QUERYING AND DATA MINING ON PARTITIONED KNOWLEDGE GRAPHS

by

AMITABH PRIYADARSHI

(Under the Direction of Krzysztof J. Kochut)

ABSTRACT

Large-scale knowledge graphs with billions of nodes and edges are increasingly common in many application areas. Their large sizes often exceed the limits of systems storing the graphs in a centralized data store, especially if placed in main memory. To overcome some of the problems related to their size, knowledge graphs can be partitioned into multiple sub-graphs and distributed as shards among many computing nodes for further processing. Querying on these shards, due to distributed joins, increases the communication cost, but by reducing the edge cuts and continually re-partitioning, based on the given or changing query workload, maintains a good average processing time. Graph embedding on knowledge graphs converts a graph into a vector space where the structure and the inherent properties of the graph may be preserved. Many embedding algorithms are based on random walks and executing such algorithms against partitioned graphs poses new challenges, such as preserving network neighborhood of nodes.

This thesis introduces a novel methods of knowledge graph partitioning that considers a set of queries (workload), as well as an adaptive partitioning method for large knowledge graphs, which adapts the partitioning in response to changes in the query workload. The resulting partitioning aims to reduce the number of distributed joins necessary to answer the queries in the workload and

thus improves the workload performance. Critical features identified in the query workload and the knowledge graph are used to cluster the queries and then partition the graph based on the clusters. Workload queries are rewritten to account for the graph partitioning. The thesis also introduces a novel method for embedding partitioned knowledge graphs. After partitioning, the method performs (partial) random walks in each partition, in parallel. Complete walks are then assembled before an embedding is created.

Our evaluation results demonstrate the performance improvements in the query workload processing time and improvements after dynamically adapting the partitioning of the knowledge graph. Another evaluation demonstrates that the performance of knowledge graph embedding is improved after partitioning the graph, and that the quality of the generated embedding is similar to the one produced on the complete, unpartitioned graph.

INDEX WORDS: *Knowledge Graph, Graph Partitioning, RDF, RDF Partitioning, Workload Aware Partitioning, Query Workload, Adaptive Partitioning, Node Embedding, Graph Representation Learning, Feature Learning, link prediction, ontology, OWL, machine learning, random walk, knowledge discovery*.

QUERYING AND DATA MINING ON PARTITIONED KNOWLEDGE GRAPHS

by

AMITABH PRIYADARSHI

MS, Bharti Vidyapeeth University, India, 2009

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial

Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2023

© 2023

Amitabh Priyadarshi

All Rights Reserved

QUERYING AND DATA MINING ON PARTITIONED KNOWLEDGE GRAPHS

by

AMITABH PRIYADARSHI

Major Professor:	Krzysztof J. Kochut
Committee:	John A. Miller
	Ismailcem Budak Arpinar

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
May 2023

DEDICATION

To my loving parents. Khalil Gibran once said that parents are like a bow, and children are like arrows. The more the bow bends and stretches, the farther the arrow flies. I fly, not because I am special, but because they stretched for me.

Also, I want to dedicate my work to my family, especially my elder brother, who always encourage me to go extra mile for my dreams. Thank you, my sister in-law, and my nephew, for helping me emotionally to get through this.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my major professor Dr. Krzysztof J. Kochut for his invaluable contribution, unparalleled support, and profound belief in my work. I would also like to extend my deepest gratitude to my committee member Dr. John A. Miller and Dr. Ismailcem Budak Arpinar for their suggestions and invaluable insight into my research. Thanks, should also goes to all my lab mates for their support and creating friendly and productive environment. Also, I’d like to acknowledge all my colleague from Office of Institutional Research for their support.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND 5

2.1 Resource Description Framework 5

2.2 Resource Description Framework Schema 7

2.3 Web Ontology Language 8

2.4 SPARQL Protocol and RDF Query Language 9

2.5 Standard RDF Triple Store with Associated SPARQL Processor 14

3 WawPart: WORKLOAD AWARE PARTITIONING OF KNOWLEDGE GRAPH 16

3.1 Introduction 17

3.2 Related Work 20

3.3 Workload Aware Knowledge Graph Partitioning 22

3.4 Implementation 35

3.5 Result and Evaluation 36

3.6 Conclusions and Future Work 40

4 AWAPart: ADAPTIVE WORKLOAD AWARE PARTITIONING OF KNOWLEDGE GRAPH 41

4.1 Introduction 42

4.2 Related Work 44

4.3 Workload Aware Adaptive Knowledge Graph Partitioning 46

4.4 Implementation 53

4.5 Experiments 55

4.6 Conclusion and Future Work 61

5 PartKG2Vec: EMBEDDING OF PARTITIONED KNOWLEDGE GRAPH 62

5.1 Introduction 63

5.2 Related Work 65

5.3 Partitioned Knowledge Graph Embedding 69

5.4 Implementation 72

5.5 Evaluation 74

5.6 Conclusion and Future Work 80

6 CONCLUSIONS AND FUTURE WORK 82

REFERENCES 84

APPENDICES

A. ORIGINAL 14 LUBM QUERY 87

B. 14 LUBM FEDERATED QUERY BASED ON WAWPART 96

C. 14 LUBM FEDERATED QUERY BASED ON RANDOM PARTITION 100

D. ORIGINAL 12 BSBM QUERY 104

E. 12 BSBM FEDERATED QUERY BASED ON WAWPART 110

F. 12 BSBM FEDERATED QUERY BASED ON RANDOM PARTITION 116

G. EXTRA 14 LUBM QUERY 123

H. EXTRA 8 BSBM QUERY 128

LIST OF TABLES

Page

Table 2.1: Example of a Turtle file describing an RDF....7

Table 2.2: Example of a Turtle file describing an RDF Schema....7

Table 2.3: Example of a Turtle file describing OWL Schema....9

Table 2.4: Sparql Query Structure....10

Table 2.5: Simple RDF Data in TTL format....11

Table 2.6: Simple Sparql Query....11

Table 2.7: Query results....12

Table 2.8: Data exposed through SPARQL endpoint: <addressbook1>....12

Table 2.9: Data exposed through SPARQL endpoint: <addressbook2>....12

Table 2.10: Federated query....13

Table 2.11: Federated query results....13

Table 2.12: RDF QUAD....14

Table 3.1: Listing of Features in Queries 7 and 9....29

Table 3.2: Partitioned graph based on Partitioning threshold.....31

Table 3.3: Original and Federated query of LUBM 2nd Query....34

Table 4.1: Original and Federated Query of Lubm 9th Query....43

Table 4.2: Listing of Features in Queries 2 and 8....49

Table 4.3: HAC Algorithm....51

Table 4.4: Knowledge Graph Adaptive Partitioning Algorithm....52

LIST OF FIGURES

Page

Figure 2.1: Graphical representation of an RDF Triple....5

Figure 3.1: Graphical representation of an RDF Triple....17

Figure 3.2: Graphical representation of SPARQL query....18

Figure 3.3: Graphical representation of a Federated SPARQL query....18

Figure 3.4: Feature: 3 Predicate in RDF Triple patterns....25

Figure 3.5: Feature: 3 Predicate-Object in RDF Triple patterns....25

Figure 3.6: Feature: Subject-Subject Join at ?x in RDF Triple patterns....25

Figure 3.7: Feature: 2 Object-Subject Join at ?y and ?z in RDF Triple patterns....26

Figure 3.8: Feature: Object-Object Join at ?z in RDF Triple patterns....26

Figure 3.9: Distance between Q7 and Q9....28

Figure 3.10: HAC of all queries, Queries 7 and 9....28

Figure 3.11: Linkages: single, complete, and average....29

Figure 3.12: Partitioning threshold for partitioning Graph into k shards from HAC....31

Figure 3.13: WawPart System Architecture....35

Figure 3.14: LUBM 14 queries runtime in milliseconds....37

Figure 3.15: Runtime are color coded to easy comparison of time taken by Query in LUBM....37

Figure 3.16: BSBM 12 queries runtime in milliseconds....38

Figure 3.17: Runtime are color coded to easy comparison of time taken by Query in BSBM....39

Figure 3.18: LUBM 14 queries average runtime....39

Figure 3.19: BSBM 12 queries average runtime 39

Figure 4.1: Example of an RDF Triple 48

Figure 4.2: Distance between Q2 and Q8 48

Figure 4.3: HAC Dendrogram of LUBM's 14 Queries 50

Figure 4.4: AWAPart System Architecture 54

Figure 4.5: LUBM's 24 queries runtime in milliseconds 56

Figure 4.6: LUBM all 24-query average runtime in milliseconds 57

Figure 4.7: LUBM 10 new queries average runtime in milliseconds 57

Figure 4.8: Example 1 of Swapping of triples associated with properties between partitions 58

Figure 4.9: Example 2 of Swapping of triples associated with properties between partitions 59

Figure 4.10: LUBM all queries average runtime of Initial vs. Adaptive partition in millisecond.60

Figure 4.11: LUBM all query average runtime when frequency of Query1 is 50% of total workload a) Total runtime in minutes. b) Total runtime in milliseconds 60

Figure 5.1: Completion of walk from partial walks 71

Figure 5.2: PartKG2Vec pipeline 72

Figure 5.3: PartKG2Vec Architecture 73

Figure 5.4: PartKG2Vec (node2vec) runtime comparison with node2vec on Yago39K 75

Figure 5.5: PartKG2Vec (node2vec) runtime comparison with node2vec on NELL....................76

Figure 5.6: PartKG2Vec (Deepwalk) runtime compared to Deepwalk on Yago39K 77

Figure 5.7: PartKG2Vec (Deepwalk) runtime compared to Deepwalk on NELL.........................77

Figure 5.8: Average divergence scores of embeddings on Yago39K....................79

Figure 5.9: Average divergence scores of embeddings on NELL 80

CHAPTER 1

INTRODUCTION

In recent years, large-scale knowledge graphs with billions of nodes and edges have been created in various application areas. A Knowledge Graph (KG) is a representation of real-world entities that includes not only relevant information but also temporal and spatial relationships between entities. Storing such knowledge graphs in a centralized data store is often not feasible due to its large size, especially if the graph is to be stored in main memory for efficiency reasons. To address this problem, a knowledge graph can be partitioned and stored as shards and distributed among computing nodes. A partitioning of a graph into subsets intends to make a graph more manageable. Partitioning of large-scale knowledge graphs offers to make a distributed graph more flexible, efficient, and maintainable. The focus in this thesis is on the problems involved with querying and data mining on partitioned knowledge graphs. The presented solutions focus on the tradeoffs in graph partitioning techniques.

A knowledge graph is an essential tool for exploring and understanding the links between various entities in each domain or data set. However, it is difficult to effectively explore and utilize graphs when their sizes are large. In addition, it is equally difficult to efficiently search through large volumes of data. We propose particular graph partitioning methods, which address problems with typical tasks on these partitioned graphs, such as querying. These problems typically involve increased communication costs due to distributed joins involving edge cuts. To overcome this, a

suitable, optimized partitioning of a knowledge graph is required to reduce edge cuts, while still considering query processing efficiency.

This dissertation presents a novel framework for processing and querying large-scale knowledge graphs. It aims to reduce the number of distributed joins in knowledge graphs and improve query workload (a set of queries performed repeatedly) performance by creating a clustering optimizing the processing of the queries, which must be processed in a distributed fashion. Furthermore, such a partitioned graph needs to be continually re-partitioned to accommodate changes in the query workloads and optimize the average processing time for the workload. To solve this problem, we introduced a method to adapt a graph partitioning in response to changes in the query workload. The proposed method takes advantage of the query workload submitted by user to partition the knowledge graph and adaptively splits or merges partitions. The adaptive approach utilizes features including queries, features in the graphs and performance metrics to identify whether clusters should be merged or split. Queries are rewritten to account for the partitioning. The proposed algorithm for efficient partitioning of a candidate knowledge graph solution, given a set of constraints, such as worst-case number of cuts, attempts to maximize the average time reduction for join operations between all partitions, to render an optimal result for any given query workload. Then, based on the results from the previous section, we introduce a new optimization procedure to guide the query execution.

Representational learning/embedding of a knowledge graph is another popular operation on graphs, whose results can be used in various machine learning tasks. State-of the art embedding techniques are often unable to achieve scalability without losing accuracy and efficiency. Graph embedding algorithms convert a graph into a vector space, where the structure and the inherent properties of the graph are preserved. Each shard in a partitioned knowledge graph can be assigned

labels indicating its identity and its neighbor shards, as well as additional custom metadata, such as links (in/out edges and connecting shards) and topological constraints that serve as a priori information about its role within the context of other nodes in the same partition (or within other partitions). Executing representational learning algorithms against these fragmented sub-graphs poses new challenges, such as maximizing the likelihood of preserving network neighborhood of nodes. The optimal solution is determined by performing portions of the embedding task on the individual graph partitions and then merging them into one overall embedding to maximize the likelihood of preserving network neighborhood of nodes. In this thesis, we present a novel graph embedding algorithm for representational learning on partitioned knowledge graphs. Our method is a segmentation technique for knowledge graphs to partition them into subgraphs according to their semantic relationships to reduce embedding complexity. The representational learning method for embedding of partitioned knowledge graphs, which partitions the knowledge graph and executes learning algorithms on each partition separately and merges their outcomes to produce the overall embedding. To illustrate our method, we use modified versions of the Node2Vec and DeepWalk algorithms to capture the structural diversity of each node in the partitioned graph by clustering them based on geometric distance between them and their closest neighbor nodes.

In this dissertation, we also present a study how a specific partitioning can be used to improve query processing time, which is a fundamental issue in distributed problem solving. Our evaluation results demonstrate the performance improvement in workload processing time and improvement after dynamically adapting the partitioning of knowledge graph triples. Preliminary evaluation shows that the partitioned knowledge graph performs considerably better than the unpartitioned knowledge graph with respect to both the number of shards and their processing time for querying

and representational learning. Using a dynamic algorithm for adaptively partitioning, we demonstrate that dynamically adapting partitioning improves runtime performance and quality for embeddings. The representational learning runtime performance is improved after partitioning of knowledge graph against complete knowledge graph and the quality of the embedding is like that of an embedding produced on the complete, unpartitioned graph. We evaluate our method on several real-world applications and show that it achieves both better performance and higher quality embeddings compared with those obtained with traditional methods.

This thesis is organized as follows. Chapter 2 provides the background. Chapter 3 discusses the WawPart, a workload aware partitioning of knowledge graphs. Chapter 4 introduces AWAPart, an adaptive workload aware partitioning of knowledge graphs. Chapter 5 presents PartKG2Vec, an embedding of partitioned knowledge graphs. Chapter 6 contains the conclusions and presents some future work.

CHAPTER 2

BACKGROUND

This chapter provides an explanation of the terminology used throughout the dissertation and covers the core concepts related to the Knowledge Graphs and some other terms that will clarify the necessary concepts used later.

2.1 Resource Description Framework (RDF)

The Resource Description Framework [1] enables users to represent machine-readable information on the Web. RDF is a language for the representation of resources. A resource is anything that can be located using a Uniform Resource Identifier (URI) on the web. RDF helps to create links between different resources which represent some semantic relationships among resources, thus, creating a graph. The basic building block of RDF is a statement, which has three parts and therefore it is called a triple. A triple consists of the following parts:

Statement: University of Georgia is a university

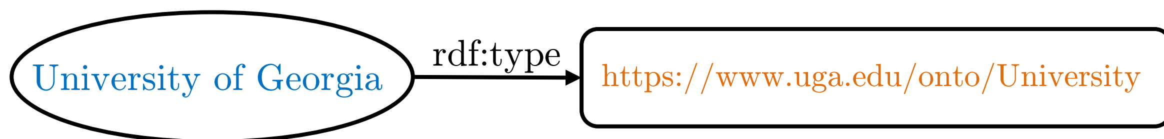


Figure. 2.1: Graphical representation of an RDF Triple

- **Subject**: a resource being described. e.g. University of Georgia
- **Predicate**: a property of that resource. e.g. rdf:type
- **Object**: a value of that property, it can be another resource or a literal. e.g. www.uga.edu

In Figure 2.1, a basic RDF triple is shown. The subject and object are represented as an oval and rectangle, respectively. The predicate is represented by an edge of the graph. The information in the triple describes UGA, an entity, that it is a university. Here, UGA and University are URIs and ”rdf:type” in a predicate representing the entity’s type.

Many RDF triples form a graph, where subject nodes are connected to object nodes via directed edges, representing predicates, or semantic dependencies. As stated before, each triple expresses some specific piece of knowledge. The intention is that an RDF graph can be processed by computers to obtain useful information represented in the graph.

Furthermore, an RDF graph allows to interconnect different resources by explicitly defining that a resource in a dataset is similar to some different resource in another dataset. This introduces a Web of data that can be read by machines. Today, hundreds of interconnected datasets provide a vast amount of knowledge in many different domains.

Different parts of the RDF graph can be defined as:

• URIs - which are used to reference resources unambiguously

• Literals – which are used to describe data values with no clear identity like ”smith@mail.com”.

• Blank Nodes - which represent resources for which a URI or a literal is not given.

By defining their own URIs, users can add information about any resource in the RDF graph. Thus, an RDF model allows users to integrate the data sources at different location on the Web. RDF triples collectively form a directed labeled graph. Following are the most popular formats in which RDF data can be serialized using:

• RDF/XML - the official XML serialization of RDF,

• N-Triples - a text format focusing on simple parsing,

• Turtle - a text formal focusing on human readability, and

• Notation 3 - a text format with advance features beyond RDF.

```
@prefix ex: <http://www.example.org/> .
ex:john      rdf:type       ex:Man .
ex:john      ex:livesIn     "New-York" .
ex:livesIn   rdf:type       rdf:Property .
```

Table. 2.1: Example of a Turtle file describing an RDF

```
@prefix ex: <http://www.example.org/> .
ex:john        rdf:type           ex:Man .
ex:Man         rdfs:subClassOf    ex:Human .
ex:john        ex:livesIn         "New-York" .
ex:livesIn     rdf:type           rdf:Property .
# After reasoning
ex:john        rdf:type           ex:Human .
```

Table. 2.2: Example of a Turtle file describing an RDF Schema

2.2 Resource Description Framework Schema (RDFS)

In Semantic Web development, a vocabulary [2] [3] is a set of terms stored in a standard format that people can reuse. A vocabulary of classes and property names has its own namespace to make it easier to use it using other sets of data. RDF Schema [4] is a description language for vocabularies which adds some extra knowledge to RDF. RDF Schema is the set of triples that are used to describe other triples in the data. Table 2.2 shows a few of the triples

from the RDF Schema vocabulary description of the Dublin Core vocabulary [5] in the form of .ttl (Turtle) file. Prefixes are included at the top of the file to assign a shorter names for namespaces of URIs. This makes it easier to include URIs in triples. rdf:type is part of an RDF vocabulary. It describes that the creator is an instance of the class Property. Similarly, RDF Schema provides an ability to describe the data in a more expressive way. For example, rdfs:label lets the user to add a label to describe the subject. Moreover, RDF Schema defines Classes and Properties that create a taxonomy for arranging the RDF data [5] using the subClassOf and subPropertyOf schema properties. Resources in the RDF data set can be grouped together into Classes. A member of a class is called an instance of that class. It is possible for a resource to be an instance of more than one class. Classes are also resources, and therefore, they can be defined by properties to add more information about the class itself. For example, the domain and range are the two properties that can be defined for a class. RDF Schema, however, provides a limited amount of reasoning. OWL overcomes this limitation.

2.3 Web Ontology Language (OWL)

W3C's Web Ontology Language [6] is used to describe complex knowledge about the entities on the web. It provides a way to represent the relationships between a group of things. Knowledge expressed by OWL can be exploited by computers. The documents represented in OWL are known as Ontologies [2], which are a form of knowledge representation. It is a way to represent all the entities that exist and the relationships between the different entities. In addition, a variety of restriction can be introduced, as well. Ontologies are a formal definition of vocabularies that allow the users to define complex structures and new relationships between vocabulary terms and between members of the classes that were defined in the Ontology. Ontologies are also collections

of RDF triples. Information about the resources mentioned in these triples or relationships between different resources is described using the OWL vocabulary. OWL builds on RDFS. Therefore, it has a vocabulary that is richer and more expressive than RDFS itself. Some of OWL's most notable features are its ability to provide a way to state transitive, inverse, and symmetrical properties. Table 2.3 shows some triples that use OWL's vocabulary. The terms defined in the vocabulary can be accessed using the prefix owl.

```
@prefix ex: <http://www.example.org/> .
ex:livesIn    rdf:type           owl:DatatypeProperty .
ex:Human      rdf:type           owl:Class .
ex:Man        rdf:type           owl:Class .
ex:Man        rdfs:subClassOf    ex:Human .
ex:John       rdf:type           ex:Man .
ex:John       rdf:type           owl:NamedIndividual .
```

Table 2.3: Example of a Turtle file describing OWL Schema

There are lots of RDF dataset available on the web. Users can extract meaningful information and make inferences from this data using the RDF query language SPARQL.

2.4 SPARQL Protocol and RDF Query Language (SPARQL)

As discussed in the previous sections, using W3Cs standards, users are able to represent structured data on the Web that can be exploited by the computers. The OWL vocabulary allows the users to create ontologies, which represent resources and relationships between various resources. SPARQL Protocol and RDF Query Language (SPARQL) [7, 8] is a W3C standard that is used for querying the data represented as RDF graphs for exploring any unknown relationships between resources. SPARQL query RDF data in the same way as SQL query data represented by the

Relation Databases. A SPARQL SELECT query comprises of different components as shown in the Table 2.4 in the order they appear in a SPARQL query.

```
#Prefix declaration
PREFIX foo: <http://www.example.org/resources/>
#Dayabase definition
FROM …
#Result clause
SELECT…
#Query pattern
WHERE{
    …
}
#Query modifier
ORDER BY
```

Table 2.4: Sparql Query Structure

Functionality of each component is described below:

• Prefix deceleration - for abbreviating URIs

• Dataset definition - stating what RDF graph(s) are being queried

• Result clause - identifying what information to return from a query

• Query pattern - specifying what to query in each dataset

• Query modifiers - allow to rearrange the query results.

A SPARQL query is executed against an RDF dataset that consists of RDF graphs. A SPARQL endpoint [9] accepts SPARQL queries that returns the result via HTTP. The results of a SPARQL query can be returned or rendered in various formats: XML, JSON, RDF, HTML.

SPARQL attempts to match a triple pattern in an RDF graph. Triple patterns are similar to a triple, except that any of the parts of a triple, i.e., subject, predicate or object, can be a variable. Matching of a triple pattern is called binding.

```
@prefix ab: <http://Whitepages.com/ns/addressbook#> .
ab:Richard   ab:homeTel   "(229) 276-5135" .
ab:Richard   ab:email     "richard49@hotmail.com" .
ab:Cindy     ab:homeTel   "(245) 646-5488" .
ab:Cindy     ab:email     "cindym@gmail.com" .
ab:Craig     ab:homeTel   "(119) 966-1505" .
ab:Craig     ab:email     "craigellis@yahoo.com" .
ab:Craig     ab:email     "c.ellis@usairwaysgroup.com" .
```

Table 2.5: Simple RDF Data in TTL format.

Consider data represented by the .ttl file shown in Table 2.5 is stored in an RDF graph. The data is available through a SPARQL endpoint. The query shown in Table 2.6 can be used to extract the information represented by a triple pattern. The RDF graph addressbook will be searched for all triples that have ab:craig as a subject and ab:email as a predicate. The answers will be bound to the variable ?craigEmail, which is an object.

```
@prefix ab: <http://Whitepages.com/ns/addressbook#> .

SELECT ?craigEmail
FROM <http://Whitepages.com/ns/addressbook>
WHERE{
    ab:Craig     ab:email     ?craigEmail .
}
```

Table 2.6: Simple Sparql Query

The query is searching for all the email address available for the URI ab:craig. In the end, values bound to ?craigEmail are returned. The result is displaced in the Table 2.7 .

?craigEmail
craigellis@yahoo.com
c.ellis@usairwaysgroup.com

Table 2.7: Query results

It is possible to query more than one RDF graph exposed through different SPARQL endpoints in a single query. This can be achieved by using federated SPARQL queries.

2.4.1 Federated SPARQL Query

Users are increasingly publishing large amounts of RDF data on the Web. This data can be queried through open SPARQL endpoints. Federated Queries [8] allow the users to combine solutions from different RDF datasets. The keyword SERVICE is used within the WHERE clause that directs a portion of a query towards a particular SPARQL endpoint. A federated query

```
@prefix ab: <http://Whitepages.com/ns/addressbook#> .
ab:Richard   ab:homeTel   “(229) 276-5135” .
ab:Richard   ab:email     “richard49@hotmail.com” .
```

Table 2.8: Data exposed through SPARQL endpoint: <http://Whitepages.com/.../addressbook1>

```
@prefix ab: <http://Whitepages.com/ns/addressbook#> .
ab:Cindy     ab:homeTel   “(245) 646-5488” .
ab:Cindy     ab:email     “cindym@gmail.com” .
ab:Craig     ab:homeTel   “(119) 966-1505” .
ab:Craig     ab:email     “craigellis@yahoo.com” .
ab:Craig     ab:email     “c.ellis@usairwaysgroup.com” .
```

Table 2.9: Data exposed through SPARQL endpoint: <http://Whitepages.com/.../addressbook2>

processor merges the results coming from the various SPARQL endpoints. Consider the data represented by .ttl file in Figure 2.6. Currently, the whole data is stored in a single RDF dataset and therefore a normal query is able to obtain the results. Suppose this data is divided in two different RDF datasets, as shown in the Table 2.9 and 2.8. To obtain

```
@prefix ab: <http://Whitepages.com/ns/addressbook#> .
SELECT ?email
FROM <http://Whitepages.com/.../addressbook2>
WHERE{
    ab:Craig  ab:email  ?email .
    UNION
    SERVICE <http://Whitepages.com/.../addressbook1>{ab:Richard ab:email ?email.}
}
```

Table 2.10: Federated query

the email addresses of the subjects *ab:craig* and *ab:richard*, a federated query has to be executed. The query is shown in the Table 2.10, which is executed against the SPARQL endpoint <http://Whitepages.../addressbook2> . The query search for the triple pattern *ab:craig ab:email ?email* locally. Whereas SERVICE keyword prompts the federated query processor to search for the triple pattern *ab:richard ab:email ?email* in the RDF dataset exposed through the SPARQL endpoint <http://Whitepages.../addressbook1>. Finally, the results are combined and displayed, as shown in Table 2.11.

?email
craigellis@yahoo.com
c.ellis@usairwaysgroup.com
richard49@hotmail.com

Table 2.11: Federated query results

There are many available RDF triple stores that provide the functionality of storing and querying the RDF data, such as Redland [10], Sesame, Jena [11, 12], Virtuoso, and many others. In this thesis, Virtuoso [13, 14] is used as the system to store and query RDF triples.

```
Create table DB.DBA.RDF_QUAD(
   G IRI_ID,
   S IRI_ID,
   P IRI_ID,
   O any,
   primary key(G, S, P, O)
);
Create bitmap index RDF_QUAD_OGPS on DB.DBA.RDF_QUAD (O, G, P, S);
```

Table 2.12: RDF QUAD

2.5 Standard RDF Triple Store with Associated SPARQL Processor

According to the documentation, “OpenLink Virtuoso is a revolutionary, next generation, high-performance virtual database engine for the Distributed Computing Age”. It provides support to store and query the RDF data. Virtuoso stores RDF triples in the form of Relational Database tables. One of the main tables of the default RDF storage system is shown in the Table 2.12. Each triple in RDF QUAD (triples here are represented as quadruples, including a graph URI, which indicates a graph to which the triple belongs to) is represented as one row in RDF triples. The table is called ”QUAD” as it stores triple with one extra column representing the graph to which that triple belongs to. Therefore, the columns represent the graph, subject, predicate, and the object. To answer the SPARQL queries, Virtuoso converts them to SQL queries and executes them against the tables, where the RDF triples are stored. Virtuoso also provides the support to run federated queries over multiple SPARQL endpoints. In Virtuoso, SPARQL and SQL share the same query

execution engine, query optimizer, and the cost model. For Federated Queries, the logic for optimizing the message flow between multiple endpoints on the Web is similar to the logic for message-optimization on a cluster. Virtuoso also provides an Interactive SQL (ISQL) which provides a faster way to perform some tasks, like bulk loading of RDF triples.

CHAPTER 3

WawPart: WORKLOAD-AWARE PARTITIONING OF KNOWLEDGE GRAPHS[*]

Abstract. Large-scale datasets in the form of knowledge graphs are often used in numerous domains, today. A knowledge graph's size often exceeds the capacity of a single computer system, especially if the graph must be stored in main memory. To overcome this, knowledge graphs can be partitioned into multiple sub-graphs and distributed as shards among many computing nodes. However, performance of many common tasks performed on graphs, such as querying, suffers, as a result. This is due to distributed joins mandated by graph edges crossing (cutting) the partitions. In this paper, we propose a method of knowledge graph partitioning that considers a set of queries (workload). The resulting partitioning aims to reduces the number of distributed joins and improve the workload performance. Critical features identified in the query workload and the knowledge graph are used to cluster the queries and then partition the graph. Queries are rewritten to account for the graph partitioning. Our evaluation results demonstrate the performance improvement in workload processing time.

Keywords: Knowledge Graph, Graph Partitioning, Query Workload

---

[*] *Reproduced with permission from Springer Nature & IEA/AIE,*
A. Priyadarshi and K. J. Kochut, "WawPart: Workload-Aware Partitioning of Knowledge Graphs," Cham, 2021: Springer International Publishing, in Advances and Trends in Artificial Intelligence. Artificial Intelligence Practices, pp. 383-395

3.1 Introduction

In today's world, the data or information is interconnected and form large knowledge graphs or information networks. Such knowledge graphs are often used to represent data in social networking systems, shopping and movie preferences, bioinformatics, and in other real-world systems. The availability of large-scale data, represented as knowledge graphs composed of millions or even billions of vertices and edges, requires large-scale graph processing systems. Such data is often too large to be stored at one place in a centralized data store and needs to be partitioned into multiple sub-graphs, often called shards, and transferred to multiple computing nodes in a distributed system. However, with the overall knowledge graph split into distributed shards, many typical graph-base tasks, such as query processing, suffer from network latency and other problems related to the original graph being partitioned. One of the techniques to improve query processing performance in such a system is to reduce the inter-node communication between graph processing systems. The graph partitioning can be an effective pre-processing technique to balance the runtime performance.

