Noname manuscript No.
(will be inserted by the editor)

An Overview on Execution Strategies for Linked Data Queries

Olaf Hartig

Abstract The publication of Linked Open Data on the Web has gained tremendous momentum over the last five years. This development makes possible (and interesting) the execution of queries using up-to-date data from multiple, automatically discovered data sources. As a result, we currently witness the emergence of a new research area that focuses on an *online* execution of *Linked Data queries*; i.e. queries that range over data that is made available using the Linked Data publishing principles.

This article provides a general overview on this new area. In particular, we introduce the specific challenges that need to be addressed and then focus on possible strategies for executing Linked Data queries. Furthermore, we classify approaches proposed in the literature w.r.t. these strategies.

Keywords Linked Data · query execution · survey · comparison · Web of Data · RDF · SPARQL

1 Introduction

Making data openly available in a machine-processable form and interlinking it to other data has become a non-negligible trend on today's World Wide Web [4,29,31]. In particular, community initiatives and research groups as well as enterprises and government initiatives contribute to what is often called the *Web of Data* [4]. Prominent publishers include the BBC, the New York Times, the UK government, Freebase (owned by Google), Best Buy, and Sears. Available data covers diverse topics such as books, movies, music, radio and television programs, reviews, scientific publications, genes, proteins, medicine, clinical trials, geographic locations, people, companies, statistical and census data, etc. While more and more publishers join the trend, the size and the coverage of the Web of Data increase continuously.

The availability of all this interlinked data presents an exciting development; the feasibility to query this emerging dataspace as if it was a huge multidatabase system opens possibilities not conceivable before. The challenge is how to implement such a query functionality.

1.1 Publishing Principles for the Web of Data

In order to identify options for implementing such a query processing functionality we briefly recall the Linked Data publishing principles [3] that specify the basis for contributing to the Web of Data[1]. These principles require data providers to use: (i) HTTP, as data access protocol, (ii) HTTP-scheme-based URIs, as identifiers for entities described in the data, and (iii) RDF, as the data model to represent data. Any HTTP-URI in an RDF triple may then be understood as a *data link* that enables Linked-Data-aware software clients to retrieve more data by looking up the URI on the Web.

For instance, the Library of Congress denotes the historic American newspaper "Richmond Dispatch" by the URI http://chroniclingamerica.loc.gov/lccn/sn85038614#title. Requesting data using this URI results in a set of RDF triples including those given in Figure 1. The last of these triples relates the Richmond Dispatch to its area of coverage –the city of Richmond, VA– using a URI minted by the GeoNames Web site. A Linked Data client may look up this URI (and, thus, follow the provided data link) to obtain data about Richmond, VA, from the GeoNames database; Figure 2 lists some of the RDF triples that can be retrieved by such a lookup.

Olaf Hartig
University of Waterloo
David R. Cheriton School of Computer Science
200 University Ave West
Waterloo, Ontario N2L 3G1, Canada
Tel.: +1-519-888-4567 ext. 33734
E-mail: ohartig@uwaterloo.ca

[1] For a more comprehensive introduction to publishing Linked Data we refer to Heath and Bizer's recent book [21].

(http://chroniclingamerica.loc.gov/lccn/sn85038614#title, dct:title, "Richmond dispatch.")
(http://chroniclingamerica.loc.gov/lccn/sn85038614#title, rdf:type, http://purl.org/ontology/bibo/Newspaper)
(http://chroniclingamerica.loc.gov/lccn/sn85038614#title, frbr:successorOf, http://chroniclingamerica.loc.gov/lccn/sn84024738#title)
(http://chroniclingamerica.loc.gov/lccn/sn85038614#title, dct:coverage, http://sws.geonames.org/4781708/)

Fig. 1 Excerpt of Linked Data about the Richmond Dispatch (properties in the given RDF triples are shortened using the following namespace prefixes: rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#, dct: http://purl.org/dc/terms/, frbr: http://purl.org/vocab/frbr/core#).

(http://sws.geonames.org/4781708/, gn:name, "Richmond")
(http://sws.geonames.org/4781708/, gn:parentCountry, http://sws.geonames.org/6252001/)
(http://sws.geonames.org/4781708/, geo:lat, "37.55376")
(http://sws.geonames.org/4781708/, geo:long, "-77.46026")

Fig. 2 Excerpt of Linked Data about Richmond, VA (properties in the given RDF triples are shortened using the following namespace prefixes: gn: http://www.geonames.org/ontology#, geo: http://www.w3.org/2003/01/geo/wgs84_pos#).

In addition to exposing their data as per the Linked Data principles, several publishers also provide a Web service for executing queries over their data. Usually, such a service supports the query language SPARQL [35] and can be accessed using the corresponding SPARQL protocol [7]. Therefore, such a service is often called a *SPARQL endpoint*.

The query language SPARQL is based on RDF graph patterns and subgraph matching: The basic building blocks of SPARQL queries are basic graph patterns (BGPs), that are, sets of triple patterns; a triple pattern is an RDF triple whose subject, predicate, and object may be a query variable, respectively. Query results in SPARQL are defined in the context of BGP matching, that is, each element of the result basically represents a matching subgraph in the queried RDF graph. More complex query patterns are defined by an algebra that operates over SPARQL result sets [35].

