

Efficient Processing of SPARQL Queries Over GraphFrames

Ramazan Ali Bahrami
Department of Computer Science
South Asian University, Delhi, India
bahram.ramazanali@gmail.com

Jayati Gulati
Department of Computer Science
South Asian University, Delhi, India
gulati.jayati@gmail.com

Muhammad Abulaish, SMIEEE*
Department of Computer Science
South Asian University, Delhi, India
abulaish@sau.ac.in

ABSTRACT

With the advent of huge data management systems storing voluminous data, there arises a need to develop efficient data analytics techniques for knowledge discovery at different levels of granularity. Resource Description Framework (RDF), mainly developed for Semantic Web, is presumably a good option when considering graph databases dealing with huge real-world data. RDF models information in the form of triples $\langle \text{subject}, \text{predicate}, \text{object} \rangle$, and is considered as a useful tool to store graph data (aka linked data) where each edge can be stored as a triple. Due to existence of huge amount of linked data, mostly in the form of graphs, graph mining has been successful in attracting researchers from different research fields for efficient handling (storage, indexing, retrieval, etc.) of graph data. As a result, various APIs like GraphX and GraphFrames are developed to facilitate relational queries over graph data. Though GraphX is older than GraphFrames and processing SPARQL queries over GraphX has been explored by some researchers, to the best of our knowledge, SPARQL query processing over GraphFrames has not been explored yet. In this paper, we present an initial study on query-specific search space pruning and query optimization approach to process SPARQL queries over GraphFrames in an efficient manner. The experimental results, in terms of low response time for query execution, are encouraging, and give way to invest more research efforts in this direction.

KEYWORDS

Graph mining, Linked data mining, SPARQL query processing, GraphFrames, GraphX

1 INTRODUCTION

The exponential growth in the amount of information available online has swayed a need for data analysis at different levels of granularity. Started with statistical data analysis focusing on drawing inferences from large data using certain summarization and interpretation techniques, the emerging concept of big data analytics aims to deal with huge amount of varied and varying data for pattern discovery and uncovering correlations. In general, it is not possible to treat such data using traditional data analysis

approaches [17]. Indexing techniques like MapReduce, based on various tools such as MyHadoop, MongoDB [2] are used to handle voluminous data. However, some of the issues in handling big data are related to the analysis and quality of data, efficiency of search algorithms, visualization of results, fault tolerance, and handling heterogeneous data and its security [2, 11].

The data analysis process becomes more challenging when data items are not independent, rather they are linked together or to some other data sources available on social media, and hence visualized in the form of graphs [1]. Joining the dots may reveal some important non-trivial information, which is otherwise not possible to reveal through analyzing data items independently. However, the task of fusing all such information together complicates the whole data processing task, starting from data storage and data indexing to data retrieval. It can be seen that the analysis of linked data in huge graphs such as subgraph isomorphism [14], shortest paths between the pair of nodes [8] are computationally expensive and most of them are NP-hard. As a result, some of the research efforts have been directed to devise efficient graph data processing techniques for analyzing linked data.

Exploring graph-oriented nature of Semantic Web has become an ideal subject for analysis. Some progress in this direction is the development of GraphX and GraphFrames APIs for processing SPARQL (Simple Protocol and RDF Query Language) queries over graph data that are stored in RDF (Resource Description Framework) format, although the RDF format is generally used to represent heterogeneous information on the Web via a labeled directed graph¹. It consists of a triple in the form of $\langle \text{subject}, \text{predicate}, \text{object} \rangle$, wherein the predicate represents the relation shared by the subject and object [12]. A node in RDF graph is represented by URI (Uniform Resource Identifier), literal or a blank node². SPARQL is a semantic query processing language to handle relational queries over RDF data. The clauses in the query are in the form of triples, $\langle ?x, R, ?y \rangle$ where variable names are preceded by '?' symbol. In this triple, x and y represent subject and object respectively, joined by a predicate R [7]. The set of such triple patterns is called Basic Graph Pattern (BGP).

Initially graph processing tasks like subgraph matching were based on GraphX framework. In [5], SPARQL queries are evaluated distributively using the graph computation framework of GraphX API. The query processing is based on query plan generator, which creates an evaluation order and direction of messages to be sent via edges. However, the proposed approach suffers with an overhead due to existence of loop edges, which requires a number of messages to be passed [5]. Also, the analysis of ordering of triple patterns has not been acknowledged in this work. Authors in [10], present an algorithm for iterative matching of BGP triples from the query with

* Correspondence author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WI '17, Leipzig, Germany

© 2017 ACM. 978-1-4503-4951-2/17/08...\$15.00

DOI: 10.1145/3106426.3106534

¹<https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>

²<https://www.w3.org/TR/rdf-sparql-query/>

graph dataset using RDD (Resilient Distributed Dataset) in GraphX. The drawback of this technique is lack of query optimization via optimal evaluation of BGP triples.

