Efficiently Processing and Storing Library Linked Data using Apache Spark and Parquet

Kumar Sharma,
Ujjal Marjit,
and Utpal Biswas

ABSTRACT

Resource Description Framework (RDF) is a commonly used data model in the Semantic Web environment. Libraries and various other communities have been using the RDF data model to store valuable data after it is extracted from traditional storage systems. However, because of the large volume of the data, processing and storing it is becoming a nightmare for traditional data-management tools. This challenge demands a scalable and distributed system that can manage data in parallel. In this article, a distributed solution is proposed for efficiently processing and storing the large volume of library linked data stored in traditional storage systems. Apache Spark is used for parallel processing of large data sets and a column-oriented schema is proposed for storing RDF data. The storage system is built on top of Hadoop Distributed File Systems (HDFS) and uses the Apache Parquet format to store data in a compressed form. The experimental evaluation showed that storage requirements were reduced significantly as compared to Jena TDB, Sesame, RDF/XML, and N-Triples file formats. SPARQL queries are processed using Spark SQL to query the compressed data. The experimental evaluation showed a good query response time, which significantly reduces as the number of worker nodes increases.

INTRODUCTION

More and more organizations, communities, and research-development centers are using Semantic Web technologies to represent data using RDF. Libraries have been trying to replace the cataloging system using a linked-data technique such as BIBFRAME.[1] Libraries have received much attention on transitioning MARC cataloging data into RDF format.[2] Data stored in various other formats such as relational databases, CSV, and HTML have already begun their journey toward the open-data movement.[3] Libraries have participated in the evolution of Linked Open Data (LOD) to make data an essential part of the web.[4] Various researchers have explored areas related to library data and linked data. In particular, transitioning legacy library data into linked data has dominated most of the research works. Other areas include researching the impact of linked library data, investigating how privacy and security can be maintained, and exploring the potential effects of having open linked library data. Obviously, a linked-data approach for publishing data on the web brings many benefits to libraries. First, once isolated library data currently stored using traditional cataloging systems (MARC) becomes a part of the web, it can be shared, reused, and consumed by web users.[5] This promotes the cross-domain sharing of knowledge hidden in the library data, opening the library as a rich source of information. Online library users can share more information using linked library resources since every library

Kumar Sharma (kumar.asom@gmail.com) is Research Scholar, Department of Computer Science and Engineering; **Ujjal Marjit** (marjitujjal@gmail.com) is System-in-Charge, Center for Information Resource Management (CIRM); and **Utpal Biswas** (utpal01in@yahoo.com) is Professor, Department of Computer Science and Engineering, the University of Kalyani, India.

resource is crawlable on the web via Uniform Resource Identifiers (URI). Most importantly, library data benefits from linked-data technology's real advantages, such as interoperability, integration with other systems, data crosswalks, and smart federated search.[6]

Numerous approaches have evolved for making the vision of the Semantic Web a success. No doubt, they have succeeded in making the library a part of the web, but there remain issues related to library big data. The term *big data* refers to data or information that cannot be processed using traditional software systems.[7] The volume of such data is so large that it requires advanced technologies for processing and storing the information. Libraries also have real concerns with large volumes of data during and after the transition to linked data. The main challenges are in processing and storage. During conversion from library data to RDF, the process can become stalled because of the large volumes of data. Once the data is successfully converted into RDF formats, there are storage issues. Finally, even if the data is somehow stored using common RDF triple stores, it is difficult to retrieve and filter. This is a challenging problem that every librarian must give attention to. Librarians should know the real nature of library big data, which causes problems in analyzing data and decision making. Librarians must also know the technologies that can resolve these issues.

The rate of data generation and the complexity of the data itself are constantly increasing. Traditional data-management tools are becoming incapable of managing the data. That is why the definition of big data has been characterized by five *V*s—volume, velocity, variety, value, and veracity.[8]

- Volume is the amount of the data.
- Velocity is the data-generation rate (which is high in this case).
- Variety refers to the heterogeneous nature of the data.
- Value refers to the actual use of the data after the extraction.
- Veracity is the quality or trustworthiness of the data.

To handle the five Vs of big data, distributed technologies such as commodity hardware, parallel processing frameworks, and optimized storage systems are needed. Commodity hardware reduces the cost of setting up a distributed environment and can be managed with very limited configurations. A parallel processing system can process distributed data in parallel to reduce processing time. An optimized storage system is required to store the large volume of data, supporting scalability to accommodate more data on demand. With these library requirements to tackle the challenges posed by library big data, a distributed solution is proposed. This approach is based on Apache Hadoop, Apache Spark, and a column-oriented storage system to process large-size data and to store the processed data in a compressed form. Bibliographic RDF data from British National Library and the National Library of Portugal have been used for this experiment. These bibliographic data are processed using Apache Spark and stored using Apache Parquet format. The stored data can be queried using SPARQL queries for which Spark SQL is used to execute queries.

Given an existing RDF dataset, we designed a schema for storing RDF data using a column-oriented database. Using column-oriented design with Apache Parquet and Spark SQL as the query

https://doi.org/10.6017/ital.v37i3.10177

processor, a distributed RDF storage system was implemented that can store any amount of RDF data by increasing the number of distributed nodes as needed.

LITERATURE REVIEW

While big data continues to rise, library data are still in traditional storage systems isolated from the web. To continue working with the web, libraries must redesign the way they format data and contribute toward the web of data. To serve library data to other communities, libraries must integrate their data with the web. Attempts to do this have been made by several researchers. The task of integration cannot be achieved by only librarians; rather, it requires a team of experts in the field of library and information technology. The advanced way for integrating resources is with linked-data technology by assigning URIs to every piece of library data. With this goal, there exist various projects related to the convergence of library data and linked data. One of these, BIBFRAME, is an initiative to transition bibliographic resources into linked-data representation. BIBFRAME aims to replace traditional cataloging standards such as MARC and UNIMARC using the concept of publishing structured data on the web. MARC formats cannot be exchanged easily with nonlibrary systems. The MARC standard also suffers from inconsistencies, errors, and inability to express relationships between records and fields within the record. That is why mostly bibliographic resources stored in MARC standards are targeted for conversion.[9] Other works include the open-data initiative from the British National Library, library catalog to linked open-data conversion, exposing library data as linked data, and building a knowledge graph to reshape the library staff directory.[10]

Linked data is fully dependent on RDF. RDF reveals graph-like structures where resources are linked with one another. Thus, RDF can improve on MARC standards because of its strong ability to link related resources. This system of revealing everything as a graph helps in building a network of library resources and other data on the web. This also makes for fast search functionality. In addition, searching a topic or book could bring similar graphs from other library resources, leading to the creation of linked-data service.[11] Such a service has been implemented by the German National Library to provide bibliographic and authority data in RDF format, by the Europeana Linked Open Data with access to open metadata on millions of books and multimedia data, and by the Library of Congress Linked Data Service.[12]

There is less discussion of library big data. Though big data in general is in active research, the library domain has received much less attention than the broader concept of big data and its challenges. This could be because most of librarians working with linked data are from nontechnical backgrounds. Now is the right time for libraries to give priority to adopting big data technologies to overcome challenges posed by big data. Wang et al. have discussed library big data issues and challenges.[13] They made some statements about whether library data belongs to the big data category. Obviously, library data belongs to big data since it fulfills some of the characteristics of big data, such as volume, variety, and velocity. Wang et al. also raise some of libraries' challenges related to library big data, such as lacking teams of experts, inability to adopt big data due to budgetary issues, and technical challenges. Finally, they point out that to take advantage of the web's full potential, library data must be transformed into a format that can be accessible beyond the library using technologies like Semantic Web and linked data.

The web has already started its work related to big-data challenges. Libraries need to transition their data into an advanced format with the ability to handle big-data issues. The main problems

related to library big data happen at data transformation and storage. To store and retrieve large amounts of data, we need commodity hardware that can handle trillions of RDF triples, requiring terabytes or petabytes of disk space. As of now, there are Semantic Web frameworks such as Jena and Sesame to handle RDF data, but these frameworks are not scalable for large RDF graphs.[14] Jena is a Java-based framework for building Semantic Web and linked-data applications. It is basically a Semantic Web programming framework that provides Java libraries for dealing with RDF data. Jena TDB is the component of Jena for storing and querying RDF data.[15] It is designed to work in a single-node environment. Sesame is also a Semantic Web framework for processing, storing, and querying RDF data. Basically, Sesame is a web-based architecture for storing and querying RDF data as well as schema information.[16]

BACKGROUND

This section briefly describes the structure of RDF triples, Apache Spark along with its features and column-oriented database system, and Apache Parquet.

