Data Fragmentation Scheme: Improving Database Security in Cloud Computing

Amjad Alsirhani, Peter Bodorik, and Srinivas Sampalli

1 Introduction

NIST [1] defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." Cloud computing involves storing data using a third-party or noncentral storage mechanism and requires the ability to access this data from anywhere at any time. It offers many services and includes a variety of models to meet users' needs at affordable prices. One of the cloud computing characteristics is scalability, whereby data is scaled around the cloud providers' servers. The robust devices cloud providers rely on to operate the cloud give users fast network speeds, high performance processing, and a vast amount of storage space.

Despite the perceived benefits of cloud computing, there are still significant security concerns surrounding storing data in the cloud. A standout among these concerns is adequately protecting the confidentiality of sensitive data. This ongoing and highly important concern has motivated us to investigate a means to enhance security in a cloud computing context and to create a viable approach to provide security to data stored in cloud. Cloud computing has many attractive advantages that encourage potential users to consider moving to a modern style of computing.

This is an extended version of a preliminary conference paper that was accepted and presented in ICCA 2017 [2].

A. Alsirhani · P. Bodorik (✉) · S. Sampalli
Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada
e-mail: amjada@cs.dal.ca; bodorik@cs.dal.ca; srini@cs.dal.ca;
http://web.cs.dal.ca/~bodorik/

J. M. Alja'am et al. (eds.), *Recent Trends in Computer Applications*,
https://doi.org/10.1007/978-3-319-89914-5_8

However, these benefits, as Hacg et al. [3] claim, come at a high price, with users facing more significant privacy risks and increased vulnerabilities when they move their, often private or sensitive, data to a cloud. More importantly, concerns arise about data confidentiality, because the data is often not under the direct control of the owner. Consequently, providers can compromise and access sensitive data, which constitutes an invasion of the database owner's privacy. As many cloud computing providers may not even trust their employees, cloud clients find it a challenging assignment to find trusted providers to store sensitive data [4]. Additionally, in some cases, the terms and conditions of cloud computing services are subject to any changes by the providers as they reserve the right to do so. Therefore, the data privacy and confidentiality risk is essential to consider [5].

Considering ongoing security-related issues, encryption rises as the most straightforward solution wherein the data is being encrypted before sending it to the cloud and thus preventing providers from obtaining sensitive information. Unfortunately, however, encrypted data cannot be regularly queried, making it hard for users to retrieve searchable data. Some proposed solutions to this dilemma suggest using asymmetric (i.e., public key) cryptography, where the key is being shared with the cloud providers. Nevertheless, the provider can still infer sensitive information to perform the decryption in response to the client's query.

Similarly, symmetric key (i.e., secret key) cryptography involves decryption on the provider's side, allowing providers to derive sensitive information in this scenario too. Consequently, neither secret key nor public key cryptography offers a suitable solution. Because there is no individual encryption algorithm that can support all Structured Query Language (SQL) queries without decryption, there is a demand for a system that provides users with security while also supporting a variety of query types. This leads us to the following questions: How can we guarantee data confidentiality while using untrusted cloud computing provider resources, and what encryption algorithms can be utilized to support a variety of queries?

2 Related Work

Work regarding the confidentiality of data stored through outsourcing is categorized into four groups: **(a)** fragmentation scheme (a column-based partition), **(b)** hardware-software solution, **(c)** sensitive data vs. nonsensitive data, and **(d)** combination of encryption algorithms. More details about these methods are provided below.

(a) Hacig and Li [6] proposed a fragmentation scheme, which is a column-based partition. On the server side, there are only vertical fragments that are encrypted. A query involving the fragment's data results in fetching the fragment from the cloud to the client and decrypting it, after which the query is executed on the fetched and decrypted fragment. Consequently, this approach is not an applicable solution for this problem because it eliminates cloud computing's main benefit in terms of providing storage. It also requires much overhead

time to execute the query. Other studies have considered this approach such as [3, 7, 8]. They essentially focus on the query optimization technique as a means of attempting to tackle the limitation of [6], which concerns performance.

(b) Bouganim and Pucheral[9] have proposed a combination of hardware and software solution for securing the data in the cloud computing environment. They claim that a standalone software solution is not guaranteed, because of Internet security vulnerability. Their idea is based on a smart card that works as a mediator between the cloud and the user with an assumption of secure communication between the user and the cloud. The smart card is encrypting/decrypting the data at inserting/retrieving time. The cloud has no control over the smart card, so it is only used to store encrypted data. This approach limits the benefit of the cloud computing because the smart cart does the process of encryption and decryption. Therefore, the limited computational power of smart card led to inefficient encryption and decryption performance. Consequently, this solution is not practical for protecting data stored in a cloud.

(c) Anciaux et al. [4] have proposed visible and hidden data concept to ensure data confidentiality. The data is split into two parts (i.e., sensitive data and nonsensitive data). The sensitive data has to be encrypted and stored in private place without public accessibility. A smart USB key mechanism is used to protect private information. In contrast, the nonsensitive data can be stored in the cloud in plaintext. The two parts of the data are joined for querying when the holders of the USB key plug it into their machine while using a distributed method for joining data. This approach has a scalability limitation because of sensitive data storage technique. Moreover, it also limits the benefits of cloud computing by storing sensitive data at the client side (using a smart USB key). Hence, this solution is not practical especially if the system deals with big data. Other studies belonging to this category can be found in [10–14].

(d) Popa et al. [15] have proposed a practical solution to improve the outsourced data confidentiality level from curious cloud providers. Their scheme involves a number of components, including encryption algorithms, proxy, and user's application. The concept behind the combination relies on the case that no existing encryption algorithm can assist all types of queries, so the authors examined encryption algorithms that support queries to be issued to encrypted data. They considered six encryption algorithms that can be used to support the fundamental query structure. Other studies belonging to this category can be found in [16–20].

3 Methodology

We propose a combination of encryption algorithms and a fragmentation technique that together form a novel contribution to this research [2]. The utilized encryption algorithms are described in Sect. 3.1. Figure 1 presents the architecture of the proposed scheme which is a hybrid cloud that mainly consists of two parts: public

