Processing OLAP Queries over an Encrypted Data Warehouse Stored in the Cloud

Claudivan Cruz Lopes[1], Valéria Cesário Times[1], Stan Matwin[2], Ricardo Rodrigues Ciferri[3], and Cristina Dutra de Aguiar Ciferri[4]

[1] Informatics Center, Federal University of Pernambuco, Recife, Brazil
[2] Institute for Big Data Analytics, Dalhousie University, Halifax, Canada
[3] Computer Science Department, Federal University of São Carlos, São Carlos, Brazil
[4] Computer Science Department, University of São Paulo, São Carlos, Brazil
{ccl2,vct}@cin.ufpe.br, stan@cs.dal.ca,
ricado@dc.ufscar.br, cdac@icmc.usp.br

Abstract. Several studies deal with mechanisms for processing transactional queries over encrypted data. However, little attention has been devoted to determine how a data warehouse (DW) hosted in a cloud should be encrypted to enable analytical queries processing. In this article, we present a novel method for encrypting a DW and show performance results of this DW implementation. Moreover, an OLAP system based on the proposed encryption method was developed and performance tests were conducted to validate our system in terms of query processing performance. Results showed that the overhead caused by the proposed encryption method decreased when the proposed system was scaled out and compared to a non-encrypted dataset (46.62% with one node and 9.47% with 16 nodes). Also, the computation of aggregates and data groupings over encrypted data in the server produced performance gains (from 84.67% to 93.95%) when compared to their executions in the client, after decryption.

1 Introduction

One of the services provided by cloud computing is often referred to as *Database as a Service (DAS)*, where data management is outsourced to a cloud provider. This allows customers to create, maintain and query their data in the cloud using their internet connection. Because data are stored in the DAS provider, there are potential risks of sensitive data, such as financial information or medical records, being stored in an untrusted host [16]. For security reasons, sensitive data may be encrypted before being sent to the cloud. However, the execution of queries over these data requires decrypting, which often causes high processing costs and can compromise data privacy if this task is performed in an unsafe data provider. Thus, the execution of queries directly over encrypted data is able to significantly improve query performance, while maintaining data privacy [15].

There are several studies in the literature dealing with mechanisms for query processing over encrypted data [1–7, 9, 13]. Also, in many database applications, data are often aggregated, integrated and stored in a data warehouse (DW), in order to be queried in a suitable manner and to help users in increasing the productivity of their decision-making processes. However, little attention has been devoted to the investigation of *how the dimensional data of a DW hosted in a cloud should be encrypted for allowing the processing of analytical queries*. A method for encrypting and querying such a DW is the focus of this article.

In this article, we investigate the development of a method for encrypting and querying a DW hosted in a cloud. To achieve this objective, we introduce in this article the following contributions:

- We describe performance tests that investigate how dimensional data should be encrypted.
- We propose a novel method for encrypting and querying a DW hosted in a cloud, which generates indistinguishable encrypted data (i.e. encrypted values different from each other) and allows the execution of joins between large fact tables and dimension tables, data aggregations, selection constraints, data groupings and sorting operations over the encrypted dimensional data stored in the cloud.
- We introduce an OLAP system based on the proposed encryption method.
- We validate the proposed OLAP system in terms of query processing.

This article is organized as follows. Section 2 surveys related work. Section 3 presents the results gathered from performance tests that investigate how dimensional data should be encrypted. Section 4 proposes a novel encryption method for DW. Section 5 details the architecture of the proposed OLAP system, which is validated experimentally in Section 6. Section 7 concludes the article.

2 Related Work

Several encryption techniques have been proposed to perform computations over encrypted data, enabling query processing directly over encrypted databases. *Symmetric Encryption* [1, 7, 10] and *Asymmetric Encryption* [1] are used to encrypt attributes that are compared by equality operations; *Homomorphic Encryption* [8–9] is applied to attributes that are used in the computation of aggregation functions, such as *sum* and *average*; *Order Preserving Encryption (OPE)* [7, 12–13] and *Bucketization* [2–3] ensure that their encrypted values maintain the same order as their corresponding original values, and are applied to attributes that are used in the computation of *max* and *min* aggregation functions, or attributes that are compared using relational operators such as $=, >, <, \geq, \leq, \neq$. Also, *Multivalued OPE (MV-OPE)* [3–4] is an OPE encryption that produces a probabilistic encryption schema, where unique values from an original dataset are encrypted to distinct encrypted values with a high probability. In this article, these distinct encrypted values are referred to as *indistinguishable encrypted values*.

