IJCTA, 9(26) 2016, pp. 425-434
© International Science Press

An Innovative Approach to RDF Graph Data Processing : Spark with GraphX and Scala

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

ABSTRACT

The number of linked data sources and the size of the linked open data graph keep growing every day. As a consequence, semantic RDF services are more and more confronted to various "big data" problems. Query processing is one of them and needs to be efficiently addressed with executions over scalable, highly available and fault tolerant frameworks. Data management systems requiring these properties are rarely built from scratch but are rather designed on top of an existing cluster computing engine. The iterative nature of algorithms in the field of graph analytics makes the simple MapReduce style of parallel and distributed processing less effective. Still, the need to provide answers even for very large graphs is driving the research. Progress, trends and directions for future research are presented. The goal of this paper is to provide an effective way to represent the RDF graph data and enhance the query processing by utilizing the unbeatable characteristics of Spark, GraphX library and Scala.

Keywords: RDF Graph data, Graph processing, Graph analytics, SPARK, SCALA, GraphX.

1. INTRODUCTION

As an evolution of the World Wide Web to enable data sharing and reuse "across application, enterprise, and community boundaries", the concept of Semantic Web has been introduced by Tim Berners-Lee. The Semantic Web has lot of building blocks such as the Resource Description Framework (RDF), the Web Ontology Language (OWL) and SPARQL[19].

A. Representation of RDF as Graph

Theoretically, data encoded using RDF is represented as a directed labeled multi-graph, making RDF similar in many respects to graph database models. In an RDF graph, vertices are the resources (IRIs), blank nodes, or literals and edges are formed from RDF triples (the triple's predicate, which is an IRI, is the edge's label). Strictly speaking, blank nodes may have no identifiers and therefore no corresponding labels, which makes an RDF graph slightly different than a multi-graph..

In few cases, RDF triple-stores are organized as graphs, while many implementations of RDF triple-stores rely on some form of a relational database as they are structured in representation [1]. Quad stores is an example of its implementations, as RDF data sets may include multiple graphs and the graph to which a triple belongs is the fourth element, making it a quadruple.

B. SPARQL as the Query Language

World Wide Web Consortium recommends SPARQL is the query language for RDF data sets,. The following appears very similar when articulated in the SPARQL query language:

[1] Department of Computer Science, Bishop Heber College, TN, India. *E-mail: babynirmala7@yahoo.co.in*

[2] Department of Computer Science, Bishop Heber College, Tiruchirappalli, TN, India. *E-mail: jgrsathiaseelan@gmail.com*

SELECT ? *x*, ? *y*, ? *z*

WHERE {*x* a Lawyer. *y* a Doctor. *z* a Lawyer.

x friend *y*.

x competes_with *z*.

y friend *z*

Extensive research has been conducted to optimize query engines for processing SPARQL queries [2]. Sophisticated indexing strategies and graph-based storage models are developed due to the unremitted research in this area.

To provide a method of publishing a variety of structured data sets as interlinked RDF data sets, a Linking Open Data (LOD) project [3] has been initiated. As of 2014, the LOD project encompassed 1014 interlinked RDF data sets spanning a multitude of knowledge areas, such as geographic, government, life sciences, linguistics, media, publications and social networking. At the center of it is DBpedia, an RDF representation of the Wikipedia, which is interlinked with a high number of other data sets. Overall, the size of the interlinked RDF graph in the LOD cloud is measured in tens of billions of RDF triples and therefore edges.

As the sizes of individual RDF data graphs continue to grow significantly, optimization of processing of SPARQL queries becomes even more essential, especially in view of the need for complex, hypothesis-driven [4] and analytics related queries. Much effort must be dedicated to distributed processing of SPARQL queries [5] [6].

Moreover, processing of federated SPARQL queries on the LOD graph is challenging and requires vigorous research. As the individual datasets considerably increase in size, RDF graph partitioning and its impact on distributed processing of SPARQL queries and RDF graph analytics [7] is of noteworthy. SPARQL query processing was formulated in terms of subgraph isomorphism and related to graph databases in [8]. A SPARQL implementation based on graph homomorphism is given in [9].

C. Graph Processing Systems

In contrast to general data processing systems (*e.g.*, Map Reduce, Dryad, and Spark) which compose data-parallel operators to transform collections and are capable of expressing a wide range of computation, graph processing systems apply vertex-centric logic to transform data on a graph and exploit the graph structure to achieve more efficient distributed execution.

