海量网络文本去重系统的设计与实现

汤建明 寇小强
(华北计算机系统工程研究所 北京 100083)

摘 要 如今网络和信息技术飞速发展,每天都有数以亿万计的文本数据产生。然而,不可避免地有很多文本内容是重复的。这样导致用户在利用搜索引擎搜索或者在网站上浏览内容时会看到很多相似的东西。这不仅给用户带来了不好的体验,对内容提供商来说,也需要花费更多的资源对重复冗余的内容进行存储。因此,对文本做一些相似度判断的基本处理,去除重复的文本有很重要的意义和价值。提出设计和实现一种基于 simhash 的文本去重系统。该系统可以对每天新产生的文本内容进行相似度计算,对于相似的内容只生成一份唯一标识并进行入库处理,有效排除了相似度太高的重复文本。

关键词 文本去重 Simhash 相似度

中图分类号 TP391 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2018.12.007

DESIGN AND IMPLEMENTATION OF MASSIVE NETWORK TEXT DEDUPLICATION SYSTEM

Tang Jianming Kou Xiaoqiang
(*National Computer System Engineering Research Institute of China*, *Beijing 100083*, *China*)

Abstract With the rapid development of the Internet and information technology in the present world, there are a large number of texts generated every day. However, it is unavoidable that many textual content is duplicated, which may lead users to see a lot of similar things when they search through search engines or browse content on websites. It not only brings a bad experience to users, but also requires more resources for content providers to store these repetitive and redundant contents. Therefore, it is of great significance and value to do some basic processing of text similarity judgment and remove duplicate text. A text deduplication system was designed and implemented based on simhash. The system can perform similarity calculation on the newly generated text content every day. For the similar content, only a unique identifier is generated and stored into the database, which effectively excludes duplicate texts with a high degree of similarity.

Keywords Text deduplication Simhash Similarity

0 引 言

随着互联网的普及,每天有数以亿计的新内容在网络上产生。这些内容或者以新闻报道的形式,或者以公众号和专栏文章的形式呈现在人们眼中。基于如此海量的文本数据,其中有着大量的冗余相似内容。有研究表明,在应用系统所保存的数据中,冗余数据约占 60% 左右[1]。因此,如何有效地来识别互联网上的重复信息,对学术界和工业界来说都是一个很有意义的课题。

举一个例子:对于基本所有的搜索引擎系统来说,它们底层的爬虫系统会去大量抓取网络上的文本等数据资源,显然,抓取到重复和高度相似的网页和文本进行存储和收录是没有意义的,这样不仅浪费存储资源,也会对后续的计算处理带来不好的影响;同时,展示重复和高度相似的检索结果对于用户来说也是不好的使用体验。对于像传统的新闻门户网站和推荐系统应用来说,保证文章质量是非常重要的。所以,有必要保证对于每天新产生的文本内容进行相似度判断和去重处理。对于相似度过高的文章只保证唯一一份内容入库,这样不仅节约了数据存储的成本,也会提高用户的

收稿日期:2018-04-30。**汤建明**,硕士生,主研领域:数据处理,图像识别,网络爬虫。**寇小强**,研究员。

检索体验。

本文基于 simhash 算法[2-3]设计和实现一个文本去重系统,采用 Ansj 开源分词工具对原始文本内容进行分词处理,数据存储采用 Rdis + Mysql + Hbase。

1 海量网络文本去重系统设计

1.1 整体架构

图 1 是本文海量网络文本去重系统的整体功能模块设计图。整个系统可以主要分成三个部分:一是分词模块对原始文本内容进行分词处理;二是判重模块对处理后的文章进行相似度判断;三是入库模块对经过去重的文本数据进行入库处理。

