太原理工大学

硕士学位论文

基于Hadoop的分布式搜索引擎研究与实现

姓名：封俊

申请学位级别：硕士

专业：软件工程

指导教师：胡彧

20100401

基于 Hadoop 的分布式搜索引擎研究与实现

摘 要

分布式搜索引擎是一种结合了分布式计算技术和全文检索技术的新型信息检索系统。它改变了人们获取信息的途径，让人们更有效地获取信息，现在它已经深入到网络生活的每一方面，被誉为上网第一站。

目前的搜索引擎系统大多都拥有同样的结构——集中式结构，即系统所有功能模块集中部署在一台服务器上，这直接导致了系统对服务器硬件性能要求较高，同时，系统还有稳定性差、可扩展性不高的弊端。为了克服以上弊端就必须采购极为昂贵的大型服务器来满足系统需求，然而并不是所有人都有能力负担这样高昂的费用。此外，在传统的信息检索系统中，许多都采用了比较原始的字符串匹配方式来获得搜索结果，这种搜索方式虽然实现简单，但在数据量比较大时，搜索效率非常低，导致用户无法及时获得有效信息。以上这两个缺点给搜索引擎的推广带来了很大的挑战。为应对这个挑战，在搜索引擎系统中引入了分布式计算和倒排文档全文检索技术。

本文在分析当前几种分布式搜索引擎系统的基础上，总结了现有系统的优缺点，针对现有系统的不足，提出了基于 Hadoop 的分布式搜索引擎。主要研究工作在于对传统搜索引擎的功能模块加以改进，对爬行、索引、搜索过程中的步骤进行详细分析，将非顺序执行的步骤进一步分解为两部分：数据计算和数据合并。同时，应用 Map/Reduce 编程模型思想，把数据计算任务封装到 Map 函数中，把数据合并任务封装到 Reduce 函数中。经过以上改进的搜索引擎系统可以部署在廉价PC构成的Hadoop分布式环境中，并具有较高的响应速度、可靠性和扩展性。这与分布式搜索引擎中的技术需求极为符合，因此本文使用 Hadoop 作为系统分布式计算平台。此外，系

统使用了基于倒排文档的全文检索技术，构建了以关键词为单位的倒排索引模块，同时结合 TF-IDF 和 PageRank 算法，改进了网页评分策略，优化了搜索结果。

最后，详细分析了在应用 Map/Reduce 编程模型实现系统模块过程中遇到的问题，及其解决方案。构建了一个 4 节点的小型分布式搜索引擎系统，通过对网络资源的爬行、索引和检索，以及对系统进行可靠性和扩展性测试，获得实验数据。在分析实验数据的基础上，验证了所提出的基于 Hadoop 的分布式搜索引擎的合理性。

关键词： Map/Reduce，Hadoop，分布式计算，搜索引擎

THE RESEARCH AND IMPLEMENTATION OF DISTRIBUTED SEARCH ENGINE BASED ON HADOOP

ABSTRACT

Distributed Search Engine is a brand new information retrieval system which is consisted of distributed computing technology and full-text retrieval technology. It has changed the way of achieving informations for people and has made it more effectively. Now it has been deep into every aspects of the Internet, and it is known as the first Step of navigation.

At present, most of the search engine system are structured similarly - centralized structure, which means all of system's modules are deployed on one server, and it also result in the server must be of high performance，meanwhile, the system still have poor stability and bad scalability. In order to deal with these disadvantanges, people have to purchase very large and expensive servers to satisfy the system requirements, however, not everyone have the ability to afford such high cost. In addition, a primitive string matching mode was adopted to gain the results in many traditional information retrieval systems. Although this method is simple, the search efficiency became very low when data volume is huge, and customers could not retrieve useful informations in time. The two disadvantages mentioned above was a big challenge to the promotion of search engine. In order to deal with this challenge, the technology of distributed computing and inverted document full-text retrieval were introduced into the search engine system.

In this paper, it summaried the advantages and disadvantages based on an analysis of several distributed search engine systems. In order to deal with the

existing drawbacks, it proposed a distributed search engine based on Hadoop. The main tasks of this paper are to improve the traditional search engine function modules, analyze the steps on the crawling, indexing, searching, in the process, and further decomposed these process that can be excuted disorderly into two parts: data computing and data combining. Then, packaged the algrithm of data computing into Map function, and the algrithm of data combining into Reduce function by using Map/Reduce programming thinkings. After the implementation of these technologies, it improved search engine system could be deployed on a Hadoop distributed environment which was structured by some low-cost PCs, so this system had high response speed, reliability and scalability. Because of the technology closed to the distributed search engine's needs. In this paper, it used Hadoop distributed computing platform as a system. Besides, this paper constructed with keywords for inverted indexing module, by using the inverted document based full-text retrieval technology. And it combined with TF-IDF and PageRank algorithm to improve the page score strategy and optimize the search results.

Finally, a detailed analysis of how to use Map/Reduce programming model to achieve system module has proposed as well as the difficulties in the implementation process, and it built a small distributed search engine system with four nodes, the experimental data was achieved by means of the crawling, indexing and retrieving through Internet, and tested system reliability and scalability. In the analysis of this experimental data, the rationality of the distributed search engine based on Hadoop has been validated.

KEY WORDS: Map/Reduce, Hadoop, Distributed compute, Search Engine

第一章 绪论

1.1 课题研究背景

随着互联网的飞速发展，大量网站如同雨后春笋般大批涌现，截止 2009 年，全球互联网新增网站达 4600 万，总计达 2.315 亿。全球数字化信息总量也呈现出快速增长势头。随着社交网站不断涌现、上网手机层出不穷，目前全世界的数字内容约为 4870 亿吉比特（GB），从如此海量的数据中检索出需要的信息就必须使用搜索引擎了，根据权威调查机构正望咨询 09 年搜索引擎用户调查报告显示，搜索引擎的渗透率持续提高，有 91%的网民每个月至少使用一次搜索引擎，此数字比上一年增加了 4 个百分点，搜索引擎是许多网民上网的第一站[1]。毫无疑问，搜索引擎已经成为互联网上人们获取信息的最重要手段。

目前，搜索引擎研究一直都处于持续升温状态，几大通用搜索引擎，如 Google、Baidu、Yahoo 等已经牢牢的控制了搜索市场，而软件巨头微软也于 2009 年 6 月推出了全新的搜索引擎 Bing，加入了搜索行列。同时，搜索引擎作为基础软件在网站内部取得了广泛应用，不仅大型门户网站如美国在线、雅虎、亚马逊等等每一个著名网站的首页都在显著位置放置了搜索框，就连迪斯尼、麦当劳、美孚石油这些传统企业也都无一例外地在它们的首页上放置了搜索框或搜索功能的链接，美国 500 强中使用搜索引擎的网站几乎达到 100%。总的来看，国外主流的搜索引擎技术已比较成熟，无论从结果、性能还是稳定性来看，都能提供令人满意的结果，并且已经在人们的日常信息获取中发挥作用。

反观国内，目前对搜索引擎潜在价值的开发明显不够，除了网易、新浪、搜狐等少数门户网站使用了搜索引擎外，大多数网站使用的还是基于数据库关键字匹配的简单信息检索系统。这种检索方式，效率低，检索结果准确性也不尽如人意。那些拥有搜索引擎的门户网站，其搜索引擎系统都采用了同样的体系结构—集中式结构，即运行各子系统的服务器在物理上集中分布。这也导致了系统对单台服务器性能要求高，易出现故障，可扩展性差等缺点。

于是，为了使系统得到最佳的性能、最好的容错性和最容易的扩展性，分布式搜索引擎应运而生了。分布式搜索引擎是由若干台计算机组成的系统，每台计算机都具有独

立的物理资源，系统内的每个子系统都可以利用这些资源高度自治而相互合作地工作，共同完成爬行、索引和搜索任务。同时，系统资源还具有冗余性，保证在一台或几台计算机出现故障后，系统不会出错和瘫痪。

分布式搜索引擎系统解决了现有系统对单台服务器性能要求高，易出现故障，可扩展性差等缺点。因此，研究这一课题是很有意义的。

1.2 搜索引擎的发展

最早现代意义上的搜索引擎出现于1994年7月。当时Michael Mauldin将John Leavitt的蜘蛛程序接入到其索引程序中，创建了Lycos，除了相关性排序外，Lycos还提供了前缀匹配和字符相近限制。1994年7月20日，数据量为54,000的Lycos正式发布。同年4月，斯坦福大学的两名博士生，David Filo和美籍华人杨致远（Gerry Yang）共同创办了超级目录索引Yahoo，随着访问量和收录链接数的增长，Yahoo目录开始支持简单的数据库搜索。1998年10月，由Larry Page和Sergey Brin两人搭档创办的搜索引擎Google正式推出，以其独特的网页排序、动态摘要、网页快照、每日更新等特点，改写了现代搜索引擎的历史，并成功地使搜索引擎的概念深入人心。从此搜索引擎进入了高速发展时期。经过多年的发展，搜索引擎的子系统划分已经很明确，一般包括3个部分[2]：

1. 网络爬行器（Crawler）：爬行器从一个或数个种子超链接出发，发现新的网页并保存这些网页的快照，然后分析它们，从中提取网页链接，扩充链接列表库以供爬行器下一轮爬行使用。爬行器循环执行这一流程从网络上抓取所需网页数据。

2. 索引器（Indexer）：索引器的功能是对网页内容进行分词，建立倒排索引文档。其中分词功能是非常重要的一环，众所周知，英文是以词为单位的，词和词之间是靠空格或标点隔开，而中文是以字为单位，句子中部分字只有连起来才能描述一个意思。中文分词的好坏直接关系到搜索的准确性，而索引文档的存储结构又关系到搜索的响应速度。

3. 查询器（Searcher）：查询器的功能是向用户提供查询界面，在收到用户的查询请求时，解析查询条件，检索倒排索引文档，并根据一定的评分策略，对匹配的结果集进行排序，并返回检索结果。

目前，除了那些大名鼎鼎的商业搜索引擎以外，大部分的搜索引擎都将这三种子系统集中部署，即集中式结构。随着索引信息量的快速增长和用户搜索量上升，系统的处

理性能、容错性、扩展性，以及灾难恢复性差的缺点日趋明显，严重影响了搜索引擎的应用。因此，在系统结构方面，下一步的发展方向是分布式结构。

1.3 分布式搜索引擎研究现状

为了解决集中式结构存在的诸多缺点，人们很早就开始了搜索引擎分布式结构的研究。近几年已经开发出几种应用分布式信息检索的系统[3]，主要有：

（1）ROADS 主题信息检索系统，它由 Loughborough 技术大学的计算机科学系、Britol 大学的 ILRT 和 Bath 大学的 UKOLN 联合开发，应用了 WHOIS++协议。WHOIS++是一种简单的分布式信息查找协议，被用于连接许多基本级服务器和索引服务器，以形成统一的目录服务器，提供灵活的方式实现跨平台查询。然而，WHOIS++并没有提供许多查询功能，虽然它易于操作，但是为了实现查询中的自动推荐，必须在客户程序上进行修改。每一个希望参加到同一目录服务之中的基本级服务器和索引服务器必须生成一个“前期知识”，以说明它所包括的款目，因此，其扩展能力也受到一定的限制。目前，其应用范围一般局限于数字图书馆领域。

（2）Issac 项目由美国国家自然科学基金赞助，由 University of Wisconsin-Madison 的计算机科学系承担，主要目的是帮助高等教育团体实现 Internet 上的资源发现。其目标是链接地理上各自分布的 metadata 数据集，使之成为一个整体上统一的数据资源库，通过利用 LDAP（Lightweight Directory Access Protocol）以及 CIP（Common Indexing Protocol）协议，来实现分布式查询提交，数据返回，交换索引信息的结构[4]。Issac 搜索引擎在分布式搜索方面具有很大进步，但是，它仍有两个致命缺陷：在查询的相关性方面，有巨大的障碍，系统常常给出许多不相关的检索内容，在使用的时候，用户将很多的精力放在内容的判断上。

（3）基于 P2P 的分布式搜索引擎。P2P 采用全分布式网络体系结构，它也称为分布式对等网络[5]。不再使用中央服务器，消除了中央服务器带来的稳定性问题。没有中央控制点，不会因为一点故障导致全部瘫痪，是真正的分布式网络。用户 PC 性能及其与网络连接方式决定网络弹性和性能。这种模式具有自组织行为，降低了拥有者的成本，提供可扩展性。特别适合在自组织网上的应用，如即时通信等。但是这种结构使得每一个节点都必须同时拥有统一的节点分布图，系统的扩展性不好，同时协调节点之间通信也成为一个挑战。

以上三个系统代表了人们在分布式搜索引擎研究领域里的不懈努力，但也都有一定些弊端。未来的搜索引擎系统大都需要更强的计算力，更高的稳定性，更好的扩展性，同时也需要一定的策略将优化搜索结果，满足用户快速获取准确信息的需求[6,7,8]。本文所讨论的基于 Hadoop 的分布式搜索引擎系统顺应了这种发展趋势，利用 Map/Reduce 编程模式[9]设计了系统模块，部署在一个分布式平台上，充分利用已有的资源，组织更为庞大的计算网络，满足搜索引擎对计算能力、稳定性和扩展性的需求，并综合利用 TF-IDF 和 PageRank 评分策略优化搜索结果。

1.4 本文的研究内容及论文章节安排

本文的目标是分析分布式计算技术的优势，以及搜索引擎主要使用的技术，结合分布式计算和倒排文档全文检索技术，设计与实现一个分布式搜索引擎。其内容和结构安排如下：

第一章简要地阐述了课题研究背景，搜索引擎的发展历史，分布式搜索引擎研究现状方面的内容，分析现有系统的不足。

第二章从理论角度出发，分析了 Map/Reduce 编程模型，倒排文档全文检索机制，中文分词三方面的原理，以及它们的使用范围。并介绍了分布式搜索引擎系统使用的相关技术，包括 Hadoop 分布式计算平台，Lucene 全文检索框架，以及相关辅助工具。为后面的研究工作奠定了基础。

第三章阐述了分布式搜索引擎系统的设计思想。并依据业务功能，将系统划分为三大业务子系统：爬行子系统、索引子系统和查询子系统。对每个子系统的工作流程进行了详细分析，利用 Map/Reduce 模型改进这三个子系统，使它们具有分布式处理的能力。

第四章深入子系统内部，从功能方面分析了子系统设计过程中包含的主要模块，以及模块内部各个类之间的关系。运用成熟的开源工具实现了系统，并讨论了实现过程中的难点，及其解决方法。

第五章是实验平台搭建和实验数据分析。本章利用 4 台计算机建立了一个小型的分布式处理环境，在此基础上部署了分布式搜索引擎系统。经过多次实验，得出了实验结果，并对不同环境情况下取得数据进行分析，最终对系统做出总体评价。

第六章是总结与展望。对论文工作进行总结，并提出今后要做的工作和研究方向。

第二章 分布式搜索引擎相关技术分析

分布式搜索引擎是以分布式计算技术和倒排文档全文索引技术相结合构建的庞大而复杂的系统，因此有必要对这两种技术进行详细分析。此外，在设计与实现过程中，还使用了许多新兴的技术，在这一章里也做了简单的介绍。

2.1 分布式计算技术

2.1.1 Map/Reduce 编程模型

在程序设计中，一项工作往往可以被拆分成为多个任务，任务之间的关系可以分为两种：一种是任务之间有相互的依赖，先后顺序不能够颠倒，这类任务是无法并行处理的；另一种是不相关的任务，可以并行执行。绝大部分大型计算工作都是属于后者。只不过，输入的数据通常是非常巨大的，并且为了能在合理时间内执行完毕，其计算任务必须分布到上百个或者上千个计算机上去执行。如何并发计算，如何分布数据，如何处理失败等等相关问题合并在一起就会导致原本简单的计算掩埋在为了解决这些问题而引入的很复杂的代码中[9]。Map/Reduce 编程模型正是为了解决这些问题而发明的。

Map/Reduce[10]是 Google 实验室于 2004 年提出的一种新的分布式程序设计模型，用于在集群上对海量数据进行并行处理。在 Google 内部，Map/Reduced 的应用非常广泛包括：分布 grep，分布排序，web 连接图反转，每台机器的词矢量，web 访问日志分析，反向索引构建，文档聚类，机器学习，基于统计的机器翻译等。值得注意的是，Map/Reduce 实现以后，它被用来重新生成 Google 的整个索引，以取代老的 ad hoc 程序去更新索引，足以证明其具有非常高的性能。

Map/Reduce 的名字源于这个模型中的两项核心操作：Map 和 Reduce，实际处理的都是大量像<key, value>对这样的简单数据类型。在模型中首先对输入的数据进行分割，将分割后的数据分配给 Map 函数，而 Map 把分配到的数据（一般为一组<key, vaule>对）映射为另外的一组<key2, vaule2>型中间数据，其映射的规则由一个函数指定；Reduce 是对 Map 输出的<key2, vaule2>型中间数据进行归约并输出最终结果，这个归约的规则也是由一个函数指定。这两项操作的规则是可以由程序设计人员的指定，正是这一点带给了 Map/Reduce 模型巨大的灵活性[11]。执行流程如图 2-1 所示。

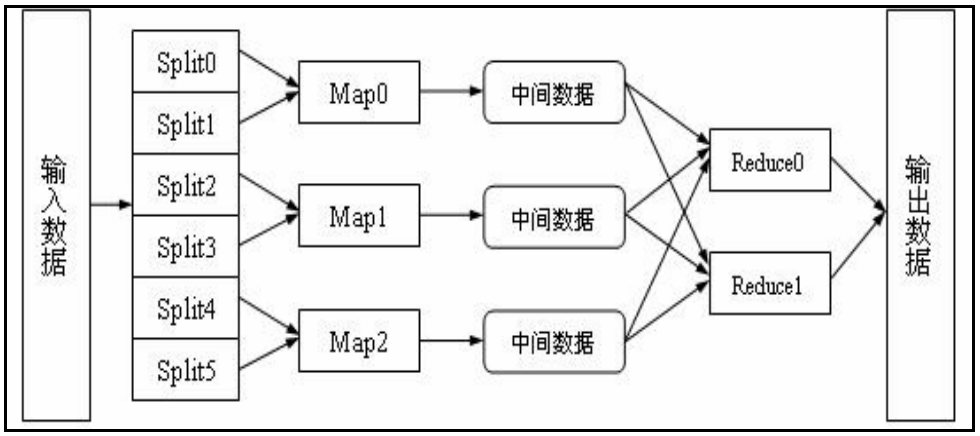


图 2-1 Map/Reduce 模型执行流程图

Figure 2-1 Map/Reduce Module Flow Chart

使用这样的函数形式实现的程序可以自动分布到一个由普通机器组成的大型集群上并发执行。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，并且管理机器之间的通讯请求。这样的模式允许程序员可以不需要有什么并发处理或者分布式系统的经验，就可以充分利用分布式系统的资源[12,13]。

2.1.2 Hadoop 平台简介

Hadoop 是一个基于 Map/Reduce 的相当成功的分布式计算平台，Hadoop 由 Apache Software Foundation 组织于 2005 年秋天正式引入，目前的最新版本是 0.19.1。Map/Reduce 是 Google 的一项重要技术，是一种简化并行计算的编程模型，它让那些没有多少并行计算经验的开发人员也可以开发并行应用。Map/Recuce 可以将被划分成多个小的 Block 的海量检索数据以分布式的方法局部计算，并应用 Map 将他们映射到一个提供 Reduce 的中心上，从而达到快速处理海量数据检索的目的。

