外包数据库中的哈希连接一致性算法

马 莎 杨 波 李康顺

(华南农业大学信息学院 广州 510642)

摘 要 外包数据库中的连接查询比范围查询更困难,因为客户端需要验证连接结果的一致性,而传统的对单个表的签名不能有效地支持对连接查询结果的直接验证。提出了使用哈希连接保证数据一致性的2种算法,它们分别通过在服务器端和客户端计算哈希函数来实现连接查询。给出了这2种算法的详细描述,证明了它们满足一致性要求,而且分析了算法在通信量、服务器端和客户端执行的代价。最后在实验中通过设置不同的参数对它们在服务器端和客户端的运行时间进行了比较,总结了它们在实际应用中的优缺点。

关键词 数据库安全,外包数据库,哈希连接,数据一致性

中图法分类号 TP392 **文献标识码** A

Algorithm for Authenticated Hash Join Processing in Outsourced Database

MA Sha YANG Bo LI Kang-shun

(Dept. of Information, South China Agricultural University, Guangzhou 510642, China)

Abstract In outsourced database, authenticated join processing is more difficult than authenticated rang queries because the previous approach for signature on a single relation can not be used to verify join results directly. This paper provided two kinds of authenticated hash join processing algorithms, which respectively compute hash functions by database service provider and by client. These two methods were described in detail and proved to guarantee the authenticity of join results. The analysis of the performance was presented with respect to cost factors, such as communication cost, server-side cost and client-side cost. Finally, we experimentally compared the two methods on the running time of server-side and client-side in different parameters and summarized their advantages and disadvantages in the application.

Keywords Database security, Outsourced database, Hash join, Data authenticity

随着网络技术和通信技术的日益成熟,数据库外包提供了一种新的数据管理模式[1,2]。它为企业节省了昂贵的管理和聘请专业管理人员的费用,在满足企业需求的同时,提供了与本地数据库一样的数据管理服务。在外包数据库(outsourced database, ODB)中,数据所有者(Data Owner, DO)将数据库外包给一个数据库服务提供方(Database Service Provider, DSP)进行数据管理,DSP能够接受客户(Client)提交的查询请求,返回查询结果。与传统的数据管理相比,ODB中的DSP已不在企业所能管辖的范围内,因此它提供的数据服务不再自可信。具有安全和隐私保护能力的数据库管理技术已成为研究ODB的一个难点。目前,一些学者关于数据的机密性[3]、完整性[4-7]、完备性[8,9]、查询的有效性[10]和隐私保护[11]等问题提出了一些解决方法和实现技术。然而对这一领域的研究还刚刚起步,还有大量十分有意义的课题等待解决[12,13]。自Y. Yang在文献[14]中第一次详细地讨论了外包数据库中采用排序合并的连接算法后,本文提出了外包数据库查询采用哈希连接的2种算法,并进行了一致性证明和代价分析,然后通过实验比较分析了这2种算法在查询处理时间上的差别,最后总结了它们在实际应用中的优缺点。

1 相关工作

保证ODB数据一致性包含两方面的含义:(1)合理性;(2)完备性。合理性指DO需要提供额外的机制来保证DSP对Client提出查询的返回结果是正确的,即返回的查询结果是真实的原始数据,没有经过任何篡改。完备性指DSP在执行查询过程中不能任意向数据库中增加元组或者删除已有的元组,以保证查询结果的内容与元组个数和未经删减过的数据库中的原始数据是完全相同的。本文提出的2种算法都将参与连接的表中的所有数据返回到客户端,目的是为了减少客户端执行连接的代价。

1.1 数字签名

数字签名就是附加在数据单元上的一些数据,或是对数据单元所做的密码变换。这种数据或变换允许数据单元的接收者用以确认数据单元的完整性并保护数据,防止被其他人进行伪造。常见的数字签名方法有RSA, DSA等[15]。在ODB中,最常用的是DO对每一个元组进行签名,Client用私

到稿日期:2011-03-22 返修日期:2011-05-17 本文受国家自然科学基金(60773175,60973134和70971043),现代通信国家重点实验室基金(9140C1108020906),广东省自然科学基金(10351806001000000,10151064201000028和9151064201000058)资助。

马 莎(1982-),博士生,讲师,主要研究领域为数据库安全,E-mail:martin_deng@163.com;杨 波(1963-),男,博士,教授,主要研究领域为信息安全;李康顺(1962-),男,博士,教授,主要研究领域为演化计算。

