Journal of Theoretical and Applied Information Technology
31st July 2017. Vol.95. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HVM: A METHOD FOR IMPROVING THE PERFORMANCE OF EXECUTING SQL-QUERY OVER ENCRYPTED DATABASE

JA'AFER AL-SARAIREH
King Hussein School of Computing Sciences, Princess Sumaya University for Technology, Jordan
E-mail: J.saraireh@psut.edu.jo

ABSTRACT

The database system is an important element of information systems for the storage and management of information. Sensitive data in database systems must be protected using encryption techniques, whose application must balance data security and the functional efficiency of query processing.

Adaptive approach is proposed in this research to improve the performance of the query over the encrypting sensitive information in databases based on generating a unique hash value for each and every sensitive da ta and translating the SQL clauses into an appropriate form to execute over the hash values attribute. In this scheme, there are no statistical characteristics between the encrypted value and hashed value.

Keywords: *Database; Cryptography; SQL; HVM, Query Processing*

1. INTRODUCTION

All databases contain some degree of sensitive and classified information subject to legal or organizational regulations concerning accessibility and protection, thus sensitive data is generally secured by some form of encryption. Database security comprises physical, network, operating system and access control security, but none of these methods provide a sufficiently secure way to store and process data securely [1, 2, and 3]. Traditional security policies cannot sufficiently protect sensitive data in the database and prevent unauthorized use. Encryption provides an effective way to store sensitive data in encrypted form [4, 5], but the query performance over the encrypted database dramatically degrades the performance [6, 7]. To get the query results from an encrypted database, the DBMS decrypts all the encrypted data and then conducts the query over them. However, this is impractical because of the prohibitive cost of decryption over all encrypted data [1, 8, and 9].

Therefore, this study proposes adaptive approach to improve the SQL query over the encrypted database systems by using hash function to generate hash value for each sensitive fields in database.

In this research six scenarios are carried out by using the proposed approach, Naïve Database Encryption (NDE) method, Alhanjouri et al., 2012 method, and Sharama et al., 2013 method.

This paper is organized as follows: The related works are presented in the following section. In section 3, the framework for proposed approach is presented. In section 4, the experiments model is introduced. The results and discussions are presented in section 5, and the study is concluded in section 6.

2. RELATED WORK

A lot of approaches have been suggested in recent literature to efficiently support queries over encrypted databases, which vary in terms of how the index of attribute values is created. The approach proposed by [4] to execute SQL over encrypted database has weaknesses such as output false joining records occurrence, which leads to increased cost of decryption records and reduced query performance [6]. A new query method suggested by [10] completes the query on both the server and client sides, with support for the range query of numerical data provide by a proposed bucket index. Hankan Hacijumus [11] proposed a method of executing SQL over the encrypted data in the database-service-provider model, which is only valid for numerical data.

A bucket index method that balances security and trade performance by the partition of the bucket was proposed by [12] based on index support by Database Management System (DBMS), focused on

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

query performance at the cost of storage space. The approach proposed by [13] is not fit for the character data, and it also used bucket index to improve query performance. Zheng-Fei Wang proposed a function to support fuzzy query over the encrypted character data [6, 14]. Their scheme converts every adjacent two characters in the sequence and converts the original string directly to another character string by a hash function. This method cannot deal with some characters, and could perform badly for larger character strings.

Another approach proposed by [15] uses characteristics matrix to express string; the matrix is also compressed into a binary string as an index. Every character string needs a matrix size of 259x256, which is large and requires extensive computation; in addition, the length of the index is over 100 bits, which is not suitable for normal database storage. A B+ tree index was established by [16] for data prior to encryption. When querying the encrypted data, first of all, it locates the protected records related to the querying predicate based on the B+ tree index; secondly, it decrypts the encrypted records to attain the results. The results of experiments show that the query performance over the encrypted data is reduced by about 20% compared with the plaintext query performance. Jun Li [17] proposed method by using an index method for range queries on encrypted numeric data, but this approach is not fit for the character data due to its particular features that differ from numerical data.

