LSQ: The Linked SPARQL Queries Dataset Technical Report

Muhammad Saleem[1], Intizar Ali[2] Aidan Hogan[3], Qaiser Mehmood[2], and Axel-Cyrille Ngonga Ngomo[1]

[1] Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig
{lastname}@informatik.uni-leipzig.de
[2] Insight Center for Data Analytics, National University of Ireland, Galway
first.lastname@insight-centre.org
[3] Universidad de Chile
aidhog@gmail.com

Abstract. In this report, we present LSQ – a Linked Dataset describing SPARQL queries extracted from the logs of a variety of four prominent public SPARQL endpoints. We argue that this dataset has a variety of uses for the SPARQL research community. For example, it can be used generate benchmarks on the fly by selecting real-world queries with specific characteristics that we describe. LSQ can also be used to conduct analyses of what SPARQL (1.1) query features are most often used to interrogate endpoints. Other use cases include characterising the behaviour of the different types of agents that are using these endpoints or finding out what queries agents are asking about a given resource. In more detail, we first motivate our dataset by reference to a number of concrete use cases, extracting a list of requirements. Next we discuss how we describe SPARQL queries in RDF, using a mix of existing vocabularies (e.g., SPIN) and a custom vocabulary. We present the hosting methods that we provide and remark on issues relating to sustainability. To conclude, we show some high-level examples of the types of conclusions that we can draw from our LSQ about the current state of SPARQL adoption on the Web.

1 Introduction

Although there are now hundreds of public SPARQL endpoints available on the Web – collectively indexing billions of facts and receiving millions of queries each month – it is clear that in terms of SPARQL technology, there is still considerable room for improvement. Many of the aforementioned public endpoints suffer from availability problems and otherwise exhibit non-standard behaviour, such as returning partial results [2]. With respect to the implementations underlying these endpoints, many of the benchmarks proposed to test and decide between them have been found to be too narrow and simplistic [1]; under more fine-grained analysis, the performance of different engines can vary by orders of magnitude without there being any clear winner [1]. Likewise, there are many open questions now being tackled relating to the usability of SPARQL endpoints (e.g., [12,17,6]), or to SPARQL caching (e.g., [22,11]): topics that would appear fundamental when talking about hosting mature, user-centric, high-traffic services on the Web.

SPARQL is a relatively new technology and the recent recommendation of SPARQL 1.1 [8] brings new challenges. Many of the challenges faced –of which some are mentioned above–, could benefit from more information about how users are currently interacting with SPARQL endpoints and in particular, what queries they are sending. Such knowledge may help to focus research on optimising those queries or query features that are most commonly required. For example, real-world queries can be used to generate benchmarks that are more representative of the type of workload a SPARQL engine will have to process in practice [18].[4] Moreover, the analysis of agent interactions could lead to novel approaches to general issues like usability, caching, etc.

Although query logs are available for public SPARQL endpoints through initiatives like USEWOD[5], the datasets involved are only accessible after having signed strict legal agreements, meaning that researchers and other interested parties are limited in their reuse of the service. Likewise, the format of the logs is ad-hoc in nature, depending on their source. Thus, although some interesting surveys of real-world SPARQL queries have been published from, for example, these logs [3,15,16], we still argue for the need for a public, openly accessible dataset containing queries extracted from endpoint logs represented in a single consistent representation.

In this report, we thus introduce the Linked SPARQL Query Log Dataset (LSQ): a public, openly accessible dataset of SPARQL queries extracted from endpoint logs. The current version that we describe in this report consists of 73.2 million triples collected from four query logs, which we gathered directly from the maintainers of the endpoints and for which we have gotten permission to make the logs public. LSQ is available from `http://aksw.github.io/LSQ/`.

The rest of this report is structured as follows: In Section 2, we outline some use cases advocating the potential impact of our dataset and derive some requirements. Section 3 discusses the LSQ dataset schema and gives examples of the RDFisation output. Section 4 introduces the four public SPARQL query logs that we have currently RDFised. Section 5 presents details of the resulting LSQ dataset and analyses of the queries it contains. Section 6 provides some concrete example queries that can be executed over the LSQ dataset in relation to the motivating use case and Section 7 concludes.

2 Motivational Use Cases

In this section, we introduce a number of potential use cases for LSQ to help motivate the potential impact and usage of the dataset. These are the use cases we foresee going forward. Based on these use cases, we extracted a list of milestones to be achieved by LSQ Dataset:

UC1 Custom Benchmarks Recently, there has been a lot interest in the creation of SPARQL benchmarks based on real-world scenarios, be they general-purpose benchmarks [13,4,23] or benchmarks targeted at specific aspects such as federation [20]. The LSQ dataset can be used to generate realistic benchmarks by selecting queries matching ad-hoc desiderata, be it to have broad coverage of the features of SPARQL

[4] `http://feasible.googlecode.com`
[5] `http://usewod.org/`; retr. 2015/04/14.

1.1, or to focus on specific aspects: e.g., researchers focusing on efficient property-path execution may wish to extract those real-world queries featuring use of property paths and basic graph patterns, but no other features.

UC2 SPARQL Adoption A variety of researchers have presented analyses of SPARQL queries in logs [3,15,16], looking at which SPARQL features are most commonly used and how they are used. Likewise, theoretical features of queries – such as othe treewid th of graph patterns or *well-designedness* [14] – can be gauged to see if hypothetical worst-cases occur often in practice. By curating various query logs, our dataset facilitates further analyses of how the SPARQL 1.1 language is being used.