钥进行验证，其缺点是如果数据库中存在大量的元组，除了DSP端的存储代价增加外，对一个元组的签名和验证也将产生大量的代价，因此通常对数据库中的每个表进行一次签名。

1.2 连接查询的一致性方法

保证连接查询的一致性比保证单表范围查询的一致性困难得多，因为每一个表都可以通过DO的签名来验证，而连接的结果由于无法预知因而不能直接验证。关于这一问题的研究成果不多，一直以来是一个难点。文献[16]提出将所有可能的连接结果存放在物化视图中。每一个物化视图对DSP来说就是一个普通表。Client提交的连接查询转化为在普通表上的选择。这种方法在实际中是不可行的，因为构建和更新物化视图需要大量的代价，此外预先定义好所有可能的连接也是不现实的。文献[17]简单提及了对外关系的每一个元组，可利用内关系在连接属性上的MB树执行连接(这种方法在文献[14]中称为Authenticated Index Nested Loop，AINL)。文献[14]第一次深入讨论了基于MB树的排序合并连接算法，并提出了3种不同的实现类型：(1) Authenticated Indexed Sort Merge Join(AISM)。AISM首先对外关系进行排序，在内关系的连接属性上使用MB树遵循"no-go-back"规律进行搜索。数据的一致性是通过在外关系表级的验证和内关系上的MB树的验证来实现的。与AINL相比，由于只需要遍历MB一次，因此显著地减少了访问MB树的代价。(2) Authenticated Index Merge Join(AIM)。AIM与AISM的不同之处在于外关系在连接属性上也存在一个MB树。连接操作转化为在这两个MB树上的某种遍历。数据的一致性是通过外关系和内关系上的两个MB树的验证来实现的。由于这两个MB树都遵循"no-go-back"，因此这种连接算法的效率最高。(3) Authenticated Sort Merge Join(ASM)。在ASM中，外关系和内关系在连接属性上都没有MB树。DSP首先执行排序合并连接，分析出能够参与连接的元组和执行连接扫描元组的顺序，目的是减少Client执行连接的代价。数据的一致性是通过外关系和内关系表级的验证来实现的。这种方法的效率在这3种连接算法中是最低的。

1.3 哈希连接的相关技术

表在某个属性上的向量是指所有元组在该属性上投影值的集合。某些表在连接属性上可能存在大量重复的值，通过计算表在该属性上投影值的集合可以通知与之相连接的表哪些是不可能有相匹配的元组，以加快连接的速度。这种方法广泛使用在哈希连接中，不足之处是会占据内存一部分空间。

哈希另外一种经常使用的技术是采用2级hash。对每一个元组计算两个hash值。第一个值用来将元组划分到它确定的分区中，第二个值用来实现相同分区内的元组产生hash table来进行连接，目的是提高哈希连接的效率。

1.4 本文工作

本文提出的外包数据库中哈希连接可以分为如下2种情况。

(1)DSP计算hash函数。DSP计算hash函数的动力是为了充分发挥外包数据库的优势，利用DSP端的资源尽早执行哈希连接，减轻Client执行的代价。与传统数据库不同的是，DSP不在信任的范围之内，Client需要额外验证DSP计算hash函数的结果。为了能够帮助Client尽快执行连接，第2节提出在传输所有元组的序列中增加一些特殊的符号，以给Client执行连接提供一些提示信息。

(2)Client计算hash函数。由于Client处于信任范围之内，因此哈希连接的计算也是可信的。Client计算哈希函数的主要考虑是尽可能减少CPU执行连接的代价，以提高查询的效率。

为了能更好地举例阐述算法的思想，我们使用的数据库和查询如图1所示，主要的符号及含义见表1。

Customer

cid	cname	tel	addr
c1	Tom	85283450	New York
c2	Mary	23517854	River
c3	John	87609432	Main
c4	Jerry	23596130	River
c5	Susan	43287650	London
c6	Smith	21097692	Tokyo

Deposit

did	amount	cid
d1	2000	c2
d2	300	c6
d3	2500	c3
d4	10000	c2
d5	5780	c4
d6	2600	c1
d7	120	c3
d8	4600	c5
d9	1800	c1
d10	6300	c6

$Q=Customer \infty Deposit$

图1 一个数据库实例和查询

表1 主要的符号与含义

符号	含义
$\|R\|,\|S\|$	R和S元组的个数
$\|Tup_R\|,\|Tup_S\|$,	R和S元组的大小
Sig_R, Sig_S	R和S的签名
$\|Sig\|$	签名的大小
$\|B\|$	一个磁盘页面的大小
M	执行哈希连接内存的大小
$C_{I/O}, C_{hash}, C_{verify}$,	I/O的代价，hash的代价和验证的代价

