International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 – 5161

Available at http://inpressco.com/category/ijcet

Research Article

An improving query optimization process in Hadoop MapReduce using ACO-Genetic algorithm and HDFS map reduce Technique

Chandra Shekhar Gautam[1] and Dr.Prabhat Pandey[2]

[1]A.P.S University Rewa, (M.P.) India
[2]OSD A.P.S University Rewa (M.P), (India)

Received 25 March 2023, Accepted 10 April 2023, Available online 13 April 2023, **Vol.13, No.2 (March/April 2023)**

Abstract

An improved query optimization process for big data using a combination of the ACO-GA algorithm and HDFS map-reduce. The methodology consists of two phases: BD arrangement and query optimization. In the first phase, the input data is pre-processed by finding the hash value using the SHA-512 algorithm and removing repeated data using HDFS map-reduce. Then, features such as closed frequent pattern, support, and confidence are extracted and managed using entropy calculation. Based on this calculation, related information is grouped using the Normalized K-Means algorithm. In the second phase, the BD queries are collected, and the same features are extracted. The optimized query is then found using the ACO-GA algorithm, and the similarity assessment process is performed [1].The paper claims that the proposed algorithm out performs other existing algorithms. However, without more details about the experimental setup and the specific metrics used to evaluate the performance of the algorithm, it is difficult to assess the validity of this claim. Additionally, it is unclear how the proposed algorithm compares to other state-of-the-art query optimization techniques. Further research and comparative analysis are needed to fully evaluate the effectiveness of this approach [4][5].

Keywords: *Hadoop Distributed File System (HDFS), Normalized K-Means (NKM) algorithm, Ant Colony Optimization-Genetic Algorithm (ACO-GA), Secure Hash Algorithm (SHA-512) ·*

Introduction

The importance of analysing large amounts of data in both commercial and academic organizations. Internet companies, for example, collect massive amounts of data through various sources such as service logs, web crawlers, and click-streams. As the amount of data collected exceeds the storage space and processing power of traditional storage systems, it is referred to as Big Data [7]. The need for software platforms that can handle dynamic multi-objective Big Data optimization problems. A Big Data processing platform is defined as a computing platform designed for processing Big Data. The current research in databases and industrial practices is focused more on performance rather than energy efficiency. However, the passage notes that traditional relational database management systems (RDBMS) or standard statistical tools are not capable of handling Big Data [6].

The challenges of analysing large amounts of data, which may require processing tens or hundreds of terabytes of data.

*Corresponding author's ORCID ID: 0000-0000-0000-0000
DOI: https://doi.org/10.14741/ijcet/v.13.2.8

Many companies rely on highly distributed software systems functioning on big clusters of commodity machines to handle such large-scale data processing. Query optimization is an essential component of analysing large data sets, as it can help minimize the number of queries required to process the data. The passage notes that there is a lot of scope for development in the area of query optimization, given the increasing importance of data in various fields such as user base creation, research, and market analysis. In many relational database management systems, multiple query plans for fulfilling queries are analysed, and a good query plan is identified to reduce the use of certain resources such as I/O. This optimization helps to select the best query access plan to process data more efficiently.

The common methods for conducting big data analytics, which typically involve writing and executing queries in SQL-like languages supported by systems such as Hadoop, Scope, and Spark.

Hadoop is an open-source framework that enables large-scale data-intensive computing applications. It implements algorithms based on Google's extensive research and experience in massive data processing. The framework is designed to make large-scale batch processing easier to use and includes the Map-Reduce

(MR) paradigm, which allows distributed processing of large datasets across clusters of computers. The Hadoop Distributed File System (HDFS) is also used as a storage system for large data sets.

MR was first introduced by Google in 2004 and has since become a popular framework for processing large datasets. The existing open-source project, Hadoop, is based on the MR paradigm and enables parallel processing of vast amounts of data, automatic partitioning of data, data distribution, fault tolerance, and load balancing management, resulting in reliable and scalable computing. The MR paradigm is based on the idea of dividing large datasets into smaller, more manageable pieces and processing them in parallel across a distributed network of hundreds or thousands of commodity machines. This approach enables high scalability and availability, as well as fault tolerance, since the system can continue to function even if some machines fail[5].

Apache Spark is the utmost extensively utilized open-source processing engines intended for BD, with rich language-integrated APIs in addition to an extensive gamut of libraries [19]. The core Spark API is centred on collections of Java/Python objects, wherein users run random functions written in these languages via operators such as map or groupBy. The draft structure for this paper is systematized as Sect. 2 surveys the associated works regarding the method proposed. In Sects. 3, a concise discussion about the proposed methodology is preferred, Sect. 4, explore the Investigational outcome and also Sect. 5 deduces the paper.

