A Hybrid Approach to Perform Efficient and Effective Query Execution Against Public SPARQL Endpoints

Maribel Acosta
Institute AIFB
Karlsruhe Institute of Technology
maribel.acosta@kit.edu
Supervised by Prof. Dr. Rudi Studer

ABSTRACT

Linked Open Data initiatives have fostered the publication of Linked Data sets, as well as the deployment of publicly available SPARQL endpoints as client-server querying infrastructures to access these data sets. However, recent studies reveal that SPARQL endpoints may exhibit significant limitations in supporting real-world applications, and public linked data sets can suffer of quality issues, e.g., data can be incomplete or incorrect. We tackle these problems and propose a novel hybrid architecture that relies on shipping policies to improve the performance of SPARQL endpoints, and exploits human and machine query processing computation to enhance the quality of Linked Data sets. We report on initial empirical results that suggest that the proposed techniques overcome current drawbacks, and may provide a novel solution to make these promising infrastructures available for real-world applications.

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**]: Human information processing; H.2.4 [**Systems**]: Query processing

Keywords

SPARQL Endpoint, RDF, Query Optimization, SPARQL Query, Hybrid System, Crowdsourcing, Microtasks, Quality Issues

1. PROBLEM STATEMENT

Linked Open Data (LOD) initiatives have promoted the publication of Linked Data (LD) sets in different knowledge domains including Life Sciences, news, geographical data, cross-domain and many others.[1] As of April 2014, more than 1,000 data sets have been made openly available using RDF and other standards and protocols.[2] One of the main mechanisms to consume a Linked Open Data set is via accessing publicly available SPARQL endpoints. Endpoints support a flexible interface to query RDF repositories: clients can pose queries against endpoints which, in theory, are capable of executing any given valid SPARQL query and return the results to the requester. Although SPARQL endpoints are acknowledged as a promising technology for RDF data access, this querying infrastructure still exhibits significant limitations in supporting real-world applications with high demands on performance (in terms of execution time and availability) and data quality.

A recent analysis by Buil-Aranda et al. [5] indicates that performance and availability vary notably between different public SPARQL endpoints. One of the reasons for the at times undesirable performance of public SPARQL endpoints is the unpredictable workload, since a large number of clients may be concurrently executing queries arbitrary complex queries against the endpoint. For instance, the DBpedia endpoint[3] processes almost 500,000 queries per day according to log files from DBpedia 3.8.[4] Scenarios like this impose new challenges for query processing solutions that access public SPARQL endpoints, in which optimization techniques should be tailored not only to produce results in a *reasonable time*, but also to take into account the endpoint performance in terms of computational resources to ensure its *availability*.

An orthogonal but equally important aspect of accessing public endpoints is the quality of the retrieved data. Recent studies reveal that RDF [8] data sets exhibit varying quality in different dimensions including completeness, semantic validity, and semantic accuracy [17]. Quality issues in LD often pose a serious problem to developers aiming to consume and integrate LD in their applications. In addition, executing queries against data with quality issues leads to low-quality results. The challenge is then to detect and correct inaccurate portions of the data set during query processing to produce *complete* and *high-quality* results. Recent research shows that certain LD quality issues can be repaired automatically [12]; however, other quality issues require further semantic interpretations that could be easily assessed by humans.

In order for the public SPARQL endpoint infrastructure to reach its full potential, more flexible client-server architectures in terms of performance and quality are needed. The main research goal of this work is to define a solution to efficiently and effectively access public SPARQL endpoints, taking into account the capabilities of endpoints and the quality requirements of clients. We identify three specific research goals: **(RG1)** Efficient query execution against public SPARQL endpoints, where efficiency is defined in terms of execution time; **(RG2)** Effective query execution enhancing data completeness; **(RG3)** Effective query execution via data cleansing, to handle LD quality issues. To accomplish our research goals, we propose a hybrid system that combines human and computational intelligence to execute queries against a given SPARQL endpoint.

