软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(4):813−825 [doi: 10.13328/j.cnki.jos.004564]

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

一种云环境下的大数据 Top-*K* 查询方法*

慈 祥, 马友忠, 孟小峰

(中国人民大学 信息学院,北京 100872)

通讯作者: 慈祥, E-mail: cixiang31415926@126.com

摘 要: Top-*K* 查询在搜索引擎、电子商务等领域有着广泛的应用.Top-*K* 查询从海量数据中返回最符合用户需求的前 *K* 个结果,主要目的是消除信息过载带来的负面影响.大数据背景下的 Top-*K* 查询,给数据管理和分析等方面带来新的挑战.结合 MapReduce 的特点,从数据划分、数据筛选等方面对云环境下的大数据 Top-*K* 查询问题进行深入研究.实验结果表明,该方法具有良好的性能和扩展性.

关键词: Top-*K* 查询;云计算;MapReduce

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 慈祥,马友忠,孟小峰.一种云环境下的大数据 Top-*K* 查询方法.软件学报,2014,25(4):813−825. http://www.jos.org.cn/1000-9825/4564.htm

英文引用格式: Ci X, Ma YZ, Meng XF. Method for top-*K* query on big data in cloud. Ruan Jian Xue Bao/Journal of Software, 2014,25(4):813−825 (in Chinese). http://www.jos.org.cn/1000-9825/4564.htm

Method for Top-*K* Query on Big Data in Cloud

CI Xiang, MA You-Zhong, MENG Xiao-Feng

(School of Information, Renmin University of China, Beijing 100872, China)

Corresponding author: CI Xiang, E-mail: cixiang31415926@126.com

Abstract: Top-*K* query has been widely used in lots of modern applications such as search engine and e-commerce. Top-*K* query returns the most relative results for user from massive data, and its main purpose is to eliminate the negative effect of information overload. Top-*K* query on big data has brought new challenges to data management and analysis. In light of features of MapReduce, this paper presents an in-depth study of Top-*K* query on big data from the perspective of data partitioning and data filtering. Experimental results show that the proposed approaches have better performance and scalability.

Key words: top-*K* query; cloud; MapReduce

随着大数据时代的到来,数据开始呈现爆炸式增长.不断积累的数据,对数据存储、分析等领域提出严峻的挑战.大数据的最终价值体现在数据的分析和利用上,而对数据处理时间的要求也越来越高.一般认为,数据价值会随时间的流逝而降低.因此,如何缩短数据处理时间、提高数据处理效率的问题,引起越来越多研究者的兴趣和关注.

数据量和信息量往往是矛盾的,海量数据并不一定意味着信息的丰富,很多时候反而会导致信息过载.对用户而言,有用的信息淹没在大数据的海啸之中.如何从大数据中快速提取出有用的信息,是目前大数据的核心问题之一.在搜索引擎、电子商务、移动 App 等诸多领域,Top-*K* 查询是一种极其常见的查询类型.用户通过对不同属性的权值设定来反映其自身偏好,而系统则根据用户提交的权值计算并返回符合该用户需求的前 *K* 个结果.Top-*K* 查询能帮助用户从大量数据中得到自己最关心的信息,因此,研究大数据背景下的 Top-*K* 查询问题具

* 基金项目: 国家自然科学基金(61379050, 91224008); 国家高技术研究发展计划(863)(2013AA013204); 高等学校博士学科点专项科研基金(20130004130001)

收稿时间: 2013-09-10; 定稿时间: 2013-12-18

有非常实际和广泛的应用价值.本文的主要工作就是在云环境下,结合 MapReduce 特性,从数据划分和数据筛选两个方面改进大数据的 Top-*K* 查询效率.

1 相关工作

1.1 Top-*K*查询

Top-*K* 查询(top-*K* query),又被称作序敏感查询(rank-aware query).该问题的研究最早出现于文献[1]中,主要为了解决多媒体的检索.由于很多应用场景对结果的排序有着内在的要求,Top-*K* 问题在提出之后立刻引起关注,并在搜索引擎、多媒体检索、关系数据库等诸多领域得到了广泛的研究和应用.从数据的存储环境,可以将其划分为集中式关系数据库和分布式系统两大类.

早期 Top-*K* 问题的研究主要围绕集中式关系数据库展开.根据数据源的访问方式,又可进一步细化为 3 大类,即:支持有序和随机的数据访问、不支持随机的数据访问以及随机访问受限的数据访问.在支持有序和随机数据访问的算法中,最具代表性的是文献[2]提出的 TA 算法(threshold algorithm),NRA 算法[2]和 Stream-Combine[3]着重解决了数据源不支持随机访问(仅支持有序数据访问)情况下的 Top-*K* 查询.在随机访问受限的场景下,系统至少要保证一组数据源有序且随机访问以可控的方式进行,典型的算法有 MPro 算法[4]、Upper and Pick 算法[5]以及 Rank-Join 算法[6].

除了常规的 Top-*K* 查询,近几年来,一些特殊类型的 Top-*K* 问题也被提了出来,例如 Reverse Top-*K*[7,8].

随着数据量的增大,分布式 Top-*K* 查询越来越受到关注.从数据划分的方式来看,分布式环境下的 Top-*K* 问题可以归纳为垂直划分和水平划分两大类.所谓的垂直划分是数据按属性进行划分,类似于关系数据库的列存储方式,早期的分布式 Top-*K* 查询研究多使用这种划分方式.文献[9]研究了网络中分散的 Web 数据库,其假设的前提是所有数据库支持随机和有序访问.文献[10]中提出了 TPUT(three-phase uniform threshold)方法.KLEE[11]则对 TPUT 进行改进.水平划分是按元组来划分,类似于关系数据库的行存储方式.在该划分方法下,文献[12]通过对查询结果的缓存来提高查询的效率.文献[13]尝试减少用户的查询等待时间,但却会带来较大的数据传输开销.文献[14]提出了一种称为 SPEERTO 的方法进行分布式 Top-*K* 查询,其核心思想是利用 Skyline 作为辅助的数据概要进行数据处理.文献[15]对文献[14]中的方法作了进一步的完善,提出了称为 DiTo 的一整套处理框架.

