Faster SPARQL Federated Queries

Antoine Abel

To cite this version:

IRISA INRIA Rennes

Faster SPARQL Federated Queries

Author

Antoine ABEL

Supervisor

Olivier DAMERON

Olivier CORBY

Master 2 BioInformatique
Parcours Informatique et Biologie Intégrative
Année universitaire 2018-2019

21st June 2019

Master bioinformatique

UNIVERSITÉ DE RENNES 1

ENGAGEMENT DE NON PLAGIAT

Je, soussigné(e) ABEL Antoine
étudiant(e) en Master d Bio-Informatique
déclare être pleinement informé que le plagiat de documents ou d'une partie de document publiés sur toute forme de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour la rédaction de ce document.

Date : 14/06/19

Signature : [illegible]

Document à compléter de manière manuscrite et à insérer obligatoirement en première page du rapport de stage.

Contents

1 Introduction

1.1 Life science databases and the need to integrate them

In life science over the years, studies of all domains gathered lots of data in different places and as fields of studies grows the number of databases also increases. Lots of research teams have their own specific database and update it at different speed. Some initiative to integrate databases have emerge through the year such as Intermine [3] and BioMart [9].

1.2 Semantic Web technologies

Semantic Web technologies are a broader initiative (i.e. not limited to life science) of the W3C for addressing data integration [1]. They provide a unified technical framework called RDF, allowing to describe entities and relations between them across several datasets (called endpoints). SPARQL is the associated query language. This will be further explained in the Background section.

Over the last two decades, they have been increasingly adopted by the Lifes science community [1]. Currently, most major databases are available in RDF or via SPARQL endpoints. Moreover, the databases now have explicit references to entities from other databases, which reflects the need for integration that we had identified previously. They now constitute the largest and most connected part of the LOD cloud [2].

1.3 Query federation

To accomodate the fact that the description of an entity in a dataset can reference other entities in another dataset, SPARQL supports query federation. This allows to write a single query in which a part concerns the first dataset, and another part concerns the second dataset as if they were one single virtual dataset.

The SPARQL engine processes such federated queries automatically. This is an important difference with relational databases, where the user would have to send the two subqueries to the respective databases, retrieve the results and perform their integration.

A correct but naive approach for processing SPARQL queries consists in sending each subquery to all endpoints. Of course, this leads to a combinatorial explosion and is not

[1]`https://www.w3.org/standards/semanticweb/`
[2]`https://lod-cloud.net/`

practical in terms of performance. State of the art query engine all follow variations of a more elaborate approach composed of three main steps:

- First, the **Query parsing** is the part where the query is divided in small parts, called statement patterns.
- Then, the **Source selection** where the federation engine chooses where to send each statement patterns.
- Finally, the **Query planning** regroups statements patterns that have the same sources and reorganise the query to optimise the join operations.

However, their performances are still not good enough to handle typical life science queries, that are complex and involve large datasets [6].

1.4 General objective

The main objective is to improve the steps of federated query processing. Section 2 provides background information on RDF and SPARQL, analyses the difficulty of processing federated queries and surveys the state of the art query engines and benchmarks. Section 3 defines the internships objectives. Sections 4 and 5 describre the materials and methods and results: we performed an in depth analysis of the query LS6 from LargeRDFbench, we proposed an improved scoring for query planning, implemented it in the Corese query engine and measured the gain of performance. Section 6 is a discussion and synthesis.

2 Background

2.1 RDF

2.1.1 URI

In RDF, resources are identified with Uniform Resource Identifier (URI). URI have the syntax of Uniform Resource Locator (URL) but are not necessarily the address of a document. Therefore URLs are a subset of URIs.

2.1.2 Triple

An RDF triple is a list of a subject, a predicate and an object (Table 1b). The subject is the URI of the resource being described. The predicate is the URI of a relation. The object is one of the values of the predicate for the subject. It can be either an URI if the relation designates another entity, or a string (Table 1a).

DrugID	rdfs:label	DB:molecularWeight
DB00738	Pentamidine	340.4195
DB00005	Etanercept	51234.9

(a) Example of data.

Subject	Predicate	Object
DB:DB00738	rdfs:label	"Pentamidine"
DB:DB00738	DB:molecularWeight	"340.4195"

(b) Example of triple.

Table 1: Data and a part of the corresponding triples.

2.1.3 Graph

A set of RDF triples constitutes a directed graph where the nodes are the resources and the arcs are the triples (Figure 1).

2.2 SPARQL

SPARQL is the language to query RDF data, defined by the W3C consortium [3].

