

PING: Progressive Querying on RDF Graphs

Angela Bonifati¹, Stefania Dumbra², Haridimos Haridimos^{3,†}, Georgia Troullinou³
and Gianni Vassiliou⁴

¹Lyon 1 University, CNRS Liris and IUF, Lyon, France

²SAMOVAR/Inst. Polytechnique de Paris, ENSIE, Evry, France

³FORTH-ICS, Heraklion, Greece

⁴ECE-HMU, Heraklion, Greece

Abstract

The exact evaluation of queries over RDF data has been extensively studied. However, in a wide array of applications, RDF queries do not even terminate, due to performance reasons. To address this, in this demonstration, we present PING, a novel system built on top of Spark, which allows to progressively answer RDF queries. PING first builds a hierarchical schema structure, which is used for effective data partitioning, sub-partitioning, and indexing. Then, it employs a novel RDF query evaluation algorithm that directly locates the different levels of the hierarchy required for query answering. This also enables answering queries progressively, by sequentially visiting the various hierarchy levels. The demonstration explains the novelty of our system and shows its effectiveness and the efficiency, on both exact and progressive query answering (PQA).

1. Introduction

Graphs are simple yet powerful abstractions for representing and analyzing semantic relationships between real-world objects. Still, graph ecosystems face key challenges, such as data model heterogeneity and query answering efficiency on large, highly interconnected datasets.

The problem. While exact query answering on RDF data has received a lot of attention in recent years, performance problems are widespread, as shown by empirical analyses of SPARQL query logs [1]. Several queries of publicly available SPARQL endpoints, such as Wikidata and DBpedia, are actually timed out, due to the fact that their evaluation on the entire RDF graph is time-consuming. As such, approaches have emerged trying to ensure the termination of queries by introducing restricted servers such as TPF [2], SAGE [3] and SmartKG [4]. However, these require a smart client to perform key operations, such as joins, and shipping intermediate results from the server to the client might require overall more time to finally evaluate the query.

Nonetheless, distributed big data infrastructures like Spark have emerged and offer increased efficiency. Indeed, Spark has been exploited for efficient query answering [5], by employing partitioning techniques, precomputing joins, and constructing indexes to reduce the amount of


ISWC 2023 Posters and Demos: 22nd International Semantic Web Conference, November 6–10, 2023, Athens, Greece

*Corresponding author.

✉ angela.bonifati@univ-lyon1.fr (A. Bonifati); stefania.dumbra@ensie.fr (S. Dumbra); kondylak@ics.forth.gr (H. Haridimos); troulin@ics.forth.gr (G. Troullinou); giannisvas@ics.forth.gr (G. Vassiliou)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

data needed for query answering. Such works like SPARQLGX [6], S2RDF [7], and WORQ [8] adopt simplistic partitioning schemes and fail to exploit multi-level hierarchical partitioning for exact query answering. However, on big RDF graphs users still have to wait for a considerable amount of time before seeing the first answer to their queries. One reason for this is that query answering on interconnected data typically requires loading large chunks of it. *Currently, an approach for progressively returning query results to users is lacking.*

Our solution. To tackle this problem, in this demo, we present the novel PING system that leverages schema information to efficiently identify the data fragments required to return the first part of the answer and to progressively return the remaining parts, thus enabling **progressive query answering (PQA)**. While schemas have been successfully used to represent RDF graphs [9, 10, 11], *ours is the first work to exploit these for fine-grained graph partitioning and progressive query answering.* The code base of PING is open source and the used datasets/queries are available online¹ as well as a video demonstrating its functionality².

2. PING System Overview

We depict the global architecture of our PING system in Figure 1 (top). The framework comprises three main parts. The *GUI* allows users to select a pre-loaded dataset or add a new one, visualize statistics regarding their partitioning, write SPARQL queries, and inspect diagrams depicting the efficiency and accuracy of evaluating them with PING’s progressive query answering module. The *query processor* exploits the hierarchical partitioning in order to perform progressive query answering. The *partitioner* processes the chosen dataset, extracts its hierarchical schema, and generates hierarchical partitions, as well as sub-partitions and indexes. We will focus hereafter on the *partitioner* and *query processor*, which are PING’s core components.

