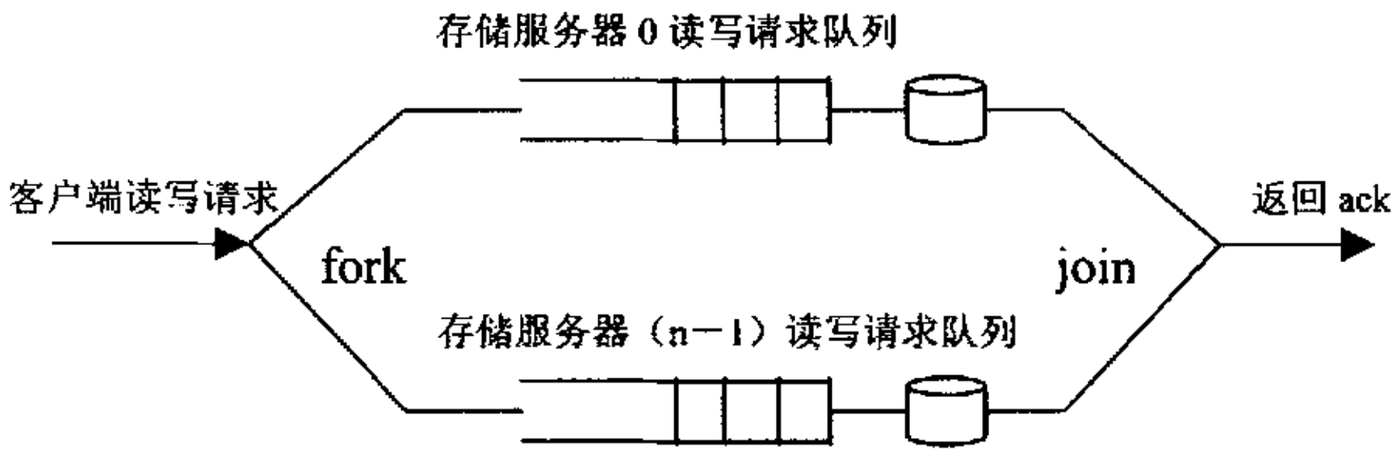


图 3.6 网络存储分组 fork/join 模型

图 3.7 给出了存储服务器和客户端间的数据通路，由于 DCFS 的读写请求在客户端是通过用户程序 Clerk 完成的（具体实现见第六章），图 3.7 中所示的数据通路指的是 clerk 和存储服务器上存储设备间的通路，而忽略了应用程序和 clerk 间通过核心的通信，另外，由于 DCFS 存储服务器是一个用户级程序，图 3.7 的服务器端数据通路中包括用户空间。数据通路包括如下三个部分：（1）从存储服务器端的存储设备到服务器网络 buffer，包括存储设备与 buffer cache 之间、buffer cache 与网络 buffer 之间的数据传输；（2）服务器网络 buffer 和客户端网络 buffer；（3）从客户端的网络 buffer 和客户端的用户空间。

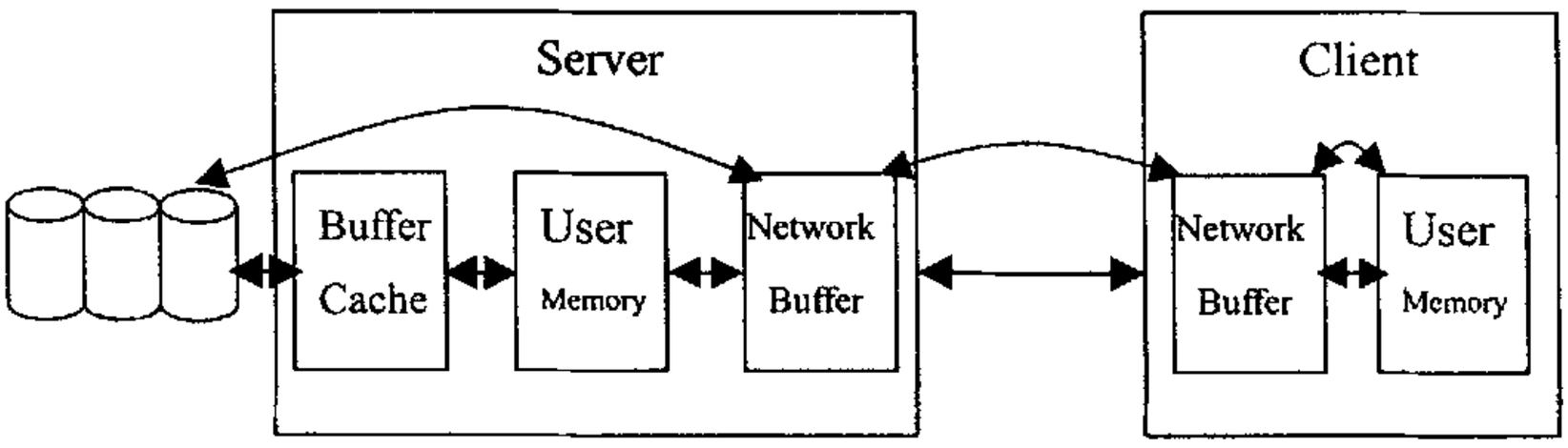


图 3.7 存储服务器与客户端数据通路

3.3.3 模型分析

设系统中有 N_{ios} 个存储服务器，分条的单位大小为 su，每个请求读写的数据字节数为 rs，该请求由 n_{ios} 个存储服务器完成。

$$n_{ios} = \begin{cases} \lceil rs/su \rceil & \lceil rs/su \rceil < N_{ios} \\ N_{ios} & \lceil rs/su \rceil \geq N_{ios} \end{cases}$$

假设客户端依次向网络分组中的存储服务器发送读写请求，启动顺序为第 1 个存储服务器，第 2 个存储服务器，…，第 n_{ios} 个存储服务器，每次读写操作时每个存储服务器读写的数据为 rs/n_{ios} 字节。图 3.8 所示的时间—空间图以写请求为例描述了三个存储服务器构成的网络分组的 stripe 工作特性，假设客户端和服务器网络的发送和接收缓存区足够大。其中 t_{c_s} 为客户端给每个存储服务器发送请求的启动时间间隔，等于数据从用户空间拷贝到核心空间的网络 buffer 的时间，t_o 表示客户端到服务器端的网络延迟。

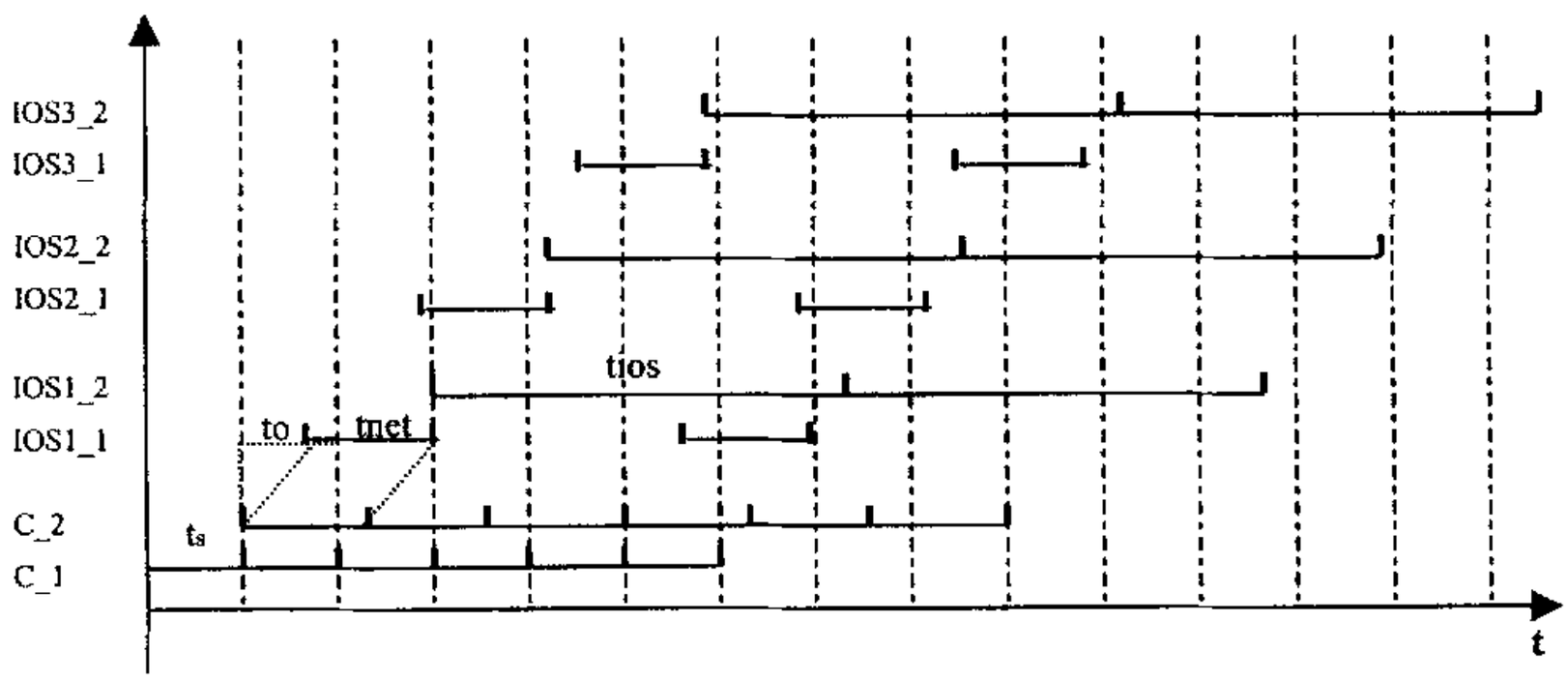


图 3.8 网络存储模型时空图

在下面的讨论中，$t_{ios\_s} = t_{ios\_proto} + t_{ios\_fs} + t_{ios\_mem}$ 表示服务器读写 su 字节数据的开销，其中，t_{ios_proto} 表示存储服务器读写请求的协议开销，t_{ios_fs} 表示读写文件的开销（DCFS 普通文件的数据以文件的形式存放在本地文件系统上），t_{ios_mem} 表示从网络 buffer 到存储服务器用户空间的开销。假设存储服务器上内存拷贝的带宽为 B_{mem}，磁盘块大小为 BS，寻道时间和传输速率分别为 t_{seek} 和 r_d。当数据可以直接读写 buffer cache 中而不需要读写磁盘时，$t_{ios\_fs} = t_{mem} = su/B_{mem}$，当数据需要读写磁盘时，$t_{ios\_fs} = t_{mem} + t_{disk} = \frac{su}{B_{mem}} + \frac{su}{BS}(t_{seek} + BS/r_d)$，为讨论方便，作者仅对需要读写磁盘的情况

进行论述。

由图 3.8 中给出了写操作的时间一空间图，并以写操作为例进行讨论。为简化讨论，设第 n_{ios} 个存储服务器最后完成 I/O 服务请求，则一个请求的完成时间为：

（1） 当 $t_{ios\_s} < t_o + n_{ios} \cdot su / B_{net}$ 时，网络成为瓶颈

则 $t = t_{c\_s} + rs / B_{net} + t_{ios\_s} + 2t_o$

$$t = 2t_o + t_{ios\_proto} + 2su/B_{mem} + rs / B_{net} + su \cdot (t_{seek} + BS / r_d) / BS \quad （3.1 式）$$

（2） 当 $t_{ios\_s} > t_o + n_{ios} \cdot su / B_{net}$ 时，服务器成为瓶颈

则当 n_{ios}<rs/su 时， $t = t_{c\_s} + n_{ios} \cdot su / B_{net} + \frac{rs}{n_{ios} \cdot su} \cdot t_{ios\_s} + 2t_o$

$$t = 2t_o + su / B_{mem} + \frac{n_{ios} \cdot su}{B_{net}} + \frac{rs}{n_{ios}.su}.(t_{ios\_proto} + su / B_{mem} + (su / BS) \cdot (t_{seek} + BS / r_d)) \quad （3.2 式）$$

当 n_{ios}>=rs/su 时， $t = 2t_o + t_{ios\_proto} + 2su / B_{mem} + rs / B_{net} + \frac{su}{BS} \cdot (t_{seek} + BS / r_d)$

3.3.4 讨论

（1） 由式 3.1，如网络为系统瓶颈，单个读写请求的完成时间与 n_{ios} 无关。由式 3.2，在存储服务器为系统瓶颈时，当 n_{ios}<rs/su 时，随着 n_{ios} 的增加，读写请求的响应时间减少，但当 n_{ios} 增加到一定数目时，文件系统的协议开销所占的比重增大，客户端开销的增加使读写响应时间增加；当 n_{ios}>=rs/su 时，单个读写请求的时间不变。

（2） 从上面的分析可知，存储服务器上的时间开销主要由网络开销和读写存储设备的开销构成，要减少存储服务器的开销可以从如下方面考虑：（a）多线程和异步 IO 的软件结构。多个线程在服务多个读写请求时，接收和发送数据的时间和读写存储设备的时间可以重叠，使存储服务器的服务时间减少，从而减少读写请求的时间开销。采用异步 I/O 可以使多个 I/O 请求并行执行，并且能够提高多个磁盘的 I/O 并行度[Wjy99]。（b）存储服务器数据缓存。（c）减少内存拷贝的次数，可以采用的方法包括采用用户级通信协议、存储服务器在核心实现以及采用特殊的系统调用（如 sendtofile）避免数据从网络 buffer 到用户程序再到 buffer cache 的拷贝。DCFS 为了提供存储服务器的处理能力采用了多线程和异步 IO 的软件结构并实现了服务器端缓存，设计了通信协议层以便利用 BCL[Ma01]等用户级通信协议，具体实现见第六章。

（3） 分条单位长度的选择也会对性能产生影响，下面进行讨论。

分条（Striping）提供了两种方式的并行性[Sch98]：一个是处理单个读写请求时的并行性，我们称为 inter-request 并行性，即每个读写请求由多个存储服务器同时服务，另一个是读写请求间的并行性，我们称为 intra-request 并行性，即多个存储服务器可以同时服务多个读写请求。较小的分条单位长度使一个文件读写请求由较多的存储服务器完成，单个

的读写请求的服务时间较少，但对于多个并发读写请求的情况，由于多个存储服务器同时服务于一个读写请求，系统的吞吐率反而较小；较大的分条单位长度使一个文件读写请求由较少的存储服务器完成，虽然对单个读写请求而言，其服务时间要增加，但一个文件系统卷的存储服务器却可以同时服务多个读写请求，系统的吞吐率要增加。

设有 N_c 个客户节点，每个客户节点上运行一个进程，每个进程每次发出一个读写请求，则每个存储服务器上的读写请求数为 $n_{ios} * N_c / N_{ios}$。因为当 su <rs/ N_{ios} 时，读写数据请求由所有存储服务器完成，为了讨论当分条单位长度不同时，读写请求由不同数量存储服务器并行处理的情况，我们仅针对 su >=rs/ N_{ios} 进行讨论。一个请求的完成时间为：

$t = t_{c\_s} + t_{ios} + 2t_o$，其中 t_{ios} 为每个读写请求在服务器上所消耗的时间，包括等待时间和服务器的服务时间，由于(3.1)和(3.2)式只给出了单个请求的响应时间，为了计算存储服务器在多个并发请求情况下的服务时间，作者引用文[Lee98]中的结果。Edward K.Lee 在[Lee98]中给出了计算磁盘阵列中每个设备响应时间的公式：

$$E(R) = \frac{E(s) \cdot (n_{disk} . N_r + N_r - n_{disk})}{N_{disk}} \qquad (3.3\ 式)$$

其中，E(R)、E(s)分别表示请求的响应时间和磁盘的服务时间，N_{disk} 和 n_{disk} 分别表示磁盘阵列中的磁盘个数和每个请求涉及的磁盘个数，N_r 表示并发的请求数。由该（3.3）式得，

$$t_{ios} = \frac{(t_{ios\_s} + t_{ios\_net}) * (N_c \cdot n_{ios} + N_{ios} - n_{ios})}{N_{ios}} \qquad （3.4\ 式）$$

$$t_{ios} = \frac{(t_{ios\_proto} + t_{ios\_fs} + t_{ios\_mem} + t_{ios\_net}) * (N_c \cdot n_{ios} + N_{ios} - n_{ios})}{N_{ios}} \qquad （3.5\ 式）$$

$$t_{ios} = \frac{(t_{ios\_proto} + \frac{rs}{n_{ios} \cdot B_{net}} + \frac{2rs}{n_{ios} \cdot B_{mem}} + \frac{rs}{n_{ios} \cdot BS}(t_{seek} + BS / r_d)) * (N_c \cdot n_{ios} + N_{ios} - n_{ios})}{N_{ios}} \qquad （3.6\ 式）$$

整理得，

$$t_{ios} = (t_{ios\_proto} + \frac{rs}{n_{ios} \cdot B_{net}} + \frac{2rs}{n_{ios} \cdot B_{mem}} + \frac{rs \cdot t_{seek}}{n_{ios} \cdot BS} + \frac{rs}{n_{ios} \cdot r_d}) * (\frac{N_c \cdot n_{ios}}{N_{ios}} - \frac{n_{ios}}{N_{ios}} + 1) \qquad （3.7\ 式）$$

将上式代入 $t = t_{c\_s} + t_{ios} + 2t_o$，并化简得，

$$t = a + b * n_{ios} + \frac{c}{n_{ios}} \qquad （3.8\ 式）$$

其中 $a = t_{c\_s} + 2t_o + t_{ios\_proto} + (\frac{rs}{B_{net}} + \frac{2rs}{B_{mem}} + \frac{rs \cdot t_{seek}}{BS} + \frac{rs}{r_d}) * (\frac{N_c - 1}{N_{ios}})$

$b = t_{ios\_proto} * \frac{N_c - 1}{N_{ios}}$，$c = (\frac{2rs}{B_{mem}} + \frac{rs \cdot t_{seek}}{BS} + \frac{rs}{r_d})$

由 3.8 式可知，当 $n_{ios}=\sqrt{\frac{c}{b}}$，即 $n_{ios}=\sqrt{\frac{c}{b}}=\sqrt{\frac{N_{ios}*b}{t_{ios\_proto}*(N_c-1)}}$ 时 t 最小，相应地，$su=rs\cdot\sqrt{\frac{t_{ios\_proto}*(N_c-1)}{N_{ios}*b}}$。可知在文件系统中并发进程数多时，即 N_c 较大时，最优 n_{ios} 的 su 较小，反之亦然。

3.4 元数据服务器结构

DCFS 的每个文件系统卷包含多个元数据服务器，元数据服务器间相互协调，维护该卷的元数据和服务元数据请求，在多元数据服务器的结构中，需要解决元数据分布和映射的问题，在本节中，作者首先分析了机群文件系统采用的元数据服务器的结构，然后给出了 DCFS 中多元数据服务器的组织方式和元数据分布映射策略。

3.4.1 多元数据服务器结构

机群/分布式文件系统为了提高元数据处理能力采用了独立的元数据服务器来处理元数据请求，现有文件系统的元数据服务器间采用的结构如图 3.9 所示。在图 3.9（a）中，文件系统中的元数据，包括 inode、目录、空闲 inode 的位图等都存放在共享的存储设备上[Tia03]，元数据和元数据服务器间的映射采用动态映射，每个元数据服务器负责处理映射到该服务器上的元数据的请求，称为集中存储结构；在图 3.9（b）中，文件系统中的元数据，包括 inode、目录、空闲 inode 的位图等分布到各个元数据服务器上，每个元数据服务器负责管理存储在该服务器上的元数据，称为分布存储结构。在第一种结构的元数据服务器组织中，元数据和服务器间的映射关系是动态，易于扩展元数据服务器，容易实现服务器间负载平衡，但由于元数据存储在共享设备上，在对空闲 inode 位图等共享数据结构进行操作时，多个元数据服务器间需要解决共享数据结构的互斥访问，并且由于元数据可以由任何一个服务器进行处理，使元数据和服务器之间的映射协议复杂，系统需要特定的服务器维护元数据的映射信息，也限制了这种结构的扩展性；在第二种结构的元数据组织中，元数据分布到各个服务器上，服务器间没有共享的数据结构，元数据的维护、映射相对简单，但在该结构中，元数据到服务器间的映射是静态的。

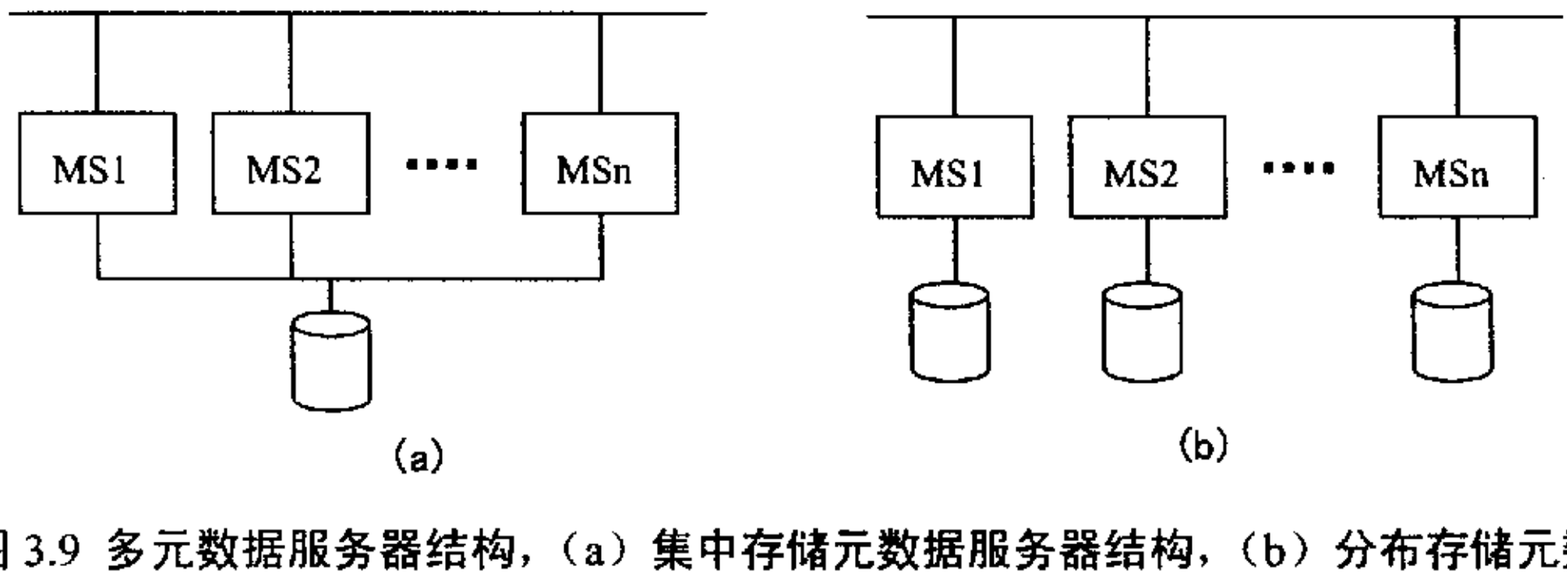


图 3.9 多元数据服务器结构，（a）集中存储元数据服务器结构，（b）分布存储元数据服务器结构

3.4.2 DCFS 元数据服务器结构

上面讨论了多元数据服务器的结构，由于元数据分布存储的结构中每个服务器维护不同的元数据，不需要实现对元数据的互斥访问，元数据操作易于实现，并且元数据映射简单、快速，因此，DCFS 采用了元数据分布存储的结构。图 3.10 中描述了 DCFS 多个元数据服务器间的组织(在 DCFS 中，我们将元数据服务器称为管理器)，在一个 DCFS 文件系统卷中，包括一个超级元数据服务器和若干个服务器，超级元数据服务器负责维护文件系统卷的信息，如超级块、根目录等，其他管理器则负责维护分配给本管理器的元数据。

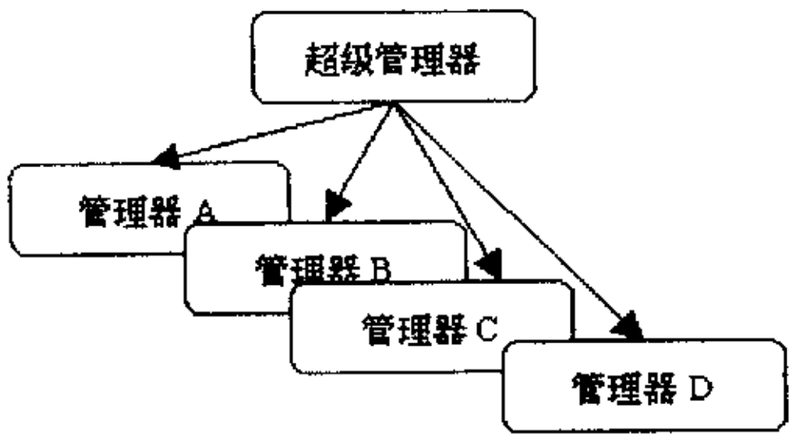


图 3.10 元数据服务器间的关系

针对分布存储结构的一些不足，我们在文[He03]中提出了两层元数据服务器的结构对 DCFS 元数据服务器的结构进行了改进。图 3.11 给出了该结构示意图。机群文件系统的元数据服务由前端的元数据缓存服务器组与后端的元数据服务器协作来提供。机群文件系统客户端的元数据请求全部由前端的元数据缓存服务器（图中的 MCS——Metadata Cache Server）完成，但是这些元数据缓存服务器本身不存储任何网络文件系统分区中的元数据，只是在内存中存在元数据副本。后端的元数据服务器在与其连接的物理存储设备中保存真正的网络文件系统元数据，同时也在内存中提供这些元数据的缓存以提高响应速度。在具体实现中可以将 MS（Metadata Server）与 MCS（Metadata Cache Server）实现为运行在相同服务器上的两个进程甚至单个进程内的两个线程。

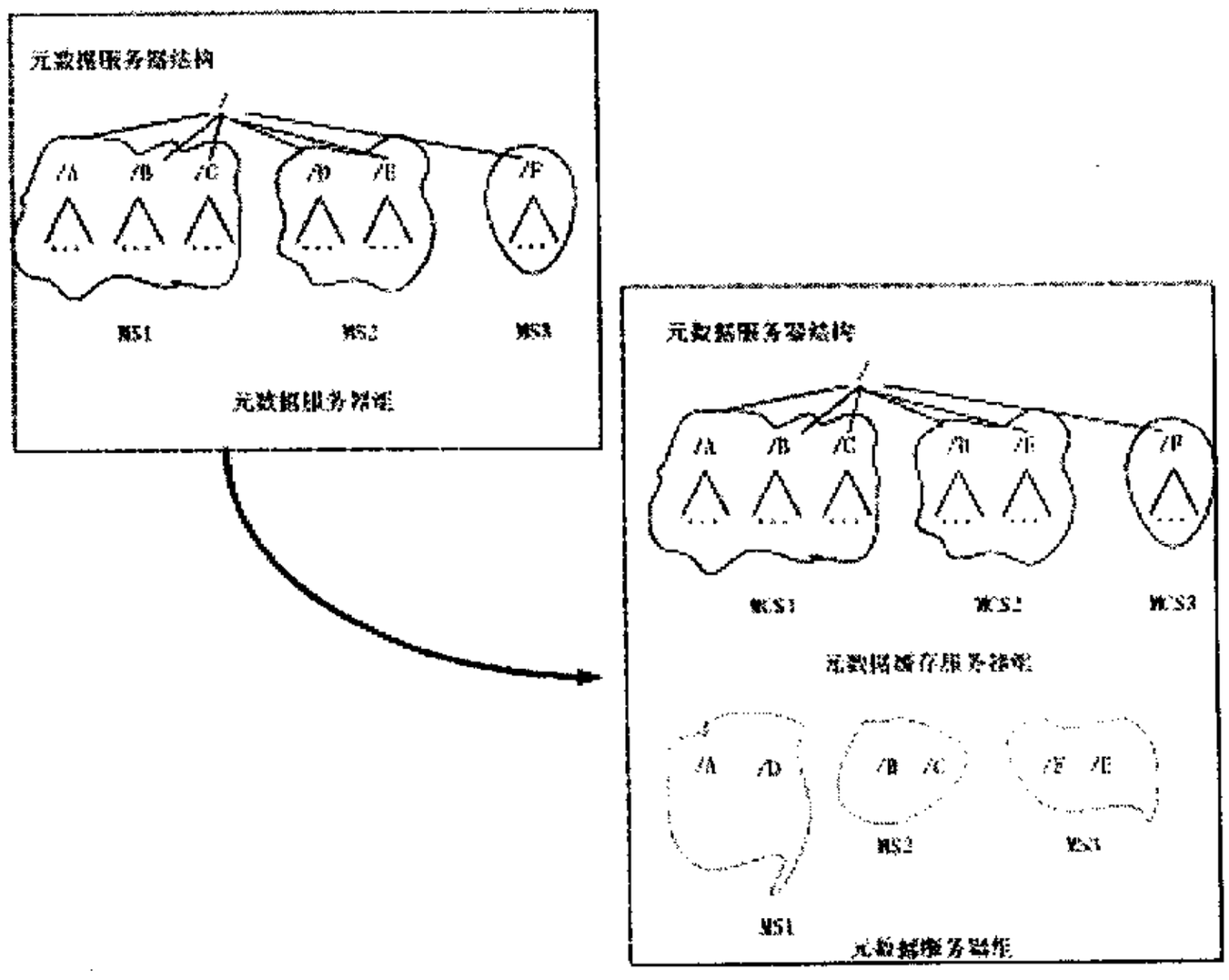


图 3.11 两层结构的元数据服务器[He03]

3.4.3 可调元数据分布策略

对于元数据分布存储结构的文件系统，需要采用合适的元数据分布策略使得元数据在服务器间均匀分布，避免某个服务器成为瓶颈，同时，文件系统协议的操作可能要涉及多个元数据服务器，合理的分布元数据可以减少网络传输的次数，提高文件系统元数据操作的性能。许多系统采用了细粒度的分布策略来管理元数据。Slice[And02]提出了“mkdir switching”的策略，在创建目录时按一定概率分布来选择服务器，另外，该系统还可以选择名字 HASH 的策略，按照文件名的 Hash 值选择服务器。Archipelago[Ji00]选择目录作为分布的粒度，根据目录路径名的 Hash 值选择服务器，在某些操作中会引起子树中的所有文件和目录重新分布。可以看到，在这些系统中，元数据分布的粒度在文件系统配置时已经确定，但在 DCFS 中，多个文件系统卷可以分配给不同的用户使用，整个文件系统统一的分布粒度并不适合所有的应用程序的负载特点，因此，我们提出了可调粒度的元数据分布策略，用户可以根据运行的应用程序目录树的结构为每个卷选择合适的粒度，使元数据在多个服务器间均匀分布。

DCFS 使用卷和粒度两层结构对元数据进行划分：（1）每个卷可以由不同的元数据服务器组进行管理，这样，卷之间的元数据服务器互不影响；（2）在每个卷内，把出目录构成的树按照如下定义的粒度将每个卷的名字空间划分为不重叠的子树，位于同一棵子树上的目录由同一个元数据服务器进行管理。

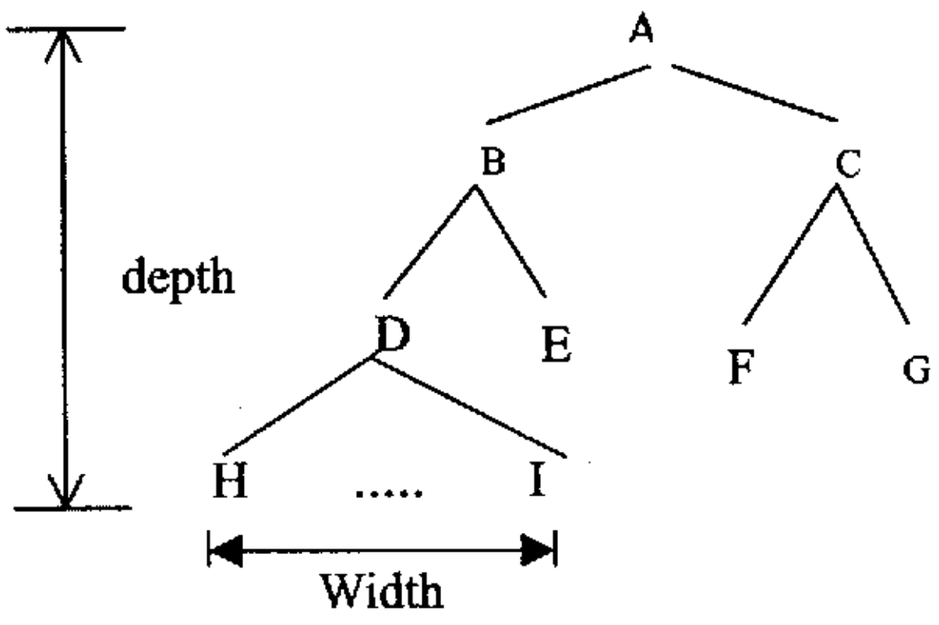


图 3.12 元数据分布粒度定义

如图 3.12 所示，作者定义了 DCFS 元数据分布的子树粒度，我们使用（*subtree_dep*，*subtree_width*）来描述粒度的大小，其中 *subtree_dep* 表示子树的深度，*subtree_width* 表示子树中每个目录的最大宽度。特别地，粒度（1，1）表示以目录为单位将元数据分配到服务器上，（∞，∞）表示以根目录下的整棵子树中所有目录由一个元数据服务器维护。图 3.13 中给出了以这两种粒度进行元数据分布的例子。

对于每个卷根目录下创建的目录，由于数量一般较少而且应用程序进行了归类，因此，对根目录下的子目录，DCFS 按轮转方法将子目录分配给元数据服务器进行管理。另外，考虑到文件系统中存在大目录，为了避免同一目录中所有的文件由一个元数据服务器管理而存在热点，因此，我们定义一个值 file_number，目录下的文件按 file_number 被分成多个组，每个组由一个元数据服务器进行管理。对于上面描述的目录粒度，当 file_number 取值为 1 时转化为文件粒度。

可调粒度元数据分布为自适应的元数据分布策略提供了手段。文[He02-3]中给出了

DCFS 基于逻辑环的负载平衡策略，DCFS 通过高可用协议的逻辑环传递各服务器的负载平衡参数。对于元数据分布，每个卷的元数据服务器定义两个参数 N_{meta} 和 N_{dmeta}，分别表示服务器上存储的该卷的元数据数量以及目录项和inode分布在不同服务器的元数据数量，当服务器上存储的元数据数量不平衡时可以降低超载服务器上元数据的分布粒度，而当服务器上 N_{dmeta} 所占的比重过大时可以增大服务器元数据分布的粒度。

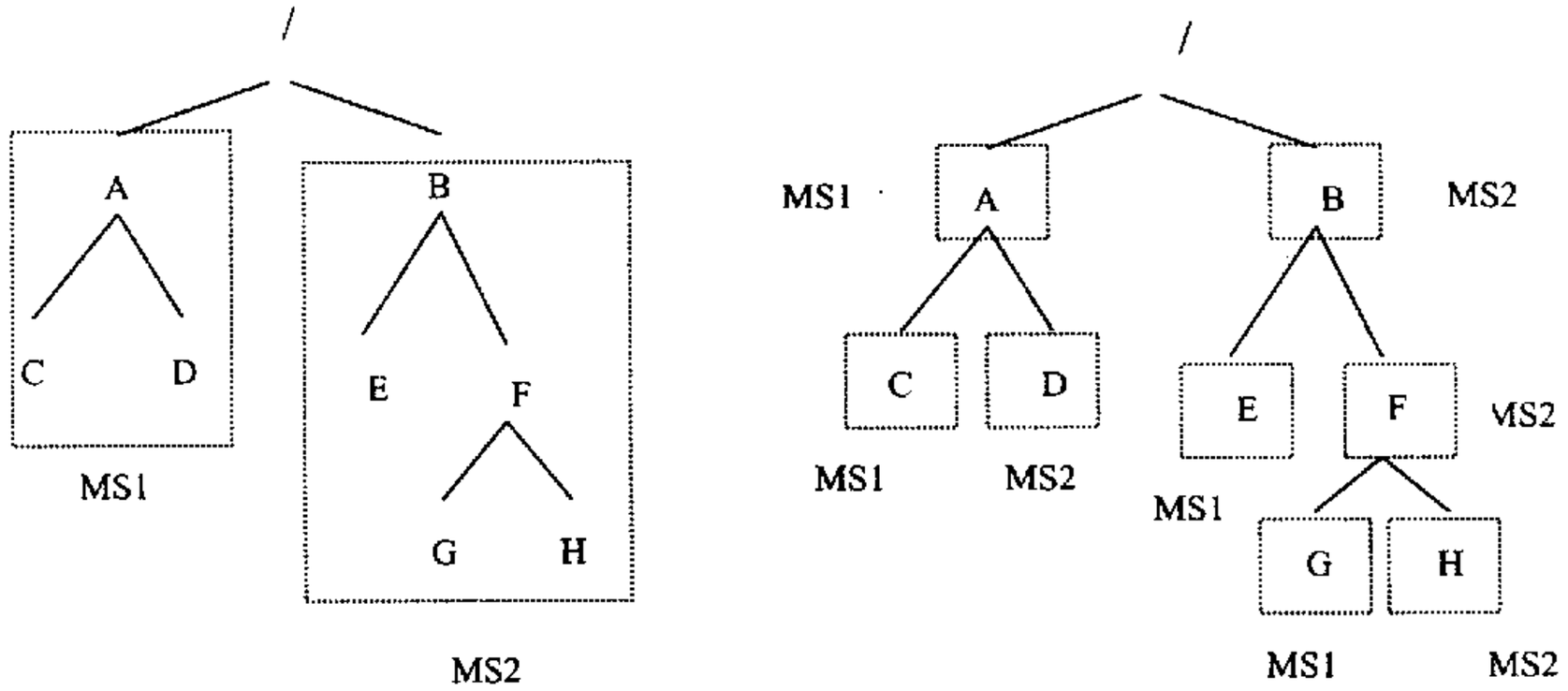


图 3.13 （a）子树粒度元数据分布；（b）目录粒度元数据分布

3.4.4 元数据到服务器的映射

元数据服务器维护了整个文件系统中一部分元数据，元数据到服务器的映射将元数据请求转发到正确的服务器上进行处理。DCFS 中每个文件用（volume, ino）的二元组标识，其中 volume 表示文件所在的卷号，ino 表示文件的索引节点号。图 3.14 给出了索引节点的结构。可以看到，通过把索引节点号空间的一部分分配给每个服务器，来达到将元数据存储、处理任务进行划分并分配给多个服务器。为了确定一个文件的元数据服务器，以（volume，ino）为输入，首先根据卷号得到该卷的元数据服务器映射表，由 ino 产生一个逻辑管理器号，并以它为索引从相应卷的元数据服务器映射表确定出哪一个服务器控制着有关该文件存储位置的元数据。系统中的每个节点都有一份元数据映射表的复制拷贝，它使得客户可以根据文件索引节点号直接同正确的服务器相联系。元数据服务器映射表的更新由 DCFS 中的管理节点进行处理，当该映射表被修改后由管理节点通知文件系统的所有节点。

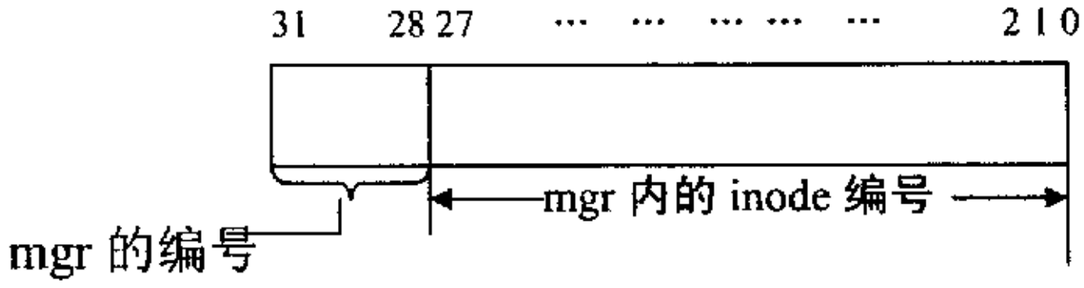


图 3.14 索引节点号结构

元数据服务器映射表提供了一层抽象，允许系统灵活地控制从逻辑（虚拟）元数据服务器到物理结点的映射。当系统中某个元数据服务器崩溃后，我们可以用新的物理结点去代替出现问题的服务器的结点，并利用新的服务器结点来处理发向映射到原崩溃结点的逻辑管理器的请求。

3.4.5 元数据分布策略试验及分析

在本小节中，我们将 DCFS 不同的卷分别配置成常用的文件粒度、目录粒度（这里的目录粒度指同一目录下的所有文件位于相同的服务器上）、子树粒度的三种粒度，在不同的负载下进行了比较，从试验可以看到，不同的负载需要合适选择合适的元数据分布粒度以获得较好的性能，而 DCFS 提供了可调粒度的分布策略使用户可以灵活的选择。以文件粒度分布元数据时，元数据可以均匀地分布到各个服务器上，但对一个文件而言，其父目录与其可能不在一个元数据服务器上，在完成一个元数据操作时需要涉及多个服务器，需要经过多次的网络传输才能完成操作请求；以子树为粒度分布元数据时，文件和父目录都在同一元数据服务器上，元数据操作请求通常只需一次网络传输就可以完成，但对于目录树分布不均匀的情况，即文件系统的文件和目录集中在某个子树下时，会造成元数据分布不均匀，使该元数据服务器成为瓶颈；而以目录粒度分布元数据时，其情况介于文件粒度和子树粒度之间。

在下面的试验中，DCFS 文件系统服务器组由 4 个元数据服务器和一个存储服务器构成，我们配置成三个 DCFS 文件系统卷，这三个文件系统卷分别配置成按文件粒度、目录粒度和子树粒度。试验所用节点配置与表 6.3 相同。在试验 1 中，每个客户端测试进程在根目录下创建一个深度为 6 的目录树，每个目录下包含 4 个子目录和 20 个文件，这样创建的文件系统目录树分布均匀；而在试验 2 中，每个客户端测试进程在同一个目录（非根目录）下创建一个深度为 6 的目录树，每个目录下包含 4 个子目录和 20 个文件，以模拟目录树不均匀的情况。

