分类号_____ 学号 M201373340

学校代码 10487 密级_____

华中科技大学

硕士学位论文

FastDFS 负载均衡算法的改进及其在水土保持网站系统的应用

学位申请人： 周博闻

学 科 专 业：水利工程

指 导 教 师：曾致远 教授

答 辩 日 期：2016 年 5 月 28 日

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science

Load Balancing Algorithm Improvement of FastDFS and Application on Website for Soil-water Conservation

Candidate : Zhou Bowen

Major : Hydraulic Engineering

Supervisor: Prof. Zeng Zhiyuan

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

May, 2016

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过的研究成果。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：

日期：　　年　　月　　日

学位论文版权使用授权书

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、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　　在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

人类的生产生活依赖地球上的自然环境，然而在人类利用自然改造自然的过程中，也给自然环境带来日益加剧的破坏。其中水土流失是我国面临的重大环境问题之一。各个水土保持监测站的自动化采集设备将采集的数据文件收集上传到水土保持网站上，不仅可以供用户查询历史数据和当前状态，而且可以供专业人士对当地水土状况进行科学的分析和研究，制定相应的水土保持对策。因此，建立一个完整高效安全的监测数据存储系统不可或缺。

随着信息化时代的发展，水土保持监测中以图像，视频等形式出现的数据增长迅速，传统的数据存储架构已经不能满足当前的需求，需要使用分布式文件存储系统。本文的第一个工作是在分析了水土保持网站的需求和比较了国内外的分布式文件存储系统之后,选定使用 FastDFS 作为文件存储系统来存储，访问和管理数据，并安装部署在了水土保持网站的模拟实验系统上。

本文的第二个工作是在研究了 FastDFS 现有的分布式文件存储系统的负载均衡算法之后，提出了自己的改进算法。原算法只考虑服务器组的总存储空间这一个因子，改进后的算法考虑服务器组性能，服务器组现有存储容量和服务器历史连接数这三个因子。实验系统的模拟实验结果表明算法改进后系统 I/O 性能更好，负载均衡效果相比原系统更明显。

本文的研究来源于水土保持网站的实际需要。将改进后的负载均衡算法部署在水土保持网站的模拟实验系统后，测试结果表明，改进后的负载均衡算法能提升实验系统的性能。

关键词： 水土保持 FastDFS 分布式存储 负载均衡 I/O 性能

ABSTRACT

Human life and production rely on natural environment on the earth. However, during the transformation of nature, also brought increasing damage to the natural environment. The water and soil erosion is one of the major environmental problems to our nation. The automatic acquisition devices of soil and water conservation monitoring stations upload data document collection to soil and water conservation website, not only can provide historical and current monitoring data, but also can provide data to professional for scientific analysis and study in the area and make corresponding countermeasures for soil and water conservation. So, it is indispensable to construct a complete and efficient distributed file system for the monitoring of soil and water conservation.

With the development of information age, more and more data appear with picture and video forms, the traditional storage architecture can't meet our current requirement, we have to use distributed file system. After analyzing the requirement of soil and water conservation website and comparing different kinds of distributed file systems at home and abroad, we decide to use FastDFS as file system to save, access and manage data, and deploy the simulate experimental system on the soil and water conservation website.

After studying the current load balancing algorithm of the FastDFS, we propose an improved algorithm for FastDFS. The current algorithm only consider the total storage space of the server group, the improved algorithm consider the performance, the existing storage space and the historical linking number of the server group. The simulation experiments show that the load balancing effect and the I/O performance of experimental system performs better after being improved.

The study in this paper comes from the actual requirement of soil and water conservation website. After deploying the improved algorithm for FastDFS on soil and water conservation simulate experimental website, the experiments show that the experimental system performs better after being improved.

Keywords: Water and Soil Conservation FastDFS Distributed Storage Load Balance I/O Performance

目 录

1 绪论

1.1 研究背景和意义

土壤和水分是地球上生物生存的根本保障。人类的生产生活都离不开土壤和水分，是人类延续下去的根本条件。然而人类活动对自然环境的破坏日益加重，已经产生了影响人类正常生活甚至生存的灾难性问题，而水土流失就是其中的环境问题之一。短期的水土流失不会产生太大的影响，容易被忽略，然而长期的水土流失产生的土壤侵蚀，可能引发各种巨大的灾害，如泥石流，荒漠化等，直接危及生命财产安全，因而水土流失是关系到人类长远生存的重大问题[1]。

二十一世纪以来，我国经济社会文化等各方面的实力显著增强，社会信息化和现代化程度明显提高。与此同时以信息化带动产业升级的观念已逐渐在各个地区、部门得到广泛的重视。为此，水利部确立了以水利信息化带动水利现代化的发展思路[2]。

本文的研究目的是随着信息时代的发展，面对在水土保持监测工作中出现的越来越多的图像数据，以及访问量和数据量的大大增加，传统的网站文件存储架构不足以满足当前需要，系统上传和下载速度缓慢，需要使用分布式文件存储系统来应对这个问题。本文通过比较适合处理海量图像和视频数据的 FastDFS 分布式文件系统来存储，访问和管理监测水土保持监测数据，并改进其负载均衡算法提高系统运行效率，方便相关专业人士和有关部门高效安全的获取相关数据资料，对当地水土状况进行科学的分析和研究，制定相应的水土保持对策。同时相关资料也对公众开放查阅，对于争取民众的支持，推动水土保持工作的进一步发展具有重大的意义。

随着互联网时代发展和移动互联网的爆炸式增长，提供高质量的网络服务已经成为每个网站运营方所需要解决的首要问题，分布式文件系统和负载均衡技术为了解决这个问题应运而生[3][4]。正因如此，分布式文件系统和负载均衡技术一直是信息科学探索的重点，吸引众多科学家和工程技术人员进行研究。

1.2 国内外研究现状

1.2.1 水土保持网络的国内外研究现状

随着信息技术发展和水土保持网络系统建设的不断前进以及世界各国对水土保持监测网络信息化的要求不断增长，水土保持网络信息化程度被各种新技术大大提高，同时也为水土保持科学的发展带来了便利和快速前进的动力。随着信息技术的爆炸式发展，欧美等发达国家的水土保持监测网络实现了高度的信息化，建立了坡面径流小区、沟道控制站、气象监测等监测预报信息化网络。发达国家在水土保持监测网络信息化发展过程中积累了大量的宝贵经验，在监测网络信息化程度得到了较大的提高的同时也培养了大批水土保持监测网络信息化领域的专业人才。而且发达国家的水土保持信息化监测设备制造水平较高，信息技术发达，因而其水土保持监测网络信息化的发展整体上大大领先于我国[5]。

我国也建立了覆盖全国范围的水土保持信息化网络。该网络以水利部水土保持监测中心为数据中心，以流域机构水土保持监测中心和省级监测总站为核心节点，利用公众网络和水利部系统部分现有网络，采用先进的遥感、地理信息系统、全球定位系统及计算机网络信息处理等技术形成覆盖全国的信息采集、传输、处理和发布系统网络体系，为全国水土保持信息化网络平台建设奠定了基础[6]。

目前我国水土保持网络数据库建设发展迅速，许多省（自治区、直辖市）在水土保持基础信息资源建设中取得了丰硕的成果。如北京市水土保持监测总站完成了北京市山区小流域水土保持基础信息库，并在 1:1 万 DEM 基础上建立了北京市山区水土保持单元—小流域地理信息数据库，完成了规范化的小流域编码，该编码已被北京市计委采纳，成为北京市基础空间信息标准，为北京市水土流失监测网络数据库的发展奠定了良好的基础。水土保持网络数据库建设为国家水土保持生态建设规划和水土保持科学研究提供了全面、丰富、翔实的数据[7]。

最初我国的水土保持监测网站因为只对监测管理人员和专业研究人士开放，访问量非常的小，故往往只配备一台服务器。通常是应用程序、数据库、文件等所有

资源都放在这台服务器上，采用传统的 Linux+Apache+MySQL+PHP 网站技术架构。随着水土监测数据信息量增长和网站对公众的开放，越来越多的监测数据导致存储空间不足，越来越多的用户访问导致网站的性能越来越差，一台服务器已经不能满足当前的需要，所以必须将应用和数据分离。此时网站使用三台服务器：应用服务器，文件服务器和数据库服务器。其网站基本架构如图 1-1 所示：

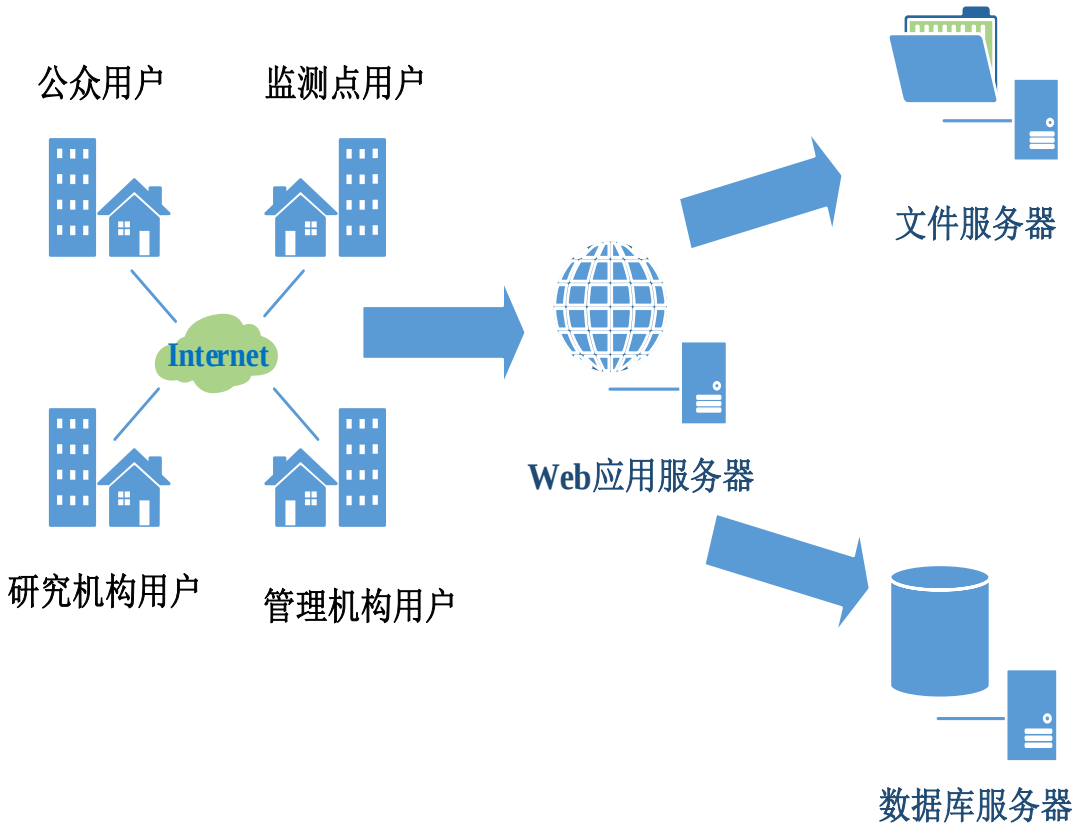

图 1-1　传统水土保持网站的基本架构

应用和数据分离以后，不同功能的服务器在水土保持网站中扮演不同的角色，水土保持网站的并发处理能力得到很大的提高，数据存储空间增大，网站的业务得

到进一步拓展。但是随着水土保持监测数据类型不断增加和网站的用户和访问量进一步增加，传统的水土保持网站的架构不能满足业务持续增长的需求，于是使用了很多前沿的信息技术代替传统的水土保持网站技术架构，特别是使用分布式技术和负载均衡技术。其网站架构如图 1-2 所示，其中 Web 应用服务器、数据库服务器、缓存服务器、文件服务器可以根据实际需求选择分布式或者非分布式架构。当 Web 应用服务器不使用分布式时，不需要负载平衡器。

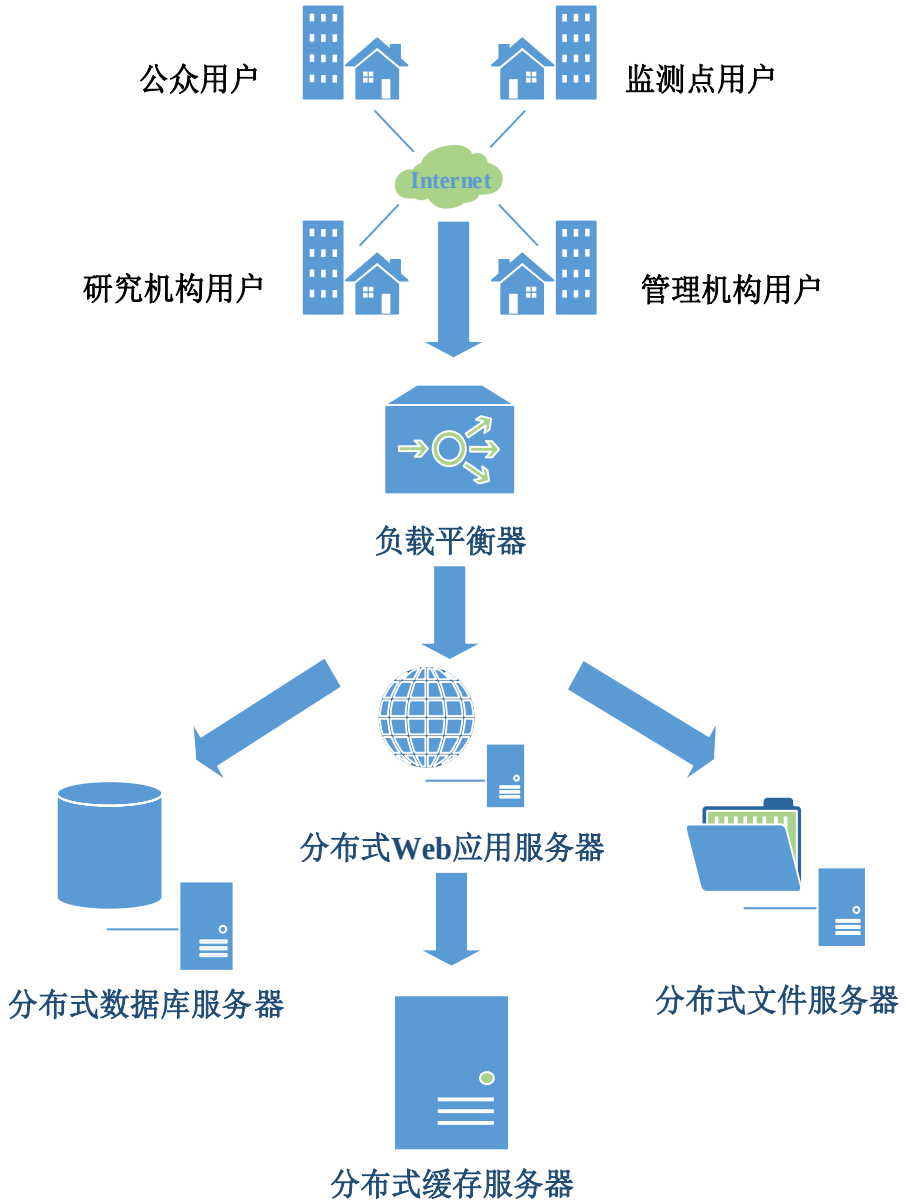

图 1-2　现代水土保持网站的基本架构

针对目前在水土保持监测网络中的实践，水土保持监测网络出现了以下三个新问题：

（1）现在各个水土保持自动监测站新配备了视频监测系统定时上传当地水土保持自动监测站的监测图像，使该网站出现了新的监测数据类型和数据流量大大增加。传统的水土保持网站架构不能满足当前监测数据存储需求。

