对等网络环境下数据管理系统上的 Top- k 查询

李继良
(徐州建筑职业技术学院电子信息工程系 江苏 徐州 221008)

【摘 要】: 对等网络(Peer- to- Peer)模型是一种新型的体系结构模型,许多优势有待于进一步发掘,拥有广阔的应用前景。提出了一种在 P2P 环境下共享数据库的新框架:基于关键词查询的数据库共享。将每个节点上的数据库看成是一个文档集,用户不用考虑数据库的模式结构信息,简化了不同节点数据库模式间的映射过程,能够较好地适应 P2P 分散和动态的特性。

【关键词】: Peer- to- peer,数据库共享,关键词查询,Top- k 查询,直方图,邻居节点自调整

1. 引言

随着计算机技术的发展,在当今软硬件技术环境下,客户/服务器模型已不能满足需求, 其单点故障和热点问题已经变得越来越不可接受。Peer- to- Peer 模型(又称 P2P 模型或对等计算模型)是一种新型的体系结构模型,具有许多优势有待进一步发掘[1]:①P2P 系统的每个成员均可贡献数据和计算资源(如未用的周期和存储资源),新成员的加入可能引入系统中原来缺乏的特殊数据或资源,因此随着系统成员增加,系统的丰富性、多样性等各种有益的特性得以扩大;②系统具有分散性,因此系统的健壮性、可用性和性能可能随着数量的增加而扩展;通过在许多间路由请求和复制内容, 系统可以隐藏数据提供者和消费者的身份,使个人的隐私得到保护[1]。因此 P2P 被认为是未来重构分布式体系结构的关键技术[2]。P2P 在搜索引擎、数据流管理、语义、协作信息过滤等领域具有广阔的应用前景。

本文提出了一种在 P2P 环境下共享数据库的新框架: 基于关键词查询的数据库共享。将每个节点上的数据库看成是一个文档集,用户不用考虑数据库的模式结构信息,简化了不同节点数据库模式间的映射过程, 能够较好地适应 P2P 分散和动态的特性。我们将 Top- k 查询扩展到 P2P 环境下的数据管理系统上,将文档集和数据库的查询统一起来,一致对待。文档集上基于直方图的分层 Top- k 查询算法 [4]同样适用于 P2P 环境下数据库上的 Top- k 查询。随着查询的进行,直方图可以自动更新, 同时根据查询的结果对邻居节点进行调整,把那些真正包含 Top- k 结果的节点作为查询发起节点的邻居,具有自适应性。基于关键词的数据库共享突破了传统的数据库共享模式,简化了数据访问方式,而基于直方图的 Top- k 查询算法提高了查询的效率。

2. P2P 环境下的数据管理系统

2.1 SEEKER 简介

SEEKER 的体系结构如图 1 所示。SEEKER 的查询结果是以数据库中的元组为结点的一棵树:连接元组树。树中任意两个相邻的元组间的边是主码- 外码连接关系,连接元组树包含至少一个查询中的关键词且每个叶结点都包含至少一个查询中的关键词。

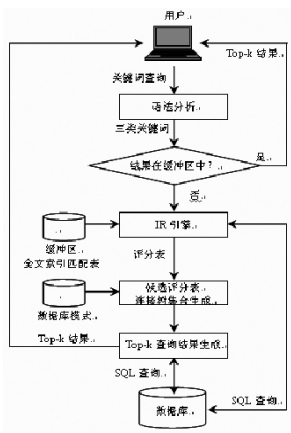


图 1 SEEKER 的体系结构

2.2 P2P 环境下基于关键词查询的数据库共享

当用户提交一个关键词查询, 搜索 P2P 网络中与关键词最为匹配的 Top- k 个结果返回给用户。P2P 系统中每个共享数据库的节点都提供一个关键词查询的接口, 不同节点返回的关键词查询结果合并生成最终的结果。图 2 给出了 P2P 环境下数据管理系统的体系结构。系统共分为四层: P2P 层、关键词检索层、Top- k 查询层和数据库管理层。