[1] `lod-cloud.net`
[2] `http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/`
[3] SPARQL endpoint: `http://dbpedia.org/sparql`
[4] USEWOD 2013 Dataset, available at: `http://data.semanticweb.org/usewod/2013/challenge.html`

WWW'15 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
`http://dx.doi.org/10.1145/2740908.2741750.`

We have implemented and empirically analyzed the performance of the core components of our system. Results show that our approach is feasible and able to outperform existing solutions.

2. STATE OF THE ART

We focus on investigating two types of related work: *SPARQL endpoint availability* and *hybrid (human/computer) systems*.

SPARQL Endpoint availability. SPARQL endpoints attempt to completely execute client queries that meet certain complexity and selectivity restrictions. Performance statistics of public SPARQL endpoints have been recently collected by Buil-Aranda et al. [5]; reported results reinforce the statement that SPARQL endpoints as a stand-alone querying infrastructure are not very likely to succeed, except for very simple queries. Verborgh et al. [15] proposed Linked Data Fragments (#LD), an architecture to execute queries against SPARQL endpoints. #LD shifts parts of the query from the server to the client; all the query operators are executed at the client while the server is dedicated to transfer data fragments to resolve the query. Data fragments are gathered by contacting the endpoint in a nested-loop fashion. While caching techniques may reduce data lookups, this approach may not scale up to large intermediate results because a relatively small number of simultaneous threads are usually allowed from the same IP in public endpoints [13].

Hybrid Systems. Approaches such as CrowdDB [6], Deco [10], and Qurk [9] target scenarios in which existing microtask platforms are directly embedded in relational query processing systems. CrowdDB [6] provides SQL-like data definition and query languages to support hybrid query execution, and attempts to reduce the number of crowdsourcing tasks in enumeration queries by exploiting structural properties of the relational data [14]. Deco [10] implements caching strategies to reuse previously crowdsourced data. Additionally, Deco [11] and Qurk [9] provide a set of physical operators and models to estimate selectivity and cardinality. These statistics in conjunction with the physical operators allow for defining physical plans that reduce execution time, monetary cost, and number of tasks. All of these approaches require user intervention on the definition of the crowd-based workflows and do not take into consideration the performance of the crowd. In contrast, we propose a solution to automatically create hybrid query-driven workflows exploiting crowd knowledge to on-the-fly decide whether the crowd is knowledgeable to resolve certain types of questions.

Furthermore, crowdsourcing has also shown to be feasible for other scenarios related to Semantic Web technologies. Amsterdamer et al. propose OASSIS [4], a system that combines general knowledge from ontologies with frequent patterns mined from personal data collected from the crowd. OASSIS provides a SPARQL-like query language where users specify sub-queries that will be evaluated against the ontology and the ones that will be mined from the crowd. The OASSIS query engine is able to order the execution of the sub-queries in a way that questions posed to the crowd are minimized. LODRefine,[5] an LD integration tool, has made available an extension that allows to manually configure and run specific data matchmaking tasks on a microtask platform. Although these approaches address different LD management problems, they are not tailored to tackle incomplete or incorrect values in RDF data.

3. PROPOSED APPROACH

We propose a hybrid query processing system that combines human and computer capabilities to execute queries against public SPARQL endpoints. Figure 1 depicts the main components of the proposed architecture. Below we describe each component.

[5]https://github.com/sparkica/LODRefine

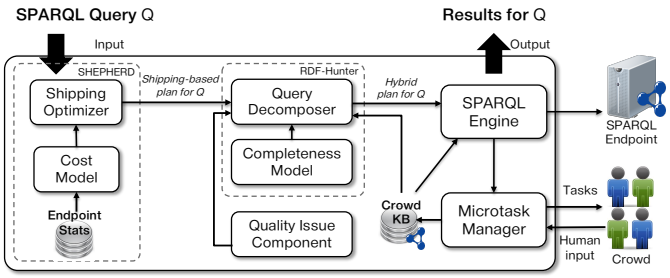

Figure 1: The proposed architecture

When a query Q is issued, Q is firstly optimized by SHEPHERD [2]. This component implements optimization techniques based on a novel cost model. The SHEPHERD optimizer is tailored to devise plans that are efficiently executed by public endpoints.