1.2 MapReduce简介

MapReduce[16]是 Google 公司提出的一种编程模型.MapReduce 会将用户的原始数据进行分块,然后交给不同的 Map 任务去处理.Map 任务会从输入中解析出 *Key*/*Value* 对,用户自行定义的 Map 函数作用于这些 *Key*/*Value* 对,并得到相应的中间结果,该结果会被写入本地硬盘.Reduce 任务从硬盘上读取数据后,根据 *key* 值进行排序,将具有相同 *key* 值的组织在一起.最后,用户自定义的 Reduce 函数会作用于这些排好序的结果并输出最终结果.图 1[16]展示了一个典型的 MapReduce 任务的执行过程.

关于 MapReduce 的介绍很多,这里不再赘述.详细实现可参考文献[16].MapReduce 模型简单,且现实中很多问题都可转化到 MapReduce 框架中进行处理.因此该模型公开后,立刻受到极大关注,并在文本挖掘、信息检索等领域得到广泛的应用.Google 的 MapReduce 有多种开源实现,应用最广泛的是 Hadoop 的 MapReduce,本文方法也是以此为基础而实现的.

围绕着 Top-*K* 查询问题,近些年来开展了很多有益的研究工作.但是关系数据库以及传统的分布式环境都很难有效应对大数据环境下的 Top-*K* 查询,主要原因在于数据对象及处理方法产生了很大的变化.

(1) 云环境和传统的分布式系统存在较大差异.

从架构上来看,传统的分布式系统,比如 P2P,节点之间基本对等.而以 Hadoop 为代表的云计算系统,其数据控制和数据处理分开,有独立的节点分别完成数据控制和数据处理的任务,节点之间有一定的层次关系.从数据存储方式来看,云环境中的数据一般以块(block)的形式进行存储,每个块中会包含一批数据;而传统的数据存储

常常以元组(tuple)为单位进行存储,块的粒度远大于元组.从这个角度来看,以元组作为存储单位设计的一些 Top-*K* 算法在云环境下并不适用,例如支持随机访问特定记录的 Top-*K* 算法.

(2) MapReduce 基本成为云环境下数据批处理的标准框架.

MapReduce 是一种典型的主从式架构,由 master 节点和 slave 节点构成,其中,master 节点负责控制流,而 slave 节点则负责具体的数据处理流.slave 节点之间一般不进行通信,也就是说,数据处理过程中不会进行实时的信息共享.另外,原始的 MapReduce 是一种批处理的方式,处理过程中不会有最终结果的任何子集产生,直到处理结束才会一次性返回所有结果.

由于存在上述巨大的差异,云环境下的大数据 Top-*K* 查询面临着新的挑战.Top-*K* 问题在 MapReduce 框架下有很直接的解决方案,即,利用 MapReduce 进行数据排序再返回前 *K* 个值.这种方案既符合 MapReduce 批处理的特点,也容易实现,但其最大的缺点就是处理时间过长.每次到来一个新的查询,就要对全部数据进行一次处理,数据量巨大和查询频繁时该方法均不可取.目前,国内外结合 MapReduce 特性对 Top-*K* 查询进行专门优化的工作不多.RanKloud[17]考虑利用 MapReduce 解决多媒体数据的 Top-*K* 检索问题.通过系统运行时统计信息的收集来决定查询结束条件,但是并不能保证检索的结果一定是前 *K* 个,可能会出现检索值小于 *K* 的情况.文献[18]探讨了利用 MapReduce 处理 Top-*K* 查询的一些基本问题,但是文章本身并没有对其提到的各种问题进行深入探讨,也未对其方法的有效性进行实验验证.总的来说,目前并没有一种经过验证的可行方案能够较好地解决云环境下利用 MapReduce 对大数据进行 Top-*K* 查询的问题.

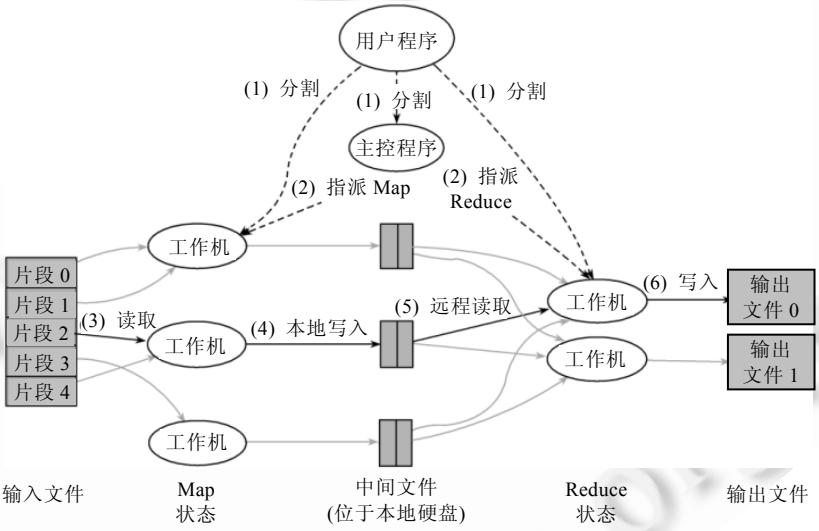

Fig.1 Execution overview of MapReduce

图 1 MapReduce 执行流程图

2 云环境下的 Top-*K* 查询

2.1 问题定义和基本概念

假设数据集为 *T*,数据集的势(cardinality)为 *m*,则 $T=\{t_i:1\leqslant i\leqslant m\}$.数据的维度 $Dim(T)=d$,因此,每个 t_i 可以表示成 $\{t_1(d),t_2(d),\ldots,t_i(d)\}$,且所有的属性均为数值型.对于 Top-*K* 问题而言,查询函数 *f* 通常是一个单调递增函数(increasingly monotone function).即,如果对 $\forall 1\leqslant n\leqslant d, t_i(n)\leqslant t_j(n)$,则 $f(t_i)\leqslant f(t_j)$.

