KBox – Transparently Shifting Query Execution on Knowledge Graphs to the Edge

Edgard Marx*, Ciro Baron*, Tommaso Soru*, Sören Auer†
*AKSW, University of Leipzig, Germany
{marx,tsoru,cbaron}@informatik.uni-leipzig.de
http://aksw.org
†Computer Science Institute, University of Bonn
auer@cs.uni-bonn.de

***Abstract*—The Semantic Web architecture choices lead to a tremendous amount of information published on the Web. However, to query and build applications on top of Linked Open Data is still significantly challenging. In this work, we present Knowledge Box (KBox), an approach for transparently shifting query execution on Knowledge Graphs to the edge. We show that our approach makes the consumption of Knowledge Graphs more reliable and faster than the formerly introduced methods. In practice, KBox demands less time and resources to setup than traditional approaches, while being twice as fast as SPARQL endpoints, even when serving a single client.**

I. Introduction

Since the inception of the Web of Data, many open knowledge graphs were made available in RDF format. Examples of such knowledge graphs are DBpedia [11], Freebase [4] and Wikidata [19]. Together, these knowledge sources alone encompass more than three billions facts covering a multitude of domains. Despite this data is freely available, lay users as well as researchers and enterprises still face difficulties in consuming RDF. The main obstacle is that using the data is still a very cumbersome and resource-demanding task. As a result, users often rely on publicly available SPARQL endpoints and RDF dump files.

On the one hand side, SPARQL endpoints are not reliable, as it has been shown that the query evaluation problem for SPARQL is "*PSPACE*-complete even without filter conditions" [15]. Therefore, high demand services are generally expensive to host, "which makes reliable public SPARQL endpoints an exceptionally difficult challenge" [17]. For instance, a study monitoring 427 endpoints for 27 months shows that SPARQL endpoints have "an average fixed HTTP cost of ∼300 ms per query". Moreover, the mean endpoint availability of the SPARQL enpoints decreased over time (i.e., from 83% in the beginning to 51% at the end of the experiment), while at least 24.3% of the SPARQL endpoints were always down [5]. To tackle the reliability problem of SPARQL endpoints, some proposed approaches reach from (i) improved indexing techniques [20], [8] to (ii) novel architecture patterns such as Linked Data fragments [17]. However, these methods often impute limitations as, for example, high network bandwidth consumption [17] as well as restrictions on SPARQL features – i.e., some indexes restrict the SPARQL query features to only basic graph patterns [8].

On the other hand side, consuming RDF dump files can be a very cumbersome, time-consuming and resource-demanding task as there is a high effort necessary for: (1) identifying; (2) downloading, and; (3) setting up the infrastructure for RDF data management including indexing the desired portion of the RDF graph (see Section V). All these obstacles make it difficult for users to query Linked Data as well as build applications on top of it.

In this work, we present the Knowledge Box (KBox) approach to transparently shift the query execution on knowledge graphs to the user or application (i.e., the edge of the network). The main contributions of this work are:

- an approach for transferring the query execution from the server to the user or application;
- a decentralized architecture for publishing and dereferencing RDF Knowledge Graphs;
- an extensive evaluation of different methods for publishing RDF on the Web, including KBox.

The remainder of this article is structured as follows. Section II presents the preliminaries. Section IV presents the approach, its concept and architecture. Section V evaluates the scalability of the approach by comparing it with the state-of-the-art semantic technologies as well as standard benchmarks using the DBpedia knowledge base. Related work is reviewed in Section III. Section VI concludes with an outlook on future work.

II. Preliminaries

The Linked Open Data (LOD) was built on top of the Resource Description Framework (RDF). The main idea is that resources are published on the Web where they can be dereferenced and consumed. A resource is the most atomic unity of the Web of Data. RDF adds meta information to existing resources and allows applications to better process the data. Therefore, the Web of Data is composed of resources and their meta information. An RDF knowledge base is composed of triples (or statements) which consist of: (1) subject, (2) predicate and (3) object. The definition of an RDF is closely following [15].

Definition 1 (RDF definition): Let I, B, and L be pairwise disjoint infinite sets of IRIs, blank nodes, and RDF literals, respectively. A triple $(v_s, v_p, v_o) \in (I \cup B) \times I \times (I \cup B \cup L)$

is called an RDF *triple*. In this tuple, v_s is the *subject*, v_p the *predicate* and v_p the *object*. $T = I \cup B \cup L$ is the set of RDF terms.

A set of such triples is called an RDF graph.[1] In this work, we refer to an RDF graph as an RDF knowledge graph or simple knowledge graph.

III. Related Work

In this section, we discuss different approaches that are related to our work. We start by discussing the most common architecture patterns for publishing RDF graphs. Thereafter, we discuss repositories designed to facilitate the RDF graph discovery. Finally, we present some approaches that make use of decentralized architectures as well as RDF.

A. LOD Architectures

The use of RDF technologies leads to a tremendous growth of data published as RDF. However, RDF graphs may be published using different approaches, according to server specifications and end-user requirements. The most common architecture patterns to publish RDF data on the Web are: (1) data dump files, (2) publication of data as individual de-referenceable resources, (3) SPARQL endpoints and (4) Linked Data Fragments.

