基于Storm的地理编码引擎

余靖毅,邬 伦,高 勇*
(北京大学遥感与地理信息系统研究所,北京 100871)

摘要:近年来,随着Web 2.0和具有位置感知能力的移动计算设备的普及应用,带来了大量含有时空语义的地理大数据。在这个背景下,以地图厂商人工方式和半自动方式更新地名地址库为基础的传统地理编码服务,已难以满足新的应用需求。本文提出一种地理大数据驱动的自适应地理编码引擎的构建思路和方法,通过引入实时计算和流式计算平台Storm,实现对网络中的多源地理大数据的爬取与实时处理,加速地名地址库及相关资源的生成与更新过程,并给出了相适应的地理编码匹配方法。在实时流式计算框架基础上,通过JTS Topology Suite实现流式并行的空间操作,设计并实现了基于Storm的地理编码引擎原型系统,满足多源地理大数据的高效处理和地理编码要求。实验结果表明,该引擎通过实时流式处理可加速地址库的扩充与更新过程,并且利用地址库持续更新的方法,提升了地理编码的匹配率和定位准确度。

关键词:地理编码;实时流处理;地理大数据;Storm
DOI:10.3724/SP.J.1047.2015.01431

1 引言

地理编码(Geocoding)是指将文本的通讯地址数据转化为数字地理坐标(如成对的经度和纬度)[1]。它提供了一种把描述成地址的地理位置信息,转换成可被用于地理信息系统的地理坐标方式[2]。目前,地理编码已被广泛运用于各类学术研究和商业应用中,如商业选址、电子商务的快递物流、犯罪制图与分析[3]、公共健康和流行病研究[4]等。

目前,常见的地址编码方法主要有3种:点模型(Point Geocoding)、街道模型(Street Geocoding)和面模型(Parcel Geocoding)[5]。街道模型利用每段街道中存储的门牌开始编号和终止编号,通过插值算法估算出某个门牌号的地理坐标[6]。街道模型的地址匹配精度受街道门牌号排列规则性影响较大,当城市门牌编号混乱时,该模型的插值匹配效率较差。但由于街道地址编码获取成本较低,数据质量较高,因此,在欧美等国家至今仍然有广泛的应用。

随着谷歌地图、高德地图、百度地图等网络地理服务模式的兴起,地址匹配作为GIS的基础服务具有重要的应用价值。大量商业组织通过地址普查和具有测绘资质的地图厂商获取了丰富的兴趣点信息,利用这些地址名称字符串的文本语义匹配,为解决中文地址匹配带来了新的思路,从而推动了点模型地理编码技术的发展[7]。其中,孙亚夫等[8]提出利用中文分词词典查找地址要素,利用最大正向匹配算法进行分词的同时,查找判断地址要素的父地址。Hutchinson[9]将人工智能技术引入到地址匹配中,提出了一种基于智能体的地理编码方法。程昌秀[10]通过自定义的规则树进行推理,并借助构建规则树和歧义栈提高地址匹配的成功率。

不论是街道模型还是点模型,地名地址库都是核心的基础组件。地名地址库的质量直接决定了地理编码的效果。社会和城市建设的快速发展,导致地址系统稳健性差,尤其是中国由于历史原因,导致门牌的缺号、跳号、重号现象比较严重。同时城市道路名、小区名、建筑物名称等更新落后,使得城市的地址和地名体系变得十分复杂,对地址数据的采集及地址数据库的建设造成了不利的影响[11]。在建立标准地址数据库的基础上,通过地址数据的预处理、分解和归类,采用分词技术可解决大多数非空间坐标地址的匹配问题。但是,对于非标准地

收稿日期:2015-04-15;**修回日期**:2015-05-29.
基金项目:国家自然科学基金项目(41271385)。
作者简介:余靖毅(1990-),男,湖北武汉人,硕士生,研究方向为地理大数据计算与挖掘。E-mail: harryyu1018@pku.edu.cn
***通讯作者**:高 勇(1974-),男,辽宁抚顺人,副教授,研究方向为地理信息检索、空间数据挖掘。E-mail: gaoyong@pku.edu.cn

址、方位词、未登录地址要素或其他词的识别率很低[12]。

近年来，Web 2.0和具有位置感知能力的移动计算设备的普及应用，带来大量具有个体标记和时空语义信息的地理大数据[13]。Facebook、Twitter、Foursquare、新浪微博、大众点评等主流应用也产生了大量地名、地点描述，以及根据发布者经历的带有空间语义的评论，为地名地址的采集提供了丰富的素材和背景知识[14-15]。在地理大数据背景下，以地图厂商人工方式更新地名库为基础的传统地理编码服务，已很难满足新的需求，迫切需要一种更加实时的、自适应的新型地理编码服务。然而，利用大数据驱动的方式获取来自互联网中的自发性地理信息（Volunteered Geographic Information，VGI），并用于构建地名地址库已成为一种新的方向和选择[16]。

针对上述问题，本文主要研究了一种地理大数据驱动的地理编码引擎。通过引入实时流处理计算平台Storm，实现多源地理大数据的实时处理，加速地名库及特征词库自动生成和更新；并针对地名库中地址的进一步分析挖掘，实现地理编码引擎的实时和自适应能力，提高地理编码的效率和准确度。

2 Storm计算平台简介

Storm（http://storm.apache.org）是一个开源的分布式实时计算系统，可简单高效可靠地处理大量数据流。其主要应用场景为实时分析、在线机器学习、持续计算、ETL、分布式RPC等[17]。此外，Storm支持水平扩展，具有高容错性，保证每个消息都会得到可靠高效的处理。在一个小集群中，每个节点每秒可处理数以百万计的消息。更重要的是，Storm的部署和运维都很便捷，可方便使用任意编程语言来开发应用。

