PIWD: A Plugin-based Framework for Well-Designed SPARQL (Extended Abstract)

Xiaowang Zhang[1,3,4], Zhenyu Song[1,3], Zhiyong Feng[2,3], and Xin Wang[1,3]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
[2] School of Computer Software, Tianjin University, Tianjin 300350, China
[3] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
[4] Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing 211189, China

Abstract. In the real world datasets (e.g.,DBpedia query log), queries built on well-designed patterns containing only AND and OPT operators (for short, WDAO-patterns) account for a large proportion among all SPARQL queries. In this paper, we present a plugin-based framework for all SELECT queries built on WDAO-patterns, named PIWD. Theoretically, we can reduce the query evaluation of WDAO-patterns to subgraph homomorphism as well BGP since the query evaluation of BGP is equivalent to subgraph homomorphism. Furthermore, our preliminary experiments on gStore and RDF-3X show that PIWD can answer all queries built on WDAO-patterns effectively and efficiently.

1 Introduction

Resource Description Framework (RDF) is the standard data model in the semantic web. The standard query language for RDF graphs is SPARQL. Though SPARQL is powerful to expressing queries over RDF graphs, generally, the query evaluation of the full SPARQL is PSPACE-complete [5].

Currently, there are some popular query engines for supporting the full SPARQL such as Jena [1]. However, they become not highly efficient when they handle some large RDF datasets[7]. Currently, gStore[7] and RDF-3X[3] can highly efficiently query large datasets. But gStore and RDF-3X merely provide querying services of BGP. Therefore, it is necessary to develop a query engine with supporting more expressive queries for large datasets.

Since the OPT operator is the least conventional operator among SPARQL operators [6], it is interesting to investigate those patterns extending BGP with the OPT operator. In fact, we investigate that queries bulit on well-designed patterns are very popular in a real world. For example, in LSQ, a Linked Dataset describing SPARQL queries extracted from the logs of four prominent public SPARQL endpoints containing more than one million available queries, queries built on well-designed patterns are over 70% [2].

Furthermore, queries with well-designed AND-OPT patterns (for short, WDAO-patterns) are over 99% among those queries with well-designed patterns in LSQ [2]. In a short, the fragment of WDAO-patterns are a natural extension of BGP.

In this paper, we present a plugin-based framework for all SELECT queries built on WDAO-patterns, named PIWD. Within this framework, we can employ any query engine evaluating BGP for evaluate queries bulit on WDAO-patterns in a convenient way.

2 Preliminaries

Well-Designed Patterns A UNION-free pattern P is well-designed if the followings hold:

- P is safe, that is, each subpattern of the form Q FILTER C of P holds the condition: $var(C) \subseteq var(Q)$.
- for every subpattern $P' = (P_1 \text{ OPT } P_2)$ of P and for every variable $?x$ occurring in P, the following condition hold: If $?x$ occurs both inside P_2 and outside P', then it also occurs in P_1.

For instance, the pattern Q is a well-designed pattern.

Note that the OPT operation provides really optional left-outer join due to the weak monotonicity [5].

3 Well-Designed And-Opt Tree

Definition 1 (WDAO-tree). *Let P be a well-designed pattern in OPT normal form. A well-designed tree T based on P is a redesigned parse tree, which can be defined as follows:*

- *All inner nodes in T are labeled by OPT operators and leaf nodes are labeled by BGP.*
- *For each subpattern $(P_1 \text{ OPT } P_2)$ of P, the well-designed tree T_1 of P_1 and the well-designed tree T_2 of P_2 have the same parent node.*

For instance, consider a WDAO-pattern $P^5 = (((p_1 \text{ AND } p_3) \text{ OPT}_2\ p_2) \text{ OPT}_1 ((p_4 \text{ OPT}_4\ p_5) \text{ OPT}_5 (p_6 \text{ OPT}_6\ p_7)))$.

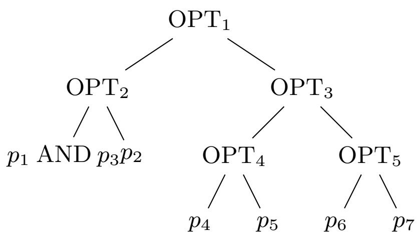
The WDAO-tree T is shown in Figure 1(a). As shown in this example, BGP - $(p_1 \text{ AND } p_3)$ is the exact matching in P, which corresponds to the non-optional pattern. Besides, in WDAO-tree, it is the leftmost leaf in T.

