文章编号:1672-3961(2009)05-0032-06

基于结构化覆盖网的连续 top-k 联接查询算法

赵科军[1,2],王新军[1,2],刘洋[2],仇一泓[2]
(1. 山东大学计算机科学与技术学院,山东 济南 250101;
2. 山东大学网络信息中心,山东 济南 250100)

摘要:由于分布式计算环境中数据的分散性,在结构化覆盖网上进行连续查询是一个富有挑战性的工作. 本文首次提出基于结构化覆盖网连续 top-k 联接查询的算法,在对新数据做联接查询前,通过预计算排序函数的估值,对不可能最终影响 top-k 结果的数据裁剪,达到减少网络流量和计算负载目的. 实验证明,这种方法在保证更多的节点参与到查询同时,能有效降低网络流量.

关键词:对等网;结构化覆盖网;连续查询;top-k
中图分类号:TP393 **文献标志码**:A

Algorithms of continuous top-k join query over structured overlay networks

ZHAO Ke-jun[1,2], WANG Xin-jun[1,2], LIU Yang[2], QIU Yi-hong[2]
(1. School of Computer Science and Technology, Shandong University, Jinan 250101, China;
2. Network Center of Shandong University, Jinan 250100, China)

Abstract: It is challenging to process continuous queries over structured overlay networks due to the distribution characteristic of the environment. An algorithm based on top-k join queries over structured overlay networks is presented. In the proposed algorithms, the data that can not contribute the finale top-k results was discarded based on the pre-computing results of ranked functions. The performance of the algorithms was analyzed. Our algorithms reduced the network traffic while more nodes participate in the query process.

Key words: P2P; structured overlay network; continuous query; top-k

0 引言

当前对等网(P2P)作为一种迅速发展的分布式计算模式越来越多的受到学术界和应用界的关注. P2P 从网络结构主要分为非结构化覆盖网和结构化覆盖网:非结构化覆盖网通过本地维护邻居节点信息,实现简单,但因为没有有效的路由,查询处理需要耗费大量带宽. 结构化覆盖网络一般通过分布式散列表(distributed hash table, DHT)来组织网络节点和消息路由,每个节点和存储对象都有一个惟一的标识(ID),通过映射关系将对象发布到节点上. 结构化覆盖网能够通过对象 ID 迅速查询到该对象,而查询所需的消息数及跳数与非结构化覆盖网相比都大大减少.

流数据模型在多种应用领域中得到广泛研究,包括网络监控、金融数据分析和传感器网络等,数据的实时性和连续性是流数据的主要特征. 针对上面的应用场合,相比传统数据库,基于对等网的数据库在分布结构上有着明显的优势. 想象如下的场景:在一个入侵检测传感器对等网络中,每个传感器节点保存有一部分攻击数据表和被攻击主机表,随着攻击的不断产生,两个表的数据随时在变化. 对于“实时获取前 k 个对本地主机威胁最大的攻击者”这

收稿日期:2008-12-18
作者简介:赵科军(1981-),男,工程师,硕士研究生,研究方向为对等网计算、数据库和信息检索. E-mail: jeff@sdu.edu.cn

样非常有价值的查询,可以采用下面的方法实现.对各种攻击威胁如非法扫描,蠕虫攻击,溢出攻击等赋予不同的威胁加权值,本地网络中的主机根据主机的重要性等也赋予不同的加权值.最后将分布在不同传感器节点上的攻击表和本地主机表进行联接,通过由不同攻击和被攻击主机的数量及它们的加权值组成的威胁排序函数计算每个攻击者的威胁值,这样就可以实时获取 top-k 个对本地威胁最大的攻击者.本文首次提出基于结构化覆盖网的 top-k 联接查询算法,其中数据以关系模型的元组形式连续地插入到网络中.考虑到对等网中数据的分散性,如何减少来源于网络不同节点流数据的连接查询是一个非常有意义的课题,与此同时应尽量减少网络流量的产生.本文给出了两个相关算法,最后通过仿真实验进行了性能分析.

