分类号 学号 M201076062
学校代码 10487 密级

華中科技大學

硕士学位论文

基于 Linux 的分布式文件系统的设计与实现

学位申请人 王 亮

学 科 专 业：软件工程

指 导 教 师：高建生 副教授

答 辩 日 期：2013.1.12

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

The Design and Implementation of Distributed File System Based on Linux

Candidate : Wang Liang

Major : Software Engineering

Supervisor: Assoc. Prof. Gao Jiansheng

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

January, 2013

摘 要

在信息技术发展到21世纪后，两个新的词汇越来越受人瞩目。其中一个是以高速发展的互联网技术为基础的“物联网”技术，目前物联网技术正蓬勃发展并极大的改变我们的生活。而另外一个词汇就是以高可靠性的分布式技术为基础的“云计算”技术。提到云计算，行业内部暂时还没有统一清晰的准确定义。但无论怎么定义云计算，分布式文件系统技术都是整个云计算体系中不可或缺的一部分。

分布式文件系统对体系架构的可扩展性和可靠性要求十分高，所以如何选择文件系统载体就是一个非常重要的问题。作为一个周知的开源系统，Linux 就是一个非常好的分布式文件系统的载体操作系统。由于其开放源码特性，选择 Linux 作为分布式文件系统的载体，可更好的利用其开源的用户态文件系统（FUSE）的源码，使得二次开发变更加容易。利用 FUSE 文件系统的源码，就可以对特定使用场景下的分布式文件系统进行需求分析。在需求分析中给出的功能和性能需求指标可以在设计建模过程中得到体现。在实现过程中，应着重的去解决全局名称管理和海量数据冗余备份等在分布式文件系统中常见的问题，并通过测试过程中去印证系统能符合分布式文件系统的各种需求。

利用 Linux 作为基础系统开发出一种分布式文件系统，可以从架构角度更进一步弄清云存储技术的技术细节。在本系统实现过程中所采用的负载均衡和冗余备份等技术能很好的提高分布式文件系统的可用性和可靠性。实践证明，本分布式文件系统能很好的满足特定的项目需求。

关键词：文件系统　　分布式　　Linux 系统

Abstract

With the development of the information technology in the 21th century, two words have attracted more and more attention in the IT field.One word is 'Internet of Things' which is mainly based on the fast developing Internet technology.It has a vigorous development and has already changed peoples' live a lot.The other word is 'Cloud Computing' as we know as another name of the high reliability distributed computing.When mentioned to the 'Cloud Computing', the industry has still not even settled on a definition of it.However,no matter how we define the cloud, the distributed file system technology may always appear as the foundation of it.

The distributed file system requires high on the scalability and reliability,so how to choose a carrier system for the distributed file system is very important. Globally know as a open source operating system, Linux is the suitable carrier of the distributed file system.Because of the open source characteristics, Linux makes the secondary development on it become more and more easier with the open source code of file system in user space(FUSE).Used the code of FUSE, it gives the requirements analysis and design modeling for the distributed file system with particular use.In the process of realization, it focused on solving the common problem of distributed file system,such as global name management and mass data redundancy.Finally, through the test it is proved that the system can meet the various needs of distributed file system.

The implementation of the distributed file system whice used Linux can make the architecture more clearly.The load balancing and redundancy mechanism can improve distributed file system availability and reliability a lot.The experimental results show that the file system can meet the specific needs of the project.

Key words：File System　　Distributed　　Linux System

目 录

1 绪论

在 21 世纪开端的这十几年里，信息技术取得了巨大的发展。尤其是计算机网络技术的高速发展，使得互联网技术已经对人类生产生活的方方面面产生了巨大的影响[1]。新的技术必然导致新的概念的问世，“云计算”正是这些新的概念中的极具影响力的一个。提到云计算，到目前也没有确切统一的定义[2]。但是有一点是被学界广泛接受的，那就是云计算技术的是离不开分布式系统这个技术基础的。

1.1 课题背景

想要具体实际的了解分布式文件系统这个概念，最好的办法就设计并开发一套相关的系统作为研究学习的参照样本。本课题来源于软件学院的软件工程实践基地某大型网络设备提供商的分布式文件系统 SDFS（Smart Distributed File System）项目。开发 SDFS 的目的是为完善公司的网络设备操作系统，持续改进并提供一个基础支撑的文件系统平台，以满足公司产品在升级换代中不断提升的功能和性能需求。本人所研究的课题是“一个基于 Linux 的分布式文件系统的设计与实现”，该课题内容是整个 SDFS 项目中的一个部分。

1.2 课题的研究目的及意义

随着云计算技术的不断在商用领域取得成功，无论是学界还是商界，都开始在这个领域中投入前所未有力度的关注[3]。其中，云存储（结构示意图如图 1.1 所示）就是这个领域中不可或缺的一个重要组成部分。云存储是在云计算概念的基础之上延伸和发展出来的一个新概念，它是指利用集群应用、网格技术或分布式文件系统等技术，将整个网络中的各种不同类型的存储设备通过软件功能集合起来协同工作，共同提供对外的数据存储和业务访问功能的一个分布式系统[4]。从软件架构的角度来讲，作为云计算平台的基础支撑特性，云存储对承载的平台业务有着十分重要的意义。而作为云存储的基础，分布式文件系统对云计算平台的重要性不言而喻[5]。

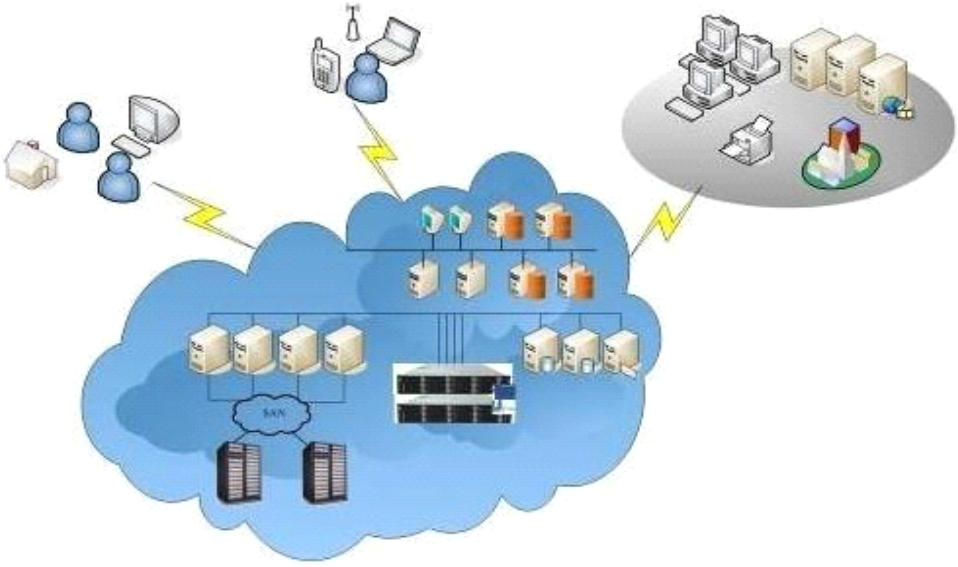

图 1.1　云存储技术结构图

从学术意义上讲，研究分布式文件系统可以从基础架构上更好的认识云计算这个似乎有些捉摸不透的概念。目前云计算尚处在起步阶段，各种云应用纷繁复杂。如何从学术的高度上把握这些应用之间的联系？那么作为整个架构的基础特性的基础，分布式文件系统的架构，从某种意义上来讲，会决定着这些云平台上所承载的业务的样式[6,7]。或者也可以更加具体的理解为，研究分布式文件系统可以很好的绕开应用层特性的多样性，而更好的看到云计算这颗大树下面的“根”。

从商用意义上讲，任何一个云计算平台都会涉及到一个分布式的文件系统的工程。任何一个商用平台系统都是在追求性能和耗费的最大比值（即最大性价比）[8]。那么，想去追求这种高性价比的平台，首先就是要从基础特性入手，去设计一个符合特定用户和特定环境的高性价比架构。具体到分布式文件系统和云计算之间的关系，那就是一个稳定可靠高可伸缩性的分布式文件系统对一个高性价比的云计算系统是有着十分重要的意义[9,10]。

因此，对具体的分布式文件系统进行研究，无论是从学术和商用的角度来讲都是十分必要的。对分布式文件系统的研究不光是能推动对云计算理论模型的学术研究，更具现实意义的是，这个研究不会只是纸上谈兵的。任何一个高性价比的分布式文件系统模型，都可能会具有十分广阔的商用前景[11,12]。

1.3 国内外相关研究情况

最早期的分布式文件系统是在上个世纪 70 年代出现的。随后，其应用范围不断逐渐扩大[13]。从最早所提出的网络文件系统（Network File System）到现在的云存储（Cloud Storage），分布式文件系统在系统规模、体系架构、性能需求和可扩展性等诸多方面都发生了非常大的变化。下面就按照时间来划分阶段，简单地介绍一些国内外研究发展情况。

在 20 世纪 80 年代，分布式文件系统更主要的是强调的操作性能和数据可靠性，其设计的主要目的是为远程文件访问提供标准 API 的。其中最为知名的是上面提到的 NFS 和 AFS(Andrew File System)。后来的分布式文件系统在很多方面都借鉴到它们的成功经验[14,15]。

NFS 在过去的二十多年间，已经推出过四个大的版本。在几乎所有主流的操作系统中都能看到 NFS 身影。事实上，NFS 业界内最富盛名的分布式文件系统。通过 Unix 类系统中的虚拟文件系统（Virtual File System），NFS 可以将用户对分布式文件系统的请求按照文件访问协议规范地来远程调用远程文件服务器端程序进行相应的操作。远程服务器端程序是建立在 VFS 机制之上的。服务器端可以按照本地文件系统处理文件的方式，去实现全局的分布式文件系统的访问。随着 NFS 规范的公开发布，NFS 规范被 Internet 工程任务组（The Internet Engineering Task Force，IETF）列入到征求意见稿（Request for Comments，RFC）[16]。这一举动促使 NFS 的诸多设计实现方式成为了分布式文件系统的参考标准，进一步促进了 NFS 在业界内的普及。由于相关技术的发展，第四版的 NFS 提供了基于会话（Session）语义一致性等最新功能。一个简单的 NFS 的网络架构示意图，如图 1.2 所示[17]。

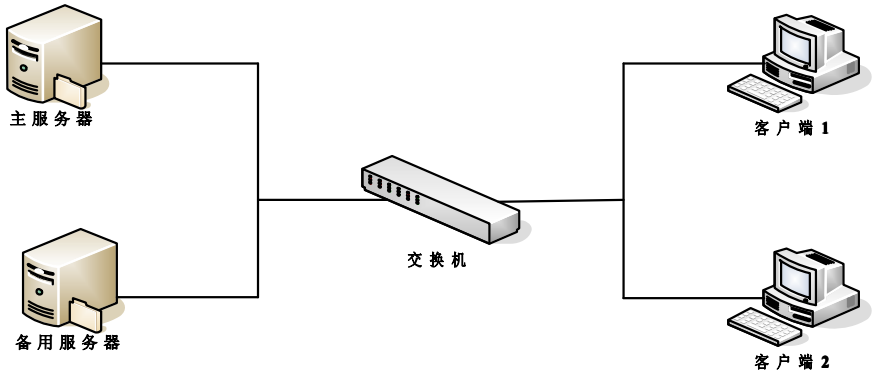

图 1.2　NFS 网络架构图

卡内基梅隆大学（Carnegie Mellon University）于 1983 年开发出来的 AFS 是除了 NFS 之外的另一个著名的分布式文件系统。不同于 NFS，AFS 是将可扩展性作为了设计和实现的核心要点。另外，AFS 也更加考虑到文件系统访问安全性的问题，着重解决了在存在不安全因素的环境中实现安全访问的问题。所以，AFS 在可扩展性、兼容性和访问授权等方面进行了特殊的设计。AFS 可以较容易地支持最大到数千个结点的分布式文件系统。在较大规模环境的情况下，AFS 系统可以充分利用本地存储空间作为整个文件系统的缓存池。在网络环境无法支持远程访问时，单个结点依然可以正常工作[18]。这种做法增加了分布式的可靠性。

早期在计算机软硬件技术尚未发展成熟的情况下，分布式文件系统的设计目标主要是为了给应用程序提供远程访问服务和给出标准接口，在实现过程中，文件系统的性能需求和数据的可靠性被更多的考虑到。AFS 在架构层面进行了有深远意义的探索。NFS 和 AFS 设计过程中形成的相关技术与协议，对以后的设计与实现系统提供了很多可参考的经验。