Statement: University of Georgia is a University

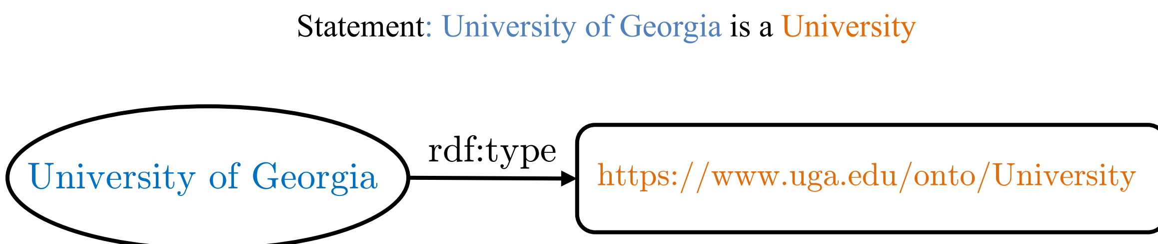


Figure. 3.1: Graphical representation of an RDF Triple

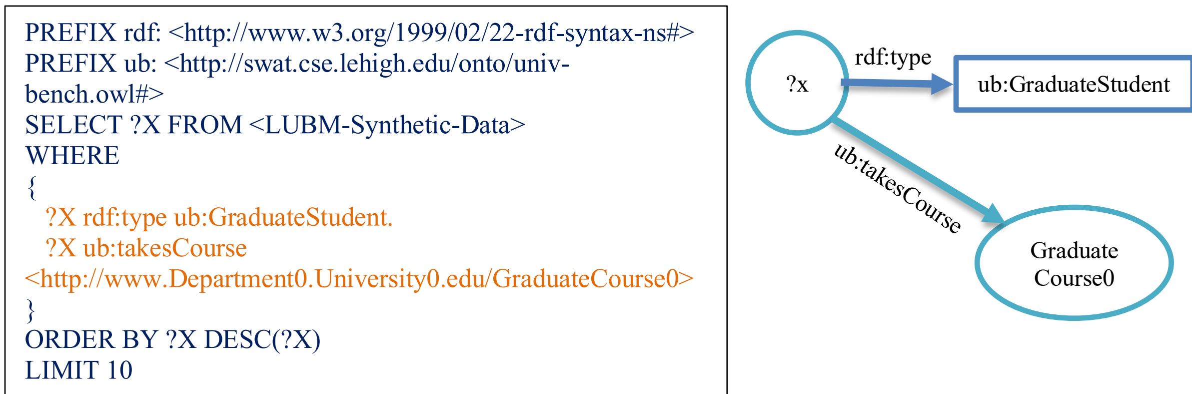


Figure. 3.2: Graphical representation of SPARQL query

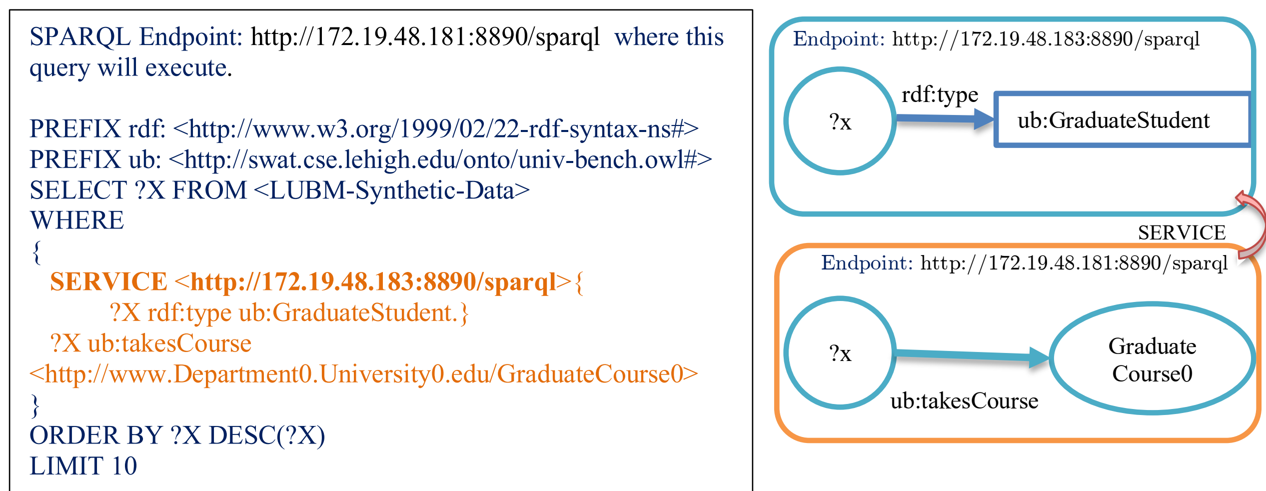


Figure. 3.3: Graphical representation of a Federated SPARQL query

In this paper, we will assume that a knowledge graph is a dataset represented using the Resource Description Framework (RDF) [7] [1]. An RDF knowledge graph is a set of RDF triples in a form of (s, p, o), where s is the subject, p predicate, and o the object, as shown in Figure 3.1. A predicate represents a semantic relationship connecting a subject and an object. Consequently, an RDF dataset is a directed graph, where nodes (vertices) and edges have types, often described using the Resource Description Framework Schema (RDFS). In many recent publications, a knowledge graph has often been defined as a Heterogeneous Information Network (HIN) [16], a form of a

directed graph, where nodes and edges have heterogeneous types. HINs are a bit more restrictive than RDF/RDFS in that if two edges are labeled by the same relationship type, their starting and ending nodes must have the same object types, respectively. RDF/RDFS does not have this restriction. Recently, graph databases, such as Neo4j [17], have also been used to represent and store knowledge graphs.

Knowledge graphs have been used for many different tasks, such as data mining, link (edge) prediction, and data classification, to mention a few. Query processing, intended to retrieve some data of interest, is one of the most common tasks. The SPARQL query language is the query language for RDF/RDFS datasets , while Cypher is an example graph query language used with the popular Neo4j graph database [17]. An example of Sparql query with graphical representation of query pattern in Figure 3.2 is shown. A critical part of a SPARQL query is a graph pattern composed of *triple patterns* involving variables. Without going into details here, a query processor matches the query pattern against the knowledge graph and reports matches as query solutions.

Typically, graph tasks become computationally intensive, when considering graphs with millions, even billions of vertices. A large knowledge graph can easily overload the memory and processing capacity even on relatively large servers, especially for in-memory knowledge graph systems. Partitioning a knowledge graph into smaller components (sub-graphs) and distributing the partitions, called shards, across a cluster of servers may enable handling of very large graphs and potentially offer performance improvements in executing different tasks. Federated sparql query is used in this paper to gather the query result which are distributed across different partitions. Figure 3.3 is the example to a federated sparql query. The Federated sparql query uses SERVICE keyword for achieving distributed joins.

Given a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges and a number $k > 1$, a *graph partitioning of* G is a subdivision of vertices of G into *subsets* of vertices $V_1, \ldots, V_k$ that partition the set V. A *balance constraint* requires that all partition blocks are equal, or close, in size. In addition, a common objective function is to minimize the *total number of cuts*, i.e., edges crossing (cutting) partition boundaries.

This chapter is outlined as follows. Section 3.2 provides an overview of related work. Section 3.3 discuss about the partitioning method. Section 3.4 is about the Architecture and section 3.5 discuss about the experiments, and section 3.6 concludes the paper.

3.2 Related Work

It is generally expected that partitioning a large-scale graph decreases query processing efficiency. However, this decrease can be moderated if the partitioning is sensitive to a query workload used daily and optimized to minimize the inter-partition communication needed by the workload. In this section, we discuss research results related to graph partitioning and their impact on querying. The graph partitioning is an NP-complete [18] problem. To alleviate this problem, many practical solutions have been developed, including spectral partitioning methods [19] and geometric partitioning methods [20]. The multilevel method was introduced to the graph partitioning by Barnard and Simon [21] and then improved by Hendrickson and Leland [22]. The multilevel method consists of three main phases: coarsening, initial partitioning, and uncoarsening. A partitioning approach of Karypis et al [23] uses recursive multilevel bisection method for bisection of a graph to obtain a k-partition on the coarsest level.

ParMETIS [24] and LogGP [25] are state-of-the-art parallel graph processing systems. Also, LogGP is a graph partitioning system that records, analyzes, and reuses the historical statistical information to generate a hyper-graph and uses a novel hyper-graph streaming partitioning

approach to generate a better initial streaming graph partition. Pregel [26] is used for graphs with billions of vertices and is built upon MapReduce [27], and its open-source Apache Giraph [28], PEGASUS [29] and GraphLab [30] used as a default partitioner of vertices. Sedge [31] implements a similar technique as Pregel, which is also a vertex-centric processing model for SPARQL query execution implemented on top of this model. These methods, however, do not take into account a set of queries, called a workload, executed against the partitioned graph. This requires distributed query processing which penalizes workload processing performance.

DREAM [32], WARP [33], PARTOUT [34], AdPart [35], and WISE [36] are workload-aware, distributed RDF systems. DREAM [32] partitions only SPARQL queries into subgraph patterns and not the RDF dataset. The RDF dataset is replicated among nodes. It follows a master-slave architecture, where each node uses RDF-3X [37] on its assigned data for statistical estimation and query evaluation. WARP [33] uses the underlying METIS system to assign each vertex of the RDF graph to a partition. Triples are then assigned to partitions, which are stored at dedicated hosts in a triple store (RDF-3X). WARP determines a query's center node and radius based on an n-hop distance. If the query is within n-hops, WARP sends the query to all partitions to be executed in parallel. A complex query is decomposed into several sub-queries and executed in parallel and then the results are combined. PARTOUT [34] extracts representative triple patterns from a query workload by applying normalization and anonymization by replacing infrequent URIs and literals with variables. Frequent URIs (above a frequency threshold) are normalized. PARTOUT uses an adapted version of RDF-3X as a triple store for their n hosts. AdPart [35] is an in-memory RDF system that re-partitions the RDF data incrementally. It uses hash partitioning that avoids the cost associated with initial partitioning of the dataset. Each worker stores its local set of triples in an in-memory data structure. AdPart provides an ability to monitor and index workloads in the form

of hierarchical heat maps. It introduces Incremental Re-Distribution (IRD) technique for data portions that are accessed by hot patterns. IRD is a combination of hash-partitioning and k-hop replication, guided by a query workload. WISE [36] is a runtime-adaptive workload-aware partitioning framework for large scale knowledge graphs. A partitioning can be incrementally adjusted by exchanging triples, based on changes in the workload. SPARQL queries are stored in a Query Span structure with their frequencies. To reduce communication overhead, the system redistributes frequent query patterns among workers. A cost model which maximizes the migration the gain while preserving the balanced partition is used for migration of triples.

Our query workload-aware knowledge graph partitioning method, called WawPart, which extracts critical features from the query workload as well as from the dataset. These features are used to establish distance among queries in a form of distance matrix and then cluster similar queries together using hierarchical agglomerative clustering. Subgraphs (partitions) associated with these features are then created from the knowledge graph data and distributed as shards in a computing cluster.

The five closely related systems discussed in the previous paragraph rely on specialized RDF/SPARQL processing systems. In contrast, these systems, ours does not rely on a specialized data store implementation and uses an *off-the-shelf* knowledge graph storage and query processing system (Virtuoso) and relies on standard SPARQL queries for distributed processing. We believe that it is an important advantage.

3.3 Workload-Aware Knowledge Graph Partitioning

Knowledge graph partitioning created by WawPart attempts to optimize query workload processing time. Critical features of both the workload queries and the knowledge graph are extracted and analyzed. Subsequently, the queries are clustered based on their similarity and the

knowledge graph is then partitioned based on the clustering. Original queries are re-written into federated SPARQL queries since partitions are distributed among computing nodes. Typically, query processing times are increased, due to the distributed joins, as connected triples may be stored in partitions distributed to different nodes (such distributed partitions are usually called *shards*.) In this paper we focus on the effects of workload-aware knowledge graph partitioning, given a query workload. Here, we do not consider the changes in the workload and necessary modifications to the partitioning.

3.3.1 Query and Knowledge Graph Feature Extraction

Searching for similarities among query graph patterns is computationally expensive, as finding overlaps among query graph patterns is similar to finding isomorphic subgraphs. To help with these problems, we identify and extract critical features of graph patterns in a workload. Similarly, we identify critical features in the knowledge graph (later, we refer to it as the dataset, as well). These features are used to analyze the similarity among triples in queries. We identify the following features in the workload query graph patterns and in the knowledge graph:

- *Predicate (P)* feature represents all triples that share a given predicate (edge label). In Figure 3.4: there is 3 Predicate features in RDF Triple patterns.
- *Predicate-Object (PO)* represents all triples that share a given predicate *and* an object value. This feature is useful in analyzing similarity of some queries. In Figure 3.5: there is 3 Predicate-Object features in RDF Triple patterns.

Other features may be useful in query analysis, as well. These include:

- *Subject-Subject (SS)* feature represents triples that share the same subject, i.e., edges sharing the same source vertex. This feature is used for analyzing queries involving

multiple predicates/edges with the same subject (often called star shape patterns). In Figure 3.6: there is Subject-Subject Join at *?x* in RDF Triple patterns.

- *Object-Subject (OS)* represents triples where one triple's object (destination vertex) is another triple's subject (source vertex). Such data representation is beneficial when a query includes pairs of triples with connection based on object-subject (sometimes called an elbow join). In Figure 3.7: there is 2 **Object-Subject Join** at *?y* and *?z* in RDF Triple patterns.
- *Object-Object (OO)* feature represents triples that share the same object (target). In Figure. 3.8: there is **Object-Object Join** at ?z in RDF Triple patterns.

Triples identified by the SS, OS, and OO features represent joins on connecting vertices, which are shared entities (RDF resources) in the knowledge graph. Typically, query patterns also involve joins based on variables shared between triple patterns. All features extracted from the workload queries and from the knowledge graph are stored as metadata. We use them to drive the knowledge graph partitioning. To extract the features from these queries, we use a query analyzer, which analyzes the entire workload. Our query analyzer has been tailored for SPARQL queries, but it can be easily modified for other graph pattern-based query languages and knowledge graph representations, such as the Cypher language used in the Neo4j graph database [17].

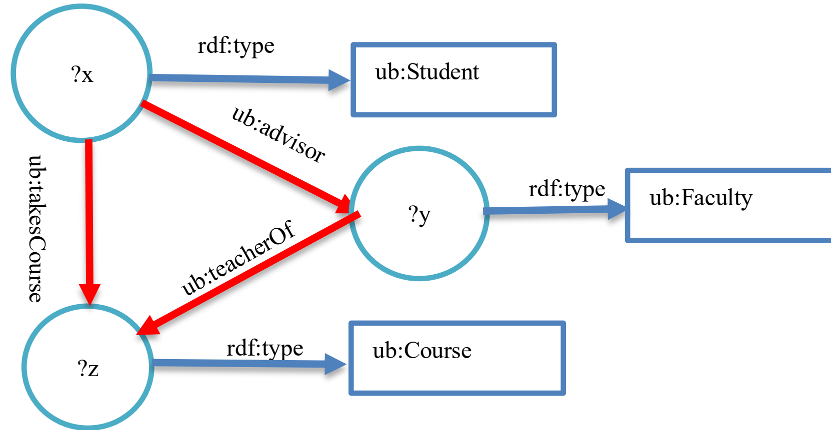


Figure. 3.4: Feature: 3 **Predicate** in RDF Triple patterns

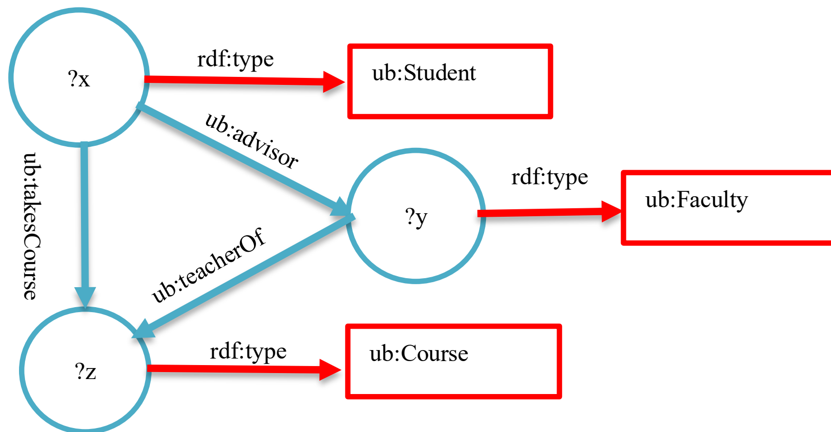


Figure. 3.5: Feature: 3 **Predicate-Object** in RDF Triple patterns

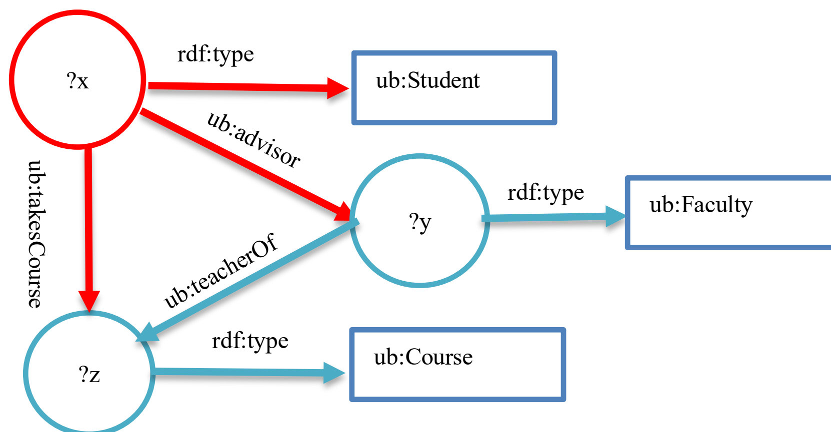


Figure. 3.6: Feature: **Subject-Subject Join** at *?x* in RDF Triple patterns

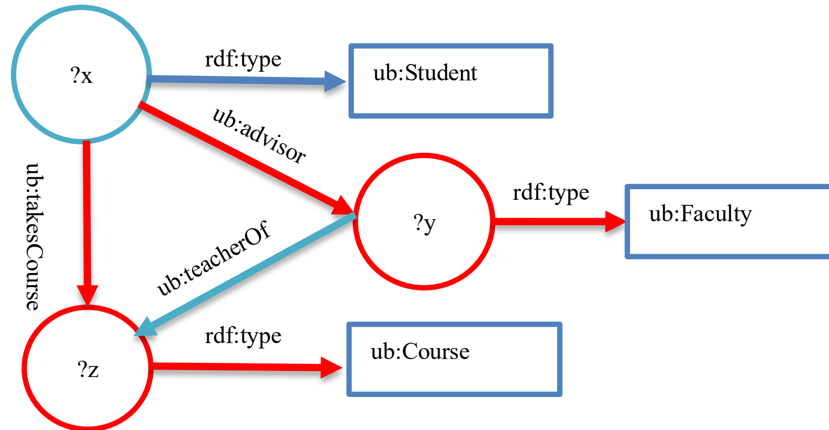


Figure. 3.7: Feature: 2 **Object-Subject Join** at *?y* and *?z* in RDF Triple patterns

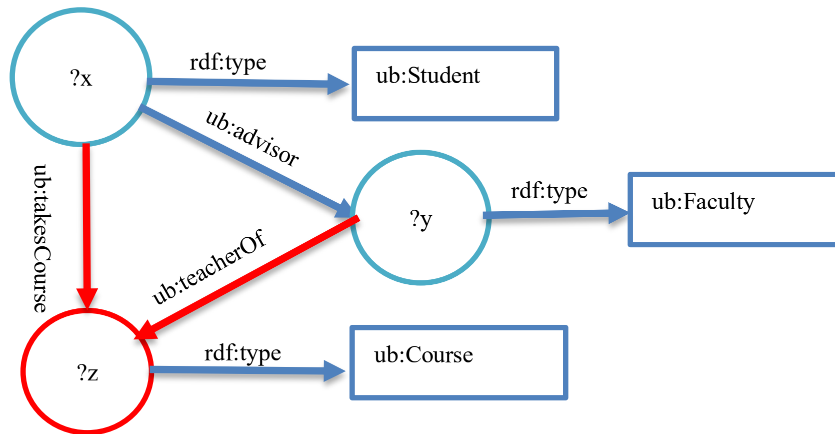


Figure. 3.8: Feature: **Object-Object Join** at ?z in RDF Triple patterns

In order to speed up handling of the features in the knowledge graph, we created indices on all triples in the knowledge graph, based on their subjects, predicates and objects. We used Apache Lucene API [38, 39] to create the indices and to quickly materialize the necessary features in the knowledge graph. For example, it is possible to materialize the *Predicate (P)* feature and find all triples using a given predicate or materialize the *Predicate-Object (PO)* and find triples sharing a

given predicate *and* object. Selected triples can then be easily assigned to their intended partitions, as needed.

3.3.2 Query Workload Clustering and Knowledge Graph Partitioning

We measure similarity between queries based on their features, as explained above. A distance matrix is often used as the basis for many data mining tasks, such as multi-dimensional scaling, hierarchical clustering, and others. We use Jaccard similarity to construct the distance matrix for the graph patterns in the workload queries. The Jaccard similarity of sets A and B is the ratio of the intersection of set A and B to the union of set A and B, or $J_{sim}(A,B) = |A \cap B| / |A \cup B|$, while the Jaccard distance between sets A and B is given by $1 - J_{sim}(A,B)$. Intuitively, if graph patterns in two queries are nearly identical, based on their features, the distance between them is zero or close to zero. We use a similarity matrix to represent the similarity of queries in the workload and use the matrix to split the workload queries into subsets of similar queries. The details of the computation are skipped here. As an example, consider 2 queries in Table 3.1 Query 7 has 4 features: (2 *PO* features: *rdf:type→ ub:Student, rdf:type→ ub:Course* and 2 *P* features: *ub:takesCourse, ub:teacherOf*) while query 9 has 6 features (3 *PO* features: *rdf:type→ub:Student, rdf:type→ ub:Course, rdf:type→ ub:Faculty* and 3 *P* features: *ub:takesCourse, ub:teacherOf, ub:advisor*). Consequently, the distance between these queries is $(1 - J_{sim}) = 1 - | Q7 \cap Q9 | / | Q7 \cup Q9 | = 0.33$. Hierarchical agglomerative clustering of all queries of LUBM shown in Figure 3.10 and the distance between query 7 and 9 is being calculated and it’s shown in red circle in the figure.

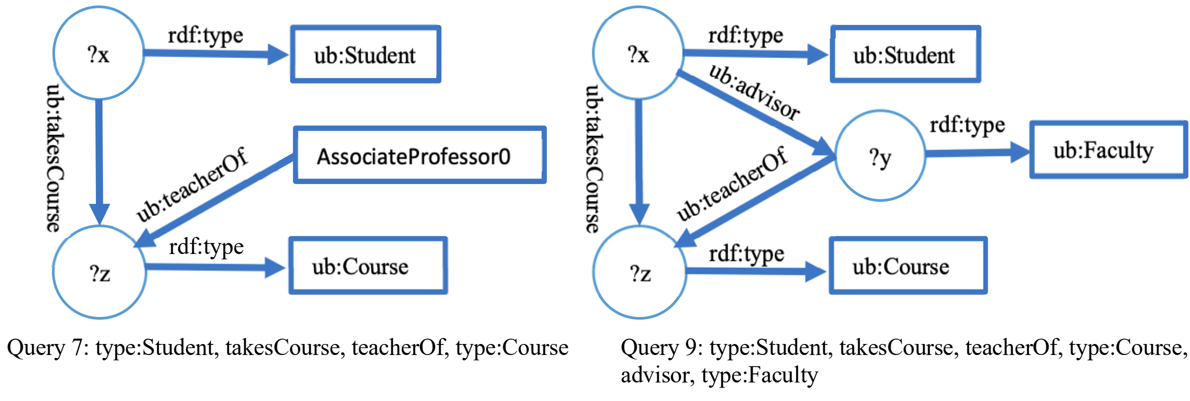


Figure. 3.9: Distance b/w Q7 and Q9 is 1- J_{sim} 1- |Q7∩Q9| / |Q7∪Q9|=(1- 4/6) = 0.33

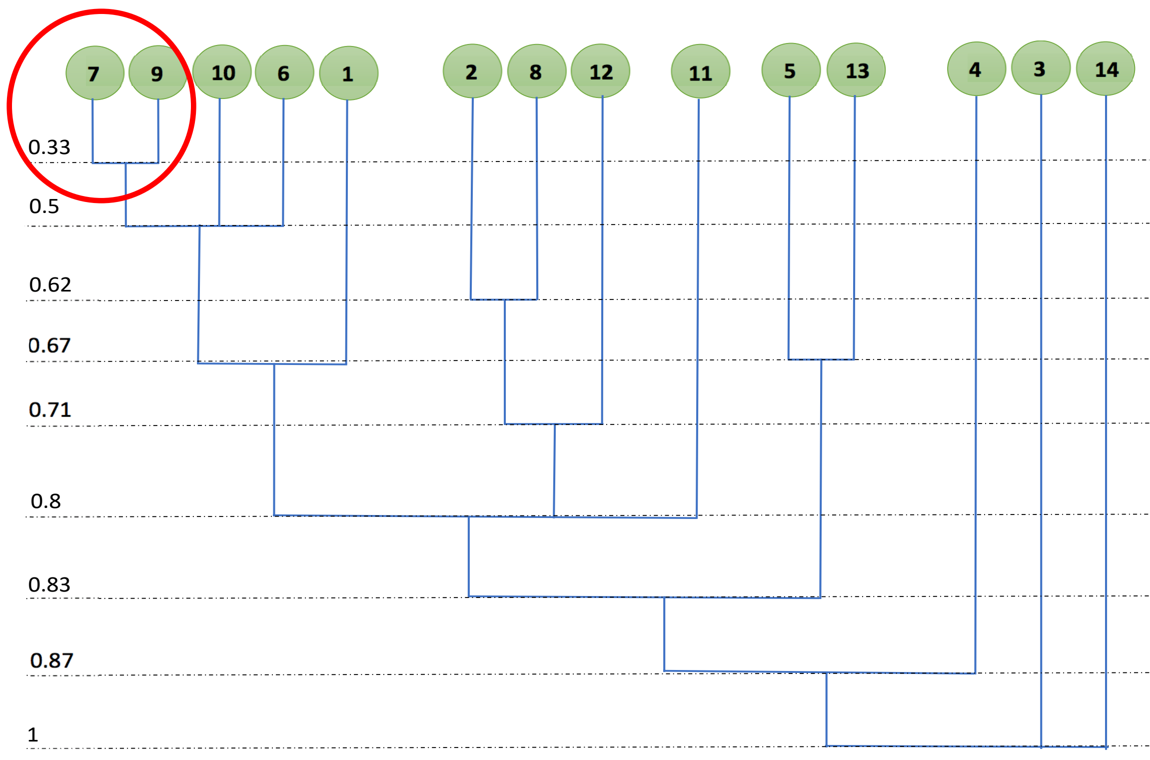


Figure. 3.10: HAC of all queries, Queries 7 and 9 are marked in red circle

- **Query 7 has 4 features:**

2 *PO* features:

rdf:type→ ub:Student

rdf:type→ ub:Course

2 *P* features:

ub:takesCourse

ub:teacherOf

- **Query 9 has 6 features:**

3 *PO* features:

rdf:type→ ub:Student

rdf:type→ ub:Course

rdf:type→ ub:Faculty

3 *P* features:

ub:takesCourse

ub:teacherOf

ub:advisor

The distance between these queries is $(1- J_{sim}) = 1- |Q7 \cap Q9| / |Q7 \cup Q9| = 1- 4/6= 0.33$

Table. 3.1: Listing of Features in Queries 7 and 9

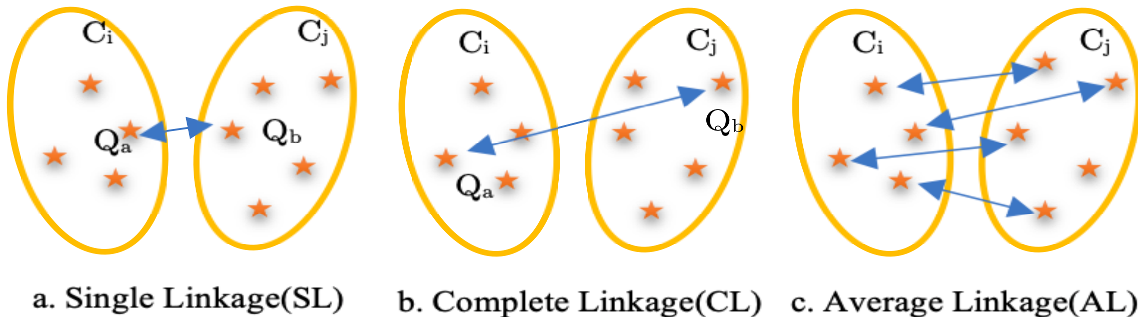


Figure. 3.11: Linkages **a)** $SL(C_i,C_j) = \min(D(Q_a,Q_b))$, **b)** $CL(C_i,C_j) = \max(D(Q_a,Q_b))$ and,

c) $AL(C_i , C_j) = \frac{1}{n_{Ci}n_{Cj}} \sum_{a=1}^{n_{Ci}} \sum_{b=1}^{n_{Cj}} D(Q_a, Q_b)$

Algorithm 1. Hierarchical Agglomerative Clustering of Query Workload	
Input	Feature Distance Matrix ***D*** of workload Query.
Output	HAC Dendrogram I
1	Assign for each D[n][n] into C[m] where $C = c_1, c_2, \ldots, c_m$
2	**while** C.size > 1 **do**
3	**for** i = 1 **to** C.size **do**
4	**if** $(c_a,c_b) = min\ d(c_a,c_b)$ in C **then** *// distance function* ***$d(c_1,c_2)$***
5	delete c_a and c_b from C
6	add *min* $d(c_a,c_b)$ in C
7	assign **I** = (old, c_ac_b, , *min* $d(c_a,c_b)$) *//(oldGroup, newGroup, distance)*
8	recalculate proximity matrix using (SL/CL/AL) **P = modifyD(** c_a ,c_b ,*min* $d(c_a,c_b)$**)** *//(oldGroup, newGroup, distance)*
9	for each P , c_m = P[i][j]
10	Update $C = c_1, c_2, \ldots, c_m$
11	Output **I**

For cluster analysis of the features in queries, we used the hierarchical agglomerative clustering (HAC), which is a method of creating a hierarchy of clusters in a bottom-up fashion (Algorithm 1). The decision as to which features should be kept together is established by HAC based on the measure of similarity between queries and selection of a linkage method. The cluster grouping is done according to the shortest distance among all pairwise distances between queries. Once the two most similar queries are grouped together, the distance matrix is recalculated. The first similarity measure is provided by the initial distance matrix. Recalculation of the distance matrix is based on the choice of the linkage from single, complete, or average. Figure 3.11 shows linkages in which a single linkage is the proximity between two nearest neighbors where a complete linkage is the proximity between the farthest neighbor. These steps are repeated until there is only a single

cluster left. We ran HAC using a single linkage against queries to obtain a dendrogram which is then used to partition the dataset. This method of partitioning reduces the inter-partition communication and reduces distributed joins in federated queries.