分布式文件系统 HDFS（Hadoop Distributed File System）是 Hadoop 的另一重要功能。HDFS 是受 Google 文件系统的启发，建立在大型集群上可靠存储大数据集的文件系统。HDFS 与 Map/Reduce 紧密集成，是 Hadoop 分布式计算的存储基石。它有自己明确的设计目标，那就是支持大的数据文件（大至 T 级），并且这些文件以顺序读取为主，以文件读的高吞吐量为目标[14]。

Hadoop 是一个能够对大量数据进行分布式处理的软件平台，是以一种可靠、高效、可扩展的软件框架。Hadoop 可靠性体现在它假设计算元素和存储会失败，因此它维护多个工作数据副本，确保能够针对失败的节点重新分布处理。Hadoop 的高效性体现在它以并行的方式工作，通过并行处理加快处理速度。Hadoop 还是可扩展的，能够处理

PB 级数据。此外，Hadoop 依赖于低端服务器甚至是普通计算机，因此它的成本比较低，任何人都可以使用，是作为分布式搜索引擎的理想平台。

目前，经过多个版本的发展，Hadoop 已经成为一个庞大的平台，包含 MapReduce、HDFS、Common、Avro、Chukwa、HBase、Hive、Pig、ZooKeeper 等许多模块，系统核心主要由两大元素构成：最底部的 Hadoop Distributed File System（HDFS 分布式文件系统），以及进行分布式计算的 Map/Reduce 引擎[15]，下面详细分析这两个模块。

2.1.3 HDFS 分布式文件系统

HDFS 是分布式计算的存储基石，它存储 Hadoop 集群中所有的数据文件。它采用了 Master/Slave 结构，由一个 NameNode，一个 NameNode 的备份（Secondary NameNode）和多个 DataNode 组成。NameNode 作为文件系统中的管理者，主要负责管理文件系统的命名空间并协调用户对文件的访问。它会将文件系统的描述性元数据保存为列表存储在内存中，供用户快速访问。DataNode 是数据的实际存储者，它将本地磁盘划分为多个块（Block）来存储数据，并在内存中保存了这些块的元数据，同时周期性的向 NameNode 报告。NameNode 则将需要存储的数据分割为多个片段并存储到这些块中。

HDFS 文件系统使用副本存储策略来实现高可靠性。系统的复制因子一般为 3，这意味着同一时间每个块都会有 3 个副本，分别位于 3 个 DataNode 上，其中一个位于不同机架的 DataNode 上。这些块的元数据都被注册在了 NameNode 上。当一个 DataNode 出现故障后，其上保存的块仍然可以通过 NameNode 上注册的冗余块进行读取。NameNode 周期性的收到来自集群内 DataNode 的心跳报告，能收到心跳证明 DataNode 工作是正常的。如果 NameNode 在指定的时间段内都没有收到一个 DataNode 的心跳报告，则说明此 DataNode 已经出现故障，这时 NameNode 把应由此 DataNode 保存的块的副本复制并存储到其他 DataNode 上，时刻保持系统中每个块都会有 3 个副本，以此来保证系统的高可靠性。

HDFS 分布式文件系统数据的读取和存储机制与一般文件系统有一些区别。当用户需要读取文件系统中的文件时，首先用户必须向 NameNode 提交读取请求，NameNode 查询元数据表后将文件的元数据（此文件分为几个块，每个块属于哪个 DataNode 等）返回给用户，最后中断与用户的连接。接下来用户直接访问相关 DataNode 获得所需块，得到完整的文件。而当用户需要保存文件时，同样必须首先向 NameNode 提交保存请求，NameNode 在文件命名空间中写入文件名，根据文件大小将文件分割为许多片段，并查

询元数据表为文件分配空闲块，最后将相关元数据 （此文件共需几个空闲块，每个块属于哪个 DataNode 等）返回给用户后中断与用户的连接，接着用户直接与相关 DataNode 建立连接，获得块的写入权限将文件写入到块中[16]。分布式文件系统结构如图 2-2 所示：

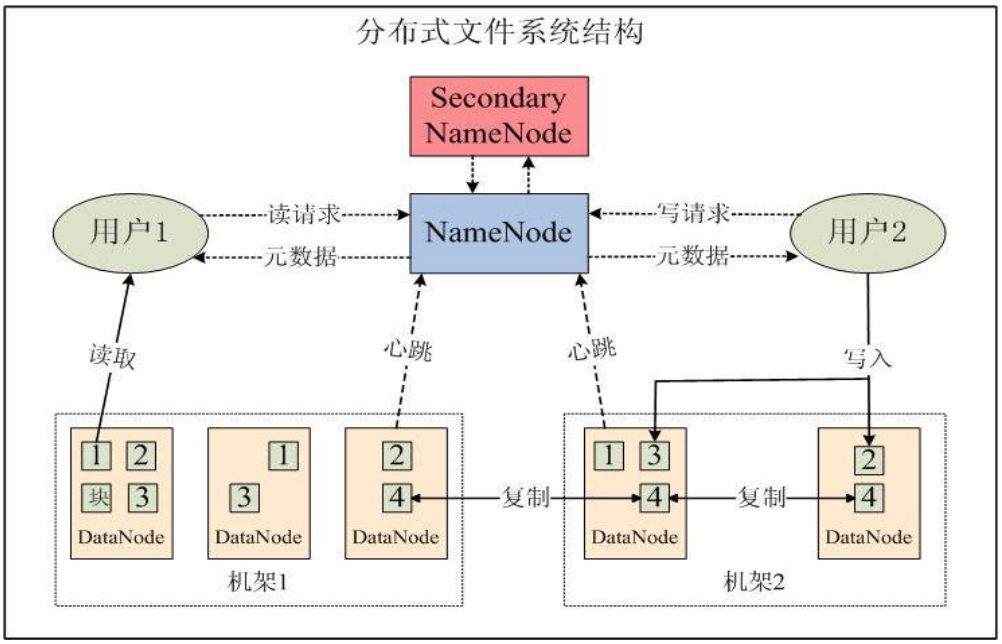


图 2-2 HDFS 分布式文件系统结构

Figure 2-2 HDFS Distributed File System Architecture

2.1.4 Hadoop 分布式计算引擎

分布式计算的 Map/Reduce 引擎也采用 Master/Slave 结构，由一个 JobTracker 和多个 TaskTracker 组成。JobTracker 通过 RPC 调度作业的每一个子任务（task）运行于 TaskTracker 上，JobTracker 则通过 TaskTracker 监控子任务的运行状态，如果发现有失败的子任务就重新运行它。TaskTracker 必须与 DataNode 部署在同一台计算机上，这是因为 Hadoop 的 Map/Reduce 引擎遵循了“移动计算比移动数据更经济”的原则[17]：数据存储在哪一台计算机上，就由这台计算机进行这部分数据的计算。这样可以减少数据在网络上的传输，降低对网络带宽的需求。Hadoop 还提供了一系列 API，使得用户可以很方便地调用系统功能，扩展系统应用。

综合 Map/Reduce 和 HDFS，就是 Hadoop 平台的整体结构了。在 Hadoop 分布式计算平台中，必须有一个 Master 节点，主要负责 NameNode 和 JobTracker 的工作，如果是大型分布集群环境，更理想的方式是将 NameNode 和 JobTracker 分别部署到由两台计算机共同组成 Master 节点，这样可以提高平台的响应速度。此外，会有多个 Slave 节点，每个 Slave 节点通常具有 DataNode 的功能并负责 TaskTracker 的工作[18]。综合

Map/Reduce 和 HDFS 的 Hadoop 整体结构如图 2-3 所示：

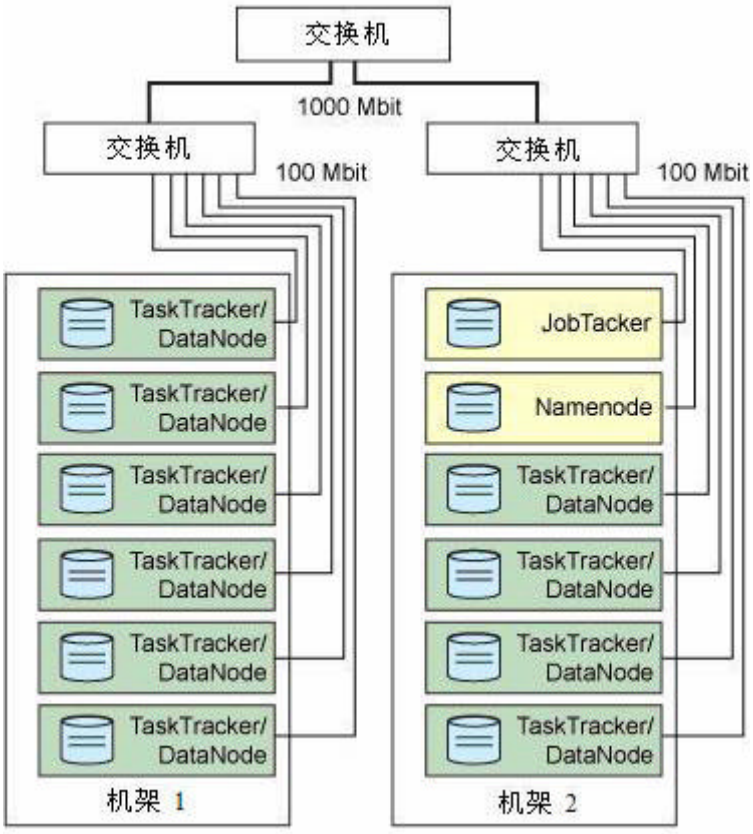


图 2-3 Hadoop 整体结构

Figure 2-3 Hadoop Architecture

2.2 倒排文档全文检索技术

2.2.1 索引技术

搜索引擎主要采用全文检索机制搜索文档库，响应用户搜索请求。所谓全文检索，就是给定一个字符串或字符串逻辑表达式，对文档库进行相应的检索，查找出包含指定字符串，或与指定表达式相匹配的所有文档。检索过程中主要是对两种类型的文档进行检索：正排文档（普通文档）和倒排文档（索引文档）。正排文档，顾名思义就是人们在日常生活中根据写作逻辑来记录信息的文档，包括文本文档、数据文档和多媒体文档等等。而倒排文档与人们日常所认识的正排文档有很大区别，但是在全文检索领域却扮演着极为重要的角色。

2.2.2 倒排文档索引原理

在全文检索系统的具体实现中，需要根据关键词快速地找到相关文档。早期的全文

检索系统是通过在正排文档中逐个扫描字符，或者采用正则表达式匹配关键字来完成检索，这种检索方式实现简单，无需任何额外数据。但是随着文档数量的增加，这种检索方式因其需要对所有的文档进行全文扫描，所以效率很低，不适合搜索引擎这种数据量大、对检索速度要求高的应用。

为了改进检索速度与质量，人们进行许多研究，也产生了多种高效的检索方法，其中得到广泛应用的是倒排文档检索方法。这种检索方法是受图书馆书目索引的启发而提出的。倒排文档是对已有正排文档的信息加工后生成的索引文件，相比正排文档来说，倒排文档在建立的过程会对词语进行过滤，那些没有实际意义的词语，如英语的“ is ”、“ a ”，中文的“ 是 ”、“ 的 ”等，都会被忽略掉，而那些有明确意义的词语，称为“ 关键词 ”，被单独提取出来建立索引。

倒排文档可以被看成一个链表数组，每个链表的表头包含关键词，其后续单元则包括所有包含这个关键词的文档标号，以及一些其他信息。这些信息可以是文档中该关键词的频率，也可以是文档中该关键词的位置等信息[19-20]，如图 2-4 所示。

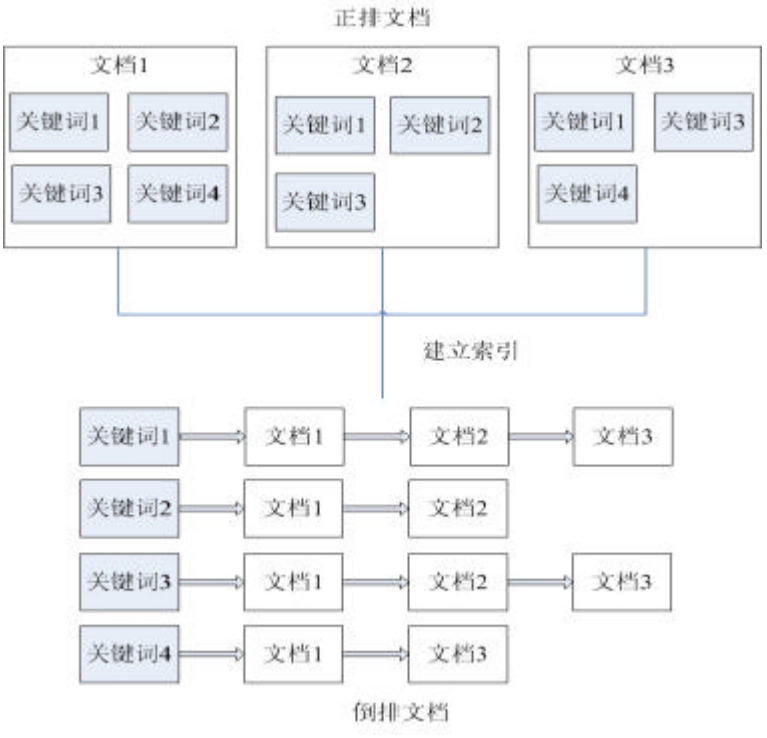


图 2-4 正排文档与倒排文档的对比

Figure 2-4 The comparison of Common Document and Inverted Document

倒排文档中关键词的个数是较少的，因此，以关键词为核心对文档进行索引是更加可行的方法。倒排文档检索的优势不仅在于关键词个数少导致检索效率的提高，还在于其特别易于同信息检索技术结合。在实际应用中，用户查询输入的查询信息中所包含的

关键词往往是很少的，完全不包含任一关键词的文档，是不会被列入结果集的。因此，以关键词为主键进行检索，只需要用查询信息中包括的关键词，进行简单的检索就能够找出所有相关的文档。

倒排文档索引的具体数据结构可以进行进一步的优化。在关键词查询上，往往采用B-Tree或哈希表进行快速查询。而文档列表的数据结构则可以采用简单的无序列表进行存储，但是此种无序列表存在一个问题，就是当多个关键词对应的文档集需要进行比较的时候，比较效率将会很低。因此，在实际应用中往往采用二叉搜索树组织文档列表。

2.2.3 Lucene 框架结构

Lucene 是 Apache Jakarta 家族中的一个开源项目，是目前最为流行的全文信息检索框架。Lucene 并不是一个完整的搜索应用程序，只是一个全文检索引擎的框架，它提供了完整的索引和检索功能，为数据访问和管理提供了简单的函数接口，从而可以方便地嵌入到各种具体的应用中实现全文检索功能[21]。Lucene 具有以下这些特点：能够处理任何类型的文本数据，如 Word、Excel、PDF、数据库和 HTML 文档通过解析器转换成标准的中间格式（如 XML）就可以进行数据索引；可通过对其语言词法分析接口进行扩展实现对中文检索的支持；通过建立倒排索引，并可以进行增量的索引，提高索引效率；对索引内容可以控制哪些字段需要索引，哪些字段不需索引，索引的字段可分为需要分词（如标题、文章内容等）与不需要分词（如作者、日期等）的类型；通过查询分析接口的实现，可以制定自己的查询语法规则；能够支持多用户并发访问。

使用 Luncene 框架进行搜索引擎的开发，除了利用 Lucene 的索引机制外，简单、高效地实现搜索功能是另外一个因素。其优点是 Lucene 能够对所有能被转化为纯文本的文档文件进行索引与检索，而不关心数据的来源、格式以及语言等细节。目前，Lucene 已经得到了广泛应用，不仅发展出了比较完整的全文检索系统，而且在许多系统应用软件、Web 应用及商业软件中都使用了 Lucene 作为核心的全文检索引擎。Lucene 已经成为当前最为流行的开源全文检索工具。

Lucene 选择面向对象的方法将搜索引擎的架构模块化，可以选择多种语言来实现这个搜索引擎而不必改变整体架构。Lucene 源码主要包括 7 个包，每个包完成特定的功能，如表 2-1 所示：

表 2-1 Lucene 包结构功能

Table 2-1 Lucene Package Architecture

Lucene 包结构功能表	
包名	功能
org.apache.lucene.analysis	语言分析器，主要用于分词，支持中文主要是扩展此类
org.apache.lucene.document	索引存储时的文档结构管理，类似于关系型数据库的表结构
org.apache.lucene.index	索引管理，包括索引建立、删除等
org.apache.lucene.queryParser	查询分析器，实现查询关键词间的运算，如与、或、非等
org.apache.lucene.search	检索管理，根据查询条件，检索得到结果
org.apache.lucene.store	数据存储管理，主要包括一些底层的 I/O 操作
org.apache.lucene.util	一些公用类

Lucene 功能模块众多，各个模块均是符合 MDA 规则的，利于用户选择有效的接口，也为用户定制自己所需的功能模块提供了高扩展性。各个模块之间保持固定的协议，将实现方法隐藏起来。从根本上说，主要包括两大功能：一是文本内容经分词后索引入库；二是根据查询条件返回结果。图 2-5 是上述两大功能的逻辑图。

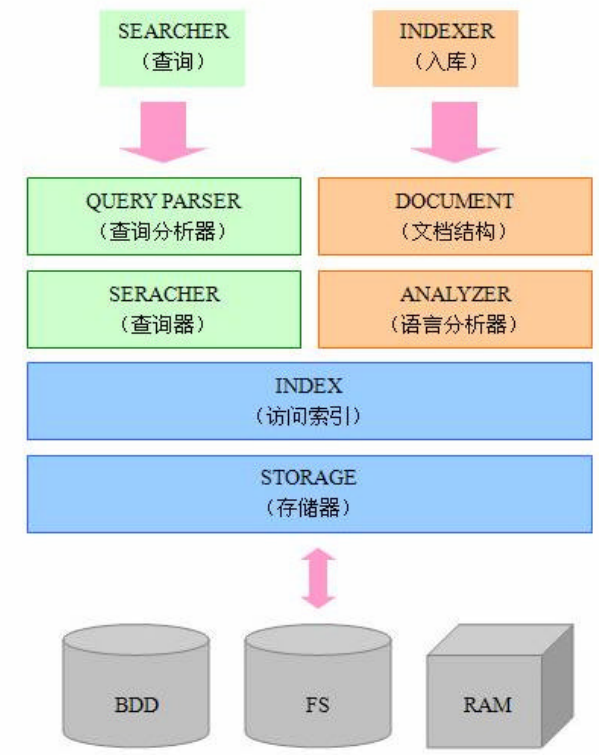


图 2-5 Lucene 逻辑结构

Figure 2-5 Lucene Logical Architecture

从图 2-5 可以了解到，Lucene 总体上可分为 5 个功能模块，分别是：语言解析（Analyzer）、索引（Index）、查询分析（Query Parser）、搜索（Searcher）、存储（Storage）。每一模块从逻辑上又可以再划分为 2 部分：交互协议部分与具体实现部分。

