Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 5, May 2015, pg.331 – 340

RESEARCH ARTICLE

Query Performance Appraisal using SPARQL & Map Reduce Technique on Web Semantics

[1]Manoj Kumar, [2]Dr. Yashpal Singh, [3]Dr. S.Niranjan
[1]M.Tech student, Department of Computer Science & Engg, GITAM, kablana, jhajhar, Haryana
[2]Associate Prof., Department of Computer Science & Engg, GITAM, kablana, jhajhar, Haryana
[3]Professor, Department of Computer Science & Engg, GITAM, kablana, jhajhar, Haryana
[1] Manojkumar_mdu@hotmail.com, [2] yashpalsingh009@gmail.com, [3] Niranjan.hig41@gmail.com

Abstract:-** The Semantic Web is an emerging technology which aims at making data across the globe semantically connected. The data is represented in a very simple statement like construct having a subject, predicate and an object. This can be visualized as a graph with the subject and the object as nodes and the predicate as an edge connecting the two nodes. When many statements like these are collected together they forms an RDF graph. There are RDF query languages to query such data, and SPARQL is one of them. According to the SP^2 Bench performance benchmarks, the SPARQL queries are very slow for RDF data with millions of triples. Hence, we aim to develop a Query Performance Appraisal using SPARQL & Map Reduce Technique on Web Semantics model of parallelization and hypothesize that this system will outperform the scalability and performance reported by the SP^2 Bench. We extend ARQ, an open source SPARQL query engine provided by the Jena framework, to work with the Hadoop Map Reduce framework and implement distributed SPARQL query processing. This thesis provides the detailed implementation and algorithmic details of our work. We contribute two novel methods to optimize RDF query engine which exploits document indexes and a join pre-processing technique. The experimental results show the merits and demerits of using Map Reduce for distributed RDF query processing and provide us a clear path for future work.

Keywords:-** Hadoop framework, Cassandra key-value, RDF Dataset, MapReduce, SPARQL, Jena framework, Turtle, RDFa Triplestore, Quadstore, HDFS, N-Triples, SP^2 Bench, Thrift, Hector

INTRODUCTION:- The vision for the Semantic Web is to make the vast knowledge present on the World Wide Web machine readable. This will make data around the globe more meaningful and interpretable by computers. The world is moving towards using Mashups. A Mashup is a service that utilizes data from different sources and provides new services. Applications are being built to churn out semantically useful knowledge from all the available data sources across the geographies, different websites, blogs and information portals. The knowledge is exposed in a very simple statement-like construct. This model is known as

Resource Description Framework (RDF). Data.gov.uk, freebase, dbpedia.org are some of the initiatives which provide huge RDF knowledge stores. To query this knowledge effectively, simple query languages have been suggested. With time, data will grow really huge, and the greatest problem the Semantic Web will face is scalability. Queries that are required to run over billions and trillions of triples have high latency. Research groups and industries are working towards developing scalable solutions that provide low response times. SPARQL is one of the query languages which is used to query RDF data. SPARQL has been recommended by W3C and is considered a key semantic web technology. Present implementations of a SPARQL engine are incapable of taking extreme loads. As reported by the SP^2 Bench, the benchmark queries that run over an RDF dataset of 25 million triples take around 100 to 1000 seconds to respond. We plan to develop a distributed SPARQL query engine to solve this problem and check the performance.

We parallelize the SPARQL query execution over a distributed RDF dataset to achieve Performance benefits. This project aims to implement a distributed SPARQL query engine using MapReduce (introduced by Google), which is a proven parallelization model. The hypothesis that can be put forward is A distributed SPARQL query execution on a MapReduce cluster should provide response times and scalability which outperforms the benchmarks provided by SP^{2}Bench. In this thesis, we discuss the implementation of a **"Query Performance Appraisal using SPARQL & Map Reduce Technique on Web Semantics"** over a Hadoop2 cluster. We extend an existing SPARQL query engine, known as Jena ARQ, which is provided by the Jena framework. We introduce new optimization strategies that were never tried in the context of RDF data. We also introduce a unique approach to preprocess a join. The Join pre-processing technique is based upon pre-computing joins over the complete RDF dataset. We suggest pre-computing joins for the commonly occurring SPARQL query patterns. This prevents the query optimizations engine from performing a data-intensive join operation during the query execution time. Theoretically, the Join pre-processing technique provides better performance and we prove the practical results with experiments. We conclude our work with a performance evaluation of the distributed SPARQL query engine and MapReduce parallelization model and the suggested optimization techniques.

