计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210100060

面向海量空间数据的分布式距离连接算法

王如斌[1,3]　李瑞远[2,3]　何华均[1,3]　刘　通[4]　李天瑞[1]
1 西南交通大学信息科学与技术学院　成都 611756
2 重庆大学计算机学院　重庆 400044
3 京东智能城市研究院　北京 100176
4 上海大学计算机工程与科学学院　上海 200444
(wrb@my.swjtu.edu.cn)

摘　要　空间距离连接是空间数据分析最基本的操作之一,具有广泛的应用场景。针对现有分布式方法的空间域选取过大、数据倾斜、自连接较慢的问题,提出了一种新的面向海量空间数据的分布式距离连接算法JUST-Join。首先,JUST-Join仅选取必要的空间区域作为全局域,能够提前过滤数据,减少无效的数据传输和不必要的计算开销;然后,同时考虑了参与连接的两个数据集的分布,从而缓解了数据倾斜问题;最后,针对自连接情形的冗余计算,采用平面扫描算法来进一步提高效率。文中使用Spark实现了JUST-Join算法,并利用真实的数据集做了大量实验。实验结果表明,JUST-Join算法在效率和扩展性方面都优于现有的最先进的分布式空间分析系统。

关键词:空间距离连接;空间分区;分布式计算;空间索引;时空数据
中图法分类号　TP338

Distributed Distance Join Algorithm for Massive Spatial Data

WANG Ru-bin[1,3],LI Rui-yuan[2,3],HE Hua-jun[1,3],LIU Tong[4] and LI Tian-rui[1]
1 School of Information Science and Technology,Southwest Jiaotong University,Chengdu 611756,China
2 College of Computer Science,Chongqing University,Chongqing 400044,China
3 JD Intelligent Cities Research,Beijing 100176,China
4 School of Computer Engineering and Science,Shanghai University,Shanghai 200444,China

Abstract　Spatial distance join is one of the most common operations for spatial data analysis,which has various application scenarios. Existing distributed methods face the problems of too large space,high data skew,and slow self-join. To this end,this paper proposes a novel distributed distance join algorithm,i.e.,JUST-Join,for massive spatial data. First,JUST-Join regards only the necessary space as the global domain,which can filter invalid data out,reducing the overhead of unnecessary data transmission and computation. Second,we consider both the spatial distributions of the two datasets,which relieves the data skew issue. Third,for the spatial self-join,we adopt plane sweep method to further improve the efficiency. We implement JUST-Join algorithm based on Spark,and conduct extensive experiments using real datasets. The experimental results show that JUST-Join is superior to the state-of-the-art distributed spatial analysis systems in terms both of efficiency and scalability.

Keywords　Spatial distance join,Spatial partition,Distributed computing,Spatial indexing,Spatio-temporal data

1　引言

随着全球定位系统和移动互联设备的普及,海量的空间数据也随之产生。对空间数据的距离连接是最常用的空间分析算子之一,具有广泛的应用场景,例如查找距离地铁站500 m范围内的兴趣点(Point of Interest,POI),帮助公司选址规划;寻找去过疫区的人群,帮助政府进行疫情防控[1];找出跨越河流的公路和桥梁,以排除洪水隐患等。

定义1(空间距离连接(Spatial Distance Join))　给定空间距离δ,以及两个空间对象集合R和S,空间距离连接要找出所有二元组(r,s),满足$r\in R,s\in S$,且空间对象r和s之间的距离$d(r,s)$小于等于δ。其形式化定义如式(1)所示:

$$R\bowtie_{\delta}S=\{(r,s),r\in R,s\in S,d(r,s)\leqslant\delta\} \tag{1}$$

空间对象r和s可以是点(point)对象,也可以是线(linestring)或多边形(polygon)等对象。假设空间点p的坐标为$\langle x^p,y^p\rangle$,$p\in r$表示p是空间对象r的其中一个空间点,那么

到稿日期:2021-01-07　返修日期:2021-05-12
基金项目:国家重点研发计划(2019YFB2101801)
This work was supported by the National Key Research and Development Program of China(2019YFB2101801).
通信作者:李瑞远(liruiyuan@whu.edu.cn)

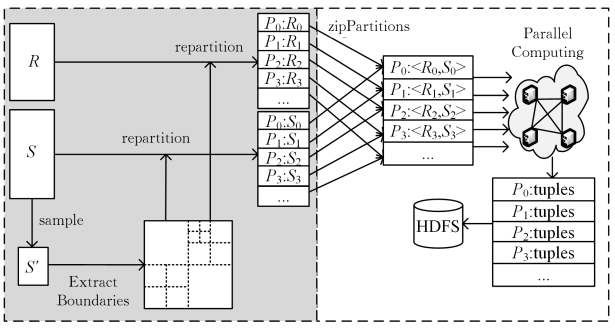
r 和 s 的距离定义为所有 $p\in r, q\in s$ 的最小欧氏距离，即：

$$d(r,s)=\min_{p\in r,q\in s} d(p,q)$$
$$=\min_{p\in r,q\in s}\sqrt{(x^p-x^q)^2+(y^p-y^q)^2} \quad (2)$$

