Optimizing Business Insights: An Enhanced Large-Scale RDF Query Processing And Loading Speed On Big Data

[1]V.Naveen Kumar, [2]Dr Ashok Kumar P S

[1]Research Scholar, Don Bosco Institute of Technology, Affiliated To Visvesvaraya Technological University, naveenvipparla@gmail.com.
[2]Research Supervisor, Dept. of CSE, ACS College of Engineering, Affiliated To Visvesvaraya Technological University, ashokdbit2017@gmail.com.

Abstract

The generic access pattern is the Resource Description Framework (RDF), while SPARQL is a widely adopted query processing framework for collecting information. Due to its information adaptability and data modeling, this subject is presently receiving greater attention. The basic RDF structure is frequently used to supply web data through a spectrum of uses, including social networks, organizations' search engines, and other databases. A key component of semantic web managing data is the effective deployment of large-scale RDF querying. With the quicker growth of RDF data, storing all RDF triples in a particular node in larger datasets is frequently impossible. Due to this, researchers are concentrating on SPARQL query processing in distributed systems, particularly using the Hadoop system. The effectiveness of SPARQL query processing is increased by the application of the Map Reduce methodology. In this paper, we developed an Improved Large Scale RDF (ILS-RDF) for information partitioning to improve query processing and loading in an effective way. With the creation of ILS-RDF concepts and representations, knowledge transfer transitioned from the ordinary web to the semantic web. RDF is a widely used format for representing and querying linked data, and optimizing its processing on big data platforms requires careful consideration of data storage, query optimization, and integration with business tools. The proposed method uses low-grade indexing and run-time indexing for the phrases in the search to speed up data fetching and cut down on nodal connection latency.

Keywords: Improved Large Scale RDF; Large datasets; Cloud storage; data partitioning; SPARQL; Big Data.

1. Introduction

The advantages of RDF data models regarding flexibility and compatibility are being felt by several companies. Due to the interoperability, getting RDF data from various sites is a complicated task [1]. RDF is a triple-based notation that is typically graph-based and comprises subjects, references, and premises. A collection of triples that are described by directed graphs may be found in the RDF [2]. Information from RDF documents that have been saved and retrieved using the SPARQL language. The triple patterns, also known as the Basic Graph Pattern, are the basis for SPARQL searches [3]. The BGP is used to extract the information from the RDF documents. An additional sort of graph, known as a query graph pattern, is included in SPARQL.

RDF data administration is the primary technique underlying a semantic web repository. This method enables the retrieval and storage of information from the network [4]. Both academics and businesses are now researching RDF data management. The architecture developed to handle RDF information is known as Hexastore [5]. However, the single-system techniques became problematic due to the rising RDF complexity. Consequently, a stand-alone system is necessary to handle the aforementioned problem [6]. Combined with the adoption of traditional clustering techniques, other ways have been created to deal with the scattered RDF data. These techniques work well for improving the efficiency of either query processing or information storage, however not alike [7]. The technique for speeding up query execution and information loading is presented in this paper. This was accomplished using the single component indexing and run time indexing techniques created for the proposed technique's RDF data encoding and data fetching mechanisms. The RDF query's assertions are transformed into numerals to minimize the model's communication burden.

The majority of RDF methods for analyzing data were created using batch processing models. These systems architectures are made to handle small datasets. These approaches don't use any encoding or splitting strategies as

they immediately execute the query over the original data [8–9]. This causes complicated systems to have a large network latency. Subject-predicate-object triples are represented as graphs in SHARD storage of RDF [10]. For graph storing, it employs a distributed computing technique. HDFS is used to hold information. Every triple store line is saved as a different section in a text document to enable data persistence [11]. While this method of keeping triple data in flat files uses a large amount of memory, it automates resiliency for data replication and makes MapReduce processes possible over numerous processors [12]. The RDF data takes some time to retrieve using this approach.

YARS2, Indexing, and searching capabilities for graph-structured data are provided via a distributed repository. Local indexes are built even before the information is added. The original data is condensed into documents that are subject, predicate, and context-ordered (SPOC). After that, a multi-way fusion is used to integrate all of the documents. Every SPOC file has an inverted index constructed for it [13]. Three techniques—random, hash, or range—are used to divide the triples [14]. The indexes comprise connect, quadruple, and keyword indexes. Indexes, which enable multi-threaded inquiry and responses between the query processors and index administrator, are used to perform inquiries.

The efficiency of query processing can be enhanced by using graph-based partitioning methods. To handle massive graphs, Sedge is used [15]. Vertex-based divisions are used to segment graphs. Replications and overlapping are both enabled. The graph is segmented again using complementary partitions, resulting in interior links along the initial partition vertices. To distribute the burden, it implements on-demand splitting, in which existing divisions are duplicated or brand-new, overlapping partitions are generated [16]. The workload is distributed using a two-level partitioning approach.

2. Related Work

RDF simulates a storage graph for RDF. In the primary value storage, the graph is arbitrarily divided into sections. The standard sharing algorithm uses node-id hashing. It allows for arbitrary queries to the RDF graph. To prevent connections, it has created in-memory graph analysis techniques [17–18]. The method utilized to discover the

similarities appears excessive. On graphs, it is simple to do graph analyses including random walks, regular expression queries, reachability enquires, proximity oracles, and community searches [19]. The semantic Hash partitioning strategy was employed by investigators [20]. Segmentation makes use of the entire Hadoop framework. The cluster's master node is in charge of segmentation, and the slave node is in charge of keeping the pieces together.

The task scheduler creates the segments and delivers the interim results of a combined inquiry, while the slave has the retrieval of data that stores the information [21]. The semantic partitioner is used by the job tracker to locate the nodes where the segments will be stored, and the master node contains a name node that holds summaries of the divisions. Either the price of transmission or the complexity of the network will rise as the graph is partitioned [22]. To organize digital data based on the best choice of inbound inquiries, the researcher identified an adaptive indexing strategy. Many methods divide the request into the RDF data using the graph partitioning methodology [23].EAGRE was created to reduce the cost of the input and the output. The graph was divided using the MiniCut technique by the researchers.

