International Journal on Semantic Web and Information Systems
Volume 12 • Issue 3 • July-September 2016

Hypermedia-Based Discovery for Source Selection Using Low-Cost Linked Data Interfaces

Miel Vander Sande, Data Science Lab, Ghent University - iMinds, Ghent, Belgium

Ruben Verborgh, Data Science Lab, Ghent University - iMinds, Ghent, Belgium

Anastasia Dimou, Data Science Lab, Ghent University - iMinds, Ghent, Belgium

Pieter Colpaert, Data Science Lab, Ghent University - iMinds, Ghent, Belgium

Erik Mannens, Data Science Lab, Ghent University - iMinds, Ghent, Belgium

ABSTRACT

Evaluating federated Linked Data queries requires consulting multiple sources on the Web. Before a client can execute queries, it must discover data sources, and determine which ones are relevant. Federated query execution research focuses on the actual execution, while data source discovery is often marginally discussed—even though it has a strong impact on selecting sources that contribute to the query results. Therefore, the authors introduce a discovery approach for Linked Data interfaces based on hypermedia links and controls, and apply it to federated query execution with Triple Pattern Fragments. In addition, the authors identify quantitative metrics to evaluate this discovery approach. This article describes generic evaluation measures and results for their concrete approach. With low-cost data summaries as seed, interfaces to eight large real-world datasets can discover each other within 7 minutes. Hypermedia-based client-side querying shows a promising gain of up to 50% in execution time, but demands algorithms that visit a higher number of interfaces to improve result completeness.

KEYWORDS

Discovery, Hypermedia, Linked Data, Query Federation, REST, Semantic Web, SPARQL

INTRODUCTION

The Web is a fully distributed system—and thus so is the Web of Data. Within this enormous collection, each data source specializes in its very own part of the truth. Some of them, like DBpedia1, contain essential facts about a broad range of subjects; others, like Drugbank2, offer a comprehensive corpus of triples about highly select topics. As a result, in order to answer any non-trivial query over the Web of Data, we likely need to consult multiple data sources. The need for such federated queries intensifies as the Linked Open Data cloud is trending toward a more decentralized graph structure, with additional linking hubs besides DBpedia arising (Schmachtenberg et al., 2014). Federation is thus necessary to achieve the Web of Data vision (Heath & Bizer, 2011): a global, machine-understandable dataspace with web-scale integration and interoperability.

In literature, the story of federated query evaluation is typically told from source selection onwards: given a fixed set of available data sources, a client determines which of these are necessary to obtain results. After that, the actual query processing against the selected sources happens. However, before any of this can take place, candidate data sources need to be located first. This process preceding

DOI: 10.4018/IJSWIS.2016070103

source selection has hardly received rigorous scientific study so far. In general, discovery is the process of finding available Linked Data sources that are relevant to a certain task, for specific definitions of "relevance" and "task". Although the description of dataset or endpoint characteristics has been covered, the act of finding, accessing, and processing such documents is still in its infancy. With the emerging Web Of Data, studying autonomous Linked Data discovery becomes a need, with a special focus on the impact on client-side tasks such as querying. For federated query execution in particular, discovery can assist in a more complete selection of accessed data sources.

Therefore, this article studies the impact of Linked Data interface discovery on federated querying. We consider any that provides client access to Linked Data sources. In total, we present three contributions.

First, we propose a discovery technique, which leverages hypermedia between Linked Data interfaces. Hypermedia allows such interfaces to function similarly to a webpage, providing the user with guidance on what type of content they can retrieve, or what actions they can perform, as well as the appropriate links to do so. Since the beginning of the Web, this has been the crucial aspect to the Web's scalability. Existing discovery works have greatly progressed in closed, custom p2p networks using custom discovery protocols, or centralized repositories that crawl metadata from different sources. However, with a scale-free network at our disposal, little of its benefits have been exploited for Linked Data querying. The novelty of our approach lies in strictly reusing hypermedia and Linked Data principles to a) discover one another, aided by links in a dataset; and b) inform the client at run-time about their discoveries through hypermedia. Furthermore, clients and servers distribute the processing cost fairly, resulting in a sustainable and scalable solution.

Second, to appropriately evaluate discovery approaches, we introduce a methodology to quantify its parameters. This includes metrics to express the functional and non-functional characteristics of one discovery approach relative to others.

Third, we implement and evaluate the approach against the lightweight Triple Pattern Fragments interface (Verborgh et al., 2014; 2016), and measure to what extent our discovery method facilitates source selection in federated query execution. We intend to enable querying multiple sources on the client while obtaining far less information than heuristics or dataset profiles.

The remainder of this paper is structured as follows. We first list a number of research questions with corresponding hypotheses and discuss related work. Then, we propose the metrics for evaluating discovery approaches. Next, we introduce a hypermedia-based discovery method applied to Triple Pattern Fragments and discuss how clients can use the outcome in federated query execution. After that, we evaluate our approach and analyze the results to assess its viability. Finally, we end with an overall conclusion and discuss future work.

PROBLEM STATEMENT

This paper tackles the practical problem of discovering Linked Data interfaces in service of federated query execution. Since such interfaces are scattered across the Web, which is connected through hyperlinks, we pose the following research question:

Question 1: Can we effectively discover distributed Linked Data interfaces by relying solely on hypermedia and Linked Data principles?

As a result, we will investigate an approach where Linked Data interfaces locate each other by a) actively following links to others, and b) reacting accordingly when being discovered. We cover several aspects of Question 1 with the following subquestions.

Question 1.1: How does the proposed discovery approach impact client and server load?

Question 1.2: Is the discovery method able to discover all relevant interfaces?
Question 1.3: Does the execution time of the discovery process scale with the size of the dataset?

In addition, we also study the effects of source discovery on federated query processing, since both processes are not necessarily independent. An effective discovery approach benefits the source selection stage it precedes. Hence, we pose the following research question.

Question 2: To what extent can the discovery of Linked Data interfaces through hypermedia facilitate source selection in federated query processing?

Source selection can be impacted on several aspects. Common are completeness, accuracy, and execution time. However, we are also interested in the load imposed on the server, i.e. the response times of requests used during query execution. Thereby, to accurately answer Question 2, we formulate three additional subquestions:

Question 2.1: Does hypermedia-based source selection retain recall compared to a naive baseline?
Question 2.2: How does hypermedia-based source selection impact the server's responsiveness?
Question 2.3: Does hypermedia-based source selection lower query execution time compared to a naive client-side approach?

Corresponding hypotheses and experiments for these questions are stated in the evaluation section, after introducing the metrics and the proposed approach.

RELATED WORK

In this section, we discuss work related to Linked Data discovery. First, we investigate the different methods to publish Linked Data on the Web. Next, we zoom in on how existing discovery approaches work and what description methods they use. Then, we look at how federated query processing, the most common application of discovery, performs source selection on the output of the discovery process. Finally, we briefly consider the discovery of services, which has so far been studied more deeply than the discovery of Linked Data.

Publishing Linked Data on the Web

When aiming to discover Linked Data on the Web, we need to be aware of possible publication methods, which need to be understood in the context of the Web's design. The Web is a distributed hypermedia system, characterised by the architectural properties derived by Fielding (2000). Its core differentiating feature is the uniform interface, implemented on the Web through the following four constraints:

1. Resources are identified by http uris.
2. Each resource can be accessed as zero or more representations (in formats such as html or rdf).
3. All exchanged messages are self-descriptive.
4. Navigation between resources happens through hypermedia controls such as links or forms.

Each publication method has its own way of partitioning a dataset in resources, as we will discuss in the following.

Data Dump

Conceptually, the simplest way to publish a dataset is a data dump: all triples in the dataset are serialized in one or more representations, which are possibly compressed into an archive. These archives are then made available from a specific http uri, often organized by version, to which other resources can link. In order to query this data, it must be downloaded (or streamed) in its entirety and processed by a local query processor. The rest resource here is thus the whole dataset; for discovery, this means we need to identify this one resource.

Linked Data Documents

Discovery changes when datasets are published in a more granular way according to the four Linked Data principles by Berners-Lee (2006):

1. Use uris as names for things.
2. Use http uris so that people can look up those names.
3. When someone looks up a uri, provide useful information, using Web standards.
4. Include links to other uris so that they can discover more things.

Note the parallels between these principles and the rest architectural constraints listed above: the Linked Data principles are a specific instance of the rest constraints with representation formats and its uris as hypermedia links. In particular, dereferencing is important: if a certain resource mentions a concept, we can discover more information about this concept by following its url. The rest notion of a “resource” thus coincides with the rdf notion of a “resource”. For example, data about the resource http://dbpedia.org/resource/Walt_Disney is available through the rest resource with that same url.

Note that dereferencing allows a discovery process to find related data from one particular authoritative source only. For instance, the aforementioned url allows to discover information about Walt Disney from the DBpedia dataset, but not from any other sources with information about the same concept. Furthermore, discovery of one resource comes down to merely retrieving its entire representation, and does not directly give access to all resources in the dataset.

SPARQL Endpoints

The sparql query language (Harris & Seaborne, 2013) allows a much more fine-grained selection of rdf data through highly specific and flexible custom queries. The sparql protocol (Feigenbaum et al., 2013) exposes result resources of such queries through http. For example, the result of the query SELECT DISTINCT ?t { ?s rdf:type ?t } executed on a dataset could be available as the resource http://dbpedia.org/sparql?query=SELECT+DISTINCT+%3Ft+%7B+%3Fs+rdf%3Atype+%3Ft+%7D. Discovery of the endpoint consists of finding its base url, in this example http://dbpedia.org/sparql.

In contrast to the interfaces discussed earlier, sparql endpoints expose an infinite number of resources, which are determined by the clients’ choice of a query. Due to this offered choice in query complexity, many such resources are expensive to compute for servers. This makes hosting a public endpoint a costly and thus relatively rare practice. Furthermore, it contributes to endpoint availability being magnitudes lower than that of regular servers (Buil-Aranda et al., 2013). So even though public endpoints are easily explored because of their extended query capabilities, their lower availability might make them less useful for subsequent tasks.

Triple Pattern Fragments

A Triple Pattern Fragments (tpf) interface (Verborgh et al., 2014; 2016) is a rest interface that extends the Linked Data principles to use forms in addition as hypermedia controls to links. Its forms consist of three fields (subject, predicate, object), such that the server offers a finite number of resources that correspond to triple-pattern queries. For instance, the result of a query for (?s, ?p, foaf:Person) on a given dataset could be available as the resource http://fragments.dbpedia.org/2014/en?object=http%3A%2F%2Fxmlns.com%2Ffoaf%2F0.1%2FPerson. Complex queries against interfaces are executed on the client side, to counter the availability issues of public endpoints. Like sparql endpoints, discovering the url of a single fragment is sufficient for access to the entire dataset; however, the server-side interface is more limited, so detailed summaries cannot be obtained directly.

tpf interfaces mitigate the dereferencing authority issue of Linked Data documents, since their hypermedia form allows clients to inquire about any uri, regardless of whether it resides in the server's dataspace. Therefore, we can discover information about a given subject in different interfaces. At the same time, the interface remains fully compatible with dereferencing. For instance, dereferencing the url http://example.org/Walt_Disney could result in an 303 redirect to the fragment resource http://example.org/dataset?subject=http%3A%2F%2Fexample.org%2FWalt_Disney, which contains all triples with this particular url as a subject.

Discovery of Linked Datasets

In the following, we describe existing work on Linked Data discovery in terms of description and strategies.

Dataset Vocabularies

Dataset vocabularies allow discovering datasets without having to access their actual interfaces. This work does not propose any new direction in this field, but employs a specific selected custom vocabulary discussed later on. Thus, we discuss the alternatives and their limitations below.

Several vocabularies exist to describe and locate datasets, and some of them are w3c recommendations. The void vocabulary (Alexander & Hausenblas, 2009) enables descriptions of an rdf dataset's characteristics. It contains concepts to describe general metadata (e.g., licenses, name, author), access metadata (e.g., sparql endpoint, data dump uri, lookup uris), and structural metadata (e.g., patterns, partitioning, vocabularies statistics). In addition, one can describe the relations with other datasets using a linkset. A void description is applicable to many fields, including query federation, data catalogs and faceted search applications. Unfortunately, applications requiring efficiency, e.g., federated query algorithms, require more fine-grained data summaries, as discussed later on. A more interface-oriented vocabulary is the sparql Service Description (Williams, 2013), which is part of the w3c sparql 1.1 recommendation. It covers a high level description of supported features by the endpoint and other characteristics useful to clients, i.e., default graph, entailment regime and so on. As its name suggests, this vocabulary only applies to interfaces, not to the other interfaces discussed above.

