MMB^{cloud}-tree: Authenticated Index for Verifiable Cloud Service Selection

Jingwei Li, Anna Squicciarini, Dan Lin, Smitha Sundareswaran, and Chunfu Jia

Abstract—Cloud brokers have been recently introduced as an additional computational layer to facilitate cloud selection and service management tasks for cloud consumers. However, existing brokerage schemes on cloud service selection typically assume that brokers are completely trusted, and do not provide any guarantee over the correctness of the service recommendations. It is then possible for a compromised or dishonest broker to easily take advantage of the limited capabilities of the clients and provide incorrect or incomplete responses. To address this problem, we propose an innovative Cloud Service Selection Verification (CSSV) scheme and index structures (MMB^{cloud}-tree) to enable cloud clients to detect misbehavior of the cloud brokers during the service selection process. We demonstrate correctness and efficiency of our approaches both theoretically and empirically.

Index Terms—Cloud service selection, brokerage system, Merkle hash tree, verification

1 Introduction

Cloud services offer a scalable variety of storage space and computing capabilities, which are widely employed by an increasing number of business owners. This has resulted in a large number of cloud service providers (CSPs), offering a wide range of resources. The availability of various, possibly complex options, however, makes it difficult for potential cloud clients to weigh and decide which options suit their requirements the best. The challenges are twofold: 1) It is hard for cloud clients to gather information about all the CSPs available for their selections; 2) It is also computationally expensive to choose a suitable CSP from a potentially large CSP pool.

In light of these difficulties, both industry and academia (see [1] for a survey) suggested introducing an additional computing layer (referred to as cloud brokerage systems) on top of the base service provisioning to enable tasks such as discovery, mediation and monitoring. In a cloud brokerage system, one of the most fundamental tasks is to provide high-quality selection services for clients. That is, a broker provides clients with a list of recommended CSPs that meet the clients' needs. With the aid of cloud brokers, clients no longer need to collect, search or compare CSPs' services and capabilities.

The underlying assumption in the existing cloud brokerage schemes [2] [3] [4] [5] [6] [7] [8] is that brokers are completely trusted and thus will always provide unbiased best available options to clients [9]. Under this assumption, none of the existing works provides guarantees over the correctness or completeness of the service selection recommendations to the cloud clients. Without the ability to verify the correctness of the service recommendation, cloud clients could be easily cheated by malicious brokers. For instance, malicious brokers could recommend their favorable CSPs as much as possible and ignore other suitable CSPs, without being caught by the clients. More seriously, due to the lack of supervision and verification of brokers' actions, malicious brokers could even recommend malicious CSPs which collect and sell clients' private resources, monitor clients' hosts during cloud service provisioning, causing major financial and confidentiality losses to the clients. Therefore, it is important to equip the clients with verification capabilities of the obtained recommendations. The clients may not need to verify each recommendation result, but they certainly need to have the ability to do so when they feel necessary.

In this work, we propose innovative authenticated index structures and verification protocols *to allow clients to verify the completeness and authenticity of brokers' answers*. This problem is related to that of authentication of query results for outsourced databases [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20]. However, the characteristics of cloud service selection actually raises a new series of challenges. First, cloud service selection typically allows cloud users to specify multiple service requirements (i.e., multi-dimensional range queries), whereas many existing works on query authentication only support range queries on one or two dimensions (e.g., verifying location-based query results [16]). Second, it is always desirable to have efficient cloud service selection and verification so that the cloud end users would not feel delay of services, but existing few works [17] [18] [19], although support authentication of multi-dimensional query results, are time consuming, resulting that they could not meet the demands of today's real-time cloud service recommendations.

In order to overcome the limitations of existing techniques, both in terms of efficiency and supported functionality, we propose a new authenticated index structure, called Multi-Merkle B^{cloud}-tree (MMB^{cloud}-tree), which is a variant of the Merkle B$^+$-tree and is specifically tailored for cloud

- *Jingwei Li and Chunfu Jia are with the College of Computer and Control Engineering, Nankai University, P.R. China, e-mail: lijw1987@gmail.com, cfjia@nankai.edu.cn.*
- *Anna Squicciarini and Smitha Sundareswaran are with the College of Information Sciences and Technology, Pennsylvania State University, USA, e-mail: {asquicciarini, ssundareswaran}@ist.psu.edu.*
- *Dan Lin is with the Department of Computer Science, Missouri University of Science and Technology, USA, e-mail: lindan@mst.edu (corresponding author).*

service selections. In particular, we first design the Merkle B^{cloud}-tree (MB^{cloud}-tree) which is an authenticated index on the most common property (i.e., Price) of CSPs, and propose the corresponding verification protocol. Then, we extend the MB^{cloud}-tree to the MMB^{cloud}-tree by integrating a multi-dimensional indexing method (i.e., iDistance [21]) with MB^{cloud}-tree, to further improve the selection quality as well as reduce the verification burden at the client side. Our approaches are proved to ensure authenticity, satisfiability and completeness of the selected results. We have also experimentally compared our approaches with the most recent related work [18], and the results demonstrate significant improvements over the state-of-the-art [18].

Our novel index structure is the core component of our *Cloud Service Selection Verification (CSSV) scheme*, which employs the idea of "separation of duties" to ensure strong security guarantees. Precisely, we introduce a trusted collector in the cloud brokerage system that separates the task of CSP information collection from the service selection. The collector does not directly interact with the cloud clients and is only in charge of gathering information from the CSPs, and hence it can be more devoted into adopting sophisticated defenses to filter out problematic data and building an authenticated database of CSPs' profiles. The collector is allowed to make profit by selling the authenticated database to one or more cloud brokers. With the available authenticated databases, the cloud brokers focus on handling probably a large number of real-time service requests from clients.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 presents the problem statement. Section 4 gives an overview of the proposed CSSV system, followed by detailed verification schemes in Section 5 and 6. Section 7 reports the experimental results. Section 8 describes the practical consideration on saving clients' computing resources in several times of verification. Finally, Section 9 concludes the paper.

2 Related Works

Since our work is related to both cloud service selections and query authentication in databases, we review the works in these two areas in the following subsections, respectively.

2.1 Cloud Service Selection

There have been several efforts on cloud service selection in the literature. A general cloud service selection framework was proposed by Goscinski and Brock [26], which includes cloud service publication, discovery and selection. Han et al. [27] described a recommendation system in cloud computing suitable for design-time decisions as it statically provided a ranking of available cloud providers. Li et al. [2] developed systematic comparator CloudCmp to help customers choose a cloud that meets their needs through measuring and comparing the elastic computing, persistent storage and networking services. Aiming at evaluating the performance and capabilities of services offered by CSPs for facilitating customers' selections, Binnig et al. [3] designed a new benchmark to fit the characteristics (e.g., scalability, pay-per-use and fault-tolerance) of cloud computing. Lenk et al. [4] further proposed a new performance measuring method for Infrastructure-as-Service offerings, taking into account the type of services running in a virtual machine.

In order to consider the subjective aspects of CSPs, Rehman et al. [5] presented a framework for monitoring cloud performance based on customers' feedback. Li and Wang [6] in addition proposed a probability method to evaluate the subjective trustworthiness of the service component as well as the whole composite service from a series of ratings given by customers. Recently, Qu et al. [7] proposed an approach for cloud service selection based on both the user feedback and cloud performance. Modeling approach has also been utilized for service selection. In [28] [29], the cloud service selection is modeled as a multi-criteria decision-making problem, and then solved by using an analytic hierarchy process. Sundareswaran et al. [9] have modeled the properties of cloud service providers in a multi-dimensional space and utilized a K-nearest neighbor search for service selection.

Unlike our work, to the best of our knowledge, existing works on cloud service selection are focused only on how to select the services that satisfy customers' requirements. None of them considers security issues involved in the service selection, and none of them provides verifiable schemes to prove the correctness and completeness of their service selection results as addressed in our work. In addition, there is one recent work proposed by Zhang et al. [30] which also involves the notion of the "trusted collector". However, the purpose and the way how the collector is used are totally different. They use the collector for securely generating and sharing location-based information, whereas we use the collector to achieve service verification in the cloud.

2.2 Database Query Authentication

Our proposed authenticated index structures are related to those developed for query authentication in outsourced databases which can be classified into two main categories: Hash-based approaches and Signature-based approaches, as shown in Table 1.