1.2 General Query Approaches for the Web of Data

Several general options for querying the Web of Data exist. In the simplest case, an application may access the SPARQL endpoint provided by a particular data publisher. While such an access may already provide the application with valuable data, this approach ignores the great potential of the Web of Data; it does not exploit the possibilities of this huge dataspace that integrates a large number of interlinked datasets. For instance, a query (such as given in Figure 3) for the geographic location of the area once covered by the Richmond Dispatch can neither solely be answered using data from the Library of Congress nor using data only from the GeoNames database. However, by combining the corresponding data, it becomes possible to answer this simple query. Hence, we are interested in approaches for executing queries over the virtual union of Linked Data from multiple providers.

The database literature focuses on two paradigms for querying distributed data provided by autonomous sources: data warehousing and federated query processing. In an earlier article, we discuss the merits and shortcomings of adapting these approaches to the Web of Data [20].

```
SELECT ?lat ?long WHERE {
  <http://chroniclingamerica.loc.gov/lccn/sn85038614#title>
                                        dct:coverage ?area .
  ?area geo:lat ?lat ;  geo:long ?long .  }
```

Fig. 3 SPARQL query that asks for the geographic location of the area once covered by the Richmond Dispatch (prefix declarations omitted).

Data warehouse approaches are based on copying data into a centralized repository in a manner similar to the collection of Web documents managed by search engines for the Web. By using such a repository, it is possible to provide almost instant query results. This capability comes at the cost of setting up and maintaining the centralized repository. Thus, query results may not reflect the most recent data and users can only benefit from the portion of the Web of Data that has been copied into the repository.

Federated query processing approaches distribute query execution over the SPARQL endpoints that publishers provide for their Linked Data sets. Building a federator for a given set of SPARQL endpoints differs not much from work on relational federation systems [11]. The advantage of using such a federator is that there is no need to synchronize copied data; instead, queries are always answered based on the original, up-to-date data. With version 1.1 of SPARQL, query federation even becomes a feature of the query language: The keyword SERVICE enables users to identify subqueries that have to be processed by remote SPARQL endpoints [34]. However, a particular downside of all SPARQL federation approaches is their limited coverage: We cannot assume that each publisher provides a SPARQL endpoint for its Linked Data. In contrast, while the Linked Data principles present a simple publishing method that can be easily added to existing workflows for generating HTML pages[2],

[2] Using the RDFa standard, Linked Data can even be embedded in HTML documents [1], allowing publishers to serve a single type of document for human and machine consumption.

providing and maintaining a (reliable) SPARQL endpoint presents a significant additional effort that not all publishers are willing (or able) to make. Therefore, querying the Web of Data as a federation of SPARQL endpoints results in ignoring a large portion of Linked Data available.

Given the limitations that data warehousing and federated query processing have in exploiting the Web of Data to its full potential, some research groups recently started to study approaches for *Linked Data query execution*, that is, an online execution of queries for which the query execution system relies only on the Linked Data principles. Hence, to obtain Linked Data that is (potentially) relevant for answering a given query, these approaches look up URIs during the query execution process itself. Therefore, Linked Data query execution focuses on live querying use cases where freshness and discovery of results is more important than an almost instant answer.

This article aims to provide readers with an informal overview on the emerging field of Linked Data query execution. To this end, we first review important characteristics of the Web and their consequences for Linked Data query execution (cf. Section 2). We then discuss possible strategies for executing Linked Data queries and classify early approaches proposed in the literature w.r.t. these strategies. For our discussion, we focus on three orthogonal dimensions: i) data source selection (cf. Section 3), ii) data source ranking (cf. Section 4), and iii) integration of data retrieval and result construction (cf. Section 5).

2 Characteristics of the Web of Data

We note that the Web of Data has certain characteristics that are significantly different from the characteristics of any traditional (distributed) database system. These characteristics have an impact on Linked Data queries and on how these can be executed. Consequently, before we focus on strategies for executing Linked Data queries, this section discusses relevant characteristics of the Web of Data.

Most importantly, the Web of Data is a virtually unbounded space; that is, in theory, looking up any randomly generated URI may result in retrieving Linked Data. Therefore, we cannot assume to ever have a complete list of all URIs based on which we would retrieve all Linked Data (even if the Web of Data would be static). Consequently, we also cannot assume that any system ever has access to all Linked Data that is –or was– (openly) available on the Web at a certain point in time.

To illustrate what the consequences of these limited data access capabilities are for Linked Data query execution, assume a query semantics that allows Linked Data queries to range over all Linked Data on the Web. Hartig formally analyzes an interpretation of SPARQL query patterns under such a "*full-Web semantics*" [17]. This analysis shows that not any approach for executing such SPARQL-based Linked Data queries can guarantee complete query results.

We emphasize, however, that instead of such a full-Web semantics a SPARQL query patterns may also be interpreted under an alternative, more restrictive query semantics. Several such semantics have been proposed [5, 14, 17], each of which restricts the range of Linked Data queries to a well-defined part of the Web of Data. A query execution approach that supports such a semantics can guarantee that for any query the computed result is the same as what is expected as the complete (and sound) result according to the corresponding semantics. While a first analysis of some of these more restrictive query semantics for SPARQL-based Linked Data queries exists [17], it requires more work to achieve a better understanding of the theoretical foundations of Linked Data query execution. Therefore, for most of the discussion in this article, we assume SPARQL-based Linked Data queries under full-Web semantics (with the caveat that completeness of query results cannot be guaranteed).

Another fundamental characteristic of the Web of Data is that the publication of Linked Data is not coordinated centrally. While such an absence of coordination is critical to the openness and the growth of the Web of Data, it introduces the following data integration issues:

- *Coreferencing:* Although URIs are used as globally unique identifiers for denoting entities in Linked Data, different providers may use different URIs for denoting the same entity. Some of the data about such a coreferenced entity will be ignored, as long as the coreference is not detected and resolved.
- *Schema heterogeneity:* Linked Data providers are free to choose the RDF vocabularies based on which they represent their data. Since different vocabularies may overlap w.r.t. the classes and properties that they define, a query expressed in terms of one vocabulary must be rewritten to benefit from data represented using a different vocabulary (alternatively, the data may be rewritten to match the vocabulary used by the query).

