Aggregators to realize Scalable Querying across Decentralized Data Sources

Maarten Vandenbrande[1]

[1]IDLab, Ghent University – imec, Technologiepark 126, 9052 Ghent, Belgium

Abstract

The hyper-centralized nature of personal data has been getting some doubt due to recent data scandals. Decentralized ecosystems, like Solid, allow for users to take back control of their data by storing it in a data pod. This decentralization of data, together with the vision of Solid where everything has to be interoperable, causes the querying of this data to be slow. We propose to address this problem by introducing aggregators. Aggregators are a network of query and reasoning agents, each of which contribute (partial) results to a query by providing an up-to-date materialized view on the decentralized data.

Keywords

Decentralization, Link-Traversal, Incremental Querying

1. Introduction

The Big Data and analytics software market was worth $90 billion in 2021 and is set to double by 2026 [1]. Companies are increasingly relying on personal data to offer personalized experiences, but data scandals (e.g. Cambridge Analytica & Equifax) and legal limitations on the collection of personal data have led to an increasing demand of ***decentralized storage of personal data*** [2], where each person has their own personal data storage or pod. The key to sustainability of this decentralized vision is that every data pod provider implements a universally accepted standard to share this data. So each service provider can also adopt this standard and thus request and process data from any data pod provider, with the person's permission. The ***Solid*** specification [3] has large potential of becoming this standard. Solid's focus is to enable the discovery and sharing of information in a way that preserves privacy. Solid is built upon existing (Semantic) Web standards, as each pod stores data in the form of documents containing Linked Data (RDF) to allow for high interoperability between services. While the community is currently gaining momentum and rapidly realizing decentralized solutions for the storage and querying of data in Solid pods, ***fundamental research questions arise from a service provisioning viewpoint concerning the scalable and performant processing of all this decentralized data***. We tackle the problem from a querying perspective: a service provider has an information need, expressed as a query, and wishes it to be answered. For example, if *Agent 1* on the left of Fig. 1 wants to query social media posts on a certain topic, it would need to query all the available posts to find the solution, which is not feasible on a web scale. Also, decentralized data inherently suffers from data variety, as there exists no central authority that

ISWC: International Semantic Web Conference, November 06–10, 2023, Athens, Greece

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

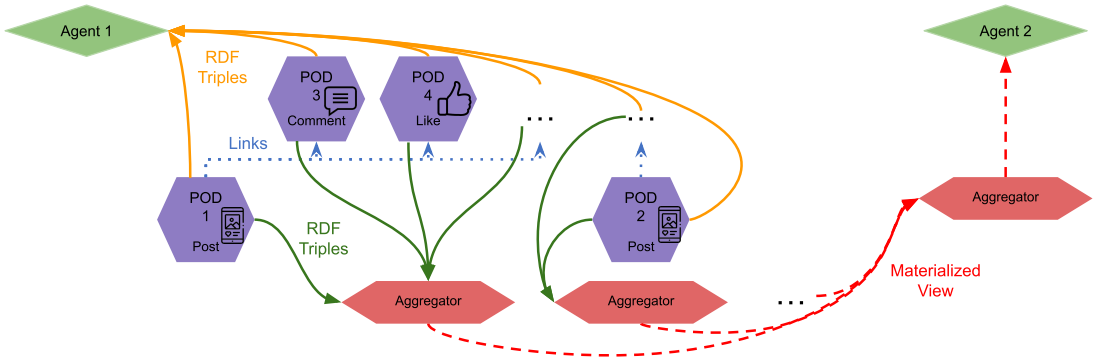

Figure 1: A decentralized social media use case with posts and comments.

can enforce one specific data format or schema. Meaning, no schema is enforced to describe this kind of social media data, so schema/ontology alignment through semantic reasoning is required to create a unified view on the variety of schema data. Semantic reasoning allows to automatically infer alignments between various schemas and hide this data variety to the applications using the data. However, semantic reasoning brings a computational overhead, adding to the challenge of generating the query results in a timely manner.

