开源关系数据库集群的并行空间连接算法实现①

范协裕[1], 任应超[2]

[1](福建农林大学 资源与环境学院, 福州 350002)

[2](中国科学院 遥感与数字地球研究所, 北京 100011)

摘 要: 当前对并行空间连接查询的研究主要集中在算法设计上, 缺少在并行关系数据库管理系统上的应用实现研究. 通过分析并行空间连接算法流程, 利用开源并行关系数据库集群项目 PL/Proxy, 提出了混合式计算迁移模式并扩展了对空间操作的支持, 并在其上实现了可扩展的基于空间划分的并行空间连接算法. 通过真实数据的实验表明: 设计实现的并行空间连接算法在空间数据划分负载均衡的情况下, 可实现近线性的加速比; 而在空间划分产生数据倾斜严重的情况下, 仍具有一定的加速比, 同时具备针对空间划分方案改进的可扩展能力. 算法的实现方式为进行并行空间数据管理研究提供了一种可行的解决方案.

关键词: 空间数据划分; 并行空间连接查询; 计算迁移; 并行关系数据库

Research and Realization on Parallel Spatial Join Query Algorithm Based on Open Source RDBMS Cluster

FAN Xie-YU[1], REN Ying-Chao[2]

[1](College of Resource and Environmental Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China)

[2](Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing 100011, China)

Abstract: Existing studies on parallel spatial join query mostly focus on algorithm process. Few of them pay enough attentions on implementation and application research. After analyzing existing algorithm process, parallel spatial join algorithms are divided into four phases which are parallel candidate tasks generating, assignment, executing and results collection. Each of them is designed and implemented on a parallel RDBMS cluster which is built on a open-source project named 'PL/Proxy'. In addition, to implement parallel computation, mixed computation migration method is proposed. Spatial extension function is implemented to support the spatial data sets operation on the cluster. Result of experiments using real data sets shows that, the implemented algorithm gains near linear speedup when data declustering scheme is optimal. Moreover, speedup is also gained while significant data skew caused by data declustering exists. The data declustering scheme is replaceable for improving the algorithm performance. A practicable solution for parallel spatial data sets management is provided in this paper.

Key words: spatial data partition; parallel spatial join query; computation migration; parallel RDBMS

随着企业 IT 数据急剧膨胀, 并行数据库管理系统(Parallel Database Management Systems, PDBMS)已成为管理海量数据的主要工具, 已经发展超过了 20 年[1]. 当前大量鲁棒的、高性能商用并行关系数据库管理着海量业务数据[2]. 空间数据库不仅要支持空间数据类型(Spatial Data Types, SDTs)、关系纪录包含几何和属性信息的统一框架, 同时必须提供高效的空间索引和空间操作算法, 比关系数据库更加复杂[3]. 主流的并行关系数据库平台及其算法无法直接应用到并行空间数据管理领域, 且目前尚未有满足复杂的大空间数据集分析的商业产品出现.

当前 MapReduce 编程模型受到广泛关注, 其在并

① 基金项目:国家高技术研究计划(863)(2013AA12A403)

收稿时间:2016-02-19;收到修改稿时间:2016-04-19 [doi:10.15888/j.cnki.csa.005462]

行空间数据分析领域也得到了广泛研究[4-7], 但其更多地应用在离线数据分析上, 不适用于实时的 OLTP(On line Transaction Process)事务. PostgreSQL 是当前被广泛应用的开源对象-关系型数据库管理系统, 也是目前功能最强大、特性最丰富的自由软件数据库系统. PostGIS 是 PostgreSQL 的一个遵循 OpenGIS 的规范空间扩展, 提供空间对象、空间索引、空间操作函数和空间操作符等空间数据管理支持. PL/Proxy 开源项目是基于 PostgreSQL 服务器端插件, 通过 PL/Proxy 可搭建基于关系表水平划分的并行 PostgreSQL 数据库集群[8], 为并行关系数据库的研究提供了一个良好的开源平台, 其目前尚未支持对空间数据的并行存储与管理.