Related Work

Sahal et al. [20] proposed an index-cantered system for reutilizing data called Indexing HiveQL Optimization for JOIN queries over a multi-session BD environment called iHOME. This methodology was divided into three groups: compute-on-iHOME, Filter-of-iHOME, and Similar-to-iHOME. The experimental results showed that the total execution time of eight JOIN queries using iHOME on Hive was reduced. However, further improvements were needed for this methodology.

Mukul and Praveen [21] propounded a heuristic-centric algorithm as a key for the issue of MJQO (i.e. Multi Join Query Ordering). This algorithm integrated '2' fundamental search algorithms, tabu search and cuckoo. The simulation evinced some exciting outcomes in respect of the propounded algorithm and deduced that the propounded algorithm could resolve the MJQO problem in lesser time than the existent methods. Binglei et al. [22] presented a framework for designing as well as constructing green databases. Firstly, a method of modelling the energy cost of query plans during query processing cantered on their resource consumption patterns was submitted. Then, the plan evaluation principles were studied. Using the cost model as a basis, the evaluation model utilized the trade-ofs betwixt power and performance of plans. Experimental outcomes revealed that, with accurate and reliable statistical data, this framework could achieve considerable energy savings and improve energy efficiency[8].

Jiajia et al. [23] recommended a framework for PGNN queries (PGNN-Probabilistic Group Nearest Neighbor) cantered on granularity classification utilizing ELM in order that the general cost was minimized. The depth classification was intended for the PGNN query, where the options of candidate objects were selected automatically cantered on different queries[12]. Then some useful features were picked for the PGNN query. Next, a DCA (depth classification algorithm) utilizing the Extreme Learning Machine (ELM) was presented, wherein a plurality voting approach was utilized. The outcomes confirmed that the depth set by the ELM classifier decreases the overall price than the default values.

Bin Zhang et al. [24] presented optimization strategies to recur queries in BD analysis. This approach used the MR consistent window slice algorithm for the re-utilization of recurring queries. And, it reduced the redundant data whilst loading input data with the fne-grain scheduling. Next, concerning data scheduling, it designed the MR late scheduling strategy that improved the data processing and it optimized the computation resource scheduling in the MR cluster. The experiment's outcomes on an assortment of workloads showed that the algorithms outperformed the top-notch approaches.

Jafarinejad and Amini [2] offered an approach for the MJQO problem in the BACO database as an extension to the MAX–MIN Ant System algorithm. Experimental evaluations of this technique indicated that preventing the pointless operations and even producing useful meta-data as feedback information to other agents (ants) made the BACO method to find better solutions, approximately 91.5–122.4% faster than previously successful methods.

Sahal et al. proposed a Multi-Query Optimization system called MOTH that utilizes metadata and histograms for slow storage considerations. MOTH investigated opportunities for coarse-grained reutilization based on non-equivalent tuple sizes and non-uniform data distribution. MOTH created three multi-query plans for fully reused-centric opportunities and two multi-query plans for partially reused-based opportunities[12]. The experimental results showed that the three fully reused-centric plans outperformed the Naive Technique plan on average by 40%, 45%, and 50%, respectively. The two partially reused-based plans outperformed the Naive Technique plan on average by 22% and 27%, respectively, on MR.

Proposed query optimization methodology

Organizations maintain different databases to store and also to process BD which is huge in volume and has different data models. Querying along with analyzing BD for insight is critical for business. This paper

enhanced the query optimization process in BD using the ACO-GA algorithm as well as the HDFS map-reduce technique. The proposed work comprises two phases namely,

