

中国科学院计算技术研究所

博士学位论文

蓝鲸分布式文件系统的资源管理

姓名：黄华

申请学位级别：博士

专业：计算机系统结构

指导教师：许鲁

20050501

## 图目录

图 1.1 TOP500 中按照体系结构分类的统计 .....	2
图 1.2 混合网络存储的结构 .....	4
图 1.3 网络存储系统的分类 .....	5
图 1.4 Storage Tank 的系统结构.....	6
图 1.5 Lustre 的系统结构 .....	7
图 1.6 Panasas 存储机群的体系结构.....	8
图 1.7 CXFS 的数据和元数据流 .....	8
图 1.8 DCFS 的系统结构 .....	9
图 1.9 蓝鲸大规模网络存储结构示意图 .....	11
图 2.1 BWFS 的总体结构.....	17
图 2.2 DLRM 的层次和功能调用 .....	18
图 2.4 BWFS 的模块部署和通信 .....	20
图 2.5 GLA 和 RG.....	21
图 3.1 ext2/3 的物理磁盘布局.....	24
图 3.2 资源组的磁盘布局.....	27
图 3.3 位图块的数据结构.....	28
图 3.4 统计块的数据结构.....	28
图 3.5 位图块、统计块和超级块的树状统计关系 .....	29
图 3.6 索引节点的三级指针位图模型 .....	31
图 3.7 索引节点位图块的数据结构 .....	32
图 3.8 间接块的数据结构.....	33
图 3.9 超级块的数据结构.....	34
图 4.1 元数据的总体架构.....	38

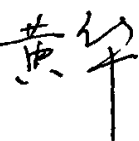
图 4.2 分片存储.....	42
图 4.3 活跃元数据绑定到三个 MS.....	44
图 6.2 蓝鲸分布式文件系统写性能 .....	56
图 6.3 重新建立文件映射的过程 .....	57
图 7.1 BWFS、NFS 大文件写的聚集带宽.....	68
图 7.2 BWFS、NFS 大文件读的聚集带宽.....	68
图 7.3 BWFS 创建小文件的聚集吞吐率 .....	70
图 7.4 BWFS 删除文件/目录的聚集吞吐率 .....	70
图 7.5 BWFS 在同一目录下创建/删除空文件的性能.....	72
图 7.6 BWFS 与某专用 NAS 以及 NFS 服务器性能对比.....	73

## 表目录

表 6.1 蓝鲸分布式文件系统读性能 .....	55
表 6.2 蓝鲸分布式文件系统写性能 .....	55
表 7.1 BWFS、NFS 大文件写的聚集带宽 .....	67
表 7.2 BWFS、NFS 大文件写的 CPU 利用率 .....	67
表 7.3 BWFS 聚集写性能相对存储节点数量的加速比 .....	67
表 7.4 BWFS、NFS 大文件读的聚集带宽 .....	67
表 7.5 BWFS、NFS 大文件读的 CPU 利用率 .....	67
表 7.6 BWFS 聚集读性能相对存储节点数量的加速比 .....	68
表 7.7 解开内核源代码包消耗的时间 .....	69
表 7.8 删除内核源代码目录树消耗的时间 .....	69
表 7.9 解开内核源代码包的聚集吞吐率 .....	69
表 7.10 删除内核源代码目录树的聚集吞吐率 .....	69
表 7.11 编译内核源代码的耗时 .....	71

## 声 明

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


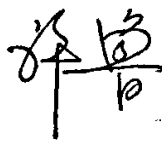
作者签名：

日期：2005.5.30

## 论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名： 导师签名： 日期：2005.5.30

## 第一章 引言

从二十世纪四十年代世界上诞生第一台电子计算机开始,人类就在不断使用信息技术改变着自身的生活生产方式,生产力获得了飞跃式提高。虽然如今的信息技术已经获得了巨大成功,但是人们从来就没有停止过追求更高性能的计算机,这种追求使得我们的信息技术仍在不断向前发展。

计算机技术的发展一直近似遵循着摩尔定律:计算机的性能每 18 个月翻两番,而且摩尔定律似乎还会继续发挥作用。半导体技术使得计算机中央处理器(CPU)的计算能力获得了巨大进步,网络技术使得计算机之间的通信能力有了迅猛发展。但是由于受到机械部件的限制,计算机外部存储器性能的发展速度远远落后于其它计算部件,造成了计算机系统的外部数据传输能力与计算能力之间的巨大差距,也因此制约了计算机整体性能的发挥。

人们在认识到存储子系统对整个计算机系统性能发挥的重要性之后;不断使用各种方式来提升存储系统本身的性能,也提出了各种新型的存储结构,以期缓解计算能力与数据输入/输出能力方面的矛盾。蓝鲸大规模网络存储系统以及其中包含的蓝鲸分布式文件系统就是在此背景下发展起来的。

本章首先介绍了目前高性能计算的发展趋势,随后总结了外部存储子系统目前面临的诸多挑战,之后介绍了目前国内外关于分布式文件系统的一些最新研究成果,最后介绍了蓝鲸大规模网络存储系统相关组成部分的概况。

### 1.1 高性能计算的发展

TOP500[1]是一个成立于 1993 的国际组织,致力于追踪和预测当今世界高性能计算的发展趋势,每年都会发布两次世界上最高性能超级计算机的排名。图 1.1 说明了按照体系结构分类的 2004 年 11 月发布的超级计算机前 500 强的分布情况(如无特别说明,下文有关 TOP500 的数据均以 2004 年 11 月的数据为准)。从图中我们可以看到采用机群(Cluster)结构的高性能计算机系统近几年来发展迅猛,从 1998 年 11 月仅有两台机群结构的超级计算机上榜,到 2000 年 11 月的 28 台,到 2002 年 11 月的 94 台,再到如今的 294 台,超过了其它结构的系统,占据了 TOP500 的 58.8 %的份额。出现这种现象的原因一方面是由于机群结构可以充分利用现有的、通用的、廉价的计算机部件(比如 CPU、内存、网络等)组成计算节点,用较低投资就可以实现高性能计算;另一方面是网络技术的进步使得机群之间的通信性能有了长足进步,千兆以太网/万兆以太网、Myrinet、InfiniBand 等技术使得网络的带宽获得了巨大的提高,通信延迟也进一步减小,从而使得机群中节点之间的通信不再

是系统瓶颈（以千兆以太网和 Myrinet 进行机群间互连的系统在 TOP500 中分别占有 35.2% 和 38.6%，从发展趋势来看，这两个数据还在增长）。

从 TOP500 统计的安装数量上看，IBM 和 HP 公司继续在高性能计算机领域处于领先地位，国内曙光公司的“曙光 4000A”和联想公司的“深腾 6800”也分别占据一席之地，分别位列第 17 位和 38 位，说明我国在高性能计算机领域也有了突飞猛进的发展。同时我国有 17 个高性能计算机系统进入了 TOP500，显示了国内强劲的高性能计算的需求。

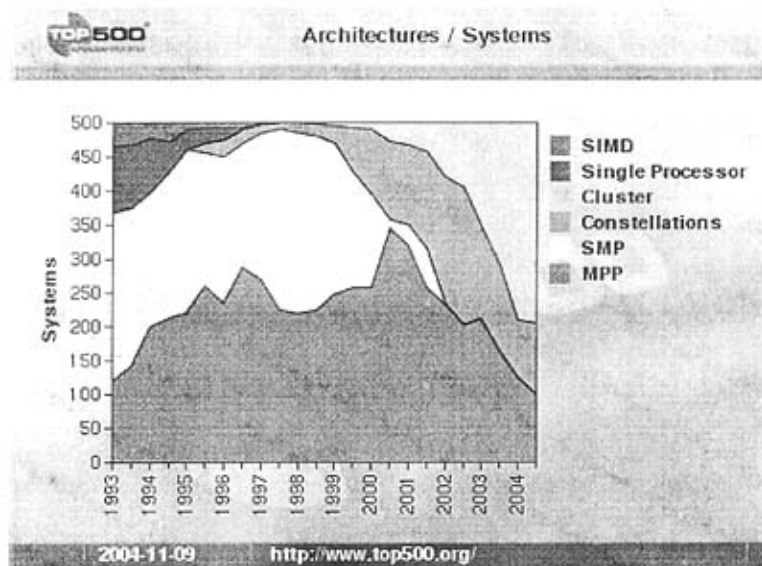


图 1.1 TOP500 中按照体系结构分类的统计（来自 TOP500）

各种应用在利用高性能计算机进行运算或者数据处理的时候，总是需要将原始数据从外部存储子系统读入，最后又将计算结果返回到外部存储子系统。外部存储子系统的性能对整个系统性能的发挥至关重要。下面我们了解一下外部存储子系统的概况。

## 1.2 外部存储子系统的发展

从最初容量数百 K 字节的一张软盘，到现在 500G 字节左右的一块硬盘，计算机外部存储设备的容量已经有了巨大的提升，同时其数据传输速度也有了飞跃式提高。但现今作为主要计算机外部存储设备的磁盘，由于受到内部机械部件的限制，其寻道时间已经难以再有明显提升，造成在容量不断变大的同时，数据传输速度提升缓慢。因此人们设计了多种方式来改观这种局面，主要是从磁盘本身、磁盘阵列、网络存储等方面入手。

### 1.2.1 硬盘技术

硬盘传输一组数据的时间包括三个部分：磁头移动的时间（寻道时间，Seek Time）、盘片转动的的时间、介质读写的时间。磁头移动主要依靠机械臂的平行移动，受到机械部件的物理限制，其速度提高的很缓慢。现在最少的平均寻道时间大约在 3.6ms 左右[2]，难以再有新的飞跃。盘片转动的的时间取决于盘片转动的速度，现在盘片的转速大致有 5400rpm, 7200rpm, 10000rpm, 15000rpm 等。同样受到机械部件的限制，而且考虑系统散热问题，转速也难有大幅度提高。虽然介质的传输速度有了不少提高，但是整个磁盘的数据传输速度受到前两者的限制，跟磁盘的容量相比，提升的速度缓慢。

### 1.2.2 磁盘阵列技术

既然单个磁盘的数据传输性能难有较大提升，人们便设计了利用多个磁盘并发提供数据传输功能来提高数据传输能力的磁盘阵列（Disk Array）。廉价冗余磁盘阵列（RAID）[3][4]采用廉价的、独立的磁盘组成一个磁盘阵列，互相协同，共同提供数据存储和传输功能。RAID 通过数据校验技术提高磁盘的可用性，通过分组并发存储技术提高阵列的数据传输能力。

磁盘阵列技术有效地提升了数据传输性能。但是由于受到单个磁盘阵列所能连接的磁盘数量的限制，以及磁盘阵列与主机之间数据传输链路的带宽限制等，单个磁盘阵列的容量和传输性能还是难以满足日益增长的数据存储和高性能计算需求。同时在机群环境下，多个计算节点需要并发共享访问同一数据集，单个磁盘阵列只能向某一服务器提供数据服务，难以满足需求。

### 1.2.3 网络存储技术

单个磁盘或者磁盘阵列直接连接到某台服务器上的方式，一般称之为直连存储（Direct Attached Storage, DAS）。在 DAS 方式下只有连接了磁盘的服务器能够使用存储资源，系统难以进行容量与性能的扩充，而且当服务器出现故障时，整个存储系统也随之变得不可用。

随着网络技术的发展，计算机网络的传输性能有了很大提高。千兆以太网(Gigabit Ethernet)[5]、万兆以太网(10Gb/s Ethernet)[6]、光纤通道(Fibre Channel)[7]、InfiniBand[8][9]等网络的传输速度已经远远超过了单个磁盘的数据传输能力，因此借助网络进行数据传输与存储的网络存储体系结构也随之成为了如今的主流发展方向。

附网存储（Network Attached Storage, NAS）[10][11]是连接在网络上的一种专用文件服务器，通过远程文件访问协议（比如 NFS[12][13]，CIFS[14]等），向网络中的其它节点提供共享文件的服务。NAS 可以有效地降低各个



使用远程文件服务节点的负载,集中管理网络中的存储资源,在各个节点之间实现文件级数据共享。NAS 还可以实现在线扩充系统容量,保证业务的连续性。NAS 服务器是专用的文件服务器,在可靠性、文件服务性能、成本等方面都比 DAS 有较大优势。但是 NAS 也存在一定的缺陷,比如 NAS 服务器的容量毕竟有限,单个 NAS 服务器的性能在大规模机群环境中存在瓶颈,进行文件传输的过程中占用大量局域网带宽等。

存储区域网 (Storage Area Network, SAN) [15] 是以数据存储为中心,通常采用光纤通道互连存储设备与应用节点的一种存储结构。SAN 将许多存储设备通过光纤交换机连接在一起,共同提供数据存储服务,容量的可扩展性极好。同时通过冗余的光纤交换机可以提高整个系统的可用性。SAN 形成独立的存储网络,不占用局域网的带宽,提供较高性能的块级数据访问接口。虽然 SAN 具有很好的性能和扩展性,但是其本身无法提供文件级数据共享,架设和管理成本较高,因此较适合于大型企业级应用。

SAN 过高的成本给它的普及使用带来很大的障碍。人们在寻找新的办法,既要有 SAN 的高性能与可扩展性,又要有较低的成本和较好的互操作性。以太网技术的发展使其数据传输能力有了数量级的提高,目前普及使用的千兆以太网已经与光纤通道具有相同数量级的性能,万兆以太网更是大大超过了光纤通道的数据传输能力。以太网成熟的技术、低廉的成本、良好的互操作性使得基于 IP 的网络存储日渐兴旺起来。目前互联网小型计算机系统接口 (Internet Small Computer Systems Interface, iSCSI) [16]、互联网光纤通道协议 (Internet Fibre Channel Protocol, iFCP) [17]、基于 IP 的光纤通道 (Fibre Channel over IP, FCIP) [18] 和 NBD (Network Block Device) [19] 等协议,使得在以太网络中进行大规模、高性能数据传输得以可能。一般将这类结构称为“IP SAN”,因为它们利用了 SAN 的结构,却又利用 IP 网络传输数据。

很多企业应用为了既能享受 SAN 的高性能和可扩展性,又能兼顾成本因素,采用了混合多种结构的拓扑。如图 1.2 所示。

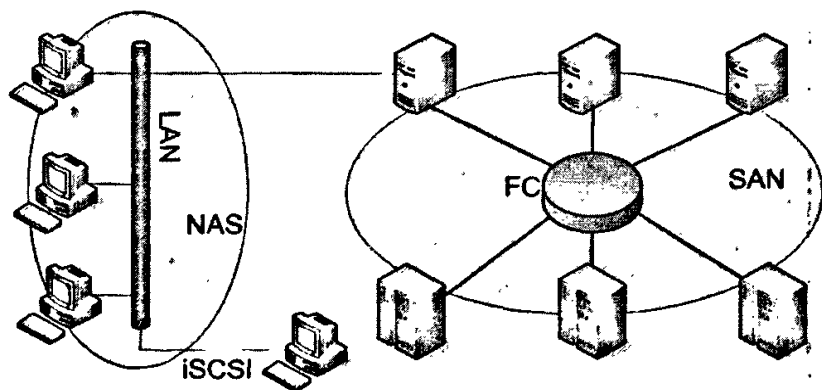


图 1.2 混合网络存储的结构

如果根据访问协议和传输通道划分,我们可以将上述几种结构作如图 1.3 所示的划分,它显示了各种结构之间的相互差异。

	以太网	光纤通道
文件级	NAS	NAS+SAN 混合结构
块级	IP SAN	SAN

图 1.3 网络存储系统的分类

机群环境下的应用程序为了能够访问外部存储子系统,以及进行数据共享和操作同步,一般都需要借助分布式文件系统。分布式文件系统是所提供的功能以及性能是影响这些应用的重要因素。下面我们将介绍一些分布式文件系统的概况。

### 1.3 分布式文件系统

无论采用何种存储结构和传输通道,存储子系统的最终任务是向机群中的计算节点提供数据存取业务。高性能的存储设备、高带宽的传输网络只是实现高性能存储子系统的物理基础,高性能的分布式文件系统才是关键因素。这一小节简单介绍当前主流分布式文件系统的概况。

- 1) 网络文件系统 (Network File System, NFS)。二十世纪八十年代初 Sun Microsystems 公司最早在 Solaris 操作系统中实现了 NFS,并将 NFS 协议规范公开,使之成为了 UNIX/Linux 环境中最为广泛使用的分布式文件系统,并被移植到包括 Windows 在内的许多其它操作系统中。目前广泛使用的 NFS 第三版本 (NFSv3) [20]采用一个服务器对应多个客户机的模式,使用无状态连接 (stateless)、基于时间的弱同步语义。NFS 服务器不保存任何状态信息的这种特性,使得 NFS 无需处理机群中任何节点的失效。在失效节点重新启动以后,客户端可以重新连接服务器,或者 NFS 服务器重新提供服务,而不会影响原先的文件服务。从文件系统的角度考察, NFS 只是一个文件服务重定向器,和其类似的还有 CIFS 协议。它们本身不参与文件系统的管理,只是将客户端的文件访问请求重定向到宿主文件系统,由宿主文件系统实现真正的文件管理。虽然 NFS 在不断改进,现在一个服务器可以支持更多的客户端,但是由于单个服务器是整个系统的集中点,在容量、性能、可靠性上难以大幅提高,导致 NFS 在

高负荷、大规模海量存储的环境中，显现出明显不足。

- 2) Global File System (GFS) [21][22]。GFS 是由 Steve Soltis 等人在明尼苏达大学设计和实现的机群环境下的分布式文件系统，现在由 Red Hat 公司拥有，是一个开放源码的软件。它吸收了 SMP 结构中处理器访问共享内存的技术，将客户端看作对称的处理器，将存储节点看作共享的内存。所有的存储节点形成共享磁盘的结构，任何客户端都可以均等地访问所有存储节点。GFS 这种“没有服务器”（serverless）的结构，使得其比传统分布式文件系统具有更高的可用性，更好的可扩展性。多个客户端通过基于设备的锁实现相互之间的同步，保证文件系统的一致性。现在的 GFS 是一个基于 SAN 的分布式文件系统，一方面利用 SAN 提供的高性能存储网络，一方面向客户端提供文件级的数据共享。
- 3) General Parallel File System (GPFS) [23]和 Storage Tank[24]。IBM 公司的 GPFS 是一个从 Tiger Shark[25]发展来的支持 SAN 或者 iSCSI 协议的分布式文件系统。它采用共享磁盘的结构，每个客户端可以同等地访问所有的存储设备。GPFS 采用了 serverless 结构，通过分布式锁同步各个节点的操作，保证一致性。Storage Tank 是 IBM 公司另外一个基于 SAN 的跨平台分布式文件系统，它采用客户机/服务器结构，支持多个元数据服务器并发操作。每个元数据服务器管理一个文件集合（fileset），这些集合互相不重叠。文件系统的元数据直接在客户机和存储节点之间通过 SAN 交换，文件系统的元数据通

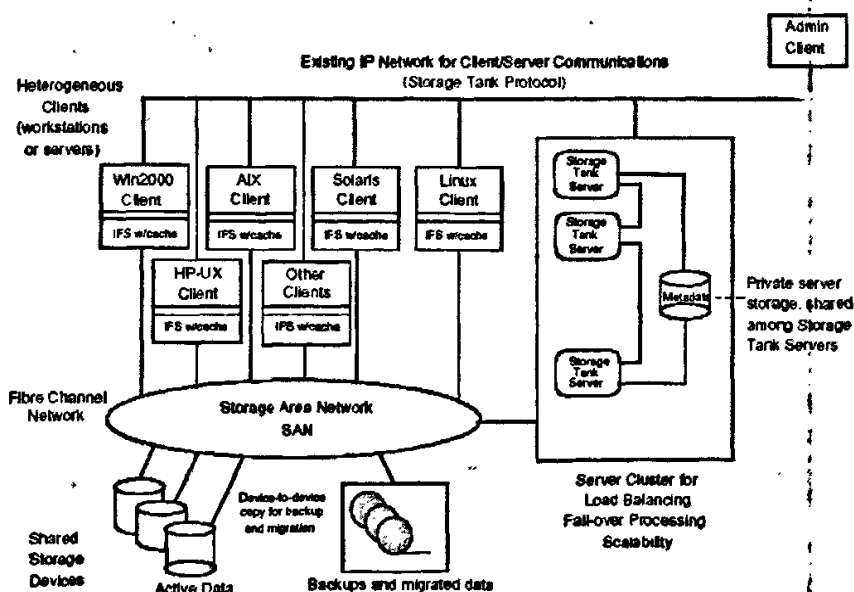


图 1.4 Storage Tank 的系统结构（来自 IBM 公司）

过以太网在客户端和元数据服务器之间交换，这样既可以充分利用 SAN 的高性能数据传输能力，又能利用以太网有效降低系统成本。

图 1.4 显示了 Storage Tank 的拓扑结构。

- 4) Lustre[26]。由 Peter Braam 等人在 Cluster File System 公司开发的 Lustre 分布式文件系统，使用“基于对象的存储设备”（Object based storage devices）[27]，以期能够获得更好的性能、安全性和可管理性。这种基于对象的存储设备，可以是软件模拟的，也可以是硬件直接支持的，具有较大的灵活性。客户端对文件的操作可以使用对象的概念，比如读取某个文件的某个块，而该块的内部映射信息由对象存储来解析。因此客户端的文件数据传输需要较少元数据服务器的参与（仅需要权限验证等），单个元数据服务器可以支持大量的客户端。为了尽量适应现有的环境，Lustre 使用网络中间层软件 Portal 支持异构网络环境。通过将元数据存放在共享的存储设备上，Lustre 使用两个元数据服务器（一个活跃的，一个备份的）来提高系统的可用性。图 1.5 是 Lustre 的系统结构。

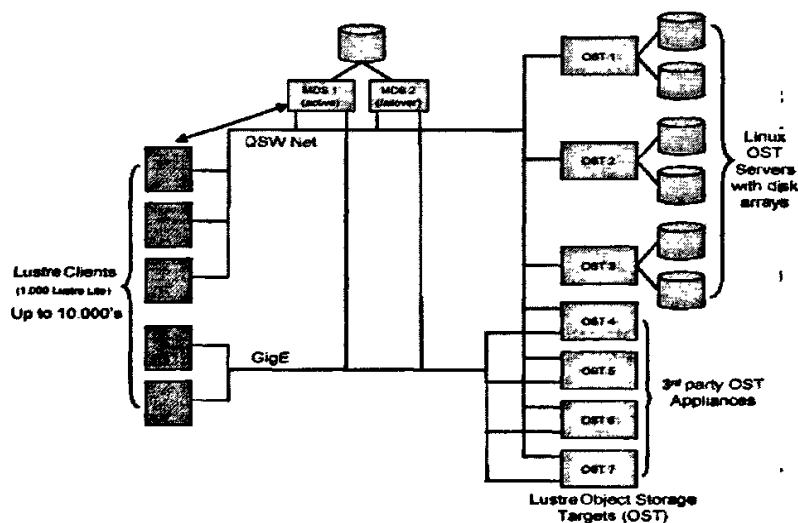


图 1.5 Lustre 的系统结构（来自 Lustre 的白皮书）

- 5) Panasas ActiveScale File System (PanFS) [28]。Panasas ActiveScale Storage Cluster 采用基于对象存储的体系结构，它的存储节点 StorageBlade 和元数据服务器 DirectorBlade 都采用刀片服务器，可以在较小的空间里容纳更多的节点，降低成本。PanFS 是 Panasas 存储机群的核心系统软件，通过在客户端安装的 DirectFLOW 软件，向应用程序提供文件级访问接口。在 PanFS 中，文件系统的数据直接在客户端和存储刀片之间传输，DirectorBlade 只进行身份验证和元数据管理操作，消除数据传输瓶颈。图 1.6 是 Panasas 存储机群的

体系结构。

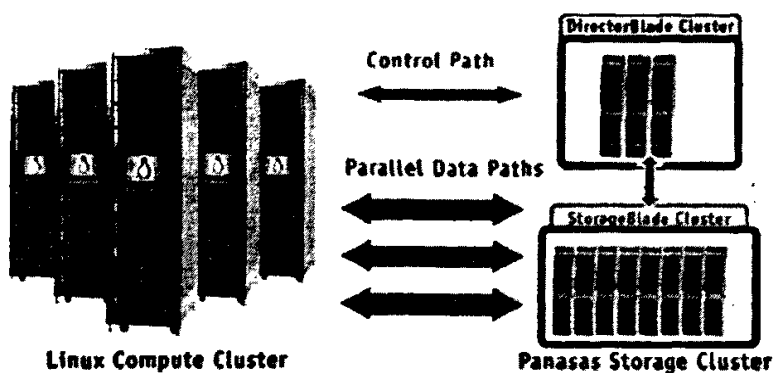


图 1.6 Panasas 存储机群的体系结构 (来自 Panasas 公司)

- 6) CXFS[29]。XFS[30]是 SGI IRIX 操作系统的文件系统，现在已经作为开放源码项目移植到 Linux 操作系统中。XFS 是纯 64 位的文件系统，具有很好的性能和可扩展性。建立在 XFS 基础上、利用 SAN 架构、跨平台的 CXFS 同样具有很好的性能和可扩展性。CXFS 中包含一个活跃的元数据服务器（可以另外配置一个备份元数据服务器），多个客户端通过 TCP/IP 网络与元数据服务器交换元数据，文件系统的数直接就在 SAN 中传输，如图 1.7。

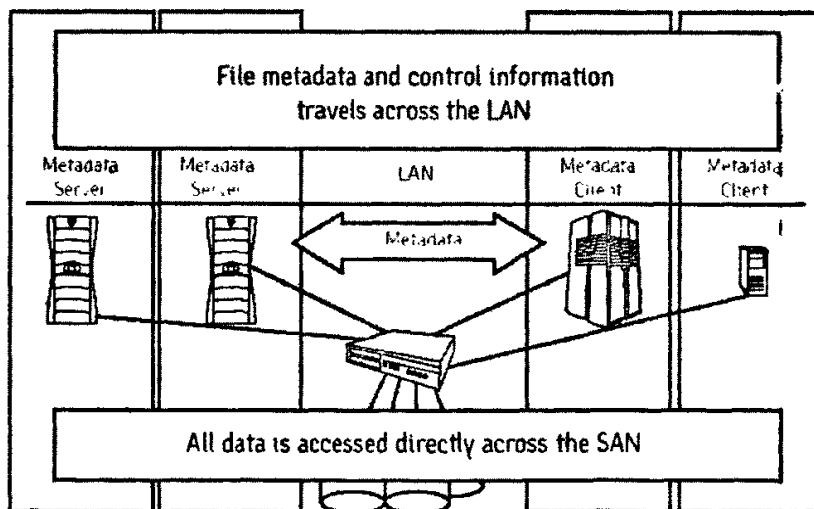


图 1.7 CXFS 的数据和元数据流 (来自 SGI 公司)

- 7) Dawning Cluster File System (DCFS) [31][32]。DCFS 是曙光 4000L 上的机群文件系统，能够管理多个存储服务器，利用多个元数据服务器提供高性能、可扩展性的元数据服务。DCFS 同样将文件系统的数与元数据分开，在客户端与存储服务器之间直接传送数据。DCFS 指定一个“超级管理者”(Super-manager)管理整个文件系统的超级块和根目录，根目录以下的目录分散到多个元数据服务器并

