基于R-树索引的Map-Reduce空间连接聚集操作*

刘 义,陈 荦,景 宁,熊 伟
(国防科技大学 电子科学与工程学院,湖南 长沙 410073)

摘 要:空间连接聚集是一种常用并且非常耗时的空间数据库操作,特别是在面对大规模空间数据集时,单机运行环境难以满足其对时空开销的需求,如何设计高效的面向云计算环境中的分布式空间连接聚集算法越来越受到人们关注。Map-Reduce作为云计算的核心模式受限于其扁平化的串行扫描操作模型,常被用来加速非索引的空间连接操作,现有工作尚无将Map-Reduce和R-树索引结合来处理空间连接聚集。因此,提出了基于R-树索引的Map-Reduce空间连接聚集算法(RSJA-MR)来更高效地返回连接聚集结果。提出一种分布式R-树索引结构以支持大规模空间数据的索引,RSJA-MR算法利用分布式R-树生成任务集,任务集的执行满足无依赖并行计算模式,很容易在Map-Reduce框架中进行表达。文中提出一种实时缓存策略以支持索引并发访问。实验结果表明:相比非索引的Map-Reduce连接聚集算法,在空间交叠连接聚集查询上,时间性能最少提升8%,在空间包含连接聚集查询上,时间性能最少提升近35%。

关键词:云计算;Map-Reduce;空间连接聚集;R-树
中图分类号: TP791 **文献标志码**: A **文章编号**:1001-2486(2013)01-0136-06

Processing spatial join aggregate in Map-Reduce based on R-tree

LIU Yi, *CHEN Luo*, *JING Ning*, *XIONG Wei*
(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: Spatial join aggregate (SJA) is a commonly used but time-consuming operation in spatial database. Especially when faced with the deluge of spatial data, SJA is difficult to be implemented on a single machine. Consequentially, how to design efficient distributed SJA algorithms is receiving more and more attention. Constrained by the sequential scan operation assumption, Map-Reduce is usually used to accelerate the non-indexed spatial join query, but none of the previous work can process SJA with both Map-Reduce and R-tree spatial index. Thus, a novel algorithm, R-tree based Spatial Join Aggregate with Map-Reduce (RSJA-MR) was proposed, which is able to return results more efficiently. A distributed R-tree index structure was presented to index the large-scale spatial data. RSJA-MR first made use of distributed R-tree to generate the tasks. Those tasks met independent parallel computation and could easily be expressed in Map-Reduce. An index cache mechanism was provided to support the concurrent access of R-tree index. The experiment results show that, compared with the non-indexed SJA , the time performance of RSJA-MR is improved at least by 8% for spatial intersection join aggregate and by 35% for spatial containment join aggregate.

Key words: cloud computing; Map-Reduce; Spatial Join Aggregate; R-tree

对地观测、GIS和传感器网络等技术的发展,极大地丰富了空间数据的产生渠道,给空间数据的查询操作提出了更严峻的挑战。空间连接聚集(Spatial Join Aggregate,SJA)是一种常用并且非常重要的空间数据库操作,同时也是一种非常耗时的操作,需要在大量空间数据上执行昂贵的连接和聚集操作。相比空间连接,返回连接结果统计信息的空间连接聚集查询更受用户欢迎。例如,在交通监控系统中,空间包含连接查询结果(寻找各个交通区域内的车辆对象)是毫无意义的(由于车辆对象的频繁移动性),而统计各个交通区域内的车辆数量更有助于进行交通调度。为提高空间连接聚集查询性能,研究人员已提出许多有效的算法[1-2]。然而,这些算法着眼于单机运行环境的单线程执行,随着数据规模的快速增长,单机环境难以满足其对时空开销的需求。

Map-Reduce[3]是Google提出的一种集群上处理超大规模数据集的分布式并行编程模型,也是目前云计算的核心计算模式。随着空间数据的急剧增长,设计云计算环境下的分布式空间查询算法越来越受到人们关注。尽管Map-Reduce框架中的空间连接[4]、kNN空间连接[5],ANN空间连接[6]以及Top-k空间连接[7]等算法已被设计和实现,然而,受限于Map-Reduce的无依赖并行计