- BD arrangement
- Query optimization

The block diagram of the proposed methodology is displayed in below Fig. 1,

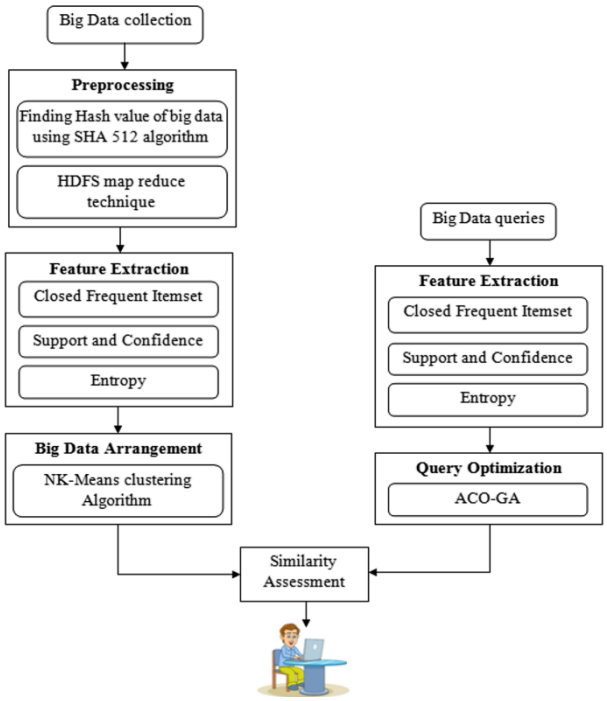

Fig: Block diagram of proposed Methodology

Big Data Arrangement

Arranging big data requires a systematic and efficient approach to organize and store large volumes of data. Here are some common techniques for arranging big data:

Distributed storage: One common approach is to use a distributed file system such as Hadoop Distributed File System (HDFS) or Apache Cassandra. These systems allow data to be distributed across multiple nodes, making it easier to store and access large volumes of data.

Data partitioning: Another technique is to partition data into smaller subsets. For example, data can be partitioned based on time, location, or some other attribute. This makes it easier to process and analyze the data.

Compression: Large data sets can take up a lot of storage space. Compression techniques such as gzip or bzip2 can be used to reduce the amount of storage needed.

Data indexing: Indexing can help speed up searches and queries on large data sets. Indexes can be created on specific attributes, making it easier to search for specific data.

Data normalization: Normalizing data involves organizing it into a consistent format. This can make it easier to compare and analyze data across different data sets.

Data archiving: Finally, data that is no longer needed for analysis can be archived. This frees up storage space and makes it easier to manage the data that is still being used.

Initially, the input data are amassed from the "Hospital Compare Datasets". The hospital compare dataset contains information associated with heart attack, pneumonia, surgery, heart failure, in addition to other conditions. The gathered data are written as,

Ds = {D1, D2, D3,……,Dn}

where Ds denotes the data set Dn represented the n-number of data.

Preprocessing

This phase executed the pre-processing of the input data. Initially, it fnds the HV for all the data utilizing the Secure Hash Algorithm (SHA-512). Then, centered on the HV, the MR process is executed utilizing HDFS. The SHA-512 and HDFS process is explained in the below subsections. 3.1.2.1 Finding the hash value of big data using the SHA-512 algorithm The input data is set at 128-bit length feld. The augmented data is separated into blocks. Then, a 64 bit word is derived as of the current data block using '8'-constants centered on the square-root of the 1st '8' prime numbers. In the succeeding stage, a 512-bit bufer is updated. SHA-512 operation is signifed in Fig. 2, The generated HV of each data block is expressed as follows,

fH(Dn) = { H (D1) , H (D2) , H (D3) , ..., H (Dn)}

where fH(Dn) represents fnd the HV of n-number of data, and H (Dn) denotes the HV of n-number of data.

HDFS

HDFS is a distributed file system that is used to store and manage large data sets in a reliable and scalable way. It is designed to work with the Hadoop MapReduce (MR) programming model, which is a popular way of processing large data sets.One of the key features of HDFS is its ability to quickly transfer data between nodes. This allows for efficient processing of large data sets across a cluster of computers. Additionally, HDFS automatically distributes data across the cluster, making it easier to manage and access data.Another important function of HDFS is its ability to remove duplicate data from the collection. This helps to reduce the amount of storage space needed and improves data processing efficiency.HDFS retrieves data based on the file name and does not change the file once it has been written.

This ensures the integrity and consistency of the data stored in HDFS.

Thus, if any changes are required to be made, then the entire fle must be rewritten. The HDFS has '2' phases for solving the query, for instance, map function as well as reduced function, which is expressed as follows,

H (Dn) = [Pf , Cf]

For all hash data value, the mapping function Pf and the reducing function Cf is processed.

Map () function The preliminary function that exists in the Map/Reduction tool is the Map() function. This function prevails on the master node (MN). It segments the input information or processes it to numerous smaller sub-processes. These sub-processes are further scattered to the worker nodes, which operates on these tiny processes. Then, an acknowledgment is delivered to the MN.

Pf = map(H (4) (Dn))

where Pf is the map() function's output, map() signifes the function which performs the mapping.

