International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

QUERY PERFORMANCE FOR LARGE RDF GRAPHS USING CLOUD COMPUTING

Anusha P A
Student
PDA College of Engineering, Gulbarga
Anusha.Alandkar.11@gmail.com

Prerana Malu
Student
PDA College of Engineering, Gulbarga
Prerana.malu@gmail.com

Dohra yasmeen
Student
PDA College of Engineering, Gulbarga
Yasmeen37@gmail.com

Pooja M Bhat
Student
PDA College of Engineering, Gulbarga
Pooja.bhat99@gmail.com

Prof.Padmapriya.Patil
Professor & Guide
PDA College of Engineering, Gulbarga
padmazapur@yahoo.co.in

Abstract: **Semantic web is an emerging area to augmented human reasoning. Semantic web technologies are being developed to present data in standardized way. And one such standard is the Resource Description Framework (RDF). Semantic web technologies can be utilized to build efficient and scalable systems for Cloud Computing. This possess significant challenges for the storage and retrieval of RDF graphs. Current frameworks do not scale for large RDF graphs and as a result do not address these challenges. In this paper, we describe a framework that we built using Hadoop to store and retrieve large numbers of RDF triples by exploiting the cloud computing paradigm. We describe a scheme to store RDF data in Hadoop Distributed File System. To determine the jobs, we present an algorithm to generate query plan, whose worst case cost is bounded, based on a greedy approach to answer a SPARQL Protocol and RDF Query Language(SPARQL) query. We use Hadoop's MapReduce framework to answer the queries**.

***Key words*—Hadoop, RDF, SPARQL, MapReduce.**

1 INTRODUCTION

Semantic Web technologies are being developed to present data in standardized way such that such data can be retrieved and understood by both human and machine. Historically, web pages are published in plain html files which are not suitable for reasoning. Researchers are developing Semantic Web technologies that have been standardized to address such inadequacies. The most prominent standards are Resource Description Framework (RDF) and SPARQL Protocol and RDF Query Language (SPARQL). RDF is the standard for storing and representing data and SPARQL is a query language to retrieve data from an RDF store. Cloud Computing systems can utilize the power of these Semantic Web technologies to provide the user with capability to efficiently store and retrieve data for data intensive applications. Semantic web technologies could be especially useful for maintaining data in the cloud. Semantic web technologies provide the ability to specify and query heterogenous data in a standardized manner.

PROBLEM DEFINITION: At present, there are few frameworks(e.g. RDF-3X, Jena, BigOWLIM)for Semantic Web technologies, and these frameworks have limitations for large RDF graphs. Therefore, storing a large number of RDF triples and efficiently querying them is a challenging and important problem. A distributed system can be built to overcome the scalability and performance problems of current Semantic Web frameworks. Databases are being distributed in order to provide such scalable solutions.

2. RELATED WORK

Researchers and enterprises are using MapReduce technology for web indexing, searches and data mining. In this section, we will first investigate research related to MapReduce. Next, we will discuss works related to the semantic web.Google uses MapReduce for web indexing, data storage and social networking. Yahoo! uses MapReduce extensively in their data analysis tasks. IBM has successfully experimented with a scale-up scale-out search framework using MapReduce technology. In a recent work, they have reported how they integrated Hadoop and System. Teradata did a similar work by integrating Hadoop with a parallel DBMS. Researchers have used MapReduce to scale up classifiers for mining petabytes of data. They have worked on data distribution and partitioning for data mining, and have applied three data mining algorithms to test the performance. Data mining algorithms are being rewritten in different forms to take advantage of MapReduce technology. In researchers rewrite well known machine learning algorithms to take advantage of multicore machines by leveraging MapReduce programming paradigm. Another area where this technology is successfully being used is simulation. In researchers reported an interesting idea of combining MapReduce with existing relational database techniques. These works differ from our research in that we use MapReduce for semantic web technologies. Our focus is on developing a scalable solution for storing RDF data and retrieving them by SPARQL queries. MapReduce technology is becoming increasingly popular in

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

the community which handles large amounts of data. It is the most promising technology to solve the performance issues researchers are facing in Cloud Computing.

