International Research Journal of Computer Science (IRJCS)
Issue 04, Volume 5 (April 2018)
ISSN: 2393-9842
www.irjcs.com

A HYBRID MODEL FOR DATA FRAGMENTATION IN DISTRIBUTED SYSTEMS

Ikechi .B. Oriji. Ifeanyi .C. Ejiofor
Department of Computer Science,
University of Port Harcourt, Port Harcourt, Nigeria
kchoriji@gmail.com, ejioforifeanyi@yahoo.com

Manuscript History
Number: **IRJCS/RS/Vol.05/Issue04/APCS10081**
https://doi:/10.26562/IRJCS.2018.APCS10081
Received: 07, April 2018
Final Correction: 19, April 2018
Final Accepted: 24, April 2018
Published: April 2018
Citation: Ikechi, Oriji & Ejiofor (2018). A HYBRID MODEL FOR DATA FRAGMENTATION IN DISTRIBUTED SYSTEMS. IRJCS:: International Research Journal of Computer Science, Volume V, 186-192.
doi:/10.26562/IRJCS.2018.APCS10081
Editor: Dr.A.Arul L.S, Chief Editor, IRJCS, AM Publications, India

Abstract-- The performance and efficiency of distributed database management systems is mainly determined by the data partitioning and distribution scheme used in their design and implementation. In this study, an optimized, hybrid, automatic and dynamic fragmentation scheme using a heuristic simple supervised learning model that is dependent on the type and frequency of incoming queries was developed. The complex fragmentation sub-problems in distributed database design, implementation and management includes delayed responses to data retrieval queries, distributed joins, and database integrity checking/semantic data control. Significant efforts in trying to solve these problems have resulted in models that are mostly static and requiring heavy/complex computations. The incremental software development methodology was used to develop a simple and automated partitioning model that uses the ID3 decision tree to classify incoming retrieval queries, logs the queries, determines the most frequent queries, and at periodic intervals partition the relevant database tables vertically or horizontally while removing previous partitions. Thirty standard retrieval queries were used to test and validate the model as well compare its performance, in terms of query response time, with an existing model. Results show average response times of 0.0476150ms, 0.0252788ms, and 0.0173417ms corresponding to the scenario before fragmentation, after fragmentation on the existing model, and after fragmentation on the proposed model respectively. Results also show a 26% average gain in response time in the proposed model over the existing one which indicates a significant improvement in performance.

Keywords: Distributed Databases; Fragmentation; Machine Learning; Hybrid model; Vertical fragmentation; Horizontal fragmentation;

I. INTRODUCTION

Recent and rapid advances in automated data collection processes and devices have led to an avalanche of ever-increasing data items stored in distributed, heterogeneous, and massive databases and data marts all over the world [4]. Thus, in recent years, we have seen an ever increasing availability of data pertaining to people, organizations, governments, the environment, goods and services, machines, and the interactions between these entities. In order to make sense of, and benefit from such huge "silos of information", efficient data mining and analysis tools are required [5]. However, for these tools to work efficiently, the data must be organized and stored properly. The importance of the database concept to the proper capture, storage, processing, and retrieval of information in organizations cannot be over-stressed. Until recent times, databases were conceived and implemented as centralized data containers.

However, with globalization, advances in ICT, and the emerging knowledge economy, organizations tend to be more diversified (functionally) and dispersed (geographically). Thus, organizations may choose to distribute data resources over local servers scattered across the world (depending on the geographical spread of their operations) rather than keeping them in a central location. The result is that we can now talk of "Distributed Databases" as an emerging and soon to be dominant concept in data organization. Formally, a distributed database is a collection of multiple interconnected databases that are physically spread out across various locations that, themselves, communicate via a computer network [14]. It is interesting to note that despite the deployment of the best data organization models and traditional analysis methods, it has still been difficult to speedily locate, retrieve and update data stored in huge data repositories. Recent research shows that distributed databases have problems relating to the distribution of resources and the search and update of such resources [9]. These problems have often been associated with the fragmentation and allocation schemes deployed during the design of such databases. Fragmentation, for instance, has been consistently linked with problems such as delayed query responses due to distributed joins and database integrity checking/semantic data control. To tackle these and other related challenges, researchers and database designers and managers are turning to "Machine Learning" which is an offshoot of artificial intelligence [11].

