分类号：　　学校代号：11845

UDC：　　密级：　　学　　号：2110803241

广东工业大学硕士学位论文

（工学硕士）

基于 Hadoop 的外观专利图像检索系统的研究与实现

王贤伟

指导教师姓名、职称：戴青云 教授

企业导师姓名、职称：无

专业或领域名称：信号与信息处理

学生所属学院：信息工程学院

论文答辩日期：2011年6月6日

A Dissertation Submitted to Guangdong University of Technology for the Degree of Master of Engineering Science

Research and Implementation of Design Patent Images Retrieval System Based on Hadoop

Master Candidate: Wang Xianwei

Supervisor: Prof. Dai Qingyun

June 2011

School of Information Engineering

Guangdong University of Technology

Guangzhou, Guangdong, P.R.China, 510006

摘 要

传统外观专利检索是基于文本的查询方式，该模式无法充分利用外观专利图片所包含的丰富视觉信息，检索范围大，且图形的相似性主要靠人工识别，工作量大、效率低。为解决传统检索模式所存在的问题，通过基于内容的外观专利图像检索技术，利用图像的形状、纹理和颜色特征对外观专利图像进行描述，根据这些图像特征与专利库中的图像提供相似性判别，实现外观专利图像查询、检索自动化，提高外观设计相似性评判的检索速度和查准率。

但图像检索是数据密集型计算过程，进行图像检索时，将消耗大量CPU资源。现有B/S单节点架构的图像检索系统，随着外观专利数量的快速增长，存在检索速度慢、并发性差、不能处理大规模的数据。在分析现有图像检索系统的基础上，提出了一种基于Hadoop的外观专利图像检索方法，将基于内容的图像检索技术与MapReduce并行计算框架相结合，把外观专利图像和图像特征库存储于HDFS。

Hadoop分布式系统进行图像检索作业时，系统对专利图像特征库进行分割。各数据分块传递给Hadoop分布式系统中各计算节点的Map任务，Map任务以键值对的形式读取专利图像库的特征数据，并提取示例图像的形状、纹理以及颜色特征，与专利特征库中的特征进行相似度匹配计算，计算结果以键值对的形式输出。Reduce任务接收各Map任务的计算结果，按相似度大小进行排序，得到图像检索结果，实现图像检索的分布式计算。

通过普通的PC机搭建Hadoop分布式环境，将开发的图像检索应用程序在Hadoop分布式系统上测试，与现有图像检索系统进行比较。实验结果表明，该方法能够均衡系统负载，提高资源利用率，有效降低了在大数据集上进行图像检索的时间，并对Hadoop分布式系统的负载均衡、可靠性以及可扩展性进行了分析。

关键词：Hadoop；MapReduce；外观设计专利；图像检索；分布式计算

Abstract

Traditional retrieval on design patent is based on the mode of text query, it can not fully utilize the abundant visual information included in the design patent images in large scale retrieval, and the similarity of images is mainly based on artificial recognition with heavy workload and low efficiency consequently. To solve the problems existed in the traditional retrieval mode, we describe design patent images utilizing the character about shape, texture and colour of images by the content-based retrieval technology. According to the character of images and the discrimination of similarity provided by the image patent database, we make the design patent image query and retrieval becoming automatical and improve the retrieval speed and accuracy of design similarity judging.

However, image retrieval, which is a data-intensive computing process, consume high CPU usage rate when do the image retrieving.Low retrieval speed, bad concurrency, low efficiency of processing mega data exist in the image retrieval system in single node of B/S structure along with high speed increasing of the design patent quantity.This paper propose a Hadoop-based image retrieval method on design patent according to the analysis of the system and apply the content-based retrieval technology to MapReduce parallel computing structure, store the design patent images and their character database in HDFS.

When the retrieval jobs processing in Hadoop distributed system, the system split the character database of patent images into several data modules and transmission them to Map tasks in each computing node of HDFS. Map tasks read the character data in the form of key/value and extract the character of shape, texture and colour, then, do the similarity matching computing with the character in database.Computing results also output in the form of key/value. Reduce tasks receive all the Map tasks computing results, order

them by similarity and output the image retrieval results.This process presents a distributed computing of image retrieval.

Building up a Hadoop distributed environment with bargain price PCs and running the retrieval application program in the distributed system compared with the retrieval system in existence, this method can balance the system loading, improve the utilizing rate of resource, reduce the time of retrieving mega data collection efficiently and analyse the loading balancing, reliability and scalability of the Hadoop distributed system.

Keywords: Hadoop; MapReduce; Design Patent; Image Retrieval; Distributed Computing

目 录

Contents

第一章 绪论

1.1 研究背景及意义

外观设计是指对产品的形状、图案、色彩或者其结合所做出的富有美感并适于工业上应用的新设计[1]。随着全球经济的一体化，我国的经济、科学技术以及文化创新都取得了长足的进步，知识产权日益成为国家发展的战略性资源和国际竞争力的核心要素。同时，各行业的产品外观设计越来越趋于多元化和个性化，外观设计逐渐成为产品的核心竞争力[2]。

外观设计专利一直被誉为“小专利，大市场”，在市场竞争中发挥着举足轻重的作用。加快专利信息化发展，是增强我国产业、行业、企业竞争力的决定性因素，我国产业的发展对外观设计专利信息的需求也大量增加，随着企业对知识产权保护意识的不断增强，对外观设计专利信息的应用将非常广泛。

随着我国企业、单位和个人对知识产权的重视以及国家的相关鼓励政策，我国外观设计专利申请数量在快速增长。截止2011年3月，我国外观设计专利达1460728件，且申请量正以每年近30万件的速度增长[3]。这对外观设计专利信息的运用提出了更高的要求。目前我国外观设计专利信息的利用状况不容乐观。一方面，外观设计专利文献检索采用关键字检索模式，其检索范围大，且图形的相近似性主要靠人工识别，工作量大、效率低，外观设计专利“相近似”判断缺乏一种客观评判尺度[4]。另一方面，外观设计专利服务形式单一，缺少外观设计专利行业数据库、行业外观设计专利态势分析和外观设计专利战略分析等等。

我国对外观设计专利实行初步审查制度，这种制度经常导致多个专利人拥有相同或相近的外观设计专利，产生外观设计侵权纠纷[6]。外观设计专利的申请以及侵权纠纷的处理过程中，需要查询相关专利信息。目前我国的专利检索主要是基于专利的著录信息(外观专利的专利号、申请人等专利信息)进行查询，该检索模式范围大，专利外观的相似性主要靠人工识别，工作量

大，效率低。

由于外观专利图像中包含的丰富视觉信息(形状、纹理及颜色等信息)，而这些视觉信息通常难以用文本进行客观的描述[6]。因此，一些学者提出了基于内容的外观专利图像检索技术[7]，通过专利图像的形状、纹理和颜色等特征对外观设计专利图像描述，对专利图像特征库进行匹配，得到检索结果。利用基于内容的图像检索技术，建立外观专利图像库，实现外观设计专利图像查询、检索自动化，提高外观设计近似性评判的检索速度和查准率。

然而，对于一个外观专利图像检索系统，随着外观设计专利数量的快速增长以及图像检索操作的高并发操作，外观专利图像检索系统的实时性和稳定性将急剧下降。因为图像检索是数据密集型计算(Data Intensive Computing)过程[8]，在处理大规模数据甚至海量数据时，传统B/S单节点系统的实时性和稳定性不能得到保障。分布式计算为以上问题提供了解决思路，Hadoop是一个分布式计算框架，提供了分布式数据存储和处理的解决方案，为海量数据的并行处理提供了基础。

通过广泛的调研和分析，以现有的广东省家具行业外观专利图像检索服务平台为研究基础，采用Hadoop技术、图像检索技术和数据库技术相结合，研究适合面向公众的外观专利检索服务平台，实现基于内容的外观专利图像检索。为公众提供快速、准确的外观专利检索，基于Hadoop的外观专利图像检索服务平台的研究与实现，对于企业的科技创新、战略决策，提高产品质量和外观专利设计水平，减少专利侵权，都有着重要和深远的意义。

1.2 国内外研究现状

目前，我国外观设计专利信息检索服务还处于发展初期的起步阶段。外观专利信息的查询一般是通过提供的关键字进行查询，通过查询的关键字与外观专利著录信息的匹配而得到相关的结果。或通过专业代理机构和专利代理人进行检索，甚至是通过下载专利说明书及向相关单位订购专利光盘进行检索查询。

国内已开发出了一些专利检索系统，如国家知识产权局开发的专利检索系统，该系统数据权威，更新及时，但不能查询专利的法律状态，且检索速

度慢。国家知识产权出版社开发的专利检索系统，该系统数据更新及时，检索格式丰富，正式用户可查询“主权项”、“法律状态”以及下载专利的全文，但不是免费对公众开放。北京东方灵盾、保定大为以及博派专利等企业也都提供各种专业情报数据库和多数据联机检索、分析和管理平台，为社会各界提供全方位、专业化的专利战略分析以及侵权分析、专利预警咨询和知识产权管理咨询等高端服务。国外一些著名的专利检索系统有：美国专利商标局、欧洲专利局、日本特许厅、汤姆森集团的DIALOG国际联机检索系统、美德日联合建立的STN国际联机检索系统和法国的Questel-Obit国际联机检索系统等专利信息服务系统。但这些国内外的专利检索系统都不提供专利图像检索。

上世纪90年代初，随着大规模图像集的不断涌现以及快速发展的数据库技术和计算机视觉技术，传统的基于文本的图像检索已经不能满足人们的需求，一些专家学者们相继提出了基于内容的图像检索技术(Content-Based Image Retrieval)。研究人员对基于内容的图像检索技术进行了深入的研究，使该技术得到了广泛的应用，如知识产权保护、新一代网上搜索、数字图书馆、医学和遥感图像的分析和处理等领域。鉴于基于内容图像的图像检索技术的重要性，国内外的许多机构都在进行相关的技术研究，并已研究出了一些系统，如IBM公司开发的QBIC[9]、Virage公司开发的Virage图像搜索引擎[10]、MIT多媒体实验室开发的Photobook[11]、哥伦比亚大学开发的VisualSEEK[12]以及华中科技大学开发的基于网格平台的分布式医学图像检索系统。

华中科技大学开发的分布式医学图像检索系统中，用户通过网格门户向网格域管理中心提交医学图像检索请求。域管理中心将请求与资源元数据进行匹配，获取可用的医学图像专家库的网络服务标识，然后将请求分发给对应的医学图像专家库。医学图像专家库提取提交图像的形状、颜色、纹理等特征，并与库存的医学图像特征进行比对，获取与提交图像的综合相似度具有最大值的一组图像。域管理中心将各医学图像专家库返回的医学图像进行整合，并返回给用户。该系统提高了系统软硬件的利用效率，降低了医疗单位实现分布式医学图像检索需求的整体成本，为医学图像资源的广泛共享提供了有效的途径。但该系统是基于网格计算实现的，网格计算中的资源一般都是不同机构中的异构资源，因而安全问题尤为重要，必须研究面向网格应

用开发的资源访问控制机制和用户身份认证机制；网格编程接口具有比传统编程接口更多、复杂的属性，对开发者有更高的要求；此外，网格计算一般需要高性能的计算机，一般用于科学界，在商业界没有得到广泛应用。

2002年，戴青云教授提出了基于内容的外观设计专利图像检索技术，通过图像的形状、纹理和颜色特征对外观设计专利图像进行描述，对专利图片的特征进行比对得到两者的相似度。利用基于内容的图像检索技术，建立外观专利图像库，实现外观设计专利图像查询、检索自动化，提高外观设计相近似性评判的检索速度和查准率。

目前，国内已开发了一些外观设计专利智能检索系统。如2008年国家知识产权局开发的中国外观设计专利智能检索系统[13]。广东省专利信息中心与广东工业大学在2009年联合开发的面向重点行业、产业的外观设计专利智能检索系统[14]，该系统将基于内容的图像检索技术、.NET网络技术以及数据库技术相结合，采用三层B/S架构实现。这些系统检索的准确度高，但检索时间无法让人满意。特别在多用户并发操作以及面对外观专利数据量的快速增长时，系统的实时性急剧降低，如何实现大规模数据的处理是提高图像检索系统实时性的关键。

对于大规模数据的处理，分布式计算是一个有效可行的方法。现行的分布式计算系统可以提供强大的计算能力，但实现比较困难。近年兴起的云计算为分布式处理的实现提供了新思路，云计算(Cloud Computing)[15]是将计算任务分布在大量计算机构成的资源池上，使各种应用系统能够根据需要获取计算力、存储空间和信息服务。云计算是分布处理(Distributed Computing)[16]、并行处理(Parallel Computing)[17]和网格计算(Grid Computing)[18]的发展，或者说是这些计算机科学概念的商业实现。云计算是虚拟化(Virtualization)、效用计算(Utility Computing)、IaaS(基础设施即服务)、PaaS(平台即服务)、SaaS(软件即服务)等概念混合演进并跃升的结果，它已经被看作IT业的新趋势，并广泛应用于天文、医学、网络安全、图形和图像处理以及互联网等领域[19-21]。

Hadoop是Cloud Computing计算模型的一个开源实现，它得到了Google、Yahoo!、IBM、Amazon、FaceBook等公司的大力支持[22-24]。Hadoop主要用于大规模及海量数据的存储和处理，提供分布式计算框架，可以轻松的实现并

行计算，Hadoop的这些优点正好可以解决外观专利图像检索所遇到的问题。对于Hadoop在图像检索方面的应用，目前国内外还没有相关的研究。

1.3 研究内容及创新点

对于图像检索技术，首先是将图片库中的图片进行特征提取，得到图片特征库；当用户提交待检索的图片时，进行待检索图片的特征提取；然后将待检索图片特征与特征库中的图片进行特征比对，根据相关的相似度量算法，计算得到图片库中与待检索图片相似的图片，按照相似度的高低排序后将检索结果返回给用户。

由于在进行图像检索的数据处理时，是一个数据密集型计算过程，需要消耗大量的CPU资源。当计算的数据量比较小时，单台服务器进行处理可以很好的完成；但是，当数据量很大时，由于单台服务器资源的限制，那么得到结果将是一个漫长的过程。而当处理的数据到达海量级别时，单台服务器将几乎无法完成检索任务。此时，可考虑采用分布式计算来完成海量数据[26]的处理。

通过对我国外观专利设计的现状和实际需求的调研基础上，密切结合用户对外观专利图像的实际需求，全面分析国内外图像检索技术和专利信息服务的现状和发展趋势，有效整合并充分利用现有的技术积累和服务经验，针对传统B/S单节点模式以及外观设计专利数据快速增长所带来的瓶颈，提出并实现了基于Hadoop的外观设计专利图像检索系统。

本文的主要内容包括以下几点。

(1) 深入分析传统外观专利图像检索的不足以及用户对于外观专利图像检索的需求，根据用户的需求和系统的业务流程，提出通过分布式计算来解决处理数据快速增长所带来的瓶颈。

(2) 将Hadoop技术与图像检索技术相结合，提出了基于Hadoop的外观设计专利图像检索系统，并完成了系统平台的体系结构设计。

(3) 在MapReduce框架之上，实现了能够执行并行计算的图像检索算法。结合HDFS，实现图像检索过程中的分布式计算，从而提高了图像检索的实时性。

(4) 通过Hadoop分布式系统平台的搭建以及测试，分析系统存在的问题，并加以优化。

本文的创新性研究工作体现在：通过分布式处理技术实现系统的水平伸缩性扩展、将空闲的资源充分利用，从而提高系统的资源利用率；运用Hadoop技术，结合图像检索技术，在Hadoop集群中通过MapReduce框架，将用户的图像检索请求分配给集群中“空闲”节点服务器进行处理，从而解决服务器高度并发访问以及海量数据处理所带来的实时性问题，并提供可靠的检索服务。

1.4 本文结构

本文共分为五章，说明了实现基于Hadoop的外观专利图像检索服务平台所采用的技术、设计思路、方法以及实现和测试的过程。其结构和内容安排如下：

第一章 介绍了课题的研究背景和意义、国内外现状以及本文研究的主要内容。

第二章 介绍了Hadoop的相关技术，主要是关于HDFS和MapReduce，此外还对图像检索技术进行了介绍。

第三章 基于Hadoop的外观专利图像检索系统的设计，在对图像检索系统总体分析的基础上，提出了系统的整体架构，并对系统各模块的设计进行了深入的介绍。

第四章 并行图像检索算法的实现，首先深入介绍了MapReduce并行编程，然后将图像检索算法与MapReduce技术相结合，实现了在Hadoop平台下的外观专利图像检索的匹配，并在Windows环境下进行开发和测试。

第五章 系统测试与分析，通过搭建Hadoop分布式系统，对分布式系统进行图像检索测试，分析系统的运行情况以及存在的问题，并进行优化处理。此外，将测试结果与传统B/S单节点模式的测试结果进行对比分析，并对Hadoop分布式系统的负载均衡、可扩展性进行了分析。

最后，总结与展望，总结了本文的主要工作内容，并对下一步的研究进行了展望。

第二章 Hadoop与图像检索相关技术

基于Hadoop的外观专利图像检索系统中，Hadoop平台提供分布式文件系统(HDFS)以及并行编程框架(MapReduce)，将基于内容的图像检索技术应用于MapReduce框架，实现分布式的外观专利图像检索系统。本章首先介绍Hadoop平台相关知识，包括HDFS和MapReduce的整体框架以及工作原理；最后对基于内容的图像检索技术及检索过程中的相似度计算进行了阐述。

2.1 Hadoop平台简介

Hadoop[26]是一个分布式系统基础架构，由Apache基金会开发。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。它是一个基于Java实现的，开源的，分布式存储和计算的项目，Hadoop项目起源于Apache Nutch(一个开源的网络搜索引擎，是Lucene项目的一部分)。Hadoop的设计思想以及原理起源于Google发表的GFS[27](Google File System)、MapReduce[28]以及BigTable[29]这三篇论文。

Hadoop主要包括HDFS[30](Hadoop Distributed File System)、MapReduce以及HBase。HDFS是GFS的开源实现，MapReduce是Google MapReduce的开源实现，HBase是Google BigTable的开源实现。后来Hadoop项目发展的子项目有Pig[31]、Hive[32]、ZooKeeper[33]等。Hadoop的核心部分是HDFS以及MapReduce[34]，Hadoop系统的结构框图如图2-1所示，最底层的是HDFS，HDFS是一个分布式文件系统，它存储Hadoop集群中所有存储节点上的数据。HDFS的上一层是MapReduce框架， MapReduce是分布式数据处理模型，也就是并行处理框架，HDFS为MapReduce的并行处理提供处理数据。目前，Hadoop已经广泛应用在大规模数据的存储[35]、搜索引擎[36]、数据挖掘[37-38]、并行处理[39-40]、生物医学[41]以及虚拟数据库[42]等领域。

Hadoop 是一个能够对大规模数据进行分布式处理的软件框架，它的广泛应用是由于具有以下优点：

(1) 开源性：Hadoop是一个开源的软件平台，得到了Yahoo!、Facebook等大型公司以及开源社区的支持。使得Hadoop得到了快速的发展，并得以完善。

