基于有向哈希树的认证跳表算法

徐 剑[1,2] 陈 旭[1] 李福祥[1] 周福才[1]
(东北大学信息科学与工程学院 沈阳 110819)[1] (东北大学软件学院 沈阳 110819)[2]

摘 要 作为一种重要的认证数据结构,认证跳表在数据认证机制中有着广泛的应用。由于哈希模式对认证跳表的代价有显著的影响,因此提出哈希模式和数据存储模式分离的思想,设计了一种新的认证哈希模式——有向哈希树,并在其基础上设计了新的认证跳表算法。应用分层数据处理、概率分析等数学方法对所提出算法的代价进行了理论分析,并与已有的认证跳表算法做了性能比较。结果表明,本算法在时间、通信和存储代价方面有了较大的改进。

关键词 认证跳表,认证哈希模式,有向哈希树,认证数据结构
中图法分类号 TP309 **文献标识码** A

Algorithm of Authenticated Skip List Based on Directed Hash Tree

XU Jian[1,2] CHEN Xu[1] LI Fu-xiang[1] ZHOU Fu-cai[1]
(School of Information Science and Engineering, Northeastern University, Shenyang 110819, China)[1]
(Software College, Northeastern University, Shenyang 110819, China)[2]

Abstract Authenticated skip list is an important authenticated data structures. It has been widely used in data authentication. Since the hash scheme has the important influence on the cost of the authenticated skip list, a new hash scheme which is based on the idea of separating the hash scheme and data storage scheme was proposed in this paper. And the new algorithm of authenticated skip list(ASL-DHT for short) based on directed hash tree was also proposed. We applied hierarchical data processing and probability analysis methods to analyze the cost of ASL-DHT, and also made an algorithm simulation to compare with that of the original authenticated skip list. The results show that, ASL-DHT algorithm has got great improvement on storage cost, communication cost, and time cost.

Keywords Authenticated skip list, Authenticated hash scheme, Directed hash tree, Authenticated data structures

认证数据结构[1,2](Authenticated Data Structure, ADS)是在认证字典[3,4]基础上发展起来的一种分布式计算模型,它可以保证数据的完整性以及端到端可信性,已在证书撤销[5]、数字时间戳[6]以及数据库外包[7,8]中得到广泛应用。

认证跳表[9]是实现 ADS 的一种重要数据结构。它是在跳表[10]基础上,利用跳表节点存储数据元素的哈希值,并通过相应的哈希计算来实现数据认证。与以往跳表不同,认证跳表在跳表节点中存储数据元素的哈希值,同时定义了一些密码学计算方法来实现数据认证。目前的认证跳表算法大都采用 Goodrich 等人提出的算法[9],在该算法中数据的存储模式和哈希模式是一致的,因此其计算代价和存储代价都较高。

本文提出数据存储模式和哈希模式相分离的设计思想,即通过计算和保存存储模式中某些特定节点的哈希值,来有效降低算法的存储和计算代价。为实现数据的存储模式和哈希模式相分离的思想,设计了与跳表存储模式相分离的有向哈希树(Directed Hash Tree, DHT),即在 DHT 中只存储跳表中部分节点的哈希值,因此其存储代价和哈希代价降低为 $O(\log n)$。在 DHT 基础上,设计了认证跳表的新的实现算法——ASL-DHT,它在数据的存储方式上与 Goodrich 认证跳表一致,只是在哈希模式方面采用本文设计的 DHT。由于采用了 DHT,ASL-DHT 在时间、通信和存储代价方面有了很大的改进。

本文第 1 节给出了有向哈希树的分析和设计思想,包括 DHT 的构建算法;第 2 节给出了 ASL-DHT 算法设计,包括 ASL-DHT 算法中的节点插入、节点删除和 ASL-DHT 特征值计算;第 3 节利用分层数据处理[11,12]及概率分析方法对算法的计算复杂度进行分析,通过实验将 ASL-DHT 算法和已有跳表算法进行性能比较;最后是全文总结。

