差别依赖验证的分布式算法

覃 昇 谈子敬 肖永松
(复旦大学计算机科学技术学院 上海 200433)

摘 要 数据质量是大数据研究的重要领域之一。数据一致性是数据质量评估的关键度量指标,它基于数据依赖来表述数据应该遵循的质量准则。差别依赖可以描述数据间的差异性,除了相等,还可以在定义中引入大于、小于等序列关系,因而具有较强的表述能力。数据依赖验证的目的是在数据集中发现违反数据依赖的部分数据,是进一步数据质量工作的基础。利用分布式计算环境来处理差别依赖验证的问题,以应对大数据的需求。提出分布式的随机三角分布算法,可以正确而高效地完成差别依赖在数据集上的检测;基于差别依赖的性质和数据分布特征,提出排序三角分布算法,更好地优化算法。实验证明,算法相较于常见分布式计算工具 SparkSQL 和 Hive 具有显著改善。

关键词 数据质量 差别依赖 分布式算法 数据依赖验证

中图分类号 TP31 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2018.11.044

DISTRIBUTED ALGORITHM FOR DIFFERENTIAL DEPENDENCY VALIDATION

Qin Sheng Tan Zijing Xiao Yongsong
(*School of Computer Science, Fudan University, Shanghai 200433, China*)

Abstract Data quality is one of the important fields in big data research. Data consistency is regarded as one key measure to evaluate data quality, which expresses the quality criteria that data should follow according to data dependency. Differential dependency (DD) can state differences between data value. In addition to equality, it can also introduce sequence relationships in the definition such as greater than and less than, so it has strong expressive ability. The goal of data dependency validation is to find those data violating dependencies in the dataset, which is a fundamental step for further data quality tasks. In response to the demand of big data, distributed computing environment was used to deal with DD validation. We presented a distributed algorithm based on triangle distribution strategy, which could detect DD on the data set correctly and efficiently. Distributed algorithm based on sorted triangle was proposed to optimize the algorithm according to character of DD and distribution characteristics of data. The experiment proves that our approach significantly outperforms the common distributed computing tools such as SparkSQL and Hive.

Keywords Data quality Differential dependency Distributed algorithm Data dependency validation

0 引 言

数据质量问题在大数据时代变得更为突出。无论系统能够以多么快的速度处理多么大的数据量,如果数据质量得不到保证,一切得到的分析结果可能都是无意义的。在对数据质量进行评估的多个维度中,数据一致性是一个常见的指标。数据一致性通常基于用逻辑表达式的形式所给出的数据依赖来进行描述。简而言之,若数据不满足给定的数据依赖,则称该数据为不一致,这通常意味着数据中存在错误或误差。

因为数据依赖的重要性,有大量的研究针对各种数据依赖展开。常见的数据依赖多数强调的是数据间的相等关系,例如函数依赖、条件函数依赖[1]等。在现

收稿日期:2018-05-21。国家自然科学基金项目(61572135);上海市科技创新行动计划项目(16DZ1100200)。**覃昇**,硕士生,主研领域:分布式数据管理。**谈子敬**,副教授。**肖永松**,硕士生。

实中,相关数据之间可能存在一定差异,而并非严格完全相等,也可能需要在依赖定义中引入大于、小于等序列关系。差别依赖[2]是一个表达能力较强的数据依赖定义形式,它满足了以上这些需求。常见的函数依赖是差别依赖的一个特例。

在数据一致性的相关工作中,数据集上的数据依赖验证是一个基础且重要的步骤:其目标是在数据集中找到不满足数据依赖的部分数据,以对其进行进一步的分析和修复。随着数据量的变大,数据依赖验证所需的内存和对处理器的要求越来越高,单机无法实现。这需要引入分布式计算技术,将问题并行处理。和单机算法不同,分布式算法需将整个问题分成若干个可并行的子问题,每个子问题由一台计算机作为一个节点(reducer)进行并行处理,最终整合结果。和设计单机算法只注重时间空间消耗不同,设计一个分布式算法需要考虑的主要因素,包括如何将问题尽可能平均地并行化分解;如何在保证正确的情况下减少并发运行时间,同时减少分发和收集过程中的数据量等。

