Multi-dimensional Query Authentication for On-line Stream Analytics

Xiangrui Chen[1], Gyoung-Bae Kim[2] and Hae-Young Bae[1]
[1] Dept. of Computer Science and Engineering, Inha University
253 Yonghyun-dong, Nam-gu Incheon, 402-751 - Korea
[e-mail: airchendb@gmail.com, hybae@inha.ac.kr]
[2] Dept. of Computer Education, Seowon University
241 Musimsero, Heungdeok-gu, Cheongju, 361-742 - Korea
[e-mail: gbkim@seowon.ac.kr]
*Corresponding author: Gyoung-Bae Kim

Received January 28, 2010; revised March 17, 2010; accepted April 15, 2010; Published April 29, 2010

Abstract

Database outsourcing is unavoidable in the near future. In the scenario of data stream outsourcing, the data owner continuously publishes the latest data and associated authentication information through a service provider. Clients may register queries to the service provider and verify the result's correctness, utilizing the additional authentication information. Research on On-line Stream Analytics (OLSA) is motivated by extending the data cube technology for higher multi-level abstraction on the low-level-abstracted data streams. Existing work on OLSA fails to consider the issue of database outsourcing, while previous work on stream authentication does not support OLSA. To close this gap and solve the problem of OLSA query authentication while outsourcing data streams, we propose MDAHRB and MDAHB, two multi-dimensional authentication approaches. They are based on the general data model for OLSA, the stream cube. First, we improve the data structure of the H-tree, which is used to store the stream cube. Then, we design and implement two authentication schemes based on the improved H-trees, the HRB- and HB-trees, in accordance with the main stream query authentication framework for database outsourcing. Along with a cost models analysis, consistent with state-of-the-art cost metrics, an experimental evaluation is performed on a real data set. It exhibits that both MDAHRB and MDAHB are feasible for authenticating OLSA queries, while MDAHRB is more scalable.

Keywords: Database outsourcing, multi-dimensional query authentication, on-line stream Analytics (OLSA), stream cube

DOI: 10.3837/tiis.2010.04.005

1. Introduction

Because of its various advantages, database outsourcing [1] is currently attracting an increasing number of researchers. The essential goal is to enable the Data Owners (DOs) to publish their data/database through reliable third parties, named service providers (SPs).

Generally, SPs are the groups/companies that can maintain the necessary platform (e.g. hardware) to support various advanced services on the basis of single or multiple databases. The clients can register their query requests directly to the SP, without communication with the DO. This framework benefits every party involved: 1. It does not matter whether the DO can afford the platform expense, 2. The SP may run various legal businesses based on the database(s), and achieve economies of scale by serving multiple DOs/clients, 3. The nearest SP can respond to the clients' queries with lower overhead with respect to the network latency [2].

However, since the honesty of the SPs is a factor beyond the control of others (e.g. DOs, clients), techniques for the security and protection of the database are vital before outsourcing actions. Furthermore, when clients obtain query results from SPs, they also want to be granted the ability to authenticate the results. Generally, query results have two dimensions that should be guaranteed, soundness and completeness. In detail, soundness assures the client of the existence of returned records in the DO's database, without modification or any unauthentic records. Completeness means that the returned records are guaranteed to cover all the records that satisfy the query. Here we illustrate the idea of this paper by extending the framework of the Next Generation SIMulation (NGSIM) program to an outsourced Intelligent Transportation System (ITS). It was initiated by the U.S. Federal Highway Admistration (FHWA) [3]. As depicted in **Fig. 1**, Cambridge Systematics, Inc. (CSI) manages the collection of relevant information (e.g. videos, sensor data, maps). Replying to requests from the client (FHWA), it can support historical traffic analysis corresponding to a specific time period, relying on the collected disk-resident data. By analyzing the historical summary report sent by CSI (DO), transportation professionals in FHWA (clients) can make substantial decisions on topics like new roadway alignments and configurations, new interchange configurations and locations, the addition of freeway auxiliary lanes, etc. It is meaningful to urban planning.

However, since traffic data is always in the form of data streams, whereas there is no stream processing approach adopted, this framework cannot support monitoring/ scheduling functions, in contrast to ITS.

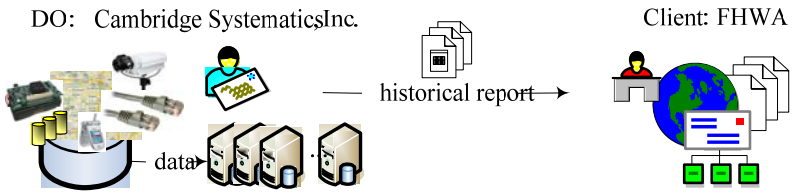
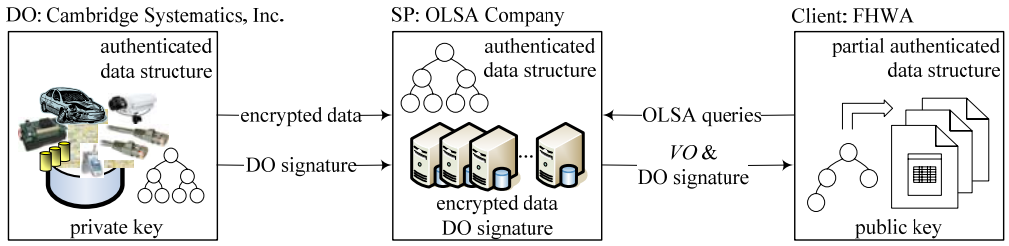


Fig. 1. Next Generation SIMulation (NGSIM).

Furthermore, most data streams reside at a rather low level of abstraction (e.g. coordinate streams), while an analyst is often more interested in higher multi-level abstraction. A multi-dimensional on-line stream data analysis is needed, since it plays the kernel role in

understanding the general statistics, trends (e.g. traffic jams) and outliers (e.g. accidents) for traffic scheduling. Research on this substantial characteristic of data streams, low-level-abstracted, sets a new direction, extending data cube technology for multi-dimensional analytics of data streams [4], i.e. On-line Stream Analytics (OLSA) [5]. There is an essential difference between On-line Analytical Processing (OLAP) [6] and OLSA: OLAP is related to OLTP for historical reporting, while OLSA handles issues like monitoring, alerting, transformation, real-time visibility and reporting [5].

In other words, OLAP cannot support on-line data stream aggregation in contrast to OLSA. However, since the expense of the necessary computational power and platform for OLSA is unacceptably high for some companies (e.g, CSI in **Fig. 2**), data stream outsourcing is unavoidable. Although relevant works on OLSA are noteworthy, none of them examines the outsourcing issue, which motivates the idea of this paper, extending NGSIM to support OLSA in outsourcing environments, taking into account query authentication issues.



OLSA query example: What is the number and average velocity of trucks (vehicle class) on the 3rd lane?

Fig. 2. NGSIM outsourcing framework example.

The outsourced NGSIM framework example is depicted in **Fig. 2**. Following an asymmetric cryptosystem (public key digital signature scheme) [7], the DO (CSI) obtains a private and public key from the certificate authority. The private key can be known and used by DO to generate the signature for the outsourced information, while the public key is published and any client can use it. To enable the client to perform query authentication, DO constructs an Authenticated Data Structure (ADS), which is much the same as a conventional index.

However, necessary additional authentication information is contained, e.g. hashing values signed by the DO. Receiving the encrypted data and DO signature, the SP maintains and updates the same ADS. When a client (FHWA) registers a continuous query, SP retrievals the local ADS and generates a Verifcation Object (VO). Obtaining the VO and DO signature sent by SP, the client can not only get the result, but also verify its correctness by reconstructing a partial ADS. Alternative implementations of query authentication approaches differ on the choice of signature techniques, design of ADS and verification algorithms [2]. In addition to the primary consideration for disk-resident data sets, authenticating continuous stream queries needs to accommodate more notable challenges, e.g. continuous validation upon fast updates, cost minimization mechanisms for communication and verification, necessary integration of relevant data sources, etc. Existing work on stream authentication considers these factors, however, they ignore the low-level-abstracted features of data streams, and do not support OLSA for high-level analysis. The idea of this paper aims to close this gap by proposing an authentication scheme for OLSA in outsourcing environments. In our work, we first improve the general approach for OLSA, stream cube [4], by proposing HRB- and HB-trees. Then, we propose MDAHRB and MDAHB, two multi-dimensional authentication schemes for OLSA in data stream outsourcing environments. Along with a cost model analysis consistent with

state-of-the-art cost metrics, an experimental evaluation is performed on a real data set (NGSIM U.S. 101 [3]). Our results exhibit that both MDAHRB and MDAHB are feasible for authenticating OLSA queries, while MDAHRB is more scalable. The rest of the paper is organized as follows. Section 2 reviews the relevant work. Section 3 presents the MDAHRB idea, while Section 4 focuses on MDAHB demonstration. A detailed experimental evaluation is illustrated in Section 5, and Section 6 concludes the paper with recommendations for future work.