As such, tackling the service information need in a decentralized pod environment requires solving a federated query and schema alignment (reasoning) problem with a high number of independent and varied data sources, requiring more complex algorithms while having less computational power per node than centralized systems. We aim to address this problem by introducing aggregators. ***Aggregators are a network of query and reasoning agents, each of which contribute (partial) results to a query by providing an up-to-date, i.e. continuously maintained, materialized view on the underlying decentralized RDF data. The materialized view refers to SPARQL query results, which also includes results from data originally modelled using different ontologies than the ones used in the query.*** By using aggregators, a service information need could be quickly resolved by just retrieving the result from one or more aggregators. To achieve this, the changes in the underlying data need to be timely reflected into changes in the materialized view. This can be achieved by re-executing the query when changes occur, but to achieve proper scalability, incremental approaches are necessary. In decentralized systems, we can perform link-traversal [4], i.e. follow links to discover new sources that might contain relevant data, e.g. in Fig. 1 *Posts* specify links to the comments, which can be dereferenced to find the associated comment data on another pod. Performing link-traversal in an incremental query engine is more difficult as certain links might not contain any relevant data at the moment of fetching, but might in the future. Moreover, a link might be deleted, causing some links to become irrelevant in the future. To allow for optimal scalability, interoperability and re-use of the aggregators, it is preferable that they can be automatically discovered, both in terms of location and offered functionality. Two challenges arise with this, a central catalogue for all aggregators is infeasible at web scale, so a decentralized solution should be used. Second, discovering and using an aggregator to find the results of a query should be faster than executing the query itself.

In summary, to realize aggregators that enable scalable and efficient querying of decentralized data sources, the following challenges need to be tackled:
C-I: Design of incremental query algorithms that are able to query decentralized sources in an efficient and scalable fashion.
C-II: Facilitate complete query answers, regardless of decentralized storage that use different ontologies, by enabling decentralized schema alignment.
C-III: Increase decentralized query scalability by enabling automated discovery and use of the aggregator.

2. State of the Art (SotA)

The SotA on ***link-traversal*** is to the best of our knowledge non-existent for incremental systems. For non-incremental systems there exist 3 types of link-traversal strategies [5]. The *top-down* strategy first finds and collects the relevant sources and data before the query is executed. The *bottom-up* strategy immediately evaluates the relevant sources. This requires incremental SPARQL operators, as it is possible that some data might be added later, however, only additions need to be supported. Finally, the *hybrid* strategy combines both approaches by first collecting a number of sources, then executing the query for some first results, before gathering additional sources. This allows for both fast and up-to-date results [6].

Incremental view maintenance, for ***incremental query engines***, has been researched extensively in the field of relational databases. The most notable one is differential dataflow [7], which describes the logic behind differential operators. These operators determine the changes in the result set based on the changes of the input set. Incremental query engines for the Semantic Web have also been researched. Schmedding et al. [8] laid the foundation of incremental SPARQL operator logic. Most incremental query engines [9, 10, 11, 12] use the Rete algorithm to incrementally update the result set. Some of these do not consider deletions and only use the symmetrical joins in the Rete network to allow for a bottom-up link-traversal approach. This research lacs to consider the computation expenses that come with computing the changes between different resource versions.

For schema alignment backed by semantic reasoning, we focus on rule-based reasoning as rule-based reasoning is more scalable and typically aligns naturally with the definitions of business logic and thus also aligns naturally with the definition of alignment rules [13]. Three inference approaches exist for enabling ***schema alignment*** using rule-based reasoning: forward-chaining, backward-chaining and hybrid. To perform incremental forward-chaining also called materialization, there exist three main algorithms [**?**]: the counting algorithm, the Delete/Rederive (DRED) algorithm and the Forward/Backward/Forward (FBF) algorithm. They support both additions and deletions. Backward-chaining can be done in two ways, goal driven or query rewriting. The advantage of backward-chaining is that no intermediate results need to be explicitly stored, however, typically requiring a computational penalty to re-compute intermediate results more often in the goal driven case or the evaluation of very large queries in the case of query rewriting [14]. First steps towards hybrid approaches [14] have been set, that use backward-chaining to prune the rules, so only the necessary ones remain to perform forward-chaining materialization. However, these hybrid approaches have focused so far on

limited reasoning problems, i.e. the computation of class hierarchies. The combination of reasoning for schema alignment and link-traversal has not been researched extensively. Only in Umbrich, et al. [15] some lightweight reasoning is applied to increase the recall of the SPARQL query, so further research in this area is required.