Plans generated by SHEPHERD are then processed by RDF-Hunter [1]. The objective of this component is to detect parts of the query Q that yield incomplete results. The RDF-Hunter query decomposer generates a hybrid plan for Q composed of two partitions: one partition is executed against the endpoint, the other is solved via human computation. The query decomposer in combination with the quality issue component is able to identify quality issues [3] in the intermediate results of Q. Potential erroneous data retrieved from the endpoint is resorted to human assessment to detect and provide correct values to build high-quality results for Q.

Hybrid (human/computer) plans for Q are executed by the SPARQL engine. Intermediate results of the human-driven sub-plan are submitted to the microtask manager, responsible for enabling human intelligence in our system via microtasks. Microtask crowdsourcing consists on resorting questions (tasks) to human contributors; each question is typically solved in parallel by a high number of contributors in a fast and affordable way. The microtask manager builds human tasks exploiting the semantics of the RDF resources to build rich user interfaces that facilitate the crowd's work. User interfaces display values (if available) of human-readable properties such as short description, picture, geo-location (which is shown in a map), links to the homepage, and corresponding Wikipedia article. The microtask manager submits the human tasks to existing microtask marketplaces such as Amazon Mechanical Turk[6] (MTurk) and CrowdFlower.[7] The SPARQL engine combines the intermediate results retrieved from the data set with the human input and produces the results for Q.

3.1 SHEPHERD: Efficient Query Execution Against Public SPARQL Endpoints

The SHEPHERD component [2] generates query plans to be efficiently executed against public endpoints while shifting workload from the server to the client. This component considers the capabilities of the addressed SPARQL endpoint, and offers a competitive performance in terms of execution time and the number of answers. To do this, SHEPHERD implements shipping policies [7] which allow for deciding where the different parts of a query will be executed according to the capabilities of endpoints. This component addresses the object from our research goal **(RG1)**.

SHEPHERD collects *Statistics* from SPARQL endpoints in order to characterize endpoint performance in terms of: execution time and size of result set. Gathering relevant statistics from public endpoints is a challenging task, since the only mechanism to retrieve information from the endpoints is by submitting SPARQL queries.

[6]https://www.mturk.com/
[7]https://www.crowdflower.com/

Therefore, queries used to collect statistics must be able to be executed by the endpoint while providing useful information. Collected statistics provide reliable estimators of performance used by the cost model. SHEPHERD implements a novel *Cost Model* able to estimate the cost of plans based on the combination of query execution time and the endpoint computational resource consumption.

The *Shipping Optimizer* implements a cost-based optimization. The shipping optimizer traverses the space of plans exploring different shipping policies. During the optimization process, SHEPHERD decides whether to place the operators at the server (i.e., endpoint), or client (i.e., SHEPHERD) according to the statistics of each endpoint. In this way, SHEPHERD explores shipping polices tailored for each public endpoint to enhance their reliability.

3.2 RDF-Hunter: Effective Query Execution Enhancing Data Completeness

RDF-Hunter [1] combines human and computer capabilities to enhance the answer completeness of queries. For instance, executing a query against the DBpedia endpoint that selects *"movies, including their producers, that have been filmed in Italy"* returns no producers for 16% of the movies in the result set. The goal of RDF-Hunter is then to find missing values in the data set, e.g., missing movie producers in DBpedia, via microtask crowdsourcing. RDF-Hunter provides a highly flexible crowdsourcing-enabled SPARQL query execution: no extensions to SPARQL or RDF are required, and users can configure the level of expected answer completeness. This component addresses our research goal **(RG2)**.

