Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 44 No. 7
July 2023

Map Reduce with Hybrid Optimization for Advanced Scalable Sparql Query Processing and Loading Speed in 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]Professor & HOD, Dept. of CSE, HKBK College of Engineering, ashokdbit2017@gmail.com.

Abstract

The expansion in web content had presented unique issues in the detailed term of adequately querying Resource Description Framework (RDF) graphs. Large amounts of information are kept on a massive range of connected servers that share storage space. Calculation methods were developed to directly conduct computation activities upon those machines, which were formerly utilized mostly for storage. In this research, we propose Smart Knowledge Graph (smart-KG) a technique that combines Transaction Processing Facility (TPF) with shipping reduced KG segments to divide the traffic load across the clients and lowering information transfers in volume to convert a specified Query language into a Hive application. We suggest a model-driven technique to accomplish this, develop a meta-model to specify a mapping between the Simple Protocol and RDF Query Language (SPARQL) metamodel and Big Data query languages. The Atlas Transformation Language (ATL) was used to carry out the transformation. According to our tests, smart-KG surpasses cutting-edge customer solutions and boosts server-side availability, enabling more affordable and equitable administration of public and decentralized KGs. To evaluate our methodology, we tried an experiment on three different datasets that each contained a sizable amount of dispersed RDF data on a potent server cluster.

Keywords: Big data; Knowledge Graphs; Partitioning; optimization; RDF; SPARQL

1. Introduction

For efficiently creating information systems, information systems engineering provides a wide range of product and process approaches and models. Business intelligence has been forced to find ways to control large and varied amounts of data since the introduction of big data. Within a short period, Model Driven Engineering (IDM) has gained attention among scientists related to computers in both the industrial and academic worlds. Therefore IDM has made various important advancements in the creation of complex systems and it is possible via enabling a focus on most abstract issues compared to traditional programming. A whole or partial program is created using models in this type of generative engineering [1].

In an attempt to offer sustainable learning models that describe information regarding entities and also relationships between certain entities, KGs possess been identified as a possible information management basis. Adopting the KG concept could lead to the development of cutting-edge goods and services with new commercial use from companies like Google, Microsoft, and Bloomberg [2]. Additionally, specialized projects and domains, including those in biomedicine, currently extensively employ KGs to integrate a variety of datasets in areas like cancer research, neuroscience, and drug discovery. Publicly available interconnected KGs that are released today that adhere to the relational data principles, use the quasi-RDF database structure and provide query access through

A group of computers known as a Hadoop cluster serve as Hadoop's foundation. Each device is referred to as a node. Large storage systems and powerful computation are made possible by the combination of its nodes' processing and storage capacities. The Hadoop Distributed File Structure (HDFS) is the name of the storage system. The MapReduce (M/R) parallel programming paradigm provides the foundation for computing power [4]. There are good reasons to think that Hadoop will

take over as the standard information processing tool in digital media and M/R would become the standard way to handle data.

The issue in the M/R language is considerably least or close to the machine. This makes it difficult for business users or new parallel processing developers to understand how else to communicate with the cluster. Giving users access to what is referred to as an "abstraction language" is one technique to make M/R development and Hadoop more straightforward. Therefore in an abstraction language was indeed syntax that was similar to language enabling the formulation of issues in the business as straightforward questions [5]. The user's request is converted below into an abstraction when he submits it, which is where the abstraction originates from. The farther in the device and the better it is for customers, the greater the degree of abstraction supplied by the language. Three abstraction languages are now available for M/R from the Apache Foundation: Hive. These 3 programming languages are intended for an audience who is a non-developer and, enable the expression of M/R tasks in a familiar to users, SQL-like programming language. The textual requests are then converted by these languages towards the M/R jobs, which are then sent to the groups for processing [6]. Generally, Hive supports the highest abstraction limits than Cascading and is an abstraction-level-heavy language. This article will only discuss

Due to the rapid growth of RDF data in recent years, it is frequently impractical to hold all RDF triples on a single node. This drives interest in SPARQL query processing in a distributed architecture, particularly inside this Hadoop platform [7]. These works achieve tremendous scalability for assessing SPARQL inquiries towards the trillions of RDF triplets by utilizing the M/R framework. However, due to cross-node connections were not allowed during the map phase of M/R, SPARQL queries frequently involve numerous connections, and all these connection procedures may be carried out by many worker nodes. Thus, SPARQL queries might require numerous jobs in M/R, which is highly costly given that every single job requires a significant amount of time to start up.

2 Related works

Data partitioning is a well-known distributed database approach for lowering cross-node communication. The fundamental concept is to place all potential join attribute values in a single worker node. To achieve this, we often need to examine prior query workloads and determine whose rows or records were likely to appear in identical queries [9]. By utilizing automatic data partitioning, this work seeks to facilitate the expandable and effective processing of SPARQL queries. Our research is grounded in the HadoopDB project, which suggests a layered framework through fusing databases with Map/Reduce [10]. The main concept was to run M/R tasks across a cluster in the database. That would allow the hybrid system to inherit the Map/Reduce framework's scalability and fault tolerance.

