Learning from the History of Distributed Query Processing

A Heretic View on Linked Data Management

Heiko Betz[1], Francis Gropengießer[1], Katja Hose[2], and Kai-Uwe Sattler[1]

[1] Ilmenau University of Technology, Germany,
{first.last}@tu-ilmenau.de
[2] Dept. of Computer Science, Aalborg University, Denmark,
khose@cs.aau.dk

Abstract. The vision of the Semantic Web has triggered the development of various new applications and opened up new directions in research. Recently, much effort has been put into the development of techniques for query processing over Linked Data. Being based upon techniques originally developed for distributed and federated databases, some of them inherit the same or similar problems. Thus, the goal of this paper is to point out pitfalls that the previous generation of researchers has already encountered and to introduce the Linked Data as a Service as an idea that has the potential to solve the problem in some scenarios. Hence, this paper discusses nine theses about Linked Data processing and sketches a research agenda for future endeavors in the area of Linked Data processing.

1 Introduction

The vision of the Semantic Web [2], i.e., the idea of having data on the Web defined and linked in a way that it can be used by machines (and humans), which automatically integrate and reuse data across various applications, has triggered the development of various new applications, and has opened up new directions in research. Basic requirements to turn this vision into reality is to use certain standards, establish links, and describe relationships between data sets that are available on the Web. The term *Linked Data*, or Linked Open Data (LOD) respectively, refers to a set of best practices for publishing and connecting structured data on the Web[3]. Tim Berners-Lee outlined a set of rules (often referred to as the "Linked Data principles" or "Design Issues") that describe how the data should be published – in practical use RDF and SPARQL have become the de-facto standards.

With the growing popularity, the number of data sources and the amount of data has been growing very fast in recent years. In its current state the Linking Open Data cloud, i.e., the cloud of available data sets published as Linked Open Data, consists of 295 data sets with a total of about 31 billion triples (as of November 2011) – making the concept of Linked Data one of the most successful trends in the Semantic Web community.

[3] http://www.w3.org/DesignIssues/LinkedData.html

Despite this success, working with Linked Data and its application still has to overcome several challenges. In addition to the problems of publishing data and assessing data quality, querying and analyzing Linked Data are ongoing research issues. With the data being distributed among multiple sources, query answering usually involves multiple nodes. Current research on query processing over Linked Data can be roughly split up into three categories: Lookup-based Query Processing (LQP), Federated Query Processing (FQP), and Materialization-based Query Processing (MQP).

However, many of the proposed LQP and FQP approaches rely to some extent on techniques from classic distributed query processing or even resemble techniques developed decades ago. So, instead of reinventing the wheel again, we propose to pay attention to lessons already learned from existing database research on distributed and federated query processing.

Hence, the goal of this paper is to point out what we can learn from federated query processing in database research and make the reader aware of several myths and open research questions. Thus, Section 2 discusses the state of the art in Linked Data query processing and distributed/federated query processing in databases. Section 3 presents nine theses discussing myths and pitfalls in Linked Data query processing. Section 4 proposes *Linked Data as a Service (LDaaS)* as a direction of future research and identifies several challenges that have to be addressed to achieve this goal. Finally, Section 5 concludes the paper.

2 State of the Art in Query Processing

Linked Data Query Processing As already mentioned in the introduction there are three main approaches towards evaluating structured queries over Linked Data: Lookup-based Query Processing (LQP), Federated Query Processing (FQP), and Materialization-based Query Processing (MQP).

LQP approaches download the data from the sources during query processing, i.e., for each query all relevant data is downloaded from the sources and the query is evaluated at a central instance over all the downloaded data. One of the techniques proposed in the literature, explorative query processing [18, 23], exploits the very nature of Linked Data by iteratively evaluating and downloading data for URIs representing results for parts of the query. This process requires only little cooperation from the sources. Instead of determining relevant sources based on intermediate query results, alternatively indexes [17] describing the data provided by the sources can be used.

In case of FQP and distributed database systems, an optimizer needs to identify relevant sources and determine subqueries that can be evaluated directly at the sources without having to transfer the actual data. Several approaches that all inherit characteristics from federated query processing are proposed for Linked Data [7, 12, 24, 32, 36], mostly using SPARQL and RDF.