2.1 Storm的工作机制

Storm集群中有2种节点：主控节点（Master Node）和工作节点（Worker Node）。主控节点通过Nimbus后台程序，负责在集群里面分配计算任务和监控集群状态。工作节点通过Supervisor的后台程序，负责监听分配给它所属机器的执行任务，根据需要启动/关闭工作进程[18]。Nimbus和Supervisor之间所有的协调工作和状态管理都通过Zookeeper（http://zookeeper.apache.org）集群完成，保证了Storm极高的可靠性和稳定性。

2.2 Storm的关键概念

Storm中的关键组件及其之间的关系如图1所示，具体包括：

(1) 计算拓扑（Topologies）：在Storm中运行的一个实时应用程序，由各个组件间的消息流动所形成逻辑上的运算拓扑。

(2) 消息元组（Tuple）：一个消息传递的基本单元。

(3) 消息流（Streams）：是Storm中最关键的抽象。一个消息流是一个没有边界的Tuple（元组）序列，而这些Tuples会被以一种分布式的方式并行地创建和处理。

(4) 消息源（Spouts）：是Topology中的消息和流数据的生产者。

(5) 消息处理者（Bolts）：是处理所有消息和流数据的逻辑业务组件，可完成过滤、聚合、查询数据库等工作。

(6) 消息分发策略（Streaming Groupings）：定义了一个Topology中每个Bolt将会接受什么样的流作为输入。Storm提供了7种分发策略：随机分组（Shuffle Grouping）、按字段分组（Fields Grouping）、广播分组（All Grouping）、全局分组（Global Grouping）、不分组（Non Grouping）、直接分组（Direct Grouping）、本地或随机分组（Local or Shuffle Grouping）。

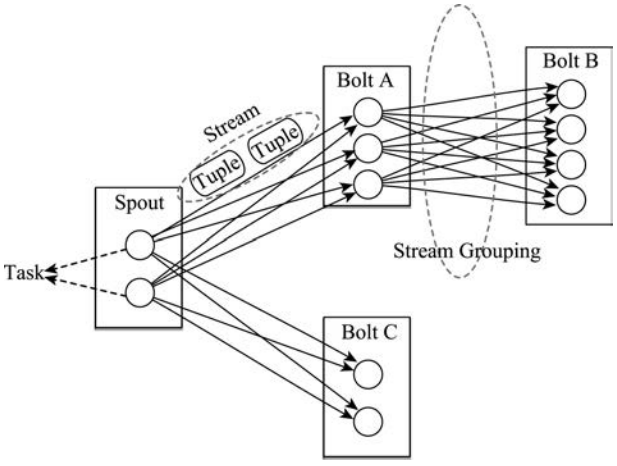

图1 Strom平台中各种组件之间关系

Fig. 1 The relationship between components in Storm

3 基于Storm的地理编码引擎设计

3.1 系统架构

本研究的目标是构建一个便于扩展、高可用、分布式的新型地理编码服务引擎。通过加入实时计算和流计算平台，实现低延迟处理爬取于网络中的多源地理大数据，实现地名库地址库、特征词库的及时更新；利用这些多种类多时态数据的实时挖掘分析，为地理编码提供高质量、高实效的基础资源；最终实现一个具有自适应能力的地理编码引擎，使地理编码的结果不仅具有更优秀的匹配率和准确率，而且具有时效性。

本引擎架构（图2）主要由4部分组成：数据获取、实时流处理、地理编码中间件和数据存储。

3.2 数据获取

对互联网上各类数据爬取是地址库持续更新的基础。本引擎中有2类爬虫：（1）主要爬取带有丰富地理信息的数据，如新浪微博和百度地图POI等。针对当前各类垂直电商，按照电商领域分别爬取相应类别中电商站点与APP中的商铺数据，如电影院POI主要爬取自猫眼电影和QQ电影票；（2）主要爬取互联网知识性文本数据（如wiki和百度百科）作为背景知识与素材建立相关实体之间关系，

