Optimizing Write Performance in Decentralized Data Ecosystems

Jitse De Smet [0009-0002-6513-5013]

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

Abstract. There is growing interest in sharing self-governed, non-public data across organisational structures. In response, interest in decentralized data ecosystems increased, and systems like Solid, IDSA, and Gaia-X started being developed. These systems aim to boost data interoperability and allow authorized third parties to write to restricted sets of organisations' data. Decentralized data ecosystems are being developed to support various use-cases, each with different optimal data structures and interfaces. To create a unified data ecosystem, these systems must embrace data and interface heterogeneity. However, this heterogeneity complicates interactions with the data ecosystem, as data-consumers like developers and data analysts need to know how to interact with each interface. To address these complexities, data reading has been abstracted using declarative queries. Unfortunately, read abstraction solutions are not fully transferable to writing, even though writing is vital in a living data ecosystem. My PhD investigates how we can abstract writing over heterogeneous data sources. Tackling the fundamental challenges of abstracting complexities involving updates over decentralized data ecosystems which are permissioned, heterogeneous in both data and interface, decentralized, and have no central authority for managing updates. As a result, data-consumers will be able to interact with the data ecosystem without needing to know how to interact with individual interfaces. This research will accelerate the adoption of decentralized data ecosystems by significantly lowering the barrier towards creating write-based apps on top of them.

1. Introduction

Recently, the world has seen an increasing interest in sharing self-governed, non-public data across organisational structures. [1, 2, 3, 4]. Central to this development is the concept of decentralization, allowing organizations to collaborate and share data without relying on a central authority. Systems designed to cover these needs are called *decentralized data ecosystems*, or more generally, *decentralized environments*. Noteworthy efforts include, Solid [2] focussing on data interoperability between people, and both IDSA [3] and Gaia-X [4] focussing on corporate data interoperability. A critical feature of these systems is enabling authorized third parties to write to a restricted set of organisations' data, though research on writing is still in its infancy.

In the landscape of these decentralized data systems, the Semantic Web has proven instrumental due to its experience in public data sharing, the ability of the RDF [5] data model to create references across organisational boundaries, and its ability to model data semantics explicitly. The semantic web has a long history of distributing usable

open data. In that history, it became clear that serving data comes with various trade-offs [6, 7] that need to be considered for each use case. This has led to the creation of many interfaces like SPARQL endpoint [8], Linked Data Platform APIs [9], TPF [10], and others. A similar trend can be expected for decentralized environments, where diverse organizations with varied data converge. As such, embracing interface heterogeneity will be key for decentralized environments aiming for longevity, as it embraces future technological advances and allows data providers to make use-case-dependent choices.

Heterogeneity does, however, increase the complexity of interacting with the data ecosystem, since data-consumers, like developers and data analysts, need to know how to interact with each interface. For read-only SPARQL [11] federated queries over heterogeneous sources, this complexity has been mitigated by introducing a query engine serving as an abstraction layer over different interaction methods[12, 13]. Contrary to read abstractions, interaction abstraction is not sufficient for writing, since writes are not safe. For instance, when my dermatologist wants to send my test results to my general practitioner (GP) using a decentralized data ecosystem, my dermatologist would need to know exactly how to send this data over to my GP, requiring them to know how my GP expects to receive this data. This contrasts reading data, reading my test results is as easy as querying all accessible data, filtering on what applies to me. As such, the solution of abstracting reads is not entirely transferable to abstracting writes.

In this PhD, I will explore how to abstract writes over decentralized, heterogeneous, permissioned data sources. Specifically, I tackle the fundamental challenges of abstracting the complexities that arise from the unique characteristics of such systems, which 1. enable fine-grained permissions, 2. expose data through heterogeneous interfaces, and 3. have no central authority for managing updates. To abstract such heterogeneous interfaces, I will use the declarative SPARQL update [14] query language, which is the standard for expressing structured update queries over RDF data. Additionally, I develop scheduling algorithms for updates across decentralized, heterogeneous, and self-governed data stores, as existing algorithms are designed for *centralized* data stores or public data [15].