Structure of RDF Triples

RDF is a schema-less data model. It implies that the data is not fixed to a specific schema, so it does not need to conform to any predefined schema. Unlike in relational tables, where we define columns during schema definition and those columns must contain the required type of data, in RDF we can have any number of properties and data using any kind of vocabulary. We only need vocabulary terms to embed properties. The vocabulary is created using domain ontology, which represents the schemas. To describe library resources we need a library-domain ontology. For example, to define a book and its properties one can use the BookOnt ontology.[17] BookOnt is a book-structure ontology designed for an optimized book search and retrieval process. However, it is not mandatory to use existing ontology and all the properties defined under it. We can use terms from a newly created ontology or mixed ontologies with required properties. RDF represents resources in the form of subject, predicate, and object. The subject is the resource being described, identified by a URI. This subject can have any number of property-value pairs. This way representation of a resource is called knowledge representation, where everything is defined as a knowledge in the form of entity attribute value (EAV). In RDF, the basic unit of information is a triple T, such that T = {Subject, Predicate, Object}. Such information when stored on disk is called a triplestore. The collection of RDF triples is called an RDF database.

An RDF database is specially designed to store linked data to make the web more useful by interlinking data from different sources in a meaningful way. The real advantage of RDF is its support of the common data model. RDF is the standard way for publishing meaningful data on the web, and this is backed by linked data. Linked data provides some rules about how data can be published on the web by following the RDF data model.[18] With such a common data model, one can integrate data from any sources by inserting new property-value pairs without altering database schema. Another important purpose of RDF is to provide resources to be processable by software agents on the web.

RDF triples are of two types: literal triples and linked triples. Literal triples consist of a URI-referenced subject and a literal object (scalar value) joined by a predicate. In linked triples, both the subject and the object consist of URIs linking by the predicate. This type of linking is called RDF link, which is the basis for interlinking the resources.[19] RDF data are queried using the SPARQL query language.[20] SPARQL is a graph-matching query language and is used to retrieve

https://doi.org/10.6017/ital.v37i3.10177

triples from the triple store. The SPARQL queries are also called semantic queries. Like SQL queries, SPARQL also finds and retrieves the information stored in the triplestore. A SPARQL query is composed of five main components:[21]

- the prefix declaration part is used to abbreviate the URIs;
- the dataset definition is used to specify the RDF dataset from which the data is to be fetched;
- the result clause is used to specify what information is needed to be fetched, which can be SELECT, CONSTRUCT, DESCRIBE, and ASK;
- the query pattern is used to specify the search conditions; and
- the query modifiers are used to rearrange query results using ORDER BY, LIMIT etc.

Hadoop and MapReduce

Hadoop is open-source software that supports distributed processing of large datasets on machine clusters.[22] Two core components—Hadoop Distributed File System (HDFS) and MapReduce—make distributed storage and computation of processing jobs possible.[23] HDFS is the storage component, whereas MapReduce is a distributed data-processing framework, the computational model of Hadoop based on Java. The MapReduce algorithm consists of two main tasks: map and reduce. The map task takes a set of data as input and produces another set of data with individual components in the form of key/value pairs or tuples. The output of the map task goes to the reduce task, which combines common key/value pairs into a smaller set of tuples. HDFS and MapReduce are based on driver/worker architecture consisting of driver and worker nodes having different roles. An HDFS driver node is called the Name-Node while the worker node is called the Data-Node. The Name-Node is responsible for managing names and data blocks. Data blocks are present in the Data-Nodes. Data-Nodes are distributed across each machine, responsible for actual data storage. Similarly, the MapReduce driver node is called the Job-Tracker and the worker node is called the Task-Tracker. Job-Tracker is responsible for scheduling jobs on Task-Trackers. Task-Tracker again is distributed across each machine along with the Data-Nodes, responsible for processing map and reducing tasks as instructed by the Job-Tracker.

The concept of Hadoop implies that the set of data to be processed is broken into smaller forms that can be processed individually and independently. This way, tasks can be assigned to multiple processors to process the data, and eventually it becomes easy to scale data processing over multiple computing nodes. Once a MapReduce program is written, the program can be scaled to run over thousands of machines in a cluster.

Spark and Resilient Distributed Datasets (RDD)

Apache Spark is an in-memory cluster computing platform, which is a faster batch-processing framework than MapReduce. More importantly, it supports in-memory processing of tasks along with data, so querying data is much faster than disk-based engines. The core of Spark is the Resilient Distributed Dataset (RDD). RDD is a fundamental data structure of Spark that holds a distributed collection of data where data cannot be modified. Rather, data modification yields another immutable collection of data (or RDD). This process is called RDD transformation. For example, figure 1 depicts an example of RDD transformation. The distributed processing and

transformation of data is managed by RDD. RDDs are fault-tolerant, meaning that the lost data is recoverable using lineage graph of RDDs.[24] Spark constructs a Direct Acyclic Graph (DAG) of a sequence of computations that needed to be performed on data. Spark has the most powerful computing engine that allows most of the computations in multistage memory. Because of this multistage in-memory computation engine, it provides better performance at reading and writing data than the MapReduce paradigm.[25] It aims at speed, ease of use, extensibility, and interactive analytics.

Spark relies on concepts such as RDD, DAG, Spark Context, Transformations, and Actions. Spark Context is an execution environment in which RDDs and broadcasting variables can be created. Spark Context is also called the master of a Spark application and allows accessing the cluster through a resource manager. Data transformation happens in the Spark application when the data is loaded from a data-store into RDDs and some filter or map functions are performed to produce a new set of RDDs. When the set of computations is created, forming a DAG, it does not perform any execution; rather, it prepares for execution in the end, like a lazy loading process. Some examples of actions are data extraction or collection and getting the count of words. Transformations are the sequence of events, and action is the final execution of the underlying logic.

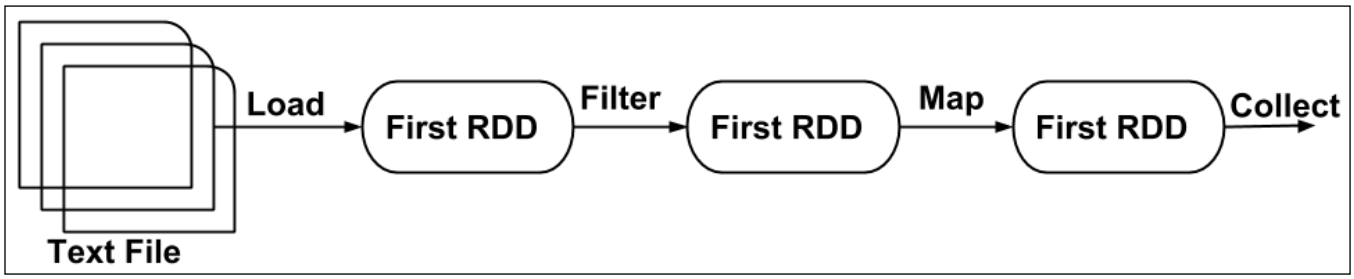

Figure 1. RDD transformations.

The execution model of Spark is shown in figure 2. The execution model is based on the driver/worker architecture consisting of the driver and the worker processes. The driver process creates the Spark context and schedules tasks based on the available worker nodes. Initially, the master process must be started, then creating worker nodes follows. The driver takes the responsibility of converting a user's application into several tasks. These tasks are distributed among the workers. The executors are the main components of every Spark application. Executors actually perform data processing, reading and writing data to the external sources and the storage system. The Spark manager is responsible for resource allocation and deallocation to the Spark job. Basically, Spark is only a computation model. It is not related to storage of data, which is a different concept. It only helps in computations and data analytics in a distributed manner. For distributed execution, the task is distributed among the connected nodes so that every node can perform tasks at the same time; it performs the desired operation and notifies the master upon completion of the task.