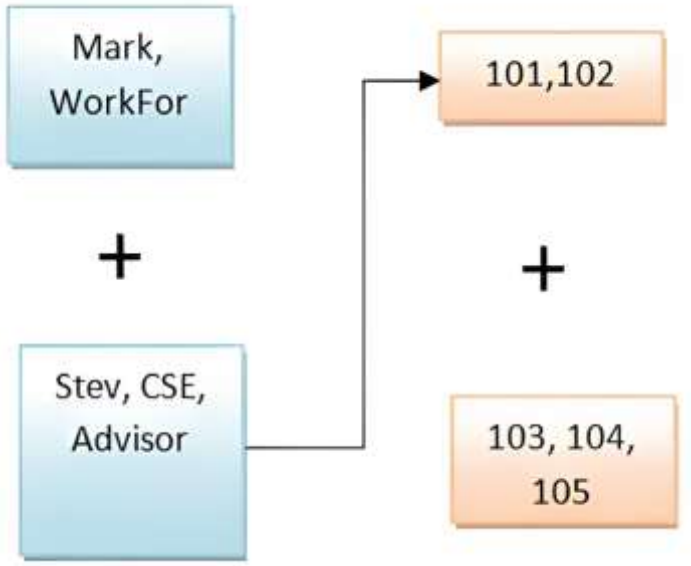
The graph information is split using the METIS method using H-RDF-3X. It takes advantage of the K-hop method, which enables communication-free execution of queries inside the k-radius [24]. However, designs like "SPARK" and "Hadoop++" are unable to survive the complexity of several computations employing the MapReduce method [25]. Scala is the foundation of the initial cluster computing device known as Spark. The durable dispersed dataset that is divided up over several machines is its major component [26]. It shows the user for useful programming platform. The durable dispersed datasets enable parallel computing [27].On this RDD, simultaneous processes are carried out. An RDD is represented by Scala objects, which can be produced in several ways, including from an HDFS file, by parallelizing the divisions across several instances, by converting the RDDs, and by changing the durability of an established RDD [28]. Minimize, aggregate, and for each are just a few of the concurrent processes that may be done on the dataset. The components of the dataset are combined by minimizing, sent to the driver programme by gathering, and iterated by for each in the user-defined procedure.

By utilizing the underlying Hadoop architecture with HDFS for collection and the MapReduce programming style for query execution simultaneous, Hadoop++ presents a novel method [29]. Units of the information are saved after being divided laterally. The Hadoop publish technique is used for archiving. The customer asked the namenode how many vacant blocks are accessible. The customer keeps the information in the data node, and the namenode provides the block addresses. Hadoop's minimum recurrence ratio is three. Consequently, the segments are triple-mirrored in the data nodes [30]. Trojan index, a brand-new indexing system, is also introduced. This is created concurrently with large data and is unrelated to DBMS indexing methods. The researchers' proposed SHAPE uses the semantic hashing algorithm to divide the RDF data. It adopts the H-k-hop RDF-3X's strategy, the simple partitioning method, and the idea of semantic hash segmentation.

3. Proposed approach

The vast amount of ILS-RDF data highlights the value of persistence. An improved design is needed for storage. The prior studies covered the dispersed environments storage structure for ILS-RDF data. The information must be retrieved after it has been stored to be queried or updated. It will require a long time to retrieve or write the information immediately because it is in tera or petabytes. To facilitate speedier accessibility, DBMSs create indexes for the information. Indexes are supplementary accessible entities that contain the result of the indexing column and a block pointer. Information may be quickly found by using indexes. The speed of the request is enhanced by an index search that is part of the execution plan [31]. Indexing is the following challenging assignment. The second major issue that arises when information is sought is how to access it. Therefore, query processing must be carried out effectively. Since ILS-RDF information includes large strings, it is quite practical to translate them into manageable ones. The strings in ILS-RDF data are so big, performing queries on them requires a lot of storage, uses a lot of network capacity, and has poor query processing. To increase efficiency, the ILS-RDF texts are transformed into linked IDs in the proposed method and to develop APIs and web services that allow business applications to interact seamlessly with the RDF data store.

The proposed system uses transformation procedures to describe the ILS-RDF assertion, converts concepts in the RDF assertion to variables, and encodes inputs using a dictionary-based representational approach. Additionally, Integrate RDF data with existing business technologies, such as CRM systems, data warehouses, and reporting tools, to provide a holistic view of data. The proposed technique, shown in Figure 1, initially divides the information into equal-sized blocks and distributes it to the member nodes for calculation**.**



-

Figure 1 Similar Size Partitioning

The following is a list of the entire encoding process. Every ILS-RDF assertion is first processed and then broken up into a variety of distinct elements including premise, condition, and argument. Find the supplicate phrases in the generated assertions, and then use the filter module to eliminate them immediately. Furthermore, categorize each phrase according to its hash value using the RS hash technique described in. Algorithm 1 provides the technique. In this case, the entire amount of groups and member nodes should match. For sorting the nodes and grouping them, this algorithm uses the hash value modulus [32]. Take the instance in Figure 1 where the categories are partitioned based on comparable dimensions to better comprehend works for the initial category assigned to the second node. The steps are shown in Figure 2.

The firm's distinct words are then moved to the central node in a subsequent stage, maintaining the individual term's integer number. The sequential approach is used

throughout the complete system of encoding the phrases. The regional thesaurus will be checked for a unique ID for every phrase. The ID is obtained immediately if the word has already been entered into the thesaurus. If the word is brand-new to the thesaurus, the phrase will receive the proper new ID. The words' IDs are kept up to date at several nodes. Once each word has been given an ID, the central node keeps track of all of those IDs. Attributing IDs to the elements in the member node1, for illustration, as depicted in Figure 3.

Algorithm 1 Replication Server Hash Function in ILS-RDF

1: a ← 63689
2: b ← 378551
3: hash ← 0
4: i ← 0
5: while i =/ length do
6. hash ← hash ∗ a + ord(str[i]) a ← a ∗ b
7. i + +
8: end

