基于 MongoDB 的轨迹大数据时空索引构建方法①

王 凯, 陈能成, 陈泽强

(武汉大学 测绘遥感信息工程国家重点实验室, 武汉 430079)

摘 要: 近年来, 随着计算机技术与无线传感器网络的发展, 轨迹大数据越来越得到人们的关注. 针对海量轨迹数据在存储与查询中出现的效率问题, 文章基于文档型非关系型数据库 MongoDB 提出了一套基于四叉树的道路网时空索引, 实现海量轨迹数据的高效查询. 通过对太原市 1915 辆出租车的 50 万条轨迹数据进行时空查询, 在不同数据量与不同并发数下测试道路网时空索引与 MongoDB 复合时空索引的效率表现. 实验结果显示道路网时空索引在数据量大于 10 万时有较好表现, 并能够适应不同并发数下的时空查询, 验证了道路网时空索引构建方法的可行性和高效性.

关键词: 非关系型数据库; 四叉树; 时空索引; 轨迹数据

Spatio-Temporal Indexing Method of Big Trajectory Data Based on MongoDB

WANG Kai, CHEN Neng-Cheng, CHEN Ze-Qiang

(State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China)

Abstract: In recent years, with the vigorous development of computer science and wireless sensor network, how to address big trajectory data is becoming a concerned issue increasingly. Because of massive trajectory data, there is an increasing focus on storage and search of big trajectory data. In view of this, based on the document type non relational database MongoDB, we propose a spatio-temporal index of road network which is based on quad-tree. For the 1915 taxis in Taiyuan, the 500,000 pieces of trajectory data are searched. With different data and different number of concurrency, we compare the efficiency of spatio-temporal index with that of MongoDB composite spatio-temporal index. Experimental results show that our method performs well when data volume is larger than 100000. It can adapt to spatio-temporal queries with different number of concurrency, proving that the method is feasible and efficient.

Key words: NoSQL database; quad-tree; spatio-temporal index; trajectory data

轨迹大数据是移动传感器数据的一种, 是指由 GPS 终端、智能手机等设备产生, 记录了移动对象的行为特征, 包括位置、时间、速度、方向等属性. 随着智能移动终端的广泛应用, 时空轨迹数据的存储、分析研究已受到学术界的广泛关注. 时空轨迹数据挖掘技术在许多领域得到了应用, 如交通协调与管理(如道路流量监控)、旅游路线推荐、自然灾害预警(如飓风预测)、环境保护(如空气质量监测)等[1].

最近几年轨迹数据的数据量正呈现逐年增长的趋势.因为数据动态更新、接入复杂、时空查询缓慢等问题, 轨迹大数据在存储与查询方面逐渐出现性能瓶颈. 这种需求引发了长期使用关系型数据库的用户的关心. NoSQL 数据库凭借对非结构化存储的支持以及较好的横向扩展能力, 在大数据的应用场景中颇受青睐[4]. 本文结合轨迹大数据中 GPS 数据轻量、非结构化的特点, 采用文档型非关系型数据库 MongoDB 存储与管理, 并设计了一套索引策略, 重点设计了基于道路网的网格时空索引[3], 通过先空间后时间的策略实现时空索引构建, 实现轨迹大数据的有序存储与高效查询.

① 基金项目:国家自然科学基金(41301441);中国博士后基金(2014M562050,2015T80829)
收稿时间:2016-09-07;收到修改稿时间:2016-10-19 [doi: 10.15888/j.cnki.csa.005770]

1　轨迹数据的MongoDB时空索引构建

轨迹数据的时空索引按照划分方法可以分为 3 类: (1)基于版本的索引方法, 采用先时间后空间的策略, 每个时间版本设计一套存储空间, 代表方法有 HR 树和 MV3R 树等, 时间查询效率高, 但存储成本高. (2)空间划分方法, 采用先空间后时间的策略, 用一维空间编码加上时间维度构建成为复合索引, 代表方法有 SETI 和 CSE 等. 这类方法构建效率较高, 但需预设空间范围, 索引结构不平衡, 且仅适用于点对象. (3)扩展时间维的多维索引方法, 采用空间和时间等同地位的策略, 在传统空间索引结构中直接增加时间维, 代表方法有 STR 树和 TB 树等, 在轨迹显示方面动态效果更好, 但是索引构建效率不高[5,6].

