华中科技大学

硕士学位论文

并行文件系统元数据管理研究

姓名：何飞跃

申请学位级别：硕士

专业：计算机软件与理论

指导教师：庞丽萍

20040412

摘 要

随着高性能微处理器、高速网络的出现和对计算能力需求的增大，以廉价硬件和软件支撑的集群系统越来越被广泛地使用，引起集群技术的迅猛发展。

集群文件系统是集群的一个重要组成部分，作为一种集群体系结构上的并行文件系统，它为用户提供一个虚拟化大容量存储器的统一访问接口和高 I/O 带宽。由于集群文件系统的文件数据分散存储在各个结点上，文件的定位需要借助元数据来完成，元数据的管理成为管理数据的一个关键。

为了提高元数据管理的可靠性，需要具有容错能力的元数据管理系统。为此，我们针对集群文件系统的元数据管理，设计了一个双元服务器系统。该系统内部由两台元数据服务器组成，通过对元数据的镜像产生副本，保证元数据的可靠性；通过主服务器失效后从服务器接管服务来屏蔽故障，保证元数据服务的连续性。系统具有集中管理方式控制简单、易于实现和维护等优点，克服了其单一失效点的缺陷，同时又避免了分布式管理的一致性维护设计与开销。在 Linux 内核空间实现元数据镜像技术、故障检测技术、IP 接管技术和恢复技术，具有对应用程序透明性的特点。系统的最终目的是将其结构推广到多机情况下，进一步提高容错能力，实现高可用性。

为了提高元数据服务器的处理效率，提出了一种寄生式元数据存储管理方法。并行文件系统的元数据寄生在本地文件系统内核中，通过增加系统调用实现对寄生元数据的操作，保证对现有系统的兼容性。将该方法应用于 PVFS（Parallel Virtual File System）的元数据管理，元数据操作性能提高大约 5～8 倍。

关键词：集群，并行文件系统，元数据管理，高可用，镜像，故障屏蔽

Abstract

With the emergence of high-performance microprocessor, high-speed network and the increscent demand of computing power, cluster of computer with cheap hardware and software is more and more used by users, which brings the quick development of cluster technologies.

Cluster file system is an important component of a cluster. As a parallel file system on the cluster architecture, cluster file system provides a unified interface of accessing great capability storage and high I/O bandwidth for the users. Owing to the file data spreading around the cluster, locating a file needs the help of metadata, so managing metadata becomes a key of managing data.

In order to improve the reliability of metadata managing, a fault-tolerant metadata management system is wanted. So we design a dual metadata server system for the cluster file system. In this system, there are two metadata servers and the reliability of metadata is provided by metadata mirroring, and the continuous metadata service is provided by failover: after the master server fails, the backup server takes over the metadata service. This system has the merits of the center metadata management: simple control, easy to implement and maintain, but overcomes the shortcoming of single point of failures. At the same time, it avoids the design and cost of keeping consistency of the distributing metadata management. We implement metadata mirroring, failure detecting, IP takeover and recovery in Linux kernel space, which is transparent to the applications. The last aim of this system is to extend its architecture to the multi-host to get greater fault-tolerant ability and come to high availability.

In order to improve the efficiency of metadata server, an autoecious metadata storing and managing method is presented. Metadata of parallel file system are located in the kernel of the local file system and operate it by adding system call, which provides the compatibility of original system. When

we apply this method on the metadata management of PVFS（Parallel Virtual File System）, the test result indicates that the improved performance is about 5~8 times greater than the original method of it.

Keywords: cluster, parallel file system, metadata management, high availability, mirroring, failover

独创性声明

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

学位论文作者签名：何百跃

日期：2004年4月8日

学位论文授权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复印手段保存和汇编本学位论文。

保密□，可_____ 年解密后适用本授权书。

本论文属于

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：何百跃　　指导教师签名：

日期：2004年4月8日　　日期：2004年4月8日

1 绪论

本章首先概述集群与并行文件系统的研究背景，接着介绍国内外并行文件研究的研究现状，引出元数据管理问题，并概要说明相关的技术，最后介绍课题主要研究工作与论文组织。

1.1 引言

20 世纪 80 年代以来，随着高性能微处理器、高速网络和高性能分布计算标准工具的出现以及对计算能力需求的增大，由 PC 机、工作站或共享存储多处理机为部件构成的集群系统已经被广泛的应用[1]。集群以廉价而又容易获取的硬件和免费或常用软件支撑，成为价格合理的并行计算平台被用于科学计算和事务处理，对于传统超级计算机来说则是高成本低收益的。集群技术的发展对超级计算机这一概念重新定义起了重要作用。当研究者们继续推动这一领域时，新的硬件和软件发展起来了，满足集群计算不断增长的需求。尤其是当 Linux 集群作为一种低成本、高性能并行计算平台成熟起来后，出现了许多提供关键服务的软件包，大量并行应用程序使用集群系统建立原型、调试并运行，代替了专用的特别是昂贵的并行计算机平台，这已成为发展趋势。

虽然集群技术有了很大的发展，现有集群系统在某些方面却缺乏文件系统对它的支持[2]。而天气、基因数据的处理、多媒体、可视化、I/O 密集型数据库、语音图象识别等应用对大量数据 I/O 的高性能有着很大的需求，对文件系统也提出较高的要求。集群文件系统作为一种集群系统上的并行文件系统，利用在高速互联的多个 I/O 结点上分布数据块，通过多个 I/O 结点的合作来提高 I/O 效率。集群文件系统为集群的用户提供了一个虚拟化的大容量存储器，并可充分利用集群中的存储资源和网络资源。未来的集群需要依靠集群文件系统实现对系统中的所有文件设备和网络资源的全局访问，并且形成一个完整的系统映像。这样，无论应用程序在集群中的什么位置，集群文件系统允许任何用户都可以对它进行访问。甚至

在应用程序从一个结点转移到另一个结点的情况下，无需任何改动，应用程序就可以访问集群系统上的文件。目前，国内外对于集群文件系统的研究，已经成为并行处理领域中的一个研究热点。努力提高集群文件系统的I/O 性能和可用性，有着重要的意义。本论文正是在这种背景下，对集群文件系统的某些关键技术做了一些研究。

1.2 国内外研究现状

1.2.1 并行文件系统研究概况

集群文件系统和并行文件系统、分布式文件系统这些概念有着很大的联系。实际上，集群文件系统是并行文件系统在集群系统结构上的一个特例，同时作为一种处于分布式环境下的文件系统，集群文件系统也具有分布式文件系统的一些特征。目前国内外对这些文件系统的研究是一个热点。按研究的范畴分类大致集中在三个方面：商业并行文件系统、分布式文件系统和研究型并行文件系统。商业并行文件系统有SGI Origin2000的XFS[3]，HP Exemplar的HFS（High Performance File System）[4]，IBM SP的GPFS（General Parallel File System）以及Intel Paragon的PFS（Parallel File System）。商业并行文件系统提供高性能I/O，但往往依赖于特定的平台。分布式文件系统有NFS（Network File System）[5]，Coda[6]，xFS[7]等。分布式文件系统允许多个客户对文件的分布存取，但通常不提供并行应用所要求的高带宽并发写。研究型并行文件系统有PPFS（Portable Parallel File System）[8]、Galley[9]、PIOUS和PVFS[2]等，它们往往侧重于某一方面，譬如PPFS侧重于自适应缓冲和预取的研究，Galley侧重于优化磁盘存取和文件的组织。本文研究的对象主要是并行文件系统特别是集群文件系统，下面介绍几个国内外典型的并行文件系统。

PVFS 是由美国 Clemson 大学研发的一个开放源代码的 Linux 集群上的并行文件系统，目前已经有两个版本：PVFS1 和 PVFS2（如果没有特别说明，本文下文中出现的 PVFS 均指 PVFS1）。PVFS 被设计成客户/服务器结构，系统中的服务器分为元数据服务器和存储服务器，前者负责管理元数据，后者负责管理用户的文件数据。PVFS 采用了元数据和用户数据相分离的结构，元数据利用本地文件系统存储，由一台集中的元数据服务器进行管理，用户数据则采用分片的形式存储在多个存储服务器上。这样可

以实现对数据的并行存取，为用户提供较高的访问带宽。PVFS 提供了一个很好的研究平台，其主要缺点是系统中存在单一失效点（集中的元数据服务器），没有任何形式的缓存，也没有数据负载平衡机制。

IBM RS/6000 环境下运行的并行文件系统 GPFS 被设计成“外观和感觉”都像一个 Unix 文件系统，这意味着用户可以继续使用普通的 Unix 命令来操作文件。GPFS 的文件并行存放在多个存储结点上，单个文件是作为“块”分散存放在不同存储结点的磁盘上。GPFS 允许用户共享访问跨多个 SP（Scalable Parallel）结点的文件操作，通过在一个 SP 系统不同结点上运行多个进程，支持对同一文件的不重迭区域的同时读/写。

Galley 是由 Dartmouth 大学开发用于工作站集群的并行文件系统，它当前被设计运行于 IBM RS/6000 组成的集群和 IBM PS-2 并行超级计算机上。Galley 的设计目标是为应用程序提供更强大的接口，能够让用户程序显式地控制文件访问的并行方式。Galley 设计了一种三维的文件结构，简单的说，一个文件是由一个或多个子文件构成的，分开存储于独立的磁盘上。每一个子文件又是由很多个块构成。采用这种方法，不仅可以让应用程序自己决定如何分布它的数据，也可以让它们自己决定后续访问的并行度。Galley 的用户接口使得各种用户库的实现变得简单，而且在这些库中可以实现用户所需的高级功能。Galley 采用基于客户/服务器的结构，它将处理机分为两类，一类是 I/O 处理机 (IOP)，作为服务器运行；另一类是计算结点机（CP）。每一个 IOP 都管理一个磁盘，并以三维的结构来存储文件。CP 上的应用程序通过库函数与 IOP 通信以取得需要的数据。Galley 并行文件系统的缺点是它没有提供任何安全机制，以及从 IOP 崩溃中恢复过来的方法。作为一种研究型的并行文件系统，Galley 也缺乏一些产品化的并行文件系统的特征如良好的可用性等。

中国科学院计算所研发的 COSMOS[10,11]是在曙光 3000 超级服务器上实现的集群文件系统。COSMOS 的底层是基于 AIX 文件系统 JFS，由核心相关层和用户层组成。核心层是在虚拟文件系统一级中实现的，它接收来自逻辑文件系统的 I/O 请求，并以一定的格式转发给用户层。用户层则由客户、元数据管理服务器和存储服务器组成，它们协调工作，共同完成核心层转发过来的 I/O 请求。COSMOS 采用了 xFS 的无服务器设计思想，引入了双粒度的合作缓存，具有单一系统映像、可扩展性、灵活性、可用性以及数据完整性等特性。

并行文件系统研究的内容主要集中在如下几个方面：

1. 单一系统映像

在文件系统被加载后，用户可以从任何一个结点上进入文件系统的根目录，看到的是一个完全一样的目录结构。为了实现该功能就必须实现文件系统的透明性。如果集群文件系统提供严格的单一系统映像的能力，那么它应当是一个实现了透明性的全局文件系统。在用户看来，它应当和单机文件系统没什么区别，其文件的组织是单一的树型结构，文件是全局存取的，不需要用户去直接关心数据的物理存储和文件访问的细节。

2. 数据放置

由于互连网络和 I/O 总线的带宽通常都比磁盘驱动器至少高一个数量级以上，并行文件系统常常采取文件分片技术将数据分布到多个存储结点上，从而应用程序可以并行的从几个磁盘上存取数据，达到提高整体 I/O 带宽的功能[12]。研究文件的分片策略以获得好的访问性能，成为一个重要的研究课题。

3. 元数据管理

由于并行文件系统的文件数据是分散存放在多个结点上的，访问数据之前需要利用元数据进行定位，元数据的管理就成为一个关键。这正是本文要研究的主题，后面再详细讨论。

4. 合作式缓存管理

通过缓存机制，系统将已经访问过的文件块保存在内存中以备再次请求时使用，这种方法在文件块被多次使用时可以提高系统性能。随着网络速度的提高，从网络结点的内存中访问数据的速度比从本地硬盘读取速度要快；同时集群中各个结点的内存可以聚集在一起在形成一个比单一结点大得多的内存，这些前提使得合作式缓存成为现实。合作式缓存是充分利用客户机上的缓存，使得对数据的访问新增了一个层次，可以减少数据块的实际磁盘读写次数，从而提高了文件系统的效率。缓存技术同时也带来了数据一致性问题，为了解决这个问题，需要设计一致性协议。

5. 容错与高可用

一般来说集群中的结点都有失效的概率，文件系统中的数据在系统运行过程中随时都有损坏或丢失的可能。硬件或软件的故障也会使得系统提供的服务失效。在这种情况下，持续保证数据的可用性，发生故障时进行灾难抢救和恢复就成为数据容错和系统高可用性主要研究的问题。这也是

本文要研究的一个主要内容。

1.2.2 元数据管理概述

元数据（metadata）这一词在很多领域和场合都能见到，对它最常见和通俗的理解是“关于数据的数据”。日常生活中的图例、图书馆目录卡和名片可以看作是元数据。在数据库管理系统中，元数据描述了数据的结构和意义，比如某个数据库中的表和视图的个数以及名称等。本文关心的是另一种元数据，即文件系统中的元数据。

在文件系统中，元数据是用来描述一个文件系统特征的数据。对于磁盘文件系统来说，一个“文件”是指按一定的组织形式存储在介质上的信息，它实际上包含两方面的信息：存储的数据本身以及有关该文件的组织和管理信息。这些关于文件组织和管理的信息就是该文件的元数据。在 UNIX 系统中，文件的元数据主要是文件目录项和索引节点结构。目录项和文件系统的名字空间相关，每个目录项对应一个文件名。索引节点结构中则存储着文件的很多重要信息，诸如访问权限、文件主、文件大小、文件的创建时间、最后存取时间、最后修改时间等属性信息，以及文件数据的物理分布信息（如直接块指针、间接块指针）。此外，还有一些重要的系统数据（如超级块信息），记录了整个文件系统的使用情况（如空闲块的大小、已使用的空间大小等），也是一种很重要的元数据。

对于处于分布式环境中的并行文件系统，文件的元数据也包括以上这些内容。所不同的是文件的物理分布不仅包括文件在磁盘上的位置，而且还包括磁盘在系统中的结点位置。因此，元数据信息要更多一些。为了提高文件读写的 I/O 性能，并行文件系统的文件数据通常不是存储在一个单一设备中，而是将这些数据均匀地分布在多个结点上，即使是一个独立的文件也可能分片存放。正确描述数据位置或文件分片信息的参数，就成为并行文件系统中最重要的元数据。

既然并行文件系统的文件被分布在多个结点机上，单个的数据对于用户来说则是没有意义的。为了让应用程序透明地使用并行文件系统，必须对这些已经分割开来的数据进行管理。并行文件系统设计中的一个关键要素就是元数据的管理。传统文件系统中元数据存放在个体服务器上，从而限制了跨服务器或跨文件系统对数据进行共享和访问的能力。在并行文件系统中通过使用一个元数据服务器在存储网络上管理元数据，可以帮助将

更多的智能功能从个体服务器转移到存储网络之中，从而使网络中的任何应用服务器都可以访问这些功能。并行文件系统客户机软件运行在应用服务器上，通过与元数据服务器的交互来获得元数据。一旦客户机软件获得了元数据，它就可以通过网络直接访问文件数据。通过使用这种方式，并行文件系统可以提供高性能的数据访问，能够实现跨异构应用服务器的数据共享。并行文件系统需要允许系统中任何服务器上的应用访问网络中的任何文件，而且不需要对应用进行任何修改。可以说，管理并行文件系统的元数据是管理数据的关键。

归结起来，并行文件系统元数据管理的重要性主要表现在两个方面：首先，元数据是最重要的系统数据。客户读写并行文件系统中的文件，首先要对数据进行定位，只有先获得文件的元数据后，才能将客户的请求转发到正确的 I/O 服务器进行数据访问。如果不能进行正确的定位，基于文件数据的并行应用程序就无法执行。因此，必须保证系统中元数据的正确性和可靠性。其次，元数据的访问性能影响着并行文件系统的性能。在并行文件系统中，元数据的访问很频繁，而元数据文件通常又很小，这样对大量小文件的访问，会对系统性能造成冲击。为了提高元数据的性能，许多人已经做了一定的研究[13-16]。

对并行文件系统元数据管理的研究，主要集中在如下几个方面：

1．元数据放置策略

元数据和用户数据可以分开单独存放，也可以存放在一起。当元数据和用户数据分开存放的时候，设计逻辑比较清晰，元数据访问流和数据访问流分开，控制比较简单。当元数据和用户数据存放在一起的时候，可以获得更好的访问并行度，但也增加了系统的复杂性。对元数据放置策略的研究，主要是为了提高元数据的访问性能和容错。

2．元数据管理方式

并行文件系统的元数据管理可以采取集中式的管理，也可以采取分布式的管理。对于结点数比较少的集群，采取集中式的管理就可以了，用一个服务器管理元数据，控制比较简单，管理和维护也方便。其缺点是元数据服务器是系统中的单一失效点，这时候要考虑提高系统的可靠性。当集群的规模比较大，结点达到成百上千的时候，元数据数量也比较可观，单个服务器无法满足要求，需要采取分布式的元数据管理，用多个结点机来管理元数据。分布式元数据管理可以获得良好的访问并行性，而且容易实

现负载平衡。但是它的控制和实现复杂，需要维护元数据的一致性，开销比较大，设计好的一致性协议减少这种开销，提高元数据的性能是主要研究的内容。

3．元数据存储技术

并行文件系统对元数据的存储，通常采取数据库技术或者是借助本地文件系统来实现。用文件系统存储元数据是比较常见的方式。有些并行文件系统采用日志文件系统来存储元数据，以提高元数据访问的性能和容错。在分布式环境下，用数据库存储元数据比较常见。DPFS 是美国西北大学并行分布式计算中心研制的一个分布式并行文件系统，它最显著的特征就是用数据库存储文件系统的元数据，使元数据管理变得容易和可靠。SQL 相对于直接管理低级的文件来说是一个高级的、可靠的接口，可以省出编程的麻烦；数据库系统提供的事务机制使得维护元数据的一致性变得容易。

4．元数据缓存技术

对于经常访问的数据，采用缓存技术可以提高文件系统的性能。元数据的访问频度非常高，而且可能是多次访问同一数据，缓存可以大大减少访问磁盘的次数。元数据缓存技术既可以在服务器实现，也可以在客户端实现。研究适合元数据应用的缓存算法，提高缓存的命中率，是元数据缓存技术研究的一个重点。