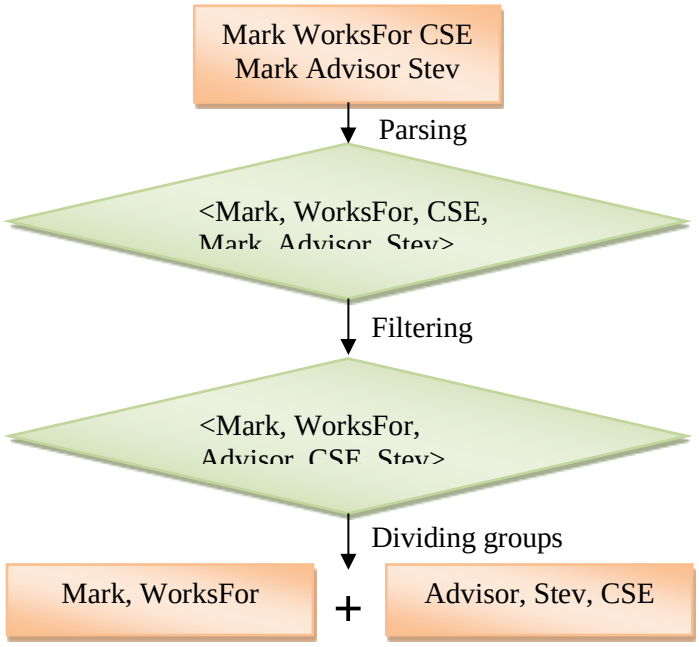


Figure 2 Encoding Procedure

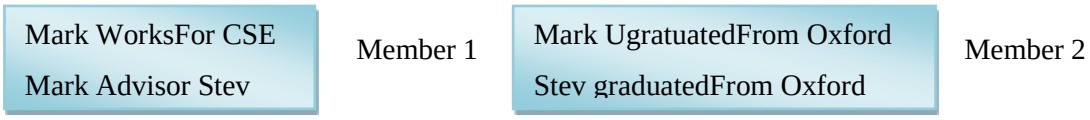


Figure 3 Partitioned into Member

All of the assertions delivered to the member nodes will be given separate IDs and returned. Figure 4 shows their form.

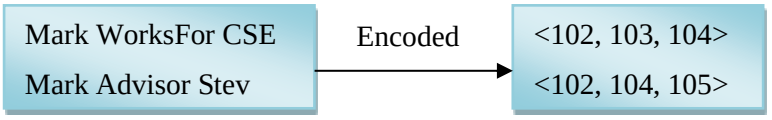


Figure 4 Encoding

The filter procedure is used throughout the complete system to locate the distinctive elements in the assertions and deliver them to the central node. This method is used for all words and will handle ILS-RDF queries with the utmost efficiency. The cost of local node calculation and communication will be significantly reduced as a result. Delivering the words and returning their IDs allows for two-way interaction under the proposed method, though. This processing method is simple to operate and highly effective for managing semantic web information.

Preparation for regularization was performed independently by other means to simplify the load and query processing:

$$S_x^* = \frac{S_x}{\sum_{x=1}^{n} S_x} \quad (2)$$

$$Z_x^* = \frac{Z_x}{\sum_{x=1}^{n} Z_x} \quad (3)$$

$$D_x^* = \frac{D_x}{\sum_{x=1}^{n} D_x} \quad (4)$$

Researchers break things up into three groups according to their length. When work has been assigned to D_x^*, it is assigned to this group; however, it is assigned to another group until all activities are divided into three groups. Researchers could use activity categorization to focus on resource planning. These SA swallows were installed inside that cloud simulator as follows:

$$SA_x = \left(sa_x^1, sa_x^2, \ldots\ldots, sa_x^n, \ldots\ldots\ldots, sa_x^d\right) \quad (5)$$

$$\forall x = 1 \text{to } 25 \text{ and } n = 1 \text{ to } 10$$

This quality procedure makes it possible to verify the

efficiency of this same sparrow region. The second sparrow is filled with OSS, and these subsequent sparrows were chosen based on the best efficacy score. It was dependent on its cloud-free bandwidth but also on Big data processing and transmission speed cost.

4.1 Single-level Indexing

The proposed technique creates the single-level indexing L1 for every sentence in the node when the ILS-RDF statement translation is complete. This will aid in employing index words to store the information. The proposed method, which adopted vertical splitting for information splitting in the nodes, is highly quick in responding to the query [33]. The proposed method created the tables for different access patterns. The regional data set Xi is stored as triples for each member node Si, enabling the search feature described beneath. S, P, and O stand for the premise, adjective, and argument in this sentence:

If it is p,
Return s.o where (s, p, o belongs to Xi). // P-index
If it is s and p,
Return o where (s, p, o belongs to Xi). // PS-index
If it is o and p,
Return s where (s, p, o belongs to Xi). // PO-index

It can be seen from the aforementioned search features that every search function uses a criterion that is specified and explained in the above steps. The assertions must be expressed as real numbers and converted the numbers into phrases to facilitate comprehension.

4.2 Query Processing

The single-level indexing L1 with a series of search queries and several join procedures is used to execute SPARQL queries. The single-level indexing makes it simple to see the outcomes of the expressions in the nodes. Assume the Q1 in Figure 5, which is analyzed using vertical tables and the predicates advisor and graduated From. It is segregated using the comparable dimension partitioning method shown in Figure 6. The parameters' ties to one another? <?student, ?professor > and <?professor, ?university >The stacked tables at the node make it simple to identify the node.

