



# 小型搜索引擎的设计与实现

姓名：孙振强  
学院：计算机与电子信息学院 / 人工智能学院  
日期：2021 年 06 月 20 日

## 摘要

本项目设计并实现了一个小型的搜索引擎系统，支持对“南京师范大学”的主站点、阳光网、研究生院网站、所有 28 个二级学院网站进行全文检索。

系统采用双节点的物理架构，使用千兆网络在局域网内连接。系统采用分布式的数据架构，使用 HDFS 和 HBase 作为底层数据存储结构。系统数据分为五个部分，分别为：非结构化的 RAW 数据、SEGMENT 数据、PAGERANK 数据，和结构化的 NNU\_PAGES 表数据、NNU\_WORDS 表数据。系统包含四个功能模块，分别为：爬虫模块、中文分词模块、PageRank 模块、检索模块。

爬虫模块使用 Java 多线程技术从互联网上爬取并解析网页数据。模块使用 Jsoup 库操作 HTML 文档，使用 UUID 库实现网页判重功能，输出 RAW 数据，并迁移到 NNU\_PAGES 表中。共计爬取到总大小为 1.2 GB 的近 20 万个网页。

中文分词模块使用 MapReduce 技术构建倒排索引。模块使用开源的“结巴分词”库实现分词操作，输入 RAW 数据，输出 SEGMENT 数据，并迁移到 NNU\_WORDS 表中。共计输出总大小为 4.2 GB 的 53 万余个关键词。

PageRank 模块使用 MapReduce 技术计算每个页面的价值。模块使用两个 MapReduce 任务分别完成构建链接图和迭代计算 PageRank 值的操作，输入 RAW 数据，输出 PAGERANK 数据，并迁移到 NNU\_PAGES 表中。

检索模块使用 Django 库搭建 Web 服务器并提供关键词搜索服务。模块使用 HappyBase 库与 HBase 进行数据通信，输入 NNU\_PAGES 表和 NNU\_WORDS 表。

本项目的代码开源在 GitHub 上，并提供公网演示地址。

最后，总结了自己在系统研发过程中遇到的困难，提出了系统未来有待完善的地方，并谈论了自己在本项目中的收获和感悟。

**关键词：**搜索引擎；Hadoop；MapReduce；HDFS；HBase；PageRank

## Abstract

This project has designed and implemented a small search engine system to support full-text search on some websites within doamin "Nanjing Normal University", including the Main Website, the Sunshine Website, the website of Graduation, and all 28 secondary school websites.

The system adopts a two-node physical structure and uses a gigabit network to connect in the local area network. The system adopts a distributed data architecture and uses HDFS and HBase as the underlying data storage structure. The system data is divided into five parts, namely: unstructured `RAW` data, `SEGMENT` data, `PAGERANK` data, and structured `NNU_PAGES` table data, `NNU_WORDS` Table data. The system contains four functional modules, namely: crawler module, Chinese word segmentation module, PageRank module, and retrieval module.

The crawler module uses Java multi-threading technology to crawl and parse webpage data from the Internet. The module uses the `Jsoup` library to manipulate HTML documents, uses the `UUID` library to check the duplication of webpages, outputs the `RAW` data, and migrates it to the `NNU_PAGES` table. A total of nearly 200,000 web pages with a total size of 1.2 GB were crawled.

The Chinese word segmentation module uses MapReduce technology to build an inverted index. The module uses the open source "Jieba" library to implement word segmentation, input `RAW` data, output `SEGMENT` data, and migrate to the `NNU_WORDS` table. In total, more than 530,000 keywords with a total size of 4.2 GB are output.

The PageRank module uses MapReduce technology to calculate the value of each webpage. The module uses two MapReduce tasks to construct the link graph and iteratively calculate the PageRank value respectively, input `RAW` data, output `PAGERANK` data, and migrate to the `NNU_PAGES` table.

The retrieval module uses the Django library to build a Web server and provide keyword search services. The module uses the `HappyBase` library to communicate with HBase and visit the `NNU_PAGES` table and the `NNU_WORDS` table.

The code of this project is open source on GitHub, and the public network demo address is provided.

Finally, I summarized the difficulties I encountered in the process of system research and development, put forward the areas where the system needs to be improved in the future, and talked about my own gains and insights in this project.

**Key words:** Search Engine; Hadoop; MapReduce; HDFS; HBase; PageRank

## 目录

<b>1 综述</b>	<b>1</b>
<b>2 系统架构</b>	<b>2</b>
2.1 物理架构 . . . . .	2
2.2 HDFS 架构 . . . . .	3
2.3 HBase 架构 . . . . .	5
2.4 数据架构 . . . . .	6
2.5 业务架构 . . . . .	7
<b>3 功能模块设计</b>	<b>7</b>
3.1 爬虫模块 . . . . .	7
3.1.1 爬虫范围 . . . . .	8
3.1.2 爬虫过程 . . . . .	8
3.1.3 RAW 数据 . . . . .	10
3.1.4 迁移到 HBase . . . . .	10
3.2 中文分词模块 . . . . .	10
3.2.1 MapReduce 设计 . . . . .	11
3.2.2 SEGMENT 数据 . . . . .	12
3.2.3 迁移到 HBase . . . . .	12
3.3 PageRank 模块 . . . . .	12
3.3.1 MapReduce 设计——构建链接图 . . . . .	13
3.3.2 MapReduce 设计——计算 PageRank 值 . . . . .	13
3.3.3 PageRank 数据 . . . . .	15
3.3.4 迁移到 HBase . . . . .	16
3.4 检索模块 . . . . .	16
3.4.1 连接 HBase 数据库 . . . . .	16
3.4.2 检索过程 . . . . .	17
3.4.3 搭建服务器 . . . . .	19
<b>4 项目演示</b>	<b>19</b>
<b>5 遇到的困难</b>	<b>19</b>
<b>6 未来的改进</b>	<b>20</b>
<b>7 自己的收获</b>	<b>21</b>
<b>参考文献</b>	<b>22</b>

## 1 综述

网页搜索引擎 (Web Search Engine) 或互联网搜索引擎 (Internet Search Engine) 是一种用于网络搜索 (互联网搜索) 的软件系统。搜索引擎在万维网 (World Wide Web) 上以文本网页搜索的方式获取特定信息。搜索结果通常以一行行的形式显示, 其中每行结果为一个网页, 该页面被称为搜索引擎结果页面 (SERPs, Search Engine Results Pages)。信息包含指向网页、图像、视频、图表、文章、论文及其它类型文件的链接。一些搜索引擎还可以挖掘数据库或开放目录中的数据。不同于仅由人工维护的网页目录, 搜索引擎使用网络爬虫来实时维护信息。不能被网页搜索引擎搜索到的互联网内容通常被称为“深网” (Deep Web) [1]。

目前使用较为广泛的搜索引擎有 Google、Bing、Baidu 等。其中, Google 和 Bing 搜索引擎在全球广泛使用, 支持世界上几乎所有语种的语言, 提供了对于世界上几乎所有网站的搜索服务。Baidu 搜索引擎在中国广泛使用, 对于中文搜索能返回质量较高的搜索结果。除了传统的文本检索模式 (Text-Web) 外, 这三种搜索引擎都提供了多模态检索和跨模态检索的功能, 例如语音检索 (Voice-Web)、图片检索 (Image-Web)、相似图片检索 (Image-Image) 等功能。

Google、Bing、Baidu 这些搜索引擎属于全文搜索引擎, 此外还有目录搜索引擎、垂直搜索引擎、元搜索引擎 [2], 下面简要介绍:

- 目录搜索引擎 (Index Search Engine): 预先设有明确的层级框架, 采用人工或半自动的方式检索网页, 将网页归档到相应的的目录下。常见的有导航页面和门户网站等;
- 全文搜索引擎 (Full Text Search Engine): 预先爬取网页信息, 并建立关键词索引, 后根据用户查询的关键词检索数据库, 返回相关页面。这是当今主流的搜索引擎, 常见的有百度和谷歌等;
- 垂直搜索引擎 (Vertical Search Engine): 针对某个特定领域而建立的专用搜索引擎。相比于全文搜索引擎, 垂直搜索引擎的检索范围很小, 但能够深入挖掘该领域的专业信息, 检索质量较高。常见的有种子搜索引擎、小说搜索引擎、表情包搜索引擎等;
- 元搜索引擎 (Meta Search Engine): 整合多个搜索引擎的结果, 并在一个界面上集中展示。元搜索引擎通常没有自己的网络爬虫和数据库, 一般实时汇总几个不同搜索引擎的搜索结果。常见的有 360 综合搜索等;

此外, 一般网站都提供有站内搜索功能, 这可以看成是一个仅搜索指定站点的小型搜索引擎。例如, 在社交媒体网站中通过关键字搜索动态, 在电商网站中通过关键词搜索相关商品, 在视频网站中通过关键字搜索视频, 在新闻网站中通过关键字搜索新闻。

