Processing RDF Using Hadoop

Mehreen Ali[1], K. Sriram Bharat[1], and C. Ranichandra[2]

[1] MS (Software Engineering), VIT University, Vellore, India
mehreen1591@yahoo.com, sriramkondeti@gmail.com
[2] Assistant Professor (Senior), VIT University, Vellore, India
cranichandra@vit.ac.in

Abstract. The basic inspiration of the Semantic Web is to broaden the existing human-readable web by encoding some of the semantics of resources in a machine-understandable form. There are various formats and technologies that help in making it possible. These technologies comprise of the Resource Description Framework (RDF), an assortment of data interchange formats like RDF/XML, N3, N-Triples, and representations such as RDF Schema (RDFS) and Web Ontology Language (OWL), all of which help in providing a proper description of concepts, terms and associations in a particular knowledge domain. Presently, there are some existing frameworks for semantic web technologies but they have limitations for large RDF graphs. Thus storing and efficiently querying a large number of RDF triples is a challenging and important problem. We propose a framework which is constructed using Hadoop to store and retrieve massive numbers of RDF triples by taking advantage of the cloud computing paradigm. Hadoop permits the development of reliable, scalable, proficient, cost-effective and distributed computing using very simple Java interfaces. Hadoop comprises of a distributed file system HDFS to stock up RDF data. Hadoop Map Reduce framework is used to answer the queries. MapReduce job divides the input data-set into independent units which are processed in parallel by the map tasks , which then serve as inputs to the reduce tasks. This framework takes care of task scheduling, supervising them and re-execution of the failed tasks. Uniqueness of our approach is its efficient, automatic allocation of data and work across machines and in turn exploiting the fundamental parallelism of the CPU cores. Results confirm that our proposed framework offers multi-fold efficiencies and benefits which include on-demand processing, operational scalability, competence, cost efficiency and local access to enormous data, contrasting the various traditional approaches.

Keywords: Semantic Web, Distributed Computing, Map-Reduce Programming, SPARQL, Graph Data, Performance Evaluation.

1 Introduction

Cloud computing is a budding concept in the IT and data dispensation communities. Various new ventures are utilizing cloud computing service to contract out data safeguarding, which results in noteworthy financial benefits. Enterprises usually stock up and access data at far-off sites in the “cloud”. As the reputation of cloud computing

N. Meghanathan et al. (Eds.): Advances in Computing & Inform. Technology, AISC 177, pp. 385–394.
springerlink.com

nurtures, the service providers come across several challenges. They have to maintain gigantic magnitudes of heterogeneous data while providing well-organized information retrieval. Hence the prominence for usage of cloud computing is scalability and query efficiency.

Semantic Web technologies [2] are being urbanized to present data in standardized way such that the data can be retrieved and understood by both human and machine. Traditionally, web pages are published in simple html files and hence are not suitable for interpretation. Instead, the machine regards these html files as a container of keywords. Researchers are cultivating these Semantic Web technologies to address such shortcomings. The most well-known standards are Resource Description Framework1 (RDF) and the SPARQL Protocol and RDF Query Language (SPARQL). RDF is the standard for piling up and expressing data and SPARQL is a query language to retrieve data from an RDF stockpile. Via OWL (Web Ontology Language) ontologies, different schemas, classes, data types and relationships can be represented without forgoing the benchmark RDF/SPARQL interface. Figure 1 reflects the fundamental organisation of Semantic Web.

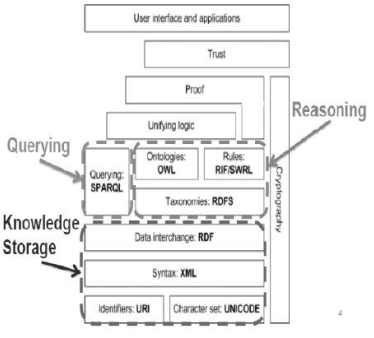

Fig. 1. Semantic Web Layer Cake

Semantic web datasets are mounting exponentially. More than any other domain, in the web arena, scalability is supreme. Nevertheless, high speed response time is also critical in the web society. Cloud computing concept offers a way out that can achieve both the goals. Present industrial tools and technologies do not dwell well in Cloud Computing scenery.