2.2.1 One DB at a time

Users can send a query to an endpoint and retrieve the results. A query is a set of triples where the subject, the predicate or the object can also be variables (with their name starting

[3] `https://www.w3.org/TR/rdf-sparql-query/`

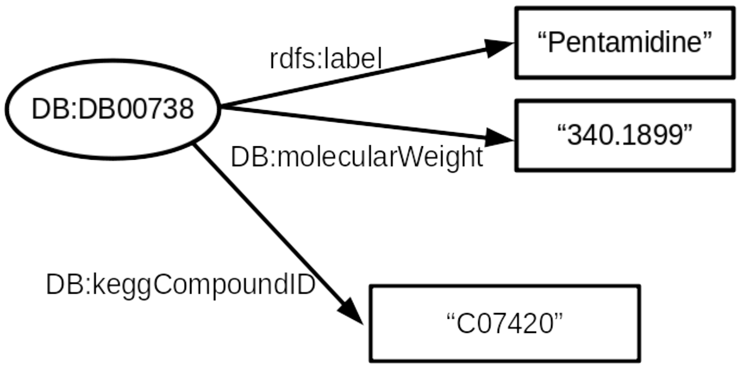


Figure 1: Example of a graph.

by a question mark). The query engine computes all the combinations of the variables' values that satisfy the query. For example, if you want the molecular weight and name of all drugs, your query will be as Figure 2 and the results will display in no particular order (Figure 3).

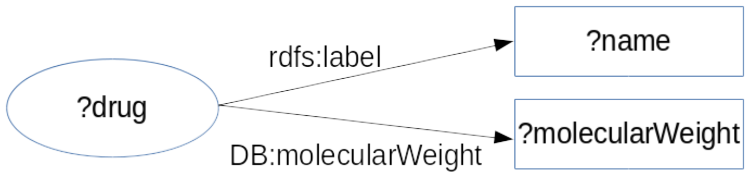
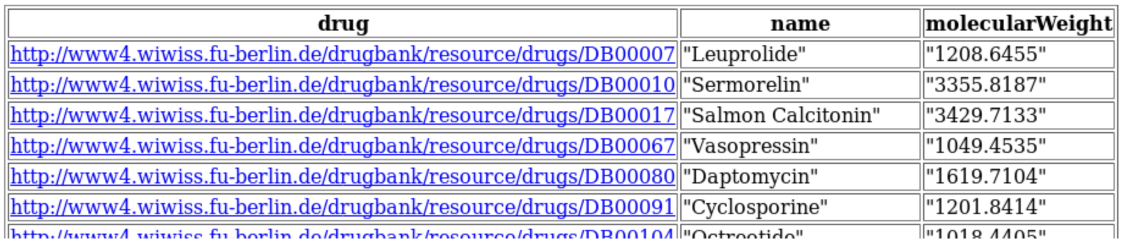


Figure 2: Example of a graph for a simple query.



drug	name	molecularWeight
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00007	"Leuprolide"	"1208.6455"
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00010	"Sermorelin"	"3355.8187"
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00017	"Salmon Calcitonin"	"3429.7133"
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00067	"Vasopressin"	"1049.4535"
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00080	"Daptomycin"	"1619.7104"
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00091	"Cyclosporine"	"1201.8414"
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB00104	"Octreotide"	"1018.4405"

Figure 3: Part of the results for a simple query.

2.2.2 Combining graphs

Because some databases have complementary information of the same resources, they can posses the same URI. This common URI creates a connection (or an "overlap") between the two databases, this is called *entity matching* (Figure 4 & 5). Most of the time, the queries needed by the scientist are complexes and needs to be sent to multiple databases because of that linking.