* **收稿日期**:2012-07-09
基金项目:国家863计划项目(2011AA12A306);国家自然科学基金资助项目(61070035)
作者简介:刘义(1979—),男,湖南华容人,博士研究生,E-mail:liu.yi.nudt@gmail.com;
景宁(通信作者),男,教授,博士,博士生导师,E-mail:jingning@nudt.edu.cn

算模型和串行扫描操作,这些算法都关注于加速非索引的空间连接操作。尽管 MPI 环境中基于 R-树[8]的并行连接算法[9]已得到研究,然而这些算法并没有针对大数据集的连接查询问题,如何利用 Map-Reduce 加速大数据集下基于 R-树索引的空间连接查询,更具有实际的价值。

基于上述认识,研究 R-树索引下的 Map-Reduce 空间连接聚集查询,提出一种云计算环境下的分布式 R-树索引结构以支持大规模空间数据的索引和连接聚集任务的快速生成。着重设计了 Map-Reduce 框架下基于 R-树索引的并行连接聚集算法,及该算法上的索引并发访问优化机制。

1　基本概念

1.1　空间连接聚集

连接元组对:给定空间关系 R 和 S, geometry 描述空间对象的空间属性,(r,s) 是 $R \triangleright \triangleleft S$ 的连接元组对,满足条件:1) $r \in R, s \in S$;2) $r.geometry$ 与 $s.geometry$ 交叠为真。

连接聚集组:设 $O'(t) = \{o_1, o_2, \cdots, o_n\}$ 是元组 t 的连接聚集组,满足条件:1) $t \in R, O'(t) \subseteq S$ 或 $t \in S, O'(t) \subseteq R$; 2) $\forall o \in O'(t)$, (t,o) 是连接元组对。

假设 R 和 S 是参与连接操作的空间关系,空间连接聚集 SJA 检索 R 中任一对象的连接聚集组中对象数目(或基于对象属性上的总和、平均值、最大值和最小值)聚集信息,表示为:

$$SJA(R,S) = \{(t, op(O'(t))) \mid \text{for all } t \in R\}$$

其中 op 表示聚集函数[10],本文以对象数目为例来讨论 SJA 操作,其他聚集操作可类似处理。

1.2　Map-Reduce 并行计算抽象

Map-Reduce 编程模型通过采用一种无依赖并行和串行同步(Independent Parallel and Serial Synchronization, IPSS)的计算抽象,简化了集群上大规模数据的处理。正是利用这种限制性计算抽象,Map-Reduce 实现了程序的自动并行处理,并在内部提供诸如输入数据划分、并行任务调度和通信、容错、负载均衡等并行分布式编程细节的自动实现,实现高可扩展性和高度并行性。

无依赖并行计算:并行计算中,给定一个作业 J,其可分解为一系列可异步并行执行的任务 T,指定一个并行任务间的通信代价 C,如果对于输入任务 T 的任一划分 $\forall T = \{t_1, t_1, \cdots, t_m\}$,若其满足 $t_i \cap t_j = \emptyset$ 以及 $C(t_i, t_j) = 0 (1 < m < |T|, 1 < i, j < m)$ 条件时,作业 J 可表示为 $J = \sum_{i=1}^{m} T(t_i)$,称满足以上条件的并行计算为无依赖并行计算。

串行同步计算:给定两个无依赖并行计算作业 J_1 和 J_2,若 J_1 和 J_2 存在一个同步和通信过程,标识为 $J_1 \rightarrow_{out} d \rightarrow J_2$,即 J_1 的数据输出为 J_2 的输入,并且 J_2 必须等到 J_1 执行完毕后才开始执行,称作业 J_1 和 J_2 之间是串行同步计算。