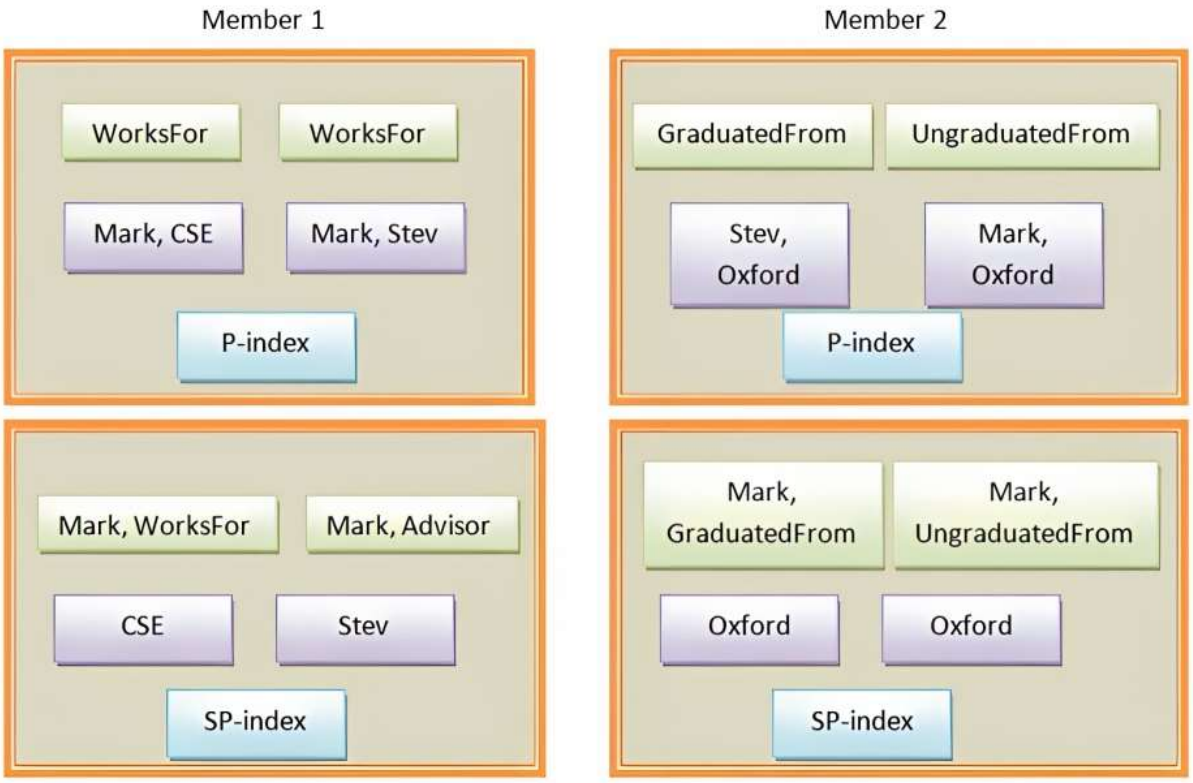


Figure 5 Single-Level Indexing

<?student, ?professor>
<?professor, ?university>

Member 1
<Mark, Stev>

Member 2

< Stev, Oxford>

Figure 6 Partitions at Nodes

The join function will be used to consolidate the outcomes once this procedure has been carried out concurrently for all of the nodes. Therefore, a concurrent hash join process is used to accomplish the join operation, which involves doing a local join first, then spreading the intermediate data to all of the nodes before sending the outcome to the central nodes. The binding parameter professor is used to execute a local join operation among node1 and node2 in the instance earlier, and Figure 7 illustrates the entire join procedure.

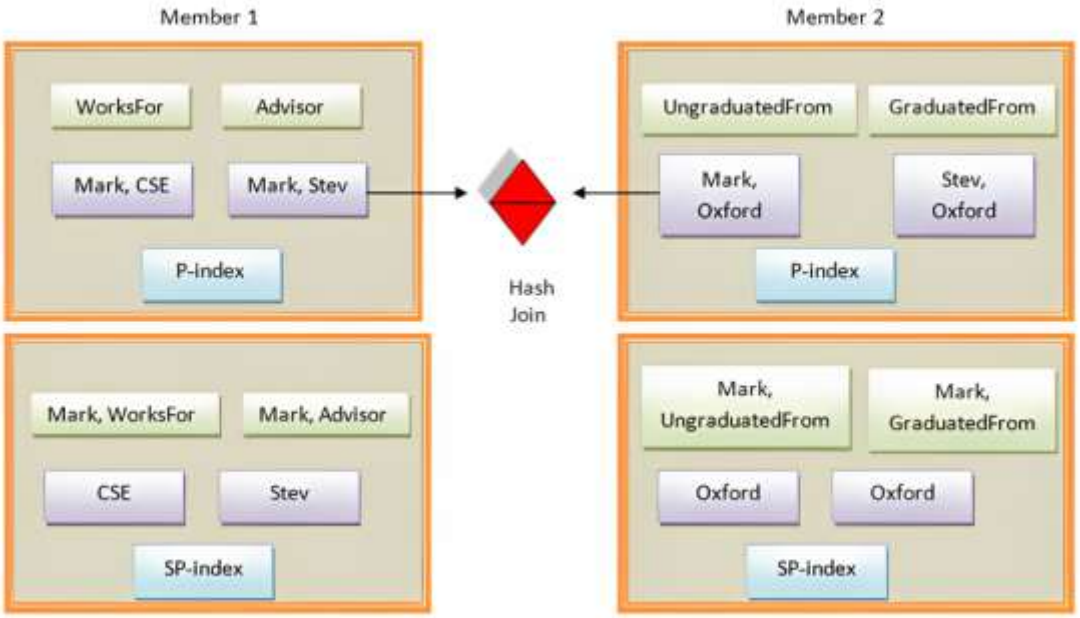


Figure 7 Hash Join with Single Level Index

The retrieval procedure is efficient with the response time at every node. This is because the proposed technique only looks for matching index tables in the L1, not elsewhere. For example, finding the vertical table graduated From by the triples and returning the outcome in the necessary time. However, the dissemination procedure must be carried out by delivering the preliminary findings to each triple to conduct the joint operation. This increases the communication cost and bandwidth while requiring inter-process interaction between the nodes. As a consequence, the proposed solution employs runtime referencing (L2, L3,...Ln) depending on the interim outcomes acquired by the query.

The procedure of a query execution plan is comparable to the technique of runtime indexing. In SPARQL, the approach of parsing and extending is used to assess the query. The query implementation strategy of ILS-RDF using SPARQL was employed in the proposed manner and the runtime operation is shown in Figure 8 alongside Algorithm 2 for query processing and runtime indexing.