The aim of this study was to develop a better technique for fragmenting or partitioning the relations or tables of distributed databases using a combination of conventional techniques together with a heuristic machine learning approach. Generally, the design of distributed databases is regarded as an optimization problem that requires the solving of several sub-problems that include, but not limited to, fragmentation, allocation, replication and query optimization. This paper proposes a new, hybrid, automatic machine – learning based model for partitioning tables vertically or horizontally in distributed databases using information on the type and frequency of queries. Specifically, it uses the supervised machine learning idea of classification to develop a technique that, periodically, analyses logged retrieval queries in order to classify them into any of two categories and then, using information on query frequency, partitions database tables vertically or horizontally. The remaining material in this paper will be presented as follows. Section II looks, briefly, at related work. The methodology used in developing the proposed is presented in section III. Section IV will look at the implementation and evaluation of the proposed system. Finally, the conclusions and recommendations for future work are presented in section V.

II. RELATED WORK

Available literature on related studies show that the distributed database design sub-problems of fragmentation, replication and allocation are usually considered together. However, just like this study, there are some significant works that considered only the fragmentation sub-problem. In one of the earliest most encompassing studies on fragmentation, [3] developed a vertical fragmentation strategy that, apart from being more general in application than any earlier wok, subsumed previous approaches. Using the square-error criterion, they derived an objective function for vertically fragmenting the relations of a distributed database. On their part, [10] developed a horizontal transaction-based fragmentation algorithm which, in conjunction with a replication routine, helped to increase database availability and reliability. They did this by extending the idea used in an existing vertical transaction-based approach. They replaced the use of attributes, in the vertical method, with predicates. In their opinion, the use of attributes and/or attribute affinity matrices does not recognize the importance of database transactions when creating fragments. Relying on key features introduced in the studies mentioned above, [13] developed a heuristic algorithm for vertically fragmenting a database. In it, he first formulated an objective function which he named Partition Evaluator (PE) which helps to analyze various algorithms using any agreed metric - either the type of input variable or objective function. Upon determining the "goodness" of any particular algorithm, a heuristic algorithm is then applied for the vertical fragmentation problem. Unlike earlier studies that used attributes affinity matrices as the input variable, [13], in this work, uses a matrix which is made up of attributes (as columns) and transactions (as rows) so that the entries or values represent the frequencies of access to the attributes for each transaction.

The work by [8] is by far the most significant with respect to the current study. In it, they developed a novel dynamic model that fragments, allocates, and replicates database relations and/or fragments. They were the first researchers, in this area, to look at the possibility of dynamically or "automatically" tweaking (designing and re-designing) distributed databases. All previous efforts were static in nature. Their work was hinged on the fact that many emerging distributed database applications had dynamic workloads resulting in changing access patterns. To significantly reduce communications cost in such environments, they proffered a mechanism of continuous fragmentation, replication, and re-allocation using recent access history. In another novel work by [7], they applied a genetic algorithm to the mixed fragmentation problem by using the algorithm iteratively to solve the attribute partitioning and tuple clustering sub-problems. Based on the contents of database relations and the empirical frequencies associated with their use, [1] developed a novel cost-based heuristic model for synchronizing the fragmentation and allocation of distributed databases. Specifically, he developed a horizontal fragmentation scheme that uses information on attribute retrieval and update frequency which is a marked departure from the usual dependence on attribute affinity or bonding.

