Partout: A Distributed Engine for Efficient RDF Processing

Luis Galárraga
Télécom ParisTech
Paris, France
luis.galarraga@telecom-paristech.fr

Katja Hose
Aalborg University
Aalborg, Denmark
khose@cs.aau.dk

Ralf Schenkel
University of Passau
Passau, Germany
ralf.schenkel@uni-passau.de

ABSTRACT

The increasing interest in Semantic Web technologies has led not only to a rapid growth of semantic data on the Web but also to an increasing number of backend applications relying on efficient query processing. Confronted with such a trend, existing centralized state-of-the-art systems for storing RDF and processing SPARQL queries are no longer sufficient. In this paper, we introduce PARTOUT, a distributed engine for fast RDF processing in a cluster of machines. We propose an effective approach for fragmenting RDF data sets based on a query log and allocating the fragments to hosts in a cluster of machines. Furthermore, PARTOUT's query optimizer produces efficient query execution plans for ad-hoc SPARQL queries.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.2.4 [**Database Management**]: Systems – Distributed Databases

Keywords

Semantic Web, Distributed Systems, RDF, Data Partitioning, Distributed Query Processing

1. INTRODUCTION

The increasing interest in Semantic Web technologies led to a rapid growth of available semantic data on the Web. Initiatives such as Wikidata[1] have emerged as native sources of semantic data. In other respects, advances in information extraction still enable efficient and accurate extraction of knowledge from natural language text and its representation in a machine-readable format – RDF (Resource Description Framework). DBpedia[2], for instance, has now reached a size of 4 million entities and 2.6 billion RDF triples extracted from Wikipedia. As the number of Wikipedia articles increases every day and as information extraction techniques are still being improved, DBpedia, Wikidata, and similar knowledge bases are continuously growing.

Approaches for query processing can be categorized into two major groups: solutions involving query-time data retrieval and data warehousing. The first category often comprises indexing [2] of data dumps or federations of SPARQL endpoints [6]. The main disadvantage of these systems is the lack of control over the data, i.e., there is no guarantee on response time or that the data is available during query evaluation.

In data warehousing, the data is downloaded from the Web, collected in a huge triple store, and updated from time to time. Query processing in such a setup strongly benefits from efficient centralized query optimization and execution. Still, the ever-growing amount of RDF data will sooner or later result in scalability problems for a single machine. Hence, approaches designed to run on clusters of machines have been proposed [3]. Similar to distributed databases and data warehouses in general, where data is often collected and stored to serve a particular use case, it is possible for many applications to derive a representative query workload. By exploiting this information, we can achieve an additional gain in performance as triples accessed together can be allocated on the same machine, optimizing for efficient local execution. PARTOUT[3] implements a query-load aware partitioning and allocation of RDF triples for efficient query processing on a cluster of machines.

2. PARTOUT

At a high level, PARTOUT consists of a dedicated central coordinator and a cluster of n hosts that store the actual data. The central coordinator is responsible for distributing the RDF data among the hosts, building efficient distributed query plans for SPARQL queries, and coordinating query execution. Each host runs a triple store, which in our implementation conforms to an adapted version of RDF-3X [4] – more details on our implementation and PARTOUT in general are available in [1].

2.1 Data Partitioning and Allocation

Partitioning and allocation is done in three main steps:

Building a global query graph. Given a query load $QL = \{q_1, \ldots, q_L\}$ as bag of SPARQL queries, either collected from a running system or estimated from queries in

[1] `http://wikidata.org`
[2] `http://dbpedia.org/`

[3] pronounced like the French word *partout*; the name is a combination of the terms *part*ition and scale-*out*

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2577302.

applications accessing the RDF data, for each query $q \in QL$, we normalize and anonymize its triple patterns by replacing infrequent URIs and literals with variables in triple patterns. Then, for each normalized triple pattern p in query q, we obtain its anonymized version $p' = \omega(p)$ by replacing its variables by the same anonymized symbol Ω. The anonymized triple patterns are used to build a global query graph $G(QL)$, where each node is an anonymized triple pattern and there exists an edge between two nodes if the sources of such anonymized patterns join in the query load.

