Leveraging Zarr to handle RDF data

Ángel Iglesias Préstamo[1], Diego Martín Fernández[1], Jose Emilio Labra Gayo[1] and Josh Moore[2]

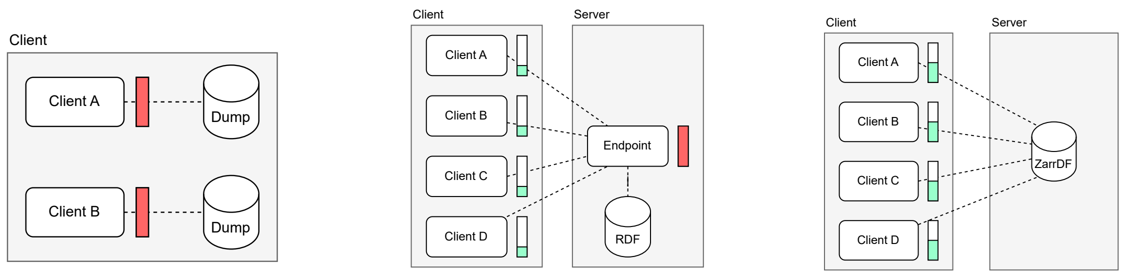
[1] *WESO Lab - University of Oviedo, Spain*
[2] *Open Microscopy Environment - German BioImaging, e.V., Constance, Germany*

Abstract
Handling large RDF graphs often requires a considerable amount of computational resources, posing challenges for processing on personal computers or resource-constrained environments. This paper proposes leveraging Zarr, a chunked and compressed storage format, to partition RDF datasets into manageable pieces. By taking advantage of Zarr's distributed architecture, RDF data can be accessed on-demand, enabling clients to retrieve specific subsets of the graph. This approach allows the efficient processing of datasets that exceed available memory while eliminating the need for dedicated servers, bridging the gap between lightweight client-side operations and the demands of handling complex knowledge graphs.

Keywords
Knowledge Graphs, RDF, Zarr, Distributed, Data storage, Serverless, Triple Patterns

1. Introduction

Knowledge graphs have been applied across numerous domains [1], including the life sciences field, where they provide mechanisms for integrating, analysing, and reasoning over diverse biological and biomedical datasets [1]. However, as these graphs grow in size and complexity, the task of managing and querying over them becomes increasingly challenging [2].



(a) RDF dumps (b) Client-server architecture (c) Proposed architecture

Figure 1: Comparative among the state-of-the-art approaches and the architecture proposed in this paper.

Many state-of-the-art approaches for RDF graph management often rely on in-memory processing [3], which becomes unfeasible when datasets exceed the capacity of available RAM. To overcome this, client-server architectures are employed [4], where RDF graphs are stored in distributed servers and accessed via SPARQL endpoints. However, this approach introduces a trade-off: while it harnesses the computational power and scalability of servers, it often comes at the cost of reduced availability and increased reliance on network infrastructure. A serverless approach, such as the one proposed in this paper, mitigates these issues by enabling distributed storage and on-demand retrieval of RDF data

iglesiasangel@uniovi.es (I. Préstamo); labra@uniovi.es (J. E. L. Gayo); josh@openmicroscopy.org (J. Moore)
https://angelip2303.github.io/ (I. Préstamo); https://labra.weso.es (J. E. L. Gayo); http://joshmoore.github.io (J. Moore)
0009-0004-0686-4341 (I. Préstamo); 0009-0003-6640-9474 (D. M. Fernández); 0000-0001-8907-5348 (J. E. L. Gayo); 0000-0003-4028-811X (J. Moore)

using Zarr, thus eliminating the need for a dedicated server. Nonetheless, this approach introduces its own trade-offs compared to traditional SPARQL-based solutions. As it is not a dedicated query engine, the proposed framework cannot fully replicate the expressive power and optimization capabilities of SPARQL. The system is particularly suited for scenarios where lightweight, parallel access to RDF subsets is required, rather than the execution of complex federated queries.

This paper proposes using Zarr to manage RDF-based knowledge graphs, storing the data as chunked arrays on the server side. By doing so, clients can retrieve specific portions of the graph on demand, reducing memory requirements while supporting efficient and scalable access patterns.

2. Related work

Efficient management of RDF data has been widely studied, leading to the development of various strategies to optimize storage and query performance [3]. One notable contribution is HDT (Header Dictionary Triples), which defines a compact RDF binary serialization format preserving the graph's querying capabilities [5, 6]. By leveraging an encoded front-coding dictionary, HDT minimizes redundancy in data storage [7].