（2）由于该系统对外开放，特别是允许普通民众通过综合查询系统对水土保持监测信息进行查询，使网站的访问压力大大增加，使网站出现了应对大并发访问的需求

（3）由于访问量和数据量的大大增加，以前传统的网站架构不足以满足当前需要，系统上传和下载速度缓慢。

面对这三个新问题的挑战，解决方案是在现代水土保持网站中使用分布式文件服务器。分布式文件系统有可扩展、低成本、高性能、易用等这些特性，以及存储容量大，能够应对高并发场景等这些优点。因为分布式文件系统有这些特性和优点，所以使用分布式文件系统可以解决在水土保持监测网络工作中的实践中出现的这三个新问题。

1.2.2 分布式文件系统的国内外研究现状

最早的分布式文件系统出现在上世纪 70 年代，通过这么多年的发展，分布式文件系统在系统规模、架构性能和可扩展性等方面发生了翻天覆地的改变[8]：

（1）第一代分布式文件系统（1980~1990）

第一代分布式文件系统的代表是 NFS[9]系统和 AFS[10]系统。因为受当时硬软件条件的限制，当时的分布式文件系统更注重于数据的可靠性和访问的性能。NFS 和 AFS 在系统架构上做出了意义重大的探索，以后的很多分布式文件系统都借鉴了他们制定的协议和使用的相关技术。

（2）第二代分布式文件系统（1990~1995）

上世纪 90 年代研究人员开发了分布式文件系统 XFS[11]，解决了分布式文件系统

在广域网上缓存的问题，将分布式文件系统应用到范围更大的广域网之中。后来出现的 TigerShark[12]分布式文件系统主要针对大规模多媒体应用，其创新之处在于资源的预留和资源调度策略的优化。

（3）第三代分布式文件系统（1995~2000）

这个阶段硬件技术和网络技术突飞猛进的发展使得单位存储的成本大大降低，然而数据总线带宽和磁盘速度的增长无法满足当时应用对数据带宽的需求。为了克服存储子系统这一整个系统的瓶颈，出现了多种不同系统架构和充分利用网络技术的分布式文件系统，比如 Google 公司的 Google File System（GFS[13]）、SGI 公司的 CXFS[14]、SUN 公司的 QFS[15]以及惠普公司的 DiFFS[16]。这个时期分布式锁、负载均衡和缓存管理技术的应用使分布式文件系统的动态性和可靠性大大增强。

（4）第四代分布式文件系统（2000 年以后）

Panasas 公司的 PanFS[17]、IBM 的 StorageTANK[18]、蓝鲸文档系统（BWFS[19]）是这个时期的分布式文件系统的杰出代表。各种应用对分布式文件系统提出了大容量、高性能、高可用性以及按需服务的要求。各种优秀的分布式文件系统在信息时代的浪潮中百花齐放，根据它们自身架构和各自优缺点的不同而被应用在其最适合的场景。

随着分布式系统的发展，负载均衡算法也得到了人们的重视和深入的研究。除了基本的负载均衡算法之外，最新研究和发展的负载均衡算法大多数采用图论模型和启发式算法相结合进行负载均衡调度，使用网络流的方法求解。Stone 利用上述理论提出了考虑计算资源和通信代价情况下最优分配策略[20]。Bokhari 进一步将其用于解决无贿赂通信图的任务分配问题[21]。Towsley 将通信图推广到串并联结构[22]。Yadav 对静态模型进行推广，使之可以用于多个处理器的情形[23]。

由于现有的各种负载均衡策略自身的复杂性，直接比较其性能的优劣有一定的难度，需要很好的实现各种策略的算法，然而在并行机器上实现这些算法很有挑战性 [24][25]。另外一个研究负载均衡策略的方法是不通过实际的性能测试仅仅从理论上进行分析、计算和比较。

1.3 论文的主要内容和结构

本文的主要内容是改进 FastDFS 现有的负载均衡算法，提高其性能并将其应用在水土保持网站上，使得系统处理图像数据更加高效。方便了相关专业人士和有关部门高效安全地获取相关数据资料，对当地水土状况进行科学的分析和研究，制定相应的水土保持对策提供数据支撑。

本文中通过分析水土保持网站的实际需求和对各种分布式文件系统特点及应用场景的比较，选择了 FastDFS 作为水土保持网后台应用的分布式文件系统。通过研究其现有的负载均衡算法的源码，找出现有负载均衡算法的不足和需要改进的地方，在现有的负载均衡算法的基础上提出改进算法，编写新算法的代码并进行了相关的实验测试，验证新算法对系统性能的改进。

本文一共分为五章：

第一章是介绍论文的研究背景，研究目的和研究内容及主要工作，探讨了相关的研究现状和研究的意义，最后给出了论文的结构安排。

第二章是介绍分布式文件系统和负载均衡技术，并重点介绍 FastDFS 的架构和工作原理。

第三章的主要内容是对 FastDFS 负载均衡算法的研究和改进。本章首先研究 FastDFS 原有负载均衡算法，找出原有负载均衡算法的不足和需要改进的地方，在原有负载均衡算法的基础上提出改进算法并对改进后的负载均衡算法进行实验测试并分析实验结果，以验证新的负载均衡算法的能够提高测试系统负载均衡性能和 I/O 性能。

第四章首先分析水土保持监测网站需求，然后给出选择 FastDFS 作为其分布式文件系统的原因以及应用 FastDFS 作为其文件系统的水土保持网站的系统架构。最后部署 FastDFS 并使用新算法在水土保持网站模拟实验系统上进行测试以验证新算法的实际效果。

第五章是总结本文的主要内容，对自己的工作进行总结然后对未来的工作进行展望。

2 分布式系统和负载均衡算法的相关理论研究

2.1 分布式文件系统的概念

分布式系统是建立在网络上支持分布式处理的软件系统。其硬件是由多个相互连接的计算机节点组成，它们在整个系统的协调和控制下执行同一个任务，对物理空间上集中的硬件，数据和程序的依赖性很小。这些硬件、数据和程序在地理上是可以分散的，没有相邻的必要性。在一个分布式系统中，一组硬件上相互独立的计算机给用户的感觉是它们是一个统一的集合体，而不是许许多多硬件上相互独立的计算机节点。分布式系统包括分布式操作系统、分布式数据库系统、分布式文件系统等[26] [27][28]。

分布式系统的特点是高度的透明性和内聚性。透明性是指对用户来说，每一个计算机节点都是透明的，无法看出这个节点是远程还是本地，数据到底存在于哪个节点或者进程在哪个节点执行。内聚性是指每一个计算机节点都是相互独立和高度自治的，每个节点都有本地的数据管理系统[29]。

分布式存储系统是一个由大量普通 PC 服务器通过网络相互连接，作为一个整体提供存储服务的系统。与传统的集中式存储相比，其优点在于数据存储更为可靠和安全，而且可扩展性使其不会像集中式存储一样成为整个系统的瓶颈。这些优点决定了分布式存储系统必然会取代集中式存储作为必选的数据存储方式[30]。分布式存储系统有如下几个特性：

（1）高可靠性：分布式存储系统具有完备的文件备份功能，各个计算机节点物理上的分散使其天生就有完美的容灾能力。

（2）可扩展性：分布式存储系统扩展方便而且几乎没有系统上限，可以根据实际需要增加和减少服务器数量，服务器集群可以扩展到几百甚至上千台规模。

（3）高性能：不管是对于单个节点或者整个集群，都需要分布式存储系统具有高性能。系统会选择对于用户自身情况来说性能最佳的节点提供文件上传和下载服务，这是集中式存储不可能做到的。

（4）易用性：对于用户来说能够方便地从分布式储存系统获得使用接口。对于开发人员来说分布式文件系统具备完善的运维和监控工具，并且能方便地和其它系统进行集成。

（5）低成本：由于分布式存储系统具备自动容错和负载均衡能力所以使其能部署在普通 PC 机器上，硬件成本非常低。而且较高的自动化程度可以实现自动运维，减少了人工成本。

Facebook、Amazon、YouTobe、Google、阿里巴巴等互联网公司的爆炸式发展开创了大数据和云计算这两大热门领域。不管是各种互联网应用还是大数据和云计算，其背后都是高可靠性、可扩展性、高性能，易用和低成本的分布式存储系统。其中具有代表性的是 Google 的 GFS 系统、Facebook 的 Facebook Haystack 系统、淘宝的 TFS 系统以及 DangaInteractive 公司的 MogileFS 系统。

2.2 企业分布式文件系统的简介

计算机科学家和相关工程技术专家虽然从上世纪 70 年代就开始研究分布式系统，但是直到这个世纪互联网大数据应用的兴起才使分布式系统登上工程应用的舞台。不同的应用场景需求催生了各种不同的企业分布式文件系统，但是无论是怎样的企业分布式文件系统，都具有这样两个特点：成本低和规模大。可以这么说， Google 和 Amazon 等互联网大数据公司重新定义了大规模分布式文件系统。下面简单的介绍下 GFS 系统、Facebook Haystack 系统和 MogileFS 系统。

2.2.1 GFS 系统

分布式文件系统中最出名的莫过于 Google File System（GFS）。GFS 文件系统是为了支持高效的海量信息存储和检索而构建在廉价服务器上的分布式文件系统。在 GFS 中服务器故障被视为正常的现象，由于设计良好的软件容错机制，在保证系统的可靠性的同时，仍表现了出色的性能和可用性，并大大降低了系统的硬件成本。GFS 系统可以被分为 GFS Master、GFS Chunk Server 和 GFS 客户端[31]，其整体架构

如图 2-1 所示：

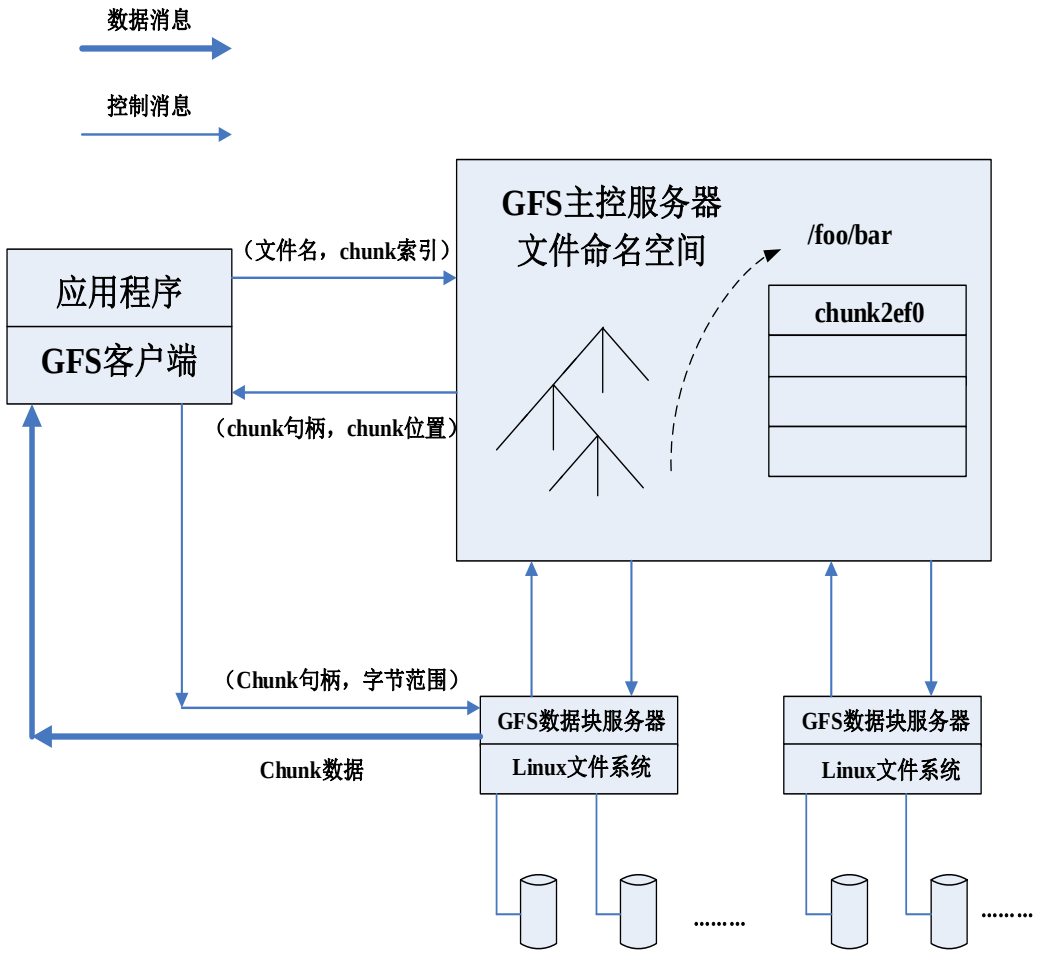

图 2-1　GFS 的架构

存储的大文件被系统分割成大小约为 64MB 的数据块（chunk），数据块在被主控服务器创建时由其分配一个 64 位全局唯一的 chunk 句柄，并通过主控服务器（维护系统的元数据实现全局控制、数据管理、 负载均衡和垃圾回收等操作。GFS 是一个具有良好扩展性能并能够在软件层面自动处理各种异常的系统。由于软件层面能够做到自动化容错，在低层可以采用廉价的错误率较高的硬件，比如廉价的 SATA 盘，这大大降低了云服务的成本，在和其他厂商竞的争中表现价格优势[32]。

2.2.2　Facebook Haystack 系统

Facebook 目前存储了 2600 亿张照片，总大小为 20PB，通过计算可以得出每张

照片的平均大小为 20PB/260GB，约为 80KB。用户每周新增照片数为 10 亿（总大小为 60TB），平均每秒新增照片数为 $10^9/7/40000$（按每天 40000S 计）[33]。其系统架构如图 2-2 所示：

