On Distributed Querying of Linked Data^{*}

Martin Svoboda, Jakub Stárka, and Irena Mlýnková

XML and Web Engineering Research Group Faculty of Mathematics and Physics, Charles University in Prague Malostranske namesti 25, 118 00 Prague 1, Czech Republic {svoboda,starka,mlynkova}@ksi.mff.cuni.cz

Abstract. The concept of Linked Data has appeared recently in order to allow publishing data on the Web in a more suitable form enabling automated processing by programs and not only by human users. Linked Data are based primarily on RDF triples, which are also modeled as graph data. Despite the research effort in recent years, several questions in the area of Linked Data indexing and querying remain open, not only since the amount of Linked Data globally available significantly increases each year. Our ongoing research effort should result in a proposal of a new querying system dealing with several disadvantages of the existing approaches identified in our previous work. They are especially related to data scaling, dynamicity and distribution.

Keywords: Linked Data, RDF, indexing, querying, SPARQL.

1 Introduction

The concept of Linked Data [3] appeared in order to extend the Web of Documents towards the Web of Data. And the reason is simple – it is often not feasible to retrieve potentially structured information from traditional documents based on HTML [12] formats tailored for users and not programs.

Linked Data do not represent any particular standard; we only talk about a set of recommended principles and techniques, which lead to the publication of data in a way more suitable for their automated processing. First, each realworld entity should be described by a unique URL identifier. These identifiers can be dereferenced by HTTP to obtain information about the given entities. And, finally, these entity representations should be interlinked together to form a global open data cloud – the Web of Data.

A particular way to follow these principles is to use the RDF (Resource Description Framework) [7] standard, where data are modeled as triples conforming to the concept of *subject-predicate-object*. An alternative means to view these triples are graphs, where vertices correspond to *subjects* and *objects*, edges represent the triples themselves and are labeled by *predicates*. At the implementation level, we can publish RDF triples in a form of RDF/XML [2] syntax and

 $^{^*}$ This work was supported by the Charles University Grant Agency grant 4105/2011 and the Czech Science Foundation grant P202/10/0573.

along the data we can also publish RDFS [4] schemata or OWL [8] ontologies restraining the allowed content of such RDF data.

In recent years, a significant effort appeared not only in a theoretical research, but also in the amount of Linked Data globally available. However, we can still identify several open problems to which attention should be paid. The goal of our ongoing research effort is to propose a new querying system for Linked Data. In particular, we want to focus on indexing structures and techniques with respect to SPARQL [10], probably the most used querying language for RDF data.

The aim of this paper is to provide a description of the system we are attempting to propose. However, in order to understand our motivation, we also need to discuss the existing approaches from the area of Linked Data indexing and querying. Their thorough overview was presented in our previous work [15]. Although these approaches represent efficient systems (or at least promising interesting proposals), when we focus on large amounts of dynamic and distributed data concurrently, these approaches start showing their bottlenecks.

Preliminary ideas of our querying system were first introduced in [14]. Now, we will discuss main aspects and issues of the architecture in more detail. They are especially related to components for managing sources, distributed databases, storages for data triples and auxiliary indexing structures. Index structures in fact represent one of the crucial parts of our work, since the majority of existing methods does not assume dynamic data. When processing queries, we need to find suitable query evaluation plans, which involves the source selection and a set of optimization strategies.

Outline. In Section 2 we present a basic overview of the existing approaches. Section 3 provides the description of the architecture of system we are working on. Finally, Section 4 concludes.

2 Related Work

The existing approaches can probably be divided into three main categories: *local* querying systems, *distributed* querying systems and *global* searching engines. It is worth noting that even though we want to focus on distributed querying, its models and algorithms, wide range of relevant ideas can be found between approaches for local querying. For simplification, we will use abbreviations S, P, O and C for *subject*, *predicate*, *object* and *context* respectively.

We start our overview of existing approaches by local querying systems. Index structures proposed by Harth and Decker [5] enable querying of local data quads with context. These structures involve Lexicon (an inverted list for keywords and two-way translation maps for term identifiers based on B⁺-trees) and Quad indices (B⁺-trees for SPOC, POC, OCS, CSP, CP, OS orderings) allowing to query in all possible 16 access patterns. Despite data quads themselves, these indices also contain statistics about data.

