Continuation Queries: Embracing Timeouts on Public SPARQL Endpoints

Thi Hoang Thi Pham[1,*], Gabriela Montoya[1], Brice Nédelec[1], Hala Skaf-Molli[1] and Pascal Molli[1]

[1]Nantes Université, LS2N, UMR 6004, F-44000 Nantes, France

Abstract

Public SPARQL endpoints, such as Wikidata, provide essential access points to large-scale knowledge graphs. However, they often suffer from strict timeouts that prevent the retrieval of complete query results. This demonstration presents the first public deployment of PASSAGE, a SPARQL query engine that guarantees query completeness through continuation queries. Instead of failing upon timeout, PASSAGE returns partial results along with a SPARQL continuation query capable of retrieving the missing results. These continuation queries can be chained iteratively until complete results are obtained. For this demo, attendees can interact with a PASSAGE loaded with 13B triples from Wikidata 2025, and observe in detail its operation during their query execution.

Keywords

Semantic Web, Public Knowledge Graph, SPARQL Endpoint, Continuation Queries.

1. Introduction

Public SPARQL endpoints, such as Wikidata[1], offer valuable access to large-scale knowledge graphs. However, to remain responsive under heavy load, they enforce fair-use policies, including timeouts and result size limits [1], to prevent a single query from monopolizing server resources. Consequently, many queries fail to complete, returning partial or no results at all. For instance, consider the query Q_{cite} in Figure 1a, which retrieves pairs of articles that cite each other. When executed on the official Wikidata endpoint[2], this query times out after 60 seconds and fails to return complete results. The inability to ensure query completeness undermines both the reliability and usability of public SPARQL endpoints.

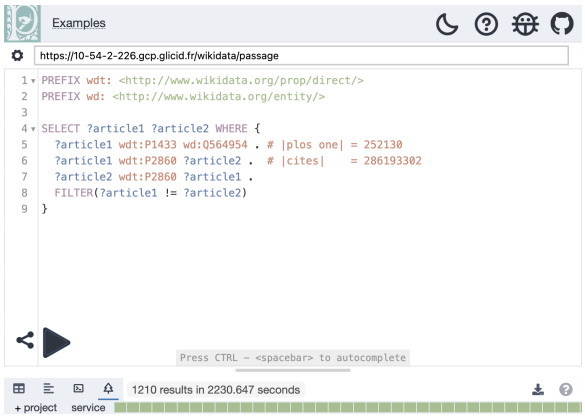
In a recent paper [2], we introduced *SPARQL continuation queries*, a novel approach to overcoming timeout limitations while preserving compatibility with existing SPARQL infrastructure. The core idea is simple: when a query exceeds server-imposed limits, the server returns the partial results *and* a SPARQL continuation query able to retrieve the missing results. This process can be repeated, allowing users to recover complete answers by chaining SPARQL continuation queries. To the best of our knowledge, our approach is the first to ensure completeness, responsiveness, and, more importantly, full compliance with the SPARQL protocol.

In this demo, we present the first public deployment of *SPARQL continuation queries* over real-world data. The PASSAGE server embeds two SPARQL query engines within a single Java Virtual Machine: the standard BLAZEGRAPH query engine and the continuation-enabled PASSAGE query engine. Both query engines operate on a shared BLAZEGRAPH journal file (.jnl), ensuring that both query engines access the same physical storage and indexes, while exposing separate SPARQL endpoints.
The BLAZEGRAPH query engine, which powers the Wikidata Query Service, fully supports SPARQL 1.1 but enforces a 60-second timeout that can yield incomplete results for complex queries. In contrast, the

ISWC 2025 Companion Volume, November 2–6, 2025, Nara, Japan
*Corresponding author.
thi-hoang-thi.pham@univ-nantes.fr (T. H. T. Pham); gabriela.montoya@univ-nantes.fr (G. Montoya); brice.nedelec@univ-nantes.fr (B. Nédelec); hala.skaf@univ-nantes.fr (H. Skaf-Molli); pascal.molli@univ-nantes.fr (P. Molli)
0000-0003-0176-2245 (T. H. T. Pham); 0000-0001-5835-0335 (G. Montoya); 0000-0003-4238-5060 (B. Nédelec); 0000-0003-1062-6659 (H. Skaf-Molli); 0000-0001-8048-273X (P. Molli)

[1]https://query.wikidata.org
[2]https://query-scholarly.wikidata.org/, following the recent split of Wikidata.