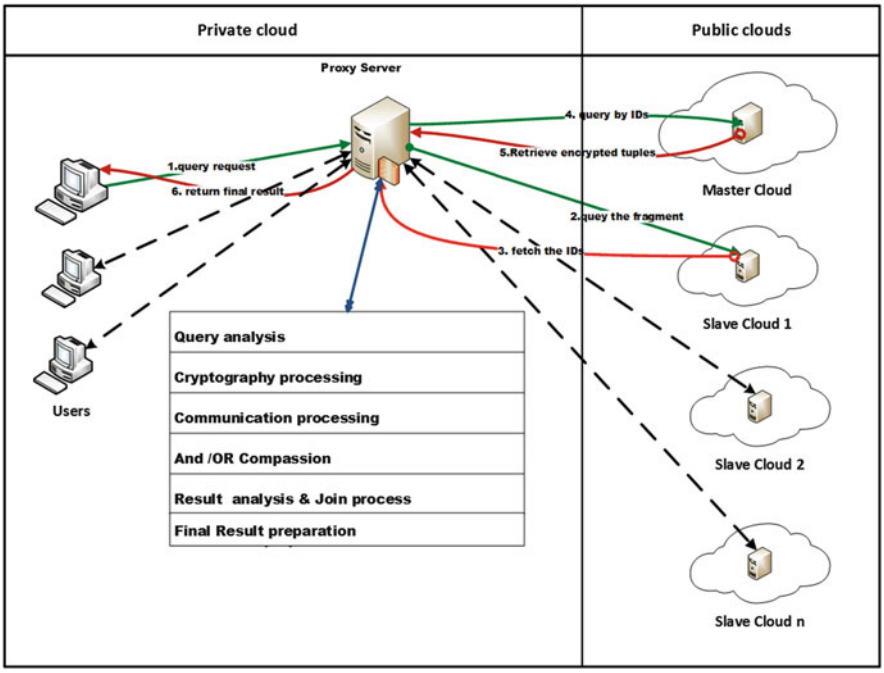

Fig. 1 The architecture of our scheme

clouds and a private cloud. The public clouds include a master cloud, which includes an encrypted copy of the whole database, and N slave clouds, which store extended columns/fragments. It is assumed that both the master and slaves are operated by different cloud providers. The public clouds should not know or be able to infer any information about any other fragments of the database stored on other providers' clouds. The concept is to improve security by concealing this information among the clouds. Of course, this assumption causes problems with scalability regarding the number of columns as ideally each of the columns should be stored on a different cloud offered by a different provider and that the providers do not collaborate. In case of a limited number of clouds, the columns can be stored in the same cloud as long as different columns are encrypted using different encryption algorithms with different keys.

The private cloud consists of a proxy and a client system generating queries. The proxy server is located in the private cloud, center of communication, between the client and the public clouds. There is no direct communication between users and public clouds. All communication has to go through the proxy, which translates any received query from the users to a query or a set of queries that are issued to the public clouds. More details are provided in subsequent sections.

3.1 Encryption Algorithms

The utilized encryption algorithms are the same as those used in the approach described in [15]. They include the Advanced Encryption Standard (AES), order-preserving encryption (OPE), homomorphic encryption (HOM), search, and deterministic encryption. In our scheme, the usage is different due to differences in the architecture between the two approaches. The usage and selection of these algorithms are based upon their ability to support users' queries.

3.1.1 Advanced Encryption Standard (AES) Algorithm

The AES algorithm is used to encrypt the entire database in the master cloud, except for the index column. The index column, a system-generated primary key, is used to identify the tuples of the relation and is replicated in each extended column fragment. A slave cloud contains an encrypted column of the base table together with an unencrypted copy of the index column. The keys for encryption/decryption are stored at the proxy server so that encryption and decryption are performed only within the private cloud. Due to the numerous security advantages of the cipher block chaining (CBC) mode, such as confidentiality, it is our best choice for encrypting the master cloud replica.

3.1.2 Order-Preserving Encryption (OPE) Algorithm

The OPE algorithm is one of the fundamental encryption algorithms used in encrypted relational databases. Most databases need comparison operation between values, particularly numerical ones. With this in mind, the OPE algorithm can be applied to encrypted data to provide a comparison operation Liu et al. [21]. The OPE algorithm is based on linear expression as well as random noise. The encryption algorithm is modeled by Eq. (1), where the coefficients a and b are the keys and v is the value that needs to be encrypted.

$$E(v) = a * v + b + \text{noise} \tag{1}$$

The noise part of the expression, which is added to improve security, is a random number in the range of zero and a coefficient a multiplied by sensitivity. The sensitivity value is 0.01 or 1, depending on whether the value is a double or an integer, respectively. The coefficient "a" can be used to control the range of the noise value. Therefore, the algorithm will produce a different ciphertexts for the same plaintext, depending on the noise, which will enhance security because the pattern will be hidden [21].

3.1.3 Homomorphic Encryption (HOM) Algorithm

The HOM encryption algorithm is needed in most encrypted relation databases due to the mathematical properties in that it can support arithmetic operations, such as multiplication, summation, averages, etc. Queries that include mathematical operations can be applied on encrypted data if the HOM algorithm is used for encryption, and hence it is becoming the solution for applications that require mathematical functionality applied on encrypted data. For instance, the result of multiplication of two encrypted data will decrypt to the sum of their corresponding plaintexts. We calculate the sum of values by using Eq. (2):

$$D(E(a) * E(b)) = a + b \tag{2}$$

3.1.4 Search Encryption Algorithm (RC4)

Word search queries are essential for a string column in a database. Many Database Management System (DBMS) support that kind of queries in the context of plaintext database. However, in context of an encrypted database, there also has to be a way to query an encrypted text. Therefore, a deterministic encryption algorithm is used to encrypt string column so that it can be queried [15]. The algorithm (search) was originally proposed in [22] to search for a string token in document files. It is maintaining the letter position as one of the algorithm's inputs. In [15], the algorithms were adapted to support a word search in the context of databases such as searching for someone's name. Serving the same purpose, RC4 is not only a deterministic encryption algorithm, but it is also fast at encrypting and decrypting [23]. Its properties make it one of the best choices for supporting string searches.

3.1.5 Deterministic Encryption Algorithm (DET)

There are many operating modes in which security and performance differ in the AES algorithm [24]. The ECB mode of the algorithm has the idempotence property, making it possible to perform the equality operation upon encrypted data. This algorithm is used to support equality and join operations [15].

Although this algorithm has the weakness of showing patterns; the confidentiality will improve when the fragmentation technique proposed herein is used. Samarati et al. provide an in-depth analysis about how obtaining data from a single column does not reveal any sensitive information [25].

3.2 Fragmentation Technique

In the following text, we discuss a fragmentation technique for a single database relation; it is intended that it be applied for each database relation that requires

confidentiality. The fragmentation and replication technique involves two aspects: a master cloud and slave clouds (i.e., column-based vertical fragmentation). In the initial configuration, the entire database is encrypted using a highly secure encryption algorithm and stored in the master cloud while not disclosing the encryption key to the master cloud provider. In fact, none of the keys are disclosed to any of the cloud providers. Furthermore, when a relation for the master cloud is created, we ensure that the relation has one column that is used as an index to the tuples of the relation. Thus, in addition to the primary key of the original relation, we create another attribute (table column) containing a dense index, which contains unique values that uniquely identify the tuples of the relation.

