Volume: 5 Issue: VI Month of publication: June 2017
DOI:

www.ijraset.com
IC Value: 45.98

Volume 5 Issue VI, June 2017
ISSN: 2321-9653

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Analysing Algorithms for Querying Encrypted Database

Mohmad Ibraheem[1], Mr. Jasbir Singh[2]
Department of Computer Science and IT, University of Jammu

Abstract**:** ***With the invent of cloud computing, database outsourcing has emerged as a new platform. After the introduction of Database-as-a-Service (DAS) model organizations are outsourcing management of their data to third party cloud service providers. A cloud database runs on a cloud computing platform, such as Amazon EC2, GoGrid, Salesforce and Rackspace. Secret and sensitive information contained in the database may be captured and leaked by a curious cloud database vendor. To protect data integrity and confidentiality database need to be encrypted before being outsourced to the cloud. Various bucketization techniques are used for executing queries on an encrypted database on a cloud. This paper is an extension of the work done by researchers. Query Optimal Bucketization (QOB) algorithm divides the server data into buckets subject to an optimality constraint. Query Based Bucketization (QBB) algorithm uses the prior information of query distribution to bucketize the server data. In this paper an improved algorithm based on QBB was proposed to reduce the total number False-Positives received while querying the encrypted database.***

Keywords: ***Database-as-a-service(DAS), Bucketization, Query Distribution, False-Positives, Entropy, Exposure Risk***

I. INTRODUCTION

Organizations are able to share data easily for a variety of purposes thanks to the recent explosive increase in the Internet usage, coupled with advances in software and networking. Cloud data storage is a service model which relies on the remote maintenance and management of data and providing availability to users over a network such as the internet [4]. Also referred to as DAS, cloud databases can use cloud computing to achieve optimized scaling, high availability, multitenancy and effective resource allocation [4,6,7]. Storing sensitive data in the cloud may lead to security fault when it resides on untrusted servers [6]. The service provider may have potential access to data above the allowed limit while providing database services. This can be fatal to the actual data owners when their sensitive data confidentiality is compromised. A possibly good solution to this problem is to store the data in its encrypted form [7]. For executing queries over remote encrypted databases various bucketization techniques are used.

The database is divided into a number of buckets each bucket is associated with a bucket ID. The bucket ID along with encrypted database is stored on the remote service provider. Database bucketization is a technique in which SQL–style range queries are executed over encrypted data on a DAS server [1, 2]. Encrypted data (tables) are partitioned into a number of buckets. Each bucket has an ID and a range defined by minimum and maximum value in the particular bucket. Client queries are mapped to the set of buckets that contain any value satisfying the range of the particular query. The original queries are translated to bucket–level queries, the encrypted buckets containing the desired values are requested. Since the client is trusted and once all encrypted buckets have been downloaded from the server, the false positives are filtered after the decryption of the downloaded records by the client.

Section II provides a review of a bucketization technique and an extension to it. The new algorithm is described in Section III. Evaluation of the algorithm is addressed in Section IV. Section V concludes the paper.

II. RELATED WORK

Various bucketization techniques were proposed by researchers for querying remote encrypted databases. Hore et al. [1] introduced Query Optimal Bucketization (QOB), which tries to minimize the bucket cost (BC) for each bucket, where BC is a function for calculating the range and density of values in the bucket. The bucket cost for a single bucket covering values from vi-vj is given by

$$BC\,(i, j) = (v_j - v_i + 1) * \sum_{i \le t \le j} (F_t)$$

QOB generates solution to the problem of bucketizing a set of values, $V = v_1, \ldots, v_n$, using at most M buckets, each value $v_1 < \ldots < v_n$ occurring once in V. The cost minimization problem follows the optimal substructure property, it is expressed as a combination of optimal solutions to two smaller sub-problems, such that one contains the leftmost *M-1* buckets covering the *(n-1)* smallest points from *V*, and the other contains extreme right. The optimal bucket partitions are built in the second part. This is achieved by attempting all possible data distributions across the M buckets and computing the cost, BC, cost for each distribution, and finally choosing the distribution the one with the minimum cost. The algorithm outputs the values of the bucket boundary for each of the M buckets

$$QOB(1,n,M) = Mini[QOB(1,n - i,M - 1) + BC(n - i + 1,n)]$$

Even if there is only a single record in a particular bucket that matches the query criteria QOB returns all the records within that bucket.

