Tree Based Fast Similarity Query Search Indexing on Outsourced Cloud Data Streams

Balamurugan Balasubramanian[1], Kamalraj Durai[1], Jegadeeswari Sathyanarayanan[1], and Sugumaran Muthukumarasamy[2]
[1]Research Scholar, Computer Science, Bharathiar University, India
[2]Computer Science and Engineering, Pondicherry Engineering College, India

Abstract: *A Cloud may be seen as flexible computing infrastructure comprising of many nodes that support several concurrent end users. To fully harness the power of the Cloud, efficient data query processing has to be ascertained. This work provides extra functionalities on cloud data query processing, a method called, Hybrid Tree Fast Similarity Query (HT-FSQS) Search is presented. The Hybrid Tree structure used in HT-FSQS consists of E-tree and R^+ tree for balancing the load and performing similarity search. In addition, we articulate performance optimization mechanisms for our method by indexing quasi data objects to improve the quality of similarity search using R^+ tree mechanism. Fast Similarity Query Search indexing build cloud data streams for handling different types of user queries and produce the result with lesser computational time. Fast Similarity Query Search uses inter-intra bin pruning technique, where it resolves the data more similar to user query. E- R^+ tree FSQ method branch and bound search eliminates certain bins from consideration, speeding up the indexing operation. The experiment results demonstrate that the Hybrid Tree Fast Similarity Query (HT-FSQS) Search achieve significant performance gains in terms of computation time, quality of similarity search and load balance factor in comparison with non-indexing approaches.*

Keywords: *Cloud, hybrid tree, fast similarity query, e-tree, r^+ tree.*

Received June 2, 2016; accepted May 1, 2017

1. Introduction

Cloud computing is receiving more attention as the end users store their data in the public cloud and then access the data at any time. So, several methods to retrieve the data from cloud server were presented by many researchers. Li *et al.* [7] had multi-keyword ranked search was applied to find the subset that was most likely to satisfy the user requirements. Wang and Ravishankar [16] Asymmetric Scalar product preserving encryption was applied to ensure secure and efficient retrieval of range queries.

Technological advancements in digital measurement and engineering made possible the capture of enormous data in several fields as astronomy, medicine, and seismology. Yiu *et al.* [19], similarity querying of metric data was outsourced using indexing and NN search resulting in the improvement of extraction of accurate processing of similarity queries. However, lack of trust remained a major setback to be addressed. With this objective, Carbunar and Tripunitara [2], a unifying trust framework was designed that not only resulted in the improvement of trust but also improved the rate of security.

Big data in cloud environment identify the trends of different social aspects and preferences of individual everyday behaviours for large data. Yun *et al.* [20], a fast approach to range aggregate queries aiming at improving the query retrieval using balanced partitioning algorithm was presented. Dastjerdi and Buyya [3] has another ontology-based approach to minimize effort of users in expressing their preferences was presented using evolutionary algorithms and fuzzy logic. Provisioning of resources in cloud environment using reinforcement learning was presented in [11] to reduce the time for retrieval. A novel indexing and retrieving mechanism using M R^* tree was presented in [18]. To improve search efficiency, Fine-grained Flexible Access Control (FFAC) [12] was presented.

In this work, we design a new method called Hybrid Tree Fast Similarity Query (HT-FSQS) Search to address the aforementioned issues. In HT-FSQS, we assume that the amount of query from the cloud user continues to increase from time to time. We propose a new hybrid tree structure that combines the advantages of both E-tree and R^+ tree, introduce a new Load-balanced R^+ tree algorithm to reduce the computation time and take the Fast Similarity Query Search indexing in to consideration to improve the quality of similarity search.

The rest of the paper is organized as follows. Section 2 summaries the related works on similarity search in cloud environment. Section 3 presents the system overview and the technical details of the proposed HT-FSQS method. Section 4 describes the experimental configurations and performance evaluations are presented in section 5. Finally, section 6 concludes the paper.

2. Related Works

Since the concept of cloud computing was proposed, similarity search, especially dynamic and fast similarity query search, has been one of the most important research components. Related works of similarity query search are mainly from different perspectives to construct a cloud computing system model aiming to attain universal results.

