Distributed SPARQL Throughput Increase: On the effectiveness of Workload-driven RDF partitioning

Cosmin Basca and Abraham Bernstein

DDIS, Department of Informatics, University of Zurich, Zurich, Switzerland
{lastname}@ifi.uzh.ch

1 Introduction

The current size and expansion of the Web of Data or WoD, as shown by the staggering growth of the Linked Open Data (LOD) project[1], which reached to over 31 billion triples towards the end of 2011, leaves federated and distributed Semantic DBMS' or SDBMS' facing the open challenge of scalable SPARQL query processing. Traditionally, SDBMS' push the burden of efficiency at runtime on the query optimizer. This is in many cases too late (i.e., queries with many and/or non-trivial joins). Extensive research in the general field of *Databases* has identified *partitioning*, in particular horizontal partitioning, as a primary means to achieve scalability. Similarly to [2] we adopt the assumption that *minimizing the number of distributed-joins as a result of reorganizing the data over participating nodes will lead to increased throughput in distributed SDBMS'*. Consequently, the benefit of reducing the number of distributed joins in this context is twofold:

A) *Query optimization becomes simpler*. Generally regarded as a hard problem in a distributed setup, query optimization benefits, at all execution levels, from fewer distributed joins. During *source selection* the optimizer can use specialized indexes like in [5], while during *query planning* better query plans can be devised quicker, since much of the optimization burden and complexity is shifted away from the distributed optimizer to local optimizers.

B) *Query execution becomes faster*. Not having to pay for the overhead of shipping partial results around, naturally reduces the time spent waiting for usually higher latency network transfers. Furthermore, federated SDBMS' incur higher costs as they have to additionally serialize and deserialize data.

The main contributions of this poster are: ***i)*** the presentation of a novel and naïve workload-based RDF partitioning method[2] and ***ii)*** an evaluation and study using a large real-world query log and dataset. Specifically, we investigate the impact of various method-specific parameters and query log sizes, comparing the performance of our method with traditional partitioning approaches.

2 Method Overview

Traditional approaches like Schism construct a graph representation where vertexes are tuples that participate in workload transactions. The graph is extended

[1] http://linkeddata.org/

[2] This work was partially supported by the Swiss National Science Foundation under contract number 200021-118000.

to include all other tuples using a partition-trained classifier. Following this idea, triples would be considered vertexes, while edges are created when any two triples participate in the same query. This is however not feasible for RDF data.

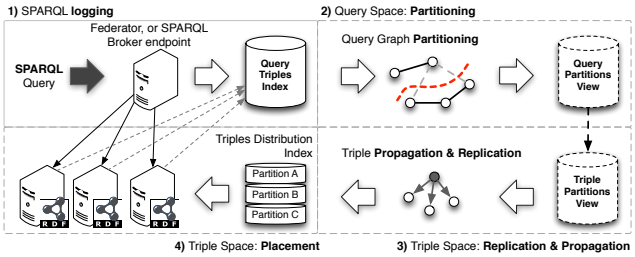

Fig. 1. A simple generalized view of the partitioning process.

Following this graph representation in our early attempts led to very dense graphs, which proved to be too large for state of the art graph partitioning software like Metis [4].[3] Next, we detail all steps seen in Figure 1 except the simpler 1^{st} phase where queries and their results are logged.

Data Representation & Graph Partitioning. Since mapping triples to vertexes does not scale well for RDF data, we pursued an intermediate representation: the **Queries graph**.

Each query in the workload becomes a vertex, while edges between queries are formed when some triples participate in more than one query, with the number of common triples as edge weights (Figure 2). Finally, we apply Metis on the newly formed queries graph, forcing balanced partitions as a result of the *graph-cut* operation.

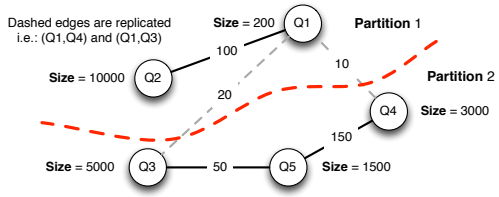

Fig. 2. Basic query log-driven data representation.