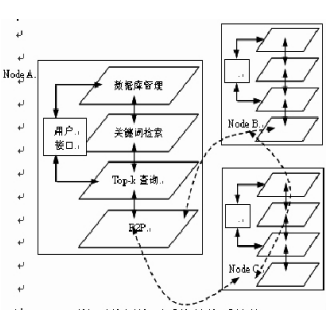


图 2 P2P 环境下数据管理系统体系结构

3. P2P 环境下数据库上的 Top- k 查询

我们首先介绍本地数据库上基于关键词的 Top- k 查询,然后应用基于直方图的分层 Top- k 查询算法进行 P2P 环境下的数据管理系统上的 Top- k 查询。

3.1 本地数据库上基于关键词的 Top- k 查询

本地数据库的 Top- k 查询由 SEEKER 来完成, 下面是 SEEKER 的一些基本定义。

一个包含 n 个关系 $R_1, \cdots, R_n$ 的数据库 DB, 可以将它看成一个有向图 G。DB 中的每一个关系就是 G 的一个顶点,每一个主码- 外码关系就是 G 的一条有向边, 方向由主码所在的关系 $R_i \in G$ 指向外码所在的关系 $R_j \in G$, 记为 $e(R_i, R_j)$, 将 G 称为模式图 (Schema Graph)。一个关键词查询 Q 由一组关键词 $kw_1, \cdots, kw_m$ 组成,表示为 $Q(kw_1, \cdots, kw_m)$。

定义 1 连接元组树 TT 是以数据库 DB 中的元组为结点的一棵树,树中任意两个相邻的元组 $t_i \in R_i$, $t_j \in R_j$, 它们之间的边 $e(t_i, t_j)=e(R_i, R_j) \in G$ 且 $(t_i \triangleright \triangleleft t_j) \in (R_i \triangleright \triangleleft R_j)$。TT 的大小是它拥有的元组的数量,记为 sizeof(TT)。

定义 2 查询结果是含至少一个 Q 中的关键词且每个叶结点都含至少一个 Q 中的关键词的连接元组树。

SEEKER 只将 Top- k 查询结果返回给用户, 因此需要给查询结果评分,然后根据分数高低来对查询结果排序。查询结果所含关键词的数量不尽相同,显然含关键词越多,得分应该越高。SEEKER 采用了以下的评分公式:

$$Score(TT, Q) = \frac{1}{sizeof(TT)} \left(\frac{m'}{m}\right)^{a} \sum_{i=1}^{sizeof(TT)} \sum_{j=1}^{m} Score(T_i, kw_j) \quad (1)$$

其中, TT 是构成查询结果的一个连接元组树, sizeof (TT)是 TT 中所含元组的个数,它与该结果的得分成反比, T_i 是 TT 中的元组, Q 是一个关键词查询, m 是 Q 中关键词的个数, $kwj \in Q$, m' 是 TT 中所含关键词的数量, a 是常数,取大值(例如 10)来保证含关键词多的查询结果比含关键词少的查询结果的得分高, Score (Ti, kwj)是元组 T_i 对于关键词 kwj 所得的分数。

3.2 P2P 环境下数据管理系统上的 Top- k 查询

本文只考虑针文本属性的关键词查询, 暂不考虑元数据和数值属性的关键词查询。

3.2.1 分层的 Top- k 查询

纯的 P2P 网络的搜索是一种广播式的搜索, 节点除了进行本地搜索还将查询被广播发送给它的所有邻居节点。搜索范围由查询跳数 TTL 给出,查询每转发一次, TTL 减 1,当 TTL 等于 0 时,就停止传播查询。为了防止回路的产生,每个查询消息都有一个唯一的编号。如果节点先前已经收到查询,则说明发生了回

路，节点不再继续传播查询。我们可以把这一查询过程用一棵查询树来表示，如图 3 所示。图中节点 a 发起查询，为查询树的根节点。查询被广播发送给节点的所有邻居节点，每次转发 TTL 减 1，TTL 为 0 或者没有邻居的节点为叶子节点。

