Tracking Federated Queries in the Linked Data

Georges Nassopoulos, Patricia Serrano-Alvarado, Pascal Molli, Emmanuel Desmontils

Tracking Federated Queries in the Linked Data

Georges Nassopoulos, Patricia Serrano-Alvarado,
Pascal Molli, Emmanuel Desmontils

LINA Laboratory, Université de Nantes – France
firstName.lastName@univ-nantes.fr

Abstract. Federated query engines allow data consumers to execute queries over the federation of Linked Data (LD). However, as federated queries are decomposed into potentially thousands of subqueries distributed among SPARQL endpoints, data providers do not know federated queries, they only know subqueries they process. Consequently, unlike warehousing approaches, LD data providers have no access to secondary data. In this paper, we propose FETA (FEderated query TrAcking), a query tracking algorithm that infers Basic Graph Patterns (BGPs) processed by a federation from a shared log maintained by data providers. Concurrent execution of thousand subqueries generated by multiple federated query engines makes the query tracking process challenging and uncertain. Experiments with Anapsid show that FETA is able to extract BGPs which, even in a worst case scenario, contain BGPs of original queries.

1 Introduction

The federation of the Linked Data (LD) interlinks massive amounts of data across the Web. Federated query engines [12, 1, 2, 6, 10], allow data consumers to query data residing in the federation in a transparent way as if they were a single RDF graph.

Query engines split user's query into subqueries distributed among SPARQL endpoints without revealing the whole federated query. Hence, data providers do not know the complete federated query in which they participate, they do not know which of their data are combined, when and by whom. Consequently, data providers have a partial access to secondary data [3, 4], unlike data warehousing approaches.

In this paper we propose FETA (FEderated query TrAcking), a query tracking algorithm that computes original federated BGPs (Basic Graph Patterns) from shared logs maintained by data providers. Concurrent execution of thousand subqueries generated by multiple federated query engines makes the query tracking process challenging and uncertain. To tackle this problem, we developed a set of heuristics that links or unlinks variables used in different subqueries of a join federated query. We experimented FETA over concurrent execution of queries of the benchmark FedBench [11]. Even in a worst case scenario, FETA extracts BGPs that contain federated BGPs used in original queries.

The paper is organized as follows: Section 2 introduces a motivating example and describes the scientific problem. Section 3 presents FETA and heuristics for query deduction. Section 4 illustrates experimental results. Section 5 overviews some related work. Finally, conclusions and future work are outlined in Section 6.

2 Background and Motivations

Given a SPARQL query and a federation defined as a set of SPARQL endpoints, a federated query engine performs the following tasks [7, 9]: (i) *query decomposition*, normalizes, rewrites and simplifies queries; (ii) *data localization*, performs source selection among defined federation and rewrites the query into a distributed query; (iii) *global query optimization*, optimizes distributed query by rewriting an equivalent distributed query with various heuristics: minimizing intermediate results, minimizing number of calls to endpoints, etc. In Figure 1, we observe that two join operations will be executed in the federated query engine with data coming from subqueries sent to SPARQL endpoints; (iv) *distributed query execution*, executes the optimized plan with physical operators available in federated query engines.

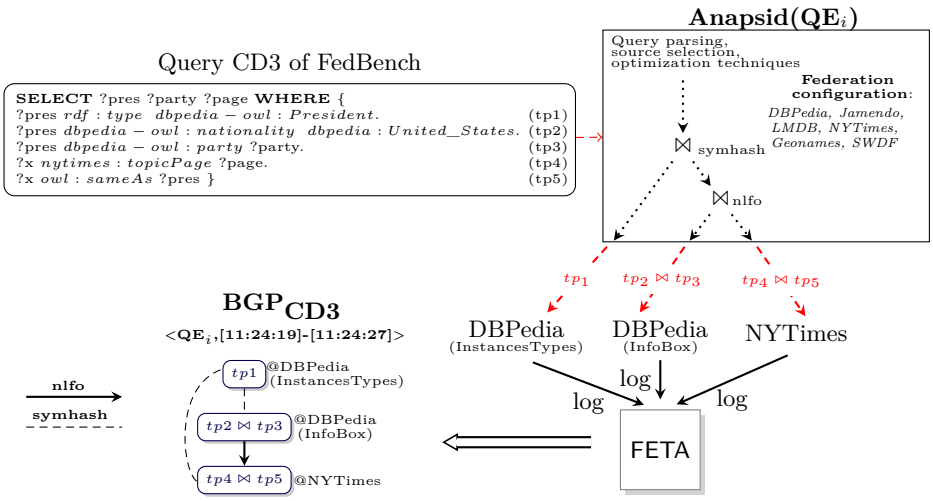

Fig. 1. Query processing and FETA's deduction for CD3.