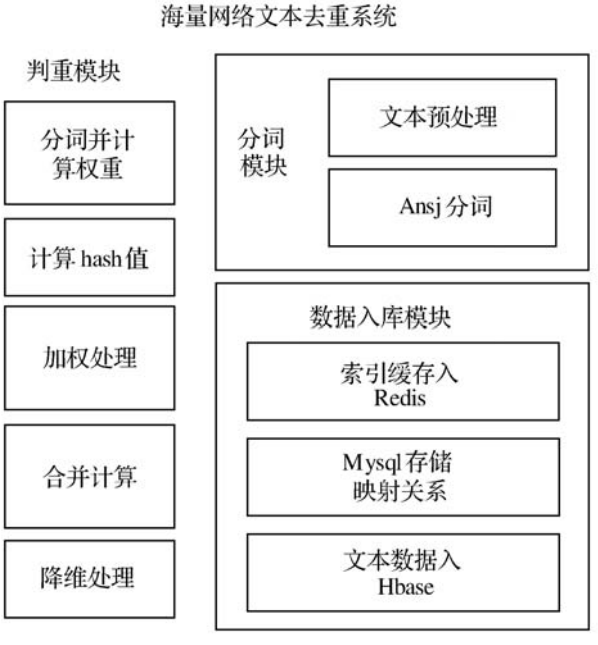

图 1 系统整体架构

1.2 工作流程

图 2 是本文海量网络文本去重系统的整体调用处理流程。整个系统会用 Maven 打成 war 包,最后部署在 Tomcat 服务器中。系统以 HttpServlet 接口的方式提供调用入口。每篇文章有一个唯一标识,记为 nid。经过预处理的文本数据以 JSON 字符串的形式进入系统,主要包含有文章来源 url、文章标题 title 和文章内容 content 等属性。最后会返回一个唯一标识,以 docId 来表示,内容相似的文章对应同一个 docId 值。系统会依次根据文章的 url、title 和 content 去 Redis 中查找是否有相同的缓存值,有则直接返回对应的 docId 值,没有则会根据文章的 url、title 和 content 计算出一个新的 docId 值,并缓存在 Redis 中。数据库 Mysql 中会保存所有 nid 和 docId 的对应关系。Hbase 中会保存文章的所有内容信息,用 docId 来唯一标识。

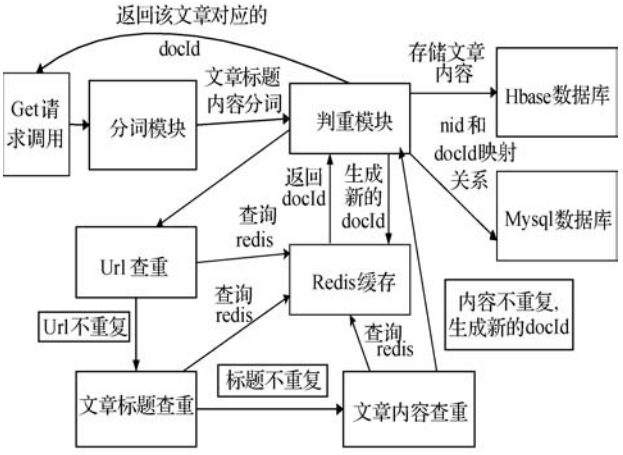

图 2 工作流程

2 关键实现技术

2.1 Ansj 分词

Ansj 是一个分词效果非常精准的中文分词器[4],它完全开源,是 ICTLAS 的 Java 实现版本,同样是构建了 Bigram[5] + HMM 的模型来进行分词。虽然它的基本原理和 ICTLAS 一样,但是 Ansj 在工程上做了一些优化,比如:用 DAT 高效实现的检索词典、用邻接表来实现分词 DAG、同时支持基于个人需求的自定义分词词典,以及支持消歧义规则的自定义等[8]特点。

2.2 simhash 文本去重算法

Simhash 是 locality sensitive hash(局部敏感哈希)的一种,最早是由 Moses Charikar 在《similarity estimation techniques from rounding algorithms》一文中提出来的。