空间连接算法是空间数据库中最复杂也是最具有代表性的算法, 诸如查询城市周边指定距离内的铁路、河流周边的工厂等极为常用的空间连接查询的具体应用, 对实时性要求很高. 并行空间连接算法主要是基于空间划分实现, 由划分、连接两个过程组成. 连接过程包含过滤、精炼两个步骤. 分为动态划分和静态划分两种. 动态划分算法中, 研究人员利用 R 树索引进行动态数据划分[9, 10]. 基于静态划分的算法中, Patel 等提出了基于划分的 Clone Join 和 Shadow Join 算法, 二者区别在于对待分片副本的策略上. 二者在连接阶段都使用 PBSM(Partition Based Spatial–Merge Join)算法. 实验证明, 与循环嵌套算法和基于 R 树的空间连接算法对比分析, 该算法具有一定优势[11]. 赵春宇博士等人提出的基于 HCMPR-tree 索引的连接采用了 PBSM 算法[12], X.Zhou 等人验证在空间连接中使用静态划分优于动态划分[13]. 以上主要针对算法进行研究, 缺少将其应用到空间数据库管理系统上的实现.

Postgres-XC 是基于 PostgreSQL 的集群项目, 该项目高效实现了部分并行查询, 并进行了空间扩展, 可处理空间数据, 但并未支持多表空间连接操作. 陈达伦等人设计了基于大规模并行处理架构的并行空间数据库中间件原型系统 GDMC, 该系统虽然实现了多表并行连接, 但在关键的空间划分上依然采用传统关系数据库使用的哈希数据划分方法[14]. Ray 等人设计了面向内存的并行空间连接算法[15], 不适用于并行空间数据库. BPSJ 是一种基于位图索引的并行连接算法框架, 虽然其运行效率很高, 但是查询结果不精确[16].

本文将首先对并行空间连接算法进行深入分析, 进而研究利用 PostgreSQL、PostGIS 和 PL/Proxy 开源项目实现并行空间连接算法需要的支撑技术, 在构建的开源并行数据库集群上设计并实现并行连接算法, 为并行空间数据管理提供一种可选的应用方案.

1　并行空间连接算法分析

在不考虑数据解域方案的情况下, 并行空间连接可具体归纳为四个步骤, 如下:

1) 创建候选任务集

在该阶段, 各个计算节点首先根据数据分布的情况, 确定候选集, 也称为粗过滤阶段. 候选集的确定方法和考虑因素有多种, 通常使用最小范围矩形作为空间对象的近似来减少运算量, 并应可装载入计算节点的内存. 另外, 存在如图 1 所示的情况是算法实现需要考虑的问题. 不同图层的空间对象在空间关系上是相邻或者相交的, 但却可能分别存储在不同的节点上.

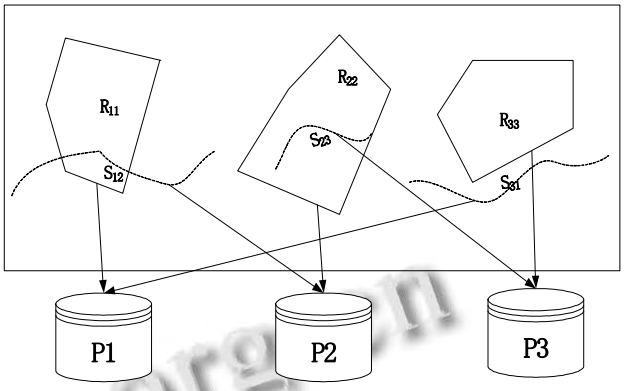

图 1　空间数据划分中空间对象分布情况

定义. 假定参与空间连接的两个关系表 R, S. R_{ij} 表示 R 的编号为 i 的分片, 该分片存储在节点 j 上, P 为在两个空间关系表上执行的操作谓词. 则定义候选集 $C_{lk} = \{<R_{ij}, S_{mn}, k, E, P>, k = \{j,n\}\}$, 表示编号为 l 且在节点 k 上执行的候选集, 其中 E 表示 R_{ij} 与 S_{mn} 相交的空间范围, P 表示空间分片 R_{ij} 与 S_{mn} 在计算节点 k 上执行的空间操作谓词.

