A Demonstration of ColChain: Collaborative Knowledge Chains*

Christian Aebeloe(✉)[0000−0003−3186−1607], Gabriela Montoya[0000−0001−5835−0335], and Katja Hose[0000−0001−7025−8099]

Aalborg University, Aalborg, Denmark
{caebel,gmontoya,khose}@cs.aau.dk

Abstract. The current architecture of the Semantic Web fully relies on the individual data providers to maintain access to their data and to keep their data up-to-date. While this may seem like a practical and straightforward solution, it often results in the data being unavailable or outdated. In this demo paper, we present a fully functioning client along with a user-friendly interface for ColChain, a system that increases availability of knowledge graphs and enables users to update the data in a community-driven way while still allowing them to query old versions.

1 Introduction

In recent years, the continuous advances of Semantic Web technologies have led to a rapid increase in the amount of data published as Linked Open Data. Naturally, the published information is subject to change and evolution [6]; errors are corrected, major updates are released, etc. However, the proliferation of data on the Web of Data and the fact that we currently rely on the data providers to maintain access to their datasets and keep them up-to-date represents a significant burden on the data providers [1, 9]. As a result, SPARQL endpoints often experience downtime [4, 8] and available data is sometimes outdated [3].

```
SELECT ?pr2 WHERE {
 dbr:President_of_the_United_States dbo:incumbent ?pr1 .
 ?pr1 dbo:party ?pa .
 ?pr2 dct:subject dbr:Category:Presidents_of_the_United_States .
 ?pr2 dbo:party ?pa
}
```

Listing 1: SPARQL query Q that finds former U.S. presidents of the same party as the current (incumbent) U.S. president.

Consider, for instance, query Q in Listing 1. Q finds all former U.S. presidents that have been a member of the same party as the current (incumbent) U.S. president. However, as of the writing of this paper, processing Q over the latest DBpedia release (version 2021-01)[1] results in `?pr1`, i.e., the current (incumbent)

[1] https://www.dbpedia.org/resources/latest-core/

president, being bound to `dbr:Donald_Trump` although the inauguration of President Biden took place months ago. While this is likely to be changed in the next release, the delay in the update shows that information available on the Semantic Web is not always up-to-date.

In this paper, we demonstrate a fully functioning client along with a user-friendly interface for ColChain (COLlaborative knowledge CHAINs) [3], a system that builds on unstructured Peer-to-Peer (P2P) networks [1] and uses replication of data fragments across several nodes to maintain high availability. Furthermore, ColChain enables users to collaboratively update datasets while also allowing users to process queries over previous versions. ColChain divides the P2P network into smaller *communities* of nodes that collaborate on keeping certain data up-to-date relying on community-wide consensus. Updates in ColChain are stored in blockchain-like chains; when a consensus for an update is reached, it is applied to the end of the chain. This allows any user to propose updates to any dataset while making malicious updates less likely. Furthermore, the update chains allow users to access previous versions of the datasets. While [3] presents the theoretical framework, this demo paper presents a working ColChain implementation with a user-friendly interface[2].

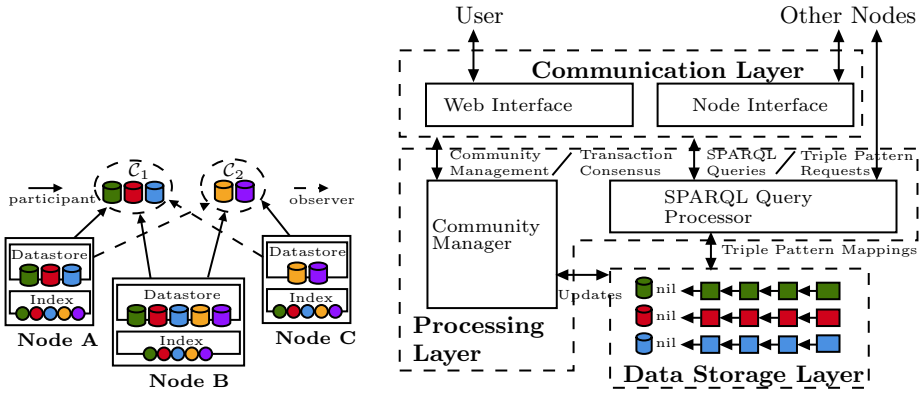
This paper is structured as follows. In Section 2, we present an architectural overview of ColChain while in Section 3 we describe the demonstration that will be conducted at the conference.