Existing work on executing Linked Data queries focuses mostly on the actual execution strategies (such as those discussed in the following sections of this article) and, so far, largely ignores the above-mentioned data integration issues.

In fact, we are only aware of a single proposal that deals with these issues in the context of Linked Data query execution: Umbrich et al. study extended query semantics for conjunctive Linked Data queries [39]. These semantics integrate i) lightweight RDFS reasoning for a fixed, a-priori defined set of vocabularies, and ii) inference rules for RDF triples with the predicate owl:sameAs [37]. The former partially addresses the issue of schema heterogeneity because it may exploit certain mappings between vocabularies. The latter

allows for making use of available information about coreferenced entities because owl:sameAs is commonly used to indicate coreferencing URIs in Linked Data [9].

In an empirical analysis, Umbrich et al. compare their extended query semantics to a baseline semantics that does not integrate inference rules. This analysis verifies that the number of solutions in a query result under the extended semantics is usually greater than the result for the corresponding query under the baseline semantics. The price for such an increase in "recall" [39] is an increase in average query execution times because, for a complete execution of queries under the extended semantics, it becomes necessary to look up more URIs than under the baseline semantics.

While Umbrich et al.'s work is the only proposal to date that addresses the aforementioned data integration issues explicitly in the context of Linked Data query execution, approaches proposed in related contexts might be adapted easily. Promising candidates for such an adaptation are proposals for query relaxation in RDF databases (e.g., [10,23,24]) and approaches for a mapping-based rewriting of queries over a federation of SPARQL endpoints (e.g., [25,28]).

In addition to characteristics of the Web that affect the more conceptual design of Linked Data query execution approaches, a number of data access specific issues must be taken into account when implementing such an approach into an actual system. In particular, any system that executes Linked Data queries must be able to deal with issues such as the following: Looking up certain URIs may result in the retrieval of an unforeseeable large set of RDF triples. Response times may vary significantly between different Web servers. Sometimes a URI lookup may take unexpectedly long or may not be answered at all. Some servers put restrictions on clients such as serving only a limited number of requests per second. Regarding the latter issue, it is important to emphasize that any Linked Data query execution system should implement a politeness policy to avoid overloading servers. In particular, any system should abide by the *robots.txt protocol*[3] that allows Web sites to demand delays between subsequent requests from the same client. For Web sites that do not provide a robots.txt file, a default minimum delay of, e.g., 500 ms [22,39] should be enforced.

We now come to our discussion of strategies for executing Linked Data queries. As mentioned in the introduction we first focus on strategies for data source selection.

3 Data Source Selection

For the execution of Linked Data queries, it is necessary to retrieve data by looking up URIs. There exist three classes of approaches for selecting the URIs that a query execution system looks up during the execution of a query: i) live exploration, ii) index-based approaches, and iii) hybrid approaches. In the following, we discuss each of these types.

3.1 Live Exploration Approaches

Live exploration approaches make use of the characteristics of the Web of Data, in particular, the existence of data links. In order to execute a given Linked Data query, live-exploration-based systems perform a recursive URI lookup process during which they incrementally discover further URIs that can be scheduled for lookup. Thus, such a system explores the Web of Data by traversing data links at query execution time. While the data retrieved during such an exploration allows for a discovery of more URIs to look up, it also provides the basis for constructing the query result. The starting point for a live-exploration-based query execution usually is a set of seed URIs; these seed URIs may be the URIs mentioned in the given query, or they are specified as an accompanying parameter for the query.

A live-exploration-based system may not need to look up all URIs discovered. Instead, certain live exploration approaches may (directly or indirectly) introduce criteria to decide which of the discovered URIs are scheduled for lookup. For instance, approaches designed to support one of the more restrictive query semantics (mentioned in Section 2) may ignore any URI whose lookup exceeds the part of the Web that is relevant according to the semantics.

We notice that live-exploration-based query execution is similar to focused crawling as studied in the context of search engines for the WWW [6,2]. However, in focused crawling a discovered URI qualifies for lookup because of a high relevance for a specific topic; in live-exploration-based query execution, the relevance is more closely related to the task of answering the query at hand. Furthermore, the purpose of retrieving Web content is slightly different in both cases: Focused crawling, or Web crawling in general, is a pre-runtime (or background) process during which a system populates a search index or a local database; then, the runtime component of such a system provides query access to the populated data structure. Live exploration approaches, in contrast, retrieve data to answer a particular query; in these approaches, link-traversal-based data retrieval is an essential part of the query execution process itself. Nonetheless, implementation techniques used for focused crawling, such as context graphs [8], might be applied in a live exploration approach for Linked Data query execution.

The most important characteristic of live exploration approaches is the possibility to use data from initially unknown data sources. This characteristic allows for serendipitous discovery and, thus, enables applications that tap the full potential of Linked Data on the Web. Another characteristic is that live exploration approaches might be used to develop query

[3] http://www.robotstxt.org/

execution systems that do not require any a-priori information about the Web of Data. Consequently, such a system might readily be used without having to wait for the completion of an initial data load phase or any other type of preprocessing. Hence, live exploration approaches are most suitable for an "on-demand" querying scenario. On the downside, data access times inherently add up due to the recursive nature of the lookup process. Possibilities for parallelizing data retrieval are limited because relevant URIs only become available incrementally. Another limitation of live exploration approaches is their inherent dependency on the structure of the network of data links as well as on the number of links. In a Web sparsely populated with links, chances are low to discover relevant data. A system may report more complete results for certain queries (under full-Web semantics), if it uses other source selection approaches.