Our partition strategy was motivated by the discovery that many applications typically have some common query patterns. Querying patterns are unique SPARQL queries that essentially provide a code that may be used to assemble a collection of related SPARQL inquiries. When processing SPARQL searches that follow any query pattern, we can help them ensure that no cross-node joins are required [11]. It should be noted that the query optimization of our system will also outperform systems that appear to be based on Hadoop because of the tremendous potential of RDF-3X in executing multiple connections, including for searches that do not follow any defined patterns. Where the authors additionally suggest evaluating SPARQL queries on HadoopDB is the part of their work that most closely resembles ours. Depending on a graph partitioner called METIS, the entire RDF graph is split into numerous enormous sub-graphs [12]. Triples close to the division boundaries would be copied to numerous nodes, and so these subgraphs would've been kept at various worker nodes. The majority of queries could be resolved using the triples contained within a single node as a result of this segmentation. The study provided describes an inquiry mechanism for linked open that propagates the search for responses to a query by utilizing the similarity connections among assets of

various databases [13–14]. This method exploits the OWL same as connections of a known Linked Data database's assets to certain other datasets to find possible results after first querying it. The "Linked" feature of Linked Data is completely utilized by this system. Therefore the initial information data sets are adequately linked in Linked Data, this strategy, unlike our solution, does not necessitate possessing a catalog of Integrated Data databases for queries [15]. Although our solution is intended to reduce these latency periods, it still has issues with latency times when searching other databases. A specific endpoint can be specified for specific body sections of the replicated SPARQL request using the SERVICE keyword, which is defined in the SPARQL 1.1 specification [16]. Similar to earlier work, suggests an approach for streamlining the concurrent execution of queries to each database in federated SPARQL queries [17]. Using an algorithm based on the latency times of the various databases, this device separates a request into various sub-requests based on their selection.

The M/R programming concept which is a large amount of data is presented by the authors. In this architecture, the function of the map is to aggregate the information, whereas the map functions the analyses the information. M/R workloads are split into reduced tasks and map tasks, which are then spread over several servers [18]. That makes it possible for developers to use such a system without any prior knowledge of simultaneous or decentralized programming [19]. The research has focused on dealing with SPARQL query processing on Hadoop and Big Data. This analysis discusses and compares those devices by calculating the execution instances and also loading instances.

Resources in RDF databases are represented either by linkages they have towards other resources or by literal values. Therefore, these relationships hold essential meaning for the interpretation of RDF databases. Therefore, there is a discrepancy between the physical manifestation in an RDF database and the hierarchical version that the user perceives [20]. Through the interfaces of a Linked Data database, a transparent rewrite of numerous SPARQL queries is used to query a SPARQL query. Execution time is directly impacted by the various SPARQL searches that are required to identify the erroneous components of a consumer SPARQL query. With this approach, users can create systematic examinations inherently without needing to be familiar with the underlying vocabularies and the IRIs beforehand (Internationalized Resource Identifier). These recommendation systems are one of the regions that need strong techniques to handle heavy amounts of Semantic Web information. RecSPARQL is a system recommended that is entirely based on SPARQL. To support flexibility and general collaborative filtering and suggestions depending on the RDF graphs; the proposed tool enhances the semantics and syntax of SPARQL.

3. Proposed System

Hence we now go over our strategy, which consists of three key components: the source-SPARQL metamodel; the target - Hive metamodel; and ATL – communication between two metamodels. Figure 1 illustrates the various phases in our device's operation. Every SPARQL requestiation provided by a customer complies with our SPARQL metamodels, and so that request would undergo a transformation or mapping utilizing the ATL programming languages, which converts the SPARQL clauses towards Hive clauses.

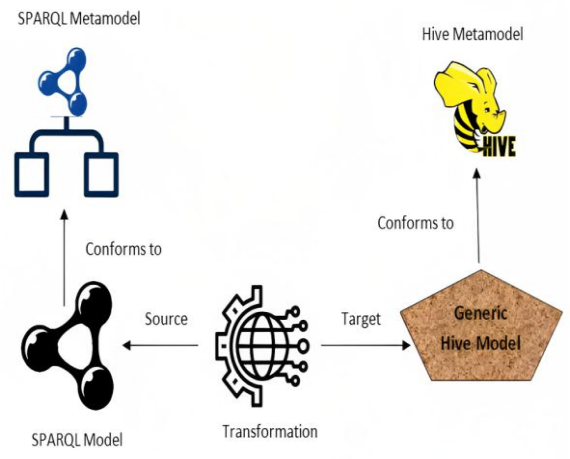


Figure 1: The proposed architecture

In reality, many RDF graphs have huge equivalent partitions for "dominant" families rather than numerous smaller, closely related families with significantly smaller sizes [21]. RDF is semi-

structured and attributes might differ between objects of the same type, this situation occurs. In addition to deploying predicate-restricted families, merging (or clustering) related families into a single partition has a significant positive impact on our partition shipping method. The idea underlying integrating these groups that cover identical criteria is that more frequently occurring query patterns may also use those overlapped predicate subsets as predicate families. Therefore, just the segment matching towards the least united groups must be transmitted as opposed to the totality of divisions responding to a query.