2. Related Work

Research on multi-dimensional query authentication for OLSA is an interdisciplinary work between databases and security. Different from traditional query authentication for ad-hoc queries with single dimensional processing, multi-dimensional query authentication involves more dimensions.

For example, since OLSA processing needs to consider both spatial and temporal dimensions, as explained in Section 2.3, query authentication for OLSA becomes a challenging task for which we provide solutions in Section 3 and 4. In this section, we will present the relevant work in these fields.

2.1 Cryptography Essentials

The public-key digital signature scheme presented in Section 1 aims at verifying the integrity of a message, which originates from the DO. Thus, it is crucial to design structures that use the minimum number of signing operations and utilize hashing instead. This is due to the high cost of signing compared with hashing [8], as confirmed by experiments with two widely used cryptography libraries, Crypto++ [9] and OpenSSL [10]. Hashing is a magic function H with two typical properties: 1. No inverse function, which means that for any x, it is easy to compute H(x), while given H(x), it is impossible to find any information about a pre-image x, 2. Collision resistance, which means that it is impossible to find a pair of (x,y) statisfying $x \neq y$ and H(x)=H(y) [7]. In this paper, we employ SHA1 [11], which takes variable length inputs and produces 160-bit (20-byte) outputs, and adopt the 128-byte digital signatures generated by RSA.

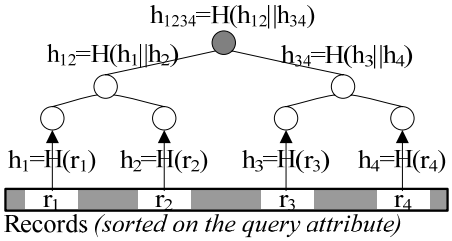


Fig. 3. A Merkle hash tree (MH-tree).

The Merkle Hash Tree. The MH-tree is a main-memory binary hash tree first proposed in [12]. It is constructed in a bottom-up manner by hashing records, which are sorted on the query attribute. As depicted in **Fig. 3**, every leaf node in the MH-tree contains the hash value of a record, and each internal node stores the hash value of the concatenation of its two children. The DO signs the hash value stored in the root and generates a DO signature sign(h_{root}, SK) using the private key SK, e.g. sign(h_{1234}, SK). When a client sends a query that corresponds to

a record r_x (e.g. r_4), the SP traverses the tree and creates a VO by inserting the hash values stored in the siblings of every accessed node, $h_1 \ldots h_n$ (e.g. h_{12}, h_3), in addition to the query result r_x (e.g. r_4). Thus, the VO should be in the form of VO($h_1 \ldots h_n$), e.g. VO(h_{12}, h_3). Given the result r_x, VO, sign(h_{root}, SK), and DO's public key PK, the client obtains the decrypted h_{root} and a hash value h'_{root} by iteratively constructing a partial ADS, i.e. $h'_{root}=H(h_{12}|H(h_3,H(r_4)))$. By matching h'_{root} against h_{root}, the correctness of r_x returned from the SP can be authenticated, where soundness is guaranteed by the collision resistance of hash function H, and completeness is ensured by the construction of the MH-tree.

2.2 Query Authentication

The issue of database outsourcing first appeared in [1] and later a great deal of work on query authentication for database outsourcing appeared [2][8][13][14][15], most of which applies the concept of the MH-tree. Previous work generally utilizes two approaches, signature- and index-based. A good survey is provided in [14], but our review is brief.

The Merkle B-tree (MB-tree) [8] is an improved MH-tree where the node fanout is determined by the PC's block size, motivated by the idea of the B+-tree. The MB-tree is disk-based, and does not consider the stream environment. Regarding authenticating queries on the data stream, references are scarce. To the best of our knowledge, [2] and [13] is the most predominant work. It is impossible to apply a signature-based approach to the data stream, because of the high signing cost upon infinite data streams, as a result of which the authentication schemes established in [2][13] are both index-based approaches. Researchers have successfully confronted most challenges within processing and authenticating continuous queries by proposing CADS [2], an elaborate indexing scheme and a virtual caching mechanism. Furthermore, it is extended to the general case where multiple DOs outsource their data to the same SP. The most relevant work to ours is [13]. They proposed structures for authenticating multi-dimensional selection and aggregation queries over sliding windows on data streams. The indices are implemented by extending the kd-tree, and were firstly applied to work for multi-dimensional queries. With similar techniques, the R-tree was extended for spatial database outsourcing [14]. Our work is distinguished from [12][13] because CADS does not consider multi-dimensional authentication. The tuple-based sliding window authentication index structure does not take into account the time dimension [13], which is a substantial attribute of data streams. Moreover, the sliding window-based aggregation approach is limited by the window size. We aim to close this gap by extending the data cube technology.

In this case, our work differs from [12][13] because we focus more on a different branch, multi-dimensional query authentication for OLSA, and support both provisions, temporal soundness [2] and multi-dimensional aggregation [13]. State-of-the-art cost metrics for evaluating query authentication schemes are given in [8]: 1. The computation overhead for the owner, 2. The DO-SP communication cost, 3. The storage overhead for the server, 4. The computation overhead for the server, 5. The client-SP communication cost, and 6. The computation cost for the client (for verification). For consistency with previous work, we follow these metrics in this paper.

2.3 On-line Stream Analytics (OLSA)

Due to the limited resources currently available to computers (e.g. memory, disk, etc.), it is impossible to store an entire data stream or to scan through it multiple times. Although previous studies (e.g. management and querying of stream data, data mining on data streams)

have established fundamental techniques to process data streams efficiently, research on OLSA is still in its infancy. Corresponding to various applications, different data structures, processing approaches and validated algorithms need to be specified, e.g. RFID [16], sequence data [17]. Most of these approaches adopt the data cube technique.

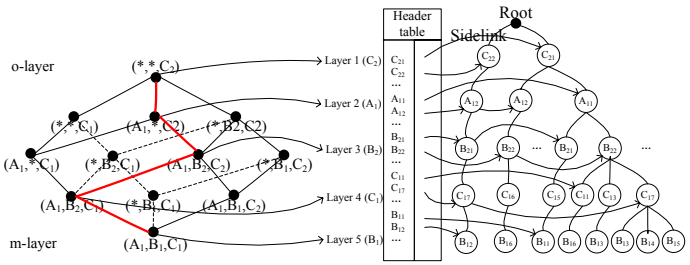


Fig. 4. Stream cube from m- to o-layer.

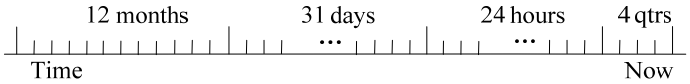


Fig. 5. A tilt time frame model.

In our work, we consider the general OLSA approach, stream cube [4], shown in **Fig. 4**, which is proposed to enable on-line, multi-dimensional, multi-level analysis of data streams. Specifically, we focus on extending it with a feasible authentication approach to facilitate OLSA in database outsourcing environments (see **Fig. 2**). There are three notable techniques within the stream cube: 1. A tilted time frame model (see **Fig. 5**), proposed as a multi-resolution model to register time-related data, 2. Critical layers, i.e. observation layer (o-layer) and minimal interesting layer (m-layer) (see **Fig. 4**), which enables flexible analysis with low overhead, 3. An efficient cubing method, which only computes and maintains the layers (cuboids) along a popular path (see **Fig. 4**, in bold red), leaving the other cuboids for query-driven, on-line computation. As illustrated in **Fig. 4**, based on these techniques, raw low-level-aggregated data streams (e.g. sensor coordinates streams) can be pre-aggregated to the m-layer, (A_1, B_1, C_1), according to the tilted time scale. Concurrently, it is further aggregated following the popular drilling path (in bold red) to reach the observation layer, (*, *, C_2). Drilling from the o- to m-layer, there are a total of five materialized layers (cuboids) during cube computation, within which relevant queries that precisely rely on these layers can be answered directly without any further computation. Meanwhile, queries that deviate from the path can be answered by on-line computation based on the reachable computed layers. As established by previous work [4] [18][19][20][21], it is feasible to utilize the stream cube for OLSA, computed and stored by a main-memory data structure, the H-tree. It is a hyper-linked tree structure first introduced by [18]. It was revised in [4] to ensure that it can be maintained in memory for efficient computation of multi-dimensional, multi-level aggregations requested by continuous OLSA queries. Detailed construction steps can be found in [4]. The proposed authentication methods are based on the authenticated data structures, HB- and HRB-tree, which will be introduced in Section 3.1 and 4.1. However, both of them are the extension of the H-tree in **Fig. 4**.