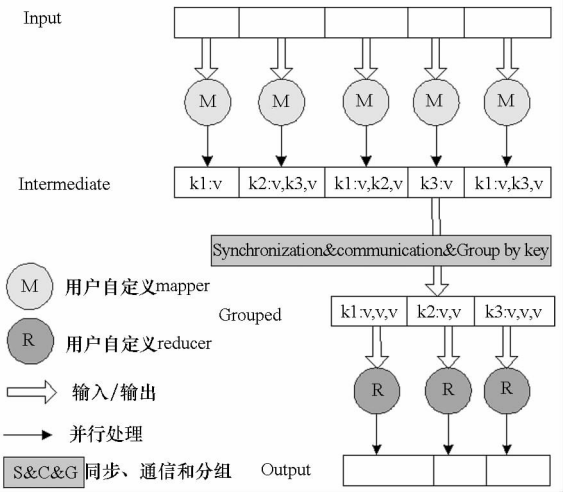

图 1　并行编程模型

Fig. 1　Parallel programming model

Map-Reduce 模型是典型的无依赖并行和串行同步的并行计算抽象。如图 1 所示,Map 和 Reduce 阶段内部的若干并行任务可以在互斥的任务划分上无通信代价地独立执行。即 Map 和 Reduce 过程(称为一次 MR 过程)中的每个阶段内部的异步并行任务($Map_0 \sim Map_m$ 或者 $Reduce_0 \sim Reduce_r$)都运行在无依赖并行的理想状态下。同时,Map 阶段和 Reduce 阶段之间是串行同步的,二者之间存在一个相对用户透明的隐式同步和通信过程。Reduce 必须等到最后一个 Map 执行完毕后才开始执行。目前,不少机构都研发基于 Map-Reduce 模型的海量数据并行系统,其中 Apache 的 Hadoop[11]是 Map-Reduce 的开源实现,也是目前学术界和工业界事实上的海量数据并行处理标准,本文采用 Hadoop 作为并行计算框架。

2　基于 R-树索引的 Map-Reduce 空间连接聚集

2.1　分布式 R-树索引结构

分布式 R 树索引结构借鉴 Master-client 分布式 R-树[12]索引结构的思想,根节点存储各个子树索引文件地址及最小外包框(MBR),不同于 Master-client 架构 R-树索引的存储架构,根节点及各个子树均存储在 Hadoop 的分布式文件系统 HDFS 上,其结构如图 2 所示。

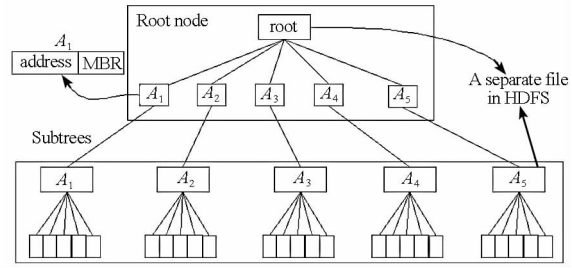

图2 分布式R树索引结构

Fig. 2 Distributed R-tree index

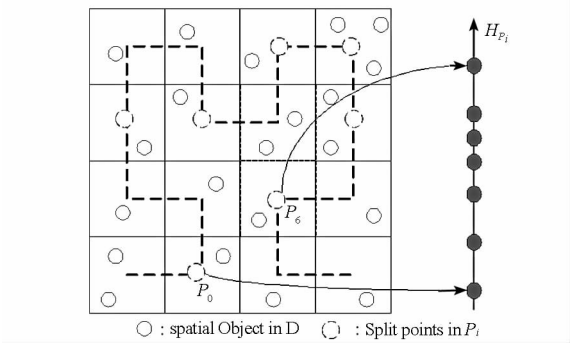

图3 切割点生成

Fig. 3 Generating split point

分布式R-树索引构建的基本思想:利用抽样技术,从海量的空间数据中并行抽样形成抽样点,并将抽样点映射到Hilbert曲线上,排序形成一个样本点,然后从样本点中依据划分数均匀提取切割点,划分函数依据切割点将空间对象分配到不同划分,每个划分采用自底向上方法依次生成子树。切割点的生成如图3所示,形成切割点后,空间划分函数变为:

$$f(o)=\begin{cases}1, & H(o)\leqslant H'[1]\\ j, & H'[j-1]<H(o)\leqslant H'[j], j=2,\cdots,P-1\\ P, & H(o)>H'[P-1]\end{cases}$$