Shastri et al. [8] proposed parallel Binary Query Bucketization (PBQB) method which can be used for execution queries over encrypted database on a cloud. The method exploits data parallelism and applies simple binary search. PBQB uses the concept of binary search which searches a key in n sorted elements in *O(log n)* time. PQBB uses p processors to do the search and the data is divided into p chunks. PBQB lets the p processors search these chunks in parallel. Each of the chunks is recursively divided into p chunks the process continues until each of the processor gets only one value to compare with the key. The complexity of PBQB is $O(\log_p n)$ [9].

Antony et al. [10] proposed query based bucketization(QBB). QBB used beforehand knowledge about the distribution of queries to partition the database. The database is partitioned according to the probability with which a value is likely to be queried from the database.

III. METHODOLOGY

This paper is an extension of the work done by researchers [1-5]. Uniform distribution techniques like QOB are expected to perform less efficiently under non-uniform query distributions [12]. It has been observed most of the query distribution is non-uniform and follow zipf distribution [11]. Query Sensitive Bucketization (QBB) algorithm, which bucketizes a database according the query probabilities characteristic of a Zipf or Zipf–like distribution.

A. QBB algorithm operates in two steps

1) *Query Mapping:* Query mapping begins by calculating the mass function of an input query distribution. This results in two arrays the first of which contains the cumulative mass values for the query distribution, and the record contains indices mapping the query distribution to the data set. Initially, QBB takes as input a query set Q where q ε Q with a frequency f_q. The query set is represented as a frequency table where values are sorted from highest to lowest frequency. QBB then takes the maximum frequency to generate the corresponding Zipf (predicted) frequency of each query value. This is given by eq. (1).

$$f_{qi} = F/r^{\rho}, \qquad \text{eq. (1)}$$

Where f_{qi} is the predicted frequency of the *i*th query *qi*, *F* is the maximum frequency, *r* is the rank of *qi*, and r is the exponent characterizing the shape of the query distribution. The value of *r* is assumed to be known and set prior to running the algorithm. The mass of a query is then given by its predicted frequency over the sum of predicted frequencies. That is given by eq. (2).

$$pmf(q) = f_q / \sum_{i=1}^{N}(f_{qi}) \qquad \text{eq. (2)}$$

Where N is the size of (number of distinct values in Q). The cumulative mass for a query (given by eq. (3)) is the sum of preceding mass values, where the sum of all mass values is equal to 1.

$$cdf(q_i) = \sum pmf(q_{i-1}). \qquad \text{eq. (3)}$$

The cumulative distribution is stored in the form of an array, used for the partitioning process in the step 2 (Bucketization).

In the last part of this step a mapping function is used to associate each query value with the location of its representative in the database. The database is assumed to be sorted in, say ascending, order the mapping function searches the data set until it finds the first instance matching the query value and stores its location index in an array. The query values are already indexed in relation to their corresponding values in the cumulative distribution, thus, the latter are now mapped to the data as well.

2) *Bucketization:* In this step, the data set is partitioned according to three factors: (1) the number of buckets M desired by the user, (2) for M buckets, the proportion p of buckets allocated to the head bucket and the proportion allocated to the tail bucket 1−p, and (3) the proportion of query mass c represented by the head bucket. Initially, the distribution is split into a head and tail, where the head contains data values representing a proportion of the total query mass as defined by c. The tail contains mass of the proportion 1-c. The precise value of c is specified by the user prior to running the algorithm. For the remaining M-1 buckets, QBB will partition the head bucket into at most [M * p] buckets, then the tail into at least [M * (1- p)] buckets. Initially, the head bucket is created by finding the first data value representing the proportion of mass closest to c per cent of the rightmost bucket. If the head bucket does not have sufficient room for [M * p] buckets, it adds the difference to the number of tail buckets such that the total number of buckets is still M. The tail is partitioned into at least [M * (1- p)] buckets of

approximately equal width, where width is determined by number of distinct values.

