一种适用于 P2P 存储系统的索引管理算法

关 中

(广州城市职业学院 广州 510230)

摘 要 PB-link Tree 通过哈希定位将 B+树分布到多个节点上,解决了动态 P2P 环境中索引的完整性和准确性问题。实验表明,即使节点频繁加入或离开系统,仍能保持数据的可靠性和一致性。而且,PB-link Tree 较之传统 DB-link Tree 在查询过程中数据传输量更小,查询时间更短。

关键词 P2P 存储系统,PB-link Tree,索引管理

An Indexing Management Algorithm Used in P2P Storage System

GUAN Zhong

(Guangzhou City Polytechnic, Guangzhou 510230, China)

Abstract PB-link tree algorithm distributes B+ tree to multi nodes through hash location and keeps integration and accuracy of the index in a dynamic P2P environment. The results show even the participating nodes join and leave the system frequently, the stored data can be still reliable and consistent. Compared with traditional DB-link Tree , it has a less data transmission and query time.

Keywords P2P storage system, PB-link tree, Indexing management

1 引言

P2P 是继 C/S 架构后新兴的网络应用模式。在传统的 C/S 架构应用系统中,客户端与服务端有明确分界,常常发生客户端能力过剩、服务端能力不足或网络拥塞的现象。P2P 系统中的使用者能同时扮演客户端及服务器多重角色,任意两个使用者之间能不通过服务器直接进行信息分享或内容交换,以构建具有自主、开放、异构、延展等特性的分布式网络应用系统。

对 P2P 数据存储及共享系统的研究成为互联网领域研究的焦点之一。P2P 存储系统可以直接构建于分布式哈希表(DHT)之上,但是无法直接使用 DHT 处理用户的查询操作。主要原因在于 DHT 只支持严格匹配的数据查询,不支持更复杂的查询语义,例如字符串属性的子串查询——根据用户给出的字符串 α 返回所有该属性值含有子串 α 的数据和数值属性的区段查询——根据用户给出的数值区段返回所有属性值在此区段里的数据。同时 DB-link Tree 应用于 P2P 存储系统时必须进行大量的数据传输,在节点之间进行中间结果的传送,在每个节点产生的中间结果和数据传输带来的网络开销都很大。因此必须对 P2P 存储系统的索引管理算法加以改进,解决执行复杂查询时带宽开销过大的问题,才能保证 P2P 存储系统的查询效率能够为用户所接受。

PB-link Tree 算法是对 B+树在 P2P 的改进,通过哈希定位的方法将 B+树的叶节点分配到各个节点上,执行复杂查询时避免中间结果的大量传输,降低带宽开销,缩短查询时间。

2 PB-link Tree 的算法框架

2.1 PB-link Tree 的构造

在不同系统中使用 B+树时索引项的构成不同,但都包含"属性值"和"数据标识"两项。其中数据标识可能是一个数值,也可能是一个字符串,还可能是数据实际存储的位置指针(比如内存地址、磁盘位置等)。总之,系统可以根据数据标识定位实际数据。PB-link Tree 对索引项中的数据标识进行哈希(使用 SHA-1 函数)[1],得到 128 位的结果 hashId,根据 hashId 对叶节点进行分布。