(a) The query Q_{cite} times out after 60 seconds on the official Wikidata endpoint and returns no results to the user.



(b) Using PASSAGE, the query Q_{cite} completes with 37 SPARQL continuation queries, taking 37 minutes to retrieve its 1210 results.

Figure 1: The query Q_{cite} retrieves articles from the journal PLOS One that cite each other. Articles can cite each other when they are submitted and published in close time frames.

PASSAGE query engine currently supports only core SPARQL [3] but enables query continuation. This configuration allows users to easily choose which engine to use for executing their SPARQL queries: PASSAGE or BLAZEGRAPH.

As part of the demonstration, the attendees will be able to query a public PASSAGE server containing the 13B statements of Wikidata (as of February 13th, 2025), showcasing:

- Execution of SPARQL queries that time out on BLAZEGRAPH but complete successfully on PASSAGE using continuations;
- Live inspection of SPARQL queries through the execution of successive SPARQL continuation queries. Each timeout offers an opportunity to monitor progress and estimate completion time;
- Integration with the COMUNICA smart client [4], enabling support for SPARQL 1.1 queries beyond PASSAGE's core SPARQL capabilities. The client decomposes the queries: core SPARQL subqueries are delegated to the PASSAGE engine, while unsupported operators are executed client-side.

2. Continuation Queries at Work

Timeouts play a crucial role in protecting shared infrastructures. An engine that supports SPARQL continuation queries does not eliminate timeouts; instead, when a query execution is interrupted, the engine returns partial results and computes a new SPARQL capable of retrieving the missing answers, called *a SPARQL continuation query.* A continuation query can itself be interrupted, leading to yet another continuation query. Assuming that each continuation query makes progress, we have proven that there exists a finite sequence of continuation queries that returns complete and correct results [2].

This demo highlights the sequences of SPARQL continuation queries that occur during query execution, as illustrated in Figure 2. On the left is the original query Q_{cite}, which was executed for 1 minute and returned 64 results. In the center is the first continuation query, also executed for 1 minute, returning 26 results. On the right is the second continuation query, which returned 23 results after another 1 minute of execution. At this point, the results are still incomplete, and more continuation queries must be executed to retrieve the remaining answers.

A key point to note is that, although the engine automatically generates each continuation query to compute the remaining work, continuation queries are standard SPARQL queries that any user can read and understand. A human user can inspect it to determine the processed parts of the query and estimate the remaining time required to complete the query execution. Continuation queries help open the black box of SPARQL query processing, allowing users to better understand and reason about how their queries are being executed.