The core of the stream processor RDF-X by Neumann and Weikum [9] is based on six B⁺-tree indices for all SPO, SOP, OSP, OPS, PSO and POS access patterns. Additionally, they also use indices with statistics (S, P, O, SP, *PS*, *PO*, *OP*, *SO* and *OS* projections) and selectivity histograms and statistics for pre-computed path or star patterns. Next, the idea of HexaStore approach by Weiss et al. [18] is based on similar *SPO*, *SOP*, *OSP*, *OPS*, *PSO* and *POS* index structures, however, these are implemented as ordered nested lists. All these lists contain only identifiers instead of strings, again.

BitMat is an approach proposed by Atre et al. [1]. Its index model is based on a matrix with three dimensions for S, P and O values (terms are translated to identifiers, which are used as matrix indices). Each cell contains a bit value equal to 1 if and only if the given triple is stored in the database, otherwise value 0. The index is organized as an ordinary file with all SO, OS, PO and PS slices stored using a bit run compression over individual slice rows.

Udrea et al. [17] introduced a model based on splitting data graphs into subgraph areas that are described by conditions limiting their content. The idea is derived from a metric defined on URIs and literals (e.g. a minimal number of edges in a data graph between a given pair of values). The index structure itself is a balanced binary tree, where internal nodes represent mentioned areas and leaf nodes store data triples conforming to these areas.

The last presented local approach is a parameterized index introduced by Tran and Ladwig [16]. Their model is based on bisimilarity relations, putting in a relation such two vertices of the data graph that share the same outgoing and ingoing edges (reflecting only predicates). Vertices from the same equivalence class have the same characteristics and, therefore, prompted queries can first be evaluated over these classes to prune required data.

Now, we move to distributed approaches. Quilitz and Leser [11] proposed a system for integrated querying over distributed and autonomous sources. The core of this approach is a language for description of distributed sources, in particular, data triples they contain, together with other source characteristics.

The purpose of a data summary index by Harth et al. [6] is to enable the source selection over distributed data sources. Data triples are modeled as points in a 3-dimensional space (S, P, and O coordinates are derived by hash functions). The index structure is a QTree based on standard R-Trees. Internal nodes act as minimal bounding boxes for nested nodes, leaf nodes contain statistics about data sources, not data triples themselves.

3 Framework

The system should provide transparent querying of distributed data – not in the context of the entire Web of Data, but only within a distributed database over which we have a full control. Linked Data are the subject of nontrivial changes in time and, thus, the aspect of the data volatility cannot be ignored in the framework architecture and index structures especially. Many existing approaches bring interesting ideas, but their indexing models only assume environments with static data. Therefore, the core part of our work is to propose an appropriate dynamic index structure.

3.1 Sources and Databases

The nature of Linked Data assumes that data are distributed within the entire global cloud of the Web of Data. Since completely centralized solutions seem not to precisely follow this idea, we want to find a suitable compromise between centralized and totally distributed approaches. For this purpose we can accept an idea that a distributed database is spread across a set of sources, as we can see in Figure 1 with a sample distributed infrastructure. Each source provides two main features – it is able to store data triples inside its local storages and provides interfaces for querying.

These sources can be viewed only as ordinary services, but we have the full control over sources we want to use in our database – either we own them completely (and decide what data they should store), or, at least, we can decide what independent sources we would like to use (and we accept data they provide). Anyway, submitting a query to a public interface of a particular source, it should transparently decompose the query into its elementary parts, decide which sources should be contacted to obtain relevant data, and, finally, to compose the entire query result. In other words, the user should define data to be used (by building its distributed database), but the query itself should be evaluated automatically without his or her explicit help.

For this purpose, we first need to have a technique for describing capabilities of individual sources. A promising concept was already introduced by Quilitz and Leser [11], as we already noted in the previous section. Anyway, we must be able to clearly describe data the given source contains. This can be achieved by a set of various conditions on triples and their S, P and O components. However, this is not an easy task, since descriptions must be as accurate as possible. But on the other hand, too complicated and big descriptions would be useless as well. Moreover, if we assume data dynamicity and query evaluation, we also need to publish statistics about given data, their versions or availability. And still we cannot end, because if we recall the second purpose of each source, we must also capture other issues of the query evaluation process. For example, if two different sources contain the same data, it would be worth to know which of them has better assumptions to execute the evaluation more efficiently.

Having defined the way how sources publish information about data they contain, we need to manage sources themselves. So, assume that we have the knowledge of sources we want to use, their locations or other technical details. Now, we must define, which sources (and, in particular, which data) constitute our database. This management seems to be easy, but cannot be omitted.