MongoDB 的默认索引是 B-tree(B 树)索引[7,8]. 本文中使用 B 树索引的 collection 和字段见表 1, 使用{"type" : 1}对 type 字段建立 B 树索引, “1”表示索引为升序.

表 1　单键索引的设置

字段	MongoDB 索引描述	说明
	{	用于单条件查询已存储数据.
sensor	{"sensor" : 1},	
time	{"time" : 1},	
foi.id	{"foi.id" : 1}	
foi.geo	{"foi. geo " : "2dsphere"}	
	}	

MongoDB 的常见的空间索引有 2d 平面索引与 2dsphere 球面索引[8]. 这两种索引均基于四叉树结合地理哈希值构建. 考虑到轨迹数据涉及时空信息, 常常需要查询特定空间、特定时间点或时段的轨迹数据, 单键索引不能满足高效时空需求, 本文建立了“传感器”+“时间”+“空间”的复合索引, 见表 2.

表 2　复合时空索引的设置

字段	MongoDB 索引描述	说明
	{	
Sensor	"sensor" : 1 ,	针对复杂观测区域的时空过滤.
time	"time" : 1 ,	
foi.geo	"foi.geo" : "2dsphere"	
	}	

本文将所有出租车数据作为点数据处理, 依据上节所提到的采取空间划分的方法构建时空索引, 采用先空间后时间策略, 首先将地理空间基于四叉树方法构建, 对不同层级的空间区域设计地理哈希编码 GeoHash 值, 用一维地理编码 GeoHash 值表示二维地理空间, 再加上时间维度形成复合索引. 考虑到上文所说的, 该类时空索引空间区域划分固定, 会因为区域分布不均而导致索引结构失衡而导致查询效率低. 具体二维地理空间编码见图 1.

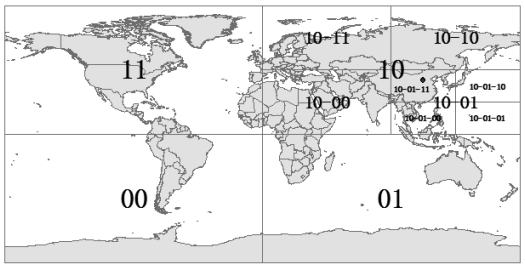

图 1　全球四叉树二维地理空间编码

针对以上问题, 本文采用预设中间层级的方法缓解此类问题. 轨迹数据沿城市道路网分布, 选择一个中间层级, 通过预先加入道路网的方法预设四叉树空间编码, 优先在道路网覆盖的空间范围内查找出租车数据, 这样将很大程度上改善因为索引结构不合理导致的效率低下问题. 中间层级的划分问题主要考虑两个方面问题的解决: “分布不均” 现象和“遗漏” 现象. 其中, “分布不均” 现象是指某些区块中出租车点数据较密集, 有些则较稀疏. 计算该中间层级每个区块中的点数, 计算均数 μ 和标准差 σ, 然后计算变异系数(Coefficient of Variation, Cv=σ/ μ), 该值越小代表该中间层级格网区域内的点数分布越均匀, 设置Cv值上限为 1.“遗漏” 现象: 某些出租车点数据不在所划定的范围内. 在概率论与数理统计中, “小概率事件”通常指发生的概率小于 5%的事件, 这里设置落入中间层级区块比例下限为 95%.

本文时空索引结构图如图 2 所示, 按照以上列出的规则, 选择一个四叉树的中间层级与道路矢量数据作叠加分析, 将道路网覆盖的区块提取出来, 编码成为道路网区块. 将每一个道路网区块为根节点, 构建新的四叉树索引, 记录地理哈希编码 GeoHash 值与时间, 用于查询属性信息. 这样每个出租车的点数据会首先定位到一个道路网区块, 而后再存储到对应道路网区块的四叉树结构中.

