Evaluating SPARQL Subqueries Over P2P Overlay Networks

Liaquat Ali, Georg Lausen
University of Freiburg
Email: {ali, lausen}@informatik.uni-freiburg.de

***Abstract*—The introduction of subqueries is one of the most interesting feature included in the latest SPARQL 1.1 specification. Existing distributed RDF storage and querying systems have not studied the evaluation of this newly included query feature. The evaluation of subqueries in distributed environment may be very inefficient and expensive in term of query response time and bandwidth usage, particularly for the correlated subqueries, where the inner query block is evaluated once for each solution of the outer query. In this paper, we study the problem of evaluating SPARQL subqueries over RDF data stored in $3nuts$ p2p network. We study $semijoin$ based optimization technique, and transformation algorithms to transform correlated queries to equivalent uncorrelated once, that would improve the efficiency of nested query evaluation in distributed environment.**

I. Introduction

Nesting of queries is one of the most powerful feature in a query language, which provides the possibility of writing in a single expression a query which uses the output of other queries. The current working draft[1] of SPARQL 1.1 allows a SELECT or AGGREGATE query within the graph pattern of another query, and introduces graph patterns {P FILTER EXISTS/¬EXISTS $\{P'\}\}$ where P' is a graph pattern to test, as a possible type of subquery. Other nesting features have also been gradually incorporated into some SPARQL implementations, like ARQ and Virtuoso.

The authors in [1] have considered the incorporation of SQL subquery features, which contains the operators IN, EXISTS, SOME, and ALL, into SPARQL. They also proposed syntax and semantics for subqueries in SPARQL by presenting the corresponding extensions for subqueries in the widely accepted formalization of SPARQL in [2].

Existing distributed RDF storage and querying systems like RDFPeers [3], Atlas [4], [5], BabelPeers [6], and GridVine [7] use p2p overlay networks to store and query RDF triples in a distributed manner. These RDF database systems store three copies of each triple indexed by the subject, predicate, and object to achieve an efficient search for triples by subject, predicate, or object. Each peer stores the triples it receives in its local database such as relational database. These distributed RDF systems have not considered the problem of distributed evaluation of SPARQL subqueries so far. The evaluation of SPARQL subqueries, particularly correlated queries, in a distributed environment is much more complex and expensive than in centralized ones, because a large number of parameters affect the performance of these queries. In particular, the tuples/triples in these systems are distributed on multiple nodes, thereby inducing communication overhead costs.

For example, consider the following SPARQL correlated subquery in Listing 1, where variable ?Age1 is shared by both outer and inner query blocks.

```
SELECT ?X
WHERE {
?X type Student.
?X age ?Age1
FILTER(NOT EXISTS (SELECT ?Age2
WHERE {?Y type Students.
?Y age ?Age2
FILTER (?Age1 > ?Age2)}))
}
```

Listing 1. SPARQL query returning the youngest student

The existing distributed RDF systems evaluate the triple patterns topdown, and for the evaluation of a correlated query, the semantics of SPARQL query prescribe that the triples of the outer query be substituted in turn into the subquery block in the filter constraint. In the given query in Listing 1, for each result of the outer query block, the inner query block is evaluated once to output the student for which the filter constraint evaluates TRUE (i.e., the youngest student). The nested iteration method [8] used for the evaluation of SPARQL correlated queries may be very expensive in distributed RDF storage and querying systems.

