Parallel Data Loading during Querying Deep Web and Linked Open Data with SPARQL

Pauline Folz[1,2], Gabriela Montoya[1,3], Hala Skaf-Molli[1], Pascal Molli[1], and Maria-Esther Vidal[4]

[1] LINA– Nantes University, France
{pauline.folz,gabriela.montoya,hala.skaf,pascal.molli}@univ-nantes.fr
[2] Nantes Métropole - Direction Recherche, Innovation et Enseignement Supérieur, France
[3] Unit UMR6241 of the Centre National de la Recherche Scientifique (CNRS), France
[4] Universidad Simón Bolívar, Venezuela
{mvidal}@ldc.usb.ve

Abstract. Web integration systems are able to provide transparent and uniform access to heterogeneous Web data sources by integrating views of Linked Data, Web Service results, or data extracted from the Deep Web. However, given the potential large number of views, query engines of Web integration systems have to implement execution techniques able to scale up to real-world scenarios and efficiently execute queries. We tackle the problem of SPARQL query processing against RDF views, and propose a non-blocking query execution strategy that incrementally accesses and merges the views relevant to a SPARQL query in a parallel fashion. The proposed strategy is implemented on top of Jena 2.7.4, and empirically compared with SemLAV, a sequential SPARQL query engine on RDF views. Results suggest that our approach outperforms SemLAV in terms of the number of answers produced per unit of time.

1 Introduction

Linked Open Data initiatives have motivated the integration of a large number of RDF datasets into the Linking Open Data (LOD) cloud [4]. Different Web-based interfaces are available to access these publicly accessible Linked Data sets, *e.g.*, SPARQL endpoints and Linked Data fragments [17]. However, the Deep Web which has around 500 times the size of the Surface Web [11, 10] has not been integrated as part of LOD cloud. Performing SPARQL queries without considering the Deep Web can potentially deliver incomplete results. For example, the execution of the SPARQL query: *Which members of the Semantic Web community are interested in* Dalai Lama*,* Barack Obama*, or* Rihanna*?* (cf. Figure 2) without the integration of the Deep Web will provide no answers [8]. Nevertheless, if data from social networks such as Twitter, Facebook, or LinkedIn were considered, the query execution could return some answers.

Two main approaches exist for data integration: data warehousing, and the virtual mediators [7]. Semantic data-warehouses such as Virtuoso with the Sponger

feature [1] allow for the implementation of wrappers able to create RDF data from unsemantified data sources, *e.g.*, Web services, CSV files; but this approach may suffer from the *freshness problem* [2], *i.e.*, data may become stale when data sources are updated.

On the other hand, a mediator relies on a global schema to provide a uniform interface for accessing the data sources. Global-As-View (GAV) and Local-As-View (LAV), are the main paradigms for mapping data sources and the global schema. In GAV mediators, entities of the global schema are described using views over the data sources, but including or updating data sources may require the modification of a large number of views [16]. Whereas, in LAV mediators, the sources are described as views over the global schema, and adding new data sources can be easily done [16]. Despite of its expressiveness and flexibility, LAV query re-writting is in general intractable, *i.e.*, NP-complete for conjunctive queries [3]. State-of-the-art LAV query rewriters efficiently solve some families of the query rewriting problem [3, 12]; nevertheless, they may not equally perform on SPARQL queries [13]. Recently, SemLAV [13], the first scalable LAV-based approach for SPARQL query processing, was proposed. Instead of enumerating the query rewritings of a SPARQL query, SemLAV selects the most relevant LAV views, accesses the selected views according to their relevance, and materializes the downloaded data into an *integrated RDF graph*. Then, the SPARQL query is executed against the integrated RDF graph.

SemLAV provides a new paradigm to execute SPARQL queries against LAV views, but because relevant views are loaded sequentially, SemLAV may get blocked loading large views. In the worst case, if the first loaded view is huge and it does not provide relevant data for the query answer, SemLAV will be blocked without producing any answer. Following a sequential view loading strategy may reduce the number of answer produced per unit of time, *i.e.*, throughput, and the time for first answer. Loading several views in parallel may overcome these limitations. However, a parallel view loading strategy will introduce the problem of concurrent writing on the integrated RDF graph. In this paper, we propose a non-blocking query execution strategy to integrate the data from the relevant views into the integrated RDF graph in a parallel fashion. We implement the proposed non-blocking strategy on the top of Jena 2.7.4; we name this new SPARQL query engine *parallel SemLAV*. Further, an empirical evaluation is conducted to study the new parallel strategy with respect to SemLAV. The Berlin Benchmark [5] and queries and views designed by Castillo-Espinola [6] are used to evaluate both query engines. Results suggest that the parallel SemLAV outperforms SemLAV with respect to answers produced per time unit.

The paper is organized as follows. Section 2 describes background and motivation. Section 3 presents strategies for integrating relevant views into the integrated RDF graph in a parallel fashion. Section 4 reports our experimental results. Finally, conclusions and future work are outlined in Section 5.

