# Improved performance of RDF data using HIVE, PIG in Hadoop

***Thesis Report***

*submitted in partial fulfillment of the requirements*

*for the award of degree of*

**Master of Engineering**

in

***Computer Science and Engineering***

*Submitted By*

**Anshul Chandel**
**(801432005)**

Under the supervision of:

**Dr. Deepak Garg**
**Professor & Head of Department**

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

**June 2016**

# CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, ***"Improved performance of RDF data using HIVE,PIG in Hadoop"*** , in partial fulfillment of the requirements for the award of degree of Master of Engineering in ***Computer Science and Engineering*** submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of ***Dr. Deepak Garg*** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Anshul Chandel
801432005
ME(CS)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

**Dr. Deepak Garg**
Professor & Head
Computer Science and Engineering Department
Thapar University

**Countersigned by**

12/6/16

**Dr. Deepak Garg**
Professor & Head
Computer Science and Engineering Department
Thapar University

**Dr. S. S. Bhatia**
Dean(Academic Affairs)
Thapar University

# ACKNOWLEDGEMENT

*No volume of words is enough to express my gratitude towards my guide,* ***Dr. Deepak Garg*** *,Professor & Head of Department, Computer Science and Engineering Department, Thapar University, who has been very concerned and has supervised the work presented in this thesis report. He has helped me to explore this vast field in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.*

*I am also thankful to Dr. Ashutosh Mishra, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.*

*I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis.*

*Most importantly, I would like to thanks my parents, friends and the almighty for showing me the right direction out of the blue, to help me to stay calm in the oddest of the times and keep moving even at times when there was no hope.*

***Anshul Chandel***

***(801432005)***

# ABSTRACT

Semantic Web Data is an efficient way to represent data in World Wide Web. Semantic data is not concerned about the structure, it is concerned about the meaning of data. In the modern generation of "Semantic Web Data" and its rapid growth requirement is well managed storage and evaluation. So cloud data services play an important role. These services are based on the MapReduce Programming Model. MapReduce Programming model is famous for its scalability, parallel processing, cost-effective solution and flexibility. Hadoop is an open source implementation of MapReduce. Hadoop has two components , one for storage part that is HDFS(Hadoop Distributed File System) and other for processing part that is MapReduce. Hadoop based extensions such as PIG and HIVE are query languages which provide high level data flow. Although SPARQL is a query language for RDF(Resource Description Framework) and it is also considered as the backbone of the semantic web based applications. Here we introduce HIVE and PIG for querying RDF data. Here we have a dataset consist 5000 triples and then we execute our queries which are based on HIVE, PIG and SPARQL on this dataset. The goal of this thesis is to compare the results of SPARQL, HIVE and PIG and analyze the retrieval time for a query in RDF data. Finally we can conclude which framework will work fast and scalable for our dataset.

# TABLE OF CONTENTS

# List of Figures

# List of Tables

# List of Abbreviations

**RDF:** (Resource Description Framework)

**SPARQL:** (SPARQL Protocol and RDF Query language)

**URI:** (Uniform Resource Identifier)

**URL:** (Uniform Resource Locator)

**HDFS:** (Hadoop Distributed File System)

**DDL :** Data Definition Language

**DML :** Data Manipulation Language

**CSV:** Comma Separated Value

**QL :** Query Language

# 1. INTRODUCTION

With the advent of semantic web data new challenges regarding the query evaluation rose. Semantic Web data represented as Resource Description Framework (RDF) is growing rapidly. RDF is a core technology of the semantic for representing data in machine-readable formats. RDF is very useful in semantic web and data integration. It gives a universal model of data which is a milestone in data integration. RDF provides a structure which is in form of the subject–property–object expressions. These structures are called as triples. Subject like resources, property or predicate denote the relation between subject and object. Subject can be a either a uniform resource identifier(URI) or a blank node. Objects are also known as literals, object can either contain a value or a URI. SPARQL is the standardized query language specifically designed for RDF, just as SQL is the standardized query language for the relational type of databases. A single SPARQL query comprises of specific set of triplets where the subject, the predicate and/or object constitute the variables. The idea behind this format is to match triples in the SPARQL query with the already existing RDF triples and consequently find the solutions of the variables. It is a challenging task to query an RDF dataset but with the use of Hadoop MapReduce infrastructures, we can spread the processing across the clusters. One of the primary concerns of the thesis is to study the big data challenges faced by developers due to its volume, variety and velocity. This thesis presents the arms of the Big Data such as Hadoop, HIVE and PIG which are used to provide innovative solutions for all the big data challenges.

## 1.1 RDF

RDF stands for resource description framework. This framework is used to represent data in semantic web data. We can say, data store in RDF is same as data store in triplestore.RDF provides universal format to store the data. Using RDF is extremely good in data integration in semantic web network. We can represent directed labeled graph into resource description framework. RDF can be represented in xml or triples. RDF is a set of triples that constitutes:

- **subject:** These are resources and identifiable by URI. Sometimes we represent node of a graph as a subject.
- **predicate:** These represent relationship between two. In the graph edges show the relationship between nodes that is same as predicate in RDF.
- **object:** These are same as subjects that means these are also resources but only one difference is that object has also values but subject does not**.**

RDF is very useful in graph, in which node are represented as a subject and object and edge is represented as predicate or relation. It is a powerful framework to represent data in semantic web network. Example a triple ( amit, country, india) that is a record of RDF framework. In this example amit is a subject and country shows the relation, and India is the value or can be an object. Every framework has its own language and RDF has its own language that is SPARQL .

SPARQL is a standard query language for RDF data. SPARQL stands for SPARQL protocol and RDF Query Language. SPARQL is dominated by join operation. Basically SPARQL is quite similar to SQL and most of the syntax or operator is same as SPARQL. We present SPARQL example in the upcoming chapter. Every SPARQL consists three arguments namely subject, predicate and object. Every query contains these three parameters and each record of RDF database matches with SPARQL query. Every SPARQL has select statement and it selects the given column. So finally it is query language for three attributes.

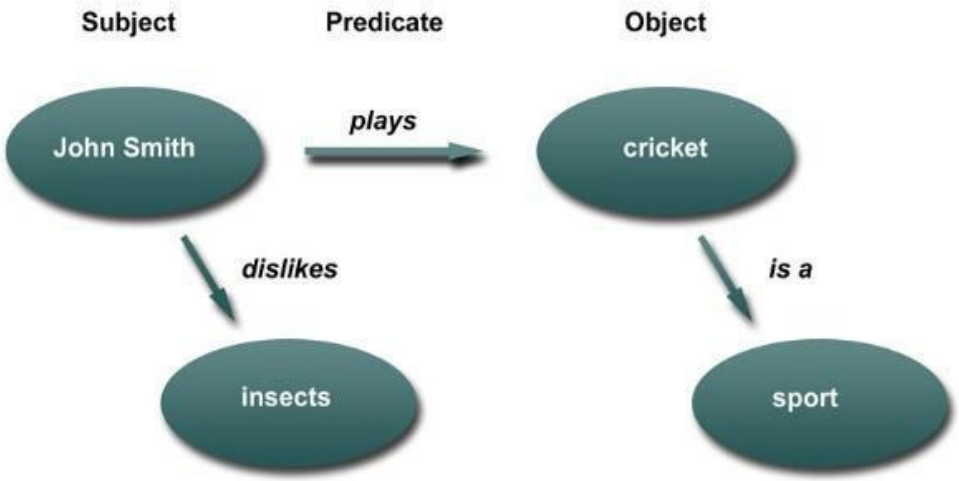


**Figure 1.1 Example of query language for three attributes**

Here we want to show how we can define many relationships into one diagram. In the above diagram, we understand what is RDF and what are the components of RDF. Here we will understand what are these and how it will work.

In the above figure, we can see the relationship diagram. We all know every RDF document consists of three parts one is subject, predicate and last object. In the above figure john smith plays cricket, means john smith is a subject, play is a relationship and cricket is a object. Object can be value or another URI depending on the user and where it will be used. Here in the second case, cricket is a sport, cricket is subject and represents IS-A relation whereas sport is an object. So we can see we have used cricket as a subject in one query and as a object in another query. Similarly we can see in the last relationship diagram john smith dislikes smith. Here john smith is a subject, dislikes is a predicate and insects as a object. So that all about RDF document. So here we could see how we define relation between these two.

