Distributed RDF Query Processing and Reasoning in Peer-to-Peer Networks

Zoi Kaoudi*

Dept. of Informatics and Telecommunications
National and Kapodistrian University of Athens, Greece

Abstract. With the vast amount of available RDF data sources on the Web increasing rapidly, there is an urgent need for RDF data management and RDFS reasoning. In this thesis, we focus on distributed RDF data management in peer-to-peer (P2P) networks. More specifically, we present results that advance the state-of-the-art in the research area of distributed RDF query processing and reasoning in P2P networks. We fully design and implement a P2P system, called Atlas, for the distributed query processing and reasoning of RDF and RDFS data. Atlas is built on top of distributed hash tables (DHTs), a commonly-used case of P2P networks. Initially, we study RDFS reasoning algorithms on top of DHTs. We design and develop distributed forward and backward chaining algorithms, as well as an algorithm which works in a bottom-up fashion using the magic sets transformation technique. We study theoretically the correctness of our reasoning algorithms and prove that they are sound and complete. We also provide a comparative study of our algorithms both analytically and experimentally. In the experimental part of our study, we obtain measurements in the realistic large-scale distributed environment of PlanetLab as well as in the more controlled environment of a local cluster. Moreover, we propose algorithms for SPARQL query processing and optimization over RDF(S) databases stored on top of distributed hash tables. We fully implement and evaluate a DHT-based optimizer. The goal of the optimizer is to minimize the time for answering a query as well as the bandwidth consumed during the query evaluation. The optimization algorithms use selectivity estimates to determine the chosen query plan. Our algorithms and techniques have been extensively evaluated in a local cluster.

1 Introduction

More than just a vision nowadays, the Semantic Web has begun to be realized by the publication of large datasets according to the principles of the Linked Data initiative[1]. The Linked Data initiative aims at connecting data sources on the Web and exposing real life data using semantic technologies offering a new way of data integration and interoperability. The result of this effort is a Web

* Dissertation Advisor: Manolis Koubarakis, Professor

[1] `http://linkeddata.org/`

of Data, where URIs identify real life things, dereferencing URIs returns RDF information about those things, and this RDF information contains related URIs which are links to other resources enabling further exploration. The Linked Data community have established a set of best practices for collaboratively publishing and interlinking structured data on the Web. There are numerous sources that expose their data on the Web in the form of Linked Data ranging from community-driven efforts to governmental bodies or scientific groups. DBpedia[2], BBC music information [18], open government data[3] are only a few examples of the constantly increasing Linked Data cloud[4].

With the vast amount of available RDF data sources on the Web increasing rapidly, there is an urgent need for RDF data management. RDF storage, query processing and reasoning have been at the center of attention during the last years in the Semantic Web community and more recently in other research fields as well. Many systems have been developed for storing and querying RDF data. The first attempts were centralized approaches, such as Jena [32], Sesame [4] and RSSDB [1]. However, managing the avalanche of available RDF data has become a challenge for such RDF stores. This has necessitated the careful performance evaluation of existing RDF stores on appropriately designed benchmarks and very big data sets and the development of novel implementations based on efficient indexing techniques and relational-style statistics-based query optimization [22, 31]. Performance results published very recently indicate that state-of-the-art systems like RDF-3X [22] can execute complex join queries on RDF data sets containing close to a billion triples in a few seconds.

Although some existing RDF stores have excellent performance, they can be overwhelmed by user requests when used in wide-area network applications such as content-sharing, Web/Grid service registries, distributed digital libraries and social networks such as the ones discussed in [16, 27] More generally, since centralized RDF stores are lacking the reliability properties typically associated with large distributed systems, (e.g., fault-tolerance, load balancing, availability), researchers have also studied parallel and distributed solutions for RDF and RDFS query processing and reasoning. These include solutions based on peer-to-peer (P2P) systems, distributed computing platforms built on powerful clusters and, more recently, cloud computing platforms using the MapReduce framework [6].

In this thesis we concentrate on RDF query processing and reasoning using P2P networks. The results of this thesis have been published in major international conferences [14, 15] of the Semantic Web community, workshops [12, 17] and journals [13, 16].