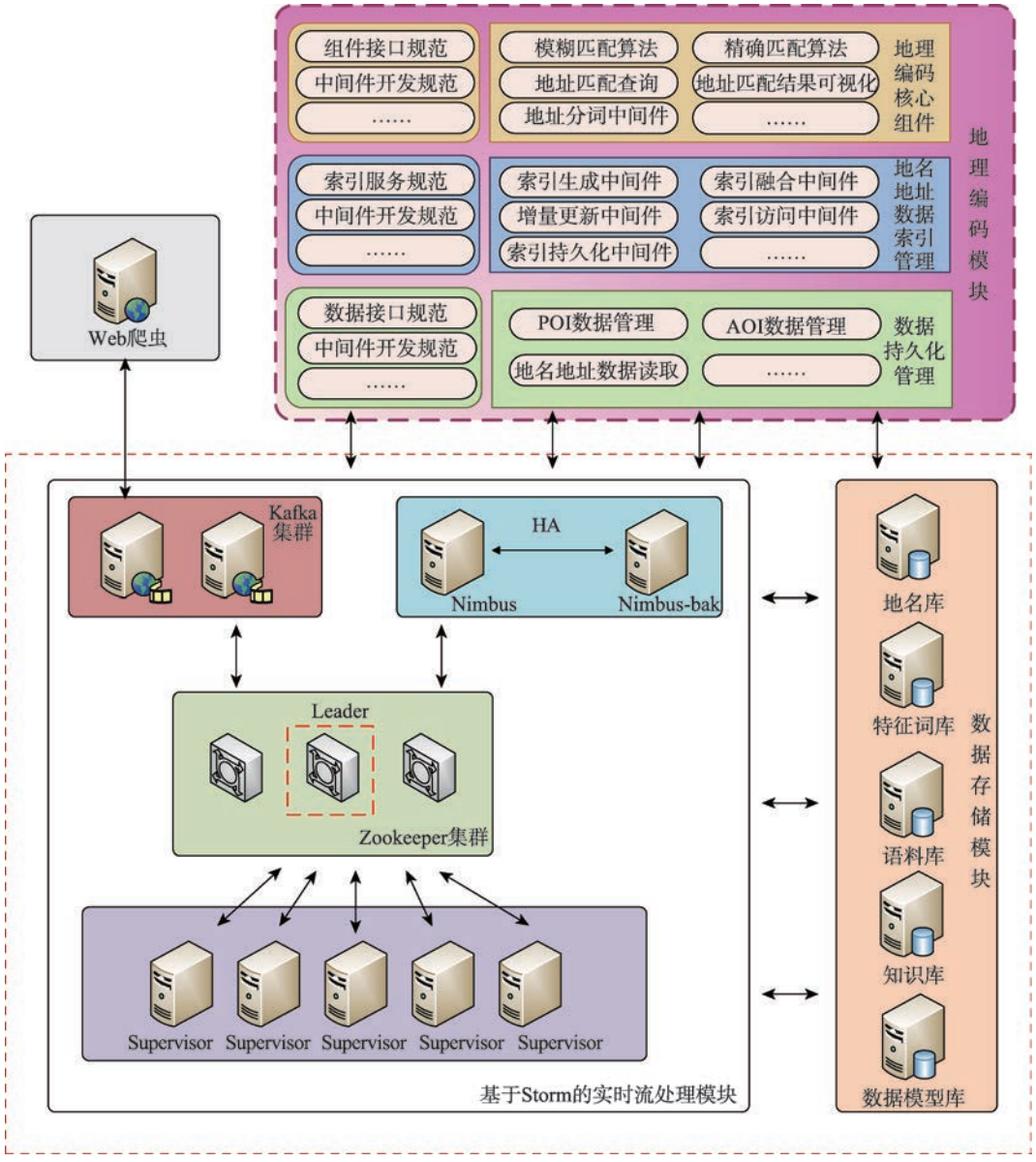

图2　系统架构

Fig. 2　System architecture

辅助地理编码的地址匹配。这些爬取的数据通过简单的处理后,会按照制定的格式分发到实时流处理模块进做一步的处理。

3.3 实时流处理

实时流处理模块是本引擎的核心部分,也是实现面向地理大数据实时处理的关键技术。其采用Kafka(http://kafka.apache.org)集群和Storm集群实现对爬取数据的流式处理和实时计算(图2)。其中,Kafka集群做为分布式消息队列,负责将数据发布到Storm集群中订阅的消息源中,而Storm集群中的计算拓扑实现对具体数据的处理(如过滤、标准化、特征词抽取、POI分类等)。Storm集群中Nimbus节点负责分配计算任务和监控集群状态,Supervisor节点负责执行具体的计算处理业务。而整个实时流处理模块由Zookeeper集群对各个工作节点进行分布式协调工作。

3.3.1 地理数据的预处理

对网络中获取的地理数据进行预处理主要有3个步骤:

(1)对数据进行过滤,去除数据中的噪声,提高数据质量。对于拼写错误、地址冗余和全半角混用可采用基本的文本处理方法解决;而地址缺失与地址歧义则需依赖其他属性与背景知识才能较为正确地处理。

(2)对爬取的POI进行融合和重新组织,生成候选POI集,最后根据候选POI集产生用于地理编码的地址库。POI融合以4个属性(名称、地址、POI分类和地理位置)为基础计算综合相似度。融合过程以百度地图和大众点评得到的综合POI为基础:首先,对各垂直领域的POI进行融合,生成各个细分领域的POI集合;然后,分别与综合POI融合;最后,转化为标准格式的地址数据。

(3)结合互联网上知识性文本,进行文本分词,计算得到词频统计和TF-IDF;基于规则模板和词频统计发现未登录词,完成特征词库的构建与更新;对这些文档进行命名实体抽取与分析,构建完善实体知识库,用于辅助地址库更新与地理编码。

3.3.2 Storm集群上实现基本空间处理

Storm是一个通用计算平台,默认不支持空间数据格式和相关操作。需先解决读入多种常用空间数据格式的数据源。WKT(Well-Known Text)和WKB(Well-Know Binary)是OGC制定的空间数据组织规范,使用WKT和WKB能很好地实现与其他空间数据格式和系统进行数据交换。

为了使Storm上的业务处理Bolt可完成基本的空间操作,引入JTS Topology Suite(http://www.vividsolutions.com/jts/JTSHome.htm)。它是一套开放源码的Java API,提供了整套完整的空间数据操作的核心算法。Storm优秀的架构设计,使得实现具体业务时基本不需考虑任务的分发和可靠性处理。因此,基于JTS Topology Suite可方便快捷地开发出分布式空间数据操作的API和相关具体业务。

由于爬取的空间数据往往文本内容较少,难以通过上下文语义完善地址描述信息,消除歧义。然而,使用其中包含的地理标签(Geotagged)进行空间划分,可明确所属行政区划,完善数据的地名地址信息,提高数据质量。图3描述了计算拓扑中,对地理数据进行空间划分的流处理过程。

