Evaluating Oxigraph Server as a triple store for small and medium-sized datasets

Masashi Yokochi[1] **and Nishad Thalhath**[2]

1 Protein Research Foundation, Osaka, Japan **2** RIKEN Center for Integrative Medical Sciences, Yokohama, Japan

BioHackathon series:
DBCLS BioHackathon 2023
Kagawa, Japan, 2023
R4

Submitted: 01 Jul 2023

Background

In recent years, bioinformatics has undergone a considerable transformation, marked by a shift towards adopting semantic web technologies to underpin (meta)data interoperability. This pivotal transition signifies a progressive step in the bioinformatics landscape, fostering enhanced data integration and accessibility. The Resource Description Framework (RDF), a key constituent of these semantic web technologies, has emerged as a vital instrument for the bioinformatics community. The RDF standard, characterised by its flexibility and semantic richness, is instrumental in facilitating advanced data interoperability and linkage. This process is further streamlined by using triplestores, specialised databases for storing and retrieving RDF triples, enabling efficient RDF data querying via the SPARQL Protocol and RDF Query Language (SPARQL) (Katayama et al., 2019).

By offering a standardised approach to data representation, RDF propels researchers to generate data that adheres to the FAIR principles, thereby contributing to enhanced discoverability, accessibility, interoperability, and reusability of data (Wilkinson et al., 2016). As the complexity and volume of bioinformatics data continue to surge, the demand for efficient triplestores has risen correspondingly. However, the heterogeneity of use cases and unique requirements in bioinformatics necessitates a diversity in triplestore implementations. This underscores the need for a plethora of triplestore technologies, each tailored to meet specific requirements, and highlights the importance of sustained innovation in this domain.

Emerging Modern Programming Languages and the Uniqueness of Rust

The continuous emergence of innovative programming languages represents a strategic response to the limitations identified in traditional languages. These modern languages, geared towards augmenting efficiency, readability, and security, are pivotal in creating complex, sturdy, and cutting-edge software systems. Rust, an open-source systems programming language, stands as a salient testament to this trend, establishing a unique standing when juxtaposed with prevalent languages such as C and Java.

Rust is noteworthy for its emphasis on safety, striking a fine balance without compromising performance. In contrast to C's direct memory management, which may lead to vulnerabilities like null pointer dereferencing and buffer overflow, Rust utilises an ‘ownership’ feature to efficiently manage memory at compile time, thereby mitigating the risk of runtime errors. Unlike Java, where garbage collection is used for memory management at the expense of performance, Rust's innovative memory management mechanism optimises performance while ensuring memory safety (Coblenz et al., 2022). Additionally, Rust enhances the Developer Experience (DX) and introduces significant performance improvements, including zero-cost abstractions that facilitate high-level abstractions without sacrificing the efficiency of hand-written lower-level code. Its concurrency model further allows for the parallel execution of tasks, avoiding data races.

Oxigraph: A Rust-based Triplestore

Oxigraph is a dual-licensed[1] open-source graph database. Developed and maintained by Thomas Pellissier Tanon, Oxigraph has been implemented using the Rust programming language and is built upon the rock-solid foundation of RocksDB. It exemplifies the progressive capabilities of Rust and RocksDB to develop robust and efficient systems for managing graph data. The core objective of Oxigraph revolves around catering to the single-node use case in graph database applications. The Oxigraph triple store is implemented as a Rust library, although the project further extends its usability by offering bindings for JavaScript via WebAssembly (WASM) and Python. It encompasses a set of utility functions aimed at facilitating the reading, writing, and processing of RDF files. By leveraging Rust's advanced capabilities, Oxigraph strongly emphasises performance. The Oxigraph project includes a server implementation that provides a standalone ready-to-deploy binary, implementing SPARQL 1.1 Query, SPARQL 1.1 Update, and SPARQL 1.1 Federated Query. specification.

