Ollscoil na Gaillimhe
University of Galway

Provided by the author(s) and University of Galway in accordance with publisher policies. Please cite the published version when available.

Title	Freshening up while Staying Fast: Towards Hybrid SPARQL Queries
Author(s)	Umbrich, Jürgen; Karnstedt, Marcel; Hogan, Aidan; Parreira, Josiane Xavier
Publication Date	2012
Link to publisher's version	http://www.deri.ie/sites/default/files/publications/ekaw2012.pdf
Item record	http://hdl.handle.net/10379/4544

Downloaded 2024-04-26T05:05:32Z

Some rights reserved. For more information, please see the item record link above.

Freshening Up While Staying Fast: Towards Hybrid SPARQL Queries

Jürgen Umbrich, Marcel Karnstedt, Aidan Hogan, and Josiane Xavier Parreira

Digital Enterprise Research Institute, National University of Ireland, Galway
firstname.lastname@deri.org

Abstract. Querying over cached indexes of Linked Data often suffers from stale or missing results due to infrequent updates and partial coverage of sources. Conversely, live decentralised approaches offer fresh results directly from the Web, but exhibit slow response times due to accessing numerous remote sources at runtime. We thus propose a *hybrid query* approach that improves upon both paradigms, offering fresher results from a broader range of sources than Linked Data caches while offering faster results than live querying. Our hybrid query engine takes a cached and live query engine as black boxes, where a hybrid query planner splits an input query and delegates the appropriate sub-queries to each interface. In this paper, we discuss query planning alternatives and their main strengths and weaknesses. We also present coherence measures to quantify the coverage and freshness for cached indexes of Linked Data, and show how these measures can be used for hybrid query planning to optimise the trade-off between fresh results and fast runtimes.

1 Introduction

Current query approaches over Linked Data offer either (i) fast query times by materialising local optimised indexes [2,10,3] that cache Web data, but at the cost of potentially incomplete or outdated results; or (ii) fresh results by accessing query relevant data from the Web at runtime, but at the cost of slower query times [4,16,8]. This trade-off between fresh and fast results becomes even more crucial as Linked Data expands and becomes more dynamic.

In previous work [18], we identified and started to address this tradeoff problem, proposing the hybrid query execution architecture as shown in Figure 1. Our envisioned query engine has two SPARQL interfaces to combine (i) the performance of a centralised SPARQL store (henceforth: *cache*) with (ii) the up-to-date results of a live SPARQL engine. We do not build yet another local or live SPARQL engine, nor do we design techniques for either: instead, we propose to take off-the-shelf engines (one live and one cache) as black boxes and investigate the techniques by which they can be combined in a complementary way. Our proposed engine can be seen as adding a live "query wrapper" for existing SPARQL stores, or as adding a SPARQL-enabled cache for live approaches. The core components of our architecture are (i) a *coherence monitor* that computes and stores statistics about the dynamicity and coverage of cache

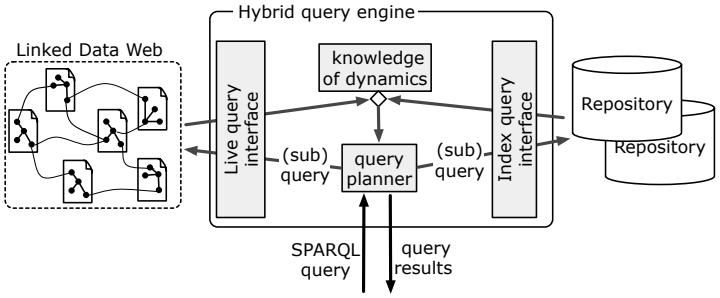

Fig. 1. Architecture of a hybrid query-engine.

data; and (ii) a *query planner* that uses these statistics to (amongst other tasks) decide which parts of the query should be run against the cache and which live. By getting the cache to quickly service query patterns for which it has good up-to-date coverage, and by running the remaining patterns of the query live, our hybrid query planner aims to optimise the aforementioned trade-off of fresh vs. fast results; the coherence monitor provides the statistics that make this feasible.