A novel *Completeness Model* for RDF data sets is implemented by RDF-Hunter. This model estimates the completeness level of portions of a data set by comparing the topology of RDF subgraphs. In our running example, the quality model compares the sub-graph of all movies and the sub-graph of movies filmed in Italy. This comparison reveals that most of the movies have at least one producer, therefore, movies without producers are considered as incomplete data according to the quality model.

The *Crowd Knowledge Base* (or Crowd KB) stores the answers retrieved from the crowd and describes the types of questions the crowd is likely to be able to solve accurately. The *Microtask Manager* aggregates results from human tasks assessed by the crowd so far, and annotates them with a confidence score. To illustrate this, consider that the crowd is enquired to provide the producer(s) of the movie Ocean's Twelve and the aggregated answer is "Jerry Weintraub is a producer of Ocean's Twelve" with a confidence of 0.90, then the high value of confidence suggests that the crowd is knowledgeable answering this type of questions.

RDF-Hunter implements query decomposition techniques to automatically decide the parts of a query that should resort to the crowd. The *Query Decomposer* combines information collected from the quality model and the crowd knowledge base to enhance the answer completeness. For example, based on the completeness model, the decomposer decides to crowdsource the producers of movies shot in Italy. However, the crowd have provided information with high confidence about the producer of Ocean's Twelve, then this particular instance will not be assessed by the crowd again.

3.3 Quality Issue Component: Effective Query Execution via Data Cleansing

The quality issue component [3] allows for detecting low-quality data retrieved from the SPARQL endpoint. Among the quality issues that can be detected are the following:

Incorrect Object. This quality issue is present in RDF triples in which the object position has an erroneous value. For instance, DBpedia asserts that one of the dates of the tournament 2005 Six Nations Championship is "0001-02-05". However, according to its Wikipedia article, the correct date in this case is "2005-02-05".

Incorrect Data Type. Objects of RDF triples can be annotated with metadata that denotes the type of data such as integer, double, date, etc. This quality issue states that the data type assigned is inaccurate. For example, the year in which the HDI (Human Development Index) of Italy was calculated in DBpedia is annotated as an integer number (xsd:integer). To allow for correctly handling the data from the example, e.g., comparison with other dates, the expected data type in this case is xsd:gYear.

Incorrect Hyperlink. This quality issue states that the content from a web page is unrelated to the subject of an RDF triple. For instance, DBpedia links the RDF resource Italian language to the page `http://etcweb.princeton.edu/dante/pdp/`, although the content of this website is not directly associated with the resource.

Analogous to the RDF-Hunter workflow (cf. Section 3.2), the *Query Decomposer* relies on the quality issue component to determine the parts of the query that might yield results with quality issues. The crowd is enquired to provide correct values to build the query results. The output from the crowd is stored in the *Crowd Knowledge Base* and exploited in subsequent executions. The combination of the quality issue module with automatic query decomposition techniques allow for realizing our research goal **(RG3)**.

4. METHODOLOGY

The methodology adopted in the development of this doctoral work adheres to the following tasks:

1. Investigation of state-of-the-art solutions relevant to the studied problem. This includes the study of literature about efficient query processing approaches published in the areas of Databases and Semantic Web, as well as publications of solutions that apply crowdsourcing to computational problems.

2. Formalization of the problem to solve and formulation of research questions and hypotheses. Then, the proposed solution is defined, identifying the novel contributions. The formal properties of the proposed solution will be demonstrated, including an analysis of the space and time complexity for the best and worst cases. The proposed solution will be implemented and made available to the community.

3. Empirical evaluation of the proposed solution to measure its performance with respect to state-of-the-art approaches. The experiments will be conducted as follows: *i*) Implementation of state-of-the-art or baselines approaches. *ii*) Definition of benchmarks to measure the quality of the proposed solution. For the experiments, appropriate LD sets, queries, and SPARQL endpoints will be selected as well as metrics to measure the efficiency and effectiveness of our solution in comparison to the state of the art. *iii*) Execution of experiments and statistical studies of the obtained results to draw conclusions about the hypotheses. *iv*) Analysis of the results.

