Efficient Top-k Join Processing over Encrypted Data in a Cloud Environment

Jong Wook Kim
Department of Media Software, Sangmyung University
Jongno-gu, Seoul 03016 - Republic of Korea
[e-mail: jkim@smu.ac.kr]
*Corresponding author: Jong Wook Kim

Received March 30, 2016; revised August 16, 2016; accepted September 18, 2016; published October 31, 2016

Abstract

The benefit of the scalability and flexibility inherent in cloud computing motivates clients to upload data and computation to public cloud servers. Because data is placed on public clouds, which are very likely to reside outside of the trusted domain of clients, this strategy introduces concerns regarding the security of sensitive client data. Thus, to provide sufficient security for the data stored in the cloud, it is essential to encrypt sensitive data before the data are uploaded onto cloud servers. Although data encryption is considered the most effective solution for protecting sensitive data from unauthorized users, it imposes a significant amount of overhead during the query processing phase, due to the limitations of directly executing operations against encrypted data. Recently, substantial research work that addresses the execution of SQL queries against encrypted data has been conducted. However, there has been little research on top-*k* join query processing over encrypted data within the cloud computing environments. In this paper, we develop an efficient algorithm that processes a top-*k* join query against encrypted cloud data. The proposed top-*k* join processing algorithm is, at an early phase, able to prune unpromising data sets which are guaranteed not to produce top-*k* highest scores. The experiment results show that the proposed approach provides significant performance gains over the naive solution.

Keywords: Cloud computing, top-k join query processing, database encryption, ranking, efficiency, rank join

http://dx.doi.org/10.3837/tiis.2016.10.028 **ISSN : 1976-7277**

1. Introduction

In recent years, top-*k* join queries have received much attention, largely due to the vast increases in data set sizes produced by a variety of applications, such as bioinformatics, e-commerce, and social media. Top-*k* join queries return the most interesting *k* tuples to the user, thereby resulting in manageable sizes of the result sets. The result sets are ordered by a user-provided preference criterion, which is expressed by a monotonic score function. The properties of monotone ranking functions of top-*k* queries enable efficient query processing by eliminating unpromising data sets that are not expected to produce the top-*k* highest scores at an early phase.

As cloud computing service is getting more attention these days, the interest in cloud-based data outsourcing, in which customers' data are remotely stored and managed by the public cloud, such as Amazon EC2 [3] and Microsoft Azure [4], has correspondingly increased. Cloud-based data outsourcing solutions have a great advantage in that they offer the data owner a low initial investment, scalability, and flexibility. However, they pose many security challenges, because the users' sensitive data are stored within the public cloud servers, which are very likely to reside outside of the trusted domain of the users. Hence, in order to protect sensitive data from unauthorized access, it is essential to encrypt sensitive information, such as financial information and health records, before such data is uploaded into the cloud servers. For example, in recent years, personal health record (PHR) systems, such as Microsoft HealthVault [33], store PHRs electronically within a cloud database. As proposed in [34, 35], to protect the patients' ownership of their own PHRs, sensitive data should be encrypted before outsourcing it to the cloud.

Data encryption is generally considered the most effective solution for protecting sensitive data from unauthorized users. However, data encryption imposes a significant amount of overhead during the query processing phase, mainly due to the limitation of directly executing operations over encrypted data. That is, in order to process a user query against encrypted data, it is necessary to transfer a large amount of data from the cloud to the client, decrypt the data and execute the query against the decrypted data. This naive solution is intuitive and straight forward; however, it is clearly impractical, due to the potentially very large costs incurred by the transfer of a large data set from the cloud to the client, followed by decryption before finally performing client-side query processing against the decrypted data.

Recently, substantial research work has been conducted into the possibility of directly executing SQL queries against encrypted data [5, 6, 7, 8, 9, 10]. However, little research has addressed top-*k* join query processing against encrypted data within the cloud computing environments. Indeed, the problem caused by encrypted data becomes more serious when considering top-*k* join queries. This is because users are often interested in focusing on a small number of top results generated from execution of the top-*k* join query, rather than browsing the entire result set. Thus, the naïve approach, which transfers the entire data set from the cloud to the client, is highly inefficient. In this paper, we investigate an algorithm which aims to support efficient top-*k* join query processing against encrypted cloud data.

1.1 Problem Definition and Naive Solution

Fig. 1 illustrates the system architecture used in this paper. Data is outsourced to the cloud servers, which might be curious about the stored data, but honestly execute the tasks assigned and return task results honestly (i.e., honest-but-curious model). We also assume that a data owner encrypts the sensitive data and stores the encrypted data within the cloud servers. Users submit a ranked query against the outsourced data in the cloud from a client machine with limited computational resources. Let us assume that the outsourced data is stored in a raw table data within a distributed file system, such as HDFS (Hadoop Distributed File System). We, further, assume that each table R_p consists of a set of sensitive (and thus encrypted) attributes S_p and a set of non-sensitive attributes N_p. To facilitate understanding of the notation used in the paper, a notation table is provided in **Table 1**. In this paper, we focus on ranked (top-*k*) equi-join queries that can be written as follows:

SELECT	select-list
FROM	$R_1, R_2, \ldots, R_n$
WHERE	equi-join-condition $(R_1, R_2, \ldots, R_n)$
ORDER BY	f $(e_1 \in S_1, e_2 \in S_2, \ldots, e_n \in S_n)$
STOP AFTER	k

Here, as is common in existing top-*k* join works, $f(\cdot)$ is a monotonic score function. Note that in this paper, we assume that the input values of a score function are encrypted data. A naïve solution to handle such a top-*k* join query over encrypted and outsourced data is summarized as follows (**Fig. 2**):

1. **Client side**: As previously described, a top-*k* join query ranks the results based on a specified score function and returns the best *k* results to the user. However, if the input values of the score function are encrypted data, performing such an operation on the cloud side is not possible. Therefore, upon receiving the top-*k* join query, the client machine rewrites it as follows:

SELECT	select-list, $e_1, e_2, \ldots, e_n$
FROM	$R_1, R_2, \ldots, R_n$
WHERE	equi-join-condition $(R_1, R_2, \ldots, R_n)$

   That is, since it is not possible to compute the values of the score function, $f(e_1, e_2, \ldots, e_n)$ on the cloud side due to encrypted data, the client machine rewrites the top-*k* join query as an equi-join query that is executable by the cloud servers. We also note that the cloud servers must send the encrypted values, $e_1, e_2, \ldots, e_n$, to the client (by their inclusion within the select list) so that the values of the score function, $f(e_1, e_2, \ldots, e_n)$, can be computed on the client side by decrypting the values of $e_1, e_2, \ldots, e_n$.

2. **Cloud side**: The cloud servers execute the above equi-join query on encrypted data, and send the results to the client system.

3. **Client side**: Upon receiving the results from the cloud, the client machine first decrypts the values of $e_1, e_2, \ldots, e_n$, computes the values of the score function, ranks the results based on the score values, and returns the best *k* results to the user.

Obviously, this naive solution of downloading all equi-join results from the cloud and performing the remaining client-side query processing, including decrypting encrypted data, is highly inefficient due to the large amount of bandwidth and client-side workload. Furthermore, considering that users are interested in only the best *k* results, the cloud-side query processing wastes the system resources to produce intermediate results, most of which are likely to be not in the top-*k*.