The third class, MQP, adopts the idea of data warehouses, collects all the data in advance and combines it into a centralized triple store. Centralized SPARQL query processing has been a hot topic of research in the past couple of years [27, 39]. Recently, the use of MapReduce [10] and cluster technology has been proposed to increase efficiency of these centralized solutions [9, 20, 21, 30].

Distributed and Federated Query Processing in Databases Distributed query processing in databases dates back to the late 1970ies. In these years, important prototype systems such as SDD-1, Distributed INGRES, and R* were developed. SDD-1 pioneered for instance optimization techniques and semijoin strategies, INGRES and R* contributed further techniques.

The process of distributed query processing – which was already developed within these systems – follows the conventional query processing strategies in database systems. For the distributed case, the rewriting and optimization phases are extended: during rewriting global relations have to be replaced by the corresponding fragmentation and reconstruction expression – a step that is called data localization. The optimization is usually split into conventional local optimization for choosing access strategies and the global optimization where join ordering and site selection are the main tasks. For site selection two fundamental options exist: data shipping where the data is transfered from the storing site to the site executing the query and query shipping where the evaluation of the (sub-)query is delegated to the storing site. In addition, hybrid strategies are also possible [11]. Furthermore, caching of results may help to reduce the communication costs, but requires to balance between the benefit of answering queries from the local cache and the costs for maintaining the cache [22].

Another important area are strategies for distributed join processing. For joining two relations stored at different sites two basic strategies can be performed: ship whole and fetch matches. Based on this, special strategies aiming at reducing the transfer costs were developed, e.g. semijoins and hashfilter joins. The problem of bursty and delayed arrivals of tuples was also addressed by particular techniques such as the XJoin [38].

In federated scenarios where heterogeneities at data, schema, and system level may exist, sometimes some of the source systems (endpoints) do not allow fetching the whole table or evaluating a join but support only parameterized selections. For this cases, the bind join [34] was proposed which is a variant of the fetch matches strategy.

However, none of the classic distributed and federated databases solutions (including commercial products) was designed to be scalable over hundreds of nodes. Providing transparent access to remote data and guaranteeing ACID properties is achieved using global schema information as well as query and transaction coordinators. Also, estimating query costs to be able to select the optimal execution plan (or better to avoid worst plans) is a challenging task, particularly in heterogeneous settings [33]. In the Linked Data scenario with RDF as a rather loosely structured data model (in contrast to comparatively well-structured SQL schemas) and SPARQL endpoints implemented in very different ways (ranging from file-based SPARQL processors to DBMS-based solutions) this problem is even more complicated.

3 Theses on Linked Data Query Processing

In this section, we list several assumptions, myths, and theses that are frequently mentioned in the research literature to motivate FQP and that can be considered hindrances for a Linked Data as a Service approach. For each thesis, we

discuss arguments to refute it and derive research challenges which we present in Section 4.

(T1) The volume of Linked Data is too big for centralized management. Due to the increasing amount of Linked Data, it seems to be almost impossible to store it in a centralized system in a native uncompressed manner. However, by using certain encoding approaches such as dictionary encoding and/or prefix encoding, the volume of Linked Data can be reduced tremendously. As an example, we consider the English DBpedia[4] version 3.7. The total number of triples is 386,546,905, which uncompressed requires 50 GB of disk space. However, the data contains much redundancy: only 23,645,703 subjects, 51,583 predicates, and 75,867,838 objects are unique. Applying dictionary encoding, which is comparable to the RDF `HDT`[5] format, reduces the size to 10,016 MB, 4,118 MB for the dictionary, and 5,898 MB for the triple data. Based on the mentioned dataset, we calculated an average string length of approx. 44 characters (measured) and assume a character size of one byte. By extrapolating based on these statistics we can approximate the required space to store an additional 1 million triples by approx. 26.3 MB (15.5 MB for triple data + 10.8 MB for strings). In consequence, saving the current LOD cloud with 31 billion triples results in approx. 800 GB. Thus, even in consideration of additional Linked Data sources and formats, by applying advanced compression techniques the data could be handled using the MQP approach in a single instance or small cluster database.