A distributed system can be fabricated to triumph over the scalability and performance issues of present Semantic Web frameworks. Datasets are being disseminated to help in providing such scalable solutions. Yet, till date, there isn't any distributed storehouse for storing and managing RDF data. We put forward a solution with a generic distributed storage system which makes use of a Cloud Computing platform. And then propose a way to tailor the system and schema explicitly to meet the demands of semantic web data. Finally, we recommend constructing a semantic web storehouse using such a storage provision.

In this paper we spotlight on the design characteristics inherent to using the MapReduce software framework to assemble highly-parallel, high-performance and scalable data management systems. Our views and discussions in these fields are

stimulated by our understanding of designing, constructing and evaluating our preliminary implementation of a triple store which is vigorous, robust, scalable and distributed.

A triple-store is a data storage and recovery environment for graph data, conventionally represented in RDF formats [15]. Our triple store stores graph data [17] as RDF triples and retorts to queries over this data using SPARQL query language. We make use of the Hadoop implementation of MapReduce to build it.

2 Hadoop: The Best Implementation of MapReduce

MapReduce is a software framework used to process and generate large datasets. Users state a map function that divides data into key/value pairs and a reduce function that amalgamates all key/value pairs on the basis of the key.

MapReduce software framework is effortlessly parallelizable for implementation on bulky clusters of commodity equipments. This facilitates the production of high-performance, highly-scalable applications. The most accepted MapReduce implementation is Hadoop. Hadoop controls the management of data on compute nodes by making use of the Hadoop Distributed File System (HDFS), scheduling the program's execution over a set of machines, managing machine breakdowns, and handling the obligatory inter-machine communication. This paves way for the design and implementation of high-level functionality by means of the MapReduce framework to put up high-performance and highly scalable applications.

A main feature of the MapReduce software framework, as articulated in the Hadoop implementation, is the utilization of a unique, centralized node, referred to as the NameNode which guides the posting of data onto compute nodes using HDFS, allocates compute jobs to the diverse nodes, traces fiascos and supervises the shuffling of data after the Map step concludes.

The unit of computation in Hadoop is referred to as a job. Here the users submit jobs to Hadoop's JobTracker module. Every job consists of two phases: Map and Reduce. The Map phase takes a key-value pair as input and possibly will output zero or more key-value pairs. In Reduce phase, the values of every key are assembled into compilations traversable by an iterator. The generated key-iterator pairs are in turn passed to the Reduce method, which as well outputs zero or more key-value pairs. When a job is presented to the JobTracker, Hadoop attempts to place the Map processes near to the entered data in the cluster. All the Map process and Reduce process work autonomously without communicating and hence beneficial for both swiftness and ease.

3 Design Goals

Our principal data system design driving force is the knack to persist and speedily query hefty data graphs. To align with Semantic Web data principles, we consider graphs characterized as subject-predicate-object triples [2][6]. A small illustration graph can be seen in Figure 1 that contains 7 triples –

John lives in Pune, John owns an object car0, car0 is a car, car0 was made by Ford, car0 was made in Delhi, Delhi is a city and Pune is a city.

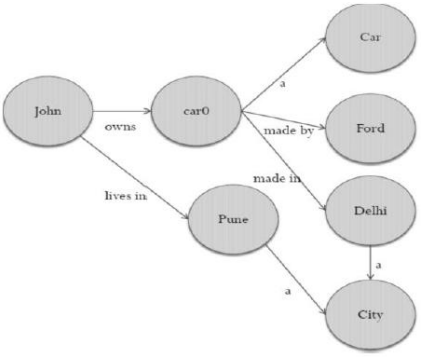

Fig. 2. A Small Graph of Triple Data

We make use of SPARQL [18] -it is the benchmark Semantic Web query language. SPARQL semantics are broad-spectrum and similar to the well-known SQL. An example SPARQL query for the above graph data is the following:

```
SELECT ?person WHERE
{
?person :owns ?car .
?car :a :car .
?car :madeIn :Delhi.
}
```

The above query written in SPARQL language has three clauses and looks for all equivalences to the variable *?person* such that `?person` owns a unit specified by the variable `?car` which is a car and was made in Delhi. The above query corresponds to the directed graph as seen in Figure 2.

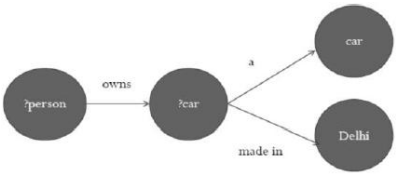

Fig. 3. A Directed Graph Representation of a Query

Processing of SPARQL queries in the perspective of a data graph such as the one above involves identifying which variables in the query clauses can be related to nodes in the data graph in order to let the query clauses align with the data triples [16]. This process of alignment for query processing is moderately universal across various data depictions and query languages. An example of this sort of alignment for our example query can be seen in Figure 4.