算法1是图3中描述地理数据空间划分的具体方法。PoiSpout从Kafka消息队列中不断读取爬取

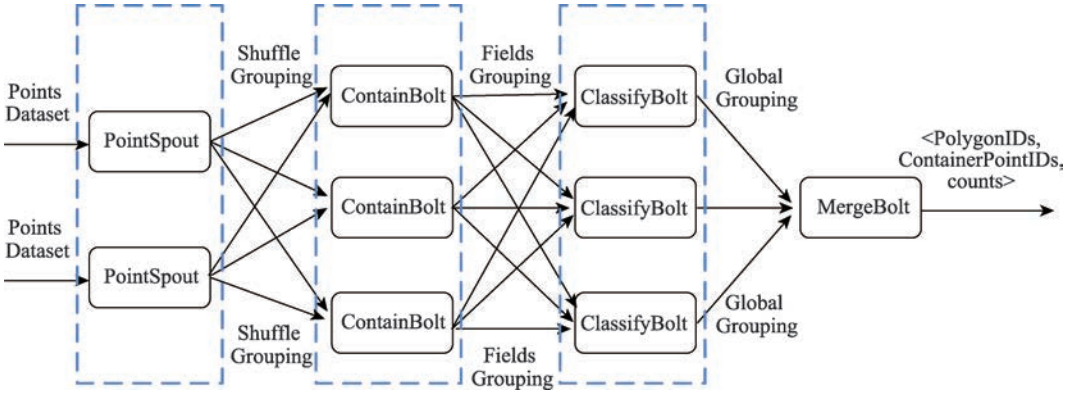

图3 Storm平台中空间分类的流处理过程

Fig. 3 The processing stream for spatial classification in Storm

到的地理数据，并将待分类的地理数据集合传入到处理流中。Contain方法由ContainBolt实现，主要根据数据的地理标签判定其所属的空间区域，然后根据空间区域对应的ID，将结果消息单元（Tuple）路由到指定的ClassifyBolt中，由其Classify实现空间区域和地理数据的关联映射。最后，将局部分类的结果实时传输至唯一的MergeBolt中，由Merge方法实现最终的统计和整合工作。对于整个处理流程，可通过在集群中实例化多个PoiSpout、ContainBolt和ClassifyBolt分布在不同的节点上实现处理能力的加速。

算法1　根据空间分布对点进行划分

输入：用于划分的点集合PointSet和目标多边形PolygonSet

输出：对应<PolygonID, ContainerPointIDs>列表，以及每个多边形中包含的点的数量

```
PolygonIDs ← getFeatures(PolygonSet)
< pointID, longitude, latitude> ϵ PointSet
/* 分组检测所属多边形过程 */
Contain(pointID, longitude, latitude, PolygonIDs)
point ← new Point(pointID, longitude, latitude)
for i ϵ PolygonIDs do
  /* 判断是否在多边形内 */
  if PolygonSet[i]contains point then
    tuple ← createTuple(i, point)
    emit(tuple)          /* 发送消息单元tuple */
  end
end
/* 空间划分过程 */
polygonKeyMap是一个HashMap结构用于记录对应多边形包含点数和点实例数组
Classify(polygonID, pointID)
if polygonID ϵ PolygonIDs then
  count, pointIDs ← polygonKeyMap.get(polygonID)
  count ← count + 1
  pointIDs.add(pointID)
  polygonKeyMap.store(count, pointIDs)
  emit(polygonIDs, count, pointIDs)
end
/* 合并统计过程 */
polygonMap是一个由polygonID作为键值，<包含点数, 点ID集合>为键的结构
Merge(polygonID, count, pointIDs)
if polygonID ϵ PolygonIDs then
  polygonKeyMap ← polygonMap .get(polygonID)
  polygonKeyMap.store(count, pointIDs)
  polygonMap.set(polygonID, polygonKeyMap)
end
return PolygonIDs, ContainerPointIDs, counts
```

3.4 地理编码中间件

地理编码中间件是整个引擎的关键组件，主要负责对地名地址数据的管理、地名地址索引的构建与维护、地址分词与匹配及结果可视化。这个模块中最重要的是进行地名地址的匹配，结合目前国内外地址匹配及地理模型的相关研究。本文采用精确匹配和模糊匹配2个阶段流程实现地址匹配工作（图4）。

由于中国现有地名、地址体系异常复杂，地址系统混乱、无序、缺乏规律性和统一的标准，加之建筑变化过快，使得门牌的缺号、跳号、重号现象比较严重。随着地理编码技术的普及，尤其是物流快递和高德地图、百度地图等电子地图应用的地址查询功能的广泛使用，用户进行地理编码的目的是得到某一具体地点上的相关实体（地标）的空间位置。所以，使用兴趣点（POI）和兴趣面（AOI）空间位置的地址等信息作为地名地址库，将地理编码匹配转化为检索匹配POI和AOI是可行有效的方法。因此，地理编码匹配模块主要采用点模型匹配和面模型匹配，使用爬取和实时处理的POI作为地址库，使用AOI和中国行政区划构建地名库，结合二者作为地理编码依赖的参照数据库。

由于随着时间的推移，会出现大量新的地名和组织机构，而且组织名称构成较为多样、不稳定、规

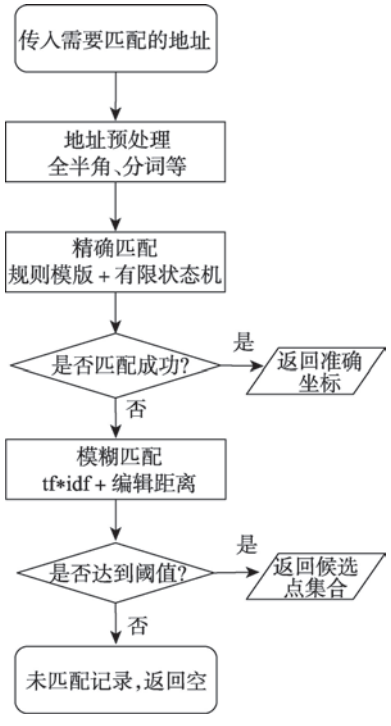

图4　地址匹配的流程

Fig. 4　Addresses matching process

律性差,导致传统方式的参照数据库中的地名和单位的名称地址滞后于社会的变化[22]。本引擎通过Storm实时流处理模块,通过收集网络中的POI和AOI,可近实时地构建更新地名库中的AOI信息和地址库对应的POI信息,地名库和地址库可覆盖较为全面的地名地址信息,使地理编码的结果更具时效性,拥有更高的匹配率和准确度。

