ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 13 (2024), e31506
eISSN: 2255-2863
DOI: https://doi.org/10.14201/adcaij.31506

An Efficient Approach to Extract and Store Big Semantic Web Data Using Hadoop and Apache Spark GraphX

Wria Mohammed Salih Mohammed[a,b] and Alaa Khalil Jumaa[a]

[a]Technical College of Informatics, Sulaimani Polytechnic University, Sulaimani 46001, Kurdistan Region, Iraq
[b]College of Base Education, University of Sulaimani, Street 1- Zone 501 Sulaimani, Kurdistan Region, Iraq
wria.mohammedsalih@spu.edu.iq, alaa.alhadithy@spu.edu.iq

KEYWORDS

Hadoop; Semantic web; GraphX; Linked data; SPARQL; HDFS; RDF; Spark

ABSTRACT

The volume of data is growing at an astonishingly high speed. Traditional techniques for storing and processing data, such as relational and centralized databases, have become inefficient and time-consuming. Linked data and the Semantic Web make internet data machine-readable. Because of the increasing volume of linked data and Semantic Web data, storing and working with them using traditional approaches is not enough, and this causes limited hardware resources. To solve this problem, storing datasets using distributed and clustered methods is essential. Hadoop can store datasets because it can use many hard disks for distributed data clustering; Apache Spark can be used for parallel data processing more efficiently than Hadoop MapReduce because Spark uses memory instead of the hard disk. Semantic Web data has been stored and processed in this paper using Apache Spark GraphX and the Hadoop Distributed File System (HDFS). Spark's in-memory processing and distributed computing enable efficient data analysis of massive datasets stored in HDFS. Spark GraphX allows graph-based semantic web data processing. The fundamental objective of this work is to provide a way for efficiently combining Semantic Web and big data technologies to utilize their combined strengths in data analysis and processing.

First, the proposed approach uses the SPARQL query language to extract Semantic Web data from DBpedia datasets. DBpedia is a hugely available Semantic Web dataset built on Wikipedia. Secondly, the extracted Semantic Web data was converted to the GraphX data format; vertices and edges files were generated. The conversion process is implemented using Apache Spark GraphX. Third, both vertices and edge tables are stored in HDFS and are available for visualization and analysis operations. Furthermore, the proposed techniques improve the data storage efficiency by reducing the amount of storage space by half when converting from Semantic Web Data to a GraphX file, meaning the RDF size is around 133.8 and GraphX is 75.3. Adopting parallel data processing provided by Apache Spark in the proposed technique reduces the required data processing and analysis time.

This article concludes that Apache Spark GraphX can enhance Semantic Web and Big Data technologies. We minimize data size and processing time by converting Semantic Web data to GraphX format, enabling efficient data management and seamless integration.

1. Introduction

Due to the growing utilization of linked data and the swift development of the Semantic Web, it turned into a W3C standard with the purpose of facilitating machine comprehension of the World Wide Web.

Semantic Web datasets are growing more extensive and sophisticated due to their increased popularity and use. The resource description framework is the Semantic Web's primary data format (RDF). Subject, predicate, and object are the three essential components of RDF. They are called triples. Subject and predicate are the only elements that can be URIs, but the object can be literals or URIs. When numerous RDF triples are merged, they form a graph structure in which the subjects and objects become nodes, and the predicates represent the edges that link these nodes. This graph-based approach enables a flexible and connected data structure, allowing for rich semantic relationships and complex data modeling inside the Semantic Web. Moreover, RDF has a standard query language called SPARQL, a W3C standard that can be used to query massive RDF datasets.

In terms of storage and management, small RDF data sets can be handled efficiently. However, with the considerable volume of Semantic Web data, it is difficult to hold the entire dataset at once and on a single machine. Furthermore, the efficient management of a large Semantic Web dataset cannot be achieved using traditional methods (Kulcu et al., 2016). Traditional methods, particularly relational databases, have disadvantages when combining Semantic Web and big data technology. Strict schemas limit flexible data representation, scalability issues with massive datasets, and difficulty performing complex graph-based queries. Adopting advanced distributed frameworks such as Apache Spark and GraphX to enable a more efficient and seamless integration of Semantic Web data with big data analytics, delivering improved scalability, flexibility, and graph processing capabilities, becomes essential to tackle these limitations The primary challenge of the work is to integrate two distinct research domains within the discipline of emergency studies; the Semantic Web and big data technologies. It mainly stores and manages big Semantic Web data using Apache Spark and HDFS. In this paper, GraphX has been chosen because it is compatible with Hadoop components; Apache Spark can work with HDFS, Hive, HBase, Yarn, and Flume. The reason for using Spark is that Spark was initially developed to enhance and replace Hadoop. GraphX in Python can be executed using GraphFrame which is based on Spark DataFrames.

DBpedia aims to collect organized, multilingual knowledge from Wikipedia and make it accessible online. DBpedia makes use of the Semantic Web and linked data technologies. Data on DBpedia is available in 111 languages. In this paper, DBpedia is the selected dataset (Hakimov, 2013). Because of its comprehensive and complete data, adherence to Semantic Web standards, interlinking with other datasets, strong research community support, scalability, real-world application potential, and open data accessibility, DBpedia is chosen as a dataset.

The proposed solution in this paper is divided into three steps. In the first step, a part of DBpedia is retrieved as a selected dataset using the SPARQL query language; SPARQL can be used to extract meaningful information from RDF files. In the second step, RDF as graph data, which consists of subject and object as nodes and predicate as the relationship between them, can be converted to GraphX data format. GraphX is a part of Apache Spark, then when RDF converts to GraphX data format, both vertex and edge tables are built. Thirdly, both tables can be stored in the HDFS and analyzed or visualized. The reason for which Spark is used instead of MapReduce is that Spark is faster than MapReduce. MapReduce depends on disk-based computation, which might be slower than in-memory computing. The disk-based MapReduce may have a limitation when dealing with iterative algorithms, which are often applied in Semantic Web analytics. The rest of this paper is organized as follows: Related research is reviewed in Section 2; Semantic Web is discussed in Section 3. In Section 4, the Hadoop HDFS is highlighted. Furthermore, Hadoop MapReduce to Apache Spark is described in Section 5. Spark, GraphX, and GraphFrame are described in Section 6. Lastly, the proposed techniques and implementation are outlined in Sections 7 and 8, respectively.

2. Related Works

Ramalingeswara Rao et al. (2021) examined distributed significant data processing. A weighted split network contains the top k user-to-user communities. The Otsuka-Ochiai coefficient is the foundation for the novel method which the authors propose to use to gauge node similarity. Using Apache Spark and Flink, the authors employed the two algorithms TUCSGF and TUCFlink. The research used programs such as Graph-Frame and GraphX as distributed graph processing tools. On the other hand, both techniques rely primarily on parameter choices such as similarity criteria and top-k values, which can be challenging to optimize for different datasets. Furthermore, the algorithms may struggle to efficiently meet the computational and memory needs if the network grows.

Baby Nirmala & Sathiaseelan (2021) emphasized the advantages of using MapReduce and in-memory processing for Semantic Web data. Social network datasets were used as a case study, and the GraphX library was used to analyze the data. The study's primary goal was to improve access to the RDF data graph. The potential privacy and ethical issues, associated with accessing and analyzing user-generated content, are one of the constraints of utilizing social media data as the selected dataset in this work.

Banane & Belangour (2020) addressed the challenge of processing and analyzing large Semantic Webs in the big data domain. The authors chose a model-driven approach that, depending on the user's preferences, converts SPARQL queries into Hive, Pig, or Spark scripts. They created a metamodel for every tool, detailing the relationships between the SPARQL metamodel and each significant data query language. The Atlas transformation language was then used to execute the transformation. They took advantage of three extremely rich datasets in distributed RDF data.

A technical overview of Apache Spark for big data analysis was provided in the paper of Salloum et al. (2016).

Apache Spark's popularity has grown due to its cutting-edge in-memory programming methodology and modules for machine learning, graph analysis, streaming, and structured data processing. This paper summarizes the essential concepts, characteristics, and elements of Apache Spark. The study also highlights the potential for further research and development for Apache Spark in big data analytics.

