

文章编号: 1000-5641(2018)05-0120-15

基于分布式平台 Spark 的空间文本查询分析

徐 阳¹, 王志杰², 钱诗友¹

- (1. 上海交通大学 计算机科学与工程系, 上海 200240;
2. 中山大学 数据科学与计算机学院, 广州 510006)

摘要: 随着基于位置服务应用的不断推广, 空间文本数据查询的应用价值(例如结合地理位置和用户标签的社交推荐)也在不断提高. 但是, 随着数据规模的迅速增长, 传统的基于单机环境实现的技术难以为用户提供低延时和高吞吐量的服务. 为此, 本文基于 Spark 平台对分布式环境下的空间文本查询算法进行了探究. 采用了面向海量空间文本数据的两层索引框架(包括全局索引和局部索引), 该框架利用了分阶段过滤的策略来处理分布式下的布尔范围查询问题. 同时, 针对空间文本相似连接提出了 Prefix-RI 结构并提出了相应的分布式算法. 基于 Spark 平台实现了所提出的分布式算法, 并通过大量的实验对比验证了所提出方法的优越性.

关键词: 分布式计算; 空间文本分析; 相似连接

中图分类号: TP311 **文献标志码:** A **DOI:** 10.3969/j.issn.1000-5641.2018.05.010

Distributed spatio-textual analytics based on the Spark platform

XU Yang¹, WANG Zhi-jie², QIAN Shi-you¹

- (1. Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China;
2. School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China)

Abstract: With the rapid development of location-based services, spatio-textual data analytics is becoming increasingly important. For instance, it is widely used in social recommendation applications. However, performing efficient analysis on large spatio-textual datasets in a central environment remains a big challenge. This paper explored distributed algorithms for spatio-textual analytics based on the Spark platform. Specifically, we proposed a scalable two-level index framework, which processes spatio-textual queries in two steps. The global index is highly scalable and it can retrieve candidate partitions with only a few false positives. The local index is designed based on pruning ability of infrequent keywords and used for each candidate partition. We implemented the proposed distributed algorithms in Spark. Extensive experiments demonstrated promising performance for the

收稿日期: 2018-07-09

基金项目: 国家重点研发计划项目(2017YFC0803700); 广东省科技计划项目(2015A030401057, 2016B030307002)

第一作者: 徐 阳, 男, 硕士研究生, 研究方向为分布式计算、大数据处理.

E-mail: xuyangit@sjtu.edu.cn.

通信作者: 王志杰, 男, 博士, 副研究员, 研究方向为数据挖掘等. E-mail: wangzhij5@mail.sysu.edu.cn.

proposed solution.

Keywords: distributed processing; spatio-textual analytics; similarity join

0 引 言

随着基于位置服务应用的不断推广, 这些应用也同时产生了海量的空间文本数据. 通常来讲, 每条空间文本数据包括一条空间坐标信息和一个文本标签集合. 例如, 餐馆的经纬度和餐馆的特色标签集合; 用户的当前位置和用户兴趣标签集合等. 基于这些数据, 用户可以通过空间和文本两个维度的过滤找到感兴趣的空間文本对象. 本文主要研究了以下 2 种空间文本查询. ① 布尔范围查询^[1], 该查询用于找到附近以 ϵ 为半径的圆形范围内包含给定关键字集合的空间文本对象. 例如, 用户可以搜索附近 1 km 内包含咖啡、奶茶等标签的餐馆. ② 空间文本相似连接^[2], 该查询对数据集的每个空间文本对象, 在与其连接的数据集中找到满足与该对象空间距离小于等于 ϵ 且文本标签集合 Jaccard 相似度^[2]大于等于 θ 的相似对象. 例如, 社交应用中需要为每个用户推荐附近 100 m 内与该用户有相似兴趣的潜在好友. 可以发现, 上述查询分析在现实生活当中有着重要的应用价值. 然而, 随着数据量的不断增长, 如何在海量的数据场景下高效地进行上述查询分析是十分具有挑战性的. 首先在单机环境下, 由于内存大小的限制, 数据量的迅速递增导致了更多的磁盘读写操作, 进一步导致了较高的处理延时; 另外, 随着数据量的迅速递增, 单机环境每分钟能够处理的请求数(吞吐量)也在不断下降.

近年来, 随着基于内存的分布式计算平台 Spark^[3] 的流行, 有许多研究工作^[4-7] 基于该平台探索了海量数据场景下的分布式解决方案. 相对于基于 Hadoop 的解决方案, 基于 Spark 平台的解决方案可以达到更低的延时和更高的吞吐量. 例如, 最近有文献提出了基于分布式内存计算的空间数据分析系统^[4]. 该系统相比于传统的数据库管理系统和基于 Hadoop 的分析系统(如 Hadoop GIS^[8]), 在空间数据的查询性能方面提高了 5~100 倍^[4]. 考虑到上述优越的特性, 本文采用 Spark 作为实现平台, 并在该平台下研究如何对上述的 2 种空间文本查询进行高效处理. 同时, 由于许多分析人员倾向于通过 SQL 语言去进行数据分析工作, 文章为上述查询提供了便捷的 SQL 接口.

另一方面, 在查询分析系统中, 索引技术有着重要的意义. 普通用户的查询解集合通常只涵盖了查询数据集的一小部分, 在这样的场景下, 索引可以通过较好的剪枝能力加速查询, 从而提升整体性能. 由于 Spark 是基于 Master-Slave 架构的分布式平台, 如何基于此设计面向空间文本数据的两层索引技术是一个待深入探索的问题. 其中, 全局索引需要过滤掉不可能满足给定查询的数据分区; 然而由于全局索引存储于 Master 节点, 受限于单台节点的内存限制, 需要探索有较好空间扩展性的全局索引方法才能应对海量数据场景的挑战. 除此之外, 局部索引对给定查询进行处理后, 通常将结果收集到 Master 节点; 由于传统的空间文本索引技术没有充分利用低频词汇的剪枝作用, 且空间开销较大, 因此, 需要设计高效的局部索引方案进行进一步的优化处理.

概括来说, 本文的主要贡献如下.

- 基于 Spark 平台实现了面向海量空间文本数据的两层索引框架, 该框架通过分阶段过滤的策略较大地提升了性能.
- 针对空间文本相似连接问题, 结合前缀过滤技术提出了 Prefix-RI 结构.

- 基于 Spark 平台, 提出了针对上述查询的分布式处理算法, 并与其他解决方案进行了充分的实验对比.

接下来, 本文将在第 1 节介绍问题定义、相关背景知识及先前的工作; 第 2 节介绍解决方案的整体框架; 第 3 节和第 4 节分别对局部索引和全局索引技术进行介绍; 第 5 节主要介绍提出的分布式处理算法; 第 6 节介绍实验结果和分析, 最后在第 7 节对全文进行总结.

1 问题定义及相关工作

1.1 问题定义

假定 \mathcal{D} 为一个空间文本数据集, \mathcal{D} 中每个空间文本对象 o_i 可由一个欧氏空间的二维点坐标 $\rho_i = (x_i, y_i)$ 和字符串集合 γ_i 组成; 其中, γ_i 的字符串个数为 $|\gamma_i|$. 因此, 一个包含了 n 个空间文本对象的数据集 \mathcal{D} 可以表示为: $\{(\rho_1, \gamma_1), \dots, (\rho_n, \gamma_n)\}$.

一个空间文本查询 Q 通常包括一个空间查询 Q_s 和一个文本查询 Q_t . 对于空间查询 Q_s , 文章主要关注常用的圆形范围查询. 圆形范围查询可由 (τ, ϵ) 定义, 其中 τ 为查询圆中心, ϵ 为查询半径. 对于文本查询, 在布尔范围查询中, 文本查询 Q_t 可以由一个目标字符串集指定, 表示必须包含目标字符串集合中的所有字符串才符合文本查询条件; 在空间文本相似连接中, 其定义为两个空间文本对象的字符串集合 γ_j 和 γ_k 的 Jaccard 相似度. 如果 γ_j 和 γ_k 的 Jaccard 相似度大于等于给定的阈值, 则满足文本查询条件. 下面给出具体的定义.