发管理。图 1.8 演示了 DCFS 的系统结构。

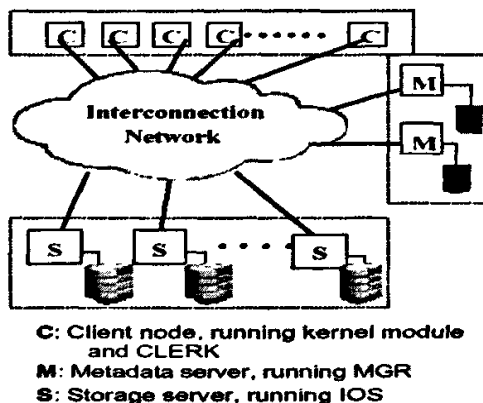


图 1.8 DCFS 的系统结构

#### 1.4 外部存储子系统面临的挑战

高性能机群计算机系统的计算能力得到了空前提高，科学计算和信息处理等应用所需要的数据和产生的数据也在不断膨胀。外部存储子系统在机群计算中扮演的角色越来越重要，它们也面临着诸多挑战。我将这些挑战总结如下：

- 1) 容量 (Capacity)。人类产生数据和记录数据的速度在大幅度增长，人类社会在过去 30 年记录的科学数据量超过了以往 5000 年数据总量之和，而且这种发展趋势还在不断加剧。比如某小型内容过滤系统每天需要记录 1T 字节的原始数据，某石油勘探企业每年大约会积累 30T 字节左右的数据，某高能物理研究所每年积累 1PB 数据。
- 2) 可用性 (Availability)。存储子系统负责提供科学计算和信息处理所需要的原始数据并记录它们产生的结果。应用的每一次计算可能需要持续数个小时，甚至许多天，也可能是积累多年的结果。保证这些数据的可用性是一个突出的问题。
- 3) 可管理性 (Manageability)。大量数据分布式存储在多个节点上，供多个应用程序和多组用户同时使用。有效地管理这些数据可以提高生产效率，还能节省管理成本。同时如何有效地管理数据的生命周期，有效组织数据的多个版本，也是当前面临的挑战之一。
- 4) 性能 (Performance)。在机群环境下，多个计算节点同时访问外部存储设备，需要很高的数据传输性能和较低的数据传输延迟。比如一个 MPEG4[33]格式的流媒体大约需要 1MBps 的数据流，一个流媒体服务提供商为了同时提供一万个在线影片观看服务，就需要大约 10GBps 的数据传输带宽。这样的性能要求给存储子系统提出了严峻

的挑战。

- 5) 可使用性 (Usability)。现今机群的规模越来越大, 曙光 4000A[34] 总共含有 2560 个处理器, 位列 TOP500 榜首的 IBM 的 BlueGene/L beta-System 处理器数量达到了惊人的 32768 个。存储子系统的规模也随着机群规模和本身存储容量的扩大迅速扩大。如何向大规模的机群提供外部存储访问, 以及提高访问的便捷性, 都是现在的存储子系统面临的新的挑战。
- 6) 可扩展性 (Scalability)。如何保证随着存储子系统规模的扩大, 系统的容量和性能也能呈现线性增长甚至超线性增长, 同时又能有效控制系统的管理成本尽量保持原来的水平, 一直是系统设计者需要重点考虑的问题之一。系统良好的可扩展性能够有效地保护用户的投资, 并且随着应用的扩展而轻松扩展。

### 1.5 蓝鲸大规模网络存储系统概况

蓝鲸大规模网络存储系统是由中国科学院计算技术研究所的国家高性能计算机工程技术研究中心承担的国家 863 高科技研究项目。该项目是在充分调研了当前高性能计算机系统, 特别是机群结构的高性能计算机系统的发展趋势下设立的。从 TOP500 的统计结果我们可以看到以及预测机群计算机是未来高性能计算机的走向。同时从上面的分析可以看到, 存储子系统已经成为整个机群环境中最重要的关键因素之一, 它的性能和可扩展性直接决定着应用程序性能的发挥。下面将简单介绍蓝鲸大规模网络存储系统的概况。

蓝鲸大规模网络存储系统的目标是支持配置上千个节点的高性能计算机机群环境, 为机群计算环境提供高性能、可扩展性强、易管理的存储子系统, 其结构如图 1.9 所示。它本身包含了高性能网络存储设备、大规模机群的动态部署技术、虚拟网络存储、分布式文件系统等内容, 下面将逐一介绍。

蓝鲸高性能网络存储设备 (Network Storage Device, NSD) 采用精心设计的硬件体系结构, 包括高性能的 RAID 控制器、超宽的内部总线、优化的网络设备等, 使之能够更好地适应网络环境下的存储需求, 提供高速磁盘访问速度和网络数据传输速度。网络存储设备采用经过精心优化的 Linux 操作系统 (NSDOS), 能够提供更好的数据传输能力, 也能够提供全面的远程设备监控操作, 提高设备的可管理性和可用性。

在机群达到一定规模以后, 机群管理将成为一个费时费力的工作。一个机群可能需要同时运行多个不同的应用环境, 或者轮流运行完全不同的操作系统。蓝鲸服务动态部署系统 (Service on Demand, SonD) [35] 是一套为大规模机群管理设计的动态部署系统, 可以轻松实现整个机群中所有节点的动

态部署, 根据应用环境和用户的需求, 简便快速地为成百上千的计算节点远程安装操作系统、应用环境、用户数据等。

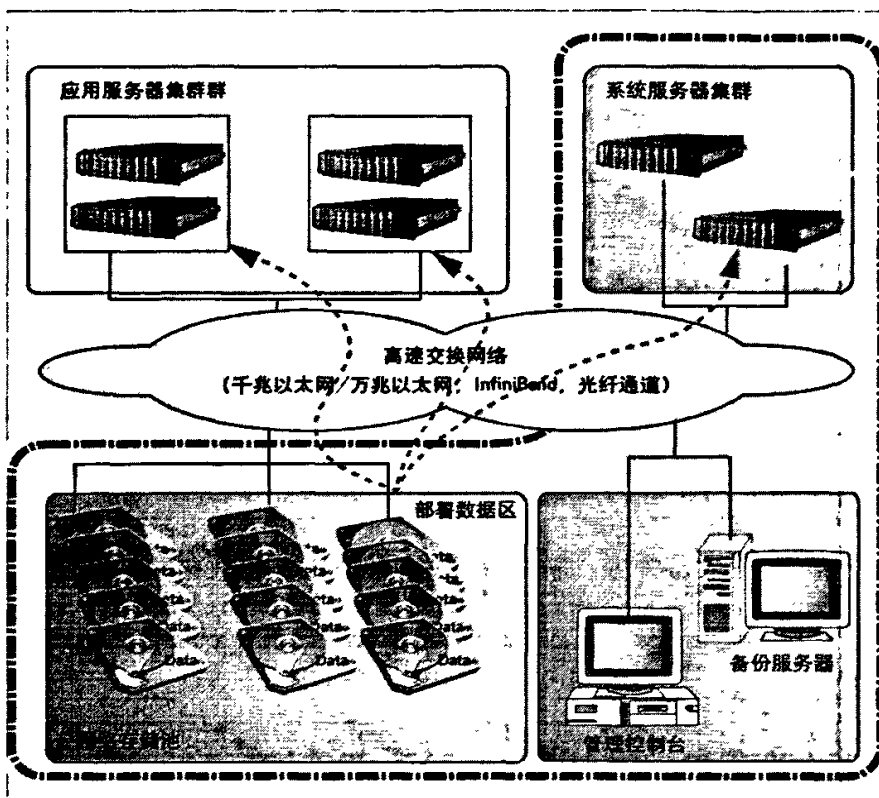


图 1.9 蓝鲸大规模网络存储结构示意图

蓝鲸虚拟网络存储系统[36][37]整合多种异构存储设备, 形成一个巨大的共享网络存储池 (Network Storage Pool)。管理员可以根据用户需要, 任意切分该存储池中的空间, 分配给用户使用。这样可以有效利用连接在一个高速局域网内的所有存储资源, 避免空闲资源的浪费。同时利用 copy-on-write、用户配额等技术, 还可以实现存储空间容量的虚拟化, 即利用较少的实际空间, 提供较大的虚拟空间, 利用每个用户不会使用所有空间的特性, 服务更多的客户。

蓝鲸分布式文件系统 (Blue Whale distributed File System, BWFS) [38][39][40][41]是整个蓝鲸大规模网络存储系统的核心系统软件, 主要向机群结构下的计算节点提供并发的、共享的远程文件访问服务。BWFS 在客户端利用 Linux 的 VFS 和 Windows 的 IFS 机制, 实现内核级文件系统。应用程序通过与文件系统相关的系统调用获得文件服务, 保持了应用程序的二进制兼容性。BWFS 遵循 NFS 的文件访问语义, 任何可以使用 NFS 的应用程序都可以访问 BWFS 提供的远程文件服务。BWFS 采用“带外” (Out-of-Band) 数据传输模式, 文件数据直接在客户端和多个存储设备之间传输, 文件系统



元数据由元数据服务器集群处理,消除数据传输的瓶颈,大大提高了系统的性能和可扩展性。BWFS 还可以在各个元数据服务器之间进行负载平衡,也可以在各个存储服务器之间进行负载平衡。

## 1.6 本文的贡献

本文的研究主要集中在蓝鲸分布式文件系统的资源管理部分。一个优异的资源管理模型和良好的设计实现,是提高分布式文件系统性能和可扩展性的关键因素。在借鉴了已有分布式文件系统的基础之上,在调研了当前国内外机群应用的需求之后,我们设计实现了蓝鲸分布式文件系统。本文的主要贡献主要有如下几点:

- (1) BWFS 的分布式分层资源管理模型 (Distributed Layered Resource Management model, DLRM)。根据系统中各个部分相对资源的不同角色——资源提供者、资源管理者、资源协调者、资源使用者等, DLRM 模型将它们划分成分布的、分层次的多个模块。这些模块逻辑上相互独立,功能上依次调用,形成一个有机的整体。这些逻辑上独立的模块一方面有利于系统的设计实现,另一方面又能根据需要任意部署在系统的任意节点上,实现负载平衡。该模型使得蓝鲸分布式文件系统可以管理多个存储设备,支持多个元数据服务器,采用“带外”数据传输模式,支持众多的文件系统客户端,支持整个系统的动态负载平衡等,是整个 BWFS 的核心。
- (2) 高效的物理存储空间管理。DLRM 模型将分布的物理存储资源虚拟成统一编址的逻辑地址空间,根据存储资源的物理特性划分成多个独立的资源组。每个资源组都有一个资源组管理器,各自独立、并行地管理对应资源组的空间分配情况,提供文件系统对象与逻辑地址之间对应关系的查询等。各个资源组管理器通过一个全局资源组管理器协调相互之间的操作。资源组管理器采用带统计信息的多级索引与动态位图管理数据块和索引节点资源,动态分配索引节点。蓝鲸分布式文件系统的物理存储空间管理可以显著提高空间管理的性能,特别是提供一个持续有效的空间管理性能——从空间使用较少的时候,到空间使用率较高的时候,都能提供较好的性能。
- (3) 全动态元数据绑定。蓝鲸分布式文件系统的所有活跃元数据全动态地分布在各个元数据服务器之间,可以实现元数据服务的动态负载平衡,提供更好的性能和可扩展性。
- (4) 文件系统的资源管理优化。元数据服务器管理文件系统的元数据,比如组织目录结构、组织文件指针、申请/释放存储空间等。元数据

服务器向资源组管理器批量申请/释放存储空间，异步释放空闲空间，在多个存储服务器之间进行分片存储 (striping) [42]、按策略的资源分配、分布式日志、客户端块映射信息缓存等技术。

- (5) 蓝鲸分布式文件系统的性能测试。针对蓝鲸当前的典型应用和未来的潜在应用，我对蓝鲸分布式文件系统进行了一些性能测试。测试结果表明蓝鲸分布式文件系统具有很好的性能和可扩展性，验证了 DLRM 模型的有效性以及上述几个主要工作的成效。同时，希望这些测试结果能够给以后系统的优化提供一些参考点。

## 1.7 论文的组织

本文第一章从分析当前高性能计算机发展的趋势入手，指出机群系统是当前和将来高性能计算机的主流；简要概括了外部存储子系统的发展历史，得出网络存储系统将是未来发展的趋势；然后简要分析了国内外分布式文件系统的主要研究成果和典型产品，总结了存储系统面临的挑战；在此基础上简单介绍了蓝鲸大规模网络存储系统的情况。

第二章主要介绍蓝鲸分布式文件系统的分布式分层资源管理模型的设计，以及该模型如何管理多个存储设备，支持多个元数据服务器，各个模块之间的通信等。

第三章主要介绍资源组管理器如何进行空间分配管理。

第四章主要介绍元数据服务器上如何进行资源管理以及资源管理的优化。

第五章简单介绍系统如何支持多个元数据服务器之间的协同工作。

第六章主要介绍蓝鲸分布式文件系统客户端如何进行有关资源的元数据信息的缓存。

第七章介绍了蓝鲸分布式文件系统的性能测试情况。

最后，对整个论文进行了总结，并给出了将来的研究方向。

## 第二章 分布式分层资源管理模型

随着信息技术的发展,科学计算、信息处理等应用对分布式数据存储提出了更大容量、更高性能等要求。传统的分布式文件系统 NFS 只能利用单个文件服务器的存储资源、计算能力和网络传输能力,其性能和可扩展性受到严重限制,难以满足日益提高的数据处理要求。为此,蓝鲸分布式文件系统提出了分布式分层资源管理模型(Distributed Layered Resource Management Model, DLRM),将数据存储多个存储节点(Storage Node, SN)上,由多个元数据服务器(Meta-data Server, MS)共同管理,采用带外(Out-of-Band)数据传输模式直接应用服务器(Application Server, AS)与存储节点之间传送数据。DLRM 模型根据不同功能将 BWFS 划分成多个层次上的多个模块,分布在系统的各个节点上,平衡各个节点的负载。DLRM 模型还实现了批量申请/释放资源、分片(Stripping)存储等功能,使得 BWFS 可以动态添加存储设备和元数据服务器,同时能够在各个存储服务器和元数据服务器之间实现动态负载均衡。

本章将从介绍模型设计的背景出发,阐述 DLRM 模型的总体框架、模块之间的通信、全局逻辑地址和资源组等。

### 2.1 模型设计的背景

文件系统[43]是管理物理磁盘空间,向应用程序提供基于“文件”(包括普通文件、目录、连接等)对象访问服务的系统软件。对磁盘文件系统来说,其最主要、最重要的资源就是物理存储空间。本文所说的资源管理是指围绕物理存储空间展开的一系列管理操作过程,包括分布式存储空间的编址、存储空间空闲资源管理、文件系统元数据的磁盘布局、存储空间的分配/释放、存储空间的缓存等。在单机磁盘文件系统中,所有这些功能都集中在一个节点上,由单个模块管理。但是在分布式文件系统的环境下,这些功能势必分布在系统中的各个节点上,由不同的模块分别管理,才能充分利用机群中多个节点协同工作的特点,提高系统性能。

#### 2.1.1 模型设计的出发点

蓝鲸分布式文件系统的设计目标是应用在大规模机群环境中,向大量客户端提供并发的、共享的、高性能的文件访问服务。蓝鲸分布式文件系统的资源管理处于核心地位,其设计与实现的优劣直接影响着整个存储系统的性能和可扩展性。因此我们在设计资源管理模型的时候,特别关注以下涉及资

源管理方面的内容:

- 1) 机群环境。蓝鲸分布式文件系统是蓝鲸大规模网络存储系统的核心, 管理海量存储空间, 向上千个客户端提供文件服务。属于同一机群中的这些客户端相互之间的数据安全性相对广域网有较好的保证。在某一时刻, 可能有多种应用同时利用蓝鲸分布式文件的文件共享进行并行计算, 它们对系统有不同的访问模式, 不同的性能需求等。同时, 这些客户端节点可能来自不同的厂商, 具有不同的配置, 无法实现真正的对等关系。机群环境下一般使用特定的存储设备来存储应用程序的数据, 而且由于存储容量需求极大, 一般配置多台存储设备才能满足需求。蓝鲸分布式文件系统必须能够适应多变的应用环境, 兼容更多的设备, 提供更广泛的支持。
- 2) 性能。蓝鲸大规模网络存储系统致力于解决机群计算环境中外部 I/O 能力与计算能力之间的不匹配关系, 努力提升数据访问速度, 消除因为数据传输能力不足带来的性能瓶颈。计算机网络, 特别是以太网技术获得了飞速发展。千兆以太网应用现在已经非常普及, 万兆以太网已经进入成熟阶段, InfiniBand 网络也已经有大量应用。这些网络的数据传输能力有了空前提高, 已经远远超过了磁盘的实际数据传输能力。因此我们利用多个存储节点并发提供数据传输的特性, 提高整个系统的外部 I/O 性能。同时利用多个元数据服务器并发提供元数据服务, 获得更好的元数据服务性能。
- 3) 动态扩展和动态负载平衡。应用的数据量和性能需求不是一成不变, 而是随着时间推移逐步提高的。为了保护用户投资, 也为了适应变化的用户需求, 蓝鲸分布式文件系统需要能够管理一个不断变化的存储子系统。当用户在容量和性能方面提出更高要求的时候, 系统通过添加存储节点、添加元数据服务器、提高网络带宽等措施满足用户的需求。在系统变得越来越庞大的时候, 系统的可扩展性将成为衡量分布式文件系统优劣的一个重要的指标。大型应用一般都会运行比较长时间的连续作业, 这样可能造成整个文件系统出现负载不均衡的现象, 也可能随着时间推移出现“老化”现象[44], 引起性能下降。蓝鲸分布式文件系统需要能够根据系统的运行状况和管理员的指令进行负载平衡, 防止出现“老化”现象。
- 4) 模块化。分布式文件系统是一个大型、复杂的系统级支撑软件, 是应用程序访问大规模存储的基础, 是整个存储子系统的核心。因此它的设计与实现的优劣直接影响整个系统的稳定性、性能、可扩展性等指标。蓝鲸分布式文件系统如何进行模块化划分、层次化的功能调用等设计都对系统的实现具有重要的意义。一个好的设计将使

系统实现变得简单有效，维护、开发、升级的成本降低。这也体现了现代软件工程的思想。

### 2.1.2 相关研究的弱点

Lustre[26]和 PanFS[28]都使用对象存储设备保存文件系统的数据，省却了由文件系统管理元数据的部分功能，但是这样的结构也使得它们不能够兼容原有的以块方式访问的磁盘设备，难以利用 SAN 架构提升系统性能。Storage Tank[24]和 DCFS[31][32]将系统的元数据静态地映射到多个元数据服务器，难以根据系统的负载情况及时调整元数据的映射关系，以便实现动态负载平衡。

## 2.2 DLRM 模型

在综合考虑了蓝鲸大规模网络存储系统当前所面临的问题与挑战之后，在参考了目前流行的分布式文件系统的设计与实现之后，我们设计了 BWFS 的总体结构，如图 2.1 所示。BWFS 采用带外 (Out-of-Band) 数据传输模式，将元数据和文件数据分离：元数据服务器集中处理元数据，数据直接在应用服务器和存储节点之间传输，分别由两种颜色的箭头表示。绑定服务器 (BS, Binding Server) 协调元数据服务器之间的操作，决定活跃元数据在各个元数据服务器之间的分布情况，进行元数据的负载平衡。管理服务器 (AD, Administration server) 负责文件系统的全局管理，同步关键操作。系统中的节点通过高速交换网络连接，例如千兆以太网、光纤通道、InfiniBand 等。

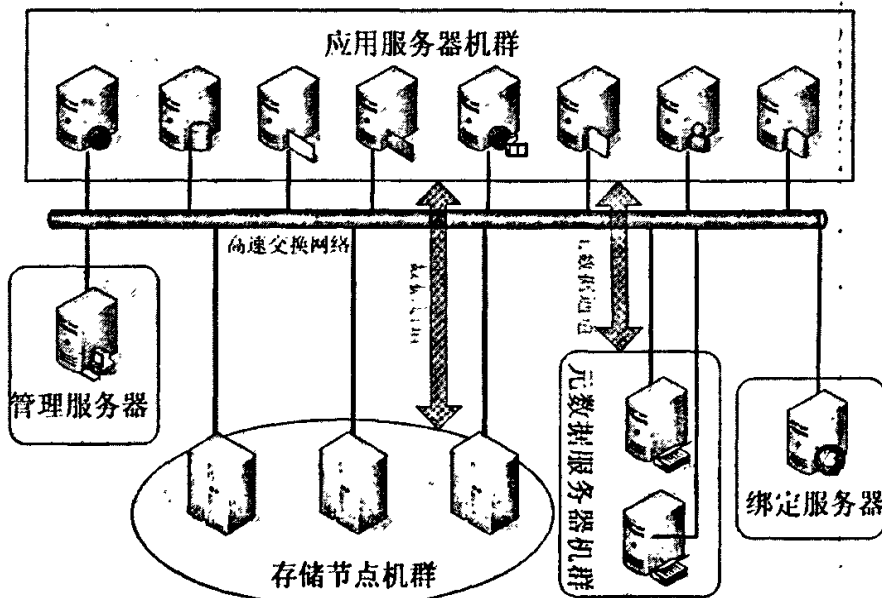


图 2.1 BWFS 的总体结构

图 2.1 演示了 BWFS 的总体结构与数据流向, 我们将此结构称为 BWFS 的“分布式分层资源管理模型”(DLRM 模型)。根据实际情况需要, DLRM 模型可以将多种功能模块安装在同一节点上, 比如管理服务器、绑定服务器和元数据服务器的软件都可以安装在同一节点上。

蓝鲸分布式文件系统没有采用 xFS[45]的无服务器结构, 而是采用了基于客户机/服务器的结构, 主要是基于以下考虑: 机群中的节点未必全是对等的。也就是说机群中的有些节点适合于作为计算节点, 有些阶段适合于作为存储节点, 有些节点作为管理节点。它们的配置不同, 优势不同, 担当的角色就不同。而且由于机群中可能同时进行多个进行计算, 为了避免相互干扰, 也不适于采用类似 xFS 的结构。

为了简化系统的设计和实现, 也为了能够兼容更多的设备, 许多分布式文件系统都将整个系统分为多个层次, 每个层次完成各自的功能, 清晰而有效。Frangipani[46]分布式文件系统采用两层结构, 底层是提供虚拟共享磁盘服务的 Petal[47], 上层是分布式文件系统的实现。得益于此结构, Frangipani 较容易地继承了 Petal 提供的一些关于磁盘的特性, 比如高可用性、动态扩展性等, 文件系统的相关功能也较容易实现。GFS[21][22]将多个存储设备组成一个“网络存储池”(Network Storage Pool, NSP), 然后通过 SAN 向客户端提供统一的共享磁盘访问服务。GPFS[23]客户端通过块设备接口访问连接在 SAN 上的存储设备, 也可以通过“虚拟共享磁盘”(Virtual Shared Disk, VSD)的软件访问连接在某主机上的单独磁盘。

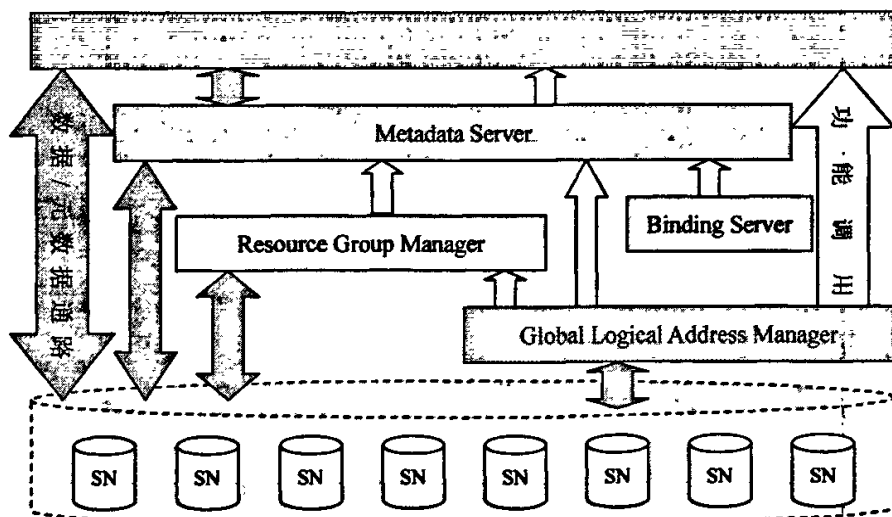


图 2.2 DLRM 的层次和功能调用

DLRM 模型同样采用多层次结构, 如图 2.2 所示。首先 DLRM 模型将多个存储节点虚拟成一个大的共享磁盘 (shared disk), 该磁盘向系统中的其它

节点提供数据存取服务,这部分的功能由全局逻辑地址管理器(Global Logical Address Manager)完成。然后 BWFS 根据需要将逻辑地址空间划分成多个资源组 (Resource Group), 每个资源组由一个资源组管理器 (Resource Group Manager) 管理其存储空间的使用情况。元数据服务器负责组织文件系统相关的元数据, 比如目录结构、文件指针等, 向客户端提供元数据服务。客户端提供文件系统访问接口, 与元数据服务器交换元数据信息, 直接通过虚拟的共享磁盘交换数据。绑定服务器为元数据服务器提供元数据绑定业务, 根据一定的策略在元数据服务器之间实现负载平衡。DLRM 的层次化的结构使得系统的设计和实现变得相对简单, 各个部分逻辑上相互独立, 通过一定的接口实现功能调用。

提供上述功能的各个模块又是逻辑独立的, 可以分布在系统中的任意节点上, 通过网络进行通信。这样系统管理员可以根据需要将模块部署在节点之间, 提供最佳性能。图 2.3 说明根据各个模块对于存储资源的角色划分以后的资源操作。图中的所有模块均围绕着资源展开各自操作: 提供资源、管理资源、组织资源和使用资源等。资源可以是静态的, 比如物理磁盘空间; 也可以是动态的, 比如元数据服务器在某个时刻管理的活跃元数据。图中所有的通信都通过计算机网络或进程间通信完成, 使用建立在 TCP/IP 协议之上的远程过程调用 (Remote Procedure Call, RPC) 或者自定义的协议。

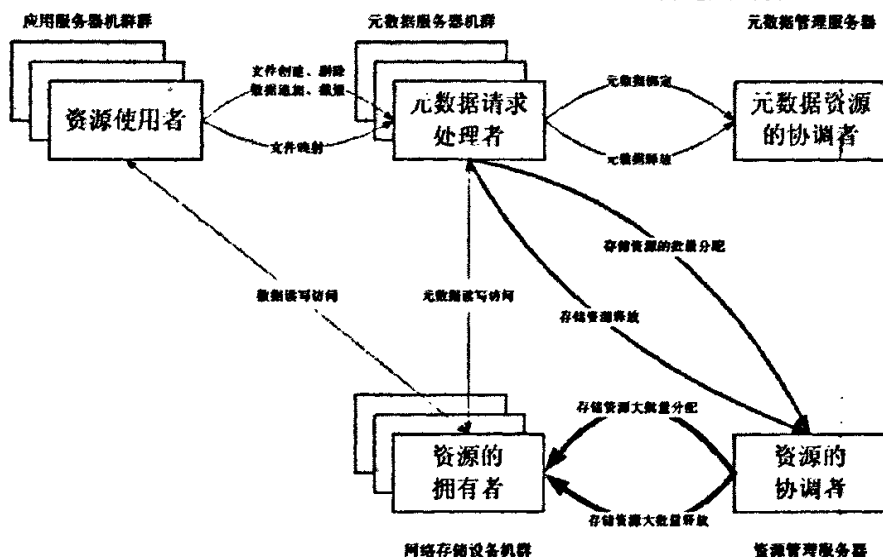


图 2.3 BWFS 中的资源操作

我们将整个 BWFS 划分成多个模块, 将它们分布在多个节点上, 采用层次分明的调用关系, 因此才将此结构叫“分布式分层资源管理模型”。下面一小节将介绍 DLRM 是如何实现虚拟存储以及如何进行资源组划分的, 其它部分将在后面的章节介绍。