在这个信息爆炸的时代, 互联网上充斥着海量的信息, 而搜索引擎能帮我们在短短几秒钟的时间内, 搜索到我们想要的结果, 这就是搜索引擎的魅力。可以说, 网络时代离不开搜索引擎, 搜索引擎是人们浏览网络世界的最重要的途径。

本项目是小型搜索引擎的设计与实现, 旨在设计一套完整的搜索引擎系统, 实现一个针对特定网站真实可用的搜索引擎系统。本项目紧密围绕课程核心内容, 使用 HDFS 和 HBase 进

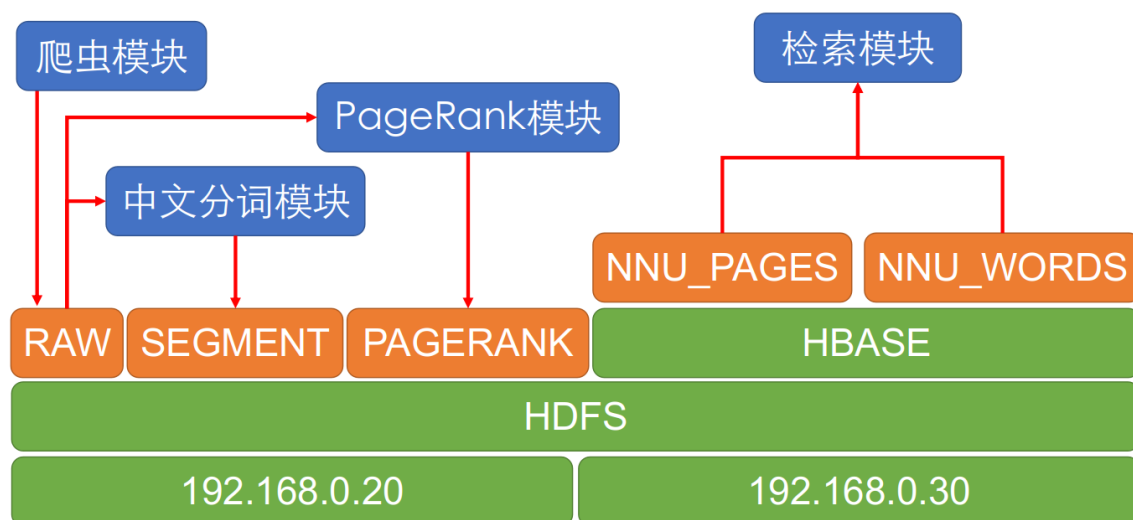


图 1: 系统架构示意图

行数据存储，使用 Hadoop MapReduce 进行数据处理，将并程序设计的思想充分融入到系统的设计之中。本文完全阐述了系统的设计与实现工作，以介绍系统架构开始，再具体介绍四个系统功能模块，后叙述项目进行过程中遇到的困难，最后提出系统在未来的改进之处，并谈谈自己的收获。

## 2 系统架构

本系统采用双节点的服务器集群架构，数据分布式存储在两个节点中，非结构化的数据存储在 HDFS 上，结构化的数据存储在 HBase 上。系统包含四个功能模块，分别是爬虫模块、中文分词模块、PageRank 模块、检索模块。图 1 展示了完整的系统架构。

本部分从物理架构、数据架构、业务架构三个层次对系统架构进行介绍。图 1 中，绿色的方框表示物理架构部分，橙色的方框表示数据架构部分，蓝色的方框表示业务架构部分。由于 HDFS 和 HBase 架构较为复杂，因此我们使用单独的两节内容分别介绍。

### 2.1 物理架构

本系统使用了两台学院配发的台式机作为服务器。台式机配置了 Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 四核四线程的中央处理器, Samsung DIMM DDR4 Synchronous Unbuffered (Unregistered) 2667 MHz 容量为 4 GB 的内存, KINGSTON SV300S3 容量为 120 GB 的固态硬盘, Ubuntu 20.04.2 LTS (Focal Fossa) 的服务器操作系统（无图形化界面）。

两台服务器位于自己路由器搭建的内网中，表 1 列出了两台服务器的相关配置信息。1 号服务器是主节点，HDFS 的 NameNode JPS 和 HBase 的 HMaster、ThriftServer JPS 均部署在 1 号服务器上。

注意，jps 是一条终端命令，可以列出系统中当前运行的 Java HotSpot VMs [3]，表 1 中的 HDFS JPS 和 HBase JPS 分别代表为 HDFS 和 HBase 提供服务的 Java HotSpot VMs。

表 1: 服务器配置表

服务器	1 号服务器	2 号服务器
Hostname	desktop-brown	slaker
IP 地址	192.168.0.20	192.168.0.30
HDFS JPS	DataNode, NameNode, SecondaryNameNode, NodeManager, ResourceManager	DataNode
HBase JPS	HRegionServer, HQuorumPeer, HMaster, ThriftServer	HRegionServer, HQuorumPeer

## 2.2 HDFS 架构

HDFS 是 Hadoop 分布式文件系统 (Hadoop Distributed File System)，支持对应用程序数据进行高吞吐量访问 [4]。HDFS 是 Hadoop 的一个子模块，因此我们需要在 Hadoop 中配置 HDFS。注意，以下过程建立在单节点配置正确的基础上，请先参照 Hadoop 官方文档 [5] 配置单节点环境。

打开 `core-site.xml` 配置文件，配置 HDFS 在 1 号服务器的 9000 端口上启动，HDFS 存储在每台服务器的 `/home/hadoop-${user.name}` 路径下。修改如下：

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://desktop-brown:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/hadoop-${user.name}</value>
  </property>
</configuration>
```

打开 `hdfs-site.xml` 配置文件，将数据副本数量改为一份，即不进行数据冗余备份。修改如下：

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
```

## Datanode Information

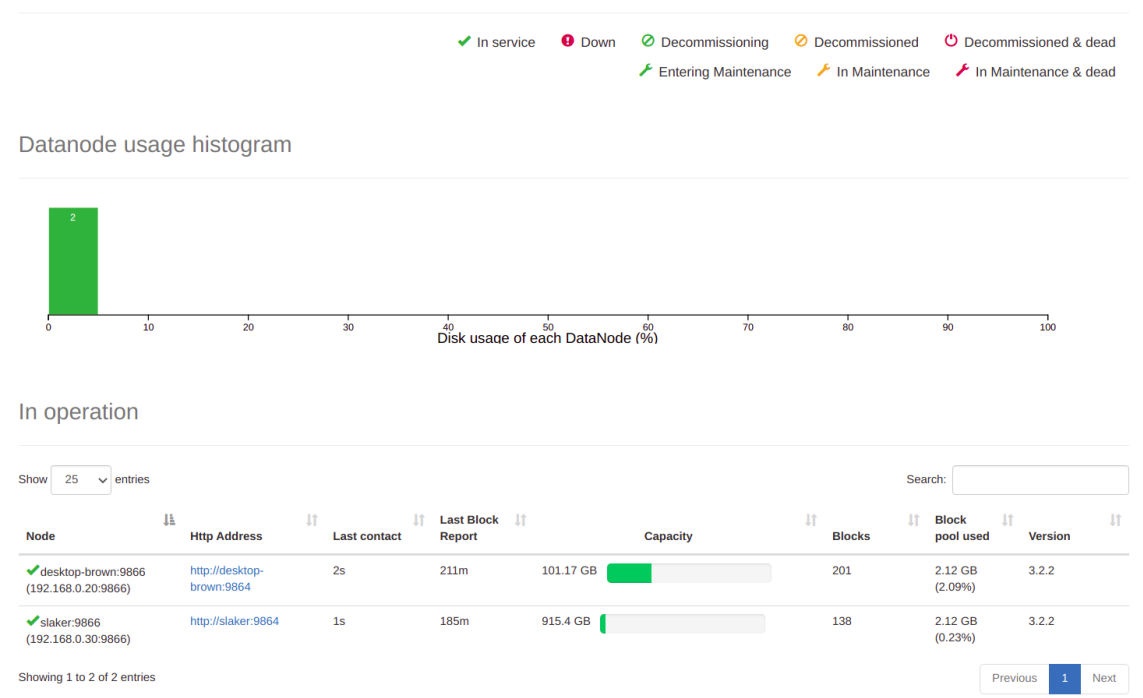


图 2: HDFS 监控页面

```
</property>
</configuration>
```

打开 `worktars` 配置文件，添加两台服务器的 hostname：

```
desktop-brown
slaker
```

至此，我们将单节点的 HDFS 环境变成了双节点的 HDFS 环境。下面，在 1 号服务器上执行命令格式化 HDFS：

```
./bin/hdfs namenode -format
```

