Abstract

With the increased use of semantic web in the recent years has led RDF (Resource description format) become the standard format for encoding machine-readable information in the Semantic Web. As a result SPARQL has become the most emerging query language for querying RDF data. Optimizing the SPARQL query in different ways reduces the querying cost and increase the speed of query execution. Thus Query optimization has been a key research area in database communities. In this paper we are going to analyze and briefly survey different categories for optimizing SPARQL queries and the research work done so far.

Keywords: SPARQL Query Optimization, RDF, Query Rewriting, Query execution Time

1. INTRODUCTION

Tim Berners-Lee has a two-part vision for the future of the Web. The first part is to make the Web a more collaborative medium. The second part is to make the Web understandable, and thus processable, by machines. Semantic web can be defined as a machine processable web of smart data [19].

The great advantage of Web 1.0 was that it abstracted away the physical storage and networking layers involved in information exchange between two machines. This breakthrough enabled documents to appear to be directly connected to one another. Click a link and you're there—even if that link goes to a different document on a different machine on another network on another continent.

In the same way that Web 1.0 abstracted away the network and physical layers, the Semantic Web abstracts away the document and application layers involved in the exchange of information. The Semantic Web connects facts, so that rather than linking to a specific document or application, you can instead refer to a specific piece of information contained in that document or application. If that information is ever updated, you can automatically take advantage of the update [20]. The word semantic itself means meaning. As such, the fundamental difference between Semantic Web technologies and other technologies related to data (such as relational databases or the World Wide Web itself) is that the Semantic Web is concerned with the meaning and not the structure of data.

1.1. RDF

The Resource Description Framework (RDF) is a language designed to support the Semantic Web. RDF is a framework for supporting resource description, or metadata (data about data), for the Web. RDF provides common structures that can be used for interoperable XML data exchange [21]. The adaptation of the relational model to the Web gives rise to RDF.

RDF databases are collections of so-called triples of knowledge, where each knowledge triple in the database is of the form (subject, predicate, object) and models the binary relation predicate between the subject and the object which are RDF terms, e.g. IRIs, literals or blank nodes. The objective of RDF is to support the interoperability of metadata. RDF allows descriptions of Web resources - any object with a Uniform Resource Identifier (URI) as its address - to be made available in machine understandable form. This enables the semantics of objects to be expressible and exploitable. Once highly deployed, this will enable services to develop processing rules for automated decision-making about Web resources [22].

RDF is based on a concrete formal model utilizing directed graphs that elude to the semantics of resource description. The basic concept is that a Resource is described through a collection of Properties called an RDF Description. Each of these Properties has a Property Type and Value [22]. Any resource can be described with RDF as long as the resource is identifiable with a URI as shown in Figure 1.

Figure 1: Resource Description framework model [22]

A Resource is anything that can have a URI, such as http://www.w3schools.com/rdf"
A Property is a Resource that has a name, such as “author” or “homepage"
A Property value is the value of a Property, such as “Raval Shruti” or "http://www.w3schools.com” (note that a property
value can be another resource)

2. SPARQL

SPARQL allows you to query for triples from an RDF database (or triple store). Superficially it resembles the Structured Query Language (SQL) used to get data from a relational database. A relational database is table based, meaning that data is stored in fixed tables with a foreign key relationship that defines the relationship between rows in the tables. A triple store stores only triples, and you can pile the triples as high as you like while describing a thing. With relational databases you are confined to the layout of the database. RDF doesn't use foreign and primary keys either. It uses URIs, the standard reference format for the World Wide Web. By using URIs, a triple store immediately has the potential to link to any other data in any triple store. That plays to the distributed strengths of the Web. Because triple stores are large amorphous collections of triples, SPARQL queries by defining a template for matching triples, called a Graph Pattern[24]. To get data out of the triple store using SPARQL, you need to define a pattern that matches the statements in the graph.

