

---

# Query Processing Optimization

---

*A Project report submitted in fulfilment of the requirements  
for the degree of Bachelor of Technology*

*by*

RITESH KUMAR (2015UCP1727),  
PRINCE RAJ (2015UCP1447),  
PRADEEP KUMAR MEENA (2015UCP1476)

*Under Guidance of*

**Dr. Pilli Emmanuel Shubhakar**

Associate Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY JAIPUR

May 2019

# Declaration

We,

RITESH KUMAR (2015UCP1727),

PRINCE RAJ (2015UCP1447),

PRADEEP KUMAR MEENA (2015UCP1476),

Declare that this thesis titled, “Query Processing Optimization” and the work presented in it are our own. I confirm that:

- This project work was done wholly or mainly while in candidature for a B.Tech. degree in the department of Computer Science and Engineering at Malaviya National Institute of Technology Jaipur (MNIT).
- Where any part of this thesis has previously been submitted for a degree or any other qualification at MNIT or any other institution, this has been clearly stated. Where we have consulted the published work of others, this is always clearly attributed. Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this Dissertation is entirely our own work.
- We have acknowledged all main sources of help.

Signed:

---

Date:

---

Dr. Pilli Emmanuel Shubhakar

Associate Professor

Department of Computer Science and Engineering

Malaviya National Institute of Technology Jaipur

*May 2019*

# *Abstract*

---

Name of the students:

**RITESH KUMAR (2015UCP1727),**

**PRINCE RAJ (2015UCP1447),**

**PRADEEP KUMAR MEENA (2015UCP1476)**

Degree for which submitted: **Bachelor of Technology**

Department: **Computer Science and Engineering**

Thesis title: **Query Processing Optimization**

Thesis supervisor: **Dr. Pilli Emmanuel Shubhakar**

Month and year of thesis submission: **May 2019**

---

The vision of the Semantic Web has realized new difficulties so we utilized an information-driven methodology for the capacity of RDF in social databases. The instinct behind our methodology is that each RDF dataset requires a custom fitted table construction that accomplishes efficient question handling by lessening the requirement of participants in the query plan and keeping invalid stockpiling underneath a given limit.

We utilized a two-stage calculation including clustering and partitioning. The grouping stage expects to decrease the time by putting away the information so that it lessens the requirement for participants in a question. This methodology does not accept a specific inquiry remaining task at hand, important for RDF learning bases with countless ad-hoc inquiries. Broad trial proof utilizing three openly accessible genuine world RDF informational collections demonstrates that our blueprint creation system gives better inquiry preparing execution than best in class stockpiling approaches. Further, our methodology is effectively actualized and supplements existing RDF-specific databases

## *Acknowledgements*

It is a matter of great pleasure and privilege for us to present our B.Tech Project report on “Query Processing Optimization”. We would like to express our gratitude towards our supervisor **Dr. Pilli Emmanuel Shubhakar**, Associate Professor, Department of Computer Science and Engineering, Malaviya National Institute of Technology, Jaipur for his time, invaluable guidance and cooperation throughout the period. He provided us with continuous encouragement and support. He has been the principle guiding force behind this.

We are grateful to respected **Dr. Girdhari Singh**, Head of Department of Computer Science and Engineering for permitting us to utilize all the necessary facilities of the institution.

We want to express special gratitude to our mentor, Mrs. Tanvi Chawla. She gave direction to our work, her time and guidance when we faced setbacks and help in times of crisis. We express our deep gratitude to Mr. Suraj and the staff of EICT Lab for providing us a healthy working environment.

We also express our heartfelt gratitude towards all the teachers of Department of Computer Science and Engineering and technical staff for all their guidance and help throughout our stay at MNIT Jaipur.

**Ritesh Kumar 2015UCP1727**

**Prince Raj 2015UCP1447**

**Pradeep Meena 2015UCP1476**

# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Objectives . . . . .	2
<b>2 Related Work</b>	<b>3</b>
2.1 RDF Data . . . . .	3
2.2 Storage Schema . . . . .	4
<b>3 Proposed Work</b>	<b>7</b>
3.1 Overview . . . . .	7
3.2 Problem Description . . . . .	7
3.3 Algorithm . . . . .	8
3.3.1 Clustering (Phase 1) . . . . .	10
3.3.2 Partitioning (Phase 2) . . . . .	14
<b>4 Tools and Technology Used</b>	<b>18</b>
4.1 Apache Hadoop . . . . .	18
4.1.1 MapReduce . . . . .	20
4.1.2 HDFS . . . . .	21
4.2 Apache Spark . . . . .	25
4.3 Java . . . . .	26
4.4 Apache Hive . . . . .	27
4.5 Eclipse IDE . . . . .	28

---

4.6	Apache Parquet . . . . .	29
4.7	Apache Avro . . . . .	30
<b>5</b>	<b>Results and Snapshots</b>	<b>31</b>
<b>6</b>	<b>Conclusion and Future Work</b>	<b>35</b>
6.1	Conclusion . . . . .	35
6.2	Future Work . . . . .	35

# List of Figures

2.1	RDF Data sample in Graph Form . . . . .	3
2.2	RDF Storage Schema Example . . . . .	4
3.1	RDF Data Clustering Example . . . . .	12
3.2	Clustering Algorithm . . . . .	12
3.3	Property Usage List . . . . .	13
3.4	Subject Property Bucket . . . . .	13
3.5	Initial Final Table list . . . . .	14
3.6	Partitioning Algorithm . . . . .	16
3.7	Final Output Custer . . . . .	17
4.1	Hadoop Ecosystem . . . . .	18
4.2	Hadoop Layers . . . . .	19
4.3	Hadoop MapReduce Stages . . . . .	21
4.4	HDFS Architecture . . . . .	22
5.1	Binary Tables . . . . .	32
5.2	Binary Table Format (ub_advisor) . . . . .	33
5.3	Nary property Tables . . . . .	33
5.4	N-ary Table in Parquet Format on HDFS . . . . .	33
5.5	Nary table as read from parquet file . . . . .	34

# List of Tables

3.1	Cluster of similar properties . . . . .	8
5.1	Size and loading times using Lubm Dataset . . . . .	31
5.2	Schema Breakdown for University100.nt Dataset . . . . .	31
5.3	Property Distribution Table . . . . .	32
5.4	Query Processing Time for Prost and Data-Centric . . . . .	32



# Chapter 1

## Introduction

Over the recent years, efforts have been put by World Wide Web Consortium to create the Semantic Web. The Semantic Web is a long-term project initiated by the World Wide Web Consortium to realise the idea of having data on the network defined and linked to the extent that it can be used by machines for self-regulation, presenting combination and reuse of data beyond various applications. It is a Web-technology that stays on top of the present web by comprising machine-readable data in files without changing the existing Web structure. It is an effort to improve the modern web so that computers can process the data existing on the World Wide Web (WWW), learn and fix it, and help humans to discover the required knowledge. It is intended to form a great distributed knowledge base system to share data instead of documents. In other words, we could say that the semantic web is a general structure that allows data to be distributed and reused over applications, enterprises and community barriers.

The Semantic Web can be defined as interconnected data that are associated such that they can easily be managed and processed by computers instead of a human. The semantic web could be treated as an enhanced version of the present “World Wide Web”, and it serves a useful way of data representation in the form of a globally connected database. It also deals with the regeneration of the presently available web of unstructured documents to a Web of information/data. The basics of the Semantic Web is based on the Resource Description Framework (RDF) data model. RDF provides a relatively much easier syntax. Many other areas of interest and use-cases exist for RDF, such as education, mobile search environments, social networking, and biology and life science, making it an emerging and challenging study field.

It is a major hurdle at the core of the Semantic Web to handle RDF data efficiently so that it can be easily scalable. Several great RDF storage solutions use relational databases to achieve this scalability and efficiency. Our approach uses the Data-Centric approach to store the RDF data so that query processing time can be reduced and optimized.

## 1.1 Motivation

The great use of the Semantic Web is triggering the growth of the existing Web. It enables users to explore, learn, share and connect information with less trouble. Humans use the Web to accomplish various jobs, but machines are not capable of performing any of these tasks without human intervention because web pages are only understandable to humans, not machines. The Semantic Web is the unfolding vision of the Web in which data could be quickly understood by machines, enabling them to achieve different complicated tasks on the Web. However, to manage these huge and scalable data is quite a difficult task to deal with. So we have to use the proper technique to store RDF data such that it becomes easy and efficient to retrieve data to reduce the query response time. It is also necessary to optimize the query processing time of the frequently used queries. There are many approaches available to efficiently store RDF data for improving query performance but hardly any methods are implemented using Big Data Technologies (they don't use the Hadoop, spark, map-reduce, hive, etc.). Also, the methods available have their own advantages and disadvantages. Some methods trade-off performance with null storage, join operation and other overheads. So to remove disadvantages of these methods and to implement those on the top of Big Data is quite a challenging task.