## 1.2 Hadoop and Map Reduce

While we are considering analytics in applications using cloud environment and the large scale applications, the recent de facto standard developed is the Map-Reduce programming model that was made popular by Google along with its open source implementations primarily the Hadoop. All such platforms were explored to perform graph pattern matching operations. Map-Reduce based systems were explored to look for indexing of RDF graphs, reasoning and scalable graph pattern. Over the course of time the Map-Reduce computing paradigm has emerged as an indispensable component of the traditional large-scale processing solutions. Now the Apache Hadoop, which forms the widely known open source Map-Reduce implementation has gained its popularity owing to the feature of its support for automatic parallelization of programs and fault tolerance over a cluster of commodity-grade machines. Thus providing an cost-effective approach for large scale data processing, easy-to-manage and easily scalable. In the MapReduce programming model, User encode their tasks as map and reduce functions, which are executed in parallel on the Mappers and Reducers respectively. This two-phase computational model is associated with I/O overhead due to the data transfer

between the Mappers and the Reducers and an inherent communication. Hadoop based systems like Pig provide high-level query languages that improve support and usability for the automatic data flow optimization which is as similar as the database systems.

Now to process the related RDF query patterns we typically require several join operations which thus due to the fine grained nature of RDF data model. For now the Hadoop is supporting only partition parallelism in account of which a single operator executes on different partitions of data across the nodes. By the result of which, The existing Hadoop based systems with the relational style join operators translate multi-join query plans into a linear execution plan with a sequence of multiple MapReduce cycles. The linear execution plan can thus significantly decrease the overall performance of the query processing in large amount of database. This has resulted in a much shorter execution workflow.

We now propose an approach to increase the degree of parallelism by enabling some form of parallel processing. Here instead of MapReduce, we used HIVE and PIG technique . PIG and HIVE are the components of Hadoop ecosystem. These are the high level data flow languages. MapReduce is the inner most layer of Hadoop ecosystem .

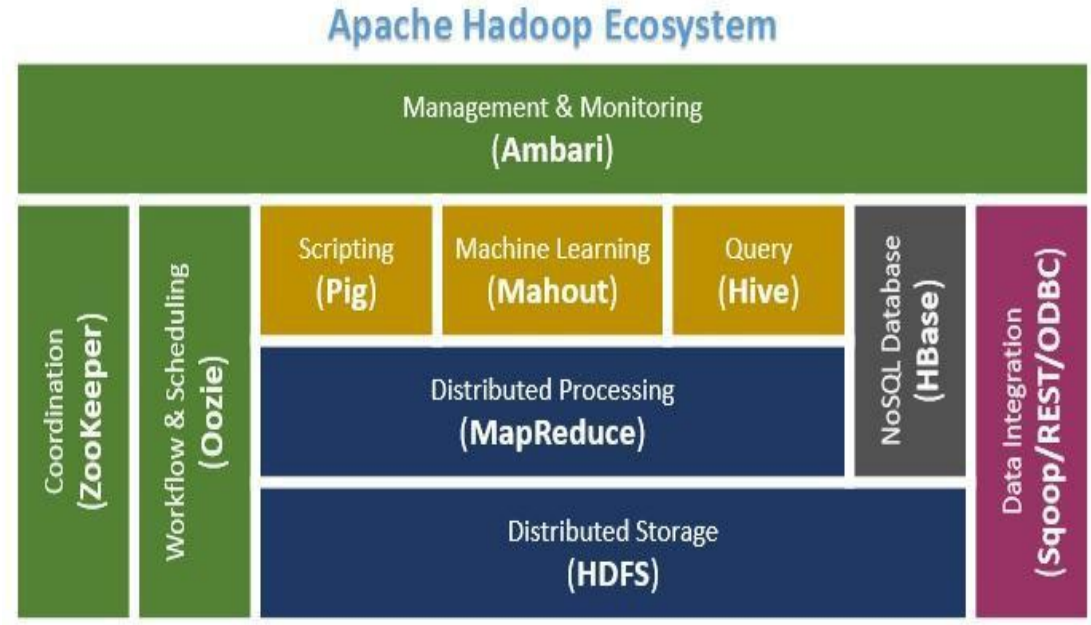


**Figure 1.2 Hadoop Architecture**

Basically Hadoop has two parts for storage and processing purpose. HDFS works as a storage part and MapReduce works as a processing part. So here the question comes why PIG and HIVE come into picture. This is because MapReduce contains a huge line of code for a small program so for it needs a lot of code a big program so that is a big problem of developer. To eliminate this problem we introduce PIG and HIVE. They are the Hadoop based extensions that work on the upper layer of Hadoop architecture. The figure above shows Hadoop architecture. In the above figure shown that MapReduce is working at the inner level and HIVE and PIG are just above the MapReduce. So it can convert its query to MapReduce programming and finally it can store the data into the HDFS which is storage part and provide distributed storage.PIG and HIVE both have their own shell and both have their own schema and language which will be explained in next part.

## 1.3 Apache PIG

Pig is an open-source high level data flow architecture. PIG provides its own shell and a simple language called PIG-LATIN that is for queries and data manipulation, PIG-LATIN is compiled to MapReduce jobs that run on Hadoop Now the question why PIG is created when we already have MapReduce, So the answer is MapReduce contains a lot of java code. Those who do not know java, they face some difficulty about that work so for those who have lack of knowledge about java, they can prefer PIG. Pig was originally developed by Yahoo in 2006, for those who have an ad-hoc way of creating and executing MapReduce jobs on very large data sets.

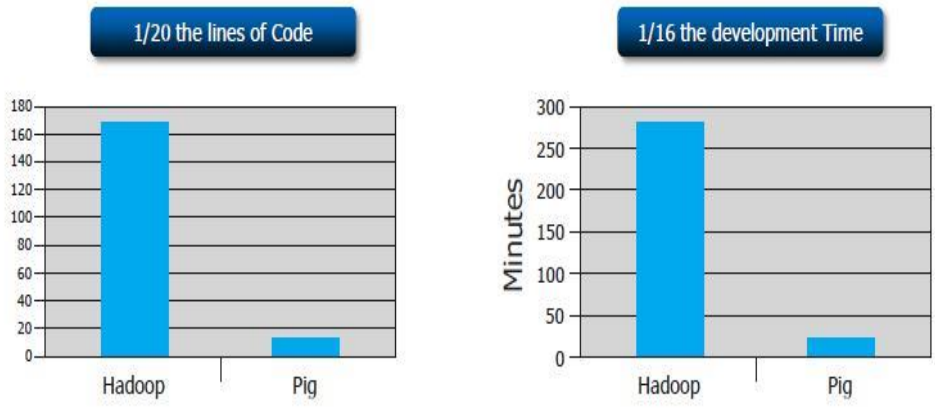


**Figure 1.3 Comparison between Hadoop and PIG**

The above figure shows PIG is quite better then Hadoop and MapReduce in terms of development time and line of code. In terms of development time PIG is 16 times faster than Hadoop and in terms of line of code it is 20 times more compact than Hadoop. PIG was created to reduce the development time. PIG has multiquery approach to answer. Pig is also created for professionals from non-Java background, to make their work better and in efficient way.

PIG-LATIN is simple language in which user can write their own function to read, write and process the data. PIG-LATIN has various inbuilt operators which can be used to load, store and do other operations .One of the big advantages of PIG is its user defined functions like java and other language. Let move to its architecture, though PIG-LATIN is a language then it is having a compiler, parser, linker, loader and many more features. First step PIG-LATIN script goes to parsing phase. Then parser constructs the parse tree of tokens, to check syntax and semantic mistakes of query. In parsing phase , it uses bottom up parser because bottom up parser is more powerful then the top down parser. In bottom up parser, Parsing can be done in the bottom up manner. After the parsing phase the second phase is compilation phase. In that phase it can check all the meanings and user defined function format so that all things should be verified in that phase. In next phase the optimization required. In this phase the code optimization is done means unnecessary instruction should be removed , for example dead code elimination and loop jamming etc.