https://doi.org/10.6017/ital.v37i3.10177

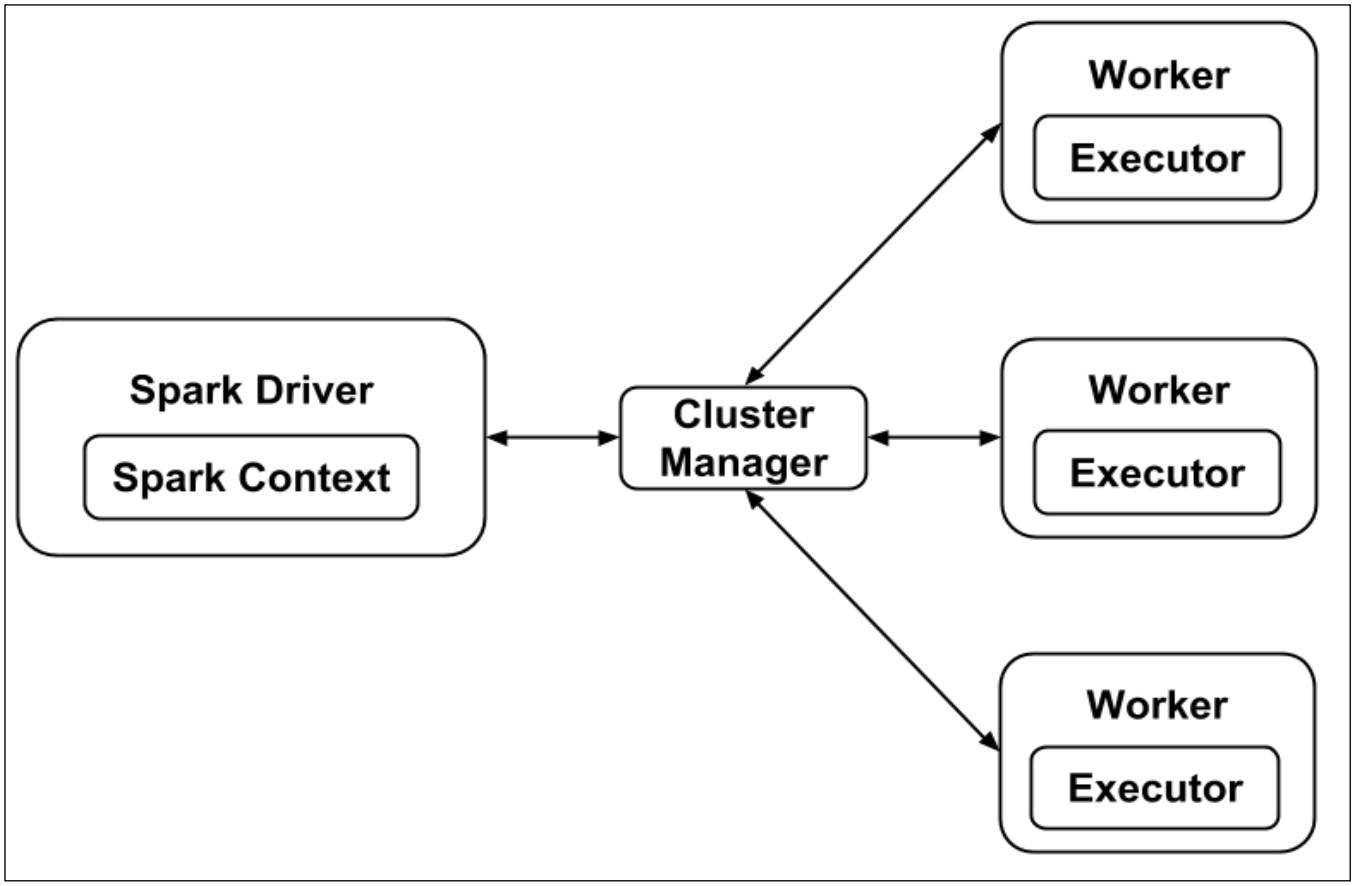

Figure 2 Execution model of Spark.

In MapReduce, read/write operations happen between disk and memory, making job computation slower than Spark. RDDs resolve this by allowing fault-tolerant, distributed, in-memory computations. In RDD, the first load of data is read from disk and then a write-to-disk operation may take place depending upon the program. The operations between first read and last write happen in memory. Data on RDDs are lazily evaluated, i.e., during RDD transformations, data will not take part until any action is called on the final RDD, which triggers the job execution. The chain of RDD transformations creates dependencies between RDDs. Each dependency has a function for calculating its data and a pointer to its parent RDD. Spark divides RDD dependencies into stages and tasks, then it sends them to workers for execution. Hence, an RDD does not actually hold the data; rather, it either loads data from disk or from another RDD and performs some actions on the data for producing results. One of the important features of RDD is its fault tolerance, because of which it can retain and recompute any of the unsuccessful partitions due to node failures. RDDs have built-in methods for saving data into files. For example, the RDD calls on saveAsTextFile(), its data are written on the specified text file line by line. There are numerous options for storing data in different formats, such as JSON, CSV, sequence files, and object files. All these file formats can be saved directly into HDFS or normal file systems.

Spark SQL and Dataframe

Spark SQL is a query interface for processing structured data using SQL style on the distributed collection of data. That means it is used for querying structured data stored in HDFS (like Hive) and Parquet. Spark SQL runs on top of Spark as a library and provides higher optimization. The

Spark dataframe is an API (application programming interface) that can perform relational operations on RDDs and external data sources such as Hive and Parquet. Like RDDs, a Spark dataframe is also a collection of structured records that can be manipulated by Spark SQL. It evaluates operations lazily to perform relational optimizations.[26] A dataframe is created using RDDs along with the schema information. For example, the Java code snippet below creates a dataframe using RDD and a schema called RDFTriple (rdf-triple schema will be discussed in the proposed approach).

```
JavaRDD<String> n_triples_ = marc_records.map(new TextToString());

JavaRDD<RDFTriple> rdf_triples = n_triples.map(new LinesToRDFFunction());

Dataset<Row> dataframe = sparkSession.createDataFrame(rdf_triples,
RDFTriple.class); dataframe.write().parquet("/full-path/RDFData.parquet");
```

The Spark dataframe uses memory management wisely by saving data in off-heap memory and provides an optimized execution plan. Conceptually, a dataframe is equivalent to the relational tables with richer optimization and supports SQL queries over its data. So, a dataframe is used for storing data into tables. Structured data from Spark dataframe can be saved into the Parquet file format as shown in the above code snippet.

Column-Oriented Database

A database is a persistent collection of records. These records are accessed via queries. The system that stores data and processes queries to retrieve data is called a database system. Such systems use indexes or iteration over the records to find the required information stored in the database. Indexes are an auxiliary, dictionary-like data structure that keeps indexes of individual records. Indexing is efficient in some cases, however, as it requires two lookup operations and it slows down the access time. Data scanning or iteration over each record resolves the query by finding the exact location of the records. It is inefficient when the size of the data is too large. As data-generation rate is increasing constantly, more and more data is going to be stored on the disk. For a fast-growing rate of data, we need a system that can adjust to more data than traditional storage systems and, at the same time, query-processing tasks should take less time. When the data gets too large, indexing and record scanning will be costly during querying. Hence, a satisfying solution is the columnar-storage system, which stores data by columns rather than by rows.[27]

A column-oriented database system stores data in corresponding columns, and each column is stored in a separate file into the disk. This makes data access time much quicker. Since each column is stored separately, any required data can directly be accessed instead of reading all the data. That means any column can be used as an index, making it auto-indexing. That is why the column-oriented representation is much faster than the row-oriented representation. Apart from this, data is stored in the compressed form. Each column is compressed using a different scheme. In the column-oriented database, the compression is always efficient as all the values belong to the same data type. Hence, column-oriented databases require less disk space, as they do not need additional storage for indexes since the data is stored within the indexes themselves. Consider an example where a database table named “Book” consisting of columns “BookID,” “Title,” and “Price.” Following a column-oriented approach, all the values for BookID are stored together under the “BookID” column, all the values for Title are stored together under “Title” column. and so on as shown in Figure 3.

https://doi.org/10.6017/ital.v37i3.10177

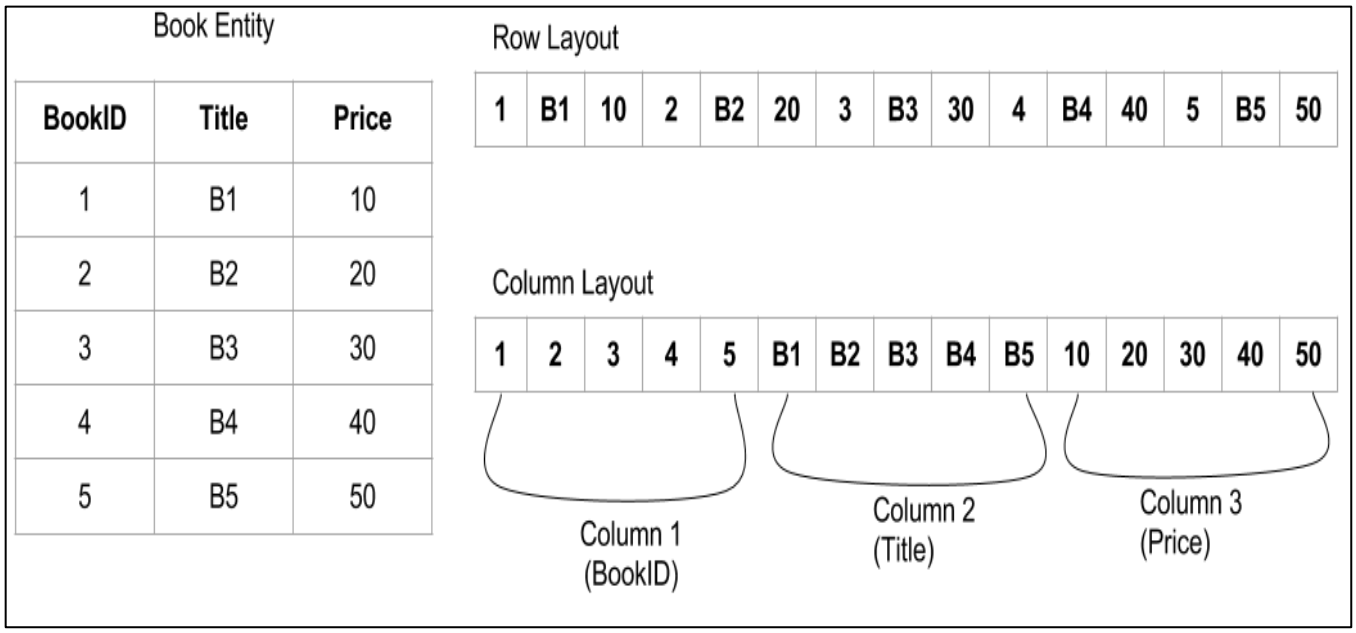

Figure 3 An example of an entity and its row and column representation.