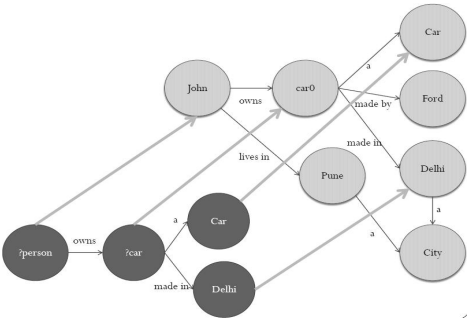

Fig. 4. Alignment of SPARQL Query Variables with Triple Data

The main functional design objectives for the triple-store are to:

1. Serve as an unrelenting store for triple data in RDF set-up.
2. Provide as a SPARQL endpoint to process SPARQL queries.

There have been several other design approaches for triple-stores with comparable design goals. Several of these triple-stores have accomplished superior performance on solo compute-node systems [14] by making use of designs based on memory mapping index information [9]. Nevertheless, disk and memory restrictions have motivated the necessity for distributed computing tactics to triple-stores [10][12].

4 Proposed Architecture of Our Framework

Our architecture comprises of two components. The upper part of Figure 5 portrays the data pre-processing component and the lower part portrays the query answering one. There are three sub-components for data generation and pre-processing. Conversion of RDF/XML to N-Triples serialization format is done using N-Triples Converter module.

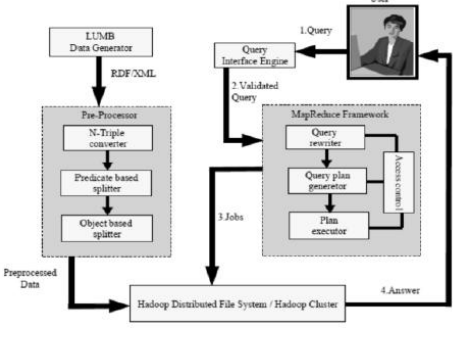

Fig. 5. The System Architecture

The PS module splits the N-Triples data into predicate files. These predicate files are in turn fed into the POS module which divides the predicate files into smaller files based on the type of objects. Our MapReduce framework has three subcomponents in it. It inputs the SPARQL query from the user and feeds it to the Query Rewriter and Query Plan Generator. This module picks the input files to decide how many MapReduce jobs are required and then pass the data to the Plan Executer module which runs the jobs using MapReduce framework. It then dispatches the query result from Hadoop to the user.

4.1 Data Partitioning and Triple Placement

In order to use a distributed computing approach to information management system design, it is generally infeasible to pass large volume input data directly to and from the user. This data passing would involve the coordinated movement of data onto and off of the compute nodes as and when the data needs to be processed. The large magnitude of data makes this approach unfeasible owing to data churn. As a result, hefty input (data and queries) and output (results of query) data sets are required to be stored directly onto the compute nodes. This direct storage of data on the compute nodes is done natively by means of the Hadoop implementation of MapReduce which involves placing data in the HDFS distributed file system.

Data is maintained in the form of flat files in the HDFS file system i.e. each line of the triple-store text file represents all triples related with a different subject. Though this approach of persisting triple data in the form of flat text files is rudimentary when evaluated against other data management approaches, it brings a level of automated robustness which is achieved by replicating the data and MapReduce operations across various nodes [7]. The data is stored in a simple, easy to read set-up that imparts itself to easier, user centric drill-down investigation of query results returned from the triple store. [13]

4.2 Query Execution

MapReduce offers only plain data manipulation techniques by splitting the data into key-value pairs, and combining all values with the same keys. For complex query processing, there is a need for data management systems to iterate over clauses of the queries to incrementally bind variables of the query to the literals in the triple store while fulfilling all the constraints of the query. All iterations consist of a MapReduce operation intended for a single clause in the query.

The initial map step maps the data in the triple store to a list of variable bindings such that the first clause of the query is satisfied. The importance of the Map step is the list of variable bindings. Once done, the Reduce step discards duplicate results and uses variable binding as the key to save them to disk.

The intermediary query binding steps iteratively bind variables to literals as and when new variables are introduced by processing succeeding query clauses and additionally sieving out the previous bindings which cannot satisfy the new clauses. These intermediate steps carry out MapReduce operations over the triple data as well as the earlier bound variables saved to disk.