接着，执行命令启动 HDFS 和管理器：

```
./sbin/start-all.sh
```

我们可以在浏览器中访问 `http://192.168.0.20:9870/`，查看 HDFS 的运行状态。图 2 展示了 Namenode 的相关信息，我们可以看到，Namenode 在两台服务器上均已启动并运行。



## 2.3 HBase 架构

HBase 是 Hadoop 数据库 (Hadoop Database), 是一个分布式的可拓展的大数据存储数据库 [6]。下面, 我们对 HBase 的配置过程进行简要说明。注意, 以下过程省略了在 `hbase-env.sh` 文件中对 Java 路径的配置。

打开 `hbase-site.xml` 配置文件, 分别配置启动分布式集群部署、路径为 HDFS 根目录下的 `code` 文件夹、zookeeper 要管理的两台服务器的 `hostname`、zookeeper 数据存储路径。修改如下:

```
<configuration>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://desktop-brown:9000/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>desktop-brown,slaker</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/zookeeper</value>
  </property>
</configuration>
```

打开 `regionservers` 配置文件, 添加两台服务器的 `hostname`:

```
desktop-brown
slaker
```

至此, 我们完成了双节点 HBase 的配置工作。下面, 在 1 号服务器上执行命令启动 HBase:

```
./bin/start-hbase.sh
```

接着, 执行命令启动 Thrift 服务器:

```
./bin/hbase thrift start
```

我们可以在浏览器中访问 `http://192.168.0.20:16010/`, 查看 HBase 的运行状态。图 3 展示了 HBase 的相关信息, 我们可以看到, Region Servers 在两台服务器上均已启动并运行。

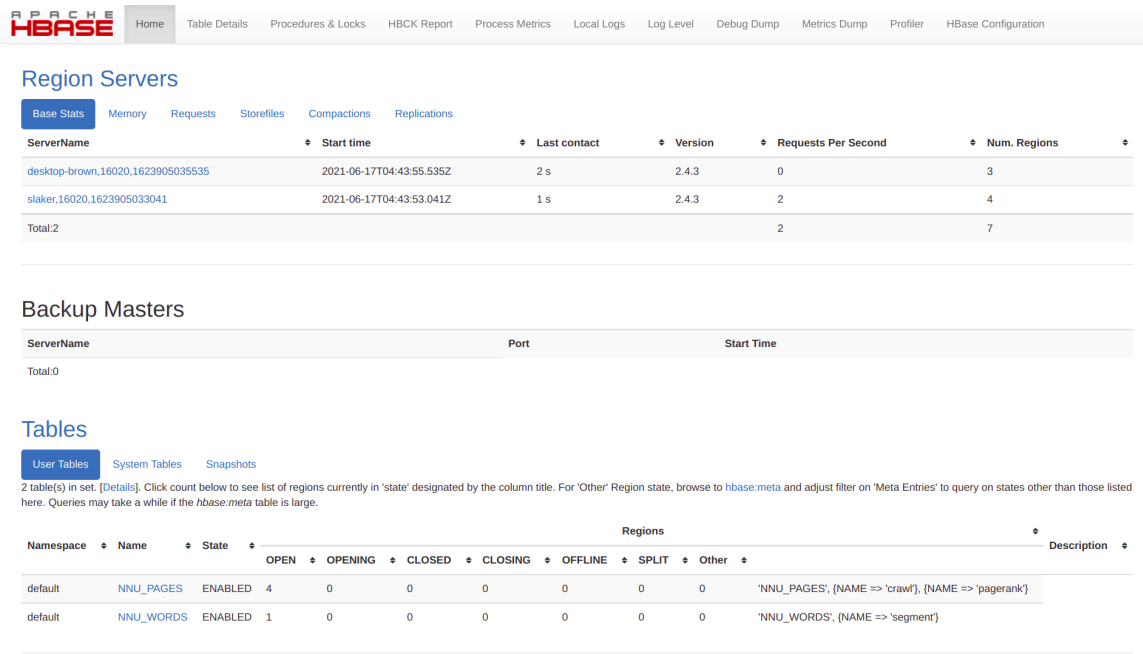


图 3: HBase 监控页面

2.4 数据架构

系统的所有数据均存储在 HDFS 和 HBase 上。其中，非结构化数据 RAW、SEGMENT、PAGERANK 存储在 HDFS 上，结构化数据 NNU\_PAGES、NNU\_WORDS 存储在 HBase 上。下面对这五部分的数据进行简要说明：

- **RAW**：爬虫模块爬取得到的网页原始数据，记录了每个页面的 URL、页面内包含的超链接、标题和正文内容；
- **SEGMENT**：经过中文分词模块处理后得到的数据，记录了每个关键词及包含该关键词的所有 URL；
- **PAGERANK**：经过 PageRank 模块处理后得到的数据，记录了每个页面的 URL 及其 PageRank 值；
- **NNU\_PAGES**：HBase 中的一张表，记录了每个页面的所有信息，表结构及样例数据如表 2 所示。该表包含两个 column family **crawl** 和 **pagerank**，分别代表由爬虫模块和 PageRank 模块产生的数据；
- **NNU\_WORDS**：HBase 中的一张表，记录了每个关键词的所有信息，表结构及样例数据如表 3 所示。该表包含一个 column family **segment**，代表由中文分词模块产生的数据。我们会在第 3.2 节中说明 **segment:URLs** 字段的数据格式；

表 2: NNU\_PAGES 表及样例数据

ID	crawl:title	crawl:body	crawl:nextURLs	pagerank:value
www.1.com	title-1	body-1	www.2.com www.3.com	1.64
www.2.com	title-2	body-2	www.1.com	0.88
www.3.com	title-3	body-3		0.32

表 3: NNU\_WORDS 表及样例数据

ID	segment:URLs
hello	www.1.com@@1-5 www.2.com@23-27 www.3.com@0-4
hadoop	www.2.com@@2-7
hbase	www.1.com@@2-6 www.3.com@101-105

## 2.5 业务架构

系统包含四个功能模块，分别处理一个完整的业务逻辑。下面对这四个功能模块进行简要说明：

- 爬虫模块：从互联网上爬取指定站点的网页内容，解析每个页面的 URL、页面内包含的超链接、标题和正文内容，输出 **RAW** 数据，并迁移至 **NNU\_PAGES** 表的 **crawl** 族中；
- 中文分词模块：输入 **RAW** 数据，使用 Hadoop MapReduce 对每个页面的标题和正文内容进行分词，对于每个关键词，收集所有含有该关键词的 URL 及关键词在页面中出现的位置，输出 **SEGMENT** 数据，并迁移至 **NNU\_PAGES** 表的 **segment** 族中；
- PageRank 模块：输入 **RAW** 数据，使用 Hadoop MapReduce 迭代地计算每个页面的 PageRank 值，输出 **PAGERANK** 数据，并迁移至 **NNU\_PAGES** 表的 **pagerank** 族中；
- 检索模块：输入 **NNU\_PAGES** 和 **NNU\_WORDS** 数据和查询关键词，先从 **NNU\_WORDS** 表中筛选出包含该关键词的所有页面，再从 **NNU\_WORDS** 表中读取这些页面的信息，排序并高亮关键词后输出搜索结果；

## 3 功能模块设计

本章我们将详细阐述系统爬虫模块、中文分词模块、PageRank 模块、检索模块这四个功能模块的设计，包括每个模块的输入数据说明、数据处理过程、输出数据说明。

### 3.1 爬虫模块

爬虫模块的目的是从互联网上爬取并解析网页数据，使用 Java 多线程进行并行爬虫作业。开始时，给定入口网页地址和域名限制范围，每次爬取并解析完一个页面后，将页面内容保存

到磁盘中，将该页面添加到“完成集合”中，以后再遇到该页面将不再重复爬取，将该页面中包含的所有超链接加入到“等待队列”中，直到“等待队列”为空才结束。

3.1.1 爬虫范围

本项目的爬虫范围是“南京师范大学”的部分网页，包括南京师范大学的主站点、阳光网、研究生院（研工部）、所有 28 个二级学院的网站。完整的域名范围如表 4 所示。其中，阳光网是南京师范大学发布各类新闻咨询的网站。