图 2.4 显示了系统中各个模块在操作系统中的部署情况。

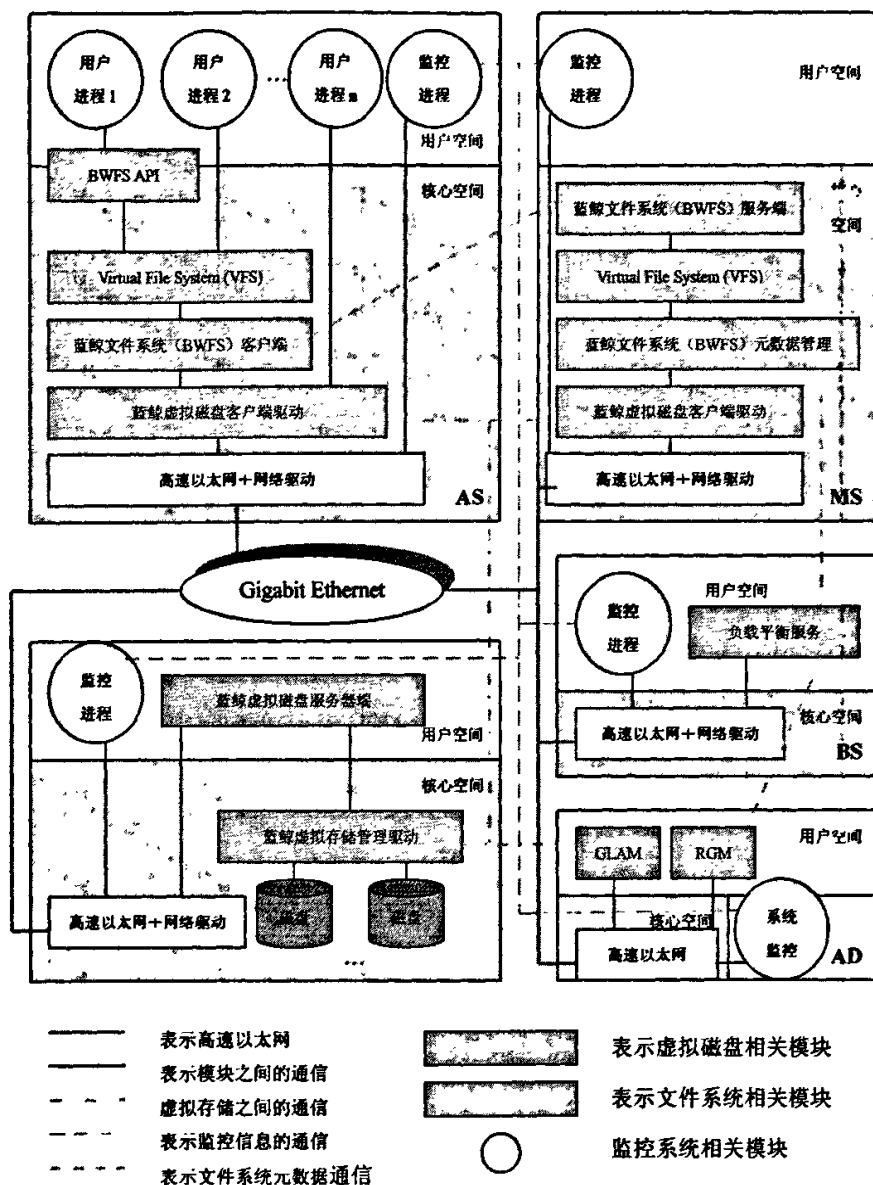


图 2.4 BWFS 的模块部署和通信

## 2.3 全局逻辑地址和资源组

为实现海量存储和支持动态添加存储设备，蓝鲸分布式文件系统需要管理多个存储设备上的多个物理磁盘。为了简化整个文件系统的设计与实现，也为了更好地兼容原有的集中式文件系统的管理模式，我们采用 64 位无符号整数将所有存储节点的所有存储空间统一编址，形成文件系统中使用的全局逻辑地址（Global Logical Address, GLA）。文件系统中使用的所有地址都使



用 GLA 表示。GLA 与存储节点以及物理磁盘之间的映射关系，由全局逻辑地址管理器（Global Logical Address Manager, GLAM）统一管理。GLAM 将这种映射关系以特定格式的文件保存在持久存储介质上，以备查找与修改。

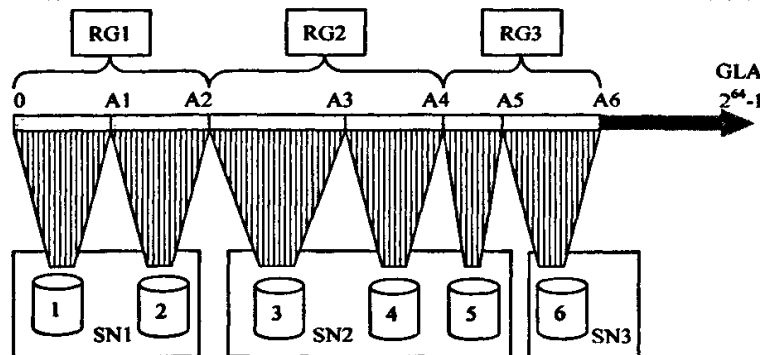


图 2.5 GLA 和 RG

GLAM 按照管理员指定的方式初始化系统中所有的存储设备，形成全局逻辑地址空间，也能够根据管理员的指令，随时将新添加到系统中的存储设备进行编址，融入到原有的地址空间中，以扩充系统的整体容量。GLAM 向上层提供全局逻辑地址到物理存储设备的映射关系查询，协助上层资源使用者正确存取数据。图 2.5 表示将三个存储节点 SN1、SN2、SN3 上的物理磁盘映射到统一逻辑地址空间后的情景，其中逻辑地址 A6 以后的空间没有映射，表示该部分暂时不可访问。如果系统新添加了存储设备，GLAM 可以将它映射到全局逻辑地址空间中现在还没有映射的地方，供文件系统访问。

GLAM 是一个独立的进程，所有对 GALM 的访问请求都串行执行，保证系统中地址管理的一致性。BWFS 中每一个需要访问存储设备的节点都安装一个经过改进的 NBD 驱动程序或者 iSCSI 驱动程序，以 GLA 为地址，通过块设备方式访问所有存储资源，形成共享磁盘（share-disk）的架构。虚拟磁盘的驱动程序在接收到读写请求以后，向 GLAM 查询此请求涉及的数据块的映射信息，然后再向相应的设备按照一定的协议发出请求。驱动程序会缓存已经查询过的映射信息，下次访问无需再次查询。

虽然 GLAM 将物理存储空间组织成了统一的逻辑地址空间，但是存储资源还是具有不同的属性，比如存取速度、磁盘可靠性等。一般的文件系统为了加快资源查找、分配/释放的速度，将整个存储空间划分成多个独立的区域，并管理多个区域的资源[30]。BWFS 采用类似的方法将整个逻辑地址空间划分成多个资源组（Resource Group, RG），每一个 RG 是一段具有连续 GLA 的存储资源，它们具有相同或者相似的物理属性。RG 的大小以及整个系统中 RG 的数量取决于整个系统容量的大小、相似属性资源的分布情况以及系统配置等。图 2.5 表示整个系统划分成三个资源组 RG1、RG2、RG3。

RG 中既可以保存文件系统的元数据（索引节点、目录数据块、间接数

据块等),也可以保存文件系统的数据(数据块)。在 BWFS 中,每一个 RG 都由一个资源组管理器(Resource Group Manager, RGM)管理该 RG 的资源使用情况。RGM 动态分配各种资源,而不是固定各种资源的占用比例,以适应不同的使用模式和有效地利用存储空间。RGM 和其它使用物理资源的模块一样都是通过 GLA 访问它所管理的资源,因此模块之间的接口简单清晰。各个 RGM 相互独立工作,并发处理各种资源管理请求,有利于系统扩展。各个 RGM 之间由位于管理服务器上全局资源管理器协调。RGM 利用动态位图和多级统计信息相结合的方法管理资源的分配情况,提高系统处理请求的效率。这部分工作的具体算法和实现将在第三章介绍。

资源组管理器是一个模块或者一个进程,可以运行在整个系统中的任意节点上面,向文件系统的其它部分提供存储空间管理服务。

## 2.4 小结

DLRM 模型是在仔细分析了多种分布式文件系统的结构,对比了它们的优缺点的之后,结合大规模海量存储的需求而设计的一个模型。DLRM 模型继承了已有分布式文件系统的一些优点,独创了一些适应海量存储的技术。该模型具有如下一些特点:(1)带外数据传输。BWFS 的所有文件数据直接在 AS 和 SN 之间交换,无需经过 MS 转发,提高系统的性能。(2)资源的批量申请/异步释放。上层以较大粒度向下层申请资源,异步释放空闲资源,减少各个层次之间的通信以及由此带来的延迟,避免出现资源碎片,提高系统性能。(3)并发资源管理。多个层次上的多个模块,并发管理不同的资源,提高资源管理的效率以及整个系统的可扩展性。(4)负载均衡。BWFS 有效地在多个 SN 之间、多个 MS 之间进行负载平衡。(5)动态扩展。模型利用虚拟存储以及资源组的划分,可以实现整个系统的动态扩展。(6)完全分布的模块。各个模块可以处在同一个节点上,也可以分别部署在不同的节点上,由多个节点分担负载,提高系统性能。

DLRM 模型较好地解决了分布式文件系统当前遇到的问题,使得蓝鲸分布式文件系统具有管理海量存储空间的潜力,进行动态扩展的可能,以及简化高效的系统架构。

本文后续章节针对 DLRM 模型中的细节有更加详细的阐述。

### 第三章 磁盘空间管理

通常意义下的文件系统[43]有两种含义：(1) 存储设备（包括硬盘、软盘、光盘、磁带等存储设备）上的一种数据结构，一般包括超级块、位图等系统数据，还有用户的目录、文件、链接等数据；(2) 操作系统中一个重要的系统级软件，用于管理上述数据结构，向用户提供基于“文件对象（file object）”的访问接口，避免用户直接管理存储空间。文件系统的基本功能是：

(a) 跟踪（tracking）记录存储设备上已经使用的空间和空闲空间，(b) 维护整个文件系统的目录树结构，(c) 记录每一个文件（包括目录、链接等特殊文件）的属性，包括其数据块位置等信息。其中跟踪磁盘空间的使用情况，确定哪些空间已经被使用，哪些空间还是空闲可用的是其它功能的基础，其实现的质量也直接影响整个文件系统的性能。BWFS 虽然管理多个存储节点组成的海量存储资源，但其文件系统的基本功能还是一样。

使用传统的本地文件系统（比如 FAT32 或者 ext2/3 等）管理大容量存储空间时，一方面遇到文件系统最大尺寸限制，另一方面由于其数据结构设计得不够精良使得系统效率较低。为了管理海量存储空间，蓝鲸分布式文件系统首先要突破文件系统容量限制。同时，蓝鲸分布式文件系统需要能够动态扩展，以便适应不断变化的应用。再者，蓝鲸分布式文件系统必须拥有较高的性能。通过将整个存储空间划分成多个资源组，蓝鲸分布式文件系统实现海量存储和动态扩展；通过动态分配数据块/索引节点、带统计信息的位图等技术，实现高性能的空间管理。

本章首先介绍相关文件中关于磁盘空间管理的设计与实现情况，然后描述蓝鲸分布式文件中关于磁盘空间管理的设计与实现。这里的磁盘空间管理是指上段中归纳的文件系统的第一个基本功能。

#### 3.1 相关研究

Linux 操作系统广泛使用 ext2/3[48]文件系统作为缺省的主文件系统。在 ext2/3 中，磁盘空间首先被划分成若干个大小相等的块（block），块的大小在创建文件系统时由管理员指定，或者由系统自行决定，一般为 1024 字节、2048 字节或者 4096 字节。块是文件系统进行空间分配的最小单位，文件系统中的所有文件（包括目录、链接等特殊文件）的大小都是块大小的整数倍。若干个连续块形成一个组（group），由组描述符（group descriptor）描述该组的情况，比如组内空间使用情况，组内相关指针等。每一个组的大小是确定的，整个文件系统由多个组首尾相连构成。组描述符总是位于组的头部，紧接着

是块位图 (block bitmap) 和索引节点位图 (inode bitmap), 之后就是索引节点表 (inode table) 和数据区域。块位图占用一个块的大小, 索引节点位图也是。如果块的大小是 4K 字节, 那么一个组能容纳的块的数目就是 32K ( $4K \times 8$ ) 个, 能拥有的索引节点的数目也是 32K ( $4K \times 8$ ) 个, 总共涵盖 128M 字节的空间。但是由于这些位图以及紧随其后的索引节点表本身占用一定的空间 (固定数量的空间), 真正可以存放用户数据的空间将比 128M 字节少一些。图 3.1 说明了 ext2/3 的物理磁盘布局。

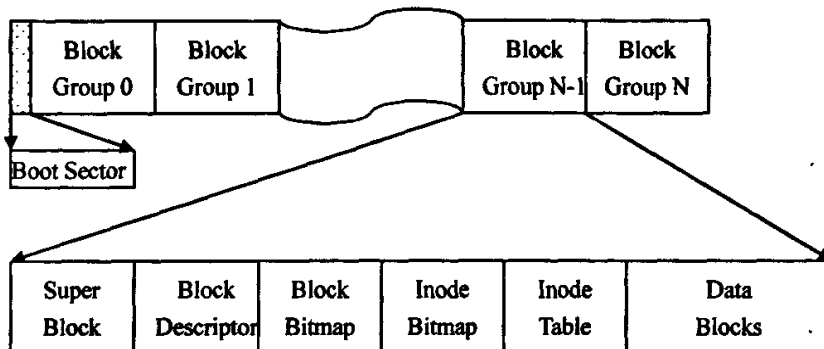


图 3.1 ext2/3 的物理磁盘布局

从上面的描述可以看出, ext2/3 固定索引节点与数据块两者之间的比例, 其使用的块和索引节点可以通过简单的计算得到其物理存储位置。但是同样由于这样的固定关系, 使得资源无法动态分配, 可能造成存储空间浪费的情况。比如在存放大量小文件的系统中, 可能出现索引节点不够使用, 即使还有数据块空间, 也无法创建新的文件; 或者在存储大文件的系统中, 无法利用索引节点表占用的空间。所有的超级块、组描述符、块位图、索引节点位图、索引节点表等数据结构必须在创建文件系统的时候就初始化, 因此在一个大容量的磁盘上创建该文件系统将消耗较长时间。

IBM 的 JFS[49][50][51] 是 AIX 操作系统中的主文件系统, 现在作为开放源码软件被移植到 Linux 操作系统中。JFS 在每一个分区上面创建一个聚合 (Aggregate), 每一个 Aggregate 又被划分成多个可独立装载的 (mountable) 文件集合 (fileset)。在目前 Linux 操作系统的 JFS 中, 一个 Aggregate 只能包含一个 fileset。Aggregate 被划分成多个分配组 (Allocation Group, AG), 以利于实现各种文件系统的分配策略, 比如利用文件数据相关性实现存储位置的连续性等。Aggregate 拥有两个索引节点表 (Inode Table)、两个索引节点位图 (Inode Map) (一个是主数据, 一个是备份数据) 和一个块位图 (Block Allocation Map) 等数据结构。每一个文件集合拥有一个文件集合的索引节点表 (Fileset Inode Table) 和文件集合的索引节点分配位图 (Fileset Inode Allocation Map)。JFS 中的所有索引节点表和索引节点分配位图都是动态、按需分配的: 创建之初只有最基本的表和位图, 随着文件的增加, 动态分配

这些数据结构；随着文件较少，可以释放这些数据结构。虽然动态分配必然增加索引节点的动态跟踪（Dynamic Tracking）开销（也就是从索引节点号到物理存储设备地址的映射开销），但是在大型文件系统中这个功能相当重要，它使得文件系统可以适用更多的场合，减少存储空间资源的浪费，更加快速地创建文件系统等。

很多其它的文件系统比如 XFS[30]，ReiserFS[52]等也都采用动态分配索引节点的方式。

### 3.2 BWFS 面临的挑战以及应对策略

蓝鲸分布式文件系统的设计目标是应用在大规模海量存储系统中，向机群环境提供分布式文件系统服务，因此面临着与许多本地文件系统不同的环境和应用需求，比它们具有更高的挑战。在存储资源的空间管理方面，这些挑战主要表现在如下方面：

- (1) 巨大存储容量。如今许多大型应用所需要的原始数据和产生的结果数据都非常庞大，需要巨大的存储空间才能满足需求，它们的数据量都在若干 Tera 字节，甚至若干 Peta 字节。如此庞大的数据量需要保存在多台存储服务器或者磁盘阵列上，它们协同工作才能满足应用的需求。传统的单机文件系统都不是针对大型应用设计的，无法管理如此众多的存储节点，难以管理如此庞大的数据量，其基于单机系统的设计基础难以胜任机群环境下的复杂应用。蓝鲸分布式文件系统需要管理众多节点上的海量存储空间，并且提供并发访问服务，才能取得较高性能。
- (2) 动态扩充能力与负载平衡能力。一般情况下，海量存储空间不是一次部署到位，而是随着应用需求的增长逐步添加的。同时，应用程序访问的数据也在不断变化，各个应用之间的负载未必完全相同。蓝鲸分布式文件系统需要能够随着存储设备的添加，将新添加的存储空间纳入到系统中来，实现业务的连续性。

为了应对以上挑战，我们在蓝鲸分布式文件系统中设计了多项新技术，在存储空间管理方面实现动态分配、负载平衡等，为整个分布式文件系统的功能和性能打下坚实基础。

首先蓝鲸分布式文件系统采用存储虚拟化技术，将多个存储设备上的存储空间统一编址形成全局逻辑地址空间。所有需要使用存储空间的上层模块都使用该全局逻辑地址进行数据访问，简化了上层模块对于多个存储节点的访问路由问题。同时，统一使用全局逻辑地址访问存储空间可以使得上层文件系统的模块可以兼容更多的设备，比如可以是 IP SAN 的存储设备、SAN

存储设备等,还可以使得文件系统的各个模块相对独立,便于各个部分的设计实现与分别优化。

经过虚拟化的存储空间来自多台存储设备,并且可能具有不同的属性。为了提高存储空间的管理效率,并且体现不同的属性,我们将这些存储空间划分成多个独立的资源组,每一个资源组各自独立地管理存储空间的使用情况,对外提供各种服务,实现并发管理,提高扩展性和性能。我们将存储空间的管理与文件系统元数据(文件属性、目录树等)分开,可以将文件系统的功能更多地分担到系统中的多个节点,充分利用各个节点的计算能力。

传统的 Linux 文件系统 ext2/3 没有实现动态数据块和索引节点分配,难以胜任海量存储系统的管理。XFS[30]、ReiserFS[52]虽然实现了动态分配索引节点,却没有提供在线迁移的功能。BWFS 设计了全新的空间管理方式,动态分配数据块和索引节点,采用带统计信息的位图管理空间分配情况,提高系统的性能,支持动态添加和在线迁移等功能。这样, BWFS 可以根据需要更好地管理存储资源,实现各种特定需求的空间资源分配,也为系统的后续优化提供了可能。

### 3.3 资源组

一个资源组 (Resource Group) 是一段连续的全局逻辑地址所表示的具有相同或者相似属性的物理存储空间。每一个资源组都有一个资源组管理器 (Resource Group Manager) 负责对其上的可用存储空间和已用存储空间进行跟踪管理。RGM 的主要任务是进行索引节点 (inode) 和数据块的分配与回收,以及索引节点和数据块的定位与查找。

将整个存储空间经过虚拟化映射到全局逻辑地址空间,然后划分成多个资源组,主要是基于以下考虑:(1) 来自不同存储节点的物理存储资源虽然形成了统一的全局逻辑地址,但它们还是可能具有不同的物理属性,表现出不同的访问特性,比如数据传输能力、访问延迟、磁盘可用性等。文件系统需要区分对待这些不同属性,以便能够更好地安排组织文件系统的的功能。(2) 将整个逻辑地址空间划分成多个独立的资源组,每个资源组管理各自存储空间的使用情况,并发工作,提高性能。(3) 实现存储空间动态添加。当系统添加了新的存储节点以后,系统管理员将新的存储空间映射到未分配的全局地址空间,然后创建一个新的资源组管理这些存储空间,实现动态添加。

在 BWFS 中,每一个资源组都被赋予一个唯一的资源组号 (RG Number)。资源组的尺寸大小可以根据实际情况确定,但是为了管理方便,同时考虑操作系统的限制和单个存储节点的容量,一般设置一个上限。在 Linux 2.4 系列的内核中,操作系统能够识别的块设备的最大容量是 2T ( $2^{41}=2^{32}\times 2^9$ ) 字节;

在 Linux 2.6 系列的内核中, 操作系统能够识别的块设备的最大容量是  $2^{64}$  字节。于是我们约定在使用 Linux 2.4 系列内核的系统中, 每一个资源组最大管理  $2T$  ( $2^{41}$ ) 字节的存储空间 (受限于块设备容量的最大值), 并且 RG 使用 GLA 表示的起始地址一定是  $2T$  的整数倍; 在 Linux 2.6 系列内核下, 资源组的最大容量由管理员确定, 但是也不能超过  $2^{64}$  字节。

由于 BWFS 的元数据服务器等使用的存储空间都是以全局逻辑地址描述, 所有文件系统内部的指针都是以此表示。因此为了能够在各个资源组之间进行数据的负载平衡, 我们尽量将多个存储节点提供的存储空间映射到独立的资源组中, 这个需要 GLAM 将存储空间映射到以资源组尺寸大小为边界的全局逻辑地址上。

下面小节分别介绍资源组管理器如何管理存储空间和索引节点。

### 3.3.1 数据块管理

数据块管理的主要任务是分配与回收该资源组中的数据块。BWFS 的数据块可以用来存放索引节点、文件系统使用的间接指针、目录项和文件的数据等。Linux 中的 ext2/3 文件系统使用单纯的、一维的位图跟踪数据块的分配情况, JFS 使用四层树状结构的位图跟踪数据块的分配情况[51]。

考虑到 BWFS 的每一个资源组的容量一般在  $2T$  字节左右, 我们采用三级带统计信息的位图结构进行数据块空间管理。利用各级统计信息, BWFS 可以加快空间管理的速度, 提高性能。图 3.2 演示一个资源组的磁盘布局情况: 它从全局逻辑地址  $x$  开始, 共有  $n$  ( $n \leq 2T$ ) 个字节长, 图中每一个方框表示 4096 个字节。资源组的起始部分是一个超级块, 它存放有关此 RG 的属性信息, 比如 RG 的长度、RG 的读写性能、空闲数据块数目等。接着超级块的是 256 个统计块 (Summary Block) 和  $m$  个位图块 (Bitmap Block); 在此之后都是被管理的数据块。在 BWFS 中, 每一个数据块的大小都是 4096 字节。

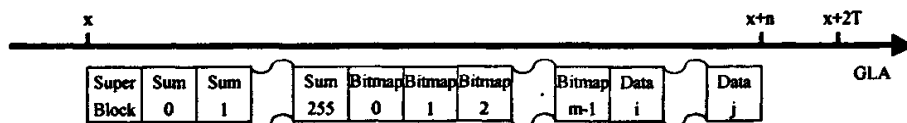


图 3.2 资源组的磁盘布局

每一个位图块实际管理 8192 个数据块的分配情况, 共计 32M 字节的存储空间。这些位图块管理的存储空间首尾相连, 一一对应: 第一个位图块 (也就是 Bitmap 0) 管理从地址  $x$  开始的 32M 字节的存储空间, 第二个位图块 (也就是 Bitmap 1) 管理从地址  $x+32M$  开始的 32M 字节的存储空间, 依此类推。位图块中有它所管理的空间中空闲的数据块数目、第一个空闲的数据块的相

对位置、最大连续空闲块的相对位置等有助于加速分配的信息。为了在分布式环境中实现分布式事务处理,提高系统的可用性和一致性,我们又将这些信息复制,存储在同一位图块的前后两半部分,参见图 3.3。采用两阶段提交机制(Two Phase Commit) [53][54],在元数据服务器的协同下,资源组管理器可以参与完成分布式事务处理。出于性能的考虑,也可以选择不采用分布式事务处理机制。如果某个资源组的大小不是 32M 的整数倍,那么最后一个位图块中的最后若干位在资源组初始化时就被置为已经使用,避免空间分配错误或者形成空间浪费。位图块中保存着整个 RG 中数据块的分配情况,统计块中的统计信息和超级块中的统计信息都是基于它们的内容计算出来的结构。

```

struct blockmap {
    struct {
        __u32    number_of_blocks;
        __u32    number_of_free_blocks;
        __u32    first_free_block;
        __u32    max_contiguous_free_blocks;
        .....
        __u32    bitmap[256];
    } working; //总共 2048 字节长
    struct {
        __u32    number_of_blocks;
        __u32    number_of_free_blocks;
        __u32    first_free_block;
        __u32    max_contiguous_free_blocks;
        .....
        __u32    bitmap[256];
    } permanent; //总共 2048 字节长
}; //总共 4096 字节长

```

图 3.3 位图块的数据结构

```

struct summary_block {
    struct {
        __u32    number_of_free_blocks;
        __u32    max_contiguous_free_blocks;
        .....
    } sum[256];
}; //总共 4096 字节长

```

图 3.4 统计块的数据结构

每一个统计块对 256 个相应的位图块进行资源分配情况的统计: 第一个



统计块 (Sum 0) 对前 256 个位图块 (Bitmap 0—Bitmap 255) 进行统计, 第二个统计块 (Sum 1) 对之后的 256 个位图块 (Bitmap 256—Bitmap 511) 进行统计, 依此类推。统计块包含一个 256 个元素的数组, 对应 256 个被统计的位图块。数组的每个元素记录对应位图块表示的空间中空闲块的数据、最大连续空闲块的数目等信息, 见图 3.4。利用这些统计信息, 资源组管理器在分配数据块资源时可以快速进行扫描定位, 选择合适的位图块进行操作。超级块中有一个 256 个元素的数组, 用来对 256 个统计块所包含的信息进行统计: 每个数组元素包含对应统计块的空闲数据块总数、最大连续空闲块数目, 总长度是 2048 字节。图 3.5 演示了它们之间的树状统计关系。

资源组管理器在数据块管理方面提供分配和释放服务。资源组管理器是一个独立的服务, 可以运行在机群内的任意节点上, 在网络上监听服务请求。元数据服务器在需要存储空间时, 根据存储空间的用途和当前系统的运行状况等因素, 从系统中所有的资源组中挑选一个, 向对应的资源组管理器申请数据块分配服务。申请数据块分配的请求携带一定的参数, 比如数据块的数目、数据块的类型 (必须是连续的或者可以是分散的) 等。资源组管理器首先检查超级块中的统计信息, 确定拥有符合条件的统计块的位置。之后检查相应的统计块, 确定拥有符合条件的位图块的位置。然后根据位图块的信息, 扫描位图块中的位图, 修改位图, 实现分配。最后将更新过的位图块同步到磁盘, 更新内存中的统计块信息和超级块信息。为了提高系统的性能, 一般将超级块和统计块缓存在内存中, 其内容即使被修改了也不会立刻同步到磁盘, 而每次修改位图块都需要同步写回磁盘。释放数据块的时候比较简单, 根据请求的内容, 修改位图块信息和位图, 同步到磁盘, 然后更新统计块信息和超级块信息。