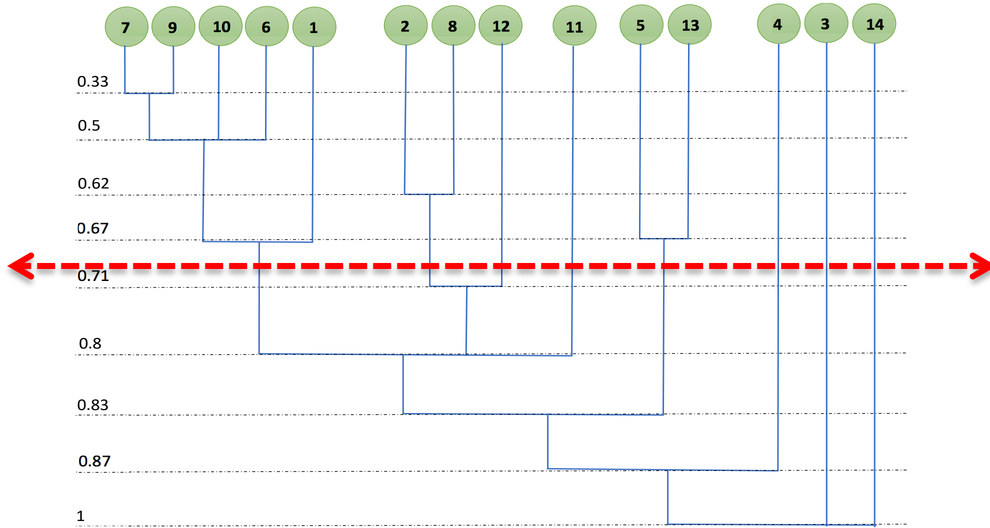


Figure. 3.12: Partitioning threshold for partitioning Graph into *k* shards from HAC

Distance : 0.67				
	A	B	C	D
Query Set	7,9,10,6,1	8,2	5,13	3,4,11,12,14
Features	GraduateStudent	GraduateStudent	memberOf	Publication
	takesCourse	University	Person	publicationAuthor
	Student	Department	hasAlumnus	Professor
	Course	memberOf		worksFor
	teacherOf	subOrganizationOf		name
	Faculty	undergraduateDegreeFrom		emailAddress
	advisor	emailAddress		telephone
	memberOf	Student		subOrganizationOf
	subOrganizationOf			ResearchGroup
				Department
				UndergraduateStudent
				Chair

Table. 3.2: Partitioned graph based on Partitioning threshold; All Red colored predicates are distributed joins for other partitions. Each partition shows query sets.

The partitioning algorithm (Algorithm 2) takes a dendrogram created for a given workload and outputs a partitioning optimized for the workload. For example, we used HAC with a single linkage on the LUBM queries to produce a dendrogram shown in Figure 3.12. A statistics module of the algorithm calculates the score for each replicated feature in every partition. These scores are based on 1) features which move together or are tightly connected with a replicated feature, 2) the size of the replicated feature and its peers, 3) dependencies of other queries on these features (replicated and their peer features) and 4) the number of distributed joins, when involved replicated features is on another partition. This score helps the algorithm to determine where to place the replicated features. The balancing module of the algorithm uses these features and also features that are not involved in any workload but present in the dataset to balance the partitions.

Algorithm 2. Knowledge Graph Partitioning Algorithm

Input	HAC Dendrogram **I**, Features $\mathbf{F_G}$
Output	Partition metadata **P**
1	Create Feature set **g** based on **I** at similarity distance **d**
2	**Statistics(g, $\mathbf{F_Q}$)** *//All Features $\mathbf{F_G}$ = (Queries features $\mathbf{F_Q}$+ Unused features $\mathbf{F_X}$)*
3	Find $\mathbf{F_R}$ replicated features in **g**.
4	Find Query distributed joins of replicated features $\mathbf{D_{QR}}$
5	Find stats $\mathbf{S_R}$ ∀ $\mathbf{F_R}$
6	Find p, q and c for C and T.
7	$\mathbf{S_R} = \sum (p_c w_1 + q_c w_2 + s_c w_3) + (p_t w_4 + q_t w_5 + s_t w_6)$
	#**p(**Peer Features**)**, **q(**Features in query**)** and **s(**Feature data size), w(weights), C(shards) and T(dataset).
8	**score for each $\mathbf{F_R}$ = ($\mathbf{D_{QR}}$ * w_7)+ $\mathbf{S_R}$**
9	**Balance_Partition(score, g , $\mathbf{F_G}$)**
10	Remove all $\mathbf{F_R}$ from sets of **g** with lower **scores** for each $\mathbf{F_R}$
11	Assign **data** associated to features set **g** into **P**
12	**Proximity_Query()**
13	Find proximity $\mathbf{F_{prox}}$ = proximity of $\mathbf{F}_{Unclustered}$ with $\mathbf{F}_{Clustered}$. // $\mathbf{F_Q = F_U + F_C}$.
14	Assign max($\mathbf{F_{prox}}$) in cluster $\mathbf{P_i}$ where its neighbor features are.
15	Assign $\mathbf{F_X} = \mathbf{F_X}$ + remaining $\mathbf{F_U}$
16	**while** F_X not empty **do**
17	$\mathbf{P_{min}}$ = Find min($\mathbf{P_i}$) by size of shard
18	$\mathbf{F_{max}}$ = Find max(**F** in $\mathbf{F_X}$) by size/count of triples
19	Assign $\mathbf{F_{max}}$ into $\mathbf{P_{min}}$
20	Output **P**

Queries are rewritten as federated SPARQL queries [40] to accommodate the distributed shards. The overall objective is to limit the number of distributed joins in federated queries. The query rewriter computes a cost-effective query execution plan by analyzing the metadata, converts the query into a federated SPARQL query and sends it to the most suitable processing node, where the query is executed. The SERVICE keyword followed by a SPARQL endpoint directs a part of the query for processing by SPARQL endpoint on remote shard. Table 3.3 shows an example of an original and a rewritten, (federated) query. If all data needed for a query is present in the same shard, the query is not rewritten. If the needed triples are in different shards, we need a federated query to obtain the required data from different shards, to produce the final result.

Original Query

```
SELECT ?X ?Y ?Z FROM <lubm>
WHERE{
       ?X rdf:type ub:GraduateStudent .
       ?Y rdf:type ub:University .
       ?Z rdf:type ub:Department .
       ?X ub:memberOf ?Z .
       ?Z ub:subOrganizationOf ?Y .
       ?X ub:undergraduateDegreeFrom ?Y
}
```

Federated Query

```
SELECT ?X ?Y ?Z FROM <lubm>
WHERE{
    ?X rdf:type ub:GraduateStudent .
    ?Y rdf:type ub:University .
    ?Z rdf:type ub:Department .
    SERVICE <http://172.19.48.185:8890/sparql> {?X ub:memberOf ?Z .}
    SERVICE <http://172.19.48.185:8890/sparql> {?Z ub:subOrganizationOf ?Y .}
    SERVICE <http://172.19.48.183:8890/sparql> {?X ub:undergraduateDegreeFrom ?Y}
}
```

Table 3.3. Original and Federated query of LUBM 2nd Query.

3.4 Implementation

We have implemented a prototype system which stores an RDF knowledge graph by partitioning it into shards and distributing them among computing nodes. Our system is deployed on cluster with a Master node and separate triple stores (Virtuoso [41]) are on shared-nothing nodes. The master node is responsible for querying and getting results along with tracking execution time. Each node with a triple store is called a Processing Node, as shown in the Figure 3.13 In the context of distributed RDF stores, the triples of the knowledge graph are partitioned into shards and assigned to different processing nodes with the help of the Partition Manager. The Query Rewriter/Processor (QRP) rewrites the incoming query into a federated query, according to the current location of features. The new federated query is executed on the specific shard (partition) with a maximum number of features to minimize the need for distributed joins. The shard where the query is executed is called Primary Processing Node (PPN) for that specific federated query. The system returns the results of that query to the user.

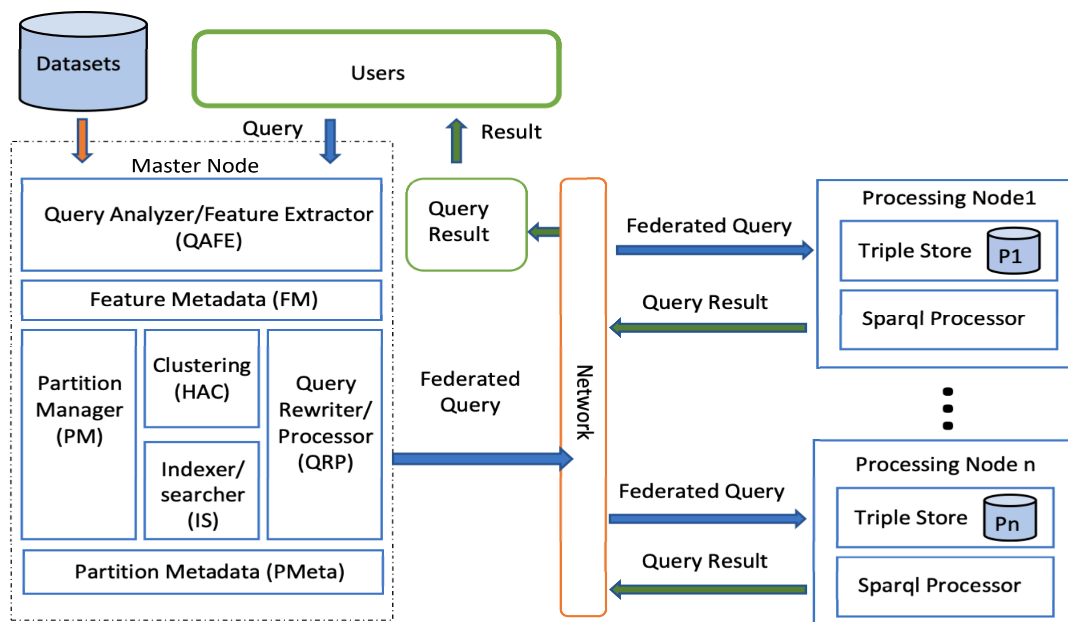


Figure. 3.13. System Architecture

In order to evaluate our knowledge graph partitioning method based on a query workload we used two synthetic datasets and associated data generation: Lehigh University Benchmark (LUBM) [42] and Berlin SPARQL Benchmark (BSBM) [43]. Both datasets were created for the purpose of testing and benchmarking Semantic Web triple store systems and their SPARQL query processing. LUBM includes a data generator, which creates synthetic OWL datasets with basic data about universities, and a set of 14 SPARQL queries for evaluation purposes. BSBM represents data about e-commerce, in which vendors offer products to customers and customers review those products. BSBM provides a dataset generator and a set of 12 SPARQL queries.

We used a cluster of Intel i5 based systems running Linux Ubuntu 18.04.4 LTS 64-bit operating system. We used relatively small machines to observe the effects of partitioning on manageable sizes of datasets. Each computer node had an instance of Virtuoso Open Source 7.2, which we used as a triple store and query processor for the triple shards. The knowledge graph partitioning systems and the experiments were coded in the Java programming language with the use of Apache Jena framework [11].

3.5 Results and Evaluation

Experiment shows the improvements of our workload-aware partitioning method over random partitions (complete sets of all triples with the same predicate were randomly assigned to partitions). For this experiment we have chosen LUBM dataset of 10 universities with 1,563,927 triples and BSBM dataset of 1000 products with 374,911 triples. The experiment shows the workload-aware partitioning has a significant improvement in the query processing time over a random partition.

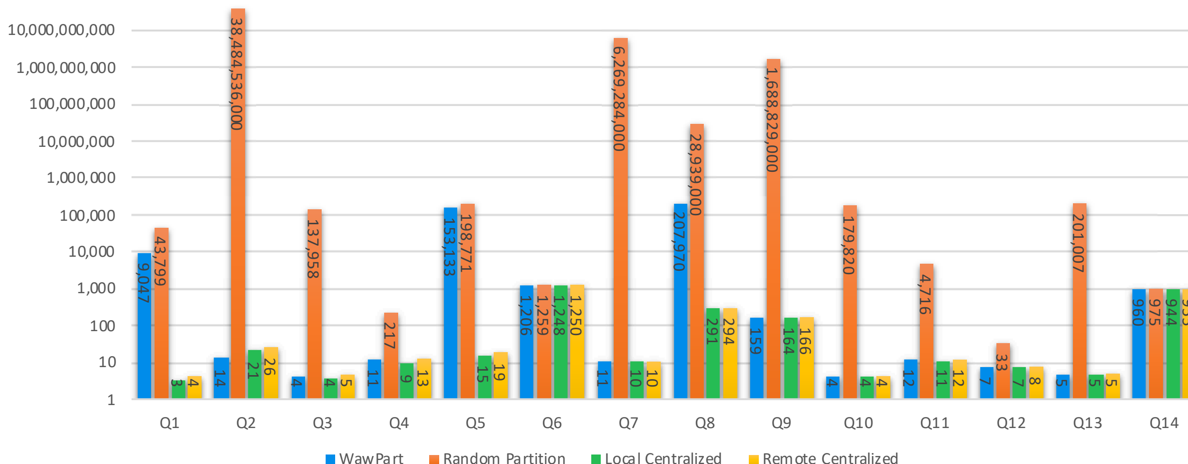


Figure. 3.14. LUBM 14 queries runtime in milliseconds.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
WawPart	9	14	4	11	153	1206	11	207	159	4	12	7	5	960
Random Part	43	445	138	217	198	1260	72	8	469	180	4	33	210	975
millisecond	second	minutes	hour	days										

Figure. 3.15: Runtime are color coded to easy comparison of time taken by Query in LUBM.

Our partitioning algorithm created shards of the LUBM dataset, which were then distributed among the nodes in our test cluster. As an example of balancing the shard sizes, WawPart splits the LUBM's 1,564k triples into three shards of 481k, 481k and 600k triples, which is within -8% to +15% of the exact average shard size. Random Partition has shards with 521k triples each. This experiment uses two distributed and two centralized datasets and demonstrates the improvement of query workload runtime performance on a distributed system. It also shows the comparison against the baseline query workload runtime. For distributed triple datasets, we used WawPart's and randomly partitioned datasets, marked as *Random Partition*. To compute network latency, we used difference in runtime between centralized datasets, one marked as *Local Centralized,* and

another marked as *Remote Centralized*. Figure 3.14 shows the runtime of all 14 LUBM queries in milliseconds. For 12 twelve queries (out of 14), performance of WawPart exceeded the *Random Partition* For two queries, 6 and 14, performance was the same because both queries include only a single triple pattern. In fact, performance of all queries in WawPart is very close to *Local Centralized*, which is a centralized dataset. Figure 3.18 shows the average runtime of all queries, and it demonstrates a dramatic improvement of WawPart's (26 second versus roughly 38 days *on Random Partition*). The average runtime for centralized datasets 0.2 seconds. For the BSBM dataset, Figure 3.16 shows the query runtime for all 12 queries. Figure 3.19 shows the average runtime of all 12 queries, where we can see the improvement of WawPart against the *Random Partition*. The distributed random partition average runtime is 1 hour 49 minutes and WawPart average runtime is only 16.7 milliseconds. The base average runtime is around 12.9 milliseconds for *Local All* in the centralized dataset. Performance of WawPart (distributed) is almost always very close to the centralized *Local All* partitioning.

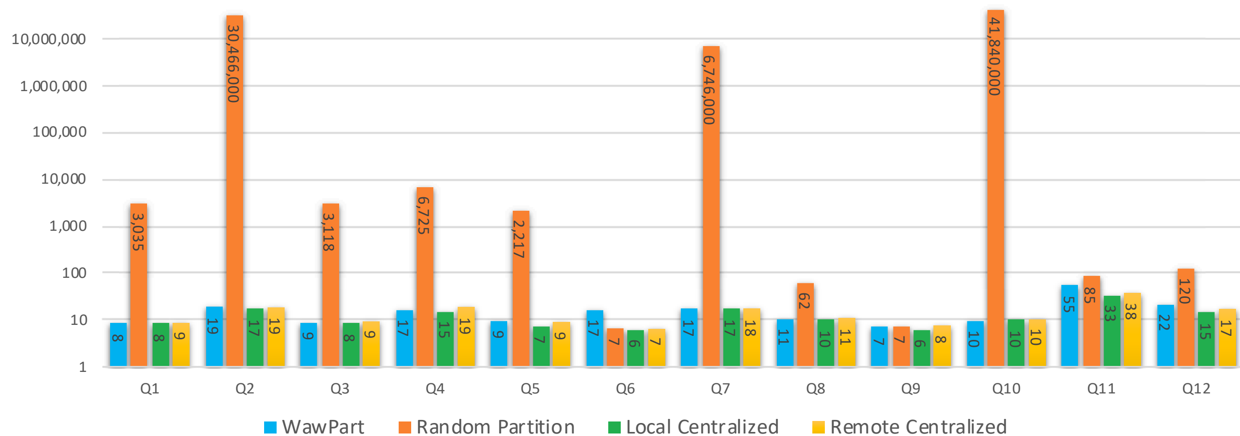


Figure 3.16. BSBM 12 queries runtime in milliseconds.

	1	2	3	4	5	6	7	8	9	10	11	12
WawPart	8	19	9	17	9	17	17	11	7	10	55	22
Random Part	3	8	3	6	2	7	2	62	7	11	85	120
millisecond	second	minutes	hour	days								

Figure. 3.17: Runtime are color coded to easy comparison of time taken by Query in BSBM

Our experiments demonstrate that partitioning the dataset without workload awareness significantly decreases the query workload performance since distributed joins are expensive. Partitioning the knowledge graph takin into account a given query workload leads to significant performance gains. In contrast these systems, ours does not rely on a specialized data.

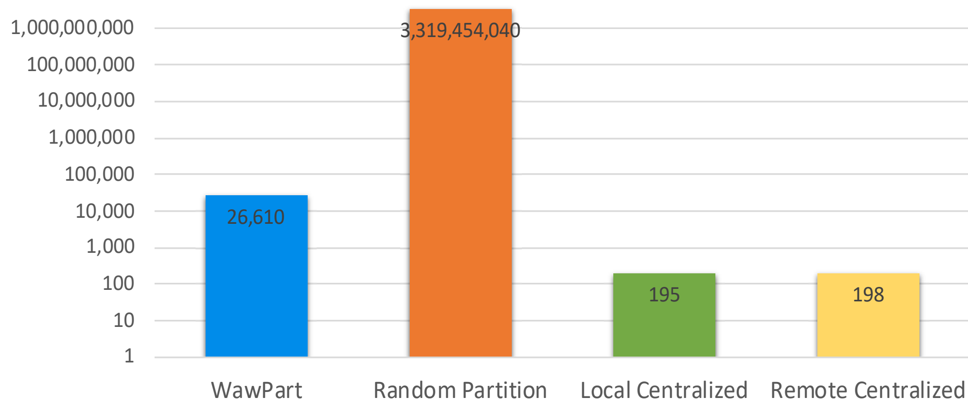


Figure 3.18. LUBM 14 queries average runtime

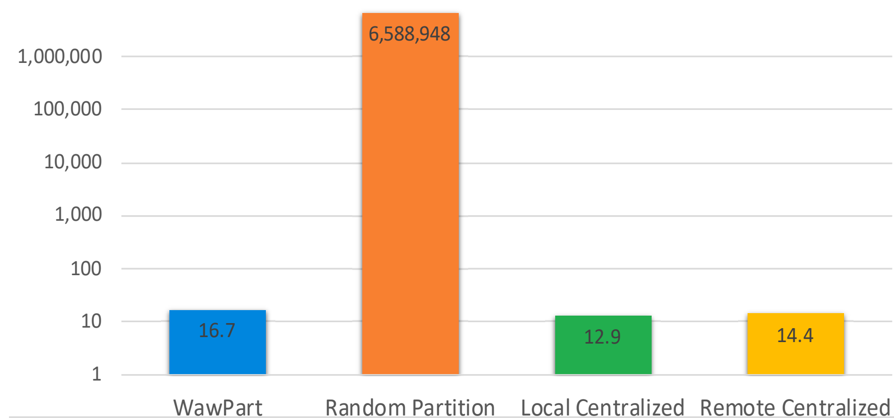


Figure 3.19. BSBM 12 queries average runtime

3.6 Conclusions and Future Work

In this paper, we have proposed a WawPart system, which is a knowledge graph partitioning and query processing system. It partitions the RDF dataset based on the query workload and aims to reduce the number of distributed joins during query execution, to improve the workload run-time. WawPart requires no replication of the data. For the evaluation of the system, we used LUBM and BSBM, two synthetic RDF/SPARQL benchmarks. Our experiments investigated the effect of workload-aware knowledge graph partitioning. The results showed a significant increase in the performance for the workload queries in comparison to a non-workload aware partitioning.

WawPart can easily be modified to function with other forms of knowledge graph representation such as graph databases, for example, Neo4j. In the near future, we plan to test WawPart on a different knowledge graph system and investigate the impact of workload-aware partitioning on different types of knowledge graphs.

Furthermore, we are planning to extend WawPart to adaptive re-partitioning due the changes in the workload. We plan to investigate the adaptability of partitioning due to (1) the changes in the frequency of queries in the workload (currently, we assume that all queries are executed with the same frequency) and (2) the changes in the composition of the workload (queries may be eliminated and new queries may be added to the workload).

CHAPTER 4

AWAPart: ADAPTIVE WORKLOAD-AWARE PARTITIONING OF KNOWLEDGE GRAPHS [#]

Abstract—Large-scale knowledge graphs are increasingly common in many domains. Their large sizes often exceed the limits of systems storing the graphs in a centralized data store, especially if placed in main memory. To overcome this, large knowledge graphs need to be partitioned into multiple sub-graphs and placed in nodes in a distributed system. But querying these fragmented sub-graphs poses new challenges, such as increased communication costs, due to distributed joins involving cut edges. To combat these problems, a good partitioning should reduce the edge cuts while considering a given query workload. However, a partitioned graph needs to be continually re-partitioned to accommodate changes in the query workload and maintain a good average processing time. In this paper, an adaptive partitioning method for large-scale knowledge graphs is introduced, which adapts the partitioning in response to changes in the query workload. Our evaluation demonstrates that the performance of processing time for queries is improved after dynamically adapting the partitioning of knowledge graph triples.

Keywords:- knowledge graphs; adaptive graph partition; query workload.

[#] *Reproduced with permission from ThinkMind, IARIA & SEMAPRO*
A. Priyadarshi and K. J. Kochut, "AWAPart: Adaptive Workload-Aware Partitioning Knowledge Graphs," in *SEMAPRO 2021, The Fifteenth International Conference on Advances in Semantic Processing*, Barcelona, Spain, 2021: Thinkmind Digital Library, 2021, pp. 12-17.

4.1 Introduction

The availability of large-scale knowledge graphs, which often holds hundreds of millions of vertices and edges, such as the ones used in social network systems or in other real-world systems, requires large-scale graph processing. Often, these datasets are too large to be stored and processed in a centralized data store, especially if it is maintained in main memory. Instead, the knowledge graph often needs to be partitioned into multiple sub-graphs, called shards, and transferred to multiple nodes in a distributed system. However, these systems frequently suffer from network latency. One of the techniques to improve the query answering performance is to reduce the inter-process communication between graph processing subsystems. While graph partitioning may be an effective pre-processing technique to improve the runtime performance, the cost of frequent partitioning of the entire large-scale knowledge graph may be prohibitive. In this case, it would be advantageous to partition the graph only once, initially, and make only necessary partitioning adjustments, afterwards. For example, such adjustments are needed in case of changes to the query workload.

First, let us talk about graph partitioning. Given a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges and a number $k > 1$, a *graph partitioning of G* is a subdivision of vertices of G into *subsets* of vertices $V_1, ..., V_k$ that partition the set V. A *balance constraint* requires that all partition blocks are equal, or close, in size. In addition, a common objective function is to minimize the *total number of cuts*, i.e., edges crossing (cutting) partition boundaries.

Our knowledge graph dataset is in the form of Resource Description Framework (RDF) [1]. RDF enables the embed-ding of machine-readable information on the web. A resource can be represented using a URL on the web. The RDF statement which comprises of three parts called a triple, consists of (s, p, o) resource, property, and value of resource. RDF Schema [4] defines

Classes and Properties that create a taxonomy for arranging the RDF data. Web Ontology Language (OWL) [6] is a language to describe complex knowledge about the things and provides a way to represent the relationships between a group of things. The documents in OWL are known as Ontologies [2]. The RDF query language SPARQL is the W3C standard that is used for querying the data in RDF graphs for exploring relationships between resources. SPARQL tries to match a triple pattern in an RDF graph. A SPARQL endpoint accepts SPARQL queries that return the result via HTTP. The partitioned RDF graph can be accessed through different SPARQL endpoints in a single query using Federated SPARQL Query [8]. The SERVICE keyword is used to direct a portion of a query towards a particular SPARQL endpoint. In Table 1 there is an example of LUBM's [42] SPARQL query and its federated query. The federated query processor merges the results coming from the various SPARQL endpoints.

Original Query

```
SELECT ?X ?Y ?Z FROM lubm
WHERE{
     ?X rdf:type ub:Student.
     ?Y rdf:type ub:Faculty .
     ?Z rdf:type ub:Course .
     ?X ub:advisor ?Y .
     ?Y ub:teacherOf ?Y .
     ?X ub:takesCourse ?Z .
}
```

Federated Query

```
SELECT ?X ?Y ?Z FROM lubm
WHERE {
     ?X rdf:type ub:Student.
     SERVICE <Sparql endpoint> {?Y rdf:type ub:Faculty .}
     ?Z rdf:type ub:Course .
     SERVICE <Sparql endpoint> {?X ub:advisor ?Y .
     SERVICE <Sparql endpoint> {?Y ub:teacherOf ?Y .}
     ?X ub:takesCourse ?Z .
}
```

Table 4.1 Original and Federated Query of Lubm 9th Query

This paper is outlined as follows. Section 2 provides an overview of related work. Section 3 discusses the partitioning method. Section 4 is about the architecture and workflow of the system. Section 5 is dedicated for the experiments, and Section 6 concludes the paper.

4.2 Related Work

Usually, partitioning a large-scale graph decreases query processing efficiency. However, this decrease can be mitigated if the partitioning is adjusted to a query workload and tuned to reduce the workload demands for inter-partition communication. Related work on graph partitioning and its implication on query processing is addressed in this section.

The graph partitioning problem is NP-complete [18]. Many practical techniques have been developed to address this issue, including spectral partitioning methods [19] and geometric partitioning methods [20]. Barnard and Simon [21] proposed the multilevel method to graph partitioning, and Hendrickson and Leland [22] enhanced it. Coarsening, initial partitioning, and uncoarsening are the three basic phases of the multilevel technique. Karypis et al. [23] employ a recursive multilevel bisection method for graph bisection to generate a k-partition on the coarsest level in their partitioning approach.

Workload-aware, distributed RDF systems include DREAM [32], WARP [33], PARTOUT [34], AdPart [35], and WISE [36]. DREAM [32] only partitions SPARQL queries into subgraph patterns, not the entire RDF dataset. The RDF dataset is replicated among nodes. It is designed in a master-slave architecture, with each node using RDF-3X [37] on its assigned data for statistical estimation and query evaluation. WARP [33] assigns each vertex of the RDF graph to a partition using the underlying METIS system. The triples are subsequently assigned to partitions, which are then stored in a triple store on dedicated hosts (RDF-3X). WARP uses an n-hop distance to compute the query's center node and radius. If the query is within n-hops, WARP sends the query

to all partitions to be executed in parallel. A complex question is broken down into multiple sub-queries, which are then run in parallel, and the results are merged. PARTOUT [34] uses normalization and anonymization to extract representative triple patterns from a query workload by substituting infrequent URIs and literals with variables. Frequent URIs (above a frequency threshold) are normalized. PARTOUT uses an adapted version of RDF-3X as a triple store for their n hosts. AdPart [35] is an in-memory RDF system that incrementally re-partitions RDF data. In an in-memory data structure, each worker stores its local set of triples. AdPart provides an ability to monitor and index the workloads in the form of hierarchical heat maps. It introduces Incremental Re-Distribution (IRD), which is a query workload-guided combination of hash partitioning and k-hop replication. WISE [36] is a workload-aware, runtime-adaptive partitioning system for large-scale knowledge graphs. Based on changes in the workload, a partitioning can be modified incrementally by trading triples. The frequencies of SPARQL queries are kept in a Query Span structure. When migrating the triples, a cost model that maximizes the migration gain while preserving the balanced partition is applied.

AWAPart, presented in this paper, is a query-adaptive workload-aware knowledge graph partitioning algorithm that extracts features from both the query workload and the dataset. These features are utilized to create a distance matrix between queries and then cluster similar queries together using hierarchical agglomerative clustering. From the knowledge graph data, subgraphs (partitions) associated with these features are produced and distributed as shards in a computing cluster. The partitioning of the graph will be adjusted in response to changes in the workload, e.g., if some queries are replaced or their execution frequencies change. This is done by updating metadata with new features from new queries. These new features are being clustered again. Scoring helps the system to swap data associated with features from one shard to another. This

swapping is done to reduce the edge cuts and minimize query runtime. Importantly, unlike the related systems, ours does not rely on a specialized data store implementation and uses an *off-the-shelf* knowledge graph storage and query processing system (Virtuoso [14]) and relies on standard SPARQL queries for distributed processing.

4.3 Workload-Aware Adaptive Knowledge Graph Partitioning

As the workload changes over time, an optimized (current) graph partition eventually becomes inefficient for the modified workload. AWAPart's goal is to adapt an existing knowledge graph partitioning to changes in the query workload, to optimize the workload processing time. Critical features from the current and modified workloads are extracted and analyzed. Features of queries in the changed workload are clustered, based on the similarity measures. The features in the new and old clusters are compared and a new optimized partition is created. The system then dynamically adjusts the deployed partitioning (shards) by exchanging triples belonging to the modified features between shards in the cluster. However, the analysis of the workload and the resulting adjustment of the partitions (shards) is infrequent. It can be performed in the background, without interrupting the process of querying. Queries in the workload are re-written to form federated SPARQL queries for processing on the cluster. Adjusting the partitioning (shards) aims to limit the number of distributed joins (utilizing triples from different shards), which decreases workload processing time.

4.3.1 Query Feature Extraction

The query feature metadata maintains the information about the triple patterns, which is referred as features in this paper, present in a set of triples. This metadata is maintained for each shard to describe the current set of triples in the shard.

The following features are used to describe various triple patterns, which are identified for the purpose of query workload clustering.

- *Property (P):* This feature represents all triples which share a given predicate P (triple's property).
- *Property-Object (PO):* This feature represents all triples sharing the same predicate P and the object (triple's property and object).

Other feature types used for query analysis are:

- *Subject-Subject Join (SSJ):* Triples sharing the same subject.
- *Object-Object Join (OOJ):* Triples sharing the same object.
- *Object-Subject Join (OSJ):* Triples connected on an entity which is the object in one triple and the subject in the other (it is referred as an "elbow" join in this paper).

We created the QueryAnalyzer which extracts the above features from the queries and creates the feature metadata. This metadata represents the features, their frequencies, neighboring features, related data sizes and distributed joins in that query. This helps the system to optimize the partition by re-adjusting the partitioning based on the updated features. Currently, our QueryAnalyzer is built for the SPARQL query language, but it can be easily adapted to a different graph pattern-based query language, such as Cypher, used in the Neo4j [45] graph database.