空间距离连接算法的研究由来已久，传统单机版的空间距离连接运算[2-3]无法支持海量空间数据，因此需要基于分布式计算框架实现大规模空间数据的距离连接运算。目前涌现出了许多优秀的支持空间距离连接运算的分布式系统，包括 SpatialHadoop[4]，Hadoop GIS[5]，GeoSpark[6]，LocationSpark[7]，SpatialSpark[8]，Simba[9]，AMDS[10]等。它们基于分布式计算框架 Hadoop[11]或者 Spark[12]，实现了丰富的空间查询操作，为空间距离连接运算也提供了不同程度的支持，但在空间数据类型以及空间连接种类的支持上还不够完善。文献[1-14]实现了两种基于 Spark 的空间连接算法，丰富了空间连接的功能。这些系统对空间距离连接的实现方法大同小异，总体可概括为两个步骤：空间分区和并行计算。

空间分区过程如图 1(a)所示。首先计算数据集 S 的空间范围作为全局空间域 G，然后从数据集 S 中以采样率 η 抽取样本集 S'，并利用 Quad-tree[15]、R-tree[16]等空间索引算法对全局空间域 G 进行划分，生成多个小范围子空间的边界，尽可能地保证每个子空间内的样本数量相同，以实现负载均衡。最后按照得到的子空间边界分别对数据集 R 和 S 中的空间对象进行重新划分，每个子空间内的数据组成一个分区(partition)。

并行计算过程如图 1(b)所示。将两个数据集合并成一个数据集，其中同一子空间内来自 R 和 S 的数据组成一个新的分区，然后利用传统单机版的空间距离连接算法，对每个分区内的空间数据执行空间距离连接运算(称为子任务)，生成空间对象二元组集合，最后将所有子任务的运算结果去重之后输出到分布式文件系统中。



(a)空间分区阶段 (b)并行计算阶段

图 1 现有分布式空间距离的连接过程

Fig. 1 Existing distributed spatial distance join process

上述分布式空间连接运算存在以下 3 点不足：

(1)空间域选取过大。在空间分区阶段，仅用数据集 S 的空间范围代表 R 和 S 两个数据集的空间范围，而不是 R 和 S 空间范围的相交部分。这样会因全局空间域选取得过大而不能提前过滤掉对最终结果没有贡献的无效数据，造成网络带宽和计算存储资源的浪费，最终降低了空间距离的连接性能。

(2)数据倾斜。样本集 S' 的空间分布虽然能近似代表数据集 S 的空间分布，但不能代表数据集 R 的空间分布。仅仅以 S' 的空间分布作为划分子空间的依据，可能会导致不同子空间内来自数据集 R 的数据量相差很大。这种数据倾斜现象会导致部分分区过大，其对应的子任务的执行时间过长，进而拖慢整个并行计算过程。当倾斜非常严重时，可能会导致任务执行失败。

(3)对空间自连接未作优化。当 $R=S$ 时，称其为空间自连接运算，例如，给定一张居住地点表，找到表中每个人的邻居关系，并将其作为 LBS(Location Based Service)推荐系统的输入。现有系统均未对空间自连接做优化，导致空间自连接运算较慢。

针对现有技术存在的不足，本文设计并实现了一种面向海量空间数据的分布式距离连接算法。首先，在空间划分阶段，同时考虑了 R 和 S 的空间范围，减少了并行计算阶段无效的数据传输和不必要的计算存储开销；然后，该算法同时考虑了 R 和 S 两个数据集的空间分布，减小了数据倾斜，大幅提高了分布式系统的负载均衡能力和执行效率；最后，针对空间自连接的情况，本文提出了一种优化方法，能够加快空间自连接运算。本文基于分布式平台 Spark 实现了所提算法，并利用真实的数据做了大量实验，实验结果表明本文方法在空间距离连接的效率和扩展性方面均优于现有方法。该工作是 JUST 项目[17-18]的一部分，因此本文方法称为 JUST-Join。

2 分布式空间距离连接 JUST-Join

在介绍具体方法之前，先给出几个关键定义。

定义 2(最小边界矩形(Minimum Bounding Rectangle, MBR)) 空间对象 r 的最小边界矩形 $r.mbr=\langle x_{\min}, y_{\min}, x_{\max}, y_{\max}\rangle$ 定义为平行于坐标轴且包含 r 中所有点的最小矩形，其中 $\langle x_{\min}, y_{\min}\rangle$ 和 $\langle x_{\max}, y_{\max}\rangle$ 分别表示矩形 $r.mbr$ 左下角和右上角的坐标，且 $x_{\min}=\min_{p\in r} x^p$，$y_{\min}=\min_{p\in r} y^p$，$x_{\max}=\max_{p\in r} x^p$，$y_{\max}=\max_{p\in r} y^p$。