本文研究的问题是:基于大规模分布式计算平台,在数据集上进行分布式差别依赖的验证。算法将基于差别依赖的特征,对部分情况进行优化,以提出更优的算法。

1 背景知识

1.1 差别依赖

首先回顾一下差别依赖的具体定义[2]。

对于关系数据表 R 中的一个属性 B,在其值域 $dom(B)$ 上定义一个二元差别 $d_B(a,b)$ $(a,b \in dom(B))$。这个二元差别满足:(1) 非负性:$d_B(a,b) \geqslant 0$,$d_B(a,b)=0$ 当且仅当 $a=b$;(2) 对称性:$d_B(a,b)=d_B(b,a)$。例如,实数域上的绝对值运算 $d_B(a,b)=|a-b|$ 是一个符合上述要求的二元差别,字符串的最小编辑距离也同样属于二元差别。

定义 1 差别函数 $\varphi[B]$,其表达形式为 $[B(\theta,c)]$,其中 θ 为判断运算符,包括 $\{=,\leqslant,\geqslant,<,>\}$,而 c 为非负实常数。元组对 (t_1,t_2) 满足 $\varphi[B]$(记作 $(t_1,t_2) \asymp \varphi[B]$),当且仅当表达式 $d_B(t_1[B],t_2[B])\ \theta c$ 成立。

例如 $(t_1,t_2) \asymp \varphi_1[date]=[date(\leqslant 10)]$ 成立,当且仅当 $|t_1[date]-t_2[date]| \leqslant 10$ 成立,即 t_1 和 t_2 所表示的日期相差在 10 天之内。两个同属性上的差别函数可以合并简化,对于差别函数 $\varphi_2[date]=[date(\geqslant 5,\leqslant 10)]$,则 $(t_1,t_2) \asymp \varphi_2[date]$ 当且仅当 $5 \leqslant |t_1[date]-t_2[date]| \leqslant 10$ 成立。

对于多属性上的差别函数 $\varphi[Z]$,其中 Z 为若干个属性组成的集合。则有:

$$\varphi[Z]=\bigwedge_{B_i \in Z}\varphi[B_i] \tag{1}$$

$(t_1,t_2) \asymp \varphi[Z]$ 成立,当且仅当对任意 $B_i \in Z$,都有 $(t_1,t_2) \asymp \varphi[B_i]$。

定义 2 一个数据表 R 的差别依赖 DD,其形式为:

$$DD: \quad \varphi_1(X) \rightarrow \varphi_2(Y) \tag{2}$$

其中:X、Y 是 R 中属性集的子集,$\varphi_1(X)$ 和 $\varphi_2(Y)$ 是两个不同的差别函数。

一个属性集 R 的实例 I 满足差别依赖(记作 $I \vDash DD$),当且仅当对任意元组对 (t_1,t_2) $(t_1,t_2 \in I)$,$(t_1,t_2) \asymp \varphi_1[X]$ 成立时,$(t_1,t_2) \asymp \varphi_2[Y]$ 也成立。

对一个信用卡交易的数据库,可以有如下差别依赖:

DD_1 $[cardno(=0)] \cap [position(\geqslant 60)] \rightarrow [transtime(\geqslant 20)]$

其含义为:当两笔交易的卡号(cardno)相同,并且发生地点(position)相距不小于 60 时,则这两笔交易的时间(transtime)一定不小于 20。

对于一个产品价格记录的数据库,可能有如下约束:当两条记录的日期(date)相距在 7 ~ 30 天时,则价格之差将在 100 ~ 900 的范围内,其表达式为:

DD_2 $[date(>7,\leqslant 30)] \rightarrow [price(\geqslant 100,\leqslant 900)]$

可以看到,差别依赖是一个具有较强表达能力的依赖类型,并且函数依赖是它的一个特例。由差别依赖的定义可知,差别依赖的约束验证,需要对数据集中的任意元组对进行比较。该算法的复杂度是元组个数的平方。当数据集具有较大规模时,单机的内存和时间将无法承受该负荷,所以使用分布式系统进行差别依赖的验证。