Next, the dcat vocabulary is used for the description of data catalogs. Data catalogs are centralized indexes or repositories that contain dataset metadata. dcat contains high-level metadata for such catalogs (e.g., title, licenses, version, etc.) and datasets (e.g., keywords, language, etc.). Querying the indexes for certain characteristics in this metadata, followed by extracting their location from the results, can discover Datasets. Popular examples are DataHub3 (which uses dcat), lov (for vocabularies)4, re3data5 (Registry of Research Data Repositories), and datacatalogs.org, a catalogue for catalogs. Despite its conception as a generic vocabulary, specific elements like dcat:byteSize make dcat mostly targeted toward data dumps, and not queryable Web interfaces. Furthermore, discovery becomes a two-step process, as it only contains detailed catalog information.

Dedicated Discovery Strategies

Vocabularies describe what can be discovered, but not how this happens. void suggests two methods for discovering descriptions: a) via backlinks from dataset documents using the void:inDataset predicate or b) by deriving to void descriptions canonically with a /.well-known/ uri. Paulheim & Hertling (2013) surveyed the state of void descriptions. The /.well-known/ mechanism has the best coverage (74% of uris), but only 14% resulted in a usable description. Using dataset catalogs results in a higher precision for retrieving an endpoint—0.48 compared to 0.19 for /.well-known/—but they are not adopted on a large scale, suffer from practical adoption issues, and depend on the implementation of redirection.

The idea behind sparql Service Description is to return the description of an endpoint on a lookup of its base (typically /sparql/). A study by Buil-Aranda et al. (2013) observed that only 35.4% of endpoints responded with 200 OK, and only 11.9% returned rdf. This low uptake prevents effective usage for discovery purposes.

In this paper, we tackle this by relying on the widely adopted fundamental components of the Web, i.e. hypermedia and http, rather than introducing a predefined complex contract that needs to be implemented by both sides.

Discovery During Querying

Other strategies can be found in link-traversal-based querying, where discovery is integrated in the query execution. Hartig (2013) identifies the live exploration method, which starts from a certain seed, i.e. from the given query or given parameter, and recursively looks up based on links. This follow-your-nose discovery is able to query priorly unknown sources and can potentially explore the entire Web of Data (assuming sufficient connectedness). However, its recursive nature causes discovery and query execution to be slow. To improve performance, live exploration is augmented with indexes that can provide seed values for the live exploration, or guide the exploration process. For instance, Ladwig & Tran (2010) added a ranked list of uris that is retrieved from an index. Other related techniques are found in focused Web crawling (Diligenti et al., 2000), where context graphs guide link traversal based on topics.

This work is based on similar principles, but deliberately separates part of the discovery from query execution. Thereby, it improves efficiency by a) supporting a range of more complex interfaces such as tpf, and b) exploiting reuse between clients by prefetching relevant interfaces in advance.

Source Selection for Federated Query Processing

Source selection determines which interfaces can potentially contribute to the result set of a particular query, and happens after the available dataset interfaces have been discovered (manually or automatically). A client evaluates a federated sparql query by decomposing it in subqueries that retrieve and join partial results. Therefore, this process is known to be an important performance factor (Saleem et al., 2015): an overestimation of candidate sources can increase execution time, an underestimation can decrease the recall of results. The exact impact on query performance (e.g., execution time), however, depends on several aspects, such as the number of sources, the amount of empty results, source response time, and parallel execution.

In this work, we study whether more detailed results of discovery can influence the efficiency of source selection, and if this reflects on the query performance. Therefore, it is important to understand its mechanics in order to design discovery for federated query processing.

Rakhmawati et al. (2013) identified three types of federation frameworks: a) federation over endpoints, b) federation using Linked Data traversal, and c) federation over custom repository apis. The first type expects all data sources to be exposed using endpoints. The second type relies on dereferenceable Linked Data documents, exposed by data providers that apply the Linked Data principles; query results are constructed by following links and looking up uris . In most federation frameworks, source selection happens triple pattern-wise, i.e. sources are selected for each triple pattern in the query, using the four source-selection strategies discussed below separately or jointly.

ASK Queries

Schwarte et al. (2011) presented FedX, which sends ASK queries to a predefined list of sources to check whether they contain triples matching a given pattern. Despite the many http requests this requires, it is one of the best performing systems in terms of execution time.

Dataset Profiling

To capture important meta-information about a dataset, compact descriptions (e.g., in void) are automatically generated. This includes statistical properties (e.g., average number of values, co-occurences), patterns (e.g., topics, clusters) (Böhm et al., 2012; Fetahu et al., 2014), and content information (e.g., present classes and properties). Some works have studied scalable methods to profile huge datasets using MapReduce (Böhm et al., 2011; Forchhammer et al., 2014), employable on heavy infrastructure such as cloud systems. Works in this area target a very broad spectrum of applications by generating as much descriptive information as possible. In contrast, our work has a very specific task, namely discovering other Linked Data interfaces in a sustainable way. Furthermore, we need to put infrastructural constraints on discovering Linked Data servers, so it can sufficiently scale on the Web. Limited loads, as well as minimal size of the description are preferred to expressiveness.

Data Indexing

Before or during query execution, the federation system gathers statistics about sources. One approach is to build a source model on a schema-level. Paret et al. (2011) and Li et al. (2014) construct a Web of Linked Classes based on classes and their relations, optionally augmented with instance statistics. Umbrich et al. (2011) published an extensive report on data summaries for live Linked Data querying. Harth et al. (2010) construct QTree indexes, which summarize instance- and schema-level elements of a dataset in a hierarchical structure combining R-trees (Guttman, 1984) and histograms. Quilitz & Leser (2008) presented the early system darq, where an index of distinct predicates is first composed to select candidate sources. The extension (Montoya et al., 2012) to the anapsid system (Acosta et al., 2011) gathers information about the distinct predicates of data sources and applies smart heuristics to estimate the source selection and improve query planning. Recently, more lightweight summaries are used in hibiscus (Saleem & Ngomo, 2014). For each distinct predicate, this approach gathers authorities for the subjects and predicates.

Cached data can also increase efficiency for future (partial) re-execution of queries. Although not using data indexing as such, Schwarte et al. (2011) extensively cache prior source selection FedX for this purpose.

Discovery of Services

The discovery of Web service functionality is different from Linked Data discovery: a Web service typically offers a handful of operations, whereas Linked Data interfaces can contain millions or even billions of triples. The process of (Semantic) Web service discovery was defined by Klusch (2014) as locating existing services that are relevant for a given request based on the description of their functional and non-functional semantics. Below are four categories of service discovery approaches, as identified by Klusch (2014).

Directory-Based Service Discovery

Directory-based discovery can be centralized and decentralized. Centralized discovery depends on registries containing descriptions. Sousuo (Klusch & Zhing, 2008) collects descriptions using meta-search in existing search engines combined with focused topic crawling, to offer free text and keyword search. Decentralized approaches use a structured peer-to-peer (p2p) network and query routing protocol. For instance, agora-p2p (Khan & Matskin, 2010) exploits a Chord ring to distribute the storage and location of services. An example from federated query processing is Atlas (Kaoudi et al., 2010), which uses Distributed Hash Table (dht) to improve source selection.

Directory-Less Service Discovery

Directory-less service discovery approaches in p2p networks include algorithms such as flooding and k-random walks, and probabilistic adaptive search. rs2d (Basters & Klusch, 2006) combines probabilistic adaptive search with owl-s, such that each peer maintains a local view of the network's semantic overlay. Directory-less approaches mostly focus on popular services and provide incomplete recall.

Hybrid Approaches

Finally, hybrid approaches work both in structured and unstructured p2p networks. meteor-s by Verma et al. (2005) uses a set of service providing and consuming peers, which are grouped on a certain topic and domain, with a central matchmaking super-peer, which maintains a global registry with the service registry concept taxonomies of all peers.

QUANTIFYING THE DISCOVERY PROCESS

In order to study our discovery process, we need to determine what parameters can and should be measured. This then allows comparing them in a qualitative and quantitative way. In particular, it allows us to identify those parameters whose optimization can or should be the focus of algorithmic design.

Preliminaries

First, we define some concepts used throughout this section and the remainder of the paper. The Web of Data can be considered a collection of interconnected datasets. Each dataset is typically modeled in rdf, which contains links between concepts and adds machine-readable semantics. We call such a dataset a Linked Dataset, defined as follows (based on Alexander & Hausenblas, 2009):

Definition 1 (Linked Dataset): A Linked Dataset D is a set of triples published, maintained or aggregated by a single provider, with $UD = \{u1, \ldots, un\}$ the set of distinct uris that identify Linked Data resources in D.

Since we can transform blank nodes into uris and back via skolemization (Cyganiak et al., 2014), we can assume in general that a Linked Dataset does not contain blank nodes. http uris are native to the Web and, thus, directly accessible through http methods. For instance, one can perform a simple http request and expect a response. A successful response is a retrieved Linked Data document, defined by Hartig (2012):

Definition 2 (Linked Data document): A Linked Data document d describes the entity identified by a uri u if there exists $(s, p, o) \in \text{data}(d)$ such that $s = u$ or $o = u$.

Note that there might be multiple Linked Data documents that describe an entity identified by u. However, according to the Linked Data principles (Berners-Lee, 2006), the uri u may also serve as a reference to a specific Linked Data document which is considered an authoritative source of data about the entity identified by u. The retrieval process of this document is called dereferencing the uri u.

Retrieving a Linked Data document is thus a possible way to obtain a fraction of a Linked Dataset over http. It is an example of a restricted Linked Data interface that only allows access to fragments about that particular uri. Other examples include data dumps, or endpoints. These interfaces differ in their expressiveness, namely all data, simple queries or complex queries.

Definition 3 (Linked Data interface): A Linked Data interface iD ∈ I provides access to a Linked dataset D on the Web through interface-specific operations, with I as the set of all interfaces.

Defining Discovery

On the Web of Documents, hyperlinks connect documents into a single global information space. The target consumers are humans, who interact with these documents through a browser. They answer queries by using search engines, following links, and interpreting text. This process always has a clear starting point, i.e. the first url typed in the browser (e.g., google.com), but no predefined end point. Humans discover relevant locations during the query solving process, while seamlessly transitioning between different Web interfaces.

Linked Data similarly connects different data sources into a single global data space, and its intended consumers are human and machine agents based on Web standards and the common data model. On the Web, the location of all possible data sources cannot be known beforehand, let alone stored in one place. Since "anyone can say anything about anything" (Berners-Lee, 1997), a specialized discovery method is necessary to find sources to answer to queries on a Web scale. We call such a method a Link Data Interface Discovery process.

Definition 4 (Linked Data Interface Discovery). Given a set of Linked Data interfaces I, Linked Data Interface Discovery LDID(I, T) is the automated process of locating the set of Linked Data interfaces $I' \subseteq I$, whose datasets contain triples that potentially contribute to the results of a task T.

Practically speaking, the output of an LDID process is a set of uris that identify Linked Data interfaces in a Web-like environment, as well as a certain list of characteristics for each of those interfaces. Note that on the Web, the complete set of interfaces I is not known before, during, nor after the discovery process. However, results gathered in closed large-scale simulated environments, where I is known, can be generalized to a subweb, i.e. Web of interfaces used by a certain application, or provide indications for the entire Web.

In the context of federated query processing, the task T is the execution of a set of queries Q. Ideally, a query application also requires only one starting point, even though several servers might be required to find the desired answer to a query. A federated query algorithm then applies source selection to the output of a discovery process in order to select the relevant interfaces for a particular query. In contrast to discovery, source selection is typically repeated for each query.

Measuring Discovery

To conclude this section, we define several parameters to measure to what extent the discovery method has succeeded. We discuss metrics for the functional requirements, the discovery outcome, and the non-functional requirements, the discovery process.

Measuring Outcome

A discovery process ideally strives for completeness: finding all interfaces that can contribute to the results of a certain task. Thus, there is a direct dependency of the task result completeness on the completeness of a discovery process. For instance, in federated query processing, the more interfaces we are able to discover, the higher our chances of finding all possible answers, i.e. reaching completeness. Interfaces can be called relevant to the executed queries, if they provide access to data that contributed to the result.