5. EXPERIMENTAL STUDIES & RESULTS

5.1 Evaluating SHEPHERD

We empirically compared the performance of the hybrid shipping policies implemented by SHEPHERD with the query shipping policy when executing queries directly against the endpoint.

Public Endpoints and Query Benchmarks: We selected four public SPARQL endpoints that exhibit different performance [5]:

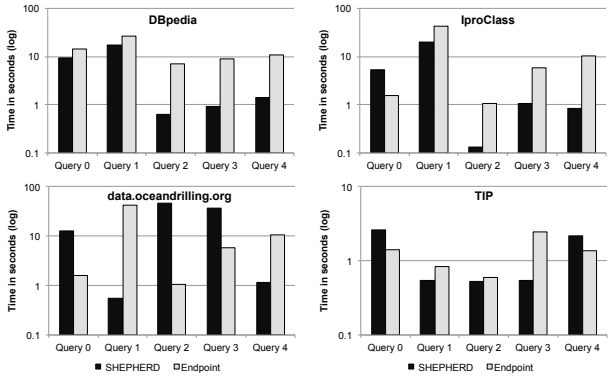

Figure 2: SHEPHERD vs. Endpoint: Runtime when benchmark queries are executed against four different public endpoints

F-Measure	Life Sciences	History	Music	Sports	Movies
Query 1	0.83	1.00	0.80	0.91	1.00
Query 2	0.80	0.68	0.44	1.00	1.00
Query 3	0.94	1.00	1.00	1.00	1.00
Query 4	1.00	0.80	0.78	1.00	0.61
Query 5	1.00	0.96	0.84	0.71	1.00
Query 6	1.00	0.98	1.00	0.81	1.00
Query 7	1.00	0.75	0.91	1.00	0.88
Query 8	1.00	0.98	0.80	0.96	0.98
Query 9	0.93	0.97	0.75	0.92	1.00
Query 10	1.00	0.75	0.66	0.63	0.97
Average	0.95	0.89	0.80	0.89	0.94

Figure 3: Heat map of F-Measure achieved when evaluating the benchmark queries against the crowd with RDF-Hunter

DBpedia, IproBio2RDF, data.oceandrilling.org, and TIP.[8] By analyzing triple patterns answerable for each endpoint, we designed a query benchmark comprising five different queries per endpoint.[9]
Implementations: SHEPHERD was implemented using Python 2.7.6. Queries were executed directly against the endpoints using the command `curl`. In the experiments, the clients were deployed on the Amazon EC2 Elastic Compute Cloud infrastructure.[10]
Evaluation Metrics: *i*) *Execution Time*: The elapsed time from the moment that the engine issues the query until it produces the last result. It is measured with the Python `time.time()` function. *ii*) *Size of Result Set*: Number of tuples produced by the engine when a query is executed.

Results: Execution Time & Size of Result Set
Figure 2 depicts the result of executing the benchmark queries in terms of the execution time (sec.). We can observe that in the majority of the cases SHEPHERD retrieves the results notably faster compared to the query shipping policy (endpoint), except in three queries. This suggests that the hybrid-shipping plans implemented in SHEPHERD can be efficiently executed against public endpoints.

Concerning the size of the retrieved result set, a similar picture emerges. For 18 of the total 20 queries tested, both SHEPHERD and the query-shipping client produced the same amount of results, while in one instance each SHEPHERD and the query-shipping client did not retrieve any answers. Both methods therefore seem to be on par in this regard.

SHEPHERD is able to execute queries against endpoints in an efficient way. This could not be achieved neither by simply moving all operator execution to the client – since this increments the bandwidth consumption and the evaluation of non-selective queries may starve the resources of the client – nor by submitting the whole query to the endpoint as shown in our experiments. These results suggest that the proposed techniques can improve efficiency of SPARQL endpoints, addressing our research goal **(RG1)**.