For the efficient evaluation of correlated SPARQL queries on distributed RDF system, an obvious optimization, we will use, is to decorrelate the query, i.e., the inner query block contains no variables from the outer query block. Thus, the inner query block needs to be evaluated only once. In Section IV we will apply some transformations, to rewrite a correlated query to an equivalent, uncorrelated one. For example, the correlated query in Listing 1 can be transformed to the following equivalent, uncorrelated query in Listing 2.

```
SELECT ?X
WHERE {
?X type Student.
?X age ?Age1
FILTER(?Age1 <= ALL(SELECT ?Age2
WHERE {?Y type Students.
?Y age ?Age2}))
}
```

Listing 2. SPARQL query returning the youngest student

As the peers in existing distributed RDF storage and querying systems use relational database, to store the triples they

[1]http://www.w3.org/TR/sparql11-query/

are responsible for, the resulting uncorrelated queries, which contain the SQL nested query operators SOME, and ALL in their bodies, can be easily evaluated on these peers. It gives now a cheaper approach to project the variable ?Age2, in the query in Listing 2, by evaluating the inner query block on corresponding peers, and to move this projection to sites responsible for the evaluation of outer query. In the evaluation of outer query, if the value of the variable ?Age1 of any tuple is less than or equal to the values projected by the variable ?Age2, then the corresponding student is inserted into the result.

Another optimization technique, we will use for the efficient processing of correlated queries is based on the idea of using $semijoin$, i.e., project the inner query block on its correlated variable, and to move this projection to sites responsible for the outer query block to compute the join.

In this paper, we study the distributed evaluation of SPARQL subquery features, included in the latest SPARQL 1.1 working draft, over RDF data stored on top of p2p overlay networks. We use the distributed SPARQL query evaluation scheme in our $3rdf$ system [9], and extend it for the evaluation of SPARQL subqueries. We apply $semijoin$ based optimization technique, and techniques based on the transformation of correlated queries to equivalent, uncorrelated once which are evaluated by underlying query subsystems more efficiently.

The remainder of the paper is organized as follows: We start with the study of several proposals that have been presented for the introduction of subqueries in SPARQL in Section II. We use the 3nuts p2p network for a distributed application which is briefly described in Section III, this section also describes our assumptions regarding data model and query language. Section IV then presents the distributed query evaluation, and optimization strategies for SPARQL subqueries. Finally, in Section V we make a conclusion.

II. Related Work

The current working draft of SPARQL 1.1 offers a feature related to the introduction of subqueries in SPARQL to allow a SELECT query within the graph pattern of another query. Such SELECT queries can also use aggregate functions. A reasonable motivation behind this feature is to reduce the cost of join in the query. The inner query block is evaluated first, and then is joined with the result of the outer query. Variables projected by the inner query block are only visible to the outer query in these nested queries.

```
SELECT ?Email ?Numberofcourses
WHERE {
?X emailAddress ?Email.
?X type Student.
{ SELECT ?X (COUNT (?course) as ?Numberofcourses)
  WHERE { ?X takesCourse ?course }
  GROUP BY ?X }
}
```

Listing 3. SPARQL query returning email addresses, and the number of courses they attend for all students.

For example, consider the SPARQL nested query in Listing 3. Evaluating group-by in the inner query block, and then computing its join with the outer query can result a significant reduction in the cost of the join.

The second feature it presents in relation to the subquery, is the usage of EXISTS/¬EXISTS operators, to allow a graph pattern to be a type of filter constraint, i.e., {P FILTER EXISTS/¬EXISTS $\{P'\}$}. The above expression in the filter constraint tests the presence or absence of a match to the graph pattern P'. For example, the filter constraint, in the SPARQL nested query in Listing 1, tests the absence of a student older then any student returned in the inner query block. This class of subqueries always share correlated variables in the inner and outer query blocks.

In real life practice, SPARQL implementations like ARQ[2], the query engine for Jena, and Virtuoso provide supports for the described subquery features, introduced in the current working draft of SPARQL 1.1 .

The authors in [1] have considered the known operators like IN, SOME, ALL, and EXISTS used in SQL nested queries, and utilized them in filter constraints, to introduce SPARQL nested queries. The set membership operator IN, is used to find a value in a result set of the corresponding inner query. The quantifier operators, SOME and ALL, are used with scalar comparison operators (e.g., $>$), to compare a value with some or all the data values returned by the inner query block. For example, the given query in Listing 2, gives the student younger then all students returned by inner query block, i.e., the youngest student.

