Observations on Bloom Filters for Traversal-Based Query Execution over Solid Pods

Jonni Hanski, Ruben Taelman, and Ruben Verborgh

IDLab, Department of Electronics and Information Systems, Ghent University – imec

Abstract. Traversal-based query execution enables the resolving of queries over Linked Data documents, using a follow-your-nose approach to locating query-relevant data by following series of links through documents. This traversal, however, incurs an unavoidable overhead in the form of data access costs. Through only following links known to be relevant for answering a given query, this overhead could be minimized. Prior work exists in the form of reachability conditions to determine the links to dereference, however this does not take into consideration the contents behind a given link. Within this work, we have explored the possibility of using Bloom filters to prune query-irrelevant links based on the triple patterns contained within a given query, when performing traversal-based query execution over Solid pods containing simulated social network data as an example use case. Our discoveries show that, with relatively uniform data across an entire benchmark dataset, this approach fails to effectively filter links, especially when the queries contain triple patterns with low selectivity. Thus, future work should consider the query plan beyond individual patterns, or the structure of the data beyond individual triples, to allow for more effective pruning of links.

1 Introduction

the traversal-based approach to query execution [7] builds upon the Linked Data principles [2] by offering a formally defined foundation for executing queries over such data [7]. This approach to query execution essentially functions by following directed *data links* between documents, integrating data discovery with query execution. Within a given *reachable subweb* of Linked Data documents and the data contained therein, bounded by the chosen *reachability-based query semantics*, this approach provides a computationally feasible means of executing queries while guaranteeing termination and completeness [6]. However, while this reachable subweb does provide bounds for the set of documents to consider via the *reachability conditions* set forth by the reachability-based query semantics, depending on the chosen conditions, not all of these documents may contain data relevant for answering a given query or even links to other documents that would satisfy the conditions. Therefore, potential to further restrict these reachability conditions may exist, by requiring the linked documents to contain either query-relevant data or further links satisfying the original conditions, provided sufficient knowledge of the data contained behind a given link is available, which is the direction we have chosen to pursue within this work.

2 Methodology

Within this work, we have chosen to explore the extension of *reachability conditions* to filter data links based on the contents of the documents they link to. Essentially, the *reachability-based semantics* [6] restrict the scope of queries to a reachable subweb of Linked Data documents, where reachable documents are those that can be discovered by following chains of links that meet the reachability conditions, starting from the initial set of URIs either provided explicitly or extracted from the query itself. This restriction facilitates computability not attainable under full-Web semantics, without having to introduce purpose-built termination mechanisms that might result in nondeterministic execution [6]. the original authors present three reachability criteria: c_{All}, c_{None} and c_{Match} [6]. Essentially, c_{All} follows all data links, c_{None} follows no links at all, and c_{Match} follows links conditionally. With c_{None} having no further room for restrictions in traversal, we focused on c_{Match}, though c_{All} would also have worked.

We have conceptually extended this criterion to further restrict the data links considered for traversal, by including the use of Bloom filters to check whether the document identified by a given URI contains potentially query-relevant data or further links matching the criterion. Bloom filter is a probabilistic data structure, that may produce false positives but never false negatives [3], making it suitable for this purpose by not facilitating the accidental pruning of links that could point to query-relevant data. the filter is essentially an array of bits with a specified length, and entries to it are added by hashing the input into another array of bits of the same length and taking a bitwise `OR`. Testing whether a filter contains a specific value works by hashing the input and doing a bitwise `AND`.

Following the RDF schema [4], an RDF statement – a triple – consists of resources, properties and literals. Resources may appear as the subject or object of a statement, properties as the predicate, and literals as the object. Within this work, we ignored literals due to their versatility with regards to languages and datatypes, as well as blank node identifiers of resources due to potentially different means of generating or skolemizing them. This left us with globally unique URI identifiers for resources and properties, for which we will generate our Bloom filters. Through the reuse of an existing membership filter vocabulary[1], we can define a Bloom filter as a combination of a dataset URI, a *projected property* or *projected resource*, and the filter itself, where a property or resource is a globally unique URI identifier of either.

Essentially, a Bloom filter defined this way offers the means to check whether a given property or resource URI occurs within the same triples together with another property or resource URI. Inspired by c_{Match}, for each data link u, if there exist Bloom filters generated for the dataset it is part of based on URI prefixes, we check whether any combination of property or resource URIs in the triple patterns in the query could be found in the dataset. Only if applicable filters exist, and only if all such filters reject every triple pattern in the query, can we reject the data link u. Otherwise, this link is accepted.

[1] http://semweb.mmlab.be/ns/membership

3 Experiments

Following the example set by prior evaluations of traversal-based query execution [9], we have chosen to evaluate our approach to Bloom filters using SolidBench[2], an adaptation of the LDBC social network benchmark [5] for the Solid initiative [10]. Within the scope of this work, we have integrated the generation of Bloom filters into the benchmark dataset preprocessing tool[3], and implemented the discovery and use of these filters in Comunica [8], a query engine framework also used in prior work on traversal-based query execution, as a set of additional components. Both the query engine components[4] and the experiments themselves[5] are available online for reproducibility.