The modification proposed in this algorithm rearranges the buckets according the ascending order of the bucket size as shown in line 24 of the following algorithm. The modification so proposed reduces the total number of false positives obtained.
The proposed algorithm is as follows

3) *Algorithm:* QBB (D, Q, M, c, p)
4) *Input:* Data set D (pre-sorted ascending), Query distribution
Q= (V, F) (where |V|=|F|= n, pre-sorted descending by F), max # buckets M, criterion value c for primary partition, and per cent of buckets p allocated to "head" bucket (where "tail" bucket gets 1−p per cent of buckets)

5) *Output:* Sorted set of QSB bucket partitions
6) Initialize

```
a)  matrix CDF[n] to 0;
b)  matrix S[M∗2] to 0;
c)  matrix IndexMap[n] to 0;
d)  nHeadBuckets =$\lceil M * p \rceil$;
e)  nTailBuckets =$\lfloor M * (1 - p) \rfloor$;
f)  CDF←CalculateMass(Q);
g)  IndexMap←MapQtoData(Q, D);
h)  find CDF[1≤k≤n]≈c;
i)  insert indexMap[k] into S;
j)  for i = 1 < nHeadBuckets do
k)      set probability to c*mass of rightmost head bucket;
l)      add total mass of previous head buckets to
m)      probability;
n)      find CDF[1≤l≤n]≈probability;
o)      insert indexMap[l] into S;
p)      insert indexMap[l]+1 into S;
    end
q)  insert indexMap[k+1] into S;
r)  insert indexMap[n] into S;
s)  set tailRange to $n-k+1$;
t)  for j = 1 < nTailBuckets−1 do
u)      insertindexMap[k+(tailRange/nTailBuckets)*j+1] into S;
v)  insert indexMap[k+(tailRange/nTailBuckets)*j+1]+1 into S;
    end
w)  sort (ascending) partition values in S;
x)  Rearrange S in ascending order of bucket size;
y)  Return: sorted set of QSB bucket partitions S;
```

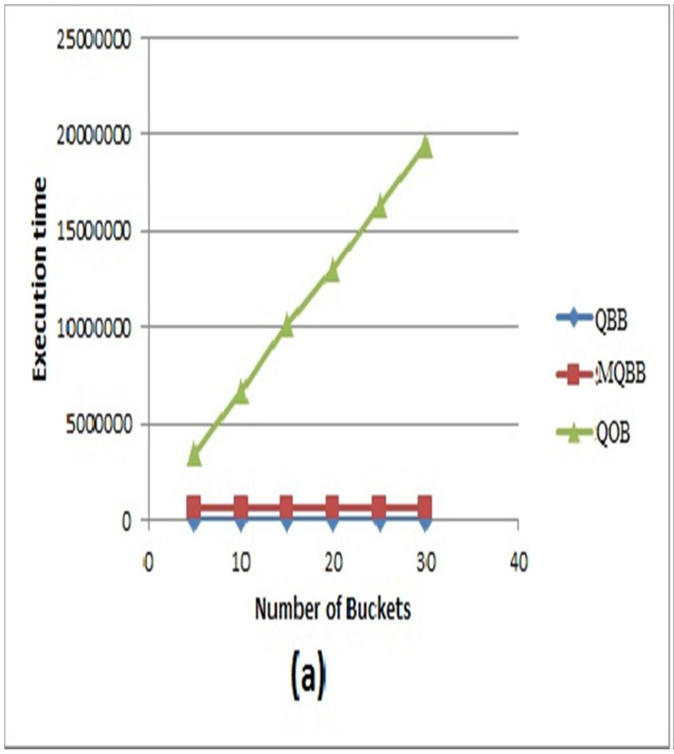
7) *Query Execution:* When a query is given by the client it is executed over the encrypted database rather than the actual database. Initially a connection is established to the encrypted database. With the help of metadata table which stores the mapping data the bucket IDs corresponding to the query is determined. The tuples satisfying the query are retrieved. A filtering process is done after the decryption in order to remove the false positives that occur in the result. The result is then displayed to the client.