The hash-based approaches [17] [19] [10] [16] [22] [20] employ the Merkle hash tree [31] or its variants to index the search keys. During a query, the service provider (i.e., resembles the cloud broker in our scenario) traverses the tree to identify the query results, and then send the results and some associated hash values of necessary nodes as proof messages to the client. Based on the received proof information, the client reconstructs the path from the nodes containing the query results to the root of the tree in order to verify the correctness of the query results. Our work also belongs to this category. Compared with existing works that also have the Merkle B^+-tree as the base structure, our work is superior to them because they either do not support multi-dimensional queries [10] [11] [12] [13] [14] [15] or do not consider the efficiency in the design [19]. Specifically, [10] [11] [12] [13] [14] [15] are for 1-dimensional queries, and [16] are for two or three-dimensional queries in location-based services. With respect to the (few) works for multi-dimensional queries, we note some important limitations. The range tree-based method proposed in [17] needs to build and embed a Merkle hash tree for each node, and

Categories	Work	Low-Dimensional Range Query	Multi-Dimensional Range Query	Comparison with Our Approaches
Hash-based Approaches	Our Approaches	✓	✓	–
	Devanbu et al. [17]	✓	✓	produce large index affecting efficiency
	Pang et al. [19]	✓	✓	inefficient for queries on non-key attributes
	Li et al. [10]	✓	✗	only support 1-dimensional query
	Yang et al. [16]	✓	✗	only support low-dimensional query
	Papadopoulos et al. [22]	✓	✓	require an extra fully trusted entity
	Yang et al. [20]	✓	✓	require MapReduce framework
	Zheng et al. [15]	✓	✗	only support 1-dimensional query
Signature-based Approaches	Mykletun et al. [11]	✓	✗	only support 1-dimensional query
	Mykletun et al. [23]	✓	✗	only support 1-dimensional query
	Narasimha et al. [12]	✓	✗	only support 1-dimensional query
	Pang et al. [13]	✓	✗	only support 1-dimensional query
	Cheng et al. [18]	✓	✓	require a large number of signatures
	Pang et al. [14]	✓	✗	only support 1-dimensional query
Others	Bajaj et al. [24]	✓	✓	require trusted hardware
	Papadopoulos et al. [25]	✓	✓	require heavy cryptographic primitives

TABLE 1
Summary of Existing Works on Authenticating Range Queries in Outsourced Database

this process is also recursively invoked for the nodes of the embedded Merkle hash tree, which makes index contruction, querying and verification extremely time consuming; the VB-tree in [19] is not efficient for queries on non-key properties because it will generate large size proof messages to cover the nodes in-between the query ranges but do not contain the query results.

Signature-based approaches [11] [12] [14] [13] [18] [23] typically construct an authenticated and unforgeable chain over the data objects in a specified order. At query execution, the service provider picks the signatures of the data objects falling in the query range to form the proof messages. Since each data object is linked with its predecessor and successor in an unforgeable way, the client is able to verify the completeness and correctness of query results by verifying the validity of signatures. Among all the related works, [18] is the most recent work related to ours since it verifies multi-dimensional range queries under assumptions that are similar to ours. Therefore, we choose it for comparison in the experimental study.

In addition, it is worth noting that some recent works [20] [24] [22] [25] also support authentication of multi-dimensional queries. Compared with them [20] [24] [22] [25], our approaches do not rely on server-hosted, tamper-proof, trusted hardware, heavy cryptographic primitives (e.g., bilinear pairings in [25]).

2.3 The Merkle Hash Tree

As our proposed data structure is developed based on the Merkle hash tree, we provide more details of this structure as follows. The Merkle hash tree [31] has a binary tree as the base structure. The leaf nodes in the Merkle hash tree contain the hash values of the original data items. Each internal node contains the hash value of the concatenation of the hash values of its two children nodes. The hash value of the root of the tree is published for verification. If there is any change to the original data values, one would not be able to compute the same hash value of the root and hence detect such data tampering.

Fig. 1 illustrates an example of the data verification using the Merkle hash tree. A verifier has the published hash value h_{root} of the root. He sends the request to the prover for data item x_2. The prover returns the requested data item along with the auxiliary authentication information $h_1, h_{3||4}$ and $h_{56||78}$, where $||$ is the symbol of concatenation which will be used throughout this paper. Upon receiving the data item and the additional authentication information, the verifier performs the following computation to check if the received x_2 is authentic. First, the verifier computes $h_2 = \mathsf{Hash}(x_2), h_{1||2} = \mathsf{Hash}(h_1||h_2), h_{12||34} = \mathsf{Hash}(h_{1||2}||h_{3||4}), h_{1234||5678} = \mathsf{Hash}(h_{12||34}||h_{56||78})$, and then he checks if the calculated root hash (i.e., $h_{1234||5678}$) is the same as the published one (i.e., h_{root}). If so, the verifier will conclude that the data is authentic.

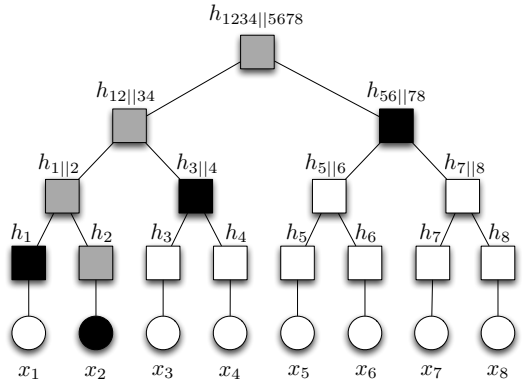


Fig. 1. Merkle Hash Tree

Based on the similar idea of the Merkle hash tree, the Merkle B$^+$-tree [32] [10] [15] have been proposed to handle multiple entries per node.

3 Problem Statement

Broker-based cloud service systems generally include three types of entities:

- **Cloud Clients** are end users who request certain types of cloud services;
- **Cloud Service Providers (CSPs)** provide a variety of cloud services;
- **Cloud Broker** is the middleman between the cloud clients and CSPs. For example, we consider the case when the broker provides service selection services (given in Definition 1) to the cloud clients.

In these settings, our main problem is how to ensure that cloud brokers recommend authentic and complete services available to clients, according to the clients' specific requests (namely *service selection queries*).

To specify CSPs' services which the clients can request to brokers, we consider ten common properties of CSPs [9]. A brief description of these properties is given in Table 2[1]. Accordingly, let us denote with $\mathcal{U} = \{\mathsf{SType}, \mathsf{QoS}, \mathsf{Security}, \mathsf{ISize}, \mathsf{OP}, \mathsf{Price}, \mathsf{Sub}\}$ the property universe for CSPs. Each property $U_i \in \mathcal{U}$ has a domain $\mathcal{D}_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}\}$ consisting of all the possible values of U_i, where n_i is the number of the possible values for property U_i. A service selection query over such properties is formally specified as follows.

Definition 1 (Service Selection Query). A cloud service selection query is specified according to the following form: $\mathcal{Q} = \{(U_1, q_1), (U_2, q_2), ..., (U_k, q_k)\}$ $(1 \leq k \leq 7)$, where $U_i \in \mathcal{U}$ and q_i $(1 \leq i \leq k)$ is a query range within the domain of the CSP property U_i.

Example 1. *Suppose that there are four CSPs' profiles stored at the cloud broker. For simplicity, we assume that these CSPs all provide storage services and we consider only three properties of each CSP as follows:*

- CSP_1: Price=*24;* Security=*high;* OP=*Windows.*
- CSP_2: Price=*29;* Security=*medium;* OP=*Windows.*
- CSP_3: Price=*18;* Security=*medium;* OP=*Unix.*
- CSP_4: Price=*15;* Security=*low;* OP=*Windows.*

If a client wants to find the CSPs which offer the storage service at the price no greater than 25 with at least medium level security guarantee (i.e., the security property has the value "medium"), the client can send the service selection query $\mathcal{Q} = \{(\mathsf{Price}, [0, 25]), (\mathsf{Security}, [medium, high])\}$ *to the cloud broker.*

Our problem, i.e. verification of cloud brokers recommendations, is compounded by the lack of trustworthiness assumptions against the broker. For instance, a broker may be *Lazy* or *Cheating*. The *Lazy* broker aims to save its computing resources, and hence returns random results to clients instead of actually processing service selection queries. The *Cheating* broker attempts to recommend its "bribery" CSPs as much as possible by intentionally dropping other better options for the cloud clients. Therefore, our original problem is mapped into that of designing an approach that enables clients to verify the correctness of the query results.

Definition 2 (Correctness of Query Results). Let $\mathcal{Q} = \{(U_1, q_1), (U_2, q_2), ..., (U_k, q_k)\}$ be a service selection query, and let $\mathcal{R}$ denote the set of CSP profiles in the query results returned by the cloud broker. We say $\mathcal{R}$ is correct with respect to $\mathcal{Q}$ if and only if $\mathcal{R}$ achieves all the following three guarantees:

- **Authenticity**: For any CSP's profile in $\mathcal{R}$, the profile must be authentic without any adversarial tampering by the broker.
- **Satisfiability**: We say $\mathcal{R}$ achieves satisfiability with respect to $\mathcal{Q}$, if any CSP $C_i = \{u_{1,i_1}, u_{2,i_2}..., u_{7,i_7}\}$ in $\mathcal{R}$ satisfies the query $\mathcal{Q}$, i.e., $u_{j,i_j} \in q_j$ holds for all the queried properties U_j in $\mathcal{Q}$.
- **Completeness**: We say $\mathcal{R}$ achieves completeness if there is no CSP $C_l = \{u_{1,1_l}, u_{2,2_l}$ $..., u_{7,7_l}\}$ which is not in $\mathcal{R}$ but satisfies the query $\mathcal{Q}$.