1.2.3 可靠性与高可用概述

在分布式环境中，可靠性是一个很重要的问题[17]。在集群系统中，随着硬件组件的增长，失效的概率也在增加。运行在集群上的应用程序成比例增加的时候，就有可能发生错误。而这种增加发生错误的可能性对于长期运行的应用程序（如大规模科学计算）特别有害。在这种情况下，必须保证在错误发生时应用程序的连续性。实际上，集群计算的两个领域：高性能（HP）计算和高可用（HA）计算是息息相关的，离开了 HA 的 HP 是不现实的。

在关键性事务计算中，对于 HA 应用程序的需求也在增加。比如在线交易、电子商务、Web 站点、在线分析处理和每天 24 小时运行的服务器应用程序。这些服务器必须不间断地运行，任何中断的代价都是可观的，意味着失去机会和信誉，并把客户推向自己的竞争对手。为这些系统提供

高可用的一种方法是使用集群系统。集群在本质上也具有提供 HA 的能力，因为并行性提供了高可用的最重要的系统结构元素：冗余。

解决可靠性问题的基本途径有两种。一种方法是避错，就是试图构造出一个不包含故障的“完美”系统，其手段是采用正确的设计和质量控制方法尽量避免把故障引进系统。另一种方法是容错，当出现某些指定的硬件故障或者软件错误时，系统仍能执行规定的一组程序或算法，或者说程序不会因系统中的故障而中止，并且执行结果中也不包含系统中故障所引起的差错[18,19]。容错的基本思想是在系统体系结构上精心设计，利用外加资源的冗余技术来达到掩蔽故障的影响，从而自动的恢复系统。容错设计对系统可靠性的提高是十分有效的，它的效果远远超过了避错技术。容错可以通过硬件或软件来实现。通过硬件实现容错可以有较好的性能，但是成本高。通过软件来实现容错有较好的灵活性。几种常用的容错技术有模块冗余、N－版本编程、控制编码和基于软件的副本[20,21]。

可靠性系统容错机制的起始点是检测错误。最常见的检测技术有自检、监视器、一致性检查、检错码等。对于永久性错误，可以通过自检检测。检测的机制包括执行系统不同部件的特殊程序和验证所期望的结果输出。自检对短暂错误不是十分有效，因为这种错误在系统中的作用会随时间消失。软件监视器是一种简单的进程，负责监视其它进程的错误。与监视器的互操作可以通过几种机制完成，最常用的是心跳检测：应用程序周期性的向监视器发送通知以表示它在工作，或者是监视器给应用程序发消息请求并等待应答来判断是否处于活动状态。当监视器超过一定时间，它就假定应用程序失效。一致性检查是通过校验一些操作的中间或最终结果是否合理来实现检测的技术。

错误被检测出来后，就可以进行诊断恢复了。最常见的恢复技术是检查点[22-24]。检查点是一种允许进程在正常运行中每隔一定时间间隔保存其状态以使失效后恢复时减少丢失工作的技术。当发生错误时，使用检查点可使受影响的进程从最后一次保存的状态而不是从开始重新运行。这个技术特别适用于保护长时间运行程序出现短暂错误的情况。检查点也适用于从软件错误中恢复应用程序。对集群可以采取故障屏蔽（failover）技术进行恢复。故障屏蔽进程是集群恢复模式的内核，是一个结点失效引起的到一个替代或备份结点的切换。在集群中，如果错误是短暂的硬件问题或软件缺陷，通常在系统重构后，失效结点就可以回到集群中。如果是永久的

硬件问题，只能通过更换失效的组件来纠正问题。

1.3 课题主要工作与论文组织

本课题的研究对象是基于集群体系结构的并行文件系统的元数据管理，重点是对元数据管理的容错设计与实现，同时也对其性能改进方面作了一定的研究。

为了提高元数据管理的可靠性和可用性，需要具有容错能力的元数据管理系统。为此，本文针对集群文件系统的元数据管理，设计了一个双元服务器系统。该系统内部由主、从两台服务器组成，通过对元数据的实时镜像，使得元数据在从服务器上有一份副本，实现了元数据的容错；系统实际由主服务器对客户提供服务，当主服务器发生失效时，通过故障屏蔽由从服务器接管主服务器上的元数据服务程序，实现了元数据的连续服务。为了保证对应用程序的透明性，本文在 Linux 内核空间实现了该系统，构成系统的关键技术有元数据镜像技术、故障检测技术、IP 接管技术、恢复技术等。本系统的最终目的是将双元服务器系统结构推广到多机情形下，进一步实现高可用性。

此外，为了提高对元数据操作的性能，本文提出了一种寄生式的元数据存储管理方法，将并行文件系统的元数据寄生在本地文件系统中，按照本地文件系统的元数据管理方式由内核进行操作，通过增加系统调用来保证对原系统的兼容性。针对 PVFS 的元数据管理，在 Linux 文件系统基础上实现了该方法，并进行了性能测试和比较。

本文各章安排如下：

第一章是绪论，概述集群与并行文件系统的研究背景，介绍国内外并行文件的研究现状，引出元数据管理问题，并讨论相关的关键技术。

第二章是元数据的集中式管理和分布式管理，介绍并行文件系统的两种元数据管理方式及其相关的若干问题，对其优缺点进行评价，为第三章的设计打下基础。

第三章是具有容错能力的双元服务器系统设计，针对集群文件系统元数据管理的容错，设计了一种兼顾集中式管理和分布式管理优点的元数据服务器系统，阐述系统的设计目标、总体方案和功能模块，对其可靠性进行预测，最终将其结构推广到多机情况下以实现高可用性。

第四章是双元服务器系统的关键实现技术，阐述元数据镜像技术以及

实现故障屏蔽功能的故障检测技术、IP 接管技术和恢复技术，介绍主要技术的实现原理和关键算法。

第五章提出一种寄生式元数据存储管理方法，达到提高元数据操作性能的目的，针对 PVFS 的元数据管理给出实现方法，并进行性能测试。

第六章是结束语，对全文进行总结并展望未来工作。

2 元数据的集中式管理和分布式管理

并行文件系统的元数据管理方式很重要，通常可分为集中式管理和分布式管理两种。在实际设计中采用哪一种管理方式，往往要根据具体情况决定，需要考虑的因素有并行文件系统的体系结构、用户需求、复杂程度与开销代价等。通过对两种管理方式的分析，可以对元数据管理相关问题有全面的认识。

2.1 集中式元数据管理

集中式元数据管理占用资源少，花费的代价低，可以节省资金而提高系统的资源利用率。从系统管理和控制的角度讲，由于只有一个元数据服务器在活动，因此比较简单。通常情况下，活动的结点越多，系统管理出错的概率也越大，带来的维护开销也越大。对系统设计人员来说，集中式管理易于实现。从可用性的角度来讲，由于系统存在单一失效点，也就存在系统失效的概率。

2.1.1 集中式管理体系结构

集群系统的结构根据其存储系统结构可以分为两类：共享存储结构和无共享结构。基于共享存储结构的集群文件系统通常都是利用存储区域网络 SAN（Storage Area Network）。SAN 的目标是消除已往存储系统结构带来的带宽瓶颈，具有稳固的存储和集中的管理，以及高速数据网络的特点。共享存储结构的缺点是需要用到专用的硬件，可扩展性也受到限制。

CXFS（Clustered File System for IRIX）是 SGI 公司研究和开发的 SAN 上的共享文件系统，它允许多个异构系统同时存取 SAN 上的数据。CXFS 的元数据管理采用客户/服务器模式，也就是用一个单独的服务器来处理所有的元数据交易，是一种集中式的元数据管理。协同存取文件系统时要求保证每个存取共享数据的文件系统具有一致的最新的元数据信息。CXFS 中的数据和元数据存取如图 2.1 所示。数据在结点机和磁盘之间直接传送，元数据则通过一个集中的服务器协同操作。元数据服务器充当一个集中的

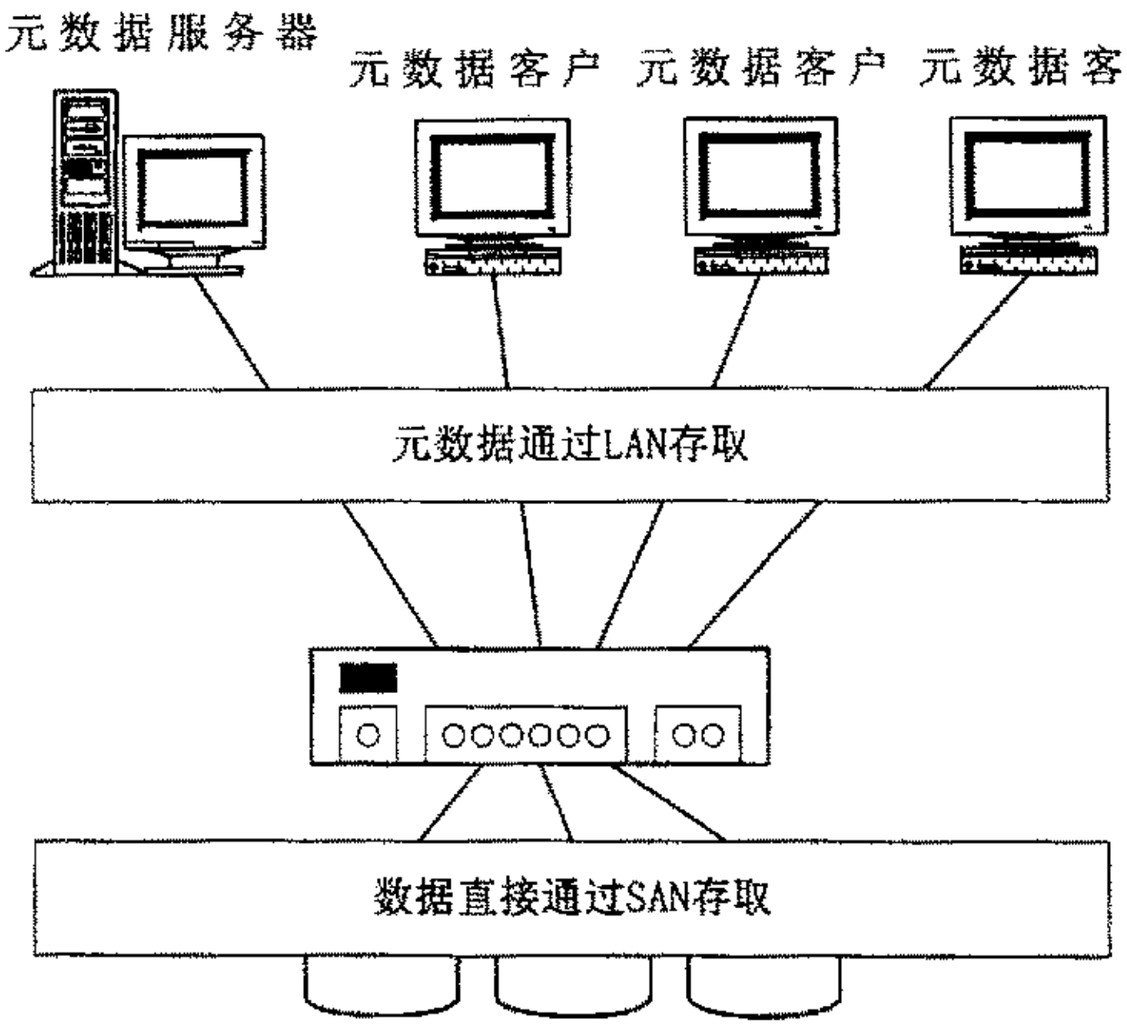


图2.1 CXFS的数据和元数据存取图

仓库，包括管理元数据日志、文件锁、缓存一致性和其它的协同功能。CXFS几乎具有本地文件存取的性能，因为只有元数据要通过服务器，数据则是直接传输的。所有的元数据请求由元数据客户通过 TCP/IP 局域网和元数据服务器联系，所有的元数据更新送向元数据服务器。

在无共享结构的集群中，每个结点有它自己的内存并且也有自己指定的存储资源，允许结点访问共有设备或资源，只要这些资源是被某一个单独的系统在某个时间所拥有和进行管理即可。这样就避免了缓存一致性和分布锁定管理的复杂性。无共享集群的一个好处是可以避免存储器内的单一失效点。实际上，如果共享存储资源失效，那么所有服务器都不能访问数据。而且，由于没有冲突和开销，可扩展性也很好。PVFS 是基于无共享集群的并行文件系统采用集中式元数据管理的一个典型例子。PVFS 的体系结构如图 2.2 所示。PVFS 系统中结点机被分成计算结点、管理结点和 I/O 结点。计算结点运行应用程序，I/O 结点存储文件数据，管理结点负责管理系统中的元数据。计算结点和 I/O 结点都有多个，而管理结点则只有一个。

PVFS 系统由元数据服务器、I/O 服务器、PVFS 本地 API 等几个部分组成。元数据服务器和 I/O 服务器都是运行在集群结点上的守护进程。

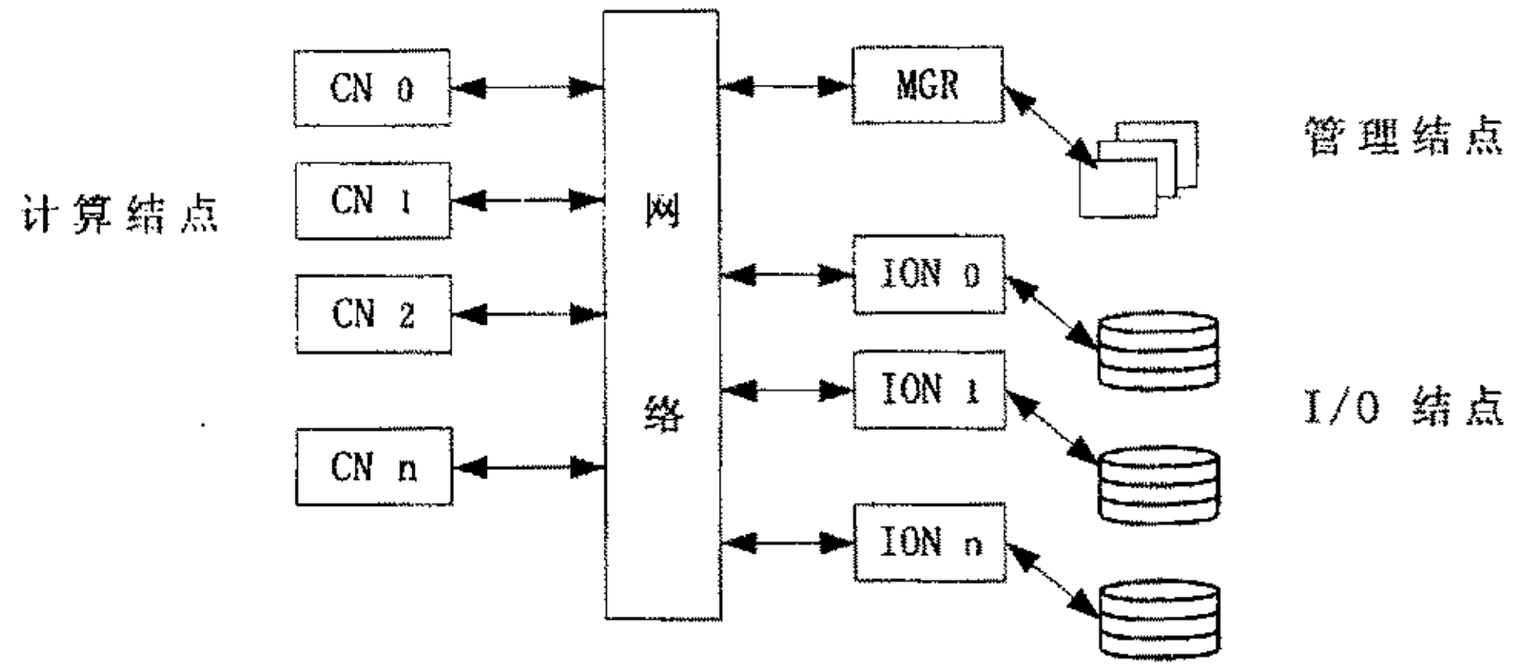
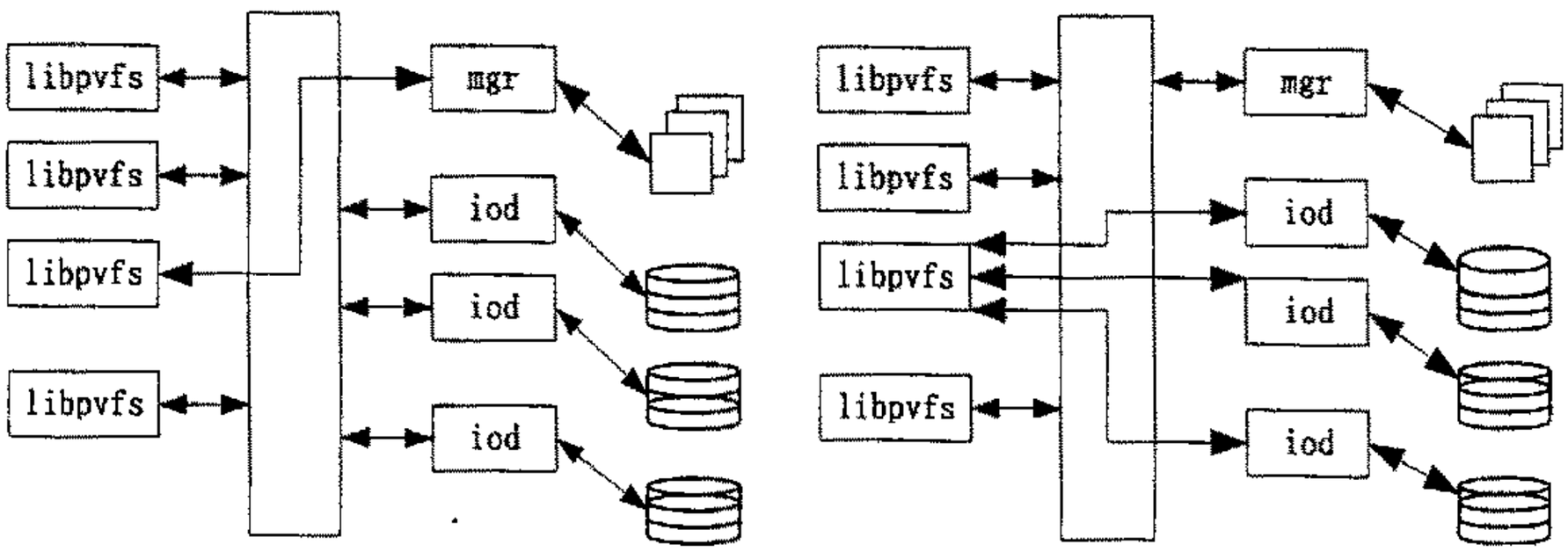