Definition 5 (Relevant Linked Data Interface): A Linked Data Interface i ∈ I, publishing a dataset D, is relevant to the query Q if a subset of D is used to compose a result of Q.

The completeness and accuracy of the discovery outcome can be measured with the default recall and precision metrics in closed-world experiments. However, in an open world such as the Web, the total number of relevant interfaces is unknown, since a) data is distributed over a huge scale-free network, where counting datasets is infeasible; b) this network is subject to constant change. Therefore, closed-world experiments performed on different scales should provide an indication on Web-scale performance.

Measuring the Process

The performance marks of an LDID process are evaluated according to the objective of a task and the process characteristics, as defined earlier. This creates a different optimal trade-off mix of parameters for each process. As a result, they are incomparable with a single global performance metric. Instead, we discuss the metric space of non-functional requirements of LDID in three quantifiable aspects.

- Bandwidth Usage: Discovery processes need to communicate with Web resources over http. Depending on their greediness, they can fundamentally differ in the amount of used bandwidth. This is measured in number of requests sent to complete the discovery.
- Discovery Execution Time: Depending on the task at hand, a discovery process can be fast or slow. For example, in live exploration querying, the query execution time is directly dependent on the amount of time it took to gather the set relevant sources. In contrast, data analysis is often performed offline, after the data is discovered and gathered. Discovery execution time is measured in the amount of time until the completion of the discovery process.
- Server Response Overhead: When a task is executed, a client relies on servers to retrieve data or to provide a service. Depending on the discovery process, the impact on the default server behavior will be different. For instance, if the client handles the complete discovery, the server overhead will be zero. However, if the server is part of the process, e.g., Web crawling, a certain percentage can be added to its response time. This is measured in the difference in time between server responses with and without discovery.

HYPERMEDIA-BASED DISCOVERY APPROACH

In this section, we introduce a hypermedia-based discovery approach to benefit federated query clients during execution. Servers discover relevant interfaces using two methods:

1. Active Discovery: servers dereference links from the interface data to discover relevant interfaces.
2. Reactive Discovery: servers react to being accessed by clients, i.e. other discovering servers or querying clients.

This combination allows the discovery of outgoing links (active), as well as incoming links (reactive). Both methods rely on three foundations: data summaries, publishing data through a Triple Pattern Fragments server and applying the Linked Data principles. They make two assumptions: a dataset has one main domain authority (e.g., http://dbpedia.org/ for DBpedia), and the owner of that domain handles the dereferencing of its uris (e.g., the DBpedia pages). Both of which can be considered a result of ownership: data providers are unlikely to adopt external due to control concerns.

In the following, we first describe dataset summarization, and then discuss both active and reactive discovery methods in more detail.

Creating Data Summaries

The discovery process is performed by an application hosting a Linked Data interface, which publishes the dataset d. Our approach uses a dataset's uris as a seed. Thereby, the application will first create a dataset summary from d in order to access these required uris efficiently. The motivation for this is twofold: as Linked Datasets are potentially huge, efficiently scanning an rdf dataset for uris is fundamental, and metadata about datasets is exchanged between servers.

Such summaries are a lightweight form of dataset profiling and represent crucial dataset information in a compact manner. As discussed in related work, they are common in federated query processors. Given the constraints above, the desired characteristics are therefore:

- A high summarization rate for decreasing scan time and exchange time over http;
- A low complexity for little server load and fast computation;
- Quick and accurate matching of triple patterns for the hypermedia construction discussed later on.

Based on these requirements, we reuse the approach from the query federation system HiBISCuS. Saleem & Ngomo (2014) propose lightweight summaries based on grouping authorities of subjects and objects per distinct predicate. In contrast to their work, we will not use the summaries for query optimization on the client, but instead to let the server a) iterate over foreign domains in the datasets, and b) generate specific hypermedia to other interfaces to inform the client.

Definition 6 (Authority Function): An authority function extracts the authority part, standardized by Dürst & Suignard (2005), of an rdf term, and is defined as:

$$authority\left(u\right) = \begin{cases} protocol\left(u\right) + hostname\left(u\right),\ if\, isUri\left(u\right) \\ nil,\ otherwise \end{cases}$$

The HiBISCuS summaries show a very high summarization ratio (≈99% for all FedBench datasets6). Furthermore, they contain information about all terms of a triple, which makes them candidates for efficient and accurate triple pattern matching.

A data summary ds is defined as a set of capabilities {cap1, …, capn}. For a source d, a capability capx is a triple (p, SA, OA) for a specific predicate p. Herein, SA is the set of distinct subject authorities of p in d, and OA the set of distinct object authorities of p in d. When p = rdf:type, an exception is made where OA is the set of distinct object (instead of their authorities). We can attach a data summary to the metadata of a fragment. An example for the Jamendo dataset (from FedBench) is given in Listing 1.

Active Discovery with Dereferencing

This subsection details an algorithm in which a server discovers others through uris from its own dataset.

Phase 1: Identify External Servers

The discovering server actively tries to identify other servers based on data summaries, the Linked Data principles, and hypermedia. This process is data-driven; it exploits the links to external datasets, which are possibly hosted by other servers. Therefore, we start by identifying a set of foreign uris P = {r1, …, rn}⊆UD in the dataset. They are uris that differ in hostname with the interface, and whose lookup is handled by other servers. We rely on the data summary generation process to populate UD with one sample per authority. For instance, the Jamendo dataset (Table 1) contains 776,611 distinct subjects and objects, but is covered by 655 samples because there are only 655 unique authorities.

Listing 1. The fragment http://dbtune.org/fragments# dataset extended with dataset demo summary metadata

```
@base <http://dbtune.org/fragments>.
@prefix void: <http://rdfs.org/ns/void#>.
@prefix hydra: <http://www.w3.org/ns/hydra/core#>.
@prefix ds: <http://semweb.mmlab.be/ns/summaries#>.
@prefix mo: <http://purl.org/ontology/mo/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<> hydra:member <#dataset>.
<#dataset> a void:DataSet, hydra:Collection;
ds:capability [
ds:predicate foaf:based_near;
ds:sbjAuthority <http://dbtune.org/>;
ds:objAuthority <http://sws.geonames.org/>
];
ds:capability [
ds:predicate rdf:type;
ds:sbjAuthority <http://dbtune.org/>;
ds:objAuthority foaf:Document, mo:MusicArtist
];
ds:capability [
ds:predicate foaf:name;
ds:sbjAuthority <http://dbtune.org/>;
].
```

Table 1. The FedBench datasets

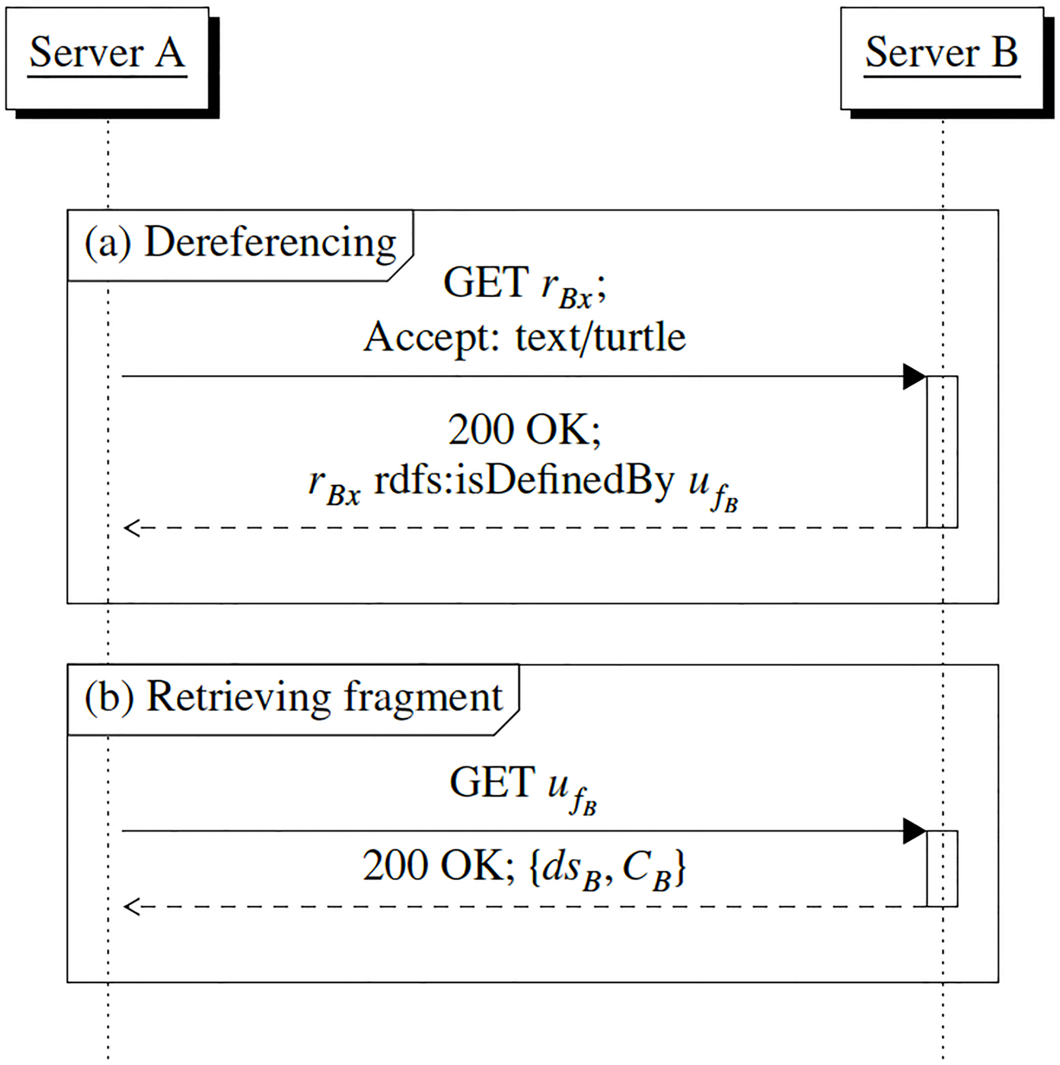
dataset	# triples	# distinct subjects	# distinct predicates	# distinct objects
DBPedia subset	42,849,609	9,495,865	1,063	13,620,028
NY Times	335,206	21,666	36	191,538
LinkedMDB	6,147,996	694,400	222	2,052,959
Jamendo	1,049,647	335,925	26	440,686
Geonames	107,950,085	7,479,714	26	35,799,392
SW Dog Food	103,465	11,974	118	37,547
KEGG	1,090,830	34,260	21	939,258
Drugbank	517,023	19,693	119	276,142
ChEBI	4,772,706	50,477	28	772,138
SP2B-10M	10,000,457	1,730,250	77	4,690,662

Phase 2: Dereference Entities on External Servers

The discovering server acts as a client and discovers other servers by dereferencing each rx ∈ R. In these responses, it looks for triples (rx, rdfs:isDefinedBy,ufx), see Figure 1 (a). The uri ufx possibly identifies a tpf fx of the dataset that contains rx. If no such triple is found, rx does not support this discovery approach, and the discovery for this ends here.

Continuing our previous example, the Jamendo server on domain dbtune.org could dereference the foreign http://sws.geonames.org/660013/. This lookup is subsequently answered by the server

Figure 1. Active discovery discovers Linked Data interfaces by dereferencing foreign URIs in the dataset and retrieving the linked Triple Pattern Fragment



on sws.geonames.org, which hosts the Geonames dataset as Linked Data documents. The response contains the triple (http://sws.geonames.org/660013/, rdfs:isDefinedBy, http://sws.geonames.org/fragments/), where the object identifies a tpf of the Geonames dataset.

Phase 3: Identify Controls and Summaries

The server requests fx and looks for hypermedia controls Cx and a data summary dsx in the response, see Figure 1 (b). The hypermedia controls reveal whether the resource is a tpf, and if so, allow requesting other tpfs of the dataset. The data summary informs the discoverer about the data that can be found in the interface, which is used later on to construct hypermedia to this server. For the fragment http://sws.geonames.org/fragments, an extract of the request cycle is shown in Listing 2. The data summary starts at line 12 and the controls at line 18.