The next section discusses the state of the art. Section 3 explains the problem I am trying to solve, followed by Section 4, which outlines the methodology. Section 5 covers evaluation, Section 6 presents preliminary results, and Section 7 details the desired impact of this research.

2. State of the Art

The state-of-the-art covers both Semantic Web and querying research over decentralized data, and the more general database management research domains.

2.1. Polyglot Systems

A single dataset can be exposed through multiple interfaces, creating a polyglot data-

base [16]. A polyglot database allows data providers to serve multiple use-cases. Interfaces may differ in structure, functionality, and feature tradeoffs [6, 7]. For example, one interface may support transactions [17], while another does not. When the same data is exposed through multiple (partially overlapping) interfaces, it is essential that an agent can discover what data an interface exposes. Meaning interfaces should describe how they relate to each-other, by describing how they relate to the underlying data. From these descriptions, data consumers can identify which interfaces expose data of interest, and interact with the best interface based on properties like availability, presence of certain query accelerators (indices), and transaction support.

2.2. Decentralized Updates over Heterogeneous Interfaces

Updates across distributed DBMSs assume a homogeneous interface that is shared across nodes in the distributed data environment [18]. Within a decentralized data ecosystem, however, interfaces can be heterogeneous. This heterogeneity of interfaces affects fundamental concepts within transaction handling and update planning compared to distributed DBMSs. Since executing read SPARQL queries across heterogeneous interfaces [12, 13] is an active research area, we can combine the formal model for executing read SPARQL queries [11] over heterogeneous interfaces with the transaction model for DBMSs [18]. Concretely, this will involve introducing the concepts of interface capability discovery into update transactions, whereby interfaces describe their capabilities via hypermedia descriptions [19].

2.3. Scheduler

To convert declarative queries, which describe the desired result, to executable code, queries are first parsed and converted into a query plan [20]. To achieve performant plans, query planning relies on statistics like cardinality estimates [21]. While statistics are relevant for update scheduling, metrics like transactions size [22] are more critical. In polyglot or decentralized systems, the scheduler must also decide which interfaces to consult for what data. While SPARQL query planning over decentralized environments is well-studied [23, 24, 10], update scheduling is not. To achieve performant updates, there is a need for scheduling algorithms for decentralized environments.

2.4. Distributed Update Transactions

ACID safety guarantees, achieved through *transactions* [17], are foundational in DBMSs. However, the rise of distributed databases prompted reevaluation of the importance of these properties. According to the CAP theorem systems must trade off Consistency (C), Availability (A) and Partition Tolerance (P) on a continues scale [25]. Distributed databases typically require Partition Tolerance and are thus forced to choose between CP and AP, and consequently between ACID and BASE [26] properties, respectively focussing consistency or availability.
Most distributed databases under centralized management choose for the BASE prop-

erties, emphasize availability and settle for *eventual* consistency. While distributed DBMS transactions are well-studied, decentralized transactions are less understood [27]. Insights from distributed transactions do not translate well into decentralized contexts.
In distributed data management, one entity manages the entire network, each node thus has the same mission, and data is replicated across the network to ensure availability [18]. In contrast, decentralized data ecosystems consist of self-governed data stores, without systematic data replication or a unifying mission. Failure tolerance also differs: in a decentralized ecosystem, the unavailability of one organization's data does not compromise the entire ecosystem, much like how the web continues to function even when individual links are broken [28]. As such, decentralized ecosystems can be understood as a collection of loosely coupled, self governed datasets, each exposed through interfaces that individually position themselves in the CAP space. Consistency guarantees remain poorly defined in these environments, highlighting the need for novel transaction models and update algorithms tailored to decentralized ecosystems.