The master cloud is the essential part of our scheme as it maintains the entire relation in one place. Hence, the encryption algorithm to be used in the master cloud needs to be secure enough to hold sensitive data. Many studies have shown that AES-CBC is a secure and reliable algorithm for the outsourced storage of sensitive data [26, 27], and, consequently, several studies have used AES-CBC to store sensitive data [28–30]. Therefore, in our scheme, we also use the AES-CBC encryption algorithm to encrypt the master cloud relation. The index column is the only column stored in plaintext so that search result on slave clouds can be used to retrieve resulting tuples from the slave cloud. The index column serves as a candidate key and is replicated in each fragment that contains an extended column of the relation. The purpose of storing an entire relation in the master cloud (and not in the client cloud) is to obtain the highest advantage of cloud computing by not storing data on the client side. As the master cloud stores encrypted data using the AES-CBC algorithm, it cannot be used by itself for search queries, and for most queries both the master and slave clouds will need to be used. However, there are some simple queries that can be answered by querying only the master cloud, that is, queries that retrieve all or some of the columns of all tuples or search queries based only on the unencrypted index column. Table 1 shows some examples of the two kinds of queries that can be submitted directly to the master cloud database without querying the slave clouds.

The configuration continues with the vertical fragmentation to create a number of encrypted replicas of the columns that are then stored in the slave clouds. Furthermore, before a column is stored in a slave cloud, it is encrypted while not disclosing the encryption key to the cloud provider. Whether a table column is replicated and which encryption algorithm is used is determined by the observed

Table 1 Examples of master cloud queries

#	Query	Result
1	Select * From Table	Return all tuples
2	Select *column_name(s)* From Table	Return entire column(s)
3	Select * From Table *Limit value*	Return number of tuple(s) based on the limit value
4	Select * From Table Where index in *(value_1, value_2 , value_n)*	Return tuple(s) based on the index value

or anticipated access pattern to the relation. We shall simply state that if the access pattern to the relation R is such that column A is frequently used by the queries with a predicate $(R.A\Theta k)$, where Θ is an arithmetic relational operator and k is a constant, then the column is replicated and encrypted using an algorithm applicable for querying using Θ. Furthermore, each encrypted column is also augmented with an index column to create an extended column. Each replicated and encrypted column is stored in a slave cloud.

If distinct queries access a column of the relation with different predicates, a column of the relation may appear more than once in a fragment, but encrypted using different encryption algorithms. For instance, if a column is accessed by two different queries with two respective predicates $(R.A = k1)$ and $(R.A > k2)$, then the column $R.A$ would appear in a fragment twice, once encrypted using the DET algorithm for queries using the predicate $(R.A = k1)$ and once encrypted using the OPE algorithm for queries using the predicates $(R.A > k2)$.

3.3 Proxy's Functionality

The proxy is an essential element of our system, as it does most of the processing. It performs the creation, insertion, encryption, decryption, query parsing, and the retrieval and composition of results. Critically, the proxy server has to be within the private cloud and must communicate with the outside world through a highly secure channel, such as Secure Sockets Layer (SSL).

When a proxy receives a client query, it parses it and transforms it into a set of sub-queries on extended columns stored in the slave clouds. As a proof of concept, we implement the process of querying the slave clouds only in serial fashion, which increases the overall delay in comparison if the slaves were queried in a parallel fashion. This effect increases with the increases in the number of predicates as for each predicate a slave is queried, such that the slaves are queried serially, one following another as opposed to in parallel. Utilizing the serial fashion has no performance effect on a query that retrieves the whole relations, that is, a query that has no predicates—only the master cloud storing the whole encrypted relation is accessed.

Each slave returns to the proxy a set of indices forming the answer of the sub-query it has received. When the proxy receives the results of sub-queries, it performs either the union or intersection algorithm on the indexes if there is more than one where condition/predicate in the query's Where clause. If these conditions are separated by OR, the union algorithm is used; otherwise, the intersection algorithm is used. Once the indices of the final result are formed, the proxy issues a query to the master relation to fetch the tuples that match them. The proxy performs all the encryption and decryption. Before inserting any values that come from the user, the proxy encrypts them before storing them in the clouds. Any values appearing in the query also need to be encrypted by the proxy before querying any slave clouds.

4 Implementation and Evaluation

4.1 Cloud Computing Tools and Configuration

As a first step in proving our concept, we created a public cloud computing account at Rackspace, which offers open-source cloud computing [31]. We then created a number of servers equal to the number of fragments that we need, plus one more for the master cloud. Both servers have almost the same configuration. However, the proxy server has different features. Table 2 shows the features of the servers.

4.2 Evaluation Method

We evaluate our method by determining the total delay using an analytical model and through modeling. In the evaluation, we concentrate on the total delay (in ms) per user SQL request as the user will receive the whole query result in one response. Our method is compared to two base methods:

(a) The first is an Unsecure method in which the DB is stored in one cloud, and there are no efforts to provide confidentiality. Thus, data is stored and communicated in plaintext. We refer to this method as the Unsecure approach.
(b) The second method is based on an approach appearing in [15], in which the data is encrypted and stored in one cloud. In that cloud, each column is encrypted using the encryption algorithms that support the desired operation(s). We refer to this approach as the Secure Centralized approach. Our method will be referred to as the Secure Distributed Approach (SDA). Experiments report delays due to communication, crypto processing (encryption/decryption), query processing, proxy delays, and the total delay. We use analytical evaluation and emulation. When emulation is used, each experiment is repeated a hundred times, and we report the average delays.

4.2.1 Major Steps

(a) We first develop an analytical model in which delays are predicted on the basis of various parameters that characterize delays of communication, cryptographic

Table 2 The configuration details

Server	OS	CPU	RAM	HD	Network	Server
Master cloud	Linux	2 vCPUs	4 GB	160 GB	400 Mb/s	XAMPP server
Slaves cloud	Linux	2 vCPUs	4 GB	160 GB	400 Mb/s	XAMPP server
Proxy server	OS X	2.4 GHz Intel Core i5	10 GB	500 GB	150 Mbps	Tomcat server 7.0.53

operations, query processing, and proxy delays. To do the evaluation using the developed analytical model, we need a set of experiments to measure the parameters that characterize the costs/delays. For instance, we measure the average delays for sending messages of various sizes. We then perform evaluation using a set of queries for which delays of user queries are predicted using the analytical method.

(b) We perform the evaluation using emulation in which we build the proxy, store the encrypted and plaintext relation in the master cloud, as well as store encrypted columns of the relation in various clouds for our method. The same queries are used as for the analytical model, and we measure delays for a set of experiments.

(c) Finally, we analyze the results.

4.2.2 Set of Queries

The queries used for evaluation are categorized into three groups as follows:

1. Select statements in which the Where clause contains one simple predicate.
2. Select statements in which the Where clause contains a conjunction of two predicates (connected by the AND Boolean operator).
3. Select statements in which the Where clause contains a conjunction of three predicates (connected by the AND Boolean operator).

4.3 Evaluation Using the Analytical Model

4.3.1 Analytical Model

Processing a query incurs the following costs/delays: communication delay, crypto (encryption/decryption) delay, query processing delay, and proxy delay.

