Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 1208 – 1222

doi:10.3906/elk-1308-45

On scalable RDFS reasoning using a hybrid approach

Tuğba KÜLAHCIOĞLU, Hasan BULUT*
Department of Computer Engineering, Ege University, Bornova, İzmir, Turkey

Received: 07.08.2013 • **Accepted/Published Online:** 24.02.2014 • **Final Version:** 23.03.2016

Abstract: Reasoning is a vital ability for semantic web applications since they aim to understand and interpret the data on the World Wide Web. However, reasoning of large data sets is one of the challenges facing semantic web applications. In this paper, we present new approaches for scalable Resource Description Framework Schema (RDFS) reasoning. Our RDFS specific term-based partitioning algorithm determines required schema elements for each data partition while eliminating the data partitions that will not produce any inferences. With the two-level partitioning approach, we are able to carry out reasoning with limited resources. In our hybrid approach, we integrate two previously mentioned methods to benefit from the advantages of both. In the experimental tests we achieve linear speedups for reasoning times with the proposed hybrid approach. These algorithms and methods presented in the paper enable RDFS-level reasoning of large data sets with limited resources, and they together build up a scalable distributed reasoning approach.

Key words: RDFS reasoning, distributed reasoning, data partitioning, data elimination, scalability

1. Introduction

Today, the World Wide Web is turning into a knowledge base that is understandable and processable by machines. Extending the human-understandable version, this new type of the Web is called the Semantic Web and it enables computers to easily find, share, and integrate data [1].

Reasoning is the ability to derive logical consequences from previously known propositions. For example, if we have the following two propositions, "All men are mortal" and "Socrates is a man", then using deductive reasoning we can infer that "Socrates is mortal". Unless they have an advanced natural language processing capability, computers cannot interpret those facts that are defined by human understandable languages.

Reasoning is a vital ability for semantic web applications since they aim to understand and interpret the data on the Web. To give computers reasoning ability, semantic web applications use knowledge representation languages to represent data [2]. These languages create a vocabulary for a specific domain, which determines general and shared meanings for the concepts of this particular domain.

Using reasoners, semantic web applications are able to do reasoning on semantic web data. However, the amount of data in semantic web supporting notations has already grown to vast amounts [3], and the reasoning of these large data sets is one of the challenges facing semantic web reasoners [4].

To solve the scalability problem of semantic web reasoners, proposed solutions [5–13] suggest applying distributed computing techniques. Some of these studies guarantee reaching full closure [5–10], while some of them argue that they eventually reach full closure with an infinite loop [11–13]. By reaching full closure, we

*Correspondence: hasan.bulut@ege.edu.tr

mean obtaining all possible inferences from a set of known propositions (for a defined set of rules), which is an expected feature of a semantic reasoner.

One of the difficulties for a distributed reasoning approach is ensuring the load-balance. Many of the deterministic approaches (the approaches that guarantee reaching full closure) [5,6,8] depend on term-based techniques. Since the term-distribution of semantic web data is highly skewed [11,14], it is hard, and some argue impossible [11], to do a balanced distribution of these data based on the terms. These systems also suffer from input data replication.

A deterministic but not term-based approach was studied in [10]. This study requires the schema triples to be known beforehand and replicates whole the schema in all nodes. As we will show in our experimental tests, the sequential distribution policy applied in their tests results in an unintended optimization, which is not necessarily the case for other data sets.

In this paper we present new approaches for scalable Resource Description Framework Schema (RDFS) reasoning. Our first approach is an RDFS specific term-based partitioning algorithm that is based on RDFS entailment rules. This algorithm replicates only required parts of schema triples and does not require replication of any data triples. Beyond not replicating any data triples, this algorithm allows us to define parts of the data that will not produce any inferences and thus can be eliminated from the reasoning process. This algorithm can be used as a data distribution algorithm in a distributed settling, and it can be used as a preprocessing algorithm in a local reasoning approach.

Instead of using supercomputers or huge computer clusters as done by similar studies, our second approach, namely two-level partitioning, allows carrying out reasoning with limited resources, such as with an ordinary PC. With this approach, we can do reasoning of a large data set, which cannot be done with available reasoning tools.

Finally, in our hybrid approach, we integrate our first approach with the second one to benefit from the advantages of both. Experimental results show that parallel implementation of the hybrid approach can achieve linear speedup.

The remainder of the paper is organized as follows. Section 2 provides information on related studies. In Section 3, we introduce the examined problem and demonstrate it with test results. In the subsequent three sections, we present our approaches to scalable RDFS reasoning. In Section 4, we present our RDFS specific term-based partitioning algorithm. In Section 5, we introduce our two-level partitioning method. In Section 6, a hybrid approach that combines advantages of previously presented methods is suggested. In the evaluation section, we compare the proposed methods. Finally, in the last section, main advantages of our approaches and conclusions are provided.

2. Related work

Several studies have used distributed hash table-based deterministic partitioning methods. In such systems, triples meet in nodes belonging to the shared terms. In [6], DHTs are used to distribute triples based on their subjects, predicates, and objects, which cause high data replication. Fang et al. [5] used DHTs to distribute triples for OWL (Web Ontology Language) reasoning, without considering the load-balance of the system.

In [7], triples were partitioned based on their subjects and objects for OWL-Horst [15] level reasoning. In another work [8], the authors compared graph, hash-based, and domain-based data partitioning techniques. They point out that graph and domain-based partitioning techniques give reasonable results, whereas hash-based partitioning has a low performance depending on the excessive data replication it requires.