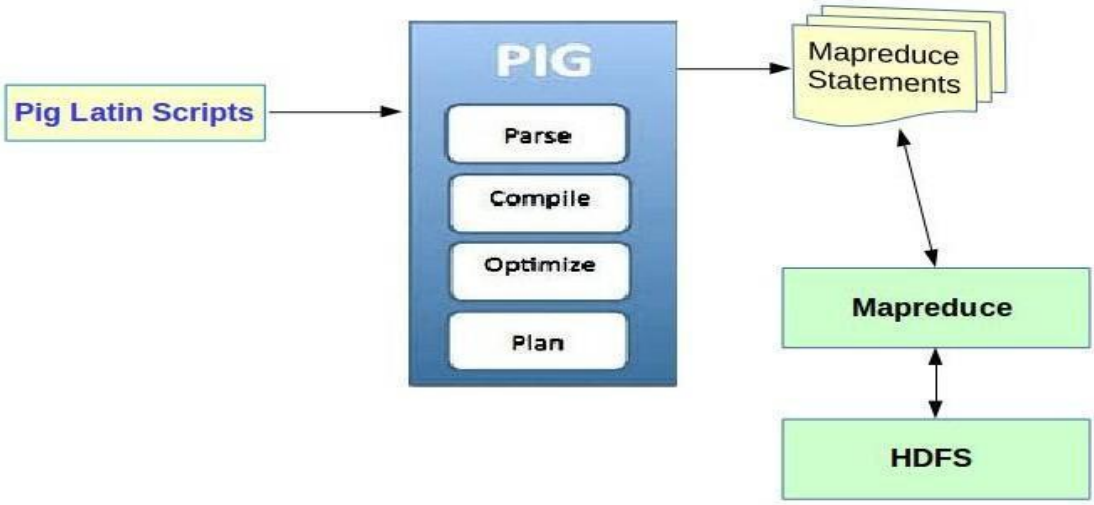


**Figure 1.4 Pig Architecture**

Final is the plan phase, only planning can be done here because we have no work to do in that phase. Finally the complied data is moved on to the MapReduce and executes the job. So here we show the diagram of PIG-LATIN structure.

Here we want to show some query example of PIG-LATIN, and how to load data and process. PIG has two modes, one is local node and other is MapReduce mode.

When we would work with our local file or file which is store in the local disk means data is stored on our machine then we work on the local mode . To open the PIG shell that is called grunt shell in Hadoop ,the command to open grunt shell on local mode in pig we should write on terminal , 'pig -x local' that command open PIG on local mode .

Suppose we want to load a file which is stored in HDFS then local mode will not work. If our file is located in HDFS and we should move on to MapReduce mode. Command is 'pig -x mapreduce'.

Suppose we want to load a file whose name is 'int.txt' that is stored in HDFS , first we want to load text file then perform operation .

Suppose we want to count to find mail id of some person through a file.

tmp = load '/t/c/m/int.txt' using pigstorage(',') as (name: chararray, info:chararray: comm:chararray)

emp= group tmp by comm;

data = filter tmp by name == 'craig' and info =='mail';

This command is filtering the database according to our query. In the first line we loaded our text file "int.txt" into PIG storage and define schema for our table, or we defined the structure of our table.

In the second line we arranged our table according to our group. In the last line we filtered our table. Means we compared each record with name 'craig' and info 'mail'. After the execution of this query we can see the result.

## 1.4 APACHE HIVE

HIVE is basically platform for structured data, HIVE is high level data flow architecture. HIVE has its own shell and language called HIVE-QL. HIVE is quite similar to SQL. As the same work as PIG, HIVE is used for structured data and it is complied into MapReduce job in Hadoop. HIVE is quite more understandable than PIG because HIVE is quite similar to SQL. People know about SQL very well. HIVE is designed for analysis operations. Basically HIVE is developed for the analytical purpose because data analytics is a big issue in this era. HIVE is using for log processing, text mining, document indexing, business intelligence, predictive modeling , hypothesis testing and many more area. HIVE was developed by Facebook, later the Apache Software Foundation took it up and developed .It further is an open source under the name Apache HIVE. HIVE has been used by many organization such as Amazon. HIVE is not a relational database but queries are quite similar to SQL. We can say it is designed for Online Transaction Processing (OLTP). HIVE stores schema into its database and processed data into HDFS. HIVE is fast and scalable.

HIVE has various components such as shell, meta store, compiler, execution engine , driver. HIVE supports primitive data types such as integer, float, string and many more. HIVE also supports complex data type array: map, structs: struct. HIVE supported DDL statements like create table, alter table, drop table. These statements are used for partitions, views etc. In HIVE load statement is used to move data HDFS into the HIVE. Insert statement is used to insert data into the table. Select statement is used to select the data from the table. these are the general statement used for data manipulation statements (DML). DDL and DML both are same as SQL but the processing is quite different from SQL because SQL execute query and store in local database but HIVE executes query and store in the HDFS. Here we want to show some other features of HIVE. One of the wonderful feature of HIVE is that it is scalable and fast. HIVE is fast because of parallel processing. HIVE is easier to understand and quite user supportive. We can use impala or Sempala if we want. Hive is comfortable to use with these two. HIVE provides meta data to store the schema of a table or data.

HIVE has its own shell and engine. So it does not need any type of external event generator. HIVE directly communicates with the MapReduce and directly stores the data into HDFS or either it can fetch data from HDFS. Actually HIVE came in picture in when people needed analytics. First HIVE is used by Facebook in 2005. HIVE is very good for loading the data, reading the data and storing the data. Figure below shown the architecture of HIVE.

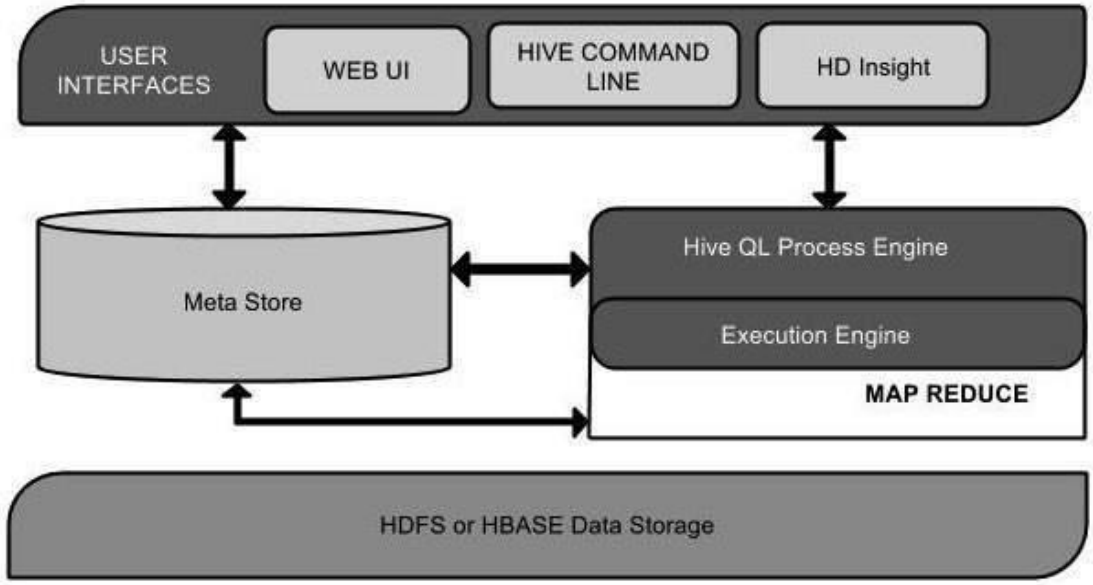


**Figure 1.5 Hive Architecture**

HIVE has three layer architecture. In the first layer, that contains user interface can be graphical user interface or web user interface or HD insight. The first layer responsible for visibility scheme where user can write their query or we can say it is the front end of HIVE architecture. In the second layer that contains main components of HIVE architecture that contains Meta Store, HIVE-QL Process Engine and Execution Engine. In the Meta Store HIVE stores schema of a table or structure of a table or Metadata of a tables, HDFS mapping and their data types , columns in a table and database Another components like HIVE-QL process Engine, HIVE-QL is similar to SQL for querying on schema info on the Metastore. It is one of the replacement of traditional approaches for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for

MapReduce job and process it. Mainly HIVE-QL process Engine is executing query on the MapReduce platform.

The conjunction part of HIVE-QL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce. Here we want to show some example query on HIVE. Here we want to load a text file into HDFS and apply some operation on that. Ex. "create table if not exists employee (name string, info string, comm string) ;" that statement creates table employee with schema. This table has three column namely name , info and comm . Then now we move on to load the table from outside. So the statement: "load data from local inpath '/tmp/mydata/myfile.txt' ", This second statement explains that we want to load the text file from the given path but we make sure that number of column in text file and employee should be same and the next command overwrite the loaded text file into the schema. "overwrite into table employee"; that command overwrite the content into the table schema. Now we have table with our content in HDFS, Now we can perform all the operations.

Now we want to put some operation on the table, first we want to select some column and put the condition. Ex:

"select comm from employee"

In the above query we want to select the column comm from employee without any condition.

"select com from employee where name = 'craig' and info = 'mail';"

Here we put some condition on the pervious statement, so when this query will executed then it selects record with name 'craig' and info 'mail'. So finally we can see HIVE query is similar to SQL so that’s why HIVE is quite similar to SQL. So mostly user prefer HIVE instead of PIG and other framework.