定义 1.1 假设 $dist(\rho, \tau)$ 为点 ρ 和 τ 的欧氏空间距离, 则布尔范围查询 $Q = \{Q_s = (\tau, \epsilon), Q_t\}$ 将在数据集 \mathcal{D} 中找出满足查询的子集 θ , 使得

$$\theta = \{o_i \in \mathcal{D} \mid dist(\rho_i, \tau) \leq \epsilon \wedge Q_t \subseteq \gamma_i\}.$$

定义 1.2 两个空间文本对象 o_j 和 o_k 的文本相似度为其字符串集合 γ_j 和 γ_k 的 Jaccard 相似度, 其为

$$sim_t(o_j, o_k) = \frac{|\gamma_j \cap \gamma_k|}{|\gamma_j \cup \gamma_k|}.$$

其中, $|\gamma_j \cap \gamma_k|$ 表示两个字符串集合相同的个数, 且 $|\gamma_j \cup \gamma_k| = |\gamma_j| + |\gamma_k| - |\gamma_j \cap \gamma_k|$. 易知 $0 \leq sim_t(o_j, o_k) \leq 1$: 当 γ_j 和 γ_k 没有相同字符串时, $sim_t(o_j, o_k)$ 为 0; 当 γ_j 和 γ_k 相同时, $sim_t(o_j, o_k)$ 为 1.

定义 1.3 给定空间文本数据集 \mathcal{D}_1 和 \mathcal{D}_2 , 假定空间距离阈值为 ϵ 且文本相似度阈值为 λ ($0 \leq \lambda \leq 1$), 空间文本相似连接将找出

$$\theta = \{o_i \in \mathcal{D}_1, o_j \in \mathcal{D}_2 \mid dist(\rho_i, \rho_j) \leq \epsilon \wedge sim_t(o_i, o_j) \geq \lambda\}.$$

1.2 相关工作

空间文本查询. 空间文本查询(如上述的布尔范围查询)近年来已经被广泛研究^[9-11]. Zhou 等^[12]将文本搜索的研究扩展到空间数据库, 他们提出了两个混合的空间文本索引, 其将 R*-tree^[13] 和倒排列表松散地组合, 并进一步提出了 Inverted file-R*-tree 结构和 R*-tree-Inverted File 结构. Chen 等^[1]提出了紧密结合空间索引和倒排文件的 IR-tree 结构. 他们使用了一个 R-Tree^[14], 并将倒排列表集成到 R-tree 的每个节点, 对相应子树的文本内容进行索引. 除了布尔范围查询外, 也有很多工作致力于研究结合了空间相似度和文本相似度的 top- k 问题. Wu 等^[15]提出了一种基于密度的方法来找出 k 个最相似的区域; Choudhury 等^[16]进一

步研究了如何对多个 top- k 查询组成的批查询进行优化. 上述工作都是基于欧氏空间下的研究, 而 Luo 等^[17]进一步研究了路网下的空间文本查询. 另外, Chen 等^[18]还研究了流数据场景下的空间文本查询处理.

空间文本相似连接. 相似连接在许多领域有着广泛的应用^[19-20], 文本相似连接和空间相似连接都是研究热点. 文本相似连接寻找两个文本数据集中所有满足相似度大于等于给定阈值的解对, 常用的相似度量函数包括 Jaccard 相似度, Cosine 相似度等; Sarawagi 等^[21]首次提出基于倒排列表支持文本相似连接, 对于每个文本对象, 将其包含的字符串映射到该对象; 在构建完该列表后, 通过扫描当前对象包含的各个字符串所映射的列表, 并累计与自身的重复元素个数来计算相似性. 为了进一步减少需要扫描的字符串个数, 文献 [22] 进一步提出了前缀过滤技术. Bayardo 等^[23]进一步提出了支持 Jaccard 相似度和 Cosine 相似度的前缀过滤技术. 在空间相似连接方面, 由于 R-Tree 索引技术在该领域的广泛应用, 常见的连接算法通常基于 R-Tree 实现. 在文献 [4] 中, 作者基于 Spark 平台和 R-Tree 索引对基于距离的空间相似连接和 k 最近邻空间相似连接进行了分布式的高效支持.

分布式分析系统. 由于数据规模的不断增长, 分布式的计算框架现在越来越受欢迎. Apache Spark^[3]是一个基于内存计算的分布式计算框架. 自发布以来, 在机器学习、图处理和流数据处理方面起着重要作用. 与面向磁盘和批量处理的 Hadoop 平台相比, Spark 可以使用分布式内存缓存和计算提供低查询延迟和高吞吐量的服务. Xie 等^[4]基于 Spark 提出了基于分布式内存计算的空间数据分析系统. 该系统实现了一系列面向海量空间数据的查询操作(圆形范围查询、 k 最近邻查询及空间相似连接等). 与其他空间数据分析系统如 Spatial Hadoop^[5]相比, 在真实数据集下的实验证明了其性能优越于其他的主流空间数据分布式分析系统.

1.3 背景知识

前缀及长度过滤技术^[22]. 给定字符串集合 γ 和 Jaccard 相似度阈值 λ , 可以为其生成长度为 $|\gamma| - \lceil \lambda \times |\gamma| \rceil + 1$ 的前缀. 对于两个字符串集合 γ_j 和 γ_k , 如果其 Jaccard 相似度大于等于 λ , 则在统一的字符串排序下, γ_j 和 γ_k 的前缀应该至少有一个字符串是相同的. 基于此, 可以大大减小搜索的字符串个数. 此外, 还可以通过长度过滤技术进行预过滤. 对于 γ_j 和 γ_k , 如果这两者的 Jaccard 相似度大于等于 λ , 则有 $\frac{|\gamma_j \cap \gamma_k|}{|\gamma_j \cup \gamma_k|} \geq \lambda$, 既 $|\gamma_j \cap \gamma_k| \geq \lambda \times |\gamma_j \cup \gamma_k|$. 根据集合的性质, 易知 $|\gamma_j| \geq |\gamma_j \cap \gamma_k|$ 且 $|\gamma_j \cup \gamma_k| \geq |\gamma_k|$, 所以进一步得到

$$|\gamma_j| \geq |\gamma_j \cap \gamma_k| \geq \lambda \times |\gamma_j \cup \gamma_k| \geq \lambda \times |\gamma_k|,$$

同理可证 $|\gamma_k| \geq \lambda \times |\gamma_j|$. 因此, 可以对 $|\gamma_j|$ 和 $|\gamma_k|$ 的大小关系进行验证, 从而初步排除不满足相似性要求的解.

布隆过滤器技术^[24]. 一个布隆过滤器由 n 个二进制位 ($\{\mathcal{B}[1], \dots, \mathcal{B}[n]\}$) 组成, 这些二进制位最开始全部初始化为 0. 通常来讲, 布隆过滤器使用一组独立的 k 个哈希函数 $\{h_1, \dots, h_k\}$. 对于需要索引的每个元素, 首先, 通过各个哈希函数将该元素映射到 $[1, n]$ 区间内的一个数; 然后, 置对应的二进制位为 1. 为了检查一个给定元素是否存在, 可以通过检查哈希函数组对该元素哈希后的各个对应二进制位是否值为 1. 如果全为 1, 则该元素有很大概率存在于布隆过滤器中, 否则一定不存在. 假定索引的数据集包含 m 个元素, 则布隆过滤器误报(false-positive)的概率为

$$Pr_{\text{false-positive}} = \left(1 - \left(1 - \frac{1}{n}\right)^{mk}\right) \approx (1 - e^{-\frac{km}{n}}).$$