1 有向哈希树分析与设计

1.1 认证哈希模式定义

本小节首先给出认证哈希模式和认证跳表中认证代价的定义。

定义 1 认证哈希模式(Authenticated Hash Scheme,

到稿日期:2010-10-09 返修日期:2010-12-16 本文受国家高技术研究发展计划 863 项目(2009AA01Z122),沈阳市自然科学基金项目(F10-205-1-12)资助。

徐 剑(1978—),博士生,助教,主要研究领域为密码学与网络安全、数据与身份认证技术,E-mail:xuj@mail.neu.edu.cn;陈 旭(1984—),女,硕士生,主要研究领域为密码学与网络安全;李福祥(1984—),男,硕士生,主要研究领域为密码学与网络安全;周福才(1964—),男,博士,教授,博士生导师,主要研究领域为网络与信息安全、可信计算、电子商务基础理论及关键技术。

AHS)AuthHS(S,H,n,m,Γ,Φ) 认证哈希模式 AuthHS 是一个六元组，S 是有序数据元素集合$\{x_1,x_2,\cdots,x_n\}$的存储结构$(x_1\leqslant x_2\cdots\leqslant x_n)$；$n$ 是数据元素个数，m 是哈希元素个数($m\leqslant n$)；H 是 S 中数据元素的哈希值集合$\{y_1,y_2,\cdots,y_m\}$的存储结构，$y_i=h(x_i)$(h 是单向哈希函数，$h:\{0,1\}^*\rightarrow\{0,1\}^n$)，即

$$S\{x_1,x_2,\cdots,x_n\}\xrightarrow{h(.)}H\{y_1,y_2,\cdots,y_m\}$$

Γ 是定义在 S 上进行哈希值选择的函数，即选择哪些哈希值参与认证计算，同时为选择的哈希值构建一个存储和管理方案 $H(\Gamma(S)\rightarrow H)$；$\Phi$ 是定义在 H 上进行数据认证的密码学算法，通过该算法来抽取数据结构的特征值和查询应答证明信息。关于数据特征值的计算方法和查询应答证明信息的产生方法可参考文献[2,9]的内容。

定义 2(认证跳表的认证代价) 基于某哈希方案的认证跳表的认证代价，是指由与认证相关的一些计算所引起的代价，包括：

(1)计算代价 $Cop(H)$：当 S 中的元素更新时，为维护哈希方案，H 中的相关元素需要重新对所产生的代价进行哈希计算；

(2)存储代价 $Stor(H)$：H 中存储哈希值的空间大小；

(3)通信代价 $Com(H)$：查询应答证明信息的大小。

1.2 认证哈希模式分析

在已有的数据认证哈希模式中，数据元素集合的存储模式和哈希模式是一致的，即在一个 AuthHS 中，$S=H$，即数据元素和哈希元素以同样的数据模式进行存储和管理。但在分布式环境中，数据的更新非常频繁，这种方法就存在着以下问题：

(1)计算代价高。对于分布式数据认证，要经常面临大量的数据插入和更新，而这些操作每次执行后，都要重新计算 H 中的所有哈希值，以维护 AuthHS。而哈希计算非常耗时，因此其计算代价是非常巨大的。

(2)存储代价高。在以往的 AuthHS 中，每个节点的哈希值都要进行存储，而不管它是否必需。

针对以上问题，本文设计了一个新的 AuthHS，即基于有向哈希树(Directed Hash Tree，DHT)的认证跳表哈希模式 AuthHS-DHT。AuthHS-DHT 的核心思想是把数据的存储模式和哈希模式相分离，即脱离原有数据结构而独立地定义哈希模式，$H\neq S$。这样，H 只存储 S 中某些节点的哈希值，因此有效降低了算法的存储和计算代价。

1.3 有向哈希树设计

1.3.1 有向哈希树定义

定义 3(有向哈希树，Directed Hash Tree，DHT) DHT T 是哈希值存储结构 H 的具体实现模式，DHT 的节点是通过对 S 中的节点进行相关选择而获得的，并按照一定次序构成一棵有向树，即

$$S\{x_1,x_2,\cdots,x_n\}\xrightarrow{\Gamma}T\{t_1,t_2,\cdots,t_m\}$$

式中，t_i 是 DHT 中的节点，存储 H 中的哈希值 y_i，$t_i\in T(1<i<m)$。

定义 4(哈希路径，Hash Path) 对于 DHT T 的任意叶节点 $t\in T$，从叶节点 t 到根节点的路径 π_s 上的所有哈希节点的集合，称为节点 t 的哈希路径。