Agathangelos et al. (2018a) covered the creation of RDF data storage and query systems for distributed contexts. Partitioning is the main topic. To increase data partitioning, the research suggests a system that uses machine learning methods. The proposed approach leverages logistic regression and random forest machine learning techniques to optimise data streaming processes with enhanced scalability, while also effectively understanding the structural aspects of a partitioned database. These techniques can efficiently handle large-scale datasets by parallelizing calculations and dividing tasks across numerous nodes. Logistic regression, a sophisticated classification technique, may be used to classify binary outcomes on large datasets in a distributed way. On the other hand, random forest is an ensemble learning approach that employs numerous decision trees trained on various data subsets, making it well-suited for dispersed contexts. Using these strategies improves data segmentation and increases system scalability, allowing for the processing and analyzing of large volumes of data with decreased computing time and resource needs.

3. Semantic Web

The Semantic Web was first described as a technique by Tim Berners-Lee to interpret the web of data. The W3C recommends the Semantic Web, which refers to connected data for the web. The Semantic Web gives meaning to the data and enables collaboration between humans and machines (Mohammed & Jumaa, 2021).

The current web structure is created as the first step in building the Semantic Web. Thus, machines can process and understand web data (Berners-Lee et al., 2001). The Semantic Web has three foundation statements, including subject, predicate, and object; this primary type of data which includes the three statements, is called RDF. Furthermore, the resource description framework schema (RDFS) defines restrictions, classes, properties, and vocabularies. An ontology can be made using web ontology languages (OWL), which comprises XML and RDF (Lahore, 2016).

However, RDF is usually used by Semantic Web communities. In RDF, the subject is a distinct resource, the object is either a literal or unique resource, and the predicate explains the relationship between the two. As a result, RDF data can be represented as a labeled, directed multigraph, as shown in Figure 1. An RDF data graph can be presented by the equation below (Lim et al., 2015):

$$G = (V, E, L, \pi) \qquad (1)$$

E is a collection of directed edges among nodes in V, where V is the set of nodes. L is a collection of edges with node labels, and π is a function with the notation (Lim et al., 2015):

$$\pi\colon V \cup E \rightarrow L \qquad (2)$$

Also, ∀vi, vj ∈ V, vi≠vj, this means $\pi(vi) \neq \pi(vj)$.

RDF can be seen as a graph where vertices can be considered as the subjects and the objects, and predicates are used to label the directed edges between the subject and objects. Figure 1 below shows the graph of the vehicle RDF file as an example.

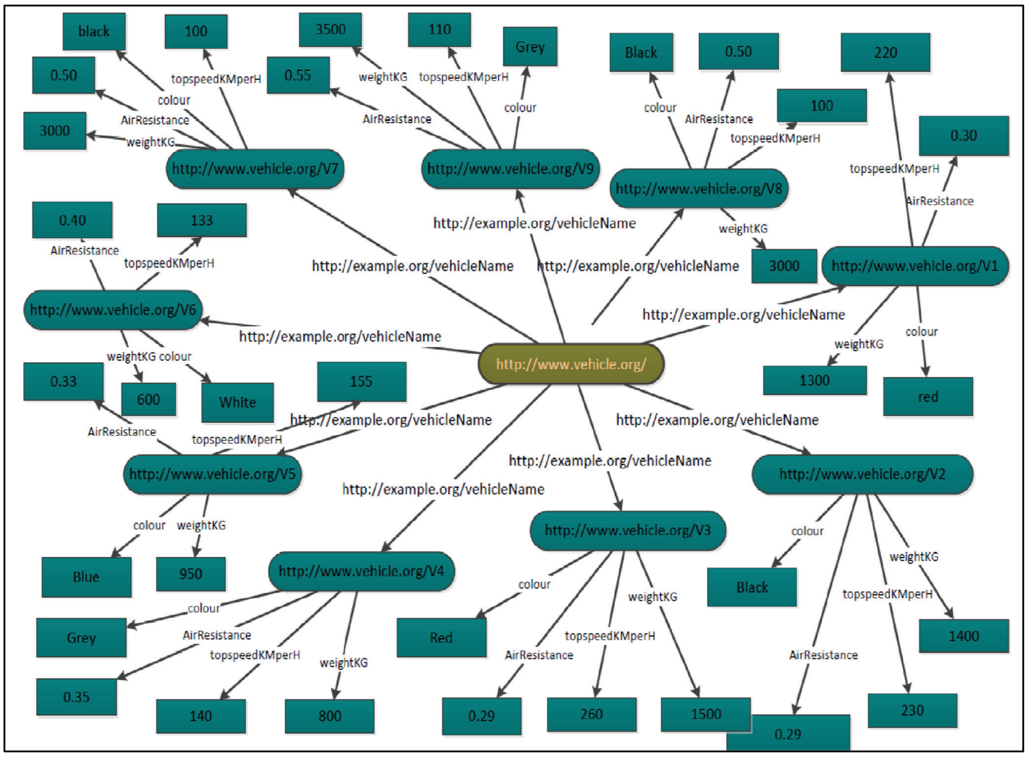

http://www.vehicle.org/
http://example.org/vehicleName
http://www.vehicle.org/V1
http://www.vehicle.org/V2
http://www.vehicle.org/V3
http://www.vehicle.org/V4
http://www.vehicle.org/V5
http://www.vehicle.org/V6
http://www.vehicle.org/V7
http://www.vehicle.org/V8
http://www.vehicle.org/V9
colour
weightKG
topspeedKMperH
AirResistance

Figure 1. Graph of vehicle RDF(Mohammed & Saraee, 2016)

Semantic Web data can have a query to extract essential information; SPARQL (protocol and RDF query language) is utilized for this purpose. SPARQL is a query language for accessing and manipulating Semantic Web data. There are tools and programming languages available to build SPARQL queries. SPARQL query can work based on the subject-predicate-object structure to know how to query the Semantic Web data because it has three columns: subject, predicate, and object (Gopalani & Arora, 2015).

DBpedia is a community project to extract multilanguage and structured knowledge from the Wikipedia website using linked data and Semantic Web technologies. The project obtains data from 111 languages from the Wikipedia website. The English version of DBpedia contains about 400 million facts, around 3.7 million things. The whole Wikipedia dataset from 110 languages is approximately 1.46 billion facts. The DBpedia dataset is about 320 standard classes and 1650 properties. Thousands of datasets on the web link to DBpedia, which makes DBpedia the central interlinking hub in the LOD (linked open data) (Salloum et al., 2016). In this research, DBpedia is used as a dataset.

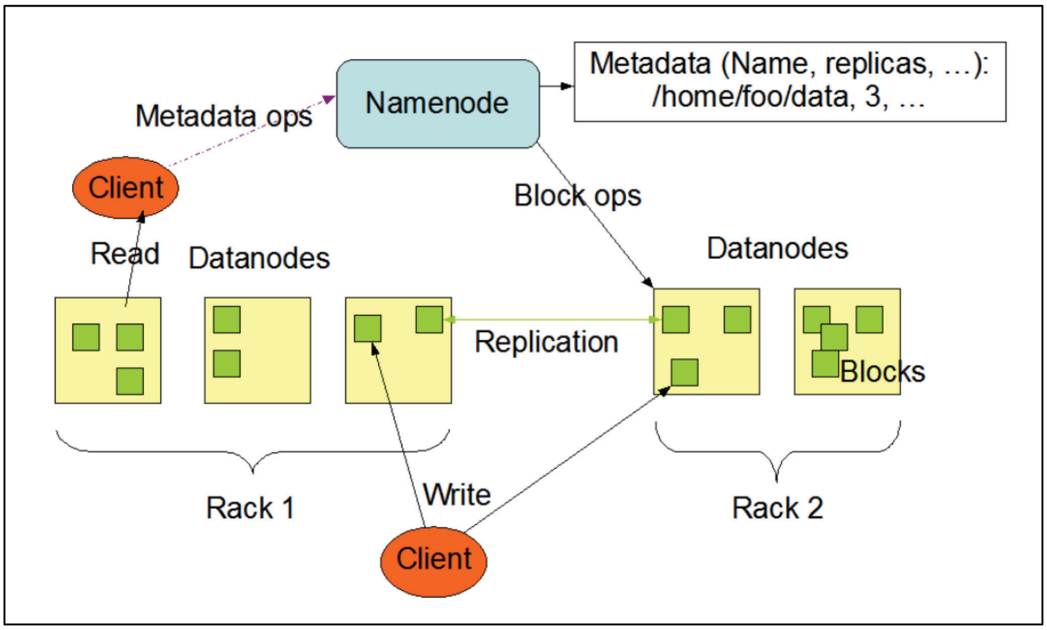


Figure 2. HDFS works (Farag Azzedin, 2013)

4. Hadoop HDFS

Hadoop is an open-source framework designed to manage and analyze large datasets effectively. It is developed in Java. Hadoop includes three different components: Yarn, MapReduce, and Hadoop Distributed File System (HDFS). HDFS divides the file data into metadata and data. It has two main benefits: fault tolerance because it has more than a copy of the data and can handle extensive data (Omar & Jumaa, 2019).