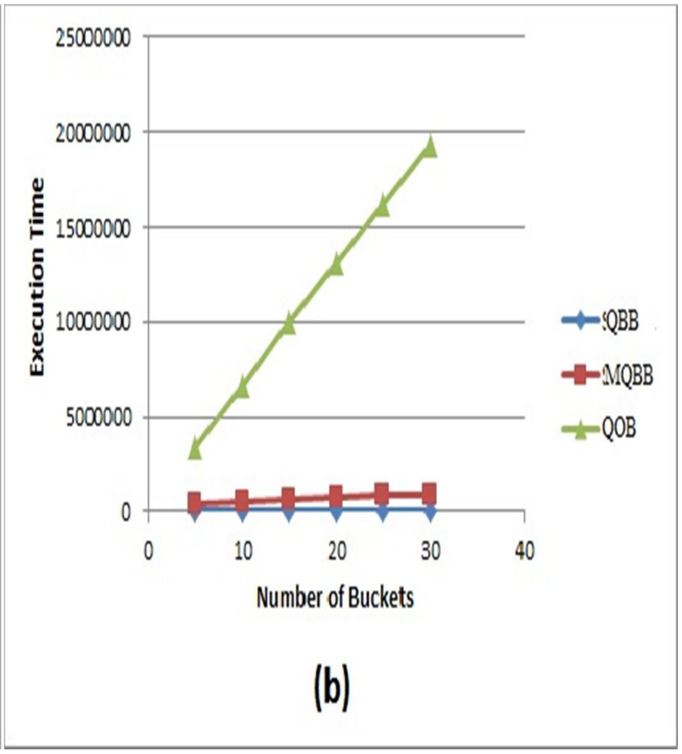
IV. PERFORMANCE EVALUATION

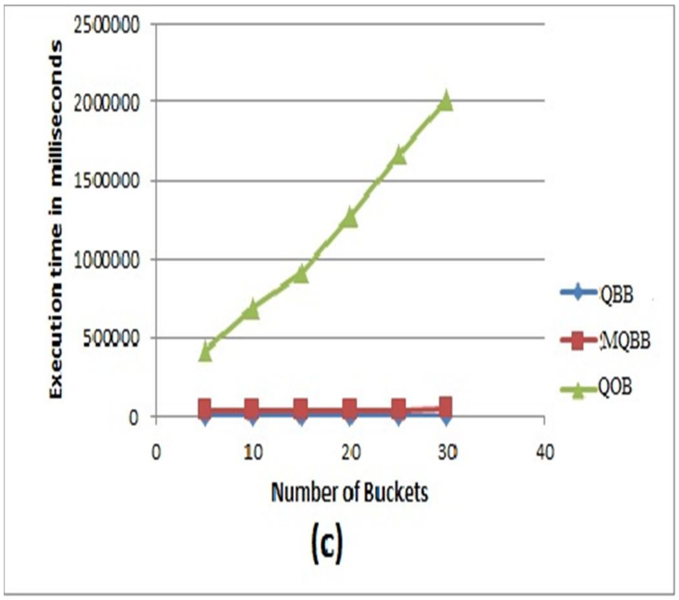
The performance of the proposed technique is evaluated for the metric of bucketization time and average false positives. The

www.ijraset.com Volume 5 Issue VI, June 2017
IC Value: 45.98 ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

metrics are calculated by executing the algorithm over three sets of data 1. Synthetic data with uniform distribution 2. Synthetic data with zipf distribution 3. Real data set.
Real data set used for performance evaluation is EDGAR log file data set assembled by the Division of Economic and Risk Analysis (DERA), U. S. Securities and Exchange Commission. It contains information in CSV format extracted from Apache log files [14]. Bucketization time is the time required by the algorithm to build the model over the data set. False positives are the subset of records received that do not satisfy the query. Since while querying a record from a data base entire bucket is received containing a number of unwanted results known as false positives.