Communication Delay

We model the communication delay of a message transfer by using the generally accepted Eq. (3):

$$D_s = a + bx \tag{3}$$

The delay is a linear function of a set-up overhead and the size of the message. As we are using Internet for communication, we are not able to determine the maximum size of packets, and hence we model communication on a message basis, as opposed to a more detailed modeling in which a message is sent in packets if it exceeds

the maximum packet size. The communication delay is affected by the propagation delay, serialization, data protocol and latency, routing and switching latencies, and queuing and buffer management, and each of these factors has an impact on the total delay [24]. Since the messages are going over Internet, however, we have no control or real knowledge on the individual delay components and hence resort to a simple model represented by Eq. (3):

Crypto Delay

The crypto delay is a delay for encryption/decryption and is modeled by Eq. (4).

$$D_c = cn \tag{4}$$

Crypto delay is directly proportional to n, where n is the number of items to be encrypted/decrypted and c is the cost of encrypting/decrypting one item.

Query Processing Delay

Under the general term of query processing delays, we include delays for the basic SQL operations of Insert, Delete, Update, and Select.

1. **Insert**
   It depends on the number of tuples to be inserted, so the time complexity to insert into a database is $\mathcal{O}(n)$, where the n is the number of tuples.
2. **Delete/Update/Select**
   To find the tuples affected by the operation, the tuples need to be identified, and thus the delay depends on finding the tuples. Before we provide equations, we note that the delay to select/find tuples of a relation depends on whether there is a fast access data structure that can be exploited to find the desired tuples.

   (i) If there is no fast access method, all tuples are scanned, and hence the delay is directly proportional to the number of tuples in the relation, which is modeled by $p\,n$ where n is the number of tuples in the relation and p is the delay to handle one tuple. If there is a fast access method, then the delay is proportional to $\log n$, where n is the number of tuples, and it is modeled as $p \log n$. Once the tuples are identified, they are processed as per delete, update, or select operation that we assume causes equivalent/same delays. However, this processing delay is directly proportional to the number of affected tuples. We need to note that the above simplifications are reasonable only under the assumption that the select queries are one variable only, that is, that they do not involve a join.

   (ii) As we deal with an evaluation in which we only have a single relation, we assume that the query does not specify a self-join. Furthermore, the equation also does not properly represent delays for an SQL query that has a group-by operator that would result in sorting of resulting tuples.

Since our experimental relation is relatively small-sized, we do not build indices explicitly, and we model the Delete/Update/Select by Eq. (5) under the assumption that there is no fast access method available and hence the processing delay is directly proportional to the number of tuples in the relation:
one tuple.

$$D_p = c + pn + psn \tag{5}$$

In Eq. (5), c is the overhead delay, n is the number of tuples in a relation, p is the delay to process one tuple, and s is the selectivity factor. The delay to process a query is thus modeled by overhead delay, c; delay of $p\,n$ for scanning all tuples of the relations for those tuples that satisfy the predicate, which could be a conjunction of simple terms; and delay of $s\ p\,n$ to handle the result.

Proxy Delays

The proxy is playing an important role as it performs several tasks that include encryption, decryption, query parsing, and key management. The proxy parses the query and rewrites it into another query or queries depending on to which slave sub-queries need to be sent. A number of sub-queries will be issued if there is a conjunction of simple terms. The time complexity for rewriting a query is modeled as $\mathcal{O}(1)$. Although there are a number of slaves, the number is small and fixed.

The proxy does both the encryption and the decryption, so the delay is already modeled within a separate section dealing with crypto delays. However, the key management is one of the proxy duties, so its delay needs to be included in the proxy overhead delay. The time complexity is equal to a search in an array by the element's position, which is $\mathcal{O}(1)$ because the keys are chosen based on their position in the array. On the other hand, the proxy must determine which encryption algorithm is needed to encrypt the query(s) value. Since we have a fixed number of the encryption algorithms, the delay to determine the encryption algorithm is also $\mathcal{O}(1)$.

The proxy also deals with the unions and intersects as more than one slave cloud may be queried in our method. The retrieved results of each slave cloud are stored in an array at the proxy. The proxy performs the union and the intersect upon the two arrays. In case there are more than two slave clouds accessed in processing a query, the result of intersect/union of two arrays will be stored on a temporary array where the intersect with a third array can be performed and so on. Thus, the delay for a union is $\mathcal{O}(n + n)$, and the intersect operation is $\mathcal{O}(m \log n)$ where m is the size of the first array and n is the size of the second array. Equation (6) is used to model the proxy's delay:

$$D_x = ax + by + c \tag{6}$$

In Eq. (6), the variable x is the number of elements in the first array, y represents the size of the second array, and a and b are constants representing the delay to process a tuple in array with the sizes x and y, respectively.

4.4 Delays Derived Using the Analytical Model

We are now ready to report on delays predicted by the analytical model. We explore two scenarios for our Secure Distributed method, one is serial and another is parallel. That is, if a user query results in more than one sub-query sent to the clouds in which columns are stored, we analyze two cases: (a) The sub-queries are issued in a serial fashion, one after another, and (b) sub-queries are issued in parallel, that is, concurrently. Once the results from sub-queries are retrieved, the proxy determines the intersection of tuples of the results. Based on the returned indices, this intersection is then sent to the master to get the query answer. We report on delays obtained by applying the analytical model on the queries described in Sect. 4.2.2. Recall that our proposed method has two variations, one serial and one parallel, that are compared to two base methods, one being Unsecure and one being Secure Centralized. As the name indicates, in the Unsecure method, there is no security, and hence the SQL queries are issued against the plaintext Databased (DB) table from which answers are derived in the usual manner. The Secure Centralized method is as described in [15] in which the whole DB is encrypted and stored in the cloud. Each column of a table is encrypted using an algorithm that supports the anticipated predicates. We calculate the total delays of each of these approaches using the analytical model while varying the selectivity to vary the number of retrieved tuples and the number of predicates.

Table 3 shows the total delays in milliseconds for queries 1, 2, and 3. As query 1 has only one predicate, there is no difference in delays between the parallel and serial versions of the Secure Distributed method. The delays for the Secure Distributed (serial and parallel) methods are higher than for the Secure Centralized method because in the centralized case only the master cloud is accessed, while the Secure Distributed method, in addition to accessing the master cloud, also accesses one slave cloud. However, the Secure Distributed method provides more security due to partitioning and distribution of the data. The selectivity factor of the predicate is varied from 0.2 to 1.0 in steps of 0.2. For each selectivity factor, there are component delays, namely, communication, query processing, crypto, and proxy, as is appropriate, these are also reported. Recall that in the Unsecure Centralized method, a query is processed in the cloud by scanning all tuples of the table. As the query result size is determined by the selectivity factor and the table is reporting delays per selectivity factor, there is no difference in delays for individual queries in this method, and hence, in Fig. 2a, we show the delays for the Unsecure Centralized method only once, that is, we do not show the delay for each individual query. Furthermore, the Unsecure method does not have the crypto component as the table is stored in the cloud in plaintext. Same statements apply for the Secure Centralized

Table 3 The total delays in milliseconds for queries 1, 2, and 3 using the analytical model