3.1 Algorithm 1: RDF in Big data

Step 1: Initialize an empty graph as Q';

Step 2: Choose an edge e randomly from Q into Q';

Step 3: update the e_0 into S

Step 4: Choose a partition randomly from S for every triple;

Step 5: Update the partition information;

Step 6: while E(Q') is less than E(Q)

{

Step 6.1: Choose an edge such that e ε $\frac{E(Q)}{E(Q')}$;

Step 6.2: insert e into Q';

Step 6.3: update the e into R

Step 6.4: for each triple t in S

{

Step 6.5: Choose the triples R'

}

}

Step 7: Keep t in each partition has at least one triple in R';

Step 8: Record the information

We employ an identical concept, but the main distinctions from the early studies are that we compute only those conditional subsets of a given generation that identify the merges, relating towards the non-empty set crossing points since some other family and maintain all of those interconnections based on Ñ cardinality. The divisions provided first by the smart-KG server are then calculated using Equation (1) wherein G' is substituted with such a collection of separations derived from the combined families:

$$\mathbf{G_{serv}} = \left\{\mathbf{G'_{\mu(f)}} | \mathbf{f \epsilon dom(\mu)}\right\} \textbf{ (1)}$$

3.2 Hive metamodel

Use Hive to create queries using the HiveQL language derived from SQL. M/R jobs are created from these searches shown in Figure 2. Simply specify a design that is connected to the data for it to function. The information is organized into tables that HiveQL can use, and this schema provides the identifiers or kinds of columns. Order by, Having, WHERE, and Group By are the clauses that make up a Hive SELECT query. Our Hive metamodel is shown in Figure 3.

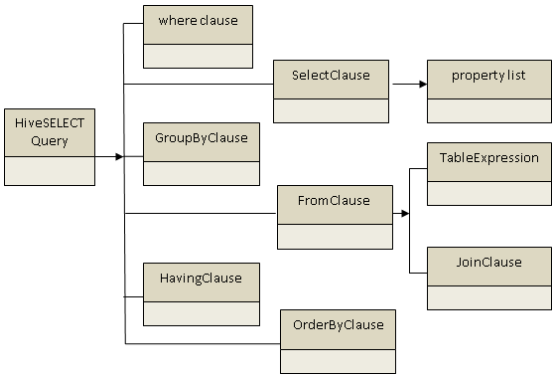


Figure 2. Proposed Hive metamodel

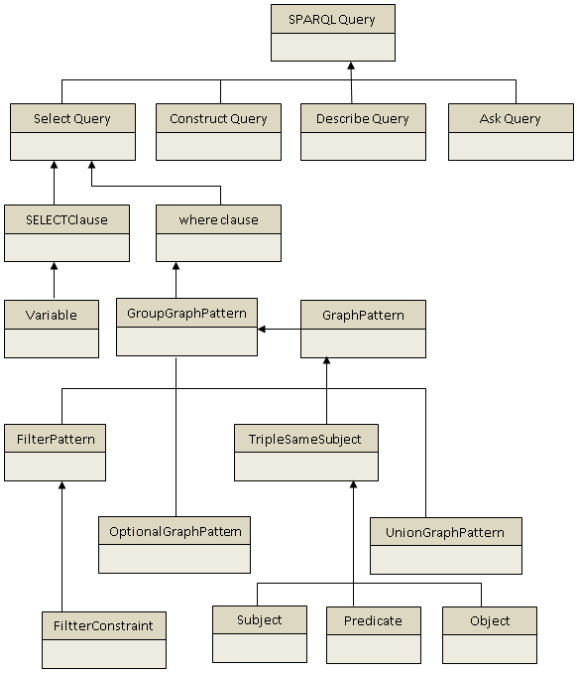


Figure 3. Proposed SPARQL metamodel

3.3 Transformation

ATL programs were made of rules which specify rectify the target model's elements are created by identifying and iterating over elements in the source model. ATL provides an extra model of request (transformations of model-based towards text [22]) in addition to the transformations of both classical models, allowing for the specification of inquiries on the models. The hybrid nature of such ATL language is one of its distinctive features.

An element of the origin metamodel of the conversion can be directly correlated including a component of the destination metamodel of the conversion thanks to the declarative part. Object Management Group (OMG) introduced Query View Transformation (QVT) as part of the Multi-Dimensional Array (MDA) strategy. A query is a request that uses a model input to choose particular model components shown in Figure 4. A model called View is derived from previous models. An input model is required for transformation to alter or construct another model.