D. Property Graphs

Graph data comes in many forms. The graph can be explicit (*e.g.*, social networks, web graphs, and financial transaction networks) or imposed through modeling assumptions (*e.g.*, collaborative filtering, language modeling, deep learning, and computer vision). The structure of a graph can be denoted as $G = (V; E)$ by a set of vertices $1V = f\ 1; : : : ; ng$ and a set of m directed edges E. The directed edge $(i; j)\ 2E$ connects the source vertex $i\ 2V$ with the target vertex $j\ 2V$. The resulting graphs can have tens of billions of vertices and edges and are often highly sparse with complex, irregular, and often power-law structure[22].

2. MAPREDUCE AS A DISTRIBUTED DATAFLOW FRAMEWORK

Cluster compute frameworks like Map Reduce and its generalizations is refer to distributed dataflow framework. Although details vary from one framework to another, they typically satisfy the following

properties like a data model consisting of typed collections, a coarse-grained data-parallel programming model composed of deterministic operators which transform collections, a scheduler that breaks each job into a directed acyclic graph (DAG) of tasks and a runtime that can endure stragglers and partial cluster failures without restarting.

In MapReduce, the programming model exposes only two data flow operators: *map* and *reduce* (*a.k.a.*, group-by). Each job can contain at most two layers in its DAG of tasks. More modern frameworks such as DryadLINQ, Pig, and Spark expose additional dataflow operators such as fold and join, and can execute tasks with multiple layers of dependencies. Distributed dataflow frameworks have enjoyed broad adoption for a wide variety of data processing tasks, including ETL, SQL query processing, and iterative machine learning. They have also been shown to scale to thousands of nodes operating on petabytes of data.

While MapReduce (MR) is a popular cluster computing paradigm, it is not well suited for graph analytics because many graph analytics tasks are iterative in nature[19].

3. STRIKING FEATURES OF APACHE SPARK AND GRAPHX

A. Spark

Spark has several features that are particularly attractive for GraphX: The Spark storage abstraction called Resilient Distributed Datasets (RDDs) enables applications to keep data in memory, which is essential for iterative graph algorithms.

(a) RDDs allows user-defined data partitioning, and the execution engine can exploit this to co-partition RDDs and co-schedule tasks to avoid data movement. This is essential for encoding partitioned graphs.

(b) Spark logs the lineage of operations used to build an RDD, enabling automatic reconstruction of lost partitions upon failures. Furthermore, Spark supports optional in-memory distributed replication to reduce the amount of re-computation on failure.

(c) Spark provides a high-level API in Scala that can be easily extended. This aided in creating a coherent API for both collections and graphs[20].

B. GraphX

GraphX is implemented on top of Spark [12], a widely used data parallel engine. Similar to Hadoop MapReduce, a Spark cluster consists of a single driver node and multiple worker nodes. The driver node is responsible for task scheduling and dispatching, while the worker nodes are responsible for the actual computation and physical data storage.

C. Spark *vs.* MapReduce

However, Spark also has several features that differentiate it from traditional MapReduce engines and are important to the design of GraphX [11][13].They are

(a) *In-Memory Caching*: Spark provides the Resilient Distributed Dataset (RDD) in-memory storage abstraction. RDDs are collections of objects that are partitioned across a cluster. GraphX uses RDDs as the foundation for distributed collectionsand graphs.

(b) *Computation DAGs:* In contrast to the two-stage MapReduce topology, Spark supports general computation DAGs by composing multiple data-parallel operators on RDDs, making it more suitable

for expressing complex data flows. GraphX uses and extends Spark operators to achieve the unified programming abstraction.

(c) *Lineage-Based Fault Tolerance:* RDDs and the data-parallel computations on RDDs are fault-tolerant. Spark can automatically reconstruct any data or execute tasks lost during failures. Programmable Partitioning: RDDs can be co-partitioned and co-located. When joining two RDDs that are co-partitioned and co-located, GraphX can exploit this property to avoid any network communication.

(d) *Interactive Shell:* Spark allows users to interactively execute Spark commands in a Scala or Python shell. Spark shell extended to support interactive graph analytics[14].

D. Significance of key library GraphX in Spark

Spark's API for working with graphs of nodes and arcs. The "GraphX: Unifying Data-Parallel and Graph-Parallel Analytics" paper by GraphX's inventors has a whole section on RDF as related work, saying "we adopt some of the core ideas from the RDF work including the triples view of graphs" [10][21].