The latest work that considers the problem of secure outsourced aggregation, which seems to be similar to our idea, is based on the unified use of a one-way chain [22]. However, it is from the perspective of data collection and aggregation outsourcing, and does not support

query answering and authentication. Meanwhile, our method is to extend the OLSA solution, the stream cube, to the outsourcing future of OLSA, and support query authentication. Also, the stream cube itself can support data aggregation.

3. Multi-dimensional Authentication Method - MDAHRB

Section 3.1 illustrates the idea of ADS in MDAHRB. Section 3.2 describes the authentication scheme. Section 3.3 summarizes the cost models of MDAHRB. The summary of notations used are listed in **Table 1**.

Table 1. Summary of notations used

Symbol	Description	Symbol	Description
R	A data stream tuple	f_x	Node fanout of structure x
S_R	Number of arrived data stream tuples	|x|	Size of object x
N	An H-tree node	h_x	A hash value on x
Q_{node}	A queue node inside N	Sig_x	A signature on x
RBN	A RB-tree indexed head table node	C_x	Cost of operation x
BN	A B-tree indexed head table node	H(x)	A hash operation on input x
d_x	Height of structure x	VO	The verification object

3.1 HRB-tree

Based on the H-tree depicted in **Fig. 4**, information can be stored, updated, aggregated and accessed at different levels. When a new tuple R arrives, it will be expanded to the m-layer schema according to a concept hierarchical tree and inserted into the H-tree, with popular path cuboids aggregated/computed inside the internal nodes. To sum up, for a specific cell's information, which may be distributed among several same-labeled nodes in the layer, the head table plays the role of linking them all together. Queries that precisely rely on these layers can be directly returned by retrieving the head table in order to find all same-labeled nodes, and examining whether it relates to the query (by checking the upper nodes). Briefly, the ADSs are constructed by indexing the head table nodes.

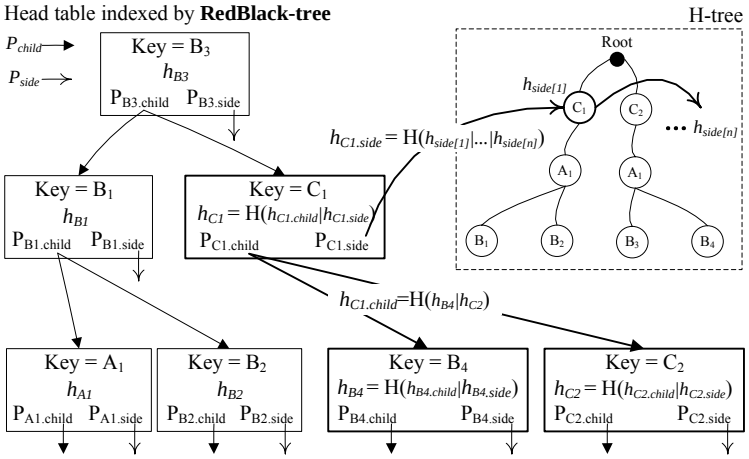


Fig. 6. HRB-tree for OLSA authentication.

For MDAHRB, the ADS improves the H-tree by adopting the Red-Black tree to organize the head table. The intuitions behind this framework are: 1. The cost of tree construction can be reduced by indexing the frequently accessed head table, 2. It is necessary to enable hash operations of query authentication through a global tree structure. The height balance and binary organized features of the Red-Black tree satisfy our intuitions. In this case, the ADS (indexing structure) is designed to be a collaboration of the H- and RB-trees, named the HRB-tree, which is illustrated in **Fig. 6**. Both the SP and the DO maintain and update the HRB-tree locally.

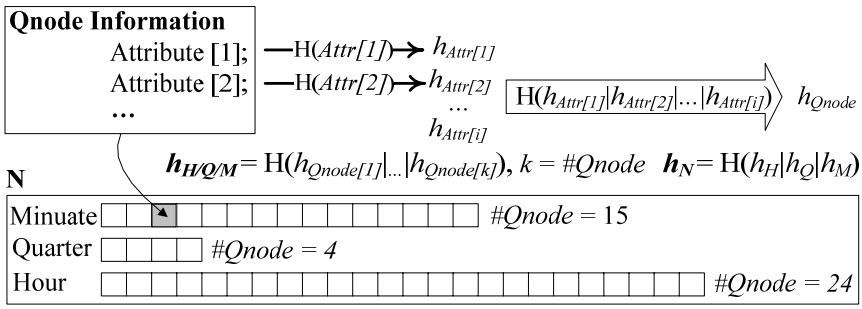


Fig. 7. H-tree node hashing.

The hashing scheme in the HRB-tree, especially the organization in part of the RB-tree, is similar to the aforementioned MH-tree, whereas it distinguishes from the MH-tree. Specifically, since it is for authenticating data streams, the hash operations in MDAHRB are expected to be performed in a query-driven manner. Moreover, the H-tree is constructed dynamically paralleling the incoming Rs, while the RB-tree is constructed before any processing , which should in principle organize all the head table nodes of the Red-Black tree.

Algorithm 1 HNodeHashing(N)
Require: N – A node in H–tree
Ensure: h_N – Hash value in node N
1. let *pHour*,*pQuarter*,*pMinuate* point to tilt time queues
2. for *pHour*[1] to *pHour*[24] do
3. $h_{Attr[i]} <= H(pHour.Attr[i])$
4. $h_{Qnode[i]} <= H(h_{Attr[1]}|…|h_{Attr[i]})$
5. $h_H <= H(h_{Qnode[1]}|…|h_{Qnode[24]})$
6. for *pQuarter*[1] to *pQuarter*[4] do
7. $h_{Attr[i]} <= H(pQuarter.Attr[i])$
8. $h_{Qnode[i]} <= H(h_{Attr[1]}|…|h_{Attr[i]})$
9. $h_Q <= H(h_{Qnode[1]}|…|h_{Qnode[4]})$
10. for *pMinuate*[1] to *pMinuate*[15] do
11. $h_{Attr[i]} <= H(pMinuate.Attr[i])$
12. $h_{Qnode[i]} <= H(h_{Attr[1]}|…|h_{Attr[i]})$
13. $h_M <= H(h_{Qnode[1]}|…|h_{Qnode[15]})$
14. $h_N <= H(h_H|h_Q|h_M)$

Algorithm 2 HTreeHashing(N, HtreeEx)
Require: N – A node in H–tree, HtreeEx – An H–tree
Ensure: HtreeEx is hashed
1. if N is a leaf node then
2. HtreeEx.HNodeHashing(N)
3. return true
4. else
5. let *Ci* be N's children
6. HtreeEx.HNodeHashing(N)
7. for all *Ci* do
8. HTreeHashing(*Ci*, HtreeEx)

Fig. 8. H-tree node hashing and H-tree hashing algorithms.

Suppose a query is detected. Before discussing RB-tree hashing, the H-tree will be hashed as follows. As the cell, each node of the H-tree (N) incorporates a tilt time frame model to maintain historical information. It is extended to support OLSA query authentication by computing the hash value of N (h_N). Taking the frame of 15 minutes, 4 quarters, 24 hours as an example, the hashing operations inside N are organized as shown in **Fig. 7**, and h_N is calculated

following the steps shown in **Fig. 8** (Algorithm 1). In detail, there are three time queues inside N. Within each queue node (Qnode), multi-dimensional information is summarized by several attributes, e.g. time, coordinates, etc. The hash values of these original dimensional records, $h_{Attr[i]}$, are computed respectively(lines 3, 7, 11). We can obtain the hash value of each Qnode by hashing the concatenation ('|') of all $h_{Attr[i]}$ (lines 4, 8, 12). Similarly, we can obtain the hash value of all queues (i.e. h_H, h_Q, h_M) (lines 5, 9, 13) and the final h_N (line 14). Based on the hash operation on N, HNodeHashing(N), it traverses and hashes the entire H-tree following the steps described in **Fig. 8** (Algorithm 2), calling HTreeHashing(N, HtreeEx) recursively.