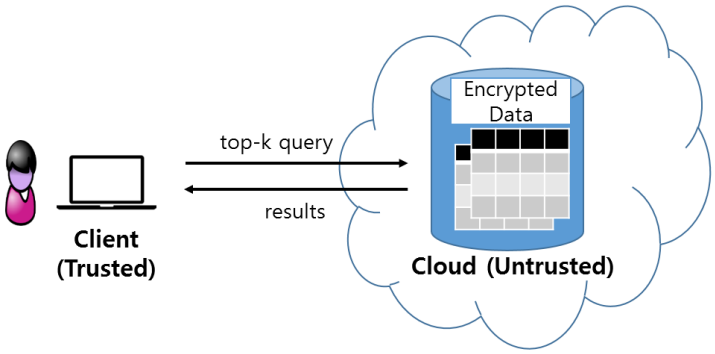


Fig. 1. A system architecture assumed in this paper

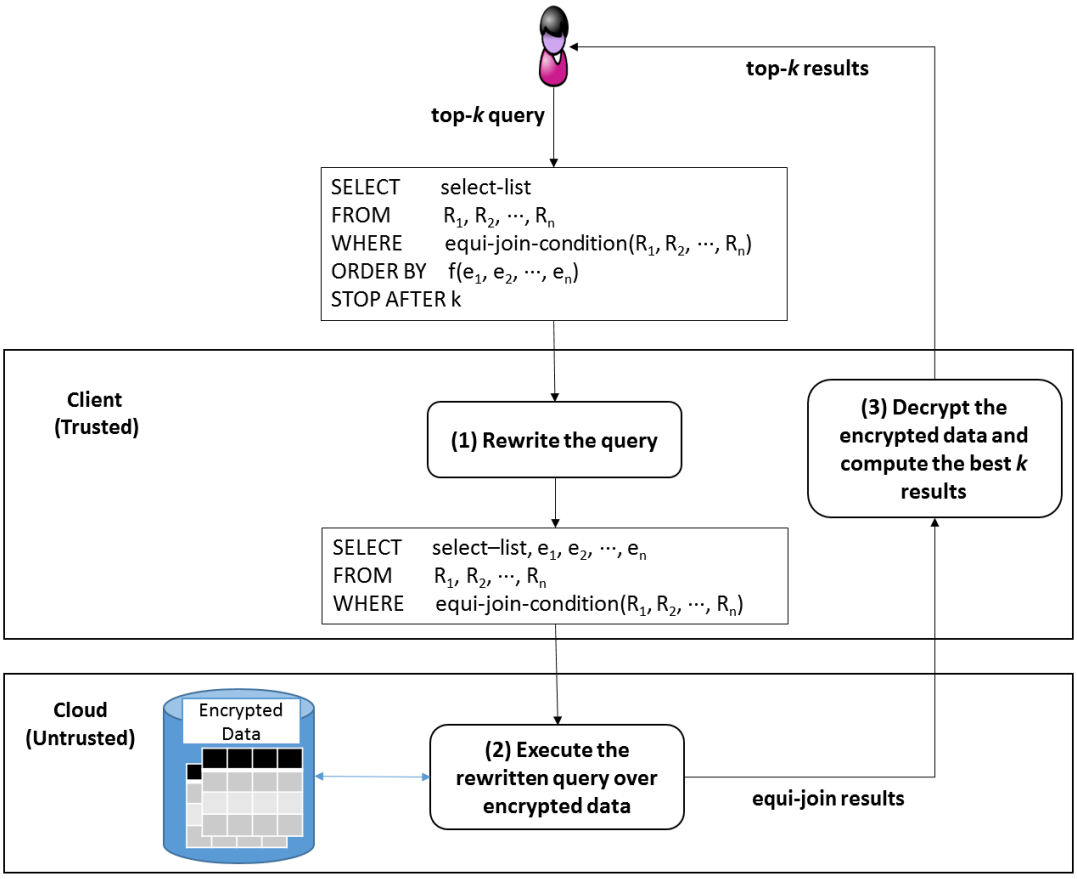


Fig. 2. A naïve solution to process a top-*k* join query over encrypted cloud data

Table 1. Notation table

Notation	Definition
R_p	A relation p
S_p	A set of sensitive attributes of R_p
N_p	A set of non-sensitive attributes of R
$R_{p,i}$	The *i*-th segment for a relation R_p
$f_{p,i}$	The counting Bloom filter for the *i*-th segment $R_{p,i}$
$f_{p,i[r]}$	The *r*-th bit of the counting Bloom filter $f_{p,i}$

1.2 Our Contribution

To address the performance problems inherent in processing a top-*k* join query against encrypted cloud data, we propose a novel top-*k* join query processing algorithm which efficiently computes the best *k* results against encrypted cloud data. The algorithm presented in this paper is able to identify and eliminate join results that are guaranteed not to produce the top-*k* highest scores at the cloud servers. In particular,

- we first develop a method for effectively estimating the portion of the join space (i.e., the top-*k* candidate space) which will produce the best *k* results by leveraging an order-preserving encryption technique in conjunction with the use of a counting Bloom filter, and
- we then present a top-*k* join processing algorithm on encrypted cloud data, based on the top-*k* candidate space identification scheme.

The rest of this paper is structured as follows: In the next section, we present the related work. In Section 3, we present the proposed algorithm for efficiently processing top-*k* join queries against encrypted data stored within the cloud servers. In Section 4, we experimentally evaluate the proposed approach and in Section 5, we conclude the paper.

2. Related Work

Threshold algorithm (TA) is the most popular method for top-*k* queries [1, 2, 11, 12]. Given *M* sorted-lists, TA algorithm assumes that each object has a single score in each list and an aggregation function, which combines independent object's scores in each list, is monotone. Many variants of TA algorithm have been proposed in the literature. An approximate-based algorithm [13, 14] leverages the probabilistic model in order to terminate earlier than the original TA algorithm. With the development of web, there have been studies to determine the ranking of objects based on the score of text data which are related with the specific objects [15, 16]. Ntarmos et al. [17] introduced algorithms for top-*k* joins in cloud NoSQL databases, such as BigTable, HBase, Cassandra, etc. Doulkeridis et al. [18] proposed efficient processing of top-*k* joins in a distributed setup where servers store fragments of relations individually. Yu et al. [19] introduced an algorithm for top-*k* queries over batch-oriented data sets in cloud computing environments. Although there has been substantial research conducted to address the processing of top-*k* queries, they are not applicable to the problem discussed in this paper, due to their failure to support encrypted data.

There have been proposals to execute SQL queries against encrypted data. Hacigumus et al. [7] proposed an early approach, which partially executes an SQL query at the server and performs final query processing on the client side. Agrawal et al. [5] proposed an order

preserving encryption scheme (OPES) by which some SQL query types can be directly handled via ciphertext (without decryption). Ge et al. [6] conducted a comprehensive study of computing SUM and AVG function values included within aggregation queries by using partially homomorphic encryption schemes. CryptDB [8] is a well-known system for processing SQL queries against encrypted data. CryptDB uses multiple data encryption schemes, such as order-preserving encryption and partially homomorphic encryption, and dynamically adjusts the layer of encryption on the DBMS server. MONOMI [9] is built on CryptDB's design of using multiple data encryption functions. In order to provide efficient analytical query processing, this algorithm uses a split client/server query execution approach which intelligently partitions the execution of each query across an untrusted server and a trusted client machine.