表 4: 爬虫的域名限制范围

序号	域名	说明	序号	域名	说明
1	www.njnu.edu.cn	南京师范大学主站点	17	xinchuan.njnu.edu.cn	新闻与传播学院
2	news.njnu.edu.cn	南京师范大学阳光网	18	sfy.njnu.edu.cn	社会发展学院
3	grad.njnu.edu.cn	南京师范大学研究生院（研工部）	19	math.njnu.edu.cn	数学科学学院
4	honors.njnu.edu.cn	强化培养学院	20	physics.njnu.edu.cn	物理科学与技术学院
5	jsjyxy.njnu.edu.cn	教师教育学院	21	hky.njnu.edu.cn	化学与材料科学学院
6	gjy.njnu.edu.cn	国际文化教育学院	22	dky.njnu.edu.cn	地理科学学院
7	jny.njnu.edu.cn	金陵女子学院	23	sky.njnu.edu.cn	生命科学学院
8	spa.njnu.edu.cn	公共管理学院	24	energy.njnu.edu.cn	能源与机械工程学院
9	sxy.njnu.edu.cn	商学院	25	d.njnu.edu.cn	电气与自动化工程学院
10	law.njnu.edu.cn	法学院	26	ceai.njnu.edu.cn	计算机与电子信息学院 / 人工智能学院
11	marx.njnu.edu.cn	马克思主义学院	27	env.njnu.edu.cn	环境学院
12	jky.njnu.edu.cn	教育科学学院	28	hy.njnu.edu.cn	海洋科学与工程学院
13	xlxy.njnu.edu.cn	心理学院	29	spxy.njnu.edu.cn	食品与制药工程学院
14	tky.njnu.edu.cn	体育科学学院	30	music.njnu.edu.cn	音乐学院
15	wxy.njnu.edu.cn	文学院	31	msxy.njnu.edu.cn	美术学院
16	wy.njnu.edu.cn	外国语学院			

3.1.2 爬虫过程

爬虫模块的整体架构如图 4 所示。其中，蓝色 Thread 方框是由 Java 线程池维护的一系列线程，绿色 WAIT\_URL\_POOL 和 FINISH\_URL\_POOL 方框分别是“等待队列”和“完成集合”，橙色 RAW 方框是存储的网页数据。

爬虫过程的算法叙述如下：

1. 设置参数：指定入口页面为南京师范大学首页（http://www.njnu.edu.cn/）并设置域名范围，指定 Java 线程池的线程个数；
2. 初始化：将入口网页添加到 WAIT\_URL\_POOL 中，初始化 Java 线程池；
3. 请求：线程池中的每个线程，若当前为空闲状态，则从 WAIT\_URL\_POOL 中取出一个 URL，请求 html 文档；
4. 解析：从 html 文档中解析出标题（title）、正文（body）、包含的超链接（nextURLs）；
5. 保存：将 URL、title、body、nextURLs 保存到磁盘中；

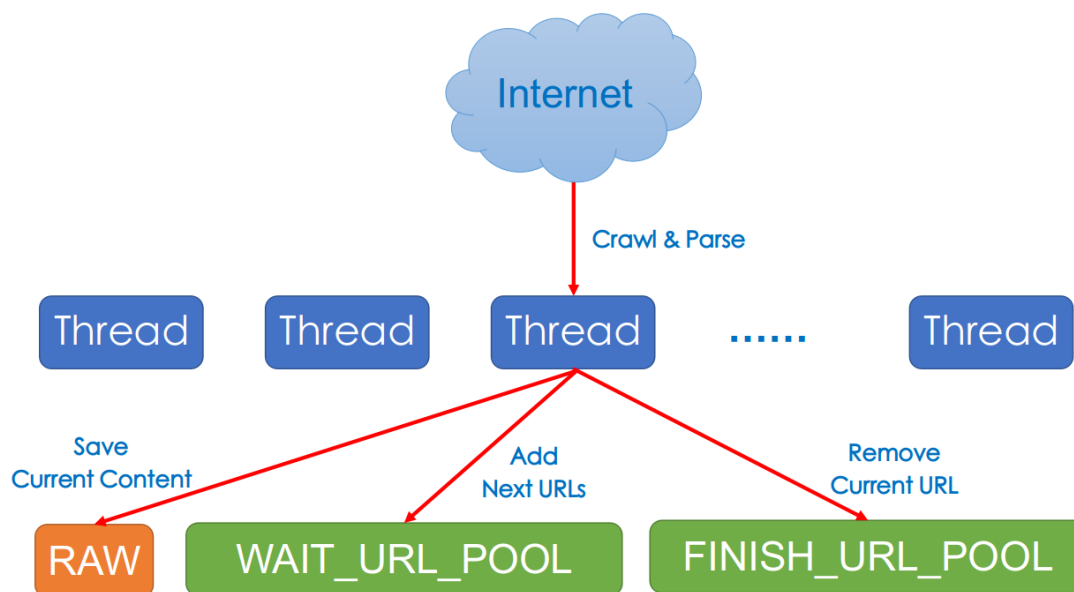


图 4: 爬虫模块架构示意图

6. 添加 nextURLs: 对于 nextURLs 中的每个 nextURL, 若不在 `FINISH_URL_POOL` 中且在域名范围内, 则将 nextURL 添加到 `WAIT_URL_POOL` 中;
7. 移 URL: 从 `WAIT_URL_POOL` 中移除当前 URL;
8. 若 `WAIT_URL_POOL` 不为空, 返回第 3 步;
9. 程序结束;

下面对爬虫程序做几点说明:

- 程序中使用 `Jsoup` 包操作 html 文档, 可以非常方便地进行请求网页、提取超链接、过滤 html 格式等操作;
- 在 `WAIT_URL_POOL` 和 `FINISH_URL_POOL` 中, 使用了 `java.util.UUID` 以加快判重效率;
- 会出现形如 “www.1.com/hello#tab” 的 URL, 后面的 “#tab” 是网页内部的锚点定位, 需要过滤掉, 变成 “www.1.com/hello”, 不然爬虫程序会重复爬取相同页面, 大大增加数据量;
- 在第 5 步保存时, 保存函数 `save()` 使用了 `synchronized` 关键字, 执行串行化存储过程, 以防止多线程出现数据丢失或不同步的问题;

### 3.1.3 RAW 数据

爬虫程序共计爬取了南京师范大学域名下的 198,649 个网页，总大小 1.2 GB。

爬虫得到的数据分块存储在磁盘中，如图 5 所示。具体来说，每个页面需要保存四个字段：URL、nextURLs、title、body，每个页面占用一行。每个文件中存储 10000 个页面，即每个文件有 10000 行。若文件超过 10000 行，则会创建一个新文件来保存数据，文件名按照格式 `part-0`、`part-1`、`part-2` …… 顺序编号。下面展示了一个 `part-*` 文件的结构样例。注意，每个 `[xxx]` 是一个字段名，字段名和内容之间使用分隔符“\t”隔开，因此 title 和 body 中的“\t”字符需要全部过滤掉。

```
[0]      [URL] www.1.com      [nextURLs] www.2.com www.3.com [title] title-1      [body] body-1
[1]      [URL] www.2.com      [nextURLs] www.1.com      [title] title-2      [body] body-2
[2]      [URL] www.3.com      [nextURLs]                [title] title-3      [body] body-3
...
[9999]   [URL] www.9999.com [nextURLs] www.1.com www.8.com [title] title-9999 [body] body-9999
```

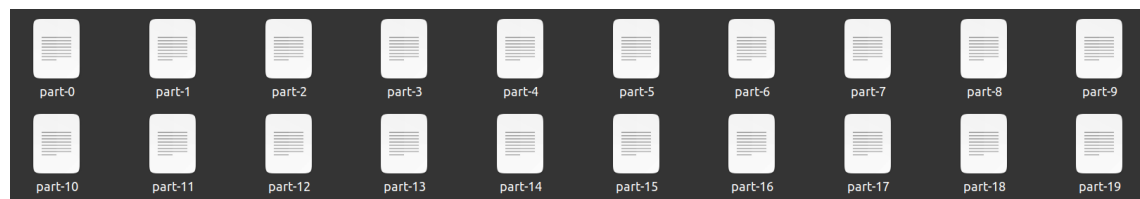


图 5: 分块存储的网页数据

图 6 是南京师范大学首页在文件中的真实数据。可以看到，`[URL]` 字段是 `http://www.njnu.edu.cn/`，`[nextURLs]` 字段中有非常多的超链接，`[title]` 字段是“南京师范大学”，`[body]` 字段是网页中的纯文本内容。

### 3.1.4 迁移到 HBase

最后，我们将分块文件中的数据迁移到 HBase 的 `NNU_PAGES` 表的 `crawl` 族中，`NNU_PAGES` 表的结构如表 2 所示。文件 `[URL]` 字段是每条记录的 ID，文件 `[nextURLs]` 字段对应表中的 `crawl:nextURLs` 字段，文件 `[title]` 字段对应表中的 `crawl:title` 字段，文件 `[body]` 字段对应表中的 `crawl:body` 字段。

## 3.2 中文分词模块

中文分词模块的目的是建立倒排索引，使用 Hadoop MapReduce 进行并行计算。对于每个关键词，会存储所有包含该关键词的网页及该关键词在网页中出现的位置。