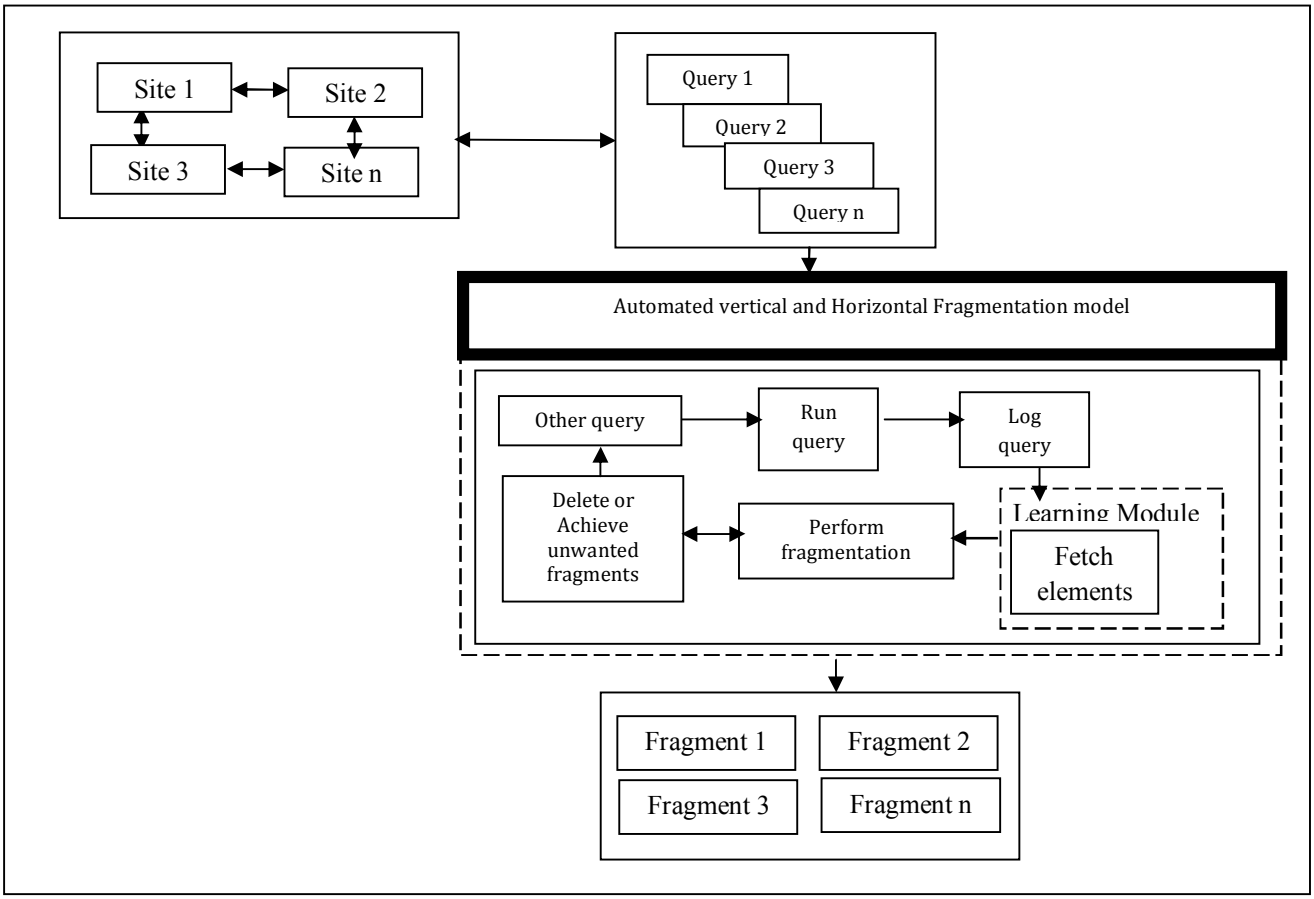
Unfortunately, the above efforts, despite their notable contributions to the research in database fragmentation, all required huge amounts of computations. This study, with its focus on developing a simple heuristic model that uses, mainly, information on database queries, drew heavily on the work by [12]. To overcome the challenge of heavy computations, [12] devised a new technique that fragments a relation vertically using the principle of 'frequent item sets' of its attributes. These refer to those attributes or items, which are being accessed or retrieved together very often.

III. METHODOLOGY

To obtain maximum value from distributed databases, in terms of performance efficiency and reduced storage and communication costs, attention must be given to the design techniques for fragmenting, allocating and replicating the data resources [6]. While fragmentation can be done without having allocation and replication in mind, allocation and replication are usually predicated on fragmentation [1]. The simple heuristic method we have developed is based on a supervised learning approach. Drawing heavily from the work of [2, 8, 12], our model relies on the access frequency and type of queries. Basically, the system logs all incoming queries, analyses each query whether it is CRUD (Create, Update, Retrieve, or Delete), groups or cluster similar queries, determine the most frequent incoming query type over a pre- set time interval, and then perform either a vertical or horizontal fragmentation on the database tables. The learning capability is enhanced over time by the continuous use of the system. That is, the more queries are analyzed the more the system's knowledge base is enhanced. In addition, the system has the ability to determine and delete fragments that are no longer needed thus ensuring optimum memory usage/management.

