Optimising Linked Data Queries in the Presence of Co-reference

Xin Wang, Thanassis Tiropanis, and Hugh C. Davis

Electronics and Computer Science
University of Southampton
{xw4g08,tt2,hcd}@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/

Abstract. Due to the distributed nature of Linked Data, many resources are referred to by more than one URI. This phenomenon, known as co-reference, increases the probability of leaving out implicit semantically related results when querying Linked Data. The probability of co-reference increases further when considering distributed SPARQL queries over a larger set of distributed datasets. Addressing co-reference in Linked Data queries, on one hand, increases complexity of query processing. On the other hand, it requires changes in how statistics of datasets are taken into consideration. We investigate these two challenges of addressing co-reference in distributed SPARQL queries, and propose two methods to improve query efficiency: 1) a model named Virtual Graph, that transforms a query with co-reference into a normal query with pre-existing bindings; 2) an algorithm named Ψ, that intensively exploits parallelism, and dynamically optimises queries using runtime statistics. We deploy both methods in an distributed engine called LHD-d. To evaluate LHD-d, we investigate the distribution of co-reference in the real world, based on which we simulate an experimental RDF network. In this environment we demonstrate the advantages of LHD-d for distributed SPARQL queries in environments with co-reference.

Keywords: SPARQL, Linked Data, distributed query, dynamic optimisation, co-reference

1 Introduction

Over years a large amount of Linked Data have been published by numerous independent individuals and organisations. When referring to resources, it is desirable to reuse existing URIs [10]. However, it is impractical to guarantee that one resource is only bound to a single URI due to the distributed nature of Linked Data. On class level, common vocabularies, such as Friend of a Friend (FOAF)[1], and Dublin Core Metadata Initiative (DCMI)[2], are shared in many

[1] http://www.foaf-project.org/
[2] http://www.dublincore.org/documents/dcmi-terms/

RDF datasets. On instance level, poor agreement is made [7]. For example, 23 different URIs are found referring to the person Tim Berners-Lee out of 1.118 billion triples [8]. This phenomenon, that multiple URIs referring to the same resource, is known as co-reference. Co-referent URIs are semantically equal[3] from the perspective of Linked Data queries. Without taking co-reference into account, Linked Data queries may leave out valid results that are not explicitly available.

A variety of work has been done to identify co-reference in Linked Data (i.e. co-reference resolution) [7,4]. Once resolved, a pair of co-referential URIs is expressed as an *owl:sameAs* [3] assertion, which states that two URIs are semantically equivalent. By examining existing *owl:sameAs* statements, it is possible to retrieve additional semantically valid results when querying Linked Data. However, the following issues persist in the above process:

1. A naïve approach to query Linked Data with co-reference requires three steps. First, retrieving co-reference for every concrete URI in a given query by consulting *owl:sameAs* statements. Second, executing the original query, as well as its co-referential queries that are obtained by replacing one or more original URIs with their co-reference. Third, combining results of all previously executed queries. This approach results in significant query overheads, and each co-referential query can only be optimised on its own (i.e. the total costs of all co-referential queries are not necessarily minimised).
2. A small amount of co-reference can potentially lead to a large amount of additional results. Thus, query processing with enhanced performance is desirable.
3. Query efficiency is closely related to the statistics of datasets. On a large scale, it is unlikely to have pre-computed statistics taking co-reference into account.

Regarding the above issues, we propose two methods to improve the performance of Linked Data queries in the presence of co-reference:

- A model named Virtual Graph (VrG), that merges all co-referential queries into a normal query with pre-existing bindings. VrG saves the overheads of sending many queries, and especially, enables optimisation regarding all co-referential queries.
- An algorithm named Ψ, that identifies independent sub-queries, which can be optimised and executed independently in parallel, without increasing communication traffic. It is worth mentioning that Ψ helps increase degree of parallelism, and can be as well used in engines that do not take co-reference into account.

Furthermore, sub-queries identified by the Ψ algorithm are optimised at runtime (i.e. dynamic optimisation [11]), using runtime statistics instead of pre-

[3] In practice, co-referential URIs usually refer to closely related resources rather than the exact same resource. However, this issue is not essential in this paper.

computed statistics. Based on the aforementioned methods, we implement a distributed SPARQL engine, named LHD-d[4] that is able to process co-referential queries with improved efficiency.