As an abstract motivating example, say that a user asks a Linked Data cache: What are the current temperatures in European capital cities? Also, say that the cache endpoint has up-to-date information on which cities are capitals. However, say that this endpoint does not have knowledge (or perhaps only has partial knowledge) of which continent the cities are on; this information will have to be retrieved live. Furthermore, information about temperatures indexed by the cache endpoint are likely to be old; hence, to avoid stale results, this information should *only* be fetched live. Ideally, our hybrid query approach would use the cache endpoint to quickly return a list of capital cities, go live over the Web to check which ones are European, and then (also live) retrieve the current temperatures of these capitals directly from source. To enable this hybrid execution, we need knowledge of the coverage and freshness of the cache.

Finding an effective trade-off between fresh and fast results by combining live and centralised caches introduces a wide range of challenges. In this work, we contribute towards the realisation of our envisioned hybrid query engine [18] by further investigating two core aspects. First, we mention various alternatives for query planning and review their strengths and weaknesses for a black box scenario in which the participating query engines are not aware of their role. Second, we propose various coherence-estimate formulae needed for generating effective hybrid query plans, comparing them against each other and discussing different approaches to obtain these statistics from a black box centralised cache.

We continue this paper with some background on Linked Data query answering (Section 2). We discuss various alternatives for creating hybrid query plans in Section 3. Next, we propose methods to probe centralised caches so as to collect coherence estimates, and present results for two public Linked Data SPARQL engines (Section 4). We then conclude and discuss future work (Section 5).

2 Background

Centralised approaches for SPARQL query execution integrate data from different sources and are designed to provide (i) fast execution times by building optimised indexes and (ii) scalability through vertical partitioning or data distribution. However, constantly maintaining a *broad and fresh* coverage of remote data from millions of sources is unfeasible in such approaches. Current centralised SPARQL endpoints cover selected subsets of Linked Data, e.g., FactForge [2]; or aim to achieve a broad coverage of Linked Data, e.g., OpenLink's LOD cache[1] and the Sindice [10] SPARQL endpoint[2] (both powered by Virtuoso [3]).

Recently, various authors have proposed methods for executing SPARQL queries "live", by accessing remote data *in situ* and at runtime [8,10,14,16,4]. This guarantees up-to-date and complete results wrt. to the accessed data. Ladwig and Tran [8] categorise three variations of such approaches, which are (i) top-down, (ii) bottom-up, and (iii) mixed strategies. Top-down evaluation determines remote, query-relevant sources using a *source-selection index*, such as inverted-index structures [10], query-routing indexes [14], or lightweight hash-based structures [16]. Bottom-up query evaluation strategies discover relevant sources on-the-fly during the evaluation of queries, starting from a "seed set" of URIs taken from the query. The seminal proposal in this area is called *link-traversal based query execution* (LTBQE) [4]. In the third strategy [8], an initial seed list potentially containing more sources than actually relevant is generated in a top-down fashion. Additional relevant sources are discovered by using a bottom-up approach. In a sense, our hybrid-query proposals follow this strategy.

While the above approaches access raw data sources directly, an orthogonal approach to live querying is that of federated SPARQL, where queries are executed over a group of possibly remote endpoints [12,13,1]. Like in our proposal, federation involves splitting and delegating sub-queries to different engines. Federated SPARQL approaches either use service descriptions that are indexed locally and used to route queries [12,13], or rely on user-specified service URIs [1].

Recent works look to combine local (i.e., centralised) and remote (i.e., live) querying on a theoretical, engineering and social level (e.g., [5,18]). However, to the best of our knowledge, no one has looked at deciding which *parts* of a query are suitable for local/remote execution. Our work also relates to research on guaranteeing (Web) cache coherence [11] and semantic caching [7]. However, such systems typically rely entirely on cached data or completely discard it. Various authors have recently discussed invalidation of *internal* SPARQL caches [9,19] but rather focus on local index coherence, not on remote (Web) coherence.