随后在 90 年代初，随着广域网技术的发展和对存储容量的需求不断提高，加利福尼亚大学设计并开发出了 xFS。由于那时出现了高性能对称多处理器的新技术，xFS 在设计过程中也借鉴了该技术。为了改变传统的分布式文件系统只能被部署并运行在局域网（LAN）的缺点，xFS 巧妙地运用了广域网技术进行缓存，用来减少爆炸式的网络传输流量。由于文件系统存在局部访问的特点，xFS 采用了多级结构[19]。xFS 利用无效写回（Invalidation-based Write Back，IWR）协议来保证缓存内数据的一致性。由于对本地存储空间加以了有效地利用，xFS 有着良好的性能表现[20]。

Frangipani 分布式文件系统利用了虚拟共享磁盘 Petal 技术作为基础技术，同时它也是一种具有分层次的存储系统架构的分布式文件系统。Frangipani 和 Petal 的分层架构使两者的设计实现都得以化繁为简。在 Frangipani 系统内，每个客户端结点同时也是一个文件服务器端，它们可以以相同的方式访问 Petal 提供的虚拟磁盘系统并能进行文件系统的管理。Frangipani 是利用分布式锁的办法解决了同步访问一致性问题。分层次的架构使得 Frangipani 具有非常好的可扩展性。由于分布式文件系统内结点的生命周期可能不同，Frangipani 支持实时动态地增加存储结点[21]。

Frangipani 也支持文件备份等常用功能，同时它也能比较好地来处理网络断开、结点失效等可能出现的故障，改善了系统的容错性。

在 1995 年到 2000 年的这 5 年中，随着网络技术的发展和各种应用需求层出不穷，网络存储技术得到了革命性的发展。基于光纤通信技术的存储域网络技术（SAN）、网络存储技术（NAS）得到了广泛应用，这同时也推动了对分布式文件系统的研究。

在这 5 年里，计算机和网络技术有了突飞猛进式的发展，存储子系统慢慢成为了整个计算机系统科学发展的瓶颈。网络技术的发展和普及应用极大地推动了网络存储技术的发展，这个阶段里，业界内出现了多种充分利用了网络技术的分布式体系架构。由于 SAN 的流行，基于其的分布式文件系统也有不少。例如 SGI 公司的 CXFS 和 HP 公司的 DiFFS[22]。

吸收了之前的各种先进技术经验，IBM 公司开发出了 General Parallel File System（GPFS）分布式文件系统，它也是当时在业界内较为流行的一个系统。GPFS 在设计时就中比较重视先进技术的运用，它的客户端采用了光纤信道或 iSCSI 技术与存储设备连接，同时也可以利用非专用网络进行连接。GPFS 是一种并行分布式文件系统，它采用了共享磁盘（Shared-disk）技术[23]。在支撑大容量的文件系统方面，GPFS 的磁采用了特殊的盘数据结构。利用分布式锁机制的 GPFS 可以较好地解决在分布式文件系统中常见的数据同步和并发访问等一致性问题。分布式锁机制在字节范围内主要被用于数据同步和动态地对元数据结点进行管理。为了解决单点失效问题，GPFS 是利用了日志技术。GPFS 中的每个结点都有自己的实时日志，当某个结点失效时，系统中未失效的结点可以替代已失效的结点核对系统日志，对元数据进行相应的恢复操作。GPFS 是可以在线地对系统进行恢复的。

除了以上所介绍的分布式文件系统外，还有出现过如 Sun 公司的 qFS、EMC 公司的 HighRoad 等在业界内有一定影响力的系统。由于应用对存储容量、性能效率和共享的需求在这一段时期内不断增长，分布式文件系统的规模和复杂性不断增大大。分布式文件系统对存储设备的直接访问、文件检索速度的优化、元数据集中化管理等都是对性能和容量的追求的直接体现。规模的不断扩大使得系统必须具有较

好的动态性，如对实时管理设备的添加删除、数据的一致性、系统可靠性的需求不断增多。同时也有如分布式锁、软升级技术、文件级的负载均衡等更多的先进技术也被应用到分布式系统的设计和实现中[24]。

在进入到 21 世纪后，NAS（其软件架构如图 1.3 所示）和 SAN 结构开始日趋成熟，行业内部开始考虑如何将它们从架构层面进行融合。随着越来越多的对网格技术研究成果的发布，分布式文件系统体系的架构得到了前所未有的发展。同时，基于各种分布式文件系统的成果使得人们对分布式架构的认识不断深入。IBM 公司推出的 StorageTank、Panasas 的 PanFS、Cluster 的 Lustre 等分布式文件系统都是是基于了网格架构。各种更高层面的一些对文件系统的应用需求被提出。如数据量比以往更大，这就产生了对海量数据处理的需求；再如随着网络带宽的不断提高，使得访问要具有更高的性能效率的需求；另外在可扩展性方面，系统规模逐渐加大，对可扩展性的要求越来越高；同时，由于海量数据的出现，这就给分布式文件系统的运行和管理带来了巨大的维护成本负担[25]。

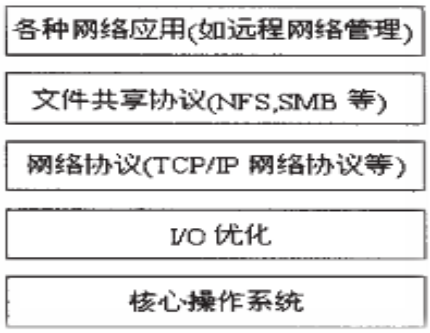

图 1.3　NAS 的软件架构图

在 GPFS 的发展成熟后，IBM 公司开发出了新一代的分布式文件系统--Storage Tank。随后基于 Storage Tank 技术，IBM 公司又推出了 Total Storage SAN File System。它们的出现也对分布式文件系统系统架构的研究做出了巨大的贡献。这两种架构运用 了 SAN 技术作为分布式系统的数据存储的基础技术，同时他们也兼顾到了一些一般的分布式文件系统特性。利用了基于策略的文件数据存储位置选择法，SAN File System 能有效地利用各种系统资源，提高访问性能，降低管理成本。

在 2005 年以后，随着云计算技术的提出和普及，一些基于云计算技术的分布式文件系统也得到了普及。Apache 基金开发的 Hadoop 分布式文件系统（HDFS）是其中的比较有代表性的一种。与以往其他的分布式文件系统相比，HDFS 在设计与实现时更加着重解决了对海量数据进行高可扩展和高可靠的管理的问题[26]。其体系架构如图 1.4 所示。

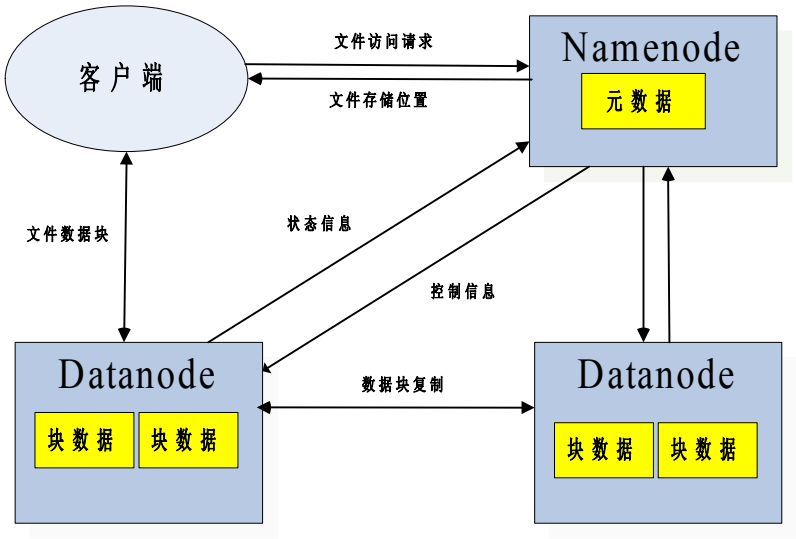

图 1.4　HDFS 系统架构图

从体系架构方面分析，HDFS 主要由唯一主结点 Namenode 和若干个从结点 Datanode 组成。Namenode 是整个系统中唯一的主结点，它主要职责是对各种元数据信息和整个文件系统树进行维护和管理，以对外提供统一的文件名称空间服务。作为从结点的 Datanode 在数量是可以非常多的[27]。Datanode 除了具备最基本的文件存储能力外,同时也具有一定的处理能力用以对本地携带的文件存储资源进行维护。一个具体的文件在 HDFS 中被划分成多个数据块，然后被分散到不同的 Datanode 上。HDFS 中的数据块都可以利用 Datanode 之间的互相复制从而获得多个冗余备份。当客户端程序访问所需文件时，首先向 Namenode 发送包含了文件名字的解析请求。然后，Namenode 将解析后的元数据（主要是文件数据块在 Datanode 上的存储位置）回送到客户端程序，进而客户端程序直接和对应的 Datanode 建立连接并进相关的文件操作。Datanode 定期将自身的状态（如当前保存的文件数据块信息）提交给 Namenode，并接受 Namenode 的管控。例如实施热点文件数据块的复制和迁移[28]。

目前上述这些分布式文件系统都还处在发展的过程中，不过我们也还是能从中总结出一些分布式文件系统的发展趋势：由于云计算技术的推动，业界内对分布式体系架构的探讨越来越多，不同的分布式文件系统的体系架构日趋一致；分布式文件系统的设计策略大同小异，基本都是利用了专用服务器等办法；在设计细节方面，各分布式文件系统都采用了一些特有技术来提高系统的性能和可扩展性。另外，协议方面的研究也是业界内关注的热点[29]。

1.4 本文的主要研究内容

本文主要做的研究工作主要有以下几点：

（1）首先介绍分布式文件系统的基本概念，在此基础上分析在 Linux 平台的特性，结合相关分析，给出系统设计上的可行性分析，得出系统设计的可行性结论，指导后续的需求工作。

（2）通过可行性分析后，做出系统的需求分析（主要包括功能需求和性能需求两个方面）。得出需求分析结论，指导后续的系统实现。

（3）基于之前的需求分析结论，对分布式文件系统进行设计，最后给出总体的设计分析总结。

（4）基于需求分析与设计，进行实现方面的工作。在完成实现后，应对系统进行较为详细的测试与分析，总结出其特点。

2 相关技术分析

要设计和实现一个基于 Linux 的分布式文件系统，有很多概念和前提条件是需要理清的。首先需要对分布式文件系统的相关概念进行说明，其次关于 Linux 系统的一些基础知识也必须阐明。

2.1 分布式文件系统的基本概念

下面从分布式的相关概念、文件系统的基本介绍和云计算与分布式的关系三个方面讨论。

2.1.1 分布式的相关概念

分布式这个知识体系是包含了许多内容的，它是整个计算机学科知识体系里很重要的一部分，分布式这个知识体系也依赖于整个信息技术业界对它加以利用和发展[30]。整个分布式计算的理论体系不但包括那些已经被证明和使用过的理论知识和已经被整个领域充分应用的传统经验，更还包括了不断推陈出新的理论知识以及还没有被充分应用的先进经验。

（1）分布式计算的简单介绍

分布式计算技术是计算机学科的一个分支，它主要研究的是将一个需要耗费十分多计算资源才可以得以解决的问题划分为很多小的部分，然后将这些被分解后的部分问题分配到许多处于不同地点的计算机上进行解决，最后把这些每台计算机上的结果收集并总结起来得到最终的解决方案[31]。分布式存储是分布式计算的基础，它指的是将数据资源分散的存储于多台独立的存储设备上。分布式存储系统一般都采用了扩展性很好的系统架构，利用许多台子服务器进行数据存储的分担。利用名称（位置）服务器来定位所存储数据的位置信息。这样做不但解决了集中式存储系统的单存储服务器的性能瓶颈问题，还提高了系统的可用性、可靠性以及可扩展性。

（2）分布式计算与 Internet 的关系

由于计算机在全球的普及，个人计算机(Personal Computer)早已走进千家万户。

与之相伴，个人计算机资源的利用率也成为了一个问题。现实情况下，非常多的计算机资源处于未被完全利用的状态，就算是在运行状态下的中央处理器（Central Processing Unit，CPU）运算潜力也远远未被很好地利用。正如我们所常遇到的，个人计算机的大多数时间都耗费在“等待”状态上面。即便是用户正在使用他们的个人计算机时,CPU 依然是在“等待”上消费，仍然是不计其数的等待（等待 I/O 结束，但事实上什么也没有做）。由于 Internet 的出现, 使得远程访问这些具有软硬件资源的“空闲”计算机系统成为了现实。通过 Internet 的互联技术，用户可以充分利用分散在各处的计算机资源，而这便是分布式计算的基础模型。