What is a graph pattern? [25]

• To define graph patterns, we must first define triple patterns:
  – A triple pattern is like an RDF triple, but with the option of a variable in place of RDF terms (i.e., IRIs, literals or blank nodes) in the subject, predicate or object positions.
  – Example:
    <http://example.org/book/book1>  
    – ?title is a variable.

A Simple SPARQL Query Example:

• Data:
  <http://example.org/book/book1>  
  <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial".

• Query:
  SELECT ?title
  WHERE
  { <http://example.org/book/book1>  
    <http://purl.org/dc/elements/1.1/title> ?title . }  

• Result:
  title
  "SPARQL Tutorial"

SPARQL also provides advanced operators (namely SELECT, AND, FILTER, OPTIONAL, and UNION) which can be used in more expressive queries.

3. SPARQL QUERY OPTIMIZATION

3.1 Introduction

The key idea of semantic query optimization is, given a query and a set of integrity constraints, to find minimal (or more efficient) queries that are equivalent to the original query on each database instance that satisfies the constraints. The constraints that are given as input might have been specified by the user, automatically extracted from the underlying database or may be implicitly given by the semantics of RDFS when SPARQL is coupled with an RDFS inference system[9]. There are various techniques proposed for SPARQL query optimization and each of them considers one or other parameter to increase the efficiency of the existing query by rewriting the query into an equivalent query.

3.2 SPARQL query Optimization Categories

The following section reviews the broad categories which have been researched so far with respect to optimizing the SPARQL queries.

3.2.1 Merging multiple join into multiway join

In a MapReduce world, it is known that the join operation requires computationally expensive MapReduce iterations. So in that they adopt traditional multi-way join into MapReduce instead of multiple individual joins. Second, by analyzing a given query, they select a good join-key to avoid unnecessary iterations. Further explaining the scenario by example the following SPARQL query can be considered[2]:

SELECT ?x ?y1 ?y2 ?y3 WHERE {
  ?x rdf:type ub:Professor. // triple pattern (tp) #1
In the above example, a SPARQL Basic Graph Pattern (BGP) contains 5 triple patterns which have a shared variable x. In general, the query requires four two-way joins, as expressed in Figure(a). However, the MR framework can evaluate the query all at once as in the five-way join expressed in Figure(b). As RDF has a fixed simple data model, it is not unusual that a SPARQL query includes several triple patterns sharing the same variable. Hence, MR will be a good choice for a SPARQL query processor.

In [2] they evaluated BGP by two operations namely MR Selection and MR Join. MR selection obtains RDF triples which satisfy at least one triple pattern and MR join merges matched triples into a matched graph. MR join can be performed iteratively when a BGP has two or more shared variables. They gave corresponding algorithms for MR selection and MR join. MR operation is computationally expensive. For this reason, they offer two join-key selection strategies: namely greedy selection and multiple selection.

As a result of this the algorithm reduces the number of join iterations, which has a considerable effect on the overall execution time.

### 3.2.2 Sparql Subqueries Optimization

As it is obvious now that RDF data is diverse and distributed over the web. As a result we need to use distributed SPARQL querying. Distributed SPARQL querying can be achieved by using the same method as applied in data integration systems for relational database. The principal idea is to decompose a user query into a set of constituent subqueries, and to route each subquery to the data sources that are capable of answering it [3].

Every subquery (regardless of type) contributes two factors that affect the cost of the overall query. The first factor, termed delay time, is the amount of time in milliseconds that elapses between the issuing of the subquery and the retrieval of the results. As such, it incorporates several subfactors including the number of endpoints that match the query, the bandwidth of the connection to each endpoint, the CPU load on each endpoint, and the size of the result set. The second factor, termed selectivity, is the number of triples in the result set [3].