图 2.2 PVFS 系统结构图

PVFS 本地 API 提供对用户空间的透明存取。系统中数据和元数据访问如图 2.3 所示(其中 mgr 是元数据服务器，iod 是 I/O 服务器，libpvfs 是 PVFS 本地 API)。对于元数据操作，应用程序通过 PVFS 库和元数据服务器之间进行通信，对于数据存取则不需要经过元数据服务器存取路径，而是客户直接和 I/O 服务器通讯，这样可以保证良好的性能。



a 元数据存取　　b 数据存取

图 2.3 PVFS 的元数据和数据存取图

2.1.2 集中式管理的性能问题

对采取集中式元数据管理的并行文件系统来说，单一的元数据服务器是系统中的一个潜在的瓶颈，尤其是当集群的规模比较大的时候。因为系统中所有的元数据都要经过元数据服务器这条集中的存取路径，对于元数

据敏感的应用来说，可能会对系统的性能造成比较大的冲击。因此，元数据管理必须精心设计，以减轻这种瓶颈作用造成的性能损害。

最常用的提高性能的措施是缓存技术，包括设计客户端的缓存和服务器端的缓存。对于客户端的缓存，当客户获得元数据后，缓存在自己的内存中，只要元数据信息没有被其它结点修改，下次再需要该元数据信息的时候就不必从元数据服务器来取了，而是从自己的缓存中取得。当然，客户端元数据缓存带来管理的代价，就是必须保证所存取到的元数据是最新版本的。对于服务器端的元数据缓存，则可以减少磁盘的读写次数，而磁盘的读写通常比从内存中读取数据要慢得多。实践表明，缓存技术可以较大的提高元数据的性能[25]。

除了缓存技术外，还可以从其它的一些方面考虑，包括设计好的数据结构、快速查找元数据的算法，优化系统中的通信协议以减少元数据服务器和其它结点之间的通信等。从硬件上来考虑，可以采用快速以太网或者千兆网来提高元数据的存取带宽。

2.1.3 单一失效点问题

集中式元数据管理最大的问题就是单一失效点问题。元数据服务器只有一个，如果它由于硬件故障或者软件错误而导致失效后，元数据服务器提供的服务也就失效了。客户无法获得正确的元数据信息，并行文件系统文件不能定位，也就无法访问。改变文件后也无法更新元数据信息，不能反映文件的最新版本。因此，系统无法继续使用。对于一些关键的并行应用，譬如银行系统，这是无法容忍的。许多系统采用日志文件系统来减少失效发生后的恢复时间，在某种程度上能减轻所造成的损失，但是并不能提高系统的可用性。对于关键应用，必须采取冗余措施来消除系统中的单一失效点，提高系统的可靠性。

2.2 分布式元数据管理

随着集群结点数增多，规模增大的时候，一个元数据服务器可能不够了。一方面系统的元数据量越来越大，需要更多的结点来存储，另一方面单一的元数据服务器会成为系统瓶颈。这时候就要采用分布式管理了。分布式管理允许元数据的并行存取，可以保证元数据的访问带宽。但是分布式管理增加了元数据的定位难度，带来一致性维护问题。解决一致性问题

需要比较复杂的协议，比如分布锁技术，开销比较大。元数据的访问性能往往取决于并行带来的好处和一致性开销两种因素的折中。从可用性的角度来讲，分布式管理具备了冗余的结构，可以对元数据采取容错措施，因此具有比较好的抗灾难和抗失效的能力。分布式管理系统控制复杂，设计和实现的难度也大。

2.2.1 分布式管理体系结构与文件定位

无论是基于共享存储结构还是非共享结构的集群文件系统，都可以采取分布式的元数据管理。GFS是一个共享存储结构的并行文件系统采取分布式元数据管理的例子，元数据由所有的机器管理。非共享结构的并行文件系统比如COSMOS是采用分布式的元数据管理。采用分布式元数据管理的集群文件系统基本上也是由存储服务器、管理服务器、客户三个部分组成，所不同的是管理服务器通常有多个。有些系统选取部分结点作为管理服务器，有些系统则是完全对等的结构，每个结点机器都可以充当管理服务器。

以 COSMOS 为例来看分布式元数据管理的相关问题。COSMOS 的系统结构如图 2.4 所示，它由核心扩层和用户层组成。用户层由客户、元数据管理器和存储服务器组成，其中元数据管理器负责维护 COSMOS 文件的元数据，处理来自客户对于 COSMOS 文件元数据的请求，通常是平均每数个结点配置一个。由于 COSMOS 文件系统中存在多个存储服务器和元数据管理器，因此对于每个文件和目录，必须明确他们的元数据和数据分别存放在哪个服务器上。COSMOS 中的每一个元数据管理器和存储分组加入系统时，都被分配一个唯一的数字。当创建一个文件时，客户会从当前可用的元数据管理器和存储分组中，选择一个元数据管理器存放文件的磁盘 inode，选择一个存储分组来存放文件数据。文件创建好后，会得到唯一标识这个文件的 inode 号，根据 inode 号就可以直接找到这个文件所在的元数据服务器和存储分组，客户根据文件 inode 号转发 I/O 请求。COSMOS 的每个元数据管理器只负责管理一部分文件的元数据以及这一部分文件的缓存一致性。

一个普通文件系统的各类数据在磁盘上的组织通常包括引导区、超级块区、inode 区和数据区。COSMOS 文件系统不考虑引导区，系统的数据和元数据是基于各结点的本地文件系统实现的。COSMOS 文件系统的超级

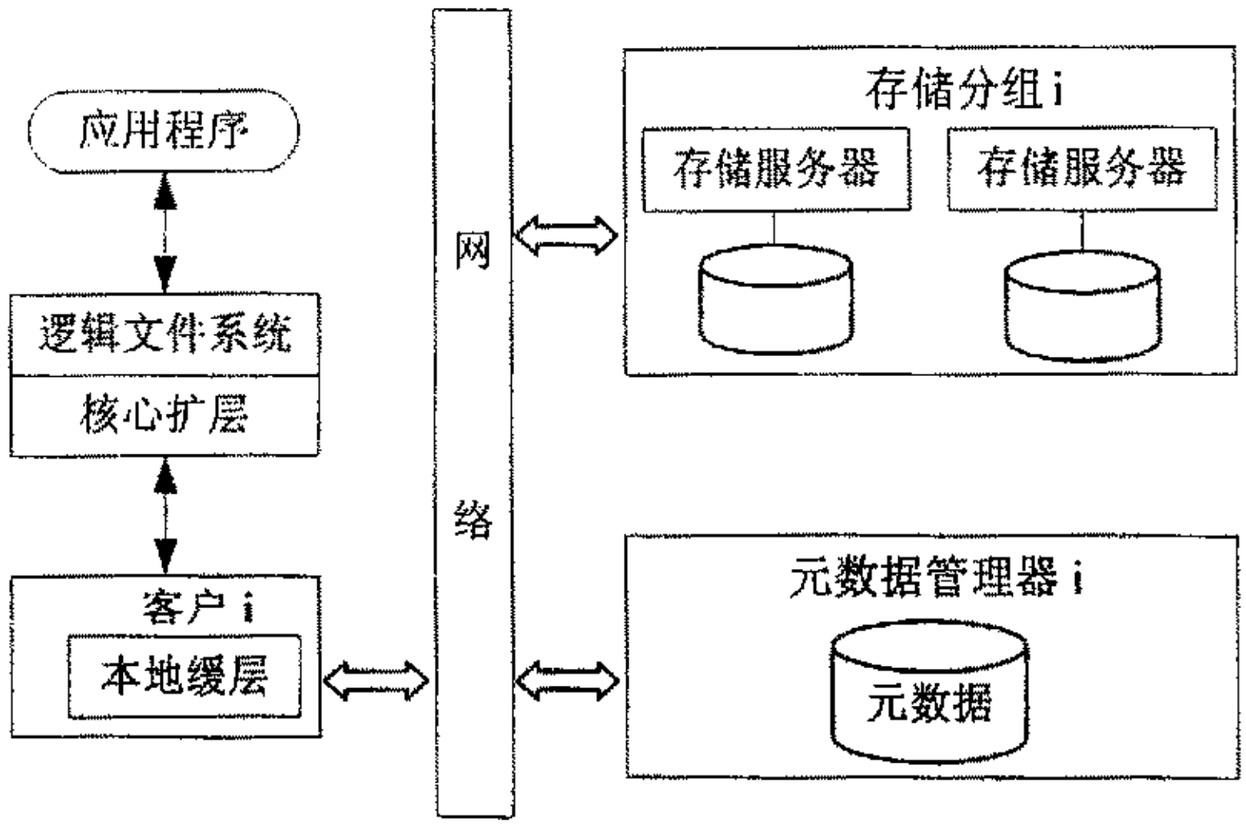


图 2.4 COSMOS 系统结构图

块用于存放文件系统一级的信息，由整个系统指定一个元数据管理器作为元数据“超级管理器”进行管理。inode 是 COSMOS 文件的元数据，根据 inode 号可以定位 COSMOS 文件。每个元数据管理器管理一个 COSMOS 文件子集，这个文件子集中的所有文件的元数据都存放于一个叫 ifile 的文件中。

COSMOS 借鉴了 xFS 的无服务器设计思想，它的文件定位还是比较简单的，相比之下 xFS 就要复杂得多。这是因为 xFS 实现了日志管理功能和合作式缓存管理，而且 COSMOS 的目录文件和元数据都存储在元数据管理器上，而 xFS 是一种完全分布和对等的结构，元数据也是存在存储服务器上的。xFS 需要查找多个表才能最终定位数据，过程很复杂。

2.2.2 元数据容错

在分布式的元数据管理中有多个元数据管理服务器，即使某个服务器失效了也至少不会影响全局。元数据的分布提高了访问的并行性，同时也可以对元数据采取容错措施，提高元数据的可靠性。对于非共享结构的并行文件系统，最常用的容错技术是复制[26-29]，即将存在于每个服务器中的元数据在多个服务器上备份。元数据相对于用户数据来说是小量数据，对元数据采取复制技术是比较适合的。

复制是把数据从一个系统通过网络拷贝到另一个系统的方法，目的是

在目标系统上维护一份原系统上的数据同样的备份。其机制通常是在原系统上通过一个额外的驱动层来中途截取写操作并把修改过的数据传送到目标系统一个相应的驱动层或守护进程。通常的复制方法有块级复制、文件系统级复制、文件级复制。块级复制是在磁盘块层截取写，把数据复制到目标系统。这种方法对使用底层磁盘卷的文件系统或应用是透明的。文件系统级复制是在文件系统层截取写，因此是特定于底层文件系统的。文件级复制是复制修改过的数据到元系统上指定的某个或一组文件中。

按复制的模式分，复制技术又可以分为三类：同步模式、异步模式和周期模式[30,31]。对同步模式来说，应用提交的写请求直到数据成功的写到原系统和目标系统才算完成。异步模式则是应用提交的写请求立即写到原系统，并加入到发送到目标系统的等待队列。队列中的数据在系统资源和网络带宽允许的情况下发送到目标系统。周期模式是周期性的发送数据到目标系统，是一种类似批处理的方式。数据复制的同步模式对写敏感的应用产生很大的性能影响，因为每次写都要传送更新数据到远端的目标系统而产生额外的负载。异步模式和周期模式可以大大减少这一影响，但都有数据丢失的危险，如果原系统失效了而此时应用数据还没有复制到目标系统，当应用程序迁移到目标系统后这些数据就丢失了。

复制也会带来一致性问题，需要采取一定的措施来保证，常见的一种策略是 ROWA (Read-One，Write-All)。例如 Coda 文件系统采用服务器之间的卷复制技术，当一个客户需要读一个文件的时候，它与该文件所属卷的可存储卷组中的一个成员服务器连接；当更新文件关闭会话时，客户端向可存储卷组中所有的成员服务器传送更新。

2.2.3 一致性维护问题

在分布式环境中，一致性问题始终是个很重要的问题。客户的并发访问、缓存的使用以及数据的复制都会带来一致性问题。对于采取分布式管理的元数据来说，同样存在这些问题，解决数据一致性问题的方法同样适用于元数据。

为了提高性能，许多并行文件系统都采用了缓存技术。当多个结点同时存取某个文件时，如何保持该文件在各个结点的缓存中的一致性以及保证读取到的文件是最新的就成为一个问题。一种解决的办法是采用文件版本号和锁，允许同时读，不允许同时写。每当一个客户缓存中的文件修改

后即向元数据服务器发更新消息，由元数据服务器更新后得到一个新的版本号，并向其它拥有该文件副本的客户发失效消息。当某个客户要写文件时需要向元数据服务器申请一把写锁，当写结束关闭文件后向元数据服务器发更新消息，由元数据服务器回收写锁，更新元数据。如果在此其间又有其它客户要写该文件则需要等待直到获得写锁，若只是读则允许，当然这时候读到的可能是脏数据，元数据服务器发失效消息后才知道，由客户重新请求数据。

Coda 采取一种称为回叫（callback）的机制来保证缓存一致性。对每个文件而言，客户从哪个服务器取得它，则该服务器记录这些客户，称该服务器记录了客户的一个回叫许诺（callback promise）。当客户更新文件的本地拷贝时，它通知服务器，服务器给有该文件拷贝的其它客户发一个失效的消息，从而丢弃还被其他客户拥有的该文件的回叫许诺。

2.3 混合管理方式

集群文件系统为了获得高I/O吞吐量，要求并行的从集群中各结点读写数据和元数据。另一方面，为了保证文件系统的一致性和POSIX语义，要求从各个结点对数据和元数据进行同步存取，这样又限制了并行性。GPFS针对不同的数据和元数据采用分布式锁和集中管理解决这个矛盾。

首先，GPFS使用分布式锁来保证磁盘上元数据的一致性，任何文件系统操作在读或者更新文件系统数据和元数据之前都需要一个合适的读/写锁来同步其它结点上的冲突操作。GPFS的体系结构基本上是基于分布式锁的。另一方面，对那些经常被不同结点存取和更新的数据或元数据采取更加集中的方式来管理：所有的冲突操作都转发到一个中心结点，由该结点来处理读或者更新请求，这是出于对锁冲突开销与转发到中心结点开销的权衡，可能后者开销更小。GPFS对不同数据采取不同的技术：用户数据的更新采取字节锁技术，对文件元数据采取动态选择的“元结点”进行集中管理，对文件系统元数据（如磁盘空间分配表）则采取分布锁技术。可见GPFS对元数据的管理既不是纯粹的分布式管理，也不是纯粹的集中式管理，而是一种混合的管理方式。

2.4 小结

并行文件系统的元数据管理方式是元数据管理中的一个重要问题，对

于具体的文件系统，应该采取合适的方式。本章对元数据管理的集中管理方式和和分布管理方式做了详细分析，对它们各自的体系结构及其相关的问题、优缺点等分别进行了描述。集中式管理具有控制简单、设计和实现难度小、维护方便等优点，如果解决了它存在的单一失效点问题，对于规模比较小的集群是一种较好的管理方式。当集群的规模很大，结点达到成百上千的时候，集中式元数据管理会成为系统中的瓶颈，这时候应该考虑分布式的元数据管理方式。一方面，分布式元数据管理由于系统本身的冗余特性，可以对元数据采取容错措施，消除单一失效点问题。另一方面，由于元数据的分布性，访问文件需要元数据定位，对元数据本身又有一次定位，增加了复杂性；而且为了维护元数据的一致性，需要设计复杂的协议并带来可观的开销。如果能对这两种管理方式扬长避短，则可能达到事半功倍的效果。下一章的具有容错能力的双元服务器系统设计正是基于这一个出发点。

3 具有容错能力的双元服务器系统设计

元数据及其服务的可靠性对集群文件系统来说非常重要，本文设计了一个具有容错能力的双元服务器系统[32]，它克服了集中式管理的单一失效点问题，又避开了分布式管理的一致性维护开销。最后我们把它推广到任意多结点的情况，以进一步的提高冗余度和容错能力，实现高可用性。

3.1 设计目标

设计一个集群文件系统元数据管理高可用系统，最主要的功能目标是保证元数据的可靠性和服务的可靠性。对于元数据的可靠性，就是要保证元数据具有容错的功能。对于服务的可靠性就是要保证元数据服务器对客户提供的服务具有连续性。

高可用系统应具备当失效事件发生时保护数据的能力。元数据的容错性就是保护元数据不丢失，在出现灾难的时候可以恢复。灾难通常可以分两种情况：用户错误或程序错误导致数据丢失；磁盘失败和主机失败导致数据部分或者全部丢失。对于第一种情况，解决的办法是保证程序的健壮性和对客户进行培训。本文的元数据容错主要是针对第二种情况。对于这种情况，主要的容错技术是数据复制。数据复制可以分为冷备份和热备份两种方式。许多服务器采取热备份的方法，因为它的实时性比较好。对于元数据的容错，本文采取镜像复制的方法。镜像不可避免的引起系统性能的下降，设计一个适合于元数据应用的镜像方案，保证镜像的效率是一个主要的目标。

高可用系统一个关键的考虑是保证服务的连续性。当服务器提供的服务由于软件故障失效后，一种解决的办法是重新启动服务。而当服务器发生硬件故障从而当掉后，系统提供的服务也就中断了。如果采取重新启动机器引导系统的策略，则这段时间内客户必须等待。从事关键性任务的集群需要用多台结点用于某种服务，在一个系统中组件（硬件或软件）失效时，重要的应用程序可以被转移到另外的服务器上，这样可避免故障并保证应用程序的可用性。这个过程就是故障屏蔽，它保证系统不会发生全局

永久性损失。在本文的双元服务器系统设计中，也是通过故障屏蔽来保证元数据服务的连续性。这种故障屏蔽过程应该是完全自动的和透明的，不需要管理员的干预或客户手动从新连接。

系统的另一个重要设计目标是透明性，即对于现有的服务器应用程序能够不加修改的纳入到该系统中，譬如在PVFS的元数据管理中，只要加入该系统而不需要改动PVFS的源代码，PVFS就能正常运行，而且提供原来不具有的容错能力。好处是当服务器应用程序由于版本的升级而改动的时候，系统不需要重新编码的开销。否则，因为修改应用程序工作量大，版本不断升级将不堪重负，更重要的是当没有源代码参考的时候修改应用程序甚至是不可能的。

其它的目标包括：简单性，即系统设计应该是简单的，并且是易于实现的；经济性，即系统不应该用到专用而昂贵的硬件，用软件的方法来实现。此外，还应该努力保证系统的性能，虽然提高系统的可靠性必然要以牺牲性能为代价，这种代价应该在可接受的范围内。