(T2) Materialization in centralized repositories violates data authority. Having a closer look at the LOD cloud, one can observe that many authors are not interested in giving up their copyrights[6]. Hence, it is not possible to materialize and integrate the affected data sets in centralized repositories. One reason for this behavior is the fear of losing data authority. A possible solution to overcome this problem is the application of multi-tenant techniques which are already used in Cloud services like Amazon S3. In this way, the owner of the data has the ability to control access to the data in a fine-grained manner. Another solution would be to change the license model. The Creative Commons license[7] (CC) keeps the name or source of the author and therefore allows common usage. To use this technique, each triple has to be expanded to a quad tuple with the fourth value describing the origin/source of the triple.

(T3) Scalability can be achieved only by distributed query processing. A widely spread assumption is that scalable query processing can only be achieved in a distributed environment. A main precondition in order to hold this assumption is that global query optimization can take place. Therefore, control about the participating nodes is necessary [19]. In a federated system, each node (source) acts autonomously, which makes it difficult to apply typical optimization techniques like, for example, pipelining. A main consequence is that additional

[4] `http://dbpedia.org`
[5] `http://www.w3.org/Submission/2011/03/`
[6] `http://thedatahub.org/group/lodcloud`
[7] `http://creativecommons.org/`

data shipping between the coordinator and the participating nodes is necessary. In [16] several approaches to federated query processing are discussed and compared to the central (warehouse) solution. On average, the centralized approach provides better query performance. Only simple, highly selective queries profit from the parallel execution within a federated system. At the latest, if complex joins have to be processed, query performance decreases dramatically in a federated system that does not allow cooperation beyond the standard. Here, network latency hampers the transport of big data volumes leading to non-predictable query response times.

In order to mitigate the problems of data shipping over high-latency networks in world-wide distributed query processing scenarios, cluster-based approaches are a promising solution. Infrastructure as a Service solutions like Amazon EC2 provide high-throughput inter-machine-connections as well as fast access to Storage as a Service solutions like Amazon S3. They combine parallelism as well as reliability and availability.

(T4) Linked Data processing is only about SPARQL processing. Although a remarkable fraction of Linked Data processing consists of SPARQL processing, data analyzing and reasoning scenarios – which in fact go far beyond simple SPARQL processing – are gaining more and more importance. Especially, the analysis of the increasing amount of spatial, temporal as well as stream data is a big research challenge. Current work in this research area shows that there is a need for appropriate SPARQL language extensions as well as for efficient and scalable implementations of operations such as spatial and temporal joins. Examples are stSPARQL (implemented in the STRABON system[8]) and SPARQL-ST [31], which constitute first language extensions that support spatio-temporal queries. [15] proposes a first approach for an RDF stream engine. Furthermore, C-SPARQL [1], SPARQLStream [8], CQELS [25], and [4] are first language extensions which introduce the ability to query linked data streams.

(T5) The problem of semantic heterogeneity can be solved by using ontologies. The problem of semantic heterogeneity is a well-known problem in the context of data integration systems; whenever the data of multiple independent sources needs to be merged, we have to consider the problem of integrating the data correctly into one consistent data set. To solve this problem, some schema integration approaches make use of ontologies, where each ontology models the interpreter's understanding of the world. Exploiting mappings of terms used in schema definitions to an ontology, the similarity of schema elements can be determined based on the explicit definitions and relationships defined in the ontology. In case different sources provide mappings to different ontologies, an additional mapping between the ontologies is needed. In this sense, ontologies can also help to identify duplicate entities modeled in heterogeneous sources.

Similar problems also arise for Linked Open Data. Different sources use different ontologies to model their RDF knowledge bases so that when trying to build a consistent merged knowledge base or answering queries over multiple sources, classes and predicates need to be mapped. However, so far Linked Data

[8] `http://www.strabon.di.uoa.gr`

links are mostly considered only on instance level (connecting entity URIs via `owl:sameAs`), establishing links between classes and predicates is less common.

Thus, when processing queries over Linked Data we have to deal with similar problems as data integration systems. When merging the data of multiple sources, we still have to define one consistent vocabulary by finding mappings between the involved ontologies. Furthermore, with the two levels (instance and schema) no longer separated in contrast to classic database systems, the problem of semantic heterogeneity has not become easier. Thus, similar to data integration systems the use of ontologies does not completely solve the problem of semantic heterogeneity but changes it.