A Reverse Encryption Algorithm (REA) represents a significant improvement over the encrypted databases is proposed by [18]. The results of REA can reduce the cost time of the encryption/decryption operations and improve the performance, but the encryption of the database is not optimally truthful and it needs some extra security by encrypting the data with another algorithm, to tighten security without degrading performance. To introduce the security in the database two tables for a single main table were suggested by [19]. The first table contains the actual data and the second one contains only that data on which the search query runs. A new approach proposed by [20] introduces a system with data encryption where sensitive columns are encrypted before they are stored to address data security. A new method of query over encrypted data in a database is proposed by [21]. It has added a layer above any kind of DBMS, which has the responsibility to manage the query over encrypted data.

3. PROPOSED APPROACH FRAMEWORK

The proposed framework is shown in figure 1. SQL query is transmitted into the dispatcher from applications. The dispatcher then distinguishes the data as sensitive or not sensitive. The queries from the client are sent to the layer with a subsystem called the Query Processor, to check in the Meta data if there is any query on an encrypted column. The Meta data contain an instance of a data structure object called Hash function, which stores the mapping between the plain and encrypted text.

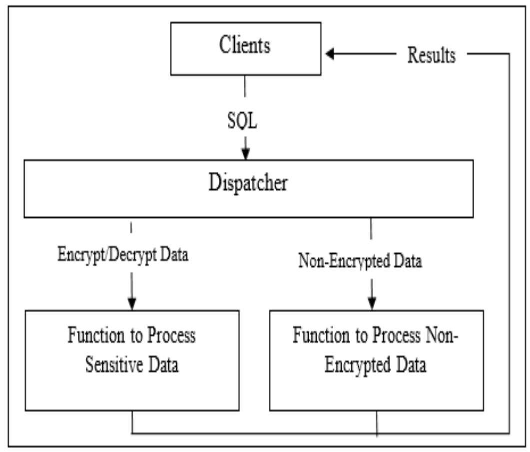


Figure 1: Architecture of the proposed framework

In the encryption/decryption layer of figure 2, the metadata module contains some mapping function to translate the input query to appropriate internal query in. While storing data, metadata is used to translate the queries in order to store the characteristic value of the encrypted data for indexing, together with the encrypted data themselves; while querying encrypted data, metadata is used to translate the user queries to appropriate queries executed on the encrypted data. The encryption and decryption module contains encryption functions and decryption functions, which encrypt and decrypt the sensitive fields, respectively. The following subsections describe the proposed HVM for store hash value for sensitive data and query over encrypted database.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

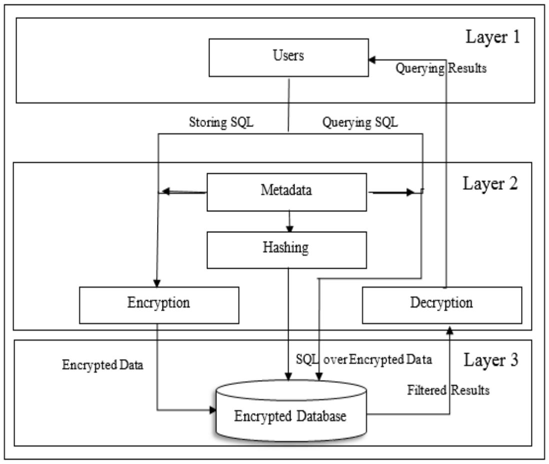


Figure 2: Architecture of encrypted storage and query over encrypted database

3.1 Query over Encrypted Data

Query Algorithm: Query over Encrypted Data by using HVM
Input: SQL, which is used to query the encrypted data
Output: Set of records satisfying the query condition
Phase 1:
1. Check if user is authorized to access data
   a. If (no) go to phase 2
   b. Else if (authorized user) go to phase 2
   c. Else exit

Phase 2:
1. Check the where clause contains encrypted data
   a. If (no) go to phase 3
   b. Else if (authorized user) go to phase 4

Phase 3:
1. Retrieve data from database
2. Exit

Phase 4:
1. Translate and modify the query conditions of SQL using the rules of metadata
2. Computing has value for query condition
3. Execute the modified query SQL
4. Retrieve all records that satisfy the hash value of query condition
5. Decrypt the records that retrieved from previous step
6. Return results
7. Exit

The following conditions are used in the architecture for our approach, as shown in figure 3:
Condition 1: Checks the user validity to the secure schema (authorized users).
Condition 2: Checks the condition of whether the query is on encrypted column
Condition 3: Is/are any record(s) found?