[2] `http://dbpedia.org`
[3] `http://www.data.gov/`, `http://data.gov.uk/`
[4] `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`

2 Dissertation Summary

In a setting where several heterogeneous sources of data are geographically distributed, P2P systems enable the aggregation and integration of these data sources in an efficient way. P2P networks have gained much attention in the last ten years, given all the good features they can provide to Internet-scale applications. There have been several proposals of P2P architectures and amongst them distributed hash tables (DHTs) [2] are the most prominent class. DHTs allow for *full distribution*, *high-performance*, *scalability*, *resilience to failures*, *robustness* and *adaptivity* in applications such as distributed digital libraries and others we mentioned above.

An RDF repository built on top of a DHT simplifies the integration of data from many distributed heterogeneous data sources compared to other distributed approaches. Additionally, DHTs can be used to ensure *efficient* query answering from all these heterogeneous sources. DHTs have been proposed for the storage and querying of RDF data at Internet scale by several works such as [5, 11, 19]. This thesis also focuses on DHTs as the P2P architecture of choice. Thus, the algorithms of this thesis can run on commodity machines deployed all over the world, as it is the case with many other P2P applications. This is in contrast to other distributed approaches that rely on distributed platforms built on powerful clusters such as [7, 9, 23] and cloud computing platforms using MapReduce [20, 28], which typically demand high-end, locally deployed infrastructures whose cost can be very high in many cases.

When designing a DHT-based system for RDF data management, there are several challenges that have to be faced. The first one is how to distribute the data among the nodes of the network. DHTs utilize an efficient protocol for indexing data items in the network and thus, an indexing scheme for RDF data that conforms with this protocol may be adopted. Another issue that has to be faced is whether RDF data and RDFS ontologies should be handled uniformly or RDFS ontologies should be globally known by all nodes.

The adopted storage scheme will help us deal with the second challenge: answering SPARQL queries efficiently. A key aspect here is to design efficient query processing algorithms which are able to combine RDF data distributed across different nodes of the network to answer user queries. Another important need in Semantic Web applications is modeling application knowledge and reasoning about this knowledge. Therefore, in the context of RDF, we have to deal not only with a huge amount of distributed data, but also with a set of RDFS ontologies that give meaning to this data. Naturally, SPARQL queries need to be answered in a way that take into account both RDF data and RDFS ontologies, as well as the RDFS entailment rules given in [10].

Another challenging issue for RDF data management in a DHT environment is query optimization. RDF query optimization techniques in a DHT-based system have to be carefully considered given that data are distributed across all nodes of the network. Although query optimization has been extensively studied in the database area and is widely used in modern DBMSs, RDF query optimization has been addressed only recently even in centralized environments [21,22,26].

The contributions of this thesis are summarized in the following paragraphs.

In this thesis we fully design and implement a DHT-based system for the distributed query processing and reasoning of RDF and RDFS data. The indexing scheme we deploy in our system is the triple indexing algorithm originally presented in [5] where each RDF triple is indexed in the DHT three times. An important aspect of our indexing scheme is that data and schema information is handled uniformly. Although other distributed approaches such as [28, 30] assume that each node keeps all RDF schema information, we adopt a more generic approach where no global knowledge about the schema is required. In this way, our system can also handle scenarios with very big ontologies where other systems such as the above might not scale.

With respect to RDFS reasoning, our contribution is the design and development of distributed *forward* and *backward* chaining algorithms on top of a DHT. The forward chaining (FC) approach has minimal requirements during query answering, but needs a significant amount of storage for all the inferred data. In contrast, the backward chaining (BC) approach has minimal storage requirements, at the cost of an increase in query response time. There is a time-space trade-off between these two approaches, and only by knowing the query and update workload of an application, we can determine which approach would suit it better. This trade-off has never been studied in detail in a *distributed Web-scale* scenario and this is a challenge we undertake. Our backward chaining algorithm is the first distributed top-down algorithm proposed for RDFS reasoning in a decentralized environment in general. Current forward chaining approaches in various distributed architectures demonstrate a big rate of redundant information occurred from the inferred RDF triples [28, 30]. Our forward chaining algorithm (FC*) is the first one that deals with an important case of generating redundant RDF information. In addition, we present an algorithm (MS) which works in a bottom-up fashion using the magic sets transformation technique [3], a technique that has not been studied in the literature for distributed RDFS reasoning. We study theoretically the correctness of our reasoning algorithms and prove that they are sound and complete. We also provide a comparative study of our algorithms both analytically and experimentally. In the experimental part of our study, we obtain measurements in the realistic large-scale distributed environment of PlanetLab as well as in the more controlled environment of a local cluster.