最常用的单调函数是加权和(weighted sum),本文亦采用加权和进行相关的计算.

假设权值向量 $w=(w_1,w_2,\ldots,w_n)$,则此时的 $f(t_i)=\sum_{n=1}^{d} w_n \bullet t_i[n]$.$f(t_i)$值越大,代表排序越高.在此定义下的

Top-K 查询就是返回最大的前 k 个值.不失一般性,本文假设 $w_i \in [0,1], \sum w_i = 1$ 且 $\exists w_j > 0$.这表明,权值向量中允许有分量为 0,但不能全为 0.表 1 对上述符号进行了归纳.

Table 1 Overview of symbols

表 1 相关符号描述

符号	具体含义
T	数据集
d	数据维度
m	数据的势
k	需要返回前 K 个结果
w	权值向量
f	查询函数,本文为加权和

图 2 展示了在此定义下的一个 Top-K 查询过程和最终结果.

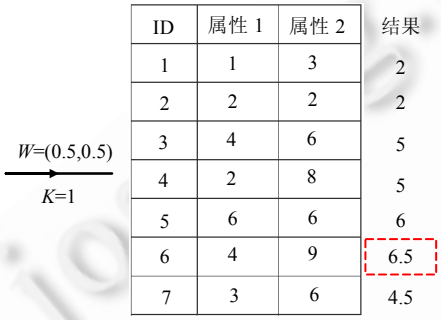

Fig.2 Typical top-K query

图 2 典型的 Top-K 查询

为了简化问题以及阐述方便,本文作如下合理的假设:

1) 数据集相对固定,或者数据的更新速度相对于整个数据集而言,可以在一定时间段内忽略不计.很多实际的应用场景符合这种假设,例如,淘宝网的商品数据虽然时刻在更新,但是相对于其整个庞大的商品基数而言,可以认为在某个固定时间内(比如 1 周)变化不大.对于变化频繁的数据集,比如流数据,本文的方法并不适用;
2) 数据分布均匀.在数据量足够大的情况下,很多场景的数据基本上符合这个要求;
3) 任意记录在其任意维的值均不为负值.现实中的应用基本符合该假设.例如,对某饭店或某商品评分,每项分值肯定大于等于 0.即使不符合,也可以通过简单的数据转换,将其数据范围转换到非负区间;
4) 所使用的服务器数量大致和数据量保持一个合理的比例.数据过多或过少,可以分别通过增加或减少服务器来实现负载平衡.

很多领域都存在着 Top-K 问题,由于存在需求上的差异,没有一种通用的方法能够适用于所有的 Top-K 问题.基于上述假设不难发现,本文方法比较适用于多次查询和参数自由变化的 Top-K 查询.这类查询在现实中很多,比如电子商务领域,用户在购买一个商品之前,可能会根据商品的多个属性组合进行搜索,以便确定是否购买.

2.2 数据划分

云环境下,数据划分的基本原则是,尽可能地将数据均匀地划分到各个服务器上.这种均匀不仅体现在数据量的均匀上,更重要的是面对特定应用时,这种划分能够尽可能地保证每个服务器上的数据对最后结果均有贡献.在以 MapReduce 为数据处理框架的云环境下,垂直划分方式不太适合,因为每个子集只有原数据集的部分属性,这样在每次计算时需要访问所有子集才能得到一个完整的加权值,而在 MapReduce 中,slave 节点之间一般

不会进行信息交换.考虑到MapReduce的这种特点,本文采用水平划分方式.进一步地,在Top-*K*领域具有代表性的水平划分方式有如下几种:随机划分、基于网格、基于角度和基于超平面.假设将记录的每个属性作为一个维度,则 *n* 维 Top-*K* 问题中的每条记录等价于 *n* 维空间的一个数据点(下文中,数据点和记录表示同一概念,可以混用,不再解释).为了便于理解这几种数据划分方式,以二维数据(即,每条记录有两个属性)为代表,具体的划分方法如图 3~图 6 所示.

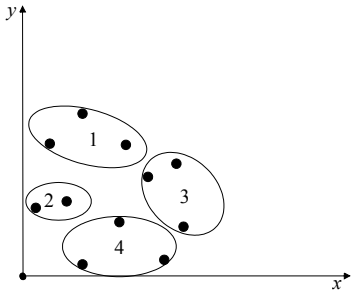

Fig.3 Random partitioning

图 3 随机划分

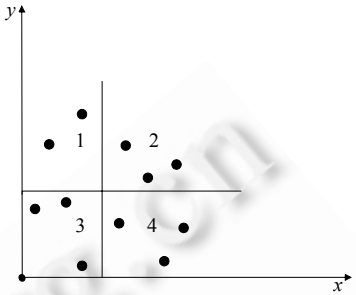

Fig.4 Grid partitioning

图 4 网格划分

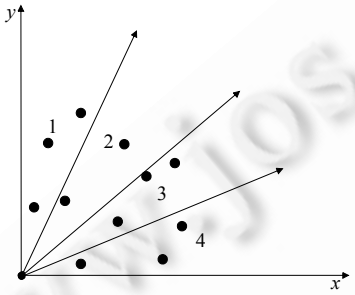

Fig.5 Angle-Based partitioning

图 5 基于角度的划分

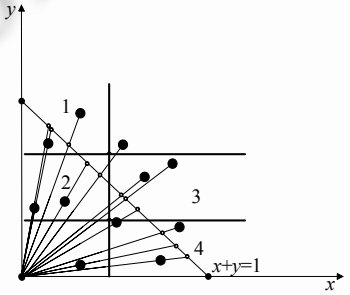

Fig.6 Hyperplane-Based partitioning

图 6 基于超平面的划分

图 3 是随机数据划分方式,对于新的数据点,通过某种方式,比如 round-robin,将数据点随机地分配到某个服务器.图 4 是网格划分,这种方法将整个数据空间划分成若干个网格,落入某个网格中的数据点则分配到相对应的服务器.图 5 描述了二维情况下基于角度的数据划分,这种方法首先将笛卡尔坐标系的数据点通过转换规则映射到超球坐标系(hyperspherical coordinate),在此基础上,对每个维度的数据进行划分,最终得到结果.图 6 是基于超平面的划分,该方法的本质是将空间数据映射到某个特定的超平面(在二维空间,超平面等价于一个直线),图例中选择的超平面为直线 $x+y=1$,具体的映射规则是,将通过数据点和原点的直线与超平面的交点作为该数据点在超平面上的映射点.完成映射之后,通过对各个数据维度进行划分来完成整个数据空间的划分.该方法可以很容易地推广到更高维空间.基于角度和基于超平面的划分都首先要对数据进行转换映射,区别在于:基于角度的划分数据坐标系发生改变,而基于超平面的划分还是在相同的坐标系.从计算复杂度来看,随机划分方式最为简单,而基于角度的划分方式最为复杂.