UC3 Caching Works on caching [22,11] try to maximise the space vs. time trade-off by re-using previous results of computation. There is the question not only of what types of data to cache (full query results, intermediate results, etc.), but also which data to cache (which results are more likely to be re-used). By analysing sequences of timestamped queries, researchers in this area can simulate caching options under realistic conditions looking for practical answers to these questions.

UC4 Usability A variety of works are now emerging in the area of the usability of SPARQL endpoints. Works on auto-completion [12,17,6] may benefit from extracting patterns from past queries and using them to guide future suggestions. Works on query relaxation and approximation [5,9,21] could benefit from analysing invalid or empty queries and seeing how agents reformulate their subsequent requests.

UC5 Optimisation Relating to the previous use cases, statistics collected from queries can help to fine-tune the parameters of query engines based on patterns in real world usage, for example, the likelihood of picking a certain join algorithm, how much memory to assign to specific query processing sub-tasks.

UC6 Meta-Querying The previous five use cases have been "internal" in the sense of helping to improve or understand things relating to SPARQL. However, as a more general (but admittedly speculative) use case, LSQ can support what we call "meta-querying": querying for queries. For example, from LSQ, one could find out what are the queries that people are asking about a resource of interest, be it a particular city, a conference, a product, the person themselves, etc.

This list of use cases is of course incomplete and indeed one could imagine further use cases that are not mentioned herein. Still, from the use cases above, we can derive a list of facets of queries and their execution that LSQ Dataset should capture:

F1 Query Features The dataset should describe the features used in individual queries (`ASK`, `OPTIONAL`, etc.) in such a manner that, e.g., queries can be extracted according to the features they use/omit (**UC1**), or the number of queries using a feature can be determined (**UC2**). Such tasks should be possible by querying the meta-data rather than needing to scan and parse the syntax of all SPARQL queries.

F2 Query Statistics The query meta-data should likewise capture high-level information about the size and "complexity" of the queries in terms of, e.g., number of triple patterns, join variables within a single query, etc., allowing, e.g., overall trends to be analysed (**UC1**, **UC5**) or queries of a certain size to be extracted (**UC2**).

F3 Queried Resources The dataset should list the constants explicitly mentioned by a given query, be they subject IRIs (**UC6**) specific predicates (**UC1**), etc.

F4 Execution Details The dataset should provide information about the execution of each query, including a timestamp (**UC3–4**, **UC6**), the endpoint it was issued to (**UC1–6**), and an anonymised identifier for the issuing client (**UC2**, **UC4**).

F5 Query Results Although the full query results would be far too large to include, the dataset should include high-level information about whether the query incurred an error (**UC4**) and if not, whether it returned results (**UC4**) and how many (**UC1–2**).

These facets circumscribe the target scope of our LSQ Dataset. However, due to practical issues – such as the prohibitive size of potential output data or a given log not containing the pertinent information – we may not be able to capture all the data for all facets across all logs. Such issues will be discussed in depth later.

3 RDF Data Model

Our goal is to create a Linked Dataset describing the SPARQL queries issued to various public SPARQL endpoints. In this section, we describe in detail the RDF data model we employ to capture the facets mentioned in the previous section. In the design of this data model, we identified a number of desiderata, as follows:

D1 Generality The data model should broadly cover the aforementioned facets. The resulting dataset should allow for use cases to be satisfied over the meta-data alone and without needing to parse the raw query text.

D2 Conciseness Since out logs contain millions of queries, the resulting data model should be concise to keep the overall dataset size manageable.

D3 Usability Basic competency questions over the dataset (e.g., get all queries with a given feature) should translate into efficient queries.

D4 Linked Data Compatibility URIs should be made dereferenceable per Linked Data Principles. Terms from external well-known vocabularies should be re-used where appropriate. Links to other datasets should be provided.

However, some of these desiderata are antagonistic. In particular, the need for conciseness conflicts with the need for generality and usability. In the data model, we thus need to strike a balance. In Figure 1, we provide an overview of the core of the schema for the LSQ data-model. Listing 1 provides an example output for a query. The main aspects of the dataset are now detailed.

Query instance Queries in the data are typed as `sq:Query`. As a practical design decision, we create query instances for each log whereby a query is linked to a single endpoint from whose log it was extracted. Hence if the same query with the same syntax is issued to the same endpoint multiple times, it is represented with a single instance of `sq:Query`. However, if queries with the same syntax are issued to different endpoints, there would instances of `sq:Query` for each endpoint.[6] Hence, instances of this class

[6] The main reason for this decision is to satisfy conciseness: if we considered query resources that were independent of an endpoint, we would need to create n-ary predicates for all of the endpoint-specific data about a query.

are linked to the query text (using `sd:text`) and to the originating endpoint (using `lsqv:endpoint`).[7]

Executions Query instances are linked to an instance of the class `lsqv:Execution` for each time the query was run against that endpoint. Each such instance provides a time (`dct:issued`) and a crytographically-hashed and salted I.P. to identify which queries are run by the same agent (`lsqv:agentId`).

