A Semantics-Oriented Storage Model for Big Heterogeneous RDF Data

HyeongSik Kim, Padmashree Ravindra, and Kemafor Anyanwu

Department of Computer Science, North Carolina State University, Raleigh, NC
{hkim22, pravind2, kogan}@ncsu.edu

Abstract. Increasing availability of RDF data covering different domains is enabling ad-hoc integration of different kinds of data to suit varying needs. This usually results in large collections of data such as the Billion Triple Challenge datasets or SNOMED CT, that are not just "big" in the sense of volume but also "big" in variety of property and class types. However, techniques used by most RDF data processing systems fail to scale adequately in these scenarios. One major reason is that the storage models adopted by most of these systems, e.g., vertical partitioning, do not align well with the semantic units in the data and queries. While Big Data distributed processing platforms such as the Hadoop-based platforms offer the promise of "unlimited scale-out processing", there are still open questions as to how best to physically partition and distribute RDF data for optimized distributed processing. In this poster, we present the idea of a *semantics-oriented RDF storage model* that partitions data into logical units that map to subqueries in graph patterns. These logical units can be seen as *equivalence classes of star subgraphs* in an RDF graph. This logical partitioning strategy enables more aggressive pruning of irrelevant query results by pruning irrelevant partitions. It also enables the possibility of semantic-query optimization for some queries such as eliminating joins under appropriate conditions. These benefits in addition to appropriate techniques for physically partitioning the logical partitions, translate to improved performance as shown by some preliminary results.

Keywords: RDF Storage Model, Partitioning Scheme, Hadoop, MapReduce

1 Introduction and Motivation

The Resource Description Framework (RDF) has been widely adopted and used to represent various datasets in many different communities such as government, life science, and finance, etc. One challenge that arises from this phenomenon is that most RDF datasets now contain a significant number of various properties and classes, e.g., 10^5 distinct properties and classes in DBPedia [3] and 400k concepts in SNOMED CT [7]. This is in contrast to popular benchmark datasets that are often used for evaluating RDF data processing systems like LUBM[1], which contain only a few hundreds of distinct properties and classes. To efficiently process such collections, data needs to be organized suitably into a storage model. A very common storage model is called *Vertical Partitioning* [1](VP) and its variants [4] which partition data in terms of the

[1] The Lehigh University Benchmark: http://swat.cse.lehigh.edu/projects/lubm

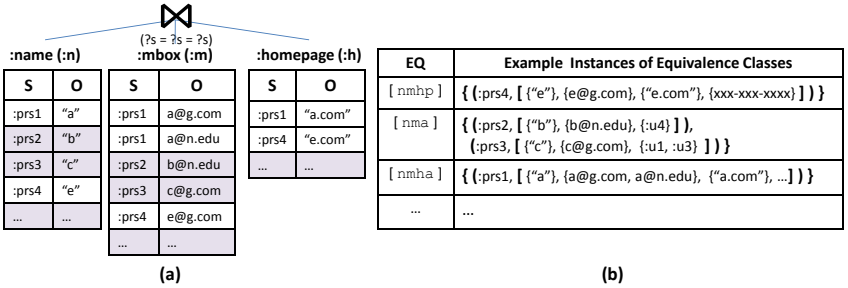

Fig. 1: A comparison of partitioning schemes: (a) vertical partitioning and its execution plan and (b) equivalence-class-based partitioning.