因此, 图 1 所示分布情况可能创建的任务候选集如下: $<R_{11},S_{12}, 1, E,\theta>$、$<R_{11},S_{12}, 2, E,\theta>$、$<R_{22},S_{23}, 2, E,\theta>$和$<R_{22},S_{23}, 3, E,\theta>$.

2) 任务分配

无论是动态还是静态算法, 任务分配的过程主要是对各个候选方案的代价进行评估, 选择最优方案.

3) 并行执行阶段

动态算法在运行过程中要不断重新调整各个运算单元的任务，并重新分配. 静态算法只要并行执行过滤操作，最后将运算结果返回.

4) 结果收集和过滤

采用最小外包矩形代替空间对象进行空间划分的并行算法可能会产生冗余结果，需要对结果集进行去除重复元组处理再返回给客户端.

如图 2 所示，不妨假设各个关系表的空间数据已经以某种方式分散到各个节点上，如循环轮询的方式(Round Robin). 首先，任务创建过程在各个节点上并行执行以避免数据迁移，创建的过程中产生各个候选集；然后，根据候选集中的数据分片所在的节点，根据代价评估模型，选择最小代价的空间关系重分布方案，将候选任务集分配到各个计算节点上；在各个节点上执行空间操作；最后对运算结果进行过滤汇总工作.

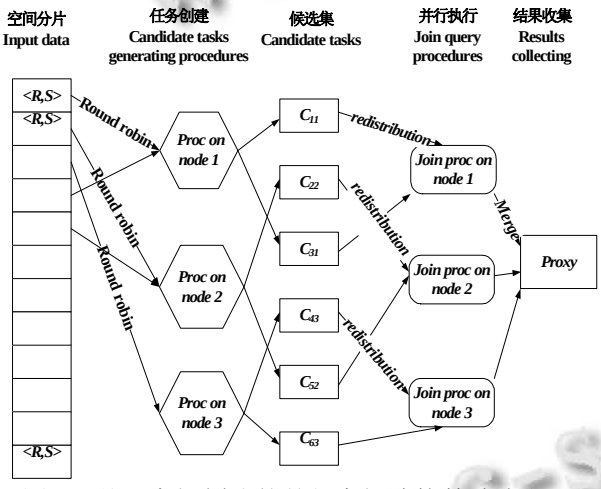

图 2 基于空间划分的并行空间连接算法流程

2 基于并行数据库集群的并行空间连接算法及实现

2.1 并行连接算法

如上一节所述，首先需要在各个节点上完成并行任务集创建、任务集重分配、并行连接以及结果过滤合并四步骤工作. 针对不同的空间操作找到一种最优的空间数据解域策略被证明是一个 NP 问题，X.Zhou 等人验证在空间连接中使用静态划分优于动态划分[13]. 因此，本文将介绍指定的静态划分策略下，基于集群的高效并行空间连接算法的实现.

首先在使用指定数据划分策略下，根据数据划分的情况构建记录数据分片分布的全局索引，并将其副本以关系表的形式存储于各个计算节点中，索引元组如下:

定义. 全局索引项 GIi= <lyr_name, frgt_no, i_node, extent, wk_load>，依次表示图层名，分片编号，所在计算节点编号，分片的空间范围和分片的负载.

算法 1 是构建候选集的过程也是粗过滤的过程. 其中输入参数中的全局空间索引在各个节点上存有备份. 使用分片的空间范围作为空间分片的近似代表，算法从全局空间索引中找出 R 与 S 的分片相交的分片作为任务候选集. 算法 AssignCandidateTasks 根据一定的规则进行分配得出任务集.