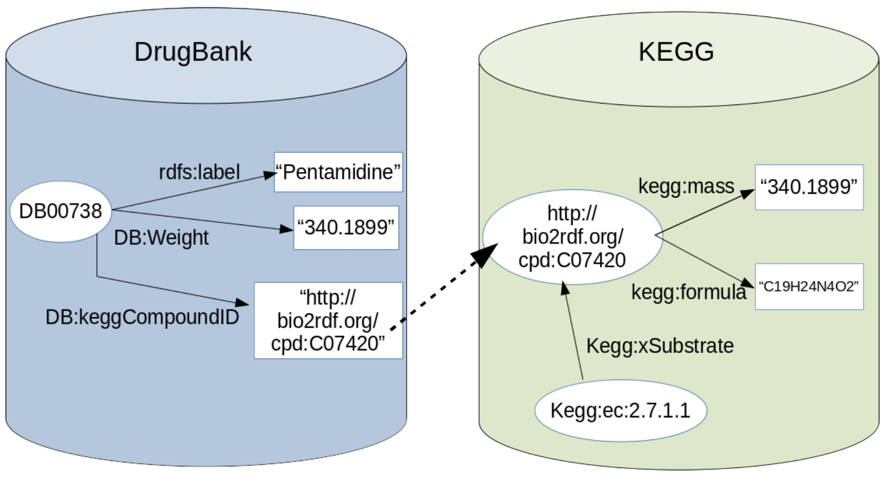


Figure 4: Illustration of linked data between two databases

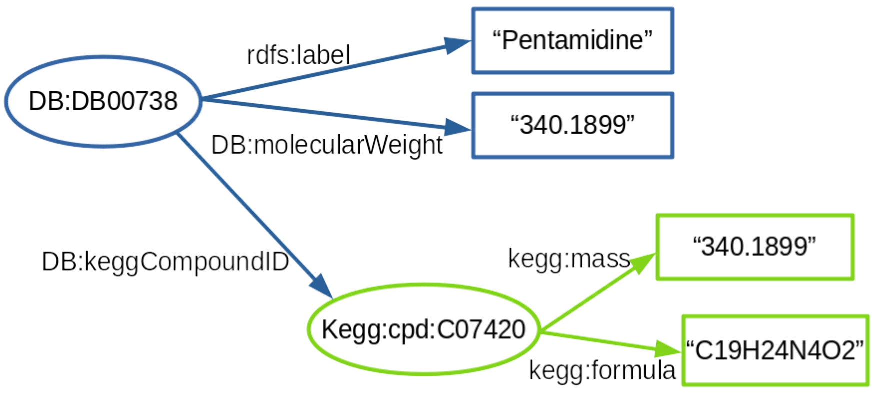


Figure 5: Graph of linked data between two databases

2.2.3 Federation

This shared information means that there is a chance the results will not be complete. As the Figure 6 shows, the results will be endpoint dependent and the user will not have the possible combinations between them. The first solution that comes in mind is to merge the endpoints together but the new dataset size will be too large to be useful. It will also make no sense since the concept of linked open data will not be preserved.

Naive method The naive way is to send each triple to each endpoint and join every results (Figure 7).

Classical method Decompose the query in subqueries and only send these subqueries to the endpoints that could potentially return some results.

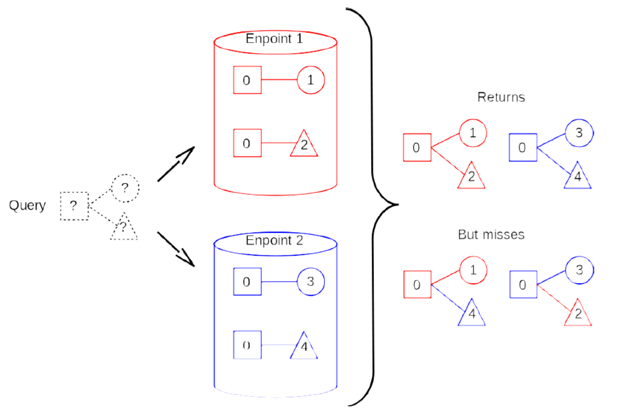


Figure 6: Federated queries: sending the whole query to each endpoint and computing the union of their results misses all the solutions obtained by combining some triples from an endpoint with other triples of another endpoint

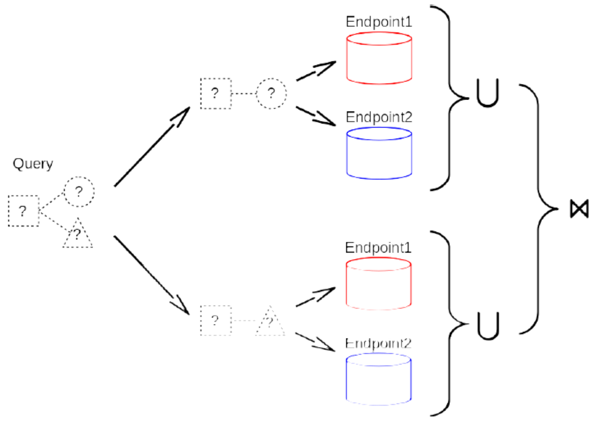


Figure 7: A correct but naive approach for processing a federated query consists in for each triple of the query (1) send it to each endpoint, (2) compute the union of the results, (3) iterate (1) and (2) over all the triples of the query, (4) compute the join of the results of (3).

Splitting the query increases the number of queries sent, the number of join operations and the evaluation time will increase but there will be no more incorrect results. The objectives of the federation engine are to gather the query fragments to reduces joins, select the relevant endpoints to send them and determine the order of processing these fragments.

2.3 Tools and benchmarks

2.3.1 Endpoints

To manage the data at each endpoint and store the triples, the most used software are Virtuoso [4], Fuseki [5], Corese [2] [6], Sesame/RDF4J [7]. Their role is to receive the queries, evaluate it and return the results. Most of them have an interface where users can directly send queries. Here we will use Virtuoso.

[4]https://virtuoso.openlinksw.com/
[5]https://jena.apache.org/documentation/fuseki2/
[6]https://project.inria.fr/corese/
[7]https://rdf4j.eclipse.org/

2.3.2 FedX

Developed in 2011, FedX [8] is implemented in Java and extends the Sesame framework. This modification adds a new source selection method that works with ASK queries sent at each endpoints. This query's result only tells if the endpoint can answer the main one, which inquires fewer searching process. Because every ASK query can be send individually, FedX introduces parallelization, a way that allows multiple tasks to be executed at the same time. This is also used to improve the joining operation.