Algorithm 3 RBNodeHashing(RBN)
Require: RBN – A node in RB-tree indexed head table
Ensure: h_{RBN} – Hash value in node RBN
1. **if** RBN is a leaf node **then**
2. $h_{RBN.child} <= H(0)$
3. **else**
4. $h_{RBN.child} <= H(h_{RBN.lchild} | h_{RBN.rchild})$
5. $h_{RBN.side} <= H(h_{side[1]} | \ldots | h_{side[n]})$
6. $h_{RBN} <= H(h_{RBN.child} | h_{RBN.side})$

Algorithm 4 RBTreeHashing(RBN, HRBtreeEx)
Require: RBN – A node in RB-tree indexed head table, HRBtreeEx – An HRB-tree
Ensure: HRBtreeEx.RBtree is hashed
1. **if** *RBN.left* = *nil* and *RBN.right* = *nil* **then**
2. HRBtreeEx.*RBNodeHashing*(*RBN*)
3. return true
4. **else if** *RBN.left* = *nil* and *RBN.right* ≠ *nil* **then**
5. RBTreeHashing(RBN.right, HRBtreeEx)
6. HRBtreeEx.*RBNodeHashing*(*RBN*)
7. return true
8. **else if** *RBN.left* ≠ *nil* and *RBN.right* = *nil* **then**
9. RBTreeHashing(RBN.left, HRBtreeEx)
10. HRBtreeEx.*RBNodeHashing*(*RBN*)
11. return true
12. **else**
13. RBTreeHashing(RBN.left, HRBtreeEx)
14. RBTreeHashing(RBN.right, HRBtreeEx)
15. HRBtreeEx.*RBNodeHashing*(*RBN*)
16. return true

Fig. 9. RB-tree node hashing and RB-tree hashing.

The RB-tree in the HRB-tree continues to be hashed after H-tree hashing. Different from the MH-tree in particular, each RB-tree node (RBN) maintains a sidelink which points to the first created N labeled with the key of RBN. When a second N with the same label is created, it will be linked by the sidelink inside the first linked N. In short, the Ns carrying the same label must be linked consecutively. The hashing method of RBN executes as depicted in **Fig. 9** (Algorithm 3). If it is a leaf node, the hash value of its child ($h_{RBN.child}$) is taken as H(0) (line 2). For an intermediate node, however, $h_{RBN.child}$ is obtained by hashing the concatenation of the hash values inside its left and right children (line 4). Specifically, the hash values of all linked $N(h_{side[n]})$ are utilized to compute $h_{RBN.side}$ (line 5), which guarantees the correctness of the information stored in the H-tree. Finally, the hash value of RBN (h_{RBN}) is returned (line 6). Similarly, based on the function of RBNodeHashing(RBN), the RB-tree can be hashed by running Algorithm 4 (**Fig. 9**). The DO signs the hash value of the root in the RB-tree (using the private key) and sends SigRBroot to the SP.

3.2 Authentication Scheme

As discussed in Section 2.2, the essential aspect of query authentication in the SP side is to construct a VO during query answering, and to enable the client to verify the correctness of the result by reconstructing a partial ADS utilizing the additional VO. In this section, we will describe the VO construction approach in the SP side, relying on the HRB-tree and the verification steps in the client side within the scheme of MDAHRB. Given a point query Q that relies on popular path cuboids in the stream cube, it will be processed according to the following four steps: 1. Access the indexed head table and get the node RBNi, the label of

which corresponds to the one-dimensional condition of Q, 2. Obtain all linked Ns by retrieving along the sidelink inside RBNi and Ns. Note that the other dimensional requirements of Q are restricted by the upper layers of N, 3. Examine whether the obtained N relates to Q by matching the labels of its upper ancestor nodes with the other dimensional conditions of Q, 4. Aggregate the attribute values of Q-related nodes and return it as the query results.

Thus, OLSA queries are answered efficiently by utilizing the pre-materialized popular path of the stream cube. Since the data structure of the stream cube (HRB-tree) is retained in main memory, continuous processing and operations is possible for answering OLSA queries. Hash operations on ADS are query driven, and the SP will hash the entire HRB-tree before the VO construction and query processing steps are forwarded. For collaborated HRB-tree hashing, following H-tree hashing (Algorithm 2), the RB-tree is hashed on the basis of the hash values of H-tree nodes by executing Algorithm 4.

Algorithm 5 ConPointQVO(Q, HRBtreeEx)
Require: Q – A point query, HRBtreeEx – An HRB-tree
Ensure: VO – The verification object
1. let pTemp point to a RBN that corresponds to Q
2. *pTemp* <= *HRBtreeEx.Rbtree.retrieve*()
3. *VO.h*$_{child}$ <= *H*(*pTemp.h*$_{lchild}$|*pTemp.h*$_{rchild}$)
4. *pTempN* <= *pTemp* -> *pSidelink*
5. while *pTempN* ≠ *nil* do
6. if the *N* that *pTempN* points to is Q-related then
7. Append relevant attribute values to *VO.side*$_{rValue}$
8. Append relevant h_x to *VO.side*$_{rHash}$
9. *pTempN* <= *pTempN* -> *pSidelink*
10. else
11. Append h_N to *VO.side*$_{urHash}$
12. *pTempN* <= *pTempN* -> *pSidelink*
13. for all *siblings* of traversed RBN in step 2 do
14. Append h_{RBN} to *VO.h*$_{sib}$
15. return VO

Algorithm 6 HRBVerification(VO, Sig_{RBroot})
Require: VO – The veri¯cation object, Sig_{RBroot} – The signature signed by the DO
Ensure: *bool* value – *true*(sound) or *false*(modified)
1. let $h_{side[\Lambda]}$ be the hash value on Q-related *N*,
2. let h_x be the temp hash values
3. for all *N* in *VO.side* do
4. $h_{side[\Lambda]}$ <= *H*(*H*(*VO.side*$_{rValue[\Lambda]}$)|*VO.side*$_{rHash[\Lambda]}$)
5. for all $h_{side[\Lambda]}$ and *VO.side*$_{urHash[\Lambda]}$ do
6. h_1 <= $h_{side[1]}$|…|$h_{side[\Lambda]}$
7. h_2 <= *VO.side*$_{urHash[1]}$|…|*VO.side*$_{urHash[\Lambda]}$
8. $h_{RBN.side}$ <= *H*(h_1|h_2)
9. h_{RBN} <= *H*($h_{RBN.side}$|*VO.h*$_{child}$)
10. h_3 <= *H*(h_{RBN}|*VO.h*$_{sib[0]}$)
11. for all *k* such that 1 ≤ *k* ≤ |*VO.h*$_{sib}$| do
12. h_4 <= h_3
13. h_3 <= *H*(h_4|*VO.h*$_{sib[k]}$)
14. *Match*(h_3, Sig_{RBroot})

Fig. 10. HRB-tree based verification.

Then, paralleling with query processing, VO is constructed. **Fig. 10** (Algorithm 5) illustrates the initial VO generation for a point query Q within an HRB-tree instance HRBtreeEx. It starts by locating one of Q's dimensional conditions specified by the client in the indexed head table, RB-tree (lines 1-2), and the RBN_k is returned. The hash values of its children are concatenated and hashed for the calculation of RBN_k's h_{child}, which is appended into the VO entity tagged by VO.h_{child} (line 3). For all the linked Ns along the sidelink of RBNk, the hash values of unrelated Ns (h_N) are directly appended to the VO as the object of VO.$side_{urHash}$ (line 11).

On the other hand, there are several time queues and many Qnodes inside each Q relevant N (**Fig. 7**). The attributes' values of related Qnodes are appended into VO as the object of VO.$side_{rV\ alue}$ (line 7), while the hash values of unrelated queues (e.g. h_H, h_Q, h_M) and Qnodes (h_{Qnode}) are inserted into the object of VO.$side_{rHash}$ (line 8). Finally, similar to the VO construction scheme of the MH-tree, the hash values of all siblings traversed (line 2) in the RB-tree (h_{RBN}) are appended to the substructure as VO.h_{sib} (line 14). Obtaining the query result, Res, additional authentication information (VO) and the signature of DO (Sig_{RBroot}), the client is enabled to verify the soundness of the returned information from the SP using HRBVerification(VO, Sig_{RBroot}). As shown in **Fig. 10** (Algorithm 6), a partial ADS' is first reconstructed (lines 3-13) and the hash value h'_{RBroot} (the final h_3 in the algorithm) at the root of ADS' is computed (line 13). By matching h'_{RBroot} against the decrypted hash value of Sig_{RBroot}, the soundness of Res is guaranteed. Specifically, lines 3-9 depict the computation of h_{RBN}.

MADHRB guarantees the soundness of query authentication. Suppose that the attribute values inside N or query results are modified. Because H(x) is a one-way, collision-resistant function, the $h_{RBN.side}$ computed by the client is different from that of the DO.

