Optimizing SPARQL Query Processing On Dynamic and Static Data Based on Query Response Requirements Using Materialization

Soheila Dehghanzadeh

Insight Center for Data Analytics, Ireland, Galway
soheila.dehghanzadeh@insight-centre.org

Abstract. To integrate various Linked Datasets, the data warehousing and the live query processing approaches provide two extremes for the optimized response time and quality respectively. The first approach provides very fast responses but suffers from providing low-quality responses because changes of original data are not immediately reflected on materialized data. The second approach provides accurate responses but it is notorious for long response times. A hybrid SPARQL query processor provides a middle ground between two specified extremes by splitting triple patterns of the SPARQL query between live and local processors based on a predetermined coherence threshold specified by the administrator. However, considering quality requirements while splitting the SPARQL query, enables the processor to eliminate the unnecessary live execution and releases resources for other queries and is the main focus of my work. This requires estimating quality of the response provided with the current materialized data, compare it with user requirements and determine the most selective sub-queries which can boost the response quality up to the specified level with least computational complexity. In this work, we discuss the preliminary result for estimating the freshness of materialized data, as one dimension of the quality, by extending cardinality estimation techniques and explain the future plan.
Keywords. RDF Data Warehouse, View Materialization, SPARQL live querying, quality estimation.

1 Problem Description

Content of the Linked Data is constantly growing and distributed among heterogeneous sources that change their data with various update rates. To process queries over the Linked Data, a data warehousing approach [13] creates a central repository of all triples that is collected by crawlers. However, crawling, storing and maintaining this huge amounts of data is a challenging task due to data volume, velocity and variety. The incremental view maintenance or more recently the higher order incremental view maintenance [10] are designed to efficiently maintain views based on an *update stream* in a relational database environment. But in a Linked Data environment individual RDF datasets or SPARQL endpoints are not designed to report every single update. Thus the data warehouse

has to extract updates by querying original sources. Hence, the view maintenance translates to the live query execution which is a very time consuming job and it is more efficient to defer it as far as current materialized data could fulfil response quality requirements.

Another approach to manage the Linked Data is live querying [6] which processes queries by dereferencing URIs and following relevant links on-demand and naturally incurs very slow response times but with high quality (fresh and complete). Thus, the more time spent on fetching data from source the higher the response quality and vice versa. This represents an inherent trade-off between the time spent on processing the query and the quality of the response which can be considered as a spectrum from a response with high quality and long retrieval time to a response with low quality but short retrieval time.

To mitigate time consuming live querying, the recently proposed hybrid query processing technique [15] suggested to combine the data warehousing with the live query processing techniques. [15] uses coherence values to split triple patterns of a query to a dynamic predicate set for live execution, and a static predicate set for local processing. The coherence of each predicate is defined as the ratio between the cardinality of live results that exist in the materialized data and the total cardinality of live results. [15] achieved individual points in the response time/quality trade-off spectrum by splitting predicates using different coherence thresholds. The coherence threshold of the hybrid approach is strictly defined by the system administrator and has no flexibility depending on response quality requirements of individual queries. However, some query requirements can be fulfilled using the existing local store with no or less live execution. Thus we hypothesize that query response requirements can be exploited to adaptively optimize the splitting process. This releases computational resources for other queries and leads to better scalability and efficient load balancing.

Motivation To motivate the described problem, consider a user who is willing to broadcast a commercial advertisement to specific emails and is satisfied with say 80% freshness in email addresses provided by the response. The incentive of being satisfied with less freshness or quality is to get faster response and consume less computational resources and pay less accordingly. Response requirements are expressed based on the response time and quality by the user or service issuing the query.

Research Question The fundamental research question is how to optimally split an SPARQL query among live and local query processors based on response time and quality requirements? Currently, there exists no automatic way to guide the splitting decision and to adaptively refine it. The fastest response is achieved by fully executing the query on materialized data. In order to compare the quality of response provided with materialized data with required quality, we need to estimate it. Thus, estimating the quality of response provided with materialized data is our first sub-problem. If materialized data couldn't fulfil user quality requirements, parts of the query must be executed lively to boost the quality of the response. Various sub-queries can be redirected to the live engine which leads to various *splitting strategies*. The second sub-problem is to estimate the

quality of the response achieved with an splitting strategy. The optimization is to choose the least costly splitting strategy which is estimated to fulfil required quality.