1.2 分布式系统及算法

在 MapReduce[3] 和 Spark 系统中,一个分布式系统由若干无共享存储空间的计算机构成,每一台计算机被视作一个节点(reducer),所有节点之间的交互通过网络中的发送和收集信息来实现。一个分布式算法由若干轮映射归约(MapReduce)操作组成,每一轮操作分为映射(Map)、转移(Shuffle)和归约(Reduce)三部分。其中:Map 为数据传输做准备,产生包含数据内容的键值对(Key-Value);Shuffle 根据数据的键值进行实际的数据传输,使得所有键值相同的数据被归约到同一个节点;当每一个 reducer 将具有相同键值的数据收集之后,便继续完成 Reduce 中之后的步骤,可能进

行统计或输出,也可能开始下一轮的 MapReduce 操作。三个操作之中,以 Map 和 Reduce 为算法核心。

1.3　分布式验证相关工作

文献[4]中主要讨论了当数据表以水平分割或垂直分割的形式存储时,如何进行条件函数依赖的检验,使得数据传输量或并发运行时间最小。文献[5]中提出了一种基于等价类的分布式环境多函数依赖冲突检测的方法,给出了冲突检测的响应时间代价模型,并且将问题化为整数规划问题,给出近似解。文献[6]中提出了一种分布式环境多函数依赖不一致性检测方法,依靠最小集合覆盖的理论,通过一次数据遍历,对多个函数依赖进行并行检测。文献[7]中将数据依赖的检测和数据清洗化归为一系列的原子操作,并针对多操作提出了并行或合并的改进。

本文研究的是差别依赖在分布式环境下的验证。如前所述,差别依赖不仅包括函数依赖,还包括一系列建立在不相等或相近的数据上的依赖条件。本文的算法设计考虑了差别依赖的定义形式,同时针对部分的数据分布特征,进一步给出优化策略。

2　三角分布算法

差别依赖的验证基于表中元组的两两比较、直观理解,这和数据库中一个表上的自连接(join)运算有相似之处。在分布式的设定下,为了实现元组的两两比较,并且保证不重复、不遗漏以及时间上的可接受,需要一定的运算技巧。文献[8]针对大数据下元组的查重问题,使用了一种三角形分布的算法。算法的本质是模仿向量的叉乘,同时根据运算的对称性,删去其中约一半的运算。从结果上看,三角分布算法是将 reducer 排列成一个三角形,元组根据一定要求,通过 MapReduce 算法,分发到对应的 reducer 中。该算法不仅保证了正确性和比较次数的最优化,同时也保证了每一台机器的时间复杂度都相同。

以文献[8]中的方法为基础,本文给出一个适用于差别依赖检测的随机三角发表算法。

2.1　随机三角算法

当 reducer 数目给定时,比如为 m,取最大的正整数 l,使得 $l(l+1)/2 \leqslant m$ 成立,此时 l 为三角分布的边长。图 1 为 $m=21$、$l=6$ 的情况,其中每一个小方块表示一个 reducer,左上角的数字为其编号。按照图 1 对每行每列进行编号,每一台reducer可以由一个整数对来表示。例如编号为 9 的 reducer 同样可以表示为(4,2)。