Federated query tracking infers federated queries from a shared log maintained by data providers. We illustrate the general process in Figure 1. A federated query engine executes a federated query on a federation of SPARQL endpoints. FETA collects the logs of a federation of data providers and infers BGPs used in federated queries. By this way, data providers that collaborate have access to secondary data. In our example, FETA allows NYTimes data provider

to know which of his data are used in conjunction with DBPedia data. Query tracking can be applied to many federated query engines, in this paper we focus on tracking queries processed by the Anapsid [1] federated query engine, with its join physical operators, nested loop with filter options (`nlfo`) and symmetric hash (`symhash`).

Time	Subquery/Answer	Endpoint
11:24:19	**(Subquery 1)** SELECT `?pres` WHERE { ?pres rdf:type dbpedia-owl:President }	DBPedia InstancesTypes
11:24:23	**(Answer)** { {var: "pres", values: "http://dbpedia.org/resource/Ernesto_Samper,..., "http://dbpedia.org/resource/Shimon_Peres,..., http://dbpedia.org/resource/Barack_Obama" } }	DBPedia InstancesTypes
11:24:21	**(Subquery 2)** SELECT ?party `?pres` WHERE { ?pres dbpedia-owl:nationality dbpedia:United_States . ?pres dbpedia-owl:party ?party }	DBPedia InfoBox
11:24:24	**(Answer)** { {var: "party", values: "http://dbpedia.org/resource/Democratic_Party_ "%28United_States%29,..., http://dbpedia.org/resource/Independent_%28politics%29", http://dbpedia.org/resource/Republican_Party_%28US%29" }, { var: "pres", values: "***http://dbpedia.org/resource/Barack_Obama***,..., ***http://dbpedia.org/resource/Johnny_Anders***,..., ***http://dbpedia.org/resource/Judith_Flanagan_Kennedy***,..." } }	DBPedia InfoBox
11:24:25	**(Subquery 3)** SELECT `?pres` ?x ?page WHERE { ?x nytimes:topicPage ?page . ?x owl:sameAs ?pres . FILTER ((***?pres=<http://dbpedia.org/resource/Barack_Obama>***) || (***?pres=<http://dbpedia.org/resource/Johnny_Anders>***) || (***?pres=<http://dbpedia.org/resource/Judith_Flanagan_Kennedy>***),...) || }} LIMIT 10000 OFFSET 0	NYTimes
11:24:27	**(Answer)** { {var: "pres", values: "http://dbpedia.org/resource/Barack_Obama" } { var: "x", values: "http://data.nytimes.com/47452218948077706853" } { var: "page", values: "http://topics.nytimes.com/top/reference/timestopics/ people/o/barack_obama/index.html" }	NYTimes

Table 1. Partial logs of DBPedia (InstancesTypes, InfoBox) and NYTimes.

Figure 1 illustrates how Anapsid processes query CD3 from FedBench[1] [11]. The goal is to find all US presidents, their party membership and pages with news about them. Table 1, presents an extraction of federated log with traces of

[1] Prefixes for queries presented in this article are:
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geonames: <http://www.geonames.org/ontology#>
PREFIX linkedMDB: <http://data.linkedmdb.org/resource/movie/>
PREFIX nytimes: <http://data.nytimes.com/elements/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX purl: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