3.2 总体方案与功能模块

具有容错能力的双元服务器系统用两台结点机做元数据服务器，一台为元数据主服务器，一台为元数据从服务器。两台服务器同在一个局域网内，用 TCP/IP 网络进行连接。主、从服务器的硬件配置可以相同，也可以不同，都配置 Linux 操作系统。其系统结构如图 3.1 所示。正常工作时，客户端向服务器提交元数据请求（包括读请求和更新请求），主服务器对客户提供服务。与此同时，主服务器将自己磁盘上的元数据镜像到从服务器，这样元数据在从服务器上有一份副本，通过冗余实现元数据的可靠性。另外，主、从服务器相互发“心跳”信息，监控对方的状态。当主服务器失效后，通过故障屏蔽，从服务器上升为主服务器并接管原来主服务器上的应用程序，利用镜像目录中的元数据继续提供服务。这样就实现了元数据服务的连续性。当原来的主服务器恢复后，重新加入到系统中作为新的从服务器，即最初的主、从服务器交换了角色，这样可以避免重新切换到原来的主服务器上去，节省了一次切换的开销。当第二次失效发生的时候，再次发生一次切换。如果是从服务器失效了，应该尽快的恢复从服务器并加入到系统中来。因为从服务器对客户不提供服务，系统进入单机状态下由主服务器继续提供服务。但是，当从服务器失效后由于只有一台服务器

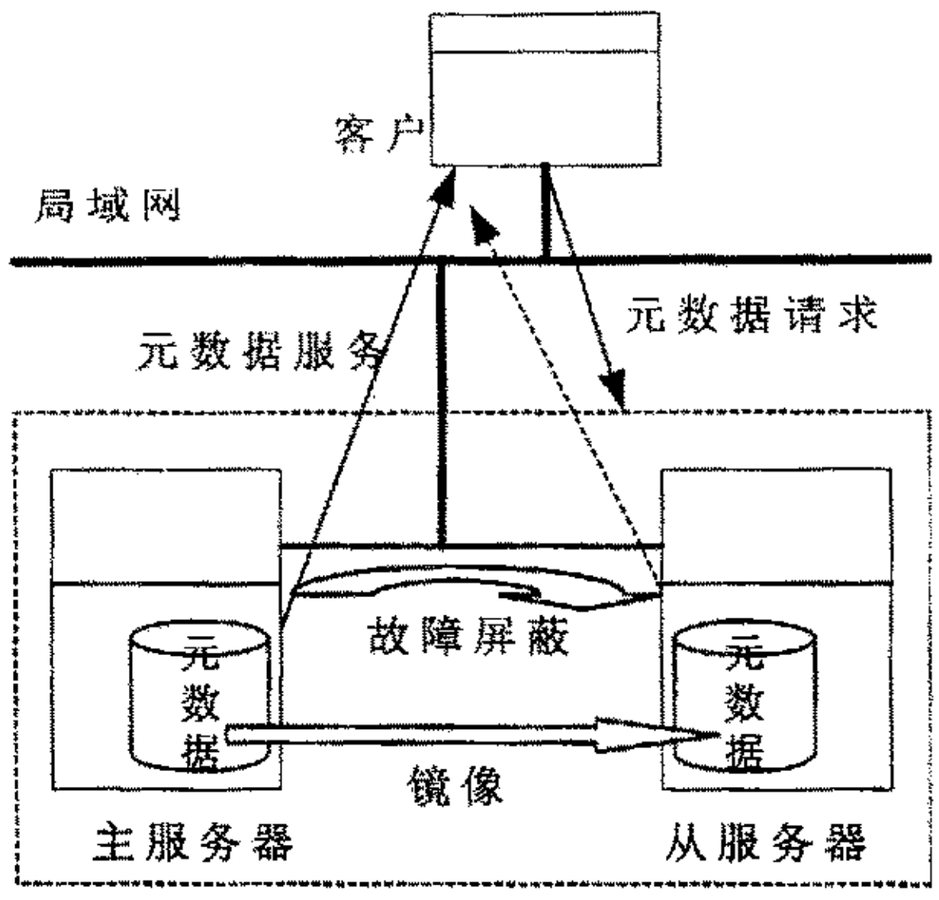


图 3.1　双元服务器系统示意图

处于活动状态，就不能再次发生失效，否则整个系统就失效了。因此虽然对客户没有影响，却潜在的增加了系统失效的概率。对于主服务器的失效也是如此，应该尽快恢复到双元服务器状态，虽然两台服务器同时失效的概率是存在的，但毕竟这种概率要小得多。不仅如此，一台服务器的失效还会导致元数据镜像的失败，在失效服务器修复之前不能保证两台服务器上元数据的一致，需要采取相应的恢复和同步措施。

双元服务器组成一个系统，对客户端来说应该是透明的。也就是说客户端并不知道在哪台服务器上获得元数据服务，以及何时发生了服务的切换。系统就像一个虚拟的服务器一样，对外部是一种透明状态，在系统内部则容许发生部分失效和和服务的切换。而且，发生切换的时候，客户端至多只能有短暂的延迟，而不应该造成功能上的失效。失效的服务器修复后添加到系统中来，客户端也应该感觉不到。

具有容错能力的双元服务器系统由两个功能模块组成：元数据镜像模块和故障屏蔽模块。元数据镜像模块负责将主服务器上的元数据所在目录文件实时镜像到从服务器一个镜像目录中，当主服务器失效后从服务器将用镜像目录中的元数据提供服务。故障屏蔽模块实现完全自动和透明的接管，保证为用户连续提供服务。故障屏蔽模块由监控模块、IP 接管模块和恢复模块等子模块组成。系统的模块结构如图 3.2 所示（元数据镜像模块的组成详见第四章元数据镜像实现部分）。

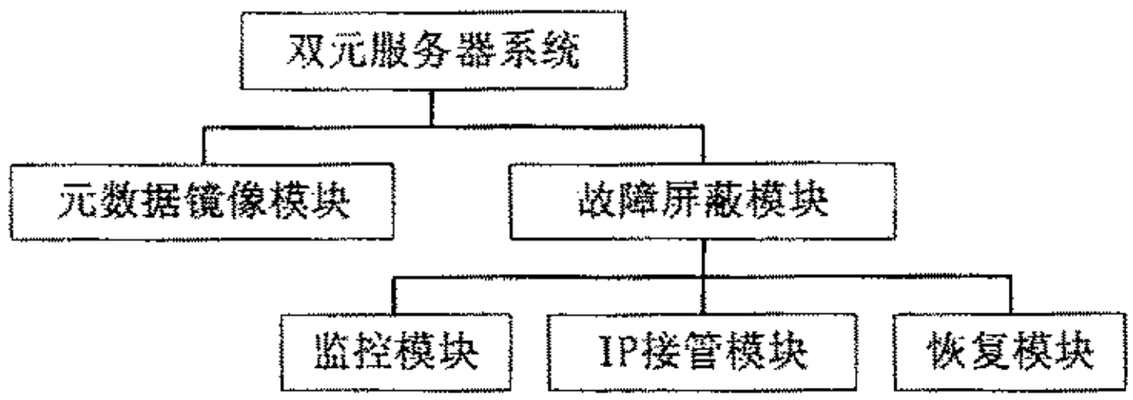


图 3.2 具有容错能力的双元服务器系统模块结构图

1. 监控模块

实现高可用的首要条件是检测出系统是否发生了故障，故障检测是实现容错计算、提高系统可靠性的重要环节。监控模块负责监视服务器状态及控制其它模块的运行。对于双元服务器系统，故障主要是指结点机的失效。本文采取基于心跳的检测机制，在活动状态下主、从服务器互相发心跳信息来给对方，以此来判断对方的状态。一旦某个结点失效，对方结点就不会再收到来自它的心跳信息，如果没有收到心跳信息的时间间隔超过了设定的某个超时值后，对方结点就认为它已经失效了。这时监控模块启动或者停止相关的模块运行。

2. IP 接管模块

实现服务接管的一个重要组成部分是 IP 地址的接管。通常客户端应用程序都知道服务器的知名 IP 地址，与它进行通信以获得服务器提供的服务。在双元服务器系统中，主服务器失效后会由从服务器接管，原来客户端所知道的服务器 IP 地址需要由从服务器来接管，这样客户端仍然利用服务器的知名 IP 地址进行访问。在本文的设计中，采用了开放源代码的项目 LVS（Linux Virtual Server）中的虚拟 IP 地址技术[33]。主、从服务器配置成双网卡，一块网卡处于活动状态，另一块网卡处于备用状态。当系统启动时，主服务器的备用网卡激活，并在上面绑定一个虚拟 IP 地址 VIP，也就是服务器的知名 IP 地址，客户通过 VIP 与主服务器进行通信。当主服务器失效时，从服务器激活自己的备用网卡并绑定 VIP。客户发往 VIP 的请求就会被从服务器接收，从而实现服务的接管。

3. 恢复模块

故障恢复是指为了接管一个已发生了故障部件的工作负载所需要做的工作。对于具有容错能力的双元服务器系统，故障恢复主要是完成元数

据的同步。主、从服务器之一失效后到恢复正常之前那段时间由于只有一个服务器运行，新生成的元数据无法镜像，因此失效服务器修复后可能存在镜像目录中的元数据不一致，需要进行一次额外的元数据同步过程。本文采取日志文件的方法，当失效服务器恢复后重新加入到系统中来时，监控模块会检测到，这时恢复模块将日志文件中的元数据更新镜像到修复后的服务器。这里有一个问题，就是由于对客户的连接信息没有备份，当失效发生时，当前的连接信息也丢失了，需要客户端重新连接一次。基于TCP/IP 连接的备份可以解决这个问题，但是要付出比较大的代价，因为需要保证两个服务器连接状态的一致性，其同步的开销很大[34]。对于长时间的关键服务，连接备份是有好处的。而元数据存取一次的时间非常短，传输的数据量也非常小，丢失一次连接没有多大的损失，连接中断后客户端会自动的重新连接，因此丢弃一次连接的做法是比较合理的。

3.3 双元服务器系统可靠性预测

1．可靠性评价标准

可靠性的度量取决于应用程序，可以从不同角度来看。评价系统可靠性最确切的方法是用一个概率函数来表示系统在任意时刻的可靠性。常用的概率函数有：

可靠度 R(t)：在给定的时间间隔[0, t]内，系统仍然能正常执行其功能的概率。和 R(t)相关的概念是失效率，即单位时间内的失效元件数与元件总数之比。

可用度 A(t)：系统在时刻 t 处于正常的概率。

可维修度 M(t)：在指定时间内，将一个失效系统恢复到运行状态的概率。和 M(t)相关的概念是修复率，即在单位时间内完成修复的概率。

但是有时候用一些简单参数来表示系统的可靠性更方便。最常见的参数有：

MTTF（Mean Time To Failure)：平均无故障时间，即系统从启动开始到首次失效的平均运行时间。

MTTR（Mean Time to Repair)：平均修复时间，即修复失效并使系统回到正确服务的平均时间。

如果不知道系统的失效率和修复率，为了评价一个容错系统的可靠性，可以构造一个与之功能相同的非容错系统，然后计算容错系统相对于

非容错系统的可靠性改进值，就是所谓的比较参数。通常计算两个系统的可靠性差值或者比值。

2．双元服务器系统的工作模式

任何时刻，具有容错能力的双元服务器系统处于下面四个系统工作模式之一：

（1）正常模式：主、从服务器都在活动状态，相互收发心跳消息进行监控；主服务器对客户提供服务，同时将元数据镜像到从服务器。

（2）主服务器失效：从服务器检测到主服务器失效，启动 IP 接管模块，接管主服务器上的服务，进入单机工作模式，同时启动恢复模块工作。

（3）从服务器失效：主服务器检测到从服务器失效，进入单机工作模式，同时启动恢复模块工作。

（4）主、从服务器同时失效：系统失效。

3．系统失效情况的 Markov 模型

对双元服务器系统做如下假设：

（1）每个服务器的失效统计独立，服从指数分布，失效率为 λ。

（2）失效服务器的修复服从指数分布，修复率为 μ。

则可以计算得单个服务器的平均无故障时间为：

$$MTTF_1 = \frac{1}{\lambda}$$

平均修复时间为：

$$MTTR_1 = \frac{1}{\mu}$$

根据双元服务器系统的工作模式，建立系统失效情况的 Markov 模型如图 3.3 所示。其中状态 0 表示正常工作状态，没有失效发生；状态 1 表示一个服务器失效（主服务器失效或者从服务器失效，在这里是等价的）；状态 2 表示两个服务器都失效，导致系统失效。

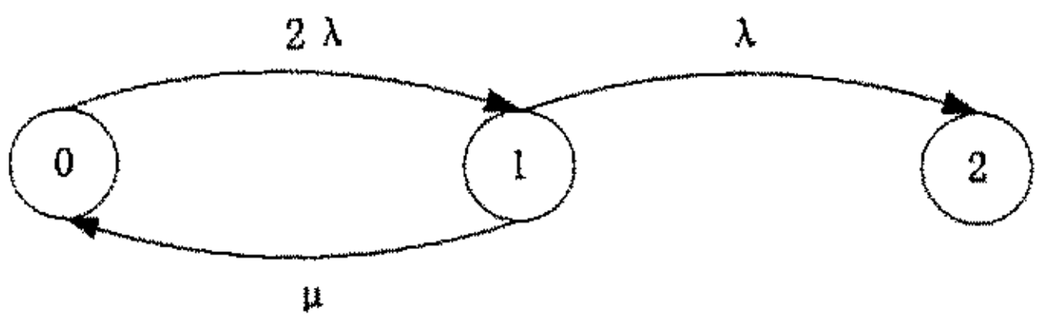


图 3.3　双元服务器系统失效情况的 Markov 模型

4. 可靠性推导结论

设在 t 时刻，系统处于状态 0、1、2 的概率分别记为 p_0（t）、p_1（t）、p_2（t），根据图 3.3 的模型建立状态方程：

$$\begin{cases} p_0'(t) = -2\lambda p_0(t) + \mu p_1(t) \\ p_1'(t) = 2\lambda p_0(t) - (\mu + \lambda) p_1(t) \\ p_2'(t) = \lambda p_1(t) \end{cases} \quad (1)$$

初始值为：

$$p_0(0) = 1, p_1(0) = 0, p_2(0) = 0 \quad (2)$$

系统可靠性为：

$$R(t) = p_0(t) + p_1(t) \quad (3)$$

系统的平均无故障时间为：

$$MTTF = \int_0^\infty R(t)dt \quad (4)$$

在式（2）条件下解方程组（1）可得：

$$p_0(t) = \frac{r_2 + 2\lambda}{r_2 - r_1} e^{r_1 t} + \frac{r_1 + 2\lambda}{r_1 - r_2} e^{r_2 t}$$

$$p_1(t) = \frac{(r_2 + 2\lambda)(r_1 + 2\lambda)}{\mu(r_2 - r_1)} e^{r_1 t} + \frac{(r_1 + 2\lambda)(r_2 + 2\lambda)}{\mu(r_1 - r_2)} e^{r_2 t}$$

其中：

$$r_1 = \frac{-(3\lambda + \mu) + \sqrt{\lambda^2 + 6\lambda\mu + \mu^2}}{2}$$

$$r_2 = \frac{-(3\lambda + \mu) - \sqrt{\lambda^2 + 6\lambda\mu + \mu^2}}{2}$$

代入式（3）和式（4）得：

$$MTTF = \frac{\lambda + \mu}{2\lambda^2} + \frac{1}{\lambda}$$

则比较参数：

$$\frac{MTTF}{MTTF_1} = \frac{1}{2}(3 + \frac{\mu}{\lambda}) = \frac{1}{2}(3 + \frac{MTTF_1}{MTTR_1})$$

通常情况下 μ >> λ（例如对于磁盘的失效情况有 λ≈0.001，μ≥0.9，二者相差 2 个数量级），可见，相对于单个服务器而言，双元服务器系统的可靠性有了很大提高，在大多数情况下已经满足可靠性要求。

3.4 双元服务器系统结构的推广

双元服务器系统不仅可以用于集中式的元数据管理，也可以用于分布式的管理，因为对于分布式管理系统中的每一台元数据服务器都可以直接用双元服务器系统代替（即以“一对伙伴”作为大粒度的容错单元），通过提高每台服务器的可靠性来提高整个系统的可靠性。但是由于在该系统中元数据的拷贝副本只有一份，不能容忍两台服务器的同时失效。本系统是在两个结点的情况下实现的，在一个复杂的集群环境中，为了进一步提高容错能力，可以将双元服务器系统结构进一步推广到多机情况下。

首先是对元数据镜像技术的推广。一份元数据副本冗余只能容忍一个结点的失效，在推广的系统中由 N 台结点构成（结点编号从 1 到 N），希望生成的元数据副本是 M 份（2<=M<=N），即 1 到 M 的镜像方式。采取的策略是结点 i 上的元数据镜像到结点 i+1, i+2, …((i+M) mod N)(1<=i<=N)，这时候结点间的镜像关系组成一个 N 个顶点的有向图，其中每个顶点的入度和出度都为 M。推广的系统结构如图 3.4 所示。这样，每份元数据都有另外的 M 份拷贝，可以容忍结点数最大为 M 的失效。

其次是对故障屏蔽的推广。在具有容错能力的双元服务器系统中，故障屏蔽是指从一个失效结点到另一个备份结点的切换的情况。在推广的系

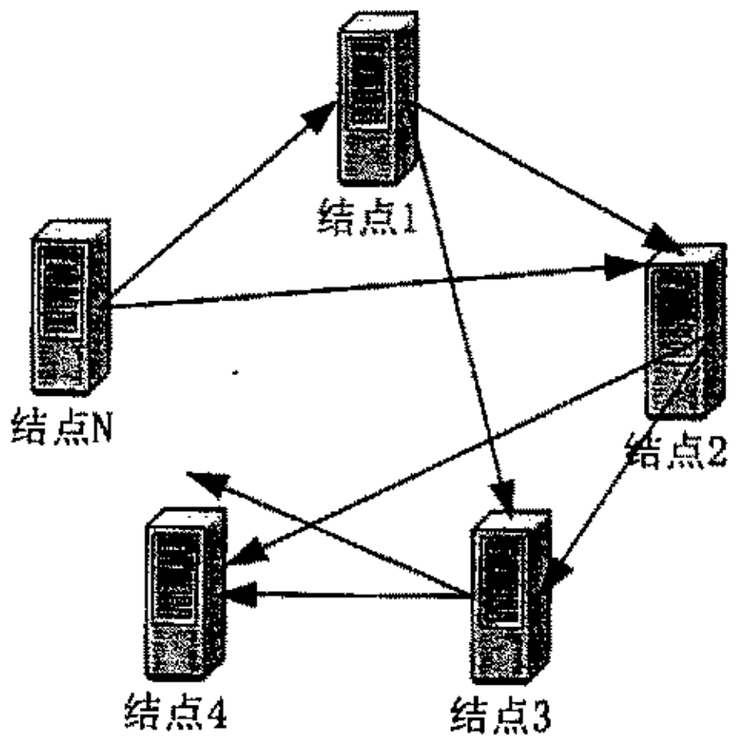


图 3.4 推广的系统结构

统结构中，每个结点都有多个备份结点，当某个结点失效的时候就会出现切换到哪个备份结点的问题。这时候需要采取一种选举算法来确定候选的接管结点。最简单的选举算法是最小结点号优先的算法，即当结点 i 失效后依次考察结点 i+1, i+2, …, 直到选中一个活动的备份结点。