The work described in this paper differs in the following ways: first, we have queried 1 billion triples. Second, a storage schema is devised which is tailored to improve query execution performance for RDF data. RDF triples is stored in files based on the predicate of the triple and the type of the object. Finally, an algorithm is described to determine a query processing plan. By using this, the input files of a job are determined and the order in which they should be run.

Jena - is a semantic web framework for Jena. True to its framework design, it allows integration of multiple solutions for persistence. However, Jena is limited to a triple store schema. Jena have very poor query performance for large datasets. Furthermore, any change to the dataset requires complete recalculation of the inferred triples.

BigOWLIM - is among the fastest and most scalable semantic web frameworks available. However, it is not as scalable as our framework and requires very high end and costly machines. It requires expensive hardware (a lot of main memory) to load large datasets and it has a long loading time. As our experiments show it does not perform well when there is no bound object in a query. However, the performance of our framework is not affected in such a case.

RDF-3X - is considered the fastest existing semantic web repository. In other words, it has the fastest query times. RDF-3X uses histograms, summary statistics, and query optimization to enable high performance semantic web queries. However, RDF-3X's performance degrades exponentially for unbound queries, and queries with even simple joins if the selectivity factor is low. This becomes increasingly relevant for inference queries, which generally require unions of subqueries with unbound objects.

3. PROPOSED METHODOLOGY:

The architecture consists of two components. The upper part of Figure depicts the data preprocessing component and the lower part shows the query answering one. There are three subcomponents for data generation and preprocessing. RDF/XML is converted to N-Triples serialization format using our N-Triples Converter component. The PS component takes the N-Triples data and splits it into predicate files. The predicate files are then fed into the POS component which splits the predicate files into smaller files based on the type of objects. The MapReduce framework has three subcomponents in it. It takes the SPARQL query from the user and passes it to the Input Selector and Plan Generator. This component selects the input files, by using our algorithm , decides how many MapReduce jobs are needed and passes the information to the Join Executer component which runs the jobs using MapReduce framework. . It then relays the query answer from Hadoop to the user.

3.1 Data Generation and Storage:

For our experiments, the LUBM dataset is used. The LUBM data generator generates data in RDF/XML serialization format. This format is not suitable for our purpose because we store data in HDFS as flat files and so to retrieve even a single triple we would need to parse the entire file. Therefore the data is converted to N-Triples to store the data, because with that format we have a complete RDF triple (Subject, Predicate and Object) in one line of a file, which is very convenient to use with MapReduce jobs. The processing steps to go through to get the data into our intended format are described in following sections.

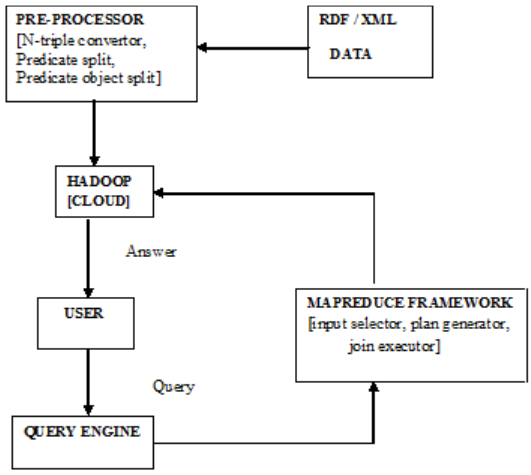


Figure 3: the proposed architecture

3.2 File Organization

We do not store the data in a single file because, in Hadoop and MapReduce Framework, a file is the smallest unit of input to a MapReduce job and, in the absence of caching, a file is always read from the disk. If we have all the data in one file, the whole file will be input to jobs for each query. Instead, we divide the data into multiple smaller files. The splitting is done in two steps which we discuss in the following sections.