1　相关工作

P2P 网络中每个节点既充当客户端接受服务,又同时作为服务器为其它节点提供服务.节点在网络中的地位是等效的,具有相同的功能,没有主次区别.P2P 网络从架构上主要分为非结构化覆盖网和结构化覆盖网.非结构化覆盖网要么采用集中的服务器提供信息查询服务,存在单点故障,要么采用洪泛(flooding)搜索机制,查询从一个节点以广播方式发送到网络中其它节点,查询效率低下,同时产生大量网络带宽.但是网络结构简单,易于实现,包括 Napster、Gnutella、Freenet.结构化覆盖网络一般通过分布式散列表(DHT)[1]来组织网络节点和消息路由,每个节点都有一个惟一的标识(ID),而每一个存储对象以其对应的散列值作为惟一标识,节点标识的名字空间和对象标识的名字空间是一致的,Chord[2]、CAN[3]、Pastry[4]等属于结构化覆盖网络.在结构化覆盖网环境中实现联接查询,PIER[5]较早提出了在 DHT 环境中实行联接查询,但是这种查询假定数据是不变的一次性查询.PeerCQ[6]也是一个在 DHT 环境中实现联接查询的系统,PeerCQ 不是基于 SQL 语句的查询,同时数据也不是以关系模型存储在 DHT 环境中,而是保存在额外的数据源中.

top-k 查询在 web 搜索引擎中得到成功的应用,如 Google 搜索引擎将计算得到最匹配的 top-k 个搜索结果返回用户.Onion[7]和 Prefer[8]实现了在传统数据库对 top-k 查询的支持.Fagin 等[9]提出了如何在多媒体检索中有效实现 top-k 选择查询的算法,其中 TA 算法支持对表进行随机访问,而 NRA 算法只允许对表进行有序访问,多种改进算法随后相继被提出[10-12].Chang 和 Hwang[13]介绍了用来计算具有高代价谓词排序查询的 MPro 算法,通过判断高代价谓词求值是否必需来减少相关谓词的求值.Balke[14]将 top-k 查询引进到了 P2P 环境中,利用超级节点的本地索引来减少不必要的网络流量.

前面的研究都有一个假设,那就是在查询执行前所有的数据都是可知的,然而在流数据环境中,数据是无法预先获取的.当前流数据的连续查询研究大部分在集中式环境中展开.Mouratidis[15]介绍了在滑动窗口中对流数据执行 top-k 查询,通过利用少量额外存储提前计算部分结果,以获取到更佳的实效性.Idreos[16-17]提出了基于 DHT 的连续查询,他们采用了两级索引的方式将数据和查询发布到结构化覆盖网中.本文将在 Idreos 等人的基础上提出基于结构化覆盖网的连续 top-k 联接查询,如何减少 top-k 联接查询的网络流量和让更多的节点参与查询过程中是本文重点要解决的问题.

2　问题定义

在本文中,研究的环境是结构化对等网数据管理模型.结构化对等网的节点采用分布式散列表(DHT)组织起来,由于 Chord 具有协议简单等特点,使用 Chord 协议来实现这部分功能,但是系统模型不是局限于 Chord 协议的,系统能够方便的移植到其他的结构化覆盖网协议中.对于系统中的节点有如下假设:网络中的节点都是平等的,每个节点都可以发布查询和存储数据.在数据管理方面,使用关系数据模型,并且采用了和 PIER[5]相同的理念,就是不同模式可以共存于系统但是系统不支持模式间映射,假设数据处于一个全局的模式内.数据被组织成关系数据模型的元组[tuple]形式,连续地从不同的节点插入和保存到系统中,称之为关系数据流.

2.1　DHT 模型

采用 Chord[2]协议作为底层的 DHT 模型,下面简单的介绍一下 Chord 协议.Chord 中每个节点和数据都有一个关键字(key),通过哈希函数如 SHA-1,MD5 等,计算得到惟一的标识 hash(key),这些标识符共享一个长度为 m-bit 的名字空间.所有节点按照他们的标识被顺时针地组织成一个环状.关键字为 key 的数据将被保存到节点标识大于或等于 hash(key)的第一个节点中,该节点称为数据的后继节点,记为 Successor(hash(key)).查询和保存一个数据能够在 $O(\log(N))$ 跳数内完成,而每个节点只需要

保存非常小的路由信息,其中 N 是系统中节点的个数.