For the purposes of evaluating the impact of Bloom filters, the following test cases were considered: *i) no filters* as the baseline for comparison, *ii) per-pod filters* generated for the full contents of a given Solid pod, *iii) per-subdirectory* filters generated for each subdirectory within a pod, and *iv) per-document* filters generated for each document. Within all the test cases, the filters were placed at the pod root for discovery prior to dereferencing individual subdirectories or documents. the experiments were executed by having both the Solid server to serve the data and the query engine to query it on the same machine. the main purpose of the experiments was to measure the differences in network request counts when using Bloom filters.

4 Observations

Initial results in Table 1 show the Bloom filters fail to prune any links, and their inclusion appears to unintentionally cause more links to be dereferenced, likely due to the URIs in filter metadata triples being picked by c_{Match} to the link queue. the average time taken to produce the first and the last result, as well as the combined diefficiency [1] value are also not significantly different.

Upon further analysis of the benchmark dataset and the queries, detailed in Table 2, this ineffective filtering appears to be caused by a handful of triple patterns matching most of the pods and documents, simply by virtue of the data being relatively uniform and merely distributed across a number of similarly structured documents and pods. For example, the worst offender in the form of `?s ldbc:hasCreator ?o` is found in 100% of the pods, and in 96% of all the documents, causing all queries with this pattern to not exclude any links in practice, even though a mere 2.5% of all triples match this pattern. Even if this were somehow addressed, additional manual tests using c_{All} instead of c_{Match} revealed no difference with a number of example queries, as the benchmark dataset mostly links to itself and is relatively uniform.

[2] https://github.com/SolidBench/SolidBench.js
[3] https://github.com/SolidBench/rdf-dataset-fragmenter.js
[4] https://github.com/surilindur/comunica-components
[5] https://github.com/surilindur/comunica-experiments

	Filter scope	Queries	Requests	$\overline{t_{first}}$ (s)	$\Delta\overline{t_{first}}$	$\overline{t_{last}}$ (s)	$\Delta\overline{t_{last}}$	Δ ***dieff@full***
1.	**no filters**	25 / 75	1,047	2.69	0.00 %	3.06	0.00 %	0.00 %
2.	**per-pod**	27 / 75	1,062	2.73	+1.49 %	3.02	-1.31 %	-10.18 %
3.	**per-subdir**	26 / 75	1,062	2.75	+2.23 %	3.03	-0.98 %	-4.02 %
4.	**per-document**	26 / 75	1,062	2.87	+6.69 %	3.13	+2.29 %	-4.10 %

Table 1. Overview of initial benchmark results for different Bloom filter configurations. the combined total HTTP request count and diefficiency values to produce all results (*dieff@full*), as well as the average time to produce the first (t_{first}) and last (t_{last}) result, are taken only for the common 10 queries that succeeded for all configurations.

5 Conclusions

Within this work, we have explored the use of Bloom filters for pruning query-irrelevant data links during traversal-based query execution over the SolidBench benchmarking dataset. Unfortunately, due to the relatively uniform nature of data contained within the benchmark and the type of triple patterns found in the associated queries, the chosen method of generating Bloom filters at triple level appears ineffective in pruning links. This leads us to conclude that the triple-level information on the co-occurrence of specific URIs, contained within Bloom filters as implemented within this work, is insufficient for pruning links when combined with triple patterns that have too many variables or when the data behind the majority of links partially matches the sought-after data. Thus, we believe triple patterns with fewer variables should be looked into for filtering, perhaps by taking into consideration the patterns' positions in the query plan, or by testing links against intermediate results rather than raw triple patterns, to the extent possible without unintentionally excluding query-relevant data. Additionally, approaches that capture both the data shape beyond individual triples and query structure beyond individual triple patterns should likely be investigated for more efficient filtering.

Acknowledgements. The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral fellow of the Research Foundation – Flanders (FWO) (1202124N).

References

1. Acosta, M., Vidal, M.E., Sure-Vetter, Y.: Diefficiency metrics: measuring the continuous efficiency of query processing approaches. In: The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II 16. pp. 3–19. Springer (2017)
2. Berners-Lee, T.: Linked data - design issues (2006), http://www.w3.org/DesignIssues/LinkedData.html
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)