(1) 哈希路径大小 $\text{size}(\pi_s)$：π_s 的节点个数，即从叶节点 t 到根节点的节点总数。

(2) π_s 入度的大小 $indeg(\pi_s)$ 是 π_s 中所有节点的入度和，即

$$indeg(\pi_s)=\sum_{v\in\pi_s}indeg(v)$$

在给出 DHT 构建方法前，先介绍两个重要的概念：塔和平行塔。

定义 5(塔，Tower，T) 一个塔 T 由存储相同元素副本的节点组成。在跳表中，一个跳表元素就是一个塔。塔的高度是它的层数(或它的顶层元素所在层)。塔的每个节点有一个指向其对应塔中后续节点的指针和指向它下层元素的指针。一个存储哨兵元素 $-\infty$ 的头塔，被作为跳表中最左边的塔，它的层数比跳表中最高层的塔还要高一层。

如果一个跳表节点是它所在塔的最顶层节点，则它是平节点。

定义 6(平行塔，Parallel Tower，PT) 一个平行塔是由同层塔构成的序列，其中没有更高的塔。使用前向指针，塔的所有平节点都可以通过一个序列到达。对于平行塔有如下性质：

(1) 平行塔大小 $size(|PT|)$ 是平行塔中塔的个数(即序列的大小)；

(2) 塔的子平行塔是与塔的前向指针所连接的平行塔，且这些子平行塔的高度低于塔的高度。

(3) 塔的平行塔是指那些通过塔某一层的前趋指针可以一步到达它们的平节点的塔。如图 1 中的塔 t_5 就是 pt_1 的平行塔。

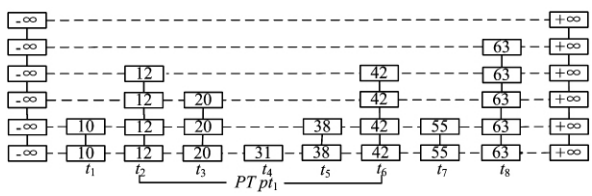

图 1 跳表中的平行塔示例

1.3.2 DHT 构建

DHT 是在原存储方案(跳表)的基础上创建和分配多个 DHT 节点(DHTNode)并将这些节点连接成树。首先对 DHTNode 的数据结构进行说明：

```
class DHTNode{
String hashValue;//DHTNode 哈希值
DHTNode parent;//指向父节点的指针
DHTNode leftchild;//指向左孩子的指针
DHTNode rightchild;//指向右孩子的指针
String KeyValue;//节点值
int level;
}
```

不同于 Goodrich 跳表的哈希结构，DHT 的每个 DHTNode 多指向其父节点、左孩子和右孩子 DHTNode 的指针。由于存储哈希值的开销远高于存储指针成员的开销，因此 ASL-DHT 算法的存储代价仍优于 Goodrich 认证跳表算法。

DHTNode 的分配算法

算法 1 DisDHTNode

输入：PT b

输出：DHTNode 结构

算法：

```
01:{ 在 T t_i(1≤i≤size(|b|)-1)的最底层节 CreateDHTNode d_i;
02:  将其左右孩子指针置为空。
03:  计算 b 的大小 size(|b|),
04:  if(size(|b|)=1) DealSigNL(b);//
05:  else if(size(|b|)>1) DealMulNL(b);//
06:  else return error;
07:}
```

显然,DHT 中存储的节点,依据其左右孩子指针的指向,可以分为 3 种类型:左右孩子都不为空;左孩子为空、右孩子不空;左右孩子都为空。依据 DHTNode 与其子平行塔上的 DHTNode 的对应关系,也可以将塔上的 DHTNode 分为 3 种类型,即分别处于顶层节点、中间节点和底层节点上。对于任意塔,它的 DHTNode 至少包含处于顶层节点的这一种。这两种分类方法将在 ASL-DHT 的特征值计算以及元素的插入与删除中分别用到。算法 1 是哈希节点(DHTNode)分配算法,算法 2 是创建 DHT 的递归算法。