形成划分函数后,每个对象依据划分函数$f(o)$分划到P个不同分区,每个分区构建一颗子树,其具体的Map-Reduce算法参考文献[13]。

2.2 并行任务生成

并行任务生成在Master节点提交作业前完成,任务生成描述如下:给定空间关系R和S,其构建的分布式R-树索引分别为A和B,同步扫描根节点,若MBR交叠,则生成一个任务。任务表示为一个三元组$T=\{A_i,B_j,BBox_{ij}\}$。其中A_i和B_j分别表示参与连接的子树索引地址,其交叠范围为$BBox_{ij}$。图4描述了任务生成过程。由于任务生成仅扫描A和B的根节点,若A节点容量为N_A,B节点容量为N_B,则任务生成空间复杂度仅为$N_A\cdot N_B$,因此,可在作业提交前高效完成。

上述的任务生成适合并行的空间连接查询,

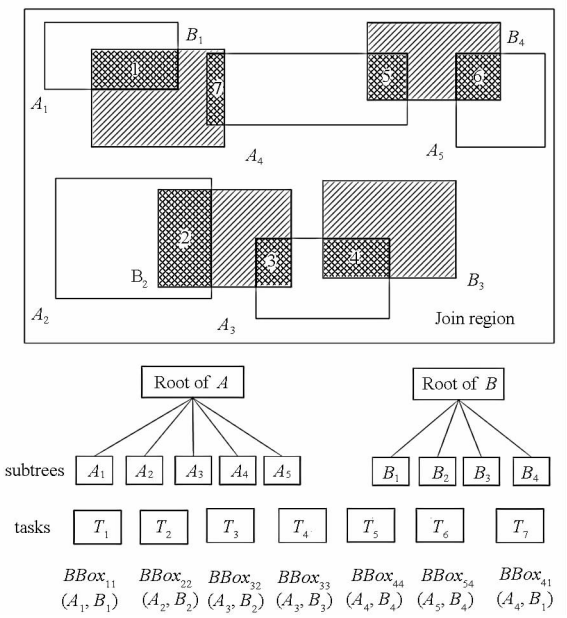

图4 任务生成

Fig. 4 Generating tasks

应用于空间连接聚集查询时,会导致连接聚集结果分散在各个子任务中,需要进一步对结果进行聚集。如图4所示,A_4中索引的空间对象的连接聚集结果会分散在任务(A_4,B_4)和(A_4,B_1)中。此外,每个任务需要对索引的每次出现进行访问,其访问代价可表述为$\sum|A_i,B_j|,(A_i,B_j)\in T$,$T$为生成的任务列表。当空间连接并行任务生成时,两个空间关系间的任务可以构成一个任务连接图(Task Connection Graph, TCG)。TCG $T_G=(V_A,V_B,E)$是一个双向图,其中顶点集V_A表示来自A的子树索引集合,顶点集V_B表示来自B的子树索引集合。边的集合构造如下:当一个边加入到V_A的结点V_A^i和V_B的结点V_B^j之间,当且仅当任务集中有一对子树索引对(A_i,B_j)交叠,其中$A_i\in V_A$, $B_j\in V_B$。即边值t_{ij}表示子树索引对(A_i,B_j)的交叠范围$BBox_{ij}$。图5表示一个从图4中的任务划分中得到的任务连通图。

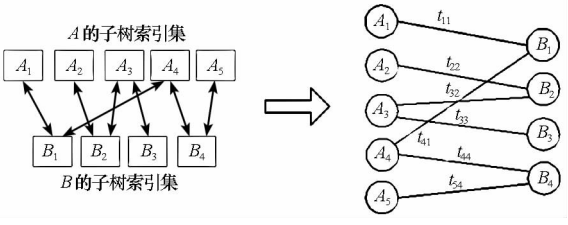

图5 依据任务划分建立TCG

Fig. 5 Building TCG according to task partition

基于TCG图,若以A为内关系,B为外关系,即计算A中索引的所有对象的连接聚集值,任务可表述为$T=\sum\{A_i,[(B_j,BBox_{ij})]\}$。基于分布式R-树索引的空间连接聚集SJA$(A,B)$可分

解为:

$$SJA(A,B) = \sum SJA(A_i,[B_j,BBox_{ij}]),A_i \in A,$$
$$B_j \in B,A_i \cap B_j = BBox_{ij}$$