Urbani et al. [9] presented a distributed reasoning system based on the MapReduce paradigm. They followed a rule-ordering approach to eliminate the need for iteration to handle the newly produced triples.

Weaver and Hendler [10] presented a straightforward partitioning strategy, which is referred to as a "one-level partitioning" strategy throughout the paper. This strategy allows partitioning data arbitrarily as long as the ontology schema is repeated in each peer. This method ensures complete reasoning because RDFS entailment rules contain at most one data triple in their bodies. Thus, inferences are made using one schema and one data triple (excluding the ones with one triple in their bodies: they do not even need a schema triple).

In [12,13], data are first partitioned randomly and inferences are computed. Then the data are exchanged between peers based on a routing strategy that combines data clustering with random exchanges. In a follow-up study [11], the clustering strategy was extended to include local and distributed clustering levels. Local clustering deals with selecting data to load from local memory to the reasoner, whereas distributed clustering is about selecting data to exchange between peers.

3. Problem definition

This paper studies reasoning of large semantic web data sets. To more clearly state the examined problem, we have performed several tests using one of the selected currently available reasoning technologies. Table 1 summarizes results of these tests (performed on a computer with Intel Xeon CPU E5504 @ 2.00 GHz, 4 GB cache, and up to 12 GB memory) using the Jena [16] RDFS Reasoner in the "RDFS_SIMPLE" mode, which considers only rules with two triples in their bodies.

Table 1. Reasoning times for different memory sizes.

Data set	Triple count	Reasoning times for different memory sizes			
		1024 MB	2048 MB	4096 MB	8192 MB
SwetoDBLP/32	0.4 M	5.1 s	4.9 s	4.9 s	4.8 s
SwetoDBLP/16	0.8 M	14.6 s	9.5 s	9.3 s	9.3 s
SwetoDBLP/8	1.6 M	-	32.2 s	19.6 s	19.3 s
SwetoDBLP/4	3.3 M	-	-	46.3 s	44.7 s
SwetoDBLP/2	6.7 M	-	-	-	114.8 s
SwetoDBLP/1	13.4 M	-	-	-	-

SwetoDBLP [17] is a large-scale ontology that contains bibliographic information of computer science publications, and its main source is the computer science bibliographic database called DBLP [18].

In the tests, the SwetoDBLP data set is used and divided into 16 parts to obtain data sets of different sizes. Reasoning is then applied to these data sets using different memory sizes on the same computer. The dashes used in the table indicate that the reasoning could not be completed because of an out-of-memory error.

As seen in Table 1, the reasoning capability for large data sets increases with the available memory size. This is because the reasoner tries to create a graph of the whole data set in the memory to carry out reasoning on the graph. Another conclusion that can be inferred from the table is that the relation between the sizes of the data sets and reasoning times is not linear. Doubling the data set size increases reasoning time more than twice.

Tests show us that even a computer with 8 GB memory cannot process data sets with tens of millions of triples. Given that there are already data sets with billions of triples [3], it is clear that there is a scalability problem to be solved with the reasoning of semantic web data.

4. RDFS specific term-based partitioning algorithm

In this section we present our RDFS specific term-based partitioning algorithm that divides the original data set into independent partitions whose unified output reaches full closure. Section 4.1 presents the algorithm and describes its development phase. More details on the algorithm can be found in [19], where we first introduced this algorithm. In Section 4.2, we provide an illustration of the algorithm. Finally, in Section 4.3, we show the effects of our algorithm on the reasoning process of the SwetoDBLP data set.

4.1. Development of the algorithm

Our algorithm is based on the RDFS entailment rules listed in Table 2. The full list of RDFS entailment rules can be found in the World Wide Web Consortium Recommendation for RDF Semantics (http://www.w3.org/TR/rdf-mt/#RDFSRules). While developing our overall approach, we considered rules with two triples in their bodies since the rules with only one triple in their bodies are inferable anytime, and moreover their results (produced inferences) do not contribute to the execution of other rules.

Table 2. Selected RDFS entailment rules.

Rule name	Rule body		Rule head
rdfs2	p rdfs:domain c	s p o	s rfd:type c
rdfs3	p rdfs:range c	s p o	o rfd:type c
rdfs5	p1 rdfs:subPropertyOf p2	p2 rdfs:subPropertyOf p3	p1 rdfs:subPropertyOf p3
rdfs7	p1 rdfs:subPropertyOf p2	s p1 o	s p2 o
rdfs9	c1 rdfs:subClassOf c2	s rdf:type c1	s rdf:type c2
rdfs11	c1 rdfs:subClassOf c2	c2 rdfs:subClassOf c3	c1 rdfs:subClassOf c3

While developing the algorithm, we defined three subgoals to reach the full closure:

1) Each partition should be able to handle reasoning of initial data triples;

2) Each partition should be able to handle reasoning of newly produced (inferred) data triples;

3) Each partition should be able to handle reasoning of schema triples.

By coming together, these three properties give partitions the ability to complete their reasoning independently of other partitions, and when combined together, their output is able to obtain the full closure for any given data set at the RDFS level. In the rest of this section, we provide details on how our algorithm achieves these three subgoals.