	Query 1					Query 2					Query 3				
S.F	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
Unsecure Centralized	130	160	186	211	237	134	160	186	211	237	134	160	186	211	237
Secure Centralized	339	569	798	1028	1258	339	569	798	1028	1258	339	569	798	1028	1258
Secure Distributed serial	445	674	904	1134	1363	536	845	1008	1244	1479	634	879	1108	1350	1585
Secure Distributed parallel	445	674	904	1134	1363	469	740	931	1163	1396	497	729	960	1111	1425

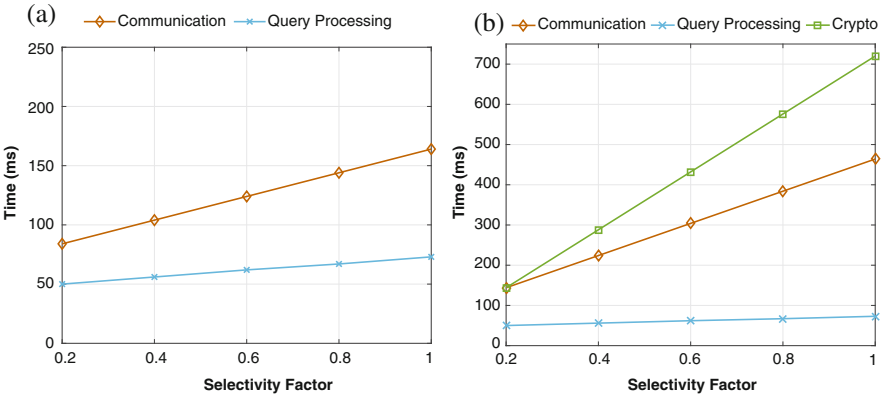

Fig. 2 The delays in milliseconds of (**a**) and (**b**) schemes for queries 1, 2, and 3. (**a**) Unsecure Centralized. (**b**) Secure Centralized

method, except that the method does have a crypto delay component. In that method, once the query is shipped from the proxy to the cloud, the query execution scans all tuples of the relation to identify the tuples that satisfy the predicate—hence there is no difference in delays in execution of queries in the cloud, and the query result size is determined by the selectivity factor—hence, in Fig. 2b we show delays only once and not for each individual query.

In Unsecure Centralized approach, unsurprisingly, the communication delay is higher than the query processing delay, and, of course, it has the smallest delays of all methods. The Secure Centralized method has higher delay than the Unsecure Centralized method, but the delay is smaller than those of the variants of the Secure Distributed method. As expected, for all methods, increase in the query complexities due to an increase in the number of predicates causes increase in delays. For the Secure Centralized method, in which the relation has encrypted columns in one cloud, there are communication delays to access the cloud, query processing delays, and crypto delays. For the Secure Distributed serial and parallel variants, there are four component delays: communication delays to access the master and the slave clouds, crypto delays, query processing delays on slaves, and proxy delays.

Figures 3, 4, and 5 show the component delays in milliseconds for queries 1, 2, and 3. Recall that a query i has i predicates. Delays for the parallel variant of the Secure Distributed method are less than those of the serial variant. It may also be observed that the crypto delays, although less than the communication delays, are significant. Further discussion of evaluation is offered below in Sect. 4.5.

4.4.1 Delays Derived Using Emulation

We measure the total delays while varying the selectivity factor and the number of predicates in the Where clause. We start the experiments with a query that has

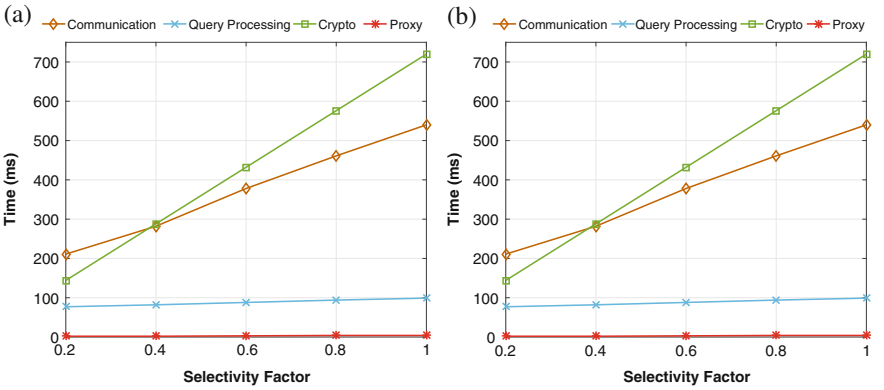

Fig. 3 The delays in milliseconds of (**a**) and (**b**) schemes for query 1. (**a**) Secure Distributed serial. (**b**) Secure Distributed parallel

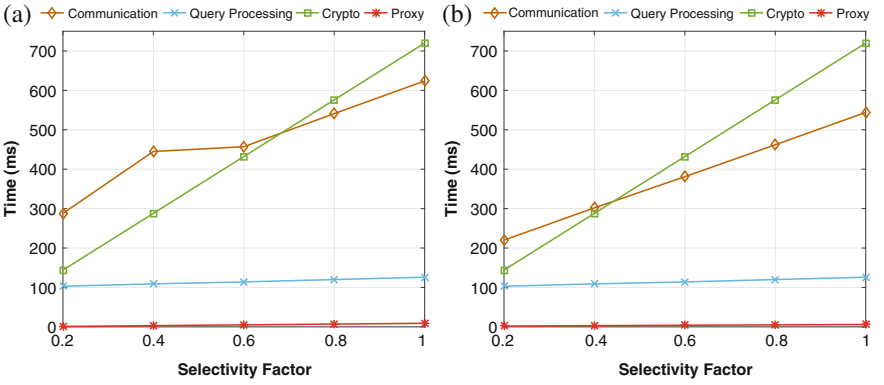

Fig. 4 The delays in milliseconds of (**a**) and (**b**) schemes for query 2. (**a**) Secure Distributed serial. (**b**) Secure Distributed parallel

a Where clause with one predicate and then we increase the number of predicates by one up to three predicates. We submit the queries a hundred times and report the averages. The experiments are applied for the Unsecure Centralized, Secure Centralized, and the serial variant of the Secure Distributed method. Table 4 shows the total delays in milliseconds for the three methods for queries 1, 2, and 3, respectively. Of course, the Unsecure method has shorter delays not only because there is no decryption process after the result is retrieved but also because the information is stored in clear text, which means that the tuples are smaller in size, resulting in less communication volume and hence delays.

