计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20160558
54(7):1592-1602,2017

一种基于 Spark 的多路空间连接查询处理算法

乔百友[1,2] 朱俊海[1] 郑宇杰[1] 申木川[1] 王国仁[1]
[1](东北大学计算机科学与工程学院 沈阳 110819)
[2](杨百翰大学计算机科学系 美国犹他州普若佛 84602)
(qiaobaiyou@mail.neu.edu.cn)

A Multi-Way Spatial Join Querying Processing Algorithm Based on Spark

Qiao Baiyou[1,2], Zhu Junhai[1], Zheng Yujie[1], Shen Muchuan[1], and Wang Guoren[1]
[1](*School of Computer Science and Engineering, Northeastern University, Shenyang* 110819)
[2](*Department of Computer Science, Brigham Young University, Provo, Utah, USA* 84602)

Abstract Aiming at the problem of spatial join query processing in cloud computing systems, a multi-way spatial join query processing algorithm BSMWSJ is proposed, which is based on Spark platform. In this algorithm, the whole data space is divided into grid cells with the same size by grid partition method, and spatial objects in each type data set are distributed into these grid cells according to their spatial locations. Spatial objects in different grid cells are processed in parallel. In multi-way spatial join query processing, a boundary filtering method is proposed to filter the useless data, which calculates the MBRs of the candidate results generated by the previous join processing, and uses these MBRs to filter the subsequent join data sets. This allows it to filter out the useless spatial objects, and reduce the redundant projection and replication of spatial objects. At the same time, a duplication avoidance strategy is applied to reduce the outputs of redundant results, and further minimizes the cost of the subsequent join processing. Many experiments on synthetic and real data sets show that the proposed multi-way spatial join query processing algorithm BSMWSJ has obvious advantages and better performance than the existing multi-way spatial join query processing algorithms.

Key words cloud computing; Spark platform; multi-way spatial join query; boundary filtering; duplication avoidance

摘 要 针对云环境下空间数据连接查询处理问题,提出了一种基于 Spark 的多路空间连接查询处理算法 BSMWSJ.该算法采用网格划分方法将整个数据空间划分成大小相同的网格单元,并将各类数据集中的空间对象,根据其空间位置划分到相应的网格单元中,不同网格单元中的空间数据对象进行并行连接查询处理.在多路空间连接查询处理过程中,采用边界过滤的方法来过滤无用数据,即通过计算前面连接操作候选结果的 MBR 来过滤后续连接数据集,从而过滤掉无用的连接对象,减少连接对象的多余投影与复制,并采用重复避免策略来减少重复结果的输出,从而进一步减少后续连接计算的代价.合成数据集和真实数据集上的大量实验结果表明:提出的多路空间连接查询处理算法在性能上明显优于现有的多路连接查询处理算法.

收稿日期:2016-08-02;**修回日期**:2016-10-20
基金项目:国家自然科学基金项目(61073063,61332006);国家海洋公益性行业科研专项经费项目(201105033)
This work was supported by the National Natural Science Foundation of China (61073063, 61332006) and the National Marine Industry Research Special Funds for Public Welfare Projects (201105033).

关键词 云计算;Spark 平台;多路空间连接查询;边界过滤;重复避免

中图法分类号 TP311.13

空间数据查询处理技术一直是空间数据管理领域的研究热点,而空间连接查询是一种常用的空间查询类型,也是该领域的重要研究课题之一.空间连接查询作为一种基本空间操作,是最耗时的操作之一,由于其复杂性和重要性使之成为决定空间数据管理系统整体性能的重要因素.特别是近年来,随着物联网技术、对地观测技术和基于位置的服务技术等技术的快速发展和广泛应用,空间数据规模急剧增加,已成为一类非常重要的大数据,在这种情况下,如何对这类大数据进行高效的空间连接查询处理已成为当前研究的热点之一.显然,传统的空间数据库技术由于其扩展性问题而难以满足这类大数据快速查询处理的要求,而 Spark[1] 作为一种新型的超大规模数据分布式并行处理平台而受到人们的广泛重视,也是目前大数据处理的关键技术.然而由于 Spark 平台并未对连接操作提供内在的支持和优化,因此研究如何利用 Spark 这种分布式并行处理平台来实现对空间大数据的高效空间连接查询处理,具有重要的理论研究意义和应用价值.

目前已有研究者在 Hadoop 平台下,就空间连接查询处理算法进行了较深入研究,并取得了一系列理论和应用成果,但这些成果主要集中在相似性连接查询算法、2 路空间连接算法和关系数据上的多路连接查询算法等方面,而对于多路空间连接查询处理算法的研究成果还相当有限,Spark 平台下的多路空间连接查询处理算法的研究才刚刚开始.在通用多路连接查询处理研究方面,Afrati 等人[2] 和 Lin 等人[3] 主要就 MapReduce 框架下,关系数据上的多路连接查询处理问题进行了研究,并给出了 3 种优化策略;而 Jiang 等人[4] 和王晓军等人[5] 的研究则主要针对 MapReduce 框架下多路连接查询中的 I/O 开销问题,并提出了 Map-Join-Reduce 的编程框架及相关的优化算法.Slagter 等人[6] 则主要从网络流量角度入手,提出通过在多个 Reducer 之间重新分配元组,从而达到减少连接时间.上述研究工作主要基于 Hadoop 平台,并且聚焦于通用多路连接查询处理优化方面,而本文的工作则主要聚焦于 Spark 平台下的多路空间连接查询处理问题.

在基于 MapReduce 的多路空间连接查询处理方面,王璟玢等人[7] 针对小规模的集中式多路连接查询处理,提出了基于 R 树连接的多路空间限制策略和多路平面扫描的优化技术,显然,并不适合大规模分布式的多路连接查询处理;Gupta 等人[8-9] 提出了 2 种多路空间连接查询处理算法 Controlled-Replicate 和 ε-Controlled-Replicate. Controlled-Replicate 将各类连接数据集中的空间对象划分并复制到第 4 象限中的所有网格单元,然后进行多路连接运算.显然这种方法造成了大量对象的复制,影响连接处理效率.为此作者又提出了改进的多路空间连接查询处理算法 ε-Controlled-Replicate,该算法减少了数据复制,在一定程度上提高了处理效率,但是还存在着复制过多的问题.

针对当前最新的代表性多路空间连接算法 ε-Controlled-Replicate 中存在的数据复制过多、影响查询处理效率的问题,本文基于最新的分布式并行计算框架 Spark,充分利用其内存计算和 RDD 分布式弹性数据集的特性,从数据划分和复制入手,提出了一种基于 Spark 的多路空间连接查询处理算法 BSMWSJ,该算法将数据空间划分成大小相同的网格单元,并将数据集中的空间对象,按照其所在空间位置复制到与其相交叠的网格单元中,每个网格单元中的数据实现并行空间连接处理.在连接过程中,采用边界过滤方法来减少无用连接数据,首先,对划分到每个网格单元的第 1 次连接所需的 2 类数据集执行连接运算,并对所生成的连接候选结果中的一类待连接数据集,计算其 MBR;其次,利用该 MBR 来实现对后续要连接数据集的过滤,从而过滤掉无结果的后续连接对象,减少后续连接的多余计算,以及连接对象的多余投影与复制,并采用重复避免策略来减少重复结果的输出,从而全面减少后续连接计算的代价,提高多路连接查询处理的效率.合成数据和真实数据集上的大量实验结果表明,本文提出的多路空间连接查询处理算法在性能上明显优于 ε-Controlled-Replicate 算法,具有良好的扩展性和适应性.