HDFS is a master/slave architecture. Any Hadoop cluster has a NameNode and several DataNodes. Typically, the NameNode is responsible for directing the HDFS namespace and granting client requests for access to specific files (Farag Azzedin, 2013). HDFS is built to handle the storage of huge files across machines in significant clusters. DataNode stores files as a sequence of blocks. It can configure the size of the blocks in DFS (distributed file system). Any block that has other duplicates depends on the configuration. NameNode has to know the network topology of the cluster (Donvito et al., 2014).

Figure 2 represents how HDFS works. Clients request file modification or file information from a NameNode computer, which uses NameNode to handle file input and output. NameNode and DataNode use the built-in web servers to find the current status of a cluster (Farag Azzedin, 2013).

5. From Hadoop MapReduce to Apache Spark

Apache Spark began as a research project in AMPLab at the University of California Berkeley The programming model was designed to facilitate a diverse range of applications, surpass the capabilities of MapReduce (Gopalani & Arora, 2015).

Apache Spark is a standard for analyzing big data after Hadoop MapReduce. Spark combines advanced programming with distributed computing. Also, Spark has the same fault tolerance and scalability capabilities as MapReduce. Apache Spark is easier and faster compared to Hadoop MapReduce. Java, Python, Scala, R, and SQL can work with Spark (Salloum et al., 2016). Spark uses resilient distributed database (RDD), which stores data in memory; it makes Spark increase the performance of the batch processing task, reaching from 10 to 100 times faster than MapReduce. Another reason to make Spark a better choice is that it allows caching the data in memory, which benefits machine learning algorithms.

Spark is efficient for multi-pass applications that need low-latency multiple parallel operations. Remarkably, the applications used for analytics, for instance, are iterative algorithms, including graph algorithms such as the PageRank algorithm (Gopalani & Arora, 2015).

6. Spark, GraphX, and GraphFrame

Apache Spark is a memory-based cluster computing system. Spark has two novel data abstractions: resilient distributed database (RDD) and data frames (DF). RDDs are a collection of partitioning objects across machines. The RDD can be effectively manipulated through Scala, Java, Python, and R programming commands. In addition, the DF (Data Frame) is a data abstraction that incorporates schema enforcement and data compression techniques (Banane & Belangour, 2019; Naacke et al., 2017). Spark can be divided into four modules: GraphX for network and graph data analysis, MLlib for machine learning algorithms, Spark-SQL for managing and querying databases, and streaming for analysis and processing of flows (Banane & Belangour, 2019; Yu et al., 2019).

Workers (executors), cluster management, HDFS, SparkContext, and a driver application, make up Apache Spark. SparkContext is the object that is produced during the execution of the Spark program, and it is in charge of the entire task execution process. As seen in Figure 3, the cluster manager connects the executors through the connection of the SparkContext object. Executors are responsible for logic and keeping the application's data safe (Amol Bansod, 2015).

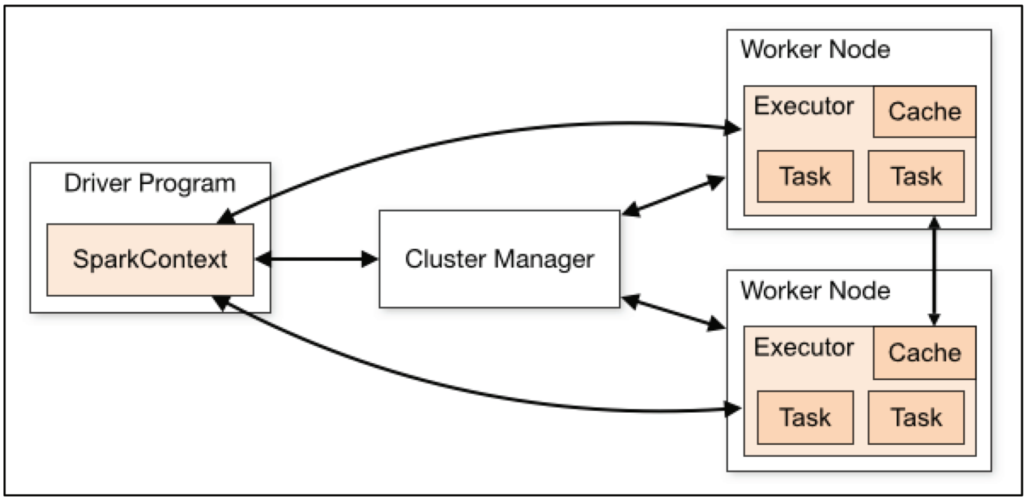


Figure 3. Apache Spark architecture

GraphX is a Spark component for graph-parallel and graph computation. GraphX is an extension of Spark RDD. This Spark RDD introduces graph abstraction. It has a vertex and edge; this can be a directed multigraph with properties. It can be said that GraphX is a graph process library. It has both user-defined attributes associated with an adjacency structure. Graph data can be created using GraphX, which is based on RDD collection, or GraphFrame, which is based on DataFrames, and both are the same (Naacke et al., 2017).

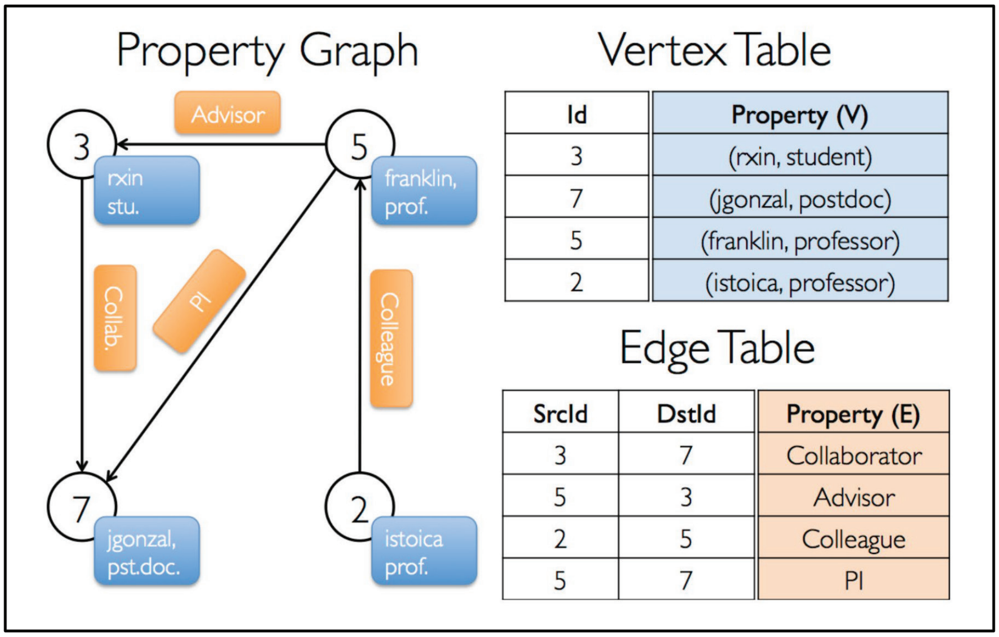
In the context of GraphX, one can observe an illustration. In particular, the vertices include relevant attributes such as occupation and username. On the other hand, the edges are represented by strings that effectively establish the collaborative relationships between individuals. It can be seen from Figure 4 that the vertex table consists of the id with the username and the occupation (vertices). The edge table includes SrcId (source id) and DstId (Destination id) with the property (edge) between vertices. Furthermore, a new package of Apache Spark for graph processing is GraphFrame which is based on DataFrame. The main goal of DataFrames is to manipulate the data using Spark, Python, and R. DataFrame can process a massive amount of data and manage a ten times larger dataset than RDD (Agathangelos et al., 2018b).