*a) **Dump Files**:* The first and most widely adopted architecture choice is publishing RDF in a raw format as dump files. Examples of serialization formats for RDF dump files are Turtle and N-triples. However, the effort required to start querying such data is very time-consuming. For instance, to download the RDF dump files and load DBpedia 2015-10 into a triple store may take as long as 15 hours using standard hardware (see section V). In addition, expertise and time are required for: (1) identifying; (2) downloading; and (3) setting up the infrastructure for RDF data management including indexing the desired portion of the RDF graph (see Section V).

Alternatively, indexing and querying functionality can be provided by the data provider itself at the expense of additional resource requirements. Publication patterns in this regard are SPARQL endpoints and LDFragments.

*b) **SPARQL endpoints**:* The second most used architecture pattern is to publish RDF graphs via directly queryable `SPARQL` [10] endpoints. With SPARQL endpoints, the client uses the HTTP protocol and the SPARQL query language[2] in order to create views on the available RDF graphs. In this approach, the data processing is performed by the sever that handles the treatment of complex SPARQL queries. There are many triple stores designed for publishing RDF data as SPARQL endpoints. Examples of such triple stores are Virtuoso [7] and Apache Jena TDB [9].

The SPARQL endpoint architecture is less cumbersome for users as the data published via SPARQL is ready-to-use. However, SPARQL endpoints suffer from reliability, scalability and resource requirement problems when compared to the method of distributing the data as dump files. The main problem is that the "majority of public SPARQL endpoints had an uptime of less than 95%" [17]. An additional challenge for publishing RDF graphs as SPARQL endpoints is that the total number of users, requests, and the processing cost can be very high due to the SPARQL expressivity and the popularity of the data. As a result, publishing RDF data via SPARQL endpoints might not be a feasible option for many data providers.

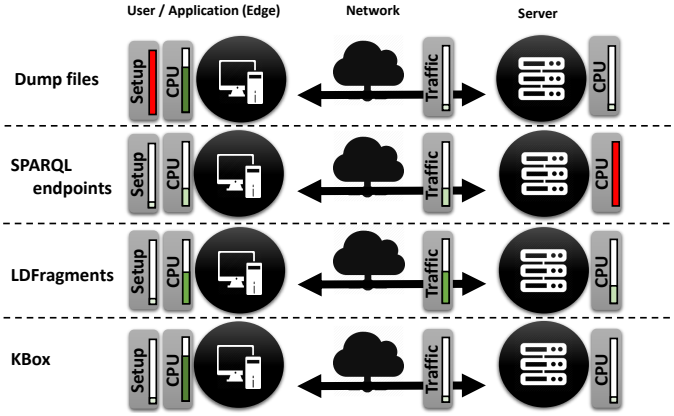


Fig. 1. Network `Traffic`, `CPU` and `Setup` time consumption requirements on the `Client` and `Server` for the different LD publication architecture patterns (`Dump files`, `SPARQL endpoints`, `LDFragments`, and `KBox`).

*c) **LDFragments**:* The third architecture pattern is LDFragments [18], which was proposed in 2014 with the aim of tackling the reliability and scalability problems prevalent with SPARQL endpoints. LDFragments introduces a low-cost query technique based on triple pattern fragments. Here, the concept of fragments can be aligned with the notion of query evaluation. However, fragments are extended to triple patterns, pages and collections. The core idea is to restrict the operations handled by the server to simple SPARQL Basic Graph Patterns (BGP). Therefore, the portion of the RDF graph defined by the BGP is transferred to the client where the view is finally projected. This architecture is more scalable compared to traditional SPARQL endpoints. However, the use of LDFragments increases the network traffic as well as the processing and memory consumption in the client. LDFragments can fetch data either from an HDT index or triple store. *HDT* [8] is a binary, compressed RDF representation addressing the problem of efficient formats for publication and exchange of RDF data.

As a result, in all of these various architecture patterns, RDF consumption is still a very cumbersome, time-consuming and resource-demanding process. Figure 1 illustrates the effect of the different architecture patterns on `CPU`, `Network Traffic`, and `Setup` time consumption.

B. RDF Repositories

A challenge with current architectures is to localize the desired RDF graph, which leads to the creation of many RDF repositories. *LOD Laundromat* [2] harvests data from

[1] http://www.w3.org/TR/rdf-concepts/

[2] https://www.w3.org/TR/sparql11-query/

the web, extracts statistical data and filters duplicate triples in order to publish the data in a uniform way. Hence, the data is republished as 'clean' dump files also providing a considerable amount of metadata. Moreover, other services aim to centralize either SPARQL endpoints or dump files. *CKAN* (Comprehensive Knowledge Archive Network)[3] is an open-source data portal repository, which provides metadata for datasets and other resources. The most well-known CKAN instance is *The Datahub*[4], which provides an API that can be used to retrieve, update, and insert datasets. Although RDF data can be stored, the client must know how to use CKAN's API, which might be considered an extra layer of difficulty. *RE3data* [14] is a project which aims to centralize repositories like CKAN. Again, the client should have knowledge of consuming data from the API.