Traditionally, there exists two main alternatives for publishing Linked Data on the web: (1) offering a public SPARQL endpoint, which enables live querying but requires altt the computational burden to be performed on the server-side, (2) offering a data dump, which requires users to set up their own endpoints, shifting all the computational load to the client [4]. Linked Data Fragments introduces a middle ground by exposing a simple triple-pattern interface on the server that shows clients how to access the dataset [4]. In this paradigm, the client assumes responsibility for query planning and execution, balancing the workload between server and client.

Parquet is a columnar storage format optimized for analytical workloads, enabling efficient querying and compression, particularly in distributed systems [8]. Unlike Zarr, which is optimized for representing multi-dimensional arrays, Parquet focuses on tabular data, making it an alternative for representing RDF-based graphs.

There exists several RDF stores and SPARQL servers, such as Apache Jena Fuseki, and its RDF storage and query engine optimized for high performance on a single machine, TDB, that contribute to efficient RDF data management [9]. In the case of Fuseki, a SPARQL server that enables query execution over RDF datasets, it can be integrated with tools like HDT to further enhance the storage efficiency [6].

3. Representing RDF using Zarr

The Zarr open standard for storing large multidimensional array data is built on top of two main abstractions; *e.g.* **chunking**, which enables distributed and parallel access to possibly larger than RAM datasets, and **compression**, which ensures efficient data storage.

By drawing inspiration from columnar storage formats, RDF graphs can be represented such that each triple corresponds to a row, with the subject, predicate, and object components stored in separate columns. Our proposed framework leverages the capabilities of Zarr for storing graph-like data structures.

In this framework, chunking is aligned with the concept of *graph neighbourhoods*, where triples are grouped by an indexing scheme (*e.g.* subject, predicate, or object), and stored in their respective chunks. This layout ensures that retrieving triples based on any of these indices is a straightforward operation: simply access the corresponding chunk. For instance, grouping triples by subject enables efficient retrieval of all outgoing edges for a given node, as all related triples are stored within a single chunk. These fundamental retrieval operations serve as the building blocks for more advanced graph-processing tasks, such as ShEx-based validation or graph traversal algorithms, providing a robust foundation for their implementation. This design not only facilitates efficient retrieval of graph subsets but also supports parallel processing, as each chunk can be accessed and processed independently.

```
prefix :    <http://example.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

:a  :name       "Alice"                 ;
    :birthdate  "1990-05-02"^^xsd:date  ;
    :enrolledIn :cs101                  .

:b :name "Bob", "Robert" .

:cs101 :name "Computer Science" .
```

Listing 1: Simple RDF-based graph that is used for showing the proposed Zarr representation.

	Predicate	Object
Chunk 1	:name	Alice
	:birthDate	1980-03-10
	:enrolledIn	:cs101
Chunk 2	:name	Bob
	:name	Robert
Chunk 3	:name	Computer Science

(a) Subject index

	Subject	Object
Chunk 1	:Alice	1980-03-10
Chunk 2	:Alice	:cs101
Chunk 3	:Alice	Alice
	:Bob	Bob
	:Bob	Robert
	:cs101	Computer Science

(b) Predicate index

	Subject	Predicate
Chunk 1	:Alice	:name
Chunk 2	:Bob	:name
Chunk 3	:cs101	:name
Chunk 4	:Alice	:enrolledIn
Chunk 5	:Alice	:birthDate
Chunk 6	:Bob	:name

(c) Object index

Figure 2: Visualization of the three indices (subject, predicate, and object) used to organize the RDF graph. The graph is partitioned into chunks based on these indices, enabling efficient retrieval of triples based on their components. For example, the subject index groups all triples related to a specific subject, which can then be accessed independently, supporting scalable and parallel query execution. Chunks can be accessed using a front-coding dictionary, which maps each RDF term to its numeric representation in lexicographical order. For instance, :Alice is the first subject (lexicographically) and thus assigned the first numeric chunk index.

Terms are represented using a front-coding algorithm that assigns indices to prefixes or suffixes that have been previously encountered, optimizing storage for repetitive patterns often found in RDF data. This approach is particularly well-suited for URIs, which frequently share namespace prefixes, or literals, where suffixes such as datatype annotations are common. By leveraging these redundancies, front-coding minimizes storage overhead while maintaining fast access and retrieval capabilities. The generated identifiers are then used to assign chunks to their corresponding index. That is, triples with the i^{th} subject will be stored in the i^{th} chunk of the subject index.

To illustrate the proposed Zarr-based representation of RDF data, consider the RDF graph in Listing 1. The graph describes basic information about two individuals, :Alice and :Bob, including their names and other properties, such as the course in which :Alice is enrolled. The graph is represented using three distinct indices: subject, predicate, and object, as shown in Figure 2. These indices organize the triples into chunks based on their components. For instance, the subject index groups triples by their subjects.