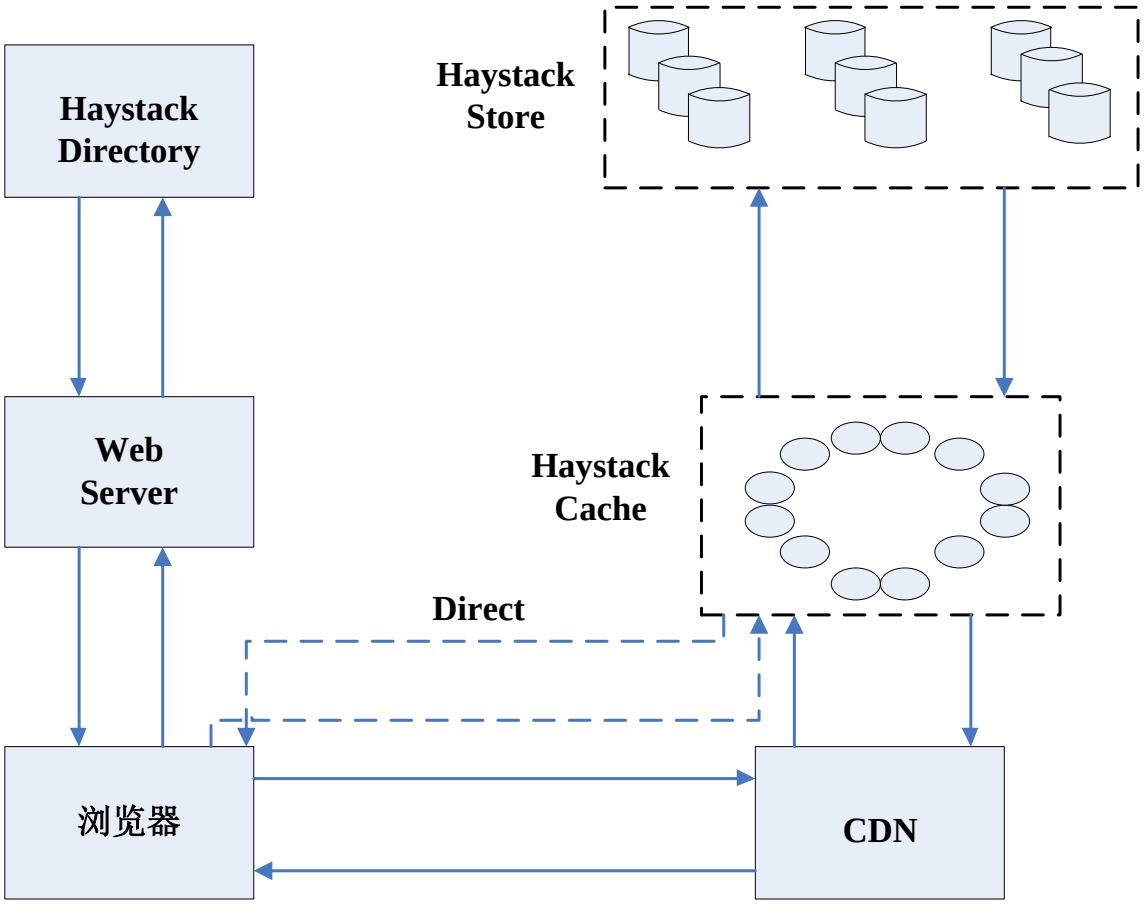

图2-2 Facebook Haystack系统架构

Facebook Haystack 的设计思路为多个逻辑文件共享一个物理文件。Facebook Haystack 系统主要由 3 个部分组成，包括目录（Directory）、存储(Store)和缓存（Cache）。Facebook Haystack 的文件读取大致流程为： Web Server 请求 Haystack 目录构造一个 URL：http://<CDN>/<Cache>/<Machine id>/<Logical volume,Photo>, 后续根据 URL 的各个部分的信息依次访问 CDN、Haystack 缓存和后端 Haystack 存储。Haystack 目录构造 ULR 时可以省略< CDN >以达到用户不经过 CDN 而直接访问 Haystack 存储

的目的。Facebook 存在对于 CDN 提供商过于依赖的问题而 Haystack 缓存正可以解决这个问题，Haystack 缓存为最近增加的照片提供缓存服务。

2.2.3 MogileFS 系统

MogileFS[34]是一个高效的分布式文件存储系统，其广泛应用在许多高负载站点上，例如我们熟知的豆瓣、大众点评、一号店、搜狗等网站上，为这些公司的网站存储着大量的图片数据。其架构如图 2-3 所示：

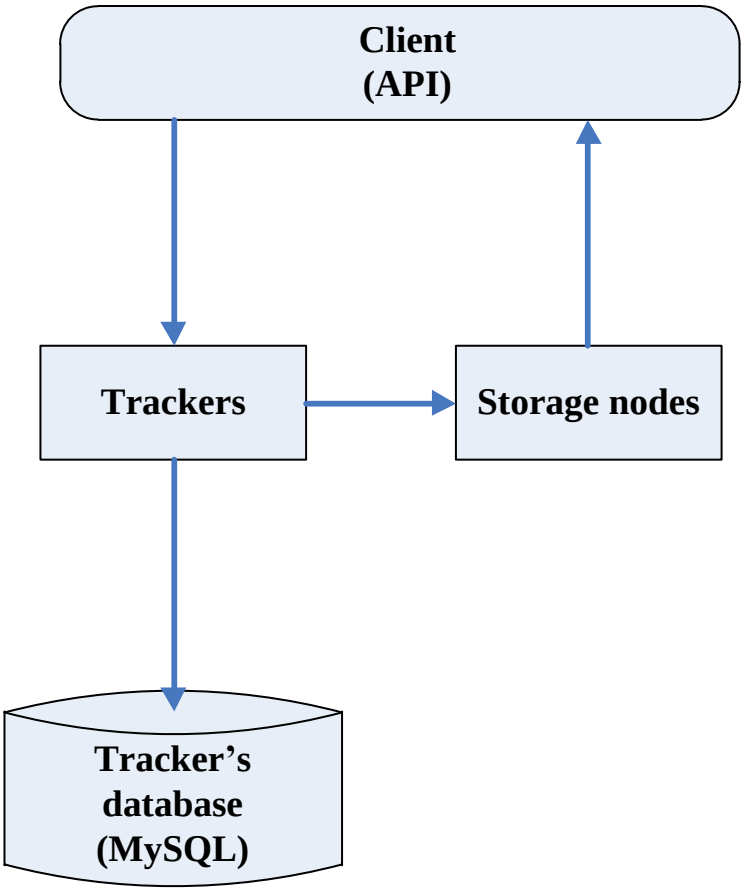

图 2-3 MogileFS 的架构

Tracker 是 MogileFS 的核心部分，在整个系统里起调度作用，文件的复制、删除、查询、 监控指令都由其发出。Tracker's database 存放了 MogileFS 的全部的元数据，元数据保存了数据的命名空间和数据存放的位置。Tracker's database 由 Tracker 调用和管理，如果存放元数据的 Tracker's database 服务器故障则整个 MogileFS 将陷入瘫

痰状态，所以 Tracker’s database 一般是拥有至少一个备用节点的高可靠性集群。Storage Node 是实际文件存放的位置，可以在上面对文件做增删改查等操作。访问文件时客户端首先和 Tracker 通信，然后 Tracker 查询 Tracker’s database 中保存的数据的命名空间和数据存放的位置后返回给客户端一个可用的 Storage Node 的地址，客户端向 Storage Node 读取数据，如果传输过程失败则会重新发送请求或者更换其它可用的服务器。

2.3 FastDFS 系统的简介

2.3.1 FastDFS 的概述

FastDFS 是由阿里巴巴公司开发的一款开源的分布式文件系统。该项目的开发从 2008 年 4 月开始，同年 7 月推出了第一个版本 V1.0，至今已经升级到 V5.05 稳定版本。FastDFS 类似于 GFS，由纯 C 语言实现，支持 Linux 等 UNIX 操作系统。在国内有非常多的公司在实际的生产环境中使用 FastDFS，其中包括 UC、支付宝、京东商城、赶集网等著名互联网公司。某大型的网盘公司的 FastDFS 集群存储 group 有 400 个，服务器总数量有 800 多台，存储总容量超过 6PB, 文件总数量达到 1 亿多个，而且还在持续增长之中。

2.3.2 FastDFS 的基本架构

FastDFS 的整体架构如图 2-4 所示，FastDFS 由客户端（Client）、跟踪服务器（Tracker Server）和存储服务器（Storage Server）这三个部分组成：

(1) 客户端（Client）: Client 是用户业务请求发起的位置，通过 TCP/IP 通信协议可以和 Tracker Server 以及 Storage Server 直接通信。

（2）存储服务器（Storage Server）：Storage Server 是 FastDFS 存储文件的位置。由于 FastDFS 不存在文件分块机制所以 Storage Server 上传文件和客户端文件系统中的文件存在一一对应的关系。Storage Server 采用分组的方式，不同的组之间是平行

关系不会相互通信，其系统总容量即为所有 Storage Server 组容量之和。

（3）跟踪服务器（Tracker Server）：Tracker Server 在整个系统主要起调度作用，调用文件以负载均衡的方式进行访问。所有 Tracker Server 之间的关系都是平等的，可以根据服务器的压力情况随时增加或者减少。所有存储组及其组内服务器的状态信息都记录在 Tracker Server 的内存中，Tracker Server 通过扫描内存中的状态信息对 Storage Server 和 Tracker Server 的访问进行应答。可以看出相比 MogileFS 中的 Tracker 来说，FastDFS 的 Tracker Server 更为简化，不存在查询文件索引的过程，不仅占用内存空间小而且使 Tracker 不会成为系统性能瓶颈。

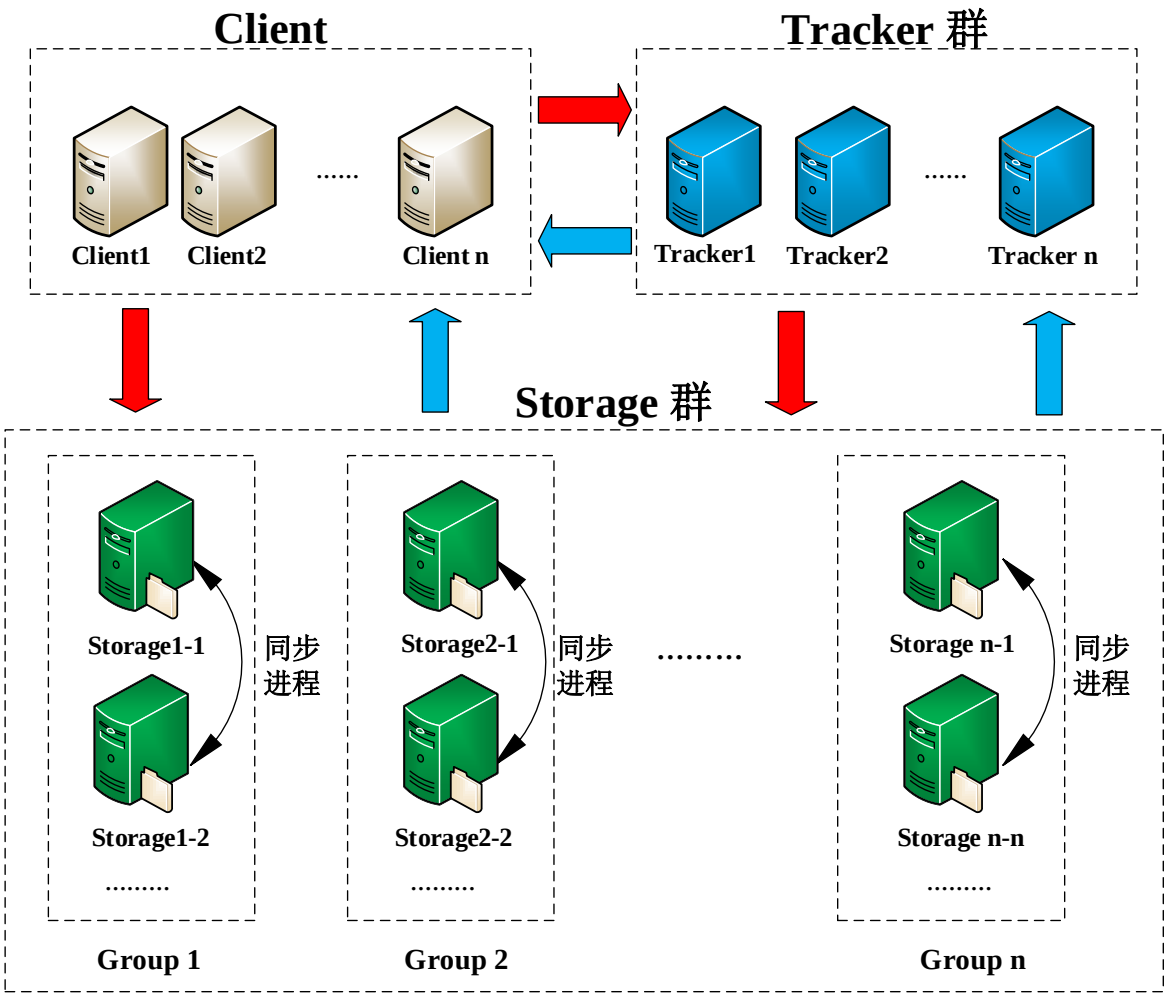

图 2-4 FastDFS 的架构

FastDFS 没有 Tracker's database 不需要存储文件的索引信息，那么它是如何做到

这一点的呢？上传文件时，Storage Server 会生成一个包含该组的组名和存放文件的文件名文件 ID 返回给 Client，通过该文件 ID, Client 可以直接根据该文件 ID 精确定位到文件在 Storage Server 中存储的位置，一个文件 ID 示例为：

Group 2/M01/02/0B/wKiDBbRxx3GOCCCCCCCCAiSQgyg98164.h

其中 Group 2 是 Storage Server 的组名，M01 是磁盘名，02/0B 是目录，wKiDBbRxx3GOCCCCCCCCAiSQgyg98164.h 是文件名。