该算法的主要思想就是降维,将高维的特征向量经过一系列的处理后映射成低维的特征向量,然后通过比较两个向量的 Hamming Distance 来判断文本是否高度相似甚至完全重复。其中,Hamming Distance 通常又称为汉明距离[7],这是信息论中的一个概念,指的是两个长度相等的字符串之间对应的各个位置上不同的字符的总个数。简单来说就是,将其中一个字符串变为另一个字符串总共需要改变的字符总个数。这样一来,通过对比文档内容的 simhash 值的汉明距离,就能够得到它们之间的相似度是多少。

2.3 Redis + Mysql + HBase 三级存储系统

Redis 是一个支持网络的、基于内存存储、数据可持久化的开源 Key-Value 数据库[9]。和传统的关系型数据库 Mysql 相比,Redis 的查询和存储性能都要高得多。由于 Redis 可基于内存存储,所以非常适合用作缓存处理,这样可以加快存储和查询速度,也能够减轻数据库的负载压力。

在本文的海量网络文本去重系统中,Redis 用来存储文章 url、title 和 content 经过处理后的索引,这样,即使是在高并发的网络场景下,每次有新的请求进入系统,也不会存在查询瓶颈。这样的缓存设计也将真正的文章内容存储解耦,使得判重模块只关注于文章内容去重,数据的存储异步地载入库模块进行,可以在很大程度上提高整个系统的处理性能。

在本系统的三级存储系统设计中,Mysql 只用来存储每篇文章的唯一标识 nid 和 docId 的映射关系,并不真正存储文章的数据内容。互联网上的海量文本数据包含的属性大不相同,Mysql 不适合用来存储这种数据,当数据量大到一定级别后,Mysql 的性能也会大打折扣。因此,文章的数据最后存储在 HBase 中。

HBase 是一种分布式的存储系统,它构建在 HDFS 之上、支持面向列的存储[6]。HBase 支持数据实时地读写,同时,它对超大规模的数据集的随机访问也支持的非常好。基于 HBase 的这些特性,非常适合用来存储海量的文本数据,其列存储属性极易扩展,不会因为文章内容各不相同的属性而带来存储困难。

3　海量网络文本去重系统的实现

3.1　分词模块

对于请求的 Json 数据,其格式大致如下:

```
json =
{
    "url": "http://www.help.com",
    "title": "测试",
    "content": "这是一个测试",
    "category": "0",
    "media": "sina",
     "imageCount": "4"
}
```

url 表示文章的网络地址,title 表示文章的标题,content 表示文章的内容,category 表示文章的属性,media 表示文章所属的媒体,imageCount 表示文章中的插图数量。在该系统中,我们只关注 title 和 content 两个字段的内容分词。

分词模块将 title 和 content 的内容切分成一个一个单独的词,采用 Ansj 提供的 ToAnalysis 分词实现,对于分词后的单词,保存在 Redis 缓存中,过期时间设置为 7 天。

3.2　内容判重模块

该模块是系统的核心模块。主要依据文章的 url、title 和 content 三个属性值来进行判重处理。

经过分词处理的文章 title 和 content,会首先经过判重模块提供的签名方法生成一个唯一的 64 位签名,该方法则基于 simhash 算法实现。simhash 算法主要分为 5 个步骤来进行:

(1) 分词　该步骤首先调用分词模块提供的方法,将待处理的语句先进行初步的分词,这样就得到了语句有效的特征向量,接着为特征向量来设置权重,这里可以根据需求自定义 n 个级别的权重。比如将 n 设为 5,则可以用 1 ~ 5 来表示每个向量的权重。在具体实现中,可以根据每个单词在整段文本数据中出现的次数来定义它对应的权重值大小。

(2) 计算 hash 值　通过 hash 函数来算出上一步中每个特征向量各自的 hash 值,hash 值可以看成用二进制数 0 和 1 所组成的 n-bit 位签名。比如"去重"的 hash 值计算出来为 110010,"系统"的 hash 值计算出来为 101001。经过这样处理之后,字符串值就变成了一系列的二进制数字组合。