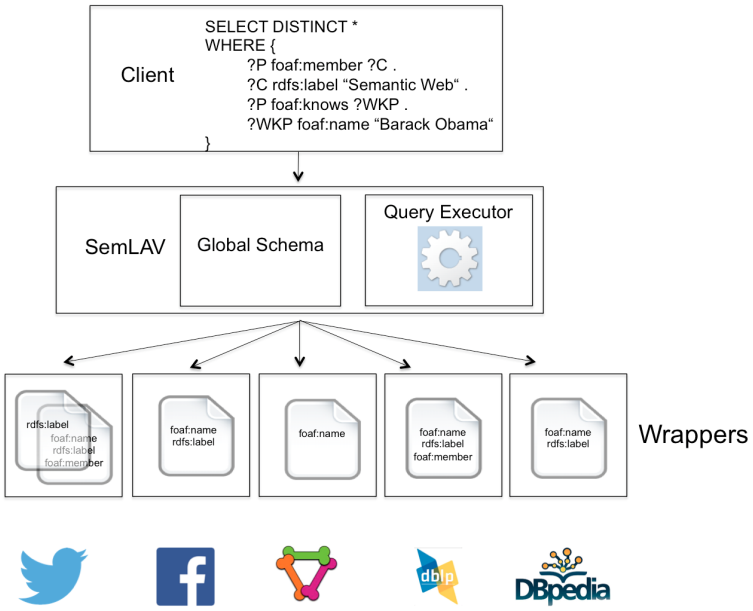

Fig. 1: SemLAV a mediator and wrapper architecture

2 Background and Motivation

SemLAV follows a mediator and wrapper architecture [18] where data from the sources are virtually integrated by SemLAV in a global schema composed by several RDF vocabularies, as shown in Figure 1. Sources are described by LAV views and can be heterogeneous, *e.g.*, from the Deep Web, RDF data sets, or relational tables. SPARQL queries are expressed in terms of the global schema and posed against the SemLAV mediator. A wrapper is specific for a data source, and retrieves data on demand; the retrieved data are transformed to match the global schema. Wrappers can be generated by tools like Karma [15] or OPAL [9]. The global schema is the interface between users and the data sources.

2.1 SemLAV Overview

Given a query and a set of views, SemLAV computes a ranked set of relevant views for answering the query, no *statistics* are used to rank the views. Relevant views are ranked based on the number of triple patterns of the original query that each view covers [13]. Views are materialized by calling the wrappers, and each time a new view is fully materialized, the original query is executed.

The benefits of SemLAV are illustrated in the following example [8]. Suppose SemLAV global schema comprises different RDF vocabularies, *e.g.*, foaf [5] and

[5] `http://xmlns.com/foaf/0.1/`

```
prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
prefix foaf:    <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT *
WHERE {
  ?P foaf:member ?C .
  ?C rdfs:label "Semantic Web" .
  ?P foaf:knows ?WKP .
  ?WKP foaf:name ?N .
   FILTER (?N="Dalai Lama" || ?N="Barack Obama" || ?N="Rihanna")
}
```

Fig. 2: A SPARQL query over Deep Web and Linked Data

rdfs [6]. Figure 2 presents a SPARQL query expressed using the global schema. Views are expressed as conjunctive queries, where RDF predicates are represented by binary predicates, *e.g.*, label(C,L) corresponds to *?C rdf:label ?L* and *?P foaf:name ?N* is expressed as name(P,N). Listing 1 defines five LAV views. Triple patterns in the query are also seen as binary predicates and BGPs are represented as conjunctive queries; the running SPARQL query is composed of four subgoals on the predicates: member(P,C), label(C, "Semantic Web"), knows(P,WKP), and name(WKP,N). The filter expression is modeled as a disjunction of atomic expressions on the equality comparison operator.

Listing 1: Views s1-s5 for Query Q

```
v1(P,A,I,C,L):-made(P,A),affiliation(P,I),member(P,C),label(C,L)
v2(A,T,P,N,C):-title(A,T),made(P,A),name(P,N),member(P,C)
v3(P,N,R,M):-name(P,N),name(R,M),knows(P,R)
v4(P,N,G,R,C):-name(P,N),gender(P,G),knows(P,R),member(P,C)
v5(P,N,R,C,L):-name(P,N),knows(P,R),member(P,C),label(C,L)
```

Given a subgoal *sg* of a conjunctive query, *e.g.*, label(C,"Semantic Web"), a view *v* is relevant for *sg*, if *sg* is part of the body of *v*, *e.g.*, v1(P,A,I,C,L) and v5(P,N,R,C,L) are relevant for label(C,"Semantic Web"). Table 1a presents the set of relevant views for each query subgoal of query in Figure 2.

SemLAV sorts relevant views according to the number of the subgoals of the query that the view defines, *e.g.*, view *v*5 is sorted first since it defines all the subgoals. Table 1b represents the sorted relevant views for query in Figure 2.