2.2 问题形式定义

本文介绍如何在结构化覆盖网中解决 top-k 联接查询的问题,连接查询部分主要讨论两表之间的等值联接(equi-join).下面是一个典型的传统数据库中 top-k 联接查询:

定义 2.1

Select $R.A1, R.A2, \cdots, R.Am, S.B1, S.B2, \cdots S.Bn$

From R, S

Where $R.Aj = S.Bj$

Order by Rank-f

Stop after k

该查询定义了在两表连接的基础上,通过计算排序函数的值,得到最符合用户需求的前 k 个查询结果.本文要做的就是将该查询在数据连续变化的结构化对等网中予以实现.在上面定义的查询中,R, S 是两个关系表:$R\{A1, \ldots Au\}$, $S\{B1, \ldots Bv\}$,其中 $1 \leq m \leq u, 1 \leq n \leq v$. $R.Aj, S.Bj$ 分别是表 R, S 的联接属性,不失一般性,我们定义表中属性的值域为[0,1].“Stop after k”表示将根据排序函数 Rank-f 计算返回前 k 个查询结果,即 top-k 个结果值.“Order by”定义了一个基于关系表 R, S 属性的排序函数 Rank-f,排序函数是用户对查询结果的一种偏好程度反应,排序函数 Rank-f 结果值越高,越符合用户的需求.假设排序函数 Rank-f 是单调的,单调排序函数定义如下:

定义 2.2 排序函数 Rank-f: $[0,1]^n \rightarrow \mathbf{R}$($\mathbf{R}$ 为实数集)是单调的当且仅当下面的条件成立:若 $ai \leq bi$ 对所有 i 成立 $(1 \leq i \leq n)$,那么 Rank-f $(a1, \ldots an) \leq$ Rank-f $(b1, \ldots bn)$ 也成立.

本文的研究的环境中,关系表的数据以元组的形式,连续地从不同的节点插入和保存到系统中,发布到系统中的 top-k 联接查询则等待新的数据来触发它们.系统中使用 NTP 协议用于保证系统时间的同步,可以达到微秒极的准确度.每一个元组都包含附加的一个时间信息 T(tuple),标识元组加入到系统的时间,同样每个查询也拥有一个发布时间信息 T(query),标识查询发布到系统时的时间.对于一个查询 q,只有当某关联元组 t 的 T(tuple) > = T(query)时,查询才能被该元组数据触发,进行连接计算.

最后,给出基于结构化对等网连续 top-k 连接查询的定义:

定义 3.3 结构化对等网连续 top-k 连接查询 R 和 S 为存储到 DHT 空间的关系表,Q 为基于关系表 R, S 的 top-k 联接查询,连续 top-k 连接查询随着元组数据连续插入,进行表连接,然后根据 Q 的排序函数 Rank-f 的结果值动态返回前 k 条查询记录.

3 连续 top-k 联接查询

传统数据库中联接查询本身就是一个耗费大量资源的查询过程,特别是在数据量较大的时候.对结构化覆盖网来说,元组数据分散在不同的节点上,相比传统数据库,进行联接查询将消耗更多的资源并且产生大量的通信流量.如何在这种分布式环境中让查询过程尽量减少联接的次数,进而减少资源的消耗和网络通信量是我们主要考虑的问题.本文提出了两种基于结构化覆盖网的连续 top-k 联接查询算法,算法一中,系统收到每条元组都进行联接查询,得到联接查询结果后计算排序函数,最后统计前 k 个结果返回给查询者.算法二中在进行联接查询前,计算排序函数的估值与现有 top-k 结果比较,去除不可能对最终结果产生影响的元组,减少了联接查询量.

3.1 元组,查询的发布

本小节描述如何在结构化对等网环境中发布元组和查询.采用一种两级索引结构[16]将元组和查询分发到系统中,其中第一级为属性级,第二级为属性值级.以定义 2.1 中的查询为例,当某个节点接收到关系 $R\{A1, \ldots Au\}$的一个元组时,该节点将把这个元组发布到两级节点中:首先元组将发布到第一级节点即属性级节点:Successor(hash($R + Ai$)),其中 $1 \leq i \leq u$;然后元组将被发布到第二级节点即属性值级节点:Successor(hash($R + Ai + Ai$.value)),其中 $1 \leq i \leq u$,Ai.value 为属性 Ai 的值.如图 1 所示.