(2) 可扩展性：Hadoop的扩展包括存储的扩展以及计算能力的扩展，Hadoop的扩展非常简单，只需要将扩展的节点添加至Hadoop集群，不需要改变Hadoop集群的已有结构。

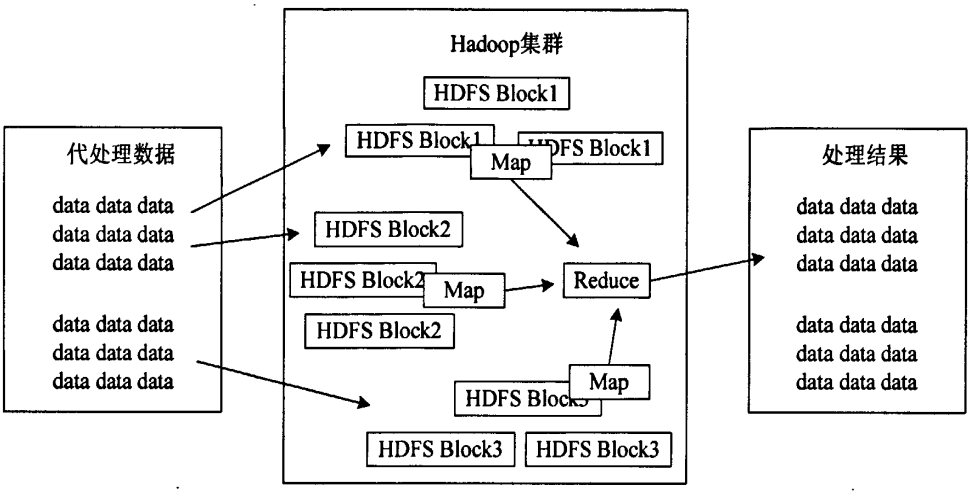

图2-1 Hadoop系统结构框图

Fig.2-1 Framework of Hadoop system

(3) 可靠性：Hadoop考虑到计算和存储节点可能失败，它可以维护多个工作任务以及数据副本(计算任务有备份任务，以防止任务执行失败，数据通过保存一定的副本来解决单点失效问题，Hadoop集群中数据的默认备份数为3)，确保能够针对失败的节点重新进行处理。

(4) 高效性：通过分布式文件系统以及MapReduce框架，使得Hadoop集群上的各节点能够并行地处理任务，从而提高了系统的整体效率。

(5) 廉价性：Hadoop集群可以由成千上万台普通的PC机组成，进行数据的存储与处理，对硬件没有特殊要求。

(6) 跨平台性：Hadoop是基于Java开发的开源软件，可以运行在多种操作系统和商用硬件上。

2.2 Hadoop分布式文件系统(HDFS)

HDFS是一个针对大文件进行优化的分布式文件系统，和其他分布式文件系统有很多类似的特点，分布式文件系统的几个基本特点是：(1)、对于整个集群有单一的命名空间。(2)、数据一致性。适合一次写入多次读取的模型，客户端在文件没有被成功创建之前无法看到文件存在。(3)、文件被分割成多个文件块，每个文件块被分配存储到数据节点上，根据系统的配置，文件块保存一定的副本数，从而保证数据的高可靠性。

HDFS以流式数据访问模式存储超大文件而设计的文件系统，HDFS可存储GB、TB甚至PB大小级别的文件，在商用硬件上运行。HDFS中，可像传统的文件系统那样进行操作(如创建、删除、移动以及重命名等，不过目前的HDFS还不支持文件的修改)，HDFS进行的复杂处理对于用户而言是透明的。

HDFS具有以下几个方面的特点：(1)、存储超大文件，HDFS非常适合大规模数据集的存储和管理。(2)、高度容错性，HDFS最大的特点是文件块的冗余存储，为MapReduce任务的“推测式执行”[34]提供了数据支撑。(3)、普通硬件，Hadoop适合部署在廉价机器上，对于大集群，节点发生故障的概率比较高，通过冗余存储，可以解决节点失效问题。(4)、流式数据访问，HDFS设计成适合批量处理，不是用户交互式的，提供高吞吐量的数据访问。

2.2.1 HDFS 整体框架

HDFS采用Master/Slave架构，由主节点NameNode、数据节点DataNode以及SecondNameNode组成。客户端通过与NameNode和DataNode的交互，访问整个文件系统。HDFS框架结构如图2-2所示。

NameNode管理文件系统的命名空间，它维护整个文件系统中文件以及索引目录，将所有的文件和文件夹的元数据保存在一个文件系统树中。这些信息以命名空间镜像以及编辑日志文件保存在本地磁盘上，可以通过这些信息设置备份的NameNode，即SecondNameNode。在Hadoop启动时，通过各DataNode收集各数据块的信息。

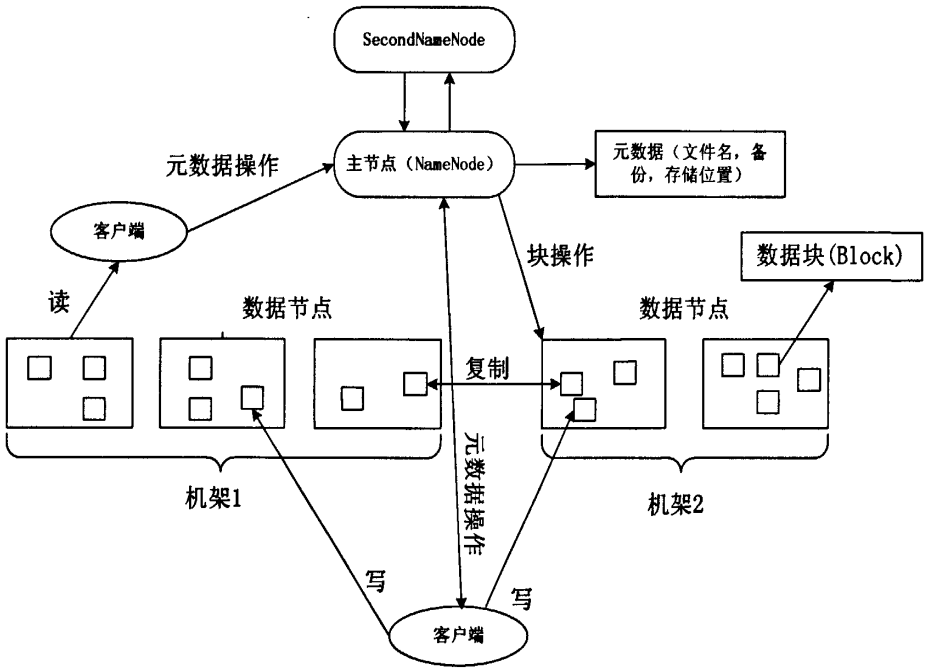

图2-2 HDFS框架结构

Fig.2-2 Framework of HDFS

DataNode负责存储每个文件的各数据块，各数据块的存储位置不是固定的，它随Hadoop系统的调整而改变，HDFS中数据块的默认大小为64M，各DataNode通过心跳向NameNode周期性的报告数据块的信息。客户端通过对NameNode请求，从而对数据节点中的数据进行读写操作。

SecondNameNode是NameNode的备份节点，它周期性将NameNode节点的命名空间镜像文件和修改日志合并，以防日志文件过大。合并过后的命名空间镜像文件在SecondNameNode节点保存了一个备份，以防NameNode节点失效，从而可以恢复Hadoop系统。

2.2.2 HDFS 中数据读取

HDFS中数据的读取需要客户端与NameNode和DataNode协同完成，客户端首先对NameNode发出读取请求以获得请求数据的信息，然后再到相应的DataNode读取数据块。图2-3所示为客户端从HDFS中读取数据的过程。

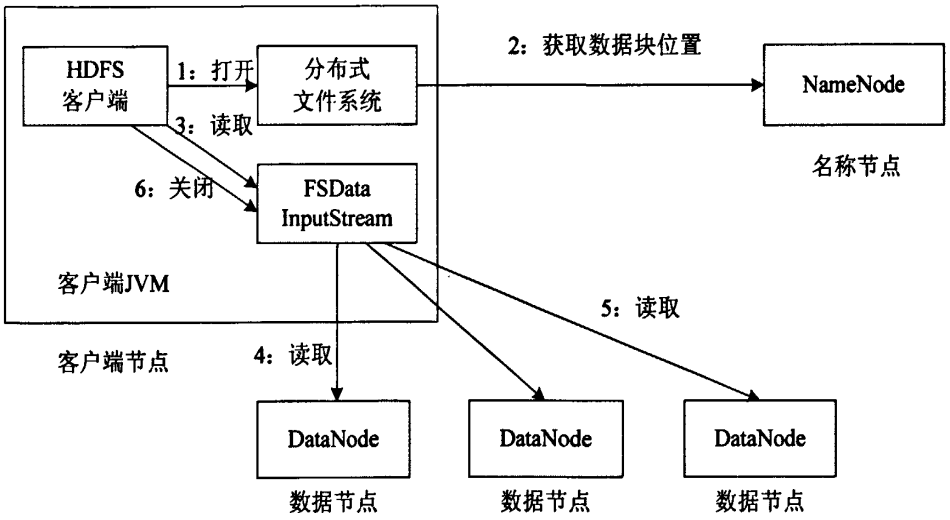

图2-3 HDFS中读取数据的过程

Fig.2-3 Process of Data-reading in HDFS

HDFS中数据的读取过程为：

(1) 客户端通过调用FileSystem对象(该对象是分布式文件系统)的open( )来读取希望打开的文件。

(2) 分布式文件系统通过RPC来调用NameNode，以确定文件开头部分的数据块位置。对于每个数据块，NameNode返回具有该数据块副本的数据节点地址。分布式文件系统返回一个FSDataInputStream(负责DataNode与NameNode之间的通信)对象给HDFS客户端进行读取数据。FSDataInputStream中封装了一个DFSInputStream对象。

(3) 客户端通过调用FSDataInputStream对象中的read( )方法，准备读取数据块。

(4) DFSInputStream中存储着文件起始数据块的地址，并与最近的数据块建立连接，从而读取数据，将数据返回给客户端。

(5) 当一个数据块读取成功后，DFSInputStream将关闭当前数据块所建立的连接，然后与文件的下一个数据块建立连接，并读取数据返回给客户端，客户端只是读取一个连续的数据流，对于各数据块的读取操作对客户端而言是透明的。

(6) 当客户端读取完成后，将调用DFSInputStream中的close( )方法关闭连接。

客户端在读取数据的时候，如果客户端与DataNode发生通信错误时，客

户端将会尝试读取该数据块的副本，同时将出错的数据块标记，并将信息传递给NameNode，以防止对出错数据块的再次操作。

2.2.3 HDFS 中数据写入

HDFS中数据的写入也需要客户端与NameNode和DataNode协同完成，图2-4所示为客户端向HDFS中写入数据的过程。

HDFS中数据的写入过程为：

(1) 客户端调用FileSystem对象(该对象是分布式文件系统)的create( )创建文件。

(2) 分布式文件系统对象通过RPC调用NameNode，在NameNode的命名

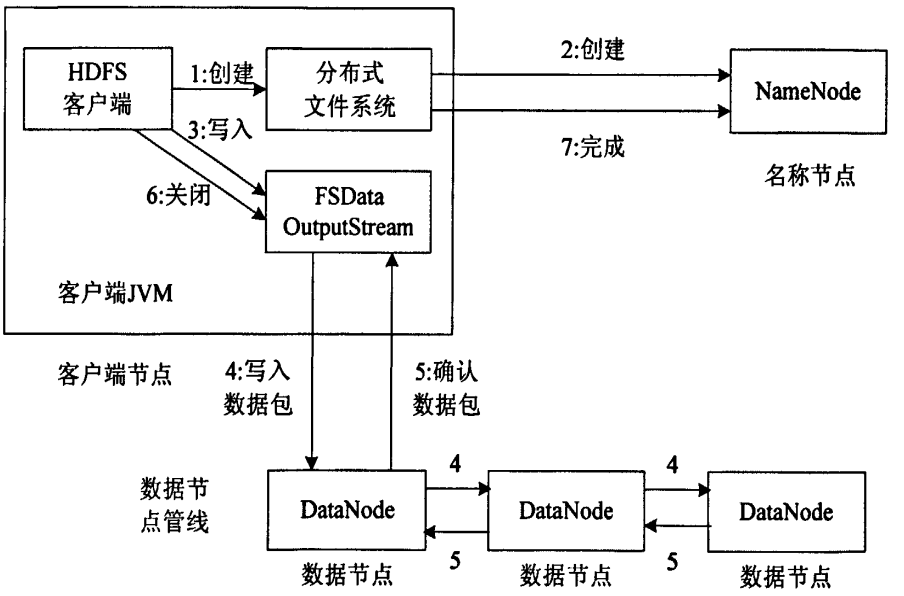

图2-4 HDFS中数据的写入过程

Fig.2-4 Process of Data-writing in HDFS

空间中创建一个新的文件(首先检查命名空间中是否存在该文件，如果存在，则创建失败)，分布式文件系统返回一个FSDataOutputStream(负责DataNode和数据节点的通信)，以便客户端进行写入操作。

(3) 客户端开始写入数据时，DFSOutPutStream将数据分为一个个数据包，写入内部的数据队列。数据队列的流动是根据NameNode的分配而确定的(确定数据块写入那些DataNode，各数据块默认副本数为3份，各DataNode

形成一个管道)。

(4) 数据流对象将数据包写入管道中的第一个DataNode，该DataNode同时会将写入的数据发送至管道中的第二个数据节点。同样，第二个数据节点会将写入的数据发送至管道中的第三个数据节点。

(5) 向DataNode节点写入数据的同时，DFSOutPutStream也管理一个确认队列，该对列用来接收各数据节点数据写入的情况，只有当收到管道中所有数据节点写入成功后，才表示当前数据块写入成功，并进入下一个数据块的写入。

(6) 当所有数据块写入成功后，将调用DFSOutPutStream的close( )，结束数据块的写入。

(7) 通知NameNode，写入操作完成。

在数据的写入期间，如果发生写入失败，管道首先被关闭，将确认队列中的数据块放入数据队列的开始，以保证写入错误的数据块能够重新写入。当前的数据块在已经写入的数据节点中被NameNode给以新的标记，而出错数据节点中的数据块将被删除。失败的数据节点将从管道中删除，另外的数据块将被写入管道中的另外两个数据节点。由于有数据节点写入失败，NameNode会被通知该数据块备份不足，将重新创建该数据块的备份。

2.3 MapReduce 并行编程模型

MapReduce是一种简化的分布式编程模式，最早由Google在2004年提出，主要用于大规模数据集的并行运算，它将大规模的数据集进行分解，在集群中的各节点进行处理，以实现分布式并行处理。在Google内部，MapReduce得到广泛的应用，比如分布排序、数据挖掘和Web访问日志分析等。MapReduce被评为2009年的十大企业新兴技术之首[43]，主要是由于MapReduce能够让企业获得梦寐以求的海量数据处理能力，而它的价格是企业完全可以接受的。

Hadoop中的MapReduce是Google提出MapReduce思想的一个开源实现，下面主要是对Hadoop平台中的MapReduce进行介绍。

2.3.1 MapReduce 整体框架

MapReduce 也是 Master/Slave 架构，MapReduce 框架由一个 JobTracker(master)和一些TaskTracker(slave)节点组成。通常，MapReduce的JobTracker和HDFS的NameNode处在同一个节点，而集群中各节点同时担任TaskTracker和DataNode角色。JobTracker主要负责MapReduce任务的调度，将MapReduce任务分配到各TaskTracker上进行执行，并监控TaskTracker的执行，如果某个TaskTracker执行失败，JobTracker将在其它节点重新执行该任务。TaskTracker主要是执行JobTracker分配的任务，通常需要利用到HDFS中的数据，而这些数据通常是位于“本地”，从而实现任务的并行处理。

MapReduce整体运行框架如图2-5所示。

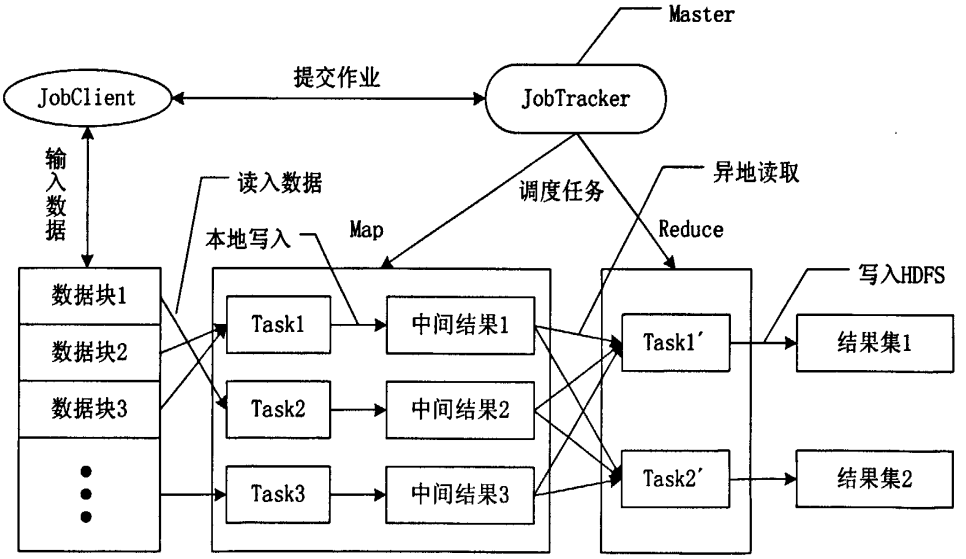

图2-5 MapReduce整体运行框架

Fig.2-5 Overall framework of MapReduce

MapReduce让程序自动分布到一个由普通机器组成的超大集群上并发执行。简而言之，就是将大数据集分解成成百上千的小数据集。每个(或若干个)数据集由集群中一个节点通过Map任务处理，产生中间结果，这些中间结果由大量的节点通过Reduce任务进行收集、合并，得到最终结果。MapReduce的run-time系统会解决输入数据的分布细节，跨越机器集群的程序执行调度，处理机器的失效，管理机器之间的通讯请求。该模式允许程序员不需要有什

么并发处理或分布式系统的经验，就可以处理超大规模的分布式系统中的资源。

2.3.2 MapReduce任务执行的各阶段

典型的MapReduce计算包括以下几个阶段：Map阶段、Combine阶段以及Reduce阶段[34]。

(1) Map阶段：MapReduce框架将输入数据拆分为大量的数据片段(split)，每个split分配给一个Map任务，Map任务将输入的split根据需求分解为一系列键/值对(Key/Value)。对于每一个输入键/值对(K1/V1)，Map任务调用用户自定义的map函数进行处理，产生一个中间结果的键/值对(K2/V2)，通常reduce函数的输入类型必须与map函数的输出类型要一致。

map函数的输入可以处理多种格式的数据，从一般的文本文件到整个数据库。一个输入分片能够被一个Map任务处理，map函数处理是通过将数据分片中的记录一条一条地处理。对于初始的HDFS数据，JobTracker通过调用InputFormat类的getSplits( )方法对数据进行分片。计算好分片后，JobTracker将根据各分片的存储信息将分片调度至各TaskTracker上。在TaskTracker上，Map任务会将输入分片传递到InputFormat的getRecordReader( )方法中，从而获得相应的RecordReader。Map任务通过调用RecordReader来读取记录并产生键/值对，将键/值对传递给map函数。RecordReader中的next( )方法会不断地读取记录，只至结尾。常见InputFormat类的实现有TextInputFormat、KeyValue-TextInputFormat、SequenceFileInputFormat以及DBInputFormat等类。