算法 2 createDHT

```
输入:节点 A,当前层数 k
输出:DHT 结构
算法:
01:{ 节点 B=A 的后面节点
02:  if(当前层数=0&&B 是塔节点)
03:      return;
04:  else if(当前层数=0&&B 是平节点)
     {
05:  连接节点 A 及 B,B 为 A 的右孩子数组中第 i 个,i++
06:      调用 createDHT(B,0);
                }
07:  else {
08:      for(自当前层到最底层)
09:        {if(A 为平节点||当前为最底层)
10:          {连接节点 A 及其上层节点,A 作为左孩子;
11:          break;}
                          }
12:        if(A 的左孩子存在){ 调用 createDHT(A 的左孩子
             节点,当前层数-1);}
13:        for(自当前层到最底层)
14:          { if(B 是尾节点) return;
15:          for(B 所在桥上的每一个同层节点 c)
16:                  { c=B 的右孩子数组中的第 j 个,j++}
17:            if(B 是平节点||当前为最底层)
18:            {连接节点 A 及 B,B 为 A 的第 i 个右孩子;
               break;}
                          }
19:        if(B 为非空节点) 调用 createDHT(B,当前层数);
                      }
```

通过这样的方法,跳表中的全部 DHTNode 就联合起来形成了一棵有向哈希树,如图 2 所示。

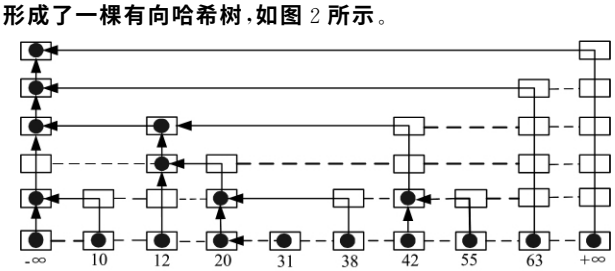

图 2 有向哈希树示例

2 基于 DHT 的认证跳表算法

2.1 插入和删除算法

向跳表中插入新的塔 S 将导致 DHT T 的结构发生变化(某些 DHTNode 的增加、缺失以及左右孩子指针的变化)以及相关 DHTNode 哈希值的重新计算。

首先对用到的变量进行说明。

p:元素指针 p,记录当前塔;

pre:记录当前节点的前驱塔;

$prepre$:记录当前节点前驱塔的前驱塔;

CAL:记录非 S 上的需重算哈希值的 DHTNode;

PRE:记录 S 的顶层 Node 的前驱上的 DHTNode;

$leavebehind$:需要在下一步骤中进一步处理其右孩子指向的 DHTNode。

在插入元素时,首先要确定其前驱和后继。因为 DHTNode 的创建依赖于其所在位置的结构,而且每一个 DHTNode 的创建还可能引起其前驱 DHTNode 的修改,因此在自顶向下插入元素的过程中,每一层都要进行判断,以保证 DHTNode 以及其前后连接的正确性。利用 1.3 节提到的第 2 种 DHTNode 分类方式进行 DHTNode 处理。

跳表插入算法:

(1) p,pre:=跳表的头塔($head$)。

(2)自 p 起依照 S 的索引在跳表中查找元素插入位置,返回查询结果 p,pre。

(3)判断 p 的索引值是否等于 S 的索引,相等则说明跳表中已存在该元素,停止插入,算法结束。

(4)插入 S,具体步骤如下:

①当前层数 i:=S 的最高层。

②自 p 起依照 S 的索引在跳表的第 i 层中查找 S 的插入位置,返回 p,pre。判断 S 的 i 层节点是否是平节点。如果是,则在 S 的 i 层创建 DHTNode 并连接其前驱 pre 和后继 p 上的相应 DHTNode,并转到⑦;否则转到③。

