ConVer-G: Concurrent versioning of knowledge graphs

Jey PUGET GIL, Emmanuel Coquery, John Samuel, Gilles Gesquière

ConVer-G: Concurrent versioning of knowledge graphs
ConVer-G : Versionnement concurrent de graphes de connaissances

Jey Puget Gil
jey.puget-gil@liris.cnrs.fr
Université Claude Bernard, LIRIS, UMR-CNRS 5205
Villeurbanne, FRANCE

Emmanuel Coquery
emmanuel.coquery@univ-lyon1.fr
Université Claude Bernard, LIRIS, UMR-CNRS 5205
Villeurbanne, FRANCE

John Samuel
john.samuel@cpe.fr
Université de Lyon, CPE Lyon, LIRIS, UMR-CNRS 5205
Villeurbanne, FRANCE

Gilles Gesquière
gilles.gesquiere@univ-lyon2.fr
Université de Lyon, Université Lumière Lyon 2, LIRIS, UMR-CNRS 5205
Villeurbanne, FRANCE

ABSTRACT

The multiplication of platforms offering open data has facilitated access to information that can be used for research, innovation, and decision-making. Providing transparency and availability, open data is regularly updated, allowing us to observe their evolution over time.

We are particularly interested in the evolution of urban data that allows stakeholders to better understand dynamics and propose solutions to improve the quality of life of citizens. In this context, we are interested in the management of evolving data, especially urban data and the ability to query these data across the available versions. In order to have the ability to understand our urban heritage and propose new scenarios, we must be able to search for knowledge through concurrent versions of urban knowledge graphs.

In this work, we present the ConVer-G (Concurrent Versioning of knowledge Graphs) system for storage and querying through multiple concurrent versions of graphs.

RÉSUMÉ

La multiplication de plateformes offrant des données ouvertes a permis de faciliter l'accès à des informations pouvant être utilisées pour la recherche, l'innovation et la prise de décision. Apportant transparence et disponibilité, les données ouvertes sont mises à jour régulièrement, nous pouvons alors observer leurs évolutions à travers le temps.

Nous nous intéressons plus particulièrement à l'évolution des données urbaines qui permet à des acteurs de mieux comprendre les dynamiques et de proposer des solutions pour améliorer la qualité de vie des citoyens. C'est dans ce contexte que nous nous intéressons à la gestion des données urbaines et à la nécessité de pouvoir interroger ces données à travers les versions disponibles. Afin d'avoir la capacité de comprendre notre héritage urbain et de proposer de nouveaux scénarios, nous devons être en mesure de chercher de la connaissance à travers des versions concurrentes des graphes de connaissances urbaines.

À travers ces travaux, nous présentons le système ConVer-G (Versionnement Concurrent de Graphes de Connaissances) permettant le stockage et l'interrogation de multiples versions concurrentes de graphes.

CCS CONCEPTS

• **Information systems** → Geographic information systems; **Resource Description Framework (RDF)**.

KEYWORDS

RDF, versioning, concurrent versioning, graph, urban data, query

MOTS CLÉS

RDF, versionnement, versionnement concurrent, graphe, données urbaines, requête

1 INTRODUCTION AND MOTIVATION

Urban environments, with their complex and ever-changing nature, particularly benefit from open data. Effective urban management requires a deep understanding of various dynamic elements, such as population growth, infrastructure development, and transport frequentations. Open data allows urban planners, policymakers, and researchers to gain insights into these dynamics, fostering better planning and decision-making processes.

Transforming 3D geospatial urban data to a knowledge graph [11] enables the representation of complex urban environments in a structured and semantically rich manner. Concurrent viewpoints of urban evolution can be captured through the versioning of these knowledge graphs. Moreover, this versioning allows the representation of different states of the urban environment over time, providing a comprehensive view of the city's evolution.

The versioning of scientific data also represents a certain interest in order to be able to exploit them[6]. Indeed, it's crucial to be able to reproduce experiments and analyses. If the data changes without versioning, it becomes difficult to reproduce the results at a later date. Versioning can help to protect the integrity of the data. If an error is introduced into the data, having versioned backups allows the data to be restored to a previous correct state. Versioning allows them to easily switch between different versions of the data for these experiments.