针对 Top-*K* 问题,随机划分和基于网格的划分效率不高,原因在于:虽然数据被划分到多个服务器上,但是每个服务器上计算的 Top-*K* 值对最终 Top-*K* 值的贡献是不同的.以加权和最大为 Top-*K* 的衡量标准,则在图 3、图 4 所示的随机划分和网格划分中,靠近右上角分区中的数据更有可能成为最终的全局 Top-*K* 值,而左下角的分区数据极可能毫无贡献.这必然会造成计算资源的浪费和计算效率的低下.最理想的状态是:每个数据分区都能计算出部分的全局 Top-*K* 值,这样就能够充分发挥系统的并行特性且充分利用计算资源.因此,基于角度的划分和基于超平面的划分是可能的候选方法,这两种数据划分方式最早都是在分布式 Skyline 计算中引入的.由于具有

一些内在的联系,Skyline的计算在很大程度上和Top-K有共通之处.但是考虑到MapReduce的特性,直接使用这两种方法都不太合适,主要原因在于,直接使用这两种划分方式对于后期的数据删选而言不够高效.因此,本文提出一种同时考虑角度和距离的划分方式.进行基于角度的划分,首先需要将欧式空间的数据点坐标转化至超球坐标.具体转换规则如下:

假设数据点的坐标$t=[t(1),t(2),\ldots,t(d)]$,则其相对应的超球坐标由一个径向坐标$r$和$d-1$个角度坐标$\phi_1,\phi_2,\ldots,\phi_{d-1}$构成,其中,

$$\begin{cases} r=\sqrt{t^2(1)+t^2(2)+\ldots+t^2(d)} \\ \phi_1=\arctan\left(\dfrac{\sqrt{t^2(1)+t^2(2)+\ldots+t^2(d)}}{t(1)}\right) \\ \ldots \\ \phi_{d-2}=\arctan\left(\dfrac{\sqrt{t^2(d-1)+t^2(d)}}{t(d-2)}\right) \\ \phi_{d-1}=\arctan\left(\dfrac{t(d)}{t(d-1)}\right) \end{cases} \tag{1}$$

考虑到第2.1节中的假设3),则$0\leqslant\phi_i\leqslant\pi/2$,对$\forall 1\leqslant i\leqslant d-1$.为了便于理解,接下来以二维空间为例解释本文的划分方法,但具体的方法可以扩展到任意维.图7是在假设有3台服务器的前提下,利用本文基于角度和距离的数据划分方式对整个数据空间进行的划分.

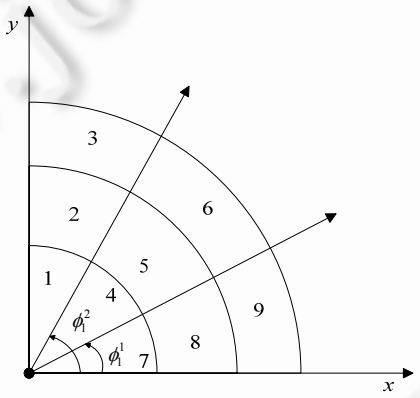

Fig.7 Angle and distance-based partitioning

图7 基于角度和距离的数据划分

具体步骤如下:

1. 依据公式(1)对整个数据空间的数据点进行数据转换,从笛卡尔坐标系转换至超球坐标;
2. 采用类似网格划分的方式对角度进行划分.此步骤划分仅考虑角度坐标,不考虑径向坐标.网格划分技术相对成熟,有很多可借鉴的划分方式,本文采用较易实现的等分划分方式,其中,等分的数量等于服务器的数量.例如在图7中,根据角度坐标将整个平面首先分成了3个部分,其中,$\phi_1^1=30,\phi_1^2=60$;
3. 经过步骤2,每个角度区间都占据了数据空间的一个部分,由于第2.1节中的假设2),我们可以认为每个角度区间所占有的数据量大致相同.在此基础上,利用径向坐标r对每个区间的数据作进一步的划分.此步骤的划分区间数量可以根据实际需求进行改变,但需保证以下两点:
   1) 在对r进行划分时粒度不能过细,至少保证二次划分的子区间包含一个块的数据量.由于第2.1节假设4)的保证,每个服务器上会有相对充足的数据量进行划分;

2) 二次划分的子区间面积相等,即,图 7 中的区间 1~区间 9 的面积应当相等.这主要是为了保证每个子区间的数据量大致相等.

以上方法是在二维空间中进行的,推广到三维空间则是对 1/8 的球体进行划分,更高维的话没有直观的几何图形,但划分方法一致,只是计算复杂度有所增加.在云环境下,相对原始的基于角度的划分,本文方法有一定的优势,详细分析在下文中会加以阐述.

2.3 基于MapReduce的大数据Top-*K*查询

2.3.1 数据筛选

在云环境下,加速 Top-*K* 计算最核心的方法有两种:

(1) 将计算过程并行化,本文通过 MapReduce 来实现;

(2) 减少计算所需的数据量.下面将结合第 2.2 节中提到的数据划分方法来阐述本文数据筛选的方法.

对于方法 2,需要思考的关键性问题是:在加权和的计算方式下,对于一个特定的 Top-*K* 查询,如果从几何角度考虑,究竟空间中满足何种性质的数据点最终会成为 Top-*K* 点?