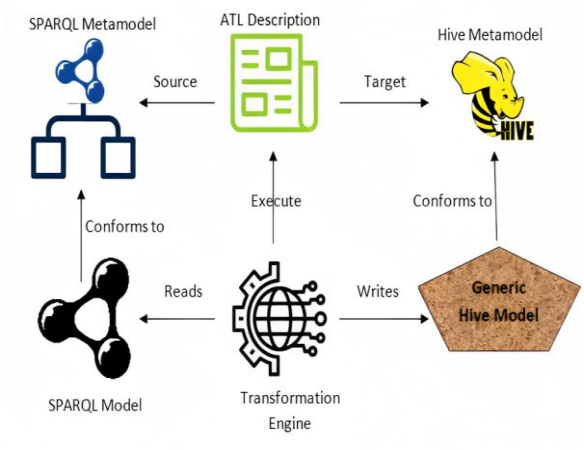


Figure 4: ATL engine

3.4 Optimization Heuristics

A SPARQL join query is made up of many expensive joins. Maximizing the number of merge connections inside the query plan is the primary objective. In this context, a merge join is frequently a sort- or merged, joined or any other connection that benefits from the presence of an index. Minimizing intermediate results to reduce the storage overhead during query execution is another equally crucial goal. This is accomplished by prioritizing the evaluation among the most desirable triple patterns. Statistics have traditionally been used to determine that triplicate patterns are more selective. A series of empirical focused on the syntactical structure of triplets designs have been assembled here to help numerous selective ones.

Based on the findings that there were relatively few if any, qualities that may meet the triple pattern provided a topic and an object, the abovementioned ordering was created. According to this, it is extremely uncommon for a topic and property combination to have much more than one instance value [23].

Algorithm 2: SPARQL in Big data

Input: Query Q

Output: M: TP → (V, P) // M-Mapping

Step 1: Initialize C = NULL // candidate variable; and T = Q;

Step 2: While (T ≠ NULL)

{

Step 3: Initialize I = NULL and S = NULL //I and S are set of variables;

Step 4: Let G(T) variable graph from T

Step 5: Compute MaxIndepentendSets // Maximum Independent Sets;

Step 6: set MaxIndepentendSets(G(T)) to I

Step 7: If |I| greater than 1

{

Step 8: Apply HEURISTIC 3,4 and 2 in I

}

Step 9: Apply RandomChooseOne(I) into S

Step 10: C ← C U S;

Step 11: Remove T from S;

Step 12: T = T\{tp|T, vars (tp)∩S≠ø};

}

Step 13: for every $(c \in C)$

{

Step 14: Pick variable c

Step 15: $T \leftarrow \{tp \in Q | c \in vars(tp), tp \in M.keys\}$;

Step 16: for each $(tp \in T)$

{

Step 17: Assign the Ordered Relation (M, tp, c);

}

}

The pseudo-code for the SPARQL method is displayed in Algorithm 2. A SPARQL join queries Q is accepted as input by HSP, which then produces a map (M) with every of Q's triplicate forms translated to a relation, and that variable would be used in a merging connection, or zero when there is no connector. To find every maximum user autonomous group of such variable graphs G(T) and store them in I, HSP first runs the function MaxIndepentendSets. All maximal autonomous sets returned by the function MaxIndepentendSets are collected in I. We must select one candidate from among all the alternatives. The HSP method then uses principles 5, 4, 3, and 2 to select the most selective set in I out of all the alternatives, i.e., the one which yields the fewest intermediate outcomes by the discussion in the preceding segments. After applying the heuristics, if there is still more than one set in I, one set is chosen at random. Then, until all triple designs in Q are covered, the three designs of Q that are used, that is, wrapped by variables in the selected independent sets, are deleted in Q.

4. Results and discussion

Table 1 depicts that |F ′ cor e |, |Gserv |, and the calculation time were sub-linearly increased towards the graph sizes.

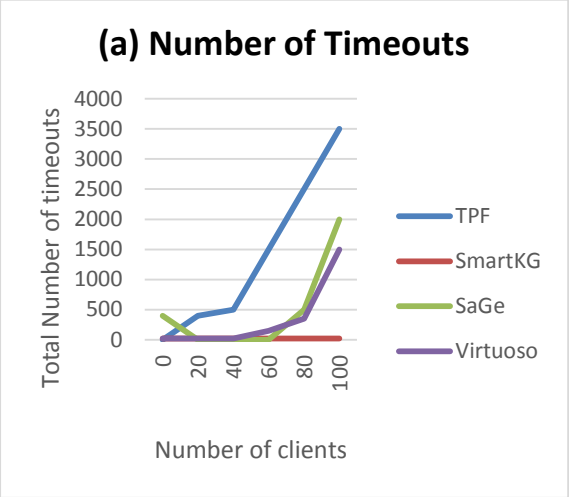
Table 1: Characteristics of KG

RDFGraphG	$\lvert P_G^F \rvert$	$\lvert P_{core}^F \rvert$	$\lvert F_{core}^F \rvert$	$\lvert G_{serv} \rvert$	C.Time(h)
WatDiv-10M	60	60	10.111	21.212	1
WatDiv-100M	60	60	22.857	37.394	8
WatDiv-1000M	60	60	39.048	52.886	13
DBpedia	219	85	35	29.977	24

Figure 5 (a) demonstrates that at such relatively low but cutting-edge graph sizes, smart-KG creates no timeouts. It that seems to be with 80 simultaneous clients, our method can effectively complete all of the workload's queries for all clients. In comparison, even with a single client configuration, TPF was unable to respond to all queries within a time out of 5 minutes. The proportion of timeouts rises with the number of clients, from 10% in a workload with just one client to an average of around 28% with 80 clients active

at once. These findings support the system's scalability constraints.