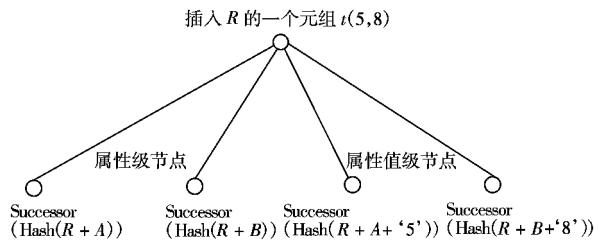

图 1 元组的发布

Fig. 1 Tuple publishing

关系表 R 包含两个属性 A, B,当 R 的一个元组 $t(5,8)$ 插入到系统时,首先元组被发布到属性级节点 Successor(Hash($R + A$)) 和 Successor(Hash($R + B$)) 上.然后该元组发布到两个属性值级节点 Suc-

cessor(Hash($R+A+$'5')) 和 Successor(Hash($R+B+$'8')) 上,这样元组 t(5,8) 就被存储到了 4 个节点上.

当某个节点提交一个连续 top-k 联接查询时,我们称这个节点为该查询的发布节点,同样以定义 2.1的查询为例,标记该查询为 Q. 查询 Q 将首先发送到下面两个属性级节点:Successor(hash($R+R.Aj$)) 和 Successor(hash($S+S.Bj$)),标记这两个节点为 rewriter R,rewriter S,其中 $R.Aj$,$S.Bj$ 分别是表 R,S 的联接属性,这两个节点称作该查询 Q 的查询重写器,因此每个查询拥有两个查询重写器,各自负责一个关系表,等待该关系表的元组数据插入. 当元组数据到达某个查询重写器并满足触发联接查询的条件(触发条件在下面的算法中讨论)时,联接查询将被重写为一个简单的选择查询(select query),发布到属性值级节点等待满足该查询的元组的插入. 图 2 说明了一个完整的查询分发过程,查询 Q:

select *
From R,S
Where $3R.A+0.4=6S.C+0.1$
Order by ($3*R.A+2*R.B+5*S.D$)
Stop after 10

是基于关系 $R(A,B)$ 和 $S(C,D)$ 的一个连续 top-k 连接查询. 首先,Q 发布到了两个重写器 $N1$:Successor(hash($R+A$)) 和 $N2$:Successor(hash($S+C$)) 上,等待元组数据的触发. 当 R 的一个元组 $t1$(0.5,0.7) 到达 $N1$ 并满足触发条件时,触发查询 Q,计算连接条件:"Where $3R.A+0.4=6S.C+0.1$",可知 $S.C=0.3$ 的元组符合连接条件,重写 Q 为简单选择查询 $rQ1$,注意排序函数 Rank-f 中已知的属性已被实际值代替:

Select 0.5,0.7,$S.C$,$S.D$
From S
Where $S.C=0.3$
Order by ($3*0.5+2*0.7+5*S.D$)
Stop after 10

$rQ1$ 发布到二级属性级节点 $N3$:Successor(hash($S+C+$'0.3')) 等待符合条件的元组. 当 S 的元组 $t2=S$(0.3,0.6) 被保存到系统时,节点 $N2$,和 $N3$ 将同时收到该元组. 在 $N3$ 节点上,该元组将触发简单查询 $rQ1$,一次连接完成,将返回一个连接结果;在节点 $N2$ 上若满足触发条件,同样将触发 Q,重写 Q 为一个简单选择查询 $rQ2$:

Selet $R.A$,$R.B$,0.3,0.6
From R
Where $R.A=0.5$
Order by ($3*R.A+2*R.B+5*0.6$)
Stop after 10

发布到属性值级节点 $N4$:Successor(hash($R+A+$'0.5')) 上等待 $R.A=5$ 的元组.