2 System Overview

Figure 1a shows an example of a ColChain network that consists of three nodes that store data from two communities, where the nodes either *participate* in or *observe* the communities. Participating nodes share a set of data fragments with the community they participate in and collaborate on keeping those fragments up-to-date. In Figure 1a, since node A participates in community $\mathcal{C}_1$, it stores $\mathcal{C}_1$'s fragments in its local datastore. Furthermore, A observes community $\mathcal{C}_2$, and thus only indexes $\mathcal{C}_2$'s fragments, relying on asking either node B or C (i.e., participants) to access $\mathcal{C}_2$'s data. Due to space restrictions, we do not go into details with aspects, such as how to create and maintain communities, but refer the interested reader to [3] for a more technically detailed description.

ColChain relies on the consensus of participating nodes within a community to apply updates collaboratively. In its current form, users of at least half the participants in a community have to actively accept the update using our interface. While this means applying a proposed update might entail an overhead on the validation, this is acceptable to let users ensure that updates are factual and non-malicious. Furthermore, as described in [3], our future work includes providing consensus protocols that make the active participation of users more scalable, e.g., by detecting malicious updates automatically or letting fragment owners specify a qualified majority.

[2] The client and source code are available at https://relweb.cs.aau.dk/colchain/



(a) Example ColChain network. (b) Architecture of a ColChain node.

Fig. 1: An example ColChain network and the architecture of a ColChain node (adapted from [3]).

2.1 Architecture of a ColChain Client

A ColChain node generally consists of several architectural layers as illustrated in Figure 1b These layers are as follows.

Communication Layer. The communication layer exposes two components: the Web interface and the node interface. The Web interface provides a GUI that allows users to interact with the system, e.g., to issue SPARQL queries (on current or previous versions of the data), propose updates, and decide whether to accept or reject updates proposed by other users. The node interface accepts messages from other nodes, e.g., when another participant accepts an update.

Processing Layer. The processing layer consists of two components: the community manager and the query processor. The community manager validates updates, and manages chains and fragments as well as community memberships. The query processor is able to process SPARQL queries over current and previous dataset versions available at any user-specified point in time.

Data Storage Layer. The data storage layer contains the node's local data store. ColChain nodes use HDT [5] as backend for storing data fragments. Changes to fragments are applied to the data storage layer by the community manager and appended to the chain for the given fragment. The data storage layer is used to process triple pattern requests by the SPARQL query processor. Furthermore, it can roll back fragments to earlier versions to allow users to process queries over those versions.

2.2 Graphical User Interface for ColChain

Consider again query Q from Listing 1 and a user who wants to suggest an update to obtain the expected result. Figure 2 shows how the user interacts with ColChain to propose an update over the corresponding

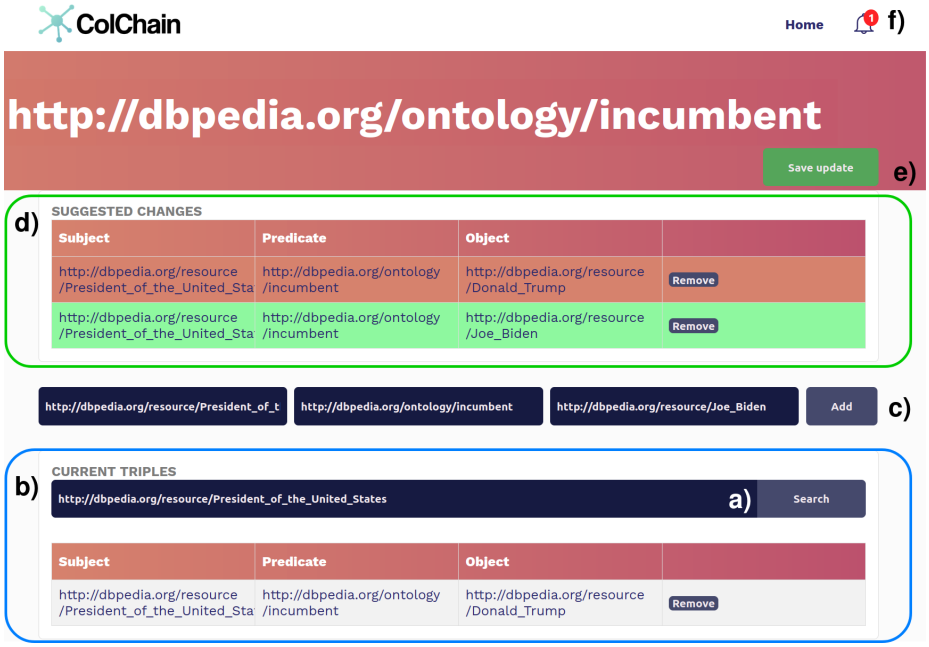


Fig. 2: ColChain's graphical interface when proposing an update to a fragment.