Apache Parquet

Parquet is a top-level Apache project that stores data in column-oriented fashion, highly compressed and densely packed in the disk.[28] It is a self-describing data format that embeds schema within the data itself. It supports efficient compression and encoding schemes that allows lowering data-storage costs and maximizes the effectiveness of querying data. Parquet has added advantages, such as limiting the I/O operation and storing data in compressed form using the Snappy method developed by Google and used in its production environment. Hence it is designed especially for space and query efficiency.

Snappy aims at compressing petabytes of data in minimal amounts of time, and especially aims for resolving big data issues.[29] The data compression rate is more than 250 MB/sec, and decompression rate is more than 500 MB/sec. These compression and decompression rates are for a single core of a system having a Core i7 processor in 64-bit mode. It is even faster than the fastest mode of zlib compression algorithm.[30]

Parquet is implemented using column-striping and assembly-language algorithms that are optimized for storing large data-blocks.[31] It supports nested data structures in which each value of the same column is stored in contiguous memory locations.[32] Apache Parquet is flexible and can work with many programming languages because it is implemented using Apache Thrift (https://thrift.apache.org/). A Parquet file is divided into row groups and metadata at the end of the file. Each row group is divided into column values (or column chunks), such as column 1, column 2, and so on as shown in figure 4. Each column value is divided into pages, and each page consists of the page header, repetition levels, definition levels, and values. The footer of the file contains various metadata, such as file metadata, column metadata, and page-header metadata. The metadata information is required to locate and find the values, just like indexing.

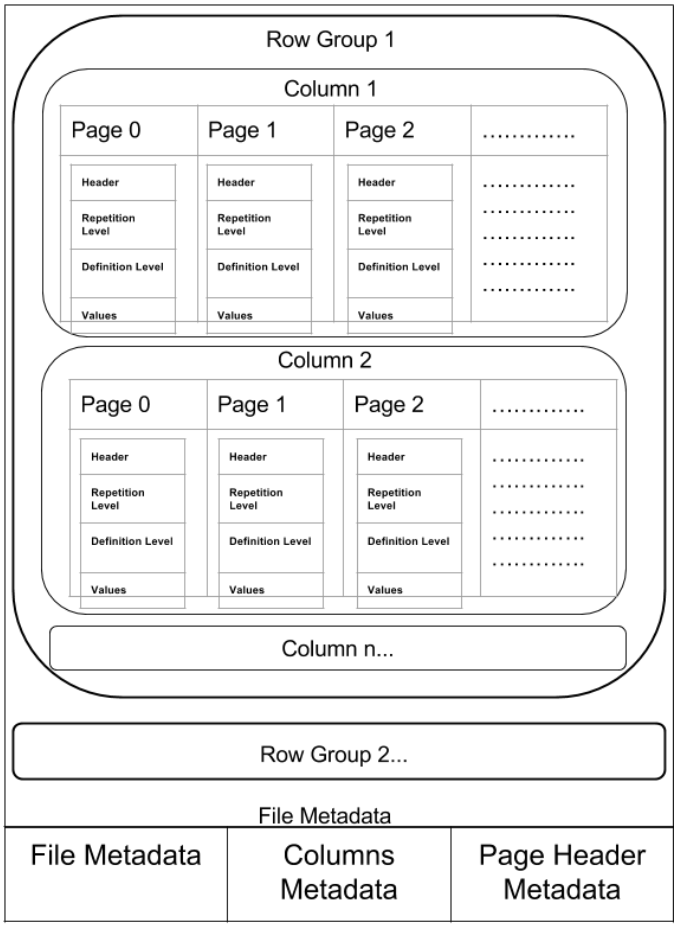

Figure 4 Parquet file structure.

THE PROPOSED APPROACH

The proposed approach relies on Spark's core APIs—RDD, Spark SQL, and Dataframe—which can operate on large datasets. RDD is used to load the initial data from the input file, process the data and transform them into triple structure. Spark dataframe is used to load the data from RDD into the triple structure and send the transformed RDF data into a Parquet file. Spark SQL is used to fetch the data stored in the Parquet file.

Processing RDF Data

Processing RDF data from large RDF/XML files requires breaking the file into smaller file components. General data-processing systems cannot handle large files because they face memory issues. At this stage, the proposed approach can process the data using an N-Triples file, hence individual RDF/XML files again need to be converted into the N-Triples file format. The process of breaking RDF/XML file into smaller file components and then converting them into N-Triples format depends upon the size of the input file. If it is not more than 500 MB then it is directly converted into N-Triples file format. Multiple RDF/XML files are converted into individual N-Triples file formats, which are again combined into one N-Triples file, as the proposed Spark application reads input from a single file.

https://doi.org/10.6017/ital.v37i3.10177

Schema to Store RDF Data

A simple RDF schema with three triple entities has been designed. This schema is an RDF triple view, which is the building block of the RDF storage schema proposed in this work. The RDF triple view is a simple Java class consisting of three attributes—subject, predicate, and object. Given an RDF dataset D, consisting of a set of RDF triples T, in either RDF/XML or N-Triples format, the dataset is transformed into a format that can be processed by a Spark application. Further, the dataset is transformed into a line-based format where the individual triple statement is placed in a line separated by a new-line (\n) character. A line contains three components—subject, predicate, and object separated by a space. Here each line is unique, using the combined information of subject, predicate, and object.

Given an RDF triple structure T_i, $T_i = (S_i, P_i, O_i)$ and $T_i \in T$, for each T an instance of RDF triple view is created to hold the triple information. The columnar schema organizes triple information into three components, storing each component separately as subject, predicate, and object columns (figure 5).

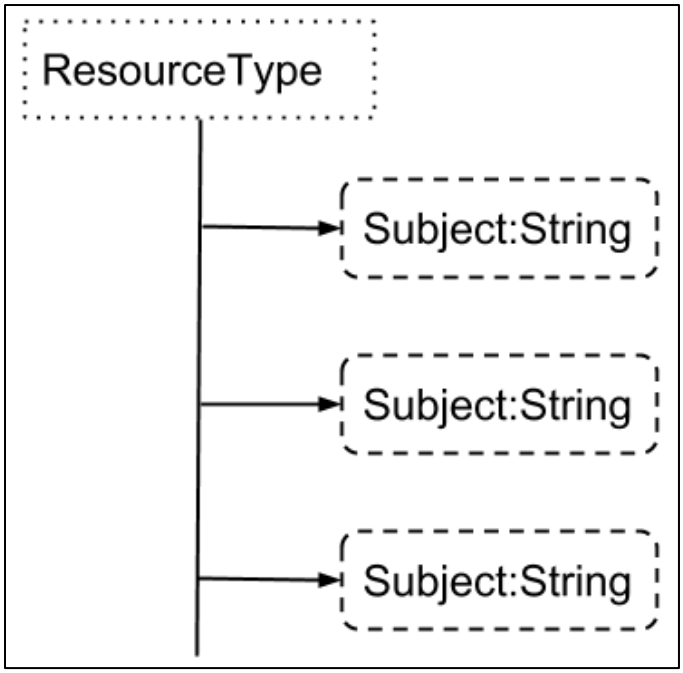

Figure 5. RDF Triple view.

RDF Storage

We store the RDF data based on RDF Triple view, which is the main schema for storing data in the triple representation. We do not need any indexing or additional information related to subject, predicate, or object to be stored on the disk. Since we can have any number of temporary dataframe tables in memory, join operations can be performed using these tables to filter the data. In the absence of expensive indexing and additional triple information, storage area can be reduced significantly. Apart from this, the compression technique used in Apache Parquet reduces lot more space than storing in other triple stores. In figure 6, we illustrate the data-storing process.