2.3.3 FastDFS 的工作原理

（1）FastDFS 的文件上传流程如图 2-5 所示：

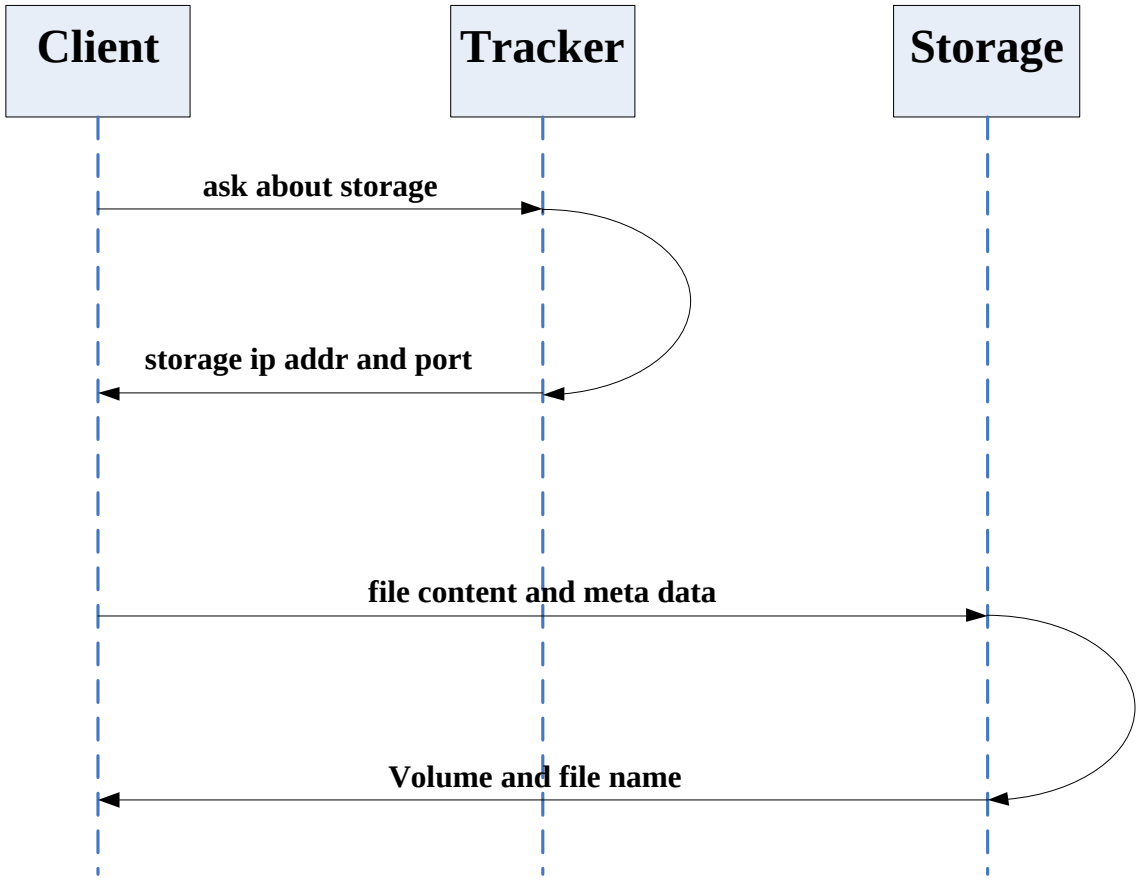

图 2-5 FastDFS 的文件上传流程图

Step 1：首先 Client 会向 Tracker Server 发送文件上传请求，询问可用的 Storage Server。

Step 2：Tracker Server 收到请求以后会根据上传文件的情况和 Storage Server 的负载

的情况经过负载均衡后返回 Client 一个可用的 Storage Server 的 IP 和端口号。

Step 3：Client 直接和 Tracker Server 返回的 Storage Server 进行通信，上传文件内容和其元数据。

Step 4：上传完成后 Storage Server 返回给 Client 其文件 ID。

（2）FastDFS 的文件下载流程如图 2-6 所示：

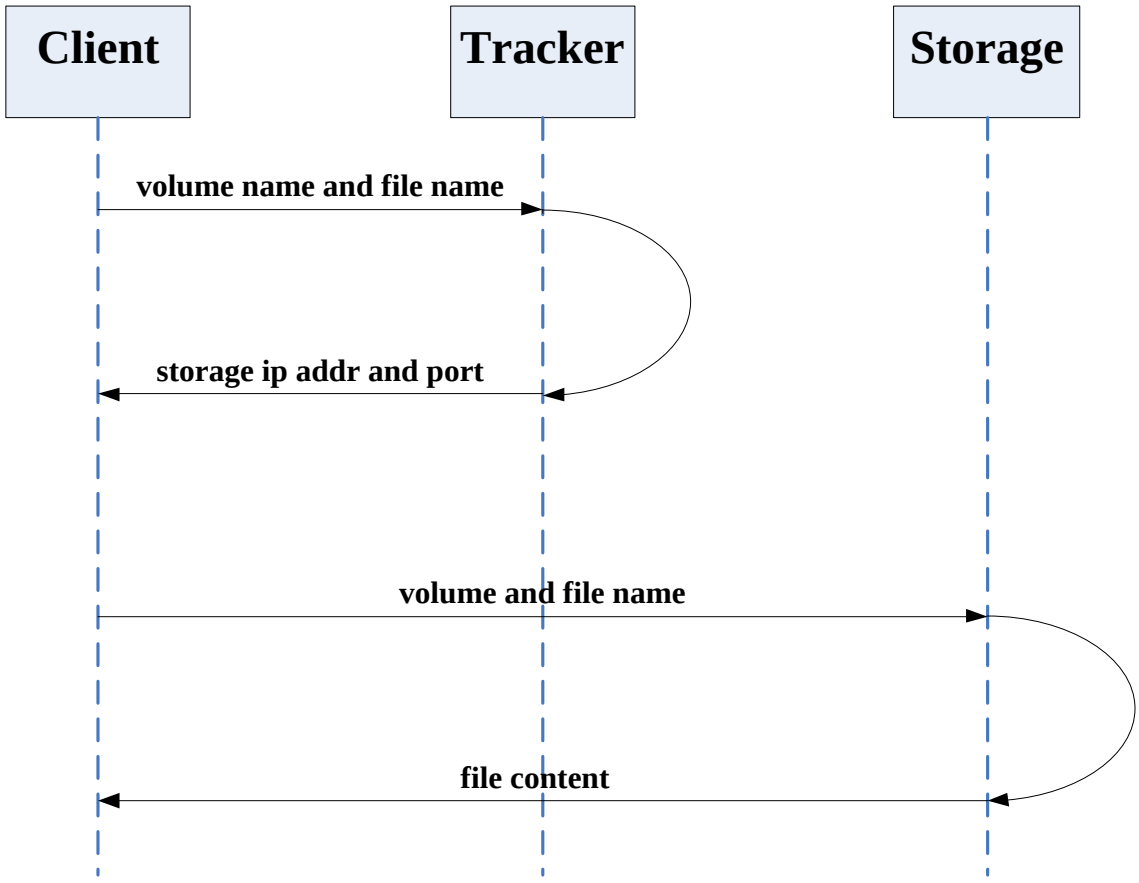

图 2-6　FastDFS 的文件下载流程图

Step 1：首先 Client 会向 Tracker Server 发送要下载的文件的文件 ID，询问能从哪个 Storage Server 下载指定的文件。

Step 2：Tracker 会返回给 Client 一台可用的 Storage Server 的地址和端口号。

Step 3：Client 会通过 Tracker Server 返回的地址和端口号和这台 Storage Server 直接建立连接。

Step 4：Storage Server 向 Client 发送数据，完成数据下载过程。

2.4 负载均衡技术的简介

2.4.1 负载均衡分技术概述

负载均衡问题可以简单描述为负载被分配到各服务器节点的过程。在一个多节点的系统里，可能会出现下面的状况：一些节点负载过高而然另外的一些节点负载过低甚至闲置。负载均衡的目的就是最大限度的调节各个节点的负载，将过载节点的负载调节到有多余处理能力的负载之上，使得所有的服务器节点既不出现闲置状况又不出现过载的状况[35]。

2.4.2 负载均衡算法的分类

负载均衡算法按照算法与系统本身的负载变化是否有关可以分成静态负载均衡算法和动态负载均衡算法。

（1）静态负载均衡算法

静态负载均衡算法事先就确定算法所有的执行流程，和系统在运行中服务器节点本身的负载情况无关。该类的典型算法有：轮叫调度算法、加权轮叫调度算法等。

（2）动态负载均衡算法

动态负载均衡算法会按照服务器节点的负载变化而调整算法的执行流程，选择出通过算法计算后最合适的服务器节点来处理请求。动态负载均衡算法会根据服务器节点的负载状况进行动态调整，例如最小连接调度，遗传算法等。研究表明，在相同的硬件配置下动态负载均衡算法的效率要比静态负载均衡算法的效率高20%~50%。动态均衡负载算法可以分为混合式，集中式和分布式三类[36]。

混合式算法目前正在研究之中，其大体思路是将节点划分为多个相互独立的组，按这些组的特点使用集中式或者分布式算法。

2.4.3 典型的负载均衡算法

(1)轮叫调度算法[37]

该算法会按照一定次序依次调度每台服务器来响应请求，当最后一个服务被使用后，那么下一次的请求就由第一台服务器来响应，就像一个轮盘一样周而复始，是一种无状态的静态负载均衡算法。这种算法的优点在于实现起来非常简洁明，不需要记录服务器的当前状态直接依次调用。其缺点也是显而易见的，当各个服务器性能和负载状况相差比较大的时候，这样简单的依次调用会导致服务器之间负载不均衡。所以轮叫调度算法比较适合处理各个服务器性能相同的情况。

（2）加权轮叫调度算法[38]

在轮叫调度算法基础上提出了加权轮叫调度算法。每个服务器会根据其性能被赋予一个权值，然后按照权值的大小由高到低依次排列，该算法会根据这个排列次序依次调度每台服务器来响应请求。加权轮叫调度算法在一定程度上解决了各个服务器之间性能不同的情况，但是这种算法仍然是一种简单无状态的静态负载均衡算法。

（3）最快算法[39]

该算法会根据服务器的响应时间的长短来分配连接，下一个响应会被分配到目前响应时间最短的服务器，所以响应请求快的服务器会得到更多的连接数。该算法在跨不同网络环境例如由互联网到移动网络的情景下特别有用。

（4）最少连接数算法[40]

该算法会根据服务器当前的连接数多少来该分配连接，当前连接数最少的服务器会处理当前服务请求。虽然这是一个动态的负载均衡算法但是和轮叫调度算法一样当各个服务器性能相差比较大的时候，这样简单的按照连接数多少来调用会导致负载不均衡。所以最少连接数算法也比较适合处理各个服务器性能相同的情况。

（5）最低缺失算法[41]

在该算法中，系统的中心节点会记录所有服务器历史以来处理请求的数量，历史处理请求最少的服务器会被分配下一个服务请求。

2.5 本章小结

本章的主要内容是介绍了分布式文件系统和负载均衡技术的一些基础知识。首先简单地介绍了分布式文件系统的概念特点和一些常见的企业分布式文件系统；然后详细介绍了 FastDFS 的基本架构和工作原理；最后介绍了负载均衡技术的概念和分类，以及一些典型的负载均衡算法。

3 FastDFS 的负载均衡算法的改进

3.1 FastDFS 现有负载均衡算法

3.1.1 FastDFS 现有 Storage Server 组的选择思想

通过阅读 FastDFS 的设计使用文档发现其关于负载均衡的源代码在 Tracker 模块的 Tracker_service.c 和 Tracker_mem.c 文件中。阅读源代码后可知 FastDFS 现有的 Storage Server 组选择流程可以用下图表示，如图 3-1 所示：

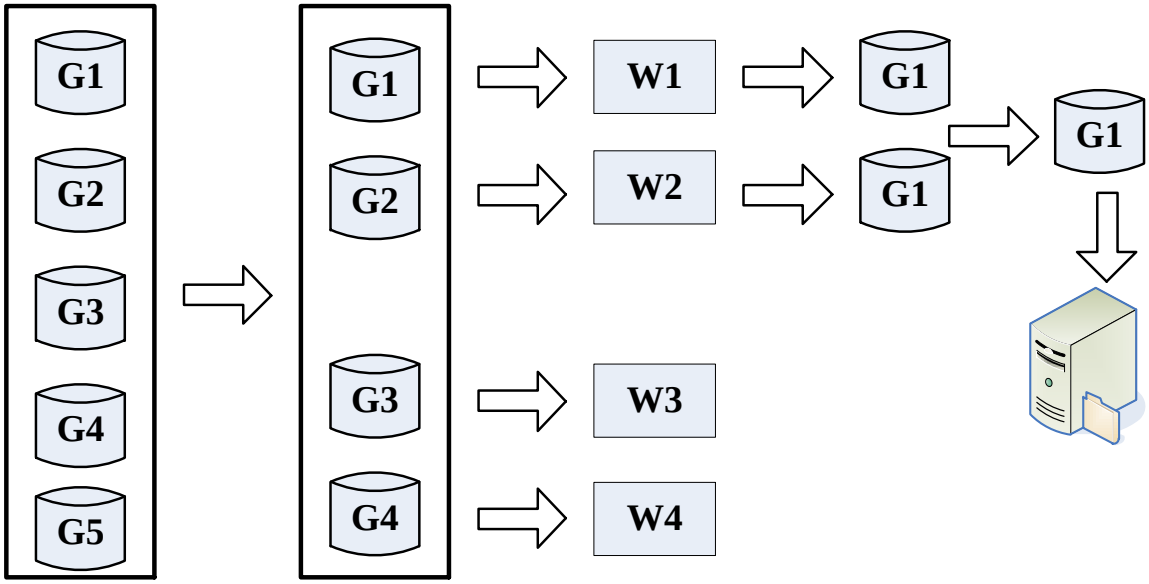

图 3-1 FastDFS 的 Storage Server 组的选择流程

FastDFS 的负载均衡由 Tracker Server 来控制，其 Storage Server 的选择流程有 5 步：

Step 1: 当 Tracker Server 接收到 Client 发出的请求以后首先会根据内存中 Storage Server 组的状态信息选出可用 Storage Server 的组。

Step 2: 在可用的 Storage Server 组中，通过选择算法计算出每个 Storage Server 组的权重值 W_n

Step 3: 将计算出来的所有权重从高到低排序，选择出权重值最高的两个 Storage Server 组作为候选使用的 Storage Server 组

Step 4:在权重值最高的两个 Storage Server 组中采用轮叫调度算法来选择响应 Client 的 Storage Server 组，例如：G1 和 G2 是权重值最高的两个 Storage Server 组，且 $W_1 > W_2$。首先会选择权重值最高的 G1 作为响应 Client 的 Storage Server 组，若 G1 由于某种原因不可用则选择权重值较低 G2 作为响应 Client 的 Storage Server 组，若都不可用则从第一步开始重新选择 Storage Server 组。

Step 5:确定了调用的 Storage Server 组后，该组中所有可以用来直接下载数据的 Storage Server 将采取轮叫调度算法来确定为 Client 提供服务 Storage Server，其地址和端口号会被 Tracker 返回给 Client。

3.1.2 FastDFS 现有 Storage Server 组选择算法参数的定义

（1）定义 1

定义 G 代表 FastDFS 的一个 Storage Server 组，*Gi* 表示第 *i* 个 Storage Server 组。S_i 为 *Gi* 的总磁盘空间，S_i 是第 *i* 个 Storage Server 组中一个 Storage Server 的磁盘空间和组内服务器数量的乘积（同一个 Storage Server 组中的服务器的磁盘空间一般相同），S_{max} 为所有 Storage Server 中总磁盘空间最大的一个。当前状态必须满足如下两个条件才会被 Tracker Server 计算其权重值 W_i，作为可用的 Storage Server 组：

① 组 *Gi* 中必须有 Storage Server 处于活跃可用状态。

② 组 *Gi* 中的 Storage Server 磁盘的剩余空间必须大于值 M，其中 M 为一个可以更改的预设值，在 FastDFS 系统设定中 M 值的大小默认为 4GB。

假设可用的 Storage Server 组数量为 number，只要组 *Gi* 满足定义 1 条件，number 的数量加 1。

（2）定义 2

定义 W_i 为组 *Gi* 的权重值，组 *Gi* 必须满足如下两个条件才会被 Tracker Server 计算其权重值 W_i，作为用来存储数据候选 Storage Server 组：

① 组 *Gi* 满足定义 1 中可用 Storage Server 组的条件。

② 组 *Gi* 的权重值必须大于 1，即 W_i＞1。

假设可以被直接用来存储数据的组 *Gi* 的数量为 availnumber，只要组 *Gi* 满足定义 2 的条件，availnumber 的数量就加 1。

W_i 计算方法的详细描述将会在第 3.1.3 节加以详细描述。

（3）定义 3

定义 requires 为采用轮叫调度算法来选择响应 Client 的 Storage Server 组的数目，requires 为一个可以更改的预设值，一般设定为 2，即会在权重值最高的 2 个 Storage Server 组中采用轮叫调度算法来选择响应 Client 的 Storage Server 组。若权重值最高的 Storage Server 组由于某种原因不可用则选择权重值较低的 Storage Server 组作为响应 Client 的 Storage Server 组，若都不可用则从第一步开始重新选择 Storage Server 组。

3.1.3 FastDFS 现有 Storage Server 组的权重值函数

通过对 FastDFS 现有负载均衡算法研究分析后可知， Storage Server 组的权重值函数中，其权重值 W_i 只和 S_i 有关，其权重值函数为公式 3-1：

$$f(s_i) = W_i^n = W_i^{n-1} + S_i / S_{\max} \tag{3-1}$$

在公式 3.1 中，$f(S_i)$ 是权重值函数, 由公式 3.1 可以看出 $f(S_i)$ 是一个迭代函数，函数从 *n*=1 开始迭代，初始值 W_i^0 为算法开始时系统给每一个组 *Gi* 赋予的一个随机值，其取值范围为[0,1),当 W_i^n 大于 1 的时候该公式停止迭代并将此时的值 W_i^n 作为与其它 Storage Server 组比较的权重值 W_i。