Reduce () function The next function, which is important in the Hadoop tool is the reduce () function. This function assembles the comprehensive sub operation results and then combined it for the generation of aggregated decisionbased results delivered as an acknowledgment of the original big demands. The reduce function is denoted as exhibited in the subsequent mathematical equation

Cf = (5) reduce(Pf)

where Pf is the mapped function's output, reduce() is the function that reduces the components and Cf is the reduced set of data.

Feature extraction

After the removal of repeated data, important features such as closed frequent itemset, support, and confidence are extracted from the original data. At last, the support and confidence value is managed cantered on the entropy calculation. The feature extraction steps are briefly explained in the below section.

Closed Frequent Itemset (o (fi)): This is the highest number of occurrences of a particular item in the frequent itemset. The closed frequent is expressed as follows,

o (fi) = {o (f1), o (f2), o (f3), …, o (fn)} (6)

Support (σt): This is the percentage of transactions on the database that encompasses item sets U and V. The support of an association rule U → V is given by,

σt = Support (U → V) = P(U ∪ V) (7)

Confidence (Ae): This is the percentage of transactions on the database with itemset U that as well encompasses the itemset V. The confidence is computed utilizing the conditional probability that is further conveyed in respects of itemset support. The equation for the confidence is given by,

Ae = Confidence (U → V) = P(V/U) = P (U ∪ V) /P(U) (8)

Entropy calculation: In this subsection, the support and confidence are managed based on entropy calculation. This entropy measure is related to the variance of a probability distribution. Entropy is assayed for the data sets utilizing Eqs. (9)

$$\text{Entropy}(\sigma_t) = -\sum_{t=1}^{n} P\sigma t \; log2\{P(\sigma t)\} \quad (9)$$

where P(σt) is the probability of getting the nth support and confidence values, when randomly selecting one from the set. The entropy contains a number of values which are the minimum and maximum value. These values are then inputted to the normalized K-Means algorithm for the BD arrangement

Big data arrangement using normalized K-Means (NKM) algorithm This section performed a BD arrangement using a normalized K-Means algorithm. Here, it initially takes the entropy value of the support and confidence features that consists of minimum and maximum values. So, perform normalization first, it can help get the useful and right information about the entropy value of the support and confidence features.

The normalization is expressed as:

$$N \propto = \frac{\text{D}-\text{Dmin}}{Dmax-Dmin} \quad (10)$$

where N∝ denotes the normalized value Dmax and Dmin represents the maximum and minimum value of the number of data D. Then, generate the number of clusters and initial centroids using K-Means. K-Means clustering technique is used for grouping N values into c number of clusters. The algorithm is utilized to resolve the clustering issue for finding the optimum value of the objective function. This algorithm finds the Euclidean distance betwixt the normalized value and the center of the cluster. The smaller value of the objective function means better clustering results.

Result and discussion

This section analyzed the performance shown by the proposed query optimization approach utilizing the NKM methodology. The system performance is analyzed and contrasted to the existing techniques centered on the data size. Here, the data sizes range from 10 to 50 MB is considered. This proposed query optimization methodology is employed in the working platform of JAVA.

Performance analysis Here, the performances shown by the proposed Normalized K Means (NKM) algorithm and the existing Fuzzy C Means (FCM) along with K-Means algorithm are contrasted in respect of statistical metrics:

- precision (Pe)
- sensitivity (Se)
- specificity (Sp)
- recall (Rc)
- accuracy (Ac)
- F-Measure (Fs)
- retrieval time (RT)
- execution time (ET)
- Memory usage

The performance comparison of the proposed NKM and the existent methods is evinced in Tables 1, 2, 3 and 4. Discussion Table 1 contrasted the performances proffered by the proposed NKM with the existing K-Means and FCM centered on sensitivity, accuracy, and specificity. The performance varies grounded on the data sizes ranging as of 10 to 50 MB. For all the considered data sizes, the proposed NKM clustering profers the accuracy within the range of 91 to 96. When the data size is 50 MB, the proposed NKM clustering has 96.25% accuracy. Similarly, when the data size is 40 MB, the proposed NKM has 92.46% specificity and 93.45% sensitivity. But the existing K-Means and FCM oFfer 86.34% specificity and 86.37% sensitivity which are lower when contrasted to the proposed NKM. For all the considered file sizes, the proposed NKM showed excellent performance than the existing methods. Thus, it is deduced that the proposed NKM has high-level performance. Graphically, this comparison can well be elucidated using Fig. 4.