Therefore, although the structure of reconstructed ADS' is the same as ADS in the DO side, the computed h'_{RBroot} differs from the original, which implies the failure of verification.

3.3 Cost Model

Consistent with the state-of-the-art cost metrics discussed in Section 2.2, we will present the analytical cost models of MDAHRB. Each cost model of MDAHRB should consist of two parts, the H- and RB-trees.

Node fanout and tree height: The node fanout of the RB-tree is no more than 2, which implies $f_{RBN} \leqslant 2$. And the #RBN (φ) equals the #head table nodes. Then the height of the RB-tree $d_{RB} \leqslant 2\lg(\varphi+1)$.

In the case of the H-tree, however, there are differences between layers. The # and the order of layers is decided by the # and the top-down appearance order of different dimension attributes along the pre-established popular path. The fanout of each layer depends on the cardinality of the following layer. For instance, as depicted in **Fig. 4**, there are five distinct dimension attributes along the popular path, appearing top-down in the order of C_2, A_1, B_2, C_1, B_1. Then there are 5 layers in the H-tree (the root node is at level 0), and the node fanout of layer 2 (A_1) equals the cardinality of layer 3 (B_2). Here, let τ represent the # of different dimension attributes, and θ_i depicts the cardinality of the ith layer. The $d_H = \tau$, and the fanout of N in the ith layer $f_{iN} = \theta_{i+1}$, $0 \leqslant i \leqslant \tau - 1$.

Storage cost: The total size of the HRB-tree is equal to: $C_{sHRB} = C_{sH} + C_{sRB}$. It is easy to find that φ and the cost of the RB-tree (C_{sRB}) remains the same when Rs are incoming. Conversely, the #N (γ) will change with continuously incoming Rs, which implies that C_{sH} will fluctuate, and the scale relies on the extensiveness of the incoming data. Until the H-tree is filled and any incoming R can find an existing path, C_{sH} turns out to be stable and we can get the final C_{sHRB}.

Similar with the advantage of the MB-tree, C_{sHRB} does not reflect the DO/SP communication cost. Only the aggregated values inside Ns are transmitted to the SP, together with the Sig_{RBroot}. Also, the SP needs to reconstruct the HRB-tree and perform the hash operations incrementally from the H- to RB-trees. Compared with the signature-based approaches referred to in Section 2.2, the small additional hash computation cost in the SP reduces the DO/SP communication cost substantially.

Construction cost (hashing): Because the HRB-tree is an in-memory structure, there are no disk I/O operations. The construction cost for building an HRB-tree depends on the hash function computations and consists of three parts: $C_{cHRB} = C_{cH} + C_{cRB}(+C_{Sig})$. Here $C_{cRB} = \varphi \cdot C_{H(RBN)}$, and it is not affected by the incoming Rs. Since C_{cH} is equal to $\gamma \cdot C_{H(N)}$, it will fluctuate when the H-tree is a partial tree. To sum up, the total cost for constructing the HRB-tree is given by $C_{cHRB} = \gamma \cdot C_{H(N)} + \varphi \cdot C_{H(RBN)}(+C_{Sig})$. Note that although both DO and SP maintain the same HRB-tree locally, there is no C_{Sig} at the SP side.

Other costs: Since there is no disk I/O, the costs of VO construction (for SP) and verification (for client) are rather low. For VO construction, the SP only needs to return relevant hash and attribute values which are retained in memory. For verification, the client needs to do several hash operations while reconstructing the partial HRB-tree, and decrypt Sig_{RBroot} using DO's public key. The cost at the client side is given as:

$$C_v = x \cdot C_{H(N)} + C_{H(RBN)} + y \cdot C_{H(h_{lchild}|h_{rchild})} + C'_{Sig}.$$

It is obvious that the in-memory processing feature of MDAHRB guarantees the feasibility of authenticating OLSA queries. Moreover, adopting an RB-tree to index the head table reduces the overhead of inserting Rs and query answering. Our experimental evaluation exhibits that this effect will scale up with the growing speed of incoming data streams. On the other hand, MDAHRB cannot support completeness verification by relying on the existing work.

4. Multi-dimensional Authentication Method - MDAHB

Since there is no detailed previous work, we propose MDAHB as a competitor of MDAHRB. Section 4.1 presents the HB-tree utilized by MDAHB. Section 4.2 introduces the authentication scheme and gives relevant algorithms. Finally, the cost models of MDAHB are discussed in Section 4.3.

4.1 HB-tree

Similar with the HRB-tree adopted in MDAHRB, we propose to index the head table using a B-tree, as shown in **Fig. 11**. It is named an HB-tree, which can be utilized for authenticating OLSA queries. The HB-tree consists of the H- and B-trees. The H-tree is the same structure shown in **Fig. 4**, while the B-tree indexed head table has its own features compared with the RB-tree in the HRB-tree. For example, the height of the B-tree is lower than that of the RB-tree, while some memory space is wasted inside the B-tree's nodes. After all, with the same intuitions that are behind the HRB-tree, the HB-tree can also reduce the construction cost and support hash operations.

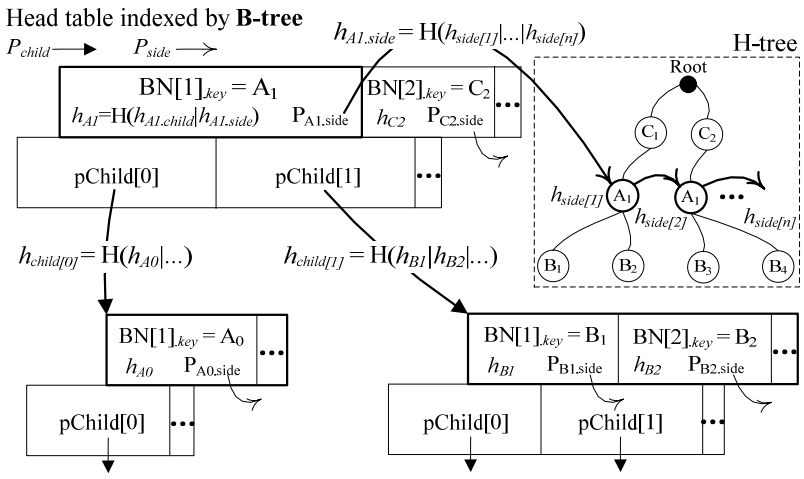


Fig. 11. HB-tree for OLSA authentication..

The MB-tree is the state-of-the-art ADS for index-based query authentication, and the HB-tree differs from the MB-tree in four aspects: 1. An MB-tree is proposed for authenticating records that are retained in disk, whereas the HB-tree is expected to handle in-memory processing, 2. The motivation of embedding the B-tree in the HB-tree distinguishes from the idea of adopting the B+-tree for the MB-tree. All the records in the B+-tree are only retained at the leaf nodes, while each B-tree node (BN) maintains its own records locally, a feature which

satisfies the requirement of indexing head table nodes, 3. The HB-tree is more complex than the MB-tree and expected to be more effective because it can aggregate records at high levels and dimensions, 4. The hash operation on BN inside the HB-tree correlates with a sidelink value which does not exist in any of the MB-tree nodes.

To sum up, all these distinguishing characteristics are due to the different motivations and application environments. As with the HRB-tree, the H-tree in the HB-tree is built dynamically, and the B-tree is constructed before any incoming Rs, by using head table nodes. The hash operations on H-tree nodes are organized by executing Algorithm 2, as shown in **Fig. 7**. Each H-tree node N maintains a hash value of h_N, which conforms to the basis of hashing on B-tree node BN. The hashing steps inside BN are summarized in **Fig. 12** (Algorithm 7).

```
Algorithm 7 BNodeHashing(BN)
Require: BN – A node in B–tree indexed head table
Ensure: h_BN[1] – Hash value in node BN
1. let count be #key, pChild[k] point to BN's children
2. for all pChild[k] such that 0 ≤ k ≤ count do
3.         if pChild[k] = nil then
4.                 h_child[k] <= H(0)
5.         else
6.                 h_child[k] <= H(h_child[k,0]|…|h_child[k,n])
7. for all BN[i] such that 1 ≤ i ≤ count do
8.         h_BN[i].side <= H(h_side[1]|…|h_side[n])
9.         h_BN[i] <= H(h_BN[i].side|h_child[i-1]|h_child[i])
```

```
Algorithm 8 BTreeHashing(BN, HBtreeEx)
Require: BN – A node in B–tree indexed head table,
HB–treeEx – An HB–tree
Ensure: HBtreeEx.Btree is hashed
1. if all BN.pChild[k] = nil (0 ≤ k ≤ count) then
2.    HBtreeEx.BNodeHashing(BN)
3.    return true
4. else
5.    for i = 0 to count do
6.    if BN.pChild[i] ≠ nil then
7.       BTreeHashing(BN.pChild[i], HBtreeEx)
8.    HBtreeEx.BNodeHashing(BN)
9.    return true
```