To evaluate our approach, we investigate co-reference in the real world. Based on the distribution of real-world co-reference, we propose a method to simulate co-reference for arbitrary datasets. We set up a RDF network containing co-reference using an evaluation framework [15,16] that extends the Berlin SPARQL Benchmark (BSBM) [2].

The remainder of this paper is organised around presenting and demonstrating the effectiveness of VrG and Ψ through the evaluation of LHD-d. In section 2 we provide the background of this work and review related approaches that LHD-d is compared to. The core techniques of LHD-d, VrG and Ψ, are described in section 3 and 4 respectively. In section 5 we describe how VrG and Ψ are deployed alongside dynamic optimisation in LHD-d. After that, in section 6, we investigate the distribution of co-reference in the real world, based on which we propose a method to simulate RDF networks having co-reference. We also describe the environment in which LHD-d is evaluated and compared with related approaches. The performance of LHD-d is thoroughly examined in two situations, respectively with or without the presence of co-reference. In section 7 we evaluate LHD-d without taking co-reference into account (by disabling VrG) and compare it with existing engines. This evaluation primarily demonstrate the effectiveness of using Ψ and runtime-statistic-based optimisation. In section 8 we assess LHD-d with the presence of co-reference and compare it with the aforementioned naïve approach. This evaluation focuses on demonstrating the effectiveness of VrG. Finally we give our conclusions and future plans in section 9.

2 Related Work

In this section we discuss the current state and issues of co-reference in Linked Data, and review relevant approaches of Linked Data query processing.

2.1 Co-reference in the Linked Data Cloud

The ubiquity of co-reference in Linked Data motivates many researchers to investigate the similarity between URIs and to infer co-reference relationships [7,14]; to study the semantic and structure of co-referential identifiers [9], and to develop efficient methods for co-reference representation and management [4].

Co-reference can be explicitly presented by *owl:sameAs*. However, many co-referential URIs are inexplicit in reality. To determine equivalent URIs, or co-reference resolution, it requires to investigate the semantic and relationship of relevant URIs. Taking [7] as an example, the authors proposed a method that

[4] LHD stands for **L**arge scale, **H**igh performance and **D**istributed. "d" stands for **d**ynamic optimisation.

uses inverse-functional property, *owl:InverseFunctionalProperty*[5], to infer the equivalence of URIs. Similarly, *owl:FunctionalProperty*, *owl:maxCardinality*, and *owl:cardinality* have also been examined to infer *owl:sameAs* statements [6]. Furthermore, a scalable candidate selection algorithm is proposed by Song et al. [14]. This algorithm firstly identifies properties that have discriminability and coverage larger than a certain threshold, as keys. These keys are used to disclose additional co-referential URIs closely related in semantics. In practice, co-reference resolution services, such as *sameas.org*[6] [4], have been established.

It is realistic to assume that there are a large number of existing *owl:sameAs* statements provided either by datasets themselves or third-party services. In this work we do not deal with co-reference resolution. We focus on improving the efficiency of Linked Data queries taking existing *owl:sameAs* statements into account. It is straightforward to add co-reference resolution as an extra layer on top of LHD-d.

2.2 Distributed Query Processing over Linked Data

Query processing over Linked Data has attracted much attention in recent years. As a result, many distributed SPARQL engines, such as DARQ [12], DSP [15], FedX [13], SPLENDID [5] and LHD [16], have been proposed to address various issues of Linked Data queries. However, none of these engines investigate the possibility of taking co-reference into Linked Data queries[7]. Since LHD-d will be evaluated without co-reference as well, it is worth providing details of representatives of existing engines. Evaluation results of [15,5,13,16] suggests that FedX and LHD have certain advantages over other approaches. We provide their details here and will compare LHD-d to these two engines in section 7.

FedX does not require statistics of datasets. Given a query, FedX sends *ASK* queries to all known datasets to identify those relevant to a certain triple pattern. This selection is accurate but introduce additional network overheads. The optimisation of FedX adopts a greedy algorithm that picks the minimum from triple patterns that are not executed. Triple patterns are ranked using heuristics, according to the number and position of variables in those triple patterns. As a result, it does not distinguish arbitrary two triple patterns having variables at the same positions. FedX adopts an novel method that executes triple patterns using multiple threads, which significantly improves query efficiency.

LHD follows almost an opposite direction of FedX. It requires detailed statistics that are retrieved from VoID [1] files. However, as admitted by the authors, detailed VoID files are unlikely to be available on a large scale. Relevant datasets are selected using the predicate-matching method, that a triple pattern is assigned to datasets that contains its predicate. This method is less accurate than

[5] The value of an inverse-functional property uniquely identifies the subject of this property.