40ème conférence sur la gestion des données (BDA), October 21-24, 2024, Orléans, France 2024.

In this paper, we present a knowledge graph loader component (QuaDer[1]) and a versioned graph query system (QuaQue[2]). We have implemented demonstration scenarios to show the feasibility of our approach. The first scenario uses a real dataset representing a district in Villeurbanne called "Gratte-Ciel" to query the knowledge graph using versioned SPARQL queries. The second scenario uses a synthetic dataset to query a knowledge graph with a large number of versions. We aim to show that our system can query a large number of versions simultaneously and that the query response time remains acceptable.

2 RELATED WORK

As mentioned by Bayoudhi et al. [3], versioning is a critical aspect of management in a project. It allows developers to track changes to the elements, manage different versions, and collaborate with other team members.

2.1 Code versioning

Version control systems [12] such as Git, Mercurial, Subversion, CVS, and Bazaar are commonly used for managing the evolution of source code. However, these systems are not specifically designed for data versioning, especially for structured data and have certain limitations. Apart from exceptions, like Git LFS, they don't provide adequate support for large files and data query. Therefore, they may not be the ideal choice for managing data versioning in a project.

2.2 Dataset versioning

2.2.1 Snapshot-based versioning. There are several versioning tools available for managing data in a project. Some popular Git-based tools (snapshot-based versioning tools) like Qri[3], QuitStore[2], GeoGig[4], and UrbanCo2Fab[9] represent versions as snapshots. These tools provide version control capabilities specifically designed for data versioning. However, one limitation of these tools is that in order to query a version, it needs to be checked out. This limits the capacity of such systems to answer queries over multiple versions at the same time. Despite this limitation, database versioning tools provide valuable functionality for managing data in a project.

On the other hand, there are also machine learning tools like DVC[8], DagsHub[5] and Weights and Biases[6] that offer advanced features for managing data versions. These tools are particularly useful for structured data and provide functionalities such as tracking changes, managing different versions, and collaborating with team members.

2.2.2 Interval-based versioning. Another approach to data versioning involves using temporal tables or bitemporal tables [7] to represent the validity of data over time. These tables store the start and end timestamps for each version of the data, allowing for querying the data at specific points in time or in a specific time interval. However, this approach has limitations, in particular the lack of support for easily identifying and querying concurrent versions.

2.2.3 Delta-based versioning. Delta-based versioning systems like Delta Lake, Apache Hudi and OSTRICH (Offset-enabled TRIple store for CHangesets) [10] provide advanced features for managing data versions. These systems store the changes made to the data as deltas, allowing for efficient querying of the data at different versions. They also provide functionalities such as time travel, which allows users to query the data at specific points in time.

3 USE CASE AND CONTRIBUTIONS

3.1 The problem

To avoid data redundancy in a quickly changing world, where each change varies little from one state to another, we must condense the stored evolution. This is often necessary for urban datasets at larger scales. The problem we aim to solve with QuaQue is the efficient querying of concurrent versioning of quad data. Traditional quad stores are designed to efficiently store and retrieve static RDF data, but they struggle when it comes to handling concurrent versioning of data and more specifically multiple versions at the same time. Concurrent versioning of quad data refers to the ability to store and query multiple versions of the same dataset at the same time. This is particularly useful in the context of urban data, where multiple versions of the same dataset can represent different points of view or different states of the data over time.

In the RDF model, a triple is a statement that consists of a subject and an object connected by a predicate. A quad is an extension of the triple that includes a fourth element, a named graph. A named graph can be represented by a set of triples sharing the same graph name. In order to version a named graph, we associate a version with this set of quads. In SPARQL, the **graph** operator is used to specify a graph name. SPARQL does not provide a way to specify a version of a named graph. We do not want to create a new operator to specify a version of a named graph because it would break the compatibility with existing SPARQL queries. Instead we slightly change the semantics of the graph operator to specify a versioned named graph. We created URI identifiers for each versioned named graphs and store them as metadata. The versioned graph identifier allows us to identify the (graph name, version) pair of a versioned named graph. By associating a variable with it, the graph operator allows us to query the set of versioned graphs.

