软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(Suppl.(2)):99−109

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

集群上一种面向空间连接聚集的并行计算模型*

刘 义, 景 宁, 陈 荦, 熊 伟

(国防科学技术大学 电子科学与工程学院,湖南 长沙 410073)

通讯作者: 刘义, E-mail: liu.yi.nudt@gmail.com

摘 要: 单机运行环境难以满足海量空间数据的连接聚集操作对时空开销的需求,集群上的并行计算是高效处理海量空间数据的连接聚集操作的关键. Map-Reduce是云计算中一种应用于大规模集群进行大规模数据处理的分布式并行编程模型,分析发现,Map-Reduce并不直接支持以既高效又自然的方式来处理具有二次归约特征的并行空间连接聚集操作.因此,提出了一种并行计算模型——Map-Reduce-Combine(MRC)来有效地处理大规模空间数据的连接聚集操作.MRC在Map-Reduce模型上增加一个Combine阶段,有效地合并分散在各个Reducer的部分聚集结果.针对并行任务划分中空间对象的单分配问题,提出了过滤优化算法,提高了MRC下处理空间连接聚集查询的效率.实验验证所提出的并行计算模型在处理空间连接聚集查询时具有良好的效率、有效性、可扩展性和简单性.

关键词: 云计算;Map-Reduce;空间连接聚集;空间查询;二次归约

中文引用格式: 刘义,景宁,陈荦,熊伟.集群上一种面向空间连接聚集的并行计算模型.软件学报,2013,24(Suppl.(2)):99−109. http://www.jos.org.cn/1000-9825/13028.htm

英文引用格式: Liu Y, Jing N, Chen L, Xiong W. Parallel computing model for spatial join aggregate on cluster. Ruan Jian Xue Bao/Journal of Software, 2013,24(Suppl.(2)):99−109 (in Chinese). http://www.jos.org.cn/1000-9825/13028.htm

Parallel Computing Model for Spatial Join Aggregate on Cluster

LIU Yi, JING Ning, CHEN Luo, WEI Xiong

(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Corresponding author: Liu Yi, E-mail: liu.yi.nudt@gmail.com

Abstract: Since processing large-scale spatial join aggregate (SJA) is usually difficult to be implemented on a single machine, parallel computing on cluster has been the key to process large-scale SJA operation efficiently. Map-Reduce has been the mainstream parallel computing technique for massive data on cluster. However, Map-Reduce does not directly support processing parallel SJA with both high efficiency and straightforward way, for it needs to perform a second reduce operation. This paper proposes a novel parallel computing model, Map-Reduce-Combine (MRC), which is able to process large-scale SJA efficiently with a simple way on cluster. MRC adds to Map-Reduce a Combine phase that can efficiently combine partial aggregate results distributed among different Reducers, which is caused by the multiple assignment of spatial object. For the spatial object assigned only once, a filter optimization method has been proposed to pick up the result of single assignment object obtained in Reduce phase and further enhance the performance of processing SJA. Extensive experiments in large real spatial data have demonstrated the efficiency, effectiveness, scalability and simplicity of the proposed parallel computing model for processing SJA on massive spatial data.

Key words: cloud computing; Map-Reduce; spatial join aggregate; spatial query; second reduce

空间数据的指数级增长给空间数据库的查询操作提出了更严峻的挑战.空间连接聚集(spatial join aggregate,简称SJA)查询是一种常用并且非常重要的数据库操作,基于空间属性信息,空间连接聚集查询返回两

* 基金项目: 国家自然科学基金(61070035, 41271403); 国家高技术研究发展计划(863)(2011AA120306, 2007AA120402); 教育部高等学校博士学科点专项科研基金(20104307110017)

收稿时间: 2012-08-05; 定稿时间: 2013-07-22

个或多个空间关系中的聚集信息.TPC-H 是在线分析处理环境中的一种典型的评价标准,其中,所有的 22 个查询中有 16 个涉及到连接聚集查询.相比于关系数据的连接聚集查询,空间连接聚集查询更加耗时,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.

空间连接聚集查询与空间范围聚集(range aggregate,简称 RA)查询[1]有些类似.指定空间关系 R 和查询窗口 q,RA 查询返回 R 中所有与 q 交叠的子集的对象数目(或基于对象属性的总和、平均值、最大值和最小值)聚集信息.但与 RA 查询相比,SJA 查询复杂度更高,当处理 RA 查询的技术应用于 SJA 查询时,会导致巨大的计算量.处理 SJA 查询需要|R|+|S|次 RA 查询,而处理半空间连接聚集 semi-SJA 需要|R|次 RA 查询.特别地,上述算法运行在单机单线程的环境中,适合于解决了小规模空间数据的聚集查询操作.由于空间数据的海量特性及空间连接聚集查询的复杂性,单机运行环境难以满足大规模空间数据连接聚集操作其对时空开销的需求.

集群上的并行计算是解决大规模数据处理操作的有效方法.集群上的并行计算可以用不同的并行计算模型表示和实现,每一种模型都有一些适应的计算应用.近年来,应用于大规模集群处理大规模数据的并行计算模型 Map-Reduce[2]及其开源实现框架 Hadoop[3],已广泛应用于互联网规模数据处理中的倒排索引构建、聚类、推荐等批量处理和分析计算中.然而,SJA 查询涉及多个异构数据源,并且由于空间对象在并行任务划分中的多分配特性,需要进行二次归约处理.Map-Reduce 模型适合于同质数据的单次归约处理,并不直接支持 SJA 中多源异构数据集连接聚集处理以及二次归约操作.针对 Map-Reduce 并行计算模型在处理 SJA 中存在的缺陷,本文扩展 Map-Reduce 模型,提出一种新的处理 SJA 的并行计算模型 MRC.本文的主要贡献包括:

(1) 抽象了支持二次归约的改进的并行计算框架的编程模型,利用无依赖并行和串行同步的形式化定义描述了 Map-Reduce 的实现,通过增加 Combine 阶段增强 Map-Reduce 的编程规范,为解决有多源异构输入和二次归约需求的数据处理提供了一种高效的并行计算模型.