(3) 加权处理　在算出了单词 hash 值的基础之上,对这些特征向量来进行加权处理,具体实现方式即:W = hash × weight,weight 代表该单词在文本中的权重。对上一步算出来的每一个 hash 值,计算它二进制数字每一位的加权结果。如果该位的值为 1,那么就和权值进行正相乘;如果该位的值为 0,就和权值进行负相乘。例如:假如上一步"去重"的权值为 3,则加权后的值计算为 W(去重) =3 3 -3 -3 3 -3;同理,假设"系统"的权值为 5,则加权后的值为 W(系统) =5 -5 5 -5 -5 5。其余所有特征向量依此类推。

(4) 合并计算　将所有经过加权计算的特征向量的加权结果累加,得到这段文本的总的加权值。例如:拿前面的"去重"和"系统"举例,累加得到"3 +5 3 -5 -3 +5 -3 -5 3 -5 -3 +5",最后结果为"8 -2 2 -8 -2 2"。

(5) 降维处理　对于以上处理后的 n-bit 的 hash 加权值,对每一位,如果该位的值大于 0 则置为 1,否则置为 0,从而得到该文本语句的 simhash 值。同样以上面的例子来看,则降维处理后的值为 101001,这就是最后这段文本的 simhash 签名。

图 3 体现了计算 simhash 签名的过程。

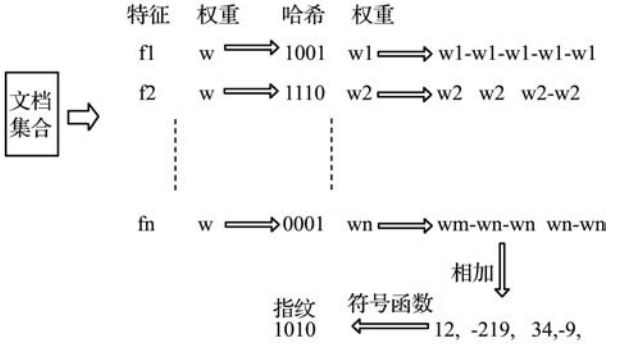

图 3　计算 simhash 签名

在得到每篇文档对应的 simhash 签名值之后,通过计算两个签名值的汉明距离来判断它们是否相似。判断标准可以根据具体业务场景来定,本系统采用业界工程实践较好的 3 作为判断标准,即对于 64 位的 simhash 值,如果汉明距离相差在 3 以内的话,就可以认为两篇文档是高度相似的。

以上通过计算 simhash 值来判断相似的过程是判重模块所提供的一个方法,具体到本系统的业务逻辑处理中,判重流程如下:

(a) 根据 url 和固定前缀值 URL_INDEX 形成的字符串去 Redis 缓存中查找是否有相同的值,如果存在,则直接返回对应的 docId 值;如果不存在,则进行步骤(b)。

(b) 根据 title 生成的 simhash 签名以及固定前缀值 TITLE_INDEX 形成的字符串去 Redis 中查找是否有相同的值,如果存在,则直接返回对应的 docId 值;如果不存在,则进行步骤(c)。

(c) 根据 content 生成的 simhash 签名以及固定前缀值 CONTENT_INDEX 形成的字符串去 Redis 中查找是否有相同的值,如果存在,则直接返回对应的 docId 值;如果不存在,则进行步骤(d)。

(d) 经过前面三步流程查找都没找到,说明这一次请求的文章是一篇新的文章,则根据该文章的 url、title 和 content 值生成一个唯一的 64 位 docId 值,同时对 title 和 content 的 simhash 值建立索引并进行缓存处理,以供下一次新的请求做查询比对处理。