上述公式表明,SJA(A,B)可分解为无依赖并行计算的多个子任务,即 A 中每个子树索引与 B 中的子树索引集进行连接聚集操作是无依赖的,因此,可容易地用 Map-Reduce 模型进行表达。特别地,在此任务划分下,索引访问代价变为$|A_i| + \sum|B_j|$,$(A_i,B_j) \in T$,即 A_i 索引只需访问一次。

2.3　RSJA-MR 算法

RSJA-MR 算法的过程分为三个阶段:任务生成、任务分发和并行任务执行,这三个阶段分别对应 Map-Reduce 过程中作业 Job 提交前的预处理、Map 阶段的任务分发和 Reduce 阶段的任务执行。

2.3.1　基本算法

基本算法基于上述的分布式索引结构和任务划分方法,进一步提出聚集查询算法。相比于 Reduce 阶段的任务执行,任务生成和任务分发的计算成本和数据通信代价非常小,所以,任务生成为串行执行,而任务分发仅在一个 Map 内执行,相反,Reducer 数 P 则限定了 RSJA-MR 的并行执行程度。

RSJA-MR 基本算法的详细步骤描述如下:

算法 1: RSJA-MR
Input: A: The distributed R-tree index of R
　　B: The distributed R-tree index of S
Output: A set of (object, count) pairs
TaskCreate(Rtree A,Rtree B) At the Job
1. readRootNode(A_{root},B_{root})
2. foreach $A_i \in A_{root}$ do
3.　　foreach $B_j \in B_{root}$ do
4.　　if $A_i \cap B_j \neq \emptyset$ do
5.　　add (A_i,B_j,$BBox_{ij}$) to tasklist
TaskAssignment(tasklist tl) At the Map
6. foreach task $t \in tl$ the Map *do*
7.　　output(key $t.A_i$,value $t.B_j$,$t.BBox_{ij}$)
RSJA(A_i,[B_{bj},$BBox_{ij}$]) At the Reducer$_k$($1 \leqslant k \leqslant P$)
8. RNode $RT_a \leftarrow$readRNode(A_i)
9. RNode $RT_b \leftarrow \emptyset$
10. foreach(B_b,$BBox$) $\in$ [B_{bj},$BBox_{ij}$] do
11.　　$RT_b \leftarrow$readRNode(B_b)
12.　　join RT_a and RT_b to get intersection pairs(e_a,e_b) with $BBox$ as the filter condition
13. foreach entry e in a pair do
14.　　compute $e.count$ and output $<e,e.count>$ result

其中,第 1 ~ 5 行为作业提交前的预处理阶段,生成任务列表,并作为 Map 阶段的输入。Map-Reduce 模型是一种基于 key/value 通信的计算模型,通过自动切分输入数据集,在独立的数据切片上应用 Map 操作产生中间结果的键值对(key/value pair)集合,然后通过分区操作(partition)确保具有同样键的数据映射到同一分区并借助混洗操作(shuffle)在无共享(shared-nothing)集群网络上传递中间结果,最后在不同的中间结果分区上应用 Reduce 操作产生最终的规约结果。RSJA-MR 的 Map 阶段对于输入的每个任务 $t = \{A_i,B_j,BBox_{ij}\}$,行 7 输出 $t.A_i$ 为键和 B_j,$BBox_{ij}$为值的键值对。Map-Reduce 框架将所有与索引 $t.A_i$ 交叠的 B 中的子树索引集聚集在一起。这样,每个 Reduce 处理的任务集变为 SJA(A_i,[B_j,$BBox_{ij}$])。行 8 首先读取 A_i 子树根节点,行 10 ~ 12 对值集[B_j,$BBox_{ij}$]中的每个元素[B_b,$BBox$],首先读取 B 中子树索引 B_b 的根节点,利用 $BBox$ 作为过滤条件得到连接元组对的结果集,最后在 13 ~ 14 行聚集计算每个元组的聚集值。上述的基本算法未考虑 B 中子树索引基于缓存的并发访问问题,称之为 NonCached RSJA-MR 算法。