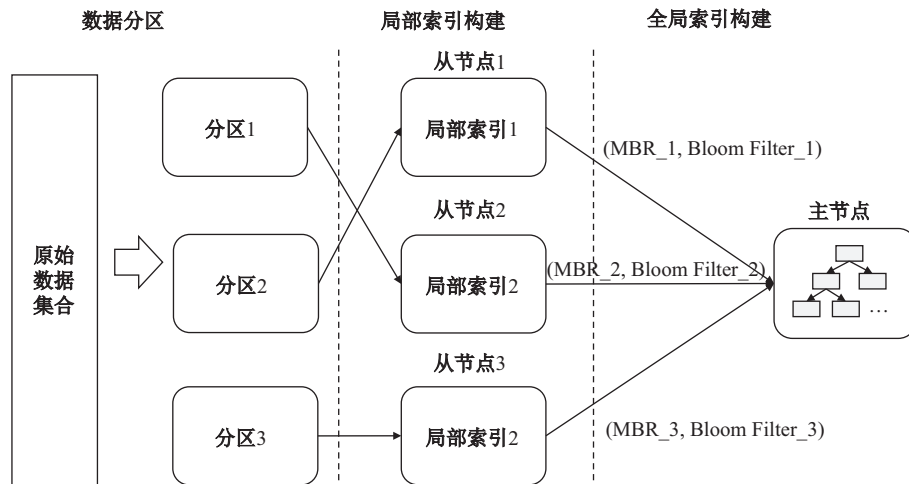


图1 基于Spark的两层索引框架

Fig.1 A two-level index framework based on Spark

2 整体框架

2.1 查询接口

通过扩展 Spark SQL 提供的接口, 本文为 2 种空间文本查询提供了便捷的 SQL 接口。例如, 给定一个布尔范围查询, 想找到离点 (20, 2) 距离在 3 km 内且包含字符串“咖啡”的空间文本对象, 那么对应的 SQL 示例如下所示,

SELECT * FROM t WHERE POINT(t.x, t.y) IN CIRCLERANGE(POINT(20, 2), 3) AND CONTAINS(t.s, “咖啡”).

2.2 框架细节

结合 Spark 平台 Master-Slave 结构的特点, 本文采用了面向空间文本查询的两层索引框架(见图 1)^[4,25]。该框架利用了分阶段过滤的策略。首先, 通过全局索引过滤掉大部分不可能包含查询解集的分区; 然后, 进一步利用局部索引技术在每个候选分区上进行进一步的加速从而达到性能提升。下面对分布式索引的 3 个构建过程进行详细阐述:

数据分区。分区阶段将数据拆分并映射到集群的各个节点。一般来说, 分区策略主要关注以下 3 个问题: ① 数据局部性, 在空间上相近的数据或者文本相似度较高的数据对象应分配给同一分区; ② 负载均衡, 所有分区的大小应大致相同; ③ 可扩展性, 分区的时间成本应该是可以接受的。在本文的问题背景下, 通过空间相似性或文本聚类分区是直观的策略。然而, 与通过空间相似性分区相比, 基于文本聚类的分区太过耗时。在实验中, 发现通过空间相似度分区的平均耗时仅为 34 s, 而基于文本聚类分区的平均耗时为数个小时; 显然, 前者具有更好的可扩展性。因此, 选择了基于空间的分区策略, 称为 STRPartitioner^[4-5]。STRPartitioner 实现了 Sort-tile-recursive 算法^[26]的第一次迭代, 以确定分区边界, 即最小边界矩形(MBR)。基于这些 MBRs, 可以进一步构建一个临时的 R-tree 来将每个对象映射到特定的分区。给定分区数目后, Sort-tile-recursive 算法会根据数据的空间相似性进行分区(数据局部性); 且通常来讲, 分区后的各个分区大小基本相近(负载均衡)。因此, STRPartitioner 很好地解决了上述 3 个问题。

局部索引构建. 对于每个分区, 可以构建局部索引对查询进行加速. 此外, 需要收集每个分区的空间边界信息(每个分区的 MBR)和文本统计信息, 以进一步构建全局索引. 它们是(id, MBR, β)的形式, 其中 id 标识分区, β 表示每个分区的文本摘要数据用以在全局索引上引入文本的剪枝作用.

全局索引构建. 使用在局部索引构建中收集的统计信息, 可以在主节点中进一步构建全局索引.

3 局部索引构建

为了更高效地支持各个分区的查询加速, 本文采用了 RI⁺索引结构^[25]. 下面, 首先介绍对于空间文本查询问题的两个重要观察, 然后进一步介绍 RI(Refined Index)^[25]结构. 最后, 将结合低频词汇的剪枝作用对 RI⁺索引的细节进行介绍.

3.1 RI 结构

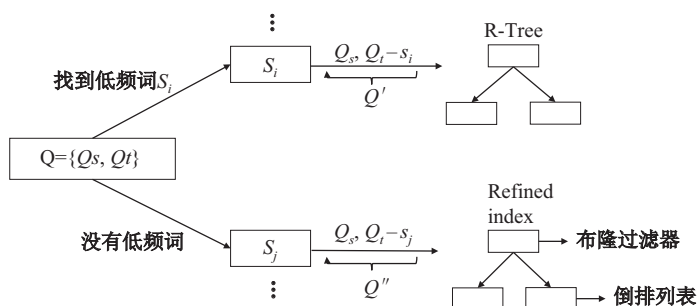
IR-Tree^[1]结构是处理布尔范围查询的经典结构. IR-Tree 结构基于 R-Tree 进行扩展, 在 R-Tree 的每一层嵌入了倒排列表以对子节点的文本数据进行索引. 这样, 在非叶子节点搜索时, 通过匹配当前的查询字符串在倒排列表中的映射节点可以快速找到满足文本过滤条件的子节点, 在空间过滤的同时结合了文本的剪枝作用, 极大地减小了搜索空间. 然而, 对于 IR-Tree 有两个重要的观察. ① IR-Tree 在每一层都嵌入了倒排列表结构, 因此, 每个字符串会被冗余地显式存储, 并导致巨大的空间开销; ② IR-Tree 在每一层的搜索都进行了准确的匹配, 这并不是必要的.

基于这两个观察, 可以对 IR-Tree 进行以下优化. ① 在非叶子节点, 不再基于倒排列表进行文本剪枝, 而是用布隆过滤器对子节点的文本信息进行抽象. 由于布隆过滤器没有显式地存储字符串, 可以达到更小的空间开销. ② 在叶子节点, 依然基于倒排列表进行准确匹配, 以保证能够找出所有满足空间和文本要求的空间文本对象并避免误报的发生.

3.2 RI⁺结构

上面提到的 IR-Tree 结构和 RI 结构都是基于空间索引结构的扩展结构, 然而往往都忽视了低频词汇在文本过滤中的剪枝作用. 由于低频词汇只被少数空间文本对象拥有, 通过基于文本索引(如倒排列表)的扩展结构对数据进行索引时, 可以通过低频词汇快速缩小搜索空间. 基于此, 这里采用了 RI⁺索引结构(见图 2). RI⁺结构基于倒排列表结构对空间文本数据集进行组织, 对于低频字符串, 将包含该字符串的对象组织为 R-Tree 结构; 对于非低频词汇, 将包含该字符串的对象组织为上述的 RI 结构.

对于布尔范围查询, 首先在查询字符串集合中找到低频词汇(在构建 RI⁺索引结构时同时维护了一个布隆过滤器以保存所有的低频词汇). 在找到低频词汇后, 将该低频词汇映射到对应的 R-Tree 结构. 由于低频词汇对应的对象数目较少, 这里只需要使用该 R-Tree 结构进行空间过滤即可; 然后, 在叶子节点进行最后的文本过滤以找出最终解. 如果没有找到低频词汇, 则直接选择查询字符串集合的第一个字符串 s_j 并映射到其对应的 RI 结构. 考虑到非低频词汇对应的空间文本对象数目较多, 基于 R-Tree 进行索引只能利用空间的剪枝作用, 无法高效地处理空间文本查询; 因此, 这里为了进一步结合文本的剪枝作用并且达到时间和空间占用上的平衡, 通过 RI 结构对非低频词汇对应的空间文本对象进行组织(注意到, RI 结构基于布隆过滤器进行优化, 相比 IR-Tree 等结构有着更低的空间占用). 最后, 通过非低频词汇 s_j 对应的 RI 结构处理新的空间文本查询 $Q'' = \{Q_s = (\tau, \epsilon), Q_t - s_j\}$.