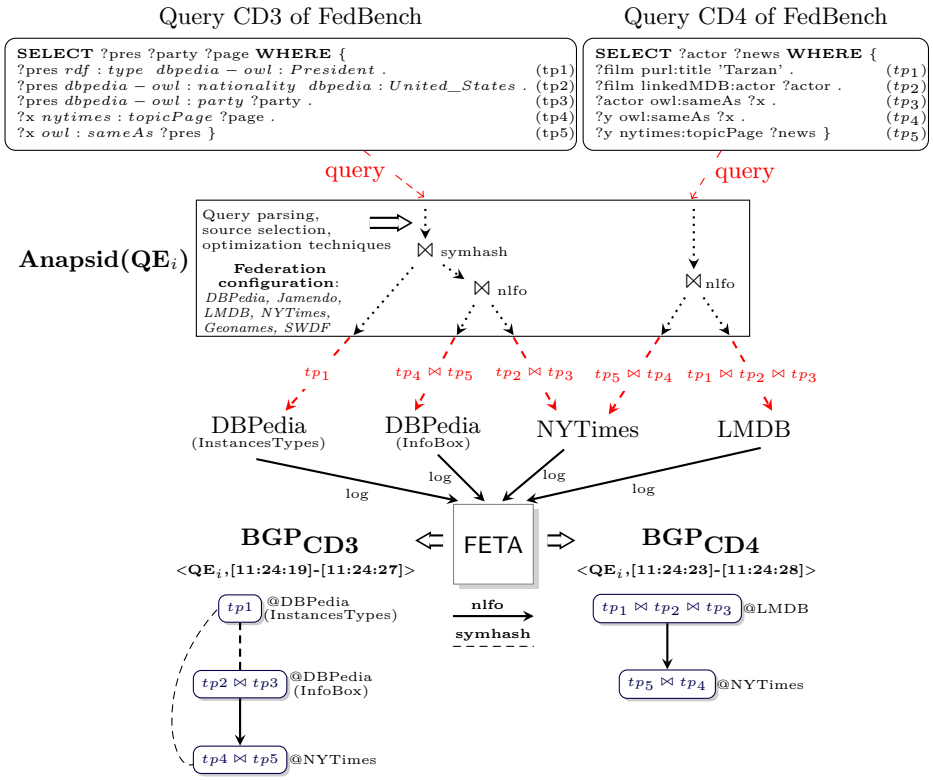

Fig. 2. Query processing and FETA's deduction for CD3, CD4 concurrent execution.

the CD3 execution. It contains some subqueries and associated answers[2] of endpoints. These traces correspond to subqueries sent by one query engine, identified by its IP address.

In Figure 1 we see that Anapsid evaluates individually $tp1$ of CD3, at DBPedia-instances types. Subsequently, this query engine choses a `nlfo` implementation to join data retrieved from DBPedia-infobox and NYTimes. Consequently, Anapsid sends $tp2 \bowtie tp3$ to DBPedia-infobox and stores intermediate results. Then, it calls NYTimes with several subqueries containing these intermediate results in filter options. This is confirmed in Table 1, where answers of subquery 2 from DBPedia-infobox are injected in the filter part of subquery 3 and sent to NYTimes. Anapsid iterates until all intermediate results are sent to NYTimes, in order to avoid reaching the endpoint's limit response. Finally, results of the `nlfo` implementation are joined locally at Anapsid with results of the first triple pattern's evaluation, using the `symhash` operator.

Operator `nlfo` is represented by an arrow because the order of the join is deduced from logs, i.e., it is possible to know in which direction the nested loop

[2] Query results are sent in *JSON* format.

is made. But, `symhash` is represented by a dash line because it is impossible to know the order of the join made locally by the query engine.

It is clear that a single federated query can generate many subqueries sent to endpoints according to physical join operators. However, such behavior can be tracked if endpoints collaborate and BGPs from the federated query can be inferred.

Federated query tracking is challenging if many federated queries having common join conditions are executed concurrently. The most challenging case is when, in addition, queries are sent by the same query engine. We propose some heuristics to separate subqueries belonging to different federated queries sent by the same query engine. Figure 2 shows the concurrent execution of queries CD3 and CD4 sent by query engine QE_i. These queries have common variable `?x`. When logs are federated, this variable joins BGPs of both queries.

Problem statement. Given a federated log containing independent subqueries, link subqueries on their common join conditions, i.e., variable, IRI or literal, if they participate to the same federated query and deduce BGPs processed by the federation. The desired output is a set of BGPs indicating (i) which endpoints evaluated which triple patterns, (ii) whom issued the federated query, and (iii) in which period of time the deduced BGP was processed.
At the bottom of Figures 1 and 2 there are deduced BGPs corresponding to queries appearing at the top of these Figures. Information about which endpoints collaborate with which triple pattern, the federated query engine that issued the query, and the time period where the queries were processed appear too.