3.4.1 精确匹配

精确匹配使用地名库,采用规则模版和有限状态机实现。首先对地址进行分词处理,然后分词结果的元组序列按照顺序传入由省、市、区、街道和门牌号5个状态节点构成的有限状态机。将元组和对应节点按照定义好的规则模版进行匹配,匹配成功则将后续元组序列传至下一个可达的状态节点继续匹配,直至终止状态。如果到达终止状态,那么该地址有效,否则无效。例如,“北京市海淀区颐和园路”是有效地址,而“武汉市湖北省鲁磨路”则是无效的。精确匹配对地址要素的顺序关系有严格限定,因此定位精度高,但匹配率较低。

3.4.2 模糊匹配

由于用于地理编码的地址数据存在多样性、复杂性和地址要素弱有序性,很难仅仅通过精确匹配的方法完成高质量的地理编码工作。因此,在地理匹配环节引入全文检索的文本匹配方法,以提升匹配率和准确度。模糊匹配部分使用TF-IDF模型和编辑距离(Levenshtein distance)[23]2种方法综合评判地址之间的相似度。

通过对互联网中采集到的地址和实际快递订单地址进行分析,发现除了传统的地址层级描述方法外,越来越多的地址以“行政区划+地理实体(地标)”的方式进行描述,如“北京市海淀区北京大学”。为了适应这种地址匹配的要求,并且充分利用POI和AOI数据的多种信息,在进行全文索引的构建和检索过程中,不仅对标准化的地址进行索引和检索,而且将实体名称、“行政区划+实体名称”和“标准地址+实体名称”同时进行索引。这样在检索过程中可更好地兼顾地址中包含组织实体名称的各种情况。

另外,地址的拼写错误是常见的错误之一,而TF-IDF算法自身对这类错误不敏感。为了解决此问题,本引擎引入编辑距离计算输入地址与候选地址之间的相似度,并综合TF-IDF评分和相似度得到最终评分与排序结果集合。

3.5 数据存储

这个模块主要是持久化引擎运行所需的各种数据资源,包括实现地理编码所依赖的地名地址库和特征词库、作为背景知识进行地址关联学习的知识库,以及用于词频统计和训练的语料库等。为了实现普通属性和空间特征的共同管理,本引擎采用PostgreSQL(http://www.postgresql.org)完成数据存储。

4 地理编码匹配实验与结果

4.1 数据集和Storm集群

本文使用网络爬虫收集百度的POI数据并存储在PostgreSQL数据库中。本次实验共爬取中国范围内4 065 884个POI记录。其中,POI的元数据包括ID、name、address、type、telephone、zipcode、longitude和latitude(表1)。通过对数据的分布进行基本分析可看出,POI数据覆盖范围较为全面,基本涵盖了中国所有省(区)(图5)。其中,广东、江苏、浙江和四川所包含的数据较多,而海南、宁夏和澳门的POI数据较少,不同省(区)之间POI数据量差距较大。这种数据量的差异也一定程度上造成不同省(区)地理编码结果匹配率的差异。

基于前面介绍的系统架构,在服务端安装部署Storm集群(Storm 0.9.3;Zookeeper 3.4.6;Kafka 2.10),在服务器集群上虚拟化出相应的节点(默认5个)并对其分配不同的角色:Nimbus、Supervisor、Zookeeper、Kafka(表2)。这种方法使整个系统的部署和扩展得到很大的提升,同时根据实际计算需求动态扩容服务节点,提供弹性计算资源。另外,这种方法使整个引擎可较为轻松方便地从本地迁移到Amazon EC2、Windows Azure和阿里云等主流云服务平台上。

表1 从百度爬取的原始POI元数据结构

Tab. 1 The metadata structure of Baidu POI data

字段	含义	举例
ID	唯一标识码	05324af8fb50b53e210220b1
Name	名称	北京大学
Address	地址	北京市海淀区颐和园路5
Type	类型/标签	高等教育,教育
Telephone	电话	(010)62752114
Zipcode	邮编	100871
Longitude	经度	116.298518
Latitude	纬度	39.993301

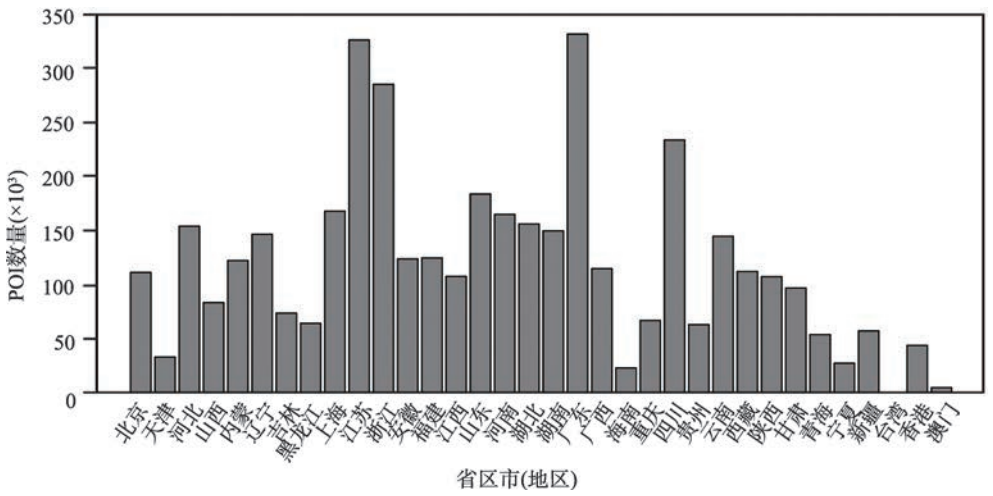

图5　POI数据按省区市(地区)分布情况

Fig. 5　The spatial distribution of POI based on Chinese provinces

表2　Storm集群中服务器的角色和配置

Tab. 2　The roles and configurations of 5 servers

服务器IP	角色	操作系统	配置信息
192.168.1.106	Nimbus;Kafka;Zookeeper	Ubuntu 14.04	1G内存;2.4GHz处理器;40G存储
192.168.1.107, 192.168.1.108	Supervisor	Ubuntu 14.04	1G内存;2.4GHz处理器;40G存储
192.168.1.121, 192.168.1.122	Supervisor;Zookeeper	Ubuntu 14.04	1G内存;2.4GHz处理器;80G存储

