Self-organizing Structured RDF in MonetDB

Minh-Duc Pham #1
supervised by Peter Boncz #2

CWI, Amsterdam, The Netherlands
[1]duc@cwi.nl
[2]boncz@cwi.nl

***Abstract*— The semantic web uses RDF as its data model, providing ultimate flexibility for users to represent and evolve data without need of a schema. Yet, this flexibility poses challenges in implementing efficient RDF stores, leading from plans with very many self-joins to a triple table, difficulties to optimize these, and a lack of data locality since without a notion of multi-attribute data structure, clustered indexing opportunities are lost. Apart from performance issues, users of huge RDF graphs often have problems formulating queries as they lack any system-supported notion of the structure in the data. In this research, we exploit the observation that real RDF data, while not as regularly structured as relational data, still has the great majority of triples conforming to regular patterns. We conjecture that a system that would recognize this structure automatically would both allow RDF stores to become more efficient and also easier to use. Concretely, we propose to derive self-organizing RDF that stores data in PSO format in such a way that the regular parts of the data physically correspond to relational columnar storage; and propose RDFscan/RDFjoin algorithms that compute star-patterns over these without wasting effort in self-joins. These regular parts, i.e. tables, are identified on ingestion by a schema discovery algorithm – as such users will gain an SQL view of the regular part of the RDF data. This research aims to produce a state-of-the-art SPARQL frontend for MonetDB as a by-product, and we already present some preliminary results on this platform.**

I. Introduction and Motivation

The Resource Description Framework (RDF) is the main semantic web technology for publishing collections of interlinked datasets on the web. In the RDF data model, a data set is represented as a collection of $<subject, predicate, object>$ triples, in which the object can be considered as the value for the property (i.e., predicate) of the described resource (i.e., subject). This collection also forms a labeled directed RDF graph. SPARQL is the W3C recommendation query language for RDF graphs, essentially allowing subgraph search.

The semantic web proponents of RDF highlight two advantages: *(i)* it is based on URIs such that not only meta-data but also data *instances* (e.g., "keys") can be standardized for interoperability over the web and *(ii)* it is extremely flexible, so the global RDF graph (the semantic web) can be extended by everyone in a grass-roots and pay-as-you-go way.

The database community has taken a critical stance towards RDF because of *(ii)*: RDF de-emphasizes the need for a schema and the notion of structure in the data, and this leads to performance issues in systems that manage large amounts of RDF data. Specifically, RDF store relies on triple tables which leads to query plans with many self-joins. Also, the lack of a multi-attribute object structure in triple storage blocks the use of advanced relational physical storage optimizations, such as clustered indexing, hash/range data partitioning, etc. Yet, studies of actual datasets have found that despite the fact that most RDF does not have a (RDFS) schema, the great majority of RDF triples do conform to regular structural patterns [1]. Additionally, the lack of a schema also makes it harder for users to formulate queries on RDF graphs. To tackle this latter problem, the semantic web community has recently been studying graph structure analysis techniques to construct visual graph summaries to help users comprehend RDF graphs [2].

Despite these issues, we do not dismiss RDF, because it is the unrivaled standard behind the vision of global data standardization (e.g., LOD, see point *(i)*), and simply because RDF currently does have significant traction in certain domains, such as the life sciences. With quickly growing RDF data volumes, there is a true need to better support it in database systems. In this PhD track, the key idea is to automatically recover the structure typically present in RDF data sets, and leverage this structure both internally inside the database system (in storage, optimization, and execution), and externally towards the users who pose queries. This idea will be realized and experimentally evaluated inside the open-source MonetDB column-store system[1], known for its adaptive storage structures (such as Recycling [3] and Cracking [4]) with the aim to create *self-organizing structured RDF*. Thus, apart from developing new RDF storage and SPARQL query evaluation and optimization techniques, this PhD research is a database architecture project.

In this PhD proposal, we will now identify what we see as the main problems in RDF data management (bad query plans, low storage locality and lack of user schema insight), and then move on to the general idea behind this project, its main research questions and our intended research approach.