GraphFrames can also be used to create vertex and edge tables as follows (Ramalingeswara Rao et al., 2021)

Vertex using DataFrame, vertices = Spark.createDataFrame ([(3,"rxin","student") ,(7,"jgonzal","postdoc") ,etc], ["id","name","role"])

Edge using DataFrame, edges = Spark.createDataFrame ([(3,7,"Collaborator") ,(7,3,"advisor") ,etc], ["SrcId"," DstId","Property"])

GraphFrame using vertices and edges, G=GraphFrame (vertices, edges)



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Figure 4. GraphX vertices and edge tables

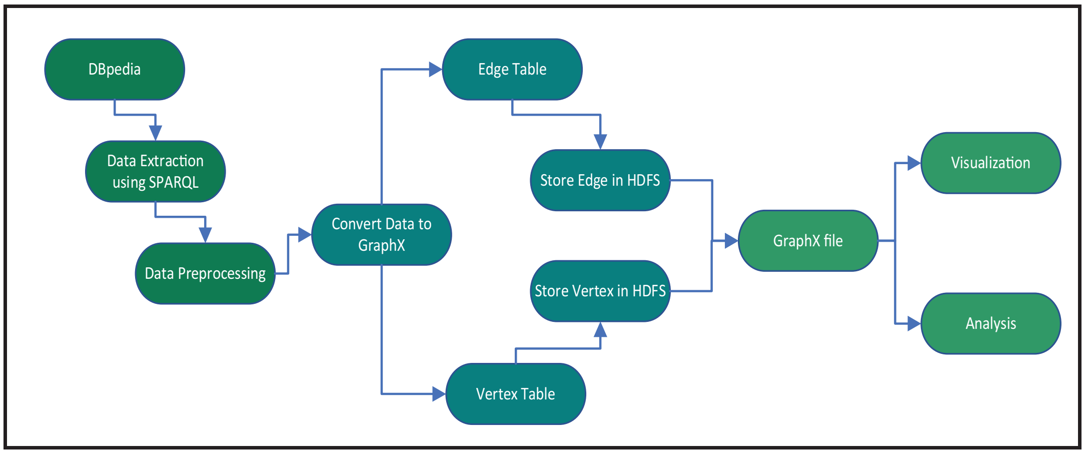


Figure 5. Proposed system architecture

7. The Proposed Technique

Figure 5 represents the roadmap of the overall approach of this study. It has three different processes; the first one is the extraction of data from DBpedia using SPARQL query language, and then it needs to be preprocessed before conversion. The second approach involves converting extracted Semantic Web data from DBpedia to a GraphX file that includes vertex and edge tables. Finally, it needs to be analyzed or visualized. For analysis, the various algorithms of GraphX are applied, such as the PageRank algorithm, inDegree, and outDegree. Also, it can find the most frequent movie according to language, country, and director.

In the extraction algorithm, as shown in Algorithm 1, SPARQL obtains the data from the DBpedia Datasets. SPARQLWrapper is a Python tool that is used to write SPARQL queries against Semantic Web datasets.

Algorithm 1. Extract data from DBpedia RDF

```
Input: Movie Semantic Web Datasets
Output: Data Vector ($Df_{i(j...m)}$, … . , $Df_{n(j...m)}$)
1. BEGIN
  2. For each Movie (i…n ) in the Semantic web Dataset
    1. READ $Movie_i$ from dataset.
    2. Analyse $Movie_i$
    3. For Each attribute (j…m) in $Movie_i$
      1. Extract required attributes ($Movie_{i,j}$)
      2. build Data_field ($Df_{i,j}$)
      3. IF a Missing value exists
        1. Ignore $Df_{i,j}$
        2. F=1
        3. continue
    4. End For
    5. If F=1 continue
    6 $Df_{i,(j...m)}$ Preprocessing
    7. Add $Df_{i,(j...m)}$ to Data Vector
  3. End For
  4. Save Data Vector ($Df_{i,(j...m)}$, ... . , $Df_{n,(j...m)}$)
  5. END
```

Additionally, data preprocessing involves cleaning up the dataset and removing any extraneous or noisy data; because the dataset is saved in vertex and edge tables, missing values are not inserted but are instead disregarded. For instance, no value returns are missing if no movie contains a second language. Finally, the data is converted into the vectors of data ($Df_{i,(j...m)}$, ... , $Df_{n,(j...m)}$), then it needs to be stored.

On the other hand, the data vectors must be converted to a GraphX file, as shown in Algorithm 2. For this reason, GraphFrame in Python is used.

Algorithm 2. GraphX Conversion Algorithm

```
Input: Data Vector ($Df_{i,(j...m)}$, ... . , $Df_{n,(j...m)}$)
Output: Edge and Vertex Tables
                        1. BEGIN
    2. For each Field in Data Vector ($Df_{i,(j...m)}$, ... . , $Df_{n,(j...m)}$)
        1. if $DF_{i,j}$ = Movie Title
            1. if a duplicate value exists
                1. ignore $DF_{i,j}$
                2. F=1
                3. Continue
        2. If f=1 Continue
            1. $DF_{i,j}$ Movie title preprocessing
            2. Add $DF_{i,j}$ ($id_{movie}$, Movie title) to Vertex Table.
            3. if ($DF_{i,j}$ !=Movie Title) and ($DF_{i,j}$ is not empty) and($DF_{i,j}$ is not duplicate)
                1. $DF_{i,j...m}$ preprocessing
                2. Add ($id_{attribute}$, $DF_{i,j...m}$) to Vertex Table.
                3. Add ($id_{movie}$, $id_{attribute}$, relationship) to Edge Table.
        4. elseif ($DF_{i,j}$ is duplicate)
                2. Find ($id_{attribute}$) in Vertex Table.
                3. Add ($id_{movie}$, $id_{attribute}$, relationship) to Edge Table.
        5. else
                1. ignore
    3. End For
    4. Save Edge and Vertex Tables in HDFS
    5. END
```

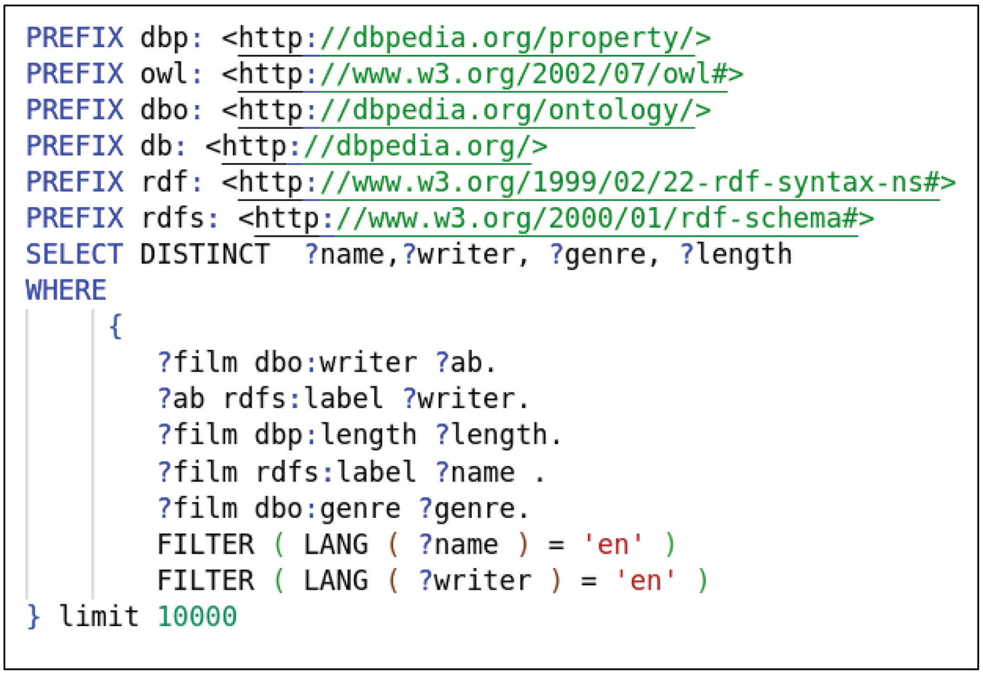
From Algorithm 2, nodes are created for all movie titles. Suppose the title of the movie is duplicated. In that case, it ignores that movie, and if it is a new title, the algorithm builds a node for that movie title with an id (idmovie, movie title), except the title; all other attributes are checked for preprocessing. It needs to build nodes for the attributes with id *($id_{attribute}$, $DF_{i,j...m}$) and store them* in the vertex table. Then the id of the movie and the id of the other attributes with the relationship between them *(id_{movie}, $id_{attribute}$, relationship)* are stored in the edge table. In the case of attribute duplication, such as the English language or the United States country, the corresponding attribute is assigned a unique identifier. Finally, the vertex and edge tables are stored in two different tables in HDFS and can be used for visualization and analysis operations.

