Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

An Enhanced Approach Of RDF Graph Data In-Memory Processing For Social Networks With Performance Analysis

M. Baby Nirmala[1] , J. G. R. Sathiaseelan[2]

Research Scholar, Department of Computer Science, Bishop Heber College(Affiliated to Bharathidasan University)Trichy, Tamilnadu, India

Associate Professor & Head, Department of Computer Science, Bishop Heber College (Affiliated to Bharathidasan University)Trichy, Tamilnadu -620017, India

---

Abstract—

Ever-growing linked open data proliferate in size and number giving rise to "Big data problems". Consequently, there is a rapid increase inf semantic data and services. It is significant to handle and provide more solutions to very large semantic data graphs are very significant. As the MapReduce algorithms are iterative and less effective, there is a need to go for in-memory processing. Ain of this paper is providing an enhanced RDF graph data processing approach with improved performance includes representation of RDF as a Directed labeled Multigraph, convert that RDF Graph as property Graph to have modeling workarounds (intermediate nodes), Reification and Singleton Property. Build this initial graph in high- performance analytic system SPARK with Graph X library using SCALA language to provide higher-order functions. So that functions and data structures can be stored in distributed memory and use SPARQL for query processing. The objective is of this paper is to propose an enhanced RDF Graph in-memory processing method that quickens the accessing of RDF Graph data, improves response times, and reduces the execution of time of RDF Graph data processing of this Social Network data.

Keywords— Large Linked Open Data; In-memory processing; RDF; MapReduce; Spark; Graph X; Scala.

I. INTRODUCTION

The Web 3.0 otherwise called the Semantic Web is introduced by Tim Berners-Lee, is the extension of the World Wide Web through standards established by the World Wide Web Consortium. This World Wide Web has progressed to support data sharing and reuse "across applications, initiatives, and community boundaries. Some of the basic building blocks of the Semantic web include Resource Description Framework (RDF), RDFS, the Web Ontology Language (OWL), and SPARQL. It has many challenges such as massiveness, indistinctness, ambiguity, inconsistency, and deceit. Automated reasoning systems should handle all these problems [1]. Query processing should be addressed proficiently with highly accessible, scalable, and fault-

tolerant frameworks. Usually, the data management systems requiring these properties are designed not from scratch but built on top of the prevailing cluster computing engine.

II. BACKGROUND

Resource Description Framework (RDF) is the key-tech for developing semantic web. In Graph database modes data is represented as directed labeled multi-graph. Theoretically, in the same way, the data encoded using RDF also can be represented. In RDF Graph, resources (IRIs), blank nodes, and literals can be represented as Vertices and edges from RDF triples. In other words, Subjects and Objects can be represented as nodes and Predicates as Edges. The edge's label is an IRI which is the predicate of the triple. RDF graph differs slightly in the case of the blank node as they have no identifiers. These RDF triples produced are used as Triple stores which are organized as graphs [1] [21].

A. SPARQL, as the Query Language

A Highly recommended query language for RDF datasets by World Wide Web Consortium is SPARQL. The following is an example of SPARQL query language:

```
SELECT? x,? y,? z
WHERE {x a Student. y a Representative. z a Scholar.
x friend y.
x competes with z.
y friend z
```

Extensive research has been conducted to optimize query engines. Sophisticated indexing approaches and graph-based storage models are developed due to the unsettled research in this area [2]. Linked Data is structured and interlinked with other data, through semantic queries they become more beneficial. It is built upon the technologies such as HTTP, RDF which serves web pages for human readers. LOD extends them to share the information which can be read by computers automatically. One of the visions of linked data is to make the Internet to be a global database [3]. Linking Open Data Heath, T et al, (2013) say "Linked Open Data (LOD) project has been initiated to provide a method of publishing a variety of structured data sets as interlinked RDF data sets". Linked open data are data that are linked and open. Tim Berners-Lee gives the perfect definition of linked open data in differentiation with linked data, "Linked Open Data (LOD) is Linked Data which is released under an open license, which does not impede its reuse for free". Large linked open data sets include DB pedia and Wikidata. LOD project incorporated 1014 interlinked RDF data sets in 2014. It spans a multitude of knowledge areas, such as geographic, government, life sciences, linguistics, media, publications, and social networking. DB pedia is one among these linked open data which is interlinked with the high number of other data sets. What was Interesting about this DB pedia is, it is an RDF representation of Wikipedia. The size of this RDF Graph in the LOD cloud is measured in tens of billions of RDF triples with lots of nodes and edges. Optimization of processing of SPARQL queries becomes indispensable as the sizer of these RDF data graphs grows significantly. There is a need for complex, hypothesis-driven [4] and