图 6: 南京师范大学首页在分块文件中存储的数据

### 3.2.1 MapReduce 设计

图 7 展示了中文分词的 MapReduce 过程。下面我们就 Map 和 Reduce 过程分别进行说明：

在 Map 过程中，每次读入的是一行数据，即一个网页的数据。以图 7 的第一行数据为例，该 Mapper 函数的 **key** 是偏移量，我们不需要用到，**value** 是 “[0] [URL] www.1.com [nextURLs] www.2.com www.3.com [title] title-1 [body] body-1”。从 **value** 中解析出该网页的 title 和 body。

对 title 和 body 进行中文分词，这里我们使用的是开源的“结巴分词 Java 版” [7]。使用结巴分词分别对 title 和 body 进行中文分词操作，得到一个关键词列表，对于每个关键词，若其出现在 title 中，则使用 **url@s-t** 标识位置，若其出现在 body 中，则使用 **url@s-t** 标识位置，其中 **s** 是关键词开始位置的偏移量，**t** 是关键词结束位置的偏移量。

对于该行数据，假设分词后，title 中有一个关键词“title”，body 中有一个关键词“body”，则 Mapper 函数会输出两个键值对 **www.1.com@0-4** 和 **www.1.com@0-3**，表示“title”出现在 www.1.com 中 title 的第 0 - 4 个字符处，“body”出现在 www.1.com 中 body 的第 0 - 3 个字符处。

在 Reduce 过程中，输入的 **key** 是关键词，原样输出，而输入的 **value** 是一个返回出现位置字符串的迭代器，我们将迭代器中的所有字符串用空格拼接起来后，就直接输出。

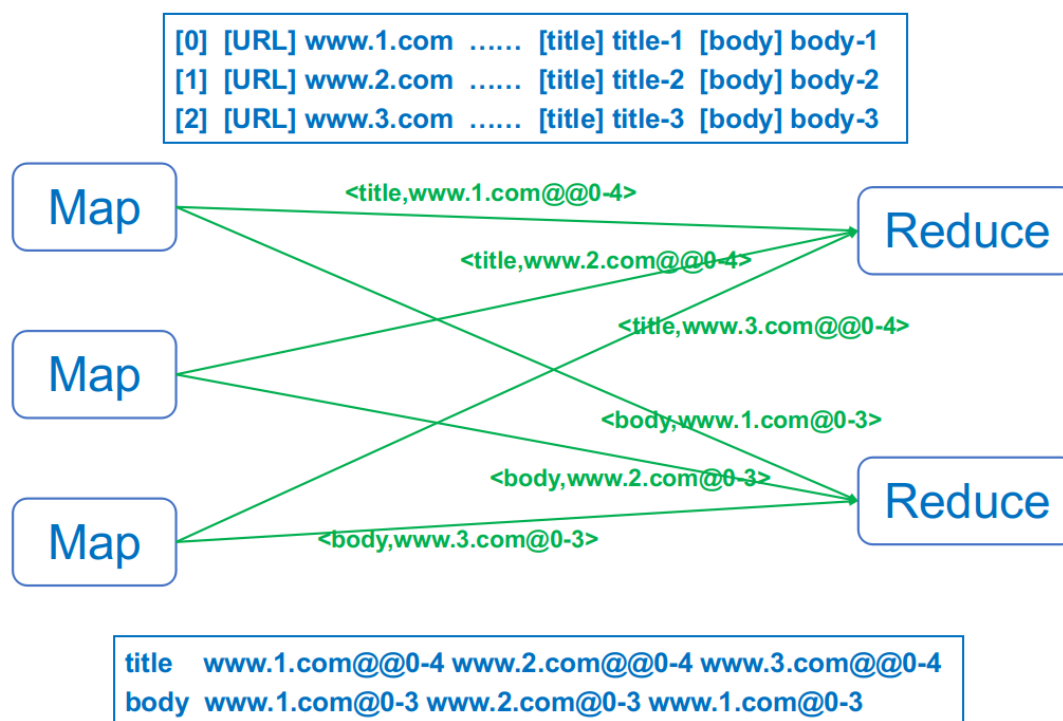


图 7: 中文分词 MapReduce 过程示意图

### 3.2.2 SEGMENT 数据

使用 Hadoop MapReduce 部署任务, 如图 8 所示。输出的倒排索引文件中, 共计有 537,406 个关键词, 总大小 4.2 GB。图 9 展示了倒排索引文件中的部分数据, 例如关键词“一两千”在 70030.htm 中 body 第 1846-1849 字符处和 6856.htm 中 body 第 1597-1600 字符处出现。


### 3.2.3 迁移到 HBase

最后, 我们将倒排索引文件中的数据迁移到 HBase 的 `NNU_WORDS` 表的 `segment` 族中, `NNU_WORDS` 表的结构如表 3 所示。文件中每行开头的关键词是每条记录的 `ID`, 每行后面所有的出现位置对应表中的 `segment:URLs` 字段。

## 3.3 PageRank 模块

PageRank 模块的目的是计算每个页面的 PageRank 值, 使用 Hadoop MapReduce 进行并行计算。开始时, 程序会执行一次 MapReduce 任务来构建网页间链接关系的有向图, 然后执行多次 MapReduce 任务迭代地更新计算每个页面的 PageRank 值。





### All Applications

- Cluster
- About Nodes
- Node Labels
- Applications
- NEW
- NEW\_SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler
- Tools

Cluster Metrics													
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources							
4	0	0	4	0	<memory:0, vCores:0>	<memory:8192, vCores:8>							

Cluster Nodes Metrics													
Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes									
1	0	0	0	0									

Scheduler Metrics													
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation										
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>										

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU vCores	Allocated Memory MB	Allocated GPUs
application_1623904995757_0004	root	Segment	MAPREDUCE	default	0	Thu Jun 17 17:53:09 +0800 2021	Thu Jun 17 17:53:10 +0800 2021	Thu Jun 17 18:09:20 +0800 2021	FINISHED	SUCCESS	N/A	N/A	N/A	N/A
application_1623904995757_0003	root	Segment	MAPREDUCE	default	0	Thu Jun 17 17:23:13 +0800 2021	Thu Jun 17 17:23:13 +0800 2021	Thu Jun 17 17:37:49 +0800 2021	FINISHED	SUCCESS	N/A	N/A	N/A	N/A
application_1623904995757_0002	root	Segment	MAPREDUCE	default	0	Thu Jun 17 17:04:23 +0800 2021	Thu Jun 17 17:05:10 +0800 2021	Thu Jun 17 17:22:45 +0800 2021	FINISHED	FAILED	N/A	N/A	N/A	N/A
application_1623904995757_0001	root	Segment	MAPREDUCE	default	0	Thu Jun 17 16:48:49 +0800 2021	Thu Jun 17 16:48:51 +0800 2021	Thu Jun 17 17:05:02 +0800 2021	FINISHED	FAILED	N/A	N/A	N/A	N/A

Showing 1 to 4 of 4 entries

图 8: Hadoop 任务管理页面

### 3.3.1 MapReduce 设计——构建链接图

图 10 展示了构建链接图的 MapReduce 过程。输入数据和中文分词 MapReduce 任务的一样，都是 RAW 数据。

在 Map 过程中，还是以第一行数据为例，输入的 key 是偏移量，我们同样不需要，输入的 value 是一行即一个网页的数据，从中解析出 URL 为 “www.1.com” 和 nextURLs 为 “www.2.com www.3.com”。遍历 nextURLs 中的每个 nextURL，每次输出一个形如 <url,nextURL> 的键值对，这里我们输出两个键值对 <www.1.com,www.2.com> 和 <www.1.com,www.3.com>。

在 Reduce 过程中，输入的 key 是 URL，原样输出，而输入的 value 是一个 nextURL 的迭代器，我们将迭代器中的所有字符串用空格拼接起来后，再前面加上每个页面的 PageRank 初始值 “1.00” 后输出。

### 3.3.2 MapReduce 设计——计算 PageRank 值

图 11 展示了计算 PageRank 值的 MapReduce 过程。该 MapReduce 过程会迭代执行多次，每次任务的输入数据是上次任务的输出数据，第一次任务的输入数据是构建链接图 MapReduce 任务的输出数据。

在 Map 过程中，以构建链接图 MapReduce 任务输出数据的第一行为例，输入的 key 是偏移量，同样不需要，输入的 value 是 www.1.com 的链路信息 “www.1.com 1.00 www.2.com www.3.com”，从中解析出当前 URL 为 “www.1.com”，PageRank 值为 “1.00”，nextURLs 为 “www.2.com www.3.com”。