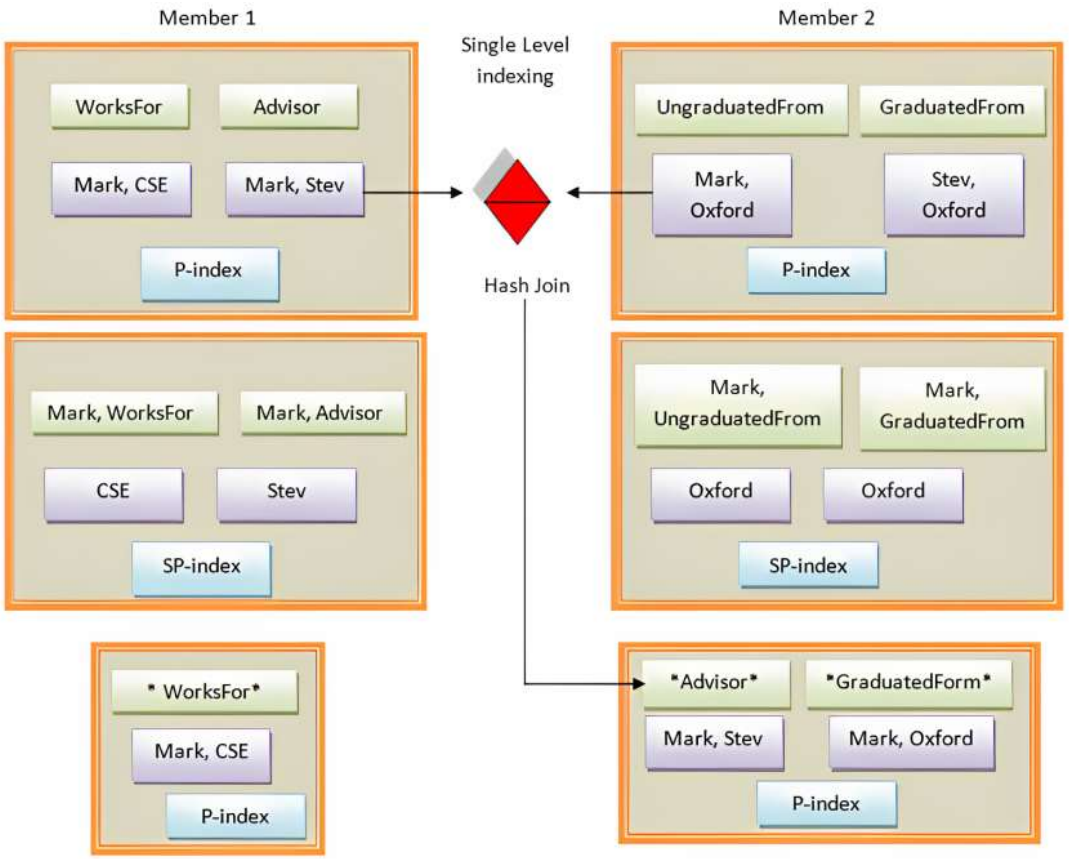


Figure 8 Hash Join with Runtime Index

Conversion and analysis of the SPARQL query into tuple calculus comes next. Additionally, the tuple calculus is used to build the join tree. To demonstrate joins, let's use a query with three or four parameters. The instance used is the search covered in Task 3. There are three independent variables in the foregoing: professor, student, and university. The search seeks to locate the professor, student, and university where the professor teaches for the CSE department and acts as the student's mentor. It also seeks to locate the student's undergraduate institution. The ILS-RDF with the SPARQL prefix and parameters have been omitted from the response for readability.

The following is the tuple calculus notation. The letters p, s, and u stand for the parameters professor, student, and university, correspondingly. The join tree for the aforementioned three triples will be using the tuple calculus. The flowchart described in Figure 9 with the triples P1, P2, and P3 acting as nodes and the connections as edges. The request contains three triples. The above query's inquiry implementation strategy comprises:

At every node, implement filter criteria accordingly. Therefore, P1's CSE has finished its function.

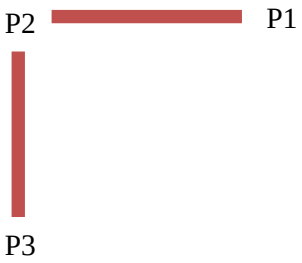


Figure 9 Query Graph

4. Experimental Evaluation

The Big data to assess how well the proposed technique performed. When compared to other techniques already in use, such as RDF-3X, the effectiveness of the new method is evaluated. Performance indicators for the proposed technique include loading time and query response time. The starting arrangement for the 2 kinds of instances is 32GB RAM, 8 cores, and 4 EC2 computational units, as well as 64GB RAM, 16 cores, and 8 EC2 computational modules. Every EC2 computational unit has the same processing power as a 20.0 GHz 2016 Xeon CPU. This collection includes the ILS-RDF-based ontology for the university domain. In addition, 14 queries with various attributes are included in this dataset. Implement monitoring tools to track query performance, system resource utilization, and bottlenecks. This enables proactive identification and resolution of performance issues. Table 1 provides the dataset dimensions that we took into account when evaluating the query.

Table 1 Dataset properties

Organizations	Triples	Size
400	55035263	9.4(GB)
800	110695322	21.8 (GB)

In Tables 2 and 3, the loading times of the information for ILS-RDF and SHARD are listed. For a dataset of 400 organizations, the proposed method takes approximately 454 seconds, and for a dataset of 800 organizations, it takes about 965 seconds. When contrasted with the other approaches, the proposed model's efficiency is observed to be higher. This is a result of the simultaneous loading procedure and the low indexes. Additionally, designing a

scaling strategy that allows the system to handle increasing data loads and query demands. This could involve vertical scaling (adding more resources to existing nodes) or horizontal scaling (adding more nodes to the cluster).

Algorithm 2 Query Processing and Runtime Indexing

Input: Q, L, R// Q: Executed Query, L: Index runtime

Output: R: Results

Run time Indexing

Step 1: L1 can be created// Single Level Indexing

Step 2: for i -> 1 to n

{ Step 3: If Ri of Q originates in L1 then

{ Step 3.1: Ri sends to all nodes}

else

{Step 3.2: Calculate Q}

Step 4: If Ri == indexable

{Step 4.1: L.index(Ri)}

}Step 5: Join (R)

Query Processing

Step 1: For Every Q do

{Step 2: Run the Query plan

Step 3: Compute (R) }

Table 2: Loading time of 400 organizations dataset

Method	Size of Memory	Time is taken for loading	Throughput (Triples/sec)
RDF-3X	20.8	12486	488.5K
SHARD	23.7	1026	3789.1k
Proposed	9.4	656	4125.7k

Table 3 Loading time of 800 organizations dataset

Method	Size of Memory	Time is taken for loading	Throughput (Triples/sec)
RDF-3X	37.4	34654	556.3k
SHARD	48.5	3024	975.0k
Proposed	30.7	1065	4962.5k

Utilizing runtime referencing L2 and single component indexing L1, the 14 LUBM requests are processed. Minimal join procedures were used in the queries, and RDF-3X and SHARD performance in both L1 and L2 were taken into consideration. Figures 10 and 11 display the response times for 14 queries that were performed to the LUBM 400 university dataset and ranged from Q1 to Q14. In the proposed approach, L2 responds more quickly than L1 does. When likened to RDF-3X and SHARD, the proposed technique performed worse on various queries, including Q1, Q3, Q4, and Q7. Implement robust access control mechanisms to ensure that only authorized users and applications can access sensitive RDF data. If dealing with sensitive business data, consider implementing data anonymization techniques to protect individual privacy while still providing valuable insights. Simple queries are executed slowly using the mechanism. This is because the RDF-3X runs the query in a matter of seconds. With this method, lookups and connections happen quite efficiently. While the proposed methodology and SHARD employ the decentralized approach, the RDF-3X adheres to the centralized system.