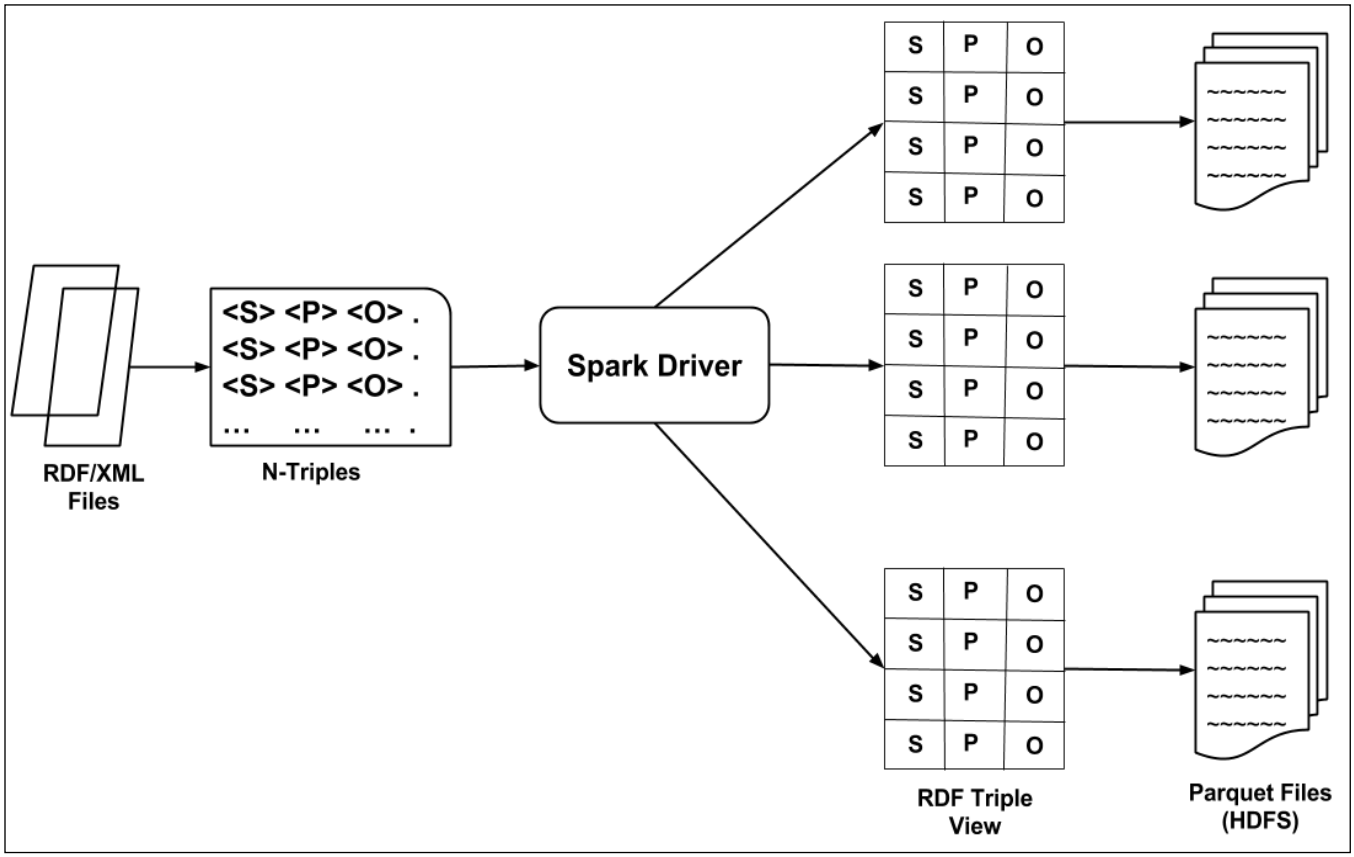

Figure 6. Data-storing process in HDFS.

The collection of triple instances is loaded into an RDD. At the end, the collection of triple instances is loaded into Spark dataframe. Spark dataframes are equivalent to the RDBMS tables and support both structured and unstructured data formats. Using a single schema, multiple dataframes can be used and can be registered as temporary tables in the memory, where high-level SQL queries can be executed on top of them. Here the concept of using multiple dataframes with a single schema is motivated to avoid joins and indexing. In the final step, the Spark dataframe is saved into HDFS files in the Parquet format. From the Parquet file, the data can be loaded back into dataframes in memory and queried using Spark SQL.

Fetching Data from Storage

Given an RDF dataset D, a SPARQL query Q, and a columnar-schema S, we use S to translate Q to Q' to perform queries on top of S. Here, the answer of query Q' on top of S is equal to the answer of Q on top of D. Query mappings M are used to transform SPARQL queries into Spark SQL queries.

For querying, first the data is loaded into a Spark dataframe from Parquet files. To query data using SPARQL, queries must follow basic graph patterns (BGP). A BGP is a set of triple patterns similar to an RDF triple (S, P, O) where any of S, P, and O can be query variables or literals. BGP is used for matching a triple pattern to an RDF graph. This process is called binding between query variables and RDF terms. The statements listed under the WHERE clause is known as BGP consisting of query patterns. For example, the query "SELECT ?name ?mbox WHERE {?x foaf:name ?name . ?x foaf:mbox ?mbox .}" has two query patterns. To evaluate the query containing two query patterns, one join is required. Based on the total number of query patterns,

https://doi.org/10.6017/ital.v37i3.10177

we need one less number of joins. That is, for *n* number of query patterns we need *n*-1 joins to resolve the values. Figure 7 illustrates the process of query execution.

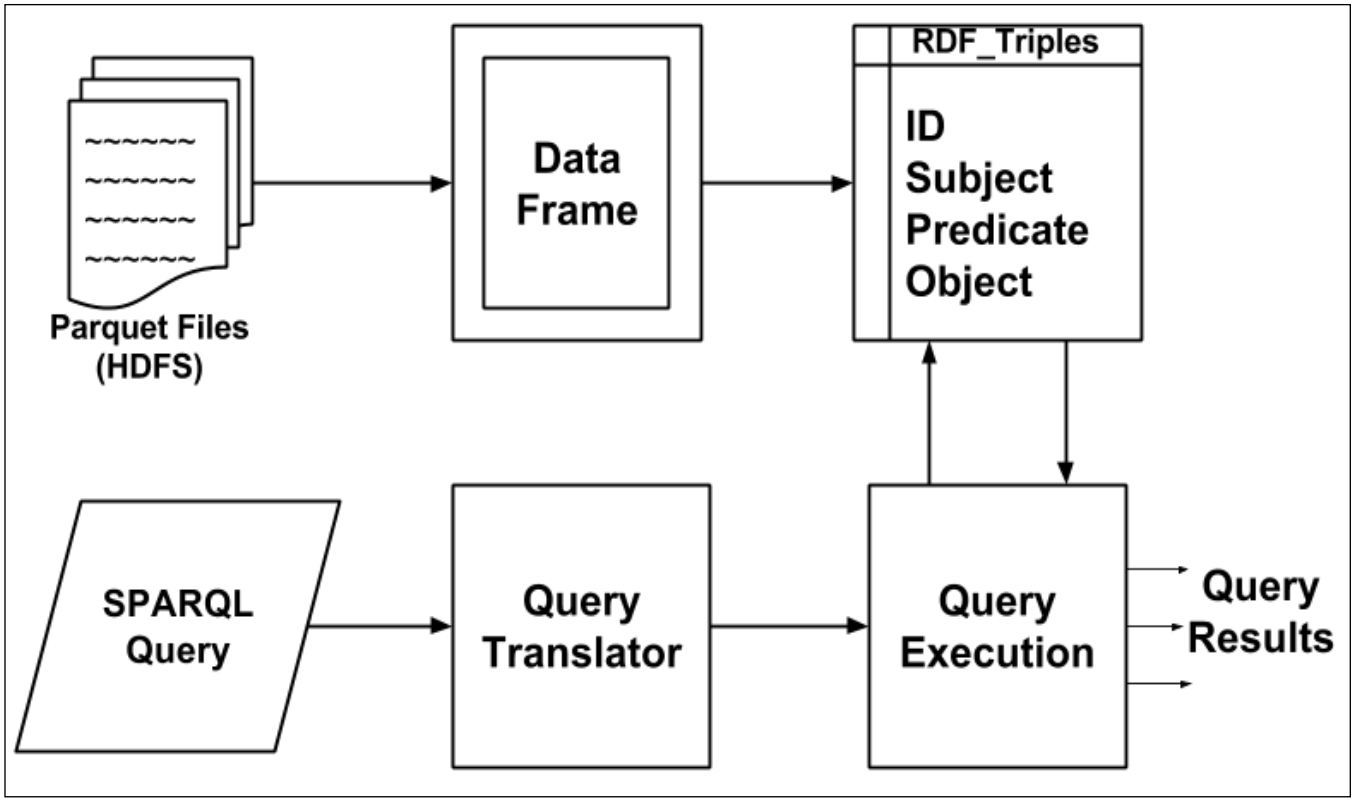

Figure 7. Process of query execution.

EVALUATION

To evaluate the proposed approach we compare the storage size with file-based storage systems such as N-Triples files and RDF/XML files. We also compare with standard triple stores such as Jena TDB and Sesame. The data-storing time is compared with Jena TDB, Sesame, and Parquet, having one, two, and three worker nodes respectively. Finally, for the purposes of the experiment, some SPARQL queries are selected and tested over RDF data stored in Parquet format into HDFS. The query performance is tested on the distributed system having one, two, and three worker nodes respectively. In the following subsections, we show the results for each of the above comparisons.