空间对象集合 R 的最小边界矩形 $R.mbr=\langle x'_{\min}, y'_{\min}, x'_{\max}, y'_{\max}\rangle$ 定义为平行于坐标轴且包含 R 中所有空间对象的最小矩形，且 $x'_{\min}=\min_{r\in R} r.mbr.x_{\min}$，$y'_{\min}=\min_{r\in R} r.mbr.y_{\min}$，$x'_{\max}=\max_{r\in R} r.mbr.x_{\max}$，$y'_{\max}=\max_{r\in R} r.mbr.y_{\max}$。

定义 3(扩展最小边界矩形(Expand Minimum Bounding Rectangle, EMBR)) 空间对象 r 的最小边界矩形 $r.mbr$ 的 ε 扩展 $r.embr(\varepsilon)=\langle ex_{\min}, ey_{\min}, ex_{\max}, ey_{\max}\rangle=\langle x_{\min}-\varepsilon, y_{\min}-\varepsilon, x_{\max}+\varepsilon, y_{\max}+\varepsilon\rangle$，其中 $\langle x_{\min}, y_{\min}, x_{\max}, y_{\max}\rangle$ 是 r 的最小边界矩形。

类似地，我们可以定义空间对象集合 R 的最小边界矩形的 ε 扩展，在此不再赘述。

JUST-Join 算法使用分而治之的思想，首先将一个大范围的空间划分成多个小范围的子空间，然后对同一子空间内来自数据集 R 和 S 的空间对象并行地执行空间连接运算。在空间分区阶段，JUST-Join 精心地选择了全局空间域，并同时考虑了 R 和 S 中数据的空间分布。并行计算阶段，在每个分区中，JUST-Join 首先对来自 S 数据集的空间对象构建局部索引；然后利用构建好的局部索引，针对该分区中来自 R 数据集的每个空间对象 r，依次查找满足距离条件的空间对象 s。

2.1 空间分区

空间分区共包含3个步骤:全局域计算、子空间划分和数据重分区。

2.1.1 全局域计算

现有分布式空间距离连接系统以空间对象集合 S 的最小边界矩形 $S.mbr$ 为全局域,选取的空间范围过大。如图2所示,仅仅落在 $S.mbr$ 中灰色部分的空间对象与 R 中任意空间对象的距离必定大于 δ,肯定不会出现在结果集中,因此可以提前过滤掉。若仍保留这些空间对象,将会造成以下额外开销:1) 不必要的计算存储开销,对于仅仅落在 $S.mbr$ 中灰色部分的空间对象,我们仍需要对其采样、划分子空间、构建索引,因而消耗很多额外的CPU资源和内存资源;2) 不必要的网络传输开销,在并行计算开始前,需要将两个数据集进行合并操作,这涉及不同机器之间的数据传输(在Spark中称为Shuffle操作)。该操作非常耗时,若不提前对冗余数据进行过滤,将会严重影响空间距离连接的运算性能。

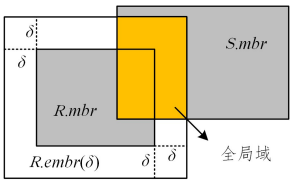


图2 全局域计算(电子版为彩色)

Fig. 2 Global space calculation

为此,JUST-Join算法提出了一种新的全局域计算方法,能够最小化全局域,提前过滤掉无效的数据,极大地提升空间距离连接的性能。如图2所示,JUST-Join首先计算空间数据集 S 的最小边界矩形 $S.mbr$,同时计算空间数据集 R 的扩展边界矩形 $R.embr(\delta)$,全局域 G 即为 $S.mbr$ 与 $R.embr(\delta)$ 重叠的部分,表示为 $G=S.mbr \cap R.embr(\delta)$,如图2中橙色部分所示。注意,这里JUST-Join并不计算空间数据集 S 扩展边界矩形。在2.1.3节的重分区过程中,对 R 和 S 中空间对象的处理方法是不同的,能够保证不漏解。

2.1.2 子空间划分

获得全局域 G 后,我们需要对 G 进行划分,得到 n 个子空间 $\{G_1,G_2,G_3,\cdots,G_n\}$,每个子空间最终会对应于分布式系统的一个分区。我们希望每个分区的数据量大体相等,这样就能够实现负载均衡,提高算法整体的运行效率;同时,本文能够保证不同分区之间相互独立,减少空间对象在不同分区内的重复出现和重复计算。因此,子空间应满足 $G_1 \cup G_2 \cup \cdots \cup G_n=G$,且 $G_i \cap G_j=\varnothing, i \neq j, 1 \leqslant i,j \leqslant n$。