表 3.1 和 3.2 给出了试验 1 中三种粒度元数据操作性能比较，表的第一行表示客户节点个数。我们可以看到子树粒度的文件系统卷的性能要优于文件粒度和目录粒度的文件系统卷，其中创建的吞吐率平均高 12%，删除的吞吐率平均高 11%。表 3.3 给出了我们利用 DCFS 的 trace 收集工具分析得到的每个 DCFS 协议操作平均的网络请求次数，我们看到，对于只涉及一个服务器的操作，如 DCFS_OPEN 和 DCFS_STATFS,都只需一次网络传输，而对于需要多个元数据服务器配合的协议操作，例如 DCFS_LOOKUP，子树粒度只需一次网络传输，而目录粒度和文件粒度平均需要 1.497 和 1.552 次网络传输。

表 3.1 试验 1 创建吞吐率比较

	1	*4*	*8*	*12*	*16*	*20*	*22*
子树粒度	149.0	578.5	1163.4	1710.5	2271.5	2777.6	3019.2
目录粒度	133.3	518.3	1033.0	1527.4	2027.4	2418.8	2639.8
文件粒度	133.8	516.0	1022.5	1487.6	1961.2	—	2571.8

表 3.2 试验 1 删除吞吐率比较

	1	*4*	*8*	*12*	*16*	*20*	*22*
子树粒度	158.1	607.0	1224.3	1797.2	2388.5	2931.1	3191.5
目录粒度	142.3	548.2	1093.6	1622.7	2123.0	2610.0	2832.8
文件粒度	136.9	529.6	1061.4	1562.5	2031.8	2505.0	2696.8

表 3.3 不同分布粒度下文件协议操作数的比较

操作类型	子树粒度	目录粒度	文件粒度
DCFS_LOOKUP	1.000	1.497	1.552

DCFS_OPEN	1.000	1.000	1.000
DCFS_CREATE	1.000	1.125	1.750
DCFS_CLOSE	1.000	1.000	1.000
DCFS_REMOVE	1.000	1.125	1.750
DCFS_STATFS	1.000	1.000	1.000

表 3.4 和 3.5 给出了试验 2 的结果，可以看到随着客户端测试进程的增加，由于子树粒度的文件系统卷把元数据分布到一个元数据服务器上，负载增重，其性能比目录粒度和文件粒度的文件系统卷性能要差 40%。

表 3.4 试验 2 创建吞吐率比较

	1	*4*	*8*	*12*	*16*	*20*	*22*
子树粒度	131.0	516.1	965.6	1387.7	1581.0	1464.7	1447.4
目录粒度	115.8	466.6	925.2	1368.6	1797.0	2211.3	2393.0
文件粒度	112.4	455.8	908.7	1328.8	1742.6	2138.0	2298.4

表 3.5 试验 2 删除吞吐率比较

	1	*4*	*8*	*12*	*16*	*20*	*22*
子树粒度	138.2	547.4	1019.8	1359.3	1168.3	1093.2	946.3
目录粒度	123.4	497.0	986.9	1453.6	1901.0	2349.4	2544.2
文件粒度	117.8	472.2	945.2	1385.2	1803.1	2199.2	2369.4

从上面的分析可以看到，对于不同类型的负载，文件系统元数据服务器应选择合适的分布粒度，元数据分布粒度的选择主要考虑两个因素：（1）负载平衡，即元数据在多个服务器间分布是否均匀；（2）元数据操作的开销，因为元数据分布会造成一个元数据操作需要多个元数据服务器间的协作，这增加了网络开销和服务器的开销。因此，在选择元数据分布粒度时需要权衡这两个因素。例如，对于应用程序生成的目录树均匀的情况，子树粒度不仅可以使元数据在服务器间均匀分布，还由于该粒度使父子目录都位于相同元数据服务器上，大多数的操作只须一个元数据服务器，使元数据操作的开销减少，而对于目录树不均匀的情况，目录粒度或其他细粒度的分布粒度更为合适。对于 DCFS 的多文件系统卷的结构，单一的元数据分布粒度不能满足多种类型应用程序负载的要求，我们在本章中给出的可调粒度元数据可以为每个卷分别选择合适的分布粒度，并为自适应元数据分布策略提供了手段。

3.5 小结

本章首先分析了机群文件系统的体系结构，提出并在 DCFS 文件系统中实现了多文件系统卷的结构，该结构可以有效管理机群文件系统的存储空间和名字空间，根据应用程序的特点灵活配置，具有良好的可扩展性。在本章中，作者还对多文件系统卷体系结构中的两个重要组成部分：存储服务器组和元数据服务器组的组织进行了描述。在 3.2.3 节中，对存储服务器组采用的网络存储分组进行了讨论和分析，分析了影响文件读写性能的因素，在 3.2.4 中，对文件系统卷中元数据服务器的结构、元数据分布与映射策略进行了讨论，提出了可调粒度的元数据策略使用户可以根据应用程序的特点选择合适的分布粒度。

第四章 元数据服务器目录操作研究

文件系统中存在着大量的目录操作，如创建、删除以及查找文件等，优化目录操作可以有效地提高文件系统元数据操作的性能。目录缓存是优化目录操作的有效途径，由于目录结构的特点，目录的访问模式体现出了与普通文件不同特点，文件系统可以利用这些特点设计合理的目录缓存算法。另外，许多机群/分布式文件系统为了提高元数据处理能力采用了元数据服务器，元数据服务器和客户端的目录缓存构成了多级的缓存结构，目录操作经过客户端缓存的“过滤”处理在元数据服务器表现出了与客户端不同的访问特性，在本章中，作者首先对元数据服务器上目录缓存的访问模式的特点进行了研究，然后描述了目录缓存管理的方法。由于传统的线性目录组织方法在大目录时目录项的查找、添加等操作时间开销很大，在本章 4.2 节中，作者介绍了与本研究小组成员提出的改进的动态 Hash 目录组织算法 LMEH[Tan03]。

4.1 多级目录缓存管理

文件系统使用目录文件存放该目录中每一个文件和子目录的目录项结构，通常目录项的结构包括如下的字段：(1)下一个有效目录项在目录文件中的真实偏移量（相对于目录文件起始位置）；(2)本目录项所对应文件的索引节点号；(3)本目录项的长度（以字节为单位）；(4)文件名长度；(5)本目录项所对应文件的名字，在 UNIX 系统中，文件名是一个长度不超过 256 的字符串。图 4.1 给出了一个目录文件的示意图。文件系统通常将目录文件以块进行划分，并以块为单位增加或缩短目录文件的长度。

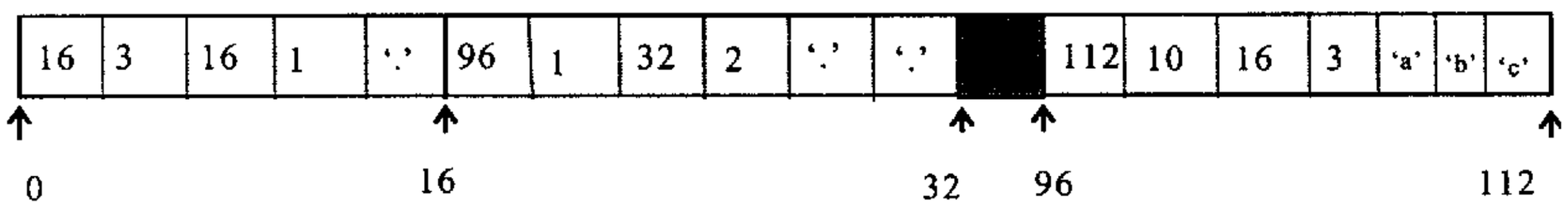


图 4.1 目录文件的组织结构

与目录结构相对应，文件系统系统中维护两类缓存：目录项（dentry）缓存和目录文件块缓存。文件系统中的两类操作涉及到目录缓存：（1）名字查找（LOOKUP）、添加和删除操作，这类操作需要操纵目录项（dentry）；（2）读目录操作（READDIR），该操作需要读目录文件。第一类操作可能需要操纵这两类缓存，例如，在 LOOKUP 中，当该操作在目录项缓存中没有查找到相应的目录项时，需要从目录文件中读入目录块，然后进行查找。在本文中，作者将目录项（dentry）缓存和目录文件数据缓存分别称其为 LOOKUP 目录缓存和 READDIR 缓存。

在本节中，我们首先总结了相关的研究，比较了两种元数据服务器目录缓存的结构，然后针对所选择的结构讨论了多级目录缓存结构中元数据服务器上的目录项的访问特性以及不同目录缓存替换算法的比较，然后根据上述结论给出了元数据服务器目录缓存管理

的方法。

4.1.1 相关研究

4.1.1.1 缓存替换算法

在本章中，我们将对元数据服务器上的目录缓存的访问特性进行研究并比较了几种替换算法。如下给出了在各种缓存结构中常用的算法：

- ✧ 近期最少使用(Least Recently Used，LRU)算法：是选择近期最少访问的行作为被替换的项。LRU 算法是根据数据使用的“历史”情况来预估未来的数据使用情况，认为当前最少使用的数据，在将来可能最少被访问。正如下面小节中指出的，在层次结构的缓存系统中，LRU 算法并不一定有很好的性能。
- ✧ 使用频率最少（Least Frequently Used，LFU）算法。这是另一个经典缓存替换算法，在需要替换缓存块以便“腾出”空间时，LFU 算法选择使用频率最少的缓存块。该算法基于如下的假设：使用频率高的缓存块比使用频率低的缓存块更可能在未来被访问，因此，LFU 用使用频率作为对未来访问情况的估计。LFU 算法在一些情况下性能较差，其中一个问题是某些缓存块在一段时间内被频繁访问，使得该缓存块的使用频率计数很高，这样即使该缓存块不再被访问也不会被替换。
- ✧ 基于频率的替换算法（Frequency Based Replacement，FBR）。FBR 算法试图结合 LRU 和 LFU 的优点，它维护了一条 LRU 的链表，但以使用频率作为选择替换的缓存块的依据。FBR 将 LRU 链表划分为三个区域：“新”、“旧”和处于中间的区域，FBR 使用两个参数来划分这三个区域：F_{new} 和 F_{old}，分别标识“新”区和“旧”区所占的比例。如果查找的缓存块位于“新”区中，其使用频率计数并不增加。

4.1.1.2 层次结构缓存管理的研究

客户/服务器结构的分布式系统为了提高系统的性能，在客户端和服务器中引入了缓存，形成了客户/服务器的多级缓存结构。对于多级的缓存结构，是否每一级的缓存都表现出相同的访问特性呢？许多研究者[Wil93][Fro96][Zho01][Ree96]对这个问题进行了研究，并根据不同级别的缓存表现的访问特性提出了有效的缓存管理算法。

D. L. Willick[Wil93]等人收集了网络文件服务器的多种 trace，利用这些 trace 对客户/服务器的多级缓存结构进行了仿真，并且比较了多种服务器端缓存的替换算法，他们发现，文件服务器表现出了和客户端不同的访问特性，随着客户端缓存规模的扩大，服务器端的时间局部性的特征减弱，LFU 和 FBR 等基于访问频率（Frequency）的缓存替换算法比 LRU 算法更有效。K. W. Froese 等[For96]研究了在分布式文件系统中客户端缓存对服务器上的负载的影响，他们的结论与[Wil93]中的结论相同，提出在服务器缓存中应当采用基于访问频率的替换算法。Zhou 等[Zho01]对文件服务器、数据库后端的存储系统的 trace 进行了分析，他们认为，服务器上的第二级缓存由于第一级缓存对访问的“过滤”作用表现出了与第一级缓存不同的特性，时间局部性减弱，提出了服务器的缓存算法需要满足的性质，并提出了 MQ（Multi-Queue）的服务器缓存替换算法。Benjamin Reed 等[Ree96]对运行在 SunOS 和 SGI IRIX 上的 NFS 服务器的 trace 进行了分析，与上述 trace 不同的是，Benjamin Reed 等收集的 trace 中包含了文件系统的元数据信息，他们的结论认为，由于元数据访问特性的

影响，文件服务器缓存表现出较强的时间局部性，LRU 缓存替换算法相比 LFU 等基于访问频率的算法仍有效。

一些研究者对目录缓存进行了研究[She86][Flo89][Shi92]。A. B. Sheltzer 等[She86]对 LOCUS 分布式操作系统中目录缓存的特性进行了研究，而 R. A. Floyd 等[Flo89]研究了 4.2BSD UNIX 系统上收集到的文件系统目录访问特性的 trace，这些研究表明，目录缓存表现出了时间局部性，在客户端节点目录缓存采用 LRU 替换算法的缓存命中率很高。K. W. Shirriff 等[Shi92]在对 Sprite 系统的研究中指出，目录缓存在客户端表现出了很强的时间局部性，在客户端缓存数量很小时（20-40KB）也可以获得较高的缓存命中率（大于 80%）。在该文中对基于目录项的缓存算法和基于整个目录的缓存算法进行了比较，认为基于整个目录的缓存算法可以获得更高的缓存命中率。

本章中的研究与前面列举的相关研究有如下不同：（1）本章针对元数据服务器目录缓存的访问特性和管理算法进行研究，而不是在[Wil93][Fro96][Zho01][Ree96]中研究的数据缓存；（2）[She86][Flo89][Shi92]的研究中主要对客户端节点的缓存进行了分析，并且在这些研究中的文件服务器同时处理元数据和数据的服务请求，服务器上的 READDIR 缓存（目录块缓存）与数据块缓存被统一管理，而在本章中，我们的研究基于 DCFS 文件系统，着重于研究元数据服务器上的目录缓存，DCFS 文件系统采用了存储服务器和元数据服务器分离的技术，服务器上不同层次的目录缓存可以根据自身的特性采用适当的管理算法。

4.1.2 目录缓存试验

4.1.2.1 目录缓存结构

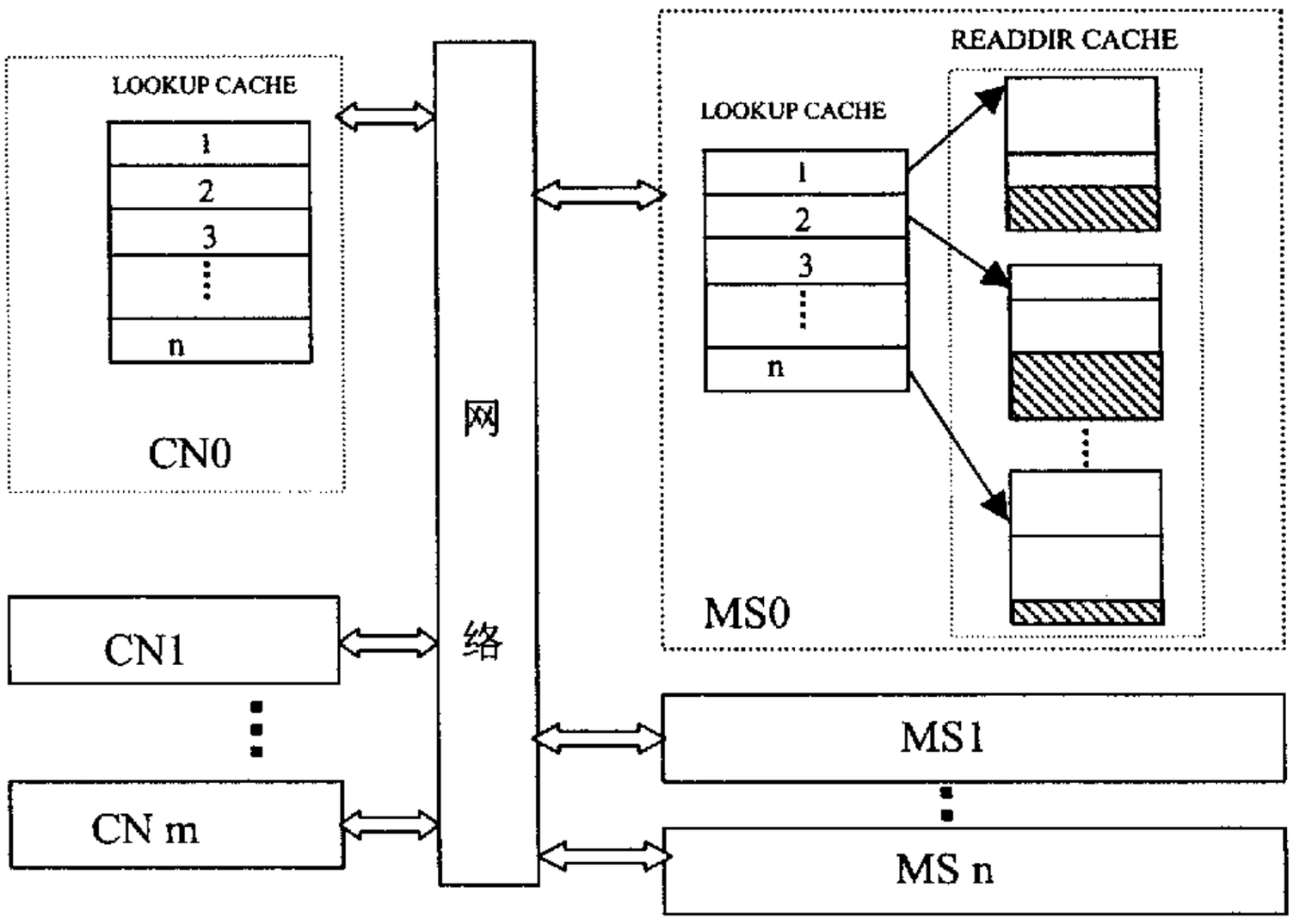


图 4.2 客户端和元数据服务器构成的多级目录缓存结构

图 4.2 给出了 DCFS 目录缓存的结构，CN 表示客户节点，MS 表示元数据服务器。目录缓存由两部分组成：客户端目录缓存和元数据服务器目录缓存。元数据服务器目录缓存包括两个层次：LOOKUP 缓存和 READDIR 缓存。图中的客户端缓存结构只包括 LOOKUP 缓存，这是因为 DCFS 采用了与 NFS 和 Coda 等文件系统相同的策略，对于 LOOKUP 操

作，如果目录项不在客户端 LOOKUP 目录缓存中，发送到服务器完成，MKDIR 等创建操作由服务器完成，而对于 READDIR 操作，DCFS 在当前的版本中在客户端没有实现 UNIX 语义的 READDIR 缓存，为了客户端正确读取的目录块，READDIR 操作需要由服务器完成，因此，DCFS 没有在客户端实现 READDIR 缓存。在本章中，我们针对图中的结构进行讨论。

元数据服务器目录缓存的管理可以采取两种方法：（1）只有 READDIR 缓存中维护着目录项，LOOKUP 缓存中的项是指向目录项在 READDIR 缓存中位置的索引；（2）LOOKUP 缓存的项为目录项，LOOKUP 缓存和 READDIR 缓存之间相互独立。在方法（1）中，由于目录项只是由 READDIR 缓存维护，目录操作相对简单，节省空间；而在方法（2）中，元数据服务器需要保证目录项在 LOOKUP 和 READDIR 两类缓存中一致，目录操作较（1）中的方法复杂。

4.1.2.2 试验平台与负载

在本章中，作者使用 FSbench[Lu03]作为试验的负载生成的工具。FSbench 是一种基于 UNIX 平台的机群文件系统基准程序，另外，它扩展 SPEC SFS[SFS01]的负载测试方法，可以用来测试多种机群文件系统，提供了多种测试共享文件的模式，以测试共享文件的并行存取性能。在本章试验中，利用 FSbench 的综合负载性能测试的功能，在机群文件系统的客户端按一定操作比例产生文件系统的负载，测试文件系统的响应时间和吞吐率。

R. A. Floyd 等在[Flo89][Shi92]中指出文件系统客户端具有很强的目录项访问的时间局部性特征，我们利用文[Thi92]中给出的基于堆栈的模型（The Stack Model）来模拟具有时间局部性的负载。设文件系统的文件集为 W＝{f1，f2，f3，…，fn}，LRU 栈用于记录文件系统文件集的访问情况，LRU 栈足够大可以容纳整个文件集，LRU 栈按照 LRU 缓存替换算法进行操作。当栈中位于位置 i 的项被访问时，该项成为最近被使用的项（Most Recently Used，MRU）并被移动到栈顶，而原来位于 1 至 i-1 的项依次后移至第 2 和第 i 项。

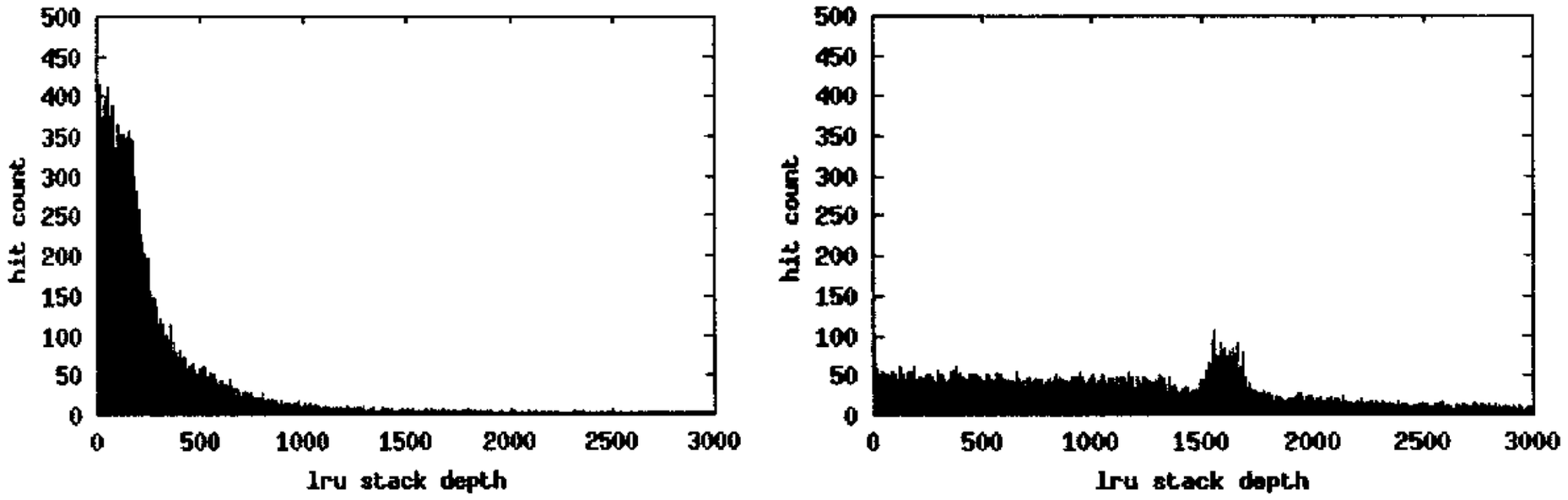
定义命中索引（Hit Index）为 LRU 栈中的位置，MRU 项在栈中的位置定义为 1。则由[Thi92]，命中索引的 marginal probability 可以表示为：

$$\Pr[hit\_index > x] = \begin{cases} \dfrac{(A^{\theta}/\theta).(C_c^{-\theta} + (1-x)C_c^{1-\theta})}{1-K}, x < C_c \\ \dfrac{(A^{\theta}/\theta).x^{1-\theta} - K}{1-K}, C_c \le x \le N_w \\ 0, x > N_w \end{cases} \quad \text{（公式 4.1）}$$

其中，N_w 表示文件系统文件集的大小，A 和 θ 为常数，分别为工作集的大小和局部性的参数，θ 越大，局部性表现的越明显。K 为常数，并且 $K = (A^{\theta}/\theta).(N_w^{1-\theta})$。由公式 4.1，产生合成负载的步骤为：（1）LRU 栈初始化为每个项存放一个唯一的文件集的索引；（2）产生随机数，并由公式 4.1 计算出 LRU 栈的位置，取出存放于该位置中的文件标识，然后根据 LRU 替换算法更新 LRU 栈。

本章的试验平台由四个节点构成，该平台上运行 DCFS 文件系统，DCFS 文件系统配置成 1 个元数据服务器、1 个存储服务器以及 2 个客户端节点。N_w 根据 SPEC SFS[SFS01]

的参数值进行设置，每种文件系统相关系统调用的比例也根据 SFS 进行设置，具体取值见 6.3 小节。A 取值为 20.0，θ 取值为 1.75 和 2.50，分别表示局部性较弱和较强的两种情况，图 4.3 给出了 θ 分别取值为 1.75、2.50 时产生的负载的 LRU 堆栈深度直方图。在下面的试验中，客户端节点目录缓存采用 LRU 缓存替换算法。



（a） （b）

图 4.3 客户端负载 LRU 和 LFU 堆栈深度直方图：（a）θ=2.50；（b）θ=1.75

4.1.2.3 目录缓存管理方法选择

表 4.1 给出了 4.1.2.1 节中介绍的两种目录缓存结构缓存命中率的比较，其中 θ 取值为 2.5，表的第一列表示客户端 LOOKUP 缓存的大小，而第二列表示 READDIR 缓存的大小。目录缓存命中率定义：$H = N_H / N_L$，其中 N_H 表示在目录缓存（包括 LOOKUP 缓存和 READDIR 缓存）中查找到指定目录项的 LOOKUP 操作次数，N_L 表示 LOOKUP 操作的总次数。由于方法（1）中 LOOKUP 缓存作为目录项的索引，因此 LOOKUP 缓存的大小取值为 READDIR 缓存所包含的最大目录项个数，方法（1）和（2）的 LOOKUP 缓存和 READDIR 缓存采用相同的缓存替换算法，表中给出的缓存命中率为这两种方法采用命中率最高的替换算法所得的结果。可以看到，方法（2）的缓存命中率高于方法（1），这是因为方法(1)中 LOOKUP 缓存的项需要指向目录项在 READDIR 缓存中的位置，READDIR 缓存的替换会引起相应的 LOOKUP 缓存中项的替换,这导致 LOOKUP 中一些最近被访问的项可能也被替换。因此，作者在元数据服务器的目录缓存结构中选择了方法（2），下面的试验中讨论了方法（2）中 LOOKUP 和 READDIR 两类缓存的访问特性。

表 4.1 LOOKUP 缓存命中率比较（θ=2.50）

CC size	LC size	方法 1	方法 2
0	256	68.8	81.3
	512	70.3	90.8
	1024	74.3	96.2
	2048	79.7	98.4
64	256	2.8	27.9
	512	11.3	64.6
	1024	20.1	85.4
	2048	36.1	94.7
256	256	3.2	4.1
	512	7.0	22.0
	1024	16.2	61.5
	2048	35.3	86.2

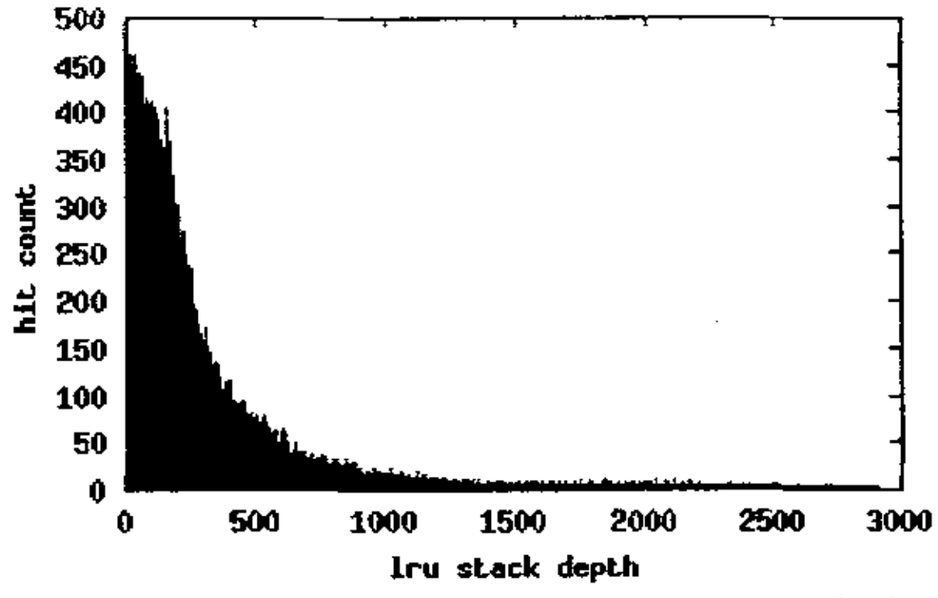
4.1.2.4 目录缓存访问特性分析

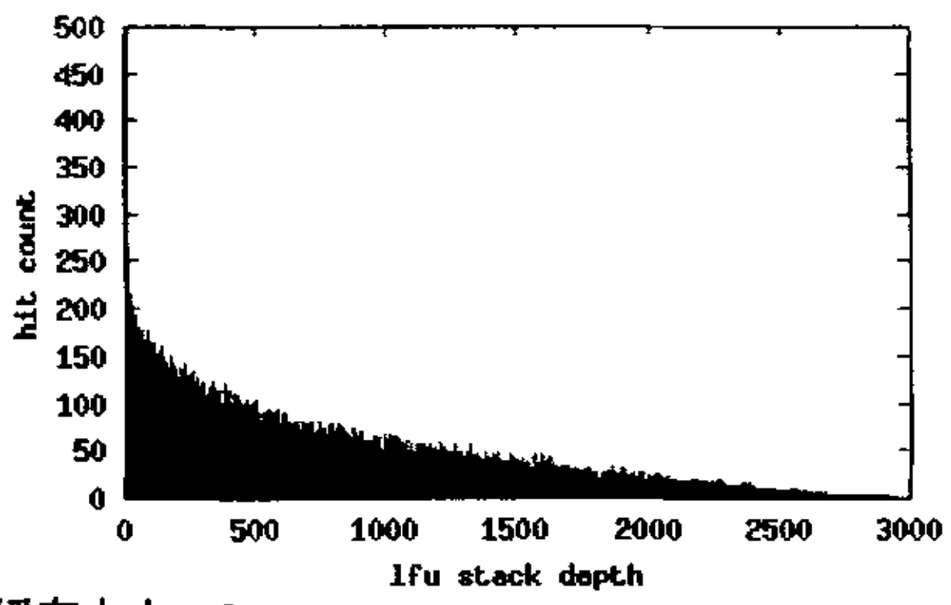
表 4.2 给出了客户端目录缓存容量变化时元数据服务器 LOOKUP 缓存的替换算法命中率的比较，其中 θ 取值为 2.50，元数据服务器 READDIR 缓存容量设置为足够大。目录缓存命中率定义为：$H = N_{HL} / N_L$，其中 N_{HL} 表示在 LOOKUP 缓存中查找到目录项的次数，N_L 表示 LOOKUP 操作总次数。为了更好地分析元数据服务器上目录项访问的特性，对每个客户端目录缓存容量的取值，我们在元数据服务器上收集了目录项访问的 trace，并分别绘制了 LRU 栈深度和 LFU 栈深度直方图，LRU 栈和 LFU 栈的容量设置为无限大，对应于横轴上取值为 depth 的点，表示在深度为 depth 的栈的位置上缓存命中的次数。可以看到，随着客户端目录缓存容量的增加，元数据服务器目录缓存的命中率逐渐下降。当客户端目录缓存容量为 0 时，元数据服务器的目录缓存命中率都较高，这是因为：（1）客户端合成的目录项访问的负载具有较强的局部性特征；（2）由于客户端目录缓存容量为 0，所有的名字解析的操作都需要由元数据服务器完成，这样，每个文件的路径上所有分量的查找由元数据服务器完成，元数据服务器上对这些共同的路径名分量的查找也使目录项访问的局部性增强，栈深度小于 10 的位置上缓存命中的次数接近 50%。从表 4.1 中缓存算法的比较可以看到，LRU 算法的命中率高于 LFU 和 FBR，图 4.4（a）至（c）反映目录项访问的 LRU 和 LFU 栈直方图也证实了这一结论。首先，随着客户端目录缓存容量的增加，由于客户端目录缓存的“过滤”作用，元数据服务器上目录项访问的 LRU 和 LFU 特征也随着减弱；其次，LRU 的访问特性较 LFU 的访问特性明显，从 LRU 栈深度直方图可以看到，多数的缓存命中在一段区间内，然后随着栈深度的增加命中次数显著减少，LRU 直方图的形状较陡，而 LFU 栈深度直方图则相对平滑；另外，对于 LRU 栈深度直方图，我们发现当客户端目录缓存大小不为 0 时，对于栈深度小于客户端缓存容量的情况，命中次数很小，这时因为如果对目录项的访问间隔小于客户端缓存容量，该目录项存在于客户端缓存中。

表 4.3 给出了客户端目录缓存容量取值为 64 时，元数据服务器 READDIR 缓存几种算法命中率的比较，θ 取值为 2.50，LOOKUP 缓存均采用 LRU 缓存替换算法。READDIR 缓存命中率定义为 $H = N_{HR} / N_R$，其中 N_{HR} 表示当 LOOKUP 缓存不命中时在 READDIR 缓存查找到目录项的次数，而 N_R 表示当 LOOKUP 缓存不命中时查找 READDIR 缓存的总次数。同样的，我们在元数据服务器收集了访问 READDIR 缓存的 trace 并分别绘制了 LRU 和 LFU 栈的直方图。从表 4.2 和图 4.5（a）至（c）可以看到：（1）随着 LOOKUP 缓存容量的增大，READDIR 缓存命中率变化与 LOOKUP 缓存不同，变化幅度相对较小，没有明显的下降，这是因为服务器端的 READDIR 缓存同时服务 LOOKUP 和 READDIR 请求，而客户端的 READDIR 请求不经“过滤”直接发送到服务器，服务器 READDIR 缓存受 READDIR 请求序列的影响使其命中率变化不大；（2）LFU 缓存替换算法的命中率高于 LRU 和 FBR，这可以从图 4.5 的 LRU 和 LFU 栈深度直方图中反映出来，相比较 LRU 的直方图，LFU 直方图的缓存命中位置集中于栈深度较小的区域，随着栈深度的增加命中次数显著减少；（3）READDIR 缓存访问 trace 的 LRU 栈深度直方图与 LOOKUP 的 LRU 栈深度直方图的表现不同，在深度值小时命中次数很高，这是因为一个 READDIR 缓存块中包含多个目录项，这些目录项的查找均指向相同的 READDIR 缓存块；（4）READDIR 缓存容量设置为远大于 LOOKUP 缓存时才能获得较高的缓存命中率。

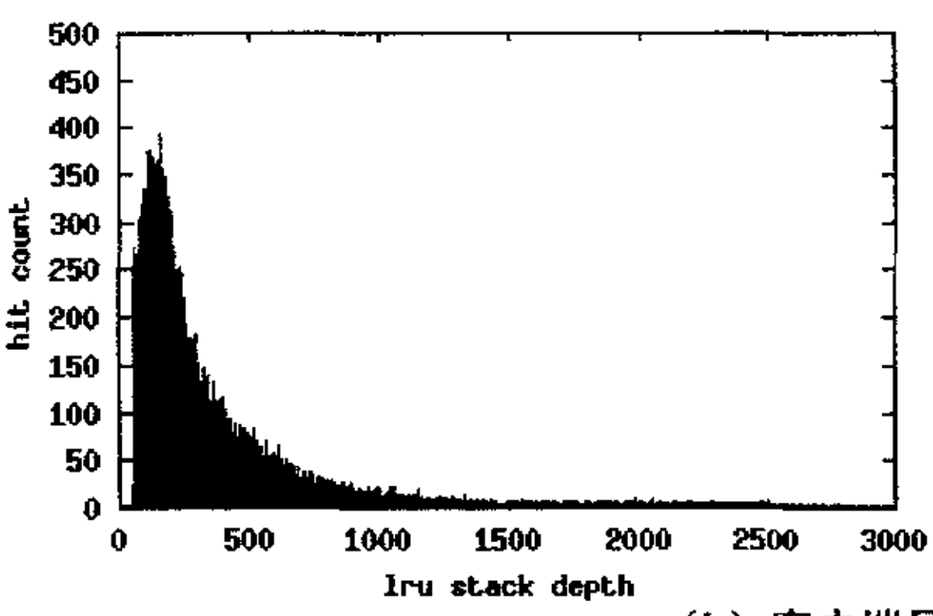
表 4.2 LOOKUP 缓存命中率比较（θ=2.50）

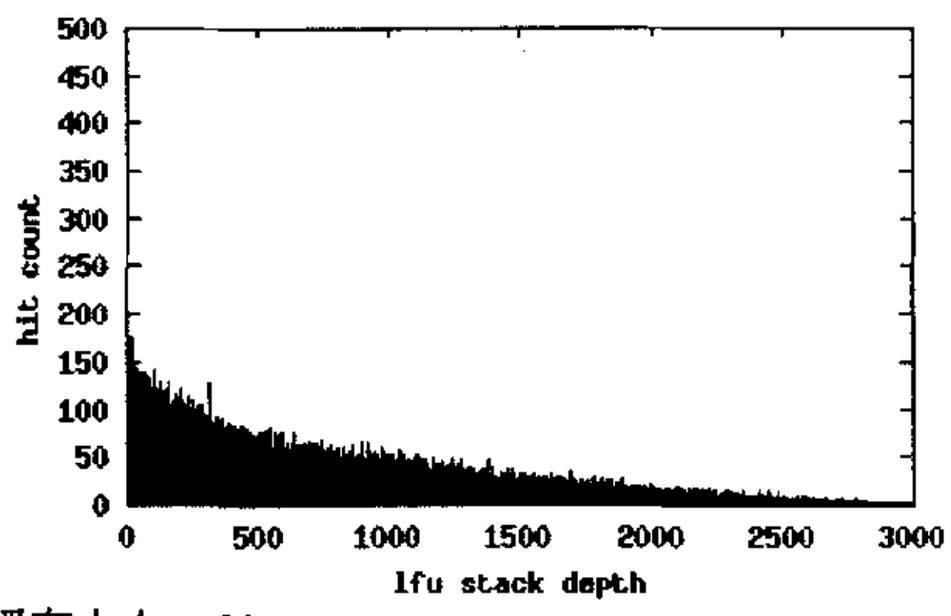
CC size	LC size	LRU	LFU	FBR
0	256	86.2	75.2	83.5
	512	93.4	78.1	91.1
	1024	97.0	82.7	96.0
	2048	98.5	93.4	98.0
64	256	51.5	10.4	40.7
	512	77.1	20.6	68.9
	1024	90.1	39.6	86.8
	2048	94.0	76.3	93.1
256	256	10.3	7.4	8.0
	512	41.1	18.6	30.3
	1024	69.3	40.8	62.2
	2048	84.1	75.3	78.9