We propose a query processing algorithm adapted to use a query graph model to represent SPARQL queries in order to avoid the computation of Cartesian products. In addition, as URIs and literals may consist of long strings that are transferred in the network and processed locally at the nodes, we show how to benefit from a mapping dictionary to further enhance the efficiency of the query processing algorithm. Although mapping dictionaries are by now standard in centralized RDF stores, our work is the first that discusses how to implement one in a DHT environment. Our experiments conducted in both PlanetLab and a local cluster showcase the importance of having a distributed mapping dictionary in our system.

In the context of query optimization, we fully implement and evaluate a DHT-based optimizer. The goal of the optimizer is to minimize the time for answering a query as well as the bandwidth consumed during the query evaluation. We propose three greedy optimization algorithms for this purpose: two static, namely NA and SNA, and one dynamic, namely DA. The static query optimization is completely executed before the query evaluation begins, while the dynamic query optimization take places during the query evaluation creating query plans incrementally. These algorithms use selectivity estimates to determine the chosen query plan. We propose methods for estimating the selectivity of RDF queries utilizing techniques from relational databases. We discuss which statistics should be kept at each network node and use histograms for summarizing data distributions. We demonstrate that it is sufficient for a node to create and maintain local statistics, i.e., statistics for its locally stored data. These local statistics are in fact global statistics needed by the optimization algorithms and can be obtained by other nodes by sending low cost messages. This is a very good property of the indexing scheme we adopt from [5] that has not been pointed out in the literature before.

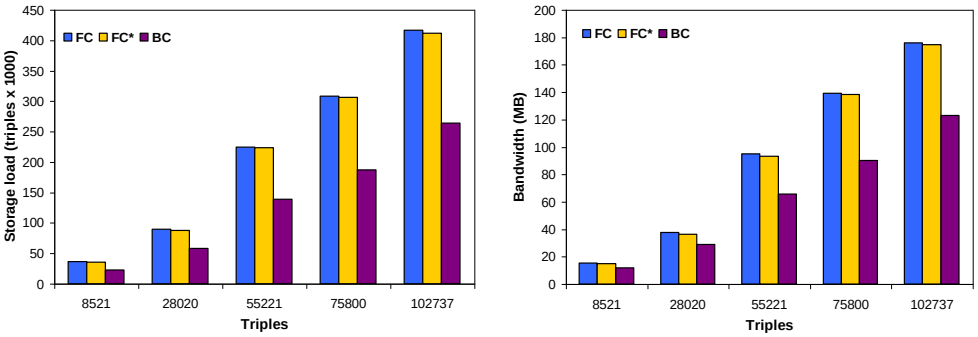
Using the above techniques, we have implemented a P2P system, called Atlas, for distributed query processing and reasoning of RDF and RDFS data. Atlas is publicly available as open source under the LGPL license[5]. Although our proposed algorithms and techniques have been implemented in Atlas using the Bamboo DHT [24], they are DHT-agnostic; they can be implemented on top of any DHT.

3 Results and Discussion

In this section, we present a brief experimental evaluation of our reasoning algorithms and optimization techniques. All algorithms have been implemented as an extension to our prototype system Atlas. In the latest version of Atlas, we have adopted SQLite as the local database of each peer since the Berkeley DB included in the Bamboo implementation was inefficient. In our algorithms, we have also utilized the dictionary encoding implemented in Atlas, where URIs and literals are mapped to integer identifiers. For our experiments, we used as a testbed both the PlanetLab network as well as a local shared cluster (`http://www.grid.tuc.gr/`). Although we have extensively tested our techniques on both testbeds, here we present results only from the cluster where we achieve much better performance. The cluster consists of 41 computing nodes, each one being a server blade machine with two processors at 2.6GHz and 4GB memory. We used 30 of these machines where we run up to 4 peers per machine, i.e., 120 peers in total.