3.3 Predicate Split (PS): In the first step, the data is divided according to the predicates. This division immediately enables to cut down the search space for any SPARQL query which does not have a variable predicate.

3.4 Predicate Object Split (POS): In the next step, the data is divided depending upon the object type. **S**o a single file is now converted to numerous files so that the data can be retrieved easily.

4. MAPREDUCE FRAMEWORK AND QUERY PLAN GENERATION:

In this section, we discuss how to answer SPARQL queries in our MapReduce framework component. Section 4.1 discusses the algorithm to select input files for answering the query. Section 4.2 represents the query plan generation.

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

Section 4.3 presents our heuristics based algorithm to generate a query plan.

4.1 Input Files Selection

1. Query 1

```
SELECT ?P ?Q ?R WHERE { ?P rdf : course_type.
?Q rdf :course_name .
?R rdf :course_stream . }
```

Before determining the jobs, we select the files that need to be inputted to the jobs. We have some query rewriting capability which we apply at this step of query processing. We take the query submitted by the user and iterate over the triple patterns.

2. Rewritten Query 1

```
SELECT ?P ?Q ?R WHERE {
?P rdf : course .}
```

4.2 Query plan generation

In this section, first we define the query plan generation problem. Then, we will present a heuristic algorithm to generate an approximate solution to answer the query.

Running example: We will use the following query as a running example in this section.

```
SELECT ?N, ?P, ?Q, ?R WHERE{
?P rdf : course_type.
?Q rdf : course_name.
?R ?N rdf : course_stream.
?P rdf : course_entry ?R.
?P rdf : course_entry ?Q. }
```

In order to simplify the notations, we will only refer to the TPs by the variable in that pattern. For example, the first TP (?P rdf : course_type) will be represented as simply P. Also, in the simplified version, the whole query would be represented as follows: {P,Q,R,PR,PQ}. We shall use the notation join(PQ,P) to denote a join operation between the two TPs PQ and P on the common variable P.

4.3 Algorithm :

Step1: Start
Step2: A = Remove non-joining variables
Step3: generation of job plans
while A ≠0 **do**
Step4 : Total number of jobs
$j = 1$
Step5: Sort all the variables in A according to number of joining variables
E={e1,……,ek}
Step6: List of join operations
Job(j)=0
temp = 0
Step7:Elimination of the variables
for *i*=1 to *K* **do**
if Can-Eliminate(A,*ei*)=true **then**
temp = temp + Join(TP(A,*ei*))
A = A - TP(A,*ei*)
job(j) = *job(j)*+ join(TP(A,*ei*))
Step8: **end if**
Step9: **end for**
Step10: A = A + temp
Step11: $j = j + 1$
Step12: **end while**
Step13: **return** join operations
Step14:Stop

Description of Algorithm :. The algorithm starts by removing all the non-joining variables from the query A. In our running example, A=*{*P,Q,NR,PQ,PR*}*, and removing the non-joining variable N makes Q=*{*P,Q,R,PQ,PR*}*. In the while loop, the job plan is generated, starting from *job(*1). In step5, we sort the variables according to the number of joining variables in the resultant triple pattern after a complete elimination of variable. The sorted variables are: *E = {Q,R,P}*, since Q, and R have joining variables= 1, and P have joining variables=2. For each job, the list of join operations are stored in the variable job_j ,. And a temporary variable temp is used to store the resultant triples of the joins to be performed in the current job. In the for loop, each variable is checked to see if the variable can be completely or partially eliminated. If yes, we store the join result in the temporary variable , update A and add this join to the current job. In our running example, this results in the following operations. Iteration 1 of the for loop:

e_1 (= Q) can be completely eliminated.

Here TP(A,Q) = the triple patterns in A containing Q = *{*Q,PQ*}*. Join-result(TP(A,Q)) = Join result (*{*Q,PQ*}*) =resultant triple after the join(Q,PQ) = P. So, temp = *{*P*}*. A = A - TP(A,Q)

= *{*P,Q,R,QP,PR*}* - *{*Q,PQ*}* = *{*P,R,PR*}*.