SemLAV identifies and ranks the relevant views of a query, and executes the query over the data collected from the relevant views. Different strategies can be followed to contact the views and load the data. For example, following a blocking strategy, views are contacted one by one in order, and a view is not contacted until all the data from the previous contacted view have been downloaded completely. This is the strategy followed by SemLAV, which is illustrated in the Figure 3a, we can see that this strategy can be blocking if the first view is huge. While the view v5 is loading we are not able to perform the query. This blocking issue can have a negative impact on the performance of the query en-

[6] `"http://www.w3.org/2000/01/rdf-schema`

Table 1: Relevant views of query Q (cf. Figure 2), and views from Listing 1.

(a) Unsorted relevant views

member(P, C)	label(C, L)	knows(P, WKP)	name(WKP, N)
v1(P,A,I,C,L)	v1(P,A,I,C,L)	v3(P,N,R,M)	v2(A,T,P,N,C)
v2(A,T,P,N,C)	v5(P,N,R,C,L)	v4(P,N,G,R,C)	v3(P,N,R,M)
v4(P,N,G,R,C)		v5(P,N,R,C,L	v4(P,N,G,R,C)
v5(P,N,R,C,L)			v5(P,N,R,C,L)

(b) Sorted relevant views

member(P, C)	label(C, L)	knows(P, WKP)	name(WKP, N)
v5(P,N,R,C,L)	v5(P,N,R,C,L)	v5(P,N,R,C,L)	v5(P,N,R,C,L)
v4(P,N,G,R,C)	v1(P,A,I,C,L)	v4(P,N,G,R,C)	v4(P,N,G,R,C)
v1(P,A,I,C,L)		v3(P,N,R,M)	v2(A,T,P,N,C)
v2(A,T,P,N,C)			v3(P,N,R,M)

gine if the performance is measured in terms of the number of answers produced per unit of time, *i.e.*, throughput.

To illustrate this problem, consider Figure 3a, where v5 is loaded first. Even if v5 covers all the query subgoals, loading v5 first reduces the throughput, because v5 is the biggest view and does not contribute to the result. On the other hand, loading both v1 and v4, which together cover all the subgoals takes less time and may produce query answers. If relevant views were loaded in parallel following a non-blocking strategy, this situation would not affect the query engine performance. This solution is illustrated in Figure 3b, where there are five threads and each of them loads one of the first five top ranked views at the time; views are allocated in *different* threads. Time to load v5 is greater than the time required to load v4 and v1 in parallel. Additionally, v4 and v1 cover all the subgoals of our running query; thus, answers are produced before loading v5 completely.

We propose a non-blocking strategy for executing SPARQL queries against views. Like SemLAV, this approach does not rely on statistics to rank and select the relevant views. The proposed strategy prevents the query engine from getting blocked until all the data are retrieved from the relevant views.

3 Our Approach

A non-blocking strategy to access the views in a parallel fashion is defined. Although this strategy improves the performance of a query engine, loading the retrieved data into the integrated RDF graph in parallel, may generate concurrency problems, *i.e.*, many processes may simultaneously add data to the integrated RDF graph. So, we define a new concurrent model for RDF, and we propose a non-blocking query execution strategy able to adapt query execution to different criteria, *e.g.*, a query is executed after a certain number of triples

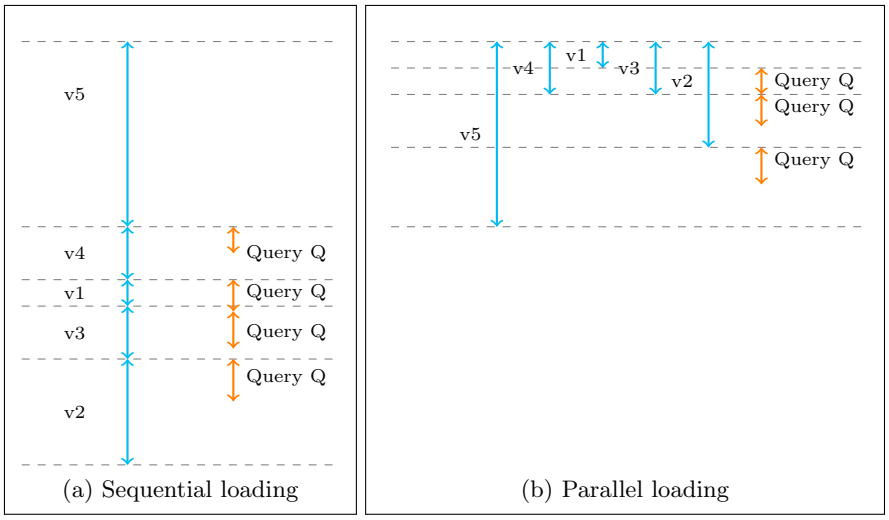

Fig. 3: Views loading and Query execution. For sequential loading just one thread is used, while for parallel loading five threads are used

are loaded into the integrated RDF graph. We implement the concurrency model and the non-blocking query execution strategy on top of Jena 2.7.4 [7] .