我们定义 P2P 环境下数据管理系统上的 Top–k 函数为：

$$Top\text{–}k(P, Q, TTL) = \max_k(\{Local\text{–}Top\text{–}k(P_i, Q) \mid P_i \in QueryTree(P, Q, TTL)\}) \quad (2)$$

其中，Local–Top–k(P_i, Q)为节点 P_i 的本地 Top–k 函数，采用 SEEKER 的评分公式(公式 1)作为本地的 Top–k 函数。maxk 为全局的 Top–k 函数，比较查询树中所有节点的 Top–k 结果，取其中分数最大的 k 个元组连接树作为最终的 Top–k 结果返回。SEEKER 需要使用 RDBMS 给出的分数（即公式 1 中 Score(T_i, kw_j)），而不同的 RDBMS 采用的评分方法可能是不一样的。

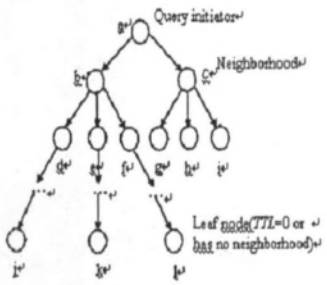


图 3 查询树

基于查询树，我们分层计算 Top–k 函数，将结果的排序和合并分布到网络中的各个节点上，实现分布式的 Top–k 查询。父亲节点除了进行本地的 Top–k 查询，还要聚集儿子节点返回的 Top–k 结果，产生最优的 Top–k 结果。叶子节点(TTL 为 0 或者没有邻居的节点)仅仅进行本地的 Top–k 查询，将 Top–k 结果返回给它的父亲节点。整个查询过程自底向上逐层进行，直到根节点得到最终的 Top–k 结果。例如：图 3 中，节点 j 查询本地信息返回本地的 Top–k 结果给父亲节点。节点 b 聚集节点 d，e，f 返回的 Top–k 结果以及本地的 Top–k 结果，产生最优的 Top–k 结果，继续返回给节点 a。节点 a 聚集节点 b 和节点 c 返回的 Top–k 结果以及本地的 Top–k 结果，产生最终的 Top–k 结果，返回给用户。

3.2.2 直方图的建立

广播式的搜索是低效的，我们采用直方图的方法提高搜索的效率，根据 Peer(节点)返回的 Top–k 结果为 Peer 构建直方图，直方图中存储的是查询关键词的分数上限。利用直方图为将来的查询估计 Peer 可能的分数上限，即查询树中以 Peer 为根的子树所包含的元组连接树的分数上限，对 Peer 进行选择，裁剪查询树中的一些无用分枝(不含真正的 Top–k 结果)，以提高搜索的效率。

观察 SEEKER 的评分公式(式(1))，我们可以在计算 Score(TT, Q)的过程中为每个查询关键词 kwj 计算一个分数，计算公式如下：

$$Score(TT, kw_j) = \frac{1}{size(TT)} \sum_{l=1}^{size(TT)} Score(T_l, kw_j) \quad (3)$$

$$Score(TT, Q) = \left(\frac{m'}{m}\right)^{\alpha} \sum_{j=1}^{m} Score(TT, kw_j) \quad (4)$$

Score(T_i, kw_j)是 RDBMS 的全文索引给出的一个分数。例如，Oracle9i 给出的是一个位于区间[0, 100]的值。对 Score(T_i, kw_j)进行归一化，除以系统的最大分数，将 Score(T_i, kw_j)映射到区间[0, 1]。由公式 3，Score(Ti, kwj)的取值范围一定在区间[0, 1]内。

性质 1 SEEKER 的评分公式

$$Score(TT, Q) = \left(\frac{m'}{m}\right)^{\alpha} \sum_{j=1}^{m} Score(TT, kw_j)$$