Figure 6 shows the component delays in milliseconds for the three approaches for query 1, which has one predicate. The higher communication delay for the Secure Distributed method, in comparison to the Secure Centralized method, is due to the

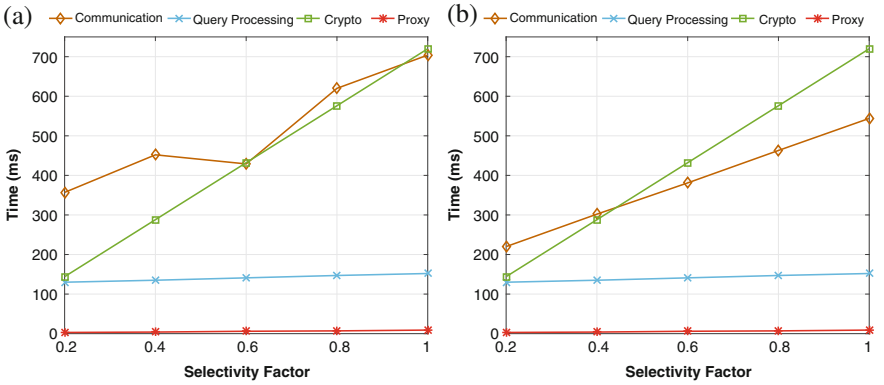

Fig. 5 The delays in milliseconds of (**a**) and (**b**) schemes for query 3. (**a**) Secure Distributed serial. (**b**) Secure Distributed parallel

communication delays of the extra round trip. It should be noted that the scale of y-axis values is not the same in Fig. 6a–c.

Figure 7 shows the delays for query 2 that has two predicates. It shows that the communication delay is much higher for the Secure Distributed method due to communication to process each predicate, which is done in a serial fashion. Figure 8 shows the component delays for query 3 with three predicates. The communication delay becomes the dominant factor for the Secure Distributed approach, increasing with the increase in the selectivity factor.

4.5 Result Analysis and Discussion

In the earlier section, we reported the component delays for each approach. In this section, we will compare the delays of each component of the analytical model with the corresponding delays of the emulation modeling across all queries. Figure 9 shows the communication delays, derived through the analytical modeling and emulation for all queries for Secure Distributed approach. In the Unsecure and Secure Centralized approaches, the communication delays are increased mostly due to the increases of the selectivity factors that produce larger results/relation that need to be communicated over the network. In our scheme, however, the communication is further increased as the number of predicates increases, as there is a sub-query issued for each predicate—hence increase in communications.

The differences in delays between the analytical model and the emulation model are higher in query 3 than in queries 1 and 2, and the trend increases with the increase in the size of data that needs to be processed and communicated, that is, with an increase in the selectivity factor. Clearly, analytical modeling

Table 4 The total delays in milliseconds for queries 1, 2, and 3 using emulation evaluation method

	Query 1					Query 2					Query 3				
S.F	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
Unsecure Centralized	130	115	143	152	171	153	160	185	200	210	155	175	207	273	220
Secure Centralized	451	585	804	983	1287	493	710	884	972	1582	504	660	775	1194	1441
Secure Distributed serial	777	1147	1492	1757	2073	976	1459	1707	2165	2223	1196	1729	1950	2291	2763

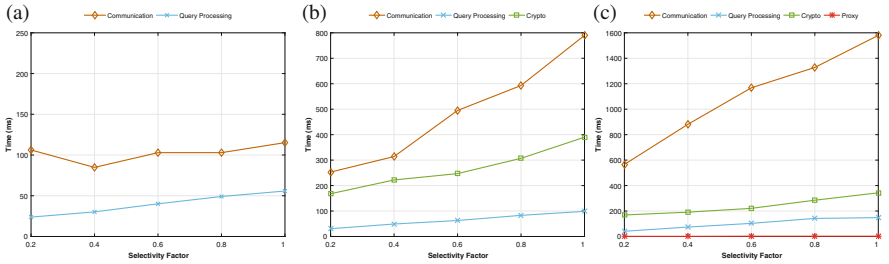

Fig. 6 The delays in milliseconds of (**a**), (**b**), and (**c**) schemes for query 1. (**a**) Unsecure Centralized. (**b**) Secure Centralized. (**c**) Secure Distributed serial

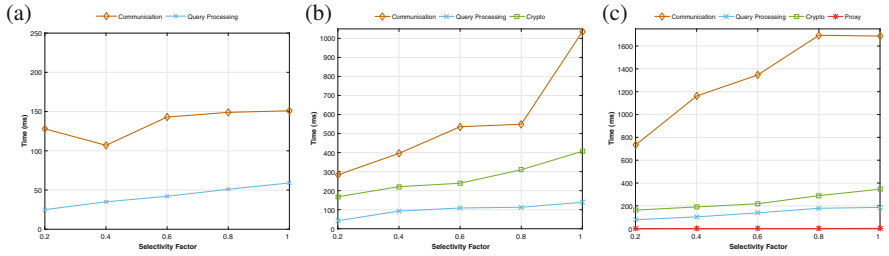

Fig. 7 The delays in milliseconds of (**a**), (**b**), and (**c**) schemes for query 2. (**a**) Unsecure Centralized. (**b**) Secure Centralized. (**c**) Secure Distributed serial

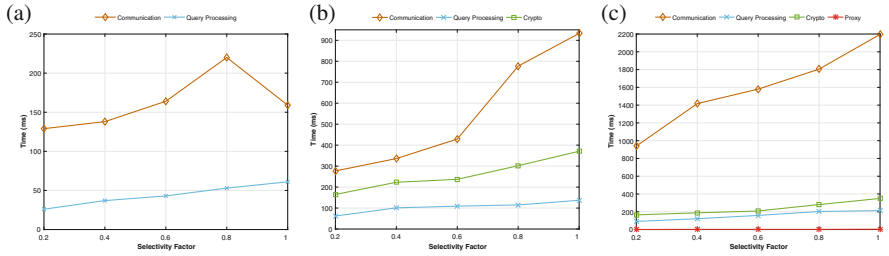

Fig. 8 The delays in milliseconds of (**a**), (**b**), and (**c**) schemes for query 3. (**a**) Unsecure Centralized. (**b**) Secure Centralized. (**c**) Secure Distributed serial

makes simplifying assumptions that are not necessarily reflected in real operations. However, the trends are correctly predicted in the analytical method.

Figure 10 shows the query processing delays, derived through the analytical modeling and emulation, for all queries for Secure Distributed approach. It can be observed that for small selectivity, delays through analytical modeling are higher than those obtained through emulation. However, for higher selectivity, delays obtained via emulation are much higher than those obtained by analytical modeling.

