ARQo: The Architecture for an ARQ Static Query Optimizer

Markus Stocker, Andy Seaborne
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2007-92
June 26, 2007*

semantic web, SPARQL, query optimization

In this paper we describe the architecture of ARQo, a first approach for SPARQL static query optimization in ARQ. Specifically, we focus on static optimization of *BasicGraphPattern* (BGP) for *in-memory* models. Static query optimization is intended as a query rewriting process where the set of triple patterns defined for a BGP are rewritten according to a specific order.

We propose a rewriting process according to the estimated execution cost of joined triple patterns in increasing order. Specifically, the estimated execution cost is a function of multiple parameters such as the estimated selectivity of joined triple patterns, the availability of indexes or pre-calculated result sets.

* Internal Accession Date Only
Approved for External Publication

ARQo: The Architecture for an ARQ Static Query Optimizer

Markus Stocker and Andy Seaborne*

HP Labs Bristol
{firstname.lastname}@hp.com

Abstract. In this paper we describe the architecture of ARQo, a first approach for SPARQL static query optimization in ARQ. Specifically, we focus on static optimization of *BasicGraphPattern* (BGP) for *in-memory* models. Static query optimization is intended as a query rewriting process where the set of triple patterns defined for a BGP are rewritten according to a specific order.
We propose a rewriting process according to the estimated execution cost of joined triple patterns in increasing order. Specifically, the estimated execution cost is a function of multiple parameters such as the estimated selectivity of joined triple patterns, the availability of indexes or precalculated result sets.

1 Introduction

Query optimization is an on-going research area for Relational Database Management Systems (RDBMS). Generally, database systems which provide a declarative language to describe *what* a query should return, require advanced optimization techniques to enable query evaluation within user friendly time frames. Modern database systems implement a number of optimization techniques on different abstraction levels. For instance, the evaluation of Query Execution Plans (QEP) is a fundamental optimization and sophisticated techniques greatly affect the query execution performance.

In this paper we present and discuss ARQo, a static query optimizer for ARQ[1]. Specifically, we focus on static optimization of SPARQL [7] *BasicGraphPatterns* (BGP) for *in-memory* models. While SPARQL query engines such as ARQ and Sesame[2] generally implement index structures for in-memory models to allow fast retrieval of triples, they still lack advanced static (*i.e.* query rewriting) optimization techniques to optimize the query at the QEP level.

ARQo is a first proposal to address the problem of SPARQL static query optimization in ARQ. The goal is to find the QEP which minimizes the execution costs. Thereby, execution costs are a function of multiple parameters which may be arbitrary combined depending whether or not the required information is

* Contact in case of questions

[1] http://jena.sourceforge.net/ARQ/

[2] http://www.openrdf.org/

available within the query execution context. For instance, index structures or pre-calculated result sets are examples of such parameters.

ARQo implements an extensible architecture for QEP cost estimation. Essentially, the framework enables the implementation of cost estimation techniques and implements a generic strategy to search for the most promising QEP, the QEP which is expected to evaluate the query fastest. Therefore, we formalize the concept of *BasicGraphPattern* (BGP), QEP plan space and QEP to meet the needs of the architecture.

Moreover, we extend the framework with cost estimation techniques which may be used within different ARQ deployments depending on the specific settings. Specifically, we intend to develop (1) a generic technique based on variable counting which is especially convenient if specialized information about the underlying ontology is unavailable. Further, we intend to implement (2) a more sophisticated technique based on the concept of joined triple pattern selectivity introduced by Bernstein *et al.* [1]. We intend to implement the approaches of Selectivity Estimation Index (SEI) and Query Pattern Index (QPI) for the estimation of joined triple pattern selectivity which the authors describe in [1].

The remainder of this paper is structured as follows. In Section 2, we briefly review some related work. In Section 3, we formalize the concepts required for the ARQo architecture, especially the BGP, the QEP and the plan space. In Section 4, we describe the proposed architecture in more details, focusing on the selected approach to identify the most promising QEP. Finally, in Section 5, we present and discuss the QEP cost estimation extensions we intend to provide.

2 Related Work

Static query optimization is an on-going research field especially for relational database management systems. Years of research have enabled modern database systems to powerful and fine tuned infrastructures which allow interactive querying for most applications.

In the literature (IBM System R [8], INGRES [9] POSTGRES[3]) the problem of finding the best QEP is mainly solved using two different approaches, query decomposition or exhaustive search. Query decomposition is a heuristic greedy algorithm that proceeds in a stepwise fashion. The major drawback of this approach is that potentially good plans are ignored by the algorithm. An exhaustive search of the plan space identifies every join combination, *i.e.* all plans for processing two-way joins are considered. The drawback of this approach is the required time for planning, which is bigger than for the query decomposition approach since the algorithm has to scan a potentially huge plan space. The benefit of using an exhaustive search approach is that good plans are not overlooked.