Hierarchical Dataset Partitioning. For capturing the underlying structure of an RDF dataset we leverage characteristic sets [12]. The characteristic set of a node is the set of all predicates, i.e., outgoing edges, attached to it. Such characteristic sets exhibit *hierarchical relationships*, due to overlaps in their sets of properties. For example, in Figure 1(a), the characteristic sets for the Protein nodes are identified and placed in a corresponding hierarchical structure. PING visits all instances, constructs their characteristic sets, and then constructs a CS hierarchy H , as shown in Figure 1(b). Based on H , we construct a multi-level partitioning L of the initial graph G comprising partitions L_i ; these regroup all instances whose characteristic set belongs to the i level of H (Figure 1(c)). The partitions are computed once, by assigning instances to their respective level, and enjoy the *modularity* and *losslessness* properties by construction. These state that the sets of instances corresponding to the hierarchical partitioning levels are pairwise disjoint and, respectively, that they contain all triples in the initial dataset, i.e., $L_i \cap L_j = \emptyset$, for all $i, j \leq |H|$ (modularity) and $L = \bigcup_{i \leq |H|} L_i$ (losslessness).

Sub-partitioning. On top of partitioning, we also implement, for each partition, a vertical partition (*VP*) step, called *sub-partitioning*, in order to further reduce the size of the data touched at query answering. For this, we split the triples of each partition L_i , into multiple vertical partitions $L_i[p]$, one per predicate p . Each vertical partition contains the subjects and the objects

¹https://anonymous.4open.science/r/PING_ISWC_2023-B9F3/README.md

²<https://tinyurl.com/ISWCPING>

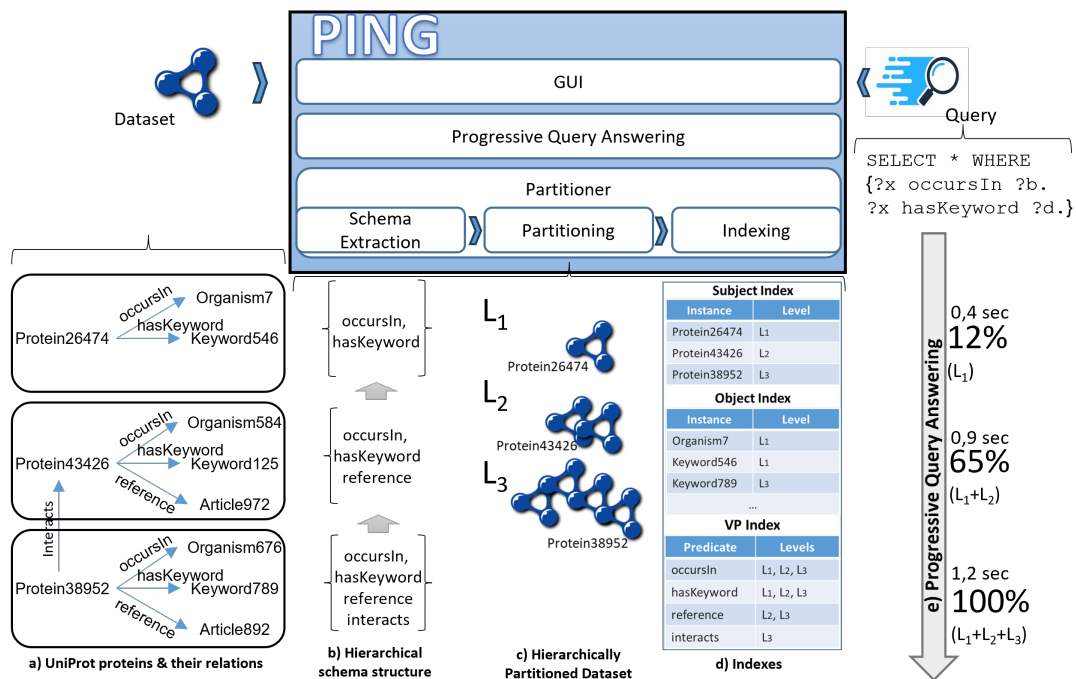


Figure 1: High-level architecture and example

for a single predicate, enabling a more fine-grained selection of data at query time.