The introduction of existential operators EXISTS/¬EXISTS in [1] are almost the same as proposed in the current working draft of SPARQL 1.1, which verifies the presence or absence of a match to the inner graph pattern. They also proposed the syntax and semantics for subqueries in SPARQL, for that they presented the corresponding extensions for subqueries in the widely accepted formalization of SPARQL in [2].

Rewriting queries is a well-known technique applied for optimization of SQL queries [8]. However, to the best of our knowledge, in this line of work the intention of analyzing subqueries is to remove a level of nesting and not to rewrite the subquery to remove correlation of variables inside the subquery as we shall do.

The evaluation of newly introduced SPARQL subquery feature is not addressed by existing distributed RDF storage and querying systems RDFPeers [3], Atlas [4], [5], BabelPeers [6], and GridVine [7] so far. The evaluation of subqueries, correlated queries in particular, may be very complex and expensive in distributed environment. We extend the distributed SPARQL query evaluation scheme in our $3rdf$ system [9] for the evaluation of SPARQL subqueries. The processing of correlated queries in our system are optimized by using $semijoin$ based technique, and through transformations to equivalent, uncorrelated queries.

[2]http://incubator.apache.org/jena/documentation/query/sub-select.html

III. System model and Data model

In this section we elaborate on our system architecture and also describe our assumptions regarding data model and query language used in the system.

A. System model

We assume to have n nodes participating in the underlying 3nuts p2p network [10]. All of these nodes have some knowledge stored as RDF triples. The 3nuts overlay network establishes a distributed search tree providing point and range queries in $\mathcal{O}(\log n)$ routing hops with high probability where n denotes the number of peers in the network. In the 3nuts network, each 3nuts peer has a local view on the *Search Tree* which enables the peer the search in the distributed tree of the entire network. It supports two basic operations: *Get(key)* for searching a certain key and retrieving the associated data items and *Put(key, value)* for storing new data items. Here, the search functionality of the 3nuts network in the search tree is used to place triples at the correct peers responsible for the corresponding index keys and on the other hand for downloading triples for a certain index key from the responsible peers. The RDF data, supposed to be distributed by a peer in the network, is converted to tuples $(\text{key}, \text{triple})$ in order to insert them into the network with the Put-operation. Three tuples are created for each triple with the different keys for subject, predicate, and object to index all three parameters. Each peer in the network is then responsible for a range of index keys, and stores the corresponding tuples for these index keys.

To perform SPARQL queries, we first transform a SPARQL statement into a sequence of so called *triple patterns*. This separation of the query into smaller partial queries reflects the single steps of execution at different 3nuts peers only with their local database and some intermediate results. The triple pattern sequence is then passed to the responsible peer which controls the distributed execution of the query. There are basically two cases in the distributed execution. In the first case, the 3nuts peer will execute the next triple pattern in the sequence if the peer is capable of resolving it with its local database because the peer is responsible for a given subject, predicate, or object and has the corresponding triples in its database. Otherwise it will use the search operation of the 3nuts to find the peer that can execute the query and transmit the query and intermediate results to that peer.

B. Data model

Each node in our system can publish RDF resources in the network. In the RDF data model, resources are expressed as subject-predicate-object expressions, called triples in RDF terminology. The subject in a RDF triple denotes the resource, and the predicate expresses a relationship between the subject and the object. Let U, and L represent URIs and Literals in RDF, a triple $(v_1, v_2, v_3) \in U \times U \times (U \cup L)$ over certain resources U, and L is called a *RDF* $triple$ [11].