IBM System R [8] is probably the most prominent and the first example of a relational database system which considers static query optimization techniques.

[3] `http://s2k-ftp.cs.berkeley.edu:8000/postgres/papers/UCB-MS-zfong.pdf`

Listing 1.1. Example BasicGraphPattern (LUBM Query 2)

```
1 ?X rdf:type ub:GraduateStudent .
2 ?Y rdf:type ub:University .
3 ?Z rdf:type ub:Department .
4 ?X ub:memberOf ?Z .
5 ?Z ub:subOrganizationOf ?Y .
6 ?X ub:undergraduateDegreeFrom ?Y .
```

The static optimizer of System R implements an exhaustive search of the plan space where the cost of a plan is calculated based on the required disk accesses and, thus, on the selectivity of conditions [6].

The Sesame open source RDF framework[4] uses some general query optimization techniques based on query rewriting according to the specificity of triple patterns. Essentially, the specificity is a function of the number and type of variables defined for the subject, the predicate and the object of a triple pattern. The query planner selects the triple patterns according to their decreasing specificity.

Harth *et al.* propose in [4] an optimized index structure for RDF [2] which is extended with statistical information about the data set in the form of occurrence counts of access patterns. This allows efficient lookup for the result set size of an access pattern and, thus, static query optimization according to the selectivity of access patterns. In our approach, we provide selectivity information about joined patterns, which is fundamental since the selectivity of two patterns may be low when considered independently but the joined pattern may be highly selective.

3 ARQo Formalization

In this section we formalize the concepts required for ARQo and the goal of SPARQL static query optimization of *BasicGraphPattern* (BGP) for *in-memory* models. To clarify the idea, we provide some figures based on the BGP of Listing 1.1 (*i.e.* the `WHERE` clause of the Lehigh University Benchmark (LUBM) [3]).

Given a BGP, B, we define B to be a graph G which is described by the set $\mathcal{G}$ of undirected connected graphs. The elements $g \in \mathcal{G}$ are the components of G. For each pair $g_i, g_j \in \mathcal{G}$, g_i, g_j are disconnected.

For SPARQL, an undirected connected graph $g \in \mathcal{G}$ is an ordered pair $g := (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set, whose elements are triple patterns (*i.e.* the nodes of g) and $\mathcal{E}$ is a set of unordered pairs of distinct triple patterns which are joined by a common variable (*i.e.* the edges of g).

In Figure 1, we display the undirected connected graph $g_1 \in \mathcal{G}$ for the BGP of Listing 1.1. Since the triple patterns of the BGP are all joined together, $\mathcal{G}$ contains only one element, *i.e.* the connected graph g_1. Note, that the numbering

[4] `http://www.openrdf.org`

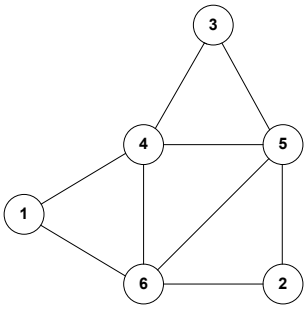

Fig. 1. Undirected connected graph $g_1 \in \mathcal{G}$

used for the nodes of g_1 corresponds to the numbering of the triple patterns of the BGP in Listing 1.1. For any two nodes $(i, j) \in g_1$, there is a path between i and j, if the triple patterns corresponding to i and j are joined together by one or more variables.

In SPARQL, the execution order of pairwise disconnected graphs $g \in \mathcal{G}$ is arbitrary, since the overall result set corresponds to the Cartesian product of the result sets for any $g \in \mathcal{G}$. Thus, the problem of statically optimizing B is a sub-problem of optimizing each $g \in \mathcal{G}$.

Definition 1. *The size of $g \in \mathcal{G}$ is the number of nodes of g.*

Definition 2. *A QEP for $g \in \mathcal{G}$ is an ordered set of the nodes of g.*

Definition 3. *For $g \in \mathcal{G}$ we define the set $\mathcal{P}$ as the QEP space of g. The size of $\mathcal{P}$ is the total number of QEPs $p \in \mathcal{P}$, i.e. the number of ordered permutations of the nodes of g.*

Given the size N of $g \in \mathcal{G}$, the size of $\mathcal{P}$ is $N!$ (for an uniprocessor machine). Thus, even for simple SPARQL queries with a few joined triple patterns, the expanded QEP space $\mathcal{P}$ is huge. Therefore, we need an efficient way how to identify the most promising QEP.