语言解析模块为索引模块提供了解析功能。Lucene 建立索引过程中，需要对被索引的文档进行分析，根据语种对文档分词。

索引模块提供 2 种访问协议。一种为搜索提供访问，另一种协议为维护索引提供服务。使用 Lucene 创建索引时，需要对文本内容建立索引；而对索引维护也可以通过该接口访问索引，更新索引信息，优化索引。

查询分析对于搜索过程而言，用户输入的查询条件，需要通过解析，用户的查询信息才能和从文本解析出的信息相匹配，才能返回给用户正确的结果。

搜索模块是用户和 Lucene 交互的一个窗口。用户输入的内容都通过该模块进入到 Lucene 的内部，经过查询分析模块解析，然后通过搜索模块进行检索，最后返回给用户相关信息，完成查询工作。

存储模块为索引的存储提供了多种便利方式。一种将索引文档存放在内存中，一种将索引文档存储在物理磁盘中，另外也可以将索引文档存储到数据库中。其中需要注意的是，RAM 存储接口批量地索引文件，加快标引速度，但是并没有永久性存储，系统重启后将丢失，所以 RAM 只适合临时存储索引文件。

2.2.4 Lucene 索引机制

Lucene 采用了段索引（segment index）与倒排文档索引相结合的方式来完成索引过程，其中段索引方式是增量产生索引的一种。因为 Lucene 使用倒排文档索引作为基础索引结构，在建立索引的过程中，还需要考虑索引更新这一值得研究的问题，索引更新涉及以下几个问题：内存中保存的索引与磁盘中保存的索引，二者结构是否相同，如果相同，就可以在内存中批量查询与标引；是每添加记录就更新索引，还是将添加的索引与原有索引都保持相同的结构，存放在不同位置，不立即更新索引，而可以分别访问。对于单计算机系统而言，索引更新后会合并为一个索引文件，磁盘中文件的数目会少一些，查询过程中打开文件的操作也会减少，这样可缩短查询响应时间，但是频繁的更新索引（在更新时，索引文件系统进入锁定状态暂停查询，防止对索引的读写冲突）会使得系统的实时性能下降，所以选择适当的索引更新时机很重要。对于分布式的索引文件系统而言，更新过程需要考虑的情况也会更加的复杂。

为了解决以上问题，在 Lucene 中采用了基于段索引的增量索引生成方式，其中合并阈值影响着内存与硬盘中索引文件的个数，每添加一个 document 将生成单一段索引（single segment index）并临时保存在内存中，当单一段索引的个数超过合并阈值后，

就会触发段合并操作，将单一段索引合并为段索引（segment index）。段索引的个数超过合并阈值时也会触发合并过程，合并为一个整体索引。单一段索引，段索引，整体索引文件格式是相同的，三者不同的地方只是规模。三者关系如图 2-6 所示。

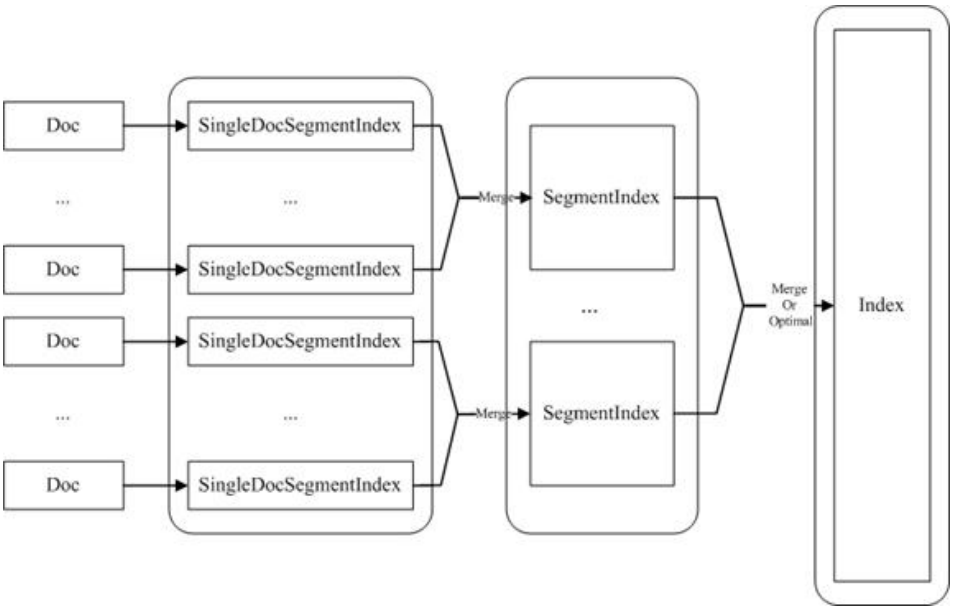


图 2-6 增量索引过程

Figure 2-6 Increamental Indexing Procedure

索引合并阈值的大小涉及多个问题：如果没有达到阈值，各段索引并不合并；如果这个阈值比较小，在生成索引时使用的内存比较小，查询（Lucene 的优化过程也合并索引）索引较快（索引文件个数少），但是生成索引的速度慢（频繁的合并索引，浪费时间）；如果设定了较大的阈值，查询速度慢（文件数量多，I/O 操作多），但是生成索引的速度快，因为不是频繁的合并索引，而是在内存中大批量合并，在 Lucene 中这个阈值默认设定为 10，阈值可以根据实际需要更改但必须大于 2。

2.2.5 Lucene 文件系统

为了对文档进行索引，Lucene 提供了五个基础的类，它们分别是 Document, Field, IndexWriter, Analyzer, Directory，这五个类是索引功能的基础，分别包装了不同的数据结构，它们协同工作就可以生成索引文档了，生成的索引文档最终是要保存到系统索引库中。Lucene 根据功能将索引划分为几个独立的文件单独保存，各个文件的具体用途如表 2-2 所示：

表 2-2 索引文件含义

Table 2-2 Index File meaning

索引文件	索引文件含义
.f(n)	规格化文件。
.fdt	包含各个域数据（field 的特性）信息
.fdx	它是指向.fdt 文件的指针。填写的是.fdt 文件中每个文档的填写 field 信息的起始位置
.fnm	各个域的名字信息。
.frq	词元（term）的频率信息
.prx	term 在文档中的位置信息
.tis	包含 term 数据信息，指向位置文件与频率文件的指针。
.tii	是.tis 文件的快表，可以快速定位.tis 文件中 term 数据信息。
.tvd	保存有 document 信息，用词元向量（TermVector）方式保存 field 的信息，同它包含一个指针表，表内的指针指向.tvf 文件中的 field 信息。
.tvf	以 TermVector 方式保存 field 数据信息。
.tvx	是.tvd 的索引文件，保存了指向.tvd 指针信息。
deletable	包含要删除的文档信息
Segments_N 与 segments.gen	保存了相关段的信息。

下面着重介绍一下 Document, Field, IndexWriter, Analyzer, Directory 这五个类在索引过程中的用途：

Document 是用来描述文档的，这里的文档可以指一个 HTML 网页，一封电子邮件，或者是一个文本文件。一个 Document 对象由多个 Field 对象组成的。可以把一个 Document 对象想象成数据库中的一个记录，而每个 Field 对象就是记录的一个字段。

Field 对象是用来描述一个文档的某个属性的，比如一封电子邮件的标题和内容可以用两个 Field 对象分别描述。

Analyzer 对象，在一个文档被索引之前，首先需要对文档内容进行分词处理，这部分工作就是由 Analyzer 来完成的。Analyzer 类是一个抽象类，它有多个实现。针对不同的语言和应用需要选择适合的 Analyzer。Analyzer 把分词后的内容交给 IndexWriter 来建立索引。

IndexWriter 是 Lucene 用来创建索引的一个核心的类，他的作用是把一个个的 Document 对象加到索引中来。

Directory 这个类代表了 Lucene 的索引的存储的位置，这是一个抽象类，它目前有两个实现，第一个是 FSDirectory，它表示一个存储在文件系统中的索引的位置。第二个是 RAMDirectory，它表示一个存储在内存当中的索引的位置。

2.3 中文分词

索引系统的核心是建立有检索意义的关键词的索引。以词为单位建立索引需要对原始文档进行分词处理，这种索引技术最早考虑的是处理英文信息，没有考虑中文语境。英文分词相对而言是比较简单的，因为英文单词虽有大小写之分，但都是由空格和标点符号隔开的。因此在对英文文档进行分词处理时，标准分词模块在扫描文档过程中，遇到字母首先要将字母转换成小写暂存在数据栈中，遇到一个空格或标点符号后返回栈内所有字母，这些字母就构成了一个单词。然而在切换到中文语境后，最大问题是中文词语并没有空格作为分隔，所以这种英文分词方法不能用于中文。中文分词被认为是中文自然语言处理中的一个最基本的环节。

2.3.1 中文分词方法

中文文档的基本语义单位汉字语词之间没有明显的间隔，对于一段中文文章，人们可以通过自己的知识来明白哪些是词，哪些不是词，但如何让计算机也能理解，这一难题从计算机发明以来，一直困扰着我国广大研究人员。因此，从计算机引入我国开始，一直有大量研究人员致力于中文分词的研究，如今已取得了丰硕的成果，多种分词技术已投入使用。但是，分词技术作为一种复杂的自然语言处理技术，仍然有很多难以逾越的技术鸿沟，如专有名词和复合词的切分、多音字词的区分，以及汉语成语的切分等。汉语词汇的构成方式非常复杂，字与字之间的组合方式灵活多变，很难找出稳定的规律。目前在中文分词领域，主要采用了三种分词方法：单字切分法、二元切分法和智能分词法[22]。

（1）单字切分法

由于中文句子可以看做是由独立的汉字组成的，某种程度上可以采用类似处理英文单词的方法处理每一个汉字。单字切分法的分割单位是每个汉字，这样只需根据汉字编码，扫描文档提取存储汉字的字节即可，实现起来比较简单，而且可以避开诸多分词难点。然而，这并不是一种好的分词方法，因为依此法分词分割出来的汉字，数量庞大（与文章的长度一致），建立的索引也相应的非常巨大，检索效率非常低。此外，没有体现出汉字之间的关系，严重影响检索的准确率。现在已经基本退出历史舞台了。

（2）二元切分法

二元切分法是对单字切分法的一种改进，它按顺序将相邻的两个汉字进行切分，每

切分一次就会回退一个汉字，继续切分。因为在汉语语境下，大量词汇是由两个汉字构成的，这种分词方法，能在一定程度上改善分词结果，因此在很长一段时间里，人们一直使用它作为主要的中文分词方式，直到今日仍有部分系统采用。但是，与单字切分法类似，它完全没有考虑到词义、语境，只是一味机械地对文章进行切分处理，也具有单字切分法相同的缺点。

（3）智能分词

智能分词是以能表达一定意义的词为基本检索单位，并根据词的出现位置进行索引和检索的中文分词方法。通常使用的是基于字符串匹配的分词方法，它是按照一定的策略将待分析的汉语字符串与一个"充分大的"词典中的词条进行配，若在词典中找到某个字符串，则匹配成功（识别出一个词）。匹配方式如下：

（1）正向最大匹配法（由左到右的方向）；

（2）逆向最大匹配法（由右到左的方向）；

（3）最少切分（使每一句中切出的词数最小）。

实际使用中，一般综合使用上述三种匹配方式，结合分词与标注一体化方法，还需通过利用各种其它的语言信息来进一步提高切分的准确率。

本文所设计系统在中科院智能分词 ICTCLAS 框架的基础上设计实现了采用正向最大匹配算法的中文分词模块。

2.3.2 中文分词的难点

中文分词虽然已有了长足进步，但是有两大难题一直没有完全突破。

1．歧义识别

歧义是指同样的一句话，可能有两种或者更多的切分方法。例如："互联网"，因为"互联"和"联网"都是词，那么这个短语就可以分成"互联 网"和"互 联网"。这种称为交叉歧义。像这种交叉歧义十分常见，由于没有人的知识去理解，计算机很难知道到底哪个方案正确。此外组合歧义、真歧义等也都难以解决。

2、新词识别

新词，专业术语称为未登录词。也就是那些在词典中都没有收录过，但又确实能称为词的那些词。最典型的是网络用语，因为目前网络上每时每刻都有新的网络用语出现，收录这些词语本身就是一项巨大的工程。然而，如果不能正确识别这些新词的话，用户就无法得到正确的检索结果。因此对于搜索引擎来说，分词系统中的新词识别十分重要。

目前新词识别准确率已经成为评价一个分词系统好坏的重要标志之一。

2.4 统一资源定位符

统一资源定位符（URL，Uniform / Universal Resource Locator 的缩写）也被称为网页地址、网页链接（在本文中统一使用“网页链接”这个术语）。是因特网上标准的资源的地址（Address)。它最初是由蒂姆·伯纳斯-李发明用来作为万维网的地址的。现在它已经被万维网联盟编制为因特网标准 RFC1738 了。

在因特网的历史上，统一资源定位符的发明是一个非常基础的步骤。统一资源定位符的语法是一般的，可扩展的，它使用 ASCII 代码的一部分来表示因特网的地址。统一资源定位符的开始，一般會标志着一个计算机网络所使用的网络协议。统一资源定位符的标准格式如下：

协议类型://服务器地址（必要时需加上端口号）/路径/文件名

对于搜索引擎而言，网页链接就像是“网页的门牌号”，它是网页的唯一标识，搜索引擎内部只依靠网页链接来区分不同的网页数据。

2.5 MD5 算法

Message Digest Algorithm MD5（消息摘要算法第五版）为计算机安全领域广泛使用的一种散列函数，用以提供消息的完整性保护。该算法的文件号为 RFC 1321。MD5 是 90 年代初由 MIT Laboratory for Computer Science 和 RSA Data Security Inc 的 Ronald L. Rivest 开发出来，经 MD2、MD3 和 MD4 发展而来。它的作用是让大容量信息在用数字签名软件签署私人密钥前被"压缩"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的大整数)。

对 MD5 算法简要的叙述可以为：MD5 以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，经过了一系列的处理后，算法的输出由四个 32 位分组组成，将这四个 32 位分组级联后将生成一个 128 位散列值。

在 MD5 算法中，首先需要对信息进行填充，使其位长对 512 求余的结果等于 448。因此，信息的位长（Bits Length）将被扩展至 N*512+448，即 N*64+56 个字节（Bytes)，N 为一个正整数。填充的方法如下，在信息的后面填充一个 1 和无数个 0，直到满足上面的条件时才停止用 0 对信息的填充。然后，在在这个结果后面附加一个以 64 位二进

制表示的填充前信息长度。经过这两步的处理，现在的信息的位长为N*512+448+64=(N+1)*512，即长度恰好是 512 的整数倍。这样做的原因是为满足后续处理中对信息长度的要求。

MD5 的典型应用是对一段信息（Message）产生信息摘要（Message-Digest），以防止被篡改。MD5 将整个文件当作一个大文本信息，通过其不可逆的字符串变换算法，产生了一个唯一的 MD5 信息摘要。MD5 算法在本文所讨论的系统中得到了广泛的应用。网页链接签名和去重、网页内容签名和去重都使用了 MD5 算法。

2.6 Robots 协议

网络爬行器抓取网页的过程，不同于一般用户浏览网页，如果控制不好，则有可能造成短时间内网站服务器访问量过大，严重影响服务器性能，甚至导致网站服务器崩溃。淘宝网（http://ww.taobao.com）就曾因为雅虎搜索引擎的爬行器抓取其数据引起淘宝网服务器的不稳定。为了避免这种情况的发生，网站需要有一种机制，来告诉爬行器哪些网页不可以抓取，哪些网页应该抓取，这就是 Robots 协议要解决的问题[23]。

目前各大搜索引擎的爬行器都拥有一个独一无二的名字，用来区分彼此，网站也可以利用这些名字控制某些爬行器的访问权限。常见搜索引擎爬行器的名字有：Google 爬行器：googlebot；百度爬行器：baiduspider；Yahoo 爬行器：slurp；Alexa 爬行器：ia_archiver；MSN 爬行器：msnbot；Altavista 爬行器：scooter；Lycos 爬行器：lycos_spider_(t-rex)；Alltheweb 爬行器：fast-webcrawler；Inktomi 爬行器：slurp 等。

爬行器进入一个网站，一般会首先访问一个特殊的文本文件 Robots.txt，这个文件一般放在网站服务器的根目录下，如 http://www.tyut.edu.cn/robots.txt 这个地址。网站管理员可以通过 Robots.txt 来规定哪些目录爬行器不能访问，或者哪些目录对于某些特定的爬行器不能访问。例如有些网站的注册用户文件目录和保密的文件目录是不希望被爬行器抓取到的，那么网站管理员就可以把这些目录规定为拒绝访问目录。Robots.txt 语法简单实用，规则如下：

1. 网站对所有目录和爬行器都没有任何限制，可以设置为：

User-agent:*

Disallow:

或者创建一个空的 Robots.txt 文件。

2. 网站不允许爬行器抓取整个网站的内容，可以这样定义：

User-agent:*

Disallow:/

3. 如果需要禁止所有的爬行器访问网站的部分目录：

User-agent:*

Disallow:/private/

Disallow:/tmp/

4. 如果需要禁止某个爬行器的访问权限（如 Google 爬行器）：

User-agent: googlebot

Disallow:/

5．如果只允许一个爬行器的访问权限（如 Google 爬行器）：

User-agent: googlebot

Disallow :

User-agent: *

Disallow :/

另一种影响搜索引擎行为的方法是使用 Robots 元数据。它采用如下格式：

<meta name="robots" content="noindex,nofollow" />

Robots 元数据标签中没有大小写之分，name="Robots"表示所有的搜索引擎，可以针对某个具体搜索引擎写为 name="googlebot"。content 部分有四个指令选项：index、noindex、follow、nofollow，指令间以“,”分隔。index 指令告诉爬行器应该抓取该网页，noindex 作用相反；follow 指令告知爬行器可以沿着该网页上的链接继续抓取下去，nofollow 作用相反。Robots Meta 标签的缺省值是 index 和 follow。

需要注意的是，Robots 协议并不是一个规范，而只是约定俗成的，所以并不能保证网站的隐私。如果爬行器的设计者不遵循这个协议，网站管理员也无法阻止爬行器对于某些网页的访问，但一般的爬行器都会遵循这些协议。本文所讨论的爬行器也充分考虑了这些因素，设计为遵守 Robots.txt 和 Robots META 标签的协议，并且起名为 CienBot。

2.7 相关工具软件

2.7.1 HttpClient

HTTP 协议是爬行器使用得最多、最重要的协议了，虽然在 JDK 的 java.net 包中

已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，JDK 库本身提供的功能还不够丰富和灵活。HttpClient 是 Apache Jakarta Common 下的子项目，用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。它有如下几个优点：实现了所有 HTTP 的方法（GET，POST，PUT，HEAD 等）；支持自动转向；支持 HTTPS 协议；支持代理服务器等。

2.7.2 HTMLParser

HTMLParser 是一个用于解析 HTML 网页的开源项目，它可以准确高效的对 HTML 网页中的格式、数据进行处理。HTMLParser 在系统主要执行两种任务：