The ith intermediate Map step's job is to identify all variables in the triple-data which satisfy the ith clause and thus save this result with the key as any of the variables in the ith clause which surfaced in previous clauses. The value of this Map step is the bindings of the variables that have not been seen in the previous clauses. The iteration of this Map step in addition rearranges the results of the prior variable bindings which have been saved to disk to the equivalent name of a variable key present in the ith clause that materialized in previous clauses.

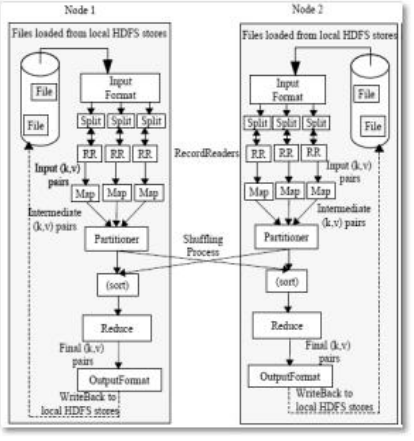

Fig. 6. Iteration of MapReduce to Process SPARQL Queries

Consequently, the ith Reduce step performs a join function on top of the intermediary results from the Map step by iterating over the entire results pairs from the prior clause and the recent clause by means of the assignment of the same key.

These map-reduce-join iterations continue until the entire clauses in the query are administered and variables are assigned such that they satisfy all the query clauses. Our triple-store design saves the intermediate results of the query processing onto local disk to increase the swiftness in processing of similar queries which might come into view later. The finishing step of MapReduce carries out filtering operations on the bound variable assignments in order to satisfy the main `SELECT` clause of the respective SPARQL query.

5 System Implementation and Initial Results

In this segment, we draft an initial design of our system and show preliminary results. In order to test the performance of our proposed design based on MapReduce framework for a scalable data management system, we built a premature version of triple store using the Cloudera Hadoop(CDH3) that we deployed on top of an Amazon EC2 [1] cloud environment comprised of 20 XL nodes running RedHat Linux and Cloudera Hadoop.

5.1 Lehigh University Benchmark(LUBM)

We made use of the LUBM benchmark [5] to assess the performance of our triple store. This benchmark generates synthetic data concerning the publishing, coursework and advising activities of the faculty and students belonging to different departments in various universities.

Once the data triples were stocked up onto the triple store, we assessed our framework's performance in responding to 1st, 9th and 14th query of LUBM as these were the queries used for studying the performance of previous triple-store. 1st query is basically very plain which asks for the all students that take a particular course. Its response is a very small set of triples. 9th query is fairly complicated query comprising of a triangular pattern of relationships and it asks for all the teachers, students and courses such that the teacher is the adviser of the student who takes a course taught by the teacher. 14th query is reasonably simple as it just asks for all the undergraduate students but the response generated includes very large set of triples.

5.2 Summary of Results

Our triple store evaluation attained the following query response time for the 3 queries for 6000 universities which corresponds to approximately 800 million triples using the LUBM benchmark when deployed on an Amazon EC2 cloud with 20 compute nodes:

Query 1: 323 sec. (approx 0.1 hr.); **Query 9**: 560 sec. (approx 0.2 hr.); **Query 14**: 100 sec. (approx 0.03 hr.)
For evaluation, in the triple store study the industrial single-machine DAMLDB triple-store achieved the following performance on the same queries in combination with the Sesame2 and Jena3 Semantic Web frameworks in order to assist in query processing.

Sesame+DAMLDB took:

Query 1: approx 0.1hr.; **Query 9**: approx 1 hr.; **Query 14**: approx 1 hr.
For Jena+DAMLDB, owing to the difficulty in loading triples into this dataset, we cannot have data on performance over 550 million triples. However based on observed trends this triple-store probably would require the following response times:

Query 1: approx 0.001 hr., **Query 9**: approx 1 hr.; **Query 14**: approx 5 hr.

It is to be noted that the only query where our framework performed evidently worse than DAMLDB was on query 1. Query 1 returns a very small subset of values bound to variables. Even though MapReduce is conventionally used to build indices, its implementation of Hadoop provides little support for accessing data stocked up in HDFS files. Conversely, DAMLDB makes use of a number of exceptional indexing optimizations for plain queries like that for Query 1. Our triple store does not implement such optimization techniques. Barring this one drawback, our approach performed better than other existing technologies owing to the vastly parallel and distributed implementations of the MapReduce framework.