For the SotA on ***discovery of the aggregator***, we can take inspiration from two fields of research. First, distributed hash table (DHT) protocols [16] are used in peer-to-peer systems to decentralize data and allow for timely lookup of that data. Each peer is responsible for a fraction of the hash table. To find an element in the network, the hash table can be used to find the location of the peer where the element is in. This idea can be used in an aggregator network where the elements are queries and the peers are aggregators. Second, approximate membership functions (AMFs) can determine if certain elements could be part of a dataset or if they will not be part of a dataset. As such, they could give false positives but will always return true negatives. Bloom Filters [17] are AMFs, as they map multiple hash functions to a bitmap. They can be used to more quickly find a corresponding aggregator for a certain query. In federated querying, they are already used to index RDF datasets to more quickly determine if a dataset is useful for a certain query [18]. Next, we can look into the SotA of the ***use of the aggregator***. After discovering a potentially useful aggregator, the research field of query containment [19] can allow for checking if and how it can be used to contribute to the query answers. Next, FedQPL [20] is a language for query plans using different heterogeneous endpoints. This would allow for defining query plans while using different aggregators and other endpoint like TPF, brTPF, SPARQL endpoints, etc.

In summary, although research has been conducted on incremental querying, many do not support deletions and none get the changes from decentralized sources, where they consider the computational expenses of calculating the changes. Furthermore, no incremental link-traversal exists that takes possible deletions of sources into account. Incremental reasoning is a well-established field, however more efficient hybrid approaches that could enable reasoning on the limited resources in the decentralized web and the combination with link-traversal have not been explored yet. Discovery techniques for RDF datasets exist, but additional research is needed on how these techniques can deal with more quickly discovering aggregators based on a query.

3. Problem Statement and Contributions

The overall goal is to investigate a discoverable network of aggregators and how they can efficiently maintain an up-to-date view on (partial) query results on decentralized RDF data that are possibly modelled by a variety of schemas that need to be automatically aligned. The following research questions (RQs) and hypotheses (H) will be tackled:

RQ-I: *Can incremental query evaluation in a decentralized environment allow for a lower latency in updating the materialized view compared to re-executing the query? (**C-I**)*

H-I: In 90% of cases, incremental query evaluation allows for a 50% decrease in latency when updating the materialized view.

RQ-II: *Can view maintenance in a decentralized environment incorporate changes (additions and deletions)? (**C-I**)*

H-II: Current link-traversal algorithms do not allow for both additions and deletions, redesigning these algorithms could enable decentralized view maintenance.
RQ-III: *Can rule-based reasoning (for schema alignment) be used to increase the recall of a query over decentralized sources when using link-traversal? (**C-II**)*
H-III: Rule-based reasoning for schema alignment can allow for a 70% increase in recall.
RQ-IV: *Can a yet to be discovered aggregator allow for a decrease in query execution time compared to local execution? (**C-III**)*
H-IV: Aggregators can allow for up to a 50% decrease in query execution time compared to local execution of a query.

Agent 2 in Fig. 1 shows how I envision aggregators to be used in the aforementioned social media use case. Multiple aggregators can work together to find the solution to the query. *Agent 2* can use the materialized view of the right aggregator, which builds on the result of all the other aggregators.

4. Methodology and Evaluation

Fig 2 shows the different envisioned components of the aggregator and will aid in explaining the function of different building blocks I aim to design during my research. The aggregator itself will be built by modifying and extending Comunica [21], this is a modular querying framework for decentralized RDF data. The modularity of Comunica allows for fast prototyping (like implementing incremental query evaluation) whilst already having a foundational query framework. Two benchmarks have been selected to evaluate the **RQ**'s. Firstly, the Train Benchmark [22] is a cross-technology performance benchmark build for continuous queries. It is a centralized benchmark that includes changes to the dataset for continuous or incremental query engines. Second, the Solidbench benchmark [23] is based on the LDBC data generator, it generates decentralized linked social media data. Solidbench currently only generates static data, which needs to be altered to be able to evaluate the incremental nature of the aggregator. All the social media data is currently also described using only one ontology, so for *RQ III* additional ontologies will need to be implemented into the benchmark.

In **RQ-I**, I will research incremental query operators for maintaining a materialized view on decentralized data **(C-I)**. This **RQ** will focus on the *Guarding* and *Incremental Query Engine* block

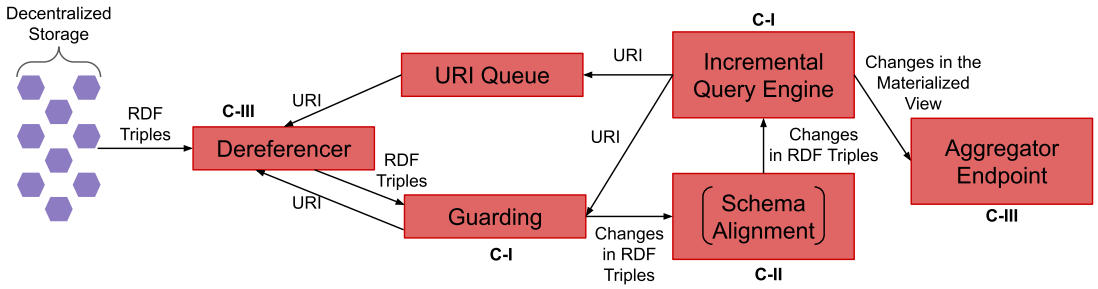

Figure 2: Aggregator architecture diagram.