HIVE and PIG both are scalable and fast framework, but there is many differences between HIVE and PIG. Actually PIG takes advantage in terms of user defined function means PIG supports user defined functions but HIVE does not support user defined functions. If we want to write on our data which is stored in dataware

house then we can go with PIG instead of HIVE because HIVE is used for analytics purpose.

# 2. LITERATURE REVIEW

In present ,Google and other search engines are best example of big data and semantic web. In the present era, data organization and data analytics are big issues. In recent years data increases exponentially, data can be unstructured or structured format. If data comes in structured format then we have no problem because to handle structured data we have RDBMS and many other framework. But if data comes in a unstructured format then we have problem because we cannot handle such a huge amount of data, [6].

This paper [1] explained basic features of RDF and told about challenges of semantic web analysis. semantic web analysis is explained by using RDF and it also told how SPARQL is useful to analysis semantic web data. Basically we convert semantic web data to RDF format , then we execute our query on the basis of that. Now the big problem comes as to how we can convert the semantic web data to RDF format because data comes from different ways and it has no universal format that means data comes from different resources have different schema so if we want to integrate that data then we have a universal format of data, for this problem RDF comes in a picture. RDF format states that we should divide our database into the three column that means our database has maximum or minimum three columns. RDF provides the universal structure of data for data integration. We know that every framework has its own language so SPARQL is a language for RDF. So we know when we divide our database into three columns then our database size increases. When database size is very large after dividing into RDF, size will be increased. So for efficient processing we need a parallel processing to reduce the time complexity.

Paper [2] explained how to process RDF into MapReduce platform. In this paper told about adoption of the Linking huge Open Data sets has led to a significant flow in the amount of Semantic Web data, particularly RDF data.

This has led to the positioning of the issue of scalable data processing techniques for the RDF format as being a critical issue in Semantic Web data analysis. The RDF data model which serves as the fine-grained data model basically represents relationships as binary relations. Now it requires a lot of join operation for

answering the simple query. On one hand where the MapReduce based processing is emerging as the *de facto* paradigm for processing huge amount of data, it is known to be inefficient for join-intensive workloads. In addition to this most of the existing techniques meant to optimize RDF data processing are not just sufficient for transferring well to the MapReduce model and significantly require lead time for pre-processing. Such a requirement will not be desirable for *on-demand* cloud database which contain huge data where the goal is to reduce the time.

One drawback of the MapReduce model is that users should convert their works into refined map and reduce code. Apache HIVE and PIG solved this issue by giving dataflow languages in Hadoop. There are not many approaches that consider the comparison of SPARQL, PIG and HIVE together. There are some papers published that address the HIVE techniques to retrieve data our system depends on translating the SPARQL query into HIVE-QL and PIG and reduce the query processing time in SPARQL. However our work needs an optimization for transforming of SPARQL to PIG and PIG to SPARQL also demands to filter its data model.

Paper [3] explained about nested triple groups approach. Existing approach support relational join operators which a simple multi join query to many MapReduce cycles. That results a high communication and I/O cost because transferring data between map phase and reduce phase more . SPARQL pattern is dominated by join operation so to reduce the time this paper introduced triple groups that are managed by set of triples .This paper also explained about nested triple group algebra and another approach which is called RAPID+ approach that is also optimization approach. So now we compare our existing approach with the RAPID+ approach.

In paper [11] explained about complexity of MapReduce and also gave solution of complexity problem.. It also told about the data sets size is increasing rapidly day by day being collected and analyzed in the industry for business intelligence, making classical warehousing solutions prohibitively expensive. Hadoop [3] is a popular approach with map-reduce implementation that is used as a solution for large data sets. Hadoop is work as store and process model means first store the data then process. However, the map-reduce programming model is complex because it requires developers to write custom programs which are not reusable.

Here we can convert our SPARQL query to PIG because SPARQL is dominated by join operation then we introduce RAPID + and nested triple group to reduce the time. we further move on to HIVE that is also a Hadoop extension that high level data framework , HIVE-QL is a language for HIVE framework. In paper [12] explained about HIVE framework and the importance about HIVE framework , this paper represents about size of data sets that is being collected and analyzed in industry section for the purpose of business intelligence which is growing rapidly, thus leading to the traditional warehousing solutions prohibitively expensive. Now the Hadoop which is the popular open-source map-reduce implementation has been used in companies like Facebook, Yahoo etc. largely for the aim of storing and processing extremely large data sets on commodity hardware. However, the map-reduce programming model is a very low level medium and requires developers to write custom programs that are hard to reuse and maintain. In the paper, we have presented HIVE, as the open-source data warehousing solution mainly built on top of the Hadoop.

Hive supports queries expressed in a SQL-like declarative language - HIVE-QL, which are thus compiled into map-reduce jobs that are executed using Hadoop. In addition, HIVE-QL enables users to plug in custom map-reduce scripts into queries.

The language largely includes a type of system that has support for tables containing primitive types, nested compositions of the same and collections like arrays and maps. The underlying IO libraries can also be extended to query data in custom formats. HIVE basically includes the system catalog called as Metastore, that contains schemas and statistics, that are useful in query compilation, data exploration, query optimization . In Facebook, the HIVE warehouse contains tens of thousands of tables and stores of over 700TB of the data and has been used extensively for both the purpose of reporting and ad-hoc analyses by more than 200 users per month.

The paper [4] explained how to convert our query to HIVE that means how we can convert SPARQL query to HIVE. We prefer HIVE and PIG because Mapreduce is full of code not feasible for developer. In paper [4] explained about querying DBpedia using HIVE-QL and also told some variant of HIVE-QL. One of the main

data hubs on the web nowadays is DBpedia. DBpedia extracts its content from different Wikipedia editions, which dramatically increase day after day according to the participation of new chapters. It became a big data environment describing 38.3 million things with over than 3 billion facts. This data explosion affects both the efficiency and accuracy of the retrieved data. From this point of view, we propose a new architecture to deal with the DBpedia using big data techniques in addition to the Semantic Web principles and technologies. Our proposed architecture introduces HIVE-QL as a query language for DBpedia instead of the SPARQL Query Language, which is considered the backbone of the semantic web applications. Additionally, this paper presents the implementation and evaluation of this architecture that minimizes the retrieval time for a query in DBpedia.

Resource Description Framework (RDF) is used to represent the semantic data and SPARQL is used for semantic web analysis. how to use SPARQL in semantic web was discussed in [link will be enter] developing knowledge in RDF/XML configuration by the usage of the LUBM datacenter and transforming the designed data to NTriplesform using N-Triple Converter was proposed in [12] this shows that a query time rate is less than rate in data size. In last few years Linked Data cloud has expanded rapidly to hold billions of triples. This is because of this large amount of data we required a scalable and dynamic approach. MapReduce computing model has become essential component of current large-scale refining solutions. Google and search engine uses linked data [7] and map reduce technique to obtain huge amount of results. How can we use Hadoop and MapReduce to obtain huge amount of semantic data was proposed in [8]. An algorithm was discussed to convert SPARQL into MapReduce[8]. A structure placed on Hadoop titled “HadoopSparql” was proposed in [9] to hold queries together based on servicing multi-way join operator and how to keep and load data from Hadoop to Resource Description Framework is given in [10]

So finally it is easy to work on PIG or HIVE instead of MapReduce, So our goal is to integrate the our dataset into PIG and HIVE then we execute our query on PIG and HIVE and compare the result. finally we will get our result.

# 3. PROBLEM STATEMENT

Now a days as the total count of users over the internet are increasing day by day, there is also a big issue regarding increase in the size of data produced, which had increased the importance of searching and storing an appropriate entity. In today's to handle the semantic data and represent the semantic data is one amongst the foremost mentioned issues in wide selection of fields of technology like web networks, data processing, data warehousing and data integration . In previous days of semantic data, we represent the semantic data in the tables which have different schema means every table has its own schema just because of different schema data integration between the web networks is quite difficult. So to handle the problem of data integration RDF came in picture .

Now a day's Semantic Web data represented as Resource Description Framework (RDF) is growing rapidly. RDF is a core technology of the semantic for representing data in machine-readable formats. RDF is very useful in semantic web and data integration. RDF is a triple structure format which provides a structure which is in form of the subject–property–object expressions. Means we divide every table with same structure that helps for data integration. But the problem comes with the size because when we divide every table in three columns then size of the table increases exponentially. Sometimes we cannot handle this problem with single cluster machine so we move on parallel processing which should be reliable. For parallel processing we have Hadoop.

RDF has its own language which is SPARQL .It takes lot of time for a single query. So instead of RDF we move on to PIG and HIVE. PIG and HIVE are used to reduce the query processing time. Our task is to define which is better among these three.