Discussion Table 2 contrasted the performances shown by the proposed NKM and the existing FCM and K-Means in respect of recall, precision, along with F-Measure. For 30 MB data size, the proposed NKM has 92.45% precision, 0.21% recall and 89.72% F-Measure. For the same data size, the existing K-Means achieves90.67% precision, 87.34% recall and 87.99% F-Measure and the existing FCM obtained 87.12% precision, 84.89% recall, and 84.44% F-Measure which are lower than the proposed NKM clustering method. Similarly, the proposed NKM achieves better performance for remaining data size such as, 10 MB, 20 MB, 30 MB, and 50 MB. The F-Measure metric is the integration of recall and precision metrics. If the system has high F-Measure, then the system is denoted as a good system. In this way, as the proposed NKM provides better F-Measure than the existent K-Means and FCM, it is regarded as a good system. Graphically, Table 2 can well be elucidated utilizing Fig. 5.

Table 1 Demonstrate the performance of the proposed NKM with the existing K-Means and FCM in terms of accuracy, specificity, and sensitivity

Data size in MB	Proposed NKM			Existing K-Means			Existing FCM		
	A_C	S_P	Se	A_C	S_P	Se	A_C	S_P	Se
10	91.23	83.53	81.56	87.23	77.87	78.43	83.12	73.43	72.56
20	92.45	87.89	85.43	90.23	80.95	79.23	84.69	78.78	74.45
30	94.67	88.90	88.78	91.89	83.56	83.45	86.25	81.34	80.79
40	95.56	92.78	93.45	92.88	86.34	86.37	88.67	83.53	84.23
50	96.78	94.67	96.12	93.23	88.84	91.23	90.34	86.15	89.45

Table 2 Comparison table for proposed NKM with K-Means and FCM based on precision, recall, and F-measure

Data size in MB	Proposed NKM			Existing K-Means			Existing FCM		
	P_e	R_c	F_s	P_e	R_c	F_s	P_e	R_c	F_s
10	89.23	87.53	87.56	86.23	82.87	88.43	83.12	73.43	82.56
20	91.45	88.89	88.43	88.23	85.95	89.23	84.69	78.78	84.45
30	92.67	90.90	89.78	90.89	87.56	83.45	86.25	81.34	80.79
40	94.56	92.78	91.45	91.88	89.34	96.37	88.67	83.53	89.23
50	95.78	93.67	92.12	92.23	92.84	91.23	90.34	86.15	90.45

Table 3 Illustrate the performance of the proposed NKM with the existing K-Means and FCM based on retrieval time, execution time

Data size in MB	Proposed NKM		Existing K-Means		Existing FCM	
	R_T	E_T	R_T	E_T	R_T	E_T
10	265	120	453	132	682	148
20	784	141	1505	154	2145	179
30	1495	187	3092	214	4655	247
40	2442	212	5171	244	6543	287
50	3699	281	7677	315	8877	349

Table 4 Memory usage analysis

Data size in MB	Proposed NKM	Existing K-Means	Existing FCM
10	5333,475	5888,654	6344,455
20	5777,633	6124,456	6755,434
30	6035,000	6577,343	7132,346
40	6477,867	6988,556	7344,676
50	6689,554	7145,667	7544,232