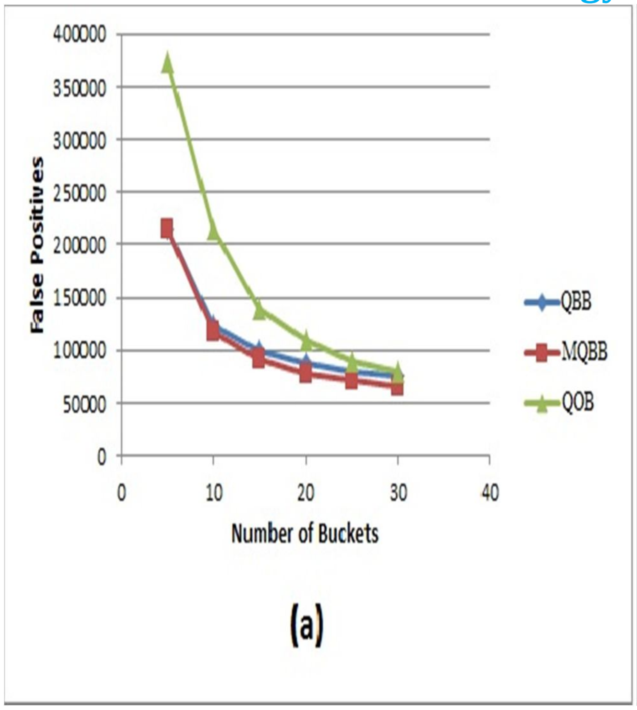


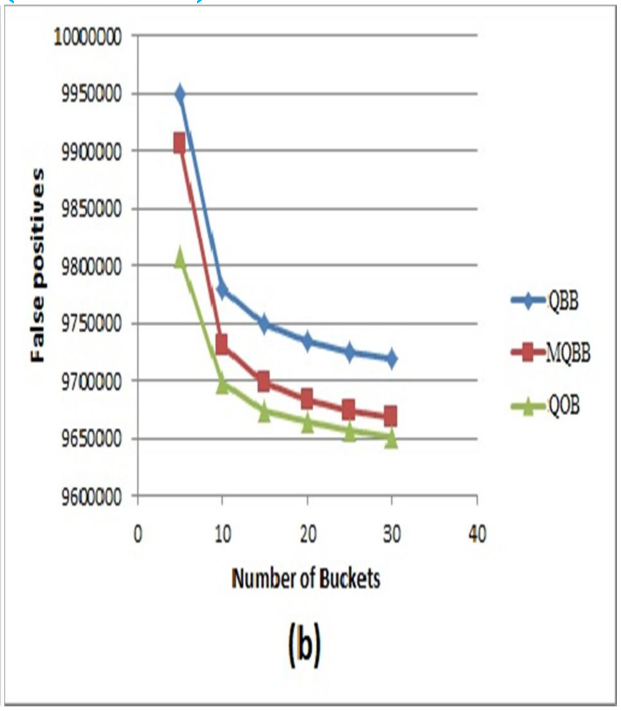


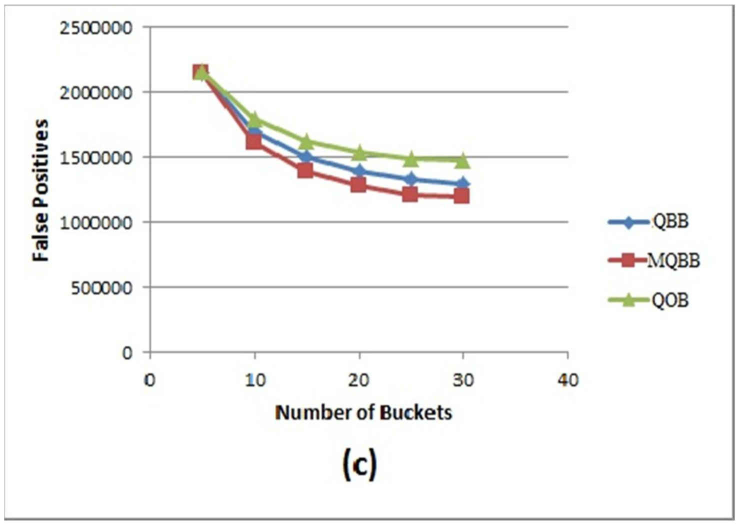


QBB: - Query Based bucketization.
MQBB: - Modified Query based Bucketization. /proposed algorithm.
QOB: - Query optimal Bucketization.