## 3.1 Gap Analysis

For semantic data with high density, more execution time required:- All the available platforms, provide the satisfying output with high query execution time

for data, but as the data rises exponentially, i.e. these platforms fails to produce useful output within time for larger dataset .

**Efficiency** :- Almost in every query execution, RDF data are used, which fall to contribute to fair level of efficiency as these provide high query processing time due to presence of triple format data, however this query processing time should be slow down as low as possible?

**Space requirement** :- Also in order to calculate similarity for data having large dimensions, it requires more space.

## 3.2 Objectives

To analyze, evaluate the query processing time for data, PIG and HIVE technique is considered in which query processing time are reduced. To detect and examine the outcomes of the overhead technique, so that time and space requirements can be lowered. The proposed scheme has been successfully implemented and comparative analysis of proposed and standard methodology has been done for our dataset which consist 5000 triple.

## 3.3 Methodology

In this thesis, First I used RDF platform for semantic data . my proposed architecture consist four phase , we have three framework ,one is SPARQL , PIG and HIVE. Our goal is to compare these framework. first load the rdf file and apply SPARQL query then second convert RDF to CSV and CSV to text file. Finally we load the text file into HDFS and execute the query on PIG and HIVE. Here we used PIG for better performance and it is quite efficient than others . Here PIG and HIVE are applied to improve time complexity.

# 4. DESIGN AND IMPLEMENTATION

## 4.1 Architecture of proposed model

The basic modules of the proposed system are described in this chapter of system architecture. Here we want to show that how my proposed model will work and what are the different phases of my architecture. Here we are working with huge dataset with 5000 tuples . Our proposed model work will work on text file , here we are using HIVE-QL and PIG-LATIN instead of SPARQL because we want reduce our query processing time and pre processing time, HIVE and PIG are used because these two provide parallel processing. Actually these two high level framework of Hadoop which transforms user job to MapReduce.

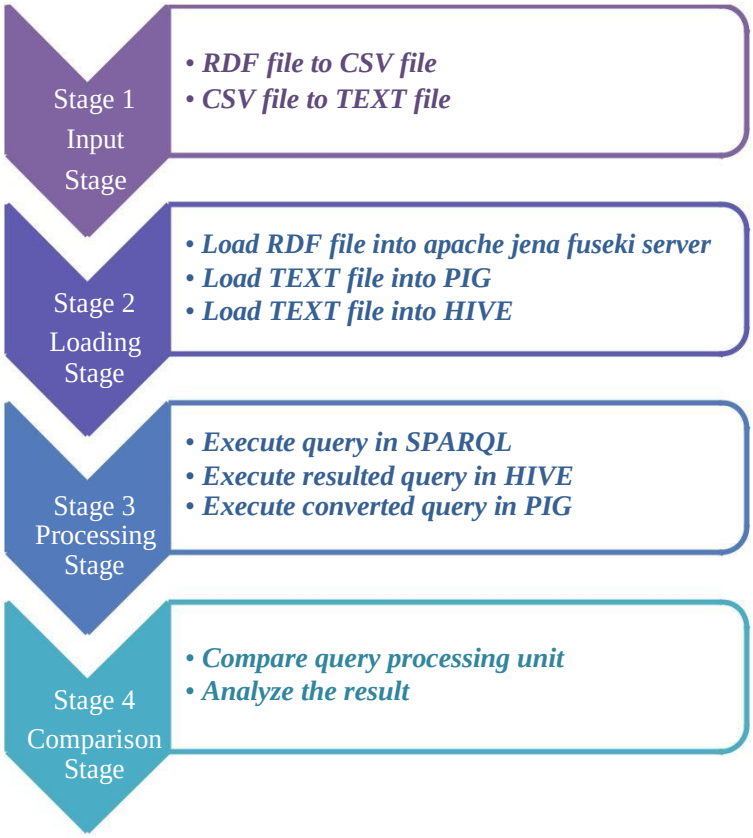


**Figure 4.1 Phases of a Proposed Architecture**

Basically Hadoop has two phases one for storing and other for processing phase , for storing phase we used HDFS and for processing phase we used MapReduce , but here we used PIG and HIVE instead of MapReduce because MapReduce

contains lot of code so we used PIG or HIVE that transforms code to MapReduce . So that approach is quite easier for developer. Our goal is to do comparison among SPARQL, PIG and HIVE so we will find which one is better for our dataset. As told , our dataset consists 5000 triple and we have to do our comparison analysis on this dataset. Actually the format of file is different for different framework . As rdf file for SRARQL and we are using apache jena fuseki server to handle the SPARQL query.

Then we are using text file for PIG and HIVE. For implementation PIG and HIVE we are using cloudera Hadoop . So basically our architecture has three phase first phase is a input phase and second phase is processing phase and last is a output phase.

In our architecture we proposed four phase means input, loading, processing and comparison phase. The proposed architecture components is shown below:

### 4.1.1 Data Input Stage

In this stage we need an input file so we called data input phase. Here we require two types of files. First file is for SPARQL and second file is for HIVE-QL and PIG-LATIN. HIVE and PIG are both are different platform both have different language, different shell but we can use text file for both platform .Either we can use text or CSV file that is up to us .

Here we have used RDF file for SPARQL then we have to convert this file to CSV or text. Then we are converting this RDF file into CSV file and later into text file so that we can use it in HIVE-QL and PIG-LATIN. After this we have two type of files text file and RDF file for PIG or HIVE and SPARQL respectively.

### 4.1.2 Loading Stage

This stage we called loading stage because in this stage we will load files into corresponding server or framework. According to last stage we have two files one is RDF format file and another is text file. So we will load these two files. First we will load RDF file into apache jena fuseki server and then we will load the text file into HIVE and PIG. After loading that phase will complete now we move on to next phase that is processing stage.

### 4.1.3 Processing stage

After loading now turn up to the processing phase. When we have both the files in our system then we will execute our query .We have RDF file in our system which consist 5000 triple then we write the query in jena fuseki server and execute the query. After the execution we will have our result. We have text file in our system then we will execute the converted query in HIVE-QL and at last we will execute the converted query in PIG, here converted query mean the same query written in PIG and HIVE means result should be same. Means after the execution output of the all queries are same but query processing time is different. After that stage we should compare the three platform and so we will say which platform is better. after this stage we will move on to next stage that is final stage is called comparison stage.

### 4.1.4 Comparison stage

In this stage, we will compare the query processing time of each platform. Here we compare the resulting query processing time of each query language, means query processing time in PIG, query processing time in HIVE and query processing time in SPARQL, and finally we will conclude which one is better.

## 4.2 Implementation of proposed model

In this part, we will show how to implement our proposed architecture, in our proposed architecture we showed different stage of our architecture, that was data input phase, loading phase, processing phase and comparison stage. We are presenting four test cases and testing each of them separately on SPARQL, PIG and HIVE. here we are applying each test case on SPARQL, PIG and HIVE. After the result we will compare our four test cases and then comparison will show you query processing time of four test cases. This part will confirm that which query language is fit for our dataset.(the results thus obtained will confirm which query language is best suitable for our dataset). We all know that SPARQL is best language for semantic web, it is quite simple but when the data set get larger in size it lacks with its time (SPARQL take more time for query execution) because of more join operation. SPARQL is very time consuming for larger dataset because

SPARQL is dominated by join operation. For reducing the time we used Hadoop for reducing the query processing time and preprocessing time. So for this we execute our query in Hadoop or we can say MapReduce. For executing the query in MapReduce we use PIG and HIVE. In this section we are applying our test cases on SPARQL, PIG and HIVE and show the result and execution time . Here we have used a dataset with 5000 triples or rows. our dataset has three column and schema is fixed for RDF file as well as text file. RDF file has structure of three column so we convert RDF to CSV file then we convert text file. We are converting rdf to text file because we want to use text file for HIVE and PIG and RDF file we are using for SPARQL. Now in the RDF file and text file we have 5000 triple in both. Then we just execute our query on separately on SPARQL, HIVE and PIG . Now we will finally compare the result among these three and conclude about best one for our dataset.

### 4.2.1 First Case (SPARQL):

Here the first test case and in first query we wish to find out the email-id of a person named 'craig'. So we execute the query in SPARQL first and then in PIG and at last in HIVE as shown in Figure 4.2, Figure 4.4 and Figure 4.6 respectively and results of all three queries same but query processing time is different for each query language. The query has taken 5.34 seconds in SPARQL for 5000 triple dataset.