## 1.2 Objectives

The primary goal of this project is to decrease the query processing time by causing changes to the data storage structure on which the query would be executed. We accomplish this goal by taking into some trade-off of expensive query processing procedures in relational databases. Reduce on average, the requirement to join tables in a query by grouping the related RDF Data in a table. Minimise the amount of redundant data processing by reducing extra data storage (e.g., null storage). Include a distributed computing paradigm to balance data processing over several computing nodes to reduce the processing time.

## Chapter 2

# Related Work

### 2.1 RDF Data

The RDF data type is the essence of the Semantic Web. It depicts an efficient method of data representation in the form of a globally connected database. The syntax of RDF data is very naive, where all data object is divided into a triple which looks like ; subject, property, object . The subject i.e. the first part of the triple denotes an entity instance, linked by a Uniform Resource Identifier (URI). The property represents an attribute of the entity, and the object is expressed as the value of the property.

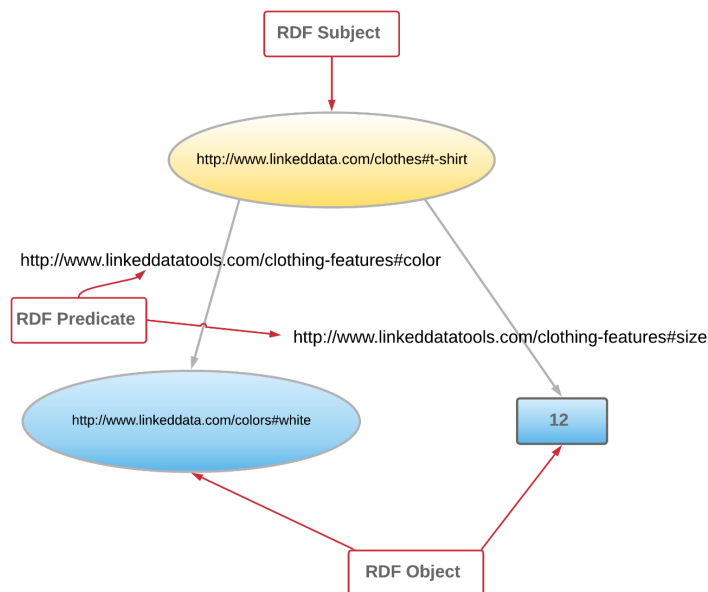


FIGURE 2.1: RDF Data sample in Graph Form

As a simple example, the figure 2.1 RDF triple expresses that the t-shirt's colour is white and its size is 12. Though the universality of the RDF data model has yet to be accepted, many areas like application and use-cases exist for RDF, such as learning, mobile search, social networking, biology, and life science presenting it as an emerging and challenging study area.

## 2.2 Storage Schema

It is a major hurdle at the core of the Semantic Web to handle RDF data efficiently so that it can be easily scalable. Several great RDF storage methods use relational databases to achieve this scalability and efficiency. Some of the storage techniques are discussed below.

<Person1, Name, Mike>  
 <Person1, Website, ~mike>  
 <Person2, Name, Mary>  
 <Person3, Name, Joe>  
 <Person4, Name, Kate>  
 <City1, Population, 200K>  
 <City2, Population, 300K>

### Query

"Find all people that have both a  
 name and website"

(a) RDF Triples

NameWebsite			Population	
Subj	Name	Website	Subj	Pop.
Person1	Mike	~mike	City1	200K
Person2	Mary	NULL	City2	300K
Person3	Joe	NULL		
Person4	Kate	NULL		

```
SELECT T.Name, T.Website
FROM NameWebsite T
Where T.Website IS NOT NULL;
```

(c) N-ary Table

TS		
Subj	Prop	Obj
Person1	Name	Mike
Person1	Website	~mike
Person2	Name	Mary
Person3	Name	Joe
Person4	Name	Kate
City1	Pop.	200K
City2	Pop.	300K

```
SELECT T1.Obj, T2.Obj
FROM TS T1, TS T2
WHERE T1.Prop=Name AND
T2.Prop=Website AND
T1.Subj=T2.Subj;
```

(b) Triple Store

Name		Website	
Subj	Obj	Subj	Obj
Person1	Mike	Person1	~mike
Person2	Mary		
Person3	Joe		
Person4	Kate		

Population	
Subj	Obj
City1	200K
City2	300K

```
SELECT T1.Obj, T2.Obj
FROM Name T1, Website T2
WHERE T1.Subj=T2.Subj;
```

(d) Binary Tables

FIGURE 2.2: RDF Storage Schema Example

To illustrate, Figure 2.2 represents an example in the form of an RDF triple regarding four persons and two cities and a query that requests the name of people who have both a name and a website in the RDF triple. Figures 2.2(b) – 2.2(d) indicates three feasible ways to store those RDF triples given in the example into relational databases, with altered

RDF queries given simultaneously in SQL. Abundant of systems use a triple-store schema where all of the RDF triples are straight away saved in a relational table consisting of three columns (Figure 2.2(b)). This method proves inefficient since each query processing takes a significant amount of time due to multiple self-joins, where a single join is itself an expensive operation. The second method illustrates the use of n-ary property tables where multiple properties of the RDF triples are kept in a single table, which removes the necessity to perform several join operations. But since only a single person has a website, so the n-ary property table needs to include a huge number of null values which causes an overhead while executing queries. The last storage structure stores each property of the RDF dataset in a different table known as binary tables. The binary table has only two fixed columns - one for the subject and other for the object part of the stored RDF triple. Though this method reduces the amount of null storage, it increases the number of join operations between different binary tables.

These approaches to store RDF data can be classified into three main categories.

1. The triple-store (Figure 2.2(b)) Relational tables that make use of a triple-store as their primary storage scheme by directly storing the triple in three different columns, namely for subject, property and object.
2. The property table (Figure 2.2(c)) The property table was suggested because of the problems present in the triple-store due to the self joins. As the basic storage structure, this approach requires the use of property tables and the Jena Semantic Web Toolkit. Property tables are used by Oracle as secondary structures, and it is known as materialised join views
3. The decomposed storage model (Figure 2.2(d)) It has been proved to scale well on column-oriented databases, with combined outcomes for row-stores. It is different from the earlier used methods as it
  - Provides a customised schema for all RDF data set triples, by including both the n-ary tables(i.e. the property tables) and the binary tables(i.e. decomposed storage) in a proper ratio and
  - Provides an efficient way to save properties together in tables analysing the composition of the data.

Earlier approaches for making property tables have required the application of generic pre-computed joins, or creation by a DBA with the learning of query usage statistics. It

---

is necessary to remark that making of our relational schema is system independent, and it could work for any of the RDF based datasets.

Another activity in RDF storage has dealt with saving previously computed routes in a relational database, which is applied to solve the queries related to graphs across the data (i.e., path, shortest route). Different approaches have been proposed for graph database approaches to RDF, including extensions to RDF query languages to run graph queries.

## Chapter 3

# Proposed Work

### 3.1 Overview

Generally, there exist two modules to handle RDF data and RDF queries apart from the database engine. They are:

1. RDF import module
2. RDF query module

The data-centric schema creation exists inside the RDF import module to handle RDF data. The process which creates schema or table structure takes an RDF data set as input. A schema is produced to store the imported RDF data in the underlying DBMS as an output of the technique.

### 3.2 Problem Description

Given a dataset which consists of RDF triples, compute a relational table schema having its properties as table columns and objects as table tuples that achieves the following standards.

1. Cluster the related properties together in a single table so that queries can access the same table without causing the need to join multiple tables for its execution.

NameWebsite			Population	
Subj	Name	Website	Subj	Pop
Person1	Mike	~mike	City1	200K
Person2	Mary	NULL	City2	300K
Person3	Joe	NULL		
Person4	Kate	NULL		

TABLE 3.1: Cluster of similar properties

*Query* : **SELECT T.Name, T.Website FROM NameWebsite T WHERE T.Website IS NOT NULL;**

2. Reducing the amount of redundant data values (i.e. null values) to be processed by removing the extra data.

Reduce the high query execution cost in relational databases by reducing the number of join operations and another table accesses. To accomplish the first criteria, we use the data-centric schema creation process by storing the similar type of RDF data together in n-ary property tables. But, as we observed in the above example given in Figure 2.2, n-ary property tables consume more amount of storage that affects query execution efficiency. Lastly, to achieve our next criteria, our schema creation algorithm tries to reduce the null storage in each table below a given threshold.

### 3.3 Algorithm

Our algorithm which creates the relational schema takes as input the RDF data set, along with two other parameters:

1. Support threshold - The similarity among the properties present in the RDF dataset is measured by using this attribute. If the properties in a cluster satisfy this threshold value then they are decided to exist in the same n-ary table otherwise not.
2. Null threshold – It is the amount of redundant null data which can be allowed to exist in a single table by combining similar properties.

The data structures that are used in our algorithm are computed in  $O(n)$  time complexity by reading the triples in the RDF dataset once (where  $n$  is the total number of triples



present in the RDF dataset we are using to create relational tables). The data structures used in our algorithm are

1. Property Usage List – It is a list type of data structure. It contains all the distinct properties and the frequencies of all those distinct properties or predicates present in the RDF dataset.