(2) 提出了前向扫描的空间连接聚集算法.在并行 SJA 处理中,针对空间对象的分配特征,提出了过滤优化算法.

(3) 对新并行计算模型下解决 SJA 问题以及过滤优化算法获得的性能优化进行了详细的分析和对比.

(4) 通过在 32 节点的集群环境下使用三组空间数据集设计了并完成了相关实验,实验结果表明在给定的数据集和实验环境下,改进的并行计算框架在处理 SJA 问题上的执行性能均优于基于 Map-Reduce 的算法.

本文第 1 节介绍相关的研究工作.第 2 节描述并行计算模型.第 3 节对 MRC 框架下的 SJA 的处理方法进行详细介绍.第 4 节介绍其过滤优化算法.第 5 节对新并行框架下 SJA 算法和过滤优化算法的代价与基于 Map-Reduce 的算法进行代价分析.第 6 节分析实验和结果.最后一节对全文进行总结.

1 相关工作

1.1 空间聚集查询

相比于空间查询,返回统计信息的空间聚集操作更受用户关注,相关的研究主要集中于空间范围聚集查询和最近邻查询上.aR-tree[4]是一种面向空间范围聚集查询的空间索引结构,aR-tree 索引在 R-tree 空间节点上增加子节点的统计信息,以此减少子节点的访问次数,因此,在空间范围聚集查询上性能明显优于 R-tree.基于类似思想,aP-tree[5]针对点目标的空间范围聚集查询进行了优化.最近邻(nearest neighbor,简称 NN)[6,7]查询和反最近邻查询(reverse nearest neighbor,简称 RNN)[8]是空间聚集查询中另外两个典型例子.给定查询目标 o,NN 查询返回与 o 具有最小距离的对象.反之,给定查询目标 o,RNN 查询返回一组结果集,其中 o 是结果集中每个元素的最近邻.文献[6,7]详细给出了道路网和空间数据库中聚集性的最近邻算法.文献[8]则提出了一种面向连续反最近邻的监测算法.在空间连接聚集上, Zhu 提出了一种 Top-k 空间连接聚集算法 TKSJ[9],该算法在两个 R-tree 索引上采用分支界定方法,缩小连接计算区域,降低计算成本.上述算法的研究均限定在单机环境中,导致其不能满足海量空间数据连接聚集应用需求.

1.2 并行计算模型

并行计算可以用多种不同的并行编程模型表示和实现,每一种模型都有与其相适应的一类计算应用.Google 公司工程师在处理互联网规模的非结构化数据时,发现大部分的处理任务可用两个简单的并行过程 Map 和 Reduce 进行处理,于是提出并开发了 Map-Reduce 并行编程模型[2],并被广泛应用于互联网规模的非结构化数据处理中的搜索引擎、统计机器翻译等.然而 Map-Reduce 作为一种大规模数据处理的通用计算框架,并不适用于迭代、图算法以及连接操作等方面的计算应用.针对传统的 Map-Reduce 框架不易于表达迭代式操作的问题,Bu 等人设计了 HaLoop[10],一种基于 Hadoop 的支持内部迭代的数据并行处理系统,有效减少了迭代过程中数据重载以及迭代中间结果持久化开销.针对 Map-Reduce 框架在处理图问题上的不足,Pan 借鉴并行计算中的成熟的消息传递接口(message passing interface,简称 MPI)技术,通过引入大同步模型,利用 MPI 和超级步同步来高效地支持 Map-Reduce 的分布式图算法[11].同时,针对 Map-Reduce 并不直接支持处理多个异构数据源的关系数据连接处理的问题,Yang 扩展 Map-Reduce 框架,增加一个 Merge 阶段处理 Map 和 Reduce 阶段输出的划分和排序数据的关系连接操作[12].然而,该并行模型并不支持空间连接聚集操作,正如关系连接算法不能应用于空间连接算法一样.借鉴文献[12]的思想,本文在继承 Map-Reduce 原有诸多特性的基础上增加一个 Combine 阶段来更高效地支持空间连接聚集操作.与 Yang 提出的模型不同,通过引入同质化将聚集连接操作前置到 Reduce 阶段处理,Combine 阶段完成各个 Reduce 阶段生成的部分聚集结果的合并处理.

2 并行计算模型

2.1 无依赖并行和串行同步计算模型抽象

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,简称 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.

定义 1(无依赖并行计算). 并行计算中,给定一个作业 J, 其可以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,若对于输入任务 T 的任意划分 $\forall T=\{t_1,t_2,\ldots,t_m\}$,若其满足 $t_i\cap t_j=\varnothing$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.

定义 2(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 之间是串行同步的.

Map-Reduce 模型是典型的无依赖并行和串行同步的计算描述抽象.如图 1(a)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~Map_m 或者 $Reduce_0$~$Reduce_n$)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.但是 Map 产生中间结果的 shuffle 过程是与 Map 以重叠方式执行,即任一 Map 执行完毕后,Reduce 就可以读取中间结果,这样可缩短并行流水处理的长度,提高并行处理效率.如果一个并行应用需要多个 MR 过程,那么一次 MR 过程与下一次 MR 过程也是链式串行同步的.通信和数据交互也仅发生在一次 MR 过程中的 Map 阶段和 Reduce 阶段以及多次 MR 过程之间.

