Research on Efficient SPARQL Query Processing for RDF Data

Yi Zhang

College of Computer Science and Technology，

Wuhan University of Science and Technology，Wuhan 430000，China.

zrri@foxmail.com

Keywords: SPARQL, Semantic Search, Hybrid Querying, Graph Data Indexing, Query Optimization.

Abstract. Semantic Web can add explicit semantics to data so that machines can not only simply display these data, but also the machines can understand, process and even integrate them. In recent years, with the comprehensive development of the Linking Open Data (LOD)project and the DBpedia project, Semantic Web data sources increase significantly and a large number of graph data in form of RDF data model are published. The Web is rapidly changing from the one containing Web pages and hyper-links only (called Document Web) to a Data Web with abundant entities and rich relationships between these entities. Based on these semantic data, lots of large search engine companies like Google are building knowledge graph to improve the quality of search, which indicates that we enter into the era of semantic search.

Introduction

Semantic search which is a grand challenge is different from the traditional document retrieval because it has to deal with finer-grained structured semantic data. The well-optimized storage and indexing techniques specified for Web documents are no longer suitable for RDF data. The exiting sort algorithms cannot be directly applied to the semantic search scenarios focusing on entities and their associations[1]. How to support SPARQL query processing and how to handle data integration from the heterogeneous data sources are totally unexplored problems. In addition, how to using the well-known keyword for the end users to query is crucial to the wide adoption of semantic search.

Index structure and Operating variables

Index structure

Mapping the RDF entry to the ID. First, assign a unique ID for each index RDF entry. This operation can save the memory space and cut down the storing complexity which can save time. Information retrieval engine will automatically generate a document number docID for every document. Naturally, virtual document is created for each RDF entry[2]. And then the docID can be used as the ID of this RDF entry. Specifically, first we create a domain ID in the inverted index. Second, for each RDF term t, its description is to be inserted into the list of terms in the domain ID[3]. Finally, docID lexical items are inserted into the document lists corresponding with RDF term t. In order to obtain an RDF ID, we can construct a query (ID, RDFTerm) to the IR engine. RDFTerm is the RDF description (string value). Figure 1 shows the structure of the ID field.

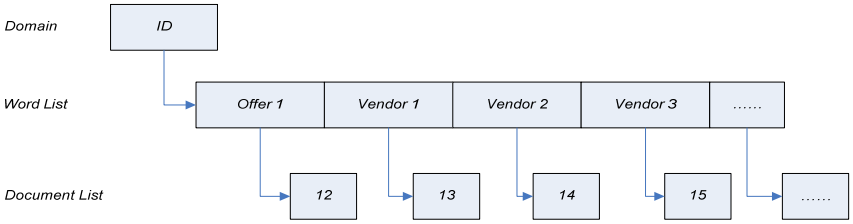

Fig.1 ID and stat domain

Index statistics. Nine categories of statistical information for each RDF items are defined. These values are encoding by 4-bit. A stat field in the inverted index is setup to store the statistical information[4]. For each RDF term, the ID of term is inserted into the terms list of stat domain. Then add a 36-bit binary string into the document appears list of RDF term. In order to obtain the statistical information of RDF term, firstly we should obtain the term ID, and then returns the corresponding document which is the string value of statistical information by querying the item (stat, ID).

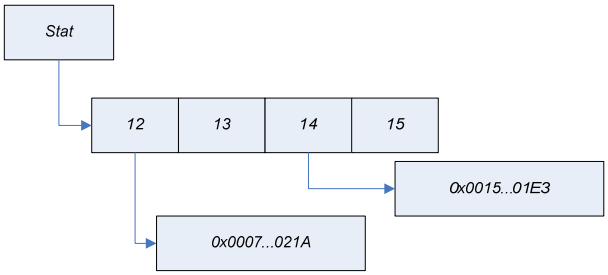

Fig.2 Shows the organizational structure of the domain stat

Index RDF triples. For any RDF triples (term, rdf: type, c), its predicate is rdf: type, which is using the type field to index. Specifically, first the ID of c is inserted into the item list of in the type domain. Then add a document whose doclD equals to the term ID into the document appears list of item c. Given an RDF concept c, we use the item query (type, c) to acquire all of the c-related RDF items[5].