算法 1 R join S 并行任务集创建算法

GenerateJoinCandidateTasks

Input: $g_1, g_2 \ldots g_m$: tuples of global index(GI) //全局索引项
local_inode : id of local partition //计算节点编号
R : spatial relation R //空间关系表R
S : spatial relation S //空间关系表S
P: join predicate //连接谓词
//输出结果定义见前文 $C_{lk} = \{<R_{ij}, S_{mn}, k, E, P>, k = //\{j,n\}\}$
Output: $C_1, C_2 \ldots\ldots C_n$: set of subtasks C
Begin
for g_k k from 1 to m in GI ***do***
if (g_k.ly_name =R) ***then***
for g_h h from 1 to m in GI ***do***
if(g_h.ly_name = S) ***then***
if (***intersects***(g_k.extent, g_h.extent) ***then*** *// build candidate //subtasks*
extent:= ***intersection(g_k.extent, g_h.extent)***
i := g_k.frgt_no, j := g_k.i_node
m := g_h.frgt_no, n := g_h.i_node
C_t := <R_{ij}, S_{mn}, -1, extent, P> *//candiate subtasks*
push C_t into C
end if
done
end if
done
AssignCandidateTasks(C); *//assign subtasks*
End

并行空间连接查询的时间代价由三部分组成，即 CPU 的处理时间、I/O 消耗和空间分片重分布延时，如下:

$$t = t_c + t_{io} + t_{trans}$$

对于分配任务来说，I/O 消耗是最重要的[17]. 可以根据情况选择不同候选集任务分配算法，如启发式算法和更加复杂的代价估计模型[18]. 由于 GenerateJoinCandidateTasks 在所有节点上都执行，并且使用相同的元数据，因此产生相同的候选集. 算法只在 R 各分片所在的节点上进行候选集分配，避免产生重复的任务.

算法 3 展示空间数据重分布与并行连接阶段的伪代码. 在该阶段中，各个节点根据第一阶段生成的候选任务集，进行数据重分布. 然后将分配到本节点的空间数据分片存储在缓冲区上，并建立本地空间索引. 最后进行空间连接操作，返回结果. 由于第一阶段的产生的候选集已将无相交区域的空间关系的分片过滤掉，因此连接算法可以在各个计算节点上并行运行.

算法 3 计算节点上的连接算法

```
SpatialJoin
Input:  C1, C2……Cend: subtasks,
Ch=<Rij,Smn, k, E, θ>
local_inode : id of local partition
Ch.Rij : fragment of R in subtask h
Ch.Smn : fragment of S in subtask h
Ch.k : target node id of subtask h
Output:  Res1, Res2….Resn: local results of R join S
Begin
for Ch h from 1 to end do
if( Ch.k = local_inode) then
if( Rij.j = k) then    //get R fragment locally
        tuples ResR := GetLocalFragment(R, E)
  else
//get R fragment intersecting with E from remote
//partition
        tuples ResR := GetRemoteFragment(R, k,E)
 end if
 if( ResR != NULL) then
          insert ResR in to local buffer;
          create spatial index for ResR in buffer
        end if
if( Smn.n = k) then
        tuples ResS := GetLocalFragment(S, E)
else
        tuples ResS := GetRemoteFragment(S, k, E)
end if
 if( ResS != NULL) then
          insert ResS in to local buffer;
          create sptial index for ResS in buffer
end if
if(ResS !=NULL and ResR !=NULL) then
     //join relations and return results
          Res :=  ResR joins ResS
end if
  End for
End
```

2.2 基于 **PL/Proxy** 集群的并行空间连接算法的实现

PL/Proxy 开源项目是基于 PostgreSQL 服务器端插件，通过编写基于 PL/Proxy 语言的函数可实现基于 PostgreSQL 并行数据库集群的存储和查询操作. 集群主节点是一台安装了 PL/Proxy 语言的 PostgreSQL 服务器的代理节点. 代理根据指定的方式分发客户端的请求到集群各个节点中，在各个数据节点上执行客户端请求的 SQL 命令或与代理节点的 PL/Proxy 函数同名的函数，最后汇集各个数据节点返回的结果返回给客户端. 客户端使用 PL/Proxy 语言编写的函数来进行数据访问和操作，并对 PL/Proxy 进行空间查询扩展(Spatial SQL, SSQL), SSQL 主要是对重复数据进行重复去除处理及对空间操作的范围提供限定[19].

