A Tool for Efficiently Processing SPARQL Queries on RDF Quads

Anas Katib, Praveen Rao, Vasil Slavov
anaskatib@mail.umkc.edu, raopr@umkc.edu, vgslavov@mail.umkc.edu

University of Missouri-Kansas City

Abstract. We present a tool called RIQ (**R**DF **I**ndexing on **Q**uads) for efficiently processing SPARQL queries on large RDF datasets containing quads. RIQ's novel design includes: (a) a vector representation of RDF graphs for efficient indexing, (b) a filtering index for efficiently organizing similar RDF graphs, and (c) a *decrease-and-conquer* strategy for efficient query processing using the filtering index to discard non-matching RDF graphs early on and query rewriting and optimization. During the demo, the user will visually gain insights on how RIQ's unique design leads to fast processing of SPARQL queries on named graphs in a real dataset.

1 Introduction

Today, datasets with billions of RDF statements are widely available for developing new Semantic Web applications (*e.g.*, Billion Triples Challenge 2012 (BTC 2012)[1], Bio2RDF[2]). Such datasets contain RDF statements expressed as quads where each statement has a *subject*, *predicate*, *object*, and *context*. Context is powerful to model provenance information, especially when integrating data from multiple sources. Quads with the same context can be modeled as a directed, labeled graph. When the context is an IRI, a dataset of quads is a collection of RDF *named graphs*. Here is an example of an RDF statement (from BTC 2012) in RDF 1.1 N-Quads: `<http://dbpedia.org/resource/Oswego> <http://dbpedia.org/ontology/country> <http://dbpedia.org/resource/United_States> <http://dbpedia.org/data/Oswego.xml>`. Interestingly, the notion of a named graph simplifies the maintenance of provenance information for entities in triples contained within the same graph (e.g., using PROV-DM[3]). That is, the provenance triples for the entities can be stored either in the same named graph or a different named graph.

The keyword `GRAPH` in a SPARQL query restricts the matching of a basic graph pattern (BGP) to within a particular RDF named graph. If this keyword is followed by a variable, then the variable is bound to an IRI (of a named graph), the matching is done over all the named graphs in the dataset. Further, provenance information can be queried for the bindings by expressing joins via triple patterns within the same named graph as shown in this example:

[1] http://km.aifb.kit.edu/projects/btc-2012 [2] http://bio2rdf.org
[3] https://www.w3.org/TR/prov-dm

```
SELECT ?x ?y ?z ?p1 ?p2 ?g WHERE { GRAPH ?g {
  ?x <http://dbpedia.org/ontology/country> ?y .
  ?x <http://dbpedia.org/ontology/populationTotal> ?z .
  ?x prov:wasGeneratedBy ?p1 . ?p1 prov:wasAssociatedWith ?p2 . }}
```

While many techniques have been proposed for indexing and query processing of large RDF datasets containing (*subject*, *predicate*, *object*) triples [6, 1, 2, 8] on a single machine, little attention has been paid towards indexing large-scale RDF datasets with quads. A reification approach can be used to represent quads as triples and use an existing approach like RDF-3X [6]. However, doing so leads to poor query performance due to increase in the dataset size and the number of triple patterns in a query [4]. Another interesting observation is that none of the previous approaches has investigated how large, complex BGPs (*e.g.*, containing undirected cycles) in SPARQL queries can be processed efficiently on large RDF datasets. State-of-the-art approaches for a local environment were slow in processing SPARQL queries containing large, complex BGPs on RDF datasets with over a billion RDF statements [4]. This is because of the large number of join operations that must be performed to process a query, which increases the cost of query optimization and execution.

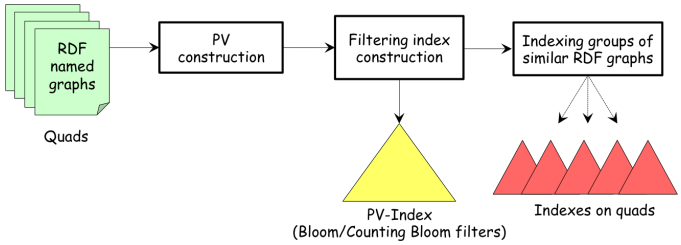
Motivated by the above reasons, we developed RIQ [4] for efficiently processing SPARQL named graph queries on RDF quads. RIQ yielded superior performance for SPARQL queries on named graphs compared to Virtuoso and Jena TDB, both of which support quads. It also outperformed a reification approach using RDF-3X as the system to process triples. (While RQ-RDF-3X [5] built quad indexes to support queries on quads and reification statements, its implementation did not support named graph queries.) This demo paper is based on our previously published work on RIQ [7, 4].

2 RIQ

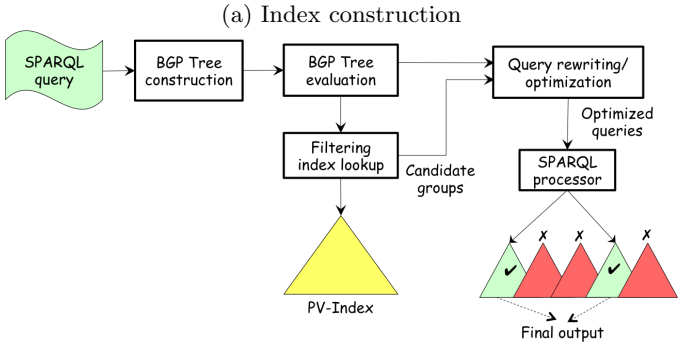
We provide an overview of RIQ's novel design to efficiently process SPARQL `select` queries on RDF named graphs and highlight our key contributions. Figures 1(a) and 1(b) show the steps during indexing and query processing in RIQ.