Many security-related issues have been studied in the areas of cloud computing and Internet of Things (IoT) literature. Han et al. proposed the multi-valued and ambiguous scheme to provide data confidentiality in data communication between the cloud and wireless body area networks [20]. Liang et al. introduced a ciphertext-policy attribute-based re-encryption scheme which relies on the PRE technology in the attribute-based encryption cryptographic setting for secure cloud storage data sharing [21]. In [22], the authors presented the cloud-centric, multi-level authentication method as a service to provide a secure communication between IoT devices and the cloud. Xu et al. investigated the secure transmission problem in the IoT with unknown eavesdroppers [23].

Another work that is related to this paper is the resource allocation problem within distributed environments, including cloud computing platforms. Wei et al. presented a cloud resource allocation model, which is based on an imperfect information Stackelberg game [24]. Shojafar et al. investigated the resource management problem for real-time vehicular cloud services [25]. [26, 27] proposed methods, which attempt to minimize the overall energy consumed in typical distributed data centers. [28, 29] presented an energy-efficient algorithm that solves the coverage problem of associating a wireless network with the minimum number of sensor nodes. The problem studied in this paper is different from the above references, in that this paper focuses on developing a methodology for efficiently processing top-*k* join queries against encrypted cloud data.

3. Efficient Top-k Join Processing over Encrypted Data

In this section, we describe the proposed algorithm for efficiently computing top-*k* join query results against encrypted cloud data.

3.1 Preliminary: Order-Preserving Encryption

Order-preserving encryption (OPE) guarantees that the order of encrypted data is identical to the order of original data, and thus allows comparison operations to be directly applied to encrypted data without decrypting it [5, 30]. In other words, any two unencrypted values x and y such that $x > y$ map to corresponding encrypted values such that $OPE(x) > OPE(y)$. OPE has recently received increased interest from the database community, because the database system can still leverage an existing index structure, such as a B^+-tree, indexed on the encrypted values, to efficiently process equality and range queries. Similarly, ORDER BY, MAX, and MIN operations can be directly applied to the encrypted data.

In this work, we rely on OPE to encrypt sensitive data to protect the data from unauthorized access, while preserving numerical ordering of plaintext. This is useful in that it provides the

capability to prune unpromising data sets that are guaranteed not to produce the top-*k* highest scores at the cloud servers, thereby alleviating the overhead of the client machine.

3.2 Identifying Top-*k* Candidate Space

Before explaining the proposed algorithm in detail, we first give an intuitive example in **Fig. 3**. Let us consider a two-way top-*k* join query of two relations, R_1 and R_2. Then, **Fig. 3-(a)** graphically represents the join space between two relations in which the area containing the blue rectangles holds the top-*k* results. Here, the X-axis represents the sensitive attribute, $e_1(\in S_1)$, of R_1, sorted from left to right by increasing values of e_1 (the Y-axis is similarly defined). The properties of the monotonic ranking function guarantee that the top-*k* results are located at the upper-right corner of this join space. Hence, if we effectively identify that portion of the join space that will contain the top-*k* results by leveraging the monotonic property of a score function, then we can significantly speed up the top-*k* join processing by not processing data points that will not contribute to the *k* highest scores. Based on this intuition, we now explain how to effectively estimate the portion of the join space which will produce the top-*k* results (hereafter we call this portion of the join space the top-*k* candidate space). The proposed approach is summarized as follows:

1. The entire join space is partitioned into multiple subspaces and the join cardinality for each subspace is computed
2. The top-*k* candidate space is estimated based on the join cardinalities.

We now explain and describe each of these steps in detail.

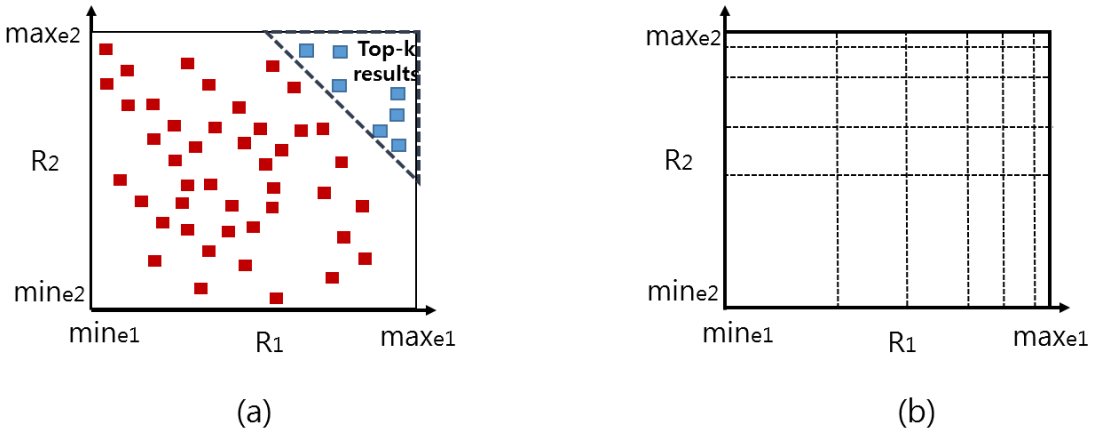


Fig. 3. (a) The monotonic property of a score function guarantees that the top-*k* results are found at the upper-right corner, and (b) increasing the proximity of the boundaries (reducing segment size) of the segments that are nearer to the higher end of the value spectrum assists to more accurately estimate the top-*k* candidate space.

3.2.1 Computing Join Cardinality

Given the number of partitions, each relation can be partitioned into multiple segments in many different ways: the simplest strategy being an equi-width partitioning. However, if we consider that the top-k results would be found at the higher end of the value spectrum due to the monotonic property of a score function, it is easy to see that an equi-width partitioning is not likely to provide the best estimate of the top-k candidate space. Instead, increasing the proximity of the boundaries (reducing segment size) of the segments that are nearer to the higher end of the value spectrum will improve the accuracy of the estimate of the top-k candidate space (**Fig. 3-(b)**).

Let us consider that given a relation R_p, each tuple, t, in R_p has an associated value, $t.e_p \in [mine_p, maxe_p]$. Let us further assume that $mine_p = b_{p,0} < b_{p,1} < \ldots < b_{p,m-1} < b_{p,m} = maxe_p$ are the boundaries used for partitioning R_p into m segments. Here, to ensure smaller segments nearer to the higher end of the value spectrum, the partition boundaries are subject to the following constraints:

$$\forall_{1 \le j \le m-1}\ \alpha \times (b_{p,i} - b_{p,i-1}) = (b_{p,i+1} - b_{p,i}) \ \text{ and } \ \alpha \ge 1.$$

Here, $\alpha = 1$ is an equi-width partitioning scheme, while $\alpha > 1$ ensures the segments closer to the top-k candidate space are smaller than the others (i.e., non equi-width partitioning).

Let $R_{p,i} = (b_{p,i-1}, b_{p,i}]$ be the i-th segment for a relation R_p. Given a top-k join query, let $j_p \in N_p$ be the join attribute of a relation R_p. Then, we build the counting Bloom filter, $f_{p,i}$, for the i-th segment $R_{p,i}=(b_{p,i-1}, b_{p,i}]$ as follows:

1. Let D_p be the set of all distinct values that appear in R_p within the join attribute j_p. If a top-k join query involves n relations (i.e., $R_1, R_2, \ldots, R_n$), a set of all distinct values that appear within the n join attributes (i.e., $j_1, j_2, \ldots, j_n$), each from a different relation, is defined as $D = D_1 \cup D_2 \cup \ldots \cup D_n$. Then, the number of bits in the counting Bloom filter is set to $|D|$.
2. Let $h(\cdot)$ be a one-to-one function that maps each value in D to a value in $[1, |D|]$. Then, the r-th bit, $f_{p,i[r]}$, of the counting Bloom filter $f_{p,i}$ is set to the number of tuples, $t \in R_p$, that satisfy the following condition:
   $$(h(t.j_p) = r) \wedge (b_{p,i-1} < t.e_p \le b_{p,i})$$
   That is, given a segment, the corresponding counting Bloom filter is constructed by counting the number of tuples which belong to that segment.