analytics-related queries, this becomes more essential. Henceforth more effort must be devoted to distributed processing of SPARQL queries [5, 6]. Moreover, processing federated SPARQL queries on the LOD graph [6] is challenging and requires vigorous research.

B. Graph Processing Systems

Graph processing systems like Map Reduce, Dyrad, and Spark compose data-parallel operators to transform collections. They are capable of expressing a wide range of computation and applying vertex-centric logic to transform data on a graph. They achieve more efficient distributed execution by exploiting the Graph Structure. Nowadays, it is obvious that social networks, web graphs, and financial transaction networks can be expressed as a graph. However, in the case of collaborative filtering, language modeling, deep learning, and computer vision, the graph model can be enforced through modeling assumptions. The structure of a graph can be denoted as G = (V; E) which has a set of vertices (v1, v2, v3… vn) and a set of m directed edges E (e1,e2, …en. The resultant of graphs can have billions of vertices and edges. They are often, highly sparse with complex, irregular, and often power-law structures [22]. Michele Knight (2021) states, "The property graph is a type of graph model where relationships not only are connections but also carry a name (type) and some properties" [23]. Alternatives in RDF are "Modeling workarounds (intermediate nodes), Reification [24] and Singleton Property" [25] can be well represented through Property Graphs.

C. MapReduce, as a distributed dataflow framework

Distributed data flow framework like MapReduce is one of the Cluster compute frameworks but not suited for graph analytics as many of their tasks are iterative [13]. Usually, a data model consists of typed collections, a coarse-grained data-parallel programming model composed of deterministic operators. They transform collections [12]. They also have a scheduler that breaks each job into a directed acyclic graph (DAG) of tasks and a runtime that can endure stragglers and partial cluster failures without restarting. MapReduce data model satisfies all the above characteristics [13]. This programming model depicts only two distributed dataflow operators such as Map and Reduce. Every job can have the utmost two layers in its DAG of tasks [12]. Dyrad, LINQ, Pig, and Spark are a few modern frameworks that uncover extra dataflow operators like fold and join and can execute tasks with multiple layers of dependencies [13]. They have extensive acceptance for a wide variety of data processing tasks, including ETL, SQL query processing, and iterative machine learning [12]. They are capable to scale for thousands of nodes operating on petabytes, exabytes, and zettabytes of data. Nevertheless, MapReduce (MR) is a prevalent cluster computing paradigm, it cannot be used for graph analytics because many graph analytics tasks are iterative [13].

D. Features of Apache Spark and Graph X

The remarkable features of Apache Spark and Graph x library are listed below

- Spark
  Spark has several features that are particularly attractive for Graph X. The Spark storage abstraction called Resilient Distributed Datasets (RDDs) is a fault-tolerant for In-Memory Cluster Computing [10]. It enables applications to keep data in memory, which is essential for iterative

graph algorithms. RDDs allow user-defined data partitioning [10]. Spark logs the lineage of operations and offers a high-level API [8].

- Graph X
  On top of Spark, Graph X is implemented [9]. It is a widely used data-parallel engine. A Spark cluster consists of a single driver node and multiple worker nodes like Hadoop MapReduce where the driver node is liable for task scheduling and dispatching and the worker nodes handle actual computation and physical data storage [11]. Joseph E. Gonzalez et. al, (2014) states Graph X accomplishes an order of magnitude performance gain over the base dataflow framework and matches the performance of specialized graph processing systems while enabling a wider range of computation[14]

E. Spark vs. MapReduce

Nevertheless, Spark also has numerous features that differentiate it from MapReduce engines which are significant to the design of Graph X [8, 10]. It has many features such as In-Memory Caching, Computation DAGs, Lineage-Based Fault Tolerance, Interactive Shell, and significance of key library Graph X in Spark [8, 14]. There is Spark's API, for working with graphs of nodes and arcs. Graph X's inventors have a whole section on RDF-related work stating "we adopt some of the core ideas from the RDF work including the triples view of graphs" [7, 11].