现有分布式系统在对 G 进行划分时,仅仅考虑了空间数据集 S 的分布,并未考虑 R 的分布,最终导致每个分区的数据量可能相差很大,从而造成严重的数据倾斜,拖慢整体的运行效率。为此,JUST-Join在划分子空间的同时考虑了空间数据集 R 和 S 的数据。其大体思想是,先分别对 R 和 S 数据集进行采样,然后利用四叉树[19]的思想对 G 进行划分,分别得到两组不同的子空间划分 G^R 和 G^S,最后将 G^R 和 G^S 进行合并,得到最终的子空间划分。注意,处理 R 和 S 时,情况有些不同。

(1)数据采样。针对 S 数据集,我们以采样率 η 随机采样与 G 有重合的数据,得到数据集 S'。对于 R 数据集,我们以采样率 η 随机采样与 $G.embr(\delta)$ 有重合的数据,得到数据集 R',因为对于与 $G.embr(\delta)$ 重合但不与 G 重合的空间对象 $r \in R$,仍然可能与某个空间对象 $s \in S$ 的距离小于 δ。文献[19]表明,对于海量空间数据,当采样率 $\eta=0.01$ 时,采样的数据能够较好地表示整个数据集的空间分布。因此,本文取 $\eta=0.01$。

(2)四叉树划分。我们分别利用 S' 和 R' 对全局域 G 进行四叉树划分,得到两组空间划分 $\{G_1^{S'},G_2^{S'},\cdots,G_m^{S'}\}$ 和 $\{G_1^{R'},G_2^{R'},\cdots,G_k^{R'}\}$。使用四叉树的原因是,四叉树能够保证所有叶子节点中的样本数目大体相同,从而达到负载均衡。

对于采样数据集 S' 来说,其四叉树划分的大体过程为:

(1) 统计 $s \in S'$ 中,$s.mbr$ 与全局空间 G 相交的样本数目 ξ,若 ξ 大于给定阈值 z,则将 G 平均分裂成4份,得到4个子空间 $\{G_1^{S'},G_2^{S'},G_3^{S'},G_4^{S'}\}$。

(2) 对于每个子空间,递归重复步骤(1)的过程,直到与每个子空间相交的样本数目小于阈值 z,或者分裂层级达到给定值 φ,停止分裂。本文采用文献[6]的方法,令 $z=|S'|/\alpha$,$\varphi=15$,其中 $|S'|$ 表示数据集 S' 中空间元素的个数,α 表示数据集 R 和 S 原始分区数的最大值。

对于采样数据集 R' 来说,其四叉树划分过程与 S' 类似,不同之处在于,其考虑的是 $r \in R'$ 的扩展最小边界矩形 $r.embr(\delta)$。最后,将得到的两组四叉树划分合并起来,得到全局四叉树划分,并对每个最终的子空间进行编号,如图3所示。合并的规则为:对于 S' 四叉树划分和 R' 四叉树划分的每个子空间,若两个子空间有交叉,则只保留分裂次数较多的子空间。

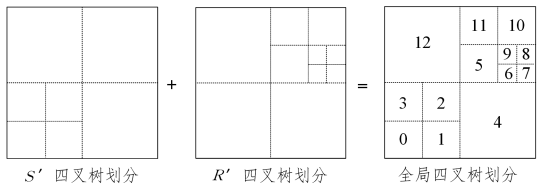


图3 子空间划分

Fig. 3 Subspace partitioning

2.1.3 数据重分区

获得子空间划分 $\{G_1,G_2,G_3,\cdots,G_n\}$ 后,需要对空间数据集 R 和 S 进行重新分区,共分成两步。

(1)编号分配。对于所有空间数据 $r \in R$ 和 $s \in S$,都分配若干个子空间编号,此过程在分布式系统中是并行处理的,不涉及不同机器之间的数据传输,因此执行效率很快。对于数据集 R 和 S,其编号分配的过程有一些细微的区别。

对于每个空间对象 $s \in S$,我们判断 $s.mbr$ 与每个子空间的位置关系。若 $s.mbr$ 与所有的子空间均不相交,那么将 s 舍弃;若 s 与某个子空间相交,则将此子空间的编号绑定在 s 上,最终 s 可能会绑定多个不同的子空间编号。

对于每个空间对象 $r \in R$,我们判断的是 $r.embr(\delta)$ 与每个子空间的位置关系。其他过程与处理 S 中的空间对象一致。

(2)数据转移。对于所有绑定了空间编号的空间对象,根据空间编号进行重新分区,最终得到 n 个数据分区。其中具有相同空间编号的数据会被分配到同一个数据分区中。若某个空间对象绑定了多个空间编号,那么将会被拷贝多份。此过程会涉及不同机器之间的数据传输,将会成为整个距离连接算法的一个瓶颈。JUST-Join 只选取了最小的全局空间范围 G,同时对落在 G 之外的数据提前进行过滤,因此极大地加快了数据重分区的处理过程。