Listing 2. A fragment of the discovered interface returns hypermedia controls and a data summary.

```
GET http://sws.geonames.org/fragments HTTP/1.1
Accept: text/turtle
Referer: http://dbtune.org/fragments
--------------------------------------------------
HTTP/1.1 200 OK

<http://sws.geonames.org/fragments>
hydra:member
<http://sws.geonames.org/fragments#dataset>.
<http://sws.geonames.org/fragments#dataset>
a void:Dataset, hydra:Collection;
ds:capability [
ds:predicate gn:parentFeature;
ds:sbjAuthority <http://sws.geonames.org/>;
ds:objAuthority <http://sws.geonames.org/>
];
...
hydra:search _:triplePattern.
_:triplePattern hydra:template
“/fragments{?subject,predicate,object}”;
hydra:mapping _:subject, _:predicate, _:object.
_:subject hydra:variable “subject”;
hydra:property rdf:subject.
_:predicate hydra:variable “predicate”;
hydra:property rdf:predicate.
_:object hydra:variable “object”;
hydra:property rdf:object.
```

Phase 4: Index Storage

The tuple {dsx, Cx} is stored in an index Ids with ufx as key. To prevent the same fragments to be requested multiple times, the presence of fx is looked up first in the set of keys. Note that in some cases, e.g., a cache expiry, it is desirable to re-request a fragment to update its summary and controls.

Active discovery ends when all sample uris have been dereferenced, so Ids contains of list of discovered summaries.

Reactive Discovery with the Referer Header

Since active discovery is limited by the foreign uris present in the dataset, its reach is insufficient for two reasons. First, queries that require knowledge about backlinks cannot be answered, leading to low result recall. Since hyperlinks are unidirectional, servers can only discover interfaces that host their outgoing links. Second, when a dataset changes, its summary in the index of other servers will be outdated. Therefore, a server needs to notify those who have discovered it about this change.

Reactive discovery tackles both issues by using the Referer header from incoming requests, inspired by the work of Mühleisen & Jentzsch (2011), to automatically create new links in the cloud. Referer is standardized in RFC2616,7 which states:

The Referer request-header field allows the client to specify, for the server's benefit, the address (uri) of the resource from which the Request-URI was obtained.

In this approach, two actors are equipped with this header:

- Querying clients a query execution application supplies the last accessed fragment, enabling discovery of servers with no explicit link in the datasets.
- Discovering servers a server doing active discovery supplies a random fragment (in practice the index fragment) on which it can be discovered in turn.

The process happens as follows, illustrated in Figure 2.

Phase 1: Header Inspection

When a server receives a request by client C, it checks whether a Referer header is present. If it is, it should be a uri ufx. Possibly, ufx identifies a Triple Pattern Fragment fx.

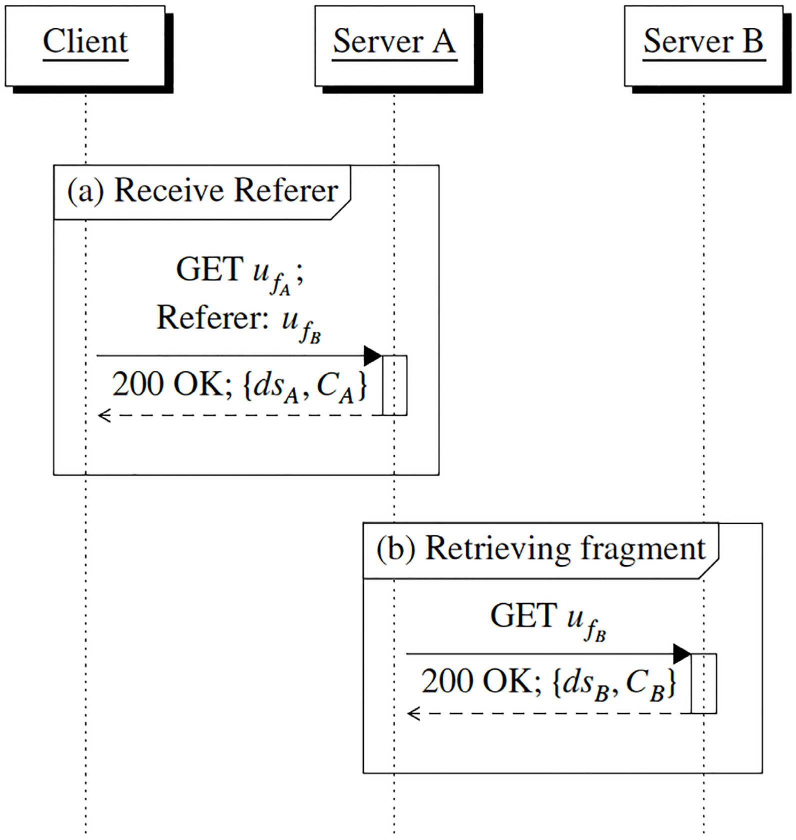
Phase 2: Dereference the Referrer

The server verifies that ufx is a foreign uri with an authority it has not (recently) visited before. If so, the server dereferences ufx to obtain the resource fx.

Phase 3: Identify Controls and Summaries

The server inspects fx in order to find the summary ds and controls Cx, analogous as in Phase 3 of the active discovery process. For example, given the request in Listing 2, the Geonames server can react by requesting the Referer http://dbtune.org/fragments given in line 3. As a result, it retrieves the summary and controls for the Jamendo server.

Figure 2. Reactive discovery discovers interfaces by retrieving fragments from the Referer header



Phase 4: Index Storage

If both are correctly extracted from the response, the tuple {dsx, Cx} is stored in an index Ids as well, with ufx as key.

Since http caches can intercept the Referer header, they could prevent reactive discovery—and caching is a crucial scalability factor for Triple Pattern Fragments (Verborgh et al., 2014). However, the cache could maintain a list of referers (for instance, in server logs), which are then passed to the server at a later stage.

FEDERATED QUERY PROCESSING

Through Hypermedia

In this section, we describe how clients consume the result of a discovery process for the execution of queries. Clients rely on hateoas (hypermedia as engine of application state), where they react autonomously to links included in http responses. First, we discuss how the hypermedia is constructed on the server. Second, we describe how the client consumes the links to retrieve more relevant fragments on the fly. As a running example, we discuss the evaluation of the query in Query 3 over the Geonames and Jamendo data sources mentioned in the previous section.

Building the Server-side Hypermedia Interface

When a fragment is requested, the server informs the client about the interfaces it has discovered in the response. Instead of sending all known interfaces, it pre-selects the ones most relevant to the requested fragment in order to aid with source selection. This reduces the chance of source overestimation on the client.

When handling a request, the server has the following data at its disposal:

1. The requested triple pattern tp = (s, p, o), extracted from the request;
2. An indexed set of data summaries Ids = {uf1 ⇒ {ds1, C1},…,ufn ⇒ {dsn, Cn}}.

Based on their summaries, the server looks for servers that can possibly answer the requested triple pattern as well. Therefore, a server iterates over all summaries in Ids. For each summary, it checks for a match using Figure 3.

First, we select the summaries to iterate over [line 1]. If the predicate is a variable, we select all summaries in the index. If the predicate is a uri, we look up its entry. Next, we iterate over the selected set of capabilities. If no entry is found, the summary does not match [line 3]. For each summary, if the subject or object is a uri, we extract its authority [line 4,5]. If it is not a uri, it's either a variable, blank node or literal, which always match. Finally, a server with a data summary only has a chance of matching the requested triple pattern if

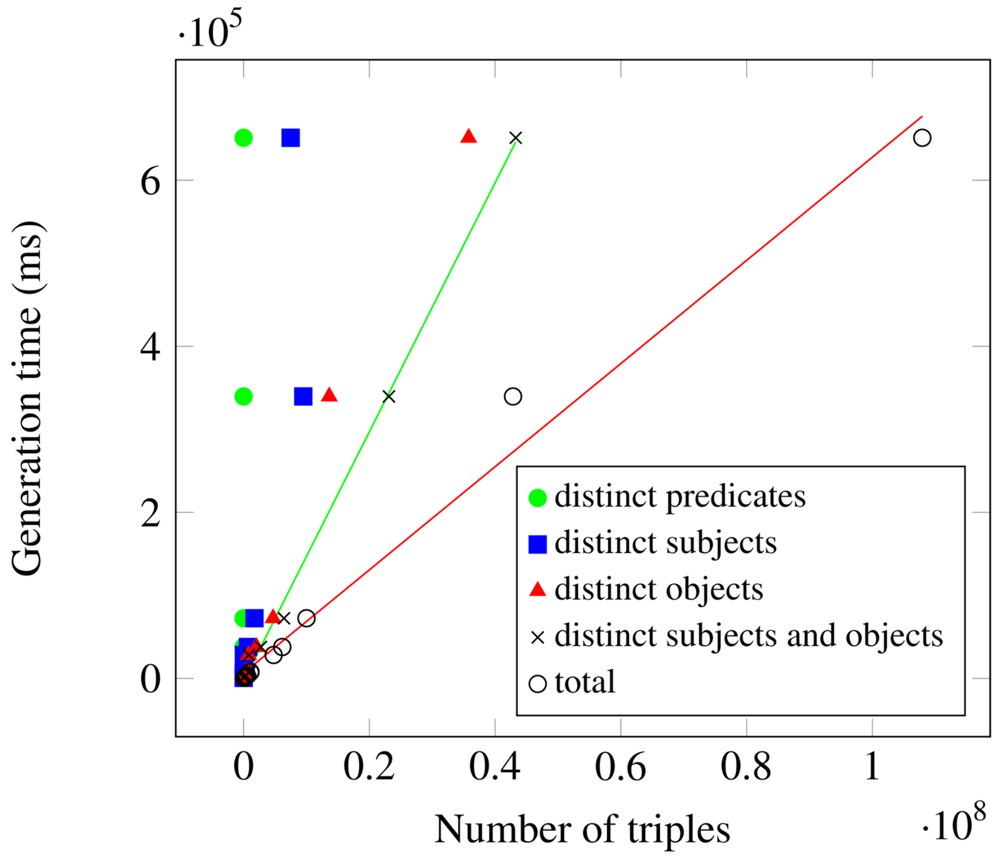
1. The predicate is rdf:type, which is an exception in the data summary, and the object in the patterns is an exact match of a in the object authority [line 9]; or
2. The subject is a variable or its subject authority is equal to the authority of the pattern's subject, and the same holds for the object [line 11].

For instance, when the Jamendo server receives a request for the tp on line 5 of Listing 3, it returns zero triples, as the dataset does not contain gn:parentFeature. Before responding, it scans the index for a match, which contains the summary for http://sws.geonames.org/fragments. As shown in line 13 of Listing 2, the Geonames summary does contain gn:parentFeature and is a match, since the subject and object of tp are variables. Similarly, when the tp on line 7 of Listing 3 is requested from

Listing 3. An example query combining the Geonames and Jamendo datasets

```
1. PREFIX foaf:<http://xmlns.com/foaf/0.1/>
2. PREFIX gn:<http://www.geonames.org/ontology#>
3.
4. SELECT ?name ?location ?news WHERE {
5. ?artist foaf:name ?name .
6. ?artist foaf:based_near ?location .
7. ?location gn:parentFeature ?germany .
8. ?germany gn:name "Federal Republic of Germany"
9. }
```

Figure 3. Determine if a given triple pattern matches a data summary, and possibly its dataset



the server, it returns triples, because foaf:name is present in the dataset. However, the Geonames summary does not match, since foaf:name is not present.

For each fragment ufx that matches the requested triple pattern (m(dsx, tp)=true), we create a direct link uri uf'x. This link points to the fragment for the same triple pattern on a remote server. The resulting set of links H = {uf′m, …, uf′n} are the hypermedia controls of the response, i.e. we add zero or more rdfs:seeAlso links8 to inform the client about the other servers. For instance, if the Geonames summary matches the triple pattern in Listing 3, line 7, the hypermedia triple in Listing 4

Listing 4: Although the Jamendo server returned no triples for the pattern tp = (?x,gn:parentFeature,?y), it contains a hypermedia control to the corresponding fragment in the Geonames dataset.

```
1. <http://dbtune.org/fragments?object=http%3A%2F%2Fwww.geonames.
org%2Fontology%23parentFeature> rdfs:seeAlso <http://sws.geonames.
org/fragments?object=http%3A%2F%2Fwww.geonames.org%2Fontology%23p
arentFeature>.
Client-Side Query Execution
```

will be added to the fragment's response. Then, the client can follow this rdfs:seeAlso link to collect (more) matching triples.

sparql queries can be federated over multiple Triple Pattern Fragments servers by extending the existing querying algorithm (Verborgh et al., 2016). Query federation systems are client applications that decompose queries, send partial queries to endpoints, and join the results. As the interface aims at pushing intelligence to the client and way from the server, the tasks above are far more native to the client-server set-up. As a result, aggregating multiple requests already allows a naive, but effective query federation system.