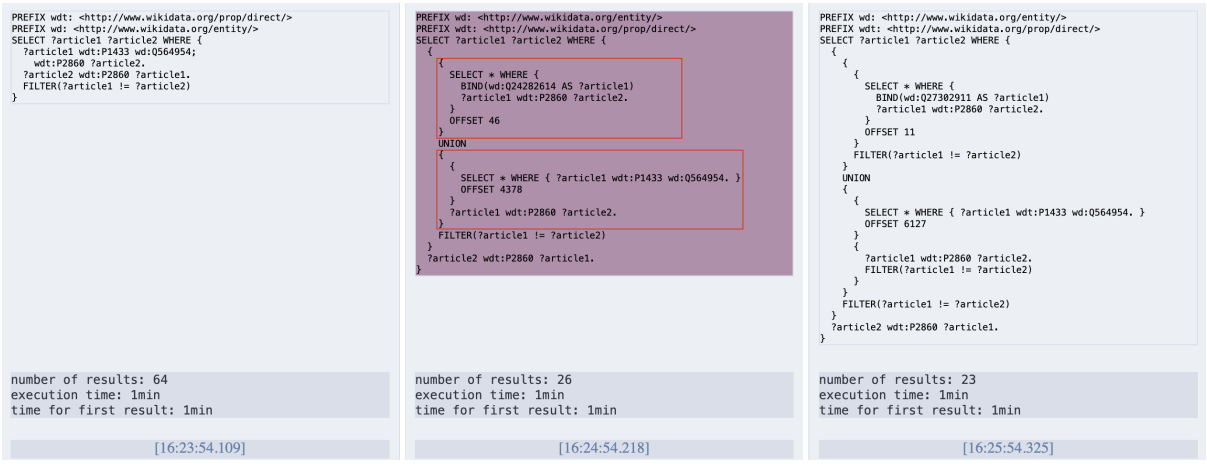


Figure 2: The query Q_{cite} requires multiple continuation queries. After 60 seconds, the initial query (left) returns 64 partial results and a SPARQL continuation query (center) containing additional OFFSET, BIND, and UNION clauses. This continuation query is then executed and returns 26 partial results, along with the continuation query (right). The process continues until the endpoint no longer returns a continuation query.

Understanding a continuation query. Let us focus on the first continuation query of Q_{cite}, shown in the center of Figure 2. The remaining work is primarily determined by the two OFFSET clauses found in the two parts of the UNION: (i) The second part of the UNION re-executes the original query, under the assumption that all articles of PLOS One have already been processed up to OFFSET 4378. The remaining task is therefore to process all PLOS One articles beyond OFFSET 4378. (ii) The first part of the UNION handles the processing of the 4378^{th} article, i.e., `wd:Q24282614`. PASSAGE has already processed all cited articles up to OFFSET 46. The remaining task is therefore to process all articles cited by `wd:Q24282614` beyond OFFSET 46.

An inspection of the second part of the UNION reveals the join order selected by PASSAGE. Since suboptimal join orders are often a major cause of slow SPARQL query execution, it is important to assess whether the chosen order is appropriate:

1. First, PASSAGE scans the innermost triple pattern `{?article1 wdt:P1433 wd:Q564954}` to retrieve the articles published in the journal PLOS One. Starting with this triple pattern is crucial, as it has the lowest cardinality (Figure 1b).
2. Then, each value of `?article1` is injected into the triple pattern `{?article1 wdt:P2860 ?article2}` to retrieve the articles cited by `?article1`.
3. Finally, if `?article1` and `?article2` are distinct, PASSAGE evaluates the pattern `{?article2 wdt:P2860 ?article1}` to check whether `?article2` also cites `?article1`.

Therefore, the join order is well chosen, and the long execution time is primarily due to the size of the search space.

Estimating remaining time. Users are often interested in estimating how long they will need to wait for their query to complete fully. In this context, the continuation query, which represents the remaining work, can be used to estimate the remaining execution time. With the first continuation query of Q_{cite} shown in the center of Figure 2, the execution time of the first part of the UNION is marginal; an article rarely cites millions of others, and 46 citations is already a relatively high number. The primary factor is OFFSET 4378, applied to the first triple pattern. Given that there are 252,130 articles in total, and 4378 have been processed in one minute, we can roughly estimate the remaining time as $\frac{252{,}130}{4378} - 1 \approx 56$ minutes. This is a coarse estimate, since it does not consider citation distribution biases, but it can be refined after each continuation query. In this case, PASSAGE terminates the execution of this query in ~37 minutes, after issuing 37 continuation queries. [3]

[3] Q_{cite} was executed after clearing the server's cache.

(a) PASSAGE x COMUNICA query plan (b) Example of PASSAGE subqueries.

Figure 3: The decomposition of query Q_{top} is complex, involving multiple service calls linked by COMUNICA's physical operators. Each service query is subject to continuation. The query execution completed in 19 minutes, providing the 10 expected results.