For our evaluation, we use the Lehigh University benchmark (LUBM) [8] that provides synthetic RDF datasets of arbitrary sizes and 14 SPARQL queries. LUBM benchmark consists of a university domain ontology modeling an academic setting and is widely used for testing RDF stores. Each dataset can be defined by the number of universities generated. For example, the dataset LUBM-1

[5] `http://atlas.di.uoa.gr`



(a) Message storage load

(b) Bandwidth

Fig. 1. Storing LUBM-1

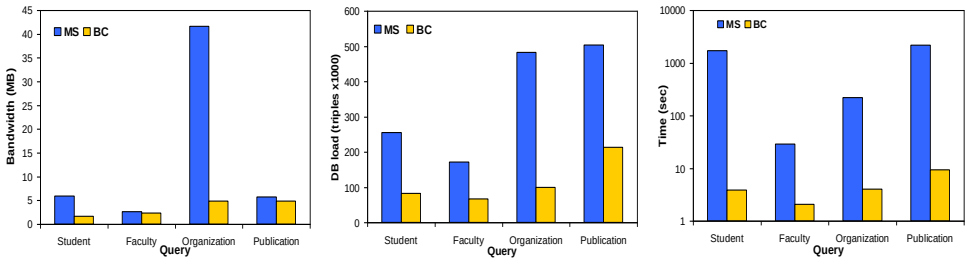
involves one university, while the dataset LUBM-10 involves 10 universities. The more universities are generated the more triples are produced. Since answering a query in the LUBM benchmark might involve RDFS reasoning, All measurements are averaged over 10 runs using the geometric mean which is more resilient to outliers.

3.1 Comparing the reasoning algorithms

In this section, we compare the performance of the forward chaining algorithm (FC) with the backward chaining algorithm (BC) when storing RDF(S) data in the network. Alongside the results of the forward chaining algorithm we have presented in [15], we also present results from the forward chaining algorithm which generates less redundant information (FC*) and the algorithm that uses the magic sets transformation (MS).

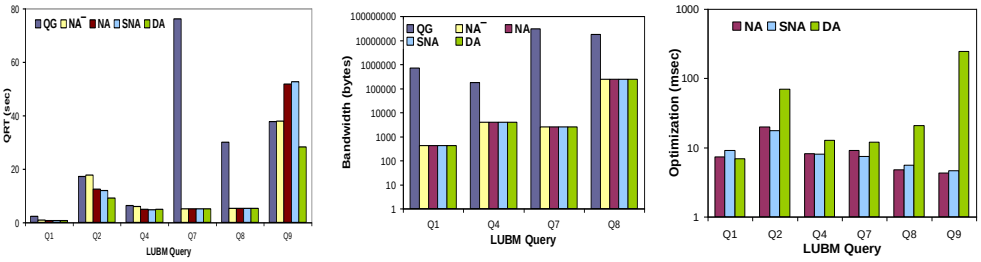
For this set of experiments, we stored an increasing number of triples from the LUBM-1 dataset which consists of $102,737$ triples. We compare the behaviour of BC, FC and FC*. Figure 1 shows results regarding the store message load and the bandwidth consumption. The x-axis shows the number of triples initially inserted in the network. The results of this experiment are similar regarding the comparison of BC with FC and FC*. However, in this case, FC and FC* produce almost the same number of messages resulting in similar bandwidth consumption and cause almost the same load in the network, with FC* performance slightly better. The difference between FC and FC* in this case is not so evident because of the nature of the LUBM schema. LUBM schema includes only small RDFS class hierarchies whose depth is at most 2. Therefore, the number of redundant triples generated from FC is very small.