For example, RDF Triples Dataset sample :

$\langle \text{Person1}, \text{Name}, \text{Mike} \rangle$   
 $\langle \text{Person1}, \text{Website}, \text{mike} \rangle$   
 $\langle \text{Person2}, \text{Name}, \text{Mary} \rangle$   
 $\langle \text{Person3}, \text{Name}, \text{Joe} \rangle$   
 $\langle \text{Person4}, \text{Name}, \text{Kate} \rangle$   
 $\langle \text{City1}, \text{Population}, \text{200K} \rangle$   
 $\langle \text{City2}, \text{Population}, \text{300K} \rangle$

On computing the Property Usage List for the RDF triples in the given example, the predicate “Website” will have a count of 1, because as we can see in the above dataset that it is defined only for the “Person1” subject. Similarly the property “Name” will have the frequency 4 and “Population” will have a frequency of 2.

2. Subject-Property Basket – This data structure is also a list that on one hand contains all the distinct subjects in the dataset and each of them contains a list of all the properties being referred to by the subject. A single value in the Subject-Property Basket structure is of the form

$$subj_i \rightarrow \{prop_1, \dots, prop_n\},$$

Where ‘ $subj_i$ ’ is the URI of any of the subject of the triple and the property bucket contains the list of predicates present in the triple with that subject. E.g., for the above data in Figure 3.2, if we create subject property basket, it would look like this.

$\text{Person1} \rightarrow \{\text{Name}, \text{Website}\}$   
 $\text{Person2} \rightarrow \{\text{Name}\}$   
 $\text{Person3} \rightarrow \{\text{Name}\}$   
 $\text{Person4} \rightarrow \{\text{Name}\}$

City1  $\rightarrow$  {Population}  
 City2  $\rightarrow$  {Population}  
 City1  $\rightarrow$  {Population}  
 City2  $\rightarrow$  {Population}

The algorithm for schema creation involves two modules :

1. Clustering
2. Partitioning

### 3.3.1 Clustering (Phase 1)

This phase focuses on finding the cluster of related predicates based on the support threshold value of each cluster. Clustering uses previous work to get maximum advantage from association rule mining to find similar attributes in the data. The idea behind this stage is to store the related predicates in a single N-ary relational table. The main reason for the n-ary tables is that the similar or predicates with similar likelihood are likely to be utilised in a query together. Therefore the predicates which are similar are stored in the same cluster and after clustering, they are transferred to a single table. Because of this, there is a reduction in the number of join operations each time a query is run. This phase creates an initial collection of relational tables. The predicates in the initial tables thus constructed will not be seen in any other tables which would be further generated and are stored in binary tables. It will consist of those predicates as well which would not need partitioning i.e., Phase II.

The clustering phase involves the following two steps :

**Step 1** - A set of similar predicates are computed by focusing on the frequent predicates used together. The terms similar predicates and clusters are used interchangeably. Collections of predicates that are found frequently in the Subject-Property Basket data structure are found by the clustering phase. The cluster support is used to measure how frequently a cluster occurs. Clusters with high support mean that all of those properties occur with many RDF subjects. In simple language, large support indicates that properties of the cluster are closely related to each other since they usually exist collectively in the data. This support threshold parameter used by the algorithm sets a metric for high support value, meaning the group of properties having support greater than or equal to

the support threshold is known to be a cluster. If we make the support threshold high enough, then this phase creates a less amount of small clusters whose likelihood will be very high. Moreover, if we provide a less support value, then this phase would create more clusters with large sizes and with very less similarity among the predicates. However, for our needs, we are going to need clusters with large sizes or the clusters in which properties are frequently together used. These are the clusters that can be found frequently in the dataset and contains a large number of properties which leads to increment in the null values stored in the n-ary tables. It can also be noted that if we compute maximum frequent itemset, then it can generate clusters which can have overlapping predicates.

**Step2** - An initial set of nal tables are computed. They contain -

- The predicates which are not present in any of the computed clusters and so they need to be kept in binary tables
- The clusters that satisfy the null threshold value and also do not have the predicates which would overlap with the predicates of other clusters do not need going through phase II.

Clusters that are added to the initial nal table list are removed from the cluster list. The output of the clustering phase is a list of initial nal tables and a set of clusters, sorted in decreasing order by their support value that will be sent to the partitioning phase for further processing.

For Example:

Consider an example with a support threshold of 15%, a null threshold of 20% for the six subject-property baskets given in data in above example. In this case we have four possible property clusters: Name, Website, Population, and Name,Website. The cluster Name occurs in 4 of the 6 property baskets, giving it support value of 66%, while the cluster Name, Website occurs in 1 of 6 property baskets, giving it support of 16%. In this case, the Name, Website is generated as a cluster, since its support value is greater than the support threshold and has the most possible properties. Note the single property Population is not considered a cluster and would be added to the initial nal table list. There are three example clusters along with their support values, while Figure 3.1(c) contains their null storage values. The output of the clustering phase in this example with support and the null threshold value of 20% would produce an initial nal table list containing {P9} (which is not present in any of the cluster) and {P7, P8} (as it meets null threshold and

also contains non-overlapping properties ).  $\{P1, P2, P3, P4\}$  and  $\{P1, P2, P5, P6\}$  output of 1st phase is sent to the next phase.

<b>Property</b>	<b>Usage</b>	<i>PC: <math>\{P1, P2, P3, P4\}</math> (54% Support)</i>	<b>(b)</b>
<b>P1</b>	<b>1000</b>	<i><math>\{P1, P2, P5, P6\}</math> (45% Support)</i>	
<b>P2</b>	<b>500</b>	<i><math>\{P7, P8\}</math> (30% Support)</i>	<b>(c)</b>
<b>P3</b>	<b>700</b>	<i><math>NullPercentage(\{P1, P2, P3, P4\}) = 21\%</math></i>	
<b>P4</b>	<b>750</b>	<i><math>NullPercentage(\{P1, P2, P5, P6\}) = 32\%</math></i>	<b>(d)</b>
<b>P5</b>	<b>450</b>	<i><math>NullPercentage(\{P7, P8\}) = 4\%</math></i>	
<b>P6</b>	<b>450</b>	<i><math>NullPercentage(\{P1, P3, P4\}) = 13\%</math></i>	<b>(e)</b>
<b>P7</b>	<b>300</b>	<i><math>NullPercentage(\{P2, P5, P6\}) = 5\%</math></i>	
<b>P8</b>	<b>350</b>	<i>Tables = <math>\{P1, P3, P4\}, \{P2, P5, P6\},</math> <math>\{P7, P8\}, \{P9\}</math></i>	<b>(e)</b>
<b>P9</b>	<b>50</b>		

(a)

FIGURE 3.1: RDF Data Clustering Example

---

### Algorithm 1 Clustering

---

```

1: Function Cluster(Baskets  $B$ , Usage  $PU$ ,  $Thresh_{sup}$ ,  $Thresh_{null}$ )
2:  $Clusters \leftarrow GetClusters(B, Thresh_{sup})$ 
3:  $Tables \leftarrow$  properties not in  $PC$  /* Binary tables */
4: for all  $c_1 \in PC$  do
5:    $OK \leftarrow false$ 
6:   if  $Null\%(c_1, PU) \leq Thresh_{null}$  then
7:      $OK \leftarrow true$ 
8:     for all  $c_2 \in PC$  if  $c_1 \cap c_2 \neq \phi$  then  $OK \leftarrow false$ 
9:   end if
10:  if  $OK$  then  $Tables \leftarrow Tables \cup c_1$ ;  $Clusters \leftarrow Clusters - c_1$ 
11: end for
12: return  $Tables, Clusters$ 

```

---

FIGURE 3.2: Clustering Algorithm

Figure 3.2 gives the pseudocode for the clustering phase and takes as parameters the Subject-Property Baskets (B), the Property Usage List (PU), the Support Threshold ( $Thresh_{sup}$ ), and the Null Threshold parameter ( $Thresh_{null}$ ).