In its purest form, live exploration approaches assume query execution systems that do not have any a-priori information about the Web of Data [16]. It is also possible, however, that a query execution system reuses data retrieved during the execution of a query as a basis for processing subsequent queries. Such a reuse can be beneficial for two reasons [15]: 1) it can improve query performance because it reduces the need to retrieve data multiple times; 2) assuming full-Web semantics, it can provide for more complete query results, calculated based on data from data sources that would not be discovered by a live exploration starting without previously retrieved data. However, since reusing retrieved data for the execution of multiple queries is a form of data caching, it requires suitable caching strategies. In particular, any system that keeps previously retrieved data has to apply an appropriate invalidation strategy; otherwise, it could lose the advantage of up-to-date query results. As an alternative to caching retrieved data, it is also possible to only keep a summary of the data or certain statistics about it. Such information may then be used to guide the execution of later queries (as in the case of index-based source selection approaches which we discuss in the following).

3.2 Index-Based Approaches

Index-based approaches ignore the existence of data links during the query execution process itself. Instead, these approaches rely on a pre-populated index which is used for identifying URIs to look up during query execution time. Hence, in contrast to index structures that store the data itself (such as the original B-tree or existing approaches for indexing RDF data [12,32,42]), the index-based approaches discussed here use data structures that index URIs as pointers to data; each of these URIs may appear multiple times in such an index because the data that can be retrieved using such a URI may be associated with multiple index keys.

A typical example for such a data structure uses triple patterns as index keys [26]. Given such a pattern, the corresponding index entry is a set of URIs such that looking up each of these URIs provides us with some data that contains a matching triple for the pattern. To enable data source ranking (discussed in the following Section 4), index entries may additionally encode the cardinality of matching triples for each indexed URI [13,26,40]. Thus, such an index presents a summary of the data available from all indexed URIs.

Source selection using such an index is based on relevance [26,38]: A URI is *relevant* for a given query if the data retrieved by looking up the URI contributes to the query result. However, the existence of a triple that matches a triple pattern from the query is not sufficient to make the corresponding URI relevant; only if such a matching triple contributes to an element of the query result, the URI is relevant.

Given that data from irrelevant URIs is not required to compute a query result, avoiding the lookup of such URIs reduces the cost of query executions significantly [13,40, 26,33]. Consequently, the focus of research in this context is to identify a subset of all (indexed) URIs that contains all relevant URIs and as few irrelevant ones as possible. While simpler approaches consider any triple pattern of a given query separately [33], more sophisticated approaches achieve a higher reduction of irrelevant URIs by taking into account joins between triple patterns [13,40,26,38].

We note that these index-based approaches are closer in spirit to traditional query processing techniques than live exploration approaches. Existing data summarization and indexing techniques may be adapted to develop an index-based approach for Linked Data query execution. For instance, Umbrich et al. adopt the concept of multidimensional histograms as a data summary for index-based Linked Data query execution [40] (the original use case for multidimensional histograms is an estimation of selectivities for multidimensional queries). Similarly, the QTree that Harth et al. use as a summary of Linked Data [13] is a combination of a histogram and an R-tree (the latter was originally proposed to index data about spatial objects).

Further data structures for index-based Linked Data query execution are proposed in the literature: In contrast to the aforementioned approach of using triple patterns as index keys, Tian et al. extract frequently used combinations of triple patterns from a given query workload and use unique encodings of these combinations as index keys [38]. For a query workload that is similar to the workload used for building an index, the authors show that their approach can prune more irrelevant URIs than the baseline approach of using triple patterns as index keys. An *inverted URI index* is another, very simple index structure [40]. In this case, the index keys are URIs, namely, the URIs mentioned in the data that can be retrieved by looking up the indexed URIs. In another approach, the index keys are properties

and classes from ontologies used for the data [33]. Umbrich et al. refer to this approach as *schema-level indexing* [40]. In their work, the authors compare the application of an inverted URI index, schema-level indexing, the aforementioned QTree, and a multidimensional histogram, for an index-based Linked Data query execution [40].

Existing work on index-based Linked Data query execution usually assumes that the set of URIs to be indexed is given. To build the index for such a set, it is necessary to retrieve the data for any given URI. Instead of populating the index based on a given set of URIs, it is also possible to build such an index using the output of a Web crawler for Linked Data (for a detailed discussion of crawling Linked Data, we refer to Hogan et al. [22]). Alternatively, populated indexes may also be a by-product of executing queries using a live exploration approach. However, in all these cases, an initial lookup of all indexed URIs is required before the index can be used for executing Linked Data queries.

After populating an initial version of an index, it is necessary to maintain such an index. Maintenance may include adding additionally discovered URIs and keeping the index up to date. The latter is necessary because what data can be retrieved from indexed URIs might change over time. While Umbrich et al. address this topic briefly [40], no work exists that discusses index maintenance for index-based Linked Data query execution in detail. However, the challenges are similar to the problems addressed in the contexts of: index maintenance in information retrieval, index maintenance for traditional Web search engines, Web caching, maintenance of data(base) caches, and view maintenance in databases.

The most important characteristic of index-based source selection is the ability to determine all potentially relevant URIs at the beginning of a query execution. This ability enables query execution systems to fully parallelize data retrieval. Such a parallelization might reduce data retrieval time for executing a query. As a result, an efficiently implemented index-based system might answer a Linked Data query faster than a live-exploration-based system (assuming both systems eventually look up the same set of URIs during the execution).