4. PROPOSED WORK

A. Representation of RDF using GraphX Systems

The possibility of using such a novel Big Data technology with RDF was intriguing. It would be interesting to output a typical GraphX graph as RDF so that SPARQL queries can be performed on it. That are not typical of GraphX processing, and then to go the other way: read a good-sized RDF dataset into GraphX and do things with it that would not be typical of SPARQL processing.

RDF and GraphX systems have much to offer each other. Dr. Payberah's work symbolizes why they're precious, at least in the case of using Spark from Scala: Spark provides higher-order functions that can hand off the functions and data to structures that can be stored in distributed memory[17] [18].

This allows the kinds of interactive and iterative (for example, machine learning) tasks that generally don't work well with Hadoop's batch-oriented MapReduce model. Apparently, for tasks that would work fine with MapReduce, Spark versions also run much faster because their better use of memory lets them avoid all the disk I/O that is typical of MapReduce jobs.

Distributed memory can be used with the help of Spark which provides a data structure called a Resilient Distributed Dataset, or RDD. Spark can take care of their distribution across computing cluster by storing data in RDDs. GraphX helps store a set of nodes, arcs, and crucially for RDF types—extra information about each in RDDs[18].

To express a "typical" GraphX graph structure as RDF, the following graph has been taken and expanded a bit.

B. Property Graph with Various Collaborations

To construct a property graph consisting of the various collaborators on the GraphX project, the vertex property might contain the username and occupation. Edges can be annotated with a string describing the relationships between collaborators:

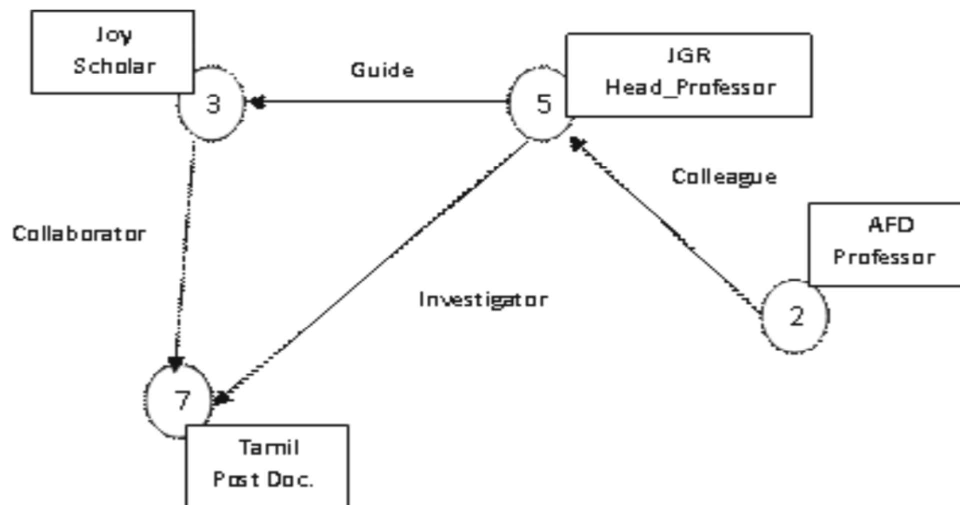

Figure 1: Property Graph

Vertex Table

Id	*Property (V - Verrtice)*
2	AFD, Professor
3	Joy, Scholar
5	JGR, Head_Professor
7	Tamil, Post Doc.

Edge Table

Source Id	*Destination Id*	*Property (E - Edge)*
2	5	Colleague
5	3	Guide
3	7	Collaborator
5	7	Investigator

C. A Scala Program that creates an RDD and stores and queries information of the this property Graph

Scala program below, like the Property Graph mentioned above, creates an RDD called users of nodes about people at a university and an RDD called relationships. This stores information about edges that connect the nodes. RDDs use long integers such as the 3L and 7L values shown below as identifiers for the nodes. It can store additional information about nodes—for example, that node 3L is named “Joy” and has the title “scholar”—as well as additional information about edges—for example, that the user represented by 5L has an “guide” relationship to user 3L. Few extra nodes and edges are added to give the eventual SPARQL queries a little more to work with.