Operating variables

The binding set is to represent the intermediate results and the final output. The binding set $S=\{B_s,V_s\}$ makes up with a binding list B_s and a variable list V_s. A binding $\beta \in B_s$ is a mapping from V_s to the ID of RDF. The binding $\beta(v)$ represents the value of a variable v which belongs to the variable list ($v \in V_s$). U is the triples complete works and t is a triplet pattern. All the variables v appears in t can be replaced by the binding $\beta(v)$ and we can obtain a result $t\{v\}$. If the formula $t\{v\} \in U$ is founded, we can say the binding $\beta(v)$ is the result of the triplet pattern t. In addition, there are two binding sets S_1,S_2. Also, there are two bindings $\beta_1 \in B_{s1},\beta_2 \in B_{s2},\beta_1(v)=\beta_2(v)$,if every variable v belongs to V($v \in V$), the formulas $\beta_1(v)=\beta_2(v)$ can be founded ($V=V_{s1} \cap V_{s2}$).

Table 1 Meaning of the symbols used in this article

Symbols	Meaning
$S=\{B_S,V_S\}$	Binding set
B_S	Binding list
V_S	List of variables
β	Binding
$\beta(v)$	The value of variable v is bound
t	A triple pattern
$t(v)$	For a triple death answers
U	Complete works of triples

Relationships of SPARQL queries and operating variables

We need further clarified their relationships between SPARQL queries and operating variables, also we need to understand how these actions help SPARQL query optimization and process given the SPARQL queries and the above definition for operating variables. Generally, SPARQL provides

a matching method based on graph pattern to access RDF graph[6]. However, for RDF graph the graph matching algorithm is NP-complete. A natural idea for solve this problem is to convert the SPARQL query graph, thus, the result we want is an executing-tree with minimum cost. Here I will introduce how to establish a query tree with equivalent and optimization which is translated from the SPARQL query graph.

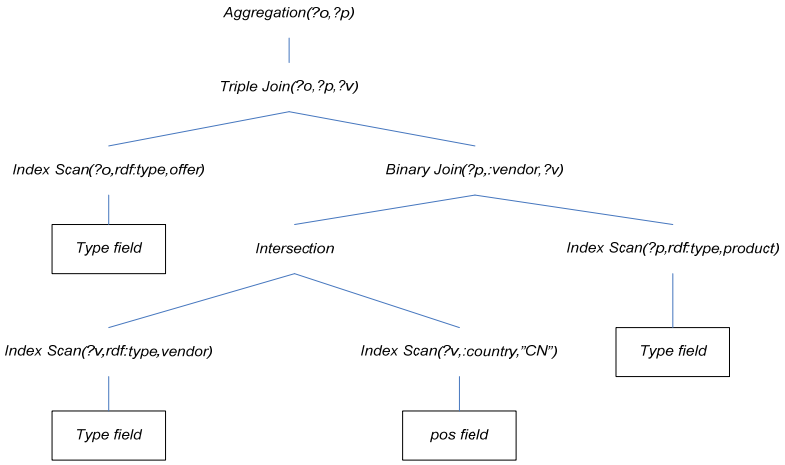

Fig.3 An executing-tree corresponding to the SPARQL query graph

Specifically, the vertex limit (that is, the concept limit or relationship limit) which can be translated into an index scan is corresponding to the specified index field. A standard point v is containing a number of restrictions: its binding is the intersection with all the index scan results[7]. Thus, is used to merge index on the scan results of the same variable v. In other words, each of the restricted standard point is converted to be a tree, where the leaf nodes are index scans, while the inner nodes are the union operation. For example, in Figure 3, two Triples $(?v, rdf:type, vendor)$ and $(?v, country, "CN")$ are translated into the index scan operation on the $rdf:type$ and pos field firstly, and then return a set of binding by the union operation (share the same variable $?v$). A standard line which is connected two-variable nodes can be translated into a binary join. In order to implement $(?p, :vendor, ?v)$, a binary join is used to merge the binding sets with union operation and binding sets with index scans link. Similarly, we implemented a triple join consists of three standard Triple patterns. These points were joined with three triples and a triple point.

