LSQ: The Linked SPARQL Queries Dataset

Muhammad Saleem[1], Muhammad 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
[2] Insight Center for Data Analytics, National University of Ireland, Galway
[3] Department of Computer Science, Universidad de Chile

Abstract. We present LSQ: a Linked Dataset describing SPARQL queries extracted from the logs of public SPARQL endpoints. We argue that LSQ has a variety of uses for the SPARQL research community, be it for example to generate custom benchmarks or conduct analyses of SPARQL adoption. We introduce the LSQ data model used to describe SPARQL query executions as RDF. We then provide details on the four SPARQL endpoint logs that we have RDFised thus far. The resulting dataset contains 73 million triples describing 5.7 million query executions.

1 Introduction

Although there are now hundreds of public SPARQL endpoints available on the Web – collectively exposing billions of facts and receiving millions of queries per month – current works suggest that in terms of SPARQL technology, there is still considerable room for improvement [9,2,1]. Many of these endpoints suffer from availability and performance issues [2]. In addition, the recent recommendation of SPARQL 1.1 [6] brings new challenges. Tackling these challenges could benefit from more data about how users are currently interacting with SPARQL endpoints and which queries they are sending. Such knowledge may help to focus research on optimising those queries or query features that are most often used.

Although query logs are available for public SPARQL endpoints through initiatives like USEWOD [4], the datasets are only accessible after having signed legal agreements, which limits re-use. Likewise, the format of the raw logs is ad-hoc in nature, depending on their source. We thus introduce the Linked SPARQL Queries Dataset (LSQ): a public, openly accessible dataset of SPARQL queries extracted from endpoint logs.[4] The current version that we describe in this paper consists of 73.2 million triples collected from four query logs, which we have gathered from the maintainers of public endpoints and for which we have gotten permission to make the logs public. We foresee a number of potential use cases for such a dataset:

UC1 Custom Benchmarks The LSQ dataset can be used to generate realistic benchmarks by selecting queries matching ad-hoc desiderata [12].

[4] The LSQ dataset is available from `http://aksw.github.io/LSQ/`.

UC2 SPARQL Adoption The data can be used by researchers to conduct analyses of features used in real-world SPARQL queries [3,10,11].

UC3 Caching Works on caching [14,8] could benefit from a dataset of real-world queries by, e.g., analysing real-world sequences of queries.

UC4 Usability Analysis of user behaviour – e.g., errors made, how they refine queries, etc. – could guide the design of better interfaces.

UC5 Meta-Querying One could find out what are the queries that people are asking about a resource of interest, be it a product, person, city, etc.

These use cases not only require details about queries, but also query executions, agents, result sizes, etc. We now describe the LSQ data model, whose goal is to comprehensively capture all such aspects of query logs.

2 RDF Data Model

Our goal is to create a Linked Dataset describing the SPARQL queries issued to various public SPARQL endpoints. In Figure 1, we provide an overview of the core of the schema for the LSQ data-model. Listing 1 provides a comprehensive example output for a query. The main aspects of the dataset are now presented.[5]

Queries in the data are typed as a subclass of `sp:Query` (e.g., `sp:Select`, `sp:Ask`, etc.). 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 `sp:Query`, linked to multiple instances of `lsqv:Execution` for each time the query was run. Each such execution instance provides a time (`dct:issued`) and a unique agent IRI computed from a cryptographically-hashed and salted I.P. address (`lsqv:agent`).

To help make the dataset as general as possible, we attach a complete SPIN representation of the query to each query instance [7]. Given that the SPIN representation may involve an arbitrary level of nesting using a variety of predicates, to make querying LSQ more convenient and efficient, 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. We also provide shortcuts to the IRIs and literals mentioned in a query so consumers can easily find all queries about a given resource.

In addition to the query structure, we also provide generic *structural statistics* [1] about the static query including the number of Basic Graph Patterns (`lsqv:bgps`) and the number of triple patterns (`lsqv:triplePatterns`). We also provide *data-driven statistics* [1] (incl. the number of results returned and the query runtime) 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 corresponding version of the dataset in question.

[5] More details are available in the technical report at `http://goo.gl/LZehl1`.