Each node in our system can also submit SPARQL queries to extract data from RDF databases stored in the network. These queries are basically composed of triple patterns. Let U, L, and V represent URIs, Literals, and Variables, a triple $(v_1, v_2, v_3) \in (U \cup V) \times (U \cup V) \times (U \cup V \cup L)$ is called a *SPARQL* $triple\ pattern$ [2]. These triple patterns are matched against the RDF triples in the input database when the query is evaluated against some RDF database.

IV. Query Evaluation

We extend the distributed evaluation of RDF conjunctive triple pattern queries considered by the existing approaches RDFPeers [3], Atlas [4], [5], BabelPeers [6], and GridVine [7] with the recently included SPARQL 1.1 subquery features. To understand the evaluation and optimization of SPARQL subqueries in our $3rdf$ [9] system, we briefly describe first, an abstract syntax of SPARQL subqueries. In the following we restrict ourselves on SPARQL SELECT queries which can appear in the inner query block of a subquery.

A. An Abstract Syntax for SPARQL Subqueries

Let UL represent the set of URIs and literals in RDF and V be a set of variables disjoint from UL. We will start with an abstract syntax of SPARQL SELECT queries, and then extend it with the definition of SPARQL subqueries.

Definition 1 (Filter Constraint): Let $?x$, $?y \in V$ be variables and a, $b \in UL$, and θ is a scalar comparison operator ($=$, $\neq$, $<$, $\leq$, $>$, $\geq$) then we define a $filter\ constraints$ recursively as follows.

- The expressions $?x\,\theta\,a$, $?x\,\theta\,?y$, $a\,\theta\,b$, and *bound(?x)* are filter constraints.
- If R_1 and R_2 are filter constraints, then $\neg R_1$, $R_1 \wedge R_2$, and $R_1 \vee R_2$ are filter constraints. □

Using the previous definition, now we can define the syntax of SPARQL graph patterns.

Definition 2 (SPARQL Graph Pattern): A SPARQL graph pattern is built recursively according to following rules.

- A triple pattern $t \in UV \times UV \times LUV$ is a graph pattern.
- If *G* is a graph pattern and *R* is a filter constraint, then *G* FILTER *R* is a graph pattern.
- If G_1, G_2 are graph patterns, then G_1 UNION G_2, G_1 OPT G_2, and G_1 AND G_2 are graph patterns. □

The syntax of a SPARQL SELECT query is defined as follows.

Definition 3 (SPARQL SELECT Query): Let *G* be a SPARQL graph pattern, and let $S \subset V$ be a finite set of variables. A SPARQL SELECT query is an expression of the form $\text{SELECT}_S(G)$. □

Now we are ready to define the syntax of SPARQL subqueries based on the definition in [1].

Definition 4 (SPARQL Subquery): Subqueries as graph patterns: If $\text{SELECT}_S(G)$ is a SPARQL SELECT query, then the expression $(\text{SELECT}_S(G))$ is a graph pattern.

Subqueries in filter constraints: If $v \in (UL \cup V)$, θ is a scalar comparison operator ($=$, $\neq$, $<$, $\leq$, $>$, $\geq$), and $\text{SELECT}_S(G)$ is a SPARQL SELECT query, then the expressions $(v\,\theta\,\text{SOME}(\text{SELECT}_S(G)))$, $(v\,\theta\,\text{ALL}(\text{SELECT}_S(G)))$, $(v\,\theta\,\text{IN}(\text{SELECT}_S(G)))$, EXISTS

(SELECT$_S(G)$)), and ¬EXISTS (SELECT$_S(G)$)) are filter constraints. □

B. Evaluation and Optimization of SPARQL subqueries

Subqueries with correlated variables inside filter constraints are more expensive to be processed than subqueries appearing as a graph pattern.

1) Evaluating Subqueries as graph patterns: These subqueries allow a SELECT query in the place of graph patterns. In these subqueries only the variables projected by the inner query blocks are visible to the outer query, and thus do not share correlated variables. The evaluation of these queries is very simple, evaluate both query blocks independently and join their solutions. For example, for the query in Listing 3, we evaluate the inner query, determine the aggregate value (number of courses) for each student, evaluate the outer query, and then join their solutions to get the email address, and number of courses they have taken.