1 相关工作

国内外对 MapReduce 框架下的连接查询处理算法及其优化技术的研究已开展得较为广泛,目前这些研究工作主要集中于相似性连接查询算法、Theta 连接和 2 路空间连接算法等方面.在多路连

接查询处理算法方面的研究相对开展的不多,而Spark环境下的多路空间连接查询处理算法的研究成果更是相当有限.

在相似性连接查询处理和Theta连接算法等方面,Luo等人[10]首次利用MapReduce模型来处理高维相似性连接查询问题,提出了1种新颖的降维技术DAA,并给出了2种并行处理框架OSFR和TSFR,DAA虽然能够减少高维向量之间的计算代价,但不能减少总的比较次数.为此Ma等人[11]提出了有效减少DAA和初始向量计算次数的方法SAX和PAA,能够将高维向量分成不同的组,并提出了基于SAX和改进SAX的相似性连接查询处理算法;文献[12]则研究了基于MapReduce框架的Top-k相似性连接处理算法,提出了分治和剪枝策略,并在此基础上提出了全分区方法和重要元组分区方法来最小化Map和Reduce任务之间数据通信量,从而达到减少后续计算代价的目的;文献[13]则主要研究了MapReduce框架下的集合相似性连接算法,提出了3阶段进行集合相似性连接的方法,实现了自连接、RS连接等示例,并提出了保证负载均衡和最小化复制的方法;文献[14]主要研究了Hadoop下大规模不确定数据上的集合相似性连接方法,并结合前缀过滤原则,提出了Map端剪枝、Reduce端剪枝和混合剪枝3种方法来减少后续比较代价;文献[15]提出了1种以Reducer为中心的代价模型和基于MapReduce框架的Theta连接模型,并在该模型基础上提出了1-Bucket-Theta随机算法,该算法在足够的统计信息支持下具有较高Theta连接效率;文献[16]则主要聚焦于非对称分片复制连接问题,提出一种基于自适应分片的优化算法AFR-AS,来降低MapReduce下任务启动开销以及非对称分片复制连接中的数据广播开销;卞昊穹等人[17]提出了一种基于Spark的等值连接优化算法,该算法结合了半连接与划分连接的优势,并充分利用Spark内存计算模型的特性,提高了等值连接处理性能.

在2路空间链接查询处理方面,Wang等人[18]对2路空间连接算法进行了研究,提出了基于负载均衡的空间对象分区方法,并采用基于带的双向平面空间扫描技术来减少连接计算的代价;文献[19-20]研究了基于MapReduce的2路空间连接算法,首次提出了2路空间查询处理算法SJMR,但该算法没有考虑过滤阶段的优化问题,导致了大量无用的计算操作,增加了查询处理代价;为此,Qiao等人[21]改进了原有SJMR算法,提出了一种基于边界过滤的空间连接查询处理算法BFSJMR,该算能够过滤掉无用的连接查询代价,从而提高了2路空间连接查询处理效率.

在多路连接查询处理方面,Afrati等人[2]和Lin等人[3]主要就MapReduce框架下的多路连接查询处理问题进行了研究,并给出了3种优化策略,然而该研究工作主要针对关系数据,显然不适合于多路空间连接查询处理.Jiang等人[4]和王晓军等人[5]就MapReduce框架下多路连接查询中巨大I/O开销问题进行了研究,提出了Map-Join-Reduce的编程框架及相关的优化算法.虽然在一定程度解决了I/O开销问题,但由于对原有的MapReduce编程框架进行了较大修改,造成了兼容问题,不利于原有框架的完整性;Slagter等人[6]提出了一种网络感知的多路连接方法,通过感知网络流量实现多个Reducer之间元组的重新分配,从而达到减少连接时间;孙莉等人[22]则就列存储数据中连接查询优化问题进行了研究,提出了基于规则的连接策略优化方法,并设计了相应的优化算法,在此基础上提出了相应的代价估算模型,实现了策略的选择;周国亮等人[23]针对联机分析处理要求,提出一种能够适合Spark环境并结合多维Bloom Filter的星型连接算法,该算法能够避免事实表数据的移动,并利用多维布隆过滤器技术来减小需要广播的数据量,该算法充分结合了广播连接和重划分连接的优势.

在基于MapReduce的多路空间连接查询处理方面,王璟玢等人[7]提出了基于R树连接的多路空间限制策略和多路平面扫描的优化技术,然而该研究主要针对小规模的集中式多路连接查询处理.Gupta等人[8-9]提出了2种多路空间连接查询算法Controlled-Replicate和ε-Controlled-Replicate.Controlled-Replicate算法采用空间划分方法,将各类数据集中的空间对象划分并复制到第4象限中的所有网格单元,然后对每个网格单元中的数据分别进行多路连接运算.ε-Controlled-Replicate算法是在Controlled-Replicate算法基础上提出的一种改进算法,主要是通过减少数据复制来降低通信代价,从而在一定程度上提高了多路空间连接查询处理的效率.

针对现有多路空间连接查询处理算法存在的问题,本文从减少数据复制和计算代价角度入手,结合Spark内存计算框架的优势,提出了一种基于Spark的多路空间连接查询处理算法,是一种类似于ε-Controlled-Replicate的多路空间连接查询算法.

2 多路空间连接查询定义

空间数据对象有多种类型,大多都是不规则的形状,因此判断 2 个空间对象是否符合某个查询谓词的代价非常昂贵.在空间连接查询处理中通常采用最小边界矩形(minimum bounding rectangle,MBR)来代表一个空间对象,仅当 2 个对象的 MBR 有交叠时,才进一步判断这 2 个空间对象是否真正有交叠,这种分步的处理方法具有更高的处理效率.本文主要针对链式多路空间连接查询,可以用一张图来形象表示,图中的节点对应空间数据集,图中的边对应于连接谓词,这样就形成了一个链图.

链式多路空间连接查询通常定义为:给定空间关系 $R_1,R_2,\cdots,R_n(n>2)$,找到一组空间对象元组 $(r_1,r_2,\cdots,r_n)$,其中,$r_1\in R_1,r_2\in R_2,\cdots,r_n\in R_n$,空间对象 r_1 和 r_2,r_2 和 r_3,$\cdots$,r_{n-1} 和 r_n 两两之间的几何属性存在相互交叠,可表示为

$$Overlap(P,R_1,R_2,\cdots,R_n)=\left\{(r_1,r_2,\cdots,r_n)\left|\begin{array}{l}\forall i\in(1,2,\cdots,n),r_i\in R_i\\ Overlap(P,r_i,r_{i+1})\end{array}\right.\right\},\quad(1)$$

其中,P 代表交叠连接谓词,$Overlap(P,r_i,r_{i+1})$ 表示空间对象 r_i 和 r_{i+1} 之间满足连接谓词 P.