[6] www.sameas.org

[7] To the best of our knowledge, the OpenLink Virtuoso is the only distributed engine that provides support of co-reference in a recent release. However, Virtuoso focus on resolving co-reference rather than improving query efficiency.

the ASKing approach of FedX, but causes no extra network overhead. LHD adopts dynamic programming that exhaustively searches for the optimal plans, and the quality of query plans only depends on the accuracy of VoID statistics. A sophisticated parallel query execution is used in LHD to maximumly exploit bandwidth of connections to remote datasets.

3 Virtual Graph: Merging Co-reference into SPARQL Queries

Assuming there is a query {?x foaf:knows p_0} and a co-reference statement {p_0 owl:sameAs p_1}, results of both {?x foaf:knows p_0} and {?x foaf:knows p_1} are semantically valid for the original query. For convenience, we say two queries are co-referential if there is a mapping between these two queries that maps a concrete URI to either itself or its co-reference. Also, we refer to the result set extended by co-reference (including original results) as co-referential results. Using existing *owl:sameAs* statements in the Linked Data cloud, it is straightforward to gather co-referential results by executing a given query and all co-referential ones. However, this method is not practical to handle complex queries, since we have to execute the Cartesian product of the co-reference of all triple patterns. To address this issue, we propose a model called Virtual Graph (VrG) that enables simultaneous optimisation and execution of all co-referential queries.

During query execution, variables are gradually bound to values. Following the same idea, VrG regards a concrete node as a variable that is bound to one value. When taking co-reference into account, a concrete node is regarded as a variable whose values are the union of its original URI and all co-reference. An example of Virtual Graph is shown in Figure 1.

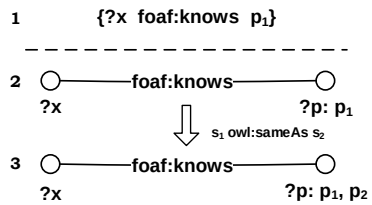


Fig. 1: ① shows a triple pattern of a SPARQL query. The corresponding VrG is shown in ②. ③ is the VrG after taking an *owl:sameAs* statement into account.

In LHD-d, the transformation of VrG is applied at the beginning of query processing. For each concrete URI denoted by v in a given query, our engine firstly generates a query {v owl:sameAs ?coref} to all datasets that may contain

co-referential URIs of v. Then v is replaced by a variable node $?v$, whose values are the union of v and its co-reference. The whole transformation is analogous to the process shown in Figure 1. The essential of VrG is to enable query optimisation algorithms to produce optimal plans w.r.t all co-referential queries. Also it enables query engines to better exploit parallelism.

4 Ψ: Parallel Sub-Query Identification

SPARQL queries are composed by Basic Graph Patterns (BGPs), which are a set of conjunctive triple patterns. A BGP can be regarded as a connected graph whose nodes (or vertices) are subjects and objects and whose triple patterns are edges. We observed that given two edges (triple patterns) whose shared node is concrete (e.g. {s p_1 ?x. s p_2 ?y}), they can be processed as two independent sub-queries without increasing network traffic. This is because the number of values of the shared node (which is concrete) is not affected by any edge that connects to it. This observation also holds for variables whose number of values does not change during execution.

We generalise the above observation as follows. We say a node has a *fixed cardinality* if, during the execution of edges connecting to it, its number of values does not change more than a certain percentage. If "removing" all fixed-cardinality nodes results in disconnected sub-graphs, these sub-graphs can be optimised and executed independently and in parallel. For example, in the graph shown in Figure 2, if both node B and C are fixed-cardinality nodes, then we have three independent sub-graphs $\{AC, AB\}, \{BC\}, \{CD, BD\}$. If only B has fixed cardinality, then the given graph cannot be further broken down[8].

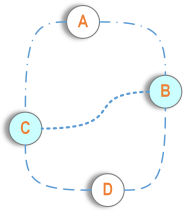


Fig. 2: If B and C are fixed-cardinality nodes, there are three independent components shown by three different types of dash lines.