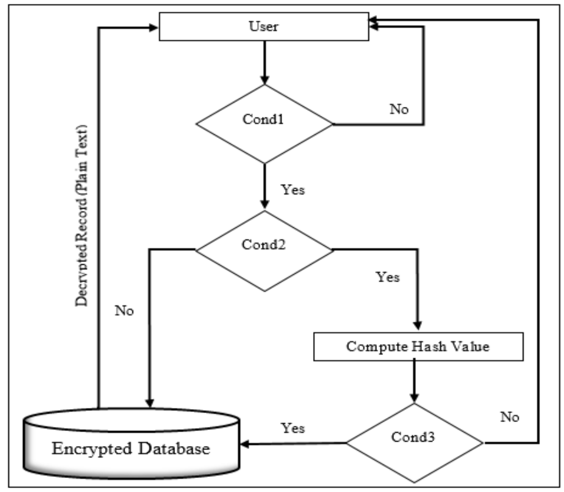


Figure 3: Model for query over encrypted database

3.2 Storing Encrypted Data

Storing Algorithm: Storing Encrypted Data and Hash Value for sensitive data // hash value function
Input: SQL is used to store encrypted data/column
1. Compute the hash value for sensitive data as follows:
   a. Convert the sensitive data to ASCII code
   b. Find the position for each digit in ASCII code and divided the input data to set of digits as follows:
      Find the number of digits Algorithm: Number of digits
      Count is number of digit set count = o
      Value = ASCII code for input
      While (value >0)
      { value = value / 10
      Digit [count] = value % 10 // $d_n, d_{n-1}, d_{n-2}, \ldots, d_2, d_1$
      Count++}
   c. Function to find the value of V1
      While (count > 0)
      *{V1= Digit [count] * power (k,count)*
      // $V_1 = d_n*(k)^n + d_{n-1}*(k)^{n-1} + d_{n-2}*(k)^{n-2} + \ldots\ldots\ldots + d_2*(k)^2 + d_1*(k)^1$
      Count--;}
   d. V_2 = bitleftshift(V_1, k)
   e. $V_3 = V_2$ Xor k
   f. Hash value = V_3
2. Encrypt value for sensitive data using AES with key size 256 bits
3. Store the encrypted data in the encrypted database
4. Store the hash value in a new column in the encrypted database

Example:
Select CustKey, Name, AcctBal
From Customer
Where AcctBal= 5296
The proposed technique intercepts this query and transform it as following:
Select CustKey, Name, decrypt (AcctBal)

From Customer
Where hash_value(AcctBal) = Hash (5296)

4. EXPERIMENTS MODEL

4.1 Experiments Setup

To show the validity and the efficiency of the proposed approach a set of experiments were carried out by using data in the database according to TCP-H benchmark [22]. Dbgen tool was used to generate data in a database automatically. AES-256 encryption algorithm was used to encrypt the account balance (ACCBAL) and Phone of customers table. All experiments were carried out on a personal computer with Intel Core i7 3.40 GHz, 8.00 GB RAM. The operating system was used is Microsoft Windows 10. The experiments were carried out on Oracle 11g. The Java programming language was used to implement the programming tasks. Each experiment was executed 10 times and the averages of results were considered. Different methods were tested to measure the response of SQL operations over the customer table, which has a number of tuples ranging from 100 to 10,000.

4.2 Scenario and Methods

The following scenarios were carried out in those experiments:

Scenario 1 (S1): *Select query* has no selected an encrypted field while *Where* statement has encrypted field. (*Select Name from Customer Where AccBal between 5643 and 9583*).

Scenario 2 (S2): *Select query* has selected an encrypted field and *where* the statement has encrypted field. (*Select Name, AccBal from Customer Where AccBal between 5643 and 9583*).

Scenario 3 (S3): *Insert* statement has an encrypted field. (*Insert into Customer values (CustKey, Name, AccBal)*)

Scenario 4 (S4): *Update* statement has no encrypted field while *where* statement has encrypted field. (*Update Customer Set Name = New_name Where CustBal = 5643*).