(T6) HTTP URIs can be used to identify endpoints. Following Tim Berners-Lee's rules for Linked Data, information should be published with HTTP URIs, so that people can find them. In RDF as a widely used format for Linked Data, it is only required that subjects and predicates are deferenceable URIs, objects might only be described with literals without involving URIs. As a consequence, looking up additional information for literal objects leads to the problem of selecting relevant sources. Therefore, additional meta information is required. Although most data sources offer meta information describing the stored data sets, this information is mostly not intended for automatic processing. In these cases, the decision whether a data source contains desired information or not can only be made by the user. Building indexes summarizing the content is a possible solution [17, 28]. An alternative solution is centralized caching and materialization of frequently used data.

(T7) The freshness of data is guaranteed only with distributed query processing. It is a commonly known fact that in order to get the freshest data, one has to query the source directly. Hence, in contrast to centralized repositories, federated solutions can guarantee that the data is always up-to-date. However, there are also ways to guarantee the freshness of the data in centralized solutions. One of the main observations is that many data sources in the LOD cloud are updated only rarely. Examples are US Census (no update since creation), Linked Sensor Data (no update since 2008), and Source Code Ecosystem Linked Data (SECOLD) (update once per year). In addition, retrieving the data from the sources can be done very efficiently using source scheduling [28]. The source is only revisited after certain waiting times and if the content has changed. Change detection is performed by using the HTTP `last-modified-since` header or some content hash – if provided by the sources. Integrating the new data into the centralized repository is a well-known problem in the context of the ETL process in data warehouse applications. Furthermore, data sources such as DBpedia started publishing live incremental updates, containing new triples and removed triples in separate files. These updates can easily be applied to the repository.

An alternative to these pull strategies is to extend sources with a push functionality. Each change is directly pushed into the central repository so that data freshness is guaranteed. Some libraries implementing the functionality are already available. Still, the extend to which existing solutions can be applied depends on the degree of cooperation offered by the sources.

(T8) Open Data is accessible as Linked Data. As already stated in the introduction, both the Open Data and Linked Data movements are a great success. Even governments and public agencies have started to publish data of common interest. The number of publicly available data sources is continuously growing. Obviously, this data is open but is it really linked? Does it fulfill the necessary rules to be processed in an automatic fashion?

The Open Data Survey[9] examined more than fifty Open Data platforms, driven by governments and international organizations. A key finding of this endeavor is that many open data sources available today do not meet the rules originally stated by Tim Berners-Lee. Particularly, they strongly vary in technical aspects. Data sources vary for example in the used file formats, access APIs, and schemas. Only 8% of the examined data sources actually act as SPARQL or SQL endpoints.

In this environment, enabling users to search for information in a convenient way instead of browsing through thousands of non-linked documents in different file formats requires data integration. Techniques for schema integration, data cleaning, and data transformation – already known from ETL processes – are necessary to overcome the data source heterogeneity.

(T9) Centralized Linked Data is not Linked Data anymore. One could argue that Linked Data is inherently distributed data because the rules state among others that URIs should be used for naming things and that HTTP URIs are used to allow to look up these names. Although this principle allows to identify and refer to remote data sources, it does not necessarily mean that URIs have to describe a real physical location. In fact, URIs can be interpreted just as keys and used for indexing data items and therefore provide a way to partition huge datasets. Hence, Linked Data processing does not necessarily require distributed processing of queries or even fetching data from different files.

4 A Research Agenda towards Linked Data as a Service

Based on the observations discussed in Section 3 we propose to consider Linked Data not as distributed data per se. Instead, Linked Data can alternatively be considered as an example of *Data as a Service* where commonly used data is stored at a central place (MQP) and made available to consumers in a timely and cost effective manner. Of course, there are limits of applicability and this might not work for all sources but it represents an alternative for a large number.

The success of sites such as Flickr and YouTube, which pioneered this principle with very specific data collections, has inspired similar approaches for general datasets. Examples of Data as a Service are Windows Azure Marketplace DataMarket[10] (formerly known as Dallas), where various sets of data are offered ranging from aggregated stock market data over weather data, open data collections from organizations and governments to sports statistics. Further examples are InfoChimps[11] or Factual[12].