in Fig 2, *Guarding* is the act of keeping the local state of the resources up to date and calculating the changes between resource versions. *Incremental Query Engine* will then use these changes to calculate the changes in the results. This allows us to compare the latency and memory consumption between re-evaluating the query versus maintaining it incrementally. Different *Guarding* and *Incremental Query Engine* techniques can then also be explored and quantified. Incremental computation will require the query engine to keep more state in the operators and will therefore use more memory, but it should have a lower latency compared to reevaluating the query. The *Incremental Query Engine* can be benchmarked with the Train Benchmark whilst the complete aggregator can be benchmarked with the Solidbench benchmark.

For **RQ-II** I will investigate techniques to perform link-traversal in an incremental query engine **(C-I)**. Here, I will focus on extracting links in the *Incremental Query Engine* that are then added to the *URI Queue*. To perform link-traversal incrementally, it will have to be deterministic. So it is guaranteed to end at some point [24] and the aggregator will not have to *guard* too many resources. Furthermore, the links that lead to other resources might be deleted and the newly found resources might therefore become irrelevant. I will develop algorithms and techniques to lower the amount of resources that are being *guarded* in the incremental query framework.

To answer **RQ-III**, research on how schema alignment (reasoning) can complement the incremental query algorithms is needed (**C-II**). Two approaches can be compared for implementing reasoning in the aggregator. A query rewriting approach should be able to make use of all the features of the query engine (incremental computations and link-traversal). A hybrid systems will require a reasoner that is able to update its resulting materialized triples when changes to the decentralized data occur, and extract links based on the chosen reachability-semantics of the link-traversal. These two approaches need to be compared and evaluated based on the amount of memory they use, the increase in recall of query results and the latency between changes in the decentralized data using the Solidbench benchmark.

Previously, we have made the assumption that the location and function of the aggregator is known. To answer **RQ-IV**, I will research how the aggregator can be discovered and used by other agents (client applications or other aggregators). A central catalogue of all aggregators is not scalable to the whole Web, so a solution needs to be investigated on how each pod can maintain a catalogue for all the aggregators that use its data. Once an agent has discovered the aggregator, it can check if it can use its result. Meaning, the aggregator should explain its materialized view (query, alignment rules, sources). I will design this decentralized catalogue and investigate different approaches to enable faster discovery of aggregators compared to actually performing the query itself using the Solidbench benchmark.

5. Preliminary or Intermediate Results

Two preliminary results have been accomplished:

Query redo: I designed a query engine that gives a continuous materialized view on given sources. This first naive implementation of an aggregator was realized as a wrapper around Comunica. This aggregator accepts queries and will execute them and cache and update the results on changes to the source data. Two main ways were investigated to check for changes, a polling strategy and a websocket strategy. As the whole query is re-evaluated when the data changes, it is not necessary to calculate the difference. This implementation of the aggregator

allowed for faster query results, but has a high latency when changes occur.

Incremental query engine: Research on implementing incremental query operators in Comunica has already begun. A naive way of *guarding* has already been implemented, together with a rudimentary incremental symmetric hash join. All used sources are constantly polled and if one changes it calculates the changes with a naive algorithm. The changes are then stored in a streaming store. This store indexes the new triples and updates subscribed triple pattern fragments if the new triple matches. The changes are then propagated to the query engine and result in the change in result set. At the moment, only an incremental symmetric hash join has been implemented.

6. Conclusions

The current paradigm shift to decentralized storage ecosystems will hit major limitations when these systems scale to the size of the current Web. I therefore propose aggregators to allow for greater scalability when querying over these large scale decentralized semantic data. These aggregators provide an up-to-date materialized view on top of this decentralized data to more quickly answer queries. In my research, I will be investigating if incremental query approaches are a possible solution to decrease the latency of these aggregators. Furthermore, I will explore link-traversal techniques when performing incremental queries. Next, I will be evaluating what types of reasoning techniques can be used to achieve schema alignment in these aggregators in a timely manner. Finally, I will make sure multiple aggregators can be used together to find a solution to a query. These aggregators will enable applications to get the results to complex queries in a timely manner and enable to scale the decentralized web.