The algorithm by Verborgh et al. (2014) constructs a dynamic iterator pipeline to solve queries over one server. A Basic Graph Pattern B = {tp1, ..., tpn} is evaluated by a bgp iterator, which recursively decomposes it into smaller iterators. For each triple pattern tp in B, the first page is fetched from the corresponding tpf. Its response tells us how many matches (cnttp1) the dataset has for each triple pattern. The pattern is then decomposed by evaluating it using a) a triple pattern iterator or the triple pattern with the smallest number of matches, and b) a new bgp iterator for the remainder of the pattern. This results in a dynamic pipeline for each of the mappings of its predecessor. Each pipeline is optimized for a specific mapping, reducing the number of requests for each path.

In this work, the client also starts with one fragment uri, but accesses more during execution. We extended the existing algorithm with an extra abstraction when fetching a tpf. Multiple fragments from different servers are aggregated and exposed as a single fragment with the following two steps:

1. Following links included in the response of the first fragment;
2. Merging the results and metadata according to an aggregation function.

The first step calls a recursive function based on rdfs:seeAlso links present in the fragments' responses. The second step aggregates both the data and the metadata from all fragments. The data are appended to a single stream as the responses come in. When all fragments have arrived and their data stream has ended, the aggregated stream ends. The count metadata are combined into a single value using an aggregation function ϑ, which is a cost function that can be optimized to the type of interface. For example, ϑ can be a weighted sum, taking into account practical differences between servers like response time, page size and so on. In our implementation, as we discuss later on, we chose a sum $\vartheta = \sum_{i=1}^{N} cnt_i$ for simplicity.

For example, the query in Listing 3 is executed against the Jamendo server from earlier. For each triple pattern, the corresponding fragment is retrieved from the server which results in:

```
?artist foaf:name ?name. # 3,505 matches
?artist foaf:based_near ?location. # 3,244 matches ?location
gn:parentFeature ?germany. # 0 matches
?germany gn:name "Federal Republic of Germany". # 0 matches
```

Typically, the algorithm would stop the execution here, as the last two patterns have no matching triples, leading to an empty resultset. However, the response for both patterns does include a link to a corresponding fragment on the Geonames server. This retrieves each link, appends the result to the original fragment resultset and sums the count metadata with ϑ. Now, both fragments do return matches, which updates the previous results:

```
?artist foaf:name ?name. # 3,505 matches
?artist foaf:based_near ?location. # 3,244 matches
?location gn:parentFeature ?germany. # 0 + 7,479,713
# = 7,479,713 matches
 ?germany gn:name "Federal Republic of Germany".# 0 + 1 = 1 match
```

After aggregation, the last triple pattern has the lowest number of matches, namely 1. Thus, the algorithm continues with the single matching triple for this pattern and binds the value http://sws.geonames.org/2921044/ to the variable ?germany. Finally, the process is repeated for the remaining three triple patterns.

EVALUATION

The experimental results presented in this section aim to provide insight in the performance of the hypermedia-based discovery approach and its impact on federated query execution. We describe the implementation, test setup and performed tests as follows.

Experimental Setup

We implemented our approach in Node.js as a plugin for the existing Triple Pattern Fragments server.9 This plugin is called the Explorer and runs in a separate single-threaded process, asynchronous to the server application, indexing data summaries as they arrive. The server communicates with the Explorer using three methods for a) starting the active discovery; b) passing a Referer header if detected in a request; and c) passing a triple pattern from the request to retrieve a set of rdfs:seeAlso links, i.e. the hypermedia controls.

To test the query execution, we selected the Fedbench (Schmidt et al., 2011) benchmark and its corresponding collection of datasets. FedBench relies on real world authoritative datasets that are prominent in the Web of Data and provides a federated query mix.10 We selected the Cross Domain (CD), Life Science (LS), and Linked Data (LD) queries from the list of pre-defined queries, as they make use of multiple servers. Queries LS7 and CD6 are not supported by our implementation, and are therefore excluded from the results. An overview of the used datasets is given in Table 1.

In total, we allocated 9 virtual machines as servers in an Amazon EC2 Virtual Private Cloud, one for each dataset. Each virtual machine has the equivalent of a dual-core Intel Xeon E5-2670 v2 CPU, 7.5 GB of RAM and 32 GB of SSD storage. The datasets KEGG and ChEBI were merged to avoid dereferencing conflicts, as they share the same authoritative hostname. On each server, we edited the /etc/hosts file to simulate the different domain names.

In addition, a server was provided with an active NGINX11 http cache and a Triple Pattern Fragments server including the Explorer module, both handling the dereferencing and the Triple Pattern Fragments interface. The data was loaded into the server as a compressed file. An extra virtual machine was created to run the FedBench client and gather server logs. A plugin for a Triple Pattern Fragments client supporting both hypermedia-based and naive federation was added to the FedBench client.

Experiment 1: Feasibility of the Discovery Approach

In the following, we assess Question 1 and test to what extent a discovery approach is feasible by using hypermedia. We list the objectives and discuss the experimental results.

Objectives

This experiment tests the five hypotheses stated below. First, to assess Question 1.1, we inspect the data summary construction, which has the biggest impact on the server. As this process solely relies on string operations, we foresee the following:

Hypothesis 1: The data summaries can be computed in a reasonable time, which scales linearly with the size and anatomy of the dataset.
Hypothesis 2: The data summaries are highly compact compared to the original dataset size.

Next, in terms of Question 1.3, we expect the number of required requests to have a major impact on the discovery execution time. Fortunately, this number is directly dependent on the amount of links (indicated by a triple's object) to other datasets, which is only a small fraction of all uris present. Therefore, the discovery execution time should scale well with the size of the dataset, and bandwidth usage should be limited:

Hypothesis 3: The discovery execution time scales linearly with the bandwidth usage.
Hypothesis 4: The discovery execution time scales linearly with the size of the dataset.
Hypothesis 5: The amount of requests is minimal compared to the total amount of triples in the dataset.

Finally, we assess the completeness of our discovery approach with Question 1.2. The active discovery is data-driven, which should cover all outgoing links, while reactive discovery should cover all incoming links. Hence, all directly relevant interfaces should be discovered:

Hypothesis 6: The discovery approach is able to reach full completeness in a closed-world environment.

Summary Complexity

We measured the summary generation time for each dataset. Each summary was constructed from its corresponding file. The results shown in Table 2 confirm that the summaries are lightweight in terms of complexity and size.

The largest dataset, i.e. Geonames with 108k triples, was summarized in only 10.85 minutes, while the smallest dataset, i.e., Semantic Web Dog Food, was summarized in less than a second. These low computation times can partially be explained by the highly efficient format, which offers a highly compressed browsable read-only index. However, the alternative interface of servers does allow these highly specialized data sources. Furthermore, no multi-threading was used during generation, which can still be fully exploited and compensate slower data sources, e.g., writeable indexes like x-rdf-3x (Neumann & Weikum, 2010) or sparql endpoints. Another factor is likely the sole use of authorities, which can be derived with a highly efficient regular expression operation.

Next, we look at the correlation between the generation time and the number of distinct subject, predicates, and objects. No clear impact can be observed of the number of distinct predicates on the generation time (Pearson's $r = 0.315$ and Spearman's $\rho = 0.043$). This is likely because the distinct subjects and objects highly outnumber the distinct predicates. For instance, Geonames and DBpedia have the highest generation times, while DBpedia has 30 times more distinct predicates than Geonames. As shown in Figure 4, we do observe a linear correlation with coefficient of determination $R2 = 0.998$ and $p = 0.000$ ($\alpha = 0.05$) between the total number of distinct subjects and distinct objects ($r = 0.999$ and $\rho = 0.988$). The dataset size, i.e. the total number of triples, shows a slightly weaker correlation with $R2 = 0.987$ and $p < 0.001$ ($r = 0.993$ and $\rho = 1.000; \alpha = 0.05$), but still indicates a clear linear relationship between size and execution time.

Table 2. The compression rate of data summaries is very high (99%) with acceptable creation time (11 min)

dataset	generation time (ms)	json size (kB)	ratio (%)	original hdt size (kB)
DBPedia subset	339,598	85.68	99.9853	584,679
NY Times	2,237	25.12	99.7786	11,344
LinkedMDB	37,840	26.10	99.9502	52,413
Jamendo	7,390	53.69	99.7059	18,258
Geonames	651,102	26.01	99.9975	1,035,674
SW Dog Food	667	83.17	96.8084	2,606
KEGG	7,755	1.62	99.9863	11,850
Drugbank	3,650	13.27	99.9217	16,942
ChEBI	28,233	1.87	99.9920	23,502
SP2B-10M	72,482	6.38	99.9980	320,662

The correlation findings above in combination with the overall low execution times, allow us to accept Hypothesis 1.

Summary Size

In terms of the summary size, the fourth column in Table 2 shows a summarization ratio above 99% compared to the already compact binary file. We consider a summary significantly compact above 85% ($\alpha = 0.05$). Thereby, Hypothesis 2 can be validated with single-value T-value of 46.655 and $p < 0.001$. Because of the authorities, summarizing the dataset creates a lot of redundancy, which allows us to greatly compress the result. The created redundancy decreases according to the number of distinct authorities present in the dataset. This potentially leads to a low summarization ratio for highly variable datasets, like sameAs.org. However, as previously pointed out, this is rare because of uri ownership, where data publishers tend to base their uris around a single domain name.

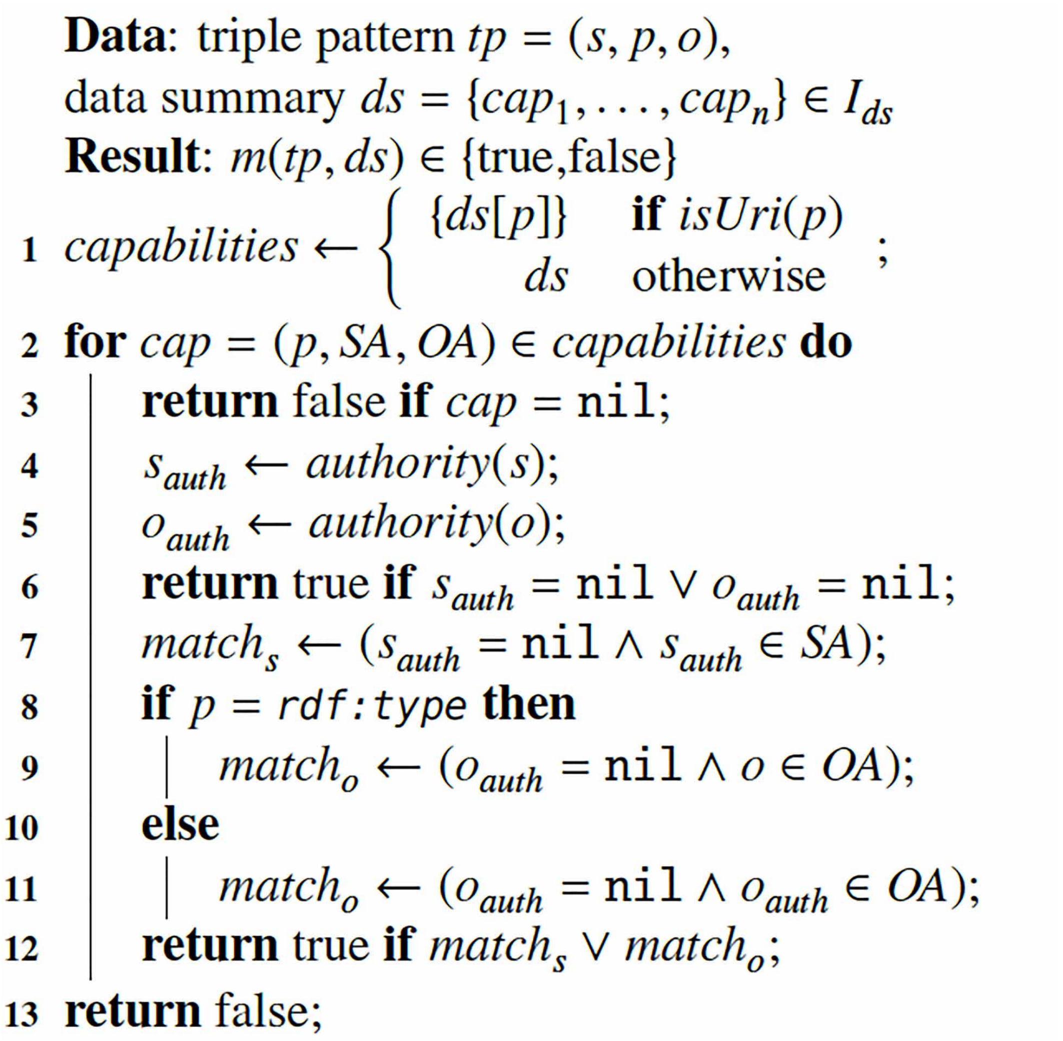
In term of absolute size, the summaries remain below a few hundred kilobytes. The largest summary DBpedia, responsible for roughly 40 million triples, results in only 85 kB. Although this size increases slightly when loaded into memory, we can conclude that they are sufficiently compact. The server can load many summaries in memory without introducing large overhead. Furthermore, the current implementation only applies a simple dictionary, thus, a more extensive compression can still massively reduce space.