Once the node and edge RDDs are defined, the program creates a graph from them. After that, code is added to output RDF triples about node relationships to other nodes (or, in RDF parlance, object property triples) using a base URI that I defined at the top of the program to convert identifiers to URIs when necessary. This produced triples such as

<http://bhc.edu.in/cs/xprgraph#AFD>

<http://bhc.edu.in/cs/xprgraph#colleague>

<http:// bhc.edu.in/cs/xprgraph#JGR>

in the output. Finally, the program outputs non-relationship values (literal properties), producing triples such as

<http:// bhc.edu.in/cs/xprgraph#Joy>

<http:// bhc.edu.in/cs/xprgraph#role> “scholar”.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
object ExamplePropertyGraph {
def main(args: Array[String]) {
val baseURI = “http://bhc.edu.in/cs/xprgraph#”
al sc = new SparkContext(“local”, “ExamplePropertyGraph”, “172.21.0.1”)
 // Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
sc.parallelize(Array(
(3L, (“Joy”, “scholar”)),
(7L, (“Tamil”, “postdoc”)),
(5L, (“JGR”, “head_prof”)),
(2L, (“AFD”, “prof”)),
// Following lines are new data
(8L, (“Anusha”, “scholar”)),
(9L, (“Arivu”, “scholar”)),
(10L, (“Nimmi”, “postdoc”)),
(11L, (“Naga”, “scholar”)),
))
// Create an RDD for edges
val relationships: RDD[Edge[String]] =
sc.parallelize(Array(
Edge(3L, 7L, “collaborator”),
Edge(5L, 3L, “guide”),
Edge(2L, 5L, “colleague”),
Edge(5L, 7L, “investigator”),
// Following lines are new data
Edge(5L, 8L, “guide”),
```

```
Edge(2L, 9L, " guide"),
Edge(5L, 10L, "investigator"),
Edge(2L, 11L, "guide ")
))
// Build the initial Graph
val graph = Graph(users, relationships)
// Output object property triples
graph.triplets.foreach( t => println(
s"<$baseURI${t.srcAttr._1}> <$baseURI${t.attr}> <$baseURI${t.dstAttr._1}> ."
))
// Output literal property triples
users.foreach(t => println(
s"""<$baseURI${t._2._1}> <${baseURI}role> \"${t._2._2}\" ."""
))
sc.stop
}
}
```

The program writes out the RDF with full URIs for each every resource, but here is the Turtle version that uses prefixes to help it fit on this page better:

```
@prefix xp: <http://bhc.edu.in/cs/xprgraph#> .
xp:AFD xp:colleague xp:JGR.
xp:AFD xp:guide xp:JGR.
xp:AFD xp:guide xp:arivu.
xp:Joy xp:collabarator xp:tamil.
xp:JGR xp:guide xp:joy.
xp:JGR xp:investigator   xp:nimmi .
xp:JGR xp:guide xp:naga.
xp:JGR  xp:guide xp:viji.
xp:Joy xp:role "scholar" .
xp:Tamil xp:role "postdoc" .
xp:AFD xp:role "professor" .
xp:JGR xp:role "head_professpr" .
xp:Anusha xp:role "scholar" .
xp:Arivu  xp:role "scholar" .
```

xp:Viji xp:role “scholar “ .

xp:Naga xp:role “scholar” .

xp:Nimmi xp:role “ scholar “ .

First SPARQL query of the RDF asked this: for each person with Guides, how many do they have?

```
PREFIX xp: <http://bhc.edu.in/cs/xprgraph#> .
SELECT ?person (COUNT(?guide) AS ?advisees)
WHERE {
?person xp:guide ?advisee
}
GROUP BY ?person
```

Here is the result:

| person | advisees |

| xp:JGR | 3 |

| xp:AFD | 2 |

The next query asks about the roles of Joy’s collaborators:

```
PREFIX xp: <http://bhc.edu.in/cs/xprgraph#> .
SELECT ?collaborator ?role
WHERE {
xp:JGR xp:collab ?collaborator .
?collaborator xp:role ?role .
}
```

As it turns out, there’s only one:

| collaborator | role |

| xp:Tamil | “postdoc” |

Does Arivu have a relationship to any prof, and if so, who and what relationship?