2.3.2　基于缓存的索引并发访问优化

索引并发访问的前提:当 B 中的子树索引同时与 A 中多个子树索引交叠时,产生了并发访问问题。如图 5 所示,B 中的子树索引 B_1 同时与 A_1 和 A_4 交叠,当 A_1 和 A_4 划分到不同的 Reducer 时,不可避免地产生对 B_1 子树索引的并发访问问题。HDFS 本身采用独占式的文件访问方式,若 B_1 被 A_1 访问时,A_5 则需要等待。特别地,连接聚集查询涉及连接操作,其时间代价比空间范围查询更高,更加导致索引并发访问效率的下降。

针对索引的并发访问问题,借鉴 Hadoop 分布式缓存机制[11]的思想,提出一种基于缓存的索引并发访问优化方法。Hadoop 分布式缓存机制的基本思想是在分布式计算作业提交前将各节点需并发访问的小规模共享数据复制分发到各个节点。与 Hadoop 分布式缓存机制不同,RSJA-MR 算法中的缓存机制为实时缓存,其基本思想是运行过程中,对需要访问的索引文件快速复制一个本地的缓存文件,以便快速释放文件资源,便于其他 Reducer 进程并发访问。当访问子树索引时,算法首先从本地检测是否存在缓存的索引文件,若无,则快速复制分布式 R-树索引上的子树索引文件到本地,然后访问本地索引文件,若有,则直接访问本地的缓存索引文件。实时的缓存机制虽然给本地带来一定的存储负载,但可有效解决索引文件的并发访问

问题。称之为 Cached RSJA-MR。

3 实验与分析

本文的实验环境为 1 台 Master 节点和 16 个 Slave 节点构成的 Hadoop 集群。每个节点的配置为 2 quad-core 2.4GHZ CPU, 16GB 内存和 167GB 本地存储磁盘。操作系统为 Red Hat Enterprise Linux 5.5。节点间通过高速 10-Gigabit Infiniband 网络互联, Map-Reduce 框架基于 Hadoop 平台 0.20.1. 实验数据采用三组真实的美国 TIGER/Line 文件的空间数据,分别为美国(除夏威夷、阿拉斯加)的道路网数据(Roads network, R)、水文数据(Hydrograph, H)和人口普查区域数据(Census, C)作为测试 SJA 的数据集。其中 R 空间对象数为 29 692 784,大小为 7.3GB。H 空间对象数为 7 066 849,大小为 2.7GB。C 空间对象数为 8 137 053,大小为 3.9GB。

在实验的预处理阶段,需要将 TIGER/Line 文件格式转换成文中图 2 所描述的分布式 R-树索引结构,使用文献[13]的算法即可完成索引生成工作,其中 R 生成 512 个子树索引,H 和 C 各生成 128 个子树索引,索引节点容量均设为 80。实验分析两组空间连接聚集查询,一组为 R×H 交叠连接聚集查询 SIJA,统计 R 中每条道路穿过河流的数量,实验中生成的任务数为 5214 个。另一组查询为 C×H 包含连接聚集查询 SCJA,统计 C 中每个人口普查区域中河流的数量,实验中生成的任务数为 1673 个。实验主要与非缓存和非索引的并行连接聚集算法进行性能对象,并分析并行加速比。

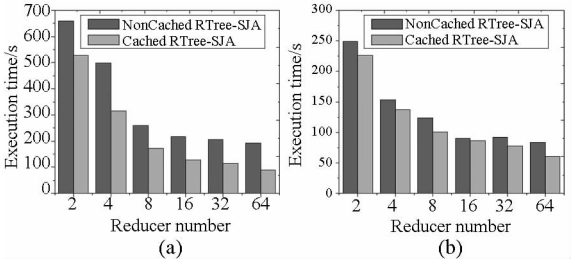

图 6 与非缓存算法性能比较

(a) R/H 交叠连接聚集 (b) C/H 包含连接聚集

Fig. 6 Performance comparison with Non-Cached RSJA-MR: (a) R/H SIJA; (b) C/H SCJA