A. Archictecture of Proposed System

The figure, below, shows the architecture of the proposed system:



Following is a brief description of the procedures (within the outermost broken lines) shown in the architecture of the proposed system: For the purpose of clarity, the labeled boxes (or procedures) are briefly described below:

Other Query

This procedure is responsible for getting incoming MYSQL queries from the front-end files containing the user interface built with HTML, cleans the query of unwanted characters and pass cleaned query to the **runQuery** procedure.

RunQuery This procedure is in charge of verifying any received queries compliance to the MYSQL query commands standard and performs the query command on the earlier connected database using any of CRUD operation and returns corresponding result.

logQuery

This procedure is in charge of listening to any incoming queries and logging them to a locally prepared file in Javascript Object Notation (JSON) format for future reference as query history data source for possible database fragmentation.

fetchElements

This procedure is responsible for studying and inspecting incoming queries from the user of the Database administration system and retrieves relevant information from the query such as table name, columns, row values etc. to decide the kind of fragmentation (Vertical, Horizontal etc.) to be performed to optimize the database tables.

performFragment

This procedure takes overtime data from the locally logged files containing the all queries made to the database for a particular duration specified by the database administrator, parses the log file to the **fetchElements** procedure, collects the kind of fragmentation to make as a promise (returned result from the **fetchElements**) and perform database fragmentation based on it. Based on the architecture of Fig 1, the following fragmentation algorithm was developed.

B. Fragmentation Algorithm

- **Start**
- Setup database Server to connect to MySQL database
- Setup program execution Server to run remote/local server for fragmentation logic execution
- Present/Display User Interface to '**user**' to collect database log – in/connection details from '**user**'
- Pass database connection details collected from user to "**database connection**" procedure
- Ask user for 'database name'
- Ask user again for 'database password'
- Ask user again for 'database username'
- Click "Connect" button to execute "**database connection**" procedure
- If database connection details are correct and connection is 'successful' then goto "**Query Collection**" module/procedure
- Else, go back to step 5
- If database name provided at step 5 already exist . . . . . .
- Hook-up already existing database, else if database does not exist .. …
- Create the database and connect to the newly created database
- Go to "**Query**" module/procedure
- Ask user for **CRUD** (create, retrieve, update, delete) SQL command
- Check if query provided in step 16 is correct and execute
- If not correct, throw an error!
- Else, execute query command by calling "**Run Query**" module/procedure
- Run Query execute SQL command. Send response to user
- Log Query into a file stored locally in the SQL system
- Repeat step 6
- After database administrator – defined period of fragmentation elapses . …..
- Fetch/retrieve SQL commands logged over time ……..
- Send all SQL commands retrieved from log file to "**Fetchelements**" module/procedure
- The "**Fetchelements**" procedure studies each SQL command and returns most frequent SQL query/command
- The most frequent SQL command is then passed to the "**PerformFragmentation**" module/procedure
- The "**PerformFragmentation**" procedure receives the most frequent SQL command and decides which type of Fragmentation to perform - **Vertical** or **Horizontal**
- Perform Fragmentation
- Empty log files
- Drop irrelevant/unnecessary earlier fragments
- Reset Fragmentation countdown
- Go to step 16
- **Stop**

C. Design of Learning Model

Based on the specifications in the architecture for the proposed system, specifically the learning module (indicated by the innermost broken lines in Figure 1.1, above,), a supervised classification learning model was used for the purpose of teaching the new system to analyze incoming database queries so as to determine the type – whether it is column or row/tuple specific. On determining the type of an incoming query, such queries are logged appropriately until the user specified time for fragmenting the database.

The classification model used is the Decision tree – specifically, the ID3(Iterative Dichotomiser) Decision tree. This classifier was chosen because apart from being easy to understand and use, the class labels of the proposed model are very categorical – in this case 'Vertical' and 'Horizontal'.