In Step 1 of the algorithm clusters are generated which are sorted by their support value, and then stored in list Clusters. Step 2 of the algorithm is performed by rst setting a list Tables to all binary tables returned from Step 1. Tables are then expanded by adding from list Clusters the clusters that do not contain overlapping properties that are below the given Null Threshold. Calculation of the null storage for a cluster ‘c’ is performed using the Property Usage list PU.

$$Null\%(c) = \frac{\sum_{\forall i \in c} (PU.maxcount(c) - PU.count(c_i))}{(|c| + 1) * PU.maxcount(c)}$$

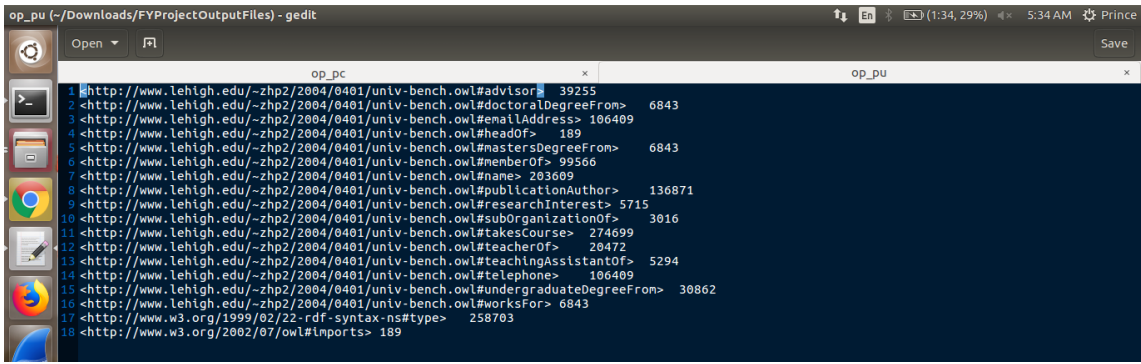


FIGURE 3.3: Property Usage List

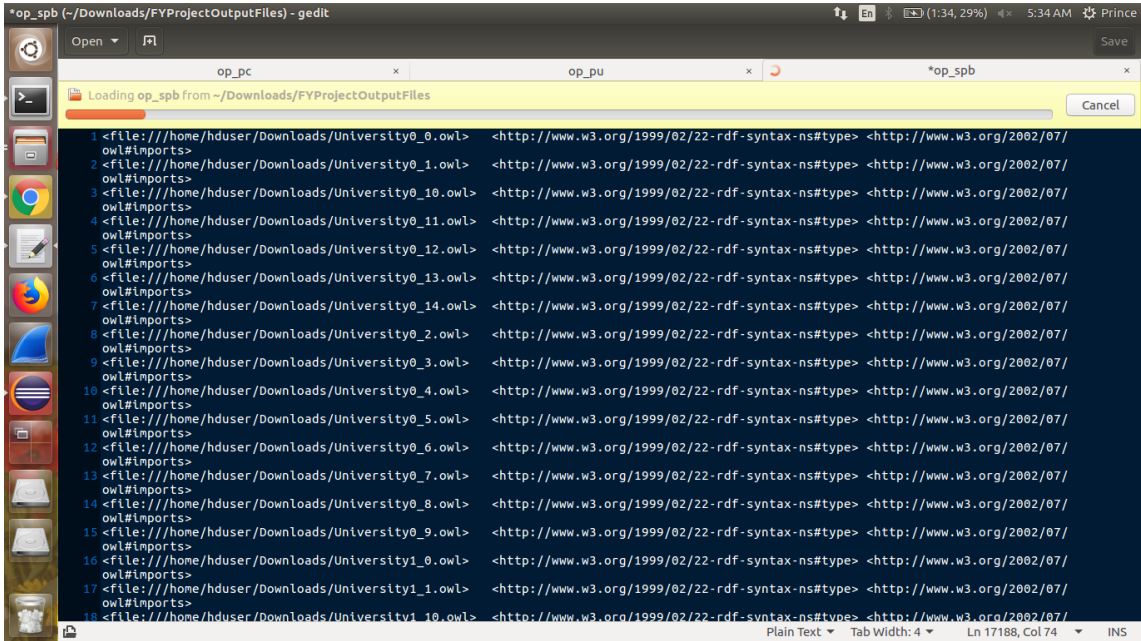


FIGURE 3.4: Subject Property Bucket

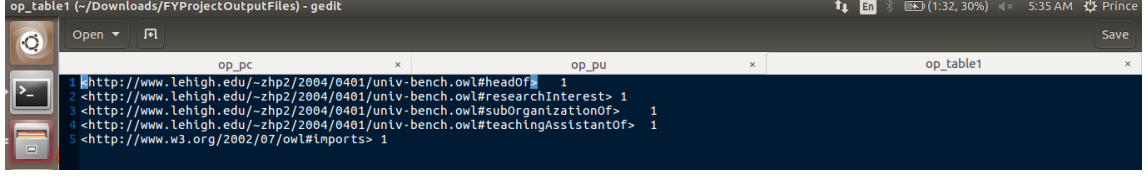


FIGURE 3.5: Initial Final Table list

Let  $\|c\|$  be the number of properties in a cluster, and  $\text{PU.maxcount}(c)$  be the maximum property count for a cluster in PU. As an example, in Figure 3.1(b) if  $c = \{P1, P2, P3, P4\}$ ,  $\|c\| = 4$  and  $\text{PU.maxcount}(c) = 1000$  (corresponding to  $P_1$ ).  $\text{PU.count}(c_i)$  is the usage count for the  $i^{\text{th}}$  property in  $c$ , the null storage percentage for  $c$  is:

### 3.3.2 Partitioning (Phase 2)

In this phase, the clusters from Phase I is given as input. It guarantees that the clusters generated hold a disjoint collection of predicates at the same time maintaining for each cluster the null value below a given threshold. The nal schema is the combination of tables created from Phase I and II.

The purpose of the partitioning phase has two-fold :

1. To partition the set of clusters returned from phase 1 so that there are no overlapping clusters, i.e. the cluster sets are disjoint and each property exists in only a single n-ary table. It ensures that each property exists in a single cluster to reduce the amount of table accesses and joins required in query execution.

For example, let us take RDF data for school stored in two n-ary tables:

Student = {studentId, name, address} and Teacher = {teacherId, name, telephone}.

As in the given example name exist in both the Student and Teacher tables so, if an RDF query asks for all names of students and teachers, then it would involve two table accesses and a union operation.

2. Ensuring the null storage threshold of the partitioned cluster must fall below a particular value. Reducing null storage also helps the tables for effective query processing.

We introduce a greedy algorithm in which we give the highest priority to the high support value clusters and keep them intact and partition the clusters having lower support clusters,

including overlapping predicates. The reason to take the greedy approach is that clusters having high support values include properties that are present collectively more frequently in the RDF data. Support value displays the ratio of RDF subjects that have all of the cluster's predicates. Most of the RDF triples will be stored together in a single table if we keep the high support clusters intact. The greedy algorithm gives priority to the cluster having the highest support value and manages both the two situations based on its null storage calculation.

### Case 1

The property cluster that satisfies the null storage threshold, i.e. the property cluster from first phase which satisfies the null threshold but contains overlapping properties is given as input. In this instance, the cluster is regarded as a table and each clusters having low support values and overlapping properties with other clusters are eliminated from those clusters having lower support. We see that pruning would probably generate overlapping cluster pieces; now these clusters remain no longer with maximum size (i.e., highest frequent itemsets) and carry related predicates. For example, suppose a list of 3 clusters  $c1 = \{A,B,C,D\}$ ,  $c2 = \{A,B,E,F\}$ , and  $c3 = \{C,E\}$  such that  $\text{support\_value}(c1) > \text{support\_value}(c2) > \text{support\_value}(c3)$ . Because our greedy algorithm takes  $c1$  as a nal table, partitioning creates overlapping cluster pieces  $c2 = \{E,F\}$  and  $c3 = \{E\}$ . In this example because  $c3 \subseteq c2$ , these clusters can be joined through the pruning process. Therefore, we join any overlapping parts in the cluster list.

### Case 2

The null threshold criteria is not met by the cluster. So, In this case, it is partitioned so that it satisfies the null threshold. In the partitioning method, the property  $p$  that causes the most null storage is iteratively removed from the cluster till it matches the null threshold. After the iteration, null storage is reduced to a maximum extent. Extra, support for clusters is monotonic: given two clusters  $c1$  and  $c2$ ,  $\text{support}(c1) \geq \text{support}(c2) \implies c1 \subseteq c2$ . Thus, the support value of the cluster after partitioning will still be greater than the given support threshold. After excluding  $p$ , two cases are studied.

- 'P' is present in the cluster having lower support value. Therefore, there is a chance for  $p$  to be kept in an  $n$ -ary table.
- 'P' is not present in the cluster having lower support value. hence  $p$  need to be stored in a binary table.

When a cluster is pruned to satisfy the null threshold criteria it is stored as a table and overlapping properties are removed from all the clusters having lower support values. Example. Looking at our existing e.g. in Figure 3.3, 2 clusters would be transferred to the partitioning stage:  $\{P1, P2, P3, P4\}$  and  $\{P1, P2, P5, P6\}$ . The cluster having the highest support value is  $\{P1, P2, P3, P4\}$  (as given in Figure 3.2(b)) and so it is managed first. Since this cluster does not satisfy the null threshold the cluster is partitioned (Case 2) by eliminating the predicate that creates the largest null storage, P2, corresponding to the predicate with least count in the Property Usage list in Figure 3.2(a). Since P2 is located in the cluster having lower support value  $\{P1, P2, P5, P6\}$  (Case 2-a), it has a possibility of being stored in an n-ary table. Eliminating P2 from  $\{P1, P2, P3, P4\}$  produces the cluster  $\{P1, P3, P4\}$  that whose null storage percentage is below the null threshold of 20% . Therefore it is recognised as a nal table. Since  $\{P1, P3, P4\}$  and  $\{P1, P2, P5, P6\}$  include overlapping predicates, P1 is then excluded from  $\{P1, P2, P5, P6\}$ , creating cluster  $\{P2, P5, P6\}$ . Because null percentage of the cluster  $\{P2, P5, P6\}$  also falls below the null threshold it would be appended to the nal table list in the next loop. Finally, Figure 3.2(e) gives the nal schema with the combination of tables returned from both phases.