Smart-KG restricts servers load and joins are performed primarily, the mean workflow completion time per customer in Figure 5 (b) illustrates our approach's exceptional performance and scalability. Effectiveness as well stays unchanged irrespective of the number of customers. Clients received over-delivery of KG partitions. It illustrates whether smart-KG generates no time delay at such small but cutting-edge graph sizes. That seems to be correct, also with 80 concurrent users, this approach is capable of completing every one of the workload's requests including all customers. By contrast, though with a single-user setup, TPF was also unable to answer everyone's requests within such a latency of 5 minutes. The fraction of timeouts jumps with the number of customers, between 10% points in a task only with one customer to such an estimation of 28% points having 80 customers engaged at once. These results supported the program's sustainability limitations. SaGe breaks out less frequently than TPF on WatDiv-100M, however, the percentage of timeouts climbs considerably even as the number of customers grows, achieving a non-negligible 15% of the total of inquiries.



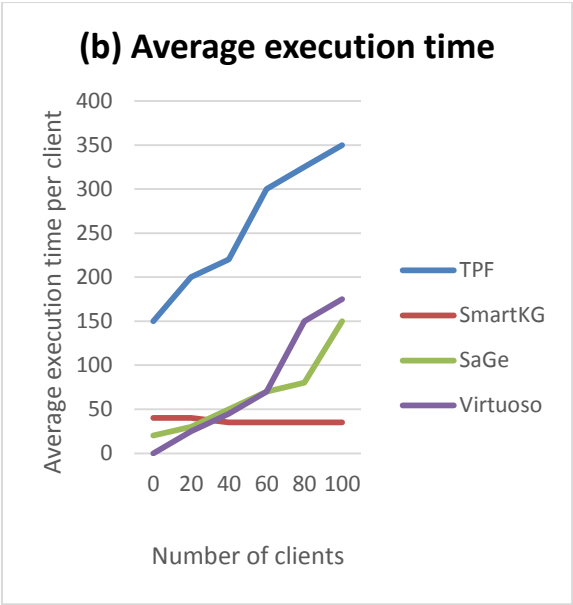
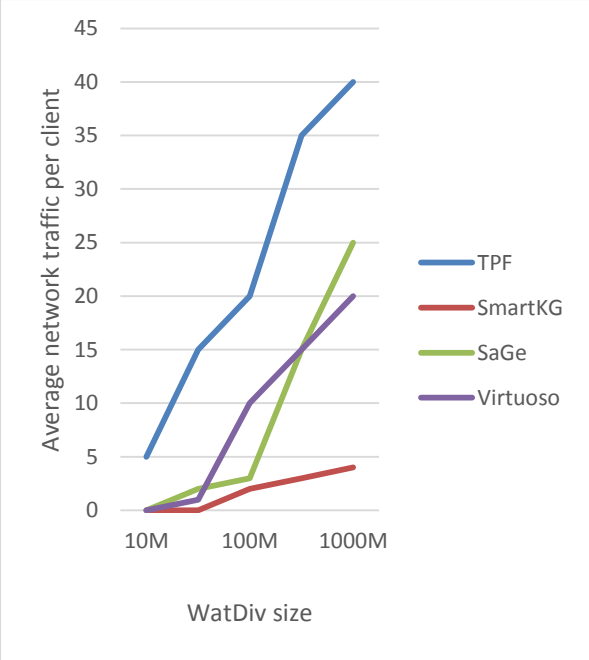


Figure 5: Performance on the WatDiv-100M workload

Figure 6 (a) demonstrates that smart-KG creates zero time delay at these small but cutting-edge graph sizes. That, also with 80 Eighty concurrent users, our technique can complete every inquiry in the task for any concurrent clients. TPF, on the other hand, has been unable to respond to all questions inside a 5-minute set-up. The proportion of queuing delays increases exponentially for clients, between 10% with a single client such a mean of 28% for 80 concurrent customers. Those findings support the system's scalability constraints. On WatDiv-100M, SaGe breaks out within fewer searches compared to TPF, however latencies rise considerably with both the number of customers, achieving a non-negligible 15 percent of requests with 80 simultaneous clients. Figure 6 (b) shows that TPF and SaGe extensively use the server CPU. Table 2 raw data sizes (in N-Triples) of the graphs and storage requirements for the evaluated systems [24].



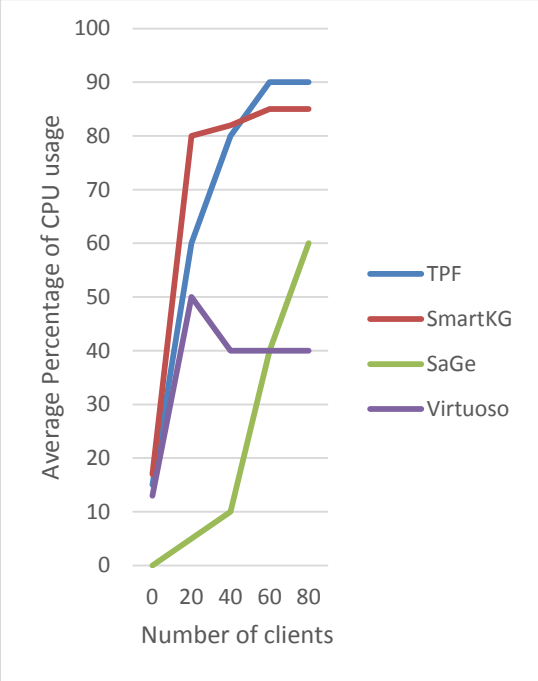


Figure 6: (a) A traffic per client (in GB) (b) Average Server CPU Usage (in %)