Scenario 5 (S5): *Update* statement and *where* statement has encrypted field. (*Update Customer Set CustBal = CustBal + 50 Where CustBal = 5643*).

Scenario 6 (S6): *Delete* operation, *where* statement has encrypted field. (*Delete from Customer Where CustBal = 5643*).

The following methods were carried out in all scenarios:

Method 1 (M1): Naïve Database Encryption (NDE), the traditional method

Method 2 (M2 Proposed Method): Using the proposed approach, which filters the reco rds the related to query conditions and then decrypt the results.

Method 3 (M3): Alhanjouri and Derawi's technique [21]

Method 4 (M4): Sharma et al.'s technique [19]

5. RESULTS DISCUSSION AND ANALYSIS

The comparison between all approaches is presented in table 1 for records number ranging from 100 to 10,000. As indicated in table 1, the average response time for the proposed approach is less than 14ms when carried out for all scenarios over the encrypted customer table with 100 records, as shown in figure 4(a). There is a significant improvement in execution query over the encrypted table, as shown in figure 4(b-d) for experiments No. 2-4.

Table 2 summarizes the percentage of improvement for the proposed scheme compared with other approaches. The percentages of improvement for all experiments are shown in figure 5. The average performance of the proposed approach is increased to 80%, 26% and 44% compared with methods 1, 3 and 4 (respectively) in experiment No. 1. On the other hand, for experiment No. 2 the average performance is enhanced by 74%, 27% and 36% compared with methods 1, 3 and 4, respectively. Also, there is better performance for the proposed scheme in experiment No. 3 and 4 compared with related works.

This improvement in the query performance and minimized CPU time cost is due to the proposed approach being based on computing a hash value for where clause conditions, then selecting all records that satisfy the hash value for where conditions. Methods 1, 3 and 4 decrypt all records in the customer table then retrieve the records that satisfy the where clause conditions. In contrast, Sharam et al. [19] used two tables for a single main table. The first table contains the actual data (CustKey, Name, Encrypt (AccBal),….), which has its sensitive data in encrypted form, while the second table contains Encrypt (CustKey), AccBal. This method consumes CPU time when executing queries over the encrypted table.

In scenario 2 Select query and Where clause has encrypted field. This scenario will require more CPU time than scenario 1 because of two encrypted fields, one in select statement and the other in where condition statement. The proposed approach reduces the CPU time cost and enhances the query execution performance in this scenario compared with other approaches when the records range from 100 to 10,000.

For insert scenario all methods have the same execution time except method 4 (Sharma method), which needs to insert a row in two tables (encrypted and query search tables). In scenarios 4 and 5 the update query is executed over the encrypted customer table with 100 to 10,000 records. The

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

proposed approach has better performance and reduced execution time compared to other methods, due to its use of hash value to filter the records, then decrypting a set of records to satisfy the where statement conditions. The delete query in scenario 6 has better performance and less response time in the proposed approach in all experiments.

On average for all scenarios, the proposed HVM approach minimizes the response time to 16ms, 27ms, 204ms and 519ms for experiments No. 1, 2, 3 and 4 respectively, as can be seen from the comparison with other approaches shown in table 3 and figure 6.

6. CONCLUSION

This research presented an enhancement to previous database encryption approach as by achieve better response time for the execution of SQL query. The experimental results indicate that the HVM approach improves the performance of query response time for all scenarios, providing excellent query response time for a variety of number of records. HVM improves the response time in comparison to NDE, alhanjouri, sharma method by 16ms, 204ms, 203ms and 519ms for experiments 1, 2, 3 and 4 respectively.

Future work will explore the application of the proposed approach to other kind of databases such as distributed DBMSs, where more focus should be given to the performance issue in the presence of more complex queries in real, large databases.

REFRENCES:

[1] Rathod, R.H. and Dhote, C.A., "A Literature Survey on performance evaluation of query processing on encrypted database", *International Journal of Engineering and Computer Science,* Vol. 3, No. 12, 2014, pp. 9637-9642.

[2] Nassar, M., Malluhi, Q., Atallah, M. and Shikfa, A., 2017. "Securing Aggregate Queries for DNA Databases", *IEEE Transactions on Cloud Computing,* 2017, doi:10.1109/tcc.2017.2682860.