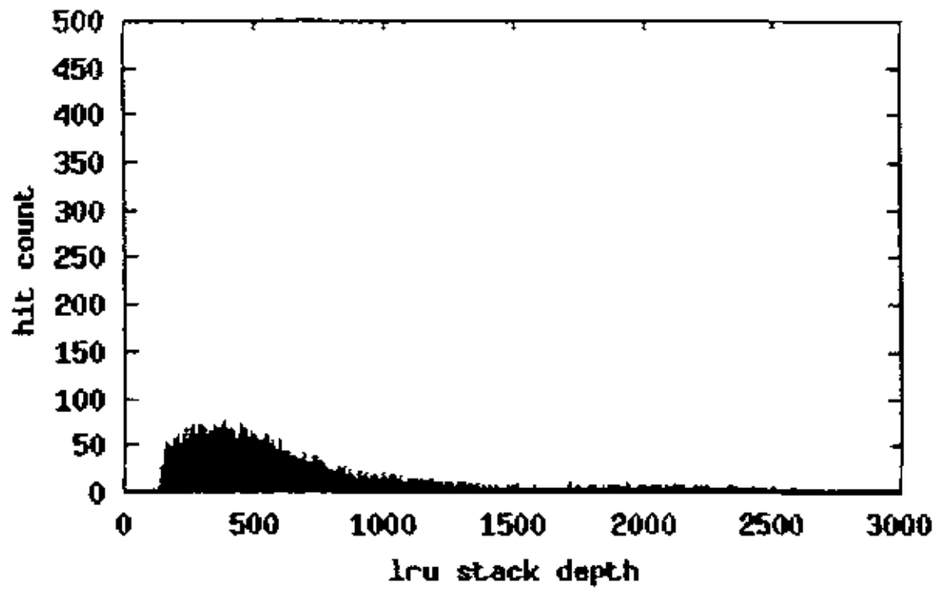


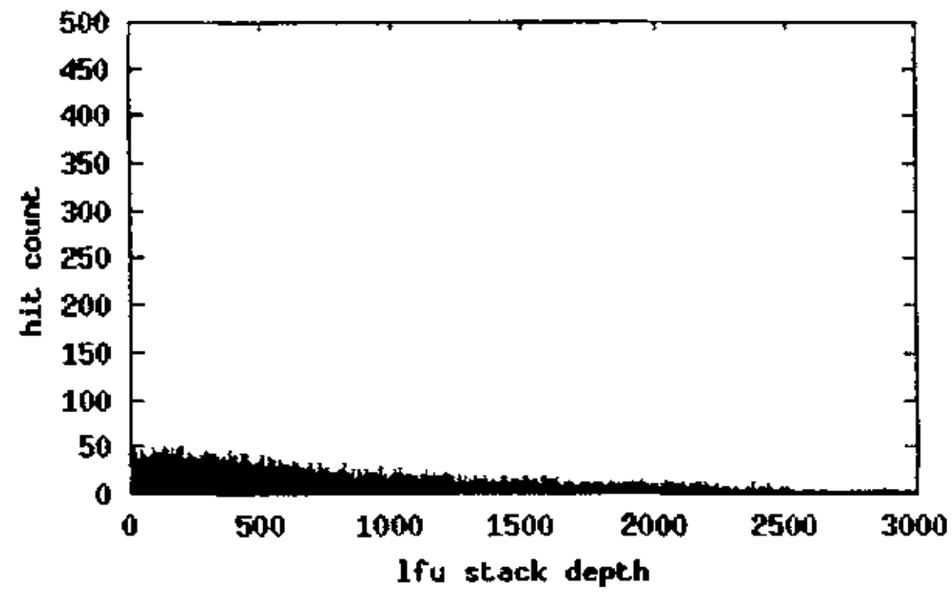
（a）客户端目录缓存大小=0





（b）客户端目录缓存大小=64



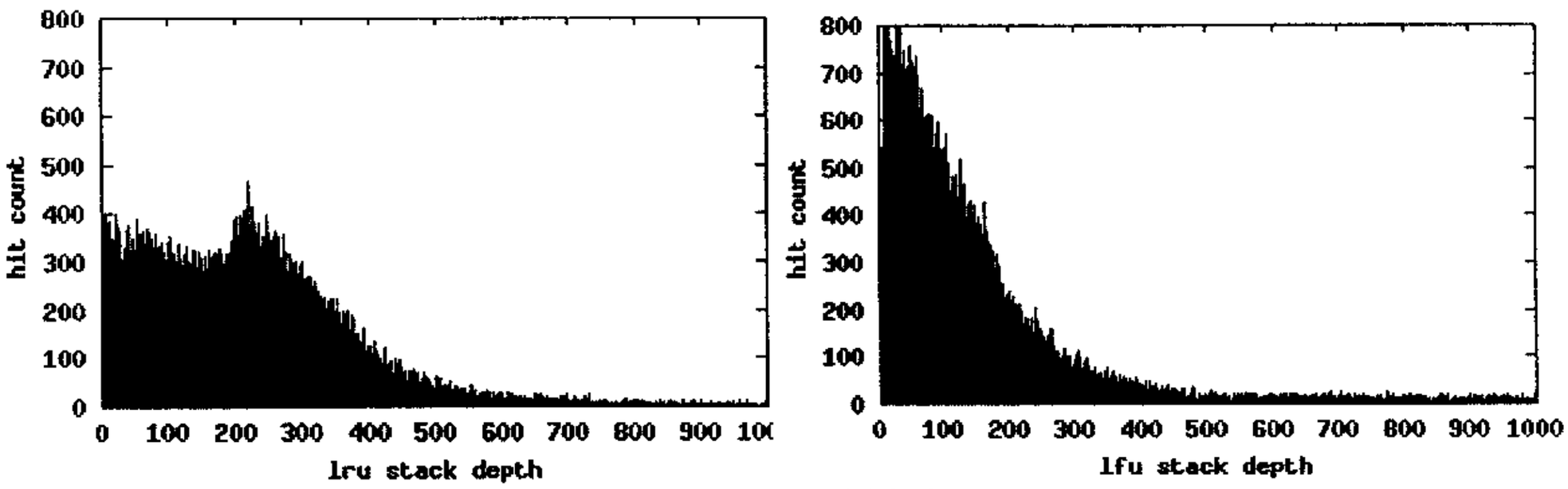


（c）客户端目录缓存大小=256

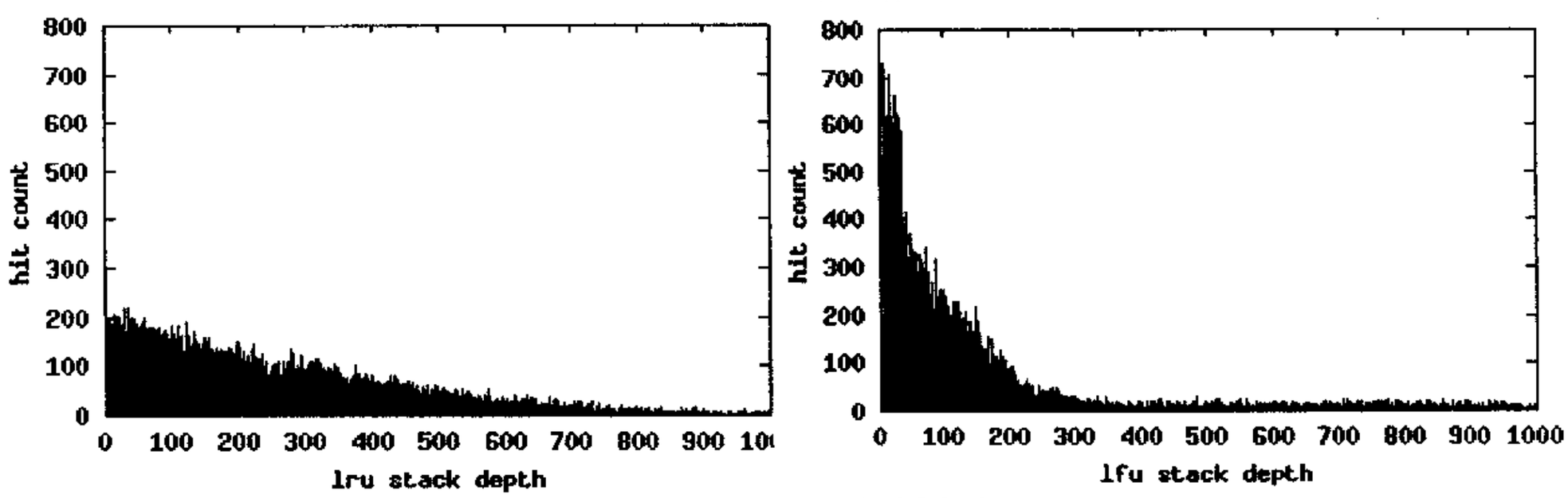
图 4.4 元数据服务器 LOOKUP 缓存 LRU 和 LFU 栈深度直方图（θ=2.50）

表 4.3 READDIR 缓存命中率比较（θ=2.50）

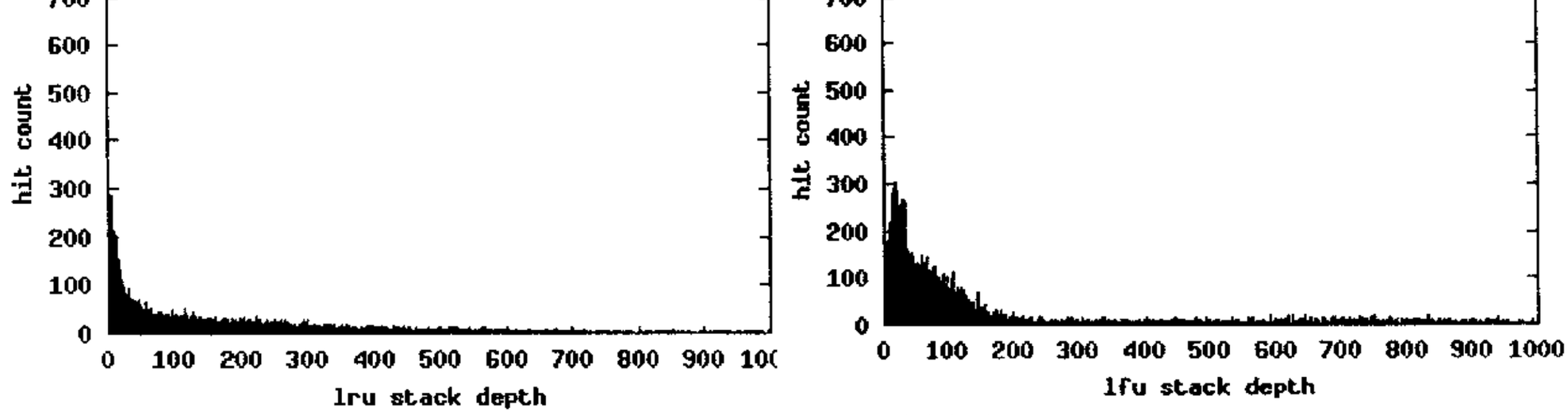
LC size	RC size	LRU	LFU	FBR
256	32	9.6	13.4	5.8
	64	17.9	28.6	11.5
	128	33.6	50.1	21.0
	256	66.8	69.5	40.1
512	32	10.5	13.2	6.6
	64	19.6	31.6	11.9
	128	35.7	51.4	23.4
	256	59.4	67.9	40.9
1024	32	9.6	14.0	7.7
	64	17.0	25.3	12.5
	128	29.8	50.4	25.3
	256	50.2	65.3	56.1



（a）LOOKUP 目录缓存大小=256



（b）LOOKUP 目录缓存大小=512



(c) LOOKUP 目录缓存大小=1024

图 4.5 元数据服务器 READDIR 缓存 LRU 和 LFU 栈深度直方图（θ=2.50）

表 4.4 LOOKUP 缓存命中率比较（θ＝1.75）

CC size	LC size	LRU	LFU	FBR
64	1024	33.9	13.6	27.1
	2048	57.1	26.3	51.3
	4096	75.5	45.7	74.9
	8192	87.3	85.8	84.8

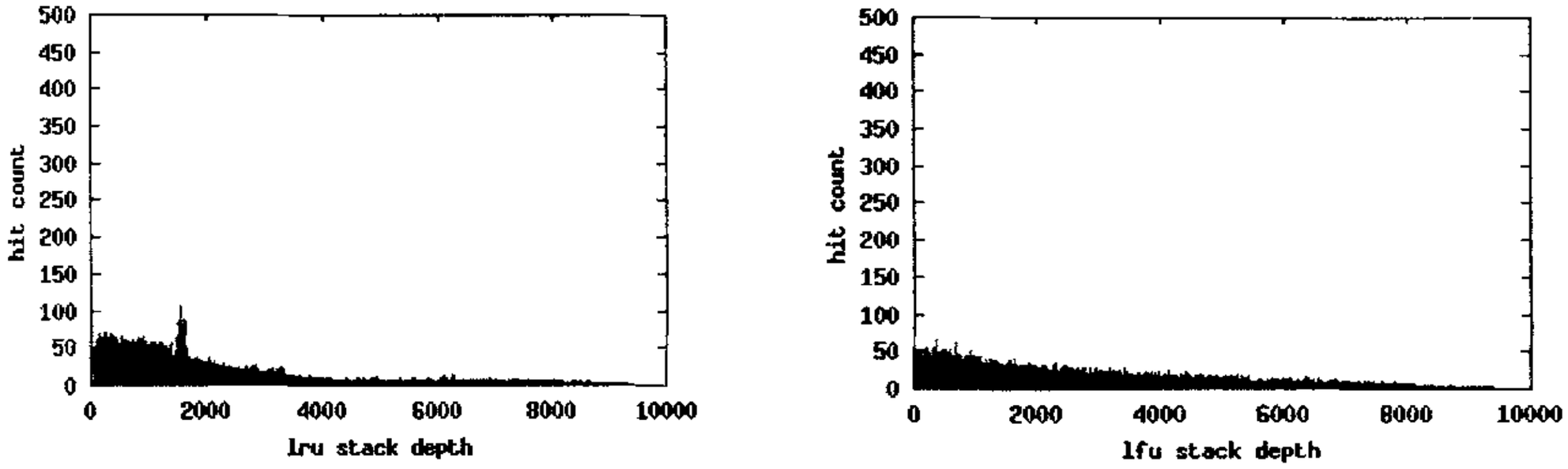


图 4.6 元数据服务器 LOOKUP 缓存 LRU 和 LFU 栈深度直方图（θ＝1.75）

表 4.5 READDIR 缓存命中率比较（θ＝1.75）

LC size	RC size	LRU	LFU	FBR(25:25)
512	64	12.4	12.4	8.6
	128	18.2	22.3	13.6
	256	33.0	41.9	26.2
	512	54.5	65.2	56.9

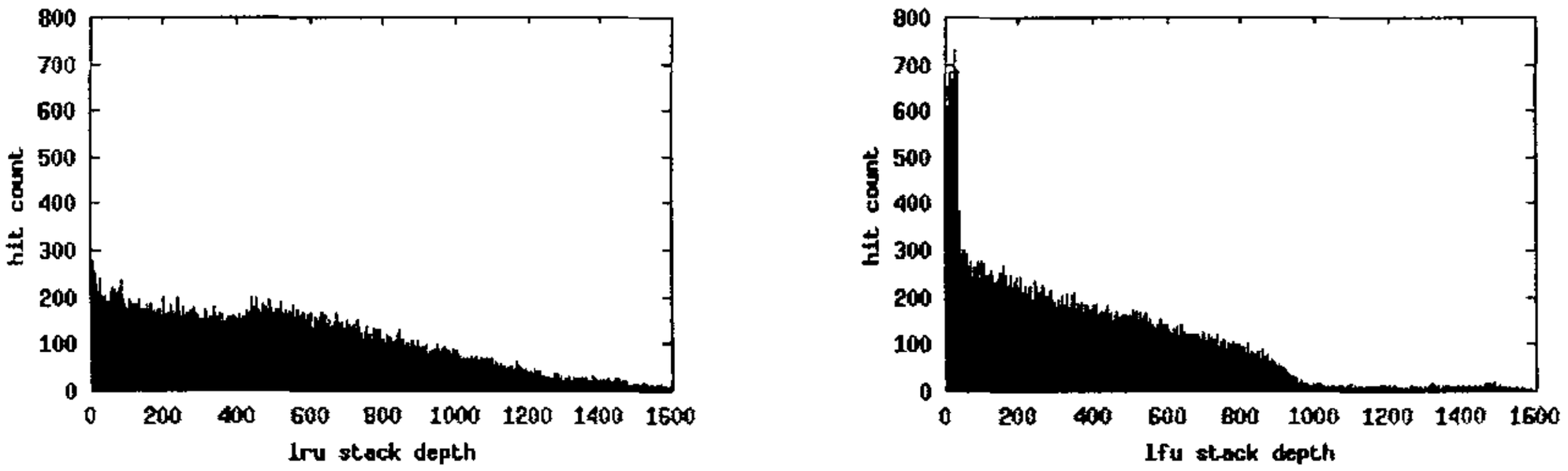


图 4.7 元数据服务器 READDIR 缓存 LRU 和 LFU 栈深度直方图（θ＝1.75）

表 4.4 和图 4.6 给出了当负载生成参数 θ 为 1.75 时 LOOKUP 缓存命中率的比较以及 LRU 和 LFU 栈深度直方图，而表 4.5 和图 4.7 给出了当负载生成参数 θ 为 1.75 时 READDIR 缓存命中率的比较以及 LRU 和 LFU 栈深度直方图。结论与上面讨论的结果相同：对于 LOOKUP 缓存，LRU 缓存替换算法的命中率优于 LFU 和 FBR,而对于 READDIR 缓存，LFU 缓存替换算法的命中率优于 LRU 和 FBR。

4.1.3 小结

从上面的讨论可以得出：

（1） 对于采用元数据服务器的文件系统，由于目录缓存和数据缓存的分离，使目录缓存可以根据自身的特点设计合理的管理方法。首先，4.1.2.2 中的试验表明服务器目录缓存应当采用 4.1.2.1 中描述的 LOOKUP 缓存和 READDIR 缓存相独立的结构，该结构使 LOOKUP 缓存和 READDIR 缓存可以根据自身的访问特性选择合适的替换算法，可以获得更高的缓存命中率。其次，元数据服务器上的 LOOKUP 缓存和 READDIR 缓存的粒度不同，所表现出的目录项访问特性也不相同，对于 LOOKUP 缓存，其缓存粒度为单个目录项，表现出了更明显的 LRU 访问特性，LRU 缓存替换算法更适合 LOOKUP 缓存；而对于 READDIR 缓存，其缓存粒度为目录块，表现出了更明显的 LFU 访问特性，LFU 缓存替换算法更适合 READDIR 缓存；

（2） 客户端目录缓存、元数据服务器 LOOKUP 缓存和 READDIR 缓存构成了多级的目录缓存结构，对于 LOOKUP 缓存，由于客户端目录缓存的“过滤”作用使 LOOKUP 缓存的缓存命中率随客户端目录缓存容量的增加而下降，并且通过分析服务器端目录访问的 LRU 栈深度直方图发现，当客户端目录缓存容量不为 0 时，服务器的 LRU 栈深度直方图在深度较小处（小于客户端缓存的大小）命中次数很小，这表明为了使服务器目录缓存获得较高的缓存命中率，其容量必须大于客户端目录缓存容量。另外，服务器端 READDIR 缓存的命中率随 LOOKUP 缓存的容量增加变化缓慢，其容量设置为远大于 LOOKUP 缓存容量时才能获得较高的缓存命中率。

4.1.4 元数据服务器目录缓存管理

从上面的分析中可以看到，在客户端目录访问负载表现出较强的时间局部性（θ=2.50）或较弱的时间局部性（θ=1.75）的情况下，元数据服务器的目录缓存表现出如下的特点：（1）LOOKUP（目录项）缓存应当采用基于 LRU 的缓存替换算法；（2）READDIR（目录文件块）缓存应当采用 LFU 的缓存替换算法。通过分析文件系统 LOOKUP 操作在元数据服务器上的处理过程，我们看到只有在 LOOKUP 缓存中不存在需要查找的目录项时，元数据服务器才需要在 READDIR 缓存中查找，目录块中的目录项在 LOOKUP 缓存中的数目反映了该目录块的使用频率，当目录项被替换时，说明该目录项已经在一段时间内没有被访问，所在的目录块被访问的可能性减少，因此相应地减少目录块的使用频率计数，这样可以避免传统 LFU 替换算法中某个缓存项由于使用频率计数过高而即使不再被访问也不能被替换问题。另外，我们通过分析用户目录操作的序列发现，用户的目录操作通常表现为相似的操作序列，例如常用的目录显示命令 ls，READDIR 操作通常会紧跟着一系列的对 READDIR 操作获取的目录块中的目录项的 LOOKUP 操作，为了避免频繁的读取文件和对目录块搜索的开销，DCFS 的 READDIR 操作将目录块中的项插入到 LOOKUP 缓存中。下面给出了 LOOKUP 缓存和 READDIR 缓存管理方法的描述。

```
rc_getblk (dir_ino, off)   //获取 READDIR 目录缓存块
{
    blk_no    = rc_lookup(dir_ino, off);
    if (blk_no >= 0) {
        rc_update_list(blk_no,1)
    } else {
```

```
        if (there are free READDIR blocks)
            blk_no = get_free_rc_blk();
        else
            blk_no = rc_replace(); //替换使用频率计数最小的块

        read the directory file's block into blk_no;
        rc_update_list (blk_no, 1);
}
}

rc_update_list (index，hint)  //更新 READDIR 缓存链表
{
    rc_elm = get_rc_elm(index); //由索引值获取缓存项
    if (hint < 0) {  //减少使用频率计数
        rc_elm.frequency--;
    }else {          //增加使用频率计数
        rc_elm.frequency++;
    }

    if (hint < 0) {  //向前搜索直到缓存项的使用频率计数小于 rc_elm 的计数
        prev_elm = rc_elm;
        while (prev_elm!=NULL) {
            if (prev_elm.frequency < rc_elm.frequency)break;
            prev_elm = get_prev_elm(prev_elm.index);
        }
        if (prev_elm != NULL) move rc_elm to the position after prev_elm
} else {   //向后搜索直到缓存项的使用频率计数小于 rc_elm 的计数
        next_elm = rc_elm;
        while (next_elm!=NULL) {
            if (next_elm.frequency > rc_elm.frequency)break;
            next_elm = get_next_elm(prev_elm.index);
        }
        if (next_elm != NULL)
            move rc_elm to the position before next_elm
        else
            move rc_elm to the head of the list
    }
}

lc_get_dentry (dir_ino, name) //获取 LOOKUP 目录缓存的目录项
```

```
{
    dentry    = lc_lookup(dir_ino, name);
    if ( dentry != NULL) {
        lc_update_list(dentry)
    } else {
        if (there are free LOOKUP cache elm)
            dentry = get_free_dentry();
        else
            dentry = lc_replace(); //替换最近最少使用的项

        read the dentry from directory file;
        lc_update_list (dentry);
    }
}

lc_update_list (index)   //更新 LOOKUP 缓存链表
{
    lc_elm = get_lc_elm (index);
    move the lc_elm to the head of the LRU list
}
```

采用上面描述的目录缓存管理算法时，目录 LOOKUP 操作、READDIR 操作等目录操作的流程描述如下：

① LOOKUP 处理过程

1. 根据文件名在 LOOKUP 目录缓存的 Hash 表中查找；
2. 如果找到，调用 lc_update_list 更新 LOOKUP 缓存链表，同时更新 READDIR 缓存的链表，并返回 inode；
3. 如果没有找到，则读入未在缓存中的目录文件块进行查找，如找到则从 LOOKUP 缓存中分配一项来存放该目录项，同时更新 READDIR 缓存的链表。

② READDIR 处理过程

1. 根据传入参数中指定的偏移查看对应项是否存放在目录缓存中；
2. 如果命中，更新 READDIR 缓存链表，然后返回目录块在缓存中的块号；
3. 对于没有命中的块：

   3.1 分配目录块缓存，从磁盘上的目录文件中读入对应的块并更新 READDIR 缓存链表；

   3.2 对于该块中的每个目录项，从 LOOKUP 缓存中分配一项来存放该目录项，并插入到 LOOKUP 缓存的 Hash 表中

表 4.6 给出了采用如上描述的缓存管理方法和其他几种方法缓存命中率的比较。第三列给出的是在上面描述的方法的命中率，LFU、LRU 和 FBR 列分别表示当服务器端

LOOKUP 缓存和 READDIR 缓存采用相同的 LFU、LRU 或 FBR 替换算法时的命中率。缓存命中率定义为 LOOKUP 操作在目录缓存（包括 LOOKUP 和 READDIR 缓存）中查找到指定目录项的比率。客户端负载与 4.1.2 小节中的试验相同，采用 FSbench 产生负载，其中 θ=2.50，可以看到，由于我们的目录缓存管理方法利用了服务器目录缓存的访问特性，在本节中给出的方法的缓存命中率高于其他的方法。

表 4.6 缓存管理方法命中率比较（θ=2.50）

LC size	RC size	NEW	LRU	LFU	FBR
256	32	38.0	34.0	27.3	13.4
	64	49.7	42.2	43.9	24.2
	128	67.2	56.6	64.0	36.6
	256	86.9	83.2	76.3	48.6
512	32	69.4	66.6	37.5	29.2
	64	76.3	70.7	50.7	37.2
	128	85.7	77.6	67.2	46.8
	256	91.4	87.7	81.2	58.0
1024	32	87.3	85.4	56.5	59.0
	64	90.2	86.8	63.2	64.7
	128	93.6	89.2	75.8	68.8
	256	95.6	93.5	85.9	76.1

4.2 大目录优化

如图 4.1 给出的目录文件结构，许多文件系统的目录文件按传统的线性方式组织，当目录中文件个数增多时，由于线性查找的开销增大，目录操作的性能将下降。许多文件系统对大目录的组织方式进行了研究[Bar98][Pre99][Rei03][XFS03]，提出了 B-树和动态可扩展 Hash 技术（Extendible Hashing）等技术。在本小节中，作者首先对几种常用的技术进行了分析，然后描述了与本研究小组成员合作提出并实现的一种改进的动态可扩展 Hash 技术（Extendible Hashing）,我们称其为受限的多层次可扩展 Hash 技术（Limited Multi-level Extendible Hashing，简称 LMEH）[Tan03]。

4.2.1 相关研究分析

Reiser[Rei03]和 XFS[XFS03]等文件系统均采用 B-树作为目录文件的组织形式。B-树是一种平衡的多路查找树，树的中间结点和叶子结点都可以用来存放目录项，由于 B-树的各结点组织成平衡的结构，B-树中包含目录项较多时仍能保持较高的查询效率。论文[Pre99]中对 B-树和动态可扩展 Hash 技术进行了分析比较，认为 B-树的目录文件组织方式的查找效率与树的深度有直接的联系，树的深度越大，其平均搜寻路径越长，需要有多次访问存储设备的操作，在存放相同数量的目录项情况下，动态 Hash 技术的平均查找时间少于 B-树的平均查找时间。另外，在 B-树中增加一个目录项，有可能需要调整树中某个子树的结构，在最坏的情况下，甚至需要调整整棵 B-树，这种调整过程同样要求多次对存储设备的操作，因此，对系统的性能会有很大的影响。同时，B-树的实现也比较复杂，这对系统的实现增加了难度。

GPFS[Bar98]和 GFS[Pre99]采用了动态可扩展 Hash（Extendible Hashing，简称为动态 Hash）技术对目录文件进行管理。文件名 Hash 函数值的 n 位有效位作为 Hash 表的索引值，

Hash 函数值的有效位由目录大小决定，随着目录项的不断增加和减少而相应地变化。当目录中添加新的目录项时，如果目录文件块中的目录项已经填满，这时需要分裂目录块，并相应地增加 Hash 函数值的有效位，图 4.8 给出了分裂的过程。如图所示，在开始时取 Hash 函数值的最低位作为有效位，此时，Hash 表的大小为 2，所有的目录项都存放在 Hash 表中的各个入口项所指向的磁盘块中。假设为第 2 个入口项所指的磁盘块已经放满，则说明取一个有效位已经不够，则增加一个有效位，如图所示，1 所指的块分裂为两块，10 仍然指向原来一块，而 11 指向新增加的磁盘块，然后依次判断原来 10 所指向磁盘块中的所有目录项，将文件名 Hash 值的第三位为 1 的目录项移到 11 所指向的磁盘块中。从上面的描述可以看出，每一次分裂所引起的 Hash 表的调整会带来较大的开销。

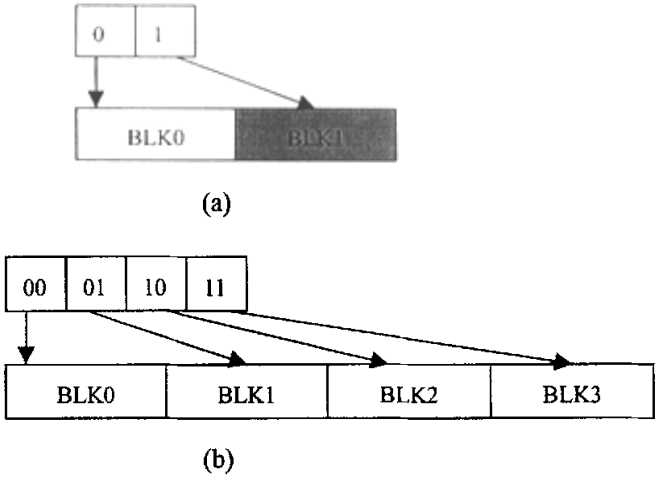


图 4.8 动态 HASH 块分裂示意图 （a）分裂前；（b）分裂后

根据 Hash 表组织方式的不同，动态 Hash 技术可以分为两类：（1）目录动态 Hash（Directory Extensible Hashing）技术；（2）无目录动态 Hash（Directoryless Extensible Hashing）技术。在目录动态 Hash 技术中，Hash 表需要用独立空间来存放，而无目录动态 Hash 技术则不需要存放 Hash 表。

GFS 采用了目录动态 Hash 的技术，将 Hash 表存放于磁盘块中，对 Hash 表的管理类似于普通文件的数据块多级间址的方法，目录块存放在磁盘块中，由 Hash 表表项中的指针指向目录块所在的磁盘块。可以看到，这种方法在添加新的目录项使得目录块发生分裂，同时必须扩大 Hash 表时，它须将原来的 Hash 表从磁盘中读入，更新完成后再将大小扩展为原来两倍的 Hash 表写回磁盘。当 Hash 膨胀到一定大小后，这种回写对系统带来了额外的开销。

GPFS 则采用了无目录动态 Hash 技术，GPFS 并不存放 Hash 表。设当前文件名 Hash 函数值的有效位为 n，要查找的文件的 Hash 函数值的 n 位有效位的值为 $m=a_0a_1\ldots a_{n-1}$，则对应的目录块的位移为 m*blk_size，如果该目录块为空洞，则减少有效位数，直到所要查找的目录块不为空洞。GPFS 采用的无目录动态 Hash 技术在 Hash 函数成生的函数值不均匀时，目录文件的大小会急剧增加，极端的情况下，目录文件的大小会很快超过文件系统定义的最大文件长度，这时，即使目录文件的所占用的实际磁盘空间并不大，但却不能继续创建新的文件。

从上面可以看到，对于 GFS 所采用的大目录管理技术，其问题主要在于 Hash 表中所有入口项都保持在同一层次，也就是，它们的有效位数任何时候都相同，因此，为了维护这种相同层次结构给系统带来了性能上的影响；而对于 GPFS，在 Hash 函数值不均匀时会产生出于目录文件的急剧膨胀而无法创建文件的情况。因此，我们提出了受限的多层次动态 Hash 技术：（1）为了避免为维护相同层次的 Hash 表所带来的开销，在此采用多层次 Hash 表方法，在下面的小节中，将详细描述该方法；（2）为了避免 Hash 表的急剧膨胀的问题，采用了与 GFS[Pre99]中类似的方法，使用索引文件存放 Hash 表并限定 Hash 函数值的最大有效位数。

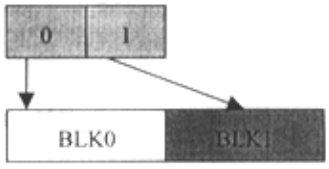
4.2.2 受限的多层次可扩展 HASH 技术（Limited Multi-level Extendible Hashing， LMEH）

采用 LEMH 技术的目录组织方式中，每一个目录用两个文件表示：（1）一个是目录项文件，用于存放该目录的所有目录项结构，它按固定大小的块进行组织，同一块中的目录项 Hash 值相同，当添加新的目录项需要扩大目录文件时，新增的目录块附加在目录文件的末尾，所以，在目录文件中不存在空洞(Hole)；（2）另一个是索引文件，按 Hash 值的顺序存放索引结构，索引结构用于记录相应 Hash 值所对应的块在目录项文件中的位置，以及一些与动态 Hash 技术相关的信息，为了防止索引文件膨胀而占用过多的磁盘空间，与 GFS[Pre99]类似，LMEH 限制 Hash 函数值的最大有效位数，当 Hash 函数值的有效位数增加到定义的最大位数后，相同 Hash 函数值的目录块按照链表的结构连接起来。所以，在 LEMH 中，当文件名 Hash 函数值有效位低于最高有效位时，查找目录项的操作包含两次的文件读写操作：从索引文件中读取索引结构和根据索引结构中的指针读取目录块；而当文件名 Hash 函数值有效位已增加到最高有效位时，目录项查找操作需要沿着目录文件块构成的链表进行搜索，可能需要多次磁盘操作。

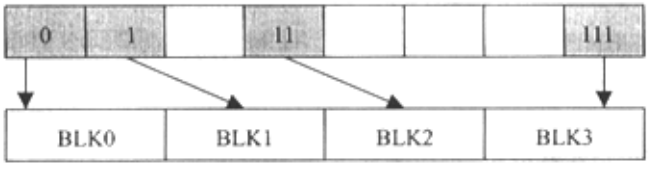
与 GFS 不同的是：LEMH 采用多层次 Hash 表结构，即每个目录块 Hash 函数值的有效位数可以不同，假设当前文件名 Hash 函数值的有效位数位 n，索引结构所指向的目录块的 Hash 函数值的有效位数位 m(m<=n)，则该目录块中的目录项的 Hash 函数值 m 位有效位相同，而 n 位有效位可以不相同。当一个目录文件块已满，进行分裂时，无需修改所有新增的入口项，也就是索引文件中索引结构的值。这样可以避免 GPFS 和 GFS 中为维护相同层次的 Hash 表所带来的开销，同时也节省了索引文件和目录项文件的空间。

图 4.9(a)至(b)中表示了目录文件块分裂的过程。图中所示有效位为 1 位时的索引文件和目录项文件结构，图中序号为 1 的索引结构对应的块 Blk1 表示该目录项块已满，当在该块中添加新的目录项时，需将它分裂为两块，新增的块 Blk2 附加在目录项文件的末尾，并且它的位置由标号为 11 的索引结构指示。而标号为 10 的索引结构在索引文件中为一个大小等于单个索引结构长度的空洞，它在目录项文件中没有对应的块。如果经过一段时间后标号为 11 的索引结构对应的块 Blk2 也满，则再次分裂，得到图 4.9(b)所示的索引文件结构，此时在文件内部只有四个索引结构的值有效。由图可以看出，此时 Hash 表中有三种不同的有效位，比如标号为 0 的索引结构的 Hash 值为 0，对应的块 Blk0 中存放的是 Hash 值形如 xx0 的，也就是 010、100、110 的目录项，这些目录项本该分别移入对应 010、100、110 所对应的块中，也就是图中所示空白索引结构所对应的目录文件块中。只有在这些有效位较低的 Hash 值所指的块需要分裂时，才一次性将各个目录项移到相应的块中，例如，

假设标号为 0 所对应的 Blk0 已满，则将所有的低三位 Hash 值为 100 的目录项都移到标号为 100 的索引结构所对应的块中，因为此时索引结构为空，没有对应的块，则为它在目录项文件的末尾分配一块，并用该块的位置值初始化索引结构。



(a)



(b)

图 4.9 受限的多层次可扩展 HASH 技术 （a）分裂前；（b）分裂后

在 LMEH 的目录文件组织方式中，增加与查询目录项的具体算法描述如下：用全局深度 global_depth 表示某一时刻的最大有效位数（这个值记录在索引文件的某个固定位置），同时在每一个索引结构中加入局部深度 local_depth 表示该块中的目录项所对应的有效位数。每一次分裂后 global_depth 值增一，而相应被分裂的块和分裂生成的块的 local_depth 都等于此时的 global_depth。

```
Insert ( parent_directory, entry_name)
{
    Index_struct = null;
    Compute global_depth;
    Hash_value = name_hash(entry_name);
    Current_hash_mask_bits = global_depth;

    While ( index_struct == null) {
        Bucket_number = hash_value   AND   current_hash_mask_bits;
        Index_struct = get_index_struct ( bucket_number );
        Current_hash_mask_bits >>= 1;
    }
    bucket_offset = index_struct . Offset_within_file;
    bucket = get_bucket (bucket_offset);
    if (bucket is full) {
        if ( index_struct . Local_depth == global_depth && global_depth reaches limitation) {
            Allocate new entry block and chained it with bucket;
            Store the entry in the new block;
```

```
            Return;
        }

        Split ( parent_directory, global_depth, index_struct); // see below;
        insert ( parent_directory,    entry_name);
    } else {
      Store the new entry into bucket;
      Return;
    }
}

Split (parent_directory, global_depth, index_struct)
{
    If ( index_struct . Local_depth < global_depth) {
            New_local_depth = global_depth;
    } else {
            global_depth ++;
            New_local_depth = global_depth;
    }
    bucket = get_bucket (index_struct.Off_within_file);
    for (every entry in bucket ) {
        hashvalue = name_hash ( entry name);
        bucket_number = New_local_depth least significant bits of hashvalue;
        if (bucket_number not equal to current bucket number) {
            new_index_struct = alloc_new_index_struct (bucket_number);
            new_bucket = alloc_new_bucket ( new_index_struct);
            store entry into new_bucket;
            new_index_struct . Local_depth = New_local_depth;
            new_index_struct . Offset_within_file = offset of new_bucket within entry file;
        }
    }
    index_struct . Local_depth = New_local_depth;
}
```

4.2.3 分析与试验

为了测试目录组织采用 LMEH 技术后所获得的性能的改进，我们在 DCFS 文件系统元数据服务器上实现了 LMEH 的原型。表 4.7 中给出了 LMEH 算法和传统的线性组织算法在

每个目录创建 5000 个文件的情况下的性能比较，表 4.8 给出了 DCFS 元数据服务器实现 LMEH 算法后吞吐率测试结果，表中第一行表示客户节点的数目。测试平台与 6.3.3 小节中介绍的相同，测试程序及测试方法也与 6.4.1 小节中介绍的相同，DCFS 配置为 8 个元数据服务器和 1 个存储服务器，客户节点数为 22，每个客户节点启动一个测试进程，每个测试进程在独立的目录中创建文件。每个创建文件的过程包括 LOOKUP 和 CREATE 操作，这些操作需要发送请求到元数据服务器，因此 LMEH 算法和线性算法的网络开销和客户端的开销相同，但 LMEH 算法可以减少每个操作在元数据服务器的服务时间，减轻服务器的负载。在表 4.7 中，我们可以看到，LMEH 算法的创建吞吐率比原 DCFS 的吞吐率有很大的提高，平均提高了 1.97 倍，其中在 20 个客户端测试进程时，LMEH 聚合吞吐率是优化前的 2.50 倍。表 4.7 给出了 LMEH 算法在每个目录创建 10000、20000、50000 和 100000 文件的性能，LMEH 算法在每个目录创建 100000 时的性能仍优于线性算法的创建 5000 个文件的测试的性能。

表 4.7 LMEH 和线性目录组织创建吞吐率比较

Client Num.	*1*	*4*	*8*	*12*	*16*	*20*	*22*
Linear	249.3	922.1	1738.3	2128.3	2603.5	2765.0	—
LMEH	482.6	1552.4	2883.8	4219.8	5610.8	6913.2	7472.2

表 4.8 LMEH 大目录创建吞吐率

Client Num.	*1*	*4*	*8*	*12*	*16*	*20*	*22*
10000	468.9	535.5	2846.3	4139.2	5497.0	6631.9	7146.9
20000	461.9	1515.1	2813.4	4018.7	5224.4	6144.3	6575.6
50000	414.9	1382.0	2582.6	3295.7	4008.8	4257.2	4458.1
100000	350.8	870.9	2209.4	2405.9	2791.3	2764.9	2809.2

4.3 小结

作者在本章讨论了元数据服务器目录操作的两个问题：目录缓存和大目录优化。在 4.1 节中，作者对 DCFS 中层次目录缓存的结构进行了讨论，发现元数据服务器上的 LOOKUP 缓存和 READDIR 缓存表现出不同的访问特性，在此基础上给出了元数据服务器目录缓存的管理方法。4.2 节通过分析现有文件系统解决大目录问题的技术，与本研究小组成员合作给出了一种改进的动态 Hash 算法 LMEH，试验表明，该方法可以有效的改进元数据服务器在大目录情况下查找与创建目录项的性能。

第五章 机群文件系统元数据一致性

DCFS 文件系统采用了元数据分布策略，每个文件系统卷可以配置成多个元数据服务器，这为增强文件系统可用性提供了可能，文件系统在某些元数据服务器发生故障时仍能由其余的服务器提供服务，同时，多元数据服务器的结构增加了维护文件系统元数据一致性的难度，因为元数据操作可能需要多个服务器间相互合作，另外，DCFS 文件系统分布式的结构要求元数据一致性协议能够在各个组成部分出现故障时进行处理。在本章中，作者描述了 DCFS 元数据一致性协议，并进行了分析。

5.1 机群文件系统一致性

机群文件系统由节点系统、存储设备、网络以及执行不同功能的软件部件构成，不同的组成部分发生故障时，机群文件系统可能会处于不一致的状态，机群文件系统高可用协议负责维护文件系统的一致性。文件系统一致性分为两个层次：元数据一致性和数据一致性。元数据一致性指保证元数据的正确性和完整性，未完成的元数据更新操作的结果不会反映到文件系统中，数据一致性除保证元数据一致性外，还保证只有成功的写操作的结果才会反映到文件系统中。在本章中，我们仅讨论元数据一致性。

许多文件系统在系统出现故障后依靠 fsck 工具将文件系统恢复到某个一致性状态，但对于机群文件系统而言，其管理的文件系统的容量非常巨大，fsck 工具执行需要很长时间，并且对于机群文件系统，希望某个节点故障尽可能少的影响其他节点的运行。Minwen Ji 等在[Ji00]中给出了 Archipelago 系统中维护文件系统可用性的协议。Archipelago[Ji00]采用了基于目录的元数据分布策略，目录在多个节点中复制，目录的复制导致了 Archipelago 必须解决客户节点元数据访问的序列化问题，因此，该系统引入了逻辑同步时钟、两阶段提交协议、写前日志（Write-ahead Log）和基于状态转换的恢复协议相结合的方法。通过分析[Ji00]中描述的协议可以看到，该协议为了保证逻辑时钟与更新操作的原子性，任何时候只能执行一个更新操作，这降低了服务器并发处理的能力。Diffs[Zho01]提出了一种维护文件系统名字空间一致性的协议。该协议将所有的与名字空间相关的操作分解为增加索引（link）和删除索引（unlink）两个操作的组合，通过解决这两个操作可能引起的名字空间的不一致的情况来减少操作的关键路径上同步日志、消息来回和临界资源互斥的开销。该协议虽然保证名字空间的一致性，但不保证文件系统操作的原子性。

元数据一致性协议的设计与文件系统采用的结构、元数据分布策略有很大的关系，不同的结构和分布策略需要设计不同的协议。DCFS 采用了元数据服务器和存储服务器分离的结构，并且采用的元数据分布策略不需要维护元数据的多个副本，对相同对象的操作都只有唯一的同步节点，这些简化了 DCFS 元数据一致性协议的设计，另外 DCFS 元数据服务器管理各种元数据缓存，需要处理元数据缓存和分布式事务间的关系。本章将给出元数据一致性协议，该协议采用了分布式事务技术和元数据日志技术来保证文件系统操作的原子性，分析表明协议的开销是可以接受的。关于 DCFS 元数据一致性协议的具体实现可见论文[Fan04]。

5.2 机群文件系统故障类型

机群文件系统元数据一致性协议应当处理如下的故障类型：

（1） 事务故障。文件系统元数据操作因为各种原因不能完成，我们称为事务故障。产生事务故障的原因包括缓存分配失败、由于产生死锁不能继续执行等，当发生事务故障时，机群文件系统需要中止（ABORT）该事务，将文件系统恢复到一个一致状态；

（2） 节点故障。节点硬件、操作系统错误等都可以造成节点故障，在本章描述的元数据一致性协议中，我们将只考虑单个节点故障的情况。节点发生故障时，元数据缓存中“脏”的元数据将会丢失，为了使故障节点在重启后恢复到一个一致性状态，元数据服务器必须在磁盘中保存有元数据更新的信息，在后面的小节中，作者将介绍 DCFS 采用的一些技术，包括元数据日志、恢复协议等。当系统中某个节点发生故障时，其余节点还需处理与故障节点相关的事务，在 5.3.3.2 小节中将进行描述。由于 DCFS 将元数据以文件的形式存储在本地文件系统中，当故障节点重启时，DCFS 假设本地文件系统可以利用日志或 fsck 等手段恢复到某个一致性状态。

（3） 存储系统故障。当存储系统发生故障时，其上存储的信息将丢失，通常用备份和复制的方法以便在发生存储系统故障时进行恢复。因为存储系统技术的进步，其可靠性很高，发生故障的几率很小，在当前的 DCFS 版本中，我们不处理此类故障。

（4） 网络故障，主要指通信链路故障。发生网络故障时，文件系统中各个节点不能发送和接收消息，DCFS 事务可能处于某个状态，DCFS 元数据一致性协议的终止（Termination）协议在发生网络故障时负责处理这类故障。

5.3 机群文件系统元数据一致性协议

5.3.1 事务定义

DCFS 的事务定义为读取、修改元数据的操作序列，是保证 DCFS 元数据一致性和完成客户端元数据请求的单位，与数据库系统中的事务一样[Ozs99]，DCFS 事务具有如下性质：（1）原子性（Atomicity）。事务中操作要么都完成，要么都不完成，即“all-or-nothing”；（2）一致性（Consistency），当 DCFS 事务成功完成时，DCFS 元数据保持一致性；（3）隔离性（Isolation），事务间相互独立，任何事务对元数据的修改只有在该事务成功提交后才能被其他事务“看见”；（4）永久性（Durability），对元数据的修改如果已写入永久存储设备中，其结果将被永久保留。