Table 2: Comparison of storage requirements (in MB) for systems with HDT backend vs original graph size

Dataset	Raw	SmartKG	TPF/SaGe
WatDiv-10M	1475	2784	114
WatDiv-100M	14875	29713	1188
WatDiv-1000M	151873	310575	12795
DBpedia	158204	122445	17911

4.1 SPARQL - Hive

SPARQL2Hive is built in Hadoop 3. x and the Hive 3.1.0 on a computer with a 2.3 GHz Intel Xeon microprocessor, which may store up to 4 TB of computer storage or 16 GB of RAM. Such three features, LUBM5, LUBM1, and LUBM2, have been used in this test. Those who get the mentioned triplet numbers: 138 million triples, 275M, and 689M, and their sizes are: 11.4 GB, 77, 22, GB, and 56, 8 GB. Table 3 displays the information acquired again for constraints among these 3 games. Table 4 illustrates the running time of 4 LUBM queries on instances.

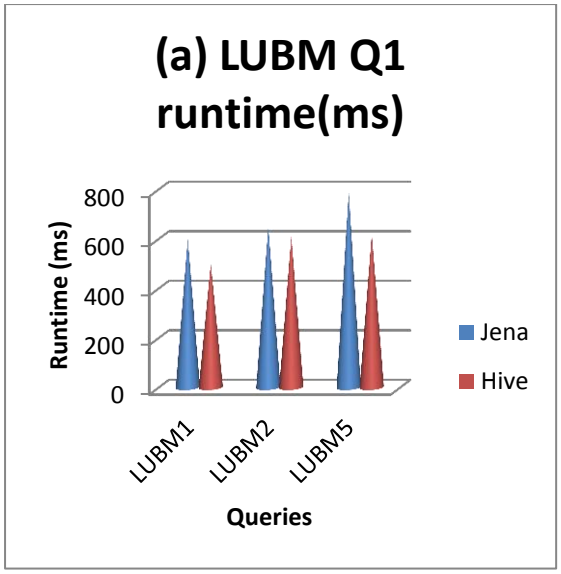
Table 3: LUBM datasets based on loading time

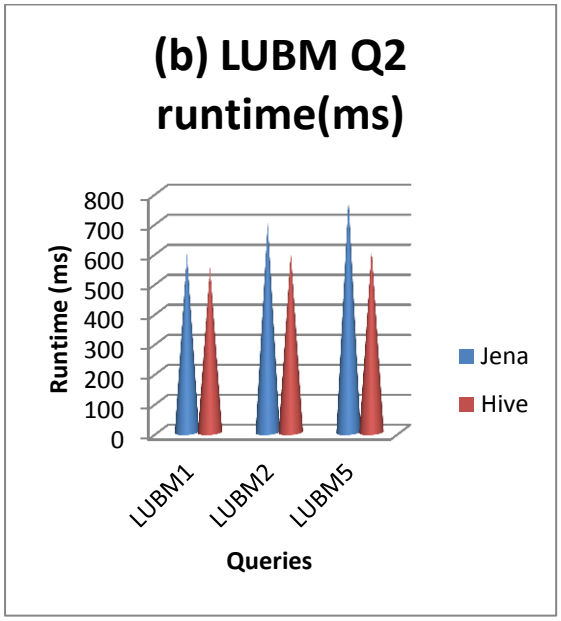
Metrics	**LUBM-1**	**LUBM-2**	**LUBM-5**
Time of loading in ms	1,27	3,07	8,1

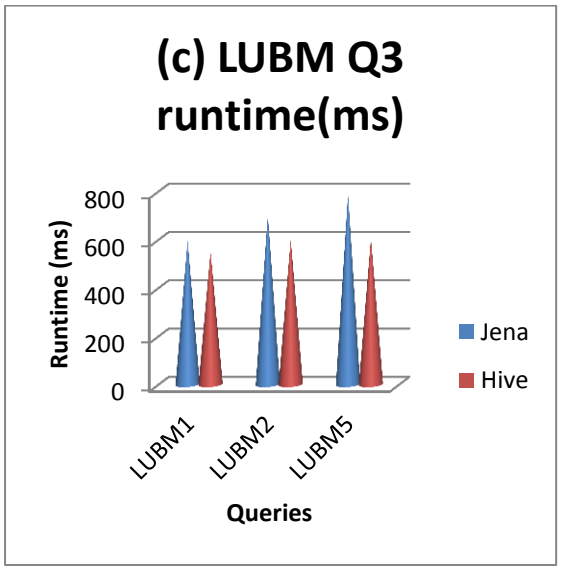
Table 4: LUBM queries running time

S.No.	**LUBM-1**	**LUBM-2**	**LUBM-5**
Query 1	483	538	754
Query 2	427	514	642
Query 3	534	585	635
Query 4	510	620	628

We make comparisons between our SPARQL to Hive scheme and Jena, besides even using the three parameters LUBM5, LUBM1, and LUBM2, in the vast majority of enquires; SPARQL2hive seems to be most powerful compared to Jena just in the executable level of LUBM Benchmark enquires. The outcomes of such an evaluation using LUBM inquiries can be seen in Figures 7 (a–d).