Bad query plans. Considering a simple SPARQL query:

```
SELECT ?a ?n   WHERE {
   ?b <has_author>     ?a.
   ?b < in_year> ''1996''.
   ?b <isbn_no>  ?n   }
```

[1]www.monetdb.org

This query looks for the author and the ISBN number of a book published in 1996. Despite the fact that a book entity always has the co-occurrence of <isbn_no> and <has_author> properties, the query plan still needs a separated join for these properties to construct the answer. This problem of having unnecessary joins is serious in most SPARQL queries as they commonly ask for many properties from a subject. However, relational query processors that know about the structure of data waste no effort here. The only joins they process are "real" joins between different entities.

Besides, being unaware of structural correlations (e.g., availability of <isbn_no> causes the occurrence of <has_author> almost a certainty) also makes it difficult to estimate the join hit ratio between triple patterns, thus, with such join-intensive queries as SPARQL queries, hard to find the optimal join order.

Low storage locality. A crucial aspect of efficient analytical query processing is data locality, as provided by a clustered index or partitioning scheme. However, without the notion of classes/tables with regular columns/attributes, it is impossible to formulate a clustered index or partitioning scheme, which RDF stores therefore do not offer.

Current state-of-the-art RDF stores such as RDF-3X[5], create exhaustive indexes for all permutations of subject(S), predicate(P), object(O) triples as well as their binary and unary projections. This abundance of access paths does not create any of the access locality that a relational clustered index or partitioning scheme offers. A Scan-Select SPARQL query may use a POS index to execute a range selection quickly, but for retrieving the other attributes needs a CPU intensive nested-loop index join into a PSO index; one for each attribute. This nested-loop join will hit the index all over the place: no locality despite so-called exhaustive indexing.

Lack of user schema insight. In handling large RDF graphs with data from multiple domains, SPARQL users often experience trouble formulating queries; since for doing so one needs an understanding of the structure of this data. SPARQL does not offer specific features for querying data with unknown and variable structure, and semantic web technologies lack schema browsing and visualization tools. Besides, a general problem in the semantic web stack is that the number of tools and their maturity is low when compared to relational tool support. As such, we conjecture that a relational view of the data would help SPARQL users better comprehend the data and allow users to enjoy the benefits of the relational tool-chain, which may still be beneficial to them even though it is incomplete, as the "irregular" triples would fall out of this view.

Roadmap. The rest of this proposal is organized as follows. In Section 2, we describe the general idea behind self-organizing structured RDF, and subsequently the research questions, initial results and future approach. In Section 3, we the discuss related work, before concluding in Section 4.

II. Self-organizing Structured RDF

Our proposal is inspired by the work on so-called "characteristic sets" (CS's) [1] showing that it is relatively easy to recover large part of the implicit class structure underlying

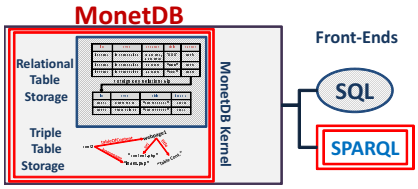


Fig. 1: RDF store architecture in MonetDB

data stored in RDF triples. Here, a characteristic set is a set of properties that occurs frequently with the same subject.

Our general idea is first to extend the initial algorithm for finding CS's and discover a rough relational schema which covers most of the input RDF triples (e.g., 85% of the dataset). This schema contains a set of CS's and foreign key relationships between them. Then, by exploiting this schema, we physically design a CS clustering scheme, focusing on *subject clustering*, that can help create real locality in selection queries, even across joins between multiple CS's. The rough relational schema could also be presented to users as a SQL view of exactly the same data. In fact, since we store data in PSO order, the subjects belonging to the same CS will re-surface as multiple, aligned, ranges in the PSO table. The storage for each such property, in fact forms a "column" so the CS could alternatively be viewed and queried as a relational table in column-store format. As shown in Figure 1, relational infrastructure is also brought into the RDF store so that the relational indexing and query optimization techniques can be applied in the RDF model.

Further, to fully leverage this storage method in SPARQL queries we propose to extend the database kernel with new query processing algorithms, that is, *RDFscan* and *RDFjoin*.

This PhD track will focus on following research questions:
i) How to efficiently and scalably detect and summarize CS's, at both bulk and trickle loads?
ii) How to generate automatic schema representations including foreign key links (relationships between CS's) with shapes and names that can be easily understood and used?
iii) How to automatically leverage relational clustered storage techniques in our self-organizing RDF storage scheme?
iv) How to integrate RDFjoin/RDFscan into a database kernel, such that it can leverage relational access methods?