Example 2. *Reconsider Example 1. The correct query results that satisfy authenticity, satisfiability and completeness should be* $\mathcal{R} = \{CSP_1, CSP_3\}$. *If the cloud broker returns* $\mathcal{R} = \{CSP_1\}$, *the response is not complete, since another qualifying* CSP_3 *is missing. If the cloud broker returns* $\mathcal{R} = \{CSP_1, CSP_2, CSP_3\}$, *it violates satisfiability property, as* CSP_2 *does not satisfy the query.*

4 An Overview of the CSSV Scheme

As aforementioned, correct service selection results should guarantee authenticity, satisfiability and completeness. To verify authenticity of a CSP's profile, a naive solution is to require the CSP to sign its profile and then let the client verify the signature. Satisfiability is also easy to verify since the client just needs to check if the profiles of the candidate CSPs in the query results actually satisfy the query. However, there is not a trivial way for the client to know if the query results returned by the broker contain all qualifying CSPs. In other words, the verification of completeness is the most challenging issue. To address this, we propose the Cloud Service Selection Verification (CSSV) scheme which is a comprehensive solution that is capable of guaranteeing all the three security requirements (i.e., authenticity, satisfiability and completeness).

In the CSSV scheme, we introduce one more entity, *the collector*, besides CSPs, cloud clients and cloud brokers. The collector acts like a certificate authority and is assumed fully trusted, which is inline with the recent work [30]. The collector is associated with a pair of public and private keys, and its public key is made available to CSPs, cloud brokers and cloud clients. Specifically, the CSSV scheme (as shown in Fig. 2) includes the following three phases:

1) **Database Construction by the collector**: The collector is responsible for collecting the profiles (including the properties discussed in Section 3 and some other information such as user ratings, etc) of CSPs, and constructing an authenticated CSP profile database that ensures the integrity of the CSPs' information. The collector sells the authenticated CSP profile database to multiple cloud brokers.

1. The actual cost of a cloud service is determined by Pricing and three other properties: Measurement Units, Pricing Units, and Pricing Sensitivity. We conduct a preprocessing to recompute the value of the Pricing property by integrating the other three properties. After the preprocessing, the Price property will be under the same unit and measurement.

Property	Description
Service Type (STtype)	type of CSP
Security	level of security
Quality of Service (QoS)	quality of service determined by user ratings
Measurement Units	unit for charge
Pricing Units	how long a service is reserved for
Instance Size (ISize)	amount of resources used at a given instance
Operating System (OP)	operating system used
Pricing	actual price for the usage of CSP
Pricing Sensitivity	whether the price varies by region
Subcontractors (Sub)	whether subcontractors are presented

TABLE 2
Properties of Cloud Service Providers

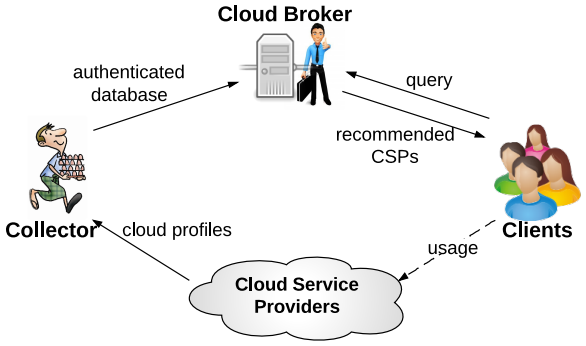


Fig. 2. Architecture for CSSV System

2) **Service Selection by the broker**: Each cloud broker handles a potentially large amount of online clients requests. For a service selection request from a consumer, the cloud broker will query the authenticated CSP profile database provided by the collector and recommend CSPs to the clients.
3) **Results Verification by the clients**: The clients execute result verification algorithms to verify the correctness of the query results returned by the broker.

In the next two sections, we will present our proposed authenticated indexes, the MB^{cloud}-tree and the MMB^{cloud}-tree, to be employed in the CSSV schemes. It is worth noting that, the novelty of our approaches not only lies in a new set of verification algorithms specific to the cloud service selection, but also gives efficient solutions (compared with the state-of-the-art) to the problem of authenticating multi-dimensional queries.

5 Basic Scheme using MB^{cloud}-tree

Our basic scheme follows the three phases of the CSSV system: the database construction, service selection and results verification. It is developed based on a variant of the Merkle hash tree [31].

5.1 Phase 1: Database Construction

In order to efficiently manage and retrieve CSPs' profiles while guaranteeing data integrity, we propose a MB^{cloud}-tree to index CSPs' profiles on the Price property of the CSPs. The MB^{cloud}-tree is a variant of the authenticated index structure, the Merkle B$^+$-tree [32] [10] [15]. The reason to choose Price as the indexing field is two-fold. First, given that most cloud providers employ a pay-per-use business model, Price is one of the most commonly occurred criteria in cloud service selection queries. Second, since there are many possible values of Price among CSPs, Price is a very selective property which makes queries more efficient.

Fig. 3 illustrates the structure of the MB^{cloud}-tree, which is developed based on the B$^+$-tree by adding one additional field to each entry to store a hash value for the subsequent authenticity check. The information stored in each tree component is described as follows:

- Each CSP is represented by a data structure $\mathcal{C}$ consisted of its unique identity (ID), a list of property values ($\mathcal{L}$) and a hash of its profile ($\mathsf{Hash}(prof)$), computed using a secure collusion free hash function.
- Each entry e_i in the leaf node stores the information of CSPs offering the same price in the form of $\langle key_i, h_i, pt_i \rangle$, where key_i is the index key and contains the value of the price, pt_i points to a list of CSPs whose price equals to key_i, and h_i is a concatenation of all the hash values of CSPs in the list. Suppose that CSP_1, CSP_2, ..., CSP_k fall into the same entry and hash values of their representation structures are $\mathsf{Hash}(\mathcal{C}_1), \mathsf{Hash}(\mathcal{C}_2) \ldots \mathsf{Hash}(\mathcal{C}_k)$, respectively. Then, the hash value of this entry will be $h_i = \mathsf{Hash}(\mathsf{Hash}(\mathcal{C}_1)||\mathsf{Hash}(\mathcal{C}_2)|| \ldots ||\mathsf{Hash}(\mathcal{C}_k))$. The leaf nodes are chained with sibling leaf nodes via pointers.
- Each internal node contains m index key values and $m+1$ pointers to the child nodes which are defined in the same way as that in the B^+-tree. Additionally, each pointer in the internal node is associated with one hash value that is computed by concatenating the hash values of entries in its child node. More specifically, let $h_{i,1}$, ... $h_{i,m}$ denote the hash values in a child node pointed by pt_i, the hash value h_i is computed as $h_i = \mathsf{Hash}(h_{i,1}||h_{i,2}, ..., ||h_{i,m})$. Notice that here we abuse notation by using the same symbol pt_i as the pointer in internal node pointing to a child node, and h_i as hash value associated with pt_i.

To construct the MB^{cloud}-tree, the information of each CSP is inserted in the same way as that in the B^+-tree. A hash value of the CSP is computed, and the hash value is

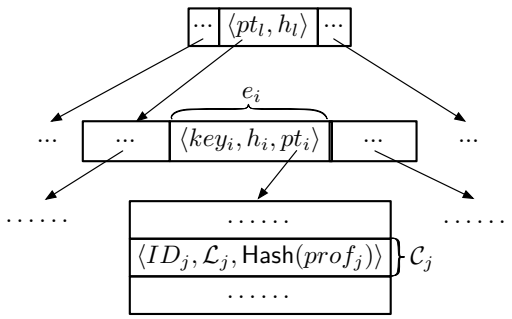


Fig. 3. Structure of Merkle B^{cloud}-tree

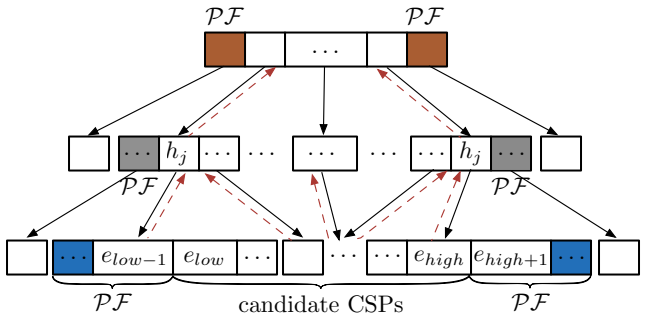


Fig. 4. Proof Message in Merkle B^{cloud}-tree

then used to compute the hash values of its ancestor nodes all the way up to the root.

After the construction, the collector signs the hash value of the root node using its private key, and publishes the obtained signature σ. Finally, the collector sends the MB^{cloud}-tree along with the profile database to the cloud brokers.

5.2 Phase 2: Service Selection

The cloud broker takes requests from clients and executes the service selection queries on the Merkle B^{cloud}-tree obtained from the collector. The service selection query algorithm consists of two main steps: 1) filtering and 2) refinement.