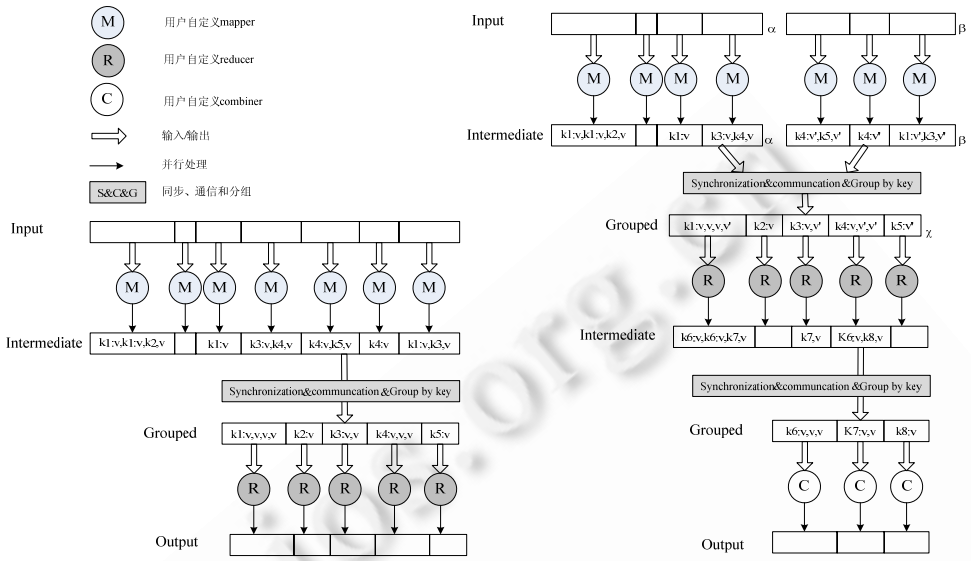


图 1 并行编程模型

2.2 Map-Reduce-Combine并行模型

Map-Reduce 模型并不直接支持二次归约操作,当需要执行二次归约的计算任务时,需要增加一个 MR 过程,导致不必要的数据传输和 MR 间的链式通信代价.Map-Reduce-Combine 模型扩展了 Map-Reduce 模型,增加一个 Combine 过程以支持二次归约,同时,对于异质数据源的连接处理,提供一个框架内的数据划分方式.图 1(b)描述了其过程,可表示为:

map $(k1,v1)_{\alpha}\rightarrow[(k2,v2)]_{\alpha}$

reduce $(k2,([v2]_{\alpha},[v3]_{\beta})\rightarrow[(k3,v4)]_{\chi}$

combine $(k3,[v4])_{\chi}\rightarrow[(k4,v5)]_{\chi}$

α,β,χ表示数据源,k 表示键,v 表示值.在新的并行模型中,Map 函数转换输入键/值对$(k1,v1)$为一系列中间结果的键/值对$[(k2,v2)]$,并根据数据来源进行同质化处理(homogenization processing,简称 HP).Reduce 函数聚集所有与 $k2$ 关联的值集,并进行数据分划处理,然后对$([v2]_{\alpha},[v3]_{\beta})$进行空间连接聚集操作,产生一系列键/值对的结果集$[(k3,v4)]_{\chi}$,Combine 函数与 Map-Reduce 框架中的 Reduce 函数类似,聚集与 $k3$ 关联的值集,并产生最终的键/值对结果集$[(k4,v5)]_{\chi}$.与 Map-Reduce 框架相比,新模型支持处理多个不同来源的数据,增加一个 Combine 阶段以支持二次归约处理.新模型本质是一个三阶段的无依赖并行和串行同步计算模型,Combine 阶段内部的异步并行任务同样运行在无依赖并行状况的理想状态下.

3 MRC 模型下的空间连接聚集操作

根据新引入的 Map-Reduce-Combine 模型,介绍如何在此模型下进行高效的 SJA 算法的设计与实现.

3.1 空间连接聚集

定义 3(连接元组对). 设变量 geometry 描述空间对象的空间属性信息,给定空间关系 R 和 S,(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.

定义 4(连接聚集组).设 $O'=\{o_1,o_2,\ldots,o_n\}$是元组 t 的连接聚集组,满足条件:1) $t\in R,O'\subseteq S$ 或 $t\in S,O'\subseteq R$;2) $\forall o\in O'$,(t,o)是连接元组对.

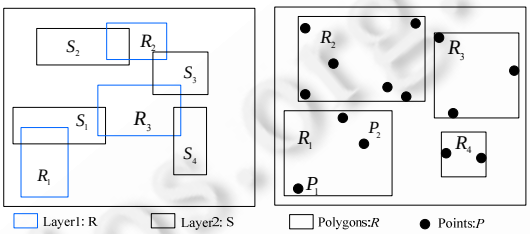
假设R和S是参与连接操作的空间关系,空间连接聚集SJA检索R中任意对象的连接聚集组中对象数目(或基于对象属性上的总和、平均值、最大值和最小值)聚集[13]信息,表示为

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

类似地,空间半连接聚集操作检索 R 中对象及其连接聚集组的统计值,表示为

$$\text{Semi-SJA}(R,S)=\{(t,\text{op}(O'(t))|\text{for all } t\in R\}.$$

本文以 count 为例来讨论 SJA 的操作,其它的聚集操作可类似处理.图 2 给出了空间连接聚集查询的例子.SJA(R,S)结果为$\langle R_1,1\rangle,\langle R_2,2\rangle,\langle R_3,3\rangle,\langle S_1,2\rangle,\langle S_2,1\rangle,\langle S_3,2\rangle$.Semi-SJA($R$,$P$)的结果为$\langle R_1,3\rangle,\langle R_2,6\rangle,\langle R_3,3\rangle,\langle R_4,2\rangle$.