Problem Definition:- We studied the areas where the system underperformed. The query takes normally 100-1000 secs for Data retrieval on big dataset. The time taking for query is relatively slow. Going further, it will be very interesting to investigate how Hadoop can implement distributed indexing, which will be a great advantage for the applications where filtering is preferred over complete scanning of the data. There are efforts to integrate the Cassandra key-value store with Hadoop framework. This will provide us a way to implement distributed indexing. We would like to evaluate a proof of concept to build a system which makes use of Cassandra key-value store to contain the distributed indexes over the RDF data. Next, we would like to implement dictionary encoding as a way to compress the data. We currently implemented only a subset of SPARQL query features and once we build a good filtering model, we would like to take this further by implementing all the features of a SPARQL and build a planet scale RDF triplestore.

We demonstrated the performance appraisal benefits that can be achieved using the novel optimization approach of join preprocessing. The system scaled well for large amount of RDF data, but the response times were still unacceptable and we should aim for better response time.

The system still lacks the precision and performance that can be achieved by implementing indexes. We witnessed this with the results of the different queries. Our system performed well with the query Specific as we used the document indexing technique to filter relevant documents. There are many areas of improvement but, to prove our hypothesis, we need to invest more time in investigating ways to implement distributed indexing which should prove beneficial results. We reached to a conclusion that the real bottleneck was the way Map Reduce model supports the Selection phase. The Hadoop framework needs to be designed for filtering when it only supports scanning at present. This work can be concluded to be successful, as we implemented, experimented and analysed the aspects of the system that we initially planned. We contributed two novel approaches for optimization in context of querying RDF data, that have been proved to be very effective. Although there are many areas of improvement, this work provides us a clear direction for the future work on response time & improvement.

BACKGROUND

1. RDF

RDF is an acronym for Resource Description Framework (RDF). It is a simple language description model that makes statements about a resource by either defining its relationships with other resources or by defining its attribute. The resource is the subject, the relationship is the predicate, and the attribute value is the object. If this is visualized from an object oriented perspective, an RDF "Subject - Predicate - Object" can be mapped to an "Object - Property - Value" relationship of an Object. However, RDF is not based upon object oriented paradigm. It does not provide encapsulation of attributes. This provides flexibility to add statements about a resource without having to modify any encapsulating entity associated with it. For instance, ":tom :drives :car" is an RDF statement, where :tom is the Subject, :drives is a predicate and the :car is an object. RDF can also be visualized as a graph where the Subject and the Object are two nodes, and the predicate forms an edge between the two. These are called Triples.

By convention, the subject and the object values are always shown as a node in the RDF graph. RDF follows a URI based vocabulary and node identification. Thus the resources can be identified by URIs or by Blank nodes. The concrete values can be literals along with the data-type specification. The Subject, Predicate and the Object can all be URIs or Blank nodes, but only the Object can be a literal. The literals are always denoted by a rectangular box in an RDF graph. Hence, in an RDF graph the Object can be presented by a Node or a Box. The box is labelled with the literal value. Various serialization formats are defined for the RDF triples. Notation-3, Terse RDF Triple language (Turtle), N-triples, RDF/XML are some of the popular formats. Website owners willing to make their web pages machine-readable should provide An RDF version of the web pages along with the HTML data. The specifications like RDF and the Micro formats should be followed to embed the RDF data into web documents.

The RDF data can be put in the meta-data section of an HTML page, or can be embedded as an attribute in the HTML tags. RDF scrapers (or parsers) pull out the required RDF relationships from the existing HTML pages. Hence, the RDFa and Microformats tags co-exist with the HTML tags and do not alter the presentation of web document.