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?craigEmail
WHERE
{
    ab:craig ab:email ?craigEmail
}
```

**Figure 4.2 Sparql Query1**

In this query we want to select the email id of a person craig so here you can see 'ab:' that is prefix which is used in every SPARQL query. So the next question is

what is the use of prefix . Here we are using prefix because we want reduce the length of our query. here we defined what 'ab:' mean which is shown in figure 4.2 that we used 'ab:' for <http://learningsparql.com/ns/addressbook#> which is a URI or you can say URL. There is little difference between URI and URL. URL is uniform resource locator and URI is the uniform resource identifier. So in this query select clause use for select the column and in where clause it check which record match with this statement so at lat which triple match with this triple it select that column and it will show that column in the result.


**Figure 4.3 Sparql Query1 Result**

This is the result of the SPARQL query of the first test case. So finally we get the email- id of a person craig, in our dataset craig having two email-id which is shown above. that SPARQL execute on our dataset which has 5000 triple. Our next is obtain the same result as above in PIG and HIVE.

### 4.2.2 First Case (PIG):

Here we want to show how we will execute query in PIG. Actually PIG is platform and PIG-LATIN is a language of PIG framework. PIG-LATIN has its own syntax and rules. So here the first rule, we should load our dataset file into HDFS before execute the query. HDFS is a Hadoop distribution file system, that is provided by Hadoop for storing purpose. First we should load our dataset file into HDFS, actually PIG worked in to two mode first one is local mode and other one is MapReduce mode. In local mode PIG load file into its local directory and In MapReduce mode PIG load the file into HDFS. So here we are using MapReduce mode so PIG can take file form HDFS. First we load the file into HDFS then we execute our query into PIG and find the result accordingly .So here the query which is shown below.

```
1  emp = LOAD '/tmp/cloudera/mydata.txt' using pigstorage(',')
2  as (name:chararray, info:chararray, comm:chararray);
3  data = FILTER emp by name == 'craig' AND info == 'mail';
4  dump data;
```

**Figure 4.4 Pig Query1**

So above figures 4.4 is shown PIG query. This query contains four statement. First statement is the load statement which load the file from HDFS . We should give the path where our file is located. In our query it is loacted in tmp directory which is in HDFS and we want to load my datset which is text file contains 5000 triple and which name is mydata.txt located in HDFS. After loading the file we shold define its schema means how many column and types of column.

Here the type of all three column is chararray means it contains character. After we defined schema, we stored our datset file into table emp, then we used filter command for selecting the person name craig which has info as mail, means we want to find email id of craig.

After the filtering we stored our table into table data. Finally dump command is used for display purpose. Finally we shown result below which is same as SPARQL query result.

```
1  RESULT:
2
3  craigellis@yahoo.com
4  c.ellis@usairwaysgroup.com
```

**Figure 4.5 Pig Query1 Result**

In the above figure one can see the result of PIG query. the result is same as SPARQL query. The difference between these two with respect to time SPARQL

takes more time then PIG. At last we will execute our query in HIVE with the same result.

### 4.2.3 First Case (HIVE):

The working of HIVE framework is same as PIG both transform job into MapReduce , but both have different languages and different syntax. HIVE is quite similar to SQL so it is easier then PIG. In HIVE, first statement is load statement that load the your dataset file from HDFS then you defined schema and execute query. Here the query is shown below in figure.

```
1 CREATE TABLE IF NOT EXITS employee(name String, info String, comm String );
2 LOAD DATA LOCAL INPATH '/tmp/cloudera/mydata.txt'
3 overwrite into table employee;
4 select comm from employee
5 where name = 'craig' and info= 'mail';
```

**Figure 4.6 Hive Query1**

Here our query is shown in above figure 4.6. It has five statements. First statement told about the schema of a table according to your dataset file. The second statement load the dataset file form HDFS in which you should mention the data path means where the file was located. First thing is very important that your dataset structure and table schema should be same. after loading the file you should overwrite your loaded file into table employee. Then we can apply your operation according to your result. Here we want to find the email-id of the person craig. Here the result is shown in figure 4.7 below.

```
1 RESULT:
2
3 craigellis@yahoo.com
4 c.ellis@usairwaysgroup.com
```

**Figure 4.7 Hive Query1 Result**

So finally we performed our operation into three language which has same result but different query processing time .

So SPARQL takes more time than others because of many join operation. SPARQL is dominated by join operation so that's why it took more time than PIG and HIVE.

### 4.2.4 Second Case (SPARQL):

Here we want to execute our query into three different platform that is SPARQL, PIG and HIVE. Here we are going to little bit depth , means we perform some other interesting operation . In this query the goal is to obtain the count of record given by previous query. Here we used group by clause in all of above language to observe which one is more efficient for that clause. We have used group by which is more often used in semantic web. Here we are using 'group by' clause because we are going to be more specific that which operation takes more time. So the query is shown below in the figure 4.8.

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT(COUNT(?craigEmail) as ?mail)
WHERE
{
    ab:craig ab:email ?craigEmail
}
    group by ?craigEmail
```

**Figure 4.8 Sparql Query2**

The query is shown above, here we are introducing group by clause. First we add our RDF file into jena fuseki server which consist 5000 triple then we add prefix 'ab:' into our query to reduce the length of a query and making easy to understand . Here we want to count the number of email-id by the person craig.

Here we used group by clause and aggregate operator count which count the number the number of email-id. The syntax is quite similar to SQL. Group by clause takes more join operations than the normal query.


**Figure 4.9 Sparql Query1 Result**

The result is shown in above, here the result is 2 because the person name craig has email-id. the query executed in jena fuseki server. Now we will execute our query which lead to same result as above on to some other platform that is PIG and HIVE.

### 4.2.5 Second Case (PIG):

Here we are introducing query in PIG. Query structure is quite same as previous query means in terms of loading and schema. The query will change in terms of group by operation, first we load our dataset file into HDFS then we can transform our text file to table with proper schema means same schema as table then we perform our operation on dataset. The difference between previous query and this query is that this query used group by clause and count the number of email-id of a person craig , this query takes more time than previous query just because of group by clause. Here the query is shown below.

```
1 emp = load '/tmp/cloudera/mydata.txt' using pigstorage(',')
2 as (name:chararray, info:chararray, comm:chararray);
3 emp1 = group emp by comm;
4 data = FILTER emp1 by name == 'craig' and info == 'mail';
5 dump data;
```

**Figure 4.10 Pig Query2**

In the above figure contains 5 statements. First statement as usual load statement which is load the data file into HDFS and then define schema same as data set file

and one thing is important then one should mention the data path in our query, means we should know about dataset file where it is located. After loading one can split our table by group by clause. Then perform filtering operation and finally stored into a table called data. At last for display one can use dump command and finally we got same result as SPARQL.

```
1  QUERY RESULT:
2
3  MAIL
4  2
```

**Figure 4.11 Pig Query2 Result**

Finally we got result same as SPARQL query. We got 2 as a result because a person craig have 2 email-id in our dataset. Result is same but query processing time is different. PIG took less time than SPARQL. Now we are going to last framework that is HIVE. HIVE query easy than PIG because it is follow SQL structure. Now we are going to show HIVE query .

### 4.2.6 Second Case (HIVE):

```
1  CREATE TABLE IF NOT EXITS employee(name string, info string, comm string)
2  ROW FORMAT DELIMITED
3  FIELDS TERMINATED BY '\t'
4  LINES TERMINATED BY '\n'
5  STORED AS TEXTFILE;
6  LOAD DATA LOCAL INPATH '/tmp/cloudera/mydata.txt'
7  OVERWRITE INTO TABLE EMPLOYEE;
8  select count(comm) from employee
9  where name = 'craig' and info = 'mail'
10 group by comm;
```

**Figure 4.12 Hive Query2**

Here we are going to show HIVE query structure, query is quite same as the previous query because group by clause some part is different. Loading phase is same as previous means one should load the dataset file from HDFS then apply our query operation.

In the above figure 4.12 first we defined our table schema means defined column and its type and one should mention fields delimiter and lines delimiter which is '\t' and '\n' respectively then one can load our dataset file from HDFS, path should be known to us. After loading the file we overwrite your file into table employee which is created by you. Then we used aggregate operator count with select clause. One thing is important that if we used count operator with column 'comm' then we should use group by operator with the same column that is used with aggregate operator count. Then we applied condition which is name = "craig" and info = "mail", we used and operator means both condition should be true to satisfy this condition. After the execution of this query you will get result 2 which is same as SPARQL and PIG query. Actually result is same in all three query means SPARQL, PIG and HIVE got same result but query processing time is different in all three. So we can conclude which query language is better for our dataset on the basis of query processing time.

# 5. RESULTS AND ANALYSIS

In this chapter we will discuss comparative analysis of SPARQL, PIG and HIVE. In the previous chapter we showed the query execution of all three framework and here we will compare time of every framework. So here we will discuss step by step query execution of the test cases.