III.RELATED WORK

Alfredo et al., (2016) proposed a framework for BFS-Based Implementation of RDF Graph Traversals over MapReduce. However, this framework could be implemented on top of the Apache Spark analytical engine since Spark exposes an optimized in-memory computation scheme for large-scale data processing [15].

Michael et al., (2017) achieved better query times with Mantona's graph-cache retrieval as compared to retrieving queries through path traversals within a cached memory RDF-graph. Still, more experiments are needed on a large scale triple-level to achieve improved retrieval results and can be expanded to include query planning algorithms, based on dynamic programming [16].

Hubert et al., (2017) made a comprehensive study comparing four SPARQL query processing strategies for Apache Spark RDF data stores and devised a SPARQL Hybrid strategy that allows for combining P Join and Br Join. Nevertheless, the interaction between data partitioning schemes and distributed join algorithms as part of a general distributed join optimization framework can be explored deeply [17].

Ahmed et al., (2020) performed a comprehensive performance analysis of Apache Hadoop and Apache Spark for Large Scale Data Sets Using Hi Bench. Yet more parameters can be added in the workloads under resource utilization and parallelization, to analyze the job performance of MapReduce and Spark when several parameter configurations replace the default values [18].

Naacke et al., (2016) suggested techniques for processing SPARQL queries over a large RDF graph in a distributed environment. However, this work can be extended by handling SPARQL queries over linked open data (LOD). Moreover, this work motivates to revisit the classical problem of multi-query optimization for distributed RDF graphs [19, 20].

Peng Peng et al., (2016) proposed techniques for processing SPARQL queries over a large RDF graph in a distributed environment. The current work can be extended by handling SPARQL queries over linked open data where interconnected RDF repositories can be treated as a virtually integrated distributed database. Furthermore observation stimulates us to go back to the classical problem of multi-query optimization in the perspective of distributed RDF graphs [6].

Georgios et al., (2016) presented a framework for retrieving, analyzing, and storing medical information as a multilayer graph, an intellectual format suitable for data fusion a. This can be enhanced by comprising medical multilingual information retrieval and advanced text semantic analysis which can locate specific points of interest or data for such a user and even selectively translate part of the retrieved documents [26].

Dong Wang et al., (2017) developed a set of graph-based RDF data management systems which follow the approach of g Store, gStore-D and g Answer which are designed to support efficient SPARQL query evaluation in a centralized and distributed/parallel environments Instead of join processing, graph exploration can be used to boost the system's performance [27].

Ahmed et al, (2020) present the empirical performance analysis between Hadoop and Spark based on a large scale dataset with fewer workloads. Nevertheless, they did not investigate with remaining workloads with more parameters, as well they did not work on semantic data [28].

IV. PROPOSED WORK

The previous work of representing RDF as the graph for University data from DB pedia has been explained theoretically without experimental proof by M. Baby Nirmala, et. al, (2016) is as follows to have an introductory knowledge about the proposed work [22]. Big data technology usage for RDF data processing is fascinating as it would be motivating to output a typical Graph X graph as RDF so that SPARQL queries can be performed on it. There is no typical way of expressing RDF graph data processing but read a good-sized RDF dataset into Graph X and query through SPARQL. Spark with Scala provides a high number of higher-order functions [16] so that the functions and data to structures can be stored in distributed memory [17, 18]. In the batch-oriented MapReduce model, this doesn't work as this permits interactive and iterative tasks like machine learning tasks. Though these tasks work well. In the Spark version, they run 10X times faster because of in-memory processing, thus reducing disk I/O that is typical of MapReduce jobs. Spark provides a data structure called a Resilient Distributed Dataset, or RDD. Spark can watch over their distribution across computing clusters by storing data in RDDs. "Graph X helps store a set of nodes, arcs, and crucially for RDF types—extra information about each in RDDs" [18].