Contribution Of The Thesis The main contribution in this thesis is to optimize query splitting according to specific requirements of each query. Choosing the splitting strategy that estimated to fulfil required quality with lowest execution time is the ultimate goal of our query processor. To do so, we break down the problem into two sub-problems mentioned in the research question. Here we propose solutions for the first sub-problem. Also quality metrics should be explicitly defined in a Linked Data query processing environment.

In Fig 1 suppose the shaded circle represents the result of executing query on the materialized data and the transparent circle represents the query results of the live engine. "A", which represents query results in the materialized data that doesn't exist in the result provided with the live engine, could contain inferred results from materialized data, data from not available sources and results which has been removed from original sources. However we relax the problem by only considering the latter reason which means we assumed all sources to be available and no inferred data is added to the materialized data. "B" represent the result set which exists both in live and local store. "C" represents the newly added query responses to the live data which still have not been reflected in the materialized data. We simplify the problem by assuming that live engine is able to cover all potential responses. With the above assumptions, *Freshness* quantifies the effect of the deletion on response quality and is defined as $B/(A+B)$. *Completeness* quantifies the effect of addition on response quality and is defined as $B/(B+C)$. In our preliminary experiment we only consider the freshness as the quality metric of the response because in our synthetic data set we are assuming that data can only be removed from real world after materialization which makes triples to become stale. However, in this thesis, we are aiming to consider real world snapshots with both addition and deletion. Thus, both completeness and freshness need to be considered and estimated as quality metrics of various splitting strategies.

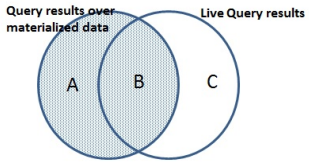


Fig. 1. freshness and completeness

Problem Relevancy By adaptively splitting the query between local and live processors, we prevent unnecessary live execution and reduce network traffic and response time.

2 Related Work

The problem of efficiently processing queries by exploiting the materialized data requires a comprehensive *view management* procedure. This includes the view selection, the view maintenance, the view exploitation and the cost modelling. Interested readers are referred to [5] for detailed explanation on each phase.

Various strategies for view selection(i.e. full or partial materialization) and view maintenance(i.e. immediate or deferred maintenance) can adjust the response time/quality trade-off. DBToaster [10] fully materialize data and immediately apply updates. Thus, there is no option to adjust the time/quality trade-off. It minimizes the cost of processing updates by converting the maintenance task to an efficient code for execution in a relational data model.

[4] chooses the best query set which fully covers a fixed query set and they didn't discuss the effect of postponing maintenance on time/quality trade-off.

The hybrid approach introduced by [15], considers the existing store of a data warehouse as a predefined set of materialized data to be exploited for responding queries. Thus, the hybrid approach actually relaxes the view selection. They use a coherence value to split the query for local or live execution(i.e, maintenance). As alluded before, to maintain the views according to response requirements, we need to adaptively refine the coherance threshold which is not addressed by [15]. On the other hand, [2] recommended RDF indices to materialize based on a given workload aiming to improve performance of the query evaluation. However, in contrast to [4], queries still need to access the original data set because indices are partially covering queries. This approach assumes the original data are not changing and materialized data never gets out-of-date.

[9] is using response requirements to defer unnecessary maintenance tasks based on user preferences when an update stream exists at the data warehouse. We are aiming to target a similar problem but in a Linked Data querying environment where an update stream doesn't exist.

There has been research to estimate the quality of query response provided by materialized data in relational database which requires accurate cardinality estimation and accuracy of involved attributes [3]. The estimation of quality metrics are based on the identity attribute and is achieved by tracking the category change of each type of tuple during each operation. However, in an RDF setting there is no notion of id for tuples. Thus applying that approach for the Linked Data is not directly possible. We hypothesize that statistics of cardinality estimation techniques can be extended to estimate the quality of a query response.

3 Approach

Following our hypothesis, we extend indexing and multi-dimensional histograms for estimating the freshness of a join.

Indexing Based Approaches We designed two indexing structures to estimate join freshness:

- **Simple** Estimate join freshness by simply multiplying freshness of join's predicates. It requires indexing predicates along with their observed freshness. This approach works very well when join result is the Cartesian product of each predicate's result set.
- **CS** Estimate join freshness by using the characteristic set [11] technique. It groups subjects with the same set of predicates together and index it as a

"subject group". The whole dataset can be summarized into a set of "subject group"s with their associated predicates and freshness for each predicate. Analogously, the join's characteristic set(s) consist of individual subject(s) with their requested property(es). To estimate the freshness of a join, we simply sum up the fresh cardinality and the total cardinality of characteristic sets that are super set of the join's characteristic set and divide the fresh cardinality by the total cardinality.