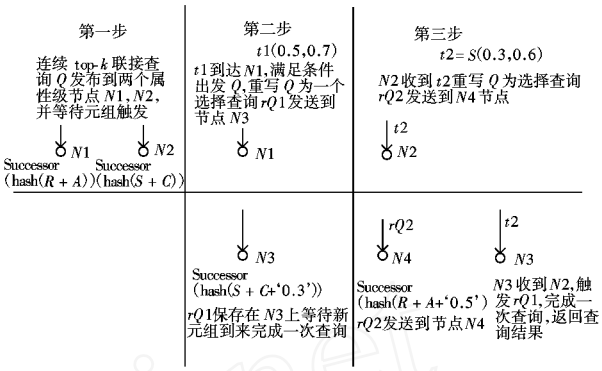

图 2　查询过程

Fig. 2　Processes of query

3.2　算法一

对于 3.1 中的连续 top-k 连接查询,首先想到的一个算法是对所有插入的数据都试图做连接查询,也就是查询的触发是无条件的,即只要有新的元组插入都将触发查询,重写联接查询为选择查询. 在二级属性节点当有新元组满足被转换后的选择查询时,连同排序函数 Rank-f 的排序值返回给查询发布节点. 发布节点根据收到查询结果的排序值比较比较更新 top-k 结果队列. 具体算法如下:

(1) 当查询重写器收到新元组时,将触发联接查询,重写查询发布并保存到第二级节点上.

(2) 在第二级节点上,新元组插入后查找本地是否有重写查询,若有并满足查询条件,将查询结果发回到查询发布节点.

(3) 查询发布节点将收到查询结果的排序函数 Rank-f 的排序值与本地的 top-k 结果队列的比较,若该排序值大于队列中第 k 项的排序函数值,则更新 top-k 结果队列.

在该算法中,对于每个连续 top-k 联接查询 Q,发布 Q 到一级节点需要 $2*(O(\log(N))$ 跳,因为每个元组数据都会触发查询,那么查询 Q 将会总共需要 $(2*(O(\log(N))+N\text{tuple}*O(\log(N))$ 跳数完成发布,其中 Ntuple 为晚于查询 Q 到达系统的所有元组的总数. 因为 top-k 的结果数据毕竟是少数的,因此绝大部分联接查询对最终的 top-k 结果并没有贡献,该算法产生了大量无用的网络流量.

3.2　算法二

算法一中需要对所有的元组都试图做连接查

询,我们将考虑通过预计算排序函数 Rank-f 的估值减少不必要的连接查询次数:当新元组到达查询重写器时,首先用该元组预计算排序函数可能最高值(通过将排序函数 Rank-f 中未知属性值设为值域的上界),与 top-k 结果队列中第 k 项(kth)数据的排序值比较. 若前一项小于后一项,说明该元组不会对最终的 top-k 数据产生影响,因此就可以不对该元组做联接查询,这样就可以减少不必要的网络消耗和节点计算资源消耗. 考虑到均衡节点负载和流量,我们将查询的 top-k 计算部分转移到查询重写器上. 每个查询重写器都有一个自己 top-k 结果子队列,而每个查询有两个查询重写器,最后我们将这两个 top-k 结果子队列发送到查询发布节点合并为最终的 top-k 结果队列. 考虑到两个 top-k 结果子队列是各自独立工作的,因此两队列中排序函数结果值有可能会相差较大,通过同步最终的 top-k 结果队列 kth 数据到两个查询重写器,这样可以进一步减少了数据联接的联接量. 具体算法描述如下:

(1) 两个查询重写器各维护一个 top-k 结果子队列,取 V = kth 项排序函数值(V 初始值设为 0. 若 kth 项不存在,不赋值,这种情况存在于查询初始阶段和后面提到 kth 同步阶段).

(2) 当新元组插入时,重写器节点上,设定排序函数中未知属性为值域的上界 1,计算排序函数与 V 比较,若大于等于 V,则该元组参与的联接查询有可能成为最终结果,重写查询发布并保存到第二级节点上;若小于 V,不触发查询.

(3) 在二级节点上,新元组插入后查找本地是否有重写查询,若有并满足查询条件,将查询结果发回到对应的查询重写器.

(4) 查询重写器收到二级节点返回的查询结果后,计算排序函数值与本地 V 值比较,若大于 V 值,将该查询结果发送到查询发布节点,同时更新本地 top-k 结果子队列.

(5) 查询发布节点拥有一个最终 top-k 结果队列,当接收到查询重写器发回的查询结果时,比较排序,更新 top-k 结果队列.