SPIN Representation To help meet the generality requirement, we attach a complete SPIN representation of the query to each query instance (for brevity, other than high-level details such as the subclasses of `sp:Query`, the full SPIN model is not shown in the schema diagram; we instead refer the reader to [10]). The SPIN representation provides a complete description of the query in RDF, allowing for arbitrarily-detailed queries to be written against the LSQ dataset.

Shortcut Triples Given that the SPIN representation may involve an arbitrary level of nesting using a variety of predicates, to meet the usability requirement, we provide shortcut triples to indicate the SPARQL query features used in the query. These triples link query instances (with the predicate `lsqv:usesFeature`) to instances of `sd:Feature`. We enumerate a comprehensive list of such feature instances in our vocabulary, including `lsqv:Filter`, `lsqv:Optional`, `lsqv:SubQuery`, etc. These shortcut triples greatly simplify the process of finding queries in LSQ based on the features they (do not) use.

Structural Statistics We also provide generic *structural statistics* [1] about the static query, as follows:

- The number of Basic Graph Patterns (`lsqv:bgps`)
- The number of triple patterns (`lsqv:triplePatterns`)
- The number of join vertices/variables (`lsqv:joinVertices`)
- The mean join vertex degree (i.e., the mean number of triple patterns containing a join variable; `lsqv:meanJoinVertexDegree`)
- The join vertex type (i.e., star, path, hybrid, and sink [19], which will be discussed later; `lsqv:joinVertexType`)

Data-driven Statistics Next we provide *data-driven statistics* [1] about the execution of the query. Since such data are not typically provided by the logs, we generate these statistics by running the query locally against an offline copy of the dataset in question. This generation was done on a machine with 16 GB RAM and a 6-Core i7 3.40 GHz CPU running Ubuntu 14.04.2 using Virtuoso 7.1 with `NumberOfBuffers = 1360000`, and `MaxDirtyBuffers = 1000000`. Of course, the resulting statistics may differ to those that occurred during the original execution logged by the public endpoint; rather they are intended to serve as a guide for, e.g., selecting benchmark queries, etc. These statistics are:

[7] Although there are some existing predicates with the label endpoint, such as `void:endpoint` or `sd:endpoint`, the domain of these predicates were not queries but, e.g., datasets or services. In general, we were careful to avoid inappropriate term re-use.

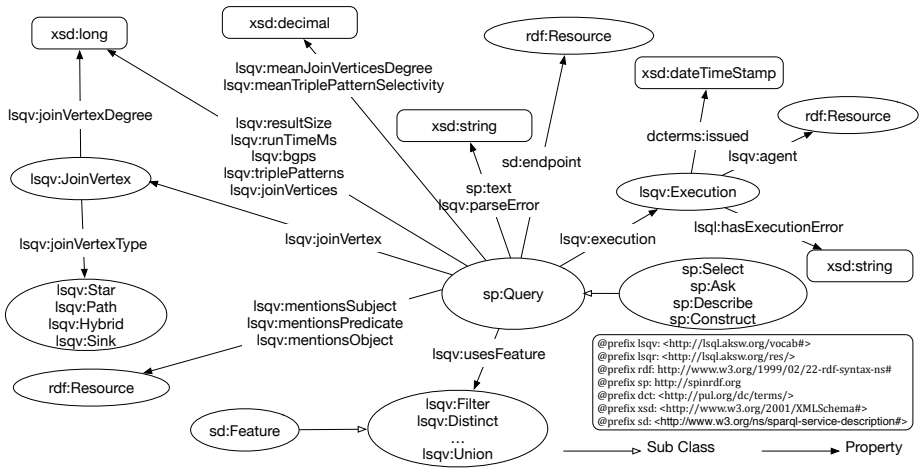

Fig. 1: First Draft of the Dataset Schema

- Number of results returned (`lsqv:resultSize`)
- Runtime of the query (`lsqv:runTimeMs`)
- Mean selectivity of triple patterns (`lsqv:meanTriplePatternSelectivity`)

Minimality To help meet the conciseness requirement, we drop redundant type triples. For example, query instances are typed once with their most specific SPIN class (`sp:Ask`, `sp:Construct`, `sp:Describe`, `sp:Select`) but the generic type triple for `sp:Query` omitted. Likewise, we do not explicitly type instances with `lsqv:Execution`. These omissions do not affect the other desiderata.

Summary The above data model is designed to cover the facets mentioned in Section 2 while striking what we argue to be a good compromise between the goals of generality, conciseness and usability. Of course, given aspects such as the SPIN representation of the query – included to increase the generality of the dataset – the goal of conciseness may be threatened. We will discuss this trade-off later in the context of the final dataset size(s).

With respect to Linked Data Compatibility, all query instances and executions are identified with dereferenceable URIs. Our data model also re-uses class and property terms from established external vocabularies, including SPIN, DC Terms and SPARQL Service Descriptions. Finally, with respect to external links, LSQ provides links to every URI mentioned in a query.

4 Current Query Logs

In this section, we provide a brief overview of the query logs we have collected and RDFised in the context of the LSQ dataset. We contacted a number of administrators of

Listing 1: An example LSQ representation of an SWDF query

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix lsqr: <http://lsq.aksw.org/res/> .
@prefix lsqrd: <http://lsq.aksw.org/res/SWDF-> .
@prefix lsqv: <http://lsq.aksw.org/vocab#> .
@prefix sp: <http://spinrdf.org/sp#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# QUERY INSTANCE META-DATA
lsqrd:q483 lsqv:endpoint <http://data.semanticweb.org/sparql> ;
 sp:text """SELECT DISTINCT ?prop
 WHERE {
       ?obj rdf:type swdf:SessionEvent .
       ?obj ?prop ?targetObj .
       FILTER (isLiteral(?targetObj)) }
       LIMIT 150""" .