③判断 pre 的 i 层节点是否是平节点。若是,则转到④;否则转到⑤。

④判断 pre 的 i 层节点是否有 DHTNode。若有,则在 S 的 i 层创建 DHTNode 并连接其前驱 pre 和后继 p 上的相应 DHTNode,并转到⑦;没有,则在 pre 的 i 层节点上创建 DHTNode,再在 S 的 i 层创建 DHTNode 并连接其前驱 pre 和后继 p 上的相应 DHTNode,结束后同样转到⑦。

⑤判断 pre 的 i 层节点是否有 DHTNode,若有则转到⑥;否则转到⑦。

⑥先删除 pre 的 i 层节点的 DHTNode,接着在 S 的 i 层创建 DHTNode 并连接其前驱 pre 和后继 p 上的相应 DHTNode。

⑦判断 i 的情况,如果 i 等于 S 的最高层,PRE:=pre 的第 i 层的 DHTNode,转到②;如果 i 等于 1,则结束(4),转到(5);否则 i:=$i-1$,CAL:=S 的 i 层节点上的 DHTNode,并转到②继续执行。

(5)算法结束,返回(4)产生的 PRE 和 CAL。

与元素的插入相同,在删除元素 S 时,也要先确定其每一层的 p 和 pre。但删除操作不同于插入的地方在于:不再需要 PRE,只需一个指针 CAL。CAL 的作用和元素插入算

法相同，也用来指示需要重新计算的哈希路径上的关键节点。

鉴于插入与删除算法相类似，本文予以省略。

2.2 特征值计算算法

建立有向哈希树之后，需要计算 ASL-DHT 的特征值。和 Goodrich 方案相似，这个特征值也是跳表左侧哨兵塔的顶层 DHTNode 的哈希值。为了得到该值，需要计算该 DHT 上的各个 DHTNode 的哈希值，利用在 1.3 节的第一种 DHTNode 分类方式，计算任意 DHTNode t 的哈希值如下

(1)如果 t 为结束节点，则将其哈希值设为 0。

(2)如果 t 的左孩子和右孩子为空，即 t 为叶子节点，则 t 的哈希值为 $h(t.Value)$。

(3)如果 t 的左孩子为空，右孩子不为空，则 t 的哈希值为 t 的元素值和 t 的右孩子的哈希值的联合哈希，即为 $h(t.Value, t.rightchild.HashValue)$。

(4)如果 t 的左孩子和右孩子都不为空，则 t 的哈希值为 t 的左孩子和右孩子的哈希值的联合哈希，即为 $h(t.leftchild.HashValue, t.rightchild.HashValue)$

3 算法代价分析与比较

3.1 代价分析

本文提出的基于 DHT 的认证跳表算法，其数据存储结构 S 采用跳表，因此其数据的查询、存储代价都与传统跳表方案相一致。在 AuthHS-DHT Δ 中，哈希方案和数据存储方案不同，因此对于认证算法的代价分析将基于 DHT 进行。

由 DHT T 的结构特征、哈希路径 π_t 的大小和入度大小可以描述 AuthHS-DHT 的计算和通信代价。由定义 2，AuthHS-DHT Δ 的代价定义如下：

(1)计算代价 $Cop(\Delta)=indeg(\pi_t)$；

(2)通信代价 $Com(\Delta)=size(\pi_t)$；

(3)存储代价 $Stor(\Delta)=size(T)$。

对 Δ 的代价分析可以转化为对 π_t 和 T 的数学期望的分析。

在本文提出的 AuthHS-DHT Δ 中，S 采用概率参数为 p 的跳表作为存储结构，层数 $L(n)=\log_{1/p}n$，对于 T 的叶节点 t，令 $C(\pi_t)$ 代表 π_t 的认证代价，则 $C(\pi_t)=size(\pi_t)+indeg(\pi_t)$，$\pi_t$ 和 T 的数学期望如下