计算迁移是当前云计算平台最大的优势，相对于数据迁移，将计算功能动态迁移到数据存储节点并进行计算，有效减少了网络传输的时间消耗. 当前实现计算迁移的方案有两种，一种是可执行程序的迁移，如果不考虑虚拟机执行 Java 可执行程序的特殊性，可将 Hadoop 平台上的计算迁移看作是这类方案的实现；另一种解释性脚本的迁移，非关系数据存储平台 Riak 使用 Javascript 解释脚本实现计算迁移的方式即属于这种方案.

为了实现任务重分布以及并行计算，本文采用 SQL 与数据库平台支持的嵌入式查询语言相结合的混合方式. 如图 3 所示，远程执行的逻辑代码由两方面组成: 一方面通过远程连接数据库，发送 SQL 脚本到

远端的执行节点，由执行节点执行来实现；另一方面根据业务需求，在各个节点上部署由 SQL 及其他复杂查询语言实现的服务器端存储过程来实现．由于每个节点都是具有完全功能的空间数据库，因此，通过在各个节点上直接部署空间存储过程代码，完成对功能逻辑代码的解释执行．

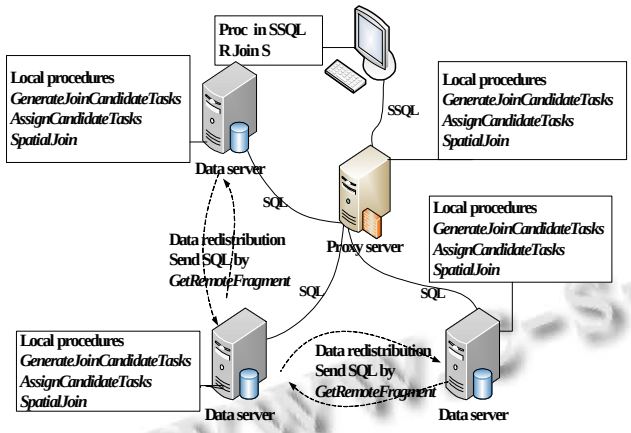

图 3 混合计算迁移方案

如图 3 所示，任务候选集需要根据代价评估模型分发任务，进而将参与运算的关系表中的数据分片拷贝到计算节点的缓冲区中，该过程在并行关系数据库中的并行连接查询中称为重分布(Data redistribution)．本文开发了 PostgresSQL 节点间数据传输的服务端器端函数． 不妨假设，需要将候选集 Cn=<Rij,Smn, n, E, θ>中的空间数据分片 Rij 中与 Smn 相交部分的元组远程传输当前节点上，以便进行下一步计算，数据传输函数的算法如下．

算法 4 服务器端函数 - 节点间空间数据传输

节点间空间数据传输 ***GetRemoteFragment***

Input: R_{ij}: fragment i of relation R on partition j
extent: extent of R_{ij} intersecting with S_{mn}
Output: SR_k : result tuples of intersection of R_{ij} and S_{mn}
Begin
//Get connection string of node j by querying meta
//data using SPI procedure
connStr = ***GetConnString***(j);
conn = ***Connect***(connStr); // *connect to node n*
Status status = ***SendQuery***(
conn, "*Select * from R where st_intersect(geom, extent)*");
if (status != OK) ***return*** NULL;
While(1) ***do***
if (***isBusy***(conn))
break;
if (! SR_k = ***get_another_result***(conn))
break;
done
End

获取数据的流程包括两个步骤：首先根据编号获取远程节点的连接字符串，主要是通过使用 SPI 接口查询集群元数据信息实现；然后，使用 libpq 客户端编程接口与远程节点建立连接，将数据查询的 SQL 脚本发送至目标计算节点执行，并收集获取的结果元组集．