B. Property Graph for LinkedIn datasets

To construct a property graph consisting of the various details on the Graph X project, the vertex property contains the Person, Skill, Position, Organization, City, Education, Degree, College, and Course.. The Property Graph drawn for the LinkedIn dataset is given below in Fig. 1 and the converted Vertices and Edge tables are given in Table 1 and 2. From the Vertex table and Edge table, the initial graph will be built for further RDF Graph data processing, thus giving improved performance through SPARQL querie

TABLE I. VERTEX TABLE

Id	Property (V Vertice), Value
1	Person, Evan Smith
2	Skill, Programming
3	Position, Software Engineer
4	Organization, Infosys
5	Education, Post-Graduation
6	College, Stanford
7	Degree, MTech
8	Course, Computer Science and Engineering
9	City

TABLE II. EDGE TABLE

SourceId	Destination Id	Property (E -Edge)
1	2	Has Skill
1	3	Works As
4	3	Has Position
4	9	lives
1	9	location
1	5	Has Education
5	6	Has Studied
5	7	Has Degree
5	8	major

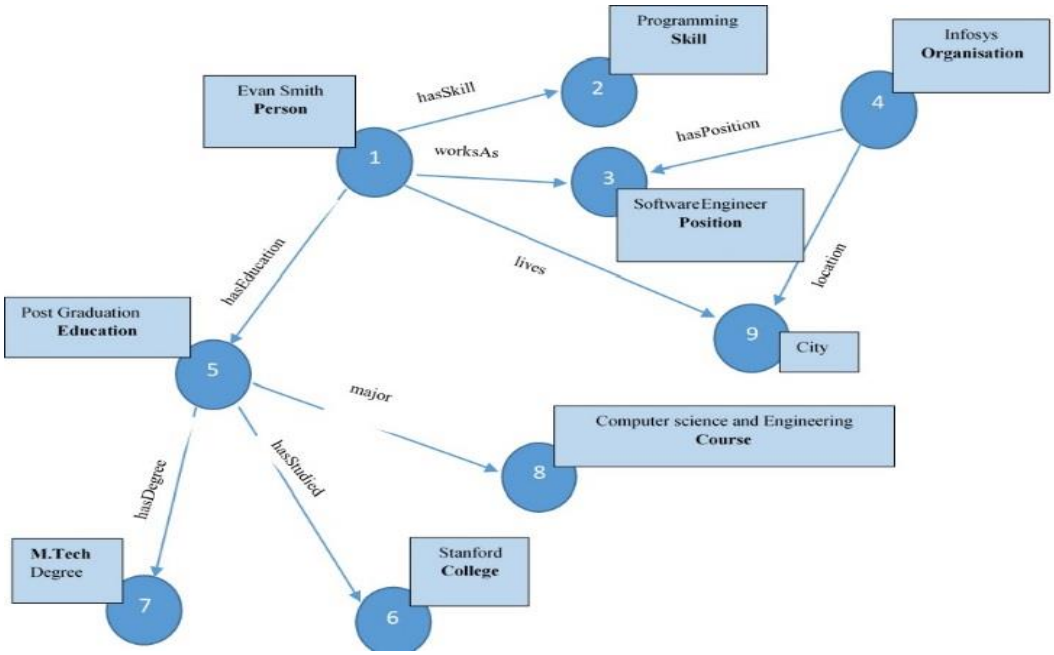

FIG 1, PROPERTY GRAPH OF THE LINKEDIN DATASET

C. Methodology

The process flow of the proposed methodology of in-memory processing of RDF graph data is given below in Fig. 2.