（3）分布式计算与云计算的关系

从云计算的起源来讲，它是充分融合并发展了传统分布式计算的理论与实践的。回顾云计算发展初期的历史，其发展的基础和动力均来自传统的分布式计算技术。2007 年年底，全球著名的 IT 企业 IBM 和 Google 开始了一个在美国大学校园内推广云计算技术的计划。这项计划的目的是系统通过新技术的推广能降低分布式计算在学术研究方面的成本，同时也能为进行相关研究的大学提供软硬件和技术方面的支持（其中包括数百台的 System x 与 BladeCenter 服务器，这些计算机系统包括了将近 1600 个处理器并支持包括 Linux、Hadoop 或 Xen 等开放源码的系统）。而这些大学的学生则可以通过网络自愿参与开发各项大规模计算的研究计划。而后，越来越多的商业企业或大学研究机构进入到云计算这个领域（如 Amazon 等），他们为支撑起自己的云计算体系，纷纷都构建了自己的云计算体系架构。纵观以上这些云体系，无论上层的应用构件如何复杂，其底层基础的组织架构，依然离不开分布式计算这个不算太新的概念与技术[32]。

2.1.2 文件系统的特点

文件系统对于操作系统(Operating System)来说是不可或缺的组成部分。文件系统通过对 OS 所管理的存储系统对象的抽象，向上层应用服务提供统一对象化后的操作接口，屏蔽对底层物理设备的资源管理和直接访问。文件系统，依据所处环境和所提供功能的不同，可被划分为四个层次，由底层到高层分别是：单处理器单用户的本地文件系统，如 Microsoft DOS 的文件系统；多处理器单用户的本地文件系

统，如 IBM 的 OS/2 的文件系统；多处理器多用户的本地文件系统，如 Linux 的文件系统；多处理器多用户的分布式文件系统,如 Apache 基金会开发的 Hadoop 分布式文件系统[33,34]。

本地文件系统指的是被文件系统所管理的资源直接存在于本地结点上，处理器在本地的总线就可以直接访问到这些资源。分布式文件系统与本地文件系统的不同之处主要在于资源的存储位置上。分布式文件系统指的是被文件系统所管理的资源并非一定存在本地结点上，处理器可以通过计算机网络去访问对应的资源。按上述的层次的分类，高层次文件系统都以低层次的系统为基础，在更高层次实现了更多的功能特性。例如多处理器单用户的本地文件系统要比低层次的单处理器单用户本地文件系统多考虑并发访问控制问题，这是由于在多处理器的情况下，两个处理器可能需要同时访问同一个资源；多处理器多用户文件系统要比多处理器单用户文件系统多解决多用户文件资源访问控制的问题，因为对于多个用户的系统，不是每一个文件资源都可以被所有用户访问；多处理器多用户分布式文件系统要比前一个层次的多处理器多用户文件系统多解决由分布式架构所带来的问题，比如访问同步问题、文件缓存一致性问题等。

伴随层次的提高，文件系统设计和实现的困难程度也会大大提高。但是，最新的分布式文件系统依然还是保持与最原始的本地文件系统近乎完全相同的抽象对象模型和操作访问接口。这样做的主要目的是在更方便的向用户提供向后的兼容性的同时，文件系统也能保持与原来一样简洁的抽象对象模型和操作访问接口。尽管如此，这一切并不意味着文件系统设计和实现的困难程度没有提高。相反，正是为了满足用户需求，以简化分布式文件访问的复杂性，软件开发者们才对最基础的文件系统架构不断进行了修改，这也提高了文件系统实现的困难程度[35]。

根据著名的摩尔定律，随着技术的不断发展，计算机性能会不断提升，而计算机硬件的造价却不断下降。现在，用户能以更低的成本购买到更快、更好、更加稳定的硬件设备。文件系统面临的新问题也都随之而来：如何管理更大的存储设备以提供更好的性能，更进一步地降低管理开销等。各种新的分布式文件系统技术层出不穷以满足日益增长的用户需求。

2.2 Linux 平台的简单介绍

整个项目中最核心支撑部分就是 Linux 系统。Linux 的核心思想就是不断集成的模块化的开源操作系统。将 Linux 系统选择为二次开发的起点，会使得项目开发变得容易。

2.2.1 Linux 的特点

Linux 是继承并发展著名的 Unix 系统的一种新时代的操作系统。回顾 Linux 的发展历史是一件非常有意思的事情。Linux 诞生于 1991 年 10 月 5 日（这就是芬南人 Linus Torvalds 第一次正式向外发布 Linux 的日子）。由于 Internet 网络在近 20 年内的快速普及和世界各地计算机爱好者的不断努力，Linux 已经成为当今世界上使用人数最多的一种类 Unix 操作系统，而且人数还在持续增加。

Linux 是一个可以被自由使用和修改的类 Unix 操作系统，它也是一个基于可移植操作系统接口标准（POSIX）和支持多用户、多任务、并且能支持多线程的操作系统[36]。在硬件平台方面，Linux 可以被部署到 32 位和 64 位硬件平台上。继承了 Unix 网络为核心的设计理念，Linux 是一个性能强大且稳定的网络操作系统。多数情况下，Linux 主要用于装载了 Intel x86 系列 CPU 的主机上。由于 Linux 开源化的特点，这个系统可由世界各地的不同的程序开发人员自行设计和实现。这样做目的是构建不受到任何商品软件的版权限制的，且能被所有人免费使用的类 Unix 系统的产品。

Linux 在 GNU 公共许可权限下，是可以被免费使用的。Linux 以它的高效能和高灵活度而著称于世。模块化的实现架构，使 Linux 既能够在性能较差的个人电脑上实现 Unix 特性，同时也能高性能的工作站上得到很好的运行。它是一个符合 POSIX 标准的，同时又具有多用户、多任务的能力的操作系统。Linux 系统所带软件不但包括全部的 Linux 操作系统自身，还包括了一些基于其的高级应用软件，如 vi 文本编辑器、gcc 高级语言编译器等[37]。同时，Linux 还包括了一个被称为 X-Windows 的图形用户界面（GUI），如同使用 Windows XP 系统一样，我们可以使用鼠标对图标、菜单或窗口进行操作。

2.2.2 Linux 文件系统的特征

最早期的 Linux 的文件系统被称作 Minix。后来随着 Linux 不断的成熟，专门为其设计的——扩展文件系统第二版（Ext2）被专业人员实现出来并加载到 Linux 系统软件包内，这也 Linux 发展历史上的一个意义重大的事件。较传统的 Minix 文件系统，Ext2 文件系统更易扩充、在性能上也得到了较大的优化。Ext2 同时也是大部分的 Linux 发布版内的标准文件系统的类型。

虽然 Ext2 是 Linux 中比较常用的文件系统，且目前还出现了带有日志功能的 Ext3 文件系统。

但是，Linux 依然能够支持很多其他类型的文件系统。像最近这些年，业界出现的好多种访问速度特别快的日志文件系统，像 SGI 公司的 xFS 文件系统，以及适用于更小型的 Reiserfs 文件系统，Linux 都能很好的支持。而像传统的 Microsoft 的 FAT 文件系统，也能够被加载为 Linux 的文件系统，并正常访问[38]。

既然有这么多种类的文件系统可以被加载到 Linux 下，那么他们之间又是通过什么机制工作的呢？

由于存在多种类型的文件系统，为了使它们能够从文件系统的逻辑类型和操作系统服务中区隔出来成为相对独立的功能模块，文件系统和操作系统底层需要通过接口层来通讯。这个接口层被称为虚拟文件系统（Virtual Filesystem Switch ,VFS）。有了 VFS，不同类型的文件系统都可被加载到 Linux 下，有一套 VFS 的通用接口可以被调用[39]。

通过软件技术，VFS 可以使文件系统的细节技术得到较大程度的屏蔽,这样内核和应用程序可以将存储在不同类型的文件系统下的数据资源都可以被看成同一的文件系统。整个 VFS 的架构如图 2.1 所示。

通常，在 Linux 文件系统中，还有一类特别类型的文件系统，它不存在于外存空间而是存储在系统的内存，它就是/proc 文件系统。

/proc 以文件系统的方式存在，并为用户获得系统内核数据的操作提供接口。它其实是一个虚拟文件系统，应用程序能使用它获得系统信息，并可以通过它设置一些内核参数。

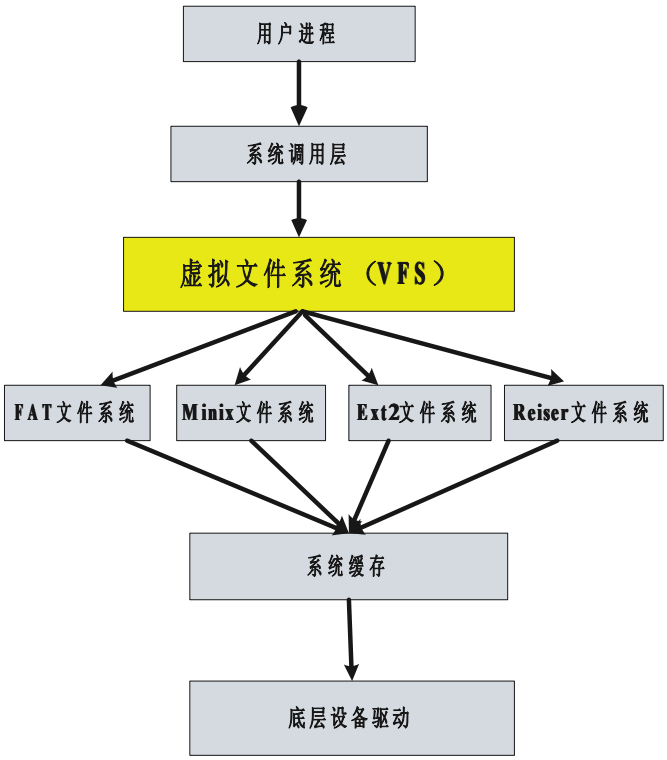

图 2.1　VFS 系统架构

Linux 系统存在一些具有特殊功能的目录和文件，这些目录和文件无论在什么版本的 Linux 系统中都是不可或缺的。整个 Linux 系统都是由这些不同功用的目录构成的。常用的 Linux 目录结构如图 2.2 所示[40]。

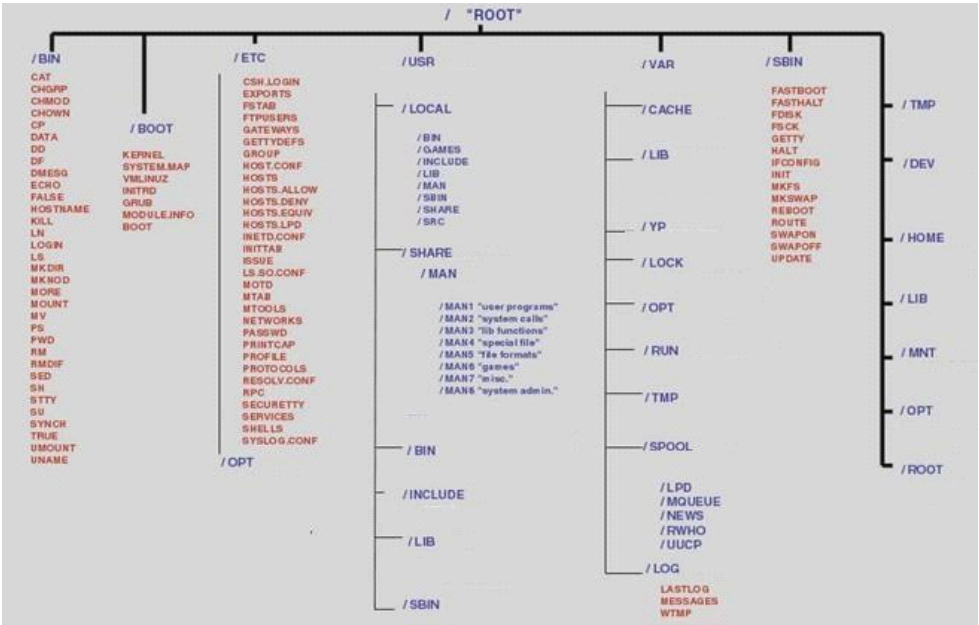

图 2.2　Linux 目录结构图

2.2.3 Linux 用户态文件系统技术

由于文件系统在操作系统中的特殊地位，在一般的系统设计与实现时都会将文件系统安排在内核（Kernel）中。但是在 Linux 系统中，为了使没有权限修改内核代码的用户也能创建属于自己的文件系统，从 2.6.14 版本后的 Linux 开始在该系统中支持用户态文件系统（File system in user space，FUSE）。与传统的内核态文件系统相比，用户态文件系统具有更好的可扩展性和更便于开发人员调试的特点，但由于需要在用户态和内核态之间进行切换，用户态文件系统在性能方面可能会受到一定的影响。同时，用户态文件系统也是一个免费的开源软件，在 GNU 通用公共许可证的允许下，它可以被自由地使用、复制、发布和修改[41]。用户态文件系统和内核的关系是比较复杂的，互相之间具体的调用关系如图 2.3 所示。