In this paper, we present an efficient processing technique of SPARQL queries using GraphFrames API, which is a new graph processing platform developed over Apache Spark using the concept of dataframes. It represents two dataframes – an edge dataframe and a vertex dataframe. The GraphFrames supports query processing, pattern matching and optimizing computations across joins [4]. Along with the capabilities of GraphX, it provides additional functionality executing queries based on views stored and maintained by the system³. The GraphFrames query planner itself suggests views to rewrite the query for efficient results. This paper introduces an optimal processing of SPARQL query utilizing the dataframe approach. In addition, a basic approach for optimizing the response time of SPARQL query over GraphFrames and its implementation is proposed. Our method employs GraphFrames API for reducing query search space and argues for the reordering of SPARQL query statements for faster execution.

The rest of the paper is organized as follows. Section 2 presents a brief overview of the existing state-of-the-art on graph data processing. Section 3 presents our proposed approach for processing SPARQL queries over GraphFrames and their optimization. In section 4, we describe our experimental setup and evaluation results. Finally, section 5 concludes the paper with future directions of research.

2 RELATED WORK

The research in the field of data analytics initially started with simple statistical methods like summarization [17]. However, for analysis of huge data, Hadoop and several other platforms following the MapReduce paradigm have been developed by the research community [13]. Hadoop framework is used for distributed data processing, which can scale up from a single server to multiple machines. Further, a system called H2RDF is developed for combining MapReduce features with NoSQL data store, enabling processing of simple as well as multi-join queries [16]. However, the drawback of Hadoop like systems is that they reside in disk, resulting in excess execution time[5].

To handle large distributed graphs, a scalable and fault-tolerant platform called *Pregel* is developed in [15]. For each iteration, a user-defined function on each vertex sends and receives messages to and from other vertices respectively. Though the implementation is based on main memory, with some data residing on disk, it cannot handle terabytes of data. Also, the partitioning of vertices onto specific machines is a static process.

Later, *Spark* came up with an in-memory based framework for query evaluation called *GraphX*. It is one of the graph processing systems running on the *Spark* data parallel framework [18]. *GraphX* provides data abstraction in the form of RDD (Resilient Distributed Dataset) [10], which increases the effectiveness of each iteration as the required data is always in memory. In [10], *GraphX* framework is employed for subgraph (or query pattern) matching, through matching RDF graph data with BGP triples extracted from SPARQL query. For example, the query shown in figure 3(a) has

two BGP triples. Though the order of clause matching, i.e., BGP triples extracted from the query, with the given graph data affects the optimality of the search operation, ordering of clauses is not taken into consideration in the technique proposed in [10].

In [19], the authors proposed a system for scalable and efficient handling of SPARQL queries based on *gStore*. Their system handles queries consisting of aggregate functions and wildcard operators over dynamic datasets. Another similar work related to the evaluation of SPARQL queries using functions on vertices is implemented in [7]. It uses *GraphLab* as the platform for execution of Semantic Web queries. However, the implementation suffers with the overhead of converting RDF data into the format interpretable by the *GraphLab*.

As discussed above, *Pregel* and *GraphLab* are graph-parallel approaches based on vertex programming models. They partition the vertices among several machines to reduce computation load. However, a real-world data graphs like online social networks follow power law degree distributions, implying that a small subset of the vertices connect to a large fraction of the graph. These type of graphs are difficult to partition for a distributed environment. To address these challenges *PowerGraph* is introduced, benefitting from the structure of vertex-programs and computation over edges as an alternative to vertices [6]. It promotes greater parallelism, reducing network communication and cost. A study in [3] presents *Graphalytics*, a framework for benchmarking big data for graph-processing platforms, which is best suited for data-intensive algorithms. The implementation of one such platform is *Neo4j*⁴.

In this paper, we use *Spark* framework for graph processing, called *GraphFrames*. The queries fired for analysis using *GraphFrames* are broken into fragments and matched against the views stored previously. It models multiple views of graphs and matches patterns iteratively [4]. The concept of dataframe provides interactive queries. It also enables parallel execution and expedites completion of interactive queries by registering the appropriate view [4].