空间连接查询处理通常分为过滤和精化 2 个阶段,在过滤阶段,通过检查 2 个空间对象的 MBR 来消除不可能成为结果的元组,从而产生候选结果集合;精化阶段则是对候选元组集合进行进一步检测,需要使用计算密集型的几何算法来实现,确定其空间属性是否真正满足其连接谓词.本文所提出的算法重点聚焦于提高过滤阶段的处理效率.

3 基于 Spark 的多路空间连接查询处理算法

本文主要从减少计算量和避免过度复制的角度来优化多路空间连接查询处理,提出了一种 Spark 平台下的多路空间连接查询处理算法 BSMWSJ,该算法采用边界过滤方法,重点是减少过滤阶段的数据复制量和计算量,下面分别从空间划分、数据投影与复制、过滤和重复避免等方面来对算法进行详细描述.

3.1 空间划分和编码

在 Spark 环境下,实现大规模并行空间连接查询处理,首先涉及到的是并行任务的划分,需要将整个算法任务拆分成多个子任务并行执行,这就涉及到数据的分区和编码:首先需要将数据划分到多个分区并进行编码,然后在每个分区上做多路空间连接运算,从而实现并行处理,并降低整个连接操作代价.本文采用网格划分方法,将整个数据空间划分成许多大小相等的网格,每个网格被称为一个分区单元,并对每个分区单元进行编码,然后将数据投影到各个分区单元中,从而实现数据划分.利用 Z-order 填充曲线对每个分区单元进行编码,从而更好地保持数据之间的空间紧邻关系,并通过 Hash 方式将每个分区单元映射给多个 Executor,Z-order 曲线编码配合 Hash 的映射方案,可以让 Executor 得到更均匀的任务映射,并且分区单元数量越多,数据分配的越均匀,有助于解决数据倾斜的问题.投影到分区单元中的多类空间数据对象会被作为 Value 值交给相应 Executor 进行处理.

图 1 所示为一个划分编码的例子,整个数据空间被划分为 16 个分区单元,采用 Z-order 填充曲线进行编码,编号依次从 0~15.划分之后分区单元连同投影到各个分区单元上的数据被分别映射给 3 个 Executor 任务进行并行连接处理.

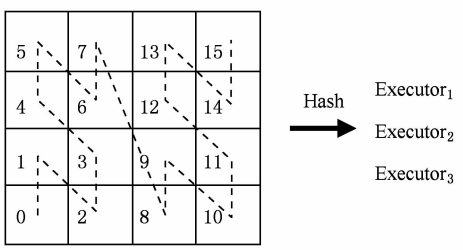

Fig. 1 Demonstration of data partition and encoding

图 1 数据划分与编码示意图

3.2 数据投影与复制操作

整个数据空间被划分成多个网格单元后,空间连接对象需要根据其所在的位置被映射这些网格单元,然后分配给多个 Executor 并行执行连接运算,这首先涉及到数据的投影和复制问题.本文采用简单策略,根据空间连接对象与网格单元的交叠情况进行投影,如果空间对象和网格单元有相交则将其投影到相应的网格中.在多路连接查询处理过程中,生成的中间结果需要根据后续连接要求将其整体复制到相应的网格单元,以进行后续连接处理.下面详细介绍空间对象投影和数据复制操作.

1) 空间对象投影.将空间数据对象根据其所在位置映射到相应的网格单元中.设 $C=(c_1,c_2,\cdots,c_n)$ 代表一个划分,c_i 代表每一个网格单元;设 R 为

一类待连接处理的空间对象集合. 若一个空间对象 $u\in R$,其 MBR 与网格单元 c_i(c_i 为该网格单元的 Z-order 编码)有交叠,则将对象 u 映射到网格单元中 c_i 中,并生成相应键值对(c_i,u),如果一个空间对象和多个网格单元有交叠,则会形成多个键值对. 投影操作可以表示为

$$Project(u,C)\rightarrow\{(c_i,u)\},\forall i,\ \text{s.t.}\ u\cap c_i\neq\varnothing. \quad (2)$$

2) 数据复制操作. 在多路连结查询处理中,需要多个数据集之间进行多次连接,数据复制操作则主要是将当前网格单元上的前 1 次连接的中间结果复制到相关的其他网格单元,从而进行后续的连接操作,其结果与投影操作类似. 若 $t\in T$ 为连接中间结果集中的元组,$t.u$ 为将要进行下一次空间连接的对象,则数据复制操作可以表示为

$$Replicate(t,C)\rightarrow\{(c_i,t)\},\forall i,\ t.u\cap c_i\neq\varnothing. \quad (3)$$

图 2 为投影与复制操作的示例,从图 2 中可以看出,对象 r_1 被投影到 6 号和 12 号网格单元,r_2 被投影到 9 号和 12 号单元,r_3 则被投影到 9 号和 11 号单元. 即 $Project(r_1,C)=\{(6,r_1),(12,r_1)\}$, $Project(r_2,C)=\{(9,r_2),(12,r_2)\}$, $Project(r_3,C)=\{(9,r_2),(11,r_2)\}$. 当执行 r_1,r_2 和 r_3 依次进行多路连接时,由于 r_2 和网格单元 9 有交叠,因此网格单元 12 中的对象 r_1 和 r_2 的连接中间结果(r_1,r_2)要被复制到网格单元 9 中,从而形成键值对$(9,(r_1,r_2))$,实现与网格单元 9 中的空间对象 r_3 的后续连接操作,避免了连接结果的丢失.

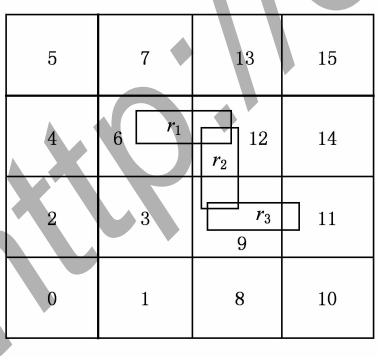

Fig. 2 An example of project and replicate operations

图 2 投影与复制操作示例

3.3 多路空间连接查询算法的总体流程

根据 Spark 并行分布式处理平台特点及其编程模型,本文提出了基于 Spark 的多路空间数据连接查询处理算法(BSMWSJ). 该算法按照 Spark 中有向无环图的思想,将算法中的每个操作作为有向无环图中的节点,依次进行连接操作. 多路空间连接查询 $Q_n=Overlap(R_1,R_2,R_3,\cdots,R_n)$,根据定义,可以表示为 $Q_n=Overlap(\cdots Overlap(Overlap(R_1,R_2),R_3),\cdots,R_n)$.