Although currently in development and maintained as a hobby project, Oxigraph exhibits competitive performance in managing simple Online Transaction Processing (OLTP) workloads. The evolution of Oxigraph is an ongoing process, hinting at the prospect of further enhancements and improvements in its performance and capabilities in the future. This evaluation pertains to the standalone Oxigraph server.

Requirements and Use Cases for Evaluating Oxigraph Server

In pursuit of a viable alternative to mainstream triple store for small and medium (sub billion triples), a comprehensive evaluation encompassing a multitude of use cases and requirements has been conducted. Below are various aspects of functionality and user experience, with a particular focus on the capabilities Oxigraph successfully addresses.

1. **Ease of Installation and Maintenance**: The triplestore should be straightforward to install, ready for immediate use, and easily maintainable. It should efficiently handle small and medium-sized datasets, ensuring optimum performance and reliability.
2. **User Accessibility**: The tool should demand minimal technical skills to run, encouraging a broader user base. This is critical in two primary contexts:
    1. Facilitating testing and development of RDF data by collaborators.
    2. Enable stakeholders to test and learn SPARQL without extensive technical expertise.
3. **Efficient Local Environment Operation**: A key requirement is the ability of the triplestore to load subset RDF dumps in a local environment swiftly and smoothly. The triplestore should also facilitate quick querying without restricting rate limiting, enhancing user productivity.
4. **Backup Provision**: The ability to distribute ready-to-load backups of triplestore data is of utmost importance. This feature can reduce the time to load triples, boosting efficiency.
5. **Standalone Operation**: A vital attribute is the standalone operation capability of the binary, which means it should have zero dependencies. This ensures the system's robustness and minimises potential issues arising from dependencies.
6. **Compatibility with Package Managers**: To ensure widespread adaptability, easy installation through popular package managers, such as Homebrew, should be provided. This feature can further simplify the installation process and enhance the user experience.
7. **Multi-Platform and Multi-Architecture Compatibility**: The triplestore should be engineered to support a diverse array of platforms, including Linux, macOS, and Windows, and accommodate multiple architectures such as x86_64, aarch64, and armv7. This

[1] Apache License Version 2.0 and MIT License.

comprehensive compatibility can enhance the user experience by enabling the tool's utilisation across various platforms and architectures.

8. **Support for Multiple RDF Serializations, SPARQL Query and Query Result Formats**: The triplestore should offer comprehensive support for various RDF serialisations, encompassing Turtle, N-Triples, and N-Quads. Moreover, it should facilitate the usage of a broad range of SPARQL query formats, including SPARQL Query, SPARQL Update, and SPARQL Federated Query. Additionally, the accommodation of multiple SPARQL query result formats, such as SPARQL Query Results in JSON, CSV, and TSV, is essential. These features collectively enhance the user experience by ensuring the tool's applicability across various RDF serialisations and SPARQL query and query result formats.

Methedology

To thoroughly evaluate the Oxigraph server, two distinctive devices were configured as our testing environments: Device A and Device B. Device A, serving as the primary test environment, is a regular computer, while Device B is a high-configuration portable computer. Both devices operate on macOS, albeit on different hardware architectures (Intel and ARM). Table 1 provides a detailed exposition of the device configurations.

Table 1: Device Configurations

	Device A	Device B
Device	MacBook Pro, 2018	MacBook Pro, 2021
Year	2018	2021
Processor	Intel Core i5	Apple M1 Max
Clock Speed	2.3 GHz	3.2 GHz
Cores	4	10
Memory	8 GB	64 GB
OS	macOS Ventura 13.4	macOS Ventura 13.4

The Oxigraph server (v0.3.18) executables were installed via the Homebrew package manager, widely used in the macOS environment. We utilised subsets of the PDB/RDF archive and chem_comp/RDF archive (Bekker et al., 2022). Both datasets are publicly available from PDBj, while the complete datasets have been deployed on the NBDC/DBCLS RDF Portal (Kawashima et al., 2018).