4.1.1. Handling initial data triples

To develop a partitioning strategy considering initial data triples, we examine rules 2, 3, 7, and 9 (these are the rules that contain data triples in their bodies):

- Rule 2 derives an inference only if the subject of the triple representing the "rdfs:domain" relationship is equal to the predicate of the triple representing "property" relationship.
- Rule 3 derives an inference only if the subject of the triple representing the "rdfs:range" relationship is equal to the predicate of the triple representing the "property" relationship.

- Rule 7 derives an inference only if the subject of the triple representing the "rdfs:subPropertyOf" relationship is equal to the predicate of the triple representing the "property" relationship.
- Rule 9 derives an inference only if the subject of the triple representing the "rdfs:subClassOf" relationship is equal to the object of the triple representing the "rdf:type" relationship

After previous analyses, it can be concluded that from the initial data triples' point of view it is possible to determine a partition parameter for each set of triples that represent a specific relationship. Our conclusions are summarized in Table 3. According to Table 3, the triples that represent an "rdf:type" relationship should be assigned to a partition based on its object. Similarly, the triples that represent rdfs level relationships should be assigned to a partition based on their subjects. All other triples (represented as "s p o" triples) should be handled based on their predicates.

Table 3. Triple specific partitioning parameters.

Triple			Partition parameter	Related rule(s)
Subject	Predicate	Object		
s	p	o	Predicate	2, 3, 7
s	rdf:type	c	Object	9
p	rdfs:domain	c	Subject	2
p	rdfs:range	c	Subject	3
p	rdfs:subPropertyOf	q	Subject	7
c	rdfs:subClassOf	d	Subject	9

Using the knowledge given in Table 3, we developed the first part (lines 1–8) of our RDFS specific partitioning algorithm given in Algorithm 1. For each type of relationship, our algorithm uses the corresponding partitioning parameter.

4.1.2. Handling inferred data triples

Handling inferred data triples requires holding the related schema triples in advance. For instance, rule 2 infers "type" triples. According to Table 3, this triple should be located with schema triples containing its object as their subjects. If its object could be known in advance, the partition could be extended with related schema triples (at the initial partition creation time). As summarized below, its object can be deduced in advance using rules 2, 3, 7, and 9.

- Rule 2 derives a triple representing the "rdf:type" relationship. The object, its partitioning parameter, of the inferred type triple comes from the object of the the "rdfs:domain" triple in the body of the rule.
- Rule 3 derives a triple representing the "rdf:type" relationship. The object, its partitioning parameter, of the inferred type triple comes from the object of the "rdfs:range" triple in the body of the rule.
- Rule 7 derives a triple representing the property relationship. The predicate, its partitioning parameter, of the property triple comes from the object of the "rdfs:subPropertyOf" triple in the body of the rule.
- Rule 9 derives an inference representing the "rdf:type" relationship. The object, its partitioning parameter, of the type triple (partition parameter of type triple) comes from the object of the "rdfs:subClassOf" triple in the body of the rule.

Algorithm 1. RDFS specific term-based partitioning.

Input: Input Triples

Output: Term-Based Partitions

```
1  for each triple in input triples do
2     if triple represents a type relationship
3        assign triple to the data partition of its object
4     else if triple is a schema triple (domain/range/subclassOf/subPropertyOf)
5        assign triple to the schema partition of its subject
6     else
7        assign triple to the data partition of its predicate
8  end for
9  for each partition to be updated in schema partitions do
10    for each triple in partition to be updated
11       term ← object of the triple
12       partition to be added ← schema partition of the term
13       call ExtendPartition (partition to be added, partition to be updated)
14    end for
15 end for
16 return
```

The results of the previous analyses are summarized in Table 4. As suggested by Table 4, the partitions should be extended with the schema partitions of the terms corresponding to the objects of the schema triples in hand. To be able to handle all subsequent inferences, the extension should continue until no more new triples can be added to a partition.

Table 4. Extension parameters for schema triples.

Triple				Extension parameter	Related rule(s)
Subject	Predicate	Object			
p	rdfs:domain		c	Object	2
p	rdfs:range		c	Object	3
p	rdfs:subPropertyOf		q	Object	5, 7
c	rdfs:subClassOf		d	Object	9, 11

Lines 9–15 of Algorithm 1 perform this extension. Each schema partition is extended with schema partitions belonging to the objects of its schema triples. This extension is implemented through a recursive algorithm given in Algorithm 2.

4.1.3. Handling schema triples

To develop a partitioning strategy considering schema triples, rules 5 and 11 are examined (rules that contain two schema triples in their bodies):

Algorithm 2. ExtendPartition.

Input: Partition to be added, Partition to be updated

Output: Updated partition

```
1  for each triple in partition to be added do
2     if triple is not already in partition to be updated
3        add triple to the partition to be updated
4        term ← object of the triple
5        call AddPartition (schema partition of term, partition to be updated)
6     end if
7  end for
8  return
```

- Rule 5 derives an inference only if the subject of one of the triples representing the "rdfs:subPropertyOf" relationship is equal to the object of another triple representing the "rdfs:subPropertyOf" relationship.
- Rule 11 derives an inference only if the subject of one of the triples representing the "rdfs:subClassOf" relationship is equal to the object of another triple representing the "rdfs:subClassOf" relationship.

Since subClassOf and subPropertyOf triples are initially distributed based on their subjects, they should be located with other subClassOf and subPropertyOf triples that hold this term as their object. Recalling the algorithm in the previous subsection, we have already extended schema partitions in a way that fulfills this requirement. Thus, we are able to obtain all inferences derived through initial and inferred schema triples.