Replication. After performing the *graph-cut*, there will still be distributed joins even on the workload queries (i.e., query $Q1$ will require a distributed join while query $Q2$ not). A straight-forward solution is to replicate the triples that reside on the border between partitions. We proceed with identifying the minimum set of triples that needs to be replicated, copying the extra triples from the smaller sized query (i.e., copy extra triples from query $Q1$ over to Partition 2).

Propagation. While the process outlined so far guarantees that each query in the workload can be executed without a single distributed join there are no guarantees for future unknown queries. A method of expanding the set of all triples which participate in all workload queries is needed. For this we rely on the principle of *(Spatial) Locality of Reference* [3] adapted to the logical graph representation.

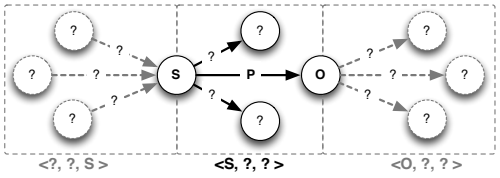

Fig. 3. Visual depiction of the propagation patterns.

In effect we *propagate* along the edges in the original RDF data graph to identify new triples "related" to existing triples which participated in all workload

[3] The resulting input edge file amounted to approx. 150GB on disk, crashing Metis.

queries. Hence, we perform an n-hop[4] edge propagation matching the following triple patterns given a triple <s,p,o> (also depicted Figure 3): *a*) siblings: $<$ *s*,?,?>, *b*) outgoing edges: <o,?,?> and *c*) incoming edges: <?,?,s>. The remaining dataset triples which have not been considered so far, are randomly distributed to the K selected partitions, or by hashing by subject.

3 Results & Conclusions

We make use of the USEWOD Data challenge [1] log file to extract 400k valid and well formed SPARQL SELECT queries that produce at least 1 result, all other log entries are discarded. We use a local instance of the `Virtuoso` RDF-store to resolve them against DBpedia 3.5.1. Furthermore, we assume a *perfect* distributed query optimizer, able to find the best possible query decomposition. Measurements were conducted on a node with 72GB of RAM, 8 Cores @2.93GHz.

We compare our method against *random partitioning*, *expert (manual) partitioning*[5] and *hash partitioning*. For the latter we hash on all possible combinations of a triple: `S`, `P`, `O`, `SP`, `SO`, `PO` and `SPO`.[6] Given the small to average size of the DBpedia dataset (ca. 43.6 million triples), we fixed the number of partitions to $K = 8$, simulating a small to medium sized cluster. Furthermore, we randomly sampled the workload, with sizes consisting of 1k, 5k, 10k, 25k, 50k and 100k queries from the total of 400k logged. The number of propagation **hops** was set to 0, 1 and 2 respectively while **replication** was enabled in all cases.

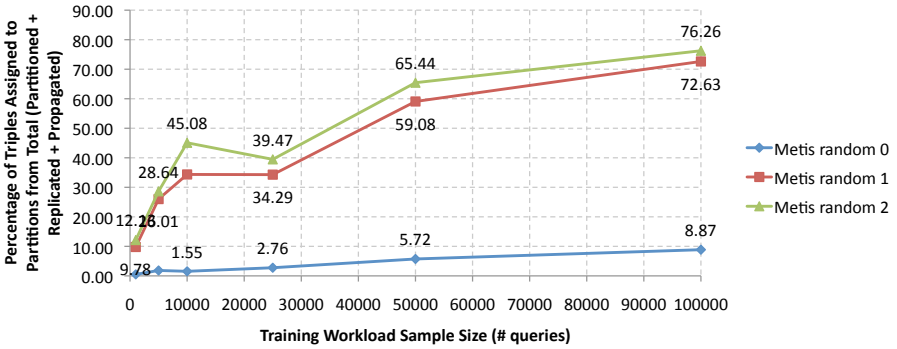

Fig. 4. The % of triples assigned to partitions from total triples, for each training query set.