2.2 并行计算

对于空间编号为 i 的分区 P_i,其中的数据可分为两部分:来自空间数据集 R 的数据 R_i,以及来自空间数据集 S 的数据 S_i。在并行计算过程中,每个分区内独立执行空间距离连接操作,共包含 3 个步骤。

(1) 局部索引构建。对 S_i 数据构建局部空间索引。文献[6]的实验结果表明,对于大数据量的情况下,R-index[16]比四叉树[15]的空间过滤效果更好,因此 JUST-Join 采用了 R-index[16]对 S_i 数据集构建局部空间索引 $Idx(S_i)$。

(2) 空间范围查询。对于每个空间对象 $r \in R_i$,利用构建好的局部空间索引 $Idx(S_i)$,查找所有与 r 距离小于 δ 的 $s \in S_i$,二元组 (r,s) 即为候选结果项。

(3) 结果验证并输出。每个分区都将得到一个二元组候选结果集合。但由于一个空间对象可能会跨多个子空间,在不同的分区之间可能会有重复的数据,导致不同分区的结果项有可能重复。如图 4 所示,在分区 0 和分区 1 中,都会产生 (r,s) 结果项。为了去除重复的结果项,分布式系统 Spatial-Spark[8]使用 Spark 提供的 distinct 方法去重,但这会导致不同机器之间的数据传输,从而降低空间距离连接算法的效率。为此,JUST-Join 采用 GeoSpark[6]的去重策略,能够避免不同分区、不同机器之间的数据传输,加快空间距离连接算法的运行效率。

JUST-Join 的去重过程大体如下。对于分区 P_i 的每个候选结果项 (r,s),我们计算 $r.embr$ 与 $s.mbr$ 相交的区域 $O = r.embr \cap s.mbr$。若 O 的左下角的顶点落在了 P_i 所表示的空间区域外,则丢弃 (r,s),否则保留 (r,s) 并输出。实际上,选取 O 的哪个角并不重要,只要按统一的标准就能够达到去重的目的。图 4 中,橙色区域即为 O,由于 O 的左下角落在了分区 0,因此分区 0 的 (r,s) 将保留并输出,分区 1 的 (r,s) 将会被丢弃。

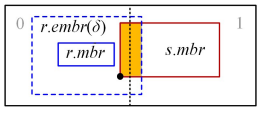


图 4 结果验证示例(电子版为彩色)

Fig. 4 Example of result verification

2.3 空间自连接优化

当空间对象集合 $R = S$ 时,称此类空间距离连接为空间自连接。现有分布式平台均没有对自连接的情况进行优化。对于空间对象集 R 的自连接,若仍按前文提到的方法进行处理,将会遇到以下 3 个问题:1) 此时全局域为 $R.mbr$,所有 R 中的对象都会与 $R.mbr$ 相交,造成全局域过滤失效;2) 前文提到的算法将会加载、采样、传输同一数据集 R 两次,造成存储空间、计算资源、网络资源的浪费;3) 在并行计算阶段,对于同一分区的 $r_i \in R, r_j \in R$,若 (r_i, r_j) 是满足条件的数据项,那么 (r_j, r_i) 必定也满足条件,前文提到的方法未对此进行优化,而是查询了两遍。

针对上述问题,JUST-Join 对空间对象集合 R 的自连接进行了优化,具体做法如下:

(1)空间分区阶段,我们令全局域 $G = R.mbr$,然后对 R 按采样率 η 进行采样,得到采样数据集 R'。接着,在四叉树划分步骤中,统计所有 $r \in R'$ 中,$r.embr(\delta/2)$ 与全局空间 G 相交的样本数目 ξ,若 ξ 大于给定阈值 z,则将 G 平均分裂成 4 份,得到 4 个子空间;针对每个子空间,递归重复前一步骤,直到与每个子空间相交的样本数目小于阈值 z,或者分裂层级达到一给定值 φ 为止,得到的四叉树划分即为最终的子空间划分,并对每个子空间进行编号。之后,我们为每个 $r \in R$ 绑定若干编号,若 $r.embr(\delta/2)$ 与某个子空间有重合的区域,则将对应的子空间编号绑定给 r。最后,对 R 按照绑定的编号进行重新分区,相同的编号划分到同一个分区中,绑定了多个编号的空间元素也将拷贝多份。

注意,与通用空间距离连接不同,首先这里直接令 $R.mbr$ 作为全局域;然后对 R 数据集只做了一次采样和重分区;最后在四叉树划分、编号绑定时使用的是 $r.embr(\delta/2)$,因为 $r_i.embr(\delta/2)$ 与 $r_j.embr(\delta/2)$ 相交等价于 r_i 和 r_j 距离小于等于 δ,使用 $r.embr(\delta/2)$ 能保证满足条件的数据 r_i, r_j 在同一个分区中,还能尽量减少每条数据绑定的编号数,减少数据重分区时不同机器之间的数据传输,提高了算法的整体运行效率。