Histogram Based Approaches Histograms and Qtrees are among successful approaches for the data summarization and the join cardinality estimation. Summerization in histogram-based approaches (histogram and Qtree) is achieved by grouping attribute values into buckets and estimating all bucket entries with one summarized value. Join between triple patterns translates to intersection of buckets, assuming that entries are uniformly distributed all over each bucket. Interested readers are referred to [14] for more explanation. To adapt the histogram-based approaches for the freshness estimation problem, we proposed keeping two entries per bucket; the number of fresh and stale entries.

Histogram based approaches require a hashing function to transfer the string representation to numeric representation for processing data. It will determine the uniformity of data distribution which is the main trick leveraged by the histogram to summarize data. Histogram bucket boundaries for each dimension are determined based on a *partitioning rule* which requires a sort and source parameter to specify buckets [12]. We investigated different sort and sources and results are presented in Section 4.

Qtree is an optimized histogram and its buckets are determined by identifying populated areas in the multi dimensional cube using a distance metric. Interested readers are referred to [14] for more detailed explanation.

Evaluation Plan In our preliminary work, we assumed that data can only be deleted from the original dataset after materialization. Thus we could only measure freshness metric. We summarized a synthetically labeled dataset to estimate the freshness of queries without executing query on the original labeled dataset. However, in fact both addition and deletion occur in original data after materialization and therefore we need to be able to estimate both freshness and completeness of the response. For that we need to either extend the cardinality estimation with statistics of both addition and deletion or create individual indexes for estimation of each quality metric. A real example of addition and deletion occurring in materialized data can be observed within consecutive snapshots of the Linked Data Observatory [8]. To solve the first sub-problem in a realistic scenario, i.e.,estimating the quality of response provided with materialized data, we summarize these snapshots with extension of above approaches, compare their estimation performance for individual quality metrics and choose the one with lowest estimation error. We use the same summarization technique for estimating the quality of response provided with an splitting strategy considering that the quality of the live sub-query has increased to 100%. Having the quality estimations of each splitting strategy, we choose the splitting strategy that is estimated to fulfil response requirements and has the lowest

execution time. To evaluate the effectiveness of considering user requirements for optimizing query processing, we will compute the difference among required quality and achieved quality of the response provided with various query processing approaches(i.e., materialized, live, hybrid, adaptive-hybrid) considering their execution time. Adaptive-hybrid approach is aiming to achieve lowest execution time and lowest quality difference.

4 Preliminary Results

Experiment set-up In this experiment, we are tackling the join freshness estimation problem. We used the BSBM benchmark [1] to generate a dataset (374,920 triples with 40 distinct predicates) and a query set. Each triple is either fresh i.e, triple exists after maintenance or stale i.e, triple doesn't exist after maintenance. To split triples between fresh and stale category, we divide predicates among 10 levels of freshness(0-10%, 10-20%, ..., 90-100%) according to r-beta distribution of predicate freshness observed in [15] and assign true or false to triples in dataset based on the freshness value of their predicate. We used the BSBM query templates to generate queries and extract individual joins out of them. In this paper, due to space limitation, we only present actual and estimated freshness for 557 subject-subject joins. To estimate freshness of joins, an index(histogram) is built ,by inserting all individual triples with their associated label to their corresponding index entry (histogram bucket), and used for join processing as explained below:

– **Indexing** We estimated the freshness of subject-subject joins using two indexing approaches: simple freshness multiplication and characteristic set [11]. Figure 2 shows actual and predicted freshness started to disagree after query 285 which is due to existence of bounded triple patterns in joins. Thus index-based approaches lack on joins having a bounded triple pattern.

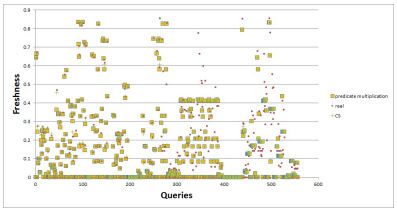
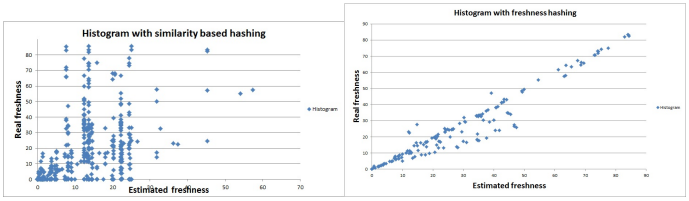