There are two Triple patterns $(?x, :p1, ?y)$, $(?x, :p2, ?y)$, may be they have the same variables, but they do not have the same edge limit. On that case, we can use a binary join and a selection operation to perform, rather than use high cost of three binary join. There is an assumption that a binding set S can be obtain by the binary join on $(?x, :p1, ?y)$ with variable $?x$ and the variable $?y$. Then it only needs to scan the binding set S to return triples for whose predicate is equal to the variable $:p2$. In addition, it should be noted that aggregate actions do not correspond to any triple patterns in SPARQL query graph model. However, if we only need a subset of the binding variables, aggregation operations may be used for this purpose. In addition, at the time of query processing for all middle binding sets (contains unused variables for further calculations)[8], we always use the aggregation operation to remove some unused binding sets. For example, in Figure 3, the connection of an aggregate operation is used to return the final binding set which is a triple join set with variables $?p$ and $?o$.

Query Optimization

SPARQL query optimization algorithm in runtime which is also known as ROS algorithm. The goal of the algorithm is optimizing SPARQL order join in the run time, thereby improving the query performance[9]. First of all we discuss the storage scheme of the RDF data. Then the presentation of

SPARQL query plan which is regardless of the order is designed and this presentation is also the input of ROS algorithm for the initial execution plan. The presentation of ROS algorithms for query plan is the connection diagram which is similar to the related database queries. ROS algorithms do not rely on any model or statistical information, but by virtue of estimating the size of the result set it can determine the cost of connection, and then select the minimum cost as the join order. Therefore, you need to determine a method of estimating the operation result sets in the connection diagram[10]. That is the method based on the sampling. In this chapter, we give some basic definitions using in ROS algorithm, including: index structure based on B-tree, the definition of connection diagram and the sampling techniques in ROS.

The basic definitions in the algorithm

Index structure based on B-tree.A prototype implementation of ROS algorithm is built on Sesame platform. Selecting Sesame platform as a background is of all it's open source RDF query engine first, then it uses the B-tree index organizations and more efficient RDF data query.

B-tree is a common data structure that using B-trees can significantly reduce the time of processing the localization records. Therefore, the access speed can be faster and faster. Now, this structure is used widely and it is typically used to the database index[11]. Each node in the B-tree is consisting of following parts: the number of keywords in one node; pointers to parent nodes; keywords; pointers to the child nodes. The number of keywords in each node and the number of child nodes owned in one node have to be restricted by the order of B-tree.

Case 1: If an n-order B-tree, the maximum number of child nodes for each node is n, the number of keywords is 1~n-1; non-root nodes at least n/2 child nodes and the number of the keywords is n/2~n-1.

The following characteristics that belongs to the B-tree:

a. In keywords sets are distributed in the whole tree

b. Any keyword is appearing and just appearing only in one node

c. The query may be end in non-leaf nodes

d. The query performance is equivalent to the binary search among the keywords universal sets

e. Level controlling automatically

There has already been B-tree index spo and pos in Sesame.

Index spo can be understood as the Hash algorithm for s, p, o of triples (s, p, o), which is formatted as a four-byte. As we all know the function of the Hash algorithm is converting a string into bytes. The hash value of technology triples is:

$$s.hashcode() = s.toCharArray().get(i).getAscall() * 31^{l-i-1}$$

The variable i represents the position where the current character in the string s, l represents the length of the string s.

Case 2: The string is "abc".hashcode()=$97*31^2+98*31^1+99*31^0$.The triples index-organized by spo, then the hash values of triples t are:

$$t.hashcode() = s.hashcode() + p.hashcode() + o.hashcode()$$

Similarly, the RDF data index-organized by pos, then the hash values of each triple t are:

$$t.hashcode() = p.hashcode() + 0.hashcode() + s.hashcode()$$

Connection Diagram. Optimization algorithms proposed in this paper can be divided into two steps. First, SPARQL graph patterns can be simplified by the RDF schema information. Second, based on the first step, finding optimal connection order for each iteration query by the connection cost and dynamic programming method can obtain the optimal query plan. In this paper, we use simplified query for schema information and two treaty rules proposed in literature (Yuxin, 2010)[5]:

Rule 1: for graph patterns which is containing the public variable:

$\{?x, rdf:type, C_1.?x, rdf:type, C_2.\cdots\cdots\}$, where C_1 and C_2 are the concepts in RDF, and $C_1 \subseteq C_2$, can be simplified to the $\{?x, rdf:type, C_1.\cdots\cdots\}$;

Rule 2: similar to the graph patterns $\{?x, rdf:type, C_1., ?x, p, o.\cdots\cdots\}$, where p is the property of non-$rdf:type$, can be directly reduced to $\{?x, p, o.\cdots\cdots\}$.