(6) kth 同步:当查询发布节点的 top-k 结果队列 kth 项变化时,将该 kth 项排序函数结果值 new$V$ 同步到两个查询重写器. 重写器用 new$V$ 与本地 V 值比较:如果 V < new$V$,则将 new$V$ 赋值给 V,同时将新的 V 值与本地 top-k 队列项的排序函数值比较,将值小于 V 的项去除.

在该算法中,根据预计算排序函数 Rank-f 的估值我们将裁减掉不可能对最终 top-k 结果产生影响的元组连接. 发布每个连续 top-k 联接查询 Q 总共需要 (2 * (O(log(N)) + (1 - P) * Ntuple * O(log(N))跳数,其中 P 为不可能对最终 top-k 结果有贡献的元组百分比,也就是元组裁减比,若 P 值较大,将减少大量不必要网络流量,在下面的仿真实验中我们将计算较算法一和算法二的性能.

4 仿真及分析

为了对本文提出算法的性能进行评估,我们使用 JAVA 编写了模拟器进行仿真测试. 底层的结构化覆盖网采用 Chord 协议实现,模拟 10 000 个节点工作. 为了测试关系表属性值在不同数据集上的性能表现,我们分别在两种不同的数据集上进行评测:

均匀随机数据(Random Data):属性值在值域[0,1]间均匀随机分布.

高斯数据(Gaussian Data):属性值以 0.5 为中心值,标准偏差为 0.16 的高斯分布.

分布式环境中,网络带宽的消耗是主要考虑的评价指标. 本文提出的两个算法中,算法一是比较原始的算法,它对所有的元组数据都试图进行联接查询,产生了大量网络流量. 而在算法二中,通过裁剪掉那些不会对最终 top-k 结果产生影响的元组联接,达到减少网络带宽的消耗的目的. 因此我们使用在联接查询前被裁剪元组所占的百分比 P,来评价对网络带宽消耗的节省. 对于算法一和算法二在均匀随机数据集和高斯数据集上得到的裁减比 P 在图中分别标识为 Random 算法一, Gaussian 算法一, 以及 Random 算法二, Gaussian 算法二.

图 3 模拟的是随着插入元组量的变化,元组裁剪比 P 的变化比较. 在这个实验中,设定 k 值为 10, 在两个数据集上,元组插入量从 10 000-100 000 变化,每个表各有两个属性参与排序函数. 由于算法一对所有的元组都做联接查询,所以它对应的裁剪比 P 一直为 0 %,即没有做任何的元组裁减,后面的实验我们主要关注算法二在两个不同数据集上的性能变化. 图中显示随着插入到系统中元组数的增加,算法二中裁剪掉的元组比也随之增加,最后裁剪比 P 趋于稳定. 可以发现,在高斯数据集上的性能表现相比平均随机数据集要差,这是因为高斯数据集的值大部分集中在中间值 0.5 附近,而在做裁剪比较时,需要将另外一个表的未知属性值都设为最大值 1, 这样排序函她的估值就会变大,因此将有更多的元组参与到联接查询中,从指标上体现出来的就是元组裁剪比会较低. 对于随机数据集,当元组量增大

时,裁剪比能够稳定在 90 %以上,具有很好的性能.

图 4 显示了不同 k 值的情况元组裁剪比的变化比较.这个实验中,设定每个表各有两个属性参与排序函数,k 从 1～20 间变化,对于每个 k 值向系统中插入了 100 000 个元组数据进行测试.从图中可以明显看出来,随着 k 值的变大,元组裁剪比随之减少,对于高斯数据集,衰减的速度更快.这是由于由于随着 k 值的变大,top-k 结果队列中第 k 项的排序函数值变得更小,将造成更多的元组参与到联接计算中,元组的裁剪比也就随着降低.对于两个不同的数据集,与前一个实验同样的原因,针对均匀随机数据集系统仍然表现出更好的性能,特别是在前半部分裁剪比稳定地保持在 80 %以上.