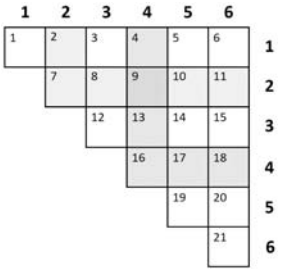

图 1　三角分布节点示意图

利用三角分布策略,可以有效实现元组的成对比较,其具体算法见算法 1。

算法 1　随机三角分布算法

输入:数据表的元组 t;
　　三角分布边长 l;
　　差别依赖 DD。
输出:违背 DD 的所有元组对。

```
1: class MAPPER
2:   method MAP(Tuple t)
3:     Iht a ← a random value from [1,l]
4:     for all p∈[1,a) do
5:       EMIT(Reducer(p,a),(L,Tuple t))
6:     EMIT(Reducer(a,a),(S,Tuple t))
7:     for all q∈(a,I] do
8:       EMIT(Reducer(a,q),(R,TupIe t))
9: class Partitioner
10:  method PARTITION(Key rid,Pair(c,t))
11:    Return rid
12: class REDUCER
13:  method REDUCE(Key rid,Pairs[(c1,t1),(c2,t2)…])
14:    Left,Right,Self ← Ø
15:    for all Pair (c,t) ∈ Pairs[(c1,t1),(c2,t2)…]
16:      if (c=L) then Left ← Left+t
17:      else if (c=R) then Right ← Right+t
18:      else if (c=S) then Self ← Self+t
19:    if Self ≠ Ø then
20:      for all t1 ∈ Self do
21:        for all t2 ∈ Self do
22:          check t1 and t2
23:    else
24:      for all t1 ∈ Left do
25:        for all t2 ∈ Right do
26:          check t1 and t2
27:    Return Pairs(t1,t2) violating the DD
```

整个算法建立在分布式结构中,共有一次 MapReduce 操作,包含 Map、Shuffle 和 Reduce 三个阶段。

在 Map 阶段(第 1 - 8 行),对每一个元组,选取一个 1 到 l 的随机数 a(第 3 行),将元组发射到第 a 行和第 a 列的所有 reducer 中(第 4 - 8 行)。发射的过程中,需要进行标记,例如一个元组的随机数为 4,则在

发送到第4列的4、9、13号机器上时,标记为L;发送到对角线上的16号机器上时,标记为S;发送到第4行的17、18号机器上时,标记为R。

在Shuffle阶段(第9-11行),数据根据机器序号进行分组。

在Reduce阶段(第12-27行),每一台非对角线上的机器,接收到标记为L和R的两类数据(第16-17行),双重循环比较L和R集合之间的元组对(第23-26行),是否符合需要验证的差别依赖;而在对角线上的机器,则只会收到标记为S的数据(第18行),收集之后对其内部的所有元组对进行两两比较(第19-22行)。全部比较结束之后,返回违背DD的元组对(第27行)。

2.2 算法正确性和时间复杂度

算法的正确性等价于任意两条元组都必须进行过比较。对于任意两条元组,设它们在第3行得到的随机数为i和j,由于差别依赖具有对称性,设$i \leqslant j$。若$i=j$时,元组会在机器(i,i)上进行比较;若$i<j$,则元组会在对应的非对角线上的机器(j,i)中进行比较。由此,任意两条元组均会得到验证,算法正确性得证。

从时间上看,算法的整体耗时主要集中在Reduce阶段的数据比较上。设元组总数为n,则取到相同随机数的元组个数平均值为n/l。

对于非对角线上的机器,其L和R集合中的元素数量均为n/l,所以总比较次数为n^2/l^2;

对于对角线上的机器,其S集合中元素数量为n/l,所以总比较次数为n^2/l^2;考虑到差别依赖的对称性,(t_1,t_2)和(t_2,t_1)的比较完全一致,所以总比较次数可以减少至$n^2/2t^2$。

综上,三角分布策略时间复杂度为$O(n^2/l^2)$,并且每一台机器上的时间平均复杂度相同。

2.3 排序三角算法

对于一部分数据集,其在某些属性上的值的上下界是已知的,并且分布大致均匀。若能通过这些已有的信息,在Map过程中就进行初筛,则可以减少部分运算时间。基于这样的想法,可以对部分数据集上的差别依赖的检验,使用如下的排序三角算法。

考虑一条左侧含有$[A(\theta p)]$的差别依赖,其中A是属性,θ是判断运算符($\theta \in \{\geqslant, >, =, <, \leqslant\}$),$p$是常数。同时$A$属性的值存在上下界,大致地均匀分布在区间$[s_{\min}, s_{\max}]$中。令:

$$t = \left\lceil \frac{s_{\max} - s_{\min}}{l} \right\rceil \tag{3}$$

t表示单位区间长度,则整个$[s_{\min}, s_{\max})$区间可以被分成l个长度为t的单位区间。在算法1中,a为1到l中的随机数(算法1第3行),而在排序三角算法中,a的取值为:

$$a = \left\lceil \frac{t[A] - s_{\min}}{t} \right\rceil \tag{4}$$

式中:$t[A]$为该元组在A属性上的取值,则$a \in [1,l]$,并且所有元组被有序分散到reducer中。令$k=p/t$,则可以根据k的值免去部分reducer上的元组比较。