图2 RI⁺索引结构的工作流Fig. 2 Workflow of the RI⁺ index

4 全局索引构建

全局索引并不总是有益的. 一方面, 使我们能够访问较少的分区, 从而节省更多的 CPU 资源, 降低网络成本; 但另一方面, 会导致额外的空间开销和时间开销. 为了充分利用文本的剪枝作用, 本文采用了 BFR 全局索引结构^[25]以应用于海量空间文本数据的场景.

BFR 索引是基于 R-Tree 结构的多叉树结构, BFR 索引的非叶子节点和 R-Tree 的非叶子节点一样, 只保存了空间边界信息. 其叶子节点的每个记录存储了一个数据分区的 id、空间边界信息及布隆过滤器. 分区空间边界信息由该分区所有点坐标信息获得; 这里数据分区对应的布隆过滤器存储了该分区下每个空间文本对象包含的字符串. 由于没有显式地存储字符串数据, 通过布隆过滤器存储文本数据的方案可以达到更低的空间占用. 同时, 通过布隆过滤器可以对相应的空间文本查询进行文本维度的过滤.

BFR 索引的查询算法. 给定布尔范围查询, 可以基于 BFR 索引进行搜索并返回符合查询的候选分区. 其中, 在非叶子节点过滤出与查询中心的距离在空间阈值范围内的子节点; 在叶子节点, 除了空间过滤外, 进一步检查查询的所有字符串是否都存在于分区对应的布隆过滤器中. 如果都存在则满足文本过滤条件, 该分区成为候选分区以进行进一步处理; 否则被丢弃.

5 分布式的空间文本查询算法实现

本节将针对研究的两种空间文本查询问题进行依次探究. 首先, 将基于上述的框架和索引结构对布尔范围查询的分布式算法进行探究; 然后, 将对空间文本相似连接进行更为深入的探索.

5.1 布尔范围查询

基于上述的两层索引框架, 布尔范围查询的分布式处理算法如下.

全局过滤. 首先, 基于全局索引选出候选分区. 具体来讲, 通过 BFR 索引结构找出所有空间边界与给定的圆形查询范围有交叉且其布隆过滤器包含所有的查询字符串的分区, 并对这些分区进行局部过滤.

局部过滤. 针对全局过滤后得到的每个分区, 基于每个分区建立的 vRI⁺v 索引结构进行布尔范围查询; 最后, 将每个候选分区处理后的结果汇集到全局节点.

5.2 空间文本相似连接

作为当下主流的分布式空间数据分析框架, 先前的工作^[4]提出了两种基于距离的空间

相似连接方案. 下面, 对这两种方案进行回顾, 并基于这两种方案进行扩展提出 6 种分布式下的空间文本相似连接方案.

5.2.1 分布式下的空间相似连接

给定空间数据集 \mathcal{R} 和 \mathcal{S} , 基于距离的空间相似连接将找出所有的 (r, s) , 满足 $r \in \mathcal{R}, s \in \mathcal{S}$ 且 $\text{dist}(r, s)$ 小于等于给定的距离阈值 ϵ . 先前的工作^[4]主要提出了两种分布式下基于距离的空间相似连接方案, 其细节如下.

(1) 首先, 基于 Sort-tiler-recursive^[26] 算法对于 \mathcal{R} 数据集进行分区, 收集其各个分区空间边界信息(MBR)并以这些分区的边界信息构建 \mathcal{R} 的 R-Tree 结构. 然后, 将这个 R-Tree 广播到各个节点. 基于这个对 \mathcal{R} 的分区边界信息构建的 R-Tree, 对 \mathcal{S} 中的每个点 s_i 求出与 s_i 距离在距离阈值 ϵ 内的 \mathcal{R} 数据集分区列表. 例如, 假设 \mathcal{R} 的分区边界中, 与点 s_i 的距离小于等于 ϵ 的分区边界对应的分区 id 列表为 $\{1, 3, 4, 8\}$, 可以将点 s_i 冗余映射为 $\{(s_i, 1), (s_i, 3), (s_i, 4), (s_i, 8)\}$. 最后, 对 \mathcal{S} 冗余映射后的数据集 \mathcal{S}' , 按每个数据的映射的分区号进行重新分区. 这样, \mathcal{R} 和 \mathcal{S}' 的分区数目相同, 且 \mathcal{R} 的任一分区(不妨设为第 i 个分区)只能与 \mathcal{S}' 的对应分区(第 i 个分区)进行连接操作.

(2) 首先, 对 \mathcal{R} 和 \mathcal{S} 分别基于 Sort-tiler-recursive 算法进行分区, 并对两个数据集分别收集各个分区空间边界信息; 显然对于 \mathcal{R} 和 \mathcal{S} 的任一分区对 $(\mathcal{R}_i, \mathcal{S}_j)$, 当前仅当两个分区的最小距离 $\text{mindist}(\mathcal{R}_i, \mathcal{S}_j) \leq \epsilon$ 时, 其可能存在满足连接条件的点对. 因此, 可以分别对这两个数据集构建基于分区边界信息的 R-Tree 结构, 并对这两个 R-Tree 进行连接操作找出所有满足 $\text{mindist}(\mathcal{R}_i, \mathcal{S}_j) \leq \epsilon$ 的分区对, 最后对这些分区对进行局部的连接操作.

5.2.2 分布式下的空间文本相似连接

上述两种方案找出了满足连接条件的局部分区对. 对于每个局部分区对, 文章进一步探索了相应的空间文本相似连接算法. 本文在第 1.3 节中回顾了前缀过滤技术, 前缀过滤作为文本相似连接当中广泛应用的技术, 能够大大减少需要扫描的字符串数目. 对于需要连接的两个空间文本数据分区 \mathcal{R}_i 和 \mathcal{S}_j , 最直接的想法是基于传统的文本相似连接算法进行扩展. 对于 \mathcal{R}_i 分区里的每个空间文本对象 $o = (\rho, \gamma)$, 首先对其包含的字符串集合根据给定的 Jaccard 阈值 λ 生成长度为 $|\gamma| - \lceil \lambda \times |\gamma| \rceil + 1$ 的前缀; 然后, 基于倒排列表结构将前缀中的各个字符串映射到 o . 依此对 \mathcal{R}_i 分区的所有数据进行索引后, 对于 \mathcal{S}_j 中的每个空间文本对象, 同样生成其文本前缀, 并基于前面生成的倒排列表结构依次扫描当前前缀当中的每个字符串. 对于映射的每个 \mathcal{R}_i 中的对象, 如果与当前 \mathcal{S}_j 的空间文本对象空间距离在给定范围内的话, 则加其为候选解. 最后, 对所有的候选解进行验证, 检查各个候选解与当前空间文本对象的文本 Jaccard 相似度是否大于等于 λ .

注意到上面的方案是先利用文本剪枝作用再利用空间距离进行过滤, 另外一种直接的思路是先进行空间过滤再进行文本剪枝. 类似第 5.2.1 节提到的第二种方案, 可以先对 \mathcal{R}_i 和 \mathcal{S}_j 分别构建 R-Tree 结构, 然后将两个 R-Tree 进行连接, 最后找出所有满足空间连接条件的叶子节点对; 对于叶子节点对, 可以依照上面提到的文本连接算法进行进一步处理. 注意由于 \mathcal{R}_i 对应的 R-Tree 中每个叶子节点可能和 \mathcal{S}_j 的 R-Tree 的多个叶子节点进行连接, 这样每次连接的时候都需要对其叶子节点构建一次基于文本前缀的倒排列表结构, 导致了额外的开销. 因此, 这里需要在构建 \mathcal{R}_i 的 R-Tree 时, 在其叶子节点同时嵌入该叶子节点包含的空间文本对象对应的基于前缀的倒排列表结构, 以避免多次创建该倒排列表的额外开销.