For both convenience and security, cloud users encrypt their data before outsourcing it to cloud storage service. However, performing similarity searching for the desired documents becomes complicated task due to the difficulty in decrypting the required document. A single server two round solution was investigated by Peng *et al.* [10] to reduce the storage and communication complexity. Searchable Encryption for multi-keyword ranked search was applied by Li *et al.* in [5] that resulted in the improvement of search functionality and search time. Another similarity search scheme over encrypted data was presented by Xia *et al*. [17] using secure index construction. On the other hand to perform fast exact similarity search, smith waterman algorithm was applied [4]. Secure storage and retrieval over internet services using key organised method was presented in [1, 9].

Similarity search, used in many data mining and information retrieval applications in cloud infrastructure, is a time consuming process. A two-stage heuristic algorithm was investigated in [14] to balance the load arising in cloud infrastructure and detect both similar and dissimilar queries. Secure and efficient ranked keyword search was performed in [15] to ensure file retrieval accuracy using one-to-many order preserving matching technique. Another research work using semantic keyword-based search [13] was presented using stemming algorithm to reduce the computation time in improving the search efficiency.

A transformation-based optimization framework was presented by Zhou and He [21] using a cost model resulting in optimizing the performance and cost for workflows in the cloud. Another method by Ma *et al.* in [8] was presented that performed scalable and reliable matching service minimizing the traffic overhead during subscription searching. Li *et al.* in [6] optimal routing for retrieving user query was presented using Exchanged Cross Cube (ECC). Distinguished from prior works, we establish a method of cloud computing system to strengthen the similarity query search based on the hybrid tree structure.

3. Construction of Hybrid Tree

The HT-FSQS constructs a hybrid tree structure which consists of both an Ensemble tree (E-tree) and a R^+ tree as shown in Figure 1. As shown in the Figure, the hybrid tree structure using E-tree and a R^+ tree is applied to the incoming file data streams ‘ $F = f_1, f_2, \ldots, f_n$ ’.

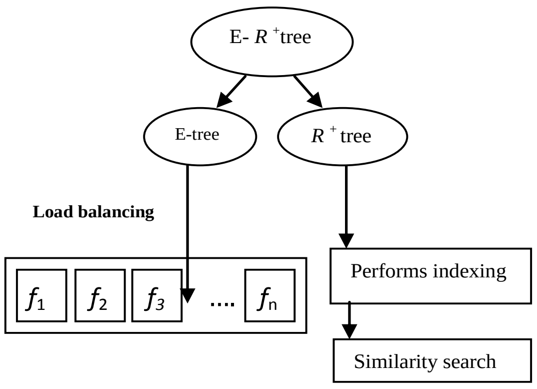

Figure 1. A Hybrid tree structure.

The Ensemble tree performs classification based on Spatio-temporal data stream that includes multiple attributes. The E-tree helps to split the data stream on cloud infrastructure in an equivalent fashion, reducing the overload factor. The tree indexes multidimensional objects which are dispersed over wider regions, providing higher similarity search results. E-tree gives good query performance by efficiently balancing the load but compromises index update performance. On the other hand, R^+ tree does pretty well when indexing with multidimensional objects, but its load balancing performance is comparatively lower than that of the E-tree. By combining together, E-R^+tree achieves a better performance for both load balancing and providing similarity search.

3.1. Ensemble tree (E-tree)

Ensemble tree classification (E-tree) performs classification on cloud data stream. The classification of data stream on tree structure clearly defines the file characteristics. The Ensemble tree (E-tree) performs classification process based on the data stream file given as input. For each file ‘f_i’ on the cloud infrastructure, the E-tree classifies the process in an efficient manner for ‘i’ users.

3.2. Construction of E- R^+ tree

The E- R^+ tree comprises of a hybrid tree structure constructed for load balancing and performing similarity search. The idea behind the construction of E- R^+ tree data structure is to group nearby data objects (i.e., quasi data objects) and characterize them with their Minimum Bounding Rectangle (MBR). Since all data objects resides within this bounding rectangle, a user queried data that does not intersect the bounding rectangle also cannot intersect any of the contained data objects. However, the HT-FSQS takes a different alternative, instead of simply building the index using the quasi data objects, Load-balanced R^+ tree or E-R^+ tree with quasi data objects is constructed. The motivation for integrity R^+ tree over the load balanced regions is that these regions (i.e., minimum bounding rectangle) represent reasonable bounds that bounds are load balanced, searching the exact user queried data at much lesser time interval. Once this index is created, different types of user queries are handled ascertaining the result at much lesser time. Figure 2 shows the construction of Load-balanced R^+ tree for quasi data objects.