(a) SJA(R,S) (b) semi-SJA(R,P)

图 2 空间连接聚集查询例子

算法 1 假定 R 和 S 均可在内存中处理,行 2 对 R 和 S 按 x 轴方向排序.行 3~行 29 依次取 R 或 S 中的首元素作为主体对象进行前向扫描的连接聚集操作.中间变量 *interLst* 用来临时存储前向扫描中被扫描的非主体对象的连接聚集值,函数 updateInterLst 对其进行更新,当主体对象扫描结束后,与 *interLst* 中的结果进行合并,即得到最终的结果.算法 1 中排序的空间复杂度为 $O(|R|\log|R|+|S|\log|S|)$,前向扫描连接聚集操作最坏情况下运行 $O(|R|\times|S|)$ 次,因此,总的空间复杂度为 $O(|R|\log|R|+|S|\log|S|+|R|\times|S|)$.

算法 1. 前向扫描的连接聚集算法 SJA(R,S).

输入: R,S.

输出: *SJALst* : A list of (object,count) pairs.

```
1. interLst ← Hashtable(tuple t,count c)
2. sorts R and S in x axis order
3. while ( R ≠ ∅ && S ≠ ∅ )
4.     let r ← Head(R) and s ← Head(S)
5.     if r.x_l < s.x_l then
6.         R.remove(r)
7.         let count ← 0
8.         while ( s ≠ ∅ and r.x_h > s.x_l )
9.             if r intersect s is true then
10.                count ← count + 1
11.                updateInterLst(interLst,s,1)
12.                s ← S.nextElement()
13.        End while
14.        count ← count + interLst.get(r)
15.        SJALst.insert(r,count)
16.        interLst.remove(r)
17.    else
18.        S.remove(s)
19.        let count ← 0
```

```
20.      while ( r ≠ ∅ and s.x_h > r.x_l )
21.          if  s intersect r is true  then
22.              count ← count + 1
23.              updateInterLst(interLst, r, 1)
24.              r ← R.nextElement()
25.      End while
26.      count ← count + interLst.get(s)
27.      SJALst.insert(s, count)
28.      interLst.remove(s)
29.  End while
```

3.2 SJA的无依赖并行任务与串行同步分解

定义 5(空间划分). 设空间连接区域为 T,空间划分 $P(T)=\{t_1,t_2,\ldots,t_m\}$,$t_i$ 为 T 的子区域, $P(T)$满足以下条件:

1) $T=\sum_{i=1}^{n}t_i$,

2) $i\neq j\Leftrightarrow t_i\bigcap t_j=\varnothing$.

本文在处理空间划分时,T 划分为一系列均匀规则的矩形网格.

定义 6. (**对象分配**). 给定一个空间关系 R 及其空间划分 $P(T)$, R_i 为 R 中分配到 t_i 的子集,R_i 满足:

1) $\forall r\in R_i$, t_i 包含 r 或 t_i 与 r 交叠为真.

2) $R=\sum_{i=1}^{n}R_i$.

若 t_i 包含 r,称 r 为单分配(single assignment,简称 SA),若 t_i 与 r 交叠,则 r 分配到 $P(T)$的多个划分中,称 r 为多分配(multiple assignment,简称 MA).

定理 1. 给定一个空间区域 T 的划分 $P(T)$,R_i 和 S_i 分别 R 和 S 中与 t_i 的关联子集,则

$$SJA(R,S)=\sum_{i=1}^{n}SJA(R_i,S_i)\ .$$

证明:根据定义 1, $T=\sum_{i=1}^{n}t_i$,则 $R=\sum_{i=1}^{n}R_i$ 和 $S=\sum_{i=1}^{n}S_i$ 成立,又由于 $i\neq j\Leftrightarrow t_i\bigcap t_j=\varnothing$,根据分配律, $SJA(R,S)=\sum_{i=1}^{n}SJA(R_i,S_i)$ 成立. □

定理 1 表明,$SJA(R,S)$可分解为无依赖并行计算的多个子任务. $SJA(R_i,S_i)$ 仅为 t_i 内的连接聚集结果,因 R_i 和 S_i 对象元组存在多分配问题,需进行合并处理.

SJA 同时存在任务分组问题,即给定一组任务集 $\sum_{i=1}^{n}SJA(R_i,S_i)$,N 个 Reducer,在考虑 Reducer 容量限制前提下,将任务集分割到这 N 个 Reducer,使得各个 Reducer 的负载最均衡.本文中的任务分组采用一组静态的空间划分函数来实现,空间划分函数能够在不同 Reducer 间系统地分布任务集.

给定空间划分 $P(T)=\{t_1,t_2,\ldots,t_m\}$,为保证负载的均衡性,通常设 $m\gg N$,每个 t_i 用一组对数(x,y)来标识,空间划分函数为每个任务单元分配一个 Reducer ID.由 N 个 Reducer 组成的 Reducer 组,其 ID 从 0 到 N–1.空间划分函数的定义为 $f:Z^+\times Z^+\rightarrow[0,1,\ldots,N-1]$,其中 Z^+表示正整数空间.本文采用 3 种空间划分方法:CMD 划分方法、Z 曲线划分方法和 Hilbert 划分方法.空间划分函数表示为 $ReducerID=f(x,y)\bmod N$,其中分别表示 CMD(x,y),$Z(x,y)$和 H(x,y)值.图 3 给出了这 3 种划分的示例,该图将 4×4 的空间任务划分映射到从 0~2 的 3 个 Reducer 上,每个 Reducer 执行一组任务子集.