考虑结构化数据类 $S\{A_1, A_2, \cdots, A_n\}$,$A_i(i=1,2,\cdots,n)$ 是 S 的属性域。对于 P2P 数据库,S 相当于表(table);对于 P2P 对象存储系统,S 相当于类(class)。对每个 A_i 建立一个 B+树进行索引管理,记为 $I_i(i=1,2,\cdots,n)$的非叶节点结构与 DB-link Tree 相同。在系统中选择 t 个强节点) $P_j(j=1,2,\cdots,t)$作为 S 的索引管理节点,称为"主节点"(通常可以从系统的信息收集层了解到系统中有哪些强节点)。也就是说,每个主节点都对 S 的所有 n 个属性的索引进行管理,每个属性的索引有 t 个副本。多副本是为了防止节点失效造成索引数据丢失。用户的查询请求涉及 S 的多个属性,可以发送给任意一个主节点 P,P 在查询所涉及的各个属性的 B+树中进行相应查找,并对中间结果求交集,将最终查询结果返回给用户。这里需注意,由于查询只在单个节点 P 上进行,在中间结果求交集的过程中不需要通过网络进行数据传输。索引数据更新时需要通知所有主节点,各个主节点根据数据变化情况独立对 B+树进行插入、删除、分裂、合并等操作。不同数据类中的数据量不同。每个主节点 P 对每类数据的索引设有存储空间阈值 T,当 B+树的存储规模超过 T 时,P 在系统中寻找多个临近节点与其自身一起对 B+树进行分布式存储,这些临近节点被称为"辅助节点"。寻找临近节点的工作可以利用信息收集层的帮助。辅助节点的个数由 B+树的规模和辅助节点的能力决定:每个辅助节点对于每一类数据的索引也设有存储空间阈值,当存储规模超过阈值时,即需要将部分索引数据迁移到其它辅助节点上。当所有辅助节点的能力之和也不足以完成所有叶节点的存储时,就需要主节点从系统中寻找更多的辅助节点并对叶节点的分布进行调整。

如某主节点 P 选择辅助节点组 $W_i\{i=1,2,\cdots,r\}$进行 B+树叶节点分布,需满足 $\sum_{i=1}^{r}|W_i| \geqslant \sum_{j=1}^{n}|L_j|$。$|W_i|$表示节点

关 中 讲师,研究方向为网络系统、数据库系统、嵌入式系统、软件工程等。

W_i 为该类数据的索引数据提供的存储空间阈值，$|L_j|$ 表示索引 L_j 中叶节点需存储空间总和。将 hashId 的地址空间[0，$2^{128}-1$]分成 r 个子区间 $R_i(i=1,2,\cdots,r)$，$|R_i|\infty|W_i|$，($|R_i|$ 表示 R_i 的区间长度，∞ 表示正比关系)，这样把整个地址空间映射到了辅助节点组 $W_i\{i=1,2,\cdots,l\}$ 上，规定节点 W_i 负责区间 R_i。由于 hashId 通过 SHA-1 算法生成，可以保证所有数据的 hashId 在地址空间中均匀分布。因此 $|D_i|\infty|R_i|\infty|W_i|$，其中 D_i 是 hashId 落在区间 R_i 中的索引项的数量。对于索引 I_j 的叶节点组 L_j，按照子区间的划分将其分为 r 份，记为 $L_{ji}(i=1,2,\cdots,r)$，L_{ji} 只含有 hashId 在区间 R_i 中的索引项。将数据类 S 的 n 个属性的 B+树中的所有叶节点都进行上述划分，得到 $L_{ji}\{j=1,2,\cdots,n,i=1,2,\cdots,r\}$。

B+树中中间节点记录的指针为〈*addr*，*num*〉结构，*addr* 为子节点的地址，*num* 为子节点维护的数据项数目。在 PB-link Tree 的分布策略下，原先指向叶节点的节点所维护的指针修改为一个列表，称为“分配列表”，包含 r 项，每一项为元组〈*node*，*add*，*r num*〉，其中 *node* 为辅助节点的标识，可以定位该辅助节点，*addr* 是叶节点在该辅助节点上的地址，*num* 为叶节点分配到该辅助节点上的索引项数目。

对于每一个数据类有多个主节点，主节点的数量可以由系统决定，也可以由用户指定。系统在进行查询处理时，首先需要定位到某个主节点，这就需要在系统中记录每一个数据类的所有主节点地址。通常，可以在 P2P 存储系统中建立 DHT，对主节点地址进行存储。也就是说，对于一个数据类，将其标识进行哈希得到 hashId，将该数据类的主节点地址列表存放在所有节点中 nodeId 离 hashId 最近的节点上，并在邻近节点上进行复制。

系统中任意节点收到用户查询请求后，在 DHT 中查看主节点列表，随机选取一个主节点，向其发送查询请求。该主节点在查询请求所涉及属性的 B+树上查找，将查询结果返回给用户。