1. 信息提取：文本信息提取，对 HTML 进行分析，过滤 HTML 网页中的标签和脚本，将网页中的有效文本信息提取出来；多媒体信息提取，例如对一些图片、声音信息进行提取。

2. 链接提取，用于将网页中的超链接提取出来，扩展网页链接库，提供给爬行器获取网页。

2.7.3 Log4j

Log4j 是 Apache 的一个开放源代码日志软件包。通过使用 Log4j，可以记录系统执行过程信息，包括执行流程、警告信息、错误信息和时间信息等。也可以控制每一条日志的输出格式，通过定义每一条日志信息的级别，能够更加细致地控制日志的生成过程，通过使用各种第三方扩展，可以很方便地将 Log4j 集成到我们的系统中，方便我们监控系统运行状况。

2.7.4 Cygwin

Cygwin 是许多自由软件的集合，最初由 Cygnus Solutions 开发，用于各种版本的 Microsoft Windows 上，运行 UNIX 类系统。Cygwin 的主要目的是通过重新编译，将 POSIX 系统（例如 Linux、BSD，以及其他 Unix 系统）上的软件移植到 Windows 上。

由于我们的系统使用 Windows 作为开发环境，所以必须利用 Cygwin 模拟 Linux 或 Unix 环境和 Shell。

2.8 本章小结

本章主要介绍了分布式搜索引擎相关的技术，首先，介绍了一种新型分布式计算模型 Map/Reduce，以及其开源实现平台 Hadoop 的系统结构。其次，介绍了全文检索技术和 Lucene 框架，及相关中文分词技术。最后，简要介绍了系统相关工具软件。

第三章 分布式搜索引擎的设计

本章阐述了分布式搜索引擎的基本功能和设计思想。从业务功能角度来看，系统可以划分为三个相对独立的子系统，分别完成爬行、索引和查询功能，三者整合在一起就构成了分布式搜索引擎系统。每个子系统都有一部分工作是可以无序执行的，因而可以应用 Map/Reduce 模型来设计，这也是本章重点讨论的内容。

3.1 系统总体结构

本文提出的基于 Hadoop 的分布式搜索引擎系统，将 Map/Reduce 技术应用到搜索引擎的工作流程中，应用 HDFS 分布式文件系统存储数据。按照搜索引擎的特点，可以将系统分为三个业务功能子系统包括：爬行器 Crawler、索引器 Indexer、查询器 Searcher。这三个子系统应用了 Map/Reduce 编程模型，运行于分布式系统环境。此外，为了便于在业务子系统中使用某些与业务无关的功能，还将这些功能封装在辅助子系统中，包括：分布式文件管理子系统，插件和属性管理子系统。本章主要讨论三个业务子系统的主要功能，如何对业务子系统进行分布式设计，以及设计过程中遇到问题和解决方案。

本文讨论的系统在设计时，采用了面向对象的设计模式，各个子系统将各自的实现细节完整的封装在内部，在子系统之间提供了相应的接口。

爬行器主要是使用 Http 协议针对具体的应用提供数据采集功能，比如新闻、论坛、博客等，此子系统包含了信息采集、链接提取、文本信息的存储等功能。

索引器完成多项工作。如对网页数据库信息进行分词，根据评分策略给网页评分，并为网页信息建立全文索引[24]，最终存储于分布式文件系统中。

查询器提供查询功能和用户界面，负责接受用户查询请求、查询倒排索引、给用户返回查询结果的列表[25]。

分布式文件系统子系统，保证数据的一致性，对整个系统提供统一的命名空间和访问接口[26]。另外，对系统的可扩展性进行了充分的考虑，加入了插件和属性管理子系统，以便系统可以随时增加新的功能，属性配置简单方便。

整体结构如图 3-1 所示：

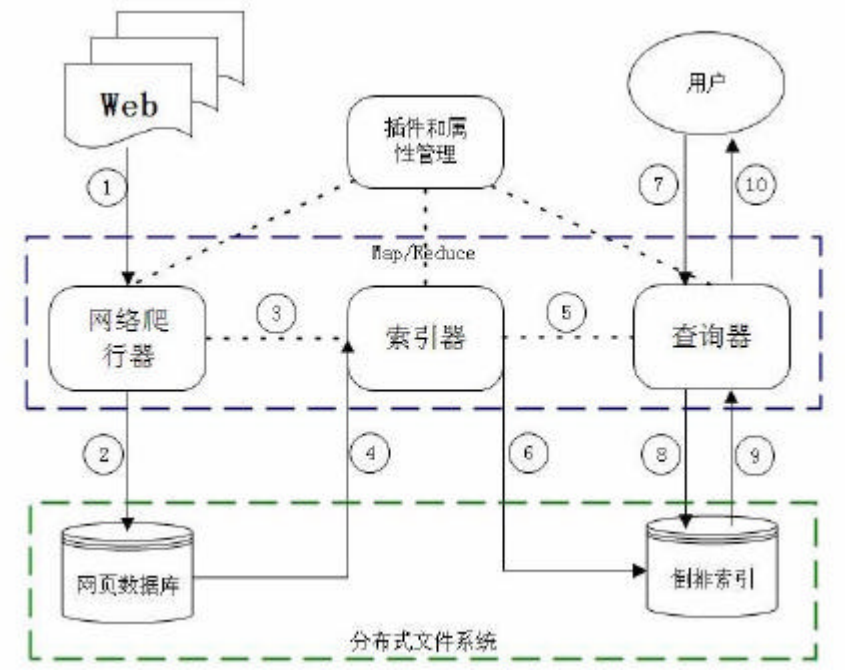


图 3-1 系统整体结构

Figure 3-1 System Architecture

图 3-1 给出了系统整体结构和子系统之间的关系，其中的数字标注了系统的执行流程：

1. 爬行器漫游于网络中抓取网页数据，对网页数据进行分析提取链接，生成链接列表以供爬行器下一次爬行使用。

2. 爬行器将网页文本数据和网页链接数据保存到网页数据库中。

3. 爬行器在完成采集工作后(所有应抓取的网页文本数据和网页链接数据都已经保存到数据库中)，通知索引器可以开始索引。

4. 索引器开始执行索引，对网页文本数据进行分词（主要包括英文分词和中文分词)，建立全文索引，并对一些倒排文档执行增量索引，新建立的索引保存在缓存中。同时，还要根据评分策略给网页评分。

5. 索引器执行全文索引和增量索引的过程中锁定部分倒排文档，因此，通知搜索器停止部分查询。当索引完成后索引器解除倒排索引文档锁定，通知搜索器新倒排索引可以提供查询。

6. 当缓存中的索引数量达到一个阈值后，索引器将索引保存到倒排索引库。

7. 用户通过搜索引擎前台界面向查询器提交搜索请求。查询器在接到请求后，解析用户查询字符串，提取出关键词和查询条件。

8. 查询器根据关键词和查询条件，生成查询任务，查询倒排索引文档。

9. 查询器获得查询结果。

10. 查询器根据网页评分对查询结果排序，将网页分值按由高到低顺序排列，并返回给客户。

以上 10 个步骤是系统主要执行流程，体现了各个子系统之间的联系，而没有体现子系统内部具体的设计，下面详细讨论各个子系统的设计。

3.2 业务子系统设计

3.2.1 分布式爬行子系统的设计

爬行器是搜索引擎系统获取网页数据的主要途径，它的主要功能是在 Web 上漫游对网页数据进行抓取。分布式爬行子系统在设计上应用 Hadoop 的 Map/Reduce 设计思想，可以充分利用硬件资源总体结构如图 3-2 所示：

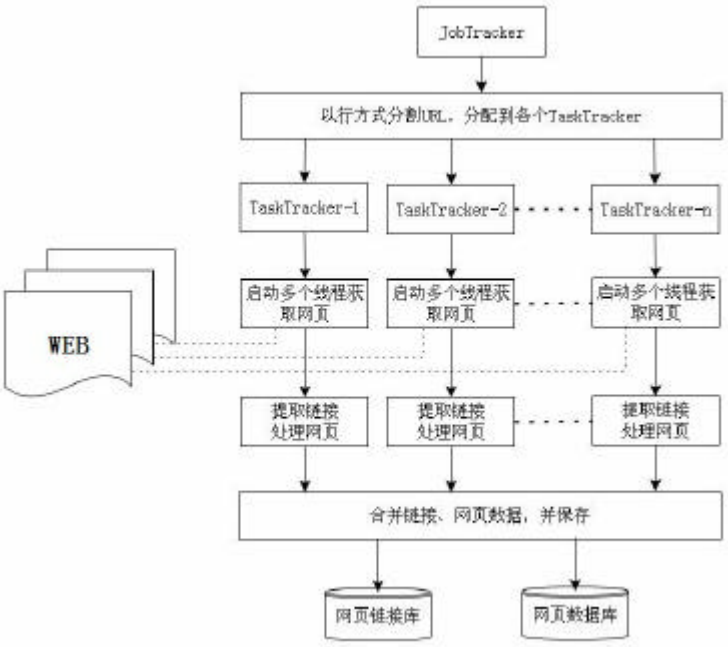


图 3-2 分布式爬行子系统总体结构

Figure 3-2 Distributed Crawl Subsystem Architecture

在爬行子系统总体结构中有以下三个方面需要注意：

1. JobTracker 运行于 Master 节点上，除了负责分割等待爬行的网页链接外，它还协调整个系统的分布式处理，但并没有参与到任务处理中，这样看似浪费了 1 个节点的计算量，但是却也带来了许多其他并行计算模型所没有的优点：整个系统更容易协调运行和扩展。在节点数量比较大时，这点浪费也就“微不足道”了。

2. JobTracker 将分割后的网页数据分配给各个 TaskTracker，每个 TaskTracker 会后

动 2 个 Map 下载任务，并监控 Map 执行状态，每一个 Map 包含多个下载线程，完成获取网页工作。在 Map 执行的最后阶段，将下载的网页以<网页链接，网页数据>数据对格式输出。

3. TaskTracker 在监测到 Map 执行结束后，启动 Reduce 合并数据对，解析网页，将网页分为两种数据 :网页链接和网页内容数据，分别保存到网页链接库和网页数据库中。

3.2.2 分布式索引子系统的设计

分布式索引子系统主要是对前一阶段爬行子系统采集的数据进行分析建立倒排索引文档[27]，它应用了 Hadoop 的 Map/Reduce 模式，主要包括以下几个方面：

1. 爬行子系统完成爬行任务后会通知索引子系统开始索引，需要索引的数据格式为<网页链接，网页内容>这种<key, value>数据对。JobTracker 以网页链接作为 key 分割网页数据。

2. 将分割后的网页数据分配给各个 TaskTracker，每个 TaskTracker 启动 2 个 Map 索引处理任务，并监控 Map 执行状态，生成单独的倒排索引，最后输出。

3. Reduce 将这些单独的倒排索引合并为一个整体并保存到倒排索引库中。

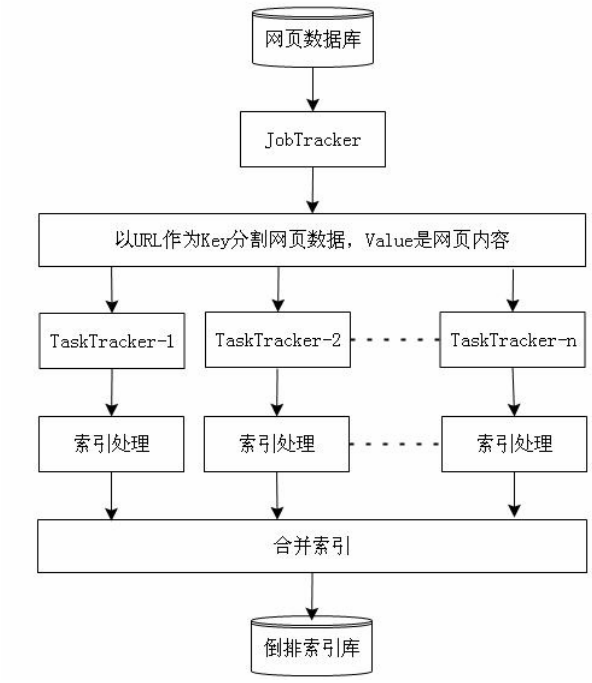


图 3-3 分布式索引子系统总体结构

Figure 3-3 Distributed Index Subsystem Architecture

图 3-3 所示就是分布式索引的总体结构，JobTracker 协调整个子系统中各 Slave 节点的分布式处理，各 Slave 节点索引处理后生成的其实都是索引片段，经过合并成为一个索引整体，最后保存到分布式文件系统中。

3.2.3 分布式查询子系统的设计

分布查询子系统与前面介绍的两个子系统有些不同，结构也要复杂一些。它增加了一个前台 Web 服务器提供查询界面，接收用户查询信息，并向用户呈现查询结果。Web 服务器使用基于 JSP 技术的 Tomcat，查询界面使用 JSP 创建[28]。后台查询服务器使用 Map/Reduce 编程模型设计，所有 Slave 节点都部署了查询模块。查询过程中，由 Master 节点对倒排索引库进行分段，每个 Slave 节点只检索一段索引，获得部分匹配结果集，经 Master 汇总排序后由 Web 服务器向用户呈现最终查询结果。

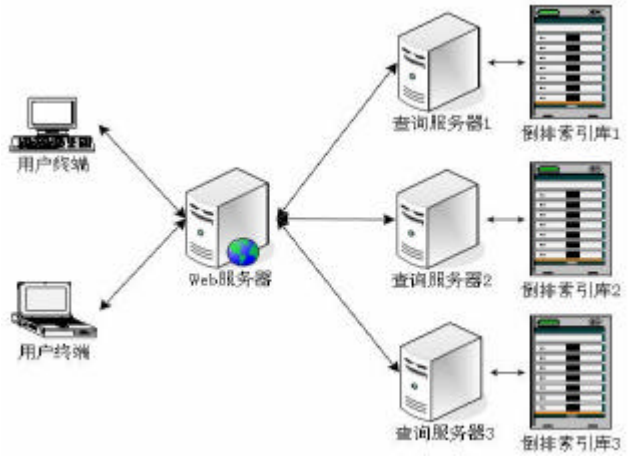


图 3-4 分布式查询示意图

Figure 3-4 Distributed Search Diagram

3.3 子系统内部执行流程

3.3.1 爬行流程

爬行器从一个或若干初始网页的 URL 开始[29]，通过 Http 协议下载网页数据，并对网页进行解析提取出网页链接和网页数据（包括：元数据、标题、内容等），根据过滤策略处理链接，生成链接列表以供爬行器下一次爬行使用；同时将网页数据保存到网页数据库中。爬行器循环执行这一流程从网络上抓取所需网页数据，直到满足终止条件为止。

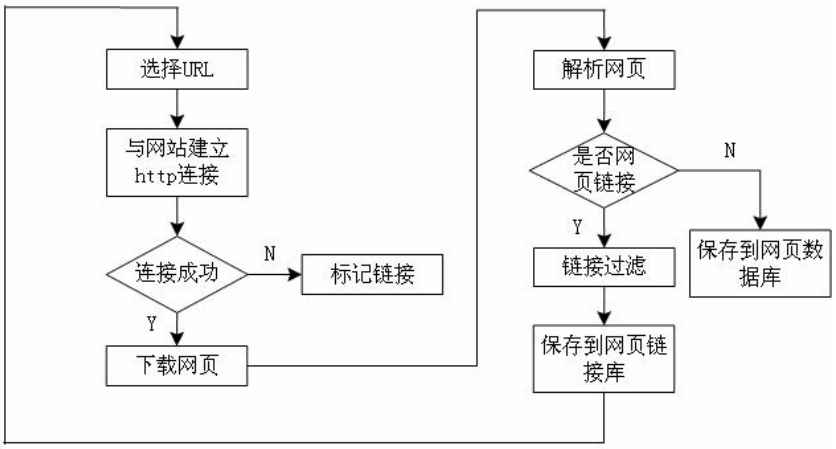


图 3-5 爬行流程图

Figure 3-5 Crawl Flow Chart

图 3-5 所示的是单个爬行器的执行流程。在具体实现过程中，子系统的下载网页、解析网页和链接过滤步骤可以无序执行，所以应用 Map/Reduce 模型实现了分布式运行，网页数据库和网页链接库采用分布式文件系统。

3.3.2 索引步骤

索引子系统总体结构，看似简单，但是从微观上看，实则包含许多复杂的处理步骤：数据预处理、分词、评分、生成 Lucene 索引、索引存储，执行这些步骤就完成了索引，索引步骤如图 3-6 所示。

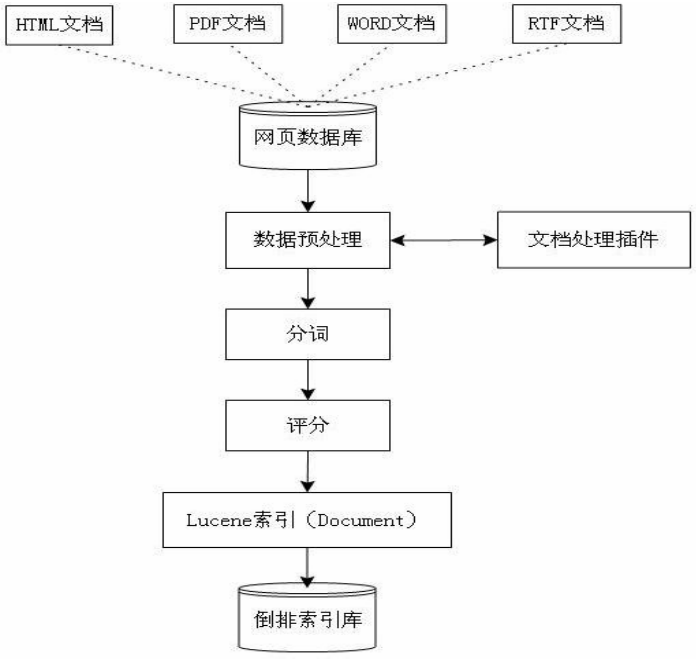


图 3-6 索引步骤

Figure 3-6 Index Flow Chart

其中评分、索引生成步骤可无序执行，使用 Map/Reduce 模式设计。中文分词模块

应用模块采用了层叠隐马尔可夫模型（Cascaded HiddenMarkov Model，CHMM），替换了 Lucene 默认的传统二元分词法[30,31]。

3.3.3 用户查询响应

Web 服务器在接收到用户查询信息后，会对查询字符串分词生成查询请求 Query，并向各个查询服务器提交查询请求。每个查询服务器检索倒排索引库，且只完成部分查询结果，所有查询结果经过汇总后按网页的评分排序，并向 Web 服务器返回这些结果，最终 Web 服务器将结果返回给用户。