Several approaches cetralize SPARQL endpoints to facilitate data search and discoverability. A good example is *SPARQLES* [6], which monitors SPARQL endpoints collected from datahub.io providing status of availability, performance and interoperability. With SPARQLES, users can access more than 500 SPARQL endpoints.

C. Decentralized Architectures

Solid [13] (*Social Linked Data*) is a platform for social Web applications which specifies protocols required for authentication, application-to-server, and server-to-server. The Solid platform is based on *CrossCloud*[5]. The core server supports storage of binary and text resources as well as RDF data, and the graph engine is based on *Apache Jena*. Solid is a good example of a decentralized storage network where a server can communicate using Access Control Lists (ACL) and the REST protocol. The data is stored in the application in a form that supports interoperability, so users can use different applications created independently, with the same data.

IV. The Knowledge Box

To overcome the problems imposed by existing architecture patterns, we propose the Knowledge Box (KBox) concept[6]. The KBox approach relies on distributing ready-to-go knowledge graphs over the Web, transparently shifting the query execution to the edge. *Edge computing* [12] is a new paradigm that consists of pushing, data and processing away from centralized infrastructure to the logical extremes of a network (i.e., the client). This approach is based on replicating the required information across distributed networks. With KBox, the knowledge graphs can be published by communities—herein also called *authorities*—in a ready-to-use format. Example of such authorities are, for instance, DBpedia [11], Freebase [4], and Wikidata [19]. The KBox architecture contains five components, shown in Figure 2:

1) the Knowledge Graph Name System (KNS);
2) the KNS Server;
3) the Knowledge Graph Name (KGN);
4) the Knowledge Graph, and;
5) the User/Application–herein also called *Edge*.

[3]http://ckan.org/
[4]https://datahub.io/
[5]http://crosscloud.org
[6]https://github.com/AKSW/KBox

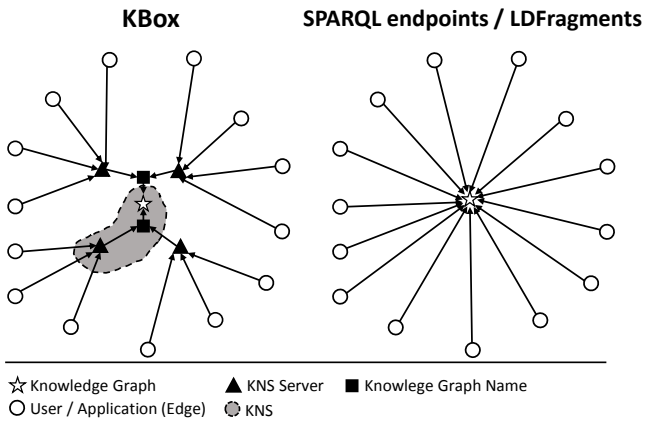


Fig. 2. Comparison between the decentralized KBox architecture and the traditional centralized architectures based on LDFragments and Triple Stores.

In the KBox architecture, the user or application performs queries targeting Knowledge Graph Names. The Knowledge Graph Names (KG Names) are the target RDF graphs that the user or application aims to query. The edge is where KBox is being executed, i.e. a user or an application on a server or a standard user machine. The Knowledge Graph Name is the name of the graph, often represented by an URI. A Knowledge Graph can have different names and be located at different Web addresses.

A. The Knowledge Graph Name System

The KNS Server, the Knowledge Graph Name and the target Knowledge Graph are important components of the KBox architecture. Together, they form the Knowledge Graph Name System or simple KNS. The KNS Server store records comprising the Knowledge Graph Name and Web Address, which enables the KBox Client to automatic identify, locate and dereference knowledge graphs published on the Web. The Knowledge Graph Name is the name of the graph represented by an URI. As mentioned before, Knowledge Graph can have different names and be located at different Web addresses. With the KNS, the Knowledge Graph can be distributed by different authorities in a decentralized manner. For instance, a version of the DBpedia KG can be distributed not only by DBpedia, but also by other authorities. Furthermore, one single version of the same graph can also be distributed by the same authority over different Web Addresses.

The KGN System reinforces the strengths of the KBox architecture that does not rely on a centralized paradigm of the traditional architectures, such as LDFragments and Triple Stores. In fact, decentralized architectures are often used in highly reliable, scalable and available systems such as the Internet itself. Figure 2 depicts a comparision between KBox and traditional architectures based on LDFragments and Triple Stores.

B. Querying Knowledge Graphs on the Edge

The rationale behind the KBox architecture is to transparently shift the execution of SPARQL queries to the Edge without adding configuration or installation overhead. In order to do that, the process flow of the query execution implemented by previous architectures (LDFragments, Triple Stores) is slightly modified (see Figure 3). In KBox, the query execution starts when the User or application provides the KBox Client with a SPARQL query and the target Knowledge Graph Name as parameters. After receiving the parameters, the KBox Client checks the availability of the desired graph in the machine. If the graph is not available, KBox invokes the KNS to resolve the given KG Name.