上述两个方案的缺陷是只利用了一个维度的剪枝能力. 比如上述的第二个方案, 首先基

于空间过滤再进行文本剪枝;当给定的空间距离阈值很大且文本相似度阈值也很大时,由于此时空间的剪枝能力几乎失效,该方案会产生大量的叶子节点对.然而,根据文本相似度限制,此时真正满足连接条件的数据对应该较少.显然,在这种场景下第二个方案的效率大大降低.同样,在空间距离阈值很小且文本相似度也很小的场景下,基于文本优先的方案也会产生效率问题.回顾在研究布尔范围查询时,文章采用了 RI 结构,其同时利用了空间和文本的剪枝能力. RI 结构同样可以应用于空间文本相似连接问题,在该问题背景下,本文对 RI 结构进行修改并提出了 Prefix-RI 结构.具体来讲,Prefix-RI 结构相对于 RI 结构的修改如下.

(1) 叶子节点不需要对其包含的所有字符串进行索引,而是对每个空间文本对象的文本前缀中的字符串进行索引.

(2) 在非叶子节点的布隆过滤器中,只存储子树中包含的空间文本对象的文本前缀中的字符串.

利用在 \mathcal{R}_i 上建立的 Prefix-RI 索引,可以对 \mathcal{S}_j 中的每一条记录在 Prefix-RI 索引上查找与其相似的对象集合.其算法细节如算法 1 所示.

算法 1: STJoin

输入: 空间距离阈值 ϵ , 文本 Jaccard 相似度阈值 λ , 数据分区 \mathcal{R}_i 对应的 Prefix-RI 索引根节点 r , 数据分区 \mathcal{S}_j 中的一个空间文本对象 $o_s = (\rho_s, \gamma_s)$

输出: \mathcal{R}_i 中可以和 o_s 进行连接的空间文本对象集合

```

1:  $prefixLen = |\gamma_s| - \lceil \lambda \times |\gamma_s| \rceil + 1$ ;
2:  $S = \emptyset, R = \emptyset$ ;
3:  $S.push(r)$ ;
4: while  $S \neq \emptyset$  do
5:    $n = S.pop()$ ;
6:   if  $n$  is a non-leaf node then
7:      $bloomFilter = n.bloomFilter$ ;
8:      $flag = \text{False}$ ;
9:     for  $i$  from 0 to  $prefixLen - 1$  do
10:      if  $bloomFilter$  contains  $\gamma_s[i]$  then
11:         $flag = \text{True}$ ;
12:        break;
13:     if  $flag == \text{True}$  then
14:       for each entry  $e_i$  of  $n$  do
15:         if  $dist(e_i.mbr, \rho_s) \leq \epsilon$  then
16:            $S.push(e_i.node)$ ;
17: else
18:    $inverted = n.invertedList$ ;
19:    $candidates = \emptyset$ ;
20:   for  $i$  from 0 to  $prefixLen - 1$  do
21:     foreach object  $o$  in  $inverted.get(\gamma_s[i])$  do

```

```

22:         if  $\text{dist}(\rho_s, o.\rho) \leq \epsilon$  and  $\lambda \times |o.\gamma| \leq |\gamma_s|$  and  $\lambda \times |\gamma_s| \leq |o.\gamma|$  then
23:             candidates.add(o);
24:         R++=verify(candidates);
25: return R;

```

在阐述该算法前, 先介绍 Prefix-RI 结构的一个重要性质.

性质 5.1 给定基于空间文本数据集 \mathcal{R} 建立的 Prefix-RI 索引, 对于 \mathcal{S} 中的任意一个空间文本对象 $o_s = (\rho_s, \gamma_s)$, 若该索引的非叶子节点 \mathcal{N} 对应的子树包含与 o_s 相似的对象, 则对于 o_s 的文本前缀, 其至少有一个字符串存在于 \mathcal{N} 对应的布隆过滤器中.

证明 不失一般性的, 假设非叶子节点 \mathcal{N} 对应的子树下存在一个与 o_s 相似的空间文本对象 o_r . 由前缀过滤技术的性质可知, o_s 和 o_r 的文本前缀至少存在一个相同的字符串 ζ . 同时, 根据前面所述的 Prefix-RI 索引的结构特点, 非叶子节点 \mathcal{N} 对应的布隆过滤器中存储了该子树下包含的所有空间文本对象的文本前缀; 因此, 可知 ζ 必然存在于 \mathcal{N} 对应的布隆过滤器中. 综上, 性质得证.

接下来对基于 Prefix-RI 索引结构的搜索算法进行详细介绍. 如算法 1 所示, 对于 \mathcal{S}_j 中的一个空间文本对象 $o_s = (\rho_s, \gamma_s)$, 首先根据文本相似度阈值 λ 计算其前缀长度(行 1). 由于 Prefix-RI 索引为树形结构, 需要从其根节点开始扫描并从上至下递归到叶子节点(行 3—5). 当扫描到非叶子节点时(行 6), 首先检查 o_s 的文本前缀当中是否至少有一个字符串存在于该节点对应的布隆过滤器(行 7—13); 当该非叶子节点满足上述条件时, 则对该节点中满足空间范围要求的子节点进行递归搜索(行 14—16). 最后, 当扫描到叶子节点时(行 17), 将 o_s 的文本前缀当中的各个字符串都映射到包含其的对象列表(行 20—21), 并进行长度过滤和空间过滤以进一步找到候选对象(行 22—23), 并对这些候选对象进行最后的验证(行 24).

可以看出, 上述的第三种方案, 通过结合空间和文本两个维度的剪枝作用, 可以更好地应对前面两种方案的失效场景. 最后, 结合第 5.2.1 节中提到的两种分布式空间相似连接方案, 可以和上述 3 种方案进行进一步的组合. 具体来讲, 首先通过这两种空间连接方案产生需要进行空间文本连接的局部分区对; 然后, 对于每个局部分区对, 采用本文介绍的 3 种空间文本数据连接方案进行进一步处理. 根据这样的组合, 总共有 6 种分布式的空间文本相似连接方案: Spatial-first, Textual-first, Hybrid, Allpairs-Spatial-first, Allpairs-Textual-first 及 Allpairs-Hybrid. 表 1 对这 6 种方案进行了详细介绍.

6 实验探究

6.1 实验设置

为了验证所提出算法的实际运行效果, 本文基于 Spark SQL 实现了相应的索引技术和 2 种空间文本查询操作. 此外, 本文采用了由 9 个节点组成的集群进行实验探究, 由于实验设备的采购时间不一致, 主要包括以下 3 种不同配置的机器: ① 1 台具有 24 核 Intel Xeon E5-2620 2.00 GHz 处理器和 192 GB RAM 的机器; ② 5 台配备 6 核 Intel Xeon E5-2603 v3 1.60 GHz 处理器和 20 GB RAM 的机器; ③ 3 台配备 6 核 Intel Xeon E5-2609 1.90 GHz 处理器和 16 GB RAM 的机器. 选择一台类型 ① 的机器作为主节点, 其他机器都作为从节点; 另外, 每个从节点使用 15 GB 内存和所有可用的 6 个 CPU 进行后续计算. 所有节点都运行在 Ubuntu 14.04.2 LTS 系统下, 且各个节点都安装了 Hadoop 2.6.1 和 Spark 1.6.3 框架.