DCFS 事务与文件协议的元数据操作相对应，例如 DCFS 文件协议中的 DCFS_LOOKUP 操作，我们将 DCFS_LOOKUP 在元数据服务器执行的过程视为一个事务。一个 DCFS 元数据操作可能只需一个服务器即可完成或需要多个服务器的配合，不同元数据操作可能由不同的元数据服务器进行服务。

5.3.2 并发控制协议

DCFS 元数据服务器对每个文件使用锁来实现元数据的互斥访问，允许执行多个并发的文件操作。DCFS 采用两阶段锁协议（2PL－2 Two Phase Locking）[Ozs99]实现多个事务

之间的并发访问。DCFS 两阶段锁协议（2PL）遵守如下原则：任何事务在第一次释放锁之后不能再获取任何一把锁。事务的执行可以分成两个阶段，取锁阶段和释放锁的阶段。DCFS 元数据服务器在处理事务的过程中只取锁而不释放锁，只有事务结束时才释放所有的锁。

5.3.3 元数据一致性协议

DCFS 元数据一致性协议包括三个部分：（1）正常操作处理协议（Normal Operation Protocol）；（2）终止协议（Termination protocol）；（3）恢复协议（Recovery protocol）。正常操作处理协议指元数据操作在没有错误发生的情况下为了维护分布式事务的原子性必须遵守的规则，结合机群文件系统操作流程和两阶段提交协议（Two-phase Commit），在 5.3.3.1 小节中，作者将描述该协议。终止协议用于分布式事务执行的过程中节点故障时，其余的非故障节点上事务的恢复处理。与终止协议对应，恢复协议在故障节点重启时将文件系统恢复到某个一致性状态。

5.3.3.1 正常操作处理协议

元数据操作可能需要多个元数据服务器的相互配合才能完成，协调者给多个元数据服务器发送元数据操作请求，每个服务器在完成服务请求后回送结果。结合元数据操作请求的处理过程和两阶段提交协议，参照[Ozs99]中给出的两阶段提交协议的状态转换图，在图 5.1 中给出了 DCFS 元数据处理过程中的状态转换。

图 5.1 中圆圈代表状态，方框代表一个操作，虚线表示消息的传递。对于一个涉及多个元数据服务器的操作，其过程简单描述如下：

（1） 协调者完成自身的操作并同步记录到日志中（由于 DCFS 采用了第三章介绍的元数据分布与映射策略，从文件系统协议操作的参数可以得知参与者，不需记录参与者的标识），发送请求给元数据服务器，进入 WAIT 状态；

（2） 参与者接收到服务请求后，开始执行服务请求。如果操作成功，在日志中同步记录 READY，回送 SUCCESS 消息给协调者，然后进入 READY 状态；如果操作失败，在日志中同步记录 ABORT，回送 ABORT 消息；

（3） 协调者如果接收到 SUCCESS 消息，如果所有的参与者都已经应答 SUCCESS,则在日志中同步记录 COMMIT，发送 GLOBAL_COMMIT 给所有的参与者，不必等待参与者的应答，协调者必须保证 GLOBAL_COMMIT 消息成功发送；协调者如果接收到 ABORT 消息，在日志中同步记录 ABORT，然后给已经参与元数据操作处理的服务器发送 GLOBAL_ABORT 消息；

（4） 参与者在接收到协调者的决定后，如果消息类型为 GLOBAL_ABORT，在日志中同步记录 ABORT，并回送应答给协调者，如果消息类型为 GLOBAL_COMMIT，在日志中记录 COMMIT，不需给协调者发送应答；

（5） 协调者接收到所有参与者的 ABORT 应答后在日志中写入结束标志。

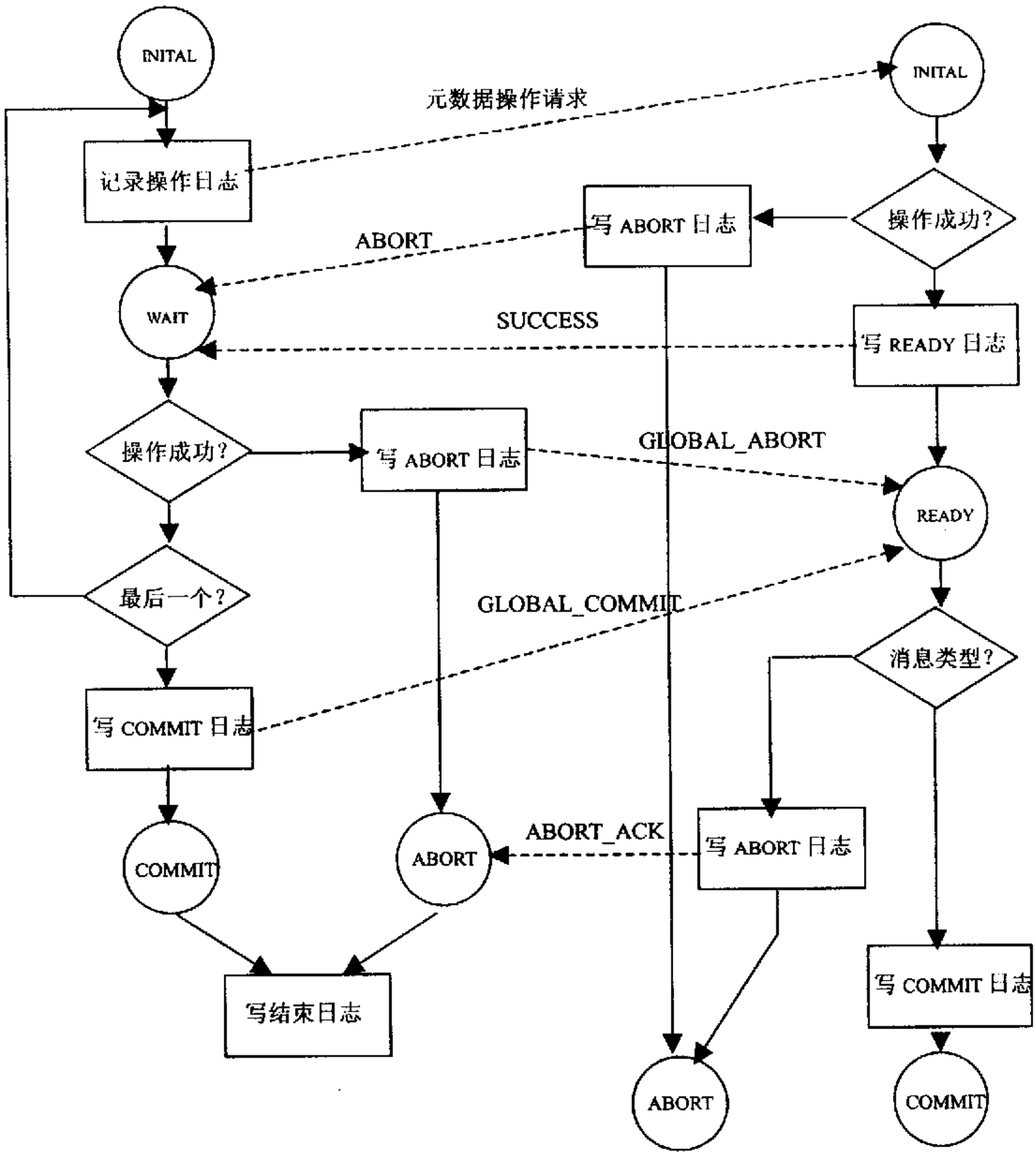


图 5.1 正常操作处理协议

5.3.3.2 终止（Termination） 协议

当节点发生故障时，DCFS 高可用协议通知其余节点，如果其余节点上正在处理的元数据分布事务与故障节点有关联，则节点执行 Termination 协议。根据执行 Termination 协议的节点类型，处理流程如下。

协调者

当参与者发生故障时，从图 5.1 可知，协调者可能处于 WAIT、ABORT 或 COMMIT 状态。针对协调者所处于的状态，作者在下面给出相应的处理流程：

（1） 当协调者处于 WAIT 状态时，它等待其他的元数据服务器回送操作的处理结果，协

调者在接收到监测协议报告参与者出故障的消息后认为该分布式事务失败，进入ABORT 流程，同步记录 ABORT 日志，并给其余的参与者发送 GLOBAL_ABORT 消息；

（2） 协调者处于 ABORT。当协调者处于该状态时，不能确定参与者是否接收到并完成了 ABORT 操作，协调者将重复发送 GLOBAL_ABORT 消息；

（3） 协调者处于 COMMIT 状态。当协调者处于该状态时，与（2）类似，协调者将重复发送 GLOBAL_COMMIT 消息；

参与者

因为参与者只有在收到协调者元数据操作的请求后才能进行相应的操作，当协调者发生故障时，参与者可能处于如下状态；

（1） 参与者处于 INITIAL 状态。这时，参与者没有给协调者回送应答，协调者仍处于 WAIT 状态，因此，参与者只需简单地执行 UNDO 操作；

（2） 参与者处于 READY 状态。参与者已经成功处理完元数据操作请求并给协调者发送了 SUCCESS 应答，由于参与者不知道分布式事务最后结果，因此不能单方面决定 ABORT 或 COMMIT 该事务，必须等待协调者重启后根据日志内容决定 ABORT 或 COMMIT 该事务后给参与者发送相应的决定；

（3） 参与者处于 ABORT 或 COMMIT 状态，在这时，参与者已经完成操作，不需进行任何操作

5.3.3.2 恢复（Recovery）协议

在本小节中，作者将描述 Recovery 协议。从图 5.1 可知，状态转换发生在处理完相应操作、记录日志并发送了消息之后，而故障发生的时间可能在它们之间，在下面，针对协调者和参与者发生故障的位置分别进行描述。

协调者

（1） 故障发生在记录操作日志前。由于协调者没有给任何一个元数据服务器发送操作请求，因此协调者只是简单地 ABORT 该事务；

（2） 故障发生在写操作日志后和给全部参与者成功发送请求之前，则协调者决定 ABORT 该事务并给参与者发送 GLOBAL_ABORT 消息；

（3） 故障发生时协调者处于 WAIT 状态，则协调者决定 ABORT 该事务并给参与者发送 GLOBAL_ABORT 消息；

（4） 故障发生在记录了 ABORT 或 COMMIT 日志后和给每个参与者发送决定消息前。因为一些参与者可能没有收到协调者的决定，因此协调者在重启后重新发送 GLOBAL_ABORT 或 GLOBAL_COMMIT 消息；

（5） 故障发生时协调者处于 ABORT，在节点重启后重新发送 GLOBAL_ABORT 消息；

（6） 故障发生在协调者处于 COMMIT 状态，在节点重启后重新发送 GLOBAL_COMMIT 消息；

参与者

（1） 故障发生在记录操作日志前。参与者在重启后不需做任何动作，因为协调者此时处于 WAIT 状态，它在参与者发生故障后决定 ABORT 该事务并给其余参与者发送 GLOBAL_ABORT 消息；

（2） 故障发生时参与者处于 READY 状态。这时参与者已经成功完成元数据操作，参与者在重启后向协调者查询的决定；

（3） 故障发生时参与者处于 ABORT。参与者在重启后不须做任何处理；

（4） 故障发生在参与者接收到 GLOBAL_ABORT 消息，并记录 ABORT 日志后和发送应答消息前，类似于（3），参与者在重启后不须做任何处理；

（5） 故障发生在元数据操作失败后记录了 ABORT 日志后，但在发送 ABORT 应答前。参与者在重启后不必做任何操作，因为协调者此时处于 WAIT 状态，它在参与者发生故障后决定 ABORT 该事务并给其余参与者发送 GLOBAL_ABORT 消息；

（6） 故障发生时参与者处于 COMMIT。参与者在重启后不须做任何处理；

5.3.3.4 协议优化

从表 1.2 中给出的各种操作比例中可以看到，LOOKUP 等只读操作占了很大比例，针对这些只读操作，DCFS 的元数据操作流程进行了相应的优化。协调者发送元数据请求给参与者，参与者接收到元数据请求后进行操作，操作完成后可以立即释放锁和发送应当。如果操作成功，返回 SUCCESS 应答，不需等待协调者的决定，如果操作失败，返回 ABORT 应答。协调者根据参与者应答的内容给客户端发送应答。可见，在只读操作中，不需要记录日志。

对于文件系统协议的各种操作，在大多数情况下只需由一个元数据服务器完成，少数涉及到两个元数据服务器，只有重命名（RENAME）操作最大可能涉及到四个元数据服务器，针对只涉及到一个和两个元数据服务器的操作，DCFS 元数据一致性协议进行了优化。对于由一个服务器完成的操作，其日志不须同步刷回磁盘，具体见 5.3.4.1 小节；对于需要由两个元数据服务器配合完成的操作，当参与者操作失败返回 ABORT 消息时，协调者不需发送 GLOBAL_ABORT 消息，并且当协调者进行恢复过程发现事务的日志记录为 ABORT 时也不需重发 GLOBAL_ABORT 消息。

5.3.4 元数据日志

5.3.4.1 元数据缓存与日志

元数据缓存与元数据一致性协议关系

图 5.2 给出元数据服务器上元数据一致性协议和缓存管理模块间的结构。为了减少记录日志读写磁盘次数，元数据服务器在日志缓存中记录日志。元数据一致性协议和缓存模块间的关系通常按照如下标准分类[Hae83]：（1）“脏”的元数据缓存在事务处理的过程中是否允许被写回磁盘，在[Hae83]中将其称为 fix/no-fix；（2）事务完成时被该事务修改的“脏”元数据缓存是否需要刷回磁盘，[Hae83]称为 flush/no-flush。根据上面的两个标准，元数据一致性协议和缓存模块间的关系通常可以分为四种：（1）fix/flush；（2）fix/no-flush；

（3）no-fix/flush；（4）no-fix/no-flush。

当元数据服务器上存在多个并发事务时，事务处理过程中如果允许被本事务修改的“脏”元数据刷回磁盘，将会增加缓存管理的复杂度，而事务处理完成时将被该事务修改的“脏”元数据缓存刷回磁盘，使元数据操作的处理时间增加，将严重影响元数据处理的性能，因此，DCFS 选择了 fix/no-flush 策略。

在 fix/no-flush 管理策略中，对于 DCFS 元数据事务处理的不同情况分别进行处理：

（1） ABORT。由于在事务处理的过程中不能将修改的缓存写到磁盘中，因此，在 ABORT 事务时，只需简单地释放使用的缓存，然后在日志中记录 ABORT；

（2） COMMIT。在 COMMIT 一个事务时，事务修改的缓存不必立即刷回磁盘，但可以被替换或由刷新线程写回磁盘；

（3） 重启。当故障节点重启后，如果事务需要进行 UNDO 处理，因为该事务修改的元数据缓存没有写回磁盘，因此不必清除磁盘；如果事务需要进行 REDO 处理，只需按照日志中内容进行 REDO 即可。

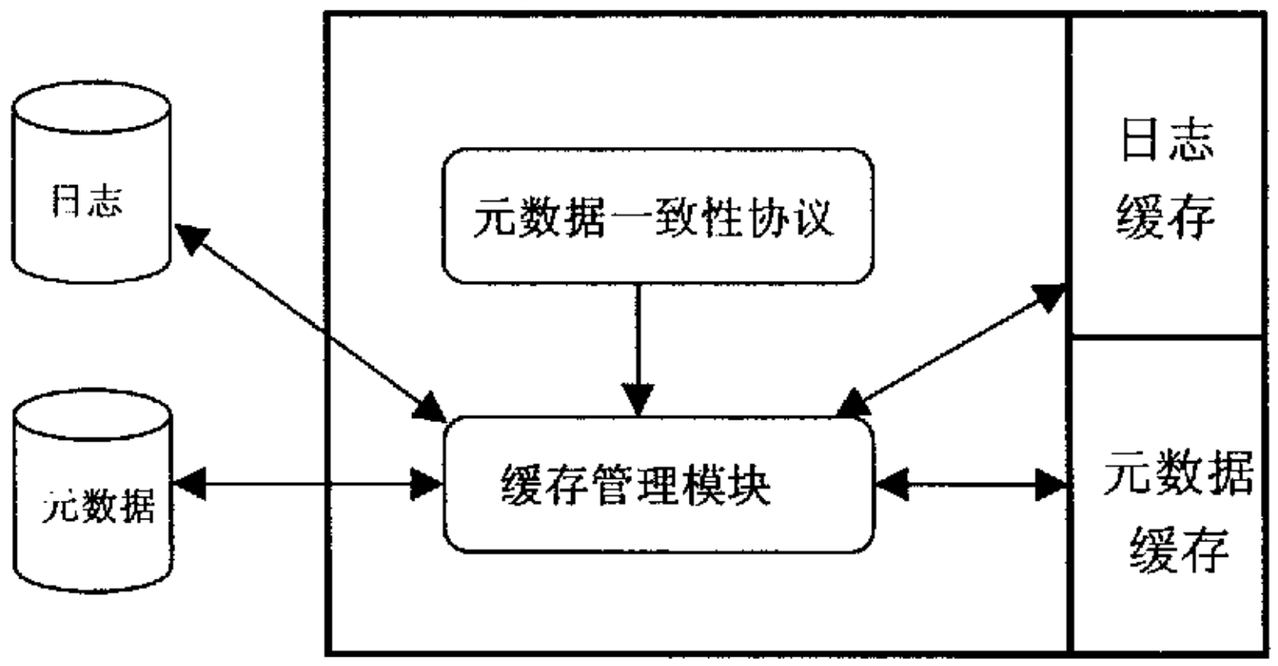


图 5.2 元数据一致性协议与缓存关系[Hae83]

WAL (Write Ahead Log)

DCFS 元数据缓存和日志缓存之间遵守 WAL（Write Ahead Log）协议[Ozs99]：

（1） 如果被修改的元数据缓存被写回磁盘，则与被修改的元数据缓存相关的事务的日志必须先被写回磁盘；

（2） 涉及多个元数据服务器的分布式事务的日志同步写回磁盘并且该事务修改的元数据缓存在事务处理过程中不能写回磁盘，保证了分布式事务在节点故障恢复过程中能得到该事务的状态信息，便于恢复。

同步日志与异步日志结合

DCFS 元数据操作可能由一个元数据服务器处理完成，也可能需要多个元数据服务器配合完成，因此，DCFS 事务可分为集中式事务，即只需一个元数据服务器参与的事务，以及分布式事务。DCFS 日志缓存根据事务的类型采用了不同的管理策略：

（1） 对于集中式事务，其日志记录在日志缓存中，由刷新线程或在日志缓存替换时异步写回磁盘，日志缓存和元数据缓存间遵守 WAL 协议；

（2） 对于分布式事务，因为两阶段提交协议的要求，其日志同步写回磁盘，保证磁盘中记录事务的状态以便恢复协议能够正确处理。

5.3.4.2 检查点

在事务处理系统中，检查点技术用于在缩短故障节点重启恢复的时间，检查点的生成通常包括三个步骤[Gra79]：（1）在日志中记录 BEGIN_CHECKPOINT；（2）激活系统其余组成部分进行检查点处理，如把“脏”缓存和日志缓存写入磁盘；（3）在日志中记录 END_CHECKPOINT。

DCFS 采用事务一致性检查点（Transaction-Consistent Checkpoint）技术[Gra79]，图 5.3（摘自[Gra79]）给出了该方法的示意图。当检查点启动时间到达时，DCFS 不再处理新到达的事务，检查点程序必须等待当前正在处理的事务完成才真正开始执行，检查点程序把日志缓存和事务修改的元数据缓存写回磁盘。当节点因为故障重启后，恢复协议只需处理日志中最新的检查点后发生的事务，在图 5.3 中，对于 T1、T2，恢复协议不需处理，而 T3 需要 REDO，T4 需要 UNDO。

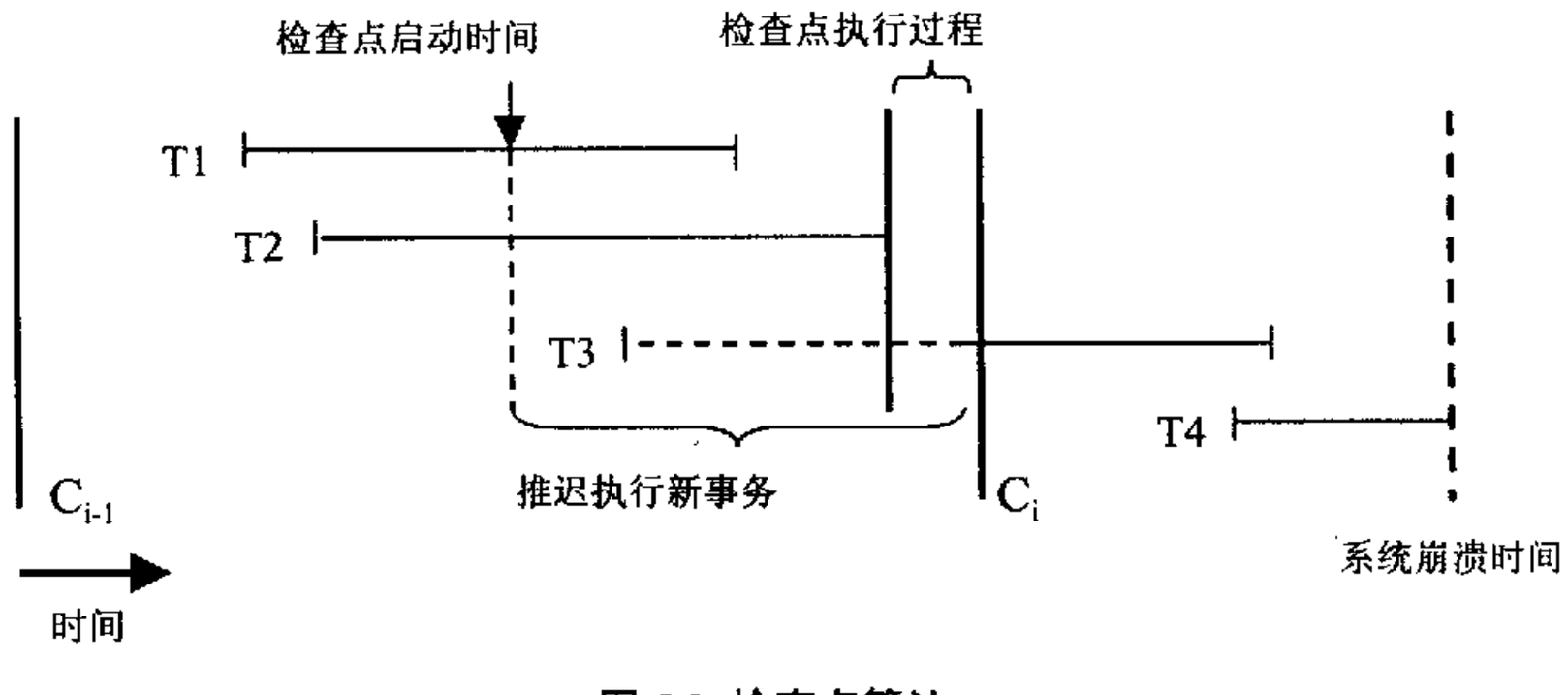


图 5.3 检查点算法

5.3.5 协议分析

作者在上面描述了 DCFS 为维护文件系统元数据的一致性的协议，可以看到，在元数据操作处理的过程中需要同步写入日志记录，多个元数据服务器间需要多次通信传递消息，这些都增加了元数据操作的处理开销，在本节中，作者对涉及到多个元数据服务器的操作的开销进行分析。在下面的分析中，我们仅对元数据操作的正常流程进行分析。

在 5.3.3.1 节对协议的描述中可以看到，对于修改文件系统元数据的操作，协调者需要写两次日志：（1）协调者本身对元数据进行的修改；（2）COMMIT 记录。这两个日志记录都需同步写到磁盘上。在元数据处理的过程中，协调者需要给参与者发送两次消息，元数据请求消息和 GLOBAL_COMMIT 决定消息。参与者在元数据请求处理的过程中需要写两次日志：（1）参与者对元数据进行的修改；（2）COMMIT 记录。其中（1）必须同步写到磁盘中。参与者只需发送一个 SUCCESS 消息。

从 5.3.3.4 小节对只读操作的优化可知，协调者不需要记录任何日志，并且协调者只需发送一次消息。参与者不需要记录日志并且只需给协调者发送一个 SUCCESS 消息。

表 5.1 中给出了元数据操作在涉及一个和多个元数据服务器的开销，每一项用三元组（*log*，*syn_log*，*msg*）表示，其中 *log* 表示写日志的次数，*syn_log* 表示同步写日志的次数，

msg 表示发送的消息数。当元数据操作涉及多个元数据服务器时，对于修改元数据的操作，协调者虽然需要写两次日志和发送两次消息，但是可以看到，最后一个消息发送不需要等待参与者的应答，可以直接给客户端发送应答，与使用标准两阶段提交协议实现的元数据一致性协议相比[Zha01]（由于 DCFS 在给客户端返回应答消息时须保证元数据修改完成，文[Zha01]中给出的基于标准 2PL 的元数据一致性协议须修改为协调者等待参与者对 GLOBAL_COMMIT 的应答，然后给客户端发送应答），本章中描述的协议少了一次消息往返的开销和一次写异步日志记录的开销；对于只读的元数据操作，由于 DCFS 对其处理过程进行了优化，不需记录任何日志。而当元数据操作只涉及一个服务器时，日志记录只需记录在缓存中。

表 5.1 协议开销分析

	协调者		参与者	
	Update	Read_only	Update	Read_only
多元数据服务器	2,2,2	0,0,1	2,1,1	0,0,1
一个元数据服务器	1,0,0	0,0,0	1,0,1	0,0,0

设文件系统元数据操作中更新操作的比例为 p_w，更新操作中涉及多个元数据服务器的比例为 p_d，t_{proto_d} 和 t_{proto_1} 分别表示涉及多个元数据服务器和一个元数据服务器时服务器的协议开销，而 t_{net}、t_r、t_{sl} 以及 t_{al} 分别表示一次消息来回的开销、只读元数据操作的平均时间开销和同步、异步日志的开销，则元数据操作在服务器上的平均执行时间可以表示为：

$$t = p_w.(p_d.(t_{proto\_d} + t_{net} + 3t_{sl}) + (1 - p_d).(t_{proto\_1} + t_{al})) + (1 - p_w).t_r \quad \text{（5.1 式）}$$

整理得，

$$t = p_w.p_d.(3t_{sl} + (1 - p_d)t_{al}) + t_{no\_trans} \quad \text{（5.2 式）}$$

其中 t_{no_trans} 表示不实现一致性协议时的元数据操作开销，由于异步写日志的开销很小，可以忽略不计，在 1.2 节中我们给出了元数据只读和更新操作的比例[Rob99]，取其中更新操作所占比例最大的一组数据，代入（5.2）式可以化简为 $t = p_d.t_{sl}/7 + t_{no\_trans}$，可以看到，实现元数据一致性开销至多增加的开销为 $t_{sl}/7$，而在实际运行环境中，当我们采用合适粒度的元数据分配策略时，涉及多个元数据服务器的操作所占的比例很小[Ji00]（根据论文[Ji00]中给出的数据推算，在更新操作中涉及多个元数据操作的比例为 0.3%），即 p_d 很小，因此实现元数据一致性协议所增加的平均开销是可以接受的。

5.4 小结

作者在本章描述了维护 DCFS 元数据一致性需要处理的故障类型，结合分布式事务处理的技术和 DCFS 元数据服务器的特点设计了 DCFS 元数据一致性协议，该协议包括正常操作处理协议、终止协议和恢复协议，给出了处理元数据缓存和一致性协议间关系的技术，并对 DCFS 元数据一致性技术进行了分析，分析表明对于 DCFS 文件系统，为实现元数据一致性协议而增加的开销是可以接受的。

第六章 DCFS 文件系统实现与性能评价

DCFS 文件系统是为曙光系列超级服务器设计的机群文件系统，本章作者将介绍曙光4000L 超级服务器硬件和软件结构，描述 DCFS 文件系统协议层次及相应的实现，并给出了 DCFS 在曙光 4000L 上读写带宽和元数据吞吐率性能测试的结果，与 PVFS 的比较表明，DCFS 文件系统的读写性能和元数据吞吐率性能优于 PVFS，说明 DCFS 采用的体系结构和服务器上的优化是有效的。

6.1 曙光超级服务器

本节将对本论文工作所基于的系统平台：曙光 4000L 超级服务器[Sun03]做简单的介绍。曙光系列超级服务器是由科学院计算所智能中心研制的基于分布式存储和消息传递的高端机群，是通用的可扩展超级服务器系统，其最新成果是曙光 4000L 超级服务器。该项目由中科院计算所知识创新工程项目支持，是计算所承担的中科院知识创新工程重大任务，得到中科院知识创新工程经费的支持，同时也得到十五 863 计划“高性能计算机及其核心软件”专项重大项目“面向网格的超级服务器——曙光 4000”经费的支持。

曙光 4000L 的总体目标是研制每秒 3 万亿次浮点峰值运算能力的 Linux 超级服务器，以支持数据密集应用为主，针对应用特点对系统进行优化设计，解决关键技术问题，实现应用与系统的一体化优化。同时，能支持科学与工程计算，网络与信息服务，事务处理多种应用。曙光 4000L 与一般的机群系统相比，不再以峰值速度为研制目标，而将重点放在特定应用的实际性能，研究关键技术以使系统的硬件性能得到最大限度的发挥，为国民经济重要行业提供“真正”高性能价格比的高性能计算系统。曙光 4000L 超级服务器的主要技术指标如表 6.1 所示：

表 6.1 曙光 4000L 超级服务器主要技术指标

	参数
峰值计算速度	每秒 3 万亿次浮点运算
系统规模	40 个机柜，322 个机架式节点，644 个 CPU
内存	644GB
外部存储	1344 块数据硬盘，100TB 存储，20 个 DLT 磁带
网络	主干、管理、计算、监控 4 套网络，256Gb/s 主干网 32Gb/s 光纤接入

6.1.1 体系结构

如图 6.1 所示，曙光 4000L 采用机群结构，系统中节点根据功能划分为计算节点、数据库节点、服务节点三种类型，每个类型节点构成一个节点池，并根据功能优化节点的配置与连接。系统有四套网络用于系统内的节点互连，包括主干网、计算网、管理网、监控

网。其中，主干网提供接入、机群内部数据传输、数据库通信、文件传输功能，根据数据密集应用的需求，网络能力在不同节点池间是不均匀分布的；计算网提供高性能的消息传递，拓扑结构可实现最大的网络对分带宽；管理网用于机群内部管理和与外部网络的连接；监控网用于实现对系统中主要硬件状态的实时监控和操作系统的维护，采集和传送系统中各种被监控的状态信息，是硬件和操作系统的管理网络。

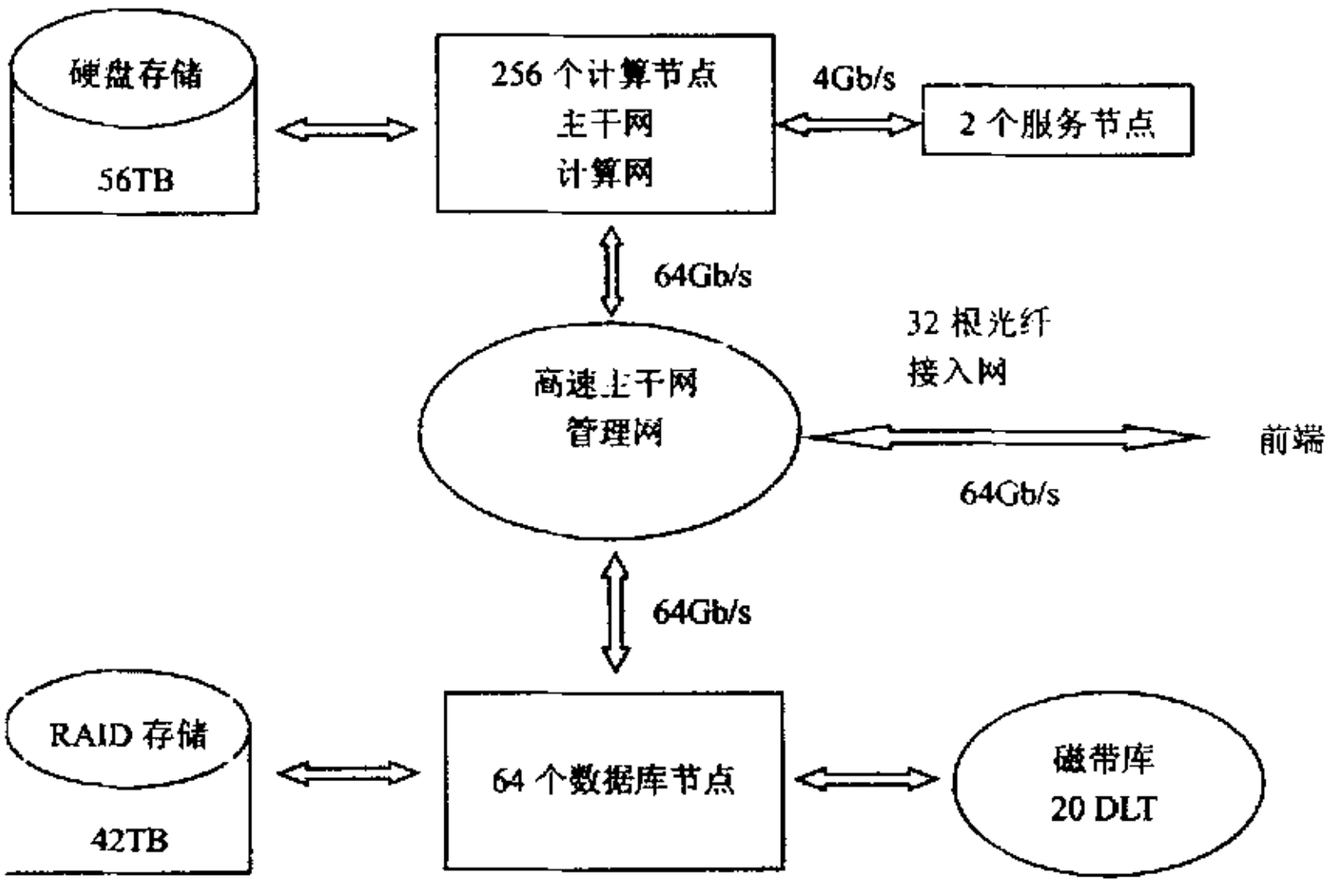


图 6.1 曙光 4000L 体系结构

曙光 4000L 的基本硬件结构共包括三类结点：（1）2 个服务节点，用于运行系统软件中机群操作系统的一些系统服务、存放系统映像、存放系统服务的数据库、联机用户的登录（login）等操作。（2）256 个计算结点，用于运行大规模并行计算应用程序；（3）64 个数据库结点，用于并行数据库应用，系统为其提供高可用软件，与 RAID 存储一起实现数据库的高可用服务。

曙光 4000L 的存储系统包括运行数据库的 RAID 存储、运行文件服务的硬盘存储和用于备份服务的磁带库。内存、RAID、硬盘和磁带库都是可扩展的，以满足用户未来数据量增长的需要。

曙光 4000L 系统共有 4 套网络，分别是主干网，管理网，计算网和监控网。其中：（1）主干网是数据服务的网络，包括 Internet 数据的宽带接入（来自前端）、数据加载、并行数据库服务、数据库查询、文件服务和并行计算服务，它是系统的核心和最关键设备。系统中的计算节点也可使用主干网进行并行计算；（2）计算网是并行计算的网络，提供 TCP/IP 通信和用户空间 PVM/MPI 通信两种并行计算方式。计算网采用 Myrinet 2000 交换机和网卡，单向单通道 2Gb/s 带宽，交换机数据交换带宽 512Gb/s。64 个需要高性能通信支持的并行计算的节点连接在计算网上；（3）管理网是系统管理的网络，有关操作系统的监控信息和系统管理员的配置、管理、操作都是通过管理网络进行，用户可通过管理网登陆到系统中任何一个节点；（4）监控网由每个节点内置的监控卡、Terminal-Switch 网络、相应的环境采集传感器、数据采集卡和硬件监控节点组成，监控网以串行的方式连接所有节点，提供硬件监控和串口控制两大功能。

曙光 4000L 的监控系统分成三个层次，分别是节点监控、系统监控和网格监控，共 7 种监控方式。

6.1.2 软件结构

曙光 4000L 超级服务器基本系统软件配置如下：

表 6.2 曙光 4000L 软件配置

	参数
操作系统	Redhat Linux7.3 （kernel 2.4.18）
编译器	gnu/g77，Intel 编译器，Java1.2，tcl/tk，Perl
库	Intel MKL
数据库	Oracle 8.1.7
工具	Linux 软件工具

其系统软件共包括以下几个部分：

- ✧ DCOS 机群操作系统，包括机群构件平台 Clustone、RMS 资源作业子系统、DCMS 系统管理子系统、DCIS 系统安装子系统、DCMM 硬件监控子系统、MultiTerm 并行命令子系统等。
- ✧ 并行编程环境，包括 BCL-4/IP 通信子系统和 PVM/MPI 并行计算子系统。
- ✧ DCFS 机群文件系统
- ✧ 高可用软件
- ✧ Spreader 智能文件浏览器
- ✧ GridView 网格监控中心
- ✧ TapeShare 远程磁带共享
- ✧ ServerScope 数据密集应用性能评价软件，包括数据库评价软件、高可用评价软件、Internet 数据处理性能评价软件。

6.2 DCFS 文件协议

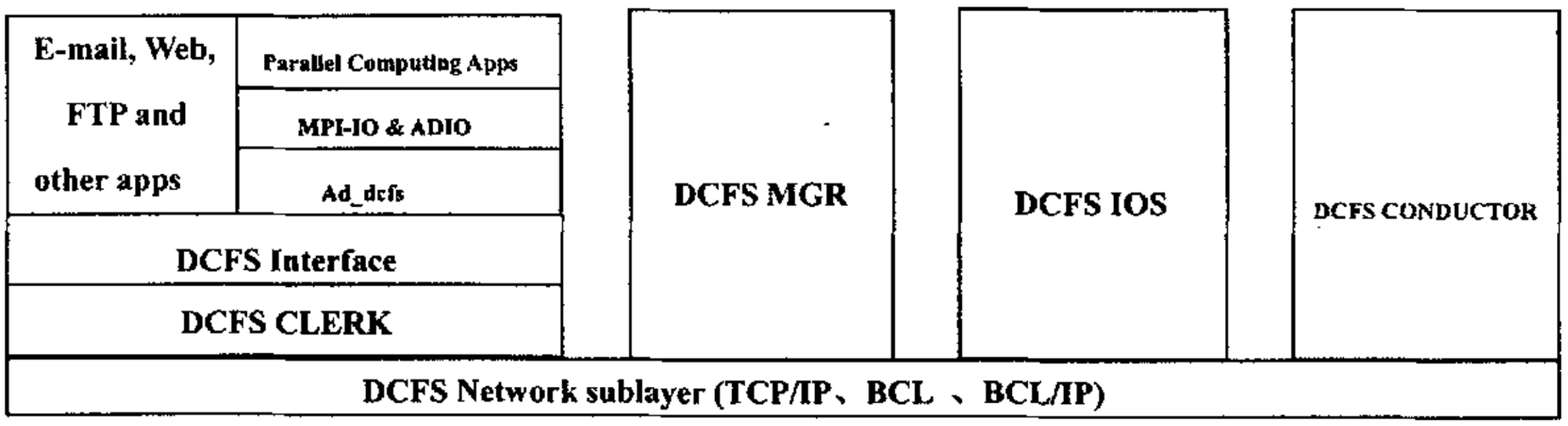


图 6.2 DCFS 文件系统协议图

图 6.2 给出了 DCFS 文件系统协议的结构示意图，从图中我们看到，DCFS 文件系统协议主要由如下部分构成：（1）应用层。Linux 环境下的应用程序，包括 E-mail，Web 服务等都可以直接在 DCFS 文件系统上运行，另外，并行程序除了可以直接在 DCFS 上运行外，还可以通过 MPI-IO 接口来使用 DCFS 文件系统；（2）DCFS 文件系统协议接口，DCFS 文件系统提供标准的 POSIX[ISO96]接口，同时在 POSIX 接口的基础上增加了对 MPI-IO 支持；

（3）DCFS 文件系统协议主体，包括客户端进程（CLERK），元数据服务器（MGR）和存储服务器（IOS）；（4）DCFS 管理协议模块，包括配置管理工具和 HA 的监测和恢复；（5）DCFS 通讯层，在网络层中，我们屏蔽了底层网络协议的细节，给上层的模块提供统一的一套接口，我们提供了对 TCP/IP、BCL 和 BCL/IP 支持。

6.2.1 应用层

DCFS 提供两类应用程序的支持：（1）Linux 环境中使用 POSIX 文件系统调用接口的程序。由于 DCFS 在文件协议接口层通过实现 VFS 层中定义的一组操作，提供了标准的 POSIX[ISO96]接口，Linux 环境中的应用程序不需重新编译就可以直接运行；（2）并行程序可以通过 MPI-IO 接口使用 DCFS 文件系统[He02-2]。

6.2.2 文件协议接口层

DCFS 文件系统接口是通过加载核心 VFS 模块的方式实现。在 Linux 平台上，核心 VFS 文件系统接口由大约 40 个用户可直接使用的文件系统调用组成[ISO96]，它们的操作对象可以划分为三大类：文件系统整体、普通文件、目录文件与符号链接。对于普通文件、目录文件及符号链接的操作又可以分为对数据（普通文件中的数据、目录文件中的目录项及符号链接的值）的存取与元数据（包括长度、日期、所有者等信息）的存取。

这些系统调用可以通过 VFS 层定义的对文件、inode、文件系统超级块及文件系统配额管理上的四组标准接口函数表来实现。DCFS 通过实现这些标准 VFS 函数或者标准中的一部分来对支持普通应用。这些标准 VFS 接口函数表中分别是（具体定义请参考 Linux 核心代码）：

- ✧ 文件操作接口函数表：包括文件的打开/关闭、数据读/写、文件指针移动、文件锁、文件数据就绪状态查询、文件的内存映射、文件缓存操作等。
- ✧ inode 操作接口函数表：除包括以上的文件子过程以外还需要实现 inode 的创建、建立/删除链接、建立符号链接、建立/删除目录、重命名、截短长度等。
- ✧ 文件系统超级块接口函数表：包括读/写/删除 inode、安装/卸载文件系统、取文件系统状态信息等。
- ✧ 文件系统配额接口函数表：包括分配/释放 block 与 inode、初始化。