以上即为整个判重模块的设计实现。

3.3 数据入库模块

数据入库模块异步地进行数据的入库存储处理。

对每一次请求,如果进过判重模块处理后为相似的文章,则在 Mysql 中添加一条记录,保存该文章 nid 和已经存在的相似文章的 docId 的映射关系;如果判重处理后为新的文章,则在 Mysql 中添加一条记录,保存该文章 nid 和新生成的 docId 值的映射关系。对每一个 docId 值,只在 HBase 中保存一份热度最高的文章内容。

我们在其他的系统中需要检索或者推荐文章时,就不会出现重复相似的文章,因为入库模块最后数据入 HBase 的过程中保证了一个 docId 值在 HBase 中只保存有唯一一篇文章。

3.4 实验结果展示

本系统采用 Java 开发实现,采用 Maven 作为项目构建工具,最后部署在 Linux 服务器上,以 war 包形式部署在 tomcat 容器中,提供一个访问接口供外部调用,具体调用形式如下:

127.0.0.1:8080/docId/getDocId? json = { }

其中,127.0.0.1 为系统部署的服务器网络地址,8080 为访问的端口号,docId 为服务名称,getDocId 为访问接口,json = { } 为请求数据,以 Json 形式传输,具体字段可见 3.1 中所描述。利用 Postman 来模拟请求,先自定义一个测试数据如下:

```
json =
{
    "url":"http://www.simhash.com",
    "title": "计算机应用与软件",
    "content": "海量网络文本去重系统实验测试,这是一段测试文本的内容。",
    "category": "0",
    "media": "test1",
    "imageCount": "4"
}
```

系统运行过程如图 4 所示。

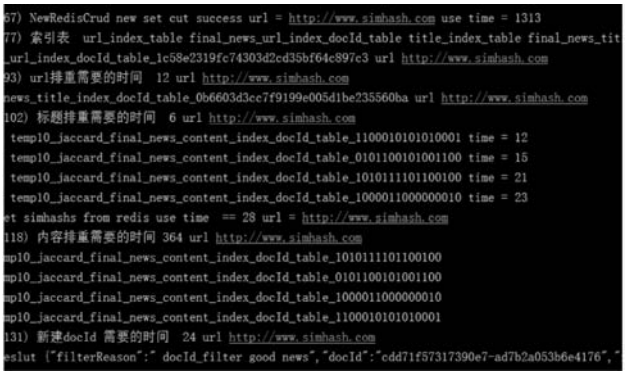

图 4 新文章请求系统运行图

可以看到,如果是一篇新的文章请求系统,则系统内部会经过 url、title 和 content 三次查找判重操作,最后会在 Redis 中缓存新的索引,并返回计算生成的新的 docId 值。最后结果以 Json 字符串形式返回,如下所示:

```
{
    "filterReason": "docId_filter good news",
    "docId": "cdd71f57317390e7-ad7b2a053b6e4176",
    "filterStatus": "good",
    "status": "success"
}
```

接下来我们再定义一个和上面相似的文本内容作为请求数据去请求该去重系统,自定义数据如下:

```
json =
{
    "url": "http://www.simhash1.com",
    "title": "计算机应用和软件 2",
    "content": "海量网络文本去重系统实验检测,这是一段相似的测试文本的内容。",
```

```
    "category": "0",
    "media": "test2",
    "imageCount": "3"
}
```


可以和上面的请求数据对比,两者高度相似,这次系统运行过程如图 5。

图 5　相似的文章请求系统运行图

从图中红线圈可以看到,系统最后判断 content 排重成功,也就是成功判定出这篇请求的文章和上一篇文章相似。在查询 Redis 判重的过程中,由于标题和内容都相似,系统也没有重新计算新的索引去缓存。最后返回的 docId 值也和上一篇相同,也就是没有重新计算生成新的 docId。