3. Problem Statement and Contributions

My research project focuses on abstracting data updates across a *large* number of *small* data stores, which are *permissioned*, *heterogeneous*, and *decentralized*. These characteristics exist in emerging decentralized data ecosystems [2, 3, 4], and contrast with distributed approaches that focus *on one or a few* data stores that are *large* and make use of centralized management techniques [29]. As such, I define my main research question as:

- **RQ I:** How to balance overall write throughput and server-side performance when updating data across a large network of permissioned, decentralized and heterogeneous RDF data stores?

I will focus my research on updating a small number of data stores (1-50) that are known at the start of query processing. I do this because the discovery of data stores is part of ongoing research [24] in the broader query processing research domain that goes beyond my project's scope. To achieve this, I will develop update query processing techniques with a focus on permissioned and heterogeneous data stores using hypermedia descriptions of the update capabilities in data stores discovered by client-side query engines. This leads to the following main hypothesis:

- **Hypothesis 1:** In a decentralized network of 50 permissioned, heterogeneous data stores, each containing up to 10,000 RDF triples, multiple concurrent SPARQL update queries can be executed without a central coordinator, producing the same final data states as with a central coordinator in distributed environments, while keeping execution time overhead limited ($\approx$ +100%).

This hypothesis addresses three open research challenges within Web decentralization and the Semantic Web. First, I focus on **updates in decentralized environments**,

while most research focuses on handling updates in a centralized datastore or distributed data stores with a centralized coordinator. This is challenging, as my research focuses on handling updates without such a centralized coordinator. Second, I focus on **permissioned** data stores, meaning data cannot be freely exchanged across data stores without prior authentication. This is challenging, as authentication is usually applied *outside* existing centralized and distributed algorithms, while I will incorporate this *within* update algorithms. Third, the data stores I focus on are **heterogeneous**, which means that different stores may expose varied update capabilities due to different client and server requirements. This is innovative, as heterogeneity in server capabilities has been primarily investigated for reading data, but not yet for writing data.
Section 1 and Section 2, highlighted interface heterogeneity as a core feature of a decentralized data ecosystem. Our research thus starts with the sub-question:

- **RQ II:** What are the requirements for update interfaces in decentralized environments, and how do they differentiate themselves?

Interface choice is case-dependent, with each interface having strengths and weaknesses. Interfaces will thus not only differ across data stores, a single data store could be exposed through multiple interfaces, serving as a polyglot system. As such, interfaces should describe their relation to the data, their interaction model, their functionality, and their feature tradeoffs. Where feature tradeoffs describes the trade-offs made in the interface design, for example, ‘maximum-lock-time’.

- **RQ III:** How can interfaces describe themselves sufficiently such that automated agents can interact with them?

Once a fundamental understanding of update interfaces is established, I will focus on updating a *single* data store. As each interface has pros and cons, a single data store may be exposed through multiple interfaces to support more use cases. This constitutes the following research questions:

- **RQ IV:** How can the scheduler discover the interfaces exposed by a single data store?
- **RQ V:** How can the scheduler select the interface most suitable for a given update query and user?
- **RQ VI:** How to schedule the execution of an update query over a single data store using the selected interface?

Finally, I will focus on updating *multiple* data stores.

- **RQ VII:** How can we schedule an update query over multiple data stores?

Additionally, I will investigate the support of ACID-transactions across multiple data stores. Supporting ACID-transactions will enable applications that require high safety guarantees to migrate to decentralized data ecosystems, transforming the ecosystems into a mature data layer. To reduce the workload of my PhD, I will focus on the atomicity property, because typical RDF does not have many consistency constraints.

- **RQ VIII:** How can we support the atomicity property of ACID-transactions

across multiple data stores?