3.2 RDF Context Representation

3.2.1 Dataset representation. Consider our RDF context containing data as well as its associated metadata. We define a dataset as a set of versioned named graphs and metadata. The metadata is stored in the default graph. $d = (M, \{((v_1, t_{g1}), G_{1,1}), ..., ((v_m, t_{gn}), G_{m,n})\})$ is a dataset where:

- we call a versioned graph $G_{m,n}$ a finite subset of triples that occurs at the version v_m of the name graph t_{gn},
- d represents a pair containing the metadata M and a set of versioned graphs $\{((v_1, t_{g1}), G_{1,1}), ..., ((v_m, t_{gn}), G_{m,n})\}$,
- d has m versions and $(v_1, ..., v_m)$ is the set of versions,
- d has n named graphs and $(t_{g1}, ..., t_{gn})$ is the set of named graphs.

[1] Quads loaDer
[2] Quads Query
[3] https://qri.dev/
[4] https://geogig.org/
[5] https://dagshub.com/
[6] https://wandb.ai/

Example 3.1. Let's assume that we have a dataset representing concurrent points of view about the height of some buildings. In the table 1 we present two versions with the following quads:

Table 1: Dataset with 2 versions and 2 named graphs

Version	Subject	Predicate	Object	Named graph
1	ex:bldg#1	height	10.5	ng:Gr-Lyon
	ex:bldg#2	height	9.1	ng:Gr-Lyon
	ex:bldg#1	height	11	ng:IGN
2	ex:bldg#1	height	10.5	ng:IGN
	ex:bldg#1	height	10.5	ng:Gr-Lyon
	ex:bldg#3	height	15	ng:Gr-Lyon

Flat model. The flat model is a classic representation, it associates each quad with a version where it occurs.

$d = (G, \{(t_{g1}, G_1), ..., (t_{gn}, G_n)\})$ is a dataset where:

- G is the default graph storing the metadata
- G_i is a named graph storing the quads

Example 3.2. After some quads transformations, the Table 2 represents *the flat model* of the dataset versioning.

4 QUAQUE: A QUERYABLE VERSIONED QUAD STORE

4.1 The query engine

The problem we aim to solve is the efficient querying of concurrent versioning of quad data. Traditional quad stores are designed to efficiently store and retrieve static RDF data, but they struggle when it comes to handling concurrent versioning of data and more specifically multiple versions at the same time. Concurrent versioning of quad data refers to the ability to store and query multiple versions of a dataset. This is particularly useful in the context of urban data, where multiple versions of the same dataset can represent different points of view or different states of the data over time.

Table 2: Flat model of the dataset versioning

Subject	Predicate	Object	Named graph
Versioned quads			
ex:bldg#1	height	10.5	vng:1
ex:bldg#2	height	9.1	vng:1
ex:bldg#1	height	11	vng:2
ex:bldg#1	height	10.5	vng:3
ex:bldg#3	height	15	vng:3
ex:bldg#1	height	10.5	vng:4
Metadata			
vng:1	is-version-of	ng:Gr-Lyon	
vng:1	is-in-version	v:1	
vng:2	is-version-of	ng:IGN	
vng:2	is-in-version	v:1	
vng:3	is-version-of	ng:Gr-Lyon	
vng:3	is-in-version	v:2	
vng:4	is-version-of	ng:IGN	
vng:4	is-in-version	v:2	

Table 3: Condensed model of the dataset versioning

Versioned quads				
Subject	Predicate	Object	Named graph	Versions
ex:bldg#1	height	11	ng:IGN	10
ex:bldg#1	height	10.5	ng:IGN	01
ex:bldg#3	height	15	ng:Gr-Lyon	01
ex:bldg#2	height	9.1	ng:Gr-Lyon	10
ex:bldg#1	height	10.5	ng:Gr-Lyon	11

The challenge with concurrent versioning of quad data is that querying it requires considering the temporal, the concurrence and the structural aspects of the data. For example, we may want to retrieve all quads that were valid during a specific time interval or find the all versions where a quad matches certain criteria.