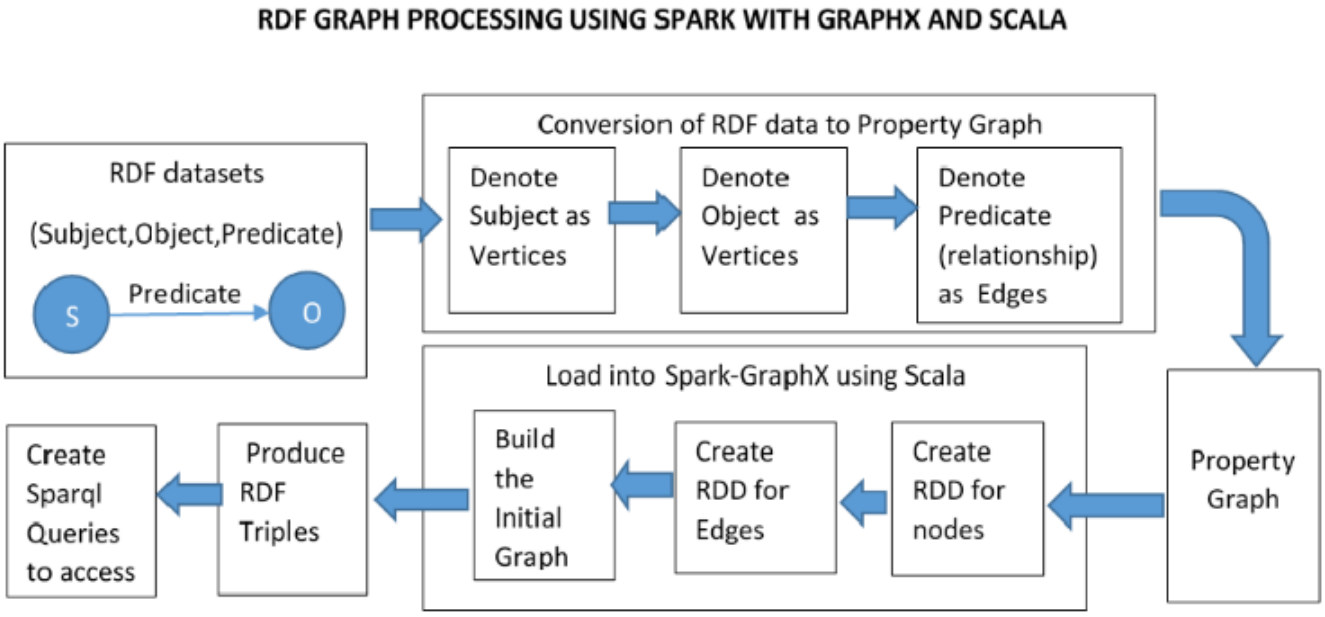

FIG 2. RDF GRAPH PROCESSING USING SPARK WITH GRAPHX AND SCALA

ALGORITHMS TO PRODUCE TRIPLES AND SEARCH VALUES

Algorithm to produce Triples
Input : RDF data sets
Output : Triples
1. Representing Subject, Object as Vertice
2. Representing relationships as edges
3. Create an RDD for the vertices
4. Create an RDD for edges
5. Build the initial property Graph
6. Producing RDF Triples

Algorithm to search values through SPARQL
Input: Queries
Result : Required Informatio

1. Select Vertices
2. from triples
3. Where <condition>

FIG 3. ALGORITHMS FOR PRODUCING TRIPLES AND SEARCH VALUES

Algorithms to process these workflows are given as follows in Figure 3. The Time Complexity of this approach is **O(1**) where the time complexity of earlier approaches are **O(n)** and **O(log n)** as they go for a complete scan and use search techniques.

D. Brief of Scala Program that creates an RDD and stores and queries information of this property Graph

Scala program created for the Property Graph mentioned above creates RDDs called nodes about Person in LinkedIn and RDDs called relationships [22]. This stores information about edges that connect the nodes. For representing, nodes RDDs use long integers such as 3L and 7L which can store extra information about nodes [22]. For example, a person can have other information like Software Engineer as Position and Post-Graduation as a relationship can be represented with 3L. Extra nodes can be added to make SPARQL queries, work a little more. Once the RDDs are created for Nodes and Edges, the initial Graph can be built. Then the code in Scala is applied to produce RDF triples using a base URI. SPARQL queries can be created to retrieve information from this in-memory graph processing system to enhance performance

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

A. Cluster Architecture

It is significant to study the performance of these frameworks and understand the importance of different parameters. In these experiments, cluster performance is presenter based on MapReduce and Spark using the Hi Bench suite. In particular, one Hi bench workload out of thirteen standard workloads is selected to represent here is Tera Sort (shuffle job) with large RDF datasets. In the future, N weight is to be used as workload which is used for Graph benchmarking. It is an iterative Graph parallel algorithm implemented in the Spark Graph X library. It computes associations between two vertices that are n-hop away.