为了解释方便,同样以二维空间的数据点为例.

假设现在有若干个数据点,这些点在二维坐标系中的位置如图 8 所示.如果现在的权值向量 w=(0.5,0.5),那么对于所有记录而言,$0.5x+0.5y$ 的值决定了其最终的排序.如果将权值向量也看作空间中的一个点(称为权值点),那么过原点和权值点可以构成一条直线(图 8 中的直线 $y=x$).此时,Top-*K* 查询有如下性质:

性质 1. 假设权值向量 $w=(w_1,w_2,\ldots,w_n)$,空间中有限数据点的集合 $t=\{t_1,t_2,\ldots,t_n\}$,则对集合 t 中任意点 t_x,计算 $L_x=\frac{t_x\bullet w}{|w|}$,可以得到集合 $L=\{L_1,L_2,\ldots,L_n\}$.如果 L 中,值小于或等于 L_x 的点有 k 个,则点 t_x 在权值向量为 w、以加权和为查询函数的 Top-*K* 查询中的最终排序为 $n–k$.

该性质的证明和解释可参见文献[19],这里不再证明.性质 1 表明:在 Top-*K* 查询中,如果以加权和作为查询函数,则数据点在空间中的排序由其在通过原点和权值点构成的直线上的投影位置所决定.可以直观地理解为:数据点在直线上的投影位置距离原点越远,其排序越高;或者说投影长度越长,排序越高.空间中点 t 到过原点和权值点的直线的投影长 $L=\frac{t\bullet w}{|w|}$,根据 L 值很容易判断点之间的排序关系.假设现在需要查询如图 8 所示的数据空间中的 Top-3 点,则这些点为 t_1,t_2,t_3,且排序关系是 $t_1>t_2>t_3$.

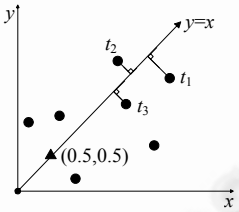

Fig.8 Geometric interpretation of top-*K* query

图 8 Top-*K* 查询的几何解释

本文在角度之外加入距离的划分因素,最大的好处就是能够确定每个划分区间的距离范围,该距离范围可以用于数据筛选.如图 9 所示,在确定数据划分和权值向量之后,通过各维度的数值区间和角度区间信息,可以计算出分区 3 所对应的投影长度区间为(L_1,L_2),也就是说,区间 3 中所有点的投影长度均大于等于 L_1,小于等于 L_2.其他区间均可得到其相对应的投影长度区间.但在面对不同的权重值时,区间内可能取到最小和最大投影值的点会发生变动,导致计算复杂度增加.为了简化计算,本文提出松弛投影范围的概念.利用此概念可以大大减少数据筛选的计算量.可以观察到,按照本文的划分方法,每个子区间都可以被一个最小外接超立方体所包围.无论权值向量如何变化,该立方体具有最小和最大投影值的点始终是$[t_{\min}(1),t_{\min}(2),\ldots,t_{\min}(d)]$和$[t_{\max}(1),t_{\max}(2),\ldots,t_{\max}(d)]$.图 10 展示了二维空间中这种计算方式(二维空间中的超立方体退化为矩形),图中虚线部分是

相对应区间的最小外接超立方体,区间 1~区间 3 对应的最小和最大投影值点均在图中标示出来.从图中还可以发现,此时的投影长度区间实际上是真实投影长度区间的一个超集,区间范围有所扩大.但是,这种方法因为最大和最小投影点固定,在面对不同的权值时,可以非常简单地计算出对应的区间,且对于实际的筛选效果影响不是很大.

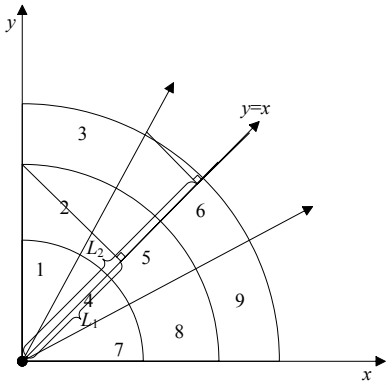

Fig.9 Computation of projected range

图 9 投影范围计算

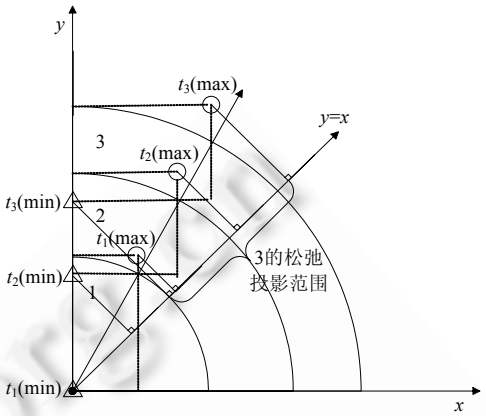

Fig.10 Relaxed projected range

图 10 松弛投影范围

数据划分完成之后,根据区间的距离信息可以确定每个区间在其相对应服务器上的排序(根据与原点间距离,由远到近),同时也可以确定松弛投影范围概念下的最小和最大投影点.将这些元数据信息以表的形式保存在 master 节点上,表 2 是一个实例.

Table 2 Metadata for data filtering

表 2 用于数据筛选的元数据

3[(0,$\sqrt{6}$/2),($\sqrt{3}$/2,$\sqrt{3}$)]	6[($\sqrt{2}$/2,$\sqrt{2}$/2),(3/2,3/2)]	9[($\sqrt{2}$/2,0),($\sqrt{3}$,$\sqrt{3}$/2)]
2[(0,$\sqrt{3}$/2),($\sqrt{2}$/2,$\sqrt{2}$)]	5[(1/2,1/2),($\sqrt{6}$/2,$\sqrt{6}$/2)]	8[($\sqrt{3}$/2,0),($\sqrt{2}$,$\sqrt{2}$/2)]
1[(0,0),(1/2,1)]	4[(0,0),($\sqrt{3}$/2,$\sqrt{3}$/2)]	7[(0,0),(1,1/2)]

表 2 中的每格代表相应区间的元数据,最前面数字表示区间号,例如第 1 格中的 3.区间号之后的两组数据分别表示松弛投影范围概念下的最小和最大投影点坐标.处于表中同一行,代表其位于同一个距离区间,例如第 1 行表示最外一层的区间 3、区间 6、区间 9,以此类推.