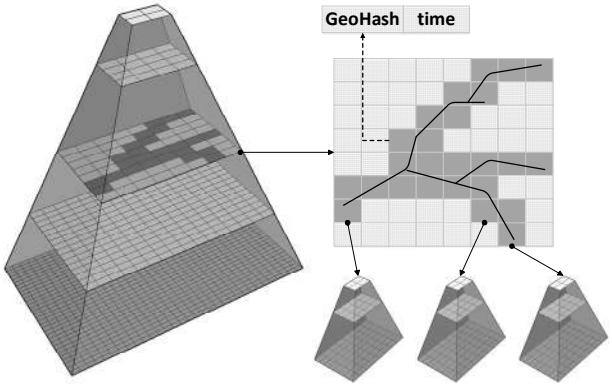

图 2 道路网时空索引结构图

2 实验

2.1 研究区域概况

太原市辖 6 个市辖区、3 个县, 代管 1 个县级市, 形成了 6 区 3 县 1 市的格局. 太原市位于山西省中北部的太原盆地, 北接忻州市, 东连阳泉市, 西交吕梁市, 南邻晋中市, 地理坐标东经 111°30′~113°09′, 北纬 37°27′~38°25′之间, 市中心位于北纬 37°54′, 东经 112°33′. 中心城区建成区面积 400 平方公里, 中心城区常住人口 300 万人, 流动人口 100 万人.

本文选用太原市中心城区作为研究区域, 中心城区的分区图即道路网分布如图 3 所示.

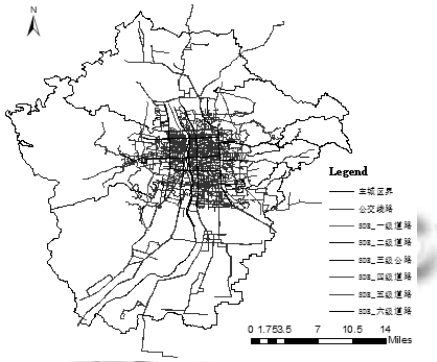

图 3 太原市中心城区分区图及其道路网分布

太原市城区的路网结构以网状为主, 呈环形放射状. 最外有环状高速路(外环路), 在老城区已经形成内环路, 形成棋盘式的格局, 横平竖直. 本文主要针对老城区的内环路区域研究轨迹数据.

2.2 测试数据

本文采用太原市 1915 辆出租车 50 万条轨迹点数据, 每次观测数据包含 5 个观测值: 车的 GPS 位置(空间坐标)、车速(数值)、车向(数值)、车的状态(文本)和观测时间(时间), 见表 3. 原始数据的观测频率约为 40 秒/次. 这些数据随机分布在太原市中心城区主干道附近.

表 3 太原市出租车轨迹数据示例

观测属性	类型	示例
GPS 位置	空间类型(经纬度点)	POINT(112.491376 37.937301)#4326
车速	数值类型(m/s)	24.01
车向	数值类型(°)	206.56
车状态	文本类型	“正常”
观测时间	时间类型	"2015-03-28T12:26:25.000+0000"

注: #4326 代表地理坐标系是 WGS-84 坐标系

2.3 格网划分与道路数据转化

随机抽取 50 万条轨迹数据中的 5000 条数据作为样本, 按照变异系数 CV 值小于 1, 点位落入道路网格至少 95%的标准, 最终确定道路网切分层为第 14 层, 网格大小为边长 0.021972656°的正方形区域, 约合实际地面边长 4.877765 km 的区域. 网格划分结果如图 4 所示, 道路网切块 178 块, 每一块建立一个新的四叉树索引.

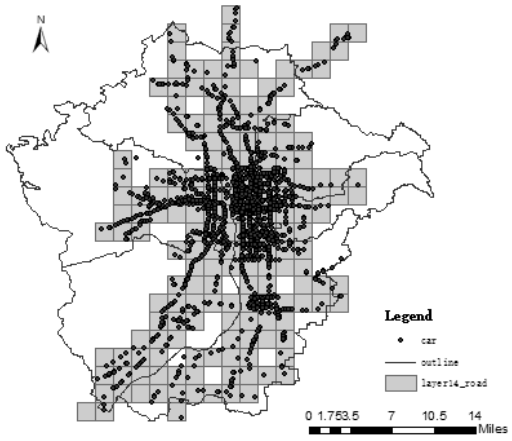

图 4 抽样数据四叉树索引第 14 层分区网格图

2.4 测试设计

本文设置以下两类测试场景综合评估索引性能. 随机选取感兴趣时段 “2015-04-01 09:29:08”-“2015-04-01 12:01:09”以及感兴趣的空间点(112.512874°E, 37.857032°N)与空间范围: 左下角坐标为 (112.512857°E, 37.856973°N), 右上角坐标为 (112.512943°E, 37.857059°N)的矩形包围盒, 测试时空过滤查询的效率.