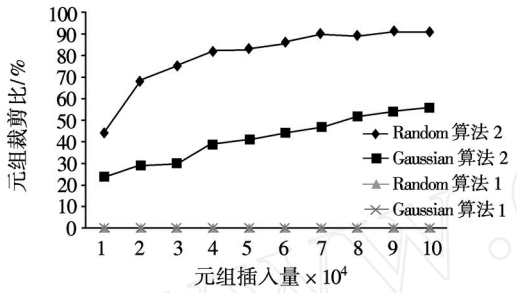

图 3　裁剪比随元组量变化图

Fig. 3　The number of tuples' effects

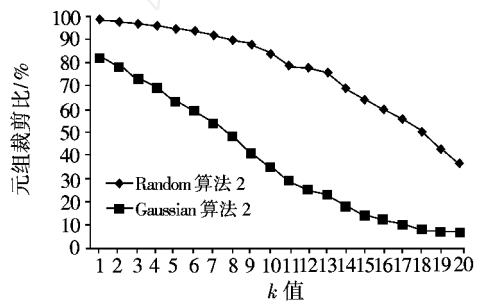

图 4　裁剪比随 k 值变化

Fig. 4　Effects of different k

图 5 显示的元组裁剪比随排序函数属性数变化的过程.这个实验中,仍然取 100 000 个元组数据做为测试样本,k 为 10.每个表参与排序函数的属性数从 1～10 递增测试,也就是说排序函数中全部属性数变化范围是 2～20.元组裁剪比随着属性数的增加而减少,这个现象也是很容易解释的,因为随着较多的属性参与排序函数,未知属性将被设为 1 的数量也将变多,因此排序函数的估值变得更大,这样更多的元组就参与到查询过程,元组的裁剪比也就随之下降.对于两个数据集,均匀随机数据依旧表现要强于高斯数据,道理和前面的实验是一样的.

实验结果表明,相比较原始的算法一不做任何的裁减处理,算法二能够有效的减少不必要的连接次数,减少不必要的网络流量.

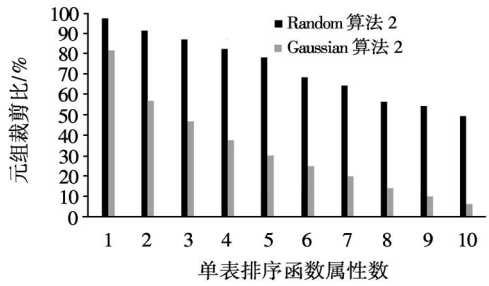

图 5　裁剪比随排序函数属性数影响

Fig. 5　Effects of ranked-function's attributes

5　结论

本文提出了基于结构化覆盖网实现连续 top-k 联接查询的算法.联接查询本身需要耗费大量计算资源,同时在结构化覆盖网这种分布式环境中,实现这种查询还需要考虑如何减少网络通信量的问题.提出的解决方案中,在对新插入的元组数据做联接查询前,首先计算该元组排序函数值的最大估值与现有结果进行比较,对那些不可能对最终 top-k 结果产生影响的元组数据予以裁剪,大大减少了网络的流量和计算量.经过实验仿真分析,该方案通过减少不必要的联接查询能够很好的减少网络的流量和计算负载.到目前为止,只讨论了基于两表之间等值联接(equi-join)的 top-k 查询,在实际应用环境中更复杂的基于多表的 multi-way 连接也有相当使用,而在分布式环境中完成多表的 multi-way 连接将更加复杂.因此在结构化对等网中实现多表 multi-way 连接的 top-k 查询将是下一步研究方向.

参考文献:

[1] KARGER D, LEHMAN E, LEIGHTON F, et al. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web[C]// Proc of the 29th Annual ACM Symp on Theory of Computing. New York: ACM, 1997:654-663.

[2] STOICA I, MORRIS R, KARGER D, et al. Chord: a scalable peer-to-peer lookup service for internet applications[C]// Annual Conf of the Special Interest Group on Data Communication. New York: ACM, 2001:124-137.

[3] RATNASAMY S, FRANCIS P, HANDLEY M, et al. A scalable content-addressable network[C]// SIGCOMM'01. New York: ACM, 2001:168-175.

[4] ROWSTRON A, RUSCHEL P. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems[C]// Int'l Conf on Distributed Systems Platforms. New York: ACM, 2001:135-141.

(下转第 57 页)

building thermal mass[J]. Building and environment, 2007, 43(10): 1633-1646.

[9] 付祥钊,张慧玲,黄光德. 关于中国建筑节能气候分区的探讨[J]. 暖通空调,2008,38(2): 44-47,17.
FU X Z, ZHANG H L, HUANG G D. Discussion of climatic regions of building energy efficiency in China[J]. HV&AC, 2008, 38(2): 44-47,17.