• **Step 1 (Filtering)**: The cloud broker identifies candidate CSPs and the hash values to be returned for verification. Specifically, upon receiving a query $\mathcal{Q} = \{(U_1, q_1), (U_2, q_2), ..., (U_k, q_k)\}$, the broker first checks if $\mathcal{Q}$ includes Price. If so, the broker will run a range query on the MB^{cloud}-tree using the query range (denoted as q_{price}) of Price. The range query algorithm is the same as that in the B$^+$-tree, which starts from the root and traverses down the tree following the pointers that point to the key values within the query range. Let e_{low} and e_{high} denote the leaf node entries which contain the minimum and maximum prices of CSPs that are within the query range, respectively. In order for the client to check completeness later, the broker also records the left neighbor entry e_{low-1} of e_{low}, and right neighbor entry e_{high+1} of e_{high} (if e_{low} or e_{high} is the leftmost or rightmost entry of MB^{cloud}-tree, then e_{low-1} or e_{high+1} would be set empty). These four entries together define the boundary of the query results. Note that if none of CSPs in the broker's database satisfies the query, e_{low-1} and e_{high+1} will be the rightmost and leftmost entries in the tree, and e_{low} and e_{high} would be empty.

Given the above four boundary entries, the cloud broker constructs a proof message ($\mathcal{PF}$) for verification as illustrated in Fig. 4. For each entry e_i between e_{low-1} and e_{high+1} (including e_{low-1} and e_{high+1}), the broker includes the representation structures of the CSPs stored in e_i in the proof. Here, e_{low-1} and e_{high+1} will be later used by the client to verify the completeness of the query results. Next, for each parent node of e_i, the broker includes the hash values of non-parent entries (the shaded boxes in Fig. 4) in the proof to facilitate the client to reconstruct the hash value of the parent node at the verification phase. The same process is applied to all the ancestor nodes of e_i all the way up to the root. In summary, the proof message $\mathcal{PF}$ contains representation structures of candidate CSPs, boundary CSPs (i.e., e_{low-1} and e_{high+1}), and a set of hash values of non-ancestor entries in the ancestor nodes.

• **Step 2 (Refinement)**: The broker further exams the properties of candidate CSPs to put those CSP profiles that satisfy all the requirements of the query in a query result set $\mathcal{R}$. Then, the broker returns $\mathcal{R}$ and the proof message $\mathcal{PF}$ to the client.

5.3 Phase 3: Results Verification

Upon receiving the results $\mathcal{R}$ and the proof message $\mathcal{PF}$, the client will verify the correctness of the results in terms of satisfiability, authenticity, and completeness. It is worth noting that this step can be optional. The client does not need to verify the results every time but, it could choose to check the correctness of $\mathcal{R}$ at random time. The cloud brokers would not know when the results will be checked and hence still need to follow the approach faithfully to protect their reputation.

To verify satisfiability of the result, the client simply double checks if the properties of CSPs in $\mathcal{R}$ meet his/her needs. Next, the client verifies the authenticity and completeness simultaneously by computing the hash value of the root using the information in the proof message. Specifically, starting from the leaf level, for entries with provided representation structures, the client computes their hash values. Then, the client concatenates both computed hash values and hash values included in $\mathcal{PF}$ to generate the hash values of parent entries. As illustrated with red dash arrows in Fig. 4, this process is repeated until the hash value of the root (denoted as h_{root}) is computed. Then, the client verifies if the published signature σ is a valid signature on h_{root}. If so, the results are authentic. Further, to ensure the completeness, the client will check the properties of boundary entries e_{low-1} and e_{high+1} to see if they are real boundaries, i.e., located outside the query range.

Finally, we prove that our approach allows the clients to verify the correctness of the results. That is, the client will detect the broker's misbehavior of any modification and deletion of qualifying results.

Theorem 1. For any query $\mathcal{Q} = \{(U_1, q_1), (U_2, q_2), ..., (U_k, q_k)\}$, let $\mathcal{R}$ denote the query results returned by the approach described above. The correctness of $\mathcal{R}$ (in Definition 2) can be verified by the client.

Proof 1. *We prove the theorem by contradiction. First, if there exists a $CSP_{wrong} \in \mathcal{R}$ whose properties do not satisfy $\mathcal{Q}$, CSP_{wrong} will be detected by the client during the satisfiability verification of Phase 3.*

Second, suppose that there exists a $CSP_i \in \mathcal{R}$ whose properties have been modified by the broker and are no longer authentic. Consequently, the hash value of CSP_i with modified properties will not match its original hash value, and hence resulting in a different hash value (denoted as $hash_{wroot}$) of the root of the Merkle B^{cloud}-tree. By verifying the validity of published signature σ on $hash_{wroot}$, the client will detect such misbehavior. Similarly, if the broker intentionally changed any information in the proof message, the change will lead to incorrect hash value of the root and hence be detected.

Third, suppose that the broker did not include one or more qualifying CSPs in the query results. This involves two cases. One is that the broker dropped several entries in the leaf node and still returned the other qualifying entries in the same node. In this case, the hash value computed from the remaining qualifying entries in $\mathcal{R}$ will not lead to correct hash value of the corresponding entry in parent node. The second case is that the broker dropped the whole leaf node with qualifying entries. The broker may or may not eventually return the hash value of the dropped leaf node in the missing leaf. In either case, the client will detect the broker's misbehavior since the client will not be able to compute the correct hash value of the root. This is because the hash value of the parent node is a concatenation of the hash values of entries in an ascending order of their prices. In other words, the hash values of entries in the query results are placed next to each other when the client concatenates them. The missing leaf node results in a missing hash value in the middle of the concatenation, which in turn affects the correctness of the hash value of the root.

6 Advanced Scheme using MMB^{cloud}-tree

The basic approach using MB^{cloud}-tree indexes only the Price property, and therefore has limited ability to deal with queries that do not include Price as one of the selection criteria, or with queries that have many other selection criteria besides Price. In either case, the basic approach may return many CSPs which satisfy only the Price criterion but not the whole query in the proof message for verification. Note that, the final query results are not affected since a refinement step is included to identify the actual qualifying CSPs. In order to balance the verification burden (i.e., reduce the number of false positives in filtering step), we propose an advanced scheme indexing multiple properties. The advanced scheme integrates the MB^{cloud}-tree idea with a multi-dimensional index (i.e., iDistance [33] [21]) to build a novel authenticated index, MMB^{cloud}-tree, on all the properties of the CSPs. Therefore, it is effective and efficient for queries that contain requirements on any property. In what follows, we first briefly review the iDistance index and then present the detailed algorithms of the advanced scheme.

6.1 iDistance

The iDistance [21] [33] is an indexing and query processing technique for K-nearest neighbor (KNN) queries on data point in multi-dimensional spaces. The key idea of iDistance is to map multi-dimensional points to a one-dimensional key value so that they can be indexed using the B$^+$-tree that is commonly available in commercial database systems. The iDistance index is built in two main steps:

1) Cluster the data points and select the cluster centers as reference points.
2) Compute the distance between a data point and its closest reference point, and use this distance plus a scaling value to form an index key for this data point. Then, index all the data points using the B$^+$-tree.

Compared with iDistance, our proposed MMB^{cloud}-tree handles range queries instead of KNN queries and supports authentication of the query results.

6.2 MMB^{cloud}-tree-based Scheme

The advanced scheme also follows the three phases of the CSSV system introduced in Section 4.

6.2.1 Phase 1: Database Construction

By leveraging MB^{cloud}-tree and iDistance, we propose a Multi-Merkle B^{cloud}-tree (MMB^{cloud}-tree for short) which treats each CSP as a multi-dimensional data point with each property being one dimension.

As a preprocessing step, we first encode each property of a CSP to a numerical value. Recall that each CSP is associated with 7 properties as described in Section 3. Let $CSP = \{(\mathsf{SType}, v_1), ..., (\mathsf{Price}, v_5), ...\}$, where v_i is the value of a property. For those numerical properties like Price and ISize, we normalize them to a value between 0 and 1 as follows:

$$v_i' = \frac{v_i - \mathsf{min}(\mathcal{D}_i)}{\mathsf{max}(\mathcal{D}_i) - \mathsf{min}(\mathcal{D}_i)}$$

where $\mathsf{max}(\mathcal{D}_i)$ and $\mathsf{min}(\mathcal{D}_i)$ are the maximum and minimum values in the domain of this property, respectively. For the remaining non-numerical properties like SType and Security, we map each non-numerical value to a numerical value and then normalize them in the same way as the numerical properties. An example is given below.