2. SPARQL Query:- SPARQL is an RDF query language, its name is a recursive acronym which stands for "SPARQL Protocol and RDF Query Language". SPARQL, which is officially endorsed by W3C, is a key semantic web technology . SPARQL Query supports triple patterns, conjunctions, disjunctions and optional patterns, and it supports writing globally unambiguous queries. The SPARQL query engine finds the triples with Graph patterns matching to that of the query. The given RDF statement is the best match because the Subject of the query pattern matches with the subject of RDF statement, and similarly the Predicate of the query pattern matches with the Predicate of the RDF statements.

3. Jena:- As the Jena web page quotes "Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine." Jena has migrated from being an in-house HP6 project to an open-source project. Jena provides APIs to parse RDF data from different formats, and write RDF in various supported formats. Jena is based on the Java programming language and has good community support. The Jena source code is open source and is extensible. Jena provides sufficient documentation to work with the Jena APIs. The Jena package also provides ARQ which is the SPARQL query engine implementing SPARQL specification. ARQ makes use of core Jena APIs. ARQ is built for small scale data and uses an in memory implementation. The RDF data is read and maintained in a customized HashMap like data structure, where indexes are created on all three columns i.e subject(s), predicate(p) and object(o). These indexes help with the quick retrieval of the required data. Jena also provides the TDB triplestore, which has been developed for large scale RDF data storage and retrieval. TDB is a centralized implementation, and it follows a similar approach of indexing as that of ARQ, by creating B-Tree indexes on the data with different keys i.e., subject-predicate-object, predicate-subject-object, object-subject-predicate and all other combinations.

4. MapReduce Programming model MapReduce is a programming model, which provides simple way of parallelizing complex tasks. MapReduce is inspired from the functional programming language that provide map and reduce primitives. The framework surrounding MapReduce model manages the concerns related to work distribution, fault-tolerance, data locality over a distributed file system and provides abstraction to implement the programming logic in Map and Reduce methods. This model is analogous to a split and aggregate model of parallelization, where a given task that needs to be performed on a set of data is managed by executing the task in parallel on the chunk of data splits and later aggregating the results of all the tasks to provide the final solution. The Map is analogous to the split phase and the Reduce is analogous to the aggregation.

Map Reduce concept

Map

(k1 , v1) −> l i s t (k2 , v2)

Reduce

(k2 , l i s t (v2)) −> l i s t (k3 , v3)

As shown above, the map and reduce functions have strict contracts. The map gets a key/value pair as an input and produces list of intermediate key/value pairs. The MapReduce framework makes sure all the keys that emerge from the output of the map phase get grouped together and this is given as input to the Reduce function which accepts a key and a list of values as input and provides a list of key/value pairs as output. The programs written in this style are automatically parallelized by the MapReduce framework. MapReduce framework does not expect any specialized cluster for parallelization, a simple cluster made up of commodity machines can be used to form a MapReduce cluster.

5. Hadoop framework

Hadoop framework is an open source implementation of the MapReduce parallelization model. It is a top-level Apache project that is being built by contributors from all over the world. Hadoop provides a distributed file system (HDFS) that is designed to run over a commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on a low-cost hardware which is prone to frequent failures. It provides high throughput access to application data and is suitable for applications with large dataset. The data is replicated at three places over the distributed nodes which provides reliability and provides an opportunity to exploit the data locality as much as possible. The big files that are put on HDFS are broken down into chunks of 64MB files that are distributed across the cluster. When a Hadoop MapReduce job is executed, the built Job Tracker and Task Schedulers assign the tasks to the nodes where the application data exist. They make sure the tasks are run successfully and the results are returned back to the user. Hadoop requires the programmers to implement simple interfaces like the Mapper and the Reducer. These customized Mappers and reducers are set into a configuration object which makes the Hadoop job runner aware of the classes it should use. Hadoop is highly scalable to large amounts of data and is very popular in the industry for its extensive library and simple usage.