4.2　实时流处理的空间划分

POI类型属性有助于地名地址的深度分析和地址匹配，但由于爬取到的POI数据属性类别相对杂乱，因此需对其进行重新分类组合。本文采用上述实时流处理模块，对一些特定的POI类型关键字进行抽取分析(表3)，包括住宅、餐饮、公园、景点、自然山及高速公路。这些类型的POI是整个数据中出现频率较高，且具有一定代表性。经过重新分类组合后，POI的空间分布如图6所示。

实验对比Storm集群的空间划分和一台单机高配PC(64-bit操作系统，2.5GHz英特尔双核处理器，8G内存)空间划分能力得到结果如图7所示。整个

表3　抽取分类POI数据

Tab. 3　Extracting and classifying POI

POI类型	标识	关键字	数量
住宅	Residence	住宅,小区	99 722
餐饮	Catering	餐饮,休闲餐饮,西式快餐	294 106
公园	Park	公园	6118
景点	Scenic	风景区,旅游区,文物古迹,旅游景点	42 935
自然山	Mountain	自然山	262 859
高速公路	Highway	国道,高速道路	8795

性能测试，分别将1、5、10、15、20、25、30万个测试点集，划分到中国的345个城市行政区划中，基于Storm的空间划分按照算法1实现。从图7可清晰的发现，使用了Storm集群的空间划分在性能上远远超过了单机的处理方式，说明基于Storm实现的实时流处理模块的确可加速空间操作，更高效地处理实时获取的各类地理数据。另外，从图7也可发现5个节点(1个Nimbus和4个Supervisor)比3个节点(1个Nimbus和2个Supervisor)相比性能只有很小幅度地提升。这是由于当计算量较小时，任务分发到Supervisor，以及网络传输所消耗的时间比重大于节点处理任务的时间。因此，随着计算量的增大，处理业务更复杂，更多的Storm节点将会极大地提升引擎的处理效率。

4.3　地理编码原型

引擎的地理编码原型(图8)主要包括地名库地址库的管理、地址匹配索引管理、匹配可视化模块等功能模块。地名库地址库的管理和地址匹配索引模块使用Storm实时流处理后的各种地理大数据，作为地名地址数据进行统一的管理维护，并增