types of properties and classes in a dataset. Given a query, matching vertical partitions are selected based on the properties in the graph pattern and then join operations are performed using the partitions. In a sense, this approach allows all vertical partitions corresponding to properties that are not in the query to be pruned out. However, despite this degree of prunability, the joins between "relevant" vertical partitions still incurs some overhead of processing irrelevant data since not all entries the vertical partitions form joined results. For example, consider the star pattern query with properties *:name*, *:mbox*, and *:homepage*. Fig. 1(a) shows the example of the execution plan and partitioned data using the VP-based approach, which results in two join operations. The violet-colored cells denote triples that are not relevant to query, but are processed and discarded during expensive join operations. Furthermore, the vertical partitioning process itself can be challenging for large heterogeneous datasets for multiple reasons. First, it may require the management of a large number file descriptors/buffers ($> 10^5$ for DBPedia) in memory during the partitioning process which can be impractical depending on hardware architecture being used. Second, a scalability is a key design objective on Hadoop-based frameworks, but the distributed file system used in Hadoop (or HDFS) does not scale well when there are numerous small files[2]. Given these challenges, there is a clear need for investigating novel storage and distribution schemes for RDF on scale-out platforms such as Hadoop.

2 Semantics-Oriented Storage Model : SemStorm

In this poster, we build on our previous works (e.g.,[5, 6]) which introduced the notion of a *triplegroup* as a first class object in our data and query model. A *triplegroup* is a group of triples related to the same resource (i.e. with the same subject), i.e. a star subgraph. Fig. 1(b) shows the example triplegroup representation of our previous example, e.g., under the equivalence class $[nmha]$, a single triplegroup instance exists, which contains a subject (*:prs1*) and objects corresponding to properties *:n*, *:m*, *:h*, and *:a*. The benefits of both the triplegroup data model and algebra have been articulated in our previous works, including shortening of query execution workflows, reducing the footprint of intermediate results which impacts I/Os in distributed processing. Here, we present an overview of an RDF storage model called *SemStorm* that is based on logically and

[2] http://blog.cloudera.com/blog/2009/02/the-small-files-problem

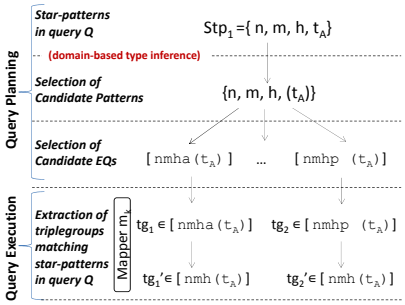

Fig. 2: Query processing in SemStorm.

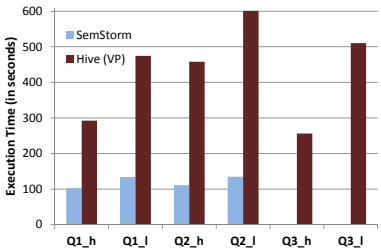

Fig. 3: An execution time of queries with type triples ($Q1$,$Q2$) and a negative query ($Q3$).

physically partitioning triplegroups in a semantics-oriented way. By semantics-oriented we mean, partitioning triplegroups into equivalence classes such that all members in an equivalence class are equivalent with respect to queries. This approach enables more aggressive pruning of logical partitions, i.e. equivalence classes than other approaches like the vertical partitioningW. For example, Fig. 2 shows that we select matching equivalence class sets (*mecs*): $[nmha]$ and $[nmhp]$, which contain all the properties in the example query, i.e. *:n*,*:m*, and *:h* (ignore a type property for now such as t_A). All other remaining equivalence classes are pruned out, e.g., $[nma]$ is not selected due to the absence of *:h*. The mecs sometimes contain extra values, e.g., objects for property *:a* and *:p* in $[nmha]$ and $[nmhp]$. We later filter such values for exact matching results.

Another unique advantage of SemStorm is that it enables additional optimizations that are not possible with other approaches, e.g., it may be possible to avoid explicitly materializing *rdf:type* triples if such triples can be inferred by the label of an equivalence class (the label of an equivalence class can be considered to be the set of properties in that equivalence class). For example, triplegroups under an equivalence class [*pubAuthor*, *rdf:type* with *Publication*] can skip materializations of *Publication* type triples if a schema file contains a triple "*pubAuthor rdfs:domain Publication*". Fig. 2 shows that type triples are not materialized in triplegroup instances such as tg_1 and tg_2, which are denoted as (t_A) for the class A. This optimization can add significant advantages because *rdf:type* triples tend to be disproportionately larger than other properties for many datasets, e.g., approx. 20% in LUBM datasets. Thus, avoiding their explicit representation reduces the amount of I/Os needed when *rdf:type* triples need to be processed and may in some cases eliminate the need to perform a join with such properties since it is implicitly captured in the equivalence class representation.