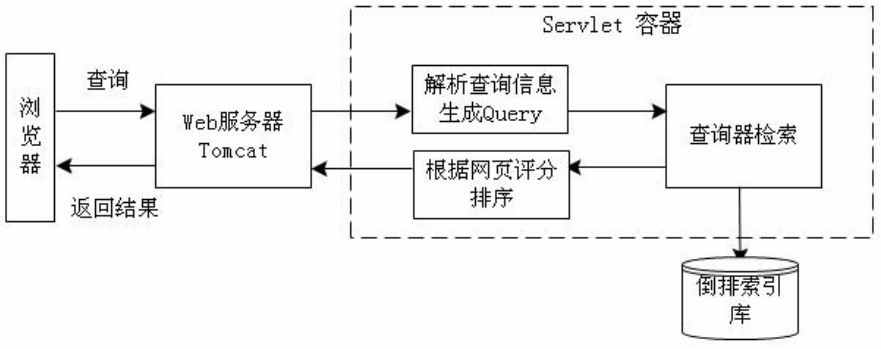


图 3-7 查询流程图

Figure 3-7 Search Flow Chart

3.4 本章小结

本章分析了分布式搜索引擎的基本结构，依据业务功能的不同，将系统设计为三个相对独立的业务子系统。最后深入分析了各个子系统的工作流程，为后面的模块实现部分打下基础。

第四章 分布式搜索引擎的实现

本章在上一章总体设计的基础上，阐述了 Map/Reduce 编程模式在分布搜索引擎系统各个模块中的具体应用方式，以及应用过程中遇到的问题，最后给出合理解决方案。

4.1 分布式爬行子系统的实现

4.1.1 子系统主要模块划分

爬行子系统的入口是 Crawler 类，它启动时记录爬行时间到日志中，接着依次调用 URL 选择与分割模块，网页获取模块，网页解析模块，链接过滤模块，以及数据存储模块，完成爬行任务。子系统主要类之间关系如图 4-1 所示。

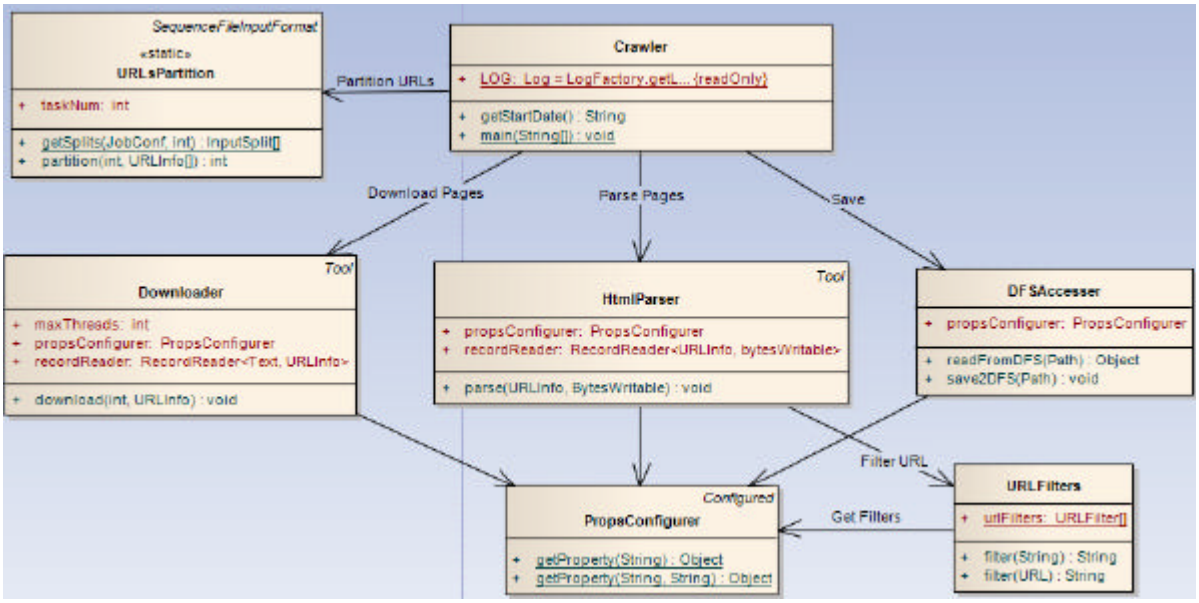


图 4-1 爬行子系统内部模块类关系图

Figure 4-1 Crawl Subsystem's Classes Diagram

4.1.2 URL 选择与分割模块

选择 URL 是爬行工作的第一步，爬行器从网页链接库中选择一个未抓取的网页链接执行采集任务。而分布式爬行器子系统要做的工作复杂的多，不仅仅是选择一个未抓取的链接这么简单，还要协调各个爬行器执行任务，防止任务冲突，为此需要预先对网页链接库数据进行分割，将分割后的链接列表分配给各个爬行器。这个模块只包含一个 URLPartition 类，其原理如下，分布式爬行器运行于 Hadoop 分布式环境下的 Slave 节点

上，统一由 Master 节点的 JobTracker 调度。在 URL 选择分割阶段， JobTracker 首先通过“心跳”获得所有 TaskTracker 的信息，形成可用 Slave 节点列表，同时访问网页链接库获得需要爬行的链接列表的规模 u，此后 JobTracker 会根据可用 Slave 的数量 n 与状态，以及网页链接实际存储的物理位置，将链接列表分割为 s 个片段（s 是 n 的整数倍），实际是个 InputSplit 数组，每个 InputSplit 包含一个列表片段。

4.1.3 分布式网页获取模块

获取网页是爬行工作的第二步，在此阶段充分利用了 Map/Reduce 模式并行计算的强大能力。在此阶段，涉及 4 个类：

- Downloader 类，实现了 org.apache.hadoop.util.Tool 接口，是分布式程序的入口，负责启动 Map/Reduce 任务。
- DownloadMapper 类，实现了 org.apache.hadoop.mapred.Mapper 接口，包含 map()方法，在 map()内部启动多个 DownloadThread 线程。
- DownloadReducer 类，实现了 org.apache.hadoop.mapred.Reducer 接口，包含 reduce()方法，在 reduce()内部合并数据。
- DownloadThread 类，java.lang.Thread 类的子类，被 DownloadMapper 调用来根据网页链接与网站建立连接，并下载网页。

类关系如图 4-2 所示：

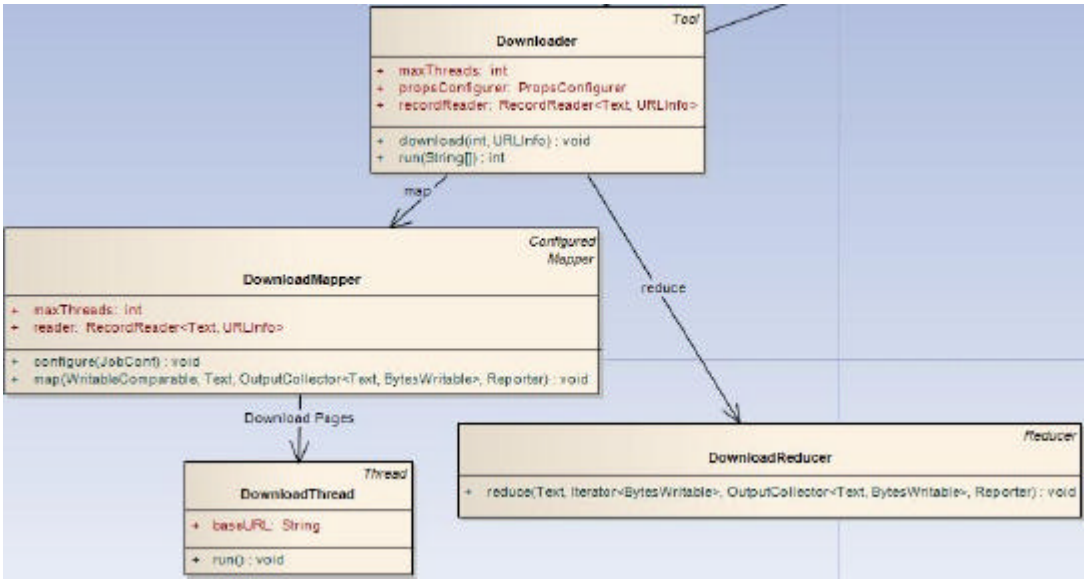


图 4-2 网页获取模块类图

Figure 4-2 Fetcher module’s Classes Diagram

执行流程如下：每个 Slave 节点上的 TaskTracker 根据分配到的链接列表新建 2 个

DownloadMapper 对象，每个 Map 对象又会启动 10 个下载线程，这样整个系统就拥有 n×2×10（n 是 Slave 节点的数量）个下载线程协同工作，因此整体性能会获得极大提高。

随后 Map 将数据映射为“<网页链接，网页数据>”这种类型的<key, vaule>对，作为中间数据输出。每个 TaskTracker 监测到 Map 的输出时，建立 1 个 DownloadReducer 类的实例，这 n 个 Reduce 汇总合并中间数据，如果需要，还可以根据 key 对中间数据排序，最后将所有中间数据作为一个整体输出。

下载线程 DownloadThread 利用 HttpClient 软件包与网站建立连接，通过发送 http 的 get 或 post 请求，并遵循 Robots 协议，获取权限范围内的网页。大部分数据直接发送 get 请求就可以获取到，而对于一些特殊的数据（如那些只有通过提交表单才能获得的数据）采用 post 请求。提供 TIMEOUT 参数是很有必要的，TIMEOUT 是链接建立的最长时间，如果对一个链接在超过 TIMEOUT 的时间内还无法建立连接，就放弃这个链接，这样可以保证系统不会因为不可用的链接而僵死。核心代码如下：

```
HttpClient httpClient = new HttpClient();
httpClient.getHttpConnectionManager().getParams().setConnectionTimeout(timeout);
GetMethod getMethod = new GetMethod(baseUrl);
getMethod.getParams().setParameter(HttpMethodParams.SO_TIMEOUT, timeout);
getMethod.getParams().setParameter(HttpMethodParams.RETRY_HANDLER,
new DefaultHttpMethodRetryHandler());
try {
int statusCode = httpClient.executeMethod(getMethod);
if (statusCode != HttpStatus.SC_OK) {
log.info("Method failed: " + getMethod.getStatusLine());
}
Header[] headers = getMethod.getResponseHeaders();
byte[] content = getMethod.getResponseBody();
```

当连接建立后，首先解析网页头信息，包括 HTTP 状态码、网页内容长度、网页字符集、网页内容编码、网页内容类型、网页重定向地址等。最后通过 getMethod.getResponseBody()获取网页数据。

4.1.4 分布式网页解析模块

超文本标记语言（HyperText Markup Language，简称为 HTML）是为网页创建和其它可在网页浏览器中看到的信息设计的一种置标语言。HTML 被用来结构化信息——例

如标题、段落和列表等等，也可用来在一定程度上描述文档的外观和语义。HTML 文档包含了复杂的信息结构，其中只有一部分是有用的信息集合，另一部分则是网页结构的描述。我们感兴趣的只是如何提取有用的信息，以便之后能有效地索引。网页解析的核心是提取出网页中的有效信息，这就要对网页结构进行分析。与获取网页阶段相似，本模块也应用了 Map/Reduce 模式，由三个类构成，这三个类的关系如图 4-3 所示。

- HtmlParser 类，实现了 org.apache.hadoop.util.Tool 接口，是分布式程序的入口，负责启动 Map/Reduce 任务。
- ParseMapper 类，实现了 org.apache.hadoop.mapred.Mapper 接口，包含 map()方法，在 map()方法内部解析网页。
- ParseReducer 类，实现了 org.apache.hadoop.mapred.Reducer 接口，包含 reduce()方法，在 reduce()内部合并解析后的网页数据。

其中 URLInfo 和 HTMLPage 分别是网页链接和网页数据的包装类，本小节暂不讨论，后面详细介绍。

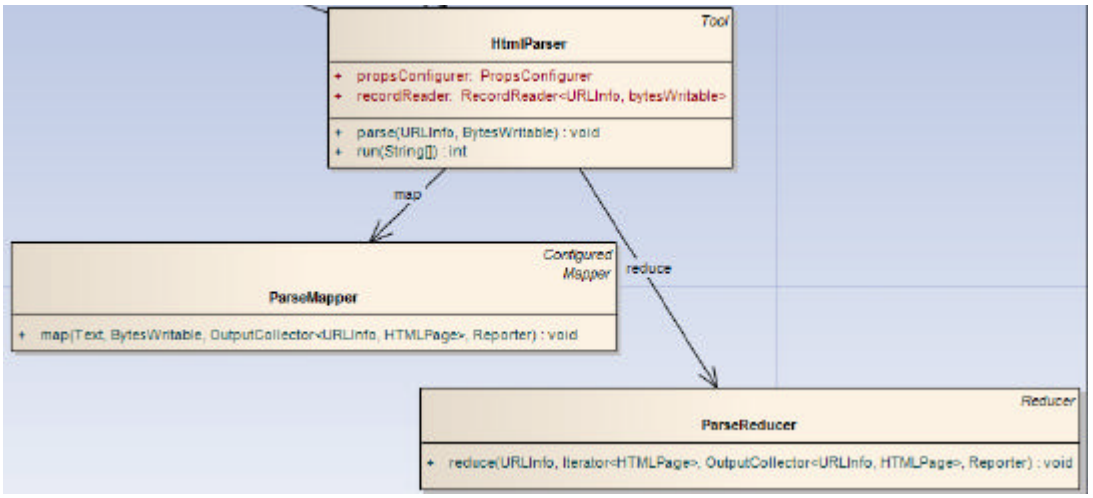


图 4-3 网页解析模块类图

Figure 4-3 Page Parser module's Classes Diagram

此阶段输入的数据正是上一阶段输出的大量形如<网页链接，网页数据>的数据对。这些数据对经过分割后分配给所有的 TaskTracker，根据网页链接从数据对中取得相应的网页数据。解析网页主要是分析与过滤 HTML 文档标签，提取标签中的属性和内容。如，从锚链接中提取 href 属性值（提取<a href="对外链接">中 href 的属性值），<base href="URL">、<img src="URL">、<frame src="URL">，以及 JavaScript 脚本中的属性值也都属于提取范围。最后由 Reduce 将所有 Map 提取出来的对外链接汇总合并。解析过程如下：

在本文的系统中我们扩展了 HTMLParser **工具包的** Parser **类，为** HtmlParser **类初始化网页内容数据。**

```
public class HtmlParser extends Parser {
public HtmlParser(byte[] contentByte){
setHtmlContent(contentByte);
setFeedback(STDOUT);
setNodeFactory(new PrototypicalNodeFactory());
}
public void setHtmlContent(byte[] htmlContent){
if(null == htmlContent)
throw new IllegalArgumentException ("网页内容不能为空");
setLexer(new Lexer(new Page(new String(htmlContent))));
}
}
```

接着分别提取 Link、Title、Meta、Body **等标签。提取** Link **标签代码如下，其余标签的处理方法类似。**

```
NodeFilter linkFilter = new NodeClassFilter(LinkTag.class);
HtmlParser parser = new HtmlParser(htmlContent);
parser.setEncoding(parser.getEncoding());
NodeList linkList = parser.extractAllNodesThatMatch(linkFilter);
for (int i = 0; i <linklist.size(); i++) {
LinkTag Link = (LinkTag) linkList.elementAt(i);
……..//逐一处理每一条 Link 标签内数据。
}
```

提取出的数据，除“网页链接”外，其余数据在爬行阶段无须使用，统一保存到网页数据库中。网页链接经过进一步处理后供爬行使用。

4.1.5 链接过滤模块

大型网站由于有统一的技术规范，其网页文件中包含的链接大多都是符合规范的可用链接，但网络上更多的是中小网站和个人网站，其中有许多都没有统一的技术规范，这些网站的网页文件中包含了大量不合规范、无效和重复的链接，只有经过规范化、有效性过滤和去重处理后才有使用价值。以上三个功能是通过插件模块 URLFilters **加载**

的，每个功能都由一个过滤器类提供。

这些过滤器为：规范化过滤器 NormalizeFilter，有效性过滤器 UsefulFilter，以及去重过滤器 DedupFilter。下面详细介绍这些过滤器。

- 规范化过滤器 NormalizeFilter 的工作包括以下几方面：

1. 链接字符串转小写。因为在 HTTP 的规范中 url 被设计为大小写不敏感，使得人们不再需要记忆复杂的网址，这一设计为上网冲浪提供了极大的便利。例如以下两个链接 http://www.tyut.edu.cn/first.html/和 HTTP://WWW.TYUT.EDU.CN/FIRST.HTML/所代表的都是同一个网页。

2. 相对链接转换为绝对链接。现在的网页中大量存在着形如“./sample2.html/”和“../../sample3.html/”这样的相对链接，这些链接在上网浏览时，是可以通过当前网页进行跳转的，当单独提取出来时，却又是毫无意义的，这就需要将它们转化为绝对链接。转化依据是：包含“./”字符串的直接添加到当前网页链接后面；包含“../”字符串的根据“../”的数量，对当前网页链接向前追溯相同数量的级别后再添加。例如，当前网页链接为 http://www.tyut.edu.cn/nic/new/，相对链接“./sample2.html/”转化为 http://www.tyut.edu.cn/nic/new/sample2.html/；相对链接“../../sample3.html/”转化为 http://www.tyut.edu.cn/sample3.html/。

3. 去除无意义的字符串。网页链接中时常会出现一些没有实际意义的字符串，规范化的过程中需要去除这些字符串。例如，http://www.tyut.edu.cn/sample3.html#test 中的“#”只是起到网页内容定位的作用，与 http://www.tyut.edu.cn/sample3.html 所代表的实际是一个网页，所以应该删除。

- 有效性过滤器 UsefulFilter 主要有两方面工作：

1. 对经过规范化处理的网页链接，如果仍不符合 URL 规范（这种链接被称为 Bad Formed URL），就会被删除。其实现原理是扫描网页链接字符串，通过正则表达式进行模式匹配运算。

2.对网页链接的抓取范围进行控制，超出抓取范围的网页链接会被忽略。其实现原理也是利用正则表达式。例如，要限定链接的范围是 tyut.edu.cn 域时，使用 http://([a-z0-9]*\.)*tyut.edu.cn/([a-z0-9]*\.)*模式串。

- 去重过滤器 DedupFilter

去重过滤器 DedupFilter 也是非常重要的步骤，因为在提取出的链接中，存在着大量重复的数据，如果不删除重复链接，那么爬行器的执行效率就会大打折扣。如何去重

呢？最简单的方法莫过于直接逐个比对字符，在多个相同的链接里删除冗余链接只保留一个。但是这种方法的缺点也很明显，链接字符串往往包含许多字符，逐位比对会耗费大量计算量和时间。在本文所讨论的系统中，我们使用了 MD5 散列函数比对法。

通过扩展 JDK 的 java.security 包中的 MessageDigest 类，使用 MD5 算法对一个网页链接字节串产生指纹（fingerprint），主要代码及相关意义如下：

```
public byte[] md5Fingerprint(){
MessageDigest md5Digest = MessageDigest.getInstance("MD5");
md5Digest.update(URL.getBytes());
//以 MD5 算法对链接计算指纹
byte[] fingerprint = md5Digest.digest();
//获得链接的指纹
}
```

指纹是一个固定长度的字节数组，比对两个定长字节数组要比比对两个不定长的字符串的速度高几个数量级，因此可以提高去重速度。

4.1.6 分布式数据存储模块

系统数据需要存储在一个适合并发访问、稳定可靠的文件系统中。HDFS 正好符合这种需求，但文件系统中存储的数据没有统一格式，都必须由用户编程控制。爬行子系统处理的数据最终都要存储到基于 HDFS 分布式文件系统的数据库中。数据的访问控制又涉及到两个独立的数据库：网页链接库和网页数据库。网页链接库中存储的是供爬行使用的网页链接；网页数据库存储的是从网页中提取的各种数据信息，也被称为网页快照，是搜索结果中摘要的数据来源，也是索引模块数据的来源。数据库的访问包含三个类，关系如图 4-4 所示：