3.1.4 FastDFS 现有 Storage Server 组的选择算法描述

下面用伪代码来描述 Storage Server 的选择算法：

W_i^0 为算法开始时系统给每一个组随机 *Gi* 赋予的初始值，且 $W_i^0 \in [0,1)$；

number=0;//定义 1 中可用的组 *Gi* 数量

availnumber=0;//定义 2 中可以被直接用来存储数据的组 *Gi* 的数量

requires=2;//定义 3 中采用轮叫调度算法来选择响应 Client 的组 *Gi* 的数目

{

for(i=1,i≤N,i++);//N 系统中 Storage Server 组的总数目

{

if(Gi 满足定义 1 中的两个条件)

{ number++

for(each group 满足定义 1 中的两个条件)

{

for(n=1; W_i^n ≤1;n++);

{

$W_i^n = W_i^{n-1} + S_i / S_{\max}$;

}

if(W_i^n ≧1)

{

availnumber++

}

}

}

}

while(availnumber＜requires)

{

函数调转到开头执行;

}

if(availnumber≧requires)

{

选取W_i^n最大的 2 个 Storage Server 组；

首先询问W_i^n最大的组是否可用，若可用则选择该组作为上传 Storage Server 组，在该组中采取轮叫调度算法选择一台提供上传服务的 Storage Server；

若W_i^n最大的组不可用则询问剩下的一个组，若该组可用则在该组中采取轮叫调度算法选择一台提供上传服务的 Storage Server；

若剩下的一个组还是不可用，则函数调转到开头执行；

}

3.2 FastDFS 现有负载均衡算法存在的问题

经过 3.1 节对 FastDFS 现有的算法分析之后得知某个 Storage Server 组是否能被选中，其关键在于权重值 W_i 的大小。根据公式 3.1 我们可以发现的权重值 W_i 大小主要由 S_i / S_{max} 来决定。而 S_{max} 是一个固定值所以权重值 W_i 大小主要由该 Storage Server 组的磁盘容量 S_i 决定，所以可以看出 FastDFS 现有的负载均衡算法是一个静态的负载均衡算法，其物理意义为：一个 Storage Server 组的硬盘容量越大，其被选中的概率就越大。但是一个 Storage Server 的性能不仅由其硬盘容量决定，还受到例如网络带宽、CPU 性能，内存大小和频率等硬软件因素和 Storage Server 组自身的负载情况影响。一个 Storage Server 是否能被选中仅受硬盘容量影响明显是不合理的，一个硬盘容量最大的 Storage Server 组不一定就具有最好的性能，而且最后选择该组的某个 Storage Server 采用的是简单的轮询算法并未考虑该 Storage Server 的负载情况，所以现有的负载均衡算法存在很大的局限性。

3.3 FastDFS 负载均衡算法的改进算法

3.3.1 新算法 Storage Server 组的选择思想

新算法的 Storage Server 选择思想与原算法的大致相同，但是 Storage Server 组的权重值 W_n 的计算方法不同以及最后一步选择 Storage Server 组中某一台服务器用动态负载均衡算法最少连接数算法代替了原有的静态负载均衡算法轮叫调度算法，其过程可以用图 3-2 表示：

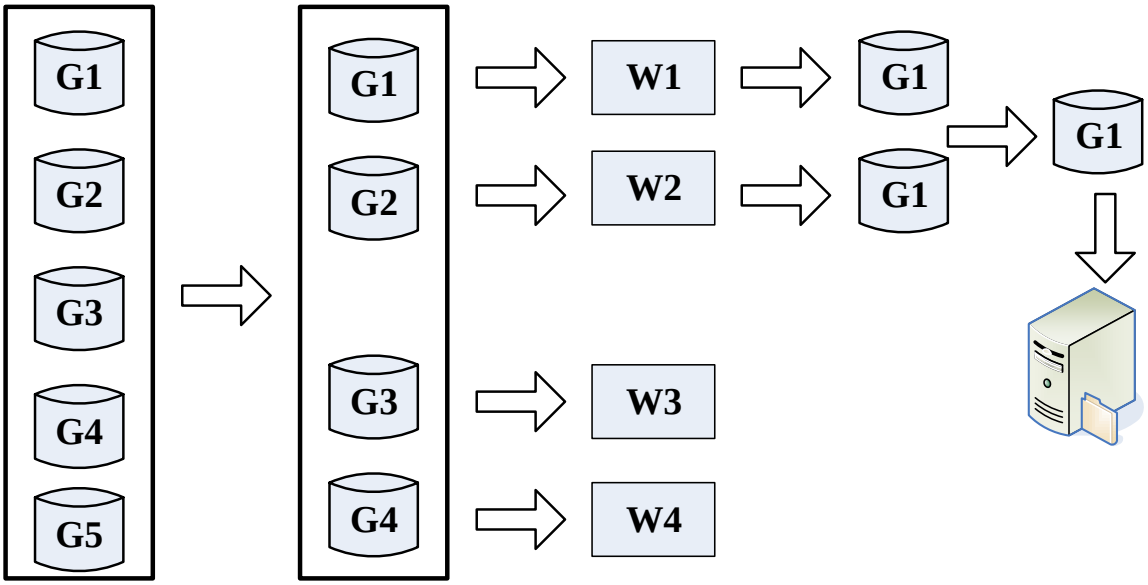

图 3-2 FastDFS 的 Storage Server 组的选择流程

FastDFS 的负载均衡由 Tracker 来控制，其 Storage Server 的选择流程有 5 步：

Step 1: 当 Tracker Server 接收到 Client 发出的请求以后首先会根据内存中 Storage Server 组的状态信息选出可用 Storage Server 的组。

Step 2: 在可用的 Storage Server 组中，通过选择算法计算出每个 Storage Server 组的权重值 W_n

Step 3: 将计算出来的所有权重从高到低排序，选择出权重值最高的两个 Storage Server 组作为候选使用的 Storage Server 组

Step 4:在权重值最高的两个 Storage Server 组中采用轮叫调度算法来选择响应 Client 的 Storage Server 组，例如：G1 和 G2 是权重值最高的两个 Storage Server 组，且 $W_1 > W_2$。首先会选择权重值最高的 G1 作为响应 Client 的 Storage Server 组，若 G1 由于某种原因不可用则选择权重值较低 G2 作为响应 Client 的 Storage Server 组，若都不可用则从第一步开始重新选择 Storage Server 组。

Step 5:确定了调用的 Storage Server 组后，该组中所有可以用来直接下载数据的 Storage Server 将最少连接数算法来确定为 Client 提供服务 Storage Server，其地址和端口号会被 Tracker 返回给 Client。

3.3.2 新算法参数的定义

在新算法中，有些参数的定义和原算法相同，参照 3.1.2 节，下面定义新算法的参数：

（1）定义 1

在原有算法中，服务器组权重值只考虑了该服务器组的总容量，我们现在动态的考虑服务器组现有的存储容量。定义 G 代表 FastDFS 的一个 Storage Server 组，*Gi* 表示第 *i* 个 Storage Server 组。C_i 为组 G_i 现有的存储容量，S_{max} 为所有 Storage Server 中总磁盘空间最大的一个。当前状态必须满足如下两个条件才会被 Tracker Server 计算其权重值 W_i，作为可用的 Storage Server 组：

① 组 *Gi* 中必须有 Storage Server 处于活跃可用状态。

② 组 *Gi* 中的 Storage Server 磁盘的剩余空间必须大于值 M，其中 M 为一个可以更改的预设值，在 FastDFS 系统设定中 M 值的大小默认为 4GB。

假设可用的 Storage Server 组数量为 number，只要组 *Gi* 满足定义 1 条件，number 的数量加 1。

（2）定义 2

在原有算法中并未考虑 Storage Server 除了磁盘容量大小以外的硬件对其性能的影响，因此我们对 Storage Server 进行黑盒测试以定量分析其性能。由于 Storage

Server 的主要功能是存储数据文件，所以对文件的读写速度可以反映一个 Storage Server 对于整个系统来说的硬件性能。由于 FastDFS 是在 Linux 操作系统下运行的分布式文件系统，所以我们使用可以在 Linux 操作系统下用来测试服务器 I/O 性能的测试软件 IOzone。

IOzone 是一个可以在不同操作系统下测试的系统文件操作性能的工具，使用该工具可以在多线程、多 CPU、并指定 CPU 缓存空间大小以及异步或者同步 I/O 读写模式的情况下测试文件系统的文件操作性能。该工具特别适合测试大规模分布式文件系统的 I/O 性能。我们使用该工具的测试选项包括：write,read,random-write,random-read,re-write,re-read。

我们通过 IOzone 测试实验用的 Storage Server 这个 6 个方面的性能，取这 6 个方面性能的算数平均值代表其性能值 P_i，得到数据如表 3-1 所示：

表 3-1 Storage Server 性能表

服 务 器	性能
Storage Server 1(CPU I7 4790K 虚拟 CPU 一个，硬盘 100G,内存 1G)	58598 KB/S
Storage Server 2(CPU I7 4790K 虚拟 CPU 一个,硬盘 150G,内存 512M)	55368 KB/S
Storage Server 3(CPU I7 4790K 虚拟 CPU 一个，硬盘 300G,内存 2G)	153790 KB/S

定义组 G_i 性能用 P_i 表示，系统中性能最好的 Storage Server 性能用 P_{max} 表示。

（2）定义 3

定义 W_i 为组 *Gi* 的权重值，组 *Gi* 必须满足如下两个条件才会被 Tracker Server 计算其权重值 W_i，作为用来存储数据候选 Storage Server 组：

① 组 *Gi* 满足定义 1 中可用 Storage Server 组的条件。

② 组 *Gi* 的权重值必须大于 1，即 $W_i > 1$。

假设可以被直接用来存储数据的组 *Gi* 的数量为 availnumber，只要组 *Gi* 满足定义 3 的条件，availnumber 的数量就加 1。

W_i 计算方法的详细描述将会在第 3.3.3 节加以详细描述。

（3）定义 4

定义 requires 为采用轮叫调度算法来选择响应 Client 的 Storage Server 组的数目，requires 为一个可以更改的预设值，一般设定为 2，即会在权重值最高的 2 个 Storage Server 组中采用轮叫调度算法来选择响应 Client 的 Storage Server 组。若权重值最高的 Storage Server 组由于某种原因不可用则选择权重值较低的 Storage Server 组作为响应 Client 的 Storage Server 组，若都不可用则从第一步开始重新选择 Storage Server 组。

3.3.3 新算法 Storage Server 组的权重值函数

新算法的 Storage Server 的权重值函数可以用公式 3-2 表示：

$$W_i^n = f(C_i, P_i) = W_i^{n-1} + (C_i / S_{\max}) \times (P_i / P_{\max}) \qquad (3\text{-}2)$$

现在我们同时考虑了 C_i 和 P_i 这两个性能指标，其中 C_i 是一个处在动态变化之中的值，所以新算法是一个动态负载均衡算法。和公式 3-1 相比加入了一个动态因子 C_i 和静态因子 P_i，不仅考虑的方面更全面而且将原有的静态负载均衡算法变成了动态负载均衡算法。

3.3.4 新 Storage Server 组选择算法的描述

下面用伪代码来描述新的 Storage Server 的选择算法：

W_i^0 为算法开始时系统给每一个组随机 *Gi* 赋予的初始值，且 $W_i^0 \in [0,1)$;

number=0;//定义 1 中可用的组 *Gi* 数量

availnumber=0;//定义 2 中可以被直接用来存储数据的组 *Gi* 的数量

requires=2;//定义 3 中采用轮叫调度算法来选择响应 Client 的组 *Gi* 的数目

{

for(i=1,i≤N,i++);//N 系统中 Storage Server 组的总数目

{

if(*Gi* 满足定义 1 中的两个条件)

{ number++

for(each group 满足定义 1 中的两个条件)

{

for(n=1; W_i^n ≤1;n++);

{

$W_i^n = f(C_i, P_i) = W_i^{n-1} + (C_i / S_{\max}) \times (P_i / P_{\max})$；

}

if(W_i^n ≧1)

{

availnumber++

}

}

}

}

while(availnumber＜requires)

{

函数调转到开头执行；

}

if(availnumber≧requires)

{

选取W_i^n 最大的 2 个 Storage Server 组；

首先询问W_i^n 最大的组是否可用，若可用则选择该组作为上传 Storage Server 组，在该组中采用最小连接数算法选择一台提供上传服务的 Storage Server；

若W_i^n 最大的组不可用则询问剩下的一个组，若该组可用则在该组中采用最小

连接数算法选择一台提供上传服务的 Storage Server；

若剩下的一个组还是不可用，则函数调转到开头执行；

}

3.4 改进算法的测试和性能评价

3.4.1 算法测试的环境和方法

测试方法为向实验系统发存储数据：将 200G 实验数据做 100 次存储，每次存储 2G。我们通过计算测试后 Storage Server 组空间利用率的方差值来评价算法的负载均衡性能, 用测试完成的平均时间来评价系统的 I/O 性能。方差是反应一组数据离散程度的统计量，测试后 Storage Serve 组空间利用率的方差值越小，说明数据在 Storage Server 组中存储的越均匀，负载均衡算法的性能越好。测试系统由 6 台虚拟机组成，其分组和 Storage Server 的初始情况如表 3-2 所示：

表 3-2 Storage Server 的初始情况

服务器组	服务器	总磁盘容量	已用空间	空间利用率
G1	Storage Server 1&2(CPU I7 4790K 虚拟 CPU 一个，硬盘 100G,内存 1G)	100G	50G	0.5
G2	Storage Server 3&4(CPU I7 4790K 虚拟 CPU 一个，硬盘 150G,内存 512M)	150G	20G	0.13
G3	Storage Server 5&6(CPU I7 4790K 虚拟 CPU 一个，硬盘 300G,内存 2G)	300G	0G	0

3.4.2 FastDFS 原有算法的测试结果

使用实验用数据对原有 FastDFS 算法的测试三次，得到实验后 Storage Server 的空间利用率如表 3-3 所示：

表 3-3 原有 FastDFS 算法实验结果

测试数据	Storage Server 1	Storage Server 2	Storage Server 3	Storage Server 4	Storage Server 5	Storage Server 6
第一组	0.80	0.81	0.46	0.47	0.32	0.31
第二组	0.82	0.83	0.45	0.45	0.31	0.30
第三组	0.81	0.82	0.47	0.46	0.30	0.29
平均值	0.81	0.82	0.46	0.46	0.31	0.30

通过上表计算可以知道 G1、G2、G3 的空间利用率分别为 0.82、0.46、0.31，三个 Storage Server 组空间利用率的方差为 0.0458。三次测试完成时间分别为 9823.24s、9538.05s、9487.37s，平均值为 9616.22s。

3.4.3 FastDFS 新算法的测试结果和性能评价

由于新的算法有服务器组性能值，现有存储容量，和现有连接数三个影响因子，所以我们采取控制变量法来对 FastDFS 新算法进行测试：

（1）只考虑现有存储容量的情况：

我们将权重值函数改为公式 3-3:

$$W_i^n = f(C_i) = W_i^{n-1} + C_i / S_{\max} \qquad (3\text{-}3)$$

使用这种权重值函数的负载均衡算法称为算法 A, 使用实验用数据测试算法 A 三次得到实验后 Storage Server 的空间利用率如表 3-4 所示：

表 3-4 算法 A 的实验结果

测试数据	Storage Server 1	Storage Server 2	Storage Server 3	Storage Server4	Storage Server 5	Storage Server 6
第一组	0.50	0.50	0.45	0.44	0.44	0.42
第二组	0.50	0.50	0.44	0.45	0.43	0.42
第三组	0.50	0.50	0.43	0.45	0.42	0.40
平均值	0.50	0.50	0.44	0.45	0.43	0.41