Example 1. Let us consider two relations, R_1 and R_2, and the top-k join query shown in **Fig. 4-(a)**. Let us assume that the possible maximum and minimum values of $R_1.e_1$ (and $R_2.e_2$) are 0 and 0.9 respectively. Let us further assume that the partition boundaries of $R_1.e_1$ are $b_{1,0}$=0, $b_{1,1}$=0.3, $b_{1,2}$=0.6, and $b_{1,3}$=0.9 (i.e., α = 1 and m=3). Similarly, the partition boundaries of $R_2.e_2$ are $b_{2,0}$=0, $b_{2,1}$=0.3, $b_{2,2}$=0.6, and $b_{2,3}$=0.9.
Since two relations are joined by $R_1.ID_1=R_2.ID_2$, the join attribute j_1 and j_2 correspond to ID_1 and ID_2 respectively. Then, D= {1,2,3,4}, because of D_1 = {1,3,4} and D_2 = {2,3,4}, Thus, the number of bits in the counting Bloom filter is set to 4 (= |D|) .

Let $h(\cdot)$ be a one-to-one function such that $h(1)=1$, $h(2)=2$, $h(3)=3$, and $h(4)=4$. Then, the counting Bloom filters for R_1 are constructed as follows:

$f_{1,1} = 1010, \quad f_{1,2} = 0001, \quad f_{1,3} = 0012.$

Similarly, the counting Bloom filters for R_2 are built as follows:

$$f_{2,1} = 0000, \quad f_{2,2} = 0102, \quad f_{2,3} = 0020.$$

Each counting Bloom filter can be further compressed using various compression techniques, such as Word-Aligned Hybrid (WAH) compression [31]. We note that the counting Bloom filters are constructed based on plaintexts by the client machine and are stored within the database of the client system for later use (as shown in **Fig. 7**). Thus, the construction of the counting Bloom filter incurs a one-time cost for the given sensitive attribute and join attribute.

Assume that the join subspace consists of n segments, $R_{1,u1}$, $R_{2,u2}$, …,$R_{n,un}$, each from a different relation in $R = \{R_1, R_2, , \dots , R_n\}$. The join cardinality of this subspace is computed using the corresponding counting Bloom filters, $f_{1,u1}, f_{2,u2}, \dots, f_{n,un}$, as follows:

$$JC(R_{1,u1}, R_{2,u2}, \dots, R_{n,un}) = \sum_{1 \le i \le |D|} (f_{1,u1}[i] \times f_{2,u2}[i] \times \dots \times f_{n,un}[i])$$

Note that the presented method in this subsection is able to compute the exact join cardinality for an equi-join from a set of relations.

Example 2. Let us continue to consider the example in **Fig. 4-(a)**. Once the counting Bloom filters are constructed as described in Example 1, the join cardinality of each subspace can be computed as shown in **Fig. 4-(b)**. For instance, the join cardinality associated with $R_{1,3}$ and $R_{2,2}$ is computed as followings:

$$JC(R_{1,3}, R_{2,2}) = \sum_{1 \le i \le 4} (f_{1,3}[i] \times f_{2,2}[i]) = 0 \times 0 + 0 \times 1 + 1 \times 0 + 2 \times 2 = 4.$$

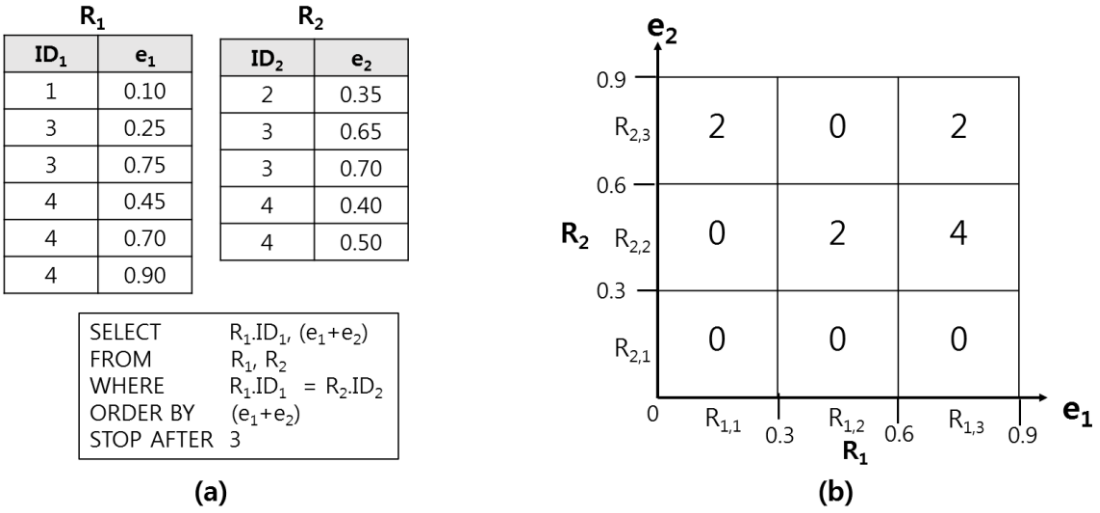


Fig. 4. (a) Two relations, R_1 and R_2, and a top-k join query, and (b) the corresponding join cardinality of each join subspace.

3.2.2 Estimating Top-k Candidate Space

Fig. 5 describes the pseudo-code that estimates the top-k candidate space of a given top-k join query. The algorithm first initializes the set of segments (which corresponds to the top-k candidate space), Set_{cand}, and the cut off score, min_k. The algorithm enumerates all possible join subspaces ($Set_{join\_subspace}$), which are then sorted in descending order ($List_{max}$) by maximum value that those join spaces can have (line 2-3). Here, given a join subspace consisting of n segments, $R_{1,u1}= (b_{1,u1-1}, b_{1,u1}]$, $R_{2,u2} =(b_{2,u2-1}, b_{2,u2}]$, …, $R_{n,un} = (b_{n,un-1}, b_{n,un}]$, and a score function, $f(\cdot)$, of the given top-k join query, the minimum and maximum values that a join result, belonging to this join subspace, can have are computed as followings:

$$min = f(b_{1,u1-1}, b_{2,u2-1}, \ldots, b_{n,un-1}),$$

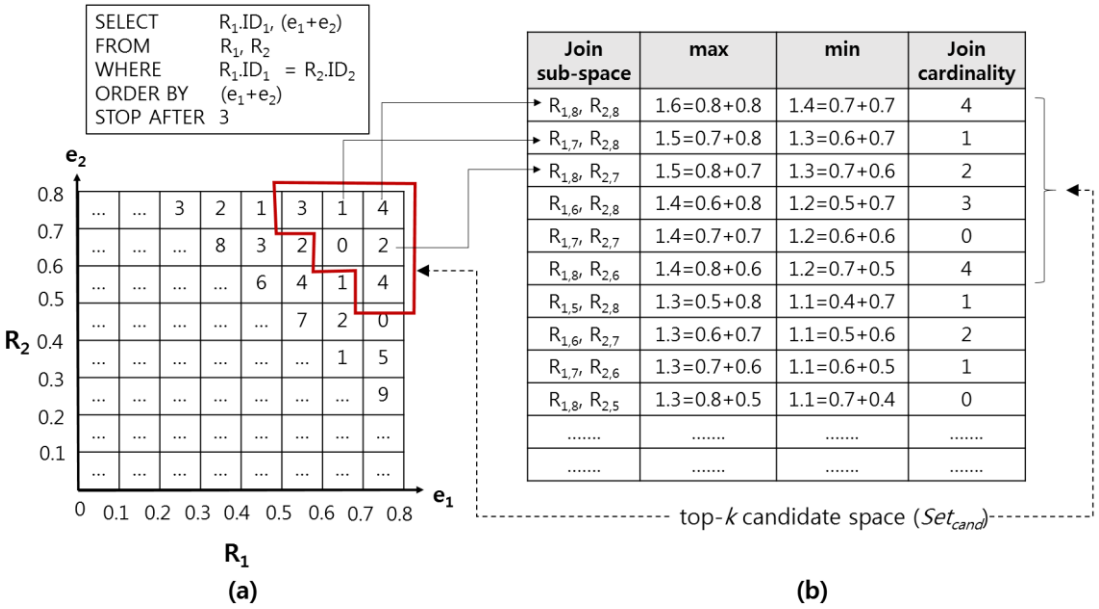
$$max = f(b_{1,u1}, b_{2,u2}, \ldots, b_{n,un}).$$

Each join subspace in $List_{max}$ is visited sequentially and added into Set_{cand}, until we find k join results (line 4-9). Here, the algorithm computes the number of join results of a given join subspace by using the counting Bloom filters presented in Subsection 3.2.1 (line 5). Note that the cut off score, min_k, is set to the minimum value that the join subspaces in Set_{cand} can assume (line 8). For the k join results already identified within the join subspaces in Set_{cand}, we can safely prune those join subspaces that possess a maximum value that is less than the cut off score, min_k (line 10-12). Finally, the algorithm returns Set_{cand} which contains the top-k candidate space.

Input: a set of relations $R = \{R_1, R_2, \cdots, R_n\}$ and a top k
Output: a top-k candidate space Set_{cand}

```
1:  Set_cand ← ∅,  min_k ← 0,  cnt ← 0
2:  Set_join_subspace ← EnumerateJoinSubSpace(R_1, R_2, ··· , R_n)
3:  List_max ← SortByMaxScore(Set_join_subspace)
4:  for each T_cur of List_max
5:      cnt ← cnt + ComputeJoinCardinality(T_cur)
6:      Set_cand ← Set_cand ∪ {T_cur}
7:      if (cnt ≥ k)
8:          min_k ← MinScore(Set_cand)
9:          break
10: for each T_cur of List_max
11:     if (T_cur ∉ Set_cand and MaxScore(T_cur) ≥ min_k)
12:         Set_cand ← Set_cand ∪ {T_cur}
13: return Set_cand
```

Fig. 5. Pseudo-code for estimating the top-k candidate space



Join sub-space	max	min	Join cardinality
$R_{1,8}$, $R_{2,8}$	1.6=0.8+0.8	1.4=0.7+0.7	4
$R_{1,7}$, $R_{2,8}$	1.5=0.7+0.8	1.3=0.6+0.7	1
$R_{1,8}$, $R_{2,7}$	1.5=0.8+0.7	1.3=0.7+0.6	2
$R_{1,6}$, $R_{2,8}$	1.4=0.6+0.8	1.2=0.5+0.7	3
$R_{1,7}$, $R_{2,7}$	1.4=0.7+0.7	1.2=0.6+0.6	0
$R_{1,8}$, $R_{2,6}$	1.4=0.8+0.6	1.2=0.7+0.5	4
$R_{1,5}$, $R_{2,8}$	1.3=0.5+0.8	1.1=0.4+0.7	1
$R_{1,6}$, $R_{2,7}$	1.3=0.6+0.7	1.1=0.5+0.6	2
$R_{1,7}$, $R_{2,6}$	1.3=0.7+0.6	1.1=0.6+0.5	1
$R_{1,8}$, $R_{2,5}$	1.3=0.8+0.5	1.1=0.7+0.4	0
.....	.....	.....	.....
.....	.....	.....	.....

Fig. 6. (a) An example top-*k* join query and the corresponding join cardinalities of join subspaces, and (b) estimates of the corresponding top-*k* candidate space.

Example 3. Let us consider the example top-*k* join query and the corresponding join cardinalities in **Fig. 6**-(a). The table in **Fig. 6**-(b) lists join subspaces with the possible maximum and minimum scores that a join result, which belongs to each join subspace, can have (Here, the score function is defined as the sum of e_1 and e_2). Furthermore, the join subspaces are sorted based on the maximum scores ($List_{max}$ in the algorithm). Then, lines 4-9 of the algorithm perform a sequential scan of the join subspaces in $List_{max}$ until $k = 3$ join results are found. Because the join cardinality of the first join subspace (which corresponds to $R_{1,8}$ and $R_{2,8}$) in $List_{max}$ is greater than 3, by line 9 of the algorithm, the iteration stops after adding the first join subspace to Set_{cand} and setting min_k to 1.4 (which is the minimum score of the first join subspace). Then, by line 10-12, the algorithm finds and adds those join spaces whose maximum score is greater than or equal to min_k into Set_{cand}. As a result, the join subspaces corresponding to ($R_{1,7}$, $R_{2,8}$), ($R_{1,8}$, $R_{2,7}$), ($R_{1,6}$, $R_{2,8}$), ($R_{1,7}$, $R_{2,7}$), and ($R_{1,8}$, $R_{2,6}$) are added into Set_{cand}. This is performed because a join result belonging to these subspaces might possess a better score than min_k. Therefore, the top-*k* candidate space consists of ($R_{1,8}$, $R_{2,8}$), ($R_{1,7}$, $R_{2,8}$), ($R_{1,8}$, $R_{2,7}$), ($R_{1,6}$, $R_{2,8}$), ($R_{1,7}$, $R_{2,7}$), and ($R_{1,8}$, $R_{2,6}$), which are highlighted as red in the figure.

3.3 Top-k Join Processing with the Top-k Candidate Space

Fig. 7 is an overview of the proposed top-*k* join processing algorithm on encrypted cloud data, based on the above top-*k* candidata sapce identification scheme:

1. **Client side**: Given a user's top-*k* join query, the client machine first estimates the top-*k* candidate space, Set_{cand}, as explained within Subsection 3.2. Given the top-*k* candidate space Set_{cand}, let [$b_{1,min}$, $b_{2,min}$, ..., $b_{n,min}$] be lower bounds of this space. Here, $b_{p,min}$ corresponds to the lower bound of the segments of a relation R_p which belong to Set_{cand}. For instance, the lower bounds of the top-*k* candidate space in the example presented in **Fig. 6** are [0.5, 0.5]. Then, the user's top-*k* join query is rewritten by the

client system as follows:

SELECT select-list, e_1, e_2, …, e_n
FROM R_1, R_2, …, R_n
WHERE equi-join-condition(R_1, R_2, …, R_n)
AND $e_1 \geq OPE(b_{1,min})$
AND $e_2 \geq OPE(b_{2,min})$
AND ……
AND $e_n \geq OPE(b_{n,min})$

Here, $OPE(\cdot)$ is the order-preserving encryption function used to encrypt the sensitive data stored in the cloud servers**.** For example, the example top-*k* join query in **Fig. 6** is rewritten as follows:

SELECT $R_1.ID_1$, (e_1+e_2), e_1, e_2
FROM R_1, R_2
WHERE $R_1.ID_1 = R_2.ID_2$
AND $e_1 \geq OPE(0.5)$
AND $e_2 \geq OPE(0.5)$

In other words, unlike the naive method in Section 3, the proposed scheme rewrites a given top-*k* join query such that unpromising tuples in each relation, which are guaranteed not to produce the top-*k* highest scores, are pruned by using a set of range predicates. We also note that since sensitive data stored in the cloud servers are encrypted by using the same order preserving encryption function, the range operation in the rewritten query can be directly applied to the encrypted data without decrypting it.