推广后的系统具更高的容错能力，达到高可用性目的。但是，这时候系统比较复杂，会有更多的问题需要考虑。其中之一就是要保证 M 份元数据副本的一致性，需要设计一致性协议。另一个值得考虑的因素是系统的性能问题。

3.5 小结

为了提高集群文件系统元数据管理的可靠性，本文设计了一个具有容错能力的双元服务器系统。该系统由主、从两台服务器组成，其中主服务器对客户提供服务，同时实时的镜像元数据到从服务器。通过对元数据的备份产生副本保证了元数据的可靠性。当主服务器失效后，通过故障屏蔽，从服务器接管主服务器上的元数据服务应用程序，保证了元数据服务的连续性。系统对客户来说是透明的，包括透明的接管服务和访问服务器，客户不需要区别是主服务器还是从服务器，也不知道何时发生了接管或者切换。系统的设计兼顾了集中式元数据管理和分布式元数据管理的优点，同时在一定程度上克服了各自的缺点。对设计者来说简化了复杂性，减小了实现的难度；对用户来说保证了可靠的元数据服务，由于不需要特殊的硬件支持，用软件的方法实现，减少了成本。同时实现了应用程序的透明性，减小了应用程序升级时带来的维护代价。对系统的可靠性预测表明，双元服务器系统的可靠性相对于集中式服务器提高了很多，例如针对磁盘的失效情况可靠性提高 2 个数量级。最后，讨论了将该系统推广到任意多结点的情况，实现更高的冗余，达到高可用性的目标。

4 双元服务器系统的关键实现技术

通过元数据镜像产生副本，保证元数据的可靠性；通过故障屏蔽，当主服务器失效后，由从服务器来接管主服务器的服务，保证元数据服务的连续性。为了保证对应用程序的透明性，在 Linux 内核中实现了元数据镜像技术、故障检测技术、IP 接管技术和恢复技术。

4.1 元数据镜像技术

在 WINDOWS 和 UINX 环境下，许多计算机厂商都推出了采用磁盘镜像技术的服务器产品。这些产品通常比较昂贵，需要用到专用的硬件如 RAID（Redundant Arrays of Inexpensive Disks）[35]。也有一些用软件实现镜像的方法[36-38]。随着 Linux 技术的发展，目前越来越多的服务器开始采用 Linux 操作系统，Linux 良好的稳定性、可靠性、可扩展性越来越得到业界的认可。然而 Linux 下的磁盘镜像方案却不多。一种方案采用 GFS 和 SAN 实现实时数据镜像，如联想的 SureFibre820/920 系列存储系统为用户提供了一套以 SAN 环境和异地集群系统为基础的方案来实现异地的数据实时复制，通过专线实现物理存储设备之间的数据交换。这种方案的成本高，付出的代价比较大。第二种方案是选择自由软件，比如 rsync 就是这样的一个软件。rsync 可以镜像保存整个目录树和文件系统，文件传输效率也比较高，能够满足绝大多数要求不是特别高的备份需求。但是这种备份操作不是实时的，难以保证备份服务器中的数据与主服务器上的一致。

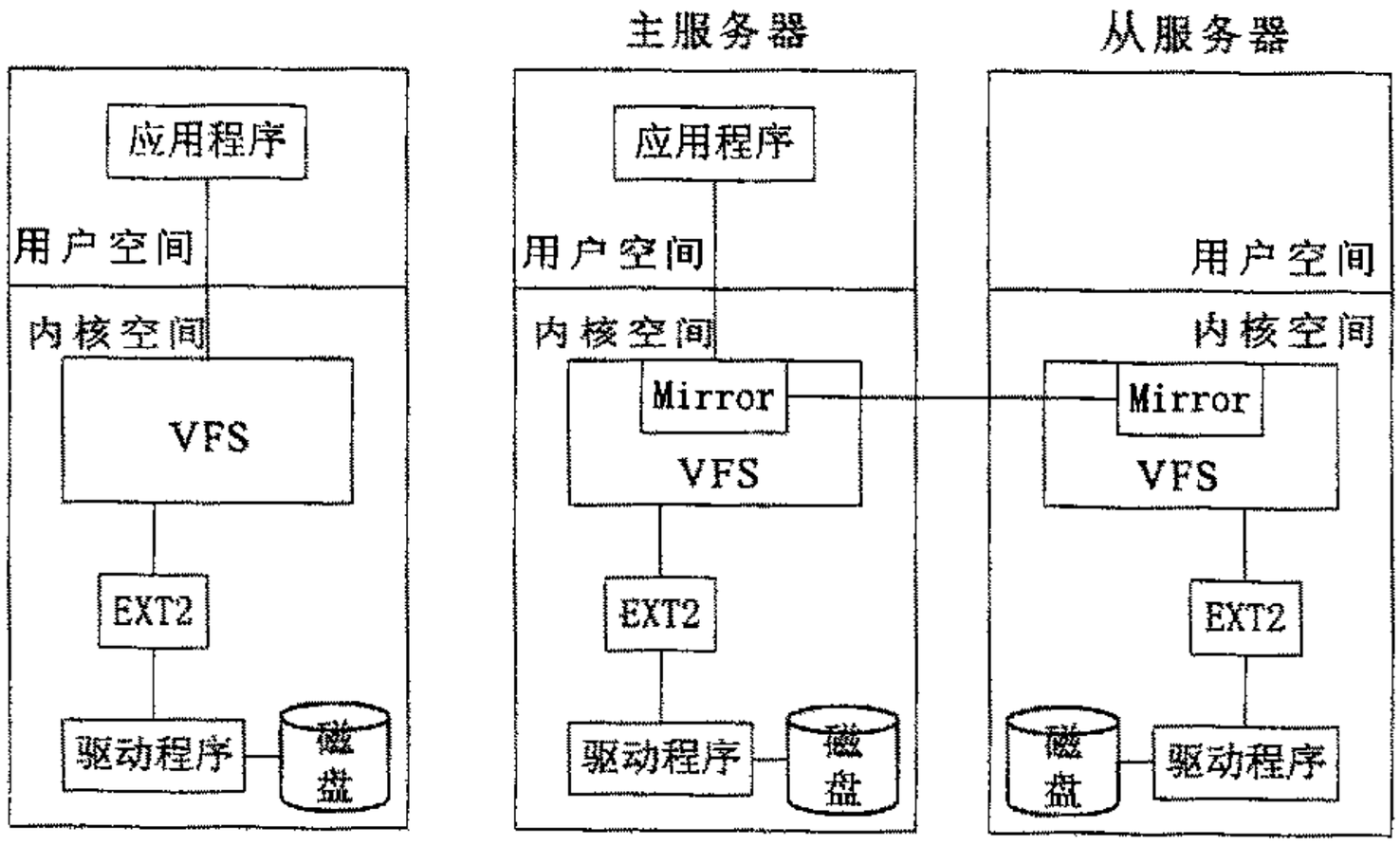
镜像算法基本上都只传送文件系统数据差值，这样可以减少数据的传输量。发现原系统和目标系统的文件差值主要有两种方法，一种方法是在服务器上运行一个守护进程，定期地扫描磁盘，寻找被更新的数据。还有一种方法是通过对原文件和目标文件作校验和分析[39]。研究表明，这些方法对大文件的读写可以获得较好的性能，对小文件的写性能比较差。并行文件系统的元数据文件通常都很小，而且经常被改写。考虑到这种特殊应用，本文采取一种基于Linux内核的磁盘镜像，同样它只传输改变的数据而不是全部数据，但它不需要周期性的扫描磁盘或者进行校验和分析，而且

具有很好的实时性。

4.1.1 实现原理

在 Linux 系统中，应用程序通过文件系统调用操作磁盘文件，实现不同计算机间的磁盘镜像，其实质就是实现不同计算机间的文件系统的镜像。而这可以通过对其中一台计算机的文件系统进行操作时，在另一台机器上也进行同样的操作来实现。比如在本机上建立一个文件，则在镜像机上也建立起一个同样的文件。

实现的方法是通过在两台机器的操作系统内核上插入一个镜像系统模块，当一台机器上的文件系统发生变化时，通过镜像系统模块，使相应的变化也同时在另一台机器上发生。未安装镜像系统时，Linux 系统的文件访问如图 4.1a 所示。应用程序通过文件系统调用接口访问文件，系统调用进入内核后，先由虚拟文件系统 VFS 进行处理，然后根据文件的类型由具体文件系统（如 Linux 的 EXT2）进一步处理，最后通过磁盘驱动程序与物理磁盘进行交互。安装镜像系统后，过程是类似的，但是系统调用进入内核后，先通过镜像系统 Mirror 的过滤，如果需要镜像则除了完成本地的操作外还要完成镜像操作。此时的文件访问如图 4.1b 所示。



a 标准 Linux 系统　　b 安装镜像系统后的 Linux 系统

图 4.1 Linux 系统的文件访问图

4.1.2 模块结构与关键算法

镜像系统核心模块包括过虑模块，网络模块和还原模块，如图 4.2 所示。除了核心模块外还应该包括其它一些模块如初始化模块等。

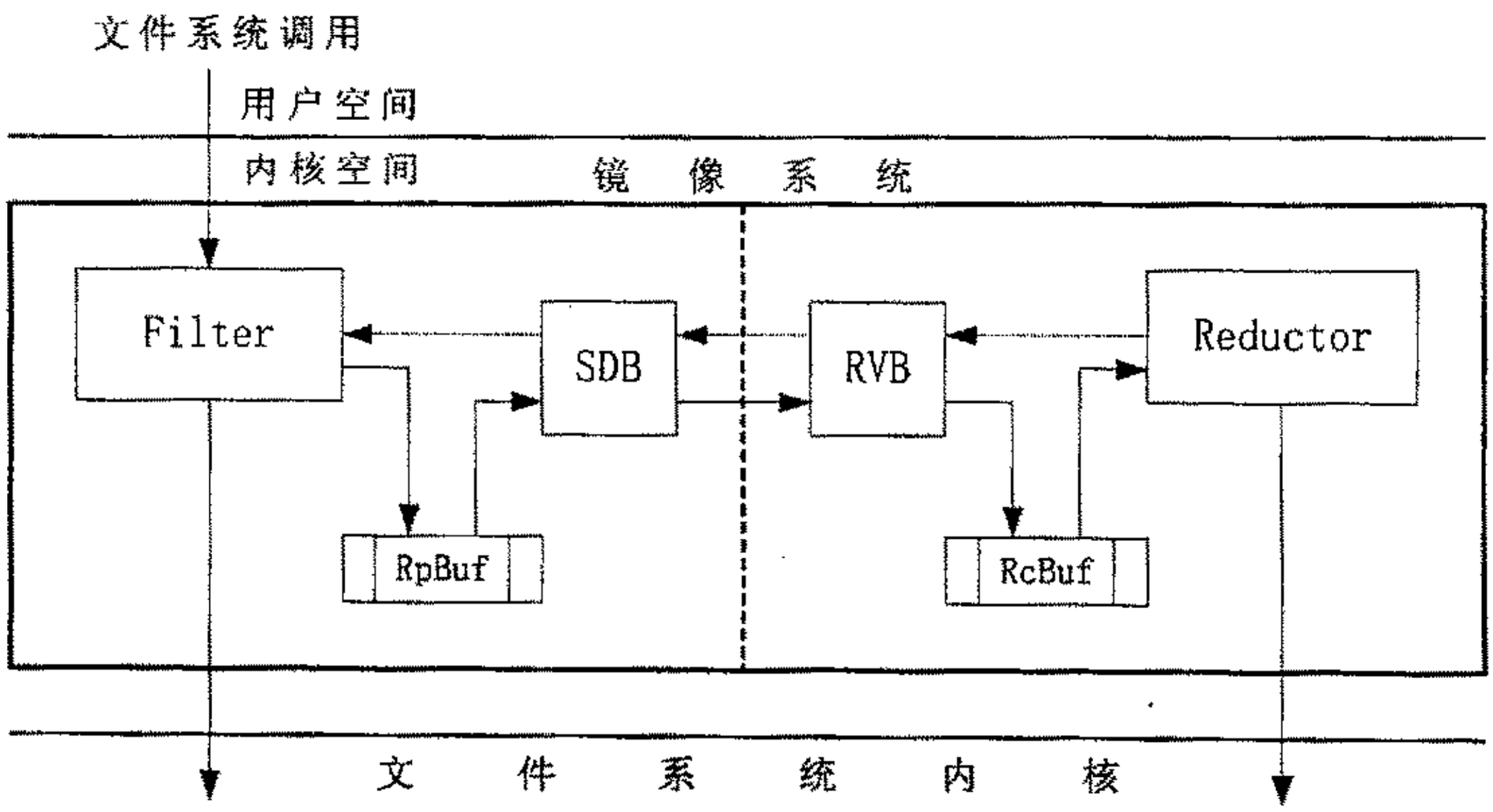


图 4.2 镜像系统核心模块示意图

过滤模块（Filter）：在主服务器上，当涉及到写文件操作时，它截获相关的文件系统调用，进行有选择的镜像。如果系统调用中的文件路径名包含在需要镜像的目录（用户根据需要设定）中时，就进行镜像；否则只执行正常的操作，不进行镜像。RpBuf是一个复制缓冲区，需要镜像的数据复制一份放在该缓冲区中，准备打包发送给从服务器。

网络模块（包括缓冲区发送模块SDB和缓冲区接收模块RVB）：SDB模块在主服务器上，负责发送RpBuf中的数据到从服务器。RVB模块在从服务器上，负责接收主服务器上传来的数据包并放到一个恢复缓冲区RcBuf中。

还原模块（Reductor）：在从服务器上，它的任务是当RcBuf中有数据时，就读取以数据包形式存放的数据，并还原成原来的系统调用参数，在从服务器上执行这个系统调用。

图4.3是实现镜像的算法流程图。在镜像算法中要考虑的是镜像分为同步镜像和异步镜像两种方式。同步镜像方式就是等待镜像数据已经发送到

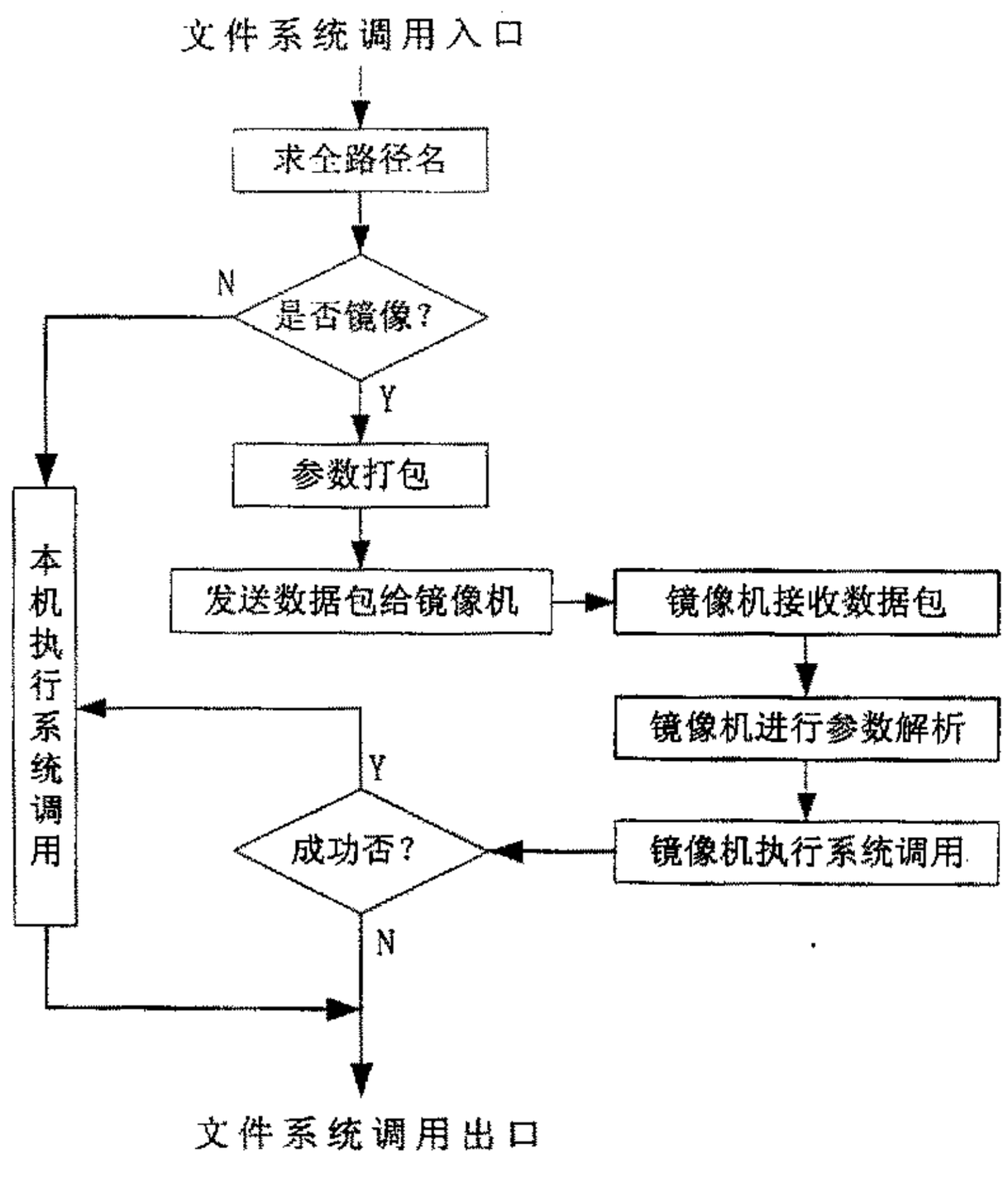


图 4.3 镜像算法流程图

从服务器，且从服务器执行相应的操作成功后，主服务器才执行操作，这样就保证主服务器和从服务器的文件无论何时总保持一致。在异步镜像方式中，主服务器不等从服务器收到数据，就进行操作。异步方式的 I/O 响应时间短，系统的效率高，但是有丢失数据的危险。