On the other hand, a live-exploration-based system is ready for use immediately, whereas an index-based system can only be used after initializing its index. Such an initialization may take a significant amount of time assuming that the system has to retrieve the data for all indexed URIs first. From the aforementioned publications, only Paret et al. take the initial retrieval time into account for the evaluation of their approaches [33]. Unfortunately, the actual setup of the corresponding experiments is not clear in this work; in particular, missing information about response times of the dedicated Web servers used for the experiment and about the number of URIs looked up, prohibit drawing conclusions from the reported measurements. However, for systems that use crawling to populate their index, we may get an idea of the initial data retrieval time by looking into related work. In particular, in their work on a data warehouse for Linked Data, Hogan et al. report the following measurements [22]: For crawling 1,000 URIs (and 100,000 URIs) with 64 threads on a single machine they report an overall crawl time of about 9 minutes (and 360 minutes); in a distributed setting, 8 machines with 64 threads each, crawl 100,000 URIs in about 63 minutes.

Another advantage of index-based approaches claimed in the literature is the ability to report query results that are more complete when compared to live exploration approaches [13]. However, the authors' understanding of completeness remains unclear, because they do not provide a precise definition of query semantics for the Linked Data queries executed by their approach (the same holds for almost all of the existing publications that propose approaches for executing Linked Data queries). However, if we assume full-Web semantics, it is indeed possible that an index-based approach computes some solutions of a query result that a live exploration approach cannot compute; this is the case if some data necessary for computing these solutions cannot be discovered by link traversal. On the other hand, a live-exploration-based execution may discover and look up URIs that are not indexed; the data retrieved by these lookups may allow for the computation of certain query solutions. In such a case, the corresponding index-based execution cannot compute these solutions. Hence, a general statement about the superiority of an index-based approach over a live exploration approach (or vice versa) w.r.t. result completeness is not possible in the context of full-Web semantics. Empirical studies that compare the suitability of both strategies for different use cases remain to be conducted.

Finally, we also emphasize that the aforementioned notion of relevance for URIs should not be carried over directly to live exploration approaches (or used in a comparison of both strategies). For a live-exploration-based query execution, the retrieval of data is not only necessary to obtain matching triples that contribute to the query result; instead, such data may also allow a query execution system to discover (and, thus, traverse) data links, through which the system may eventually obtain additional matching triples.

3.3 Hybrid Approaches

Hybrid source selection approaches combine an index-based approach with a live exploration approach and, thus, aim to achieve the advantages of both approaches without inheriting their respective shortcomings. For instance, a hybrid approach may use an index to determine a suitable set of seed URIs as input for a subsequent live exploration process. This process may than feed back information for updating, for expanding, or for reorganizing the index. An alternative

idea is a hybrid approach that uses an index only for prioritizing discovered data links and, thus, for controlling a live exploration process.

To the best of our knowledge, the only query execution strategy that implements such a hybrid approach is the "mixed strategy" proposed by Ladwig and Tran [26]. This strategy is based on a ranked list of URIs to look up. To obtain an initial version of this list for a given query, the approach exploits an index. Additional URIs discovered during query execution are then integrated into the list. Ladwig and Tran's main technical contribution is an adaptive source ranking approach which we discuss in the following section.

4 Data Source Ranking

Instead of merely selecting URIs to look up, execution strategies for Linked Data queries may rank the resulting set of URIs such that the ranking represents a priority for looking up any (selected) URI. Such a data source ranking may allow a query execution system to i) report solutions of a query result as early as possible, ii) reduce the time for computing the first k solutions, or iii) compute the maximum number of solutions in a given amount of time [26]. In this section, we briefly summarize existing work on source ranking for Linked Data query execution.

Harth et al. complement their index-based source selection approach with source ranking [13]. Selected URIs are ranked using an estimate for the number of query solutions that the data of each URI contributes to. The basis for estimating these numbers are cardinalities recorded in the QTree entries for each (indexed) URI. By using the QTree, a query execution system might obtain an estimate of how many matching triples for a given triple pattern are available from each URI. Based on these triple-pattern-specific estimates, Harth et al. compute the estimates for the whole query recursively. For the computation, the authors take join cardinality estimations into account.

Ladwig and Tran introduce a ranking approach that includes multiple scores [26]:

1. *Triple frequency - inverse source frequency (TF-ISF).* TF-ISF is an adaptation of the well-known TF-IDF measure used in information retrieval; Ladwig and Tran define TF-ISF for a pair of an indexed URI and a triple pattern (from the query). The computation of such a TF-ISF measure is based on the triple-pattern-specific cardinality of the corresponding URI (that is, how many matching triples for the pattern can be obtained by looking up the URI). Similar to Harth et al. [13], Ladwig and Tran's approach obtains these cardinalities from an index. However, due to a different index structure used by Ladwig and Tran, the obtained cardinalities are accurate (in Harth et al., they are estimates only, because the QTree is an approximate index structure [13]).
2. *(URI-specific) join cardinality estimates.* For each URI, this score approximates the number of query solutions that can be computed using only the data from the given URI. While this score is similar to the estimates that Harth et al. use for ranking, it neglects joins based on data from multiple URIs. For the computation of such a (URI-specific) join cardinality estimate, Ladwig and Tran propose an approach that, again, uses the triple-pattern cardinalities recorded in their index, as well as partial query solutions (generated using a random sample of data already retrieved during the query execution).
3. *In-link scores.* This score is calculated based on incoming data links, that is, references to a given URI in the data available from other URIs. Interlinkage information about a set of URIs may be recorded in an additional data structure that complements the main index used for index-based source selection. However, Ladwig and Tran propose to obtain such information during the execution of a query. As a result, the in-link score can also be used to rank URIs selected by live exploration approaches.