3 PROPOSED APPROACH

In this section, we present our proposed approach for efficient processing of SPARQL queries over GraphFrames. The aim is to query RDF graph data in an efficient and optimized manner using GraphFrames API. To retrieve results at a faster pace, search space pruning and ranking of query clauses is also implemented. For matching a given SPARQL query graph (aka subgraph) the extracted BGP triples from query graph are matched against the RDF graph data (dataset to be queried). It is studied that the order in which the clauses of a query are fired, affects the processing time of the query. The research work presented in [5, 10] highlights the importance of sequencing BGP triples for better performance of the system. It also suggests that direct loading of graph data into RDD saves time, in comparison to loading it into memory and then transferring back to RDD [10]. For reducing query search space (aka graph data pruning), certain predicates (or the edge labels) that need not to be matched as per the given query are removed. This reduces the number of candidate values for variables in the query, also limiting the data to be matched. Moreover, the query is optimized by

³<https://graphframes.github.io/>

⁴<https://neo4j.com/product/>

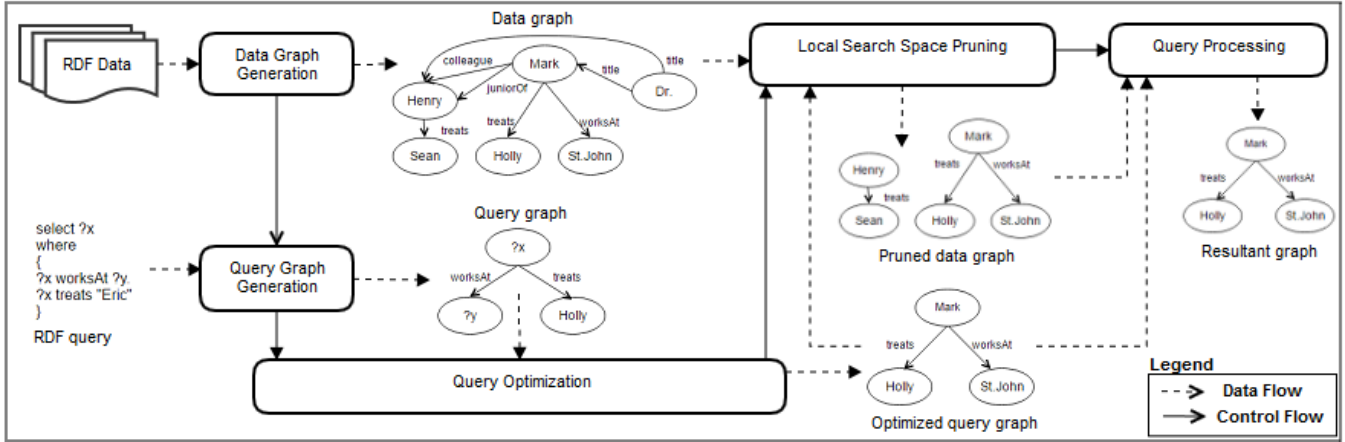


Figure 1: Work-flow diagram of the proposed approach for efficient processing of SPARQL queries over GraphFrames

matching certain triples prior to others based on a ranking criteria. The idea behind query optimization is to check different clauses of the given query and rearrange them according to a ranking criteria to minimize response time.

Figure 1 presents the work-flow of the proposed approach for processing SPARQL queries over GraphFrames, which consists of five working modules – *data graph generation*, *query graph generation*, *query optimization*, *local search space pruning*, and *query processing* to perform various inter-related tasks. Functioning details of these modules are presented in the following sub-sections.

3.1 Data Graph Generation

The data graph generation module aims to scan RDF graph dataset and generate two separate lists as CSV files – (i) *nodeList* containing the list of all node labels, and (ii) *edgeList* containing all edges connecting a pair of nodes in *nodeList*. The edgeList contains edge information in the form of $\langle \text{subject}, \text{predicate}, \text{object} \rangle$ triples. Using *nodeList* and *edgeList*, RDF data can be modeled as an unweighted and directed graph.

For an exemplar data given in the following paragraph, the list of extracted $\langle \text{subject}, \text{predicate}, \text{object} \rangle$ triples from different statements is shown in table 1. It should be noted that in many cases same node acts as a subject for one triple and as an object for another triple. All such triples extracted from the original data are stored in a separate file for answering SPARQL queries.

Exemplar Data: *Dr. Mark, who is junior of Dr. Henry, works at St. John hospital. Dr. Henry and Dr. Pam work at St. Paul. Dr. Henry is treating Eric and Dr. Pam is treating Tanya.*

Figure 2 presents a visualization of the RDF data triples given in table 1.