[9] `http://wwwdb.inf.tu-dresden.de/opendatasurvey/`
[10] `http://datamarket.azure.com/browse/Data`
[11] `http://www.infochimps.com/marketplace`
[12] `http://www.factual.com/data-apis/places`

Providing the LOD cloud as a nucleus for Open Data on such a platform and combine it with sophisticated SPARQL processing capabilities as well as additional support beyond the current standard protocols such as OData[13] or GData[14] would open new opportunities for data producers and consumers but also raise challenges for data management techniques. Such a *Linked Data as a Service* (LDaaS) platform could play multiple roles: a central registry and market place for datasets, but also a cache and data processing service decoupling the data producers who are not willing or able to provide processing capacities for data consumers. In the following paragraphs, we discuss a list of challenges that we think have to be addressed towards achieving this goal.

(C1) Linked Data Processing beyond SPARQL processing. As already mentioned in Section 3 (thesis T4), Linked Data processing requires much more than only SPARQL processing. Although several approaches for stream data as well as spatial and temporal data processing have been proposed, those approaches have two drawbacks. First, current techniques for reasoning over Linked Data in a distributed fashion are iterative and hence very expensive. Second, currently available SPARQL extensions lack standardization. The current SPARQL standard does not contain any aggregation functions as well as support for nested, spatial, and temporal queries. Aggregation functions and nested query support will be included in SPARQL 1.1[15] – but this is still a draft. The vision is to have an LDaaS platform, which integrates all necessary techniques for analyzing and reasoning over spatial, temporal, and stream data into one environment that is accessible via a standardized language.

(C2) Exploit newly available Infrastructure/Platform as a Service. Within the past few years, research mainly focused on developing solutions for different kinds of problems that can be solved on commodity hardware in unreliable environments. Much effort has been spent on distributed query processing in unreliable P2P environments and MapReduce [10]. The reasons mainly were the costs for the purchase and maintenance of big clusters. Currently, the idea of clusters and data centers experiences a rebirth under the term *Cloud*, offering new benefits and opportunities. Cloud providers such as Amazon, Microsoft, or Google offer infrastructure, storage, and even software as a service to customers following a pay-per-use model. They provide the illusion of infinite resources and guarantee availability and reachability. The computational power of a big cluster can be rented for the time for which it is actually needed.

There are some advantages that come along with the infrastructure as a server paradigm, e.g., Amazon EC2. Due to virtualization techniques, inter-virtual-machine-communication-costs are reduced tremendously either because they are run on the same physical host or because they are transparently connected via high capacity networks. Hence, they can mitigate the problems stated in Section 3 (thesis T3). Moreover, because of availability guarantees of over 99%, system failures can almost be neglected. In the unlikely event of a virtual machine

[13] `http://www.odata.org`
[14] `http://developers.google.com/gdata`
[15] `http://www.w3.org/TR/sparql11-query`

failure, monitoring services such as Amazon CloudWatch and virtual machine controller tools such as Amazon Command Line Tools enable easy system recovery. Thus, it is not necessary to deal with fail-stop failure scenarios. Further advantages of infrastructure as a service are that usually automatic load balancing (Amazon EC2) and scalability (Google AppEngine) as well as fast access to infinite storage such as Google BigTable or Amazon S3 are provided.

First steps in using these new Cloud services are already made with the CommonCrawl Project[16]. Here, Amazon's Elastic MapReduce Service can be used to analyze Web content stored in Amazon S3.

(C3) Address the opportunities of modern hardware architectures for query processing. For databases, processing big data sets is still a challenge. First, IO costs are still the dominating factor – there is a big gap in access time between external disks and main memory. Database researchers and vendors have addressed this issue with the following approaches:

(1) Reduce the amount of data to be read and fetch the data as fast as possible from disk. Examples of such strategies are index structures but also specialized data layout strategies such as column stores.
(2) Avoid IO operations completely by keeping and processing all data in main memory [5]. Today, single machines with more than 2 TB of RAM are available and new architectures such as RAMCloud [29] have been proposed, which can easily store the LOD cloud based on the estimation given in Section 3 (thesis T1). However, as shown in [6], there is another gap in access time between main memory and cache, which has to be also taken into account by cache-aware algorithms.
(3) Finally, as stated in [37], further improvements of performance require the exploitation of concurrency and parallelization.

Although parallel query processing is a well-studied problem [13, 14], modern multi-core and many-core architectures pose new challenges and opportunities for fine-grained parallelization at all levels ranging from intra-operator level up to the workload level.