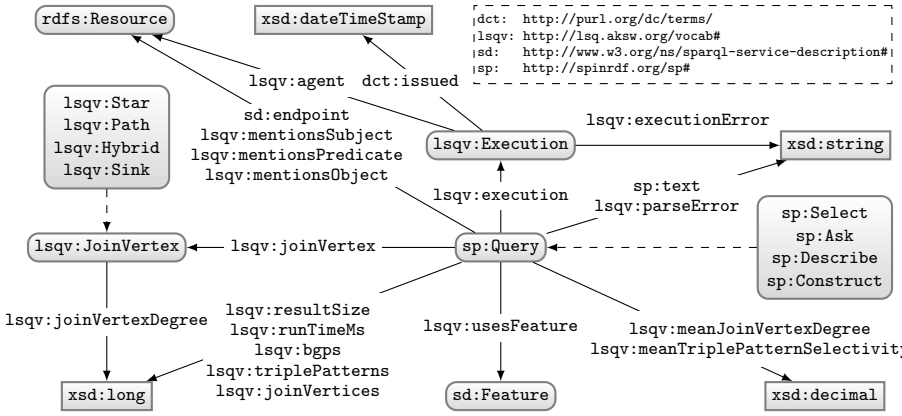

Fig. 1: LSQ data model (dashed lines indicate sub-classes)

Of course, the resulting statistics may differ to those that occurred during the original execution logged by the public endpoint. Likewise, these statistics are computed for a static version of the dataset using Virtuoso 7.1 (16 GB RAM, 6-Core i7 3.40 GHz CPU), where results may vary in other environments. These data are intended as a guide to query performance/result-size that is provided "as is" and which a consumer can choose to use or not use as they see fit.

Regarding Linked Data compatibility, we ensure that all query instances and executions are identified with dereferenceable IRIs. Our data model also re-uses class and property terms from established external vocabularies, including SPIN, DC Terms and SPARQL Service Descriptions. LSQ provides external links to every resource mentioned in a query. A SPARQL endpoint is also provided.

3 LSQ Dataset Statistics

We applied our extraction process over four SPARQL query logs: DBpedia (logs from 30/04/2010–20/07/2010; a dataset with 232 million triples), Linked Geo Data (LGD) (24/11/2010–06/07/2011; with 1 billion triples), Semantic Web Dog Food (SWDF) (16/05/2014–12/11/2014; with 300 thousand triples) and the British Museum (BM) (08/11/2014–01/12/2014; with 1.4 million triples). Given that the logs were in different formats (Virtuoso, Sesame and OWLIM), we wrote scripts to extract and normalise data from each source, mapping them to the target schema outlined in Section 2. In this section, we give some insights about the types of queries (and executions) that the resulting LSQ dataset describes.

Query Analysis: Table 1 provides high-level analysis of the queries appearing in the four logs. While the majority of queries are `SELECT` (91.6% overall), SWDF contains a large number of `DESCRIBE` queries (31.1%). The BM query log contains a noticeably high ratio of parse errors (77.63%), compared with DBpedia (35.27%),