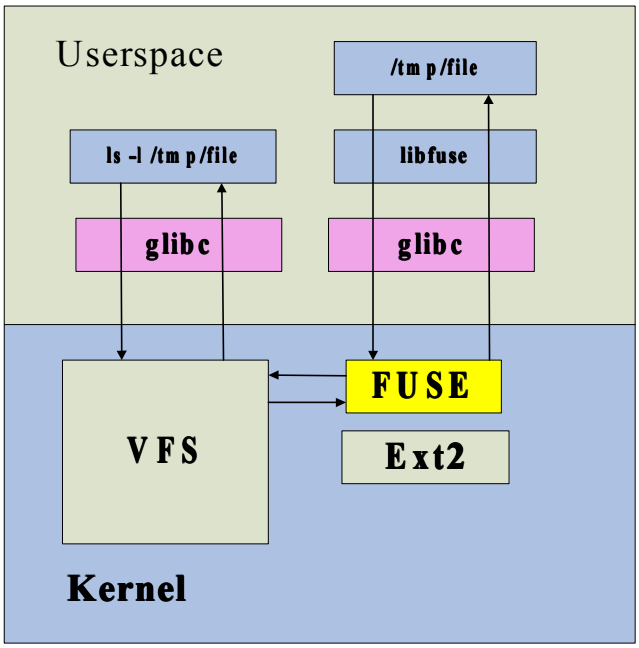

图 2.3　用户态文件系统交互图

2.3 本章小结

本章通过对分布式技术和文件系统技术这两个基本的技术概念的进行拆分介绍，进一步阐述了与分布式文件系统有关的一些基本概念和背景知识。本章首先主要介绍了分布式技术的一些基本相关概念，然后由这些基本概念延伸出分布式技术

与 Internet 和云计算技术等概念之间的联系。随后，从文件系统技术的介绍入手，简单地讲解了文件系统技术的基本概念和背景知识。最后，根据之前介绍的文件系统的基本概念，较为详细的介绍了 Linux 文件系统的一些技术细节，为分布式文件系统的设计与实现提供了理论基础。

3 分布式文件系统的需求分析和设计

需求分析和设计阶段对整个系统的实现是起着决定性作用的。在需求分析和设计中，对整个系统会进行一个全面的分析和规划。在这个过程中，即会对系统模型进行定性的详细描述也会从细节的角度对系统进行定量分析。这一过程决定了在实现过程中所围绕的重点和方向。

3.1 分布式文件系统的需求

本分布式文件系统的主要用途是基于 Linux 系统来实现对分布式的文件系统进行统一管理。从系统需求角度来讲，主要分为系统的功能需求与性能需求两个方面。对分布式文件系统的功能需求主要是分布式系统和文件系统的功能的集合。性能方面的需求主要是针对分布式系统需要对运行性能有一个最低的度量化标准[42]。本节会从这两方面分开来介绍。

3.1.1 需求背景概述

本需求主要是为项目的设计与开发人员进行相关工作提供需求层面的参考与指导。本系统的主要用户和维护人员为使用某商用分布式网络管理系统的网络管理员，这些网络管理员一般具有较多的网络管理经验并且熟悉系统的基本功能。系统的主要目标是为分布在不同物理位置的网络结点提供统一的文件系统管理入口，简化系统的管理结构。

3.1.2 系统功能需求

功能需求是对项目最基础的功能进行的描述。从典型的应用场景进行需求建模，我们可以得到以下几个方面的需求模型：

1）文件系统的加载与卸载

因为分布式系统的开放特性，系统内的结点随时可加入或者退出系统。所以，对于要加入分布式系统的文件存储结点要有统一的加载方式。而当存储结点退出

时，分布式系统要实时的卸载结点上的文件系统。基本的功能主要包括以下几个方面：

（1）服务器和客户端程序的创建

整个系统启动时服务器程序需要被首先被自动创建出以提供对应的分布式文件服务。

而当客户端结点启动时，客户端系统要能自动创建出客户端文件系统守护进程来和服务器端进行交互。

（2）文件结点的加载

有新结点加入系统时，需要上报远程文件系统加载事件到服务器。服务器通过响应该事件来加载新结点。

（3）加载路径的统一管理

分布式系统需要统一分配远程文件系统的服务器端加载路径，确保每个文件结点的加载路径唯一且不重复。

（4）文件结点的卸载

远程文件系统退出或者失效时，客户端系统需要上报远程文件系统退出事件。当服务器系统内部需要自动且正确的处理文件结点的退出事件，同时要对分配的路径进行统一的回收。

（5）内部资源的分配与管理

从结点加入到退出的整个生命周期中，会用到许多的系统内部资源（如通信信道或定时器等）。这就需要在整个过程中，对内部资源进行统一而有序的管理，避免资源泄漏等严重问题。

2）全局的文件名称服务

由于分布式系统需要对不同物理位置上的文件存储结点进行统一的管理，这样就需要对每一个文件系统有一个全局路径的分配。这个全局路径是每个文件结点在系统内的唯一标识。

其需求内容主要包括以下几个方面：

（1）全局路径名称的统一管理

由于每个结点需要被统一加载并有所区别的唯一标识，这就要求对全局的名称格式进行集中化管理，使得每个被加载入系统中的结点能通过所分配的名称就能被正常的访问。

（2）文件结点名称的分配和回收原则

当有结点加入时，需要从路径资源池内挑选并分配给文件系统结点。当结点退出或失效时，需要将其获得的路径资源回收到路径池内。这就要求有统一的分配和回收原则来进行相应的操作。

（3）地址资源的统计功能

系统应能实时统计地址资源使用情况，用来分析当前系统所承载的业务量的大小，便于管理人员了解系统使用情况。

3）基本的文件系统操作

在架构层次，整个文件系统是由若干个分布式结点组成的。对于每个文件结点，在本地架构内文件结点需要提供一些基本的文件系统操作的功能来满足相应的本地操作需求，具体内容包括以下几个方面：

（1）目录相关操作

如创建、删除目录和目录改名等常见文件系统操作。不同于本地文件系统，本系统应支持对远程目录的操作。

（2）文件相关操作

如文件更名、删除和改变路径等常见文件系统操作。不同于本地文件系统，本系统应支持对远程文件的操作。

（3）工作路径相关操作

如切换工作目录，获取当前工作路径等常见文件系统操作。不同于本地文件系统，本系统应支持对远程工作路径的操作。

4）海量数据的存储

分布式文件系统内往往存在着大容量或者数量较多的小文件分块，这就对文件系统的提出了海量数据存储的功能要求。同时，由于每个文件系统结点上的负载可能较大区别，这就对文件系统发现并存储大容量文件提出了负载均衡的需求。具体

需求包含以下几个方面：

（1）海量文件的负载均衡

当多个文件在多个文件结点上的负载出现一定程度的不均衡分配后，系统应能自动发现不均衡事件的出现，并能自动进行负载均衡操作。

（2）文件的冗余备份

系统管理员应能通过一定的方式指定对具体的文件结点进行冗余备份的操作，方便管理员的管理操作。

5）提供基本的应用编程接口

由于文件系统是操作系统最基本的特性之一，所以为了其他模块能进行相关的文件操作，分布式系统需要为上层应用程序提供统一的文件系统编程接口，具体的需求主要包括以下几个方面：

（1） 文件的打开关闭

如同普通 Linux 系统提供的文件打开关闭编程接口（传统 Linux 中的 open 和 close 函数），本系统应提供支持远程操作的对应编程接口。这些操作函数的返回应保持和传统接口相同的形式。

（2） 文件的读写

如同普通 Linux 系统提供的文件读写编程接口（传统 Linux 中的 read 和 write 函数），本系统应提供支持远程操作的对应编程接口。这些操作函数的返回应保持和传统接口相同的形式。

（3） 文件权限管理

如同普通 Linux 系统提供的文件权限管理操作（传统 Linux 中的 chmod 函数），本系统应提供支持远程操作的对应编程接口。这些操作函数的返回应保持和传统接口相同的形式。

（4） 文件工作路径操作

如同普通 Linux 系统提供的文件权限管理操作（传统 Linux 中的 cd 函数），本系统应提供支持远程操作的对应编程接口。这些操作函数的返回应保持和传统接口相同的形式。

3.1.3 系统性能需求

性能需求是除功能需求外，软件项目需求的另一个重要组成部分。性能需求描述了用户对系统性能的期望，为开发和测试人员提供定量的参考值。本项目的主要性能需求包括以下几个方面：

（1）海量文件处理性能

系统的海量数据处理性能主要是指当系统的负载达到一定的阈值后（系统默认为 75%），系统应立即执行负载均衡操作。

（2）系统并行承载终端性能

在典型的应用场景下，系统要承载大量的分布式网络结点接入。因此，对于本系统要求能同时承载 100 个分布式终端接入。

（3）分布式命令最大响应时延

在分布式的架构下，任何客户端的操作都需要有一定的响应时间需求。对于本系统，要求在分布式情况下基本文件命令操作（如复制、删除和修改等）响应时间不大于 1 秒。

（4）并发请求性能

并发请求性能需求是描述的最大同时处理的来自客户端的命令数量。对于本系统，要求该分布式文件系统在服务器端最大同时存在 10 个并发请求的情况下，能正确处理所有请求。

（5）可靠性

系统的可靠性主要是描述是软件维持在其性能水平的能力。对本系统，由于文件结点的规模并非十分巨大，所以便要求分布式系统的平均无故障时间不小于 10000 小时。

3.2 系统总体设计

为了能对分布式文件系统进行实现，本节在上文进行了详细的需求分析基础上，可根据给出的分析结果给出系统在架构和概念上的设计。本节给出的系统总体设计可以为后面的模块设计和系统实现给出参照和指导。

3.2.1 系统架构描述

在系统的总体架构层面，本系统分为客户端系统和服务器端程序。服务器在启动时，主动运行服务器端程序。

服务器端可分为主服务器和备用服务器。在主服务器端，系统只启动服务器程序，而在备用服务器端同时存在着服务器端和客户端程序。主服务器端主要负责提供路径名称服务。备用服务器主要是对主服务器的冗余备份。在主服务器失效时，备用服务器可替代主服务器提供服务。同时，在文件结点接入到主服务器端后，应该由客户端的操作系统启动客户端模块。客户端负责将用户的文件操作请求发送到主服务器端，主服务器端在收到客户请求后负责将命令解析并分发到对应的结点上去处理，并负责讲结点处理结果回报给请求用户的客户端。系统的总体架构如图 3.1 所示。

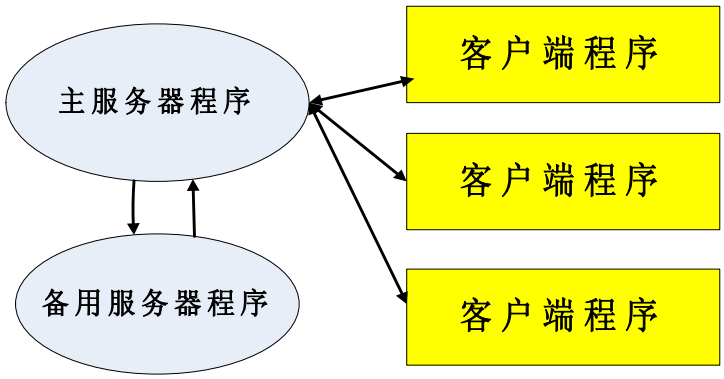

图 3.1 系统总体架构图

3.2.2 系统组网描述

在组网方面，本系统可将主服务器程序部署在主服务器，备用服务器程序部署到备用服务器之上。

主服务器和备用服务器通过局域网互联，形成主备模式。在主备服务器的局域网段之外，分布着各个文件结点客户机。每个文件结点客户机可通过主备服务器局域网提供的统一防火墙出口加入系统。防火墙的主要作用是防止客户机的恶意网络攻击。