Based on the aforementioned encryption techniques, data encryption schemas and encryption systems have been proposed. In [4], an MV-OPE encryption schema is proposed but the processing of grouping operations, used in many database applications, has to be done in the client, after decryption. *CryptDB* [7] is a system that enables the processing of SQL queries over encrypted data. However, range constraints and sorting operations are executed over values encrypted by an OPE encryption, which leaks the order of encrypted data and reveals the distribution of the original values since unique original values are encrypted to the same encrypted value [12].

In [6], a data encryption schema is proposed where each database column is encrypted using homomorphic encryption, an indexing mechanism based on MV-OPE and a secure hash function. This approach does not enable the execution of data groupings and sorting operations in the same query, because these computations must be specified over different columns of database tables, while most DBMS require that these operations are specified in a query over the same columns.

In [5], each record is encrypted as a unique value by using a symmetric encryption, and the values involved in selection constraints are given as input to an encryption function to produce indistinguishable encrypted values. Nevertheless, this data encryption schema does not allow the execution of grouping operations over encrypted values because each record in the database is encrypted as a unique value.

In summary, to the best of our knowledge, all the existing methods suffer from particular limitations that constrain their use in practical DW implementations. We propose a novel method that does not suffer from any of these limitations and scales with respect to the number of processors used in its parallelization.

3 Performance Tests

To compute analytical queries over an encrypted DW, there is a need for: joining large fact tables and dimension tables; performing aggregations that are usually based on the sum aggregate function; computing selection constraints (i.e. range/equality constraints); and executing data groupings and sorting operations. To achieve this, we carried out performance tests that investigated the following hypotheses:

- Hypothesis 1. *Primary keys of dimension tables and fact tables, and foreign keys of fact tables should be left unencrypted.*
- Hypothesis 2. *Sum aggregation functions should be calculated directly over the encrypted data stored in the server.*
- Hypothesis 3. *Encrypted DWs can be used in the processing of selection constraints, data groupings and sorting operations.*

For executing the performance tests, we considered a client-server system architecture. The client is responsible for encrypting the data before sending them to the server, mapping the user analytical queries to queries based on the encrypted DW, decrypting the data returned by the execution of analytical queries over the encrypted DW in the server and computing query operations that were not executed in the server. The server provides all DBMS functionalities by accessing the encrypted data. We executed performance tests based on the Star Schema Benchmark (SSB) [11], which is the standard benchmark for the performance evaluation of DW modeled according to the star schema. We executed all SSB queries over synthetic datasets created with scale factor 1, which produced about 6M records in the fact table. We defined the test configurations of Table 1, and for each of them, each SSB query was executed five times, and the averages of the elapsed time (collected in seconds) were calculated. Experiments were conducted on a laptop with 2.8 GHz Core i7-2640M processor, 6 GB RAM, 5400 RPM SATA 1 TB HD, Debian 7.0 64 bits, PostgreSQL 9.2 and JRE7, which played the role of client and server. Network costs were not computed and the encryption algorithms used to encrypt the datasets were implemented in Java.

Table 1. Test Configurations Used in the Experiments

Hypothesis	Test Configuration	Description
1	KEY-ENC	Primary/foreign keys were encrypted by Blowfish [10].
	KEY-NONENC	Primary/foreign keys were kept unencrypted.
2	MEASURE-SYM	Measures were encrypted by Blowfish.
	MEASURE-HOM	Measures were encrypted by the homomorphic encryption proposed in [8].
3	ALL-MVOPE	Descriptive attributes/measures were encrypted by MV-OPE encryption proposed in [4].
	ALL-OPE	Descriptive attributes/measures were encrypted by OPE encryption defined in [13].
	SYM	Descriptive attributes/measures were encrypted by Blowfish.

In investigating Hypothesis 1, we found that the test configuration KEY-ENC (i.e. primary and foreign keys encrypted) presented drawbacks w.r.t. KEY-NONENC (i.e. keys unencrypted). First, the encryption of primary and foreign keys resulted in an increasing of up 500% in their sizes, degrading the performance of the join operation in 33.26% w.r.t. the use of unencrypted keys, as shown in Figure 1(A). Second, when the SSB queries were executed, primary and foreign keys were used only in the computation of joins, which revealed associations between the data items of the tables associated with the joins being processed, independent of keeping the key attributes encrypted or not. In addition, we can assume that primary and foreign keys of dimensional data schemas are often surrogate keys as is highly recommended by the literature [14]. Therefore, keeping these keys unencrypted in a DW provides better performance than encrypting them, and does not affect data confidentiality because they are often composed by artificial values that do not display any semantic information.