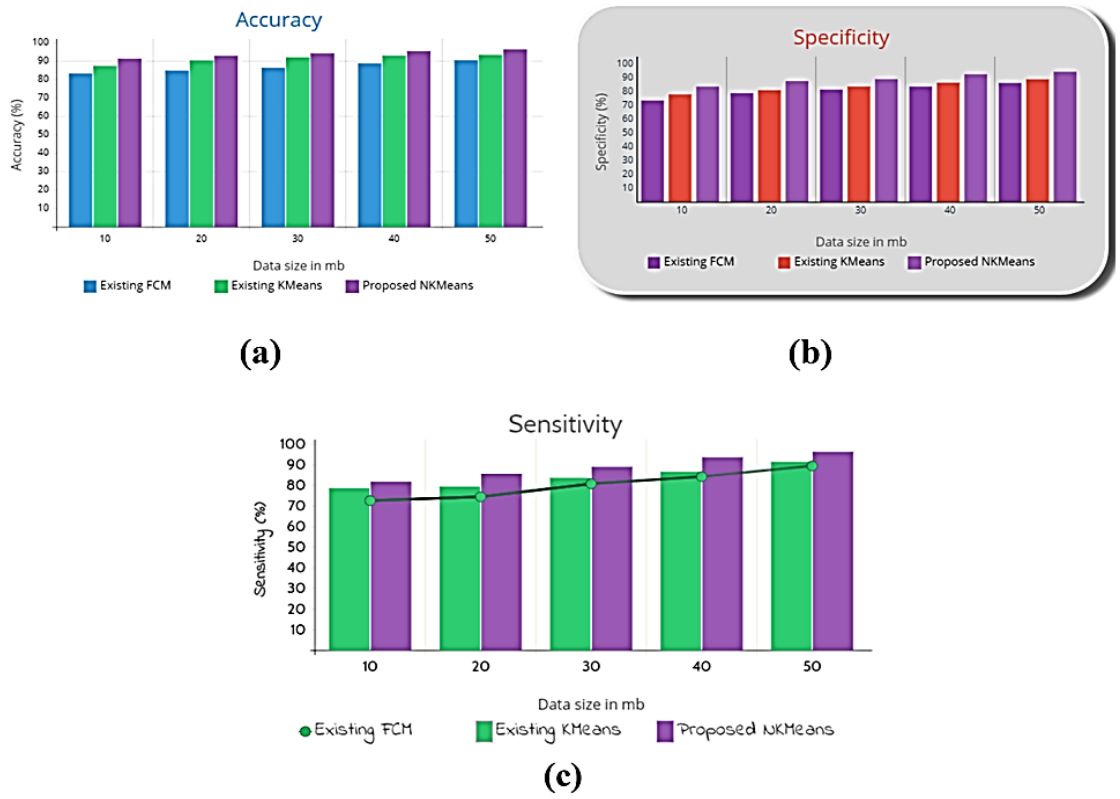

Fig. 4 Comparative analysis for the proposed NKM with the K-Means and FCM in terms of (a) accuracy, (b) specificity and (c) sensitivity

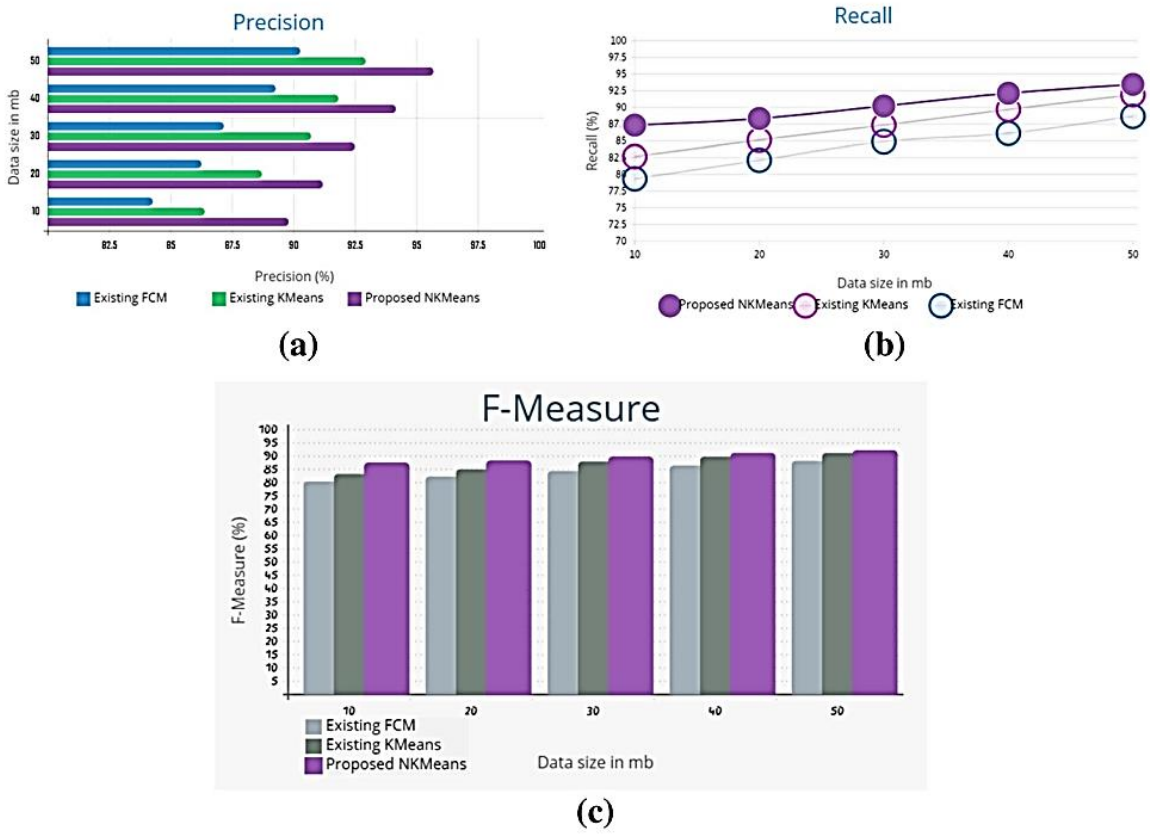

Fig. 5 Demonstrate the performance of the proposed NKM with the existing K-Means and FCM in terms of a precision, b recall and c F-Measure.