2.3.3 HiBISCus

HiBISCus [5] is a modification of FedX made by Fluidops. It creates indexes which retain information from ASK queries. The difference with FedX is that the engine searches inside this indexes instead of sending queries, it suppresses the response time induced by ASK queries. The company does not exist anymore and their website is down resulting in the precise documentation being unreachable and little to no support.

2.3.4 PPinSS

Designed and implemented by Guillaume Alviset in 2018, PPinSS is a modification of HiBISCus. This tools generates different indexes with more information. It manages to select fewer sources while giving the same results and being faster as shown in Figure 8. His benchmarks were made with the Life Science (LS) queries.

2.3.5 Grafeque

Vijay Ingalalli worked on Grafeque in 2018 for the Federated Query Scaler project of Dyliss' team [8]. It is a modification of HiBISCus' query planning. The indexes changes again and a second view on the hyper-graph that represents the triples changes the source selection.

2.3.6 Benchmarks

A lots of benchmarks exist but they do not explore the federation with linked data the same ways [6]. Here we will focus on the queries from FedBench [7] and its newer version, LargeRDFBench [4]. These queries have been created in a way that should cover every needs

[8]http://www.irisa.fr/dyliss/public/federatedQueryScaler/

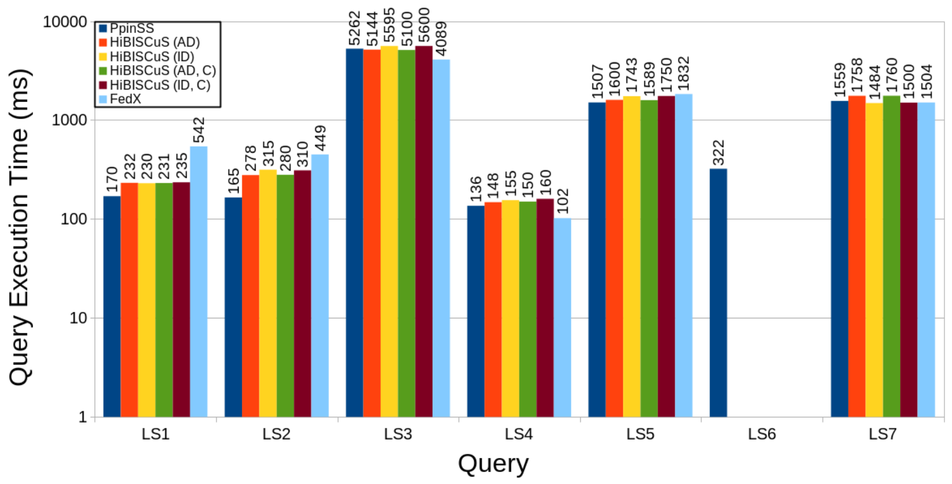


Figure 8: Comparisons of query execution time after source selection for each approach. LS6 has timed out for both HiBISCuS original and character index and also for FedX original. Figure taken from Guillaume's internship report.

researchers may have. After seeing most of them, I have made the observation that they always forgot to check if the results are correct, they only check if they are present.

3 Internship objectives

Thanks to PPinSS the queries evaluation is faster. Its new indexes allows the source selection method to be more strict and grants faster queries evaluation with fewer intermediate results to join. But because of the error on LS6 for HiBISCus, the comparison is not complete. I will continue the benchmarks with HiBISCus and PPinSS, as it needs to be fully functional with each tools and each query and I will add Grafeque. Finally, because FedX is outdated, a migration to the Corese federation engine is considered.

4 Material and methods

4.1 Benchmarks

To compare the performances between HiBISCus, PPinSS and Grafeque, we used queries from FedBench [7] and LargeRDFBench [4]. The focus of the analysis are the queries for the life science domain from FedBench. A bash script has to launch every query with each version of the tools we wanted and specified within the arguments how many times the test needs to be run. The endpoints are created on virtual machine on the GenOuest cluster [9]. They run on Debian (v8.11) with Virtuoso (v7.2) to store the triples. All the virtual machines specifications are summarise in Table 2.

Name	Size(mb)	Number of triple	CPU	RAM(Go)
KEGG	118	1,090,830	2	4
DrugBank	100	517,023	2	4
ChEBI	829	4,772,706	2	8
DBPedia-subset	6,200	42,849,609	2	8
GeoNames	12,000	107,950,085	2	8
Semantic Web Dog Food	0.68	103,595	2	4
New York Times	50	335,198	2	4
Affymetrix	8,800	44,207,146	2	8
Jamendo	141	1,049,647	2	4
Linked Movie Database	851	6,147,996	2	4
TCGA-A	4,100	35,329,868	2	8
TCGA-M	53,000	415,030,327	4	32
TCGA-E	39,000	344,576,146	4	32

Table 2: Endpoints specifications. TCGA-A was added for HiBISCus and PPinSS but because the summaries for Grafeque did not included any TCGA, they were not used for further benchmarks.