To investigate Hypothesis 2, we collected the elapsed time for computing the sum aggregation function over encrypted measures. Results showed that the encryption of measures by using homomorphic encryption (MEASURE-HOM) produced performance gains of 20.54% when compared to measures encrypted through a symmetric encryption (MEASURE-SYM), as depicted in Figure 1(B). This gain occurred because with homomorphic encryption, the sum aggregation function was calculated directly over the encrypted measure values stored in the server, whereas using the symmetric encryption, the sum had to be executed in the client after decryption since the symmetric encryption does not enable the computation of the sum operation over encrypted data. Therefore, results indicated that the computation of aggregations in the server is more efficient than their calculations in the client after decryption.

To investigate Hypothesis 3, we collected the elapsed time for performing selection constraints, data groupings and sorting operations over encrypted descriptive attributes and encrypted measures, since these operations are usually specified over these attributes in analytical queries. Results showed that ALL-OPE had performance gains of 38.91% and 8.69% w.r.t. SYM and ALL-MVOPE, respectively, as shown in Figure 1(C). When compared to SYM, the ALL-OPE's gains occurred because the encrypted values generated by an OPE encryption (such as ALL-OPE) preserved the same order as their respective original values, enabling the execution of the aforementioned operations directly over the encrypted data in the server. On the other hand, by using a symmetric encryption (SYM), only equality constraints were executed over encrypted data in the server, whereas range constraints, data grouping and sorting operations were performed in the client, after decryption. The ALL-OPE's performance gains w.r.t. ALL-MVOPE were modest because encrypted values produced by an MV-OPE encryption (such as ALL-OPE) also preserves the order of their respective original values, and allows the computation of selection constraints and sorting operations over the encrypted data in the server. However, data groupings were executed in the client after decryption, because the encrypted values produced by MV-OPE encryption are distinct from each other.

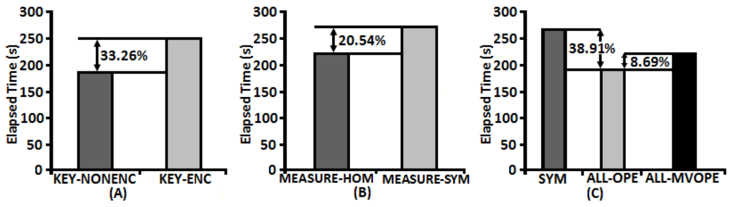

Fig. 1. Query Processing Performance for each Test Configuration

Because a DW is a high-redundant dimensional database, we have that encrypting a DW based on fixed encrypted values (as done by OPE) means a DW with a high degree of redundant encrypted values and a vulnerability that could be explored by statistical attacks [3–4]. However, such vulnerability can be minimized through the use of an encryption technique that produces indistinguishable encrypted values (as performed by MV-OPE). We conclude that MV-OPE is best suited than OPE for encrypting descriptive attributes and measures, since MV-OPE caused a small overhead w.r.t. OPE in the processing of selection constraints, data groupings and sorting operations, and because MV-OPE is more appropriate than OPE to prevent statistical attacks over encrypted data.

4 A Method for Encrypting a DW

Based on the collected results of Section 3, we propose a novel method for encrypting a DW, which is detailed by the Algorithm *Encrypt*. It receives the original value to be encrypted and its respective attribute type as input, and generates as output the respective encrypted value(s). The algorithm tests the attribute type to define the kind of encryption that will be applied over the original value. If the attribute type is a primary key or a foreign key (Line 1), then the original value itself is returned (Line 2). If the attribute type is a measure (Line 3), then two encrypted values are returned: *EncrMeasure1* and *EncrMeasure2* (Line 7). *EncrMeasure1* is obtained by applying a homomorphic encryption over the original value (Line 4), while the *EncrMeasure2* is computed by a MV-OPE encryption that produces an ordered set of integer values for the original value, and an integer value (i.e. encrypted value) is chosen randomly from this set (Lines 5 and 6). If the attribute type is a descriptive attribute (Line 8), then a similar MV-OPE encryption process as outlined before is executed, i.e., an ordered set of integers is generated for the original value (Line 9), and an encrypted value (i.e. *EncrDescriptiveAtt*) is chosen from this set and returned by the algorithm (Lines 10 and 11). To execute selection constraints, data groupings and sorting operations over *EncrMeasure2* and *EncrDescriptiveAtt*, the MV-OPE encryption generated by the Algorithm *Encrypt* satisfies the following properties:

1. *The generation of an ordered set of integer values must be an order preserving process.* This ensures that the generated encrypted values preserve the order of their respective original values, enabling selection constraints and sorting operations be executed directly over the encrypted values.
2. *An encrypted value must be chosen randomly from an ordered set of integer values.* This guarantees that the generated encrypted values are different from each other with a high probability, improving the security of the encrypted DW against statistical attacks because all encrypted values are likely to be indistinguishable.
3. *Different multivalued encrypted values belonging to the same ordered set of integers can be re-encrypted to a unique value.* This is for eliminating the indistinguishability between encrypted values from the same ordered set of integers (as stated in Property 2), ensuring that data groupings are performed over the encrypted values without decrypting them.

```
Algorithm Encrypt
Input: Original Value, Attribute Type
Output: Encrypted Value(s)
1:  if Attribute Type is in {Primary Key, Foreign Key}
2:    return Original Value
3:  if Attribute Type is Measure
4:    EncrMeasure1 = Homomorphic(Original Value)
5:    Set = GenerateOrderedSetOfInt(Original Value)
6:    EncrMeasure2 = ChooseIntFromSet(Set)
7:    return {EncrMeasure1, EncrMeasure2}
8:  if Attribute Type is Descriptive Attribute
9:    Set = GenerateOrderedSetOfInt(Original Value)
10:   EncrDescriptiveAtt = ChooseIntFromSet(Set)
11:   return EncrDescriptiveAtt
```

5 Querying an Encrypted DW Hosted in the Cloud

Our proposed method detailed in Section 4 is used by an OLAP system outlined in this section. The proposed OLAP system allows that analytical queries be processed directly over the encrypted DW without post-processing in the client, and that the analytical query processing makes use of the scalability provided by the cloud. Figure 2 shows the architecture of the proposed OLAP system, which has four main components: *DaaS*, *Scalable Layer*, *Secure Host* and *Client Layer*. The *DaaS* and the *Scalable Layer* are components deployed in the cloud (*server*), and therefore, are considered vulnerable elements, whereas the *Secure Host* and the *Client Layer* are deployed in the user environment (*client*) and are seen as secure elements.

The *DaaS* maintains an encrypted DW whose data are partitioned horizontally among several database management systems *DW-1*, *DW-2*, …, *DW-n*, so that each *DW-i* has an embedded query processor to compute analytical queries. Also, the *DaaS* contains a *DW Master* which is a repository that contains the addresses of each *DW-i*. The address of a *DW-i* is used to open a connection with the *DW-i* in order to execute analytical queries.

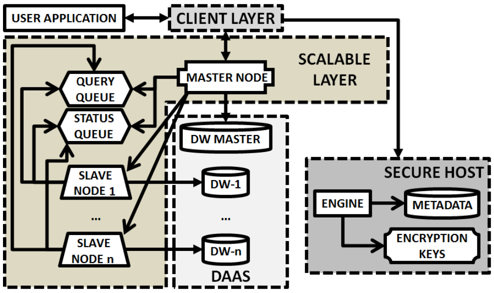

Fig. 2. The Proposed OLAP System Architecture

Client Layer is responsible for receiving user's queries from *User Application* and for sending back the responses of these queries to the *User Application*. The user's analytical queries and their respective answers are built based on the logical data schema of the original DW, ensuring a transparent encryption for *User Application*.

To process these analytical queries over an encrypted DW, *Client Layer* interacts with *Secure Host*, which rewrites and transforms them into queries based on the logical data schema of the encrypted DW. *Secure Host* is also responsible for encrypting query parameters and for decrypting the analytical queries' results received from *Scalable Layer*. For this, *Secure Host* keeps metadata about the logical data schemas of the original DW and of its corresponding encrypted DW, and holds the encryption keys used in the encryption and decryption processes.

Scalable Layer is aimed at providing a scalable mechanism to perform analytical queries over an encrypted DW stored in the *DaaS*. It is composed by a *Master node*, *Query Queue* and *Status Queue*, and several *Slave nodes*. *Master node* is responsible for distributing analytical queries between *Slave nodes*, and for merging the partial results obtained from *Slave nodes* into a merged final result set to be sent to *Client Layer*. *Slave nodes* are responsible for submitting analytical queries to be processed in a specific *DW-i* and for post-processing the retrieved results in order to compute the data groupings. Finally, *Query Queue* is used by *Master node* for sending messages to *Slave nodes* in order to request the execution of analytical queries over the encrypted DW stored in the *DaaS*, while *Status Queue* is used by *Slave nodes* to notify the *Master node* about the ending of an analytical query processing, so that the *Master node* can retrieve the partial results from each *Slave node*.