8. Proposed Technique Implementation

8.1. Extracting Dataset from DBpedia

To manage the vast volume of data included in DBpedia, it is necessary to employ a query that can effectively choose a specific subset of this dataset, for this reason, the SPARQL query has been used. The research can effectively address questions related to movie characteristics by leveraging movie data from DBpedia, which contains comprehensive and structured information about various movies, including details about the cast, title, genres, release dates, language, country, and more. Furthermore, using DBpedia as a data source assures data consistency, interoperability, and the availability of a community-curated resource, which increases the dataset's dependability and quality. These features might include the selection of movie-related resources or entities, filtering based on specified properties or attributes (e.g., movie genre, release date, country, language), and any other parameters used to ensure the extracted data's relevance and coherence.

In the first step, around 10,000 triples were extracted from DBpedia using SPARQL, here is a sample of the SPARQL query as shown in Figure 6.



```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX db: <http://dbpedia.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT  ?name,?writer, ?genre, ?length
WHERE
    {
        ?film dbo:writer ?ab.
        ?ab rdfs:label ?writer.
        ?film dbp:length ?length.
        ?film rdfs:label ?name .
        ?film dbo:genre ?genre.
        FILTER ( LANG ( ?name ) = 'en' )
        FILTER ( LANG ( ?writer ) = 'en' )
} limit 10000
```

Figure 6. SPARQL to show 10,000 movies on DBpedia

Figure 6 shows a sample of the SPARQL query that was used in this paper because the data needs to be extracted, such as language, country, directors etc. In SPARQL, namespace has been used to define the URLs, and filter some data because it is available in different languages.

8.2. Data Preprocessing

The dataset used in this work is the DBpedia dataset, which contains information from movies. Ten thousand movies with seven columns (imdbid, title, country, languages, directors, genre, and URLs) have been retrieved as a sample. Additionally, a validation process to check for duplicate entries may have been presented to ensure the originality of movie titles. The researchers may have taken suitable procedures to eliminate or combine duplicate items, assuring data accuracy and consistency, by confirming the availability and non-duplication of movie titles in the database. The country and language of the movie should be checked to ensure the data is correct; for instance, Iraq and Iraq_country should be the same country. Indonesian and Indonesian_lang are not two different inputs. The DBpedia collection represents the directors' information as names rather than URLs. This is because linking a director to another URL shows a distinct RDF file unique to that director. Consequently, to maintain consistency and simplicity in the dataset, the researchers isolated and concentrated on the names of the directors. The dataset is more manageable and eliminates the complexity of storing additional RDF files connected with each director by utilizing the directors' names instead of URLs. This choice is a practical means of maintaining the dataset's clarity and usefulness while also giving useful information on the film makers involved. To handle conditions where a movie may contain more than one director, language, or country, programming language techniques are used to extract and represent these multiple values. The data preparation for each variable required identifying if multiple values were associated with a single movie. If many values were discovered, a loop was used to review the list and retrieve them. Following that, all of the retrieved variables were associated with the same movie item in the dataset.

8.3. Building Edge and Vertex Tables from RDF

In the first step of converting to GraphX, the data needs to be read and corrected to avoid duplication; after that, vectors need to be created to create a vertex table. The vertex table consists of the following vectors:

- Vector (imdb, title)
- Vector (langId, Language)
- Vector (countryId, Country)
- Vector (urlId, URL)
- Vector (directorId, director)
- Vector (lengId, length)
- Vector (writerId, writer)
- Vector (genreId, genre)
- etc.

These vectors cannot be duplicated; two titles are not allowed to enter, the same country cannot enter twice, etc.

An edge table consists of the nodes from the vertex and their relationships. Table 1. A and Table 1. B show examples of both vertex and edge tables for one movie.

Table 1. A. vertex table for one movie

Id	Attr
0468569	The Dark Knight
L1	English
L2	Mandarin
C1	United State
C2	United Kingdom
D1	Christopher Nolan
G1	Action
G2	Crime
G3	Drama
G4	Thriller

Table 1. B. Edge table for one movie

Srcid	Dstid	Relationship
0468569	L1	Language
0468569	C1	Country
0468569	L2	Language
0468569	G1	Genre
0468569	C2	Country
0468569	G2	Genre
0468569	G3	Genre
0468569	G4	Genre

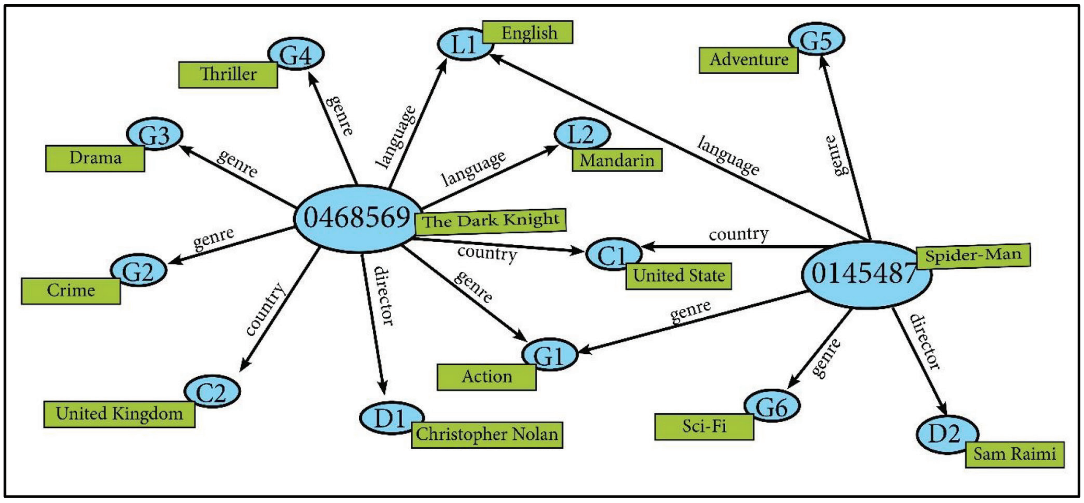
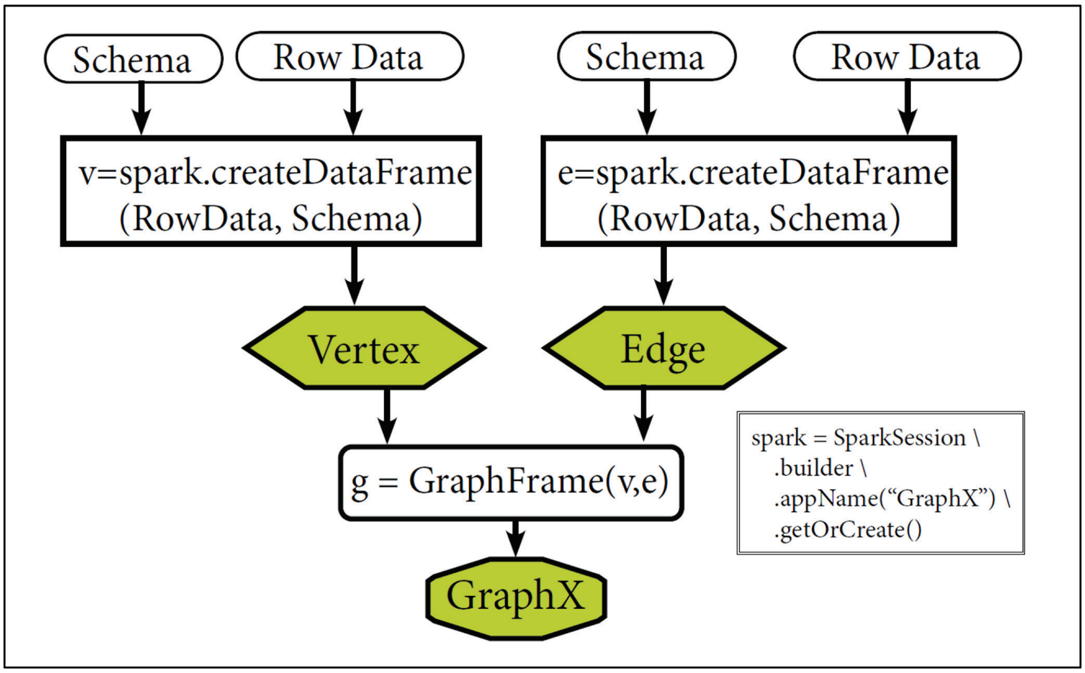