我们既要更新 PageRank 值，又要保持链路结构。我们规定，当输出 value 的第一个字

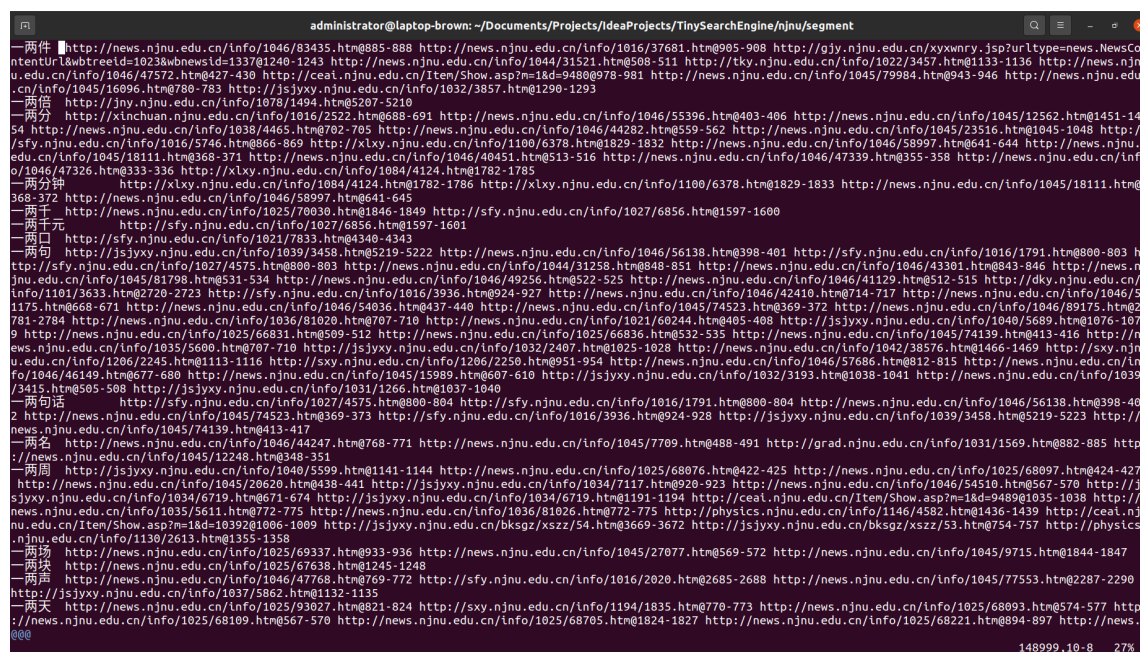


图 9: 倒排索引文件中的部分数据

符为“G”时，表示链路结构，第一个字符为“R”时，表示 PageRank 值。因此，该 Mapper 函数会输出一个 `<www.1.com,Gwww.2.com www.3.com>` 的键值对来保持链路结构，输出两个 `<www.2.com,R0.50>` 和 `<www.3.com,R0.50>` 的键值对，表示 `www.1.com` 将其 PageRank 值平分给它的两个链出页面 `www.2.com` 和 `www.3.com`，这样，`www.2.com` 和 `www.3.com` 各获得 0.50 的 PageRank 值。

在 Reduce 过程中，输入的 `key` 是 URL，输入的 `value` 中，既包含以“G”开头的链路结构，又包含以“R”开头的 PageRank 值。Mapper 函数将所有的 PageRank 值累加起来，再加上随机浏览的阻尼系数，得到最终的 PageRank 值 [8]。

形式化地，设当前页面为  $A$ ，阻尼系数为  $d$ ，函数  $PR(\cdot)$  返回页面的 PageRank 值。集合  $W$  是从页面  $A$  链出的所有页面集合，则页面  $A$  的 PageRank 值  $PR(A)$  的更新公式为

$$PR(A) = (1 - d) + d \cdot \sum_{w \in W} PR(w)$$

对于处理 `www.1.com` 的 Reducer 函数，所有键值对的 PageRank 值之和为 1.00，这是 `www.2.com` 将自己的 PageRank 值全部给了 `www.1.com`。设阻尼系数  $d = 0.85$ ，则页面  $A$  的 PageRank 值  $PR(A)$  更新为

$$PR(A) = (1 - 0.85) + 0.85 \times 1.00 = 1.00$$

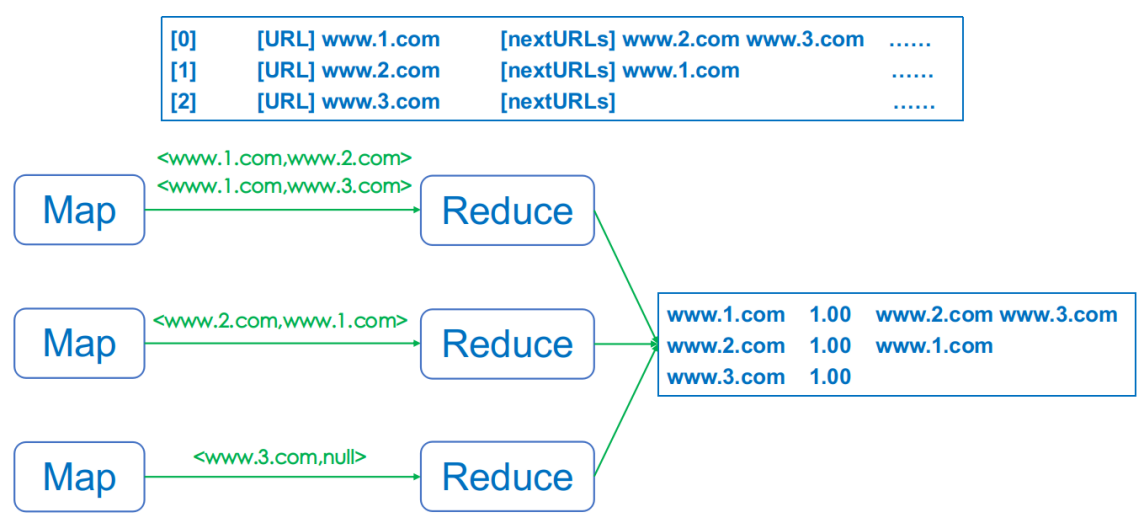


图 10: 构建链接图 MapReduce 过程示意图

3.3.3 PageRank 数据

真实的 PageRank 数据中，链出页面过于庞大，不好演示。因此，我们使用样例数据演示 PageRank 模块在各个阶段的输出数据。

对于样例数据：

[0]	[URL] www.1.com	[nextURLs] www.2.com www.3.com	[title] title-1	[body] body-1
[1]	[URL] www.2.com	[nextURLs] www.1.com	[title] title-2	[body] body-2
[2]	[URL] www.3.com	[nextURLs]	[title] title-3	[body] body-3

经过构建链接图的 MapReduce 任务后，输出数据：

```
www.1.com 1.00    www.2.com www.3.com
www.2.com 1.00    www.1.com
www.3.com 1.00
```

经过 1 次计算 PageRank 值的 MapReduce 任务后，输出数据：

```
www.1.com 1.000000    www.2.com www.3.com
www.2.com 0.575000    www.1.com
www.3.com 0.575000
```

经过 2 次计算 PageRank 值的 MapReduce 任务后，输出数据：

```
www.1.com 0.638750    www.2.com www.3.com
www.2.com 0.575000    www.1.com
www.3.com 0.575000
```

经过若干次计算 PageRank 值的 MapReduce 任务后，输出数据最终稳定在：

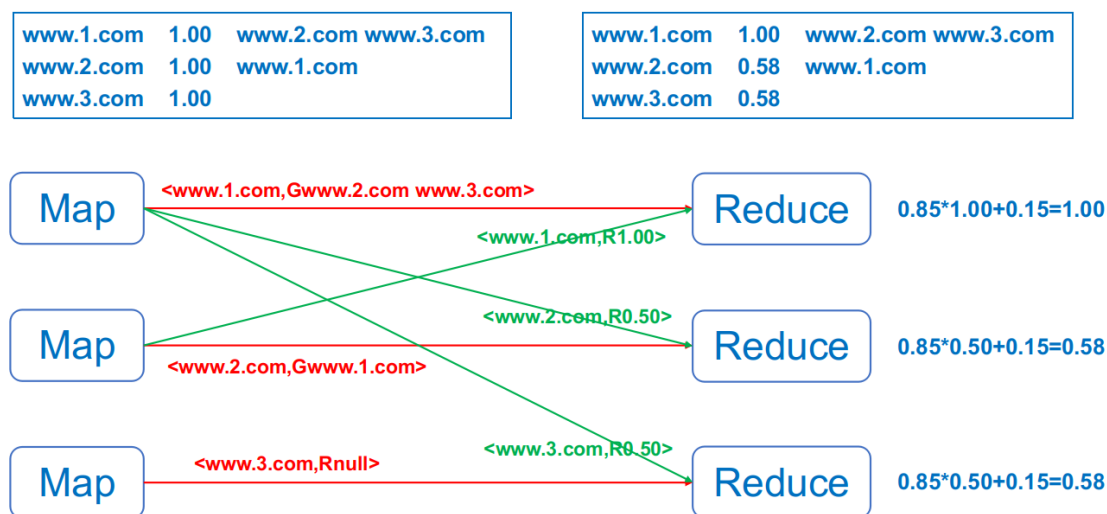


图 11: 计算 PageRank 值 MapReduce 过程示意图

```
www.1.com 0.437921 www.2.com www.3.com
www.2.com 0.338731 www.1.com
www.3.com 0.338731
```