(2) Combine阶段：Map阶段完成后，Map的输出键值对(中间结果)通常暂时保存于内存中。在某些情况下，为了让Map的输出更加简洁，需要提供一个执行Reduce类型功能的Combiner类。当使用了Combiner时，combiner函数在每一个Map任务的机器上执行。通常这个combiner函数的代码和reduce函数的代码实现上是一致的。reduce函数和combiner函数唯一的不同就是MapReduce对于这两个函数的输出处理上不同。对于reduce函数的输出是直接写到最终的输出文件。对于combiner函数来说，输出是写到中间文件，且被发送到Reduce任务中。

(3) Reduce阶段：Reduce任务处理中间键(K2)以及该键的相关值集合(提供给reduce函数的中间值是通过一个iterator来提供的)，得到计算结果(键值对K3/V3)，这些操作是通过用户自定义的reduce函数完成。通常，一次reduce函数的执行会产生0个或者1个输出值。

MapReduce处理过程可以表示为如下过程：

(1) Map：(K1，V1)→ list(K2，V2)

(2) Combine：(K2，list(V2))→ list(K2，V2)

(3) Reduce：(K2，list(V2))→ list(K3，V3)

2.3.3 MapReduce 作业执行流程

要运行一个MapReduce作业，需要通过客户端提交MapReduce作业。MapReduce作业的整个运行过程如图2-6所示。

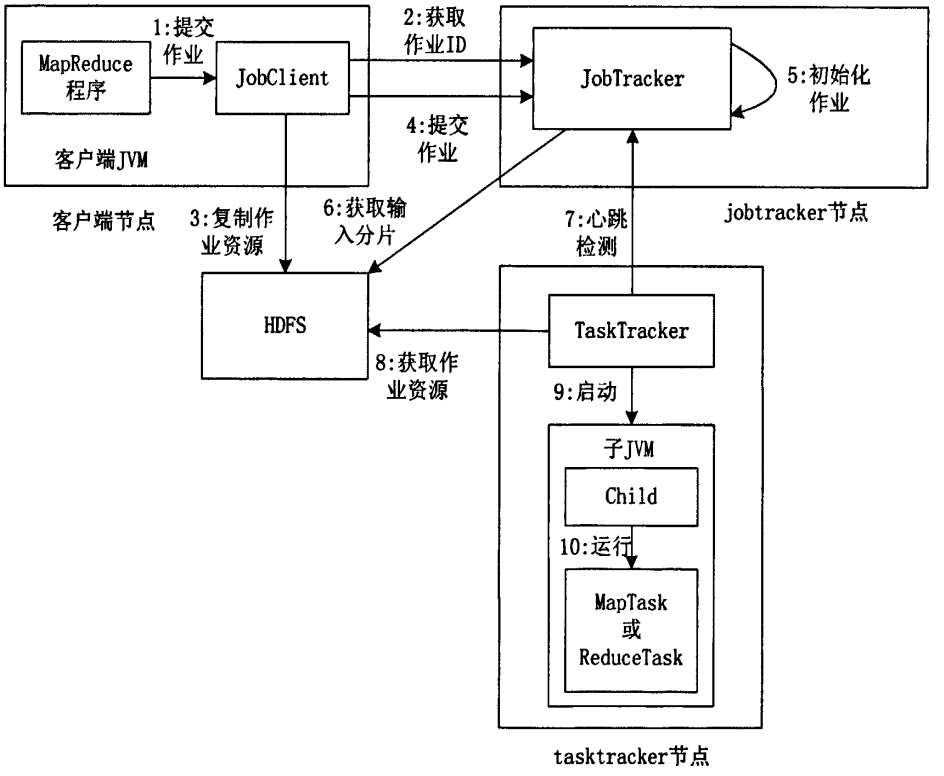

图2-6 MapReduce作业执行流程

Fig.2-6 Executive of MapReduce

运行MapReduce作业，将主要执行以下步骤：

(1) 客户端运行一个MapReduce程序(即作业)，JobClient新建一个JobClient实例，并调用该实例的submitJob( )方法，向JobTracker请求一个新的作业ID。JobTracker将检查该作业的相关信息(如输出目录是否已经存在，如果存在，该作业将不会被提交，并返回错误给MapReduce程序)。

(2) JobClient计算作业的输入划分，如果划分无法计算，作业将不会被提交，并返回错误给MapReduce程序。

(3) 将运行作业所需的资源(包括JAR文件、配置文件以及输入划分)复制至以ID号命名的目录中(在jobtracker文件系统中)，通过对JobTracker的submitJob( )调用，初始化作业。

(4) 创建运行任务列表，作业调度器从共享文件系统中获取JobClient已计算好的输入划分信息。为每个分片创建一个Map任务，同时根据MapReduce程序设置的Reduce任务数进行创建Reduce任务，并指定相应的任务ID号。

(5) TaskTracker通过心跳(heartbeat)与JobTracker进行通信，JobTracker为各TaskTracker节点分配Map以及Reduce任务，将Map以及Reduce任务指派给空闲的TaskTracker。对于Map任务，JobTracker会考虑到tasktracker的网络位置，选取的tasktracker的位置距输入分片最近，理想情况下，map任务的输入分片是data-local(数据本地化)的，即Map任务的TaskTracker以及DataNode是处于同一个节点。

(6) TaskTracker分配好任务后，将本地化作业的JAR文件，将MapReduce程序需要的数据由分布式缓存复制到本地磁盘。然后为任务建立一个本地工作目录，将运行所需内容放至该文件夹。新建一个TaskRunner实例，启动一个新的Java虚拟机运行MapReduce程序。MapReduce程序的执行将以2.3.2节所示的过程进行执行。

(7) MapReduce作业在执行期间，Map和Reduce任务将定期向TaskTracker报告执行的进度。同时，TaskTracker通过心跳定期向JobTracker报告该节点MapReduce任务的执行状况。

(8) JobTracker收到作业最后一个任务完成的通知后，将把作业标志为“成功”，当JobClient查询状态时，将得知任务已经成功执行。最后，JobTracker清空作业的工作状态，同时指示各TaskTracker节点清空作业的工作状态(如删除中间结果等)。

MapReduce任务在执行过程中，可能会产生任务的失败，这通常是由于子任务的失败、子JVM退出、TaskTracker失败或JobTracker失败所造成的。对于前两种情况，TaskTracker会注意到有一定的时间没有收到任务的进度更新，那么将该任务标记为failed。JobTracker通过TaskTracker的心跳被告知一个任务尝试失败，JobTracker将重新调度该任务的执行，但JobTracker会尝试避免在之前发生错误的TaskTracker上重新执行该任务。由于Hadoop中的MapReduce任务的执行是“推测式执行”的，当某个任务的一个进程率先执行完后，另一个推测副本将被“杀死”。

对于TaskTracker节点发生失败时，TaskTracker将无法向JobTracker发送心跳。此时，JobTracker将会将发生错误的TaskTracker上的任务移除，并将重新调度一个TaskTracker作业。对于JobTracker发生失败时，此时将无法弥补产生的错误，不过这种错误发生的几率很小。

2.4 基于内容的外观专利图像检索技术

2.4.1 基于内容的外观专利图像检索流程

目前，主流的图像检索技术还是基于文本的图像检索。基于文本的图像检索技术主要是利用文本对目标图像进行描述，如图像的作者、年份以及图像所表达的主旨等信息。对于外观专利的图像检索，国内外主要还是以外观设计专利文献作为关系型数据库进行存储。检索过程以“专利号”、“专利名称”、“专利公告号”、“发明人”等著录信息作为关键字进行检索，该检索模式检索范围大，且图像的相似性主要靠人工识别，工作量大，效率低。

针对传统基于文本的外观专利检索方式的不足，专家学者们提出了基于内容的外观设计专利图像检索技术[7]，通过图像的颜色、纹理和形状特征对外观设计专利图像进行描述，对专利图片的特征进行比对得到两者的相似度。利用基于内容的图像检索技术，建立外观专利图像库，实现外观设计专利图像查询、检索自动化，提高外观设计相近似性评判的检索速度和查准率。

图像检索的流程图如图2-7所示。

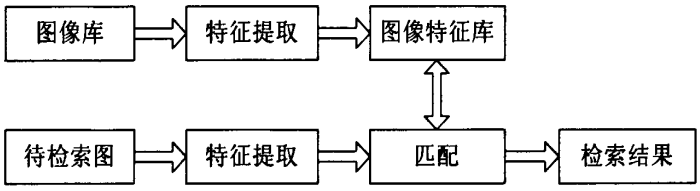

图2-7 图像检索流程图

Fig.2-7 Process of image retrieval

对于外观专利图像的检索过程，可以表示为：

(1) 将外观专利图像库中的图像进行特征提取(特征包括形状、纹理以及颜色)，形成外观专利图像特征库，以关系型数据库的形式进行保存。

(2) 当用户提交一幅图像进行检索时，提取待检索图像的特征，特征提取的方式与外观专利图像库提取的特征相同。

(3) 将待检索图像的特征与外观专利图像库中的特征进行匹配(逐条记录进行相似度计算，并按照相似度由大到小进行排序)，得到匹配结果，也即图像检索的结果。

基于内容的图像检索过程，主要是待检索图像与图像库中图像特征的相似度计算。图像特征的提取包括形状、纹理以及颜色等多维特征[44-46]，通过多特征的融合，可以有效地实现图像检索。

2.4.2 外观专利图像检索相似度计算

本文的核心是通过分布式计算来解决图像检索的实时性问题，分布式计算主要是关于图像之间的相似性计算，下面对图像的相似性计算进行阐述。

假设图像检索的示例图像为 P_0，图像库中的图像为 P_i，$i \in \{1,2,...,I\}$，I 指图像库中图像总数。对图像 P_i 特征提取，其形状、纹理以及颜色特征，分别用 $S_i \in R^L$，$T_i \in R^M$ 以及 $C_i \in R^N$ 表示，其中 L，M 和 N 分别表示图像的形状、纹理以及颜色特征的维数。相似度计算中，以形状特征为主。图像形状特征采用Hu不变矩和高斯描绘子两种算法描述，纹理特征采用灰度共生矩阵提取，颜色特征以颜色直方图描述。

对示例图像 P_0 进行特征提取，得到 P_0 的形状、纹理以及颜色特征，分别表示为 $S_0 \in R^L$、$T_0 \in R^M$ 以及 $C_0 \in R^N$，其中 L、M 和 N 分别表示图像的形状、

纹理以及颜色特征的维数。将P_0的特征与特征库中各图像特征逐条计算相似度，得到P_0和P_i的形状相似度D_{S_i}、纹理相似度D_{T_i}以及颜色相似度D_{C_i}，分别表示为：

$$D_{S_i}=1-\frac{\left(\sum_{l=1}^{L}(S_0^l-S_i^l)^2\right)^{1/2}}{\max_i\left(\sum_{l=1}^{L}(S_0^l-S_i^l)^2\right)^{1/2}} \tag{2.1}$$

$$D_{T_i}=1-\frac{\left(\sum_{m=1}^{M}(T_0^m-T_i^m)^2\right)^{1/2}}{\max_i\left(\sum_{m=1}^{M}(T_0^m-T_i^m)^2\right)^{1/2}} \tag{2.2}$$

$$D_{C_i}=1-\frac{\sum_{n=1}^{N}\min(C_0^n,C_i^n)}{\sum_{n=1}^{N}C_0^n} \tag{2.3}$$

其中，(2.1)式和(2.2)式是P_0与P_i形状和纹理特征相似度的欧式距离，(2.3)式是P_0与P_i颜色特征相似度的直方图相交距[47]，并都作了归一化处理。

设R表示两个图像的相似度，R值越大表示两个图像的相似度越高，$R_{\max}$为1，表示两个图像是同一幅图像，示例图像P_0与图像库中图像P_i的相似度可以表示为：

$$R_{0i}=W_1D_{S_i}+W_2D_{T_i}+W_3D_{C_i} \tag{2.4}$$

其中W_1、W_2以及W_3分别表示形状、纹理以及颜色的相似度权重，$i\in\{1,2,...,I\}$，且满足

$$W_1+W_2+W_3=1 \tag{2.5}$$

对特征库中的特征逐条计算相似度，得到计算结果$R_{01},R_{02},R_{03},...,R_{0I}$。对其进行由大到小排序得到检索结果$R_{01}^{'},R_{02}^{'},R_{03}^{'},...,R_{0I}^{'}$。

2.5 本章小结

本章主要介绍Hadoop以及图像检索相关技术。首先，对Hadoop平台进行整体介绍；其次，对HDFS的整体框架、数据读取以及写入进行了介绍；再次，详细介绍了MapReduce并行编程模型；最后，对基于内容的图像检索技术进行简要介绍，并对外观专利图像检索过程中的相似度计算进行了阐述。

第三章 基于Hadoop的图像检索系统设计

对基于 Hadoop 的图像检索系统的设计，需要对系统的需求及以往的工作进行分析。本章首先分析了外观专利图像检索系统的需求，进而分析了传统 B/S 架构的外观专利图像检索系统的不足。从而提出了基于 Hadoop 的外观专利图像检索系统，并对系统的设计进行了详细介绍。

3.1 外观专利图像检索系统需求

对于外观设计专利图像检索系统，需要建立一个在WEB环境下基于内容的外观设计专利图像分析服务平台。该平台主要包括以下主要内容和功能：

(1) 在WEB环境下，提供基于著录信息的外观专利检索。

(2) 在WEB环境下，提供基于内容的外观设计专利图像检索与分析，能够应对多用户、高并发时图像检索的实时要求。

(3) 提供符合人的视觉图像感知的图像相似性度量，能给出外观设计专利图像之间的相似性参考量化值。

(4) 对于用户提交的图像检索请求，需要提供高吞吐量、高可靠性的服务质量。

(5) 系统能够实现海量数据的检索，以及系统根据发展的需求，可以有良好的可扩展性。

(6) 建立专题外观设计专利行业库与外观设计专利信息分析系统，为行业、企业提供市场预测、技术走势、行业发展方向及同业对手产品外观设计专利相似性分析等增值服务。并可为政府管理和决策提供依据。

3.2 传统图像检索系统架构及存在不足

3.2.1 传统图像检索系统架构

当前，我国的外观专利数量达到150万件。对于目前数据量的外观专利

图像检索，采用传统B/S单节点架构，可以满足当前的需求，传统的外观专利图像检索系统的架构如图3-1所示。

传统B/S架构的外观设计专利图像检索系统采用三层架构开发，分别是

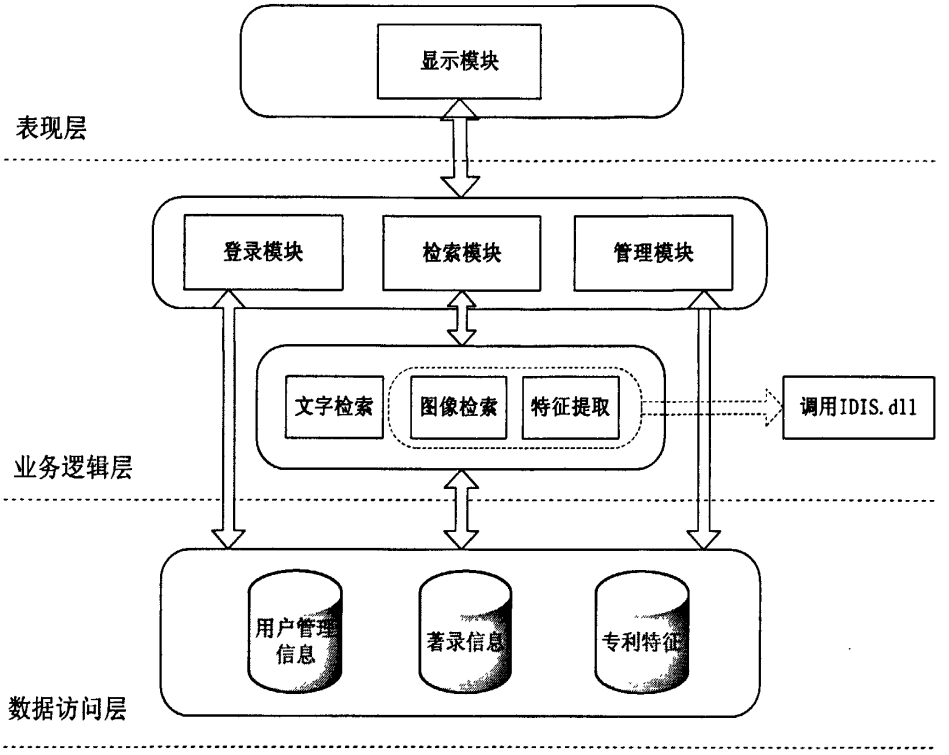

图3-1 传统B/S架构的外观设计专利图像检索系统

Fig.3-1 Traditional B/S framework of design patents image retrieval system

表示层、业务逻辑层以及数据链路层[48]：

(1) 表示层：主要是通过Web浏览器来实现，用户通过浏览器的交互页面与系统进行交互，最终将结果在浏览器上呈现给用户。

(2) 业务逻辑层：主要负责逻辑业务的处理，主要包括文字检索、图像检索和特征提取模块。在业务逻辑层，根据用户的请求类别进行相应的业务处理。

文字检索模块主要实现专利的专利号、名称、发明人等信息的精确及模糊检索。图像检索模块主要实现用户将待检索的图片上传至服务器，并选择查询的相关关键字、形状、纹理或颜色查询方式以及查询结果的数量后进行图像检索。服务器通过提取待检索图片的特征与特征库中的特征进行匹配，按相似度的大小得到图像检索结果。最终将视图信息以及专利的著录信息返回给用户。特征提取模块是调用实现的图像处理动态库。

(3) 数据链路层：数据链路层存储图像检索所需的特征库、外观专利的著录信息以及专利图片，主要用于处理业务逻辑层与数据库的交互。

3.2.2 传统图像检索系统架构的不足及解决方法

对于当前数量的外观设计专利，传统架构的外观设计专利图像检索系统，在少量用户并发访问时，可以及时的响应用户的请求。但由于图像检索的过程主要是待检索图像的特征与特征库中特征的相似度计算过程，该过程是一个数据密集型计算过程，需要消耗大量的CPU资源，而计算机的CPU资源是相当有限的，计算机资源的相对数量如图3-2所示。