fragment with `dbo:incumbent` as the identifier. The user searches for the URI `dbr:President_of_the_United_States` (Figure 2a) and finds the triple with `dbr:Donald_Trump` as object (Figure 2b), which they then remove. The user then adds the triple with `dbr:Joe_Biden` as object to the fragment (Figure 2c). Figure 2d shows the changes made by the user. Once the user saves the update (Figure 2e), it is forwarded to the other participants in the community, which are notified (Figure 2f). Once a majority accepts the update, it is applied across the community, and the updated index is sent to the observers.

2.3 Implementation Details

ColChain is implemented in Java 8. The Web interface and node interfaces (Figure 1b) are implemented as Java 8 servlets using Jetty[3]. We implemented the query processor as an extension of Apache Jena[4], thus it can process any SPARQL query that Jena can process (e.g., queries with `UNION` or `OPTIONAL`). As previously mentioned, ColChain nodes use HDT [5] as backend for storing data fragments, as well as Prefix-Partitioned Bloom Filter (PPBF) indexes [2] to index the fragments available locally or remotely. The chains of updates are stored in persistent storage separately from the fragments. However, if possible, ColChain also stores the update chains temporarily in main memory.

[3] https://www.eclipse.org/jetty/
[4] https://jena.apache.org/

3 Demonstration

At the conference, we will demonstrate ColChain using two scenarios that attendees can explore and interact with. We will run a network with the data from LargeRDFBench [7], which comprises 13 interlinked datasets with over a billion triples in total. Furthermore, we will run a separate network with data from a subset of DBpedia that includes update chains back to version 2015-04, i.e., attendees will have the opportunity to explore query answers over different versions of DBpedia. ColChain will be showcased using networks with varying numbers of nodes and community sizes that follow different distributions (e.g., Zipfian as in [3]). A video demonstration of ColChain using the DBpedia scenario is available on our website[5].

To ease interaction with the system, we will provide several interesting SPARQL queries for attendees to explore each scenario. Attendees will be invited to build upon these queries, formulate queries on their own, explore query answers over different versions, and propose updates. For instance, attendees could propose the update shown in Figure 2. Query Q from Listing 1 could then be processed over the updated data as well as over DBpedia version 2015-04 when `?pr1` would be bound to `dbr:Barack_Obama`.

Acknowledgments. This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B, Aalborg University's Talent Programme, and the Poul Due Jensen Foundation.

References

1. Aebeloe, C., Montoya, G., Hose, K.: A Decentralized Architecture for Sharing and Querying Semantic Data. In: ESWC 2019. pp. 3–18 (2019)
2. Aebeloe, C., Montoya, G., Hose, K.: Decentralized Indexing over a Network of RDF Peers. In: ISWC 2019. pp. 3–20 (2019)
3. Aebeloe, C., Montoya, G., Hose, K.: ColChain: Collaborative Linked Data Networks. In: WWW 2021. pp. 1385–1396 (2021)
4. Aranda, C.B., Hogan, A., Umbrich, J., Vandenbussche, P.: SPARQL Web-Querying Infrastructure: Ready for Action? In: ISWC 2013. pp. 277–293 (2013)
5. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). J. Web Semant. **19**, 22–41 (2013)
6. Pelgrin, O., Galárraga, L., Hose, K.: Towards Fully-fledged Archiving for RDF Datasets. Semantic Web (2021)
7. Saleem, M., Hasnain, A., Ngomo, A.N.: LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation. vol. 48, pp. 85–125 (2018)
8. Vandenbussche, P., Umbrich, J., Matteis, L., Hogan, A., Aranda, C.B.: SPARQLES: monitoring public SPARQL endpoints. Semantic Web **8**(6), 1049–1065 (2017)
9. Verborgh, R., Sande, M.V., Hartig, O., Herwegen, J.V., Vocht, L.D., Meester, B.D., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. J. Web Sem. **37-38**, 184–206 (2016)

[5] https://relweb.cs.aau.dk/colchain#demonstration