3 Query Planner

Traditional query planning within closed systems focuses on optimising for performance by ordering the execution of query operators to minimise intermediate

[1] http://lod.openlinksw.com/sparql
[2] http://sparql.sindice.com/

results. Such query planning often relies on *selectivity estimates*, which indicate the amount of results a given operation will generate. For hybrid query planning, we wish to optimise for both speed *and* freshness. Thus, analogous to (and in combination with) selectivity estimates that optimise for speed, we need other metrics to optimise for freshness. Recalling that (sub-)queries will often be answered faster by materialised indexes than live engines, in the interest of speed, we wish to push as much of the query processing to the cache endpoint. However, we only want to send requests for which the cache has fresh data available. Hence, along with selectivity estimates, we also need *coherence estimates* to measure how well synchronised the cache is wrt. the Web. We will propose and test concrete measures and techniques for computing coherence estimates for a cache endpoint in Section 4. For now, we introduce high-level approaches for creating hybrid query plans that optimise for both speed and freshness.

Split types The core aim of the query planner is to decide which parts of the input query should go to the cache, which parts should go live, and how results can be combined. In terms of splitting the query, there are two high-level options:

Multi-split In this approach, there is no restriction placed on how many splits are made in the query or on which parts go where. This is the most general case. However, if the query is split into many parts, processing the query will involve a lot of coordination of interim results and synchronisation between the cache and the live engines.[3] Also, if the cache is accessed through a public interface, it may not allow a sufficient query rate for this approach to work.

Single-split Another simpler (but more restricted) option is to split the query into two: a cache and a live sub-query. Much less coordination is then required between engines. Assuming nested evaluation, an open question is whether the cache or live request should be run first.[4] We argue that going to the cache first makes more sense: (i) this would only require a single query to be run against the cache's materialised index, (ii) the cache can quickly return initial results, (iii) results from the cache give more information (bindings) to the live engine for finding query-relevant sources from the Web.

Reordering strategies As per traditional selectivity optimisation, before the query can be split, the query (join) tree needs to be reordered to decide which query primitives are executed first.[5] Here we also wish to optimise for freshness; hence, the ordering can include selectivity for speed and/or coherence for freshness.

Selectivity-based ordering Query patterns in the join tree are first ordered by selectivity. Selectivities for each pattern can be computed by rule-based

[3] Where supported, SPARQL 1.1 `VALUES` could be helpful to ship bindings, but would still require a synchronisation point for each split.

[4] If both parts of the query are deemed to have low selectivity, they could be run in parallel and hash-joined.

[5] We focus on optimising simple BGPs. Aside from features like `OPTIONAL`, `MINUS` and (`NOT`) `EXISTS`, other query features and optimisations can be layered above.

estimates or variable-counting techniques [13]), from analysis of prevalence of patterns in Web data, by sampling data from the cache using probe queries, or (where supported) by posing SPARQL 1.1 `COUNT` queries or queries for statistical summaries against the cache indexes. Patterns that match and return fewer data are executed earlier to minimise interim results.

Coherence-based ordering Query patterns are ordered by the coherence of cache data available for them; coherence gauges the coverage and freshness of cache indexes for answering a pattern (more coherence implies broader and fresher cache coverage). Coherence measures can be computed for patterns based on analysis of the dynamicity of Web data, using probe queries against the cache which can be compared with live results, or (assuming cooperation of the cache) by listening for updates to the cache indexes [19]. A single-split strategy is appropriate for this ordering, where patterns that can be best answered by the cache will be executed first and the rest then answered live.