As we can visually observe in Figure 4 that although the increase in number of triples reached through the partitioning process (excluding the triples not connected at all) is significant from 50k to 100k queries, there are diminishing returns as the expansion process starts to slow down. Indeed doubling the number of queries to log, yields approximatively a 10% increase at this point. Therefore, we observe that a training log size of 50k queries represents an optimal point.

Performance Impact of Graph Partitioning. Even-though the general problem of graph partitioning is known to be NP-hard, the approximating algorithm implemented in Metis performs very well, finalizing the queries graph cut in 0.17 seconds for 1k queries and 0.71 seconds for 100k queries.

[4] Multiple hops only enabled for in & out edges to avoid an expensive avalanche effect.
[5] Each large dump (if $> 1M$ triples) to own partition, remainder grouped together.
[6] We use of the `cityhash` family of hash functions, due to low-collision rate and speed.

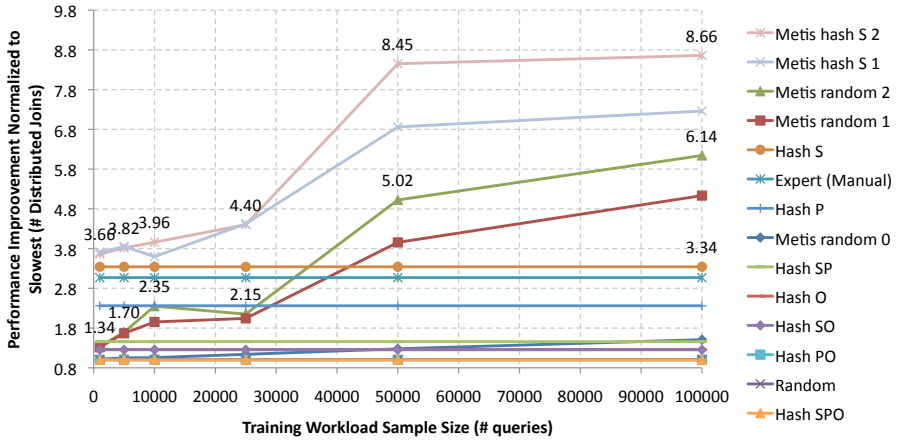

Fig. 5. The performance improvement relative to the lowest performing method (hash SPO).

Number of Hops Impact when Propagating. Figure 5 plots the relative performance improvement over the lowest performing method. Non-workload driven partitioning methods appear as horizontal lines. The worst performing ones are the *Hash SPO* method with *Random* & *Hash PO/SO* exposing similar performance levels. Hashing by subject *Hash S* is performing best, followed closely by the *Expert (manual)* distribution method. This could suggest that the majority of the workload queries are dominated by star-shaped basic graph patterns and contain few joins.[7] When using random distribution of remaining triples our method performs unsatisfactory for smaller workload sizes, but becomes substantially better by 50k queries. At 100k queries it exposes a 6.14 performance factor being 2.81 and 3.1 times better than *Hash S* and *Expert* respectively. When remaining triples are distributed based on subject hashes, the method outperforms all other methods at all workload sizes. At best (*Metis hash S 2*) our method is between 3.6 and 8.66 times better than the lowest performing and up to 5.32 times better than hashing by subject. In essence the best case partitioning would produce on average of 0.10 distributed joins per query.

References

1. B. Berendt, L. Hollink, V. Hollink, M. Luczak-Rsch, K. H. Mller, and D. Vallet. Usewod2011 - 1st international workshop on usage analysis and the web of data. in 20th international world wide web conference (www2011), hyderabad, india, 2011.
2. C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: a workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 3:48–57, Sept. 2010.
3. P. J. Denning. The locality principle. *Communications of the ACM*, 48, July 2005.
4. G. Karypis and V. Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/~metis, 2009.
5. Y. Yan, C. Wang, A. Zhou, W. Qian, L. Ma, and Y. Pan. IEEE Xplore - Efficient Indices Using Graph Partitioning in RDF Triple Stores. In *ICDE2009: IEEE 25th International Conference on Data Engineering, 2009.*, pages 1263 – 1266, 2009.

[7] A fact we intend to investigate in depth in the near future