Through the KNS, the KBox Client checks if the graph is available and published by some authority. In case the KG Name can be resolved using the KNS, KBox Client dereferences and streams the target KG to the Client (i.e., the *Install* process). If the graph is already available, the KBox Client performs the SPARQL query returning the resulting view to the user or application.

C. Architecture

The architecture of KBox server is very simple and consist of a KNS table served by a `HTTP` server (Figure 4). The server operates in a passive mode, and all intelligence is shifted to the client. The KNS table contains KNS entries in format (KGN, URL). When a KGN requested by the user is not locally available, the KBox client opens a `HTTP` connection with the server to check its real URL. As soon as a KGN entry is found in the KNS table, a `HTTP` connection is open with its remote location (URL), and its content is streamed to the client in a correspondent local URL (Table I).

The architecture of KBox client comprises KBox `Core` and `Kibe` libraries (Figure 5). The `Core` library contains the core functions of KBox. It was designed to facilitate application development and dynamic resolution of resources. KBox `Core` is built uppon `URI`, `UNICODE`, `Operational System` (OS) layers and is useful for dereferencing and uniquely identifying resource in the `Network` as well as in the `File System`. It allows users to access and share resources among different applications. Furthermore, the `Core` library enables users to have a local mirror for resources located in the network. By avoiding protocol overheads and resource duplication, applications can perform a more efficient resource storage and access.

The `Kibe` library is an extension of the `Core`. It is designed specially to perform operations of KBox `Core` library into RDF Knowledge graphs. Thus, the `Kibe` library contains both the SPARQL and the Resource Description Framework (RDF) layers. The RDF layer allows to execute operations in RDF data such as read, serialize and decentralization as well as the process. The SPARQL layer comes after the `RDF` layer and is encharged of performing SPARQL operations in knowledge graphs. The `RDF` and `SPARQL` layers are accessible through the Apache Jena framework[7]. On top of `Kibe` there is the `Application`, which can perform all operations of previous layers. For instance, dynamically dereference, aggregate, uniquely identify and execute SPARQL query operations against published knowledge graphs.

Repository	URL
Remote	protocol://domain/subdomain/resource
Local	file:///drive:/kbox/protocol/domain/subdomain/resource

TABLE I
COMPARISION BETWEEN THE REMOTE AND LOCAL REPRESENTATION OF THE SAME URI IN KBOX.

V. EVALUATION

The evaluation was designed to measure the performance of KBox in comparison with other approaches. In order to do that, we divided the evaluation in two parts.

The first part of the evaluation was projected to measure the *setup* time. That is, the time necessary to start using each approach, herein named *setup*. For this part, we use a standard Windows 7 machine equipped with an Intel Core M 620 processor, 6GB of RAM and a 1TB SSD.

The second part of the evaluation was designed to measure the individual performance of each approach when executing SPARQL queries. Therefore, it evaluates the average resource consumption—`RAM`, `DiskSpace`, `CPU`, Network Traffic (`NT Traffic`)—of each approach. In this part of the evaluation, we benchmark `KBox`, `SPARQL endpoints`, and `LDFragments` in both Client and Server side. The set of SPARQL queries selected for the second part were extracted from an SPARQL benchmark. All experiments were performed using the version 2015-10 of the DBpedia knowledge graph. The client was a standard Windows 7 machine equipped with an Intel Core M 620 processor, 6GB of RAM and a 1TB SSD. The server was a virtual instance equipped with a 2.8MHz CPU, 8GB of RAM, and 300GB of disk space. The experiments on SPARQL enpoints were performed using an open-source *Virtuoso Server*[8] version 7.0.0. We implemented KBox as open-source and made it publicly available at `http://github.com/aksw/kbox`.

SPARQL benchmarks Although several benchmarks can be used for assessing the performance of the approaches [3], [1], [16], some of them rely on synthetic data or on synthetic queries[3], [1]. In this work, we use the FEASIBLE benchmark [16]. It consists of a set of 175 queries generated from DBpedia query logs (DBpedia-175). FEASIBLE was chosen because it is based in real SPARQL queries. Moreover, it has a better query selection as well as a larger set of query types.

A. Results

Setup To compare the amount of resources necessary to setup each of the approaches, we took into consideration the