## 5.1 First Test Case

In the first test case we executed our query in three different platform. So here we present a graph which represent the query processing time in these three platforms. The result is shown below .

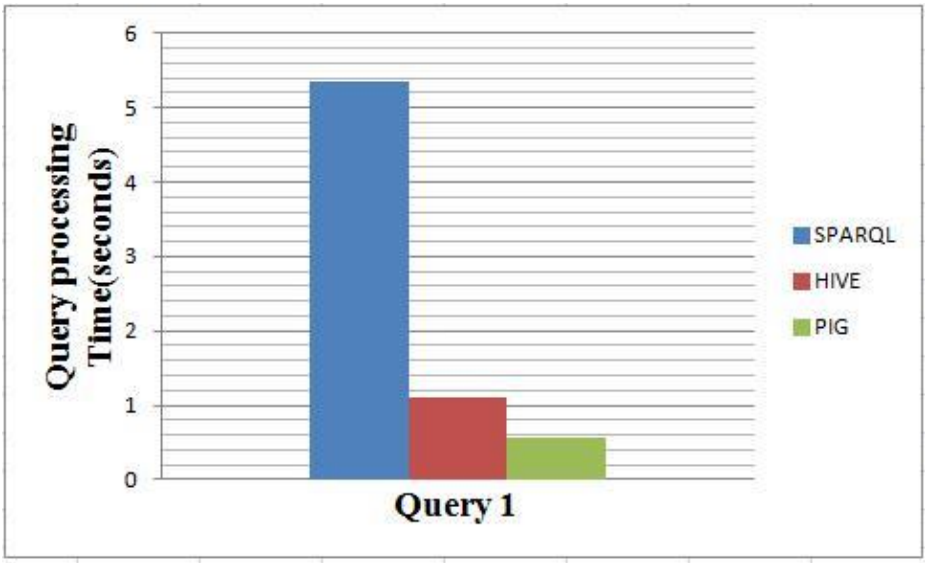


**Figure 5.1 Comparison Analysis of SPARQL, PIG and HIVE for query1**

In this graph we can see which query language is working good for our dataset. According to graph PIG perform better than other two in terms of time. In the first test case PIG takes 0.57 second , hive takes 1.1 second and SPARQL takes 5.34 second . So we can say PIG is better than HIVE and SPARQL

## 5.2 Second Case

In the second case we executed query in PIG, HIVE and SPARQL. SPARQL takes more time than PIG and HIVE because SPARQL has many join operation. So finally we represent a graph which is shown below.


**Figure 5.2 Comparison Analysis of SPARQL, PIG and HIVE for query2**

In this graph we can see that PIG takes less time than HIVE and SPARQL. In the second case we introduce group by clause so here SPARQL takes more time than the previous query because join operation increased. Actually PIG takes 0.74 second and HIVE takes 1.4 second and SPARQL takes 7.5 second. So finally we conclude that PIG is better than others.

## 5.3 Final analysis

Here we want to show complete result of both test cases. In first test case we used simple without any clause, here our analysis is based on 5000 triple dataset. First test case is quite simple that basically simple query without any special operator but In the second test case we used aggregate operator with group by clause. So our complete analysis is based on both test cases.

So in below figure 5.3 you can see directly which query language is better for our dataset. Here we can easily see, time taken by PIG is less than others. In our both the test cases PIG has performed better than HIVE and SPARQL. In first test case SPARQL takes more time than others, but in second test case SPARQL takes more time than query processing time in first case by SPARQL . In both the test cases PIG take less time than others.

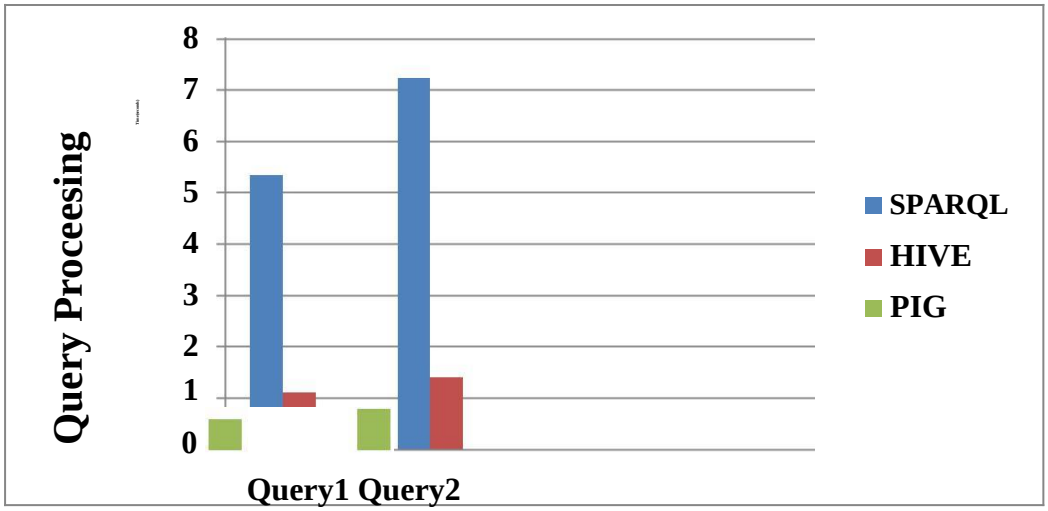


**Figure 5.3 Comparison Analysis of SPARQL, PIG and HIVE**

## 5.4 Architecture Evaluation

Our proposed architecture is presented in the form of a layered architecture as shown in Fig. We presented our proposed architecture via some criteria described in previous chapter as shown in Table 1; those criteria are as follows:

- Availability: The degree of a system to be operable and in a committable state at the start of a mission.
- Clearly defined context: The possibility to identify the context from the description of the architecture.
- Appropriate level of abstraction: The possibility to view the system within the framework as a whole
- Hiding of implementation details: The possibility to hide any implementation detailed information in the description of the architecture

- Clearly defined functional layers: The possibility to specify a function in layer description within the system?
- Interoperability: The extent to which systems can exchange information
- Modularity: The possibility to change the implementation of a particular layer while the functionality and the interfaces remain the same
- Upgradeability: The degree to which a system can easily be improved in functionality.
- Modifiability: The extent to which a system can be modifiable.
- Accessibility: The degree to which many people can access a system as possible.
- Usability: The measure of how easy it is to use the system.
- Stability: The extent to which the system rarely exhibits failure.
- Efficiency: The degree to which the system is running in an efficient time and generates correct output

**Table 1. Architecture Evaluation**

| Sr No. | Criterion | Proposed Architecture |
|---|---|---|
| 1. | Availability | Yes |
| 2. | Clearly defined context | Yes |
| 3. | Hiding of implementation details | Yes |
| 4. | Abstraction | No |
| 5. | Clearly defined functional layers | Yes |
| 6. | Interoperability | Yes |
| 7. | Modularity | Yes |
| 8. | Upgradability | Yes |
| 9. | Modifiability | Yes |
| 10. | Accessibility | Yes |
| 11. | Usability | Partially |
| 12. | Stability | Partially |
| 13. | Efficiency | High |

# 6. CONCLUSION AND FUTURE SCOPE

This thesis presents a system for efficient storage and analysis of RDF data using apache HIVE and PIG. HDFS for this thesis, provides storage of RDF data. As described in previous chapter, graph showed the best results compared with other storage schemas for RDF documents. In this schema, each property from the RDF document corresponds to a column in the table. This work is done to propose a new method that is capable of efficiently querying huge amounts of semantic data content on Hadoop environment based on HIVE or PIG. This thesis discusses our proposed method and shows the implementation of this method based on SPARQL, HIVE-QL query and PIG-LATIN.

During the evaluation of PIG and HIVE on a computer cluster, Results shows query processing time in SPARQL, HIVE and PIG. In the result we have executed the query in SPARQL, PIG and HIVE the following results can be summarized:

- PIG shows best performance for selective lookup queries. Hereby, nearly constant runtimes independent from the respective dataset size can be achieved.
- Unselective queries can lead to longer runtimes, which can no longer be considered as interactive runtimes. In the worst case, these queries can lead to out-of-memory-errors,
- Partitioning did not prove beneficiary for the runtimes.
- PIG perform better than HIVE and SPARQL in terms of time but for storage and other respect they are same but PIG is quite efficient for larger database.
- On average, PIG performs one order of magnitude faster compared with Apache HIVE and SPARQL.
- PIG is 35% faster than HIVE for arithmetic operation and 105% faster than SPARQL for arithmetic operation .
- PIG is 45% faster than HIVE for filtering operation and 95% faster than SPARQL for filtering operation .

- PIG is 30% faster than HIVE for joining operation and 85% faster than SPARQL for joining operation.