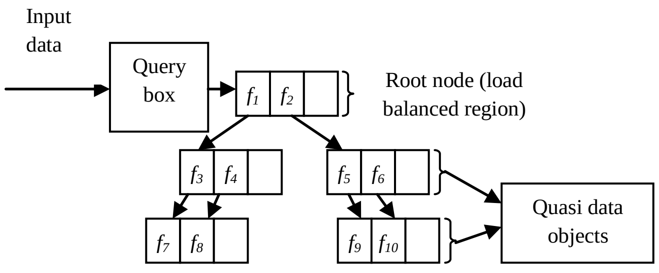

Figure 2. Load-balanced R^+ tree for quasi data objects.

The level above the leaves store root or the load balanced regions (i.e., the used queried data or file). So, no overlap is said to occur at this level between the nodes. The leaves of the hybrid tree (i.e., E- R^+ tree) store quasi data objects. All quasi data objects that belong to a given load balanced regions are inserted below that region. This also makes searching an easier procedure since there is only one place to insert a quasi data object, while in conventional R-tree a data object is inserted into any node, resulting in large amounts of computation. The E- R^+tree carries out multidimensional indexing with the quasi data objects.

Since the E-tree and R^+ tree overlap in space, at the leaves of the E-tree, a link list is built for each query with pointers to each of the R^+tree nodes contained by intersecting the query and the MBRs. Each query corresponds to the file obtained through different user queries. Figure 3 shows an example of the hybrid structure for E- R^+ tree.

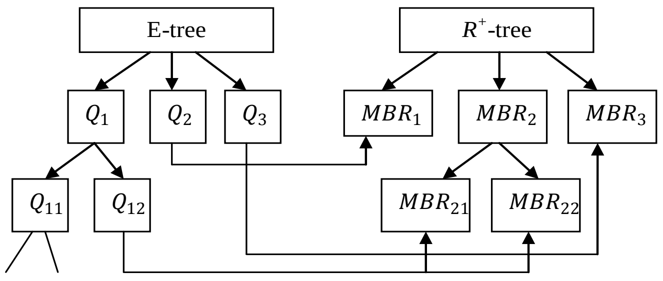

Figure 3. Example hybrid structure of E- R^+ tree.

As shown in the Figure, the structure of E- R^+ tree includes the ensembles corresponding to the data objects (files) to support cloud data processing. The search begins with the E-tree. For Example, as show in Figure, when a query 'Q_1' arrives, search starts from the E-tree and identifies that 'Q_{12}' intersects with the query. In this manner, all the quasi data objects under ensemble 'Q_{12}' is ascertained.

Algorithm 1: Algorithm for Load-balanced R^+tree.

```
Input: Query '$Q = Q_1, Q_2, \dots, Q_n$', File '$F = f_1, f_2, \dots, f_n$',
Minimum Bounding Rectangle '$MBR = MBR_1, MBR_2, \dots, MBR_n$'
Output: Optimized computation time
1: Begin
2: For each Query '$Q$' and File '$F$'
3: Make search for quasi data objects
4: Build a Load-balanced $R^+$tree for quasi data objects
5: Built link list by intersecting the query and the MBRs
6: Make search for quasi data objects
7: End for
8: End
```

In addition, since 'Q_{12}' also links to 'MBR_{21}' in the R$^+$ tree, the HT-FSQS also makes a search for quasi data objects indexed under 'MBR_{21}' in the R^+ tree also. Therefore, the link lists from E-tree nodes pointing to R^+ tree nodes improves the search time or the computation time to quickly search the exact user queried data. The linked list in the HT-FSQS used therefore speeds up the search as the HT-FSQS do not have to initiate the process from the root for each user queried data. Algorithm 1 is for Load-balanced R^+tree, as in the algorithm for each cloud data query processing that comprises of queries obtained from the cloud user, a hybrid tree structure combined E-tree and R $^+$tree is presented. This algorithm aims at reducing the computation time to quickly search user queried data by applying linked list.