To facilitate the training of the proposed query analysis sub – system of the proposed model, the following standard SQL queries were used:

1 SELECT Fname, Sname FROM students
2 SELECT Fname FROM students
3 SELECT * FROM students
4 SELECT * FROM students where Fname = "John"
5 SELECT Fname, Sname, Sex, Age FROM students
6 SELECT Fname, Sname, Sex, Age FROM students where age = "20'
7 SELECT * FROM students WHERE Fname = "Amos" and age = "25"
8 SELECT Reg_no, course, dept FROM students
9 SELECT * FROM staff
10 SELECT Emp_name, Emp_no FROM staff WHERE Emp_no = "BA100"
11 SELECT Emp_name, students, Emp_date FROM staff WHERE status = "Manager" OR status = "casual"
12 SELECT Emp_name FROM staff WHERE Branch_id ="Lagos"

Based on the above queries, the following table (Table I below) was constructed. In it, key information, necessary for describing and classifying each of the queries are stored. The purpose of this information is to help the query analyzer, the 'fetchQuery' procedure in our proposed model, determine the root node, internal nodes and leaves of the decision tree that will be used to classify the incoming queries. There are several statistical measures that have been developed for this purpose. In this study, we used the **Entropy** and **Information Gain** metric or measure.

Thus, we have that, the entropy is

$$\text{Entropy}(D) = \sum_{i=1}^{k} -P_i \log_2 P_i$$

And, the information gain is

$$\text{Gain}(D,B) = \text{Entropy}(D) - \sum_{V \in \text{Values}(B)} \frac{|D_v|}{|D|} \text{Entropy}(D_v)$$

Based on the values in Table I, an entropy value of '1' was obtained. Also, from Table I we obtained the largest information gain value of '1' for the attribute in column 'F' indicating that the most relevant determinant of whether a query is classified as 'vertical' or 'horizontal' is the presence or absence of the WHERE clause.

Table I - Table Showing Values of Attributes of SELECT Statements

S.No	A	B	C	D	E	F	
	Is First Statement SELECT?	First Statement followed by Asterisks ?	First Statement followed by one or more Colns?	FROM Statement following B or C?	FROM Statement followed by table name?	WHERE Key Word following E?	Class label
1	Yes	No	Yes	Yes	Yes	No	Vertical
2	Yes	No	Yes	Yes	Yes	No	Vertical
3	Yes	Yes	No	Yes	Yes	No	Vertical
4	Yes	Yes	No	Yes	Yes	Yes	Horizontal
5	Yes	No	Yes	Yes	Yes	No	Vertical
6	Yes	No	Yes	Yes	Yes	Yes	Horizontal
7	Yes	Yes	No	Yes	Yes	Yes	Horizontal
8	Yes	No	Yes	Yes	Yes	No	Vertical
9	Yes	Yes	No	Yes	Yes	No	Vertical
10	Yes	No	Yes	Yes	Yes	Yes	Horizontal
11	Yes	No	Yes	Yes	Yes	Yes	Horizontal
12	Yes	No	Yes	Yes	Yes	Yes	Horizontal

IV. IMPLEMENTATION AND EVALUATION

We strongly believe that one of the means of determining the effectiveness of any fragmentation method, and by extension the design of a distributed database, is through the query response time. Thus to implement the proposed system and compare its performance with an existing system, the one developed by [12], we used thirty (30) retrieval queries. Of this number, twenty-six were queries for retrieving records, specifically employee names, corresponding to each alphabet from A – Z.

This was to ensure that every section of an alphabetically sorted database table was searched. The remaining four were for retrieving records randomly from any part of the database table. To facilitate the evaluation of the proposed system, the queries were executed before and after fragmentation. Table II shows the results from the implementation of the existing and proposed system.