是单调递增的，即给定两个元组连接树 TT_1 和 TT_2，假如对查询 Q 包含的所有 kw_j，都有

$$Score(TT_1, kw_j) \geq Score(TT_2, kw_j)$$

则

$$Score(TT_1, Q) \geq Score(TT_2, Q)$$

假设 Peer 返回的 Top–k 结果不仅包含元组连接树的分数，而且包含查询中每个关键词的分数。根据性质 1，我们可以利用 Peer 返回的 Top–k 查询结果构建 Peer 的直方图。由于 P2P 环境下的查询与查询跳数相关，因此我们为 Peer 建立不同跳数的直方图。设查询 Q 在 Peer A 上的跳数为 TTL，Peer A 返回的与查询 Q 最相似的 k 个元组连接树为 $\{TT_1, TT_2 \cdots, TT_k\}$，Score($TT_i$, kwj)表示元组连接树 TT_i 对关键词 kw_j 的分数，Score(TT_i, kw_j)已经归一化。对 Q 中的每个关键词 kwj，我们计算 kwj 在 Top–k 结果中的分数上限，即，构建 Peer A 跳数为 TTL 的直方图为：

$$Histogram_{A,TTL} = \{(kw_j, Score_{kw_j,A,TTL} = \max_{1 \leq i \leq k} Score(TT_i, kw_j) \mid kw_j \in Q\} \quad (5)$$

计算 Peer A 相对于查询 Q 的分数上限

$$UpperScore_{Q,A,TTL} = \sum_{j=1}^{m} Score_{kw_j,A,TTL}$$

，由性质 1，可以保证 $UpperScore_{Q,A,TTL} \geq Score(TT_i, Q), 1 \leq i \leq k$。我们用 $UpperScore_{QATTL}$ 估计以 Peer A 为根的子树 QueryTree (A, Q, TTL)所包含的元组连接树的分数上限。

对将来的查询 Q′（假设查询 Q′在 Peer A 上的跳数为 TTL′），我们可以利用直方图估计 Peer A 的分数上限，采用的计算公式为：

$$UpperScore_{Q',A,TTL'} = \left(\frac{m''}{m}\right)^{\alpha} \sum_{kw_j \in Q' \wedge kw_j \in Histogram_{A,TTL} \wedge TTL = \min(TTL_1 \mid TTL_1 \geq TTL')} Score_{kw_j,A,TTL} + \sum_{kw_j \in Q' \wedge \neg\exists Histogram_{A,TTL}(kw_j \in Histogram_{A,TTL} \wedge TTL \geq TTL')} 1 \quad (6)$$

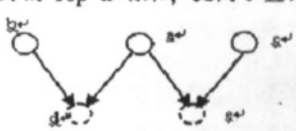
式中，m 表示查询 Q′包含的关键词个数，m″表示 Peer A 跳数大于等于 TTL′的直方图中包含 Q′的关键词个数。对 Q′中的每个关键词 kwj，假如 Peer A 的直方图中存在跳数大于等于 TTL′的直方图并且包含 kwj，我们取跳数与 TTL′最接近的直方图中的分数上限 Scorej,A,TTL 计算 Peer A 的分数上限；否则，我们取最大分数 1 计算 Peer A 的分数上限。由于经过归一化的 Score(Ti, kwj)取值范围在 0 到 1 之间，取最大分数 1 能够保证估计的分数上限一定大于实际的分数上限。

3.2.3 邻居节点自调整

利用直方图对 Peer 进行选择，删除了查询树中一些无用的分枝，提高了搜索效率，但只对搜索路径进行了选择，并没有对查询路径进行优化。假如真正的 Top–k 结果在查询树的叶子节点，我们仍然需要遍历从根节点到叶子节点查询路径上的所有节点。为了更快地找到真正含有 Top–k 结果的目标节点，我们需要根据查询结果对查询路径进行优化，这就需要节点能够动态地调整它的邻居节点。

假如返回的 Top–k 结果不光包含结果的分数，还包含结果的源节点(来自哪个节点)，则我们可以根据 Top–k 结果调整节点的邻居，使得节点与真正包含 Top–k 结果的节点直接相连。调整邻居的过程如图 4 所示。