6. Apache Cassandra key-value store

Cassandra is a highly scalable distributed database that provides features of a key-value store as well as column oriented databases. Cassandra can be easily installed on a cluster of commodity machines. Although Cassandra has many interesting features, we use Cassandra as a key-value store for storage of large inverted indexes over a collection of document. Cassandra provides very fast searches for keys. We have presently used Cassandra only as a standalone system, but when the data storage requirement increases we can easily setup a cluster wide Cassandra installation. Clients communicate with Cassandra using Thrift APIs. We implemented a Data Access Layer which talks to Cassandra using a library called Hector.

Algorithm For Basic Graph Pattern Matching:-

This algorithm is split into two phases, the Selection phase and the Join phase. One Map Reduce job is run for each phase. The Selection phase filters the input RDF data triples that match at least one of the query patterns. If the Basic Graph Pattern has joins to be evaluated then the output of the Selection phase is used by the Join phase. The Join phase runs iteratively when the Basic Graph Pattern consists of joins on multiple keys. We discuss the input, output and the processing done by each phase.

1. Selection Phase :-

Input The list of query patterns appearing in a Basic Graph Pattern is the main input to the Selection phase.

?name1 :likes: p
?name2 :loves: ?name3

Processing The selection phase scans the local the local RDF data record-wise, where each record is an RDF triple. This triple is matched with all the given query patterns until it matches with at least one of them. All the triples that match at least one query pattern are grouped on the basis of the query pattern number they match with.

Output The selection phase generates different files for each query pattern from the given list. The file is created only when the query pattern selects at least one RDF triple. The files are generated on the Map Reduce cluster's distributed file system.

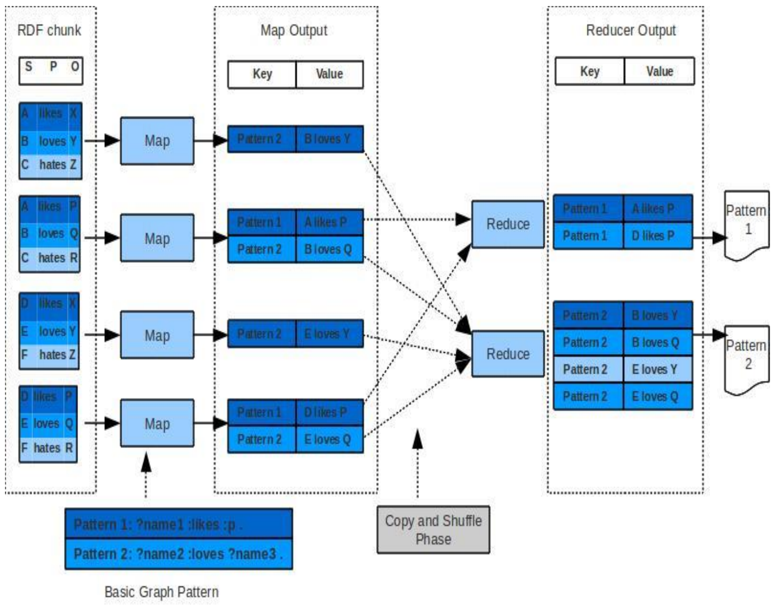

Fig:- Map Reduce Selection Phase

2. Map Phase:-

The map function receives a Hadoop generated record number as the key, which is irrelevant to the map logic. The value is a triple, read from the RDF data chunk. The mapper iterates over the list of query patterns and matches the query pattern with the input triple. The matching algorithm checks if the pattern subject, predicate or the object is concrete. A concrete node has an explicit value which can be a URI, a Blank node or a Literal. A pattern containing a variable at the place of the subject, predicate and the object will not be concrete. The input triple: subject, predicate and object are matched with the corresponding subject, predicate and object of the query pattern. A variable in a query pattern matches with any value at the corresponding place in the input triple. This is a valid match because the value in the input is bound with the corresponding variable, which can be a candidate for the final query result. If the input triple matches with at-least one query pattern, the mapper emits the query pattern number as the key and a string representation of the bindings as a value.

3. Reduce Phase:-

The input key to the reduce phase is a join key binding and the input value is a list of all the values that were associated with the input key. The reducer task is initialized with the list of patterns that need to be joined in this phase. The reducer iterates through the list of values for a given input key and fetches the tag. With the help of the tag, it verifies whether the group values have come from all the pattern files that are being joined. If the group does not have a value from some pattern file that needs to be joined, that group is an invalid group, and it is rejected. The valid group values are emitted as a key, value pair, where the key is a new file name, which is a concatenation of all the pattern file names that are joined and the value is the binding. Thus if Pattern1, Pattern2 and Pattern3 are joined, the new file name formed will be Pattern1Pattern2Pattern3.