[8] A more subtle case is that cardinality of both B and C are only changed by AB and AC respectively, while BC and BD have comparable cardinality at B, and BC and CD have comparable cardinality at C. That is, B and C are not fixed-cardinality nodes w.r.t all connecting edges, but they are w.r.t some edges. In this case $\{CB\}, \{CD, BD\}$ can still be executed in parallel, and we say this two components form a partial parallel group. However, identifying all partial parallel group can be costly and not worthy in practice.

Utilising the aforementioned idea, we propose the algorithm Ψ[9] (shown in Algorithm 1) that quickly breaks a connected graph into independent components. At the beginning the algorithm creates a sub-graph for each edge (the loop at line 1). Then all nodes are scanned and sub-graphs that share a none-fixed-cardinality node are merged (the loop at line 4). At the end of this algorithm, all remaining sub-graphs can be processed in parallel. The time complexity of the first loop is linear to the number of edges $|E|$. The merge operation in the second loop can be done in constant time by maintaining a hash table that maps a node to the set of sub-graphs connected to it. Therefore, the complexity of the second loop is linear to the number of vertices $|V|$. The complexity of Ψ (upper bound) is $O(\max(|E|, |V|))$.

Algorithm 1: Ψ(V,E)

input : A connected graph (V, E)
output: Independent sub-graphs

1 **foreach** $e \in E$ **do**
2 $sg_{\{e\}} \leftarrow e$;
3 **end**
4 **foreach** $v \in V \wedge \neg$ `fixCard(`v`)` **do**
5 *merge sub-graphs containing* v;
6 **end**

In practice, concrete nodes always have fixed-cardinality. Besides, if we can know in advance that the cardinality of a variable node will probably remain the same, that node can be regarded as a fixed-cardinality node as well. For example, in {?person foaf:firstName ?frstN. ?person foaf:familyName ?fmName}, the cardinality of *?person* is probably fixed during execution, since a dataset usually contains both the first name and family name of a person[10]. Besides, heuristics can be used to identify fixed-cardinality nodes. For example, if the estimations of the cardinality of a variable $?v$ w.r.t all its connected triple patterns are close[11], the number of bindings of $?v$ probably will not change. Also, if the number of existing bindings of $?v$ is very small[12], it probably will not change. The effectiveness of the above two heuristics depends on the accuracy of cardinality estimation. We enable these heuristics in LHD-d since runtime statistics are used (described as below).

[9] Ψ=PSI=**P**arallel **S**ub-query **I**dentification
[10] Property schemas are required to accurately predict the invariance of a node's cardinality. For instance, in this example we need to know that both properties have the same domain, have close numbers of distinct subjects, and are closely relevant.
[11] $90\% < \frac{card(T_i,?v)}{card(T_j,?v)} < 110\%$, that T_i, T_j are triple patterns connecting to $?v$.
[12] $|?v| < \min(card(T, ?v))/10$, that T connects to $?v$.

5 Dynamic Optimisation Using Runtime Statistics

The effectiveness of query optimisation is closely related to the accuracy of cost estimation [11]. On a large scale, the most promising way of obtaining statistics is from VoID [1] files provided by RDF datasets. However, these statistics are unlikely to take co-reference into account. To this end, LHD-d exploits statistics that become available at runtime, and interleaves query optimisation and execution. Each time a triple pattern is executed, its result size is used to estimate cost of remaining triple patterns.

Cost estimation

We denote by $sel(t, n)$ the selectivity of a node (either a subject or an object) w.r.t a triple pattern t, and by $|p|$ the number of triples having p as predicate in relevant datasets. $sel(t, n)$ and $|p|$ are obtained from available statistics such as VoID files. For more details of these values please refer to SPLENDID [5] and LHD [16]. The cardinality $card(t)$ of a triple pattern $t : \{s\ p\ o\}$ is estimated as:

$$card(t) = |s| \cdot sel(t, s) \cdot |p| \cdot |o| \cdot sel(t, o) \tag{1}$$

where $|s| = 1/sel(t, s)$ if s is a variable having no bindings (i.e. an unbound variable does affect the cardinality), otherwise $|s|$ is the number of values of s. $|o|$ is determined in the same manner. During query execution, $|s|$ and $|o|$ are updated as new bindings becoming available.

The cost of a triple pattern depends on the execution method. If it is evaluated over relevant datasets without attaching existing bindings, the cost is estimated as $card(t) \cdot c$, where c is a constant. If existing bindings, presumably from s, are attached, the cost is estimated as $|s| \cdot c + 1 \cdot sel(t, s) \cdot |p| \cdot |o| \cdot sel(t, o) \cdot c$.