Triples in the entire knowledge graph are indexed based on their subject, predicate and object, and the graph can be searched using any of them. For instance, it is easy to materialize the predicate feature and locate all triples with a given property P or any other triple pattern using a feature discussed above. For indexing the initial dataset of N-Triples, Apache Lucene API [38] is used to accelerate searching for triple features, while creating an initial partition [15] tailored to the initial query workload.

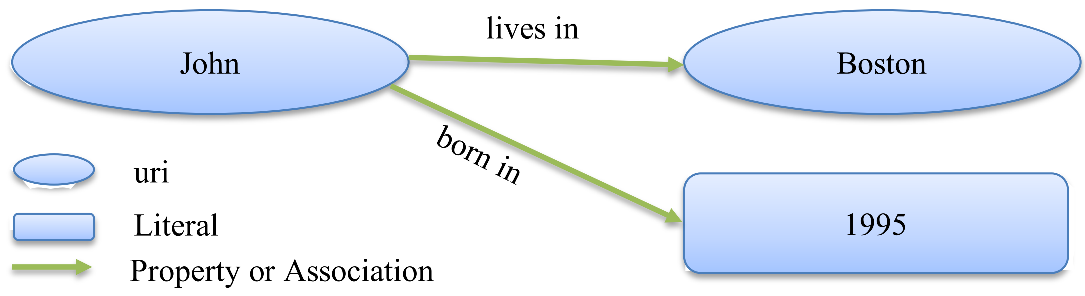


Figure. 4.1: Example of an RDF Triple

4.3.2 Query Workload Clustering and Knowledge Graph Adaptive Partitioning

The distance matrix is used as an input data for data mining, such as multi-dimensional scaling, hierarchical clustering, etc. To measure the similarity between queries in a workload, based on their features, Jaccard similarity is used which generates a distance matrix. Clustering uses this distance matrix. The Jaccard similarity of sets A and B is the ratio of the intersection of sets A and B to the union of sets A and B. $J_{SIM} = |A \cap B| / |A \cup B|$.

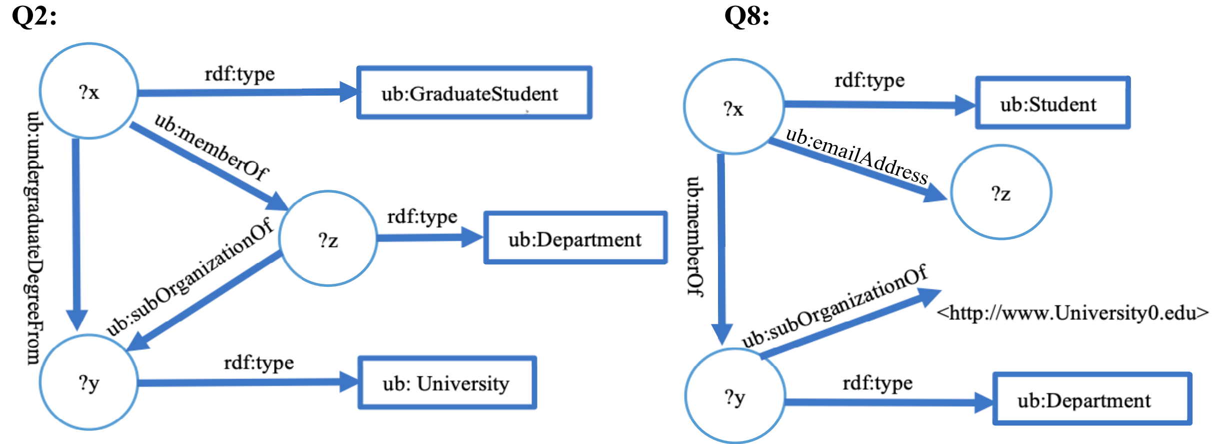


Figure 4.2 Distance between Q2 and Q8 is $1\text{-}J_{sim} = 1\text{-} (|Q2 \cap Q8|/|Q2 \cup Q8|) = (1\text{-}3/8) = 0.625$

- Query 2 has 6 features:

3 *PO* features:

rdf:type→ ub:GraduateStudent

rdf:type→ ub:Department

rdf:type→ ub:University

3 *P* features:

ub:memberOf

ub:subOrganizationOf

ub:underGraduateDegreeFrom

- Query 8 has 5 features:

2 *PO* features:

rdf:type→ ub:Student

rdf:type→ ub:Department

3 *P* features:

ub:memberOf

ub:subOrganizationOf

Table 4.2 Listing of Features in Queries 2 and 8

In Figure 4.2 and Table 4.2, query 2 has 6 features: (3 *PO* features: *rdf:type→ub:GraduateStudent, rdf:type→ub:Department, rdf:type→ub:University* and 3 *P* features: *ub:memberOf, ub:subOrganizationOf, ub:underGraduateDegreeFrom*) while query 8 has 5 features (2 *PO* features: *rdf:type→ub:Student, rdf:type→ub:Department,* and 3 *P* features: *ub:emailAddress, ub:subOrganizationOf, ub:memberOf*). The Jaccard similarity, which is the ratio of the intersection of both sets to the union of both sets, is 3/8. Now, the distance between two similar sets should be 0 and the Jaccard similarity of two identical sets returns 1. Therefore, the distance between queries Q2 and Q8 is $(1\text{-}J_{SIM}(Q2, Q8)) = 1 - 3/8 = 0.625$.

We used the Hierarchical agglomerative clustering (HAC) algorithm (Figure 4.3), which is a method of creating a hierarchy of clusters in a bottom-up fashion. The creation of clusters is based on the measure of similarity between clusters and the selection of linkage method. The shortest pairwise distance between queries determines the grouping. The distance matrix is recalculated once the two most similar clusters are being grouped together. Jaccard is used to create this distance matrix.

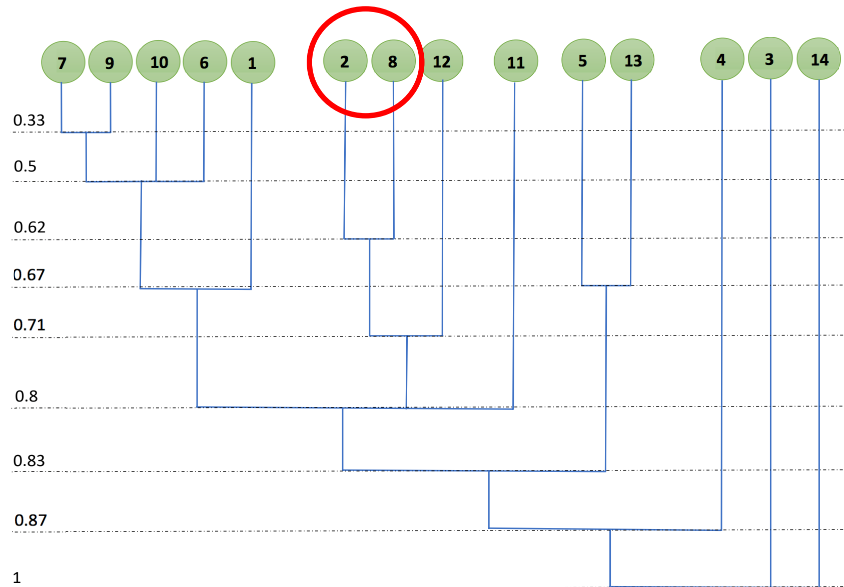


Figure 4.3: HAC Dendrogram of LUBM's 14 Queries, Queries 2 and 8 are marked in red circle

This distance matrix is used to start the HAC. Recalculation of the distance matrix is based on the choice of linkage from single, complete, or average (Figure 3.11). Single linkage is the proximity between two nearest neighbors, complete linkage is the proximity between the farthest neighbor and average linkage is arithmetic mean of all proximities between each object on each cluster with every object on another cluster. Running HAC using a single linkage on LUBM queries gives a dendrogram (Figure 4.3). Clustering is computed periodically, based on the changes in the query workload and generates new dendrograms.

4.3.3 Hierarchical Agglomerative Clustering of Queries

The adaptive partitioning algorithm (Table 4.4) takes the initial partitioning and a new query workload as input and outputs the partition minimizing the distributed joins, based on the new workload, with its new sets of features. To eliminate replication of data, only one copy of query features is stored in the shards. For removal of the replication and to decide in which shard the only copy of triples associated with the selected features will be transferred, the algorithm compares the statistics for each P or PO feature in each shard.

Input	Feature Distance Matrix ***D*** of workload Query
Output	HAC Dendrogram I
1	Assign for each D[n][n] into C[m] where m = n*n
2	**while** C.size > 1 **do**
3	**for** i = 1 **to** C.size **do**
4	**if** (c_a,c_b) = *min* $d(c_a,c_b)$ in C //Distance *funct.* ***d(c1,c2)***
5	delete c_a and c_b from C
6	add *min* $d(c_a,c_b)$ in C
7	assign **I** = (old, $c_a c_b$, , *min* $d(c_a,c_b)$)
8	recalculate proximity matrix using (SL/CL/AL) **P = modifyDistance(** c_a ,c_b ,*min* $d(c_a,c_b)$**)**
9	for each P , c_m = P[i][j]
10	Update C = $c_1, c_2, \ldots, c_m$
11	Output **I**

Table 4.3 HAC Algorithm

Input	Initial Partition **P,** features $\mathbf{F_G}$, New Queries workload $\mathbf{Q_{new}}$
Output	Adaptive Partition **A**
1	Add queries $\mathbf{Q_{new}}$ and its frequency $\boldsymbol{f}$ in Q_{old}
2	Avg query execution time(T_{base}) =$\left(\sum_{Q=1}^{n}\left(\frac{\sum_{i=1}^{f} T_{Qi}}{f}\right)\right)/n$

3	Analyze Query $\mathbf{Q}_{new}$ for features $\mathbf{F}_{\mathbf{Q}new}$
4	Run **HAC** on $\mathbf{F_Q}$, where $\mathbf{F_Q} = \mathbf{F}_{\mathbf{Q}old} + \mathbf{F}_{\mathbf{Q}new}$
5	Create Feature set **g** based on HAC at similarity distance **d**
6	**Statistics (g, $\mathbf{F_Q}$)**
7	Find key features $\mathbf{F_K}$ in **g**.
8	Find distributed joins of workload $\mathbf{D}_{\mathbf{Q}(old+new)} = (D_Q * f)$
9	Find stats $\mathbf{S_K}$ for each $\mathbf{F_K}$
10	Find p, q, s for shard C_i and complete dataset T
11	$\mathbf{S_K} = (p_c w_1 + q_c w_2 + s_c w_3) + (p_t w_4 + q_t w_5 + s_t w_6)$ //key features in p (peer features), in q (query), s (triple size) and w_1 to w_6 are weights. c and t are cluster and total.
12	**Score** for each $\mathbf{F_K} = [min\,(D_{QR}) * w * f] + S_K$ //D_{QR}(distributed joins of $\mathbf{F_K}$ in all query in every shard), w (weight) and S_K (key feature stat score).
13	**Balance_Partition (Score, g, $\mathbf{F_G}$)**
14	select all F_K from g with highest scores for each F_K
15	Assign data associated to features set **g** into **P'.**
16	**Proximity_Query ()**
17	Find $\mathbf{F_{prox}}$= proximity of $\mathbf{F_{Unclustered}}$ with $\mathbf{F_{Clustered}}$
18	Assign max($\mathbf{F_{prox}}$) in cluster P_i' with their neighbor F.
19	Assign $F_X = F_X$ + remaining F_U
20	**while** F_X not empty **do**
21	$\mathbf{P'_{min}}$ = Find min**(P')** by size of data
22	$\mathbf{F_{max}}$ = Find max**(F** in $\mathbf{F_X}$**)** by size of data
23	-Assign $\mathbf{F_{max}}$ into $\mathbf{P'_{min}}$
24	Avg execution time$(T_{new}) = \left(\sum_{Q=1}^{p+n} \left(\frac{\sum_{i=1}^{f} T_{Qi}}{f}\right)\right)/(p+n)$
25	**if** avg**($\mathbf{T}_{new}$)** < avg**($\mathbf{T}_{base}$)** **then** A = P'
27	**else** Revert back and no change in P, A=P
28	Output **Adaptive Partition *A***

Table 4.4 Knowledge Graph Adaptive Partitioning Algorithm.

The statistics use other feature patterns, such as SSJ, OOJ and OSJ and distributed joins in queries. The statistics comprise of (1) out degree sequence (hops) starting from the key feature (q) in a query graph pattern and its successive (peer) feature (p) present in the sequence, (2) triple size ratio (s) of the key feature and its successive (peer) features in shards and in the complete dataset, and (3) distributed joins in the queries. To balance the partition, the algorithm uses the statistics to determine the out degree of other features in the query to the key feature. It also uses features that are not involved in the workload, but present in the dataset. The algorithm monitors the query execution time and stores the statistics. It outputs the changes to shard compositions, based on the above information. Triples associated with the selected features are moved between shards and the partition metadata is updated. This operation is infrequent, and we assume that the system adjusts the partitioning only after identifying a significant change in the workload processing (the system monitors the execution time for each query). Typically, once the execution time increases significantly (given a threshold) the current partitioning is modified and an exchange of triples takes place. Queries from the new workload run according to the updated partition metadata and the runtime of the queries are being recorded.

4.4 Implementation

AWAPart stores an RDF dataset by partitioning it into sub-graphs, based on the initial query workload, and distributing the sub-graphs as shards among the nodes in a cluster. As the query workload changes, AWAPart establishes a new partitioning optimized for the new workload and dynamically adjusts the shards by triggering exchanges of subsets of triples between shards. The system is deployed on a single Master Node which controls the adaptive partitioning and a set of independent, share-nothing Processing Nodes, each with an installed triple store and a SPARQL query processor. The Master Node (Figure 4.4) is responsible for the overall workload analysis. It

also controls the movement of triples subsets among the nodes in the cluster to adjust the partitioning. As the Master Node receives the query, the QueryAnalyzer and Feature Extractor (QAFE) starts the query feature extraction and updates the feature metadata. The Partition Manager (PM) uses the Hierarchical Agglomerative Clustering (HAC) module to cluster the extracted features. Using this HAC information, the Partition Metadata (PMeta) is updated. The dataset is indexed (IS) and according to PMeta, triples are searched and stored as shards. These shards are being uploaded to the processing nodes for the first time. A new query is sent to Query Rewriter and Processor (QRP), which rewrites the query into a federated query, based on the Partition Metadata (PMeta).

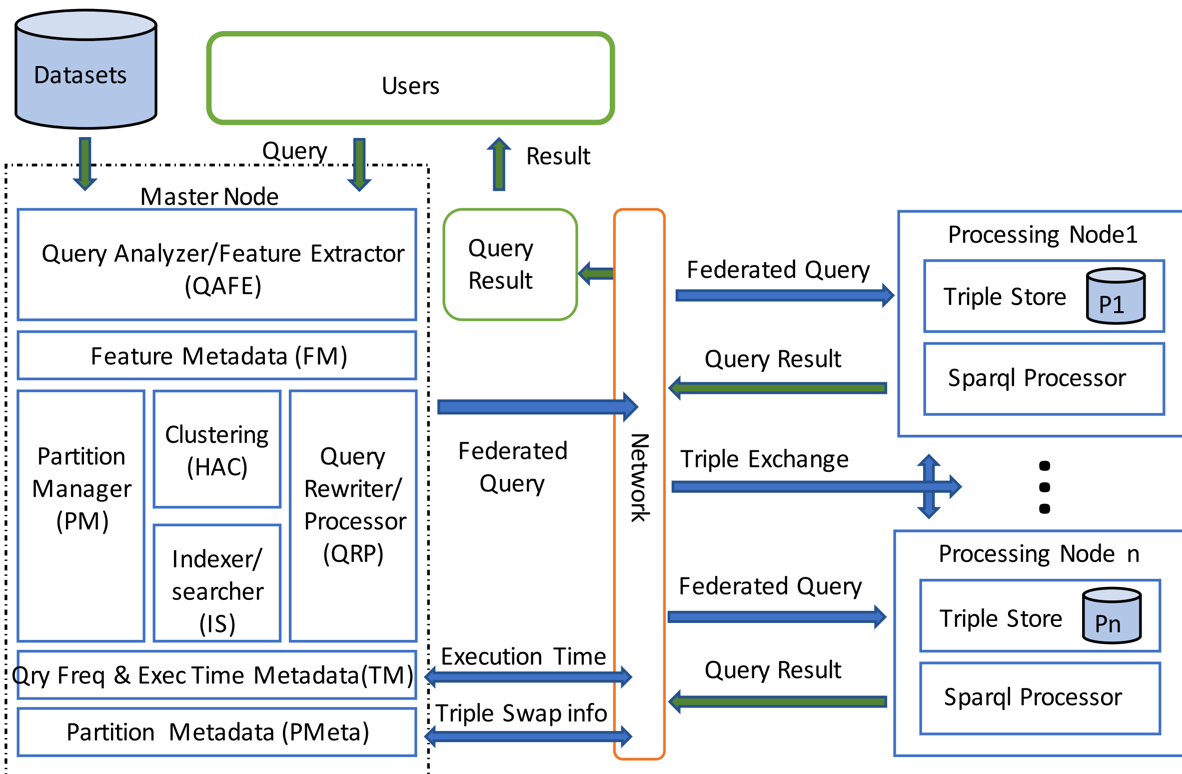


Figure 4.4 AWAPart System Architecture

This federated query is then sent to the processing node where it is going to be executed. The node where the query is executed is called the Primary Processing Node (PPN). The PPN is selected to

minimize the distributed joins by selecting the shard with the highest number of features for the query. Adjustment of the partitioning of the RDF data is triggered by the Partition Manager (PM), due to changes in the workload query set and/or query frequency. The PM computes a new partition and, if the current shards require modifications, triples with selected features are exchanged between Processing Nodes to achieve a desired partitioning. The metadata of each Processing Node that was involved in triple swaps is updated to reflect the current state of triples in the shards. The PM uses the information stored in the Query Frequency and Execution Time Metadata (TM) and in the Feature Metadata (FM) with clustering information given by the Clustering Unit (HAC) to update Partition Metadata (PMeta). TM stores the information of every unique query and its average runtime.

4.5 Experiments

The synthetic dataset and queries in the Lehigh University Benchmark (LUBM [42]) were used for the evaluation of AWAPart, our knowledge graph adaptive partitioning method based on a query workload. LUBM includes basic information organized as a knowledge graph about a set of universities and related entities. It includes a set of 14 SPARQL queries intended for benchmarking of knowledge graph storage/query systems. The experiments were conducted on a cluster of Intel i5-based systems running Linux Ubuntu 18.04.4 LTS 64-bit OS. A relatively small cluster was selected to focus on the effects of repartitioning of the datasets of manageable sizes. There are many available RDF triple stores that provide the functionality of storing and querying the RDF data, such as Redland [10], Sesame, Jena [11], Virtuoso, etc. In the experiments, an instance of OpenLink Virtuoso [14] was installed on each node in the cluster. The knowledge graph partitioning and adaptive repartitioning systems, as well as the experiments were coded in Java with the use of the Apache Jena framework.

Two experiments were used to evaluate the effects of adaptive knowledge graph partitioning system, based on workload. (1) The first experiment was designed to evaluate the effectiveness of the adaptive partitioning to accommodate the changes in the set of queries in the workload. (2) The second experiment was created to evaluate the adaptive partitioning in response to the changes in the frequency of specific queries in the workload (the set of queries in the workload is unchanged, but some queries are executed more often than initially). An LUBM dataset of 10 universities, which included 1,563,927 triples was created and used. The initial partition [15] is created based on the initial query workload. The experiments show that the AWAPart system offers significant performance improvements over a system where the initial partitioning was unchanged.

Experiment 1

This experiment demonstrates the effects of changes in the composition of the workload query set on their performance, when executed on the initial partition and then on the adaptive partitioning. The changes to the workload included additions of new and/or deletions of existing queries. The modified workloads runtime on the initial partition and on our adaptive partitioning for the LUBM

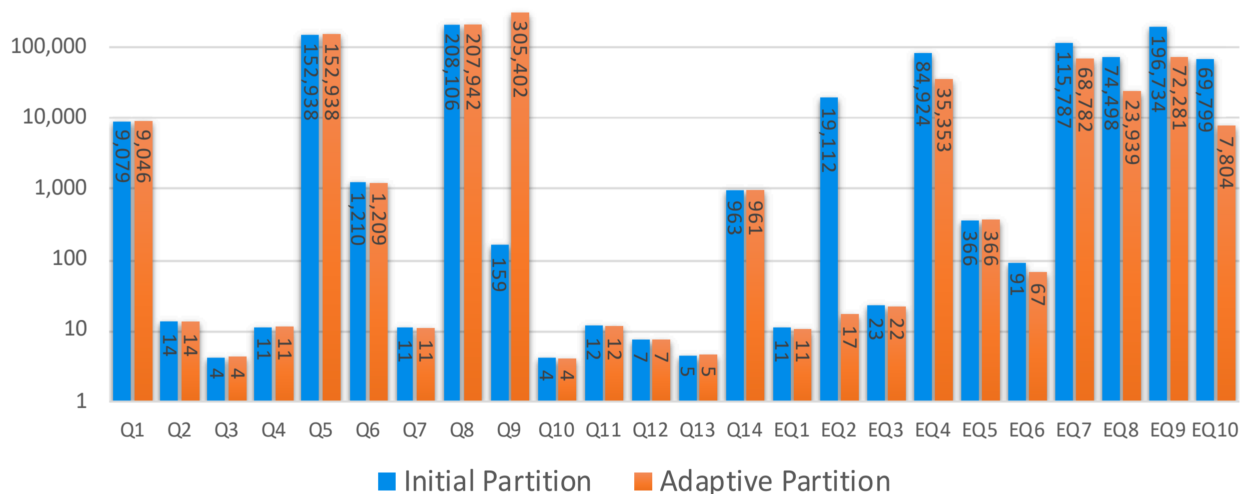


Figure 4.5. LUBM’s 24 queries runtime in milliseconds

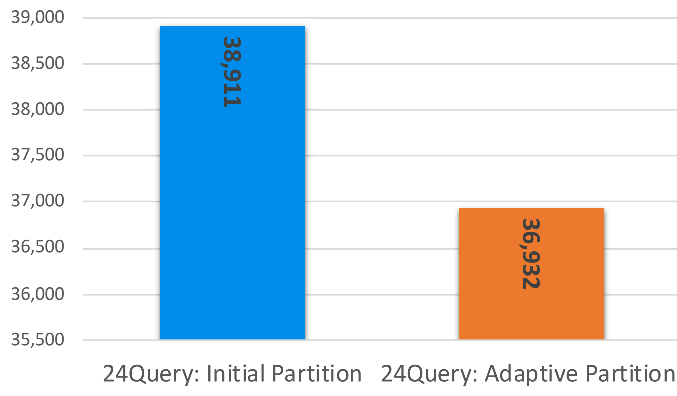


Figure 4.6. LUBM all 24-query average runtime in milliseconds

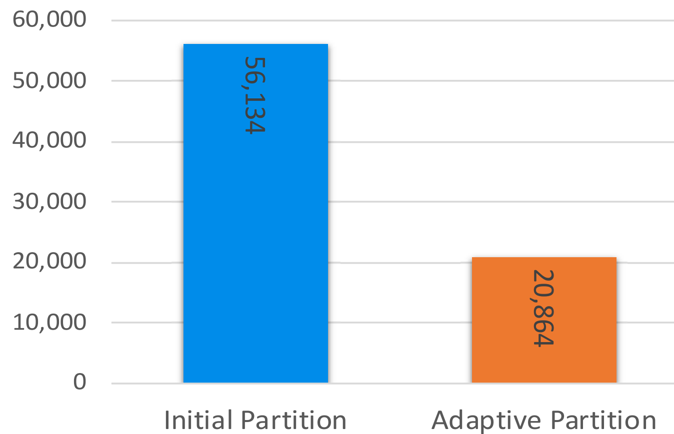


Figure 4.7. LUBM 10 new queries average runtime in milliseconds

dataset were evaluated. Figure 4.5 shows 10 extra queries [46] EQ1 to EQ10 and 14 old queries Q1 to Q14 for the LUBM dataset and their runtimes. EQ1 to EQ10 are a mixture of linear, star, snowflake, and complex queries. The figures show the improvement in runtime performance for queries from EQ1 to EQ10 in milliseconds. Except for Q9, the performance of the other 13 original queries do not change. Figure 4.6 shows the average runtime of all 24 queries on the initial partition versus the adaptive partition in milliseconds. An overall improvement of 2 seconds of the adaptive partition over the initial partition is shown. Despite the drop in performance of a single query, the overall performance gains are clearly visible. If Q9 were replaced in the new workload

composition (Figure 4.5), the performance gain would be even higher. Figure 4.7 shows that the improvement of the average runtime of the 10 new queries (EQ) on the initial partition is approximately 56 seconds, while the adaptive partition decreases it to 21 seconds. It is an improvement of 63% in the average runtime of the newly introduced queries on the adaptive partition over the initial partition. This experiment shows that the system can successfully adapt the partitioning with changes in the workload. At regular intervals, the system takes a snapshot of the current query workload and adapts the partitioning, which improves the workload runtime performance. In Figure 4.8 and Figure 4.9 triples associated with properties is being swapped between partitions to improve the workload response time by reducing distribted joins in queries.

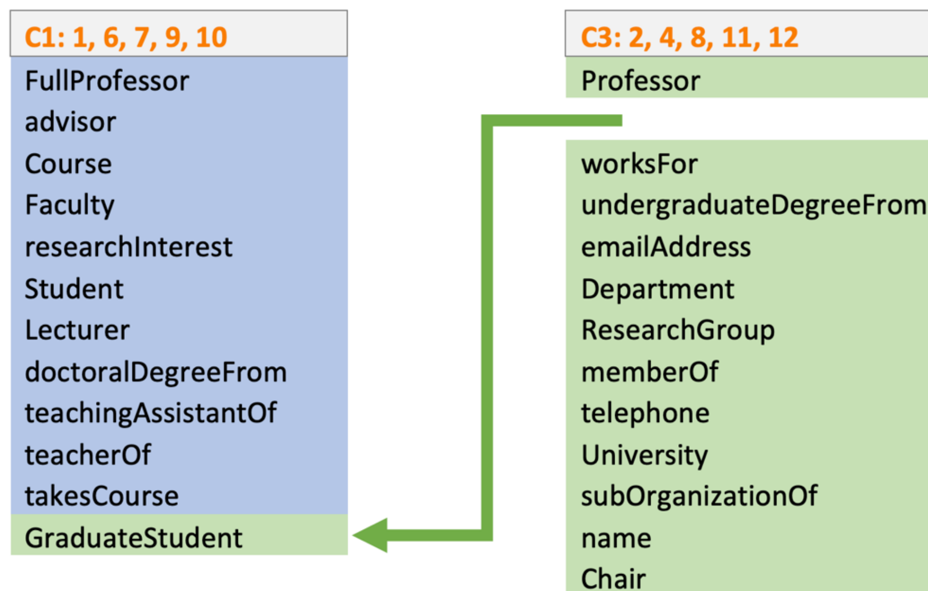


Figure 4.8. Example 1 of Swapping of triples associated with properties between partitions.

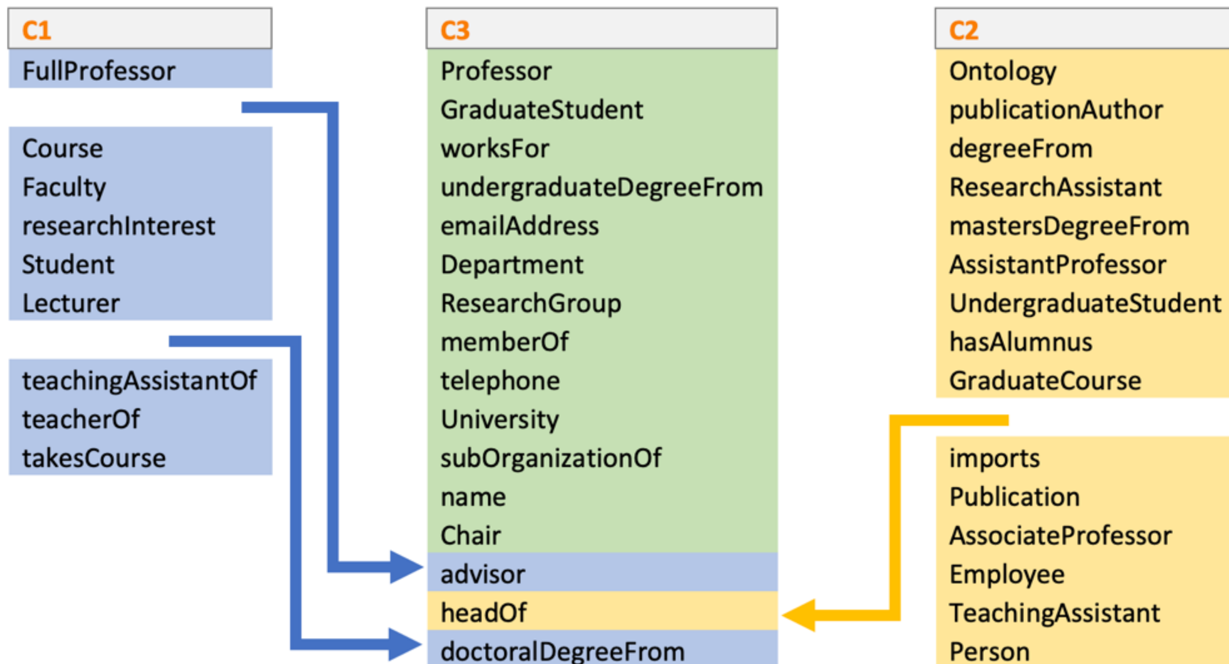


Figure 4.9. Example 2 of Swapping of triples associated with properties between partitions.

Experiment 2:

This experiment examined the effects of the changes in the relative frequency of queries in the workload executed on the initial partition as compared to the adaptive partitioning and so the workload query frequency distribution was altered. For example, if Q1 in LUBM is executed more frequently than the other 13 queries. The workload frequency share of query Q1 was increased to 50% of the whole workload. Figure 4.10 shows the changes in the runtime of queries Q1 and Q2. Queries 1 and 2 shares the same features. Our system swaps the queries based on score. This swapping reduces the distributed joins of Q1 but increases the distributed joins in the less frequently executed Q2, while maintaining the average runtime for the workload with evenly

distributed queries. However, when the workload frequency is biased towards Q1, Figure 4.11 shows the improvement in the average workload performance by comparing the average runtime of the initial partition with biased workload frequency and adaptive partitioning with the biased workload frequency. The figure shows an improvement of approximately 17% of the adaptive partitioning over the initial partition, when the workload frequency is biased towards Q1.