如图2所示,取$k=1, l=4$。记dif_i为在编号为i的reducer中,需要比较的两个元组的属性A上值的差的范围。例如$dif_7=(p,3p)$,因为在这个reducer中需要比较的两个元组的A属性的值分别属于$[3p,4p)$和$[p,2p)$,所以两个数据之差的范围为$(p,3p)$。

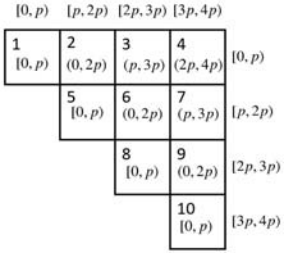

图2 节点数据示意图

图3显示了在图2条件下各reducer中可能进行的运算,显然针对特定的θ,并非所有reducer都有工作。记num_θ为θ取不同运算符时所需要的reducer数量,在忽略少量的边界情况和暂定k为整数时,计算可得:

$$num_{\leqslant} = num_{<} = (l+l-k)(k+1)/2 \tag{5}$$

$$num_{=} = l-k \tag{6}$$

$$num_{\geqslant} = num_{>} = (l-k)(l-k+1)/2 \tag{7}$$

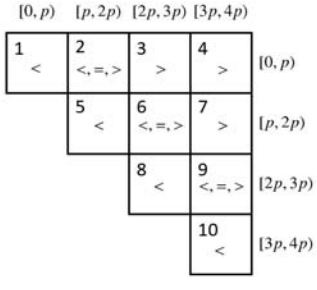

图3 各节点和运算符的对应关系

当初始条件都给定时,可以计算出对应的num_θ,所以只需要这些数量的reducer工作就可以完成所有的比较,剩下的reducer无需运行。所以可以在map阶段,直接减少数据的发射量,数据发射(emit)时,若目标reducer不需要进行运算,则可以跳过这个当前的发射步骤,减少Shuffle阶段的数据运输量。

另一方面,既然可以减少需要使用的reducer,那自然也可以用这些reducer来加速整个过程。

在已知reducer数目情况下(数目为m),可以令边

长 l' 为变量,根据之前 k 和 num_θ 的定义式,解出满足 $num_\theta \leqslant m$ 的最大 l' 值,显然有 $l' \geqslant l$。当三角分布边长从 l 变大到 l' 以后,由于每一台机器分配到的运算量的复杂度相同,而总运算量不变,所以每一台使用到的机器上分到的运算量会相应变少,总时间减少。

如果 $k = p/t$ 不为整数,则需要在上述 num_θ 的基础上进行一定的增加。从图 2 中可以看出,若 $k = 1.5$,θ 取 $<$,则编号为 3 和 7 的 reducer 同样需要进行运算,$num_<$ 会增大。在这个情况下,在一部分的行上需要额外增加一个 reducer,所以每个 num_θ 都会增加,增加的数目上界为 l。在这些额外加入的 reducer 中,并不是所有的元组对都符合 $[A(\theta p)]$,所以还需要进行检验。

3　实验分析

实验使用的分布式环境是 Spark 2.0.0,主要的启动参数见表 1。

表 1　Spark 启动参数表

参数名	值
Executor 数目	100 个
Executor 内存	3 GB
Executor 的 CPU 核数	2 个
Driver 内存	5 GB
Driver 的 CPU 核数	2 个

实验所使用的测试数据集有两个,均为人为构造的大数据集。两个数据集均含有属性 A,该属性的值域为 $[0,100)$,不同点在于,数据集 Data1 在属性 A 上的值为均匀分布,数据集 Data2 在属性 A 上的值呈正态分布,平均值为 50,方差为 20,即在 $[30,70]$ 范围内包含了总数据的约 70%。

在实验 1 和实验 2 中,只使用 Data1 作为测试数据测试三角算法的性能,而在实验 3 中,使用 Data1 和 Data2 作为测试数据,比较随机三角算法和排序三角算法的优缺点。