$$E[size(\pi_t)]\leqslant 2(1-p)L(n)+O(1) \tag{1}$$

$$E[indeg(\pi_t)]\leqslant (1-p)\frac{(1+p)^2}{p}L(n)+O(1) \tag{2}$$

证明：本文利用分层数据处理，采用概率分析方法来计算 π_t 和 T 的相关数学期望。

假设跳表大小是无限的，平行塔 PT 最左边塔为 L，其顶层跳表节点 u 的后继指针指向其下一跳表节点 v。令 L_v 表示 v 所在的塔，如果 $L_v\in PT$，那么节点 v 是 L_v 顶层节点的概率是 $1-p$；如果 $L_v\notin PT$，那么 L_v 有更高层数的概率为 p。任意平行塔 PT 中有平均 $1/p$ 个塔是连续且高度相同的，即每个塔所在平行塔的大小平均为 $1/p$。令 Y 表示平行塔 PT 的大小，则 Y 服从参数为 p 的几何分布，即

$$Y\sim G(k,p), P(Y=k)=q^{k-1}p(q=1-p), E[Y]=\frac{1}{p}$$

下面，对式(1)和式(2)进行证明。

首先，逆向遍历元素 x 的查找路径 π_t，即从元素 x 底层节点开始，逆向查找到 DHT 的根节点。按最坏情况分析，假设 π_t 到达层 $L(n)$。这里可以把 π_t 分为两部分：子路径 π_1 和 π_2。路径 π_1 到达层 $L(n)$，路径 π_2 完成逆向查找并最后到达 π_t 的第一个跳表节点。对于 π_1 引起的认证代价，显然有

$$\pi_t=\pi_1\cup\pi_2 \tag{3}$$

$$size(\pi_t)=size(\pi_1)+size(\pi_2) \tag{4}$$

$$indeg(\pi_t)=indeg(\pi_1)+indeg(\pi_2) \tag{5}$$

π_2 的节点、入度和大小的平均值都是常数，即

$$E[size(\pi_t)]=O(1) \tag{6}$$

$$E[indeg(\pi_2)]=O(1) \tag{7}$$

子路径 π_2 可进一步分解为节点集合 L 和 U。集合 L 中的节点是向左逆向遍历所访问的节点，集合 U 是向上移动操作所访问的节点。令随机变量 X 表示跳表中高度大于等于 $L(n)$ 的塔的数目，Z 表示跳表中大于层 $L(n)$ 的最高塔的大小，有 $|L|\leqslant X, |U|\leqslant Z$。因为一个塔的层数大于等于 $L(n)$ 的概率为 $1/n(1-p)$，因此 $X\sim B(n,1/n(1-p))$，即 X 服从二项分布，同时 $Z\sim G(1-p)$。则有 $E[L]\leqslant E[X]=1/(1-p)$，$E[U]\leqslant E[Z]=1/(1-p)=O(1)$。由此可得，$size(\pi_2)$ 的数学期望为 $E(size(\pi_2))=E[L]+E[U]=O(1)$，式(6)得证。又因为每个 π_2 中的节点，都有常数级入度，因此 $E(indeg(\pi_2))=O(1)$，式(7)得证。

在分析 $size(\pi_1)$ 的时候，假设跳表向左无限，用随机变量 $C_k(i)$ 表示已经执行到第 i 步向上(或向左)移动操作，还有 k 次操作剩余时的认证代价。若向上移动，则 $C_k(i)=X_U+C_{k-1}(i+1)$，若向左移动，则 $C_k(i)=X_L+C_k(i+1)$，其中 X_U 和 X_L 分别表示向上或向左时遇到 DHTNode 的 0-1 随机变量。到达节点的后继指针指向节点为平节点的概率为 $1-p$，到达节点为非平节点的概率为 p，可知 $\Pr[X_U=1]=p(1-p)$，X_U 的期望值 $E[X_U]=p(1-p)$；因为向左移动离开平行塔 PT(即 $|P|=1$)的概率为 p，$|P|>1$ 的概率为 $1-p$，可知 $\Pr[X_L=1]=p+p(1-p)$，同理得 X_L 的期望 $E[X_L]=p+p(1-p)$，应用条件期望

$$\begin{aligned}E[C_k]&=pE[C_k|up]+(1-p)E[C_k|left]\\&=p(p(1-p)+E[C_{k-1}])+(1-p)(p+\\&\quad p(1-p)+E[C_k])\end{aligned}$$