3.1 A Concurrency Model for the Integrated RDF Graph

Regarding our approach, we need a model that can handle concurrent insertions. However, RDF stores like Jena do not handle concurrent insertions, they are only able to favor one type of operation, *e.g.*, reads or insertions. This strategy is implemented thanks to locks, but read and insert locks are mutually exclusive, *i.e.*, they cannot be simultaneously activated. Existing RDF stores assume that there are more readers than writers and follow the multiple-readers/single-writer strategy (MRSW)[8]. According to MRSW, many readers may read simultaneously, while a writer must have exclusive access. MRSW assumes writers have the priority to keep data up-to-date. Nevertheless, in our proposed approach, data insertions are going to be more frequent than data reads. A reader is the query engine that accesses the integrated RDF graph during query execution, while writers are the wrappers of the relevant views which load the data into the integrated RDF graph. The query engine cannot execute the query more often than loading views into the integrated RDF graph, because executing the query is expensive, and doing so too often may lead to performance degradation.

In other words, our proposed approach prioritizes read operations over insertions, *i.e.*, a single-reader/multiple-writers strategy (SRMW) [14] is followed to

[7] http://jena.apache.org/
[8] https://jena.apache.org/documentation/notes/concurrency-howto.html

manage concurrency on the integrated RDF graph. So the reader, *e.g.*, a query execution engine, will have a higher priority rather than a writer, *e.g.*, a wrapper loading a view. Additionally, two insert locks cannot be activated at the same time due to the specification of the integrated RDF model. However, the query engine divides each view into blocks of n triples to allow for the loading of portions of several views at the same time. A lock is requested before starting a block loading, and it is released after n triples have been loaded completely. In our example, the first block of v5 is loaded, then the first block of v4, and to load the second block of v5, it may be necessary to wait until all the first blocks of the currently loading views are already loaded. However, this order may fluctuate depending on the system time allocation among the threads.

3.2 A Non-Blocking Strategy for SPARQL Query Execution

We implement a non-blocking strategy that is able to execute a query according to the following criteria; the selection of the criteria can be either configured or provided by the user during query execution.

- View dependent: the reader is woken up after a new view is loaded; thus, if v is a new loaded view, then the query engine will re-execute the query against the integrated RDF graph. If enough data is loaded into the integrated RDF graph from v, then the query engine will be able to generate new results when it is executed. This criterion is also implemented by SemLAV.
- Time dependent: the reader is woken up after a period of time t, *i.e.*, if t is n milliseconds, the query engine will re-execute the query against the RDF graph every n milliseconds. If enough data is loaded into the integrated RDF graph during the period t, the query engine will be able to generate new results. But, the concurrency model prioritizes the reader over writers; thus, if the writers are stopped and not able to load enough data into the integrated RDF graph, the query will be inefficiently executed.
- Data dependent: the reader is woken up after a certain number n of triples are inserted into the integrated RDF graph by the writers; thus, the query engine will re-execute the query against the RDF graph whenever n new triples are integrated. If the n new triples contribute to the results, then the query engine will be able to generate new answers when it is executed.
- Two-phase execution: the reader is woken up either after a period of time t or a certain number n of triples are inserted into the integrated RDF graph by the writers. In the first phase, the reader performs ASK queries to check if new results can be produced, if the answer is `true`, the second phase is launched. The second phase strategy will directly execute the query, then the reader will be woken up either after a period of time t or a certain number n of new triples have been inserted into the integrated RDF graph.

4 Experimental Evaluation

The Berlin SPARQL Benchmark (BSBM) [5], and queries and views proposed by Espinola-Castillo [6] are used to compare the performance of parallel SemLAV

with respect to SemLAV. Our goal is to reproduce the experiments reported by Montoya et al. [13]; therefore, we used the Berlin Benchmark dataset composed of 10,000,736 triples using a scale factor of 28,211 products, 16 out of 18 queries, and nine out of the ten defined views proposed by Espinola-Castillo [6]. In SemLAV experiments, some queries and views were not considered because they included constants and some of the evaluated rewriters only process queries with variables. Five additional views were defined to cover all the predicates in the evaluated queries, *i.e.*, 14 views were evaluated. Furthermore, 476 views were produced by horizontally partitioning each original view into 34 parts, such that each part produces 1/34 of the answers given by the original view.

Queries and views are described in Tables 2a and 2b. The size of the complete answer is computed by including all the views into the Jena RDF triple store and by executing the queries against this centralized RDF dataset. The Jena 2.7.4 library with main memory setup is used to store and query the integrated RDF graphs. We executed parallel SemLAV with a timeout of 10 minutes.