---

### Algorithm 2 Partition Clusters

---

```

1: Function Partition(PropClust  $C$ , PropUsage  $PU$ ,  $Thresh_{null}$ )
2:  $Tables \leftarrow \phi$ 
3: for all  $clust_1 \in C$  do
4:    $C \leftarrow (C - clust_1)$ 
5:   if  $Null\%(clust_1, PU) > NullThresh$  then
6:     repeat
7:        $p \leftarrow$  property causing most null storage
8:        $clust_1 \leftarrow (clust_1 - p)$ 
9:       if  $p$  exists in lower-support cluster do continue
10:      else  $Tables \leftarrow Tables \cup p$  /* Binary table */
11:      until  $Null\%(clust_1, PU) \leq NullThresh$ 
12:    end if
13:     $Tables \leftarrow Tables \cup clust_1$ 
14:  forall  $clust_2 \in C$  do  $clust_2 \leftarrow clust_2 - (clust_2 \cap clust_1)$ 
15:  Merge cluster fragments
16: end for
17: return  $Tables$ 

```

---

FIGURE 3.6: Partitioning Algorithm

Figure 3.6 provides the pseudocode for the partitioning phase. It takes the property cluster



list(C) as arguments from Phase I, ordered in decreasing scale by support value, the property usage list (PU), and the null threshold (Threshnull). The algorithm rst initialises the nal table list to null. Subsequently, it parses each property cluster clust1 in list C, from the cluster having the highest support to smallest. Next, clust1 is expelled from the cluster list C. The algorithm then verifies that clust1 matches the null threshold. If that is the case, it holds clust1 a nal table (i.e., Case 1), and each clusters having lower support values with predicates overlapping clust1 are pruned and cluster pieces are joined. If clust1 is not able to satisfy the null threshold, it must be partitioned (i.e., Case 2). The algorithm searches predicate p causing most storage in clust1 (leading to the least property usage count for clust1 in PU) and excludes it. If ‘P’ is present in a cluster having lower support(i.e., Case 2-a), iteration proceeds, else (i.e., Case 2-b) ‘P’ is appended to Tables in a binary table. Partitioning proceeds until clust1 satisfies the null storage threshold. When partitioning completes, the algorithm holds clust1 as a nal table, and prunes all clusters having lower support of properties overlapping with clust1 while joining any cluster pieces. Following screenshot shows the output of partitioning phase.

```

op_mergedTable (~/Downloads/FYProjectOutputFiles) - gedit
1 http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#headOf
2 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#researchInterest>
3 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#subOrganizationOf>
4 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teachingAssistantOf>
5 <http://www.w3.org/2002/07/owl#imports>
6 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#name> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
7 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#emailAddress> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#memberOf> <http://
www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#telephone>
8 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#publicationAuthor>
9 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#advisor>
10 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#undergraduateDegreeFrom>
11 <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#doctoralDegreeFrom> <http://www.lehigh.edu/~zhp2/2004/0401/univ-
bench.owl#mastersDegreeFrom> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teacherOf> <http://www.lehigh.edu/~zhp2/2004/0401/univ-
bench.owl#worksFor>

Bracket match found on line: 1
Plain Text Tab Width: 4 Ln 1, Col 1 INS

```

FIGURE 3.7: Final Output Cluster

## Chapter 4

# Tools and Technology Used

### 4.1 Apache Hadoop

Hadoop is an open-source framework based on distributed storage and processing of data. It is used mainly for big data storage and its processing using distributed system. Hadoop is combined with many other tools of apache and its whole ecosystem can be used to perform analytics on data, prediction using data, data mining and in other machine learning applications.

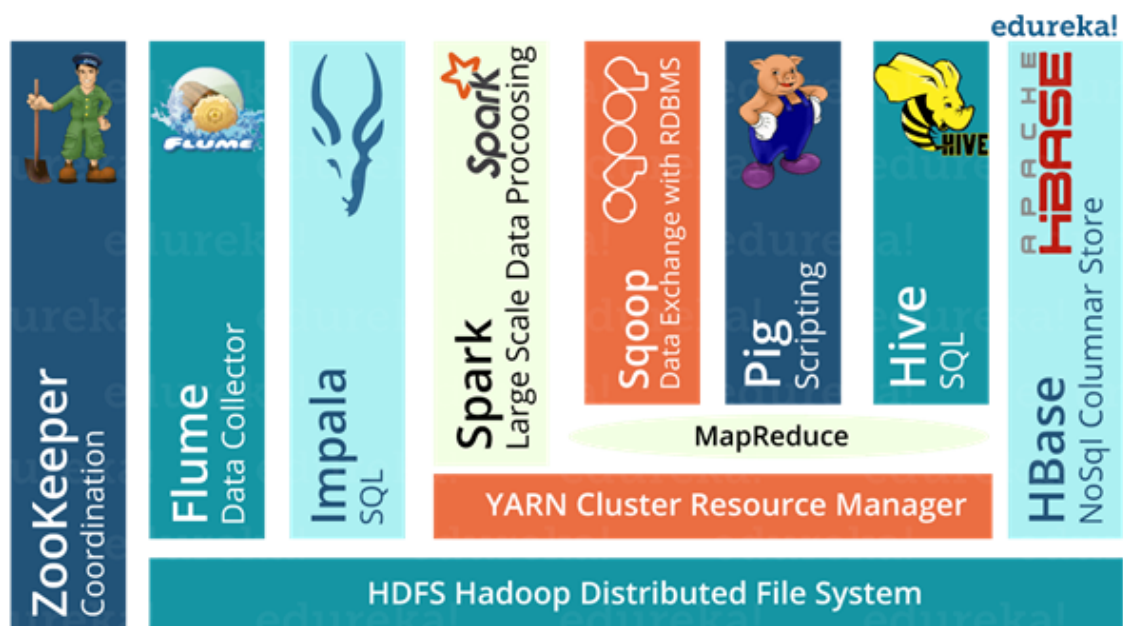


FIGURE 4.1: Hadoop Ecosystem

It also provides the support for the storage of structured as well as semi structured data. It provides flexibility to the users for storing, analysing the big amount of data. It is also easy to scale its performance and storage for increasing amount data.

Hadoop uses distributed file system and also different distributed processing technique which makes it best tool for data analytic and data mining application. Hadoop is capable of handling all the V's of Big Data i.e. it is very easy to maintain and work with the large volume of data, it is also able to handle various forms of data(structured, semi structured and unstructured data), it is also very efficient to collect the big data with the same velocity it is being produced in modern world from internet, servers, and mobile phones, social media, emails and sensor data from the internet of things IoT.

Hadoop is an Apache open-source framework, available for all. It is written in Java and allows the distributed processing of large data-sets across distributed system of computers. It uses simple programming models for processing. Single or thousands of servers can be used in hadoop ecosystem each offering local computation and storage and also scaling in it is quite efficient and easy.

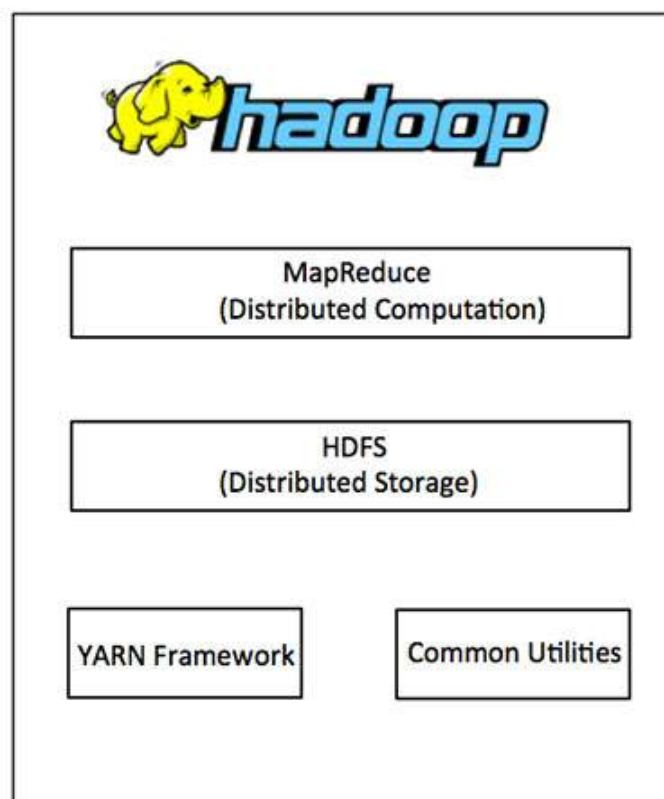


FIGURE 4.2: **Hadoop Layers**

## **Hadoop Architecture**

Hadoop has the following two major Parts:

1. One is the processing part of hadoop - MapReduce.
2. Second is the storage part - Hadoop Distributed File System (HDFS).

### **4.1.1 MapReduce**

MapReduce is a parallel programming paradigm for distributed applications. It provides the best approach to process large amount of data stored on distributed system. As Hadoop is a distributed system which uses commodity hardware to store data in a reliable and fault tolerant manner so this programming paradigm is well suited for that. The MapReduce program runs on Hadoop.