4.2 Architecture

The architecture of ConVer-G consists of three main components: the quads loader (QuaDer), the query engine (QuaQue), and the storage (PostgreSQL).

QuaDer is a loader component that is responsible for the versioning of quad data. It stores the different versions of the dataset in a condensed form, allowing for efficient storage of the data. QuaDer uses a relational database to store the different versions of the dataset. The system uses a bitstring representation to store the presence of quads in a set of versions. For each new version of the dataset, QuaDer adds a new bit to the bitstring representing the presence of the quad in that version. If the quad is present in the version, the bit is set to 1, otherwise it is set to 0.

Example 4.1. The Table 3 represents the condensed model of the dataset versioning.

QuaQue is a queryable versioned quad store that is designed to query concurrent versioning of quad data. It is built on top of a Jena Fuseki edited version and uses a relational database to store and query multiple versions of the same dataset at the same time. QuaQue uses the previous relational database to query the different versions of the dataset at the same time.

4.3 Demonstration scenarios

We conducted a series of experiments to evaluate the ability of QuaQue in terms of query translation and scalability. We used two datasets and ran a set of queries against them:

- a urban dataset with multiple versions representing the evolution of a city over time with concurrent versioning of quad data (the urban aspect of the dataset is not important for this paper). The dataset contains around 430000 quads repartitioned in 6 versioned graphs and 15000 metadata triples.
- a synthetic dataset created with BSBM [4]. This dataset is composed of more than 175000000 quads reparted in 1000 versioned graphs.

We ran two types of queries against the datasets. The first type of queries are non aggregative queries that retrieve all quads that match certain criteria. The second type of queries are more complex queries that involve aggregation.

4.3.1 Non aggregative queries. We ran a set of non aggregative queries against the datasets to evaluate the ability of QuaQue to retrieve all quads that match certain criteria. The queries involve using the bitstring representation to retrieve the quads that match certain criteria. To compute the existence of a quad in a version, we use the bitwise AND operation between all the bitstrings and the result is the bitstring representing the presence of the quad in the versions.

Query 1: Retrieve all resources and versions where a triple matches certain criteria

```
SELECT ?version ?subj ?obj WHERE {
    GRAPH ?vng { ?subj rdf:type ?obj . }
    ?vng vers:is-in-version ?version .
}
```

This query is equivalent to the flat model. It retrieves all versioned quads and their associated version. It is useful to test the ability of QuaDer to insert correctly the versioned quads and the ability of QuaQue to retrieve them with their associated version.

Query 2: Create the differences graph between two versioned graphs

```
SELECT ?subj ?pred ?obj WHERE {
{ SELECT ?subj ?pred ?obj WHERE {
    GRAPH <vng1> { ?subj ?pred ?obj . }
} } MINUS {
    SELECT ?subj ?pred ?obj WHERE {
    GRAPH <vng2> { ?subj ?pred ?obj . }
} } }
```

This query tests the ability of QuaQue to get the differences between two versioned graphs. It retrieves all quads that are present in the first versioned graph but not in the second versioned graph. It is useful to test the ability of QuaQue to compute the differences between two versioned graphs and compare the result with other versioned triple store like OSTRICH.

4.3.2 Aggregative queries. We ran a set of aggregative queries against the datasets to evaluate the ability of QuaQue to perform complex queries that involve aggregation.

Query 3: Find the maximum value of a resource by version

```
SELECT ?version MAX(?o) WHERE {
    GRAPH ?vng {
        ?s bsbm:v01/vocabulary/rating2 ?o .
    }
    ?vng vers:is-in-version ?version .
} GROUP BY ?version
```

Query 4: Count the number of elements that match certain criteria by version

```
SELECT ?version COUNT(?subj) WHERE {
    GRAPH ?vng { ?subj rdf:type ?obj . }
    ?vng vers:is-in-version ?version .
} GROUP BY ?version
```

Query 5: Count the number of elements that match certain criteria by graph

```
SELECT ?graph COUNT(?obj) WHERE {
    GRAPH ?vng { ?subj rdf:type ?obj . }
    ?vng vers:is-version-of ?graph .
} GROUP BY ?graph
```