These two rules are intended to guarantee the specialization of TP rather than the generalization in addition; these rules can shorten the query time. But the rules exists assumed premise which is not mentioned in literature (Yuxin, 2010) [5]. The assumption for the rule 1 is the query engine has a certain amount of reasoning capacity. The assumption for the rule 2 is the defined domain and the value domain has been ignored in the triples.

Therefore, for the rule 1, we assume that the query engine with reasoning capacity. For different query engine, the results of the implementation by simplifying with rule 1 are different.

For the rule 2, it is necessary to known the defined domain and the value domain of predicate p a single concept from a set of multiple conceptual. If it is a single concept, you can directly simplify with rule 2, if not, the graph pattern will not be simplified.

In addition, the rcffs is used in RDF: subPropertyof represents the inheritance relationships between properties, and tuples with a child attribute is contained in the tuples with parent attribute[12]. Therefore, some of the TP with attributes inheritance relationships in the graph pattern can be simplified.

We add the third rule in this paper to simplify the graph patterns:

Rule 3: For some graph patterns which are containing the public variables such as:

$\{?x, p_1, ?o., ?y, p_2, ?o.\cdots\cdots\}$, where p_1 is p_2 SubPropertyOf, and the graph patterns can be simplify to the variables: $\{?x, p_1, ?o.\cdots\cdots\}$. The matching results sets of the SubPropertyOf is contained in matching results sets of the parent attribute pattern. The result by intersection of two results sets is equal to the result set of the matching SubPropertyOf.

Sampling Techniques.SPARQL query efficiency is usually connected with the performing mode of join operations, the size of the dataset and the size of the intermediate results sets in join operations. The key for SPARQL graph pattern optimization is to find the optimal query plan that triples model of optimal join order.

Performance Analyses

In this chapter, the RDF query processing method based on LUBM standard test data sets is used to validate the proposed method. RDF query response time under 4 different data sets is tested, and the performance of the query is compared with the existing query algorithms.

Focused on the following groups of different sizes of the LUBM test data sets, data size is as listed in table 2. The RDF triples in table 2 is the result of (Jena, 2002) [13].

Table 2 The size of the test set (unit: Tens of thousands of)

Univ.	1	50	100	300
RDF Triples	10	1000	2000	6000

LUBM queries are tested on different data sets respectively and the query response time is as listed in table 3.

Table 3 The query response time on LUBM (unit: Second)

	Q1	Q2	Q4	Q5	Q6	Q8	Q9
1	0.090	0.101	0.045	0.169	0.131	0.235	0.279
50	0.588	0.690	0.317	1.158	1.104	2.750	6.881
100	1.008	1.510	0.408	1.667	1.909	4.349	8.619
300	3.057	7.772	1.337	5.680	6.713	14.585	45.620

The query response time of 7 typical queries on LUBM on the current algorithm is compared with that of the query plans on ProvBase and H2RDF in this experiment. Parts of this queries on LUBM require the query processing engine with reasoning capacity on the relationships between the sub-Property and the subClassof. However, the current algorithm, do not support the reasoning capacity of RDF data. In order to work effectively, we need to preprocess which is reasoning all the implicit triples. The query response time of 100 universities data sets on LUBM on the three algorithms is as shown in Figure 4.