(2)并行计算阶段,每个空间分区 P_i 中仅有来自集合 R 的数据 R_i。为了避免重复计算,每个分区内,我们采用了平面扫描算法(Plane Sweep Algorithm)[20],包括以下 3 个步骤:

1)数据排序。对所有 $r \in R_i$ 按 $r.embr(\delta/2).x_{\min}$ 做升序排序,得到集合 $R_i' = \{r_1, r_2, \cdots, r_n\}$,其中 n 是 R_i 的空间对象数量。

2)距离计算。采用平面扫描算法,从左往右依次扫描 R_i',得到候选集 $C_i = \{(r_j, r_k)\}$,满足:$1 \leqslant j < k \leqslant n, r_j \in R_i', r_k \in R_i'$,且 $r_j.embr(\delta/2).x_{\min} \leqslant r_k.embr(\delta/2).x_{\min} \leqslant r_j.embr(\delta/2).x_{\max}$,即 $r_j.embr(\delta/2)$ 和 $r_k.embr(\delta/2)$ 在 x 轴方向有重合。对候选集 C_i 中的每个候选二元组 (r_j, r_k) 做进一步提纯,验证 $O = r_j.embr(\delta/2) \cap r_k.embr(\delta/2)$,如果 O 为空,则表示 r_j 与 r_k 距离大于 δ,丢弃该二元组;否则判断 O 的左下角是否位于子空间 P_i 内,若不在,则将该二元组作为重复结果丢弃;否则判断 $d(r_j, r_k)$ 是否小于等于 δ,若是,则保留该二元组,否则丢弃。最终得到结果集 $D_i = \{(r_j, r_k), r_j \in R_i', r_k \in R_i', d(r_j, r_k) \leqslant \delta, 1 \leqslant j < k \leqslant n\}$。因为满足条件 $j < k$,所以该方法避免了重复计算。

3)结果扩充。为了与未做优化的自连接结果保持一致,需要对结果集 D_i 进行扩充。首先对于 D_i 中的每个二元组 (r_j, r_k),构造与之对等的二元组 (r_k, r_j) 并加入 D_i 中,然后对 R_i 中的每个空间对象 r 构造二元组 (r,r) 并加入 D_i 中。最后将所有分区的 D_i 合并输出,即为自连接的最终结果。

通过上述优化,既可以避免空间分区阶段中出现的数据

重复加载、重复计算、冗余传输的问题，又可以避免并行计算阶段的重复查询，极大地提升了空间自连接的运算性能。

3 实验

3.1 实验数据与实验环境

本文使用文献[21]提供的 OSM(Open Street Map)空间数据集。表 1 列出了实验中使用到的点(points)数据和多边形(polygons)数据，这些数据遍布全球且在空间上分布不均匀。

表 1 实验数据集

Table 1 Details of experimental datasets

数据集	数据量/亿	原始文件/GBs	坐标点/亿
Points	2	5.4	2
Polygons	1.14	19	7.64

本文的实验环境是 5 台服务器组成的分布式集群，单台服务器配置为 CentOS 7.4、30 核 CPU、100 GB 内存和 1T 硬盘。用到的大数据组件为 Hadoop 2.7.3 和 Spark 2.3.3。

3.2 实验设置

为了验证本文全局域计算的优势，在全球空间范围内划分出 5 个长宽跨度均为 180°的正方形区域，即 $Grid_1$，$Grid_2$，$Grid_3$，$Grid_4$ 和 $Grid_5$，其中 $Grid_1$ 与其他 4 个区域相交部分的面积分别是 $Grid_1$ 面积的 20%，40%，60%，80%，本文称之为空间重合度(overlap)，$Grid_1$ 与自身的空间重合度为 100%。实验中，$Grid_1$ 包含的数据集为 R，其他区域包含的数据为 S。实验以两个数据集的 $overlap$ 为变量，观察 $overlap$ 分别为 20%，40%，60%，80%，100%的空间距离连接计算耗时。另外，为了观察实验性能随 δ 的变化，本文统计了 δ 分别为 0，50，100，150 和 200 m 时的空间距离连接的计算耗时。注意，由于部分系统不支持空间距离连接，仅支持空间连接，即 $\delta=0$ 的情况，因此在以 $overlap$ 为变量时，δ 的默认取值为 0。以 δ 为变量时，$overlap$ 的默认取值为 60%。