Discussion Table 4 contrasted the performances proffered by the proposed NKM and the existing FCM and K-Means centered on memory usage (in kilobytes). For 40 MB data size, the proposed NKM occupies 6477867 kilobytes memory storage during execution. But the existing K-Means and FCM occupy 7145667 kilobytes and 7544232 kilobytes. Similarly, for the other remaining data size, the proposed NKM occupies less memory than the existing methods. Thus, it is deduced that the proposed NKM acquires high-level performance than the existent approaches. Graphically, Table 4 can well be elucidated utilizing Fig. 7

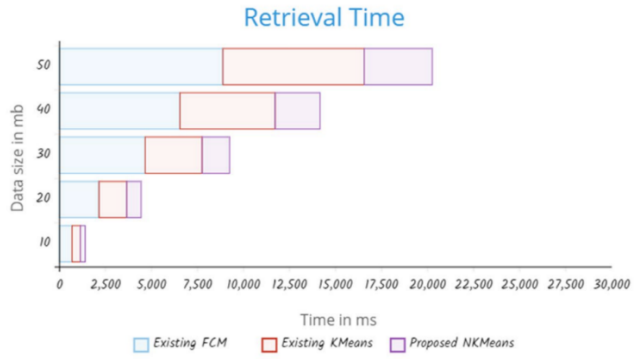

Fig. 6 Comparative analysis for the proposed NKM with the existing K-Means and FCM based on (a) retrieval time and (b) execution time

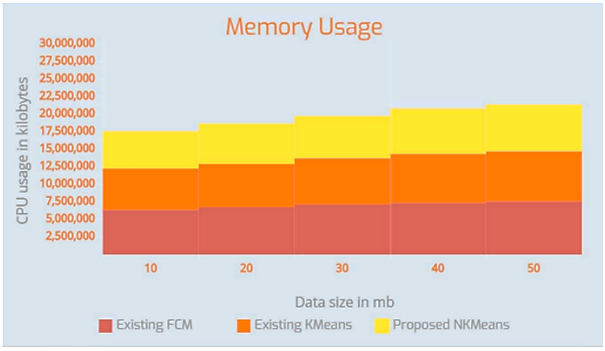

Fig. 7 Memory usage analysis graph

Conclusion

Query optimization in BD becomes a propitious research direction on account of the popularity of huge data analytical systems like the Hadoop system. This paper proposed an improved query optimization process in BD using the ACO-GA algorithm and HDFS map reduce technique. The proposed work contains two phases namely, the BD arrangement phase and query optimization phase. The proposed system's performance was analyzed using data size. The file size ranges from 10 to 50 MB. The performance analysis showed that the proposed system has 96.25% accuracy for 50 MB data size which is higher on considering the other existent methods namely FCM as well as K-Means. The comparison results corroborated that the proposed work provides higher accuracy and takes less time for retrieving the query. Also, the proposed system occupies less memory storage. Hence, this proposed system is better when compared with the existent system. In the future, the efficiency of this system can well be enhanced further by including feature selection to reduce the retrieval time and also by using advanced optimization algorithms.

References

[1]. Rawat, J.S., Kishor, S., Kumari, M.: A survey on query optimization in cloud computing. Int J Adv Technol Eng Sci 4(10), 2348 (2016)

[2]. Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C., Huang, Y.: SHadoop: improving mapreduce performance by optimizing job execution mechanism in hadoop clusters. J Parallel Distrib Comput. 74(3), 2166–2179 (2014)

[3]. J Wolf, D Rajan, K Hildrum, R Khandekar, V Kumar, S Parekh, and KL Wu 2010, "Flex: A slot allocation scheduling optimizer for mapreduce workloads", In Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware, Springer-Verlag, pp. 1-20

[4]. Barba-González, C., García-Nieto, J., Nebro, A.J., Cordero, J.A., Durillo, J.J., Navas-Delgado, I., Aldana-Montes, J.F.: jMetalSP: a framework for dynamic multi-objective big data optimization. Applied Soft Computing 69, 737–748 (2018)