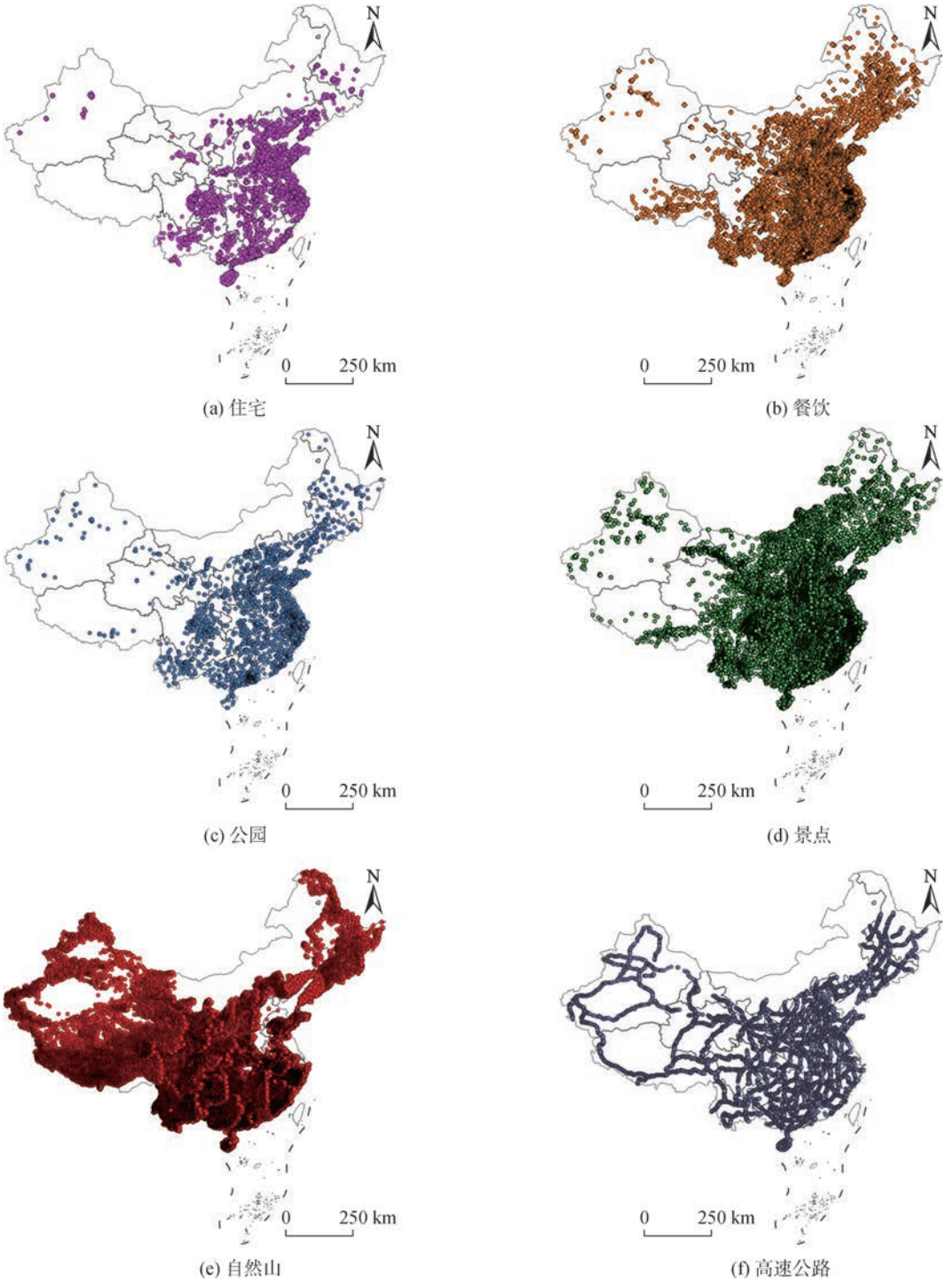

图6 各类型POI数据的空间分布

Fig. 6 The spatial distributions of POI with different types

量式地更新地址匹配的索引。地址匹配及其可视化模块用于地址交互式匹配,使用Bootstrap框架等技术提供简洁、友好的Web交互界面。此外,引擎通过REST方式提供Geocoding服务,实现不同系统之间的互操作。

为了衡量本引擎在地理编码方面的有效性,使用匹配率和准确率2个指标来衡量算法的优劣。地理编码实验中以爬取的4 065 884个全国范围POI为基础的地址数据,根据上述的方法对数据进行预处理,生成符合地理编码的标准地址模型。另外,选用全国省级、市级、县级、乡级和村级5级行政区划单元制作成地名库。这使地名库能覆盖大部分地

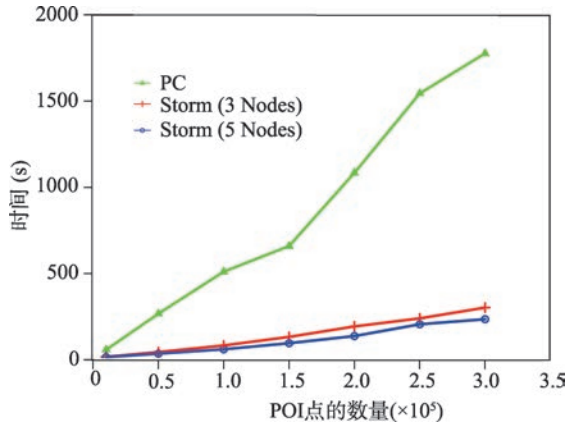

图7　数据处理速度对比图

Fig. 7　Comparison of time efficiency for single desktop PC and Storm clusters

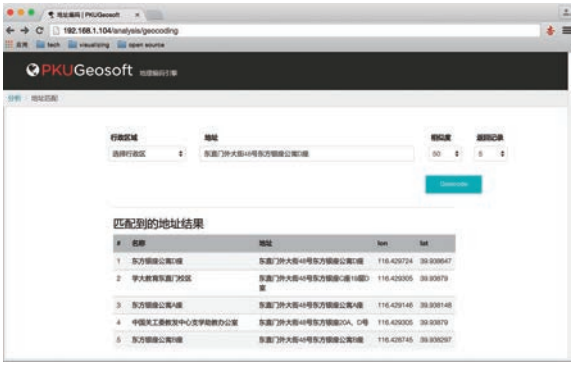

图8　地理编码引擎效果图

Fig. 8　The results of geocoding engine

址中出现的行政地名要素，提升准确匹配的概率。

随机抽取1万条POI数据的地址进行实验，与地址库中的记录进行完全匹配。另外，为了进一步检验引擎匹配的有效性，从地址库中抽取能完全准确匹配的地址作为基础，对其增加或去掉部分地址要素、调整地址要素的顺序，构造测试地址数据500条。地理编码匹配实验的统计结果如表4所示，其中，耗时是指平均匹配一条地址所用的时间。

实验表明，本引擎地理编码的匹配率和准确率均超过95%。通过分析发现，导致地址匹配失败的主要原因是背景知识不能很好地处理同一地点多种地址描述的匹配要求。而分词不正确也是地址数据在计算全文检索的相似度和排序时出现匹配错误的另一个重要因素。宋子辉[22]做过类似的中文地址匹配实验，其结果为匹配率98%，准确率93%，速度0.2 s/条。本引擎由于采用网络中的地理大数据作为地址库来源，使得地址库可以拥有数据来源丰富、质量高、覆盖范围广的数据。并且通过引入Storm实时流处理框架加速了由地理大数据到地址库的持续扩充过程，保证地址库数据的丰富性和时效性。由于地址匹配算法对地址库的质量由较强的依赖性，随着本引擎地址库持续丰富可有效地提升匹配的精度。

表4　地理编码匹配实验结果

Tab. 4　Geocoding matching results

地址来源	条数	匹配率(%)	准确率(%)	耗时(s/条)
随机抽取地址	10 000	100	97.3	0.075
人工构造地址	500	98.6	95.6	0.075

本引擎匹配速度在普通台式机上匹配效率大约是13条/s，达到了实用水平。匹配过程中，由于精确匹配部分使用了有限状态机，因此耗时较短，主要匹配开销在模糊匹配部分。为了使模糊匹配有较高的效率，采用了最新的Lucene实现全文检索，通过修改源码实现Lucene原生索引存储在cache中，减少索引文件反复的I/O读取，并采用一些工程优化方法提升匹配效率。由于Lucene本身不支持分布式，因此，通过Storm将索引数据的维护和匹配查询分布到不同的节点上，将匹配请求作为数据流，将会进一步缩短本引擎的匹配时间，提升匹配的吞吐量。

5　结论

本文基于实时流处理平台Storm，提出了地理大数据驱动的自适应地理编码引擎的构建思路和方法。相比于传统的地理编码引擎方法，该方法通过引入实时计算和流式计算的相关计算框架，实现了多源地理大数据的实时处理，加速地名地址库，以及相关资源的生成与更新。而且整个引擎架构具有良好的可扩展性，可方便地通过增加节点提速大量复杂的空间操作与分析，节省整个流程的处理时间，实现地理编码引擎的实时和自适应能力，不过该引擎还需进一步完善。在未来的研究工作中，还需深入探究如何通过多源数据辅助挖掘地名地址信息，解决同一地点多种地址描述的问题；通过引入机器学习的方法，实现POI与地址之间的关联学习与分析，从而更好地提升地理编码的准确度和效率。

参考文献：

[1] Goldberg D W. Advances in geocoding research and prac-

tice[J]. Transactions in GIS, 2011,15(6):727-733.