# STRUCTURAL META-DATA
lsqrd:q483 lsqv:bgps 1 ; lsqv:triplePatterns 2 ; lsqv:joinVertices 1 ;
 lsqv:meanJoinVerticesDegree 2.0 ;
 lsqv:usesFeature lsqv:Filter , lsqv:Distinct , lsqv:Limit ;
 lsqv:mentionsSubject "?obj" ;
 lsqv:mentionsPredicate "?prop" , rdf:type ;
 lsqv:mentionsObject "?targetObj" , swdf:SessionEvent ;
 lsqv:joinVertex lsqr:q483-obj .
lsqr:q483-obj lsqv:joinVertexDegree 2 ; lsqv:joinVertexType lsqv:Star .

# DATA-SENSITIVE META-DATA
lsqrd:q483 lsqv:resultSize 16 ; lsqv:runTimeMs 6 ;
 lsqv:meanTriplePatternSelectivity 0.5007155695730322 ;

# QUERY EXECUTION META-DATA
lsqrd:q483 lsqv:execution lsqrd:q483-e1 , lsqrd:q483-e2 , lsqrd:q483-e3 , lsqrd:
    q483-e4 .
lsqrd:q483-e1 lsqv:agent lsqr:A-WlxKE0QQRlhCUBdGRx1QGVRbQRNsN2YUWF5W ;
 dct:issued "2014-05-22T17:08:17+01:00"^^xsd:dateTimeStamp .
lsqrd:q483-e2 lsqv:agent lsqr:A-WlxKE0QQRlhCUBdGRx1QGVRdRBNsN2YUW1pS ;
 dct:issued "2014-05-20T14:34:35+01:00"^^xsd:dateTimeStamp .
lsqrd:q483-e3 lsqv:agent lsqr:A-WlxKE0QQRlhCUBdGRx1QGVRdRBNsN2YUW1pS ;
 dct:issued "2014-05-20T14:28:37+01:00"^^xsd:dateTimeStamp .
lsqrd:q483-e4 lsqv:agent lsqr:A-WlxKE0QQRlhCUBdGRx1QGVRdRBNsN2YUW1pS ;
 dct:issued "2014-05-20T14:24:13+01:00"^^xsd:dateTimeStamp .