We retrieved two RDF datasets comprising approximately 0.5 billion triples in total via the rsync service of PDBj. Each dataset consists of a bundle of gzipped RDF/XML files corresponding to each PDB or CCD (also known as PDB ligand) entry. Table 2 provides an overview of the subset RDF. Several helper scripts were prepared to load the RDF graph into the Oxigraph server. The primary script decompresses all the gzipped files and places them into subdirectories, each containing up to 25 RDF files. This pre-processing stage aims to optimize Oxigraph's parallel loading performance without exceeding the operating system's limitations; by default, the maximum number of files open for a process is 256. The scripts and data used for this study are available at the project GitHub repository and at the Zenodoarchive (Yokochi & Thalhath, 2023).

Table 2: Subset RDF Datasets and tripes count (as of June 28, 2023)

Data Source	Triples Count	Graph URI	Entries Count
PDB/RDF	417,415,997	http://rdf.wwpdb.org/pdb	10,026 of 206,656 PDB entries

Data Source	Triples Count	Graph URI	Entries Count
chem_comp/RDF	81,664,755	http://rdf.wwpdb.org/cc	48,642 of 48,642 CCD entries

The Oxigraph server comes equipped with an inbuilt YASGUI, which is exposed at the root of the HTTPS server. For all query analyses, we employed this YASGUI as the query interface. Furthermore, we validated that the default /query endpoint functions correctly via `curl` as well.

Multiple Oxigraph Server Setup and Federated Query

In order to evaluate and deploy multiple SPARQL endpoints, we utilized the Caddy Server as a reverse HTTP proxy server, and configured it with two SPARQL endpoints from the Oxigraph server. These two triplestore instances were independently loaded with PDB/RDF and chem_comp/RDF datasets. A federated query was then crafted utilizing both endpoints, in an effort to scrutinize the federated search capabilities of the Oxigraph server. A schematic representation of this setup is provided in Figure 1. Configuration file for Caddy reverse proxy is provided in the GitHub repository.

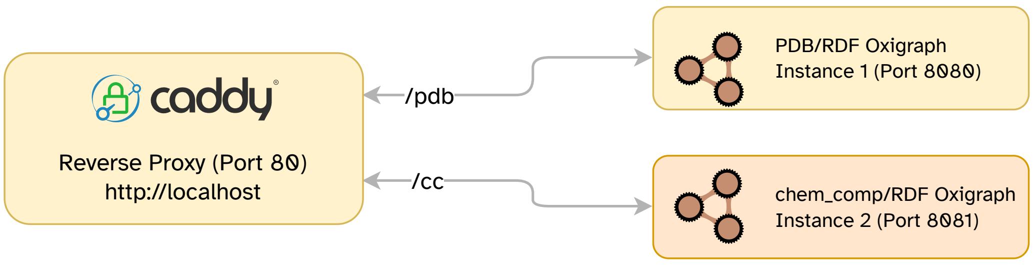

Figure 1: Figure: Schematic representation of the Multiple Oxigraph server setup and Caddy reverse proxy.

We drafted an example of a federated SPARQL query encompassing PDB/RDF and chem_comp/RDF endpoints.

SPARQL example:

The SPARQL query ties to find ligands in complex with serine/threonine phosphatase in the subset of the PDB archive (4.8% of all PDB entries) and retrieve the InChI code of hit ligands from chem_com/RDF. We got three types of ligand compounds from the subset of the PDB archive (4.8% of the entire PDB/RDF); `GLYCEROL`, `{2-[5-hydroxy-4-(hydroxymethyl)-6-methylpyridin-3-yl]ethyl}phosphonic acid`, and `CITRIC ACID`. We succeeded in getting the InChI code of the three ligands.

```
PREFIX pdbo: <http://rdf.wwpdb.org/schema/pdbx-v50.owl#>

SELECT DISTINCT ?pdb_entry ?ligand ?inchi
WHERE {
  # GO:0004721 -> protein serine/threonine phosphatase activity
  ?sub pdbo:pdbx_sifts_xref_db_segments.xref_db_acc "GO:0004721" ;
       pdbo:of_datablock ?pdb_entry .
  ?pdb_entry pdbo:has_pdbx_entity_nonpolyCategory ?entity_nonpoly_cat .
  ?entity_nonpoly_cat pdbo:has_pdbx_entity_nonpoly ?nonpoly .
  ?nonpoly pdbo:pdbx_entity_nonpoly.name ?ligand ;
           pdbo:pdbx_entity_nonpoly.comp_id ?comp_id .
  FILTER (?ligand!="water" && !STRENDS(?ligand, " ION"))
  ?pdb_entry pdbo:has_chem_compCategory ?cc_cat .
  ?cc_cat pdbo:has_chem_comp ?cc .
  ?cc pdbo:chem_comp.id ?comp_id ;
      pdbo:link_to_chem_comp ?ccd_entry .

 SERVICE <http://localhost/cc/query> {
  ?ccd_entry pdbo:has_pdbx_chem_comp_descriptorCategory ?cc_desc_cat .
  ?cc_desc_cat pdbo:has_pdbx_chem_comp_descriptor ?cc_desc .
  ?cc_desc pdbo:pdbx_chem_comp_descriptor.type "InChI" ;
           pdbo:pdbx_chem_comp_descriptor.descriptor ?inchi .
 }
}
```

Results

The PDB/RDF and chem_comp/RDF archives are provided as multiple gzip compressed RDF/XML formatted files, each representing an individual entry. The simultaneous decompression of these initial resources and the generation of Oxigraph's persistent data files takes place during the loading process. To conserve local storage space, we devised two scenarios for loading the RDF archives:

1. Decompression of individual files on-the-fly, followed by their removal post successful load.
2. Decompression of all files concurrently, and processing them as a bulk operation.

The first scenario primarily mitigates storage space requirements, while the second one is designed to enhance loading performance. Loading a subset of the PDB/RDF archive consumed 11.24 hours for individual file loading and 2.48 hours for bulk loading on Device-A. These durations include the initial decompression process (refer to Figure 2) The parallel loading capability of the Oxigraph server led to variations in loading time for the same triple data loading task. A similar pattern was observed on Device-B; the higher-specification machine effortlessly managed a sub-billion triples dataset.

The final size of Oxigraph's internal persistent files was anticipated to be 3-4 times larger than the original compressed archive. This is attributed to Oxigraph's storage method, which organizes triples in three combinations of orders; `s->p->o`, `p->o->s`, and `o->s->p` where `s`, `p` , and `o` denote subjects, predicates, and objects, respectively [2].

To avoid unnecessary performance degradation, it is advisable to configure the local OS's full-text indexing service (for instance, `mdworker` known as Spotlight® on macOS) to disregard the current working directory before initiating the loading of RDF files.

