ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

Efficient Optimization of Sparql Basic Graph Pattern

Ms.M.Manju[1], Mrs. R Gomathi[2]

PG Scholar, Department of CSE, Bannari Amman Institute of Technology, Sathyamangalam, Tamilnadu, India[1]

Associate Professor/Senior grade, Department of CSE, Bannari Amman Institute of Technology, Sathyamangalam, Tamilnadu, India[2]

ABSTRACT— In this paper, we formalize the problem of Basic Graph Pat-tern (BGP) optimization for SPARQL queries and main memory graph implementations of RDF data. We define and analyze the characteristics of heuristics for selectivity based static BGP optimization. The heuristics range from simple triple pattern variable counting to more sophisticated selectivity estimation techniques. Customized summary statistics for RDF data enable the selectivity estimation of joined triple patterns and the development of efficient heuristics. Using the Lehigh University Benchmark (LUBM), we evaluate the performance of the heuristics for the queries provided by the LUBM and discuss some of them in more details.

KEY WORDS— SPARQL, query optimization, selectivity estimation

I. INTRODUCTION

SPARQL is an RDF query language, that is, a query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format. The SPARQL query processor will search for sets of triples that match these four triple patterns, binding the variables in the query to the corresponding parts of each triple.we focus on selectivity-based static Ba-sic Graph Pattern (BGP) optimization for SPARQL queries [and main memory graph implementations of RDF [9] data. In SPARQL, a BGP is a set of triple patterns where a triple pattern is a structure of three components which may be concrete (i.e. bound) or variable (i.e. unbound). The three components which form a triple pattern are respec-tively called the subject, the predicate and the object of a triple pattern. Sets of triple patterns, i.e. Basic Graph Pat-terns, are fundamental to SPARQL queries as they specify the access to the RDF data.

Query optimization is a fundamental and crucial subtask of query execution in database management systems. We focus on static query optimization, i.e. a join order optimization of triple patterns performed before query evaluation. The optimization goal is to find the execution plan which is expected to return the result set fastest without actually executing the query or subparts. This is typically solved by means of heuristics and summaries for statistics about the data.

The problem we are going to tackle in this paper is best explained by a simple example. Consider the BGP displayed in Listing 1 which represents a BGP of a SPARQL query executed over RDF data describing the university domain. Typically, there are a number of different subjects working, teaching, and studying at a university (e.g. staff members, professors, graduate, and undergraduate students). They are all of *type* Person in our RDF dataset. We know that the dataset contains a huge number of RDF resources of type Person among others of type Publication, Course, Room.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

The OWL schema ontology used to describe the vocabulary for the RDF dataset states that the property for the social security number is inverse functional. Therefore, the object of the property uniquely determines the subject. Hence, the second triple pattern in our BGP of Listing 1 matches only one subject with the social security number ”555-05-7880”. Our schema ontology specifies further that the domain of the social security number property is a class of type Person. Therefore, we can state that the subject with social security number ”555-05-7880” is of type Person (or our data is inconsistent).

The focus in our work is on main memory graph implementations of RDF data (i.e. in-memory models). Currently most RDF toolkits support both in-memory and on-disk models. Relational database management systems (RDBMS) are commonly used as persistent triple stores for on-disk models. Because of the fundamentally different architectures of in-memory and on-disk models, the considerations regarding query optimization are very different. Whereas query engines for in-memory models are native and, thus, require native optimization techniques, for triple stores with RDBMS back-end, SPARQL queries are translated into SQL queries which are optimized by the RDBMS. It is not our goal in this paper to analyze optimization techniques for on-disk models and, hence, we are not going to compare in-memory and on-disk models. Furthermore, we focus on the evaluation of the presented optimization techniques without comparing the figures with the performance of alternative implementations. A comparison of implementations requires a comprehensive study that goes beyond the scope of this pa-per. In fact, the query performance of query engines is not just affected by static query optimization techniques but, for instance, also by the design of index structures or the accuracy of statistical information. Finally, our focus is on static query optimization techniques. Hence, we do not discuss optimal index structures for RDF triple stores, neither in-memory nor on-disk, as this too is a research topic that goes beyond the scope of this paper.

Listing 1: Example BGP

```
?x rdf:type uv:Person .
?x uv:hasSocialSecurityNumber ”555−05−7880
```

Our focus on main memory graph implementations, i.e. in-memory models, has an important limitation: scaling. In-deed, the few gigabytes of main memory clearly limit the size of RDF data which may be processed in main memory. Therefore, we might question the relevance of studying optimization techniques for RDF in-memory models. We argue, that in-memory models are important for a number of reasons. First, optimized queries on in-memory models run much faster than on-disk. Second, 64-bit architectures pose virtually no more limits to the theoretical amount of main memory in computers. Third, in a cluster, distributed in-memory models could be used for parallel query evaluation. Finally, optimization techniques and customized summary statistics of RDF data are important for native RDF persistent stores as they do not rely on relational database technology and, hence, require a native optimizer.