4.2. Illustration of the algorithm

To better understand the algorithm, it is illustrated on a sample triple set: {"a rdf:type x", "b rdf:type x", "x rdfs:subClassOf y", "y rdfs:subClassOf z"} . Table 5 shows the resulting partitions. Data partitions and extended schema partitions totally include 5 triples in 4 partitions (belonging to only two terms). Recalling that we have 4 triples initially, only one triple is replicated. The derived inferences are final and do not need any iteration or triple exchange.

Table 5. Illustration of RDFS specific term-based partitioning algorithm.

Term	Initial schema partition	Extended schema partition	Data partition	Inferences
x	x rdfs:subClassOf y	x rdfs:subClassOf y	a rdf:type x	a rdf:type y
		y rdfs:subClassOf z	b rdf:type x	b rdf:type y
				a rdf:type z
				b rdf:type z
				x rdfs:subClassOf z
y	y rdfs:subClassOf z	y rdfs:subClassOf z		

4.3. Analysis of SwetoDBLP data set

In this section, the SwetoDBLP data set is analyzed based on its term distribution and partition sizes obtained by the RDFS specific term-based partitioning algorithm.

Figure 1 shows the terms and corresponding data partitions created by Algorithm 1 and Algorithm 2. The partitions colored with gray indicate data partitions with corresponding empty schema partitions and thus can be eliminated from the reasoning process since RDFS entailment rules do not contain a rule with two data triples in its body.

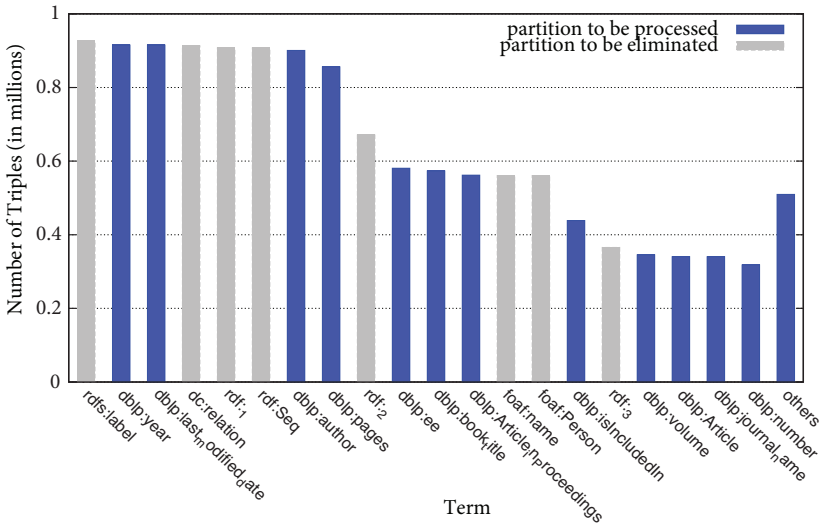

Figure 1. Terms-partition sizes for SwetoDBLP data set.

For this example, it is possible to obtain full closure (for rules given in Table 1) with processing only around 55% of the original data set. Meanwhile, 45% of the data set, around 6M triples, can be eliminated from the reasoning process, which dramatically reduces the computation load. Thus, beyond not replicating the data triples, this approach reduces the data set size that needs to be processed to reach full closure.

5. Two-level partitioning method

This section presents our two-level partitioning method, which enables processing large data sets with limited resources. In the first subsection, we give details of the proposed method, and in the second subsection we provide results of the experimental tests carried out.

5.1. Method

As mentioned before, in [10] it is proved that by replicating the whole schema, data can be arbitrarily partitioned among nodes. This approach guarantees reaching full closure for RDFS level reasoning. However, following that approach, the amount of data assigned to a particular node can exceed the maximum data size that can be processable by this node.

To overcome this limitation, we suggest adding a second-level partitioning. As shown in Figure 2, this second-level partitioning divides the data into further partitions, which are at or below the size of the node capacity. These partitions need to be executed sequentially by the assigned node.

5.2. Experimental tests

Both one-level and two-level partitioning methods are implemented to be able to compare them. Tests are carried out on up to 16 computers. The computer specifications are as follows: Intel Core 2 Duo CPU E4600 2.40 GHz processor, Linux 26.38-8-generic Ubuntu operating system, and 1 GB available memory.