# SPIN REPRESENTATION
lsqrd:q483 a sp:Select ;
 sp:distinct true ; sp:limit "150"^^xsd:long ;
 sp:resultVariables ( [ sp:varName "prop"^^xsd:string ] ) ;
 sp:where (
   [ sp:subject [ sp:varName "obj"^^xsd:string ] ;
     sp:predicate rdf:type ;
     sp:object <http://data.semanticweb.org/ns/swc/ontology#SessionEvent>
   ]
   [ sp:subject [ sp:varName "obj"^^xsd:string ] ;
     sp:predicate [ sp:varName "prop"^^xsd:string ] ;
     sp:object [ sp:varName "targetObj"^^xsd:string ]
   ]
   [ a sp:Filter ;
     sp:expression [ a sp:isLiteral ; sp:arg1 [ sp:varName "targetObj"^^xsd:string
          ] ]
   ]
) .
```

widely-used public SPARQL endpoints.[8] Unfortunately, a number of administrators were unable to provide their logs or to give permission to make their logs publicly available. At this point in time, we have collected and RDFised four query logs from four popular SPARQL endpoints, which we now introduce.[9]

DBpedia is a Linked Dataset that is extracted from Wikipedia with crowd-sourced community efforts. The dataset is exposed as SPARQL endpoint at `http://dbpedia.org/sparql` through a Virtuoso instance. The DBpedia query log we have currently obtained spans from April 30, 2010 to July 20, 2010 (these queries refer to DBpedia v.3.5.1). The log records over 1.7 million query executions.

Linked Geo Data contains a collection of spatial Linked Datasets that have been extracted from Open Street Map. The data are accessible via a public SPARQL endpoint at `http://linkedgeodata.org/sparql` through a Virtuoso instance. The Linked Geo Data (LGD) query log spans from November 24, 2010 to July 6, 2011. The log records over 1.6 million query executions.

Semantic Web Dog Food: SWDF is an effort to generate a Linked Dataset about papers, presentations and people participating in top Semantic Web related conferences and workshops. The dataset is typically accessible through a SPARQL endpoint at `http://data.semanticweb.org/sparql` through a Sesame interface.[10] The Semantic Web Dog Food (SWDF) log spans from May 16, 2014 to November 12, 2014 and records over 1.4 million query executions.

British Museum provides a Linked Data representation of an online collection containing records of more than 3 million artefacts. The British Museum (BM) Linked Dataset is accessible at `http://collection.britishmuseum.org/sparql` through an OWLIM/GraphDB interface. The log we have acquired spans from November 8, 2014 to December 1, 2014 and contains over 800 thousand query executions.

Overview of datasets In Table 1, we present some high-level statistics of the *original* Linked Datasets corresponding to the time of the logs (e.g., DBpedia refers to DBpedia v.3.5.1), including the number of distinct triples, subjects, predicates, objects and classes over which queries would have been issued. The statistics were collected by downloading and locally analysing the data. These datasets will be used later to generate controlled estimates of data-sensitive statistics for queries, such as runtimes, result sizes, triple pattern selectivity, etc.

[8] We acknowledge Dimitris Kontokostas, Jens Lehmann, Richard Cyganiak, and Hugh Glaser for the provision of the query logs of DBpedia, Liked Geo Data, Semantic Web Dog Food and British Museums respectively.

[9] We are currently in the process of RDFising two more logs (from BioPortal and Strabon), which we hope to make available in the coming weeks; likewise, we are in the process of obtaining and RDFising extended logs from, e.g., DBpedia.

[10] Unfortunately, at the time of writing, this SPARQL endpoint was not available.

Table 1: Basic statistics of the original datasets over which queries from the logs were executed

DATASET	TRIPLES	SUBJECTS	PREDICATES	OBJECTS	CLASSES
DBpedia	232,536,510	18,425,128	39,672	65,184,191	244
LGD	1,032,026,569	238,509,864	30,882	492,282,120	1,113
SWDF	294,870	30,856	185	93,051	126
BM	1,359,400	483,877	27	684,733	1

5 Linked SPARQL Query Dataset Statistics

We applied the RDFisation process to the four logs mentioned in the previous section. Given that the logs were in different formats, we created custom scripts to extract and normalise data from the four different sources, mapping them to the target schema outlined in Section 3. We now give a more in-depth analysis of the resulting datasets, as well as an analysis of the unique queries, query executions and agents mentioned therein. Our goal is to provide insights into the scope and usefulness of the dataset, as well as its limitations.

5.1 Query Analysis

Table 2 provides a high-level analysis of the queries (both query executions and unique queries) appearing in each of the four logs. While the majority of queries are `SELECT` (91.6% overall), SWDF contains a large number of `DESCRIBE` queries (31.1%). BM contains a noticeably high ratio of parse errors (77.63%), compared with DBpedia (35.27%), SWDF (13.75%), or LGD (4.35%). Conversely, while LGD is the lowest in terms of parse errors, it generates the highest ratio of runtime errors (16.08%), followed by DBpedia (5.54%), SWDF (0.05%), and BM (0%); one likely explanation is that runtime errors – which include timeouts – are more frequent against larger datasets; likewise, as we will see later, the vast majority of BM queries we found were quite simple and uniform, with no joins, and hence are less likely to cause problems during execution.[11]

Table 3 shows the distribution of queries with respect to different triple-pattern-level join types as we defined previously in [19]. The idea is to count individual join variables within a Basic Graph Pattern as individual joins and type them depending on how they connect triple patterns. We say that a join vertex has an "outgoing link" if it appears as a subject of a triple pattern, and that it has an "incoming link" if it appears as predicate or object. The types are then as follows:

STAR has multiple outgoing links but no incoming links.
PATH has precisely one incoming and one outgoing link.

[11] We suspect that the uniformity of BM queries may be due to one agent asking a high volume of simple queries to the endpoint; unfortunately the BM log did not include agent data, so we cannot confirm nor refute this possibility.

Table 2: High-level analysis of the queries and query executions in the LSQ dataset for each log (QE = Query Executions, UQ = Unique Queries, PE = Parse Errors, RE = Runtime Error, ZR = Zero Results, SEL = SELECT, CON = CONSTRUCT, DES = DESCRIBE; percentages are with respect to UQ)

DATASET	QE	UQ	PE	RE	ZR	SEL	CON	DES	ASK
	№	№	№	№	№	%	%	%	%
DBpedia	1,728,041	1,208,789	426,425	69,523	176,257	94.6	0.9	0.1	4.4
LGD	1,656,254	311,126	13,546	50,059	143,574	89.3	2.3	0.0	8.4
SWDF	1,411,483	99,165	13,645	475	25,674	68.8	0.0	31.1	0.1
BM	879,426	129,989	100,916	0	29,073	100	0.0	0.0	0.0
Overall	5,675,204	1,749,069	554,532	120,057	374,578	91.6	1.2	2.3	4.9

Table 3: Percentage of unique queries containing different types of joins (a query may contain multiple join types)

DATASET	STAR	PATH	HYBRID	SINK	NO JOIN
	%	%	%	%	%
DBpedia	38.58	8.60	6.79	6.31	61.23
LGD	28.18	9.46	7.57	1.24	72.00
SWDF	10.70	11.25	4.01	0.93	84.25
BM	0.00	0.00	0.00	0.00	100.00
Overall	33.05	8.79	6.62	4.51	66.51

HYBRID has at least one incoming and outgoing link and three or more links.