Some of the input for calculating the aforementioned scores may be refined based on information that gradually becomes available during the query execution process. For instance, the data used to obtain samples for join cardinality estimation grows; similarly, additional interlinkage information becomes available. As a consequence, Ladwig and Tran propose a regular recalculation of scores and resulting ranks. Based on certain thresholds, the authors study the trade-off between the benefits of a more accurate ranking that can be achieved by recalculating scores more frequently and the higher costs incurred by a more frequent recalculation [26].

The source ranking approaches as summarized above are designed to achieve the objectives mentioned in the beginning of this section. We note that similar objectives characterize the problem of top-k query processing where only the k top-ranked result elements need to be computed. Wagner et al. study top-k processing for Linked Data queries [41]. In particular, the authors propose a top-k approach for the index-based source selection strategy. Although this approach is about ranking (intermediate) result elements, it enforces an indirect ranking of the URIs to look up. The additional information that is necessary for this approach is, again, assumed to be recorded in the pre-populated index.

5 Integration of Data Retrieval and Result Construction

The actual process of executing a Linked Data query may consist of two separate phases: During the first phase, a query execution system selects URIs and uses them to retrieve data from the queried Web (as discussed before); during a subse-

quent, second phase, the system generates the query result using the data retrieved in the first phase. Instead of separating these two phases, it is also possible to integrate the retrieval of data into the result construction process. Hereafter, we use the term *integrated execution approaches* to refer to Linked Data query execution approaches that apply the latter idea. Analogously, *separated execution approaches* clearly separate data retrieval from result construction by two consecutive phases. We emphasize that separating or integrating data retrieval and result construction is a design decision that is orthogonal to what source selection strategy (and source ranking strategy) is used for a particular Linked Data query execution approach. In the following, we discuss both types of approaches, separated execution and integrated execution.

5.1 Separated Execution Approaches

Due to the clear separation of data retrieval and result construction, separated approaches are straightforward to implement. In particular, index-based source selection lends itself naturally to such an implementation. Consequently, most of the aforementioned publications on index-based approaches assume a two-phase execution strategy [13, 40, 33]. However, it is also easy to develop a separated execution approach based on live exploration: During the first phase, a query execution system retrieves data by recursively traversing all data links that qualify according to the lookup criterion applied; during the second phase, the system generates the query result.

The downside of separated execution approaches is that the query execution system can report first elements of the query result only after it has completed the data retrieval phase. Looking up a large set of selected URIs or retrieving the complete set of reachable data may exceed the resources of an execution system or it may take a prohibitively long time. We note that a combination of data source ranking and thresholds, for constraining the overall time of the data retrieval phase or the maximum number of lookups, may address these problems partially (for the price of missing some elements of the query result).

5.2 Integrated Execution Approaches

Integrated approaches may allow a query execution system to report first solutions for a (monotonic) query early, that is, before data retrieval has been completed. Furthermore, certain integrated approaches may require significantly less query-local memory than any separated execution approach; this may hold in particular for approaches that process retrieved data in a streaming manner and, thus, do not require to store all retrieved data until the end of a query execution process. For instance, Ladwig and Tran introduce a symmetric-hash-join-based approach which presents such a case [26, 27]. However, although this approach may not need to hold retrieved data, it requires query-local memory for materializing all intermediate query results (and, depending on the query, only a certain fraction of all these intermediate results may eventually become a part of the overall query result). Nonetheless, it is reasonable to assume that for most Linked Data queries the overall memory required for holding all intermediate query results is smaller than the memory required for holding all discovered data. To the best of our knowledge, an analysis of the trade-offs of materializing intermediate results vs. holding all retrieved data has not been conducted for Linked Data query execution approaches.

As for the separated execution strategy discussed before, it is possible to use any type of source selection (and source ranking) as a basis for an integrated execution approach. A manifold of combinations are conceivable; in particular, live exploration may be combined with an integrated execution approach in a multitude of ways. In fact, this combination is the combination most widely studied in the literature. We refer to query executions that present such a combination as *link traversal based query executions (LTBQE)*, and discuss them separately in the following section.

5.3 Link Traversal Based Query Execution (LTBQE)

Before we refer to particular LTBQE approaches proposed in the literature, we first outline a naive example of such an approach in order to illustrate the idea of combining live-exploration-based source selection with an integrated execution strategy: Using a set of seed URIs as a starting point, a query execution system may alternate between two types of execution stages, that are, link traversal stages and result computation stages. Each link traversal stage consists of looking up URIs that the system finds in the data retrieved during the previous link traversal stage. During the result computation stage that follows such a link traversal stage, the system generates a temporary, potentially incomplete query result using all data retrieved so far; from such a result, the system reports those solutions that did not appear in the result generated during the previous result computation stage. Hence, the system incrementally explores the queried Web of Data in a breadth-first manner and produces more and more elements of the query result during that process. We emphasize that such a naive, breadth-first LTBQE approach is unsuitable in practice, because completely recomputing a partial query result during each result computation stage is not efficient.

Schmedding proposes a version of the naive approach that addresses this problem [36]. The idea of Schmedding's LTBQE approach is to recursively adjust the currently computed query result each time the execution system retrieves

additional data. Schmedding's main contribution is an extension of the SPARQL algebra operators that makes the differences between query results computed on different input data explicit; by using the extended algebra, a query execution system might compute a query result using only i) the additionally retrieved data and ii) the previously computed result (instead of recomputing everything from scratch).

