BatWAn: A Binary and Multi-Way Query Plan Analyzer

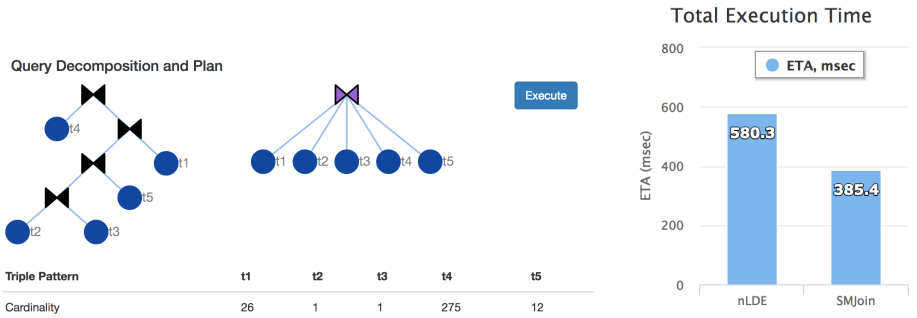
Mikhail Galkin[1,2,3], Maria-Esther Vidal[2]

[1] Smart Data Analytics (SDA), University of Bonn
{galkin|vidal}@cs.uni-bonn.de
[2] Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
[3] ITMO University, Saint Petersburg, Russia

Abstract. The majority of existing SPARQL query engines generate query plans composed of binary join operators. Albeit effective, binary joins can drastically impact on the performance of query processing whenever source answers need to be passed through multiple operators in a query plan. Multi-way joins have been proposed to overcome this problem; they are able to propagate and generate results in a single step during query execution. We demonstrate the benefits of query plans with multi-way operators with BatWAn, a binary and multi-way query plan analyzer. Attendees will observe the behavior of multi-way joins on queries of different selectivity, as well as the impact on total execution time, time for the first answer, and continuous results yield over time. The demo is available at: http://ec2-34-212-221-44.us-west-2.compute.amazonaws.com:9000/.

1 Introduction

SPARQL query engines rely on join operators to merge intermediate results collected from the evaluation of two subqueries of a given query. Query engines seek to construct query plans that consist of join operators to optimize execution time, network exchange, and tackle network delays. Despite classic examples, e.g., symmetric hash join [3], nested loop join [3], and XJoin [5], query engines usually implement their own join operators to enhance the engine performance, e.g., ANAPSID [2] utilizes the *agjoin* while FedX employs the *controlled bound worker join* (CBJ). Moreover, query engines are able to combine various join operators to achieve (at their estimation) the most effective and efficient query plan. For instance, nLDE [1] builds a query plan composed of both classic and ANAPSID operators. However, all the mentioned operators are binary, i.e., they accept two arguments (intermediate results of two subqueries). On the other hand, multi-way join operators, e.g., *SMJoin* [4], accept more than two arguments as an input. An arbitrary operator arity provides several trade-offs that affect total query execution time, time until the first answer, and continuous result yield. Those trade-offs of employing a multi-way join operator instead of a binary join are compared in BatWAn, a binary and multi-way query plan analyzer. BatWAn is designed to shed light on the characteristics of the SPARQL queries where multi-way query plans outperform binary join plans. Attendees will be able to select different queries and observe how the shape of the query plans



(a) Query plan (b) Query execution result

Fig. 1: **High Selective Query.**

and their operators affect the values of experimental metrics like total execution time, time for the first answer, and continuous generation of answers over the time. The demo includes a set of queries to be used in the analysis of the performance of the binary and multi-way join operators.

2 The BatWAn Architecture

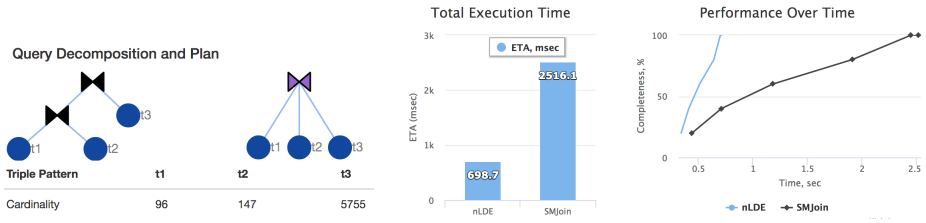
BatWAn accepts a SPARQL query with an assumption that every triple pattern is answered by one specific source. BatWAn pushes the query to two independent query planners, i.e., binary plans are produced by nLDE [1], whereas multi-way plans are built with the extension of ANAPSID tailored to employ SMJoin [4].

```
select * where {
?d1 dct:subject dbr:Category:Hereditary_cancers .
?d1 foaf:isPrimaryTopicOf wikipedia_en:Cowden_syndrome .
?d1 dbo:wikiPageExternalLink
    <http://www.cancer.net/cancer-types/cowden-syndrome> .
?d1 dct:subject dbr:Category:Epidermal_nevi,_neoplasms,_cysts .
?d1 dct:subject
    dbr:Category:Deficiencies_of_intracellular_signaling_peptides_and_proteins }
```