3.2 Query Graph Generation

As shown in Figure 1, this module aims to analyze SPARQL queries to identify BGP triples for query graph generation. The BGP triples form the fundamental units for query graph, which is searched in the RDF data graph for query answering. There are generally two types of SPARQL queries – (i) Chain queries, and (ii) Star-join

Table 1: RDF triples extracted from the exemplar data given in section 3.1

Subject	Predicate	Object
Mark	juniorOf	Henry
Mark	worksAt	St. John
Mark	colleague	Henry
Dr.	title	Mark
Dr.	title	Henry
Dr.	title	Pam
Henry	treats	Eric
Henry	worksAt	St. Paul
Pam	worksAt	St. Paul
Pam	treats	Tanya

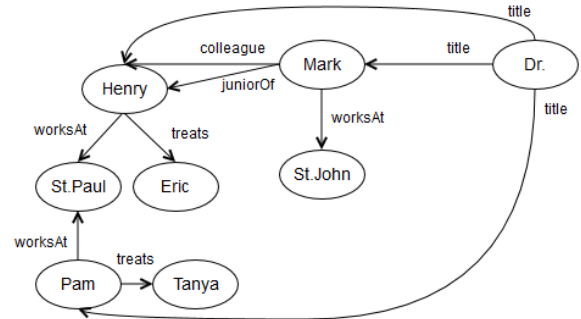


Figure 2: Data graph corresponding to the RDF triples given in table 1

queries. In *chain queries*, object of one triple is used as the subject of another triple, whereas in *star-join queries* all triples share the same subject [10]. For example, “ $?p \text{ relatedTo } ?q, ?q \text{ manages budget}$ ” is a chain query, whereas “ $?p \text{ relatedTo } ?q, ?p \text{ supervisedBy } ?r$ ” is a star-join query. Figure 3 presents an exemplar SPARQL query, extracted BGP triples from the query statements, and query graph.

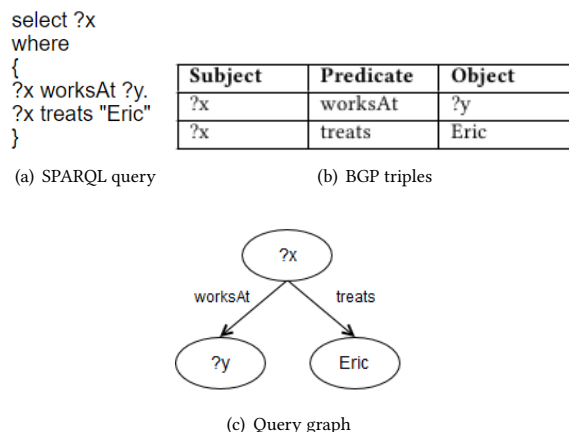


Figure 3: An exemplar (a) SPARQL query, (b) Extracted BGP triples from the query and (c) Query graph

3.3 Query Optimization

The *query optimization* module takes BGP triples extracted from an SPARQL query as input and reorder the query clauses to generate an optimized query execution plan. For an SPARQL query with n clauses there can be $n!$ number of query execution plan with the clauses in a specific order, which is a very large number if the query has many clauses. Therefore, the aim of this module is to identify the best possible order of firing the query clauses such that the response time of the subgraph (query) matching is lowest.

In order to determine the optimal ordering of clauses in a query, we consider the frequency count of each predicate (or edge) in the underlying RDF data graph. The least frequent occurring edges in RDF data graph are given higher priority, and accordingly the clauses of the SPARQL queries are rearranged in non-decreasing order of their respective predicates' frequency count. This reordering can be proved to be beneficial in terms of response time reduction for huge data graphs. The overhead of frequency calculation is very low as compared to the results it may provide – as it needs to be calculated only once for a given data graph. Hence, this technique can be deployed for optimizing the subgraph (query graph) matching process.

Table 2: Frequency count of the edges (predicates) in the RDF data graph shown in figure 2

Edge Label	Frequency
juniorOf	1
colleague	1
treats	2
title	3
worksAt	3

Table 2 shows the frequency count of the edges in the RDF data graph shown in figure 2. The SPARQL query in Figure 3(a) comprises of two clauses (each of which forms a BGP triple) as shown in figure 3(b). Hence, there can be two ways in which the

query clauses can be ordered – (i) $\langle ?x \text{ worksAt } ?y \rangle$ followed by $\langle ?x \text{ treats } ?Eric \rangle$, and (ii) $\langle ?x \text{ treats } ?Eric \rangle$ followed by $\langle ?x \text{ worksAt } ?y \rangle$. Since the frequency count of the triple $\langle ?x \text{ treats } ?Eric \rangle$ is lower than the frequency count of $\langle ?x \text{ worksAt } ?y \rangle$, the second ordering of the query clauses is considered as the optimized query execution plan.