本系统的主要组网图如图 3.2 所示。

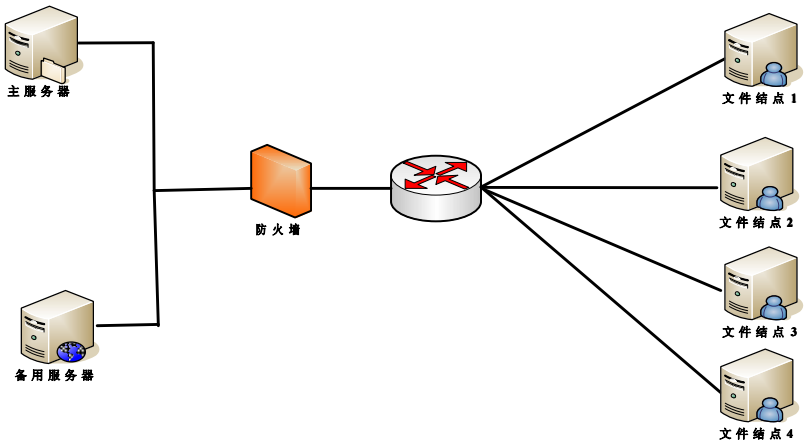

图 3.2　系统组网图

3.3　系统功能设计

在完成对系统的架构设计后，需要对每一个具体的模块进行详细的功能设计。功能设计的主要目的就是细化模块功能定义，并尽量给出靠近实现层面的解决方案。下面就是对本分布式文件系统的详细设计。

3.3.1　服务器模块的设计

在服务器端，系统可被分为主服务器模块和备用服务器模块，其主要区别就在于是否能提供全局的名称解析服务。在主服务器模块内，需要有一个能管理全局名称的二级子模块，这个子模块可以被称为名称服务子模块（Global name server module）。同时，主服务器还需要监听来自于客户端的请求。当有客户端请求加入系统时，主服务器模块需要验证并加载客户端。在主服务器端内这个二级子模块被称作请求处理子模块（Request processing module）。另外，由于系统内可能存在大负载量的文件，主服务器模块需要及时找到并处理这些文件。在主服务器模块内，负责做这项工作的子模块被称为负载均衡器子模块（Load balancer module）。除了上面所述，主服务器模块内还存在着一个备份管理子模块（Backup management module），它主要负责将自身的信息备份到备用服务器上。以下就从具体的四个子模块进行详细的设计分析：

（1）请求处理子模块（Request processing module）

请求服务子模块主要负责管理并响应来自于各个客户端模块的各种文件操作请求。这些请求主要包括客户端加入和退出系统请求、客户端认证请求、客户端名称服务请求以及客户端基本文件操作请求等。另外，为了响应这些请求，请求处理子模块会发送一些响应报文。这些响应报文主要包括加入请求响应、认证请求响应、基本操作响应等。这一系列通信过程主要遵从一个传输协议，这个协议主要由两部分组成，一个部分是协议头部分，一个是协议内容部分。协议头主要包括报文长度、报文保留字段和报文头校验码等。协议内容部分主要包括了报文类型、附带数据段和报文内容校验码。报文主要结构主要如图 3.3 所示。图中第二行的数据表示该字段所占存储空间的大小，第三行表示该字段在报文中是否是必选字段，其中 M 表示该字段为必选，O 表示该字段是可选字段。

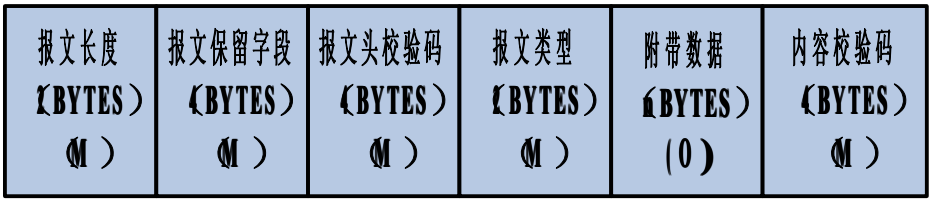

图 3.3　报文结构图

协议结构，如图 3.3 所示。本协议结构一共包括 6 个字段。在这 6 个字段当中，有 5 个是必选的字段（在图中用 M 表示），只有附带数据字段是可选的报文字段（在图中用 O 表示）。

协议中的第一部分是报文长度字段，一共占 2 字节，最大值为 65535，即 64KB。协议中的第二部分是报文保留字段，一共占 4 字节，主要是为了方便以后协议扩展所保留。协议中的第三和第六部分分别是报文头和报文内容校验码，它们均占 4 字节，可采用 CRC32 等主流校验和算法，其用途主要是校验对应的数据段传输过程中的正确性。

协议中的第四部分是报文类型字段，其主要表示报文所请求或响应的事件类型。协议中的第五部分是附带数据内容，其主要根据报文类型决定，如果某具体的报文类型需要附带对应类型的数据，则该字段可以附加入报文，否则这个字段并非必要。报文类型和附带数据这两个字段的具体含义可参照表 3.1 所示。

表 3.1　报文协议表

报文类型	报文类型值	附带数据
客户端加入请求	0x0001	无
客户端加入响应	0x0002	无
客户端退出请求	0x0003	无
客户端退出响应	0x0004	无
名称服务请求	0x0005	结点名称
名称服务响应	0x0006	0x0000 结点正常 0x0001 结点异常
文件操作请求	0x0007	0x01 创建目录 + 目录路径 0x02 删除目录+目录路径 0x03 移动文件 + 旧文件路径 + 新文件路径 0x04 删除文件 + 文件路径 0x05 清空回收站+ 结点名称 0x06 还原回收站+ 结点名称 0x07 拷贝文件 + 源文件路径 + 目标文件路径
文件操作响应	0x0008	0x0000 操作成功 0x0001 操作失败
负载统计上报	0x0009	负载数据
负载上报确认	0x000a	无
负载均衡请求	0x000b	当前负载量
负载均衡响应	0x000c	0x0000 处理成功 0x0001 处理失败
负载量查询请求	0x000d	结点编号 例：查询结点001就是0x0001
负载量查询返回	0x000e	负载数据

（2）名称服务子模块（Global name server module）

名称服务子模块主要是为接入的远程文件系统结点分配对应的全局名称。考虑本系统典型为分布式组网（具体的组网图如图 3.2 所示）。当客户端接入到系统后，客户端会请求挂载到服务器端。如果该请求被服务器端认证通过，服务器的请求处理子模块会将其挂载到服务器的/mnt 目录下，具体的目录名称格式为/mnt/fs_node_xxx，其中 xxx 为接入系统的客户端顺序编号，第一个接入系统的为 fs_node_001，以后接入的以此类推。需要特别指出的服务器端的本地文件系统也会

被挂接，但挂接的结点号为 0,也就是说服务器的本地文件系统的挂接格式为 fs_node_000。

在名称服务子模块中维持着一个具体的表（List）。这个表的所存储的对象就是各个结点与服务器的通信连接号（形如 Socket 中的 iFd）和结点编号的映射关系。因为文件结点可以动态的加入或退出，所以该表需要具有动态增减容量的功能。具体如图 3.4 所示。

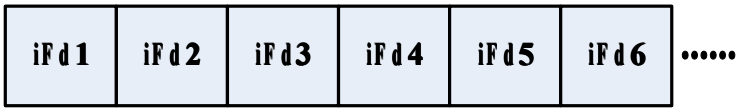

图 3.4　结点编号映射表

当其他服务器端子模块需要和获得对应的需要的通信连接号时，即可通过调用本子模块的查询接口，获得对应的通信连接号。如若输入的结点编号无法在表里查询到，即返回一个无效的连接号（一般可设置为-1）来表示。

（3）负载均衡器子模块（Load balancer module）

主服务器负责处理负载过大的客户端。客户端模块内有一个定时统计本地文件系统负载和将实时统计值发送给主服务器的功能（具体报文格式可参加图 3.5 所示）。主服务器收到统计数据后，可在负载均衡子模块内建立一个负载队列(Load Queue)来管理全局各个文件结点实时的负载情况。负载队列如图 3.5 所示。

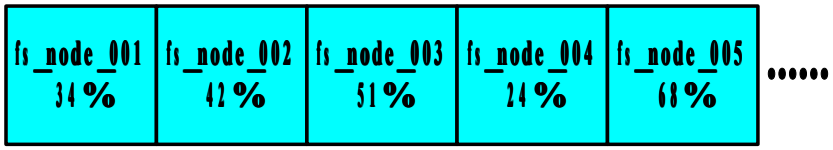

图 3.5　负载队列图

当客户端发现本地的文件系统负载达到系统设定的阀值时（一般为本地文件系统最大负载的 75%），客户端就主动向服务器发送负载均衡请求。主服务器收到负载均衡请求后，需在本地的负载队列中需找到能负载较低的一个文件结点，并采用远程文件拷贝的办法将负载较重的结点内的大文件拷贝到负载较轻的结点中去，然后实时刷新负载队列。

（4）备份管理子模块（Backup management module）

备份管理子模块主要是用来管理系统的例行备份作业的。在本系统中，服务器端管理人员可以设置日常需备份的结点编号和备份频率。当备份管理子模块读取到有备份设置时，该子模块需要逐一的执行备份脚本。

备份的原则是尽量将备份文件放入负载量较小的文件结点（结点负载量可通过查询负载均衡子模块里的负载队列得到），同时需要有一个能记录备份源和目的结点的对应关系表，便于在文件删除时能同时清除备份文件。另外，备份管理子模块需要通过检查备份源和目的映射表关系去避免循环备份的问题（即将备份的目的结点的文件又备份到源结点上的问题）。

3.3.2 客户端模块的设计

在客户端模块内，需要有一个负责和主服务器交互的模块，这个模块在客户端内被称为请求管理模块（Request management module）。另外，由于每一个客户端结点上同时也是文件系统的结点之一，所以在每个客户端模块内也有一个负责处理本地文件系统基本操作的子模块，这个子模块是本地文件系统模块（Local file system module）。以下就从这两个子模块具体讲起：

（1）请求管理子模块（Request management module）

请求管理子模块的主要作用有两方面。第一方面，负责建立并维护和服务器端的通信链接。当客户端初始化时，请求管理子模块负责向服务器端发送链接请求，并等待服务器端接受。当服务器端接受链接请求后，请求管理子模块就可以发送各种请求。当客户端需要退出系统时，请求管理子模块负责关闭对应的通信链接并清理相应的资源。另外一方面就是负责管理客户端发往服务器的各种请求。本系统中，在可为用户提供命令行（Command Line）和 Web 化两种管理方式。当客户端有相应事件或者有相应的用户输入请求时，客户端的请求管理子模块就要根据输入请求去封装对应的报文，并将封装好的报文发送到服务器端。具体的报文封装格式可参见图 3.3 所示。当发送完报文后，请求管理子模块应会等待来自于服务器端的应答。当请求管理子模块收到来自于服务器端的应答后，需要将应答解析并结果回传至命令行或 Web 模块。这些结果主要被用于给与用户操作结果应答。整个交互过程的顺序图如图 3.6 所示。

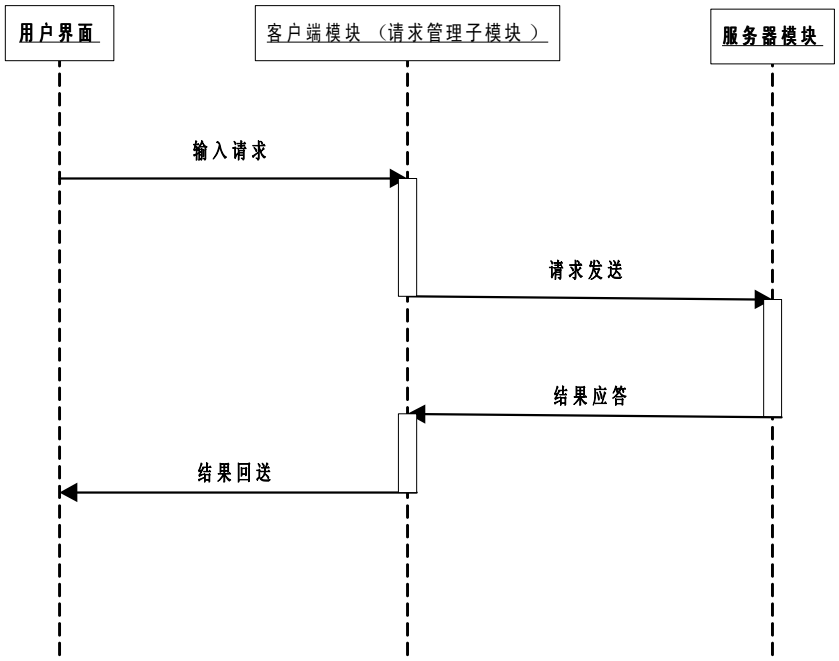

图 3.6　交互顺序图

（2）本地文件系统子模块（Local file system module）

本地文件系统子模块主要继承于 Linux 的用户态文件系统（FUSE）。其主要功能包括维护文件系统的一些基本操作接口和管理来自于服务器端的本地文件系统操作请求。

由于本地文件系统子模块是继承于 FUSE 的，在该子模块中只用实现 FUSE 中给定的一些操作接口。这些接口定义在 FUSE 源码中的 fush.h 头文件的 fuse_operations 结构体中。