2 HADSP

这一节介绍HADSP(Hash at DSP)算法，即在DSP端计算hash函数。首先给出它的算法描述，然后证明算法满足一致性要求，接着通过一个实例进行解释，最后分析执行的代价。

2.1 算法描述

第1步 DSP在委托数据库上执行hash连接。

第2步 DSP产生验证对象VO，以减少Client进行连接的代价。这里的VO包括R和S所有的元组、R和S在表级上的签名Sig_R和Sig_S以及一个R和S的哈希连接启发式链表Φ。为了让Client能够验证链表Φ，VO还包括hash1和hash2的定义，以及分别出现在链表Φ中hash值的顺序链表$Order_1$和$Order_2$。连接启发式链表Φ是为了减少Client不必要的连接，以提高连接效率。假设hash1产生了n个不同的值$\{w_1^1, w_2^1, \cdots, w_n^1\}$，hash2产生了$m$个不同的值$\{v_1^2, v_2^2, \cdots, v_m^2\}$。不失一般性，用#代表hash1产生的桶的区分，用@代表hash2产生的桶的区分，$r(\cdot)$和$s(\cdot)$分别代表R和S的某些元组，链表Φ可写为：

$$\Phi(w_1^1,\cdots,w_n^1;v_1^1,\cdots,v_n^1)=$$

$$\underbrace{\#\overbrace{@r(\cdot),s(\cdot)}^{v_1^1}@\cdots\overbrace{@r(\cdot),s(\cdot)@}^{v_n^1}\#}_{w_1^1}$$

$$\cdots$$

$$\underbrace{\#\overbrace{@r(\cdot),s(\cdot)}^{v_1^1}@\cdots\overbrace{@r(\cdot),s(\cdot)@}^{v_n^1}\#}_{w_n^1}$$

hash 值的顺序链表包含所有计算 hash 得到的值，它们按照出现在链表 Φ 中的顺序进行排序，即 $Order_1=\{w_1^1, w_2^1, \cdots, w_n^1\}$；$Order_2=\{v_1^2, v_2^2, \cdots, v_m^2\}$。

第 3 步　Client 接受 R 和 S 以及验证对象 VO，并对 R 和 S 进行验证。如果验证通过，继续执行下一步；否则 R 或 S 中的数据已经被篡改，查询过程终止。

第 4 步　读取连接启发式链表 Φ，连接其中按照某种顺序出现的元组。获取这些可能进行连接的元组对的方法是通过扫描链表 Φ 中一些特殊的符号，以获得 R 和 S 中相同 hash1 和 hash2 值的元组。其中，相同 hash1 的值体现在链表 Φ 中相邻＃之间的元组，相同 hash2 的值体现在相邻@之间的元组。客户端可直接对两个相邻＃之间的两个相邻@之间的元组执行连接。

第 5 步　当所有的元组按照链表 Φ 中出现的顺序读取完一遍，可对 $Order_1$、$Order_2$ 和链表 Φ 进行验证。

2.2　举例说明

DSP 在 Customer 和 Deposit 上进行自然连接。假设用〈key〉代表码为 key 所在的元组，对所有元组在连接属性上计算 hash1 和 hash2 所得的结果如图 2 所示。

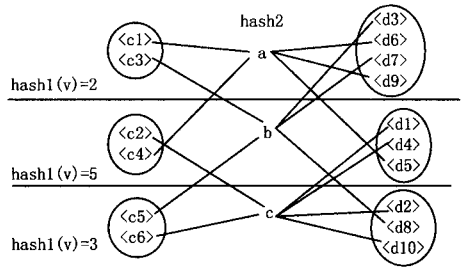


图 2　计算 hash1 和 hash2 的结果

接着，DSP 产生验证对象 VO。VO 包括对 Customer 和 Deposit 两个表的签名 $Sig_{customer}$ 和 $Sig_{deposit}$，$Order_1=\{2,5,3\}$，$Order_2=\{a,b,c\}$。链表 Φ 的生成可以通过以下两步来完成：