[3] Ali, A. and Afzal, M.M., 2017. "Database Security: Threats and Solutions", *International Journal of Engineering Inventions*, vol. 6, No. 2, 2017, pp.25-27.

[4] Agrawal, R. and Kiernan, J., "Watermarking relational databases", In *Proceedings of the 28th international conference on Very Large Databases*, 2002, pp. 155-166.

[5] Wu, J. and Chen, J., "Research on the Method of Cloud Computing Storage Security based on the Homomorphic Encryption Method", *Advanced Science and Technology Letters,* Vol.139, 2016, pp.443-449.

[6] Wang, Z.F., Wang, W. and Shi, B.L., " Storage and query over encrypted character and numerical data in database". In *Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on* (pp. 77-81). IEEE.

[7] Guo, C., Zhuang, R., Jie, Y., Ren, Y., Wu, T. and Choo, K.K.R., "Fine-grained Database Field Search Using Attribute-Based Encryption for E-Healthcare Clouds", *Journal of medical systems*, Vol. 40, No. 11, 2016, pp. 1-8.

[8] Mc Brearty, S., Farrelly, W. and Curran, K., "The performance cost of preserving data/query privacy using searchable symmetric encryption", *Security and Communication Networks,* Vol. 9, 2016, pp. 5311-5332

[9] Jang, Y.D. and Kim, J.H., "A Comparison of the Query Execution Algorithms in Secure Database System", *International Journal of Electrical and Computer Engineering*, Vol. 6, No. 1, 2016, pp.337=343.

[10] Hacigümüş, H., Iyer, B., Li, C. and Mehrotra, S., "Executing SQL over encrypted data in the database-service-provider model". In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, pp. 216-227.

[11] Hacıgümüş, H., Iyer, B. and Mehrotra, S., "Efficient execution of aggregation queries over encrypted relational databases", In *International Conference on Database Systems for Advanced Applications*, 2004, pp. 125-136.

[12] Hore, B., Mehrotra, S. and Tsudik, G., "A privacy-preserving index for range queries", In *Proceedings of the Thirtieth international conference on Very Large Databases (VLDB Endowment*), 2004, pp. 720-731.

[13] Yu, H., Zhao, L., Xu, W.J., Niu, X.M. and Shen, C.X., "Research on a new method for database encryption and cipher index", *Dianzi Xuebao(Acta Electronica Sinica)*, 33(12), 2005, pp.2539-2542.

[14] Wang, Z.F., Dai, J., Wang, W. and Shi, B.L., "Fast query over encrypted character data in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

database", *Communication and Information Science*, Vol. 4, No. 4, 2004, pp.289-300.

[15] Zhu, H., Cheng, J., Jin, R. and Lu, K., "Executing query over encrypted character strings in databases". In *Frontier of Computer Science and Technology, 2007. FCST 2007. Japan-China Joint Workshop on* (pp. 90-97). IEEE.

[16] Wang, Z.F., Tang, A.G. and Wang, W., "Fast query over encrypted data based on B+ tree", In *International Conference on Apperceiving Computing and Intelligence Analysis*, 2009, pp. 132-135

[17] Li, J. and Omiecinski, E.R., "Efficiency and security trade-off in supporting range queries on encrypted databases", *In IFIP Annual Conference on Data and Applications Security and Privacy*, 2005, pp. 69-83.

[18] Mousa, A., Nigm, E., El-Rabaie, E.S., Faragallah, O.S. and Faragallah, O.S., "Query Processing Performance on Encrypted Databases by Using the REA Algorithm". *International Journal of Network Security*, Vol. 14, No. 5, 2012, pp.280-288.

[19] Sharma, M., Chaudhary, A. and Kumar, S.," Query processing performance and searching over encrypted data by using an efficient algorithm. *International Journal of Computer Applications,* Vol. 41, No. 4, 2013, pp.1308.4687.

[20] Arasu, A., Eguro, K., Kaushik, R. and Ramamurthy, R., "Querying encrypted data". In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 1259-1261.