Table 2: Queries and their answer size, number of subgoals, and views size, source [13]

(a) Query information

Query	Answer Size	# Subgoals
Q1	6.68E+07	5
Q2	5.99E+05	12
Q4	2.87E+02	2
Q5	5.64E+05	4
Q6	1.97E+05	3
Q8	5.64E+05	3
Q9	2.82E+04	1
Q10	2.99E+06	3
Q11	2.99E+06	2
Q12	5.99E+05	4
Q13	5.99E+05	2
Q14	5.64E+05	3
Q15	2.82E+05	5
Q16	2.82E+05	3
Q17	1.97E+05	2
Q18	5.64E+05	4

(b) Views size

Views	Size
V1-V34	201,250
V35-V68	153,523
V69-V102	53,370
V103-V136	26,572
V137-V170	5,402
V171-V204	66,047
V205-V238	40,146
V239-V272	113,756
V273-V306	24,891
V307-V340	11,594
V341-V374	5,402
V375-V408	5,402
V409-V442	78,594
V443-V476	99,237
V477-V510	1,087,281

Experiments are also run on the same platform than SemLAV experiments, *i.e.*, on a Linux server with 128 GB of memory, 124 processors where 20 GB of RAM are allocated for the experiments. Wrappers are implemented for each view and to load data from RDF files, *i.e.*, 476 wrappers are available.

4.1 Implementation

We use critical section and lock to implement the single-reader/multiple-writers SRMW concurrency model in Jena 2.7.4. The number of threads impacts the SPARQL engine performance; thus, we consider this number as one of the independent parameters of our study.

Table 3: Result of SemLAV and parallel SemLAV on BSBM using the View Dependent Criterion with 20 threads (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	SemLAV				parallel SemLAV			
Query	TT	TFA	Throughput	#EQ	TT	TFA	Throughput	#EQ
1	606,697	6,370	37.3501	15	604,254	30,481	**88.7036**	7
2	600,656	260,333	0.9823	66	605,729	**72,515**	**0.9883**	16
4	660,938	104,501	0.0004	47	359,635	288,558	**0.0008**	20
5	632,809	116,037	0.8916	28	457,269	257,097	**1.2339**	14
6	625,173	43,306	0.1892	24	273,662	211,313	**0.7203**	9
8	627,612	5,393	0.8990	42	318,475	24,877	**1.7716**	7
9	5,107	1,235	5.5240	18	2,453	1,839	**11.5006**	3
10	607,841	9,810	4.9243	44	439,562	32,438	**6.8094**	15
11	601,042	8,352	4.9800	43	105,684	31,660	**28.3219**	6
12	609,509	5,784	0.9822	121	372,481	15,542	**1.6072**	16
13	671,893	183,844	0.8910	124	392,147	**41,799**	**1.5266**	20
14	636,387	29,201	0.5419	24	333,754	201,864	**1.6905**	14
15	645,172	2,911	0.4373	37	388,061	20,016	**0.7270**	18
16	648,826	2,531	0.4348	46	306,694	15,390	**0.9198**	7
17	644,090	1,504	0.3060	32	278,330	5,894	**0.7082**	7
18	651,094	> 600,000	0.0000	12	509,646	**259,598**	**1.1071**	13

4.2 Impact of the Non-Blocking Query Execution Criteria

The goal of the experiment is to study the impact of the non-blocking query execution criteria on the query engine performance. We hypothesize that parallel SemLAV will outperform SemLAV in terms of throughput and time for the first answer. We measure the following metrics: *i*) total time (TT) in milliseconds; *ii*) time for first answer (TFA) in milliseconds; *iii*) throughput (answer/millisecond); and *iv*) number of times the original query is executed (#EQ).

We evaluate parallel SemLAV for the non-blocking query execution criteria defined in Section 3 with different number of threads, *i.e.*, the number of writers and the configuration of the non-blocking query execution strategy. We use setups with different number of threads 5, 10, and 20. Results suggest that 20 threads is the best number for writers. All the results are available at the project web site `https://sites.google.com/site/semanticlav`.

Table 4: Result of SemLAV and parallel SemLAV on BSBM using the View Dependent Criterion with 5 threads (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	SemLAV				parallel SemLAV			
Query	TT	TFA	Throughput	#EQ	TT	TFA	Throughput	#EQ
1	606,697	6,370	37.3501	15	601,221	13,235	35.3143	8
2	600,656	260,333	0.9823	66	646,416	**87,166**	0.9261	25
4	660,938	104,501	0.0004	47	406,008	**91,383**	**0.0007**	50
5	632,809	116,037	0.8916	28	601,055	**88,752**	**0.9387**	29
6	625,173	43,306	0.1892	24	317,213	61,451	**0.6214**	25
8	627,612	5,393	0.8990	42	410,306	7,202	**1.3751**	12
9	5,107	1,235	5.5240	18	2,687	**987**	**10.4991**	4
10	607,841	9,810	4.9243	44	631,503	11,438	4.7398	31
11	601,042	8,352	4.9800	43	300,244	9,879	**9.9691**	13
12	609,509	5,784	0.9822	121	508,837	9,048	**1.1765**	37
13	671,893	183,844	0.8910	124	532,783	**54,758**	**1.1236**	40
14	636,387	29,201	0.5419	24	463,967	62,251	**1.2161**	28
15	645,172	2,911	0.4373	37	600,885	8,390	**0.4695**	36
16	648,826	2,531	0.4348	46	462,310	4,820	**0.6102**	12
17	644,090	1,504	0.3060	32	311,895	2,533	**0.6320**	17
18	651,094	> 600,000	0.0000	12	600,102	**264,917**	**0.9402**	37