[7]http://jena.apache.org/

[8]http://virtuoso.openlinksw.com/

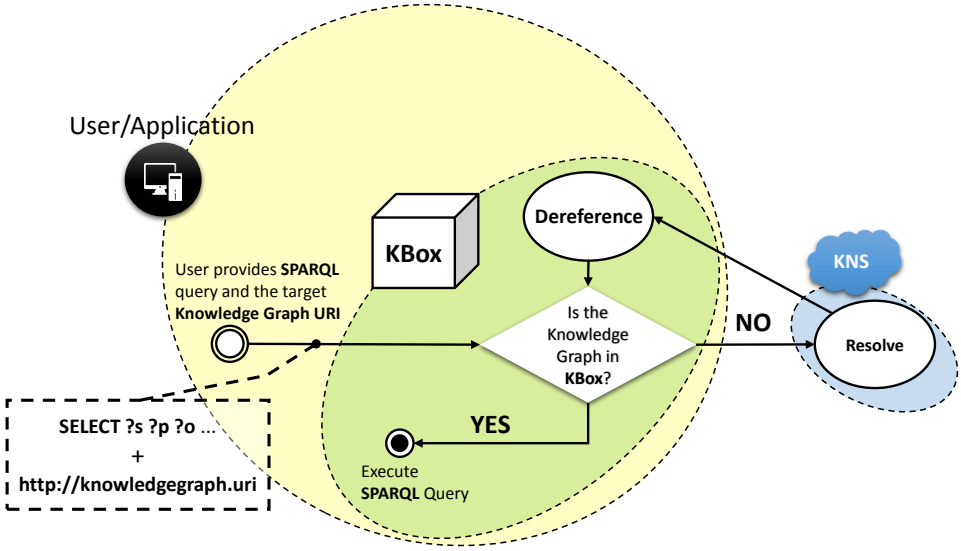


Fig. 3. An overview of the KBox process flow. The process starts when the users provides a SPARQL query and a Knowledge Graph URI; thereafter, KBox checks the availability of the given graph; if the requested knowledge graph is not ready for querying, it is located using a KNS (Knowledge Name Service) and installed. If the graph exists, it is used directly for querying.

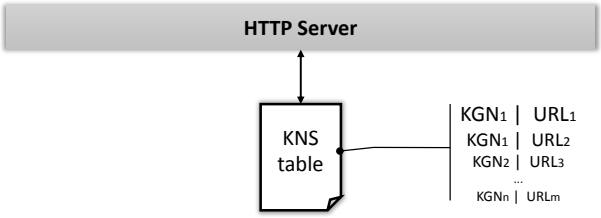


Fig. 4. Overview of the KBox Server Architecture.

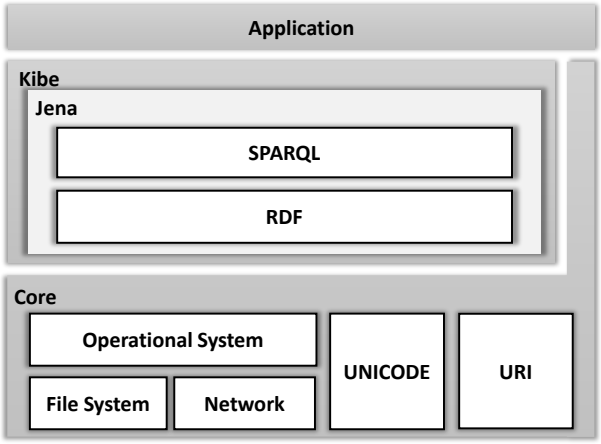


Fig. 5. Overview of the KBox Client Architecture. Depiction of the KBox Client Architecture consisting in `Core` and `Kibe` libraries.

time for dereferencing and indexing the knowledge graph as well as the amount of memory.

Table III shows the `Dereferencing`, `Indexing` and `Total` time as well as `RAM` required to setup LDFragments, SPARQL endpoints and KBox using DBpedia 2015-10, where:

- `Dereferencing` is the time in seconds required to dereference the knowledge graph;
- `Indexing` is the time in seconds required to index the knowledge graph;
- `Total Time` is the sum of `Dereferencing` and `Indexing`, and;
- `RAM` is the total amount of `RAM` required for `Indexing` and/or `Dereferencing`.

The values of `Indexing` and `RAM` displayed for `LDFragments` in Table III are an estimation, based on the growth of Table II.

We tried to setup an HDT version of LDFragments using the same standard user machine first dedicating 4GB of RAM to the loading process, and secondly dedicating 40GB of RAM to the loading process. In both environments, the loading process failed and returned an unexpected `OUT OF MEMORY` error. In order to explore this clear limitation of LDFragment, we conducted an empirical study depicted in Table II. When it comes to large datasets, the loading stage of LDFragments did not perform well due to the linear growth of the memory consumption. If the amount of resources needed grows linearly, we estimated that the loading process of the whole DBpedia dump file would take ∼2 hours consuming ∼60GB of RAM. We have omitted the results for the triple store version of LDFragments because they resemble the SPARQL endpoint results.

a) Resource consumption: Table IV shows the results obtained by each approach when evaluating the FEASIBLE benchmark on either Client (C) or Server (S) sides, where:

- `CPU` is the average percentage of consumed `CPU` per query;
- `DiskSpace` is the total amount of storage in Gigabytes (GB) required for storage;

Approach	Time(s)	RAM(MB)	Input Size(MB)
LDFragments$_{HDT}$	8	487	61
LDFragments$_{HDT}$	15	1,187	456
LDFragments$_{HDT}$	91	2,160	2,212
LDFragments$_{HDT}$	∼6,441	∼60,000	∼122,000

TABLE II
COMPARING THE RUNTIME FOR INDEXING DIFFERENT SLICES OF RDF DATA WITH LDFRAGMENTS. THE VALUES OF THE LAST LINE ARE BASED ON THE GROWTH OF THE VALUES IN THE PREVIOUS LINES.