过滤系统调用时，只对写操作过滤，对读操作则不过滤，因为只有写操作才产生新数据。在 Linux 系统中，涉及到写文件系统的调用大概有十几个，它们是：mkdir，rmdir，creat，open，link，unlink，rename，write，truncate，ftruncate，chmod，fchmod，chown，fchown，lchown，utime（其中 open 只有在打开时有 O_CREAT 标志创建新文件，或者有 O_TRUNC 标志将现存文件的长度截短为 0 的时候，才改写文件系统的内容）。对这些系统调用过滤的实现方法就是用一个新的函数来取代系统调用表 sys_call_table[]里原来的那个函数，当然这个新的函数也应该完成原来的系统调用的功能，但还应该做一些额外的事情，包括判断是否镜像，如果

镜像则要提取参数，调用网络模块发送数据包等。

如何判断一个文件操作是否镜像比较关键，本文采取根据文件名判断的方法，因为总可以保证主从服务器上的文件名一致。对于其它的对象如inode 号或者打开文件描述符等，由于两机上资源的使用情况差异，就很难或者根本不可能保证一致。不仅希望显式的系统调用能起作用，而且希望象 cp、cat 这些命令工具也能起作用，例如在当前目录/home/bob/project中执行命令：cp file1 file2，其中的 file1、file2 通常给出的都是短文件名，判断镜像的时候我们希望文件名是/home/bob/project/file1，而不仅仅是file1。因此，首先要得到文件从根目录开始的全路径名，然后和用户设定的镜像目录 MIRROR_PATH 进行比较，如果 MIRROR_PATH 是该文件全路径名的一个前缀，表明文件在镜像目录中，需要镜像。判断镜像的算法描述如下：

```
int isMirror(const char *pathname){//根据全路径名判断是否镜像
    if (!strncmp(pathname, MIRROR_PATH, MIRROR_PATH_LEN))
        return 1; //如果全路径名以镜像目录为前缀则镜像
    return 0; // 否则不需要镜像
}
```

在上面的算法中，pathname 是文件的全路径名。求全路径名的算法如下：

```
char * get_fullpath(const char * filename){//filename 是短文件名
    int fd;
    struct file * file;
    char * pathname = NULL; //初始化全路径名指针
    //分配存放文件全路径名的空间
    pathname = kmalloc(MAX_PATHNAME, GFP_KERNEL);
    if ( ! pathname) goto out; //分配空间失败
    fd = sys_open(filename, O_RDONLY, 0); //打开文件得到文件句柄
    file = fcheck(fd); //调用内核算法 fcheck 得到文件句柄的 file 结构
    //如果 file 结构为空，跳过下面的执行块
    if (file) {
        struct dentry * d_entry = file->f_dentry; //文件目录项
        char * dir = d_entry->d_iname; //文件的路径分量名
        strcpy(pathname, dir); //拷贝文件路径分量名到全路径名中
```

```
        //从当前路径分量开始逐层得到全部路径分量，直到根'/'为止
        while (d_entry->d_parent && d_entry->d_parent != d_entry) {
            //分配一个临时空间，存放当前路径分量
            char * tmp = kmalloc(MAX_PATHNAME, GFP_KERNEL);
            if ( ! tmp ) goto out; //分配空间失败
            d_entry = d_entry->d_parent; //当前目录项指针指向父目录
            dir = d_entry->d_iname; //当前路径分量名
            strcpy(tmp, dir); //拷贝当前路径分量名到临时空间
            strcat(tmp, "/"); //在当前路径分量名前加'/'
            pathname = strcat(tmp, pathname); //添加到已得到路径的首部
        } //while
        pathname = pathname++; //去掉'//'前面的'/'
    } //if
out:
    return pathname; //返回全路径名
}
```

还原模块比较简单，在从服务器上建立一个内核线程，它相当于一个守护进程，负责接收并处理来自主服务器上的镜像请求和传送的数据。内核线程首先从接收缓冲区的数据包中解析出系统调用参数，包括系统调用的名称以及该系统调用自己所带的参数，然后利用这些参数执行该系统调用一遍。如果是同步镜像，则还需要把执行的结果发送给主服务器，以保证主、从服务器上两次系统调用结果的一致性。

4.1.3 性能分析

磁盘镜像系统必然会影响系统的性能，为了评价它对文件系统性能的影响程度，本文作了简单的测试。

1. 测试环境

---------- 主服务器配置 ----------

操作系统：Linux 7.1 发行版，内核 2.4.2-2

CPU：PIII933

内存：256M

网卡：100M，IP 地址：192.168.1.10

---------- 从服务器配置 ----------

操作系统：Linux 7.2 发行版，内核 2.4.7-10

CPU：PIII550

内存：128M

网卡：100M，IP 地址：192.168.1.201

2. 测试方法

测试是在主服务器上进行的，其方法是在安装镜像系统和没有安装镜像系统的条件下，在镜像目录中创建 1000 个文件，然后写入指定大小的数据，测试平均每秒钟写的字节数，也就是写带宽；然后从这些文件中读出指定大小的数据到缓冲区中，测试读带宽。测试程序是在基准测试程序 LMbench 中的小测试程序 lat_fs 基础上改写的。lat_fs 测试平均每秒钟创建和删除大小分别为 0，1k，4k，10k 的文件个数，其中的创建操作就包含写数据。这里对文件的大小作了改变，只进行小文件的测试（元数据文件的大小通常很小，例如 PVFS 元数据文件的大小只有 112 字节），同时添加了读测试，输出内容也作了相应的改变。

对于写操作，文件大小为 512 字节时测试结果如表 4.1 所示。连续进行了六次测试，计算其平均值。结果表明，安装镜像系统后的写带宽降低了 30%左右，或者说镜像系统所占的开销约为 30%。这是因为，对于写操作，不仅要完成本地的写，而且还要完成镜像写，需要将数据通过网络传送到镜像服务器。测试中发现，影响写带宽的因素主要是网络，也和服务器的负载程度有关。

表 4.1 文件大小为 512 字节时的写带宽

单位：字节/秒

安装镜像系统后	安装镜像系统前	测试次数
12109745	17964912	1
12343298	17977528	2
12069778	17952314	3
12385099	17728532	4
12851406	17877095	5
12439261	17864620	6
12366431	17894166	平 均

当减小文件的大小时，发现由于镜像损失的带宽比会增大。表 4.2 是文件大小为 256 字节时的写带宽测试。在这种情况下，镜像系统所占的开销约为 39%。在传送镜像数据时，除了传送文件的内容外，还要传送一定的头部作为控制字节，文件越小时，头部所占的比重就越大，因此损失的实际有用数据的带宽比也越大。

表 4.2 文件大小为 256 字节时的写带宽

单位：字节/秒

安装镜像系统后	安装镜像系统前	测试次数
6142035	8969867	1
5250205	8845888	2
5196914	8957313	3
5243753	8944794	4
5621432	8951049	5
5213849	8907446	6
5444698	8929393	平 均

对于读文件操作，文件大小分别为 512 字节和 256 字节的时候，测试结果如表 4.3 和表 4.4 所示，镜像系统所占的开销分别为 9%和 7%。因为读操作不镜像，不需要网络传送数据，因此此时的开销主要是由于安装镜像系统后对系统调用过滤引起的。此外还发现，改变文件的大小对读带宽几乎没有影响。

表 4.3 文件大小为 512 字节时的读带宽

单位：字节/秒

安装镜像系统后	安装镜像系统前	测试次数
150588235	172972973	1
156097561	168421053	2
151479290	169536424	3
156097561	169536424	4
155151515	172972973	5
156097561	170666667	6
154251954	170684419	平 均

表 4.4 文件大小为 256 字节时的读带宽

单位：字节/秒

安装镜像系统后	安装镜像系统前	测试次数
92753623	104918033	1
92753623	104918033	2
97709924	107563025	3
96969697	101587302	4
99224806	104065041	5
95522388	101587302	6
95822344	104106456	平 均

4.2 故障检测术

本文在 Linux 内核空间实现了心跳检测，其工作由两个内核线程 SendHeartbeatThread 和 RecvHeartbeatThread 来完成。正常工作时，两台服务器上的 SendHeartbeatThread 线程每隔一段时间（1 秒钟）就向对方发自己的心跳信息(UDP 包)，而 RecvHeartbeatThread 线程每收到一个对方结点来的心跳信息，就根据自己的时钟，将此结点的时间戳更新一次。一旦某个结点失效，对方结点就不会再收到来自它的心跳信息，当这个时间超过一定的间隔时间 (设为三个心跳时间)后，对方结点就认为它已经失效了。判断失效的条件为：

(jiffies – LastPeerHeartTime) > HEARTBEATDEADTIME

这里的变量 LastPeerHeartTime 为收到对方结点最后一次心跳的时间，是一个全局变量。jiffies 则是 Linux 内核中的一个全局量，是系统的当前时间。HEARTBEATDEADTIME 为设置的超时时间，定义为：

```
#define HEARTBEATDEADTIME    3*HZ    // 3 次 heartbeat 间隔时间
```

心跳数据包是 UDP 包而不是 TCP 包，这是因为 TCP 是一个面向连接的协议，当连接中断后需要重新连接，每次连接需要进行握手同步，会造成同步洪泛（SYN flood）问题。UDP 是无连接的协议，不会有这个问题。实现 UDP socket 时，可以是面向连接的，不过不象 TCP 那样需要进行三次握手，面向连接的 UDP socket 仅仅记录对方服务器的 IP 地址和端口就

立即返回了。当程序知道要多次发送数据包到对方结点时，面向连接的UDP socket能提高性能。

此外，为了避免被别的结点发心跳包欺骗服务器，通常采取数字签名技术，甚至考虑加密。在本文的实现中，只是简单的通过在发送的心跳包中设定一个特殊键值，收到数据包后比较这个键值来判断是否是对方结点发来的心跳包。通常在集群系统内部认为结点都是可以信任的，所以不考虑安全问题。

4.3 IP接管技术

1. 工作原理

在主、从服务器中使用两个网络接口，一个服务接口和一个备用接口。当主服务器对客户提供服务时，使用一个对客户公开的IP地址VIP。主服务器失效后，从服务器上的备用接口启动并接管VIP。这样，就可以使用与主服务器一样的IP地址继续对客户提供服务。当IP接管发生后，客户机还面临着这样一个问题：同一个IP地址对应着不同的MAC地址。因此，还需要接管MAC地址。在局域网内可以采取ARP欺骗技术来达到这个目的。

2. ARP协议与欺骗

IP数据包通过以太网发送时，以太网设备并不识别32位的IP地址，而是以48位以太网地址传输数据包的。因此，必须把IP目的地址转换成以太网网目的地址。地址解析为这两种不同的地址形式提供了映射：32位IP地址和48位MAC地址。ARP协议为IP地址到对应的MAC地址之间提供了动态映射。

ARP工作时，送出一个含有所希望IP地址的以太网广播数据包。目的主机，或另一个代表该主机的系统，以一个含有IP地址和以太网地址对的数据包作为应答。发送者将这个地址对高速缓存起来，以节约不必要的ARP通信。每个主机都有一个ARP高速缓存，它存放了最近IP地址到硬件地址之间的映射关系。该ARP表通常是动态的转换表，也就是说，会被主机在一定的时间间隔后刷新。这个时间间隔就是ARP高速缓存的超时时间。高速缓存中每一项的生存时间一般为20分钟。

在实现TCP/IP协议的网络环境下，一个IP数据包需要根据路由表来选择路由。IP包经过路由选择后，得到下一站路由器地址或目的主机的地

址，然后在 ARP 高速缓存中去寻找相对应 MAC 地址，如果没找到，则在局域网内广播一个 ARP 请求分组，该分组中带有下一站的 IP 地址或目的主机的 IP 地址。该局域网内的其它主机收到这个 ARP 分组后就会将分组中携带的 IP 地址与自己的地址进行比较，如果相同，则将自己的 48 位 MAC 地址包装在 ARP 应答包中，回给请求的主机。发出请求的主机收到应答包后，就将这个新的 IP 地址与 MAC 地址映射关系添加到自己的 ARP 缓存中，然后根据这个 MAC 地址发 IP 数据包。

分析 ARP 的工作原理后，实现 ARP 欺骗就很容易了。可以让接管主机发送一个类型为 ARP_REPLY 的免费 ARP 响应包（TCP/IP 协议并没有规定必须在接收到 ARP_ECHO 后才可以发送响应包），并指定免费 ARP 包中的<源 IP 地址，目的 IP 地址，源 MAC 地址，目的 MAC 地址>四元组，其中的源 IP 地址为被接管主机的 IP 地址，源 MAC 地址为接管主机的 MAC 地址。这样就可以通过虚假的 ARP 响应包来修改客户机上的动态 ARP 缓存，以后客户机发送数据包到原来的 IP 地址的时候，实际上是发送到接管主机上。下面给出一个 ARP 欺骗的例子。

考虑同一网段的三台主机：

A: IP 地址 192.168.1.1 硬件地址 AA:AA:AA:AA:AA:AA

B: IP 地址 192.168.1.2 硬件地址 BB:BB:BB:BB:BB:BB

C: IP 地址 192.168.1.3 硬件地址 CC:CC:CC:CC:CC:CC

假设主机 A（客户机）连接到主机 B（主服务器）上，现在主机 C（从服务器）已经接管主机 B 的 IP 地址。接管后，要让主机 A 相信主机 C 就是主机 B，如果仅仅把主机 C 的 IP 地址改成和主机 B 的一样：192.168.1.2，是不能可靠工作的。因为主机 A 的 ARP 高速缓存中 IP 地址 192.168.1.2 对应的 MAC 地址没有改变，还是主机 B 原来的硬件地址 BB:BB:BB:BB:BB:BB。这个时候需要用 ARP 欺骗的手段让主机 A 把自己的 ARP 缓存中的关于 192.168.1.2 映射的硬件地址改为主机 C 的硬件地址 CC:CC:CC:CC:CC:CC。可以让主机 C 发送一个 ARP_REPLY 的响应包给主机 A，指定 ARP 包中的源 IP 地址为主机 B 的 IP 地址，源 MAC 地址为主机 C 的 MAC 地址，这样通过虚假的 ARP 响应包来修改主机 A 上的动态 ARP 缓存。

3．主要实现函数

IP 接管的主要实现函数如下：

```
void ip_takeover(void){
    int err = setup_if(BACKUPDEV, VIRTUALSERVERADDR,
                       in_aton("255.255.255.0"));
    if (!err) {
        create_kernel_thread((int (*)(void *))arp_send_thread_fn,
                             &arp_send_thread);
    }
}
```

其中，setup_if()函数激活备用网络接口 BACKUPDEV，并将虚拟 IP 地址 VIRTUALSERVERADDR 绑定在它上面。create_kernel_thread()函数创建一个内核线程 arp_send_thread，然后执行 arp_send_thread_fn()函数，发送 ARP 免费包。

免费 ARP 包的目的 IP 地址和源 IP 地址都是双元服务器的虚拟 IP 地址，以太网目的地址是所在局域网的广播地址（全 1 地址），以太网源地址是绑定在这个虚拟 IP 地址的网卡的 MAC 地址。操作字段的操作类型为 ARP_REPLY。

发送免费 ARP 包的函数如下：

```
void send_gratuitous_arp(void)
{
    u32 vip = VIRTUALSERVERADDR;
    struct net_device *dev = dev_get_by_name(BACKUPDEV);
    arp_send(ARPOP_REPLY, ETH_P_ARP, vip, dev, vip, NULL,
             dev->dev_addr, NULL);
}
```

调用 IP 接管模块的时机有两个：

（1）系统启动的时候，由主服务器来调用。谁是主服务器可以根据模块安装参数由用户来指定。

（2）当监控模块检测到当前的主服务器失效的时候，由从服务器来调用。

4.4 恢复技术

在双元服务器系统中，当主服务器失效后由从服务器接管过来，需要

重新启动服务程序，如果客户访问以前的文件，由于已经对元数据采取了镜像，可以利用镜像目录中的元数据提供服务。对于新建立的文件则会产生新的元数据，写入到从服务器的元数据目录中。另一方面，当发生一个结点失效（无论是主服务器失效还是从服务器失效）进入单机模式的时候，新生成的元数据无法镜像，如果在此期间有元数据访问，就会存在两机镜像目录中的元数据不一致，失效结点加入到系统中来时需要进行一次额外的元数据同步过程。恢复工作主要是针对这一点，本文采取的策略比较简单，使用了一个镜像系统记录文件 MIRROR_SYSLOG 和内核恢复线程 MIRROR_RECOVER_THREAD。监控模块检测到对方结点失效发生时，就启动 MIRROR_RECOVER_THREAD 线程，将镜像失败的元数据信息写入 MIRROR_SYSLOG 文件。当失效结点修复后重新加入到系统中来，监控模块会检测到，这时候通知 MIRROR_RECOVER_THREAD 线程，将记录在 MIRROR_SYSLOG 文件中的元数据信息一次性的镜像到恢复后的服务器，从而完成同步过程。

4.5 小结

在具有容错能力的双元服务器系统中，通过元数据镜像产生副本冗余，保证了元数据的可靠性。通过对文件系统调用内核函数的过滤，当本机上更新文件系统时，使这一更新也在镜像机上同步发生，从而实现镜像，这个过程是完全自动和对用户透明的。由于不需要专用的硬件，用软件方法来实现镜像，减少了用户的成本。讨论了镜像系统的模块组成、关键算法及其实现方法。镜像实现了元数据容错的好处，由于镜像的开销又不可避免的引起系统性能的下降。性能测试表明对于小写操作，镜像的开销约占 30%左右；对于只读操作，镜像开销约占 8%左右。双元服务器系统的另一个关键技术是通过故障屏蔽保证元数据服务的连续性。故障屏蔽技术包括基于心跳的检测和监控技术、IP 接管技术和恢复技术等，所有的技术都是在 Linux 内核空间实现，对用户和应用程序来说都是透明的，现有的服务器应用程序可以不加任何修改就能纳入到本系统中，减少了应用程序升级时的维护开销。用户通过对内核模块的随时加载和卸载，可以方便的进行控制和管理。

5 一种寄生式元数据存储管理方法

为了提高对元数据操作的性能，本文提出了一种寄生式的元数据存储管理方法，将并行文件系统的元数据寄生在本地文件系统内核中，由内核进行统一的管理。为了保证对系统的兼容性，通过增加系统调用及其接口实现对寄生元数据的操作。给出了一个基于 PVFS 元数据管理的实现例子。

5.1 基于文件的元数据存储

在并行文件系统中，元数据的存储除了用数据库外，基本上都是基于本地文件系统文件实现的，即用一个元数据文件来存储相应文件的元数据信息。用文件存储元数据编码少、实现简单。相对于用户数据来说，并行文件系统中的元数据通常都是小量的，因此这些元数据文件也很小。但是如果并行文件系统的文件非常多，则元数据文件也非常多，因为它们是一一对应的。这样就造成系统中大量的小文件。