3.4 Local Search Space Pruning

Like query optimization module, this module also aims to minimize response time of query execution through reducing the number of RDF triples to be searched in the dataset. Since both RDF data graph and SPARQL query graph are converted by the *Data Graph Generation* and *Query Graph Generation* modules respectively in the form of $\langle \text{subject}, \text{predicate}, \text{object} \rangle$ triples, a table lookup mechanism is implemented to remove irrelevant triples from data graph with respect to the given query with the help of an in-built function of GraphFrames library. This reduces the number of triples in the RDF file, leading to a substantial amount of reduction in the search space (original data graph). Since the removal of triples from original data graph is always with respect to a particular query, this pruning is termed as local search space pruning.

For example, consider the RDF triples extracted from the RDF dataset given in table 1 and the BGP triples extracted from the SPARQL query of figure 3(a) shown in figure 3(b). Considering only the predicate values of BGP triples, it can be observed that only those edges of the data graph labeled as either “worksAt” or “treats” are required by this query. Therefore, RDF graph data is scanned for these predicate values and remaining predicate values along with their subjects and objects data are removed from the original dataset to create a temporary reduced dataset for query processing. For the given SPARQL query in figure 3(a), the pruned RDF triples and the corresponding data graph are shown in table 3 and figure 4, respectively.

Table 3: Pruned RDF data triples with respect to the SPARQL query shown in figure 3(a)

Subject	Predicate	Object
Henry	treats	Eric
Henry	worksAt	St.Paul
Pam	worksAt	St.Paul
Pam	treats	Tanya
Mark	worksAt	St.John

3.5 Query Processing

This module takes the optimized BGP triples and locally pruned RDF data triples as inputs and performs a subgraph matching to identify relevant triples for the query. The subgraph (or query) matching starts with the first clause of the query. A clause consisting of a fixed value for a vertex (subject/object) would further narrow down the search space, reducing the response time of the query. This is because, fixed value for either subject or object leads to the reduction in the candidate set of values for the other vertex variables joined by an edge (converse of subject/object). On the contrary, if there is no fixed-value vertex in the given query, then the set of candidate

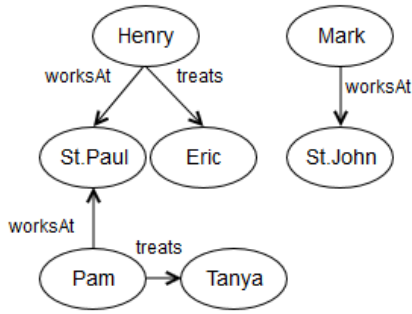


Figure 4: Data graph corresponding to the pruned RDF triples shown in table 3

values for subject/object is equal to the entire pruned dataset. This reasoning works well when the size of the data is huge. Figure 5 exemplifies the subgraph (or query) matching process. It also shows the number of triples extracted against normal ordering of query clauses (*order-1*) and the optimized ordering of the query clauses (*order-2*). Figure 6 shows the resultant RDF triples and the corresponding data graph for the SPARQL query of figure 3(a).

4 EXPERIMENTAL SETUP AND RESULTS

This section presents the experiment setup and evaluation results of the proposed approach for efficient SPARQL query processing over GraphFrames. The dataset used in this study is produced by the Lehigh University benchmark⁵ (LUBM) data generator [9]. A synthetic OWL (Web Ontology Language) dataset for a university which corresponds to a university domain ontology is created. Table 4 presents the statistics of the generated dataset. The LUBM data is converted into RDF format using the JAR file *rdf2rdf-1.0.1-2.3.1.jar*. A data preprocessing module is implemented using Java to extract nodes and edges from RDF data and store them into CSV files. Table 5 shows the original queries fired on the LUBM dataset and the optimized version of these queries using the optimization process discussed in section 3.3 is shown in Table 6.

For a given query, if all edges in the query have fixed labels then they are filtered to create a new pruned graph. This idea is implemented by filtering the given dataset based on the edge labels from the individual clauses of the query. Hence, we take into account the frequency of the edge labels that are part of the query for deciding the order of filtering. The edges of the dataset are ranked in non-decreasing order of their frequency count and the edge having least count is considered as highest ranked edge. Filtering of the ranked edges can be done using two approaches. In first approach (*Approach-I*), edges are filtered based on their ranks, prior to motif creation. The *motif* is a template created from the query for pattern matching and data retrieval. As an example, for the query, “*?X <takesCourse> <GraduateCourse6>; ?X <type> <GraduateStudent>*”, the corresponding motif will be $(X)-[e1]->(Y); (X)-[e2]->(Z)$, which can be represented as a relation schema $Motif(X, e1, Y, X, e2, Z)$. This implies that there are two edges originating from node X to two different nodes Y and Z . The second approach (*Approach-II*) first retrieves the motif and then

⁵<http://swat.cse.lehigh.edu/projects/lubm/>

filters the edges according to their ranks. Though the first approach gives a lower execution time, it works only when all edge labels are known. Hence, it is observed that in most of the cases, first approach of edge filtering gives efficient results, as shown figure 7.