通过上表计算可以知道 G1、G2、G3 的空间利用率分别为 0.50、0.45、0.42，三个 Storage Server 组空间利用率的方差为 0.01089。三次测试完成时间分别为 8698.23s、8568.45s、8369.05s，平均值为 8545.24s。通过和原有算法空间利用率的方差值 0.0458 和测试完成平均时间 9616.22s 比较发现：算法 A 相比原系统明显提高了测试系统的负载均衡性能和系统的 I/O 性能。说明加入现有存储容量这个因子能提高测试系统的负载均衡性能和 I/O 性能。

（2）只考虑服务器组性能值的情况：

我们将权重值函数改为公式 3-4:

$$W_i^n = f(P_i) = W_i^{n-1} + P_i / P_{\max} \qquad (3\text{-}4)$$

使用这种权重值函数的负载均衡算法称为算法 B, 使用实验用数据测试算法 B 三

次得到实验后 Storage Server 的空间利用率如表 3-5 所示：

表 3-5 算法 B 的实验结果

测试数据	Storage Server 1	Storage Server 2	Storage Server 3	Storage Server 4	Storage Server 5	Storage Server6
第一组	0.75	0.74	0.29	0.28	0.43	0.44
第二组	0.73	0.74	0.28	0.27	0.42	0.43
第三组	0.74	0.75	0.28	0.28	0.44	0.42
平均值	0.74	0.74	0.28	0.28	0.43	0.43

通过上表计算可以知道 G1、G2、G3 的空间利用率分别为 0.74、0.28、0.43，三个 Storage Server 组空间利用率的方差为 0.03667。三次测试完成时间分别为 8838.34s、9264.05s、9145.27s，平均值为 9082.55s。通过和原有算法空间利用率的方差值 0.0458 和测试完成平均时间 9616.22s 比较发现：算法 B 相比原系统提高了测试系统的负载均衡性能，但是效果并不明显，其原因是算法 B 和原算法类似是静态的负载均衡算法且各组 $P_i / P_{\max}$ 的值和 $S_i / S_{\max}$ 的值相近；能够明显提高系统的 I/O 性能。实验结果说明加入服务器性能值这个因子能明显提高测试系统的 I/O 性能，对测试系统的负载均衡性能提升不明显。

（3）只考虑服务器当前连接数的情况：

我们使用原有 FastDFS 的权重值公式，但在 Storage Server 组内选择服务器时采用最少连接数算法，把这种情况称为算法 C。使用实验用数据测试算法 C 三次得到

实验后 Storage Server 的空间利用率如表 3-6 所示：

表 3-6 算法 C 的实验结果

测试数据	Storage Server 1	Storage Server 2	Storage Server 3	Storage Server 4	Storage Server 5	Storage Server6
第一组	0.82	0.80	0.45	0.47	0.31	0.32
第二组	0.81	0.82	0.46	0.45	0.32	0.29
第三组	0.83	0.81	0.45	0.46	0.30	0.30
平均值	0.82	0.81	0.45	0.46	0.30	0.30

通过上表计算可以知道 G1、G2、G3 的空间利用率分别为 0.82、0.46、0.30，三个 Storage Server 组空间利用率的方差为 0.04729。三次测试完成时间分别为 9754.21s、9987.57s、9879.05s，平均值为 9873.61s。通过和原有算法空间利用率的方差值 0.0458 和测试完成平均时间 9616.22s 比较发现：算法 C 相比原算法对实验系统的负载均衡性能几乎没有任何改变，实验系统的 I/O 性能反而略有降低。说明加入当前连接数这个因子并不能明显改变系统的负载均衡性能，实验系统的 I/O 性能反而略有降低。实验系统的 I/O 性能略有降低的原因是每次 Storage Server 组内选择服务器时每个服务器的连接数都被设置为随机数，所以可能存在组内的某个服务器被设置的连接数一直比较小所以被长期调用，降低了实验系统的 I/O 性能。

（4）同时考虑服务器组现有存储容量和性能值的情况

通过算法 A、算法 B、算法 C 的测试结果可知：服务器现有存储容量这个因子

能明显提升实验系统的负载均衡性能；服务器组性能值能提高测试系统的负载均衡性能，但是效果并不明显；服务器当前连接数这个因子几乎不能改变系统的负载均衡性能。服务器现有存储容量和服务器组性能值这两个因子能明显提高实验系统的I/O 性能；服务器当前连接数这个因子会使实验系统的 I/O 性能略有降低。综合上面实验结果，只需要验证同时考虑服务器组现有存储容量和性能值的情况。

我们将权重值函数改为公式 3-2:

$$W_i^n = f(C_i, P_i) = W_i^{n-1} + (C_i / S_{\max}) \times (P_i / P_{\max}) \qquad (3\text{-}2)$$

但在 Storage Server 组内选择服务器时仍采用原来的轮叫调度算法，把这种情况称为算法 D。使用实验用数据测试算法 D 三次得到实验后 Storage Server 的空间利用率如表 3-7 所示：

表 3-7 算法 D 的实验结果

测试数据	Storage Server 1	Storage Server 2	Storage Server 3	Storage Server 4	Storage Server 5	Storage Server6
第一组	0.54	0.53	0.31	0.30	0.46	0.47
第二组	0.58	0.59	0.30	0.31	0.46	0.46
第三组	0.55	0.54	0.32	0.28	0.48	0.47
平均值	0.55	0.55	0.31	0.30	0.47	0.47

通过上表计算可以知道 G1、G2、G3 的空间利用率分别为 0.55、0.31、0.47，三个 Storage Server 组空间利用率的方差为 0.09956。三次测试完成时间分别为 8788.03s、8458.25s、8219.45s，平均值为 8488.58s。通过和算法 A 和算法 B 空间利

用率的方差值和测试完成平均时间比较发现，算法 D 相比算法 A 和算法 B 提高了系统的负载均衡性能和系统的 I/O 性能，说明同时加入服务器组现有存储容量和性能值这两个因子能明显提高测试系统的负载均衡性能和 I/O 性能。

3.4.4 FastDFS 新算法的优化

经过上面是实验过程可以得出结论：服务器现有存储容量这个因子能明显提升实验系统的负载均衡性能；服务器组性能值能提高测试系统的负载均衡性能，但是效果并不明显；服务器当前连接数这个因子几乎不能改变系统的负载均衡性能。服务器现有存储容量和服务器组性能值这两个因子能明显提高实验系统的 I/O 性能；服务器当前连接数这个因子会使实验系统的 I/O 性能略有降低。同时引入性能 P_i 和存储容量 C_i 后这两个因子以后新算法的负载均衡效果和 I/O 性能要明显好于原算法，但是使用最小连接数算法选择组内提供上传服务的 Storage Server 这一措施几乎不能改变系统的负载均衡性能而且使 I/O 性能略有降低。经过查找文档得知，FastDFS 分布式文件系统的磁盘 I/O 由专门的线程处理，系统负载和连接数不存在线性关系，系统负载基本上不受连接数影响。

综合上面的实验结果，对新算法的优化采用了最低缺失算法来代替最小连接数算法。虽然连接数和系统负载不存在线性关系，系统负载基本上不受连接数影响，但是统计 Storage Server 组内服务器的历史连接数，调用历史连接数最小的服务器来提供服务，从理论上来说能够使 Storage Server 组内服务器的调用平均化，有利于提高 FastDFS 分布式文件系统 I/O 性能。把对新算法优化后的算法称为算法 E，使用实验用数据测试算法 E 三次得到测试完成时间分别为 8137.63s、8079.45s、8219.45s，平均值为 8145.32s。通过和算法 D 的平均完成时间 8488.58s 比较发现使用最低缺失算法能够提高测试系统的 I/O 性能。

3.5 本章小结

本章的主要内容是首先研究了 FastDFS 分布式文件系统现有的负载均衡算法并

分析了现有的负载均衡算法存在的问题和不足之处，针对这些问题和不足之处我们引入了 Storage Server 的性能 P_i 和组 G_i 现有的存储容量 C_i 这两个新因子，以及使用最小连接数算法来选择组内提供上传服务的 Storage Server。测试实验表明同时引入性能 P_i 和存储容量 C_i 后这两个因子以后新算法的负载均衡效果和 I/O 性能要明显好于原算法，但是使用最小连接数算法选择组内提供上传服务的 Storage Server 这一措施几乎不能改变系统的负载均衡性能，反而使系统的 I/O 性能有所降低。经过查找文档得知，FastDFS 分布式文件系统的磁盘 IO 由专门的线程处理，系统负载和连接数不存在线性关系，系统负载基本上不受连接数影响。故最后对新算法进行优化时采用了最低缺失算法来代替最小连接数算法，测试实验表明，最低缺失算法能够提高测试系统的 I/O 性能。

4 FastDFS 在水土保持网站系统的应用

4.1 水土保持网站的需求分析

以某省的水土保持网站为例，水土保持网站可以被归为机构信息类网站。水土保持网站的信息包括基础信息、土壤侵蚀信息、综合治理信息、预防监督信息和综合信息[42]。水土保持工作的主要管理部门分为两级：水利厅和监测站点行政管理机构，监测站点又可以细分为市级监测站点和县级监测站点[43]。水土保持网站结构分为两大部分：外门户和内网门户[44]。外网门户即网站首页，内网门户分为以下几个部分：生产建设项目方案上报管理系统、水土保持数据录入和管理系统、水土保持专题信息发布系统、水土保持公报数据上报系统、水土保持年报数据上报系统、水土保持监测信息系统、生产建设项目监测数据上报管理系统[45] [46][47][48]。

水土保持网站系统分布如图 4-1 所示：

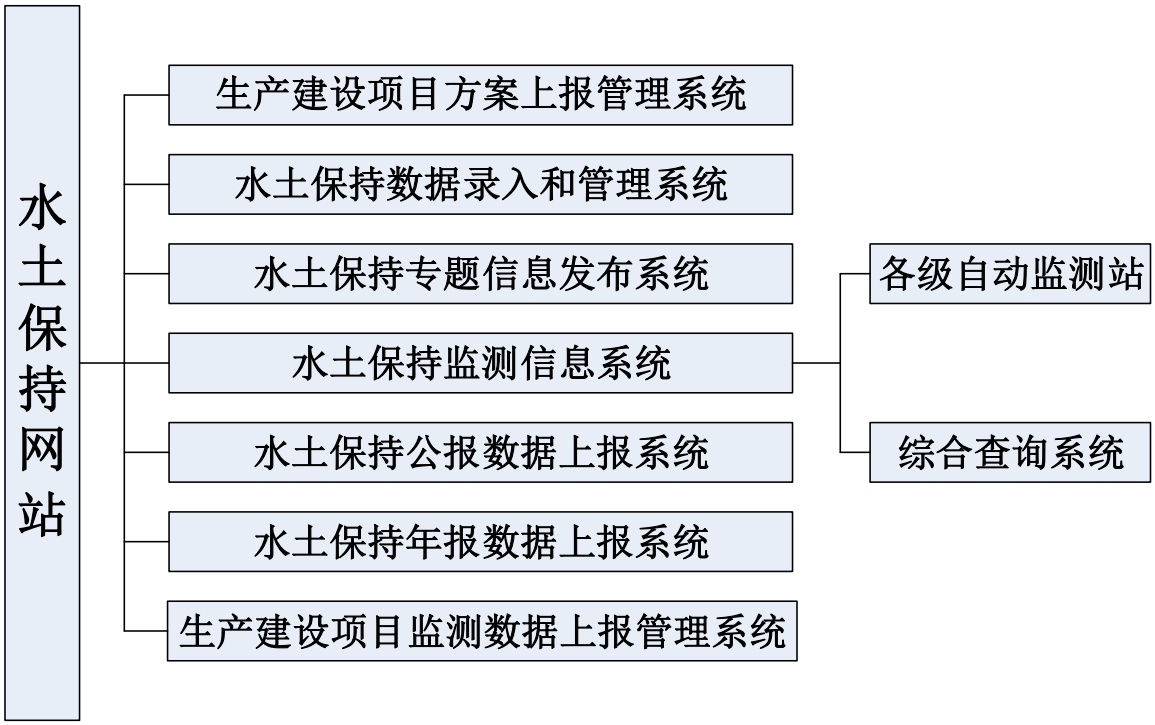

图 4-1 水土保持网站系统分布图

其中水土保持监测信息系统是一个新添加的对外公开的系统，其主要功能分为

两部分：一是用于各级水土保持自动监测站自动向其上传水土保持监测数据；二是专业人士和公众通过综合查询系统对水土保持监测信息进行查询。监测信息主要包括水位、气象和 TDR 的数据及每日汇总。

针对某省新增了水土保持监测信息系统的实际情况，水土保持网站有了如下的新需求：

（1）各个水土保持自动监测站新配备了视频监测系统定时上传当地水土保持自动监测站的监测图像，使该网站的数据量大大增加。

（2）由于该系统对外开放，特别是允许普通民众通过综合查询系统对水土保持监测信息进行查询，使网站的访问压力大大增加，使网站出现了应对大并发访问的需求。

（3）由于访问量和数据量的大大增加，以前传统的网站架构不足以满足当前需要，系统上传和下载速度缓慢，需要将应用和数据分离的文件存储系统来应对这个问题。

4.2 选用 FastDFS 分布式文件系统的原因

面对某省的水土保持监测网站出现的新需求，决定在网站后端采用分布式文件系统来满足这些需求。根据实际情况，新的分布式文件系统必须满足下面的要求：

（1）新的分布式文件系统必须可靠性高，能够满足网站现在存储和访问需求。

（2）新的分布式文件系统适合网站提供基于中小文件的服务，特别是存储和访问图像数据，以应对新产生的存储需求。

（3）新的分布式文件系统必须易扩展，能够应对未来逐渐增加的存储和访问需求。

（4）面对有限的经费，新的分布式文件系统必须简约实用并具有自动容错和负载均衡能力，建设和维护成本低。

和现有通用的分布式文件系统相比，FastDFS 分布式文件系统的独特之处主要有如下三个方面：

（1）轻量级

从架构上来说，FastDFS 抛弃了其它分布式文件系统一般具有的用来保存数据索引的 Name Node，只有 Tracker Server 和 Storage Server 。由于不需要对文件进行分块存储所以不需要记录文件的索引信息，Tracker Server 只记录 Storage Server 的分组信息和 Storage Server 的状态信息，系统资源占用非常少。Client 访问 Tracker Server 时，Tracker Server 只需要扫描其内存中 Storage Server 分组和状态信息，根据当前状况给出应答，由于不需要读取文件的索引信息直接根据文件 ID 定位到 Storage Server 上，所以 Tracker Server 的非常轻量化，不存在成为系统瓶颈的可能。

对于绝大多数网站来说，用户与之交互的文件都比较小，绝大部分数据在 500M 以下，所以一味的追求文件分块存储是不明智的，既没有多大实际用处而且增加了系统的复杂度。FastDFS 舍弃了文件分块存储功能，省去了系统查询文件索引的中间环节，与其他分布式文件系统相比显得更加简洁高效。需要注意的是，由于 FastDFS 不支持文件分块存储，所以其不适合应用在需要分布式计算功能的场景。对于水土保持监测网站这样的应用场景，FastDFS 所提供的功能基本上都能满足其需要。