4. Research Methodology and Approach

The work of this PhD can be split in 3 main parts, each providing answers to a subset of **RQ II-VIII**. We start with a discussion on the study of interfaces (**RQ II-III)**, after which we discuss query updates over a single data store (**RQ IV-VI**). Finally, we discuss updating multiple data stores (**RQ VII-VIII**). When all work is done, I will be able to provide an answer to **RQ I** in my dissertation.

4.1. Study of update interfaces in a personal decentralized data ecosystem

The functional choice of different interfaces (such as SPARQL [8], LDP [9], and TPF [10]) for different read/write granularity (SPARQL queries, arbitrary documents, triple patterns, etc), result in different non-functional characteristics, such as 1. permission semantics, 2. varying safety guarantees, 3. high availability, 4. internal transaction support, and 5. cross-interface transaction support. Additionally, interfaces also deviate based on the way they structure data while interacting with agents. They can be modelled 1. symmetrically, meaning the write unit is the same as the read unit, or 2. asymmetrically, for example being query-based or event-based.
To answer **RQ II and III**, I will investigate how this variety of interfaces can semantically describe themselves, describing: 1. their structure in relation to the data, 2. their supported features, and 3. the feature tradeoffs made. Previous research has mainly focused on the read structure of interfaces [30, 31, 32] and therefore lacks a description of why data is accessible like it is and how similar data can be made accessible in the future. Existing approaches to self-descriptiveness [19] can serve as a starting point in this process.

4.2. Update query processing against a single permissioned data store

To answer **RQ IV, V and VI**, I start by focussing on a single data store. Although decentralization emphasizes heterogeneity between data stores, understanding single-store updates is essential before scaling complexity.
The optimal interface choice depends on the query type and load [6, 7]. Since the interface choice is dependent on the expected consumption and a single data store might be consumed in various ways, it can be expected that a single data store could be exposed through multiple interfaces, serving diverse use cases. A single data store can thus be considered a polyglot system. As such, schedulers should discover the interfaces exposing the data-store, and select the most suitable interface for a given query and user. This choice can be based on the query type, expected load, and the functional and non-functional features of the interface, such as high availability, low server load, transaction support, or even dedicated functionality for certain data such as dedicated indexes.
After interpreting the interface descriptions of a data store and choosing the interfaces to work with, the scheduler dictates which operations will be executed in what order.

The schedule should be optimized based on the (non-)functional features of the interfaces, as well as using traditional optimize-then-execute techniques. For example, many insert operations can be grouped together to reduce the required number of requests. Instead of creating and optimizing one big group, it might also be worthwhile creating multiple groups, as to group operations that can be optimized on certain interfaces. The outcome of this task is at least one such scheduling algorithm that can handle updates across multiple interfaces in a single data store.

4.3. Scheduling across heterogeneous interfaces

Update queries across decentralized environments span multiple data stores, each of which could be exposed through multiple interfaces. If the creation of data across data stores has a clear order, a scheduler should be able to recognize it. For example, creating a calendar event and inviting others, involves first creating the agenda item in my data store and later sending invitations to the data stores of my invitees.
In decentralized data ecosystems, a single update query may cover different data stores. For instance, an event may refer only to a calendar item in one context, and to both the calendar item and the invitations in another. Updating the event in the latter context, requires a scheduler to update resources across multiple data stores. The outcome of this task should serve as an answer to **RQ VII**.
Although decentralized data ecosystem lack a shared mission across nodes, some nodes may support cross-data-store transactions as part of their interface choice. In the remainder of this PhD, I aim to provide an answer to **RQ VIII**: how can we support the atomicity property of ACID in a decentralized data ecosystem. Given the previous example, the atomicity property would assert that an update over the calendar item and invitations would only succeed if both the calendar item and all invitations accept the update.
Additionally, support for the atomicity property would be foundational to future research.