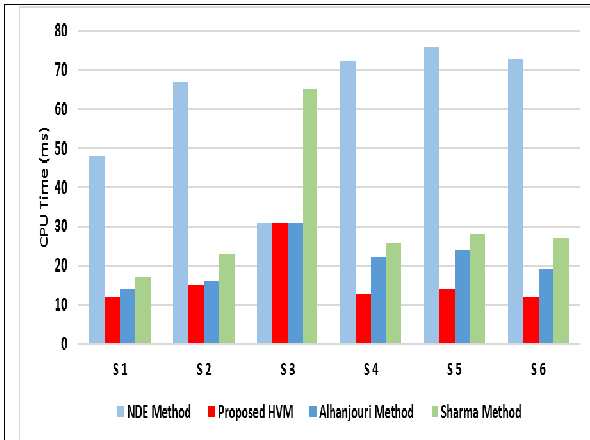
[21] Alhanjouri, M. and Al Derawi, A.M., "A New Method of Query over Encrypted Data in Database using Hash Map", *International Journal of Computer Applications*, Vol. 41, No.4, 2012, pp. 46-51

[22] TPC-H, Benchmark Specification., 2017, *http://www.tpc.org.*

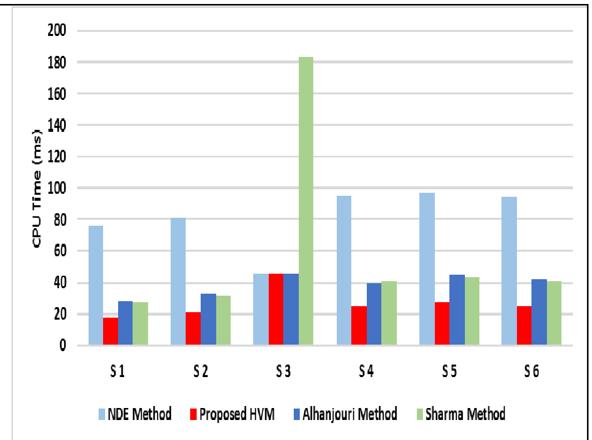
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1 CPU time Cost for Executing Query

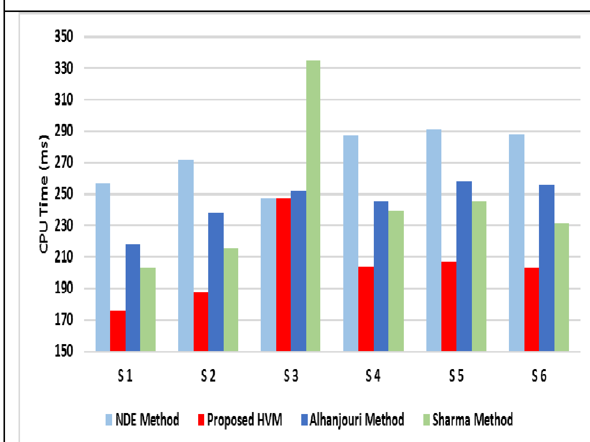
	Method	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
Experiment No. 1 (using 100 records)	NDE	48	67	31	72	76	73
	Proposed HVM	12	15	31	13	14	12
	Alhanjouri	14	16	31	22	24	19
	Sharma	17	23	65	26	28	27
Experiment No. 2 (using 500 records)	NDE	76	81	46	95	97	94
	Proposed HVM	18	21	46	25	27	25
	Alhanjouri	28	33	46	40	45	42
	Sharma	27	32	183	41	43	41
Experiment No. 3 (using 1000 records)	NDE	257	272	247	287	291	288
	Proposed HVM	176	187	247	204	207	203
	Alhanjouri	218	238	252	245	258	256
	Sharma	203	215	335	239	245	231
Experiment No. 4 (using 10000 records)	NDE	986	1052	675	1074	1081	1085
	Proposed HVM	472	509	675	483	483	493
	Alhanjouri	627	693	675	684	671	671
	Sharma	619	684	1084	635	631	635