实验所使用的差别依赖 DD,其左侧共有两个差别函数,其中包含有差别函数 $\phi(A) = [A(\theta,30)]$。在实验 1 和 2 中,指定 θ 为 $<$,在实验 3 中,将取 θ 分别为 $<$、$=$ 和 $>$,进行两种三角分布的性能比较。

3.1　时间比较

目前在分布式平台上,用来处理结构化数据的常见工具有 SparkSQL 和 Hive。

SparkSQL 是 Spark 上的一个模块,可以从外部读入数据内容,将其转化为特有的 DataFrame 数据结构,分布式地进行存储和运算。同时它包含了部分数据库常见的操作函数,可在分布式平台上进行用户的查询或修改操作。

Hive 是基于分布式平台的一个数据仓库工具,可以将结构化的数据文件映射为一张数据库表。在 Hive 上可以进行结构化查询语言 SQL 的语句查询。当输入 SQL 语句之后,Hive 会把 SQL 语句转换为 MapReduce 任务,然后在分布式系统上进行运行。

通过将差别约束的验证改写为 SQL 语句,在 Hive 和 SparkSQL 下运行。

例如,在表格 Table1 上有差分依赖:

$$DD_3 \qquad [A(>10)] \rightarrow [B \leqslant 20)]$$

现需要返回违背 DD_3 的元组对的数量,可以用如下 SQL 语句来描述:

```
SELECT COUNT( * )
FROM Table AS X,Table AS Y
WHERE(X.A - Y.A) >10 AND ABS(X.B - Y.B) >20
```

实验 1 使用随机三角分布、SparkSQL 和 Hive 进行差别依赖的验证,比较不同数据量下的时间。此处三角分布的边长取 $l = 14$。

需要注意的是,由于 Hive 的运行时间过长,图 4 中使用了对数纵坐标来显示运行时间,横坐标为实验数据集的元组个数。当数据量为 20 万行时,Hive 的执行时间已经超过了 7 个小时,由此省略了 40 万行时 Hive 的运行结果。从图 4 可知,三角算法的时间非常快,是 SparkSQL 和 Hive 时间的 1/10 ~ 1/100。当数据量达到 40 万行时,SparkSQL 的耗时已经达到了近 2 个小时,而三角分布只需要不到 5 分钟,时间优势非常明显。

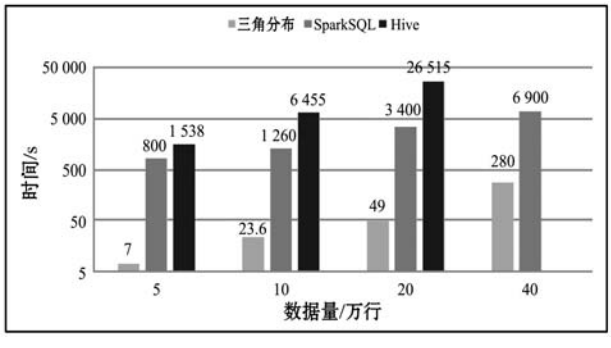

图 4　实验 1 的实验结果

3.2　不同边长下性能比较

以随机三角分布的边长和数据量作为变量,比较各情况下检验差别依赖所需要的时间。

观察图 5 中同一条曲线,当边长一定的时候,

reducer 数目不变,显然数据量的增加会导致总时间增加。另一方面,在同数据量的情况下,随着边长的增加,参与的 reducer 数目会增加,时间随之下降,这和预期一致。

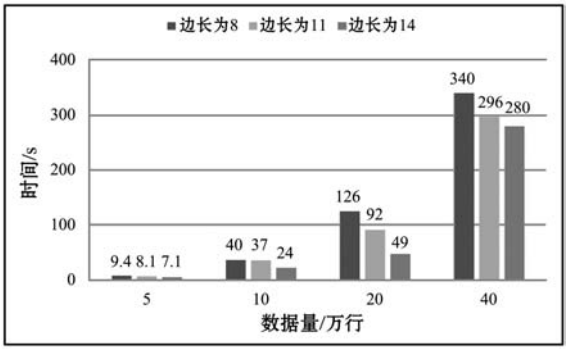

图 5 实验 2 的实验结果

需要注意 reducer 数目的增加并不会使得总时间一直下降。从实验结果看,当 reducer 数目已经较大时,再增加其数目(即增加边长)对继续缩短时间的帮助变得不再明显。这是因为中间的 Shuffle 过程是需要有实际的数据传输的。若过度增加边长和 reducer 数目,会造成数据传输量增大,而每个 reducer 上的工作减少却不明显,这可能反而会造成总时间的增加。