Fig. 2. freshness estimation in subject-subject joins using indexing approaches

– **Histogram** Histogram requires a proper hashing technique to transfer data from the string representation to the numerical representation.
Choose a Proper Hashing Figure 3(a) shows actual versus predicated freshness using the histogram with similarity-based hashing (mixed hashing

proposed in [14]). In figure 3(a), joins with different actual freshness have



(a) (b)

Fig. 3. predicted vs real freshness in histogram using (a)similarity based hashing (b) freshness hashing for S-S joins

been predicted with similar values due to similarity among bounded objects. This observation along with the fact that histogram estimates neighbouring entries with the same value, strikes the idea of keeping entries with similar freshness close together. Hence, we proposed to sort dimension entries based on their freshness values. Figure 3(b) depicts that sorting dimension entries based on their observed freshness (freshness hashing) leads to better freshness estimates for joins.

Estimation Error We quantified the estimation error of proposed techniques using RMSD normalized error [7] to compare the estimation error over the course of storage space. The normalized RSMD of histograms using the freshness hashing is plotted in Figure 4 and it shows the estimation error of the histogram and the Qtree converged to 0.07 in summary size of 3000 buckets while the simple predicate multiplication (an indexing based approach) consumes less space with a lower estimation error. The error of histogram based approaches can be further decreased by increasing the summary size or implementing more advanced type of histograms.

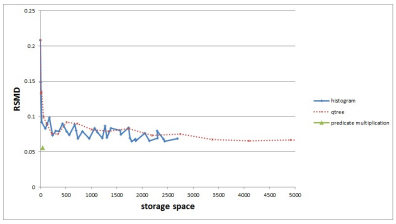


Fig. 4. Freshness estimation error in s-s join in Qtree and histogram using sort hashing

Reflections Traditional approaches estimate join freshness by multiplying freshness of join counterparts. We showed that, this approach mainly lacks on joins with bounded triple patterns(Figure 2). We compared its estimation performance with adapted histograms which leads to less estimation error only by increasing the histogram's summary size. We are planning to reduce the estimation error by using histograms with advanced hashing and adapting other cardinality estimation techniques such as sampling and wavelet.

Acknowledgements I would like to thank Manfred Hauswirth, Josiane Xavier Parreira and Marcel Karnstedt for their valuable comments on the paper.

References

1. Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5:1–24, 2009.
2. Roger Castillo, Christian Rothe, and Ulf Leser. *RDFMatView: Idexing RDF Data for SPARQL Queries*. Professoren des Inst. für Informatik, 2010.
3. Debabrata Dey and Subodha Kumar. Data quality of query results with generalized selection conditions. *Operations Research*, 61(1):17–31, 2013.
4. François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *Proceedings of the VLDB Endowment*, 5(2):97–108, 2011.
5. Jonathan Goldstein and Per-Åke Larson. Optimizing queries using materialized views: a practical, scalable solution. In *ACM SIGMOD Record*, volume 30, pages 331–342. ACM, 2001.
6. Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing sparql queries over the web of linked data. In *The Semantic Web-ISWC 2009*, pages 293–309. Springer, 2009.
7. Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.
8. Tobias Käfer, Jürgen Umbrich, Aidan Hogan, and Axel Polleres. Towards a dynamic linked data observatory. *LDOW at WWW*, 2012.
9. Alexandros Labrinidis and Nick Roussopoulos. Exploring the tradeoff between performance and data freshness in database-driven web servers. *The VLDB Journal*, 13(3):240–255, 2004.
10. Daniel Lupei, Amir Shaikhha, Christoph Koch, Andres Nötzli, Oliver Andrzej Kennedy, Milos Nikolic, and Yanif Ahmad. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. Technical report, 2013.
11. Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 984–994. IEEE, 2011.
12. Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. Improved histograms for selectivity estimation of range predicates. *ACM SIGMOD Record*, 25(2):294–305, 1996.
13. Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice. com: Weaving the open linked data. In *The Semantic Web*, pages 552–565. Springer, 2007.
14. Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. Comparing data summaries for processing live queries over linked data. *World Wide Web*, 14(5-6):495–544, 2011.

15. Jürgen Umbrich, Marcel Karnstedt, Aidan Hogan, and Josiane Xavier Parreira. Hybrid sparql queries: fresh vs. fast results. In *The Semantic Web–ISWC 2012*, pages 608–624. Springer, 2012.