- `RAM` is the average consumed amount of RAM per query;
- `NT Traffic` is the average network traffic in Kilobytes (KB) per query, measured during the benchmark execution, and;
- `Runtime` is the average runtime per query in seconds.

The graphs displayed in Figure 6 show the normalized values of Table IV. The normalization was done using the highest value in the respective field. For instance, if the consumption of `CPU` by `SPARQL endpoint` in the server was `0.98` while for `KBox` `0.00`, the values for `CPU` were normalized using `0.98`. We setup a working Node.js version of LDFragments on top of an HDT index.

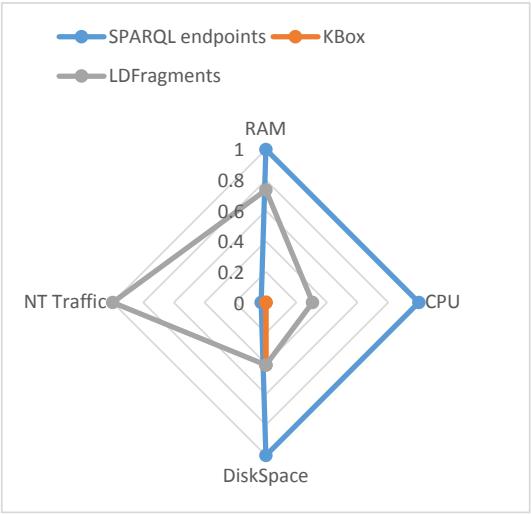
b) Scalability: Figure 7 displays the average time in seconds per query for executing the FEASIBLE benchmark with different number of concurrent clients.

B. Discussion

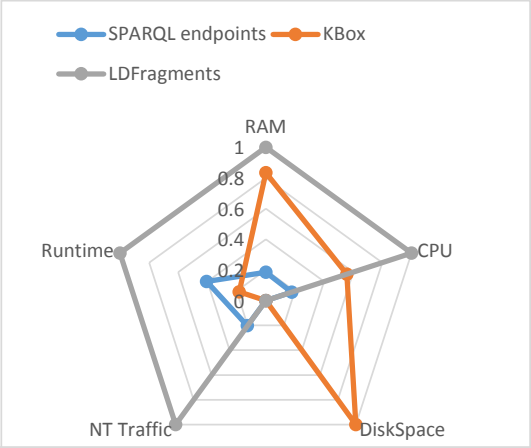
a) Setup: The results in Table III shows that KBox requires less time and resource to setup than other approaches. LDFragments HDT is less time-demanding than traditional SPARQL endpoints. However, when comparing the amount of RAM necessary for indexing, LDFragments HDT became an unreasonable method for ordinary users.

Other than LDFragments HDT, both SPARQL endpoints and KBox requires much less RAM. Precisely, KBox consumed only 50MB which is 80 and 1,200 times respectively less than SPARQL endpoints and LDFragments HDT. Furthermore, the time required to dereference the Knowledge Graph using KBox is slightly higher than other approaches, but when adding the time required for indexing, the KBox setup required almost three times less time than the second best approach (LDFragments).

The difference in the dereferencing time between KBox and other approaches is due to its architecture that requires to resolve the graph name before starting to dereference it. Thereafter, as the graph is also stored in the server in a compressed format, KBox needs extra time to download and decompress the data at the client. Moreover, as the graph is distributed in a ready-to-go format, the indexing time is dismissed.



(a) Comparing the average resource consumption (`RAM`, `DiskSpace`, `NT Traffic`, `CPU`) on queryng `DBpedia 2015-10` with `SPARQL endpoints`, `KBox` and `LDFragments` on `Server` side with one `Client`.



(b) Comparing the average resource consumption (`RAM`, `Disk`, `NT Traffic`, `CPU`, `Runtime`) for queryng DBpedia 2015-10 with `SPARQL endpoints`, `LDFragments` and `KBox` on `Client` side.

Fig. 6. Comparing average resource consumption (`DiskSpace`, `NT Traffic`, `RAM`, `CPU`, `Runtime`) of different approaches (`SPARQL endpoints`, `KBox` and `LDFragments`) on queying `DBPedia 2015-10` on either `Server` and `Client` side. The values displayed in these charts are normalized with the highest values from Table IV. In these charts, `DiskSpace` reflects the total disk space consumption; `Runtime` is the average query runtime; `NT Traffic` is the average network traffic per query. We considered either incoming and outcomig network traffic (I/O), and; `CPU` is the average CPU consumed by the process (`Client`/`Server`).

Approach	Dereferencing(s)	Indexing(s)	Total Time(D+I)	RAM(MB)
LDFragments$_{HDT}$	2,350	~6,441	~8,791	~60,000
SPARQL endpoints	2,350	52,536	54,886	4,000
KBox	3,000	—	3,000	50

TABLE III
COMPARING THE TOTAL TIME(S) AND RAM (MB) FOR SETUP DIFFERENT APPROACHES USING DBPEDIA 2015-10.