Figure 7. Vertex and edge for two movies

If there is another movie, and the movie has an “action” genre, the movie also connects to “G1” in the edge table, and if its language is “English”, it connects to “L1”. Figure 7 shows the vertex and edge for two movies.

8.4. Building GraphX

GraphX is a powerful tool that enables the construction of graph datasets by using collections of edges and vertices. When GraphX is made, both vertex and edge tables are required. GraphFrame can gather and join both vertices and edges. **createDataFrame()** is a method that can create vertex and edge tables. DataFrame needs both rows of data and schema to build vertex and edge. The schema in


Schema
Row Data
Schema
Row Data
v=spark.createDataFrame (RowData, Schema)
e=spark.createDataFrame (RowData, Schema)
Vertex
Edge
g = GraphFrame(v,e)
GraphX

```
spark = SparkSession \
  .builder \
  .appName("GraphX") \
  .getOrCreate()
```

Figure 8. GraphX building

the vertex table are (id and Attr) columns. Also, for the edge table, the columns are (srcid, dstid, and relationship). GraphX collects both edge and vertex, as shown in Figure 8.

The "imdbid" and the movie's title are extracted from the RDF file using SPARQL. Data row can create the first vector, and using the schema with the vector, we can make the vertex using the **createDataFrame()** method in Python using the following codes.

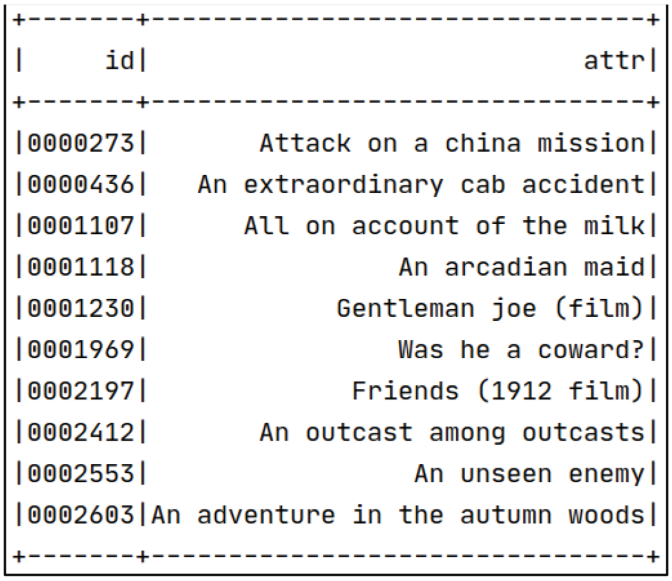
v = Spark.createDataFrame([("0468565", "The Dark Knight"),], ["id", "attr"])

After extracting the title of the movie, other attributes need to be extracted as well; for example, the following codes can be used to create a vertex for the genre of the movie.

v = Spark.createDataFrame([("G1", "Action"),], ["id", "attr"])

"G1" which represents the "action" genre, connects to all action movies. Moreover, the same technique is used for other attributes such as language, country, director, etc. As a result, the vertex table is created, which includes all vertices from the data, a sample of the vertex table is shown in Figure 9.

Also, the edge table can be built from a vertex table and based on RDF data to show the joining of vertices. The first vector is "id", which can be either the "id" of the movie or the "id" of the other attributes such as genre, country etc. For example, the following codes are used if the edge is created to connect the first movie and the first genre.



```
+-------+--------------------------------+
|     id|                            attr|
+-------+--------------------------------+
|0000273|       Attack on a china mission|
|0000436|   An extraordinary cab accident|
|0001107|      All on account of the milk|
|0001118|                 An arcadian maid|
|0001230|             Gentleman joe (film)|
|0001969|                Was he a coward?|
|0002197|              Friends (1912 film)|
|0002412|        An outcast among outcasts|
|0002553|                  An unseen enemy|
|0002603|An adventure in the autumn woods|
+-------+--------------------------------+
```

Figure 9. Sample of vertex table

e = Spark.createDataFrame([(*“0468565”, “G1”* , *“Genre”*) ,] ,["src" , "dst" , "relationship"])

The “G1” represents the “action” genre of the movie and should relate to other action movies. The “relationship” shows the associated point between source (src) and destination (dst) from the edge table. Furthermore, if there is a movie in the English language, the code is as follows:

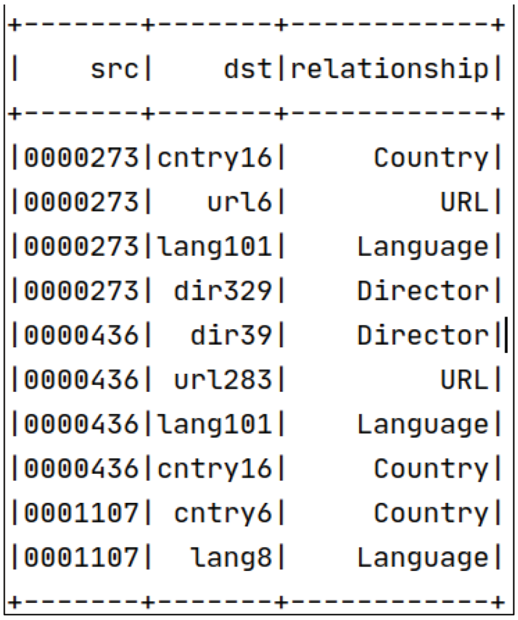
e = Spark.createDataFrame([(*“0468565”*,*“L1”*,*“Language”*),], ["src","dst","relationship"])

“L1” represents the language which can be English, and the relationship between the movie “0468565” and the “L1” is language, because the language of the movie is English.

The sample of the edge table shown in Figure 10 has the source id and destination id for vertices and their relationship.

There is a difference between the RDF file and GraphX; after converting the file to GraphX, the size of GraphX is less than the RDF file as shown in Figure 11, for instance a sample of the RDF file is around 133.8MB but when the file is converted to GraphX, the size of the vertex is 29.6MB and the size of the edge table is 45.7MB. The total GraphX size is 75.3MB. The file size is reduced because all repeated files are written only once.