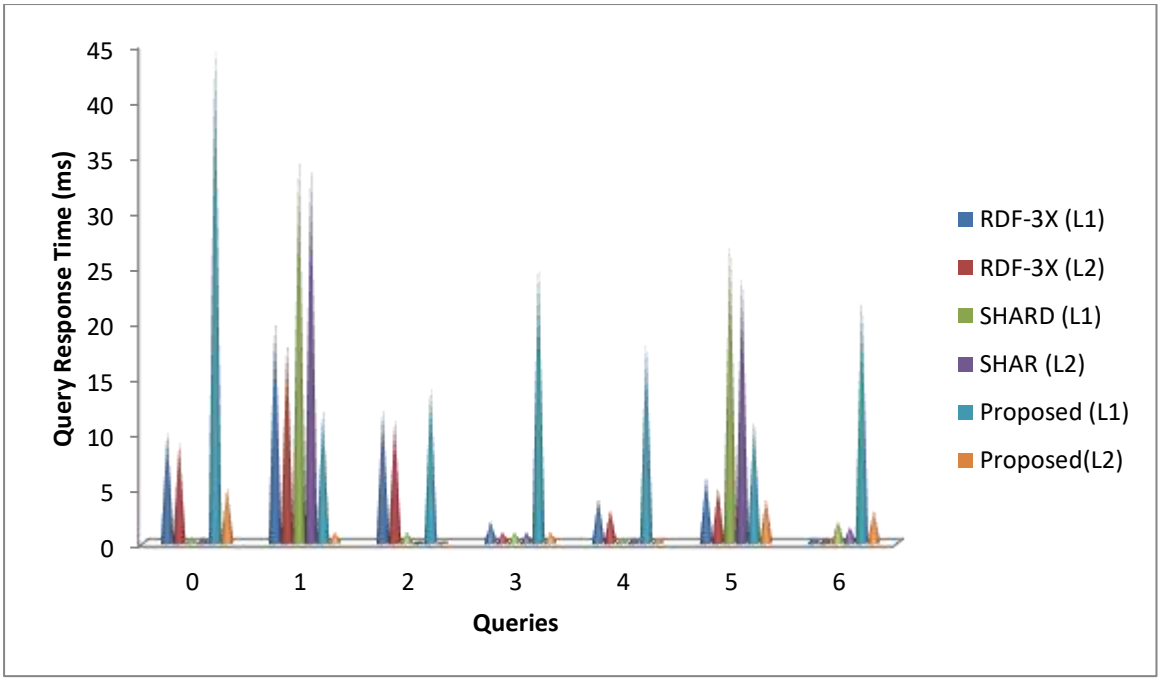


Figure 10: The response time from Q1 to Q7 in single level referencing L1 and L2 for 400 organizations

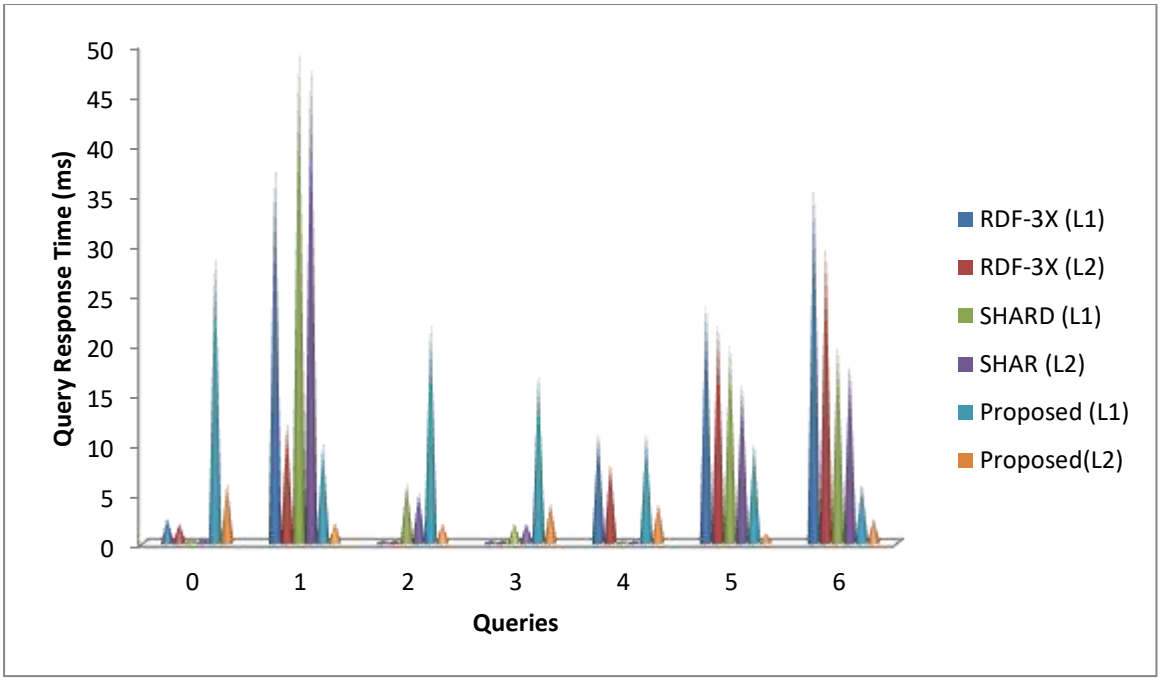


Figure 11: The response time from Q8 to Q14 in single level referencing L1 and L2 for 400 organizations

Figure 12 and Figure 13 display the reply time of the interrogations using the educational dataset LUBM 800. These data sets contain additional triples, which add additional overhead to the procedures. For Q2 and Q9, the proposed solution produced a slower response time. The proposed approach is made for sophisticated systems, but the parallel and distributed technique is not supported in different ways. Continuously monitor and analyze system performance to identify areas for improvement. Adjust caching strategies, indexing techniques, and query optimization rules as needed. Gather feedback from business users and application developers to refine the system's features and capabilities based on real-world usage scenarios.