DCFS 并不支持所有的 Linux 文件系统调用，但它所支持的调用应该可以满足一般应用对文件系统的需求。DCFS 支持文件与文件系统调用如下：

- ✧ 普通文件系统调用：dup、dup2、lseek、access、creat、open、close、read、write、readv、pread、writev、pwrite、stat、fstat、lstat、newstat、newlstat、newfstat、truncate、ftruncate、chmod、fchmod、utime、utimes、chown、lchown、fchown、sync、fsync、fdatasync 与 fcntl（文件锁部分）。
- ✧ 目录文件调用：mkdir、rmdir、getcwd、getdents、chdir、fchdir、mknod 及 rename。
- ✧ 符号链接/硬链接调用：readlink、link、unlink 与 symlink。
- ✧ 文件系统调用：statfs 与 fstatfs。

DCFS 可以基于 POSIX 文件接口提供 MPI-IO 的支持。MPI-IO[Tha96][Tha99]应用程序编程接口是消息传递编程界面 MPI-2 标准的一部分，其主要目的是为了优化对单个共享文

件的并行存取性能（协调 I/O 请求或者通过优化的合作式缓存）。MPI-IO 是对 MPI 的全面扩充，它允许程序员将信息放入到接口层次上，继承了 MPI 的派生数据类型来描述文件访问模式，另外它还引入了集合 I/O 的概念。

MPI-IO 与传统文件系统的关系可以用图 6.3 来表示，MPI-IO 中定义了一个抽象数据 I/O 接口 ADIO[Tha96][Tha99]，它与物理文件系统的关系类似于 MPI 编程接口中的抽象数据接口 ADI 与底层通信协议如 TCP/Socket、共享内存之间的关系，即 ADIO 提供一组标准界面及相关的语义说明，当把 MPI-IO 移植到某类特殊文件系统时，用该文件系统的系统调用来实现这组界面。

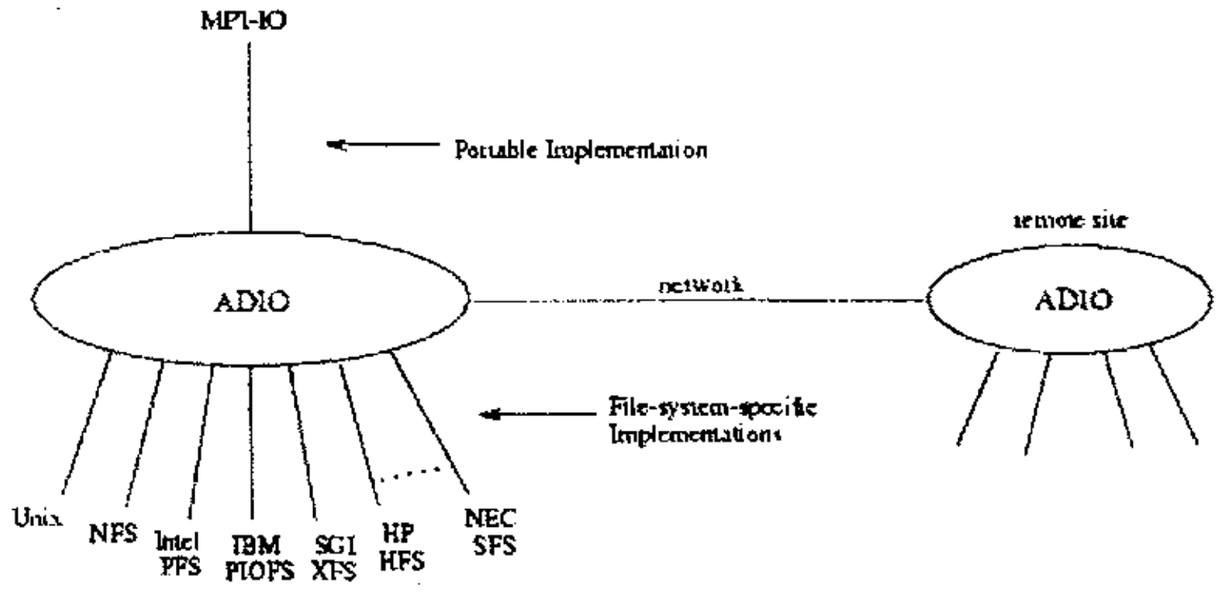


图 6.3　ADIO 示意图[Tha96]

6.2.3 DCFS 文件协议主体

DCFS 文件系统协议主体由客户端模块、元数据服务器和存储服务器构成。文件系统客户端模块如图 6.4 所示，它由 DCFS 的 VFS 核心子模块（Broker）与用户级 DCFS 客户进程（Clerk）构成。如 6.2.2 所述，VFS 核心子模块（Broker）实现了操作系统 VFS 层中定义的一组操作接口；为了方便的支持 Linux 和 AIX 等操作系统，DCFS 在客户端使用了用户级的进程 Clerk 负责向服务器转发核心模块中的请求，Broker 和 Clerk 间通过消息队列和管道传递消息，使用映射到 Clerk 用户空间的共享内存传递数据。

DCFS 元数据服务器负责维护文件 inode、目录文件等元数据并处理相关的元数据请求。DCFS 采用了多元数据服务器的结构，使用了可调粒度元数据分配机制将元数据分布到各个元数据服务器上。DCFS 元数据服务器是一个用户级多线程程序，它 DCFS 元数据协议服务线程 FPST 和元数据缓存刷新线程 MCFT 构成。文件协议服务线程 FPST 负责处理相关的 DCFS 文件元数据、目录文件以及元数据缓存管理，元数据缓存刷新线程 MCFT 负责把元数据缓存中“脏”的元数据刷回磁盘。

DCFS 存储服务器采用了网络存储分组技术，并可以调整分条（stripe）的粒度。存储服务器也是一个用户级多线程程序，它由主控制线程 CT、I/O 调度线程 IOST、I/O 执行线程池 IOXTP 和刷新线程 IOFT 构成。其中 I/O 调度线程 IOST 负责处理来自 DCFS 客户端的读写数据请求并指派对应 I/O 执行线程进行具体操作。I/O 执行线程负责处理存储服务器上的各种请求和维护服务器端的文件数据缓存。

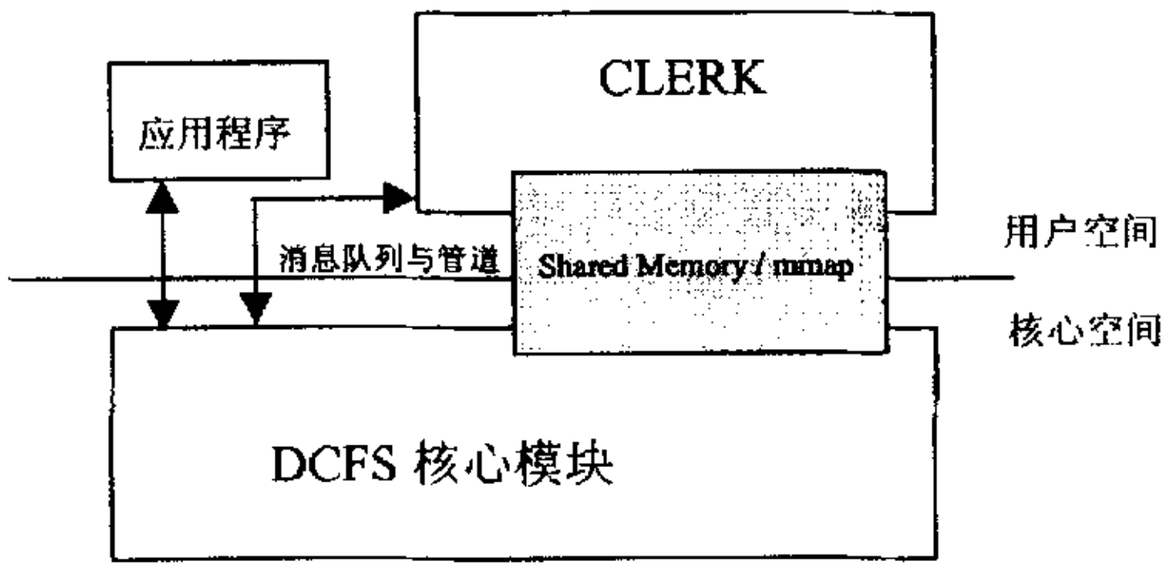


图 6.4　DCFS 客户结构示意图

6.2.4 DCFS 管理协议模块

DCFS 的配置管理模块[He01]由配置管理节点进程以及客户节点、元数据服务器节点与存储服务器节点上的配置管理协议处理部分组成。配置管理进程由配置管理线程、状态监测与恢复模块与通信模块三个部分构成。为了保证该模块本身的高可用，该模块由运行在不同节点上的主进程与备份进程组成。

在目前的设计与实现中，配置管理工具可完成的任务如下：（1）启动/停止 DCFS 配置管理主/从服务器；（2）启动/停止 DCFS 元数据服务器与存储服务器；（3）启动/停止 DCFS 客户进程与核心模块；（4）以批方式安装 DCFS 文件系统卷；（5）动态增加 DCFS 文件协议层组件（包括客户进程及核心模块、元数据服务器与存储服务器）；（6）动态扩充存储服务器 I/O 子系统容量。

6.2.5 DCFS 通信协议层

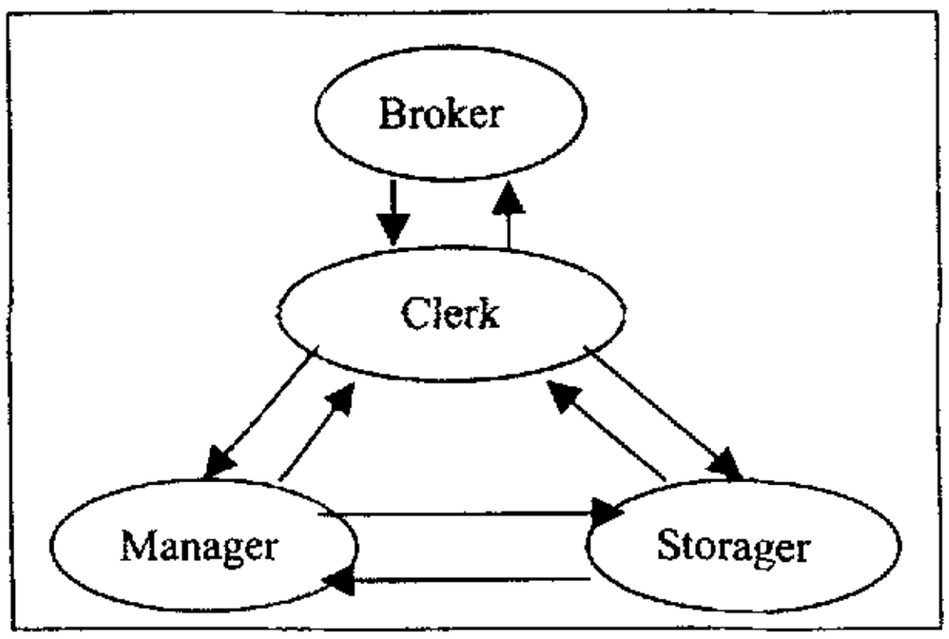


图 6.5　DCFS 中的消息传递通路示意图[He01]

高速互连网络的发展使机群文件系统可以获得更高的性能[He02-2]。目前机群系统中使用了 Gigabit Ethernet，Myrinet, SCI 等高速互连网络，提供给上层软件的通信协议包括 TCP/IP、GM、BCL、VIA 等，但不同的网络协议提供给上层软件接口和使用方式不尽相同，另外，DCFS 文件系统各组成部分间可以使用不同的通信协议传递消息，因此，DCFS 增加了通信协议层，向上层的模块提供统一的使用接口而屏蔽掉底层的网络协议细节。这

些接口包括基本通信对象的创建与管理、消息收发以及异常处理。

DCFS 通信协议层[He01]主要针对 DCFS 各组成部分之间的两类通信需求：(1) 配置管理与高可用子系统部件间的通信，它们使用标准的 UDP 协议进行通信；(2)文件系统主体（核心模块、clerk、存储服务器和元数据服务器）间的消息传递。图 6.5 给出了 DCFS 中核心模块 Broker、clerk、存储服务器和元数据服务器间消息传递通路。在当前版本中，DCFS 文件系统 Clerk、存储服务器和元数据服务器之间使用 TCP 或 BCL 协议进行通信；客户端核心模块 Broker 与 Clerk 进程使用 UNIX 标准的管道文件、消息队列以及 IOCTL 系统调用进行消息传递，我们称之为 jaywalker 协议。无论是否使能 DCFS 的私有数据/元数据缓存，Broker 与 Clerk 都通过核心到用户进程空间的内存映射进行大块数据的交换，以避免不必要的内存拷贝。

DCFS 将通信的基本对象称为通信服务端口，对应各种协议，可以划分为以下几种具体情况：

- ✧ 当使用 TCP 或 UDP 协议时，对应的服务端口创建方所创建的 TCP 或 UDP 服务端口就是 DCFS 通信服务端口；而对于其它进程与 TCP 端口建立的连接，DCFS 也称之为通信服务端口，但是它是本进程内部的远程通信端口；而对应需要向远程进程的 UDP 端口发送或接收消息的 DCFS 对象，其远程通信端口也只是一个 socket;
- ✧ 使用 BCL 协议时，由于通信双方都需要拥有对应的 BCL 端口才能通信，因此这种 BCL 端口与 DCFS 的通信服务端口概念是相一致的，此时远程通信端口仅仅是一个 BCL 端口号而已；
- ✧ 对于 Broker 与 Clerk 之间的通信，我们将 Broker 中的消息队列以及 Clerk 中的管道文件称之为 Broker 和 Clerk 各自的 DCFS 通信服务端口。

通过以上分类，我们就可以得到一个完整的、统一的 DCFS 消息传递界面。

6.3 DCFS 读写性能测试

6.3.1 负载选择

我们选择 iozone[Ioz03]作为每个客户节点上运行的测试程序，iozone 是由惠普公司设计开发的测试文件系统读写带宽的测试程序，被广泛地用于文件系统性能测试中。为了同步多个客户节点的测试进程，类似于[Jon00]，我们采用了 MPICH 的 barrier 机制来达到程序同时运行的目的：选择一个客户节点作为测试的主节点，其余客户节点称为从节点，主节点接收用户的测试命令，然后将命令分发到每个从客户节点，所有的测试进程在 barrier 处等待然后同时执行，每个测试进程在测试结束后将测试结果返回主节点。

我们将服务器读写字节数定义为负载规模，在读写性能测试中，我们定义两种负载：（1）读写规模随客户进程数变化的负载，我们称为第一种负载模式；（2）固定读写规模负载，我们称为第二种负载模式。由于在科学计算类程序中，顺序读写占了很大比重，所以在本节读写带宽测试中，每个测试进程的读写类型均为顺序读写。在读写规模随客户进程数变化的负载中，每个测试进程读写相同大小的文件，随着测试进程数的增加，每个存储服务器读写的数据量随着增加。在固定读写规模负载中，服务器读写的字节总量不随客

户进程数的增加而变化，每个测试进程读写文件的字节数为 S/N_c，其中，S 为服务器读写总量，N_c 为客户进程数。在本节的读写带宽测试中，客户端测试进程每次读写的记录大小为 1MB。

6.3.2 读写性能评价参数定义

设系统中客户进程数为 n_c，每个客户进程读写文件的字节数为 s，带宽 $B_i = s_i / t_i$，聚合带宽为 $B_{n_c} = \sum_{i=1}^{n_c} B_i$。基于聚合带宽，我们定义如下参数：（1）最高带宽，对于固定数目的存储服务器配置，文件系统获得的最高带宽值；（2）服务器加速比，文件系统最高带宽与一个存储服务器配置时最高带宽的比值，我们用该值表示服务器端的可扩展性，该值越大，表示存储服务器的可扩展性越好。

6.3.3 测试平台

本章中的测试是在曙光 4000L 超级服务器[Sun03]中通过一个千兆交换机连接起来 32 节点的平台中进行的,平台配置如下表 6.3。在本章测试中，DCFS 和 PVFS v1.5.4 (简称 PVFS) 均使用 TCP/IP 协议通信，我们使用 netperf[Net03]测得节点间网络带宽为 106.2MB/sec。

表 6.3 测试平台配置

	型号或性能指标
节点数	32
每节点处理器数	2
处理器类型	Intel Pentium III Coppermine
处理器主频	1GHz
结点内存	2GB
磁盘	Seagate Ultra320，73GB
以太网卡类型	Intel® PRO/1000 Network Connection
操作系统	Linux RedHat 7.2 (2.4.18 核心)

6.3.4 DCFS 读写性能分析

图 6.6 和 6.7 给出了 DCFS 在第一种测试模式得小文件测试的读写性能。表 6.4 给出了相应的性能参数。从中可以看到，DCFS 在一个存储服务器配置时最高聚合读写带宽分别为 86.10MB/s 和 100.66MB/s，网络带宽利用率达到了 81.1%和 94.8%。DCFS 的聚合读写带宽随着存储服务器的增加而增加，我们注意到，DCFS 加速比在 2、4、8 个存储服务器时分别为 1.703、2.454、4.297，我们认为 DCFS 的服务器可扩展性是可以接受的，因为在小文件测试中，读写请求的响应时间主要由网络开销和文件协议的开销构成，服务器负载较轻，存储服务器的增加不能使读写带宽线性增长。在每种服务器配置的测试中，聚合带宽随着客户节点数的增加而达到最高聚合带宽，而且聚合带宽在最大客户数时没有下降。图 6.8 至 6.9 给出了 DCFS 在第二种负载模式的小文件测试中的读写性能。表 6.5 给出了相

应的性能参数。其结果与第一种负载模式小文件的测试结果相似，可以看到 DCFS 具有较好的读写性能和客户端及服务器的可扩展性。

表 6.4 测试模式 1 小文件测试性能参数

Num. Of IOSes	Read		Write	
	Max Bandwidth (MB/sec)	*Speedup*	*Max Bandwidth (MB/sec)*	*Speedup*
1	86.828	1.000	100.663	1.000
2	147.937	1.703	175.145	1.740
4	213.037	2.454	212.205	2.108
8	373.135	4.297	331.043	3.289

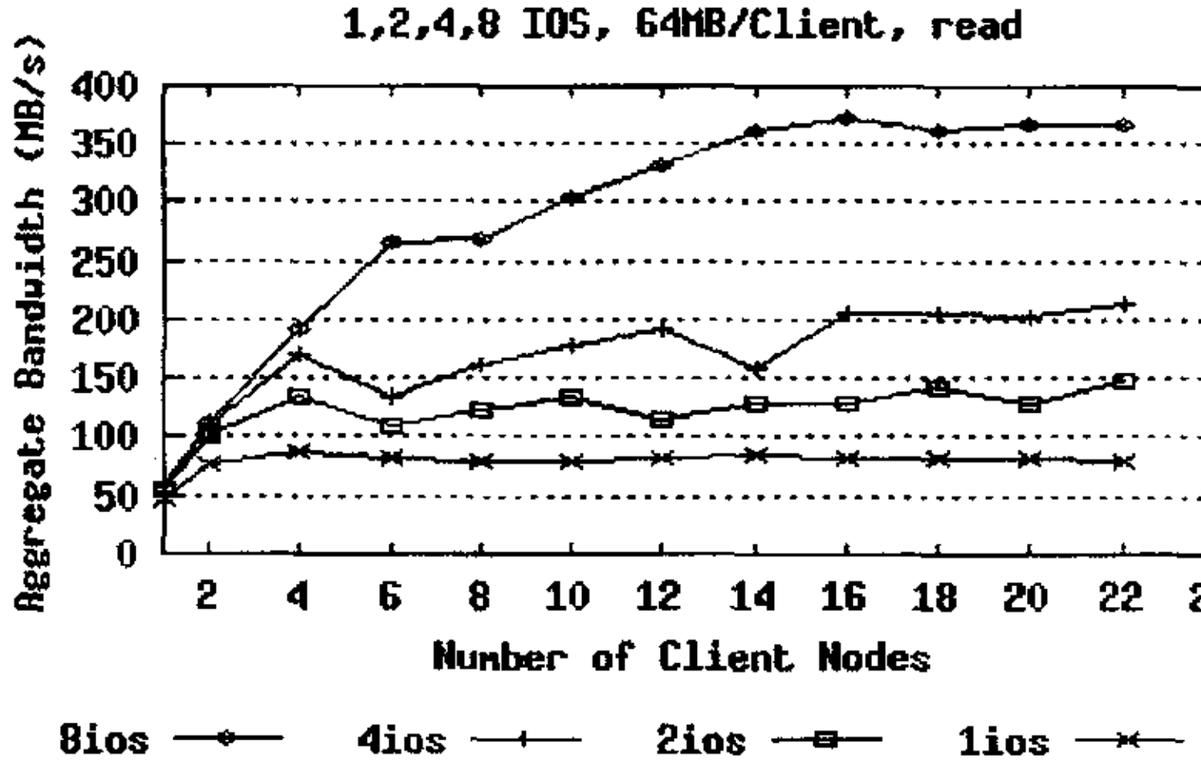


图 6.6 测试模式 1 小文件读聚合带宽

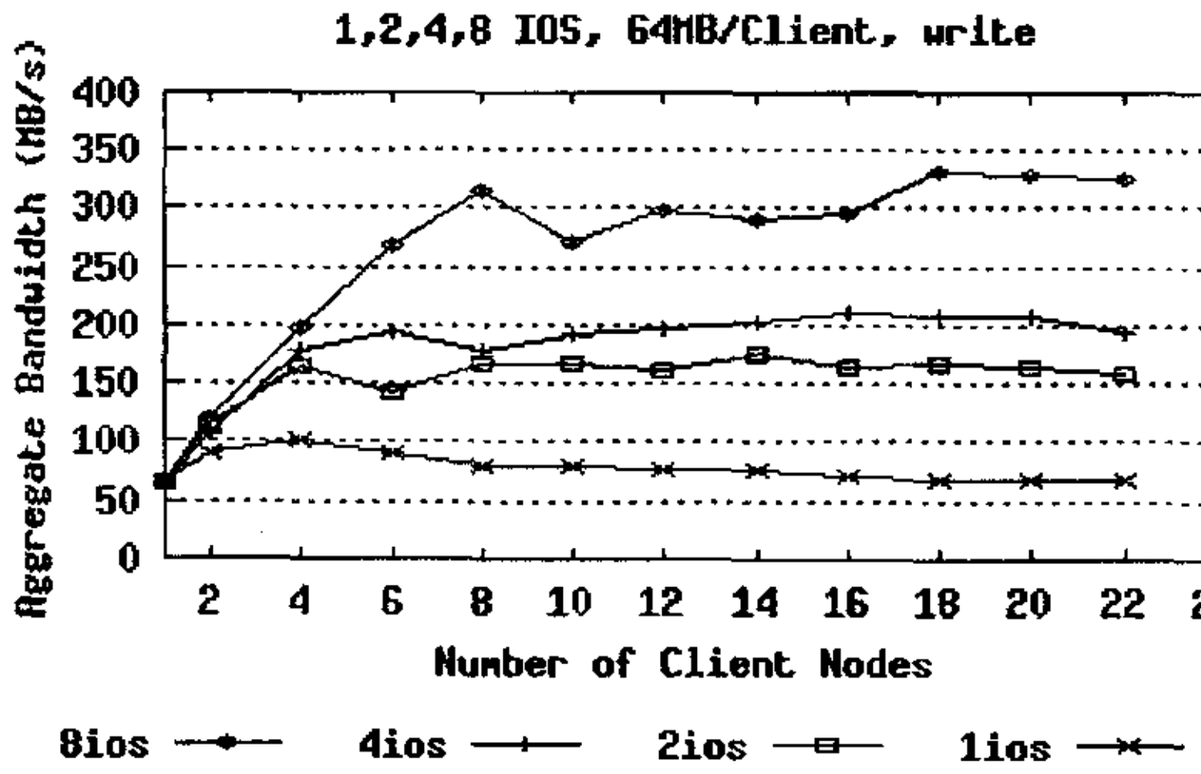


图 6.7 测试模式 1 小文件写聚合带宽

表 6.5 测试模式 2 小文件测试性能参数

Num. Of IOSes	Read		Write	
	Max Bandwidth (MB/sec)	*Speedup*	*Max Bandwidth (MB/sec)*	*Speedup*
1	90.054	1.000	103.594	1.000
2	141.378	1.570	191.198	1.846
4	205.922	2.287	206.504	1.993
8	384.295	4.267	319.717	3.086

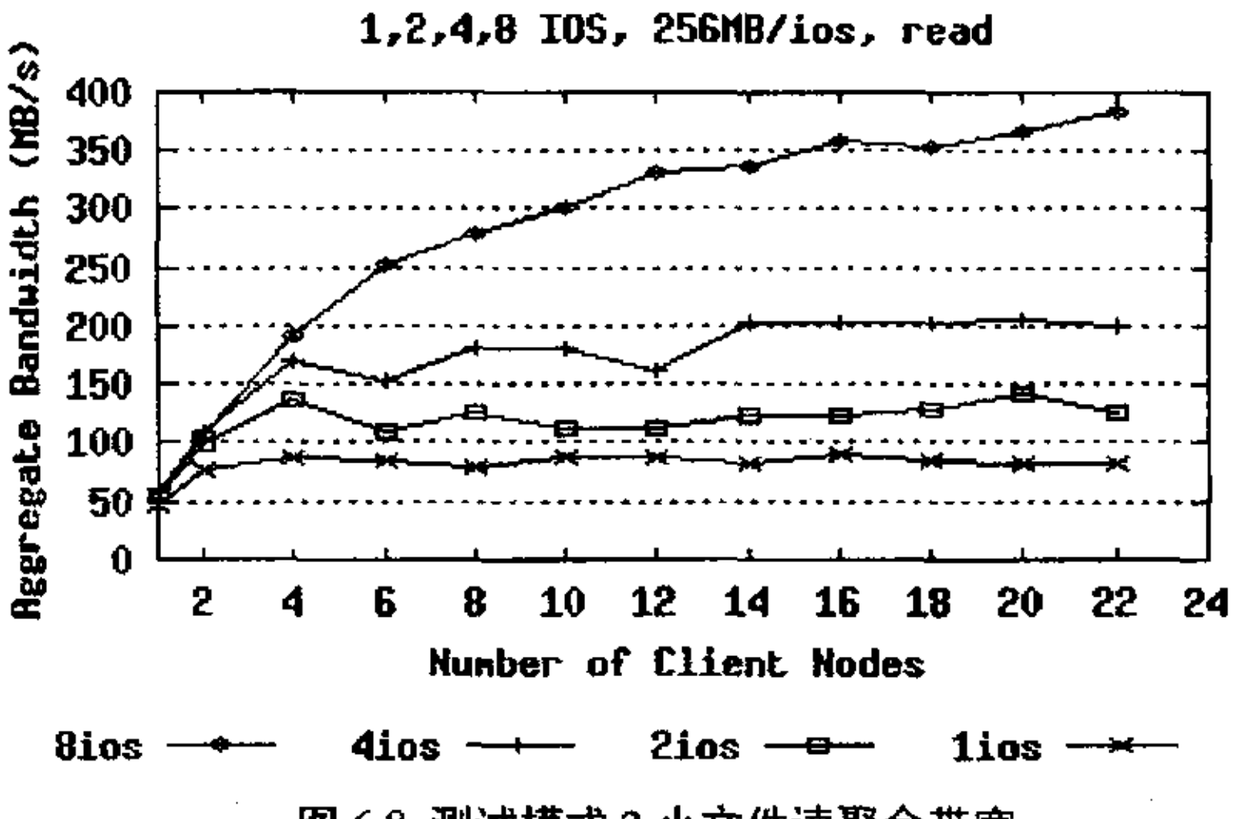


图 6.8 测试模式 2 小文件读聚合带宽

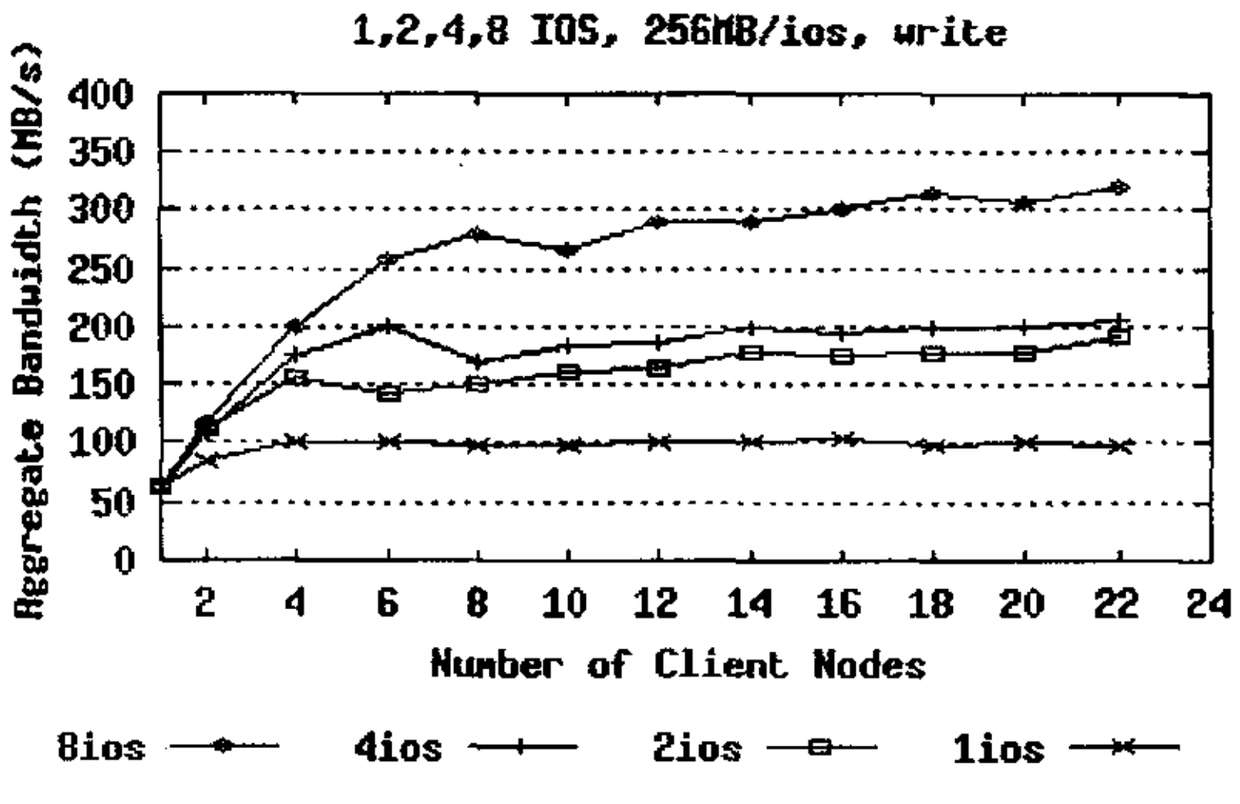


图 6.9 测试模式 2 小文件写聚合带宽

图 6.10 至 6.11 给出了 DCFS 在第一种测试模式下大文件测试中的读写性能，每个测试进程读写不同的文件，文件大小为 1024MB。表 6.6 给出了相应的性能参数。从第 6.3.1 节测试模式的定义可知，每个存储服务器上读写的字节数为 $1024*N_c/N_s$，其中 N_c、N_s 分别为客户节点数和存储服务器数，当客户节点数较小时，存储服务器的数据保存在内存中（Linux 系统核心将文件的数据存放在 buffer cache 中），随着客户节点数的增加，每个服务器读写的字节数将超过 buffer cache 的容量，服务器上的开销以读写磁盘为主，所以该测试反映了

DCFS 读写操作在以网络开销为主到读写磁盘为主过程中的性能变化。对于写带宽，随着客户节点数的增加很快达到饱和。DCFS 写聚合带宽在到达最高带宽后维持不变，对应于 2、4、8 个存储服务器，服务器加速比分别为 1.849、3.223 和 4.745，说明 DCFS 在大文件测试中也具有较好的客户端可扩展性和服务器端可扩展性。对于读带宽，我们注意到，DCFS 在到达最高带宽后逐渐下降，然后稳定在某个带宽水平上，这是因为随着服务器读写字节数的增加，buffer cache 不能缓存所有文件的数据，需要从磁盘中读出数据，表 6.7 给出了存储服务器上本地文件系统测试的结果，为了模拟存储服务器的行为，对应 N_{ios} 个服务器和 N_c 个客户端，读写的数据总量为 $1024MB*N_c/N_{ios}$，每个读操作的数据量与 DCFS 存储服务器上缓存块大小相同，读带宽的单位为 MB/s。从表中可以看到，随着数据量的增大，本地文件系统读性能逐渐下降，并影响 DCFS 读性能，以一个存储服务器的配置为例，当读写的数据总量大于等于服务器物理内存容量（2GB）时，本地文件系统的读性能下降，并稳定在一定的带宽水平上，DCFS 的读聚合带宽表现出相同的趋势，因此，存储服务器上本地文件系统性能下降是造成 DCFS 读聚合带宽的下降的原因。DCFS 读聚合带宽对应于 2、4、8 个存储服务器，服务器加速比分别为 2.134、3.529、7.316，说明 DCFS 的读带宽的可扩展性较好。

表 6.6 测试模式 1 大文件测试性能参数

Num. Of IOSes	Read		Write	
	Max Bandwidth (MB/sec)	*Speedup*	*Max Bandwidth (MB/sec)*	*Speedup*
1	44.472	1.000	63.654	1.000
2	94.891	2.134	117.698	1.849
4	156.945	3.529	205.166	3.223
8	325.366	7.316	302.038	4.745

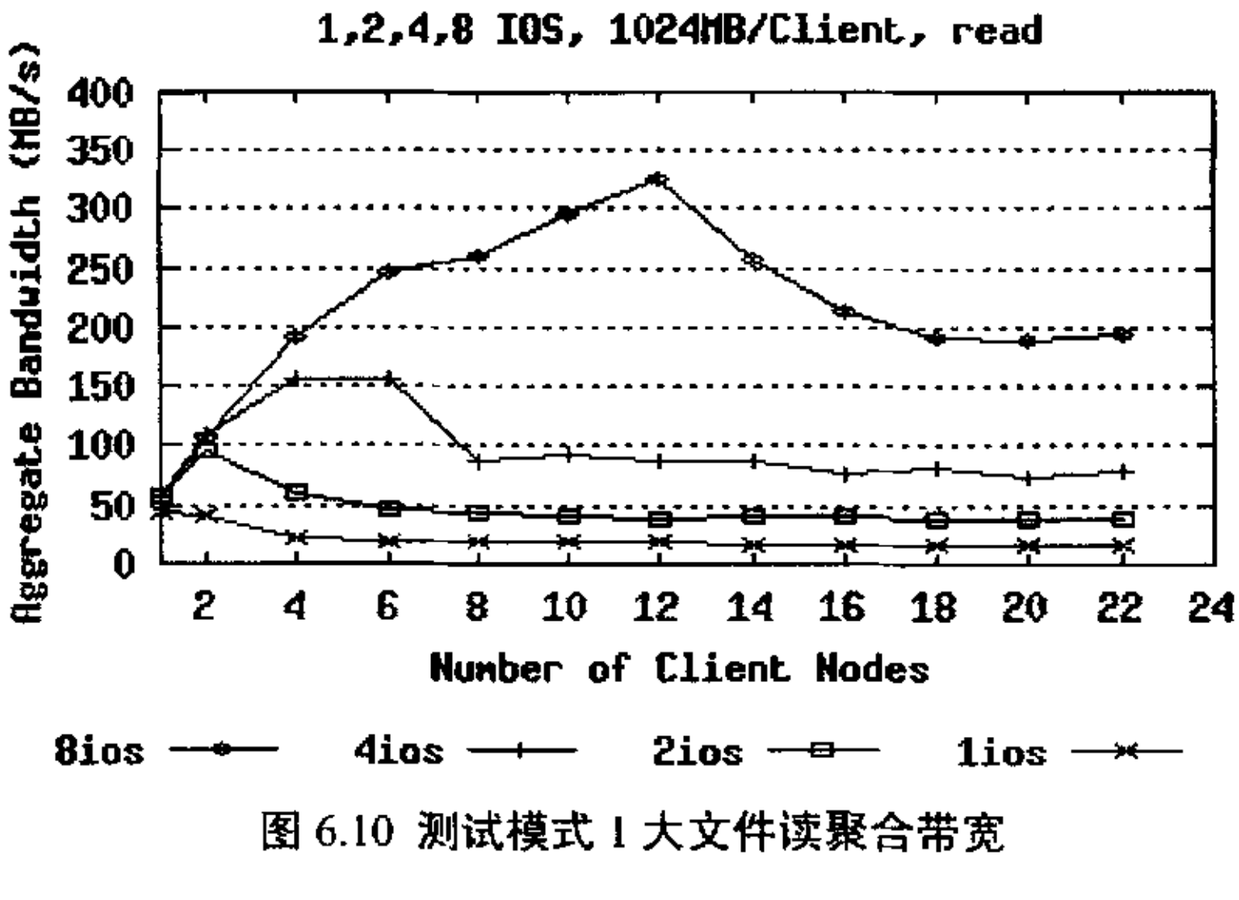


图 6.10 测试模式 1 大文件读聚合带宽

表 6.7 存储服务器本地文件系统读性能测试

Number Of Ioses	Number of Client Nodes							
	1	2	4	8	12	16	20	22
1	1491.087	160.257	81.042	63.427	63.842	58.414	60.048	57.543
2	1491.929	1505.184	155.221	84.495	64.892	64.013	61.144	56.426
4	1489.141	1478.521	1501.410	154.313	93.630	799.57	69.689	68.475
8	1475.056	1489.851	1478.938	1499.405	693.768	166.728	108.099	98.811

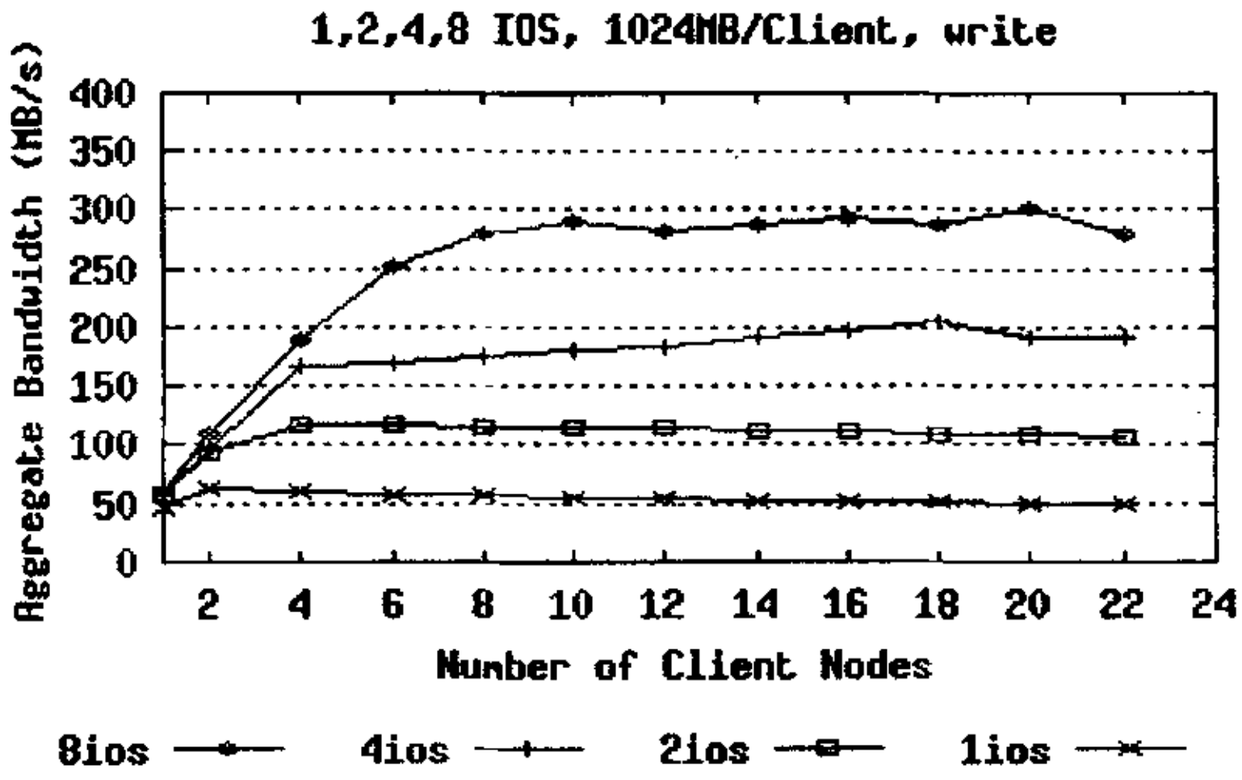


图 6.11 测试模式 1 大文件写聚合带宽

为了反映存储服务器在大数据量读写情况下的性能，我们进行第二种负载模式下的大文件测试：每个存储服务器读写字节数为 6144MB，远远大于存储服务器物理内存容量（2GB）。图 6.12 和 6.13 给出了 DCFS 的读写聚合带宽，表 6.8 给出了相应的性能参数值，为了比较 DCFS 和存储服务器上本地文件系统性能，表 6.9 给出了本地文件系统测试的结果，其中线程的个数与客户节点个数相同，读写数据总量为 6144MB，可以看到，存储服务器磁盘读性能在测试进程数大于 1 时下降，然后稳定在 33MB/s，而磁盘的写性能较为稳定。DCFS 读写聚合带宽表现出了和磁盘性能相似的趋势，读聚合带宽在到达峰值带宽后稍有下降，稳定于某个饱和值，在 2、4、8 个服务器配置下，服务器加速比为 1.949、3.217、5.718；写聚合带宽稳定于峰值带宽。DCFS 聚合读写带宽随着客户节点数的增加很快达到最高聚合带宽，可以看到 DCFS 具有较好的可扩展性。

表 6.8 测试模式 2 大文件测试性能参数

Num. Of IOSes	Read		Write	
	Max Bandwidth (MB/sec)	*Speedup*	*Max Bandwidth (MB/sec)*	*Speedup*
1	31.192	1.000	59.578	1.000
2	60.790	1.949	111.725	1.875
4	100.347	3.217	198.840	3.337
8	178.361	5.718	292.576	4.911