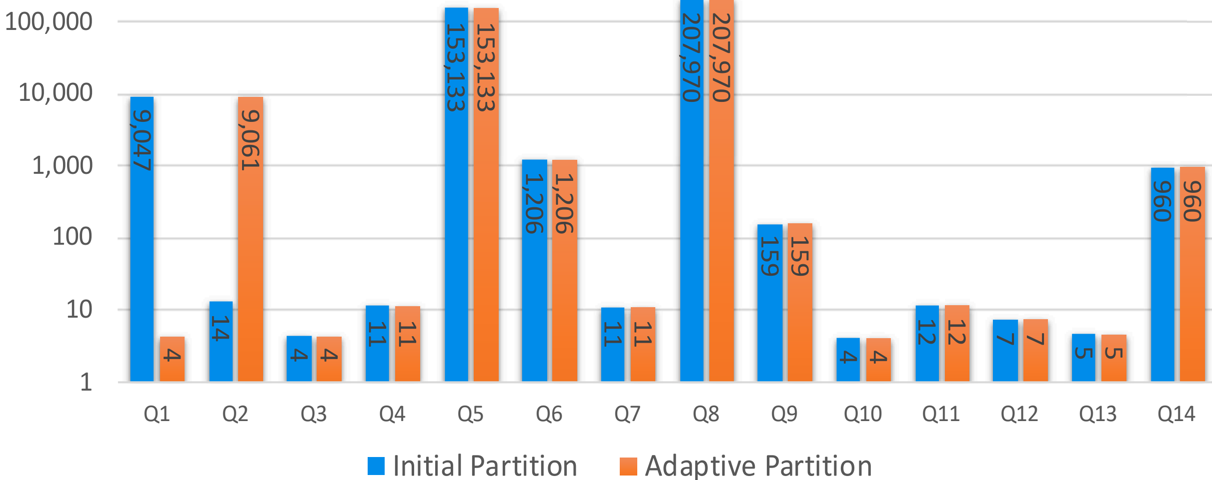


Figure 4.10. LUBM all queries average runtime of Initial vs. Adaptive partition in milliseconds

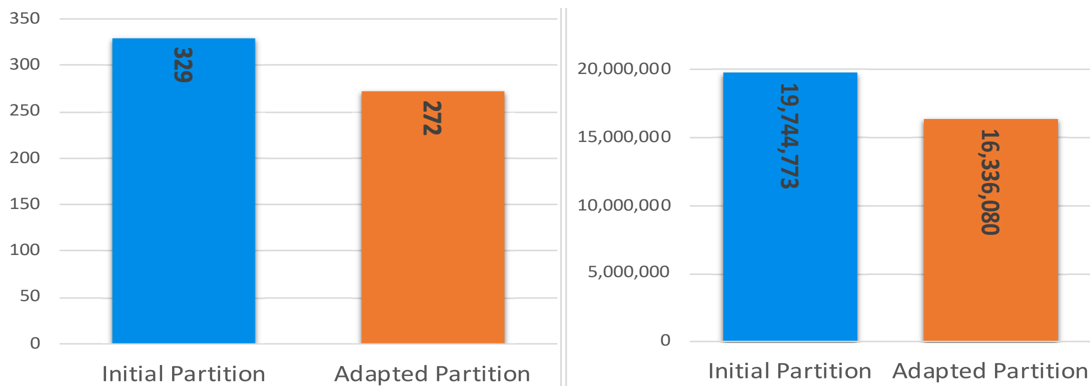


Figure 4.11. LUBM all query average runtime when frequency of Query1 is 50% of total workload a) Total runtime in minutes. b) Total runtime in milliseconds.

The experiment shows that, when a query has a higher frequency than others, the performance of the adaptive partition against the initial partition is improved. Consequently, the system is adaptive to the changes in the workload. Again, at regular intervals, e.g., daily or after a set number of queries, the system takes a snapshot of the current query workload and query frequencies and, if needed, adapts the partitioning, which improves the average performance of the workload.

4.6 Conclusion And Future Work

In this paper, a system is proposed which is a distributed knowledge graph query processing system that adaptively partitions the graph according to changing workload. It aims to reduce the number of distributed joins during query execution that eventually leads to a reduced run-time for the queries achieving better performance. The system is adaptive with the new workload and the system learns the workload regularly and modifies the partition, which eventually improves the partition's overall runtime performance. Our experiments show the runtime comparison of workload aware initial partition versus adaptive partition. The results depict a significant increase in the performance of the queries. There is no need for replication of the data while optimizing the runtime of the workload queries.

In the future, a study of an evolving knowledge graph in terms of its schema and instances should be undertaken. Also, it will be interesting to examine how the adaptive partitioning handles the evolving datasets along with the evolving workload queries.

CHAPTER 5

PartKG2Vec: EMBEDDING OF PARTITIONED KNOWLEDGE GRAPHS [*]

Abstract. Large-scale knowledge graphs with billions of nodes and edges are increasingly common in many domains. Such graphs often exceed the capacity of the systems storing the graphs in a centralized data store, not to mention the limits of today's graph embedding systems. Unsupervised machine learning methods can be used for graph embedding, which can then be used for various machine learning tasks. State-of the art embedding techniques are often unable to achieve scalability without losing accuracy and efficiency. To overcome this, large knowledge graphs are frequently partitioned into multiple sub-graphs and placed in nodes of a distributed computing cluster. Graph embedding algorithms convert a graph into a vector space where the structure and the inherent property of the graph is preserved. Running such algorithms against these fragmented sub-graphs poses new challenges, such as maximizing the likelihood of preserving network neighborhood of nodes. Also, the learned embeddings of the individual graph partitions need to be merged into one overall embedding to maximize the likelihood of preserving network neighborhood of nodes. This paper introduces a novel method for embedding of partitioned knowledge graphs. It partitions the knowledge graph and executes learning algorithm in parallel on the partitions and merge their outputs to produce an overall embedding. Our evaluation demonstrates that the runtime performance is improved after partitioning of knowledge

---

[*] *Reproduced with permission from Springer Nature & KESM*
A. Priyadarshi and K. J. Kochut, "PartKG2Vec: Embedding of Partitioned Knowledge Graphs," in *International Conference on Knowledge Science, Engineering and Management*, 2022: Springer, pp. 359-370.

graph against complete knowledge graph and the quality of the embedding is similar to that of an embedding produced on the complete, unpartitioned graph.

Keywords: Knowledge graphs, Graph partitioning, Feature learning, Node embedding, Graph representation learning.

5.1 Introduction

Recently, large-scale knowledge graphs, have been used for representation of transportation networks, e-commerce and shopper preference networks, social and communication networks, and many other real-world systems. Such graphs often hold hundreds of millions, or even billions of vertices and edges. Many data processing methods rely on large-scale graph analytics, which are often based on node and graph embedding and node feature extraction, which can further be used in various machine learning tasks. However, state-of the art techniques are not scalable to large graphs without losing accuracy and/or efficiency. A knowledge graph often needs to be partitioned into multiple sub-graphs, called shards, and stored at multiple computing nodes, which then requires distributed or parallel graph processing. Graph embedding, also known as network embedding, is a frequently used technique for learning low-dimensional representations of graph's vertices, attempting to capture and retain the graph's structure, as well as its inherent properties. Many tasks on graphs, such as link prediction, node classification, and visualization, greatly benefit by embedding a very large, web-scale graph into a low-dimensional vector space. More specifically, we might be interested in estimating the most likely labels for nodes in a network, or predicting user interests in a social network, or we might be interested in predicting functional labels of proteins in a protein-protein interaction network [48]. Similarly, in a link prediction task [49], we might want to know if a pair of nodes in a graph should be connected by an edge. Link

prediction is beneficial numerous fields. For example, in bioinformatics, it aids in the discovery of novel protein interactions [50], and it can recognize “real-world buddies” on social networks [51]. A knowledge graph (KG) is a directed graph G (V, E) whose nodes $v_i \in V$ are entities and edges $e_i \in E$ are relations connecting entities. Knowledge graphs are often represented as RDF [1] datasets, where triples (v_i, e_i, v_j) represent some type of semantic dependency between the connected entities and nodes/entities are identified by URI’s. Target nodes in triples are either URIs or literals, and edge/relationships have types represented by URIs, as well. RDFS [4] is used to define a schema for an RDF knowledge graph. KGs are closely related to Heterogenous Information Networks (HIN) [16].

Various approaches to graph embedding have been presented in the machine learning literature e.g., [52-54]. They function well on smaller networks, but real-world knowledge graphs, which often have millions of nodes and billions of edges, present a far more difficult situation. For example, a decade ago, the Twitter’s followee-follower network had 175 million active users and approximately twenty billion edges [55]. The majority of the existing graph embedding algorithms do not scale up to networks of this magnitude.

Knowledge graphs can be partitioned into smaller subgraphs, with a hope that many tasks can take advantage of distributed and/or parallel processing. Given a graph G = (V, E), where V is a set of vertices and E is a set of edges and a number $k > 1$, a graph partitioning of G is a subdivision of vertices of G into subsets of vertices V_1, ..., V_k that partitions the set V. A balance constraint requires that all partitioned subgraphs be equal, or close, in size. In addition, a common objective is to minimize the total number of cut edges (min-cut), i.e., edges crossing (cutting) partition boundaries.

In this paper we propose PartKG2Vec [47], an algorithm for scalable feature learning in partitioned knowledge graphs. Our approach creates embeddings based on random walks in partitioned knowledge graphs and offers significant runtime improvements due to performing the walks in parallel. This is important in random walk-based methods, especially in semantics based, such as metapath2vec [56], due to the high cost of selecting the next node at each step during a random walk.

The rest of the paper is structured as follows. In Section 2, we briefly discuss related work. We present the technical details for PartKG2Vec in Section 3. In Section 4, we briefly explain the implementation of PartKG2Vec. In Section 5, we empirically evaluate PartKG2Vec. We conclude with a discussion of the PartKG2Vec framework and highlight some interesting directions for future work in Section 6.

5.2 Related Work

Recently, graph representation learning has attracted a lot of attention. In general, there are two types of graph representation learning methods: unsupervised and supervised. The goal of unsupervised approaches is to learn low-dimensional representation that preserves the structure of a given graph. The supervised methods work in the same way as the unsupervised methods, but for a specific prediction task, such as node or graph classification. Only unsupervised approaches are discussed in this paper.

Unsupervised *embedding methods* map a graph's nodes and edges, into a continuous vector space. Several of the graph embedding techniques have been motivated by the Word2Vec algorithm **[57]**, which originated in natural language processing. One type of this algorithm relies on the *skip-gram embedding models*, where a word's embedding is optimized to predict its context or adjacent

words. A random walk in a graph is akin a sequence of words in a sentence, where the nodes visited in the walk can be thought of words in a sentence.

Deepwalk [58] is one of the first approaches to embedding of graph-structured data. Deepwalk relies on the parallels between graph nodes and words and its neural networks are trained to maximize the likelihood of predicting the context nodes for each target vertex in a graph, in terms of vertex proximity.

node2vec [59], is a popular unsupervised graph embedding algorithm, which extends Deepwalk's sampling strategy. It utilizes random walks. Also, node2vec utilizes breadth-first and depth-first to capture both local and global community structures, resulting in more informative embeddings.

LINE [60], which is an acronym of Large-scale Information Network Embedding, produces embeddings ensuring first- and second-order proximity. For first order, LINE minimizes the graph regularization loss, and for second order, decodes embeddings into context conditional distributions for each node, which is computationally expensive. Negative sampling is used by LINE to sample negative edges based on a noisy distribution over edges. Finally, LINE combines first and second order embeddings with concatenation.

HARP [61], or Hierarchical representation learning for networks lowers the number of nodes in the graph by coarsening the graph in a hierarchical manner. Iteratively grouping nodes into super nodes, it creates a graph with similar properties to the original network, resulting in graphs of reduced size. Existing approaches, such as LINE or Deepwalk, are then used to learn node embeddings for each coarsened graph. The random walk technique on G_{t-1} uses the embeddings learned for G_t as initialized embeddings at time-step t. This technique is repeated until each node in the original graph is embedded.

5.2.1 Embedding of Partitioned Graphs

PyTorch-BigGraph [62], also known as PBG, is a multi-relation embedding system that can scale to graphs with billions of nodes and trillions of edges by incorporating various improvements to existing multi-relation embedding systems. PBG can trains very large embeddings using a distributed cluster using graph partitioning. The adjacency matrix is decomposed into N buckets, with each bucket training on the edges individually. PBG then performs distributed execution across multiple machines or swaps embeddings from each partition to disk to reduce memory use.

MILE [63], or Multi-Level Embedding, is a graph embedding framework that can scale to large graphs. It uses a hybrid matching technique to repeatedly coarsen the graph into smaller ones while maintaining its structure. It then uses known embedding methods on the coarsest graph and uses a graph convolution neural network to refine the embeddings to the original graph. It is independent of the underlying graph embedding techniques and may be applied to a wide range of existing graph embedding methods without requiring them to be modified. It has been demonstrated that, MILE dramatically improves graph embedding time (by an order of magnitude).

Accurate, Efficient and Scalable Graph Embedding [64] relies on the GCN [65]model and its variants are strong graph embedding tools for enabling graph classification and clustering. A unique graph sampling based GCN parallelization strategy that achieves excellent scalable performance on very large graphs without sacrificing accuracy. To scale, it uses parallelism within and across many sampling instances for the graph sampling step and devises an efficient data structure for concurrent accesses. Data partitioning improves cache utilization within the sampled graph. On several large datasets, its parallel graph embedding exceeds state-of-the-art approaches in terms of scalability, efficiency, and accuracy.

PartKG2Vec, presented in this paper, is a parallel processing of partitioned knowledge graph to generate the random walk for the embedding. PartKG2Vec is a graph embedding system capable of handling big graphs. A parallelization approach that achieves good scalability on very large graphs while maintaining accuracy. PartKG2Vec can be used with any random walk generator algorithm with minor adjustments.

The nodes in the knowledge graph are partitioned into the sub-graphs using METIS [24, 66]. First, random walks of the subgraphs in a partitioned graph generate partial random walks. These partial walks are then combined to form complete walks. This way, the likelihood of preserving network neighborhoods of nodes in a d-dimensional feature space is maximized. The random walks are performed independently, starting with the initial nodes within each partition. Some of these walks will be incomplete (shorter than a desired length), as they reach a partition boundary. These walks are then completed with fragments of random walks in the neighboring partitions. A full set of complete walks is then used for representational learning to generate knowledge graph embedding. PBG and PartKG2Vec use partitioning to support Knowledge Graph which is too large for a single machine and also helps in distributed training of the model. PBG creates buckets from the cross edges (p_i, p_j), These buckets are loaded and subdivided among the CPU threads for training. PartKG2Vec is different, as we create the metadata of cut edges for each partition and complete or partial random walks are generated separately in each partition. Before the representation learning, the partial random walks are completed by concatenation with other walk fragments (from neighboring partitions). MILES repeatedly coarsening the graph into smaller graph, where multiple nodes in graph are collapsed to form super nodes and edges between them are the union of edges. Whereas in PartKG2Vec, we partition the knowledge graph to reduce the size of graph while maintaining the structure.

5.3. Partitioned Knowledge Graph Embedding

Our method, PartKG2Vec, (1) partitions a knowledge graph into *k* partitions, (2) distributes the resulting shards to *k* computing nodes, (3) in each shard, *complete* and/or *partial random walks* are created; a walk is partial, if a cut edge on a walk is encountered, before a desired walk length is reached, (4) a complete set of random walks is obtained from already existing complete walks in individual shards and by concatenating partial walks with sub-walks in neighboring partitions; a sub-walk is a fragment of a walk beginning with the target node in a cut edge terminating the corresponding partial walk, and (5) using the complete walks for graph embedding. Further downstream, graph embeddings can be used to solve other problems, including link prediction, node classification, and many other tasks.

Knowledge graphs used in the method presented in this paper are represented as RDF datasets. However, other graph representations can be easily adapted and used in PartKG2Vec.

5.3.1 Graph Indexing and Partitioning and Segregator

In order to speed up the process of learning embedding for the knowledge graph, we created indices, using Apache Lucene [38], on all triples in the knowledge graph, based on their subjects, predicates and objects. Using these indexes, the graphs tripes (edges) of the form (S, P, O) can be efficiently searched, similarly as we have done in our prior work WawPart [15] and in AWAPart [44]. This indexing helps the system to convert the knowledge graph into a representation suitable for graph partitioning, as the URIs used in triples are also converted to numeric identifiers. This new graph representation is then partitioned into several sub-graphs using METIS [24, 66]. We have experimented with other ways to partition the knowledge graph, including bisection methods,

community detection methods, and others, but found METIS to produce the best partitions with a low number of cut edges in acceptable runtime.

The partitioning outputs the node list and their partition identifiers. Our system compares this list with the complete graph to produce the list of cut edges. Consequently, along with its edges, each partition stores information about the cut edges. The cut edges are replicated across the shards which share the cut edges. That is, a cut edge {u, v} is stored with both partitions, to which the nodes u and v belong.

5.3.2 Random Walk Generation

The created knowledge graph partitions (sub-graphs) are stored as shards at computing nodes for the processing and random walk generation. Figure 1(a) shows an example of two partitions *P1* and *P2* with vertices [54 d, u, v] and {*m, n, o, p, q, v, u*}, respectively. The partitions are connected with a cut edge {*u, v*}. A modified node2vec algorithm attempts to generate random walks of *walk_length* length, within each partition. However, if a walk-in partition P1 encounters a cut edge {*u, v*} transitioning to partition P2, the random walk is terminated and recorded it as a *partial_walk*. The node *v* is recorded as the *exit node* of P1 and an *entry node* of P2. Figure 1(b) shows a partial random walk, interrupted because it attempted to cross to *P2* using the cut edge {*u, v*}. Node *v* is an exit node for partition p1 and an entry node for partition p2. The *exit* and *entry nodes* and the current walk length is stored with the partial walk. This data is later used to complete the partial walk, as described below. A random walk in a partition may traverse a node *v* which is also a node in a cut edge. However, even though the node is in a cut edge, the walk does not terminate at *v* (as a partial walk) and continues within the same partition. A *sub-walk* is a sub-sequence of nodes in a random walk, beginning at a node *v* of a cut edge, but not crossing to the other partition. The modified node2vec algorithm also records and indexes all *sub-walks* within each partition. Figure

5.1(c1) and Figure 5.1(c2) show random walks and sub-walks in P2. In Figure 5.1(c3), a sub-walk does not exist, because the walk in the figure does not traverse node v, which is in the cut edge.

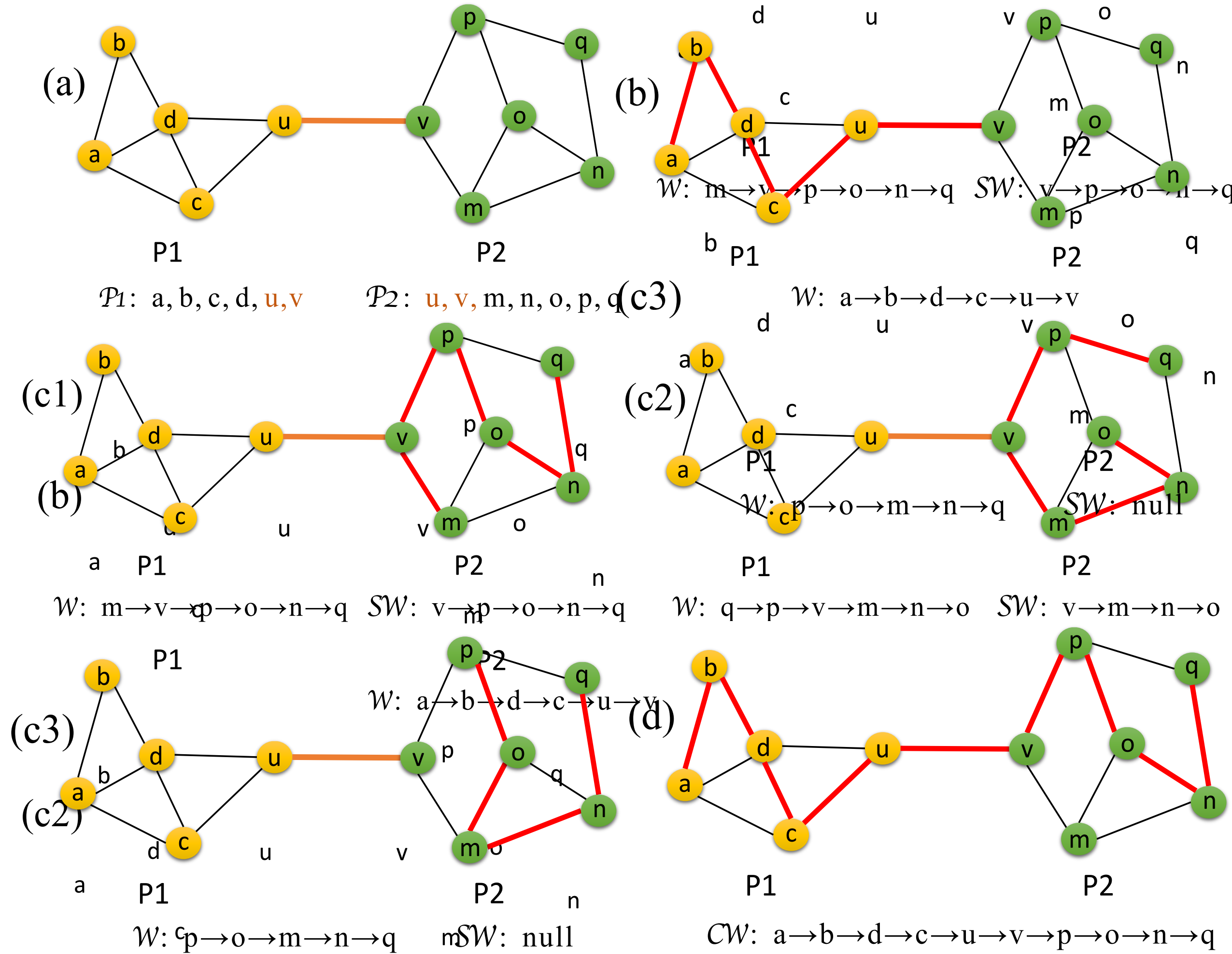


Figure 5.1. Completion of walk from partial walks. (a) Nodes in partition p1 and p2. (b) Random walk on partition p1 is interrupted because of the exit node $\{v\}$. (c) Random walks W and sub walks SW passing through the entry node $\{v\}$ on p2. (d) Formation of a complete walk CW from a partial walk in P1 and a sub-walk in P2.

5.3.3 Accumulator and Graph Embeddings

The Random Walk Accumulator collects all of the partial and complete random walks collected within all partitions. Complete walks do not need any further work, as they already have the desired walk length. However, any partial walks must be extended to the required walk length. For each partial walk, a sub-walk with the same starting node as the partial walk's exit node is randomly selected and concatenated to the partial walk to make a complete random walk. As shown in Figure 5.1(d), a complete random walk *a, b, d, c, u, v, p, o, n, q* is created using the sub-walk from Figure 5.1(c1) and the partial walk from Figure 5.1(b). Once the full set of complete random walks is created, it can be used by the representation learning module.

Input	Knowledge Graph KG
1	Indexing the Knowledge graph G
2	Numerical Representation of KG into N
3	Partitioning N into k parts using Metis
4	Segregate the Graph N into P_1 to P_k
5	Distributes P_1 to P_k for Random walk generation $RW_{P1 \text{ to } Pk}$
6	Aggregates all $PRW_{P1 \text{ to } Pk}$ into CRW
7	Learn Embedding EMB_N on CRW
8	Search in index to construct EMB_N into EMB_{KG}.
Output	Output $\mathbf{EMB_{KG}}$

Figure 5.2. PartKG2Vec pipeline

5.4 Implementation

Figure 5.2 shows the processing pipeline used in PartKG2Vec, while Figure 5.3 shows the architecture of the system. A knowledge graph is given as input and its embedding is produced as

the output. KG2Index Converter indexes the knowledge graph using Lucene [38, 39]and converts it to a format suitable for partitioning. The graph is partitioned using METIS [24, 66, 67] into *k* partitions by the Partition Engine. The partition data (the edge lists) are sent to the Graph Partition Segregator, which creates the final partitions and identifies cut edges to be included with each partition.

The *k* partitioned sub-graphs (shards) are then sent to the *k* processing nodes to produce random walks. All partial (PRW) and complete walks (CRW) are transferred from the processing nodes to the master node. The master-node runs an Accumulator, which gathers all the walks (partial/sub-walks and complete walks) and other critical information. Already complete walks are simply retained, but the Accumulator uses partial walks and matching sub-walks to create complete walks (of the desired length). At the end of accumulation process, a corpus of complete random walks is finalized. This set of random walks is then used for representation learning. Finally, the Lucene index is applied to restore the original node identifiers (URIs) in the knowledge graph embedding.

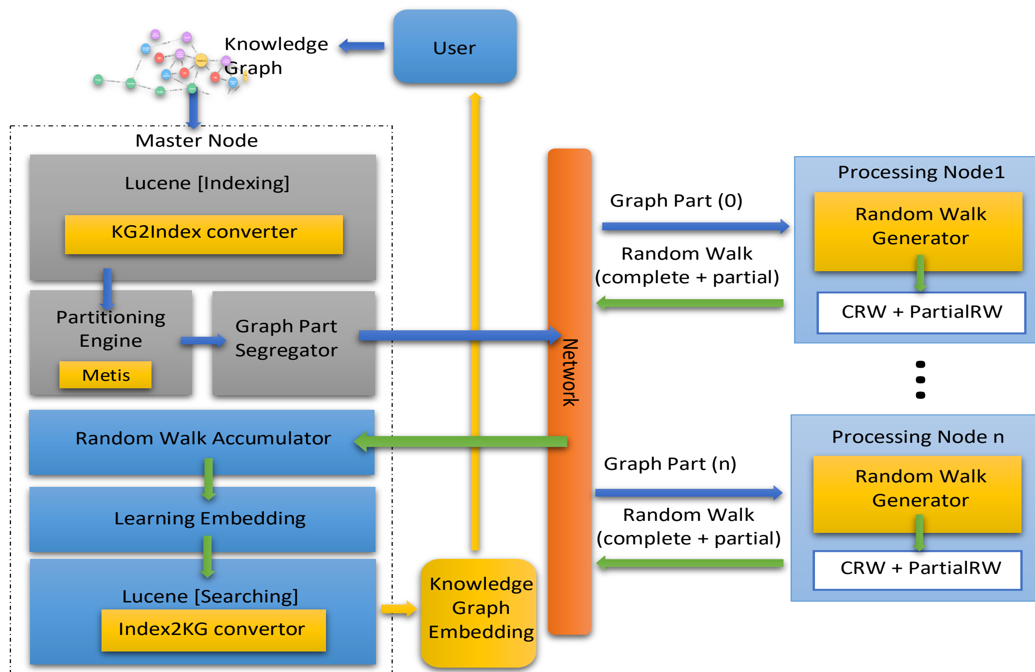


Figure 5.3. PartKG2Vec Architecture

5.5 Evaluation

Two popular datasets, Yago39K [68] [69, 70] and NELL [71] were used for the evaluation of PartKG2Vec. Yago39K contains a subset of the Yago knowledge base [69, 70], which includes data extracted from Wikipedia, WordNet and GeoNames. Yago39K contains 123,182 unique entities (nodes) and 1,084,040 edges, using 37 different relation types. NELL is a knowledge graph mined from Web documents and contains 49,869 unique nodes, 296,013 edges, using 827 relation types. The evaluation experiments discussed here were conducted on an Intel i7-based cluster.

Two experiments were used to evaluate the performance of PartKG2Vec. The first experiment was designed to evaluate the runtime of producing the embeddings on the complete vs. partitioned graph. The second experiment intended to compare the graph embeddings produced by PartKG2Vec (based on the modified node2vec and DeepWalk algorithms) with the embeddings produced by the original algorithms on un-partitioned graphs. In the two experiments, both knowledge graphs (Yago39K and NELL) were partitioned into N=10 partitions. We set all walk parameters to their default values, namely the number of walks to 10, walk length to 80, number of workers to 8, and the window size to 10, and the walk parameters of p and q both set to 1.

Experiment 1

This experiment demonstrates the improvement in the runtime of the random walk generation on the partitioned graph as compared to the random walks produced on the complete graph by the original algorithms. In Figure 5.4 and Figure 5.5, the runtime of node2vec and PartKG2Vec (with modified node2vec) on Yago39K and NELL is shown, while in Figure 5.6 and Figure 5.7, the runtime of Deepwalk and PartKG2Vec (with modified Deepwalk) on Yago39K and NELL is shown. Graph preprocessing and the 10 iterations of random walk generation are shown. Node2vec

did not require all of the steps before graph preprocessing and accumulation of random walks. All these extra steps were required for PartKG2Vec, but it did not require considerable time. Consequently, we can consider these steps as insignificant. Figure 4 indicates that the time required by PartKG2Vec for graph preprocessing took 32% of the time required by node2vec (695 vs 2175 seconds). Ten iterations of random walk generation were used in both algorithms, but PartKG2Vec runs in parallel, so it took only 17.75% of the time required by the original node2vec (480 vs. 2702 seconds) on the complete knowledge graph.