Datasets

For evaluation, we use two datasets. Dataset 1 contains bibliographic data from the National Library of Portugal (NLP) (http://opendata.bnportugal.pt/eng_linked_data.htm). From NLP, we choose the NLP Catalogue datasets in RDF/XML formats. The datasets are freely available to reuse and contain metadata information from NLP Catalogue, the National Bibliographic Database, the Portuguese National Bibliography, and the National Digital Library. The datasets are available as linked data, which were produced in the context of the European Library. The size of the RDF/XML file is 6.46 GB with more than 45 billion RDF triples.

Dataset 2 contains bibliographic data from the British National Library (https://www.bl.uk/bibliographic/download.html). From the British National Bibliography collection we choose the BNB LOD Books dataset. The datasets are publicly available and contain bibliographic records of different categories, such as books, locations, bibliographic resources, persons, organizations, and agents. The datasets are divided into sixty-seven files in RDF format. However, we combine them into one file in N-Triples format to fit the requirement of the large size of the input data. The combined file is 22.52 GB and contains more than 16 billion RDF resources in N-Triples format, making it suitable for the proposed approach. From this conversion, we get more than 150 billion RDF triples.

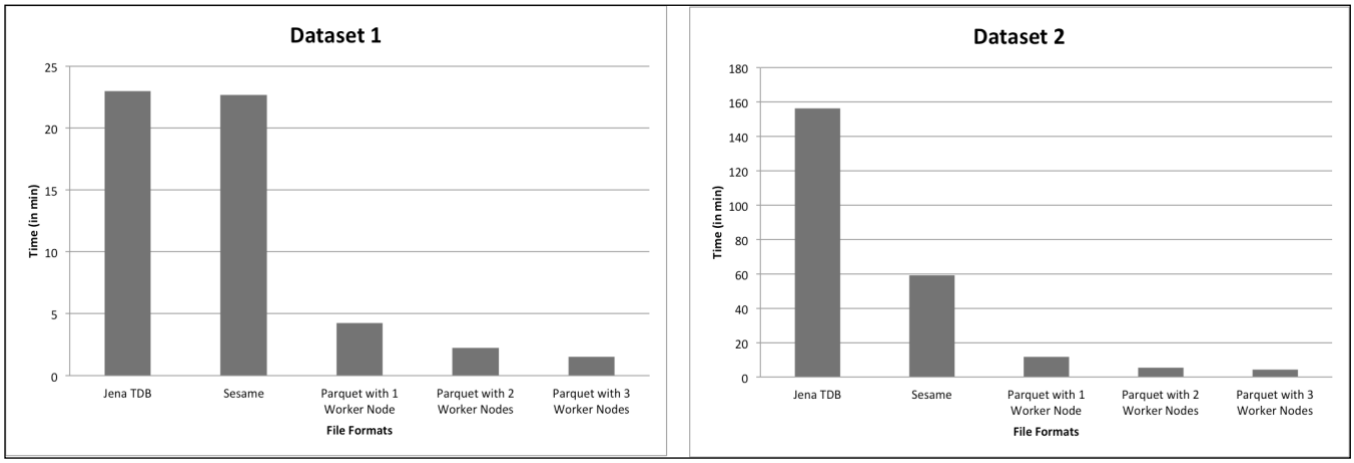

Figure 8. Data storage time for different file formats.

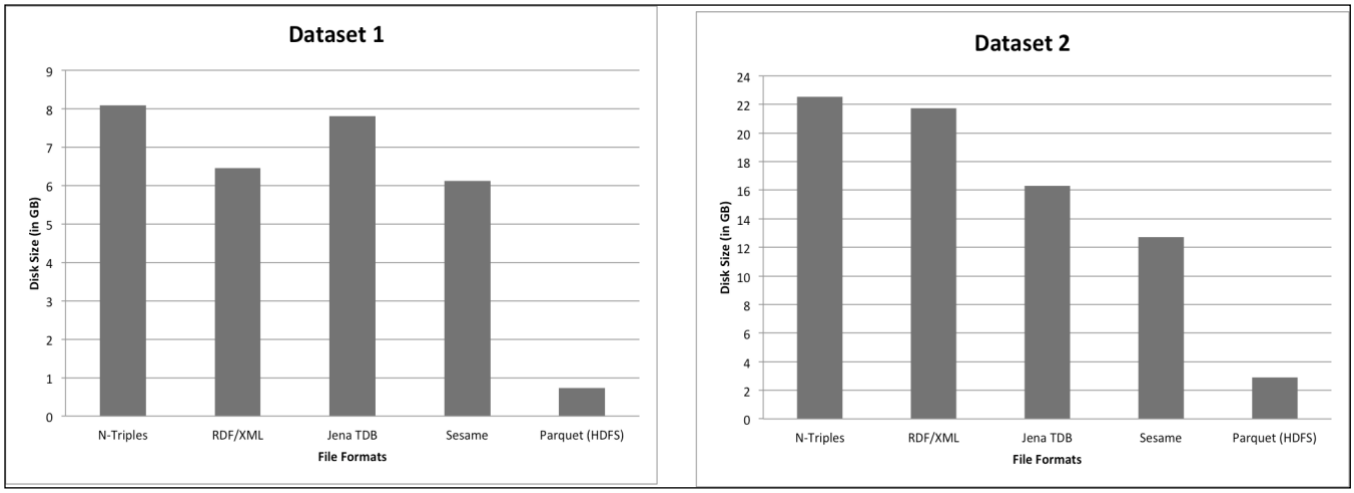

Figure 9. Disk size for different file formats.

Disk Storage

Figure 8 shows the data-storing time using Sesame, Jena TDB, and Parquet for the above two datasets. Data from raw RDF files are stored in Jena TDB and Sesame. Individual files are processed for storing into Jena TDB and Sesame to avoid memory overflow as Jena or Sesame models cannot load data at once from the large files. To store data in Parquet format we run the program separately on different worker nodes. Figure 9 presents the total disk size required for each of these file formats and triple stores for the two datasets.

https://doi.org/10.6017/ital.v37i3.10177

Query Performance

For testing, the SPARQL queries are converted manually at this stage. We run some of the selected queries over bibliographic RDF data stored in Parquet file format in HDFS. We run the following type of queries on worker nodes 1, 2 and 3 respectively. The queries are listed below:

Q1) The first query is to fetch the count of RDF triples present in the storage.

Query: SELECT (count(*) as ?count) WHERE ?s ?p ?o .

Q2) The second query is to fetch the entire dataset in SPO format. It fetches data in the N-Triples format.

Query: SELECT * { ?s ?p ?o } .

Q3) The third query is to fetch resources that belong to books with the subject "English language Composition and exercises."

Query: SELECT ?s WHERE ?x rdf:type Bibo:Book . ?x DC:Subject <http://bnb.data.bl.uk/id/concept/lcsh/English_language_Composition_and_exercises> .

Q4) The fourth query is to fetch resources that belong to books with the subject "English language Composition and exercises" and creator "Palmer Frederick."

Query: SELECT ?s WHERE ?x rdf:type Bibo:Book . ?x DC:Subject <http://bnb.data.bl.uk/id/concept/lcsh/English_language_Composition_and_exercises> . ?x DC:Creator <http://bnb.data.bl.uk/id/person/Palmer_Frederick>.

Q5) The fifth query is to fetch objects having predicate DCTerms:isPartOf.

Query: SELECT ?name WHERE ?s DCTerms:isPartOf ?name .

Figure 10 shows the query response time for the above queries on different worker nodes for two different datasets. The queries are executed in the distributed environment. It shows that increasing the number of worker nodes decreases the query response time.

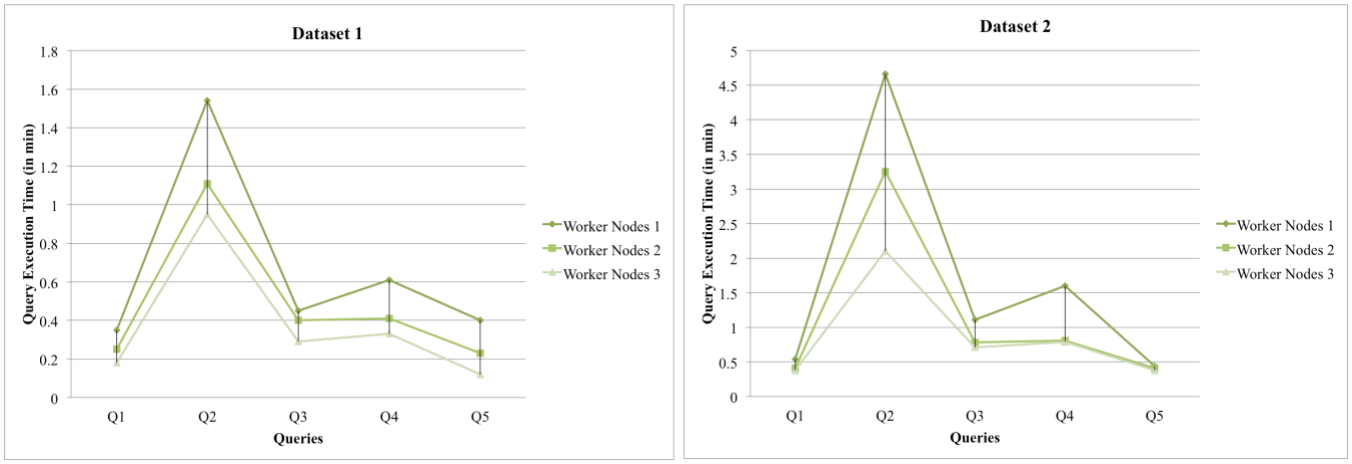

Figure 10. Query response time with different numbers of worker nodes.