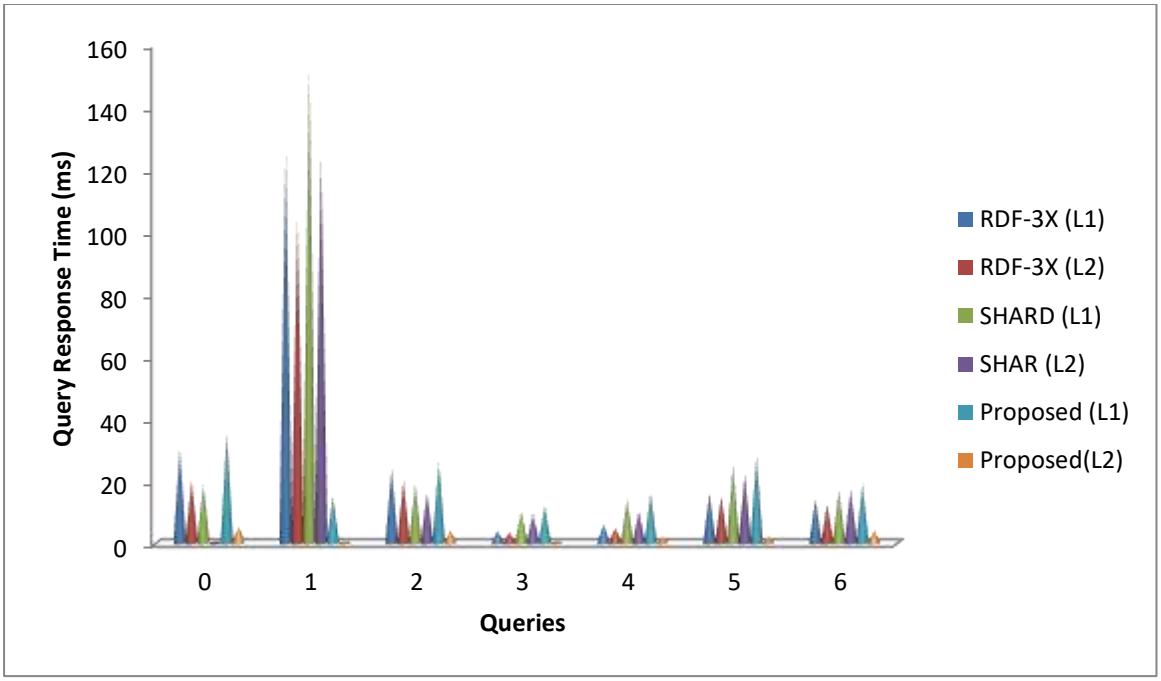


Figure 12: The response time from Q1 to Q7 in single level referencing L1 and L2 for 800 organizations

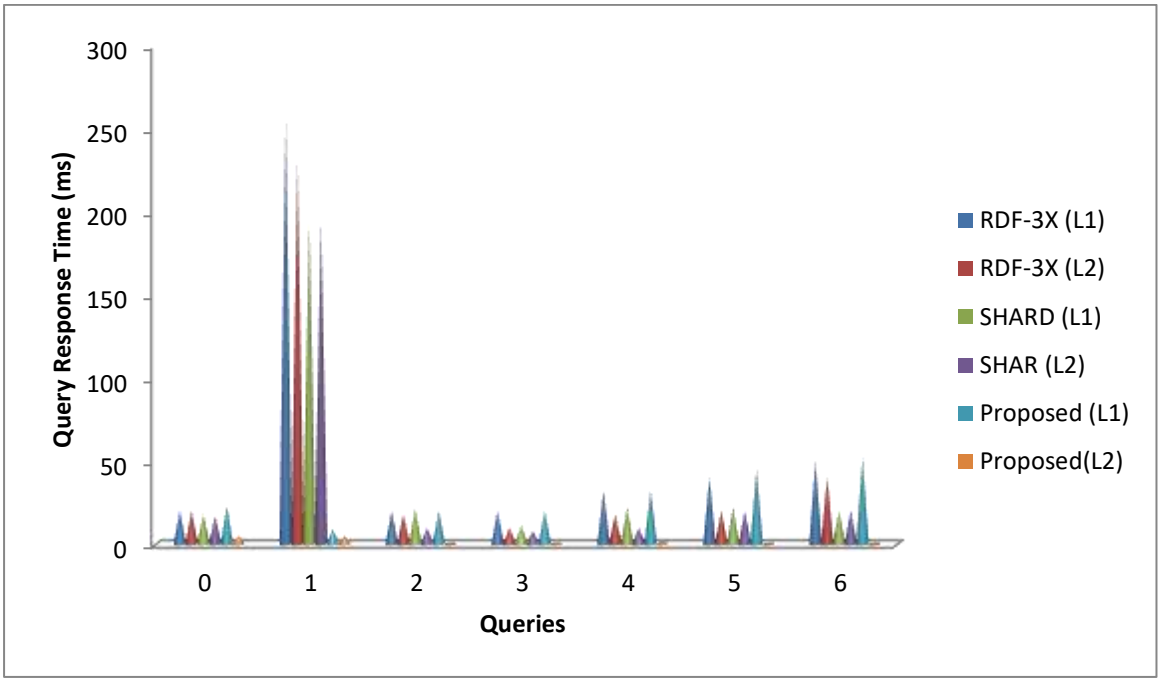


Figure 13: The response time from Q8 to Q14 in single level referencing L1 and L2 for 800 organizations

5. Conclusions

In this study, we created an effective ILS-RDF data processing mechanism for accelerating information extraction and query execution over massive amounts of information. To encrypt the information in the ILS-RDF, the proposed technique took advantage of the comparable dimension division methodology. To speed up data loading,

single-level indexing and runtime time indexing are utilized. The proposed model's effectiveness is evaluated using the LUBM benchmark. The proposed technique delivered acceptable results while analyzing queries and showed excellent efficiency when it came to extracting information. By developing an improved large-scale RDF system on big data requires careful consideration of data storage, query optimization, integration with business technology, and ongoing performance tuning. By combining distributed computing, efficient query processing, and seamless integration with business tools, creating a powerful platform that accelerates query processing and loading speed while providing actionable insights for an organization.

References

[1] Chandra, V. V., Sai, P. C., &Mandapati, S. (2022, May). An Efficient Framework for Load Balancing using MapReduce Algorithm for Bigdata. In 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC) (pp. 791-794).IEEE.

[2] Yadav, N. S., Krishna, M., Sapthami, I., Rao, C. M., &Parameswari, D. L. Sustainable Efficient Solutions for Smart Agriculture: Case study. In IoT and Big Data Analytics for Smart Cities (pp. 175-199). Chapman and Hall/CRC.

[3] Makpaisit, P., &Chantrapornchai, C. (2021). VEDAS: an efficient GPU alternative for store and query of large RDF data sets. Journal of Big Data, 8(1), 1-34.

[4] Garikapati, P., Balamurugan, K., Latchoumi, T. P., &Malkapuram, R. (2021). A Cluster-Profile Comparative Study on Machining AlSi 7/63% of SiC Hybrid Composite Using Agglomerative Hierarchical Clustering and K-Means. Silicon, 13, 961-972.

[5]Yao, Z., Chen, R., Zang, B., & Chen, H. (2021). Wukong+ G: Fast and Concurrent RDF Query Processing Using RDMA-Assisted GPU Graph Exploration. IEEE Transactions on Parallel and Distributed Systems, 33(7), 1619-1635.

[6] Cisneros-Cabrera, S., Michailidou, A. V., Sampaio, S., Sampaio, P., &Gounaris, A. (2021). Experimenting with big data computing for scaling data quality-aware query processing. Expert Systems with Applications, 178, 114858.