Example 3. *Reconsider the CSPs in Example 1. Their non-numerical properties are transformed as follows (the value after the hyphen denotes the numerical value after the mapping):*

- Security*: high-2, medium-1, low-0*
- OP*: Windows-1, Unix-0*

Then, a CSP can be represented as a 3-dimensional point (corresponding to three properties in the example):

$$CSP_1 : \{\frac{26-15}{29-15}, \frac{2}{3}, \frac{1}{2}\}; \quad CSP_2 : \{\frac{29-15}{29-15}, \frac{1}{3}, \frac{1}{2}\}$$
$$CSP_3 : \{\frac{18-15}{29-15}, \frac{1}{3}, \frac{0}{2}\}; \quad CSP_4 : \{\frac{15-15}{29-15}, \frac{0}{3}, \frac{1}{2}\}$$

Next, we propose a two-level clustering for the CSPs. First, CSPs of the same service type are grouped together. This is because most of time a service selection query targets a single type of service. Even if a query contains multiple service types, the query can be easily re-written to multiple queries with one service type per query. This first level clustering helps restrict the subsequent service selection to

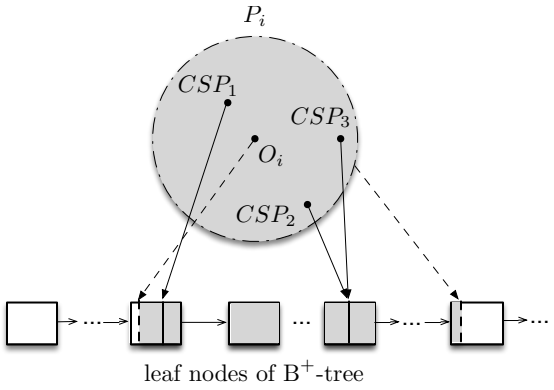


Fig. 5. An Example of Index Key Generation in 2-Dimension Space

the CSPs of the same service type and avoid checking the CSPs of non-matching service types.

Within each group of CSPs of the same service type, we further conduct a K-means clustering [34] using the Euclidean distance (defined in Equation 1) between each pair of CSPs for clustering. Note that v'_{il}, v'_{jl} $(l = 1, 2, \ldots, 7)$ are the normalized property values.

$$\mathsf{dist}(CSP_i, CSP_j) = \sqrt{(v'_{i1} - v'_{j1})^2 + \ldots + (v'_{i7} - v'_{j7})^2} \quad (1)$$

After the clustering, we obtain $n \times t$ clusters (or partitions), where n is the total number of service types and t is the tunable clustering parameter. Let O_1, O_2, ..., $O_{n\times t}$ denote the cluster centers of each partition. Then, for a CSP_j, we first identify the nearest O_i of CSP_j and then compute its index key using the following formula:

$$\mathsf{Key}(CSP_j) = S_1 \cdot (i-1) + S_2 \cdot \mathsf{dist}(O_i, CSP_j) \quad (2)$$

where S_1 is a scale factor that reserves a range of values for each partition and ensures that the index keys of data points in different partitions will not overlap; S_2 is another scale factor that reserves a range of values for data points at a similar distance to its reference point. These index keys are then used to insert the CSPs to the B$^+$-tree. It is worthwhile noting that if two data points in an identical partition have the same distance from the reference point in this partition, they would have the same index key. We treat this case, by linking both points and appending them to the same entry in leaf node, in the same way as that for resolving CSPs having identical Price in the basic scheme.

For illustration purposes, we use 2-dimensional data points to show how the index keys are mapped to the leaf nodes in the B$^+$-tree. As shown in Fig. 5, the CSPs in partition P_i are stored in multiple continuous leaf nodes in a B$^+$-tree, where the dashed lines indicate the starting and ending entries of this partition in the tree. More specifically, the value $S_1 \cdot (i-1)$ determines the position of the starting entry in the tree. The reference point is the starting entry as its $\mathsf{dist}(O_i, O_i)$ is 0. As for other CSPs in this partition, the closer they are to the reference point, the closer they will be stored to the reference point. For CSPs at the same distance to the reference point such as CSP_2 and CSP_3, their index keys are the same, and would be linked together and stored in the same entry in leaf node.

After building the $n \times t$ MB^{cloud}-trees (one per partition) over the index key defined in Equation (2), the collector adds one more level on top of these trees by having a single global root node $h = \mathsf{Hash}(h_1||\ldots||h_{n\times t})$ concatenating the hash values of the roots of the $n \times t$ MB^{cloud}-trees, where h_i for $i = 1, \ldots, n \times t$ are the root hashes of MB^{cloud} trees. Then, the collector signs and publishes the root signature and the parameters of the MMB^{cloud}-tree including the list of reference points, the scale factors S_1 and S_2, and the index key value of the rightmost entry of each partition. After that, the collector sends the MMB^{cloud}-tree and the CSP profile database to the cloud brokers.

6.2.2 Phase 2: Service Selection

Consider a query $\mathcal{Q}$ ={(U_1, q_1), (U_2, q_2), ..., (U_k, q_k)}. At the filtering step, the cloud broker first selects the partitions of the same service type as specified in $\mathcal{Q}$. Then, the broker runs the range query containing the properties other than the service type in each selected partition. The range query consists of three main steps:

• **Step 1 (Query Normalization)**: A query may include just a subset of properties. In order to unify the follow-up process, the query normalization adds the domains of other non-query properties to the query. After normalization, the query $\mathcal{Q}$ will be re-written in the form:

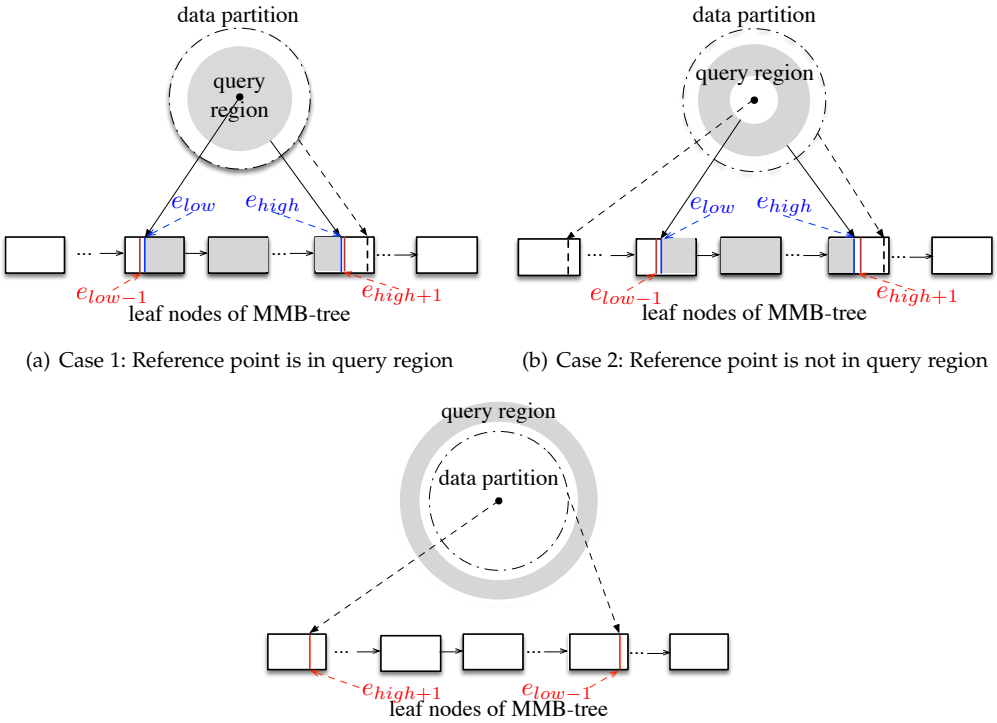
$$\begin{aligned}\mathcal{Q}' \;=\; & \{(U_1, [q_{low_1}, q_{up_1}]), \ldots, (U_{k-1}, [q_{low_{k-1}}, q_{up_{k-1}}]),\\ & (U_{k+1}, [\mathsf{min}(\mathcal{D}_{k+1}), \mathsf{max}(\mathcal{D}_{k+1})], \ldots,\\ & (U_7, [\mathsf{min}(\mathcal{D}_7), \mathsf{max}(\mathcal{D}_7)])\} \end{aligned} \quad (3)$$

where $\mathcal{D}_{k+1}$ to $\mathcal{D}_7$ are the domains of non-query properties. Note that $\mathcal{Q}'$ contains total 6 properties excluding the service type (i.e., U_k) in Q that has already been considered during the partition selection.

• **Step 2 (Query Transformation)**: Convert the multi-dimensional query $\mathcal{Q}'$ into a one-dimensional query interval. More precisely, given the reference point O_i = $[(U_1, v_1), ..., (U_7, v_7)]$ of the partition, the one-dimensional query interval is formed by the closest point (Q_c) and the farthest point (Q_f) in the query range to the reference point. Q_c and Q_f are computed as follows.

For each property U_i in $\mathcal{Q}'$, compare q_{low_i} and q_{up_i} with the reference point's property v_i. If q_{low_i} is farther from the reference point, i.e., $|q_{low_i} - v_i| > |q_{up_i} - v_i|$, we include q_{low_i} in the farthest point's property list. If the reference point's property is in the query range, we use it for the closest point. If not, we use q_{up_i} for the closest point. In the opposite case when $|q_{low_i} - v_i| \leq |q_{up_i} - v_i|$, we select q_{up_i} to be the farthest point's property. If the reference point's property is in the query range, we use it as for the closest point's property; otherwise we include q_{low_i} in the closest point. Finally, we compute the index keys of the closest and farthest points using Equation (2), and obtain the query interval $[Key_{Q_c}, Key_{Q_f}]$. Fig. 6 visualizes different scenarios of how a range query maps to the interval query. For better understanding, let us step through the query normalization and transformation process using the following example.