In the next experiment, we use the LUBM dataset whose schema contains several independent class hierarchies. We created a network of 156 nodes in the cluster and stored the complete LUBM-20 dataset consisting of $2,782,435$ triples. We want to retrieve the instances of the following classes of the LUBM



(a) Bandwidth consumption (b) DB load (c) Time

Fig. 2. MS vs. BC for LUBM-20



(a) Total query response time (b) Total bandwidth usage (c) Optimization time

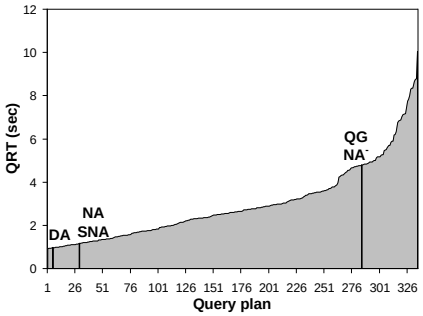
Fig. 3. Query optimization performance for LUBM-50

schema: `ub:Student`, `ub:Faculty`, `ub:Organization`, `ub:Publication`, where `ub` is the appropriate namespace. For MS, we sent a request with the predicate `m_type` and argument the class name, while for BC we run the query that asks for all instances of the respective class. Figure 2 shows the bandwidth consumption, the total DB load and the time required for each algorithm to terminate. In this experiment as well, we observe that BC outperforms MS. Since we deal with a bigger dataset the advantage of using BC is more evident, as shown in Figure 2(c).

3.2 Comparing the optimization algorithms

In the following, QG denotes that the query graph is used to avoid Cartesian products but no other optimization is utilized. The naive algorithm using the bound-is-easier heuristic is denoted by NA^-, while the naive and semi-naive algorithm using the analytical estimation is denoted by NA and SNA, respectively. Finally, DA denotes the dynamic optimization algorithm. Details about these algorithms can be found in [14]. In this section, we compare and evaluate the optimization algorithms. For this set of experiments, we store all the inferred triples of the LUBM-50 dataset ($9,437,221$ triples) in a network of 120 peers. Then, using each optimization algorithm, we run the queries.

In all graphs of Fig. 3, the x-axis shows the LUBM queries while the y-axis depicts the metric of interest. Figure 3(a) shows the query response time (QRT)



(a) LUBM Q2 query plan space

LUBM Query	Min QRT	Max QRT	DA QRT
Q2	0.91 s	10.06 s	0.96 s
Q4	2.54 s	17.39 s	2.89 s
Q7	1.53 s	16.62 s	1.55 s
Q8	2.88 s	10.20 s	3.06 s
Q9	3.30 s	64.06 s	4.41 s

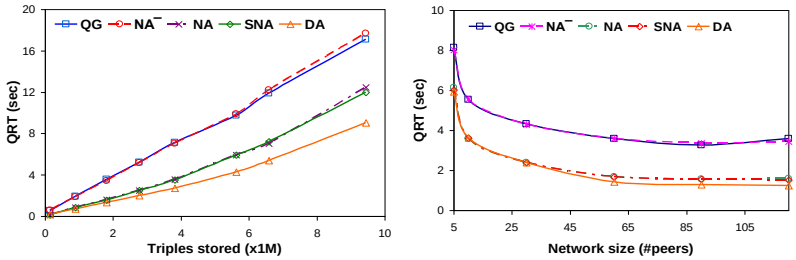
(b) QRT of LUBM query plans

Fig. 4. Exploring the query plan space for LUBM-10

for the different LUBM queries. QRT is the total time required to answer a query and it also includes the time required by the query optimizer for determining a query plan (optimization time). Figure 3(b) shows the total bandwidth consumed during query evaluation.

Queries Q2 and Q9 consist of 6 triple patterns having only their predicates bound. In both queries, there exists a join among the last three triple patterns (in the order given by the benchmark) and the combination of all three triple patterns is the one that yields a small result set. DA finds a query plan that combines these three triple patterns earlier than the other algorithms. This results in producing smaller intermediate result sets, as it is also shown by the bandwidth consumption in Fig. 3(b), and thus results in better QRT. Although NA and SNA perform close to DA for query Q2, they fail to choose a good query plan for Q9 affecting both the QRT and the bandwidth consumption. At this point, we should note that QG and NA^- depend on the initial order of a query's triple patterns. For this reason, both algorithms choose a relatively good query plan for query Q9 since the order in which its triple patterns are given by the benchmark is a good one. Q4 is a star-shape query with all its triple patterns sharing the same subject variable, while only the first two triple patterns have two bound components. Therefore, since these two triple patterns are the more selective ones, all optimization algorithms choose the same query plan and perform identically in terms of both QRT and bandwidth. The same holds for query Q7 where QRT is significantly reduced when using either optimization technique compared to QG. Q8 is a query similar to Q7.