Query optimisation

Given a (sub-)graph, we use a minimum-spanning-tree-based algorithm, shown in Algorithm 2, to find the order of triple pattern execution in real time. Each time the algorithm is called, it maintains a list of remaining edges ordered by their costs from low to high. If an edge has two possible costs, the smaller one is chosen. Then the algorithm returns and removes the minimum edge (i.e. an edge belongs to the MST), which is going to be executed. It also returns edges whose subjects and objects are all bound (i.e. edges that do not belong to the MST), which are used to prune existing bindings.

Algorithm 2: NextEdges(V,E)

input : A connected (sub-)graph (V, E)
output: *next* a set of edges to be executed

1 $edges \leftarrow sort(E)$;
2 $next \leftarrow edges[0]$;
3 $next \leftarrow next \cup findBoundEdges(edges)$;
4 $E \leftarrow edges - next$;

The overview of query execution of LHD-d is shown as Algorithm 3. Firstly a given query is broken into sub-graphs. For each sub-graph a new thread is created. At each step, minimum-cost triple patterns are selected (lines 6) and executed (line 7 to 8). Then cost of remaining edges (executed edges are removed at the end of algorithm $NextEdges(V, E)$) are updated using runtime statistics and $Execute(V, E)$ is called recursively. It should be noticed that a sub-graph can be further divided in future call of $Execute(V, E)$ w.r.t updated edge cost.

Algorithm 3: Execute(V,E)

input : A connected (sub-)graph (V, E)

1 **if** *E is empty* **then**
2 return;
3 **end**
4 $components \leftarrow \Psi(V, E)$;
5 **foreach** *sub-graph* $(V', E') \in components$ *create a new thread* **do**
6 $next \leftarrow NextEdges(V', E')$;
7 evaluate $next[0]$;
8 use remaining edges of *next* to prune bindings;
9 update costs of edges in E';
10 $Execute(V', E')$;
11 **end**

6 Experimental Environment

We use an evaluation framework that extends BSBM [2] to set up the experiment environment. For more details of the evaluation framework please refer to [15,16]. In this section we further study the distribution of co-reference in Linked Data to set up an environment in which LHD-d is evaluated.

6.1 Distribution of Co-Reference in Linked Data

Some research implies that co-reference follows a power law distribution [9], without giving explicit evidence. We analyse the data of Billion Triple Challenge (BTC) 2011[13], which can be regarded as a snapshot of real world Linked Data. We counted the number of resources involved in 1, 2, 3 ... *owl:sameAs* statements respectively, and produce the diagram shown in Figure 3. We find that points in Figure 3 approximate a power law distribution $p(x) = \alpha x^{-\beta}$ where $\beta = 2.528$. The scale-free property of power law distribution allows us to replicate the real-world co-reference distribution on a smaller scale.

Generation of co-reference is achieved by linking resources using *owl:sameAs*. To reproduce the distribution of real-world co-reference, we use a power law

[13] http://challenge.semanticweb.org/

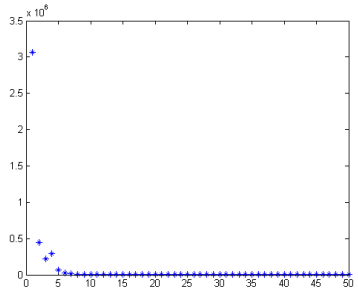


Fig. 3: The horizontal axis presents categories of resources having 5, 10, 15 ... co-reference respectively, while the vertical axis presents the number of resources falling in each category.

random number generator that accepts two parameters which are the power law exponent $\beta = 2.528$ and the number of elements (i.e. distinct resources that have co-reference). For a given resource, we use this generator to decide the number of *owl:sameAs* statements that link this resource with other randomly chosen resources. We also take into account that resources of BSBM data fall into different classes. We generate co-reference for each class separately to make sure that resources are only equivalent to those of the same class. Furthermore, numbers that are larger than the total number of instances of a class are discarded, since the maximum number of co-references a resource can have is the cardinality of the class to which it belongs.

6.2 Experimental Settings

We generate about 70 million triples using the BSBM generator, and 0.18 million *owl:sameAs* statements following the aforementioned method. All the triples (including the *owl:sameAs* statements) are distributed over 20 SPARQL endpoints which are deployed on 10 remote virtual machines having 2GB memory each. All SPARQL endpoints are set up using Sesame 2.4.0 and Apache Tomcat 6 with default settings. Engines to be evaluated are run on a machine having an Intel Xeon W3520 2.67 GHz processor and 12 GB memory.