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 ; rdf:type 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-WlxKEOQQRlhCUBdGRx1QGVRbQRNsN2YUWF5W ;
  dct:issued "2014-05-22T17:08:17+01:00"^^xsd:dateTimeStamp .
lsqrd:q483-e2 lsqv:agent lsqr:A-WlxKEOQQRlhCUBdGRx1QGVRdRBNsN2YUW1pS ;
  dct:issued "2014-05-20T14:34:35+01:00"^^xsd:dateTimeStamp .
lsqrd:q483-e3 lsqv:agent lsqr:A-WlxKEOQQRlhCUBdGRx1QGVRdRBNsN2YUW1pS ;
  dct:issued "2014-05-20T14:28:37+01:00"^^xsd:dateTimeStamp .
lsqrd:q483-e4 lsqv:agent lsqr:A-WlxKEOQQRlhCUBdGRx1QGVRdRBNsN2YUW1pS ;
  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 ] ]
     ]
 ) .
```

Table 1: 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 2: 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

SWDF (13.75%), or LGD (4.35%).[6] 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%). Often these are timeouts, which will, in practice, occur more frequently for larger datasets.

Table 2 shows the popularity of join types as defined previously in [13]. The idea is to count individual join variables within a BGP 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. Star has multiple outgoing links but no incoming links. Path has precisely one incoming and one outgoing link. 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 2, we see that most queries are Star (33.1%) or contain no join (66.5%); again we see the uniformity of BM queries suggesting the influence of one agent.

Table 3 shows the mean values for various query features across all query logs. These features are often considered, e.g., when designing SPARQL bench-

[6] We suspect that for BM, one automated agent is asking a high volume of simple potentially "invalid" queries to the endpoint; unfortunately the BM log did not include agent data, so we can neither confirm nor refute this possibility.

Table 3: 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 4: 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

marks [1,5]. 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 quite low (e.g., 0.35 for SWDF vs. 0.78 for DBpedia). We also see that slower runtimes correspond with larger dataset sizes. We again see that the BM queries often return zero results, suggesting again a high volume of simple, synthetic queries.

Tables 4 and 5 show the percentage use of (groups of) different SPARQL features [3]; a query is counted in a group if it uses one such feature. 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%).

Execution and Agent Analysis: Thus far we have analysed unique queries. We now look at (a) whether the same queries tend to be executed many times and (b) how many agents are responsible for how many 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 1, we can see that a given (syntactically identical) 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

Table 5: Percentage of queries using various classes of features

Dataset	Solution Mod.	Aggregates	(¬)Exists	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

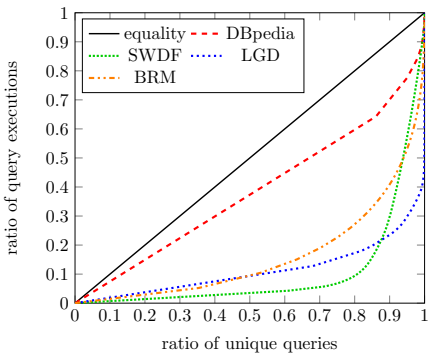

Fig. 2: Lorenz curve for distribution of executions per unique query

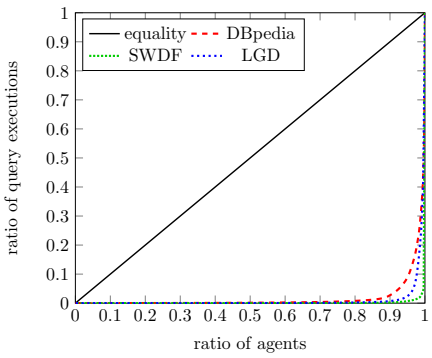

Fig. 3: Lorenz curve for distribution of executions per unique agent

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 are run a great many times and form the overall majority of executions.

Regarding unique agents, DBpedia had 3,041, LGD had 725 and SWDF had 274; we did not have agent data for BM. Figure 3 presents the Lorenz curve of how executions are distributed amongst agents, 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 most queries encountered in these logs are from a few high-volume, automated agents; this should potentially be taken into account by users of the LSQ dataset (again, our goal is to provide the queries from real-world logs "as is").

4 Conclusions and Future Directions

In this paper we presented LSQ, which is (to the best of our knowledge) the first public Linked Dataset describing SPARQL queries issued to endpoints. We introduced various use cases for LSQ, detailed our data model, and analysed

the results of RDFising logs from four endpoints. The current version of LSQ contains 73 million triples describing 5.7 million query executions.

We are currently collecting logs from other SPARQL endpoints (e.g., Bioportal, Strabon) that will be added into LSQ. We likewise hope to update and extend logs from current endpoints (esp. DBpedia). We will also link the dataset with the benchmark generation framework FEASIBLE to ease the development of benchmarks customised towards specific software applications or algorithms. The Linked Dataset, a SPARQL endpoint, and complete dumps are all available on the LSQ homepage – `http://aksw.github.io/LSQ/` – along with pointers to code, a VoID description, example LSQ queries, and various other dataset assets.

Acknowledgements: This work was supported in part by a research grant from the German Ministry for Finances and Energy under the SAKE project (Grant No. 01MD15006E), by Science Foundation Ireland (SFI) under Grant No. SFI/12/RC/2289, by the Millennium Nucleus Center for Semantic Web Research under Grant No. NC120004 and by Fondecyt Grant No. 11140900.

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. B. Berendt, L. Hollink, V. Hollink, M. Luczak-Rösch, K. Möller, and D. Vallet. Usage analysis and the web of data. *SIGIR Forum*, 45(1):63–69, 2011.
5. O. Görlitz, M. Thimm, and S. Staab. Splodge: Systematic generation of SPARQL benchmark queries for linked open data. In *ISWC*. 2012.
6. S. Harris, A. Seaborne, and E. Prud'hommeaux, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013.
7. H. Knublauch, J. A. Hendler, and K. Idehen, editors. *SPIN – Overview and Motivation*. W3C Member Submission, 22 February 2011.
8. 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*, 2011.
9. 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.
10. F. Picalausa and S. Vansummeren. What Are Real SPARQL Queries Like? In *SWIM*, 2011.
11. L. Rietveld and R. Hoekstra. Man vs. machine: Differences in SPARQL queries. In *USEWOD*, 2014.
12. M. Saleem, Q. Mehmood, and A.-C. Ngonga Ngomo. FEASIBLE: A featured-based SPARQL benchmark generation framework. In *ISWC*, 2015. (to appear).
13. M. Saleem and A.-C. Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In *ESWC*, 2014.
14. G. T. Williams and J. Weaver. Enabling fine-grained HTTP caching of SPARQL query results. In *ISWC*, pages 762–777, 2011.