Discovery Execution Time and Bandwidth Usage

To evaluate the performance of our hypermedia-based discovery approach, we deployed 8 servers, one for each dataset. We excluded the SP2B dataset since it uses local and is not linked to any dataset. Each server was provisioned with a pre-computed data summary and the server application. The active discovery process was executed on all machines simultaneously, also triggering the reactive discovery.

Table 3 shows the results for the active discovery phase. All servers were able to discover their direct neighbours, of which three (DBpedia, LinkedMDB, Drugbank) in less than 10s, four (NY Times, Jamendo, Geonames, SW Dog Food) in less than 7min and one outlier (KEGG–ChEBI) in only 5ms. This outlier is due to the absence of external uris in the KEGG–ChEBI dataset, which requires no requests. The difference between the two other clusters also correlates with their differences in request count. Thus, higher bandwidth usage reflects on a higher execution time. The highest execution time was measured for the SW Dogfood dataset. Although it has the lowest number of triples, the maximum of 771 requests was sent. Its high number of links to personal author websites, resulting in

Figure 4. The data summary generation time increases linearly with the number of distinct subjects and objects



```
Data: triple pattern $tp = (s, p, o)$,
data summary $ds = \{cap_1, \ldots, cap_n\} \in I_{ds}$
Result: $m(tp, ds) \in \{\text{true,false}\}$
$capabilities \leftarrow \begin{cases} \{ds[p]\} & \textbf{if } isUri(p) \\ ds & \text{otherwise} \end{cases}$;
for $cap = (p, SA, OA) \in capabilities$ do
    return false if $cap = \texttt{nil}$;
    $s_{auth} \leftarrow authority(s)$;
    $o_{auth} \leftarrow authority(o)$;
    return true if $s_{auth} = \texttt{nil} \vee o_{auth} = \texttt{nil}$;
    $match_s \leftarrow (s_{auth} = \texttt{nil} \wedge s_{auth} \in SA)$;
    if $p = \mathtt{rdf{:}type}$ then
        $match_o \leftarrow (o_{auth} = \texttt{nil} \wedge o \in OA)$;
    else
        $match_o \leftarrow (o_{auth} = \texttt{nil} \wedge o_{auth} \in OA)$;
    return true if $match_s \vee match_o$;
return false;
```

a high number of unique samples, can explain this. In general, the low number of requests is a result of using a data summary as seed. Despite the large number of triples in a dataset, e.g., DBpedia, the request count can be very low (e.g., 16 requests), due to low authority variance in its uris.

Table 4 shows the results of the reactive discovery phase. Each server has discovered the interfaces that were connected through a backlink. Reactive discovery is cheap in terms of bandwidth usage, as it only needs one requests per discovery. As a result, its execution time is mostly determined by time taken to reconstruct the summary from the response, as shown in the last two columns.

To conclude, we can validate Hypothesis 3, as execution time is directly related to the number of requests. We observe a linear correlation ($\alpha = 0.05$) with R2 = 0.889 and p = 0.002 (r = 0.943 and $\rho = 0.952$). Also, we can validate Hypothesis 5, since the number of requests sent, is only a fraction ($\mu = 0.1\%$ with $\sigma = 0.001$) of the total number of triples. However, we falsify Hypothesis 4, because no relation is shown between a dataset size and the discovery execution time. No correlation can be observed since R2 = 0.082 (r = −0.287 and $\rho = 0.698$) and p = 0.245.

Table 3. Active discovery is able to find interfaces for all links. No links are found for KEGG–Chebi, since it does not contain any outgoing links. (#L=number of links, #F=number of fragments, #S=number of fragments stored in index, #Req=number of sent requests, SCT=Summary construction time, ET=Execution time)

Dataset	Links Found	#L	#F	#S	#Req	SCT (ms)	ET (ms)
DBpedia	OWL, NY Times	2	1	1	16	41	1,073
NY Times	DBpedia, Geonames	2	2	1	240	13	30,666
LinkedMDB	DBpedia, Geonames	2	2	2	15	118	6,704
Jamendo	Geonames	1	1	1	655	31	218,549
Geonames	DBpedia	1	1	1	261	129	24,853
SW Dog Food	OWL, DBpedia, Geonames, Drugbank	4	3	3	771	211	382,787
Drugbank	RDFS, DBpedia, KEGG–ChEBI, OWL	4	2	2	22	126	7,609
KEGG–ChEBI	–	0	0	0	0	–	5

Completeness of the Discovery Outcome

In terms of completeness, the combination of active and reactive discovery result in a recall of 1.0, as all interfaces relevant to the discovery task are found ($\mu = 1.0$). Here, a relevant interface provides access to a dataset to which a link is included in the discoverer's dataset. Therefore, these results validate Hypothesis 6.

If the number of stored summaries deviates in Table 3, it was already discovered by reactive discovery, or visa versa. During active discovery, both DBpedia and SW Dog Food found links to OWL or RDFS, but were detected as not being fragments. Completeness in context of a query task is discussed in the next section.

Experiment 2: Impact on Federation Query Execution

In this second experiment, we measure the impact of our discovery approach on client-side federated query execution to assess Question 2. We ran FedBench on different set-ups, combining a client variant and a specific index. Two variants of clients were implemented: a) a naive client that sends every request to every server, without consuming hypermedia, and b) a hypermedia-based client which implements the query execution algorithm described in this paper. The naive client will act as a baseline to measure the impact of the hypermedia-based client. It mimics existing federation systems that solely rely on triple pattern to do both source selection and query execution. We do not use other state-of-the-art federation systems as baseline, since the comparison would be unfair. Such systems are highly tailored to the efficient, but computationally heavy sparql interface, which enables very low execution times. Our efforts aim at a sustainable solution for both client and server, thus using the slower, but lightweight interface. We aim at investigating the role of hypermedia in source-selection, rather than optimizing execution speed.

In addition, we prepared two distinct index provisions: a) populated with all summaries of the other datasets (full index), and b) populated with the summaries from its discovery outcome (discovery index). The full index setup provides insight on the general performance of hypermedia-based source selection. It creates the ideal scenario where each interface is able to provide a link to any other interface. This isolates the query execution from any limitations of the discovery approach, i.e., incomplete indexes. The discovery index setup provides insight in the relation between discovery outcome and the query algorithm. Here, each interface has a different, more selective index, which enables studying the impact on query completeness and execution time.

Table 4. Reactive appends the backlinked interfaces to the active discovery results. (#F=number of fragments, #S=number of fragments stored in index, #Req=number of sent requests, ASCT=Avg. Summary construction time, AET=Avg. Execution time)

Dataset	Links Found	#F	#S	#Req	ASCT (ms)	AET (ms)
DBPedia	Geonames, Drugbank, SW Dog Food, LinkedMDB, NY Times	4	4	1	56	73
NY Times	DBpedia	1	1	1	123	142
LinkedMDB	–	0	0	–	–	–
Jamendo	–	0	0	–	–	–
Geonames	LinkedMDB, NY Times, SW Dog Food, Jamendo	4	4	1	49	53
SW Dog Food	–	0	0	–	–	–
Drugbank	–	1	1	1	36	76
KEGG -ChEBI	Drugbank	1	1	–	30	48

First, we list the objectives of this experiment. Next, we discuss the results in terms of query result completeness, server overhead and query execution time.

Objectives

This second experiment tests the three following hypotheses. First, we address the recall of query results from Question 2.1. Since we expect the recall of the discovery outcome to be high, the hypermedia controls should be sufficient to produce high result recall as well (> 0.75). Therefore, we state the following:

Hypothesis 7: The query results sustain the high recall reached by the naive baseline.

Second, in correspondence to 2.2, we expect a limited impact of the hypermedia control generation process on the server. The summaries are very compact, directly available in the main memory, and contain all information for efficiently producing the links, given the algorithm described in the previous section.

Hypothesis 8: The addition of generating hypermedia introduces a minimal computational overhead on the server.

Third, given that hypermedia can be generated at low-cost, sending hypermedia to the client should result in a more selective source selection. These benefits are unlikely to outweigh the extra overhead. Thus, in correspondence with Question 2.3, we hypothesize the following:

Hypothesis 9: A hypermedia-based source selection decreases average query execution time drastically compared to the naive triple pattern-wise source selection on all sources.

Query Result Completeness

In the following, we study the completeness and accuracy of the discovery outcome from Table 3 and Table 4.

A FedBench run was performed with each server as starting point. In terms of completeness, Table 5 shows the measured recall of each query for each run, with the number of results from the naive client run as baseline. In general, many queries returned no results, which results in zero or undefined recall (naive client returned 0 results, marked with "–"). Empty result sets are likely caused by the limitations of the current client, which only consumes the hypermedia controls of one fragment response, i.e. the direct neighbors of that particular server. Therefore, the client cannot retrieve data from interfaces that are more than one link away from the starting point. Given these findings, we can only falsify Hypothesis 7, which clearly needs a more powerful execution algorithm to reach a better coverage in recall.

Using DBpedia as starting point reaches 1.0 recall for most of the queries, because its index also contains summaries for most datasets. Moreover, queries LD5 and LD10 contain predicates from DBpedia, and, thus can be answered by using starting points with links to the DBpedia summary. However, this means the DBpedia server itself should also be a good starting point, but this is not reflected in the results. This likely caused by an error during execution, e.g., a failed request.

The low recall scores demand a more intelligent client algorithm that can deal with multiple hops, while keeping query execution scalable. Another first enhancement would be to define heuristics to estimate the levels of neighboring links to traverse. While it is unfeasible to blindly traverse interfaces, this would potentially consume the whole Web, a possible improvement is a dynamic hop count based on, for instance, the number of joins in a query. The client could also rely on link traversal during execution to reach non-indexed interfaces that are out of immediate reach.

Server Response Overhead

We tested the impact of our discovery approach on a server responding to a querying client. As stated above, a client solves complex queries by retrieving a series of fragments from a server. Hypermedia is added to each response, thus the overhead is measured in hypermedia construction time.

Figure 5 shows the average http response time per server. An overhead below 10% is considered acceptable ($\alpha = 0.05$). All servers respond within 3ms on average, confirming the lightweight character of Triple Pattern Fragments servers in combination with an hdt data source. Hypermedia control construction takes up a steady 5% of the total response time, which is negligible. The T-value is -16.927192 with $p < 0.001$. Therefore, we can validate Hypothesis 8, as response time increase fits our expectations.

Note that the construction time can increase with the amount of summaries in the index, which is for the current set-up a maximum of 7. However, the current index is implemented as a simple compact JavaScript object, which can be replaced with a more efficient lookup index as well.