保证跨分区的线和面空间对象在空间查询时的准确性是基于范围划分必须处理的关健问题之一，在多个分区上拷贝跨分区的空间对象是最为高效的方法[15]．副本拷贝会导致查询结果可能包含重复元组的问题．针对该问题，本文采用在不同分区上存储副本，并通并在数据汇集节点使用动态 Hash 表对重复结果进行去重的方法，该方法通过在动态 Hash 函数中存储各个返回元组的标识达到快速去重的效果[16]．

3 实验与分析

3.1 实验环境

实验使用 9 个安装了 64 位 CentOS5.5 操作系统的节点，各节点配置一致(4GB 内存，2.49GHz 双核处理器)．其中一台作为代理节点安装有扩展自 PL/Proxy 的并行空间查询语言(SSQL)，其余的均为数据节点，每台节点上均安装有 PostgresSQL 8.4.4+PostGIS1.5 作为空间数据库的服务器节点(IP 为 192.168.111. 150-158)．

表 1 实验数据集

数据集名称	数据类型	记录数
铁路	线	62391
城市	点	3542

实验采用 Hilbert+Round Robin 数据划分方案[12]，使用数据量大小做为负载均衡度量，在此基础上，虽然各个节点在数据量大小上趋于平均，空间线对象可能在数据集分片的记录数上产生严重的倾斜现象．算法首先根据设定的 Hilbert 曲线阶数将数据集划分成等面积的分片，并根据 Hilbert 编码对分片进行排序，再根据平均负载参数，将若干编码相邻的分片组成负载均衡的数据分区，最后使用 Round Robin 方法将分区

依次分配到不同的节点上. 每个分片内可包含多个空间对象, 空间对象的 Hilbert 编码计算采用其中心点.

在实验中, 使用 PostgreSQL 的临时表实现算法的缓冲池来存储临时数据, 临时表大小的上限可在服务器的配置文件中设置. 各个节点上都部署了前文所述的获取远程节点数据的函数(GetRemoteFragment 函数). 空间连接算法做为存储过程部署在各个节点上. 客户端通过调用部署在代理节点上的同名函数来调用连表查询, 代理节点负责调用各个节点的同名函数实现连接查询.

3.2 实验及结果分析

设计并进行了两组实验如下:

(1) 实验一

① 实验设置

使用“铁路”和“城市”数据集, 选择“城市中”10km 范围内有铁路经过的城市. 使用 Hilbert 曲线对“铁路”数据进行静态划分. 集群数据节点数为 4, 并将城市节点数据迁移到各个计算节点上.

② 实验结果

在单个节点上执行运算, 耗时 39.8s;

集群运行时间 12.1s;

集群中耗时最长数据节点耗时 11.7s;

并行加速比 39.8/12.1=3.29.

(2) 实验二

① 实验设置

使用“铁路”和“城市”数据集, 选择“城市中”1km 范围内有铁路经过的城市. 分别使用 Hilbert 曲线对“铁路”和“城市”数据进行静态划分. 集群数据节点数为 8.

② 实验结果

在单个节点上执行运算, 耗时 250.8s;

集群算法运行时间 140.7s;

集群中耗时最长数据节点耗时 139.5s;

并行加速比 250.8/140.7=1.78.

实验一中“城市”数据集在各个节点上有备份, 不需要进行重分布, 且对“铁路”的 Hilbert 划分产生的结果较为理想, 各分片的数据量大小相差较小的情况下, 节点间分片的记录数相差也较小, 在这种理想情况下, 并行系统达到了近线性加速比.

实验二的两个数据集各自进行空间划分, 采用数据量大小做为负载均衡的度量, “铁路”数据集在划分时, 由于线空间对象的复杂度不一, 虽然各节点数据量大小趋于一致, 但在数据的记录数上产生了很大的差异. 数据集记录数最小与最大的有接近 5 倍的差距(各节点上有跨分区副本的情况下最少的仅有 10900 条数据, 而最多的达到 51727 条数据). 可见在以数据量大小为负载度量时, 数据划分结果是负载均衡的, 但是以记录数为度量时, 产生了严重的负载倾斜, 导致了整体查询时间受到耗时最长的节点的限制. 这方面, 可通过使用动态负载均衡算法进行改进, 也可在静态算法的前期进行改进. 即使如此, 算法仍获得了一定的加速比.