(1)对所有的元组用＃进行划分，每两个＃之间的元组 hash1 的值是相同的。从图 2 中可以得到：

Φ′＝{＃c1,c3,d3,d6,d7,d9＃c2,c4,d1,d4,d5＃c5,c6,d2,d8,d10＃}

(2)对每两个＃之间的元组用@进行划分，每两个@之间的元组 hash2 的值是相同的。令

Φ_1'＝{＃c1,c3,d3,d6,d7,d9＃}

Φ_2'＝{＃c2,c4,d1,d4,d5＃}

Φ_3'＝{＃c5,c6,d2,d8,d10＃}

计算 hash2，使用分隔符@后：

Φ_1''＝{＃@c1,d6,d9@c3,d3,d7@@＃}

Φ_2''＝{＃@c4,d5@@c2,d1,d4@＃}

Φ_3''＝{＃@@c5,d8@c6,d2,d10@＃}

最后，Φ＝{＃@c1,d6,d9@c3,d3,d7@@＃@c4,d5@@c2,d1,d4@＃@@c5,d8@c6,d2,d10@＃}。

Client 接受 Customer 表和 Deposit 表以及验证对象 VO＝{R,S,$Sig_{Customer}$,$Sig_{Deposit}$,$Order_1$,$Order_2$,Φ}，对 Customer 表和 Deposit 表进行签名的验证。如果验证未通过，说明 R 或 S 中的数据已经被篡改，查询过程终止。否则，Client 读取连接启发性链表 Φ，获得进行连接的元组对：{〈c1,d6〉〈c1,d9〉〈c3,d3〉〈c3,d7〉〈c2,d1〉〈c2,d4〉〈c4,d5〉〈c5,d8〉〈c6,d2〉〈c6,d10〉}。如果所有的元组按照链表 Φ 中的顺序读取完一遍，可对 $Order_1$，$Order_2$ 和链表 Φ 进行验证。

2.3　一致性证明

(1)正确性证明

如果 rs 是不正确的连接结果，则(1)r 与 s 不匹配，或者(2)r 或 s 都已被篡改。第(1)种情况不会发生，因为客户端在本地执行连接，不会产生不匹配的连接结果。第(2)种情况也不会发生，因为如果 r 或 s 是无效的或被篡改过，对 R 或 S 的验证将不会通过。

(2)完整性证明

如果 rs 是有效的，但它是被 Client 遗漏的连接结果，则(1)Client 没有收到 r 或没有收到 s，或者(2)Client 并没有认为 r 和 s 是匹配的元组。第(1)种情况不会发生，因为如果 Client 没有收到 r 或 s，对 R 或 S 的验证不会通过。第(2)种情况不会发生，如果 Client 遗漏了正确的 rs，则说明 r 或 s 的 hash1 或者 hash2 函数计算的结果是错误的。由于在算法的最后需要对链表 Φ 进行验证，即不正确的 r 或 s 会使得对链表 Φ 的验证失败。

2.4　代价分析

(1)通讯线路上的代价

HADSP 通讯线路上 VO 的大小主要包括 R 和 S 所有的数据，此外包括签名、hash 值的顺序链表和 Φ 链表。

$$|R|*|Tup_R|+|S|*|Tup_S|+2*|Sig|+|Ord_1|+|Ord_2|+|\Phi|$$

(2)DSP 计算的代价

DSP 计算的代价主要包括读取数据和生成 Φ 的代价。其中生成 Φ 主要是对所有的元组计算 2 个 hash 值并进行排序的过程。

$$\left(\frac{|R|*|Tup_R|}{|B|}+\frac{|S|*|Tup_S|}{|B|}\right)*C_{I/O}+2*(|R|+|S|)*C_{hash}+O((|R|+|S|)^2)$$

(3)Client 计算的代价

Client 计算的代价主要包括验证签名和执行哈希连接的代价。假设 2 级 hash 使得相邻分隔符“@”之间的连接可以在内存中执行，Client 计算的代价为：

$$\left(\frac{|R|*|Tup_R|}{|B|}+\frac{|S|*|Tup_S|}{|B|}\right)*C_{I/O}+2*C_{verify}$$

3　HAC

这一节介绍 HAC(Hash At Client)算法，即在 Client 端计算 hash 函数。如果 R 可以放在内存中，为提高查询响应效率，可用接受到的 S 中的每一分组探测 R 所有的元组进行哈希连接。由于这一哈希连接可在内存中执行，因而能获得较高的查询效率。如果 R 不能完全放在 Client 的内存中，R 和 S 需要分别进行分组，以便 R 和 S 每一对应分组的连接能够执行在内存中。首先给出它的算法描述，接着通过一个实例进行解释，然后证明算法满足一致性要求，最后对执行的代价进行分析。