表 1 分布式的空间文本相似连接方案描述

Tab. 1 Distributed spatio-textual similarity join algorithms	
方案简称	方案描述
Spatial-first	基于 5.2.1 小节中的第一种分布式空间连接方案产生局部分区对. 然后, 对每个局部分区对, 利用空间优先的空间文本连接方案对其进行处理
Textual-first	基于 5.2.1 小节中的第一种分布式空间连接方案产生局部分区对. 然后, 对每个局部分区对, 利用文本优先的空间文本连接方案对其进行处理
Hybrid	基于 5.2.1 小节中的第一种分布式空间连接方案产生局部分区对. 然后, 对每个局部分区对, 利用基于 Prefix-RI 索引的空间文本连接方案对其进行处理
Allpairs-Spatial-first	基于 5.2.1 小节中的第二种分布式空间连接方案产生局部分区对. 然后, 对每个局部分区对, 利用空间优先的空间文本连接方案对其进行处理
Allpairs-Textual-first	基于 5.2.1 小节中的第二种分布式空间连接方案产生局部分区对. 然后, 对每个局部分区对, 利用文本优先的空间文本连接方案对其进行处理
Allpairs-Hybrid	基于 5.2.1 小节中的第二种分布式空间连接方案产生局部分区对. 然后, 对每个局部分区对, 利用基于 Prefix-RI 索引的空间文本连接方案对其进行处理

本文使用了两个真实的海量数据集. ① OSM 数据集, 其包含 3 000 万条记录. 数据集的空间部分来自于 OpenStreetMap 项目^[27], 每个点由二维坐标表示; 另外, 每一个点关联了从 SNAP^[28] (斯坦福网络数据集项目)收集的文本数据集中提取的字符串集合. ② TX-CA 数据集^[27]. 该数据集中的点坐标代表了美国德克萨斯州和加利福尼亚州的真实道路网络和街道. 此外, 将每个点的所在州、县和城镇名称相结合作为该点相关联的字符串集合. 该数据集总共拥有 2 600 万条记录, 包括TX数据集共 1 400 万条, CA 数据集共 1 200 万条.

基于以前的工作^[4-5], 本文主要关注吞吐量(每分钟处理的请求数)和时间延迟两个性能指标. 对于布尔范围查询, 开启了 10 个线程不断地发出查询来模拟用户并发请求; 总共进行 500 次查询以进一步计算性能指标. 为了生成测试集, 首先从原始数据集中随机选择一条记录, 并使用其点坐标作为查询位置; 然后, 为了进一步了解字符串数目对查询性能的影响, 还从该记录中随机选择 1 到 4 个字符串作为查询字符串集合. 此外, 本文还尝试探索查询区域对性能的影响; 这里, 查询区域定义为查询半径与数据集边界的长和宽较小值的百分比, 默认值为 1%. 在实验中, 查询区域比例在 {1%, 5%, 10%, 15%, 20%} 之间变化. 对于空间文本相似连接, 其空间距离阈值变化范围为 {0.01, 0.02, 0.03, 0.04, 0.05} (近似对应了 1% 至 5% 的查询区域); 其文本相似度阈值变化范围为 {0.6, 0.7, 0.8, 0.9}, 这里对文本相似度阈值的选择有以下考虑. ① 许多与文本相似连接相关的工作都将文本相似度阈值限定在 0.5 以上^[23,29], 并且较小的文本相似度阈值没有实际意义; ② 当前的取值范围能够很好地覆盖实际应用场景. 比如, 社交推荐等应用通常指定文本相似度阈值在 0.6 至 0.7 范围内变化; 文本去重等应用通常指定文本相似度阈值在 0.8 至 0.9 范围内变化. 在所有实验中, HDFS 块大小为 64 MB; 由于集群总共有 72 个 CPU 核可用, 本文将数据集默认分为 144 个分区.

由于当下缺乏直接支持上述 2 种空间文本查询的分布式系统, 在实验中, 对于布尔范围查询的实验对比, 论文采取了以下 3 种方案. ① 基于当下主流的空间数据分布式分析系统进行扩展. 由于这些系统大多基于 R-Tree 进行索引, 可以通过在 R-Tree 的叶子节点层加入倒排列表结构使这些平台能够处理布尔范围查询. 这里由于文献 [4] 的大量实验对比证明了其杰出的性能, 本文基于文献 [4] 提出的系统进行了扩展, 此方案在实验对比中简称为 Baseline1. ② 基于 IR-Tree 实现的分布式空间文本分析方案. IR-Tree 主要用于局部索引的构建, 该方案依然使用了 BFR 索引作为全局索引技术, 此方案在实验对比中简称为 Baseline2. ③ 基于 RI⁺索引实现的方案, 在实验中简称为 RI⁺. 对于空间文本相似连接对提出的 6 种方案(Spatial-first, Textual-first, Hybrid, Allpairs-Spatial-first, Allpairs-Textual-first 及 Allpairs-Hybrid)进行了实验对比.

6.2 实验结果分析

索引空间开销. 针对 OSM 数据集, 统计了两层索引的维护开销(在 TX-CA 数据集上的结果与此相似). 首先, 针对 3 000 万条空间文本数据进行索引时, 采用的全局索引的空间占用仅为 49.57 MB. 通常, 维护全局索引的 Master 节点的物理内存可达到上百 GB, 因此, 所采用的全局索引也能很好地应对更大的数据集带来的挑战. 对于局部索引, IR-Tree 的空间占用为 59.8 GB 而 RI^+ 的空间占用为 40.4 GB. 可以发现, 采用的两层索引框架在空间占用方面具有较好的可扩展性.

查询字符串数目的影响. 如图 3 所示为查询字符串数目对布尔范围查询性能的影响. 从图中可以看到, 这 3 种方案在查询字符串数目增大的时候性能都有提升. 这是由于只有 1 个查询字符串的时候, 通常会有较多符合查询条件的解; 一方面查询命中的分区会更多, 另外在每个分区下的处理时间也更大. 当查询字符串数目增大时, 满足条件的解数目大大减少, 带来了更强的剪枝能力. 另外, 当查询字符串数目在 2 到 4 之间变化时, 此时性能变化不大; 这说明此时解集数目基本没有太大变化. 注意到在有多查询字符串时, 本文采用的索引技术由于更好地利用了低频词汇的剪枝能力, 取得了更好的性能.

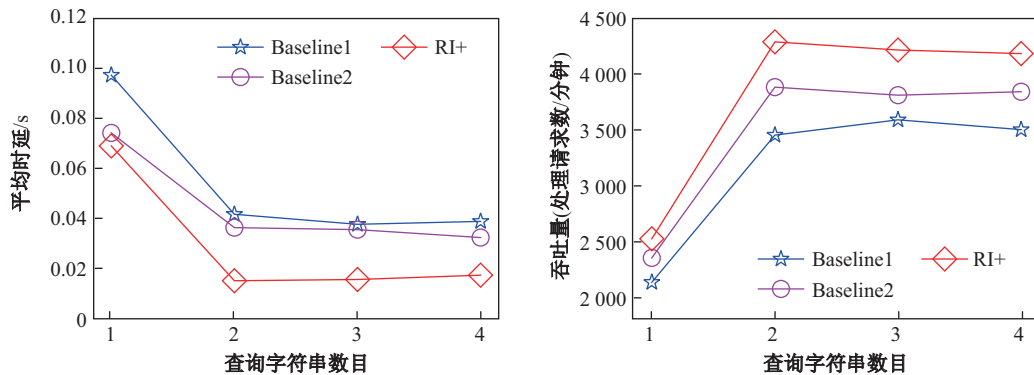


图3 查询字符串数目的影响(OSM 数据集)

Fig. 3 Effect of query keywords (OSM Dataset)

查询范围的影响. 如图 4 所示, 当查询范围从 1% 增大到 20% 时, 3 种方案都有性能的下降. 一方面, 查询范围越大, 该查询在全局索引过滤时命中的分区个数越多; 另外一方面, 局部索引的空间剪枝能力也在变小. 但是, 可以观察到, 由于 Baseline1 没有很好地利用文本的过滤能力, 在查询范围变大时性能下降明显要比其他两个更快; 另外由于 RI^+ 索引技术更好地利用了低频词汇的剪枝能力, 其性能依然比 Baseline2 要更好, 达到了更好的稳定性.