Fig. 12. B-tree node hashing and B-tree hashing algorithms

Given a node BN, we first compute the hash value of its children $h_{child[k]}$, where k should be no more than #Key+1. If there is no child, represented by pChild[k] = nil (line 3), we set the hash value as H(0) (line 4). Otherwise, the hash value is obtained by hashing the concatenation of all the keys' hash values inside the child (line 6). Specifically, each key inside BN is associated with a sidelink which links all the same-labeled Ns together, by which each key's side hash value ($h_{BN[i].side}$) is computed (line 8) like calculating $h_{RBN:side}$ in the HRB-tree. Then in particular, we can get the hash value of the *i*th key inside BN ($h_{BN[i]}$) by hashing the concatenation of its side hash value $h_{BN[i].side}$ and the hash values of its two adjacent children ($h_{child[i-1]}$, $h_{child[i]}$) (line 9). Thus, the entire HB-tree can be hashed by calling the function of BTreeHashing(BN, HBtreeEx) summarized in **Fig. 12** (Algorithm 8) recursively. The DO signs the hash value of the B-tree's root (h_{Broot}) using the private key, and sends the signature Sig_{Broot} to the SP. Taking into consideration the side hash values $h_{BN[i].side}$ makes the hashing process rather more complicated than the MB-tree. However, because all the operations are in-memory and the hash cost is cheap, the seemingly complicated processing provides a basis for HB-tree-based authentication, MDAHB.

4.2 Authentication Scheme

In this section, we will introduce the authentication scheme, MDAHB, based on the HB-tree. It is similar to the steps of MDAHRB, what distinguishes it is the B-tree indexed head table. Like other index based authentication approaches, MDAHB contains two stages, VO construction in the SP side and verification by the client. Overall, it is query-driven. Supposing a OLSA query Q is detected, the SP constructs the VO by executing Algorithm 9 in **Fig. 13**. After hashing the entire HB-tree, the Q-related key is retrieved in the B-tree indexed head table (line 2), and represented by the *k*th key contained in BN (BN[k]). First of all, the concatenation value of its adjacent children are inserted into the entity of VO, $VO.h_{BN[k].child}$ (line 3). Then, as

with MDAHRB, the related values of the side-hash and the attributes are appended into separate objects, $VO.side_{rValue}$, $VO.side_{rHash}$, $VO.side_{urHash}$ (lines 4-12). Since there are other keys inside a single BN, we need to append the hash values of BN[k]'s siblings ($h_{BN[i]}$, i != k) into $VO.h_{sib1}$ (line 14). Similarly, the hash values of all the traversed BN's siblings are inserted into $VO.h_{sib2}$ (line 16).

Algorithm 9 ConPointQVO(Q, HBtreeEx)
Require: Q – A point query, HBtreeEx – An HB-tree
Ensure: VO – The verification object
1. let pTemp point to a BN[k] that corresponds to Q
2. *pTemp* <= *HBtreeEx.Btree.retrieve*()
3. $VO.h_{BN[k].child} <= h_{child[k-1]} | h_{child[k]}$
4. *pTempN* <= *pTemp* -> *pSidelink*
5. **while** *pTempN* ≠ *nil* **do**
6. **if** the *N* that *pTempN* points to is Q-related **then**
7. Append relevant attribute values to $VO.side_{rValue}$
8. Append relevant h_x to $VO.side_{rHash}$
9. *pTempN* <= *pTempN* -> *pSidelink*
10. **else**
11. Append h_N to $VO.side_{urHash}$
12. *pTempN* <= *pTempN* -> *pSidelink*
13. **for all** *siblings* of *BN*[*k*] inside BN **do**
14. Append $h_{BN[i]}$ to $VO.h_{sib1}$, $i \neq k$
15. **for all** *siblings* of traversed BNs in step 2 **do**
16. Append h_{BN} to $VO.h_{sib2}$
17. **return** VO

Algorithm 10 HBVerification(VO)
Require: VO – The verification object
Ensure: *bool* value – *true*(sound) or *false*(modified)
1. let $h_{side[i]}$ be the hash value on Q-related N_i,
2. let h_x be the temp hash values
3. **for all** N_i in *VO.side* **do**
4. $h_{side[i]} <= H(H(VO.side_{rValue[i]}) | VO.side_{rHash[i]})$
5. **for all** $h_{side[i]}$ and $VO.side_{urHash[i]}$ **do**
6. $h_1 <= h_{side[1]} | \ldots | h_{side[i]}$
7. $h_2 <= VO.side_{urHash[1]} | \ldots | VO.side_{urHash[i]}$
8. $h_{BN[i].side} <= H(h_1 | h_2)$
9. $h_{BN[i]} <= H(h_{BN[i].side} | VO.h_{BN[i].child})$
10. **for all** $VO.h_{sib1[i]}$ **do**
11. $h_{BN} <= H(h_{BN[i]} | VO.h_{sib1[0]} | \ldots | VO.h_{sib1[i]})$
12. $h_3 <= H(h_{BN} | VO.h_{sib2[0]})$
13. **for all** *k* such that $1 \leq k \leq |VO.h_{sib2}|$ **do**
14. $h_4 <= h_3$
15. $h_3 <= H(h_4 | VO.h_{sib2[k]})$
16. *Compare*(h_3, Sig_{Broot})

Fig. 13. HB-tree based verification.

By implementing Algorithm 10 in **Fig. 13**, which rebuilds a partial B-tree indexed head table and matches the computed h'_{Broot} against the DO's signature Sig_{Broot}, the client is able to verify the soundness of the query result. Since the process executes in a manner similar to the inverse version of Algorithm 9, here we omit the detailed description. Likewise, the soundness is guaranteed by MDAHB, though the head table index structure differs from MDAHRB. Due to the one-way and collision-resistant properties of the hash function applied in the index structure (HB-tree), any modification of the attributes' value inside N ultimately leads to a contradiction in the client's verification.

4.3 Cost Model

We will compare MDAHRB with MDAHB in the same aspects. Each cost model of MDAHB should consist of two parts, the H- and B-trees.

Node fanout and tree height: The node fanout and height of the H-tree is the same as MDAHRB as follows: $d_H = \tau$, $f_{iN} = \theta_{i+1}$, $0 \leqslant i \leqslant \tau\text{-}1$. Suppose the maximum #Key in each BN in the B-tree indexed head table is σ ($\sigma \geqslant 2$), which means that the node fanout of B-tree $f_{BN} = \sigma+1$, we may deduce that $d_B \leqslant \log_\sigma[(\varphi+1)/2]$. It is clear that the cost of the head table tree in the HB-tree is much lower than that in the HRB-tree, i.e. $d_B < d_{RB}$.

Storage cost: Similarly, the total cost of the HB-tree $C_{sHB} = C_{sH} + C_{sB}$. For C_{sH}, there is no difference between the HB- and HRB-trees. However, C_{sB} is definitely larger than C_{sRB} due to the features of the B- and Red-Black trees. This is because there is no redundance in the key space that exists in BN and is wasted in the RB-tree.

Construction cost (hashing): The HB-tree is also retained in memory. No disk I/O implies that hash computation is the dominant construction cost. Likewise, it contains three parts: $C_{cHB} = C_{cH} + C_{cB}(+ C_{Sig})$. Given φ head table nodes, there are mostly φ/σ BNs in the

HB-tree. Consequently, the cost of hashing the B-tree $C_{cB} \leqslant (\varphi/\sigma) \cdot C_{H(BN)}$, and it is not affected by S_R. Since C_{cH} is equal to $\gamma \cdot C_{H(N)}$, it will fluctuate until the H-tree is filled. For simplicity, the cost for constructing the HB-tree is given by $C_{cHB} \leqslant \gamma \cdot C_{H(N)} + (\varphi/\sigma) \cdot C_{H(BN)}$ ($+ C_{Sig}$). We will compare C_{cHB} and C_{cHRB} in Section 5.

Other costs: The cost models of VO construction and query verification at the SP and DO sides are similar to those of MDAHRB. Because all issues are handled inside memory, the costs are much lower than that of ADS construction. The verification cost model at the client side is given by: $C'_v = x \cdot C_{H(N)} + C_{H(BN)} + y \cdot C_{H(h_{BN[i].side}|h_{child[i-1]}|h_{child[i]})} + C'_{Sig}$.