5.2 Evaluating RDF-Hunter

We empirically assessed the answer completeness achieved by RDF-Hunter. To allow for *reproducibility*, setting details of these experiments and further results are available online.[11]
Crowdsourcing Platform: We use the CrowdFlower platform.

Query Benchmark and Gold Standard: We designed a benchmark of 50 queries that yield incomplete results for the DBpedia SPARQL endpoint. We designed 10 queries each to belong in one of these five categories: *Historical*, *Life Sciences*, *Movies*, *Music*, and *Sports*. We built a gold standard of the missing answers of the 50 queries that required to be collected from the crowd.
Evaluation Metrics: *i*) *Precision*: Fraction of the answers collected from the crowd that actually corresponds to the query answers. *ii*) *Recall*: Fraction of the missing answers that are collected from the crowd. *iii*) *F-Measure*: Combines the values of precision and recall to measure the accuracy of the crowd output.

We crowdsourced a total of 502 RDF triples, and collected 1,619 answers from the crowd.

Results: F-Measure
We report the results of F-measure using a heat map (cf. Figure 3). The darkest color represents values of F-measure equals to 1.0. Columns represent the five query categories, while rows correspond to the benchmark queries.

We can observe that in 21 out of 50 queries, the crowd was able to correctly and completely provide all the missing values. Furthermore, for 39 queries the achieved recall is greater than 0.80. In each column, it is noticeable that although the queries belong to the same category, the crowd achieved a variable range of accuracy (F-measure). This observation provides evidence of the importance of identifying portions of the domain where the crowd is knowledgeable. Thus, in subsequent requests, RDF-Hunter will exploit this knowledge to avoid crowdsourcing these two questions again. These results showed that answer completeness can be enhanced with our solution, as formulated in our research goal **(RG2)**.

5.3 Detecting LD Quality Issues

We measured the quality of the crowd when detecting erroneous triples from an LD set. To allow for *reproducibility*, all the settings of these experiments and additional results are available online.[12]
Crowdsourcing Platform: We use the MTurk platform.
Data Set and Gold Standard: We used 1,512 triples resulting from the DBpedia Evaluation Campaign [16] in which 58 LD experts participated. We generated a gold standard by evaluating these triples and resolving the conflicts via mutual agreement.
Metrics: *i*) *Precision*: Measures the fraction of the triples that were correctly identified as erroneous by the crowd.

After removing duplicates and broken links from the contest data we submitted 1,073 triples to MTurk: 509 for *Incorrect Object* task,

[8] Available at `http://iproclass.bio2rdf.org/sparql`, `http://data.oceandrilling.org/sparql` and `http://lod.apc.gov.tw/sparql`
[9] `http://people.aifb.kit.edu/mac/shepherd/`
[10] `https://aws.amazon.com/ec2/instance-types/`
[11] `http://people.aifb.kit.edu/mac/rdf-hunter`
[12] `http://people.aifb.kit.edu/mac/DBpediaQualityAssessment/`

341 for *Incorrect Data Type* task, and 223 for *Incorrect Hyperlinks* task. A total of 80 distinct workers participated in this experiment.

Results: Precision

For the *Incorrect Object* task, the MTurk worker reached a precision of 0.90, while the LD experts 0.72. Most of the incomplete values that are extracted from Wikipedia occur with the predicates related to dates. In addition, there were 52 DBpedia triples whose values might seem erroneous, although they were correctly extracted from Wikipedia. We found out that the LD experts classified all these triples as incorrect. In contrast, the workers successfully answered that 50 out of this 52 were correct, since they compared the DBpedia and Wikipedia values in the microtasks.

In the *Incorrect Data Type* task, the experts reached 0.83 of precision on *finding* this type of quality issue, while the precision of the crowd on *verifying* these triples is 0.51. The low performance of the MTurk workers compared to the experts is not surprising, since this particular task requires certain technical knowledge about data types and, moreover, the specification of values and types in LD.