Example 4. *For simplicity of illustration, we consider only four properties in this example, which are* SType, Price, Security *and* OP. *Suppose that a client's query is* $\mathcal{Q} = \{(\mathsf{Price}, [0, 30]), (\mathsf{Security}, [0, 2]), (\mathsf{SType}, 1)\}$. *After*



(a) Case 1: Reference point is in query region

(b) Case 2: Reference point is not in query region

(c) Case 3: Partition does not overlap query region

Fig. 6. Positional Relations between Partitions and Query Regions

selecting the partitions of SType $= 1$*, the normalized query would look like* $\mathcal{Q}' = \{(\text{Price}, [0, 30]), (\text{Security}, [0, 2]), (\text{OP}, [0, 1])\}$ *(Notice that since* OP *is the non-query property, its domain needs to be included in the normalized query). Then, consider a reference point*

$$O = [(\text{Price}, 40), (\text{Security}, 0), (\text{OP}, 1), (\text{SType}, 1)]$$

The farthest point of Q' *regarding to* O *would be*

$$O_f = [(\text{Price}, 0), (\text{Security}, 2), (\text{OP}, 0), (\text{SType}, 1)]$$

and the closest point would be

$$O_c = [(\text{Price}, 30), (\text{Security}, 0), (\text{OP}, 1), (\text{SType}, 1)]$$

We now discuss the reason for choosing the Price *property in the farthest and closest points. It is clear that the* $q_{low_{\text{Price}}}$ *which is* 0 *in* $\mathcal{Q}$ *is farther from the reference point than* $q_{up_{\text{Price}}}$*. Thus, the value at* Price *of* O_f *is set as* 0*, equaling to* $q_{low_{\text{Price}}}$*. Furthermore, since the reference point's* Price *property (i.e.,* 40*) is not in the query range (i.e.,* $[0, 30]$*),* $q_{up_{\text{Price}}} = 30$ *is then included in the closest point* O_c*. Similarly, since* $q_{low_{\text{Security}}} = 0$ *is closer from reference point, the upper bound* $q_{up_{\text{Security}}}$ *should be included in* O_f*. Further, the reference point's* Security *property (i.e.* 0*) just falls in the query range (i.e.,* $[0, 2]$*), and thus it (i.e., the* Security *property of reference point) is included in the closest point* O_c*. For* OP *property, since* $q_{low_{\text{OP}}}$ *is farther than* $q_{up_{\text{OP}}}$ *from reference point and the* OP *property of reference point is in query range,* $q_{low_{\text{OP}}}$ *and reference point's* OP *property are respectively included in* O_f *and* O_c*.*

- **Step 3 (Querying the Tree)**: After the query transformation, we execute the one-dimensional query $[Key_{Q_c}, Key_{Q_f}]$, where Key_{Q_c} and Key_{Q_f} are respectively computed from Q_c and Q_f according to equation (2), on the MB^{cloud}-tree of the corresponding partition in the exact the same way as described in the Phase 2 of the basic scheme. This also includes the same refinement step.

At the end, the broker sends the query results of the selected partitions and the associated proof messages to the client.

6.2.3 Phase 3: Results Verification

The process of results verification is the same as that in the basic approach in terms of satisfiability and authenticity check. The only difference is the verification of boundary entries during the completeness check. In the basic approach, the boundary entries e_{low-1} and e_{high+1} can be easily verified by comparing their price values directly with the query range to see if they are outside the query range. In this advanced approach, the verification of e_{low-1} and e_{high+1} is not straightforward since their associated one-dimensional index keys in the MMB^{cloud}-tree are not comparable with the multi-dimensional query ranges. Therefore, the client needs to first convert his/her multi-dimensional query to the one-dimensional query $([Key_{Q_c}, Key_{Q_f}])$ using the same algorithm in the Phase 2 of this approach. Note that, the client has sufficient information for the query transformation as they are published by the collector.

After the transformation, the client performs the boundary verification for each data partition. Two possible cases may arise. The first case is when the query result set is not empty. In this case, the client checks if e_{low-1} is smaller than Key_{Q_c}, and if e_{high+1} is greater than Key_{Q_f}. If so, they are real boundary entries. The second case is when no query

result has been returned from this data partition. Then, the client needs to check if e_{high+1} equals to the published index key of the leftmost entry of that partition. If so, the boundary is correct. After verifying that all the pairs of the received boundary entries are correct, the client can conclude that the obtained query results are complete.

Similar to the basic approach, the advanced approach also has the ability to detect any misbehavior of the cloud broker, as demonstrated in the following theorem.

Theorem 2. For any query $\mathcal{Q} = \{(U_1, q_1), (U_2, q_2), ..., (U_k, q_k)\}$, let $\mathcal{R}$ denote the query results returned by the MMB^{cloud}-tree-based approach. The correctness of $\mathcal{R}$ (in Definition 2) can be verified by the client.

We omit the detailed proof here since it is similar to that of the basic approach.

In addition, it is worth noting that our approaches can be easily extended to answer service selection queries that contain multiple query ranges. For example, suppose that the non-numerical property OP of CSPs have three values: Mac OS, Windows and Linux, which are mapped to numeral values 2, 1, and 0, respectively. Given a service selection query that specifies the operating system as Linux or Mac OS, i.e., our system can answer this query by treating it as two separate queries: OP=[2] and OP=[0].

7 Performance Study

In this section, we compare the performance of our proposed two schemes with the state-of-the-art VR-tree schema [18] in terms of database construction, service selection and results verification. Performance is measured in terms of the CPU time. In what follows, we first provide the experimental settings and then report the results.

7.1 Experimental Settings

We implemented our approaches using the Polarssl cryptographic library [35]: the hash used in algorithms used MD5, and the collector's signature was realized using RSA signing algorithm (`rsa_pkcs1_sign` in Polarssl in particular). The parameters S_1 and S_2 in MMB^{cloud}-tree were set as 1000 and 100, respectively. The VR-tree is simulated using the R-tree library [36]; the signature in the VR-tree is simulated according to Boneh's scheme in [37] and a pairing-based cryptographic library PBC [38]. All the tests were conducted on a Mac OS X machine with processor of 1.7 GHz Intel Core i7 and memory of 8 GB 1600 MHz DDR3.

To generate the CSP dataset, we first analyze the available manifests of top ten CSPs in 2013 including Salesforce, Amazon, Microsoft, Oracle, Google, SAP, SoftLayer, Terremark, Rackspace and NetSuite, and extract a set of information based on the property universe illustrated in Table 2. Next, we identify the acceptable numeric values for each of the properties, according to the maximum and minimum service levels offered for a given property by any of the CSPs. This gives us a starting set of CSP points which will shape the experimental CSP dataset. Specifically, we then use a pseudo random number generator to generate a subset of the total possible combinations, filter out the outliers and use 5000 of them representing synthetic CSPs for our experiments.

7.2 Experimental Results

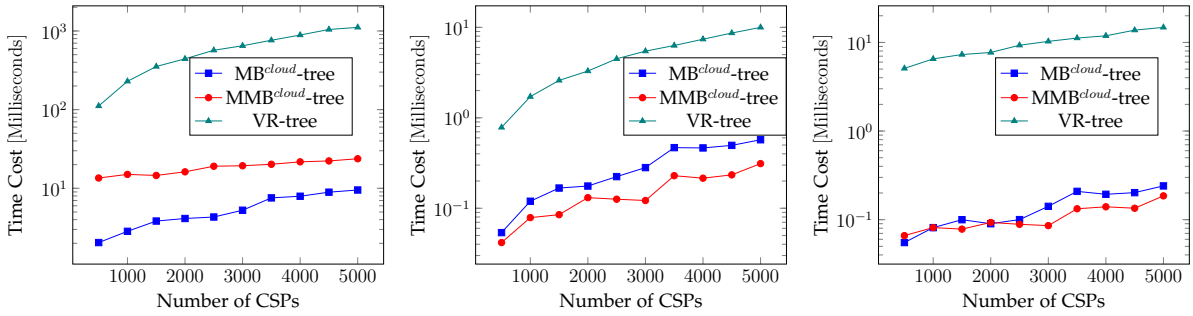
7.2.1 Effect of the Number of CSPs

In the first set of experiments, we randomly generate the service selection queries including four properties, Price, SType, Security and QoS in particular, whereby Price is randomly selected from [1, 100], and SType, Security and QoS are selected from [1, 3]. We set the number of partitions to be clustered in MMB^{cloud}-tree as 6, and compare the performance of three schemes by varying the total number of service providers from 500 to 5000.