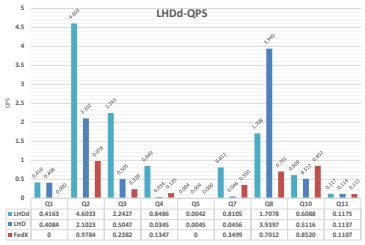
In the following sections we will provide details of LHD-d, and evaluate it afterwards in the above environment.

7 Evaluating LHD-d in the Absence of Co-reference

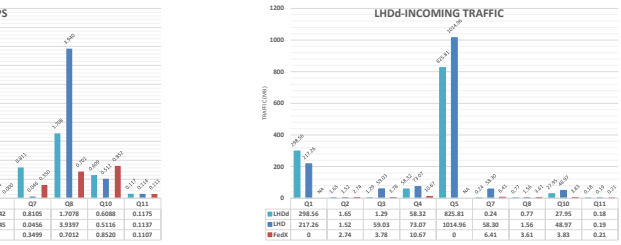
In this section we evaluate LHD-d without co-reference, and compare it with up-to-date engines, namely FedX and LHD. In this evaluation VrG is disabled in LHD-d, and we focus on demonstrating the effectiveness of using Ψ with the runtime-statistic-based query optimisation.

7.1 Evaluation Results and Analysis

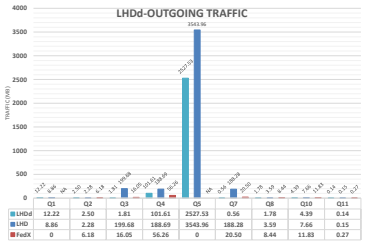
The QPS, incoming traffic, outgoing traffic, and transmission rate of FedX, LHD and LHD-d are shown in Figure 4a, 4b, 4c and 4d. "0" and "NA" stand for failures of execution.



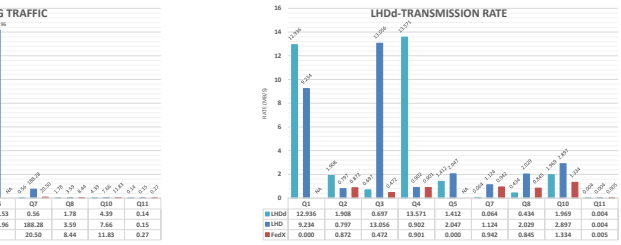
(a) QPS



(b) Incoming traffic



(c) Outgoing traffic



(d) Average transmission rate

Fig. 4: Evaluation without co-reference

It is shown in Figure 4a that LHD-d has an higher QPS over LHD on most queries. Especially, significant performance boost is shown on Q2, Q3, Q4 and Q7. The boost on Q2 and Q4 is primarily due to increased transmission rate (Figure 4d), on Q3 is due to decreased network traffic (Figure 4b and 4c), and on Q7 is due to both factors. LHD-d is slower than LHD on Q8 (but still two times faster than FedX), which is due to its relatively slow transmission rate. On Q10 LHD-d shows slight improvement, but FedX is still the one with highest QPS.

LHD-d produces the smallest amount of network traffic on most queries (Figure 4b and 4c). It is worthy noticing that in LHD-d parallelisation is determined by the Ψ algorithm without increasing network traffic, and each sub-query is optimised with an aim of minimum traffic. Compared to the network traffic of FedX and SPLENDID (recalling that SPLENDID produces more traffic than FedX), we conclude that using runtime statistics yields more accurate estima-

tions and leads to query plans that are closer to optimal. The results further reinforce the previous discussion that the existing cost models or VoID statistics are not sufficiently accurate.

The transmission rate of LHD-d varies on different queries. On Q1, Q2 and Q4 LHD-d has even higher transmission rate than LHD, while on Q3, Q7 and Q8 its transmission rate is relatively low. A closer look reveals that LHD-d produces a small amount of network traffic on Q3, Q7 and Q8, and still has highest QPS on these queries.

In summary, LHD-d better balances between reducing network traffic and increasing average transmission rate, and thus shows a higher overall efficiency. The primary advantage of LHD-d results from the Ψ algorithm and the usage of runtime statistics. It also demonstrates that dynamic optimisation is promising for large scale Linked Data queries, in which cases detailed statistics are difficult to obtain.