Indexing. To speed up query evaluation, we generate appropriate indexes, so that the necessary sub-partitions can be directly identified during query execution. PING constructs property, subject, and object indexes (*VP*, *SI*, and *OI*, respectively as shown in Figure 1(d)). Thus, we can directly identify to which partitions each such instances belong.

Progressive Query Answering and Visualization. In order to perform PQA, PING implements a novel algorithm. This iterates over all query triple patterns and inspects all their symbols. Depending on whether they correspond to a predicate, to a subject, or object constant, it then inspects the corresponding vertical partitioning and index structures to collect the set of all levels whose instances mention them. Note that a query can be (at least partially) answered on a particular set of levels, if the latter contains all its triple pattern symbols; we call this set a slice. Next, PING computes the minimal, duplicate-free, slice that covers all query symbols, by taking the intersection of all such slices. This is then used for query answering, by iterating over the cartesian product of its individual triple pattern levels.

As shown in Figure 1(e), the user can select to execute the query only on a subset of the levels, trading efficiency for accuracy, as only a subset of the results will be returned. *To the best of our knowledge, the PING system is the first to enable this trade-off for performing PQA over KGs.*

3. Demo Overview

To demonstrate the functionalities of PING, we will use three synthetic datasets, i.e., Uniprot (3GB), WatDiv (13GB & 100GB), and LDBC Social Network Benchmark (18GB), and a real one, DBpedia (30GB). For each dataset, we will use 6 example queries of different shapes: 2 star, 2 chain, and 2 complex queries. The demonstration will proceed in six phases:

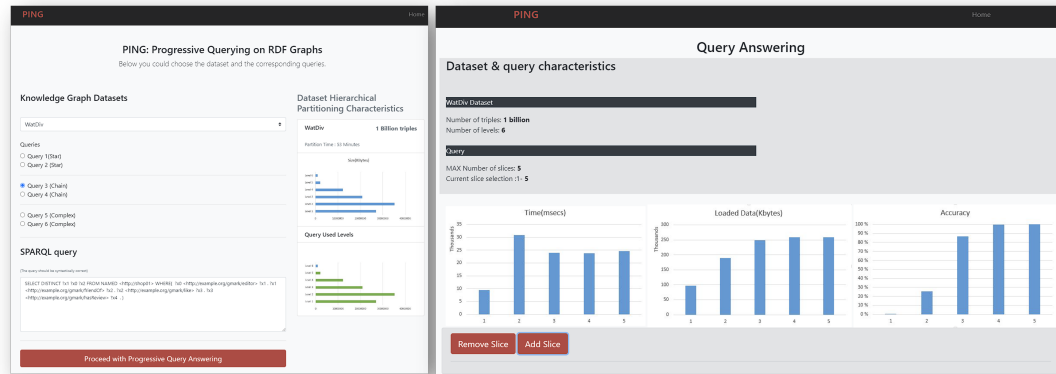


Figure 2: A screenshot of the PING system

1. Overview. The demonstration will start by explaining the various choices made for data partitioning, sub-partitioning, and indexing. We will highlight that the characteristics of hierarchical partitioning are dataset dependent, by inspecting the partitioning of each dataset, the distribution of its triples across the various levels, as well as its sub-partitions and indexes.

2. Progressive Query Answering. We will then focus on PQA. Through the GUI, the user will be able to select example queries and also set the number of partitions on which this query will run. We will visualize the result and also we will discuss the trade-off between accuracy (percentage of returned results vs. the total results) and execution time. We will show that by increasing the number of visited partitions more data are added to the result and, thus, the query answering accuracy improves, albeit resulting in an increase in execution time as well.