Table 7 shows the partial results obtained through executing the optimized queries given in table 6. Figure 8 presents a comparison results in terms of execution times (in seconds) for normal and optimized queries. It can be observed from this figure that filtering and reordering works in majority of the cases. Despite of the fact that this approach may not provide much difference in execution times for small datasets, the cost of reordering depends on frequency calculation of the edges, which takes a linear scan of the original dataset. But, it seems very useful for processing SPARQL queries over large graph datasets.

Table 4: A statistics of LUBM graph dataset

Vertex categories and their numbers	Total #vertices	Total #edges
Department: 15	17191	67466
FullProfessor: 125		
AssociateProfessor: 176		
AssistantProfessor: 146		
Lecturer: 93		
UndergraduateStudent: 5916		
GraduateStudent: 1874		
TeachingAssistant: 407		
ResearchAssistant: 547		
Course: 828		
GraduateCourse: 799		
Publication: 5999		
ResearchGroup: 224		

5 CONCLUSION

In this paper, we have presented an efficient SPARQL query processing approach over GraphFrames. Starting from extracting triples from RDF graph data, we have shown how ordering of query clauses and pruning of original graph dataset help in reducing response time of SPARQL queries. It can be observed from the experimental results that the time needed to fetch data from graph can be curtailed by reducing the search space using the predicates of the query clauses. Also, reordering the query clauses is beneficial in reducing search time for subgraph (or query) matching. The proposed approach works well for large datasets. For smaller dataset, the effect of query clause reordering becomes data dependent, which can be improved in our work by taking into account the frequency count of the vertices along with the edge labels.

REFERENCES

- [1] Charu C Aggarwal. 2011. An introduction to social network data analytics. *Social network data analytics* (2011), 1–15.
- [2] Vibha Bhardwaj and Rahul Johari. 2015. Big data analysis: Issues and challenges. In *2015 International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO)*.
- [3] Mihai Capotă, Tim Hegeman, Alexandru Iosup, Arnau Prat-Pérez, Orri Erling, and Peter Boncz. 2015. Graphalytics: A Big Data Benchmark for Graph-Processing Platforms. In *Proceedings of the GRADES’15*. ACM, New York, USA, 7:1–7:6.