#	Pattern	Queries	Pods	Documents	Triples
1.	`?s ldbc:hasCreator ?o`	20.00 %	100.00 %	96.22 %	2.49 %
2.	`?s ldbc:creationDate ?o`	33.33 %	100.00 %	84.71 %	3.39 %
3.	`?s ldbc:id ?o`	60.00 %	100.00 %	84.71 %	2.37 %
4.	`?s ldbc:isLocatedIn ?o`	6.67 %	100.00 %	84.71 %	2.37 %
5.	`?s ldbc:locationIP ?o`	6.67 %	100.00 %	84.71 %	2.37 %
6.	`?s ldbc:firstName ?o`	40.00 %	100.00 %	1.28 %	0.01 %
7.	`?s ldbc:lastName ?o`	40.00 %	100.00 %	1.28 %	0.01 %
8.	`?s (ldbc:content|ldbc:imageFile) ?o`	6.67 %	95.62 %	83.43 %	2.36 %
9.	`?s (ldbc:hasPost|ldbc:hasComment) ?o`	6.67 %	94.24 %	1.21 %	0.90 %
10.	`?s ldbc:content ?o`	26.67 %	92.15 %	54.44 %	1.37 %
11.	`?s ldbc:replyOf* ?o`	6.67 %	91.62 %	48.48 %	1.24 %
12.	`?s rdf:type ldbc:Comment`	20.00 %	91.62 %	48.48 %	1.24 %
13.	`?s ldbc:hasTag ?o`	6.67 %	88.68 %	24.58 %	2.00 %
14.	`?s rdf:type ldbc:Post`	26.67 %	88.15 %	34.95 %	1.12 %
15.	`?s ldbc:hasPerson ?o`	6.67 %	78.47 %	1.00 %	0.12 %
16.	`?s ldbc:knows ?o`	6.67 %	78.47 %	1.00 %	0.12 %
17.	`?s ldbc:hasPerson <http://.../pods/00000006597069767117/profile/card#me>`	1.33 %	0.20 %	0.00 %	0.00 %
18.	`?s ldbc:hasCreator <http://.../pods/00000004398046512167/profile/card#me>`	10.67 %	0.07 %	0.18 %	0.00 %
19.	`?s ldbc:hasCreator <http://.../pods/00000006597069767117/profile/card#me>`	10.67 %	0.07 %	0.07 %	0.00 %
20.	`?s ldbc:hasCreator <http://.../pods/00000000000000000933/profile/card#me>`	10.67 %	0.07 %	0.07 %	0.00 %
21.	`?s ldbc:hasCreator <http://.../pods/00000000000000001129/profile/card#me>`	10.67 %	0.07 %	0.01 %	0.00 %
22.	`?s ldbc:hasCreator <http://.../pods/00000002199023256684/profile/card#me>`	10.67 %	0.07 %	0.01 %	0.00 %
23.	`<http://.../pods/00000004398046512167/profile/card#me> ldbc:likes ?o`	1.33 %	0.07 %	0.00 %	0.00 %
24.	`<http://.../pods/00000006597069767117/profile/card#me> ldbc:likes ?o`	1.33 %	0.07 %	0.00 %	0.00 %
25.	`<http://.../pods/00000000000000001129/profile/card#me> ldbc:likes ?o`	1.33 %	0.07 %	0.00 %	0.00 %
26.	`<http://.../pods/00000000000000000933/profile/card#me> ldbc:likes ?o`	1.33 %	0.07 %	0.00 %	0.00 %
27.	`<http://.../pods/00000002199023256684/profile/card#me> ldbc:likes ?o`	1.33 %	0.07 %	0.00 %	0.00 %
	...	...	...	...	...
141.	`<http://.../pods/00000015393162789111/posts#893353506423> ldbc:replyOf* ?o`	1.33 %	0.00 %	0.00 %	0.00 %

Table 2. When looking at the data stored in Solid pods in the SolidBench benchmark, and comparing it against individual triple patterns in instantiated queries without taking traversal or query planning into consideration, 16 patterns out of 141 match most of the pods, while the remaining patterns match $\leq 0.20\%$ of all pods each. These 16 patterns are also found in a considerable share of the queries.

4. Brickley, D., Guha, R.: Rdf schema 1.1. W3C recommendation, W3C (Feb 2014), https://www.w3.org/TR/2014/REC-rdf-schema-20140225/
5. Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.D., Boncz, P.: The ldbc social network benchmark: Interactive workload. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 619–630 (2015)
6. Hartig, O.: Sparql for a web of linked data: Semantics and computability. In: Extended Semantic Web Conference. pp. 8–23. Springer (2012)
7. Hartig, O., Freytag, J.C.: Foundations of traversal based query execution over linked data. In: Proceedings of the 23rd ACM conference on Hypertext and social media. pp. 43–52 (2012)
8. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: A Modular SPARQL Query Engine for the Web. In: Proceedings of the 17th International Semantic Web Conference. pp. 239–255. Springer International Publishing (Oct 2018). https://doi.org/10.1007/978-3-030-00668-6_15
9. Taelman, R., Verborgh, R.: Link traversal query processing over decentralized environments with structural assumptions. In: International Semantic Web Conference. pp. 3–22. Springer (2023)
10. Verborgh, R.: Re-decentralizing the web, for good this time. In: Linking the World's Information: Essays on Tim Berners-Lee's Invention of the World Wide Web, pp. 215–230 (2023)