5.1 Computing OLAP Queries over Indistinguishable Encrypted Data

In detail, *User Application* submits an analytical query to *Client Layer*, which forwards the query to the *Secure Host's Engine* in order to be mapped into an analytical query to be executed over the encrypted DW. To achieve this, the *Engine* searches the metadata for mapping attributes and tables specified in the given analytical query into the corresponding attributes and tables of the encrypted DW. Also, it removes the *group by clause* since this operation cannot be executed directly over the indistinguishable encrypted values stored in the encrypted DW. Next, *Engine* obtains the necessary encryption keys to encrypt each parameter's value defined in the analytical query and generates public keys to be sent to *Slave nodes*. These keys are used by *Slave nodes* for re-encrypting the indistinguishable encrypted values in order to eliminate their indistinguishability and thus, enabling the computation of data groupings (as stated in Section 4). Then, *Engine* sends a set *K = {QRY, G, PuK}* back to *Client Layer*, where *QRY* corresponds to the mapped analytical query (based on the encrypted DW and without data groupings), *G* is the list of attributes specified in the *group by clause*, and *PuK* are the generated public keys.

Master node receives the set *K* from *Client Layer* and sends a request to *DW Master* to obtain the address of each *DW-i* where the analytical query will be executed. The *Master node* then issues a set of messages of the type *{ID, K, ADDRi}* to *Query Queue*, where *ID* is the analytical query identifier, *K* is the set *K* and *ADDRi* is the address of a specific *DW-i*. Further, the *Slave nodes* read the messages queued in the *Query Queue* and send the analytical query *QRY* to the respective *DW-i* to be processed. Each message read from *Query Queue* is dequeued to avoid reprocessing. Further, *Slave nodes* obtain the result set from the *DW-i* and compute the data groupings by executing the Algorithm *ExecuteGrouping*. This algorithm takes the result set, the list of attributes *G* specified in the *group by clause*, and the public keys *PuK* as input, and executes an iterative process (Lines 1 to 8) to generate as output a result set *G'* with the query results grouped by *G* (as specified in the user's analytical query).

For each attribute specified in the *group by clause* (Line 2), the algorithm obtains its respective public key from *PuK* (Line 3) and its respective indistinguishable encrypted value from the record read (Line 4), and uses both to produce a unique encrypted value (Line 5), which is stored in a record of identifiers (Line 6). A unique encrypted value is obtained by mapping the indistinguishable encrypted value selected by a query into a sorted identifier. This identifier is generated for the ordered set of integers to which the indistinguishable encrypted value belongs. Therefore, there is a 1:1 association between ordered sets of integers and their identifiers, and this association eliminates the indistinguishability since indistinguishable encrypted values belonging to the same ordered set of integers will be mapped to the same identifier in a query. Also, a same ordered set of integers is associated with distinct identifiers in different query executions with a high probability because the identifier of an ordered set is randomly generated in different query executions. Further, the original values remain unknown to the server because the generated identifiers do not reveal any information about the original values except their ordering.

Next, the list of measure values are collected from the record read (Line 7) and used together with the identifiers obtained for the attributes in *G* in order to compose the new result set *G'* whose sum of measure values is grouped by *G* (Line 8). A result set *G'* is represented by a hash table *HT* composed by two columns denoted by *k* and *v*. For each row (*k, v*) in *HT*, *k* is the concatenation of each identifier (obtained for the attributes specified in *G*), and *v* is a list of measure values. A new row (*k', v'*) is added to *HT* when the value *k'* is not found in *HT*; otherwise, each value of *v'* is added to the corresponding value of *v* stored in an existing row (*k, v*) of *HT*, so that *k = k'*. When all records in the result set are processed, the new result set *G'* is returned (Line 9).

```
Algorithm ExecuteGrouping
Input: Result Set, G, PuK
Output: Result Set With Data Groupings
1:  for each Record in Result Set
2:    for each Attribute in G
3:      PKey = GetPublicKeyFromPuK(PuK, Attribute)
4:      Indist = GetIndistFromRecord(Record, Attribute)
5:      Id = GenerateOrderedSetOfIntID(Indist, Key)
6:      Save(Id, RecordOfIds)
7:    Measures = GetMeasuresFromRecord(Record)
8:    G' = AddGrouping(G', Measures, RecordOfIds)
9:  return G'
```