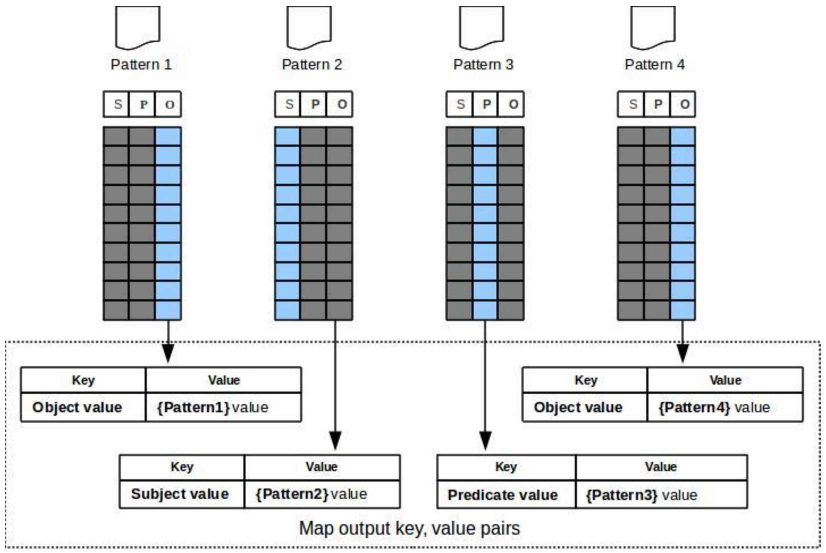

Fig:- Mapping technique for multi-way join

Results

Raw implementation test This tested the average response times of the queries with no optimizations applied to our system. The response times were tested on a dataset of 5M triples. These were tested on the 5 nodes cluster. The results are shown in Table 1.

No. of triples	Q1	Q3a	Q3b	Q3c	Q10
250K	212	233	188	78	65
1M	334	354	265	134	122
5M	743	542	429	178	166
25M	1675	1231	722	455	134

Table 1:- Raw implementation test

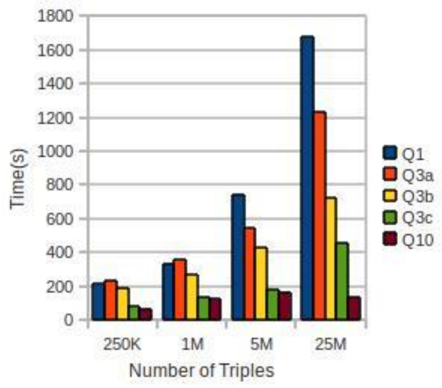

Fig:- Results for benchmark queries on the Raw implementation

Speed up This test was to witness the system performance with the increasing number of nodes for a consistent data-size. The three clusters: the single-node cluster, the 5 nodes cluster and the 8 nodes cluster were loaded with a data size of 25M triples.

The document indexes were loaded onto a standalone Cassandra Key value store. The Cassandra key-value store was shared by all the three clusters. Tests were run on each cluster, one at a time, and we also performed other tests on all the clusters simultaneously. However, Cassandra's performance was consistent.