3.3. Fast Similarity Query Search

SSFast Similarity Query Search indexing is implemented in HT-FSQS method to measure the quality of similarity search. Fast Similarity Query Search uses the inter-intra bin pruning technique, when it resolves the data more similar to the user query. The HT-FSQS method branch and bound search eliminate some bins from consideration and reduces the computation time.

The HT-FSQS introduced a novel indexing algorithm called, Fast Similarity Query Search for faster retrieval and also improves the quality of similarity search. In order to form the index, inter-intra bin pruning technique is applied where bins are created that contains similar queries. For each bin in inter-intra bin pruning technique, the HT-FSQS calculates the lower limit on the distance between a given user query and the most similar element of the bin.

Let us consider a vector 'V' from user queried data 'Q' and a bin 'B'. The lower limit between the user queried data 'Q' in vector 'V' and similar user queried data in that bin 'B' is given as below.

$$DIS(V,B) = \sqrt{(B_2 - B_1)^2 + (V_2 - V_1)^2} \quad (1)$$

The Equation given above quickly calculates the best order with minimum distance in inch to present the bins to the user query search. With the lower limit obtained from (1), the bound allow searching the bins in best first order, resulting in pruning bins from the entire search space without having to analyze the contents. Followed by this, a branch and bound search is applied to eliminate certain bins as they do not contain similar query search. To do this, given the best match, identifies whether any pruned data items in a specific bin may be more similar to the user query.

Suppose we have ‘$curr_{best_{value}} \leq DIS(V,B) - DIS(B,C)$’ and ‘$DIS(V,B) - DIS(B,C) < DIS(V,C)$’. By simplifying the above conditions, the resultant function obtained is

‘$curr_{best_{value}} \leq DIS(V,C)$’, which presents that ‘*C*’ quasi data object is not a better match for the user queried data present in vector ‘*V*. ’ than the current best value

‘$curr_{best_{value}}$’. Therefore, the pruning is continued with the ‘C’ quasi data object till best and fast similarity query search is provided. Algorithm 2 explains about the Fast Similarity Query Search process..

Algorithm 2: Fast similarity query search algorithm.

```
Input: Vector ‘$V$’, Query ‘$Q = Q_1, Q_2, \ldots, Q_n$’, Bin ‘$B = B_1, B_2, \ldots, B_n$’, Current best value ‘$curr_{best_{value}}$’
Output: Improved quality of similarity search
1: Begin
2:  For each Query ‘Q’ from set of queries ‘S’
3:  Create Bin ‘B’
4:      Measure distance between Vector ‘$V$’ and Bin ‘$B$’ using (1)
5:      While ‘$S \neq 0$’
6:      If ‘$S = \{Q_i \mid DS(Q, S_i) - DS(S_i, S_j < curr_{best_{value}})\}$’
7:      Retrieve corresponding Bin ‘B’
8:      Else
9:      Remove corresponding Bin ‘B’
10:     Continue with ‘$S_j$’ to prune with other set of quasi data objects
11:     End if
12:     End while
13:     End for
14: End
```

As shown in the Algorithm, the Fast Similarity Query Search algorithm creates a bin for each user queried data. The objective of Fast Similarity Query Search algorithm is to improve the quality of similarity search using inter-intra bin pruning technique and branch and bound search that eliminates unnecessary bins from consideration, reducing the computation time. For each user queried data from a set of queries, initially bins are created using distance measure. Then, by applying inter-intra bin pruning technique, quasi data objects more similar to user query are obtained and eliminate the unnecessary bin. This in turn not only reduces the computation time but also results in the improvement of quality of similarity search.

4. Experimental Setup

HT-FSQS method uses JAVA platform to perform similarity query search on cloud data processing based on the review comments provided by the customer using OpinRank dataset. The review comments in OpinRank dataset are placed in the cloud server and whenever, a query is posed by the cloud user, more similar comments are retrieved. Based on the similar queries, a decision regarding to visit to a hotel or not is made. This method is widely used to perform efficient cloud data processing with the tests and training samples. Hotel Customer Service Reviews (e.g., OpinRank Dataset-Reviews from TripAdvisor) is taken to perform the experimental work.