Approach	CPU(%)		DiskSpace(GB)		RAM(MB)		NT Traffic(KB)		Runtime(s)
	C	**S**	**C**	**S**	**C**	**S**	**C**	**S**	
SPARQL endpoints	8	98	0	48	74	738	44.65	0.51	4.9
LDFragments$_{HDT}$	45	30	0	18	400	434	220.18	18.16	12.1
KBox	25	0	62	18	334	0	0	0	2.2

TABLE IV
COMPARING AVARAGE CPU, DISKSPACE (GB) AND RAM (MB) CONSUMPTION EITHER ON (**C**)LIENT AND (**S**)ERVER SIDE FOR DIFFERENT APPROACHES ON QUERYING DBPEDIA 2015-10. IN THIS TABLE, `CPU` IS THE AVERAGE PERCENTAGE OF `CPU` CONSUMED BY THE PROCESS (CLIENT/SERVER) ON EXECUTING EACH QUERY. `DISKSPACE` SHOWS THE TOTAL DISK SPACE CONSUMPTION IN GIGABYTES (GB); `RAM` SHOWS THE AVERAGE CONSUMPTION OF RAM PER QUERY IN MEGABYTES (MB); `NT TRAFFIC` IS THE AVERAGE NETWORK TRAFFIC IN KILOBYTES (KB). WE CONSIDERED EITHER INCOMING AND OUTCOMIG NETWORK TRAFFIC (I/O), AND; `RUNTIME` IS THE AVERAGE RUNTIME PER QUERY IN SECONDS (S).

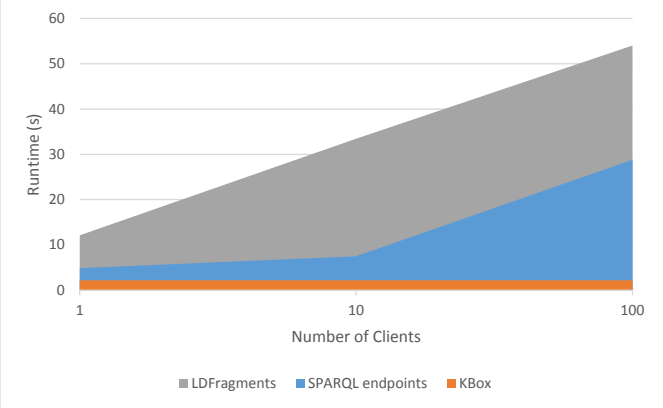


Fig. 7. An overview of the behavior of KBox, LDFragments and SPARQL endpoints in a concurrent enviroment. process flow. The graph displays the average time in seconds per query on executing FEASIBLE benchmark with different number of concurrent clients. In the graph, the runtime per query of KBox remains constant while LDFragments and SPARQL grows resp. linearly and exponentially.

b) Resource consumption: The results in Table IV and Figure 6 show that KBox is more efficient in terms of `DiskSpace`, `RAM`, `NT Traffic`, and `CPU` than other approaches on the *Server* side (when executing SPARQL queries). This is mainly due to KBox shifting the query execution to the user or application, while other approaches centralize the execution in a single point (cf. Figure 2). Although the average amount of `DiskSpace`, `RAM` and `NT Traffic` is fair for SPARQL endpoints, it is interesting to observe the average `CPU` load required per query execution (98%). This outcome demonstrates the high resource requirements for publishing RDF graphs as SPARQL endpoints, which contributes a lot to the reliability and scalability problems.

The results obtained on the *Client* side demonstrate that the KBox approach presents an intermediate resource-demanding. Table IV and Figure 6 show that KBox requires more `CPU`, `DiskSpace` and `RAM` than SPARQL endpoints while dispensing the `NT Traffic`. In the other side, the `Runtime` measurements show that KBox is on average twice as fast than traditional SPARQL endpoints when executing SPARQL queries (even when serving the graph to a single client). This result is because—when executing an SPARQL query in the Edge—KBox avoids data transfer overheads such as the (de)serialization process and the `HTTP` protocol.

c) Scalability: The results in Figure 7 show that the runtime per query of KBox remains constant while the one for SPARQL endpoints grows exponentially. Therefore, they demonstrate that KBox is more scalable than traditional architectures based on SPARQL endpoints.

d) Outcomes: Although the results obtained by KBox are encouraging, we highlighted that each approach has their own strength and weaknesses. For instance, differently from SPARQL endpoints and LDFragments, KBox is indicated for scenarios where the knowledge graph is not updated frequently, since the process of publishing the graph requires similar resources as the setup time for SPARQL endpoints (see Table III). However, users and enterprises can use strategies to circumvent this limitation such as splitting their graphs in static and dynamic parts.

VI. CONCLUSION & FUTURE WORKS

We presented Knowledge Box, an approach that transparently shifts the query execution on knowledge graphs to the edge of the network. We show that KBox is a viable alternative for publishing large knowledge graphs providing different (and in many aspects superior) performance characteristics when compared to other data publication design patterns. In particular, Knowledge Box:

- demands less time and resources to setup than traditional approaches;
- pushes the CPU consumption from the server to the client, while being twice as fast as SPARQL endpoints (even when serving a single client), and;
- relies on a decentralized architecture.