[7] Sun, Y., Zhao, T., Yoon, S., & Lee, Y. (2021). A Hybrid Approach Combining R*-Tree and k-d Trees to Improve Linked Open Data Query Performance. Applied Sciences, 11(5), 2405.

[8] Latchoumi, T.P., Ezhilarasi, T.P. &Balamurugan, K. Bio-inspired weighed quantum particle swarm optimization and smooth support vector machine ensembles for identification of abnormalities in medical data. SN Appl. Sci. 1, 1137 (2019).

https://doi.org/10.1007/s42452-019-1179-8
[9] Sun, Y., Chun, S. J., & Lee, Y. (2022). Learned semantic index structure using knowledge graph embedding and density-based spatial clustering techniques. Applied Sciences, 12(13), 6713.
[10] Wang, W., Zhang, Y., Ge, G., Jiang, Q., Wang, Y., & Hu, L. (2021). A Hybrid Spatial Indexing Structure of Massive Point Cloud Based on Octree and 3D R*-Tree. Applied Sciences, 11(20), 9581.
[11] Mountasser, I., Ouhbi, B., Hdioud, F., &Frikh, B. (2021). Semantic-based Big Data integration framework using scalable distributed ontology matching strategy. Distributed and Parallel Databases, 39(4), 891-937.
[12]M. Anand, N. Balaji, N. Bharathiraja, A. Antonidoss, A controlled framework for reliable multicast routing protocol in mobile ad hoc network, Materials Today: Proceedings, 2021, ISSN 2214-7853
[13] Bilidas, D., Ioannidis, T., Mamoulis, N., &Koubarakis, M. (2022). Strabo 2: Distributed Management of Massive Geospatial RDF Datasets. In International Semantic Web Conference (pp. 411-427).Springer, Cham.
[14] Benítez-Hidalgo, A., Barba-González, C., García-Nieto, J., Gutiérrez-Moncayo, P., Paneque, M., Nebro, A. J., ...&Navas-Delgado, I. (2021). TITAN: a knowledge-based platform for Big Data workflow management. Knowledge-Based Systems, 232, 107489.
[15] Sneha, P., & Balamurugan, K. (2023). Investigation on Wear Characteristics of a PLA-14% Bronze Composite Filament.In Recent Trends in Product Design and Intelligent Manufacturing Systems (pp. 453-461).Springer, Singapore.
[16] Yadav, N. S., Gogula, S., Sharma, G. K., Rao, C. M., &Parameswari, D. L. IoT and Big Data Analytics-Based Intelligent Decision-Making Systems. In IoT and Big Data Analytics for Smart Cities (pp. 101-119). Chapman and Hall/CRC.
[17] Parameswaran, T., Reddy, Y. P., Nagaveni, V., &Sathiyaraj, R. Era of Computational Big Data Analytics and IoT Techniques in Smart City Applications. IoT and Big Data Analytics for Smart Cities, 1-22.
[18] Hirakata, T., &Amagasa, T. (2021, November). A Dynamic Load-balancing Method for Distributed RDF Stream Processing Systems.In The 23rd International Conference on Information Integration and Web Intelligence (pp. 410-414).
[19] Devaki, K., &Leena Jenifer, L. (2022). A Study on Challenges in Data Security During Data Transformation. In Computer Networks, Big Data and IoT (pp. 49-66).Springer, Singapore.
[20] Arooj, A., Farooq, M. S., Akram, A., Iqbal, R., Sharma, A., &Dhiman, G. (2021). Big data processing and analysis in internet of vehicles: architecture, taxonomy, and open research challenges. Archives of Computational Methods in Engineering, 1-37.
[21] Latchoumi, T. P., Swathi, R., Vidyasri, P., & Balamurugan, K.

(2022, March). Develop New Algorithm To Improve Safety On WMSN In Health Disease Monitoring. In 2022 International Mobile and Embedded Technology Conference (MECON) (pp. 357-362). IEEE.

[22] Xia, Q., Zhou, L., Ren, W., & Wang, Y. (2022). Proactive and intelligent evaluation of big data queries in edge clouds with materialized views. Computer Networks, 203, 108664.

[23] Wei, R. (2022). Load Balancing Optimization of In-Memory Database for Massive Information Processing of Internet of Things (IoTs). Mathematical Problems in Engineering, 2022.

[24] Vankdothu, R., Hameed, M. A., Bhukya, R., &Garg, G. (2022). Entropy and sigmoid based K-means clustering and AGWO for effective big data handling. Multimedia Tools and Applications, 1-18.

[25] Dubey, A. K., Gupta, R., & Mishra, S. (2021, March). Data stream clustering for big data sets: A comparative analysis. In IOP Conference Series: Materials Science and Engineering (Vol. 1099, No. 1, p. 012030). IOP Publishing.

[26] Santipantakis, G. M., Kotis, K. I., Glenis, A., Vouros, G. A., Doulkeridis, C., &Vlachou, A. (2022). RDF-Gen: generating RDF triples from big data sources. Knowledge and Information Systems, 64(11), 2985-3015.

[27] Ren, Y., Huang, D., Wang, W., & Yu, X. (2023). BSMD: A blockchain-based secure storage mechanism for big spatio-temporal data. Future Generation Computer Systems, 138, 328-338.

[28]Karimi, Y., HaghiKashani, M., Akbari, M., &Mahdipour, E. (2021). Leveraging big data in smart cities: A systematic review. Concurrency and Computation: Practice and Experience, 33(21), e6379.

[29] Bilal, B. M., Ilham, C., &Azeddine, Z. (2021). An empirical study on the evaluation of the RDF storage systems. Journal of Big Data, 8(1).

[30]Umamageswari, N. Bharathiraja, D. Shiny Irene,A Novel Fuzzy C-Means based Chameleon Swarm Algorithm for Segmentation and Progressive Neural Architecture Search for Plant Disease Classification,ICT Express,2021,ISSN 2405-9595

[31]Amer, A. A., Abulwafa, S. S., & El-Hadi, M. M. (2021, March). A Proposed Framework for Building Semantic Search Engine with Map-Reduce.In International Conference on Advanced Machine Learning Technologies and Applications (pp. 469-477).Springer, Cham.

[32] Ali, W., Saleem, M., Yao, B., Hogan, A., &Ngomo, A. C. N. (2021). A survey of RDF stores & SPARQL engines for querying knowledge graphs. The VLDB Journal, 1-26.

[33]Yadav, S., Yeruva, S., Kumar, T. S., & Susan, T. (2021, October). The Improved Effectual Data Processing in Big Data Executing Map Reduce Frame Work. In 2021 IEEE Mysore Sub Section International Conference (MysuruCon) (pp. 587-595). IEEE.