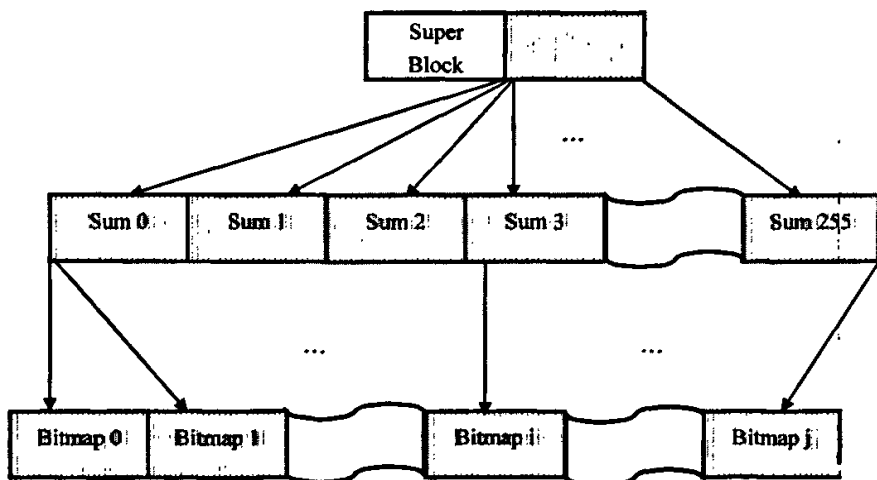


图 3.5 位图块、统计块和超级块的树状统计关系

如果需要利用分布式事务处理的机制提高系统的可用性和在系统某些节

点出现故障的情况下保证一致性,我们可以利用位图块的工作区域(working)和持久区域(permanent)实现两阶段提交:先修改工作区域,然后做一个标记,表明进入可提交阶段,返回可提交的应答;然后根据提交请求修改持久区域,完成提交。

在扫描统计信息,决定符合条件的数据块位置的时候,可能同时出现若干个满足要求的位置。我们可以采用“最先匹配法”、“最大匹配法”、“最小匹配法”或者它们的组合来决定即将分配的数据块的位置。在 BWFS 中,元数据服务器向资源组管理器申请资源都是使用批量申请/释放的方式,而不是单个数据块的粒度为申请单位。批量/释放申请多个数据块可以明显减少两个模块之间的通信,也可以尽量避免出现资源碎片,提高整个系统的性能。这在后面的章节还有更多描述。

利用三级带统计信息的位图结构,每一次空间资源分配最多只需要三次内存扫描:第一次在超级块中最多扫描 2K 字节,第二次在统计块中最多扫描 4K 字节,第三次在位图块中最多扫描 2K 字节。通过这种方式可以避免大规模扫描所有的位图块,特别是在资源空闲率较低的情况下,大大提高了系统的效率。

### 3.3.2 索引节点管理

在 UNIX/Linux 操作系统中,任何文件系统的每一个文件(包括普通文件、目录文件、链接文件、管道等)都有一个内存结构的索引节点(inode)与之对应,这是虚拟文件系统(Virtual File System, VFS) [55]可以处理多种不同物理文件系统的关键。于是,为了更加方便地实现 UNIX/Linux 操作系统下的文件系统,ext2/3、JFS、XFS 等文件系统都有一个磁盘索引节点的数据结构与每一个文件相对应。文件的属性信息和一些内部指针都包含在索引节点中,比如文件的大小、访问属性、属主、修改时间、数据块指针等。物理磁盘上的索引节点的大小根据不同文件系统的需要而不同,比如 ext2/3 的索引节点是 128 字节长;JFS、XFS 的索引节点是 512 字节长,这主要是由于 JFS 和 XFS 都是 64 位的文件系统,每一个索引节点需要记录更多的信息的缘故。

本章中所提到的索引节点如果没有特别说明都是指蓝鲸分布式文件系统的物理索引节点。为了提供 64 位的扩展能力,我们在这一版本的蓝鲸分布式文件系统中使用 512 字节长的索引节点。资源组管理器的索引节点管理主要是分配和释放索引节点占用的存储空间,动态跟踪索引节点号(inode number)和其存储位置之间的映射关系。

BWFS 管理众多的存储节点,海量存储空间。如果分配固定比例的存储

空间给索引节点，在以大文件占多数的系统中，会浪费大量的分配给索引节点的存储空间；在系统含有大量小文件的情况下，可能会导致索引节点的存储空间不够用。在图 3.2 中没有出现管理索引节点的位图以及索引节点表的存储位置，并不是该图有纰漏之处，而是我们采用动态分配索引节点位图和索引节点表的方式。所有的索引节点位图和索引节点表都是根据需要在数据区动态分配，动态跟踪的。下面将介绍资源组管理器是如何进行索引节点管理的。

BWFS 的索引节点不是集中在某一个资源组中，而是分布在各个资源组中，可以由管理员指定，也可以由系统根据当前的负载情况决定，尽量平衡各个资源组的负载。既然由多个资源组分布式管理所有的索引节点，就需要在各个资源组之间同步一些关键操作，比如索引节点号和资源组之间的映射管理等。因此我们引入全局资源组管理器（Global Resource Group Manager, GRGM）进行各个资源组管理器之间的同步。BWFS 在整个文件系统中管理  $4G(2^{32})$  数量的索引节点，这些节点如何在各个资源组之间分布由全局资源组管理器协调。全局资源组管理器向资源组管理器提供索引节点的分配与回收服务，每次分配与回收的粒度是 8192 个索引节点，这也是资源组管理器中一个索引节点位图管理的数量和一个索引节点表的包含的索引节点的数量。全局资源组管理器将这些索引节点和资源组之间的映射关系记录在文件中，向整个系统提供查询服务。全局资源组管理器同步操作所有的分配与回收、查询服务，保证系统的全局一致性。

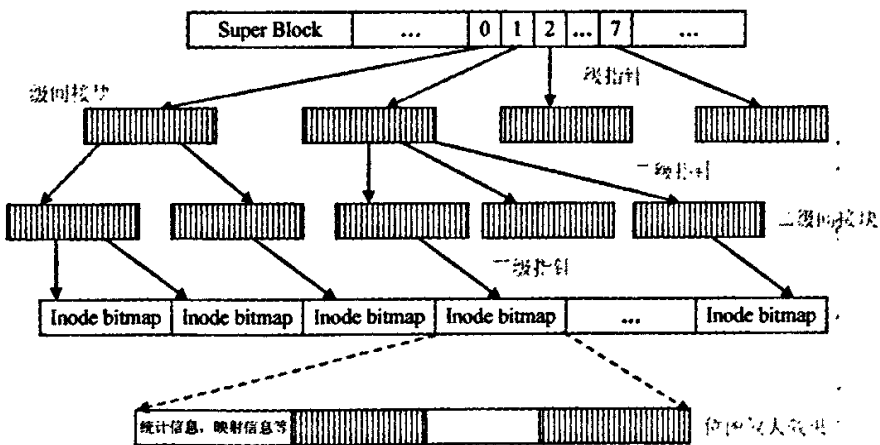


图 3.6 索引节点的三级指针位图模型

在每一个资源组中，为了既能实现快速动态分配索引节点，又能实现快速跟踪索引节点的存储位置，我们设计了三级指针位图模型，如图 3.6 所示。图中最上方是超级块，其中含有 8 个一级指针，分别指向 8 个一级间接块；每个一级间接块又含有 256 个二级指针，指向 256 个二级间接块；每个二级间接块含有 256 个三级指针，指向 256 个真正的索引节点位图块；一个索引

空间给索引节点，在以大文件占多数的系统中，会浪费大量的分配给索引节点的存储空间；在系统含有大量小文件的情况下，可能会导致索引节点的存储空间不够用。在图 3.2 中没有出现管理索引节点的位图以及索引节点表的存储位置，并不是该图有纰漏之处，而是我们采用动态分配索引节点位图和索引节点表的方式。所有的索引节点位图和索引节点表都是根据需要在数据区动态分配，动态跟踪的。下面将介绍资源组管理器是如何进行索引节点管理的。

BWFS 的索引节点不是集中在某一个资源组中，而是分布在各个资源组中，可以由管理员指定，也可以由系统根据当前的负载情况决定，尽量平衡各个资源组的负载。既然由多个资源组分布式管理所有的索引节点，就需要在各个资源组之间同步一些关键操作，比如索引节点号和资源组之间的映射管理等。因此我们引入全局资源组管理器（Global Resource Group Manager, GRGM）进行各个资源组管理器之间的同步。BWFS 在整个文件系统中管理  $4G(2^{32})$  数量的索引节点，这些节点如何在各个资源组之间分布由全局资源组管理器协调。全局资源组管理器向资源组管理器提供索引节点的分配与回收服务，每次分配与回收的粒度是 8192 个索引节点，这也是资源组管理器中一个索引节点位图管理的数量和一个索引节点表的包含的索引节点的数量。全局资源组管理器将这些索引节点和资源组之间的映射关系记录在文件中，向整个系统提供查询服务。全局资源组管理器同步操作所有的分配与回收、查询服务，保证系统的全局一致性。

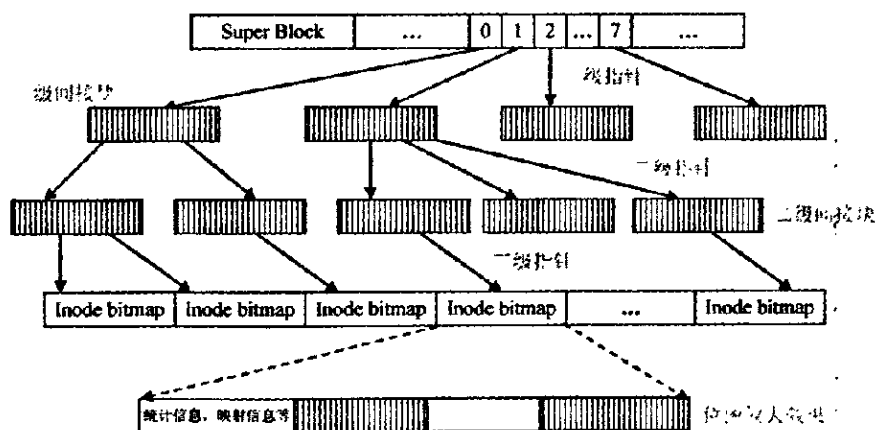


图 3.6 索引节点的三级指针位图模型

在每一个资源组中，为了既能实现快速动态分配索引节点，又能实现快速跟踪索引节点的存储位置，我们设计了三级指针位图模型，如图 3.6 所示。图中最上方是超级块，其中含有 8 个一级指针，分别指向 8 个一级间接块；每个一级间接块又含有 256 个二级指针，指向 256 个二级间接块；每个二级间接块含有 256 个三级指针，指向 256 个真正的索引节点位图块；一个索引

节点位图块管理 8192 个索引节点, 存储有这些索引节点的存储位置、统计信息、分配情况等数据, 如图中最下面的放大效果。按照这样的层次关系; 每一个资源组都能管理  $4G(2^{32})$  个索引节点。在实际使用过程中, BWFS 将全体正在使用的索引节点分配到多个资源组, 每一个资源组存储不重叠的一部分。这样在每一个资源组的这个树状结构中将出现很多空指针, 表示对应的索引节点没有被分配, 或者没有存储在此资源组中。将这一棵树的所有叶子节点 (索引节点位图块) 顺利线性连接起来, 就是一个稀疏文件。该稀疏文件保存着此资源组中所有索引节点位图块, 每个位图块占用 4K 字节的一个数据块, 管理 8192 个索引节点, 顺序映射。每一个索引节点位图块管理的 8192 个索引节点形成一个连续的索引节点表, 从数据块中分配, 占用 4M 字节存储空间。索引节点位图块记录它所管理的索引节点表的位置, 记录分配的信息, 比如空闲索引节点的数量、第一个空闲索引节点的位置等, 加速分配与回收。二级间接块实际上存储一个 256 个元素的数组, 每个元素记录索引节点位图块的存储位置, 以及该位图块的一些统计信息, 比如空闲索引节点数量等, 以利于分配时加速查找。同理, 一级间接块中的 256 元素的数组和超级块中的 8 个元素的数组都记录下一级间接块的存储位置和统计信息。这样就形成了一个树状结构。为了实现分布式事务处理, BWFS 同样将索引节点位图块分为工作区域和持久区域两个部分, 以便利用两阶段提交机制。索引节点位图块的数据结构如图 3.7 所示, 间接块的数据结构如图 3.8 所示。

```

struct inode_bitmap {
    struct {
        __u32    number_of_inodes;
        __u32    number_of_free_inodes;
        __u32    first_free_inode;
        __u64    start_block_number_of_this_inode_table;
        .....
        __u32    bitmap[256];
    } working; //总共 2048 字节长
    struct {
        __u32    number_of_inodes;
        __u32    number_of_free_inodes;
        __u32    first_free_inode;
        __u64    start_block_number_of_this_inode_table;
        .....
        __u32    bitmap[256];
    } permanent; //总共 2048 字节长
}; //总共 4096 字节长

```

图 3.7 索引节点位图块的数据结构

本资源组的相对值，也就是从零开始编号的地址。这样，资源组中的所有地址都与资源组映射到的全局逻辑地址没有必然联系，两者相互独立。利用此特性，我们可以将资源组内的数据进行整体迁移，然后修改全局资源组管理器记录的映射关系，系统就可以无需修改其它数据结构继续工作。系统还可以将部分索引节点位图块、索引节点表从一个资源组迁移到其它资源组，重新为其建立映射关系，而无需修改使用索引节点的文件系统内部数据结构，达到在线迁移的目的。利用这样的在线迁移，系统可以平衡各个资源组的负载情况，尽量满足系统的需求，提高系统的可靠性和性能。

```

struct inode_indirect_block
{
    __u32    magic1;
    __u32    magic2;
    __u64    totao_size;    // in bytes
    __u32    blocksize;
    __u64    number_of_blocks;
    __u64    number_of_free_blocks;
    __u32    number_of_sumary_blocks;
    __u32    number_of_block_bitmap;
    __u64    data_start;
    __u32    rgnno;
    __u32    inode_size;
    __u32    inode_table_size;
    .....
    struct
    {
        __u64    blockno;    //指向一级间接块
        __u32    number_free_inodes;
        .....
    } bitmap[8];
    struct {
        __u32    number_of_free_blocks;
        __u32    max_contiguous_free_blocks;
        .....
    } sum[256];
}; //总共 4096 字节长

```

图 3.9 超级块的数据结构

### 3.5 小结

蓝鲸分布式文件系统管理着众多存储节点上的海量存储资源，本章介绍了它的物理空间管理系统。BWFS 采用虚拟存储技术将全部物理存储资源映

射到全局逻辑地址空间，然后将其划分成多个资源组，每个资源组有一个资源组管理器，支持在线添加存储资源；资源组管理器独立、并发地管理各自的存储空间，实现动态分配索引节点，采用三级带索引的位图实现高效快速的空间管理；BWFS 还实现了资源组之间的负载平衡和在线迁移等高级特性，有效支持海量存储系统的要求。

## 第四章 元数据管理

文件系统的元数据是指文件系统中用来描述文件属性、目录结构、空间使用情况等信息的数据。这些元数据对于文件系统来说至关重要，因为它们直接关系着整个文件系统的正确性、一致性和可用性等。文件系统元数据结构的设计与实现的优劣直接影响着文件系统的整体性能，是系统评价和优化的一个重要部分。

在蓝鲸分布式文件系统中，存储空间使用情况的管理由资源组管理器负责。元数据服务器主要提供目录树组织、文件属性管理、为文件申请/释放存储空间等。元数据服务器接收来自蓝鲸分布式文件系统客户端的远程过程调用，比如创建/删除文件和目录、文件查询、分配数据块、改变文件属性等，组织文件系统的数据结构。元数据服务器代表客户端向资源组管理器批量申请/释放存储空间，缓存新近申请/释放的存储空间等。元数据服务器采用的批量资源申请/释放策略、资源异步释放策略、数据分片存储策略、全动态元数据绑定等使得蓝鲸分布式文件系统具有较好的性能和较好的可扩展性。

本章主要介绍蓝鲸分布式文件系统中与资源相关的系统设计与优化。

### 4.1 背景

蓝鲸分布式文件系统管理着大量存储资源，向上千个客户端提供文件系统访问服务，其性能和可扩展性至关重要。蓝鲸分布式文件系统必须随着应用软件需求的提高管理更多的存储空间，提供更好的性能。

蓝鲸分布式文件系统所应用的环境与本地单机文件系统、NFS 所处的环境存在许多差异，这些差异主要是由所处的环境和服务的应用决定的。(1) 海量存储空间。蓝鲸分布式文件系统需要管理多个存储节点上的庞大存储资源。这些存储资源在编址形成统一的全局逻辑地址空间以后又被划分成多个资源组。如何充分利用各个存储节点并发传输数据，并尽量平衡各个节点的负载，是蓝鲸分布式文件系统面临的一个问题。多个资源组之间存在属性差异，区分这种差异，将不同的数据存储到合适的空间，是提高蓝鲸分布式文件系统性能与可靠性的一个途径。(2) 性能和可扩展性。在传统的分布式文件系统 NFS、CIFS 等中，单个文件服务器向多个客户端提供服务，其性能和扩展性是整个系统的瓶颈。这主要是由于其单个服务器难以承受大规模的数据访问。DCFS[31][32]、Storage Tank[24]等系统引入了多个元数据服务器写作提供服务的结构，但是它们都将整个文件系统的元数据目录树静态地划分到各个元数据服务器，不能在系统运行时时刻动态映射。蓝鲸分布式文件系统



意识到应用是千变万化的,应用对于文件系统的访问模式也是千变万化的。元数据在各个服务器之间静态映射的实现,难以根据系统当前状态进行负载平衡,容易出现个别服务器过于繁忙,而有些服务器处于相对很空闲的状态。因此蓝鲸分布式文件系统决定将活跃元数据(Active Metadata)全动态地映射到所有的元数据服务器中间,以便可以充分利用各个元数据服务器的处理能力。(3) 分布式环境。蓝鲸分布式文件系统的各个部分可以分布在整个系统的不同节点上,它们之间依靠 TCP/IP 网络进行通信。虽然计算机网络的数据传输性能有了飞跃式提高,传输延迟也进一步缩小,但是过多的计算机网络通信肯定会造成性能下降。我们需要尽量减少各个模块之间通信,提高每一次处理的速度,才能取得更好的性能。于是在蓝鲸分布式文件系统中,我们采用批量申请资源的方式分配物理存储空间资源,加大每一个文件预分配的块数,减少模块之间的通信。同时,在存储空间资源被释放需要回收时,我们采用暂时缓存的方式,实现异步释放,以期在不久的将来再次使用这些资源。通过这些方式,明显减少模块之间的通信,也减少了由于通信带来的延迟。

下面我将阐述蓝鲸分布式文件系统在处理与存储资源相关的元数据处理方面的设计、实现与优化。

## 4.2 元数据服务器的总体架构

元数据服务器是 BWFS 用来进行元数据处理的节点,它响应 BWFS 客户端的文件系统访问请求,组织/修改文件系统的元数据,并返回结果。元数据服务器采用与 NFS (Network File System) 相似的结构:客户端通过远程过程调用(Remote Procedure Call) [56]申请文件系统相关的服务;元数据服务器上的服务程序(ENFSd)接收到服务请求,进行数据解包和验证等工作,通过 VFS 机制提交给物理文件系统;BWFS 的物理文件系统是经过改进的 ext3 文件系统,我们称之为 EXFS。其层次结构如图 4.1 所示。

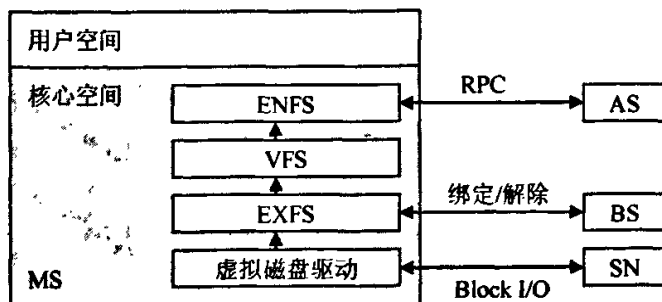


图 4.1 元数据的总体架构

ENFS 模块接收 AS 上相应模块的远程文件访问请求,遵循 NFS 的协议标准。ENFS 除了提供 NFS 的诸如 create、lookup、getattr 等调用以外,增加

一些调用（比如 `getbmap`，用于将文件内部块号映射为全局块号）和扩充了它们的返回值，以支持 BWFS 的“带外”数据传输模式和多元数据服务器结构等特性。EXFS 从 `ext3` 继承了部分功能，添加了适合分布式环境的功能特性，比如资源的批量申请/释放、有策略的资源分配、分布式日志等，使之能够获得更好的性能和可扩展性。

### 4.3 资源的批量申请/异步释放

文件系统的功能就是帮助用户管理磁盘存储空间的使用情况。物理存储空间是文件系统的唯一资源。在通常的本地文件系统中，空间管理和文件系统内部数据结构组织都是在同一节点上完成，它们之间的关系相对简单。BWFS 将空间管理交由资源组管理器负责，由多个资源组管理器分布式并行管理。元数据服务器上的 EXFS 主要负责文件系统的内部数据结构的维护，比如文件的属性信息、文件的目录树、文件的数据块指针等。这些数据结构都需要相应的存储空间保存，因此元数据服务器一方面自身需要存储空间保存文件属性、目录结构等元数据，另一方面代表应用服务器申请和释放文件数据的空间。

在本地文件系统的实现中，如果因为某种需要，系统将为其分配存储空间，一般设计成通过函数调用实现。但是在 BWFS 中，我们将空间分配的任务交由资源组管理器完成，主要是为了分担各个功能的负载，并发管理各个资源组的存储空间。这样做的结果是空间分配的调用需要在两个不同的模块之间进行，这两个模块还可能分布在不同的节点上，通过网络通信完成。为了减少功能调用的次数以及由此带来的访问延迟，BWFS 设计了批量申请和释放资源的模型。

文件系统分配存储空间的粒度是块（block），一个块的大小可以是 512 字节、1024 字节、2048 字节、4096 字节或者更大。目前文件系统常用的块大小是 4096 字节，BWFS 也采用这个粒度。元数据服务器向资源组管理器申请资源时，通常情况下一申请多个块：比如 8192 个块，刚好是一个位图块管理的数据块数量。资源组管理器可以根据当前空闲资源分布情况返回 8192 或者少于 8192 个数据块给元数据服务器，元数据服务器使用特定的数据结构记录这些存储资源，以供分配使用。通常情况下元数据服务器会从各个资源组都申请一些存储空间，统一缓存，形成一个临时的“存储池”——一个临时可用资源的集合。

在存储空间因为文件删除、截短等因素被释放以后，同样是首先被归入临时存储池管理，而不是直接向资源组管理器释放。这种方式既能加快每一个资源释放操作的执行，又能更好地缓存这些数据块。

在文件系统需要存储空间时,首先检查存储池中是否存在合适的资源可供使用。如存在,则直接从存储池中分配;否则,元数据服务器会向某个选定的资源组申请存储资源。在存储资源被释放时,它们都是首先被放入存储池,以便在不久的将来被再次使用。这样做的好处就是能够充分利用系统缓存(System Cache)的作用,提高系统的性能。

采用批量申请方式以后,如果元数据服务器出现失效,可能造成存储空间资源丢失,因为缓存在元数据服务器上的存储空间被资源组管理器标识为已经使用,但实际上可能已经被释放。这是在性能与空间浪费之间的权衡,可能丢失的空间相对整个存储空间来说是很小一部分,相对系统整体性能,它的重要性排在其次。

蓝鲸分布式文件系统为临时存储池设置了一些条件,在满足这些条件情况下,元数据服务器就向资源组管理器释放存储空间:时间超时、空间达标或者需要显式同步的时候。BWFS 在系统中有一个专门的内核进程——异步释放资源的进程,负责释放存储池中的资源。它平时处于睡眠等待状态,有三种方式可以使其进入工作状态,扫描存储池,释放存储空间:(1)每隔一定时间被系统唤醒一次,通常情况是 30 秒钟。(2)存储空间的容量超过一定的限度,该限度可以由系统管理员设定。(3)需要显式同步的时候,比如 sync 被调用的时候,或者是文件系统卸载的时候。在被唤醒之后,异步释放资源的进程扫描存储池,将多余的存储空间释放给资源组管理器。文件系统的其它工作可以继续展开,互相独立地工作(除非需要访问与存储池相关的关键数据结构,需要被锁定的除外)。

#### 4.4 索引节点的查找

每一个索引节点在被读取或者被写入磁盘的时候,都需要确定索引节点的存储位置。系统根据索引节点号唯一区分某一个文件系统中的索引节点,因此文件系统的功能就是必须能够根据索引节点号查找到其存储的物理位置, BWFS 的这部分功能由全局资源组管理器与资源组管理器共同完成。元数据服务器首先通过全局资源组管理器查得某个索引节点由哪个资源组管理器负责存储与管理,然后通过该资源组管理器查得此索引节点的实际存储位置。元数据服务器在获得了索引节点号和物理存储位置之间的关系以后,就可以通过块设备接口存取索引节点。利用与本地文件系统基本相同的访问途径,可以充分利用操作系统提供的索引节点缓存管理,简化文件系统的设计;同时还能兼容更多的设备和更多的方式访问,提高系统的适应能力。

#### 4.5 空间资源的分配策略

经过虚拟化的分布式存储资源形成了全局逻辑地址空间, 由于这些存储资源具有不同的物理属性, BWFS 将它们划分成多个资源组。多个独立的资源组管理器并行管理这些存储资源的使用情况, 提供索引节点与全局逻辑地址之间的映射关系查询等。

资源组的属性主要体现在资源组所管辖的存储资源的物理属性、资源组当前的空间使用率、资源组当前的负载等。存储资源的物理属性包括它们的大小尺寸、数据传输速率、数据传输延迟、数据可用性等。蓝鲸分布式文件系统根据不同用途可能需要不同属性的存储资源: 元数据尽量保存在高速率、延迟小、高可用的存储空间中; 用户数据可以放在吞吐率比较大的存储空间中。同时用户的数据跟随着应用的不同, 可能需要不同属性的存储空间。

蓝鲸分布式文件系统通过运行在每一个节点上的监控代理获得这些节点的负载情况, 这些负载信息将有助于对整个系统进行负载平衡和选择资源分配策略。元数据服务器获得这些信息以后, 结合存储空间使用的目的, 有选择地在各个资源组之间进行空间分配。目前蓝鲸分布式文件系统首先将用于存放元数据的空间和用于存放数据的空间加以区分, 尽量将元数据存放在可用性更好、性能更好的资源组中。

#### 4.6 分片存储

蓝鲸分布式文件系统管理多个存储节点上的海量存储空间。为了能够取得最理想的数据吞吐率, BWFS 需要充分利用每一个存储节点的传输带宽, 尽量并发访问每一个存储节点。在各个存储节点之间进行分片 (striping; 或称之为条带化) 存储是实现并发访问的一种最直接、最有效的方式。分片存储就是将一个文件或者整个文件系统的数​​据分成若干份, 并将它们分别有序地存储在各个存储单元之中。