(C4) Realistic benchmarking and metrics. Benchmarking is an essential task to evaluate query processing techniques. Benchmarks are important not only to compare different approaches but also to improve performance of systems. This can be seen in the relational database world, where the benchmarks of the Transaction Processing Council (TPC) are widely accepted by the major vendors and help to improve performance by orders of magnitude. For processing RDF data in general as well as particularly Linked Data, also several benchmarks have been proposed in recent years. Examples are the DBpedia benchmark, its successor the Berlin SPARQL benchmark (BSBM), and FedBench [35] for federated systems.

However, compared to the maturity of the commercial TPC benchmarks, query processing benchmarks for RDF and Linked Data might have to be improved. First, a benchmark should model a representative scenario – in case of TPC these are, among others, TPC-C for transaction processing and TPC-H

[16] `http://commoncrawl.org`

for analytical queries. Though, BSBM allows to measure the performance of storage systems that expose SPARQL endpoints [3], it models an e-commerce scenario and the query mix does not really reflect the patterns of typical queries on Linked Data. Second, appropriate metrics are needed to capture the performance goals, e.g., execution times for queries in isolation, throughput in terms of queries per time in case of concurrent queries as the composite Query-per-Hour Performance Metric at a given database size (QphH@Size) in TPC-H, or even price/performance metrics similar to TPC-H's $/QphH@Size.

A first step towards appropriate benchmarking of Linked Data query processing is done with FedBench [35]. This comprehensive benchmark suite particularly addresses the heterogeneity in federated systems. Still, with the vision of LDaaS and aspects such as aggregation, reasoning, and streams, it will be necessary to develop further benchmarks.

(C5) Simplify publishing and exploit crowdsourcing. The first obstacle for publishing Linked Data, which was already mentioned in Section 3 (thesis T8), is that the native data format for many datasets is not RDF. Instead, the data needs to be converted into RDF, e.g., from relational databases, Excel files, or by applying information extraction on text documents. In the ideal case, before publishing the data, it should also be cleaned, contradicting or false triples should be removed, duplicates should be detected and merged, etc. When merging data from different sources, this can become an expensive and time-consuming process, which might also interfere with licensing issues as mentioned in Section 3 (thesis T2) .

The problem is that all these steps are very time-consuming and involve human interaction to run properly. Whereas data cleaning creates added value for the data owner, this is not always true for links to other data sets. Usually, only links to a subset of available data sets are interesting for the publisher, links to others are therefore not detected and encoded. Consumers of the published data, however, might be interested in the links to other sources but they cannot easily add links to the original data set. Finally, making the data accessible via SPARQL endpoints involves hardware and resources and does not directly represent any added value to the publisher.

The whole publishing process could very much benefit from crowdsourcing in different ways. In today's web of data, if missing or wrong information is detected, the only way to change the data is to communicate with the publisher – which is often unsuccessful. First initiatives[17] and visions on collaborative knowledge networks [26] are already available but it is still a long way until crowdsourcing can be used efficiently and the process of publishing the data is easy enough with low effort so that more people are encouraged to publish Linked Data.

5 Conclusion

In this paper, we have discussed the state-of-the-art in Linked Data query processing and pointed out several pitfalls that research in the context of distributed

[17] `http://pedantic-web.org/`

and federated query processing has already identified. We have also discussed nine wide-spread myths about Linked Data query processing that researchers and practitioners occasionally encounter. Based on these observations and myths, we proposed a Linked Data as a Service (LDaaS) platform and a research agenda towards reaching this goal. The next few years will show to what extent researchers learned from the history of distributed and federated query processing and which parts of the research agenda were the most challenging ones.