A. Schema exploration and Summarization

We observe that there is quite a bit of regularity in RDF data, even in web-crawled data which is considered the dirtiest data encountered in practice. The structure typically surfaces as:

- Certain kinds of subjects have the same set of properties, i.e., CS (belong to the same class).
- Certain classes are connected, over the same kind of property paths ("foreign key" relationships)

Figure 2 shows the data structure captured from an example DBLP-like data. In this figure, in addition to the structured data, real-world RDF data also contains irregularities which may be caused by the occurrence of different schematic structures, data dirtiness, missing, or duplicated values.

Aiming to reduce the amount of CS's and enrich the schema, we extend the initial algorithm for finding CS's [1].

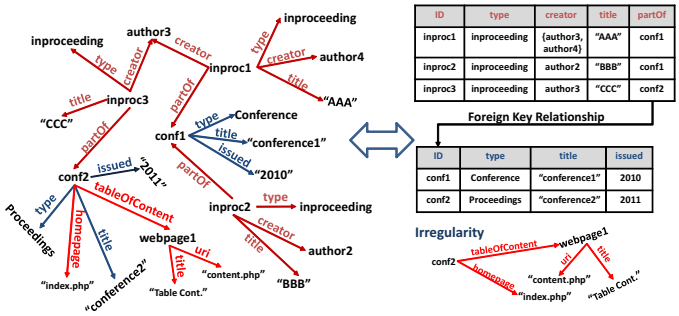


Fig. 2: Structure recognized from an example RDF graph

Generalization. In contrast to the original CS algorithm which created a different CS for each unique combination of attributes, we allow attributes of kind 0..n (NULLABLE attributes) if a significant minority fraction of the subjects has at least one occurrence. This reduces the number of CS's.

Typed Properties. After defining the initial set of CS's we further analyze the type of the literal properties that have been grouped. For literal objects, we look at the atomic type. In case of URI objects, we type them using initial CS membership. Here, we increment the number of CS's as we will create a separate CS variant for each different combination of types; the advantage being in faster processing of each CS variant, as the types of the columns are known and homogeneous.

Relationships. The above process also leads to a foreign key graph between CS's. That is, as a URI property of one CS always refers in the object field to members of one other CS, this is a foreign key between these two CS's.

Schema fine-tuning. The schema can be further fine-tuned, e.g. by reducing 0..n attributes to 0..1 or in case the multiplicity is > 2 splitting it off into a separate table (CS), or by unifying CS's that are 1-1 linked; which is often the case for blank nodes. Further, while the detection algorithm only collects CS's with high support, these CS's over relationships may refer to small (infrequent) CS's that would normally not be detected, but should be included to complete the schema. Thus, rather than looking at direct support, we add incoming links to the CS to the tally in order to determine the support of a CS.

With the explored structure, we can provide an important feature for SPARQL users: *RDF schema summarization*. The schema generated by the methods sketched above may still be quite large. Thus, we envision methods to reduce the schema size during a query session. This can be done by reducing the support thresholds, but a more advanced form is to use keyword search to identify relevant CS's. In both cases we will show a schema consisting of these selected CS's plus other CS's reachable from them over foreign key links. The system can present these reduced schemas to the SQL toolchain by extending the SQL catalog with a new artificial schema holding references only to these tables and their relationships; still allowing existing SQL tools to be used unmodified.

B. Subject clustering

While loading RDF triples, current RDF stores typically assign object identifiers (OIDs) to Subject (S), Predicate (P), Object (O) in order of appearance This order might be quite random and uncorrelated with the access paths of interest to the database users. Given the fact that the OID order (whatever it happens to be) is heavily exploited in RDF systems, this is in fact the direct cause of non-locality in RDF query plans.

Given the discovered CS's, to obtain real locality we would like to order the OIDs in a meaningful way. For S OIDs:

- we group them by characteristic sets.
- within a characteristic set, we can then further sub-order them on some index keys (i.e. property values). An extreme form of this is to adopt a multi-table clustering strategy for this ordering over foreign keys.

Similarly, the O OIDs used for literals should be ordered in a way that is meaningful to SPARQL value comparison semantics, such that comparisons on the O identifiers can be used for executing value range-predicates.