蓝鲸分布式文件系统采用类似 RAID 0[33][34]的方式将文件的数据分散到各个资源组之间。虚拟存储技术已经对元数据服务器屏蔽了关于存储节点的有关细节, 元数据服务器看到的是被划分成多个资源组的统一的存储空间。为了尽量利用各个存储节点的带宽, 在进行资源组划分时, 尽量将一个存储节点划分成一个资源组。在某一个时刻, 所有可用的资源组形成资源组列表, 元数据服务器将文件数据按照一定大小分片存储在各个资源组中。但是与 RAID 0 不同的是, 蓝鲸分布式文件系统不要求所有的资源组具有相同的尺寸大小, 因此可能不会恰好平均利用每一个资源组的存储空间: BWFS 一方面尽量平衡各个资源组的使用率和负载, 另一方面尽量充分利用各个资源组的

存储空间。

在蓝鲸分布式文件系统中, 分片 (stripe, 或称之为条带) 的大小可以根据需要动态调整, 缺省值是 4M 字节。我们将每一个文件的数据分片存储到符合要求的资源组中, 在各个资源组之间依次存储每一个分片, 如图 4.2 所示。分片存储的哈希函数如下:

$$\text{index} = (\text{iblock} + \text{ino}) \div 1024 \bmod \#rg$$

iblock 表示某个文件内部的块号, 每个块是 4096 个字节; ino 是文件的索引节点号; #rg 表示当前可用的资源组数量。计算结果用来选择在哪一个资源组中分配资源, 这个 index 是将所有可用的资源组进行排序以后的编号, 一般按照资源组号的大小进行排序。如果在选择根据计算结果得出的资源组以后发现该资源组不符合分配要求 (比如可用空间比率已经抵达下限、数据带宽不够、负载很高等), 就自动向资源组列表后面搜索, 直到搜索到满意的为之; 或者再次循环到该资源组, 分配失败。

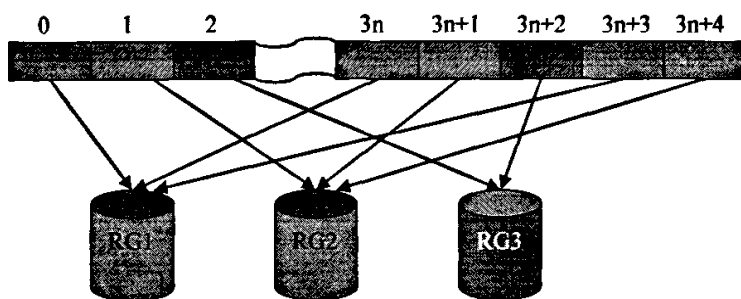


图 4.2 分片存储

(某个文件分片存储到三个资源组中)

蓝鲸分布式文件系统主要利用文件内部数据块号进行单个文件级的数据分片存储, 利用文件索引节点号将各个文件的开始部分分散在多个资源组之间, 避免在小文件的情况下都使用同一个资源组。由于可用的资源组数量是动态变化的, 哈希函数可以动态适应这种变化, 在配置不同数量资源组的情况下均能有效地进行数据分片存储。同时, 资源组数量的变化, 不影响已经进行了分片存储的文件数据的访问, 因为文件数据块的跟踪不需要涉及这个分片存储的哈希函数。

如果资源组数量增加了, 新创建的文件将会被自动分片存储到新加入的资源组中, 原先已经存储的文件可以继续使用, 但是却无法利用新近添加的资源组的存储能力与带宽。为了我们设计了一系列工具, 可以将已经存在的文件重新分散到各个资源组中, 以便更好地实现分片存储的初衷。

#### 4.7 元数据服务器集群

单个元数据服务器能够有效支持的应用服务器数量有限, 特别是在元数

据操作频繁的应用中更是如此,元数据服务器经常成为整个系统的瓶颈所在。为了消除这种瓶颈,多个元数据服务器组成的集群是一种可供选择的方案。

DCFS[31][32]将整个文件系统的根目录划分成多个部分,分别由多个元数据服务器管理。如果根目录包含的某个一级子目录被映射到某个元数据服务器管理,那个该子目录所包含的所有文件和子目录都被映射到此元数据服务器管理。这种映射关系非常简单而又清晰,易于实现。缺点是一旦确定某个子目录的映射关系,此后不再改变。Storage Tank[24]采用了类似的映射策略,也不能实现动态负载平衡,容易造成某个元数据特别忙,而其它的元数据服务器却相对比较空闲。

蓝鲸分布式文件系统抛弃了上述两个分布式文件系统采用的固定映射策略,采用全动态映射策略——系统中的任意活跃元数据可以映射在任意一个元数据服务器上,我们称这种映射关系叫做“绑定”。蓝鲸分布式文件系统的绑定关系包含如下几层含义:(1)只有活跃元数据(active metadata)才被绑定。活跃元数据是指正在使用的元数据,或者刚刚被使用过的元数据。不活跃的元数据是指那些暂时没有被访问到的元数据,或者是被访问了很长时间以后短期内再也没有被访问的元数据。不活跃的元数据没有绑定关系,一方面是考虑到动态绑定的特性,也就是说所有的绑定关系是在元数据即将成为活跃元数据的时刻才被决定绑定在某个元数据服务器上;另一方面是考虑到实际消耗内存资源和查找速度的问题,因为记录全部的绑定关系必然消耗较多的内存,增加各种操作的复杂度。(2)任意绑定关系。任何一个活跃元数据可能绑定在任意一个元数据服务器上,这种绑定关系是由绑定服务器根据一定策略作出的,后面一章将有具体介绍。这样任意绑定关系消除了系统可能存在的固有的不平衡特性,比如按照目录绑定的话,可能某些目录的访问频率比别的目录大很多倍,失去了负载平衡的优势。(3)单一绑定关系。在任何时刻,一个活跃元数据只能绑定在某一个元数据服务器上,所有元数据服务器上绑定的活跃元数据没有交集,它们的合集是全体活跃元数据。(4)动态改变。由于负载平衡或者特殊需要,活跃元数据的绑定关系可以动态改变。(5)绑定关系不具有持久性。也就是说活跃元数据的当前绑定关系在系统全部重新启动运行之后将不再保留,全部重新绑定。图 4.3 演示了在某个时刻活跃元数据绑定在三个元数据服务器上的情形。

元数据绑定的最主要目的是确定元数据的操作权限,也就是说该元数据可以由谁负责读取、修改等。只有绑定在某个元数据服务器上的活跃元数据才能被该元数据服务器拥有并具有管理权限。

每一个元数据服务器上都有本地绑定表(Local Binding Table, LBT)来确定某个元数据是否绑定在本地。只有绑定在本地的元数据才能被此元数据服务器修改,没有绑定在本地的元数据只有一个“存根”(stub)数据结构,

用来形成目录树结构。我们在 EXFS 的每一个内存索引节点中添加一个数据项，记录该索引节点是否绑定在本地。如果索引节点没有绑定在本地，则该数据项指示其被绑定在何处，此时这个索引节点就是一个“存根”，只是用来形成目录树，不能用来读写索引节点。如果与某个元数据相关联的索引节点被从内存中删除了（比如因为文件被删除、文件关闭一段时间以后），我们将此索引节点记录的绑定关系转移到一个独立的哈希链表（空闲绑定表）中，继续保持一段时间，而不是马上申请解除该元数据的绑定关系。这样的策略主要是考虑到元数据使用的时间局部性，希望在不久的将来再次使用时，可以节约重新要求绑定服务器实施绑定的开销。为了节约内存空间，BWFS 使用一个独立的内核进程，定时扫描空闲绑定表，将那些已经许久没有使用过的绑定关系解除，释放内存。

在下一章我们将介绍绑定的策略。

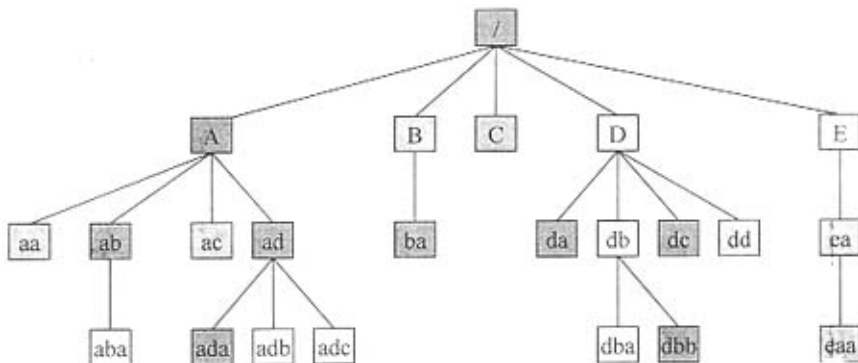


图 4.3 活跃元数据绑定到三个 MS

（三种颜色分别对应三个元数据服务器）

#### 4.8 分布式日志

在本地文件系统系统中，整个文件系统由单个操作系统或者是单一映像的操作系统管理，带日志技术的文件系统一般也是采用单个日志系统 [50][51]。蓝鲸分布式文件系统可以拥有多个元数据服务器，它们协同管理整个文件系统的目录树结构。因此我们设计了蓝鲸分布式文件系统的分布式日志。

蓝鲸分布式文件系统采用共享磁盘的结构，每一个元数据服务器、应用服务器等都可以同等地访问所有的存储资源。因此我们为系统中的每一个元数据服务器申请了独立的日志存储空间，它们可以将各自的日志记录到不同的日志区域中，相互不会造成任何干扰。同时由于共享磁盘的结构，使得某一元数据服务器由于种种原因失效以后，其它元数据服务器可以迅速读取它的日志数据，进行快速日志恢复和重建，其它节点可以继续正常工作，提高

系统的可用性。

如果某一个操作涉及到多个元数据服务器，需要它们合作才能完成，那么此操作将是分布式事务处理。为了降低实现的难度以及获得更加快捷的日志恢复，我们为蓝鲸设计了这样的策略：将需要多台元数据服务器合作完成的操作集中到某台服务器上完成。通过将某些元数据暂时绑定到同一台元数据服务器，蓝鲸分布式文件系统实现操作的集中，以便更好地利用日志记录文件系统的修改，保证文件系统的一致性与完整性。

#### 4.9 小结

本章首先分析了蓝鲸分布式文件系统所面临的主要挑战和需要重点解决的问题，随后讲述了我们的应对策略：批量申请资源/异步释放资源、数据分片存储、空间资源分配策略、元数据服务器集群技术、分布式日志等。这些围绕资源管理开展的研究与优化工作对于蓝鲸分布式文件系统的整体性能、可扩展性、高可用性等巨大作用。



## 第五章 元数据绑定

元数据服务器负责处理蓝鲸分布式文件系统的元数据，代表客户端进行资源分配和释放，组织磁盘结构等。单个元数据服务器的性能和可扩展性毕竟有限，为此我们引入了多元数据服务器组成的机群结构。在这样的结构中，绑定服务器负责活跃元数据在各个元数据服务器之间的映射关系，也就是决定元数据的管理权限归哪一个元数据服务器拥有。绑定服务器对元数据服务器提供这种绑定服务，协调它们之间并行元数据的操作，维护整个系统的一致性和进行负载平衡。

绑定服务器根据元数据之间的相关性、当前系统的负载、用户的特定需要等因素进行元数据绑定。绑定关系可以随着各种因素的变化动态改变。

本章涉及的内容在[57][58]中有详细的讨论。为了整体内容的完整性，我在这里总结一下元数据绑定服务的相关内容。

### 5.1 组织架构

从图 2.3 中我们可以看出，绑定服务器处在元数据服务器“后端”，作为元数据服务器之间的协调者工作。绑定服务器只在系统运行时刻协调活跃元数据在各个元数据服务器之间的映射关系，它本身不承载任何关于系统的持久信息。

元数据服务器在访问或者新建任何元数据的时候，检查此元数据的绑定关系，只有绑定在本地的元数据才能被该元数据服务器修改。如果某个元数据没有绑定在本地，那么它会指导应用服务器向正确的元数据服务器申请服务。但是从用户和应用程序的角度看，整个系统是一个具有统一名字空间、单一映象的分布式文件系统，元数据在各个元数据服务器之间的映射关系对它们是透明的，而且元数据映射关系的转移对它们也是透明的。

蓝鲸分布式文件系统中的元数据以索引节点号为唯一标识。元数据服务器在向绑定服务器查询和申请绑定服务时，将需要绑定的元数据的索引节点号、该索引节点的父节点的索引节点号、绑定策略等通过远程过程调用发送给绑定服务器。绑定服务器会根据当前的绑定策略、当前的系统负载的因素决定该元数据与元数据服务器之间的绑定关系。如果绑定调用的返回结果表明此元数据绑定在本地，那么就由该元数据服务器负责管理此元数据，否则将对此元数据的访问转移到其绑定的元数据服务器，本地记录一个“存根”，以便形成一棵连续的目录树。

## 5.2 影响绑定的因素

为了提高系统的性能和可扩展性,蓝鲸分布式文件系统引入了多个元数据服务器组成的集群技术。多个元数据服务器并行工作,共同管理整个文件系统的元数据资源,就存在冲突的可能和负载均衡的问题。为了解决这些问题,我们引入了绑定服务器,它负责管理元数据在各个元数据服务器之间的映射关系,也就是绑定关系。

那么到底有哪些因素可能影响绑定服务器作出如何映射的决定呢?下面我们将讨论影响绑定的重要因素。

- 1) 上下文关系。每一个元数据都处在一定的上下文中,至少都处在同一个蓝鲸分布式文件系统中,它们之间必然存在某种联系,比如是同一颗目录树的不同节点。同时,这些元数据可能还有应用级的关联关系:比如同属于一个应用程序,将会被顺序访问;或者是分别属于不同部门的数据,没有应用级的联系(没有联系也是一种关系)。这些上下文关系是元数据访问的一种特性,而文件系统访问的局部性又加强了上下文关系的重要性。利用好这些上下文关系可以显著提高元数据访问的性能,不至于出现绑定的抖动现象等。
- 2) 负载情况。采用多个元数据服务器的结构就是希望利用元数据并行处理技术来提升系统性能,于是我们尽量将元数据的负载平均分担到各个元数据服务器。绑定服务器利用采集到的元数据服务器的负载情况,决定绑定策略。
- 3) 特定需求。在某些特定情况下,元数据服务器需要将某些特定元数据绑定在自身,才能完成特定的操作,或者是有利于完成这样的操作。在这种情况下,元数据服务器就向绑定服务器申请这些涉及的元数据的绑定关系,而绑定服务器因此也获知这样的信息,于是满足元数据服务器的请求。
- 4) 管理员的配置。管理员可以根据需要配置特定的绑定关系。

以上这些因素都是影响绑定服务器作出绑定决定的因素。这些因素有一定的优先级别,对于每一次特定的绑定都可能有不同的优先次序,视具体情况而定。

## 5.3 绑定策略

蓝鲸分布式文件系统是一个管理海量存储空间分布式文件系统,需要提供很高的性能和良好的可扩展性,同时需要提供在线动态平衡的功能,因此蓝鲸分布式文件系统的绑定算法需要设计得简单而又灵活,这样才能有好

的性能;需要具有公平的策略,使得每个元数据服务器尽量均分所有的负载;同时,尽量不破坏文件系统本身具有的逻辑性,比如某些文件都属于同一个应用程序,它们的访问具有局部性等。

每一个元数据服务器上都有监控程序,定时采集各个元数据服务器的负载情况。绑定服务器根据这些负载情况,考虑其它因素做出绑定决策。元数据服务器在申请绑定服务时,可以指定希望的绑定类型,比如绑定在自身、绑定在其它元数据服务器等。元数据服务器每次在申请绑定服务时,都是有一定原因的,比如新建一个文件或者目录、访问一个未曾访问过的文件等。元数据服务器在申请绑定服务时,将申请的目的传递给绑定服务器,以便绑定服务器做出更好的决策。

#### 5.4 动态负载平衡

活跃元数据在被首次访问时由绑定服务器决定绑定在某个元数据服务器上,之后有关此元数据的操作都是由该元数据服务器完成。但是这种绑定关系不是一成不变的,它可以随着用户的特定需要随时改变,也可能是出于负载均衡的目的需要重新绑定部分元数据在各个元数据服务器之间的映射关系。

蓝鲸分布式文件系统在每个节点都有监控程序随时采集节点的运行状况,其中包括节点的负载情况。绑定服务器可以获得所有元数据服务器当前的负载情况,并对这些负载情况加以评估。如果绑定服务器发现系统中的所有元数据服务器都出于比较繁忙的状态,其程度已经高出了预先设置的上限,那么它会报告管理员,希望为系统添加元数据服务器;如果绑定服务器发现系统中有些元数据服务器的负载较高,有些服务器的负载较低,并且存在较大的差异,甚至有些已经超过最高上限时,就认为系统需要进行负载平衡。前一种情况下如果添加了新的元数据服务器,就相当于后一种情况。于是在这些情况下,绑定服务器就会通知负载较高的元数据服务器,转移一部分元数据的绑定关系。被通知可以转移掉部分元数据绑定关系的高负载的元数据服务器会首先要求解除部分元数据的绑定关系,然后再重新申请这些元数据的绑定。绑定服务器在收到绑定申请的时候,就可以将这些元数据绑定在负载较轻的元数据服务器上,达到负载平衡的目的。

为了不至于出现抖动,负载平衡时机的选择非常重要。一方面要保证这种负载平衡具有较好的敏感度,能够及时进行负载平衡;另一方面,由不能太频繁地进行负载转移,毕竟绑定转移也需要一定开销,因为绑定转移了以后还涉及到应用服务器上元数据映射的转移(下一章介绍)。系统中节点的负载与应用程序的具体行为有直接的关系,也与用户的使用方式、外界事务的

变化有着一定的联系。某些系统可能会出现负载突变的情况，某些系统只有较平缓的变化。于是负载均衡的灵敏度设置还不能一概而论，一般需要在实际应用环境中反复测试和不断调整才能获得最佳的效果。

### 5.5 小结

本章主要描述了绑定服务器如何在各个元数据服务器之间实现元数据绑定，以及影响绑定策略的因素、绑定的策略和动态负载均衡等。这些技术都是有效实现元数据绑定、提高系统性能和可扩展性的关键所在。

## 第六章 客户端的资源使用

蓝鲸分布式文件系统的客户端目前可以运行在 Linux 和 Windows 2000 操作系统中,是利用操作系统提供的虚拟文件系统(Virtual File System, VFS)或者是可安装文件系统(Installable File System, IFS)接口实现的真正内核级的文件系统。应用程序可以通过文件系统相关的应用程序接口(Application Program Interface)或者系统调用(System Call)获得文件系统服务。正因为如此,蓝鲸文件系统实现了真正的二进制兼容性,使得原有的应用程序无需进行任何修改就可以使用蓝鲸分布式文件系统。蓝鲸分布式文件系统向应用程序提供统一的名字空间,实现文件系统的单一映象。

本章解释了 NFS 在数据传输方面存在的瓶颈,说明了蓝鲸分布式文件系统采带外数据传输的缘由,以及利用缓存块映射信息的技术解决由此带来的性能问题。

### 6.1 应用服务器的系统架构

蓝鲸分布式文件系统应用服务器的架构如图 2.4 所示,由一系列应用程序工具和可加载内核模块组成。应用程序级的工具主要包括 mount 工具、监控程序、自动化运行脚本、配置文件等,可加载内核模块主要是一个符合 VFS (或者 IFS)接口的文件系统模块和共享虚拟磁盘设备的模块。

在蓝鲸分布式文件系统中,应用服务器需要与元数据服务器集群、存储服务器集群、资源组管理等节点或者模块进行通信。应用服务器与元数据服务器之间通过 TCP/IP 网络通信,使用远程过程调用交换信息。蓝鲸分布式文件系统应用服务器与元数据服务器之间的通信继承了 NFSv3 协议,并进行了扩展。应用服务器与存储节点之间可以使用以太网、光纤、InfiniBand 等计算机网络互连。一般情况下,为了节约初次建造的成本和管理运营费用,我们采用千兆以太网互连所有的应用节点和存储节点。

蓝鲸分布式文件系统采用了带外(Out-of-Band)数据传输模式,应用服务器和元数据服务器交换文件系统的元数据信息,文件数据直接应用服务器和存储节点之间交换。这样可以避免文件数据经过元数据服务器转发,减少数据传输的路径,克服传统的分布式文件系统(比如 NFS 和 CIFS)的瓶颈,提高系统的性能和可扩展性。

## 6.2 数据带外传输模式

传统的分布式文件系统 NFS 以及 CIFS 都采用单个服务器的结构, 采用服务器/客户机的管理和数据传输模式[13][14]。客户端的所有元数据服务都通过服务器获得, 客户端的所有文件数据也是由服务器提供。

蓝鲸分布式文件系统是蓝鲸大规模网络存储的核心组件, 管理海量存储空间。这些存储空间由分布于多个存储节点上的物理存储空间组成, 并且可能随着应用程序需求的提升随时添加新的存储设备。如果继续按照 NFS 的结构模式, 客户端所需要的所有元数据和文件数据都由单个元数据服务器提供, 那么必然出现数据处理的瓶颈, 因为文件数据由元数据服务器转发, 增加了数据传输的路径, 增加了元数据服务器的负载。我们为蓝鲸分布式文件系统设计了“带外”数据传输模式: 客户端的所有元数据由元数据服务器提供, 客户端所访问的文件数据直接与应用服务器与存储节点之间传输, 减少中间转发的过程, 避免瓶颈产生, 提高系统性能。在这种情况下, 控制信息(元数据)不在数据传输的路径上, 所以称为“带外”模式, 参见图 2.1。

在 NFS 中, 元数据和数据都由服务器负责管理, 客户端不直接与存储设备交换数据, 客户端访问文件数据的时候, 只要指定文件的索引节点号和文件内部的相对位置, 由服务器负责根据这些信息存取数据。在 Lustre 和 Panasas, 这部分功能由对象存储设备完成。在蓝鲸分布式文件系统中, 应用服务器直接与存储节点交换数据, 也就是通过安装在其上的共享虚拟磁盘驱动存取数据。在进行数据存储之前, 它需要将索引节点号和文件内部的相对位置转化为共享虚拟磁盘的地址, 也就是全局逻辑地址。

为了将文件内部的相对位置转化成全局逻辑地址, 我们添加了一个客户端和元数据服务器之间的远程过程调用, 叫做 `getblock`。它的主要任务就是将索引节点的节点号、文件内部的相对位置、是否是新创建的块等参数传递给元数据服务器, 元数据服务器根据这些参数决定是否为该文件新分配数据块, 或者是查询相关数据结构获得文件内部的相对位置与全局逻辑地址之间的映射关系, 返回给客户端。文件内部的相对位置一般都是以文件内部的块号表示, 这里的块就是文件系统为文件分配数据块的最小粒度, 在蓝鲸分布式文件系统中, 此块的大小是 4096 字节。客户端在获得了文件内部块的映射以后, 就可以利用相关的内核函数, 访问共享虚拟磁盘, 存取数据了。

应用服务器在访问任何普通文件的数据之前, 都必须获得文件内部块号和全局逻辑地址的映射关系。元数据服务器根据客户端提供的参数分配或者查询这样的映射关系, 返回给客户端。多个应用服务器可以独立地、并发地访问相同的数据或者不同的数据, 而真正的数据传输工作不需要元数据服务器的参与, 减轻了元数据服务器的负担, 减少了数据传输的路径, 显著提高

了系统性能。

### 6.3 元数据信息的缓存

应用服务器访问的所有元数据都由元数据服务器提供，任何更改的元数据必须由元数据服务器负责写回到持久介质。如何保持元数据信息在应用服务器与元数据服务器之间的同步，以及各个应用服务器之间的一致性，是蓝鲸分布式文件系统必须回答的一个问题。蓝鲸分布式文件系统遵循 NFS 的共享语义，实现跟随时间的弱同步——应用服务器在首次取得某个元数据信息以后，在一定时间间隔内，使用缓存中的此元数据信息，只有当超时或者用户显式要求时，才从元数据服务器更新此元数据。使用跟随时间的弱同步语义，使得应用服务器不至于忙于与元数据服务器的通信，提高效率。当然这样也有可能造成应用服务器使用过时的元数据，比如读到过时的数据、取得过时的属性等。应用程序在访问蓝鲸分布式文件系统时，就像访问 NFS 一样，只要从应用程序的角度尽量避免出现这样的情况就可以了。

相比 NFS，蓝鲸分布式文件系统的客户端多了关于文件内部块号与全局逻辑地址之间的映射信息。这个信息是应用服务器在访问文件数据时从元数据服务器那里申请 `getblock` 服务获得的。如果应用服务器不缓存该信息，势必使得其在每次访问一个系统缓存中不存在的数据时，都需要向元数据服务器申请服务，这将造成整个系统的数据访问性能低下。为了消除这种状况，我们设计了块映射信息的缓存机制：每次从元数据服务器获得多个块的映射信息，并且缓存这些映射信息，以便后续使用。下文将介绍如何存储这些映射信息、如何获取这些信息、信息如何替换、信息何时失效等。

#### 6.3.1 块映射信息的存储

应用服务器在访问每一个普通文件时使用块映射信息将文件内部块号转化为全局逻辑地址，这些信息与每一个普通文件相关联。文件系统在访问每一个文件时，都会使用一个对应的内存索引节点来表示。我们使用内存索引节点的数据结构中未使用的内存空间，或者是为每一个内存索引节点的数据结构分配额外的内存空间，来存储对应普通文件的块映射信息。如果是目录文件或者其它特殊文件，对应的索引节点将没有块映射信息。在使用 Red Hat Linux 8.0 中，我们使用了操作系统已有的索引节点内部没有使用的 120 个字节存储块映射信息，在其它系列的操作系统中，可能需要额外分配一定的空间存储块映射信息。用来存储块映射信息的内存空间是有限的，一方面是由于操作系统中的索引节点内部的空余空间有限，另一方面由于内核内存空间很宝贵，不能分配很多额外的内存空间。

为了使用有限的内存空间表示更多的块映射信息,蓝鲸分布式文件系统使用基于范围(extent-based)的表示方式:内部起始块号+全局逻辑地址+块数。使用这种方式表示的基础是有一定数量连续的内部块号对应一定数量的连续全局逻辑地址。为了提高这种表示方式的有效性,蓝鲸分布式文件系统尽量为每一个文件分配连续地址的存储空间,存储空间的预分配是一个很好的解决方案。蓝鲸分布式文件系统预分配的数量是可以配置的,一般情况下预分配的大小是 1M 字节,也就是 256 个数据块。较大的预分配块可以保证较好的数据块连续性,减少应用服务器与元数据服务器之间交换块映射信息的次数,提高性能。蓝鲸分布式文件系统将这些基于范围的表示组织成一个列表,以供存取数据的时候使用。

### 6.3.2 块映射信息的替换

用来存储块映射信息的内存空间是有限的,相对一个较大的文件来说,这些缓存的块映射信息还是少量的,因此必然存在块映射信息的替换。蓝鲸分布式文件系统使用最近最少使用(LRU, Least Recent Used)策略替换过时的块映射信息。在应用服务器需要访问文件的数据时,首先检查缓存的块映射信息是否含有符合需求的信息,如果有,则使用该信息存取数据;如果没有,则需要向元数据服务器申请块映射信息。在此次使用完了块映射信息以后,就将该映射信息转移到列表的头部,表示最近使用的信息。如果在向元数据服务器申请块映射信息以后,列表已经满了,那么最后的那个元素将被替换掉,这样可以保证缓存的信息都是最新的。