图中，节点 d，e 包含真正的 Top–k 结果，配置前，节点 a 要取得节点 d，e 的结果需要首先访问节点 b，c。根据返回的 Top–k 结果重新配置邻居后，节点 d，e 成为节点 a 的邻居节点，节点 b，c 不再直接与节点 a 相连，这样，查询 q 可以直接发送到目的节点 d，e，减少了访问节点 b，c 的过程。邻居节点的重配置一方面缩短了查询路径，减少了访问节点的数目，提高了查询的效率；另一方面，由于 P2P 网络中的查询都有一个跳数的限制，调整邻居节点进一步扩展了搜索的范围，原来需要 n 次转发才可以找到节点现在只需要 1 步就能找到，可以搜索到网络中先前没有访问到的节点，得到更多更好的 Top–k 结果，提高了查询的精度。



(a) 配置前，节点 d，e 包含真正的 Top-k 结果　(b) 配置后，节点 d，e 成为节点 a 的邻居节点

图 4 根据 top–k 结果配置邻居节点

4. 结论

P2P 环境下的数据库共享是 P2P 目前研究 （下转第 93 页）

3.2.1 分组划分函数

PreTreatment 函数根据消息 M(注意:这里的 M 既可能是初始消息,也可能是剩余消息)长度的不同进行预处理。伪代码如下:

```
Function PreTreatment(M:String, Context: MD5Content)
Begin
If Length(M)<=56 then
Begin
//将消息拷入分组存储区中
  CopyMemory(@Context.Buffer[0], @pChar(M)[0], Length(M));
//填充附加比特,其中 Padding 为一个含 64 个元素全为零的 BYTE 数组
CopyMemory(@Context.Buffer[Length(M)], @Padding, 56- Length(M));
//附加消息长度
  CopyMemory(@Context.Buffer[56], @Context.Len, 8);
End;
If (Length(M)>56) and (Length(M)<=64) then
Begin
//填充当前分组
    CopyMemory(@Context.Buffer[0], @pChar(M)[0], Length(M));
CopyMemory(@Context.Buffer[Length(M)],@Padding,64- Length(M));
//由于只定义了一个 512bit 大的分组存储区,故需先对当前分组进行处理后再将四个中间变量代入下一个分组。
    GroupDisposal(@Context.Buffer, Context.ChaVar);
     //对新增的分组进行填充和附加长度     CopyMemory(@Context.Buffer[0], @Padding[64- Length(M)], 56);
     CopyMemory(@Context.Buffer[56], @Context.Len, 8);
End;
End;
```

3.2.2 报文分组逻辑处理函数

GroupDisposal 函数用于对每个报文分组进行逻辑处理,具体流程见图 3。其中,AA, BB, CC, DD 为处理过程中所用到的中间变量,分别被赋予 ChaVar[0]~[3]的值。

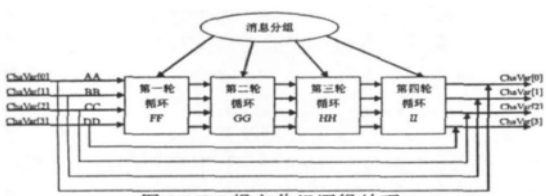


图 3 MD5 报文分组逻辑处理

伪代码如下:

```
Procedure GroupDisposal(Buffer: Pointer; Var ChaVar: MD5ChaVar);
Var
 AA, BB, CC, DD: DWORD;//定义四个中间变量
Begin
   // Evaluate 函数将四个链接变量分别赋值给四个中间变量
 Evaluate(AA,BB,CC,DD, ChaVar[0],ChaVar[1],ChaVar[2],ChaVar[3]);
   MainLoop(AA、BB、CC、DD);
  //将得到的中间变量赋值给链接变量
Evaluate(ChaVar[0],ChaVar[1],ChaVar[2],ChaVar[3],AA,BB,CC,DD);
End;
```

其中 MainLoop 为主循环运算函数,由 FF、GG、HH、II 四个循环函数构成,每个循环函数运算 16 次,共 64 次。下面为 FF 函数为例进行介绍(其余的三个循环函数与此类似)。

```
Procedure FF (Var A: DWORD; BB, CC, DD, Mj: DWORD; Si: BYTE; Ti: DWORD); Begin
Inc(A, F(BB, CC, DD) + Mj + Ti);
LeftRot(A, Si);
Inc(A, BB);
End;
```