Query Execution Time

We measure the impact of using summary indexes and hypermedia on the average execution time. Figure 6 shows the average execution time for the naive client on index-free servers and the hypermedia-based client on servers with a full index. The results achieved a recall value of 1.0, and are ordered descending according to the number of results. We note a one-tailed T-value of 1.2 with a corresponding $p = 0.122$, which can be considered insignificant ($\alpha = 0.10$). For most queries, we notice a small performance increase for the hypermedia-based client, caused by its more selective nature. However, the queries LS3, CD3, and LD4 show an increase of 50% or more, which indicates impact. They contain a triple pattern with a bound to the object, which is very selective. This enables the indexes to be very selective for one or more generic patterns, i.e. ?x rdf:type ?y, ?x owl:sameAs ?y, or ?x foaf:type ?y. This strictly reduces the amount of requests sent, in contrast to the naive client, which keeps accessing all servers. The query CD6 shows a slight decrease in performance. Since its only bounded object is a literal, the index cannot be selective at first, as literals are not included in the summaries. Thus, the hypermedia overhead outweighs the gain on future iterations of the query algorithm.

Table 5. The recall of each query's results depends on the starting dataset. Due to only using links one level deep, only interfaces with the relevant interfaces as direct neighbours contribute to recall

	DBP	NYT	LMD	JMD	GN	SDF	DB	KC
CD1	1.00	0.86	0.86	0.00	ERR	0.86	0.86	0.00
CD2	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD3	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD4	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD5	0.00	0.00	ERR	0.00	ERR	0.00	0.00	0.00
CD6	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD7	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LS1	1.00	0.00	0.00	0.00	ERR	0.00	1.00	1.00
LS2	1.00	0.00	0.00	0.00	ERR	0.00	0.96	0.96
LS3	–	–	–	0.00	ERR	–	–	–
LS4	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LS5	–	–	–	0.00	ERR	–	–	–
LS6	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD1	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD2	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD3	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD4	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD5	0.00	1.00	1.00	ERR	ERR	1.00	1.00	0.00
LD7	0.32	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD8	–	–	–	0.00	ERR	–	–	–
LD9	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD10	0.00	1.00	1.00	0.00	ERR	1.00	1.00	0.00

(DBP=DBpedia,NYT=NY Times,LMD=LinkedMDB,JMD=Jamendo, GN=Geonames,SDF=SW Dog Food,DB=Drugbank,KC=KEGG–ChEBI)

Next, we compared the execution time for the discovery index set-up with the full index set-up. We selected the queries that were answered with 1.0 recall by at least one starting point. If there are multiple starting point for one query, we chose the one with the highest execution time. The results are shown in Figure 7. We noted a one-tailed T-value of 1.866 with a corresponding $p = 0.042$ ($\alpha = 0.10$).

There is an insignificant decrease for most queries, with the exception of LS3, CD4 and CD2. These queries clearly benefit from the smaller index, which only gives a few links to check.

In general, the findings above are insufficient to accept Hypothesis 9. However, the results indicate potential for hypermedia in source selection, in particular when using more targeted indexes created by discovery. A more thorough evaluation is advised with a larger number of datasets and a more extensive query mix.

CONCLUSION

This paper discussed the impact of Linked Data discovery on source selection for query federation. This is a crucial aspect for evolving the Web towards a global, machine understandable data space.

Figure 5. Hypermedia construction time is negligible compared to the total response time

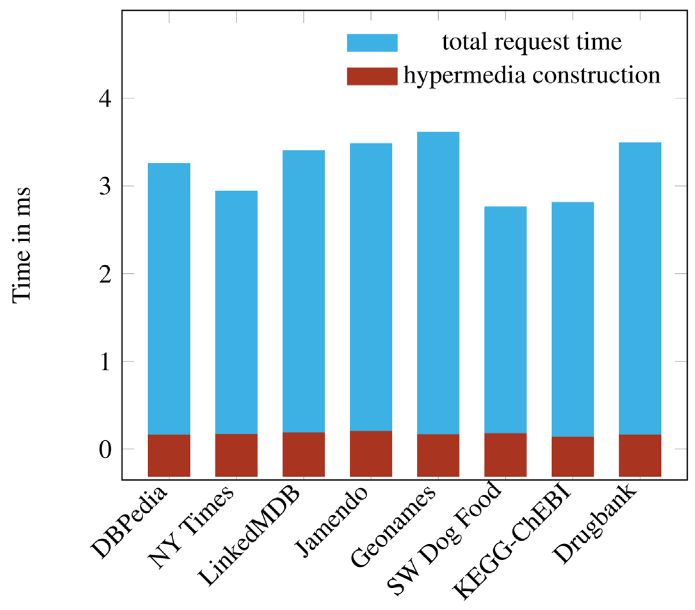


Figure 6. Hypermedia-based querying with full-index servers decrease the average execution time with 50% compared to the naive federation approach

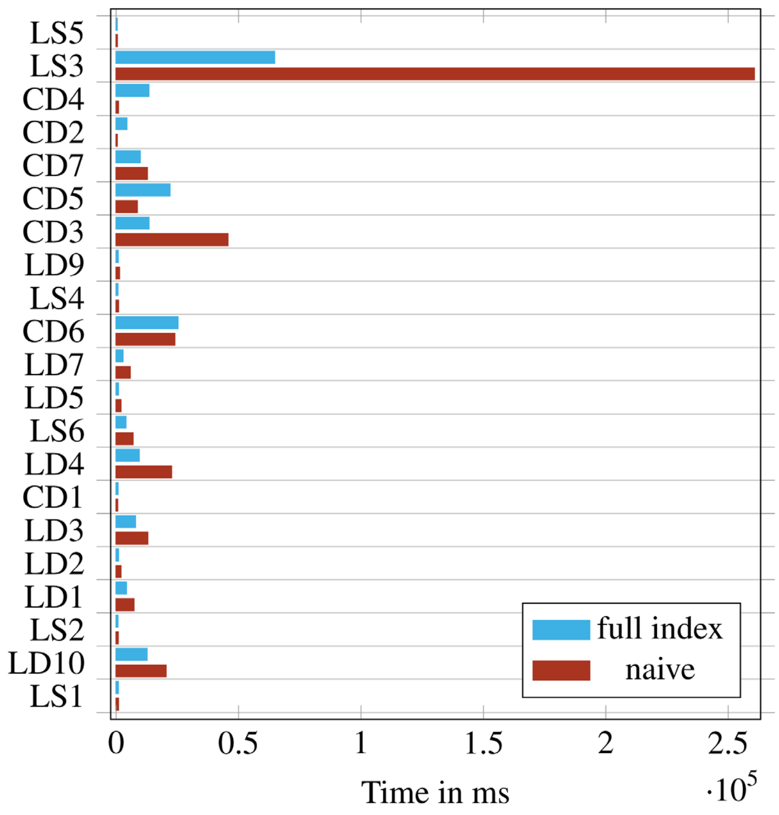
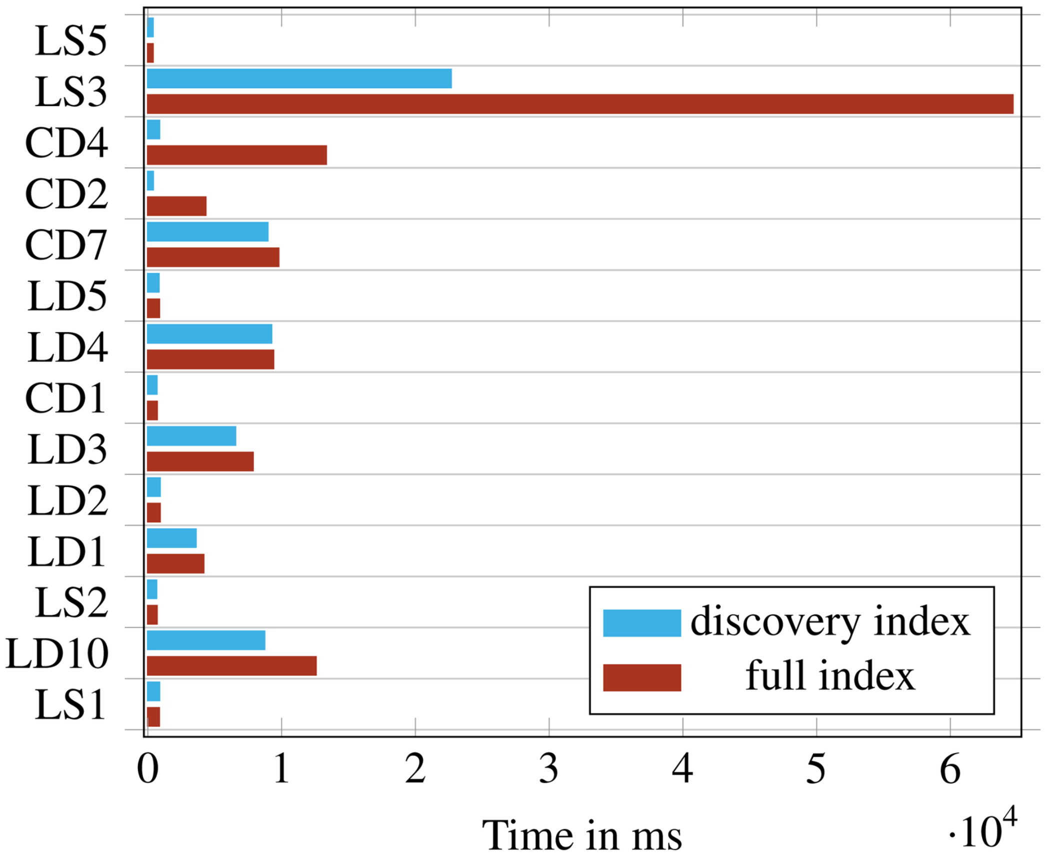


Figure 7. Hypermedia-based querying with a discovery index is slightly faster compared to full-index servers, and for some queries up to 50% faster



Therefore, we proposed a hypermedia-based approach to benefit federated query processors. In addition, we defined a framework to evaluate this approach, covering the quantitative aspects of a discovery process. These quantitative aspects cover the non-functional requirements completeness and accuracy, and the functional requirements bandwidth usage, execution time, and server overhead. Existing processes using discovery can thus be formalized, enabling comparability between approaches and the development of proper methodology. However, the categorization and metrics are high-level, and need to be studied in-depth in the future.

Just like hyperlinks fade the boundaries between different Web interfaces for humans, a discovery process should fade the distinction between query execution and federated query execution. The experimental results are promising in terms of server overhead and completeness in the pre-query phase. When executing federated queries, our approach shows a general decrease in execution time, but very poor recall. We conclude that the link traversal has been simplified, but still requires algorithms that intelligently consume hypermedia. For future work, we imagine use cases that do not need 100% completeness, in favor of speeding up query execution time. One direct improvement is taking into account practical differences between Linked Data interfaces, such as response times or page size. Also, query efficiency can be improved further by integrating existing source selection methods. The query algorithm itself can take advantage of the data summaries currently used for discovery. Finally, future work may research whether preprocessing the data server-side introduces benefits both for the discovery mechanism as for query execution as such.

REFERENCES

Acosta, M., Vidal, M. E., Lampo, T., Castillo, J., & Ruckhaus, E. (2011). ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *The Semantic Web–ISWC 2011* (pp. 18–34). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-25073-6_2

Alexander, K., & Hausenblas, M. (2009). Describing Linked datasets–On the design and usage of VoID, the Vocabulary of Interlinked Datasets. Proceedings of the Linked Data on the Web Workshop (LDOW 09), Madrid, Spain: CEUR.

Basters, U., & Klusch, M. (2006). RS2D: fast adaptive search for semantic web services in unstructured P2P networks. In The Semantic Web-ISWC 2006 (pp. 87-100). Berlin, Germany: Springer-Verlag.

Berners-Lee, T. (1997, January 6). Axioms of Web architecture: Metadata. Retrieved from https://www.w3.org/DesignIssues/Metadata.html

Berners-Lee, T. (2006, July 27). Linked Data – Design issues. Retrieved from http://www.w3.org/DesignIssues/LinkedData.html

Böhm, C., Kasneci, G., & Naumann, F. (2012). Latent topics in graph-structured data.*Proceedings of the 21st ACM international conference on Information and knowledge management* (pp. 2663-2666). Maui, HI: ACM.