假设总的数据点为 m 个,总的划分区间是 n 个,因为在划分时保证每个划分区间的面积相等且有第 2.1 节中提及的假设 2),可以认为每个区间的数据点大致相等,为 m/n 个.那么,通过比较 m/n 与 k 值,就可以确定最终计算所需区间.算法 1 描述了利用表 2 中元数据进行数据筛选的过程.

Algorithm 1. Data Filtering.

Input: k and w;

Output: Data Partitioning which will be used as the data source of MapReduce.

1. $p=\lceil K/(m/n)\rceil$; /*p is upper bounds of $K/(m/n)$*/
2. Scan metadata table, get line 1 to p;
3. for i=1 to p;
4. for j=1 to q ; /*q is the number of column*/
5. $V_{\min}[i][j]=t_{\min}\cdot w/|w|$; /*Minimal relaxed projection value*/
6. $V_{\max}[i][j]=t_{\max}\cdot w/|w|$; /*Maximal relaxed projection value*/

7. end for;
8. Compare all pairs of ($V_{min}[i][j]$,$V_{max}[i][j]$);
9. if $V_{max}[i][s] \leqslant V_{min}[i][t]$;
10. Delete data partitioning *s*;
11. else;
12. Output *s*;
13. end for;

以表 2 中数据为例,具体步骤如下:

(1) 如果 $m/n>k$,则可以保证最终的 Top-*k* 值只出现在划分中最外侧区间(图 9 中的区间 3、区间 6、区间 9),在此基础上,根据表 2,对这 3 个区间作进一步计算.如果 3 个区间的松弛投影区间均有重叠部分,则 3 个区间不能进一步筛除,需要全部计算,否则可以进一步筛除.假如权值 w=(0.5,0.5),则可以利用表 2 中的数据计算出区间 3、区间 6、区间 9 的松弛投影区间分别为 $(\sqrt{3}/2,3\sqrt{6}/4)$,$(1,3\sqrt{2}/2)$,$(1/2,3\sqrt{6}/4)$.3 个区间均互有重叠,因此无法进一步筛除,需要全部进行计算.又假设权值 w=(0,1),则此时区间 3、区间 6、区间 9 相对应的松弛投影区间为 $(\sqrt{6}/2,\sqrt{3})$,$(\sqrt{2}/2,3/2)$,$(0,\sqrt{3}/2)$.区间 3 和区间 6 有重叠,但区间 9 的最大松弛投影值为 $\sqrt{3}/2$,比区间 3 的最小松弛投影值 $\sqrt{6}/2$ 还要小,区间 9 可以被进一步筛除.因此,在权值 w=(0,1)的情况下,需要计算的区间仅仅是区间 3 和区间 6.

(2) 如果 $k/2 \leqslant m/n \leqslant k$,则除了最外侧之外,还要加入次外侧区间(图 9 中的区间 2、区间 5、区间 8);然后,对区间 2、区间 5、区间 8 采用类似步骤 1 中的方法进一步筛除数据.

(3) 如果 $k/3 \leqslant m/n \leqslant k/2$,则计算所需的区间还要再往内侧增加,以此类推.

实际中,m/n 的值不光大于 *k*,常常远大于 *k*,例如搜索引擎中的海量数据.而我们最关心的往往只是结果第 1 页的几十个数据,因此,采用本文的数据划分和筛选方法,一般情况下仅需计算区间 3、区间 6、区间 9 所对应区间的 block 中的数据即可,减少了 2/3 甚至更高的数据量,大大提高了效率.

2.3.2 具体 Top-*K* 计算流程

图 11 是在本文方法下的 Top-*K* 查询框架.

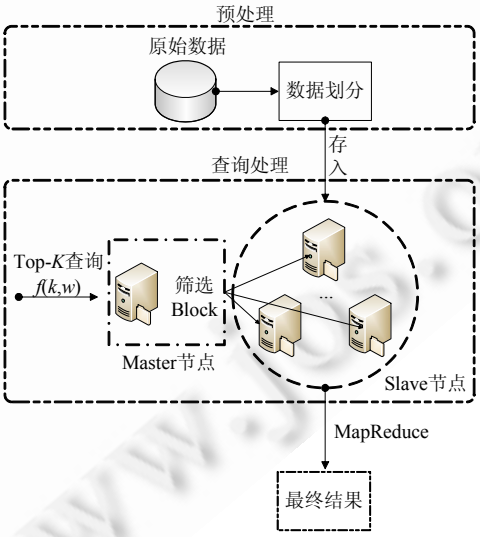

Fig.11 Top-*K* query in cloud

图 11 云环境下的 Top-*K* 查询

从整个过程来看,包括预处理和查询处理两个部分,具体步骤如下:

(1) 对数据进行预处理,主要是利用上文的方法进行数据划分,最终数据以 block 形式存入 HDFS 中;

(2) 用户提交新的查询,master 节点接受参数 k 和 w,利用其上的元数据信息按照算法 1 的步骤对区间进行筛选,确定参与最终计算的区间号;

(3) MapReduce 任务只对涉及到的区间进行计算,返回 Top-K 值.

3 实验和结果分析

3.1 实验环境

实验在一个由 11 个节点组成的集群上进行,其中,1 个节点作为 master 节点,其余 10 个节点作为 slave 节点.节点配置如下:CPU:Q9650 3.00GHz,Memory:8GB,Disk:500GB,OS:64bit Ubuntu9.10 server,节点上运行的 Hadoop 版本为 1.0.0.由于没有合适规模的真实数据集,本文按照表 3 的数据格式生成了一批均匀分布的数据,其中每条记录的维度为 10,所有值均为 0~1000 之间的整数,共 10 亿条记录,大约 75G 的数据量.