本文与 GeoSpark，SpatialSpark 和 LocationSpark 做了对比实验。当 δ 大于 0 时，GeoSpark 用空间对象 r 的最小边界矩形 $r.mbr$ 的中心点 c 与空间对象 s 进行距离计算，当 r 为点时，r 与 c 等价，因此 GeoSpark 支持点与其他空间数据类型的空间距离连接；当 r 不是点时，r 与 c 不等价，此时 GeoSpark 仅仅支持近似的空间距离连接，与 JUST-Join 和 SpatialSpark 的精确计算没有可比性。另外，LocationSpark 只支持 MBR 与点之间的空间连接($\delta=0$)，因此 LocationSpark 的实验结果是数据集 Polygons 中每个多边形对象的 MBR 与数据集 Points 的计算耗时。

本文根据文献[21]，将所有数据集初始的分区数设为 1024。但 LocationSpark 做了内部优化，其分区数不受初始值的约束。

3.3 实验结果

图 5 给出了点与点之间的空间距离连接耗时随着距离 δ 和重合度 $overlap$ 的变化情况。由图可知，随着 δ 增大，所有方法的计算耗时均增加，因为 EMBR 越大，空间数据的拷贝传输量越多，产生的结果项就越多。JUST-Join 比 SpatialSpark 和 GeoSpark 快，因为我们选取的全局域最小，能够提前过滤大量数据，并且综合考虑了两个数据集的分布来划分子空间，实现了高效的负载均衡。SpatialSpark 将两个集合空间域的最小边界矩形 $(R.mbr \cup R.mbr).mbr$ 作为全局域，但仅仅以样本集 S' 对全局域进行划分生成子空间，这导致当两个数据集的 $overlap$ 越小时，子空间之间的数据倾斜越严重。SpatialSpark 在空间重分区中使用了 Spark 的 groupByKey 和 join 算子，且使用 Spark 的 distinct 算子实现去重，这 3 个算子均为 Spark 的 Shuffle 操作，对数据倾斜极为敏感。因此，图 5 中，随着 $overlap$ 变小和 δ 的增大，SpatialSpark 的数据倾斜会越严重，计算耗时越长，且当 $overlap$ 小于等于 40%、δ 大于 150 m 时，Shuffle 操作的失败导致计算任务无法完成。

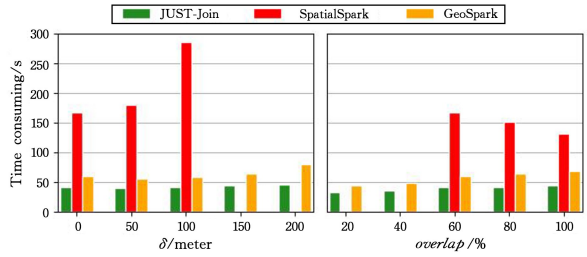


图 5 点与点的空间距离连接

Fig. 5 Point-Point spatial distance join

图 6 给出了多边形与点的空间距离连接实验性能。JUST-Join，GeoSpark 和 SpatialSpark 的现象与图 5 中相似。另外增加了 LocationSpark 的对比，由于没有优化全局域的计算，也没有综合考虑两个样本集的空间分布，LocationSpark 的性能同样不如 JUST-Join。

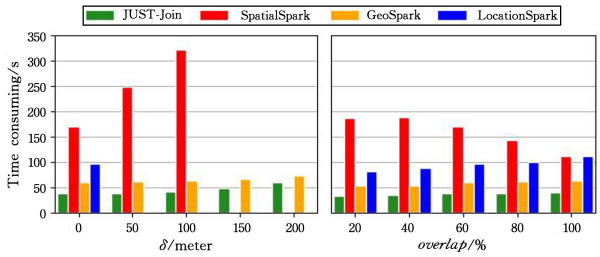


图 6 多边形与点的空间距离连接

Fig. 6 Polygon-Point spatial distance join

图 7 给出了多边形与多边形的空间距离连接性能，JUST-Join 的性能仍然最好。SpatialSpark 因为 Shuffle 操作的失败，在 δ 大于等于 100 或 $overlap$ 小于等于 40%时无法完成计算。如 3.2 节中提到的，δ 大于 0 时，GeoSpark 仅支持多边形与多边形之间的近似空间距离连接，因此没在此处参与比较。

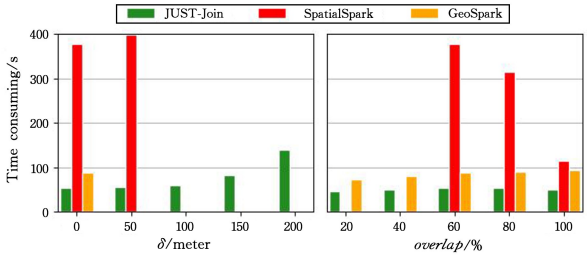


图 7 多边形与多边形的空间距离连接

Fig. 7 Polygon-Polygon spatial distance join

如图 8 所示，本文还使用 $Grid_1$ 内的点和多边形数据对空间自连接做了对比实验。随着 δ 增大，优化后的性能均好于优化前的性能，空间自连接优化效果明显。