4 PIWD Demonstration

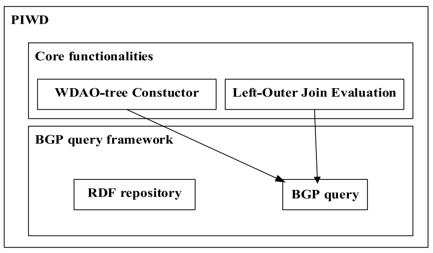
PIWD is written in Java in a 2-tier design shown in Figure 1(b). The bottom layer consists of any BGP query framework which is used as a black box for evaluating BGPs. Before answering SPARQL queries, the second layer provides the rewriting process and left-outer join evaluation, which lead to the solutions.

BGP query framework supports both query and RDF data management, such as gStore, RDF-3X and so on, which solve the problem of subgraph isomorphism.

[5] We give each OPT operator a subscript to differentiate them so that readers understand clearly.



(a) WDAO-tree



(b) PIWD architecture

Fig. 1. PIWD Overview

PIWD provides the left-outer join between the BGPs. That is, the problem of answering well-designed SPARQL has been transformed into the problem of subgraph isomorphism and left-outer join between triple patterns.

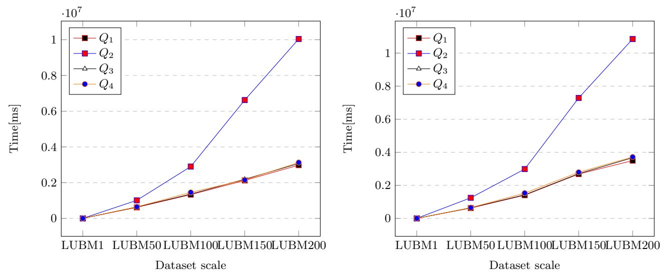
5 Experiments and Evaluations

Experiments All experiments were carried out on a machine running Linux, which has one CPU with four cores of 2.40GHz, 32GB memory and 500GB disk storage. All of the algorithms were implemented in JAVA. gStore[7] and RDF-3X[3] are used as the underlying query engines to handle BGPs. In our experiments, there is no optimization in our OPT operation. In our experiments, we used LUBM1, LUBM50, LUBM100, LUBM150 and LUBM200 as our query datasets. The queries over LUBM were designed as four different forms, which corresponds to different WDAO-trees. OPT nesting in Q_2 is the most complex among four forms. Furthermore, we built AND operators in each query.

Evaluation on PIWD The variation tendencies of query response time are shown in Figure 2. Query efficiency is decreased with higher response time when OPT nesting becomes more complex. Furthermore, there has been a significant increase in query response time when the dataset scale grows up. For instance, we observe Q_2, which corresponds to the most complex pattern in our four experimental SPARQL patterns. When the dataset is ranging from LUBM100 to LUBM200, its query response time extends more than five times even though the dataset scale extends two times.

6 Conclusion

In this paper, we have presented PIWD, which is a plugin adaptable for any BGP query framework to handle WDAO-patterns. Theoretically, PIWD rebuilds the query plan based on WDAO-tree. After employing BGP query framework on WDAO-tree, PIWD supports left-outer join operation between triple patterns. Our experiments show that PIWD can deal with complex and multi-level nested



(a) Performance on gStore (b) Performance on RDF-3X

Fig. 2. Query Response Time over LUBM

WDAO-patterns. In the future, we will further handle other non-well-designed patterns and deal with more operations such as UNION. Besides, we will consider OPT operation optimization to improve efficiency of PIWD and implement our framework on distributed RDF graphs by applying the distributed gStore [4].

Acknowledgments

This work is supported by the program of the program of the National Key Research and Development Program of China (2016YFB1000603), the National Natural Science Foundation of China (NSFC) (61502336), and the open funding project of Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education (K93-9-2016-05). Xiaowang Zhang is supported by Tianjin Thousand Young Talents Program and the project-sponsored by School of Computer Science and Technology in Tianjin University.

References

1. Carroll J.J., Dickinson I., Dollin C., Reynolds D., Seaborne A., and Wilkinson K. (2004). Jena: Implementing the semantic web recommendations. In: *Proc. of WWW 2004*, pages 74–83.
2. Han X., Feng Z., Zhang X., Wang X., and Rao G. (2016). On the statistical analysis of practical SPARQL queries. In: *Proc. of WebDB 2016*, article 2.
3. Neumann T. and Weikum G. (2010). The RDF3X engine for scalable management of RDF data. *VLDB Journal*, 19(1):91–113.
4. Peng P., Zou L., Özsu M.T., Chen L., and Zhao D. (2016). Processing SPARQL queries over distributed RDF graphs. *VLDB J.*, 25(2): 243–268.
5. Pérez J., Arenas M., and Gutierrez C. (2009). Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):30–43.
6. Zhang X. and Van den Bussche J. (2014). On the primitivity of operators in SPARQL. *Inf. Process. Lett.*, 114(9):480-485.
7. Zou L., Özsu M.T., Chen L., Shen X., Huang R., and Zhao D. (2014). gStore: A graph-based sparql query engine. *VLDB J.*, 23(4):565–590.