Regarding the choosing of P identifiers, there are typically few values P that are very frequent. But there may be a long tail of infrequent P's (often spelling errors or low-value noisy properties). Such P's could be treated differently than the frequent P's.

In our clustered storage, we focus on the ordering of the subject OIDs, which we call *Subject Clustering*. Figure 3 shows an example of Subject Clustering in which a loaded basic triple storage (e.g., PSO) is moved to the clustered representation following a reorganization exploiting CS's. Subjects corresponding to a CS are physically grouped together. The irregular data containing triples that do not belong to any CS is stored separately in the basic triple storage.

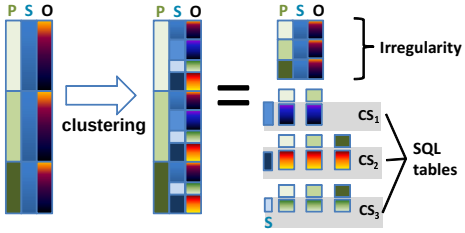
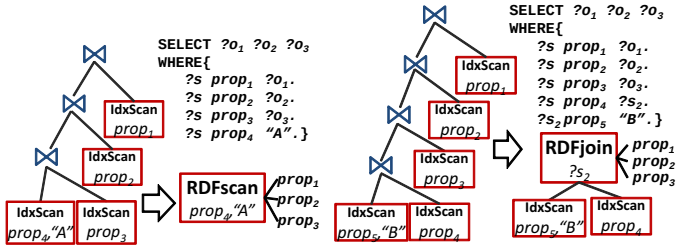


Fig. 3: Subject clustering

C. RDFscan/RDFjoin

The core idea of our novel RDF storage proposal is to store RDF data that has been recognized as conforming to a characteristic set together in an aligned way, such that for a whole stretch of subjects we get aligned stretches of Objects; as all properties will have multiplicity "exactly one" (in case of 0..1 attributes, the missing values will be represented as SQL NULL). This CS-wise storage can be exploited to provide CPU efficient algorithms which are faster than multiple merge- or index-lookup joins in processing typical star-patterns in SPARQL queries. Specifically, we propose a new operator, *RDFscan*, that delivers a tuple stream for multiple properties in one go. A slight variant is the *RDFjoin*, which does the same, but receiving a stream of candidate subjects. The latter operator was recently proposed as "Pivot Index Scan"[6], though our RDFscan/RDFjoin leverages CS-wise storage: eliminating all join effort when producing a star that stems from a single CS. Figure 4 shows how RDFscan and RDFjoin strongly reduce the number of joins in SPARQL queries.



(a) RDFscan (b) RDFjoin

Fig. 4: Example of using RDFscan and RDFjoin.

D. Preliminary Evaluation

To test our ideas, we studied queries 6 and 3 of the 10GB size RDF-H benchmark – a straight 1-1 mapping of the TPC-H benchmark to SPARQL (see *sf.net/projects/bibm*). We used the MonetDB+HSP[7] prototype that pursues a classical exhaustive indexing approach, storing RDF in 6 ordered projections on all P, S, O order permutations. This prototype was hand-modified to do subject-clustering into CS's, so LINEITEM subjects form one stretch of S-OIDs and ORDER subjects another. Further, within each CS, we applied sub-ordering: we ordered the LINEITEM and ORDERS CS-es internally on resp. the shipdate and orderdate attributes. Finally, we added in Netteza-style Zone-Maps that allow us, given an LINEITEM date selection, to find a range of referenced S-OIDs so a restriction on shipdate can be pushed to ORDERS (and vice versa a restriction on orderdate restricts LINEITEM), exploiting the strong correlation on all dates between tables. The final result is still a PSO table, but very intelligently organized. We acknowledge that a self-organizing RDF system would need workload analysis in order to derive the usefulness of such subject-clustering on dates. As RDF-H is fully regular, this can be taken as a best-case analysis of what state-of-art relational storage and execution can bring to SPARQL. In the near future, we will further test and develop our self-organizing RDF algorithms on dirty data, such as web crawls, where we expect the gain to be less, but still nonzero. The primary purpose of the experiment is to show viability and as a morale boost going into this PhD track.