[2]https://github.com/oxigraph/oxigraph/wiki/Architecture

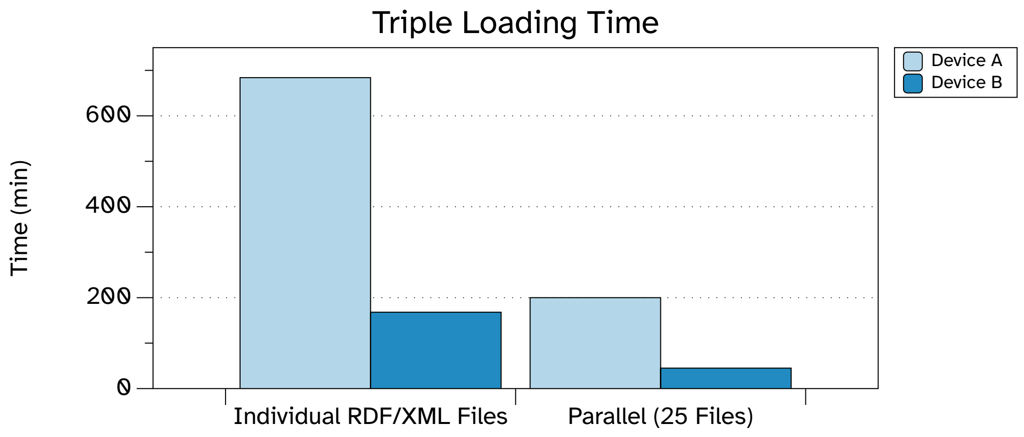

Figure 2: Figure: Triple loading time for individual files and parallel loading

Ensuring the interoperability of the resulting persistence files is a key step in distributing a ready-to-use triple store persistant files. We have confirmed that there are no hindrances in transferring ready-to-use triple stores between differing CPU architectures (x86_64 and AArch64, albeit both running the same version of macOS). The cross-compatibility of the triple store's backups must be validated across Linux, macOS, and Windows environments. This necessity arises from the dependency of backup files on the RocksDB implementations unique to each platform. Notably, RocksDB serves as a dedicated key-value storage system for the Oxigraph server.

We assessed the performance of query evaluation using a straightforward SPARQL query that retrieves triples by altering LIMIT numbers (see Table 2). Higher configuration machines demonstrated superior performance compared to known public endpoints, attributable to the absence of interference from other users and network latency. Consequently, users can enhance performance by investing in local machines. However, the most substantial advantage of deploying a triple store locally is the user's liberation from timeout limitations imposed by triple store providers. This freedom allows users to execute more complex or time-consuming SPARQL queries.

Table 3: Simple triple SELECT query performance (in seconds)

LIMIT	Device A	Device B
1	0.01	0.012
10	0.013	0.013
100	0.03	0.038
1000	0.189	0.146
10000	0.93	0.611
100000	6.844	3.343
1000000	66.384	22.158

Discussion

These findings encourage that modern implementations of the SPARQL endpoint are ready to be expanded to billion-scaled databases which cover almost all large datasets that are available at present. As for small datasets, the Oxigraph server allows an endpoint to be deployed entirely on memory, ensuring the fastest response as much as possible.

While well-established SPARQL benchmarking frameworks such as the Berlin SPARQL Benchmark (BSBM) (Bizer & Schultz, 2009) exist to evaluate the performance of SPARQL triplestores, the focus of this study is on appraising the Oxigraph server's completeness for our specific use cases. Our assessment entails exploring facets of triplestore management in a production environment beyond SPARQL performance. We scrutinized aspects including the loading, backup, deployment, maintenance, and usability of the Oxigraph server.

RedStore, a discontinued RDF triplestore, fulfilled many prerequisites and has many common features with the Oxigraph server. RedStore was written in C using the Redland library as a lightweight triplestore. In terms of features, RedStore was quite capable. It possessed a built-in HTTP server and was available as an application for Mac OS X. It supported a broad range of RDF formats, with the only runtime dependency being Redland. RedStore was compatible with the `rdfproc` command line tool, allowing offline operations. Despite its discontinuation, RedStore's features underscore the potential of the lightweight, efficient triplestore in RDF data management.

Oxigraph has potential use cases for small datasets, such as a lightweight in-memory graph database for web browsers, edge computing, and serverless environments. Experimental serverless implementation for tiny datasets is already available, proving its strength in working with light RDF datasets [3].

Conclusions

All the software applications employed in this study are conveniently accessible through standard package managers in various operating systems (in this study, homebrew), thereby mitigating the challenges inherent in setting up Oxigraph locally or even multiple endpoints in a local environment for specific use cases. Available better-configured portable computers possess the capacity to manage sub-billion triple databases effortlessly. Furthermore, we have illustrated that medium-sized databases can be seamlessly migrated even on standard portable machines. This highlights the pivotal role of distributing compressed and optimised persistent files for enhancing user experience. Throughout this study, we did not encounter any hash collisions within the persistent internal files of Oxigraph, an issue that had been a concern for the Oxigraph developer and potential users.