The training model for OpinRank dataset comprises of hotel reviews located in 5 different cities (London, New York, San Francisco, Last Vegas d Chicago). The performance measure is evaluated with the aid of Java platform and CloudSim simulator. The OpinRank dataset has been chosen in HT-FSQS method to perform cloud data processing as it gives a clear picture and helps in analyzing the queries made by new set of travelers regarding facilitations provided by hotel rooms in 5 different cities. The training set included in OpinRank dataset is 250,000 reviews. For experimental purpose, the HT-FSQS HT-FSQS used 350 reviews that include attributes namely, date of review, review title and full review made by the tourists. So to study the HT-FSQS method using the CloudSim simulator, we proposed a simulation environment that has the following parameters: the number of client requests that varies between 2 and 14 with query size of range between 4 and 28. The following parameters including load balancing factor, computation time to quickly search the exact user queried data and quality of similarity search in cloud service is evaluated. Comparisons were made with the state-of-the-art works namely, Multi Keyword Query over Encrypted data (MKQE) [7] and Hierarchical Encrypted Index (HEI) [16].

5. Discussion

This section presents the performance analysis of the HT-FSQS method and compared against the existing MKQE [7] and HEI [16]. The experimental results using CloudSim simulator in Cloud environment are compared and analyzed through table and graph form given below.

5.1. Impact of Computation Time to Quickly Search the Exact User Queried Data

Computation time is one of the most important standard metrics to measure the performance of similarity search in cloud environment. The computation time to search exact user queried data depends upon the queries issued by the cloud user and the time taken to extract single user query.

$$CT = Q_i * Time(extract\ user\ query) \tag{2}$$

From Equation (2), the computation time '*CT*' is measured and obtained according to the queries 'Q_i' issued by the cloud user and the time required to extract user queried data. It is measured in terms of milliseconds (ms). Lower the computation time, the efficiency of the method is said to be better.

Table 1. Tabulation for computation time.

Query (Q)	Computation time (ms)		
	HT-FSQS	MKQE	HEI
4	1.92	2.28	2.65
8	2.58	3.88	4.05
12	4.14	5.44	5.74
16	5.52	6.82	7.08
20	7.05	8.35	8.65
24	8.85	9.99	10.03
28	10.32	11.62	11.92

To support transient performance, in Table 1 a Load-balanced R$^+$ tree algorithm is applied and comparison made with two methods MKQE [7] and HEI [16]. From the above tabulation, computation time refers to the time taken to quickly search the exact user queried data at minimum time interval. Lower the time interval, higher is the number of queries addressed by the cloud server and therefore lower the computation time is said to be. Figure 4 depicts computation time with the number of user queried data. The total number of user queried data in our method is 28.

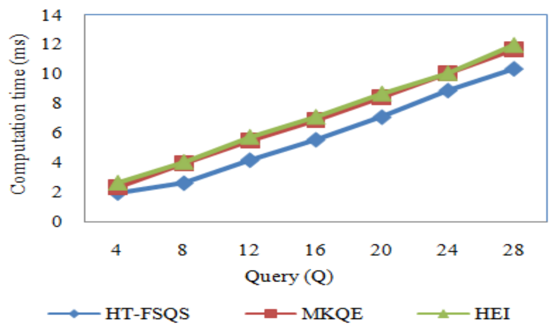

Figure 4. Measure of computation time.

As shown in the Figure, the HT-FSQS method ensures lower computation time when compared to MKQE [7] and HEI [16]. The computation time is reduced with the application of Load-balanced R^+ tree algorithm. Higher the number of user queried data, higher the amount of computation time is. With the application of Load-balanced R^+ tree algorithm, linked list created with the hybrid E-tree R^+ tree ensures search for quasi data objects in a load balanced region. Furthermore, by identifying the quasi data objects (nearby data objects) by constructing a link between the query and minimum bound rectangle, the computation time to search the user queried data is reduced using HT-FSQS method.