2.2 PB-link Tree 的操作

2.2.1 查询操作

在 PB-link Tree 上进行查询操作的过程如下：用户将查询请求提交给某个主节点，如果查询为简单查询，只涉及单个属性。例如，查询所有属性 X 的值在 a 与 b 之间的数据，则在属性 X 的 B+树上查找 a 和 b 所在的叶节点 A 和 B，那么 A 节点中属性值大于 a 的索引项、B 节点中属性值小于 b 的索引项以及 A 和 B 之间的叶节点上的所有索引项的合集即为查询结果。在查找过程中，非叶节点上进行的查找在主节点上完成，过程与 DB-link Tree 相同。当查找进行到叶节点时，根据分配列表将查找命令发送给所有辅助节点，即将〈*node*，*add*，*r num*〉中的 *addr* 以及查找目标值(a 或者 b)发送给 *node*。各辅助节点直接在相应位置的叶节点上定位包含目标值的索引项，将查询结果直接返回给查询发起节点。整个查询的最终结果为 $R=\bigcup_{i=1}^{r}R_i$。其中 R_i 为第 i 个辅助节点返回的查询结果。当用户提交的查询为联合查询时，对涉及到的每个属性在相应的 B+树上进行定位，方法同简单查询时一样。注意到 PB-link Tree 对每个叶节点的索引项按照 hashId 进行分配，这样每个数据的所有属性的索引项必然都存储在同一个辅助节点上。因此，可以让辅助节点在对每个属性的相关查询得到中间结果后，在本地对这些中间结果求交集，得到 R_i，返回给查询发起节点。查询发起节点合并所有 R_i，得到 $R=\bigcap_{i=1}^{r}R_i$，即为整个查询的最终结果。PB-link Tree 使用这种哈希定位的方法成功地避免了在执行联合查询时对中间结果求交集所需要的大量数据传输。PB-link Tree 中所有的中间结果求交集的操作都在节点本地进行。使用网络带宽传输的数据必然都是最终查询结果的子集，没有中间结果的传输。这种方法降低了查询过程中的带宽消耗，缩短了查询时间。

2.2.2 插入、删除操作

在 PB-link Tree 上插入索引项的过程如下：设插入的索引项的属性值为 x，hashId 为 h，首先在 B+树上查找 x。当查找进行到叶节点时，由 h 可以确定负责该索引项的辅助节点。将叶节点的位置 *addr* 和待插入的索引项发送给该辅助节点，辅助节点在相应叶节点上完成插入操作，同时在主节点上修改相应分配列表中的 *num* 域。删除索引项的操作与插入操作基本相同：1)在 B+树上查找目标索引项的属性值；2)在 hashId 对应的辅助节点上完成索引项的删除操作；3)相应修改主节点分配列表中的 *num* 域。

2.2.3 分裂、合并操作

在 Active Update 策略下，由于 PB-link 树的叶节点按照数据 ID 分割不能完全均匀，不能根据叶节点空满的程度决定是否要进行节点分裂或合并，故需采用 B 树判断的标准，对 PB-link 树列表元组中的 *num* 域求和，确定是否要进行分裂或合并。当发现某个叶节点要进行分裂或合并时，索引节点向负责该叶节点的辅助索引节点发送分裂或合并命令。收到命令后，对相应的节点进行分裂或合并。Active Update 策略可以保证 PB-link 树不退化，严格保证 PB-link 树的 B 树语义，但这种策略引起的网络开销和所涉及的节点较多。

在 Lazy Update 策略下，辅助节点根据自身存储能力和存储状况确定是否叶节点分裂。叶节点越大，对存储需求越大。当某辅助节点对某数据类的索引数据的存储超过存储空间阈值时，即主节点要求分裂时，主节点在系统中再寻找一个临近节点作为辅助节点，之后索引数据膨胀时将通过叶节点分裂逐渐将索引项分布到新的辅助节点。采用 Lazy Update 策略会减少叶节点分裂、合并的频率，降低节点间维持一致性所造成的网络开销。