根据上述分析,并行空间连接聚集查询分为 3 个串行同步阶段.1) 任务划分,基于空间划分的并行对象元组分配. 2) 任务执行,无依赖并行执行 $SJA(R_i,S_i)$.3) 合并阶段,对各个 $SJA(R_i,S_i)$的结果进行进一步的聚集.

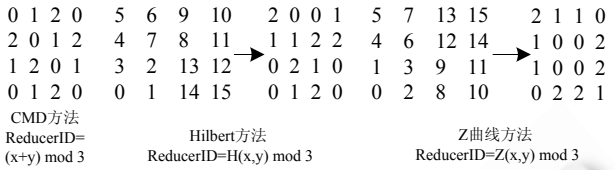


图 3 任务分组示例

3.3 实例: SJA的实现

不失一般性,本文中涉及的空间对象由对象标识符、最小外包框(由左下角坐标$[x_l,y_l]$和右上角坐标$[x_h,y_h]$表示),几何实体和其他属性数据组成.输入的键值对表示格式如图 4 所示.

Key	Value			
键名	对象标识符 (*oid*)	最小外包框MBR ($[x_l,y_l],[x_h,y_h]$)	几何实体 (*Geometry*)	其它属性数据 (*ad*)

图 4 空间对象的键值对表示

在 Map-Reduce 框架下处理并行空间连接聚集查询需要两轮 MR 过程,第 1 轮 MR 过程完成对象元组分配和 SJA(R_i,S_i),第 2 轮 MR 过程完成合并处理.图 5 给出了基于原始 Map-Reduce 的 SJA 算法过程.

Function	Input:(Key,Value)	Application Logic	Output:(Key,Value)
Map	(*keyname*,*o*)	[*t*]=*f*(*o.MBR*,*P*(T))	[(*ti*,*o*+*dst*)]
Reduce	(*ti*:*Ri*,*Si*)	SJA(R_i,S_i)	(*o*+*dst*:*count*)

Function	Input:(Key,Value)	Application Logic	Output:(Key,Value)
Map	(*o*+*dst*:*count*)	null	(*o*+*dst*:*count*)
Reduce	(*o*+*dst*:*list*(*count*))	*c*=*sum*(*list*(*count*))	(*o*+*dst*:*c*)

图 5 基于原始 Map-Reduce 的 SJA 算法

在第 1 轮 MR 过程的 Map 中,函数 $f(o.MBR,P(T))$计算对象元组 o 分配的空间划分子集$[t]$,对于$\forall t_i \in [t]$,输出键值对$(t_i,o+dst)$, dst 标识数据源. Map-Reduce 框架聚集与 t_i 关联的数据子集,并根据 dst 进行数据分划,形成 R_i 和 S_i.Reduce 函数执行 SJA(R_i,S_i),与 SJA(R,S)不同,R_i 和 S_i 中存在数据复制,即$\exists i,j(i \neq j),r,s$,使得 $r \in R_i,R_j$ 和 $s \in S_i,S_j$ 同时成立,即存在连接元组(r,s)同时配到多个划分子集中.SJA(R_i ,S_i)在进行连接操作时,引用文献[14]的参考点方法避免连接结果冗余,为简单化,本文选用左下点为参考点,即(r,s)同时分配到多个空间划分子集时,(r,s)仅在点$(\max(r.x_l,s.x_l),\max(r.y_l,s.y_l))$所在的划分子集内进行连接聚集操作.

在第 2 轮 MR 过程的 Map 中,Map 函数仅完成数据的映射操作,Map-Reduce 框架按对象元组进行重新分组,Map-Reduce 框架将分布在各个 SJA(R_i ,S_i)的结果聚集在一起,Reduce 函数完成最后的统计操作.

由于 Map-Reduce 框架不支持二次归约操作,因此,需要借助第 2 轮 Map-Reduce 框架中的 Group-by 机制执行结果数据的重新分划.所以,Map-Reduce 框架虽然可执行 SJA 的运算,但是不易表达这种具有二次归约计算的空间连接聚集操作.

改进的并行框架下 SJA 的计算充分利用并行 SJA 计算特点,根据空间对象多分配需要进行二次归约计算的需求,增加一个 Combine 阶段完成二次归约计算.对于 SJA 算法只需按 Map-Reduce-Combine 的方式完成.图 6 给出了新并行框架下 SJA 的算法过程.新并行计算框架继承了 Map-Reduce 框架的简洁性,用户指定 3 个自定义的 Map,Reduce 和 Combine 函数,即可实现具有二次归约计算的 SJA 操作.

Function	Input:(Key,Value)	Application Logic	Output:(Key,Value)
Map	(*keyname*,*o*)	[*t*]=*f*(*o.MBR*,*P*(T))	[(*ti*:*o*+*dst*)]
Reduce	(*ti*:*Ri*,*Si*)	SJA(*Ri*,*Si*)	(*o*+*dst*:*count*)
Combine	(*o*+*dst*:*list*(*count*))	*c*=*sum*(*list*(*count*))	(*o*+*dst*:*c*)

图 6 支持 MRC 模型的 SJA 算法

4 新并行计算框架下 SJA 过滤优化算法

对象元组分配函数[*t*]=*f*(*o.MBR*,*P*(*T*))需要考虑两种情况:

1) [*t*].size>1,对象元组 *o* 分配特征为 MA;

2) [*t*].size=1 的分配特征为 SA.下面对两种情况分别进行讨论.