Job(1) = *{*join(Q,PQ)*}*.

5. CASE STUDY 1:

Flow of execution:

1. design of GUI
2. creation of database
3. linking the database and GUI
4. execution of normal queries
5. result analysis

1.The designing of the framework is done using ASP.NET with Visual C#. Here the case study deals with course entry and course search(course type, course name, course stream, subject name).The design includes two modules namely administration (admin) and user. Admin handles course entry details and user deals with course search. The implementation module is as follows

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

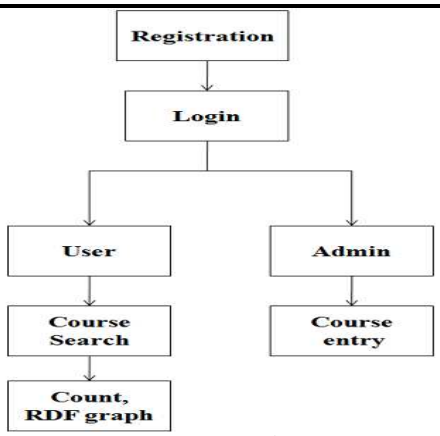


Figure 5.1: design of GUI

2. Creation of tables : Here we create tables like login, register, course_ entry and course_search. For example: course_entry table

Course_type	Course_name	Course_stream	Subject_name
UG	B.E	EC	HDL
UG	B.E	CS	OOPS
PG	M.TECH	EC	VLSI

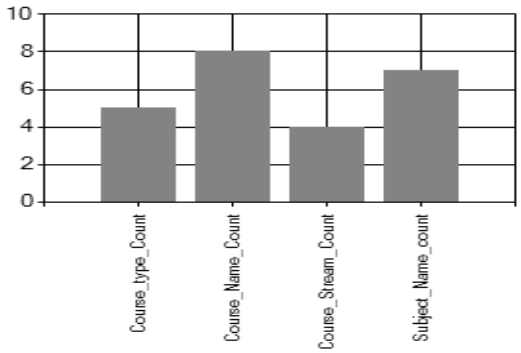
Table 1:Course_entry table

3. Linking database and GUI : Here the code is written to retrieve the data from SQL Server through GUI, using the SQL commands like select, update, delete and insert.
4. Execution of normal SQL Query : For example
Select * from course_entry where course_type='UG', course_name='B.E', course_stream ='EC'
So the resultant table is

Course_type	Course_name	Course_stream	Subject_name
UG	B.E	EC	HDL

Table 2: Query result

5. RESULT ANALYSIS:



Figure

5.2: Result Analysis
The graph indicated in the above figure are the normal query results of case study. Here the count for each query is determined and the time taken can be shown in the graph. The result shown is for smaller datasets and extend this case study for larger datasets.

6.CASE STUDY 2:

In this case study, an RDF graph is generated for query given by the user. A Graphical User Interface (GUI) is designed for the below block diagram shown. Firstly, user sends a query which is searched in hadoop cluster(cloud)/server then if it is found , the file is downloaded and converted to RDF/ Xml format which is passed on to SQL to SPARQL translation and then finally the RDF graph is obtained which display the time taken to answer the queries.

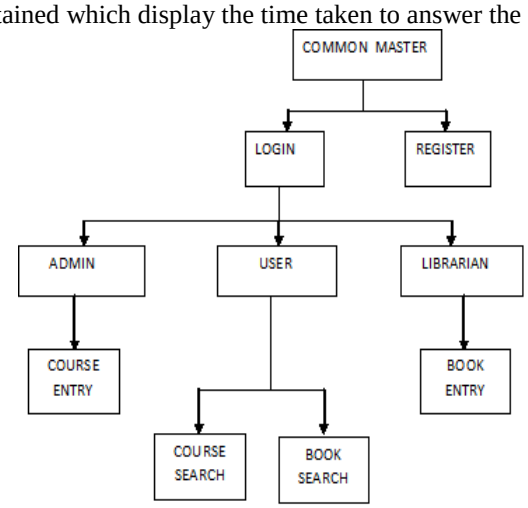


Figure 6.1: design of GUI

The RDF/XML format of the course entered will be created in Root Folder. For example,
Input : id=1

```
coursetype=UG
Durcourse=4
coursename=BE
coursestream=EC
subjectname=VLSI
```