3 实验结果及数据分析

实验在一个基于事件驱动的模拟平台上进行，用于模拟节点之间的通信及定时操作。在模拟中，PBlink 树按照节点带宽的大小区分强弱节点，而 DB-link 树算法则随机选择节点。模拟平台在 4 台高性能集群服务器上进行。每台服务器有 PIII-Xeon700MHz 处理器，2GB 内存，服务器之间采用高速以太网连接。

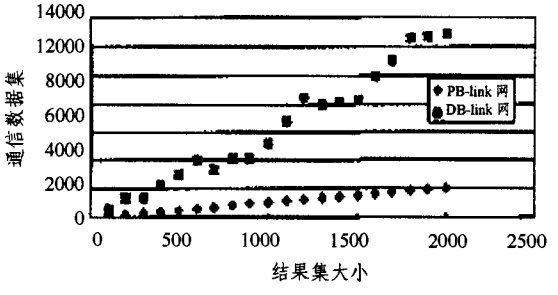

图 1 PB-link Tree 与 DB-link Tree 进行复杂查询时的通信开销

PB-link Tree 与 DB-link Tree 在进行复杂查询时的通信开销见图 1，及查询时间见图 2。通信开销以查询过程中所需要传输的数据量计算。查询的结果为数据标识列表，其中每

(下转第 144 页)

列对明文的依赖性不能过大，以免造成信息的泄漏。

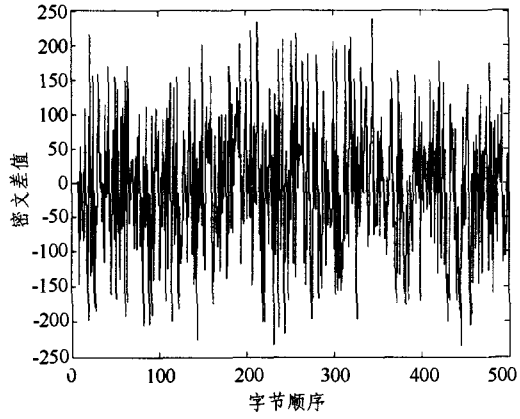

图 4　用改进后的算法对两个仅首字节相差 1 比特的明文进行加密后所得密文的差值

结束语　对一种基于迭代混沌映射的加密系统进行了详细分析，指出了其中存在的安全漏洞，并用选择明文的方式对其进行了攻击。然后，我们提出了相应的改进方法。最后对设计基于迭代混沌映射的分组加密算法时应当注意的问题进行了总结。

参 考 文 献

[1] Xiang Tao, Liao Xiaofeng, Guo Ping, et al. A novel block cryptosystem based on iterating a chaotic map. Physics Letters A, 2006, 349: 109-120

[2] Baptista M S. Cryptography with chaos. Physics Letters A, 1998, 240: 50-57

[3] Wang Yong, Liao Xiaofeng, Xiang Tao, et al. Cryptanalysis and improvement on a block cryptosystem based on iteration a chaotic map. Physics Letters A, 2007, 363: 277-281

[4] Alvarez G, Montoya F, Romera M, et al. Cryptanalysis of dynamic look-up table based chaotic cryptosystems. Physics Letters A, 2004, 326: 211-218

[5] Alvarez G, Montoya F, Romera M, et al. Keystream cryptanalysis of a chaotic cryptographic method. Computer Physics Communications, 2004, 156: 205-207

[6] Wong K-W. A fast chaotic cryptographic scheme with dynamic lookup table. Physics Letters A, 2002, 298: 238-242

[7] Wong K-W. A combined chaotic cryptographic and hashing scheme. Physics Letters A, 2003, 307: 292-298

[8] Wei Jun, Liao Xiaofeng, Wong K-W. Analysis and improvement for the performance of Baptista's cryptographic scheme. Physics Letters A, 2006, 354 (1): 101-109

[9] Pareek N K, Patidar V, Sud K K. Cryptography using multiple one-dimensional chaotic maps. Commun Nonlinear Science and Numerical Simulation, 2005, 10: 715-723

(上接第 140 页)