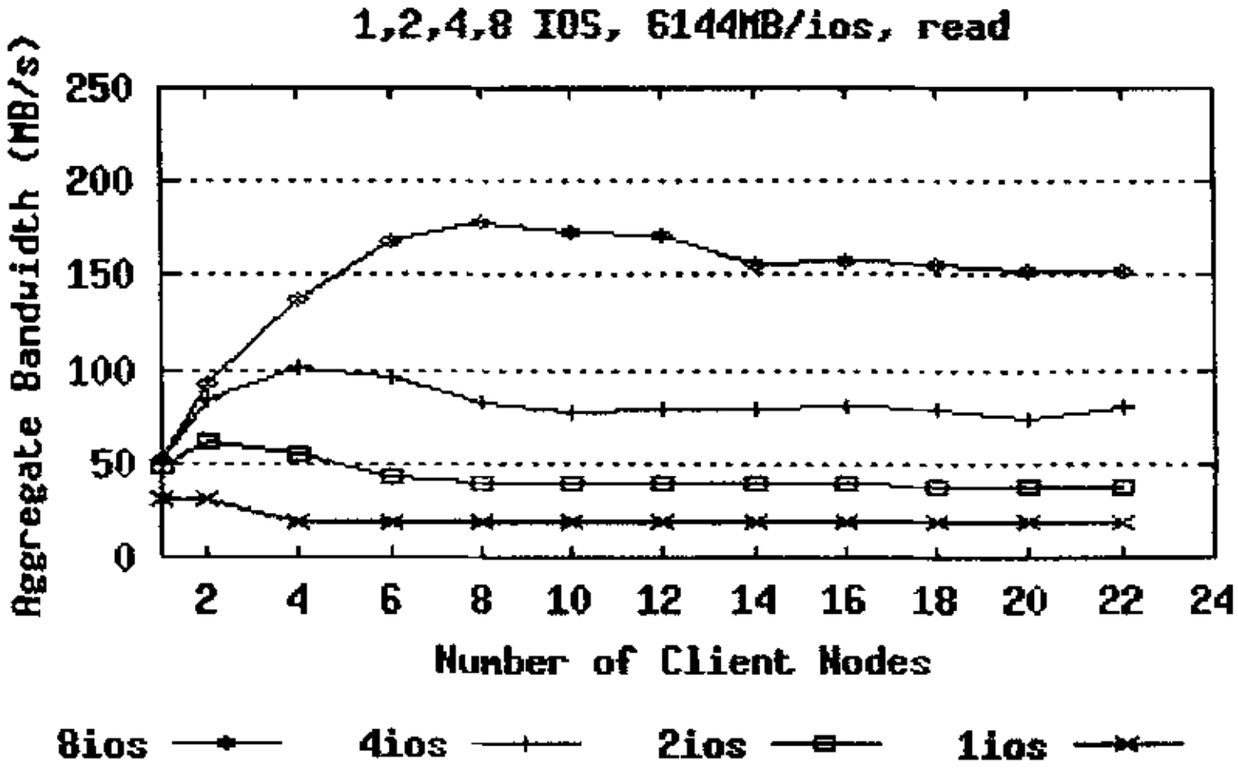


图 6.12 测试模式 2 大文件读聚合带宽

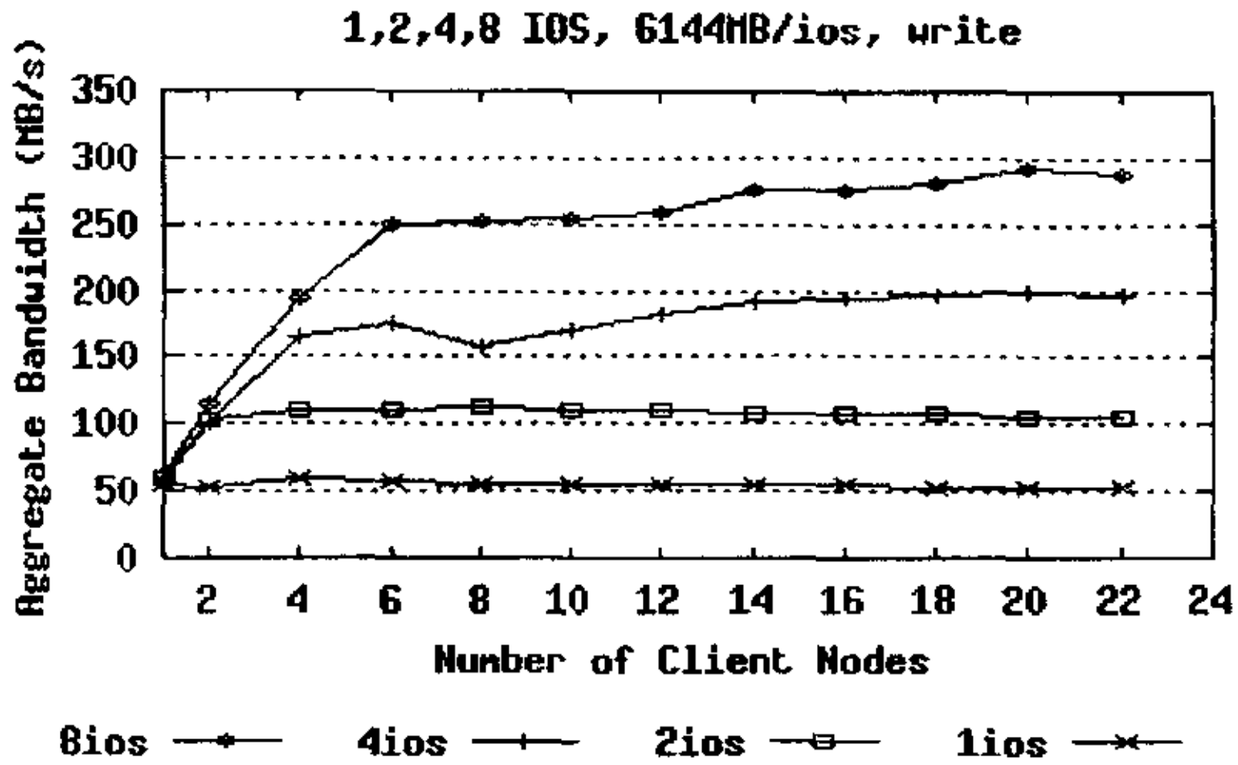


图 6.13 测试模式 2 大文件写聚合带宽

表 6.9 存储服务器本地文件系统测试

Num. Of Threads	Total Size (MB)	Read Bandwidth (MB/sec)	Write Bandwidth (MB/sec)
1	6144	66.800.	63.193
2	6144	41.476.	63.902
4	6144	34.943.	63.292
6	6144	33.860.	62.789
8	6144	33.880.	62.709
10	6144	33.850	61.661
12	6144	33.399	61.809
14	6144	33.155	60.922
16	6144	34.107	61.355

6.3.5 DCFS、PVFS 读写性能比较

图 6.14 给出了 DCFS 和 PVFS 小文件读写性能的比较，DCFS 和 PVFS 均配置成 8 个存储服务器和 1 个元数据服务器，每个存储服务器读写 256MB 数据。由于每个存储服务器上读写的数据较少，其数据在测试过程中都位于本地操作系统的 buffer cache 中，服务器上读写的开销很小，所以该测试结果反映了 DCFS 和 PVFS 对网络的利用率。从图中可以看到，DCFS 的写性能要优于 PVFS，DCFS 的最高写聚合带宽比 PVFS 高 58.0%，而读性能要差于 PVFS，DCFS 的最高读聚合带宽比 PVFS 差 19.0%，这是 PVFS 存储服务器端在读操作处理时使用了 sendfile 系统调用，buffer cache 中的数据在核心中直接通过 socket 发送，而 DCFS 由于在存储服务器上采用了缓存，读操作时需要把数据读到缓存中然后发送，增加了开销。

图 6.15 给出了 DCFS 和 PVFS 大文件读写性能的比较，DCFS 和 PVFS 均配置成 8 个存储服务器和 1 个元数据服务器，每个存储服务器读写 6144MB 数据。DCFS 的写性能要优于 PVFS，DCFS 的最高写聚合带宽比 PVFS 高 43.9%，读性能在节点数少时（小于 4 个客户节点）要差于 PVFS，随着测试节点数增加，DCFS 的读性能要优于 PVFS，DCFS 的最高读聚合带宽比 PVFS 高 31.2%。

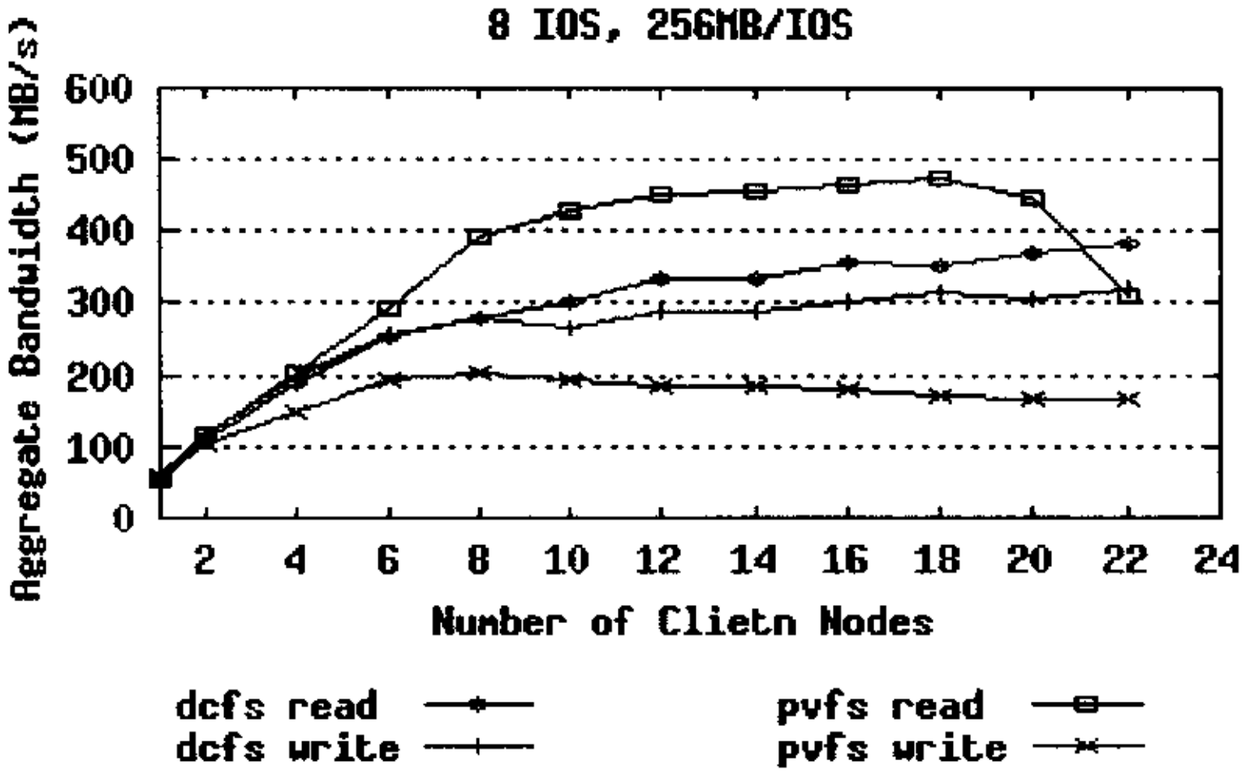


图 6.14 DCFS 和 PVFS 小文件读写性能比较

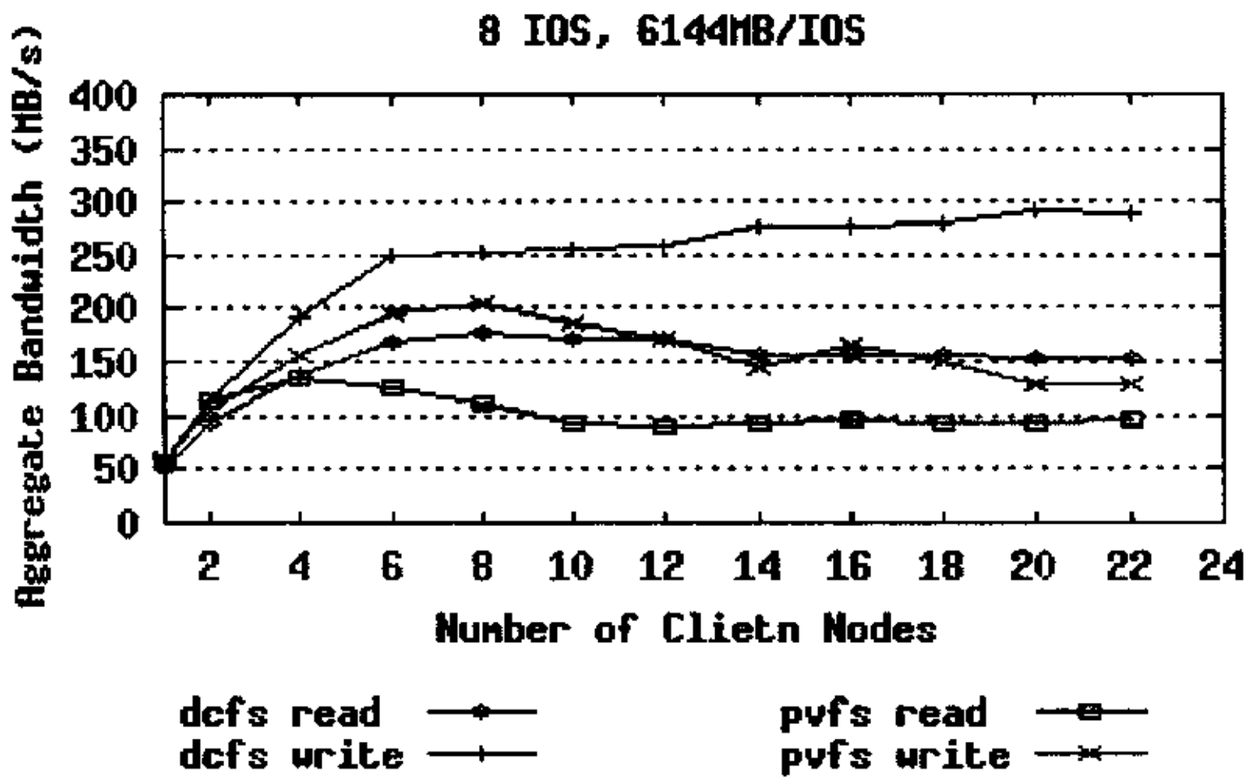


图 6.15 DCFS 和 PVFS 大文件读写性能比较

6.4 元数据服务器性能评价

6.4.1 测试方法

6.4.1.1 吞吐率测试

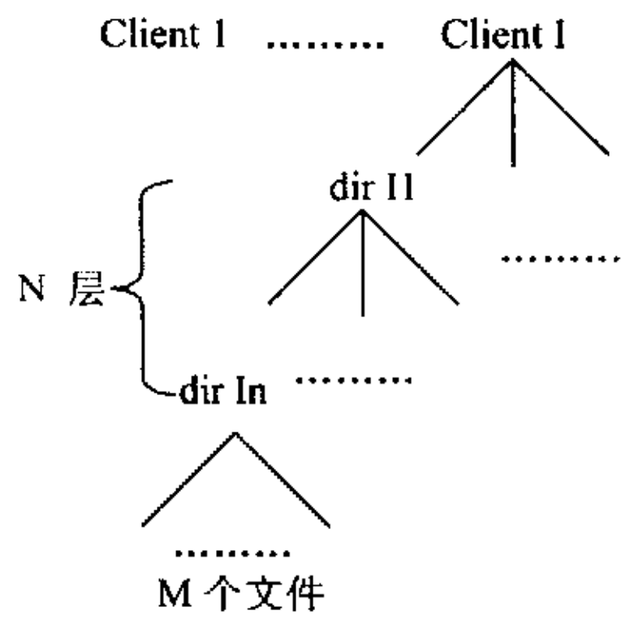


图 6.16 吞吐率测试示意图

图 6.16 给出了在吞吐率测试中目录树结构，每个客户端测试进程在根目录下创建一棵深度为 N+1，宽度为 M 的子树，为了使测试纯粹地反映元数据服务器的性能而不涉及存储服务器，文件的长度设为为 0 字节。文件系统在不同目录大小和层次的情况下，吞吐率的性能不同。在机群文件系统吞吐率测试中，我们选择了两种负载：（1）反映机群文件系统在不同目录大小下性能，相应的，子树的深度固定为 1，每个目录下创建 M 个文件；（2）反映机群文件系统在不同目录层次下性能，相应的，子树的深度为 N+1，每个目录下创建的文件固定为 M。

5.4.1.2 综合负载测试

FSbench[Lu03]是本研究小组为了全面、方便、有效的衡量机群/分布式文件系统的性能而设计的一个文件系统性能测试工具包。FSbench 是一种基于 UNIX 平台的机群文件系统基准程序，能用于测试常用的机群文件系统性能指标，如读写带宽和吞吐率，另外，它扩展 SPEC SFS[SFS01]的负载测试方法并用来测试多种机群文件系统，提供了多种测试共享文件的模式，以测试共享文件的并行存取性能。

图 6.17 是 FSbench 的总体结构，它在结构上分为 FSbench 管理节点和 FSbench 客户两部分。管理节点和客户节点可以配置在一起也可以分开配置。图 6.18 是 FSbench 客户节点的结构。

FSbench 在结构上分为 FSbench Manager，FSbench Client，FSbench syncd 三种模块和 FSBMGR，FSBMCR 两种脚本。管理模块的 FSBMGR 脚本程序读入和分析全局配置文件和测试参数配置文件，并把任务分解后在客户节点运行 FSBMCR 脚本程序，FSBMCR 脚本激活 FSbench Client 父进程并处理基准程序的输出结果。FSbench Manager 管理 FSbench Client 模块运行时各个阶段的同步，FSbench Client 运行嵌入了同步机制的基准测试程序，FSbench Client 父进程派生出许多子进程来执行实际的基准程序操作。子进程之间的同步通过共享内存方式实现。FSbench syncd 守护进程负责同步消息的接收和转发。同步机制使用基于 TCP 的 socket 实现。FSBMGR 脚本汇总所有的结果最后生成测试报告。

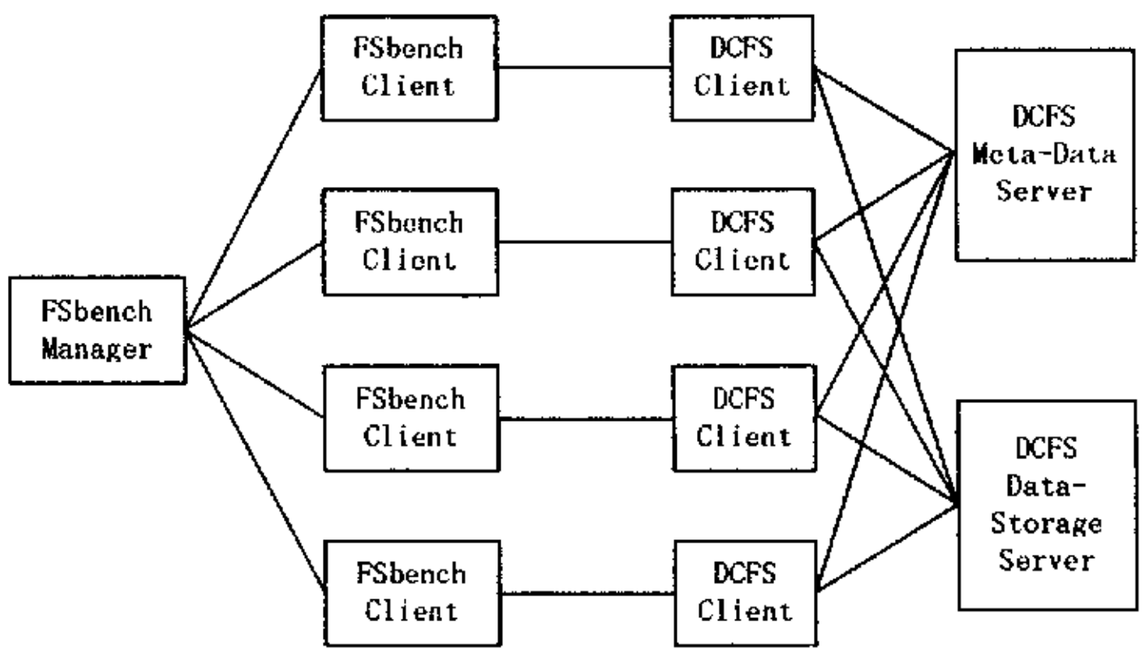


图 6.17 FSbench 的总体结构示意图

FSbench 提供了不同节点的多个进程并发协同工作的框架。在客户节点运行 I/O 带宽的基准程序程序时，FSbench 是一个测试文件系统聚集带宽的基准程序程序。在客户节点运行模拟工作负载生成程序时，FSbench 是测试系统吞吐率和响应时间的基准程序。

在本章的元数据性能测试中，利用 FSbench 的综合负载性能测试的功能，在机群文件系统的客户端按一定文件系统调用比例产生文件系统的负载，测试文件系统的响应时间和

吞吐率。

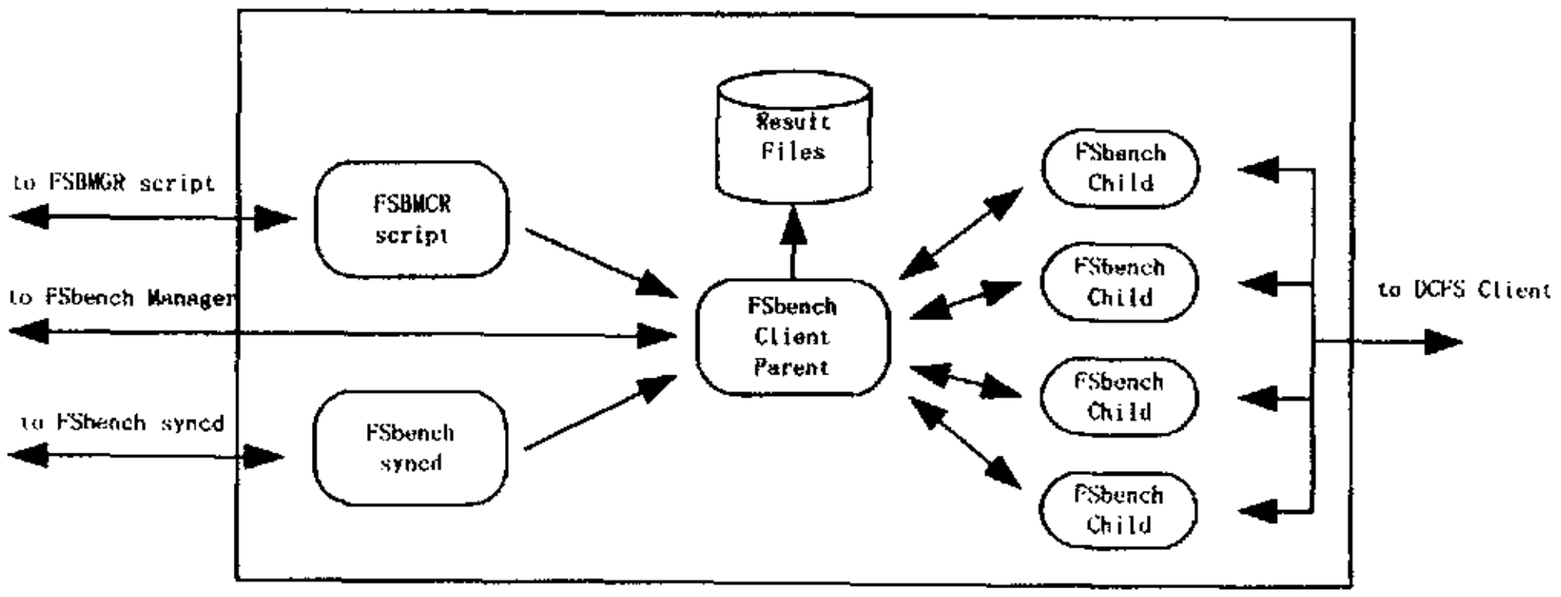


图 6.18　FSbench Client 结构示意图

6.4.2 性能参数

6.4.2.1 吞吐率

式（6.1）给出了机群文件系统吞吐率的定义，其中 t 是测试时间，n_c 是客户端数，m 是每个客户端的并发测试进程数，Ops_{ij} 是每个测试进程在时间 t_{ij} 内执行的所有系统调用数。

$$TP = \sum_{i=1}^{n_c}\sum_{j=1}^{m} Ops_{ij} / t_{ij} \qquad (6.1\text{ 式})$$

6.4.2.2 可扩展性

在并行计算中，我们常用固定问题规模加速比、固定时间加速比[Hwa98]等性能参数来表示一个并行程序的可扩展性。由于这些性能参数定义主要针对并行程序，假设运行在 k 个处理器集上，独占资源直到完成。但在分布式应用中，多个用户同时运行在系统中，竞争系统中资源，因此，在机群文件系统等分布式系统中需要另外定义吞吐率可扩展性的度量。

利用 P. Jogalekar 等在[Jog00]中的定义的生产力（productivity）参数，作者在本节中给出在机群/分布式文件系统中元数据吞吐率性能可扩展性的定义。Productivity 参数表示如下：

$$F(k) = \lambda(k) \cdot f(k) / c(k) \qquad (6.2\text{ 式})$$

其中，k 表示系统规模，$\lambda(k)$ 表示系统在规模 k 时的吞吐率，$f(k)$ 是系统性能参数的函数，表示在规模 k 下系统所获得的服务质量，$c(k)$ 表示系统开销。从上面表达式可以看出，productivity 与 $\lambda(k)$ 和 $f(k)$ 成正比，而与 $c(k)$ 成反比。基于 productivity 的定义，分布式系统可扩展性可以定义为：

$$\varphi(k,k') = F(k') / (F(k) \qquad (6.3\text{ 式})$$

对于一个可扩展性好的机群/分布式文件系统，用户希望在服务器配置一定的情况下，能够支持尽可能多的客户进程，并且保持一定水平的带宽或吞吐率，或者通过增加服务器个数来获得更高的性能。为了分别表示上面的两种可扩展性，我们分别定义如下的文件系统吞吐率的客户端可扩展性和服务器端可扩展性参数。

客户端可扩展性

由上面给出分布式系统的扩展性参数的定义，客户端可扩展性参数定义为：

$$\varphi\ (1,n_c) = (F(n_c)/F(1) \qquad (6.4\ 式)$$

其中 $F(n_c) = \lambda(n_c)\cdot f(n_c)/c(n_c)$，$F(1)$ 为 $n_c = 1$ 时的取值。我们定义 $c(n_c) = n_c$，由于本章采用的测试系统是一个闭系统，$\lambda(n_c) = n_c$，并且用户希望在系统中并发的用户数增多时仍保持较好的平均吞吐率，因此，定义 $f(k) = TP_{n_c}/n_c$，代入公式 6.4，得

$$\varphi(1,n_c) = (n_c \cdot \frac{TP_{n_c}}{n_c} \cdot \frac{1}{n_c})/(1 \cdot \frac{TP_1}{1} \cdot \frac{1}{1}) = \frac{TP_{n_c}/n_c}{TP_1} \qquad (6.5\ 式)$$

服务器可扩展性

同样的，我们定义元数据服务器的可扩展性参数，其中，定义 $f(k) = TP_{n_s,n_c}$，表示用户希望在增加新的服务器时能够获得更高的聚合吞吐率：

$$\varphi\ (1,n_s) = (F(n_s)/F(1) \qquad (6.6\ 式)$$

$$\varphi(1,n_s) = (n_c \cdot TP_{n_s,n_c}/n_s)/(n_c \cdot TP_{1,n_c}/1) = \frac{TP_{n_s,n_c}/n_s}{TP_{1,n_c}} \qquad (6.7\ 式)$$

6.4.2.3 全局响应时间

在 FSbench 测试中，我们使用吞吐率和响应时间曲线反映机群文件系统元数据性能，该曲线如图 6.19 所示，横坐标是聚合吞吐率 TP，单位是 Ops/s（操作数/秒）。纵坐标是平均响应时间，单位是毫秒。在进行 FSbench 的综合负载测试时，客户端产生的负载水平不断增加，直到文件系统的实际吞吐率开始下降。在每个负载水平的测试中，记录文件系统的平均响应时间和实际吞吐率，作为产生吞吐率和响应时间曲线的数据。同时为了比较不同的文件系统的响应时间，我们采用了 SFS[SFS01]中全局响应时间作为性能参数，该参数定义如下：

$$O.R.T = \frac{\sum_{i=1}^{n}\left[\frac{R_i + R_{i-1}}{2} * (TP_i - TP_{i-1})\right]}{TP_n} \qquad (6.8\ 式)$$

全局响应时间表示了吞吐率/响应时间曲线下面积与峰值吞吐率的比值，它表示了在整个测试的负载范围内，文件系统对各种操作响应时间的快慢，是系统在平均负载水平下响应时间的度量。

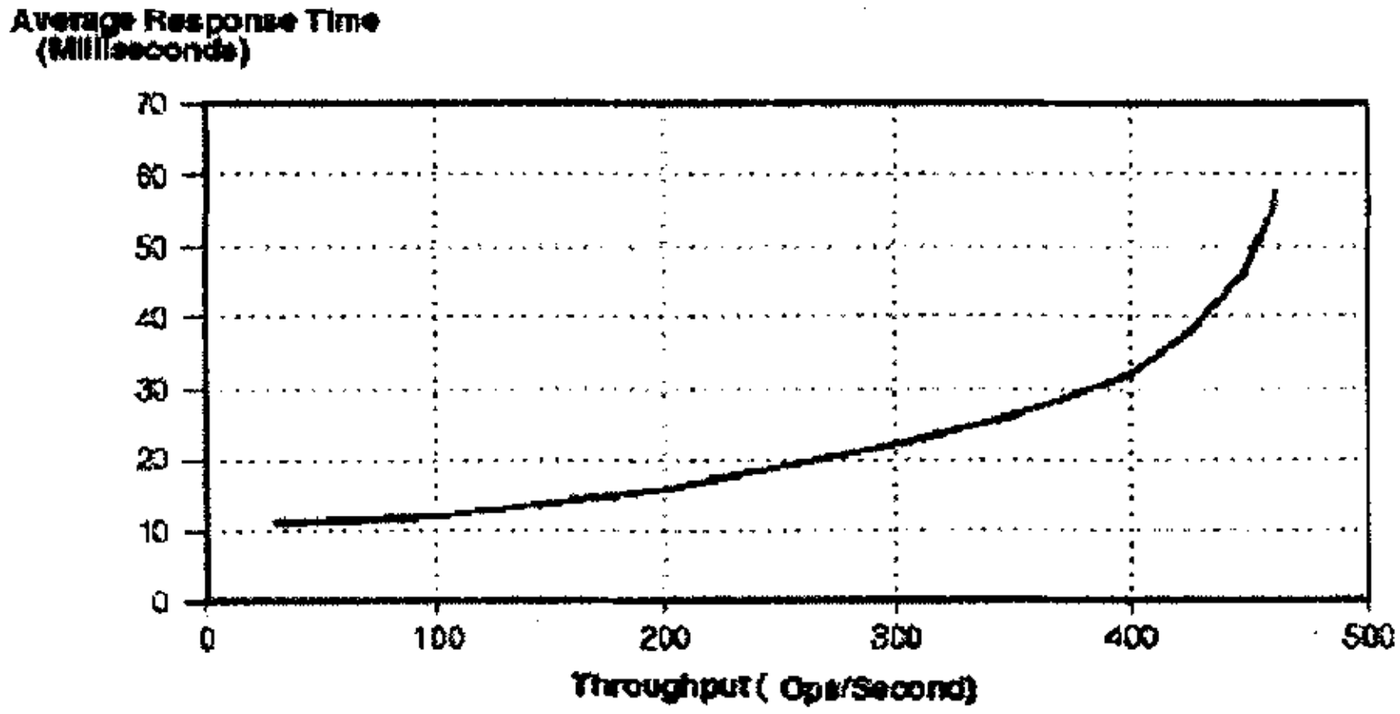


图 6.19 吞吐率和响应时间曲线

6.4.3 吞吐率测试

6.4.3.1 开销分析

在 DCFS 中，每个元数据操作请求的时间开销由客户端处理开销、网络开销和元数据服务器的处理开销，其中客户端的处理开销包括在核心的处理时间、CLERK 的处理时间以及核心与 CLERK 通讯的时间。网络开销与网络硬件的性能和每个元数据请求发送和接收的字节数相关，而元数据服务器的开销与具体的元数据请求类型相关。在下面，作者对元数据吞吐率测试中采用的两种负载分别分析元数据请求的处理开销。

在吞吐率测试的第一种负载中，每个客户进程在各自的目录下创建和删除文件，对创建操作而言，DCFS 生成一个文件的开销由三个部分组成：目录 LOOKUP 的时间、在父目录中添加目录项的时间开销和创建文件 inode 的时间开销。

对于第一项，在测试的配置中，DCFS 的每个目录 LOOKUP 请求需要发送到元数据服务器完成。在本章的元数据测试中，除了特别注明，元数据服务器都没有使用大目录优化，因此这部分开销随着目录中创建的文件个数的增加而增加。对于第二和第三项开销，由于在测试中把 DCFS 文件系统卷配置成按子树粒度分布元数据，因此父目录和文件都位于同一个元数据服务器上，这两步操作只需一个 CREATE 请求就可完成，在没有进行大目录优化的元数据服务器上，在父目录中添加目录项需要顺序检查目录中目录项。DCFS 在元数据服务器上使用了元数据缓存，元数据操作请求在缓存中完成，避免了 PVFS 中频繁的小文件的读写。DCFS 对创建文件请求进行了优化，不需要发送创建文件的请求到存储服务器，存储服务器上的文件只有在第一次写时才创建。

对于删除操作，在 DCFS 中，删除一个文件的开销由四个部分组成：目录 LOOKUP 的时间、在父目录中删除目录项的时间开销、删除文件 inode 的时间开销和删除存储服务器上的文件的开销。

由于元数据服务器采用了目录缓存管理，目录 LOOKUP 操作通过查找 HASH 表可以

快速查找到目录项，并定位到目录文件块缓存的位置，然后删除该文件。DCFS 对删除存储服务器上文件的过程进行了优化，元数据服务器把删除文件的请求转发给存储服务器后立即给客户端发送应答消息，而不必等待存储服务器的返回，这样，使存储服务器和元数据服务器的处理异步，提高了元数据操作的性能。

对于元数据测试中的第二种负载模式，创建和删除文件时，对于位于第 N 层的文件，需要发送 N 个 LOOKUP 消息，这使得在创建和删除文件中需要多次穿越网络，开销随层数的增加而增加。

6.4.3.2 DCFS 吞吐率测试分析

图 6.20 和 6.21 给出了 DCFS 配置成 1、2、4、8 个元数据服务器时每个客户测试进程创建和删除 1000 个文件的创建、删除吞吐率。从图 6.20 可以看到，在 DCFS 只配置成一个元数据服务器服务器，在客户并发的测试进程数增加到 14 个节点时，创建和删除的客户端扩展性参数为 0.5，由扩展性定义可知每个测试进程的平均创建和删除吞吐率仍有单个测试进程吞吐率的 0.5，同时，聚合吞吐率稳定于饱和值而没有明显下降；在 DCFS 配置成 8 个元数据服务器服务器，在客户并发的测试进程数增加到 22 个节点时，每个测试进程的平均创建和删除吞吐率仍有单个测试进程吞吐率的 0.5，其聚合吞吐率仍保持增长。DCFS 吞吐率在一个服务器配置时随着客户测试进程数的增加趋于饱和，增加到 2 个服务器时其吞吐率在并发测试进程较多时有较大的提高，其中服务器扩展性度量在 20 个测试进程时为 0.8，即增加一个服务器平均可以获得 80%的性能提高。我们看到，在 4、8 服务器时，吞吐率与 2 个服务器配置相比提高不大，这是因为服务器相对负载较轻，增加服务器对性能的提高不大。

图 6.22 和 6.23 给出了 DCFS 配置成 1、2、4、8 个元数据服务器时每个客户测试进程创建和删除 5000 个文件的创建、删除吞吐率。在 DCFS 只配置成一个元数据服务器服务器时，每个测试进程的平均创建吞吐率下降得较快，相应的，客户端可扩展性参数也下降得很快，这是因为在这个测试中元数据服务器很快饱和，但其创建吞吐率保持在 460Ops/s 以上，而在 DCFS 配置成 8 个元数据服务器服务器时，在客户并发的测试进程数增加到 22 个节点时，客户端可扩展性度量大于 0.5，即每个测试进程的平均创建和删除吞吐率仍有单个测试进程吞吐率的 0.5。在本章的性能评价中，DCFS 没有使用第四章中介绍的 LMEH 优化算法，因此在该测试中服务器的负载较重。吞吐率在一个服务器配置时随着客户测试进程数的增加趋于饱和，服务器增加时其吞吐率有较大的提高，对于创建吞吐率，在 2、4、8 个服务器配置时，服务器扩展性度量在 20 个测试进程时分别为 1.11、0.97、0.72，对于删除吞吐率，在 2、4、8 个服务器配置时，服务器扩展性度量在 20 个测试进程时分别为 1.32、0.86、0.45。

图 6.24 至 6.27 分别给出 DCFS 在每个客户测试进程创建和删除 8 层和 12 层目录文件的吞吐率结果。我们看到，DCFS 在配置成一个元数据服务器时，随着客户端测试进程的增加达到饱和，对与 8 层目录的测试，在客户并发的测试进程数增加到 14 个节点时，创建和删除的客户端扩展性参数大于 0.5，对于 12 层目录测试，聚合吞吐率很快达到饱和并下降，但当 DCFS 在配置成 8 个元数据服务器时，创建和删除吞吐率的客户端可扩展性都较好，其聚合吞吐率呈线性增长。我们看到，在 8 层目录测试中，2 个服务器配置的聚合吞吐率相比 1 个服务器提高较大，而 4、8 个服务器时性能提高不多，这是因为服务器相