Both orderings have inherent advantages and disadvantages. The selectivity-based ordering should minimise interim results but makes no guarantees about freshness. In particular, (in)coherent patterns will be mixed throughout the query tree. This raises another crucial question for the query planner: *where to split a query*. One option is to apply selectivity ordering but use coherence estimates to decide split positions, where either (i) a multi-split is used to fully leverage the performance of the cache while guaranteeing freshness or (ii) a single-split is used at the lowest incoherent pattern, which will minimise coordination but end up running coherent patterns live that the cache could answer. For single-splits, one could also use a pre-defined threshold of coherence to support user-defined expectations of freshness.

Conversely, the coherence-based ordering maximises freshness but makes no guarantees about the size of interim results. This ordering lends itself well to a single-split strategy, where all of the highly-coherent patterns can be first sent directly to the cache; for deciding where to split, a threshold can be used as mentioned before, or a simple fixed-position split strategy can be employed (e.g., always send only the least coherent pattern live). However, in this approach, the cache sub-query may have a low selectivity and require the cache to materialise a lot of results. In the best case, high selectivity and high coherence correlate with each other such that there is no conflict in the fresh vs. fast trade-off. However, as we will see for experiments in the next section, this is not necessarily the case.

Finally, we highlight that the hybrid query planner is responsible for ordering, splitting, delegating and executing sub-queries. The sub-queries sent to the cache or to the live engines are then subject to internal optimisations. This is particularly relevant for single-split coherence-ordering, where the sub-query delegated to the cache endpoint will be reordered according to internal selectivities.

In summary, hybrid query planning is a challenging problem and presents many alternative strategies to explore. We will leave further investigation of these alternatives for future work. For now, we focus on the extraction of coherence measures from public cache endpoints, as required for the above methods.

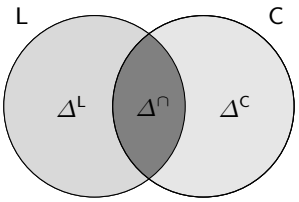

Fig. 2. Result set (Venn) diagram

```
SELECT ?sIn ?pIn ?oOut ?pOut
WHERE {
 ?sIn ?pIn <entityURI> .
 <entityURI> ?pOut ?oOut .
}
```

Fig. 3. Entity-query template

4 Coherence Estimation

In the general case, the degree of optimisation that can be achieved for a query depends on the quality of the statistics available to the query planner. As previously discussed, we first and foremost need coherence estimates to gauge the extent to which, for a given query pattern, the cached indexes are coherent with respect to current information available on the Web. We identify two main approaches to obtain knowledge about coherence for different query patterns.

Cache-independent estimates involve monitoring a large range of Linked Data sources to build a comprehensive, global picture of the dynamicity of the Web of Data. Previous empirical studies [17,15] have shown varying levels of dynamicity across Linked Data sources; furthermore, we speculate that dynamicity varies by the schema of data [17]. In term of benefits, cache-independent estimates can be applied generically to any store (and indeed to other use-cases) [6]; however, they give no indication as to the specific coverage or update rates, etc., of the cache engine at hand.

Cache-specific estimates involve periodically comparing the results of a centralised store against the current version of Linked Data on the Web. Versus a broad empirical study, cache-specific analysis is more sensitive to the particular update patterns and coverage of that store. As shown later in our experiments, analogous coherence estimates can vary for different caches.

For the above mentioned reason, we focus on *cache-specific estimates* where Figure 2 illustrates how the results from the live engine (L) and results from the cached centralised engine (C) may diverge. The cache may return results that live querying does not ($\Delta^{\mathsf{C}} := \mathsf{C} \setminus \mathsf{L}$), some of which may be stale, others of which may be accurate but involve a source that the live engine did not access. Conversely, the live approach may find answers that the cache could not ($\Delta^{\mathsf{L}} := \mathsf{L} \setminus \mathsf{C}$), some of which may be caused by remote data changing, others of which may be from sources that the store did not cache. Some of the results—which we deem to be coherent—are the same for the cache and the Web ($\Delta^{\cap} := \mathsf{L} \cap \mathsf{C}$).