(1) 不同数据量下的测试

数据量越大, 越能体现索引性能的优劣, 本文采用不同数据量测试索引在时空查询中的性能表现.

(2) 不同并发数下的测试

设置测试每秒并发用户数分别为 1、10、20、50、100、200, 测试本文索引性能.

2.5 测试环境与工具

(1) 硬件环境

采用选取 1 台服务器存储 MongoDB 数据, 一台超级本作为客户端, 硬件配置见表 4 和表 5.

表 4　服务器硬件配置

名称	配置
硬件	NF5270M3
CPU	Intel Xeon E5-2665 (2*8 核, 主频 2.40 GHz)
内存	32GB (DDR3, 1333MHz)
硬盘	405GB(SAS, 15000rpm)
网络	1Gbps

表 5　客户端硬件配置

名称	配置
硬件	Thinkpad X1 Carbon
CPU	Intel Core i5-5200U 2.2GHz
内存	4GB (DDR3, 1600Mhz)
硬盘	256GB(SSD, 600MB/s)
网络	1.0Gbps

(2) 软件环境

测试采用开源测试工具 Apache JMeter 测试 MongoDB 的 2d 平面空间索引与 2dsphere 球面空间索引构建的复合时空索引与本文道路网时空索引的效率, 软件环境如表 6 和表 7 所示.

表 6　服务器软件环境

软件	类型	版本
系统	Windows Server	Windows 2008 R2 Server Enterprise (64-bit)
JDK	Java 运行环境	1.8.0(64-bit)
MongoDB	数据库	3.0 (WiredTiger)

表 7　客户端软件环境

软件	类型	版本
系统	Windows 7	Ultimate(64-bit)
JDK	Java 运行环境	1.8.0(64-bit)

3　测试结果与分析

3.1 不同数据量的测试分析

按照不同数据量, 分别对三种方式构建的时空索引进行效率测试: 第一种方式使用 MongoDB 的 2dshpere 二维球面空间索引, 加上时间维度和传感器 id 构建复合时空索引; 第二种方式使用 MongoDB 的 2d 二维平面空间索引, 加上时间维度和传感器 id 构建复合时空索引; 第三种方式使用本文基于四叉树方式改进的道路网格时空索引. 实验结果见表 8.

从效率测试结果发现, 每种索引在查询的数据量增大时, 时空查询响应时间也相应呈现上升趋势. 第一种用 2dsphere 球面空间索引构建复合时空索引的方式, 在数据量小于 10000 时的时空查询效率良好, 但是随着数据量的增大, 该索引愈渐显示出劣势, 查询异常缓慢. 第二种用 2d 平面空间索引构建的复合时空索引的方式, 在不同数据量下测试效率一直较好, 随着数据量的增大, 该索引的效率略有下降, 但是相比第一种方式构建方式效率高. 第三种基于四叉树道路网格构建索引的方式在数据量较小的时候效率比前两种方式要慢, 因为匹配道路网区块需要耗费一段相对固定的时间(35ms-45ms). 但是随着数据量的增大, 在处理 10 万数据量以上的时空查询时, 该方式构建的索引要比前两种方式时空查询效率高.

表 8　不同数据量时空查询响应时间

数据量	响应时间(ms)		
	MongoDB 复合时空索引 (2dsphere)	MongoDB 复合时空索引(2d)	道路网格时空索引
500	38	36	53
1000	45	44	59
5000	49	45	69
10000	61	49	75
50000	158	51	77
100000	227	92	80
500000	1016	116	91

3.2 不同并发数下的测试分析

使用 Jmeter 测试工具, 对 50 万条出租车轨迹数据在 1、10、20、50、100、200 的并发数下, 测试时空查询的响应时间. 每次请求响应完毕后立即进行下一次请求, 分别连续重复测试 5 次, 以尽量减少个例异常对结果的影响. 根据聚合报告中“90% Line” 参数(90%用户的响应时间), 在指定时间段及空间范围车辆信息查询的实际场景中, 比较 MongoDB 复合时空索引与本文索引方法的效率. 实验结果见表 9.