数据集大小的影响. 如图 5 所示为数据集大小对布尔范围查询的影响. 当数据规模从 1 千万增长到 6 千万 ($2 \times$ OSM 数据集) 时, 从实验结果来看, 文章提出的方案展现了更好的扩展性; 当数据规模增大时, 文章提出的方案性能下降趋势更慢.

全局索引的有效性验证. 为了验证文章采用的全局索引的有效性, 通过图 6 展示了使用 R-Tree 作为全局索引和使用 BFR 索引作为全局索引这两种方案的性能对比. 基于前面的实验结果, 这里局部索引直接采用了 RI^+ 索引结构. 可以看到, 相对于只利用空间剪枝的全局索引, BFR 索引技术达到了更好的性能.

文本相似度阈值的影响. 如图 7 所示为文本相似度阈值对空间文本相似连接的性能影响. 随着文本相似度阈值的增大, 6 种方案的耗时都在下降. 显然文本相似度阈值越大, 根据前缀过滤技术的原理, 此时需要扫描的字符串数目越小; 因此, 需要验证的空间文本对象对数目也在下降. 注意到, 在这 6 种方案中, 由于 Hybrid 方案同时结合了空间和文本的剪枝能力, 在

实验中取得了最好的性能. 相比之下, 由于 Allpairs-Hybrid 引入的分区对数目较多, 因此相比 Hybrid 的额外开销也会更大. 另外, 实验发现基于文本优先的连接方案取得了最差的性能(包括 Textual-First 和 Allpairs-Textual-First 两个方案).

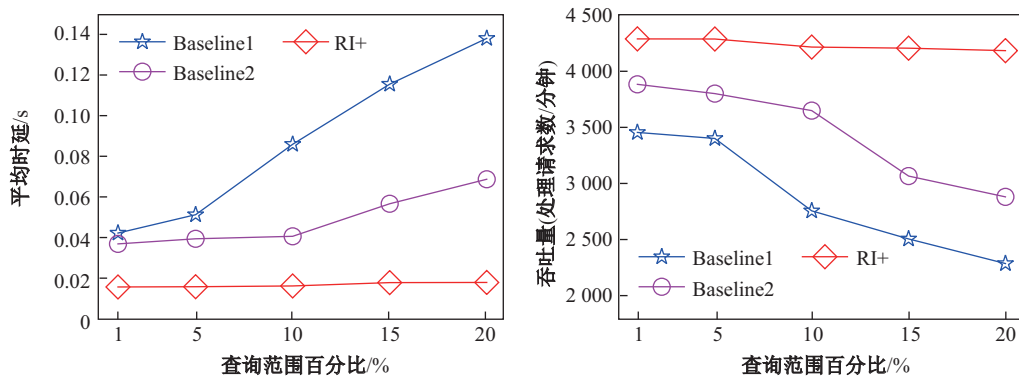


图4 查询范围的影响(TX-CA 数据集)

Fig.4 Effect of the query area (TX-CA Dataset)

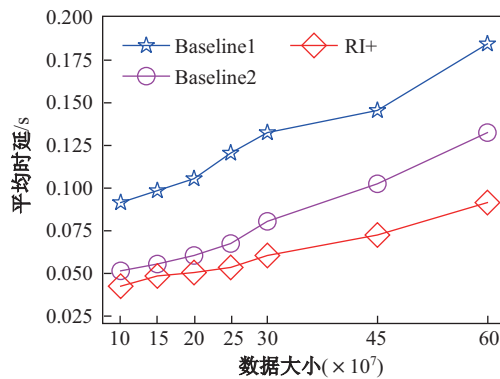


图5 数据集大小的影响(OSM 数据集)

Fig.5 Effect of data size (OSM Dataset)

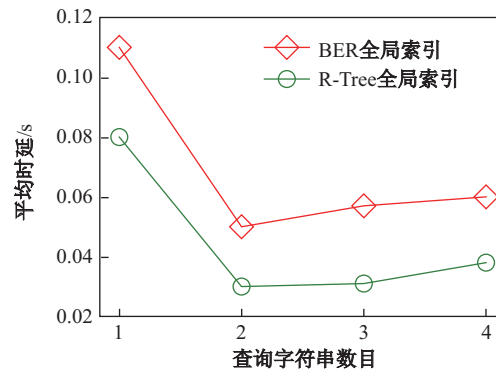


图6 全局索引的有效性(TX-CA 数据集)

Fig.6 Effect of the global index (TX-CA Dataset)

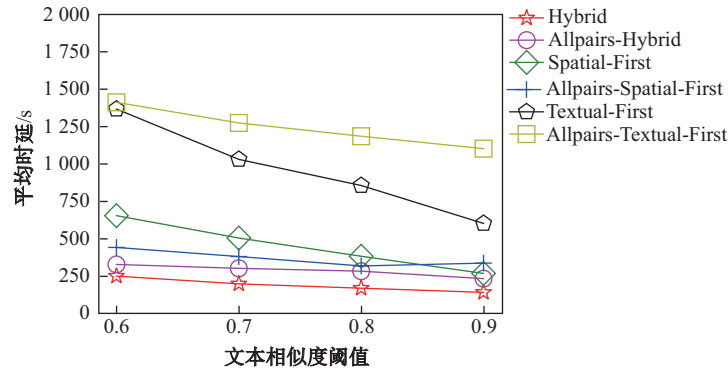


图 7 文本相似度阈值的影响(OSM 数据集)

Fig. 7 Effect of textual similarity thresh-old (OSM Dataset)

空间距离阈值的影响. 如图 8 所示, 随着空间距离阈值的增大, 所有方案的耗时都在增大. 但对比可以发现, Hybrid 方案消耗的时间最少且增长趋势更为缓慢; 这主要由于其结合了文本的剪枝作用, 所以, 对于查询空间范围的增大更为不敏感, 在实验中展现了更好的适应性.

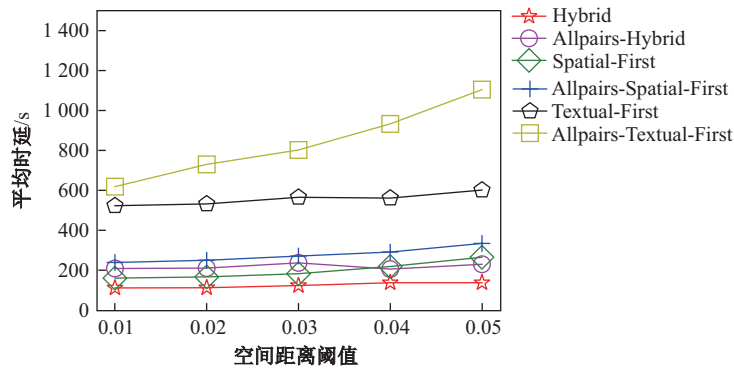


图 8 空间距离阈值的影响(OSM 数据集)

Fig. 8 Effect of spatial distance thresh-old (OSM Dataset)

7 总 结

本文采用了分布式环境下的两层索引框架(包括全局索引和局部索引), 基于该框架本文提出了分布式的处理算法以解决空间文本相似连接等查询问题. 同时, 本文基于 Spark 平台实现了所提出的分布式算法, 并在两个真实的数据集下进行了充分的实验对比, 实验结果验证了本文提出的算法的有效性. 随着越来越多的工作致力于解决路网场景下的查询问题^[30-34], 在将来的工作中, 我们将进一步研究路网下的空间文本查询问题.

[参 考 文 献]

- [1] CHEN L, CONG G, JENSEN C S, et al. Spatial keyword query processing: An experimental evaluation[C]// International Conference on Very Large Data Bases. Trondheim, Norway: VLDB Endowment, 2013, 6(3): 217-228.
- [2] BOUROS P, GE S, MAMOULIS N. Spatio-textual similarity joins[J]. Proceedings of the VLDB Endowment, 2012, 6(1): 1-12.
- [3] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]// USENIX Association Usenix Conference on Networked Systems Design and Implementation. New York: ACM, 2012, 70(2): 2.