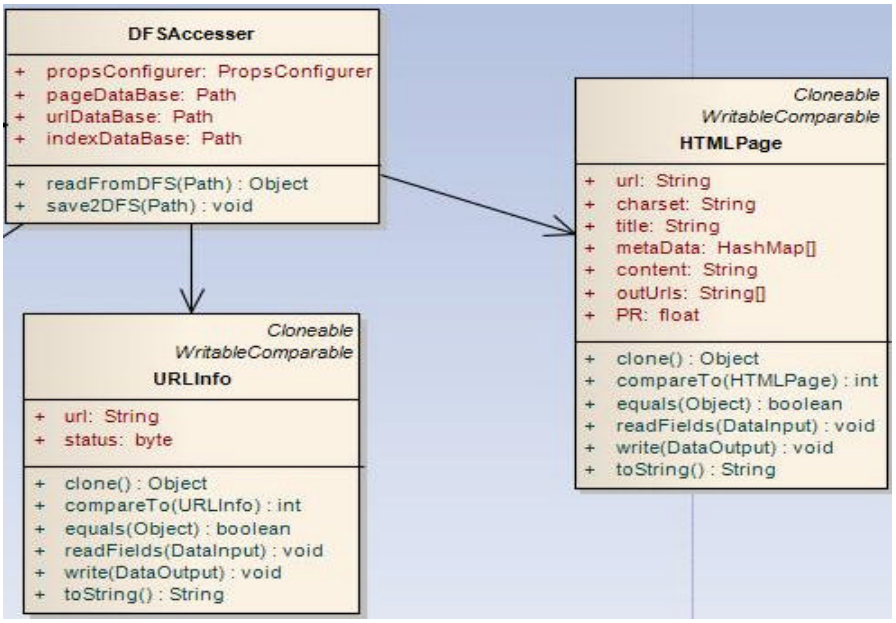


图 4-4 数据存储访问模块类图

Figure 4-4 Data Accessor module's Classes Diagram

数据库的存储和访问控制被封装在 DFSAccesser **类中，包含了三个数据库的访问控制：网页链接库（**urlDataBase**），网页数据库（**pageDataBase**），以及索引数据库（**IndexDataBase**）。其中爬行子系统只涉及到前两个数据库，索引数据库在索引子系统和搜索子系统中才会使用到。**

由于在 HDFS **文件系统中实际存储的数据是字节，如果没有合理的控制手段，则保存和读取的只是一些无意义的字节而已。**HDFS **提供了一种实现自有数据存取的机制。所以网页链接包装类** URLInfo **和网页数据包装类** HTMLPage **都必须实现** org.apache.hadoop.io.WritableComparable **接口，实现** readFields(DataInput in) **和** write(DataOutput out)**这两个方法，只有这样才能使得系统可以明白字节的含义，得到完整的数据包装类实例。**

另一点需要注意的是，Hadoop **平台对数据的处理方式。**Hadoop **的** Map/Reduce **两种任务实际处理的大量数据都是形如**<key, value>**的数据对，然而为数众多的** Java **基础类对象都是无法作为** key **的。为了进一步拓展处理的数据类型，那些需用作** key **的数据类型，必须实现包装类，并实现** compareTo()**方法。例如** java.lang.String **是无法充当** key **的，**String **的包装类** org.apache.hadoop.io.Text **却可以。在本文的系统中，因为** URLInfo **和** HTMLPage **都实例化了** compareTo()**方法，所以这两个类都可以在** Hadoop **平台中充当** key **的角色。**

4.2 分布式索引子系统的实现

根据分布式索引流程将子系统划分为 4 个模块，分别为：多格式文档统一处理模块、中文分词模块、网页评分模块和分布式索引生成模块，本节详细分析各个模块的实现。

4.2.1 多格式文档统一处理模块

Web 技术日新月异，新兴数据格式大量涌现，网页数据已经从单纯的 HTML 文档格式，发展成为包括 PDF、WORD 和 RTF 等多种文档格式，这就要求索引工具拥有灵活多变的数据处理方式。在本文的索引系统模块中，使用了文档处理插件机制来扩展数据处理方式。这样做的优点是，文档格式处理与索引系统模块没有紧密耦合，当有新的文档格式需要处理的时候，只需增加新的插件而不必更改系统整体结构就可满足需要。目前系统包含了主流的文本文档、PDF 文档、WORD 文档和 RTF 文档处理插件，其中文本文档是最重要也是使用最广泛的文档格式[32]。

通过建立一个通用的 DocumentHandler 接口，开发能够用来处理多种文档格式的统一处理插件 DocumentHandlerPlugin，系统中所有的文档处理器都实现了 DocumentHandler 接口，并通过 DocumentHandlerPlugin 来加载，从而为索引系统添加了多种文档格式的统一处理能力。相应的文本、PDF、WORD 和 RTF 文档的处理类分别为：TxtHandler、PdfHandler、WordHandler 和 RTFHandler，它们统一通过 DocumentHandlerPlugin 类加载，其关系如图 4-5 所示：

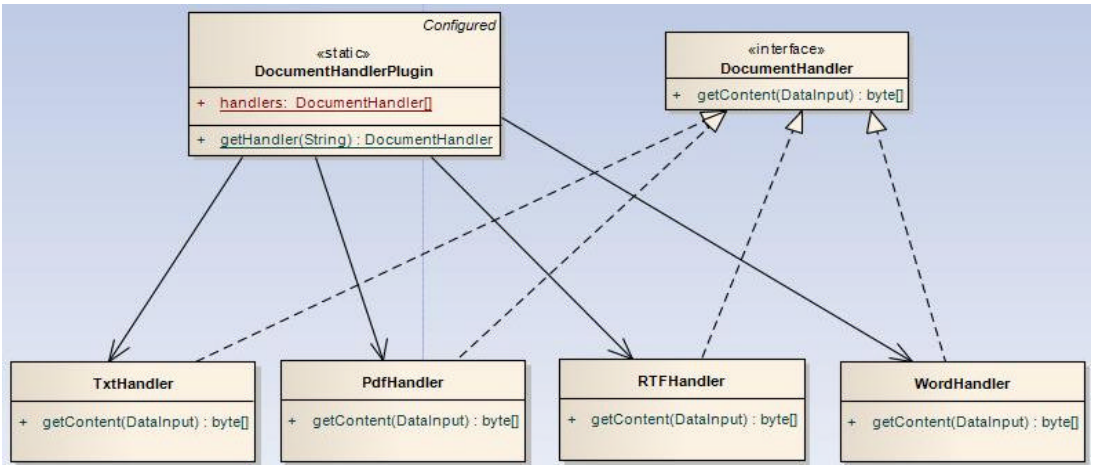


图 4-5 文档处理模块类图

Figure 4-5 Document Handler module's Classes Diagram

- DocumentHandler 接口定义如下：

```
public interface DocumentHandler{
```

public byte[] getDocument(DataInput in) throws IOException;

}

getContent(DataInput in)方法用于提取文档内容。DocumentHandler 接口的引入，屏蔽了不同文档格式的差别。这样设计的系统不仅可以最大限度地为用户屏蔽各种文档格式之间的差异性，而且为系统的可扩展性预留了广阔的空间。

4.2.2 中文分词模块

分词效果的好坏直接影响着索引结果的相关度排序及索引的效率和准确度，因此索引系统必须要处理好分词。由于英文是由空格和标点符号来分隔单词的，因此对于英文文档的处理比较简单，使用 Lucene 的标准分析器 StandardAnalyser 就可以很好的完成英文分词。而中文分词却涉及到许多问题，在我们的系统中扩展了中科院的 ICTCLAS 中文分词工具，构建本文的智能简体中文分析器 ItelChineseAnalyzer。

分词模块的最底层使用了 Lucene 的 Analyzer 抽象类。ItelChineseAnalyzer 继承了 Analyzer 抽象类，该抽象类中定义了一个公有的抽象方法 tokenStream（String fieldName, Reader reader）返回的类型是 TokenStream。该方法是用于分析文本，所有 Analyzer 的具体子类都在这个方法中实现了从文本中提取索引词组的策略、算法。而返回的 TokenStream 类是即可从文本或者从查询词组中枚举 token 序列的抽象类。对于 ItelChineseAnalyzer，它是用于将 Reader 中读入的 Stream 分解成最原始的词组（Token），在 ItelChineseAnalyzer 分解 Stream 之后，一个或多个 TokenFilter 会用于过滤这些词组中无意义的词组。模块涉及的类如图 4-6 所示：

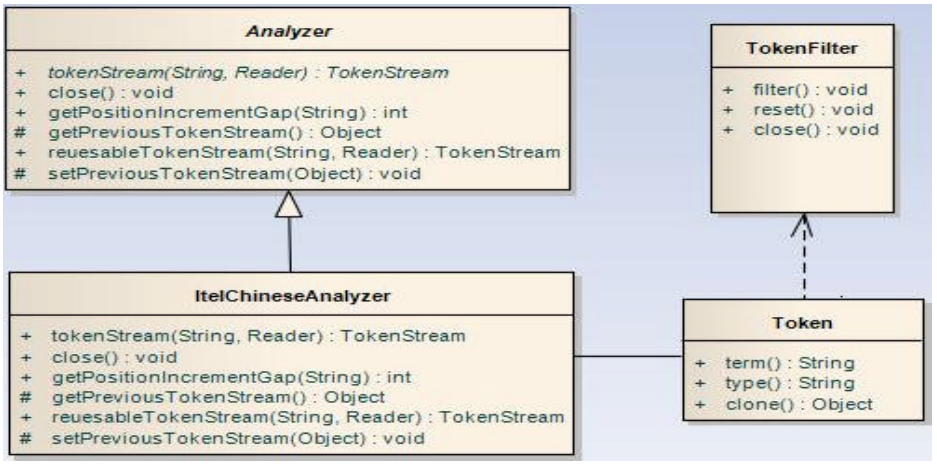


图 4-6 分词模块类图

Figure 4-6 Analyzer module's Classes Diagram

模块采用了层叠隐马尔可夫模型（Cascaded HiddenMarkov Model，CHMM），将汉

语词法分析的所有环节都统一到了一个完整的理论框架中，获得很好的总体效果，并在分词精度与分析速度这两个相互制约的因素上取得了一定的进步，达到速度与精度的平衡，为系统汉语词法分析提供保障。

4.2.3 网页评分模块

每当用户使用搜索引擎搜索信息时，总是希望第一时间找到自己想要的结果。这就要求检索时对结果按信息匹配程度和网页的重要程度对网页排序。每个网页的得分都是在索引时就已经计算好，检索时只需对匹配网页按得分由高到低返回网页序列。目前主流的网页评分策略主要是以网页内容和网页的链接状况作为依据，在本文讨论的系统中 TFIDFScoreFilter 和 PRScoreFilter 类实现网页评分，这两个类分别实现了 TF-IDF 和 PageRank 这两种评分策略，下面对其原理进行分析。

- TF-IDF 评分策略

TF-IDF（term frequency－inverse document frequency）是一种用于网页检索与文本挖掘的常用评分技术。TF-IDF 是一种统计方法，用以评估某一词语对于一个文档的重要程度。原理如下：词语的重要性随着它在文档中出现的次数成正比增加，但同时会随着它在文档库中出现的频率成反比下降。TF-IDF 评分策略常被搜索引擎应用，作为文档与用户查询之间相关程度的度量。TF-IDF 顾名思义，由两部分构成：词频（TF 全称 term frequency）和逆向文档频率（IDF 全称 inverse document frequency）。

词频指的在一个给定的文档中，某一个特定的词语在该文档中出现的次数。实际应用中由于同一个词语在长文档里可能会比短文档有更高的词频，而不管该词语重要与否，所以词频通常会被正规化，以防止它偏向长的文档。对于在某一给定文档里的词语 t_i 来说，它的词频可表示为：

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{4-1}$$

公式 4-1 中 $n_{i,j}$ 是该词在文档 d_j 中的出现次数，而分母则是在文档 d_j 中所有词语的出现次数之和。

逆向文档频率与词频完全不同的一个概念，它从文档的数量关系出发，是一个词语普遍重要性的度量。某一特定词语的逆向文档频率，可以由总文档数目除以包含该词语的文档数目，再将得到的商取对数得到：

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d : d \ni t_i\}|} \tag{4-2}$$

公式 4-2 中|D|代表文档库中的文档总数，$|\{d : d \ni t_i\}|$代表包含词语 t_i 的文档数目。最终的 TF-IDF 如 4-3 所示：

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \cdot \mathrm{idf}_i \tag{4-3}$$

某一给定定文档内的高词语频率，以及该词语在整个文档集合中的低文档频率，可以产生出高权重的 TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语，突出特定词语与文档的相关度。

- PageRank 评分策略

PageRank 是一种根据网页之间相互的链接关系计算网页排名的技术，以 Google 公司创办人拉里·佩奇（Larry Page）之姓来命名。通过 PageRank 可以体现网页的相关性和重要性。

它的基本思想是：如果一个网页被多次引用，那么这个网页可能很重要；虽然一个网页没有被多次引用，但是被重要的网页引用，那么这个网页也可能是很重要的；一个网页的重要性被平均的传递到它所引用的网页，计算公式如下：

$$PageRank(p_i) = \frac{d}{N} + (1-d)\sum_{p_j} \frac{PageRank(p_j)}{Out(p_j)} \tag{4-4}$$

公式 4-4 中 p_i 是被当前正被计算 PageRank 得分的网页，$PageRank(p_i)$ 是网页 p_i 的 PageRank 得分，p_j 是引用 p_i 的网页，$Out(p_j)$ 是 p_j 链出网页的数量，而 N 是参与计算的所有网页的数量，d 是阻尼系数，一般取值 0.15。

这个公式体现了一种随机浏览的概念，即人们上网随机打开一些网页，点击一些链接。一个网页的 PageRank 值也影响了它被随机浏览的概率。为了便于理解，这里假设上网者不断点击网页上的链接，最终到了一个没有任何链出的网页，这时候上网者会随机到另外的网页开始浏览。

为了体现对那些有链出的网页的公平，引入了阻尼系数 q（damping factor）的概念（q = 0.15）。其意义是，在任意时刻，假想的随机浏览者停止在某网页后继续浏览的概率的算法被用到了所有网页上，估算网页可能被上网者放入书签的概率。

在实际应用中，采用了迭代的方法计算网页 PageRank 得分，也就是先给每个网页一个初始分值（默认为 1），然后利用 4-4 的公式，循环进行有限次（100 次）运算得到近似的网页得分。

4.2.4 分布式索引生成模块

索引是子系统执行过程的最后一个步骤，由 IndexGenerator 类调用 Lucene 相关类完成这一工作。Lucene 索引逻辑上是由 Segment、Document、Field 和 Term 构成的。四者之间的关系如图 4-6。

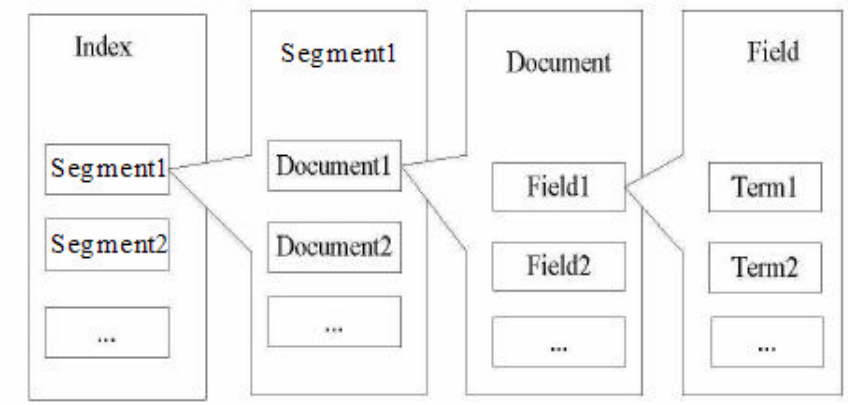


图 4-7 Lucene 索引的结构

Figure 4-7 Lucene Index Document Architecture

每个完整的倒排索引，都是由若干个索引片段 Segment 组成，每个片段都是一个独立的子索引。片段中包含若干个文档 Document 集合，文档代表索引和检索结果的一个实体，类似于数据库中的表。文档又由若干的字段 Field 组成，类似于数据库中的列，字段是项 Term 的集合，项是索引中最小单位，项是通过对文本数据分词得到的，也可以看做是文档中的关键词[33,34]。在 Lucene 的索引结构中，字段的概念比较重要，比如 URL、标题、网页内容等都可以设置为字段，另外字段还有是否存储、分词、可检索的属性。对于用户的检索请求而言，其输入的基本单位为 Term 或 Term 的组合，返回的数据结构为 Document 的集合。

本模块包括三个类，关系如图 4-8 所示

- IndexGenerator 类，实现了 org.apache.hadoop.util.Tool 接口，是分布式程序的入口，负责启动 Map/Reduce 任务。
- IndexMapper 类，实现了 org.apache.hadoop.mapred.Mapper 接口，包含 map()方法，在 map()方法内调用 IndexGenerator 类执行 Document 生成任务。
- IndexReducer 类，实现了 org.apache.hadoop.mapred.Reducer 接口，包含 reduce()方法，在 reduce()方法内通过调用数据存储模块 DFSAccesser，将索引保存到倒排索引库中。

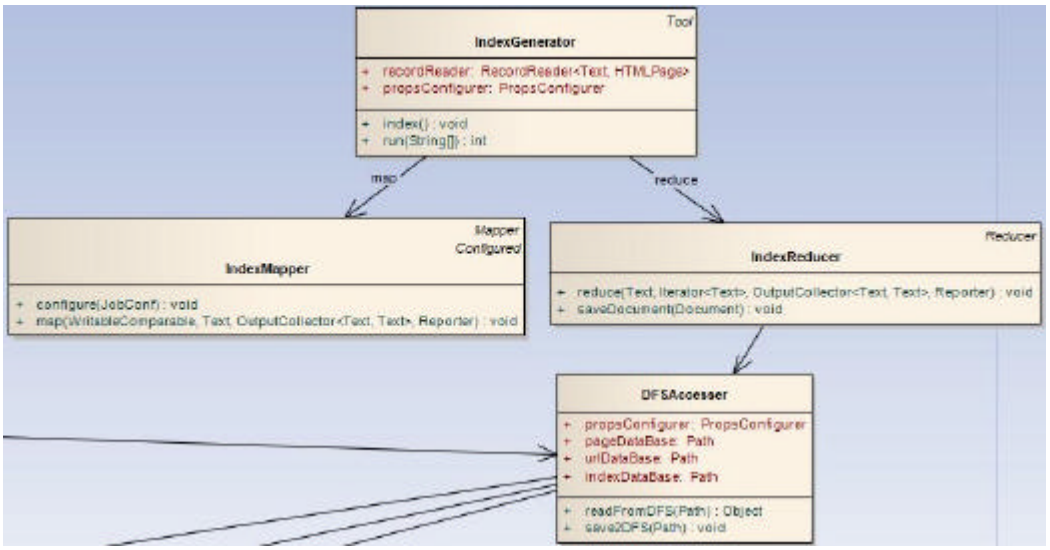


图 4-8 索引生成模块类图

Figure 4-8 Indexer module's Classes Diagram

IndexGenerator **类在生成索引时，主要做了以下两方面工作：**

1. **生成索引字段。包括主题、网页链接等数据量小、查询率高的字段，保存到索引中；而像网页内容字段，因数据量大，仅索引而不保存。**

2. **建立** Document **文档对象，将索引字段添加到文档。最终通过调用数据存储模块** DFSAccesser**，将索引保存到倒排索引库中。**