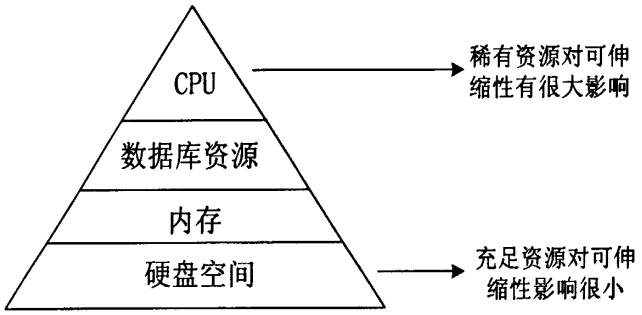

图3-2 计算机资源的相对数量

Fig.3-2 Quantities of computing resources

随着外观专利数量的逐年快速增加(近七年我国外观设计专利的申请数量如表3-1所示)，系统的检索时间在多用户并发时将急剧下降。如果在大量操作同时并发时，系统可能无法响应，此时需要对系统进行扩展。

表3-1 近七年我国外观设计专利的申请数量

Table 3-1 Quantities of application of design patents in last seven years

年份(年)	2004	2005	2006	2007	2008	2009	2010
申请数(万件)	8.9	11	13.9	17.4	19	22.3	26

系统的伸缩性扩展主要是通过以下两个方面：

(1) 垂直伸缩(按比例增加)：即使用新的、快速的硬件取代慢速的硬件。然而，垂直伸缩相对昂贵。

(2) 水平伸缩(按比例增加)：为现有的应用程序添加额外的、负载均衡的服务器，以实现水平伸缩。这种伸缩可以保护当前的硬件资源，如果其中

有部分服务器无法正常工作，应用程序还是可以正常运行，水平伸缩相对便宜。

垂直伸缩在一定的规模下对系统性能的提高非常明显，但当规模增大到一定的程度时，会到达性能瓶颈，如图3-3所示。主要是由于添加额外的硬件时，应用程序必要的稳定性开始影响其吞吐量(单位时间处理的信息量)，而水平伸缩扩展则不会出现这种瓶颈。

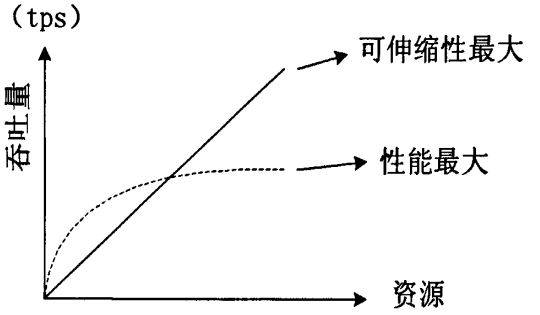

图3-3 垂直伸缩模式下资源与性能的关系

Fig.3-3 Relation of resource and performance in the mode of vertical scaling

要实现系统的水平伸缩扩展，可通过分布式处理来实现。分布式处理作为一种大规模及海量数据处理的解决方案，就是让网络上的多台服务器作为一个单独的系统协同完成一些大规模的计算以及数据的存储。从而解决单服务器模式的性能瓶颈，给客户提供更好的访问质量，提高服务器响应速度，提高服务器及其他资源的利用率，避免了网络关键部位出现单点失效[49]。

目前，分布式处理技术的研究前沿主要是网格计算和云计算(Hadoop框架在很多云计算构架中得到了广泛应用)，但网格计算和云计算的实际应用案例比较少。传统主流的分布式系统有DCOM、Java EJB、Java RMI、Microsoft .NET Web Service[50]以及Microsoft .NET Remoting[51]。由于Hadoop的可扩展性、可靠性、高效性和集群搭建的廉价性，以及MapReduce并行程序开发的简便性，本文通过Hadoop平台来解决图像检索系统的实时性问题。

3.3 基于 Hadoop 的外观专利图像检索系统的设计

3.3.1 分布式系统整体框架

通过对传统B/S单节点架构的外观专利图像检索系统的分析，其存在检索实时性、系统的扩展性以及服务质量的不稳定性等问题。

对于检索的实时性，通过Hadoop平台的MapReduce框架，实现检索过程中图像匹配计算的分布式计算，将检索任务分解为小的粒度[62]，执行并行计算，从而极大地提高图像检索的速度。由于Hadoop集群添加节点的便捷性，对于系统的扩展将非常方便。对于服务质量的稳定性，传统架构中，如果一个检索请求发生错误，那么此次检索将以失败而结束。采用Hadoop平台后，由于Hadoop中的任务是采用“推测式执行”方式[63]，此时将大幅度提高检索服务质量的稳定性。

针对以上情况，设计出了基于Hadoop的外观设计专利图像检索系统，该系统的整体框架如图3-4所示。

该系统的整体构架可以分为三个层次，即表现层、业务逻辑层以及数据及数据处理层，各层的主要作用如下：

表现层：用户在客户端提交本地示例图像或文字检索关键字，通过Internet将检索信息传送至Web服务器，以及将Web服务器的返回结果呈现给用户。

业务逻辑层：根据用户检索类别(图像检索或文字检索)执行相应业务处理。

数据及数据处理层：对于图像检索，示例图像提交给Hadoop分布式系统，提取图像特征，通过实现的MapReduce程序，将示例图像特征与HDFS中的专利特征库进行匹配，得到相似度计算结果，将HDFS中存储的专利图像通过Internet返回给用户；对于文字检索，将文字检索信息与HDFS中存储的著录信息库进行匹配，得到匹配结果，将HDFS中存储的专利图像通过Internet返回给用户。

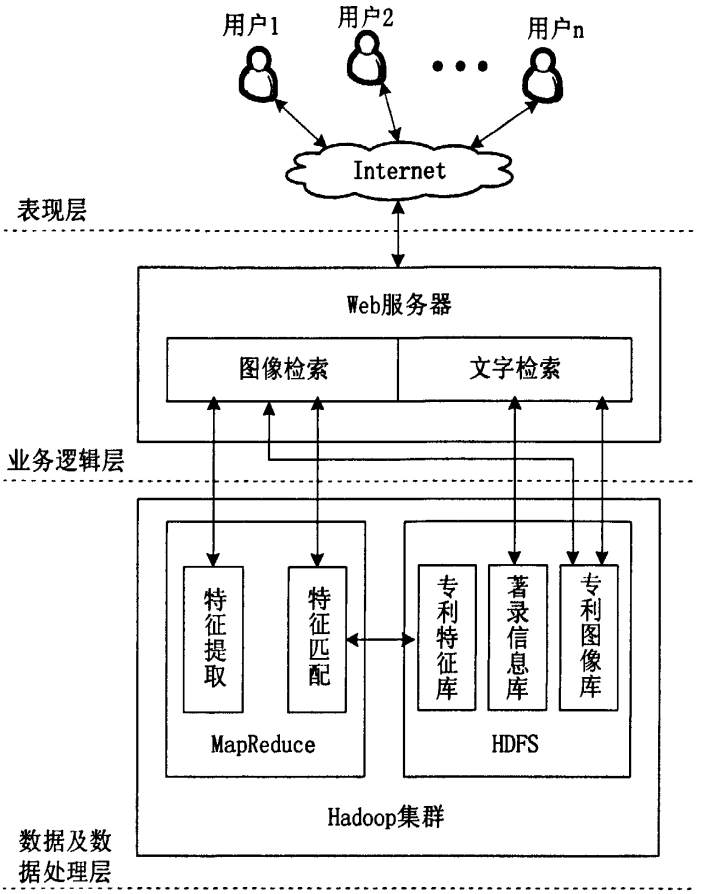

图3-4 基于Hadoop的外观设计专利图像检索系统的整体构架

Fig.3-4 Overall framework of design patents image retrieval system based on Hadoop

基于Hadoop的外观设计专利图像检索系统的核心部分是数据及数据处理层的图像检索过程(MapReduce模块)以及专利信息的存储(HDFS模块)，表现层以及业务逻辑层可以采用传统的设计方法。下面分别对HDFS模块以及MapReduce模块的设计进行阐述。

3.3.2 HDFS 模块设计

HDFS是一个针对大文件进行优化的分布式文件系统，HDFS采用Master/Slave架构，由主节点NameNode和数据节点DataNode组成。HDFS为检索系统提供外观专利图像库、著录信息库以及特征库的存储，为MapReduce提供相关信息的交互以及处理所需的数据。

HDFS的性能是通过Hadoop集群中NameNode节点以及各DataNode节点

的整体性能来体现的。HDFS的存储能力是各DataNode节点存储能力之和，HDFS的管理性能主要体现在NameNode节点。

对于原始的外观专利数据，需要对其进行相应的预处理后才可以存储于HDFS中。由于基于内容的外观设计专利图像检索，是根据专利图像的形状、纹理以及颜色等视觉特征对专利图像进行描述，结合外观专利的著录信息、各视图图像的特征信息，实现图像检索。因此，需要对著录信息以及专利图像做以下几个步骤的处理：

(1) 将外观设计专利图像库存储于HDFS中，用于专利图像的特征提取以及用户获取图像检索结果。

(2) 原始的著录信息是文本格式，需要对著录信息中专利的各关键信息进行提取，如专利的“申请号”、“申请人”、“发明人”、“发明日期”以及“地址”等信息，将提取后的信息存储于HDFS中。

(3) 对外观设计专利图像库图像进行特征提取，提交MapReduce任务，Map阶段，map函数每次读入一幅图像，提取其形状、纹理以及颜色特征。Reduce阶段，将Map阶段提取的图像特征数据存储于HDFS中。

(4) MapReduce框架可以处理多种数据格式，从一般的文本文件到整个数据库文件。为便于检索中Map任务的执行，将每个图像的特征作为一条特征记录，提取的特征数据以文本格式存储于HDFS中。

HDFS中存储的外观专利图像检索相关数据，可以设置数据的备份数(可以通过配置Hadoop集群中hdfs-sites.xml文件的dfs.replication属性进行修改)，数据默认的备份数为3份，可以根据Hadoop集群的规模以及系统需求设置更合理的备份数，备份的数据块一般都是随机存储于Hadoop集群中其它的DataNode节点上。HDFS中的备份机制可以提高系统的吞吐量以及数据的完整性，并为MapReduce框架中任务的并行执行提供了数据的支撑[54]。

外观专利信息存储于HDFS中是以数据块的形式存储，HDFS系统中默认的数据块大小为64M(可以通过配置Hadoop集群中hdfs-sites.xml文件的dfs.block.size属性进行修改)，这为MapReduce中Map任务的输入提供了数据分片的基础。

3.3.3 MapReduce 模块设计

MapReduce模块实现的主要功能是图像检索过程中的图像匹配计算，也即相似度的计算。MapReduce任务是由客户端提交一个MapReduce作业，由JobTracker进行相应的初始化以及调度，将MapReduce程序分发到各TaskTracker节点进行运行。MapReduce程序在运行时需要读取存储于HDFS中相关的图像特征数据。MapReduce模块实现外观设计专利图像匹配的工作框图如图3-5所示。

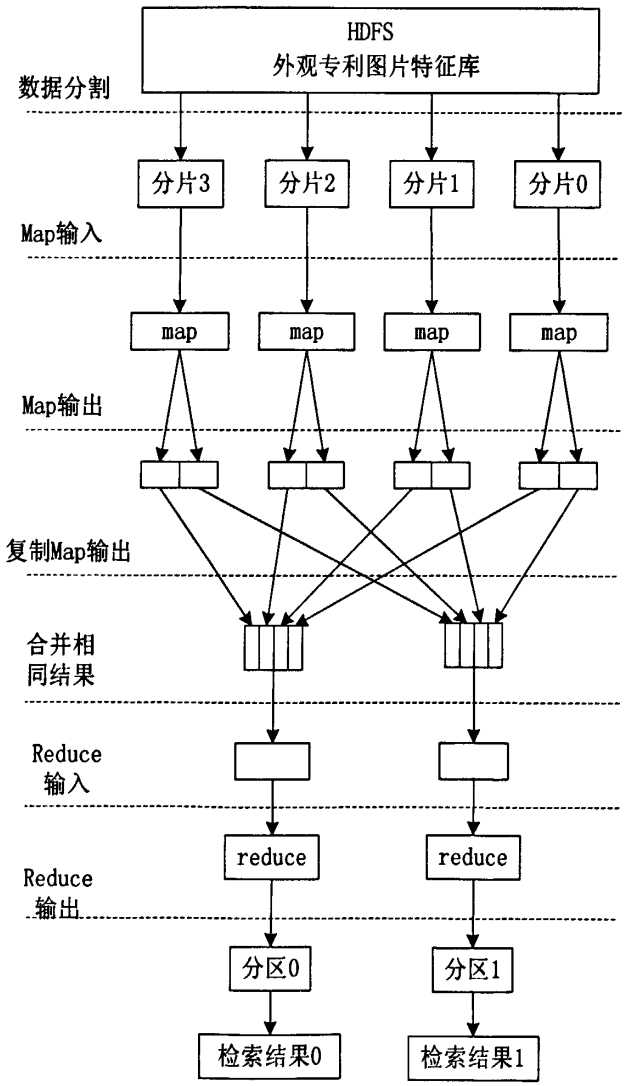

图3-5 实现并行计算的MapReduce工作框图

Fig.3-5 Framework of parallel computing with MapReduce

MapReduce的工作流程可以分为以下几个方面：

(1) MapReduce程序在运行时，将HDFS中存储的外观设计专利图像特征库进行分割，得到图像特征数据的分片(外观设计专利图像的特征包括形状、纹理、颜色等特征，一幅图像的特征数据以文本形式的一行数据表示)。

(2) 对于每个数据分片，将其由DataNode节点传送至各TaskTracker节点，理想状况下是“本地计算的”[34](即DataNode节点与TaskTracker节点处于同一个节点)。各Map任务读取数据分片，将数据分片进行分解为一条一条的图像特征数据，以键/值对的形式生产，以外观专利图像特征在分片中的偏移距离为Key值，外观专利图像的特征值为Value值。然后将这些Key/Value对传递给map( )函数进行相似度的匹配，匹配结果以相似度和特征库中的图像名作为Key/Value对，最终得到中间结果。

(3) 对于Map任务输出的中间结果(Key/Value对)，将其进行相同结果合并(如果存在相同记录的数据，将只取一个结果)以及排序(将单个Map任务的中间结果以Key值进行排序，以提高Reduce任务的执行效率)，将最终的中间结果传递给Reduce任务。

(4) Reduce任务读取Map任务输出的中间结果，对收集的中间结果进行汇总、按相似度的大小进行排序，最终得到图像检索的结果，并将结果写入HDFS中。

3.3.4 Hadoop系统中图像检索过程

在以上系统设计的基础上，外观专利图像检索任务在Hadoop框架下的执行可分为以下几个步骤：

(1) 用户提交检索请求：由JobClient向JobTracker请求一个新的图像检索作业ID，JobTracker检查作业的相关信息，计算作业图像特征库的输入划分，将运行图像检索作业所需要的资源(作业JAR文件、配置文件以及计算所得的特征库的划分)复制到以作业ID命名的JobTracker的文件系统当中，通知JobTracker作业准备执行。

(2) 检索请求的初始化和分配：JobTracker接到作业任务，将作业放入一个内部队列中，交由作业调度器进行调度，并对该作业进行初始化。

JobTracker将作业分配到集群的多个TaskTracker，TaskTracker节点上存储着HDFS中的部分特征数据。一个Map任务，它的输入数据一般都是本地数据，可以减少网络传输所带来的带宽消耗，从而减小了系统的阻塞。

(3) 检索请求的Map阶段：有任务的TaskTracker将JAR文件(实现图像检索的MapReduce程序)本地化。同时，将所需特征数据从HDFS复制到本地的文件系统中，建立TaskRunner实例运行任务。MapReduce对HDFS中外观设计专利图像的特征库进行分割，将数据块分发到集群各工作节点。各工作节点执行Map任务，处理特征库中每条特征记录，根据检索条件和检索方式计算各图像特征值与示例图像的相似度，以相似度和特征库中的图像名作为Key/Value对(中间结果)，将中间结果输出到本地文件系统中。

(4) 检索请求的Reduce阶段：MapReduce框架对中间结果进行分区，将Key值相同的结果合并，将其传递给Reduce任务(Reduce任务数根据需要可以设置多个，但不超过Map的任务数)。Reduce任务接收Map输出的键值对，对Map的输出进行汇总、排序，得到图像检索的结果，最终将结果写入HDFS。

(5) 检索请求的完成：当JobTracker收到作业完成信息，JobTracker将任务标识为成功，通知用户。最后，JobTracker清空作业的工作状态，TaskTracker清空相关中间输出，用户得到最终的检索结果。

3.4 本章小结

本章首先对外观专利图像检索系统的需求进行了介绍，然后对传统B/S架构下的外观专利图像检索系统进行了介绍，分析了该架构下系统的不足以及存在问题的解决思路。通过以上的分析，提出了基于Hadoop的外观设计专利图像检索系统，并对其整体框架以及核心模块(HDFS模块和MapReduce模块)的设计进行了详细介绍，并给出了图像检索任务在Hadoop框架下的执行过程。

第四章 分布式图像检索算法实现

该系统的核心是分布式图像检索算法的实现，主要是基于内容的图像检索算法在 MapReduce 编程模型中的实现。本章详细介绍了 MapReduce 程序中 Map 函数、Reduce 函数以及 Run 函数的实现，并在 Windows 环境下的伪分布式系统中进行开发和测试。

4.1 MapReduce 并行编程实现准备工作

根据MapReduce程序的工作原理和提出的外观专利图像检索系统，要实现基于MapReduce的外观专利图像检索算法，要完成以下三方面的工作[34]。

(1) map( )函数的实现。

(2) reduce( )函数的实现。

(3) 作业运行的实现。

对于map函数，它是由Mapper接口来实现的，该接口有一个map( )方法。因此，Mapper相当于一个函数对象，通过其处理每一对<Key,Value>，最终将中间结果写入到context中，Mapper接口如下所示。

```
public interface Mapper<KeyIn K1, ValueIn V1, KeyOut K2, ValueOut V2>
extends JobConfigurable, Closeable
{
    void map(K1 key, V1 value, Context context)  throws IOException;
}
```

Mapper接口有4个形式参数，它们指定map函数的输入键K1、输入值V1、输出键K2以及输出值V2。

map( )函数需要输入一个键值对key/value，map( )方法对传入的值value进行处理。对于处理的结果(中间结果)，将其交由map( )的另一个参数context，该实例参数收集map函数的输出键/值对。Context记录着map( )函数运行的一些统计信息，这些统计信息，通过“心跳”传递给JobTracker。通过各

TaskTracker节点的统计信息，从而使JobTracker了解作业的整体运行状况。

map( )函数处理的流程可以表示为：map( )函数从InputSplit中读取输入流，InputSplit由InputFormat生成。map( )函数读取输入流中的键值对Key/Value，并对Value进行业务处理。对于处理的结果，将被框架有序化的排序后写入context中，存储于TaskTracker节点的本地文件系统中。

如果中间结果需要归并处理，可以使用combiner，combiner的形式与reduce函数一样，区别在于combiner的输入键/值对是中间的键/值对类型，combiner的输出将交由reduce函数进行处理。

对于reduce函数，它是由Reducer接口来实现的，该接口有一个reduce( )方法。Mapper接口如下所示。

```
public interface Reducer<KeyIn K2,ValueIn V2,KeyOut K3,ValueOut V3>
extends JobConfigurable, Closeable
{
    void reduce(K2 key, Iterator<V2> values, Context context) throws
IOException;
}
```