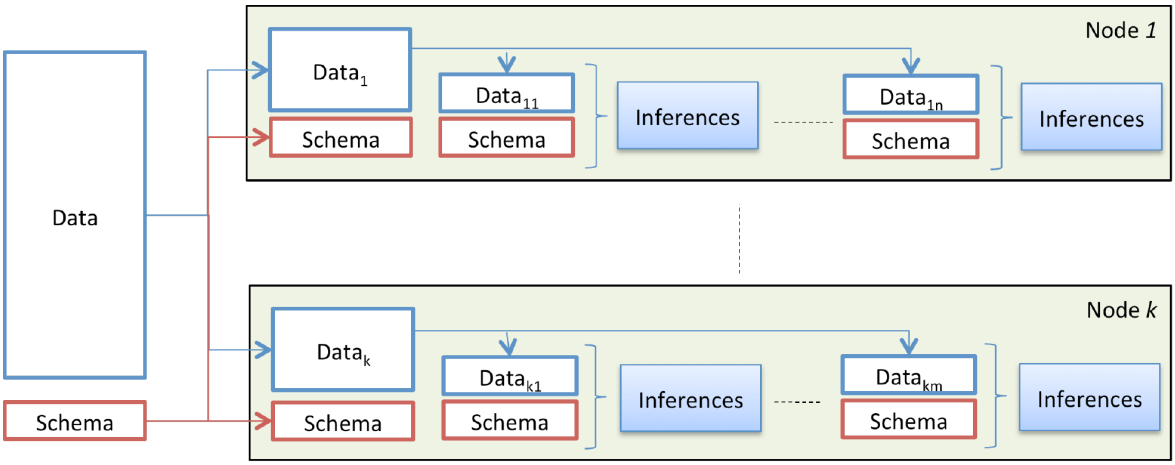

Figure 2. Two-level partitioning method.

5.2.1. Determining threshold value

This method depends on dividing the data into partitions that are at or below the size of the node capacity. We have carried out tests to be able to define this threshold value. Table 6 summarizes the results of these tests. As can be seen from the table, we are not able to process a partition of one million triples with a computer with 1 GB available memory. so the threshold value should be smaller than this size.

Table 6. Time and output results for different threshold values.

Threshold value	Average time	Output size
10,000	58.29	17.9 M
50,000	63.43	17.4 M
100,000	68.22	17.3 M
500,000	78.03	16.7 M
1,000,000	-	-

On the other hand, we are able to process 500,000 triples in 78 s on average. To be able to identify the time and output size relationship with the decreasing value of threshold, we repeated tests with smaller threshold values as given in Table 6. For a partition of 50,000 triples, time increases 8.81%, whereas output is decreasing 2.79%. The imbalance gets even worse with the increasing size of triple count.

As a result of the above tests, the threshold triple size is selected as 10,000 triples for the current test setup. In other words, partitions with higher triple sizes are divided into subpartitions of 10,000 triples.

5.2.2. Test results

Table 7 shows the results of the tests carried out on the SwetoDBLP [17] data set. The dashes used in the table indicate that the reasoning could not be completed because of an out-of-memory error.

As seen in Table 7, using the one-level partitioning method [10], tests could only be carried out on 16 computers. On the other hand, two-level partitioning could complete reasoning even on a single computer. This clearly shows that carrying out reasoning with limited resources can be achieved using the two-level partitioning method.

Table 7. Reasoning times for different methods.

Node count	Triple count (per node)	Reasoning times for different methods (in seconds)			
		One-level partitioning		Two-level partitioning	
		Sequential	Random	Sequential	Random
1	13.4 M	-	-	551.3	667.7
2	6.7 M	-	-	279.8	327.4
4	3.3 M	-	-	143.5	163.5
8	1.6 M	-	-	76.9	81.5
16	0.8 M	43.3	70.8	40.9	44.8

An important feature of our tests is that they are repeated for two different distribution policies. In [10], data is sequentially distributed in all tests. For example, if the triple count per node is n, the first n triple of the original data set is sent to the first node, the following n triple is sent to the second node, etc.

In order to understand the impact of this distribution method, we shuffled the data file and applied the same distribution policy on this new data set. In short, we distributed the data randomly over nodes. As a result of tests, it is observed that random distribution gives worse results than sequential distribution for both methods. To understand the reason for this, the data set and its output should be examined in more detail.

In the original data file, triples belonging to the same resources are generally located next to each other. This kind of placement minimizes repetitions because triples that can lead to the same inferences are located in the same node. Since it is not necessarily the case for every data set, a reasonable method should not be dramatically affected by the distribution policy.

Table 8 shows output sizes for the tests carried out on 16 computers. The increase in the size of the output is detrimental to the system because it shows the increase of repetition.

Table 8. Output sizes for different methods.

	One-level partitioning		Two-level partitioning	
Distribution policy	Sequential	Random	Sequential	Random
Output size	17.1 M	26.0 M	18.5 M	29.7 M

When Tables 7 and 8 are examined together, it is seen that two-level partitioning takes less time, although it produces more repetition. This is mainly because of the nonlinear increase of the reasoning times with the increasing amount of data, which can also be observed from the values in Table 1. Since two-level partitioning works on smaller partitions, it produces more repetitions with less time.

6. Hybrid approach

The methods described so far have their own advantages. Besides avoiding input replication in a distributed setting, the RDFS specific term-based partitioning algorithm gives us the chance to eliminate the parts of the data that do not produce any inference and thus reduces the computation load. On the other hand, by using the two-level partitioning method, it becomes possible to perform reasoning with limited resources. In this section, a hybrid approach that combines advantages of both methods is demonstrated and results of experimental tests are given.