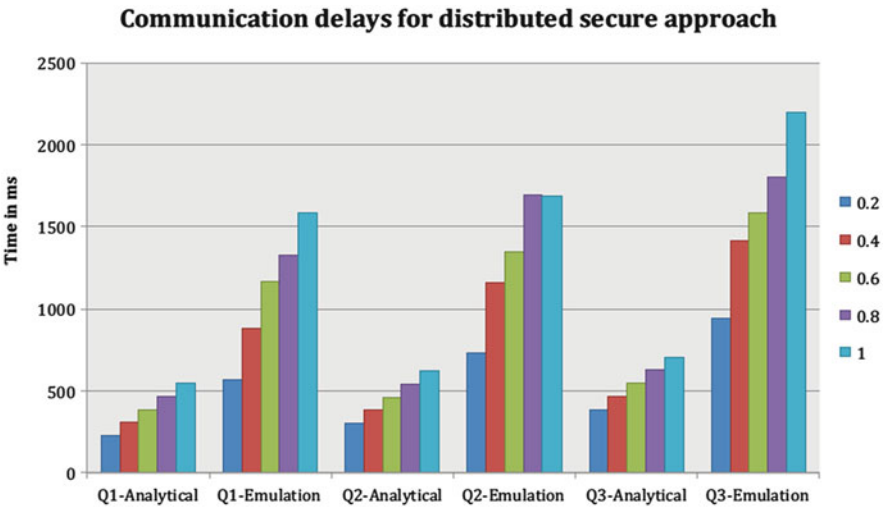

Fig. 9 Communication delays for the Secure Distributed Approach

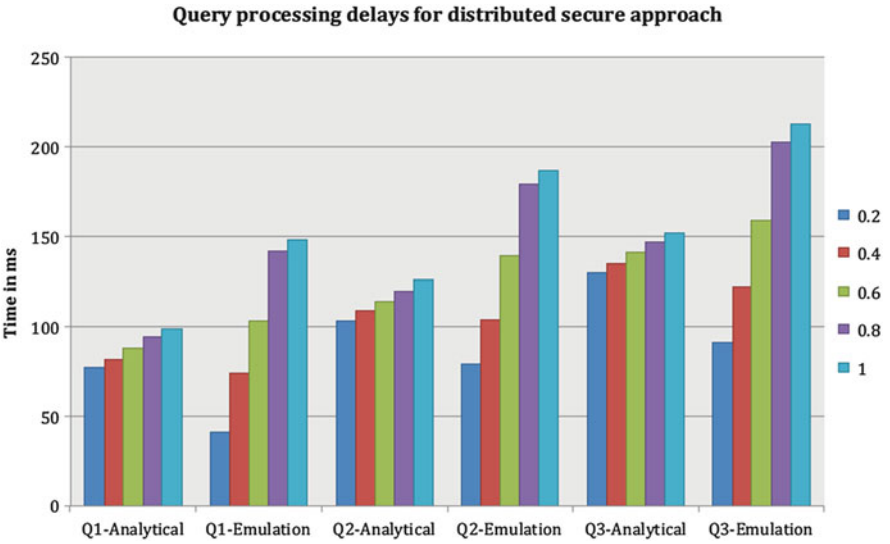

Fig. 10 Query processing delays for the Secure Distributed Approach

Figure 11 shows the crypto processing delays, derived through the analytical modeling and emulation, for all queries for the Secure Distributed approach. The crypto delays are directly proportional to the size of data to be encrypted or decrypted. Recall that the decryption process is needed only for the final result retrieved from the master cloud, and therefore, the measurement of the

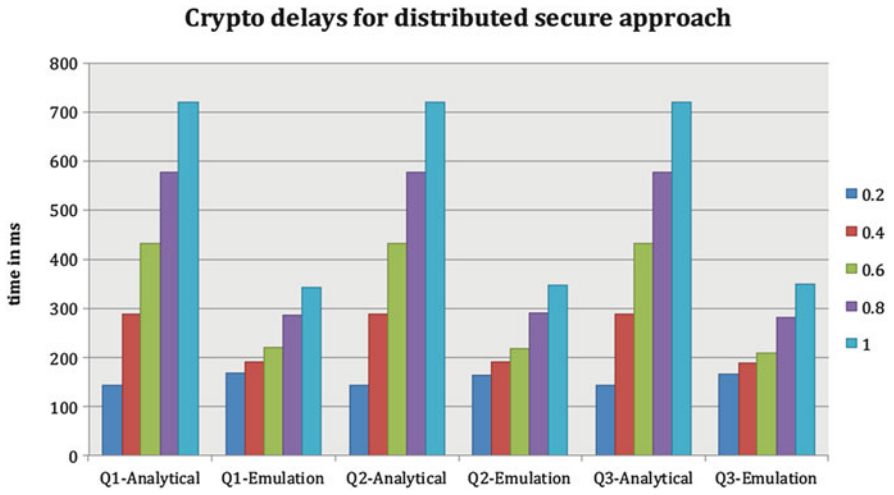

Fig. 11 Crypto delays for the Secure Distributed Approach

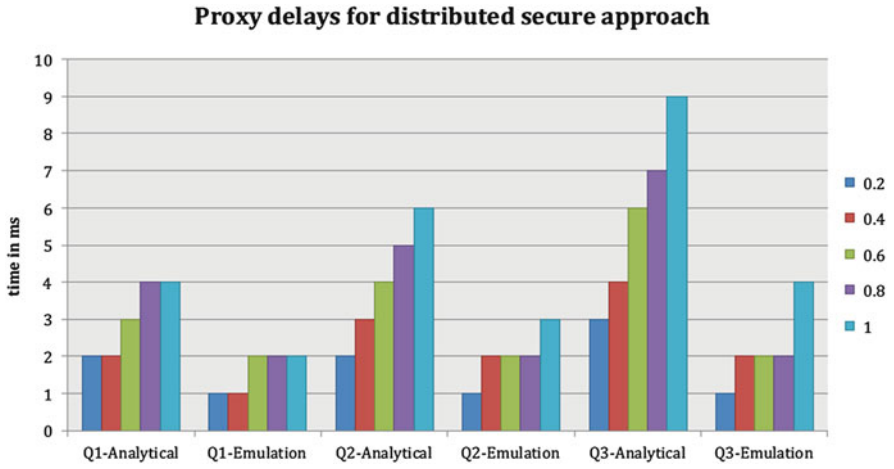

Fig. 12 Proxy delays for the Secure Distributed Approach

decryption operation depends on the selectivity factor. Thus, the delays obtained by the analytical model are the same for both the Secure Centralized and Secure Distributed approaches. The same applies for the delays derived through emulation.

Figure 12 shows the proxy processing delays, derived through the analytical modeling and emulation, for all queries for the Secure Distributed approach. Recall that the proxy delays include delays to perform the intersection of results returned from slave clouds and delays associated with transforming the user query to sub-queries on clouds as well as the processing results obtained from the clouds. In general, the proxy delays in both models (i.e., analytical and emulation) are much

less than the delays of the other components. The disparities in the delays between analytical and emulation approaches for the proxy delays are much lower than the disparities in the other components.

4.6 Discussion on Analytical Versus Emulation Modeling

We observe that the delays derived from analytical modeling, except crypto delays, are smaller than those derived from emulation. In the analytical model, we determine the component delays separately. Thus, the sum of an individual component delays forms the total delay of the analytical model. The analytical model is driven by first calibrating the models' parameters through measurements of average delays in communication, query processing, and decryption and then using such parameters in analytical modeling to predict the delays of the methods under consideration. When making the measurements, we use communication over the Internet and measure decryption and query processing delays that are performed on infrastructure provided by the cloud. Delays due to emulation are also obtained by using the cloud infrastructure for query processing and decryption, the Internet for communication, and thus, variability in both the analytical and the emulation approaches can be expected because both depend on when measurements are being done and the load on the environment when measurements are being done. Although this variability in performance is obvious for Internet delays, it also applies to measurement of the other components as the processing is performed in the cloud. Delays in cloud infrastructure are also variable as they depend on the load, resources allocated by the cloud provider, and the cloud's quality of software that is used to measure the load and to allocate resources to handle it.