Table 5: Result of BSBM over SemLAV and parallel SemLAV using the View Dependent Criterion with 10 threads (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	SemLAV				parallel SemLAV			
Query	TT	TFA	Throughput	#EQ	TT	TFA	Throughput	#EQ
1	606,697	6,370	37.3501	15	602,508	17,819	**41.3346**	8
2	600,656	260,333	0.9823	66	608,174	**70,504**	**0.9843**	25
4	660,938	104,501	0.0004	47	332,060	127,329	**0.0009**	50
5	632,809	116,037	0.8916	28	505,404	128,097	**1.1164**	29
6	625,173	43,306	0.1892	24	272,134	98,736	**0.7243**	25
8	627,612	5,393	0.8990	42	323,938	11,994	**1.7418**	12
9	5,107	1,235	5.5240	18	2,479	1,489	**11.3800**	4
10	607,841	9,810	4.9243	44	601,192	17,710	**4.9787**	31
11	601,042	8,352	4.9800	43	168,108	16,997	**17.8051**	13
12	609,509	5,784	0.9822	121	390,470	11,081	**1.5331**	37
13	671,893	183,844	0.8910	124	409,106	**39,892**	**1.4633**	40
14	636,387	29,201	0.5419	24	326,745	91,049	**1.7268**	28
15	645,172	2,911	0.4373	37	496,533	11,419	**0.5682**	36
16	648,826	2,531	0.4348	46	321,641	9,723	**0.8771**	12
17	644,090	1,504	0.3060	32	252,595	3,643	**0.7803**	17
18	651,094	> 600,000	0.0000	12	600,785	**221,434**	**0.9391**	37

The View Dependent Criterion: The thread which executes the query is woken up when a new view is loaded. Table 3 shows the result of SemLAV and parallel SemLAV using the view strategy, *i.e.*, re-execute the query after a new view is loaded. Parallel SemLAV outperforms SemLAV in terms of throughput and total execution time. But surprisingly, the time for first answer is increased, for all queries except queries 2, 13, and 18; for these queries the time for the first answer is at most half of the SemLAV time. In most queries the time for first answer is increased because the number of times the original query is executed (#EQ) in parallel SemLAV is less than in SemLAV; furthermore, parallel SemLAV breaks the views ranking established by SemLAV, *i.e.*, SemLAV starts by loading the view ranked in first place and executes the query. However, parallel SemLAV loads views in parallel, and the query is re-executed when a new view is loaded, which is not necessarily the first ranked view by SemLAV. In setups with 5 and 10 threads, the time for first answer is better than for 20 threads, but the throughput is lower as shown in Tables 4 and 5.

Table 6: Result of BSBM over SemLAV and parallel SemLAV using the Time Dependent Criterion with 20 threads; queries are executed every 500 msecs (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	SemLAV				parallel SemLAV			
Query	TT	TFA	Throughput	#EQ	TT	TFA	Throughput	#EQ
1	606,697	6,370	37.3501	15	604,465	28,033	**67.3762**	16
2	600,656	260,333	0.9823	66	602,164	**73,074**	**0.9941**	17
4	660,938	104,501	0.0004	47	370,372	262,367	**0.0008**	102
5	632,809	116,037	0.8916	28	465,548	254,253	**1.2119**	27
6	625,173	43,306	0.1892	24	266,556	184,145	**0.7395**	83
8	627,612	5,393	0.8990	42	334,311	18,176	**1.6877**	17
9	5,107	1,235	5.5240	18	2,343	1,772	**12.0405**	4
10	607,841	9,810	4.9243	44	460,109	31,589	**6.5054**	28
11	601,042	8,352	4.9800	43	114,680	23,886	**26.1002**	19
12	609,509	5,784	0.9822	121	357,470	15,481	**1.6746**	22
13	671,893	183,844	0.8910	124	363,735	**41,237**	**1.6458**	24
14	636,387	29,201	0.5419	24	305,013	161,527	**1.8498**	94
15	645,172	2,911	0.4373	37	412,315	20,019	**0.6842**	23
16	648,826	2,531	0.4348	46	302,547	12,336	**0.9325**	14
17	644,090	1,504	0.3060	32	235,062	5,910	**0.8386**	21
18	651,094	> 600,000	0.0000	12	509,085	**276,665**	**1.1083**	99

The Time Dependent Criterion: The thread which executes the query is woken up each 500 milliseconds. Table 6 shows the result of SemLAV and parallel SemLAV using the time dependent strategy for 20 threads. The results also show that parallel SemLAV outperforms SemLAV in terms of throughput and

total execution time; however, the time for first results is increased as when the view dependent criterion is executed.