[5]. Song, J., Ma, Z., Thomas, R., Ge, Yu.: Energy efficiency optimization in big data processing platform by improving resources utilization. Sustainable Computing: Informatics and Systems 21, 80–89 (2019)

[6]. Mahajan, D., Blakeney, C., Zong, Z.: Improving the energy efciency of relational and NoSQL databases via query optimizations. Sustainable Computing: Informatics and Systems 22, 120–133 (2019)

[7]. Rini John, and Nikita Palaskar, "A survey of various query optimization techniques", International Journal of Computer Applications, vol. 975, pp. 8887

[8]. Roy, C., Pandey, M., Rautaray, S.S.: A proposal for optimization of data node by horizontal scaling of name node using big data tools. In: Proceedings of the 3rd International Conference for Convergence in Technology (I2CT), IEEE, pp. 1–6 (2018)

[9]. Dwivedi, J., Tiwary, A.: Big data analytics: an overview. Int. J. Sci. Technol. Res. 5(07) (2016)

[10]. Elham Azhir ,Mehdi Hosseinzadeh , Faheem Khan , and Amir Mosavi : Performance Evaluation of Query Plan Recommendation with Apache Hadoop and Apache Spark. 10(19), 3517(2022)

[11]. Deepak Kumar, Vijay Kumar Jha: An improved query optimization process in big data using ACO-GA algorithm and HDFS map reduce technique. Springer

Science+Business Media, LLC, part of Springer Nature (2020)

[12]. Song, J., Ma, Z., Thomas, R., Ge, Yu.: Energy efciency optimization in big data processing platform by improving resources utilization. Sustainable Computing: Informatics and Systems 21, 80–89 (2019)

[13]. Panahi, V.; Navimipour, N.J. Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. *Concurr. Comput. Pract. Exp.* 2019, *31*, e5218.

[14]. Pasquale Salza, Filomena Ferrucci. Speed up genetic algorithms in the cloud using software containers. (2019)

[15]. Mahajan, D., Blakeney, C., Zong, Z.: Improving the energy efciency of relational and NoSQL databases via query optimizations. Sustainable Computing: Informatics and Systems 22, 120–133 (2019)

[16]. Bao, C., Cao, M.: Query optimization of massive social network data based on hbase. In: Proceedings of the IEEE 4th International Conference on Big Data Analytics (ICBDA), pp. 94–97 (2019)

[17]. Sahal, R., Nihad, M., Khafagy, M.H., Omara, F.A.: iHOME: index-based join query optimization for limited big data storage. J. Grid Comput. 16(2), 345–380 (2018)

[18]. Rawat, J.S., Kishor, S., Kumari, M.: A survey on query optimization in cloud computing. Int J Adv Technol Eng Sci 4(10), 2348 (2016)

[19]. Kiranjit Pattnaik, Bhabani Shankar Prasad Mishra: A Review on Parallel Genetic Algorithm Models for Map Reduce in Big Data. International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 IJERTV5IS080400 Vol. 5 Issue 08, August-2016

[20]. Panahi, V.; Navimipour, N.J. Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. *Concurr. Comput. Pract. Exp.* 2019, *31*, e5218.

[21]. Rani, S.; Rama, B. MapReduce with Hadoop for Simplified Analysis of Big Data, International Journal of Advanced Research in Computer Science, May-June 2017, Volume 8, No. 5, ISSN No. 0976-5697, pp. 853-856.

[22]. Joseph, C.W.; Pushpalatha, B., A Survey on Big Data and Hadoop, International Journal of Innovative Research in Computer and Communication Engineering, ISSN(Online): 2320-9801, March 2017, Vol. 5, Issue 3, pp. 5525-5530.

[23]. Ferrucci, F., Salza, P., and Sarro, F. (2016). Using Hadoop MapReduce for Parallel Genetic Algorithms: A Comparison of the Global, Grid and Island Models - Appendix. https: //doi.org/10.6084/m9.figshare.5091898.

[24]. Fu, W., Menzies, T., and Shen, X. (2016). Tuning for Software Analytics: Is It Really Necessary? Information and Software Technology, 76:135–146.

[25]. Salza, P., Ferrucci, F., and Sarro, F. (2016a). Develop, Deploy and Execute Parallel Genetic Algorithms in the Cloud. In Genetic and Evolutionary Computation Conference (GECCO), pages 121–122.

[26]. Salza, P., Ferrucci, F., and Sarro, F. (2016b). Elephant56: Design and Implementation of a Parallel Genetic Algorithms Framework on Hadoop MapReduce. In Genetic and Evolutionary Computation Conference (GECCO), pages 1315–1322