SINK has multiple incoming links but no outgoing links.

From Table 3, we see that most of the queries are either STAR (subject–subject join; 33.1%) or contain no join (66.5%). As mentioned before, in the case of the British Museum, none of the queries contained a join. However, we note that 37% of queries contain a single triple pattern: the other queries without joins contain multiple BGPs or, e.g., are `DESCRIBE` queries looking a single URI without a `WHERE` clause (see SWDF in Table 2), etc.

Table 4 shows the mean values for various query features across all query logs. These features are important, for example, when selecting queries for inclusion in SPARQL benchmarks [1,7]. The SWDF queries are generally more complex, on average, in terms of the number of BGPs and total number of triple patterns. However, they contain fewer joins among triple patterns and the join vertex degree is also on a lower side (e.g., 0.35 for SWDF vs. 0.78 for DBpedia). LGD and DBpedia queries exhibit, on average, both the largest number of results and the longest runtimes; this is likely due to the associated datasets being much larger than either SWDF or BM. With respect to BM, we see that the queries are not only uniform (all containing a single triple pattern), but they do not return results over the dataset; this again suggests that the British Museum log contains a high volume of simple, synthetic queries.

Table 4: Comparison of the mean values of different query features across all query logs (RS = Result Size, TPs = Triple Patterns, JVs = Join Vertices, MJVD = Mean Join Vertex Degree, MTPS = Mean Triple Pattern Selectivity)

Dataset	RS	BGPs	TPs	JVs	MJVD	MTPS	Runtime (ms)
DBpedia	87.57	1.81	2.22	0.40	0.78	0.002	20.26
LGD	161.90	1.75	2.16	0.37	0.75	0.030	32.28
SWDF	19.65	2.57	2.94	0.26	0.35	0.025	11.98
BM	0.00	1.00	1.00	0.00	0.00	0.000	6.78
Overall	122.45	1.74	2.04	0.24	0.45	0.013	26.40

Table 5: Percentage of queries using various specific SPARQL features

Dataset	UNION	OPTIONAL	DISTINCT	FILTER	REGEX	SERVICE	Sub-Query
DBpedia	4.42	36.20	18.44	23.47	2.90	0.0005	0.00
LGD	9.65	25.10	22.25	31.10	1.25	0.0000	0.01
SWDF	32.71	25.32	45.40	0.95	0.06	0.0012	0.02
BM	0.00	0.00	100.00	0.00	0.00	0.0000	0.00
Overall	7.64	31.78	23.30	23.19	2.22	0.0004	0.01

Table 6: Percentage of queries using at least one feature from a given categorisation (Solution Mod. includes the solution modifiers ORDER BY, OFFSET, and LIMIT; Aggregates include GROUP BY, HAVING, AVG, SUM, COUNT, MAX, and MIN; Negation include MINUS, NOT EXISTS, and EXISTS; Binding includes VALUES and BINDING; Graph includes FROM, FROM NAMED, and GRAPH)

Dataset	Solution Mod.	Aggregates	Negation	Binding	Graph
DBpedia	1.036	0.001	0.001	0.000	0.002
LGD	60.443	0.007	0.000	0.000	0.000
SWDF	33.265	2.405	0.001	0.008	0.001
BM	0.000	0.000	0.000	0.000	0.000
Overall	18.117	0.174	0.001	0.001	0.001

Tables 5 and 6 show the percentage use of groups of different SPARQL features; a query is counted in a given group if it uses one of the associated features. In general, we found that the SPARQL 1.1 features are rarely used; however, in the case of DBpedia and LGD, this may be due to the age of the logs. The most widely used feature is `OPTIONAL` (31.78%), followed by `DISTINCT` (23.3%) and `FILTER` (23.19%). Solution modifiers (i.e., `LIMIT`, `OFFSET`, `ORDER BY`) are also quite often used (18.11%).[12]

[12] Indeed, similar observations were made previously by Arias et. al [3].

Table 7: Statistics about the agents executing queries

Dataset	Agents	Exec/Agent		Max. Agent	
	№	μ	σ	№	%
DBpedia	3,041	568.2	6,343.0	266,836	15.5
LGD	725	2,284.5	19,437.4	286,677	17.3
SWDF	274	5,151.3	80,916.2	1,341,874	95.1

5.2 Execution and Agent Analysis

Thus far we have analysed statistics of unique queries. In this section, we are interested in looking at (a) whether the same queries tend to be executed many times and (b) whether or not some agents tend to be responsible for many of the query executions.

With respect to the number of times a given query is executed, if we take the total number of query executions (5,675,204) and the total number of unique queries (1,749,069) from Table 2, we can see that a given query is executed on average about 3.2 times in the scope of the logs defined. To compare this distribution for the four logs, Figure 2 provides a Lorenz curve, which shows what (maximal) ratio of unique queries account for what (minimal) ratio of query executions. For example, we see that for SWDF, 80% of the unique queries account for about 10% of the overall executions, or equivalently that the top 20% most frequently executed queries account for 90% of all executions. On the other hand, the executions for DBpedia are much more evenly spread. For LGD, the sharp ascent of the curve suggests that a handful of unique queries account for the majority of executions.

With respect to the distribution of executions amongst agents, Table 7 provides high-level statistics for DBpedia, LGD and SWDF (we do not have agent information in the logs for the British Museum). We see that the number of unique agents ranges from 274 in the case of SWDF to 3,041 in the case of DBpedia. We also see that, on average, each agent is responsible for hundreds or thousands of query executions; however, we can see that this mean value is associated with a high standard deviation, particularly in the case of SWDF, which implies that a few (presumably automated) agents are responsible for a great many queries. This can likewise be seen in the number of queries issued by agent with the most query executions, who is responsible for 15.5–17.3% in DBpedia and LGD respectively, but a notable 95.1% in the case of SWDF. To better illustrate this, Figure 3 presents the relevant Lorenz curve, in which we can see a heavy skew; for example, 90% of the agents with fewest executions are cumulatively responsible for fewer than 3% of the total executions (2.7% for DBpedia, 0.7% for LGD, and 0.2% for SWDF). From this curve, we posit that the vast majority of queries encountered in these logs are from a handful of high-volume, automated agents; this should be taken into account by users of the LSQ dataset.

6 The LSQ Dataset in Practice

We have made the LSQ dataset available through three media: (i) dereferenceable Linked Data, (ii) flat dumps and (iii) a SPARQL endpoint. In this section, we wish to provide

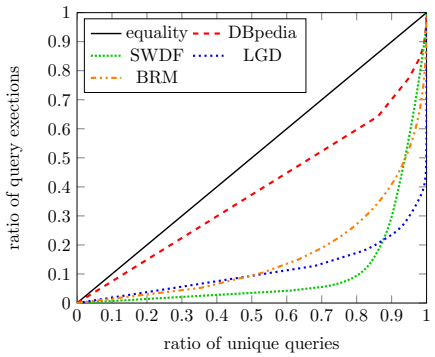

Fig. 2: Lorenz curve for distribution of executions per unique query

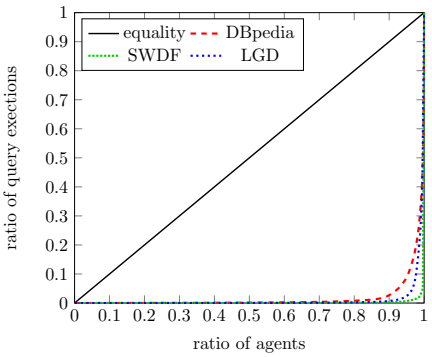

Fig. 3: Lorenz curve for distribution of executions per unique agent

some concrete queries that can be issued against the LSQ SPARQL endpoint in order to derive insights relevant to each of the use cases discussed in Section 2.

UC1 Facilitating Benchmark Generation: LSQ can help users generate custom benchmarks by selecting real-world queries meeting certain criteria. Listing 2 is an example SPARQL query over LSQ that provides a list of 50 queries with additional parameters set for both structural and data-driven criteria useful for creating custom benchmarks [1].

Listing 2: UC1: Collect 50 benchmark queries having more than 10 results, fewer than 5 triple patterns, using filters, with an execution time below 50 ms