实验 1 用来测试缓存策略对 RSJA-MR 算法性能的影响,实验结果如图 6 所示。Reducer 数 P 决定了算法的并行度,P 从 2 变化至 64。从图 6 可看出,缓存策略可有效提高算法的性能,不同 P 设置条件下,缓存的并行算法均优于非缓存的并行算法,并且随着并行度 P 的增加,算法执行时间依次递减。对于 R/H 交叠连接聚集(图 6(a)),性能提升在 19% ~53%,对于 C/H 包含连接聚集查询(图 6(b)),性能提升在 9% ~27% 间。主要原因在于,R 和 H 数据规模相差太大,造成子树索引间的交叠过多(高达 5K 多任务)。因此,索引并发访问成为限制 RSJA-MR 算法性能的重要因素。但对 C 和 H 而言,数据规模相差不大,子树索引交叠不多,因此,索引并发访问造成的性能损耗并不大。总体而言,缓存策略提高了索引访问效率,提升了 RSJA-MR 的性能。

实验 2 测试与非索引的并行空间连接聚集查询的性能对比,实验结果如图 7 所示。本文采用基于缓存的 RSJA-MR 与非索引的 Map-Reduce 连接聚集算法进行对比,称为 NonIndex SJA-MR。NonIndex SJA-MR 算法分为两个 MR 过程,第一个 MR 过程计算连接元组对,采用文献[4]的算法。第二个 MR 过程计算每个连接元组对中元素的聚集值,其基本思想是对于连接元组对中需要计算的每个元组,Map 阶段统计计算元组的部分聚集值,Reduce 阶段完成汇总。其中连接区域划分为 800×800 网格,每个网格对应一个连接任务。从图 7 中可看出,与非索引的并行连接聚集算法相比,RSJA-MR 获得明显的性能提升。特别是对 C/H 包含连接聚集,当 $P=2$ 时,算法最高提升近 100%,但对于 R/H 交叠连接聚集操作,RSJA-MR 仅略优于 NonIndex SJA-MR 算法,主要原因在于,R/H 连接聚集产生了太多的子树索引连接对。相对于交叠连接聚集操作,在 R-树索引上进行包含连接聚集操作时,可快速过滤掉大部分候选集,获得了较好的过滤优化性能,特别地,对于 128×128 个 C 和 H 的子树索引对,仅生成 1673 个任务,通过并行计算和 R-树索引过滤的双重优化,产生了较好的时间性能收益。

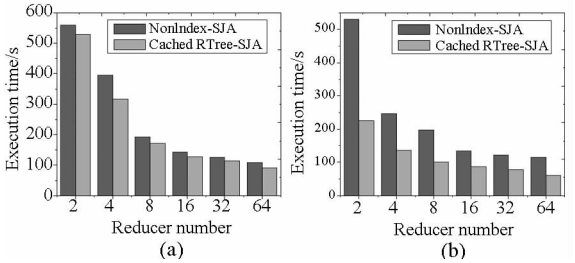

图 7 与非索引算法性能比较

(a) R/H 交叠连接聚集; (b) C/H 包含连接聚集

Fig. 7 Performance comparison with Non-Index RSJA-MR: (a) R/H SIJA; (b) C/H SCJA

实验 3 分析 RSJA-MR 并行算法的加速比。RSJA-MR 利用 Map-Reduce 集群上多 Reducer 的

并行执行完成连接聚集查询,并行加速比旨在测试 Reduce 数 P 对于算法性能的加速情况。这里,并行加速比的定义如下:

$$\text{Speedup} = \frac{\text{串行执行时间}}{\text{并行执行时间}}$$

为对比性,串行算法执行环境为同样配置的计算节点,串行算法不存在并发访问问题,因此,不采用缓存优化。下面测试非缓存和采用缓存的 RSJA-MR 算法与串行算法的并行加速比。