图 3 为 BSMWSJ 多路空间连接查询处理算法的处理流程,这里仅以 4 路空间连接 $Q_4=Overlap(R_1,R_2,R_3,R_4)$的处理过程为例来进行说明.

$Q_4=Overlap(R_1,R_2,R_3,R_4)$的多路空间连接查询算法的总体处理流程主要包括 4 个操作步骤:

步骤 1. 根据网格划分编码方法对 R_1,R_2,R_3,R_4 数据集进行投影,并将编码值作为 Key 值,将每个空间对象的标识及其 MBR 等属性信息作为 $Value$ 值,形成一系列的键值对,并分别将数据集 R_1,R_2,R_3,R_4 的投影结果放到弹性分布式数据集 RDD_1,RDD_2,RDD_3 和 RDD_4 中.

步骤 2. 计算 $Overlap(R_1,R_2)$,即对 RDD_1 和 RDD_2 执行 $Cogroup$ 操作,将 RDD_1 和 RDD_2 中的数据根据 Key 值聚集到一起得到RDD_{12},对 RDD_{12} 中对象执行空间连接运算. 在运算过程中,首先利用边界过滤策略对 RDD_{12} 进行过滤,去掉不可能有结果的数据对象,然后进行实际空间连接运算,执行重复避免策略,并形成连接中间结果;对连接中间结果执行数据复制操作,形成中间结果数据集 $RDDresult_{12}$.

用一个例子来说明该复制操作的具体处理过程. 假设 2 个空间对象 $r_1\in R_1,r_2\in R_2$,若 r_1 与 r_2 有交叠,则说明它们是连接中间结果,此时利用复制操作 $Replicate$ 计算出 r_2 所跨的所有分区单元,并将其编码作为 Key 值,将 r_1 和 r_2 的 MBR 属性信息等组合在一起作为 $Value$ 值,形成一组 Key-$Value$ 键值对放到中间连接结果 $RDDresult_{12}$ 中. 若 r_1 与 r_2 没有交叠,则不进行处理,这样就避免了无用数据的复制,减少了后续计算代价.

步骤 3. 按照与步骤 2 相同的计算方法计算 $RDDresult_{12}$ 和 RDD_3 之间的连接运算,最终得到 R_1,R_2,R_3 的连接结果 $RDDresult_{123}$.

步骤 4. $RDDresult_{123}$ 与 RDD_4 执行 $Cogroup$ 操作,生成 RDD_{1234},在此基础上进行边界过滤、连接运算处理,并将结果直接输出,形成 $RDDresult_{1234}$,并保存到 HDFS 文件系统. 由于是最后一步连接操作,故不在需要进行复制操作.

上述为 BSMWSJ 多路空间连接算法的处理流程,从中可知除了开始和结束步骤,中间处理步骤是相同的,这也是由链式多路连接查询的性质决定的.

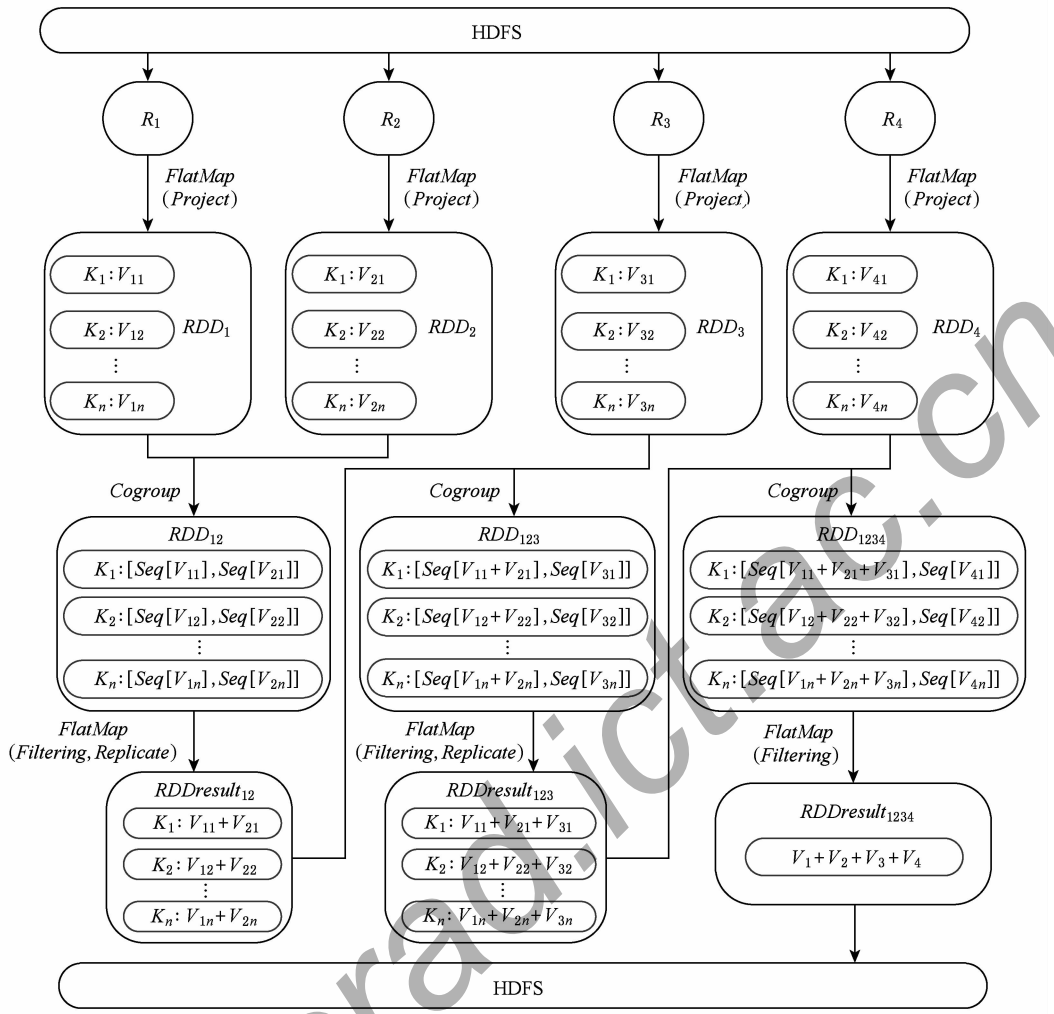

Fig. 3 The processing flow of BSMWSJ multi-way join algorithm

图 3 BSMWSJ 多路连接算法处理流程

3.4 过滤策略

空间连接查询处理通常由过滤和精化 2 个阶段构成,BSMWSJ 算法主要是从减少数据复制和降低计算代价的角度出发,对过滤阶段进行优化. 在 BSMWSJ 算法中,多路连接实际上被拆分成多个 2 路连接来依次并行执行连接运算,在执行连接运算的过程中,采用边界过滤策略,去掉不可能产生结果的元组,并仅对可能有结果的元组进行复制,大大减少存储和后续计算的代价,具体包括 2 种策略.