Figure 1: Bucketization Time in milliseconds for QBB, MQBB, and QOB on (a) Synthetic data with zipf distribution (b) Synthetic Data with uniform distribution (c) Real data.







QBB: - Query Based bucketization.
MQBB: - Modified Query based Bucketization. /proposed algorithm.
QOB: - Query optimal Bucketization.

Figure 2: False Positives for QBB, MQBB, and QOB on (a) Synthetic data with zipf distribution (b) Synthetic Data with uniform distribution (c) Real data.

V. CONCLUSIONS

This paper introduced a modification to the existing QBB algorithm. QBB algorithm used prior information about query distributions. The bucketization time for proposed algorithm is very less in comparison to QOB algorithm and is comparable to the QBB algorithm for all kinds of data. The number of false positives in case of proposed algorithm is lesser than QOB and QBB in case of synthetic data with zipf distribution and real data. In case of synthetic data with normal distribution false positives for the proposed algorithm are greater than QOB but lesser than QBB. It is commonly agreed that internet media access patterns follow zipf-like distributions rather than uniform distribution[13]. As evident from the results it is concluded that the proposed algorithm has the best performance in terms of false positives and bucketization time if the data is zipf distributed. In future, extensive research will be done considering the real time data sets from different servers. Also, further study of query distribution will be carried out as there can be other distributions that more accurately represent real-world queries.

REFERENCES

[1] B. Hore, S. Mehrotra, and G. Tsudik, “A privacy-preserving index for range queries,” in Proceedings of the 30th International Conference on Very Large Databases, pp. 720–731, 2004.
[2] H. Hacigumus, B. Hore, B. Iyer, and S. Mehrotra, “Search on encrypted data,” in Secure Data Management in Decentralized Systems (T. Yu and S. Jajodia, eds.), vol. 33 of Advances in Information Security, pp. 383–425, Springer US, 2007.
[3] Younho Lee, “Secure Ordered Bucketization”, IEEE Transactions on Dependable and Secure Computing, Vol. 11, Issue 3, 2014.
[4] Hacigumus, H., Iyer, B., Li, C., Mehrotra, S. Executing SQL over Encrypted Data in the Database Service Provider Model, SIGMOD 2002, June 4-6, Madison, Wisconsin, USA.
[5] M. Alwarsh and R. Kresman, “On querying encrypted databases,” in Proceedings of the 10th International Conference on Security and Management, pp. 256–262, 2011.
[6] Bouganim, L., Pucheral, L. Chip-Secured Data Access: Confidential Data on Untrusted Servers, In Proc. of the 28th VLDB Conference, 2002.
[7] Maheshwari, U., Vingralek, R., Shapiro, W. How to build a Trusted Database System on Unstrusted Storage OSDI 2000.
[8] Samraddhi Shastri, Ray Kresman, and Jong Kwan Lee,” An Improved Algorithm for Querying Encrypted Data in the Cloud” 2015 Fifth IEEE International Conference on Communication Systems and Network Technologies.
[9] Akl. Selim G. “The design and analysis of parallel algorithms”. p. cm. ISBN 0-13-200056-3.
[10] Anuja Antony , Silpa Joseph “Multidimensional Bucketization for Frequent Queries over Encrypted Data,” International Journal of Innovative Research in Computer and Communication Engineering November 2015.
[11] Rajeev Motwani, Sergei Vassilvitskii “Distinct Values Estimators for Power Law Distributions”.
[12] Tracey Raybourn, “Bucketization techniques for encrypted databases: quantifying the impact of query distributions (Unpublished masters thesis)”, Graduate College of Bowling Green State University, 2013.
[13] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang, “Does internet media traffic really follow Zipf-like distribution?,” SIGMETRICS Performance Evaluation Review, vol. 35, no. 1, pp. 359–360, 2007
[14] (2017) U.S. SECURITIES AND EXCHANGE COMMMISSION .Availiable https://www.sec.gov/data/edgar-log-file-data-set.html.

.