- [4] XIE D, LI F, YAO B, et al. Simba: Efficient in-memory spatial analytics[C]// International Conference on Management of Data. New York: ACM, 2016: 1071-1085.
- [5] ELDAWY A, MOKBEL M F. Spatial Hadoop: A MapReduce framework for spatial data[C]//International Conference on Data Engineering. New York: IEEE, 2016: 1352-1363.
- [6] XIE D, LI F, YAO B, et al. Simba: Spatial in-memory big data analysis[C]//ACM Sigspatial International Conference on Advances in Geographic Information Systems. New York: ACM, 2016: 86-89.
- [7] YAO B, ZHANG W, WANG Z J, et al. Distributed in-memory analytics for big temporal data[C]//International Conference on Database Systems for Advanced Applications. Berlin: Springer, 2018: 549-565.
- [8] AJI A, WANG F, VO H, et al. Hadoop-GIS: A high performance spatial data warehousing system over MapReduce[J]. Proceedings of the VLDB Endowment, 2013, 6(11): 1009-1020.
- [9] YAO B, LI F, HADJIELEFTHARIOU M, et al. Approximate string search in spatial databases[C]//International Conference on Data Engineering. New York: IEEE, 2010: 545-556.
- [10] YAO B, TANG M, LI F. Multi-approximate-keyword routing in GIS data[C]//ACM Sigspatial International Conference on Advances in Geographic Information Systems. New York: ACM, 2011: 201-210.
- [11] LI F, YAO B, TANG M, et al. Spatial approximate string search[J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(6): 1394-1409.
- [12] ZHOU Y, XIE X, WANG C, et al. Hybrid index structures for location-based web search[C]//International Conference on Information and Knowledge Management. New York: ACM, 2005: 155-162.
- [13] BECKMANN N, KRIEGER H P, SCHNEIDER R, et al. The R*-tree: An efficient and robust access method for points and rectangles[J]. ACM Sigmod Record, 1990, 19(2): 322-331.
- [14] GUTTMAN A. R-trees: A dynamic index structure for spatial searching[C]//International Conference on Management of Data. New York: ACM, 1984: 47-57.
- [15] WU D, JENSEN C S. A density-based approach to the retrieval of top-k spatial textual clusters[C]//International Conference on Information and Knowledge Management. New York: ACM, 2016: 2095-2100.
- [16] CHOUDHURY F M, CULPEPPER J S, SELLIS T. Batch processing of top-k spatial-textual queries[C]//International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data. New York: ACM, 2015: 7-12.
- [17] LUO C, LI J, LI G, et al. Efficient reverse spatial and textual k nearest neighbor queries on road networks[J]. Knowledge-Based Systems, 2016, 93: 121-134.
- [18] CHEN Z, CONG G, ZHANG Z, et al. Distributed publish/subscribe query processing on the spatio-textual data stream[C]//International Conference on Data Engineering. New York: IEEE, 2017: 1095-1106.
- [19] YAO B, LI F, KUMAR P. K nearest neighbor queries and knn-joins in large relational databases (almost) for free[C]//International Conference on Data Engineering. New York: IEEE, 2010: 4-15.
- [20] 徐石磊, 王雷, 胡卉芪. 基于分布式系统OceanBase的并行连接[J]. 华东师范大学学报(自然科学版), 2017(5): 1-10.
- [21] SARAWAGI S, KIRPAL A. Efficient set joins on similarity predicates[C]//International Conference on Management of Data. New York: ACM, 2004: 743-754.
- [22] CHAUDHURI S, GANTI V, KAUSHIK R. A primitive operator for similarity joins in data cleaning[C]//International Conference on Data Engineering. New York: IEEE, 2006: 5.
- [23] BAYARDO R J, MA Y, SRIKANT R. Scaling up all pairs similarity search[C]//International Conference on World Wide Web. New York: ACM, 2007: 131-140.
- [24] FAN L, CAO P, ALMEIDA J, et al. Summary cache: A scalable wide-area web cache sharing protocol[J]. IEEE/ACM Transactions on Networking, 2000, 8(3): 281-293.
- [25] XU Y, YAO B, WANG Z J, et al. Skia: Scalable and efficient in-memory analytics for big spatial-textual data[EB/OL]. (2018-05-15) [2018-06-18]. <https://www.researchgate.net/publication/326352693>.
- [26] LEUTENEGGER S T, LOPEZ M A, EDGINGTON J. STR: a simple and efficient algorithm for R-tree packing[C]//International Conference on Data Engineering. New York: IEEE, 1997: 497-506.
- [27] OpenStreetMap Foundation. Openstreetmap project[EB/OL]. (2016-02-12) [2018-06-18]. <http://www.openstreetmap.org>.
- [28] LESKOVEC J, KREYL A. SNAP Datasets: Stanford large network dataset collection[EB/OL]. (2014-06-01) [2018-06-18]. <http://snap.stanford.edu/data>.
- [29] SAHAMI M, HEILMAN T D. A web-based kernel function for measuring the similarity of short text snippets[C]//International Conference on World Wide Web. New York: ACM, 2006: 377-386.

证了数据可追溯、不可篡改.在此基础上,本文在区块链系统上构建了倒排索引,提高了查询效率,且支持复杂查询.同时,实现了基于 REST 的微服务架构,支持多方接入.由于研究时间和现有知识水平的限制,本文的研究工作还不够完善,需要进一步探索和研究,作为下一步的研究内容,将要进行的工作主要有以下几点.

(1) 目前系统吞吐量(TPS)较低,主要原因是节点共识耗时太长,下一步工作考虑优化PBFT共识算法.

(2) 支持当新节点加入区块链系统或者节点故障重启后数据快速同步.

(3) 区块链中数据为全备份,这会占用大量的存储和网络带宽.因此下一步工作将考虑在数据安全、有效的情况下让区块链系统支持分片(Sharding).

[参 考 文 献]

- [1] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system [R/OL]. [2018-06-11] <https://bitcoin.org/bitcoin.pdf>.
- [2] ETHEREUM FOUNDATION. Ethereum [EB/OL]. [2018-6-11]. <https://www.ethereum.org>.
- [3] R3CEV. R3 [EB/OL]. [2018-06-11]. <https://www.r3.com>.
- [4] LINUX FOUNDATION. Hyperledger [EB/OL]. [2018-06-11]. <https://www.hyperledger.org>.
- [5] 袁勇,王飞跃. 区块链技术发展现状与展望[J]. 自动化学报, 2016, 42(4): 481-494.
- [6] RIPPLE. Ripple [EB/OL]. [2018-06-11]. <https://ripple.com>.
- [7] GOOGLE. LevelDB [EB/OL]. [2018-06-11]. <http://leveldb.org>.
- [8] APACHE. CouchDB [EB/OL]. [2018-06-11]. <http://couchdb.org>.

(责任编辑: 李 艺)

(上接第134页)

- [30] YAO B, CHEN Z, GAO X, et al. Flexible aggregate nearest neighbor queries in road networks[C]//International Conference on Data Engineering. New York: IEEE, 2018: 1-12.
- [31] MA J, YAO B, GAO X, et al. Top-k Critical Vertices Query on Shortest Path[J]. IEEE Transactions on Knowledge and Data Engineering, 2018, 99(1): 1-13.
- [32] XIE D, LI G, YAO B, et al. Practical private shortest path computation based on oblivious storage[C]//International Conference on Data Engineering. New York: IEEE, 2016: 361-372.
- [33] YAO B, XIAO X, LI F, et al. Dynamic monitoring of optimal locations in road network databases[J]. The International Journal on Very Large Data Bases, 2014, 23(5): 697-720.
- [34] XIAO X, YAO B, LI F. Optimal location queries in road network databases[C]//International Conference on Data Engineering. New York: IEEE, 2011: 804-815.

(责任编辑: 张 晶)