3.3 两种三角分布算法性能比较

排序三角算法相较于随机三角算法,可以在边长相同的情况下省去部分的 reducer,或者可以增加 reducer 利用率,减少时间,但这两点都建立在数据分布较为平均的情况下。现对于两个数据集,进行两种算法的性能比较。

3.3.1 边长相同时 reducer 的减少

当边长固定时,随机三角算法的 reducer 数目是不变的,而排序三角算法需要的 reducer 数目可以通过计算得出。以边长 $l=10$ 为例,计算得出 θ 取不同运算符时的 reducer 数量,和随机三角算法进行比较,如表 2 所示。

表 2 reducer 数目计算结果

θ	随机三角算法/个	排序三角算法/个	reducer 减少百分比
<	55	27	50.9%
=		10	81.8%
>		36	34.5%

分析表 2 得出,使用优化算法,对于同一个差别依赖,其使用的 reducer 数目得到了显著的下降,特别是在取值为 = 的时候。实际上,reducer 可减少的数目与常数值和该属性的取值范围有很大关系:当常数值接近于属性值取值范围的中间值时,不论 θ 的取值多少,都可以减少很大占比的 reducer 数目;而当常数值接近于属性值的一端时,至少也有两个 θ 的取值可以使得 reducer 数目减少很多。

3.3.2 时间上的比较

排序三角算法相较于随机三角算法,依赖于原始数据的具体分布。虽然三角边长可以增大,但若原始数据分布不均匀,会使得某些机器上的比较次数变多,进而影响总体运行时间。本次实验的数据集为均匀分布数据集 Data1 和正态分布数据集 Data2,行数为 20 万行;随机三角算法的边长 $l=10$,与排序三角算法在 θ 的不同取值下进行比较,如表 3 所示。

表 3 时间比较结果

数据集	随机三角算法时间/s	θ	排序三角算法时间/s	时间减少百分比
Data1	101	<	63	37.6%
		=	21	79.2%
		>	78	22.8%
Data2	98	<	107	-9.2%
		=	26	73.5%
		>	97	1.0%

由表 3 可得,在均匀分布的数据集上,排序三角分布算法非常出色,可以更加高效地利用所有的 reducer 进行计算。而在正态分布的数据集上,当运算符为 < 和 > 时,时间和随机分布的耗时差距不大。这说明排序三角算法相对来说依赖于初始数据和初始条件,以及此时的方差和的取值是采取何种三角分布算法的分界点。

对于排序三角算法,若数据分布不均匀,会导致部分 reducer 上的工作量超过平均值,而 MapReduce 算法的时间,依赖于所有 reducer 中最晚结束的时间,所以非平均数据集的排序三角算法的时间会长于使用相同数量 reducer 的随机三角算法的时间。对于一个数据集更适合于何种三角分布算法,可以先通过抽样来对数据分布进行分析。

另外,对比表 2 和表 3 可以得出,即使是均匀分布的数据,时间减少的百分比,也不如上一个实验中预估 reducer 减少的百分比。这主要是因为在之前的分析中,部分因素被忽略了:一部分原因是在 Shuffle 步骤的耗时是固定的(大致只与元组数量有关),这部分时间无法按比例降低;另一方面,边长增加后,由于不为整数,所以也无法做到将所有的有效比较完全均匀分配到每一个 reducer 中,依旧会存在部分的冗余比较无

法避免。

4 结　语

差别依赖用于描述数据表中元组对间属性变化量之间的关系。差别依赖的约束验证需进行元组的两两比较,通过引入分布式计算技术可以有效提高该过程的可用性。本文针对该问题进行研究,在分布式系统上提出了随机三角分布的分布式算法,实现了差别依赖的大数据检验。本文提出了排序三角分布,在均匀分布的数据集上,能够更快速高效地完成约束验证。通过实验,两个算法的时间优势非常明显。