In Fig. 3(c), we show the total optimization time in msec on a logarithmic scale. For QG and NA^- the optimization overhead is negligible and is not shown in the graph. The optimization time contains the time for retrieving the required statistics from the network, the time for the selectivity estimation and the time spent by the optimization algorithm. As expected, DA spends more time than the other optimization algorithms since it runs at each query processing step. However, the optimization time is still one order of magnitude smaller than the time required by the query evaluation process. Therefore, although DA requires more time than the other optimization algorithms, the system manages to per-



(a) Varying dataset size for query Q2 (b) Varying network size for query Q2

Fig. 5. Varying dataset and network size

form efficiently for all queries when DA is used. We observe similar results for the bandwidth consumed by the query optimizer. NA and SNA consume $\sim 2KB$ while DA consumes $\sim 7KB$, still one order of magnitude less than the bandwidth spent during query evaluation. We omit this graph due to space limitations.

3.3 Effectiveness of query optimization

In this section, we explore the query plan space of the LUBM queries to show how effective the optimization algorithms are. The size of the query plan space of a query consisting of N triple patterns is $N!$. Since query plans that involve Cartesian products are very inefficient to evaluate in a distributed environment, we consider only triple pattern permutations which do not produce any Cartesian product. In this experiment, we store the LUBM-10 dataset in a network of 120 peers and run all possible query plans for several LUBM queries. In Fig. 4(a), we depict the QRT of all possible query plans for query Q2 in ascending order. The query plan space of Q2 consists of 335 query plans which do not involve any Cartesian product. In this figure, we highlight the position of the query plans chosen by the different optimization algorithms. We observe that DA chooses one of the best query plans, while NA and SNA perform worse choosing the 27th best query plan. NA^- performs poorly choosing one the worst query plans. Similar results are observed for the other queries as well. In Fig. 4(b), we list the QRT for all queries of the best and the worst query plan together with the QRT when using DA. We observe that the QRT when using DA is very close to the QRT of the optimal query plan for all queries. Note that without the query plans that involve Cartesian products, the difference between the min and the max QRT of all queries is not very large.

3.4 Varying the dataset and network size

In these sets of experiments, we study the performance of our system when varying the number of triples stored in the network and the number of peers. We show results only for Q2 which involves a join among three triple patterns.

Figure 5(a) shows the behavior of our system using each optimization algorithm as the dataset stored in the network grows. In a network of 120 peers, we stored datasets from LUBM-1 to LUBM-50. Every time we measured the QRT of query Q2 using each optimization algorithm. As expected, QRT increases as the number of triples stored in the network grows. This is caused by two factors.

Firstly, the local database of each peer grows and as a result local query processing becomes more time-consuming. Secondly, the result set of query Q2 varies as the dataset changes. For example, for LUBM-1 the result set is empty, while for LUBM-50 the result set contains 130 answers. This results in transferring larger intermediate result sets through the network which also affects the QRT of the query. Besides, this experiment brings forth an interesting conclusion regarding the optimization techniques. While query plans chosen by NA, SNA and DA perform similarly up to approximately $1.8M$ triples stored (i.e., LUBM-10), we observe that for bigger datasets the query plan chosen by DA outperforms the others. This shows that the system becomes more scalable with respect to the number of triples stored in the network when using DA. Similar results are observed for Q9, while for the rest queries all optimization algorithms choose the same query plan independently of the dataset size.