1) 边界过滤. 在进行连接执行过程中,利用前面已完成的连接结果来过滤即将要连接的数据集,即首先统计前 1 次已完成连接结果中相关连接对象的边界 MBR,并利用该 MBR 来过滤掉后续要连接数据集中不可能有结果的空间对象,从而减少后续连接计算代价. 具体操作可以表示为

$$Filter(t_i,c)\rightarrow\{(c,t_i)\},\forall i,\ \text{s.t.}\ c.mbr_s\bigcap t_i\neq\varnothing,\quad(4)$$

其中,c 代表一个分区单元,t_i 为划分到分区单元 c 中的一个空间对象($t_i\in T$),T 为将要进行连接的数据集. 若 $J_c=R\bowtie\cdots\bowtie S$ 为分区单元 c 中已经执行完成的多次空间连接操的结果集,则 $c.mbr_s$ 为集合 J_c 中相对应的集合 S 中的空间对象的边界 MBR.

图 4 所示是一个边界过滤的例子,其中 3 个数据集 R,S,T 依次进行 3 路连接运算 $R\bowtie S\bowtie T$,投影到网格单元 3 中的空间对象如图 4 所示,$R\bowtie S$ 的结果分别为$(r_1,s_1),(r_1,s_2),(r_1,s_3)$,可以得到本次连接结果集中的对应 S 集合中的对象为 s_1,s_2,s_3,其边界 MBR 为图 4 中虚线所示,在与数据集 T 中对象进行连接运算时,可以直接过滤掉投影到网格单元 3 中的与该 MBR 不相交的空间对象 t_1,t_4,t_5,从而避免了这些空间对象分别与 s_1,s_2,s_3 进行连接运算,大副减少了计算代价.

2) 复制阶段过滤. 在多路连结查询处理过程中,需要对前 1 次连接处理之后的中间结果进行数据复制操作,将其复制到其他可能会产生连接结果

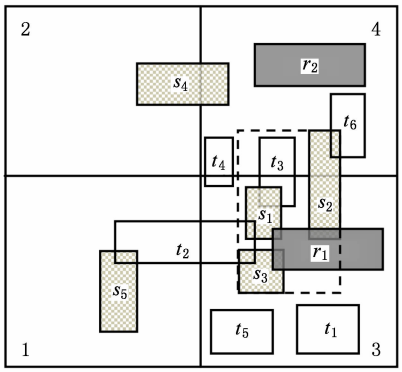

Fig. 4 An example of boundary filtering

图 4 边界过滤示例

的网格单元中,执行后续连接操作,避免丢失连接结果. 在对中间连接结果复制中,仅对涉及跨网格连接对象的中间结果进行复制,这样减少了数据复制和计算量,提高了系统的整体性能.

设 $C=(c_1,c_2,\cdots,c_n)$ 表示一个数据空间划分,若某个网格单元 $c_j\in C$ 上,其前 m 个数据集的连接结果集合 $S=R_1\bowtie R_2\bowtie\cdots\bowtie R_m$;则对于任意 $s_i\in S$, $s_i=(r_{1i},r_{2i},\cdots,r_{mi})$,若空间对象 $s.r_{mi}$ 与其他网格单元 c_k 存在交叠,则保留 s_i,并调用 $Replicate(s_i,c_k)$ 复制操作将其复制到 c_k 网格单元,并生成相应的键值对;否则将其过滤掉. 具体操作可以表示为

$$Filter(s_i,c_j)\rightarrow\{(s_i)\},\quad \forall k,s_i.r_{mi}\cap c_k\neq\varnothing\ c_k\neq c_j, \tag{5}$$

$$Replicate(s_i,c_k)\rightarrow\{(c_k,s_i)\},\quad \forall k,s_i.r_{mi}\cap c_k\neq\varnothing\ c_k\neq c_j. \tag{6}$$

在图 4 的示例中,网格单元 3 中数据集 R 和 S 的连接结果内仅有元组 (r_1,s_2) 中的 s_2 对象和网格单元 4 相交叠,因此仅将 (r_1,s_2) 复制到网格单元 4 中,以便与数据集 T 中的空间对象进行后续连接操作. 可见这种复制阶段的过滤策略能够减少中间数据的复制量,从整体上减少了系统的计算代价.

3.5 重复避免策略

在 Spark 环境下的多路空间连接查询处理中,数据被划分到多个网格单元中,进行并行处理. 由于在数据划分编码过程中,跨越多个分区的空间对象被投影到多个分区,并且对部分中间结果需要进行复制,如果不采取措施,就会导致多个网格单元输出相同的结果,这就需要进行去重操作,从而增加系统开销、降低了系统效率,因此需要进行重复避免. 在 BSMWSJ 算法中,采用了重复避免策略,仅让一个网格单元来负责输出结果,具体策略为在 2 个跨多个网格单元的空间对象进行连接时,仅让这 2 个空间对象相交叠而成的左下角交点所在的网格单元负责输出连接结果,这样就避免了结果的重复输出,减少了后续处理代价.

图 5 所示为重复避免的例子,其中集合 S 中的对象 s_1 被投影到其所交叠的网格单元 2,3,6,8,9,12,R 集合中的对象 r_1 则被投影到网格单元 3,6,9,12,r_2 对象被投影到了 4 个网格单元 8,9,10,11 中,如果不进行重复避免,在进行连接处理中,网格单元 3,6,9,12 就会输出相同的连接结果 (r_1,s_1),而网格单元 8 和 9 也会输出相同连接结果 (r_2,s_1),显然出现了重复.

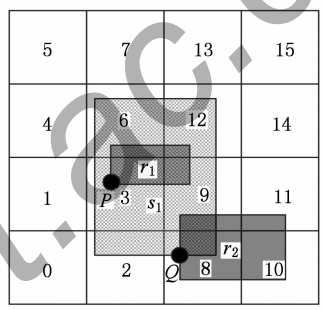

Fig. 5 An example of duplication avoidance

图 5 重复避免示例

根据所提出的重复避免策略,如图 5 所示,对象交叠部分所形成的对象的右下角(图 5 中点 P 和点 Q)所在的网格单元负责输出,即由网格单元 3 负责处理输出 r_1 和 s_1 的连接结果,网格单元 8 负责处理输出 r_2 和 s_1 的连接结果,显然该策略避免了重复处理和结果的重复输出,降低了计算代价.

3.6 多路空间连接查询处理算法

基于 Spark 分布式大数据处理框架,结合上述多路空间连接查询处理思路,设计实现了多路空间连接查询处理算法. 下面以 3 路空间连接查询处理为例来给出具体的多路空间连接查询处理算法,算法描述如算法 1.

算法 1. 多路空间连接查询处理算法(BSMWSJ).