Böhm, C., Lorey, J., & Naumann, F. (2011). Creating VoID descriptions for Web-scale data. *Web Semantics: Science, Services, and Agents on the World Wide Web*, *9*(3), 339–345. doi:10.1016/j.websem.2011.06.001

Buil-Aranda, C., Hogan, A., Umbrich, J., & Vandenbussche, P. Y. (2013). Sparql web-querying infrastructure: Ready for action? In *The Semantic Web–ISWC 2013* (pp. 277–293). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-41338-4_18

Cyganiak, R., Wood, D., & Lanthaler, M. (2014, February 25). RDF 1.1 Concepts and Abstract syntax (W3C Recommendation). Retrieved from http://www.w3.org/TR/rdf11-concepts/

Datahub. (n. d.). Retrieved from http://datahub.io/

Datasets. (n. d.). Retrieved from http://fedbench.fluidops.net/resource/Datasets

DBpedia. (n. d.). Retrieved from http://dbpedia.org/

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., & Gori, M. (2000). Focused Crawling Using Context Graphs. *Proceedings of the 26th International Conference on Very Large Data Bases* (pp. 527-534). Cairo, Egypt: ACM.

DrugBank. (n. d.). Retrieved from http://www.drugbank.ca/

Dürst, M., & Suignard, M. (2004). Internationalized resource identifiers (IRIs) (No. RFC 3987). Retrieved from http://www.rfc-editor.org/info/rfc3987

Feigenbaum, L., Williams, G. T., Clark, K. G., & Torres, E. (2013, March). SPARQL 1.1 protocol (W3C Recommendation). Retrieved from http://www.w3.org/TR/sparql11-protocol/

Fetahu, B., Dietze, S., Nunes, B. P., Casanova, M. A., Taibi, D., & Nejdl, W. (2014). A scalable approach for efficiently generating structured dataset topic profiles. In *The Semantic Web: Trends and Challenges* (pp. 519–534). Basel, Switzerland: Springer International Publishing. doi:10.1007/978-3-319-07443-6_35

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation).

Forchhammer, B., Jentzsch, A., & Naumann, F. (2014). LODOP-Multi-Query Optimization for Linked Data Profiling Queries. In *International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES)*. Anissaras, Greece: CEUR.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching.*Proceedings of the 1984 ACM SIGMOD international conference on Management of data* (Vol. 14, No. 2, pp. 47-57). New York, NY: ACM. doi:10.1145/602259.602266

Harris, S., & Seaborne, A. (2013, March 21). SPARQL 1.1 query language (W3C Recommendation). Retrieved from http://www.w3.org/TR/sparql11-query/

Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K. U., & Umbrich, J. (2010). Data summaries for on-demand queries over linked data.*Proceedings of the 19th international conference on World Wide Web* (pp. 411-420). Raleigh, NC: ACM. doi:10.1145/1772690.1772733

Hartig, O. (2012). SPARQL for a Web of Linked Data: Semantics and computability. In *The Semantic Web: Research and Applications* (pp. 8–23). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-30284-8_8

Hartig, O. (2013). An overview on execution strategies for Linked Data queries. *Datenbank-Spektrum*, *13*(2), 89–99. doi:10.1007/s13222-013-0122-1

Heath, T., & Bizer, C. (2011). Linked Data: Evolving the Web into a Global Data Space (1st ed.). Morgan & Claypool.

Hypertext transfer Protocol HTTP/1.1. (1999). The Internet Society. Retrieved from http://tools.ietf.org/html/rfc2616#\page-140

Kaoudi, Z., Kyzirakos, K., & Koubarakis, M. (2010). SPARQL query optimization on top of DHTs. In *The Semantic Web–ISWC 2010* (pp. 418–435). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-17746-0_27

Khan, A. B., & Matskin, M. (2010). Agora framework for service discovery and resource allocation. Proceedings of the Internet and Web Applications and Services (ICIW 2010) (pp. 438–444). Barcelona, Spain: IEEE. doi:10.1109/ICIW.2010.72

Klusch, M. (2014). Service discovery. In *Encyclopedia of Social Network Analysis and Mining* (pp. 1707–1717). New York, NY: Springer.

Klusch, M., & Zhing, X. (2008). Deployed semantic services for the common user of the web: A reality check. *Proceedings of the IEEE International Conference on Semantic Computing* (pp. 347-353). Santa Clara, CA: IEEE. doi:10.1109/ICSC.2008.12

Ladwig, G., & Tran, T. (2010). Linked data query processing strategies. In P. Patel-Schneider et al. (Eds.), *The Semantic Web–ISWC 2010* (pp. 453–469). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-17746-0_29

Li, X., Niu, Z., & Zhang, C. (2014). Towards efficient distributed SPARQL queries on linked data. In X.-h. Sun et al. (Eds.), *Algorithms and architectures for parallel processing* (pp. 259–272). Basel, Switzerland: Springer International Publishing. doi:10.1007/978-3-319-11194-0_20

Linked Open Vocabularies. (n. d.). Retrieved from http://lov.okfn.org/

Montoya, G., Vidal, M.-E., & Acosta, M. (2012). A Heuristic-Based Approach for Planning Federated SPARQL queries. In *Consuming Linked Data (COLD 2012)*. Boston, MA: CEUR.

Mühleisen, H., & Jentzsch, A. (2011). Augmenting the Web of data using referers. Proceedings of the Linked Data on the Web workshop (LDOW 2011). Hyderabad, India: CEUR.

Neumann, T., & Weikum, G. (2010). x-RDF-3X: Fast querying, high update rates, and consistency for RDF databases. *Proceedings of the VLDB Endowment*, *3*(1-2), 256–263. doi:10.14778/1920841.1920877

Nginx. (n. d.). Retrieved from http://nginx.org/

Paret, E., Woensel, W. V., Casteleyn, S., Signer, B., & Troyer, O. D. (2011). Efficient querying of distributed RDF sources in mobile settings based on a source index model. *Procedia Computer Science*, *5*, 554–561. doi:10.1016/j.procs.2011.07.072

Paulheim, H., & Hertling, S. (2013). Discoverability of SPARQL endpoints in Linked Open Data.*Proceedings of the ISWC 2013 Posters & Demonstrations Track*. Sydney, Australia: CEUR.

Queries#classification. (n. d.). Fluidops.net. Retrieved from http://fedbench.fluidops.net/resource/Queries#Classification

Quilitz, B., & Leser, U. (2008). Querying distributed RDF data sources with SPARQL. In S. Bechhofer, M. Hauswirth, J. Hoffmann, & M. Koubarakis (Eds.), *The Semantic Web: Research and Applications* (Vol. 5021, pp. 524–538). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-540-68234-9_39

Re3data.org. (n. d.). Retrieved from http://www.re3data.org/

Rakhmawati, N. A., Umbrich, J., Karnstedt, M., Hasnain, A., & Hausenblas, M. (2013). Querying over federated SPARQL endpoints—a state of the art survey. In *Knowledge Engineering and the Semantic Web*. Berlin, Germany: Springer-Verlag.

Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., & Ngonga Ngomo, A. C. (2015). A fine-grained evaluation of SPARQL endpoint federation systems. Semantic Web, (Preprint), 1-26.

Saleem, M., & Ngomo, A.-C. N. (2014). Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *The Semantic Web: Trends and Challenges* (pp. 176–191). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-319-07443-6_13

Schmachtenberg, M., Bizer, C., & Paulheim, H. (2014). Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014* (pp. 245–260). Basel, Switzerland: Springer International Publishing. doi:10.1007/978-3-319-11964-9_16

Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., & Tran, T. (2011). Fedbench: A benchmark suite for federated semantic data query processing. In *The Semantic Web–ISWC 2011* (pp. 585–600). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-25073-6_37

Schwarte, A., Haase, P., Hose, K., Schenkel, R., & Schmidt, M. (2011). FedX: a federation layer for distributed query processing on linked open data. In *The Semantic Web: Research and Applications* (pp. 481–486). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-21064-8_39

Server.js linked Data fragments. (n. d.). Retrieved from https://github.com/LinkedDataFragments/Server.js

Umbrich, J., Hose, K., Karnstedt, M., Harth, A., & Polleres, A. (2011). Comparing data summaries for processing live queries over Linked Data. *World Wide Web (Bussum)*, *14*(5-6), 495–544. doi:10.1007/s11280-010-0107-z

Using Seealso. (n. d.). W3C. Retrieved from http://www.w3.org/wiki/UsingSeeAlso

Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., & Vander Sande, M. Van de Walle, R. (2014). Querying datasets on the Web with high availability. In P. Mika et al. (Eds.), Proceedings of the 13th International Semantic Web Conference (Vol. 8796, pp. 180–196). Berlin, Germany: Springer-Verlag.

Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., & De Meester, B. … Colpaert, P. (2016). Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. *Journal of Web Semantics*. doi:10.1016/j.websem.2016.03.003

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., & Miller, J. (2005). METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. *Information Technology Management*, *6*(1), 17–39. doi:10.1007/s10799-004-7773-4

Williams, G. (2013). SPARQL 1.1 service description (W3C Recommendation). Retrieved from http://www.w3.org/TR/sparql11-service-description/

Miel Vander Sande graduated cum laude as Bachelor in Multimedia and Communication technology at the University College of West-Flanders, Belgium in 2008. He continued his studies with in Industrial Sciences at the University College of Ghent, specializing in Information and Communication technology. In 2010 he researched low cost interfaces for indoor human activity tracking, based on RFID and WIFI technology as his Master thesis for the Polytechnic University of Valencia in Spain. He received his MSc in Industrial Science degree cum laude that same year. Finally, he finished his education with an additional year of teacher studies in Informatics. In September 2011, Miel joined Data Science Lab (http://datasciencelab.ugent.be) of Ghent University as a full-time researcher. His main interests and expertise are (Linked Open) Data publishing, Read-Write Web, technological support for open data legislation, ontology mapping and data transformation. Miel was involved in several Flemish and European research projects involving Semantic Web technologies and Linked Open Data

Ruben Verborgh is a researcher in semantic hypermedia at Ghent University – iMinds, Belgium and a postdoctoral fellow of the Research Foundation Flanders. He explores the connection between Semantic Web technologies and the Web's architectural properties, with the ultimate goal of building more intelligent clients. Along the way, he became fascinated by Linked Data, REST/hypermedia, Web APIs, and related technologies. He's a co-author of two books on Linked Data, and has contributed to more than 150 publications for international conferences and journals on Web-related topics.

Anastasia Dimou holds an MSc in e-Government technologies from University of Trento, Italy and an MSc in Web Science from Aristotle University, Greece. Since 2013 she has served as a researcher in Semantic Web and Linked Open Data technologies. Anastasia has been involved in different research projects. Her main interest and expertise is related to Linked Data Generation and Publication, Data Quality and Integration, and Knowledge Representation and Management.

Pieter Colpaert is a researcher at DataScience Lab - Ghent university - iMinds. He holds a master degree in engineering from College Ghent since 2012. Before starting his research position at DataScience Lab, he was running a digital signage start-up called FlatTurtle (http://flatturtle.com) which focuses on data visualisation, co-founded a not-for profit organisation to promote Open Knowledge in Belgium (http://okfn.be), and started The DataTank (http://thedatatank), an open-source tool for smart cities which enables them to open up their data in a machine-readable fashion.

Erik Mannens has served as Professor at Data Science Lab / CTO Data Science at iMinds / Research Manager at Ghent University since 2005 where he has successfully managed +50 projects. He received his PhD degree in Computer Science Engineering (2011) at UGent, his Master's degree in Computer Science (1995) at K.U. Leuven University, and his Master's degree in Electro-Mechanical Engineering (1992) at KAHO Ghent. Before joining iMinds-Ghent University-DataScienceLab in 2005 as research manager, he was a software engineering consultant and Java architect for over a decade. His major expertise is centered around big data analytics, metadata modeling, semantic web technologies, broadcasting workflows, iDTV and web development in general. He is involved in several projects as senior researcher and finished up his PhD on Semantic News Production; he was co-chair of the W3C Media Fragments Working Group and actively participating in other W3C's semantic web standardization activities (Media Annotations, Provenance, Hydra, Linked Data Platform, and eGovernment). Since 2008 Erik is paving the Open Data path in Flanders. He stood at the cradle of the first Hackatons and is a founding member of the Open Knowledge Foundation. (Belgian Chapter). Since then, he is frequently invited as Open Data evangelist at national and international events. He currently actively participates in W3C's eGov and Data On The Web working groups. Furthermore, his team is owner of the Open Sourced Linked Open Data Publishing frameworks TheDataTank, R&Wbase, RML, and Linked Data Fragments. On all of these subjects he has published +180 papers and book chapters. He is also member of the technical committee and/or organizing committee of several high level journals and conferences. His full curriculum can be found on LinkedIn.