### 3.3.4 迁移到 HBase

最后，我们将最后一次输出文件中的数据迁移到 HBase 的 `NNU_PAGES` 表的 `pagerank` 族中，`NNU_PAGES` 表的结构如表 2 所示。文件中每行开头的 URL 是每条记录的 ID，每行后面的 PageRank 值对应表中的 `pagerank:value` 字段。

## 3.4 检索模块

检索模块的目的是根据检索关键字返回搜索结果。由于服务器是使用 Python 语言搭建的，因此该模块的代码都是基于 Python 的。我们使用 HappyBase 库 [9] 与 HBase 进行数据通信，使用 Django 库 [10] 搭建 Web 服务器的框架。

### 3.4.1 连接 HBase 数据库

HappyBase 库是 Python 语言中和 HBase 建立数据通信的库之一，使用起来非常方便。使用表 2 作为样例数据，如下是一段实现查询 `www.1.com` 和 `www.2.com` 网页信息的示例代码：

```
import happybase
connection = happybase.Connection(host="192.168.0.20", port=9090)
table = connection.table('NNU_PAGES')
```

```
urls = ["www.1.com", "www.2.com"]
rows = table.rows(urls)
connection.close()
print(rows)

# {
#     b'www.1.com': {
#         b'crawl:title': b'title-1',
#         b'crawl:body': b'body-1',
#         b'crawl:nextURLs': b'www.2.com www.3.com',
#         b'pagerank:value': b'1.64',
#     },
#     b'www.2.com': {
#         b'crawl:title': b'title-2',
#         b'crawl:body': b'body-2',
#         b'crawl:nextURLs': b'www.1.com',
#         b'pagerank:value': b'0.88',
#     },
# }
```

返回值保存在变量 `rows` 中，这是一个字典对象，`key` 是每条记录的 ID，即每个网页的 URL，`value` 还是一个字典对象，其 `key` 是每个字段的名称，`value` 是记录在该字段的值。因为 HBase 是将文本内容序列化后存储的，因此所有的 `key` 和 `value` 都是 Python 字节序列。

### 3.4.2 检索过程

检索过程如图 12 所示。对于输入的查询关键词，需要先对其进行分词操作，因为查询关键词可能由多个基础词语复合而成。例如，对于查询关键词“南京师范大学计算机学院”，倒排索引表 `NNU_WORDS` 中肯定不存在，因此我们需要将其切分为“南京”、“师范”、“大学”、“计算机”、“学院”这样的关键词序列（这只是一个示例切法，当然还存在其它的切分方法），对每个关键词分别检索，最后再汇总搜索结果。

对于每个关键词，我们按如下步骤进行检索：

1. 在倒排索引表 `NNU_WORDS` 中搜索该关键词；
2. 若搜索结果为空，返回空集，程序结束；
3. 提取出所有含有该关键词的 URL 列表；
4. 在网页信息表 `NNU_PAGES` 中搜索 URL 列表，得到每个网页的信息；

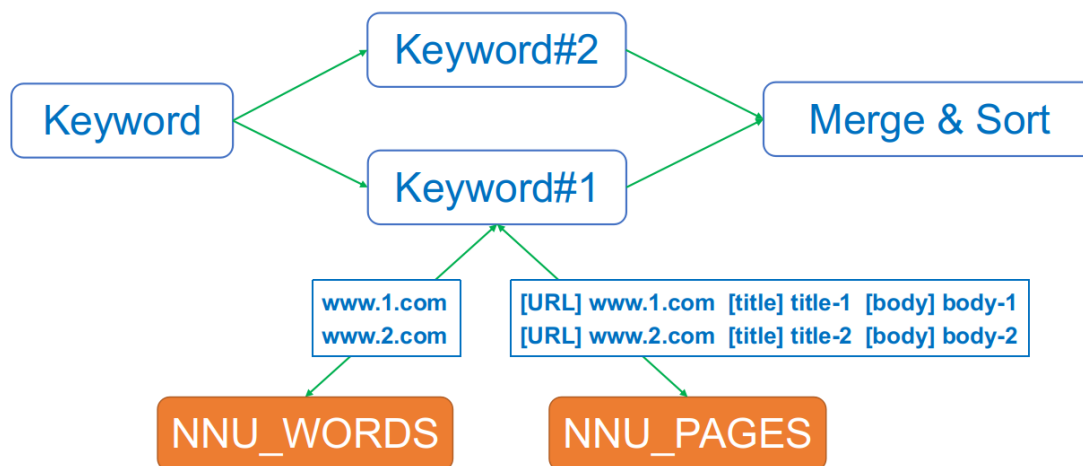


图 12: 检索过程示意图

## 5. 返回网页信息列表；

在获得了每个关键词对应的网页信息列表后，我们按如下步骤计算搜索结果：

1. 将每个关键词的网页信息列表汇总在一起；
2. 根据网页得分（综合考虑网页的 PageRank 值、关键词在标题中出现的次数、关键词在正文中出现的次数）对网页降序排序；
3. 返回前 100 个网页作为搜索结果；

注意，在第 2 步中，网页得分需要综合考虑网页的质量和网页的相关性，而 PageRank 值反映了网页的质量，关键词在标题和正文中出现的次数反映了网页的相关性。更进一步，我们认为关键词在标题中出现一次，其相关性大于在正文中出现一次，即标题的权重肯定要大于正文的权重，这也是我们在构建关键词出现位置时，使用“@@”和“@”区分标题和正文的原因所在。

形式化地，设一个网页  $A$  的 PageRank 值为  $PR(A)$ ，网页得分为  $SC(A)$ ，关键词在标题中出现的次数为  $N_T(A)$ ，出现一次得  $S_T$  分，关键词在正文中出现的次数为  $N_B(A)$ ，出现一次得  $S_B$  分。则网页得分为  $SC(A)$  的计算公式为

$$SC(A) = PR(A) + S_T \cdot N_T(A) + S_B \cdot N_B(A)$$

在计算过程中，我们取  $S_T = 0.5$ ， $S_B = 0.1$ 。





图 13: 查询关键词“计算机”返回的部分搜索结果

### 3.4.3 搭建服务器

我们使用 Python 的 Django 库搭建 Web 服务器的后端框架，使用 Vue 搭建前端框架。Web 守护进程运行在 1 号服务器上。

由于 1 号服务器是实验室中的服务器，因此没有公网 IP。接下来，我们将 Web 服务器挂载到阿里云服务器上，该阿里云服务器具有公网 IP 并绑定了域名。我们使用 frp 组件 [11] 开启 DDNS 功能，将 Web 服务器绑定到阿里云服务器的一个端口上。最后使用 nginx 配置反向代理，将用户在 80 端口上的访问重定向至指定端口。

由于搭建服务器不是本项目的重点，因此我们一笔带过，仅进行简要说明。

## 4 项目演示

本项目的开源地址是 <https://github.com/enderman19980125/TinySearchEngine/>。

本项目的演示地址是 <http://www.conybrown.cn/apps/retrieve-njnu/>。

图 13 展示了在搜索引擎中输入“计算机”作为查询关键词所返回的前三条搜索结果。我们可以看到，搜索结果的质量还是比较好的。

## 5 遇到的困难

本项目在进行过程中遇到了不少的困难，自己也经历了颇为曲折的研究过程。在我坚持不懈的钻研下，终于得以攻克这些难题。下面简要叙述自己在本项目中遇到最大的三个困难：

- **未完全过滤网页中的特殊字符，导致 MapReduce 出错**：在 html 文档中存在着大量各种各样的控制字符和不可见字符，这些字符对于后续的处理过程来说，简直是一颗巨大的隐藏炸弹。在保存网页的 title 和 body 信息中，自己并没有把不可见字符 `^M` 过滤掉。而 Hadoop 认为 `^M` 是一个换行符，因此 Hadoop 误将一行内容切分成了两行，导致程序在搜索定位点（`[URL]`、`[nextURLs]`、`[title]`、`[body]`）时出错；
- **中文分词 MapReduce 内存溢出**：在中文分词程序执行到 Reduce 过程 66% 时，发生错误 Error: Java heap space，即 Java 堆空间不足。解决方法：在 JVM 启动参数中添加 `-Xmx2048m` 参数，设置程序最大可用内存为 2 GB，并在 `mapred-site.xml` 配置文件中设置 `mapreduce.reduce.memory.mb` 为 2048；
- **HappyBase 无法连接到 HBase**：在 Python 中使用 HappyBase 库一直无法与 HBase 建立连接，后发现自己理解错了 HBase 的端口。HBase 使用 2181 端口进行集群间的数据通信，使用 16010 端口开放网页端可视化监控页面，使用 9090 端口启动 Thrift 服务器。而 API 需要与 HBase Thrift 服务器建立连接，而自己之前一直试图让 API 与 2181 端口建立连接；