A QEP for B is described by an unordered set, $\mathcal{Q}$, whose elements $p \in \mathcal{P}$ are QEPs. The size of $\mathcal{Q}$ equals the size of $\mathcal{G}$. Thus, the QEP for B is described by the QEP of each component of G.

Therefore, we state the optimization problem as the search for the most promising QEP p within the plan space $\mathcal{P}$ for each undirected connected graph $g \in \mathcal{G}$, building, hence, the unordered set $\mathcal{Q}$ of QEPs for the components of G, where G is the graph which reflects the BGP B.

Note, that a QEP $p \in \mathcal{P}$ can be represented as a Directed Acyclic Graph (DAG). We define $\mathcal{D}$ as the set of DAGs d for a $g \in \mathcal{G}$. For any $d \in \mathcal{D}$, the unordered set $\mathcal{E}$ of joined triple patterns (*i.e.* edges) of $g \in \mathcal{G}$ is transformed to a set $\mathcal{A}$ of directed edges (*i.e.* arcs).

In Figure 2, we show the DAG corresponding to the QEP which executes the BGP of Listing 1.1 top-down. For any two nodes $(i, j) \in d_1$, there is a directed

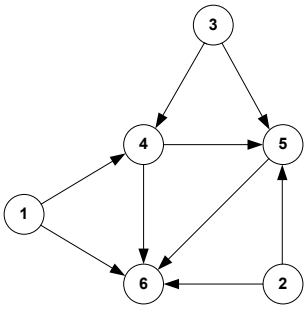

Fig. 2. DAG $d_1 \in \mathcal{D}$

path between i and j, if the triple patterns corresponding to i and j are joined together by one or more variables and i is executed first.

Hence, there is a clear relationship between the set $\mathcal{P}$ of QEPs and the set $\mathcal{D}$ of DAGs. More precisely, we can state the following function,

$$f : QEP \rightarrow DAG \tag{1}$$

which is injective, not surjective and not bijective. Thus, a QEP can be uniquely mapped to a DAG, but a DAG is an abstraction for one or more QEPs. Following Equation 1, a QEP may be described as a materialization of an undirected connected graph $g \in \mathcal{G}$ in the form of a directed acyclic graph $d \in \mathcal{D}$.

4 ARQo Architecture

In this section we concisely describe the architecture of ARQo. Essentially, ARQo is a plug-in module for ARQ with three main components: (1) the optimization core, a (2) heuristic broker and an (3) extensible pool of heuristic techniques.

4.1 The Optimization Core

The optimization core implements the main technique used to statically optimize SPARQL Basic Graph Patterns (BGP). As described in Section 3, we intend to abstract a BGP as a graph G, where G is a set $\mathcal{G}$ of undirected connected graphs, *i.e.* the components of G.

We, hence, optimize each $g \in \mathcal{G}$, *i.e.* we search for the most promising materialization of g in the form of a DAG. This task is mainly solved by the following steps. Given a $g \in \mathcal{G}$, we (1) provide g with estimated execution costs for the nodes and edges. Thereby, costs are estimated by heuristics which are implemented in ARQo and are extensible for user specific purposes. Then, we (2) identify the Minimum Spanning Tree (MST) of g by means of an algorithm which

returns an ordered set $\mathcal{T}$ of joined triple patterns (*i.e.* edges). For instance, the Kruskal MST algorithm [5] identifies the edges of the MST in increasing weight.

The elements $t \in \mathcal{T}$ are ordered according to the total cost for each element t, *i.e.* the edges of g. The total cost of t is calculated as a function of the costs of the edge and the related nodes. Thus, we define the partial order relation $(\mathcal{T}, \leq)$ with the properties of reflexivity, antisymmetry and transitivity. Moreover, we define an order relation $(\mathcal{N}, \leq)$ for the elements $t \in \mathcal{T}$, where $\mathcal{N}$ is the set of nodes for a specific $t \in \mathcal{T}$. Thus, we get a set $\mathcal{T}$ with edges t which are ordered by increasing estimated total cost, where each $t \in \mathcal{T}$ is a set $\mathcal{N}$ of the nodes of t. The set $\mathcal{N}$ is a pair of nodes ordered by increasing estimated costs. Finally, we (3) extract the nodes from $\mathcal{T}$ to a set $\mathcal{Q}$ by selecting the nodes according to the two order relations $(\mathcal{T}, \leq)$ and $(\mathcal{N}, \leq)$. The set $\mathcal{Q}$ features a list of distinct nodes, *i.e.* triple patterns and reflects the most promising QEP. Note that the elements $q \in \mathcal{Q}$, *i.e.* the nodes of g, represent a specific QEP and, thus, define a DAG. We may describe $\mathcal{Q}$ as a materialization of g in the form of the DAG specified by the elements $q \in \mathcal{Q}$. In fact, the order relation of $\mathcal{Q}$, which is specified by $(\mathcal{T}, \leq)$ and $(\mathcal{N}, \leq)$, allows to construct the corresponding DAG by selecting the head and tail nodes $q \in \mathcal{Q}$ according to the order relation of $\mathcal{Q}$ and the join relationship between node pairings.