3.1　算法描述

HAC 算法示意图如图 3 所示。

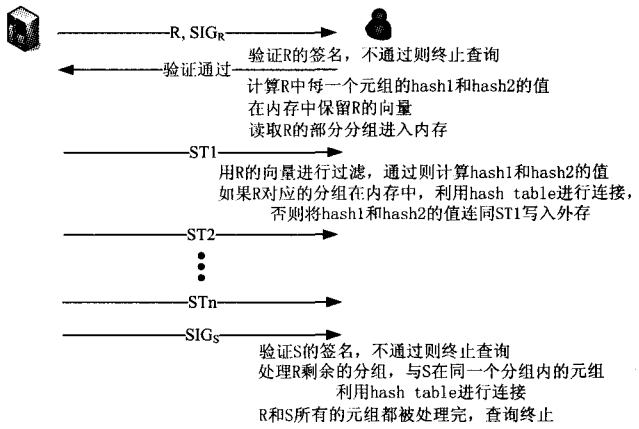


图 3　HAC 算法示意图

第 1 步　DSP 将小表 R 和对 R 的签名全部传送到 Client 端。

第 2 步　Client 用公钥对 R 进行验证。如果验证通过，继续执行下一步；否则 R 中的数据已经被篡改，查询过程终止。

第 3 步　Client 读取 R，在连接键值上计算函数 hash1，将其进行分组；同时 Client 还需计算 hash2，用于 hash table，哈希连接。

第 4 步　在内存中保留 R 的向量。

第 5 步　如果内存空间不够，将其写入外存。

第 6 步　读取 R 的剩余部分，重复第 3 步，直到 R 的所有元组全部读完。

第 7 步　读取 R 中的部分分组进入内存，对每一个分组建立 hash table。

第 8 步　DSP 将 S 中的一个元组 ST1 发送到 Client 端，待 Client 处理完成后，准备发送下一个元组。

第 9 步　Client 对 DSP 发送过来的元组，利用 R 的向量进行过滤。如果未通过，说明 ST1 没有相匹配的元组，只需将其保存在外存中。

第 10 步　对通过过滤后的数据计算 hash1 函数的值，并计算 hash2 函数的值。

第 11 步　计算 hash1 得到的值所在的分组正好在内存中，则利用 hash2 函数的值与内存中在同一分组内的 R 中的元组进行连接，并输出查询结果。如果 hash1 函数的值所在分组不在内存中，则将 hash1 和 hash2 函数的值与 STi 放在一起写入磁盘。这里需要说明的是，尽管可能在第 12 步未通过验证，查询过程中止，但是在读取 S 的同时完成一部分过滤和计算可节约 CPU 的时间和减少盲目读取 S 中所有的数据引来的不必要的 I/O 代价。

第 12 步　继续读取 S 的剩余部分，重复第 8 步，直到 S 的所有元组全部读完。

第 13 步　Client 对 S 进行验证。如果验证通过，继续执行下一步；否则 S 中的数据已经被篡改，查询过程终止。

第 14 步　读取 R 中剩余的分组，并读取 S 中通过过滤且与 R 在同一个分组内的元组，即首先获取 hash1 函数值相同的 R 和 S 的分组，然后利用 hash2 函数值进行组内的 hash 连接。直到 R 和 S 所有的分组都被处理完。

3.2　举例说明

沿用前面的例子，用 CTi(i=1,…,6) 和 DTi(i=1,…,10)代表 Customer 中的 6 个元组和 Deposit 中的 10 个元组。

1)DSP 将 Customer 中的 6 个元组 CTi(i=1…6)依次传给 Client，此外还有对 R 的签名 Sig$_R$。

2)Client 验证 R。如果通过验证，继续执行下一步；否则 R 中的数据已被篡改，查询终止。

3)在内存中保留 Customer 在连接属性 cid 上的向量 V_{cid}={c1,c2,c3,c4,c5,c6}。

4)计算每一个元组的 hash1 和 hash2 的值，如图 2 所示。假设内存中只能容纳 Customer 的两个分组 hash1(v)=2 和 hash1(v)=5，如图 4 所示。剩下一个分组的数据，需要将它们放在外存中，直到所有的数据处理完一次。

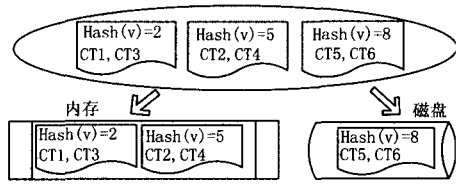


图 4　Customer 表的存储示意图

5)DSP 发送 Deposit 表的第一个元组 DT1(d1,2000,c2)给 Client，待 Client 处理完后准备发送下一个元组。

6)Client 接受 DT1，计算它的连接属性 hash1 的值。因为 DT1 在连接属性上的值是 c2，c2 在 V 中，并且它 hash1 的值是 5，说明可在内存中执行连接；如果不在 V 中，说明 Customer 中不存在与之相连接的元组，保留其值以供 Depositor 进行验证。同时计算 c2 的 hash2 的值，从图 2 可知 hash(c2)=c。因为在 R 中 hash1(v)=5 且 hash2(v)=c 的元组只有一个 CT2，因此将它们进行连接，得到一个连接结果〈c2,d1〉。