As shown in Fig. 7(a), the database construction time of three schemes increases with the number of CSPs. The reason is straightforward. The more CSPs, the more insertion operations need to be conducted to build the authenticated indices. Moreover, both our methods outperform the VR-tree by one or two orders, because the VR-tree involves a large number of signatures for the CSPs as well as partitions in total construction, while our methods only need lightweight hash operations. In addition, we also observe that the MMB^{cloud}-tree-based approach takes longer time to construct the authenticated CSP database. This is because, 1) MMB^{cloud}-tree requires an additional step to cluster the CSPs; 2) MMB^{cloud}-tree has one more layer than the MB^{cloud}-tree, resulting that it needs to compute more hash values during the tree construction. However, it is worth noting that the data construction just needs to be conducted once and can be executed offline, which will not affect the performance of service selection and results verification.

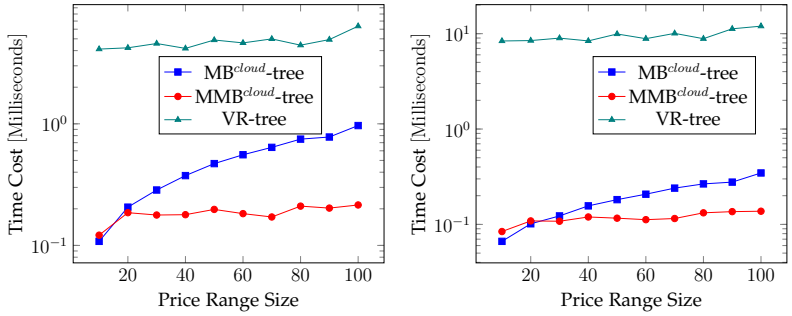
Regarding the performance of service selection (Fig. 7(b)), the selection time of three schemes also grows with the increase of the CSPs. The MMB^{cloud}-tree is significantly faster than both the MB^{cloud}-tree and the VR-tree. This is because the MMB^{cloud}-tree considers all the properties to be indexed. Specifically, the MMB^{cloud}-tree used in the advanced approach reduces the number of candidate CSPs to be refined and makes the refinement phase in service selection much more efficient. Especially when the number of CSPs is large and not fully satisfying CSPs fall into the Price query range, the benefits of the MMB^{cloud}-tree become more significant. In contrast, the VR-tree needs to execute a number of modular multiplications to aggregate the selected CSP signatures and all the partition signatures which is time consuming.

Next, Fig. 7(c) shows the results verification time of the three schemes, with respect to the same queries resolved in selection in Fig. 7(b). As expected, the MMB^{cloud}-tree achieves the best performance among all. This is again attributed to the efficient MMB^{cloud}-tree which returns much fewer number of candidate CSPs for verification. The VR-tree is almost two orders of magnitude slower than either one of our schemes. This is because the results verification of VR-tree needs at least two bilinear pairing operations and a number of modular multiplications (depending on the number of signatures returned), which are much slower than computing hash values in MB^{cloud}-tree and MMB^{cloud}-tree. In addition, it is worthwhile noting that the MMB^{cloud}-tree approach is not always better than MB^{cloud}-tree approach, and we will further illustrate this point in our next set of experiments.



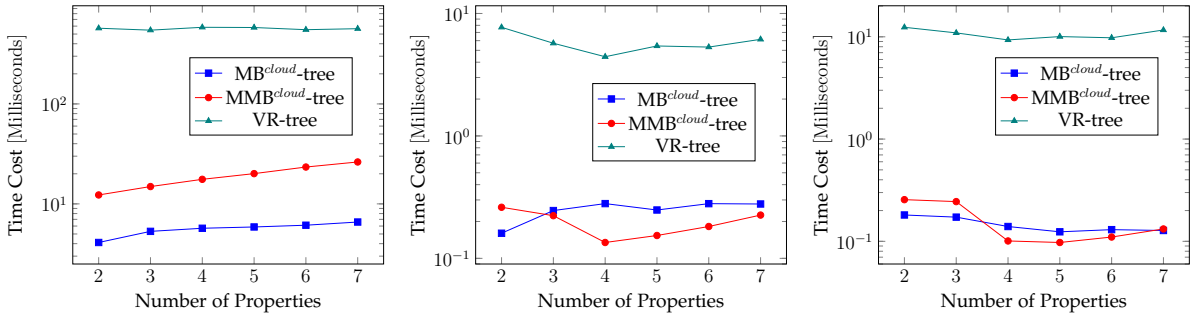
(a) Database Construction (b) Service Selection (c) Results Verification

Fig. 7. Effect of the Number of CSPs



(a) Service Selection (b) Results Verification

Fig. 8. Effect of the Queried Price Range



(a) Database Construction (b) Service Selection (c) Results Verification

Fig. 9. Effect of the CSP Properties

7.2.2 *Effect of the Queried Price Range*

In this set of experiments, we generate a 4-dimensional (SType, Price, Security and QoS) CSP database. We vary the price range size in queries and evaluate the effect of the query range of Price in 4-properties. The total number of CSPs in this test is 2500.

Since the database construction time is not affected by the query range, we omit the figure of database construction here. Fig. 8(a) shows the performance of the service selection. We can observe that the service selection time taken by MB^{cloud}-tree increases with the price range, while that of the MMB^{cloud}-tree and the VR-tree is less affected by the variation of queried price range. This is because, with the increase of price range, more candidate CSPs fall into the price query range in the MB^{cloud}-tree adopted by basic approach and hence more time is needed to filter the results. In the worst case, when the price range size is 100, the MB^{cloud}-tree scheme needs to refine all the total 2500 CSPs. On the contrary, the VR-tree and MMB^{cloud}-tree index all properties of CSPs and hence achieve a balanced performance when varying the different parameters of queries. In addition, we observe that, in the case of 10-ranged queries, MB^{cloud}-tree is better than MMB^{cloud}-tree. The reason is that, in the small sized price range, the Price is more selective, and thus MB^{cloud}-tree is able to select similar (or even fewer) number of candidate CSPs for refinement. As for the results verification (Fig. 8(b)), three schemes demonstrate similar performance trends as that in service selection (Fig. 8(a)) due to the same reason.

7.2.3 *Effect of the Number of Properties*

In the third set of experiments, we fix the number of CSPs as 2500, and vary the number of CSP properties from 2 to 7, following the order Price, SType, Security, QoS, ISize, OP, Sub.

Notice that, except Price ranging from 1 to 100, ISize ranging from 1 to 6 and Sub ranging from 1 to 2, the rest of properties all range from 1 to 3.

Regarding the database construction shown in Fig. 9(a), the MB^{cloud}-tree and the VR-tree has constant performance while the MMB^{cloud}-tree takes slightly more time to construct the indexes when the number of CSPs' properties increase. The reason is that, the dominated computation in database construction in MMB^{cloud}-tree is the clustering step. Time cost in this step essentially grows with the dimensions of data to be clustered.

As for the service selection (Fig. 9(b)), the MMB^{cloud}-tree approach still performs best in most cases. The performance of the MB^{cloud}-tree decreases with the increase of the number of properties due to the decreased selectivity of the indexed Price property. The VR-tree is very time consuming due to the need to compute and aggregate multiple signatures. The verification performance of the three scheme (as shown in Fig. 9(b)) is similar to the service selection.

In summary, our proposed MB^{cloud}-tree and MMB^{cloud}-tree-based methods significantly outperform the state-of-the-art VR-tree in all the tested cases. The MMB^{cloud}-tree performs consistently well for various types of queries. Only in few specific cases, such as the case of queries with fewer properties or a small size price range, the MB^{cloud}-tree achieves better performance than the MMB^{cloud}-tree.

8 Practical Consideration

Recall our proposed both schemes work by building authenticated index (i.e., the MB^{cloud}-tree and MMB^{cloud}-tree) for results verification. For each time of verification, our schemes guarantee 100% probability of catching the dishonest action of cloud broker. In spite of this, our methods still demand a verification cost on millisecond level (in each time), which might impose computational on client side for a large number of verifications. In this section, we establish a probabilistic model and analyze the number of verifications needed for guaranteeing a threshold confidence (say T) of catching the dishonesty.

Assume the cloud service selection is executed for several times, and the broker cheats for probability p in each time of query. Denote $A_{k,n}$ the probability of catching cheating with n queries, of which only the k are verified. We have $A_{k,n} = (1-p)^{k-1}p$. This is because, 1) The clients in our both schemes can perfectly verify the correctness of query results; 2) The cloud broker should make cheating and be caught in the last query (i.e., the nth query); 3) The first $k-1$ queries verified by clients should be honest to avoid being caught in verification. It is clear that the probability $A_{k,n}$ is independent of n.

Based on the probabilistic formula of $A_{k,n}$, we can further compute the probability of catching cheating with at most k times for verification is $\sum_{i=1}^{k} A_{i,n} = \sum_{i=1}^{k}(1-p)^{i-1}p = 1-(1-p)^k$. It is clear that if we want to let the cheating be caught not less than a threshold probability T (i.e., $1-(1-p)^k \geq T$), the client should verify at least $\log_{1-p}(1-T)$ times.