In future work, we plan to improve the Knowledge Graph Name resolution on the KNS system by designing more optimized methods. We aim to serve more knowledge graphs and support more frequently changing knowledge graphs by incorporating methods for update propagation. We will also engage the community on distributing their knowledge graphs in a ready-to-go format over the KNS system. We see this work as a preliminary step for increasing the distribution and use of RDF knowledge graphs.

REFERENCES

[1] Alu, G., Hartig, O., zsu, M.T., Daudjee, K.: Diversified Stress Testing of Rdf Data Management Systems. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C.A., Vrandecic, D., Groth, P.T., Noy, N.F., Janowicz, K., Goble, C.A. (eds.) Semantic Web Conference (1). Lecture Notes in Computer Science, vol. 8796, pp. 197–212. Springer (2014), http://dblp.uni-trier.de/db/conf/semweb/iswc2014-1.html#AlucHOD14

[2] Beek, W., Rietveld, L., Bazoobandi, H., Wielemaker, J., Schlobach, S.: LOD Laundromat: A Uniform Way of Publishing Other Peoples Dirty Data. In: ISWC 2014, pp. 213–228. Lecture Notes in Computer Science, Springer International Publishing (2014)

[3] Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International Journal on Semantic Web and Information Systems (IJSWIS) 5(2), 1–24 (2009)

[4] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 1247–1250. ACM (2008)

[5] Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: Sparql web-querying infrastructure: Ready for action? In: International Semantic Web Conference. pp. 277–293. Springer (2013)

[6] Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbusshe, P.Y.: SPARQL Web-Querying Infrastructure: Ready for Action? In: Proceedings of the 12th International Semantic Web Conference, Sydney, Australia (2013)

[7] Erling, O., Mikhailov, I.: Virtuoso: Rdf support in a native rdbms. In: Virgilio, R.D., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management, pp. 501–519. Springer (2009), http://dblp.uni-trier.de/db/books/collections/Virgilio2009.html#ErlingM09

[8] Fernndez, J.D., Martnez-Prieto, M.A., Gutirrez, C., Polleres, A., Arias, M.: Binary RDF Representation for Publication and Exchange (HDT). Web Semantics: Science, Services and Agents on the World Wide Web 19, 22–41 (2013), http://www.websemanticsjournal.org/index.php/ps/article/view/328

[9] Grobe, M.: Rdf, Jena, SparQL and the 'Semantic Web'. In: Farally-Semerad, G., McRitchie, K.J., Rugg, E. (eds.) SIGUCCS. pp. 131–138. ACM (2009), http://dblp.uni-trier.de/db/conf/siguccs/siguccs2009.html#Grobe09

[10] Lee Feigenbaum, Gregory Todd Williams, K.G.C.E.T.: Sparql 1.1 protocol. W3C recommendation, W3C (Mar 2013), https://www.w3.org/TR/sparql11-protocol/

[11] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal 6(2), 167–195 (2015)

[12] Lpez, P.G., Montresor, A., Epema, D.H.J., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M.P., Felber, P., Rivire, E.: Edge-centric Computing: Vision and Challenges. Computer Communication Review 45(5), 37–42 (2015), http://dblp.uni-trier.de/db/journals/ccr/ccr45.html#LopezMEDHIBFR15

[13] Mansour, E., Sambra, A.V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A., Berners-Lee, T.: A Demonstration of the Solid Platform for Social Web Applications. In: Proceedings of the 25th International Conference Companion on World Wide Web. pp. 223–226. International World Wide Web Conferences Steering Committee (2016)

[14] Pampel, H., Vierkant, P., Scholze, F., Bertelmann, R., Kindling, M., Klump, J., Goebelbecker, H.J., Gundlach, J., Schirmbacher, P., Dierolf, U.: Making Research Data Repositories Visible: The re3data.org Registry. PLoS One 8(11), e78080 (Nov 2013), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3817176/

[15] Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. ACM Trans. Database Syst. 34(3), 16:1–16:45 (Sep 2009), http://doi.acm.org/10.1145/1567274.1567278

[16] Saleem, M., Mehmood, Q., Ngonga Ngomo, A.C.: Feasible: A Feature-Based Sparql Benchmark Generation Framework. In: International Semantic Web Conference (ISWC) (2015), http://svn.aksw.org/papers/2015/ISWC_FEASIBLE/public.pdf

[17] Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the web with high availability. In: International Semantic Web Conference. pp. 180–196. Springer (2014)

[18] Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-Scale Querying through Linked Data Fragments. In: Proceedings of the 7th Workshop on Linked Data on the Web (Apr 2014), http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_04.pdf

[19] Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM 57(10), 78–85 (2014)

[20] Yuan, P., Liu, P., Wu, B., Jin, H., Zhang, W., Liu, L.: Triplebit: A Fast and Compact System for Large Scale RDF Data. Proc. VLDB Endow. 6(7), 517–528 (May 2013), http://dx.doi.org/10.14778/2536349.2536352