This setup allows us to access these endpoints without any conflict in case of external connections.

[9]https://www.genouest.org/

4.2 LS6

The conclusion of Guillaume's work opens to the analysis of LS6, as we can see in the Figure 8. LS6's evaluation with FedX and HiBISCus returned a time-out error with no apparent reason even though their performances with the other queries were acceptable.

First, we have searched for an optimisation of the joining operations by finding every achievable joining order. Only the joining between statements that has common variables are made, otherwise the number of results is increased and the federation engine already optimises it. Given that the ASP is useful for solving search problems, an attempt was made. Then most of the scripts used were written in Python 3, easier and faster to implement.

```
1 SELECT ?drug ?title WHERE {
2         ?drug DB:drugCategory DB:micronutrient .
3         ?drug DB:casRegistryNumber ?id .
4         ?keggDrug w3:22-rdf-syntax-ns\#type bio2rdf:kegg\#Drug .
5         ?keggDrug bio2rdf:bio2rdf\#xRef ?id .
6         ?keggDrug <http://purl.org/dc/elements/1.1/title> ?title .
7 }
```

Listing 1: LS6. Statements from line 2 to 6 were named B1-B5 for better identification when researching with the joining operations. URI were shortened.

The particularity on this query is that HiBISCus selects 7 sources while PPinSS selects only 5 sources, one per statement. In fact, the statement on lines 5 and 6 can be retrieved from two different endpoints. We can see in the query (Listing 1) that the lines 3 and 5 have in common the variable "**?id**", the fist objective was to bring them closer.

Then, the idea of switching every statement of the query leads to new scripts. To be able to do benchmarks with it, I have made a script that finds and rewrites the query in every possible order then launches the evaluation of every new query. This leads to 120 new queries to launch.

4.3 Scoring change

The whole analysis of LS6 brought to us the idea that the query planning of HiBISCus could be improved. No new scripts were written but the debug mode of Eclipse and lots of printing in the terminal were useful.

To change the query planning on HiBISCus, I proposed a new scoring function for the patterns and a new query planning algorithm based on this score.

4.4 Corese

Here Corese will replace FedX in PPinSS. Thanks to Guillaume, PPinSS also comes in a separate package that can be added to any federated engine. It will be used to implement PPinSS to Corese. After the complete migration, a benchmark has been planned, mainly focusing on the Life Science domain queries.

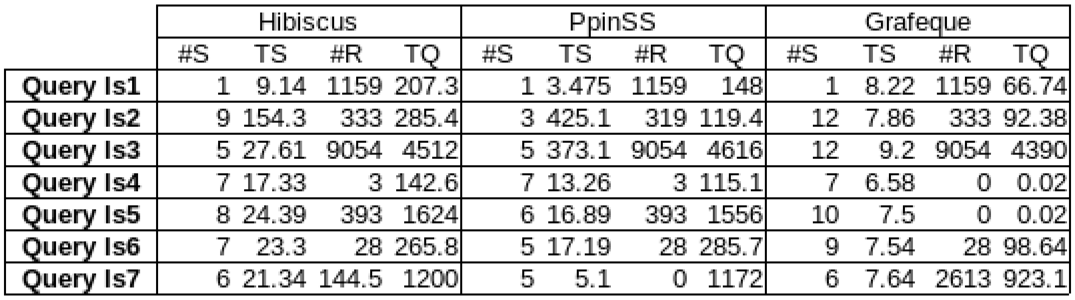
5 Results

5.1 Benchmarks

Once the job on the GenOuest cluster is launched, a log file appears in the user's home directory containing any texts output. For our benchmarks, this log file's format is as follow:

- Current query's name.
- Name of the tool.
- Number of sources selected, Sources selection time (in milliseconds), Number of results, Query evaluation time (in milliseconds).

Having every result in one line separated by a coma allows us to extract the results and to facilitate analysis.



	Hibiscus				PpinSS				Grafeque			
	#S	TS	#R	TQ	#S	TS	#R	TQ	#S	TS	#R	TQ
Query ls1	1	9.14	1159	207.3	1	3.475	1159	148	1	8.22	1159	66.74
Query ls2	9	154.3	333	285.4	3	425.1	319	119.4	12	7.86	333	92.38
Query ls3	5	27.61	9054	4512	5	373.1	9054	4616	12	9.2	9054	4390
Query ls4	7	17.33	3	142.6	7	13.26	3	115.1	7	6.58	0	0.02
Query ls5	8	24.39	393	1624	6	16.89	393	1556	10	7.5	0	0.02
Query ls6	7	23.3	28	265.8	5	17.19	28	285.7	9	7.54	28	98.64
Query ls7	6	21.34	144.5	1200	5	5.1	0	1172	6	7.64	2613	923.1