在未来的工作中,本文将考虑多差别依赖的分布式检测算法。针对多依赖,通过合并检测的方式以减少重复的工作量。另外,经分析,数据传输量这一要素相对被忽略。在 Spark 等并行处理的系统中,Shuffle 步骤会进行数据的传输,数据传输量在 reducer 间的分派情况也会实际影响到最终的总计算时间。我们将由此探讨边长、数据转移量以及时间上的关系,以期进一步进行算法优化。

参 考 文 献

[1] Bohannon P, Fan W, Geerts F,et al. Conditional functional dependencies for data cleaning[C]//International Conference on Data Engineering. IEEE, 2007:746-755.

[2] Song S, Chen L. Differential dependencies: Reasoning and discovery[J]. Acm Transactions on Database Systems, 2011, 36(3):1-41.

[3] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]//Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation—Volume 6. USENIX Association,2004: 10-10.

[4] Fan W, Geerts F, Ma S,et al. Detecting inconsistencies in distributed data[C]//International Conference on Data Engineering. IEEE, 2010:64-75.

[5] 李卫榜, 李战怀, 姜涛. 分布式大数据多函数依赖冲突检测[J]. 计算机学报, 2017,40(1):144-160.

[6] 李卫榜, 李战怀, 陈群,等. 分布式大数据不一致性检测[J]. 软件学报, 2016, 27(8):2068-2085.

[7] Khayyat Z, Ilyas I, Jindal A,et al. BigDansing: a system for big data cleansing[C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015:1215-1230.

[8] Chu X, Ilyas I, Koutris P. Distributed data deduplication [J]. Proceedings of the VLDB Endowment, 2016,9(11): 864-875.

(上接第 244 页)

[7] Comaniciu D, Meer P. Mean Shift: A Robust Approach Toward Feature Space Analysis[C]//IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002:603-619.

[8] Sun J, Li Y, Tang C K, et al. Lazy Snapping[J]. Acm Transactions on Graphics, 2004, 23(3):301-306.

[9] 凌财进,曾婷,张超, 等. 一种改进的基于分水岭的图像分割算法[J]. 计算机测量与控制,2016(6):214-217.

[10] 张喆, 韩德强, 杨艺. 基于证据马尔可夫随机场模型的图像分割[J]. 控制与决策, 2017, 32(9):1607-1613.

[11] Besag J. On the Statistical Analysis of Dirty Pictures[J]. Journal of the Royal Statistical Society, 1986, 48(3):259-302.

[12] Peng B, Zhang L, Yang J. Iterated Graph Cuts for Image Segmentation[C]//Asian Conference on Computer Vision. Springer, Berlin, Heidelberg, 2009:677-686.

[13] Zivkovic Z. Gentle ICM energy minimization for Markov random fields with smoothness-based priors[J]. Journal of Real-Time Image Processing, 2016, 11(1):235-246.

[14] Chapelle O, Schölkopf B, Zien A. Semi-Supervised Learning [C]//Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. Springer-Verlag, 2006: 588-595.

[15] Wu Z, Leahy R. An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation[J]. IEEE Trans. pattern Anal. machine Intell, 1993, 15(11):1101-1113.

[16] Wang S, Siskind J M. Image segmentation with minimum mean cut[C]//Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001. IEEE, 2001:517-524.

[17] Torun O, Yüksel S E. Hyperspectral image segmentation using normalized cuts[C]//Signal Processing and Communication Application Conference. IEEE, 2016:1717-1720.

[18] Wang S, Siskind J M. Image segmentation with ratio cut [J]. Pattern Analysis & Machine Intelligence IEEE Transactions on, 2003, 25(6):675-690.

[19] 韩守东, 赵勇, 陶文兵,等. 基于高斯超像素的快速 Graph Cuts 图像分割方法[J]. 自动化学报,2011, 37(1): 11-20.

[20] Zhu X J, Goldberg A B, Brachman R, et al. Introduction to Semi-Supervised Learning[M]. Morgan and Claypool Publishers, 2009.