Query 6: Count the number of version that match certain criteria by graph

```
SELECT ?graph COUNT(DISTINCT ?version) WHERE {
    GRAPH ?vng { ?subj rdf:type "sensor" . }
    ?vng vers:is-in-version ?version ;
         vers:is-version-of ?graph .
} GROUP BY ?graph
```

These queries allow the analysis of the impact of the bit vector for the optimization of SPARQL queries. The work of Sarah Cohen [5] allows us to rewrite queries with arbitrary aggregation functions using views. In our case, we then use the bit vector to optimize the computation of aggregation functions. For example, to compute the sum of values for a resource by version, we use the bitwise sum operation between the bit vectors representing the count values of the resource in each version (e.g., *"What's the sum of the height of all buildings by version?"*).

4.4 Discussion

4.4.1 Bitstring use. The bitstring representation is a key feature of QuaDer that allows for efficient storage and retrieval of versioned quad data. By using a bitstring to represent the presence of quads in a set of versions, we use the GRAPH operator to get all graph's versions that satisties a basic graph pattern by performing a bitwise AND operation between the bitstring of all satisfied quads. This is due to the condensed representation. The more we are able to keep the condensed representation, the more we are able to optimize the query translation. In future work, we plan to investigate the use of the bitstring representation to optimize the translation of more complex queries, such as aggregation and sorting.

4.4.2 Link with metadata. The metadata is stored in the default graph. The metadata contains information about the dataset, such as the versions and named graphs. The metadata is used to associate a versioned named graph with its version (using the *is-in-version* property) and named graph (using the *is-version-of* property). We can also add metadata about the dataset, such as the creation date, the author, and the description. By storing the metadata in the default graph, we can easily access it using the basic graph pattern without GRAPH operator. This allows us to retrieve the metadata and use it to query the different versions of the dataset.

4.4.3 Translation extension. QuaQue partially implements the translation of SPARQL operators. We have only implemented the translation for the basic operators, such as SELECT, GRAPH, JOIN, GROUP BY and basic graph patterns. This choice was made to demonstrate the feasibility of the approach and to evaluate the capabilities of the condensed model on complex queries such as aggregation.

Covering all the SPARQL operators would greatly enhance the capabilities of QuaQue as a query engine. It would enable researchers and data scientists to perform complex queries and analysis on concurrent versioned quad data. By implementing the translation for all SPARQL operators, QuaQue would be able to handle a wide

range of query types, including filtering, aggregation, sorting, and joining. This would allow users to express their queries in a familiar and expressive language, making it easier to explore and analyze the data. Additionally, extending the translation to all SPARQL operators would make QuaQue compatible with existing SPARQL query tools and libraries, enabling seamless integration with other data processing and analysis pipelines.

4.4.4 Configuration of the versioning representation. Extending the implementation of QuaDer to handle a target representation such as RDF-star, Java, relational database, or property graphs would allow to understand which storage engine would be the more efficient to manage condensed model. By incorporating support for these additional representations, QuaDer would offer researchers and data scientists the flexibility to choose the most suitable representation for their specific use cases and requirements.

Incorporating these target representations into QuaDer would not only enhance its versatility but also contribute to the scientific community by providing a comprehensive and extensible platform for concurrent versioning of quad data. Researchers and practitioners would have the freedom to choose the most appropriate representation based on their specific needs, leading to more accurate and efficient analysis of evolving datasets.

4.4.5 RDF graphs annotation. RDF graphs are a suitable choice for representing heterogeneous data. They provide a flexible and standardized way to express relationships between entities. Additionally, RDF-star[7] and Property Graphs [1] offer advanced features for representing complex data structures. However, it's important to note that the representation of data in RDF graphs can be complex, which may introduce some limitations. We can use the RDF-star representation to annotate the RDF graphs with versions. This would allow us to query the different versions of the dataset.

5 CONCLUSION

In this chapter, we have explored the capabilities of QuaQue, which contains alternative SPARQL operators. QuaQue offers a new approach to querying evolving graphs by providing an alternative to classic SPARQL queries.