Implementation Issues. Triplegroups can be generated easily using a group-by operation on a subject field of triples using a single MR job; they are then categorized based equivalence class and stored in HDFS. Each equivalence class could be mapped into a physical file, but it is likely that such 1:1 mappings could cause the many file issue in case that many distinct equivalence classes are generated. To relieve the issue, we need a heuristic that clusters equivalence classes into a smaller number of files, e.g., group equivalence classes that share a specific set of properties and store them together. We also need to consider building indexes to locate matching equivalence classes from physical files, e.g., mappings between equivalence class and their offsets in files.

Preliminary Evaluation. We evaluated three types of queries using the LUBM datasets (450GB, Univ. 20k) on a 80-node Hadoop cluster in VCL[3], where each node was equipped with 2.33 GHz dual core CPUs, 4GB RAM, and 40GB HDD. Hive 0.12[4] was selected for the VP approach. Query $Q1$ retrieves a list of publication authors with *Publication* type triples, and $Q2$ additionally retrieves name (or title) of the publications. We evaluated two variations of queries, with object field of some non-type triple pattern bounded (high selectivity, denoted with postfix h), and same query with unbounded object (low selectivity marked with l). Fig. 3 shows that SemStorm was 3 times faster than Hive for $Q1$ because SemStorm can process queries using a Map-only job and save the disk I/O for all the type triples. The execution time of Hive increased from $Q1$ to $Q2$ due to reading additional property relation *:name* but the execution time of SemStorm was almost constant because both queries read the same equivalence classes. Finally, $Q3$ was a negative query, which produces no answers. While SemStorm determined that there are no answers (due to no matching equivalence classes) even before launching the job, Hive executed join operations, producing 0 answer. The details are available in the project website.[5]

Related Work. Our approach might be similar with Property Table [2], which groups triples that tend to be together. However, the main difference is that the property table is query-driven, which is built gradually based on query logs. However, SemStorm is data-driven one, which directly can be constructed from the datasets without any query logs. In addition, while the property table approach mainly suffers from its storage inefficiencies, e.g., a lot of NULLs and left-over tables, our approach does not, i.e. all triples can be transformed into triplegroups without any leftovers.

Acknowledgment The work presented in this paper is partially funded by NSF grant IIS-1218277.

References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: Proc. VLDB. pp. 411–422 (2007)
2. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: Proc. WWW Alt. pp. 74–83 (2004)
3. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. In: Proc. SIGMOD. pp. 145–156 (2011)
4. Husain, M., McGlothlin, J., Masud, M., Khan, L., Thuraisingham, B.: Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing 23(9), 1312–1327 (2011)
5. Kim, H., Ravindra, P., Anyanwu, K.: Scan-Sharing for Optimizing RDF Graph Pattern Matching on MapReduce. In: Proc. CLOUD. pp. 139–146 (2012)
6. Ravindra, P., Kim, H., Anyanwu, K.: An Intermediate Algebra for Optimizing RDF Graph Pattern Matching on MapReduce. In: Proc. ESWC. pp. 46–61 (2011)
7. Salvadores, M., Horridge, M., Alexander, P.R., Fergerson, R.W., Musen, M.A., Noy, N.F.: Using SPARQL to Query Bioportal Ontologies and Metadata. In: Proc. ISWC. pp. 180–195 (2012)

[3] Virtual Computing Lab: http://vcl.ncsu.edu

[4] https://hive.apache.org

[5] http://research.csc.ncsu.edu/coul/RAPID+/ISWC2014.html