Future Directions

PDBj aims to deploy their complete sets, exceeding 11 billion triples, which include PDB/RDF, chem_comp/RDF, BIRD/RDF, and VRPT/RDF. Here, BIRD represents the Biologically Interesting Molecule Reference Dictionary and VRPT denotes wwPDB validation reports that assess coordinates and author-provided experimental data.

Additionally, we have plans to perform a comparative evaluation of the Oxigraph Server against industry-standard triplestores. This endeavour will leverage an appropriate evaluation framework, with the intent to publish the resultant findings.

Acknowledgements

The authors would like to express their sincere gratitude to the participants of BioHackathon 2023. Their collaborative efforts and valuable feedback significantly contributed to the advancement of our project. We extend our appreciation to the organizers for creating such an engaging and fruitful platform. We would like to particularly acknowledge Thomas Pellissier Tanon for his exceptional work in developing and open sourcing Oxigraph. Furthermore, we would like to recognize the Protein Data Bank in Japan (PDBj) and its collaborators for their generosity in providing the necessary RDF data.

Declaration of Generative AI and AI-assisted Technologies in the writing process

[3]https://github.com/nishad/serverless-sparql-endpoint/

In preparing this work, the authors utilised Grammarly and the OpenAI API to enhance readability, rectify grammatical errors, and refine language use. After using these tools, the authors conducted a rigorous review and editing process to ensure the manuscript's accuracy and relevance. The authors bear complete responsibility for the final content of the manuscript.

References

Bekker, G.-J., Yokochi, M., Suzuki, H., Ikegawa, Y., Iwata, T., Kudou, T., Yura, K., Fujiwara, T., Kawabata, T., & Kurisu, G. (2022). Protein Data Bank Japan: Celebrating our 20th anniversary during a global pandemic as the Asian hub of three dimensional macromolecular structural data. *Protein Science*, *31*(1), 173–186. https://doi.org/10.1002/pro.4211

Bizer, C., & Schultz, A. (2009). The Berlin SPARQL Benchmark: *International Journal on Semantic Web and Information Systems*, *5*(2), 1–24. https://doi.org/10.4018/jswis.2009040101

Coblenz, M., Mazurek, M. L., & Hicks, M. (2022). Garbage collection makes rust easier to use: A randomized controlled trial of the bronze garbage collector. *Proceedings of the 44th International Conference on Software Engineering*, 1021–1032. https://doi.org/10.1145/3510003.3510107

Katayama, T., Kawashima, S., Micklem, G., Kawano, S., Kim, J.-D., Kocbek, S., Okamoto, S., Wang, Y., Wu, H., Yamaguchi, A., Yamamoto, Y., Antezana, E., Aoki-Kinoshita, K. F., Arakawa, K., Banno, M., Baran, J., Bolleman, J. T., Bonnal, R. J. P., Bono, H., … Takagi, T. (2019). *BioHackathon series in 2013 and 2014: Improvements of semantic interoperability in life science data and services* (No. 8:1677). F1000Research. https://doi.org/10.12688/f1000research.18238.1

Kawashima, S., Katayama, T., Hatanaka, H., Kushida, T., & Takagi, T. (2018). NBDC RDF portal: A comprehensive repository for semantic data in life sciences. *Database: The Journal of Biological Databases and Curation*, *2018*, bay123. https://doi.org/10.1093/database/bay123

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., … Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*(1), 160018. https://doi.org/10.1038/sdata.2016.18

Yokochi, M., & Thalhath, N. (2023). *RDF Subset From PDBj for Evaluating Oxigraph Server*. Zenodo. https://doi.org/10.5281/zenodo.8098467