In order to test coherence, we propose to probe both the cache's endpoint and the Web with a broad range of simple queries and compare the results,

characterising parts of the cache's index that are likely to be stale or missing. We view results as consisting of variable–binding pairs (i.e., C, $\mathsf{L} \subset \mathbf{V} \times \mathbf{UL}$ reusing common notation for the set of all query variables, URIs and literals resp.) and we exclude answers involving blank nodes to avoid issues of scoping and inconsistent labelling. We identified two possible methods by which queries can be used to test coherence.

Document-based estimates Assuming the SPARQL cache uses *Named Graphs* to track the original source of information on the Web, we can compare the data for a Web document against the data cached in the corresponding graph using `GRAPH` queries. However, (and as is the case for the two caches we test later) many stores do not have consistent naming of graphs: sometimes the graph may indeed refer to a particular Web document, but oftentimes the graph will be a high-level URI (e.g., `http://dbpedia.org`), informally indicating a dump from which the data were loaded but which cannot be directly retrieved.

Triple pattern estimates Thus, we instead focus on triple-based estimates. To ensure lightweight statistics with broad applicability, our notion of coherence for triple patterns centres around predicates. This restricts our approach to triple patterns with a constant as predicate; other patterns are assigned a default estimate. To derive a comparable set of results for generating triple-pattern estimates, we probe the cache and the live engine with a broad set of entity queries (see Figure 3). To quantify the coherence of predicates based on the results, we study two measures. To present these, we apply notation from Figure 2 to the results of the probe queries, adding subscripts to indicate results for a certain query, e.g., Δ^{L}_q. We denote results involving a predicate p as, e.g., $\Delta^{\mathsf{L}}_q(p) := \{r \in \Delta^{\mathsf{L}}_q : (\texttt{?pIn}, p) \in r \vee (\texttt{?pOut}, p) \in r\}$, and say $p \in \Delta^{\mathsf{L}}_q$ iff $\Delta^{\mathsf{L}}_q(p) \neq \emptyset$.

Query-based coherence: The coherence of a predicate p is measured as the ratio of queries for which p appeared in Δ^{L}. For the full set of queries $\mathcal{Q}$, let $M_q(p)$ denote for how many queries a live result involving the predicate was missing at least once in the cache results ($M_q(p) = |\{q \in \mathcal{Q} : p \in \Delta^{\mathsf{L}}_q\}|$). In addition, we count $L_q(p) = |\{q \in \mathcal{Q} : p \in \mathsf{L}_q\}|$ as the queries for which the live engine returned at least one result containing p. The *query-based coherence* of the predicate p is then computed as:

$$\mathsf{coh}_q(p) = 1 - \frac{M_q(p)}{L_q(p)} \quad .$$

Result-based coherence: For this measure, we inspect the ratio of missing results for a predicate p, rather than the fraction of stale queries. Let $M_r(p)$ denote the count of all live results involving the predicate p that were missed by the cache, summated across all queries ($M_r(p) = \sum_{q \in \mathcal{Q}} |\Delta^{\mathsf{L}}_q(p)|$). Let $L_r(p)$ denote the count of all results involving p retrieved by the live engine ($L_r(p) = \sum_{q \in \mathcal{Q}} |\mathsf{L}_q(p)|$). The *result-based coherence* is then:

$$\mathsf{coh}_r(p) = 1 - \frac{M_r(p)}{L_r(p)} \quad .$$

Experimental setup To test the different coherence-estimate proposals, we conducted experiments with two SPARQL stores that cache a broad range of Linked Data: "the Semantic Web Index" hosted by Sindice, and the "LOD Cache" hosted by OpenLink (see Section 2). We randomly sampled 12 thousand URIs from the 2011 Billion Triple Challenge dataset[6], which contains over 2.1 billion quadruples taken from over 7.4 million RDF/XML documents on 791 pay-level domains.[7] For each URI, we ran the entity query listed in Figure 3 against both caches and live with our LTBQE implementation, giving us a large set of results to compare. Experiments were run in early March 2012; we extracted coherence estimates for 2,550 predicates in OpenLink and 1,627 predicates in Sindice.