B. Hibench Suite with Workload

Hi Bench is a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput, and system resource utilization. It contains a set of Hadoop, Spark, and streaming workloads, including Sort, Word Count, Tera Sort, Repartition, Sleep, SQL, PageRank, Nutch indexing, Bayes, Kmeans, N Weight, and enhanced DFSIO, etc. It also contains several streaming workloads for Spark Streaming, Flink, Storm, and Gear pump.
Workloads: There are a total of 29 workloads in Hi Bench. The workloads are divided into 6 micro categories, ml (machine learning), SQL, graph, Web search, and streaming. The workload used here is Tera Sort which is a standard benchmark created by Jim Gray. Its input data is generated by the Hadoop Tera Gen example program.

C. Results of Hadoop and Spark with Tera Sort workload with varied input splits task is given below

TABLE III. EXPERIMENTAL HADOOP CLUSTER

Server Configuration	Processor	2.5 GHz
	Main Memory	16 GB
	Local Storage	2 TB
	Operating System	Intel ® Xeon ® CPU ©3.40 GHz
Node Configuration	Main Memory	4 GB
	No. of nodes	4
	Local storage	1 TB each 4 TB total
	Operating System	Ubuntu 16.04.2
Software	Operating System	Ubuntu 16.04.2
	JDK	1.8.0
	Hadoop	2.4.0
	Spark	2.1.0
Workload	Micro Benchmark	Tera soft
Datasets	Db pedia, Kaggle	LinkedIn data

TABLE IV. EVALUATION RESULTS OF RDF GRAPH DATA PROCESSING IN HADOOP AND SPARK IN TERASORT WORKLOAD

Data Size(GB) X	**Execution Time(sec) Y**					
	MR 256MB	**SPARK 256MB**	**MR 512MB**	**SPARK 512MB**	**MR 1024MB**	**SPARK 1024MB**
50	**2347**	**147**	**2347**	**158**	**2347**	**160**
100	**4143**	**1143**	**4145**	**1142**	**4142**	**1142**

150	**6142**	**1243**	**6146**	**1343**	**6143**	**1443**
200	**9147**	**1345**	**12647**	**1544**	**12444**	**1645**
250	**10247**	**2047**	**15147**	**2043**	**15147**	**2045**
300	**20147**	**2258**	**20147**	**2258**	**21554**	**2258**
350	**21114**	**2347**	**22547**	**2347**	**24147**	**2347**
400	**25247**	**2447**	**25245**	**2447**	**25250**	**2447**
450	**26243**	**3153**	**27658**	**3785**	**29149**	**3439**
500	**31256**	**4947**	**31257**	**4757**	**31258**	**3957**
550	**37248**	**5147**	**34148**	**4147**	**32649**	**4047**
600	**42153**	**7648**	**40147**	**6147**	**39147**	**2645**

TABLE V. **THE BEST EXECUTION TIME OF MAPREDUCE AND SPARK WITH TERASORT WORKLOAD**

Workload	Split sizes (MB)	Execution time (sec)
MapReduce input splits (Tera Sort)	256	21114
Spark input splits (Tera Sort)	512 & 1024	3785 & 3439