As soon as PIG supports nested data types, future work could refine the property table storage schema for storing multi-value properties more efficiently without the overhead of duplicate rows. Additionally, the evaluation of SP2 Bench showed that the translation was not optimal for complex queries and therefore leaves room for specific query optimization. Related software products show a hybrid of in-memory and on-disk join evaluation. This feature is planned for Impala and may overcome the limitation of Impala, which became visible in form of out-of-memory errors during the evaluation. In future works, Impala 2.0 could be compared with other hybrid system. All in all, Sempala provides an efficient complement to MapReduce-based approaches for selective queries at the present time. The results show how our proposed method outperforms using SPARQL in retrieving the data for future task, use Apache Spark and Impala as an attempt to get better results to the system and to get better Efficiency and Accessibility.

# REFERENCES


[1] Quilitz, B., & Leser, U. 2008. Querying distributed RDF data sources with SPARQL. Springer Berlin Heidelberg. 524-538.

[2] Kim, H., Ravindra, P.,and Anyanwu, K. 2011. From sparql to mapreduce: The journey using a nested triplegroup algebra. Proc. VLDB Endow, 4(12), 14261429.

[3] Ravindra, P., Kim, H., and Anyanwu, K. 2011. An intermediate algebra for optimizing RDF graph pattern matching on MapReduce. In The Semantic Web: Research and Applications, Springer Berlin Heidelberg.46-61.

[4] Ismail, A. S., Al-Feel, H. A. Y. T. H. A. M., and Mokhtar, H. M. Querying DBpedia Using HIVE-QL.

[5] Ravindra, P., Hong, S., Kim, H., and Anyanwu, K. 2011. Efficient processing of RDF graph pattern matching on MapReduce platforms. In Proceedings of the second international workshop on Data intensive computing in the clouds , ACM.13-20.

[6] Arias, M., Fernández, J. D., Martínez-Prieto, M. A., and de la Fuente, P. 2011. An empirical study of real-world SPARQL queries.

[7] Anyanwu, K., Kim, H., and Ravindra, P. (2013). Algebraic optimization for processing graph pattern queries in the cloud. Internet Computing, IEEE,17(2), 52-61.

[8] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N.,and Bizer, C. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web, 6(2), 167-195.

[9] Ravindra, P., Kim, H., and Anyanwu, K. Optimization of Complex SPARQL Analytical Queries.

[10] Schätzle, A., Przyjaciel-Zablocki, M., Neu, A., and Lausen, G. 2014. Sempala: Interactive SPARQL query processing on hadoop. In The Semantic Web– ISWC 2014, Springer International Publishing.164-179.

[11] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., and Murthy, R. 2009. Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, 2(2), 1626-1629.

[12] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N.,and Murthy, R. 2010. Hive-a petabyte scale data warehouse using hadoop. In Data Engineering (ICDE), 2010 IEEE 26th International Conference on IEEE.9961005.

[13] Schätzle, A., Przyjaciel-Zablocki, M., and Lausen, G. 2011. PigSPARQL: Mapping SPARQL to Pig Latin. In Proceedings of the International Workshop on Semantic Web Information Management, ACM.

[14] N. Shadbolt, T. Berners-Lee and W. Hall, 'The Semantic Web Revisited', IEEE Intell. Syst., vol. 21, no. 3, pp. 96-101, 2006.

[15] Getting Started with Hadoop', 2015. [Online]. Available: http://hortonworks.com/get-started/. [Accessed: 24- Jun- 2015].

[16] Introduction to Hadoop. [Online]. Available: http://hadoop.apache.org/. [Accessed: 24- Jun- 2015].

[17] B. Enrico, G. Marco and I. Mauro, 'Modeling apache hive based applications in big data architectures', in the 7th International Conference on Performance Evaluation Methodologies and Tools, 2015, pp. 3038.

[18] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann, 'DBpedia - A crystallization point for the Web of Data', Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, no. 3, pp. 154-165, 2009.

[19] E. Dumbill, "Big data and the semantic web at war, Indifferent, or Intimately Connected," in Strata Conference New York 2011, 2011.

[20] H. Christian Bizer and B. Christian, 'special issue on Linked Data', International Journal on Semantic Web and Information, 2014.

[21] H. Mohammad Farhan, K. Latifur, K. Murat and H. Kevin, Data intensive query processing for Semantic Web data using Hadoop and MapReduce. IEEE 3rd International Conference. The University Of Texas at Dallas, 2011

[22] L. Chang, Q. Jun, Q. Guilin, W. Haofen and Y. Yong, 'Hadoopsparql: a hadoop-based engine for multiple SPARQL query answering', in 9th Extended Semantic Web Conference, 2012, pp. 474-479.

[23] J. Dean and S. Ghemawat, 'MapReduce', Communications of the ACM, vol. 51, no. 1, p. 107, 2008. [11] H. Mohammad Farhan, D. Pankil, K. Latifur and

T. Bhavani, 'Storage and retrieval of large RDF graph using hadoop and MapReduce',in Springer, Berlin Heidelberg, 2009, pp. 680-686.

[24] N. Tu Ngoc and S. Wolf, 'SLUBM: An Extended LUBM Benchmark for Stream Reasoning.', inOrdRing@ ISWC, 2013, pp. 43-54. [13] D. Jin-Hang, W. Hao-Fen, N. Yuan, and Y. Yong, 'HadoopRDF: A scalable semantic data analytical engine', in Intelligent Computing Theories and Applications, 2012, pp. 633-641.

[25] H. Albert and P. Lynette, 'Distributed RDF triplestore using HBase and Hive', the University of Texas at Austin, 2012. [15] L. Jens, I. Robert, J. Max, J. Anja and K. Dimitris, 'DBpedia-a large-scale, multilingual knowledge base extracted from Wikipedia', Semantic Web Journal, vol. 5, pp. 1-29, 2014.

[26] H. Al-Feel, 'A Step towards the Arabic DBpedia', International Journal of Computer Applications, vol. 80, no. 3, pp. 27-33, 2013. [17] M. Morsey, J. Lehmann, S. Auer, C. Stadler and S. Hellmann, 'DBpedia and the live extraction of structured data from Wikipedia', Program: electronic library and information systems, vol. 46, no. 2, pp. 157-181, 2012.

[27] [18] T. Carl Lagoze (2008, The Open Archives Initiative Protocol for Metadata Harvesting', 2015. [Online]. Available: http://www.openarchives.org/OAI/2.0/openarchivesp rotocol.2008-12-07.htm. [Accessed: 15- Jun- 2015]. [19] The DBpedia Data Provision Architecture, 2015. [Online]. Available: http://wiki.dbpedia.org/about/aboutdbpedia/architecture. [Accessed: 22- Jun- 2015]

[28] Hive Wiki,2015[Online]. Available:http://www.apache .org/hadoop/hive . [Accessed: 22- Jun- 2015] [21] Introduction to HDFS: what is the Hadoop Distributed File System (HDFS),2015[Online]. Available: http://www.1.ibm.com/software/data/infosphere/had oop/hdfs/ [Accessed: 22- Jun- 2015]

[29] F. Haytham Tawfeek and K. Mohamed Helmy, 'OCSS: Ontology cloud storage system', 2011, pp. 913.

[30] K. Mohamed Helmy and F. Haytham Tawfeek, 'DOCSS: Distributed Ontology cloud storage system ', 2012, pp. 48-52.

[31] H. Neil B and A. Paris, 'Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues-An Exploratory Study', in Transactions on Pattern Languages of Programming III, 2013, pp. 168-194.

[32] G. Aurona J, B. Andries and V. Alta J, 'Towards a semantic web layered architecture', in the International Conference on Software Engineering Innsbruck, Austria, 2007, pp. 353-362.

[33] G. Aurona J, B. Andries and V. Alta J, 'Design and evaluation criteria for layered architectures', in the 8th International Conference on Enterprise Information System, Paphos, Cyprus., 2006.

[34] M. Odersky. (2014), the Scala Language Specification.,2015[Online].Available: http://www.scalalang.org/docu/files/ScalaReference.pdf [Accessed: 1- Jul- 2015]

[35] H. Al-Feel, M. Koutb and H. Suoror, 'Toward An Agreement on Semantic Web Architecture', Europe, vol. 49, no. 3, pp. 806-810, 2009.

# List of Publications

Accepted research paper titled " **Comparison and Analysis of RDF data using SPARQL,HIVE,PIG in Hadoop** " in Springer, " 2016 ICT4SD International Conference on ICT for Sustainable Development " , GOA.

# Video Presentation

https://www.youtube.com/channel/UC3Khx5hGz-0gYWvAO07Tc7A