Table 3 Data format

表 3 数据格式

ID	属性1	属性2	…	属性10
1	23	134	…	565
2	1	36	…	23
…	…	…	…	…
10^9	983	345	…	87

3.2 实验结果与分析

首先需要说明的是,本文的数据划分作为预处理过程出现,一次预处理可以为后续的多次查询提供服务,因此,预处理时间未计入总时间.下面的实验若无特别说明,计算数据量均为全部数据,参数均取 k=5000,w=(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1).

3.2.1 执行时间对比

目前尚未发现与本文特别相关的方法,因此为了进行对比,本文另外实现了一种原始的 Top-K 算法和一种简单改进的 Top-K 算法.原始的 Top-K 算法思路非常直接,根据权重值,利用 MapReduce 对原始的数据进行排序,然后返回所需的 k 个最大值.简单改进的 Top-K 算法中的 Mapper 不输出所有的值,而仅输出本地 Top-K 到 Reducer 中,最后由 Reducer 返回全局 Top-K.图 12 展示了这 3 种方法的执行时间.

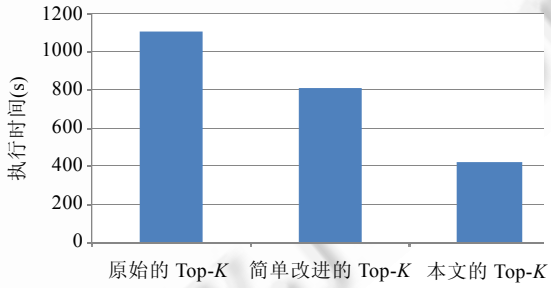

Fig.12 Comparison of execution time

图 12 执行时间对比

从图 12 中可以看出:原始方法时间最长;简单改进后的算法执行时间有较大幅度的缩短;本文方法的执行时间最短,且相对原始方法执行时间减少得很明显.

3.2.2 扩展性

从两个方面考察本文方法的扩展性:

- 首先,保持数据量不变,改变计算的服务器数量.从图 13 可以发现:随着服务器数量的增加,执行时间虽然不是完全线性地减少,但是减少的趋势是很明显的;
- 然后,保持记录数量,但改变数据维度,即,对所有的数据记录分别取其前 2 个、4 个、6 个、8 个及 10 个属性进行计算.从图 14 可以看出:计算所需的时间呈上升趋势,但上升的幅度相对稳定,未出现随着维度的增加计算时间大幅增加的情况.

以上两个方面都说明了本文方法的可扩展性较好.

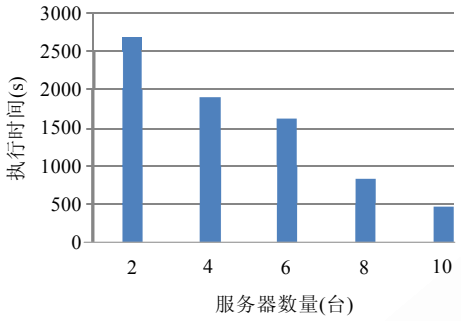

Fig.13 Scalability (servers)

图 13 可扩展性(服务器数量)

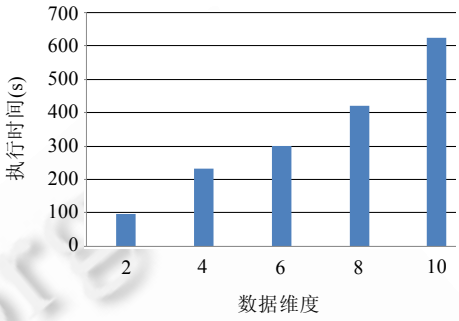

Fig.14 Scalability (data dimension)

图 14 可扩展性(数据维度)

3.2.3 不同 k 值对执行时间的影响

图 15 展示了不同 k 值下执行时间的变化情况.从实验结果来看:当数据大小固定时,执行时间并不是随着 k 值的增加而增加,也就是说,本文方法对 k 的取值不是特别敏感.分析发现:虽然实验中 k 的取值已经较大,但由于整个记录规模巨大,使得每个区间的记录数也非常大,导致面对不同 k 值时,筛选出的候选 block 集合基本一致,也就是说,这些不同 k 值实际计算的数据基本相同,因此,最终计算时间相差不多.

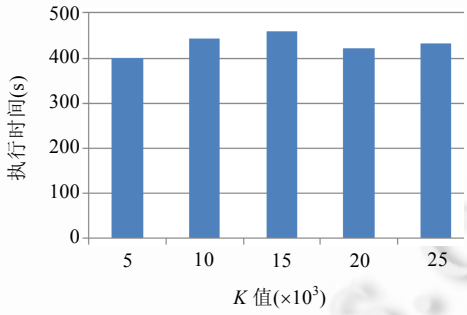

Fig.15 Effect of K on execution time

图 15 K 值变化对执行时间的影响

3.2.4 不同权值对执行时间的影响

取 4 组比较有代表性的权值类型进行比较,分别是等值权重 w_1=(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1)、极度偏向某一属性的 w_2=(1,0,0,0,0,0,0,0,0,0)、较为偏向某一属性的 w_3=(0.73,0.03,0.03,0.03,0.03,0.03,0.03,0.03,0.03,0.03)以及有偏向但是不明显的 w_4=(0.2,0.1,0.1,0.05,0.1,0.05,0.15,0.15,0.05,0.05).

从图 16 中可以看出:w_1 和 w_4 的执行时间大致相同,而 w_2 和 w_3 的执行时间大致相同.主要是因为,当根据本文的方法对某属性计算极度或较高偏好时,候选的区间会被进一步地筛选,有部分区间会在这个步骤被排除,总的计算量降低,导致最终的执行时间变短.