```
PREFIX lsqv: <http://lsq.aksw.org/vocab#>
PREFIX sp: <http://spinrdf.org/sp#>
SELECT ?query FROM <http://data.semanticweb.org>
 WHERE {
   ?id sp:text ?query ; lsqv:resultSize ?rs ; lsqv:triplePatterns ?tp ;
      lsqv:runTimeMs ?rt ; lsqv:usesFeature lsqv:Filter .
  FILTER (?rs > 10 && ?tp <5 && ?rt < 50 ) }
LIMIT 50
```

UC2 SPARQL Adoption The Linked SQ dataset can also be used to gain insights into how the SPARQL query language is being used in practice, be that to find out how features are used and combined (Listing 3) or to see, for example, what kinds of joins are most common (Listing 6)

Listing 3: UC2: The number of queries using both `UNION` and `FILTER`

```
PREFIX lsqv: <http://lsq.aksw.org/vocab#>
SELECT COUNT(?queryId) AS ?unionFilterCount
WHERE { ?queryId lsqv:usesFeature lsqv:Union , lsqv:Filter . }
```

UC3 Caching The Linked SQ dataset can also be used to find useful patterns to cache, commonly repeated queries, or to create realistic caching benchmarks using the

timestamp of execution times. Listing 4 gives an example of an LSQ query that finds the most frequently executed queries that take a long time to compute but have small result sizes that can be cheaply cached.

Listing 4: UC3: Get top query executions for smaller result set queries.

```
PREFIX lsqv:<http://lsq.aksw.org/vocab#>
PREFIX sp:<http://spinrdf.org/sp#>
SELECT DISTINCT ?query COUNT(?exs) AS ?exsCount
 WHERE {
   ?id sp:text ?query ; lsqv:resultSize ?rs ; lsqv:execution ?exs ; lsqv:runTimeMs
        ?rt .
   FILTER (?rs < 100 && ?rt > 10000)}
  GROUP BY ?query ORDER BY DESC(COUNT(?exsCount))
```

UC4 Usability From the Linked SQ Dataset, one can derive a list of queries that resulted in parse errors, runtime errors, or empty results. One can also look at which agents issued such queries, and how their queries evolved over time. Listing 5 gives a small example of a query looking for parse errors encountered by a given agent, ordered by time.

Listing 5: UC4: Which queries resulted in a parse error for a given agent?