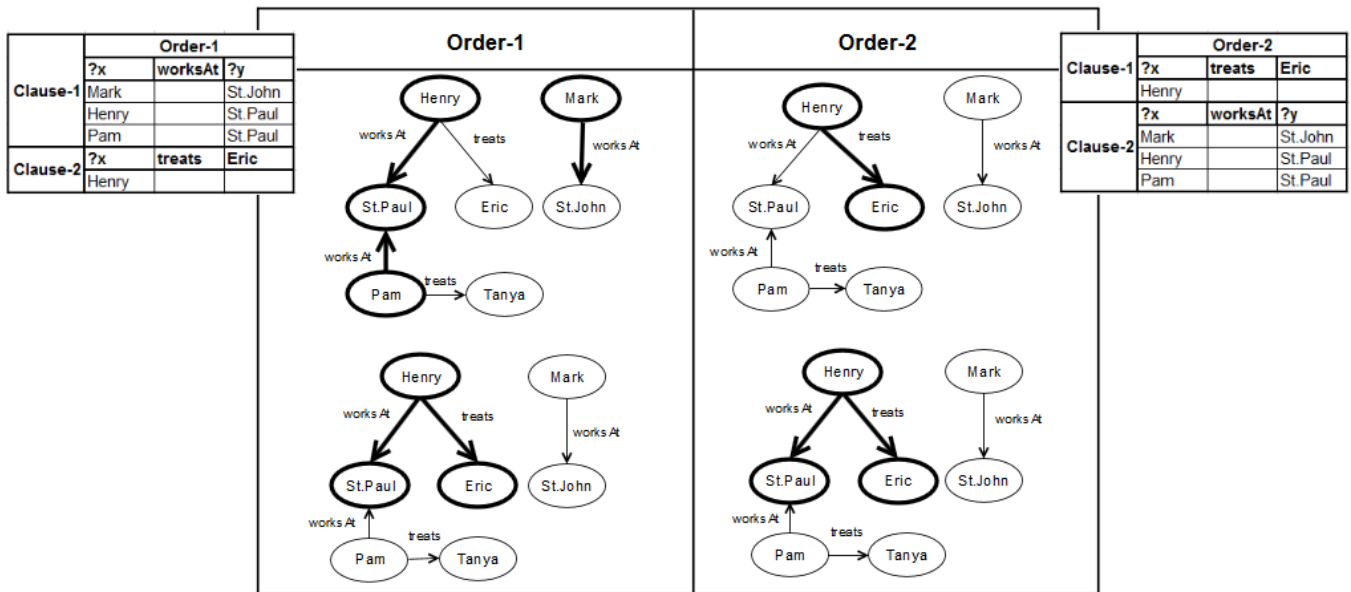


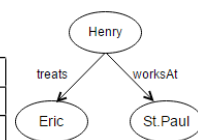
Figure 5: An illustration of subgraph (or query) matching process

Table 5: A list of six exemplar SPARQL queries

Query ID	SPARQL Query
Q1	?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse? ;http://www.Department3.University0.edu/GraduateCourse6> . ?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent>
Q2	?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#publicationAuthor> <http://www.Department7.University0.edu/AssociateProfessor12> . ?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?Y
Q3	?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse> ?Z . ?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent>
Q4	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#ResearchGroup> . ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf> <http://www.Department8.University0.edu>
Q5	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent> . ?Y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department> . ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf> ?Y
Q6	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent> . ?Y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#University> . ?Z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department> . ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf> ?Z

Subject	Predicate	Object
Henry	treats	Eric
Henry	worksAt	St.Paul

(a) RDF triples



(b) Data graph

Figure 6: Resultant (a) RDF triples, and (b) Data graph after executing the SPARQL query of figure 3

- [4] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. 2016. Graphframes: an integrated api for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*. ACM, 2.
- [5] Gergo Gombos, Gábor Rácz, and Attila Kiss. 2016. Spar (k) ql: SPARQL evaluation method on Spark GraphX. In *Future Internet of Things and Cloud Workshops (FiCloudW)*, *IEEE International Conference on*. IEEE, 188–193.
- [6] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *OSDI*, Vol. 12. 2.
- [7] Eric L Goodman and Dirk Grunwald. 2014. Using vertex-centric programming platforms to implement SPARQL queries on large graphs. In *Proceedings of the 4th Workshop on Irregular Applications: Architectures and Algorithms*. IEEE Press, 25–32.

Table 6: Optimized version of the SPARQL queries given in Table 5

Query ID	Optimized SPARQL Query
OptQ1	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent> . ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse> <http://www.Department3.University0.edu/GraduateCourse6>
OptQ2	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?Y. ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#publicationAuthor> <http://www.Department7.University0.edu/AssociateProfessor12>
OptQ3	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent> . ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse> ?Z
OptQ4	?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf> <http://www.Department8.University0.edu> . ?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#ResearchGroup>
OptQ5	?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf> ?Y . ?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent> . ?Y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department>
OptQ6	?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent> . ?Y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#University> . ?Z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department> . ?X <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf> ?Z

Table 7: Execution results of the optimized SPARQL queries given in Table 6

Query ID	#Triples shown/ #Retrieved triples	Resultant triples
OptQ1	3/4	1. http://www.Department3.University0.edu/GraduateStudent80 2. http://www.Department3.University0.edu/GraduateStudent13 3. http://www.Department3.University0.edu/GraduateStudent16
OptQ2	3/14	1. http://www.Department7.University0.edu/AssociateProfessor12/Publication4, http://swat.cse.lehigh.edu/onto/univ-bench.owl#Publication 2. http://www.Department7.University0.edu/AssociateProfessor12/Publication2, http://swat.cse.lehigh.edu/onto/univ-bench.owl#Publication 3. http://www.Department7.University0.edu/AssociateProfessor12/Publication9, http://swat.cse.lehigh.edu/onto/univ-bench.owl#Publication
OptQ3	3/3738	1. http://www.Department0.University0.edu/GraduateStudent52, http://www.Department0.University0.edu/GraduateCourse16 2. http://www.Department1.University0.edu/GraduateStudent61, http://www.Department1.University0.edu/GraduateCourse36 3. http://www.Department1.University0.edu/GraduateStudent82, http://www.Department1.University0.edu/GraduateCourse22
OptQ4	3/18	1. http://www.Department8.University0.edu/ResearchGroup12 2. http://www.Department8.University0.edu/ResearchGroup11 3. http://www.Department8.University0.edu/ResearchGroup14
OptQ5	3/1874	1. http://www.Department11.University0.edu/GraduateStudent79, http://www.Department11.University0.edu 2. http://www.Department4.University0.edu/GraduateStudent3, http://www.Department4.University0.edu 3. http://www.Department4.University0.edu/GraduateStudent6, http://www.Department4.University0.edu
OptQ6	4/1834646	1. http://www.Department11.University0.edu/GraduateStudent79, http://www.University740.edu, http://www.Department11.University0.edu 2. http://www.Department11.University0.edu/GraduateStudent79, http://www.University453.edu, http://www.Department11.University0.edu 3. http://www.Department11.University0.edu/GraduateStudent79, http://www.University157.edu, http://www.Department11.University0.edu 4. http://www.Department11.University0.edu/GraduateStudent79, http://www.University252.edu, http://www.Department11.University0.edu