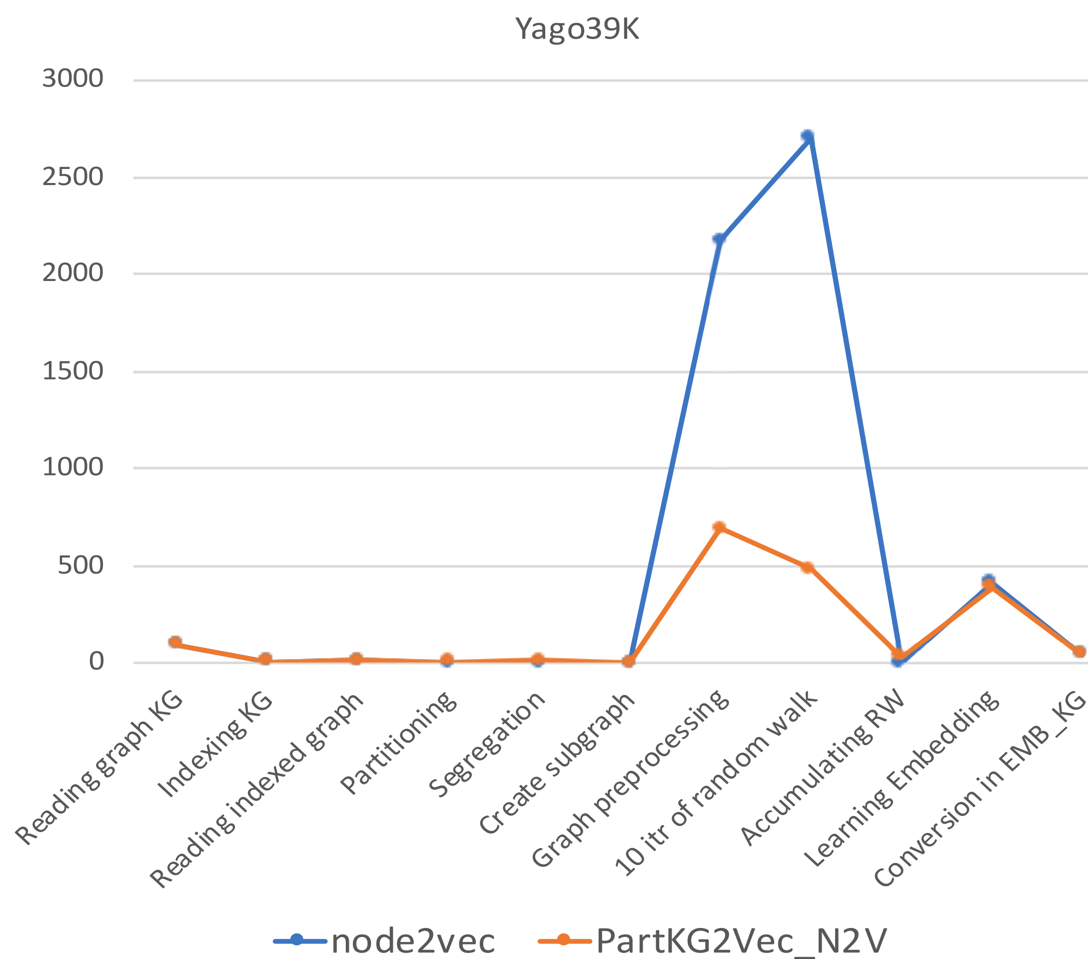


Figure 5.4. PartKG2Vec (node2vec) runtime comparison with node2vec on Yago39K

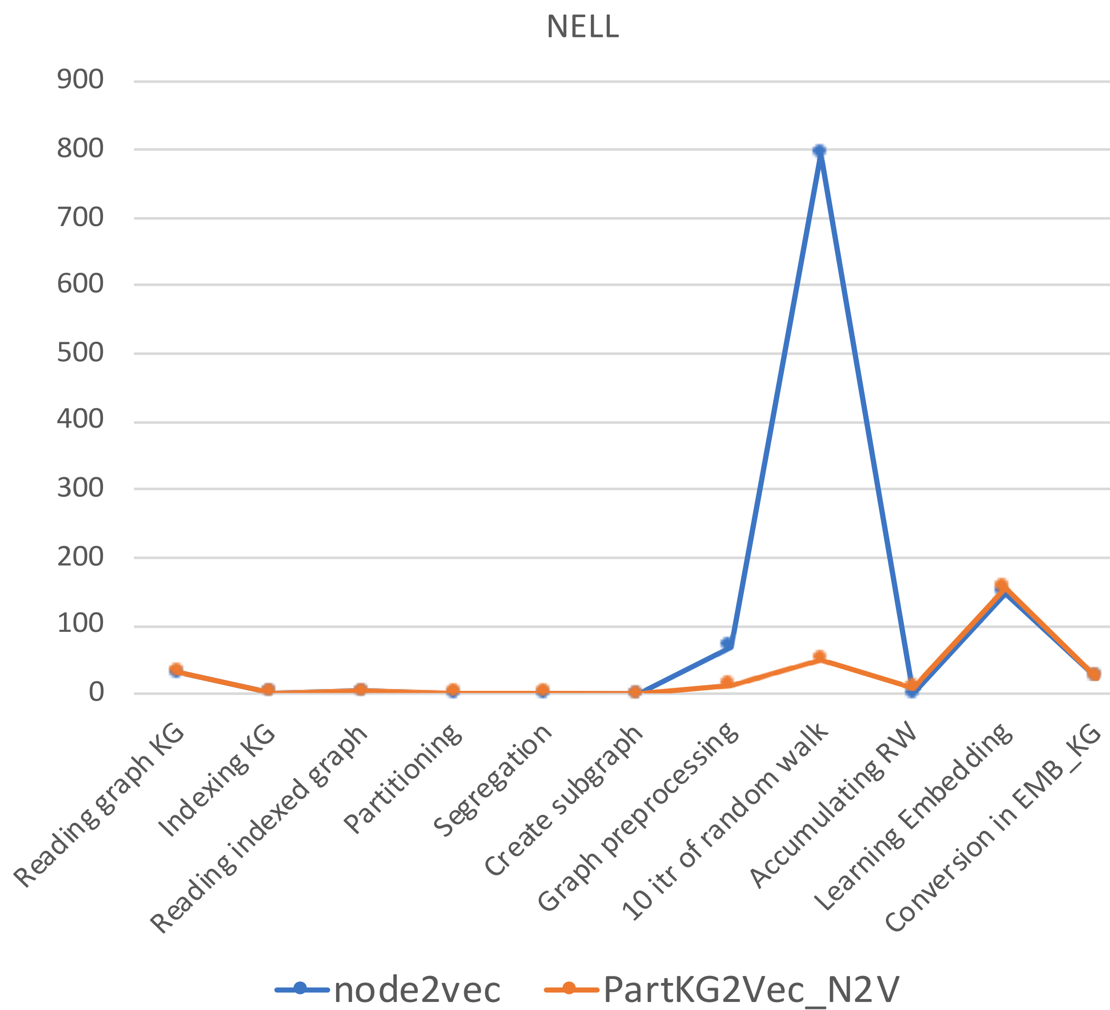


Figure 5.5. PartKG2Vec (node2vec) runtime comparison with node2vec on NELL

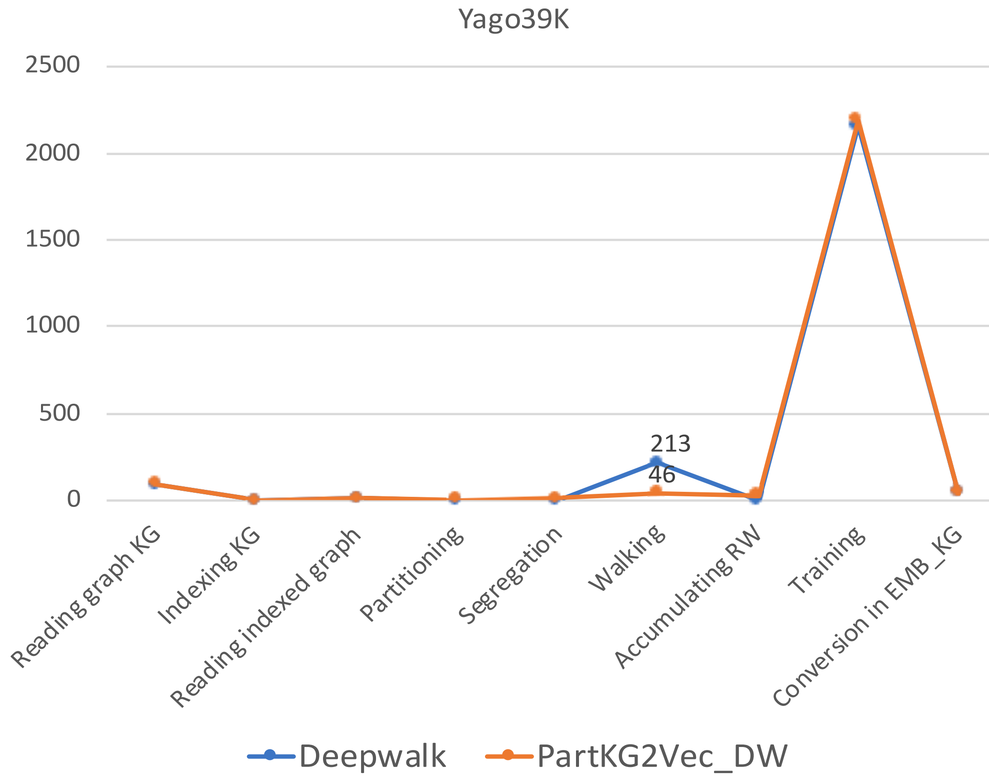


Figure 5.6. PartKG2Vec (Deepwalk) runtime compared to Deepwalk on Yago39K

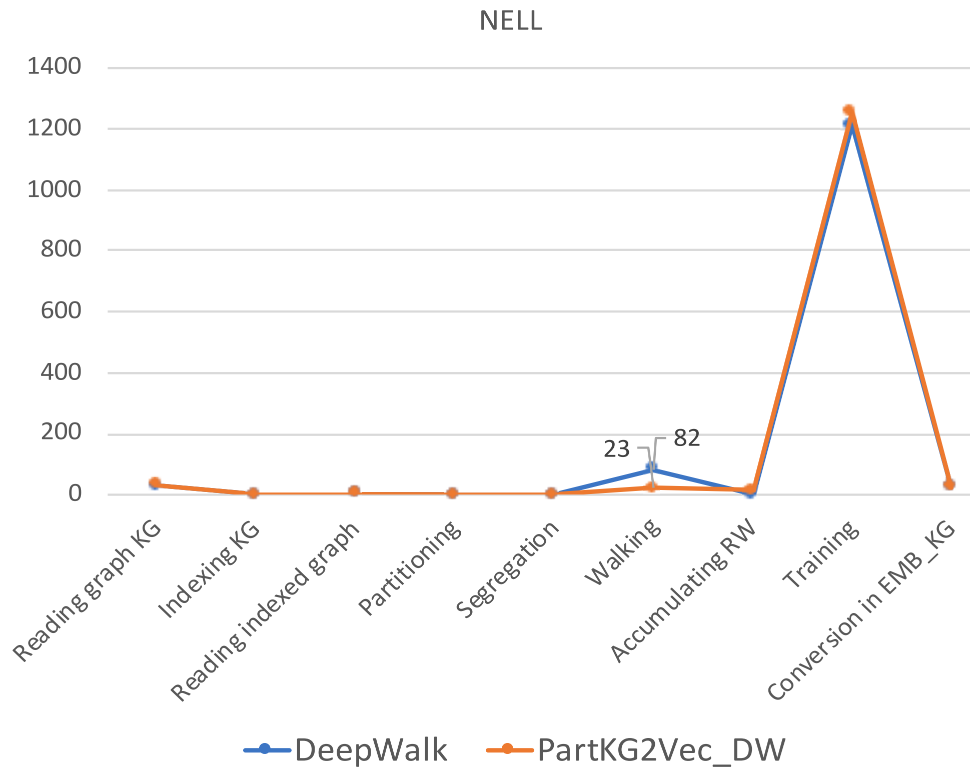


Fig. 5.7. PartKG2Vec (Deepwalk) runtime compared to Deepwalk on NELL

Similarly, Figure 5, shows that the time required by PartKG2Vec for graph preprocessing was only 20.5% of the time required by node2vec (13.5 vs. 66 seconds), on the NELL dataset. Ten iterations of random walk generation was used in both algorithms, but PartKG2Vec_N2V runs in parallel, so it took only 6.5% of time (51 vs. 794 seconds) of the time taken by node2vec on the complete graph. Learning the graph embedding takes the same time for node2vec and PartKG2Vec, because at this point both algorithms work on the similar random walk pool.

In Figure 5.6, with the results for the YAGO dataset, shows that the time required for the generation of random walks by PartKG2Vec is only 21.6% of the time used by Deepwalk (46 vs. 213 seconds), and in Figure 5.7. for NELL dataset, the time required for the random walk generation by PartKG2Vec is only 28% of the time used by Deepwalk (23 vs. 82 seconds).

Experiment 2

This experiment demonstrates the similarity of the embeddings based on the random walks generated by PartKG2Vec vs. node2vec and Deepwalk. Again, the experiment used the same two knowledge graphs (NELL and Yago39K) and produce their embeddings by PartKG2Vec vs. node2vec and Deepwalk, with varied dimensions $d \in \{128, 64, 32, 16\}$. The algorithms were executed 25 times, for each dimension. To compare the produced embeddings, the average divergence scores [72] $S_{A,d}$ were computed. Broadly speaking, a divergence score is the result of comparing a graph with the edges re-created from an embedding produced for a graph and the original graph. When comparing embeddings, a lower divergence score indicates a better embedding and, conversely, a higher divergence score means that a given embedding is not as good. Figure 5.8 shows divergence scores of the embeddings of the Yago39K dataset produced by

node2vec and PartKG2Vec_N2V (with a modified node2vec) and embeddings produced by Deepwalk and PartKG2Vec_DW (a PartKG2Vec implementation on Deepwalk). The embeddings have very similar divergence scores at every dimension. Incidentally, node2vec (and PartKG2Vec_N2V) produce better embeddings than those produced by Deepwalk and PartKG2Vec_DW. In Figure 5.9, Comparing the embeddings produced for the NELL dataset leads to similar conclusions, as the divergence scores for node2vec and PartKG2Vec_N2V and for Deepwalk and PartKG2Vec_DW are very similar.

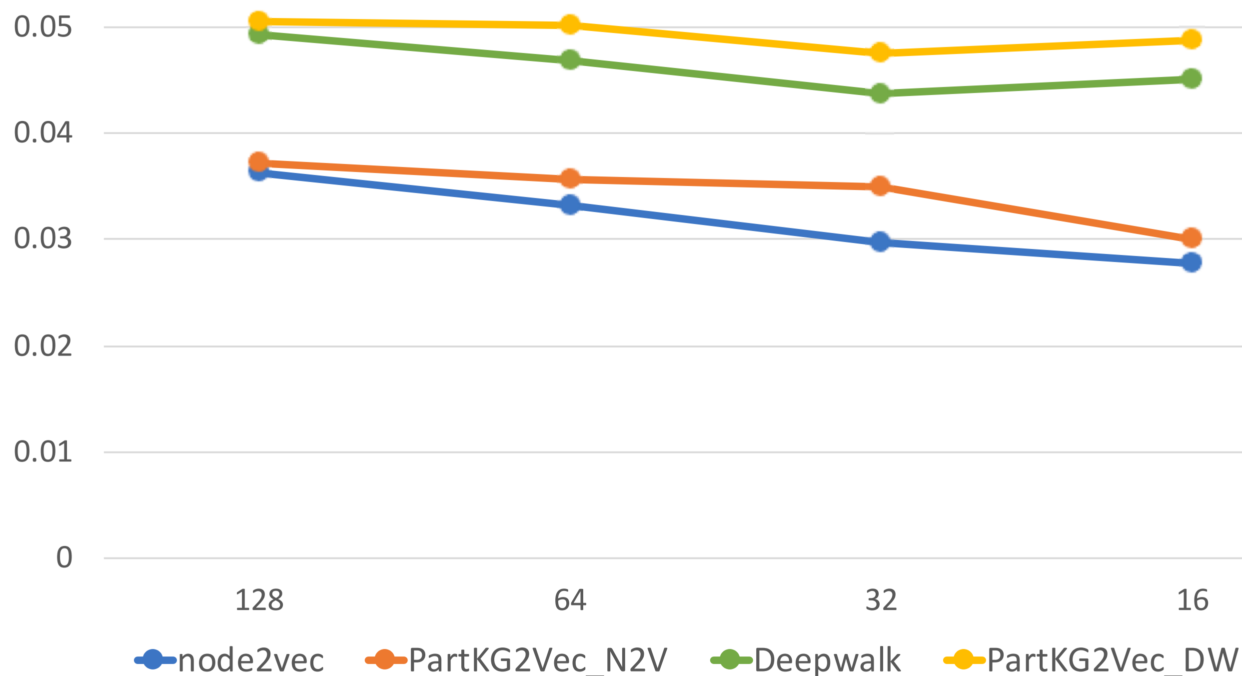


Figure 8. Average divergence scores of embeddings on Yago39K produced by node2vec and Deepwalk and their corresponding PartKG2Vec_N2V and PartKG2Vec_DW methods.

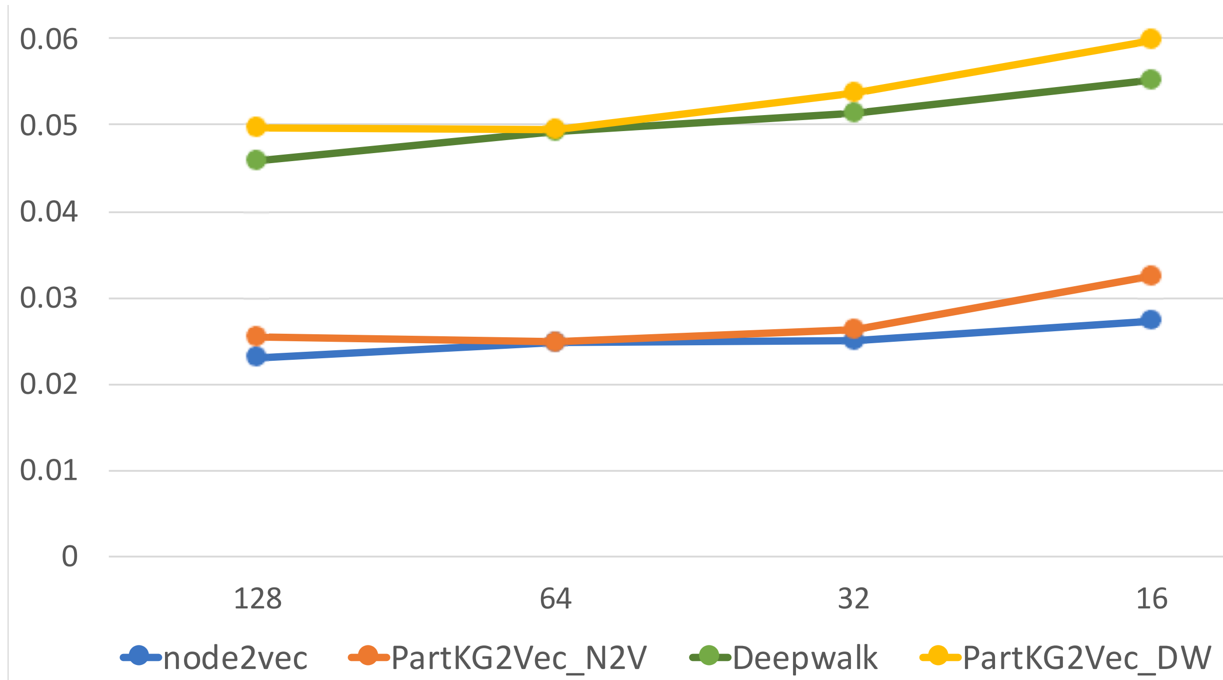


Figure 9. Average divergence scores of embeddings on NELL produced by node2vec and Deepwalk and their corresponding PartKG2Vec_N2V and PartKG2Vec_DW methods.

This demonstrates that the embeddings generated from node2vec on the original graph are very similar to those generated by PartKG2Vec_N2V and the embeddings generated from Deepwalk are similar to those from PartKG2Vec_DW.

5.6 Conclusions and Future Work

We proposed a system, PartKG2Vec, to create embeddings of partitioned knowledge graphs. The method uses modified node2vec and Deepwalk random walk algorithms to take advantage of the partitioning and perform in parallel. Our experiments showed that the embeddings produced on the original knowledge graphs were very similar to those produced by our method on the partitioned graphs. Importantly, PartKG2Vec offers significant performance improvements over

the embedding algorithms on the unpartitioned (original) knowledge graphs, which would improve the runtime of embedding very large graphs.

In the future, we intend to study other embedding algorithms utilizing different types of random walks, especially incorporating the semantics in knowledge graphs, such as metapath2vec [56] and RegPattern2Vec [73].

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This dissertation focused on different operations on partitioned knowledge graphs, such as querying and data mining, specifically considering graph embedding. The thesis introduced frameworks to tackle three problems. First, WawPart, which is a knowledge graph partitioning and query processing system. It partitions an RDF dataset based on the query workload and aims to reduce the number of distributed joins during query execution, to improve the workload run-time. The second framework, called AWAPart, is the extension of WawPart and is as an adaptive re-partitioning system to address the changes in the query workload. We investigated the adaptability of partitioning in response to (1) the changes in the frequency of queries in the workload (AWAPart assumes that all queries are executed with the same frequency) and (2) the changes in the composition of the workload (queries may be eliminated and/or new queries may be added to the workload). AWAPart adapts the partition of the graph according to changes in the query workload. It also, focuses on reducing the number of distributed joins during query execution, that eventually leads to a reduced run-time for the queries in the workload. The last framework, PartKG2Vec, focuses on creating embeddings of partitioned Knowledge Graphs. The method uses a modified node2vec and a modified DeepWalk, both of which are based on a form of graph random walk algorithms, to take advantage of the partitioning and perform the random walks in parallel and produce partial random walks. Hence, an additional step of merging the partial walks is required to create the complete random walks. Our experiments showed that the

embeddings produced on the original knowledge graphs are very similar to those produced by our method on the partitioned graphs. Importantly, PartKG2Vec offers significant performance improvements over the embedding algorithms on the unpartitioned (original) knowledge graphs, which improves the runtime of embedding very large graphs.

In the future, in the context of querying on partitioned knowledge graph it would be interesting to modify the framework of WawPart and AWAPart with other forms of knowledge graph representation such as graph databases, for example, Neo4j. Also, it would be interesting to investigate the impact of partitioning on evolving knowledge graphs in terms. Furthermore, more research should be conducted on how the adaptive partitioning handles the evolving datasets along with the evolving workload queries. Finally, additional studies should be devoted to extending PartKG2Vec to other embedding algorithms utilizing different types of random walks, especially incorporating the semantics in knowledge graphs, such as metapath2vec and RegPattern2Vec.

REFERENCES

[1] RDF Working Group. "Rdf - semantic web standards." https://www.w3.org/RDF/ (accessed July 1, 2021.

[2] World Wide Web Consortium. "Ontologies - w3c." https://www.w3.org/standards/semanticweb/ontology (accessed July 1,2021).

[3] D. Brickley and L. Miller, "FOAF vocabulary specification 0.91," ed: Citeseer, 2010.

[4] World Wide Web Consortium. "Rdfs - semantic web standards." https://www.w3.org/2001/sw/wiki/RDFS (accessed July 1,2021.

[5] "Dcmi: Home." http://dublincore.org/ (accessed.

[6] World Wide Web Consortium. "Owl - semantic web standards." https://www.w3.org/OWL/ (accessed July 1,2021.

[7] World Wide Web Consortium. "Sparql query language for rdf." https://www.w3.org/TR/rdf-sparql-query/ (accessed July 1,2021.

[8] World Wide Web Consortium. "Sparql 1.1 federated query." https://www.w3.org/TR/sparql11-federated-query/ (accessed July 1,2021.

[9] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. Ngonga Ngomo, "A fine-grained evaluation of SPARQL endpoint federation systems," *Semantic Web,* vol. 7, no. 5, pp. 493-518, 2016.

[10] "Redland rdf libraries." http://librdf.org/ (accessed July 1,2021.

[11] B. McBride, "Jena: A semantic web toolkit," *IEEE Internet computing,* vol. 6, no. 6, pp. 55-59, 2002.

[12] A. Jena, "Semantic web framework for java," 2007.

[13] O. Erling and I. Mikhailov, "RDF Support in the Virtuoso DBMS," in *Networked Knowledge-Networked Media*: Springer, 2009, pp. 7-24.

[14] OpenLink Software Documentation Team. "Openlink Virtuoso." https://virtuoso.openlinksw.com/ (accessed July 1,2021.

[15] A. Priyadarshi and K. J. Kochut, "WawPart: Workload-Aware Partitioning of Knowledge Graphs," Cham, 2021: Springer International Publishing, in Advances and Trends in Artificial Intelligence. Artificial Intelligence Practices, pp. 383-395.

[16] Y. Sun and J. Han, "Mining heterogeneous information networks: a structural analysis approach," *Acm Sigkdd Explorations Newsletter,* vol. 14, no. 2, pp. 20-28, 2013.

[17] "Neo4j." http://www.neo4j.org (accessed.

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*, 1974: ACM, pp. 47-63.

[19] W. Donath and A. Hoffman, "Algorithms for partitioning of graphs and computer logic based on eigenvectors of connections matrices," *IBM Technical Disclosure Bulletin,* vol. 15, 1972.

[20] J. R. Gilbert, G. L. Miller, and S.-H. Teng, "Geometric mesh partitioning: Implementation and experiments," *SIAM Journal on Scientific Computing,* vol. 19, no. 6, pp. 2091-2110, 1998.

[21] S. T. Barnard and H. D. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and experience,* vol. 6, no. 2, pp. 101-117, 1994.

[22] B. Hendrickson and R. Leland, "A multi-level algorithm for partitioning graphs," 1995.

[23] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing,* vol. 20, no. 1, pp. 359-392, 1998.

[24] G. Karypis, "METIS and ParMETIS," in *Encyclopedia of parallel computing*: Springer, 2011, pp. 1117-1124.

[25] N. Xu, L. Chen, and B. Cui, "LogGP: a log-based dynamic graph partitioning method," *Proceedings of the VLDB Endowment,* vol. 7, no. 14, pp. 1917-1928, 2014.

[26] G. Malewicz *et al.*, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010: ACM, pp. 135-146.

[27] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: distributed graph-parallel computation on natural graphs," in *OSDI*, 2012, vol. 12, no. 1, p. 2.

[28] "Apache Giraph. https://github.com/apache/giraph/."

[29] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, 2009: IEEE, pp. 229-238.

[30] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041,* 2014.

[31] S. Yang, X. Yan, B. Zong, and A. Khan, "Towards effective partition management for large graphs," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012: ACM, pp. 517-528.

[32] M. Hammoud, D. A. Rabbou, R. Nouri, S.-M.-R. Beheshti, and S. Sakr, "DREAM: distributed RDF engine with adaptive query planner and minimal communication," *Proceedings of the VLDB Endowment,* vol. 8, no. 6, pp. 654-665, 2015.

[33] K. Hose and R. Schenkel, "WARP: Workload-aware replication and partitioning for RDF," in *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, 2013: IEEE, pp. 1-6.

[34] L. Galárraga, K. Hose, and R. Schenkel, "Partout: a distributed engine for efficient RDF processing," in *Proceedings of the 23rd International Conference on World Wide Web*, 2014: ACM, pp. 267-268.

[35] R. Harbi, I. Abdelaziz, P. Kalnis, N. Mamoulis, Y. Ebrahim, and M. Sahli, "Accelerating SPARQL queries by exploiting hash-based locality and adaptive partitioning," *The VLDB Journal,* vol. 25, no. 3, pp. 355-380, 2016.

[36] X. Guo, H. Gao, and Z. Zou, "WISE: Workload-Aware Partitioning for RDF Systems," *Big Data Research,* vol. 22, p. 100161, 2020.

[37] T. Neumann and G. Weikum, "RDF-3X: a RISC-style engine for RDF," *Proceedings of the VLDB Endowment,* vol. 1, no. 1, pp. 647-659, 2008.

[38] A. Lucene, "Apache Lucene-Overview," *Internet: http://lucene. apache. org/iava/docs/[Jan. 15, 2009],* 2010.

[39] A. Białecki, R. Muir, G. Ingersoll, and L. Imagination, "Apache lucene 4," in *SIGIR 2012 workshop on open source information retrieval*, 2012, p. 17.

[40] World Wide Web Consortium. "Sparql 1.1 federated query." https://www.w3.org/TR/sparql11-federated-query/ (accessed.

[41] OpenLink Software Documentation Team. "Openlink Virtuoso." https://virtuoso.openlinksw.com/ (accessed.

[42] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Journal of Web Semantics,* vol. 3, no. 2-3, pp. 158-182, 2005.

[43] C. Bizer and A. Schultz, "Berlin SPARQL benchmark (BSBM) specification-v2. 0," ed: September, 2008.

[44] A. Priyadarshi and K. J. Kochut, "AWAPart: Adaptive Workload-Aware Partitioning Knowledge Graphs," in *SEMAPRO 2021, The Fifteenth International Conference on Advances in Semantic Processing*, Barcelona, Spain, 2021: Thinkmind Digital Library, 2021, pp. 12-17.

[45] "Neo4j." http://www.neo4j.org (accessed July 1,2021.

[46] C. Basca and A. Bernstein, "x-Avalanche: Optimisation Techniques for Large Scale Federated SPARQL Query Processing."

[47] A. Priyadarshi and K. J. Kochut, "PartKG2Vec: Embedding of Partitioned Knowledge Graphs," in *International Conference on Knowledge Science, Engineering and Management*, 2022: Springer, pp. 359-370.

[48] P. Radivojac *et al.*, "A large-scale evaluation of computational protein function prediction," *Nature methods,* vol. 10, no. 3, pp. 221-227, 2013.

[49] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology,* vol. 58, no. 7, pp. 1019-1031, 2007.

[50] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani, "Global protein function prediction from protein-protein interaction networks," *Nature biotechnology,* vol. 21, no. 6, pp. 697-700, 2003.

[51] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011, pp. 635-644.

[52] M. A. Cox and T. F. Cox, "Multidimensional scaling," in *Handbook of data visualization*: Springer, 2008, pp. 315-347.

[53] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Nips*, 2001, vol. 14, no. 14, pp. 585-591.

[54] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science,* vol. 290, no. 5500, pp. 2319-2323, 2000.

[55] S. A. Myers, A. Sharma, P. Gupta, and J. Lin, "Information network or social network? The structure of the Twitter follow graph," in *Proceedings of the 23rd International Conference on World Wide Web*, 2014, pp. 493-498.

[56] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 135-144.

[57] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111-3119.

[58] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701-710.

[59] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855-864.

[60] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067-1077.

[61] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," in *Proceedings of the AAAI conference on artificial intelligence*, 2018, vol. 32, no. 1.

[62] A. Lerer *et al.*, "Pytorch-biggraph: A large scale graph embedding system," *Proceedings of Machine Learning and Systems,* vol. 1, pp. 120-131, 2019.

[63] J. Liang, S. Gurukar, and S. Parthasarathy, "Mile: A multi-level framework for scalable graph embedding," *arXiv preprint arXiv:1802.09612,* 2018.

[64] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Accurate, efficient and scalable graph embedding," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019: IEEE, pp. 462-471.

[65] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907,* 2016.

[66] G. Karypis and V. Kumar, "METIS--unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.

[67] "hMetis: a hypergraph partitioning package, http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview." (accessed.

[68] X. Lv, L. Hou, J. Li, and Z. Liu, "Differentiating concepts and instances for knowledge graph embedding," *arXiv preprint arXiv:1811.04588,* 2018.

[69] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*, 2007: ACM, pp. 697-706.

[70] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A large ontology from wikipedia and wordnet," *Journal of Web Semantics,* vol. 6, no. 3, pp. 203-217, 2008.

[71] G. Wan, B. Du, S. Pan, and G. Haffari, "Reinforcement learning based meta-path discovery in large-scale heterogeneous information networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, vol. 34, no. 04, pp. 6094-6101.

[72] A. Dehghan-Kooshkghazi, B. Kamiński, Ł. Kraiński, P. Prałat, and F. Théberge, "Evaluating Node embeddings of complex networks," *arXiv preprint arXiv:2102.08275,* 2021.

[73] A. Keshavarzi, N. Kannan, and K. Kochut, "RegPattern2Vec: Link Prediction in Knowledge Graphs," in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 2021: IEEE, pp. 1-7.

APPENDIX A

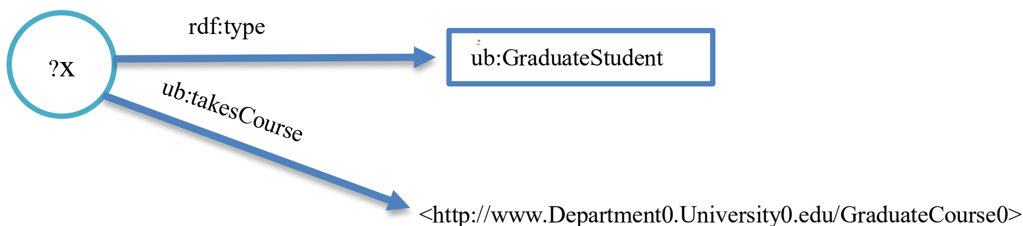
ORIGINAL 14 LUBM QUERY

Following are the 14 Lehigh University Benchmark LUBM Queries.