3. Exact Query Answering. When all partitions are used, the query can be answered with 100% accuracy. We will discuss the impact of our partitioning/sub-partitioning/indexing scheme on answering queries considering the entire dataset. We will also show the comparative performance of PING with respect to the state-of-the-art S2RDF and WORQ systems, empirically establishing that our method boosts performance for all types of queries (star, chain, complex).

We illustrate the demonstration scenario in Figure 2. In the left screenshot, the user first chooses the WatDiv dataset, whose characteristics are computed and displayed in the right-hand side panel. As captured by the corresponding histogram, the dataset contains 1B triples and is partitioned by PING into 6 levels, of varying sizes. Next, the user selects, from the associated queries of different complexities, the chain-shaped Query 3, shown at the bottom of the screen. PING analyzes it and indicates in green which are the valid slices on which the query can be partially answered. PING can perform PQA on any subset of these slices and report the

runtime, memory consumption, and accuracy (right screenshot). In our example, the user chose to evaluate on all 5 slices and can inspect the corresponding statistics. In general, slices can be freely added or dropped, following the desired balance between efficiency and accuracy.

To conclude, in this demonstration we present PING, the *first system enabling progressive query answering over KGs*. PING uses a hierarchical schema structure to partition KGs and enables progressive query evaluation. As such, it offers minimal latency and allows trading query accuracy for efficiency.

Acknowledgments

This research project was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “2nd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers” (iQARuS Project No 1147) and by the SafePolymed (GA 101057639) EU project.

References

- [1] A. Bonifati, W. Martens, T. Timm, An analytical study of large SPARQL query logs, *VLDB J.* 29 (2020) 655–679.
- [2] R. Verborgh, M. V. Sande, O. Hartig, J. V. Herwegen, L. D. Vocht, B. D. Meester, G. Haendonck, P. Colpaert, Triple pattern fragments: A low-cost knowledge graph interface for the web, *J. Web Semant.* 37-38 (2016) 184–206.
- [3] T. Minier, H. Skaf-Molli, P. Molli, Sage: Web preemption for public SPARQL query services, in: *WWW, ACM*, 2019, pp. 1268–1278.
- [4] A. Azzam, J. D. Fernández, M. Acosta, M. Beno, A. Polleres, SMART-KG: hybrid shipping for SPARQL querying on the web, in: *WWW, ACM / IW3C2*, 2020, pp. 984–994.
- [5] G. Agathangelos, G. Troullinou, H. Kondylakis, K. Stefanidis, D. Plexousakis, RDF query answering using Apache Spark: Review and assessment, in: *ICDE Workshops*, 2018.
- [6] D. Graux, L. Jachiet, P. Genevès, N. Layaida, SPARQLGX in action: Efficient distributed evaluation of SPARQL with Apache Spark, in: *ISWC*, 2016.
- [7] A. Schätzle, M. Przyjaciół-Zablocki, S. Skilevic, G. Lausen, S2RDF: RDF querying with SPARQL on Spark, *PVLDB* 9 (2016) 804–815.
- [8] A. Madkour, A. M. Aly, W. G. Aref, WORQ: workload-driven RDF query processing, in: *ISWC*, 2018, pp. 583–599.
- [9] K. Kellou-Menouer, N. Kardoulakis, G. Troullinou, Z. Kedad, D. Plexousakis, H. Kondylakis, A survey on semantic schema discovery, *VLDB J.* 31 (2022) 675–710.
- [10] N. Kardoulakis, K. Kellou-Menouer, G. Troullinou, Z. Kedad, D. Plexousakis, H. Kondylakis, Hint: Hybrid and incremental type discovery for large RDF data sources, in: *SSDBM, ACM*, 2021, pp. 97–108.
- [11] M. Meimaris, G. Papastefanatos, Hierarchical characteristic set merging for optimizing SPARQL queries in heterogeneous RDF, *CoRR abs/1809.02345* (2018).
- [12] T. Neumann, G. Moerkotte, Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins, in: *ICDE, IEEE Computer Society*, 2011, pp. 984–994.