2. **Cloud side**: The cloud servers process the rewritten query on encrypted data, and send back the results to the client system. Once again, we note that unlike the naive approach, the cloud servers need to compute equi-join results that belong to the top-*k* candidate space.
3. **Client side**: After receiving the results from the cloud servers, the client machine decrypts the encrypted data, e_1, e_2, e_3, …, e_n, ranks the results based on the given score function, and returns the best *k* results to the user.

Unlike the naive solution in Subsection 1.1, the proposed approach improves the performance of the top-*k* join processing as follows: First, by effectively estimating the top-*k* candidate space that contains the best *k* results, the cloud servers are able to prune unpromising tuples that are known to produce non-qualifying top-*k* highest scores at an early phase. This yields a reduction in execution time at the cloud server. More importantly, with the proposed scheme, the cloud servers compute and return only the equi-join results, all of which belong to the top-*k* candidate space, to the client. Thus, there is a significant reduction in the client-side query processing time, due to a reduction in the amount of data decryption and results ranking.

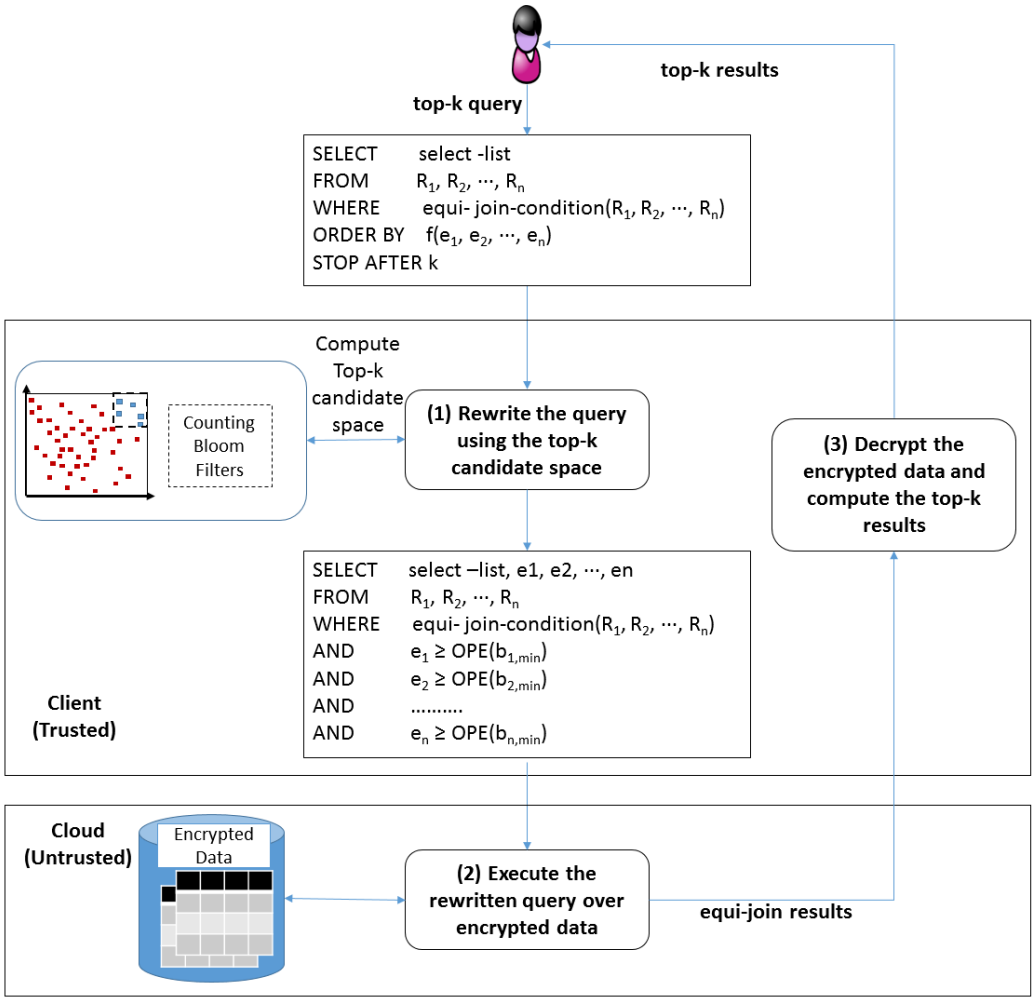


Fig. 7. An overview of the proposed top-*k* join processing approach.

Unlike the naïve approach, the proposed approach restricts the computation of the equi-join results to those that belong to the top-*k* candidate space.

4. Experimental Evaluations

In this section, we experimentally evaluate the performance of the proposed approach. First we describe the experimental setup and then we will discuss the results.

4.1 Experimental Setup

In order to evaluate the proposed approach, we used LINEITEM and PARTSUPP relations from the TPC-H benchmark [32]. The LINEITEM relation contains 6M tuples, and the PARTSUPP relation contains 0.8M tuples respectively. In the experiments, we focused on the two-way (which is the most common type of join) top-*k* join query over these two relations. We report results for the proposed approach (*Pruning*) in Section 3 with varying values of α as well as results from the naive approach (*Naive*) in Subsection 1.1. In support of the proposed approach, we built, for each relation, 100 counting Bloom filters as explained in Subsection 3.2.

4.2 Results and Discussion

Table 2 shows the number of tuples which belong to the top-*k* candidate space for each relation for varying values of *k*. Here, *k* varies from 100 to 100000. This experiment evaluates the first step of the proposed algorithm as described in Subsection 3.3. Recall that the first step of the proposed approach is to rewrite a given top-*k* join query into an equi-join query that can be executable against the encrypted data by the cloud servers. However, unlike the naïve approach, the proposed approach can prune those tuples in each relation that do not belong to the top-*k* candidate space by using a set of range predicates. As can be seen in the table, for each relation, the number of tuples which belong to the top-*k* candidate space decreases, as *k* decreases. Thus, as the value of *k* decreases, more tuples are pruned at an early phase. **Table 2** also illustrates a performance comparison between an equi-width partitioning ($\alpha=1$) specification and a non equi-width partitioning ($\alpha > 1$) specification. As can be seen in the table, a non equi-width partitioning approach can prune more tuples than an equi-width partitioning scheme. This validates the assertion that the top-*k* candidate space is more accurately estimated by increasing the proximity of the boundaries (reducing segment size) of the segments that are nearer to the higher end of the value spectrum.

