

# Evaluating Adaptive Query Processing Techniques for Federations of SPARQL Endpoints

Maribel Acosta and Maria-Esther Vidal

Departamento de Computación  
Universidad Simón Bolívar  
Caracas 89000, Venezuela  
{macosta,mvidal}@ldc.usb.ve

**Abstract.** We present ANAPSID and illustrate the benefits of adaptive semantic data management techniques for accessing a federation of SPARQL endpoints. ANAPSID adapts query execution schedulers to data availability and runtime conditions, implements physical SPARQL operators that detect when an endpoint becomes blocked or data traffic is bursty, and opportunistically produces results as quickly as data arrives from the sources. Additionally, ANAPSID operators implement main memory replacement policies to move previous computed matches to secondary memory avoiding duplicates. We show ANAPSID performance with respect to a variety of RDF engines for queries of diverse complexity, and show that ANAPSID may outperform existing engines.

## 1 Introduction

Following the design rules of Linked Data, the number of available SPARQL endpoints that support remote query processing is quickly growing. Further, SPARQL has been extended to support the specification of queries against federations of endpoints, and several engines have already implemented this feature [3, 4]. However, finding execution plans for federated queries and execute them against remotely stored datasets is not always feasible. The lack of statistics about the performance of the endpoints, unpredictable data transfers and endpoints availability, make the traditional optimize-then-execute paradigm not completely applicable in the context of federations of endpoints. Contrary, it may be necessary to decompose federated queries into simple sub-queries, so that the endpoints will be capable to execute them in a reasonable time, and follow delay-hiding features and routing techniques to react to endpoints availability and execute first non-delayed parts of the plan. We present ANAPSID [1], an adaptive query engine for the SPARQL 1.1 federation extension<sup>1</sup> that adapts query execution schedulers to data availability and runtime conditions. ANAPSID stores information about the available endpoints and the ontologies used to

---

<sup>1</sup> <http://www.w3.org/TR/rdf-sparql1-query/>

describe the data, to decompose queries into sub-queries that can be executed by the selected endpoints. Also, adaptive physical operators are executed to produce answers as soon as responses from remote available sources are received. We demonstrate the benefits of these techniques and compare their performance with respect to state-of-the-art RDF engines that provide access to federations of endpoints. The demo is published at <http://maribelacosta.com/anapsid/> and illustrates the techniques defined in [1].

## 2 ANAPSID Architecture

ANAPSID is based on the architecture of wrappers and mediators; lightweight wrappers are created around SPARQL endpoints to properly build the URL to contact the endpoint, and to convert the results in appropriate formats. Mediators maintain information about capabilities of endpoints and statistics that describe their content and performance; also, they decompose user queries into sub-queries in the available endpoints. Additionally, ANAPSID implements a query engine and a set of operators able to gather data from different endpoints; opportunistically, it produces results by joining tuples even when one of the endpoints becomes blocked. The **agjoin** and **adjoin** are based on non-blocking join operators, defined to quickly produce answers from streamed data accessible through a wide-area network, and hide delays by working with tuples previously received; **adjoin** is a non-commutative operator that is required to dereference data from SPARQL endpoints. The **aoptional** and **aunion** implement adaptive versions of the *optional* and *union*, respectively. Finally, ANAPSID execution engine implements delay-hiding features which react to endpoints availability. Eddy operators [2] are developed to reschedule incoming tuples through non-delayed operators of the plan; it can also introduce new operators to execute the query in small stages or to produce partial answers; e.g., it may change an **adjoin** by an **aoptional**, and treat joins with unavailable tuples as non matching joins. Figure 1 shows the components that comprise the ANAPSID architecture.