7)继续以类似于 6)的方式处理 DT2-DT10。如果通过过滤且与 DTi 相匹配的分组不在内存中，则将 DTi，hash1 和 hash2 的值保存在磁盘上。

8)Client 验证 Deposit。如果通过验证，则继续执行下一步。

9)继续读取磁盘上 Customer 中 hash1(v)=3 的分组，并依次读取相应 Depositor 中通过过滤且 hash1 值也为 3 的分组(DT2,DT8,DT10)，用 hash2 的值进行连接。输出最后的连接结果{〈c1,d6〉〈c1,d9〉〈c3,d3〉〈c3,d7〉〈c2,d1〉〈c2,d4〉〈c4,d5〉〈c5,d8〉〈c6,d2〉〈c6,d10〉}。

3.3　一致性证明

(1)正确性证明

如果 rs 是不正确的连接结果，则 1) r 与 s 不匹配，或者 2) r 或 s 都已被篡改。第 1)种情况不会发生，因为客户端在本地执行连接，不会产生不匹配的连接结果。第 2)种情况也不会发生，因为如果 r 或 s 是无效的或被篡改过，对 R 或 S 的验证将不会通过。

(2)完整性证明

如果 rs 是有效的，但它是被 Client 遗漏的连接结果，则(1)Client 没有收到 r 或没有收到 s，或者(2)Client 并没有认为 r 和 s 是匹配的元组。第(1)种情况不会发生，因为如果 Client 没有收到 r 或 s，对 R 或 S 的验证就不会通过。第(2)种情况不会发生，Client 已经获得所有过滤后的 R 和 S 的元组，通过本地计算 hash1 和 hash2 的值执行 hash join 找出所有匹配的 rs，不会遗漏可能的 rs。

3.4　代价分析

(1)通讯线路上的代价

HAC 通讯线路上的代价主要包括 R 和 S 所有的数据和签名：

$$|R|*|Tup_R|+|S|*|Tup_S|+2*|Sig|$$

(2)DSP 计算的代价

DSP 计算的代价主要包括读取数据的代价：

$$2*(|R|+|S|)*C_{hash}+2*C_{verify}+\left(\frac{(2*|R|*|Tup_R|-M)+(2*|S|*|Tup_S|-2*|S|*|Tup_S|*M\div(|R|*|Tup_R|))}{B}\right)*C_{I/O}$$

$$\left(\frac{|R|*|Tup_R|}{|B|}+\frac{|S|*|Tup_S|}{|B|}\right)*C_{I/O}$$

(3)Client 计算的代价

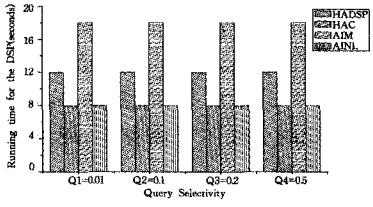
Client 计算的代价主要包括执行 hash 连接的代价和验证签名的代价：

4 实验分析

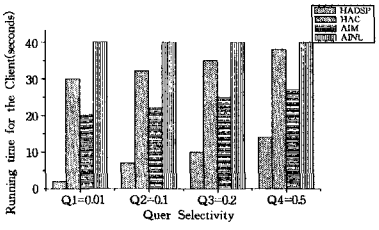
利用 openSSL 提供的密码函数库[18]实现了两种哈希连接算法，分析对比了验证结构的大小和查询执行时间。

实验中设计了两个关系 R 和 S。R 表包含 3 个属性 a1，a2 和 a3，S 每一个表包含 3 个属性 a1，a4 和 a5。$R.a1$，$R.a2$，$S.a1$，$S.a4$ 的类型为 NUMBER(32)，它们的值在$[1,10^7]$之间均匀选择。$R.a3$ 和 $S.a5$ 的类型为 VCHAR，通过定义不同的字符串长度可控制 R 和 S 规模的比例。$S.a1$ 是主键，$R.a1$ 是引用它的外键。模拟过程中的参数包括查询选择因子和元组的大小。默认 R 包含 2500 个元组，S 包含 106 个元组，P 为 20%，Q 为 1%。在每一个组实验中，与 AINL[4,9] 和 AIM[3] 相比，我们变化其中的某些变量，而保持其它变量的值不变。实验使用了如下的查询语句(通过修改 w_1 的值可以改变查询选择因子)。