2) Evaluating Subqueries in filter constraints: The proposed syntax for the SPARQL subqueries [1] extends the SPARQL simple filter constraint defined in Definition 1 to nested filter constraints defined in Definition 4 for the introduction of SPARQL subqueries. For example, the queries in Listing 1, and Listing 2 use SPARQL SELECT queries as nested filter constraints.

The current working draft of SPARQL 1.1, and implementations like ARQ have considered only the EXISTS (SELECT$_S(G)$)), and ¬EXISTS (SELECT$_S(G)$)) nested filter constraints for the introduction of this type of subqueries. We study also only the evaluation of subqueries in our $3rdf$ system, which are part of the SPARQL 1.1.

The use of EXISTS and ¬EXISTS in the above nested filter constraints needs correlated variables to make sense, means there is some variable occurring in both the inner query block in filter constraint (e.g., EXIST(SELECT$_S(G)$)), and the outer query block. The semantics of SPARQL query prescribe that the inner query block of correlated query is evaluated once for each solution of the outer query. For example, consider the correlated query in Listing 1 where the inner query block is evaluated for each value of the variable ?Age1 in the outer query. Since the triples are placed on multiple nodes in distributed RDF systems, the evaluation of SPARQL correlated queries in these systems may be very costly in term of query response time and bandwidth usage.

As the peers in the distributed RDF systems evaluates the triple patterns of a SPARQL query on their local relational databases, the processing of a correlated query would be made efficient, by transforming it to an equivalent uncorrelated query, which contains SQL operators like SOME, ALL in its body. The resulting uncorrelated query can now be easily evaluated on responsible peers. For the efficient evaluation of correlated queries in our $3rdf$ system, we first transform it to an equivalent, uncorrelated nested query. The inner query block of resulting query is now evaluated independently, and then used to filter the solutions of the outer query block.

Definition 5 (Query equivalence): Let D be an RDF database and G_1, G_2 two graph patterns. G_1 and G_2 are equivalent, i.e., $G_1 \equiv G_2$, if and only if $[[G_1]]_D = [[G_2]]_D$. Two queries $Q = (\text{SELECT}_S(G_1))$, and $Q' = (\text{SELECT}_S(G_2))$ are equivalent, i.e., $Q \equiv Q'$, if and only if $G_1 \equiv G_2$. □

The following propositions are based on transformations in [12], however the consequences of removing correlated variables are not considered in that paper. The transformations of EXISTS and ¬EXISTS correlated queries to their equivalent, uncorrelated forms are given as follows.

Proposition 1 (Transforming EXISTS *queries):* Let G be a graph pattern of the form $(G_1$ FILTER (EXISTS (SELECT$_{?S_2}(G_2$ FILTER$(v\,\theta\ ?S_2)))))$ where $v \in (UL \cup V)$, $Q_2 = (\text{SELECT}_{?S_2}(G_2))$, and $\hat{\theta}$ is the inverse operator to θ. Then, G is equivalent to the following graph pattern.

- $(G_1$ FILTER $(v\,\theta$ SOME $(Q_2)))$ □

Proposition 2 (Transforming ¬EXISTS *queries):* Let G be a graph pattern of the form $(G_1$ FILTER (¬ EXISTS (SELECT$_{?S_2}(G_2$ FILTER$(v\ \theta\ ?S_2)))))$ where $v \in (UL \cup V)$, $Q_2 = (\text{SELECT}_{?S_2}(G_2))$, and $\hat{\theta}$ is the inverse operator to θ. Then, G is equivalent to the following graph pattern.