3 FETA: FEderated query TrAcking

Figure 3 describes how FETA processes a federated log. A federated log is a sequence of subqueries with answers as described in Table 1. The goal is to link different subqueries participating in the same join, in order to reconstruct federated BGPs. $T_{\bowtie}$, *Same queries* and *Common join condition*, allow to join subqueries' BGPs. *NLFO*, *Same Concept/Same As* and *Not Null Join* verify joins and potentially unlink BGPs.

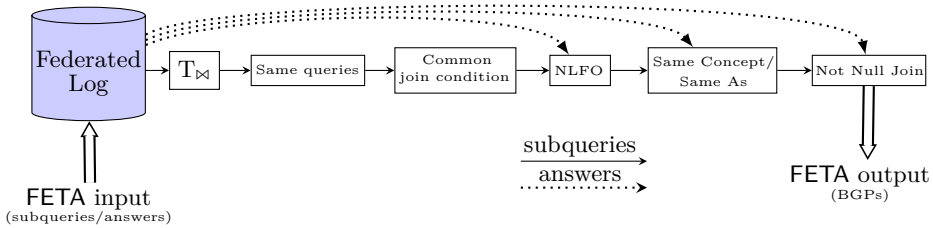

Fig. 3. Workflow processing of FETA's heuristics.

$T_{\bowtie}$ identifies subqueries of the same time interval that will be analyzed together. For instance, all subqueries in Table 1, are captured in the interval of few seconds. The challenge is to choose the appropriate time interval. A small window may separate subqueries pertaining to the same federated query. A large window may join subqueries of different federated queries.

Same queries merges identical subqueries but also subqueries differing only in their offset values. Same queries are sent twice to the same endpoint to be sure obtaining an answer and to different endpoints in order to have complete answers. For instance, every query in Table 1 is sent twice consecutively to each selected endpoint. Additionally in Figure 6, we observe that the first two triple patterns of CD6 are evaluated separately to different endpoints, with the aim of having complete answers. Similar queries with different offsets, on the other hand, are sent to avoid reaching the endpoint's limit response. For instance, the subqueries below which are sent to Geonames for evaluating query CD7, are merged by FETA and considered as a unique subquery without the part limit/offset:

```
SELECT ?location ?news WHERE {
?y <http://data.nytimes.com/elements/topicPage> ?news
?y <http://www.w3.org/2002/07/owl#sameAs> ?location }
LIMIT 10000 OFFSET 0
```

```
SELECT ?location ?news WHERE {
?y <http://data.nytimes.com/elements/topicPage> ?news
?y <http://www.w3.org/2002/07/owl#sameAs> ?location }
LIMIT 10000 OFFSET 10000
```

Common join condition, joins BGPs of queries having common projected variables or triple patterns with common IRI/literal on their subjects or objects. We are aware that, in general, subqueries are joined on their common projected variables. However, we consider also IRIs and literals, even if it can produce some noise on our deduction approach, because they are used in some cases as a common join condition. For instance, in Figure 4, the IRI *dbpedia:Barack_Obama*, is a join condition between triple patterns of CD2. We presume that subqueries with common join condition, closely in time, may be joined locally at the query engine using the symmetric hash operator. For instance, in Table 1, all subqueries have variable `?pres` in common and thus we suppose they are joined at Anapsid.

Nested Loop with Filter Options (NLFO), verifies if BGPs, joined in the previous heuristic, were executed with a nested loop operator. In particular, we group queries varying only in their filter values, if these values are contained in answers of a previously evaluated subquery, with which we confirm that they are joined. For instance, in Table 1 we identify that filter values of subquery 3 correspond to answers of variable `?pres`, e.g., *<http://dbpedia.org/resource/Barack_Obama>*, for subquery 2. This certifies a `nlfo` between subqueries 2 and 3, discarding a global `symhash` join among the three subqueries.

Same Concept/Same As, verifies if answers of joined queries correspond to same concepts or concepts related with a *sameAs* property.[3] If this is not the case, concerned BGPs are unlinked. For instance, in Table 1, answers of the triple pattern of subquery 1 have the same concept with the second triple pattern of subquery 3, for variable `?pres`, i.e., *<http://dbpedia.org/ontology/President>*.

Not Null Join, verifies if a join returns an empty set of answers. If this is the case, concerned BGPs are unlinked. For instance, in Table 1, triple patterns in subqueries 1 and 3 have a common value for projected variable `?pres`, i.e., *<http://dbpedia.org/resource/Barack_Obama>* and therefore they remain linked in the same BGP.