In an early research publication on the topic, Hartig et al. describe the idea of LTBQE using an approach that presents an even tighter integration of link traversal and result construction than the two aforementioned approaches [18]. Instead of performing multiple result computation stages that always compute a whole query result (from scratch as in a the naive approach or incrementally as proposed by Schmedding), the authors introduce a strategy for executing a single result construction process only; this process computes the solutions of a query result by incrementally augmenting intermediate solutions such that these intermediate solutions cover more and more triple patterns of the query. For such an augmentation, the process uses matching triples from data retrieved via link traversal. At the same time, the process uses the URIs in these matching triples for further link traversal. Hence, this strategy deeply intertwines result construction and link traversal. For a more detailed description of the process, we refer to [18, 20].

We emphasize that LTBQE as described by Hartig et al. presents a general strategy rather than a concrete, implementable algorithm. In a more recent publication, Hartig and Freytag provide a formal, implementation-independent definition of this LTBQE strategy and use this definition to formally analyze the strategy [19]. A variety of approaches for implementing this strategy are conceivable. Implementation approaches studied so far can be summarized as follows:

- Hartig et al. themselves study an implementation that uses a synchronized pipeline of iterators [18]. In a follow-up paper on this implementation, Hartig proposes a heuristics-based approach for query planning [16].
- Ladwig and Tran make use of symmetric hash joins, as mentioned before. More precisely, their implementation approach employs an asynchronous, push-based pipeline of symmetric hash join operators [26]. In later work, the authors extend this approach and introduce the *symmetric index hash join* operator. This operator allows a query execution system to incorporate a query-local RDF data set into the query execution [27].
- Miranker et al. introduce another push-based implementation [30]. The authors implement LTBQE using the well-known Rete match algorithm.

Since LTBQE approaches combine an integrated execution and live-exploration-based source selection they inherit the advantages and limitations of these two strategies (as discussed in Sections 3.1 and 5.2). That is, as all live-exploration-based systems, LTBQE systems are able to make use of data from initially unknown data sources and can readily be used without first populating and maintaining supporting data structures. Furthermore, an LTBQE system can be built to report first solutions early. On the downside, data retrieval may not be parallelized as effectively as is possible with index-based source selection; moreover, a sparsity of data links reduces the chances for discovering potentially relevant data and may thus result in missing a larger number of solutions for queries under full-Web semantics.

6 Conclusions

In this article, we have provided a general overview on the new field of Linked Data query execution. We have introduced three dimensions along which approaches for executing Linked Data queries can be classified. Each of these dimensions captures a particular aspect of a full Linked Data query execution approach. In the context of these dimensions, we have discussed different strategies that possible Linked Data query execution approaches may apply. Furthermore, we have outlined how the discussed strategies are implemented in the Linked Data query execution approaches that have been proposed in the literature so far. We conclude this overview by classifying these proposed approaches along our three dimensions:

Publication	Source Selection	Source Ranking	Integr. Exec.
Harth et al. [13,40]	index-based	yes	no
Hartig et al. [18,16,19]	live expl.	no	yes
Ladwig and Tran [26] ("bottom up")	live expl.	yes	yes
Ladwig and Tran [26] ("top down")	index-based	yes	yes
Ladwig and Tran [26] ("mixed strategy")	hybrid	yes	yes
Ladwig and Tran [27]	live expl.	no	yes
Miranker et al. [30]	live expl.	no	yes
Paret et al. [33]	index-based	no	no
Schmedding [36]	live expl.	no	yes
Tian et al. [38]	index-based	no	n/a
Umbrich et al. [40] (multidim. histograms)	index-based	yes	no
Umbrich et al. [40] (schema-level index)	index-based	no	no
Umbrich et al. [40] (inverted URI index)	index-based	no	no
Wagner et al. [41]	index-based	yes	yes

Table 1 Classification of existing work on Linked Data query execution along the dimensions of i) data source selection, ii) data source ranking, and iii) integration of data retrieval and result construction.

References

1. B. Adida, M. Birbeck, S. McCarron, and I. Herman. RDFa Core 1.1 – Syntax and Processing Rules for Embedding RDF through Attributes. W3C Rec., Online: http://www.w3.org/TR/rdfa-core/.