- $(G_1$ FILTER $(v\,\hat{\theta}$ ALL $(Q_2)))$ □

Example 1: Using the transformation in Proposition 2, we can transform the correlated query in Listing 1 to an equivalent, uncorrelated query in Listing 2. The inner query block in Listing 2 can now be processed independently to project the variable ?Age2, and this projection is sent to the site responsible for the outer query to filter its results. □

```
SELECT ?X
WHERE {
?X type Student.
?X advisor ?Adv
FILTER(EXISTS (SELECT ?Adv
WHERE {?P type Publication
?P author ?Adv}))
}
```

Listing 4. SPARQL query returning the students whose advisors have some publications

Another complementary approach, we use for the efficient processing of correlated queries is based on the idea of using $semijoin$. Let A and B represent the outer and inner query blocks of a correlated query respectively, the $semijoin$ of A by B on condition J is defined as the subset of $A - tuples$ for which there are matching $B - tuples$ that satisfy J, i.e., Semijoin $(A, B; J) = \{a \in A \mid \exists b \in B\,(J\,(a, b))\}$.

For the distributed evaluation of correlated SPARQL queries, it may be cheaper to project the inner query block on the correlated variable, and to move this projection to peers responsible for the outer query block. For EXISTS queries, the semijoin of outer query block by inner block is produced by outputting the tuples of the outer block as soon as the first matching tuple of the inner block is encountered, and then advance to the next tuple of the outer query block. For example, for the query in Listing 4, the inner query block is

projected on the correlated variable ?Adv, and this projection is moved to the site for the outer query block to compute the join. In the same way as for the processing of ¬EXISTS queries, tuples of the outer query block are output if no matching tuple is found in the inner query block.

C. Query processing

We extend the query processing algorithm adopted in our 3rdf system [9], where the triple patterns contained in the query are iteratively resolved by a chain of nodes. In the process, each node adds to an intermediate result all triples of its local database, which are qualified for the evaluated query so far. The last node in the line of nodes then has the complete result and returns it to the requesting node. We restrict ourselves to the processing of SPARQL subqueries to depth one. As we use the transformation functions to remove the correlation, the query blocks of resulting uncorrelated queries can be processed independently. The query request takes the parameters $(id, t_i, Q_1, Q_2, C, I, IP(p))$ where id denotes the query id, t_i is the currently active triple pattern, Q_1 represent the outer query block, Q_2 is the inner query block, I accumulates the intermediate result for Q_1, C accumulates the intermediate result for Q_2, *IP(p)* is the IP address of node p that posed the query. Initially, t_i is the first triple pattern in the inner query block with $i = 1$, Q_2 contains all triple patterns except t_i, C and I are empty.

The node, which initiates a new SPARQL nested query request, routes the query request to node r_i through prefix search using one of the constants in t_i. When node r_i receives the query request, it first resolves the triple pattern t_i with a local relational query on the local triple table R, i.e., it computes the relation $T = \pi_X(\sigma_{SC}(R))$ where SC is a selection condition and X represents the positions of variables. For example, if t_i is $(?s_i, p_i, o_i)$ then $T = \pi_{\text{subject}}(\sigma_{\text{predicate}=p_i \wedge \text{object}=o_i}(R))$. Then, r_i computes a new relation with intermediate results $C = \pi_S(C \bowtie T)$ where the set S identifies the attributes of C and T that exist in answer variables, or are needed for the evaluation of the remaining triple patterns. If C is not empty then r_i sends the query message $(id, t_{i+1}, Q_1, Q_2, C, I, IP(p))$ to the node responsible for the evaluation of the next triple pattern t_{i+1}.

When the last node for the evaluation of inner query Q_2 is reached, and the last triple pattern in Q_2 is evaluated, it sets t_i as the first triple pattern in the outer query block Q_1, and sends the message to gather with the intermediate result C, generated for Q_2, to the node responsible for t_i. The triple patterns in Q_1 are iteratively evaluated by chain of nodes, in a same way described for the processing of query block Q_2, and intermediate results are stored in I. As Q_1 is the outer query, it may use the results of the inner query Q_2, accumulated in C, to filter out its results, during the evaluation of Q_1. When the last node for the processing of Q_1 is reached, it creates the query result and sends it back to the start node p using $IP(p)$.