There are two important tasks involved in the MapReduce algorithm,

1. Map
2. Reduce

In MapReduce, first it perform map task and after this it perform reduce task. The Map takes input in the form of key, value pair also generates output in the form of key, value pair. In Reduce task it takes the output from a map as an input and then combines those data tuples into a smaller set of tuples.

The processing of data which are stored in distributed system is quite easy to perform with the help of map reduce, this is one of the best advantage of map reduce. MapReduce has two stages during processing of data, those are called mappers and reducers. Every data processing problem cannot be done in the form of mapper and reducer. But, if there is way in which data can be processed in the MapReduce form then its application can be very easily scaled for hundred thousand or more number of machines just by changing the configuration. This scalability attracts many programmers to use the MapReduce programming model.

Steps in a MapReduce job:

In this MapReduce paradigm, mapreduce program is sent to the machine where the data

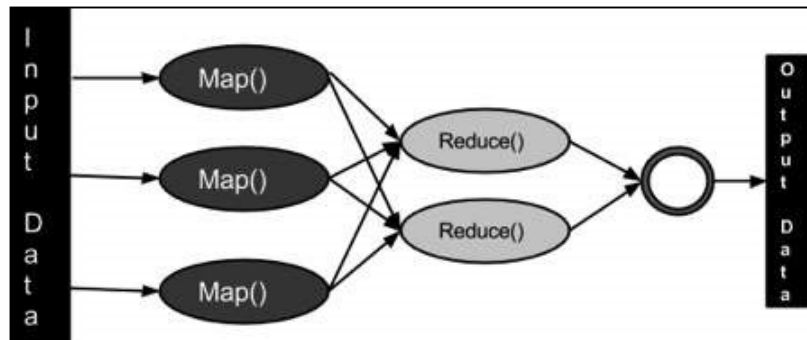


FIGURE 4.3: Hadoop MapReduce Stages

is stored instead of bringing data to the program as the data which had to be brought for the processing is too larger than the program.

The execution of mapreduce program involves three stages, map stage, shuffle stage, reduce stage.

1. Map and shuffle stage - This job process the input data. The data taken as input is generally stored in the Hadoop file system in the form of file or directory. The mapper function takes input file line by line and treat it as key, value pair. The address of the line is treated as key and the whole line as its corresponding value. Small chunks of data is produces as the output of the mapper.
2. Reduce stage – In this stage two process occurs one is shuffling and other is reduce. The Reducer takes the output of the mapper stage and process it further. After processing, the output produced is stored in the HDFS.

The Map Reduce tasks are performed on appropriate machines during a MapReduce job, i.e. on the machine where data relevant to the program is stored. It reduces the traffic as most of the data computation takes place on nodes on local disks.

#### 4.1.2 HDFS

Hadoop File System is based on a distributed file system design. Initially, Google published two papers to solve the problem of storing and handling of big data. In 2003 Google File System (GFS) was introduced and later in 2004 MapReduce was published. These were only published in papers but not implemented. But after some years Yahoo implemented both the papers which were published by Google earlier. In the year 2006-07 HDFS was

published, and in the year 2007-08 MapReduce was introduced. Later this project was made open source so that everyone can use it and can contribute to it which made the Hadoop very powerful tool to work as now there are many tools available which run on the top of it and has boosted its performance up to a great extent. It runs on commodity hardware which is configured in such a way that It provides highly fault-tolerance. HDFS can store a massive amount of data and provides a more convenient and efficient solution. It uses multiple machines to store such huge data, the files are stored across those machines. These files are stored on various commodity hardware (depends on the replication factor ) which provides security from any types of data loss, in case if any system goes down, this makes it fault tolerant. HDFS also makes applications available for parallel processing.

#### Features of HDFS

- It is the best option for distributed storage and processing.
- We can interact to HDFS with the command line also as provided by Hadoop.
- The built-in servers of name node and data node help users to check the status of the cluster quickly.
- Streaming access to file system data.

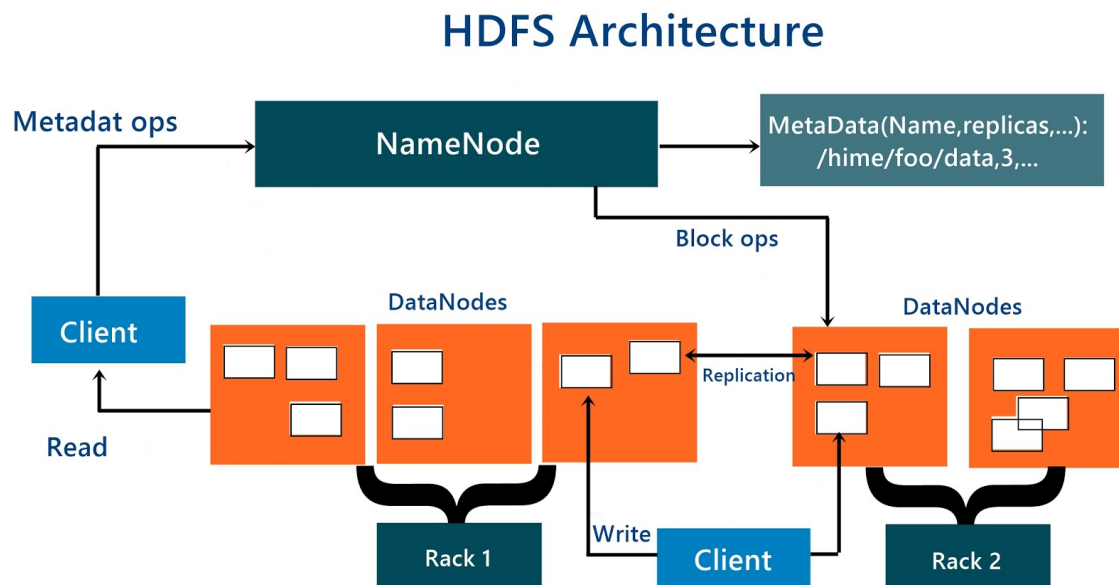


FIGURE 4.4: HDFS Architecture

HDFS follows master-slave architecture. There are the following elements in this architecture:

- **Namenode** - The namenode is one of the important element of the whole Hadoop environment. It acts as a manager for the Hadoop as it handles all the request of the client to store data on it. It also manages the metadata of the data being stored. Some of the metadata stored by name node are the filename, size, blocks assigned. Namenode also takes care of splitting data into blocks on Hadoop. If the block size of HDFS increases then the size of metadata of the file to be stored decreases. Namenode is maintained on high-level hardware (not the commodity hardware as if the namenode goes down whole data node will go down) that contains an operating system and the namenode software. The system having the namenode acts as the manager as it provides the following features -
  - It manages the stored files and the metadata of the file.
  - It also regulates the different client's permissions to access the file.
  - Namenode itself takes care of splitting the data into blocks on Hadoop.
  - It also executes file system operations such as renaming, opening, and closing files and directories.
- **Datanode** - The datanode is commodity hardware where actual data is stored. Each cluster has datanode to store data. The actual read and write operation is performed on the data node itself as instructed by the namenode. When a client wants to perform any operation like creation, deletion, or replication of files then it contacts to namenode and requests to operate, then namenode instructs the datanode according to the client's request to perform the corresponding operation on the data.
- **Block** - In general, all the files which are to be stored in hadoop are divided into many small chunks of data and then stored on same or different nodes of the cluster. The smallest segment of data which can be stored on Hadoop is called block. Hadoop have to perform read or write on atleast a single block at a time. Its default size is 64MB, but it can also be increased just by changing the configuration of HDFS.

HDFS has 5 services which are listed following:

1. Namenode
2. Secondary Namenode

3. Job Tracker
4. Datanode
5. Task Tracker

The above 3 services, namely namenode, secondary namenode and job tracker are master services and the rest below 2 services are slave services.

Every master services can talk to each other, and similarly, each slave services can talk to each other. Namenode is the master node, and its corresponding slave node is Datanode. Job Tracker is the master node, and its corresponding slave node is Task Tracker. Namenode can talk to Datanode, and similarly, Job Tracker can talk to Task Tracker. Now MapReduce is used for computation in Hadoop. So, to process the data MapReduce program is used, and instead of getting data to the program, we apply the program to the data. This being taken care of by job tracker. The job tracker assigns the task to the task tracker of each node of the cluster.

### **How Does Hadoop Work?**

Job tracker first asks namenode for the metadata of the data on which program has to be applied. Then it contacts the task tracker of (the nearest ) each node in which data is stored and ask to execute the program on its data, locally. This process is called map.

In the case when the system got suddenly crashed and went dead while performing map operation on input splits then task tracker informs job tracker that system has got down and mapping is not completed then job tracker assigns mapping task to another task tracker having same local data.

Task tracker use to sends heartbeat signal to job tracker in every 3 seconds that it is alive. If the task tracker didn't get any heartbeat till 10 heartbeats of time then task tracker automatically assigns that task to different task tracker with same data node. Along with the heartbeat task tracker also sends the number of available slots in the task tracker so that job tracker can assign more task to the task tracker if required.

Job tracker is also a single point of fail over as if it goes down entire process will go down so to avoid this it is maintained by high-level hardware.