输入:3 类待连接数据集、数据空间范围、分区数量和输出目录($dataSet_1$, $dataSet_2$, $dataSet_3$, $dataspaceRange$, $partitionNumber$, $outputFileDir$);

输出:连接结果集.

/* 投影操作函数定义 */

① def $projectOperation(mbr$: MBR, $extend$: MBR, $partitionNumber$: Int)={

② $ZOrder.getZOrder(mbr, dataspaceRange, PartitionNumber).map(splitNum\Rightarrow(splitNum, mbr))$};

```
/* 创建 RDD 函数定义 */
③ def createRdd(sc:SparkContext, filePath:
   String):RDD[(Int, MBR)]={
④  sc.textFile(filePath).map(line⇒{
⑤   val mbr=MBR(line.split(" "))
⑥   mbr.flatMap(cur⇒projectOperation(cur,
    dataspaceRange,partitionNumber))})};
/* 对 3 个数据集执行投影操作,创建 RDD */
⑦ RDD1=createRdd(sc,dataSet1);
⑧ RDD2=createRdd(sc,dataSet2);
⑨ RDD3=createRdd(sc,dataSet3);
⑩ RDDresult12=RDD1.cogroup(RDD2);
/* 执行聚合、过滤、连接和数据复制处理 */
⑪ result12=RDDresult12.flatMap(partition
   ⇒{
⑫   parExtend=getBound(partition._2._1);
⑬   filterSet=partition._2._2.filter(mbr
    ⇒mbrIntersect(mbr,parExtend));
⑭   result=For(i←filterSet;j←partition._
    2._1;If(isDuplicates(i,j,partition._1)
    && mbrIntersect(i,j)))yield(partition._
    1,(i,j));
⑮   result.flatMap(pairs⇒ZOrder.
    getZOrder(pairs._2._1,dataspaceRange,
    partitionNumber).map(zValue⇒
    (zValue,pairs._2)))});
/* 连接结果与第 3 个数据集进行连接处理 */
⑯ RDDresult123=RDDresult12.cogroup(RDD3).
   flatMap(partition⇒{
⑰   parExtend=getBound(partition._2._1);
⑱   filterSet=partition._2._2.filter(mbr
    ⇒mbrIntersect(mbr,parExtend));
⑲   For(i←filterSet;j←preSet;If
    (isDuplicates(i,j._1,partition._1)
    && mbrIntersect(i,j._1)))yield
    (partition._1,(i,j._1,j._2))});
  /* 保存最终连接结果 */
⑳ RDDresult123.saveAsTextFile(outputFileDir).
```

3.7 算法正确性分析

链式多路空间连接查询本质是一个迭代求解的处理过程,本文提出的 BSMWSJ 算法同样采用迭代方式来处理链式多路空间查询;然而 BSMWSJ 算法充分利用了 Spark 处理架构的并行处理特性,首先将各类数据集进行划分,然后在划分后的子空间中进行并行迭代连接处理,从而从总体上提高了多路空间连接查询的处理效率.下面以 3 路空间连接查询 $S=R_1 \bowtie R_2 \bowtie R_3$ 为例来说明 BSMWSJ 算法的正确性.

根据 BSMWSJ 算法的查询处理过程,首先将 R_1,R_2 和 R_3 数据集投影到各个网格单元,由于采用简单的投影策略,即只要某一空间对象 O_i 和某个网格单元 C_j 有交叠就将其投影到 C_j 中.因此只要 2 个空间对象 a 和 $b(a\in R_1, b\in R_2)$ 存在相互交叠,则 a 和 b 必然被投影到相同的一个或多个网格单元,由于采取重复避免策略,因此只由某个网格单元 C_i 负责进行连接计算,并输出连接结果 (a,b),同时 (a,b) 会被复制到与空间对象 b 有交叠的其他网格单元.在进行后续第 2 次连接运算中,元组 (a,b) 中的对象 b 又会和数据集 R_3 中空间对象进行空间连接运算,其执行过程与第 1 次空间连接类似,由于根据 R_2 集合中对象的交叠情况对第 1 次的连接结果进行了复制操作,因此不会发生丢解的情况,故本文提出的算法是正确的.

4 性能评价

为了验证本文所提出的多路空间连接查询处理算法的有效性,在真实数据集和合成数据集上做了一系列实验,并和当前最新的多路空间连接查询处理算法 ε-Controlled-Replicate 进行了比较分析.由于目前没有找到有关基于 Spark 平台的多路空间连接查询处理算法的研究工作,而 ε-Controlled-Replicate 算法的研究内容和目标与本文提出的算法最相似,但该算法是在 Hadoop 环境下实现,为此在 Spark 下重新实现了该算法,并和本文提出的算法进行了比较,下面就具体实验环境及结果对比情况进行详细说明.

4.1 实验环境

实验环境由 15 台 IBM PC 机架式服务器组成的 Spark 集群构成,其中 1 台为管理节点,其余为计算节点.每台服务器的配置为 E5-2620 CPU(6 核,2.0 GHz)、32 GB 内存和 6 TB 的硬盘,每台服务器都安装了 Centos6.4 系统和相应的 Spark 集群计算软件.

4.2 实验结果分析

本文采用真实数据和合成数据对算法的性能进行了测试,真实数据来自 Census2000 TIGER 地图文件数据集,其中道路数据的数量有 2 092 079 个,

水文数据的数量为 37 950 个. 合成数据由脚本生成,模拟真实数据分布(高斯分布),分别合成了3类数据集,3类数据集的大小相同,个数均为 250 万个空间对象,整体数据空间范围为 100 000×100 000,每个空间对象的最大 MBR 为 100×100. 本文首先就网格划分粒度、任务数量对 BSMWSJ 算法的影响进行了分析,之后与 ε-Controlled-Replicate 算法进行了比较,下面给出具体的实验结果.

1) 网格划分粒度对算法性能的影响

由于数据实际分布存在数据倾斜的现象,因此数据空间的不同划分粒度对算法的性能具有一定的影响,因此选择合适的划分粒度至关重要. 图 6 为 3 组数据集数据个数分别为 300 万、450 万和 600 万个空间对象,采用 BSMWSJ 算法执行 3 路空间连接时,其执行时间随划分粒度的变化情况.

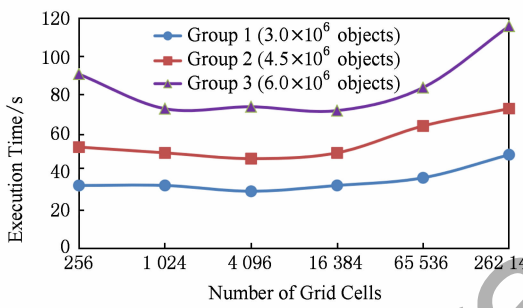

Fig. 6 Execution time of BSMWSJ with the number of grid cells

图 6 BSMWSJ 算法执行时间随网格单元数量变化情况

从图 6 可以看出,随着划分粒度的增大,连接查询执行时间逐渐变小,到一定程度后又开始增大. 这是因为划分粒度小时,由于存在数据倾斜导致数据分配不均匀,个别任务的运行时间较长,影响了整体的性能. 当划分粒度变大时,网格单元中的数据对象能够更加均匀地分配给任务去执行,因而时间减少,但随着划分的网格数量的进一步增加,就导致了跨网格单元对象越来越多,造成投影和复制的数据量大大增加,从而造成了计算量的增加,因此划分粒度要适合,这里选择划分 4 096 个网格单元为最佳选择.