To conclude this section, although analytical modeling underestimated delays in comparison to emulation, it is useful as it showed the general trend in delays and thus is well suited to provide guidelines on the general trend when variables, such as selectivity or type of issued queries, are varied.

5 Conclusion

Cloud computing technology is attractive for storing and managing data. However, concerns over confidentiality concerning storing sensitive data prevent many public and commercial organizations from moving to the cloud. A number of researchers have attempted to provide solutions to the security concerns associated with outsourcing storage. The aim of this research was to investigate how to prevent untrustworthy cloud providers from obtaining sensitive data. We proposed a combination of encryption algorithms in [15] and obfuscation by distributing data among different clouds for confidentiality of storing data in the cloud. Specific encryption algorithms provide the user with strong confidentiality but when

querying of encrypted data is required than specific encryption algorithms, which support querying of encrypted data, may have to be used even though they may not be strong enough for certain type of attacks. However such attacks may be thwarted by through obfuscation by distribution of data. We built a prototype for our proposed method to demonstrate its feasibility, determine overhead delays, and compare it to base methods reported in the literature.

We used two modeling techniques to determine the delays of our proposed method and the base methods used for comparison. One method was based on an analytical model, while the other method used emulation using a prototype. We calibrated our analytical models' parameters by measurements. When we compared delays derived through analytical modeling to those derived through emulation, we observed that in most cases the analytical model underestimated the delays. Thus, the analytical model can be used to analyze the behavior of the system regarding the trends on delays, but it cannot be used to predict such delays accurately. Of course, the same can be applied for any method that is predicting delays when the load on the system is not taken into account, the load in our case being the load that affects the communication over the Internet and the load that affects the processing delays on a cloud infrastructure.

Acknowledgments This work is partially supported by Aljouf University represented by the Saudi Arabian Cultural Bureau in Canada. The authors would like to thank the anonymous reviewers for their constructive comments.

References

1. P. Mell, T. Grance, and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," 2011.
2. A. Amjad, P. Bodorik, and S. Sampalli, "Improving database security in cloud computing by fragmentation of data," in *2017 International Conference on Computer and Applications (ICCA)*. IEEE, Sept 2017, pp. 43–49.
3. H. Hacig, "Query Optimization in Encrypted Database," pp. 43–55, 2005.
4. N. Anciaux, M. Benzine, L. Bouganim, P. Pucheral, D. Shasha, and I. Rocquencourt, "GhostDB : Querying Visible and Hidden Data Without Leaks," 2007.
5. A. Hudic, S. Islam, P. Kieseberg, S. Rennert, and E. R. Weippl, "Data confidentiality using fragmentation in cloud computing," *International Journal of Pervasive Computing and Communications*, vol. 9, no. 1, pp. 37–51, 2013. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84878829696&partnerID=tZOtx3y1
6. H. Hacig and C. Li, "Executing SQL over Encrypted Data in the Database-Service-Provider Model," vol. 7, 2002.
7. B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB Journal*, vol. 21, no. 3, pp. 333–358, Aug 2011. [Online]. Available: http://link.springer.com/10.1007/s00778-011-0245-7
8. B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," pp. 720–731, Aug 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316752
9. L. Bouganim and P. Pucheral, "Chip-Secured Data Access : Confidential Data on Untrusted Servers," 2002.

10. S. Y. Ko and K. Jeon, "The HybrEx Model for Confidentiality and Privacy in Cloud Computing," 2011.
11. K. Zhang, X. Zhou, Y. Chen, and X. Wang, "Sedic : Privacy-Aware Data Intensive Computing on Hybrid Clouds Categories and Subject Descriptors," pp. 515–525, 2011.
12. Z. Zhou, H. Zhang, X. Du, P. Li, and X. Yu, "Prometheus : Privacy-Aware Data Retrieval on Hybrid Cloud," pp. 2643–2651, 2013.
13. C. Zhang, E.-c. Chang, and R. H. C. Yap, "Tagged-MapReduce : A General Framework for Secure Computing with Mixed-Sensitivity Data on Hybrid Clouds," pp. 31–40, 2014.
14. K. Y. Oktay and S. Mehrotra, "SEMROD : Secure and Efficient MapReduce Over HybriD Clouds The University of Texas at Dallas," pp. 153–166, 2015.
15. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB : Protecting Confidentiality with Encrypted Query Processing," pp. 85–100, 2012.
16. E.-O. Blass, G. Noubir, and T. D. Vo-Huu, "Epic: Efficient privacy-preserving counting for mapreduce," Cryptology ePrint Archive, Report 2012/452, 2012, http://eprint.iacr.org/2012/452.
17. J. J. Stephen, S. Savvides, R. Seidel, and P. Eugster, "Practical confidentiality preserving big data analysis," in *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. Philadelphia, PA: USENIX Association, Jun. 2014. [Online]. Available: https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/stephen
18. T. Mayberry, E.-o. Blass, and A. H. Chan, "PIRMAP : Efficient Private Information Retrieval for MapReduce," pp. 371–385, 2013.
19. E.-o. Blass, R. D. Pietro, R. Molva, and M. Onen, "PRISM — Privacy-Preserving Search in MapReduce," pp. 180–200, 2012.
20. S. D. Tetali and T. Millstein, "MrCrypt : Static Analysis for Secure Cloud Computations," pp. 271–286, 2013.
21. D. Liu, S. Wang, and C. I. C. T. Centre, "Programmable Order-Preserving Secure Index for Encrypted Database Query," *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 502–509, Jun. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6253544
22. D. Xiaodong, S. David, and W. Adrian, "Practical Techniques for Searches on Encrypted Data."
23. N. Singhal and J. P. S. Raina, "Comparative Analysis of AES and RC4 Algorithms for Better Utilization," pp. 177–181, 2011.
24. W. Stallings, *Cryptography and network security: Principles and practice.* Upper Saddle River, N.J. : Prentice Hall, 1999.
25. P. Samarati and I. C. Society, "Protecting Respondents' Identities in Microdata Release," vol. 13, no. 6, pp. 1010–1027, 2001.
26. J. Daemen, *The design of Rijndael : AES - the advanced encryption standard with 17 tables*. Berlin [u.a.]: Springer, 2002.
27. J. Blomer, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," *Lecture notes in computer science.*, no. 2742, pp. 162 – 181, 2003.
28. E. M. Mohamed, "Enhanced Data Security Model for Cloud Computing," pp. 12–17, 2012.
29. A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, R. Ramamurthy, P. Upadhyaya, and R. Venkatesan, "Secure database-as-a-service with Cipherbase," *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, p. 1033, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2463676.2467797
30. G. Nalinipriya and R. Aswin Kumar, "Extensive medical data storage with prominent symmetric algorithms on cloud - A protected framework," *International Conference on Smart Structures and Systems - Icsss'13*, pp. 171–177, Mar. 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6623021
31. "Rackspace: The Leader in Hybrid Cloud." [Online]. Available: http://www.rackspace.com/