The total number of user queried data in our HT-FSQS is 28. The MKQE, HEI and proposed HT-FSQS method consume 2.28, 2.65 and 1.92ms for minimum user queried data. Also they consume 11.62, 11.92 and 10.32ms for maximum user queried data. The comparison shows the proposed HT-FSQS method offer 24.05 and 30.49% reduction for minimum and maximum user queried data compared to existing MKQE and HEI due to the design of hybrid tree model that considers both the load factor and performs similarity search accordingly.

5.2. Impact on Quality of Similarity Search

In order to measure the quality of similarity search '*SS*, the queries handled 'Q_h' and queries issued 'Q_i' are considered. The mathematical formulates for measuring the quality of similarity search is as given below.

$$SS = (Q_h / Q_i) * 100 \qquad (3)$$

From Equation (3), the quality of similarity search is measured in terms of percentage (%). Higher the quality of similarity search, the efficiency of the method is said to be high.

Table 2. Tabulation for quality of similarity search.

Query (Q)	Quality of similarity search (%)		
	HT-FSQS	MKQE	HEI
4	90.14	78.25	65.89
8	88.35	77.19	63.14
12	85.21	74.11	63.13
16	81.32	70.32	60.28
20	78.19	67.29	56.14
24	75.22	64.17	55.99
28	73.14	62.39	53.23

The targeting results of quality of similarity search rate using HT-FSQS method with two state-of-the-art methods [7, 16] in Table 2 presented for comparison based on the number of queries for fast similarity query search in cloud environment.

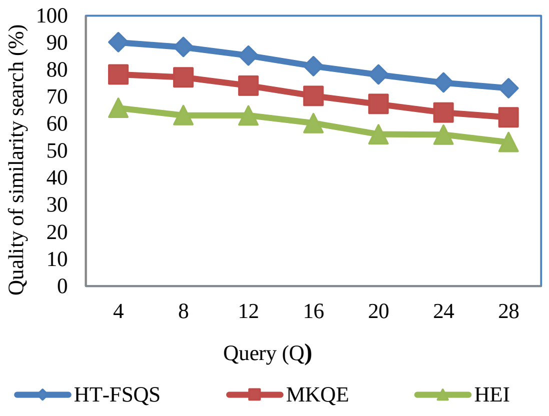

Figure 5. Measure of quality of similarity search.

Figure 5 depicts the measure of quality of similarity search made with respect to the user queried data in the range of 4 to 28 at different time in cloud environment. The increase in number of user queried data gradually decreases the quality of similarity search. The Fast Similarity Query Search using inter-intra bin pruning technique with branch and bound search results in the improvement of quality of similarity search by 13.67% and 26.88 using HT-FSQS method when compared to MKQE and HEI. With the inter-intra bin the data objects more similar to the user queried data is obtained in best first order that in turn retrieve the similar search. With this, the time to perform similarity search is reduced. Followed by this the branch and bound search eliminates the unnecessary bins, therefore providing more accuracy search results.

5.3. Impact on Load Balancing Factor

The load balance factor in HT-FSQS measures the number of client requests handled successfully by the cloud server at a particular time interval. The load balance factor measure is obtained as follows. It is measured in terms of percentage (%).

$$LBF = ((Client_H)*t) / Client_R \tag{4}$$

From Equation (4), the load balance factor '*LBF*' is measured on the basis of the client requests made (through user queried data) '*Client$_R$*' and the client requests handled '*Client$_H$*' by the cloud server at a specific time interval '*t*' respectively.

Table 3.Tabulation for load balance factor.

Client Requests (R)	Load Balance Factor (%)		
	HT-FSQS	MKQE	HEI
2	0.175	0.143	0.048
4	0.180	0.163	0.063
6	0.195	0.178	0.071
8	0.214	0.190	0.085
10	0.228	0.205	0.092
12	0.239	0.218	0.125
14	0.247	0.222	0.1322

As listed in Table 3, HT-FSQS method measures the load balance factor in cloud environment while performing similarity search with respect to client requests. It is measured in terms of milliseconds (ms). The load balance factor for performing similarity search in cloud environment using HT-FSQS method offers comparable values than the state-of-the-art methods.