从整个系统的代码量来说，由于简洁的系统架构，最新版本的 FastDFS 包括 C 客户端 API、FastDHT 客户端 API 和 PHP extension 等，代码行数不足 5 万行。比较小的代码量也体现了 FastDFS 轻量级的特点。

（2）分组方式

FastDFS 采用是 Storage Server 分组的存储方式。在同一个 Storage Server 组中的服务器之间也是对等的，每个 Storage Server 上保存的文件完全相同，互为备份关系，可以在同组内任意一台 Storage Server 上对文件进行操作，所以同一个 Storage Server 组中的服务器的磁盘容量一般相同。根据木桶效应，一个 Storage Server 组的容量为该组 Storage Server 中磁盘容量最小的一个，所以同一个组的 Storage Server 的磁盘容量最好相同，其他软硬件配置也最好一致。

这种分组方式的主要优点是灵活可控，系统扩展能力强。当某个 Storage Server 组的负载压力比较大的时候，可以纵向在该 Storage Server 组添加新的服务器来提升其承受负载的能力；当 FastDFS 整个系统容量不足的时候，可以横向增加一台或者

多台服务器将它们配置成一个新的 Storage Server 组来增加系统容量。

（3）对等结构

由上文的可知在 FastDFS 架构之中，对于同一组内的 Storage Server 或者 Tracker Server，它们之间的相互关系也是对等的。对于 Tracker Server 和同组的 Storage Server 之间都不存在 Master-Slave 结构，也就不存在 Master 失效的问题。在传统 Master-Slave 结构的分布式文件系统之中，若 Master 出现故障，则必须将 Slave 的地位提升到 Master，实现起来系统的逻辑结构会比较复杂。对于 FastDFS 这样非 Master-Slave 结构的系统来说，每个节点都是 Master，系统不需要实现 Master 到 Slave 提升逻辑，系统不存在单点故障的问题。

和现有通用的分布式文件系统相比，FastDFS 分布式文件系统的优势如下：

（1）FastDFS 使用简便，可以直接使用，不需要二次开发。

（2）FastDFS 增加 Storage 和 Tracker 都比较简便：增加 Storage Server 组内服务器时配置文件不需要做任何的修改，系统将自动把该组服务器中的文件复制到该服务器，；增加 Tracker Server 只需要在 Client 和 Storage Server 配置文件中增加一行 Tracker Server 设置就可以实现。

（3）FastDFS 有非常详细的设计使用文档和运行日志记录。

（4）FastDFS 在性能方面比其它分布式文件系统有明显的提高，具体体现在如下几个方面：

①FastDFS 使用 C 语言编写相比其它分布式文件系统的编写语言（例如 MogileFS 使用 Perl 语言编写）其开发语言更为低层和高效，而且 FastDFS 整个代码量不到两万行，运行效率高。

②FastDFS 没有 Tracker’s database，将 Tracker’s database 的功能合并到 Tracker 中，文件同步直接点对点不经过 Tracker 中转。减少了系统流程中的中间环节，整体性能更佳。

③FastDFS 使用了 sendfile 方式传输文件，磁盘文件不经过内存被直接发送到网卡内存缓冲区，系统资源占用的少，文件传输的效率更高。

正因为 FastDFS 具有以上的特点和优势，所以 FastDFS 可以说是一个把简约实

用做到极致的分布式文件系统，资源需求量非常小，建设成本非常低，中小网站完全有能力使用。FastDFS 非常适合基于中小文件服务的站点，特别是以文件大小在4KB 到 500MB 之间为对象的在线服务，特别适合储存图像和视频文件。对于像水土保持网这样的中小站点，面对在监测过程中新出现的图像以及越来越大的访问压力，综合有限网站建设和后期维护经费，所以我们选择使用 FastDFS 这样经济适用的分布式文件系统来应对新的需求。

4.3 水土保持网站系统的架构

4.3.1 系统总体架构

以某省的水土保持网站系统为例，其采用的是典型的 3 层架构设计，整个系统分为 3 层：客户终端展现层、Web 应用层以及数据库和文件服务层，其架构如图 4-2 所示：

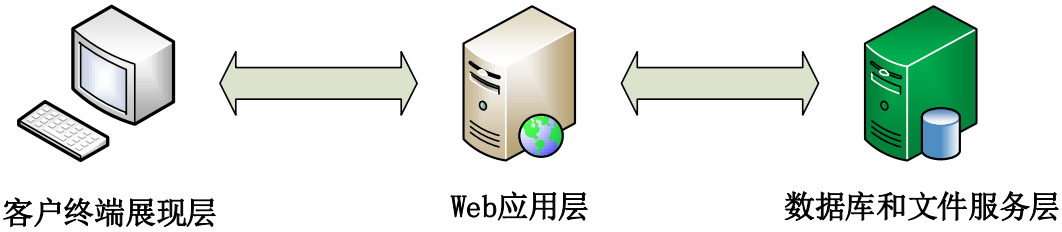

图 4-2 系统架构图

其中 Web 应用服务器安装 Apache 2.4.2 和 Tomcat 7.0；数据库和文件服务层使用 Oracle 12c 和 FastDFS 5.05。所有服务器使用 Ubuntu 14.04 操作系统；开发机环境为 Windows 7 SP1；Java 执行环境为 Jdk 1.6+和 Jre 1.6+。

4.3.2 系统的硬件配置

以某省的水土保持网站系统为例，其硬件由 Web 应用服务器、数据库服务器、分布式文件服务器、防火墙等设备组成，系统的硬件结构如图 4-3 所示：

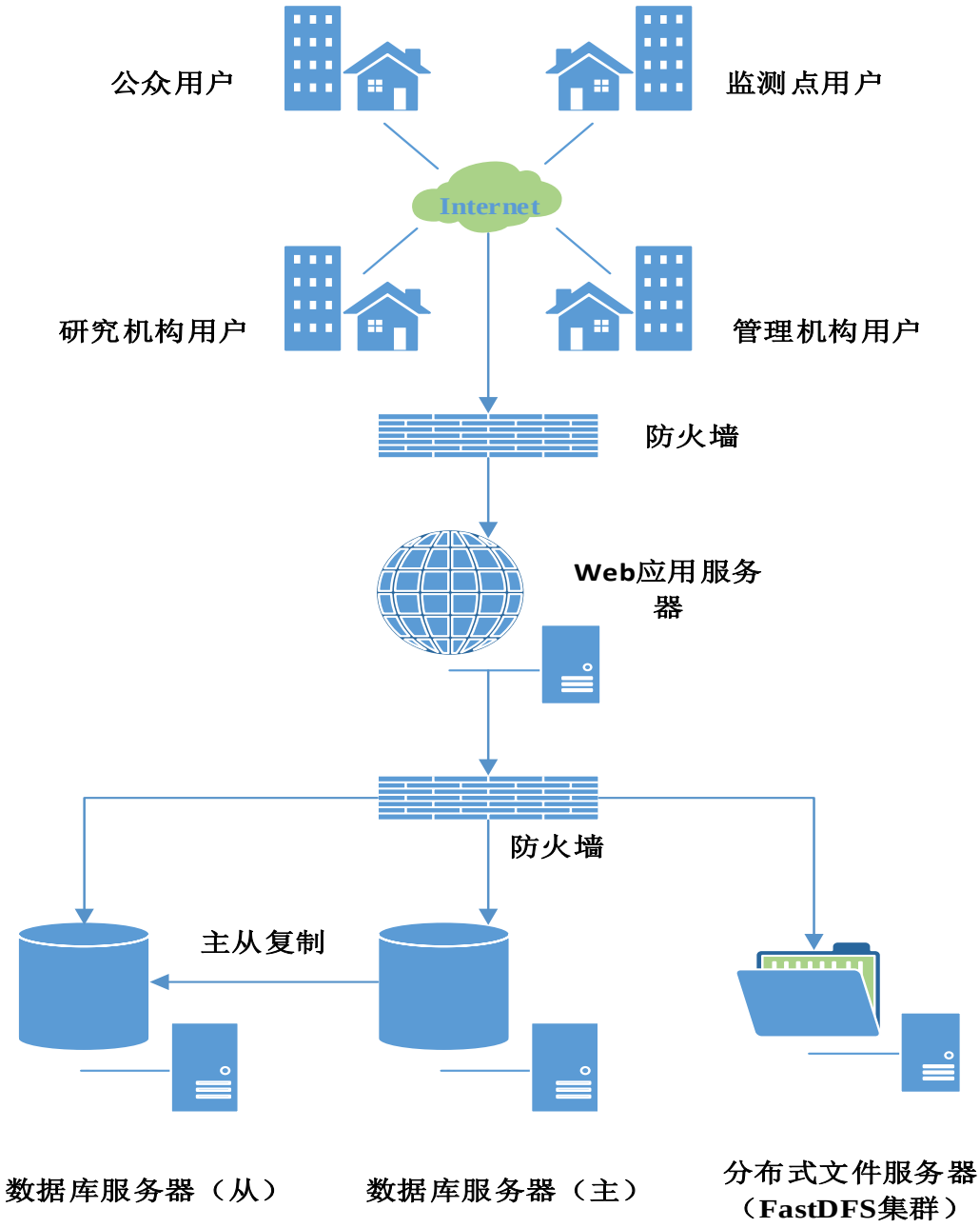

图 4-3 系统硬件结构图

FastDFS 集群的硬件结构如图 4-4 所示：

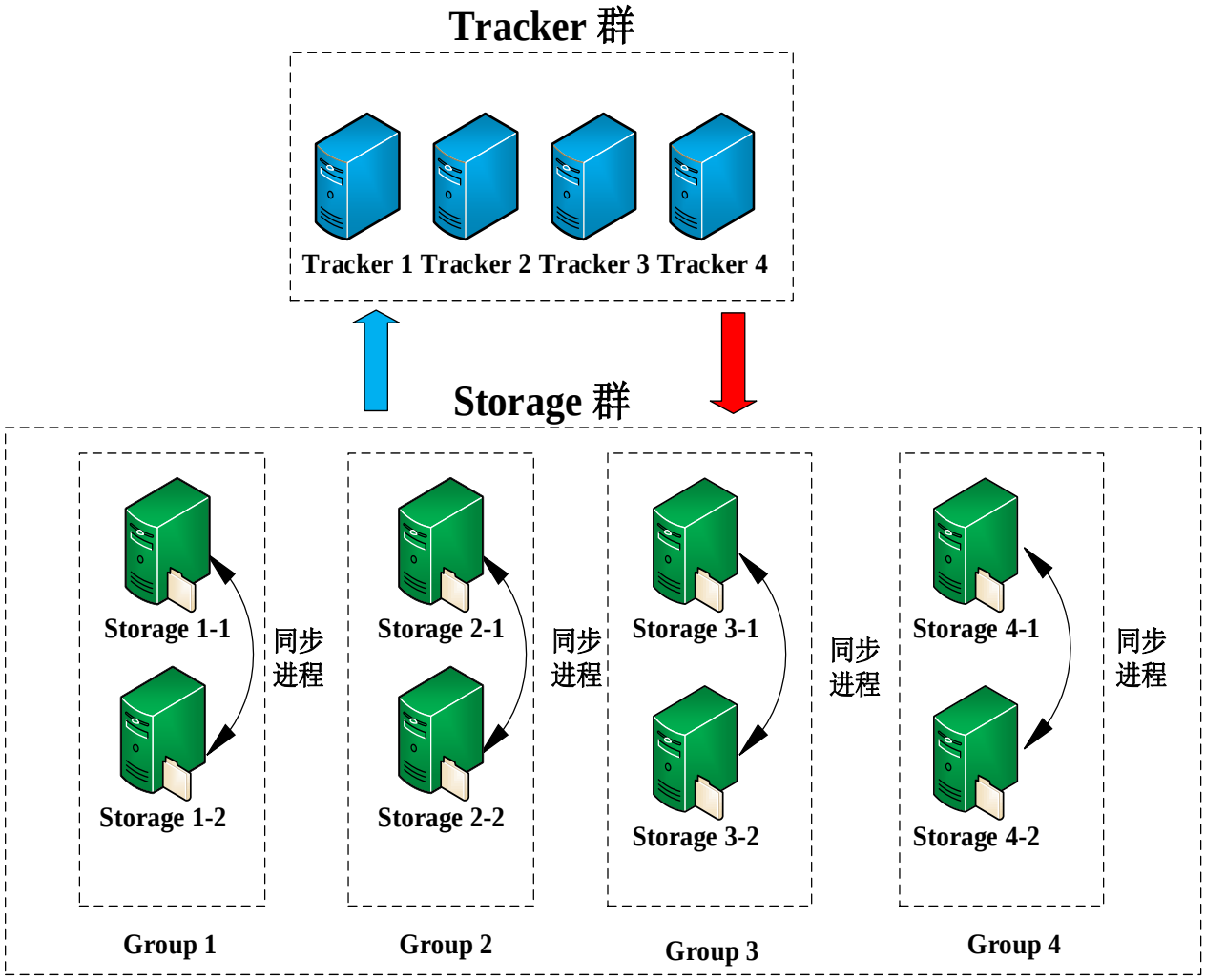

图 4-4　FastDFS 集群硬件结构图

4.3.3　系统的软件体系

以某省的水土保持网站系统为例，其软件体系采用的是 J2EE 三层架构，其中使用了 Hibernate，Spring 以及 Struts 这三个开源框架。其中表现层的 Struts 是系统的基础架构，为应用程序提供框架支持；Hibernate 是持久层的低层结构，实现 Java 对象模型和数据库之间的连接，为业务层提供数据库访问功能；中间的逻辑层使用 Spring 框架来实现业务逻辑，是连接 Hibernate 和 Struts 之间的桥梁，通过 Spring 框架可以自动生成 DAO 组件,通过 DAO 组件对数据库进行操作。Spring 和 Hibernate 一起组

成了整个系统的底层结构，实现了数据的传输和存储。

用户采用浏览器访问，Web 服务端安装 Tomcat 作为服务器用来部署和运行水土保持系统软件，水土保持软件主要通过 Struts 和 Hibernate 框架实现，中心数据库使用针对图像监测数据文件存储的 FastDFS 分布式文件系统。

4.4 FastDFS 在水土保持网站模拟实验系统中的测试

4.4.1 FastDFS 的安装

(1)首先在网址 http://code.google.com/p/fastdfs/downloads/list 下载所需安装文件，版本号为 V5.05 稳定版本。若系统没有安装 libfastcommon，则必须首先安装 libfastcommon 且满足 liberfastcommon 的安装路径为/usr 。

(2) 在 FastDFS_v5.05.tar.gz 所在文件夹下执行 shell 代码：sudo tar vxzf FastDFS_v5.05.tar.gz /home/soar/FastDFS，其中/home/soar/FastDFS 是解压目录。

(3)进入解压目录/home/soar/FastDFS 执行 shell 代码：sudo./make.sh。

(4)在解压目录/home/soar/FastDFS 下执行 shell 代码：sudo./make.sh install

(5)完成以上步骤后，在命令行信息最后看到如下的 shell 代码，表明完成安装：

```
#ln -fs /usr/local/lib/libfastcommon.so.1/usr/local/lib/libfastcommon.so
#ln -fs /usr/local/lib/libfdfsclient.so.1/usr/local/lib/libfdfsclient.so
sh ./fdfs_link_library.sh
```