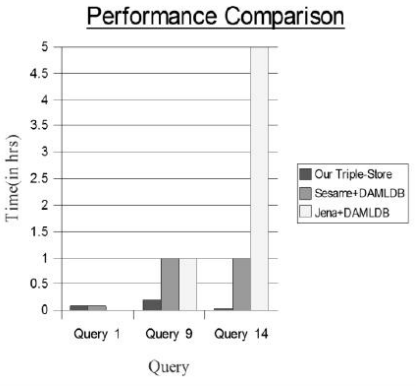

Fig. 7. Graph Comparing Performances of Various Triple-Stores

6 Future Directions

On basis of our understanding of the initial operation of our proposed framework, we have quite a few short and long-term actions needed to further improve the performance and acceptance equally from design as well as software framework perspective.

Foremost improvement needed is a more effectual method for indexing data. In addition to this, we need a supporting alternative to Map-Reduce that supports native indexing as a replacement for the basic Map operations over all the stored data elements.

In addition to this we can make improvements in performance of our design which can be made available by caching the partial results locally for high performance parallel operations as well as globally using a entity like NameNode which will track the local caching of the partial results. To achieve this we require additional potential in the software framework to keep track of the partial results which were cached beforehand and perhaps to decide which cached results can possibly be discarded to produce free disk space in the cloud. This should be done only if it is a deployment concern.

A major advancement that can be done to support the design of information systems would be a substituting software framework that aids in data linking. As an alternative of storing lists of data in the form of flat files in constructs similar to HDFS, a software framework can be built that can offer a inherent linked-data construct which couples the data elements and the pointers to related data. The resulting linked-data framework will provide more rapid localized processing of queries without a need for exhaustive searching of the data set for each and every query request.

References

[1] Amazon. Amazon EC2 Instance Types (2010), http://aws.amazon.com/ec2/instance-types/

[2] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American Magazine (May 17, 2001)

[3] Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Proceedings of the USENIX Symposium on Operating Systems Design & Implementation, OSDI, pp. 137–147 (2004)
[4] DeWitt, D., Stonebraker, M.: MapReduce: A major step backwards, http://database-column.com, http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/ (retrieved August 28, 2010)
[5] Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2), 158–182 (2005)
[6] Grigoris, A., van Harmelen, F.: A Semantic Web Primer, 2nd edn. The MIT Press (2008)
[7] Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable Distributed Reasoning Using MapReduce. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 634–649. Springer, Heidelberg (2009)
[8] Hendler, J.: Web 3.0: The Dawn of Semantic Search. IEEE Computer (January 2010)
[9] Kolas, D., Emmons, I., Dean, M.: Efficient Linked-List RDF Indexing in Parliament. In: The Proceedings of the Scalable Semantic Web (SSWS) Workshop of ISWC 2009 (2009)
[10] Li, P., Zeng, Y., Kotoulas, S., Urbani, J., Zhong, N.: The Quest for Parallel Reasoning on the Semantic Web. In: Liu, J., Wu, J., Yao, Y., Nishida, T. (eds.) AMT 2009. LNCS, vol. 5820, pp. 430–441. Springer, Heidelberg (2009)
[11] LinkingOpenData (2010), http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[12] Mika, P., Tummarello, G.: Web Semantics in the Clouds. IEEE Intelligent Systems 23(5), 82–87 (2008)
[13] Husain, M., McGlothlin, J., Masud, M.M., Khan, L., Thuraisingham, B.: Heuristics Based Query Processing for Large RDF Graphs Using Cloud Computing. Journal of Latex Class Files 6(1) (January 2007)
[14] Project Voldemort (2010), http://project-voldemort.com/
[15] RDF. Resource Description Framework (RDF) (2010), http://www.w3.org/RDF/
[16] Rohloff, K., Schantz, R.: High-Performance, Massively Scalable Distributed Systems using the MapReduce Software Framework: The SHARD Triple-Store. In: International Workshop on Programming Support Innovations for Emerging Distributed Applications, PSIEtA (2010)
[17] Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: Evaluation of Triple-Store Technologies for Large Data Stores. In: 3rd International Workshop on Scalable Semantic Web Knowledge Base Systems, SSWS 2007, Vilamoura, Portugal (2007)
[18] SPARQL. SPARQL Query Language for RDF (2010), http://www.w3.org/TR/rdf-sparql-query/