4　结　语

海量网络文本去重系统用于对互联网上充斥着的大量重复网页内容和文章进行去重处理。由于采用多模块的解耦设计,可以保证系统在高并发的访问情况下做到快速响应。整个系统采用 Maven 作为项目构建工具,易于移植和扩展。同时,本系统提供标准的 HttpServlet 接口,采用 Json 作为通用的网络数据传输格式,可以很方便地接入到其他需要进行文本去重处理的工程项目中。未来,该系统将添加图片判重处理模块,这样使得系统的适用范围更广,可以识别出相似的图片,这样对包含有插图的文本内容可以进行更为精确地去重判断和处理。在系统的并发处理和存储性能方面也将展开进一步的研究工作,从而提高系统的性能水平。

参　考　文　献

[1] Clements A T, Ahmad I, Vilayannur M, et al. Decentralized deduplication in SAN cluster file systems[C]//USENIX annual technical conference. USENIX, 2009: 101 - 114.

[2] Charikar M S. Similarity estimation techniques from rounding algorithms[C]//Proceedings of the thirty-fourthannual ACM symposium on theory of computing. ACM, 2002: 380 - 388.

[3] Manku G S, Jain A, Sarma A D. Detecting near-duplicates for web crawling[C]//Proceedings of the 16th international conference on world wide web. ACM, 2007: 141 - 150.

[4] 张海军, 史树敏, 朱朝勇, 等. 中文新词识别技术综述[J]. 计算机科学, 2010, 37(3): 6 - 10, 16.

[5] 王笑旻. 基于 Bigram 的特征词抽取及自动分类方法研究[J]. 计算机工程与应用, 2005, 41(22): 177 - 179.

[6] 葛微, 罗圣美, 周文辉, 等. HiBase: 一种基于分层式索引的高效 HBase 查询技术与系统[J]. 计算机学报, 2016(1): 140 - 153.

[7] 张焕炯, 王国胜, 钟义信. 基于汉明距离的文本相似度计算[J]. 计算机工程与应用, 2001, 37(19): 21 - 22.

[8] 奉国和, 郑伟. 国内中文自动分词技术研究综述[J]. 图书情报工作, 2011, 55(2): 41 - 45.

[9] 崔丹, 史金鑫. 基于 Redis 实现 HBase 二级索引的方法[J]. 软件, 2016, 37(11): 64 - 67.

(上接第 26 页)

[2] Ferraiolo D, Kuhn R. Role based access control[C]//Proceedings of 15th NIST-NCSC National Computer Security Conference, 1992: 554 - 563.

[3] Ferraiolo D F, Sandhu R, GavriIa S, et al. Proposed NIST standard for roIe-based access control[J]. ACM Transactions on Information and System Security, 2001, 4(3): 224 - 274.

[4] 杨柳, 危韧勇, 陈传波. 一种扩展型基于角色权限管理模型(E-RBAC)的研究[J]. 计算机工程与科学, 2006, 28(9): 126 - 128.

[5] 林伟炬, 刘列根, 张宇. 一个通用的权限管理模型的设计方案[J]. 微计算机信息, 2009, 25(15): 1 - 3.

[6] 徐南方, 苏星晔. 单点登录技术专利分析[J]. 中国新通信, 2018(4): 100.

[7] 钱超. 分布式 session 服务系统及方法: CN, CN 102752323 A[P]. 2012.

[8] Siriwardena P. OAuth 2.0[J]. Advanced Api Security, 2014: 91 - 132.

[9] Jones M, Bradley J, Sakimura N. JSON Web Token (JWT)[EB]. Internet Engineering Task Force (IETF), 2015.

[10] PivotalSoftware. Spring-cloud-security[EB/OL]. (2016 - 05 - 25)[2017 - 10 - 23] http://cloud. spring. io/spring-cloud-security/spring-cloud-security. html/.

[11] Cloudfoundry. User Account and Authentication[EB/OL]. (2015 - 01 - 17)[2016 - 07 - 17] https://docs. cloudfoundry. org/uaa/.

[12] Cloudfoundry. OAuth2.0[EB/OL]. (2006 - 05 - 26)[2008 - 03 - 08] https://oauth. net/2/.