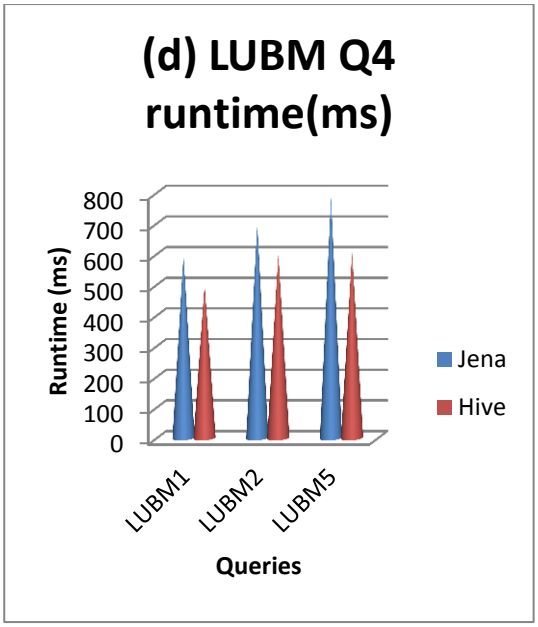


Figure 7: LUBM queries run time

4.2 Big Data query

Based on the above discussion, we may infer whether SPARQL2Hive is indeed a scalable, resilient, yet fault-tolerant system. These findings demonstrate SPARQL2Hive's performance whenever the Relational data space is large. It does not take much time for SPARQL2Hive to enter the data. It translates a specified Query language to a HiveQL application. In comparison to the Jena architecture, where function becomes a bit confusing since the query passes across a series of phases which take a very long time, notably for entering information shown in Table 5.

Table 5: Hive runtime based on LUBM queries

S.No.	Hive		
	LUBM-1	LUBM-2	LUBM-5
Query 1	482	539	753
Query 2	431	515	642
Query 3	537	584	634
Query 4	508	622	628

Furthermore, systems that provide better options for Hadoop have lately surfaced. The real objective is to give support for iterative methods. This is true for a wide number of information mining approaches that gradually improve results till an

ideal answer was obtained. M/R is (was) unsuitable for such a form of execution. Throughout this regard, technologies such as Spark and Flink represent significant breakthroughs. Spark includes writing sophisticated treatments that comprise various Map-Reduce phases.

This is also possible using YARN, however, the information from every step should be stored in HDFS until being utilized within the first phase. This requires a considerable amount of time or space. YARN processes are time-consuming both start and run. There are significant delays. On the alternative, Spark takes significantly greater use of such a cluster's memory system and also handles the workpiece directly. The therapies could be made up of a variety of languages, including Python, Java, and Scala

4.3 **Query Planner and Optimization**

The optimization strategies we developed were validated by our research using databases. Concerning Heuristic 1, we found that, given a predetermined range for just objectivism and subjectivism, there are only fewer qualities that meet the particular threefold structure. We also found a limited number of variables that matched a threefold pattern for the measured value of the attribute or objects. In a similar vein, we discovered that it is extremely uncommon for a topic and attribute pair to get more than one object result. Whenever the variable seems to have the result rdf: type, there is an exception to the rule because this triple should not be regarded as selective since it is a fairly common property.

Hence for the queries whose triplets patterns show syntactical dissimilarities—(that is, they include various numbers of variables (or/and shared variables) that are based in various positions—our algorithms generally show to be extremely effective. The use of HEURISTICS 1–4 with each question in our workload is described in the sections that follow. A significant proportion of the searches in both databases considered s-s joins, which suggests star-shaped connections upon that triple pattern's subject component, then s-o joins. The heavier the star-shaped connections established at the corresponding location of such triple patterns, the lower the ratio of independent variables across triplicate patterns.

This would be the scenario if questions SP2b or SP2a succeeded after query Y1. In addition to connection methods and factors about how merging connections are executed (variables that are sorted), we additionally calculated associated costs by using the RDF-3X price structure to make comparisons of a prime source using HSP and CDP.

Table 6 displays the implementation timings of our research for the YAGO and SP2Bench datasets, respectively. The estimated implementation timings need not account for completing the project on time (less than 4% of total execution time), the effort needed to convert the variables in each triple pattern into unique identifiers, or just the duration needed to convert these identifiers again to characters in the eventual query result. RDF-3X sorts or arranges the search queries to just decompress one component per grouping of duplication, which speeds up the quality based on the authentication to URIs and literals and also the ensuing decompression. Additionally, researchers do not include this time in their calculations.