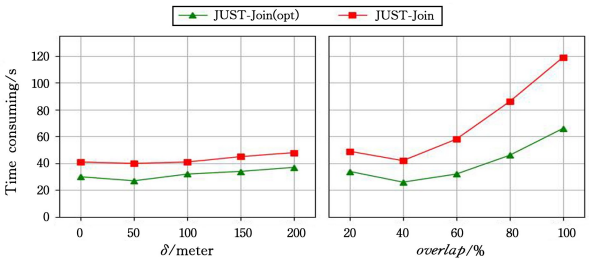


图 8 空间自连接

Fig. 8 Spatial self-join

结束语 本文针对现有的分布式空间距离连接算法在全局域计算和子空间划分方面存在的不足，提出了一种新的全局域计算方法，并通过综合考虑两个空间对象集合的空间分布差异，实现了可提前过滤数据，保证了空间距离连接算法 JUST-Join 高效且负载均衡。经过实验对比，JUST-Join 的性能超出目前最优的分布式空间分析系统 GeoSpark 20%以上。此外，JUST-Join 对空间自连接做了特别的优化，性能相比优化前提升了 20%～80%。本文实现的空间距离连接是一种离线的分析算法，在后续的工作中将研究大规模空间数据的实时空间距离连接。

参 考 文 献

[1] HE H, LI R, WANG R, et al. Efficient suspected infected crowds detection based on spatio-temporal trajectories[J]. arXiv: 2004.06653, 2020.

[2] JACOX E H, SAMET H. Spatial join techniques [J]. ACM Transactions on Database Systems (TODS), 2007, 32(1): 7.

[3] CHEN D H, LIU L X, LE J J. Research on a Spatial Join Query with Keyword Search[J]. Computer Science, 2009, 36(7): 150-152.

[4] ELDAWY A, MOKBEL M F. Spatialhadoop: A mapreduce framework for spatial data[C]// ICDE. IEEE, 2015: 1352-1363.

[5] 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.

[6] YU J, WU J, SARWAT M. Geospark: A cluster computing framework for processing large-scale spatial data[C]// SIGSPATIAL. 2015: 1-4.

[7] TANG M, YU Y, MALLUHI Q M, et al. Locationspark: A distributed in-memory data management system for big spatial data [J]. PVLDB, 2016, 9(13): 1565-1568.

[8] YOU S, ZHANG J, GRUENWALD L. Large-scale spatial join query processing in cloud[C]// ICDE. IEEE, 2015: 34-41.

[9] XIE D, LI F, YAO B, et al. Simba: Efficient in-memory spatial analytics[C]// ICDE. 2016: 1071-1085.

[10] YANG K, DING X, ZHANG Y, et al. Distributed Similarit y Queries in Metric Spaces[J]. Data Science and Engineering, 2019, 4(2): 93-108.

[11] DEAN J, GHEMAWAT S. MapReduce: a flexible data processing tool[J]. Communications of the ACM, 2010, 53(1): 72-77.

[12] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster computing with working sets[C]// Proceedings of 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010.

[13] QIAO B, HU B, ZHU J, et al. A top-k spatial join querying processing algorithm based on spark[J]. Information Systems, 2020, 87: 101419.

[14] WHITMAN R T, MARSH B G, PARK M B, et al. Distributed spatial and spatio-temporal join on apache spark[J]. TSAS, 2019, 5(1): 1-28.

[15] FINKEL R A, BENTLEY J L. Quad trees a data structure for retrieval on composite keys[J]. Acta informatica, 1974, 4(1): 1-9.

[16] GUTTMAN A. R-trees: A dynamic index structure for spatial searching[C]// SIGMOD. 1984: 47-57.

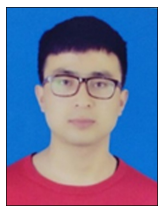
[17] LI R, HE H, WANG R, et al. Just: Jd urban spatio-temporal data engine[C]// ICDE. IEEE, 2020: 1558-1569.

[18] LI R, HE H, WANG R, et al. Trajmesa: A distributed nosql storage engine for big trajectory data[C]// ICDE. IEEE, 2020: 2002-2005.

[19] ELDAWY A, ALARABI L, MOKBEL M F. Spatial partitioning techniques in SpatialHadoop[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1602-1605.

[20] PREPARATA F P, SHAMOS M I. Computational geometry: an introduction[M]. Springer Science & Business Media, 2012.

[21] PANDEY V, KIPF A, NEUMANN T, et al. How good are modern spatial analytics systems? [J]. Proceedings of the VLDB Endowment, 2018, 11(11): 1661-1673.



WANG Ru-bin, born in 1994, postgraduate. His main research interests include spatio-temporal data management.

LI Rui-yuan, born in 1990, Ph. D. His main research interests include spatio-temporal data management and mining.

(责任编辑：李亚辉)