



EPEI: An RDF Retrieval System Based on Efficient Predicate-Entity Indexing

Yipeng Liu¹, Chuangxin Fang², and Yuming Lin¹(✉)

¹ Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China

ymlin@guet.edu.cn

² College of Information Engineering, Northwest A&F University, Yangling, Shaanxi 712100, China

Abstract. This paper proposes an RDF retrieval system that can swiftly establish indexes and maintains a relatively smaller index size with the following features: 1) using predicate-entity index structure to reduce disk space usage and construction time; 2) query enhancements based on worst-case optimal join; and 3) a user-friendly graphical interface. The experiment results demonstrate that our system uses less disk space, achieves more rapid index construction, and maintains competitive query performance compared to the latest RDF retrieval systems.

Keywords: RDF · Indexing · Retrieval

1 Introduction

Currently, knowledge graphs provide knowledge support for numerous intelligent applications and have become a critical infrastructure. RDF (Resource Description Framework) stands as one of the mainstream approaches to knowledge representation. However, as the volume of knowledge graphs continues to grow, effectively retrieving massive RDF data becomes extremely important. Existing RDF retrieval systems can be categorized into relational-based and graph-based systems according to their storage mechanisms [1]. Relational-based systems like RDF-3X [4] and Jena-LTJ [3] utilize relational database techniques, including B+trees for indexing RDF data, whereas graph-based systems such as gStore [6] and QDags [2] exploit RDF and SPARQL features to devise tailored storage methods, they both consider RDF data as a graph, gStore adopts an adjacency list for data storage, while QDags uses an adjacency matrix. Prominent RDF retrieval systems such as RDF-3X and gStore demonstrate strong query performance, yet suffer from large index sizes and slow construction speeds.

In this paper, we introduce EPEI, featuring fast index construction, compact index size, and enhanced query speed using a join method called Leapfrog Triejoin [5]. In addition, EPEI provides a user-friendly graphical interface.

2 System Architecture and Key Techniques

The system architecture of the proposed EPEI is illustrated in Fig. 1(a).

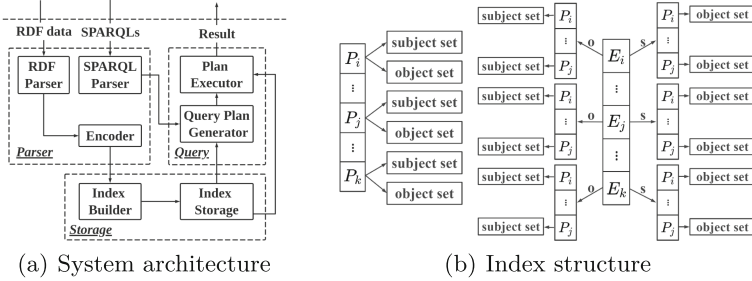


Fig. 1. The architecture of the system and index structure

The Parser Module. This module parses RDF triples and SPARQL queries, and assigns IDs to subjects, objects, and predicates during the RDF parsing.

The Storage Module. First, the index builder constructs predicate indexes, mapping predicates to subject and object sets (left side of Fig. 1(b)). Then, it builds entity indexes (right side of Fig. 1(b)), which form a two-tier map where the first tier maps entities to predicate maps similar to predicate indexes, yet the data indexed is constrained by the entity and this entity’s position in the triples. Once the index construction is complete, these indexes are stored for future use. To accelerate construction, multithreading is used.

The Query Module. Once the system parses SPARQL queries, the query plan generator sorts the variables by their frequency in triple patterns, giving priority to those most frequent. When frequencies are equal, variables more common in univariate triple patterns are prioritized. The generator then constructs a query plan table, assigning sorted variables to rows and triple patterns to columns, with each variable placed in the appropriate cell. Figure 2 illustrates an example where SPARQL’s triple patterns are displayed on the left, and the corresponding query plan is depicted on the right.

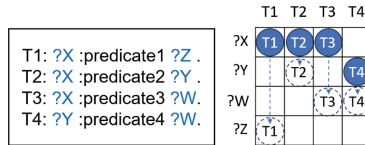


Fig. 2. Example of a query plan

The executor executes the plan table from top to bottom. For every row, the executor employs constants in triple patterns to retrieve sets from the indexes

Table 1. Comparison of index size (GB), build time (s).

	WatDiv100M		Wikidata	
	size	time	size	time
RDF-3X	5.09	745	14.72	1275
Jena-LTJ	15.67	1145	26.02	1783
gStore	8.2	597	41.02	1947
EPEI	2.11	89	9.38	172

Table 2. Query time (ms) for WatDiv100M.

	C	L	SF	S
RDF-3X	1870.4	30.01	21.94	6.52
Jena-LTJ	2963.2	48.12	22.41	19.28
gStore	610.66	34.11	60.11	6.51
EPEI	857.26	5.63	10.86	4.11

for variable cells in the row that have no set. Subsequently, join operation is executed among sets in the row, and constants in triple patterns and join results are used to retrieve sets from the indexes for the other variable in every triple pattern. If all of the variable cells have a set when reaching the table bottom, each present value in every join result set forms a result tuple. Once all result tuples are found, the executor utilizes pre-loaded hashmaps that map IDs to entities and predicates to produce the final results.