4.4.2 FastDFS 的配置

(1) 配置及启动 Tracker Server

A. 修改配置文件/home/soar/FastDFS/conf/tracker.conf，设定 HTTP 端口、Tracker Server 对 Storage Server 供服务的端口和最大连接数等配置信息。

B. 将 http.conf 文件拷贝到/etc/fdfs 目录下，执行 shell 代码:

```
sudo cp /home/soar/FastDFS/conf/http.conf /etc/fdfs/
```

C. 进入/usr/local/bin/目录，启动 Ttracker 服务器，执行 shell 代码:

sudo fdfs_trackerd/home/soar/FastDFS /conf/tracker.conf

(2) 配置及启动 Storage Server

A.修改配置文件/home/soar/FastDFS/conf/storage.conf：可以设定所属组、文件的存储位置和向 Tracker Server 发送心跳检测时间间隔等信息。

B.进入/usr/local/bin/目录, 执行如下 shell 代码，来启动 Storage 服务器：

sudo fdfs_storaged /home/soar/FastDFS /conf/storage.conf

4.4.3 新算法在模拟实验系统中的部署和测试

将优化后的整个系统的源码使用 ant 打包，生成 Filemanage.war。然后放在 Tomcat 下的 Webapps 中再启动 Tomcat，整个 FastDFS 分布式文件系统就部署完成了。FastDFS 分布式文件系统拥有 8 台服务器，一共有 4 种配置，每种配置两台：第一种内存大小为 1G，存储空间大小为 100G；第二种内存大小 512M，存储空间大小为 200G；第三组内存 2G，存储空间大小为 400G；第四种内存大小为 2G，存储空间大小为 600G。

对新算法在水土保持网站的模拟实验系统上进行了并发下载实验，实验内容为使用每张大小为 100KB 总大小为 50G 的图片文件进行三次不同并发用户数的文件下载实验，三次实验并发数分别为 100、500、1000。实验分为 3 个组，每个组的系统配置如表 4-1 所示：

表 4-1 实验组的系统配置

系统配置	第一组	第二组	第三组	第四组
服务器数目	2	4	6	8
存储节点数目	2	4	6	8
跟踪节点数目	1	2	3	4

和原算法的对比实验结果如表 4-2 所示：

表 4-2 并发下载实验结果

系统	并发数	第一组	第二组	第三组	第四组
原算法	100	1.0 秒	0.9 秒	0.9 秒	0.8 秒
	500	2.2 秒	2.1 秒	2.0 秒	1.8 秒
	1000	4.4 秒	4.3 秒	4.1 秒	4.0 秒
新算法	100	0.8 秒	0.7 秒	0.5 秒	0.4 秒
	500	1.7 秒	1.6 秒	1.4 秒	1.1 秒
	1000	3.6 秒	3.3 秒	2.9 秒	2.5 秒

通过分析和对比原算法和新算法实验数据可以得出如下结论：新算法提升了系统的 I/O 性能，特别当存储服务器和跟踪服务器数目越多时，I/O 性能的提高越明显。当服务器数目较少时，新算法因为要收集更多服务器的信息，系统性能反而没有原系统好。

经过一段时间的测试运行数据表明系统在访问量较大出现高并发的时刻负载均衡性能和 I/O 性能得到了明显的提高，文件在各个服务器存储的更为均匀。在访问量较低的时刻对负载均衡性能和 I/O 性能优势并不明显，甚至新算法的性能要比原算法差，这是在访问量较低的情况下使用新算法反而系统要收集更多服务器的信息，新算法对性能的提升效果不足以弥补系统要收集更多服务器的信息对系统性能的损失。所以改进后的算法在高并发的情景下才能够较好的发挥其负载均衡效果。当访问量较小时，原有的算法反而运行效率更高。

4.5 本章小结

本章围绕 FastDFS 在水土保持网站系统的应用展开。首先介绍了水土保持网站的系统架构，并针对网站新添加的水土保持监测信息系统分析了网站新的需求。然后针对这些新的系统使用了 FastDFS 作为存储图像和视频监测数据的分布式文件系统。最后在水土保持网站上部署配置了 FastDFS 并使用了新算法在水土保持网站的模拟实验系统中进行了测试，实验结果表明：系统在访问量较大出现高并发的时刻负载均衡性能和 I/O 性能得到了明显的提高，文件在各个服务器存储的更为均匀。在访问量较低的时刻对负载均衡性能和 I/O 性能优势并不明显，甚至比原算法的性能差。

5 总结和展望

5.1 总结

本文主要研究了 FastDFS 的负载均衡算法，并对算法进行改进并应用在水土保持网站上。算法改进后影响因子由不变的 Storage Server 组总容量变为 Storage server 组的性能和动态变化的 Storage Server 组的现有容量；在 Storage Server 组内选择服务器时，由简单的轮询算法变成最低缺失算法。测试实验结果表明，算法改进后系统性能得到了提高，新算法的负载均衡效果要明显好与原算法，能较好地改善系统的负载不均衡度。将新算法应用到水土保持网站之后，改进了系统的负载均衡和 I/O 性能，提高了网站的网络服务质量，达到了预期的效果。需要指出的是，改进后的算法在高并发的情景下才能够较好的发挥其负载均衡效果。当访问量较小时，原有的算法反而运行效率更高。

随着我们国家社会主义现代化事业不断向前迈进，国家对水土保持监测的需求越来越大，各种相关的监测数据的信息量呈现快速增长的趋势，水土保持网络信息化的程度也随之越来越高。水土保持信息化对我国水土状况的研究，制定相应的水土保持对策具有重大的意义。

5.2 展望

对于算法的改进，注意到当访问量较小时，原有的算法反而运行效率高于改进后的算法，所以对于 FastDFS 的负载均衡算法改进研究下一步可以向自适应算法方向拓展。例如使用基于代理的自适应负载均衡（Agent Based Adaptive Balancing）算法。Tracker Server 启动一个包含自适应的逻辑线程来定时监测 Storage Server 的实时的负载状态，每台 Storage Server 定时向 Tracker Server 上传一个表明其负载状况的文件，这个文件用自然数来表示其当前负载状况的：0 表示空载，100 表示满载，大于 100 表示其过载。根据服务器的整体负载状况，Tracker Server 有两种负载均衡的策略可

以选择：当负载小于 50 时，表明系统访问量不大，Tracker Server 使用系统原有的负载均衡算法；当负载大于 50 时，表明系统访问量较大处于高并发状态，Tracker Server 使用本文中改进后的负载均衡算法。这样使系统能够自适应不同的访问情况，无论是低访问量还是高并发状态，都能使系统处于性能最佳的状态。

对于整个水土保持网站系统来说，可以增加缓存服务器和消息队列服务器来以及采用带负载均衡服务器的应用服务器集群以面对未来越来越大的访问压力。

随着信息技术蓬勃发展特别是分布式文件系统和负载均衡技术的深入研究，我相信我国的水土保持网络信息化事业能够不断的向前迈进！

致 谢

时光荏苒，韶华易逝。三年的研究生生涯即将结束，在这充实而有意义的三年里我在专业水平、个人体魄、生活习惯等方面都取得了很大的提高，在此我要真诚的感谢自己不懈的努力和实验室老师同学们在工作生活中对我无微不至的帮助，在华中科技大学求学的这宝贵的三年将会成为我这一生中最美好的回忆。

在论文即将完成之际，首先要感谢我的导师曾致远教授。曾老师学识渊博，治学严谨，从我进入实验室以来，曾老师就在各方面对我严格要求，在学习研究过程中给了我详细的指导。论文从开题到定稿，曾老师多次在百忙之中抽出宝贵的时间详细审阅并提出了修改意见，在这上面倾注了大量的心血，对论文的顺利完成给予了巨大的帮助。与此同时，曾老师平易近人，积极向上的生活态度也深深的影响了我。此外，我还要在此感谢付必涛，黄正军等老师在这三年里对我的教育和帮助，使我顺利地完成了研究生课题和论文。我还要感谢这三年来和我朝夕相处的同学和朋友们，我们一起学习，一起工作，一起出差，一起玩耍，相互帮助分享喜怒哀乐，共同度过了一段美好时光。

最后，我要感谢辛勤培育我的父母，感谢你们二十多年的养育之恩以及在我求学路上的坚定支持，没有父母作为我坚强的后盾我不可能这么顺利的完成自己的学业。我的求学生涯即将结束，唯有在走上工作岗位之后继续努力拼搏，去为家人创造更美好的生活，才能报答他们对我的养育之恩！

参考文献

[1] 杨光,丁国栋,屈志强.中国水土保持发展综述[J].北京林业大学学报(社会科学版),2006,5(增):72-77.

[2] 李菲.基于 WebGIS 的水保动态信息发布系统的设计与实现[D].武汉:华中科技大学,2007.

[3] 赵水宁,邵军力.多 Web 服务器负载均衡技术的研究[J].电信科学,2001(7) : 6- 8.

[4] 戴翔,周晓峰.计算机集群中负载均衡技术的研究[J].常州工学院学报,2005,18(6) : 43- 46.

[5] 熊欢.监测站远程自动数据采集系统设计与实现[D].武汉:华中科技大学,2015.

[6] 张侠.水土保持研究综述[J].地质技术经济管理,2004(6):29-30.

[7] 水利部水土保持司.水土保持信息化建设现状及基本思路[J].中国水利, 2004(4): 49-51.

[8] 修长虹,赵云飞,宋继侠.基于 Linux PC 集群负载均衡的研究与实现[J].沈阳师范大学学报: 自然科学版, 2006, 24(2) : 192- 195.

[9] 郭劲,李栋,张继征等.Iscsi,CIFS,NFS 协议的性能评测[J].小型微型计算机系统, 2006,27(5):833-836.

[10] Andrew File System. Maintenance Release for Unix/Linux[EB/OL]. [2014, 01,24]. http: //www.openafs.org/.

[11] 魏薇,孙世新.XFS 日志文件系统的关键技术研究[J].电脑开发与应用,2005,3 (26): 26-27.

[12] 李柱.分布式文件系统小文件性能优化技术研究与实现[D].长沙:国防科学技术大学,2008.

[13] 李永盛.基于并行文件系统的集群高可用性研究与应用[D].沈阳:中国科学院沈阳计算技术研究所,2009.

[14] Jacob C. Thebault-Spieker. Can Protocol and Application Layer Statistics Improve Client-server Responsiveness[C].43rd Annual Midwest Instruction and Computing Symposium 2010, 2010:258-268.

[15].U.S. Rawat,Shishir Kumar. Distributed Encrypting File System for Linux in User-space [J].International Journal of Computer Network and Information Security,2012,4(8).16-28.

[16] Likun Liu,Yongwei Wu,Guangwen Yang et al.ZettaDS: A Light-weight Distributed Storage System for Cluster[C].The Third ChinaGrid Annual Conference , 2008: 158-164.

[17] 张媛,于冠龙,卢泽新等.并行网络文件系统 PNFS 性能评测与分析[J].计算机工程与应用,2009, 45(35):65-69.

[18] 董晓明,李小勇,程煜.分布式文件系统的写性能优化[J].微型电脑应用,2012，28(12):8-11.

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

[20] H. Stone. Multiprocessor Scheduling with the Aid of Network Flow Algorithms [J]. IEEE Transactions on Software Engineering, 1997, 3(1):185-203.

[21] H. Bokhari. On the Mapping Problem [J]. IEEE Transactions on Computers, 1981, 30(5):207-214.

[22] F. Towsley. Allocating Programs Containing Branches and Loops Within a Multiple Processor System[J]. IEEE Transactions on Software Engineering, 1986 , 12(10) : 1018-1024.

[23] K. Yadav, P. Singh, H. Kumar. Scheduling Algorithm: Tasks Scheduling Algorithm for Multiple Processors with Dynamic Reassignment [J]. Journal of Computer Systems, Networks, and Communications, 2008:1-9.

[24] Norman M G, Thanisch P. Models of Machines and Computation for Mapping in Multicomputer[J].ACM Computing Surveys,1992,25(3):263-302.

[25] Zheng G B. Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing [D].University of Illinois at Urbana-Champaign, 2005.

[26] Andrews GR, R. D. Shlichting.R.Hyaes, and T. D. M. Purdin.The Design of the Saguaro Distributed Operating System[J].IEEE Transactions on Software Engineering,

1987,13(l),10-4108.

[27] Popke G J.and B.J.Walker. The Locus Distribute System Architecture[M].the MIT Press , 1985.

[28] Sihna.P.K.et al. The Galaxy Distributed Operating System [J].IEEE Computers, 1991, August, 34-41.

[29] LevyE.and Silberscehatz . A Distributed Files Systems: Concepts and Examples[J]. ACM Computing Surveys,1990,22,321-374.

[30] Satyanarayanan M.A Survey of Distributed File Systems[J]. Annual Review of Computer Science,1990,4,73-104.

[31] 于庆.分布式文件系统 FastDFS 架构剖析[J].程序员,2010,11:63-65.

[32] 杨传辉.大规模分布式文件存储系统原理解析与架构实战[M].北京:机械工业出版社,2013,74.

[33] 张友东.浅谈 Facebook 图片存储系统 Haystack 概要[EB/OL]. [2011,05,16]. http;//storage.ctocio. com. cn/349/12120849. shtml.

[34] MogileFS. Mogilefs[EB/0L]. [2012, 01,09]. http://code.Google.com/p/mogilefs/.

[35] Hamdi M. and Lin C.K. Dynamic Load Balancing of Data Parallel Applications on a Distributed Network [C]. In 9th International Conference on Supercomputing, ACM, 1995: 170-179.

[36] Willebeek Le Mair, M H Reeves A P. Strategies for Dynamic Load Balancing on Highly Parallel Computers[J].IEEE Transactions on Parallel and Distributed Systems,1993, 4(9) : 979-993.

[37] 刘同.负载均衡技术在数据库集群系统中的应用与实现[D].长沙:国防科学技术大学,2009.

[38] 魏雪波.基于 MooseFS 的云存储系统的研究与实现[D].成都:电子科技大学大学,2013.

[39] Mary-Lo V. Heuristic Algorithms for Task Assignment in Distributed Systems [J]. IEEE Transaction on Computers, 1998, 31(11), 1384-1397.

[40] Bany Wilkinson Michael Allen.并行程序设计[M]. (陆鑫达译)北京:机械工业出版

社,2005.

[41] Buyya R.高性能集群计算:编程和应用[M]. (郑纬明译).北京:电子工业出版社,2001.

[42] 郭索彦.生产建设项目水土保持监测实务[M].北京:中国水利水电出版社,2014.

[43] 周全,鄢铁平,廖伟.湖北省水土保持监测网络与信息系统建设[J].中国水土保持,2008（5）:45-47.

[44] 中华人民共和国水利部.全国水土保持信息化规划（2013-2020 年）[J].2013:28-30.

[45] 袁爱萍,杨坤,韩新启,等.北京市水土保持监测数据管理系统的研发与应用[J].中国水土保持,2006,12(4):48-50.

[46] 胡建,张亚美.广东省水土保持信息系统需求分析[J].中国农村水利水电,2002,(12):45-46.

[47] 朱亚军,杨小琴,汪兰芳.湖北省水利“一张图”服务共享平台建设与应用[J].水利信息化,2014,(1):54-58.

[48] 胡建,孙书.广东省水土保持信息系统需求分析[J].广东水利水电,2005,(6):45-46.