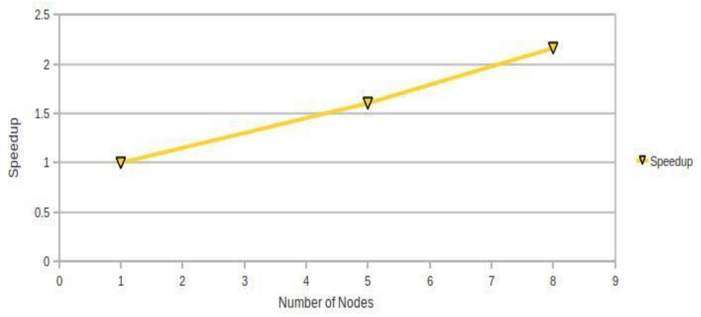

Fig:- Speed up of the Optimized implementation

Future work

We studied the areas where the system underperformed. Going further, it will be very interesting to investigate how Hadoop can implement distributed indexing, which will be a great advantage for the applications where filtering is preferred over complete scanning of the data. There are efforts to integrate the Cassandra key-value store with Hadoop framework. This will provide us a way to implement distributed indexing. We would like to evaluate a proof of concept to build a system which makes use of Cassandra key-value store to contain the distributed indexes over the RDF data. Next, we would like to implement dictionary encoding as a way to compress the data. We currently implemented only a subset of SPARQL query features and once we build a good filtering model, we would like to take this further by implementing all the features of a SPARQL and build a planet scale RDF triplestore.

REFERENCES

[1] Dbpedia. http://dbpedia.org/About, 2010.

[2] Jeffery Dean and Sanjay Ghemawat. MapReduce : Simplified data processing on large clusters. OSDI, San Francisco, CA, 2004.

[3] The Foundation. The apache software foundation. http://www.apache.org/ foundation/, 2010.

[4] The Apache Software Foundation. MapReduce. HADOOP. http://hadoop.apache.org/mapreduce/, 2007.

[5] The Apache Software Foundation. Cassandra wiki. http://wiki.apache.org/cassandra/, 2009.

[6] The Apache Software Foundation. Thrift wiki. http://wiki.apache.org/thrift/, 2009.

[7] The Apache Software Foundation. Heart proposal.

http://wiki.apache.org/incubator/HeartProposal, 2010.

[8] Freebase. http://www.freebase.com/, 2010.

[9] UK Government. Resource description framework (RDF). http://data.gov.uk/, 2010.

[10] Allan Hollander. The semantic naturalist.

http://cain.ice.ucdavis.edu/semanticnaturalist/?c=Data-Linking, 2008.

[11] S. Lee J. Myung, J. Yeon. SPARQL basic graph pattern processing with iterative MapReduce. MDAC, Raleigh, NC, USA, April 2010.

[12] Z. Miao J. Wang. Querying heterogeneous relational database using sparql. Eigth IEEE/ACIS International Conference on Computer and Information Science, 2009.

[13] Merja Karjalainen. Uniform query processing in a federation of rdfs and relational resources. Proceedings of the 2009 International Database Engineering & Applications Symposium, Cetraro Calabria, Italy 2009.

[14] Lance. http://blog.computationalcomplexity.org/2010/07/ drowning-in-data.html, 2010.

[15] Georg Lausen Christoph Pinkel Michael Schmidt, Thomas Hornung. SP2Bench: A SPARQL performance benchmark. IEEE International Conference on Data Engineering, 2009, Beijing, China, 2009.

[16] Latifur Khan Mohammad Farhan Husain, Pankil Doshi and Bhavani Thuraisingham. Storage and retrieval of large RDF graph using hadoop and mapreduce. Proceedings of the 1st International Conference on Cloud Computing, Beijing, China, 2009.

[17] J. Freytag O. Hartig, C. Bizer. Executing SPARQL queries over the web of linked data. ISWC 09, 2009.

[18] OpenJena. Jena a semantic web framework for java. http://jena.sourceforge.net/, 2010.

[19] OpenJena. SPARQL tutorial - a first SPARQL query. http://www.openjena.org/ARQ/Tutorial/query1.html, 2010.

[20] G. Tummarello P. Mika. Web semantics in the clouds. Yahoo Research, 2009.

[21] Chris Dollin Paolo Castagna, Andy Seaborne. A parallel processing framework for rdf design and issues. HP Laboratories, 2009.

[22] Jacopo Urbani. RDFS/OWL reasoning using the MapReduce framework. Master's thesis, Vrije Universiteit - Faculty of Sciences, Dept. of Computer Science,2009.

[23] W3C. Resource description framework (RDF). http://www.w3.org/RDF/,November 2006.

[24] W3C. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.

[25] W3C. RDFa. http://www.w3.org/TR/xhtml-rdfa-primer/, 2010.