Fragmentation. The second step takes a set of triples T (content of the triple store) and a query load QL and produces a set of disjoint partitions of T. Similar to horizontal fragmentation of relations, we extract a set of simple predicates from QL. A *simple predicate* is a constraint on a triple component (subject, property, or object). Examples are $prop = type$ or $isIRI(obj)$. For each node p' in $G(QL)$, each position that does not contain Ω creates a simple predicate. SPARQL filter conditions are also used to extract simple predicates. Based on the set of simple predicates $S(QL)$, we generate the set M' of minterms, i.e., all conjunctive combinations of simple predicates in their positive or negated form. Each minterm $m \in M$ defines a non-overlapping partition of the data set T. Given a triple store T and a set of simple predicates $S(QL)$ as input, the COM_MIN algorithm [5] iteratively computes a minimal subset of independent simple predicates for partitioning.

Fragment Allocation. Once fragments have been defined, we allocate them to the n hosts in our cluster. The allocation procedure has two contradicting goals, (1) assign fragments that are used together in a (part of a) query to the same host to guarantee local execution, and (2) balance the load over all nodes in the cluster. This is achieved by means of an iterative greedy routine that sorts fragments in descending order by load and in each step assigns a fragment to the most beneficial host. The load of a fragment is a function of its size and the number of queries in QL that use that fragment. The benefit of allocating a new fragment to a host, is inversely proportional to the host's current load and directly proportional to the number of already allocated fragments that join with the new fragment. Both the load and the benefit of a fragment with respect to a host use the information encoded in the global query graph $G(QL)$.

2.2 Query Processing

Distributed query processing in PARTOUT starts when the user issues a query against the coordinator and consists of three phases. First, the query is parsed and converted into an RDF-3X execution plan using the statistics in the coordinator manifest file (built at allocation time). This plan resembles an operator tree where the leaves access the data sources. Since this plan is not optimized for a distributed setup, the second phase transforms the centralized query plan into a distributed query plan by (a) resolving the locations of the data sources i.e., which hosts are relevant to a triple pattern, and (b) computing the optimal place of evaluation for the non-leaf operators. This phase utilizes a modified version of the RDF-3X cost model that takes into account remote data transfers. Once the coordinator has found a cheap query execution plan, it sends the plan to the relevant hosts, which execute the assigned operators and retrieve the results to either the coordinator or other host.

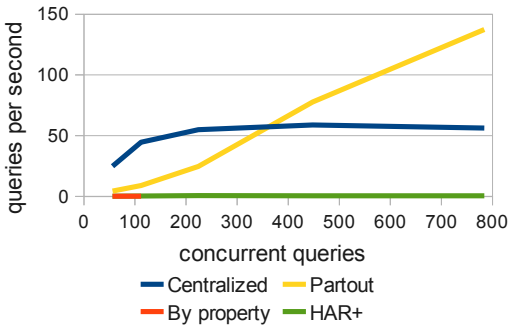

Figure 1: Throughput for the BTC 2008 dataset.

3. RESULTS AND CONCLUSIONS

The evaluation depicted in Figure 1 compares PARTOUT in terms of throughput, against other approaches for RDF query processing, including (a) a centralized RDF-3X setup, (b) PARTOUT using naive partitioning by predicate and (c) an improvement of the approach presented in [3], which implements RDF data partitioning and allocation based on graph partitioning techniques. Unlike the original approach, this solution avoids Map-Reduce operations and is therefore faster. Moreover, both approaches allow data replication. The experiments use the Billion Triple Challenge 2008 dataset[4] (+500M triples). 30 random queries keeping a balance between star and path queries were extracted and used as query load for partitioning. They were run in parallel to simulate a concurrent workload. Using one coordinator and 3 hosts, PARTOUT outperforms other approaches mainly because of its load-balancing policy which allows different computers to process queries locally in parallel against a much smaller subset of the data. Nevertheless, the absence of data replication often results in some queries being penalized when some of their relevant fragments are distributed among several hosts. As future work, we envision to combine data partitioning and allocation with smart replication of fragments to improve the performance of the system both in terms of throughput and response time.

4. REFERENCES

[1] L. Galárraga, K. Hose, and R. Schenkel. Partout: A Distributed Engine for Efficient RDF Processing. *CoRR*, abs/1212.5636, 2012.

[2] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *WWW*, pages 411–420, 2010.

[3] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL Querying of Large RDF Graphs. *PVLDB*, 4(11):1123–1134, 2011.

[4] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.

[5] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

[6] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *ISWC*, pages 601–616, 2011.

[4] `http://challenge.semanticweb.org/`