5. Experimental Evaluation

We implemented MDAHRB and MDAHB on a P4 2.6GHz CPU with 2GBytes of RAM, using the Crypto++ library [9]. All methods are implemented using Microsoft Visual C++ 6.0. The performance study is based on the real data set of NGSIM U.S. 101 [3]. It is about traffic information such as flow, speed and density, which is obtained from actual detailed vehicle trajectories collected by Cambridge Systematics, Inc. under the framework shown in **Fig. 1**.

Table 2. System parameters

Parameter	Default	Range
Data size	100K	10K, 50K, **100K**, 200K, 500K
#Head table node	6K	2K, 4K, **6K**, 8K, 10K

We compare the construction cost of MDAHRB and MDAHB in detail. The HRB- and HB-trees are both constructed dynamically with the incoming Rs. We split the cost evaluation into two sub-sections. Section 5.1 analyzes the insertion cost without hashing the tree, while Section 5.2 focuses on the hashing cost when a query request signal is detected by the DO. The data size SR and #head table nodes are the two main factors that can affect the performance. We take them as the evaluation parameters and summarize their ranges and default values (per timestamp) in **Table 2**. For simplicity, we consider only the OLSA point query that can be handled directly by drilling along a popular path. Since there is no obvious overhead gap between these two schemes during the VO construction and query verification stages (both of which take about 10-18 ms), we do not cover those experimental evaluations in this paper.

5.1 Insertion Cost

First, we inllustrate the effect of the data size SR, after setting #head table nodes to the default value (6K).

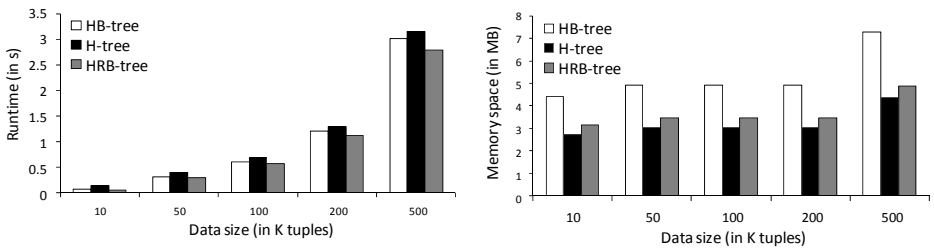


Fig. 14. Construction cost vs. Data size.

Fig. 14 shows the insertion cost per timestamp at the DO. Ideally, the runtime and memeory space overhead of all trees should increase linearly, because the dynamic construction is data-driven. But the memory space cost trend is not clear. This is due to the extensiveness of incoming data that may affect the size of the H-tree. This means that some later Rs can find the existing paths and new Ns are seldom built. Note that the gap between the HRB- and HB-trees increases with increasing data size, especially for runtime overhead. This is because of the search efficiency of the B- and RB-trees.

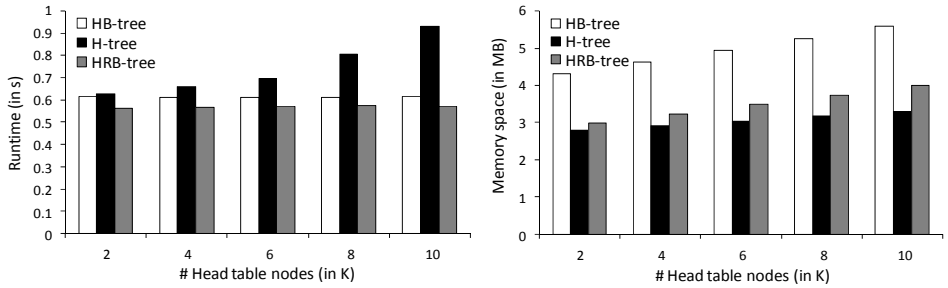


Fig. 15. Construction cost vs. #Head table nodes.

Setting the data size to the default value (100K), **Fig. 15** depicts the insertion cost against #head table nodes, changes in which affect the size and search efficiency of the head table. Among three approaches, increasing the memory space is stable. Because of the additional cost of indexing, the HB- and HRB-trees take more space than the H-tree. However, the runtime overhead of the H-tree grows linearly, while the HRB- and HB-trees remain largely unaffected. This is due to the advantage of indexing the head table, and the HRB-tree performs best by utilizing this. Overall, the improved HRB- and HB-trees both perform better than the original H-tree. And the HRB-tree exhibits lower insertion cost than the HB-tree.

5.2 Hashing Cost

In this section, we compare the hashing cost on the HRB- and HB-trees versus the data size and #head table nodes. For each experiment, we demonstrate the cost from the perspectives of runtime overhead and memory space usage. Specifically, the cost of the HRB- and HB-trees consists of two parts, the H-tree and the indexed head table. We demonstrate it in detail through **Fig. 16**, **Fig. 17** and **Fig. 18**.

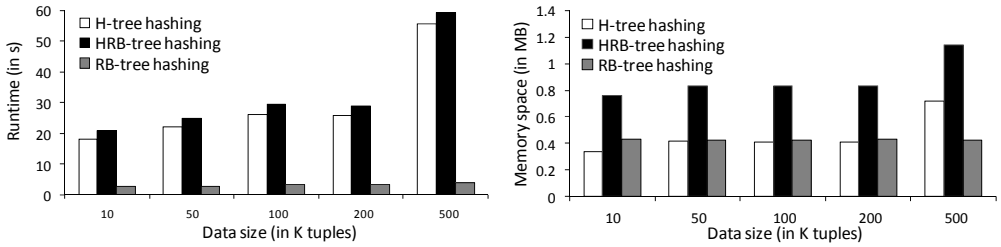


Fig. 16. HRB-tree hashing cost vs. Data size.

Fig. 16 exhibits the hashing cost on the HRB-tree against data size. It is obvious that the increasing hashing cost is due to the dynamic construction of the H-tree, driven by the continuous data streams.

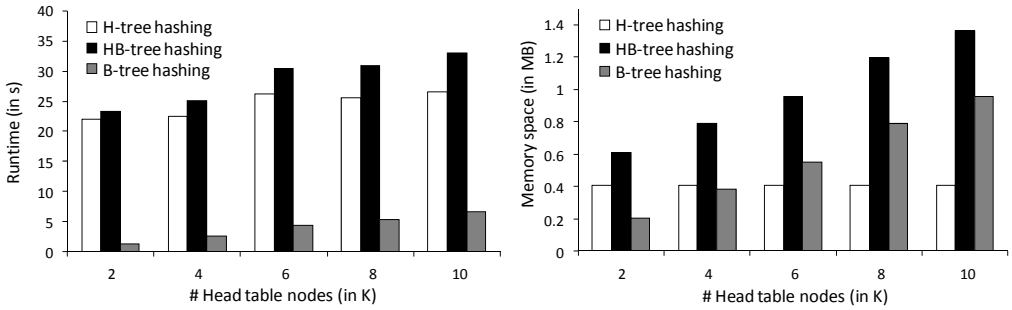


Fig. 17. HRB-tree hashing cost vs. #Head table nodes.

Conversely, **Fig. 17** shows that when #head table nodes increases, the hashing cost on the indexed head table is dominant and increases linearly. Likewise, considering the HB-tree, doing the same experiment yields consistent conclusions.

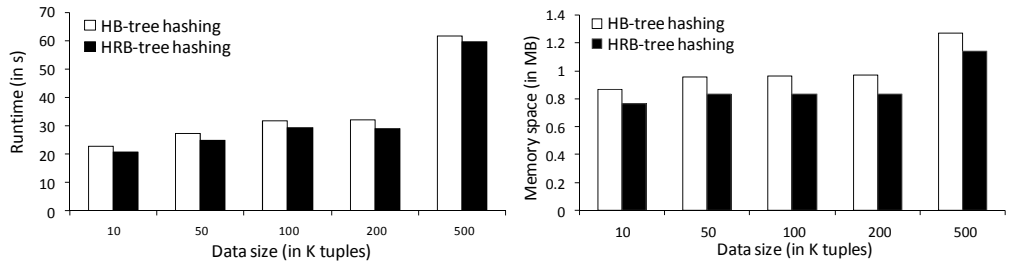


Fig. 18. Hashing cost vs. Data size.

To sum up, we compare the hashing cost between the HRB- and HB-trees as follows. **Fig. 18** summarizes the hashing cost verus data size. In this case, we have concluded that the cost increase is mainly due to the construction method of the H-tree. Because it is embedded inside both the HRB- and HB-trees, the overhead gap is stable and it is the result of the different properties of the two indexing schemes. Overall, the HRB-tree performs better than the HB-tree. And when the H-tree is constructed fully, there will be no increase against data size.