2) 执行时间随任务数量的变化情况

Spark 环境下,通常任务数越多表示并行度越高,执行时间就越快. 图 7 是网格单元数为 64 的情况下,在 3 组不同大小的数据集(分别为 300 万、450 万和 600 万个空间对象)上分别执行 BSMWSJ 算法时,当并行任务数量不同时的算法执行时间变化情况.

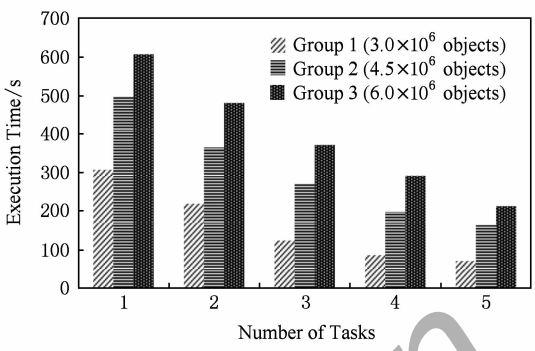

Fig. 7 Execution time of BSMWSJ with the number of tasks

图 7 BSMWSJ 算法执行时间随任务数量的变化情况

从图 7 可以看出,当数据量一定时,随着任务个数的增加,执行时间下降,但下降的幅度慢慢趋缓,到一定程度后,执行时间不再下降,这主要是由于任务的开启会带来一定的代价,增加任务的数量能够提高算法的并行度,降低查询的响应时间,但这种降低并不是线性的. 这也说明在数据集大小一定的情况下,任务的数量不一定越多越好,因此任务数量要适当.

3) 算法性能比较

图 8 是网格单元个数为 64、Spark 任务数为 64 时 2 种算法的执行时间随数据集大小变化的情况. 从中可以看出,2 种算法的执行时间都随着数据集数量的增加而增大,这和理论预期是一致的. 然而 BSMWSJ 算法明显优于 ε-Controlled-Replicate 算法,这是因为 BSMWSJ 算法在投影和复制操作中进行了相应的优化,在连接处理中采用了边界过滤方法进行过滤,减少了数据复制操作的数量,从而降低了实际计算代价.

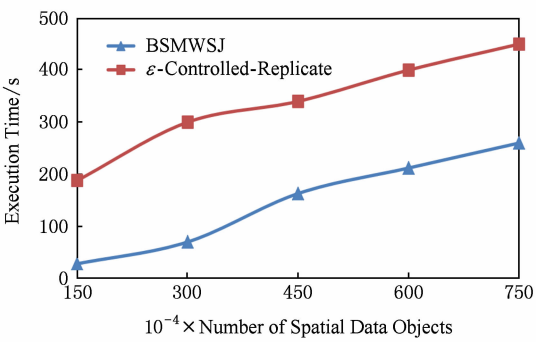

Fig. 8 Comparison of execution time with different dataset size

图 8 不同数据量下算法执行时间比较

图 9 是当空间数据对象大小增大情况下 2 种算法的执行时间变化情况比较,其中 3 类空间连接数据对象的数量分别为 200 万个,共计 600 万个空间

对象,划分的网格单元数量为 64 个,任务个数为 64,空间对象的最大 MBR 依次设置为 100×100 到 500×500.

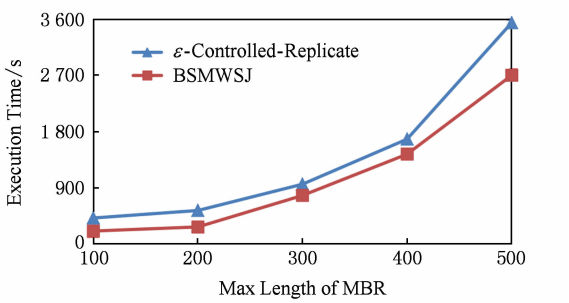

Fig. 9 Comparison of execution time with max length of MBR

图 9 不同 MBR 最大长度下的运行时间比较

从图 9 可以看出,2 种算法的执行时间随着空间对象 MBR 最大长度的增加而快速增加,这主要是由于当空间对象的 MBR 增大时,对象之间的交叠增加,投影和复制的数据对象就会越来越多,其计算量必然大幅增大,从而造成执行时间的大幅增加.

图 10 是在真实数据集上执行 3 路空间连接查询,3 个数据集中空间对象个数分别为 200 万、3.7 万和 200 万,网格单元个数为 64 时,2 种算法的执行时间随着任务数量变化的情况比较.

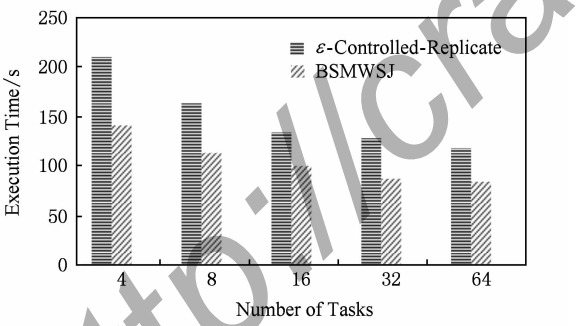

Fig. 10 Execution time of the algorithms varying with the number of tasks

图 10 算法执行时间随任务数量变化情况

从图 10 中可以看出,随着空间任务个数的增加,查询执行时间快速下降,但下降的幅度慢慢趋缓,到一定程度后执行时间不再下降,这主要是由于任务的开启会带来一定的代价造成的,这同理论分析相一致.但从 2 种算法执行时间比较来看,本文提出的 BSMWSJ 算法要优于 ε-Controlled-Replicate,这主要是由于 BSMWSJ 算法采取的数据投影方式避免了数据的大量复制,其边界过滤策略能够过滤掉一部分不会成为结果的对象,降低了计算和通信代价.

从 2 个方面的比较可以看出,BSMWSJ 算法的性能明显要高优于 ε-Controlled-Replicate 算法.

5 总 结

本文针对现有云环境下的多路空间连接查询处理算法存在的性能优化方面的不足,提出了一种基于 Spark 的多路空间连接算法 BSMWSJ,该算法采用网格划分方法对数据空间进行划分,并基于空间对象所在的位置来进行数据投影和复制,计算过程中采用边界过滤方法来过滤掉无用的连接对象,并通过缩小复制范围来减少连接对象的多余复制,从而减少算法的计算代价.实验表明:本文所提出的多路空间连接查询处理算法要明显优于 ε-Controlled-Replicate 算法,并具有良好的性能和扩展性.在后续工作中将进行更大规模的实验研究,并进一步改进相关算法,大幅提高数据投影、复制和过滤的效果,从而提高算法性能,同时也将考虑结合索引技术来进一步提高算法的性能.