4 Experiments

We analyzed the collection of 7 federated queries of Cross Domain (CD) of the benchmark FedBench [11]. Datasets are those concerned by these queries: DBPedia, Jamendo, LMDB, NYTimes, SWDF and Geonames. Virtuoso OpenLink[4] 6.1.7 is hosting SPARQL endpoints. We used Anapsid 2.7 as federated query engine with the cache disabled. Answers of endpoints to the subqueries they received, are captured with tcpdump 4.5.1[5]. FETA is implemented in Java 1.7 and the collected federated log is stored in a CouchDB database[6].

We evaluated FETA under two configurations. In the first configuration, one Anapsid client processes all federated queries sequentially with a delay between each query. In the second one, one Anapsid client processes all queries concurrently, each one into an individual thread. Executing queries concurrently from a single client is clearly a worst case scenario for FETA because the IP address of the client cannot be used to split subqueries of the federated log. For the scope of this paper, we suppose that all endpoints concerned by federated queries share their logs.

With the first configuration, FETA reconstructs correctly all federated BGPs of the CD collection. [7] We focus now on the second experiment. In this case, Anapsid produces 529 subqueries.[8] Size of queries and answers logs are of 300KB and 14MB, respectively.

Table 2 shows the number of BGPs produced after each heuristic following the FETA execution workflow shown in Figure 3. FETA processes this federated log in approximately 90 seconds. $T_{\bowtie}$ is not significant here, we consider it big enough to cover the execution of all federated queries of CD. Initial log contains 238 SELECT subqueries as they are unlinked, there are 238 BGPs. *Same queries*

[3] Note that we do not consider generic concepts for this heuristic, e.g., *<http://www.w3.org/2002/07/owl#Thing>*.
[4] http://virtuoso.openlinksw.com/
[5] http://www.tcpdump.org/
[6] http://couchdb.apache.org/
[7] Each deduced BGP corresponds to the federated query, once simplified and rewritten by the query engine at the *query decomposition* phase.
[8] Note that we subsequently remove ASKs and consider only SELECT subqueries.

FETA Heuristic	Number of produced BGPs
Same queries	109
Common join condition	1
NLFO	1
Same Concept/Same As	2
Not Null Join	4

Table 2. Number of BGP's produced by heuristic.

heuristic removes or merges more than 60% of subqueries and their respective answers, producing 109 BGPs. *Common join condition* produces a single BGP because chaining among queries. *NLFO* confirms joins, by identifying the injection of answers from a subquery into subqueries which vary only in their filter values. *Same Concept/Same As* heuristic unlinks some joins and certifies others. *Not Null Join* certifies that a symmetric hash is certainly possible, because an intersection of answers on a common projected variable of every two subqueries.

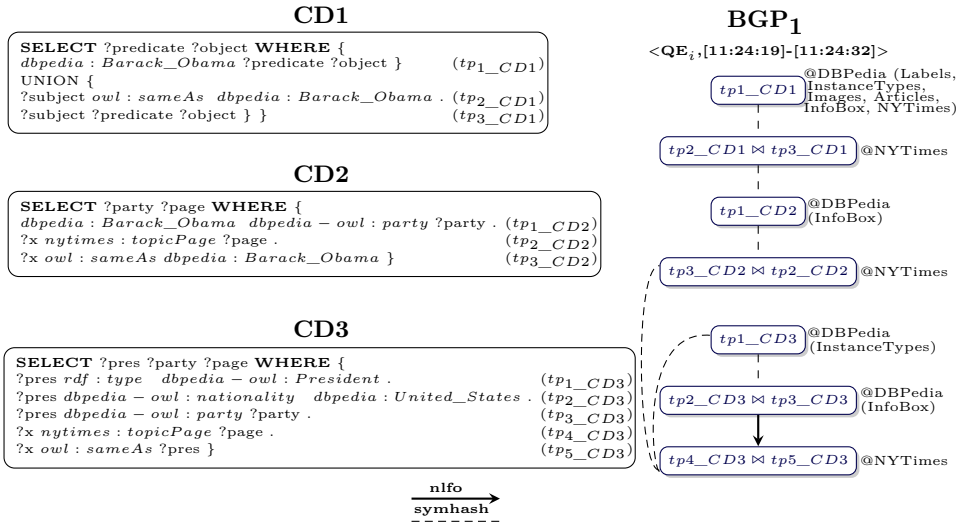

Fig. 4. FETA's deduced BGP_1 for CD concurrent execution.