When compared to other graph representations, such as adjacency matrices, this tabular layout supports hypergraphs, where relationships involve more than two RDF nodes. The format can also be extended to include additional columns for metadata, such as provenance information or graph identifiers in the case of RDF datasets. A potential future extension involves defining fixed columns based on a schema. For example, using Shape Expressions (ShEx), a node in the graph with a predetermined set of outgoing edges could be represented as fixed columns in the format.

4. Preliminary results

The framework has been implemented in Rust, a language known for its performance, memory safety, and concurrency capabilities. Rust's ownership model and zero-cost abstractions make it particularly suitable for handling large datasets and parallel operations [10].

However, this approach comes with inherent trade-offs. Zarr, as a storage solution, is not a dedicated query engine. Hence, it cannot match the expressiveness of SPARQL. Instead, the Zarr-based framework focuses on lightweight and parallel access to RDF subsets. It is particularly suited for applications where:

- The dataset size exceeds available memory.
- Query patterns are simple.

The choice of Rust not only aligns with these objectives but also offers a robust foundation for future work, such as schema-driven serialization or integration with graph-based validation tools. The framework has been open-sourced and is available at Github[1].

5. Conclusions and future work

This paper presents a novel approach for storing RDF data using Zarr, leveraging its chunking and compression features to enable scalable and efficient handling of large graphs without the need for a dedicated server. By aligning chunking with graph neighbourhoods and utilizing front-coding for storage optimization, the framework supports efficient access patterns and hypergraph representation.

Moving forward, the main focus will be on assessing its performance compared to established formats like HDT, Triple Pattern Fragments, and Parquet. Additional efforts could focus on supporting schema-driven serialization and its integration with existing RDF workflows (*e.g.* schema-based validation).

Acknowledgments

We acknowledge the support for this research provided by the Chan Zuckerberg Initiative DAF, an advised fund of the Silicon Valley Community Foundation, under grant number 2021-237467. We thank NumFOCUS, a nonprofit that supports open code for better science, for their promotion of open practices in research, data, and scientific computing, and their promotion of Zarr. This work has also been partially funded by the Project ANGLIRU: "Applying kNowledge Graphs to research data interoperabiLIty and ReUsability", code: PID2020-117912RB.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] A. Hogan, C. Gutierrez, M. Cochez, G. d. Melo, S. Kirrane, A. Polleres, R. Navigli, A.-C. N. Ngomo, S. M. Rashid, L. Schmelzeisen, S. Staab, E. Blomqvist, C. Amato, J. E. Labra-Gayo, S. Neumaier, A. Rula, J. Sequeda, A. Zimmermann, Knowledge Graphs, Springer International Publishing, 2022. doi:10.1007/978-3-031-01918-0.

[2] S. A. Hosseini Beghaeiraveri, J. E. Labra Gayo, A. Waagmeester, A. Ammar, C. Gonzalez, D. Slenter, S. Ul-Hasan, E. Willighagen, F. McNeill, A. J. Gray, Wikidata subsetting: Approaches, tools, and evaluation, Semantic Web (2023) 1–27. doi:10.3233/sw-233491.

[3] W. Ali, M. Saleem, B. Yao, A. Hogan, A.-C. N. Ngomo, A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs, 2021. arXiv:2102.13027.

[4] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, Triple Pattern Fragments: a low-cost knowledge graph interface for the Web, Journal of Web Semantics 37–38 (2016) 184–206. URL: http://linkeddatafragments.org/publications/jws2016.pdf. doi:doi:10.1016/j.websem.2016.03.003.

[1]https://github.com/weso/ZarrDF

[5] J. Fernández, M. A. Martínez-Prieto, C. Gutierrez, A. Polleres, M. Arias, Binary RDF Representation for Publication and Exchange (HDT), Journal of Web Semantics 19 (2013) 22–41. doi:10.1016/j.websem.2013.01.002.

[6] M. A. Martínez-Prieto, M. Arias, J. D. Fernández, Exchange and Consumption of Huge RDF Data, in: The Semantic Web: Research and Applications, Springer, 2012, pp. 437–452.

[7] M. A. Martínez-Prieto, N. Brisaboa, R. Cánovas, F. Claude, G. Navarro, Practical compressed string dictionaries, Information Systems 56 (2016) 73–108. URL: https://www.sciencedirect.com/science/article/pii/S0306437915001672. doi:https://doi.org/10.1016/j.is.2015.08.008.

[8] Apache Software Foundation, Parquet, 2024. URL: https://parquet.apache.org.

[9] Apache Software Foundation, Jena Fuseki, 2024. URL: https://jena.apache.org/documentation/fuseki2/.

[10] W. Bugden, A. Alahmar, Rust: The Programming Language for Safety and Performance, 2022. URL: https://arxiv.org/abs/2206.05503. arXiv:2206.05503.