II. RELATED WORK

The execution time of queries is heavily influenced by the number of joins necessary to find the results of the query. Therefore, the goal of query optimization is (among other things) to reduce the number of joins required to evaluate a query. Such optimizations typically focus on histogram-based selectivity estimation of query conditions.

Piatetsky *et al.* introduce in the concept of selectivity estimation of a condition. In Selinger *et al.* present the System R optimizer, a dynamic programming algorithm for the optimization of joins. Likewise, POSTGRES implements an exhaustive search optimization algorithm. In con-trast, INGRES introduced an optimization technique based on query decomposition. Estimation of conditions are often supported by histogram distributions of attribute values. More recently, developments in deductive and ob-ject oriented database technology showed the need for more cost-effective optimization techniques as the traditional techniques work well for queries with only a few relations to join. Steinbrunn *et al.* summarizes and analyzes in randomized algorithms for the problem of query optimization where the overall goal is to search the solution space for the global minima moving randomly between connected solutions according to certain rules. Further, the authors de-scribe deterministic, genetic and hybrid algorithms as techniques for the problem of cost effective query optimization. PostgreSQL is and example of an open source databases system experimenting with genetic algorithms for query optimization.

Related to the Semantic Web, Pérez *et al.* analyze in the semantics and complexity of SPARQL. Harth *et al.* investigate the usage of optimized index structures for RDF. The authors argue that common RDF infrastructures do not support specialized RDF index structures. The index proposed by the authors supports partial keys and allows selectivity computation for single triple patterns. Hartig *et al.* [8] present a SPARQL query graph model (SQGM) which supports all phases of query processing, especially query optimization. The authors refer to a discussion on the Jena mailing list which showed that a simple rearrangement of a SPARQL query leads to an improvement of factor 220.

III. PROPOSED SYSTEM

BGP Abstraction. As discussed in Section 3.1, we abstract a BGP as an undirected graph B which is characterized by the connected components g ∈ G, where each g is an ordered pair g = (N, E) consisting of a set N of triple patterns (i.e. the nodes of g) and a set E of triple pattern pairs (i.e. joined triple patterns/edges of g). The connected graph g ∈ G represents a subset of (transitively) joined triple patterns of B. In the following we describe the algorithm for the optimization of g = (N, E).

Based on the BGP abstraction for g ∈ G, we perform a variation of the deterministic minimum selectivity approach to identify the execution plan pg which is optimal according to the algorithm and the selectivity estimations. The optimization algorithm constructs a solution in a deterministic manner applying a heuristic search. Eventually, the algorithm identifies an order for the elements of the set N (i.e. triple patterns). Note that this is not a total order on N. The triple patterns in the resulting execution plan are not necessarily ranked by estimated selectivity.

Optimization Algorithm. In Algorithm 1, we provide the pseudo-code for the core optimization algorithm. The algorithm first selects the edge with minimum estimated selectivity from *g* = (*N*, *E*). The corresponding nodes are marked as visited and added to the final execution plan *pg* ordered by estimated selectivity, i.e. the more selective node is added first to the execution plan. After selecting the first edge *e* ∈ *E*, the core optimization algorithm iteratively selects the edge which satisfies the two properties (1) minimum estimated selectivity and (2) visited node. With each iteration a new node is added to the final execution plan. The property of minimum estimated selectivity is motivated in the deterministic minimum selectivity optimization approach according to which good solutions are generally characterized by selective intermediate results. The second property, i.e. visited node, ensures the iterative selection of a triple pattern, i.e. a node *n* ∈ *N*, which joins with the previous partial execution plan. This is an important characteristic of good execution plans as result sets will never

Algorithm 1 Find optimized execution plan EP for g ∈ G

```
N ← Nodes(g)
E ← Edges(g)
```

```
EP[size(N)]
e ← SelectEdgeMinSel(E)
EP ← OrderNodesBySel(e)
while size(EP) ≤ size(N) do
e ← SelectEdgeMinSelV isitedNode(EP,E)
EP ← SelectNotV isitedNode(EP, e)
end while
return EP
```

Figure 1: Optimized DAG d1 ∈ D with highlighted
node with only outgoing directed edges