Output :

```
<NewDataSet>
  <Table>
    <id>1</id>
    <coursetype>UG</coursetype>
    <durcourse>4</durcourse>
    <coursename>BE</coursename>
    <coursestream>EC</coursestream>
    <subjectname>VLSI</subjectname>
  </Table>
</NewDataSet>
```

SPARQL Query:
1.SELECT ?: course_type where
{ ? rdf: course.}

2. Select ?:course_stream where
{ ? rdf : course.}
3. Select ?:course_stream where
{ ? rdf : course.}

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

RESULT OF CASE STUDY 2:

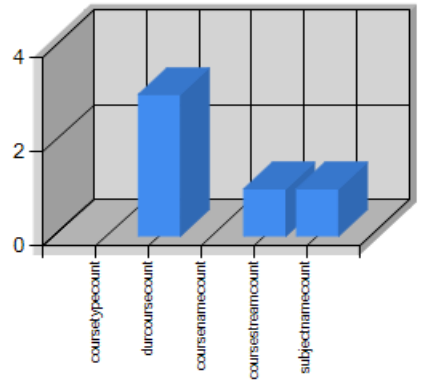


Figure 6.2 : result analysis

The RDF graph indicated in the above figure are the SPARQL query results of case study. Here the count for each query is determined and the time taken can be shown in the graph. The result shown is for larger datasets.

7. CONCLUSION

At present, there are few frameworks(e.g. RDF-3X, Jena, BigOWLIM)for Semantic Web technologies, and these frameworks have limitations for large RDF graphs. To overcome the performance, we describe a framework which is capable of handling large amount of RDF data. Since the framework is based on Hadoop, which is a distributed and highly fault tolerant system, it inherits these two properties automatically.

In case of distributed system, the time taken to answer the query using SPARQL is comparatively less than that of normal SOL query language. The results indicate that for very large datasets Hadoop RDF is preferable and more efficient.

REFERENCES

[1]. **"**Heuristics Based Query Processing for Large RDF Graphs Using Cloud Computing" by Mohammad Husain, James McGlothlin, Mohammad M. Masud, Latifur Khan, Bhavani Thuraisingham

[2]. A. Chebotko, S. Lu, F. Fotouhi, "Semantics Preserving SPARQL-to-SQL Translation", Technical report TR-DB-112007-CLF, 2007

[3]. Jacopo Urbani, Spyros Kotoulas, Eyal Oren and Frank van Harmelen, "Scalable Distributed Reasoning Using MapReduce", International Semantic Web Conference, 2009.

[4]. Mohammad Farhan Husain, Pankil Doshi, Latifur Khan and Bhavani Thuraisingham, "Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce, CloudCom '09": Proceedings of the 1st International Conference on Cloud Computing, 2009.

[5]. Mohammad Farhan Husain, Latifur Khan, Murat Kantarcioglu and Bhavani Thuraisingham, "Data Intensive Query Processing for Large RDF Graphs Using Cloud Computing Tools", IEEE Cloud 2010, pp. 1-10, Miami, Florida, July 2010

[6]. J. Wang, S. Wu, H. Gao, J. Li, B. C. Ooi, "Indexing Multi-dimensional Data in a Cloud System", ACM Int'l. Conference on Management of Data (SIGMOD), 2010.

[7]. Jesse Weaver and James A. Hendler," Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples", Proceedings of the 8th International Semantic Web Conference, 2009