2. S. Batsakis, E. G. M. Petrakis, and E. Milios. Improving the Performance of Focused Web Crawlers. *Data & Knowledge Engineering*, 68(10):1001–1013, 2009.
3. T. Berners-Lee. Design Issues: Linked Data. Online at http://www.w3.org/DesignIssues/LinkedData.html.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *Int. Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.
5. P. Bouquet, C. Ghidini, and L. Serafini. Querying The Web Of Data: A Formal Approach. In *Proc. of the 4th Asian Semantic Web Conference (ASWC)*, 2009.
6. S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.
7. K. G. Clark, L. Feigenbaum, and E. Torres. SPARQL Protocol for RDF. W3C Rec., Online at http://www.w3.org/TR/rdf-sparql-protocol/, Jan. 2008.
8. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused Crawling Using Context Graphs. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB)*, 2000.
9. L. Ding, J. Shinavier, Z. Shangguan, and D. L. McGuinness. SameAs Networks and Beyond: Analyzing Deployment Status and Implications of owl:sameAs in Linked Data. In *Proc. of the 9th International Semantic Web Conference (ISWC)*, 2010.
10. P. Dolog, H. Stuckenschmidt, H. Wache, and J. Diederich. Relaxing RDF Queries based on User and Domain Preferences. *Journal of Intelligent Information Systems*, 33(3):239–260, 2009.
11. O. Görlitz and S. Staab. Federated Data Management and Query Optimization for Linked Open Data. In *New Directions in Web Data Management 1*, pages 109–137. 2011.
12. A. Harth and S. Decker. Optimized Index Structures for Querying RDF from the Web. In *Proc. of the 3rd Latin American Web Congress (LA-Web)*, 2005.
13. A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data Summaries for On-Demand Queries over Linked Data. In *Proc. of the 19th Int. Conf. on World Wide Web (WWW)*, 2010.
14. A. Harth and S. Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proc. of the 26th AAAI Conference*, 2012.
15. O. Hartig. How Caching Improves Efficiency and Result Completeness for Querying Linked Data. In *Proc. of the 4th Linked Data on the Web workshop (LDOW)*, 2011.
16. O. Hartig. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *Proc. of the 8th Extended Semantic Web Conference (ESWC)*, 2011.
17. O. Hartig. SPARQL for a Web of Linked Data: Semantics and Computability. In *Proc. of the 9th Extended Semantic Web Conference (ESWC)*, 2012.
18. O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL Queries over the Web of Linked Data. In *Proc. of the 8th International Semantic Web Conference (ISWC)*, 2009.
19. O. Hartig and J.-C. Freytag. Foundations of Traversal Based Query Execution over Linked Data. In *Proc. of the 23rd ACM Conference on Hypertext and Social Media (HT)*, 2012.
20. O. Hartig and A. Langegger. A Database Perspective on Consuming Linked Data on the Web. *Datenbank-Spektrum*, 10(2), 2010.
21. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011.
22. A. Hogan, A. Harth, J. Umrich, S. Kinsella, A. Polleres, and S. Decker. Searching and Browsing Linked Data with SWSE: the Semantic Web Search Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4), 2012.
23. A. Hogan, M. Mellotte, G. Powell, and D. Stampouli. Towards Fuzzy Query-Relaxation for RDF. In *Proc. of the 9th Extended Semantic Web Conference (ESWC)*, 2012.
24. H. Huang, C. Liu, and X. Zhou. Approximating Query Answering on RDF Databases. *World Wide Web*, 15(1):89–114, 2012.
25. A. K. Joshi, P. Jain, P. Hitzler, P. Z. Yeh, K. Verma, A. P. Sheth, and M. Damova. Alignment-based Querying of Linked Open Data. In *Proc of the 11th Int. Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, 2012.
26. G. Ladwig and D. T. Tran. Linked Data query processing strategies. In *Proc. of the 9th International Semantic Web Conference (ISWC)*, 2010.
27. G. Ladwig and D. T. Tran. SIHJoin: Querying Remote and Local Linked Data. In *Proc. of the 8th Extended Semantic Web Conference (ESWC)*, 2011.
28. K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources. In *Proceedings of OTM Conferences*, 2010.
29. P. Mika and T. Potter. Metadata Statistics for a Large Web Corpus. In *Proc. of the 5th Linked Data on the Web Workshop (LDOW)*, 2012.
30. D. P. Miranker, R. K. Depena, H. Jung, J. F. Sequeda, and C. Reyna. Diamond: A SPARQL Query Engine, for Linked Data Based on the Rete Match. In *Proc. of the Workshop on Artificial Intelligence meets the Web of Data (AImWD)*, 2012.
31. H. Mühleisen and C. Bizer. Web Data Commons – Extracting Structured Data from Two Large Web Corpora. In *Proc. of the 5th Linked Data on the Web Workshop (LDOW)*, 2012.
32. T. Neumann and G. Weikum. RDF-3X: a RISC-style Engine for RDF. In *Proc. of the 34th Int. Conf. on Very Large Data Bases (VLDB)*, 2008.
33. E. Paret, W. Van Woensel, S. Casteleyn, B. Signer, and O. De Troyer. Efficient Querying of Distributed RDF Sources in Mobile Settings based on a Source Index Model. *Procedia CS*, 2011.
34. E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 Federated Query. W3C Rec., Online at http://www.w3.org/TR/sparql11-federated-query/, Mar. 2013.
35. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Rec., Online at http://www.w3.org/TR/rdf-sparql-query/, Jan. 2008.
36. F. Schmedding. Incremental SPARQL Evaluation for Query Answering on Linked Data. In *Proc. of the 2nd Int. Workshop on Consuming Linked Data (COLD)*, 2011.
37. M. Schneider. OWL 2 Web Ontology Language, RDF-Based Semantics (Second Edition). W3C Rec., Online at http://www.w3.org/TR/owl2-rdf-based-semantics/, Dec. 2012.
38. Y. Tian, J. Umbrich, and Y. Yu. Enhancing Source Selection for Live Queries over Linked Data via Query Log Mining. In *Proc. of the Joint Int. Semantic Technology Conference (JIST)*, 2011.
39. J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Improving the Recall of Live Linked Data Querying through Reasoning. In *Proc. of the 6th Int. Conference on Web Reasoning and Rule Systems (RR)*, 2012.
40. J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing Data Summaries for Processing Live Queries over Linked Data. *World Wide Web*, 14(5–6):495–544, 2011.
41. A. Wagner, T. Tran, G. Ladwig, and A. Harth. Top-K Linked Data Query Processing. In *Proc. of the 9th Extended Semantic Web Conference (ESWC)*, 2012.
42. C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. In *Proc. of the 34th International Conference on Very Large Data Bases (VLDB)*, 2008.