The reducer takes care of all the output produced by each task tracker and produces the final output. The number of reducers is equal to the number of output files. Reducer can be in any system of the cluster. It combines all the result of the task tracker and process it



to produce the final result. It stores it in either its local data node or in other data node. During sending heartbeat it also mentions, that the data has been processed in particular data node and then namenode include it in its metadata.

## 4.2 Apache Spark

park is the computational tool developed by Apache Software. It has more computational power than Hadoop default computational engine, i.e. MapReduce. It has faster processing speed than MapReduce. It can also be used in standalone mode. It is also an open-source distributed framework. As it can also be used on the top of Hadoop, so can also be used as a cluster-computing framework. As spark supports the distributed system, so it also provides parallel processing of data. The main feature of spark is that it supports in-memory processing in contrast with MapReduce. In-memory processing means that the intermediate output that is produced during the processing of data is stored in the memory itself rather than storing it back on HDFS as done in MapReduce. This processing technique increases the performance of computation. It also provides some other features like SPARQL, which provides the user to perform interactive queries.

Features:

- Spark is quite faster than the traditional solution MapReduce of cluster computation. On memory, it is 100 times faster and ten times faster on the disk. The main reason for more rapid calculation is in-memory processing.
- Spark supports many languages for programming. The languages that are supported by spark are java, python, scala. So the users have a variety of option to work with Spark for developing applications for Hadoop.
- Spark also provides support of various libraries for different applications like ML, Deep Learning, Data Querying, streaming data and some standard algorithms.
- Spark also provides excellent fault tolerance towards data processing through Resilient Distributed Datasets (RDD) abstraction.
- Spark is also able to work with the real-time produced data, which was a big problem for the early version of Hadoop. In the early versions of the hadoop, it was only able to process the data which are present on the system.

### 4.3 Java

Java is a high-level programming language. It is a general-purpose programming language. Java is platform independent, but Java Virtual Machine(JVM), which is used to compile java platform is platform dependent. So suitable JVM is used to compile and run java in different platform of the machines. Sun Microsystems developed Java. It runs on different platforms like Windows, Linux, Mac OS, and so on. It provides many features like object-oriented, class-based, polymorphism, Inheritance, abstraction, and so on. One of the most popular features of Java is that it provides Write Once, Run Anywhere (WORA), that is Java program can be compiled once and can be run on all platforms without the need of recompilation.

The java program that we write is converted by java compiler into a binary file. It consists of byte codes. Byte codes are binary instruction for the interpreter. The Java interpreter inspects the byte codes, checks it out to ensure that it is safe to execute and then it performs the actions specified within the JVM. The Java interpreter can either run in standalone mode or it can be part of a web browser where it can be called automatically to run applets in a web page. It is easy to write, compile and debug a java program.

#### Main Features of JAVA

- Java is a platform-independent language - As java produces a byte code which is saved as .class file when it is compiled. The Java byte code present in the class file is then run on JVM when required. So the .class file produced after compilation can be run on any platform like Linux, Windows, and Mac OS etc. on the top of Java Virtual Machine. But Java Virtual Machine is machine and platform dependent. So the JVM required for windows will be different from the JVM required for the MacOS. However, the output produced with different class file or the different byte code of the same program can run on both of the devices. This makes Java a platform independent language.
- Java is a platform-independent language - As java produces a byte code which is saved as .class file when it is compiled. The Java byte code present in the class file is then run on JVM when required. So the .class file produced after compilation can be run on any platform like Linux, Windows, and Mac OS etc. on the top of Java Virtual Machine. But Java Virtual Machine is machine and platform dependent. So the JVM required for windows will be different from the JVM required for the

MacOS. However, the output produced with different class file or the different byte code of the same program can run on both of the devices. This makes Java a platform independent language.

- Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Java is an Object-Oriented language - Object-oriented programming is a way of organizing programs as a collection of objects; here object is an instance of a class. Four main concepts of Object-Oriented programming are:
  - Simple - Java is simple to use and provides many useful features to use like operator overloading, multiple inheritance, garbage collector and many more.
  - Robust Language - Java is one the most robust or reliable language. Its feature of garbage collection and exception handling makes it robust. It is also very easy to debug java programs.
  - Secure - Java also provides security from accessing the memory with pointers which are restricted to be used. So the problems like stack corruption or buffer overflow don't arise in java.
  - Multithreading - Java supports multithreading. This means that java can perform multiple tasks simultaneously .i.e the different part of the program can be executed in a parallel manner can be completed in less time. This also increases the utilisation of the processor as CPU, in this case, will also using its full capacity to run the program.
  - Portable - It is also very portable as the byte code generated can be carried to any platform for execution

## 4.4 Apache Hive

Apache Hive is an information distribution center framework based over Hadoop. It is utilized for breaking down organized and semi-organized information. It gives the deliberation to the complex Hadoop MapReduce. Fundamentally, it gives a structure to the

information and empowers us to perform queries written in HQL (Hive Inquiry Language that like SQL articulations). Hive compiler internally compiles these HQL inquiries into MapReduce jobs. Subsequently, Client doesn't need to stress over composition complex MapReduce programs, rather, HQL questions can be utilized to process information utilizing Hadoop. It is exceptionally valuable for clients who are OK with SQL. Since most information warehousing applications work with SQL-based questioning dialects, Hive helps convenience of SQL-based applications to Hadoop. It likewise underpins Information Definition Language (DDL), Information Control Language (DML) and Client Characterized Capacities (UDF). The hive was at first created by Facebook.

Advantages of Hive:

- It is very useful tool for the people which does not have a strong programming background as it provides easy HQL for directly querying the data
- It can easily be extended for the large datasets without any difference in the performance of the system.
- It is as an efficient tool for Extracting, Loading and Transforming the data.
- It stores the metadata information in RDBMS this reduces the time to check the semantics of the query.

### **Where to use Apache Hive?**

Apache Hive provides advantage of both MapReduce and SQL queries both. So it is used by various peoples in the industry. It is the best tool to work for the people who have a good command over SQL. It can also be used to perform data mining and data analytics in a very easy manner. Therefore Apache Hive can be used in the following fields:

- Data Warehousing
- Ad-hoc Analysis

## **4.5 Eclipse IDE**

Eclipse is an IDE (Integrated Development Environment) used for developing computer programs. It is the mainly used as Java IDE. It is written mostly in Java. It provides a base workspace to work on. It also supports various extensible plug-in system that can be used

to customize the environment. It is primarily used for developing Java applications. But can also be used to develop various applications written in various other languages like Ada, C, C++, C, Python, R, Ruby (including Ruby on Rails framework), and many more via plug-ins. Documents can also be developed in eclipse by using LaTeX and packages for the software Mathematica in eclipse. Eclipse also include development tools for different languages like for java and scala Java Development Tool is there, for C/C++ there is Eclipse CDT and Eclipse PDT for PHP.

Initially Eclipse development kit contains only the Java development tools, which is used for Java development. But Users can use eclipse plug-ins so that it can also be used for developing other languages. It is free and open-source software so anybody can contribute to it in addition to provide some features or support for some other languages.

## 4.6 Apache Parquet

parquet is a top level apache project which has gained momentum over past few years. Basically since its inception. parquet is referred as columnar storage format so it is a storage format or facilitates serialization that optimises various data i/o aspects. Some the key benefits to use parquet are as follows:-

- Efficient storage
- Efficient data io and cpu utilisation
- interoperability

while working with big data , storage is not very expensive but its optimisation plays quite significant role like in terabyte region optimising even 5-10% of storage is a great achievement. So apache parquet allows a much more optimise and compressed data storage so there are a lot of data efficiency when using parquet primarily for storage standpoint. Generally tables are stored row wise for example if a table is having three columns a, b, c then in storage firstly value of a, b and c of 1st is stored then 2nd and then son on, but in parquet format data is stored column wise i.e firstly all the data for column a is stored then column b is stored and then c. As most of the queries used is on the columns so if data is stored column wise then it will also provide great performance for the queries also. But in contrast in row wise storage, the read and write operation for queries will become very expensive as in most cases it has to read the whole dataset for getting the value of simple

query. As parquet is columnar data storage so is universally understood by all the tools of the hadoop ecosystem. So if the user project includes different hadoop tools like spark, hive, or any other then it very much provide interoperability.

## 4.7 Apache Avro

Apache Avro is a data serialization system which is independent of any language. It was developed by Doug Cutting. It provides data formats. It is very helpful because it provides data formats that can be utilised by multiple languages. It is a useful tool for data serialization in Hadoop. Avro has a construction based framework. It's read and write tasks are related with a language-free pattern. It is utilized to serialize the information which has a worked in construction. It serializes the information into a conservative paired configuration, and afterward this twofold organization can be deserialized by any application. Information structures statement for Avro is done in JSON group. It bolsters different dialects, for instance, Java, C, C++, C, Python, and Ruby.

## Chapter 5

# Results and Snapshots

Analysing the different size of dataset by comparing their size of output cluster generated corresponding to their input and also their loading time.