```
PREFIX xp: <http:// bhc.edu.in/cs/xprgraph#> .
SELECT ?person ?relationship
WHERE {
?person xp:role “prof” .
{ xp:arivu ?relationship ?person }
UNION
{ ?person ?relationship xp:arivu }
}
```

And here is our answer:

| person | relationship |

| xp:AFD | xp:guide |

After going through the above program one may have the following questions?

(a) What about properties of edges? For example, what if anyone wanted to say that an xp:advisorproperty was an rdfs:subPropertyOf the Dublin Core property dc:contributor?

(b) The ability to assign properties such as a name of "Joy" and a role of "scholar" to a node like 3L is nice, but what if a consistent set of properties that will be assigned to every node—for example, person data is aggregated from two different sources that don't use all the same properties to describe these persons?

Neither of those were difficult with GraphX. An approach to let a GraphX program read in any RDF and then perform GraphX operations on it[21].

6. CONCLUSIONS AND FURTHER SCOPE

Thus the RDF graph data can be represented effectively and the query processing can be enhanced by the very challenging properties of Spark with its library Graphx and the higher-order function language Scala . This work is the beginning to go for RDF Graph analytics. More insights can be derived from this structure like unknown relationships. Pattern matching and shortest route algorithm can be applied and many valuable insights can be analysed in the future by taking any RDF repositories into consideration.

REFERENCES

[1] B. McBride, "Jena: Implementing the RDF model and syntax specification." in *SemWeb*, 2001.

[2] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "SPARQL basic graph pattern optimization using selectivity estimation," in *Proceedings of the 17th international conference on World Wide Web. ACM,* 595–604,2008.

[3] Heath, T., and Bizer, C., "Linked data: Evolving the web into a global data space. Synthesis lectures on thesemantic web: theory and technology",2013.

[4] G. Gosal, K. J. Kochut, and N. Kannan, "Prokino: an ontology for integrative analysis of protein kinases in cancer" *PloS one,* **6(12)**, 2011.

[5] J. Huang, D. J. Abadi, and K. Ren, "Scalable SPARQL querying of large RDF graphs," *Proceedings of the VLDB Endowment*, 4(11), 1123–1134, 2011.

[6] P. Peng, L. Zou, M. T. O¨ zsu, L. Chen, and D. Zhao, "Processing SPARQL queries over linked data–a distributed graph based approach," arXiv preprint arXiv:1411.6763, 2014.

[7] L. Zou, M. T. O¨ zsu, L. Chen, X. Shen, R. Huang, and D. Zhao, "gstore: a graph-based SPARQL query engine," *TheVLDB JournalThe International Journal on Very Large Data Bases*. **23(4)**, 565–590, 2014.

[8] R. Angles and C. Gutierrez, "Querying RDF data from agraph database perspective," in *The Semantic Web: Research and Applications. Springer,* 346–360, 2005.

[9] O.Corby and C. Faron-Zucker, "Implementation of SPARQL query language based on graph homomorphism," *in Conceptual Structures: Knowledge Architectures for Smart Applications. Springer,* 472–475,2007.

[10] Reynold S. Xin, Daniel Crankshaw, Ankur Dave,Joseph E. Gonzalez, Michael J. Franklin Ion Stoica,"*GraphX: Unifying Data-Parallel and Graph-Parallel analytics"*, 2014.

[11] Mohammed Guller," Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analytics", 2015.

[12] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, Matei Zaharia, "Spark SQL: Relational Data Processing in Spark" in *ACM SIGMOD Conference*, 2015.

[13] M. Zaharia *et al.* "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing", *NSDI*, 2012.

[14] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, Ion Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework", *in the Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[15] Bob DuCharme,"Spark and SPARQL; RDF Graphs and GraphX", 2015. Retreived from "http://www.snee.com/bobdc.blog/2015/03/spark-and-sparql-rdf-graphs-an.html."

[16] Amir H. Payberah, "A Crash course in Scala", 2015. Retrieved from "https://www.sics.se/~amir/files/download/dic/scala.pdf"

[17] Amir H. Payberah, "Spark and Resilient Distributed Datasets", 2015. Retrieved from "https://www.sics.se/~amir/files/download/dic/spark.pdf"

[18] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", 2012.

[19] John A. Miller, Lakshmish Ramaswamy, Krys J. Kochut, and Arash Fard, "Directions for Big Data Graph Analytics Research", *International Journal of Big Data*, **2(1)**, September 2015.

[20] Baby Nirmala M., Sathiaseelan .J.G.R.," Big semantic data analytics: A theoretical study on implementation of technologies and tools", *International Journal of Applied Engineering Research,* ISSN 0973-4562, **10(20)**, 2015.

[21] James A. Scott., "Getting started with Apache Spark", MapR technologies Inc, 2015, First Edition.

[22] Ryza, Uri Laserson, Sean Owen, and Josh Wills, "Advanced Analytics with Spark", O'Reilly Media, Inc.,2015.

[23] Robinson, Jim Webber, and Emil Eifrem, "Graph Databases*",* O'Reilly Media, Inc.,2013.