[2] 王凌云,李琦,江洲.国内地理编码数据库系统开发与研究[J].计算机工程与应用,2004(21):167-212.

[3] Grubesic T H. Sex offender clusters[J]. Applied Geography, 2010,30(1):2-18.

[4] Uhlmann S, Galanis E, Takaro T, *et al*. Where's the pump? Associating sporadic enteric disease with drinking water using a geographic information system, in British Columbia, Canada, 1996- 200[J]. Journal Water Health, 2009,7(4):692-698.

[5] Zandbergen P A. A comparison of address point, parcel and street geocoding techniques[J]. Computers, Environment and Urban Systems, 2008,32(3):214-232.

[6] Goldberg D W. Improving geocoding match rates with spatially- varying block metrics[J]. Transactions in GIS, 2011,15(6):829-850.

[7] 于焕菊,李云岭,齐清文.顾及实体空间关系的地址编码方法研究[J].地理与地理信息科学,2013,29(5):49-77.

[8] 孙亚夫,陈文斌.基于分词的地址匹配技术[C].中国地理信息系统协会第四次会员代表大会暨第十一届年会论文集.北京:中国地理信息系统协会,2007:1-13.

[9] Hutchinson M. Developing an agent-based framework for intelligent geocoding[M]. Perth: Curtin University of Technology, 2010.

[10] 程昌秀,于滨.一种基于规则的模糊中文地址分词匹配方法[J].地理与地理信息科学,2011,27(3):26-29.

[11] 于焕菊,齐清文,李云岭.街道的城市地址编码模型与实验[J].地球信息科学学报,2013,15(2):175-179.

[12] 张雪英,闾国年,李伯秋,等.基于规则的中文地址要素解析方法[J].地球信息科学学报,2010,12(1):9-16.

[13] Lu Y, Liu Y. Pervasive location acquisition technologies: opportunities and challenges for geospatial studies[J]. Computers, Environment and Urban Systems, 2012,36(2): 105-108.

[14] Li L, Goodchild M, Xu B. Spatial, temporal, and socioeconomic patterns in the use of Twitter and Flickr[J]. Cartography and Geographic Information Science, 2013,40 (2):61-77.

[15] Goldberg D W, Wilson J P, Knoblock C A. Extracting geographic features from the internet to automatically build detailed regional gazetteers[J]. International Journal of Geographical Information Science, 2009,23(1):93-128.

[16] Gao S, Li L, Li W, *et al*. Constructing gazetteers from volunteered Big Geo-Data based on Hadoop[J]. Computers, Environment and Urban Systems, doi:10.1016/j.compenvurbsys.2014.02.004, in press.

[17] Leibiusky J, Eisbruch G, Simonassi D. Getting started with Storm[M]. Sebastopol, CA: O'Reilly, 2012.

[18] Yang W, Liu X, Zhang L, *et al*. Big Data Real-Time Processing Based on Storm[C]. 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2013:1784-1787.

[19] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008,51(1):107-113.

[20] White T. Hadoop: The definitive guide (third ed.)[M]. Sebastopol, CA: O'Reilly, 2012.

[21] Zaharia M, Chowdhury M, Franklin M J, *et al*. Spark: cluster computing with working sets[C]. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 2010.

[22] 宋子辉.自然语言理解的中文地址匹配算法[J].遥感学报,2013,17(4):788-801.

[23] Levenshtein V I. Binary codes capable of correcting deletions, insertions, and reversals[J]. Soviet Physics, 1966,10 (8):707-710.

A Geocoding Engine Based on Storm

YU Jingyi, WU Lun and GAO Yong*

(*Institute of Remote Sensing and Geographical Information System, Peking University, Beijing* 100871, *China*)

Abstract: The explosion in geographical data with spatio-temporal characteristics has led a surge in the demand of adaptive geocoding engine construction driven by Big Geo-Data, when Web 2.0 techniques popularize and mobile devices that are capable of location-awareness become prevalent. The traditional geocoding service, which maintains gazetteers manually or semi-automatically by authoritative mapping agencies, cannot satisfy the needs of the latest researches. In order to solve the problems related to efficient storage and manipulation of massive Geo-Data in GIScience and related fields, our research proposes a method to build the adaptive geocoding engine in a geo-data-driven approach using Storm, a real-time and stream computing platform, thus to process multi-source network spatio-temporal data in real-time and accelerate the progression of building and maintaining gazetteers. Based on these data, an adaptive matching method of geocoding is built on the next stage. A prototype system of geocoding engine based on Storm is designed and implemented, which can process and geocode the multiple-source Geo-Data effectively. Experiments that were conducted on the POI datasets from Baidu reveals a high matching rate, which is more than 98%, and a accuracy rate of above 95%, while the average corresponding time per geocoding is about 75ms, which is practically applicable. The cases certify that real-time Storm-based streaming spatial operations not only consume an order of magnitude less time than traditional desktop stand-alone operations, but also enhance the matching rate and improve the positioning precision, which implies that the proposed solution is both feasible and practically effective. Our work offers new insights on collecting and processing POI datasets, enriching and building gazetteers, improving geocoding results in real-time with the use of Storm clusters. It makes contributions to apply real-time streaming computation methods to GIS for the state-of-the-art of Geo-Data computing, analytics and mining.

Keywords: geocoding; real-time stream processing; Big Geo-Data; Storm

***Corresponding author:** GAO Yong, E-mail: gaoyong@pku.edu.cn