Results We first explored the difference between our two measures. We measured the correlation between both measures across all predicates using Kendall's τ which measures the agreement in ordering for two measures in a range of $[-1, 1]$, where -1 indicates perfectly inverted ordering and 1 indicates the exact same ordering. We found a τ value of 0.98 for OpenLink and 0.95 for Sindice, with a negligible p-value, indicating very strong agreement between both coherence measures. Henceforth, we use the result-based formula $\mathsf{coh}_r(.)$ as it returns more granular results—e.g., it had 8% fewer scores of precisely 0 than $\mathsf{coh}_q(.)$—and thus results in fewer ties and fewer trivial decisions when ordering patterns.

Using the $\mathsf{coh}_r(.)$ measure, we observed that 67% of the tested predicates in the OpenLink index are entirely up-to-date ($\mathsf{coh}_r(p) = 1$), versus 30% of the predicates for the Sindice endpoint. In contrast, information for 14% of the tested predicates for OpenLink are entirely missing or out-of-date ($\mathsf{coh}_r(p) = 0$), versus 40% for Sindice; these high percentages are due to partial coverage of Web sources, outdated data-dumps in the index, and predicates with dynamic values. Hence we see that these caches are not up-to-date for many predicates, which motivates our current investigation of hybrid-query techniques.

Furthermore, we analysed the correlation for coherence estimates of the same predicates *across* both endpoints. The τ-score was 0.16, again with a negligible p-value. The low correlation highlights the endpoint-specific nature of these measures: the hybrid query approach tackles both the endpoint-independent problem of dynamicity and the endpoint-specific problem of index coverage and updates.

Finally, we looked at the correlation between the selectivity of predicates (i.e., how often they occur) and their coherence, which may have potential consequences for query planning. Specifically, for each endpoint, we compared the number of (live) results generated for each predicate across all queries and their $\mathsf{coh}_r(p)$ value. The τ-value for OpenLink was 0.1, indicating that less selective patterns tend to have slightly lower coherence; the analogous τ-value for Sindice was -0.03, indicating a negligible correlation in the opposite direction. Though limited, we take this as anecdotal evidence to suggest that correlation between

[6] http://challenge.semanticweb.org/

[7] We considered using the SPARQL 1.1 `SAMPLE` keyword, but (i) Virtuoso does not support SPARQL 1.1; (ii) `SAMPLE` does not guarantee randomness of results. A pay-level-domain is the level that one pays to register (e.g., `bbc.co.uk`, `dbpedia.org`).

the selectivity and coherence of predicates is weak, if any. This observation makes the choice between selectivity- and coherence-based ordering even more crucial.

Predicate–domain estimates So far, we naïvely assume a single coherence value for predicates in all cases, ignoring subject or object URIs: keeping information for each subject/object would have a high overhead. However, the coherence of predicates may vary depending on the site from which the subject/object originated. We can thus generalise subject/object values into pay-level-domains (PLD) and then track coherence for predicate–domain pairs. We mapped the entity URIs of the queries to their PLDs (581 PLDs with a maximum of 74 queries per domain) and resolved the coherence of predicates for individual PLDs. We found that the difference between coherence values across all PLD pairs was ≤ 0.1 for $\sim$40% of the OpenLink and $\sim$15% of the Sindice predicates. All remaining predicates exhibited a higher variance for coherence values across different PLDs.

Maintenance Assuming the cooperation of the endpoint, various methods can be used to learn about content changes or updates to the centralised index (similar in principle to internal SPARQL caching proposals [19]). Data providers may push change notifications to the endpoints and/or the endpoints can learn about changes by actively monitoring remote sources [15]; this information can then be pushed to the coherence monitor. In a strict black box scenario, where only the public SPARQL interface is available for the cache, one has to periodically rerun or update the gathered statistics, where queries that are observed to return static results could be probed less frequently in an adaptive monitoring setup [6]. In addition, the system could perform a demand-based or "lazy" maintenance of statistics that is based on, e.g., (i) keeping only frequent query patterns up-to-date or (ii) actively updating coherence estimates as hybrid queries are processed.