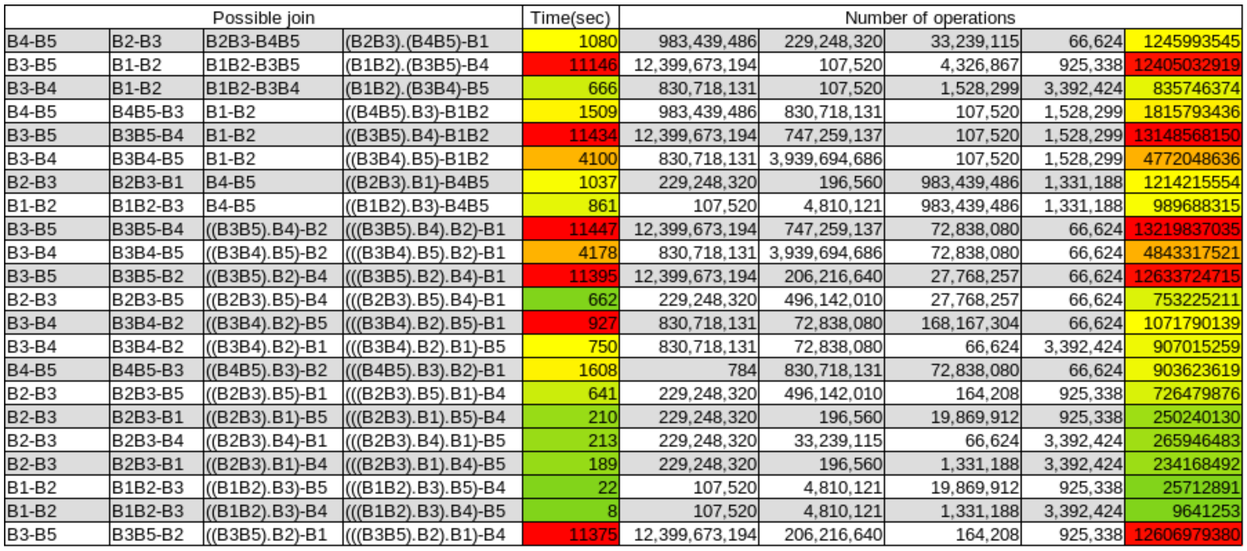
Figure 9: Queries results. #S is the number of sources selected. TS is the time for source selection. #R is the number of results. TQ is the time of the query evaluation and source selection for HiBISCus and PPinSS and only query evaluation for Grafeque.

We can see in Figure 9 that Grafeque does not get results for LS4 and LS5 even though it got the right sources. The reason has not yet been found.

For PPinSS the lack of results for LS7 may be an error in the joining process because of the keyword *FILTER* inside the query. Without it, the query sends the correct results.

5.2 LS6

This variation can be explained by the joining order of the query. HiBISCus joins every statements pattern one after the other, but one joining operation asks for more than 25,000 results to be passed on the next part of the query. A simple swap of two triple patterns can reduce it to 164 results because the first patterns return fewer results than the second. That trick is not written on the scientific paper of HiBISCus and it is not how the query was first designed. PPinSS does not have this particular problem thanks to its source selection. It scans the variables used in the three triple patterns and gathers them in an exclusive group statement assigned to a single endpoint.