In the next set of experiments, we start networks of 5, 10, 30, 60, 90 and 120 peers and store the LUBM-10 dataset. We then run the queries using all optimization techniques. In Fig. 5(b), we show the QRT for Q2 as the network size increases. We observe that QRT improves significantly as the network size grows up to 60, while it remains almost the same for bigger network sizes. The decrease in the QRT for small networks is caused by the fact that the more peers join the network the less triples are stored in each peer's database and thus local processing load is reduced. The same result was observed in other queries where QRT either improved or remained unaffected as the number of peers increased.

3.5 Discussion

We have also experimented with different datasets using the SP^{2}B benchmark [25] as well as a real world dataset of the US Congress vote results presented in [29]. The results were similar to the ones observed using LUBM. For all datasets, DA consistently chooses a query plan close to the optimal regardless of the query type or dataset stored and without posing a significant overhead neither to the total time for answering the query nor to the bandwidth consumed. On the contrary, the static optimization methods are dependent on the type of the query and the dataset, which make them unsuitable in various cases (as shown earlier for query Q9). In addition, we have also tested indexing all possible combinations of the triples' components, as proposed in [19]. In this case, we have used histograms at each peer for combinations of triples' components as well. However, we did not observe any difference in the choice of the query plan and thus, showed results only with the triple indexing algorithm. This results from the nature of the LUBM queries which mostly involve bound predicates and object-classes for which we kept an exact distribution in both cases.

4 Conclusions

We presented a system that is able to support full-fledged management of RDF data in a large-scale decentralized environment. In particular, we fully designed

and implemented a P2P system, called Atlas, for the distributed query processing and reasoning of RDF and RDFS data. One of the main focus of this thesis was to enable Atlas to support RDFS reasoning. Two well-known reasoning techniques we studied are *forward chaining* and *backward chaining*. Our work was the first that investigated the trade-off of the two algorithms in detail in a *distributed Web-scale* scenario. Our forward chaining algorithm was the first one that dealt with an important case of generation of redundant RDF information, while our backward chaining algorithm was the first distributed backward chaining algorithm proposed for RDFS reasoning in a decentralized environment in general. Moreover, a magic sets transformation technique for distributed RDFS reasoning has not been studied in the literature before.

Finally, this thesis addressed the problem of query optimization over RDF data stored on top of DHTs and fully implemented and evaluated a DHT-based optimizer. Our optimization algorithms ranged from static to dynamic ones and focused on minimizing the size of the intermediate results computed during query evaluation. In this way, we achieved to decrease the time required for answering a query and the bandwidth consumed during the query evaluation. Our algorithms utilized selectivity estimates to determine a query plan that minimizes the size of the intermediate results. We also proposed methods for estimating the selectivity of RDF queries utilizing techniques from relational databases. We defined which statistics are required at each node of the network for the computation of the selectivity estimates and used histograms for summarizing the distribution of these statistics.

References

1. S. Alexaki, V. Christophides, G. Karvounarakis, and D. Plexousakis. On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs. In *Proceedings of the 4rth International Workshop on the Web and Databases (WebDB 2001, co-located with SIGMOD 2001)*, Santa Barbara, California, USA, 2001.
2. H. Balakrishnan, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Looking up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, 2003.
3. C. Beeri and R. Ramakrishnan. On the Power of Magic. In *PODS '87: Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 269–284, New York, NY, USA, 1987. ACM.
4. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 1st International Semantic Web Conference (ISWC 2002)*, Sardinia, Italy, 2002.
5. M. Cai and M. Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In *Proceedings of the 13th World Wide Web Conference (WWW 2004)*, New York, USA, 2004.
6. J. Dean and S. Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. In *Proceedings of the USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, pages 137–147, 2004.
7. O. Erling and I. Mikhailov. Towards Web Scale RDF. In *Proceeindgs of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS2008)*, Karlsruhe, Germany, October 2008.
8. Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.
9. A. Harth, J. Umbrich, A. Hogan, and S. Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In *Proceedings of the 6th International Semantic Web Conference (ISWC/ASWC 2007)*, pages 211–224, Busan, South Korea, 2007.
10. P. Hayes. RDF Semantics. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-mt/`.