Therefore, at each stage of query processing the intermediate result sets are iteratively constrained. The algorithm terminates when all nodes $n \in N$ have been visited and the optimal execution plan *pg*, i.e. a well defined order for the elements of *N*, is returned as a result. Directed acyclic graphs of execution plans which satisfy the second property, i.e. visited node, of the edge selection process for BGP abstractions described above, feature a common characteristic: there is only one node that has only outgoing directed edges, i.e. the node which is executed first in the execution plan. Nodes with only outgoing directed edges do not join with the previous partial execution plan and, hence, result in a Cartesian product of two intermediate result sets. For instance, the execution plan which executes the triple patterns of Listing 2 top-down, abstracted as DAG creates two Cartesian products for the intermediate result sets of the first three triple patterns (highlighted by the three nodes labeled 1, 2, and 3 which are nodes with only outgoing directed edges). In contrast, the optimized execution plan, abstracted as DAG in does never create Cartesian products of intermediate result sets. This is highlighted by the DAG in Figure 3 with one node with only outgoing directed edges (i.e. node 5). This node represents the first triple pattern in the optimized execution plan for the BGP in Listing 2.

Selectivity Estimation Heuristics. In order to decide the selection of edges during the optimization process, the core optimization algorithm requires figures about the selectivity of graph patterns. The extensible pool of selectivity estimation heuristics is the component intended to provide the required selectivity figures to the core optimizer. Heuristics are used to weight the nodes and edges of a BGP abstraction. Given a weighted connected graph $g \in G$ the core optimization algorithm is able to proceed with the iterative selection of nodes based on the deterministic
minimum selectivity optimization approach described above.

IV. RESULT AND DISCUSSION

The Lehigh University Benchmark (LUBM) is developed to facilitate the evaluation of Semantic Web repositories in a standard and systematic way. The LUBM data set is a benchmark data set designed to enable researchers to evaluate a semantic web repository's performance . The LUBM data generator generates data in RDF/XML serialization format. Therefore, we convert the data 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 .

V. CONCLUSION

The paper summarizes the research we have been doing on static Basic Graph Pattern (BGP) optimization based on selectivity estimation for main memory graph implementations of RDF data.

We formalized the problem of BGP optimization and we presented the architecture for the optimizer that has been implemented for ARQ. Further, we discussed a number of heuristics for the selectivity estimation of joined triple patterns. The heuristics range from simple variable counting techniques to more sophisticated selectivity estimations based on the probabilistic bound predicates should not be considered as joins. Framework that builds on top of tailored summary statistics for RDF data.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

As the evaluation clearly showed, the characteristics of the heuristics greatly influence the selected ordering of the triple patterns of a BGP and, hence, the query execution performance. In our experience, we found the following properties of heuristics to be important for the problem of BGP optimization. First, the optimizer should avoid Cartesian products as intermediate result sets. Second, the selectivity should not be limited in lower bound estimation. Third, the selectivity of joined triple patterns should be a function of the estimated selectivity of the join (i.e. the size of the result set) and the selectivity of the more selective triple pattern involved in the join. Finally, as we noticed multiple times, bound predicates should not be considered as joins.

REFERENCES

[1] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, W3C, 2004.

[2] D. Beckett and B. McBride. RDF/XML Syntax Specification. Technical report, W3C, 2004.

[3] A. Bernstein, C. Kiefer, and M. Stocker. OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation. Technical Report ifi-2007.03, University of Zurich, Department of Informatics, Winterthurerstrasse 190, 8057 Zurich, Switzerland, March 2007.

[4]A. Bernstein, M. Stocker, and C. Kiefer. SPARQL Query Optimization Using Selectivity Estimation. In Poster Proceedings of the 6th International Semantic Web Conference (ISWC), Lecture Notes in Computer Science. Springer, 2007.

[5] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. Technical report, HP Laboratories, 2003.

[6] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics: Science, Services and Agents on the World Wide Web, 3(2–3):158–182, 2005.

[7] A. Harth and S. Decker. Optimized Index Structures for Querying RDF from the Web. In Proc. of the 3rd Latin American Web Congress, page 71, 2005.

[8] O. Hartig and R. Heese. The SPARQL Query Graph Model for Query Optimization. In Proc. of the 4th European Semantic Web Conf., 2007.

[9] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C, 2004.

[10] B. J. Oommen and L. Rueda. The Efficiency of Modern-Day Histogram-Like Techniques for Query Optimization. *The Computer Journal*, 45(2):494–510, 2002.

[11] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *Proc. of the 5th Int. Semantic Web Conf.*, pages 30–43, 2006.

[12] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the Number of Tuples Satisfying a Condition. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 256–276, 1984.

[13] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In The VLDB Journal, pages 486–495, 1997.

[14] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2008.

[15] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 23–34, 1979.

[16] M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. *VLDB Journal: Very Large Data Bases*, 6(3):191–208, 1997.