With regard to this in [3] they have developed an approach of dmst. A directed minimum spanning tree (DMST) for a root node is a minimum cost set of edges that provides a path from to all other nodes in the graph. In the context of query optimization, such a tree establishes an ordering of patterns which assigns bindings to all variables in the shortest possible time. This ordering can then be extended with any remaining edges (those not represented in the DMST) to give a heuristic solution to the query optimization problem. The main idea of the algorithm is to break any cycles while at the same time increasing the cost of the tree as little as possible. Notice that the algorithm does not produce a solution by building up a tree in an incremental fashion. They have also proposed two algorithms so static query plans are generated using one of two minimum spanning tree algorithms: Edmonds’ algorithm or Prim’s algorithm.

Another interesting scheme for optimizing subqueries is to transform correlated queries into equivalent uncorrelated queries for increasing efficiency of distributed query evaluation. Thus Due to the distributed placement of triple on multiple peers, the processing of correlated queries in these systems may be very expensive in terms of query response time and bandwidth usage. In [4] authors have proposed a way to for such transformation so that the the inner query block contains no variables from the outer query block. Thus, the inner query block needs to be evaluated only once. As a result they applied optimization technique based on the idea of using semijoin, and used transformation algorithms to transform correlated queries to equivalent, uncorrelated ones in order to make the distributed evaluation of nested queries efficient.

SPARQL query over views is often rewritten into an equivalent batch of SPARQL queries for evaluation over the base data. As the semantics of the rewritten queries in the same batch are commonly overlapped, there is much room for sharing computation when executing these rewritten queries. This observation motivates us to revisit the classical problem of multi-query optimization (MQO) in the context of RDF and SPARQL. So in [8] a novel algorithm is proposed to efficiently identify common subqueries with a fine-tuned cost model, partitions input queries into groups and rewrites each group of queries into equivalent queries that are more efficient to evaluate.
3.2.3 Rewriting queries by Transformation rules

The web contains distributed and large RDF datasets. We need to retrieve the RDF data with the help of query processing but with minimal time and cost. The techniques are developed by the database community to optimize the query by transforming the original query to an equivalent but optimized query through some pre-defined transformation rules.

In [1] focus is on the following research issues: (a) determination of the ontology mapping types, which can be used in the context of SPARQL query rewriting, (b) modeling of the mappings between a source ontology and the target ontologies, (c) rewriting of the SPARQL queries posed over a source ontology in terms of the target ontologies. Query rewriting is done by exploiting a predefined set of mappings which is based on the different mapping types. The SPARQL query rewriting process lies in the query's graph pattern rewriting. The rewritten query is produced by replacing the rewritten graph pattern to the initial query's graph pattern. Consequently the rewriting process is independent of the query type (i.e. SELECT, CONSTRUCT, ASK, DESCRIBE) and the SPARQL solution sequence modifiers (i.e. ORDER BY, DISTINCT, REDUCED, LIMIT, OFFSET). Graph pattern operators (AND, UNION, OPTIONAL, FILTER) remain the same during the rewriting process. Variables, literal constants, operators and built-in functions appearing in a FILTER expression, remain also the same.

Example: The SPARQL syntax of the source query is shown below

```sparql
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
SELECT ?name ?author
ORDER BY ?name LIMIT 20
```

Another query rewriting model is based on SQGM (SPARQL query graph model). On top of the SQGM [7] defined transformations rules to simplify and to rewrite a query.

The definition of SQGM:

SPARQL query graph model (SQGM) represents a SPARQL query. It is a tuple (OP, DF, r, dflt, NG) where

- OP denotes the set of all operators necessary to model the query,
- DF denotes the set of all dataflows necessary to model the query,
- r is an operator responsible for generating the result of the query (r 2 OP),
- dflt is an operator providing the default RDF graph of the queried RDF dataset (dflt 2 OP),
- NG is the set of graph operators that provide the named graphs (NG _ OP).