Query Comparison

For comparing query response time, the proposed approach is tested with the first dataset as mentioned above. Though at this stage the proposed approach requires further research to be compared with other distributed triple storage systems. Also, it requires more worker nodes and larger datasets compatible for parallel processing in the distributed environment. With a smaller setup, it will be hard to analyze the performance of the individual approaches, as they may produce similar results. We compare the proposed approach with the standard Jena TDB solution in a single-node environment. The following SPARQL queries are tested against dataset 1.

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

prefix DC: <http://purl.org/dc/terms/>

prefix rdau: <http://rdaregistry.info/Elements/u/>

prefix foaf: <http://xmlns.com/foaf/0.1/>

Q1. SELECT (count(*) AS ?count) { ?s ?p ?o }

Q2. SELECT * { ?s ?p ?o }

Q3. SELECT ?x WHERE { ?x rdf:type DC:BibliographicResource. }

Q4. SELECT ?x WHERE { ?x rdf:type <http://schema.org/Book>. ?x rdau:P60339
'Time Out Lisboa'. }

Q5. SELECT ?s WHERE  {?s DC:isPartOf
<http://data.theeuropeanlibrary.org/Collection/a0511>. ?s foaf:page
'http://www.theeuropeanlibrary.org/tel4/record/3000115318515'. }
```


https://doi.org/10.6017/ital.v37i3.10177

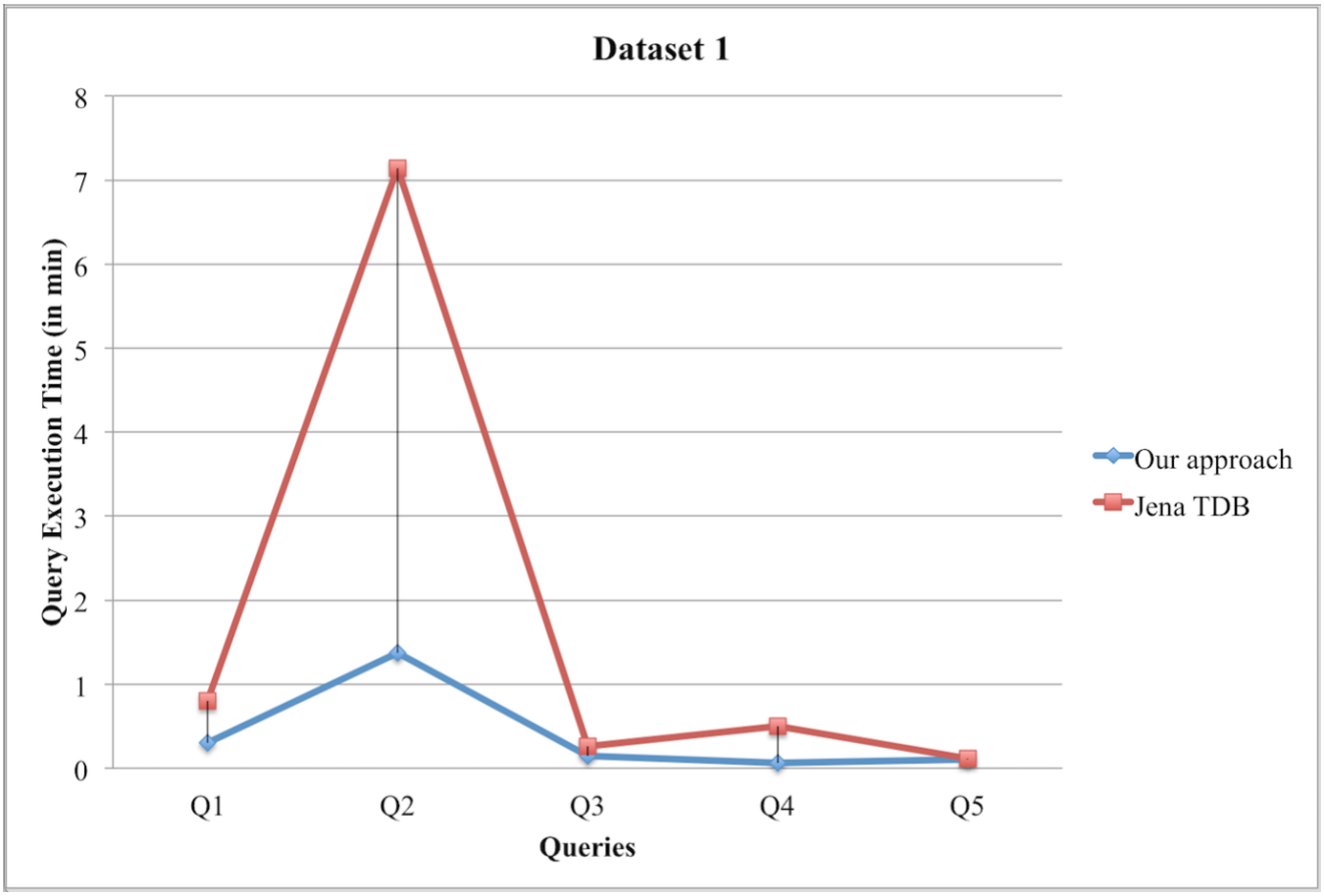

Figure 11. Query comparison.

We are interested in measuring the query response time with the above queries. First, we test with Jena TDB. We then test the proposed approach on a single-node environment. We execute the above set of queries multiple times to record the average performance. As mentioned above, no indexing is used in the storage. RDF triples are stored as they appeared in the N-Triples file. Queries are executed without indexing and are still getting better performance than Jena TDB, as shown in figure 11.

Discussion

In this article, we claim that Apache Spark and column-oriented databases can resolve library big data issues. Especially when dealing with RDF data, Spark can perform far better than other approaches because of its in-memory processing ability. Concerning RDF data storage, the column-oriented database is suitable for storing the large volume of data because of its scalability, fast data loading, and highly efficient data compression and partitioning. A column-oriented database system requires less disk, reducing the storage area. As a proof, we have shown the data storage comparison and the performance of the columnar-storage for RDF data using Parquet formats in HDFS. As shown in the results, Apache Parquet takes much less disk space as compared to other storage systems. Also, the data-storing time is relatively very small as compared to others. We observed that the result of query 2 is the entire dataset stored in Parquet format. The size of this resultant dataset is 22.52 GB, which is the same as the original size. The same dataset when stored with Parquet format is reduced to 2.89 GB. This shows that Parquet is a very optimized

storage system that can reduce the storage cost. We have shown the query response time for five different SPARQL queries on distributed nodes for two different datasets. We believe with better schema for storing RDF triples the proposed approach can be improved, and with the used technologies a fast and reliable triple store can be designed.

CONCLUSION AND FUTURE WORK

Librarians all over the globe should give priority to integrating library data with the web to enable cross-domain sharing of library data. To do this, they must pay attention to current trends in big data technologies. Because the data-generation rate is increasing in every domain, traditional data processing and storage systems are becoming ineffective because of the scale and complexity of the data. In this article, we present a distributed solution for processing and storing a large volume of library linked data. From the experiment, we observe that the processing of large volume of the data takes significantly less time using the proposed approach. Also, the storage area is reduced significantly as compared to other storage systems. In the future we plan to optimize the current approach using advanced technologies such as GraphX, machine learning tools, and other big-data technologies for even faster data processing, searching, and analyzing.

REFERENCES

[1] Eric Miller et al., "Bibliographic Framework as a Web of Data: Linked Data Model and Supporting Services," Library of Congress, November 11, 2012, https://www.loc.gov/bibframe/pdf/marcld-report-11-21-2012.pdf.

[2] Brighid M. Gonzales, "Linking Libraries to the Web: Linked Data and the Future of the Bibliographic Record," *Information Technology and Libraries* 33 no. 4 (2014): 10, https://doi.org/10.6017/ital.v33i4.5631; Myung-Ja K. Han et al., "Exposing Library Holdings Metadata in RDF Using Schema.org Semantics," in International Conference on Dublin Core and Metadata Applications DC-2015, São Paulo, Brazil, September 1–4, 2015, pp. 41–49, http://dcevents.dublincore.org/IntConf/dc-2015/paper/view/328/363.

[3] Franck Michel et al., "Translation of Relational and Non-relational Databases into RDF with xR2RML," in Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 2015, pp. 443–54, https://doi.org/10.5220/0005448304430454; Varish Mulwad, Tim Finin, and Anupam Joshi, "Automatically Generating Government Linked Data from Tables," *Working Notes of AAAI Fall Symposium on Open Government Knowledge: AI Opportunities and Challenges* 4, no. 3 (2011), https://ebiquity.umbc.edu/_file_directory_/papers/582.pdf; Matthew Rowe, "Data.dcs: Converting Legacy Data into Linked Data," *LDOW* 628 (2010), http://ceur-ws.org/Vol-628/ldow2010_paper01.pdf.

[4] Virginia Schilling, "Transforming Library Metadata into Linked Library Data," Association for Library Collections and Technical Services, September 25, 2012, http://www.ala.org/alcts/resources/org/cat/research/linked-data.

[5] Getaneh Alemu et al., "Linked Data for Libraries: Benefits of a Conceptual Shift from Library-Specific Record Structures to RDF-Based Data Models," *New Library World* 113, no. 11/12 (2012): 549–70 (2012), https://doi.org/10.1108/03074801211282920.

[6] Lisa Goddard and Gillian Byrne, "The Strongest Link: Libraries and Linked Data," *D-Lib Magazine,* 16, no. 11/12 (2010), https://doi.org/10.1045/november2010-byrne.