The first design feature of RIQ is a new representation for an RDF graph called a Pattern Vector (PV) to facilitate efficient indexing and filtering during query processing. A PV is a vector of vectors, one for each pattern in {SPO, $SP?$, $S?O$, $?PO$, $S??$, $?P?$, $??O$}. These patterns are based on the nature of triple patterns that can appear in a BGP of a query. To construct the PV of an RDF graph, for each quad, we generate a tuple, one for each canonical pattern. Each tuple is hashed, and the hash value is stored in the vector for the canonical pattern. A PV is also constructed for a BGP in a query. Essentially, the concept of PVs allow us to map a quad/triple in the data and a triple pattern in a query to a common plane of reference. This will enable us to quickly test if a triple pattern in a BGP has a match in the data.

The second design feature of RIQ is the PV-Index to effectively organize millions of PVs in the database. This index provides two benefits: First, it creates



(a) Index construction



(b) Query processing

Fig. 1. Indexing and query processing steps in RIQ

groups of similar RDF graphs efficiently, which are indexed separately. Second, it serves as the filtering index to speed up query processing. We employ locality sensitive hashing [3] on the PVs to group similar PVs efficiently (and as a result, similar RDF graphs) with high probability. We summarize the PVs in a *group* using the notion of a union operation on PVs; this preserves a necessary condition for finding a BGP match and is discussed later. The individual vectors in a PV are kept sorted. So we can compute the union of the PVs in linear time. The union of a group of similar PVs is stored compactly using a combination of Bloom filters (BFs) and Counting Bloom filters (CBFs). The BFs and CBFs for all the groups constitute the PV-Index. For each group, we also store the ids of the graphs belonging to it and index these graphs (including the context of each quad) using a SPARQL processor that supports named graphs/quads.

The third design feature of RIQ is a streamlined approach for query processing via a *decrease-and-conquer* strategy: the PV-Index is searched to quickly discard most of the non-matching RDF graphs early on during query processing followed by methodical query rewriting and optimization. A query plan called a BGP Tree is generated to process individual BGPs and constructs like `UNION`, `OPTIONAL`, etc. A key *necessary condition* that forms the basis for BGP matching is as follows: If a BGP has a subgraph match in an RDF graph, then for every canonical pattern, its vector in the BGP's PV is a subset of its vector in the graph's PV. Thus, we identify a superset of RDF graphs that contain a subgraph match for the BGP. On each group of the PV-Index, we evaluate the BGP Tree and test the necessary condition for the BGPs using the group's BFs/CBFs. We

methodically rewrite the original query, generate optimized SPARQL queries for the candidates identified after filtering, and execute these queries (using a SPARQL processor that supports quads) to produce the final (exact) output.

3 Demonstration Scenarios

RIQ was implemented using C++, Java, Python, and Django. A user will visually experience three scenarios and gain insights on how and when the *decrease-and-conquer* approach of RIQ leads to superior performance than its competitors. The user will use the **BTC 2012** dataset containing about 1.4 billion RDF statements. He/she will select prebuilt indexes and a few statistics related to index construction will be shown. Next, the user will choose (or edit) from a variety of SPARQL queries on named graphs/quads. These queries will contain large, complex BGPs, small BGPs, or multiple BGPs with keywords like `UNION` and `OPTIONAL`. The queries will also have different selectivities based on the number of named graph matches in the dataset. The wall-clock time taken by RIQ and its competitors will be shown. The user will also observe the benefit of RIQ's streamlined approach to query processing and visualize key intermediate steps. Finally, the user will observe use cases when RIQ is a better choice for local query processing in federated SPARQL queries. A video showing the features of RIQ is available at `http://vortex.sce.umkc.edu:8080/RIQ`.

Acknowledgments: This work was supported by the National Science Foundation under Grant Nos. 1115871 and 1620023. We thank Vinutha Nuchimaniyanda for her assistance.

References

1. M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. Matrix "Bit" loaded: A scalable lightweight join query processor for RDF data. In *Proc. of the 19th WWW Conf.*, pages 41–50, 2010.
2. M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee. Building an efficient RDF store over a relational database. In *Proc. of 2013 SIGMOD Conference*, pages 121–132, 2013.
3. T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the Web. In *Proc. of 11th WWW Conf.*, pages 432–442, 2002.
4. A. Katib, V. Slavov, and P. Rao. RIQ: Fast processing of SPARQL queries on RDF quadruples. *Journal of Web Semantics*, 37(C):90–111, 2016.
5. J. Leeka and S. Bedathur. RQ-RDF-3X: going beyond triplestores. In *Proc. of the 5th Intl. Work. on Data Engineering Meets the Semantic Web*, pages 263–268, 2014.
6. T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1):91–113, 2010.
7. V. Slavov, A. Katib, P. Rao, S. Paturi, and D. Barenkala. Fast processing of SPARQL queries on RDF quadruples. In *Proc. of WebDB '14*, pages 1–6, 2014.
8. P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu. TripleBit: A fast and compact system for large scale RDF data. *In Proc. of VLDB Endow.*, 6(7):517–528, 2013.