References

1. D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Querying RDF streams with C-SPARQL. *SIGMOD Rec.*, 39(1):20–26, Sept. 2010.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 29–37, 2001.
3. C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
4. A. Bolles, M. Grawunder, and J. Jacobi. Streaming SPARQL - Extending SPARQL to process data streams. In *ESWC'08*, 2008.
5. P. A. Boncz. Main Memory DBMS. In *Encyclopedia of Database Systems*, pages 1669–1670. 2009.
6. P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in MonetDB. *Commun. ACM*, 51(12):77–85, 2008.
7. C. Buil-Aranda, M. Arenas, and Ó. Corcho. Semantics and Optimization of the SPARQL 1.1 Federation Extension. In *ESWC (2)*, pages 1–15, 2011.
8. J.-P. Calbimonte, O. Corcho, and A. J. G. Gray. Enabling ontology-based access to streaming data sources. In *ISWC'10*, pages 96–111, 2010.
9. H. Choi, J. Son, Y. Cho, M. K. Sung, and Y. D. Chung. SPIDER: a system for scalable, parallel / distributed evaluation of large-scale RDF data. In *CIKM*, pages 2087–2088, 2009.
10. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
11. M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance Tradeoffs for Client-Server Query Processing. In *SIGMOD Conference 1996*, pages 149–160, 1996.
12. O. Görlitz and S. Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *COLD'11*, 2011.
13. G. Graefe. Encapsulation of Parallelism in the Volcano Query Processing System. In *SIGMOD Conference 1990*, pages 102–111, 1990.
14. G. Graefe. Parallel Query Execution Algorithms. In *Encyclopedia of Database Systems*, pages 2030–2035. 2009.
15. S. Groppe, J. Groppe, D. Kukulenz, and V. Linnemann. A sparql engine for streaming rdf data. In *SITIS'07*, pages 167–174. IEEE, 2007.
16. P. Haase, T. Mathäß, and M. Ziller. An evaluation of approaches to federated query processing over linked data. In *I-SEMANTICS'10*, pages 5:1–5:9, 2010.
17. 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'10*, pages 411–420, 2010.
18. O. Hartig. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *ESWC (1)*, pages 154–169, 2011.
19. K. Hose, R. Schenkel, M. Theobald, and G. Weikum. Database Foundations for Scalable RDF Processing. In *Reasoning Web*, volume 6848 of *Lecture Notes in Computer Science*, pages 202–249. Springer, 2011.

20. J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL Querying of Large RDF Graphs. *PVLDB*, 4(11):1123–1134, 2011.
21. M. F. Husain, J. P. McGlothlin, M. M. Masud, L. R. Khan, and B. M. Thuraisingham. Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. *IEEE Trans. Knowl. Data Eng.*, 23(9):1312–1327, 2011.
22. D. Kossmann, M. J. Franklin, and G. Drasch. Cache investment: integrating query optimization and distributed data placement. *ACM Trans. Database Syst.*, 25(4):517–558, 2000.
23. G. Ladwig and T. Tran. SIHJoin: Querying Remote and Local Linked Data. In *ESWC (1)*, pages 139–153, 2011.
24. A. Langegger and W. W. M. Blöchl. A Semantic Web middleware for virtual data integration on the Web. In *ESWC*, pages 493–507, 2008.
25. D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC'11*, pages 370–388, 2011.
26. S. Metzger, K. Hose, and R. Schenkel. Colledge - A Vision of Collaborative Knowledge Networks. In *SSW'12*, 2012.
27. T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.
28. E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *Int. J. of Metadata and Semantics and Ontologies*, 3:37–52, 2008.
29. J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for RAMCloud. *Commun. ACM*, 54(7):121–130, July 2011.
30. N. Papailiou, I. Konstantinou, D. Tsoumakos, and N. Koziris. H2RDF: adaptive query processing on RDF data in the cloud. In *WWW*, pages 397–400, 2012.
31. M. Perry, P. Jain, and A. P. Sheth. SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries. In *Geospatial Semantics and the Semantic Web*, volume 12, pages 61–86, 2011.
32. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, pages 524–538, 2008.
33. M. T. Roth, F. Ozcan, and L. M. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *VLDB 1999*, pages 599–610, 1999.
34. M. T. Roth and P. M. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *VLDB '97*, pages 266–275, 1997.
35. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: a benchmark suite for federated semantic data query processing. In *ISWC'11*, pages 585–600, 2011.
36. 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.
37. H. Sutter. The Free Lunch is Over. Dr. Dobb's Journal, 30(3), March 2005.
38. T. Urhan and M. J. Franklin. XJoin: A Reactively-Scheduled Pipelined Join Operator. *IEEE Data Engineering Bulletin*, 23:2000, 2000.
39. L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. gStore: answering SPARQL queries via subgraph matching. *Proc. VLDB Endow.*, 4(8):482–493, May 2011.