其中,F 为四个基本函数(输入 3 个 32bit,产生 1 个 32bit 的输出)之一,其变换情况见表 1。M[j]表示消息的第 j 个子分组;S[i]表示参数循环左移的位数;T[i]值由式子 生成,这样做是为了通过正弦函数和幂函数来进一步消除变换中的线性;LeftRot 函数的作用是将 32 位的参数 A 循环左移 Si 位。

轮	基本函数	非线性函数(b, c, d)
f_F	F(b, c, d)	$(b \wedge c) \vee (\sim b \wedge d)$
f_G	G(b, c, d)	$(b \wedge d) \vee (c \wedge \sim d)$
f_H	H(b, c, d)	$b \oplus c \oplus d$
f_I	I(b, c, d)	$c \oplus (b \wedge \sim d)$

表 1 基本函数变换表

在具体实现时,为了提高算法的运行速度和降低运算强度,不再进行 M[j]、T[i]和 S[i]的计算、查找和替换,而是直接将相应的值代入函数中。

3.2.3 分组循环处理函数

CycleDisposal 函数用于当消息长度大于一个分组长度时,对消息不断地进行分组的划分和逻辑处理。

```
CycleDisposal(M:String;Context:MD5Content)
Begin
   //Count,记录当前处理到消息 M 的哪个位置
  Count:=0;
//不断地将消息拷贝入分组存储区并处理
  While Count + 63 < Length(M) do
  Begin
    CopyMemory(@Context.Buffer[0], @pChar(M)[Count], 64);
    GroupDisposal(@Context.Buffer[0], Context.ChaVar);
    Inc(Count, 64);
 End;
End;
```

4. MD5 安全性分析

MD5 算法不基于任何的假设和密码体制,采用直接构造方法,以 32 位字为基本运算单元,非常实用且应用广泛。尽管王小云教授找出了 MD5 的缺陷,但分析其机制,我们得知:① MD5 没被破解,找到的只是 MD5 的碰撞,无法逆推得到明文,但却可以伪造身份;②MD5 碰撞的发现,仅对随机内容产生影响,而对有特殊意义或者格式的内容仍是安全的。

(上接第 117 页)

的热点。数据库上的关键词检索标志着数据库与 IR 的结合,用户查询数据库不再需要关心数据库的模式结构,只需要输入一组关键词,就能获得与查询相关的元组连接树。基于关系数据库上的关键词检索,我们提出了一种 P2P 环境下共享数据库的新框架,大大简化了不同节点之间的数据库模式映射。接着将 Top- k 查询扩展到 P2P 环境下的数据管理系统上,将文档集和数据库的查询统一起来,提供统一的查询接口。

参考文献:

1. S. Gribble, A. Halevy, et al. What Can Database Do for Peer- to- Peer? In: Proc. of the 4th WebDB. Santa Barbara, 2001. 31- 36.
2. A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In: Proc. of the 19th ICDE. Bangalore: IEEE Press, 2003. 505- 518.
3. W. S. Ng, B. C. Ooi, K. L. Tan, and A. Zhou. PeerDB: A P2P- based system for distributed data sharing. In: Proc. of the 19th ICDE. Bangalore: IEEE Press, 2003. 633- 644.
4. G. Halotia, A. Hulgeri, C. Nakhey, S. Chakrabarti, S. Sudar- shan. Keyword searching and browsing in databases using BANKS. In: Proc. of the 18th ICDE. San Jose: IEEE Press, 2002. 431- 440.
5. S. Agrawal, S. Chaudhuri, G. Das. DBXplorer: a system for keyword-based search over relational databases. In: Proc. of the 18th ICDE. San Jose: IEEE Press, 2002. 5- 16.
6. V. Hristidis, Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In: Proc. of the 28th VLDB. Hong Kong: Morgan Kaufmann Publishers, 2002. 670- 681.
7. C. Palmer, J. Steffan. Generating network topologies that obey power law. In: Proc. of GLOBECOM. San Francisco: IEEE Press, 2000. 434- 438.