考虑到本系统应用环境中并非常用到该结构体里的每一个接口，本子模块对源码进行了裁剪，只实现了部分接口。

对于维护来自服务器端的操作请求，本子模块主要是在服务器端收到请求管理子模块的文件请求后，服务器端的请求处理子模块会解析出操作的目的结点号并将结点号转换成通信链接号，然后通过通信链接号将操作转发到对应的目的结点上（以上过程中的报文结构如图 3.3）。这时，远程操作转换成了本地操作，交由本地文件系统子模块去处理。本地文件系统子模块处理完后要将结果返回给服务器端。这个过程的顺序如图 3.7 所示。

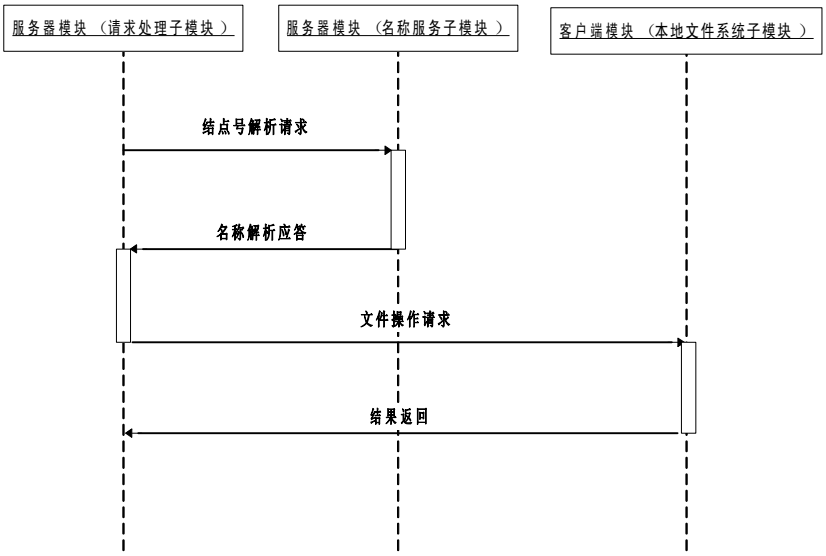

图 3.7　子模块交互顺序图

3.4　本章小结

本章主要从典型的应用场景开始，综合考虑了功能和性能方面的系统需求。在充分总结了各方面的应用需求基础上，文章首先分别对分布式文件系统的主要功能和性能进行了分析，并根据分析的结果给出具体的需求分析说明。然后，根据前文所给出的需求分析说明，从由高到低层面给出了系统的设计方案和架构草图，为系统实现提供了依据和参考。

4 分布式文件系统的实现

实现分布式文件系统，是将相关理论转化为理论化实践的关键一步。对分布式文件系统的实现，可以有助于更好的从直观上理解分布式文件系统的相关理论知识。下面就从系统的实现环境、系统的功能实现和系统的界面与测试等几个方面来具体的解释相关知识。

4.1 系统实现的环境

系统的实现环境可分为运行环境和开发环境。在运行环境中，主要是由服务器端环境和客户端环境构成的。而开发环境主要是描述的系统开发过程中所采用的开发平台和工具。

4.1.1 系统运行环境

1）硬件环境

主备服务器均采用高档服务器主机：

（1）Intel Xeon E3-1280 3.6GHz，硬盘 1TB

（2）SVGA 显示器

（3）RAM 不小于 2GB

客户端微机配置的最低要求为：

（1）Intel Core2 i3-2120，硬盘 500GB

（2）SVGA 显示器

（3）RAM 不小于 1GB

2）软件配置

主备服务器均需安装：

（1）Linux 操作系统发行版软件 Fedora 13（内核版本 2.6.33）；

（2）gcc 编译器（版本 4.3）；

客户端需安装：

Linux 操作系统发行版软件 Fedora 13（内核版本 2.6.33）；

4.1.2 系统开发环境

由于本系统主要运行环境都是 Linux 操作系统发行版软件，所以本系统的开发也主要是在 Fedora 下进行的。另外，用户态文件系统（FUSE）的源代码主要是用标准的 C 语言编写的，因此本系统的主要代码也均是采用的 C 语言开发的，编译也都是在 Linux 下的 gcc 编译器上完成的。gcc 编译器是一种高效的编译器，其编译的平均效率较其他种类的编译器要高 20%到 30%。gcc 编译器的版本也较多，基本可以兼容目前常见的硬件平台。gcc 编译器主要编译过程包括预编译、编译、汇编以及链接等过程。具体过程如图 4.1 所示。

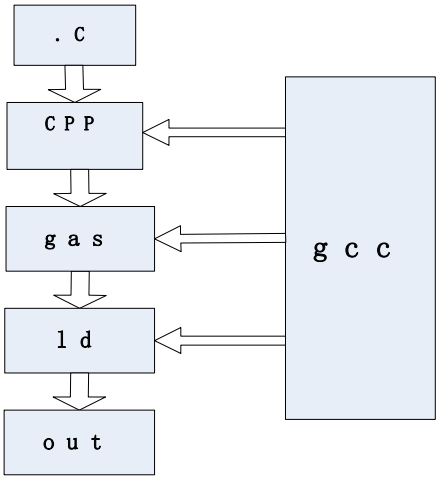

图 4.1 gcc 编译过程示意图

4.2 系统功能实现

在系统设计的设计过程中，分别从服务器和客户端对整个系统进行了设计。那么，在系统的功能实现过程中，也是从这两个方面来完成的。

4.2.1 服务器模块的实现

在实现服务器模块的环节中，可以根据设计环节中的四个子模块去进行对应的系统实现。

1）请求处理子模块的实现

请求处理子模块是服务器端所有程序中，最核心的一个部分，它主要管理着服务器与客户端的通信。主服务器初始的时候，请求处理子模块会新建一个 socket，

并 bind 到一个本地知名端口（定义为宏 MAIN_SERVER_PORT），当有客户端请求 connect 到服务器时，系统调用 accept 函数接受链接请求并通过 Linux 的 EPOLL 机制，将新链接的 iFd 加入 EPOLL 队列中。当服务器端程序请求退出时（出现服务器异常时），请求处理子模块会调用 epoll_del 将对应的 iFd 从 EPOLL 队列中删除。这里采用 EPOLL 的主要原因是实现异步多路 IO 复用，既当有多个 socket 连接建立后，请求处理子模块可以轮询每个连接而不用在某一个连接上进行循环等待，这样提高了服务器的处理效率。

对于传输协议的封装，请求处理子模块统一提供一个数据结构 struct_protocol_header 来表示协议头部分，用 struct_protocol_body 来表示协议内容部分，如图 4.2 所示。当请求处理子模块有协议发送时，可先调用内部函数 init_protocol_msg 来初始化函数，然后调用名称服务子模块的接口查询对应结点的通信链接句柄，最后通过 socket 提供的 send 函数将初始化好的 buffer 内的数据通过建立好的数据发送并等待返回。发送完成后，可调用 del_protocol_msg 来回收发送所用到的 buffer 资源。另外，为了给客户端程序提供统一的通信方式，请求处理子模块还提供一个 sendRemoteRequest 函数给客户端调用。

```
struct struct_protocol_header{
    unsigned short msg_Len;  //消 息 长 度
    char reserve_bytes[4];   //保 留 位
    char checksum[4];        //头 部 校 验 和
};

struct struct_protocol_body{
    unsigned short msg_type;           //消 息 类 型
    unsigned short msg_content_len;    //消 息 内 容 长 度
    char *msg_content;                 //消 息 内 容 指 针
    char checksum[4];                  //内 容 校 验 和
};
```

图 4.2　报文结构体示意图

2）名称服务子模块的实现

如设计中所介绍，名称服务子模块主要是就是为每个结点分配结点号，并将结点号和通信连接号对应记录下来。由于在请求处理子模块实现中，当分布式结点和服务器之间的通信是采用的 socket 的方式，则对应的连接号就是 socket 通信中被

accept 后的通信句柄号(iFd)。因为 socket 中每个有效句柄（iFd）都大于等于 0，所以这里我们采用-1 这个数来表示无连接或连接失效。当文件结点加入系统后，名称服务子模块会管理这些结点的序号与通信连接号的对应关系，在设计中是利用了表（List）这个抽象的概念来说明了这个关系的。在实现过程中，这个“表”需要有动态特性，所以我们采用了向量（Vector）这个常用的数据结构。向量这个数据结构的动态性比较好，因为其能根据目前“表”内所存储的数据多少来动态分配内存。当结点加入时，请求管理子模块将其和结点的通信句柄（iFd）传入名称服务子模块，名称服务子模块把通信句柄加入向量中并将返回的向量下标作为结点号传给请求处理子模块进行挂接的动作。当有结点名称查询请求发生时，请求处理子模块依然将句柄传入名称服务子模块中，名称服务子模块查询向量，并将向量索引返回给请求处理子模块。当结点退出时，请求处理子模块将结点通信句柄传入名称服务子模块中搜索，查找到向量中对应的句柄位置后，名称服务子模块返回对应的向量下标使对应的结点能正确的卸载，然后再将对应的向量下标对应位置的值置为无效（一般可认为无效值是-1）。

3）负载均衡器子模块的实现

在负载均衡的设计过程中，我们采用了“负载队列”的方式来管理系统内当前的负载量。而在实现过程中，“负载队列”是被具体化为数据结构中常用的数组（Array）结构的。之所以采用数据，是因为可利用数组的下标和结点号一一对应起来，方便查找对应结点的负载量的动作。当结点加入系统后，定时 10 分钟向主服务器模块发送当前负载量信息（这个动作由客户端模块完成），服务器端收到消息后，由请求处理子模块解析出消息中的负载量，然后向名称服务子模块请求解析结点号并将所得结果一并传给负载均衡子模块。负载均衡子模块得到负载量和结点号后，便可以实时刷新数组中对应位置的负载值。负载均衡子模块定时 30 分钟触发一次检查，每次检查中会把将数组中的负载量重新按照大小排入一个链表中，这个链表的结点结构包含了结点号和负载量数据。然后负载均衡子模块遍历链表结点，检查每个负载量是否大于 75%。如若大于 75%，则通知请求处理子模块通过拷贝的方式进行负载均衡等相关操作和处理。

4）备份管理子模块的实现

备份管理子模块是为了给系统管理人员提供进行例行性备份自动化的执行接口。系统的管理人员可以利用备份管理脚本进行日常的系统冗余备份等操作，如图4.3 所示。在脚本中主要有以下这些字段：

（1）命令类型

主要是指的通过何种命令进行备份。本系统目前仅支持 Copy 和 Move 命令。

（2）源结点号和目的结点号

主要指的备份过程中文件的来源和备份目的。结点号和挂接过程中的结点号相同即可。

（3）备份间隔日期

主要指的是两次备份中所间隔的日期长短（必须为整数）。若为 0 则意味着每日进行备份。

（4）备份执行时间

主要是指的每次备份过程发生在当天的什么时间。本系统默认采用的 24 小时制时间。

```
1 # 注释
2 CMD = Copy
3 # 源结点号
4 SRC = fs_node_00x
5 # 目的结点号
6 DST = fs_node_0xx
7 # 备份间隔日期 0为每日备份
8 DATE = 1
9 # 备份执行时间 24小时制
10 TIME = 12:00
```

图 4.3　备份脚本示意图

在实现过程中，本系统采用了 Linux 常用的 crontab 的方式。当备份管理子模块读到主服务器下的文件系统存在/cron 目录，备份管理子模块会逐一读取备份脚本并检查格式的正确性。若格式检查不正确，则不做任何操作。若格式检查正确，则将脚本内容转化为对应格式的 crontab 脚本，写入系统的/etc/crontab 脚本中，如图 4.4 所示，方便 Linux 启动日常服务。