Acknowledgments

Supervised by Femke Ongenae & Pieter Bonte. This research is partly funded by the SolidLab Vlaanderen project (Flemish Government, EWI and RRF project VV023/10) and the FWO Project FRACTION (Nr. G086822N).

References

[1] H. Ray, V. Dan, Worldwide big data and analytics software forecast, 2022-2026, IDC (2022).
[2] M. Maes, et al., Solidmonitor: privacy, persoonlijke data & datakluizen, 2023. URL: https://solidlab.be/wp-content/uploads/2023/01/IMEC_SOLIDMonitor_2022.pdf.
[3] The solid specification, 2022. URL: https://solidproject.org/TR/protocol.
[4] O. Hartig, M. T. Özsu, Walking without a map: Ranking-based traversal for querying linked data, in: The Semantic Web–ISWC 2016, Springer, 2016, pp. 305–324.
[5] G. Ladwig, T. Tran, Linked data query processing strategies., in: ISWC (1), 2010, pp. 453–469.
[6] J. Umbrich, M. Karnstedt, A. Hogan, J. X. Parreira, Hybrid sparql queries: fresh vs. fast results, in: The Semantic Web–ISWC 2012: 11th International Semantic Web Conference, Springer, 2012, pp. 608–624.

[7] M. Abadi, F. McSherry, G. D. Plotkin, Foundations of differential dataflow, in: Foundations of Software Science and Computation Structures: 18th International Conference, Springer, 2015, pp. 71–83.
[8] F. Schmedding, Incremental sparql evaluation for query answering on linked data., in: COLD, 2011.
[9] D. P. Miranker, R. K. Depena, H. Jung, J. F. Sequeda, C. Reyna, Diamond: A sparql query engine, for linked data based on the rete match, AImWD 2012 (2012) 7.
[10] X. Pu, J. Wang, P. Luo, M. Wang, Aweto: efficient incremental update and querying in rdf storage system, in: Proceedings of the 20th ACM international conference on Information and knowledge management, 2011, pp. 2445–2448.
[11] G. Szárnyas, B. Izsó, I. Ráth, D. Harmath, G. Bergmann, D. Varró, Incquery-d: A distributed incremental model query framework in the cloud, in: Model-Driven Engineering Languages and Systems: 17th International Conference, Springer, 2014, pp. 653–669.
[12] M. Rinne, E. Nuutila, S. Törmä, Instans: High-performance event processing with standard rdf and sparql, in: 11th International Semantic Web Conference ISWC, volume 914, Citeseer, 2012, pp. 101–104.
[13] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, J. Banerjee, Rdfox: A highly-scalable rdf store, in: The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Springer, 2015, pp. 3–20.
[14] P. Bonte, R. Tommasini, F. De Turck, F. Ongenae, E. D. Valle, C-sprite: efficient hierarchical reasoning for rapid rdf stream processing, in: Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, 2019, pp. 103–114.
[15] J. Umbrich, A. Hogan, A. Polleres, S. Decker, Link traversal querying for a diverse web of data, Semantic Web 6 (2015) 585–624.
[16] P. Felber, P. Kropf, E. Schiller, S. Serbu, Survey on load balancing in peer-to-peer distributed hash tables, IEEE Communications Surveys & Tutorials 16 (2013) 473–492.
[17] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (1970) 422–426.
[18] C. Aebeloe, et al., Decentralized indexing over a network of rdf peers, in: The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Springer, 2019, pp. 3–20.
[19] M. Spasić, M. V. Janičić, Soundness and completeness of sparql query containment solver specs, arXiv preprint arXiv:2210.07083 (2022).
[20] S. Cheng, O. Hartig, Fedqpl: A language for logical query plans over heterogeneous federations of rdf data sources, Association for Computing Machinery, 2021, p. 436–445.
[21] R. Taelman, J. Van Herwegen, M. Vander Sande, R. Verborgh, Comunica: a modular sparql query engine for the web, in: The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Springer, 2018, pp. 239–255.
[22] G. Szárnyas, B. Izsó, I. Ráth, D. Varró, The train benchmark: cross-technology performance evaluation of continuous model queries, Software & Systems Modeling 17 (2018) 1365–1393.
[23] R. Taelman, R. Verborgh, Evaluation of link traversal query execution over decentralized environments with structural assumptions, arXiv preprint arXiv:2302.06933 (2023).
[24] O. Hartig, Sparql for a web of linked data: Semantics and computability, in: The Semantic Web: Research and Applications: 9th Extended Semantic Web Conference, ESWC 2012, Springer, 2012, pp. 8–23.