```
PREFIX lsqv: <http:// lsq .aksw.org/vocab#>
PREFIX lsqr: <http:// lsq .aksw.org/ res/>
PREFIX sp: <http:// spinrdf .org/sp#>
PREFIX dct: <http:// purl .org/dc/terms/>
SELECT ?query ?time ?error
 WHERE {
    ?id  sp: text  ?query  ;  lsqv : parseError  ? error  ;  lsqv : execution  ?ex  .
    ?ex  dct : issued  ?time  ;
         lsqv : agent  lsqr : A-WlFJE0QQRlhBVRNGRx1QGVdaRhNsN2YUW15R .
 }
ORDER BY ?time
```

UC5 Optimisation Given a particular workload of queries, an optimiser can decide how to configure indexes, etc., to improve the performance of typical queries. Administrators can use LSQ to derive some default statistics for what is most common across different databases. For example, Listing 6 provides a query to see how frequently queries containing paths return zero results, which may motivate optimisations to pre-filter empty paths; one could consider a similar example to find path queries that take the longest time, which may suggest to materialise indexes for specific paths.

Listing 6: U5: The number of empty-result queries with path joins

```
PREFIX lsqv: <http://lsq.aksw.org/vocab#>
SELECT COUNT(?id) AS ?starQueries
WHERE {
  ?id lsqv:joinVertex ?joinVertex ; lsqv:resultSize 0 .
  ?joinVertex lsqv:joinVertexType lsqv:Path . }
```

UC6 Meta-querying The final example query in Listing 7 shows how one can find all the queries relating to a given resource, in this case Michael Jackson.

Listing 7: UC6: Fetch all queries asking about Michael Jackson

```
PREFIX sp:<http://spinrdf.org/sp#>
SELECT DISTINCT ?query
 WHERE {
   ?id sp:text ?query .
   { ?id lsqv:mentionsSubject <http://dbpedia.org/ontology/Michael_Jackson> }
   UNION
   { ?id lsqv:mentionsObject <http://dbpedia.org/ontology/Michael_Jackson> }
 }
```

7 Conclusions and Future Directions

In this report we presented LSQ, the first, to the best of our knowledge, Linked Dataset describing SPARQL queries. We showed a variety of use cases where LSQ can be helpful. We provided various statistics about the static and runtime features of SPARQL queries collected from four well-known SPARQL endpoints. It was shown that most of the queries are simple, i.e., few triple patterns, few triple pattern joins, and small result sizes and runtimes. Furthermore, we have looked in to the agent analysis and concluded that most of SPARQL queries are generated by automated agents. Finally, we provided various SPARQL queries to collect the statistics related to the discussed use cases. As future work, we are collecting logs from other SPARQL endpoints (e.g., Bioportal, Strabon) and will be added into LSQ. Furthermore, we will provide an API access where user can submit their queries.

References

1. G. Aluç, O. Hartig, M. T. Ozsu, and K. Daudjee. Diversified stress testing of rdf data management systems. In *ISWC*, 2014.
2. C. B. Aranda, A. Hogan, J. Umbrich, and P. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *ISWC*, pages 277–293, 2013.
3. M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world SPARQL queries. *CoRR*, 2011.
4. C. Bizer and A. Schultz. The berlin sparql benchmark. *IJSWIS*, 2009.
5. A. Calì, R. Frosini, A. Poulovassilis, and P. T. Wood. Flexible querying for SPARQL. In *OTM*, pages 473–490, 2014.
6. S. Campinas. Live SPARQL auto-completion. In *ISWC Posters & Demos*, pages 477–480, 2014.
7. O. Görlitz, M. Thimm, and S. Staab. Splodge: Systematic generation of sparql benchmark queries for linked open data. In *ISWC*. 2012.
8. S. Harris, A. Seaborne, and E. Prud'hommeaux, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013. Available at `http://www.w3.org/TR/sparql11-query/`.
9. A. Hogan, M. Mellotte, G. Powell, and D. Stampouli. Towards fuzzy query-relaxation for RDF. In *ESWC*, pages 687–702, 2012.
10. H. Knublauch, J. A. Hendler, and K. Idehen, editors. *SPIN – Overview and Motivation*. W3C Member Submission, 22 February 2011. Available at `http://www.w3.org/Submission/spin-overview/`.

11. T. Lampo, M. Vidal, J. Danilow, and E. Ruckhaus. To cache or not to cache: The effects of warming cache in complex SPARQL queries. In *OTM*, pages 716–733, 2011.
12. J. Lehmann and L. Bühmann. Autosparql: Let users query your knowledge base. In *ESWC*, pages 63–79, 2011.
13. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. Dbpedia sparql benchmark - performance assessment with real queries on real data. In *ISWC*, 2011.
14. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. In *ISWC*, pages 30–43, 2006.
15. F. Picalausa and S. Vansummeren. What are real sparql queries like? In *SWIM*, 2011.
16. L. Rietveld and R. Hoekstra. Man vs. machine: Differences in sparql queries. In *USEWOD*, 2014.
17. L. Rietveld and R. Hoekstra. YASGUI: feeling the pulse of linked data. In *EKAW*, pages 441–452, 2014.
18. M. Saleem, Q. Mehmood, and A.-C. Ngonga Ngomo. FEASIBLE: A featured-based sparql benchmark generation framework. In *ISWC*, 2015.
19. M. Saleem and A.-C. Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In *ESWC*, 2014.
20. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *ISWC*, 2011.
21. R. D. Virgilio, A. Maccioni, and R. Torlone. A similarity measure for approximate querying over RDF data. In *EDBT/ICDT*, pages 205–213, 2013.
22. G. T. Williams and J. Weaver. Enabling fine-grained HTTP caching of SPARQL query results. In *ISWC*, pages 762–777, 2011.
23. H. Wu, T. Fujiwara, Y. Yamamoto, J. Bolleman, and A. Yamaguchi. Biobenchmark toyama 2012: an evaluation of the performance of triple stores on biological data. *JBMS*, 2014.