Technical view on continuation queries. From a technical perspective, one might be surprised to see OFFSET without ORDER BY, as OFFSET is generally unreliable when the order of results is not explicitly defined [5]. However, PASSAGE extensively uses subqueries comprising a single triple/quad pattern with an offset: by assuming a deterministic evaluation order for such patterns, the resulting triples/quads are always produced in the same order and the OFFSET actually allows skipping results already produced. Although PASSAGE uses the BLAZEGRAPH engine to satisfy this assumption by scanning through its augmented B+Tree indexes (the deterministic order is that of the chosen index), other SPARQL engines based on other indexing data structures (e.g., HDT [6]) can provide deterministic ordering on single triple/quad patterns.
A second remark concerns the evaluation time of a triple/quad pattern with an OFFSET: are all preceding results read and discarded, or are they efficiently skipped? Again, PASSAGE uses BLAZEGRAPH's indexing augmented B+Trees to initialize the departure point of each triple/quad pattern scan. A counter maintained within each node of the indexes allows PASSAGE to directly navigate to the desired offset in logarithmic time. Consequently, even on large datasets, the impact of OFFSET on triple/quad pattern evaluation remains marginal. It is worth noting that PASSAGE could use other SPARQL engines providing such a convenience, such as HDT [6].

In summary. Before PASSAGE, a timeout was an event to dread; it meant the query had failed, and the results would not be delivered as expected. With PASSAGE, a timeout becomes an event to embrace: it marks the moment when the first results arrive, progress becomes measurable, and the remaining work can be estimated.

3. PASSAGE X COMUNICA: Beyond Core SPARQL

Currently, PASSAGE provides continuation queries only for core SPARQL queries, i.e., queries with projections, triple/quad patterns, joins, unions, optionals, binds, and filters [3]. PASSAGE cannot directly execute more complex queries like the query Q_{top} depicted in Figure 3a that retrieves the top 10 brightest stars, since this query includes a property path, an ORDER BY, and a DISTINCT modifier.

With PASSAGE X COMUNICA, we extended the COMUNICA smart client with PASSAGE to support full SPARQL 1.1 queries while ensuring termination. We declared the capacities of the PASSAGE endpoint within COMUNICA, enabling the client to decompose SPARQL queries so that only supported subqueries are delegated to PASSAGE.

Figure 3 shows the physical plan of Q_{top} using PASSAGE X COMUNICA. Each green square represents a call to PASSAGE. When multiple green squares appear on the same line, this indicates that several continuation queries were required to complete a subquery. The plan involves several inner joins, and

Figure 4: PASSAGE (/passage) and BLAZEGRAPH (/sparql) run on the same server and the same data. Users can enjoy the best of both worlds.

the query is processed as follows:

1. COMUNICA transitively retrieves all subclasses (`wdt:P279`) of stars (`wd:Q523`) from the PASSAGE server;
2. For each found subclass `?x`, the original query Q_{top} is rewritten by replacing the property path pattern with the triple pattern `{?star wdt:P31 ?x}`;
3. The resulting basic graph pattern now conforms to core SPARQL and can be executed in full by PASSAGE, as shown in Figure 3b;
4. The ORDER BY and LIMIT clauses are applied as solution modifiers on the aggregated results and executed client-side by COMUNICA.

Executing Q_{top} directly on Wikidata results in a timeout. However, with PASSAGE x COMUNICA, the query completes successfully in 19 minutes. It required 689 service queries, transferred 683KB from the client to the server, and 987KB from the server to the client, ultimately retrieving the top 10 brightest stars: the Sun, SN 1054, Tycho's Supernova, Sirius A, Sirius, Canopus, Alpha Centauri, Arcturus, Alpha Centauri A, Vega. [4] The decomposition may generate many service calls, but termination is guaranteed.

4. Demonstration (https://youtu.be/_yFwC0UAeqA)

We deployed a public endpoint containing the 13B statements of Wikidata as of February 13th, 2025 in a 1.8TB BLAZEGRAPH journal. The server embeds two SPARQL query engines within a single Java Virtual Machine: the standard BLAZEGRAPH query engine[5] and our PASSAGE query engine[6].

We deployed a web user interface available at https://passage-org.github.io/passage-comunica/. It includes an interactive widget that displays the PASSAGE x COMUNICA physical plan, updated in real-time. Users can type their SPARQL queries or choose among query examples from Wikidata or WDBench [7]. [7]

Thanks to this setup, we can fairly compare BLAZEGRAPH and PASSAGE side by side on the same hardware and data, as illustrated in Figure 4. Both engines can execute the same query; users need to change the endpoint URL: use the `/sparql` suffix for BLAZEGRAPH and `/passage` for PASSAGE.