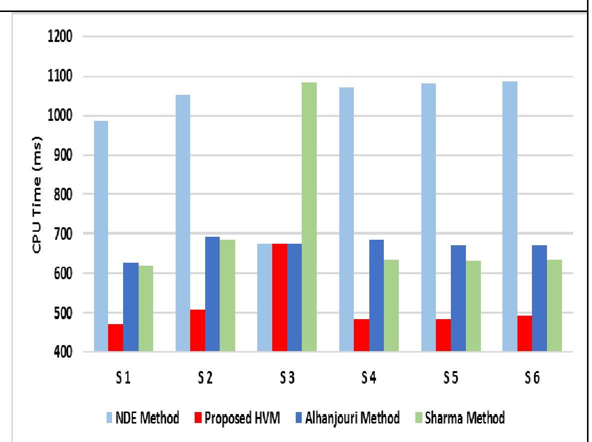
(a) Experiment No. 1



(b) Experiment No. 2



(c) Experiment No. 3

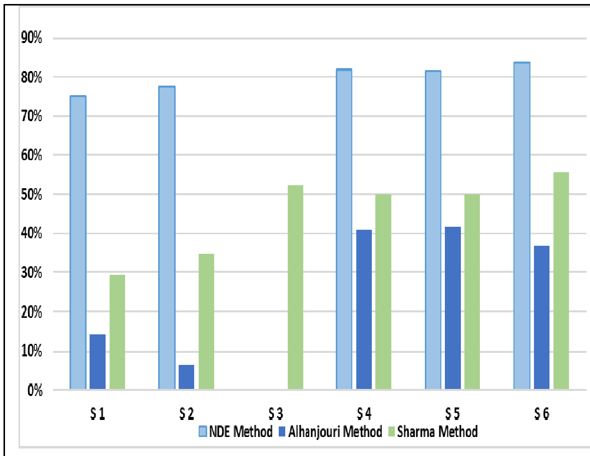


(d) Experiment No. 4

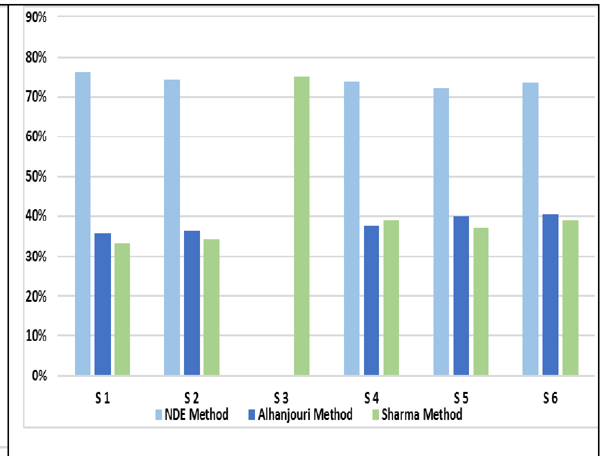
Figure 4 CPU time Cost

Table 2 Percentage of improvement for proposed approach compared with other methods

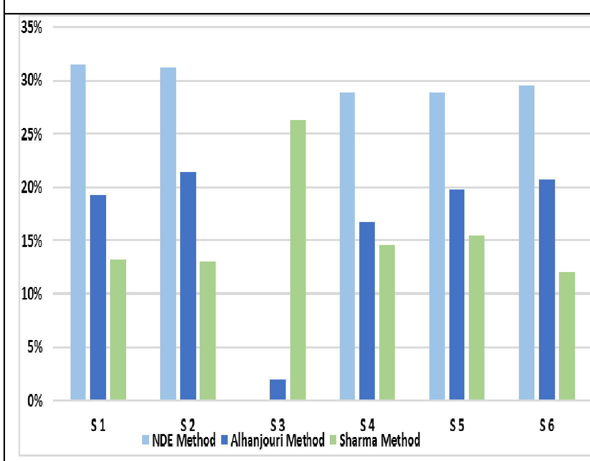
	Method	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
Experiment No. 1 (using 100 records)	NDE	75.00%	77.61%	0.00%	81.94%	81.58%	83.56%
	Alhanjouri	14.29%	6.25%	0.00%	40.91%	41.67%	36.84%
	Sharma	29.41%	34.78%	52.31%	50.00%	50.00%	55.56%
Experiment No. 2 (using 500 records)	NDE	76.32%	74.07%	0.00%	73.68%	72.16%	73.40%
	Alhanjouri	35.71%	36.36%	0.00%	37.50%	40.00%	40.48%
	Sharma	33.33%	34.38%	74.86%	39.02%	37.21%	39.02%
Experiment No. 3 (using 1000 records)	NDE	31.52%	31.25%	0.00%	28.92%	28.87%	29.51%
	Alhanjouri	19.27%	21.43%	1.98%	16.73%	19.77%	20.70%
	Sharma	13.30%	13.02%	26.27%	14.64%	15.51%	12.12%
Experiment No. 4 (using 10000 records)	NDE	52.13%	51.62%	0.00%	55.03%	55.32%	54.56%
	Alhanjouri	24.72%	26.55%	0.00%	29.39%	28.02%	26.53%
	Sharma	23.75%	25.58%	37.73%	23.94%	23.45%	22.36%