另一方面，并行文件系统的元数据访问非常频繁，因为对数据的访问必须以元数据的访问为前提，而且经常要多次访问元数据。在基于文件的存储方式中，一次元数据的存取往往需要多次文件操作，最终需要通过多次系统调用才能完成。以 PVFS 为例子，创建一个文件所引起的元数据操作需要上十次的系统调用，包括一次写文件、多次读文件和 open、close、stat 等操作。元数据文件本身也是文件，因此它也有自己的元数据，对元数据文件的操作实际上应该包含对它自己的元数据操作。对大量小文件的操作尤其是读写操作会影响系统的效率。在并行文件系统中，影响元数据访问效率的因素除了网络延迟外，主要就是元数据服务器处理客户请求的延迟。为了提高元数据操作的性能进而提高元数据管理的效率，本文提出了一种寄生式元数据存储管理方法[40]。

5.2 寄生式元数据存储管理方法

对于单机文件系统来说，元数据信息的存储和管理都是由操作系统内核来完成的。以 UNIX 类操作系统为例，元数据信息存储在磁盘上，主要

是索引节点(inode 结构)和目录项(dentry 结构)这两个数据结构。在 inode 数据结构中记录着文件在存储介质上的位置与分布以及一些文件的属性等信息。在 dentry 数据结构中存放着文件名。通过文件系统目录树这样一种机制，根据一个文件的文件名就可以在磁盘上找到该文件的索引结点，并在内存中建立起代表该文件的 inode 结构。每个索引节点有一个唯一的索引节点号，内核用它来建立哈希函数以提高查找的效率。应用程序获得元数据可以通过 stat 系统调用，元数据的更新则在相应的每一条系统调用内由内核完成。譬如写文件会改变文件的大小，每一次写操作完成后，write 系统调用就会修改代表文件大小的元数据信息：内存中 inode 结构的 i_size 分量，这样的 inode 是“脏”的，在一定的时机这些“脏”的 inode 会写到磁盘上[41]。

本文提出的并行文件系统的寄生式元数据存储管理，正是建立在单机文件系统的元数据管理内核机制上的。通过对有关的数据结构和内核进行扩充，生成新的内核系统。在这种系统中，除了本地文件系统的元数据外，还有“寄生”的并行文件系统的元数据，它们在内核中共存，通过共同的管理机制如缓存管理、查找算法等来实现管理。和单机文件系统一样，通过系统调用来完成对元数据的操作。给出相应的访问接口，对用户来说系统就是透明的，内核则可以区分是本地文件系统的元数据还是并行文件系统的元数据，并采取相应的操作。下一节将针对 PVFS 的元数据管理，给出这种寄生式元数据存储管理的具体实现方法。

5.3 PVFS 寄生式元数据存储管理实现

5.3.1 元数据结构

元数据结构是 inode 和 dentry 这两个数据结构，但 PVFS 文件系统文件要增加一些元数据信息（主要是代表文件分片在集群中结点位置信息的几个参数：第一个分片的结点号 base，分片个数 pcount 以及分片的大小 ssize），因此要对现有的元数据结构进行扩充。为方便起见，本文通过对 inode 数据结构进行改造达到这个目的。Linux 的本地文件系统是 EXT2。EXT2 文件系统在磁盘上的 inode 数据结构是 ext2_inode，其中有一个指针数组项 i_block[EXT2_N_BLOCKS]，指向文件数据块的磁盘位置。由于 PVFS 的元数据文件并不包含数据块的磁盘位置信息，因此将这个指针数

组改造成一个联合结构：

```
union {
    struct {
     __u32   i_block[EXT2_N_BLOCKS];
    } local;
    struct {
     __u32   i_p_parameter1;
     __u32   i_p_parameter2;
     __u32   i_p_parameter3;
     __u32   i_p_parameter4;
     __u32   i_p_parameter5;
     __u32   i_p_reserved[10]; //EXT2_N_BLOCKS = 15
    } pfs;
} fcat;
```

如果文件是本地文件系统文件，就用 local 结构的 i_block 数组存放文件数据块的指针；如果文件是 PVFS 文件系统文件，则用 pfs 结构存放它在集群中的位置信息：pcount，base，ssize 等。本文中不是直接扩充 inode 结构，其原因是，可以保持磁盘 inode 结构 128 字节大小不变，不会浪费额外的磁盘空间（磁盘上的物理块通常是 512 字节，刚好存放 4 个 inode 结构。由于 inode 数据结构通常不能跨块存放，因此若增加其大小，则平均每块都会浪费若干空间）。

inode 结构不仅在磁盘上有，在内存中也有。因此，内存中的 inode 数据结构也要做相应的扩充。这样一来，扩充后的 inode 数据结构和 dentry 结构就全面的记录了 PVFS 文件系统的元数据。

5.3.1 元数据操作功能实现

本文实现的这种寄生式元数据存储管理方法，是由内核来完成 PVFS 文件系统元数据操作，但对用户应该是透明的，因此需要提供外部功能服务接口。而系统调用正是应用程序和操作系统内核之间的功能接口，是用户程序和内核进行交互的低层方法。系统调用的目的是使得用户可以使用操作系统提供的有关设备管理、文件系统和进程控制以及存储管理等方面

的功能，而不必了解系统程序的内部结构和有关硬件细节，从而起到减轻用户负担和保护系统的作用。为了保持与原系统的兼容性，采用增加系统调用的方法来达到管理 PVFS 元数据的目的。

在 Linux 系统中，系统调用是作为一种异常类型来实现的。通过执行相应的机器代码指令产生异常信号。产生中断或异常的重要效果是系统自动将用户态切换为核心态，并对它进行处理。这就是说，执行系统调用异常指令时，自动地将系统切换为核心态，并安排异常处理程序的执行。

Linux 用来实现系统调用异常的指令是：int $0x80，这一指令使用中断向量号 128（即 16 进制的 80）将控制权转移给内核。为达到在使用系统调用时不必用机器指令编程，在标准的 C 语言库中为每一系统调用提供了一段短的子程序，完成机器代码的编程工作。事实上，机器代码段所要做的工作只是将送给系统调用的参数加载到 CPU 寄存器中，接着执行指令 int $0x80。运行系统调用的返回值将送入 CPU 的一个寄存器中，标准的库子程序取得这一返回值，并将它送回用户程序。

为使系统调用的执行成为一项简单的任务，Linux 提供了一组预处理宏指令，它们可以用在程序中。宏指令取一定的参数，然后扩展为调用指定的系统调用的函数。宏指令格式类似于_syscallN（parameters），其中 N 是系统调用所需要的参数数目，而 parameters 则用一组参数代替，分别说明系统调用返回值的类型、系统调用名以及系统调用本身所带的参数类型和名字。一旦宏指令用特定系统调用的相应参数进行了扩展，得到的结果是一个与系统调用同名的函数，就可以在用户程序中执行这一系统调用。

添加系统调用的方法很简单，其步骤为：

（1）添加内核源代码；

（2）连接新的系统调用；

（3）重建新内核；

（4）启动新内核，使用新的系统调用。

为完成 PVFS 的元数据操作，增加了十个系统调用，如表 5.1 所示。

5.3.2 性能分析

1. 测试环境与结果

本文用延迟作为指标，测试了寄生式元数据存储管理方法的性能，并与 PVFS 所采用的基于文件的管理方法进行了比较。测试是在服务器端进

表 5.1 增加的系统调用

系统调用	功　　能
pstat	获得 PVFS 文件系统文件的元数据
pcreat	建立代表 PVFS 文件系统文件的 dentry 结构
punlink	删除代表 PVFS 文件系统文件的 dentry 结构
paccess	验证是否具有存取 PVFS 文件系统文件的许可权
putime	改变 PVFS 文件系统文件的存取时间和修改时间
pchmod	改变 PVFS 文件系统文件的存取模式
pchown	改变 PVFS 文件系统文件的文件主
prename	改变 PVFS 文件系统文件的文件名
pmkdir	创建 PVFS 文件系统目录
prmdir	删除 PVFS 文件系统目录

行的。为了避免修改 PVFS 服务器源代码接口，采取模拟测试，即用自己写的程序来模拟 PVFS 的行为。节点配置环境为：

操作系统：Linux7.2 发行版，内核在 2.4.18 上重建

CPU：PIII933

内存：256MB

测试的内容包括 mkdir，creat，stat，unlink 和 rmdir 的平均时间，测试结果如表 5.2 所示，其中：

比值=文件管理方式的延迟/寄生管理方式的延迟

2．性能评价

从表 5.2 中可以看出，寄生式元数据存储管理比基于文件的元数据存

表 5.2　两种方法测得的延迟时间对比

单位：微秒

内　容	文件管理方式	寄生管理方式	比值
mkdir	345	81	4.3
creat	328	45	7.3
stat	124	15	8.3
unlink	141	28	5.0
rmdir	301	60	5.0

储管理，延迟减小了很多，性能提高了大约 5～8 倍。值得注意的是 stat 的性能提高了 8.3 倍，在 PVFS 的元数据管理中，最主要的就是这一操作。因此，这个结果是比较满意的。

下面以创建文件的 creat 为例，分析这两种方法的性能为什么会有这么大的差异。在基于文件的存储管理方法中，创建一个文件时，元数据服务器会创建一个元数据文件。在创建元数据文件之前，首先要检查是否有对目录的写权限，由于目录的元数据信息也是用一个文件来记录的，因此先打开目录元数据文件，从中读出元数据，与请求者的身份进行比较。对于每层目录都要进行这样的操作。验证通过后，在指定目录下创建一个同名的元数据文件，然后打开该文件，用 fstat 调用获得元数据文件的 inode 号等，然后连同请求者的 uid、gid 和文件的创建时间等元数据信息写入到元数据文件中。从这个过程可以看出，creat 涉及到多次文件的读、一次文件写和 fstat 等操作。而用寄生式元数据存储管理方法，就像本地文件系统一样，creat 只需要一条系统调用 pcreat 就完成了。对于其它的操作也是一样。因为 PVFS 原来的元数据管理是建立在文件操作基础上，由用户空间的应用程序完成的，一次操作涉及多次系统调用；而本文的寄生式元数据存储管理，所有的元数据操作都是在内核空间中完成的，一次操作只需一次系统调用的开销，效率自然要高些。

5.4 小结

并行文件系统的元数据通常是利用元数据文件来存储，对元数据的访问转化为对元数据文件的读写操作。这样做的好处是实现起来方便，也很自然。但是元数据的访问却变得不直接了，一次元数据访问通常需要多次文件读写，最终要通过多次系统调用才能完成。元数据文件一般都很小，元数据的访问又很频繁，对大量小文件的频繁读写会对服务器的性能造成比较大的影响。为了提高元数据服务器的处理效率，本章提出了一种寄生式元数据存储管理方法，将并行文件系统文件的元数据寄生在本地文件系统内核中，按照本地文件系统的元数据管理方式由内核完成操作。为了保证对现有系统的兼容性，通过增加系统调用来实现对寄生元数据的操作，这样对元数据的一次访问只需要一次系统调用的开销。针对 PVFS 的元数据管理，在 Linux 文件系统内核基础上实现了寄生式元数据存储管理。通过对内核中的 inode 数据结构进行扩充改造，将 PVFS 的元数据寄生在其

中，并增加了十余个系统调用来完成对 PVFS 元数据的操作。对采取寄生式元数据存储管理的 PVFS 进行性能测试，与原来基于文件操作的方式进行比较，结果表明元数据操作性能提高了大约 5～8 倍。

6 结束语

随着高性能微处理器、高速网络等的出现，以廉价而通用的 PC 机或工作站构成的集群系统越来越得到用户的青睐，也引起了集群技术的迅猛发展。集群文件系统是集群的一个重要组成部分，也是集群技术研究的一个热点。由于集群文件系统中的文件数据是分散存储在各个结点上的，对文件的存取首先要用元数据来定位，元数据的管理就成为管理数据的一个关键。

并行文件系统元数据管理的一个重要方面是保证它的可靠性。在使用过程中会出现主机或者磁盘失效的情况，造成数据部分或者完全丢失。通过对元数据的容错，当失效事件发生时，能够保证元数据不丢失，在出现灾难的时候可以恢复。另一方面，当服务器由于硬件故障失效后，系统提供的服务也就中断了。如果重新启动机器和服务器，这段时间内客户必须等待。为了提供连续的服务，系统必须具有屏蔽故障的能力。本文设计了一个具有容错能力的双元服务器系统来解决这两个问题。通过对元数据的实时镜像产生副本，实现了元数据的容错；通过采取一份备份服务器，当主服务器失效后，从服务器接管其服务程序，实现了服务的连续性。该系统的设计兼顾了集中式元数据管理和分布式元数据管理的优点，实际上是对它们二者的扬长避短，既消除了集中式管理的单一失效点问题，又避开了分布式管理的一致性维护设计与开销。对设计者来说大大简化了系统的复杂性，易于实现。对用户来说保证了比较高的可靠性和可用性，易于管理和维护。

并行文件系统的元数据具有数据量小、访问频繁的特点，应采取有效的备份策略和方法。一些现有的备份工具或者代价昂贵，需要采取专用的硬件设备；或者实时性比较差，不能满足要求。通常的备份算法通过磁盘扫描或者对文件系统做校验和分析，对大文件的读写性能比较好，却不适合元数据这种特定的应用。本文提出一种基于Linux内核的镜像技术，通过在操作系统中插入内核模块对文件系统调用过滤达到实时镜像目的。由于完全由软件方法实现，用户成本低。应用程序通过系统调用接口访问文件

系统，因此对用户是透明的。

通过故障屏蔽实现对元数据服务的连续性。用基于心跳的原理实现了故障检测；通过使用服务器虚拟IP地址及其接管技术，实现了元数据服务的接管；用基于类似于日志的系统记录文件及其内核监督线程，实现了故障恢复。故障屏蔽过程是完全自动的和透明的，不需要管理员的干预或客户手动从新连接。系统对现有的服务器应用程序是透明的，即不需要对它做任何修改就可以应用到该系统中，节省了应用程序升级时的维护开销。

元数据管理的另一个重要方面是要保证它的性能。通常的方法是采取元数据缓存技术，很多人在这方面做了研究。本文从另外的角度做了有意义的研究，提出了一种寄生式元数据存储管理方法来提高元数据管理的效率。在通常的基于文件存储的元数据管理中，对元数据的一次访问要多次操作文件，最终需要多次系统调用才能完成，因此影响了处理的效率。在本文的寄生式元数据存储管理方法中，并行文件系统的元数据寄生在本地文件系统内核中，由内核完成操作，一次元数据访问只需要一次系统调用的开销。为了保持系统的兼容性，采取增加系统调用的方法达到操作寄生元数据的功能。针对 PVFS 的元数据管理，在 Linux 文件系统内核基础上实现了这种方法，对其性能进行测试，并与原来基于文件操作的方式进行比较，结果表明元数据操作性能提高了大约 5～8 倍。

下一步的工作包括：

1．对双元服务器系统体系结构的进一步改进。由于只有主服务器对客户提供服务，从服务器对客户来说是没有发挥作用的，如果使主从服务器同时对客户提供服务，就可以进行负载平衡，进一步提高性能。为了进一步提高冗余和可靠性，还可以将系统推广到多机情况下。

2．通过从多方面实现冗余来提高系统可靠性。现在的系统只是对服务器实现了冗余，是一种大粒度的容错单元。实际上，失效的形式是各种各样的，包括网络的失效、接口的失效、软件的失效、进程的失效等等，都可以考虑通过冗余来实现容错。

3．进一步研究提高元数据性能的方法。元数据具有数据量小、访问频繁等特性，而小文件的读写往往性能比较差。如何提高元数据的访问性能是一个值得研究的问题。

致 谢

本论文是在我的导师庞丽萍教授的悉心指导下完成的。非常感谢庞老师在这三年对我研究工作的指导以及生活上的关心和帮助。庞老师在学术上的高深造诣、学风上的严谨求实、教育上的孜孜不倦以及工作上一丝不苟的精神都使我受益匪浅。

感谢实验室主任金海教授为我提供良好的科研环境和带来先进的研究课题，他对计算机科学发展前沿的敏锐洞察力大大拓宽了我的眼界。感谢李胜利教授和石柯副教授对论文提出的修改意见，他们在科学研究上的执着和默默奉献的精神永远值得我学习。感谢韩宗芬教授、章勤副教授在学习和生活上给予我的关怀。

感谢课题组组长徐婕博士和岳建辉师兄两年里对我的帮助，和她们的讨论使我受益颇多。感谢林运章、程斌、许俊、杨俊杰、唐唯、秦航等课题组内其他同学对我的帮助，和他们共事使我很开心。感谢陈宝利、易川江、唐丹等同学在 Linux 方面对我的帮助。感谢研究生期间我的室友李运发、顾海波、董兴昌、张西刚以及同学王志平、陈勇、周润松、丁俊民等对我学习和生活上的帮助。

感谢老同学谭谦仁、何斌、沙立华等在生活中给予我不断的鼓励，感谢我的老师卢艳玖、彭绍先多年来对我默默的帮助。特别感谢我以前的单位领导王旭高级工程师，她对我的谆谆教导和培养使我终生受益，对我物质上和精神上的支持，使我永远难忘。最后，感谢我平凡而伟大的父母，我欠他们的实在太多了，父母的养育之恩永远也报答不完。

参考文献

[1] Rajkumar Buyya 编. 高性能集群计算: 结构与系统（第一卷）. 第 1 版. 郑纬民, 石威, 汪东升等译. 北京: 电子工业出版社, 2001. 6~31

[2] Ibrahim F. Haddad. PVFS: A Parallel Virtual File System for Linux Clusters. Linux Journal, 2000, 2000(80): 5~12

[3] Jaechun No, Rajeev Thakur, Alok Choudhary. High-performance scientific data management system. Journal of Parallel and Distributed Computing, 2003, 63(4): 434~447

[4] Rajesh Bordawekar, Steven Landherr, Don Capps, et al. Experimental evaluation of the Hewlett-Packard Exemplar file system. ACM SIGMETRICS Performance Evaluation Review, 1997, 25(3): 21~28

[5] Jeff Ballard. NFS: hunting for a cross-platform file system. Network Computing, 1998, 9(12): 101~104

[6] M. Satyanarayanan. The evolution of Coda. ACM Transactions on Computer Systems (TOCS), 2002, 20(2): 85~124

[7] Thomas E.Anderson, Michael D.Dahlin, Jeanna M.Neefe, et al. Serverless network file systems. ACM Transactions on Computer Systems, 1996, 14(1): 41~79

[8] Jaechun No, Rajeev Thakur, Alok Choudhary. High-performance scientific data management system. Journal of Parallel and Distributed Computing, 2003, 63(4): 434~447

[9] Nils Nieuwejaar, David Kotz. The Galley Parallel File System. Parallel Computing, 1997, 23(4): 447~476

[10] Jianyong Wang, Zhiwei Xu. Cluster file systems: a case study. Future Generation Computer Systems, 2002, 18(2002): 373~387

[11] 贺劲, 徐志伟, 孟丹等. 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, 39(2): 129~135

[12] Jose Renato Santos, Richard R. Muntz, Berthier Ribeiro-Neto.