Example 2: Consider the evaluation of SPARQL correlated subquery in Listing 1. We will first transform it to an equivalent, uncorrelated query in Listing 2, and then evaluate it on our overlay network. Let r_1, r_2 be the nodes responsible for the evaluation of the two triple patterns in the inner query block Q_2, and r_3, r_4 be the nodes responsible for the evaluation of the triple patterns in the outer query Q_1. When Q_2 is evaluated by nodes r_1, and r_2 respectively, newly computed intermediate result C to gather with the remaining query block Q_1 is sent to the node r_3, responsible for the evaluation of the first triple pattern in Q_1. After processing the first triple pattern in Q_1 by node r_3, node r_4 evaluates the second triple pattern of Q_1, and filters out results using the values in C. Finally, node r_4 sends the result back to the start node p. □

V. CONCLUSIONS

In this paper, we studied the problem of distributed evaluation of SPARQL subqueries over p2p network. We extended the evaluation of RDF conjunctive triple pattern queries in current distributed RDF systems with the newly included SPARQL subquery features. Due to the distributed placement of triple on multiple peers, the processing of correlated queries in these systems may be very expensive in term of query response time and bandwidth usage. We applied optimization techniques, based on the idea of using $semijoin$, and transformation of correlated queries to equivalent, uncorrelated queries, in order to make the distributed evaluation of nested queries efficient.

REFERENCES

[1] C. G. Renzo Angles, "Subqueries in sparql," in *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2011.

[2] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," *ACM Trans. Database Syst.*, vol. 34, pp. 16:1–16:45, 2009.

[3] M. Cai and M. Frank, "Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network," in *Proceedings of the 13th international conference on World Wide Web*, ser. WWW '04. New York, NY, USA: ACM, 2004, pp. 650–657.

[4] Z. Kaoudi, M. Koubarakis, K. Kyzirakos, I. Miliaraki, M. Magiridou, and A. Papadakis-Pesaresi, "Atlas: Storing, updating and querying rdf(s) data on top of dhts," *Web Semant.*, vol. 8, pp. 271–277, November 2010. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2010.07.001

[5] E. Liarou, S. Idreos, and M. Koubarakis, "Evaluating conjunctive triple pattern queries over large structured overlay networks," in *International Semantic Web Conference*, 2006, pp. 399–413.

[6] F. Heine, "Scalable p2p based rdf querying," in *Proceedings of the 1st international conference on Scalable information systems*, ser. InfoScale '06. New York, NY, USA: ACM, 2006.

[7] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. V. Pelt, "Gridvine: Building internet-scale semantic overlay networks," in *In International Semantic Web Conference*, 2004, pp. 107–121.

[8] W. Kim, "On optimizing an sql-like nested query," *ACM Trans. Database Syst.*, vol. 7, pp. 443–469, September 1982. [Online]. Available: http://doi.acm.org/10.1145/319732.319745

[9] L. Ali, T. Janson, and G. Lausen, "3rdf: Storing and querying rdf data on top of the 3nuts overlay network," *Database and Expert Systems Applications, International Workshop on*, vol. 0, pp. 257–261, 2011.

[10] T. Janson, P. Mahlmann, and C. Schindelhauer, "A self-stabilizing locality-aware peer-to-peer network combining random networks, search trees, and dhts," in *ICPADS*, 2010, pp. 123–130.

[11] C. Gutierrez, C. Hurtado, and A. O. Mendelzon, "Foundations of semantic web databases," in *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ser. PODS '04. New York, NY, USA: ACM, 2004, pp. 95–106.

[12] C. G. Renzo Angles, "Sql nested queries in sparql," in *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2010.