Query 1:

This query bears large input and high selectivity. It queries about just one class and One property and does not assume any hierarchy information or inference.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from < LUBM_Synthetic_Data >
WHERE
{
	?X rdf:type ub:GraduateStudent .
	?X ub:takesCourse <http://www.Department6.University29.edu/GraduateCourse41>
}
```

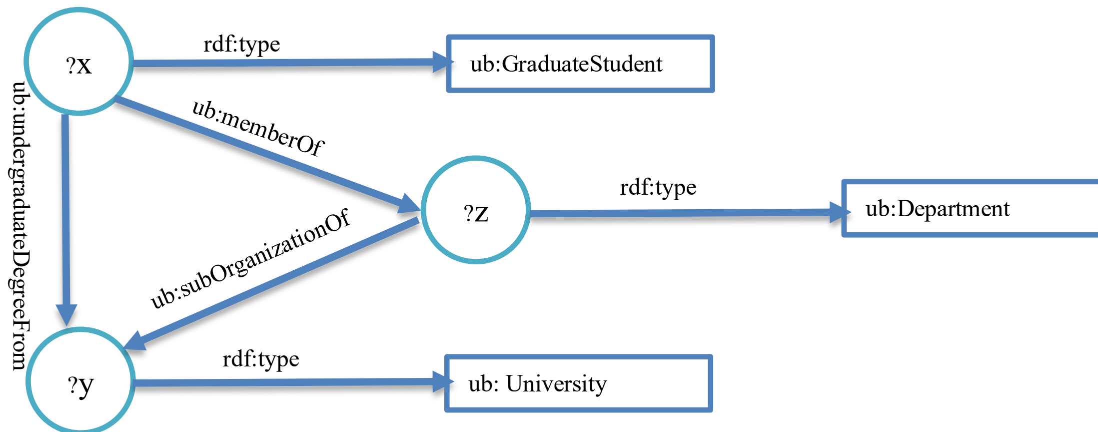


Query 2:

This query increases in complexity: 3 classes and 3 properties are involved. Additionally, There is a triangular pattern of relationships between the objects involved.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z FROM <LUBM_Synthetic_Data>
WHERE
{
	?X rdf:type ub:GraduateStudent .
	?Y rdf:type ub:University .
	?Z rdf:type ub:Department .
```

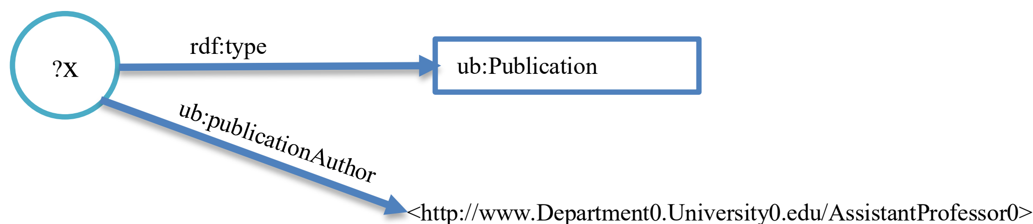
```
        ?X ub:memberOf ?Z .
        ?Z ub:subOrganizationOf ?Y .
        ?X ub:undergraduateDegreeFrom ?Y
}
```



Query 3:

This query is similar to Query 1 but class Publication has a wide hierarchy.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Publication .
  ?X ub:publicationAuthor<http://www.Department0.University0.edu/AssistantProfessor0>
}
```

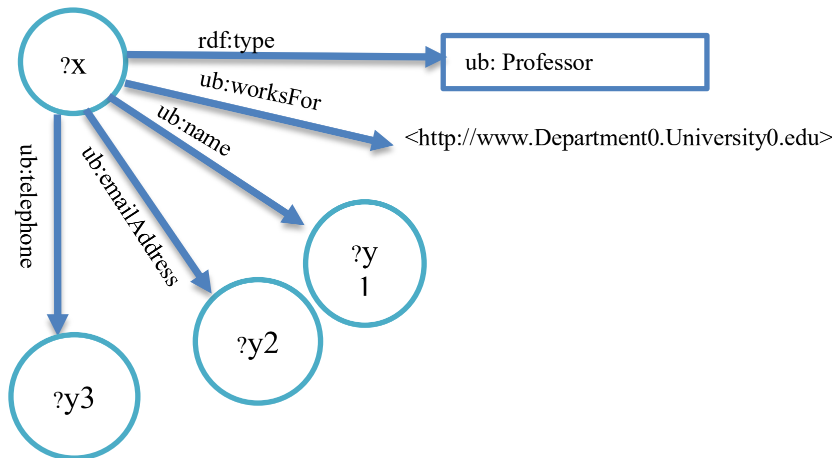


Query 4:

This query has small input and high selectivity. It assumes subClassOf relationship between Professor and its subclasses. Class Professor has a wide hierarchy. Another feature is that it queries about multiple properties of a single class.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y1, ?Y2, ?Y3 FROM <LUBM50U_PART>
```

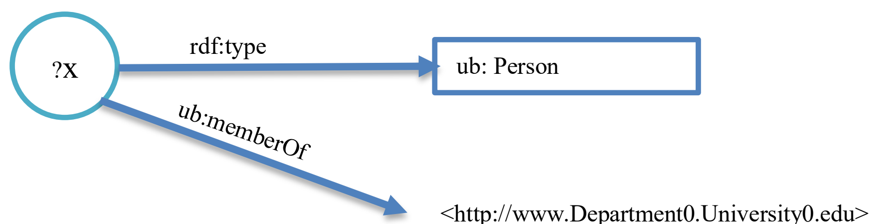
```
WHERE
{
  ?X rdf:type ub:Professor .
  ?X ub:worksFor <http://www.Department0.University0.edu> .
  ?X ub:name ?Y1 .
  ?X ub:emailAddress ?Y2 .
  ?X ub:telephone ?Y3
}
```



Query 5:

This query assumes subClassOf relationship between Person and its subclasses and subPropertyOf relationship between memberOf and its subproperties. Moreover, class Person features a deep and wide hierarchy.

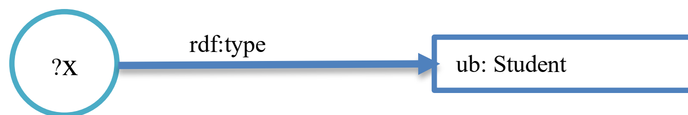
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Person .
  ?X ub:memberOf <http://www.Department0.University0.edu>
}
```



Query 6:

This query queries about only one class. But it assumes both the explicit subClassOf relationship between UndergraduateStudent and Student and the implicit one between GraduateStudent and Student. In addition, it has large input and low selectivity.

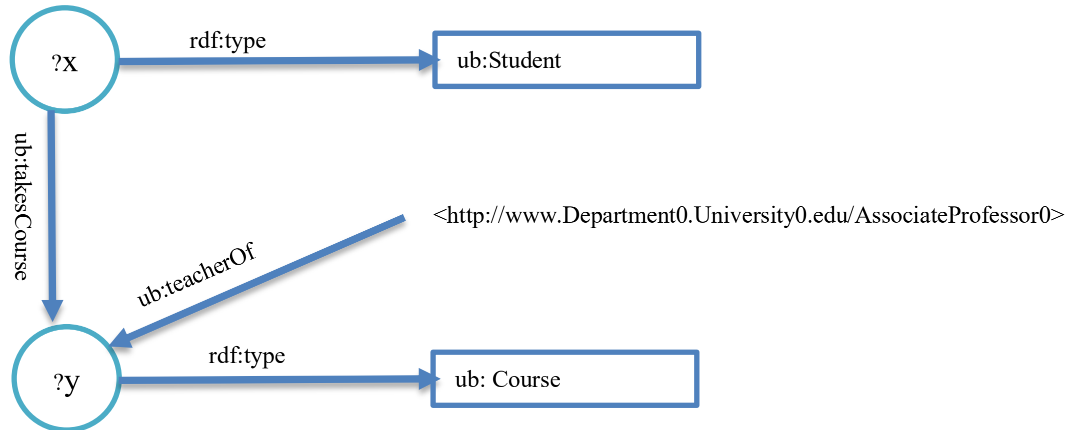
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X  FROM <LUBM_Synthetic_Data>
WHERE {
?X rdf:type ub:Student
}
```



Query 7:

This query is similar to Query 6 in terms of class Student, but it increases in the Number of classes and properties and its selectivity is high.

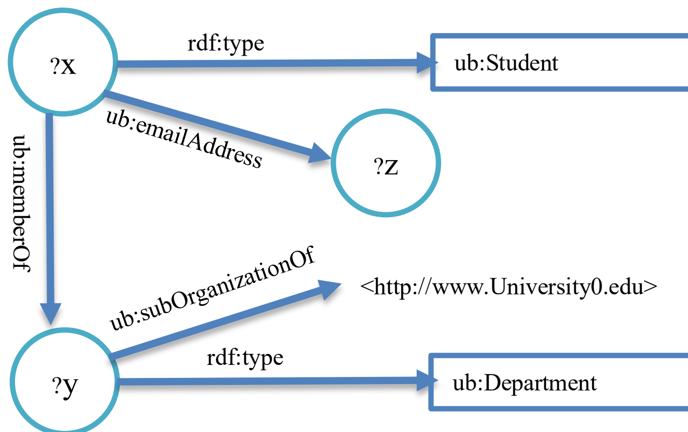
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Course .
  ?X ub:takesCourse ?Y .
  <http://www.Department0.University0.edu/AssociateProfessor0> ub:teacherOf ?Y
}
```



Query 8:

This query is furthermore complex than Query 7 by including one more property.

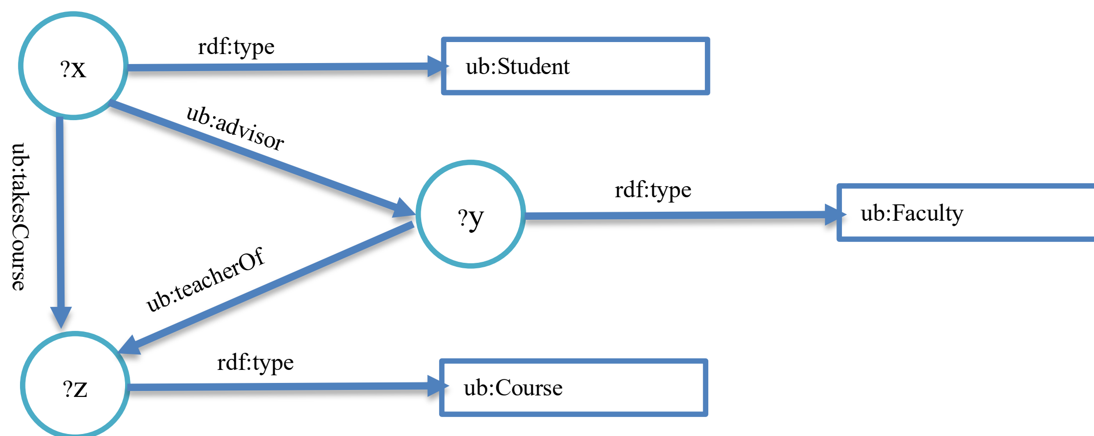
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Department .
  ?X ub:memberOf ?Y .
  ?Y ub:subOrganizationOf <http://www.University0.edu> .
  ?X ub:emailAddress ?Z
}
```



Query 9:

Besides the aforementioned features of class Student and the wide hierarchy of class Faculty, like Query 2, this query is characterized by the most classes and properties in the query set and there is a triangular pattern of relationships.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?Z rdf:type ub:Course .
  ?X ub:advisor ?Y .
  ?Y ub:teacherOf ?Z .
  ?X ub:takesCourse ?Z
}
```

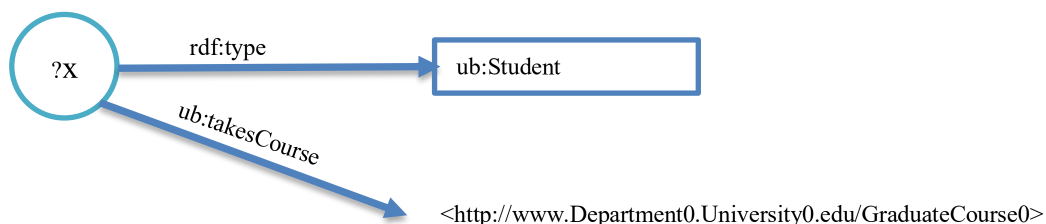


Query 10:

This query differs from Query 6, 7, 8 and 9 in that it only requires the (implicit) subClassOf relationship between GraduateStudent and Student, i.e., subClassOf relationship between UndergraduateStudent and Student does not add to the results.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Student .
```

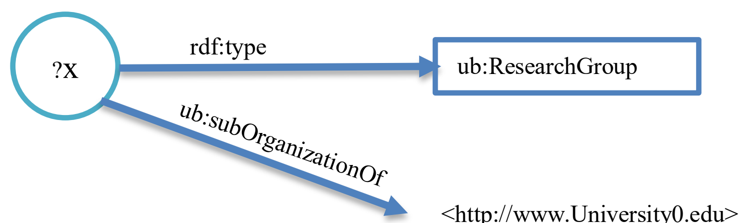
```
  ?X ub:takesCourse <http://www.Department0.University0.edu/GraduateCourse0>
}
```



Query 11:

Query 11, 12 and 13 are intended to verify the presence of certain OWL reasoning capabilities in the system. In this query, property subOrganizationOf is defined as transitive. Since in the benchmark data, instances of ResearchGroup are stated as a sub-organization of a Department individual and the later suborganization of a University individual, inference about the subOrgnizationOf relationship between instances of ResearchGroup and University is required to answer this query. Additionally, its input is small.

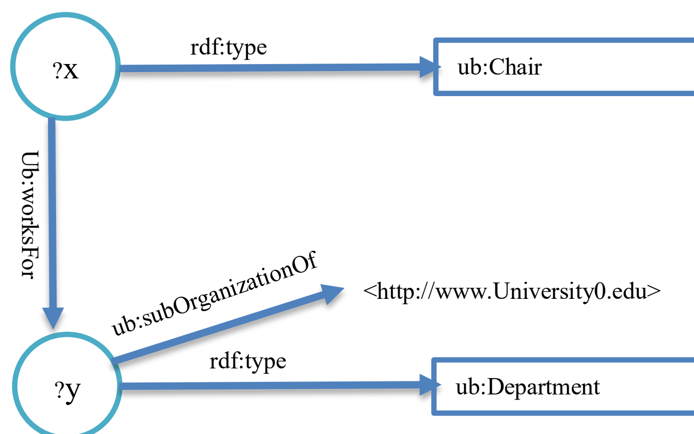
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:ResearchGroup .
  ?X ub:subOrganizationOf <http://www.University0.edu>
}
```



Query 12:

The benchmark data do not produce any instances of class Chair. Instead, each Department individual is linked to the chair professor of that department by property headOf. Hence this query requires realization, i.e., inference that professor is an instance of class Chair because he or she is the head of a department. Input of this query is small as well.

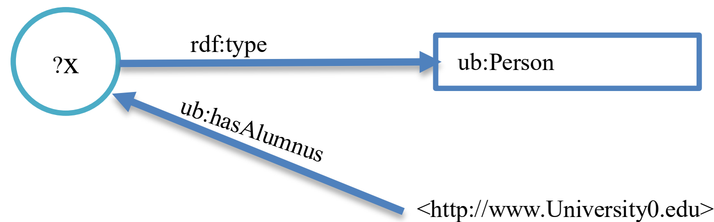
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Chair .
  ?Y rdf:type ub:Department .
  ?X ub:worksFor ?Y .
  ?Y ub:subOrganizationOf <http://www.University0.edu>
}
```



Query 13:

Property hasAlumnus is defined in the benchmark ontology as the inverse of property degreeFrom, which has three subproperties: undergraduateDegreeFrom, mastersDegreeFrom, and doctoralDegreeFrom. The benchmark data state a person as an alumnus of a university using one of these three subproperties instead of hasAlumnus. Therefore, this query assumes subPropertyOf relationships between degreeFrom and its subproperties, and also requires inference about inverseOf.

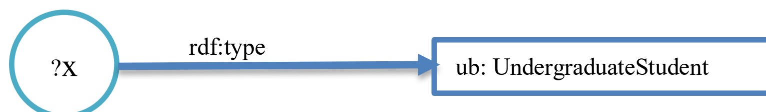
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:Person .
  <http://www.University0.edu> ub:hasAlumnus ?X
}
```



Query 14:

This query is the simplest in the test set. This query represents those with large input and low selectivity and does not assume any hierarchy information or inference.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X FROM <LUBM_Synthetic_Data>
WHERE
{
  ?X rdf:type ub:UndergraduateStudent
}
```



APPENDIX B

14 LUBM FEDERATED QUERY BASED ON WAWPART

Following are the 14 Lehigh University Benchmark (LUBM) Federated Queries based on WawPart partition.

Query 1: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
	?X rdf:type ub:GraduateStudent .
	SERVICE <http://172.19.48.181:8890/sparql> {?X ub:takesCourse
<http://www.Department6.University29.edu/GraduateCourse41>}
}
```

Query 2: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z from <LUBM_PART>
WHERE
{
	?X rdf:type ub:GraduateStudent .
	?Y rdf:type ub:University .
	?Z rdf:type ub:Department .
	?X ub:memberOf ?Z .
	?Z ub:subOrganizationOf ?Y .
	?X ub:undergraduateDegreeFrom ?Y
}
```

Query 3: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
```

```
{
  ?X rdf:type ub:Publication .
  ?X ub:publicationAuthor
<http://www.Department0.University0.edu/AssistantProfessor0>
}
```

Query 4: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y1, ?Y2, ?Y3 from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:Professor .
  ?X ub:worksFor <http://www.Department0.University0.edu> .
  ?X ub:name ?Y1 .
  ?X ub:emailAddress ?Y2 .
  ?X ub:telephone ?Y3
}
```

Query 5: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:Person .
  SERVICE <http://172.19.48.183:8890/sparql> {?X ub:memberOf
<http://www.Department0.University0.edu>}
}
```

Query 6: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
       ?X rdf:type ub:Student
}
```

Query 7: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y from <LUBM50U_PART>
WHERE
{
```

```
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Course .
  ?X ub:takesCourse ?Y .
  <http://www.Department0.University0.edu/AssociateProfessor0> ub:teacherOf ?Y
}
```

Query 8: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z from <LUBM_PART>
WHERE
{
  SERVICE <http://172.19.48.181:8890/sparql>{?X rdf:type ub:Student .}
  ?Y rdf:type ub:Department .
  ?X ub:memberOf ?Y .
  ?Y ub:subOrganizationOf <http://www.University0.edu> .
  ?X ub:emailAddress ?Z
}
```

Query 9: runs on cluster 4: http://172.19.48.184:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?Z rdf:type ub:Course .
  ?X ub:advisor ?Y .
  ?Y ub:teacherOf ?Z .
  ?X ub:takesCourse ?Z
}
```

Query 10: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:Student .
  ?X ub:takesCourse <http://www.Department0.University0.edu/GraduateCourse0>
}
```

Query 11: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
    ?X rdf:type ub:ResearchGroup .
    ?X ub:subOrganizationOf <http://www.University0.edu>
}
```

Query 12: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:Chair .
  ?Y rdf:type ub:Department .
  ?X ub:worksFor ?Y .
  ?Y ub:subOrganizationOf <http://www.University0.edu>
}
```

Query 13: runs on cluster 4: http://172.19.48.184:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:Person .
  <http://www.University0.edu> ub:hasAlumnus ?X
}
```

Query 14: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_PART>
WHERE
{
  ?X rdf:type ub:UndergraduateStudent
}
```

APPENDIX C

14 LUBM FEDERATED QUERY BASED ON RANDOM PARTITION

Following are the 14 Lehigh University Benchmark (LUBM) Federated Queries based on random partition.

Query 1: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE
{
	?X rdf:type ub:GraduateStudent .
	?X ub:takesCourse <http://www.Department6.University29.edu/GraduateCourse41>
}
```

Query 2: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z from <LUBM_RANDOM _PART>
WHERE
{
	SERVICE <http://172.19.48.181:8890/sparql> {?X rdf:type ub:GraduateStudent .}
	SERVICE <http://172.19.48.181:8890/sparql> {?Y rdf:type ub:University .}
	?Z rdf:type ub:Department .
	?X ub:memberOf ?Z .
	SERVICE <http://172.19.48.181:8890/sparql> {?Z ub:subOrganizationOf ?Y .}
	?X ub:undergraduateDegreeFrom ?Y
}
```

Query 3: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM _PART>
WHERE
{
```

```
  ?X rdf:type ub:Publication .
  ?X ub:publicationAuthor
<http://www.Department0.University0.edu/AssistantProfessor0>
}
```

Query 4: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y1, ?Y2, ?Y3 from <LUBM_RANDOM_PART>
WHERE
{
      SERVICE <http://172.19.48.183:8890/sparql> {?X rdf:type ub:Professor .}
      SERVICE <http://172.19.48.181:8890/sparql> {?X ub:worksFor
<http://www.Department0.University0.edu> .}
      ?X ub:name ?Y1 .
      SERVICE <http://172.19.48.181:8890/sparql> { ?X ub:emailAddress ?Y2 .}
      SERVICE <http://172.19.48.183:8890/sparql> {?X ub:telephone ?Y3}
}
```

Query 5: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE
{
  ?X rdf:type ub:Person .
  SERVICE <http://172.19.48.182:8890/sparql>{?X ub:memberOf
<http://www.Department0.University0.edu>}
}
```

Query 6: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE
{
      ?X rdf:type ub:Student
}
```

Query 7: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y from <LUBM5_RANDOM_PART>
WHERE
{
```

```
  ?X rdf:type ub:Student .
  SERVICE <http://172.19.48.183:8890/sparql>{?Y rdf:type ub:Course .}
  SERVICE <http://172.19.48.181:8890/sparql>{?X ub:takesCourse ?Y .}
  SERVICE
<http://172.19.48.181:8890/sparql>{<http://www.Department0.University0.edu/AssociateP
rofessor0> ub:teacherOf ?Y}
}
```

Query 8: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z from <LUBM_RANDOM_PART>
WHERE
{
 ?X rdf:type ub:Student .
 ?Y rdf:type ub:Department .
 ?X ub:memberOf ?Y .
  SERVICE <http://172.19.48.181:8890/sparql>{?Y ub:subOrganizationOf
<http://www.University0.edu> .}
  SERVICE <http://172.19.48.181:8890/sparql>{?X ub:emailAddress ?Z}
}
```

Query 9: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y, ?Z from <LUBM_RANDOM_PART>
WHERE
{
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  SERVICE <http://172.19.48.183:8890/sparql>{?Z rdf:type ub:Course .}
  SERVICE <http://172.19.48.183:8890/sparql>{?X ub:advisor ?Y .}
  SERVICE <http://172.19.48.181:8890/sparql>{?Y ub:teacherOf ?Z .}
  SERVICE <http://172.19.48.181:8890/sparql>{?X ub:takesCourse ?Z}
}
```

Query 10: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE
{
  ?X rdf:type ub:Student .
  SERVICE <http://172.19.48.181:8890/sparql>{?X ub:takesCourse
<http://www.Department0.University0.edu/GraduateCourse0>}
}
```

Query 11: runs on cluster 2: http://172.19.48.182:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE
{
  ?X rdf:type ub:ResearchGroup .
  SERVICE <http://172.19.48.182:8890/sparql>{?X ub:subOrganizationOf
<http://www.University0.edu>}
}
```

Query 12: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X, ?Y from <LUBM_RANDOM_PART>
WHERE
{
  ?X rdf:type ub:Chair .
  SERVICE <http://172.19.48.182:8890/sparql> {?Y rdf:type ub:Department .}
  ?X ub:worksFor ?Y .
  ?Y ub:subOrganizationOf <http://www.University0.edu>
}
```

Query 13: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE

    SERVICE <http://172.19.48.183:8890/sparql> {?X rdf:type ub:Person .}
  <http://www.University0.edu> ub:hasAlumnus ?X
}
```

Query 14: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X from <LUBM_RANDOM_PART>
WHERE
{
  ?X rdf:type ub:UndergraduateStudent
}
```

APPENDIX D

ORIGINAL 12 BSBM QUERY

Following is the Original 12 Berlin Benchmark (BSBM) Queries.

Query 1:

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label
WHERE {
    ?product rdfs:label ?label .
    ?product a bsbm-inst:ProductType1 .
    ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
    ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
    ?product bsbm:productPropertyNumeric1 ?value1 .
      FILTER (?value1 > 400)
}
ORDER BY ?label
LIMIT 10
```

Query 2:

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?label ?comment ?producer ?productFeature ?propertyTextual1 ?propertyTextual2
?propertyTextual3 ?propertyNumeric1 ?propertyNumeric2 ?propertyTextual4
?propertyTextual5 ?propertyNumeric4
WHERE {
  bsbm-inst-dataFromProducer2:Product78 rdfs:label ?label .
  bsbm-inst-dataFromProducer2:Product78 rdfs:comment ?comment .
  bsbm-inst-dataFromProducer2:Product78 bsbm:producer ?p .
  ?p rdfs:label ?producer .
  bsbm-inst-dataFromProducer2:Product78 dc:publisher ?p .
  bsbm-inst-dataFromProducer2:Product78 bsbm:productFeature ?f .
  ?f rdfs:label ?productFeature .
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual1 ?propertyTextual1.
```

```
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual2 ?propertyTextual2.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual3 ?propertyTextual3.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric1 ?propertyNumeric1.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric2 ?propertyNumeric2.
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual4 ?propertyTextual4
}
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual5 ?propertyTextual5
}
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric4 ?propertyNumeric4
}
}
```

Query 3:

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?product ?label
WHERE {
    ?product rdfs:label ?label .
    ?product a bsbm-inst:ProductType1 .
       ?product bsbm:productFeature bsbm-inst:ProductType1 .
       ?product bsbm:productPropertyNumeric1 ?p1 .
       FILTER ( ?p1 > 10 )
       ?product bsbm:productPropertyNumeric3 ?p3 .
       FILTER (?p3 < 400 )
    OPTIONAL {
        ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
        ?product rdfs:label ?testVar }
    FILTER (!bound(?testVar))
}
ORDER BY ?label
LIMIT 10
```

Query 4:

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label ?propertyTextual
WHERE {
    {
       ?product rdfs:label ?label .
       ?product rdf:type bsbm-inst:ProductType1 .
       ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
```

```
        ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
        ?product bsbm:productPropertyTextual1 ?propertyTextual .
        ?product bsbm:productPropertyNumeric1 ?p1 .
        FILTER ( ?p1 > 200 )
     } UNION {
        ?product rdfs:label ?label .
        ?product rdf:type bsbm-inst:ProductType2 .
        ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
        ?product bsbm:productFeature bsbm-inst:ProductFeature3 .
        ?product bsbm:productPropertyTextual1 ?propertyTextual .
        ?product bsbm:productPropertyNumeric2 ?p2 .
        FILTER ( ?p2> 400 )
     }
}
ORDER BY ?label
OFFSET 5
LIMIT 10
```

Query 5:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?productLabel
WHERE {
   ?product rdfs:label ?productLabel .
  FILTER bsbm-inst-dataFromProducer2:Product78 != ?product)
   bsbm-inst-dataFromProducer2:Product78 bsbm:productFeature ?prodFeature .
   ?product bsbm:productFeature ?prodFeature .
   bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric1 ?origProperty1 .
   ?product bsbm:productPropertyNumeric1 ?simProperty1 .
  FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 -
120))
   bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric2 ?origProperty2 .
   ?product bsbm:productPropertyNumeric2 ?simProperty2 .
  FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 -
170))
}
ORDER BY ?productLabel
LIMIT 5
```

Query 6:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT ?product ?label
WHERE {
    ?product rdfs:label ?label .
    ?product rdf:type bsbm:Product .
    FILTER regex(?label, "trackable")
}
```

Query 7:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?productLabel ?offer ?price ?vendor ?vendorTitle ?review ?revTitle ?reviewer
?revName ?rating1 ?rating2
WHERE {
       bsbm-inst-dataFromProducer2:Product78 rdfs:label ?productLabel .
    OPTIONAL {
        ?offer bsbm:product bsbm-inst-dataFromProducer2:Product78 .
        ?offer bsbm:price ?price .
        ?offer bsbm:vendor ?vendor .
        ?vendor rdfs:label ?vendorTitle .
        ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#DE> .
        ?offer dc:publisher ?vendor .
        ?offer bsbm:validTo ?date .
        FILTER (?date > 2000-06-20 )
    }
    OPTIONAL {
       ?review bsbm:reviewFor bsbm-inst-dataFromProducer2:Product79 .
       ?review rev:reviewer ?reviewer .
       ?reviewer foaf:name ?revName .
       ?review dc:title ?revTitle .
    OPTIONAL { ?review bsbm:rating1 ?rating1 . }
    OPTIONAL { ?review bsbm:rating2 ?rating2 . }
    }
}
```

Query 8:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?title ?text ?reviewDate ?reviewer ?reviewerName ?rating1 ?rating2 ?rating3
?rating4
WHERE {
	?review bsbm:reviewFor bsbm-inst-dataFromProducer2:Product78 .
	?review dc:title ?title .
	?review rev:text ?text .
	FILTER langMatches( lang(?text), "EN" )
	?review bsbm:reviewDate ?reviewDate .
	?review rev:reviewer ?reviewer .
	?reviewer foaf:name ?reviewerName .
	OPTIONAL { ?review bsbm:rating1 ?rating1 . }
	OPTIONAL { ?review bsbm:rating2 ?rating2 . }
	OPTIONAL { ?review bsbm:rating3 ?rating3 . }
	OPTIONAL { ?review bsbm:rating4 ?rating4 . }
}
ORDER BY DESC(?reviewDate)
LIMIT 20
```

Query 9:

```
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX bsbm-inst-dataFromRatingSite1: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/>

DESCRIBE ?x
WHERE { bsbm-inst-dataFromRatingSite1:Review192 rev:reviewer ?x }
```

Query 10:

```
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>


SELECT DISTINCT ?offer ?price
WHERE {
	?offer bsbm:product bsbm-inst-dataFromProducer2:Product78 .
	?offer bsbm:vendor ?vendor .
	 ?offer dc:publisher ?vendor .
	?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#US> .
	?offer bsbm:deliveryDays ?deliveryDays .
	FILTER (?deliveryDays <= 3)
	?offer bsbm:price ?price .
    ?offer bsbm:validTo ?date .
```

```
    FILTER (?date > 2000-06-20 )
}
ORDER BY xsd:double(str(?price))
LIMIT 10
```

Query 12:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-export: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/vocabulary/export/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bsbm-inst-dataFromVendor1: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/>