Comparing random data allocation and data striping in multimedia servers. ACM SIGMETRICS Performance Evaluation Review, 2000, 28(1): 44~55

[13] G.Ganger, M.McKusick, C.Soules, et at. Soft Updates: A Solution to the Metadata Update Problem in File systems. ACM Transactions on Computer Systems, 2000, 18(2): 127~153

[14] Gregory R. Ganger, Dawson R. Engler, M. Frans Kaashoek, et al. Fast and flexible application-level networking on exokernel systems. ACM Transactions on Computer Systems (TOCS), 2002, 20(1): 49~83

[15] Li-Chi Feng, Ruei-Chuan Chang. Using Asynchronous Writes on Metadata to Improve File System performance. J.SYSTEMS SOFTWARE, 1999, 35(95): 43~54

[16] Mark Roantree. Metadata management in federated multimedia systems. Australian Computer Science Communications, 2002, 24(2): 147~155

[17] Felix C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Computing Surveys (CSUR), 1999, 31(1): 1~26

[18] Faith Fich, Eric Ruppert. Hundreds of impossibility results for distributed computing. Distributed Computing, 2003, 16(2): 121~163

[19] Miguel Castro, Rodrigo Rodrigues, Barbara Liskov. BASE: Using abstraction to improve fault tolerance. ACM Transactions on Computer Systems (TOCS), 2003, 21(3): 236~269

[20] R. Guerraouim, A. Schiper. Software-Based Replication for Fault Tolerance. IEEE Computer Journal, 1997, 30(4): 68~74

[21] T. W. Page, R. G. Guy, J. S. Heidemann, et al. Perspectives on optimistically replicated, peer-to-peer filing. Software—Practice & Experience, 1998, 28(2): 155~180

[22] E. N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, et al. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys (CSUR), 2002, 34(3): 375~408

[23] 汪东升，郑纬民． 高可用集群计算．小型微型计算机系统，2000, 21(11): 26~29

[24] Zhang Youhui, Wang Dongsheng. A checkpoint-based high availability run-time system for Windows NT clusters. ACM SIGOPS Operating Systems Review, 2002, 36(2): 12~18

[25] F. Pérez, J. Carretero, F. García, et al. Evaluating ParFiSys: a high-performance parallel and distributed file system. Journal of Systems Architecture: the EUROMICRO Journal, 1997, 43(8): 533~542

[26] Evans, Matthew. FTFS: The Design of a Fault Tolerant Distributed File-System: [Undergraduate Honor's thesis]. Lincoln: University of Nebraska, 2000.

[27] TZUNG-SHI CHEN, CHIH-YUNG CHANG, JANG-PING SHEU, et al. A Fault-Tolerant Model for Replication in Distributed-File Systems. Proc.Natl.Sci.Counc.ROC(A), 1999, 23(3): 402~410

[28] Ling Zhuo, Cho-Li Wang, Francis C. M. Lau. Document replication and distribution in extensible geographically distributed web servers. Journal of Parallel and Distributed Computing, 2003, 63(10): 927~944

[29] Ricardo Jiménez-Peris, M. Patiño-Martínez, Gustavo Alonso, et al. Are quorums an alternative for data replication? ACM Transactions on Database Systems (TODS), 2003, 28(3): 257~294

[30] Roberto Baldoni, Carlo Marchetti. Three-tier replication for FT-CORBA infrastructures. Software—Practice & Experience, 2003, 33(8): 767~797

[31] R. Guerraoui, S. Frolund. Implementing E-Transactions with Asynchronous Replication. IEEE Transactions on Parallel and Distributed Systems, 2001, 12 (2): 133~146

[32] 庞丽萍，何飞跃，岳建辉等. 并行文件系统集中式元数据管理高可用系统设计. 计算机工程与科学，录用编号: 23383

[33] 于义科，严永福. 高性能可伸缩LVS集群的研究和实践. 江西农业大学学报, 2001, 23(5): 75~78

[34] Guang Tan, Hai Jin, Liping Pang. Layer 4 Fault Tolerance: Reliability Techniques for Cluster System in Internet Services. Lecture Notes in Computer Science, 2002, 2326(2002): 60~60

[35] Kai Hwang, Hai Jin, Roy S.C. Ho. Orthogonal striping and mirroring in distributed RAID for I/O-centric cluster computing. IEEE Transactions

on Parallel and Distributed Systems, 2002, 13(1): 26~44

[36] Miguel Castro, Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS), 2002, 20(4): 398~461

[37] 罗忠海，刘心松，陈勇． 基于通用计算机网络实现磁盘镜像．计算机学报，1998, 21(10): 881~889

[38] 李翌，吕光宏．Windows NT下的双机磁盘镜像系统的原理及实现．四川大学学报，2000, 32(6): 100~103

[39] Edward Swierk, Emre Kiciman, Nathan C. Williams, et al. The Roma personal metadata service. Mobile Networks and Applications, 2002, 7(5): 407~418

[40] 庞丽萍，何飞跃，徐婕等．PVFS 寄生式元数据管理的设计与实现．计算机工程，录用编号: 34824

[41] 毛德操，胡希明．LINUX内核源代码情景分析（上册）．第1版．杭州：浙江大学出版社，2001．415~687

附录 1　攻读学位期间发表论文目录

[1] 庞丽萍，何飞跃，岳建辉，徐婕. 并行文件系统集中式元数据管理高可用系统设计. 计算机工程与科学，录用编号：23383. 署名单位：华中科技大学

[2] 庞丽萍，何飞跃，徐婕，岳建辉. PVFS 寄生式元数据管理的设计与实现. 计算机工程，录用编号：34824. 署名单位：华中科技大学

并行文件系统元数据管理研究

作者：何飞跃
学位授予单位：华中科技大学

参考文献(42条)

1. 参考文献
2. Rajkumar Buyya.郑纬民.石威.汪东升 高性能集群计算:结构与系统 2001
3. Ibrahim F Haddad PVFS:A Parallel Virtual File System for Linux Clusters 2000(80)
4. Jaechun No.Rajeev Thakur.Alok Choudhary High-performance scientific data management system 2003(04)
5. Rajesh Bordawekar.Steven Landherr.Don Capps Experimental evaluation of the Hewlett-Packard Exemplar file system 1997(03)
6. Jeff Ballard NFS:hunting for a cross-platform file system 1998(12)
7. M Satyanarayanan The evolution of Coda 2002(02)
8. Thomas E Anderson.Michael D Dahlin.Jeanna M Neefe Serverless network file systems 1996(01)
9. Jaechun No.Rajeev Thakur.Alok Choudhary High-performance scientific data management system 2003(04)
10. Nils Nieuwejaar.David Kotz The Galley Parallel File System 1997(04)
11. Jianyong Wang.Zhiwei Xu Cluster file systems:a case study 2002
12. 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[期刊论文]-计算机研究与发展 2002(2)
13. Jose Renato Santos.Richard R Muntz.Berthier Ribeiro-Neto Comparing random data allocation and data striping in multimedia servers 2000(01)
14. G Ganger.M McKusick.C Soules Soft Updates:A Solution to the Metadata Update Problem in File systems 2000(02)
15. Gregory R Ganger.Dawson R Engler.M Frans Kaashoek Fast and flexible application-level networking on exokernel systems 2002(01)
16. Li-Chi Feng.Ruei-Chuan Chang Using Asynchronous Writes on Metadata to Improve File System performance 1999(95)
17. Mark Roantree Metadata management in federated multimedia systems 2002(02)
18. Felix C G(a)rtner Fundamentals of fault-tolerant distributed computing in asynchronous environments 1999(01)
19. Faith Fich.Eric Ruppert Hundreds of impossibility results for distributed computing 2003(02)
20. Miguel Castro.Rodrigo Rodrigues.Barbara Liskov BASE:Using abstraction to improve fault tolerance 2003(03)
21. R Guerraouim.A Schiper Software-Based Replication for Fault Tolerance 1997(04)
22. T W Page.R G Guy.J S Heidemann Perspectives on optimistically replicated,peer-to-peer filing 1998(02)
23. E N Elnozahy.Lorenzo Alvisi.Yi-Min Wang A survey of rollback-recovery protocols in message-passing systems 2002(03)
24. 汪东升.郑纬民 高可用集群计算[期刊论文]-小型微型计算机系统 2000(11)
25. Zhang Youhui.Wang Dongsheng A checkpoint-based high availabinty run-time system for Windows NT clusters 2002(02)
26. F Pérez.J Carretero.F García Evaluating ParFiSys:a high-performance parallel and distributed file system 1997(08)
27. Evans Matthew FTFS:The Design of a Fault Tolerant Distributed File-System 2000
28. TZUNG-SHI CHEN.CHIH-YUNG CHANG.JANG-PING SHEU A Fault-Tolerant Model for Replication in Distributed-File Systems 1999(03)
29. Ling Zhuo.Cho-Li Wang.Francis C.M.Lau Document replication and distribution in extensible geographically distributed web servers 2003(10)
30. Ricardo Jiménez-Peris.M Patino-Martinez.Gustavo Alonso Are quorums an alternative for data replication? 2003(03)
31. Roberto Baldoni.Carlo Marchetti Three-tier replication for FT-CORBA infrastructures 2003(08)
32. R Guerraoui.S Frolund Implementing E-Transactions with Asynchronous Replication 2001(02)
33. 庞丽萍.何飞跃.岳建辉.徐婕 并行文件系统集中式元数据管理高可用系统设计[期刊论文]-计算机工程与科学 2004(11)
34. 于义科.严永福 高性能可伸缩LVS集群的研究和实践[期刊论文]-江西农业大学学报 2001(5)
35. Guang Tan.Hai Jin.Liping Pang Layer 4 Fault Tolerance:Reliability Techniques for Cluster System in Internet Services 2002
36. Kai Hwang.Hai Jin.Roy S C Ho Orthogonal striping and mirroring in distributed RAID for I/O-centrie cluster computing 2002(01)
37. Miguel Castro.Barbara Liskov Practical byzantine fault tolerance and proactive recovery 2002(04)
38. 罗忠海.刘心松.陈勇 基于通用计算机网络实现磁盘镜像[期刊论文]-计算机学报 1998(10)
39. 李翌.吕光宏 Windows NT下的双机磁盘镜像系统的原理及实现[期刊论文]-四川大学学报（工程科学版） 2000(6)
40. Edward Swierk.Emre Kiciman.Nathan C Williams The Roma personal metadata service 2002(05)
41. 庞丽萍.何飞跃.徐婕.岳建辉 PVFS寄生式元数据管理的设计与实现[期刊论文]-计算机工程 2004(20)
42. 毛德操.胡希明 LINUX内核源代码情景分析 2001

相似文献(10条)

1. 期刊论文 魏文国.谢赞福.陈潮填.陈国华 并行文件系统的关键技术与框架设计 -计算机工程2004,30(13)
论述并行文件系统的工作负载特征—基于空间和时间的文件访问模式,并对非连续数据访问技术进行比较研究;给出并行文件系统设计的原则与目标,最后提出并行文件系统的框架.

2. 期刊论文 郑法.郑东 高性能集群文件系统的研究 -计算机工程2004,30(z1)
从解决高性能计算机I/O瓶颈面临的问题着手,详细分析了并行文件系统(PVFS)的结构、存取机制、管理机制和工作机制,为高性能集群文件系统的建立提供了一种行之有效的解决方法.

3. 学位论文 佟强 Linux集群上并行I/O与核外存储策略的研究与实现 2002
为了满足处理大规模数据的需求,该文主要研究和实现Linux集群上的并行I/O与核外存储策略.并行I/O是一个很广泛的领域,包括硬件系统,操作系统支持,语言、编译器和运行系统支持,I/O特征与性能分析,I/O密集型并行应用.该文着重于Linux集群上的并行文件系统的研究与核外存储系统库的设计与实现.首先,该文阐述了并行I/O的系统结构,主要内容涉及磁盘存储系统、RAID和iSCSI,文件系统,互联网络,网络文件系统,并行文件系统,并行I/O界面.然后,介绍了Linux集群上的并行文件系统PVFS（Parallel Virtual File System）.该系统有开放的源代码,并支持多种应用程序接口（API）,因此具有良好的应用前景.最后,提出了一种全新的核外存储问题的实现方案.

4. 期刊论文 曾碧卿.陈敏.邓会敏.曾志文.ZENG Bi-qing.CHEN Min.DENG Hui-min.ZENG Zhi-wen 一种基于集群的新型并行文件系统研究 -信息技术2005,""(8)
首先阐述了基于集群的并行文件系统和并行I/O研究的必要性及其现状,然后提出了一种应用于集群中并行文件访问的新型并行文件系统CLUPFS,介绍了CLUPFS中的数据表示,包括文件分割模型、文件拆分后物理子文件与逻辑视图之间的转换,最后指出了CLUPFS并行文件系统进一步研究与实现的方向.

5. 期刊论文 庞丽萍.蒙廷友.石柯.程斌.唐维.PANG Li-ping.MENG Ting-you.SHI Ke.CHENG Bin.TANG Wei 集群流媒体文件系统MFS设计与实现 -计算机工程与科学2005,27(6)
文章描述了WanLan集群视频服务器上的集群流媒体文件系统(MFS)的设计与实现.MFS是一种支持MPEG文件格式的分布式流媒体文件系统,它由MFS的客户端、管理节点、数据节点以及元数据服务节点组成.MFS流媒体集群文件系统实现了单一系统的逻辑映像、数据和元数据的高可用以及系统自动配置.

6. 学位论文 李永盛 基于并行文件系统的集群高可用性研究与应用 2008
文件系统或裸设备被广泛使用在各种关键的集群系统中。随着应用对可用性要求越来越高，这两种技术都暴露出一些缺点。
文件系统有两个主要问题。第一，冗余问题，既任一时刻它只允许一台主机访问，可以使用热备份软件来切换资源，但是切换的时间段内应用也无法访问数据。第二，性能问题。除一台主机外，集群内的其它主机资源被浪费。
裸设备可在多机间共享，冗余性和性能得到了提高。但它也有一些缺点。首先，裸设备在任何一台主机上被修改后，其它主机必须进行信息同步，这需要停止I/O访问。第二，为了消除I/O访问的“热点”，裸设备使用前需要仔细规划。即便如此，当存储扩容时还是需要停止I/O访问并在扩容后重新进行规划以保障性能上的平衡，这在管理方面和可用性方面都是很不利的。
本文由此展开。首先，介绍了并行文件系统的相关概念，由于它允许多机同时访问，克服了普通文件系统和裸设备的缺点，而集中了二者的优点。第二，介绍了新的基于并行文件系统集群的实施过程，分析了各子系统的高可用性实现方式。第三，提出了用于可用性指标测试的算法。第四，进行了实验测试和可用性指标的对比，验证可用性的有效提升。
[关键词]集群，并行文件系统，GPFS，裸设备，高可用性

7. 期刊论文 封仲淹.万继光.李锡武.李旭.FENG Zhongyan.WAN Jiguang.LI Xiwu.LI Xu 多级集群文件系统的全局命名空间的设计与实现 -计算机工程2006,32(21)
随着存储数据的爆炸性增长和集群技术的快速发展,集群文件系统的研究越来越成为一个焦点,该文设计并实现了一个多级集群文件系统(MCFS)的全局命名空间,MCFS系统采用元数据分层管理思想,建立一个统一的元数据树,从而使整个系统运行在一个松耦合、异构的环境下实现全局命名空间,提高了系统的并行性和可扩展性.在介绍了MCFS命名空间管理的同时,进行了相应的试验测试和性能分析.

8. 学位论文 秦航 基于集群文件系统的元数据容错研究 2004
为了解决PVFS中元数据管理的瓶颈，高可用性集群文件系统元数据容错系统MDFTS 以PVFS为基础平台，对系统中元数据的故障进行检测与诊断，并进行检查点恢复。为了达到复杂的元数据管理一致性，采用了一个无集中式服务器的体系结构，保证所有的数据和元数据能够存放到系统的任意地方，并且在操作的过程中可以动态迁移；采用元数据的磁盘日志结构和内存日志结构相结合的方式对元数据进行管理，减少了fsck对庞大的文件系统中元数据的扫描时间；为了实现故障恢复，提出了元数据容错的设置检查点算法和回卷恢复的算法，提高了文件系统元数据服务的可用性；给出了基于元数据故障的随机过程模型，可以通过减少检错时间提高文件系统的可用度。系统在操作系统应用层实现，通过修改元数据结构和相关的系统调用，使得集群文件系统内部各个数据节点和元数据管理节点相互协作，统一调度，支持高可用性。
测试结果表明，元数据容错系统可以针对系统模拟的不同类型的故障进行错误检测，并能够对系统和应用进行切换与恢复。

9. 期刊论文 贺东鸿 集群Web服务器系统及负载平衡技术在情报信息网中的应用 -现代图书情报技术2004,""(4)
随着集群的广泛应用,集群管理的重要性显得越来越明显.分析了负载平衡集群服务器的结构和特点.探讨了基于情报信息网络负载平衡的主要技术.网络负载平衡提高了诸如Web服务器、FTP服务器和其它关键任务服务器上的服务程序的可用性和可伸缩性.

10. 学位论文 杨俊杰 并行文件系统数据容错研究 2004
实现并行文件系统的容错主要有两种方式：软件磁盘阵列和数据复制。这两种技术对系统的负载平衡和可扩展性都没有提供很好的支持。在PVFS的基础上对并行文件系统数据容错进行研究，采用了一种以每个文件为一个复制组、以PVFS子文件为复制单位的数据复制策略，并利用该策略实现了并行文件系统的数据容错。用户应用程序可以根据对文件可靠性的不同要求，指定文件在系统中存放的副本个数，从而使不同文件有不同的可靠性。
元数据管理器按照数据放置策略使一个I/O节点的所有子文件的副本可以平均分布在所有其他I/O节点上。在任一I/O节点失效后，其上的工作负载能够平均分布到所有其它没有失效的I/O节点上。节点重新加入集群进行数据一致性的恢复时，系统使一致性恢复负载也能够平均分布到整个集群内。以上几点使此数据复制技术与传统的数据复制技术相比更有利于系统的负载平衡、增强系统的可扩展性。