If your SPARQL query can be executed with BLAZEGRAPH, we recommend using it. Otherwise, switch to PASSAGE to retrieve partial results, observe progression, diagnose issues such as suboptimal join orders, and estimate whether the final results are worth waiting for. There is no need to choose one over the other; this setup offers the best of both worlds.

The demo is fully live, and users are encouraged to submit their queries and request explanations about the results and execution behavior.

[4] Q_{top} was executed after clearing the server's cache.
[5] https://10-54-2-226.gcp.glicid.fr/wikidata/sparql with a 60-second timeout.
[6] https://10-54-2-226.gcp.glicid.fr/wikidata/passage with a 60-second timeout and 10,000 results limit.
[7] The web interface, as well as endpoints with smaller datasets, such as WatDiv [8] or WDBench [7], can also be deployed locally. All code is publicly available on the GitHub platform at https://github.com/passage-org.

5. Conclusion

PASSAGE redefines the meaning of timeouts during SPARQL query processing. Rather than signaling failure, it becomes an opportunity: the moment when partial results become visible, offering users both progress and perspective. Importantly, in our deployment, PASSAGE and BLAZEGRAPH are not alternatives but complements; they run on the same server, the same data, and combine their strengths in a unified system.

Our roadmap includes extending support in PASSAGE for more advanced SPARQL features, such as aggregates (COUNT/COUNT DISTINCT), GROUP BY, and property paths. In addition, we plan to enhance performance through parallel execution by partitioning single triple/quad patterns using OFFSET with LIMIT.

Acknowledgments

We would like to thank Erwan Boisteau-Desdevises and Izzedine Issa Ahmat for their valuable support and contributions to this work. This work is supported by the French ANR project MeKaNo – Search the Web with Things (*ANR-22-CE23-0021*).

Declaration on Generative AI

During the preparation of this work, the authors used https://chatgpt.com in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] C. B. Aranda, A. Hogan, J. Umbrich, P. Vandenbussche, SPARQL Web-Querying Infrastructure: Ready for Action?, in: The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II, volume 8219 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 277–293. doi:10.1007/978-3-642-41338-4_18.

[2] T. H. T. Pham, G. Montoya, B. Nédelec, H. Skaf-Molli, P. Molli, Passage: Ensuring Completeness and Responsiveness of Public SPARQL Endpoints with SPARQL Continuation Queries, in: Proceedings of the ACM on Web Conference 2025, WWW '25, Association for Computing Machinery, New York, NY, USA, 2025, p. 47–58. doi:10.1145/3696410.3714757.

[3] J. Pérez, M. Arenas, C. Gutiérrez, Semantics and complexity of SPARQL, ACM Trans. Database Syst. 34 (2009) 16:1–16:45. doi:10.1145/1567274.1567278.

[4] R. Taelman, J. V. Herwegen, M. V. Sande, R. Verborgh, Comunica: A Modular SPARQL Query Engine for the Web, in: The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II, volume 11137 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 239–255. doi:10.1007/978-3-030-00668-6_15.

[5] C. B. Aranda, A. Polleres, J. Umbrich, Strategies for Executing Federated Queries in SPARQL1.1, in: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II, volume 8797 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 390–405. doi:10.1007/978-3-319-11915-1_25.

[6] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, M. Arias, Binary RDF representation for publication and exchange (HDT), J. Web Sem. 19 (2013) 22–41. doi:10.1016/j.websem.2013.01.002.

[7] R. Angles, C. Buil-Aranda, A. Hogan, C. Rojas, D. Vrgoč, WDBench: A Wikidata Graph Query Benchmark, in: The Semantic Web – ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23–27, 2022., Springer-Verlag, Berlin, Heidelberg, 2022, p. 714–731.

[8] G. Aluç, O. Hartig, M. T. Özsu, K. Daudjee, Diversified Stress Testing of RDF Data Management Systems, in: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I, volume 8796 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 197–212. doi:10.1007/978-3-319-11964-9_13.