## 6 未来的改进

本项目实现了一个简易粗糙的搜索引擎系统，未来还有非常大的改进空间。下面我们就该搜索引擎在未来的改进，提出三点有建设性的意见：

- **自动化流程**：目前该搜索引擎中各模块之间还需要手动迁移数据，而真正的搜索引擎应该实现自动化的数据流，并及时爬取更新旧的网页信息，添加新的网页信息，几乎不再需要人工的介入；
- **PageRank 算法的连贯性**：PageRank 算法在启动时，需要指定每个网页的初始 PageRank 值。而在搜索引擎运行了一段时间后，需要重新使用 PageRank 算法进行更新操作。如果此时对新添加的网页使用初始值，对已添加的网页使用当前值，执行 PageRank 算法。由于马尔可夫性质，算法肯定会收敛，但算法的正确性或精确度是否会受到影响？换句话说，PageRank 算法对于初始值是否敏感，自己在查阅了相关文献资料后，并没有发现回答；
- **研究更智能的页面得分度量算法**：目前搜索引擎中，综合考虑了网页的 PageRank 值、关键词在标题和正文中出现的次数，并使用了一个加权平均公式计算最终的页面得分。但这种做法不能保证搜索结果的精确度，远比不上成熟搜索引擎的搜索质量。后续过程中，试图考虑动态的弹性度量算法，并综合考虑多维度的评价指标，包括但不限于：关键词的 TF-IDF 值、关键词在文本中出现的位置、标题的长度、网页的发布时间等；



## 7 自己的收获

通过本项目的锻炼，自己收益良多，不仅仅是在并行程序设计方面，更体现在经验和精神层面。

自己对于并行程序设计方面的收获，总结如下：

- **HDFS 一地存储，随处访问：**RAW 数据的偶数分块文件存储在 1 号服务器上，奇数分块文件存储在 2 号服务器上，但文件对所有访问该 HDFS 的连接都是可见的。若在 HDFS 中访问一个不在当前节点中的文件，则 HDFS 会通过网络将该文件从存储节点上调度过来。特别说明，这里“一地存储”的说法是不严谨的，因为 HDFS 支持数据冗余备份存储；

Table Regions

Base Stats		Localities		Compactions						
Name(3)	Region Server	ReadRequests (6,670)	WriteRequests (0)	StorefileSize (1.22 GB)	Num.Storefiles (9)	MemSize (0 MB)	Start Key	End Key	Region State	
NNU_PAGES,.1623942271679.b3753ca6a79240e0738ce26a079c47b7.	desktop-brown:16030	3,753	0	562 MB	5	0 MB		http://news.njnu.edu.cn/info/1045/13656.htm	OPEN	
NNU_PAGES,http://news.njnu.edu.cn/info/1045/13656.htm,1623946055344.1c2ba3b928b13ab1a45665c4b6bd2eed.	slaker:16030	1,081	0	332 MB	2	0 MB	http://news.njnu.edu.cn/info/1045/13656.htm	http://news.njnu.edu.cn/sylm/tt/318.htm	OPEN	
NNU_PAGES,http://news.njnu.edu.cn/sylm/tt/318.htm,1623946055344.15b05099fa788b78f702b5a0da3d0ecf.	slaker:16030	1,836	0	356 MB	2	0 MB	http://news.njnu.edu.cn/sylm/tt/318.htm		OPEN	

图 14: NNU\_PAGES 表的数据存储状态

Table Regions

Base Stats	Localities	Compactions							
Name(4)	Region Server	ReadRequests (94)	WriteRequests (557,795)	StorefileSize (4.05 GB)	Num.Storefiles (10)	MemSize (0 MB)	Start Key	End Key	Region State
NNU_WORDS,.1623944740629.12bfa5e0dd16dfac3f7f69c2821c1af1c.	slaker:16030	19	240,673	1.41 GB	3	0 MB		\xE5\x90\x8D\xE6\xA0\xA2	OPEN
NNU_WORDS,\xE5\x90\x8D\xE6\xA0\xA2,1623944740629.3b473d6b5b60268cda2dc70811846596.	slaker:16030	1	31,963	309 MB	2	0 MB	\xE5\x90\x8D\xE6\xA0\xA2	\xE5\xA9\xB7\xE5	OPEN
NNU_WORDS,\xE5\xA9\xB7\xE5,1623944726940.11d3039bd65eff6d44c558912be49b6f.	desktop-brown:16030	15	71,225	762 MB	3	0 MB	\xE5\xA9\xB7\xE5	\xE6\x97\xB6\xE5\x80\x9A	OPEN
NNU_WORDS,\xE6\x97\xB6\xE5\x80\x9A,1623944726940.76135c0165f5d4f68a899ca8adf1e1d1.	desktop-brown:16030	59	213,934	1.60 GB	2	0 MB	\xE6\x97\xB6\xE5\x80\x9A		OPEN

图 15: NNU\_WORDS 表的数据存储状态

- **HBase 自动数据平衡**: 在数据迁移过程中, 所有的数据都是从 1 号服务器上传到 HBase 中的, 但我惊奇地发现, HBase 自动做了数据平衡。如图 14 和 15 所示, `NNU_PAGES` 表的 1.22 GB 有 562 MB 存储在 1 号服务器上, 有 688 MB 存储在 2 号服务器上, `NNU_WORDS` 表的 4.05 GB 有 2.35 GB 存储在 1 号服务器上, 有 1.70 GB 存储在 2 号服务器上;

自己对于经验和精神上的收获, 总结如下:

- **系统架构能力的提升**: 该系统包含四个功能模块和五部分数据, 自己脑海中必须非常清楚每个功能模块的作用, 熟悉每部分数据的结构。在设计系统架构时, 必须要考虑到系统的完整性和流畅性, 同时兼顾系统的鲁棒性和健壮性。自己也体会到了架构师对于能力和经验的要求非常高;
- **调试查错能力的提升**: 自己在研发系统的过程中, 遇到了许多大大小小的 bug, 如何读懂报错信息、如何定位错误位置、如何分析错误原因, 是每个研发人员都躲不开的夺命三连问。通过这次系统研发的经历, 自己对于 bug 的解决处理能力又上了一个台阶;
- **StackOverflow 等问答网站的使用**: 我们遇到的绝大多数 bug, 前人都已经踩过坑了, 并且告诉后人应该如何从坑里爬出来。StackOverflow 这类问答网站上涵盖了各种环境各种语言的 bug, 并且有许多大牛提供了非常优质的解决方案。合理利用报错信息和问答网站, 往往能起到事半功倍的效果;
- **坚持不懈的毅力**: 自己在搭建环境和编写代码的过程中, 遭遇了好几次一筹莫展无法继续下去的情况, 几度想要放弃, 但最终都慢慢熬下来了。身处绝境时, 不妨出去散散心, 换个环境放松心态, 缓解一下焦虑的情绪, 可能就会发现“山重水复疑无路, 柳暗花明又一村”。往往打败自己的不是困难本身, 而是自己气急败坏的心态;

## 参考文献

- [1] Search engine - wikipedia. [https://en.wikipedia.org/wiki/Search\\_engine](https://en.wikipedia.org/wiki/Search_engine). Accessed: 2021-06-18.
- [2] 搜索引擎 \_ 百度百科. <https://baike.baidu.com/item/> Accessed: 2021-06-18.
- [3] The jps command. <https://docs.oracle.com/en/java/javase/13/docs/specs/man/jps.html>. Accessed: 2021-06-18.
- [4] Hdfs architecture guide. [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html). Accessed: 2021-06-18.
- [5] Apache hadoop 3.2.2 -hadoop: Setting up a single node cluster. <https://hadoop.apache.org/docs/r3.2.2/hadoop-project-dist/hadoop-common/SingleCluster.html>. Accessed: 2021-06-18.

- 
- [6] Apache hbase –apache hbase™ home. <https://hbase.apache.org/>. Accessed: 2021-06-18.
  - [7] huaban/jieba-analysis: 结巴分词 (java 版). <https://github.com/huaban/jieba-analysis>. Accessed: 2021-06-18.
  - [8] Pagerank - wikipedia. <https://en.wikipedia.org/wiki/PageRank>. Accessed: 2021-06-18.
  - [9] Happybase –; happybase 1.2.0 documentation. <https://happybase.readthedocs.io/en/latest/>. Accessed: 2021-06-18.
  - [10] The web framework for perfectionists with deadlines | django. <https://www.djangoproject.com/>. Accessed: 2021-06-18.
  - [11] fatedier/frp: A fast reverse proxy to help you expose a local server behind a nat or firewall to the internet. <https://github.com/fatedier/frp>. Accessed: 2021-06-18.