In the sequence, each *Slave node* notifies the *Master node* by sending a message to *Status Queue*. A message is composed by *{ID, ADDRsh}*, where *ID* is the analytical query identifier and *ADDRsh* is the address of the *Slave node*. Then, *Master node* reads messages queued in *Status Queue* (each message read is dequeued to avoid reprocessing), and obtains the result set from each *Slave node* identified by *ADDRsh*. *Master node* merges all result sets into a merged final result set by executing the Algorithm *MergeResults* detailed as follows. It receives a result set *G'* produced by a *Slave node* and a merged result set *MRS* as input, and produces a new *MRS* merged with *G'*. For each row (*k', v'*) in *G'* (Line 1), it tries to find a row (*k, v*) of *MRS* so that *k = k'* (Line 2). If a row is found (Line 3), then each measure value of *v'* in (*k', v'*) is added to the corresponding measure value of *v* of the found row (Line 4). Otherwise, the row (*k', v'*) is inserted into *MRS* (Line 6). When all rows of *G'* are processed, the merged result set *MRS* is returned (Line 7).

After merging all result sets, the *Master node* sends the final result set to *Client Layer*, which forwards it to *Secure Host's Engine* for decryption. *Engine* obtains the necessary encryption keys, decrypts the final result set, and sends it back to *Client Layer*. Finally, *Client Layer* delivers the final results to *User Application*.

```
Algorithm MergeResults
Input: Result Set G', Merged Result Set MRS
Output: Merged Result Set MRS
1:  for each (k', v') in G'
2:    Row = FindRowInMergedResultSet(k', MRS)
3:    if Row is found
4:      AddMeasureValues(v', Row)
5:    else
6:      InsertRowInMergedResultSet((k', v'), MRS)
7:  return MRS
```

6 Performance Evaluation

We defined three test configurations: *All Processing in the Server (ALL-SRV)*, which is the test configuration for the OLAP system proposed in Section 5, where all DW's attributes were encrypted according to the encryption method of Section 4, and the homomorphic encryption defined in [8]; *Non Encrypted Baseline (NONENC-BLN)*, where a DW with unencrypted values was considered; and *Post-Processing in the Client (POST-CLI)*, where primary and foreign keys were kept unencrypted, descriptive attributes were encrypted by the OPE encryption defined in [13], measures were encrypted by using Blowfish [10], and the *Secure Host* was responsible for decrypting the final results and computing the sum aggregation functions and data groupings.

Using the benchmark SSB [11], we created datasets with scale factor 1 for each test configuration. The *Client Layer* and *Secure Host* were deployed on a laptop with 2.10 GHz Pentium T4300 processor, 4 GB RAM, 5400 RPM SATA 500 GB HD, Debian 6.0 32bits and JRE7. The *Scalable Layer* was deployed on the Windows Azure using computers with shared CPUs, 768 MB RAM, 20 GB HD and Windows Server 2008 R2. The *DaaS* was deployed on the Windows SQL Azure. The bandwidth network used between the *Client Layer* and the *Master node* was 6Mbps, while the bandwidth network used between the components of the *Scalable Layer* itself and between the components of the *Scalable Layer* and the *DaaS* was 5Mbps. Network costs between the *Client Layer* and the *Secure Host* were not computed. The encryption algorithms were implemented in Java.

Experiments were performed using 1, 2, 4, 8 and 16 *Slave nodes*, so that the SBB datasets were equally partitioned among 1, 2, 4, 8 and 16 *DW-i*. For each test configuration, each SSB query was executed five times, and the averages of the following metrics were collected in seconds: *Query*: average time spent by each *Slave node* to process an analytical query; *Retrieve*: time spent by the *Master node* to collect the query results obtained from each *Slave node* and merge them; *Total Server*: server time (i.e. sum of *Query* and *Retrieve*); *Download*: time spent by the *Client Layer* to retrieve the final results of an analytical query from the *Master node*; *Post Processing*: time spent by the *Secure Host* to decrypt the analytical query results; and finally, *Total Client*: client time (i.e. sum of *Download* and *Post Processing*).

We investigated the impact of *scaling out* on query processing performance by distributing the execution of analytical queries across *Slave nodes* located in the *Scalable Layer* of the proposed system architecture of Figure 2. Using *ALL-SRV* (i.e. a DW encrypted by the proposed method) and *NONENC-BLN* (i.e. an unencrypted DW), we collected the elapsed time for computing all SSB queries. Results showed that the overhead of *ALL-SRV* w.r.t. *NONENC-BLN* ranged from 9.47% to 46.62%, and decreased with the increase in the number of *Slave nodes* (Figure 3(A)). This decrease occurred because with the increase in the number of *Slave nodes*, the workload of each *Slave node* was minimized shortening the average time spent by each *Slave node* to complete the processing of all SBB queries (i.e. *Query*).