With QuaQue, we can achieve everything that a classic triple store can do. QuaQue modifies a set of operators and functions that allow us to query all versions of a graph. This means that we can retrieve historical data and analyze the evolution of the graph over *transaction* time. This feature is particularly useful in scenarios where data changes frequently and we need to track the changes and perform analysis on different versions of the graph.

In future research, it would be interesting to integrate an hybrid approach that combines transaction versioning and existence versioning (where transaction versioning is used to track the changes in the database and existence versioning is used to track the changes of the entities in the real world). This would allow us to query the graph at a specific point in time and retrieve the historical data. This would be particularly useful in scenarios where we need to analyze the evolution of the graph over time and perform analysis on different versions of the graph.

[7] https://www.w3.org/2021/12/rdf-star.html

ACKNOWLEDGEMENTS

This work, titled *ConVer-G: Concurrent versioning of knowledge graphs*, is supported and funded by the IADoc@UDL (Université de Lyon, Universite Claude Bernard Lyon 1) and LIRIS UMR 5205. We would like to express our gratitude to the BD team and the members of the Virtual City Project[8] for their invaluable advice and assistance.

REFERENCES

[1] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. 2019. RDF and Property Graphs Interoperability: Status and Issues. *AMW* 2369 (2019), 1–11.
[2] Natanael Arndt. 2020. *Distributed Collaboration on Versioned Decentralized RDF Knowledge Bases.* Ph. D. Dissertation. Universität Leipzig. https://doi.org/10.33968/9783966270205-00
[3] Leila Bayoudhi, Najla Sassi, and Wassim Jaziri. 2020. A survey on versioning approaches and tools. In *International Conference on Intelligent Systems Design and Applications*. Springer, 1155–1164.
[4] Christian Bizer and Andreas Schultz. 2009. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)* 5, 2 (2009), 1–24.
[5] Sara Cohen, Werner Nutt, and Yehoshua Sagiv. 2006. Rewriting queries with arbitrary aggregation functions using views. *ACM Transactions on Database Systems (TODS)* 31, 2 (2006), 672–715.
[6] Jens Klump, Lesley Wyborn, Mingfang Wu, Julia Martin, Robert R Downs, and Ari Asmi. 2021. Versioning data is about more than revisions: A conceptual framework and proposed principles. *Data Science Journal* 20, 1 (2021), 12.
[7] Krishna Kulkarni and Jan-Eike Michels. 2012. Temporal features in SQL:2011. *SIGMOD Rec.* 41, 3 (oct 2012), 34–43. https://doi.org/10.1145/2380776.2380786
[8] Philipp Ruf, Manav Madan, Christoph Reich, and Djaffar Ould-Abdeslam. 2021. Demystifying mlops and presenting a recipe for the selection of open-source tools. *Applied Sciences* 11, 19 (2021), 8861.
[9] John Samuel, Sylvie Servigne, and Gilles Gesquiere. 2018. Urbanco2fab: comprehension of concurrent viewpoints of urban fabric based on git. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4 (2018), 65–72.
[10] Ruben Taelman, Miel Vander Sande, Joachim Van Herwegen, Erik Mannens, and Ruben Verborgh. 2019. Triple storage for random-access versioned querying of RDF archives. *Journal of Web Semantics* 54 (2019), 4–28.
[11] Diego Vinasco-Alvarez, John Samuel Samuel, Sylvie Servigne, and Gilles Gesquière. 2021. Towards a semantic web representation from a 3D geospatial urban data model. In *SAGEO 2021, 16ème Conférence Internationale de la Géomatique, de l'Analyse Spatiale et des Sciences de l'Information Géographique. (Actes de la Conférence SAGEO 2021).* La Rochelle [Online Event], France, 227–238. https://hal.science/hal-03240567
[12] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. 2018. Version Control System: A Review. *Procedia Computer Science* 135 (2018), 408–415. https://doi.org/10.1016/j.procs.2018.08.191 The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

Received June 3, 2024; accepted July 12, 2024; revised September 5, 2024

[8] https://projet.liris.cnrs.fr/vcity/