3.2 Storages and Indices

Data triples are stored in physical storages. Their role, however, might be a bit different comparing to traditional relational databases or others. The model of RDF triples is so simple that we can store data directly in indices, but as we will see, this still does not mean that physical storages should not be included in the architecture of querying systems. Although triples really are easy to grab,

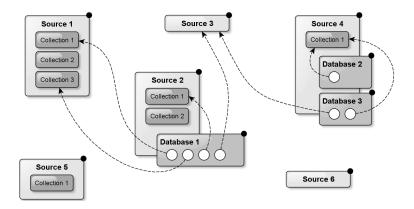


Fig. 1. Sample infrastructure with sources and distributed databases

it would be misleading to think that we do not need to handle different data differently. Relational databases allow users to create schemata and explicitly declare how their data should be stored in relational tables. However, we are not offered similar features in existing native approaches for RDF data.

Therefore, we assume that a storage is a component for storing RDF triples, but we do not have any further assumptions on their internal structure or characteristics. The only important is to comply with an agreed public interface. As a consequence, we can work with native storages, we can create wrappers around relational databases, or even to access remote storages via network. In other words, it would be interesting to access local storages within a particular source with the same (or at least very similar) interfaces as we would use between distributed sources during the query evaluation phase. In fact, we indeed need to achieve this behavior, since if we formulate a query on a given source against a particular database, apparently, some data may be available locally and other not – but from the point of the query evaluation process, both these data play the same role. A sample storages configuration can be seen in Figure 2.

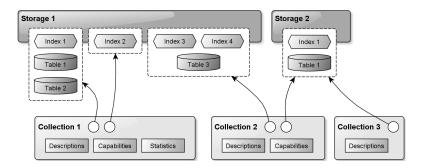


Fig. 2. Sample collections and storages composition

The shared idea by the majority of indexing methods is the way of storing string values of URIs and literals, because there is a high probability that strings (or substrings) may have multiple occurrences in the database. Therefore, it is very effective to store these strings only once in a special storage, assign them unique integer identifiers, and use them in RDF triples instead of the original terms. As a consequence, frequently executed value equality tests during the query evaluation may then be executed much faster and the space required for storages decreases as well.

Having a particular domain of our problem, we should know at least something about data and even queries, and, thus, to design our database effectively. We do the same years and years in relational databases, so we should be able to select appropriate storages and indexing structures directly for our situation in RDF approaches, too. This functionality should be one of the core parts of our system. When storing data in a local storage of a source within a given database, users should be encouraged to choose from a palette of implemented approaches, best conforming to their situation.

The main disadvantage of the majority of interesting models for indexing RDF data is the static nature of indexing structures themselves. We do not enable working with extremely volatile data, but it is not a good idea to strictly assume only a static database. Therefore, one of our main goals is to extended some existing approach [1, 13] towards support for adding, modifying and removing data from storages and associated indices. Another problem could represent the necessity to heuristically configure indices. For example, in the index structure proposed by Tran and Ladwig [16], we need to define sets of predicates that are used for restricting ingoing and outgoing edges from vertices when constructing equivalence classes based on a relation on vertices. Unfortunately, this configuration may not always be feasible easily.

3.3 Queries

We have chosen SPARQL as a querying language. Having a query statement, we first need to parse it into an internal representation of a graph pattern. For simplicity, we can assume that this pattern is built from a set of triples, where we can use variables instead of only fixed terms. They may serve for joining individual patterns together, or may only state that we do not care about values of a corresponding triple particle. The latter purpose in fact corresponds to the idea of joining – if we first evaluate each pattern separately, then we need to join intermediate results together – joining only those pairs of triples that have equal values at positions of corresponding variables.

The problem is that our database is distributed between several sources. We have descriptions of data these sources provide, thus, we need to decide for each pattern (or even more complicated subqueries), where relevant data are located. This problem is referred as a source selection. Whereas data summary index by Harth et al. [6] works with detected summaries about data, we would like to rely on discussed descriptions. However, relevant data can be split between sources in different ways, or they can even partially or fully overlap. Therefore, this source selection is not always simple and we must take care what sources we want to access. We can see a sample query evaluation in Figure 3.

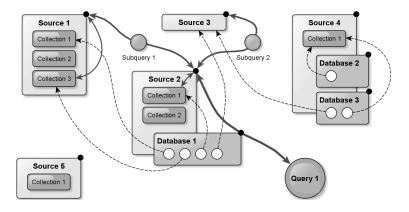


Fig. 3. Distributed evaluation process of a sample query

When we have decided which sources contain relevant data, we are still not finished with preparing the query evaluation plan. Data in sources are physically maintained in storages and these storages may have different capabilities to return required data. Thus, during the source selection, we also need to consider these capabilities. And moreover, different indices may be available, too.