$$Q=\delta_{R.a2<w_1}(R\infty S)$$



(a) DSP 的运行时间

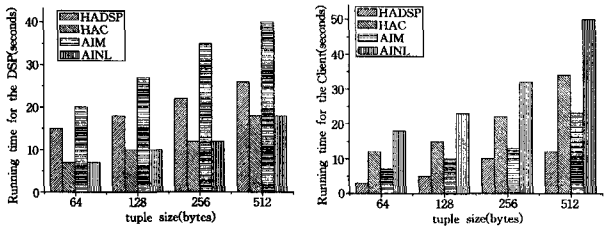


(b) Client 的运行时间

图 5 DSP 和 Client 随查询因子变化的运行时间的比较

实验将本文提出的 2 种算法与 AINL 和 AIM 在 DSP 和 Client 查询中的运行时间上进行了比较。第一组数据假设数据库大小不变，通过改变查询因子比较运行的时间。图 5(a)描述了在不同选择因子的查询下 4 种算法 DSP 端运行时间的比较。从中可以看出，由于 HADSP，HAC 和 AINL 都需要操作所有的数据，因而各自的运行时间变化均不大；同时，HADSP 中 DSP 还要预处理绝大部分全表的连接，AIM 要在连接属性的 MH 树上做多轮搜索，所以这两种算法 DSP 运行的时间要长。图 5(b)描述了在不同选择因子下 4 种算法 Client 运行时间的比较。容易看出，HAC 和 NAIL 运行时间是最长的，这主要是因为 HAC 中 Client 要实现整个哈希连接；NAIL 中 Client 要实现整个嵌套循环连接。与 HAC 相比，HADSP 通过做一部分预连接能达到减轻 Client 的执行代价的目的。需要指出的是，随着选择因子的增大，对 MH 树的验证计算 hash 函数的代价增加，AIM 中 Client 的运行时间会增加。

第二组数据假设选择因子不变，为 0.1，通过改变 R 的元组大小比较查询运行的时间。改变 R 中元组的大小实际上就是改变 R 和 S 的大小比例。从图 6 可以看出，这 4 种算法随着 R 与 S 比例的增加，DSP 和 Client 的运行时间都会有不同程度的增加。不过因为 Client 比 DSP 查询处理的能力要弱得多，所以这种变化对 Client 的影响比对 DSP 的影响要大一些。



(a) DSP 的运行时间　　(b) Client 的运行时间

图 6 DSP 和 Client 随元组大小变化的运行时间的比较

从以上两组数据可以看出，HADSP 算法能充分利用 DSP 端的资源，更好地发挥外包数据库的优势，使 Client 在运行时间上优势非常明显；尤其对于等值连接，HADSP 比 AIM 获得了更高的效率。此外，在使用这 2 种算法时还需要考虑以下几个因素。

(1)数据更新的频率。如果数据更新频率过快，AIM 需要数据的拥有者重新上传 MH 树给 DSP，这必然会引来额外的代价。而 HADSP 和 HAC 不存在这个问题。因为 HADSP 和 HAC 中使用的 hash 函数是动态生成的，可以灵活变化，因而可以更好地适应数据动态的更新。在大量数据更新频率很高的情况下，不宜采用 AIM 算法。

(2)安全性。本文关注的是保证数据库中的数据没有被恶意篡改，这可以保证数据在一定程度上的安全性，但不是绝对的安全，比如数据仍然是明文，数据的分布、连接的信息等都可能会泄漏给敌手。已经证明，要保证数据库查询是绝对安全的，需要将所有表的元组返回给用户，而不能包含其它任何数据[19]。从这个意义上说，HDC 能够更好地扩展，以适应这种安全性要求下的场景。

结束语 提出了外包数据库中 2 种保证哈希连接一致性的算法。这 2 种算法有各自实用的场景：HADSP 适用于数据更新较频繁而 Client 计算资源不足的领域；HAC 适用于安全性要求较高的领域。本文详细描述了每一种算法，并通过一个例子加以说明，同时给出了一致性证明与代价分析。最后通过实验数据比较分析了它们在不同的参数下，DSP 和 Client 运行时间的比较。本文对研究外包数据库中的哈希连

(下转第 221 页)

[5] 朱博,戴先中,李新德. 基于"原型"的机器人开放式室内场所感知算法研究[J]. 模式识别与人工智能

[6] Zender H,Mozos O M,Jensfelt P,et al. Conceptual Spatial Representations for Indoor Mobile Robots[J]. Robotics and Autonomous Systems,2008,56:493-502

[7] 杨震. 物联网及其技术发展[J]. 南京邮电大学学报:自然科学版,2010,30(4):9-14

[8] International Telecommunication Union UIT. ITU Internaet Reports 2005:The Internet of Things[R]. 2005