情况 A. [*t*].size>1,对象元组 *o* 分配特征为 MA.

已知[*t*].size>1,设定函数 $c(o,t_i)$为对象元组 *o* 在划分区域 t_i 内的连接聚集值,则 $c(o,T)=\sum c(o,t'),t'\in[t]$成立.利用新并行框架执行 1 次 MRC 过程即可获取 *o* 的连接聚集结果.

情况 B. [*t*].size=1,对象元组 *o* 分配特征为 SA.

已知[*t*].size=1 成立,$t'\in[t]$成立,则 t'为 [*t*] 中单一元素,有 $c(o,T)=c(o,t')$ 成立,即对象元组 *o* 的连接聚集结果仅在 t'区域内取得.情况 B 表明,对象元组 *o* 在 Reduce 阶段的计算结果即为最终结果.

新并行计算框架下过滤优化的基本原理:对 MA 的对象元组,Reduce 函数将其发送到 Combine 阶段进行合并处理;对 SA 的对象元组,Reduce 函数直接将其写入结果集.

具体实现:在 Map 阶段对对象元组进行分配时,根据情况 A 和情况 B 对对象元组增加一个对象分配标识 *at*,Map 函数输出的键值对形式变为 $(t_i:o+dst+at)$,Reduce 函数对输出结果的 *key* 进行判断,若 *key* 中 *at* 为 SA,则直接将其写入结果集,否则,发送到 Combine 进行合并处理.

5 代价分析

本节对提出的新并行计算框架下 SJA 算法以及过滤优化算法中的计算代价进行理论分析.这里,对于给定的 SJA(*R*,*S*),在空间划分 *P*(*T*)下,*Pr*(.)函数表示 SA 对象元组占全部对象元组的比例.

5.1 新并行框架下SJA优化代价分析

基于原始 Map-Reduce 框架的 SJA 算法通过增加一个 Map-Reduce 过程进行二次归约处理. Map 阶段仅完成数据重新映射操作,利用 shuffle 阶段的 Group-by 机制将部分聚集结果重新划分到各个 Reduce 进行处理.新并行计算模型下直接采用一个 Combine 阶段执行二次归约处理.成本节省表现 Map 阶段不必要的数据传输和 Map 的启动和计算代价.

传输代价:空间划分 *P*(*T*)下,*Du*(.)函数表示对象元组的冗余系数.则 Map 阶段输入的数据量为 $U_{map}=R\cdot(1+Du(R))+S\cdot(1+Du(S))$,设定每字节传输成本为 C_{byte},则新计算框架下节省的数据传输代价为 $C_{byte}\bullet U_{map}$.

5.1.1 Map 启动代价

Map 输入为第 1 次 MR 过程产生的数据输出.设第 1 阶段 Reducer 数为 *N*, *block*[*i*] 为第 *i* 个 Reducer 输出的结果,则启动的 Map 任务数为

$$M=\sum_{i=1}^{N}\left[block[i]/chunksize\right].$$

chunksize 为 HDFS 文件块大小,默认为 64M,[·]为向上取整,显然, $M\geqslant N$ .C_{start} 为 Map 任务启动代价,则计算框架下节点的 Map 启动代价最少 $C_{start}\cdot M$.由于 Map 不参与任何计算,本文不考虑 Map 计算代价.

通过改进并行计算框架,节省的总的成本大于为 $TotalCost=C_{byte}\cdot U_{map}+C_{start}\cdot M$.由此过见,通过改进 Map-Reduce 框架,增加一个 Combine 阶段,可有效支持具有二次归约特征的 SJA 处理,减少不必要的数据传输

和 Map 映射操作代价,提高了算法效率.

5.2 过滤优化代价分析

过滤优化节省成本表现在 3 个方面:传输,I/O 和 Combine 计算.

传输代价:设定集群环境下每字节传输成本 C_{byte}, SA 对象元组的传输的数据量为 $U = R \times Pr(R) + S \times Pr(S)$, 则传输代价为 $C_{byte} \cdot U$.

I/O 代价:新并行框架继承 Map-Reduce 框架的文件通信以及 Sort-Merge 模式 Group-by 操作,Reduce 阶段的数据需写入本地的临时文件系统,Combine 远程读取各个 Reduce 阶段输出的已本地排序的数据.单位对象寻址代价为 C_{seek}, SA 对象元组数量可表示为 $U = |R| \times Pr(R) + |S| \times Pr(S)$,则寻址代价为 $C_{seek} \cdot U$.其中 $|R|$ 和 $|S|$ 分别表示 R 和 S 的势.

5.2.1 Combine 计算代价

Combine 阶段 SA 对象元组单位计算代价为 C,显然,计算代价表示为 $C \cdot U$.

通过过滤,优化算法比 MRC 算法节省的总的成本 $TotalCost = C_{byte} \cdot U + C_{seek} \cdot U + C \cdot U$.由此可见,过滤算法能避免 SA 对象不必要的数据传输、I/O 和 Combine 的合并处理,更有效地提高了算法效率.

6 实验与分析

6.1 实验设置

实验环境为 1 台 Master 节点和 32 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,原始 Map-Reduce 框架及改进并行模型均基于 Hadoop 平台 0.20.2.