If we want to find a suitable query evaluation plan, which is a must, since the complexity of different plans may be significantly different, we have to consider all the following aspects: different sources, storages, indices and different algorithms available for executing operations. The theoretical goal could be to find the optimal plan, but in practice we must settle only for approximations. Usually, we cannot inspect all possible plans, so we have to use only suitable heuristics. Another problem is that we are often forced to use incomplete and only approximate statistics.

The general idea of all optimizations is to avoid processing of irrelevant data wherever possible and to perform all computations effectively. If the existing approaches are not able to directly access only the required data, they at least attempt to prune data using other methods or ideas. For example, we can move data filtering selections as close as possible to their fetching, or we can perform data pruning before the phase of joining. Probably the most important position in query optimization techniques has the join ordering. It is quite interesting that we can use similar ideas to the nested loop algorithm from relational databases. However, there are other aspects that need to be considered, too.

4 Conclusion

In this paper we described the architecture of the querying system we are proposing. Issues related to this architecture can be divided into two groups: data and queries. First, we discussed observations and ideas related to the model of a distributed database spread across a set of selected sources, motivation and features of physical storages for RDF triples and indexing structures supporting the query evaluation process. Finally, this process must discuss methods for finding optimal query evaluation plans, selection of relevant distributed sources and a set of optimization techniques, too.

Although the existing solutions already focus on the same area, these approaches do not target at all three main open challenges concurrently – data scaling, distribution and dynamicity.

References

- Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix "Bit" loaded: A Scalable Lightweight Join Query Processor for RDF Data. In: Proceedings of the 19th Int. Conf. on World Wide Web. pp. 41–50. WWW '10, ACM, NY, USA (2010)
- Beckett, D.: RDF/XML Syntax Specification (Revised) (2004), http://www.w3. org/TR/rdf-syntax-grammar/
- Bizer, C., Heath, T., Berners-Lee, T.: Linked Data The Story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
- 4. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema (2004), http://www.w3.org/TR/rdf-schema/
- Harth, A., Decker, S.: Optimized Index Structures for Querying RDF from the Web. In: Third Latin American Web Congress, 2005. LA-WEB 2005. IEEE (2005)
- Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data Summaries for On-demand Queries over Linked Data. In: Proceedings of the 19th Int. Conf. on World Wide Web. pp. 411–420. WWW '10, ACM, NY, USA (2010)
- 7. Manola, F., Miller, E.: RDF Primer (2004), http://www.w3.org/TR/rdf-primer/
- McGuinness, D.L., Harmelen, F.v.: OWL Web Ontology Language: Overview (2004), http://www.w3.org/TR/owl-features/
- Neumann, T., Weikum, G.: RDF-3X: A RISC-style Engine for RDF. Proc. VLDB Endow. 1, 647–659 (August 2008)
- 10. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (2008), http://www.w3.org/TR/rdf-sparql-query/
- Quilitz, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In: The Semantic Web: Research and Applications. Lecture Notes in Computer Science, vol. 5021, pp. 524–538. Springer Berlin / Heidelberg (2008)
- Raggett, D., Hors, A.L., Jacobs, I.: HTML 4.01 Specification (1999), http://www.w3.org/TR/html401/
- Stuckenschmidt, H., Vdovjak, R., Houben, G.J., Broekstra, J.: Index Structures and Algorithms for Querying Distributed RDF Repositories. In: Proc. of the 13th Int. Conf. on World Wide Web. pp. 631–639. WWW '04, ACM, NY, USA (2004)
- Svoboda, M., Mlynkova, I.: Efficient Querying of Distributed Linked Data. In: Proceedings of the 2011 Joint EDBT/ICDT Ph.D. Workshop. pp. 45–50. PhD '11, ACM, New York, NY, USA (2011)
- Svoboda, M., Mlynkova, I.: Linked Data Indexing Methods: A Survey. In: On the Move to Meaningful Internet Systems: OTM 2011 Workshops. pp. 474–483. Springer (2011)
- Tran, T., Ladwig, G.: Structure Index for RDF Data. In: Workshop on Semantic Data Management (SemData@VLDB) 2010 (2010)
- Udrea, O., Pugliese, A., Subrahmanian, V.S.: GRIN: A Graph Based RDF Index. In: Proceedings of the 22nd National Conference on Artificial Intelligence – Volume 2. pp. 1465–1470. AAAI Press (2007)
- Weiss, C., Karras, P., Bernstein, A.: Hexastore: Sextuple Indexing for Semantic Web Data Management. Proc. VLDB Endow. 1, 1008–1019 (August 2008)