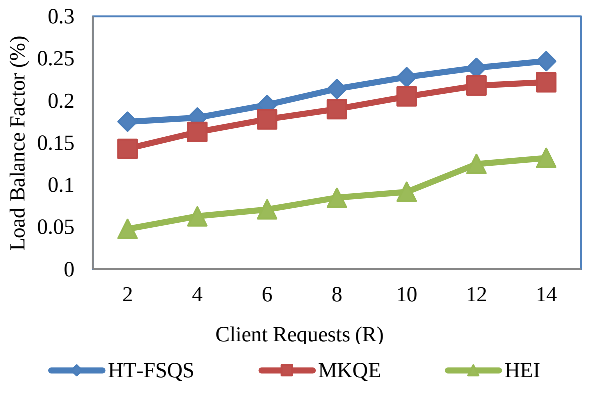

Figure 6. Measure of load balance factor.

Figure 6 presents the variation of load balance factor with respect to client requests while measuring the similarity search. All the results provided in figure 8 confirm that the proposed HT-FSQS method significantly outperforms the other two methods, MKQE [7] and HEI [16]. The load balance factor is improved in the HT-FSQS method using the hybrid E-R^+ tree structure. By applying the hybrid E-R^+ tree, the quasi data objects are characterized by minimum bounding rectangle where the bounds are load balanced. Therefore by integrating both E-tree and R^+ tree, the data objects resides within the bounding rectangle and hence resulting in the improvement of load balance factor by 10.95% and 59.32% compared to MKQE and HEI respectively.

6. Conclusions

HT-FSQS method is provided based on the inter-intra pruning, branch and bound method for efficient similarity query search in cloud environment. This method improves the quality of similarity search and reduces the computation time required for quickly searching the exact user queried data. As the method uses Load-balanced R^+tree algorithm, HT-FSQS method not only improves the load balance factor but also minimizes the computation time through efficient retrieval of similar query using for quasi data objects. By applying the Fast Similarity Query Search algorithm in HT-FSQS method, the quality of similarity search is improved where quasi data objects for similar query is retrieved through pruning technique. Finally, with the branch and bound method, unnecessary bins are removed improving the grade of performance on the cloud environment. A series of simulation results are performed to test the computation time, quality of similarity search and load balance factor to measure the effectiveness of HT-FSQS method. Experiments conducted on varied simulation runs shows improvement over the state-of-the-art methods. The results show that HT-FSQS method offers better performance with an improvement of load balance factor by 10.95% and reduces the computation time by 27.24% compared to MKQE and HEI respectively.

References

[1] Brindha T. and Shaji R., "An Instance Communication Channel Key Organizer Model for Enhancing Security in Cloud Computing," *The International Arab Journal of Information Technology*, vol. 13, no. 5, pp. 509-516, 2016.

[2] Carbunar B. and Tripunitara M., "Payments for Outsourced Computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 313-320, 2012.

[3] Dastjerdi A. and Buyya R., "Compatibility-Aware Cloud Service Composition under Fuzzy Preferences of Users," *IEEE Transactions on*

Cloud Computing, vol. 2, no. 1, pp. 1-13, 2014.

[4] Korpar M., Šošić M., Blažeka D., and Šikić M., "SW# Db: GPU-Accelerated Exact Sequence Similarity Database Search," *PloS one*, vol. 10, no. 12, pp. 1-14, 2015.

[5] Li H., Liu D., Dai Y., Luan T., and Shen X., "Enabling Efficient Multi-Keyword Ranked Search Over Encrypted Mobile Cloud Data Through Blind Storage," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 127-138, 2015.

[6] Li K., Mu Y., Li K., and Min G., "Exchanged Crossed Cube: A Novel Interconnection Network for Parallel Computation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 11, pp. 2211-2219, 2013.

[7] Li R., Xu Z., Kang W., Yow K., and Xu C., "Efficient Multi-Keyword Ranked Query Over Encrypted Data in Cloud Computing," *Future Generation Computer Systems*, vol. 30, pp. 179-190, 2014.