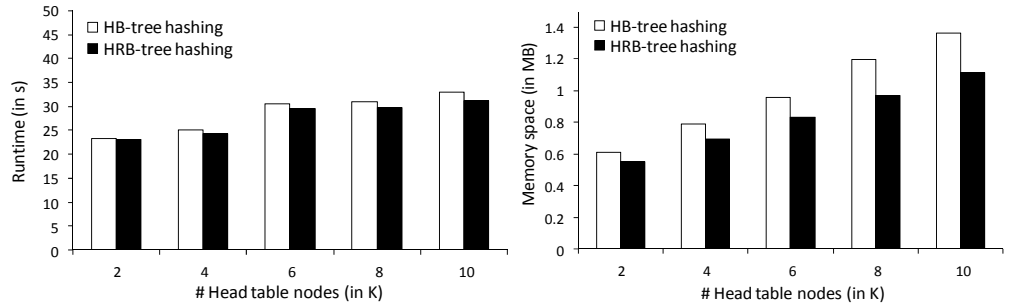


Fig. 19. Hashing cost vs. #Head table.

Fig. 19 assesses the effect of #head table nodes for these two schemes. Since both the B-tree and the Red-Black tree are two classical indices with reliable searching effciency, the runtime gap between the HB- and HRB-trees is not greatly affected by increasing #head table nodes.

However, the memory space gap becomes increasingly wider because of the wasted key space in the B-tree indexed in the HB-tree. Also, the gap will always grow if we increase the #head table nodes, even if the H-tree has already been filled. In this sense, the HRB-tree is more scalable and effcient than the HB-tree in the case of very large #head table nodes. All in all, the HRB-tree-based authentication method, MDAHRB, outperforms the HB-tree-based method, MDAHB. From the constructing point of view, Section 5.1 confirms that the HRB-tree exhibits lower insertion cost than the HB-tree. Also, Section 5.2 affims that the HRB-tree is more scalable and efficient than the HB-tree.

6. Conclusions

Aiming at covering the gap between OLSA and query authentication, this paper proposed two index-based authentication schemes, MDAHRB and MDAHB, which are based on the general approach for OLSA, the stream cube. Through the performance evaluation, both schemes are affirmed to be feasible. Moreover, the improved HRB- and HB-trees both perform better than the original H-tree for the stream cube. Considering the aspect of scalability, MDAHRB is more general than MDAHB. Future research work needs to support completeness verification, which both MDAHRB and MDAHB have not considered. And, the authentication schemes should be extended to support OLSA range queries, or even more complicated query types. For continuous data streams, it is impossible to retain all data in main memory. Consequently, the backup/mirror issues of the information on disk that least affects OLSA should be considerd. Also, research on how to execute a hybrid query which relies on both main memory and disk data, together with an appropriate authentication solution, is meaningful. Furthermore, whether the multi-dimensional analysis approach of the stream cube can be extended as a spatio-temporal cube to support spatio-temporal OLSA processing and outsourcing is an open problem.

References

[1] H. Hacigumus, B. Iyer, S. Mehrotra, “Providing database as a service,” in *Proc. of 18th Int Conf.on Data Engineering (ICDE)*, IEEE Computer Society, pp.29-38, 2002.
[2] S. Papadopoulos, Y. Yang, D. Papadias, “CADS: Continuous authentication on data streams,” in *Proc. of 33th Int Conf. on Very Large Data Bases (VLDB)*, pp.135-146, 2007.
[3] C. S. Inc., “NGSIM: Next generation simulation,” *http://ngsim.camsys.com*, 2005.
[4] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, Y. D. Cai, “Stream cube: An architecture for multi-dimensional analysis of data streams,” *Distributed and Parallel Databases*, vol.18, no.2, pp.173-197, 2005.
[5] M. J. Franklin, “From moore to metcalf: The network as the next database platform,” in *Proc. of 26th Int Workshop on Management of Data*, 2007.
[6] E. F. Codd, S. B. Codd, C. T. Salley, “Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate,” *Codd and Date. Inc*, 1993.
[7] W. Mao, “Modern Cryptography: Theory and Practice,” *Pearson Education*, 2003.
[8] F. Li, M. Hadjieleftheriou, G. Kollios, L. Reyzin, “Dynamic authenticated index structures for outsourced databases,” in *Proc of. 25th Int Conf. on Management of Data*, pp.121-132, 2006.
[9] Crypto++ library. http://www.eskimo.com/weidai/benchmark.html
[10] Openssl. http://www.openssl.org.
[11] “Fips pub 180-1: Secure hash standard,” *National Institute of Standards and Technology*, 1995.
[12] R. Merkle, “A certified digital signature,” in *Proc of. 9th Int Conf. on Advances in Cryptology (CRYPTO)*, pp.218-238, 1989.
[13] F. Li, K. Yi, M. Hadjieleftheriou, G. Kollios, “Proof-infused streams: Enabling authentication of

sliding window queries on streams," in *Proc of. 33th Int Conf. on Very Large Data Bases (VLDB),* pp.147-158, 2007.
[14] Y. Yang, S. Papadopoulos, D. Papadias, G. Kollios, "Authenticated indexing for outsourced spatial databases," *International Journal on Very Large Databases*, vol.18, no.3, pp.631-648, 2009.
[15] Y. Yang, D. Papadias, S. Papadopoulos, P. Kalnis, "Authenticated join processing in outsourced databases," in *Proc of. 28th Int Conf. on Management of Data*, pp.5-18, 2009.
[16] H. Gonzalez, J. Han, X. Li, "Flowcube: Constructuing RFID flowcubes for multi-dimensional analysis of commodity flows," *in Proc of. 32th Int Conf. on Very Large Data Bases (VLDB)*, pp.834-845, 2006.
[17] E. Lo, B. Kao, W. S. Ho, S. D. Lee, C. K. Chui, D. W. Cheung, "OLAP on sequence data," in *Proc of. 27th Int Conf. on Management of Data*, pp.649-660, 2008.
[18] J. Han, J. Pei, G. Dong, K. Wang, "Efficient computation of iceberg cubes with complex measures," in *Proc of. 20th Int ACM SIGMOD Conf. on Management of Data*, pp.1-12, 2001.
[19] Y. Chen, G. Dong, J. Han, B. W. Wah, J. Wang, "Multi-dimensional regression analysis of time-series data streams," in *Proc of. 28th Int Conf. on Very Large Data Bases (VLDB)*, pp.323-334, 2002.
[20] Y. D. Cai, D. Clutter, G. Pape, J. Han, M. Welge, L. Auvil, "MAIDS: Mining alarming incidents from data streams," in *Proc of. 23 th Int ACM SIGMOD Conf. on Management of Data,* pp.919-920, 2004.
[21] M. Cho, J. Pei, K. Wang, "Answering ad hoc aggregate queries from data streams using prefix aggregate trees," *Knowl. Inf. Syst.,* vol.12, no.3, pp.301-329, 2007.
[22] S. Nath, H. Yu, H. Chan, "Secure outsourced aggregation via one-way chains," in *Proc of. 28 th Int ACM SIGMOD Conf. on Management of Data*, pp.31-44, 2009.

Xiangrui Chen is currently a master's degree candidate at the Database Laboratory, Inha University, Incheon, Korea. He received a B.S. degree from Chongqing University of Posts and Telecommunications, China, in 2008. His research interests include data mining, data outsourcing, cloud computing, ubiquitous computing and information systems. He has received a full scholarship from the Institute of Information Technology Assessment (IITA) in Korea.

Gyoung Bae Kim is an Associate Professor at Seowon University, Korea Republic. He received a B.S. degree from Inha University, Korea Republic, in 1992, an M.S. degree from Inha University, Korea Republic, in 1994. and a Ph. D in Computer Science and Engineering from Inha University, Korea Republic, in 2000. He worked as a senior researcher from 2000 to 2004 and he worked as an assistance professor at Seowon University from 2004 to 2009. Prof. Kim's areas of interest include moving object databases, storage systems etc.

Hae Young Bae is a Professor at Inha University, Korea Republic. He received a B.S. degree from Inha University, Korea Republic, in 1974, an M.S. degree from Yonsei University, Korea Republic, in 1978, and a Ph. D in Computer Engineering from the Soongsil University, Korea Republic, in 1989. He worked as the dean of the Graduate School of Information Technology and Telecommunication at Inha University from 2004 to 2006, and he worked as the dean of the Graduate School at Inha University from 2006 to 2009. Prof. Bae's areas of interest include spatial databases, multimedia databases etc.