5. Evaluation Plan

While various distributed DBMS benchmarks exist [33], they are built for the SQL query language and assume homogeneous interfaces over relational databases. The SolidBench [24] benchmark for SPARQL queries over decentralized environments only focuses on read-queries over immutable data stores. SolidBench groups data into a single data store per person, exposing their social media data, like posts and comments. Since this domain model allows for modifications and creations, I will extend SolidBench [24] to include update queries over data stores that are designed to allow updating, while being heterogeneous in terms of data contained *within* data stores and update capabilities *across* data stores. This benchmark will include the following evaluation metrics: 1. update execution time, 2. number of HTTP requests, 3. number and scope of transaction locks, 4. recovery time of a transaction rewind, 5. inconsistent state durations, and 6. robustness against random server failures. Server failures

could be caused by the server rejecting access while our client thinks access is granted, or, the server becoming unreachable for a long period of time.

6. Preliminary Results

In prior work [34], I showed that update abstraction over Solid pods using LDP is feasible and can be implemented with limited overhead. I enabled automated agents to discover where new RDF-resources need to be created, where existing RDF-resources reside, and how to update them in a Solid pod. For this work, I interpreted Solid pods as a dataset being exposed by various HTTP-interfaces, while they are typically interpreted as a collection of HTTP-resources containing RDF-data. The outcome was the *Storage Guidance Framework* (SGF) and *Storage Guidance Vocabulary* (SGV). SGV generically describes the relation between RDF-resources in Solid pods and the HTTP interfaces exposing them. I then described the structure of Solid pods using this vocabulary, designed the algorithms for SGF, which describe how to consume and act according to the vocabulary. Lastly, I implemented a query engine using SGF. As a result of SGF, the engine could interpret a Solid pod as a collection of RDF data instead of a collection of documents containing RDF data, thereby eliminating the access path data dependency.
To measure the performance impact, I set up SolidBench-based Solid Pods with different fragmentation strategies. One strategy created a new HTTP-resource for each post based on the post's ID field. As such, modifying the post's ID required the post to move HTTP-resource. Fig. 1 shows the execution time for different operations, comparing SGF-based and oracle-based execution. In Fig. 1, operations O3 and O4 result in invalid states using the oracle implementation, as it does not detect required moves or invalid states. The original work concluded that the overhead of this abstraction is limited to four times the execution time, and twice the number of HTTP requests, compared to an oracle base approach that knows what HTTP-resource should be updated and performs the update directly.

Operation	Average Time SGF (ms)	Average Time Oracle (ms)
Insert complete post (O1)	91.851,395	76.672,217
Update post; no move (O2)	129.497,887	65.628,874
Update post; and move (O3)	177.991,908	80.729,940
Update post; not allowed (O4)	79.745,654	63.785,574
Delete post (O5)	130.769,232	69.931,699

Fig. 1: Average execution time (in ms) of various operations on a Solid pod with fragmented posts for SGF-based and oracle-based query execution over 100 runs.

This work provided insights into the challenges of abstracting updates over heterogeneous data sources, and the potential benefits of doing so. It will serve as a foundation for my PhD research, which will extend this work to more complex data ecosystems.

7. Conclusion

By the end of my PhD, I aim to have studied and developed query algorithms and techniques that allow an abstraction layer to update data across decentralized, heterogeneous, self-governed data stores. These advancements will allow data-consumers, like developers and data analysts, to interact with decentralized data ecosystems, while being shielded from the internal details of the underlying system. The increased ease of use will accelerate the adoption of decentralized data ecosystems.

Acknowledgements. Jitse De Smet is a predoctoral fellow of the Research Foundation – Flanders (FWO) (1SB8525N). Ph.D. advisors: dr. Ruben Taelman, and prof. dr. Ruben Verborgh.