[8] Ma X., Wang Y., and Pei X., "A Scalable and Reliable Matching Service for Content-Based Publish/Subscribe Systems," *IEEE Transactions on Cloud Computing*, vol. 3, no. 1, pp. 1-13, 2015.

[9] Munadi K., Arnia F., Syaryadhi M., Fujiyoshi M., and Kiya H., "A Secure Online Image Trading System for Untrusted Cloud Environments," *SpringerPlus*, vol. 4, no. 1, pp. 270-277, 2015.

[10] Peng N., Luo G., Qin K., and Chen A., "Query-Biased Preview over Outsourced and Encrypted Data," *The Scientific World Journal*, vol. 2013, pp. 1-13, 2013.

[11] Peng Z., Cui D., Zuo J., and Lin W., "Research on Cloud Computing Resources Provisioning Based on Reinforcement Learning," *Mathematical Problems in Engineering*, vol. 2015, pp. 1-12, 2015.

[12] Ren W., Zeng L., Liu R., and Cheng C., "F2AC: a Lightweight, Fine-Grained, and Flexible Access Control Scheme for File Storage in Mobile Cloud Computing," *Mobile Information Systems*, vol. 2016, pp. 1-9, 2016.

[13] Shu J., Sun X., Zhou L., and Wang J., "Efficient Keyword Search Scheme in Encrypted Cloud Computing Environment," *International Journal of Grid and Distributed Computing*, vol. 7, no. 5, pp. 65-76, 2014.

[14] Tang X., Alabduljalil M., Jin X., and Yang T., "Load Balancing for Partition-Based Similarity Search," *in Proceedings of the 37th international ACM SIGIR Conference on Research and Development in Information Retrieval*, Gold Coast, pp. 193-202, 2014.

[15] Wang C., Cao N., Ren K., and Lou W., "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1467-1479, 2012.

[16] Wang P. and Ravishankar C., "Secure and Efficient Range Queries on Outsourced Databases using Rp-trees," *in Proceedings of the 29th International Conference on Data Engineering*, Brisbane, pp. 314-325, 2013.

[17] Xia Z., Zhu Y., Xingming S., and Wang J., "A Similarity Search Scheme over Encrypted Cloud Images Based on Secure Transformation," *International Journal of Future Generation Communication and Networking*, vol. 6, no. 6, pp. 71-80, 2013.

[18] Xiang J., Zhou Z., Shu L., Liu C., and Wang Q., "MR*-Tree: Novel Indexing and Retrieving Mechanism for Spatial Objects in Mobile PowerPoint Pages," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, pp. 1-11, 2015.

[19] Yiu M., Assent I., Jensen C., and Kalnis P., "Outsourced Similarity Search on Metric Data Assets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 2, pp. 338-352, 2012.

[20] Yun X., Wu G., Zhang G., Li K., and Wang S., "Fastraq: A Fast Approach to Range-Aggregate Queries in Big Data Environments," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 206-218, 2015.

[21] Zhou A. and He B., "Transformation-based Monetary Cost Optimizations for Workflows in The Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 85-98, 2014.

Balamurugan Balasubramanian received his MCA degree from St. Joseph’s College, Trichy during 2007 and currently a research scholar in Bharathiar University, Coimbatore, India. His research interest includes parallel and distributed computing, and network.

Kamalraj Durai received his MCA degree from St. Joseph’s College, Trichy during 2007 and currently a research scholar in Bharathiar University, Coimbatore, India. His research interest includes parallel and distributed computing, and database.

Jegadeeswari Sathyanarayanan received her M.Sc Computer Science degree from Kanchi Mamunivar Centre for Post Graduate Studies, Puducherry during 2005 and currently a research scholar in Bharathiar University, Coimbatore, India. Her research interest includes parallel and distributed computing, and network.

Sugumaran Muthukumarasamy received his M.Sc degree in mathematics from University of Madras during 1986 and M.Tech degree in computer science and data processing from Indian Institute of Technology, Kharagpur, India in 1991, and obtained his Ph.D from Anna University, Chennai in 2008. He is currently working as Professor of Computer Science and Engineering at Pondicherry Engineering College, India. His areas of interests are theoretical computer science, analysis of algorithms, parallel and distributed computing, and Spatial-Temporal data.