Figure 3(B) depicts the values of *Query* collected for *ALL-SRV* and *NONENC-BLN*. For this metric, the overhead of *ALL-SRV* w.r.t. *NONENC-BLN* ranged from 50.11% (with 1 *Slave node*) to 17.72% (with 16 *Slave nodes*). Results showed that when the proposed system was *scaled out*, the use of an encrypted DW did not impair the processing performance of analytical queries, because the overhead caused by the proposed encryption method was reduced when compared to an unencrypted DW.

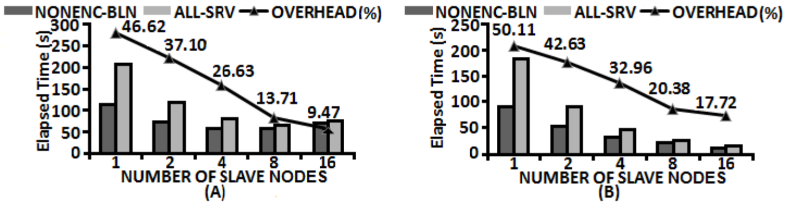

Fig. 3. Query Processing Performance of *ALL-SRV* w.r.t. *NONENC-BLN*

We also compared the elapsed time for computing all SSB queries using *ALL-SRV* (i.e. all query operations were executed in the server) with the elapsed time produced by *POST-CLI* (i.e. the client was responsible for computing sum aggregation functions and data groupings). Figure 4(A) shows that *ALL-SRV* obtained performance gains w.r.t. *POST-CLI* that varied from 84.67% to 93.95%, by ranging the number of *Slave nodes* from 1 to 16. These gains occurred because *POST-CLI* does not compute aggregates and data groupings over the server's encrypted data, and consequently, a large volume of data were transferred from *Slave nodes* to the *Master node*, and from the *Master node* to the *Client Layer*. This increased the workload of the *Master node* that had to merge a greater number of records and the workload of the *Secure Host* that had to decrypt a larger number of records and to compute aggregates and data groupings over the decrypted data records. Figure 4(B) shows that *ALL-SRV* produced performance gains w.r.t. *POST-CLI* of at least: 97.60% in the processing performed by the *Master node* (i.e. *Retrieve*); 97.50% in the time spent for transferring data records from the *Master node* to the *Client Layer* (i.e. *Download*); and 96.20% in the post-processing executed by the *Secure Host* (i.e. *Post Processing*).

We also gathered the elapsed time on query processing distributed between the client and the server. Figure 5(A) shows that *ALL-SRV* produced performance gains w.r.t. *POST-CLI*, which: ranged from 63.21% to 82.13% in the server; and were greater than 97.38% in the client, indicating that the calculation of aggregates and data groupings in the client (after decryption) impaired the overall system's performance on the processing of analytical queries. As shown in Figure 5(B), *POST-CLI* spent from 62.77% up to 77.20% of its runtime in the client, illustrating that the post-processing in the client executed by *POST-CLI* was time consuming. Figure 5(C) shows that *ALL-SRV* spent from 67.33% up to 89.33% of its runtime in the server, because *ALL-SRV* enabled the execution of analytical queries directly over the encrypted data in the server, while the *ALL-SRV*'s remaining time was the time required to transfer the queries' encrypted results to the client, and the time for decrypting these results in the client.