实验数据采用 3 组来自美国 TIGER/Line 文件格式的空间数据,分别为美国(除夏威夷、阿拉斯加)的道路网数据(Road 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.

实验预处理阶段需要将 TIGER/Line 文件格式转换成文中图 4 所描述的键值对存储形式,使用一个简单的 MR 过程即可完成转换工作,实验中并行处理框架的输入输出都基于分布式文件系统 HDFS.实验从空间划分 $P(T)$、节点数 N 和划分函数 3 个方面测试算法的性能影响.本文实验中,$P(T)$将连接区域划分为 $2^k \times 2^k$ 均匀网格,k 从 7 到 10,默认值为 8.节点数 N 从 4 到 32(每节点 Reducer 数默认为 8),默认值为 32. 划分函数包含 CMD 划分、Z-曲线划分和 Hilbert 划分,默认为 CMD 划分.Combiner 数固定为 8.

6.2 空间交叠连接聚集性能评估及结果分析

实验 1 在 R 和 H 上评估空间交叠连接聚集(spatial intersection joins aggregate, SIJA)的性能. SIJA 实验测试原始 Map-Reduce 框架(MR)、新并行计算框架(MRC)及新并行框架下过滤优化(MRCF)的总体执行性能.

图 7 表示 SIJA 随 k 值变化情况.可以看出,随着 k 值增加,划分密度增加,SIJA 执行时间减少,表明 k 增加,空间对象连接区域变小,性能增加;当 k>8 时,SIJA 执行时间反而有所上升,表明 k 过大,导致空间对象冗余程度增加,增加了 Reduce 阶段冗余避免的计算量,导致性能下降,因此,存在一个最佳的空间划分.从图中还可看出,基于 MRC 的 SIJA 算法性能比基于原始 MR 的 SIJA 算法性能最少提升 16%,同时,MRC 下的过滤优化导致处理 SIJA 的性能更进一步提升,实验结果与第 5 节的理论分析相符.既然 MRCF 获得最佳性能,下面的实验中,算法固定为 MRCF.

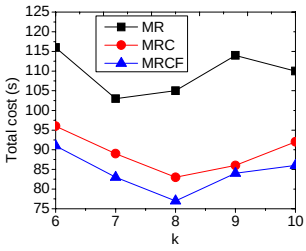


图 7 不同 k 值性能比较

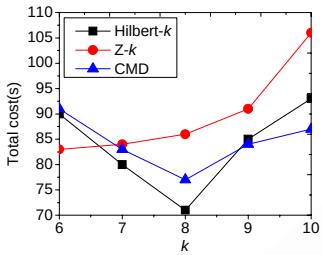


图 8 k 值对 MRCF 算法的效果

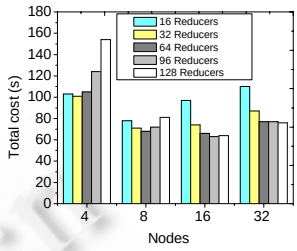


图 9 节点数对下 MRCF 的效果

图 8 表明不同任务划分方式下 MRCF 的性能,从图中可看出,在大多数 k 值条件下,Hilbert-k 和 CMD 划分性能优化 Z-k 划分,表明 Hilbert-k 和 CMD 划分下,任务分配的负载更均衡.

图 9 表示 MRCF 随节点变化的情况,随着节点数的增加,SIJA 执行时间呈下降趋势,原因是随着节点数量增加,系统的并行性和处理能力均增加,但当 N>16 时,SIJA 的执行时间反而回升,主要是因为 16 节点上处理能力可达 128 cores 的计算容量,固定计算强度下的 SIJA,过度分发其计算任务,反而导致通信代价增加,降低了性能.固定 N 的情况下,当 Reducer 数 $r \leqslant 8N$ 时,MRCF 随 r 的增加性能增加,主要是因为每个节点有 8 个核,可同时并行执行 8 个任务.当 $r > 8N$ 时,每个节点的 Reducer 任务反而不能在一轮循环中完成,导致性能下降.

6.3 空间包含半连接聚集性能评估及结果分析

实验 2 在数据集 C 和 H 上评估空间包含半连接聚集(spatial contain semi-join aggregates,简称 semi-SCJA)的性能. SIJA 实验测试原始 Map-Reduce 框架(MR)、新并行计算框架(MRC)及新并行框架下过滤优化(MRCF)的总体执行性能.

定理 2. 并行 Semi-SCJA(R,S)中,R 的分配特征为 MA,S 的分配特征为 SA.

证明:若 r 和 s 为连接元组对,即 r contain s 为真,根据冗余避免技术,r 和 s 仅在参考点(max($r.x_l$,$s.x_l$), max($r.y_l$,$s.y_l$))所在网格内执行包含操作,由于 max($r.x_l$,$s.x_l$)=$s.x_l$ 和 max($r.y_l$,$s.y_l$)=$s.y_l$ 成立,故对于 $\forall\, s \in S$,s 只需分配到点($s.x_l$, $s.y_l$)所在的网格即可. □

与 SIJA 相比,并行 Semi-SCJA(R,S)对 R 执行多分配,对 S 执行单分配,因此 SCJA(R_i,S_i)进行精炼操作时,不必要进行冗余避免操作.同时,由于仅对 R_i 中对象进行前向扫描的连接聚集操作,故每次前向扫描结果即为该对象的最终结果.由于多面体与线包含计算的极其复杂性,本文采用一种近似算法,判断 r contain s 为真时只需满足 $r.geometry$ contain $s.MBR$ 为真即可.