[10] BRAUN K, CHATURVEDI N. An inverse gray-box model for transient building load prediction[J]. HVAC&R Research, 2002, 8(1): 73-99.

[11] LIAO Z, DEXTER A. A simplified physical model for estimating the average air temperature in multi-zone heating system[J]. Building and Environment, 2004, 39: 1013-1022.

[12] WANG S W, XU X H. Simplified building model for transient thermal performance estimation using GA-based parameter identification[J]. International Journal of Thermal Sciences, 2006, 45: 419-432.

[13] 山东省墙材革新与建筑节能办公室,山东建筑大学,山东省建筑科学研究院,等. DBJ14-036-2006 公共建筑节能设计标准[S]. 济南: [出版者不详], 2006.
Shandong Province Wall Materials Innovation &Energy Saving Office, Shandong Jianzhu University, Shandong Provincial Academy of Building Research, et al. DBJ14-036-2006 Design standard for energy efficiency of public buildings[S]. Jinan: [s. n.], 2006.

[14] 杨维,李歧强. 粒子群优化算法综述[J]. 中国工程科学, 2004, 5: 87-94.
YANG W, LI Q Q. Survey on particle swarm optimization algorithm[J]. Engineering Science, 2004, 5: 87-94.

[15] 中国建筑科学研究院. GB50176-93 民用建筑热工设计规范[S]. 北京: 中国计划出版社, 1993.
China Academy of Building Research. GB50176-93 Civil buildings thermal design specification[S]. Beijing: China Planning Press, 1993.

(编辑:孙培芹)

(上接第 37 页)

[5] HUEBSCH R, HELLERSTEIN M, LANHAM N, et al. Querying the internet with pier[C]// VLDB '02. Berlin: VLDB Endowment, 2002: 321-332.

[6] GEDIK B, LIU L. PeerCQ: A decentralized and self-configuring peer-to-peer information monitoring system[C]// 23rd IEEE International Conference on Distributed Computing Systems, USA: IEEE Computer Society, 2003.

[7] CHANG Y, BERGMAN L, CASTELLI V, et al. The onion technique: indexing for linear optimization queries[C]// SIGMOD2000. New York: ACM, 2000: 391-402.

[8] HRISTIDIS V, PAPAKONSTANTINOU Y. Algorithms and applications for answering ranked queries using ranked views[J]. VLDB Journal, 2004, 13(1): 49-70.

[9] FAGIN R, LOTEM A, NAOR M. Optimal aggregation algorithms for middleware[C]// In PODS. New York: ACM, 2001: 102-113.

[10] ChANDHURI S, GRAVANO L, MARIAN A. Optimizing top-*k* selection queries over multimedia repositories[J]. TKDE, 2004, 16(8): 992-1009.

[11] ARIAN A, BRUNO N, GRAVANO L. Evaluating top-*k* queries over web-accessible databases[J]. TODS, 2004, 29(2): 319-362.

[12] THEOBALD M, WEIKUM G, SCHENKEL R. Top-*k* query evaluation with probabilistic guarantees[C]// VLDB '2004. New York: Springer-Verlag, 2004: 648-659.

[13] CHANG K C, HWANG S. Mining probing: supporting expensive predicates for top-*k* queries[C]// SIGMOD '02. New York: ACM, 2002: 346-357.

[14] BALKE W, NEJDL W, SIBERSKI W, THADEN U. Progressive distributed top-*k* retrieval in peer-to-peer networks[C]// ICDE '05. Washington: IEEE Computer Society, 2005: 174-185.

[15] MOURATIDIS K, BAKIRAS S, PAPADIAS D. Continuous monitoring of top-*k* queries over sliding windows[C]// SIGMOD '06. New York: ACM, 2006: 635-646.

[16] IDREOS S, TRYFONOPOULOS C, KOUBARAKIS M. Distributed evaluation of continuous equi-join queries over large structured overlay networks[C]// ICDE '04. Washington: IEEE Computer Society, 2005: 43-54.

[17] IDREOS S, LIAROU E, KOUBARAKIS M. Continuous multi-way joins over distributed hash tables[C]// EDBT '08. New York: ACM, 2008: 594-605.

(编辑:陈丽萍)