Reducer的输入也是键/值对，但区别在于其值是一个迭代器Iterator。

MapReduce框架是基于Java实现的，但对于map函数与reduce函数的输入和输出类型，它们不是使用内置的Java类型，而是经过重新封装的，这样可以提高网络数据的传输以及写入持久存储过程的效率。

Hadoop使用自己的序列化格式Writable类，对于MapReduce处理的数据类型(即KeyIn、ValueIn、KeyOut以及ValueOut的类型)都是扩展于Writable类，Writable类的层次结构如图4-1所示。

实现map函数与reduce函数后，要实现MapReduce作业，需要设置相关的运行参数，通过Run( )函数来实现。Run( )函数可以表示如下所示。

```
Run( String[] args ):
Configuration conf = new Configuration();
Job job = new Job(conf, "test");  //创建作业，设置作业名“test”
job.setJarByClass(test.class);  //设置作业的类文件名
job.setMapperClass(testMap.class);  //设置Mapper的类文件名
```

```
job.setReducerClass(testReduce.class);  //设置Reducer的类文件名
job.setCombinerClass(testReduce.class);  //设置Combiner的类文件名
job.setInputKeyClass(IK.class); //设置Map和Reduce输入键的处理类型
job.setInputValueClass(IV.class); //设置Map和Reduce输入值的处理类型
job.setOutputKeyClass(OK.class); //设置Map和Reduce输出键的处理类型
job.setOutputValueClass(OV.class);  //设置Map和Reduce输出值的处理类型
FileInputFormat.addInputPath(job, new Path(args[0]));  //设置作业的数据输入路径
FileOutputFormat.setOutputPath(job, new Path(args[1]));  //设置作业的数据输出路径
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

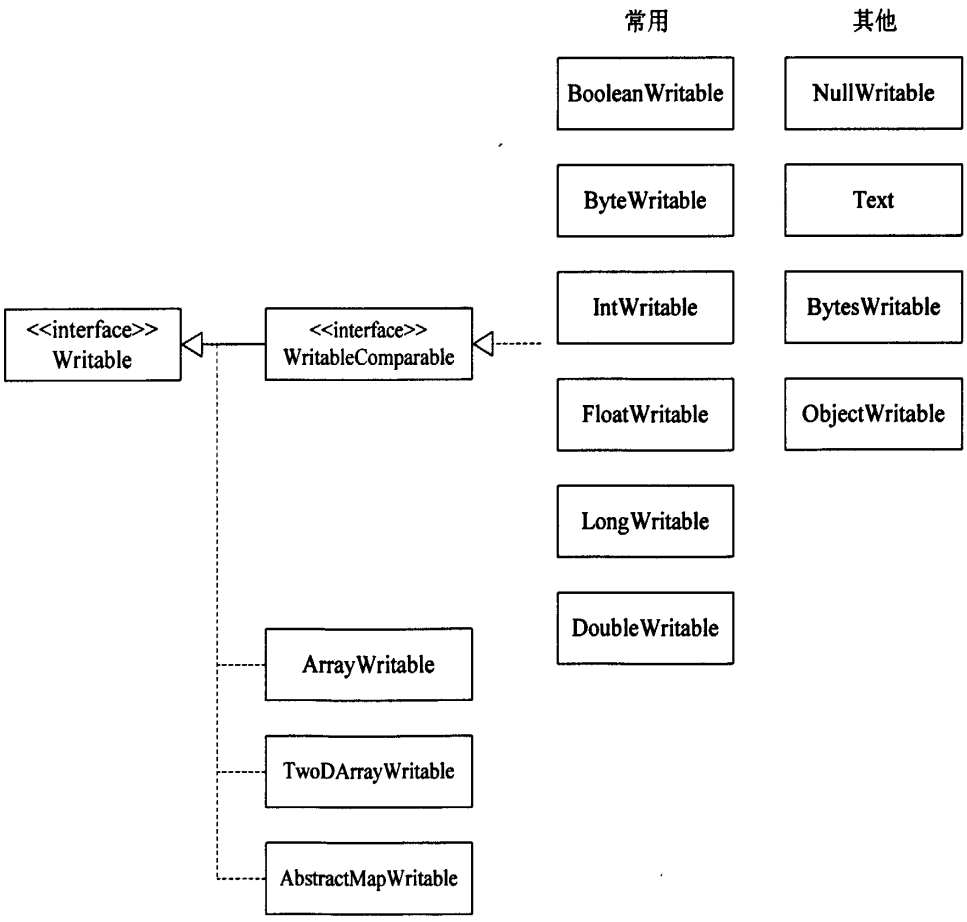

图4-1 Writable类的层次结构

Fig.4-1 Structure of class Writable

一个完整的MapReduce程序，需要实现Mapper接口中的map( )方法、Reducer接口中的reduce( )方法以及run( )函数，在以上的基础上，就可以将MapReduce作业提交给Hadoop系统进行运行。

4.2 MapReduce 程序实现

实现一个基于MapReduce的并行计算，关键是map函数和reduce函数的实现。数据预处理中，图像的特征库以文本形式存储于HDFS当中，Map任务分配的数据为文本格式，采用TextInputFormat[34]格式来处理输入数据。Map函数输入的键值对中的键(key)表示一条特征在整个文件中的偏移量(以字节为单位)，值(value)是一幅图像的特征。

将特征库中的数据分割，分配给各Map(一般均衡分布在集群节点上)任务处理，利用集群各节点资源实现并行计算。实现图像匹配计算的map函数和reduce函数可用如图4-2所示的流程图表示。

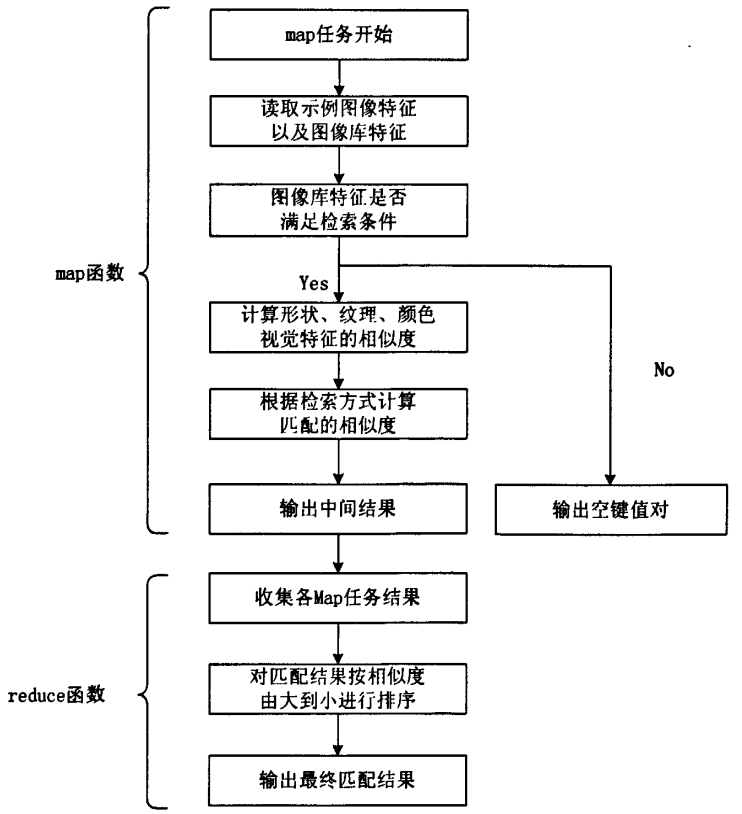

图4-2 MapReduce 算法流程图

Fig.4-2 Process of MapReduce

图4-2中MapReduce算法的实现对应MapReduce整体运行框架中的Map模块和Reduce模块。Map任务执行时，由HDFS中读取相关特征数据，map函数对输入数据进行判断，满足检索条件将进行相似度计算，将结果写入本地文件系统。Reduce任务执行时，执行Reduce任务的节点收集各Map任务的中间结果，对其按相似度由大到小排序，将检索结果存储于HDFS中。Map任务和Reduce任务的执行过程中，JobTracker进行相关的任务调度，协调各节点以及主节点之间的信息交互。

4.2.1 Map 函数实现

Map函数实现的主要功能是：将HDFS中的外观专利图像特征库的数据进行读取至各Map任务(特征库中数据的分割在提交MapReduce作业时完成)，将待检索的图像特征与图像特征库中的数据进行相似度匹配计算，得到它们之间的相似度。存储于HDFS的外观专利图像特征库，经过了预处理(3.3.2节所示)。为方便map函数的处理，特征库数据是以文本格式存储，以一幅图像提取的特征做为一条记录(文本中的一行数据)。因此，map函数的各参数可以表示为：

map(Object key,Text value,Context context)

对于map函数从各分割的数据块中读取数据，以特征库中特征数据的偏移量为Key值，一条特征的值为Value值。对于Key值，此时只是做为一条特征记录的标示，在map函数内不做处理。对于Value值，就是需要处理的数据，它是以文本格式读取的，Hadoop不是使用内置的String类型。因此，此时将其类型设置为Text类型。在map函数里，读取的图像特征不能直接使用，需要将其由Text类型转化为String类型。

对于图像的特征，将特征数据存储于一个CharactSet类中，以便处理。图像检索的过程中涉及到形状、纹理以及颜色等特征，对于不同的检索方式(形状、纹理以及颜色的组合，三个分量的不同权值)，将得到不同的相似度，通过CompareManner类来实现，该类提供了不同检索方式的相似度计算方法。最后将计算结果写入到上下文中，至此，Map函数完成其任务。

Map函数的实现可以如下所示。

```
/*
 * @key-in：Map的输入key，为Object类型(特征表中记录的行偏移量)
 * @value-in：Map的输入value，为Text类型(特征表中的一行记录)
 * @key-out：Map的输出key，为DoubleWritable类型(检索的相似度)
 * @value-out：Map的输出value，为Text类型(专利图片的专利号)
 *
 *  功能：根据特征表中的每个专利图片的特征值，与待检索的专利图片的特征值进行匹配，得到两者的相似度。
 *  将相似度以及专利的专利号输出给Reduce函数。
 */
public class ImageSearchMapper extends Mapper<Object, Text, DoubleWritable, Text>
{
  public void map(Object key,Text value,Context context) throws IOException,InterruptedException
  {
    //进行两条记录进行匹配,C0为待检索图片的特征，C为数据库中的特征
    String charactOrigin = value.toString();
    CharactSet C = new CharactSet(charactOrigin);
    //读取待检索图片的特征
    String charactSearch = ReadSearchCharact();
    CharactSet C0 = new CharactSet(charactSearch);

    //设置检索方式
    int searchManner = ReadSearchManner();
    CompareManner cManner = new CompareManner();
    //各种检索方式的计算结果
    float resultByShape = 0;
    float resultByTexture = 0;
```

```
float resultByColor = 0;
float result = 0;

//searchManner: 检索方式(0：形状＋纹理  1：形状  2：纹理  3：颜色)
if(searchManner = = 0 || searchManner = = 1 || searchManner = = 2 || searchManner = = 3)
{
    resultByShape = cManner.CompareByShape(C0, C);
}
   //计算纹理特征的相似度
if(searchManner = = 0 || searchManner = = 2)
{
    resultByTexture = cManner.CompateByTexture(C0, C);
}
   //计算颜色特征的相似度
if(searchManner = = 3)
{
    resultByColor = cManner.CompareByColor(C0, C);
}

//根据检索方式得到检索结果，不同方式各分量的权值不同
//按形状+纹理模式
if(searchManner = = 0)
{
    result = (float) (resultByShape * 0.85 + resultByTexture * 0.15);
}
//按形状模式
if(searchManner = = 1)
{
```

```
            result = resultByShape;
        }
        //按纹理模式
        if(searchManner = = 2)
        {
            result = (float) (resultByShape * 0.6 + resultByTexture * 0.4);
        }
        //按颜色模式
        if(searchManner = = 3)
        {
            result = (float) (resultByShape * 0.6 + resultByColor * 0.4);
        }

        //获取专利号
        String filenameString = C.getFilename();
        Text fileName = new Text(filenameString);
        //转换结果为DoubleWritable
        DoubleWritable compResult = new DoubleWritable((double)result);
        //将结果写入上下文
        context.write(compResult, fileName);
    }
}
```

4.2.2 Reduce 函数实现

Reduce函数的主要功能是收集各Map任务的计算结果，即将各TaskTracker节点上执行的Map任务输出的中间结果收集，并按相似度的大小，由大到小进行排序，Reduce函数可以表示为：

reduce(DoubleWritable key,Text value,Context context)

Reduce函数的Key值为DoubleWritable型，即map函数计算的相似度，

Value值为Text型，为特征库中图像的专利号，通过reduce函数排序后，将计算结果写入context，最终将其存于HDFS中。

由于MapReduce排序的结果是按照由小到大进行排序，因此，需要对DoubleWritable的比较方式进行重写，即按相似度由大到小排序，使Reduce函数的输出按Key值降序排序，其实现如下所示。

```
public class DoubleWritableDecreasingComparator extends DoubleWritable.Comparator
{
  public int compare(DoubleWritable a,DoubleWritable b)
  {
     return -super.compare(a, b);
  }

  public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)
  {
      return -super.compare(b1, s1, l1, b2, s2, l2);
  }
}
```

对于Reduce函数，其实现如下所示：

```
/*
 * @key-in：Reduce的输入key，为DoubleWritable类型(检索的相似度)
 * @value-in：Reduce的输入value，为Text类型(专利图片的专利号)
 * @key-out：Reduce的输出key，为DoubleWritable类型(检索的相似度)
 * @value-out：Reduce的输出value，为Text类型(专利图片的专利号)
 *
 *  功能：实现由Map得到的匹配结果进行按相似度由大到小排序输出。
 */
public class ImageSearchReduce extends Reducer<DoubleWritable, Text, DoubleWritable, Text>
{
```

```
    public void reduce(DoubleWritable key,Text value,Context context)
throws IOException,InterruptedException
    {
        //汇总Map结果，直接写入目的文件夹
        context.write(key,value);
    }
}
```

4.2.3 Run 函数实现

在实现了map函数以及reduce函数后，需要设置MapReduce作业的相关信息，以便MapReduce作业可以正常运行，这包括作业的作业名、JAR包、Mapper和Reducer类的设置、输入输出键值对类型的设置、输入输出数据的路径设置等。

Run函数的实现可以表示为：

```
Run(String[] paths)
{
  Configuration conf = new Configuration();
  //创建作业，设置作业名“ImageSearch”
  Job job = new Job(conf, " ImageSearch");
  //设置作业的类文件名
  job.setJarByClass(ImageSearch.class);
  //设置Mapper的类文件名
  job.setMapperClass(ImageSearch Mapper.class);
  //设置Reducer的类文件名
  job.setReducerClass(ImageSearch Reduce.class);
  //设置Combiner的类文件名
  job.setCombinerClass(ImageSearch Reduce.class);

  //设置Map函数输入键的处理类型
```

```
        job.setInputKeyClass(Object.class);
        //设置Map函数输入值的处理类型
        job.setInputValueClass(Text.class);
        //设置Map和Reduce输出键的处理类型
        job.setOutputKeyClass(DoubleWritable.class);
        //设置Map和Reduce输出值的处理类型
        job.setOutputValueClass(Text.class);
        //设置排序规则，由大到小排序
        job.setSortComparatorClass(DoubleWritableDecreasingComparator.cla-
        ss);
        //设置作业的数据输入路径
        FileInputFormat.addInputPath(job, new Path(args[0]));
        //设置作业的数据输出路径
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

4.3 Windows 环境下的测试

4.3.1 开发环境搭建

设计开发阶段，可在Windows环境下进行MapReduce程序的开发以及测试。在此之前，需要搭建开发环境。MapReduce程序的开发是在Eclipse上进行的，由于在测试时，需要访问分布式文件系统(HDFS)，此外，需要分布式计算框架。因此，需要在Windows下提供Hadoop平台的访问。

基于以上的需求，在Windows上搭建Hadoop平台可以分为以下几个步骤[56]：

(1) 安装JDK，因为Hadoop的实现是基于java的，需要安装JDK以及JRE环境，以提供运行环境(Hadoop要求JDK的版本为1.6以上)。

(2) 安装Cygwin[66]，Cygwin是一个在Windows平台上运行的Linux模拟

环境，提供类似Linux环境的操作。在安装的过程中，需要安装OpenSSL[67](以提供无密码访问)、Base Category下的sed(提供在Eclipse上进行Hadoop的编译)、Editors Category下的sed(提供Vim编辑)等包。

(3) 配置JDK以及Cygwin的环境变量。

(4) 安装sshd服务以及启动sshd服务，配置ssh登录，以实现各节点之间无密码登录。

(5) 安装Hadoop，本文实验使用的Hadoop版本为0.20.0，在Hadoop的安装过程中需要修改一些配置文件。这些配置文件位于conf目录下，分别是hadoop-env.sh、core-site.xml、hdfs-site.xml和mapred-site.xml共四个文件。对于Windows环境下的Hadoop平台的搭建，masters和slaves不用修改。

(6) 至此完成Hadoop的安装。

启动Hadoop：在启动Hadoop之前，需要格式化名称节点，否则Hadoop无法正常启动，首先执行格式化命令(./hadoop namenode –format)。然后运行./start-all.sh启动Hadoop，如图4-3所示。可以通过执行jps命令查看Hadoop是否正常启动，正常启动的话，会显示NameNode以及JobTracker进程正常启动。

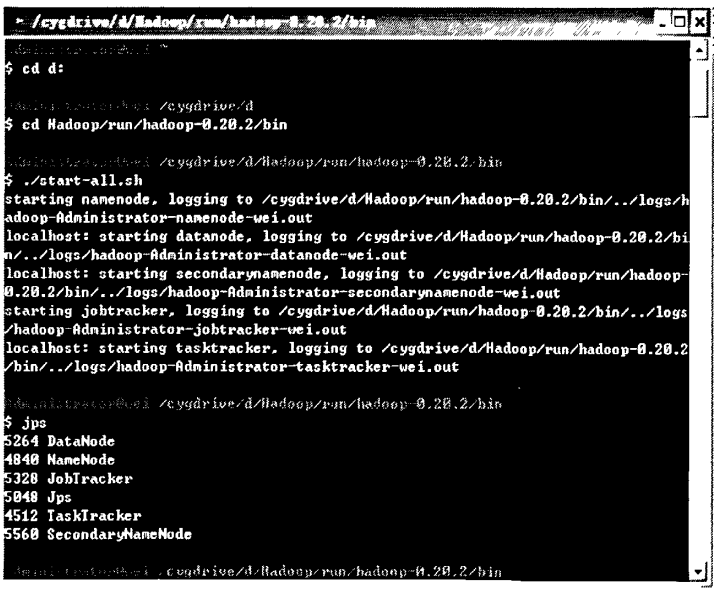

图4-3 启动Hadoop

Fig.4-3 Hadoop booting

在Hadoop运行之后，可以通过web端口了解分布式文件系统的状况，通过http://localhost:50070端口查看，如图4-4所示。通过该界面，可以查

看Hadoop系统的一些信息，包括系统的存储能力以及所剩空间，分布式集群中存活的节点以及失败的节点(由于是在Windows环境下搭建的伪分布式系统，所以存活的节点数为1)。此外，可以查看HDFS的文件目录(通过Browse the filesystem连接查看)。