Figures 4-7 present BGPs extracted by FETA for all concurrently executed CD queries. Note that query plans established by Anapsid for each query, may differ depending on endpoints availability and when operators are blocked. Ideally, FETA should reconstruct 8 BGPs. The CD collection consists of 7 queries but CD1 is a union query normally decomposed in 2 BGPs. FETA extracted 4 BGPs containing the 8 original BGPs. Even if this result is not precise, extracted BGPs with endpoints' information give valuable information to data providers

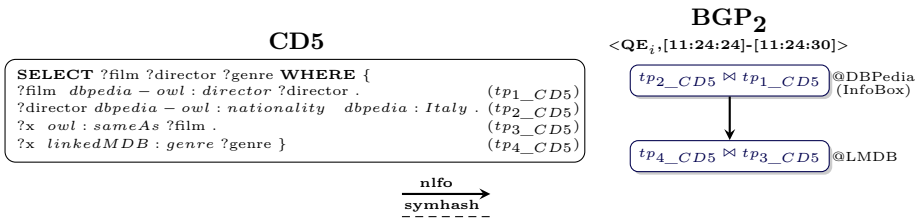

Fig. 5. FETA's deduced BGP_2 for CD concurrent execution.

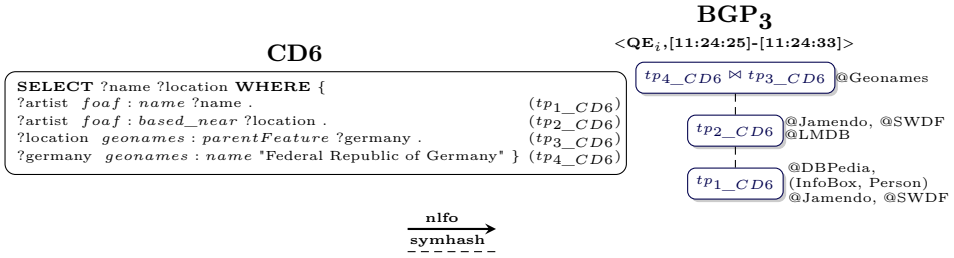

Fig. 6. FETA's deduced BGP_3 for CD concurrent execution.

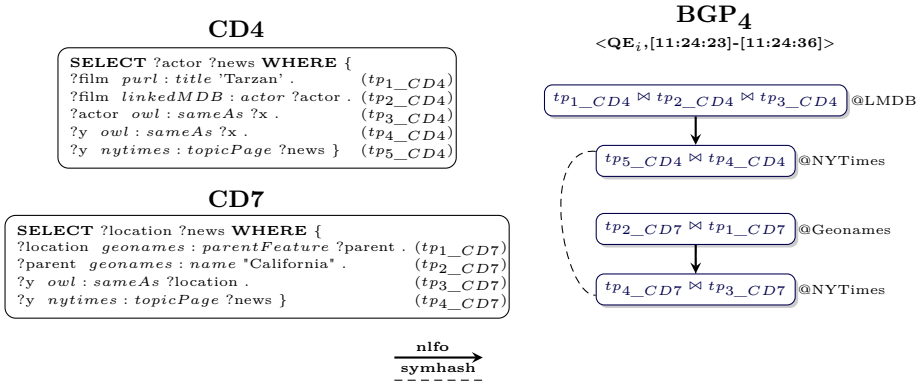

Fig. 7. FETA's deduced BGP_4 for CD concurrent execution.

bout how their data are processed and (potentially) joined with other endpoints. In the following paragraphs, we explain how FETA deduces each particular BGP.

Figure 4 describes how FETA processes federated queries CD1, CD2 and CD3. CD1 is composed of two BGPs separated by a union, which we expect to identify individually. In fact, these two BGPs were deduced as a single BGP because they have a common IRI, *dbpedia:Barack_Obama* and also share common answers for both variables `?predicate` and `?object` which concern *Barack Obama*. Next, we observe that CD1, CD2, and CD3 were grouped in one BGP. CD1 and CD2

were not separated because of the common IRI *dbpedia:Barack_Obama*, which is actually also a join condition between triple patterns of CD2. BGPs of CD2 and CD3 were not separated because results of CD2 are included in CD3 for their common triple pattern, $?x$ *nytimes* : *topicPage* $?page$, but also for the other triple patterns of CD2.