参 考 文 献

[1] Apache. Apache Spark™ is a fast and general engine for large-scale data processing [EB/OL]. 2012 [2016-07-26]. http://spark.apache.org

[2] Afrati F N, Ullman J D. Optimizing multiway joins in a Map-Reduce environment [J]. IEEE Trans on Knowledge and Data Engineer, 2011, 23(9): 1282-1298

[3] Lin Yuting, Agrawal D, Chen Chun, et al. Llama: Leveraging columnar storage for scalable join processing in the MapReduce framework [C] //Proc of the 2011 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2011: 961-972

[4] Jiang Dawei, Tung A K H, Chen Gang. MAP-JOIN-REDUCE: Toward scalable and efficient data analysis on large clusters [J]. IEEE Trans on Knowledge and Data Engineering, 2011, 23(9): 1299-1311

[5] Wang Xiaojun, Sun Hui. Research of optimizing multiway joins based on MapReduce [J]. Computer Technology & Development, 2013, 23(6): 59-66 (in Chinese)
(王晓军,孙惠. 基于 MapReduce 的多路连接优化方法研究[J]. 计算机技术与发展, 2013, 23(6): 59-66)

[6] Slagter K, Hsu C, Chung Y, et al. SmartJoin: A network-aware multiway join for MapReduce [J]. Cluster Computing, 2014, 17(3): 629-641

[7] Wang Jingfen, Peng Zhixing. Research of optimization algorithm for multi-way spatial Join [J]. Journal of Chinese Computer Systems, 2013, 34(11): 2431-2436 (in Chinese)
(汪璟玢,彭志星. 多路空间连接优化算法研究[J]. 小微型计算机系统, 2013, 34(11): 2431-2436)

[8] Gupta H, Chawda B, Negi S, et al. Processing multi-way spatial joins on Map-Reduce [C] //Proc of the 16th Int Conf on Extending Database Technology. New York: ACM, 2013: 113-124

[9] Gupta H, Chawda B. ε-Controlled-Replicate: An improved controlled-replicate algorithm for multi-way spatial join processing on map-reduce [C] //Proc of the 15th Int Conf on Web Information Systems Engineering. Berlin: Springer, 2014: 278-293

[10] Luo Wuman, Tan Haoyu, Mao Huajian, et al. Efficient similarity joins on massive high-dimensional datasets using MapReduce [C] //Proc of the 13th IEEE Int Conf on Mobile Data Management. Piscataway, NJ: IEEE, 2012: 1-10

[11] Ma Youzhong, Meng Xiaofeng, Wang Shaoya. Parallel similarity joins on massive high-dimensional data using MapReduce [J]. Concurrency & Computation Practice & Experience, 2015, 28(1): 166-183

[12] Kim Y, Shim K. Parallel Top-k similarity join algorithms using MapReduce [C] //Proc of the 28th IEEE Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2012: 510-521

[13] Vernica R, Carey M, Li C. Efficient parallel set-similarity joins using MapReduce [C] //Proc of the 2010 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2010: 495-506

[14] Ma Youzhong, Meng Xiaofeng. Set similarity join on massive probabilistic data using MapReduce [J]. Distributed and Parallel Databases, 2014, 32(3): 447-464

[15] Okcan A, Riedewald M. Processing theta-joins using MapReduce [C] //Proc of the 2011 ACM SIGMOD Int Conf anaon Mgement of Data. New York: ACM, 2011: 949-960

[16] Pan Wei, Li Zhanhuai, Chen Qun, et al. An optimization for processing MapReduce-based asymmetric fragment and replicate join [J]. Journal of Computer Research and Development, 2012, 49(1): 296-302 (in Chinese)
(潘巍, 李战怀, 陈群, 等. 面向MapReduce的非对称分片复制连接算法优化技术研究[J]. 计算机研究与发展, 2012, 49(1): 296-302)

[17] Bian Haoqiong, Chen Yueguo, Du Xiaoyong, et al. Equi-join optimization on Spark [J]. Journal of East China Normal University: Natural Science, 2014, 2014(5): 263-280 (in Chinese)
(卞昊穹, 陈跃国, 杜小勇, 等. Spark上的等值连接优化[J]. 华东师范大学学报: 自然科学版, 2014, 2014(5): 263-280)

[18] Wang Kai, Han Jizhong, Tu Bibo, et al. Accelerating spatial data processing with mapreduce [C] //Proc of the 16th IEEE Int Conf on Parallel and Distributed Systems. Piscataway, NJ: IEEE, 2010: 229-236

[19] Zhang Shubin, Han Jizhong, Liu Zhiyong, et al. Spatial queries evaluation with MapReduce [C] //Proc of the 8th IEEE Int Conf on Grid and Cooperative Computing. Piscataway, NJ: IEEE, 2009: 287-292

[20] Zhang Shubin, Han Jizhong, Liu Zhiyong, et al. Sjmr: Parallelizing spatial join with MapReduce on clusters [C] //Proc of 2009 IEEE Int Conf on Cluster Computing and Workshops. Piscataway, NJ: IEEE, 2009

[21] Qiao Baiyou, Zhu Hunhai, Shen Muchuan, et al. A boundary filtering based spatial join query processing optimization algorithm [C] //Proc of the 12th Int Conf on Fuzzy Systems and Knowledge Discovery. Piscataway, NJ: IEEE, 2015: 1764-1769

[22] Sun Li, Li Jing, Liu Guohua. Join strategy optimization in column storage based query [J]. Journal of Computer Research and Development, 2013, 50(8): 1647-1656 (in Chinese)
(孙莉, 李静, 刘国华. 列存储数据查询中的连接策略优化方法[J]. 计算机研究与发展, 2013, 50(8): 1647-1656)

[23] Zhou Guliang, Sa Churila, Zhu Yongli. Star join algorithm based on multi-dimensional bloom filter in Spark [J]. Journal of Computer Applications, 2016, 36(2): 353-357 (in Chinese)
(周国亮, 萨初日拉, 朱永利. Spark环境下基于多维布隆过滤器的星型连接算法[J]. 计算机应用, 2016, 36(2): 353-357)

Qiao Baiyou, born in 1970. PhD and associate professor in Northeastern University. Member of CCF. His main research interests include cloud computing, virtualization technology, big data and spatial data management.

Zhu Junhai, born in 1989. Master. His main research interests include big data management and spatial data management.

Zheng Yujie, born in 1993. Master candidate. Her main research interests include big data management and spatial data management.

Shen Muchuan, born in 1992. Master. His main research interests include cloud computing, virtualization technology and big data.

Wang Guoren, born in 1966. Professor and PhD supervisor in Northeastern University. Senior member of CCF. His main research interests include cloud computing, big data, memory computing, and database theory.