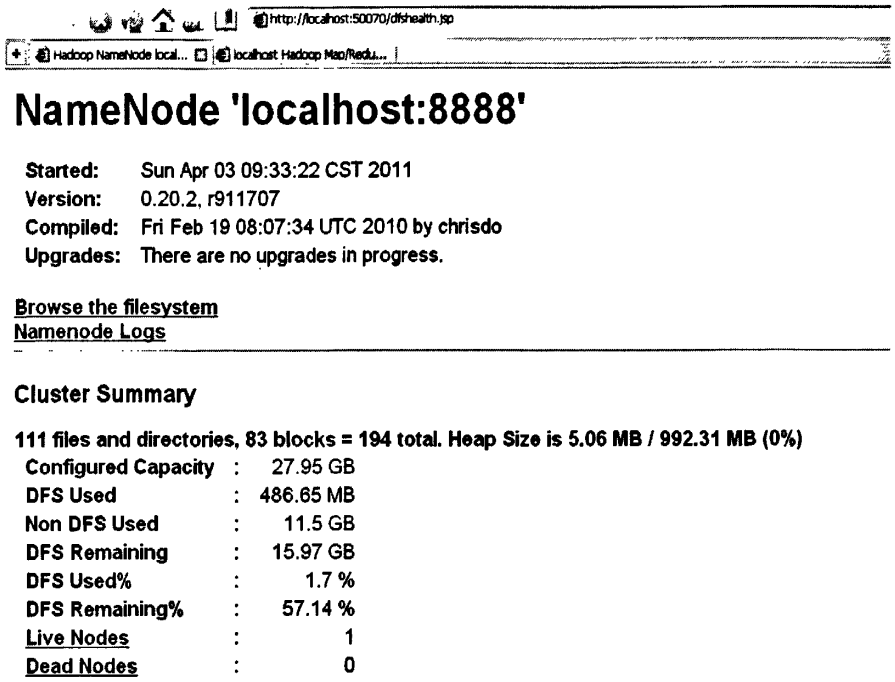

图4-4 Hadoop的分布式文件系统状况

Fig.4-4 Status of HDFS

对于Hadoop集群的任务执行情况，可以通过http://localhost:50030进行查看集群任务的执行情况。通过该界面，可以了解到Hadoop系统中MapReduce作业的执行情况，Map任务、Reduce任务、计算节点数量、Map任务的槽数、Reduce任务的槽数，以及具体MapReduce任务执行的相关信息。

为了方便MapReduce程序的开发，Hadoop提供了Eclipse插件，通过该插件，可以让我们像开发普通的Java程序一样方便，根据map函数、reduce函数以及Run函数的设计实现，在搭建的开发环境上的实现如图4-5所示。

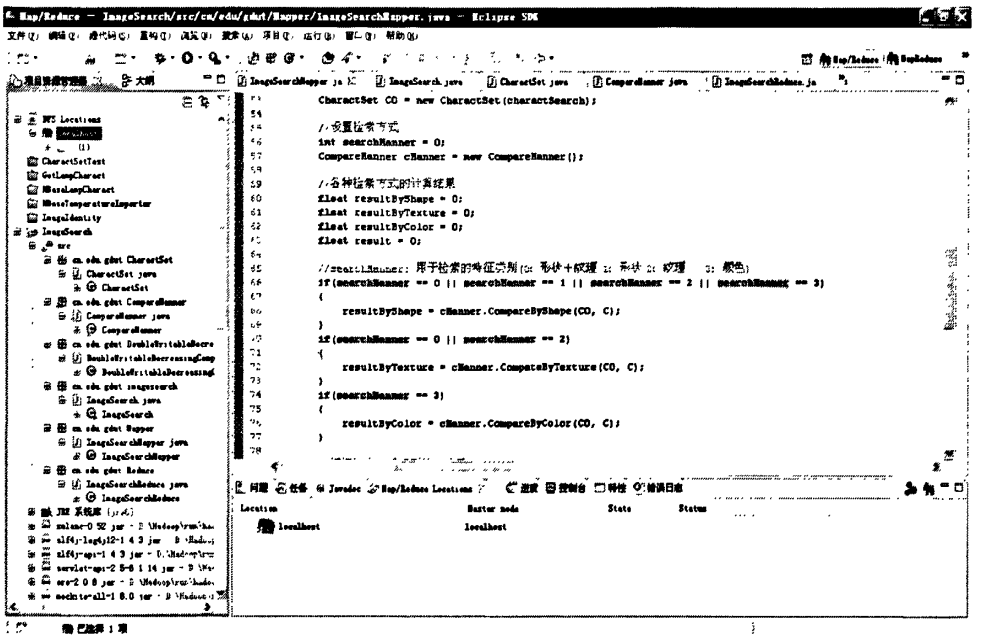

图4-5 MapReduce程序开发环境

Fig.4-5 Development environment of MapReduce

4.3.2 MapReduce 程序测试

以外观设计专利的灯具类别作为测试类别，实验的数据量为208087条特征记录，测试按如下步骤进行：

(1) 通过Cygwin启动Hadoop，以便在Eclipse中通过Hadoop插件直接访问Hadoop伪分布式文件系统。

(2) 通过DFS在HDFS中新建图像特征数据的存储路径ImageSearch-DataInput，将专利图像的特征数据上传至HDFS中的ImageSearchDataInput文件夹。

(3) 配置图像检索相似度计算的数据输入、输出路径，对于Hadoop的伪分布式文件系统，数据输入路径为：hdfs://localhost:8888/ImageSearch-DataInput，数据输出路径为：hdfs://localhost:8888/ImageSearchDataOutput。

(4) 完成以上工作后，开始运行ImageSearch程序，就像一般的Java应用程序开发一样(在Eclipse上运行ImageSearch程序时，实际上是向Windows环境下的Hadoop伪分布式文件系统提交了一个图像检索的作业，作业的运行过程与真正的Hadoop分布式文件系统处理过程是一致的)。对

于作业的运行情况，可以通过Eclipse的控制台窗口进行查看，执行过程的相关信息如图4-6 a)、b)所示。通过控制台输出的信息，可以详细的了解到ImageSearch的执行情况，如Map和Reduce任务的任务数、完成情况、计算结果的输出、处理的数据大小以及Map和Reduce任务的输入和输出的记录数。通过ImageSearch的执行过程，可以知道执行208087条特征数据检索的时间为11秒。

```
11/04/03 09:54:54 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
11/04/03 09:54:54 WARN mapred.JobClient: No job jar file set.  User classes may not be found. See JobConf(Class) or J
11/04/03 09:54:55 INFO input.FileInputFormat: Total input paths to process : 1
11/04/03 09:54:55 INFO mapred.JobClient: Running job: job_local_0001
11/04/03 09:54:55 INFO input.FileInputFormat: Total input paths to process : 1
11/04/03 09:54:55 INFO mapred.MapTask: io.sort.mb = 100
11/04/03 09:54:55 INFO mapred.MapTask: data buffer = 79691776/99614720
11/04/03 09:54:55 INFO mapred.MapTask: record buffer = 262144/327680
11/04/03 09:54:56 INFO mapred.JobClient:  map 0% reduce 0%
11/04/03 09:55:01 INFO mapred.LocalJobRunner:
11/04/03 09:55:02 INFO mapred.JobClient:  map 35% reduce 0%
11/04/03 09:55:04 INFO mapred.LocalJobRunner:
11/04/03 09:55:05 INFO mapred.JobClient:  map 48% reduce 0%
11/04/03 09:55:07 INFO mapred.LocalJobRunner:
11/04/03 09:55:08 INFO mapred.JobClient:  map 65% reduce 0%
11/04/03 09:55:10 INFO mapred.LocalJobRunner:
11/04/03 09:55:11 INFO mapred.JobClient:  map 85% reduce 0%
11/04/03 09:55:13 INFO mapred.MapTask: Starting flush of map output
11/04/03 09:55:13 INFO mapred.LocalJobRunner:
11/04/03 09:55:14 INFO mapred.JobClient:  map 100% reduce 0%
11/04/03 09:55:15 INFO mapred.MapTask: Finished spill 0
11/04/03 09:55:15 INFO mapred.TaskRunner: Task:attempt_local_0001_m_000000_0 is done. And is in the process of commit
11/04/03 09:55:15 INFO mapred.LocalJobRunner:
11/04/03 09:55:15 INFO mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.
11/04/03 09:55:15 INFO mapred.LocalJobRunner:
11/04/03 09:55:15 INFO mapred.Merger: Merging 1 sorted segments
11/04/03 09:55:15 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 6916705 bytes
```

图4-6 a) ImageSearch作业测试执行过程一

Fig.4-6 a) Process of ImageSearch test(first)

```
11/04/03 09:55:15 INFO mapred.LocalJobRunner:
11/04/03 09:55:17 INFO mapred.TaskRunner: Task:attempt_local_0001_r_000000_0 is done. And is in the process of commiti
11/04/03 09:55:17 INFO mapred.LocalJobRunner:
11/04/03 09:55:17 INFO mapred.TaskRunner: Task attempt_local_0001_r_000000_0 is allowed to commit now
11/04/03 09:55:17 INFO output.FileOutputCommitter: Saved output of task 'attempt_local_0001_r_000000_0' to hdfs://loca
11/04/03 09:55:17 INFO mapred.LocalJobRunner: reduce > reduce
11/04/03 09:55:17 INFO mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.
11/04/03 09:55:18 INFO mapred.JobClient:  map 100% reduce 100%
11/04/03 09:55:18 INFO mapred.JobClient: Job complete: job_local_0001
11/04/03 09:55:18 INFO mapred.JobClient: Counters: 14
11/04/03 09:55:18 INFO mapred.JobClient:   FileSystemCounters
11/04/03 09:55:18 INFO mapred.JobClient:     FILE_BYTES_READ=6950921
11/04/03 09:55:18 INFO mapred.JobClient:     HDFS_BYTES_READ=137082632
11/04/03 09:55:18 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=13902570
11/04/03 09:55:18 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=8768841
11/04/03 09:55:18 INFO mapred.JobClient:   Map-Reduce Framework
11/04/03 09:55:18 INFO mapred.JobClient:     Reduce input groups=202959
11/04/03 09:55:18 INFO mapred.JobClient:     Combine output records=208087
11/04/03 09:55:18 INFO mapred.JobClient:     Map input records=208087
11/04/03 09:55:18 INFO mapred.JobClient:     Reduce shuffle bytes=0
11/04/03 09:55:18 INFO mapred.JobClient:     Reduce output records=208087
11/04/03 09:55:18 INFO mapred.JobClient:     Spilled Records=416174
11/04/03 09:55:18 INFO mapred.JobClient:     Map output bytes=6500529
11/04/03 09:55:18 INFO mapred.JobClient:     Combine input records=208087
11/04/03 09:55:18 INFO mapred.JobClient:     Map output records=208087
11/04/03 09:55:18 INFO mapred.JobClient:     Reduce input records=208087
```

图4-6 b) ImageSearch作业测试执行过程二

Fig.4-6 b) Process of ImageSearch test(second)

(5) ImageSearch任务完成后，可以查看图像检索匹配的结果，一个示例计算结果如下所示：

1.0 200530147148X\000002.jpg

0.9733616709709167 200530147148X\000006.jpg

0.9723684787750244 200530147148X\000005.jpg

0.9630621075630188 200530147148X\000004.jpg

0.9561417102813721 2006300553580\000002.jpg

0.9528391361236572 2007300004450\000002.jpg

0.9484995603561401 2007303032861\000003.jpg

0.9483898878097534 99332814\000004.TIF

0.9469575881958008 2005300897076\000004.tif

0.9462974071502686 2005300897061\000006.tif

0.9459644556045532 2005301389004\000003.tif

0.9438977837562561 00350139\000004.TIF

0.942190945148468 99323286\000004.TIF

0.941891074180603 2005300897076\000006.tif

0.9394361972808838 2005301521328\000006.TIF

0.9390071630477905 200730110127X\000004.jpg

... ...

计算的结果是按相似度的大小进行排序，检索结果的第一列(reduce函数输出的Key值)是相似度的大小(相似度最大为1，由于测试的图像是灯具专利库中的图像，所以检索到本身)。检索结果的第二列(reduce函数的Value值)是专利图片的名称，它包含了专利图片的专利号以及该专利的视图信息。

4.4 本章小结

本章主要介绍了基于MapReduce的外观设计专利图像检索算法的实现，首先对MapReduce的编程框架中的Map函数、Reduce函数、以及Run函数进行了介绍；然后详细的对图像检索算法的MapReduce程序的实现过程进行了阐述；最后，对于实现的MapReduce程序，在Windows下的Hadoop伪分布式系统中进行了测试，并得到预期的图像检索匹配结果。

第五章 系统测试及分析

在Linux环境下，通过几台普通的PC机搭建一个Hadoop分布式文件系统，在该系统上进行图像检索的测试。在Hadoop分布式系统中，通过在不同的节点数下进行图像检索测试，将其测试结果与传统B/S架构下的图像检索系统的测试结果进行对比，并对分布式的图像检索系统的性能进行分析。

5.1 系统搭建

5.1.1 软硬件环境

Hadoop为MapReduce提供数据的分布式存储以及分布式计算平台，主节点(NameNode/JobTracker)管理Hadoop分布式系统中的数据、监控各工作节点(DataNode/TaskTracker)的状态。使用Java语言实现图像检索的算法和MapReduce程序，通过四台普通PC机搭建一个Hadoop分布式系统(一台作为Master节点，另外三台作为Slave节点)，分布式系统中节点通过100M/s的以太网连接，Hadoop版本号为0.20.2，JDK版本为jdk1.6.0_13，各PC机配置如表5-1所示。

表5-1 Hadoop分布式系统中各PC机配置

Table 5-1 PC configuration in Hadoop DFS

节点	硬件配置(CPU、内存)	IP	操作系统
HadoopMaster	AMD 3200+ 2.0GHz、1G	192.168.1.130	Linux
HadoopDataNode1	Intel Celeron 2.66GHz、512M	192.168.1.131	Linux
HadoopDataNode2	Intel Celeron 2.66GHz、512M	192.168.1.132	Linux
HadoopDataNode3	Intel Celeron 2.66GHz、512M	192.168.1.133	Linux

5.1.2 Hadoop 分布式文件系统搭建

对于Hadoop分布式文件系统，一般要求有3台以上的实体机，并在Linux环境下进行搭建，Hadoop系统的搭建可以按如下步骤：

(1) 在各节点安装Linux操作系统，在安装的过程中，创建相同的用户和密码，这里分别使用root和123456。

修改机器名，通过# hostname 机器名，修改四台PC机的机器名，同表5-1。

通过#vim /etc/hosts，在/etc/hosts中添加集群中PC机的机器名以及相应的IP。注意，需要在每个PC机上都要添加，否则可能造成各节点之间无法正常通信。各节点的/etc/hosts文件内容如下所示。

```
# /etc/hosts
192.168.1.130  HadoopMaster
192.168.1.131  HadoopDataNode1
192.168.1.132  HadoopDataNode2
192.168.1.133  HadoopDataNode3
```

(2) 安装openssh-server服务，并开启该服务，建立各节点的ssh无密码登录：首先在HadoopMaster上实现无密码登录本机，在HadoopMaster上执行如下操作。

```
# ssh -keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

完成后会在~/.ssh/生成两个文件：id_dsa 和id_dsa.pub。这两个是成对出现，类似钥匙和锁。再把id_dsa.pub追加到授权key里面(当前并没有authorized_keys文件)：$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys。完成后可以实现无密码登录本机：$ ssh localhost。

其次，实现HadoopMaster无密码登录其它节点，把 HadoopMaster上的id_dsa.pub 文件追加到各 DataNode 的 authorized_keys 内(以192.168.1.1-31节点为例)：

A、拷贝HadoopMaster中的id_dsa.pub文件：$ scp id_dsa.pub root@192.168.1.131:/root/.ssh。

B、登录192.168.1.131，执行 $ cat id_dsa.pub >> .ssh/authorized_keys

对于HadoopDataNode2和HadoopDataNode3做如上类似操作。完成以上操作后，即可实现主节点无密码登录其他数据节点。如果不能实现无密码登录，可以修改DataNode的authorized_keys：# chmod 600 authorized_keys。

(3) 关闭防火墙($ sudo ufw disable)，如果不关闭，可能会出现主节点找不到DataNode。

(4) 安装JDK，并添加环境变量(通过#vim /etc/profile实现)，如下：

```
export JAVA_HOME=/usr/java/jdk1.6.0_13
export JRE_HOME=/usr/java/jdk1.6.9_13
exportCLASS_PATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$CLASS_PATH
export JAVA_BIN=/usr/java/jdk1.6.0_13/bin
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
```

(5) 安装Hadoop(将Hadoop安装包解压到相应的目录)，各节点都需要安装Hadoop，并将Hadoop的环境变量添加到/etc/profile文件中，如下：

```
export HADOOP_HOME=/usr/Hadoop/hadoop-0.20.2
export PATH=$HADOOP_HOME/bin:$PATH
```

(6) 配置Hadoop(Hadoop的主要配置都在hadoop-0.20.2/conf目录下)：

(a) 在conf/hadoop-env.sh中配置Java环境(Master与DataNode的配置相同)：

在hadoop-env.sh文件中添加

```
export JAVA_HOME=/usr/java/jdk1.6.0_13
```

(b) 配置conf/masters 和conf/slaves 文件(只在Master节点上配置)：

```
# masters:
192.168.1.130
```

```
# slaves:
192.168.1.131
192.168.1.132
192.168.1.133
```

(c) 配置conf/core-site.xml, conf/hdfs-site.xml 及conf/mapred-site.-xml(各slave节点同master节点)，core-site.xml文件的配置如下所示：

```
# core-site.xml
<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/usr/Hadoop/tmp</value>
    </property>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://192.168.1.130:9000</value>
    </property>
</configuration>
```

hdfs-site.xml文件的配置如下：

```
# hdfs-site.xml
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
</configuration>
```

mapred-site.xml文件的配置如下：

```
# mapred-site.xml
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>192.168.1.131:9001</value>
    </property>
</configuration>
```

通过以上步骤，完成Hadoop分布式文件系统的搭建。在HadoopMaster

节点上，进入hadoop-0.20.2/bin目录，首先格式化文件系统($ hadoop namenode –format)，然后启动Hadoop($ start-all.sh)。正常启动后，用jps 命令查看进程，NameNode 上的结果如下：

```
# jps
[root@HadoopMaster bin]# jps
3453 NameNode
3590 SecondaryNameNode
3654 JobTracker
3768 Jps
```

此时，各Slave节点也已经正常启动，在各Slave节点上用jps命令查看进程，以HadoopDataNode1为例：