8 Evaluating LHD-d in the Presence of Co-Reference

In this section we evaluate the efficiency of LHD-d with co-reference taken into account (thus VrG is enabled), and compare it with the naïve approach of processing co-referential queries described in section 1. In the naïve approach, each co-referential query is executed individually using LHD-d without enabling VrG. It still benefits from Ψ and runtime optimisation. This evaluation focuses on demonstrating the effectiveness of VrG. To demonstrate the consequence of taking co-reference into account, we compare the performance of LHD-d with or without co-reference. In the remainder of this section we use LHD-d* to represent the evaluation results obtained in the presence of co-reference, and LHD-d to represent the results obtained without co-reference.

8.1 Evaluation Results and Analysis

We show in Table 1 that both LHD-d and the naïve approach produce the same sizes of results with co-reference taken into account. This confirms the ability of VrG to fully retrieve additional results due to co-reference. Meanwhile, the result sizes are raised many times (even orders of magnitude on specific queries) by the small proportion of additional co-reference statements. The result sizes of Q5 and Q11 remain the same for different reasons. Q5 selects for products that share the same feature with a given product. There are 14499 distinct products in our dataset, all of which are already contained in the result of Q5 without co-reference. By turning on co-reference, many more intermediate results are generated (demonstrated by the network traffic of Q5 in Figure 5b), but the final result does not change. Q11 does not have concrete subjects or objects, so its result size remains the same.

Three factors are relevant to the significant amount of additional results. First, the same vocabulary is shared by all endpoints. Second, in our datasets co-reference exists between instances of all classes. Consequently, Cartesian product

is likely to be produced by the co-reference of the concrete subjects and objects in a query. Finally, instances of the same class have similar relationships with instances of other classes. Therefore, each co-referential URI may well lead to a valid result.

Table 1: Result sizes of co-reference

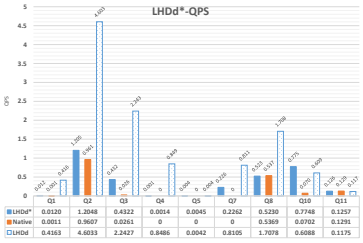
	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q7**	**Q8**	**Q10**	**Q11**
LHD-d*	7397	103	23	65510	14499	1579	101	32	10
Naïve	7397	103	23	NA	NA	NA	101	32	10
LHD-d	53	29	8	29	14499	63	21	12	10

We present the QPS, the incoming and outgoing traffic, and the transmission rate of LHD-d*, LHD-d, and the naïve approach respectively in Figure 5a, 5b, 5c and 5d. "0" and "NA" stand for time out.

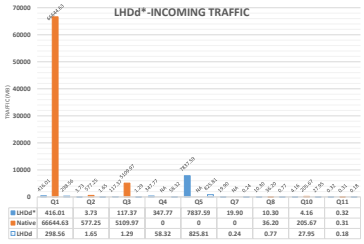
It is shown in Figure 5a that the efficiency of query processing decreases multifold times after introducing co-reference, especially for the naïve approach. Moreover, the naïve approach runs out of time on several queries (Q4, Q5 and Q7) that have a large result size. Though having low QPS on a few queries, LHD-d*, or VrG, substantially increase the efficiency of co-reference query processing. Furthermore, on Q10 LHD-d* has an even higher QPS than LHD-d, indicating a query plan that overcomes the negative impact of co-reference, is generated. Q11 has no co-reference, and the three approaches show close QPS.

From the network traffic of both LHD-d* and the naïve approach (Figure 5b and 5c), it is shown that co-reference significantly increase the sizes of intermediate results. Recalling that the usage of VrG is the only difference between LHD-d* and the naïve approach, we conclude that optimising co-reference queries w.r.t all co-reference yields better query plans. On the contrary, although the naïve approach produces optimal plans for each co-referential query, the total query time is not well controlled. In the meantime, LHD-d* shows much smaller network traffic over the naïve approach. LHD-d* and the naïve approach have the same amount of traffic on Q11, which is slightly larger than that of LHD-d. The extra traffic over LHD-d is due to searching for co-reference of Q11.

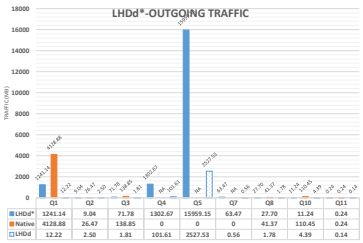
The transmission rate of LHD-d* are not always larger than that of LHD-d. This is further confirms that VrG enables LHD-d* to generate query plans that are tailored for co-reference. If the same LHD-d's query plans are used, the transmission rate of LHD-d* would always be no less than LHD-d, since more traffic is generated in the case of LHD-d*.