表 9 不同并发数时空查询响应时间

并发数	响应时间(ms)		
	MongoDB 复合时空索引 (2dsphere)	MongoDB 复合时空索引 (2d)	道路网格时空索引
1	1016	116	91
10	3855	128	103
20	8305	166	126
50	11804	218	131
100	22753	272	159
200	51318	316	198

测试结果表明三种索引构建方式在并发数增大时, 时空查询的响应时间均呈现上升趋势. 尤其采用第一种用 2dsphere 构建时空索引的方式耗时增长幅度最大, 并在并发数为 200 时出现了等待 51s 的情况, 已经远远超出了用户可接受范围. 2d 方式构建的时空索引和本文道路网时空索引一直表现良好, 并且本文索引一直略优于采用 MongoDB 的 2d 平面索引方式构建的时空索引. 该优势在并发数越大时越明显. 在并发数为 200 时, 本文索引的时空查询响应时间为第二种索引构建方式的 62.66%.

3.3 综合分析

综合以上实验结果, 发现如下结论:

(1) 合理的索引构建方式对索引性能发挥很重要. 同样是用 MongoDB 的空间索引构建的复合时空索引, 在不同数据量以及不同并发数下, 使用 2d 平面索引方式构建的时空索引的测试结果都要远好于使用 2dsphere 球面空间索引方式构建的时空索引.

(2) 当索引数量较小时, 索引的优势很难体现. 当数据量越大的时候, 本文索引优势越明显.

(3) 当并发数较小时, 也很难看出索引的优势. 并发用户较多时, 索引优势逐渐体现.

(4) 本文道路网索引在查询时首先需要匹配道路网区块, 在处理小数据量数据时并不具有优势. 随着数据量的增大, 每个道路网格下的四叉树逐渐增加深度. 由于道路网格的合理分块, 每一个区块下的四叉树的深度增长也相对平衡, 因而可以保证合理的查询效率, 通过表 8 与表 9 的实验测试结果可以得到验证.

4 结语

本文基于文档型非关系型数据库 MongoDB 实现对海量轨迹数据的存储, 并以太原市中心城区为例, 构建索引策略, 根据太原市中心城区道路网设计了基于四叉树的道路网时空索引, 对比 MongoDB 的自带方式构建的复合时空索引, 通过时空查询响应时间证实索引的有效性, 为非关系型数据库在轨迹数据时空索引的应用上提供重要参考.

参考文献

1 吉根林,赵斌.时空轨迹大数据模式挖掘研究进展.数据采集与处理,2015,30(1):47–58.

2 陈能成,张伟杰,王晓蕾.传感器观测服务的性能评估与方法.测绘通报,2015,(4):61–64.

3 邓泽,刘汪洋,陈丹.一种面向动态连续查询的查询索引.计算机应用与软件,2015,32(12):8–11.

4 Sadalage PJ, Fowler M. NoSQL distilled: A brief guide to the emerging world of polyglot Persistence-Addison-Wesley professional. Addison-Wesley Professional, 2012: 11.

5 Nguyen-Dinh L, Aref W, Mokbel M. Spatio-temporal access methods: part2(2003-2010). IEEE Data Engineering Bulletin, 2010, 33(2): 46–55.

6 Comer D. The ubiquitous B-tree. ACM Computing Surveys (CSUR) , 1979, 11(2): 121–137.

7 龚俊,柯胜男,朱庆,等.一种集成 R 树、哈希表和 B*树的高效轨迹数据索引方法.测绘学报,2015,44(5):570–577.

8 程显峰.MongoDB 权威指南.北京:人民邮电出版社,2011.

9 张尧,甘泉,刘建川.基于 MongoDB 的地理信息共享数据存储模型研究.测绘,2014,(4):147–150.