Table 2. The number of tuples which belong to the top-*k* candidate space as a function of *k*

LINEITEM relation (#tuple = 6M)				
	k=100	k=1000	k=10000	k=100000
Pruning (α=1.00)	7769	22941	106980	425357
Pruning (α=1.10)	2549	11340	68863	366672

PARTSUPP relation (#tuple = 0.8M)				
	k=100	k=1000	k=10000	k=100000
Pruning (α=1.00)	47940	71866	127994	224272
Pruning (α=1.10)	30165	55349	107858	210886

Next, we evaluate the second step of the *Naive* and *Pruning* approaches. **Table 3** shows the number of join results that are produced by executing the rewritten query at the cloud servers. Note that the join results produced by the cloud servers must be sent back to the client so that the client-side query processing can be completed. This client-side query processing includes decryption of data, computing the value of the score function and ranking the results based on the given score function. Hence, a lower number of join results produced by the cloud servers will yield better performance. As can be seen from the table, the behavior of the proposed top-*k* join processing scheme is such that the number of results that are produced by the cloud servers is significantly reduced. This will also reduce the processing overhead requirements of the client machine.

Table 3. The number of join results produced by the cloud servers on varying *k*

	k=100	**k=1000**	**k=10000**	**k=100000**
Naive	24000000	24000000	24000000	24000000
Pruning (α=1.00)	1851	8282	68712	478391
Pruning (α=1.10)	402	3090	37037	388100

Fig. 8 shows the execution times for processing the rewritten query against encrypted data by the cloud servers. In this experiment, we used a cluster consisting of 6 Amazon EC2 nodes in which the rewritten query is executed by Hadoop MapReduce jobs. As can be seen in **Fig. 8** the proposed scheme significantly outperforms the naive solution as measured by execution time. This is because the proposed top-*k* join processing scheme prunes those tuples that are known to produce non-qualifying top-k highest scores at an early phase by using a set of range predicates, which leads to reduced execution times at the cloud servers.

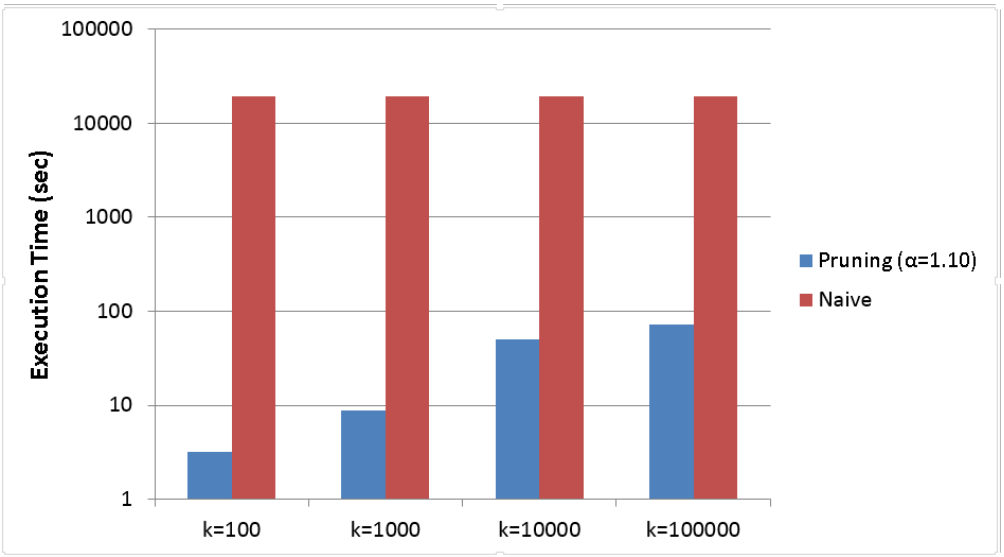


Fig. 8. The execution times for running the rewritten query over encrypted data at the cloud servers

We also study the impact of α on the number of results produced by the cloud servers. In this experiment, α varies in the range of 1.10, 1.15 and 1.20 while k varies from 100 to 100000. As shown in **Table 4**, when k is small, a slightly better result is observed with a higher value of α. On the other hand, when k is large, better results are obtained with a lower value of α. These experimental results imply that a lower value of α is suitable when the number of results returned to users is large (k = 10000, 100000), while a higher value of α is appropriate for the case when the number of results returned to users tends to be small (k = 100, 1000).

Table 4. The impact of α on the number of join results produced by the cloud servers

	k=100	**k=1000**	**k=10000**	**k=100000**
Pruning (α=1.10)	402	3090	37037	388100
Pruning (α=1.15)	369	2858	43844	412442
Pruning (α=1.20)	337	2442	46606	473339

Finally, **Fig. 9** compares the execution times of the third step of the *Naive* and *Pruning* schemes, as the number of results returned to the user (k) varies from 100 to 100000. Note that after receiving the join results from the cloud servers, the client machine must perform the remaining client-side query processing, i.e. result data decryption, results ranking based on the given score function, and returning the best k results to the user. We considered a scenario where the client machine has 3.0 GHz of CPU and 8GB of memory. **Fig. 9** indicates that as the number of results returned to the user decreases, the execution times of the proposed scheme

decrease. The reason for this is that with the proposed top-*k* join processing scheme, more join results are pruned at the cloud side, thereby reducing the number of results returned to the user. As can be seen in the figure, the proposed scheme (*Pruning*) outperforms the naive solution (*Naïve*) in the third step. This is because the proposed top-*k* join processing scheme prunes unpromising join results at the cloud side. This results in a reduction in the overhead resource requirements of the client side. The proposed scheme significantly outperforms the naive solution when the number of results returned to the user is small. Considering that in many applications the value of *k* is typically small, our experimental results are very encouraging.

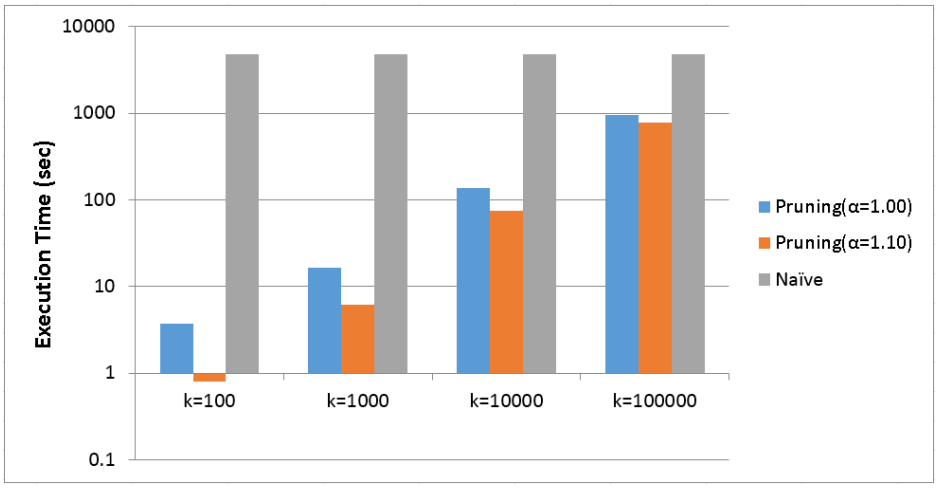


Fig. 9. The execution times of performing client-side query processing for *Naive* and *Pruning* schemes

5. Conclusion

Within the cloud computing environment, data encryption is considered the most effective solution for protecting sensitive data from unauthorized users. However, data encryption imposes a significant amount of overhead during query processing, due to the limitation of directly executing operations over encrypted data. The problem caused by encrypted data becomes more serious when considering top-*k* join queries. In this case, users are often interested in a small number of top results that are produced via the top-*k* join query, rather than the entire result set. Thus, the naive solution which transfers the entire data set from the cloud to the client is highly inefficient. In this paper, we proposed a novel top-*k* join processing algorithm on the massive amount of encrypted data in the cloud computing environments. The algorithm presented in this paper prunes join results which are guaranteed not to produce the top-*k* highest scores at the cloud servers. The experiment results validated that the proposed technique provides significant performance gains over the naive solution.