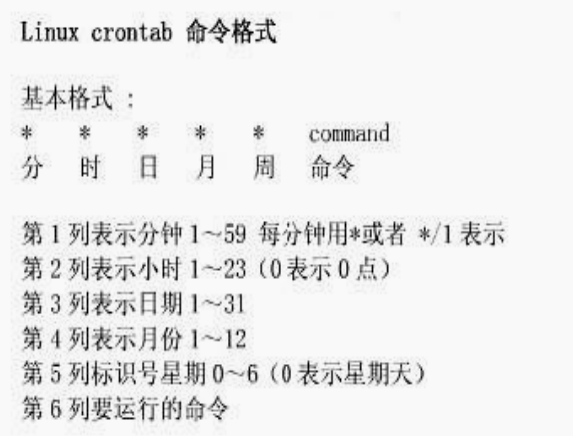

图 4.4　crontab 脚本格式图

备份管理子模块应检查脚本内容的源和目的结点号，如果源和目的结点相等则不进行备份。如果批量备份过程中，通过多次检查发现有形成循环备份的情况，则不执行相应的备份操作。

4.2.2　客户端模块的实现

客户端模块的实现对整个系统的实现也有着重要的意义。通过客户端的实现，可以使得分布式系统的远程功能得以实现。下面就将客户端的各个子模块分开来分别介绍。

（1）请求管理子模块的实现

请求管理子模块的主要作用是将来自于用户界面的消息封装成对应的协议格式发送给服务器端，并等待服务器的响应。在实现过程中，请求管理子模块和所在结点同时启动，这个启动可以将编译好的模块（Rmmd）放入 Linux 的/etc/init.d 这个目录下。在 Linux 启动过程中，这个目录下的模块都会被执行。Rmmd 启动后会新建一个 socket 链接，然后向服务器端发出 connect 请求（服务器端的 IP 地址和端口均为已知）。在服务器端接受连接请求后，Rmmd 就会进入等待状态，其所等待的是来自于用户界面的消息。当 Rmmd 收到来自用户界面分发的消息后，它会被唤醒。唤醒后，Rmmd 将消息重新进行封装（封装协议见表 3.1），并封装好的消息发往服务器端（sendRemoteRequest 函数），然后 Rmmd 又一次进入等待状态，直到被来自于服务器端的应答消息再次唤醒。这个进程状态的迁移过程主要是利用了

socket 中的 recv 函数中的旗标 MSG_WAITALL 来实现的。当结点退出时，Rmmd 会调用 close 函数来关闭这个 socket 链接。也就是说，在结点的整个生命周期过程中，这个 socket 都是可用的。

（2）本地文件系统子模块的实现

本地文件系统子模块是运行在单个结点上的文件系统进程（Fsd）。Fsd 和 Rmmd 的生命周期一样，也是随结点的启动而启动，当结点退出时，Fsd 才会退出。其主要是采用了 Linux 的 FUSE 源码，在 FUSE 源码的基础上，考虑到并非每个 FUSE 的接口都常用到，所以在本系统中主要实现了文件操作接口，如图 4.5 所示。这些接口和一般的 Linux 接口的区别主要在于对于需要文件路径的接口中（如 mkdir、rmdir、opendir 等函数），要首先判断输入路径是否在本地结点内。如果是在本地结点内的文件系统操作，则调用 Linux 的传统操作函数（如 Linux 提供的 open、mkdir 和 rmdir 等）。如果是远程文件系统操作，则需要调用请求管理子模块的接口（sendRemoteRequest 函数），利用 Rmmd 的和服务器端的通信链接，将文件操作发送到服务器端进行操作分发到对应的结点上进行操作。另外，Fsd 还负责上报本地的文件负载量到服务器端，同样也是采用表 4.1 中的协议格式。Fsd 上报负载过程中，采用了 Linux 的定时器，每隔 10 分钟定时器超时一次，然后统计本地负载量并调用 sendRemoteRequest 函数将本地文件结点负载量上报到服务器端。

```
struct fuse_operations {
    int (*mknod) (const char *, mode_t, dev_t);
    int (*mkdir) (const char *, mode_t);
    int (*rmdir) (const char *);
    int (*rename) (const char *, const char *);
    int (*link) (const char *, const char *);
    int (*chmod) (const char *, mode_t);
    int (*chown) (const char *, uid_t, gid_t);
    int (*open) (const char *, struct fuse_file_info *);
    int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);
    int (*write) (const char *, const char *, size_t, off_t,
                  struct fuse_file_info *);
    int (*statfs) (const char *, struct statvfs *);
    int (*flush) (const char *, struct fuse_file_info *);
    int (*fsync) (const char *, int, struct fuse_file_info *);
    int (*opendir) (const char *, struct fuse_file_info *);
    int (*readdir) (const char *, void *, fuse_fill_dir_t, off_t,
                    struct fuse_file_info *);
    int (*access) (const char *, int);
    int (*create) (const char *, mode_t, struct fuse_file_info *);
    int (*lock) (const char *, struct fuse_file_info *, int cmd,
                 struct flock *);
} ;
```

图 4.5　FUSE 中的文件操作接口图

4.2.3 客户端界面的实现

在系统设计过程中，我们将系统的管理划分为了命令行和 Web 化管理两种方式。这两种方式也是主要的用户界面所面向的。

（1）命令行的典型界面

提到命令行，传统的 Microsoft DOS 和 Unix 系统都是利用了这种方式为用户提供操作界面的。其特点就是使用简洁的命令和命令参数进行各种复杂的操作[43]。本分布式系统命令行的典型界面如图 4.5 所示。命令行就是在命令提示符下键入命令的用户界面（如图中的<DFS_SERVER>就是一个命令提示符）。一个利用命令行方式将一个结点的文件（如图中的 fs_node_001 下的 file1）拷贝到另外一个结点上（如图中的 fs_node_002 下的 file2），如图 4.6 所示。这样的一个跨结点的拷贝过程完全是在命令行下所进行的。在拷贝过程中会有交互式的命令提示，只有当用户确认操作，命令行才会生效。否则，命令行是不被执行的。其他的命令行管理过程均类似于此。

```
<DFS_SERVER>
<DFS_SERVER>dir
Directory of DFS_SERVER

  0     drw-          -  Nov 09 2012 17:24:44   logfile
  1     drw-          -  Nov 09 2012 18:15:27   fs_node_001
  2     drw-          -  Nov 09 2012 18:15:29   fs_node_002
  3     drw-          -  Nov 09 2012 18:15:31   fs_node_003
<DFS_SERVER>copy fs_node_001/file1 fs_node_002/file2
Copy fs_node_001/file1 to fs_node_002/file2?[Y/N]:Y

 Copy file  fs_node_001/file1 to fs_node_002/file2...Done.
<DFS_SERVER>cd fs_node_002
<DFS_SERVER>dir
Directory of fs_node_002

  0     -rw-       1048  Nov 09 2012 18:51:54   file2
```

图 4.6　命令行界面图

（2）Web 管理页面的典型界面

相较传统的命令行管理方式，Web 方式更方便管理人员通过远程来操作分布式系统。Web 管理页面是使得处于物理位置不同的管理员用户也可以通过 Internet 的 http 服务协议来管理整个分布式系统的。首先，远程管理员可以通过自己的账户和密码在登录页面登录。然后在登录成功后，管理员可以在操作页面中通过下拉列表选择操作类型和在文本框内输入各种参数后，点击提交即可完成如命令行管理一样

的操作。具体的管理页面如图 4.7 所示。(图中浏览器的地址栏中的地址为虚拟网下，主服务器被部署分配到的 IP。采用的浏览器版本为 IE8.0。)

图 4.7　管理页面图

4.3　系统测试

本系统的测试主要分为白盒测试和黑盒测试。白盒测试是代码级的测试，它是一种主要通过对虚拟环境下的被测函数输入不同的函数参数值，运行函数后去检查函数的输出是否符合预期的测试方法。不同于白盒测试，黑盒测试是一种系统级的测试，它主要是按照构造的系统测试案例逐一地去检查系统功能是否达到设计预期，这些测试案例并不直接依赖于系统的内部实现，而更多的关注所有的设计点在实现过程中是否得到了正确完整的实现[44,45]。

4.3.1　白盒测试的实现

白盒测试主要是在 Google 公司推出的 Gtest 单元测试框架环境下进行的。Gtest 的主要工作原理是将被测函数作为接口函数封装入对应的动态库里，然后在 Gtest 的主进程中对该动态库进行加载，并调用被测函数传入不同的参数并期待返回值与预期相符，如若不符则会提示该测试案例失败[46]。

对本系统的白盒测试，主要是从系统对外提供的接口函数入手，逐一对接口函数传入不同的输入参数值，然后对输出参数和函数返回值与设计过程中的预期进行

比较。通过这种比较可以发现代码实现中的一些代码级别的错误。本次测试主要针对的函数是图 4.5 中所列出的用户态文件系统的各个接口函数。测试案例均对各个被测函数的异常和正常情况进行了覆盖。在进行了完整的测试后，得出的测试结果如图 4.8 所示。

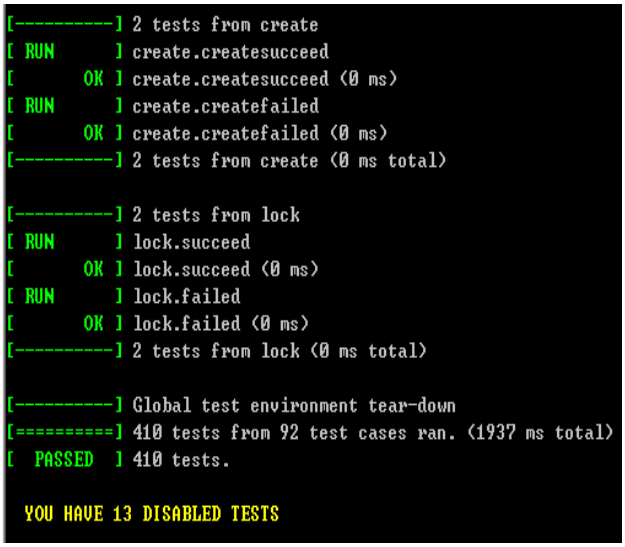

图 4.8　单元测试结果图

分析 gtest 给出的测试结果，会发现其中有 13 个测试案例暂时无法执行。这是因为 FUSE 中的一些接口在本次实现中并未真正实现。但为了测试案例的完备性和测试集的可扩展性，在本次案例中依然包括了相应的测试案例，但在实际测试中均不执行。

测试中所执行的 410 个测试案例均能正常通过测试，这说明我们的实现是符合需求和设计标准的。

4.3.2　黑盒测试的实现

黑盒测试主要是用来发现系统层面的缺陷的[47]。在项目开发完毕后，本系统可在虚拟的试验网环境下进行部署（虚拟环境的组网示意图可参考图 3.2 的系统组网图），然后我们可以根据之前第三章提出的功能和性能需求设置对应的测试案例。本次黑盒测试采用的测试方法为等价类划分和边界值测试的方法。具体的测试方法与测试结果示意如图 4.9 所示。

测试案例编号	测试需求点	测试用例内容	测试结果	备注
113	负载均衡功能	当有所有结点负载量均大于门限时，不发生负载均衡调度，但服务器端有告警发出	Passed	
114	备份管理功能	当无备份管理设置时，系统正常运行	Passed	
115	备份管理功能	当设置一个备份设置时，且备份脚本里的源结点和目的结点都存在时，备份正常	Passed	
116	备份管理功能	当设置一个备份设置时，且备份脚本里的源结点不存在时，备份正常	Passed	
117	备份管理功能	当设置一个备份设置时，且备份脚本里的目的结点不存在时，备份正常	Passed	
118	备份管理功能	当设置一个备份设置时，且备份脚本里的源结点和目的结点都存在时，将备份间隔时间设置为系统允许的最大值，备份正常	Passed	
119	性能测试	海量文件存储性能测试	Passed	
120	性能测试	多终端接入测试	Passed	
121	性能测试	分布式文件操作性能测试	Passed	
122	性能测试	可靠性性能测试	Blocked	最大值性的 测试时间较长，暂阻塞
测试结果统计				
总案例数	通过案例数	失败案例数	阻塞案例数	通过率
122	113	0	9	92.62%

图 4.9　测试结果记录图

从测试结果中可以看出，在所设置的 122 个测试案例中，本次黑盒测试执行了 112 个测试案例。在 112 个已执行的测试案例中，全部案例均测试通过。但整个测试集中，存在 9 个暂无法执行的测试案例，这无法执行的 9 个案例均是因为环境暂不具备且不影响关键使用功能的测试案例，故在此次测试中暂忽略。通过测试结果统计，我们可以发现测试通过率为 92.62%，这个测试通过率已经说明在系统级层面，本系统已经得到较为完备和准确的实现。

4.4　本章小结

本章主要从系统的实现角度，首先介绍了分布式文件系统的运行环境和实现环境；接着对照着系统内的各个功能模块逐一地去实现了系统功能；然后分别对系统的服务器界面和客户端界面给出了的简单介绍；最后对整个系统进行了比较完备的白盒和黑盒测试并给出了相应的测试结论。