Possible join				Time(sec)	Number of operations				
B4-B5	B2-B3	B2B3-B4B5	(B2B3).(B4B5)-B1	1080	983,439,486	229,248,320	33,239,115	66,624	1245993545
B3-B5	B1-B2	B1B2-B3B5	(B1B2).(B3B5)-B4	11146	12,399,673,194	107,520	4,326,867	925,338	12405032919
B3-B4	B1-B2	B1B2-B3B4	(B1B2).(B3B4)-B5	666	830,718,131	107,520	1,528,299	3,392,424	835746374
B4-B5	B4B5-B3	B1-B2	((B4B5).B3)-B1B2	1509	983,439,486	830,718,131	107,520	1,528,299	1815793436
B3-B5	B3B5-B4	B1-B2	((B3B5).B4)-B1B2	11434	12,399,673,194	747,259,137	107,520	1,528,299	13148568150
B3-B4	B3B4-B5	B1-B2	((B3B4).B5)-B1B2	4100	830,718,131	3,939,694,686	107,520	1,528,299	4772048636
B2-B3	B2B3-B1	B4-B5	((B2B3).B1)-B4B5	1037	229,248,320	196,560	983,439,486	1,331,188	1214215554
B1-B2	B1B2-B3	B4-B5	((B1B2).B3)-B4B5	861	107,520	4,810,121	983,439,486	1,331,188	989688315
B3-B5	B3B5-B4	((B3B5).B4)-B2	(((B3B5).B4).B2)-B1	11447	12,399,673,194	747,259,137	72,838,080	66,624	13219837035
B3-B4	B3B4-B5	((B3B4).B5)-B2	(((B3B4).B5).B2)-B1	4178	830,718,131	3,939,694,686	72,838,080	66,624	4843317521
B3-B5	B3B5-B2	((B3B5).B2)-B4	(((B3B5).B2).B4)-B1	11395	12,399,673,194	206,216,640	27,768,257	66,624	12633724715
B2-B3	B2B3-B5	((B2B3).B5)-B4	(((B2B3).B5).B4)-B1	662	229,248,320	496,142,010	27,768,257	66,624	753225211
B3-B4	B3B4-B2	((B3B4).B2)-B5	(((B3B4).B2).B5)-B1	927	830,718,131	72,838,080	168,167,304	66,624	1071790139
B3-B4	B3B4-B2	((B3B4).B2)-B1	(((B3B4).B2).B1)-B5	750	830,718,131	72,838,080	66,624	3,392,424	907015259
B4-B5	B4B5-B3	((B4B5).B3)-B2	(((B4B5).B3).B2)-B1	1608	784	830,718,131	72,838,080	66,624	903623619
B2-B3	B2B3-B5	((B2B3).B5)-B1	(((B2B3).B5).B1)-B4	641	229,248,320	496,142,010	164,208	925,338	726479876
B2-B3	B2B3-B1	((B2B3).B1)-B5	(((B2B3).B1).B5)-B4	210	229,248,320	196,560	19,869,912	925,338	250240130
B2-B3	B2B3-B4	((B2B3).B4)-B1	(((B2B3).B4).B1)-B5	213	229,248,320	33,239,115	66,624	3,392,424	265946483
B2-B3	B2B3-B1	((B2B3).B1)-B4	(((B2B3).B1).B4)-B5	189	229,248,320	196,560	1,331,188	3,392,424	234168492
B1-B2	B1B2-B3	((B1B2).B3)-B5	(((B1B2).B3).B5)-B4	22	107,520	4,810,121	19,869,912	925,338	25712891
B1-B2	B1B2-B3	((B1B2).B3)-B4	(((B1B2).B3).B4)-B5	8	107,520	4,810,121	1,331,188	3,392,424	9641253
B3-B5	B3B5-B2	((B3B5).B2)-B1	(((B3B5).B2).B1)-B4	11375	12,399,673,194	206,216,640	164,208	925,338	12606979380

Figure 10: Results of the Join searching. From left to right: The 4 steps of the joining, the full joining time, the number of operations for each steps of the joining and the total number of operations.

The tests with ASP were promising at first, but the limitation over arrays with the ASP solver used and the complexity of the rules applied for our operations leads us to simply use Python.

To represent a statement, I have created a class named **Bloc** which contains an array with 0 and 1 that adds up when two statements are joined, a name and the number of result that statement should return. The name is needed to keep tracks of the join order and the number of results will shows us the worst number of operations which is linked to the query evaluation time.

Figure 10 shows that there is one order for which the join operation will take a minimum time. Even if the times displayed is not the real ones, it gives us an idea of the worst cases and demonstrate the importance of the query planning.

To confirm the engine capability, tests were realised with the other queries and the other available domains: Crossed domain and Linked data.

```python
B1 = Bloc([1,0,0,0],"B1",48)
B2 = Bloc([1,1,0,0],"B2",2240)
B3 = Bloc([0,1,1,0],"B3",102343)
B4 = Bloc([0,0,1,0],"B4",8117)
B5 = Bloc([0,0,1,1],"B5",121158)
```

Listing 2: Initialisation of the statements for the python's scripts.

5.3 Scoring changes

To stop HiBISCus from hitting 120 seconds for LS6, I have searched for a way where its statements pattern have a better joining order. Initially, HiBISCus increase the score of the statement when there are no common variables with any statements previously analysed. The statement with the smaller score is then placed in the ordered list for joining. If a common variable were found, no action was taken. Therefore I have added a clause to decrease the score when has a common variable is encountered. This simple change managed to reorganise the query joining order and improve the time on LS6.

5.4 Corese

The implementation of PPinSS may be delayed, researches about how to do it continues. As it uses differents indexes, the source selection method will changes. We will use of the .jar files to launch the life science queries for the benchmarks without PPinSS.

6 Conclusion

Most of the internship was trial and error, trying to find the option that can make a difference when launching benchmarks. The lack of documentation made the development much slower but with the migration to Corese, it will be easier for future works. The benchmarks shows that PPinSS is slightly better that HiBISCus at evaluating queries for the Life Science domain. It also allows the discovery of an improvement to HiBISCus with its score modification. Unfortunately, in its actual state, Grafeque is not an improvement but its time for sources selection is a good sign. Finally, all the debugging, all the time spent on searching in the code and the documentations being not always relevant indicates that a migration to Corese, an engine with a live working team, will be helpful.