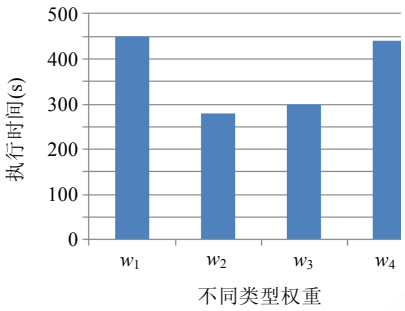

Fig.16 Effect of weight value on execution time

图 16 权值对执行时间的影响

4 结 语

本文针对云环境下的大数据 Top-*K* 查询问题,利用空间角度和距离对整个数据进行划分,考虑到 MapReduce 在计算中 slave 节点之间不进行信息实时共享的特性,在数据划分的基础上提出了一种简便的数据筛选方法.实验结果表明:在绝大多数情况下,本文方法能够大幅减少计算量,提高计算效率.同时,本文的方法也有较好的扩展性.

未来将会对现有工作进行一系列的改进,主要包括:

- 考虑将本文方法进一步扩展到数据非均匀分布的情况;
- 考虑增加缓存层,对 Top-*K* 查询的结果进行缓存,因为实际中同样的查询很可能会重复出现;
- 同时,考虑进一步细化数据划分,使得面对较小的 *k* 值时处理时间能够进一步缩短.

References:

[1] Fagin R. Combining fuzzy information from multiple systems. Journal of Computer and System Sciences, 1999,58(1):83−99. [doi: 10.1006/jcss.1998.1600]

[2] Fagin R, Lotem A, Naor M. Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences, 2003,66(4): 614−656. [doi: 10.1016/S0022-0000(03)00026-6]

[3] Güntzer U, Balke W, Kießling W. Towards efficient multi-feature queries in heterogeneous environments. In: Proc. of the Int'l Conf. on Information Technology: Coding and Computing (ITCC 2001). Piscataway: IEEE, 2001. 622−628. [doi: 10.1109/ITCC.2001.918866]

[4] Chang KCC, Hwang SW. Minimal probing: Supporting expensive predicates for top-*k* queries. In: Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2002. 346−357. [doi: 10.1145/564691.564731]

[5] Bruno N, Chaudhuri S, Gravano L. Top-*K* selection queries over relational databases: Mapping strategies and performance evaluation. ACM Trans. on Database Systems, 2002,27(2):153−187. [doi: 10.1145/568518.568519]

[6] Ilyas IF, Aref WG, Elmagarmid AK. Supporting top-*k* join queries in relational databases. In: Proc. of the 29th Int'l Conf. on Very Large Databases. San Fransisco: Morgan Kaufmann Publishers, 2003. 207−221. [doi: 10.1007/s00778-004-0128-2]

[7] Vlachou A, Doulkeridis C, Kotidis Y, Nørvåg K. Reverse top-*k* queries. In: Proc. of the 26th IEEE Int'l Conf. on Data Engineering. Piscataway: IEEE, 2010. 365−376. [doi: 10.1109/ICDE.2010.5447890]

[8] Vlachou A, Doulkeridis C, Kotidis Y, Nørvåg K. Monochromatic and bichromatic reverse top-*k* queries. IEEE Trans. on Knowledge and Data Engineering, 2011,23(8):1215−1229. [doi: 10.1109/TKDE.2011.50]

[9] Marian A, Bruno N, Gravano L. Evaluating top-*k* queries over Web-accessible databases. ACM Trans. on Database Systems, 2004, 29(2):319−362. [doi: 10.1145/1005566.1005569]

[10] Cao P, Wang Z. Efficient top-*K* query calculation in distributed networks. In: Proc. of the 23th Annual ACM Symp. on Principles of Distributed Computing. New York: ACM Press, 2004. 206−215. [doi: 10.1145/1011767.1011798]

[11] Michel S, Triantafillou P, Weikum G. KLEE: A framework for distributed top-*k* query algorithms. In: Proc. of the 31st Int'l Conf. on Very Large Data Bases. New York: ACM Press, 2005. 637−648. http://dl.acm.org/citation.cfm?id=1083667

[12] Zhao KP, Tao YF, Zhou SG. Efficient top-*k* processing in large-scaled distributed environments. Data and Knowledge Engineering, 2007,63(2):315−335. [doi: 10.1016/j.datak.2007.03.012]

[13] Dedzoe WK, Lamarre P, Akbarinia R, Valduriez P. ASAP top-*k* query processing in unstructured P2P systems. In: Proc. of the 10th IEEE Int'l Conf. on Peer-to-Peer Computing. Piscataway: IEEE, 2010. 1−10. [doi: 10.1109/P2P.2010.5569974]

[14] Vlachou A, Doulkeridis C, Nørvåg K, Vazirgiannis M. On efficient top-*k* query processing in highly distributed environments. In: Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2008. 753−764. [doi: 10.1145/1376616.1376692]

[15] Vlachou A, Doulkeridis C, Nørvåg K. Distributed top-*k* query processing by exploiting skyline summaries. Distributed and Parallel Databases, 2012,30(3-4):239−271. [doi: 10.1007/s10619-012-7094-2]

[16] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1):107−113. [doi: 10.1145/1327452.1327492]

[17] Candan KS, Kim JW, Nagarkar P, Nagendra M, Yu RW. RanKloud: Scalable multimedia data processing in server clusters. IEEE MultiMedia, 2011,18(1):64−77. [doi: 10.1109/MMUL.2010.70]

[18] Doulkeridis C, Nørvåg K. On saying "enough already!" in MapReduce. In: Proc. of the 1st Int'l Workshop on Cloud Intelligence. New York: ACM Press, 2012. 7−7. [doi: 10.1145/2347673.2347680]

[19] Tsaparas P, Palpanas T, Kotidis Y, Koudas N, Srivastava D. Ranked join indices. In: Proc. of the 19th IEEE Int'l Conf. on Data Engineering. Piscataway: IEEE, 2003. 277−288. [doi: 10.1109/ICDE.2003.1260799]

慈祥(1986－),男,安徽无为人,博士生,CCF 学生会员,主要研究领域为云计算,大数据管理.
E-mail: cixiang31415926@126.com

马友忠(1981－),男,博士生,CCF 学生会员,主要研究领域为 Web 数据管理,云数据管理.
E-mail: ma_youzhong@163.com

孟小峰(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为 Web 数据管理,移动数据管理,XML 数据管理,云数据管理.
E-mail: xfmeng@ruc.edu.cn