Listing 1.1: High Selective Query

The resulting plan is visualized using *Cytoscape.js*[4] library. Additionally, BatWAn shows an estimated number of intermediate results, i.e., answers of each triple pattern. Therefore, BatWAn categorizes input queries as high selective (where each triple pattern returns < 1000 intermediate results) and low selective (where triple pattern cardinality is not limited). An example of a high selective query is presented in Listing 1.1. The query is composed of five triple patterns. Binary and multi-way plans for the high selective query are illustrated in Fig. 1a; the maximal cardinality of query triple patterns does not exceed 275

[4] http://js.cytoscape.org/



(a) Query plan (b) Query execution result

Fig. 2: **Low Selective Query.**

intermediate results. Having obtained the plan, BatWAn depicts query execution results using *Highcharts.js*[5] library against a DBpedia 2015 HDT endpoint hosted by a *Triple Pattern Fragments* JS server deployed on a Ubuntu 16.04 (64 bits) Dell PowerEdge R805 server, AMD Opteron 2.4 GHz CPU, 64 cores, 256 GB RAM. Total execution time of nLDE and SMjoin are reported as in Fig. 1b.

```
select * where {
?d1 dct:subject dbr:Category:Anticonvulsants .
?d1 dct:subject dbr:Category:World_Health_Organization_essential_medicines .
?d1 rdf:type    wikidata:Q8386 }
```

Listing 1.2: Low Selective Query

Furthermore, BatWAn visualizes values of metrics that capture the performance of low selective queries, i.e., queries with triple patterns that return > 1000 intermediate results. An example of a low selective query is presented in Listing 1.2. As in previous example, BatWAn visualizes the binary and multi-way plans (Fig. 2a). Furthermore, the comparison between the total execution time of nLDE and SMJoin and the continuous behavior of these plans over the time are illustrated (Fig. 2b).

3 Demonstration of Use Cases

We motivate our work by observing how binary and multi-way plans exhibit different behavior on high- and low-selective queries. Thus, our ambition is to devise use cases that allow for the understanding of both the characteristics of these queries and their corresponding plans, and the behavior exhibited by the evaluated query engines. We will demonstrate the following use cases:
Effects on Total Query Execution time. We will show that in queries as the ones presented in Listing 1.1 and Listing 1.2, binary and multi-way plans exhibit opposite behaviors in terms of total execution time. In high-selective queries composed of very selective triple patterns, multi-way plans are able to independently merge results from different triple patterns. Contrary, in binary plans, intermediate results are passed through multiple operators in a query plan;

[5] https://www.highcharts.com/

thus, the query engine needs to wait until all the tuples arrive and joins are performed. This negatively impacts on the execution time as observed in Fig. 1b. On the other hand, in low selective queries, binary plans are built in a way, that nested loop joins can be used to produce instantiations that drastically reduce the amount of intermediate results and speed up execution time. In contrast, SMJoin works under the assumption of zero knowledge about the cardinality of the triple pattern fragments, and evaluates every low selective triple pattern independently; thus, it requires a long time to collect all the results.

Effects on First Produced Result time. The attendees will observe the impact and benefits of producing results incrementally. We will demonstrate that independently of the selectivity of the queries, i.e., low and high, both types of plans produce the first result almost at the same time.

Effects on Continuous results yield. The goal of this use case is to show the effects and benefits of the continuous behavior of a query engine. Attendees will be able to visualize the number of results produced over a period of time. More importantly, they will observe that independently of the number of intermediate results generated during the execution of binary and multi-join plans, query answers are produced at almost the same rates. However, because binary plans are able to utilize nested loop joins and drastically reduce the number of intermediate results, binary plans are faster than multi-join plans over time.

4 Conclusions

BatWAn provides a visual comparison how join operators and query plans affect the performance of a SPARQL query engine. Particularly, BatWAn shows a trade-off between binary and multi-way join operators in three use cases. Attendees will be able to explore all the use cases, understand why query plans play an important role in a query engine performance, and observe how the continuous behavior of a query engine allows for overcoming adverse query execution conditions like a large number of intermediate results or sub-optimal query plans.

References

1. M. Acosta and M. Vidal. Networks of linked data eddies: An adaptive web query processing engine for RDF data. In *14th ISWC, USA*, pages 111–127, 2015.
2. M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *ISWC*, pages 18–34, 2011.
3. A. Deshpande, Z. G. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
4. M. Galkin, K. Endris, M. Acosta, D. Collarana, M. Vidal, and S. Auer. Smjoin: A multi-way join operator for sparql queries. In *13th SEMANTICS*, 2017, Accepted for publication.
5. T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, 23(2):27–33, 2000.