[9] 沈苏彬,毛燕琴,范曲立,等. 物联网概念模型与体系结构[J]. 南京邮电大学学报:自然科学版,2010,30(4):1-8

[10] Zhao Yi-yang,Liu Yun-hao,Lionel M N. VIRE:Active RFID-based Localization Using Virtual Reference Elimination[C]// Proc. of the International Conference on Parallel Processing (ICPP). Xi'an,China,2007

[11] Mao Guo-qiang,Fidan B,Anderson B D O. Wireless sensor network localization techniques[J]. Computer Networks,2007,51:2529-2553

[12] Li Xin-de, Dai Xian-zhong, Jean D, et al. Fusion of imprecise qualitative information[J]. Applied Intelligence, 2010, 33(3): 340-351

[13] 孙正兴,冯桂焕,周若鸿. 基于草图的人机交互技术研究进展[J]. 计算机辅助设计与图形学学报,2005,17(9):1889-1899

[14] 周若鸿,孙正兴,张莉莎,等. 草图理解技术研究进展[J]. 计算机科学,2004,31(4):140-146

[15] Bimber O,Raskar R. Modern Approaches to Augmented Reality [C]// Proc. of the International Conference on Computer Graphics and Interactive Techniques. San Diego,USA,2007

[16] Graham R L. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set[J]. Inform. Processing Lett., 1972:132-133

(上接第 202 页)

接查询有一定的意义。目前对外包数据库中的查询处理研究还有大量的问题没有解决,例如如何能既安全又有效地实现各种复杂类型的查询,这将是下一步要努力的方向。

参 考 文 献

[1] Hacigumus H,et al. Executing SQL over encrypted data in the database-service-provider model[C]// Proceedings of the ACM SIGMOD International Conference on Managment of Data. Madison,WI,United states,June 2002:216-227

[2] 田秀霞,王晓玲,高明,等. 数据库服务——安全与隐私保护[J]. 软件学报,2010,21(5):991-1006

[3] Agrawal R,et al. Order preserving encryption for numeric data [C]// Proceedings of the ACM SIGMOD International Confe-rence on Management of Data (SIGMOD 2004). Paris, France, June 2004:563-574

[4] Mykletun E, Narasimha M, Tsudik G. Authentication and integrity in outsourced databases[J]. Trans. Storage, 2006, 2(2): 107-138

[5] Xie M,et al. Integrity auditing of outsourced data. VLDB Endowment[C]// Proceedings of the 33rd International Conference on Very Large Data Bases. 2007:782-793

[6] Narasimha M,Tsudik G. Authentication of outsourced databases using signature aggregation and chaining[C]// 11th International Conference of Database Systems for Advanced Applications. Berlin,Germany,April 2006:420-36

[7] Wang H,et al. Dual encryption for query integrity assurance[C]// 17th ACM Conference on Information and Knowledge Management(CIKM'08). October 26-30,2008,Napa Valley,CA,United states,2008:863-872

[8] 咸鹤群,冯登国. 外包数据库模型中的完整性检测方案[J]. 计算机研究与发展,2010,47(6):1107-1115

[9] Hweehwa P, Kian-Lee T. Verifying completeness of relational query answers from online servers[J]. ACM Transactions on Information and Systems Security,2008,11:9-1

[10] Emekci F,et al. Privacy preserving query processing using third parties[C]// 22nd International Conference on Data Engineering (ICDE '06). Atlanta,GA,United states,April 2006:27

[11] Jun L,Omiecinski E R. Efficiency and security trade-off in supporting range queries on encrypted databases[C]// Proceedings of 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security. Berlin,Germany,Aug. 2005:69-83

[12] 朱勤,陆志明. 基于信息隐藏的外包数据库版权保护系统[J]. 计算机科学,2010,37(1):163-166

[13] 夏辉,柏文阳,汪星,等. 数据库安全模型及其应用研究[J]. 计算机应用研究,2005(7):146-147

[14] Yang Y,et al. Authenticated join processing in outsourced databases[C]// SIGMOD. RI,United States,2009:5-17

[15] Wenbo Mao. Modern Cryptography:Theory and Practice[M]. Beijing:Publishing House of Electronics Industry,2004

[16] Pang H H,Tan K L. Authenticating query results in edge computing[C]// Proceedings of 20th International Conference on Data Engineering. Los Alamitos,CA,USA,2004:560-71

[17] Li F, et al. Dynamic authenticated index structures for outsourced databases[C]// 2006 ACM SIGMOD International Conference on Management of Data. Chicago, IL, United States, June 2006:121-132

[18] Open SSL project[EB/OL]. http://www.openssl.org,2010-02-01

[19] Chor B,Goldreich O,Kushilevitz E,et al. Private information retrieval[J]. Journal of the ACM,1988,45(6):965-982