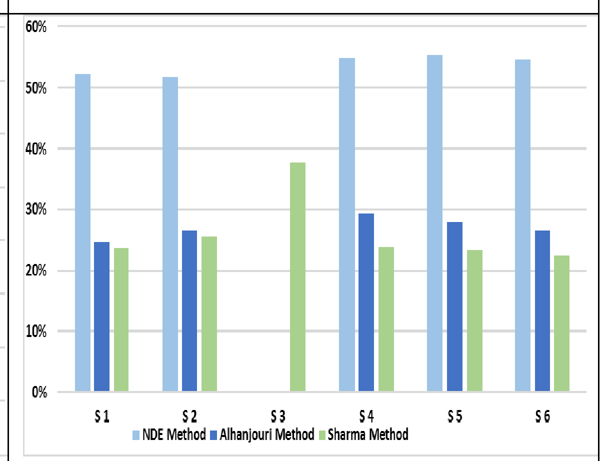
(a) Experiment No. 1



(b) Experiment No. 2



(c) Experiment No. 3



(d) Experiment No. 4

Figure 5 Percentage of improvement for proposed approach compared with other methods

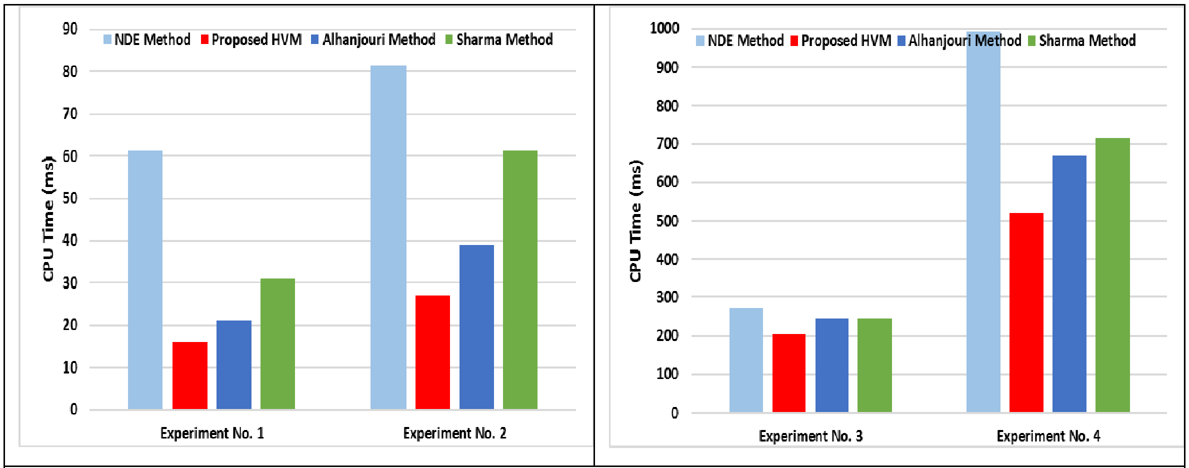


Figure 6 Time cost on average for all Scenarios for the proposed approach compared with other methods

Table 3 Time cost on average for all Scenarios for the proposed approach compared with other methods

Method	Experiment No. 1	Experiment No. 2	Experiment No. 3	Experiment No. 4
NDE	61	82	274	992
Proposed HVM	16	27	204	519
Alhanjouri	21	39	245	670
Sharma	31	61	245	715