### 6.3.3 块映射信息的失效

蓝鲸分布式文件系统遵循 NFS 的同步语义,客户端每隔一定时间刷新本地缓存的元数据信息。应用服务器更新了本地的元数据信息以后,如果发现自上次更新以来文件的修改时间戳已经改变,那么就需要重新获取块映射信息。这样,先前此文件的块映射信息就全部失效,系统缓存的此文件的所有数据块也全部失效。应用服务器再次访问此文件的数据块时,就必须重新申请块映射服务,重新读取数据块内容。

### 6.3.4 性能测试和对比

本小节主要对比测试了蓝鲸分布式文件系统在使用块映射信息缓存技术前后的性能。我们在测试环境中配置一个客户端,一个存储节点,以及必要的系统服务器,它们之间通过千兆以太网互连。每一个结点均采用 Intel Xeon 2.40GHz 的 CPU,客户端节点配备 1024MB 内存,存储节点配备 3072MB 内存。存储节点采用 3ware 9500 SATA RAID 控制器连接 10 块 160GB 7200



RPM 的 Seagate 硬盘,配置成 RAID 0 的磁盘阵列。所有的测试均为采用 Linux 的命令 dd 顺序读写一个大文件获得,并且所有测试开始前均没有任何该大文件的缓存。

我们测试的结果如表 6.1 和表 6.2 所示,其中横向数据表示客户端每次从元数据服务器申请映射的块的个数,分别是 1、16、64、256 和无穷大,纵向表示每次读写数据块的大小。无穷大表示一次为测试文件分配所有的数据块,并且连续,客户端只需要一次映射就能获得所有的块映射信息。测试结果用图形表示分别如图 6.1 和图 6.2。从中我们可以看出,蓝鲸分布式文件系统采用每次映射多个数据块的策略大大提高了系统的读写性能。

表 6.1 蓝鲸分布式文件系统读性能 (单位: KB/sec)

Block Size \ Cache Blocks	Read with 1 cache block	Read with 16 cache blocks	Read with 64 cache blocks	Read with 256 cache blocks	Read with $\infty$ cache blocks
1K	11586	72315	69905	67650	66363
2K	11902	73843	71332	71820	69905
4K	12108	78643	75984	71331	69905
8K	12192	77672	73326	71331	70849
16K	12577	78252	76538	71820	71331
32K	12577	75984	77672	73843	74898
64K	13443	73843	78840	75983	75437

表 6.2 蓝鲸分布式文件系统写性能 (单位: KB/sec)

Block Size \ Cache Blocks	Write with 1 cache block	Write with 16 cache blocks	Write with 64 cache blocks	Write with 256 cache blocks	Write with $\infty$ cache blocks
1K	10412	59918	67216	69905	69442
2K	10527	63167	68685	69905	68985
4K	10538	64860	69136	69905	69905
8K	10623	64329	68985	69905	69905
16K	10667	64329	68985	69905	69905
32K	10699	64329	69905	69905	69905
64K	10776	64727	69905	70734	70734

从以上测试结果所得的图表中我们可以看到:通过一次请求获得多个块映射信息,减少了交换块映射的网络通信、开销与延迟;通过有效的缓存存储机制、替换机制和失效机制,尽量减少内核内存空闲占用和保持信息有效性。测试结果符合我们预期的目标,大大提高了系统的性能,说明有效的块映射信息缓存有助于提高系统的性能。但是我们也看到,当缓存一定量的块映射信息以后,蓝鲸分布式文件系统的性能趋于稳定,缓存的数量对于性能

没有明显的影响。因此我们在实际和实现蓝鲸分布式文件系统时,一般选择缓存 256 个数据块的映射信息。

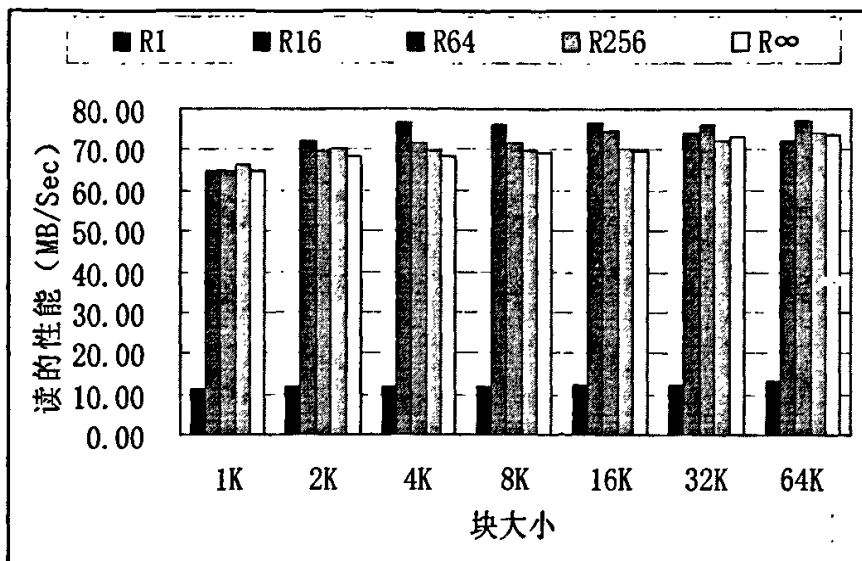


图 6.1 蓝鲸分布式文件系统读性能

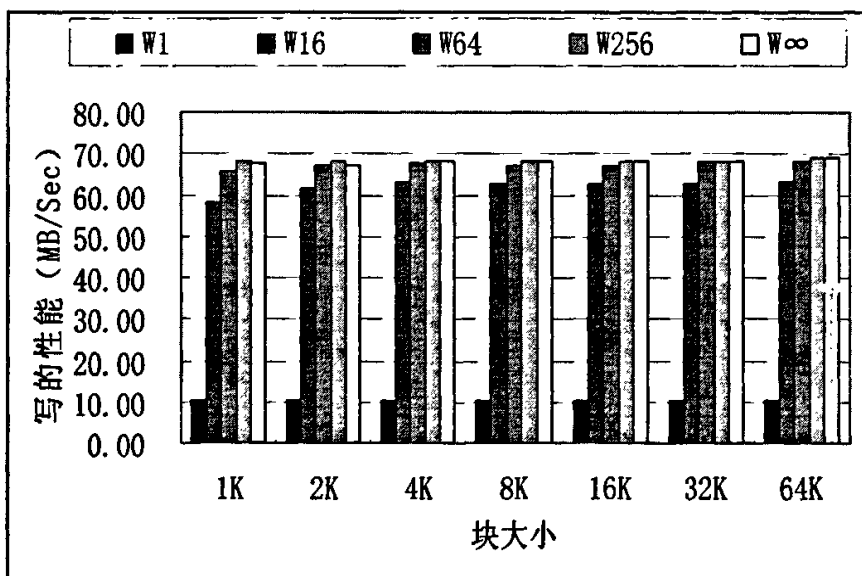


图 6.2 蓝鲸分布式文件系统写性能

#### 6.4 文件映射

在蓝鲸分布式文件系统中,多个元数据服务器协同工作,共同管理元数据操作。绑定服务器负责元数据在各个元数据服务器之间的映射关系,进行负载平衡。在任何时刻,活跃元数据只能绑定在某一个元数据服务器上。应用程序通过蓝鲸分布式文件系统的客户端软件访问文件系统时,应用服务器

需要确定所访问的文件涉及的元数据绑定位置，也就是由哪一个元数据服务器提供服务。于是，应用服务器需要记录每一个元数据对应的元数据服务器，以便能够访问它们。我们将这种关系叫做“文件映射”，如图 2.3。

应用服务器在连接到蓝鲸分布式文件系统进行系统初始化的时候，确定了整个文件系统的“根目录”的绑定位置。在访问到根目录下面的文件或者目录时，应用服务器会首先向绑定根目录的元数据服务器发出文件服务器请求，也就是假设它们被绑定在与父目录相同的元数据服务器上。如果即将访问的文件或者目录确实绑定在此元数据服务器上，那么此次访问将由此元数据服务器负责；否则，此元数据服务器会查询自身的本地绑定表，或者向绑定服务器查询需要访问的元数据的绑定位置，并向应用服务器返回访问不成功的信息，并指示该元数据目前绑定的元数据服务器的地址。于是应用服务器就可以修改此元数据的绑定地址，并向正确的元数据服务器申请服务。在极少数情况下，这种转移可能经历多次才能得到正确的绑定关系。

在发生了元数据的动态负载平衡以后，元数据在元数据服务器之间的绑定关系改变了，但是应用服务器记录的元数据的绑定关系却没有得到更新。这主要是为了节约动态负载平衡的开销，让应用服务器主动更新自身记录的绑定关系。于是应用服务器在某次根据自身记录的绑定关系向元数据服务器申请服务的时候，可能得到一个应答，指示元数据的绑定关系已经改变，并将正确的元数据服务器的地址返回。应用服务器根据此地址修改自身的绑定关系，并向其申请文件服务。在极少数情况下，这种转移可能经历多次才能得到正确的绑定关系。图 6.3 显示了在发生元数据转移以后重新建立文件映射的过程。

应用服务器上的这种文件映射对应用程序来说都是透明的。蓝鲸分布式文件系统对应用程序提供了单一映象的文件系统，提供了兼容 NFS 的语义。应用程序无法得知多个元数据服务器的存在，也无法感知元数据的动态分布和迁移。

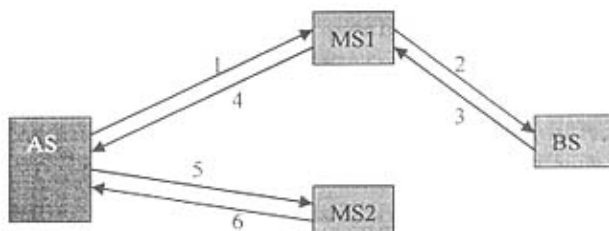


图 6.3 重新建立文件映射的过程

(1. 申请元数据服务 2. 查询新的绑定关系

3. 返回新的绑定位置 4. 返回错误并指示新的元数据绑定位置

5. 向新的元数据服务器申请服务 6. 返回正确结果)

## 6.5 数据一致性

在 NFS 协议或者 CIFS 协议中,所有的数据存取都是由元数据服务器负责进行的,即使某几个应用服务器在某次写操作时,各自只写入了几个字节数据,服务器也会保证数据的正确性。在蓝鲸分布式文件系统中,客户端得到文件的块映射信息之后,直接以“页”(page)的方式读写数据——文件系统读写磁盘的最小单位是“页”,在 Linux 操作系统中,通常页的大小是 4096 字节。如何保证蓝鲸分布式文件系统同样实现 NFS 的语义,就必须解决下面的问题[59]。

假设有如下情况发生:应用服务器 A 和应用服务器 B 都同时读取某个文件的第一块,4096 字节的数据。在这两个操作都完成之后,应用服务器 A 和应用服务器 B 内存中都含有第一个数据块的缓存,它们含有相同的数据。之后应用服务器 A 改写了此文件的开始 10 个字节(假设这是一条记录的长度)的内容,然后应用服务器 B 跳过头 10 个字节,直接改写了第二个 10 个字节的内容,然后应用服务器 A 和应用服务器 B 顺序地将它们修改过的缓存写回到磁盘。这样处理的后果是应用服务器 A 所作的修改被应用服务器 B 写入的内容覆盖了,从结果看,似乎应用服务器 A 没有写入任何修改的数据。这个结果显然违背了 NFS 的共享语义。

蓝鲸分布式文件系统设计了一套用于处理应用服务器之间数据一致性的模型:任何应用服务器在写入整页数据时,都可以直接修改,这不会破坏数据的一致性;在写入非整页数据时,必须向元数据服务器申请写入的令牌,只有取得令牌的应用服务器才有权限写入非整页数据,数据写完之后马上释放令牌。系统中有多个令牌,每一个文件的每一个页(大小刚好等于一个块)都有一个令牌,也只能生成一个令牌。令牌只有在申请的时候才生成,在使用完毕之后就会删除。使用这种方式,可以同步各个应用服务器之间的写入操作,提高整个文件系统的数据一致性。

## 6.6 资源定位

应用服务器、元数据服务器等都需要通过共享虚拟磁盘访问分布在多个存储节点上的数据。它们在访问这些数据时,使用全局逻辑地址,以块设备接口的方式读写数据。那么蓝鲸分布式文件系统怎么处理全局逻辑地址与各个存储节点上的物理存储设备之间的映射关系呢?

全局逻辑地址管理器(Global Logical Address Manager)负责将整个系统中的存储节点上的存储空间统一管理,形成线性的 64 位地址——全局逻辑地址(Global Logical Address)。全局逻辑地址管理器记录全局逻辑地址与各个

存储节点以及其上的物理存储空间的映射关系，向系统中的所有需要通过全局逻辑地址访问存储空间的节点提供查询服务，实现资源定位。

应用服务器通过改进的 NBD 或者 iSCSI 驱动程序等访问共享磁盘。这些驱动程序接收的访问请求全都使用全局逻辑地址表示，驱动程序需要查找全局逻辑地址对应的存储节点以及其上的物理存储设备。如果使用千兆以太网，不同的存储节点使用不同的 IP 地址标识，不同的物理存储设备使用不同的网络监听端口标识。驱动程序在获得了全局逻辑地址对应的 IP 地址及端口之后，就可以将块设备访问请求通过 NBD 协议或者 iSCSI 协议发送到存储节点上的服务程序。

全局逻辑地址与存储节点以及物理存储设备之间的映射关系不会在系统运行时刻动态改变，因此应用服务器上的驱动程序可以缓存两者之间的映射关系。这样，映射关系只有在第一次访问的时候需要查询，在以后的访问中无需再次查询，可以直接使用缓存的结果。这些缓存只会记录在应用服务器的内存中，在应用服务器重新启动或者蓝鲸分布式文件系统彻底退出的时候就会消失，下次需要使用时再次向全局逻辑地址管理器查询。采用这样的方式，全局逻辑地址与存储节点以及物理存储设备的映射关系可以在蓝鲸分布式文件系统不对外提供服务的情况下改变，在系统重新对外提供服务时就可以获得新的映射关系，实现一定程度的数据迁移和负载平衡。

## 6.7 小结

本章首先描述应用服务器的带外数据传输技术，它使得蓝鲸分布式文件系统避开了 NFS 系统中的数据传输瓶颈。然后研究了元数据信息的缓存策略、缓存替换、缓存失效等有关内容，指出有效的块映射信息缓存极大地提高了系统的性能。之后我们又阐述了蓝鲸分布式文件系统如何解决由于动态绑定造成的文件映射问题，以及由于带外数据传输模型带来的数据一致性问题。最后解释了蓝鲸分布式文件系统如何将全局逻辑地址映射到具体的存储节点以及物理存储设备。

## 第七章 性能分析与评价

大型系统的性能测试与分析是一项庞杂的任务,涉及到多个方面的工作,而且测试结果对于不同的环境、不同应用、不同使用模式有不同参考价值[60][61][62][63][64][65][66]。蓝鲸分布式文件系统是蓝鲸大规模网络存储系统的核心组件,应用的范围非常广泛,应用的模式千差万别,应用系统的规模也是各不相同,因此进行系统性能的测试、衡量与分析是一项复杂而艰巨的工作。我们在本章中描述的测试分析尽量贴近用户使用的环境,选取典型的应用加以测试,希望这样的测试结果对用户的使用和系统的开发有指导意义。

本章中我们首先设定了评价蓝鲸分布式文件系统的衡量标准,通过这些标准反应系统的性能和可扩展性,并且我们将蓝鲸分布式文件系统的测试结果与 NFS 的测试结果进行了对比分析。一个分布式系统的性能和可扩展性不仅需要通过各种基准测试和分析,更需要实际应用环境的检验。我们在本章给出了蓝鲸分布式文件系统在应用环境中的运行对比结果,说明蓝鲸分布式文件系统的优势所在。

### 7.1 性能测试的目的

我们有选择地设计了一些测试用例,选取了一些测试工具,对蓝鲸分布式文件系统进行了宏观性能测试。通过对这些测试结果的分析,以及部分地与 NFS 进行性能对比,来验证和说明蓝鲸分布式文件系统采用的分布式分层此资源管理模型的有效性,对蓝鲸分布式文件系统中与资源相关的模块设计与优化的有效性,以及由此带来的整体上较高的性能和较好的可扩展性。同时,我们也会发现系统在某些应用中,蓝鲸分布式文件的性能和可扩展性不尽如人意,希望这些测试结果和对比分析能给系统以后的开发带来借鉴作用和具有一定的参考意义。

### 7.2 性能评价模型

#### 7.2.1 性能衡量的标准

蓝鲸分布式文件系统适用的范围很广,利用蓝鲸分布式文件系统进行海量数据处理的应用种类很多,它们各自定义性能的标准也不一样[67]。为此我们选择了有代表性的几种类型的应用,分别考察了影响这些应用的分布式文件系统的有关指标。下面我们就分别讨论与这些应用关心的有关分布式文

件系统的一些指标:

**数据传输带宽 (Bandwidth)**。数据密集型应用很关注客户端所能获得的数据传输带宽,包括单个客户端的带宽和系统总体的聚集带宽。这类应用的典型代表就是流媒体服务、信息采集服务等。流媒体服务 (Stream Service) 是指提供在线多媒体浏览的服务,特别是以在线观看影片、在线收听广播、在线欣赏音乐等应用为代表。衡量一个流媒体服务器性能最重要的指标就是能够同时支持的客户端数量。一个提供在线观看影片的流媒体服务器,如果每一个流的带宽是 512Kbps,提供 1000 个客户端同时在线,那么大约需要 512Mbps 的带宽能力。如果需要更加清晰的影片,则需要提供更高的带宽。在信息采集服务中,数据传输带宽同样至关重要。如果有更好的带宽,信息采集程序就可以使用更高的采集频率,采集更多的样本,并且同时采集更多的数据流。

**吞吐率 (Throughput)**。吞吐率一般是指单位时间内完成的事务 (Transaction) 的数量。这里的事务处理是指创建文件、删除文件、读写小文件等操作。这些操作占用较少的数据带宽,却占用大量的元数据服务器的计算资源。比如在办公开发环境中,都是以较小文件操作为主,伴随较多的文件创建、删除等;在一个信息处理应用中,有大量临时小文件的创建与删除操作。这些应用频繁访问文件系统,进行各种类型的事务处理,对系统事务处理能力有较大要求。

**可扩展性 (Scalability)**。有些应用既不要求单个客户端有很高的数据传输带宽,也不要求系统具备很好的吞吐率,而是要求系统具有较好的扩展性能,也就是说分布式文件系统能够服务于尽量多的客户端进行并行计算。比如某些进行石油天然气勘探的并行计算应用在每个应用服务器上只需要大约 2MByte/sec 左右的带宽,而且都是读取大文件,几乎没有事务处理过程。如果分布式文件系统能够支持更多的客户端同时进行计算,效率将有较大提高。这种可扩展性能在这些应用中尤其重要。

## 7.2.2 影响性能的因素

影响蓝鲸分布式文件系统性能的因素很多,包括硬件系统所能提供的性能,比如磁盘的输入/输出性能、网络传输性能、存储节点数量、客户端数量等;也有软件系统的因素,包括模型的好坏、算法的优劣、设计与实现的质量等;也有衡量标准和用户应用的因素。我们可以通过配置更好的硬件环境获得更好的性能,但是在通常情况下,系统整体的性能受限于系统最薄弱的环节,这是木桶原理告诉我们的。为了提高整个系统的性能,我们应该最优先解决影响性能的主要瓶颈,优化在影响系统性能中占主导地位的因素。

### 7.2.3 性能评价模型

针对蓝鲸分布式文件系统比较有代表性的应用,我们选取了一些指标作为性能测试和衡量的标准,它们分别是带宽、吞吐率、系统效率、可支持的最大并发客户数量等。蓝鲸分布式文件系统主要应用在机群环境中,因此我们假设在整个机群环境中,客户端的软硬件配置相同或者基本相同,存储节点、元数据服务器等担任各种类型功能的服务器配置都各自基本相同。

我们假设存储节点能够提供的 I/O 读写带宽是  $BW_{io}$ , 应用服务器与存储节点之间网络带宽是  $BW_{net}$ , 总共有  $N_{sn}$  个存储节点,  $N_{as}$  个应用服务器,  $N_{ms}$  个元数据服务器; 每一个应用服务器获得的数据传输带宽是  $BW_{as}$ , 系统获得的聚集带宽是  $BW_{ys}$ ; 每一个应用服务器获得的吞吐率是  $T_{as}$ , 系统获得的聚集吞吐率是  $T_{ys}$ ; 系统的效率因子是  $E$ , 在不同配置下可能有不同的效率因子, 它是个变数。单个应用服务器与单个存储节点之间的最大数据传输能力受限于  $BW_{io}$ 、 $BW_{net}$  两者的较小值, 也就是说在配置单个应用服务器和单个存储节点的蓝鲸分布式文件系统中, 应用服务器所能取得的最大数据传输带宽是:

$$BW_{ys} = BW_{as} = \min(BW_{io}, BW_{net}) \times E \quad \text{公式 (1)}$$

在配置多个应用服务器和多个存储节点的蓝鲸分布式文件系统中, 系统能取得的最大聚集带宽是:

$$BW_{ys} = \min(BW_{net} \times N_{as}, \min(BW_{io}, BW_{net}) \times N_{sn}) \times E \quad \text{公式 (2)}$$

在公式(2)中, 前一个  $BW_{net}$  表示 AS 所能获得的最大网络带宽, 后一个  $BW_{net}$  表示 SN 所能获得的最大网络带宽, 两者可能具有不同的值。由于系统的聚集性能受到这些参量的最小值制约, 所以在进行系统配置的时候, 应该尽量选用性能较好的网络与磁盘设备, 避免硬件环境成为瓶颈。

蓝鲸分布式文件系统事务处理的能力主要取决于元数据服务器的性能、元数据服务器的数量、存储节点的延迟等。我们希望通过增加元数据服务器的方式线性提高系统的吞吐率, 下面公式表示最理想的结果:

$$T_{ys} = \sum T_{as} = T_{as} \times N_{as} \sim N_{ms} \quad \text{公式 (3)}$$

公式(3)表示, 系统的聚集吞吐率是所有 AS 吞吐率的综合, 在平均状态下



是每一个 AS 吞吐率的  $N$  倍 ( $N$  为 AS 的数量)。在理想状态下, 系统的聚集吞吐率与 MS 的数量成正比, 体现了系统随着 MS 数量的增加具有良好的可扩展性。

在蓝鲸分布式文件系统的性能测试中, 我们会在不同情况下测试效率因子  $E$  的值, 此值越大, 说明系统的效率越高, 协议开销损失越小。同时我们还特别关注配置多个存储节点的情况, 系统聚集性能的变化。如果我们计算在配置多个存储节点下的聚集性能和配置单个存储节点下的聚集性能的比值, 那么此值越高, 说明系统随着存储节点增加的可扩展性越好。我们将此值定义为系统随存储节点的加速比 (Speedup):

$$Speedup = \frac{N_{\text{个存储节点时的 } BW_{\text{sys}}}}{1\text{个存储节点时的 } BW_{\text{sys}}} \quad \text{公式 (4)}$$

### 7.3 测试环境

本次测试的地点是坐落在浙江省台州市开发区的中科院计算所台州分所, 测试的软件环境为: 蓝鲸分布式文件系统优化版 (Build 65), 蓝鲸服务动态部署系统 3.0; 应用服务器的操作系统是 Red Hat Linux 9.0, 系统服务器 (元数据服务器、绑定服务器、管理服务器等相关模块全部安装在同一个节点上, 称为系统服务器) 和存储节点的操作系统是 Red Hat Linux 8.0。硬件环境如下: 所有节点均采用 Intel Xeon 2.40GHz CPU, Intel 82545EM 千兆以太网控制器, 应用服务器和系统服务器配置 1024M 字节内存, 存储节点配置 2048M 字节内存; 每一个存储节点都配置 3ware 9500 SATA 磁盘阵列控制器, 12 块 160G 的 Seagate SATA 硬盘配置成 RAID10; 所有节点通过一个 NETGEAR JGS524 千兆交换机互连。在下面的所有测试中, 蓝鲸分布式文件系统配置了一个系统服务器: NFS 服务器是某个存储节点, 宿主文件系统是 EXT3, 所有软件均采用操作系统的缺省设置。为了避免缓存的影响, 每一次测试时都开始于全新的环境, 系统中不存在数据和元数据的缓存信息。元数据服务器进行文件数据块预分配的大小是 1M 字节, 应用服务器每次申请块映射的数量也是 1M 字节, 也即 256 块, 分片的大小是 4M 字节。。

我们首先测试了系统的一些基本参数: netperf[68]测得的应用服务器与存储节点之间的网络速度为  $910 \times 10^6 \text{bps}$ , 存储节点的磁盘阵列的读写速度分别是 121MB/s 和 133MB/s。

### 7.4 测试用例及工具

对分布式文件系统进行性能测试的方法、工具、用例不计其数, 而且用户实际使用的硬件环境、软件环境、数据集等都是千差万别, 我们无法针

对每一种情况逐个测试对比。因此我们在现有的硬件环境中,选取了常见的几种使用环境,分别设计了测试用例。希望利用这些有针对性的测试用例的结果验证系统模型,指导系统的再设计与开发,并对用户的实际使用提供有价值的参考信息。

**大文件的读写。**大文件的读写主要是模拟信息采集、数据分析等环境。在这些应用环境中,每一个应用服务器在使用蓝鲸分布式文件系统时,主要涉及的是大文件的读写。这些大文件,从几十 M 字节到数百 G 字节,甚至更大。我们称这类应用是“数据密集型”应用。这些应用主要使用蓝鲸分布式文件系统的海量数据存取能力,涉及的文件数量较少。在运行时刻,应用服务器与元数据服务器之间的信息交换较少,整个系统的负载主要是应用服务器与存储节点之间的数据传输。我们使用 Linux 的 dd 命令,在每个应用服务器上连续读或者写 20 个不同的大文件,每个文件 1G 字节,记录每次测试的耗时和 CPU 的使用率。为了获得比较准确的测试结果,每次测试我们都会重复多次,然后计算平均值。将此测试用例在 NFS 环境中运行,记录结果。所有上述测试均为多台应用服务器并发运行。

**小文件写入与删除操作。**在一些信息采集与检索环境中,大量信息都是以几 K 字节长的小文件保存,系统正常运行时刻的数据量有上千万个这样的小文件。在应用程序检索过滤完毕以后,原始信息以小文件的形式被保存;一段时间以后这些文件又被删除,以便存放新的信息,以此往复。为了模拟这种小文件操作环境,我们设计了下面的测试用例:在多台应用服务器上,将 Red Hat Linux 8.0 自带的内核源代码包 linux-2.4.18-14.tar.gz 拷贝到蓝鲸分布式文件系统加载的目录,然后多次解压缩到不同的目录;卸载蓝鲸分布式文件系统,消除缓存影响;再将这些文件全部删除。每一个内核源代码包含有 8010 左右个的子目录和文件。记录每次解压缩消耗的时间和 CPU 使用率,以及每次删除消耗的时间和 CPU 使用率。将此测试用例在 NFS 环境中运行,记录结果。所有上述测试均为多台应用服务器并发运行。

**编译 Linux 内核。**Andrew 基准程序 (Andrew benchmark) [69][70]主要模拟软件开发流程中文件系统相关的负载。它基本上是一个消耗 CPU 的操作集合,经常用来在应用程序一级比较文件系统的性能。我们在测试中模拟 Andrew 基准程序部分功能,在多个应用服务器上同时独立地编译 Red Hat Linux 操作系统自带的内核 Linux 2.4.18-14 的情况,观察蓝鲸分布式文件系统大规模软件开发环境下的性能以及可扩展性。具体过程是在各个应用服务器上首先将内核源代码包 linux-2.4.18-14.tar.gz 拷贝到不同的目录,解包,编译;执行的命令次序是:make dep;make bzImage;make modules。