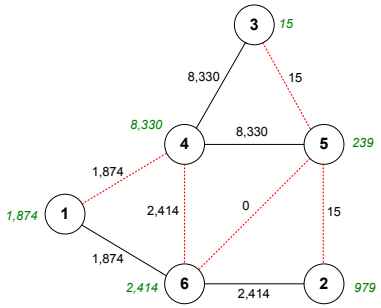

Fig. 3. MST for the undirected connected graph $g_1 \in \mathcal{G}$ of Listing 1.1

In Figure 3, we show the undirected connected graph $g_1 \in \mathcal{G}$ for the BGP of Listing 1.1. For each node and edge, we provide the selectivities of the corresponding (joined) triple pattern(s). Further, we highlight the MST. Based on this solution, we may build the ordered set $\mathcal{T}$ of edges with the elements,

$$\mathcal{T} = \{(5,6), (5,3), (5,2), (1,4), (6,4)\}$$

Finally, the ordered set $\mathcal{Q}$ of distinct nodes which reflects the most promising QEP contains the elements,

$$\mathcal{Q} = \{5,6,3,2,1,4\}$$

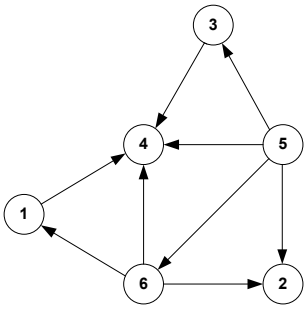

Fig. 4. DAG of the most promising QEP

Listing 1.2. Optimized BasicGraphPattern (LUBM Query 2)

```
?Z ub:subOrganizationOf ?Y .
?X ub:undergraduateDegreeFrom ?Y .
?Z rdf:type ub:Department .
?Y rdf:type ub:University .
?X rdf:type ub:GraduateStudent .
?X ub:memberOf ?Z .
```

In Figure 4 we display the DAG which corresponds to the most promising QEP and reflects the selected sequence of distinct nodes (*i.e.* triple patterns) of the set $\mathcal{Q}$, whereas in Listing 1.2 we list the reordered optimized BGP.

4.2 The Heuristic Broker

The heuristic broker is a component which selects the best heuristic cost estimation for joined triple pattern according to specific settings. Such settings may be the availability of specialized statistical information about the underlying ontology, cached result sets for joined triple patterns or specialized access paths for triple pattern evaluation. The main goal of the broker is to provide the optimization core with the best available cost estimation heuristic according to the settings, or with the heuristic specifically selected by the user.

4.3 The Extensible Pool of Heuristic Techniques

The extensible pool of heuristic techniques implements a number of different heuristics to estimate the execution cost of (joined) triple patterns. Such heuristics are required by the optimization core to select the most promising QEP. In fact, they provide the estimated execution costs for the nodes and edges of each $g \in \mathcal{G}$. We implement the third component as a typical framework which allows to extend specific, user-defined, techniques. In Section 5, we describe in more details the techniques we intend to provide out-of-the-box.

5 ARQo Extensions

In this section, we briefly introduce the two main heuristics for (joined) triple pattern cost estimation we intend to develop for ARQo.

5.1 Variable Counting

The first heuristic is a simple variable counting technique. The goal is to provide a simple optimization technique also if specialized statistical information about the underlying ontology is not available to ARQo. This heuristic will provide the optimization core with a constant cost of 1.0 for edges. However, the cost of a node will be a function of the number and type of variables defined in the corresponding triple pattern. In fact, a higher execution cost will be assigned to triple patterns with more variables. Moreover, a variable subject leads to a higher execution cost as a variable predicate or object.

5.2 Selectivity Estimation

A second heuristic is an implementation of the Selectivity Estimation Index (SEI) and the Query Pattern Index (QPI) first introduced by Bernstein *et al.* and described in [1]. The SEI is used to estimate the cost execution of triple patterns, *i.e.* nodes, whereas the QPI is used to estimate the cost of joins, *i.e.* edges. Based on specialized statistical indexes about the underlying ontology, the SEI and QPI are expected to result into improved SPARQL static query optimization (as illustrated by the evaluation performed by Bernstein *et al.* in [1]).