```
# jps
[root@HadoopDataNode1 bin]# jps
3458 TaskTracker
3377 DataNode
3548 Jps
```

对于Hadoop分布式文件系统，可通过#hadoop dfsadmin –report查看Hadoop集群的状况，也可以通过web界面查看(http://HadoopMaster:50070/)。完成Hadoop分布式文件系统的搭建后，就可以将Eclipse环境下开发的图像检索程序打包为ImageSearch.jar，将其拷贝至Hadoop的安装的bin目录下，设置好图像检索所需的数据后，就可以提交图像检索作业进行图像检索。

5.2 Hadoop 分布式文件系统中图像检索测试

5.2.1 测试数据

以外观设计专利的灯具类别[68]作为实验数据，截止2010年之前，灯具的外观专利申请量为34000多件，灯具的专利图片量为208000多幅。将Windows环境下的图像特征数据导入文本文件中，文本的每一行表示一幅图像的特征数据。为了便于测试在大规模下的数据处理，将灯具的特征数据重复使用，

以加大数据量的处理，将图像的特征数据存储于HDFS中。

对于20万条特征记录，此时数据量的大小为63M，Hadoop分布式文件系统中，数据块的默认大小为64M，因此，HDFS将20万条特征记录的文本数据作为一个数据块进行存储。在进行图像检索时，将该数据块作为一个分块给Map任务。

当前系统共用三个节点，可以分别提供3个Map任务以及Reduce任务的上限，为了测试系统的性能，将分别测试20万、40万、60万、80万、100万以及120万条特征的图像检索。

5.2.2 图像检索作业测试

向Hadoop分布式文件系统提交图像检索作业，以20万条特征记录为例。

```
$ ./hadoop jar ImageSearch.jar cn.edu.gdut.imageSerach.Image-
    Search /ImageSearchDataInput20 /ImageSearchDataOutput20
```

执行过程如图5-1所示。