个数据标识的大小为 4 个字节。实验中数据类有 4 个数值属性，共有 5000 个数据。查询均为对数据类的 4 个属性的联合查询。实验测试了在各种大小的查询结果集下两种算法的通信开销(数据传输量)和查询时间(指发出查询请求到收到全部查询结果之间的时间长度)。

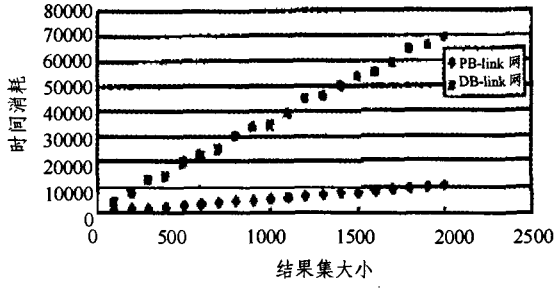

图 2　PB-link Tree 与 DB-link Tree 进行复杂查询时的查询时间

从图 1 和 2 可以算出 PB-link 树的通信开销和查询时间平均是 DB-link 树的 0.21 和 0.143。这说明 PB-link 树优化了 P2P 环境分布式结构化索引的效率，降低了查询代价，可见在广域网环境下 PB-link 的带宽开销和查询时间均远小于 DB-link Tree。

4　Lazy Update 与 Active Update 的性能对比

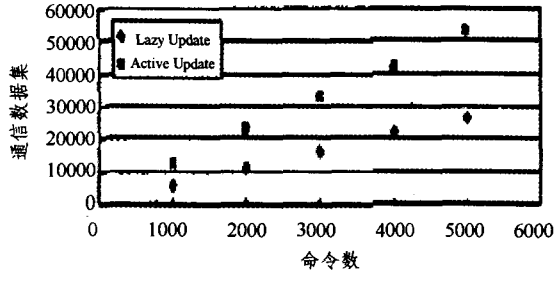

图 3　Lazy Update 和 Active Update 执行插入、删除命令时的平均查询时间对比

在 PB-link Tree 的节点分裂、合并策略上分别使用 Active Update 策略和 LazyUpdate 策略。实验首先进行一定次数的随机插入、删除操作，然后进行数据查询。同样测试了两种策略下的数据传输量和查询时间(见图 3，图 4)。

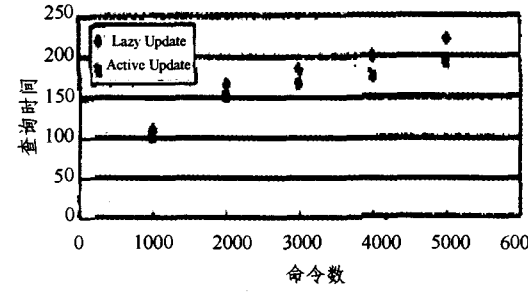

图 4　Lazy Update 和 Active Update 执行插入、删除命令时的数据传输量对比

从图 3 和图 4 中可以看到，Lazy Update 执行插入、删除命令时所需的通信开销远小于 Active Update，而二者执行一定的插入、删除命令后的查询时间相差很少。因此，Lazy Update 更加适合于 P2P 环境下的索引维护。

结束语　PB-link Tree 算法解决了已有算法在执行联合查询时带宽开销过大的问题。通过使用分布式 B+树实现数值属性区段查询、字符串属性的子串查询和多属性之的联合查询。通过哈希定位策略将 B+树叶节点分布到 P2P 系统中的多个节点上，保证一份数据的多属性索引项存储于同一个节点，避免联合查询中中间节点求交集时的大量数据传输，减少了查询开销，提高了查询效率。

参 考 文 献

[1] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large -scale peer-to-peer systems. Lecture Notes in Computer Science, 2001, 2218: 329-350

[2] Shi S. Making Peer-to-Peer Keyword Searching Feasible Using Multi-levelartitioning // 3rd International Workshop onPeer-to-Peer Systems. February 2004: 400-408

[3] Zegura E W. How to modal an internetwork // Proc. of the INFOCOM'96. 1996: 594-602

[4] Dingledine R. The free haven project: Distributed anonymous storage service // Proc. of the Workshopon Design Issues in Anonymity and Unobservability. 2000(3): 67-95