Next, we concisely describe the SEI and the QPI and present an approach how both may be combined to estimate more accurately the selectivity of joined triple patterns defined in SPARQL queries.

SEI: The Selectivity Estimation Index. The SEI is used to estimate the selectivity [8] of a triple pattern. Given a triple pattern T, the formula

$$sel(T) = sel(S) \times sel(P) \times sel(O)$$

estimates the selectivity of T. Thus, the selectivity of a triple pattern is calculated as the multiplication of the selectivity for the building blocks of a triple pattern, *i.e.* the subject S, the predicate P and the object O. We refer to Bernstein *et al.* [1] for a detailed description of the SEI technique.

QPI: The Query Pattern Index. The QPI provides information about the selectivity of joined triple patterns, more precisely, the selectivity of two joined triple patterns. The ontology schema is used to automatically generate the QPI.

In fact, the `rdfs:domain` and `rdfs:range` properties of `rdf:Property` instances, allow to identify pairs of `rdf:Property` instances which can be joined because of a matching class for the `rdfs:domain` and `rdfs:range` properties.

For a detailed discussion about the QPI technique we refer to Bernstein *et al.*[5] [1].

Combining the SEI and the QPI. The QPI is essentially a list of two joined triple patterns with variables defined for the subject and the object[6]. However, SPARQL queries do not always define variables for the subject and the object of joined triple patterns. Thus, the QPI enables the retrieval of the selectivity for joined patterns which are an upper bound for the selectivity of the pattern defined in the query. For instance, consider the following SPARQL query (we list only the WHERE clause)

```
?x rdf:type ub:GraduateStudent .
?x ub:undergraduateDegreeFrom ?y .
```

The QPI allows a lookup for the selectivity of

```
?x rdf:type ?y .
?x ub:undergraduateDegreeFrom ?z .
```

which only approximates the real selectivity of the joined pattern defined in the SPARQL query. In fact, the QPI selectivity of the pattern is an upper bound for the selectivity of any possible query pattern which can be built for the two joined patterns. Thus, we know that the selectivity of a SPARQL query pattern is potentially lower.

We intend to use the SEI selectivity estimation to estimate the selectivity for a given subject and object. The QPI selectivity may be reduced using the SEI selectivity estimation by a specific formula.

5.3 Other Heuristics

Of course we may think of many other cost estimation heuristic. For instance, we could extend ARQ with a cache for the selectivity of executed query patterns. This incremental cache could be exploited by a corresponding heuristic which considers the cached selectivities while estimating the cost of (joined) triple patterns. As another example, we could extend ARQ with a cache for result sets of query patterns and implement a heuristic which assigns lower execution costs to triple patterns whose result sets are cached. As an example for a user specific setting, if someone provides a simple index about the occurrences of distinct predicates, we may implement a simple heuristic which considers this specific index to estimate the execution cost of triple patterns.

[5] The QPI technique is not yet published. However, the authors have submitted both techniques as contribution to the ISWC2007.

[6] Note, that the variables are of the set (?x, ?y, ?z), *i.e.* the joined triple pattern defines a common variable where the class type matches for the `rdfs:domain` and `rdfs:range` properties

6 Future Work and Conclusions

As future work, we intend to implement the proposed ARQo architecture with its three components based on the description provided in this paper. We will investigate different cost estimation techniques which allow QEP evaluation also for specialized settings which may include caching of result sets or learning systems which incrementally store selectivity information of query patterns. Finally, we intend to extensively evaluate the performance of ARQ extended with ARQo on different data sets and retrieval tasks.

We believe, ARQo will provide ARQ a robust static query optimization module for basic graph patterns and in-memory models which is extensible and adaptable to specific settings but will lead to improved performance also for generic ARQ deployments.

References

1. Abraham Bernstein, Christoph Kiefer, and Markus Stocker. OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation. Technical Report ifi-2007.03, Department of Informatics, University of Zurich, 2007.
2. Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema. Technical report, W3C, 2004.
3. Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
4. Andreas Harth and Stefan Decker. Optimized index structures for querying rdf from the web. In *3rd Latin American Web Congress*, 2005.
5. Jr. Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.
6. Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 256–276, New York, NY, USA, 1984. ACM Press.
7. Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. Technical report, W3C, 2007.
8. P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, New York, NY, USA, 1979. ACM Press.
9. Eugene Wong and Karel Youssefi. Decomposition – a strategy for query processing. *ACM Trans. Database Syst.*, 1(3):223–241, 1976.