References

[1] Alberto Anguita et al. 'NCBI2RDF: Enabling Full RDF-Based Access to NCBI Databases'. In: *BioMed research international* 2013 (2013), p. 983805.

[2] Olivier Corby, Rose Dieng-Kuntz and Catherine Faron-Zucker. 'Querying the Semantic Web with Corese Search Engine'. In: *Proc. of the 16th European Conference on Artificial Intelligence (ECAI 2004)*. Ed. by R. Lopez de Mantaras and editors L. Saitta. 2004, pp. 705–709.

[3] Alex Kalderimis et al. 'InterMine: extensive web services for modern biology'. In: *Nucleic acids research* 42.Web Server issue (2014), W468–W472.

[4] Muhammad Saleem, Ali Hasnain and Axel-Cyrille Ngonga Ngomo. 'LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation'. In: *Journal of Web Semantics* 48 (2018), pp. 85–125. ISSN: 1570-8268. DOI: `https://doi.org/10.1016/j.websem.2017.12.005`. URL: `http://www.sciencedirect.com/science/article/pii/S1570826817300719`.

[5] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. 'HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation'. In: *Proceedings of the The European Semantic Web Conference ESWC2014*. Vol. 8465. Lecture Notes in Computer Science. 2014.

[6] Muhammad Saleem et al. 'How Representative is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks'. In: *Proceedings of ACM Conference (TheWebConf WWW19)*. 2019. DOI: `10.1145/3308558.3313556`.

[7] Michael Schmidt et al. 'FedBench: A Benchmark Suite for Federated Semantic Data Query Processing'. In: *The Semantic Web – ISWC 2011*. Ed. by Lora Aroyo et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 585–600. ISBN: 978-3-642-25073-6.

[8] Andreas Schwarte et al. 'FedX: Optimization Techniques for Federated Query Processing on Linked Data'. In: *International Semantic Web Conference*. 2011.

[9] Damian Smedley et al. 'The BioMart community portal: an innovative alternative to large, centralized data repositories'. In: *Nucleic Acids Research* 43.W1 (Apr. 2015), W589–W598. ISSN: 0305-1048. DOI: `10.1093/nar/gkv350`. eprint: `http://oup.prod.sis.lan/nar/article-pdf/43/W1/W589/7476086/gkv350.pdf`. URL: `https://doi.org/10.1093/nar/gkv350`.

List of Figures

List of Tables

Master de Bioinformatique de l'Université de Rennes 1, années 2018-2019.
ABEL Antoine

Résumé du stage

Depuis plusieurs années nous observons l'augmentation des données dans domaine de la vie et la multiplication des bases de données. Le chevauchement des données entre ces bases pousse les scientifiques à développer de nouvelles manières d'évaluer les requêtes pour gagner du temps et recouvrir toutes les bases de données. Pouvoir envoyer une requête sur plusieurs bases de données est la base de la fédération des requêtes. De nombreux moteurs de fédération ont été développé, nous allons travailler avec: FedX, HiBISCus (une modification de FedX), PPinSS (une modification de HiBISCus), Grafeque (une modification de HiBISCus). L'objectif du stage était de reprendre les tests avec HiBISCus et PPinSS et ajouter Grafeque aux outils utilisés. Les benchmarks déjà effectués ont montré qu'HiBISCus n'évaluait pas correctement LS6, nous avons donc étudié cette requêtes et son traitement par les différents outils. L'exploration du code source étant assez fastidieux, des scripts en python ont servis à simuler les jointures en changeant l'ordre des triplets. Cela nous a permis de trouver une modification du calcul du score pour l'étape de planification de la requête. En outre, le développement de FedX n'étant plus maintenu en raison du rachat de l'entreprise qui le développait, une migration vers le moteur Corese est donc préparée.

Mots clés: Web sémantique, SPARQL, RDF, requêtes fédérés.

Internship abstract

For several years we have observed the increase of data in the life science domain and the multiplication of databases. Overlapping data between these databases is forcing scientists to develop new ways to evaluate queries to save time and cover all databases. Being able to send a query on several databases is the basis of query federation but many methods exists. Many federation engines have been developed, we will work on a few of them: FedX, HiBISCus (a modification of FedX), PPinSS (a modification of HiBISCus), Grafeque (a modification of HiBISCus). The objective of the internship was to continue the tests with HiBISCus and PPinSS and add Grafeque to the tools used. The benchmarks already carried out showed that HiBISCus did not evaluate LS6 correctly, so we studied this query and its evaluation by the different tools. Reading through the entire source code was quite tedious, python scripts were used to simulate the joints by changing the order of triples. This allowed us to find a modification of the scoring method for the query planning. In addition, FedX is no longer maintained in development due to the company that developed it being bought, a migration to the Corese engine is prepared.

Key words: Semantic Web, SPARQL, RDF, federated queries.