整理得 $E[C_k]=E[C_k-1]+2(1-p)$，即 $E[C_k]=2(1-p)k$，可得 $E[size(\pi_1)]=2(1-p)L(n)$，式(1)得证。

可以用类似方法分析 π_1 的入度大小 $indeg(\pi_1)$。因此式(2)得证。

为简化比较，本文选择跳表参数 $p=1/2$，节点总数约为 $2n$。表 1 给出了 Goodrich 认证跳表算法和 ASL-DHT 算法复杂度比较。

表 1 Goodrich 认证跳表算法和 ASL-DHT 算法复杂度比较

	Size(π_t)	indeg(π_t)	Size(T)	S
Goodrich	1.5logn	3logn	2n	2n
ASL-DHT	logn	2.25logn	1.25logn	2n

3.2 实验比较

本节对 ASL-DHT 算法和 Goodrich 认证跳表算法进行性能比较。实验在 Inter(R) Core(TM)2 CPU 6300 1.86 GHz，2G 内存的机器上进行，操作系统为 Windows XP，算法实现利用 Sun 的 Java JDK 1.5，单向哈希函数的实现利用 Java 提供的 SHA-1 算法。

(下转第 63 页)

[8] Dellarocas C. The Digitization of Word-of-Mouth: Promise and Challenges of Online Reputation Mechanism [J]. Management Science, 2006, 49(10): 1407-1424

[9] Yu B, Singh M P. A Social Mechanism of Reputation Management in Electronic Communities [C]// Proceedings of Fourth International Workshop on Cooperative Information Agents (CIA 2000). Boston, USA, 2000: 154-165

[10] Wasserman S. Social Network Analysis: Methods and Applications (1st ed) [M]. Cambridge: Cambridge University Press, 1994

[11] Golle P, Leyton-Brown K, Mironov I. Incentives for sharing in peer-to-peer networks[C]// Wellman MP, Shoham Y, eds. Proceedings of the 3rd ACM Conf. on Electronic Commerce. New York: ACM Press, 2001: 264-267

[12] Buragohain C, Agrawal D, Suri S. A game theoretic framework for incentives in P2P systems[C]// Shahmehri N, Graham R L, Carroni G, eds. Proceedings of the 3rd Int'l Conf. on Peer-to-Peer Computing (P2P 2003). Los Alamitos: IEEE Press, 2003: 48-56

[13] Friedman E, Resnick P. The social cost of cheap pseudonyms [J]. Journal of Economics and Management Strategy, 2001, 10 (2): 173-199

(上接第 35 页)

两种算法的存储空间大小对比如图 3 所示。图 4 给出的是 Goodrich 认证跳表算法和 ASL-DHT 算法在元素认证时进行 Hash 计算次数的对比。

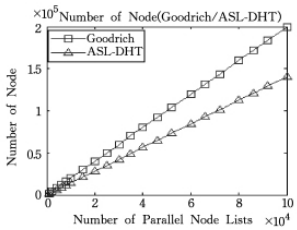

图 3 算法空间大小对比(Goodrich/ASL-DHT)

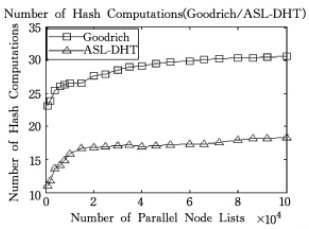

图 4 Hash 计算次数对比(Goodrich/ASL-DHT)

图 5 和图 6 给出的是 Goodrich 和 ASL-DHT 算法元素插入时更新节点数目和插入算法的运行时间对比。

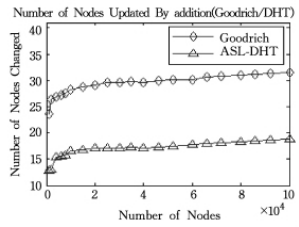

图 5 元素插入时更新节点数目对比(Goodrich/ASL-DHT)