**同一目录下空文件/空目录的删除与创建。**因为蓝鲸分布式文件系统的采用了 ext2/3 组织目录的方式[71],为了测试这种方式处理大目录的性能,设

计了此测试用例。我们在单个目录下分别创建空文件/目录若干，然后删除它们，记录操作消耗的时间。

## 7.5 测试结果与分析

### 7.5.1 大文件读写

我们首先进行了蓝鲸分布式文件系统分别在配置 1 个、2 个、4 个存储节点情况下，1 个、2 个、4 个、8 个、16 个应用服务器并发写入大文件的测试，测试结果如表 7.1，各个应用服务器的 CPU 利用率如表 7.2 所示，配置不同存储节点情况下的加速比如表 7.3 所示。我们将大文件写的聚集带宽绘制成图，如图 7.1 所示。从图表数据中我们可以得出如下一些信息：

- 在所有的配置情况下，蓝鲸分布式文件系统的大文件聚集写性能都优于 NFS，特别是在客户端数量较多的情况下尤其明显。
- 在配置 1 个存储节点的情况下，随着客户端的增加，聚集写性能有所下降。排除误差影响之外，主要是因为多个应用服务器的同时写操造成存储节点随机访问磁盘，增加了磁盘的寻道时间，系统性能下降。
- 在配置 2 个存储节点和 4 个存储节点的情况下，聚集写性能有了很大幅度提高，最高达到 4.7 的加速比，这主要得益于多个存储节点同时提供数据传输服务，文件系统在各个资源组之间的分片存储发挥了巨大作用。
- 在配置 2 个存储节点的情况下，在 4 个应用服务器的时候，聚集写性能已经达到了最高值，之后基本保持这样的水准。这主要是因为存储节点已经出于饱和状态，即使添加更多的应用服务器，也难以提供更高的数据传输率。
- CPU 利用率越高，单个应用服务器获得的带宽越大。这主要是因为 BWFS 和 NFS 都是基于以太网传输数据的，网络协议开销较大。

之后我们同样进行了蓝鲸分布式文件系统并发读取大文件的测试，测试结果如表 7.4，各个应用服务器的 CPU 利用率如表 7.5 所示，配置不同存储节点情况下的加速比如表 7.6 所示。我们将大文件读的聚集带宽绘制成图，如图 7.2 所示。从图表数据中我们同样可以得出如下一些信息：

- 在误差允许的范围之内，蓝鲸分布式文件系统的大文件聚集读性能优于 NFS，特别是在客户端数量较多的情况下尤其明显。
- 在配置 2 个存储节点和 4 个存储节点的情况下，聚集读性能有了很大幅度提高，最高达到 2.87 的加速比，这主要得益于分片存储使得

多个存储节点并发提供数据传输服务。

- CPU 利用率越高, 单个应用服务器获得的带宽越大。这主要是因为 BWFS 和 NFS 都是基于以太网传输数据的, 网络协议开销较大。

从以上测试中我们可以看到, 蓝鲸分布式文件系统采用的带外数据传输技术使其具有比 NFS 好很多的大文件读写性能, 特别是在客户端数量较多的情况下。蓝鲸分布式文件系统在不同配置, 特别是多个存储节点、多个客户端的配置下, 具有较好的加速比, 说明分片存储技术能够充分利用多个存储节点的并发数据传输能力, 使得 BWFS 具有较好的可扩展性。

表 7.1 BWFS、NFS 大文件写的聚集带宽 (单位: MB/s)

客户端数量 系统配置	1	2	4	8	16
1 个 SN	89.82	89.04	81.10	77.28	69.72
2 个 SN	88.28	125.64	154.57	151.70	148.95
4 个 SN	85.33	142.22	222.61	295.74	330.99
NFS v3	55.35	49.35	45.01	52.85	64.25

表 7.2 BWFS、NFS 大文件写的 CPU 利用率

客户端数量 系统配置	1	2	4	8	16
1 个 SN	22-24%	10-11%	4-5%	1-2%	0-1%
2 个 SN	20-23%	13-14%	8-9%	3-5%	2-3%
4 个 SN	19-20%	14-15%	10-11%	6-7%	3-4%
NFS v3	13-14%	4-5%	1-2%	0-1%	0-0%

表 7.3 BWFS 聚集写性能相对存储节点数量的加速比

客户端数量 系统配置	1	2	4	8	16
1 个 SN	1.000000	1.000000	1.000000	1.000000	1.000000
2 个 SN	0.982855	1.411051	1.905919	1.962992	2.136403
4 个 SN	0.950011	1.597260	2.744883	3.826863	4.747418

表 7.4 BWFS、NFS 大文件读的聚集带宽 (单位: MB/s)

客户端数量 系统配置	1	2	4	8	16
1 个 SN	74.74	73.14	77.28	88.09	100.51
2 个 SN	73.14	93.09	124.12	143.367	167.18
4 个 SN	74.85	113.46	169.17	232.66	289.13
NFS v3	78.77	58.51	49.05	27.77	23.41

表 7.5 BWFS、NFS 大文件读的 CPU 利用率

客户端数量 系统配置	1	2	4	8	16
1 个 SN	17-18%	7-8%	3-4%	2-3%	1-1%
2 个 SN	17-18%	10-12%	6-7%	3-4%	1-2%
4 个 SN	16-18%	11-12%	8-9%	5-6%	2-3%
NFS v3	16-18%	5-6%	2-2%	0-0%	0-0%

表 7.6 BWFS 聚集读性能相对存储节点数量的加速比

客户端数量 系统配置	1	2	4	8	16
1 个 SN	1.000000	1.000000	1.000000	1.000000	1.000000
2 个 SN	0.978592	1.272765	1.606108	1.627506	1.663317
4 个 SN	1.001472	1.551272	2.189053	2.641162	2.876629

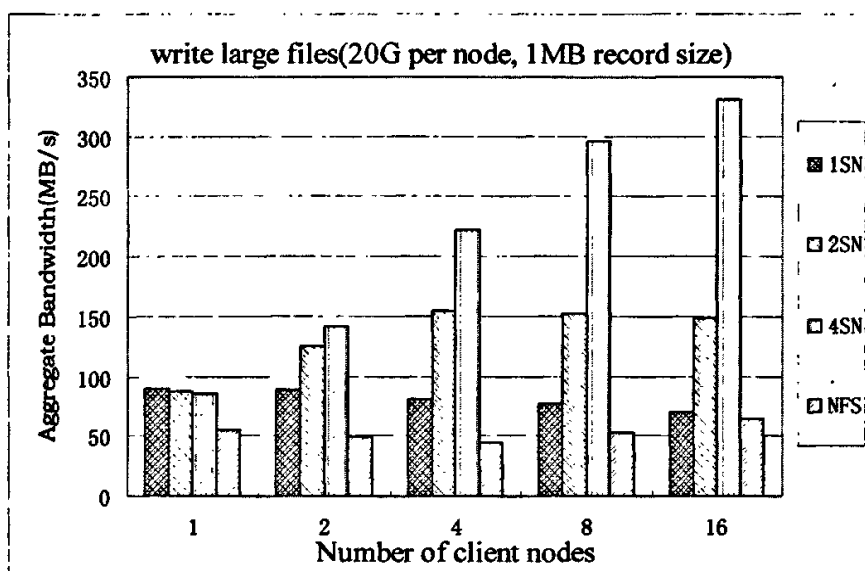


图 7.1 BWFS、NFS 大文件写的聚集带宽 (单位: MB/s)

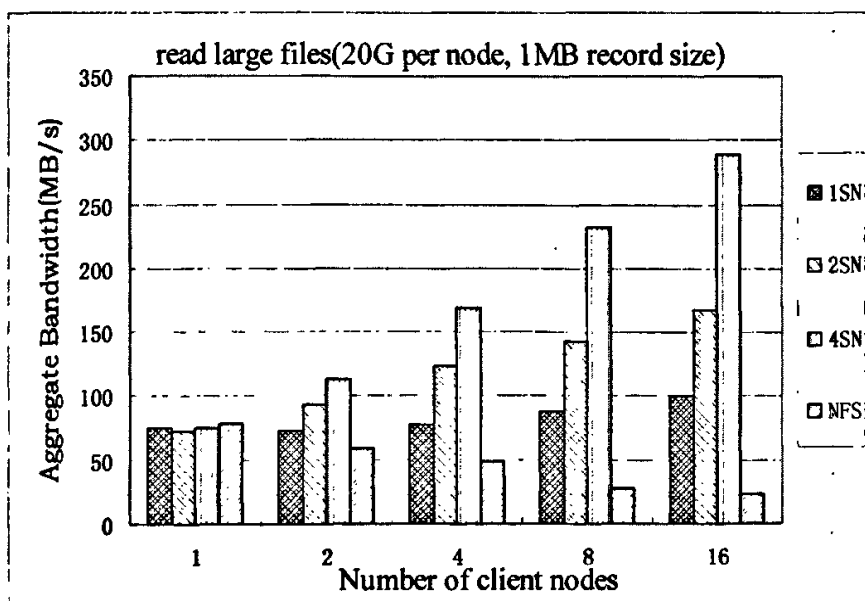


图 7.2 BWFS、NFS 大文件读的聚集带宽 (单位: MB/s)

## 7.5.2 小文件写入与删除

我们测试了蓝鲸分布式文件系统在配置一个元数据服务器, 分别配置 1 个、2 个存储节点情况下解包内核源代码和删除已经解开的内核源代码的性能。这里只测试了配置一个元数据服务器的情况, 主要是因为蓝鲸分布式文件系统的多个元数据服务器集群的设计与另有其他研究人员[72]主持, 并且现在仍然处在研究开发阶段。测试结果如表 7.7 和表 7.8 所示, 聚集吞吐率分别如表 7.9 和表 7.10 所示。

表 7.7 解开内核源代码包消耗的时间 (单位: 秒)

客户端数量 系统配置	1	2	4	8	16
1 个 SN	50.10	53.59	70.10	134.16	280.71
2 个 SN	47.89	52.51	69.33	123.44	283.66
NFS	43.08	56.19	78.49	125.71	231.64

表 7.8 删除内核源代码目录树消耗的时间 (单位: 秒)

客户端数量 系统配置	1	2	4	8	16
1 个 SN	32.93	37.43	450.99	1018.74	2023.41
2 个 SN	43.22	61.13	314.01	968.45	1837.25
NFS	37.64	38.56	41.21	59.45	96.75

表 7.9 解开内核源代码包的聚集吞吐率 (单位: 文件/秒)

客户端数量 系统配置	1	2	4	8	16
1 个 SN	159.87	298.93	457.04	477.63	456.55
2 个 SN	167.27	305.07	462.12	519.10	451.80
NFS	185.92	285.10	408.20	509.73	553.27

表 7.10 删除内核源代码目录树的聚集吞吐率 (单位: 文件/秒)

客户端数量 系统配置	1	2	4	8	16
1 个 SN	243.24	428.00	71.04	62.90	63.34
2 个 SN	185.35	262.06	102.04	66.17	69.76
NFS	212.83	415.47	777.56	1077.93	1324.66

将创建文件/目录与删除文件/目录的聚集吞吐率用图形表示见图 7.3、图 7.4。从中我们可以看出, 蓝鲸分布式文件系统创建文件/目录的吞吐率随着应用服务器的增加有明显增加, 从 4 个服务器开始趋于平缓, 在 8 个应用服务器时达到最大, 说明在此种类型的应用中, 蓝鲸分布式文件系统的元数据服务器的处理能力在配置 8 个应用服务器时已经基本达到了极限。配置 2 个存储节点的 BWFS 比配置 1 个存储节点的 BWFS 性能稍高, 但优势不明显,

主要是因为小文件/目录的创建主要消耗元数据服务器的能力,对数据传输的要求相对小很多的缘故。BWFS 与 NFS 创建文件的性能相当。

BWFS 删除文件/目录的聚集性能在 2 个应用服务器时达到最高,之后迅速下降。我们判断这主要是因为删除文件/目录操作时元数据服务器上大量的同步操作造成的。NFS 在删除文件的性能方面明显优于蓝鲸分布式文件系统的性能,说明我们在这方面还有很多优化工作需要做,特别是如何将同步操作转化成异步操作等。在实际使用环境中,文件删除操作在所有文件操作中的比例很低,特别是集中删除大量文件的操作比例更低。因此蓝鲸分布式文件系统在这方面的性能弱势并不影响它的整体性能发挥,对现实应用程序的正常运行不会造成明显影响。

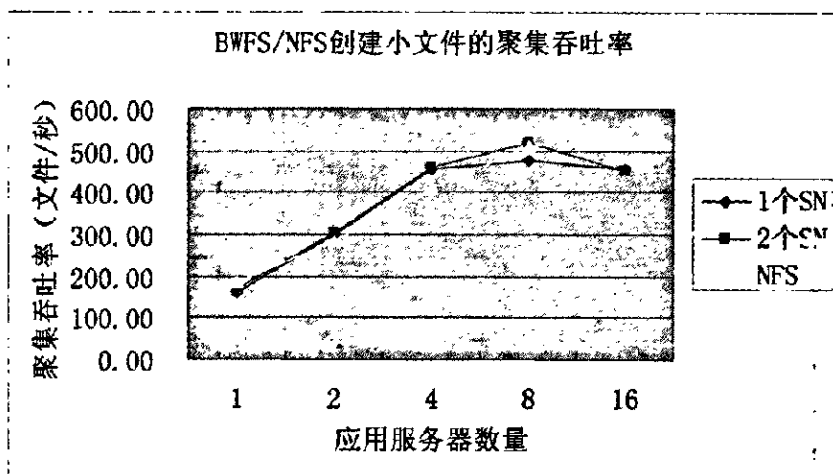


图 7.3 BWFS 创建小文件的聚集吞吐率

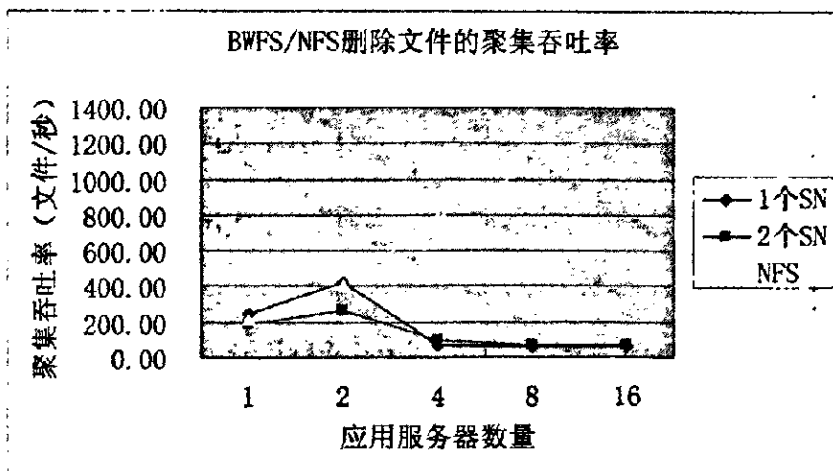


图 7.4 BWFS 删除文件/目录的聚集吞吐率

### 7.5.3 编译 Linux 内核

我们测试了在多台应用服务器上，使用配置 2 个存储节点的蓝鲸分布式文件系统的环境，并发地、独立地编译 Linux 内核的情景，测试结果如表 7.11 所示。

表 7.11 编译内核源代码的耗时（单位：秒）

客户端数量	1	2	4	8	16
耗时 (Time)	5982.90	6100.36	7191.10	N/A(注)	8512.46
加速比 (speedup)	1.00	1.96	3.33	N/A(注)	11.25

注：N/A 表示测试没有顺利完成。

从测试结果我们可以看到，随着应用节点的增加，编译内核消耗的时间也有所增长，这主要是多个节点并发访问蓝鲸分布式文件系统，造成单个节点的吞吐率有所下降，使得编译程序不能充分利用本地 CPU 进行编译计算。但是如果如下定义整个系统进行内核编译的加速比 speedup：

$$\text{speedup} = \frac{N \text{ 个节点单位时间内完成编译总数}}{1 \text{ 个节点单位时间内完成编译总数}} \quad \text{公式 (5)}$$

如表 7.11 中第三行所示，那么我们可以看到虽然系统整体性没有取得线性提高，但是加速比也达到了不错的水平，说明蓝鲸分布式文件系统在处理类似编译内核的工作时具有较好的可扩展性。

### 7.5.4 同一目录下空文件/空目录的删除与创建

我们测试了蓝鲸分布式文件系统在配置 2 个存储节点、单个元数据服务器、单个客户端的情况下，在同一目录种创建空文件/删除空文件的性能，其结果如图 7.5 所示。图中横轴表示目录中已经含有的文件总数，纵轴表示创建/删除 1000 个空文件所消耗的时间。我们总共在同一目录下创建了 50 万个文件，然后删除它们，测得每创建/删除 1000 个文件消耗的时间。从图中我们可以看到，随着目录中含有文件数量的增加，每创建 1000 个文件所需要的时间也明显增加，这主要是 ext2/3 随着目录变大，处理目录结构的开销也增加了：创建从 499000 到 500000 个文件消耗的时间是创建从 1 到 1000 个文件时间的 11—12 倍！删除文件的操作在目录较大时，也会消耗更多时间：删除从 499000 到 500000 个文件消耗的时间是删除从 1 到 1000 个文件时间的 4 倍左右。蓝鲸分布式文件系统继承了 ext2/3 的未排序列表方式组织目录结构，因此在处理含有大量文件的大目录应用时，表现出比较低的目录处理性能，当然也会影响到其它操作的性能。

我在这里没有给出在同一目录下创建/删除子目录的性能数据，主要是因为 2.4 系列内核的 Linux 中，同一目录下最多只能含有 65536 左右个子目



录, 而 ext2/3 只能含有 32000 个左右子目录, 性能变化及其微小。

ReiserFS、XFS 等文件系统由于采用了 B+ 树结构保存目录文件, 性能有了飞跃式提高, DCFS 在这方面也有相当多的优化[73]。BWFS 在下一步的优化中将对这一部分重点关注。

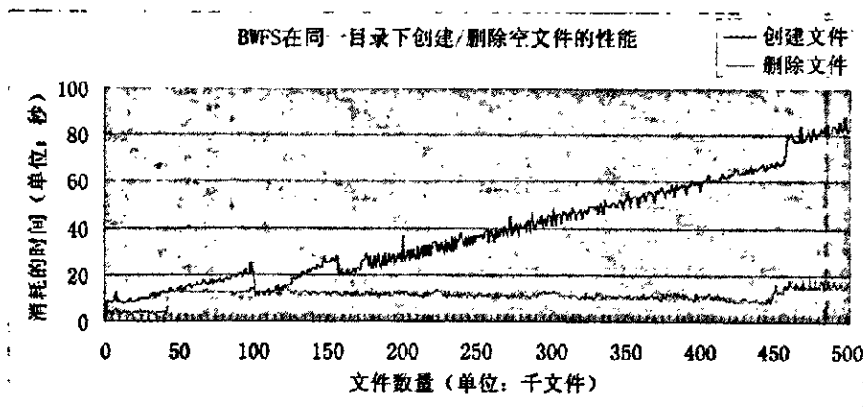


图 7.5 BWFS 在同一目录下创建/删除空文件的性能

## 7.6 实际使用案例分析

2004 年底, 我们在某公司进行了真实环境下的石油天然气应用的测试, 主要是针对蓝鲸分布式文件系统与传统的 NFS 环境以及专用的 NAS 系统进行了性能对比测试。由于测试环境和测试结果涉及商业机密, 本文中不描述测试的具体细节, 只提供相关的粗略数据, 以及最后的测试结论。如果有论证需要, 请联系本文作者。

测试环境中配置如下: (1) 采用 Cisco6509 作为核心交换机。(2) 计算节点总共 128 台, 统一采用某公司的刀片服务器, 配置双 Xeon 2.8 GHz CPU, 2GB 内存, 单个千兆网卡, 其中每 14 台刀片为一组, 利用 4 跟千兆以太网绑定后连接在 Cisco 交换机上, 全部安装标准 Linux RedHat 9.0 (kernel-2.4.20-8)。(3) 某专用 NAS 服务器配置如下: 单 Pentium 4 2.8GHz CPU, 2GB 内存, 3Ware 3w-9500-12-S RAID 控制器, 9 块 250GB(7200rpm) SATA 硬盘, RAID5 (8+1), 双千兆网卡绑定后连接到核心交换机, 采用 Linux 操作系统, 内核为 2.6.8.1smp。(4) 某 NFS 服务器配置如下: 双 Xeon 2.8G CPU, 打开超线程 (Hyper-Threading) 开关, 2GB 内存, 两块 73GB SCSI 的系统硬盘, 安装标准 Linux Red Hat 9.0, 通过双千兆网卡连接到核心交换机, 同时通过 1 条 SCSI 320 线连接某公司的磁盘阵列 (配置 Intel 80303 CPU, 128MB RAM, 14 块 146GB Ultra 320 SCSI 磁盘, RAID5 (13+1))。(5) 蓝鲸分布式文件系统配置的存储节点如下: 单 Xeon 2.8G CPU, 3GB 内存, 双千兆网卡绑定连接到核心交换机, 10 块 160G (7200rpm) SATA, RAID5 (9

+1)。元数据服务器配置如下：单 Xeon 2.4G CPU，1GB 内存，单千兆网卡连接到核心交换机。

测试中使用的应用软件为 Paradigm EPOS 3.0 update1，数据采用某三维工区 191.6GB 数据量，15971040 道，2 毫秒采样，6 秒记录，测试作业选择三维叠前时间偏移，输出道集 19.6G，2GB 偏移数据体。作业参数的选择近似于日常生产中的同类作业的参数。测试的结果如图 7.6 所示，横轴表示计算节点的数量，纵轴表示时间；时间越少，性能越好。

从图中我们可以看到，BWFS 相比其它专用 NAS 和 NFS 服务器，有较好的性能，同时具有很好的可扩展性。随着计算节点的增加，BWFS 完成一个作业所需要的时间在不断减少，呈现很好的扩展能力；而某专用 NAS 和某 NFS 服务器却在 128 个节点时，比 96 个节点消耗更多的时间，说明存储子系统已经成为影响应用程序发挥性能的瓶颈。BWFS 确实随着节点增加，完成作业所需要的时间一直在减少，在测试环境所达到的 128 个节点时，也能表现是非常好的性能。

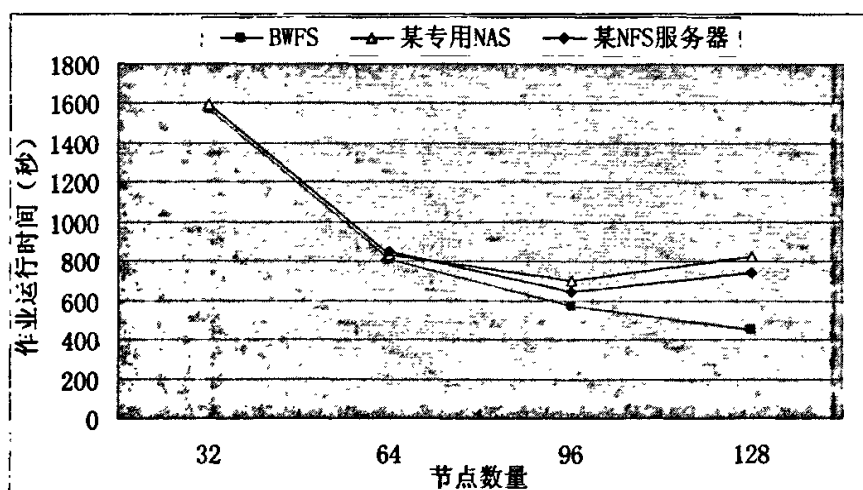


图 7.6 BWFS 与某专用 NAS 以及 NFS 服务器性能对比

## 7.7 小结

本章结合实际应用环境，从典型应用程序出发，列举了蓝鲸分布式文件系统比较关心的性能指标，给出了性能评价模型。然后有针对性地设计了许多测试用例，分别测试了多种情况下蓝鲸分布式文件系统的性能，并对测试结果进行了较为详尽的分析。蓝鲸分布式文件系统在大文件读写方面具有很好的性能和可扩展性，相比 NFS 有很大的提高。在小文件处理方面，创建小文件的性能与 NFS 相当，但是删除文件的性能比 NFS 相差较多。不过由于

在实际使用环境中删除文件的操作毕竟占有相当少数，并不影响蓝鲸分布式文件系统的整体使用性能。蓝鲸分布式文件系统在处理大目录的性能上还比较弱，在以后的研究中亟待加强。我们使用商业上常见的石油天然气勘探软件，在典型环境下，针对实际数据，对比测试了蓝鲸分布式文件系统与典型NAS、NFS的性能，突显了蓝鲸分布式文件系统在支持大量节点并发访问下的优势。蓝鲸分布式文件系统之所以能够取得较好的性能和扩展性，源于它的分布式分层资源管理模型，以及此模型中使用的将元数据和数据分离的带外数据传输技术，分片存储到多存储节点的技术等。

另一方面，我们还有更多针对蓝鲸分布式文件系统的测试与分析工作需要去做，特别是细粒度的跟踪测试和对比工作。这些跟踪测试和对比工作可能发现蓝鲸分布式文件系统中更多的设计缺陷、实现缺陷以及性能瓶颈等，对蓝鲸分布式文件系统未来的研究与发展有重要意义。

## 第八章 结束语

蓝鲸分布式文件系统是蓝鲸大规模网络存储系统的核心部件,它管理整个系统的存储空间,向用户提供全局统一的、共享的、单一映象的文件系统访问接口。本文较深入地讨论了蓝鲸分布式文件系统资源管理情况,阐述了资源管理的模型、实现技术、性能测试与分析等。上述研究成果对于蓝鲸大规模网络存储系统的开发和推广有重要意义,同时对国内外分布式文件系统研究人员也有一定的借鉴意义和参考价值。

本章首先介绍蓝鲸分布式文件系统的研究现状,之后总结了本文所做贡献,然后对蓝鲸分布式文件系统未来的研究与发展提出一些自己的想法,最后是自己在项目研究中的一些体会与感想。

### 8.1 研究现状

蓝鲸大规模网络存储系统是国家“八六三”计划支持的重点研究项目,到目前为止已经取得了诸多研究成果,某些方面在国内国际都处于领先地位。蓝鲸分布式文件系统作为蓝鲸大规模网络存储系统的核心部件,管理海量存储空间,提供统一的、共享的、单一映象的分布式文件系统接口,支持大量客户的并发访问,具有很好的性能和可扩展性。目前蓝鲸分布式文件系统兼容 NFS 的文件共享语义,实现应用程序的二进制兼容,可以在多个存储节点之间分片存储,实现动态添加各种服务器,能够有效实现负载平衡、在线迁移、快速恢复等功能。

蓝鲸大规模网络存储系统不仅在科研上取得了成果,还在科研成果产品化方面取得了良好效益。目前蓝鲸大规模网络存储系统已经在多家企业成功应用,取得了良好的社会效益和经济效益,为科研成果产品化探索了新的思路 and 方向。

### 8.2 本文工作总结

本论文主要研究了蓝鲸分布式文件系统的资源管理模型、设计与实现的优化、性能测试与对比分析等。对分布式文件系统来说,最主要的资源就是存储资源。本文围绕存储资源展开,研究了存储资源的管理、分配、使用、负载平衡等技术。通过这些研究,取得了如下成果:

- **BWFS 的分布式分层资源管理模型 (DLRM)。**DLRM 模型是针对大规模海量存储系统设计的,充分考虑了分布式机群环境的特点,

简化分布式文件系统的设计与实现,努力提高系统的性能和可扩展性。该模型中包含了众多的特性,包括带外数据传输、独立的模块分布、虚拟共享磁盘结构、并发资源组管理、全动态的元数据分布等,这些特性对蓝鲸分布式文件的性能和可扩展性起关键作用。

- 高效的物理存储空间管理。蓝鲸分布式文件系统将整个系统的共享存储空间划分成多个资源组,各自独立地管理存储空间的使用情况,跟踪数据块以及索引节点的映射等。资源组管理器的数据块和索引节点动态分配、多级带统计信息的指针模型等技术使得蓝鲸分布式文件系统可以管理大规模的存储系统,动态添加存储资源,提供高性能的资源管理,实现负载平衡,进行数据迁移等。
- 全动态的元数据绑定。绑定服务器平衡协调各个元数据服务器的负载,凭借全动态绑定的元数据服务器机群技术,使得整个分布式文件系统具有最佳性能。
- 多种文件系统资源管理优化。蓝鲸分布式文件系统的元数据服务器进行有策略的资源分配,实现批量资源申请/异步释放,在多个资源组之间实现分片存储,实现分布式的文件系统日志,利用多个元数据服务器协同工作形成机群处理,有效地实现块映射信息的缓存。这些特性使得蓝鲸分布式文件系统可以充分利用多个存储节点提供的数据传输带宽,充分利用多个元数据服务器的计算能力,同时又能很好地维护文件系统的一致性,显著降低文件系统的故障恢复时间等。
- 较好的性能和可扩展性。通过大量测试、对比、分析我们看出,蓝鲸分布式文件系统具有较好的性能和可扩展性,特别是大文件的并发读写,比传统的 NFS 有更好的性能和扩展性,读性能最多提高了 10 倍,写性能提高了 5 倍左右,有效支持的并发客户端数量也至少翻了一番。这也验证了蓝鲸分布式文件系统的分布式分层资源管理模型的有效性和先进性。

同时,在本论文的研究中,特别是通过大量测试也发现蓝鲸分布式文件系统的存在的一些不足:

- 文件的删除处理。由于蓝鲸分布式文件系统的元数据服务器在删除文件的时候需要将索引节点同步到磁盘,使得其性能和 NFS 相比有很大差距。
- 目录组织效率低下。蓝鲸分布式文件系统继承了 ext2/3 的目录组织方式,在目录文件较大的情况下(超过 5 万个目录项以后),处理效率显著降低。

### 8.3 下一步研究方向

蓝鲸分布式文件系统是为大规模海量存储系统设计的, 应用在中高端科学计算、信息处理等环境中, 使得其面临更多的挑战、更高的要求。我个人觉得蓝鲸分布式文件系统未来在资源管理领域还有如下一些方面需要进一步研究:

- 改善文件/目录删除操作的性能。当前的蓝鲸分布式文件系统在删除文件时, 需要将此被删除文件的索引同步到磁盘, 引起删除文件操作的性能低下。下一步亟待解决的就是如何将同步操作转化为异步操作, 提高文件删除操作的性能。
- 大目录的有效支持。从前一章的测试结果可以看到蓝鲸分布式文件系统处理大目录的性能非常低下, 改善目录项的组织结构, 提高大目录的处理能力非常有必要。
- 元数据一致性。多个元数据服务器协同工作, 在元数据发生转移的情况下如何保证系统的一致性, 如何在出现节点故障的情况下尽快恢复系统一致性。
- 数据一致性。有些应用需要比 NFS 更高要求的文件共享语义, 如何在保证性能的基础上, 实现更好的数据一致性是下一节点的研究内容之一。
- 提高系统的可用性。做为管理海量存储系统的机群文件系统, 蓝鲸分布式文件系统的高可用性一直是弱项。下一阶段需要着重加强提高可用性的研究, 包括实现利用失效接替 (fail-over) 提高关键服务的正常运行; 利用各个存储节点间的数据校验, 实现类似 RAID 的功能, 提高数据可用性等; 开发类似 fsck 的工具, 恢复文件系统的一致性。
- 进一步更加深入、细致、周全的性能测试。为了能够将系统中存在的软件设计缺陷、性能性能瓶颈、应用运行状况等信息反应出来, 我们在下一阶段需要进行更加深入、细致、周全的性能测试, 包括宏观测试、微观测试。宏观测试是指使用一定的应用软件, 通过使用蓝鲸分布式文件系统, 测试其性能状况。微观测试是利用安装在系统中的各个“探针”, 分析系统在运行过程中各个模块的行为和性能, 为进行局部优化提供依据。

当前的蓝鲸分布式文件系统已经取得了阶段性成果, 为以后进一步研究与开发提供了稳固的平台和实践基础。下一步我们需要继续深入研究各种文件系统、分布式文件系统、大规模分布式文件系统的先进技术, 提高系统的性能、可扩展性, 实现系统的高可用性等。同时作为一个产品, 还有很多实

用化的工作需要开展。

#### 8.4 体会与感想

蓝鲸分布式文件系统是一个高度复杂的大型系统，每一个研究人员都倾注了大量的时间和精力，多年矢志不渝的勤奋努力才有了今天的成果。在蓝鲸分布式文件系统的开发过程中，我们密切注意国际国内最新的研究成果，参考大量的国内外文献，紧密结合国内实际情况，始终保持研究的前沿性、实用性。在开发过程中，我们也充分利用最有力的开发工具，加强设计开发的质量管理，保证研究与开发有效性。

每一项研究工作，特别是涉及面大、复杂度高的项目，都是多人合作，经过多年努力才能完成的。在此过程中，非凡的毅力与恒心，紧密的合作与交流，正确的理论与方法都是成果的关键因素。经过这三年多的参与和锻炼，我已经基本掌握了进行科研工作的理论和方法，懂得如何去分析问题和解决问题，如何进行团队合作和交流，培养了我持之以恒的作风。这三年多的研究工作将使我受益终生！

## 参考文献

- 
- [1] <http://www.top500.org>, 2004.
  - [2] <http://www.seagate.com/products/discfamily/cheetah/index.html>, 2004.
  - [3] David Patterson, Garth Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks(RAID). In ACM SIGMOD Conference, pages 109-116, Jun 1988.
  - [4] 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 April 1989.
  - [5] IEEE P802.3z Gigabit Task Force,  
<http://grouper.ieee.org/groups/802/3/z/index.html>, 2004.
  - [6] IEEE P802.3ae 10Gb/s Ethernet Task Force,  
<http://grouper.ieee.org/groups/802/3/ae/>, 2004.
  - [7] The Fibre Channel Industry Association, <http://www.fibrechannel.org/>, 2004 .
  - [8] Mazin Yousif. InfiniBand 1.0 Architecture Overview, February 27, 2001. Intel Developer Forum, Spring 2001.
  - [9] Andrew Lockey, Storage over the InfiniBand Fabric, March 1, 2001. Intel Developer Forum, Spring 2001.
  - [10] Dave Hitz, A Storage Networking Appliance, TR3001. Network Appliance, Inc. 10/2000.
  - [11] Marc Farley, Building Storage Network, McGraw-Hill, 2000.
  - [12] R. Sandberg, Sun Network Filesystem Protocol Specification, Technical Report, Sun Microsystems, Inc., 1985.
  - [13] S. Shepler, B. Callaghan. Network File System (NFS) version 4 Protocol, The Internet Engineering Task Force, <http://www.ietf.org/rfc/rfc3530.txt>, April 2003.
  - [14] Paul Leach and Dan Perry, CIFS: A Common Internet File System, Microsoft Interactive Developer, November 1996.
  - [15] Jon Tate, Angelo Bernasconi, Peter Mescher, Fred Scholten, Introduction to Storage Area Networks, <http://www.ibm.com/redbooks/>, 2004.
  - [16] J. Satran, K. Meth, Internet Small Computer Systems Interface (iSCSI), RFC 3720 , The Internet Engineering Task Force,  
<http://www.ietf.org/rfc/rfc3720.txt>, April 2004.
  - [17] Charles Monia, Rod Mullendore, iFCP - A Protocol for Internet Fibre Channel Networking,  
<http://www.ietf.org/internet-drafts/draft-ietf-ips-ifcp-14.txt>, December 2002.



- [18] M. Rajagopal, E. Rodriguez, Fibre Channel Over TCP/IP (FCIP), RFC 3821, The Internet Engineering Task Force, <http://www.ietf.org/rfc/rfc3821.txt>, July 2004.
- [19] T. Breuer, A. Marin Lopez and Arturo Gar. The Network Block Device. Linux Journal. Issue 73, Posted on Monday, May 01, 2000.
- [20] B. Callaghan, B. Pawlowski, P. Staubach, NFS Version 3 Protocol Specification, RFC 1813, The Internet Engineering Task Force, <http://www.ietf.org/rfc/rfc1813.txt>, June 1995.
- [21] Steve Soltis, Grant Erickson, Ken Preslan, Matthew O'Keefe, and Tom Ruwart, The Design and Performance of a Shared Disk File System for IRIX, Fifteenth IEEE Symposium on Mass Storage Systems, 1998.
- [22] Kenneth W. Preslan, Andrew P. Barry, etc. A 64-bit, Shared Disk File System for Linux, Storage Conference, 1999.
- [23] Frank Schmuck and Roger Haskin, GPFS: A Shared-Disk File System for Large Computing Clusters, Proceedings of the Conference on File and Storage Technologies (FAST'02). 28-30 January 2002, Monterey, CA, pp. 231-244. (USENIX, Berkeley, CA.)
- [24] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, B. Hillsberg, IBM Storage Tank-A heterogeneous scalable SAN file system, IBM SYSTEMS JOURNAL, VOL 42, NO 2, 2003.
- [25] Haskin, R.L. Tiger Shark-a scalable file system for multimedia. IBM Journal of Research and Development, vol.42, no.2, p. 185-97, March 1998.
- [26] Peter Braam, The Lustre Storage Architecture, <http://www.lustre.org/docs/lustre.pdf>, 2004.
- [27] Garth A. Gibson, David F. Nagle, etc., NASD Scalable Storage Systems, Proceedings of USENIX 1999, Linux Workshop, Monterey, CA, June 9 - 11, 1999.
- [28] Garth Gibson, Object Storage Architecture, PANASAS White Paper, <http://www.panasas.com>, 2003.
- [29] Laura Shepard and Eric Eppe, SGI® InfiniteStorage Shared Filesystem CXFS™: A High-Performance, Multi-OS Filesystem from SGI, White Paper, Silicon Graphics, Inc. <http://www.sgi.com>, 2003.
- [30] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck, Scalability in the XFS File System. Proceedings of the USENIX, 1996.
- [31] 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.
- [32] Jin Xiong, Sining Wu, Dan Meng, Ninghui Sun, Guojie Li, Design and Performance of the Dawning Cluster File System, International Conference

- on Cluster Computing (Cluster 2003), Hong Kong, December 1-4, 2003.
- [33] Overview of the MPEG-4 Standard, INTERNATIONAL ORGANISATION FOR STANDARDISATION, ISO/IEC JTC1/SC29/WG11 N4668, March 2002.
- [34] “863”计划最新成果——曙光 4000A 通过鉴定验收,  
<http://www.cas.cn/html/Dir/2004/06/30/3316.htm>, 2004.
- [35] 刘振军、许鲁、尹洋, 蓝鲸 SonD 服务动态部署系统, 计算机学报, 2005 年 5 月。
- [36] 王敏, 一种虚拟化资源管理服务模型及其实现, 计算机学报, 第 28 卷, 第 5 期, 856—863 页, 2005 年 5 月。
- [37] 卫建军, 基于对象的多种资源管理系统, 计算机工程, 已录用。
- [38] 黄华, 张建刚, 许鲁, 蓝鲸分布式文件系统的分布式分层资源管理模型, 计算机研究与发展, 已录用。
- [39] 杨德志, 黄华, 张建刚, 许鲁, 大容量、高性能、高扩展能力的蓝鲸分布式文件系统, 计算机研究与发展, 已录用。
- [40] 黄华, 张建刚, 许鲁, 蓝鲸分布式文件系统的物理资源管理模型, 计算机工程, 已录用。
- [41] 黄华, 张敬亮, 张建刚, 许鲁, 蓝鲸分布式文件系统的客户端元数据缓存模型, 计算机科学, 已录用。
- [42] K. Salem and H. Garcia-Molina, "Disk striping," in Proceeding of the IEEE 2nd International Conference on Data Engineering (ICDE'86), Los Angeles, CA, Feb. 1986, pp. 336--342.
- [43] David J. Barnes, Object-Oriented Programming with Java: An Introduction, ISBN: 0-13-086900-7, Prentice-Hall.
- [44] Keith A. Smith and Margo I. Seltzer, File System Aging - Increasing the Relevance of File System Benchmarks, Measurement and Modeling of Computer Systems, 203-213, 1997.
- [45] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, R. Y. Wang. Serverless Network File Systems. Award Paper in Proc. Fifteenth Symposium on Operating Systems Principles. pp. 109-126. December 1995. Also appeared as University of California Technical Report CSD-98-983.
- [46] Chandramohan A. Thekkath, Timothy Mann, Edward K. Lee, Frangipani: A Scalable Distributed File System, Symposium on Operating Systems Principles (SOSP), 1997.
- [47] E. K. Lee, C. Thekkath, Petal: Distributed virtual disks. In Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), pages 84-92, October.

- 1996.
- [48] David A Rusling, The Linux Kernel,  
<http://www.tldp.org/LDP/tlk/tlk-title.html>, 2004.
- [49] Juan I. Santos Florido, Journal File Systems, Published in Issue 55 of Linux Gazette, <http://www.linuxgazette.com/issue55/florido.html>, July 2000.
- [50] Steve Best JFS Log:How the Journaled File System performs logging, Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, October 10-14, 2000, Atlanta, Georgia, USA.
- [51] Steve Best, Dave Kleikamp, JFS layout: How the Journaled File System handles the on-disk layout, Linux Technology Center, IBM, May 2000.  
<http://www-106.ibm.com/developerworks/library/l-jfslayout/>
- [52] Hans Reiser, The Reiser File System, <http://www.namesys.com/>
- [53] Jim Gray and Andreas Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, Inc., 1993.
- [54] George Samaras, Kathryn Britton, Andrew Citron, and C. Mohan. TwoPhase Commit Optimizations and Tradeoffs in the Commercial Environment. In Proceedings of the 9th IEEE International Conference on Data Engineering, April 1993.
- [55] Neil Brown, The Linux Virtual File-system Layer, 1999.  
<http://www.cse.unsw.edu.au/~neilb/oss/linux-commentary/vfs.html>
- [56] R. Srinivasan, RPC: Remote Procedure Call Protocol Specification Version 2, RFC 1831, The Internet Engineering Task Force,  
<http://www.ietf.org/rfc/rfc1831.txt>, 1995.
- [57] 田颖, 许鲁, 分布式文件系统中的负载平衡技术, 计算机工程, 第 29 卷, 第 19 期, 2003 年 11 月。
- [58] 田颖, 分布式文件系统中的负载平衡技术研究, 硕士学位论文, 中国科学院计算技术研究所, 2003 年。
- [59] 范勇, Lease 设计报告, 国家高性能计算机工程技术研究中心内部技术文档, 2004。
- [60] 贺劲, 机群文件系统性能与正确性研究, 博士学位论文, 中国科学院计算技术研究所, 2002 年。
- [61] 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, 中国科学院计算技术研究所, 2001 年。
- [62] 胡雨壮, 分布式文件系统吞吐率优化研究, 硕士学位论文, 中国科学院计算技术研究所, 2002 年。
- [63] 杜聪, 徐志伟, COSMOS 文件系统的性能分析, 计算机学报, 第 24 卷第 7 期, 2001 年 7 月。

- 
- [64] K K Ramakrishnan, A model of file server performance for a heterogeneous distributed system, Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols, p.338-347, August 05-07, 1986, Stowe, Vermont, United States.
- [65] Rajesh Bordawekar, Juan Miguel del Rosario, and Alok Choudhary. Design and evaluation of primitives for parallel i/o. In Proceedings of Supercomputing '93, pages 452-461, Portland, Oregon, November 1993.
- [66] Murthy V. Devarakonda, Ajay Mohindra, Jill Simoneaux, and William H. Tetzlaff. Evaluation of design alternatives for a cluster file system. In USENIX 1995.
- [67] Mary Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John Ousterhout. Measurements of a Distributed File System. In Proceedings of the Thirteenth Symposium on Operating Systems Principles, pages 198-211, October 1991.
- [68] netperf, <http://www.netperf.org/>, 2005.
- [69] J. Howard et al. Scale and performance in a distributed file system. ACM Transactions on Computer Systems, 6(1), 1988.
- [70] 杜聪, 机群文件系统的实现、评测与分析, 硕士学位论文, 中国科学院计算技术研究所, 2001 年。
- [71] Daniel Phillips, A Directory Index for Ext2, Proceedings of the 5th Annual Linux Showcase & Conference, 2001.
- [72] 杨德志, 元数据服务器集群设计报告, 国家高性能计算机工程技术研究中心内部技术文档, 2004。
- [73] Rongfeng Tang, Dan Meng, Sining Wu, Optimized Implementation of Extendible Hashing to Support Large File System Directory, IEEE International Conference on Cluster Computing (CLUSTER'03), 12 01 - 12, Hong Kong, 2003.

## 致 谢

一个不经意的选择，使我成为一名学习计算机专业的大学生。然后又是不经意的选择，使我走上了硕博连读的道路，从此在科研的道路上越走越远，越陷越深。幸好我对计算机产生了浓厚的兴趣，觉得它就是我最正确的选择，我会为之而疯狂。

科研工作是辛苦乏味的，但我在计算所遇到了我认为最好的两个导师：徐志伟和许鲁。徐老师循循善诱，谆谆教诲，为人师表，严厉又可亲。他让我懂得科研的方法和追求的目标，他用实际行动教书育人。他的渊博的知识，锐利的眼光，敏捷的思维，严谨的作风，突显个人魅力。许老师高屋建瓴，博大精深，身先士卒，威严又和蔼。他让我懂得科研需要恒心，团结就是力量，坚持才能胜利。他总是为科研倾注所有，以行动带领大家，用成就证明努力。得到这样两位导师的指导，三生有幸。感谢他们给予我的关心与爱护、鼓励与鞭策，这将使我受用终身；也祝愿他们以及他们各自的家人，身体健康，万事如意。

感谢张建刚、范忠磊、韩晓明三位副研究员，他们在过去三年中给了我巨大帮助和悉心指导，使我的科研工作得以顺利进行。他们忘我的工作作风、优异的师德品德，我将铭刻在心。

感谢硕士阶段课题组的李伟副研究员、李丙辰博士、冯百明博士、龚奕利博士、汤海鹰博士等，在硕士阶段的学习期间，从他们那里获得了很多帮助，学到了很多知识。

感谢国家高性能计算机工程技术研究中心的全体同事和同学，尤其是“蓝鲸”课题组的同事和同学们，和你们一起经历的科研，使得这篇论文成为可能。这些同事有秦平、田志勇、汤文辉、韩月、石红、黄朝晖、李静、肖展业、徐薇、刘珂、李亮等，这些同学有田颖、胡风华、杨德志、王敏、范勇、张敬亮、张军伟等，和你们一起共事是我的荣幸，共同为了“蓝鲸”奋斗的经历将是我一生的财富。在此特别感谢黄治，在工程中心的这几年，你给了我很多帮助。

感谢中国科学院研究生院的老师们，感谢计算技术研究所的老师们，你们给予了我许多科学知识和科研的本领。特别感谢计算所研究生部的几位老师，你们的辛勤工作才使得我在计算所生活学习的顺利顺利。

感谢所有关心我的同学和朋友们，我无法罗列全部的名字，却不会忘记你们的友情：刘鹏、杨文汇、高韬、张欣、单静、何雷、何丁山、付饶、陶然、张宇、张小华、王丹、汪冬、江四红、李耀伟、朱钱虎、王东、朱义锋、

陆锋、季建峰、姚东香、易旸、朱晨畅、张雁、程艳等。和你们在一起的日子，永远是最快乐的时光。

感谢我的大姑、二姑、叔叔、三姑和小姑，感谢我的大姨、舅舅和小姨，在我困难的时候，是你们帮助了我，你们的亲情是我永远的牵挂。感谢我的爷爷奶奶，你们将永远活在我美丽的童年里。感谢我的外公外婆，祝愿你们健康长寿。

感谢父母对我的生育养育之恩。父亲的坚强果断、勤劳勇敢、热情好善、孝敬爱心，将是我永远的精神财富，也是我做人的楷模；父亲勇于探索、不懈追求、尽善尽美的做事作风，我将永世难忘；深切缅怀我的父亲，热切告慰他的在天之灵。母亲的勤劳质朴、任劳任怨的精神，是我学习的榜样；多年无私无怨的支持，多年润物无声的教诲，多年殷殷切切的期盼，是我前进的动力和精神之来源。同样感谢我的继父，他的勤劳与无私，照顾与关怀，使我安心学习，顺利成长。

遇见你是一个奇迹，爱上你是一种默契——你就是我的妻子吴芹。由衷感谢你多年来的无怨无悔的支持，感谢你给予的源源不断的灵感，感谢你的悉心理解，感谢你的热情帮助。没有你，就没有本文，就没有我的今天。

## 作者简历

姓名：黄 华

性别：男

出生日期：1978.10.30

籍贯：江苏省张家港市

- 2000.9–2005.7      中国科学院计算技术研究所，  
计算机系统机构专业，  
硕博连读生。
- 1996.9–2000.7      北京大学，  
计算机科学技术系计算机软件专业，  
获理学学士学位。

### 【攻读博士学位期间发表的论文】

- [1] 黄华，张建刚，许鲁，蓝鲸分布式文件系统的分布式分层资源管理模型，计算机研究与发展(已录用)。
- [2] 杨德志，黄华，张建刚，许鲁，大容量、高性能、高扩展能力的蓝鲸分布式文件系统，计算机研究与发展(已录用)。
- [3] 黄华，张敬亮，张建刚，许鲁，蓝鲸分布式文件系统的物理资源管理模型，计算机工程（已录用）。
- [4] 黄华，张建刚，许鲁，将 NBD 移植到 Windows 平台，计算机工程（已录用）。
- [5] 黄华，张建刚，许鲁，蓝鲸分布式文件系统的客户端元数据缓存模型，计算机科学（已录用）。

### 【攻读博士学位期间参加的科研项目】

- [1] 国家“八六三”高技术研究发展计划基金项目（2002AA112010）
- [2] 中国科学院计算技术研究所创新项目“蓝鲸网络存储产品化”
- [3] 中国科学院百人计划项目“以网络存储为核心的服务系统”

### 【攻读博士学位期间的获奖情况】

- [1] 2003 年度所长奖学金优秀奖

作者：[黄华](#)  
学位授予单位：[中国科学院计算技术研究所](#)

## 相似文献(4条)

1. 期刊论文 [杨德志](#), [黄华](#), [张建刚](#), [许鲁](#), [Yang Dezhi](#), [Huang Hua](#), [Zhang Jiangang](#), [Xu Lu](#) [大容量、高性能、高扩展能力的蓝鲸分布式文件系统 - 计算机研究与发展](#)2005, 42 (6)

应用需求和计算机技术的发展使网络化存储系统成为网络服务器系统中I/O子系统研究的热点. 作为网络存储系统关键部件, 分布式文件系统的研究具有非常重要的意义. 蓝鲸分布式文件系统(BWFS)是国家高性能计算机工程技术研究中心基于对国内外现有研究成果的分析和研究, 自主设计实现的分布式文件系统. 它着重于大容量、高I/O吞吐率和高扩展能力等方面特性. BWFS已经用到BW1K网络存储系统中, 并通过BW1K的初步评测数据验证了这些特性.

2. 期刊论文 [范勇](#), [张建刚](#), [许鲁](#), [FAN Yong](#), [ZHANG Jiangang](#), [XU Lu](#) [蓝鲸分布式文件系统网络容错的软件技术 - 计算机工程](#)2006, 32 (18)

蓝鲸分布式文件系统(BWFS)克服了传统存储模式在性能、容量、共享、可扩展性、可管理性等方面的局限性, 通过采用连接复制、通道切换、请求重构等网络容错软件技术, 在无额外硬件支持的前提下提高可用性. 文章针对系统可用性, 分析了BWFS所面临的网络相关故障, 从系统软件角度阐述了BWFS所采用的网络容错技术, 并对其效能进行了相应的测试比较.

3. 期刊论文 [杨德志](#), [许鲁](#), [张建刚](#), [YANG De-Zhi](#), [XU Lu](#), [ZHANG Jian-Gang](#) [BWMS元数据分布信息缓存管理 - 计算机科学](#)2007, 34 (10)

BWMS是BWFS的分布式文件系统元数据服务子系统. 它充分利用系统访问负载的动态性和局部性特征, 通过简单的集中决策机制管理元数据请求负载在多个元数据服务器的分布. 为降低集中决策点可能的瓶颈限制, 集中决策点位于元数据请求处理路径的末端. 本文介绍各个元数据服务器上用来降低对后端集中决策点的压力, 提高元数据访问效率的元数据分布信息缓存, 并通过测试数据评估缓存命中率对后端集中决策点和元数据访问效率的影响.

4. 学位论文 [范勇](#) [蓝鲸分布式文件系统数据一致性语义研究](#) 2006

本文研究了蓝鲸网络存储系统的核心系统软件——蓝鲸分布式文件系统(BWFS)的数据一致性语义问题.

首先归纳数据一致性语义的主要类型, 概要地介绍若干有代表性意义的分布式文件系统针对相关问题的解决方案; 然后从描述数据一致性问题入手, 分析蓝鲸分布式文件系统的数据一致性问题; 在此基础上, 从节点间数据传输保障、文件属性更新机制和数据一致性语义模型等三个方面对蓝鲸分布式文件系统数据一致性语义进行研究. 取得如下主要成果:

1) 提出并实现基于软件的网络容错技术.

节点间数据传输的正确性及完整性是蓝鲸分布式文件系统数据一致性的基础. 针对蓝鲸分布式文件中影响较大的通道连接软故障中断异常采用独立于系统硬件 支持的连接复制、通道切换、请求重构等软件技术加以容错, 实现应用层透明的数据 无损传输, 提高了系统的可用性, 使得蓝鲸分布式文件系统的数据一致性有所保障.

2) 设计并实现自适应的带外模式文件属性更新机制.

设计并实现一种自适应的带外模式文件属性更新机制, 该机制结合机会更新、被动更新、周期更新等多种文件属性更新方式, 能够根据当前的网络状态自适应地调整文件属性的更新周期, 允许用户动态地设置更新周期基数及更新周期调整幅度.

3) 设计并实现基于授权机制的在线可调整的数据一致性语义模型.

设计并实现一种数据一致性语义模型, 该模型基于授权(可剥夺的文件级粒度的多态文件锁)机制, 允许用户在线设置蓝鲸分布式文件系统的数据一致性语义, 以满足不同应用模式的需求. 该模型提供超时一致性、释放一致性、写一致性、读写一致性等四种数据一致性语义, 支持从类NFS语义到类UNIX语义等多种数据一致性语义的兼容性.

本文链接: [http://d.g.wanfangdata.com.cn/Thesis\\_Y1005201.aspx](http://d.g.wanfangdata.com.cn/Thesis_Y1005201.aspx)

授权使用: 中科院计算所(zkyjsc), 授权号: 2de4b0f3-9c17-4e66-884c-9e40012eb13b

下载时间: 2010年12月2日