综上, 在既有的数据划分方案下, 当数据分布理想的情况下, 平台算法可获取近线性的加速比, 即使在不理想的数据分布情况下, 仍获得了一定的加速比. 另外, 平台还为采用更加复杂的动态负载均衡算法提供了可扩展的空间, 只需替换任务分配函数即可.

4 结论

通过在 PL/Proxy 构建的并行 PostgreSQL 关系数据库集群上实现计算迁移、空间扩展等关键技术, 在集群上实现了并行空间连接查询算法. 实验表明在不同的数据划分情况下, 实现的算法都取得一定的加速比, 具备可扩展性, 为并行空间查询研究提供了一个可行的实现方案.

参考文献

1 Özsu MT, Valduriez P. Distributed and parallel database systems. ACM Computing Surveys (CSUR), 1996, 28(1): 125–128.

2 Andrew P, Erik P, Alexander R. A comparison of approaches to large-scale data analysis. SIGMOD. Rhode Island, USA. ACM. 2009.

3 Shekhar S, Chawla S.谢昆青译.空间数据库.北京:机械工业出版社,2004.

4 Zhong YQ, Han JZ, Zhang TY, Li ZH, Fang JY, Chen GH. Towards parallel spatial query processing for big spatial data. 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). 2012

5 张书彬,韩冀中,刘志勇,王凯.基于 MapReduce 实现空间查询的研究.高技术通讯,2010,(7):719–726.

6 Cary A, Yesha Y, Adjouadi M, Rishe N. Leveraging cloud computing in Geodatabase Management. IEEE International Conference on Granular Computing. San Jose, CA. IEEE. 2010. 73–78.

7 Nathan TK. Alternative Approaches to Parallel GIS Processing [M.S. Thesis]. Arizona, USA: Arizona State University, 2007.

8 pgFoundry. PgFoundry:PL/Proxy:Project Info. http://pgfoundry.org/projects/plproxy/. 2012.

9 Brinkhoff T, Kriegel HP, Seeger B. Efficient processing of spatial joins using R-trees. ACM, 1993, 22.

10 Mutenda L, Kitsuregawa M. Parallel R-tree spatial join for a shared-nothing architecture. Proc. 1999 International Symposium on Database Applications in Non-Traditional Environments, 1999. (DANTE '99). IEEE, 1999. 423–430.

11 Patel JM, Dewitt DJ. Clone join and shadow join: two parallel spatial join algorithms. Proc. of the 8th ACM International Symposium on Advances in Geographic Information Systems. ACM. 2015. 54–61.

12 赵春宇.高性能并行 GIS 中矢量空间数据存取与处理关键技术研究[博士学位论文].武汉:武汉大学,2006.

13 Zhou X, Abel DJ, Truffet D. Data partitioning for parallel spatial join processing. Geoinformatica, 1997, 1262(2): 178–196.

14 陈达伦,陈荣国,谢炯.基于 MPP 架构的并行空间数据库原型系统的设计与实现.地球信息科学学报,2016,18(2): 151–159.

15 Ray S, Simion B, Brown AD, Johnson R. Skew-resistant parallel in-memory spatial join. SSDBM'14 ACM. New York, USA. 2014.

16 Shohdy S, Yu S, Agrawal G. Load balancing and accelerating parallel spatial join operations using bitmap indexing. IEEE International Conference on High Performance Computing. 2016. 396–405.

17 刘宇,孙莉,田永青.并行空间连接查询处理.上海交通大学学报,2002,(4):512–515.

18 Shekhar S, Ravada S, Kumar V, et al. Declustering and load-balancing methods for parallelizing geographic information systems. IEEE Trans. on Knowledge & Data Engineering, 1998, 10(4): 632–655.

19 范协裕,任应超,邓富亮,等.基于代理的并行空间查询语言.计算机工程,2013,11(11):61–64.