Dataset	I/p Size(in GB)	O/p Size(in GB)	Loading Time(in min)
University10.nt	0.214	0.053	1.04
University100.nt	2.28	0.577	7.35
University500.nt	11.45	1.98	50.05

TABLE 5.1: Size and loading times using Lubm Dataset

The following table is showing the number of properties generated in binary and n-ary tables for the “University100.nt dataset”.

Total no. of Properties	18
No. of properties in Binary Table	8
No. of properties in N-ary Table	10

TABLE 5.2: Schema Breakdown for University100.nt Dataset

”University100.nt” dataset is used and the total number of tables generated is shown by the table given table below.

Datasets	Binary Tables	3-ary Tables	4-ary Tables	4-ary Tables	Total Tables
University10.nt	8	1	0	2	12

TABLE 5.3: Property Distribution Table

Below Table 5.4 shows the time taken by different type of queries to access different tables for different datasets. Here vp stands for “Vertical Partitioning” or the “Binary Tables” and WPT stands for “Wide Property Tables” or “N-ary Tables”.

Query	Time on PProST(in ms)	Time on Data-Centric Storage(in ms)
VP	11874	11340
WPT	4141	2860
VP & WPT	6329	5192

TABLE 5.4: Query Processing Time for ProST and Data-Centric

Snapshots of the resulting schema generated

## Browse Directory

/ritesh/output/relationalTable/binary\_table

Go!

Show25entries

Search:

		Permission		Owner		Group		Size		Last Modified		Replication		Block Size		Name	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		owl_imports	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_advisor	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_headOf	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_publicationAuthor	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_researchInterest	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_subOrganizationOf	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_teachingAssistantOf	
		drwxr-xr-x		eict		supergroup		0 B		May 16 16:27		0		0 B		ub_undergraduateDegreeFrom	

Showing 1 to 8 of 8 entries

Previous

1

Next

FIGURE 5.1: Binary Tables



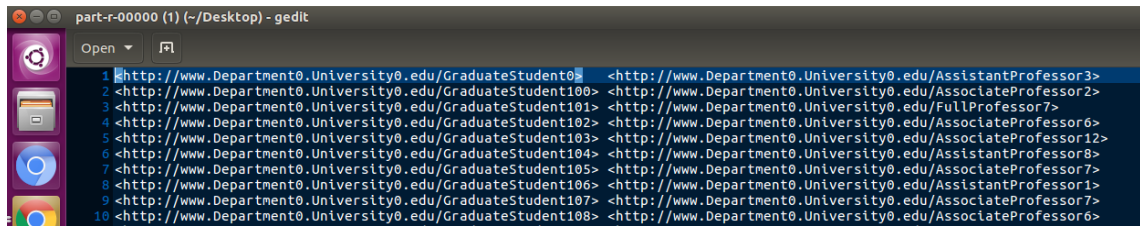


FIGURE 5.2: Binary Table Format (ub\_advisor)

## Browse Directory

/ritesh/output/relationalTable/wide\_property\_table

Go!

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	eict	supergroup	0 B	May 17 17:42	0	0 B	0
drwxr-xr-x	eict	supergroup	0 B	May 17 17:42	0	0 B	1
drwxr-xr-x	eict	supergroup	0 B	May 17 17:42	0	0 B	2

Showing 1 to 3 of 3 entries

Previous 1 Next

Hadoop, 2017.

FIGURE 5.3: Nary property Tables

## Browse Directory

/ritesh/output/relationalTable/wide\_property\_table/0

Go!

Show 25 entries

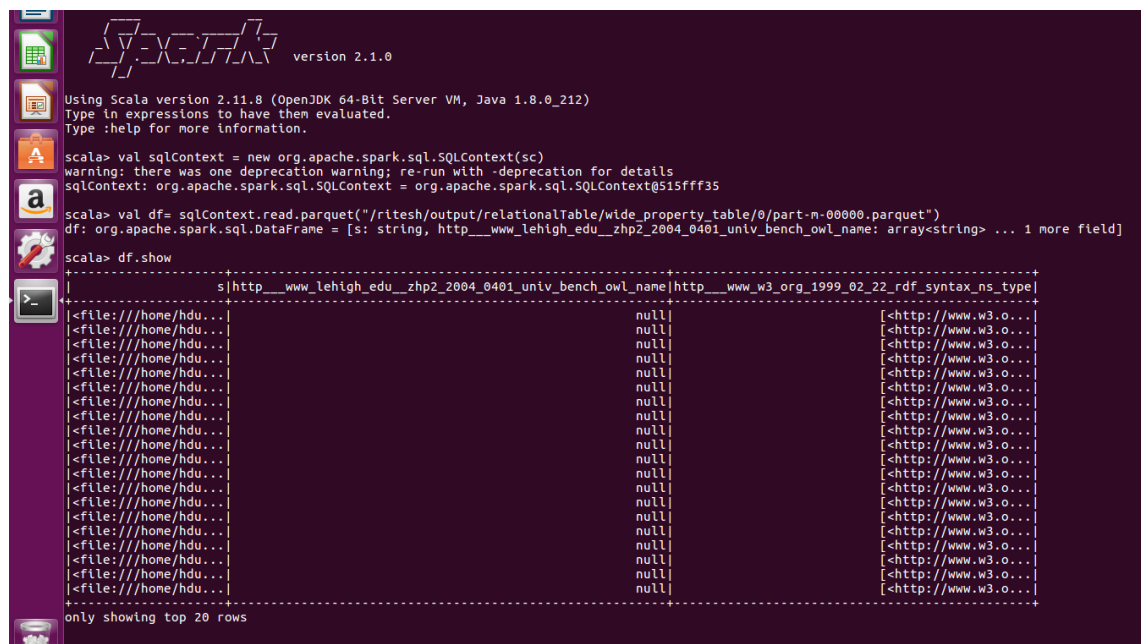
Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	eict	supergroup	0 B	May 17 17:42	3	128 MB	__SUCCESS
-rw-r--r--	eict	supergroup	599 B	May 17 17:42	3	128 MB	__common_metadata
-rw-r--r--	eict	supergroup	1.73 KB	May 17 17:42	3	128 MB	__metadata
-rw-r--r--	eict	supergroup	12.18 MB	May 17 17:42	3	128 MB	part-m-00000.parquet
-rw-r--r--	eict	supergroup	1.78 MB	May 17 17:42	3	128 MB	part-m-00001.parquet

Showing 1 to 5 of 5 entries

Previous 1 Next

FIGURE 5.4: N-ary Table in Parquet Format on HDFS



```

scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@515fff35

scala> val df= sqlContext.read.parquet("/ritesh/output/relationalTable/wide_property_table/0/part-n-00000.parquet")
df: org.apache.spark.sql.DataFrame = [s: string, http__www_lehigh_edu_zhp2_2004_0401_univ_bench_owl_name: array<string> ... 1 more field]

scala> df.show
+-----+-----+-----+
|s|http__www_lehigh_edu_zhp2_2004_0401_univ_bench_owl_name|http__www_w3_org_1999_02_22_rdf_syntax_ns_type|
+-----+-----+-----+
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
|<file:///home/hdu...|null|[<http://www.w3.o...|
+-----+-----+-----+
only showing top 20 rows

```

FIGURE 5.5: Nary table as read from parquet file

## Chapter 6

# Conclusion and Future Work

### 6.1 Conclusion

We have displayed an data driven construction creation approach for putting away RDF information in relational databases. This methodology separates an essential structure from RDF information and accomplishes a decent harmony between utilizing n-ary tables and two-fold tables to tune RDF stockpiling for efcient inquiry preparing. Initial, a grouping stage nds every single related property in the informational collection that are contender to be put away together. Second, the groups are sent to a partitioning stage to advance the capacity of additional information in the basic database. We contrasted our data driven methodology and best in class approaches for RDF stockpiling, to be specific to the triple store and decomposed storage. This methodology demonstrates enormous improvement over the triple store and the disintegrated methodology.

### 6.2 Future Work

Extending this project in the future, the following tasks can be implemented:

1. Approach can be implemented to handle multivalued properties.
2. To handle cases regarding Reication (It is an RDF property that allows statements to be made about other RDF statements).

# Bibliography

1. World Wide Web Consortium (W3C) : <http://www.w3c.org/>
2. W3C Semantic Web Activity: <http://www.w3.org/2001/sw/>
3. PRoST (Partitioned RDF on Spark Tables) : <https://arxiv.org/pdf/1802.05898.pdf>
4. LUBM Dataset : <http://swat.cse.lehigh.edu/projects/lubm/>
5. Data-Centric Storage : <https://ieeexplore.ieee.org/document/5175913>
6. Avro Schema : <https://avro.apache.org/>
7. Apache Parquet : <https://parquet.apache.org/>
8. PRoST Project : <https://github.com/tf-dbis-uni-freiburg/PRoST>
9. J. J. Levandoski and M. F. Mokbel, “RDF Data-Centric Storage”, University of Minnesota, Tech. Rep. UM-CS-TR-9999, 2009.
10. Matteo Cossu, Michael Färber and Georg Lausen, “PRoST: Distributed Execution of SPARQL queries Using Mixed Partitioning Strategies”, University of Freiburg