[7] T. Nasser and R. S. Tariq, "Big Data Challenges," *Journal of Computer Engineering & Information Technology* 4, no. 3 (2015), https://doi.org/10.4172/2324-9307.1000133.

[8] Alexandru Adrian Tole, "Big Data Challenges," *Database Systems Journal* 4, no. 3 (2013): 31–40, http://dbjournal.ro/archive/13/13_4.pdf.

[9] Carol Jean Godby and Karen Smith-Yoshimura, "From Records to Things: Managing the Transition from Legacy Library Metadata to Linked Data," *Bulletin of the Association for Information Science and Technology* 43, no. 2 (2017): 18–23, https://doi.org/10.1002/bul2.2017.1720430209.

[10] Corine Deliot, "Publishing the British National Bibliography as Linked Open Data," *Catalogue & Index*, issue 174 (2014): 13–18, http://www.bl.uk/bibliographic/pdfs/publishing_bnb_as_lod.pdf; Gustavo Candela et al., "Migration of a Library Catalogue into RDA Linked Open Data," *Semantic Web* 9, no. 4 (2017): 481–91, https://doi.org/10.3233/sw-170274; Martin Malmsten, "Exposing Library Data as Linked Data," IFLA satellite preconference sponsored by the Information Technology Section: Emerging Trends in 2009, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.181.860&rep=rep1&type=pdf; Keri Thompson and Joel Richard, "Moving Our Data to the Semantic Web: Leveraging a Content Management System to Create the Linked Open Library," *Journal of Library Metadata* 13, no. 2–3 (2013): 290–309, https://doi.org/10.1080/19386389.2013.828551; Jason A. Clark and Scott W. H. Young, "Linked Data is People: Building a Knowledge Graph to Reshape the Library Staff Directory," *Code4lib Journal* 36 (2017), http://journal.code4lib.org/articles/12320; Martin Malmsten, "Making a Library Catalogue Part of the Semantic Web," Humbolt University of Berlin, 2008, https://doi.org/10.18452/1260.

[11] R. Hastings, "Linked Data in Libraries: Status and Future Direction," *Computers in Libraries* 35, no. 9 (2015): 12–28, http://www.infotoday.com/cilmag/nov15/Hastings--Linked-Data-in-Libraries.shtml.

[12] Mirjam Keßler, "Linked Open Data of the German National Library," In ECO4r Workshop LOD of DNB, 2010; Antoine Isaac, Robina Clayphan, and Bernhard Haslhofer, "Europeana: Moving to Linked Open Data," *Information Standards Quarterly* 24, no. 2/3 (2012)**<<QY: page range?>>**; Carol Jean Godby and Ray Denenberg, "Common Ground: Exploring Compatibilities between the Linked Data Models of the Library of Congress and OCLC," OCLC Online Computer Library Center, 2015, https://files.eric.ed.gov/fulltext/ED564824.pdf.

[13] Chunning Wang et al., "Exposing Library Data with Big Data Technology: A Review," 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), pp. 1-6, https://doi.org/10.1109/icis.2016.7550937.

[14] B. McBride, "Jena: a Semantic Web Toolkit," *IEEE Internet Computing* 6, no. 6 (2002): 55–59, https://doi.org/10.1109/mic.2002.1067737; Jeen Broekstra, Arjohn Kampman, and Frank Van

Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," *International Semantic Web Conference*, ed. J. Davies, D. Fensel, and F. van Harmelen (Berlin and Heidelberg: Springer, 2002), https://doi.org/10.1002/0470858060.ch5.

[15] "Apache Jena—TDB," Apache Jena, accessed August 22, 2018, https://jena.apache.org/documentation/tdb/.

[16] "Sesame (framework)," Everipedia, July 15, 2016, https://everipedia.org/wiki/Sesame_(framework)/.

[17] Asim Ullah et al., "BookOnt: A Comprehensive Book Structural Ontology for Book Search and Retrieval," 2016 International Conference on Frontiers of Information Technology (FIT), 211–16, https://doi.org/10.1109/fit.2016.046.

[18] Tom Heath and Christian Bizer, "Linked Data: Evolving the Web into a Global Data Space," *Synthesis Lectures on the Semantic Web: Theory and Technology* 1, no. 1 (2011): 1–136, https://doi.org/10.2200/s00334ed1v01y201102wbe001.

[19] Christian Bizer et al., "Linked Data on the Web (LDOW2008)," Proceeding of the 17th International Conference on World Wide Web—WWW 08, 2008, pp. 1265–66 (2008), https://doi.org/10.1145/1367497.1367760.

[20] Eric Prud and Andy Seaborne, "SPARQL Query Language for RDF," W3C Recommendation, January 15, 2008, https://www.w3.org/TR/rdf-sparql-query/.

[21] Devin Gaffney, "How to Use SPARQL," Datagov Wiki RSS, last modified April 7, 2010, https://data-gov.tw.rpi.edu/wiki/How_to_use_SPARQL.

[22] Tom White, *Hadoop: The Definitive Guide* (Sebastopol, CA: O'Reilly Media,, 2012), https://www.isical.ac.in/~acmsc/WBDA2015/slides/hg/Oreilly.Hadoop.The.Definitive.Guide.3rd.Edition.Jan.2012.pdf.

[23] Dhruba Borthakur, "The Hadoop Distributed File System: Architecture and Design," Hadoop Project Website, 2007, http://svn.apache.org/repos/asf/hadoop/common/tags/release-0.16.3/docs/hdfs_design.pdf; Seema Maitrey and C. K. Jha, "MapReduce: Simplified Data Analysis of Big Data," *Procedia Computer Science* 57 (2015), 563–71 (2015), https://doi.org/10.1016/j.procs.2015.07.392.

[24] Michael Armbrust et al., "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York: ACM, 2015), 1383–94, https://doi.org/10.1145/2723372.2742797.

[25] Abdul Ghaffar Shoro and Tariq Rahim Soomro, "Big Data Analysis: Apache Spark Perspective," *Global Journal of Computer Science and Technology* 15, no. 1 (2015), https://globaljournals.org/GJCST_Volume15/2-Big-Data-Analysis.pdf.

[26] Salman Salloum et al., "Big Data Analytics on Apache Spark," *International Journal of Data Science and Analytics* 1, no. 3–4 (2016): 145–64, https://doi.org/10.1007/s41060-016-0027-9.

[27] Daniel J. Abadi, Samuel R. Madden, and Nabil Hachem, "Column-Stores vs. Row-Stores: How Different are They Really?," in Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (New York: ACM, 2008), 967–80, https://doi.org/10.1145/1376616.1376712.

[28] Deepak Vohra, "Apache Parquet," in *Practical Hadoop Ecosystem* (Berkeley, CA: Apress, 2016), 325–35, https://doi.org/10.1007/978-1-4842-2199-0_8.

[29] "Google/Snappy," GitHub, January 04, 2018, https://github.com/google/snappy.

[30] Jean-loup Gailly and Mark Adler, "Zlib Compression Library," 2004, https://www.repository.cam.ac.uk/bitstream/handle/1810/3486/rfc1951.txt?sequence=4.

[31] Sergey Melnik et al., "Dremel: Interactive Analysis of Web-Scale Datasets," *Proceedings of the VLDB Endowment* 3, no. 1–2 (2010): 330–39, https://doi.org/10.14778/1920841.1920886.

[32] Marcel Kornacker et al., "Impala: A Modern, Open-Source SQL Engine for Hadoop," in Proceedings of the 7th Biennial Conference on Innovative Data Systems Research, Asilomar, California, January 4–7, 2015, http://www.inf.ufpr.br/eduardo/ensino/ci763/papers/CIDR15_Paper28.pdf.