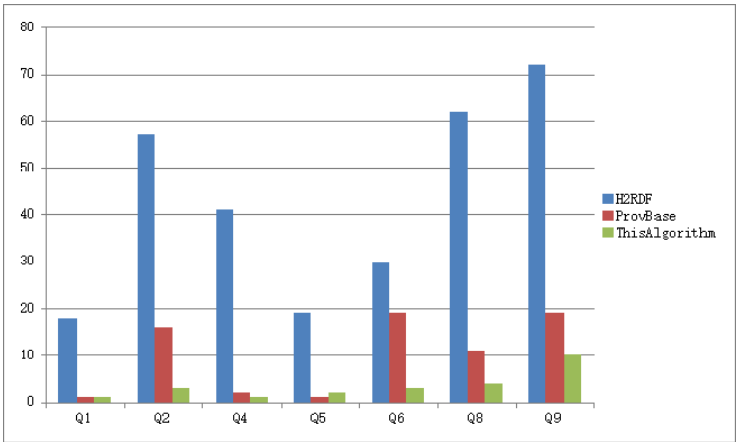

Fig.4 The query response time of 100 universities data sets on LUBM (unit: Second)

The results on Figure 4 show that query performance of these typical queries on ThisAlgorithm is similar or superior obviously to that on Provlpase and H2RDF.

Conclusion

In the paper, a new RDF query engine to processing of SPARQL queries efficiency is supported. The main tasks of the query engine are (1) a set of solution plans to index the triples and a efficiency operating variables to query and process optimal; (2) a set of RDF statistics to estimate execution cost of the query plan, and (3) a main tree optimization algorithm to determine the optimal query plan. Experimental results show that our method is efficient and can be extended to query large sets of RDF triples. An additional query optimization which can expand the initial operating variables is introduced to handle general SPARQL query graph pattern. The future, I plan to further study the FILTER clause and the named graph in SPARQL rules.

References

[1] Lei Zou,M. Tamer Ozsu,Lei Chen,Xuchuan Shen,Ruizhe Huang,Dongyan Zhao. gStore: a graph-based SPARQL query engine[J]. The VLDB Journal,2014,234:.

[2] Faisal Alkhateeb,Jean-FranCois Baget,Jerome Euzenat. Extending SPARQL with regular expression patterns (for querying RDF)[J]. Web Semantics: Science, Services and Agents on the World Wide Web,2009,72:.

[3] Chang Liu,Haofen Wang,Yong Yu,Linhao Xu. Towards Efficient SPARQL Query Processing on RDF Data[J]. Tsinghua Science & Technology,2010,156:.

[4] Groppe, S.; Groppe, J.; Kukulenz, D.; Linnemann, V. A SPARQL Engine for Streaming RDF Data[J]. Signal-Image Technologies and Internet-Based System, 2007. SITIS '07. Third International IEEE Conference on 2007. 167 - 174

[5] Ye Yuxin, Ouyang Dantong. Optimize SPARQL by Combining Semantic Reduction and Selectivity Estimation[J]. Aata Electrorrica Sirrica,2010,38 (5): 1205—1210

[6] Kumar, N.; Kumar, S. Querying RDF and OWL data source using SPARQL[J]. Computing, Communications and Networking Technologies (ICCCNT),2013 Fourth International Conference on 2013. 1 - 6

[7] Letao Qi; Lin, H.T.; Honavar, V. Clustering remote RDF data using SPARQL update queries[J]. Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on 2013. 236 - 242
[8] Leida, M.; Chu, A. Distributed SPARQL Query Answering over RDF Data Streams[J]. Big Data (BigData Congress), 2013 IEEE International Congress on 2013. 369 - 378
[9] Mihok, B.; Stocking, R.; Holmes, D. Augmenting Data Collection and Analysis of Operational Simulations with RDF and SPARQL[J]. Aerospace Conference, 2008 IEEE. 1 - 10
[10] Holst, T.; Hofig, E. Investigating the Relevance of Linked Open Data Sets with SPARQL Queries[J]. Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual. 230 - 235
[11] Tauer, G.; Rudnicki, R.; Sudit, M. Approximate SPARQL for error tolerant queries on the DBpedia knowledge base[J]. Information Fusion (FUSION), 2013 16th International Conference on 2013. 850 - 856
[12] Fang Du; Haoqiong Bian; Yueguo Chen; Xiaoyong Du. Efficient SPARQL Query Evaluation in a Database Cluster[J]. Big Data (BigData Congress), 2013 IEEE International Congress on 2013. 165 – 172
[13] McBride B. Jena: A Semantic Web Toolkit [J]. IEEE Internet Computing, 2002. 6(6):55-59