对负载较轻，增加服务器对性能的提高不大。在 12 层目录测试中，服务器增加可以较大的提高其吞吐率。

从上面的分析可以看到，DCFS 元数据服务器可以提供较好的吞吐率性能，并且使客户端具有较好的可扩展性。

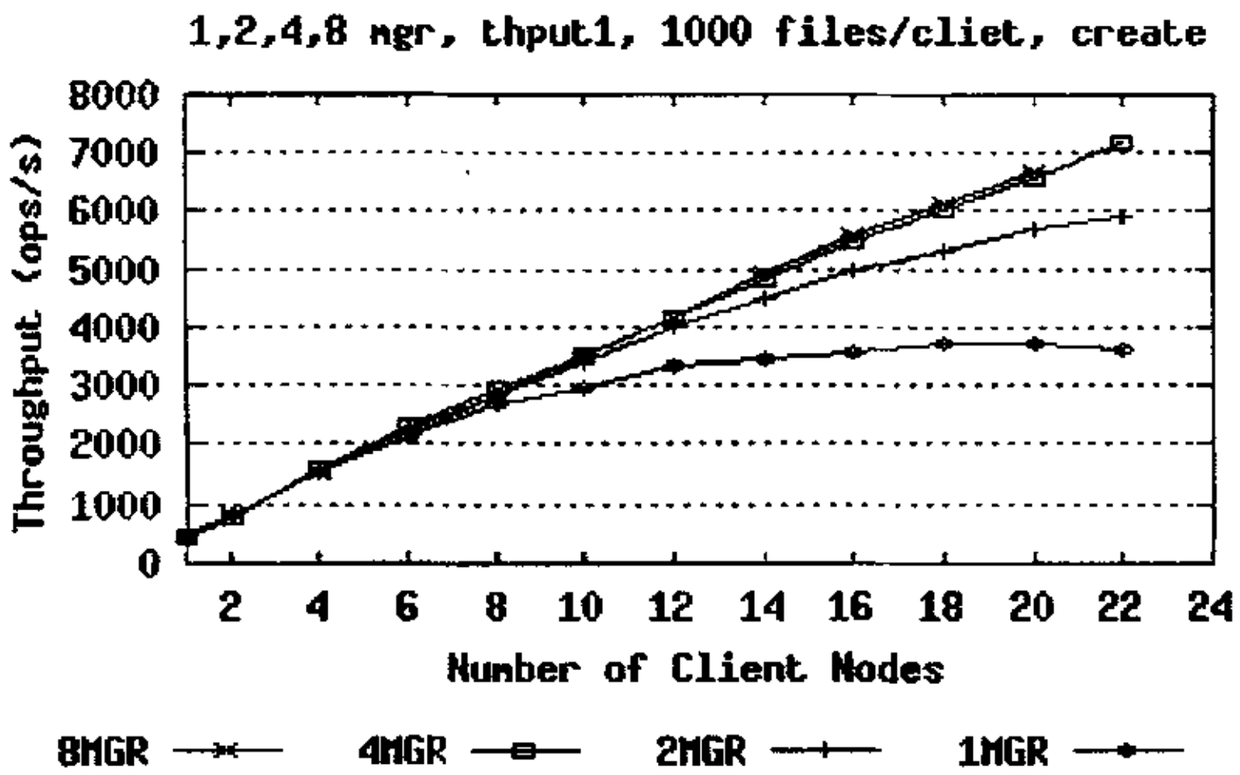


图 6.20 负载模式 1 每个目录 1000 个文件的创建吞吐率

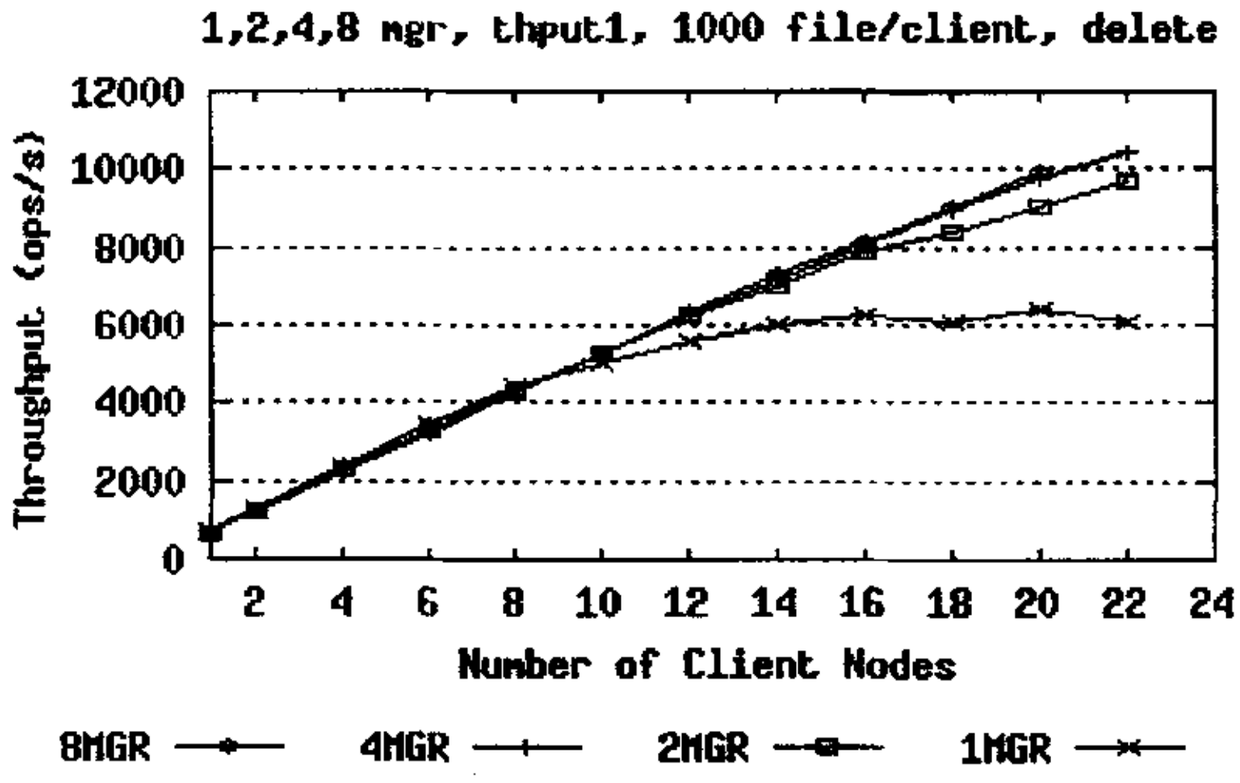


图 6.21 负载模式 1 每个目录 1000 个文件的删除吞吐率

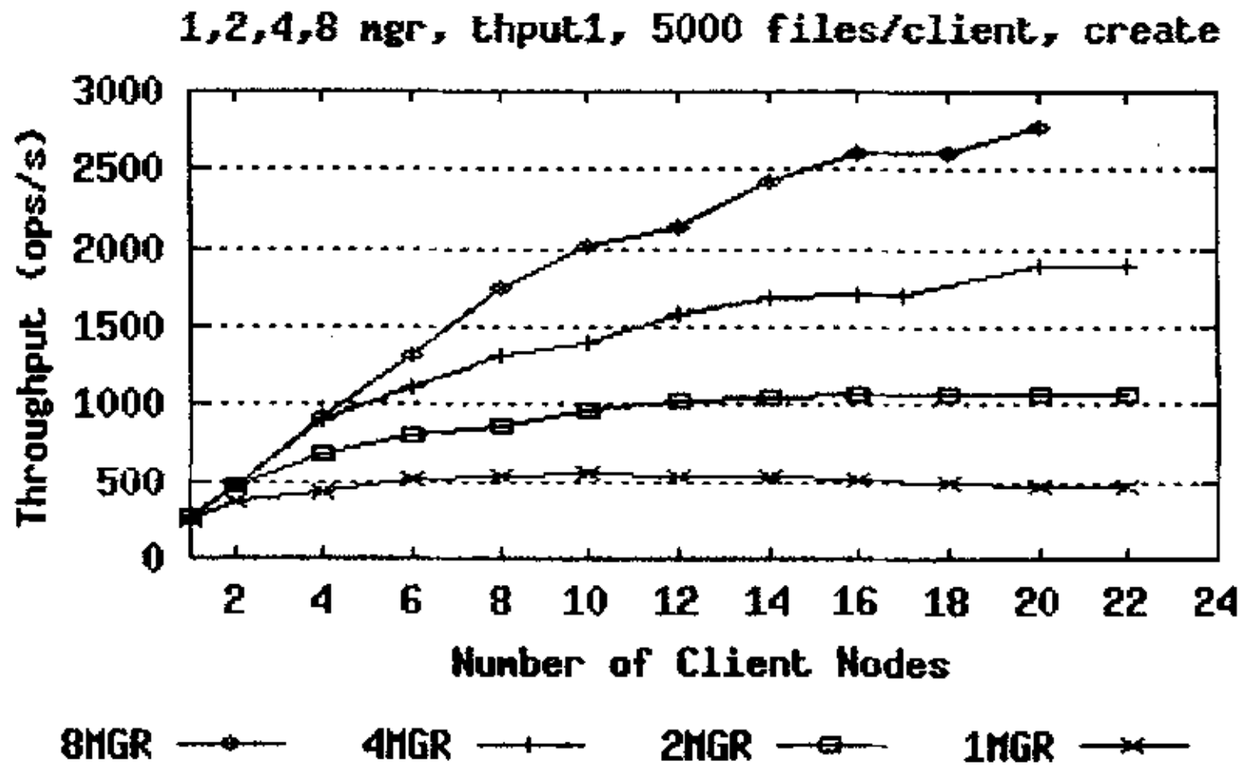


图 6.22 负载模式 1 每个目录 5000 个文件的创建吞吐率

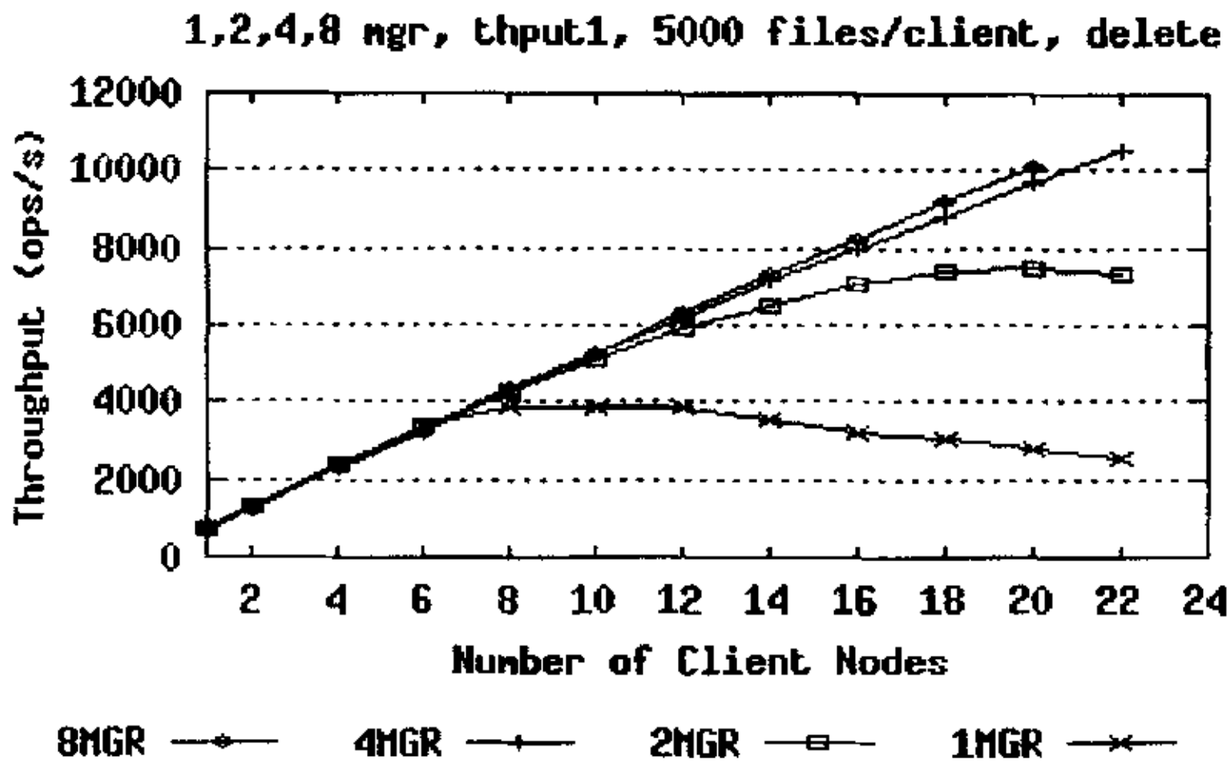


图 6.23 负载模式 1 每个目录 5000 个文件的创建吞吐率

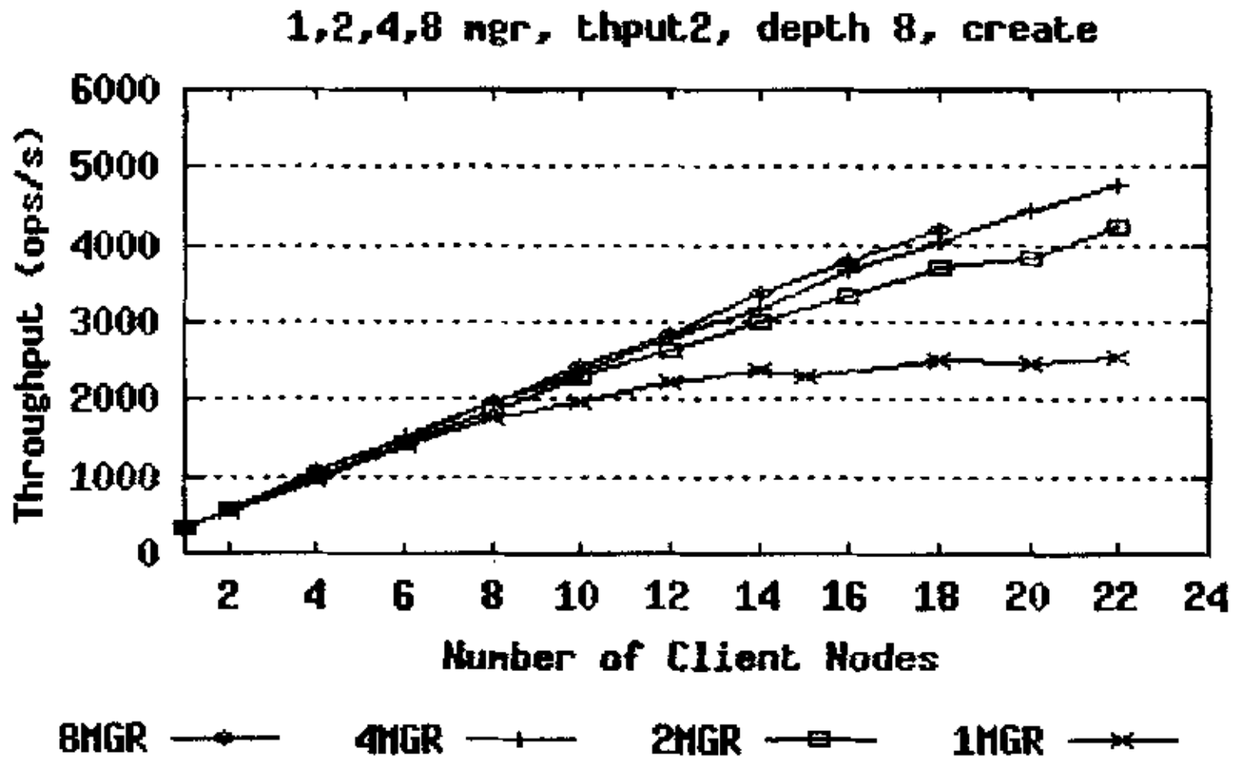


图 6.24 负载模式 2 的 8 层目录的创建吞吐率

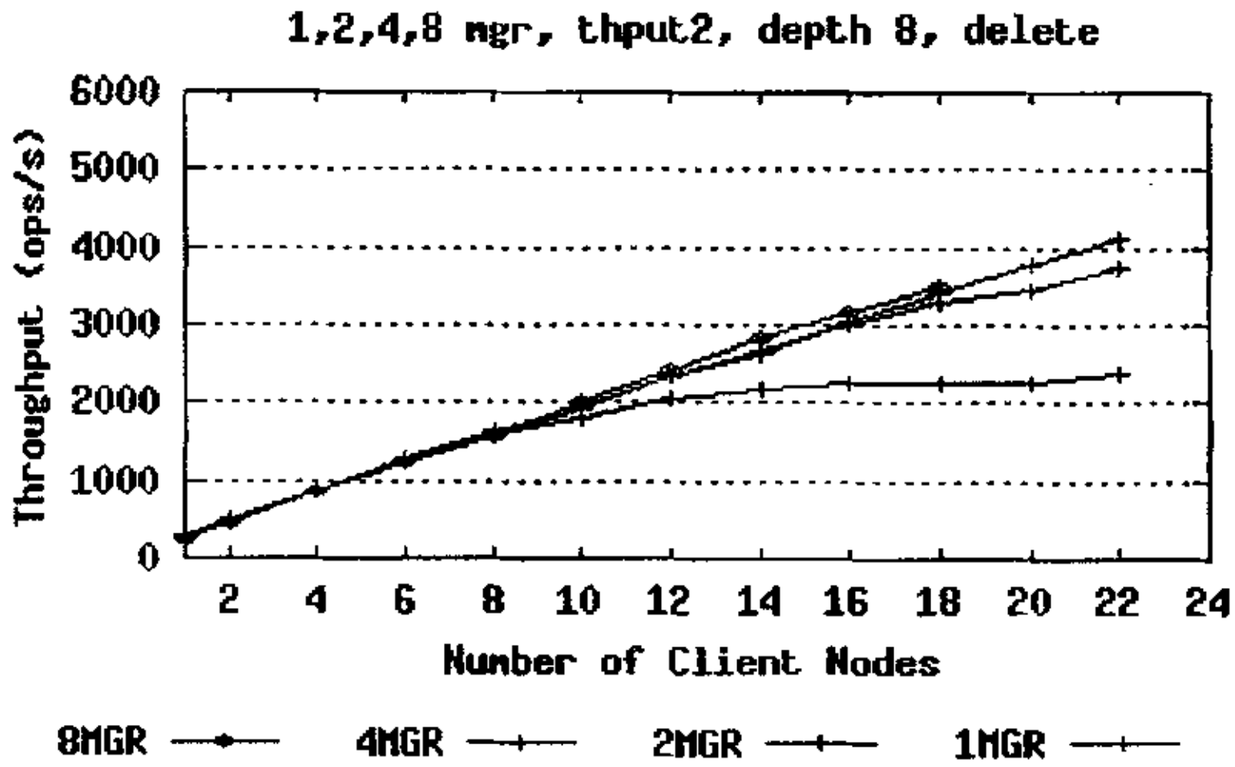


图 6.25 负载模式 2 的 8 层目录的删除吞吐率

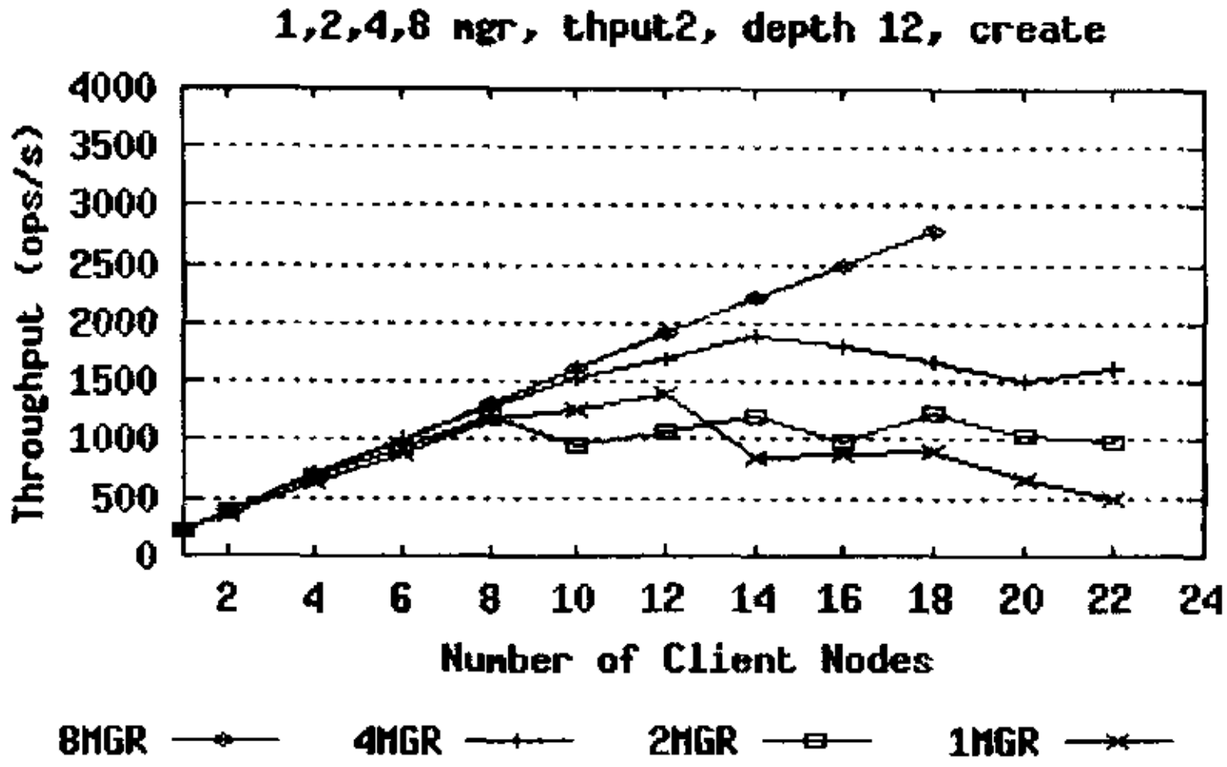


图 6.26 负载模式 2 的 12 层目录的创建吞吐率

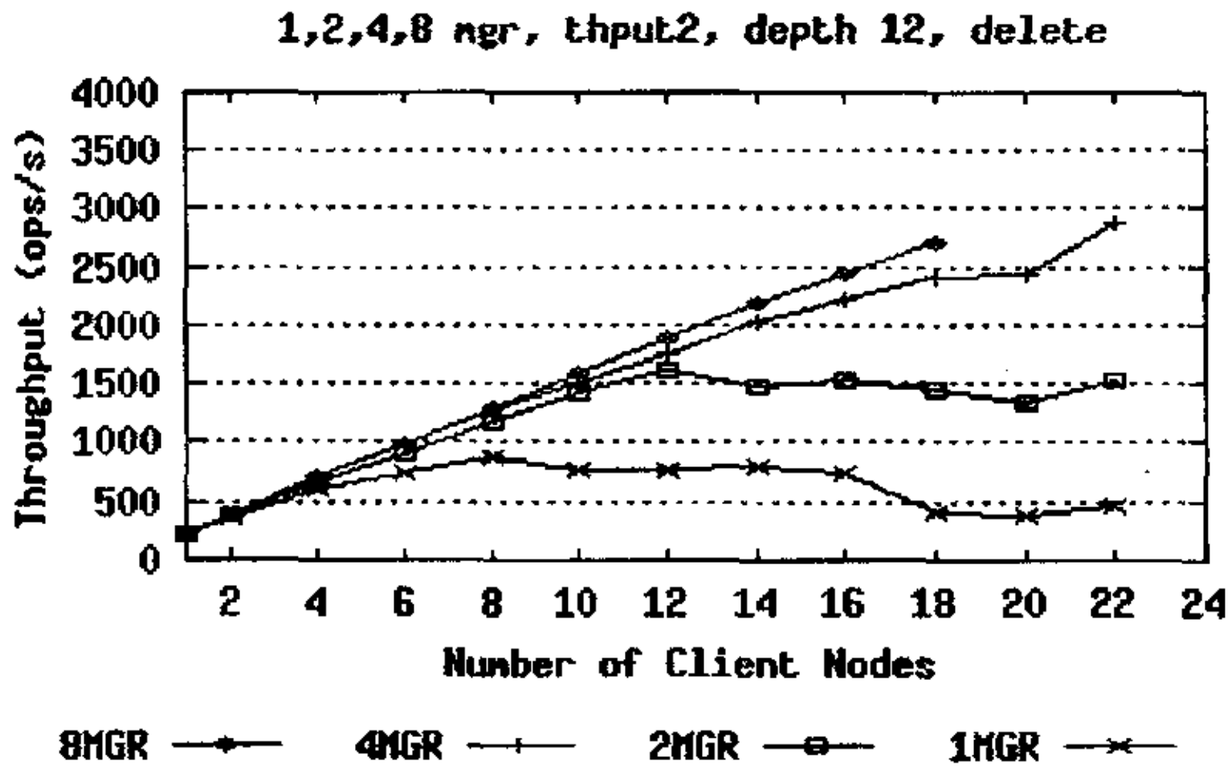


图 6.27 负载模式 2 的 12 层目录的删除吞吐率

6.4.3.3 DCFS、PVFS 吞吐率比较

图 6.28 到 6.31 给出了 DCFS 和 PVFS 在一个元数据服务器配置时的吞吐率性能比较。从中可以看到，DCFS 吞吐率性能在所有情况下都优于 PVFS，DCFS 在每个客户测试进程创建和删除 1000 和 5000 个文件时的最高聚合创建、删除吞吐率分别是 PVFS 的 13.622、11.949、2.044 和 7.231 倍，而在每个客户测试进程创建和删除 8 层和 10 层目录文件的最高聚合创建、删除吞吐率分别是 PVFS 的 6.946、5.276、7.633 和 4.428 倍，DCFS 的吞吐率性能平均是 PVFS 的 7.391 倍。PVFS 和 DCFS 体系结构相似，都是基于 Client/Server 结构的机群文件系统，而且在具体实现时，客户端采用了和 DCFS 类似的核心模块和客户端进程配合的软件结构，每个元数据操作的开销和 DCFS 一样，由客户端开销、网络开销和元数据服务器开销构成。DCFS 的元数据性能优于 PVFS 是因为 DCFS 元数据服务器采用和更为高效的软件结构，PVFS 的元数据以文件的形式按目录树的结构存储在服务器上的本地文件系统中，没有实现服务器端的元数据缓存管理，所有的元数据操作都需要读写文件，而 DCFS 实现了元数据缓存，避免了频繁的小文件的读写操作，提高了元数据服务器处理元数据操作的性能。另外，DCFS 采用了多个元数据服务器的结构，可以进行扩展，可以

提供更高的元数据性能。

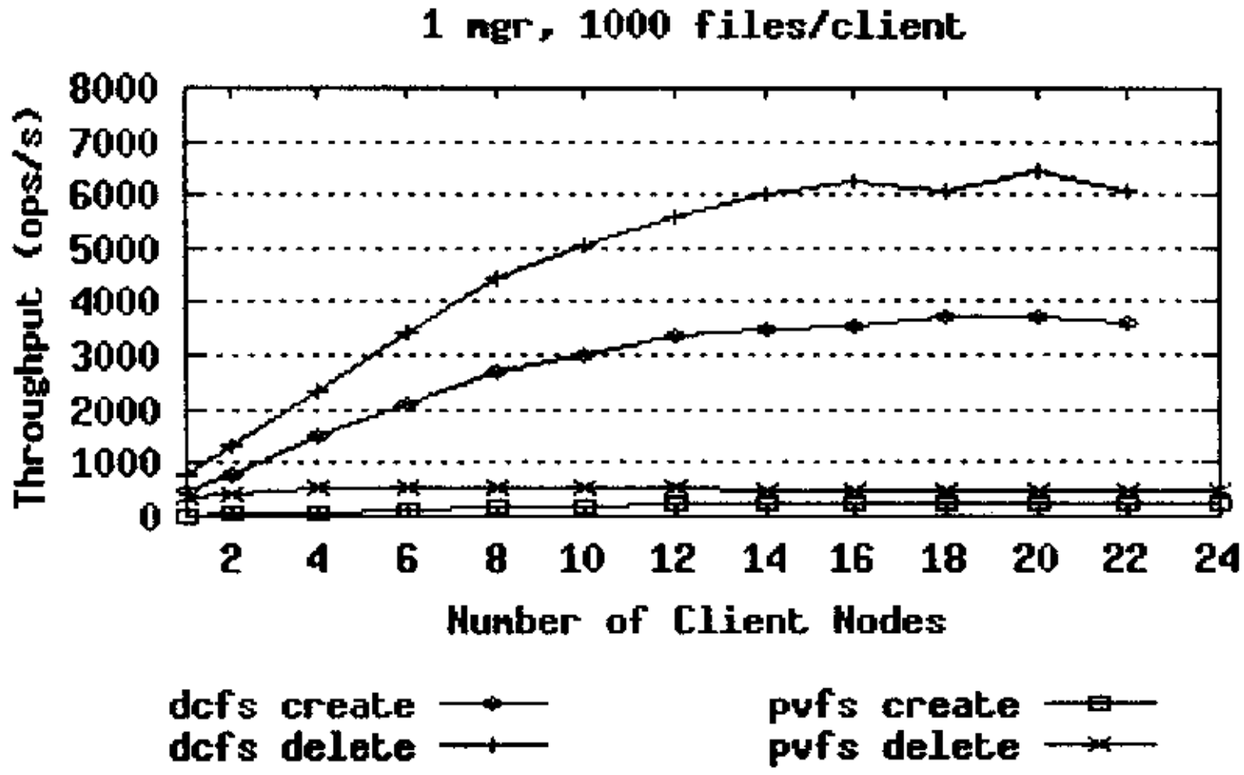


图 6.28 负载模式 1 每个目录 1000 个文件的创建和删除吞吐率比较

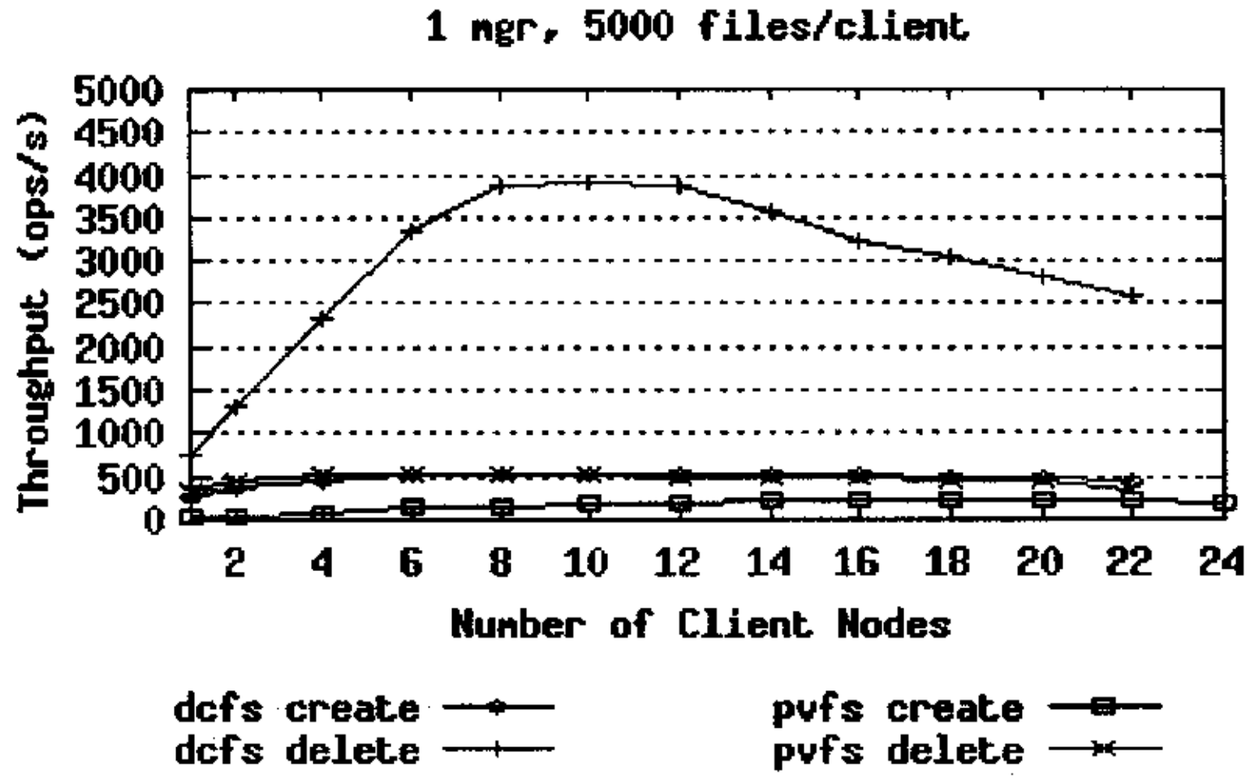


图 6.29 负载模式 1 每个目录 5000 个文件的创建和删除吞吐率比较

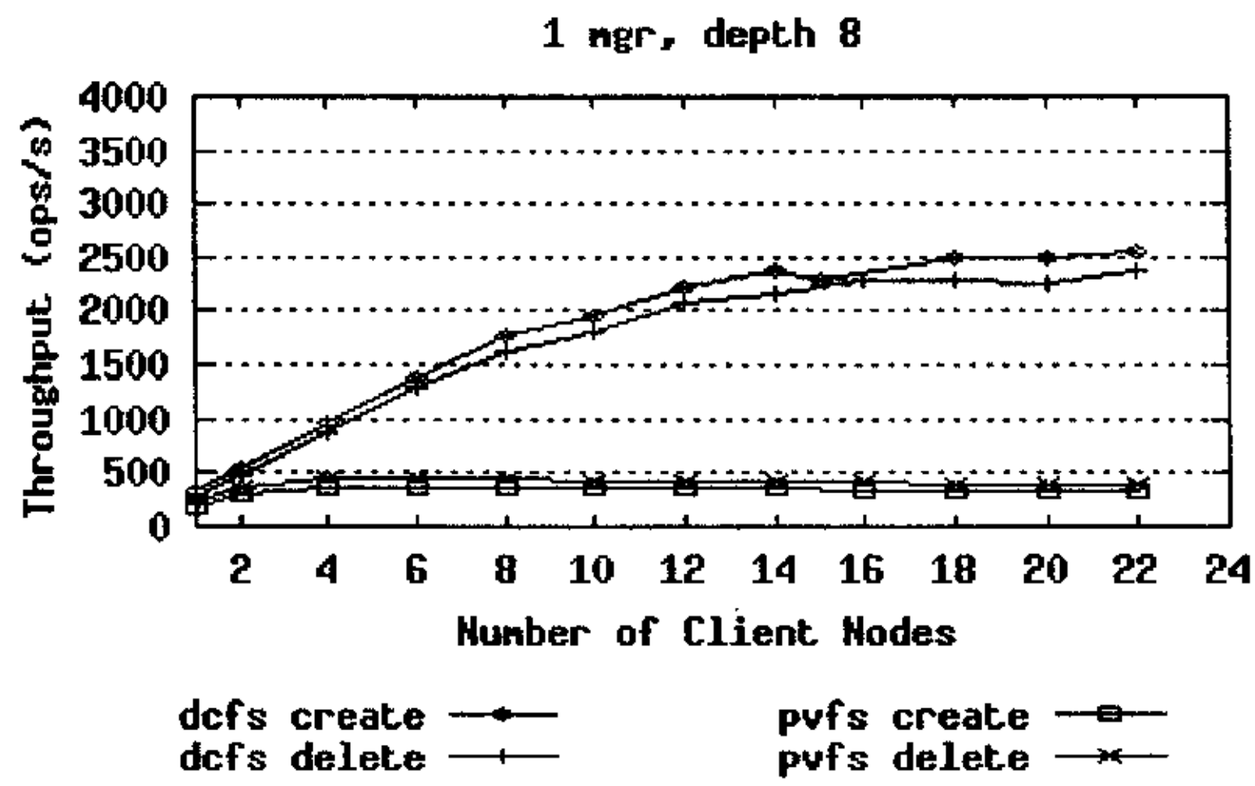


图 6.30 负载模式 2 的 8 层目录的创建和删除吞吐率比较

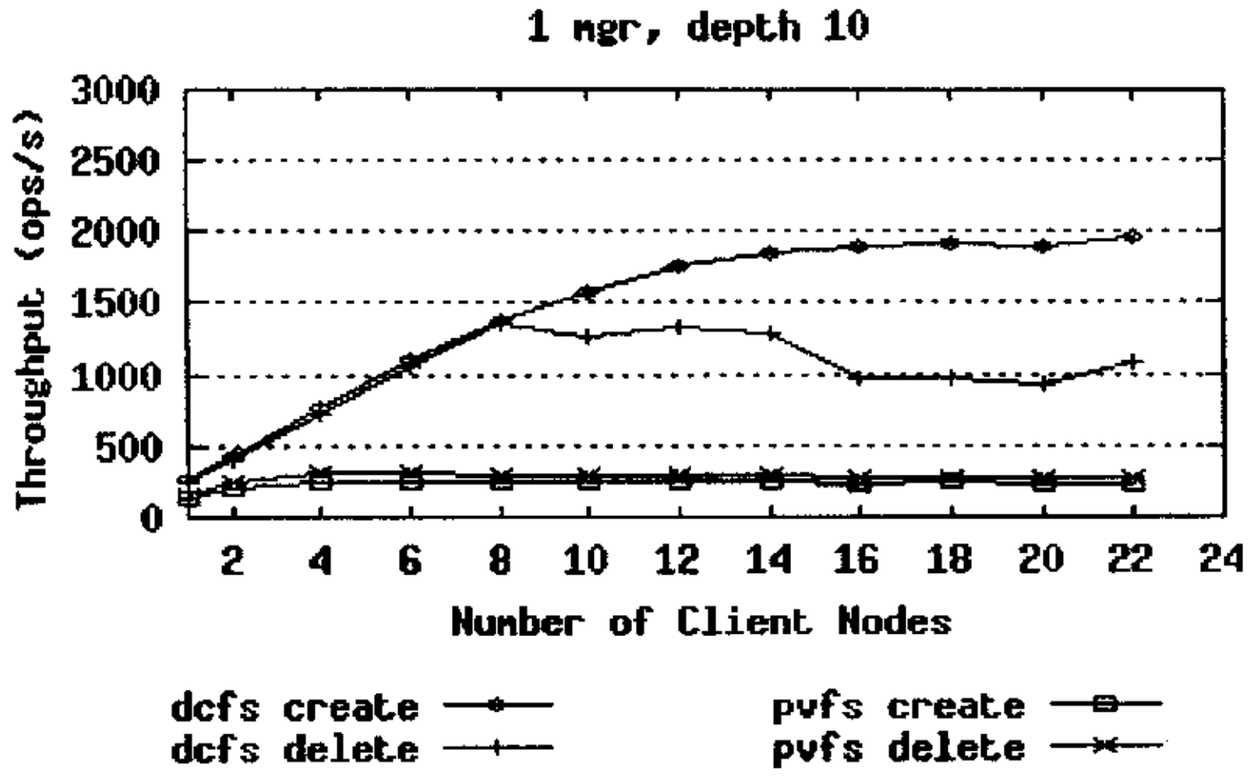


图 6.31 负载模式 2 的 10 层目录的创建和删除吞吐率比较

6.4.4 综合负载测试

图 6.32 给出给出了 DCFS 和 PVFS 都配置成 1 个元数据服务器和 8 存储服务器时，用 FSbench 进行综合负载测试的结果，表 6.10 则给出了在负载测试水平为 160ops/s 和 480ops/s 时每个系统调用的平均响应时间的比较。在综合负载测试中，系统调用比例参考 SFS[SFS01] 中的操作比例而定，由于 SFS 是针对 NFS 设计的文件系统测试程序，[Rob99]中给出操作是 NFS 文件协议操作，而 FSbench 并不针对特定的文件系统，所以负载综合测试中定义的操作是文件系统调用，对于与 SFS 中的操作相对应的系统调用，如 read、write 等，我们将其比例设置为 SFS 中相应的操作比例，而如果 SFS 的操作没有对应的文件系统调用，我们在系统调用中寻找合适的替代者，如 SFS 中的 LOOKUP 操作，其功能为查找目录项并返回文件属性，我们用系统调用 stat 代替，另外，对于没有合适文件系统调用替换的操作（NULL、COMMIT）和 FSbench 不支持的系统调用（readlink），我们将其操作比例分散到其他的系统调用中。文件集大小、文件大小的分布也都是参照 SFS 确定的。

表 6.10 不同负载水平下 DCFS 和 PVFS 系统调用响应时间比较

Syscall	Percent	160 ops/s		480ops/s	
		PVFS	DCFS	PVFS	DCFS
open	3	6.717	1.669	28.302	1.827
read	18	4.075	4.140	10.002	5.654
write	9	4.162	8.251	9.617	13.565
lseek	8	0.003	0.003	0.002	0.003
close	1	0.009	0.873	0.009	1.689
opendir	1	2.813	1.952	32.670	2.159
readdir	9	1.313	0.067	38.817	0.071
rewinddir	1	0.003	0.004	0.003	0.003
closedir	1	0.006	0.030	0.005	0.027
create	1	50.429	2.516	94.272	2.666
unlink	1	24.693	1.742	50.660	1.780
mkdir	1	8.041	2.246	47.462	2.268
rmdir	1	8.938	2.753	36.165	—
stat	27	17.898	1.658	43.560	1.785
fstat	11	3.700	0.342	11.786	0.382
access	7	6.112	1.348	28.011	1.454

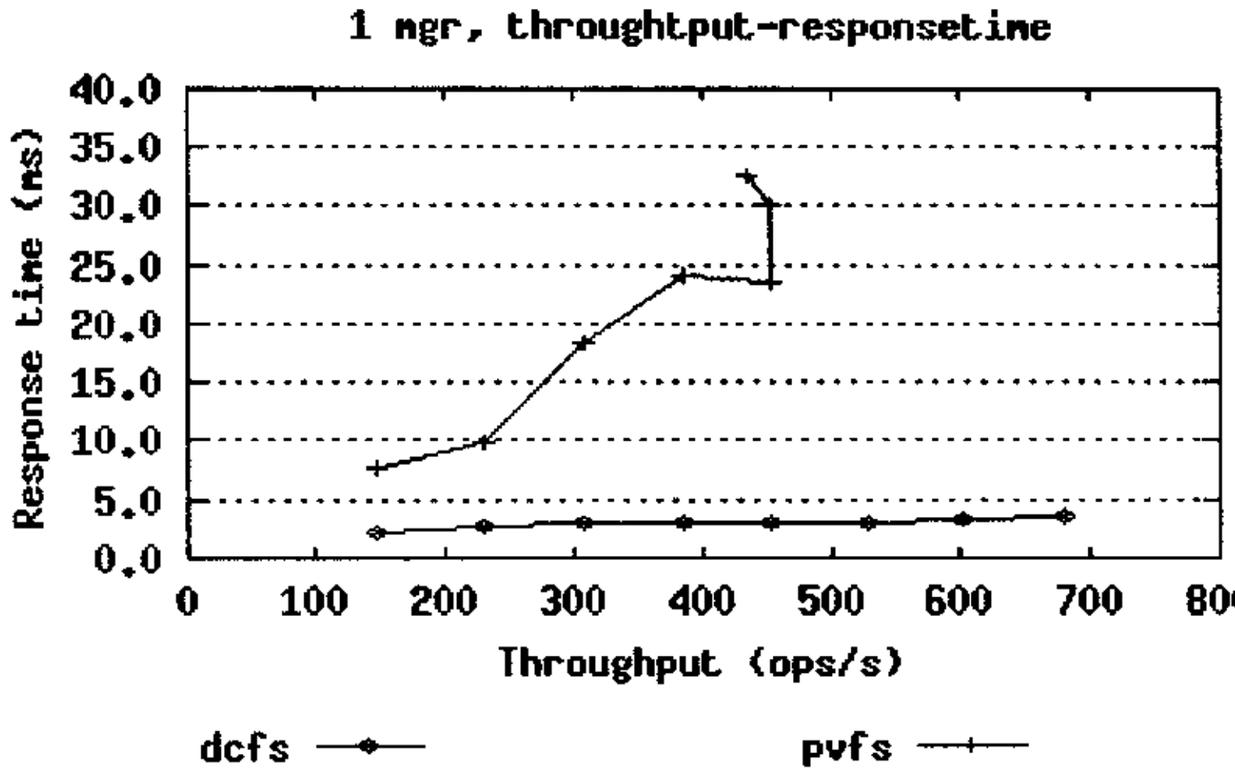


图 6.32 DCFS 和 PVFS 的负载响应时间曲线

从图可以看到，PVFS 系统调用的平均响应时间随着负载测试水平的增加而增加得较快，在负载测试水平为 480Ops/s 时已经达到了饱和，而 DCFS 的平均响应时间在负载水平增加时变化幅度相对较小。在负载水平 160Ops/s 到 480Ops/s 的负载水平范围内，PVFS 的全局响应时间为 11.314ms，DCFS 的全局响应时间远小于 PVFS，为 2.059ms，仅为 PVFS 的 18.2%。从表中可以看到，PVFS 的写响应时间要小于 DCFS，这是因为 PVFS 的写操作完成后不需要修改元数据，而 DCFS 需要修改文件长度值，因此 DCFS 在存储服务器完成写操作后需要给元数据服务器发送元数据请求，比 PVFS 多了一次网络开销，而且在负载综合测试中文件读写请求传送的字节数较小，网络开销占了很多比重，因此在本节的测试中，DCFS 写的相应时间大于 PVFS 写响应时间，另外，我们看到，PVFS 的 stat 系统调用的响应时间远大于 DCFS，这是因为 PVFS 写操作后不修改文件长度值，元数据服务器在处理 stat 操作时必须给存储服务器发消息来统计文件长度。 从表 6.10 可以看到，DCFS 所有元数据系统调用的响应时间都小于 PVFS，并且在负载增加时元数据系统调用响应时间的增加的幅度较小，这说明了 DCFS 在元数据服务器上采用的技术，如元数据缓存等可以极大地提高文件系统元数据操作的处理能力。

6.5 小结

在本章中，作者首先介绍了曙光 4000L 的体系结构和软件结构，然后描述了 DCFS 文件系统协议的实现。作者定义了机群文件系统读写性能的参数，在曙光 4000L 上对 DCFS 文件系统进行了测试并和 PVFS 进行了对比，试验结果表明，DCFS 在大文件读写、小文件写性能上优于 PVFS，并具有良好的客户端可扩展性和服务器端可扩展性。在第 6.4 节中，作者定义了元数据性能参数：吞吐率和元数据可扩展性，同样地，在曙光 4000L 上进行了测试并与 PVFS 进行了比较，可以看到，由于 DCFS 采用了多元数据服务器的结构和元数据缓存等优化策略，使得 DCFS 的元数据性能在所有情况下都优于 PVFS。