In Figures 5 and 6, we can see that BGPs of CD5 and CD6 were well reconstructed. CD4 and CD5 are linked in the same BGP with *Same Concept/Same As*, as both concern films. Subsequently, *Not Null Join* separated CD4 from CD5 on the content of the `?film` variable, as CD4 concerns films related to Tarzan while CD4 concerns films of Italian directors. In a similar way, CD6 and CD7 are linked with *Same Concept/Same As*, as both concern localizations but they were separated because they have no common answer for variable `?location` as the first concerns the Federal Republic of Germany and the second California, USA.

Figure 7 shows FETA's deduced BGP, grouping CD4 and CD7. On the other hand CD4 and CD7 share common concepts but also answers because, for the currently employed heuristics, we infer that these two queries share the same triple pattern $?y$ *nytimes* : *topicPage* $?news$.

From this experiments we conclude that (i) it is possible to reconstruct precise federated BGPs if federated queries are different enough, and (ii) reconstructed BGPs contain all original BGPs, i.e., false joins are not deduced.

5 Related Work

Federated query tracking is related to web tracking [8]. In web tracking, a first-party website authorizes a third party to learn about its users. Analogously, FETA plays the role of the third party. However, logs collected in federated query tracking is the result of the execution of a physical plan in distributed query processing compared to a more simple web navigation flow in web tracking.

Extracting information from raw logs is traditionally a data mining process [5] involving the following steps: (i) *data selection* identifies the target dataset and relevant attributes that will be used to derive new information; (ii) *data cleaning* removes noise and outliers, transforms field values to common units, generates new fields and finally brings the data into the structured data schema that is used for storage, e.g., relational databases, XML; (iii) *data mining* applies data analysis and discovery algorithms based on machine learning, pattern recognition, statistics and other methods; (iv) finally *evaluation* presents the new knowledge in a form that will be also understandable from the end user, e.g., through visualization.

FETA can be located at the data mining step because it transforms raw logs into a sequence of sets of BGPs. In experiments, we presented one extraction for a period of time. Repeating extractions will generate a sequence of extracted BGPs than can be used for association rules mining of frequent pattern detection.

6 Conclusions and future work

Federated query tracking allows data providers to access secondary data in Linked Data federation. We proposed FETA, a federated query tracking approach that extracts original federated Basic Graph Patterns from a shared log maintained by data providers. FETA links and unlinks variables present in different subqueries of the federated log by applying a set of heuristics we presented in this paper.

Even in a worst case scenario with Anapsid, FETA extracts BGPs that contain original BGPs of federated join queries. Extracted BGPs with endpoints' information give valuable information to data providers about which triples are joined, when and by whom.

We think FETA opens several interesting perspectives. First, heuristics can be improved in many ways by better using semantics of predicates and answers. Second, we conducted experiments on one slice of federated log. Repeating BGPs extractions on successive slices allows to apply traditional data mining techniques such as extracting frequents patterns. Third, we limited experiments to Anapsid. Extending to FedX query engine [12] will challenge proposed heuristics because FedX physical operators produce slightly different query traces.

References

1. M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, Proceedings, Part I*, pages 18–34, 2011.
2. O. Görlitz and S. Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23*, 2011.
3. S. Grumbach. Intermediation Platforms, an Economic Revolution. *ERCIM News*, 2014(99), 2014.
4. S. Grumbach, M. Rafanelli, and L. Tininini. Querying Aggregate Data. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, Philadelphia, Pennsylvania, USA*, pages 174–184, 1999.
5. J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.
6. O. Hartig, C. Bizer, and J. C. Freytag. Executing SPARQL Queries over the Web of Linked Data. In *The Semantic Web - ISWC 2009 - 8th International Semantic Web Conference, Chantilly, VA, USA, October 25-29, Proceedings*, pages 293–309, 2009.
7. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.
8. J. R. Mayer and J. C. Mitchell. Third-Party Web Tracking: Policy and Technology. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May, San Francisco, California, USA*, pages 413–427. IEEE Computer Society, 2012.
9. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

10. B. Quilitz and U. Leser. *Querying distributed RDF data sources with SPARQL.* Springer, 2008.
11. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fed-Bench: A Benchmark Suite for Federated Semantic Data Query Processing. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, Proceedings, Part I*, pages 585–600, 2011.
12. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, Proceedings, Part I*, pages 601–616, 2011.