TABLE II - Analysis of Queries Response Times

	Before Fragmentation (ms)	After Fragmentation		Percentage Gain in Response Time Between Existing and Proposed Systems
		Existing Method (ms)	Proposed Method (ms)	
Query-1	0.0010100	0.0020175	0.00177233	12.15216853
Query-2	0.0010600	0.0025550	0.0022678	11.2407045
Query-3	0.0066300	0.0032670	0.002876	11.96816651
Query-4	0.0070400	0.0040700	0.0034492	15.25307125
Query-5	0.0098900	0.0053550	0.004617	13.78151261
Query-6	0.0105800	0.0056400	0.0044692	20.75886525
Query-7	0.0137800	0.0088450	0.007716	12.7642736
Query-8	0.0187000	0.0102200	0.008058	21.15459883
Query-9	0.0196100	0.0107850	0.008252	23.4863236
Query-10	0.0220900	0.0104661	0.00810006	22.6064798
Query-11	0.0237700	0.0137180	0.011052	19.43431987
Query-12	0.0282300	0.0157940	0.01124	28.83373433
Query-13	0.0364000	0.0178900	0.012268	31.42537731
Query-14	0.0397400	0.0181310	0.013404	26.07136948
Query-15	0.0433000	0.0188930	0.013758	27.17937861
Query-16	0.0509400	0.0190990	0.01418	25.75527515
Query-17	0.0563100	0.0221913	0.0152	31.50465791
Query-18	0.0611800	0.0251914	0.016066	36.22429233
Query-19	0.0662300	0.0281969	0.017722	37.14903007
Query-20	0.0768800	0.0301999	0.019242	36.28453614
Query-21	0.0785100	0.0345800	0.020312	41.26084442
Query-22	0.0802600	0.0369750	0.024166	34.6423259
Query-23	0.0805400	0.0376050	0.024977	33.58064087
Query-24	0.0819600	0.0378600	0.027118	28.37295298
Query-25	0.0864700	0.0395300	0.027872	29.49152542
Query-26	0.0885800	0.0439100	0.029012	33.92849009
Query-27	0.0912700	0.0462950	0.031174	32.66227454
Query-28	0.0917800	0.0467750	0.032258	31.03580973
Query-29	0.0934900	0.0472450	0.032566	31.06995449
Query-30	0.0942200	0.0485400	0.032911	32.19818706
				26.44237137

The results, in terms of response time, for the various queries, shown in Table II indicate that there is a significant reduction in the time taken for retrieving the requested queries as we move from implementing the system without fragmentation to the point when we have fragments in the existing and proposed systems. From the results in Table II, we obtained average response times of 0.0476150ms, 0.0252788ms, and 0.0173417ms for the no fragmentation scenario, the existing system after fragmentation, and the proposed system after fragmentation, respectively. Also, a comparison of the response times from the existing and proposed systems, in Table II, showed a 26% gain in average response time in favor of the proposed system. This can be explained by the reduced search space for records made possible by the continuous elimination of unwanted fragments in the proposed model.As the database and its tables grow over time, it is expected that further time savings will be recorded. For instance, Fig 2 shows that at the early stage of testing the system, when the queries were relatively few, there was noticeable overlap between the plots for the existing and proposed systems. From the graph in Fig 2, it is very obvious that our optimized model performs the retrieval operations for the test queries faster than the existing system.

In addition, the graph also shows that the query response times are shorter for instances where fragmentation has taken place compared to when there are no fragments. This suggests that fragmentation and indeed fragments may actually enhance database search and retrieval.