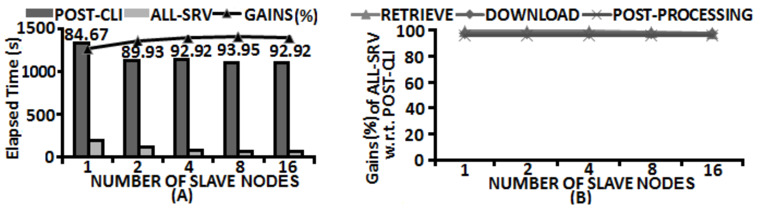

Fig. 4. Query Processing Performance of *ALL-SRV* w.r.t. *POST-CLI*

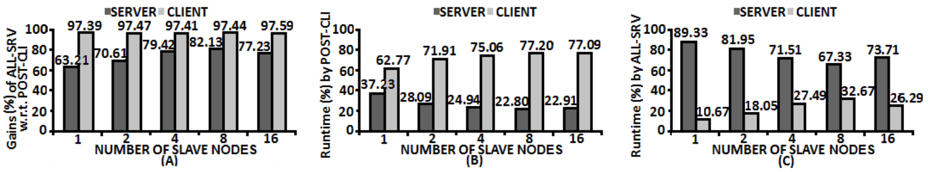

Fig. 5. Distribution of Queries Processing between the Client and the Server

7 Conclusion and Future Work

We proposed a new method for encrypting a DW and enabling the processing of OLAP queries over an encrypted DW that was validated experimentally. Results showed that: primary and foreign keys must be kept unencrypted; the computation of aggregates using measures encrypted by homomorphic encryption favored the analytical query processing performance; and the use of an MV-OPE encryption enables the computation of selection constraints, data groupings and sorting operations over descriptive attributes and measures. Also, we proposed an OLAP system based on the novel encryption method. This system performs analytical queries over encrypted DWs partitioned among several nodes in the DAS. Results showed that the overhead caused by our encryption ranged from to 9.47% (with 16 *Slave nodes*) to 46.62% (with 1 *Slave node*); and that the computation of aggregates and data groupings over encrypted data in the server produced performance gains that ranged from 84.67% to 93.95% when compared to their executions in the client, after decryption. As future work, we intend to evaluate the performance of the proposed OLAP system with different data scalability, and to investigate if data groupings should be computed by the component *DaaS* of the proposed system using indistinguishable encrypted values. A study of the proposed encryption method in terms of security guarantees is another indication of future work.

Acknowledgments. This work has been funded by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under the grants 246688/2012-2 and 246263/2012-1, by the São Paulo Research Foundation (FAPESP) under the grant 2011/23904-7, by the Natural Sciences and Engineering Research Council of Canada, and by the Canadian Bureau for International Education.

References

1. Kadhen, H., Amagasa, T., Kitagawa, H.: A Novel Framework for Database Security Based on Mixed Cryptography. In: Proc. ICIW, Venice, Italy, pp. 163–170 (2009)
2. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure Multidimensional Range Queries over Outsourced Data. The VLDB Journal 21(3), 333–358 (2012)
3. Kadhen, H., Amagasa, T., Kitagawa, H.: Optimization Techniques for Range Queries in the Multivalued-Partial Order Preserving Encryption Scheme. Knowledge Discovery, Knowledge Engineering and Knowledge Management 272, 338–353 (2013)
4. Kadhen, H., Amagasa, T., Kitagawa, H.: MV-OPES: Multivalued-Order Preserving Encryption Scheme: A Novel Scheme for Encrypting Integer Value to Many Different Values. IEICE Trans. Inf. & Syst. 93-D(9), 2520–2533 (2010)
5. Chen, K., Kavuluru, R., Guo, S.: RASP: Efficient Multidimensional Range Query on Attack-resilient Encrypted Databases. In: Proc. CODASPY, New York, USA, pp. 249–260 (2011)
6. Liu, D., Wang, S.: Programmable Order-Preserving Secure Index for Encrypted Database Query. In: Proc. CLOUD, Washington, USA, pp. 502–509 (2012)
7. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: Processing Queries on an Encrypted Database. Commun. ACM 55(9), 103–111 (2012)
8. Castelluccia, C., Chan, A.C.F., Mykletun, E., Tsudik, G.: Efficient and Provably Secure Aggregation of Encrypted Data in WSN. ACM Trans. Sen. Netw. 5(3), 1–36 (2009)
9. Liu, D.: Securing Outsourced Databases in the Cloud. In: Security, Privacy and Trust in Cloud Systems, pp. 259–282. Springer, Heidelberg (2014)
10. Schneier, B.: Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In: Fast Software Encryption, Cambridge Security Workshop, London, UK, pp. 191–204 (1993)
11. O'Neil, P., O'Neil, E., Chen, X.: The Star Schema Benchmark. Online Publication of Database Generation Program (2007), `http://www.cs.umb.edu/~poneil/StarSchemaB.pdf`
12. Wang, P., Ravishankar, C.V.: Secure and Efficient Range Queries on Outsourced Databases using Rp-trees. In: Proc. ICDE, Brisbane, Australia, pp. 314–325 (2013)
13. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order Preserving Encryption for Numeric Data. In: Proc. SIGMOD, Paris, France, pp. 563–574 (2004)
14. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd edn. John Wiley & Sons (2013)
15. Suciu, D.: SQL on an Encrypted Database: Technical Perspective. Commun. ACM 55(9), 102–102 (2012)
16. Adabi, J.D.: Data Management in the Cloud: Limitations and Opportunities. IEEE Data Eng. Bull. 32(1), 3–12 (2009)