References

1. 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. ACM (2023).
2. Verborgh, R.: Re-Decentralizing the Web, For Good This Time. In: Seneviratne, O. and Hendler, J. (eds.) Linking the World's Information: Essays on Tim Berners-Lee's Invention of the World Wide Web. ACM (2023).
3. International Data Spaces Association. https://internationaldataspaces.org/
4. Gaia-X. https://gaia-x.eu/
5. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C (2014).
6. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-Scale Querying through Linked Data Fragments. In: LDOW (2014).
7. Hartig, O., Letter, I., Pérez, J.: A formal framework for comparing linked data fragments. In: The Semantic Web–ISWC 2017. Springer (2017).
8. Feigenbaum, L., Todd Williams, G., Grant Clark, K., Torres, E.: SPARQL 1.1 Protocol. W3C (2013).
9. Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0. W3C (2015).
10. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., et al: Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web. Journal of Web Semantics. (2016).
11. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C (2013).
12. Heling, L., Acosta, M.: Federated SPARQL Query Processing over Heterogeneous Linked Data Fragments. In: Proceedings of the ACM Web Conference 2022 (2022).
13. 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. Springer International Publishing (2018).
14. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 Update. W3C (2013).

15. Aebeloe, C., Montoya, G., Hose, K.: ColChain: Collaborative linked data networks. In: Proceedings of the Web Conference 2021 (2021).
16. Khine, P.P., Wang, Z.: A review of polyglot persistence in the big data world. Information. 10, (2019).
17. Gray, J., McJones, P., Blasgen, M., Lindsay, B., Lorie, R., Price, T., Putzolu, F., Traiger, I.: The recovery manager of the System R database manager. ACM Computing Surveys (CSUR). 13, (1981).
18. Jentzsch, R.: Fundamentals of Database Systems. Journal of Database Management. 12, (2001).
19. Hydra. https://www.hydra-cg.com/spec/latest/core/
20. Hartig, O., Heese, R.: The SPARQL Query Graph Model for Query Optimization. In: European Semantic Web Conference. Springer (2007).
21. Acosta, M., Hartig, O., Sequeda, J.: Federated RDF query processing. (2019).
22. Shute, J., Vingralek, R., Samwel, B., Handy, B., Whipkey, C., Rollins, E., Oancea, M., et al: F1: A distributed SQL database that scales. (2013).
23. Hartig, O.: Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: Extended Semantic Web Conference. Springer (2011).
24. Taelman, R., Verborgh, R.: Evaluation of Link Traversal Query Execution over Decentralized Environments with Structural Assumptions. arXiv preprint arXiv:2302.06933. (2023).
25. Brewer, E.: CAP twelve years later: How the “rules” have changed. (2012).
26. Pritchett, D.: BASE: An ACID Alternative: In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability. Queue. 6, (2008).
27. Chen, H., Li, C., Zheng, C., Huang, C., Fang, J., Cheng, J., Zhang, J.: G-tran: a high performance distributed graph database with a decentralized architecture. Proceedings of the VLDB Endowment. 15, (2022).
28. Berners-Lee, T., Cailliau, R., Groff, J.-F., Pollermann, B.: World-Wide Web: the information universe. Internet Research. 2, (1992).
29. Ezéchiel, K.K., Kant, S., Agarwal, R.: A systematic review on distributed databases systems and their techniques. Journal of Theoretical and Applied Information Technology. 96, (2019).
30. Prud’hommeaux, E., Bingham, J.: Shape Trees Specification. (2021).
31. Turdean, T., Zucker, J., Balseiro, V., Capadisli, S., Berners-Lee, T.: Type Indexes. (2022).
32. Cyganiak, R., Zhao, J., Hausenblas, M., Alexander, K.: Describing Linked Datasets with the VoID Vocabulary. W3C (2011).
33. Zhang, H., Qu, L., Wang, Q., Zhang, R., Cai, P., Xu, Q., Yang, Z., Yang, C.: Dike: A Benchmark Suite for Distributed Transactional Databases. In: Companion of the 2023 International Conference on Management of Data (2023).
34. De Smet, J.: Abstracting Data Updates over a Document-oriented interface of a Permissioned Decentralized Environment (2024).