TABLE I: MonetDB+HSP performance on RDF-H (SF=10), using various optimizations (time in seconds)

Query Plan	Scheme	ZoneMaps	Q3		Q6	
			Cold	Hot	Cold	Hot
Default	ParseOrder	No	37.50	19.66	28.25	6.52
	Clustered	No	18.01	15.32	9.27	3.27
		Yes	2.13	2.02	n.a	
RDFscan/ RDFjoin	ParseOrder	No	3.34	2.93	8.64	2.16
	Clustered	No	2.13	2.01	1.47	0.44
		Yes	0.89	0.78	n.a	

The results in Table I show that clustered RDF store with all optimizations including RDFscan/RDFjoin and Zone-Map index can accelerate the original MonetDB+HSP by a factor > 40. Using CPU efficient SPARQL query operators, i.e., RDFscan/RDFjoin, can help improve the performance of the system by an order of magnitude.

III. Related work

The exploration of CS's from RDF triples is somewhat discussed in existing RDF stores using the property tables approach[8], [9], [10], [11]. In this approach, each property table corresponding to a set of properties (e.g., CS) needs to be obtained from input RDF triples. However, most of the early RDF systems[8], [9] do not provide automatic methods for exploring the schema. They rely on the DBA modeling such regular data, but given that RDF graphs often contain many different structures, this limits the applicability (and observed popularity) of this approach.

Automated methods for detecting property tables from RDF triples have been proposed by Levandoski et al.[10], Wang et al.[11]. However, the methods do not attempt to find relationships between property tables as we propose, nor do they attempt to make the schemas human-readable, nor do they aim to export the regular tables as a SQL views on the data, nor do these efforts focus on leveraging such storage inside database kernels with new algorithms such as RDFjoin.

Recently, Matono and Kojima[12] construct so-called paragraph tables which are similar to property tables from adjacent RDF triples that are physically correlated. However, this method relies on well-structured input RDF documents and the parse order of RDF triples. Neumann et al.[1] extract the characteristic sets from RDF triples but merely use them for improving the cost model of a query plan.

IV. Conclusion

We propose PhD research into self-organizing RDF storage in MonetDB, where on RDF ingestion the system automatically detects structure in the data. This knowledge is then used to store the data in structured form, to accelerate RDF star join patterns through new RDFscan/RDFjoin algebraic operators, and to aid the RDF user in comprehending the data by offering a SQL view on it that elicits the schema at variable degrees of detail. The main research questions are around *i)* detecting so-called characteristic sets efficiently, *ii)* algorithms to generate a usable schema from this, *iii)* leveraging this storage schema in new RDFscan/RDFjoin operators, and *iv)* integrating these new operators in a database kernel.

References

[1] T. Neumann and G. Moerkotte, "Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins," in *ICDE*. IEEE, 2011.
[2] S. Goyal and R. Westenthaler, "Rdf gravity (rdf graph visualization tool)," *Salzburg Research, Austria*, 2004.
[3] M. Ivanova, M. Kersten, N. Nes, and R. Gonçalves, "An architecture for recycling intermediates in a column-store," *TODS*, vol. 35, no. 4, 2010.
[4] Idreos, Kersten, and Manegold, "Database cracking," in *CIDR*, 2007.
[5] T. Neumann and G. Weikum, "Rdf-3x: a risc-style engine for rdf," *VLDB*, vol. 1, no. 1, pp. 647–659, 2008.
[6] A. Brodt, O. Schiller, and B. Mitschang, "Efficient resource attribute retrieval in rdf triple stores," in *CIKM*. ACM, 2011, pp. 1445–1454.
[7] Tsialiamanis, Sidirourgos, Fundulaki, Christophides, and Boncz, "Heuristics-based query optimisation for sparql," in *EDBT*, 2012.
[8] K. Wilkinson, "Jena property table implementation," 2006.
[9] E. Chong, S. Das, G. Eadon, and J. Srinivasan, "An efficient sql-based rdf querying scheme," in *VLDB*, 2005, pp. 1216–1227.
[10] J. Levandoski and M. Mokbel, "Rdf data-centric storage," in *Web Services, ICWS 2009*. IEEE, 2009, pp. 911–918.
[11] Y. Wang, X. Du, J. Lu, and X. Wang, "Flextable: using a dynamic relation model to store rdf data," in *DASFAA*, 2010, pp. 580–594.
[12] A. Matono and I. Kojima, "Paragraph tables: A storage scheme based on rdf document structure," in *DEXA*. Springer, 2012, pp. 231–247.