图 10 表示不同 k 值条件下 3 种算法的性能,性能曲线与 SIJA 类似,过滤优化的 MRCF 算法获取最佳性能.图 11 表明不同划分对 MRCF 算法的性能影响,与 SIJA 不同,3 种不同划分表现出相近的性能,表明划分函数采用 round-robin 的任务分配方式,对空间任务划分方式并不敏感.图 12 表示不同节点下 MRCF 的效果,实验结果与 SIJA 类似.

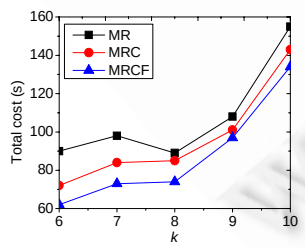


图 10 不同 k 值性能比较

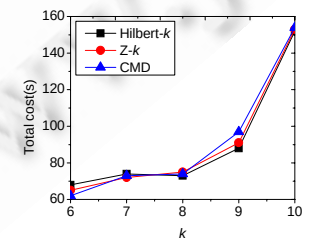


图 11 k 值对 MRCF 算法的效果

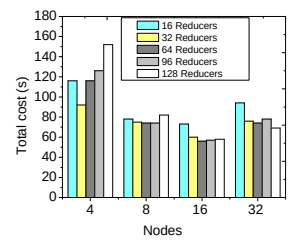


图 12 节点数对下 MRCF 的效果

7 结 论

本文针对目前 Map-Reduce 并行计算模型在处理多源异构空间数据集连接聚集操作上的不足,在开源的

Map-Reduce 框架 Hadoop 基础上通过增加一个 Combine 阶段实现了一种支持二次归约的改进型并行计算框架.通过引入 Combine 阶段,有效减少了基于原始 Map-Reduce 模型中执行二次归约的 Map 计算代价,为分布式的空间连接聚集操作提供了一个简单、高效的并行计算模式.通过提出过滤优化算法,Reduce 阶段直接输出 SA 对象元组的连接聚集结果,避免数据通信和 Combine 阶段不必要的合并操作,更进一步地提高了算法的性能.实验验证相比于原始的基于 Map-Reduce 的 SJA 算法,新计算框架下的 SJA 算法更简单高效.

References:

[1] Sathish G, Agarwal PK, Arge L. CRB-Tree: An efficient indexing scheme for range aggregate queries. In: Proc. of the Int'l Conf. on Database Theory. 2003. 143−157.

[2] Dean J, Ghemawat S. MapReduce: Simplified data processing on large Clusters. In: Proc. of the OSDI 2004. USENIX Association, 2004. 137−150.

[3] Tom White. Hadoop: The Definitive Guide. Sebastopol: Yahoo! Press, 2009.

[4] Lazaridis I, Mehrotra S. Progressive approximate aggregate queries with a multiresoluiton tree structure. In: Proc. of the SIGMOD Conf. 2001.

[5] Tao YF, Papadias D. Range aggregate processing in spatial databases. IEEE Trans. on Knowledge and Data Engineering, 2004, 16(12):1555−1570.

[6] Yiu ML, Nidos M, Dimitris P. Aggregate nearest neighbor queries in road networks. IEEE Trans. on Knowledge and Data Engineering, 2005,17(6):820−833.

[7] Papadias D, Tao YF, Mouratidis K, Hui CK. Aggregate nearest neighbor queries in spatial databases. ACM Trans. on Database Systems, 2005,30(2):529−576.

[8] Xia T, Zhang DH. Continuous reverse nearest neighbor Monitor. In: Proc. of the ICDE. 2006.

[9] Zhu ML, Papadias D, Zhang J, Lee DL. Top-*k* spatial joins. IEEE Trans. on Knowledge and Data Engineering, 2005,17(4): 567−579.

[10] Bu Y, Howe B, Balazinska M, Ernst MD. HaLoop: Efficient iterative data processing on large clusters. In: Proc. of the 36th Int'l Conf. Very Large Databases. 2010. 285−296.

[11] Pan W, Li ZH, Wu S, Chen Q. Evaluating large graph processing in MapReduce based on message passing. Chinese Journal of Computers, 2011,34(10):1768−1784 (in Chinese with English abstract).

[12] Yang HC, Dasdan A, Hsiao RL, Parker DS. Map-Reduce-Merge: Simplified relational data processing on large clusters. In: Proc. of the SIGMOD. 2007. 1029−1040.

[13] Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M. Data cube: A relational aggregation operator generalizing group-by, cross-tabs and subtotals. In: Proc. of the Int'l Conf. on Data Engineering. 2000.

[14] Dittrich, JP, Seeger B. Data redundancy and duplicate detection in spatial join processing. In: Proc. of the 16th Int'l Conf. on Data Engineering. 2000. 535−546.

附中文参考文献:

[11] 潘巍,李战怀,伍赛,陈群.基于消息传递机制 MapReduce 图算法研究.计算机学报,2011,34(10):1768−1784. http://www.jos.org.cn/1000-9825/3192.htm [doi: 10.3724/SP.J.1001.2009.03192]

刘义(1979－),男,湖南华容人,博士,主要研究领域为空间数据库,地理信息系统.
E-mail: liu.yi.nudt@gmail.com

陈荦(1973－),男,博士,教授,CCF 会员,主要研究领域为地理空间信息处理.
E-mail: luochen@nudt.edu.cn

景宁(1963－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为数据库系统,地理信息系统,空天资源规划决策技术,空间数据可视化技术.
E-mail: ningjing@nudt.edu.cn

熊伟(1976－),男,博士,副教授,CCF 会员,主要研究领域为空间数据库,地理信息系统.
E-mail: xiongwei@nudt.edu.cn