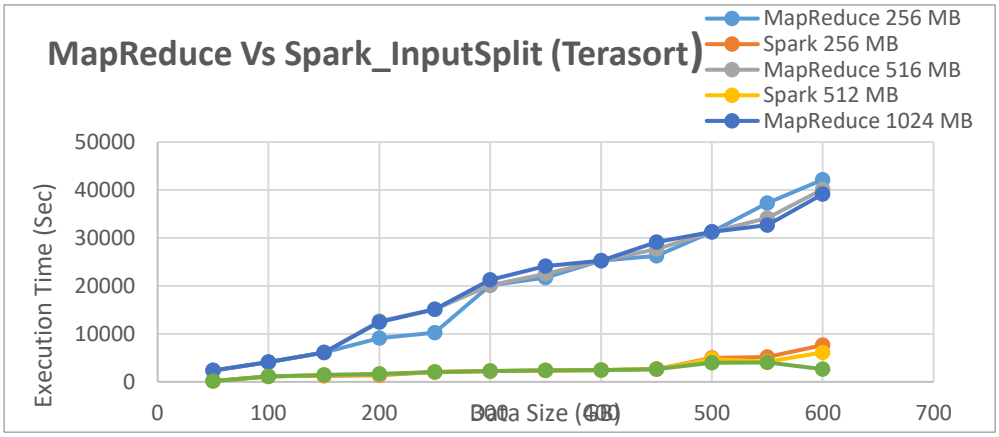

FIG 4. VISUALIZATION OF EXECUTION TIME FOR VARIOUS SIZED RDF GRAPH DATA PROCESSING IN HADOOP AND SPARK WITH TERASORT WORKLOAD

VI. CONCLUSION AND FUTUREWORK

In this work, the time complexity of the proposed work has been improved and performance analysis of RDF Graph Data Processing with Hadoop-MapReduce and Spark-Graph X has been done. This work can be further extended by evaluating the performance of RDF graph data processing in other cloud platforms such as Microsoft Azure public IaaS clouds, Google Compute Engine, or Rackspace with varying data size up to 10 to 100 terabytes (TB) of data and with workload N weight in Hi bench suite. This study helped to illustrate the improvement in the performance while running through SPARK platform added with Graph X library. This work is the commencement to drive for semantic graph analytics to further the depth of research. Future work can include deriving more insights from this structure like unknown relationships. Pattern matching. Connectivity Analytics and Centrality Analytics are some other research areas.

REFERNCES

1. Mc Bride,"Jena: Implementing the RDF model and syntax specification." in SemWeb'01: Proceedings of the Second International Conference on Semantic Web - Volume 40, May 2001 Pages 23–28. Available from https://dl.acm.org/doi/10.5555/2889966.2889969.
2. M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "SPARQL basic graph pattern optimization using selectivity estimation," in Proceedings of the 17th international conference on World Wide Web. ACM, 595–604, 2008. Available from https://dl.acm.org/doi/10.1145/1367497.1367578
3. Heath, T., and Bizer, C., "Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology", 2011. Available from https://dl.acm.org/doi/10.1145/1772690.1772907
4. Gosal, K. J. Kochut, and N. Kannan, "Prokino: an ontology for integrative analysis of protein kinases in cancer" PloS one, 6(12), 2011. Available from https://doi.org/10.1371/journal.pone.0028782
5. J. Huang, D. J. Abadi, and K. Ren, "Scalable SPARQL querying of large RDF graphs," Proceedings of the VLDB Endowment, 4(11), 1123–1134, 2011. Available from https://dl.acm.org/doi/10.14778/3402707.3402747
6. P. Peng, L. Zou, M. T. O¨ zsu, L. Chen, and D. Zhao, "Processing SPARQL queries over linked data–a distributed graph based approach," arXiv preprint arXiv:1411.6763, 2014. Available from https://arxiv.org/abs/1411.6763
7. Reynold S. Xin, Daniel Crankshaw, Ankur Dave, Joseph E. Gonzalez, Michael J. Franklin Ion Stoica, "GraphX: Unifying Data-Parallel and Graph-Parallel analytics", SPARK Summit 2014. Available from https://arxiv.org/abs/1402.2394;
8. Mohammed Guller," Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analytics", 2015. Available from https://link.springer.com/book/10.1007/978-1-4842-0964-6
9. Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark (SIGMOD). 1383–1394.Available from https://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf
10. M. Zaharia et al. "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing", NSDI, 2012. Available from http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf

11. James A. Scott., “Getting started with Apache Spark”, MapR technologies Inc, 2015, First Edition. Available from
12. https://www.bigdata-toronto.com/2016/assets/getting_started_with_apache_spark.pdf
13. John A. Miller, Lakshmish Ramaswamy, Krys J. Kochut, and Arash J.Z. Fard, "Directions for Big Data Graph Analytics Research," International Journal of Big Data, (IJBD), Vol. 2, No. 1 (September 2015) pp. 15-27. Available from http://cobweb.cs.uga.edu/~jam/papers/
14. John A. Miller, Lakshmish Ramaswamy, Krys J. Kochut, and Arash Fard, "Research Directions for Big Data Graph Analytics," Proceedings of the 4th IEEE International Congress on Big Data (ICBD'15), New York, New York (June-July 2015) pp. 785-794. Available from DOI: 10.1109/BigDataCongress.2015.132
15. Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). USENIX Association, Broomfield, CO, 599--613. Available from
16. https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-gonzalez.pdf
17. Alfredo Cuzzocrea, Mirel Cosulschi, and Roberto de Virgilio, “An Effective and Efficient MapReduce Algorithm for Computing BFS-Based Traversals of Large-Scale RDF Graphs”, Algorithms, 2016. Available from https://www.mdpi.com/1999-4893/9/1/7
18. Amir H. Payberah, “A Crash course in Scala”, 2015. Available from https://www.sics.se/~amir/files/download/dic/scala.pdf
19. Amir H. Payberah, “Spark and Resilient Distributed Datasets”, 2015. Available from https://www.sics.se/~amir/files/download/dic/spark.pdf
20. Michael J. Lewis, George K. Thiruvathukal, Michael J. Papka, Andrew Johnson, “A Distributed Graph Approach for Pre-processing Linked RDF Data Using Supercomputers”, Journal of Bigdata,2020. Available from https://dl.acm.org/doi/10.1145/3066911.3066913 and
21. https://ecommons.luc.edu/cgi/viewcontent.cgi?article=1141&context=cs_facpubs
22. Hubert Naacke, Oliver Cure, Bernd Amann, “SPARQL Query Processing with Apache Spark”,ACM, 2016. Available from https://arxiv.org/abs/1604.08903
23. Hubert Naacke, Oliver Cure, Bernd Amann, “SPARQL Graph Pattern Processing with Apache Spark”, ACM, 2017. Available from https://dl.acm.org/doi/10.1145/3078447.3078448
24. Baby Nirmala. M., Sathiaseelan. J.G.R.,” Big semantic data analytics: A theoretical study on the implementation of technologies and tools”, International Journal of Applied Engineering Research, ISSN 0973-4562, 10(20), 2015. Available from https://www.ripublication.com/Volume/ijaerv10n20spl.htm
25. M. Baby Nirmala , J.G.R Sathiaseelan “An Innovative Approach to RDF Graph Data Processing: Spark with GraphX and Scala ”, IJCTA, 9(26) 2016, pp. 425-434© International Science Press. Available from
26. https://serialsjournals.com/abstract/46497_55.pdf
27. Michelle Knight, Property Graph. Available from https://www.dataversity.net/what-is-a-property-graph/ on 13-Apr-2022.
28. Notes on Reification. Available from https://jena.apache.org/documentation/notes/reification.html on 13-Apr-2022.

29. Vinh Nguyen, Oliver Boldenreider, Amit Sheth, “Don't like RDF reification?: making statements about statements using singleton property”, WWW '14: Proceedings of the 23rd international conference on World wide web, April 2014, Pages 759–770. Available from http://dl.acm.org/citation.cfm?id=2567973
30. Georgios Drakopoulos, Vasileios Megalooikonomou, “A Graph Framework for Multimodal Medical Information Processing”, eTELEMED 2016 : The Eighth International Conference on eHealth, Telemedicine, and Social Medicine (with DIGITAL HEALTHY LIVING 2016 / MATH 2016). Available from https://frailsafe-project.eu/images/frailsafe/results/eTELEMED-2016- UoP.pdf
31. Dong Wang, Lei Zou, and Dongyan Zhao, “gst-store: Querying Large Spatiotemporal RDF Graphs”, Data and Information Management2017; 1(2):84–103. Available from DOI: https://doi.org/10.1515/dim-2017-0008
32. N. Ahmed, Andre L. C. Barczak , Teo Susnjak and Mohammed A. Rashid, “A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using Hi Bench”, Journal of Big Data (2020) 7:110. Available from https://doi.org/10.1186/s40537-020-00388-5