```
$ ./hadoop jar ImageSearch.jar cn.edu.gdut.imagesearch.ImageSearch /ImageSearch
DataInput20 /ImageSearchDataOutPut20
11/04/03 15:27:20 INFO input.FileInputFormat: Total input paths to process : 1
11/04/03 15:27:20 INFO mapred.JobClient: Running job: job_201104030933_0001
11/04/03 15:27:21 INFO mapred.JobClient:  map 0% reduce 0%
11/04/03 15:27:35 INFO mapred.JobClient:  map 35% reduce 0%
11/04/03 15:27:38 INFO mapred.JobClient:  map 53% reduce 0%
11/04/03 15:27:41 INFO mapred.JobClient:  map 72% reduce 0%
11/04/03 15:27:44 INFO mapred.JobClient:  map 92% reduce 0%
11/04/03 15:27:47 INFO mapred.JobClient:  map 100% reduce 0%
11/04/03 15:27:56 INFO mapred.JobClient:  map 100% reduce 33%
11/04/03 15:28:02 INFO mapred.JobClient:  map 100% reduce 100%
11/04/03 15:28:05 INFO mapred.JobClient: Job complete: job_201104030933_0001
11/04/03 15:28:05 INFO mapred.JobClient: Counters: 17
11/04/03 15:28:05 INFO mapred.JobClient:   Job Counters
11/04/03 15:28:05 INFO mapred.JobClient:     Launched reduce tasks=1
11/04/03 15:28:05 INFO mapred.JobClient:     Launched map tasks=1
11/04/03 15:28:05 INFO mapred.JobClient:     Data-local map tasks=1
11/04/03 15:28:05 INFO mapred.JobClient:   FileSystemCounters
11/04/03 15:28:05 INFO mapred.JobClient:     FILE_BYTES_READ=6917030
11/04/03 15:28:05 INFO mapred.JobClient:     HDFS_BYTES_READ=68541316
11/04/03 15:28:05 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=13833771
11/04/03 15:28:05 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=8768841
11/04/03 15:28:05 INFO mapred.JobClient:   Map-Reduce Framework
11/04/03 15:28:05 INFO mapred.JobClient:     Reduce input groups=202959
11/04/03 15:28:05 INFO mapred.JobClient:     Combine output records=208087
11/04/03 15:28:05 INFO mapred.JobClient:     Map input records=208087
11/04/03 15:28:05 INFO mapred.JobClient:     Reduce shuffle bytes=6916709
11/04/03 15:28:05 INFO mapred.JobClient:     Reduce output records=208087
11/04/03 15:28:05 INFO mapred.JobClient:     Spilled Records=416174
11/04/03 15:28:05 INFO mapred.JobClient:     Map output bytes=6500529
11/04/03 15:28:05 INFO mapred.JobClient:     Combine input records=208087
11/04/03 15:28:05 INFO mapred.JobClient:     Map output records=208087
11/04/03 15:28:05 INFO mapred.JobClient:     Reduce input records=208087
```

图5-1 Hadoop系统中图像检索的执行过程

Fig.5-1 Process of image retrieval in Hadoop

对于图像检索过程中Map任务以及Reduce任务的完成情况以及相关信息，可以通过web界面查看，如图5-2所示

User: wei
Job Name: ImageSearch
Job File: hdfs://localhost:8888/tmp/hadoop-wei/mapred/system/job_201104030933_0001/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Sun Apr 03 15:27:20 CST 2011
Finished at: Sun Apr 03 15:28:04 CST 2011
Finished in: 44sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	1
	Launched map tasks	0	0	1
	Data-local map tasks	0	0	1
FileSystemCounters	FILE_BYTES_READ	240	6,916,790	6,917,030
	HDFS_BYTES_READ	68,541,316	0	68,541,316
	FILE_BYTES_WRITTEN	6,916,981	6,916,790	13,833,771
	HDFS_BYTES_WRITTEN	0	8,768,841	8,768,841
Map-Reduce Framework	Reduce input groups	0	202,959	202,959
	Combine output records	208,087	0	208,087
	Map input records	208,087	0	208,087
	Reduce shuffle bytes	0	6,916,709	6,916,709
	Reduce output records	0	208,087	208,087
	Spilled Records	208,087	208,087	416,174
	Map output bytes	6,500,529	0	6,500,529
	Map output records	208,087	0	208,087
	Combine input records	208,087	0	208,087
	Reduce input records	0	208,087	208,087

图5-2 图像检索作业统计信息

Fig.5-2 Statistical information of image retrieval

图像检索作业的时序图如图5-3所示。

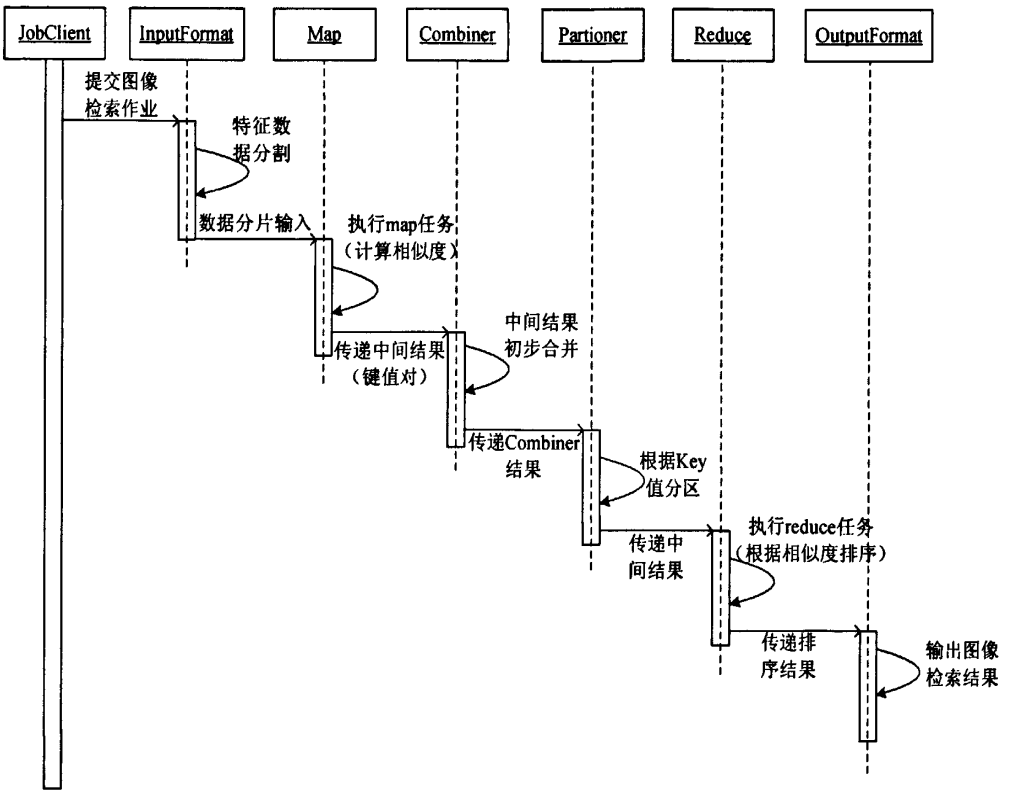

图5-3 图像检索作业执行时序图

Fig.5-3 Sequence diagram of image retrieval

5.3 Hadoop 分布式文件系统性能分析

5.3.1 与传统 B/S 单节点系统性能对比

图像特征的数据量分别为20万、40万、60万、80万、100万以及120万条时，在不同节点数(节点数分别为1、2、3)以及B/S单节点模式下，测试图像检索的耗时，实验结果如图5-4所示。

实验表明，B/S单节点模式，随着图像特征数据递增，检索时间线性递增，当数据量达到大规模甚至海量级别时，其检索耗时将非常漫长，甚至无法完成检索任务。Hadoop集群节点数为1时，检索速度比B/S单节点模式慢，这是由于Hadoop分布式系统在执行MapReduce程序时，任务的初始化、分配以及清空作业的耗时所造成的。

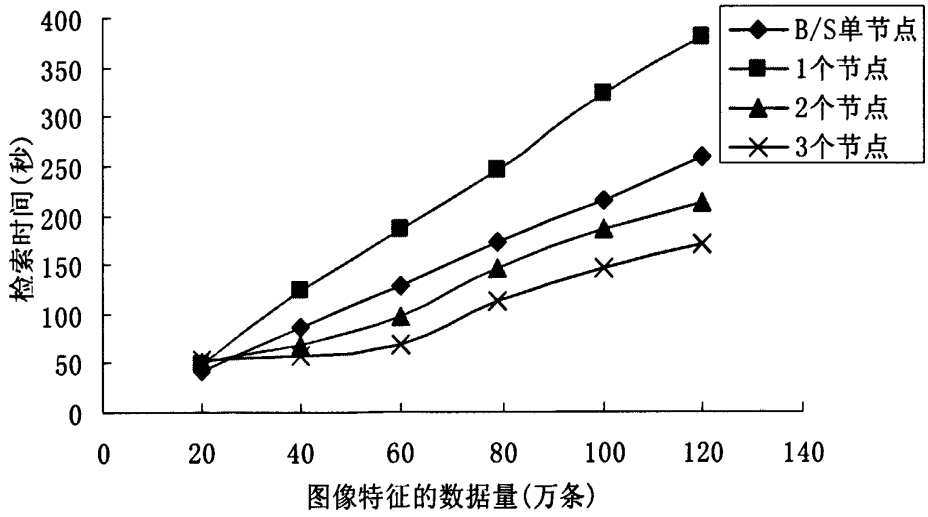

图5-4 不同节点数下检索时间的对比

Fig.5-4 The contrast of retrieval time when in different configuration

Hadoop分布式系统节点数为2和3时，图像特征数据量为20万条记录，此时Hadoop分布式系统检索速度比B/S单节点模式慢，因为此时数据量为63MB，Hadoop将整个图像特征库作为一个Map任务的输入进行处理(即只有一个节点执行一个Map任务，任务的初始化、分配以及清空作业的耗时造成检索时间的延长)。数据量为40万、60万、80万、100万以及120万时，图像检索的速度有了很大提高。数据量为40万时，此时有2个Map任务，分布式系统节点数为2和3的检索时间非常接近，但相比B/S单节点模式，检索速度分别提高了21%和34%。数据量为60万时，此时有3个Map任务，分布式系统节点数为2和3的检索时间相对于B/S单节点模式分别提高了25%和47%。

数据量继续增大时(数据量大于60万)，此时Hadoop分布式系统(节点数为2和3)的检索时间成线性增长，因为Map任务已超出了分布式系统的节点数，部分节点会分配多个Map任务，而一个节点同一时刻只能处理一个Map任务，从而使检索时间呈线性增长。

5.3.2 负载均衡性能分析

通过向Hadoop分布式系统(节点数为3)提交检索任务，在不同时间点以及

不同数据量下测试各节点的负载情况，记录各节点CPU的使用率，时间采样平均分布在检索任务的执行时间内，统计结果如图5-5、图5-6以及图5-7所示。

实验表明，数据量为40万时，只有2个Map任务。2个Map任务分别分配给DataNode1和DataNode3，t1和t2时刻(时间点间隔为10s)，两个节点的Map任务在执行中。t3时刻，DataNode3节点的Map任务执行完毕，并在该节点开始执行Reduce任务，DataNode1节点的Map任务还在执行。在t4时刻，DataNode1节点上的Map任务完成，该节点将Map任务产生的中间结果交由DataNode3进行Reduce处理。t5时刻，只有DataNode3在执行Reduce任务，DataNode1和DataNode2处于空闲状态。t6时刻，整个检索任务完成，各节点处于空闲状态。处理80万(时间点间隔为25s)和120万(时间点间隔为40s)数据量时，各节点的负载情况类似于处理40万数据量的负载情况。

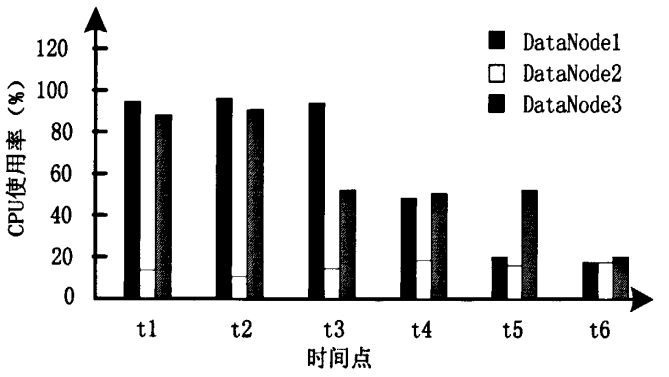

图5-5 处理40万条数据时各节点CPU使用率

Fig.5-5 The state of CPU when processing data base which contains 400,000 records

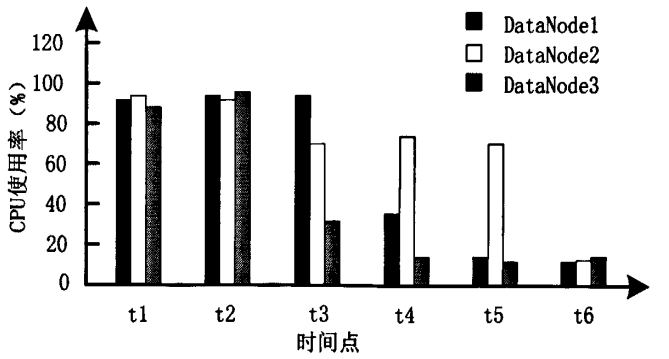

图5-6 处理80万条数据时各节点CPU使用率

Fig.5-6 The state of CPU when processing data base which contains 800,000 records

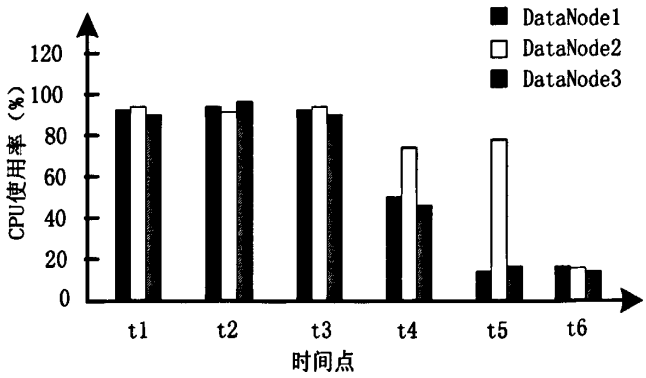

图5-7 处理120万条数据时各节点CPU使用率

Fig.5-7 The state of CPU when processing data base which contains 1,200,000 records

检索系统以分布式计算框架(Hadoop)为基础，主要由MapReduce的执行以及分布式文件系统(HDFS)组成。将传统单节点模式工作转变为集群各节点之间的协同工作。分布式系统构架中，机器集群作为硬件资源池，将并行任务拆分给空闲节点进行处理，能极大地提高计算效率，这种资源的无关性，也便于集群的扩展，系统的性能随集群规模的增大成线性增加。

该系统构架下，系统负载的均衡体现在以下四个方面：

(1) 数据存储，HDFS支持数据的均衡，某个节点空闲空间低于某个临界点时，自动启动任务将负载重的节点中数据转移到空闲节点。当某个文件请求突然增多时，在其他节点创建新的副本以降低I/O操作的负载，实现动态的负载均衡。

(2) Map任务和Reduce任务初始分配时，根据节点状况尽可能合理分配任务，Map任务分配在距离数据近的节点上，实现Map任务的本地读取数据，减少传输数据所带来的带宽损耗，Reduce任务分配在空闲的节点上，便于某个Map任务完成后，立即启动Reduce任务。

(3) MapReduce任务执行过程中时刻监督以及及时调整，Map任务和Reduce任务执行过程中，可能出现某个任务失败的情况，通过在其他TaskTracker上执行的备份任务来解决任务失败问题，加快故障恢复的速度，提高系统的检索速度。

(4) 可扩展性，Hadoop可以在普通的商用硬件上良好运行，方便的添加节点以扩展系统的资源，提高系统的存储和计算能力。

总之，该系统架构提供了一种廉价有效的方法扩展服务器带宽、增加吞

吐量、加强网络数据处理能力以及提高网络的灵活性和可用性。解决网络拥塞问题，服务就近提供，实现地理位置无关性。为用户提供更好的访问质量，提高服务器响应速度以及其他资源的利用效率，避免了网络关键部位出现单点失效。

5.3.3 可扩展性分析

Hadoop框架可以运行在普通的PC机上，对硬件没有很高的要求，Hadoop的数据冗余机制以及MapReduce任务的尝试和推测式执行，保证了分布式计算的高可靠性。分布式系统的节点数可以方便的按需进行扩展，从而提高分布式系统的存储和计算能力。

实验表明，当处理的数据量大于20万条特征时，Hadoop分布式系统在处理等量数据时，检索速度随着节点数的增加而逐渐加快。当数据量持续增加并达到Hadoop分布式系统并行处理Map任务上限时(上限特征数据量为60万条，由分布式系统的节点数决定)，分布式系统节点数2和3的检索时间开始线性增长。因为Map任务已经超出了分布式系统并行处理的能力，部分节点分配了多个Map任务，而一个节点在同一时刻只能处理一个Map任务，造成一些Map任务需要排队等待。可以向Hadoop分布式系统添加节点，增强分布式系统并行处理Map任务的能力，提高分布式系统的检索速度。

5.4 检索结果图

根据提交的灯具示例图像，进行外观设计专利的图像检索，图5-8、图5-9是两个示例图像检索的实验结果。

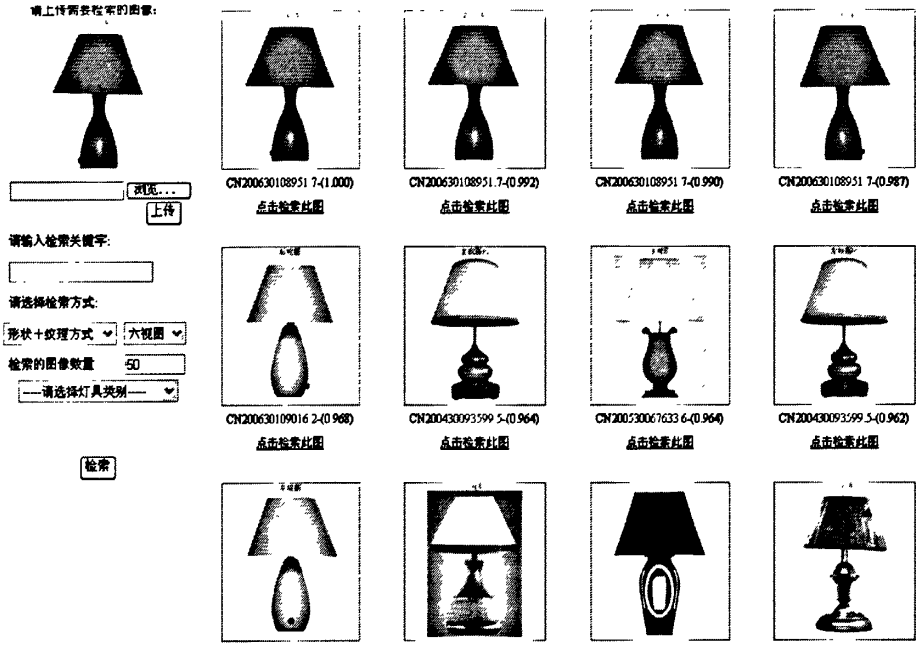

图5-8 检索结果示例一

Fig.5-8 The first image retrieval result

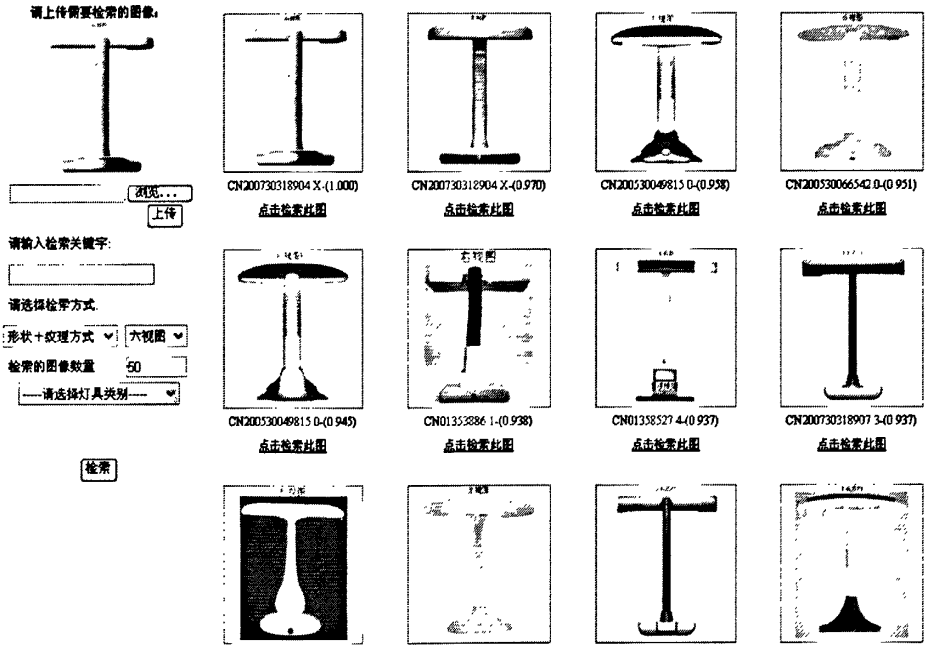

图5-9 检索结果示例二

Fig.5-9 The second image retrieval result

5.5 本章小结

本章主要是对Hadoop分布式文件系统进行图像检索测试以及性能的分析。首先，介绍了Hadoop分布式文件系统搭建的软硬件环境以及其搭建的过程；其次，对Hadoop分布式文件系统在不同特征数据量下进行图像检索测试，得到相关的测试结果；再次，将Hadoop分布式文件系统的测试结果与传统B/S单节点系统的测试结果进行性能上的比对，并得到相关的结论，并由此分析了Hadoop分布式文件系统在进行图像检索过程中的负载均衡情况，以及给出了该系统的可扩展性分析；最后，给出了图像检索的实验效果图。

总结与展望

随着经济全球一体化的发展和知识产权经济的蓬勃发展，专利发展水平已成为衡量一个地区综合实力、发展能力和核心竞争力的战略性标志。外观设计专利一直被誉为“小专利，大市场”，在市场竞争中发挥着举足轻重的作用，这迫切的需要加快外观专利信息化的发展。外观设计专利的检索中，获取的信息主要是图像，基于内容的图像检索技术为外观设计专利的检索指明了方向，但传统B/S单节点的图像检索系统存在检索速度慢、并发性差以及不能处理大规模的数据。

针对传统检索模式存在的问题，本文提出了基于Hadoop的外观专利图像检索系统。主要对Hadoop分布式系统关键技术进行了研究，将Hadoop分布式存储和MapReduce并行计算框架运用于外观设计专利的图像检索。把传统的B/S单节点的图像检索任务进行分解，在Hadoop分布式系统中通过各节点协同完成图像检索任务，以提高图像检索的速度、并发性以及处理数据的能力。本文围绕系统的设计与实现，做了以下工作：

(1) 通过对基于图像的外观专利检索技术，以及其国内外研究现状进行了充分的调研，分析了现有系统存在的问题，提出通过分布式来解决存在的问题。

(2) 介绍了开源的Hadoop平台，对HDFS、MapReduce编程框架的运行原理、特点和应用领域进行了分析，接着对基于内容的图像检索技术进行了阐述。

(3) 通过对外观设计专利图像检索的需求以及传统B/S单节点图像检索框架的不足，提出了基于Hadoop的外观专利图像检索系统，对图像检索系统的HDFS模块以及MapReduce模块的设计进行了阐述。

(4) 通过在Windows环境下的开发以及简单的测试，在MapReduce框架中实现了外观设计专利图像检索算法，从而实现了在Hadoop分布式系统中通过分布式计算来提高图像检索的速度、并发性以及系统的数据处理能力。

(5) 搭建Hadoop分布式系统，在该系统上运行图像检索作业。通过在不同数据量下的测试，与传统B/S单节点图像检索模式的比较，验证了基于Hadoop的图像检索系统在图像检索速度、并发性、数据处理能力方面的优势，并对该系统的负载均衡、可扩展性进行了分析。

当前Hadoop分布式系统的规模很小，当处理的数据量达到一定级别时，会超过分布式系统并行执行Map任务的上限，此时会降低Hadoop分布式系统并行执行的效率。

未来的工作主要有以下几点：

(1) 扩展Hadoop分布式系统的节点数，调整分布式系统的相关参数，以提高Hadoop分布式系统的数据存储、处理能力以及工作效率。

(2) 对于用户提交的检索信息，将根据检索信息进行数据的筛选，减小检索的范围，从而提高系统的检索速度。

(3) 优化图像检索的算法，提高检索的准确度和速度。

参考文献

[1] 孙璐. 外观设计专利保护研讨会综述[J]. 知识产权，1998，(4)：33-34.

[2] 陈宇萍. 外观设计专利图像检索系统研究[J]. 科技管理研究，2005，(4)：162-164.

[3] 中华人民共和国国家知识产权局. 专利检索[EB/OL]. 北京：2008[2011]. http://www.sipo.gov.cn/sipo2008/zljs/.

[4] 方骥，戴青云. 基于图像内容的外观专利自动检索系统[J]. 计算机工程与应用，2004，(34)：209-211.

[5] 鲍黎黎. 外观设计保护制度与我国经济发展之适应[J] . 知识产权，2006，(4)：78-80.

[6] 戴青云，李海鹏. 基于纹理和形状特征的外观设计专利图像的检索方法[J]. 计算机工程与应用，2002，(3)：27-29.

[7] 周明全，耿国华，韦娜. 基于内容图像检索技术[M]. 北京：清华大学出版社，2007.

[8] 郑湃，崔立真，王海洋，徐猛. 云计算环境下面向数据密集型应用的数据布局策略与方法[J]. 计算机学报，2010，33(8)：1472-1480.

[9] Flickner M. Query By Image and Video Content：The QBIC System [J]. IEEE Computers，1995，28(9)：23-32.

[10] Bach Jeffrey R，Fuller Charles，Gupta Amarnath，et al. Virage image search engine：An open framework for image management[C]. Proceedings of SPIE Storage and Retrieval for Still Image and Video Databases IV，1996，76-87.

[11] Pentland A，Picard R，Sclaroff S. Photobook：Content-based Manipulation of Image Database[J]. International Journal of Computer Vision，1996，18(3)：233-254.

[12] Smith J R，Chang S F，VisualSEEk：A fully automated content-based image query system[C]. Proceedings of the 1996 4th ACM International Multime- dia Conference，Boston，MA，USA，1996，87-98.

[13] State intellectual property office of China. Intelligent of Design Patent Search [EB/OL]. Beijing：2008[2011]. http://www.disc.gov.cn/dpss/serv- let/Invoker/Login.

[14] GuangDong intellectual property research and development center. GuangDong Patent Information Service Platform[EB/OL]. Guangzhou：2009[2011]. http://www.gdzl.g-

ov.cn.

[15] 陈全，邓倩妮. 云计算及其关键技术[J] . 计算机应用，2009，29(9)：2562-2566.

[16] 石柯，徐胜超，唐晓辉，江锋，章勤. 一种分布式环境下的新型高性能计算平台[J]. 小型微型计算机系统，2006，27(9)：1782-1787.

[17] Erik S. Gough，Michael D. Kane. Evaluating Parallel Computing Systems in Bioinformatics[C]. Proceedings of the Third International Conference on Information Technology：New Generations，IEEE，2006，233-238.

[18] Zhang Shuai，Zhang Shufen，Chen Xuebin，Huo Xiuzhen. The Comparison Between Cloud Computing and Grid Computing[C]. 2010 International Conference on Computer Application and System Modeling，IEEE，2010，(11)：72-75.

[19] Amazon. Amazon Elastic Compute Cloud (Amazon EC2)[EB/OL]. 2008 [2011]. http://aws.am azon.com/ec2/.

[20] Amazon. Amazon Simple Storage Service (Amazon S3) [EB/OL]. 2008 [2011]. http://aws.amazon.com/s3/.

[21] 王庆波，金淳，何乐，赵阳. 虚拟化与云计算[M]. 北京：电子工业出版社，2009.

[22] Gupta Rajeev，Gupta Himanshu，Nambiar Ullas，Mohania, Mukesh. Efficiently querying archived data using Hadoop[C]. Proceedings of the 19th International Conference on Information and Knowledge Managem- ent，2010，1301-1304.

[23] Thusoo Ashish，Jain Namit，Shao Zheng，et al. Hive：A petabyte scale data warehouse using hadoop[C]. Proceedings of the 26th IEEE International Conference on Data Engineering，2010，996-1005.

[24] Wang Fei，Ercegovac Vuk， Beymer David，et al. Large-scale multimodal mining for healthcare with MapReduce[C]. Proceedings of the 1st ACM International Health Informatics Symposium，2010，479-483.

[25] 程莹，张云勇，徐雷，房秉毅. 基于Hadoop及关系型数据库的海量数据分析研究[J]. 电信科学，2010，(11)：47-50.

[26] Apache Software Foundation. Hadoop[EB/OL]. 2008[2011]. http://hadoop.apache.org/.

[27] Sanjay Ghemawat，Howard Gobioff，Shun-Tak Leung. The Google filesystem[C]. Proceedings of the 19th ACM Symposium on Operating Systems Principles. Bolton Landing：ACM，2003，29-43.

[28] Jeffrey Dean，Sanjay Ghemawat. MapReduce：simplified data processing on large clusters[C]. Proceedings of the 6th Symposium on Operating Systems Design and Implementation. San Francisco：Google Inc，2004，107-113.

[29] Fay Chang，Jeffrey Dean，et al. Bigtable：A Distributed Storage System for Structured Data[C]. 7th OSDI，2006，276-290.

[30] Konstantin Shvachko，Hairong Kuang，Sanjay Radia，et al. Hadoop distributed file system for the Grid[C]. Proceedings of the Nuclear science Symposium Conference Record (NSS/MIC)，IEEE，2009，1056-1061.

[31] Apache Software Foundation. Pig[EB/OL]. 2008[2011]. http://pig.apach- e.org.

[32] Apache Software Foundation. Hive[EB/OL]. 2008[2011]. http://hive.apach- e.org.

[33] Apache Software Foundation. Zookeeper [EB/OL]. 2008[2011]. http://zookeeper.apache.org/.

[34] Tom White. Hadoop：The Definitive Guide [M]. O'Reilly Media，Inc.，2009.

[35] 黄晓云. 基于HDFS的云存储服务系统研究[D]. 大连：大连海事大学，2010.

[36] 封俊. 基于Hadoop的分布式搜索引擎研究与实现[D]. 太原：太原理工大学，2010.

[37] 桂兵祥，何健. 基于高性能云的分布式数据挖掘方法[J]. 计算机工程，2010，36(5)：76-78.

[38] Yang Shengqi，Wang Bai，Zhao Haizhou，et al. Efficient Dense Structure Mining using MapReduce[C]. 2009 IEEE International Conference on Data Mining Workshops，2009，332-337.

[39] Jeffrey Dean，Sanjay Ghemawat. MapReduce：A Flexible Data Processing Tool [J]. Communications of The Acm，2010，53(1)：72-77.

[40] Tian Xia. Large-Scale SMS Messages Mining Based on Map-Reduce[C]. Proceedings of the International Symposium on Computational Intelligence and Design，2008，7-12.

[41] Aaron McKenna，Matthew Hanna，Eric Banks，et al. The Genome Analysis Toolkit：A MapReduce framework for analyzing next-generation DNA sequencing data[J]. Genome Research，2010，(20)：1297-1303.

[42] Yulai Yuan，Yongwei Wu，Xiao Feng，et al. VDB-MR：MapReduce-based distributed data integration using virtual database [J]. Future Generation Computer Systems，2010，(26)：1418-1425.

[43] InfoWorld. 2009's top 10 emerging enterprise technologies[EB/OL]. 2009 [2011]. http://www.infoworld.com/t/development-frameworks/2009s-top-10-emerging-enterprise-technologies-467.

[44] 谢邦旺，王加俊. 一种基于轮廓的图像检索算法[J]. 中国图象图形学报，2008，13(7)：1367-1373.

[45] 李清勇，胡宏，施智平，等. 基于纹理语义特征的图像检索研究[J]. 计算机学报，2006，29(1)：116-123.

[46] 黄元元，何云峰. 一种基于颜色特征的图像检索方法[J]. 中国图象图形学报，2006，11(12)：1768-1774.

[47] 丁国祥，吴仁炳，汪祖媛，王煦法. 组合颜色、空间和纹理特征的图像检索[J]. 小型微型计算机系统，2004，25(12)：2251-2253.

[48] Andrew Troelsen (美). 王少葵，张大磊等译. C#与.NET 3.0高级程序设计[M]. 北京：人民邮电出版社，2008.

[49] 刘小虎，蒋从锋，王乘. 基于网格的分布式虚拟环境仿真海量数据管理[J]. 计算机工程与设计，2008，29(4)：931-933.

[50] 廖军，谭浩. 新一代开放分布式处理技术——Web Services[J]. 计算机应用，2004，24(8)：5-9.

[51] Tom Barnaby (美). 黎媛，王小峰等译. .NET分布式编程—C#篇[M]. 北京：清华大学出版社，2004.

[52] 胡艳丽，张维明，肖卫东，汤大权. 计算网格中基于时间均衡的并行粗粒度任务调度算法[J]. 小型微型计算机系统，2008，29(1)：124-129.

[53] Jason Venner. Pro Hadoop[M]. Apress，Inc.，2009.

[54] 曹宁，吴中海，刘宏志，张齐勋. HDFS下载效率的优化[J]，计算机应用，2010，30(8)：2060-2065.

[55] Hadoop开发者. Hadoop开发者入门专刊[EB/OL]. 2010[2011]. http://www. Hadoop-or.com.

[56] Red Hat，Inc.. Cygwin [EB/OL]. [2011]. http://www.cygwin.com/.

[57] OpenSSL官网. OpenSSL[EB/OL]. [2011]. http://www.openssl.org/.

[58] 世界知识产权组织. 国际外观设计分类表(第8版)(中文版) [M]. 北京：知识产权出版社，2006.

攻读硕士学位期间发表的学术论文

王贤伟，戴青云，姜文超，曹江中. 基于MapReduce的外观设计专利图像检索方法. 小型微型计算机系统(已录用)

学位论文独创性声明

本人郑重声明：所呈交的学位论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明，并表示了谢意。本人依法享有和承担由此论文所产生的权利和责任。

论文作者签名：王贤伟　日期：2011.6.9

学位论文版权使用授权声明

本学位论文作者完全了解学校有关保存、使用学位论文的规定，同意授权广东工业大学保留并向国家有关部门或机构送交该论文的印刷本和电子版本，允许该论文被查阅和借阅。同意授权广东工业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、扫描或数字化等其他复制手段保存和汇编本学位论文。保密论文在解密后遵守此规定。

论文作者签名：王贤伟　日期：2011.6.9

指导教师签名：[illegible]　日期：2011.6.9

致 谢

值此论文完成之际，我要特别感谢在三年研究生生活中给予我指导、帮助、关心的老师、同学、亲人们：

首先，感谢我的导师戴青云教授，三年的研究生学习和工作中，戴老师给予了我全力帮助和精心的指导。从课题的选择、研究工作的展开、项目实现的指导以及最后论文的形成，戴老师都倾注了大量的心血。在这三年中，戴老师严谨的治学态度和高尚的人格魅力使我印象深刻、受益终生。在戴老师悉心教导下，不仅让我掌握了科学的学习和科研方法，而且在为人处事方面也得到了很大的提高，跟戴老师学习的这三年时光，将是我人生中一笔宝贵的财富。

同时，在研究生阶段的项目设计开发过程中得到了姜文超老师、曹江中老师的很多有益的建议、指导和帮助，在此表示深深的谢意。

感谢我的家人，特别是我的父母，是他们一直无私的付出才能让我顺利地完成我的学业，才能让我在学生时代更为专心地学习，也正是有他们一如既往的鼓励，才能让我更从容地面对许多困难，面对成长路上的挫折。

感谢实验室的兄弟姐妹：李旭明、黄军记、曹璐、胡晓、邓潇宇、张其良，感谢你们在学习和生活中给予我的支持和帮助，永远怀念和你们共同度过的美好时光。感谢给过我帮助的所有师兄、师姐和师弟、师妹以及同学们。

王贤伟

2011年4月8日于广州