Table 6: Query Execution Time (in ms) for YAGO queries (Warm Runs)

	Y1	**Y2**	**Y3**	**Y4**
MonetDB/HSP	6.05	8.67	25.70	2.33
RDF-3X/CDP	15.80	9.94	81.21	90.46
MonetDB/SQL	7.68	9.08	538.66	1.115

5. Conclusion

Recently, big data technologies, including Hadoop or NoSQL systems, have been used to store vast quantities of RDF graphs. We need to use SPARQL rather than Hadoop technologies to manage this information. From this study, we introduce a prototype design methodology for converting SPARQL queries into such an Apache HiveQL program only with the smart-KG client. This approach implements a query decomposer, planner, and executor specifically intended to manage TPF and partition shipping. The analysis

demonstrates how smart-KG greatly exceeds the condition, particularly with growing client concurrency and difficult BGP requests. Additionally, we demonstrate that smart-KG increases servers' availability while using substantially less CPU and RAM than the majority of the assessed solutions, mostly at the expense of adequate user capabilities and lessening the network traffic on TPF. Additionally, as we've seen in this research, there are currently a lot of performance concerns with the RDF language, particularly when thinking over huge volumes of information. There are numerous ways to approach them, most of which are currently under development. Another investigation involves the usage of such RDF overviews. Throughout taking RDF graphs into account, it was feasible to minimize the number of information sets 6it provides and maintain maximum accuracy, allowing for more efficient manipulation.

References

[1] Ravikumar, S., &Kavitha, D. (2021). A new adaptive hybrid mutation black widow clustering based data partitioning for big data analysis. Wireless Personal Communications, 120(2), 1313-1339.

[2] Shetty, S., Rao, B. D., &Prabhu, S. (2021). Growth of relational model: Interdependence and complementary to big data. International Journal of Electrical and Computer Engineering, 11(2), 1780.

[3] Lim, J., Kim, B., Lee, H., Choi, D., Bok, K., &Yoo, J. (2021). An Efficient Distributed SPARQL Query Processing Scheme Considering Communication Costs in Spark Environments. Applied Sciences, 12(1), 122.

[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] Beheshti, A., Benatallah, B., Motahari-Nezhad, H. R., Ghodratnama, S., &Amouzgar, F. (2022). BP-SPARQL: A Query Language for Summarizing and Analyzing Big Process Data. In Process Querying Methods (pp. 21-48).Springer, Cham.

[6] Li, M., Peng, P., Tian, Z., Qin, Z., Huang, Z., & Liu, Y. (2022). Optimizing Keyword Search Over Federated RDF Systems. IEEE Transactions on Big Data, (01), 1-18.

[7] 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.

[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] Patel, J., Patel, R., Shah, S., & Patel, J. A. (2021). Big Data Analytics for Advanced Viticulture. Scalable Computing: Practice and Experience, 22(3), 302-312.

[10] Groppe, S. (2021). Semantic Hybrid Multi-Model Multi-Platform (SHM3P) Databases.In ISIC (pp. 16-26).

[11] Draschner, C. F., Stadler, C., BakhshandeganMoghaddam, F., Lehmann, J., &Jabeen, H. (2021, October). DistRDF2ML-Scalable distributed in-memory machine learning pipelines for rdf knowledge graphs. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management (pp. 4465-4474).

[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] Naghib, A., JafariNavimipour, N., Hosseinzadeh, M., &Sharifi, A. (2022). A comprehensive and systematic literature review on the big data management techniques in the internet of things. Wireless Networks, 1-60.

[14] 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.

[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] Janev, V., Vidal, M. E., Pujić, D., Popadić, D., Iglesias, E., Sakor, A., &Čampa, A. (2022). Responsible Knowledge Management in Energy Data Ecosystems. Energies, 15(11), 3973.

[17] Pavel, A., Saarimäki, L. A., Möbus, L., Federico, A., Serra, A., & Greco, D. (2022). The potential of a data-centred approach & knowledge graph data representation in chemical safety and drug design. Computational and Structural Biotechnology Journal.

[18] 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.

[19] Jiang, W., Yan, L., Tu, Y., Zhou, X., & Ma, Z. (2022). PG-explorer: Resource Description Framework data exploration with property graphs. Expert Systems with Applications, 198, 116789.

[20] Rodriguez-Garcia, M., Balderas, A., &Dodero, J. M. (2021). Privacy Preservation and Analytical Utility of E-Learning Data Mashups in the Web of Data. Applied Sciences, 11(18), 8506.

[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] Ceballos, O., RamírezRestrepo, C. A., Pabón, M. C., Castillo, A. M., &Corcho, O. (2021). SPARQL2Flink: Evaluation of SPARQL Queries on Apache Flink. Applied Sciences, 11(15), 7033.

[23] Chawla, T., Singh, G., & Pilli, E. S. (2021). MuSe: a multi-level storage scheme for big RDF data using M/R. Journal of Big Data, 8(1), 1-26.

[24] Kumar, D., &Jha, V. K. (2022). A review on recent trends in query processing and optimization in big data. Wireless Personal Communications, 124(1), 633-654.