Table 7: Result of BSBM over SemLAV and parallel SemLAV using the Data Dependent Criterion with 20 threads; queries are executed whenever 500 triples have been inserted in the integrated RDF graph (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	SemLAV				parallel SemLAV			
Query	TT	TFA	Throughput	#EQ	TT	TFA	Throughput	#EQ
1	606,697	6,370	37.3501	15	604,668	27,306	**62.8580**	10
2	600,656	260,333	0.9823	66	603,706	**68,132**	**0.9916**	14
4	660,938	104,501	0.0004	47	343,267	234,513	**0.0008**	21
5	632,809	116,037	0.8916	28	431,564	162,773	**1.3074**	16
6	625,173	43,306	0.1892	24	248,937	165,997	**0.7918**	14
8	627,612	5,393	0.8990	42	318,207	17,766	**1.7731**	8
9	5,107	1,235	5.5240	18	2,717	1,731	**10.3831**	4
10	607,841	9,810	4.9243	44	459,995	24,917	**6.5070**	15
11	601,042	8,352	4.9800	43	112,908	25,505	**26.5099**	7
12	609,509	5,784	0.9822	121	377,970	15,762	**1.5838**	15
13	671,893	183,844	0.8910	124	385,730	**42,222**	**1.5520**	24
14	636,387	29,201	0.5419	24	304,364	163,948	**1.8538**	17
15	645,172	2,911	0.4373	37	410,031	13,808	**0.6880**	19
16	648,826	2,531	0.4348	46	315,466	13,349	**0.8943**	8
17	644,090	1,504	0.3060	32	297,911	4,792	**0.6616**	9
18	651,094	> 600,000	0.0000	12	520,845	**302,575**	**1.0833**	13

The Data Dependent Criterion: The query thread is woken up each time the integrated RDF graph grows up to 500 new triples. Table 7 shows the results of SemLAV and parallel SemLAV using data dependent strategy for 20 threads. As in previous experiments, parallel SemLAV outperforms SemLAV in terms of throughput and total execution time for all queries; but the time for the first result is increased for the majority of the queries.

The Two-phase Criterion: The first phase of this strategy performs an `ASK` query and when it returns `true`, the second phase is conducted. First, the second phase executes the original query, then the query engine will be woken up either each n milliseconds or when n triples are inserted into the integrated RDF graph. Table 8 reports on the results for the two-phase strategy when the query is executed whenever 500 triples are inserted into the integrated RDF graph. Parallel SemLAV outperforms SemLAV in terms of throughput for all the queries, but throughput values of parallel SemLAV are lower than in previous experiments.

Table 8: Result of SemLAV and parallel SemLAV on BSBM using the Two-phase Criterion with 20 threads; queries are executed whenever 500 triples have been inserted in the integrated RDF graph (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	SemLAV				parallel SemLAV			
Query	TT	TFA	Throughput	#EQ	TT	TFA	Throughput	#EQ
1	606,697	6,370	37.3501	15	604,693	26,624	**62.4690**	4
2	600,656	260,333	0.9823	66	603,290	**72,463**	**0.9923**	8
4	660,938	104,501	0.0004	47	358,149	261,954	**0.0008**	11
5	632,809	116,037	0.8916	28	441,166	169,437	**1.2789**	13
6	625,173	43,306	0.1892	24	275,440	186,320	**0.7156**	6
8	627,612	5,393	0.8990	42	329,872	24,852	**1.7104**	7
9	5,107	1,235	5.5240	18	2,572	1,966	**10.9685**	3
10	607,841	9,810	4.9243	44	475,523	25,193	**6.2945**	15
11	601,042	8,352	4.9800	43	111,739	25,490	**26.7872**	7
12	609,509	5,784	0.9822	121	396,899	16,209	**1.5083**	14
13	671,893	183,844	0.8910	124	369,586	**44,197**	**1.6197**	10
14	636,387	29,201	0.5419	24	308,277	155,879	**1.8302**	10
15	645,172	2,911	0.4373	37	400,752	14,299	**0.7040**	18
16	648,826	2,531	0.4348	46	330,846	12,741	**0.8527**	8
17	644,090	1,504	0.3060	32	274,087	5,984	**0.7192**	8
18	651,094	> 600,000	0.0000	12	517,814	**285,958**	**1.0896**	13

Table 9: Throughput of SemLAV and parallel SemLAV (PS) using the Data-Dependent Criterion each 500 triples (DDC), Time-Dependent Criterion each 500 milliseconds (TDC), and Two-phase Criterion that combines ASK queries with DDC. With 20 threads for each criterion (**bold** font is used to highlight the values where parallel SemLAV outperforms SemLAV)