11. F. Heine, M. Hovestadt, and O. Kao. Processing Complex RDF Queries over P2P Networks. In *Proceedings of Workshop on Information Retrieval in Peer-to-Peer-Networks (P2PIR 2005)*, Bremen, Germany, 2005.
12. Z. Kaoudi, M. Koubarakis, K. Kyzirakos, M. Magiridou, I. Miliaraki, and A. Papadakis-Pesaresi. Publishing, Discovering and Updating Semantic Grid Resources using DHTs. In *CoreGRID Workshop on Grid Programming Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments*, Heraklion, Crete, Greece, June 2007.
13. Z. Kaoudi, M. Koubarakis, K. Kyzirakos, I. Miliaraki, M. Magiridou, and A. Papadakis-Pesaresi. Atlas: Storing, Updating and Querying RDF(S) Data on Top of DHTs. *Journal of Web Semantics*, 2010.
14. Z. Kaoudi, K. Kyzirakos, and M. Koubarakis. SPARQL Query Optimization on Top of DHTs. In *Proceedings of the 9th International Conference on The Semantic Web (ISWC 2010)*, Shanghai, China, 2010.
15. Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS Reasoning and Query Answering on Top of DHTs. In *Proceedings of the 7th International Conference on The Semantic Web (ISWC 2008)*, Karlsruhe, Germany, 2008.
16. Z. Kaoudi, I. Miliaraki, M. Magiridou, E. Liarou, S. Idreos, and M. Koubarakis. Semantic Grid Resource Discovery in Atlas. *Knowledge and Data Management in Grids*, 2006. Talia Domenico and Bilas Angelos and Dikaiakos Marios D. (editors), Springer.
17. Z. Kaoudi, I. Miliaraki, M. Magiridou, A. Papadakis-Pesaresi, and M. Koubarakis. Storing and Querying RDF Data in Atlas. 3rd European Semantic Web Conference, 11 - 14 June 2006, Budva, Montenegro, Demo paper, 2006.
18. G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. Media Meets Semantic Web - How the BBC Uses DBpedia and Linked Data to Make Connections. In *Proceedings of the 6th European Semantic Web Conference (ESWC)*, Heraklion, Crete, Greece, 2009.
19. E. Liarou, S. Idreos, and M. Koubarakis. Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In *Proceedings of 5th the International Semantic Web Conference (ISWC 2006)*, Athens, GA, USA, November 2006.
20. P. Mika and G. Tummarello. Web Semantics in the Clouds. *IEEE Intelligent Systems*, 23(5):82–87, 2008.
21. T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. In *Proceedings of 34th International Conference on Very Large Data Bases (VLDB 2008)*, volume 1, pages 647–659, Auckland, New Zealand, 2008.
22. T. Neumann and G. Weikum. Scalable Join Processing on Very Large RDF Graphs. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 627–640, Providence, Rhode Island, USA, 2009.
23. E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, and F. van Harmelen. MaRVIN: A platform for large-scale analysis of Semantic Web data. In *Proceedings of Web Science Conference*, 2009.
24. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *USENIX Annual Technical Conference*, 2004.
25. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^2Bench: A SPARQL Performance Benchmark. In *Proceedings of the 29th International Conference on Data Engineering (ICDE2009)*, Shanghai, China, 2009.
26. M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL Basic Graph Pattern Optimization using Selectivity Estimation. In *Proceedings of the 17th International World Wide Web Conference (WWW 2008)*, Beijing, China, 2008.
27. C. Tryfonopoulos, S. Idreos, and M. Koubarakis. LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. In *ECDL*, pages 25–36, 2005.
28. J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable Distributed Reasoning using MapReduce. In *Proceedings of the 8th International Semantic Web Conference (ISWC2009)*, October 2009.
29. M.-E. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres. Efficiently Joining Group Patterns in SPARQL Queries. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010)*, pages 228–242, Heraklion, Greece, 2010.
30. J. Weaver and J. Hendler. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *8th International Semantic Web Conference (ISWC2009)*, October 2009.
31. C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. In *Proceedings of 34th International Conference on Very Large Data Bases (VLDB 2008)*, volume 1, pages 647–659, Auckland, New Zealand, 2008.
32. K. Wilkinson, C. Sayers, H. A. Kuno, and D. Raynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB 2003, co-located with VLDB 2003)*, Berlin, Germany, 2003.