第七章 结论

机群作为当前高性能计算机主流的体系结构，越来越多的应用程序运行在机群系统上，但对于数据密集型等应用而言，由于当前缺少好的机群文件系统，限制了这些应用程序的性能，而且应用程序对机群文件系统提出了更高的需求[DOE02]，这些需求包括性能、可扩展性、可管理性等，本文对机群文件系统的设计进行了讨论，首先论述了应用程序对机群文件系统的需求以及网络、存储系统对机群文件系统的影响，总结了机群文件系统设计的若干关键问题，然后针对机群文件系统服务器设计的几个问题进行了研究，包括机群文件系统体系结构、元数据服务器目录操作管理和元数据一致性协议，最后，本文介绍了DCFS（曙光机群文件系统）的设计和实现，给出了 DCFS 在曙光 4000L 上性能测试的结果。在本章，作者将对本文的内容进行总结，然后给出机群文件系统进一步研究的方向。

7.1 本文工作的总结

本文总结了机群文件系统设计的目标和关键技术，并针对其中几个问题进行了研究。本文的主要工作如下：

设计良好的机群文件系统结构可以充分地发挥底层硬件的性能，使机群文件系统具有良好的扩展性、可管理性，本文第三章首先对机群文件系统的体系结构进行了总结，比较了客户/服务器和基于共享设备的两种机群文件系统结构，并根据曙光 4000L 的体系结构选择了客户/服务器结构，同时指出为了适应机群系统不断扩大的规模和机群系统中访问特性各异的各种应用，需要合理地对文件系统的资源进行划分，为此，作者提出了多文件系统卷的文件系统结构，将机群文件系统资源划分为多个文件系统卷，可以根据每个卷上运行的负载特点进行配置，该结构具有简单易于实现、可扩展、易管理、灵活的特点。在第三章中，作者还对文件系统卷中两个组成部分——存储服务器和元数据服务器结构进行了研究：（1）为了使文件系统存储空间易于扩展并发挥存储服务器并行处理的能力，DCFS 每个卷中的存储服务器按网络存储分组进行组织。在第三章中，作者提出了网络磁盘分组的分析模型，对网络存储分组的组织形式进行了分析，指出影响读写性能的因素包括磁盘和网络的硬件性能、存储服务器软件结构以及分条参数的选择；（2）作者在第三章中对元数据服务器的组织和元数据的分布与映射策略进行了讨论，首先比较了两种元数据服务器的组织结构，给出了 DCFS 选择分布存储结构的理由，然后针对分布存储元数据服务器结构中元数据的分布和映射策略进行讨论，指出合理的元数据分布粒度需要在负载平衡和减少元数据操作开销间进行权衡，为了适应 DCFS 多文件系统卷的特点，作者提出了可调粒度的元数据分布策略，使得用户可以根据应用程序的模式灵活选择文件系统卷的元数据分布粒度。

目录操作在文件系统的操作中占了很高的比例，目录操作的性能直接影响了文件系统的性能，为了提高机群文件系统元数据服务器上目录处理能力，作者在第四章对目录操作中的两个问题进行了研究：（1）元数据目录缓存管理；（2）大目录优化。许多机群文件系

统采用了独立的元数据服务器服务于元数据操作请求，而这种独立元数据服务器的结构使得我们可以根据目录缓存的特点设计服务器目录缓存管理方法。作者首先对元数据服务器目录缓存的两种结构进行了比较，试验结果表明元数据服务器的 LOOKUP 缓存和 READDIR 缓存应当采取相互独立的结构，针对这种结构，作者通过试验发现，客户端目录缓存和元数据服务器上的 LOOKUP 目录缓存和 READDIR 缓存构成了一个多级的目录缓存结构，元数据服务器上的 LOOKUP 缓存和 READDIR 缓存表现出了不同的特性，作者根据 LOOKUP 缓存和 READDIR 目录缓存的特性提出了目录缓存的管理方法，试验表明该管理方法可以提高目录缓存命中率。为了提高线性的目录文件组织方法在大目录时的性能，在第四章中，作者和本研究小组成员合作对大目录优化进行了研究，我们首先比较了 B-树和动态 HASH 两种常用的大目录管理算法，指出了 GFS 和 GPFS 采用的动态 HASH 技术中的缺点，然后提出了 LMEH 动态 HASH 的目录管理算法，在 DCFS 上的试验表明，该方法较线性的目录管理算法有较大的提高。

DCFS 文件系统卷采用了多元数据管理器的结构，这种结构在提高了元数据性能和可扩展性的同时也增加了元数据一致性的维护的难度。作者在第五章中首先分析了相关研究中元数据一致性协议，指出元数据一致性协议的设计与文件系统结构、元数据服务器的实现密切相关，然后结合 DCFS 元数据分布和元数据缓存策略，基于数据库中分布式事务的技术给出了元数据一致性协议，该协议保证了元数据一致性，并采取了优化策略减少协议的开销，分析表明其增加的开销是可以接受的。

我们在曙光 4000L 上设计并实现了 DCFS，第六章给出了 DCFS 文件系统协议设计与具体实现以及机群文件系统性能评价的方法。机群文件系统的性能评价需要选择合适的负载类型、定义合理的性能参数，在第六章中，我们从机群文件系统读写性能、元数据操作性能和综合负载评价三个方面对 DCFS 进行了性能评价，其中定义了读写带宽性能和元数据吞吐率的可扩展性度量。曙光 4000L 上的测试表明，DCFS 与类似结构的 PVFS 文件系统相比，在读写性能上，DCFS 除了在小文件的读带宽性能上比 PVFS 差，在其余情况下 DCFS 的读写性能优于 PVFS；由于 DCFS 采用了多元数据服务器结构和元数据缓存，在元数据性能上要远优于 PVFS；在综合负载测试中，由于 DCFS 在元数据服务器上采取的优化技术使其在全局时间度量上远远小于 PVFS。这些结果表明，DCFS 在存储服务器和元数据服务器上采取的技术，如多元数据服务器结构、服务器端缓存等是可行的。

7.2 后续工作

后续的研究工作可以在如下几个方面进行：

（1） 负载驱动的自适应机群文件系统研究。当前机群/分布式文件系统并不能够适合所有的应用模式，负载驱动的自适应机群文件系统的研究旨在根据不同的应用模式来调整文件系统以适合该应用模式，使应用性能最优。负载驱动的自适应文件系统的研究包括对机群文件系统 trace 的研究和自适应的文件系统结构、技术的研究：（a）只有当机群文件系统的配置，包括机群文件系统的结构、缓存的策略等与机群系统中应用程序的访问模式相一致时，才能充分发挥机群文件系统的性能。许多研究者对不同使用环境下的应用程序的访问特性进行了研究[Bak91][Ous85][Ros00][Ell03]，我们注意到，在这些研究中，更多地关注应用程序读写特性的研究，缺少对元数据

访问特性以及服务器“看到”的访问特性的研究，另外，DCFS 等文件系统采用了元数据服务器和存储服务器分离的结构，需要研究这种结构的存储服务器和元数据服务器表现出的访问特性，这些访问特性可以作为机群文件系统优化、自适应文件系统设计的依据；（2）作者在本文中提出了多文件系统卷的机群文件系统结构，可以根据应用程序的访问特性配置文件系统卷，但在当前的实现中，每个文件系统卷的配置是静态的，如何在系统运行的过程中自动地调整文件系统的参数是一个有趣而困难的课题。自适应机群文件系统不仅在文件系统的结构上是动态的，即可以根据应用程序的访问模式对文件系统的结构进行重组，而且，在文件系统语义、文件系统客户端和服务器的优化策略上都可以根据应用程序的特性进行选择。

（2） 大规模机群文件系统的研究。在高性能计算机领域，机群系统的规模不断扩大，T. M. Ruwart[Ruw03]在 FAST 2003 上的报告对几个大实验室的文件系统需求进行了分析，可以看到许多应用对大规模机群文件系统提出了要求，大规模机群文件系统的研究包括大容量机群文件系统、目录树巨大的机群文件系统以及支持大量客户节点的并发访问等内容，这给现有的机群文件系统的结构提出了巨大的挑战。DCFS 后续的研究可以在现有系统的基础上，分析现有技术在大规模系统的应用环境中的不足，进一步提出相应的关键技术。

参考文献

[Ale97] A. D. Alexandrov, M. Ibel, K. E. Schauster and C.J. Scheiman. Extending the Operating System at the User Level: the Ufo Global File System, In Proceedings of 1997 USENIX Technical Conference, Jan 6-10, 1997, Anaheim, California, USA.

[And95] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roseli, and R. Y. Wang. Serverless Network File Systems. 15th ACM symposium on Operating System Principles, Dec.1995.

[And02] D. C. Anderson, J. S. Chase, A. M. vahdat. Interposed Request Routing for Scalable Network Storage. ACM Journal on Transactions on Computer Systems (TOCS), Vol. 20, No. 1, Feb. 2002.

[Att94] C. R. Attanasio, M. Butrico, C. A. Polyzois, S. E. Smith. Design and Implementation of a Recoverable Virtual Shared Disk. IBM Technical Report no. RC19843, Nov. 1994.

[Bak91] M. G. Baker, J. H. Hartmann, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. The measurement of a Distributed File System. In the Proceeding of the Thirteenth ACM Symposium on Operating Systems Principles, Pacific Grove, CA, October 1991, pp. 198-212.

[Bar98] J. Barkes, M. R. Barrios, F. Cougard, P. G. Crumley, D. Marin, H. Reddy, T. Thitayanun. GPFS: A Parallel File System. IBM International Technical Support Organization, SG24-5165-00, April 1998.

[Bod03] N. J. Boden et al., Myrinet: A Gigabit-per-second Local Area Network, IEEE Micro, 1995, Vol.15, No.1, pp29-36.

[Bra02] P. J. Braam. The Lustre Storage Architecture. 2002.

[Bre00] P. T. Breuer, A. Marín Lopez and Arturo Gar. The Network Block Device. Linux Journal. Issue 73, Posted on Monday, May 01, 2000.

[Bur00] R. C. Burns. Data Management in a Distributed File System for Storage Area Networks. Ph.D dissertation, University of California, Santa Cruz, March 2000.

[Cal00] B. Callaghan. NFS Illustrated, Addison-Wesley. April 2000

[Car00] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters', Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, Oct. 2000, pp.317-327.

[Che94]P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, RAID: High-Performance Relible Secondary Storage, ACM Computing Surveys, Vol.26, No. 2, 1994. 145-188

[Che96] S. Chen, D. Towsleg. A Performance Evaluation of RAID Architectures. IEEE Transactions on Computers, Vol. 45, No. 10, October 1996, pp.1116-1130.

[DAFS01] DAFS Collaborative. DAFS: Direct Access File System Protocol. 2001.

[Dev95] M. Devarakonda, A. Mohindra, J. Simoneaux and W. H. Tetzlatt. Evaluation of Design Alternatives for a Cluster File System. In Proceedings of 1995 USENIX Technical Conference, New Orleans, LA, Jan. 1995, pp.35-46.

[DOE01] DOE National Nuclear Security Administration and the DOD National Security Agency. Statement of Work: SGS File System. 2001.

[Ell03] D. Ellard, J. Ledlie, P. Malkani, M. Seltzer. Passive NFS Tracing of Email and Research Workloads. Preceeding of USENIX conference on File and Storage Technologies, 2003, pp. 203-217.

[EMC03] EMC White paper. Leveraging Networked Storage for Your Business. March 2003.

[Fan04] Z. Fan, J. Xiong, J. Ma. A Failure Recovery Mechanism for Distributed Metadata Servers in DCFS2. Accepted in 7th International Conference on High Performance Computing and Grid in Asia Pacific Region (HPC Asia 2004), Jul. 22-24, 2004.

[Far00] M. Farley. Building Storage Network. McGraw-Hill, 2000.

[Fen01] 冯军。机群文件系统性能优化中的关键问题研究.硕士学位论文,中科院计算所,2001 年 6 月

[Flo89] R. A. Floyd and C. S. Ellis. Directory Reference Patterns in Hierarchical File Systems. IEEE Transactions on Knowledge and Data Engineering. Vol.1 No. 2, June 1989.

[Fro96] K. W. Froese, R. B. Bunt. The Effect of Client Caching on File Server Workloads. In Proc. of the 29th Hawaii International Conference on System Sciences, Jan. 1996 , pp. 150-159

[Geo99] P. Geoffray, L. Prylli and B. Tourancheau, BIP-SMP: High Performance Message Passing over a Cluster of Commodity SMPs, In SC99: High Performance Networking and Computing Conference, IEEE, Nov. 1999.

[Gra79] J. N. Gray. Notes on Data Base Operating Systems. In R. Bayer, R. M. Grayham, and G. Seegmuller(eds.), Operating Systems: An Advanced Course, New York, Springer-Verlag, 1979, pp.393-481.

[Gro96] E. Grochowski and R. Hoyt. Future Trends in Hard Disk Drives. IEEE Trans. On Mag, Vol.32, No.3, May 1996, pp.1850-1854.

[Guy90] R. G. Guy, J. S. Heidemann, W. Mak,, T. W. Page Jr., G. J. Popek, D. Rothmeier, Implementation of the Ficus Replicated File System. In Proceedings of the USENIX Conference, Anaheim, California, Jun. 1990

[Hae83] T. Haerder. Principles of Transaction-Oriented Database Recovery. ACM Computer Survey, Vol.15, No.4, Dec. 1983, pp.287-317.

[Han01] 韩冀中。机载 SAR 实时嵌入式成像系统设计研究。博士学位论文,中科院计算所，2001 年 6 月。

[Has98] R. L. Haskin, Tiger Shark-a scalable file system for multimedia. IBM Journal of Research and Development, Vol. 42, No.2, March 1998, pp.185-97.

[Har94] J. H. Hartman, J. K. Ousterhout. The Zebra Striped Network File System. Proceeding of the 14th Symposium on Operating System Principles, Asheville, North Carolina, United State, 1994, pp. 29-43.

[He01] 贺劲，史小冬，吕毅。曙光机群文件系统 DCFS 总体设计。NCIC 技术报告,TR-FS-01-0813。

[He02-1] 贺劲，徐志伟，孟丹，马捷，冯军。基于高速通信协议的 COSMOS 机群文件系统性能研究. 计算机研究与发展.2002 年第 2 期

[He02-2] 贺劲。机群文件系统性能与正确性研究。博士学位论文，中科院计算技术研究所，2002 年 6 月。

[He02-3] 贺劲，孟丹。曙光机群文件系统 DCFS 负载平衡策略。计算机工程与应用，2002 年。

[He03] 贺劲，吴思宁等。网络文件系统中的元数据存取优化研究。微电子学与计算机，2003 年

[Hen96] J. L. Hennssy and D. A. Patterson..Computer Architecture A Quantitative Approach. Morghan Kaufmann Publishers, Inc., second edition, 1996.

[Hit00] D. Hitz. A Storage Networking Appliance. TR2001.Network Appliance, Inc. Oct. 2000.

[How88] J. H. Howard. A Overview of the Andrew File System. In the Proceedings of UESNIX Winter Conference, Dallas, Texas, February 1988, pp.23-26.

[Hwa98] K. Hwang and Z. Xu. Scalable Parallel Computing: Technology, Architecture, Programming. WCB/McGraw-Hill Inc, 1998.

[Ifi01] InfinibandSM Trade Association, InfinibandTM Architecture Specification Volume 1, Jun. 2001.

[ISO96] (ISO/IEC) [IEEE/ANSI Std 1003.1, 1996 Edition] Information Technology — Portable Operating System Interface (POSIX®) — Part 1: System Application: Program Interface (API) [C Language]. ISBN 1-55937-573-6.

[Ioz03] iozone, http://www.iozone.org/

[Ji00] M. Ji, E. W. Felten etc. Archipelago: An Island-Based File System for High Availability and Scalable Internet Services. In Usenix Windows System Symposium, August 2000.

[Jia01] 蒋观海。CA 端到端的企业存储管理解决方案。http://www0.ccidnet.com/news/observe/2001/06/20/54_49644.html

[Jog00] P. Jogalekar, M. Woodside. Evaluating the Scalability of Distributed Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 6, Jun. 2000, pp. 589-603.

[Jon00] T. Jones, A. Koniges, R. K. Yates. Performance of the IBM General Parallel File System. Proceedings of the International Parallel and Distributed Processing Symposium, May 2000.

[Kal97] M. Kalyanakrishman, R. K. Iyer and J. U. Patel, Reliability of internet hosts: a case study from the end user's perspective. In: Proceedings of the sixth international conference on computer communications and networks, IEEE, Sep. 1997

[Lee93] E. K. Lee, R. H. Katz. An Analytic Performance Model of Disk Arrays. Proceeding of the 1993 ACM SIGMETRICS conference on Measurement and Modeling of Computer Systems, Santa Clara, California, USA, 1993, pp. 98-109.

[Lig99] W. B. Ligon III and R. B. Ross. An Overview of the Parallel Virtual File System, Proceedings of the 1999 Extreme Linux Workshop, June, 1999.

[Lu04] 吕毅等。Fsbench：一个机群文件系统基准程序。计算机工程，2004 年。

[LVM02] LVM HOWTO. http://www.die.net/doc/linux/HOWTO/LVM-HOWTO.html.

[Ma01] 马捷。基于 SMP 结点的机群通信提供关键技术的研究，博士学位论文,中科院计算技术研究所，2001 年 6 月。

[Mat97] J. N. Matthews, et al. Improving the Performance of Log-structured File Systems with Adaptive Methods. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, Sant Malo, France, Oct. 1997, pp.238-251.

[Moh94] A. Mohindra, M. Devarakonda. Distributed Token Management in Calypso File System. In Proceedings of the IEEE Symposium on Parallel and Distributed Processing, New York, 1994.

[Mry03] Myricom Inc., The GM Message Passing System, Http://www.myri.com, 2003.

[Mum94-1] L. B. Mummert, J. M. Wing and M. Satyanarayanan. Using Belief to Reason About Cache Coherence. CMU Technical Report, CMU-CS-94-151.

[Mum94-2] L. B. Mummert and M. Satyanarayanan. Variable Granularity Cache Coherence. ACM Operating Systems Review, Jan. 1994, Vol.28, No.1, pp.55-60.

[Mum94-3]L. B. Mummert, et al. Using Belief to Reason About Cache Coherence. In Proceedings of the Symposium on Principles of Distributed Computing, August 1994.

[Mum96] L. B. Mummert. Exploiting Weak Connectivity in a Distributed File System. Ph.D. dissertation. Computer Science Division, School of Computer Science, Carnegie Mellon University, Dec. 1996.

[Net03] Netperf, http://www.netperf.org/netperf/NetperfPage.html

[NFS89] NFS: Network File System Protocol Specification. RFC 1094, March 1989.

[NFS95] NFS Version 3 Protocol Specification. RFC 1813, Jun. 1995.

[NFS03] Network File System (NFS) version 4 Protocol. RFC 3530, Apr. 2003.

[Nie96] Galley: A New Parallel File System for Parallel Applications. Ph.D. dissertation, Department of Computer Science, Dartmouth College, Hanover.

[Ope04] http://www.openafs.org

[Ous85] J. K. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer and J. G. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In the Proceeding of the Tenth ACM Symposium on Operating Systems Principles. Orcas Island, WA, Dec. 1985, pp.15-24.

[Ozs99] M. T. Ozsu, P. Valduriez. Principles of Distributed Database Systems. Prentice-Hall, INC. 1999

[Pat88] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks(RAID). In ACM SIGMOD Conference, June 1988, pp. 109-116.

[Pat89] D. Patterson. P. Chen, G. Gibson and R. Katz, Introduction to Redundant Arrays of Inexpensive Disks(RAID), In Proc. IEEE Conf. on Data Engineering. Los Angeles, CA, Apr. 1989.

[Pat02] D. Patterson. A New Focus for a New Century: Availability and Maintainability >> Performance. Proceeding of the Conference on File and Storage Technologies (FAST'02), 2002.

[Pre99] K. W. Preslan etc. A 64-bit, Shared Disk File System for Linux. Sixteenth IEEE Mass Storage Systems Symposium, 1999.

[Pre00] K. W. Preslan etc. Implementing Journaling in a Linux Shared Disk File System. In Proceeding of the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies. 2000.

[Ree96] B. Reed and D. D. E. Long. Analysis of Caching Algorithms for Distributed File Systems. Operating Systems Review, Vol. 30, No. 3, July 1996, pp. 12–21.

[Rei03] Reiserfs. http://www.namesys.com.

[Rie99] E. Riedel. Active Disks – Remote Execution for Network-Attached Storage. Ph.D thesis, November 1999, CMU.

[Rob99] D. Robinson. The advancement of NFS benchmarking: SFS 2.0. In Proceedings of the 13th USENIX Systems Administration Conference (LISA '99), pp.175-185.

[Ros90] D. S. H. Rosenthal. Evolving the Vnode Interface. In Proceedings of USENIX Summer Conference, Anaheim California, June 1990, pp.107-117

[Ros00] D. Roselli, J. R. Lorch, and T. E. Anderson. A Comparison of File System Workloads. Proceeding of 2000 USENIX Annual Technical Conference, San Diego, California, USA, Jun. 18-23, 2000.

[Ruw03] T. M. Ruwart. Storage on the Lunatic Fringe. Proceeding of the Conference on File and Storage Technologies (FAST'03), 2003.

[Sch98] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. VLDB Journal: Very Large Data Bases, Vol.7, No.1 1998, pp. 48-66,.

[Sch02] F. Schmuck, R. Haskin. GPFS: A Shared-Disk File system for Large Computing Clusters. Proceeding of the Conference on File and Storage Technologies (FAST'02), 2002, pp.231-244.

[SFS01] Standard Performance Evaluation Corporation (SPEC). SFS 3.0, 2001.

[She86] A. B. Sheltzer, R. Lindell, G. J. Popek. Name Service Locality and Cache Design in a Distributed Operating System. In Proc. 6th Int. Conf. on Distributed Computing Systems, pages 515--523, Cambridge, Massachusetts, May 1986, IEEE.

[Shi92] K. W. Shirriff and J. K. Ousterhout. A Trace-Driven Analysis of Name and Attribute Caching in a Distributed System. In USENIX Conference Proceedings, Winter 1992.

[Sin97] P. K. Sinha. Distributed Operating System: Concepts and Design. IEEE PRESS, 1997.

[Sis02] Sistina software Inc. White Paper. The Logical Volume Manager (LVM). Apr. 2002

[Sol97] S. R. Soltis. The Design and Implementation of a Distributed File system based on Shared Network Storage. Ph.D dissertation, University of Minnesota.

[Sta97] T. Stabell-Kul. Security and Log Structured File Systems. ACM Operating Systems Review, Vol.31, No.2, April 1997, pp.9-10.

[Sun03] 孙凝辉。曙光 4000L 超级服务器，中科院计算所创新重大项目验收总结报告，2003。

[Sxd02] 史小冬。分布式文件系统高可用问题研究。博士学位论文，中科院计算技术研究所，2002 年 6 月。

[Tan03] Rongfeng Tang, Dan Mend, Sining Wu. Optimized Implementation of Extendible Hashing to Support Large File System Directory. 2003 IEEE International Conference on Cluster Computing, Hong Kong, Dec. 1-4, 2003.

[Tia03] 田颖。分布式文件系统中的负载平衡技术研究。硕士论文，中科院计算技术研究所，2003 年 6 月。

[Tha96] R. Thakur, W. Gropp, and Ewing Lusk, An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In Proc. of the 6th Symposium on the Frontiers of Massively Parallel Computation, Oct. 1996, pp. 180-187.

[Tha99] R. Thakur, W. Gropp, Ewing Lusk. On Implementing MPI-IO Portably and with High Performance. In Proc. Of the Sixth Workshop on I/O in Parallel and Distributed Systems, May 1999, pp. 23-32.

[Thi92] D. Thiebaut, J. L. Wolf, I. S. Stonc. Synthetic Trace for Trace-Driven Simulation of Cache Memories. IEEE Transactions on Computers. Vol. 41, No.4, Apr. 1992, pp. 388-410.

[Top03] http://www.top500.org/lists/2003/11/trends.php

[T10-00] T10 Working Draft NCITS TBD-200X, Project 1355D, Revision 3, Oct.1, 2000.

[Vah99] U. Vahalia 著，聊鸿斌等译。UNIX 高级教程：系统技术内幕. 清华大学出版社, May 1999。

[VIA97] Virtual Interface Architecture Specification Version 1.0. December 16, 1997.Compaq Computer Corp., Intel Corporation, Microsoft Corporation.

[Wan97] R. Y. Wang, T. E. Anderson, M. D. Dahlin. Experience with a Distributed File System Implementation. Technical report, University of California, Berkeley, Computer Science Division, June 1997.

[Wan99] R. Y. Wang. Improving the I/O Performance and Correctness of Network File Systems. Ph.D thesis, Spring 1999, UC Berkeley.

[Wjy99] 王建勇。可扩展的单一映象的文件系统.博士学位论文,中科院计算所,1999 年 6 月。

[Wjy02] J. Wang, Z. Xu. Cluster File System: A Case Study. Journal of Future Generation Computer Systems, Vol. 18, No.3, Jan. 2002, pp.373-387.

[Wil96] J. Wilkes, R. Golding, C. Staelin and T. Sulliran. The HP AutoRAID Hierarchical Storage System. ACM Transactions on Computer Systems, Vol. 14, No.1, Feb. 1996, pp.108-136.

[Wil93] D. L. Willick, D. L. Eager et al. Disk Cache Replacement Policies for Network Fileservers. In Proceedings of the 13th International Conference on Distributed Computing Systems, May 1993, pp. 2-11.

[Wu02] 吴思宁。Linux 机群文件系统的研究与实现，微电子学与计算机，2002 年第 5 期。

[XFS03] XFS detailed design – Directory structures. http://oss.sgi.com/projects/xfs/design_docs/

[Zha01] Zheng Zhang, C. Karamanolis. Designing a Robust Namespace for Distributed File services. 20th IEEE Symposium on Reliable Distributed Systems, New Orleans, LA, USA, Oct. 28-31, 2001.

[Zc99] 张驰。可扩展的单一映象机群文件系统的设计与实现.硕士学位论文,中科院计算所,1999 年 6 月。

[Zhk99] 张华开。机群系统上可扩展 I/O 界面的实现.硕士学位论文,中科院计算所,1999 年 6 月

[Zho01] Yuanyuan Zhou and J. F. Philbin. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. Proceedings of the 2001 USENIX Annual Technical Conference, Jun. 25-30, 2001.

致 谢

首先，我要感谢我的博士导师徐志伟研究员，感谢徐老师在学术研究工作中给予的指导以及生活中的照顾。徐老师严谨求实的研究作风，渊博的学识令我受益非浅。徐老师为国家的事业无私奉献的精神也令我深深敬佩，我以自己能有幸接受徐老师的教诲而深感荣幸。

我要感谢孙凝晖和孟丹研究员，衷心感谢他们在这三年中工作上的指导协助，是他们出色的领导为我们提供了优良的学习和研究环境，使我的论文能够顺利完成。

感谢智能中心文件系统研究小组全体成员，他们是贺劲、熊劲、唐荣锋、史小冬、吕毅、范志华和曹立强。文件系统小组团结、自由、民主的工作和学习氛围以及与各位共同的研究经历使我印象深刻，我取的每一个进步与各位的帮助是分不开的。贺劲博士作为我在文件系统领域的引路人，在学习和工作中都给予了我极大的帮助，其对问题的敏锐洞察和分析的能力值得我好好学习；与熊劲在 DCFS 文件系统实现过程的合作令人难忘；与唐荣锋关于大目录优化以及与范志华关于元数据一致性协议的讨论与合作总是令人愉快。

感谢智能中心所有同仁在这三年中给予的关心与热心帮助，尤其感谢高阿姨、王阿姨以及钟大爷，是他们的工作使得我可以整天“以室为家”。

感谢所有帮助与关心我的朋友，特别是卢隽、邱国铭、张宁、蔡涛、赵一三、孙越栋，感谢他们十几年来对我的一贯支持与帮助。

感谢计算所教育处各位老师的辛勤工作。

最后，我要感谢我的父母和弟弟，是他们在背后的支持和鼓励使我在二十多年的求学生涯中一步步顺利地走过来。

作者简历

姓名：吴思宁　　性别：男　　出生日期：1975 年 4 月 27　　籍贯：广西宾阳县

2000.9——2004.3　　中科院计算所智能计算机研究开发中心博士研究生
1997.9——2000.7　　北京工业大学计算机学院，获硕士学位
1993.9——1997.7　　南京理工大学计算机系，获学士学位

【攻读博士学位期间发表的论文】

1. 吴思宁，贺劲，熊劲，孟丹，DCFS 机群文件系统服务器组关键技术研究，微电子与计算机，2003 年第 6 期
2. 吴思宁，贺劲，熊劲，孟丹，DCFS 机群文件系统服务器组的设计与实现，2002 全国开放式分布与并行计算学术会（DPCS2002），2002 年 10 月 26－29 日，武汉
3. 吴思宁.Linux 机群文件系统的研究与实现，微电子学与计算机，2003 年第 5 期
4. 吴思宁，熊劲，贺劲，“文件系统快速路径解析的方法 ”，发明专利一项，受理号：2002-12-10，02157880.X
6. *Jin Xiong, Sining Wu, Dan Men, Ninghui Sun, and Guojie Li*， Design and Performance of the Dawning Cluster File System，2003 IEEE International Conference on Cluster Computing, December 1-4, 2003, Hong Kong
7. Rongfeng Tang, Dan Mend, Sining Wu. Optimized Implementation of Extendible Hashing to Support Large File System Directory. 2003 IEEE International Conference on Cluster Computing, Hong Kong, Dec. 1-4, 2003,

【攻读博士学位期间参加的科研项目】

1. 国家 863 高技术研究发展计划资助重点课题：“高性能计算机及其应用系统——曙光 3000 系统”（编号 863-306-ZD01-01），参与 COSMOS 机群文件系统从 AIX 平台到 Linux 平台的移植。
2. 国家 863 高技术研究发展计划“高性能计算机及其核心软件”专项重大项目：“面向网格的超级服务器——曙光 4000”，参与曙光 4000L 中 DCFS 机群文件系统研制。

机群文件系统服务器关键技术研究

作者：吴思宁
学位授予单位：中国科学院计算技术研究所

相似文献(10条)

1. 学位论文 贺劲 机群文件系统性能与正确性研究 2002

计算机存储系统,特别是相对慢速的外存储系统一直是影响计算机整体性能的"瓶颈".目前,机群系统已经逐渐成为超级计算机与超级服务器的主流结构,但外存储系统仍然是它进一步提高性能与可靠性的主要障碍.近年来,随着网络技术的飞速发展,基于机群节点间高速互连网络的机群文件系统已成为解决机群外存储系统问题的有效策略之一.该文结合曙光机群文件系统DCFS的设计与实现,对如何有效提高机群文件系统性能、保证应用的正确执行方面进行了较有成效的研究,主要内容包括：以COSMOS文件系统为原型系统,研究了机群文件系统中通信子系统对整体性能的影响;研究了机群文件系统结构优化对元数据操作的性能影响;以支持MPI-IO并行计算为目标,研究了在支持客户端缓存情况下,如何实现文件系统客户端缓存一致性语义的协议.

2. 会议论文 吴岷.刘炜.沈美明.温冬蝉 机群系统中合作缓存的研究 2001

文章主要涉及到的内容包括在(TH-CluFS)机群文件系统中将磁盘缓存整个文件内容和内存缓存文件块内容结合为统一的合作缓存;采用加强的Hash算法解决缓存块定位以及Hash冲突;利用写通过的策略解决缓存一致性问题.

3. 学位论文 刘炜 可扩展机群文件系统的研究 2001

在该文中,对机群文件系统的分布式命名机制、文件迁移机制和全局缓存方法等领域进行了研究,取得了有价值的研究结果.作者将其中一些成果应用于一个机群系统上的文件系统的设计与实现,取得了比较好的效果.该文的主要贡献包括以下几个方面:(1)提出了一种新的分布式命名机制,提供了单一的文件系统映像.(2)提出了一种基于机群环境的自治文件迁移算法.(3)提出了一种基于内存-磁盘的统一缓存模型,提供了更大的缓存空间.(4)提出了一种基于文件粒度的分级复制方法.(5)设计和实现了一个机群文件系统TH-CluFS.TH-CluFS具有比Coda文件系统更好的性能,基本上接近于NFS的性能,在某些方面还超过的NFS.

4. 学位论文 何军 机群文件系统合作缓存技术研究 1998

随着机群系统计算性能的不断提高,机群系统的应用程序对文件系统性能的需求不断增加.现有的网络文件系统或分布式文件系统已无法满足应用需求,因此需要对进一步提高机群文件系统性能的技术进行深入研究.当前计算机硬件技术的发展,尤其是高速开关式局域网络网络性能的快速提高,使得利用缓存间相互合作来减少磁盘访问的合作缓存技术成为一种非常有潜力的提高机群文件系统的性能的技术.

5. 期刊论文 吴思宁 Linux机群文件系统的研究与实现 -微电子学与计算机2002,19(5)

机群是当前高性能计算领域的主流体系结构,随着Linux操作系统的成熟和pc机性能的提高,用Linux和pc节点构造的机群系统越来越多地应用于高性能计算领域.I/O子系统和机群文件系统的性能对整个机群系统的性能有重要的影响,本文在曙光3000 cosmos[6,10]机群文件系统的基础上,针对曙光3000 LLinux机群系统设计实现了基于Linux的机群文件系统cosmos-L,对cosmos-L进行了性能测试,并与PVFS文件系统进行了比较.

6. 学位论文 欧新明 可移植的单一映象机群文件系统的研究与实现 2000

机群系统文件一极的单一映象虽然很多人做过研究,但是上前为止还没有一个被广泛应用的商业版本,这主要是因为这些系统都不具备可移植性,只能应用于某种特定的操作系统平台上.另一方面,传统的分布式文件系统由于受文件卷要领的束缚,文件不能在各个机器之间自由分布,因此并不适合机群系统的需要.CluFS文件系统是一个为机群系统设计的 可移植的单一文件系统.该文论述了CluFS的设计思想和实现中采用的几个关键技术,并对 其性能进行分析.

7. 学位论文 李晖 基于日志的机群文件系统高可用关键技术研究 2005

近年来机群系统以其低成本、高性能而逐渐成为高性能计算的主流平台、作为解决机群系统外存储瓶颈上的有效手段的机群文件系统因此得到了很大的发展。一个机群文件系统必须要满足机群计算环境的需要，为应用提供高性能、可扩展、高可用的文件服务。由于机群文件系统本身结构复杂，实现复杂而且整个系统规模很大，这些因素就决定了对高可用技术的依赖。本文将研究基于日志的机群文件系统高可用的关键问题以及解决策略，同时给出了一些评价方法以及具体的评测结果。具体内容以及研究成果如下：

(1)研究了基于日志的机群文件系统高可用技术的关键问题。本文分析了不同类型的机群文件系统的高可用需求以及高可用技术，对机群文件系统高可用相关的概念进行了介绍，描述了机群文件系统高可用领域的研究内容，并在分析的基础上提出了基于日志的机群文件系统高可用技术，分析了其中的关键问题，给出了相应的解决策略，并对其正确性和完备性给予了证明。

(2)实现了DCFS2机群文件系统高可用模块。作为文中策略的一个实际应用，本文给出了DCFS2机群文件系统高可用的设计与实现技术，给出了系统中利用日志来保证机群文件系统一致性的方法。主要内容包括：以DCFS2机群文件系统为原型系统，研究了单一以及多个元数据服务器下如何使用日志来保证文件系统的一致性：研究了机群文件系统日志对元数据操作的性能影响：研究了客户端的高可用问题。

(3)提出了机群文件系统高可用性的分级的定义。机群文件系统的高可用性的高低一直缺乏有效的定性或定量的分析方法，由于软件系统不能象硬件系统那样进行定量分析，我们根据机群文件系统的应用模式，将影响机群文件系统高可用性的因素进行分析，以机群文件系统的故障因素和恢复目标因素为线索，采用分级的方法对机群文件系统高可用性进行了定义，提出了机群文件系统高可用性的分级的定义。

(4)对基于日志的高可用技术进行了评价。目前在高可用技术的评价上尚没有完善的评价体系，本文从功能性，正确性，性能，恢复时间等多个方面对基于日志的高可用技术进行了评价，并给出了各种情况下的具体的测试结果。 文中还讨论了下一步的研究方向，包括多节点故障恢复等方面。

8. 学位论文 李剑宇 基于对象存储的机群文件系统数据通路关键技术研究 2007

近年来机群系统凭借良好的可扩展性、可用性以及极高的性价比成为高性能计算机和超级服务器的主流结构.然而,磁盘性能的改善远远落后于CPU处理速度、内存性能、互连网络带宽的提高,使得I/O系统成为严重制约机群系统性能提高的瓶颈.机群文件系统作为缓解机群系统I/O瓶颈的重要手段,具有重要的研究意义.

为了提供可扩展的文件服务,大规模机群文件系统的发展趋势是数据通路与元数据通路相分离、基于对象存储的网络存储系统和多元数据服务器的体系结构.

针对这种结构的大规模机群文件系统,本文重点研究了其中的几个关键问题,包括I/O性能的关键优化技术、多负载的高效支持以及数据的高可用.本文的主要贡献在于:1)设计和实现了基于对象存储的大规模机群文件系统LionFS,重点研究了在这种新的存储体系结构下,I/O性能的关键优化技术,包括:异步化、直接递送的数据传输机制以及基于前端负载访问信息的预取技术,测试表明,由于直接递送的方式减少了传输过程中客户端和对象设备的内存拷贝,使得读写性能都有比较大的提高,分别为:读24%,写28%,而且采用预读技术后系统的数据通路达到并发流水,使读带宽增长了70%;2)提出了一种类会话的I/O访问机制以及面向文件访问请求的全局调度机制.本地环境通过连续的文件布局并配合CFQ磁盘调度策略能够很好地改善多个并发顺序流的聚合性能.然而,机群文件系统基于网络传输的数据访问方式使得存储服务器的驱动方式与负载构成不同于本地环境,限制了上述策略的优化效果.为此,我们设计了类会话的I/O访问机制以充分发挥本地系统ext3数据块预留机制的优化效果,并在此基础上提出了面向文件访问请求的全局调度机制以缓解多负载并发访问磁盘时的相互干扰.性能测试表明:综合使用上述策略能够明显改善机群环境下多个并发顺序流的聚合性能.3)提出了一种基于复制技术的数据高可用机制,针对复本一致性、故障记录和数据恢复、在线恢复等问题,该机制扩展了标准的文件锁协议来保持单复本的Unix文件共享语义;利用轻量级的故障记录机制来降低日志操作的性能开销并通过重放更新的修复策略在故障排除后进行高效的数据修复;最后借助文件冻结技术实现在线修复,满足数据高可用的需求,性能数据表明:该机制在故障发生后仍然能够保持系统具有较好的性能,而且在故障排除后能够快速完成数据的恢复.

9. 会议论文 邢晶.马捷 机群文件系统元数据访问行为分析与优化研究 2006

机群系统已成为目前构建高性能计算机的主流系统,而提高机群系统中负责提供IO服务的机群文件系统性能是构建高性能计算机的关键.本文以提高机群文件系统的元数据处理性能为出发点,通过采用统计和踪迹剖析两种方式来分析元数据访问行为,得到的元数据访问规律如下:1,元数据只读操作在机群文件系统的元数据实现中所占比例高达64.6%;2,涉及目录对象的操作在元数据实现中占较高比例,达17%;3,对元数据进行操作包含一个冗长的定位过程.利用元数据访问行为分析的结论,我们对LionFS元数据服务器的实现进行了优化,测试结果表明优化后的元数据处理性能比优化前提高了30%,证明了该研究的有效性.

10. 学位论文 王毓 机群文件系统的数据放置策略研究 2009

近年来机群系统凭借其良好的扩展性、可用性以及高性价比成为当前高性能计算机的主流体系结构。而由于磁盘性能的提升速度远远落后于CPU、内存以及通信网络带宽的发展速度，从而使得I/O子系统越来越成为机群系统的性能瓶颈。设计高性能的机群文件系统是缓解这一问题的重要手段。实际应用中所产生的数据可能在系统中不同的存储节点之间造成访问和存储的不均衡，从而导致某个节点成为系统的性能瓶颈，降低了系统的整体性能。高效的数据放置策略通过将数据在系统中所有存储节点之间合理有效地分布存放，充分利用所有节点的聚合带宽，提高系统的吞吐量，从而成为缓解或者消除I/O子系统性能瓶颈的重要技术。

本文总结了影响机群文件系统I/O性能的数据放置的几个关键问题，并结合国内外相关研究工作，提出了有效的解决方案。主要的贡献如下：

1)提出了适用于目前机群系统中衡量节点负载的二元组方法及负载均衡机制。这种方法主要考虑节点的I/O访问负载，其次考虑节点的磁盘空间利用率。对节点的I/O访问负载失衡和磁盘空间利用饱和的情况进行区分，采取不同的数据迁移策略，使系统中的负载处于动态近似均衡状态，从而可以充分利用系统中所有节点的带宽，提高系统的I/O性能。

2)提出了存放文件时基于负载的概率选择法。在选择文件的存放位置时，首先考虑节点的I/O访问负载。当访问负载不均衡时，以节点的访问负载大小所占比重的概率选择该节点进行存放；当访问负载近似均衡时，以节点的磁盘空间利用率所占比重的概率选择该节点进行存放。这种方法可以有效地支持多复本机制，同时还支持single、stripe两种不同的文件模式。实验测试表明，通过这种方法选择文件的存放位置，可以提高系统的吞吐量。

3)提出了机群系统发生伸缩时的数据迁移策略。对节点失效的情况，首先基于负载进行快速的数据恢复，然后再进行负载均衡的数据迁移操作。对节点加入的情况，首先进行负载均衡的数据迁移操作，再将新节点投入到创建文件时的使用。通过采取不同的数据迁移机制，在保证系统数据可靠性的同时，又能尽量减小数据迁移对系统I/O性能的影响。通过这种方法进行节点伸缩时的数据迁移，可以提升节点动态变化时系统的吞吐量。

关键词：机群文件系统；数据放置；负载；负载均衡；数据迁移