Acknowledgement

This research was supported by a 2014 Research Grant from Sangmyung University.

References

[1] R. Fagin, "Combining fuzzy information from multiple systems," *Journal of Computer and System Sciences*, 58(1), 89-99, 1999. Article (CrossRef Link).

[2] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, 64(4), pp. 614-656, 2003. Article (CrossRef Link).

[3] Amazon EC2. https://aws.amazon.com/ec2/. Article (CrossRef Link).

[4] Microsoft Azure. https://azure.microsoft.com/. Article (CrossRef Link).

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. of the ACM SIGMOD international conference on Management of data*, 2004. Article (CrossRef Link).

[6] T. Ge and S. Zdonik, "Answering aggregation queries in a secure system model," in *Proc. of the International Conference on Very Large Data Bases*, 2007. Article (CrossRef Link).

[7] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. of the ACM SIGMOD international conference on Management of data*, 2002. Article (CrossRef Link).

[8] R.A. Popa, C.M.S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proc. of the ACM Symposium on Operating Systems Principles*, 2011. Article (CrossRef Link).

[9] S. Tu, M.F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proc. of the International Conference on Very Large Data Bases*, 2013. Article (CrossRef Link).

[10] W.K. Wong, B. Kao, D.W.L. Cheung, R.Li, and S.M. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proc. of the ACM SIGMOD international conference on Management of data*, 2014. Article (CrossRef Link).

[11] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," in *Proc. of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database system*, 2001. Article (CrossRef Link).

[12] S. Nepal and M.V. Ramakrishna, "Query processing issues in image(multimedia) databases," in *Proc. of the International Conference on Data Engineering*, 1999. Article (CrossRef Link).

[13] B. Arai, G. Das, D. Gunopulos, and N. Koudas, "Anytime measures for top-k algorithms," in *Proc. of the International Conference on Very Large Data Bases*, 2007. Article (CrossRef Link).

[14] M. Theobald, G. Weikum, and R. Schenkel, "Top-k query evaluation with probabilistic guarantees," in *Proc. of the International Conference on Very Large Data Bases*, 2004. Article (CrossRef Link).

[15] K. Chakrabarti, V. Ganti, J. Han, and D. Xin, "Ranking objects based on relationships," in *Proc. of the ACM SIGMOD international conference on Management of dat*a, 2006. Article (CrossRef Link).

[16] T. Cheng, X. Yan, and K.C.C. Chang, "EntityRank: searching entities directly and holistically," in *Proc. of the International Conference on Very Large Data Bases*, 2007. Article (CrossRef Link).

[17] N. Ntarmos, I. Patlakas, and P. Triantafillou, "Rank join queries in NoSQL databases," in *Proc. of the International Conference on Very Large Data Bases*, 2014. Article (CrossRef Link).

[18] C. Doulkeridis, A. Vlachou, K. Norvag, Y. Kotidis, and N. Polyzotis, "Processing of Rank Joins in Highly Distributed Systems," in *Proc. of the International Conference on Data Engineering*, 2012. Article (CrossRef Link).

[19] R. Yu, M. Nagendra, P. Nagarkar, K.S. Canda, and J.W. Kim, "Data-Utility Sensitive Query Processing on Server Clusters to Support Scalable Data Analysis Services," *Lecture Notes in Business Information Processing*, 74, pp. 155-184, 2011. Article (CrossRef Link).

[20] N.D. Han, L. Han, D.M. Tuan, H.P. In, and M. Jo., "A Scheme for Data Confidentiality in Cloud-assisted Wireless Body Area Networks," *Information Sciences*, 284, pp. 157-166, 2014. Article (CrossRef Link).

[21] K. Liang, M.H. Au, J.K. Liu, W. Susilo, D.S. Wong, G. Yang, Y. Yu, and A. Yang, "A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing," *Future Generation Computer Systems*, 52, pp. 95-108, 2015. Article (CrossRef Link).
[22] I. Butun, M. Erol-Kantarci, B. Kantarci, and H. Song, "Cloud-centric multi-level authentication as a service for secure public safety device networks, *IEEE Communications Magazine*, 54(4), pp. 47-59, 2016. Article (CrossRef Link).
[23] Q. Xu, P. Ren, H. Song, and Q. Du, "Security Enhancement for IoT Communications Exposed to Eavesdroppers With Uncertain Locations," *IEEE Access*, 4, pp. 2840 – 2853, 2016. Article (CrossRef Link).
[24] W. Wei, X. Fan, H. Song, X. Fan, and J. Yang, "Imperfect Information Dynamic Stackelberg Game Based Resource Allocation Using Hidden Markov for Cloud Computing," *IEEE Transactions on Services Computing*, 2016. Article (CrossRef Link).
[25] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-efficient Adaptive Resource Management for Real-time Vehicular Cloud Services*," IEEE Transactions on Cloud Computing*, 2016. Article (CrossRef Link).
[26] H. Dou, Y. Qi, W. Wei, and H. Song, "Minimizing Electricity Bills for Geographically Distributed Data Centers with Renewable and Cooling Aware Load Balancing," in *Proc. of International Conference on Identification, Information, and Knowledge in the Internet of Things (IIKI)*, 2015. Article (CrossRef Link).
[27] N. Cordeschi, M. Shojafar, and E. Baccarelli, "Energy-saving self-configuring networked data centers. *Computer Networks*," 57(17), pp. 3479-3491, 2013. Article (CrossRef Link).
[28] C. Li, Z. Sun, H. Wang, and H. Song. "A Novel Energy-Efficient k-Coverage Algorithm Based on Probability Driven Mechanism of Wireless Sensor Networks," *International Journal of Distributed Sensor Networks*, 2016. Article (CrossRef Link).
[29] Z. Sun, Y. Zhang, Y. Nie, W. Wei, J. Lloret, and H. Song, "CASMOC: a novel complex alliance strategy with multi-objective optimization of coverage in wireless sensor networks," *Wireless Network*, 2016. Article (CrossRef Link).
[30] A. Boldyreva, N. Chenette, Y. Lee, and A. ONeill, "Order-preserving Symmetric Encryption," in *Proc. of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2009.Article (CrossRef Link).
[31] K. Wu, E.J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *JACM Transactions on Database Systems*, 31(1), 1-38, 2006. Article (CrossRef Link).
[32] Transaction Processing Performance Council. http://www.tpc.org. Article (CrossRef Link).
[33] Microsoft Healthvault. http://www.healthvault.com. Article (CrossRef Link).
[34] M. Li, S. Yu, K. Ren, and W. Lou, "Securing Personal Health Records in Cloud Computing: Patient-Centric and Fine-Grained Data Access Control in Multi-owner Settings," in *Proc. of International ICST Conference on Security and Privacy in Communication Networks*, 2010. Article (CrossRef Link).
[35] K.H. Huang, E.C. Chang, and S.J. Wang, "A Patient Centric Access Control Scheme for Personal Health Records in the Cloud." in *Proc. of Fourth International Conference on Networking and Distributed Computing*, 2014. Article (CrossRef Link).

Jong Wook Kim is an assistant professor of Media Software at Sangmyung University. His research interests include Web data mining, information retrieval, and database systems. Kim has a PhD from the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. He was a software engineer in Teradata Corporation. Contact him at jkim@smu.ac.kr.