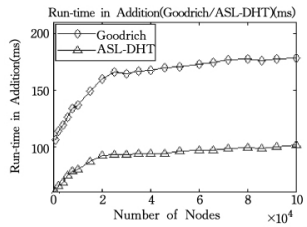

图 6 插入算法的运行时间对比(Goodrich/ASL-DHT)

Goodrich 和 ASL-DHT 算法元素删除时更新节点数目和元素删除算法的运行时间对比如图 7 和 8 所示。

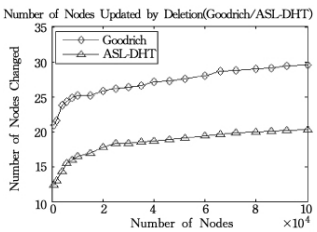

图 7 元素删除时更新节点数目对比(Goodrich/ASL-DHT)

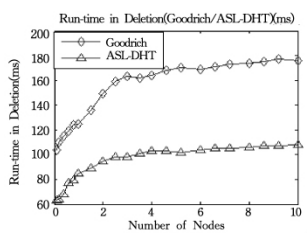

图 8 删除算法的运行时间对比(Goodrich/ASL-DHT)

以上的实验结果表明,在存储空间大小、元素插入时更新节点数目、元素插入算法的运行时间、元素删除时更新节点数目、元素删除算法的运行时间等方面,ASL-DHT 算法均优于 Goodrich 认证跳表算法。

结束语 针对 Goodrich 认证跳表算法存在的问题提出数据存储方案和哈希方案相分离的思想,并依据此思想设计并实现了一种新的基于有向哈希树的认证跳表算法,从而有效地解决了 Goodrich 认证跳表的节点哈希值冗余和节点哈希值重计算量大的问题。理论代价分析和实验比较结果表明,ASL-DHT 在存储空间大小、元素插入时更新节点数目、元素插入算法的运行时间、元素删除时更新节点数目和元素删除算法的运行时间方面都优于 Goodrich 认证跳表算法,具有很高的效率和可行性及重要的理论价值。

参 考 文 献

[1] Tamassia R. Authenticated Data Structures[C]// Proceedings of the Algorithms-ESA 2003. LNCS, September 2003: 2-5

[2] Papamanthou C, Tamassia R. Time and Space Efficient Algorithms for Two-Party Authenticated Data Structures[C]// Proceedings of ICICS 2007. LNCS, 2007: 1-15

[3] Crosby S A, Wallach D S. Super-efficient Aggregating History-independent Persistent Authenticated Dictionaries [C]// Proceedings of ESORICS 2009. LNCS, June 2009: 671-688

[4] 周永彬,卿斯汉,薛源,等. 基于时间约束的认证字典分类方法[J]. 计算机科学, 2004, 31(7): 20-22

[5] Muñoz J L, Forne J, Esparza O, et al. Certificate revocation system implementation based on the Merkle hash tree[J]. International Journal of Information Security, 2004, 2(2): 110-124

[6] Blibech K, Gabillon P. CHRONOS: an authenticated dictionary based on skip lists for timestamping systems[C]// Proceedings of the 2005 Workshop on Secure Web Services. ACM Press, Nov 2005: 84-90

[7] 朱勤,于守健,乐嘉锦,等. 外包数据库系统安全机制研究[J]. 计算机科学, 2007, 34(2): 152-156

[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] Goodrich M, Tamassia R, Schwerin A. Implementation of an authenticated dictionary with skip lists and commutative hashing [J]. DISCEX II, 2001, 55(9): 889-903

[10] Pugh W. Skip lists: a probabilistic alternative to balanced trees [J]. Communications of the ACM, 1990, 33(6): 668-676

[11] Tamassia R, Triandopoulos N. Computational bounds on hierarchical data processing with applications to information security [C]// Proceedings of ICALP 2005. LNCS, 2005, 3580: 153-165

[12] Xu Jian, Zhou Fu-cai, Li Xin-yang, et al. Hierarchical Data Processing Model and Complete Tree Key Management Mechanism [C]// Proceedings of ICYCS 2008. IEEE Computer Society, Nov 2008: 1606-1612