	Throughput				
Query	SemLAV	PS	PS+DDC	PS+TDC	PS+DDC+ASK
1	37.3501	**88.7036**	62.8580	67.3762	62.4690
2	0.9823	0.9883	0.9916	**0.9941**	0.9923
4	0.0004	**0.0008**	**0.0008**	**0.0008**	**0.0008**
5	0.8916	1.2339	**1.3074**	1.2119	1.2789
6	0.1892	0.7203	**0.7918**	0.7395	0.7156
8	0.8990	1.7716	**1.7731**	1.6877	1.7104
9	5.5240	11.5006	10.3831	**12.0405**	10.9685
10	4.9243	**6.8094**	6.5070	6.5054	6.2945
11	4.9800	**28.3219**	26.5099	26.1002	26.7872
12	0.9822	1.6072	1.5838	**1.6746**	1.5083
13	0.8910	1.5266	1.5520	**1.6458**	1.6197
14	0.5419	1.6905	**1.8538**	1.8498	1.8302
15	0.4373	**0.7270**	0.6880	0.6842	0.7040
16	0.4348	0.9198	0.8943	**0.9325**	0.8527
17	0.3060	0.7082	0.6616	**0.8386**	0.7192
18	0.0000	1.1071	1.0833	**1.1083**	1.0896

4.3 Discussion

Table 9 summarizes the results of the throughput with 20 threads in the different empirical evaluations. In all experiments, parallel SemLAV outperforms SemLAV in terms of the throughput and total execution time. However, none of the defined execution criterion dominates other criterion. For instance, parallel SemLAV with query execution every 500 milliseconds is the best execution strategy for $query2$; whereas parallel SemLAV with execution strategy whenever 500 triples have been inserted into the integrated RDF graph is the most suitable strategy for $query5$. We repeat the experiments with different number of threads. In setup with 20 threads, parallel SemLAV outperforms SemLAV in terms of throughput and total execution time but it increases time for first answer. Preliminary results suggest that there is a tradeoff between throughput and time for first answer. To confirm these results, in the future, we plan to evaluate parallel SemLAV with different time and data setups.

5 Conclusions and Future Work

We tackle the problem of executing SPARQL queries against LAV views in a parallel fashion. The query execution model relies on an RDF graph that temporally materializes the data retrieved from the relevant views of a SPARQL query. The query engine respects a concurrency model that prioritizes the execution of queries against the integrated RDF graph over loading data from the views. Additionally, a non-blocking query execution strategy allows for the execution of a SPARQL query on an RDF graph depending on different criteria. Similarly than SemLAV, our proposed parallel query execution model, named parallel SemLAV, was implemented on top of Jena. We empirically compared parallel SemLAV and SemLAV in terms of the impact of the non-blocking strategy on the query engine throughput. The observed results suggest that independently of the criterion followed by the non-blocking query engine strategy, parallel SemLAV outperforms SemLAV in terms of throughput. One limitation of our current implementation is inherent from the techniques implemented by Jena to handle concurrent insertions in an RDF graph. To overcome this limitation, we plan to consider a graph database engine as the RDF store backend, in order to provide more robust concurrency management of the RDF graph for incremental query processing.

Acknowledgement

We thank Maxime Pauvert and Nicolas Brondin, both students of the Computer Science Department at the University of Nantes for implementing the non-blocking strategy.

References

1. Virtuoso sponger. White paper, OpenLink Software.

2. S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart. *Web Data Management*. Cambridge University Press, New York, NY, USA, 2011.
3. Y. Arvelo, B. Bonet, and M.-E. Vidal. Compilation of query-rewriting problems into tractable fragments of propositional logic. In *AAAI*, pages 225–230. AAAI Press, 2006.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
5. C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
6. R. Castillo-Espinola. *Indexing RDF data using materialized SPARQL queries*. PhD thesis, Humboldt-Universität zu Berlin, 2012.
7. A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.
8. P. Folz, G. Montoya, H. Skaf-Molli, P. Molli, and M. Vidal. Semlav: Querying deep web and linked open data with SPARQL. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pages 332–337, 2014.
9. T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, and C. Schallhart. OPAL: automated form understanding for the deep web. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 829–838, 2012.
10. T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. DIADEM: thousands of websites to a single database. *PVLDB*, 7(14):1845–1856, 2014.
11. B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the Deep Web. *Commun. ACM*, 50(5):94–101, 2007.
12. G. Konstantinidis and J. L. Ambite. Scalable query rewriting: a graph-based approach. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors, *SIGMOD Conference*, pages 97–108. ACM, 2011.
13. G. Montoya, L. D. Ibáñez, H. Skaf-Molli, P. Molli, and M.-E. Vidal. SemLAV: Local-As-View Mediation for SPARQL. *Transactions on Large-Scale Data- and Knowledge-Centered Systems XIII, Lecture Notes in Computer Science, Vol. 8420*, pages 33–58, 2014.
14. G. L. Peterson and J. E. Burns. Concurrent reading while writing II: the multi-writer case. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 383–392, 1987.
15. M. Taheriyan, C. A. Knoblock, P. A. Szekely, and J. L. Ambite. Rapidly integrating services into the linked data cloud. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2012.
16. J. D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.
17. R. Verborgh, O. Hartig, B. D. Meester, G. Haesendonck, L. D. Vocht, M. V. Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. V. de Walle. Querying datasets on the web with high availability. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pages 180–196, 2014.
18. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.