Semantic Web and big data technology have been used in various sectors to improve operations by providing essential insights. Furthermore, converting RDF data to GraphX can reduce data size by utilizing the efficiency of distributed graph processing and compression techniques.



```
+-------+-------+------------+
|    src|    dst|relationship|
+-------+-------+------------+
|0000273|cntry16|     Country|
|0000273|   url6|         URL|
|0000273|lang101|    Language|
|0000273| dir329|    Director|
|0000436|  dir39|    Director|
|0000436| url283|         URL|
|0000436|lang101|    Language|
|0000436|cntry16|     Country|
|0001107| cntry6|     Country|
|0001107|  lang8|    Language|
+-------+-------+------------+
```

Figure 10. Edge table

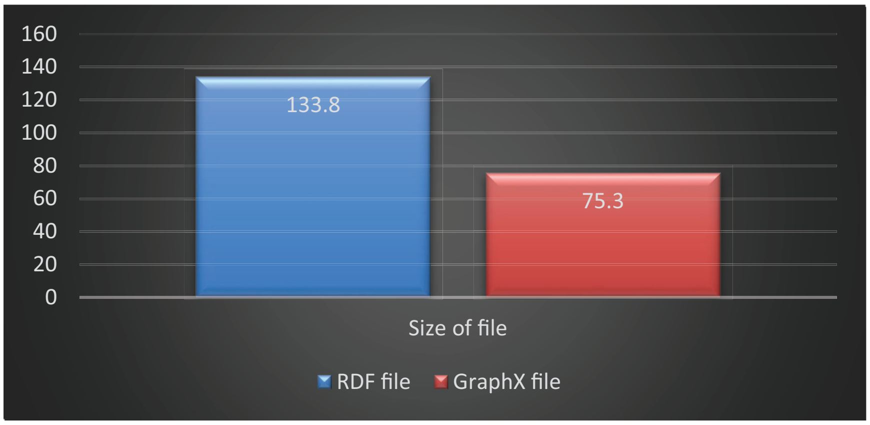


Figure 11. RDF and GraphX size comparison

8.5. Storing Data in HDFS

Apache Spark GraphX cannot store data by itself. The data can be stored in the local file, and it can be stored on the HDFS. The benefits of storing on the HDFS is that the data are stored on the distribution system, which performs better than the local file. HDFS is used to store both edge and vertex tables. As previously mentioned, the parquet file format stores data using DataFrame in the Python programming language as following codes.

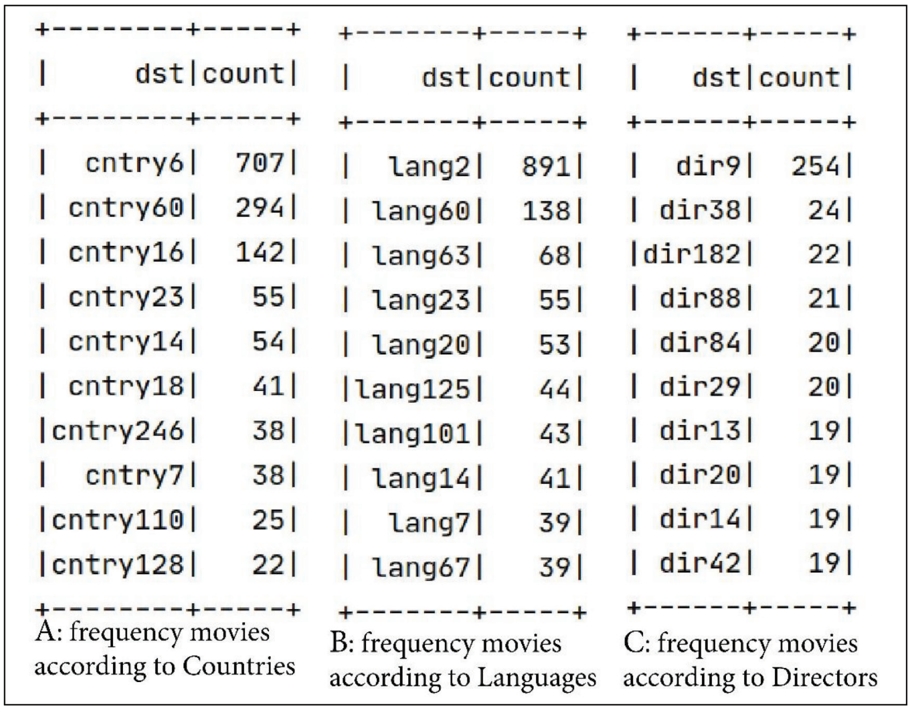
```
e.write.mode('append').format("parquet").save(edgefile)
v.write.mode('append').format("parquet").save(vertexfile)
```

8.6. Data Visualization and Analysis

GraphX data can be retrieved using different techniques in Python, The code provided implements a filtering mechanism to display specific data, as illustrated in Figure 16.

```
g.edges.filter("relationship = 'Country'").groupBy('dstid').count().sort(desc('count')).
show(10)
g.edges.filter("relationship = 'Language'").groupBy('dstid').count().sort(desc('count')).
show(10)
g.edges.filter("relationship = 'Director'").groupBy('dstid').count().sort(desc('count')).
show(10)
```

The number of movies is counted according to each country, and the result is 707 movies for the United States, which is "cntry6". According to the used dataset, the United States is the largest film-producing country. Also, the second largest country is India which is "cntry60" as shown in Figure 12-A. Furthermore, the analysis reveals that the majority of languages utilised in movies is English, denoted as "lang2," followed by Hindi, represented as "lang60," as depicted in Figure 12-B. Similar patterns can be observed for directors, as illustrated in Figure 12-C.



```
+--------+-----+
|     dst|count|
+--------+-----+
|  cntry6|  707|
| cntry60|  294|
| cntry16|  142|
| cntry23|   55|
| cntry14|   54|
| cntry18|   41|
|cntry246|   38|
|  cntry7|   38|
|cntry110|   25|
|cntry128|   22|
+--------+-----+
```

A: frequency movies according to Countries

```
+-------+-----+
|    dst|count|
+-------+-----+
|  lang2|  891|
| lang60|  138|
| lang63|   68|
| lang23|   55|
| lang20|   53|
|lang125|   44|
|lang101|   43|
| lang14|   41|
|  lang7|   39|
| lang67|   39|
+-------+-----+
```

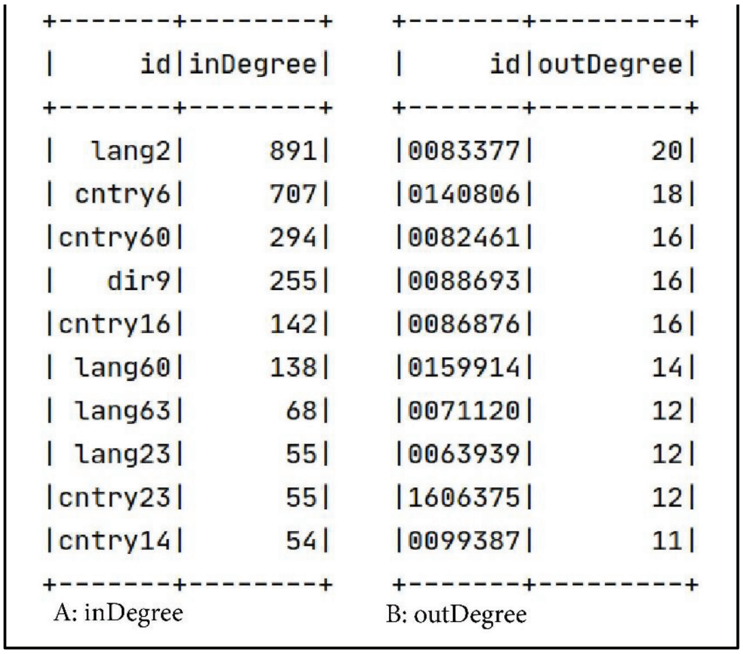
B: frequency movies according to Languages

```
+------+-----+
|   dst|count|
+------+-----+
|  dir9|  254|
| dir38|   24|
|dir182|   22|
| dir88|   21|
| dir84|   20|
| dir29|   20|
| dir13|   19|
| dir20|   19|
| dir14|   19|
| dir42|   19|
+------+-----+
```