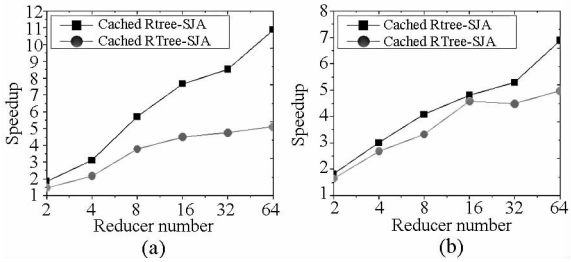

图 8　并行加速比

(a)R/H 交叠连接聚集;　(b)C/H 包含连接聚集

Fig.8　Parallel speedup:(a)R/H SIJA;(b) C/H SCJA

图 8 给出实验结果,(a)为 R/H 交叠连接聚集操作的加速比,(b)为 C/H 包含连接聚集操作的加速比。从图 8 可以看出,在实验在 16 节点集群环境中,在 Reducer 数从 2 变化到 64 的情况下,(a)中缓存算法的加速比为 1.85 ~ 10.89,非缓存算法的加速比为 1.48 ~ 5.1。(b)中缓存算法加速比为 1.82 ~ 6.88,非缓存算法的加速比为 1.65 ~ 4.97。随着并行度 Reducer 数的增加,RSJA-MR 算法的加速比增加。对于非缓存的 RSJA-MR 算法,从中可看出,当 Reducer 数大于节点数 16 时,加速比增长平缓,甚至下降,索引的并发访问成为限制 RSJA-MR 算法性能的一个非常重要因索。通过缓存优化,提升了索引的并发访问效率,取得了较好的并行加速比,即当 Reducer 数大于节点数时,仍能提升加速比。

4　结论

针对目前 Map-Reduce 框架下在处理基于空间索引的连接聚集查询操作问题上研究的不足,提出一种基于分布式文件系统的 R-树索引结构,利用任务连通图 TCG 分析提出了面向连接聚集查询的任务划分,以适应 Map-Reduce 框架的易并行和串行同步的并行计算抽象,详细提出了 Map-Reduce 框架下任务划分与任务执行算法。针对索引并发访问问题,提出一种基于缓存的并发访问机制,有效地提高了效率。实验结果表明,该算法能将 R-树索引与 Map-Reduce 并行计算模型进行有效地结合,相比于非索引的连接聚集操作,提高了时间性能,具有较好的加速比。

致谢:感谢国防科大电子科学与工程学院研究生实验室为研究提供高性能计算集群作为实验环境。

参考文献(References)

[1] Zhu M L, Dimitris P, et al. Top-k spatial joins[J]. IEEE Transactions on knowledge and data engineering, 2005, 17(4):567 - 579.

[2] Yu C, Cui B, et al. Efficient index-based knn Join processing for high-dimensional data. Inf. Softw. Technol, 2007,49(4): 332 - 344.

[3] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM,2008(1): 1958 - 2008.

[4] Zhang S B, Han J Z, et al. SJMR: parallelizing spatial join with Map-Reduce on clusters [C]//Proceedings of Cluster, 2009.

[5] Zhang C, Li F F, et al. efficient parallel kNN Joins for large data in Map-Reduce[C]//Proceedings of the EDBT, Berlin, Germany,2012:38 - 49.

[6] Wang K, Han J Z, et al. accelerating spatial data processing with MapReduce[C]//Proceedings of ICPDS,2010.

[7] Liu Y, Chen L, et al. parallel top-k spatial joins with MapReduce[C]//Proceedings of IADIS ECDM, Italy, Roma, 2011:197 - 202.

[8] Guttman A. R-trees: a dynamic index structure for spatial searching[C]//Proceedings of ACM SIGMOD , 1984: 47 - 57.

[9] Bringkhoff T, Kriegel H P, et al. parallel processing of spatial joins using R-trees[C]//Proceedings of IEEE ICDE, 1996: 258 - 265.

[10] Gray J, Bosworth A, et al. data cube: a relational aggregation operator generalizing group-by, cross-tabs and subtotals[C]// Proceedings of IEEE ICDE, 2000.

[11] White T. Hadoop: The definitive guide[M]. Yahoo! Press, Sebastopol,2009.

[12] Schnitzer B. , Leutenegger S T. Master-Client R-trees: a new parallel R-tree architecture [C]//Proceedings of the SSDBM, 1999.

[13] Liu Y, Jing N, et al. parallel bulk-loading of spatial data with MapReduce: an R-tree case[J]. Wuhan University Journal of Natural Sciences,2011,16(6):513 - 519.