6.1. Method

The proposed method is modeled in Figure 3 and the basic steps of the method can be listed as follows:

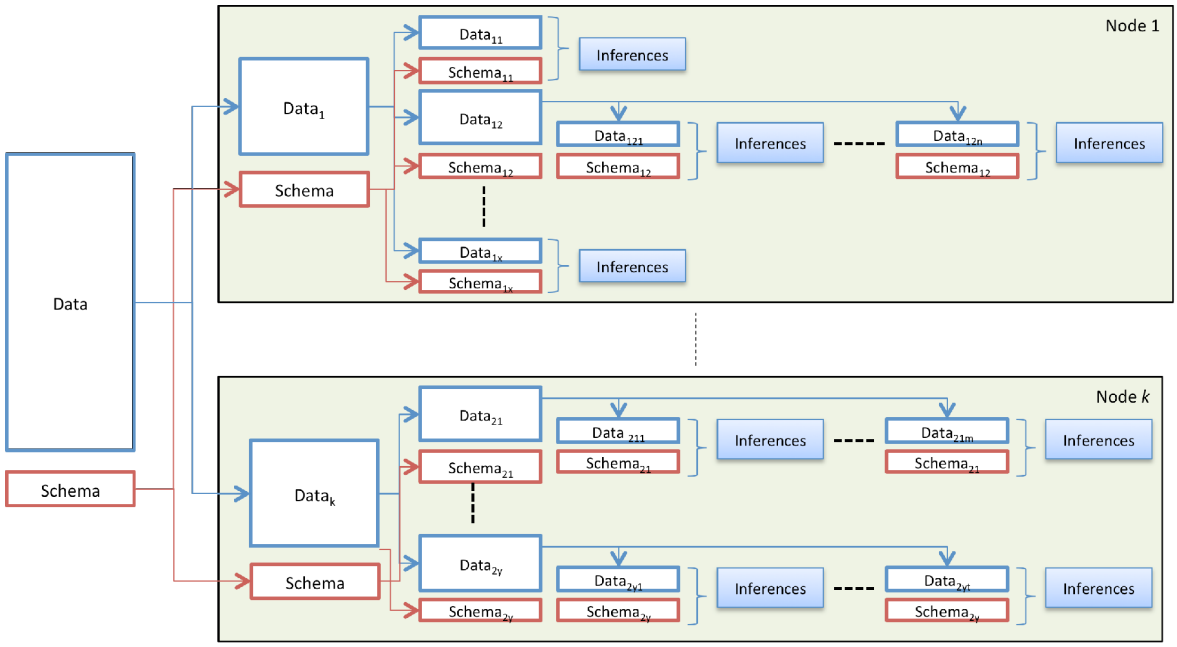

Figure 3. Hybrid approach.

- The Data are equally distributed over nodes as done in the first step of the two-level partitioning approach. The Distribution can be done using a sequential or random partitioning algorithm.
- Each node processes its data using the RDFS specific term-based partitioning algorithm.
- Nodes apply reasoning on resulting partitions of the second step. If any partition is over the capacity of the node's processing capability, the partition is divided into smaller parts as done in the second step of the two-level partitioning approach.

6.2. Experimental tests

The proposed approach is tested over the same cluster of 16 computers. Similar to the previous tests, Jena is used as a reasoning tool, the threshold triple count is defined as 10,000 triples, and 1 GB of memory is used on each computer.

Figure 4 shows reasoning times of the SwetoDBLP data set using the hybrid approach with two different distribution policies. The reasoning time on a single computer, which was 551 s in the two-level partitioning method, is 427 s for the hybrid approach.

Figure 5 shows the speedup graph for the hybrid approach. For both distribution policies, the method shows near-linear speedup. Doubling the node count nearly halves the reasoning time.

In Table 9, the table that was created in the problem definition (Table 1) is updated with test results of the hybrid approach for different memory sizes. The values are comparable since the same computer and data sets are used while creating both.

As can be seen from the table, the hybrid approach can process the whole data set with 1 GB of memory, which originally cannot be processed even using 8 GB of memory. Besides enabling the processing of data sets that are otherwise not processable, the hybrid approach completes reasoning in shorter times. Moreover, with the hybrid approach, the reasoning times increase almost linearly with the data set sizes.

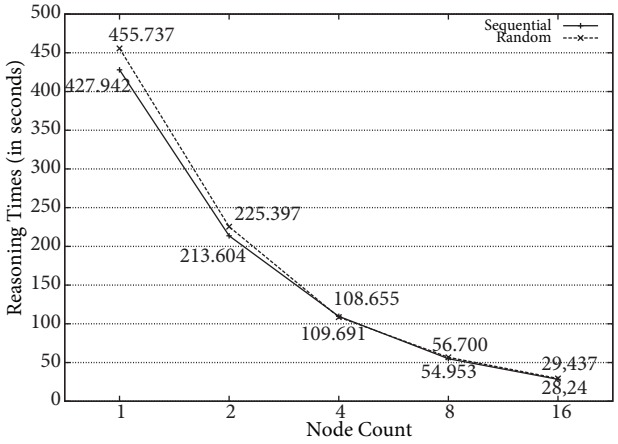

Figure 4. Reasoning times using hybrid approach.

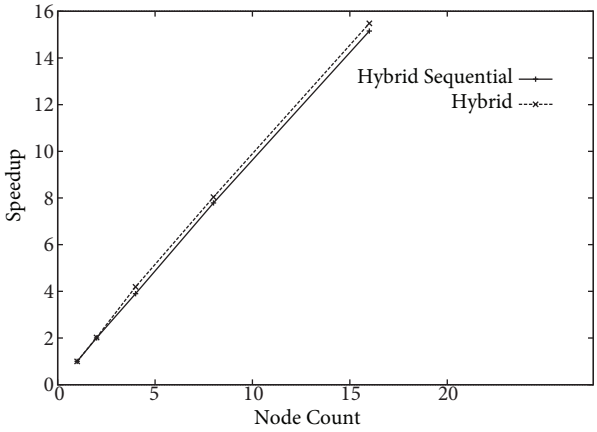

Figure 5. Speedup graph for the hybrid approach.