C: frequency movies according to Directors

Figure 12. Frequency of movies according to relationship

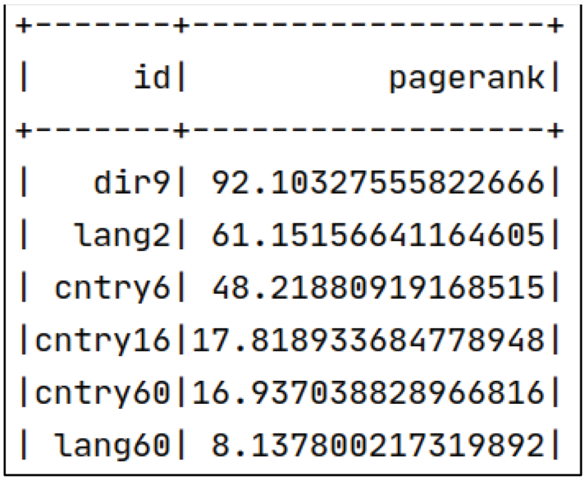
The inDegree algorithm, displayed in Figure 13:A, measures how many edges enter a vertex. The term "outdegree" also refers to the number of edges that branch off from the vertex, as seen in Figure 13:B. The PageRank technique can be used to measure the significant vertex nodes from the dataset webpage, as shown in Figure 14.



```
+-------+--------+    +-------+---------+
|     id|inDegree|    |     id|outDegree|
+-------+--------+    +-------+---------+
|  lang2|     891|    |0083377|       20|
| cntry6|     707|    |0140806|       18|
|cntry60|     294|    |0082461|       16|
|   dir9|     255|    |0088693|       16|
|cntry16|     142|    |0086876|       16|
| lang60|     138|    |0159914|       14|
| lang63|      68|    |0071120|       12|
| lang23|      55|    |0063939|       12|
|cntry23|      55|    |1606375|       12|
|cntry14|      54|    |0099387|       11|
+-------+--------+    +-------+---------+
```

A: inDegree
B: outDegree

Figure 13. Indegree and outdegree of dataset



```
+-------+-----------------+
|     id|         pagerank|
+-------+-----------------+
|   dir9| 92.1032755822666|
|  lang2| 61.1515664116460 5|
```

Figure 14. PageRank

9. Conclusion

The future of the World Wide Web (WWW) is going in the direction of linked data and the Semantic Web. They are rich data sources because they continuously produce enormous amounts of data. Many challenges are faced when storing big and heterogeneous data because big Semantic Web data can have the property of variety, velocity, and volume. Semantic Web data can be represented as a graph because the Semantic Web can have three main parts: subject, predicate, and object. Furthermore, big data technologies such as Spark and Hadoop are used to store big Semantic Web data. One of the parts of Spark is GraphX which can accept graph data.

Combining Semantic Web and big data technology addresses data integration, scalability, complex querying, data quality, and knowledge discovery challenges. It facilitates the effective integration of disparate sources, the scalable processing of massive datasets, optimized querying, enhanced data quality, and extracting information from linked data.

This paper demonstrates the benefits of combining the Semantic Web and big data technologies to solve the problem of storing and analyzing big Semantic Web data. Semantic Web graphs can be converted to GraphX data format. Also, it can be partitioned into two different tables, one for storing the data's vertices and edges. Also, for distributing data, Spark can be used instead of MapReduce for data processing and analysis because it is faster and easier than MapReduce. Moreover, for storing data, Hadoop distributed file system was utilized. A sample dataset in this paper was extracted from DBpedia using the SPARQL query. The data size has been reduced by around half when compared to the RDF file; therefore, when the data convert to GraphX, the file is significantly smaller. It affects the result of the data analysis. Notably, the GraphX algorithm is directly used with GraphX, such as the outdegree, indegree, and PageRank algorithm.

In the future, we want to use big data technologies to evaluate and process various linked data and Semantic Web sources. Our research agenda focuses on large-scale Semantic Web data mining using Semantic Web technologies, big data frameworks, and sophisticated data mining methods. We want to reveal significant insights and patterns hiding inside the sizeable, linked web of data by combining these robust technologies, contributing to rapidly growing knowledge discovery in this exciting and expanding domain.

References

Agathangelos, G., Troullinou, G., Kondylakis, H., Stefanidis, K., & Plexousakis, D. (2018a). Incremental data partitioning of RDF data in SPARK. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11155 LNCS*, 50-54. https://doi.org/10.1007/978-3-319-98192-5_10

Agathangelos, G., Troullinou, G., Kondylakis, H., Stefanidis, K., & Plexousakis, D. (2018b). RDF query answering using apache spark: Review and assessment. *Proceedings - IEEE 34th International Conference on Data Engineering Workshops, ICDEW 2018*, 54-59. https://doi.org/10.1109/ICDEW.2018.00016

Azzedin, F. (2013). Towards a scalable HDFS architecture. In *2013 International Conference on Collaboration Technologies and Systems (CTS)* (pp. 155-161). IEEE.

Baby Nirmala, M., & Sathiaseelan, J. G. R. (2021). *An Enhanced Approach Of RDF Graph Data In-Memory Processing For Social Networks With Performance Analysis*, *18*(6). http://www.webology.org

Banane, M., & Belangour, A. (2019). RDFSpark: a new solution for querying massive RDF data using Spark. *International Journal of Engineering &Technology, 8*(3), 288-294.

Banane, M., & Belangour, A. (2020). A new system for massive RDF data management using big data query languages pig, hive, and Spark. *International Journal of Computing and Digital Systems*, *9*(2), 259-270. https://doi.org/10.12785/IJCDS/090211

Bansod, A. (2015). Efficient Big Data Analysis with Apache Spark in HDFS. *International Journal of Engineering and Advanced Technology*, *4*(6), 313-316.

Berners-Lee, T., Hendler, J., Lassila, O. (2001). *The Semantic Web. Scientific American, 284*(5), 34.

Donvito, G., Marzulli, G., & Diacono, D. (2014). Testing of several distributed file-systems (HDFS, Ceph and GlusterFS) for supporting the HEP experiments analysis. *Journal of Physics: Conference Series*, *513*(TRACK 4). https://doi.org/10.1088/1742-6596/513/4/042014

Gopalani, S., & Arora, R. (2015). Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means. *International Journal of Computer Applications, 113*(1), 8-11. https://doi.org/10.5120/19788-0531

Hakimov, S. T. A. D. (2013). Semantic Question Answering System over Linked Data using Relational Patterns. *EDBT/ICDT*, 83-88. https://doi.org/10.1145/2457317.2457331

Kulcu, S., Dogdu, E., & Ozbayoglu, A. M. (2016). A survey on semantic web and big data technologies for social network analysis. *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, 1768-1777. https://doi.org/10.1109/BigData.2016.7840792

Lahore, Z. A. (2016). Semantic Web Mining in E-Commerce Websites. In *International Journal of Computer Applications, 137*(2), 1-4. https://doi.org/10.5120/ijca2016908748

Lim, S. H., Lee, S., Ganesh, G., Brown, T. C., & Sukumar, S. R. (2015). Graph Processing Platforms at Scale: Practices and Experiences. *ISPASS 2015 - IEEE International Symposium on Performance Analysis of Systems and Software*, *2*, 42-51. https://doi.org/10.1109/ISPASS.2015.7095783

Mohammed, W., Mohammed, S., & Khalil Jumaa, A. (2021). Storage, Distribution, and Query Processing RDF Data in Apache Spark and GraphX: A Review Survey. *International Journal of Mechanical Engineering*, *6*(3).

Mohammed, W., & Saraee, M. (2016). Sematic Web Mining Using Fuzzy C-means Algorithm. *British Journal of Mathematics & Computer Science*, *16*(4), 1-16. https://doi.org/10.9734/bjmcs/2016/25471

Naacke, H., Amann, B., & Curé, O. (2017). SPARQL graph pattern processing with apache spark. *5th International Workshop on Graph Data Management Experiences and Systems, GRADES 2017 - Co-Located with SIGMOD/PODS 2017*. https://doi.org/10.1145/3078447.3078448

Omar, H. K., & Jumaa, A. K. (2019). Big Data Analysis Using Apache Spark MLlib and Hadoop HDFS with Scala and Java. *Kurdistan Journal of Applied Research*, *4*(1), 7-14. https://doi.org/10.24017/science.2019.1.2

Ramalingeswara Rao, T., Ghosh, S. K., & Goswami, A. (2021). Mining user-user communities for a weighted bipartite network using Spark GraphFrames and Flink Gelly. *Journal of Supercomputing*, *77*(6), 5984-6035. https://doi.org/10.1007/s11227-020-03488-4

Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. In *International Journal of Data Science and Analytics, 1*(3-4), 145-164. https://doi.org/10.1007/s41060-016-0027-9

Yu, J., Zhang, Z., & Sarwat, M. (2019). Spatial data management in apache spark: the GeoSpark perspective and beyond. *GeoInformat*ica, 23(1), 37-78. https://doi.org/10.1007/s10707-018-0330-9