In the *Incorrect Hyperlink* task, the extremely low precision of 0.15 of the contest's participants was unexpected. We discarded the possibility of a malfunction of the tool used during the contest. The MTurk workers achieved a precision of 0.94. The 6% of the links that were not properly classified by the crowd corresponds to those web pages whose content is in a different language than English or, despite they are referenced from the Wikipedia article of the subject, their association to the subject is not straightforward.

These experiments suggest that hybrid human and computer techniques can assess effective tasks of data cleansing, and thus address our research goal **(RG3)**.

6. CONCLUSIONS AND FUTURE WORK

In this doctoral work, we study the problem of query execution against public SPARQL endpoints. We propose a hybrid architecture that combines query processing techniques with crowdsourcing able to: i) ship workload from endpoints to the client, and ii) produce complete and high-quality results, while achieving competitive performance with existing solutions. Our initial experimental study demonstrates that our proposed solution can enhance the performance of public SPARQL endpoints by devising query plans according to the capacity of the endpoint. Another important result is the feasibility of automatically incorporating human computation via microtask crowdsoourcing into LD-specific problems by exploiting the structure of RDF graphs. The data retrieved from the crowd can be further incorporated into LD sets to enhance the quality of subsequent SPARQL query answers.

Future work will focus on empirically analyzing the performance of our proposed architecture on different scenarios, including a more extensive study with different public SPARQL endpoints. This will allow for generalizing the properties observed so far.

Acknowledgements

The author would like to thank to Andreas Harth, Elena Simperl, and Maria-Esther Vidal for their guidance and fruitful discussions during the development of this doctoral work.

7. REFERENCES

[1] M. Acosta, E. Simperl, F. Flöck, and M. Vidal. RDF-Hunter: Automatically crowdsourcing the execution of queries against RDF data sets. Under review.

[2] M. Acosta, M. Vidal, F. Flöck, S. Castillo, C. B. Aranda, and A. Harth. SHEPHERD: A shipping-based query processor to enhance SPARQL endpoint performance. In *Proc. International Semantic Web Conference, ISWC, Posters & Demonstrations Track*, pages 453–456, 2014.

[3] M. Acosta, A. Zaveri, E. Simperl, D. Kontokostas, S. Auer, and J. Lehmann. Crowdsourcing linked data quality assessment. In *Proc. International Semantic Web Conference, ISWC*, pages 260–276, 2013.

[4] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *Proc. International Conference on Management of Data, SIGMOD*, pages 589–600, 2014.

[5] C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *Proc. International Semantic Web Conference, ISWC*, pages 277–293, 2013.

[6] M. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *Proc. International Conference on Management of Data, SIGMOD*, pages 61–72, 2011.

[7] M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *Proc. International Conference on Management of Data, SIGMOD*, pages 149–160, 1996.

[8] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven evaluation of linked data quality. In *Proc. International Conference on World Wide Web, WWW*, pages 747–758, 2014.

[9] A. Marcus, D. R. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[10] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.

[11] H. Park and J. Widom. Query optimization over crowdsourced data. *PVLDB*, 6(10):781–792, 2013.

[12] H. Paulheim and C. Bizer. Improving the quality of linked data using statistical distributions. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):63–86, 2014.

[13] M. Salvadores, M. Horridge, P. R. Alexander, R. W. Fergerson, M. A. Musen, and N. F. Noy. Using SPARQL to query bioportal ontologies and metadata. In *Proc. International Semantic Web Conference, ISWC*, pages 180–195, 2012.

[14] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *Proc. International Conference on Data Engineering, ICDE*, pages 673–684, 2013.

[15] R. Verborgh, O. Hartig, B. D. Meester, G. Haesendonck, L. D. Vocht, M. V. Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. V. de Walle. Querying datasets on the web with high availability. In *Proc. International Semantic Web Conference, ISWC*, pages 180–196, 2014.

[16] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, and J. Lehmann. User-driven quality evaluation of dbpedia. In *Proc. International Conference on Semantic Systems, I-SEMANTICS*, pages 97–104, 2013.

[17] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web Journal*, 2015. (To appear).