Table 9. Reasoning times for native and hybrid approaches.

Data set	Triple count	Reasoning times for different memory sizes (in seconds)							
		1024 MB		2048 MB		4096 MB		8192 MB	
		Native	Hybrid	Native	Hybrid	Native	Hybrid	Native	Hybrid
SwetoDBLP/32	0.4 M	5.1	4.5	4.9	4.4	4.9	4.3	4.8	4.3
SwetoDBLP/16	0.8 M	14.6	8.2	9.5	7.9	9.3	7.9	9.3	7.8
SwetoDBLP/8	1.6 M	-	14.9	32.2	14.8	19.6	14.7	19.3	14.6
SwetoDBLP/4	3.3 M	-	28.6	-	28.5	46.3	28.3	44.7	27.3
SwetoDBLP/2	6.7 M	-	59.3	-	58.7	-	55.7	114.8	55.4
SwetoDBLP/1	13.4 M	-	145.3	-	143.5	-	143.1	-	142.4

As seen from Table 9, increasing the memory size has little or no effect on processing times with the hybrid approach. For example, processing times for the SwetoDBLP/32 data set changes from 4.5 to 4.3 s for 1 GB to 8 GB memory sizes. For the whole SwetoDBLP data set, the change is between 145.3 and 142.4 s.

The fitted curve shown in Figure 6 is created based on the values of reasoning with 1 GB memory using the hybrid approach, and it shows that the method has a close-to-linear increase in time with the increasing amount of data. The slope of the line in Figure 6 is 4.32, which decreases to only 4.20 for 8 GB memory tests. It can be concluded that increasing the memory size beyond 1 GB has little effect on the performance of the proposed hybrid method.

7. Evaluation

In this section, methods that have been discussed so far are compared. Figure 7 shows reasoning times that have been obtained using these methods. In the one-level partitioning method, for cluster sizes of up to eight computers, we received out-of-memory errors. Hence, we could only provide results for the cluster size of 16 computers for that method.

When Figure 7 is analyzed, it is observed that the hybrid approach gives better results than other methods. The main reasons of this improvement are:

- Carrying out reasoning with smaller schema parts.
- Eliminating data parts that do not lead to any inferences.

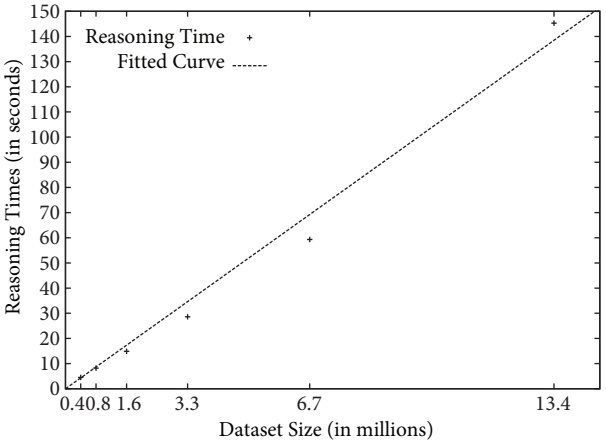

Figure 6. Fitted curve for reasoning times with 1 GB memory using hybrid approach.

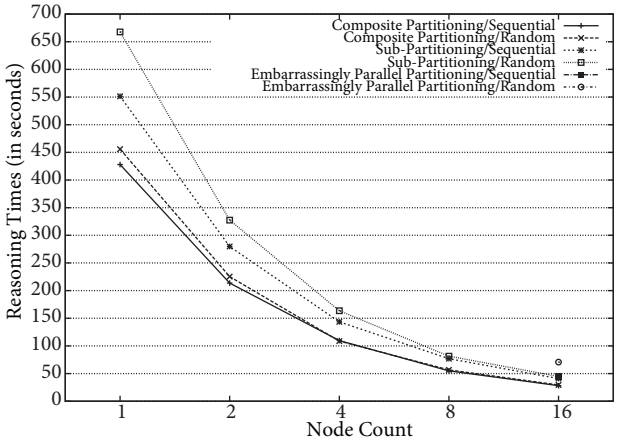

Figure 7. Reasoning times for different approaches.

In Figure 8, the methods are compared in terms of speedup, and Table 10 gives the exact speedup values. Results indicate that the hybrid approach has a higher speedup than the two-level partitioning method.

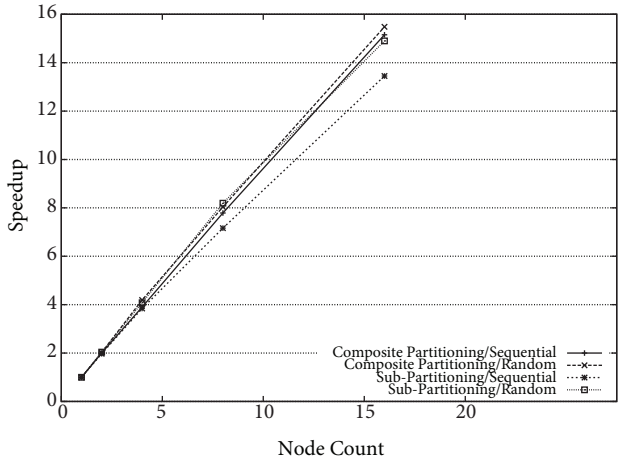

Figure 8. Speedup graph for two-level partitioning and hybrid approaches.