In Fig. 10, we provide some numerical examples about confidence of detecting dishonest action proportional to the number of times to be verified. In this numerical example,

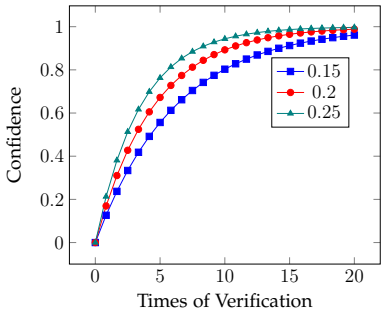


Fig. 10. Number of Times Verified

suppose the cloud service selection has been queried 100 times. We present three cases wherein we assume there are respectively 0.15, 0.2 and 0.25 queries cheated by cloud broker, and show the confidence of detecting this cheating. It is clear from Fig. 10 that, in order to achieve a detection confidence of 0.9, we only need to check a small set of queries. Precisely, the clients will need to check 8, 11 and 15 queries in the 15%, 20% and 25% case for ensuring 0.9 confidence. Only a subset of the query results are to be verified for high confidence results, and we can check only part of the query results to save computing resources.

9 Conclusion

In this paper, we presented an innovative Cloud Service Selection Verification (CSSV) system to achieve cheating-free cloud service selection under a cloud brokerage architecture. The core of our system is an efficient authenticated index structure to ensure the authenticity, the satisfiability and the completeness of the service selection results. Our theoretical and experimental results demonstrate the effectiveness and efficiency of our schemes compared with the state-of-the-art. As part of our future work, we plan to consider a verifiable scheme for best service selection query whereby the broker returns only the best CSP instead of all candidate CSPs with respect to a client's request.

References

[1] I. Petri, M. Punceva, and O. F. Rana, "Broker emergence in social slouds," *2013 6th International Conference on Cloud Computing*, pp. 669–676, 2013.

[2] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *IMC '10: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. New York, New York, USA: ACM Press, 2010, pp. 1–14.

[3] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: towards a benchmark for the cloud," in *DBTest '09: Proceedings of the Second International Workshop on Testing Database Systems*. ACM Request Permissions, Jun. 2009.

[4] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What are you paying for? performance benchmarking for Infrastructure-as-a-Service offerings," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, 2011, pp. 484–491.

[5] Z. ur Rehman, O. K. Hussain, S. Parvin, and F. K. Hussain, "A framework for user feedback based cloud service monitoring," in *2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, 2012, pp. 257–262.

[6] L. Li and Y. Wang, "Subjective trust inference in composite services." *AAAI*, 2010.

[7] L. Qu, Y. Wang, and M. A. Orgun, "Cloud service selection based on the aggregation of user feedback and quantitative performance assessment," in *2013 IEEE International Conference on Services Computing (SCC)*, 2013, pp. 152–159.

[8] L. Xin and A. Datta, "On trust guided collaboration among cloud service providers," in *2010 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2010, pp. 1–8.

[9] S. Sundareswaran, A. Squicciarini, and D. Lin, "A brokerage-based approach for cloud service selection," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. IEEE, Aug. 2012, pp. 558–565.

[10] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM Request Permissions, Jun. 2006.

[11] E. Mykletun, M. Narasimha, and G. Tsudik, "Signature bouquets: immutability for aggregated/condensed signatures," in *Computer Security – ESORICS 2004*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 3193, pp. 160–176.

[12] M. Narasimha and G. Tsudik, "DSAC: integrity for outsourced databases with signature aggregation and chaining," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, 2005, pp. 235–236.

[13] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, "Verifying completeness of relational query results in data publishing," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005, pp. 407–418.

[14] H. Pang, J. Zhang, and K. Mouratidis, "Scalable verification for outsourced dynamic databases," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 802–813, 2009.

[15] Q. Zheng, S. Xu, and G. Ateniese, "Efficient query integrity for outsourced dynamic databases," in *CCSW '12 Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*. ACM, 2012, pp. 71–82.

[16] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *The VLDB Journal*, vol. 18, no. 3, pp. 631–648, 2009.

[17] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic data publication over the internet," *Journal of Computer Security*, vol. 11, no. 3, pp. 291–314, 2003.

[18] W. Cheng, H. Pang, and K.-L. Tan, "Authenticating multi-dimensional query results in data publishing," in *Proceedings of the 20th IFIP WG 11.3 Working Conference on Data and Applications Security*, 2006, pp. 60–73.

[19] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, 2004, pp. 560–571.

[20] Z. Yang, S. Gao, J. Xu, and B. Choi, "Authentication of range query results in mapreduce environments," in *Proceedings of the Third International Workshop on Cloud Data Management*, 2011, pp. 25–32.

[21] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "iDistance: an adaptive B+-tree based indexing method for nearest neighbor search," *ACM Trans Database Systems (TODS)*, vol. 30, no. 2, pp. 364–397, Jun. 2005.

[22] S. Papadopoulos, D. Papadias, W. Cheng, and K.-L. Tan, "Separating authentication from query execution in outsourced databases," in *IEEE 25th International Conference on Data Engineering (ICDE)*, 2009, pp. 1148–1151.

[23] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *ACM Transactions on Storage (TOS)*, vol. 2, no. 2, pp. 107–138, 2006.

[24] S. Bajaj and R. Sion, "CorrectDB: SQL engine with practical query authentication," *Proceedings of the VLDB Endowment*, vol. 6, no. 7, pp. 529–540, 2013.

[25] D. Papadopoulos, S. Papadopoulos, and N. Triandopoulos, "Taking authenticated range queries to arbitrary dimensions," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 819–830.

[26] A. Goscinski and M. Brock, "Toward dynamic and attribute based publication, discovery and selection for cloud computing," *Future Generation Computer Systems*, vol. 26, no. 7, pp. 947–970, Jul. 2010.

[27] S. M. Han, M. M. Hassan, C. W. Yoon, and E. N. Huh, "Efficient service recommendation system for cloud computing market," in *ICIS '09 Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, 2009, pp. 839–845.

[28] M. Godse and S. Mulik, "An approach for selecting Software-as-a-Service (SaaS) product," *CLOUD '09. IEEE International Conference on Cloud Computing, 2009*, pp. 155–158, 2009.

[29] S. K. Garg, S. Versteeg, and R. Buyya, "SMICloud: a framework for comparing and ranking cloud services," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, 2011, pp. 210–218.

[30] R. Zhang, Y. Zhang, and C. Zhang, "Secure top-K query processing via untrusted location-based service providers," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 1170–1178.

[31] R. C. Merkle, "Protocols for public key cryptosystems." *IEEE Symposium on Security and privacy*, pp. 122–134, 1980.

[32] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, Jan. 2004.

[33] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish, "Indexing the distance: an efficient method to KNN processing," in *Proceedings of the 27th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., Sep. 2001, pp. 421–430.

[34] J. MacQueen and others, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. California, USA, 1967, pp. 281–297.

[35] Polarssl. [Online]. Available: https://polarssl.org

[36] R-tree code c version. [Online]. Available: http://www.superliminal.com/sources/sources.htm

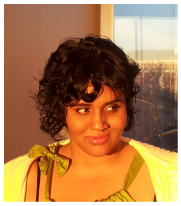
[37] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Advances in Cryptology — EUROCRYPT 2003*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 2656, pp. 416–432.

[38] B. Lynn. (2010) The pairing-based cryptography (pbc) library. [Online]. Available: http://crypto. stanford. edu/pbc

Jingwei Li received the BS degree in mathematics in 2005 from the Hebei University of Technology and the PhD degree in computer application technology in 2014 from Nankai University, China. Now he is a post doctoral research fellow at The Chinese University of Hong Kong. His research interests include applied cryptography, cloud security and storage systems.

Anna Cinzia Squicciarini is an assistant professor at the college of Information of Information Science and Technology, at the Pennsylvania State University. During the years of 2006-2007 she was a post doctoral research associate at Purdue University. Squicciarini's main interests include access control for distributed systems, privacy, security for Web 2.0 technologies and grid computing. Squicciarini earned her Ph.D. in Computer Science from the University of Milan, Italy, in 2006. Squicciarini is the author or co-author of more than 60 articles published in refereed journals, and in proceedings of international conferences and symposia. She is an IEEE member.

Dan Lin is an assistant professor at Missouri University of Science and Technology. She received the PhD degree in Computer Science from the National University of Singapore in 2007, and was a post doctoral research associate at Purdue University for two years. Her main research interests cover many areas in the fields of database systems, information security, cloud computing, and vehicular ad-hoc networks.



Smitha Sundareswaran received the bachelor's degree in electronics and communications engineering in 2005 from Jawaharlal Nehru Technological University, Hyderabad, India. She is currently a PhD candidate in the College of Information Sciences and Technology at the Pennsylvania State University. Her research interests include policy formulation and management for Distributed computing architectures.

Chunfu Jia got his Ph.D. in Engineering from Nankai University in 1996. He has finished his post doctoral research from the University of Science and Technology of China. Now he is a professor in Nankai University, China. His current interests include computer system security, network security, trusted computing, malicious code analysis, etc.