5 Conclusion

In this paper, we proposed a hybrid query architecture that aims to combine the strengths of centralised cache endpoints, which provide fast results, and the strengths of novel live querying approaches, which provide fresh results from a broad range of Linked Data sources. First, we proposed different techniques by which coherence and selectivity estimates can be combined with reordering and hybrid-split strategies to design an effective hybrid query plan that (potentially) speeds up live results while freshening up cache results. Second, we discussed different coherence estimate formulae and various methods of extracting the necessary information from endpoints to compute the values, e.g., based on probing queries. We conducted an empirical study based on two public endpoints and showed that data are often stale or missing, and that the proposed coherence measures are indeed well-suited to identify this. Furthermore, we showed that the coherence estimates differ for the same predicates across different endpoints and also across different data providers for the same endpoints.

Given the potential scope and dynamicity of Linked Data, query engines will need to employ a wide range of techniques to efficiently offer fresh results with

broad coverage. We believe that our hybrid query proposals make a significant step in this direction by combining both centralised and decentralised query paradigms to exploit both cached and live results. Towards making our proposals a concrete reality, our next steps involve implementing, evaluating and further exploring the combinations of hybrid query techniques presented herein.

Acknowledgements: *This research has been supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-II).*

References

1. C. B. Aranda, M. Arenas, and Ó. Corcho. Semantics and optimization of the SPARQL 1.1 Federation extension. In *ESWC*, 2011.
2. B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. FactForge: A fast track to the web of data. *SWJ*, 2011.
3. O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. In *Networked Knowledge – Networked Media*. Springer, 2009.
4. O. Hartig, C. Bizer, and J. C. Freytag. Executing SPARQL queries over the web of Linked Data. In *ISWC*, 2009.
5. O. Hartig and A. Langegger. A database perspective on consuming Linked Data on the web. *Datenbank-Spektrum*, 2010.
6. T. Käfer, J. Umbrich, A. Hogan, and A. Polleres. Towards a Dynamic Linked Data Observatory. In *LDOW at WWW*, 2012.
7. M. Karnstedt, K. Sattler, I. Geist, and H. Höpfner. Semantic Caching in Ontology-based Mediator Systems. In *"Web und Datenbanken", Berliner XML-Tage*, 2003.
8. G. Ladwig and T. Tran. Linked Data query processing strategies. In *ISWC*, 2010.
9. M. Martin, J. Unbehauen, and S. Auer. Improving the performance of Semantic Web applications with SPARQL query caching. In *ESWC*, 2010.
10. E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 2008.
11. S. Podlipnig and L. Böszörményi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 2003.
12. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, 2008.
13. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: A federation layer for distributed query processing on linked open data. In *ISWC*, 2011.
14. T. Tran, L. Zhang, and R. Studer. Summary models for routing keywords to Linked Data sources. In *ISWC*, 2010.
15. J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, and S. Decker. Towards dataset dynamics: Change frequency of linked open data sources. In *LDOW*, 2010.
16. J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over Linked Data. *WWWJ*, 2011.
17. J. Umbrich, M. Karnstedt, and S. Land. Towards understanding the changing web: Mining the dynamics of linked-data sources and entities. In *KDML*, 2010.
18. J. Umbrich, M. Karnstedt, J. X. Parreira, A. Polleres, and M. Hauswirth. Linked Data and Live Querying for Enabling Support Platforms for Web Dataspaces . In *DESWEB at ICDE*, 2012.
19. G. T. Williams and J. Weaver. Enabling fine-grained HTTP caching of SPARQL query results. In *ISWC*, 2011.