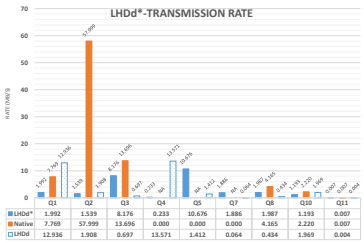
(a) QPS



(b) Incoming traffic



(c) Outgoing traffic



(d) Average transmission rate

Fig. 5: Evaluation with co-reference

9 Conclusion and Future Plan

In this paper we investigate efficiency issues of Linked Data queries in the presence of co-reference. For addressing these issues we propose two techniques called Virtual Graph and Ψ. VrG is able to merge all co-referential queries into a single query with pre-existing bindings. Thus, VrG enables query optimisation algorithms to produce optimal plans w.r.t all co-referential queries. Ψ breaks a query into sub-queries that can be optimised and executed in parallel. We combine Ψ with runtime optimisation to improve query efficiency without relying on detailed pre-computed statistics.

The aforementioned techniques are deployed in LHD-d. We compare LHD-d with representative engines, LHD and FedX, without co-reference, and demonstrate the advantage of using Ψ with runtime optimisation. We also evaluate LHD-d in the presence of co-reference, and demonstrate that VrG significantly improves query optimisation and thus reduces query response time.

In the future we plan to integrate co-reference resolution into our optimisation techniques, which will significantly expand the ability of Linked Data queries. In addition, it is worth further investigating optimisation techniques that consume runtime statistics, since reliable statistics are unlikely to be available in large scale Linked Data.

References

1. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets on the design and usage of VoID , the "Vocabulary of Interlinked Datasets". In: proceedings of the Linked Data on the Web Workshop (LDOW) , at the International World Wide Web Conference (WWW) (2009)
2. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International Journal On Semantic Web and Information Systems (IJSWIS) - Special Issue on Scalability and Performance of Semantic Web Systems 5(2), 1–24 (2009)
3. Carroll, J., Herman, I., Patel-Schneider, P.F.: OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition) (2012), http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/
4. Glaser, H., Jaffri, A., Millard, I.: Managing Co-reference on the Semantic web. In: proceedings of the Linked Data on the Web Workshop (LDOW) , at the International World Wide Web Conference (WWW) (2009)
5. Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: proceedings of the Consuming Linked Data Workshop(COLD) (2011)
6. Hogan, A.: Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora (2011)
7. Hogan, A., Harth, A., Decker, S.: Performing object consolidation on the semantic web data graph. In: proceedings of 1st I3: Identity, Identifiers, Identification Workshop (2007)
8. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and browsing Linked Data with SWSE: the Semantic Web search engine. Semantic Search over the Web pp. 361–414 (2012)
9. Hu, W., Chen, J., Zhang, H., Qu, Y.: How matchable are four thousand ontologies on the semantic Web. The Semantic Web: Research and Applications 6643, 290–304 (2011)
10. Hyland, B., Villazón-Terrazas, B., Atemezing, G.: Best practices for publishing Linked Data (W3C editor's draft 13 March 2013) (2013), https://dvcs.w3.org/hg/gld/raw-file/default/bp/index.html
11. Özsu, M., Valduriez, P.: Principles of distributed database systems (1999)
12. Quilitz, B.: Querying distributed RDF data sources with SPARQL. The Semantic Web: Research and Applications pp. 524–538 (2008)
13. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: proceedings of the International Semantic Web Conference (ISWC) (2011)
14. Song, D., Heflin, J.: Automatically generating data linkages using a domain-independent candidate selection approach. The Semantic WebISWC 2011 pp. 649–664 (Oct 2011)
15. Wang, X., Tiropanis, T., Davis, H.C.: Evaluating graph traversal algorithms for distributed SPARQL query optimization. In: proceedings of the Joint International Semantic Technology Conference (JIST) (2011)
16. Wang, X., Tiropanis, T., Davis, H.C.: LHD: Optimising Linked Data query processing using parallelisation. In: proceedings of the Linked Data on the Web Workshop (LDOW) , at the International World Wide Web Conference (WWW) (2013)