核心代码如下：

```
public class IndexGenerator {
   public static Document createDocument(HTMLPage htmlPage) throws
   IOException, InterruptedException {
   Document doc = new Document();
   doc.add(new Field("url", htmlPage.getUrl(), Field.Store.YES,
   Field.Index.NOT_ANALYZED));
   doc.add(new Field("title", htmlPage.getTitle(), Field.Store.YES,
   Field.Index.ANALYZED));
   doc.add(new Field("content", htmlPage.getContent()));
   return doc;
   }
}
```

其中需要注意的是，网页链接保存在索引中，但不分词；网页标题字段保存在索引中，对网页标题分词；网页内容字段不保存在索引中，但对网页内容分词。

```
IndexWriter indexWriter = getWriter(indexFilePath);
```

```
indexWriter.addDocument(doc);
indexWriter.optimize();
indexWriter.close();
```

通过 IndexWriter 将 Document 对象暂时保存到内存中，最终保存到分布式倒排索引库中。经过以上步骤网页索引就完成了，用户就可以通过供查询模块提供查询功能，搜索信息了。

4.3 分布式查询子系统的实现

查询子系统是用户提交查询和获得查询结果的平台，是搜索引擎系统实现中的一个重要子系统，性能稳定、使用方便的查询器可以提高用户对系统的满意度。本节将对分布式查询器的具体实现方式进行分析。

4.3.1 前台查询界面

系统利用 JSP 技术构建前台，响应用户查询请求，并向用户返回查询结果。首先，通过 search.html 页面接收用户输入的字符串，将其包装到 Form 表单中的 query 字段，并以 Get 方式提交。

图 4-9 前台查询界面

Figure 4-9 Front Query Interface Diagram

接着 search.jsp 从表单中提取用户查询字符串 query，将字符串作为参数调用后台的 SearchBean 对象，由 SearchBean 负责查询。最后，由 SearchBean 返回的查询结果集合，也是通过 search.jsp 以列表的方式显示给用户。

图 4-10 查询结果界面

Figure 4-10 Search Results Interface Diagram

4.3.2 分布式查询器模块

本模块扩展了 Lucene 的查询模块，继承了 Lucene 查询功能的优点，同时应用了 Map/Reduce 模式，可分布式执行查询任务。查询的过程中需要调用 Lucene 中的四个类，分别是：

1．IndexSearcher 类

IndexSearcher 称为索引检索器，是 Lucene 中最基本的检索工具，所有的检索都会用到索引检索器，但是在使用索引检索器之前，还要做一些准备工作，即对索引检索器进行初始化。初始化时需要设置索引库参数，这样才能让索引检索器定位索引，进行查询。索引检索器最重要的方法是 search 方法，通过接受 Query 对象做为参数并返回 Hits 对象。

2．Term 类

Term 是查询的基本单元，由一对字符串元素组成段的名称和字段的值。Term 对象在索引子系统的实现过程中已涉及到，但它是通过中文分词模块对数据进行分词处理后自动生成的，所以在索引过程中一般不需要考虑。而在查询过程中，会创建 Term 对象

并与 TermQuery 共同使用。

3．Query 抽象类

在分层的查询器构建方式中，最底层的 org.apache.lucene.search.Query 是一个高度抽象的查询类，通过继承这个类可以派生出许多查询类，实现不同类型的查询，这样的查询类包括：关键词查询类（TermQuery）、多关键词查询类（MultiTermQuery）、布尔表达式查询类（BooleanQuery）等。在本文所讨论的查询器模块中主要使用了以上三个查询类，下面详细介绍这些类的功能。

关键词查询类是最基本的查询类，它以单个关键词作为查询条件，返回包含了这一关键词的所有文档集合。关键词查询类在搜索引擎系统中特别有用，因为用户在实际使用中，往往对所搜索信息的详情并不十分了解，仅输入一个关键词就可以获得相关网页信息，极大的方便了用户的使用。

多关键词查询类提供了多关键词查询的功能，其基本的思想是先通过索引检索器在索引库中找到符合匹配条件的标记集，然后通过标记集再查找文档集合。具体实现多关键词查询器的包括：通配符查询器、模糊查询器和正则表达式查询器。

布尔表达式查询类将各种查询类型组合成复杂的查询方式，也是一种高级的查询方式。构成布尔表达式查询的各个查询条件被称为子查询，子查询使用特殊的逻辑词限定匹配模式，逻辑词可以是表示逻辑“或”的 OR，表示逻辑“与”的 AND，以及表示逻辑“非”的 NOT，还可以是这些逻辑词的组合[35]。

4．Hits 类

Hits 类是查询结果的集合的包装类，它是一个查询结果文档（匹配特定查询的 Documents）指针的容器，通过 Hits.doc(n)方法可以获得指定文档对象。出于性能考虑，Hits 并没有一次性从索引中加载所有匹配的文档，而是采用增量加载，每次查询只加载一部分，通过多次加载才能返回全部结果。

利用这四个类以及它们所提供的相关方法，就可以实现一个通用的查询模块。在模块内部利用 Map/Reduce 模型，将查询任务分解为两部分：分布式查询和结果汇总，分别封装在 SearchMapper 类和 SearchReducer 类中。程序的入口是 SearchBean 类，它调用 SearchMapper 和 SearchReducer 对象执行分布式查询和汇总任务，三个类关系如图 4-11：

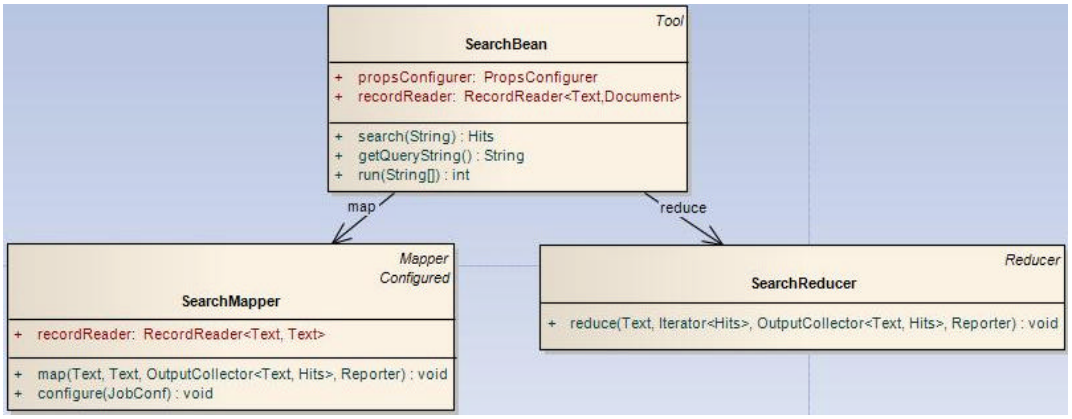


图 4-11 查询模块类图

Figure 4-11 Search module's Classes Diagram

查询逻辑的全部实现集中在 SearchMapper **类的** map **方法中，主要包括以下四个步骤：**

- **初始化检索器，设置索引库参数，让其定位索引。**

IndexSearcher searcher = new IndexSearcher(DFSAccesser.indexDataBase);

- **创建中文分词对象，负责提取关键词。**

ItelAnalyzer analyzer = new ItelAnalyzer();

- **创建查询请求对象，使用中文分词对象解析用户查询字符串。**

Query query = QueryParser.parse(queryString, "content", analyzer);

- **检索索引库，获得匹配结果集。**

Hits hits = searcher.search(query);

检索完成后，所有的 map **输出** Hits，SearchReducer **对象接收** Hits **集合，经** reduce **方法汇总排序后，将** Hits **集合合并为一个** Hits **对象，返回给前台界面。**

4.4 本章小结

本章深入业务子系统内部，详细分析了三个子系统内部模块实现过程中相关的重要类的实现，以及类之间的关系。对那些可以分布式执行任务的模块，应用 Map/Reduce **模式，并讨论其实现细节和难点。**

第五章 系统测试与分析

5.1 实验环境

1. 硬件环境：CPU：酷睿 1.86G 双核 CPU；内存：1G DDR-2；硬盘：160G；

2. 软件环境：操作系统：Windows XP + CygWin；Java 环境：JDK 6.0；Hadoop 版本：Hadoop-0.19.1。

3. 网络环境：由上述配置 4 台 PC 构成，PC 之间由 100M 网互联。四台 PC 中的一台作为 Master 节点，其计算机名为 ff，另外 3 台作为 Slave 节点，计算机名分别为：WYW、cien 和 yhx。外网连接使用了校园网。

5.2 分布式搜索引擎配置

由于本文所讨论的分布式搜索引擎是基于 Hadoop 平台开发的，系统的分布式处理依赖于平台提供的基础功能，所以必须首先配置 Hadoop 分布式平台。平台的配置分为两部分：SSH 服务配置和 Hadoop 核心配置。

5.2.1 SSH 服务配置

SSH 为 Secure Shell 的缩写，由 IETF 的网络工作小组（Network Working Group）所制定；SSH 为建立在应用层和传输层基础上的安全协议。传统的网络服务程序，如 FTP、POP 和 Telnet 其本质上都是不安全的；因为它们在网络上用明文传送数据、用户帐号和用户口令，很容易受到中间人（man-in-the-middle）攻击方式的攻击。就是存在另一个人或者一台机器冒充真正的服务器接收用户传给服务器的数据，然后再冒充用户把数据传给真正的服务器。

而 SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。透过 SSH 可以对所有传输的数据进行加密，也能够防止 DNS 欺骗和 IP 欺骗。SSH 的另一项优点为其传输的数据是经过压缩的，所以可以加快传输的速度。在 Hadoop 平台环境下，Master 节点为了通过 SSH 全权控制 Slave 节点，要求保证 SSH 无密码连接所有 Slave 节点，配置步骤如下：

（1）在构建分布式平台的所有的计算机上安装 sshd 服务，所有计算机建立 ff 用户。

（2）免密码设置。在 Cygwin 控制台中输入 ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa ，这时会提示保存位置：Enter file in which to save the key（/home/ff/.ssh/id_dsa）:直接回车则 id_dsa.pub 保存到/home/ff/.ssh/目录下，在设置密钥时 cat /home/ff/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys/home/ff/.ssh/是上一步 id_dsa.pub 保存时的目录。

（3）在 Cygwin 控制台中执行 ssh-keygen -t rsa 命令，将为 Master 节点的当前用户 ff 生成其密钥对，密钥对的保存路径使用缺省的/home/ff/.ssh/id_rsa, 要求输入 passphrase 的时候，直接回车。这样生成的证书以及公钥将存储在/home/ff/.ssh 目录，形成两个文件 id_rsa，id_rsa.pub。然后将 id_rsa.pub 文件的内容复制到每一台机器（包括本机）的 /home/ff/.ssh/authorized_keys 文件的尾部，密钥就设置成功了。

（4）启动 SSH 服务，测试连接。从 Master 节点分别向各个 Slave 节点发起 SSH 连接请求，确保不需要输入密码就能使 SSH 连接成功。通过“ ssh 计算机名 ”命令，连接远程计算机，如未提示输入密码，则 SSH 服务就配置成功了。

5.2.2 Hadoop 核心模块配置

（1）平台参数配置 :解压 hadoop-0.19.1.jar 文件，打开 conf 文件夹编辑 hadoop-env.sh 文件，添加 JDK 安装路径，再编辑 hadoop-site.xml 文件，添加如下 XML 配置：

```
<property>
<name>fs.default.name</name>
<value>hdfs://ff:9000/</value>
</property>
<property>
<name>mapred.job.tracker</name>
<value>ff:9001</value>
</property>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
```

参数 fs.default.name 指定 NameNode 的计算机名和端口号。参数 mapred.job.tracker 指定 JobTracker 的计算机名和端口号。参数 dfs.replication 指定 HDFS 中每个 Block 被复制的次数，起数据冗余备份的作用。在本文的系统中，设置为 3。

（2）配置 Masters 节点列表和 Slaves 节点列表，Hadoop 平台通过这两个列表保证

Master 节点对所有 Slave 节点的控制，以及 Slave 节点之间的协同计算。在 4 节点实验平台环境下，Masters 节点列表包含一台计算机名为 ff 的计算机；Slaves 节点列表包含 3 台计算机名分别为 WYW、cien 和 yhx 的计算机。

（3）将配置好的文件复制到所有节点中，执行格式化操作，建立一个新的分布式文件系统。在 Cygwin 控制台中输入 bin/hadoop namenode -format，平台会自动格式化所有 Slave 节点，并在 Hadoop 所在盘的根目录下创建 tmp/hadoop-ff 和 tmp/hadoop-SYSTEM 目录，用来存放执行过程中的数据和结果。

（4）启动 Hadoop 守护进程。在 Cygwin 控制台中输入 bin/start-all.sh，Hadoop 平台自动在 Master 节点上启动 NameNode、Secondary-NameNode 和 JobTracker 守护进程，在所有 Slave 节点启动 DataNode 和 TaskTracker 守护进程，同时记录日志。通过 ps –ef 命令可以看到系统中启动的 Java 进程。Hadoop 平台启动过程如图 5-1。

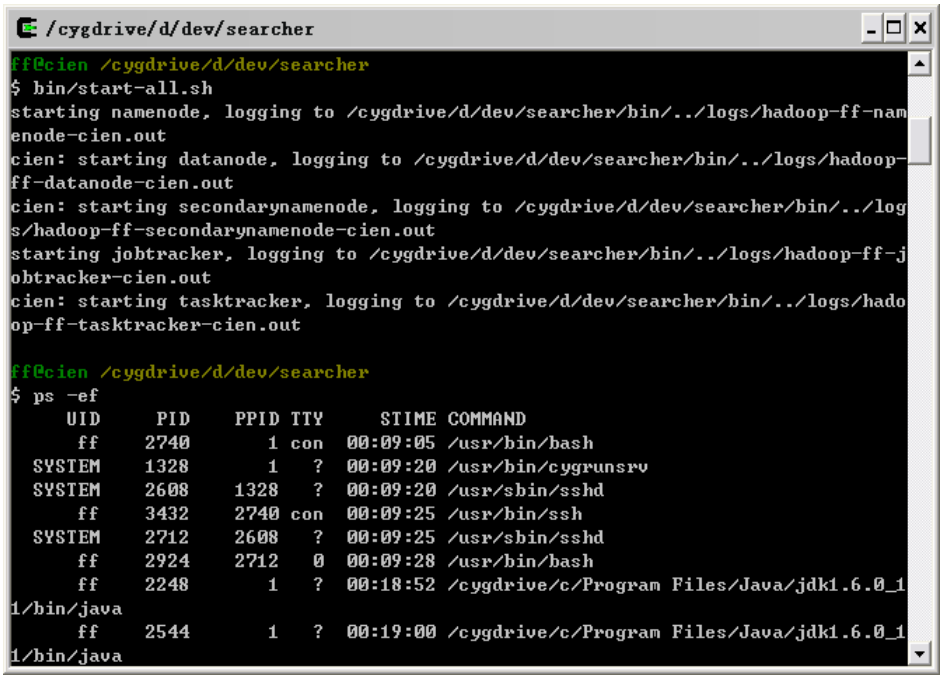
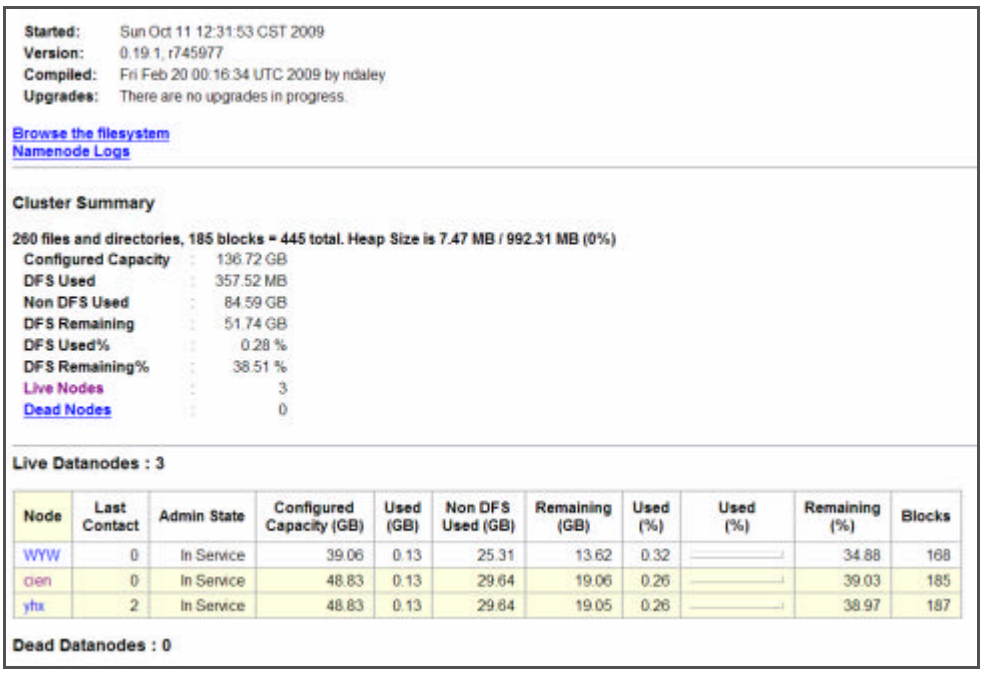


图 5-1 Hadoop 启动过程

Figure 5-1 Hadoop Startup Procedure Diagram

（5）查看 NameNode 和 JobTracker 的网络监视器，它们的默认访问方式为：NameNode - http://ff:50070/ 和 JobTracker - http://ff:50030/。当 Hadoop 分布环境配置成功后，在浏览器中输入：http://ff:50070/，可以看到如图 5-2 所示的运行情况图。



Started: Sun Oct 11 12:31:53 CST 2009
Version: 0.19.1, r745977
Compiled: Fri Feb 20 00:16:34 UTC 2009 by ndaley
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

260 files and directories, 185 blocks = 445 total. Heap Size is 7.47 MB / 992.31 MB (0%)

Configured Capacity	136.72 GB
DFS Used	357.52 MB
Non DFS Used	84.59 GB
DFS Remaining	51.74 GB
DFS Used%	0.28 %
DFS Remaining%	38.51 %
Live Nodes	3
Dead Nodes	0

Live Datanodes : 3

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
WYW	0	In Service	39.06	0.13	25.31	13.62	0.32		34.88	168
cien	0	In Service	48.83	0.13	29.64	19.06	0.26		39.03	185
yhx	2	In Service	48.83	0.13	29.64	19.05	0.26		38.97	187

Dead Datanodes : 0

图 5-2 4 节点的 Hadoop 分布式系统

Figure 5-2 4-Nodes Hadoop Distributed System Diagram

图中所示的是 Hadoop **分布式平台的监控信息，可以看出** Hadoop **分布式平台成功启动** 3 **个** Slave **节点，计算机名分别为** WYW、cien **和** yhx**，节点列表显示了每个节点的运行状况和存储空间使用情况。**