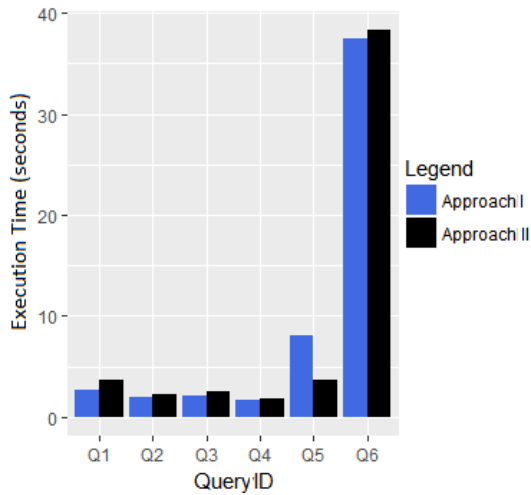


Figure 7: Comparison of two approaches (*Approach-I* and *Approach-II*) of graph filtering

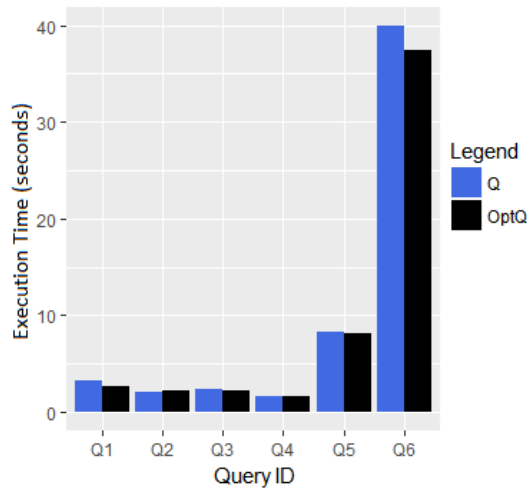


Figure 8: Comparison of execution times of original and optimized SPARQL queries

[8] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weikum. 2010. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 499–508.

[9] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web* 3, 2 (2005), 158–182.

[10] Besat Kassaie. 2017. SPARQL over GraphX. *arXiv preprint arXiv:1701.03091* (2017).

[11] Avita Katal, Mohammad Wazid, and RH Goudar. 2013. Big data: issues, challenges, tools and good practices. In *Contemporary Computing (IC3), 2013 Sixth International Conference on*. IEEE, 404–409.

[12] Graham Klyne and Jeremy J. Carroll. 2006. Resource Description Framework (RDF): Concepts and Abstract Syntax. (2006).

[13] Chang Liu, Jun Qu, Guilin Qi, Haofen Wang, and Yong Yu. 2012. Hadoopsparql: a hadoop-based engine for multiple sparql query answering. In *Extended Semantic Web Conference*. Springer, 474–479.

[14] Anna Lubiw. 1981. Some NP-complete problems similar to graph isomorphism. *SIAM J. Comput.* 10, 1 (1981), 11–21.

[15] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 135–146.

[16] Nikolaos Papailiou, Ioannis Konstantinou, Dimitrios Tsoumakos, and Nectarios Koziris. 2012. H2RDF: adaptive query processing on RDF data in the cloud. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, 397–400.

[17] Robert F Woolson and William R Clarke. 2011. *Statistical methods for the analysis of biomedical data*. Vol. 371. John Wiley & Sons.

[18] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. 2013. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2.

[19] Lei Zou, M Tamer Özsu, Lei Chen, Xuchuan Shen, Ruizhe Huang, and Dongyan Zhao. 2014. gStore: a graph-based SPARQL query engine. *The VLDB journal* 23, 4 (2014), 565–590.