Translating a SPARQL Query to an SQGM:

The process for constructing an SQGM from a SPARQL query is described in Algorithm 1. It takes a query as input and returns the corresponding SQGM. The SPARQL query is given as a tuple (DS, GP, SM, R) where DS is the queried RDF dataset, GP is a graph pattern, SM is a set of solution modifiers, and R is the result

Algorithm 1 Translating a SPARQL query q into an SQGM Q

INPUT: q := (DS, GP, SM, R) – a SPARQL query
OUTPUT: Q := (OP, DF, r, dflt, NG) – an SQGM representing q

1. Generate operators for the RDF dataset DS;
2. Generate operators for the graph pattern GP;
3. Generate operators for the set of solution modifiers SM;
4. Generate operators for the result form R;

In the query rewriting phase, the generated SQGM is transformed into a semantically equivalent one to achieve a better execution strategy when processed by the plan optimizer. For instance, rules may aim at simplifying complexly formulated queries by merging graph patterns, e.g., avoiding join operations, and eliminating redundant or contradicting restrictions.

Two SQGMs q and q0 are semantically equivalent, if the equation

\[ \text{ResultD}(q) = \text{ResultD}(q0) \]

holds for any RDF dataset D, where ResultD(q) denotes the result set of evaluating q on D.

3.2.4 Selectivity Estimation

Selectivity of a triple pattern T, denoted SEL(T), is the fraction of triples satisfying the pattern. Selectivity is fundamental
because it quantifies the size of intermediate result sets of triple patterns. Thus, the overall goal is to find the ordering of query patterns which minimizes the intermediate result sets for each stage during query execution. Selectivity can be calculated either by an exact formula or an estimation which is mostly based on statistics about the underlying data. Because an exact triple pattern selectivity computation basically requires the pattern to be executed, we cannot rely on exact information since this would require as much time to perform the optimization as it is required to execute the query. Thus, we base our optimization model on statistical information about the ontological resources in order to get an estimation of the selectivity for each triple pattern. This allows ranking the patterns according to their estimated selectivity which is expected to reduce the intermediate result set sizes. This may results in considerable performance improvement.

A cost function reflects the selectivity estimation and is used to rank triple patterns in increasing order of selectivity [10]. The cost function returns a value between 0 and 1, thus, it is basically a normalization to [0,1] of the estimated selectivity. Overall cost for a triple pattern as follows:

\[ c(t) = c(s) \cdot c(p) \cdot c(o) \]  

where \( c(t) \) is the overall cost for a triple pattern \( t \) and \( s, p, o \) are respectively the subject, predicate and object of \( t \). Thus, the expected execution cost for \( t \), \( c(t) \), is modeled as the multiplication of the expected cost for the subject \( c(s) \), predicate \( c(p) \), and object \( c(o) \).

Many researches have been carried out based on selective estimation in the field of sparql query optimization. [10] focuses on static query reordering in order to get an execution plan which is optimal according to the selectivity of triple patterns. Whereas another work [11] focus on two common SPARQL graph patterns (star and chain patterns) and propose to use Bayesian network and chain histogram for estimating the selectivity of them. Yet another work [12] propose a way of computing property correlations based on ontology itself in order to improve the execution performance of the SQL translated from SPARQL statement queries. In [12] a number of heuristics for the selectivity estimation of joined triple patterns are used. The heuristics range from simple variable counting techniques to more sophisticated selectivity estimations based on the probabilistic framework that builds on top of tailored summary statistics for RDF data.

4. CONCLUSION

Query results are generated by accessing relevant data from database and manipulating that data as per our needs. Since database structures are complex and the user may fire complex queries including joins the result yielded from database may be obtained in different order and different data structures. Each different way may need different processing time. Some query may differ in processing time from a second to hours depending upon the way we selected. So we need to reduce this processing time for the best utilization of our query retrieval. Same is also true for the Sparql queries which query the RDF data. As a result we need to perform Query optimization to the existing query methods and reduce the processing time as less as possible.

So after reviewing different approaches it can be concluded that Sparql query optimization is a vast research area in semantic web and can be extended in several ways. Also the categories for the research area in sparql query optimization are as described above.

References