（6**）分布式平台配置成功后，将已经打包为** Search.jar **的系统，拷贝到所有节点的** Hadoop **安装目录下面，分布式搜索引擎系统的部署就完成了。**

5.3 测试数据分析

5.3.1 系统性能测试

搜索引擎业务的特点是：系统查询反馈快，网站爬行和索引构建时间长。由于查询时间很短，容易出现误差，所以，在性能测试阶段选取了执行时间长，误差相对较小的爬行和索引任务的总时间。为了体现不同 Slave **节点数量对系统性能的影响，在实验过程中分别启动了一台、两台和三台** Slave **节点，构成** 2、3、4 **节点的分布式环境。**

爬行的网站限定为 www.163.com**，每个** Slave **节点配置为启动** 2 **个** Map **进程和** 1 **个** Reduce **进程，每个** Map **启动** 10 **个下载线程。**Ping www.163.com **服务器的网络延迟平均为** 60~70ms**。第一次爬行开始于** 2009 **年** 10 **月** 11 **日，第二次爬行开始于** 2009 **年** 10 **月** 12 **日，由于网站信息更新，第二次爬行网页数量比第一次多** 231 **个，但对结果影响不大。**

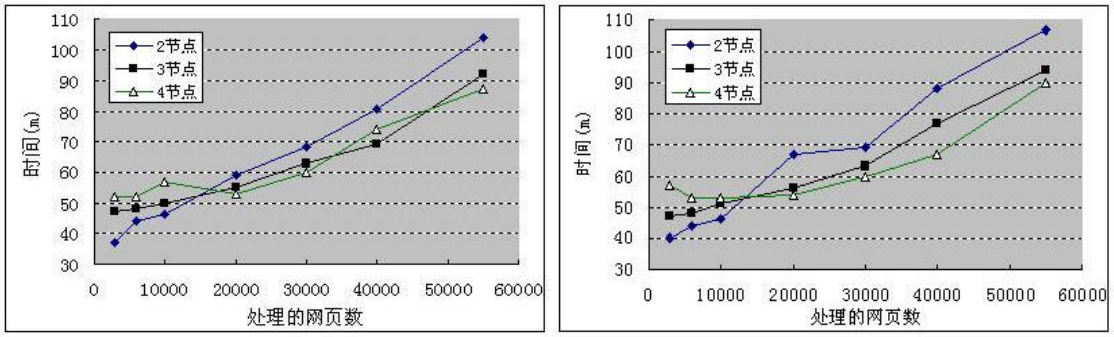


图 5-3 性能测试结果

Figure 5-3 Performance Test Result Chart

从图 5-3 中可以看出，在网页量较小时，2 节点比 4 节点效能更高。这是因为在 2 节点情况下，JobTracker 无需做分割任务运算，所有任务集中在单一 TaskTracker 运行，不需要与其他 TaskTracker 传递数据，与 JobTracker 通信的数据量也少。而随着处理网页量的增加，4 节点系统处理效能比 2 节点逐渐提高， 3 节点系统性能基本处于两者之间。实验证明随着系统节点数量的增加，系统性能在一定程度上得到了提高。

同时也发现 JobTracker 没有好的任务动态分割算法，在一些节点的 Map 结束时，另一些节点还未完成，导致 Reduce 处于等待状态，拖慢了整个系统的性能。

5.3.2 系统可靠性测试

Hadoop 平台的 HDFS 文件副本参数是设置文件块 Block 被复制的次数，起数据冗余和备份的作用。在本文所讨论的系统中将其设置为 3，意味着同一时间，每个 Block 在 3 台 Slave 节点中都会有副本，保证了数据的完整性。在测试中，随机选择了计算机名为 yhx 的计算机手动关闭，模拟任一 Slave 节点故障环境，运行监测结果如图 5-4 所示：

Live Datanodes : 2

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
WYW	0	In Service	39.06	0.13	25.31	13.62	0.32		34.88	168
cien	0	In Service	48.83	0.13	29.64	19.06	0.26		39.03	185

Dead Datanodes : 1

Node
yhx

图 5-4 一节点故障系统状态

Figure 5-4 One Node Failure System Status Diagram

从如 5-4 中可以看出，3 个 Slave 节点中，计算机名为 WYW 和 cien 的节点运行正常，计算机名为 yhx 的节点，已脱离系，成为死节点。此时，对已完成的包含 58730 网

页的索引库进行查询操作，针对不同关键词进行查询，结果如表 5-1：

表 5-1 可靠性测试结果

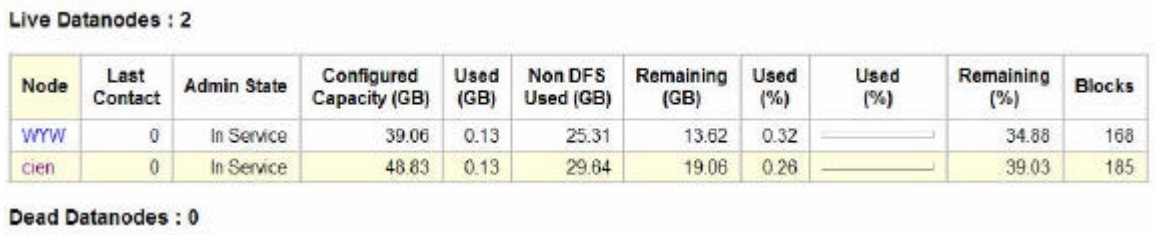
Table 5-1 Reliability Test Result

关键词	4 节点环境	1 节点故障
汽 车	259	259
军 事	227	227
大 学	198	198

可以看到任一 Slave 节点故障，都不会影响到系统的查询结果的完整性，证明系统稳定可靠。

5.3.3 系统扩展性测试

在扩展性测试中，首先构建了 3 个节点系统，Master 节点计算机名为 ff，两台 Slave 节点计算机名分别为 WYW 和 cien，系统状态如图 5-5 所示：



Live Datanodes : 2

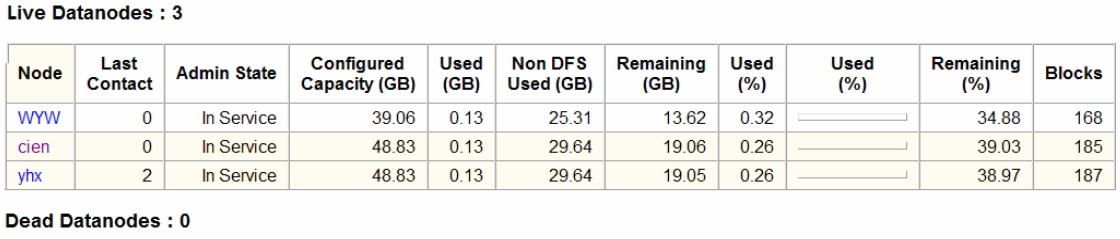
Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
WYW	0	In Service	39.06	0.13	25.31	13.62	0.32		34.88	168
cien	0	In Service	48.83	0.13	29.64	19.06	0.26		39.03	185

Dead Datanodes : 0

图 5-5 3 节点系统状态

Figure 5-5 3-Node System Status Diagram

为了添加新节点，扩展系统规模（要添加的节点计算机名为 yhx），只需将计算机名 yhx 的新节点添加到 Master 节点的 Slave 列表中，格式化 NameNode 并重启分布式平台，新节点就已经加入了平台。因此，系统规模很容易就可以得到扩充，如图 5-6 所示：



Live Datanodes : 3

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
WYW	0	In Service	39.06	0.13	25.31	13.62	0.32		34.88	168
cien	0	In Service	48.83	0.13	29.64	19.06	0.26		39.03	185
yhx	2	In Service	48.83	0.13	29.64	19.05	0.26		38.97	187

Dead Datanodes : 0

图 5-6 4 节点系统状态

Figure 5-6 4-Node System Status Diagram

5.3.4 中文分词测试

对分词模块进行分词测试，输入“毫无疑问，搜索引擎已经成为互联网上人们获取信息的最重要手段。目前，搜索引擎研究一直都处于持续升温状态。”这样一断中文，

分词后效果如图 5-7 **所示：**

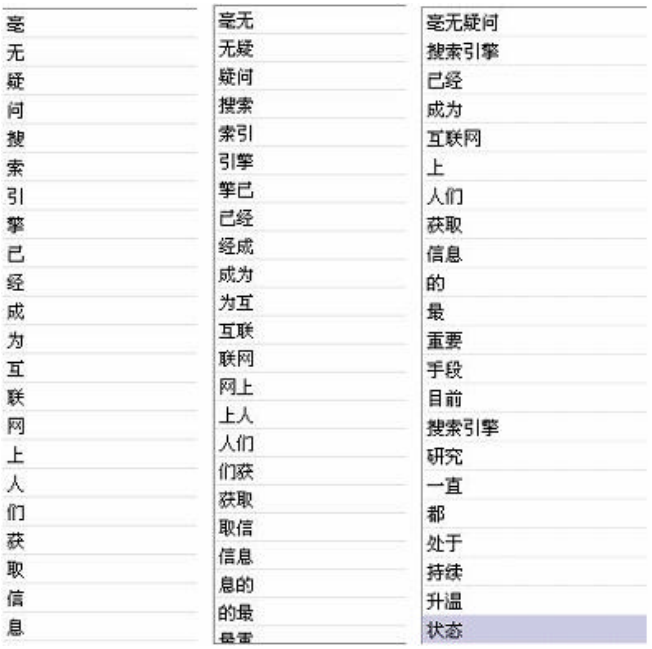


图 5-7 三种分词方法效果对比

Figure 5-7 Three Analyzer Result Comparison Chart

从图中可以很直观地看到采用智能分词后，字（词）从单字切分法的 51 **个字，二元切分法的** 43 **个词，到智能切分法的** 22 **个词，词汇数量有大幅下降，准确率大幅提高，也就意味着索引效率的提高。实验结果表明，对于搜索引擎系统，采用智能分词是非常有必要的。**

5.4 本章小结

本章介绍了部署分布式搜索引擎系统的步骤，在系统成功运行后，测试系统在不同条件下的运行数据，经分析后得出结论。

第六章 总结与展望

近年来，分布式搜索引擎这一新兴的研究领域已成为一个大的研究热点。它包含了分布式计算，全文检索，中文分词，以及查询结果优化等一系列技术。其中基于 Map/Reduce 的编程模型由于其在分布式计算方面有着巨大的潜力，对它的研究也是日新月异。本文采用开源的 Hadoop 分布式计算平台，基于 Map/Reduce 编程模型改进现有搜索引擎的模块，使其具有分布式处理的能力，但受科研条件所限只建立了 4 节点系统。通过实验证明，该技术是可行的。

6.1 工作总结

本文介绍了目前搜索引擎技术的研究现状，分析了几种分布式搜索引擎的关键技术，总结了它们的优缺点，在此基础上引入一种全新的分布式编程模型 Map/Reduce，分析了这种分布式编程的基本运行原理，介绍了其开源实现平台 Hadoop，在此基础上提出了基于 Hadoop 的分布式搜索引擎系统。本文围绕系统的设计与实现，做了以下工作：

（1）介绍了 Map/Reduce 编程模型的运行原理、特点和应用领域，进而分析了其开源实现平台 Hadoop 的系统结构，节点功能，接着介绍了倒排文档基本原理和全文检索技术。

（2）根据搜索引擎的业务特点，将系统划分为三个相对独立的子系统，并分析子系统工作原理，以及各子系统的执行流程，包含的模块设计与实现。

（3）在 Hadoop 分布式计算平台上，扩展 Lucene 全文检索框架，实现子系统各个功能模块。模块的实现符合 Map/Reduce 编程模型规范，可分布式运行。

（4）通过实验，证明 Map/Reduce 模型应用于搜索引擎可以提高系统性能、可靠性和扩展性。

6.2 展望

随着计算机技术的发展，其计算能力进一步提高，集群规模进一步扩大，未来 Hadoop 分布式计算平台会有更大的发展空间。但是在研究过程中也发现，Hadoop 平台的

JobTracker 还缺乏好的任务分割与调度算法，偶尔会出现一些 Slave 节点满载，而另一些节点空闲的情况。今后通过引入更加智能的动态负载均衡机制，添加 JobTracker 动态任务分割与调度算法，充分利用节点计算能力。同时，改进中文分词和网页评分策略，那么将构建出性能更好、搜索准确率更高的分布式搜索引擎。

总之，分布式搜索引擎是当今最具发展潜力的科学技术之一，它的出现极大的改变了人们获取信息的途径，其未来的应用推广也必将让我们的生活方式发生重大转变。

参考文献

[1] 周洪美.2009年中国搜索引擎用户市场调查报告[R].北京：正望咨询，2009.

[2] 张卫丰，徐宝文，周晓宇，李东，许蕾.Web搜索引擎综述[J].计算机科学，2001，28(9)：24-28.

[3] 蒋建洪.主要分布式搜索引擎技术的研究[J].科学技术与工程，2007，7(10)：2418-24241.

[4] 徐高潮.分布计算系统[M].北京：高等教育出版社，2004：12-27.

[5] 董华山，孙济庆.基于P2P的分布式检索模式的研究[J]. 情报学报，2004，23(6)：683-688.

[6] 陈国良.并行计算：结构、算法、编程(第2版)[M].北京：高等教育出版社，2003：59-87.

[7] 卡勒(Culler，D.E.)等著，李晓明等译.并行计算机体系结构：硬件/软件结合的设计与分析（第2版）[M].北京：机械工业出版社，2002：10-23.

[8] 彭洪汇，林作栓.Internet上的搜索引擎和元搜索引擎[J].计算机科学，2002，29(9)：30-32.

[9] Dean J，Ghemawat S.Map/Reduce：Simplied Data Processing on Large Clusters[C].San Francisco：OSDI 2004，2004：137-1501.

[10] Jeff Dean.Experiences with MapReduce，an Abstraction for Large-Scale Computation[R].Proc.15th International Conference on Parallel Architectures and Compilation Techniques，2006.

[11] Owen O'Malley.Programming with Hadoop's Map/Reduce[R].ApacheCon EU，2008.

[12] Ralf Lammel，Data Programmability Team.Google's MapReduce Programmig Model—Revisited[R].Redmond，WA，USA：Microsoft Corp.2007.

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

[14] Dhruba Borthakur. Hadoop Distributed File System[EB/OL].http://hadoop.apache.org/，2007-06.

[15] Yang H C，Dasdan A，Hsiao R L，etc.Map-Reduce-Merge：Simplified Relational Data Processing on Large V lusters [C].International Conference on Management of Data Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data.2007：1029-1040.

[16] Tom White.Hadoop The Definitive Guide[M].United States of America：OReilly，2009：35-87.

[17] 曹羽中.用Hadoop进行分布式并行编程，第1部分 [EB/OL].http://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop1/index.html，2008-05-22.

[18] 朱珠.基于Hadoop的海量数据处理模型研究和应用[D].北京：北京邮电大学，2008.

[19] 申展，江宝林，陈祎.全文检索模型综述[J].算机科学，2004，37(5)：61-64.

[20] 韩升，刘广志.全文检索系统的数据预处理研究[J].计算机技术与发展，2006，16(3)：208-21.

[21] 管建和，甘剑峰.基于Lucene全文检索引擎的应用研究与实现[J].计算机工程与设计，2007，28(2)：489-491.

[22] Chooi-Ling，Masayuki Asahara，Yuji Matsumoto.Pruning False Unknown Words to Improve Chinese

Word Segmentation[A].Proceedings of the 18th Pacific Asia Conference on Language，Information and Computation[C].Tokyo：Logico-linguistic Society of Japan，2004：139-149.

[23] MARTUN KOSTER.Robot in the web：threat or treat[J].ConneXions，1995，9(4)：5-12.

[24] 王斌，张刚，孙健.大规模分布式并行信息检索技术[J].信息技术快报，2005，3(2)：1-91.

[25] 姚树宇，赵少东.一种使用分布式技术的搜索引擎[J].计算机应用与软件，2005，22(10)：127-129.

[26] 苏频，李凡长.基于DFS的并行粒计算模型及应用[J].广西师范大学学报，自然科学版，2006，24(4)：66-69.

[27] 郎小伟，王申康.基于Lucene的全文检索系统研究与开发[J].计算机工程，2006，32(4)：94-99.

[28] 李晓明，闫宏飞，王继民.搜索引擎[M].北京：科学出版社，2005：22-36.

[29] 秋哲，符滔滔.开发自己的搜索引擎Lucene2.0+Heritrix[M].北京：人民邮电出版社，2007：61-97.

[30] 钱兵，王永成.面向搜索引擎的自然语言理解的设计与实现[J].计算机应用研究，2006，23(12)：260-262.

[31] 屈培，葛蓁.Nutch-0.8.1中文二分法中文分词的实现[J].计算机时代，2007，7：71-73.

[32] 潘以锋.基于Lucene的网站全文检索系统的开发[J].广西教育学院学报，2006，85(5)：63-66.

[33] Erik Hatcher，Otis Gospodnetic .Lucene in Action (In Action series)[M].USA：Manning Publications Co.，2004：107-154.

[34] W.Clay Richardson，Donald Avondolio.Professional Portal Development with Apache Tools：Jetspeed，Lucene，James，Slide[J].John Wiley&Sons，2004：59-67.

[35] Chau M，Qin Jialun.SpidersRUs：Automated development of vertical search engines in different domains and languages[C].Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries，2005：110-111.

致 谢

值此论文完成之际，首先向我尊敬的导师胡彧致以最诚挚的感谢和无比的敬意。胡老师严谨的治学作风、渊博的学识，对知识锲而不舍追求以及优秀的人格魅力，给了我深刻的影响，使我受益非浅，是我毕生学习的典范，并将成为我今后献身科学事业的动力。三年的硕士生涯中，导师为我创造了良好的研究和学习环境，并在生活上、思想上给了我极大的帮助、关心和鼓励。本文是在导师胡彧的悉心指导下完成的，胡老师在论文各个阶段富有启发性的意见使本文得以顺利完成。

感谢太原理工大学，感谢计算机与软件学院，学 校三年来给我提供的良好的学习环境、便利的科研条件、完善的服务体系，是我得以顺利完成学业的有力保障。在三年的学习和科研工作中，得到了学院各位老师多方面的指导和帮助，感谢他们的大力支持和无私关怀。

感谢我的父母以及关心支持我的亲人、朋友，他们在我攻读研究生学位期间给了我最无私的帮助、关心和鼓励。我还要感谢我的同门师兄妹，沈晓瑞、康振华、梁学贤、毕晋芝、卫培培以及同一教研室的许多同学，他们在我的论文完成过程中给予了我很大的启迪、支持和热情的帮助。在此，我向他们一并表示真挚的感谢。也向一切给予我帮助，使我顺利完成硕士研究生学业的人们表示衷心的谢意。

最后，衷心感谢为评阅本论文而付出辛勤劳动的各位专家和学者。

攻读硕士学位期间发表的学术论文目录

[1] 胡彧，封俊.Hadoop下的分布式搜索引擎研究[J].计算机系统应用，2010.

基于Hadoop的分布式搜索引擎研究与实现

作者： 封俊
学位授予单位： 太原理工大学
被引用次数： 3次