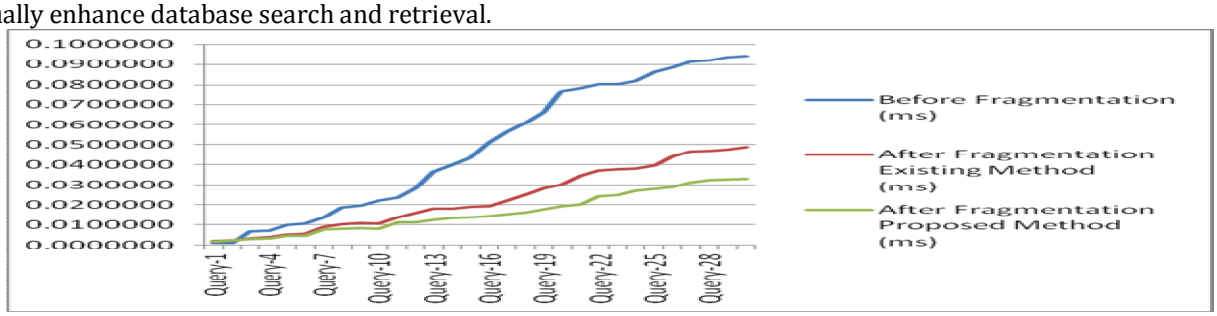


Fig 2 Plot of response time values

V. CONCLUSION AND RECOMMENDATIONS

One of the most critical aspects of distributed database design and management is fragmentation. If the fragmentation is done properly, we can expect to achieve better throughput from such systems. In this study, we set out to develop an optimized hybrid model for fragmenting data in a distributed environment. The results obtained from implementing the new system showed that the aim of the study has been achieved. In spite of several significant features of the model we have developed, there are still key features that need to be built into subsequent improvements or studies. For instance, there is need to design the fragmentation method, using preferably machine learning techniques, so that it can produce only non overlapping fragments that can be archived rather than being constantly deleted . Also, when fragments increase, there may be need to build an index of fragments so as to facilitate searches. Furthermore, we recommend that future studies expand the classifier used in the study by including queries that are not only retrieval queries, for example, insertion, update and delete queries.

REFERENCES

1. Abdulla, H. I. (2014). A synchronized design technique for efficient data distribution. Computers in Human Behaviour Vol. 30 Pg. 427 – 435.
2. Al-Sayyed, R.M.H., Al Zaghoul, F.A., Suleiman, D., Itriq, M. and Hababeh, I. (2014) A New Approach for Database Fragmentation and Allocation to Improve the Distributed Database Management System Performance. Journal of Software Engineering and Applications, **7**, 891-905.
3. Chakravarthy, S., Muthuraj, J., Varadarajan, R. and Navathe, S. B. (1994). An Objective Function for Vertically Partitioning Relations in Distributed Databases and Its Analysis. Journals of Distributed Databases and Parallel Databases, Vol. 2 Pg. 183 – 207.
4. Chowriappa P., Dua .S., and Todorov .Y. (2014). Introduction to Machine Learning in Healthcare Informatics. In: Dua, S., Dua, P., Rajendra, A.U. (eds) Machine Learning in Healthcare Informatics. Springer – Verlag, Berlin Heidelberg.
5. Cleophas, T. J. and Zwinderman, A. H. (2015). Machine Learning in Medicine - a Complete Overview.
6. Foder, A. and Lungu, I. (2016). Implementation of Fragmentation and Replication Methods in Distributed Systems. Journals of Information Systems and Operations Management, Vol. 10(2) Pg. 373 – 383.
7. Gorla, N., Ng, V. and Law, D. M. (2012). Improving Database Performance with a mixed Fragmentation Design. Journal of Intelligent Information Systems, Vol. 39 pg. 559 – 576.
8. Hauglid, J. O., Ryeng, N. H. and Norvag, K. (2010). DYFRAM: Dynamic Fragmentation and Replica Management in Distributed Database Systems. Journal of Distributed Parallel database Vol. 28 Pg. 157 – 185.
9. Jain, G. (2016). Distributed Data Management: Challenges and Solution on Distributed Storage. International Journal of Computer and Electronics Research, 5(2).
10. Khalil, N., Eid, D. and Khair, M. (1999). Availability and Reliability Issues in Distributed Databases using Optimal Horizontal Fragmentation. In: Bench-Capon, T., Soda, G., Tjoa, A. M. (eds) DEXA'99. Lecture Notes in Computer Science, Vol. 1677, Pg. 771 – 780. Springer-Verlag, Berlin, Heidelberg.
11. Kubat, M. (2015). An Introduction to Machine Learning. Springer, Switzerland.
12. Ramesh D., Vikas K., Chiranjeev K., & Amit K. (2014). An Apriori-Based Vertical Fragmentation Technique for Heterogeneous Distributed Database Transactions. In Mohapatra D. P. & Patnaik S. (eds.), Intelligent Computing, Networking, and Informatics, Advances in Intelligent Systems and Computing, 243, 687-695. DOI: 10.1007/978-81-322-1665-0_69
13. Runceanu, A. (2008). Fragmentation in Distributed Databases. In: K.Elleithy (ed.), Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering.
14. Silberschatz, A., Korth, H. F. and Sudarshan, S. (2011). Database System Concepts (6th Ed.). McGraw-Hill Inc., New York.