3 Evaluation and Demonstration

We initially run the widely recognized WatDiv benchmark with a dataset comprising 108,997,714 triples. The benchmark provides four types of queries: complex (C), linear (L), snowflake (SF), and star (S). We also run the Wikidata Graph Pattern Benchmark (WGPPB) [3], which uses a Wikidata sub-graph with 141,101,754 triples and provides 17 query patterns of different widths and shapes, including cyclic and acyclic queries. The experiments are carried out on an i5-12400F 4.40 GHz CPU, Ubuntu 22.04 as the OS, and 32 GB of RAM.

The comparison of index size and construction time on WatDiv and Wikidata datasets is presented in Table 1, while the comparison of query time is displayed in Table 2 and Fig. 3. The index construction process in EPEI utilized three threads and the query time in Fig. 3 was performed using $\log_2(t + 1)$. Since gStore must load the index cache into memory and its index size on Wikidata exceeds our equipment’s memory capacity, the associated query time is not shown in Fig. 3. The experiment shows that our system achieves faster data index construction and indexes are smaller, outperforming RDF-3X and Jena-LTJ in the majority of query types. However, for some cyclic queries, such as S1, S2, and S3, the query efficiency of EPEI is relatively low.

Finally, EPEI’s interactive graphical interface is shown in Fig. 4, which enables a variety of RDF database operations. For more demonstrations, the source code of EPEI is available on Github¹, and a video can be found on YouTube².

¹ <https://github.com/MKMaS-GUET/EPEI>.

² <https://m.youtube.com/watch?v=WJeGAlsdBvQ>.

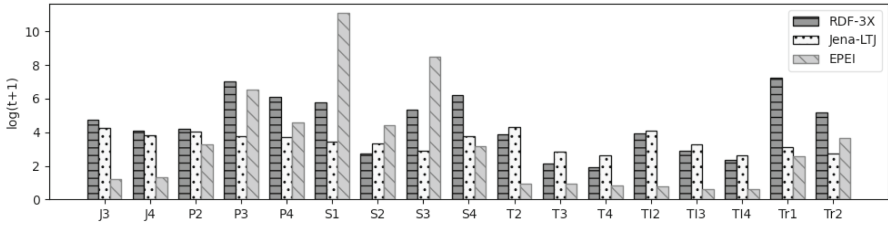


Fig. 3. Query time (ms) for Wikidata

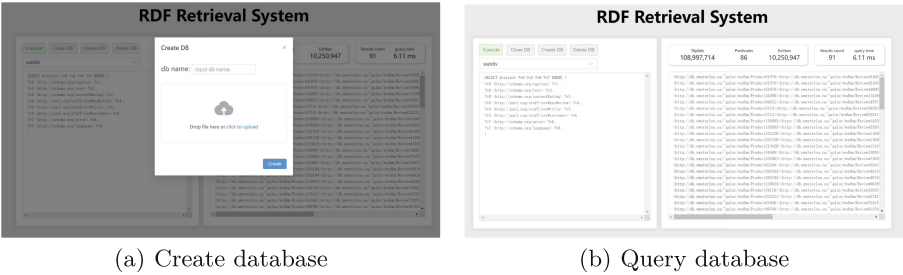


Fig. 4. Interactive graphical interface

4 Conclusion and Future Work

The EPEI system represents a step forward in RDF retrieval, offering smaller indexes and quicker indexing than current systems. The integration of Leapfrog Triejoin enhances query efficiency. However, the system struggles with some cyclic queries. This is mainly due to the simplistic rules for query plan generation, which limits adaptability to diverse situations. Future work will aim to design better plan generator and employ multithreading for faster query execution.

Acknowledgement. This work was supported by National Natural Science Foundation of China (Nos. 62062027, 62362015, U22A2099), Innovation Project of GUET Graduate Education (No. 2023YCXSO65) and the project of Guangxi Key Laboratory of Trusted Software.

References

1. Ali, W., Saleem, M., Yao, B., Hogan, A., Ngomo, A.C.N.: A survey of rdf stores & sparql engines for querying knowledge graphs. *VLDB J.*, 1–26 (2022)
2. Arroyuelo, D., Navarro, G., Reutter, J.L., Rojas-Ledesma, J.: Optimal joins using compressed quadrees. *ACM Trans. Database Syst.* **47**(2), 1–53 (2022)
3. Hogan, A., Riveros, C., Rojas, C., Soto, A.: A worst-case optimal join algorithm for SPARQL. In: Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.) *ISWC 2019. LNCS*, vol. 11778, pp. 258–275. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30793-6_15

4. Neumann, T., Weikum, G.: The rdf-3x engine for scalable management of rdf data. *VLDB J.* **19**, 91–113 (2010)
5. Veldhuizen, T.L.: Triejoin: a simple, worst-case optimal join algorithm. In: *Proc. 17th ICDT*, pp. 96–106 (2014)
6. Zou, L., Özsu, M.T., Chen, L., Shen, X., Huang, R., Zhao, D.: gstore: a graph-based sparql query engine. *VLDB J.* **23**, 565–590 (2014)