Table 10. Speedup values for two level partitioning and hybrid approaches.

Node count	Hybrid approach		Two-level partitioning	
	Sequential	Random	Sequential	Random
1	1	1	1	1
2	2.0	2.0	1.9	2.0
4	3.9	4.1	3.8	4.0
8	7.7	8.0	7.1	8.1
16	15.1	15.4	13.4	14.9

When test results are analyzed for a specific method, it can be seen that the performance of the two-level partitioning method reduces with the random distribution. As discussed previously, the reason is the unintended optimization that comes with the sequential distribution. On the other hand, as seen from test results, the hybrid approach is not affected by the distribution policy, since it analyzes all data to find and group related terms into a partition.

8. Conclusion

In this paper, the problem of reasoning of large semantic web data sets has been studied. Both local and distributed solutions are proposed to overcome this problem of semantic web. Two different methods with different advantages are developed and then these approaches are integrated to create a hybrid approach that has the advantages of both. We have provided the test results and their analyses.

The main advantages of the developed hybrid method are as follows:

- Unlike similar term-based partitioning algorithms, data triples are not replicated.
- Only the required parts of the schema triples are replicated in different partitions.
- Only the data triples that may lead to inferences are included in the process.
- The method does not require the schema to be known beforehand. Since it analyzes all triples and distinguishes data and schema triples, the whole schema can directly be obtained.

As a result, the proposed hybrid approach for reasoning of a large semantic web data set can perform reasoning on computers with limited capacity and makes the problem of reasoning more convenient to parallelization. In addition, the hybrid approach proposed in this paper achieves almost linear speedups.

Acknowledgments

Tuğba Külahcıoğlu thanks the Scientific and Technological Research Council of Turkey (TÜBİTAK) and Turkcell Academy for support during her MSc studies in which this study was conducted.

References

[1] Berners-Lee T, Hendler J, Lassila O. The semantic web. Sci Am 2001; 284: 34–43.

[2] Antoniou G, Harmelen VF. A Semantic Web Primer. 2nd ed. Cambridge, MA, USA: MIT Press, 2008.

[3] Bizer C, Mika P. The semantic web challenge. J Web Semant 2009; 8: 341–374.

[4] Li P, Zeng Y, Kotoulas S, Urbani J, Zhong N. The quest for parallel reasoning on the semantic web. In: 5th International Conference on Active Media Technology; 22–24 October 2009; Beijing, China.

[5] Fang Q, Zhao Y, Yang G, Zheng W. Scalable distributed ontology reasoning using DHT-based partitioning. In: 3rd Asian Semantic Web Conference; 2008; Bangkok, Thailand. Berlin, Germany: Springer-Verlag. pp. 91–105.

[6] Kaoudi Z, Miliaraki I, Koubarakis M. RDFS reasoning and query answering on top of DHTs. In: 7th International Semantic Web Conference; 2008; Karlsruhe, Germany. Berlin, Germany: Springer-Verlag. pp. 499–516.

[7] Soma R, Prasanna VK. A data partitioning approach for parallelizing rule based inferencing for materialized owl knowledge bases. In: 21st ISCA International Conference on Parallel and Distributed Computing and Communication Systems; 2008; New Orleans, LA, USA. pp. 19–25.

[8] Soma R, Prasanna VK. Parallel inferencing for OWL knowledge bases. In: International Conference on Parallel Processing; 2008. pp. 75–82.

[9] Urbani J, Kotoulas S, Oren E, Harmelen FV. Scalable distributed reasoning using MapReduce. In: 8th International Semantic Web Conference; 2009. pp. 634–649.

[10] Weaver J, Hendler J. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In: 8th International Semantic Web Conference; 2009. pp. 682–697.

[11] Kotoulas S, Oren E, Harmelen FV. Mind the data skew: Distributed inferencing by speeddating in elastic regions. In: 19th International World Wide Web Conference; 2010. pp. 531–540.

[12] Oren E, Kotoulas S, Anadiotis G, Siebes R, Teije AT, Harmelen FV. Marvin: Distributed reasoning over large-scale semantic web data. J Web Sem 2009; 7: 305–316.

[13] Oren E, Kotoulas S, Anadiotis G, Siebes R, Teije AT, Harmelen FV. MARVIN: A platform for large-scale analysis of Semantic Web data. In: WebScience 2009: Society On-Line.

[14] Ding L, Finin T. Characterizing the semantic web on the web. In: International Semantic Web Conference; 2006. pp. 242–257.

[15] Horst HJ. Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. J Web Semant 2005; 3: 79–115.

[16] Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K. Jena: implementing the semantic web recommendations. In: 13th International World Wide Web Conference; 2004. New York, NY, USA: ACM Press. pp. 74–83.

[17] Aleman-Meza B, Hakimpour F, Arpinar IB, Sheth AP. SwetoDblp ontology of computer science publications. J Web Semant 2007; 5: 151–155.

[18] Ley M. The DBLP computer science bibliography: evolution, research issues, perspectives. Lect Notes Comput Sc 2002; 2476: 481–486.

[19] Kulahcioglu T, Bulut H. Overcoming limitations of term-based partitioning for distributed RDFS reasoning. In: 5th Workshop on Semantic Web Information Management; 2013. New York, NY, USA: ACM. pp. 7:1–7:4.