CONSTRUCT {
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:product ?productURI .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:productlabel ?productlabel .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:vendor ?vendorname .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:vendorhomepage ?vendorhomepage .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:offerURL ?offerURL .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:price ?price .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:deliveryDays ?deliveryDays .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:validuntil ?validTo }
WHERE {
     bsbm-inst-dataFromVendor1:Offer39> bsbm:product ?productURI .
     ?productURI rdfs:label ?productlabel .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:vendor ?vendorURI .
     ?vendorURI rdfs:label ?vendorname .
     ?vendorURI foaf:homepage ?vendorhomepage .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:offerWebpage ?offerURL .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:price ?price .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:deliveryDays ?deliveryDays .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:validTo ?validTo
}
```

APPENDIX E

12 BSBM FEDERATED QUERY BASED ON WAWPART

Following is the 12 Federated Berlin Benchmark (BSBM) Queries.

Query 1: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label FROM <BSBM_MP>
WHERE {
    ?product rdfs:label ?label .
    ?product a bsbm-inst:ProductType1 .
    ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
    ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
    ?product bsbm:productPropertyNumeric1 ?value1 .
      FILTER (?value1 > 400)
}
ORDER BY ?label
LIMIT 10
```

Query 2: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?label ?comment ?producer ?productFeature ?propertyTextual1 ?propertyTextual2
?propertyTextual3 ?propertyNumeric1 ?propertyNumeric2 ?propertyTextual4
?propertyTextual5 ?propertyNumeric4 FROM <BSBM_MP>
WHERE {
  bsbm-inst-dataFromProducer2:Product78 rdfs:label ?label .
  bsbm-inst-dataFromProducer2:Product78 rdfs:comment ?comment .
  bsbm-inst-dataFromProducer2:Product78 bsbm:producer ?p .
  ?p rdfs:label ?producer .
  SERVICE <http://172.19.48.181:8890/sparql>{
```

```
     bsbm-inst-dataFromProducer2:Product78 dc:publisher ?p .
}
  bsbm-inst-dataFromProducer2:Product78 bsbm:productFeature ?f .
  ?f rdfs:label ?productFeature .
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual1 ?propertyTextual1.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual2 ?propertyTextual2.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual3 ?propertyTextual3.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric1 ?propertyNumeric1.
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric2 ?propertyNumeric2.
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual4 ?propertyTextual4
}
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual5 ?propertyTextual5
}
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric4 ?propertyNumeric4
}
}
```

Query 3: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?product ?label FROM <BSBM_MP>
WHERE {
    ?product rdfs:label ?label .
    ?product a bsbm-inst:ProductType1 .
       ?product bsbm:productFeature bsbm-inst:ProductType1 .
       ?product bsbm:productPropertyNumeric1 ?p1 .
       FILTER ( ?p1 > 10 )
       ?product bsbm:productPropertyNumeric3 ?p3 .
       FILTER (?p3 < 400 )
    OPTIONAL {
        ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
        ?product rdfs:label ?testVar }
    FILTER (!bound(?testVar))
}
ORDER BY ?label
LIMIT 10
```

Query 4: runs on cluster 3: http://172.19.48.183:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label ?propertyTextual FROM <BSBM_MP>
```

```
WHERE {
    {
       ?product rdfs:label ?label .
       ?product rdf:type bsbm-inst:ProductType1 .
       ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
      ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
       ?product bsbm:productPropertyTextual1 ?propertyTextual .
      ?product bsbm:productPropertyNumeric1 ?p1 .
      FILTER ( ?p1 > 200 )
    } UNION {
      ?product rdfs:label ?label .
       ?product rdf:type bsbm-inst:ProductType2 .
       ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
      ?product bsbm:productFeature bsbm-inst:ProductFeature3 .
       ?product bsbm:productPropertyTextual1 ?propertyTextual .
      ?product bsbm:productPropertyNumeric2 ?p2 .
      FILTER ( ?p2> 400 )
    }
}
ORDER BY ?label
OFFSET 5
LIMIT 10
```

Query 5: runs on cluster 3: http://172.19.48.183:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?productLabel FROM <BSBM_MP>
WHERE {
  ?product rdfs:label ?productLabel .
 FILTER bsbm-inst-dataFromProducer2:Product78 != ?product)
  bsbm-inst-dataFromProducer2:Product78 bsbm:productFeature ?prodFeature .
  ?product bsbm:productFeature ?prodFeature .
  bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric1 ?origProperty1 .
  ?product bsbm:productPropertyNumeric1 ?simProperty1 .
 FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 -
120))
  bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric2 ?origProperty2 .
  ?product bsbm:productPropertyNumeric2 ?simProperty2 .
 FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 -
170))
}
ORDER BY ?productLabel
LIMIT 5
```

Query 6: runs on cluster 3: http://172.19.48.183:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT ?product ?label FROM <BSBM_MP>
WHERE {
    ?product rdfs:label ?label .
    SERVICE <http://172.19.48.185:8890/sparql>{?product rdf:type bsbm:Product .}
    FILTER regex(?label, "trackable")
}
```

Query 7: runs on cluster 3: http://172.19.48.181:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?productLabel ?offer ?price ?vendor ?vendorTitle ?review ?revTitle ?reviewer
?revName ?rating1 ?rating2 FROM <BSBM_MP>
WHERE {
       SERVICE <http://172.19.48.183:8890/sparql>{ bsbm-inst-
dataFromProducer2:Product78 rdfs:label ?productLabel .}
    OPTIONAL {
        ?offer bsbm:product bsbm-inst-dataFromProducer2:Product78 .
        ?offer bsbm:price ?price .
        ?offer bsbm:vendor ?vendor .
        SERVICE <http://172.19.48.183:8890/sparql>{?vendor rdfs:label ?vendorTitle .}
        ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#DE> .
        ?offer dc:publisher ?vendor .
        ?offer bsbm:validTo ?date .
        FILTER (?date > 2000-06-20 )
    }
    OPTIONAL {
       ?review bsbm:reviewFor bsbm-inst-dataFromProducer2:Product79 .
       ?review rev:reviewer ?reviewer .
       ?reviewer foaf:name ?revName .
       ?review dc:title ?revTitle .
    OPTIONAL { ?review bsbm:rating1 ?rating1 . }
    OPTIONAL { ?review bsbm:rating2 ?rating2 . }
    }
}
```

Query 8: runs on cluster 1: http://172.19.48.181:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?title ?text ?reviewDate ?reviewer ?reviewerName ?rating1 ?rating2 ?rating3
?rating4 FROM <BSBM_MP>
WHERE {
	?review bsbm:reviewFor bsbm-inst-dataFromProducer2:Product78 .
	?review dc:title ?title .
	?review rev:text ?text .
	FILTER langMatches( lang(?text), "EN" )
	?review bsbm:reviewDate ?reviewDate .
	?review rev:reviewer ?reviewer .
	?reviewer foaf:name ?reviewerName .
	OPTIONAL { ?review bsbm:rating1 ?rating1 . }
	OPTIONAL { ?review bsbm:rating2 ?rating2 . }
	OPTIONAL { ?review bsbm:rating3 ?rating3 . }
	OPTIONAL { ?review bsbm:rating4 ?rating4 . }
}
ORDER BY DESC(?reviewDate)
LIMIT 20
```

Query 9: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX bsbm-inst-dataFromRatingSite1: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/>

DESCRIBE ?x
WHERE { bsbm-inst-dataFromRatingSite1:Review192 rev:reviewer ?x }
```

Query 10: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>


SELECT DISTINCT ?offer ?price FROM <BSBM_MP>
WHERE {
	?offer bsbm:product bsbm-inst-dataFromProducer2:Product78 .
	?offer bsbm:vendor ?vendor .
	 ?offer dc:publisher ?vendor .
```

```
       ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#US> .
       ?offer bsbm:deliveryDays ?deliveryDays .
       FILTER (?deliveryDays <= 3)
       ?offer bsbm:price ?price .
    ?offer bsbm:validTo ?date .
    FILTER (?date > 2000-06-20 )
}
ORDER BY xsd:double(str(?price))
LIMIT 10
```

Query 12: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-export: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/vocabulary/export/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bsbm-inst-dataFromVendor1: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/>

CONSTRUCT {
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:product ?productURI .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:productlabel ?productlabel .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:vendor ?vendorname .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:vendorhomepage ?vendorhomepage .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:offerURL ?offerURL .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:price ?price .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:deliveryDays ?deliveryDays .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:validuntil ?validTo }
WHERE {
     bsbm-inst-dataFromVendor1:Offer39> bsbm:product ?productURI .
   SERVICE <http://172.19.48.183:8890/sparql>{?productURI rdfs:label ?productlabel .}
     bsbm-inst-dataFromVendor1:Offer39> bsbm:vendor ?vendorURI .
   SERVICE <http://172.19.48.183:8890/sparql>{?vendorURI rdfs:label ?vendorname .}
     ?vendorURI foaf:homepage ?vendorhomepage .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:offerWebpage ?offerURL .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:price ?price .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:deliveryDays ?deliveryDays .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:validTo ?validTo
}
```

APPENDIX F

12 BSBM FEDERATED QUERY BASED ON RANDOM PARTITION

Following is the 12 Federated Berlin Benchmark (BSBM) Queries.

Query 1: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label FROM <BSBM_RP>
WHERE {
    ?product rdfs:label ?label .
    ?product a bsbm-inst:ProductType1 .
    SERVICE <http://172.19.48.183:8890/sparql>{?product bsbm:productFeature bsbm-
inst:ProductFeature1 .}
    SERVICE <http://172.19.48.183:8890/sparql>{?product bsbm:productFeature bsbm-
inst:ProductFeature2 .}
    SERVICE <http://172.19.48.185:8890/sparql>{?product bsbm:productPropertyNumeric1
?value1 .}
       FILTER (?value1 > 400)
}
ORDER BY ?label
LIMIT 10
```

Query 2: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?label ?comment ?producer ?productFeature ?propertyTextual1 ?propertyTextual2
?propertyTextual3 ?propertyNumeric1 ?propertyNumeric2 ?propertyTextual4
?propertyTextual5 ?propertyNumeric4 FROM <BSBM_RP>
WHERE {
  bsbm-inst-dataFromProducer2:Product78 rdfs:label ?label .
  SERVICE <http://172.19.48.183:8890/sparql>{
      bsbm-inst-dataFromProducer2:Product78 rdfs:comment ?comment .
```

```
  }
  SERVICE <http://172.19.48.183:8890/sparql>{
       bsbm-inst-dataFromProducer2:Product78 bsbm:producer ?p .
  }
  ?p rdfs:label ?producer .
  SERVICE <http://172.19.48.185:8890/sparql>{
     bsbm-inst-dataFromProducer2:Product78 dc:publisher ?p .
  }
  SERVICE <http://172.19.48.183:8890/sparql>{
     bsbm-inst-dataFromProducer2:Product78 bsbm:productFeature ?f .
  }
  ?f rdfs:label ?productFeature .
  SERVICE <http://172.19.48.183:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual1 ?propertyTextual1.
  }
  SERVICE <http://172.19.48.185:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual2 ?propertyTextual2.
  }
  SERVICE <http://172.19.48.183:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual3 ?propertyTextual3.
  }
  SERVICE <http://172.19.48.185:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric1 ?propertyNumeric1.
  }
  SERVICE <http://172.19.48.183:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric2 ?propertyNumeric2.
  }
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual4 ?propertyTextual4
  }
OPTIONAL {
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyTextual5 ?propertyTextual5
  }
OPTIONAL {
  SERVICE <http://172.19.48.183:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric4 ?propertyNumeric4
  }
 }
}
```

Query 3: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?product ?label FROM <BSBM_RP>
WHERE {
    ?product rdfs:label ?label .
    ?product a bsbm-inst:ProductType1 .
    SERVICE <http://172.19.48.183:8890/sparql>{
       ?product bsbm:productFeature bsbm-inst:ProductType1 .
```

```
    }
    SERVICE <http://172.19.48.185:8890/sparql>{
      ?product bsbm:productPropertyNumeric1 ?p1 .
    }
       FILTER ( ?p1 > 10 )
       SERVICE <http://172.19.48.183:8890/sparql>{
           ?product bsbm:productPropertyNumeric3 ?p3 .
        }
       FILTER (?p3 < 400 )
    OPTIONAL {
        SERVICE <http://172.19.48.183:8890/sparql>{
            ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
        }
        ?product rdfs:label ?testVar }
    FILTER (!bound(?testVar))
}
ORDER BY ?label
LIMIT 10
```

Query 4: runs on cluster 1: http://172.19.48.183:8890/sparql

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label ?propertyTextual FROM <BSBM_RP>
WHERE {
    {
       ?product rdfs:label ?label .
       ?product rdf:type bsbm-inst:ProductType1 .
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productFeature bsbm-inst:ProductFeature1 .}
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productFeature bsbm-inst:ProductFeature2 .}
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productPropertyTextual1 ?propertyTextual .}
       SERVICE <http://172.19.48.185:8890/sparql>{
             ?product bsbm:productPropertyNumeric1 ?p1 .}
       FILTER ( ?p1 > 200 )
    } UNION {
       ?product rdfs:label ?label .
       ?product rdf:type bsbm-inst:ProductType2 .
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productFeature bsbm-inst:ProductFeature1 .}
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productFeature bsbm-inst:ProductFeature3 .}
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productPropertyTextual1 ?propertyTextual .}
       SERVICE <http://172.19.48.183:8890/sparql>{
             ?product bsbm:productPropertyNumeric2 ?p2 .}
       FILTER ( ?p2> 400 )
    }
```

```
}
ORDER BY ?label
OFFSET 5
LIMIT 10
```

Query 5: runs on cluster 3: http://172.19.48.183:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?productLabel FROM <BSBM_RP>
WHERE {
  SERVICE <http://172.19.48.181:8890/sparql>{?product rdfs:label ?productLabel .}
 FILTER bsbm-inst-dataFromProducer2:Product78 != ?product)
  bsbm-inst-dataFromProducer2:Product78 bsbm:productFeature ?prodFeature .
  ?product bsbm:productFeature ?prodFeature .
  SERVICE <http://172.19.48.185:8890/sparql>{
bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric1 ?origProperty1 .
  }
  SERVICE <http://172.19.48.185:8890/sparql>{
       ?product bsbm:productPropertyNumeric1 ?simProperty1 .
  }
 FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 -
120))
  bsbm-inst-dataFromProducer2:Product78 bsbm:productPropertyNumeric2 ?origProperty2 .
  ?product bsbm:productPropertyNumeric2 ?simProperty2 .
 FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 -
170))
}
ORDER BY ?productLabel
LIMIT 5
```

Query 6: runs on cluster 1: http://172.19.48.181:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT ?product ?label FROM <BSBM_RP>
WHERE {
    ?product rdfs:label ?label .
    ?product rdf:type bsbm:Product .
```

```
    FILTER regex(?label, "trackable")
}
```

Query 7: runs on cluster 5: http://172.19.48.185:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?productLabel ?offer ?price ?vendor ?vendorTitle ?review ?revTitle ?reviewer
?revName ?rating1 ?rating2 FROM <BSBM_RP>
WHERE {
       SERVICE <http://172.19.48.181:8890/sparql>{
              bsbm-inst-dataFromProducer2:Product78 rdfs:label ?productLabel .}
    OPTIONAL {
        SERVICE <http://172.19.48.183:8890/sparql>{
              ?offer bsbm:product bsbm-inst-dataFromProducer2:Product78 .}
        ?offer bsbm:price ?price .
        SERVICE <http://172.19.48.183:8890/sparql>{?offer bsbm:vendor ?vendor .}
        SERVICE <http://172.19.48.181:8890/sparql>{?vendor rdfs:label ?vendorTitle .}
        SERVICE <http://172.19.48.183:8890/sparql>{
              ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#DE> .}
        ?offer dc:publisher ?vendor .
        ?offer bsbm:validTo ?date .
        FILTER (?date > 2000-06-20 )
    }
    OPTIONAL {
       SERVICE <http://172.19.48.181:8890/sparql>{
              ?review bsbm:reviewFor bsbm-inst-dataFromProducer2:Product79 .}
       SERVICE <http://172.19.48.181:8890/sparql>{?review rev:reviewer ?reviewer .}
       ?reviewer foaf:name ?revName .
       SERVICE <http://172.19.48.181:8890/sparql>{?review dc:title ?revTitle .}
    OPTIONAL { ?review bsbm:rating1 ?rating1 . }
    OPTIONAL { ?review bsbm:rating2 ?rating2 . }
    }
}
```

Query 8: runs on cluster 1: http://172.19.48.181:8890/sparql

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?title ?text ?reviewDate ?reviewer ?reviewerName ?rating1 ?rating2 ?rating3
?rating4 FROM <BSBM_RP>
WHERE {
	?review bsbm:reviewFor bsbm-inst-dataFromProducer2:Product78 .
	?review dc:title ?title .
	SERVICE <http://172.19.48.185:8890/sparql>{ ?review rev:text ?text .}
	FILTER langMatches( lang(?text), "EN" )
	SERVICE <http://172.19.48.183:8890/sparql>{
		?review bsbm:reviewDate ?reviewDate.}
	?review rev:reviewer ?reviewer .
	SERVICE <http://172.19.48.185:8890/sparql>{?reviewer foaf:name ?reviewerName.}
	SERVICE <http://172.19.48.185:8890/sparql>{
		OPTIONAL { ?review bsbm:rating1 ?rating1 . }}
	SERVICE <http://172.19.48.183:8890/sparql>{
		OPTIONAL { ?review bsbm:rating2 ?rating2 . }}
	SERVICE <http://172.19.48.185:8890/sparql>{
		OPTIONAL { ?review bsbm:rating3 ?rating3 . }}
	OPTIONAL { ?review bsbm:rating4 ?rating4 . }
}
ORDER BY DESC(?reviewDate)
LIMIT 20
```

Query 9: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX bsbm-inst-dataFromRatingSite1: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/>

DESCRIBE ?x
WHERE { bsbm-inst-dataFromRatingSite1:Review192 rev:reviewer ?x }
```

Query 10: runs on cluster 5: http://172.19.48.185:8890/sparql

```
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bsbm-inst-dataFromProducer2: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/>


SELECT DISTINCT ?offer ?price FROM <BSBM_RP>
WHERE {
	SERVICE <http://172.19.48.181:8890/sparql>{
		?offer bsbm:product bsbm-inst-dataFromProducer2:Product78 .}
	SERVICE <http://172.19.48.181:8890/sparql>{?offer bsbm:vendor ?vendor .}
	 ?offer dc:publisher ?vendor .
	SERVICE <http://172.19.48.183:8890/sparql>{
		?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#US> .}
	?offer bsbm:deliveryDays ?deliveryDays .
	FILTER (?deliveryDays <= 3)
	?offer bsbm:price ?price .
```

```
    ?offer bsbm:validTo ?date .
    FILTER (?date > 2000-06-20 )
}
ORDER BY xsd:double(str(?price))
LIMIT 10
```

Query 12: runs on cluster 1: http://172.19.48.181:8890/sparql

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-export: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/vocabulary/export/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bsbm-inst-dataFromVendor1: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/>

CONSTRUCT {
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:product ?productURI .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:productlabel ?productlabel .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:vendor ?vendorname .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:vendorhomepage ?vendorhomepage .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:offerURL ?offerURL .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:price ?price .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:deliveryDays ?deliveryDays .
     bsbm-inst-dataFromVendor1:Offer39> bsbm-export:validuntil ?validTo }
WHERE {
     SERVICE <http://172.19.48.183:8890/sparql>{
             bsbm-inst-dataFromVendor1:Offer39> bsbm:product ?productURI .}
     ?productURI rdfs:label ?productlabel .
     SERVICE <http://172.19.48.183:8890/sparql>{
             bsbm-inst-dataFromVendor1:Offer39> bsbm:vendor ?vendorURI .}
     ?vendorURI rdfs:label ?vendorname .
     ?vendorURI foaf:homepage ?vendorhomepage .
     bsbm-inst-dataFromVendor1:Offer39> bsbm:offerWebpage ?offerURL .
     SERVICE <http://172.19.48.185:8890/sparql>{
             bsbm-inst-dataFromVendor1:Offer39> bsbm:price ?price .}
     SERVICE <http://172.19.48.185:8890/sparql>{
             bsbm-inst-dataFromVendor1:Offer39> bsbm:deliveryDays ?deliveryDays .}
     SERVICE <http://172.19.48.185:8890/sparql>{
             bsbm-inst-dataFromVendor1:Offer39> bsbm:validTo ?validTo }
}
```

APPENDIX G

EXTRA 14 LUBM QUERY

Following is the extra 14 Lehigh University Benchmark (LUBM) Queries.

1. Linear Queries:

Query 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?researchGroups ub:subOrganizationOf ?department .
	?department ub:name "Department1" .
}
```

Query 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?department ub:subOrganizationOf ?university .
	?professor ub:worksFor ?department .
	?student ub:advisor ?professor .
	?student ub:memberOf <http://www.Department1.University0.edu>
}
```

Query 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE
{
	?resgroup ub:subOrganizationOf ?department .
	?professor ub:worksFor ?department .
	?student ub:advisor ?professor .
	?student ub:memberOf <http://www.Department1.University0.edu> .
}
```

Query 4:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?advisor ub:emailAddress ?email .
	?advisor ub:worksFor ?department .
	?department ub:name "Department1" .
}
```

2. Star Query:

Query 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?student ub:advisor ?advisor .
	?student ub:name ?name .
	?student ub:undergraduateDegreeFrom ?university .
	?student ub:takesCourse
	<http://www.Department1.University0.edu/GraduateCourse33> .
}
```

Query 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?professor ub:emailAddress ?mail .
	?professor ub:telephone ?phone .
	?professor ub:doctoralDegreeFrom ?doctor .
	?professor ub:name "FullProfessor1" .
}
```

Query 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?student ub:memberOf ?department .
	?student ub:takesCourse ?course .
	?student ub:advisor ?advisor .
	?student ub:teachingAssistantOf ?tacourse .
	?student ub:emailAddress ?email .
	?student ub:name ?name .
```

```
        ?student ub:telephone ?telephone .
        ?student ub:undergraduateDegreeFrom <http://www.University0.edu> .
}
```

Query 4:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
        ?student ub:memberOf ?department .
        ?student ub:takesCourse ?course .
        ?student ub:advisor ?advisor .
        ?student ub:teachingAssistantOf ?tacourse .
        ?student ub:emailAddress ?email .
        ?student ub:name "GraduateStudent71" .
        ?student ub:telephone ?telephone .
        ?student ub:undergraduateDegreeFrom ?university .
}
```

3. Snowflakes Query:

Query 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
        ?student ub:advisor ?advisor .
        ?advisor ub:worksFor ?department .
        ?department ub:subOrganizationOf ?university .
        ?student ub:name ?name .
        ?student ub:telephone ?tel .
        ?student ub:takesCourse <http://www.Department12.University1.edu/Course1> .
}
```

Query 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
        ?department ub:name ?name .
        ?resgroup ub:subOrganizationOf ?department .
        ?department ub:subOrganizationOf <http://www.University0.edu> .
        ?student ub:memberOf ?department .
        ?student ub:advisor ?professor .
        ?student ub:takesCourse ?course .
}
```

Query 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?department ub:name ?name .
	?resgroup ub:subOrganizationOf ?department .
	?department ub:subOrganizationOf ?university .
	?student ub:memberOf ?department .
	?student ub:advisor ?professor .
	?student ub:takesCourse
	<http://www.Department1.University0.edu/GraduateCourse33> .
}
```

4. Complex Query:

Query 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?department ub:subOrganizationOf ?university .
	?resgroup ub:subOrganizationOf ?department .
	?student ub:memberOf ?department .
	?department ub:name ?name .
	?student ub:advisor ?professor .
	?publication ub:publicationAuthor ?professor .
	?publication ub:publicationAuthor
	<http://www.Department1.University10.edu/AssociateProfessor1>.
}
```

Query 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?department ub:subOrganizationOf ?university .
	?resgroup ub:subOrganizationOf ?department .
	?student ub:memberOf ?department .
	?student ub:advisor ?professor .
	?student ub:takesCourse ?course .
	?publication ub:publicationAuthor ?professor .
	?publication ub:publicationAuthor
	<http://www.Department1.University10.edu/AssociateProfessor1> .
	?publication ub:name ?title .
}
```

Query 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT * FROM <LUBM>
WHERE {
	?student ub:advisor ?advisor .
	?advisor ub:worksFor ?department .
	?department ub:subOrganizationOf <http://www.University0.edu> .
	?head ub:headOf ?department .
	?head ub:emailAddress ?email .
	?head ub:doctoralDegreeFrom ?alma .
	?student ub:name ?name .
	?student ub:telephone ?tel .
	?student ub:takesCourse ?course .
}
```

APPENDIX H

EXTRA 9 BSBM QUERY

Following is the extra 8 Berlin Benchmark (BSBM) Queries.

Query 1:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix rev: <http://purl.org/stuff/rev#>

Select ?productType ?reviewCount
{
 { Select ?productType (count(?review) As ?reviewCount)
  {
   ?productType a bsbm:ProductType .
   ?product a ?productType .
   ?product bsbm:producer ?producer .
   ?producer bsbm:country %Country1% .
   ?review bsbm:reviewFor ?product .
   ?review rev:reviewer ?reviewer .
   ?reviewer bsbm:country %Country2% .
  }Group By ?productType
 }
} Order By desc(?reviewCount) ?productType
Limit 10
```

Query 2:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT ?otherProduct ?sameFeatures
  {?otherProduct a bsbm:Product .
    FILTER(?otherProduct != %Product%)
    { SELECT ?otherProduct (count(?otherFeature) As ?sameFeatures)
      { %Product% bsbm:productFeature ?feature .
        ?otherProduct bsbm:productFeature ?otherFeature .
        FILTER(?feature=?otherFeature)
      }Group By ?otherProduct
    }
  } Order By desc(?sameFeatures) ?otherProduct
Limit 10
```

Query 3:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix dc: <http://purl.org/dc/elements/1.1/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?product (xsd:float(?monthCount)/?monthBeforeCount As ?ratio)
{

 { Select ?product (count(?review) As ?monthCount)
   { ?review bsbm:reviewFor ?product .
     ?review dc:date ?date .
     Filter(?date >= "%ConsecutiveMonth_1%"^^<http://www.w3.org/2001/XMLSchema#date>
&& ?date < "%ConsecutiveMonth_2%"^^<http://www.w3.org/2001/XMLSchema#date>)
   } Group By ?product
 }

 { Select ?product (count(?review) As ?monthBeforeCount)
   { ?review bsbm:reviewFor ?product .
     ?review dc:date ?date .
     Filter(?date >= "%ConsecutiveMonth_0%"^^<http://www.w3.org/2001/XMLSchema#date>
&& ?date < "%ConsecutiveMonth_1%"^^<http://www.w3.org/2001/XMLSchema#date>) #
   } Group By ?product
     Having (count(?review)>0)
 }
} Order By desc(xsd:float(?monthCount) / ?monthBeforeCount) ?product
Limit 10
```

Query 4:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?feature (?withFeaturePrice/?withoutFeaturePrice As ?priceRatio)
{
 { Select ?feature (avg(xsd:float(xsd:string(?price))) As ?withFeaturePrice)
   { ?product a %ProductType% ;
              bsbm:productFeature ?feature .
     ?offer bsbm:product ?product ;
            bsbm:price ?price .
   } Group By ?feature
 }
 { Select ?feature (avg(xsd:float(xsd:string(?price))) As ?withoutFeaturePrice)
   {
     { Select distinct ?feature
       { ?p a %ProductType% ;
            bsbm:productFeature ?feature .
       }
     }
```

```
      ?product a %ProductType% .
      ?offer bsbm:product ?product ;
             bsbm:price ?price .
      FILTER NOT EXISTS { ?product bsbm:productFeature ?feature }
    }Group By ?feature
  }
}Order By desc(?withFeaturePrice/?withoutFeaturePrice) ?feature
Limit 10
```

Query 5:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?country ?product ?nrOfReviews ?avgPrice
{
  { Select ?country (max(?nrOfReviews) As ?maxReviews)
    {
      { Select ?country ?product (count(?review) As ?nrOfReviews)
        { ?product a %ProductType% .
          ?review bsbm:reviewFor ?product ;
                  rev:reviewer ?reviewer .
          ?reviewer bsbm:country ?country .
        } Group By ?country ?product
      }
    } Group By ?country
  }
  { Select ?country ?product (avg(xsd:float(xsd:string(?price))) As ?avgPrice)
    { ?product a %ProductType% .
      ?offer bsbm:product ?product .
      ?offer bsbm:price ?price .
    } Group By ?country ?product
  }
  { Select ?country ?product (count(?review) As ?nrOfReviews)
    { ?product a %ProductType% .
      ?review bsbm:reviewFor ?product .
      ?review rev:reviewer ?reviewer .
      ?reviewer bsbm:country ?country .
    } Group By ?country ?product
  } FILTER(?nrOfReviews=?maxReviews)
} Order By desc(?nrOfReviews) ?country ?product
```

Query 6:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?reviewer (avg(xsd:float(?score)) As ?reviewerAvgScore)
```

```
{
  { Select (avg(xsd:float(?score)) As ?avgScore)
    {
      ?product bsbm:producer %Producer% .
      ?review bsbm:reviewFor ?product .
      { ?review bsbm:rating1 ?score . } UNION
      { ?review bsbm:rating2 ?score . } UNION
      { ?review bsbm:rating3 ?score . } UNION
      { ?review bsbm:rating4 ?score . }
    }
  }
  ?product bsbm:producer %Producer% .
  ?review bsbm:reviewFor ?product .
  ?review rev:reviewer ?reviewer .
  { ?review bsbm:rating1 ?score . } UNION
  { ?review bsbm:rating2 ?score . } UNION
  { ?review bsbm:rating3 ?score . } UNION
  { ?review bsbm:rating4 ?score . }
} Group By ?reviewer
Having (avg(xsd:float(?score)) > min(?avgScore) * 1.5)
```

Query 7:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?product
{
  { Select ?product
    {
      { Select ?product (count(?offer) As ?offerCount)
        {
          ?product a %ProductType% .
          ?offer bsbm:product ?product .
        }Group By ?product
      }
    }Order By desc(?offerCount)
    Limit 1000
  }FILTER NOT EXISTS
  {
    ?offer bsbm:product ?product .
    ?offer bsbm:vendor ?vendor .
    ?vendor bsbm:country ?country .
    FILTER(?country=%Country%)
  }
}
```

Query 8:

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
```

```
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?vendor (xsd:float(?belowAvg)/?offerCount As ?cheapExpensiveRatio)
{
 { Select ?vendor (count(?offer) As ?belowAvg)
   {
     { ?product a %ProductType% .
       ?offer bsbm:product ?product .
       ?offer bsbm:vendor ?vendor .
       ?offer bsbm:price ?price .
       { Select ?product (avg(xsd:float(xsd:string(?price))) As ?avgPrice)
         { ?product a %ProductType% .
           ?offer bsbm:product ?product .
           ?offer bsbm:vendor ?vendor .
           ?offer bsbm:price ?price .
         } Group By ?product
       }
     } . FILTER (xsd:float(xsd:string(?price)) < ?avgPrice)
   } Group By ?vendor
  }
  { Select ?vendor (count(?offer) As ?offerCount)
    { ?product a %ProductType% .
      ?offer bsbm:product ?product .
      ?offer bsbm:vendor ?vendor .
    } Group By ?vendor
  }
} Order by desc(xsd:float(?belowAvg)/?offerCount) ?vendor
limit 10
```