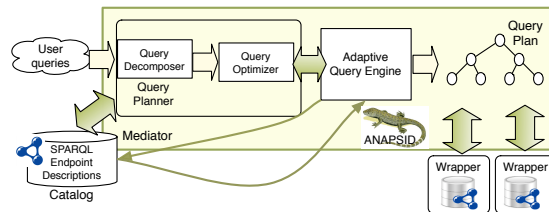


Fig. 1. The ANAPSID Architecture

### 3 Demonstration of Use Cases

Consider the following query specified in SPARQL 1.1: *Retrieve the clinical trials, the information of the studied drugs, and the genes of the diseases: Breast Cancer, Colorectal Cancer, Ovarian Cancer, and Lung Cancer.*

```
SELECT DISTINCT ?CT1 ?CT2 ?CT3 ?CT4 ?I ?II ?TGD ?GN
WHERE {{SERVICE <http://virtuoso.bd.cesma.usb.ve/sparql> {
  ?A2 <http://data.linkedct.org/resource/linkedct/condition_name> "Colorectal Cancer" .
  ?CT1 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
  ?CT1 <http://data.linkedct.org/resource/linkedct/condition> ?A2 .
  ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II }} .
{SERVICE <http://virtuoso.bd.cesma.usb.ve/sparql> {
  ?A4 <http://data.linkedct.org/resource/linkedct/condition_name> "Breast Cancer" .
  ?CT2 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
  ?CT2 <http://data.linkedct.org/resource/linkedct/condition> ?A4 .
  ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II}} .
{SERVICE <http://virtuoso.bd.cesma.usb.ve/sparql> {
  ?A6 <http://data.linkedct.org/resource/linkedct/condition_name> "Ovarian Cancer" .
  ?CT3 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
  ?CT3 <http://data.linkedct.org/resource/linkedct/condition> ?A6 .
  ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II}} .
{SERVICE <http://virtuoso.bd.cesma.usb.ve/sparql> {
  ?A8 <http://data.linkedct.org/resource/linkedct/condition_name> "Lung Cancer" .
  ?CT4 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
  ?CT4 <http://data.linkedct.org/resource/linkedct/condition> ?A8 .
  ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II}} .
{SERVICE <http://http://www4.wiwiss.fu-berlin.de/drugbank/sparql>
  ?II <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/target> ?TGD.
  ?TGD <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/genbankIdGene> ?GN.}} .}
```

This query may time out after several hours without producing any answer when it is run against state-of-the-art RDF engines as ARQ 2.8.8<sup>2</sup> or Virtuoso<sup>3</sup>. ARQ executes all the joins as *Nested Loop joins* (Figure 2(a)), and invokes many times the different endpoints which fail executing sub-queries because the maximum number of allowed requests is exceeded. Virtuoso has to dereference the URL that instantiates the variable ?II, and either reaches the timeout of one day with no answers, or returns a proxy error: **the proxy server could not handle the request GET/sparql**. In contrast, ANAPSID schedules the sub-queries against the services in a bushy-tree plan as shown in Figure 2(b), and combines sub-plans with the physical operators **agjoin** and **adjoin** to fire the corresponding adaptive procedures that hide delays, and opportunistically produce results by joining tuples previously received from their corresponding endpoints. ANAPSID can return the answers in less than 7 secs.

We will demonstrate the performance of different RDF store engines when queries of diverse complexity are executed against the SPARQL endpoints of the Linked Datasets: LinkedCT<sup>4</sup>, Drugbank<sup>5</sup>, and DBPedia<sup>6</sup>. We will illustrate that:

<sup>2</sup> <http://jena.sourceforge.net/ARQ/>

<sup>3</sup> <http://virtuoso.openlinksw.com>

<sup>4</sup> <http://virtuoso.bd.cesma.usb.ve/sparql>

<sup>5</sup> <http://www4.wiwiss.fu-berlin.de/drugbank/sparql>

<sup>6</sup> <http://dbpedia.org/sparql>

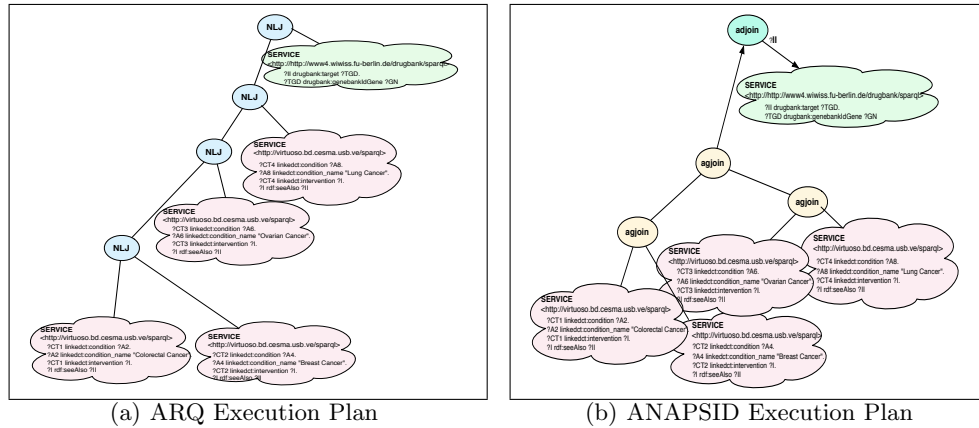


Fig. 2. Diverse Execution Plans

- ARQ 2.8.8. BSD-style<sup>7</sup> failed evaluating the majority of the queries after 12 hours. In other cases, ANAPSID overcame ARQ execution time by a factor of 4,160.56.
- Virtuoso endpoints were not able to execute these queries, because they could not dereference the corresponding URIs before meeting the timeout.
- ANAPSID adaptive operators are able to produce all answers in almost the time required by the Symmetric Hash Join [2] to produce the first tuple.

## 4 Conclusions

We present ANAPSID and illustrate results that suggest that our proposed techniques may reduce execution times by up to several orders of magnitude, and are able to produce answers when other engines fail. Also, depending on data distributions and transfer delays, ANAPSID may overcome existing non-blocking operators.

## References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: AN Adaptive query ProcesSing engine for sparql endpoints. In *Accepted at ISWC*, 2011.
2. A. Deshpande, Z. G. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
3. B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, pages 524–538, 2008.
4. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: A federation layer for distributed query processing on linked open data. In *ESWC (2)*, pages 481–486, 2011.

<sup>7</sup> <http://sourceforge.net/projects/jena/>