5 总结与展望

本章的主要内容就是对全文的工作进行评价与总结，简单地阐述了相关研究的基本内容、方法，以及所取得的结论。同时，本章也会讨论本文在科学技术方面的意义，以及本文存在的优缺点和尚待改进和完善之处，为后续的工作指明方向。

5.1 全文总结

本文首先分析了分布式文件系统的国内外相关研究状况之后，给出一个基于Linux 操作系统的分布式文件系统的模型，然后由高到低的层次对该模型的分析与设计，将该模型在系统中得以实现并运用。

（1）从研究分布式计算的基本概念、云计算与分布式计算的关系开始，分析了分布式技术在现实中的应用，以及分布式文件系统技术的发展与现状，同时总结出了分布式文件系统的未来发展趋势。然后对现有的 Linux 操作系统的相关知识进行了简单的介绍，分析了使用 Linux 作为分布式文件系统的载体系统的可能性，给出了设计和实现分布式文件系统的一种思路。

（2）在分析了分布式文件系统的功能和性能的需求基础上，提出了一个使用Linux 系统作为基础系统的分布式文件系统模型，并对该模型的各个功能模块做了较为详细的分析与阐述。

（3）最后按照需求分析，设计并实现了一种分布式文件系统平台，同时说明了该模型在设计和实现中的主要思路与方法，证明了该系统具有我们分析的目前分布式文件系统发展的主流趋势。

综上所述，本文对正处在蓬勃发展中的分布式文件系统技术进行了较为深入的研究，提出了一个基于 Linux 的分布式文件系统的模型，并在现实中进行了设计和实现等工作。

5.2 展望

由于学术水平和研究时间的限制，本课题在对一个基于 Linux 操作系统的分布

式文件系统的设计和实现环节上还略显薄弱，仍然存在有待完善和值得继续探讨的问题。

（1）方法的优化：本分布式文件系统模型中提出了对文件结点进行负载均衡的支持模型，模型中的对每个部分使用的“负载队列”的方式进行管理，因此在模型研究方面对于队列管理的方式存在着继续研究和优化的可能。

（2）功能的完善：尽管本分布式文件系统实现了所提出的大部分功能需求，但是系统功能并未全部实现出来（如 FUSE 中的一些接口并未得到实现）。系统的功能结构还需要继续完善，提出一个可以根据现实网络环境进行自适应的系统模型还有一定的探索空间。

致 谢

随着论文工作的完成，我的硕士阶段也即将结束，回忆这两年多的时光，心中感慨万分。在这两年多里，我身边总有人给予我温暖的关怀。

首先我想要感谢的是我的导师高建生副教授。高老师学识渊博、思维开阔，引导着我走上了学术研究的道路，本篇毕业论文也倾注了高老师的不少心血，高老师严谨的学术风格和活跃的学术思维都深深影响着我，这些都必将令日后的我受益无穷。在论文完成之际，我必须向高老师致以最真诚的敬意。

此外，我还想感谢软件学院所有给过我帮助的老师们。作为软件学院的硕士生，我的学习、工程实践和论文工作得到各位的极大支持和关心。如果没有他们的无私的教导，我的学业是无法顺利完成的。

我还想感谢研二实习期间所在的文件系统二期项目组的李颂、杨波涛、江洋、晏政双等软件工程师在软件工程实践过程中热情的帮助，以及研究生阶段同寝室的张楹、卢畅等同学在这两年多来的每个阶段对我的建议和帮助。

最后我想特别感谢我的父母，他们一直都在我身后给我提供强大的前进动力。他们给予我的信任、理解和关爱伴随着我走过了过去岁月中的每一段路，他们是我的精神支柱。正是因为有了他们，才有了现在的我。

感谢其他所有给与过我帮助的老师、同学和朋友们，感谢所有人一直以来对我的关心、支持和帮助。

最后，衷心祝愿母校——华中科技大学的未来更加美好！

参考文献

[1] 匡胜徽,李勃. 云计算体系结构及应用实例分析. 计算机与数字工程,2010(3):60-63

[2] 王德政,申山宏,周宁宁. 云计算环境下的数据存储. 计算机技术与发展,2011(4):81-84

[3] 洪沙, 杨深远. 云计算关键技术及基于 Hadoop 的云计算模型研究. 软件导刊, 2010, 9(9): 9-11

[4] Alexandria. California Inventors Develop Distributed File Data Securing System. US Fed News Service, 2007(7): 32-36

[5] 康德华, 杨学良. 分布式文件系统透明性的研究. 计算机研究与发展, 1993(2): 1-8

[6] 彭国庆, 周冠宇. 云计算: 分层体系结构研究. 移动通信, 2010, 34(16): 54-58

[7] 陈康, 郑纬民. 云计算: 系统实例与研究现状. 软件学报, 2009, 20(5): 1337-1348

[8] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, Stavros Tripakis, Peter Niebert, From simulink to SCADE/lustre to TTA: a layered approach for distributed embedded applications. ACM Press, 2003: 153-162

[9] 孙宜进. 朱杰. 基于 Linux 和 SOPC 系统的 NFS 开发平台设计. 信息技术, 2005(3): 128-130

[10] Pawlak. Rough Sets. Int. J. Comput. Inf. Sci, 1982, 11: 321-356

[11] 沙伯海. 蔡海滨. 基于 Linux 下网络服务安全可靠性研究. 计算机工程与设计, 2005, 26(3): 738-742

[12] Yao, Y. Y Lingras, P. J. Interpretation of Belief Functions in the Theory of Rough Sets. Information Sciences, 1992(104): 81-106

[13] McClean S. And Scotney B. U sing Evidence Theory for the Intergration of

Distributed Databased . Int. J. of Intelligent Systems, 1997, 12: 763-776

[14] Shafer G. Perspective, on the Theory and Practive of Belief Function. Int. J. Approx. Reason, 1997, 4: 323-362

[15] 黄华, 杨德志, 张建刚. 分布式文件系统的历史与现状. 中国计算机报, 2005, 7(12): 1-5

[16] P. Schwan. Lustre. Building a file system for 1000-node clusters. In: the Linux Symposium. Ottawa, CA, 2003: 402-408

[17] 杨昕, 沈文海. Lustre 并行文件系统的发展及在气象领域的应用前景. 应用气象学报, 2008, 19(2): 243-249

[18] 张江陵, 冯丹. 海量信息存储. 北京: 科学出版社, 2003: 194-201

[19] 谭志虎, 裴先登, 谢长生. 附网存储: 一种新的网络存储方案. 电子计算机与外部设备, 1999, 23(1): 3-28

[20] 刘兆春, 李光辉, 王庆国. 并行文件系统 PVFS. 信息技术, 2005, 4: 108-111

[21] 庞丽萍. PVFS 寄生式元数据管理的设计与实现. 计算机工程, 2004, 30(20): 66-67

[22] Farley, M. SAN 存储区域网络. 孙功星译. 北京: 机械工业出版社, 2001: 385-405

[23] S. V. Anastasiadis, K. C. Sevcik. Parallel application scheduling on networks of workstations. Journal of Parallel and Distributed Computing, 1997, 43(2): 109-124

[24] 林清滢. 基于 Hadoop 的云计算模型. 现代计算机, 2010, 7(7): 114-116

[25] 许春玲, 张广泉. 分布式文件系统 Hadoop HDFS 与传统文件系统 Linux FS 的比较与分析. 苏州大学学报, 2010, 30(4): 5-9

[26] 王峰, 雷葆华. Hadoop 分布式文件系统的模型分析. 电信科学, 2010, 12(5): 95-99

[27] Leslie Lamport. Paxos made simple. SIGACTN. SIGACT News, 2001: 18-25

[28] John Dunagan, Nicholas J. A. Harvey, Michael B. Jones et al. FUSE: Lightweight

Guaranteed Distributed Failure Notification, 2007: 11-16

[29] 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: 231-244

[30] 徐高潮. 分布计算系统. 北京: 高等教育出版社, 2004: 22-27

[31] 郑欣杰, 朱程荣, 熊齐邦. 基于 Map/Reduce 的分布式光线跟踪的设计与实现. 计算机科学, 2007, 33(22): 83-85

[32] W. R. Stevens, S. A. Rago. UNIX 环境高级编程. 第 2 版. 尤晋元, 张亚英, 戚正伟译. 北京: 人民邮电出版, 2006: 367-395

[33] Tom White. Hadoop: The definitive guide. First Edition, The United States of America, O'Reilly, 2009: 5-10

[34] W. R. Stevens. UNIX 网络编程第 1 卷套接口 API. 第 3 版. 杨继张译. 北京: 清华大学出版社, 2006: 220-225

[35] 叶军. 朱华生. 嵌入式 LinuxNFS 方式下应用程序的实现. 微计算机信息, 2007, 23(8): 73-75

[36] 覃灵军, 冯丹, 刘群. 基于对象存储系统的动态负载均衡算法. 计算机科学, 2006, 33(5): 88-92

[37] 谭支鹏, 冯丹. Cfslight: 对象存储系统形式化研究. 计算机科学, 2006, 133(12): 16-18

[38] William D. Norcott, Don Capps. Iozone Filesystem Benchmark, 2006: 2-9

[39] M. Kallahalla, E. Riedel, R. Swaminathan et al. Plutus scalable secure file sharing on untrusted storage. In USENIX File and Storage Technologies(FAST), 2003: 3-7

[40] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton et al. Oceanstore: An architecture for global-scale persistent storage. In Proceedings of ACM ASPLOS. ACM, November 2000: 3-10

[41] 毛德操, 胡希明. Linux 内核源代码情景分析. 杭州: 浙江大学出版社, 2001:

117-122

[42] D. E. Denning, M. Bellare, S. Goldwasser et al. Descriptions of Key Escrow Systems, February 1997: 2-6

[43] Andrew W. Leung, Ethan L. Miller, Stephanie Jones. Scalable Security for Petascale Parallel File Systems. Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE 10-16 November 2007: 1-12

[44] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed object location in a dynamic network. In Proceedings of ACM Symposium on Parallel Algorithms and architectures(SPAA), 2002: 2-7

[45] Reed B C, Chron E G, Burns R C, et al. Authenticating network-attached storage. IEEE Micro 20, 1, 2000: 49-57

[46] 王红,刘金甫,杨小辉. 可视化测试软件平台 GTEST. 测控技术, 2005(3): 45-48

[47] 向润. 黑盒测试方法探讨. 软件导刊, 2009(1): 33-35

基于Linux的分布式文件系统的设计与实现

作者： 王亮
学位授予单位： 华中科技大学

参考文献(16条)

1. 匡胜徽,李勃 云计算体系结构及应用实例分析[期刊论文]-计算机与数字工程 2010(03)

2. 王德政,申山宏,周宁宁 云计算环境下的数据存储[期刊论文]-计算机技术与发展 2011(04)

3. 洪沙,杨深远 云计算关键技术及基于Hadoop的云计算模型研究[期刊论文]-软件导刊 2010(09)

4. 康德华,杨学良 分布式文件系统的透明性研究[期刊论文]-计算机研究与发展 1993(02)

5. 沙伯海,蔡海滨 基于Linux下网络服务安全可靠性研究[期刊论文]-计算机工程与设计 2005(03)

6. 杨昕,沈文海 Lustre并行文件系统的发展及在气象领域的应用前景[期刊论文]-应用气象学报 2008(02)

7. 刘兆春,李光辉,王庆国,柴守海 并行文件系统PVFS[期刊论文]-信息技术 2005(04)

8. 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)

9. 林清滢 基于Hadoop的云计算模型[期刊论文]-现代计算机（专业版） 2010(07)

10. 许春玲,张广泉 分布式文件系统Hadoop HDFS与传统文件系统Linux FS的比较与分析[期刊论文]-苏州大学学报（工科版） 2010(04)

11. 王峰,雷葆华 Hadoop分布式文件系统的模型分析[期刊论文]-电信科学 2010(12)

12. 郑欣杰,朱程荣,熊齐邦 基于MapReduce的分布式光线跟踪的设计与实现[期刊论文]-计算机工程 2007(22)

13. 叶军,朱华生 嵌入式Linux NFS方式下应用程序的实现[期刊论文]-微计算机信息 2007(08)

14. 覃灵军,冯丹,曾令仿,刘群 基于对象存储系统的动态负载均衡算法[期刊论文]-计算机科学 2006(05)

15. 谭支鹏,冯丹 对象存储系统形式化研究[期刊论文]-计算机科学 2006(12)

16. 王红,刘金甫,杨小辉 可视化测试软件平台GTEST[期刊论文]-测控技术 2005(03)