Ontology Schema-specific Rule Materialization

Seungwoo Lee, Chang-Hoo Jeong, Jung-Ho Um, Taehong Kim, Hanmin Jung

Dept of Computer Intelligence Research, KISTI
245 Daehak-ro, Yuseong-gu, Daejeon, 305-806, Korea
{swlee, chjeong, jhum, kimtaehong, jhm}@kisti.re.kr

Abstract. The reasoning should tackle big data issues like other domains as the size of ontology grows bigger and bigger. Especially, rule-based reasoning should overcome the following challenges: duplicate elimination and rule matching efficiency. To deal with these challenges, we introduce a new rule-based reasoning method which materializes each generic instance rule into several schema-specific instance rules and combines with Hadoop framework to deal with billions of triples. The experiment shows the materialization remarkably improves the efficiency of rule-based reasoning by reducing the amount of required memory and making it linear to the data size.

Keywords: rete reasoning; rule materialization; RDFS rule; OWL Horst rule

1 Introduction

Rule-based reasoning is a process that derives new knowledge – it is represented in triples composed of subject, predicate and object in ontology reasoning – from given set of knowledge by matching more than one rules. However, the reasoning process is also suffering from big data issues like other domains as the size of ontology has become bigger and bigger. To achieve efficient reasoning with overcoming big data issues, we have several challenges and two of them are follows: duplicate elimination and rule matching efficiency.

First, separate input sets of triples may derive same – i.e., duplicate – triples by one rule. Even different rules may derive duplicate triples. Urbani et al.[1] pointed out that reasoning might derive 50 times more duplicates than unique derived triples in their preliminary simulation. So, we need an efficient mechanism for eliminating duplicates and this challenge should be overcome by all means to achieve the scalability of reasoning process. Second, some parts of reasoning rules are often so generic to cause too many matches of triples. Rules are generally defined from the semantics of vocabularies of ontology description languages such as RDF (Resource Description Framework), RDFS (RDF Schema) and OWL (Web Ontology Language). Therefore, these rules are always valid independently of any specific ontology and have generic triple patterns in their condition. These rules often cause inefficiency in matching them to given set of triples because the generic patterns could be matched to too many given triples, most of which are eventually filtered out when joining with triples

matched to remaining patterns. So, we need an efficient mechanism for reducing such unnecessary triple matches.

In this paper, we present a method that removes unnecessary pattern matches, eventually reduces join operations in rule-based reasoning selectively fetches input triples, and efficiently eliminates duplicates derived by reasoning rules.

The remaining part of this paper is as follows: in section 2, some related works are explored and in section 3, our main approaches are described. Some experiments justifying our approaches are given in section 4, which is followed by conclusion in section 5.

2 Related Work

Rule-based reasoning is implemented widely based on rete algorithm[2][3] due to its efficiency in pattern matching. This algorithm achieves efficiency of pattern matching by enabling more than one rule to share triples matched to their common triple patterns. Most time-consuming operation in rete occurs when joining triples from more than one pattern because join operation causes repeated search and comparison of corresponding values to common variables. To perform this efficiently, indexing mechanisms such as hashing or Pyramid technique are usually adopted[3][4]. Rete has a big advantage in efficiency but also has a severe disadvantage in scalability. It requires quite large memory spaces because it maintains all triples matched to each pattern and joined by more than one pattern in main memory. This makes it impossible for rete-based reasoning to process billions of triples.

To achieve scalability of rule-based reasoning, recent research such as WebPIE[1] utilized Hadoop, a distributed and parallelized computing framework. It showed that the performance of Hadoop-based reasoning is highly dependent on how to design mappers and reducers for each rule. So, it designed rule-specific mappers and reducers and succeeded to achieve web-scale reasoning. To eliminate duplicate derivation in early stage, it tried to get mappers to group input triples by considering the output of the rule, not joining key. It, in addition, maintained schema triples in main memory to improve load balances between parallelized computing nodes.

In this paper, we describe a new approach that first removes unnecessary pattern matches in rete framework by materializing rules based on given ontology schema, next fetches input triples selectively by implementing rule-specific input formats, and finally eliminates duplicate derivations by grouping rules having same output forms.

3 Rule Materialization

In our previous work[5], we applied dynamic materialization on some rules having genric patterns in RDFS and OWL semantics[6][7] and in this paper, we extend the materialization to all rules having schema triple patterns in RDFS and OWL semantics. Triples can be divided into two types: schema and instance triples. Schema indicates triples defining classes, relationships between them, and attributes related to them while instance means triples describing individuals, relationship between them,

Table 1. RDF and RDFS rules[6]

	id	entailment rules
rdf	1	(*u p v*) → (*p* rdf:type rdf:Property)
	2	(*u p v*) (if *v* is a XML literal and _:n is a bland node allocated to *v*) → (_:n rdf:type rdf:XMLLiteral)
rdfs	1	(*u p v*) (if *v* is a plain literal and _:n is a bland node allocated to *v*) → (_:n rdf:type rdfs:Literal)
	2	(*p* rdfs:domain *c*) (*u p v*) → (*u* rdf:type *c*)
	3	(*p* rdfs:range *c*) (*u p v*) → (*v* rdf:type *c*)
	4a	(*u p v*) → (*u* rdf:type rdfs:Resource)
	4b	(*u p v*) → (*v* rdf:type rdfs:Resource)
	5	(*p* rdfs:subPropertyOf *q*) (*q* rdfs:subPropertyOf *r*) → (*p* rdfs:subPropertyOf *r*)
	6	(*p* rdf:type rdf:Property) → (*p* rdfs:subPropertyOf *p*)
	7	(*p* rdfs:subPropertyOf *q*) (*u p v*) → (*u q v*)
	8	(*c* rdf:type rdfs:Class) → (*c* rdfs:subClassOf rdfs:Resource)
	9	(*c* rdfs:subClassOf *d*) (*u* rdf:type *c*) → (*u* rdf:type *d*)
	10	(*c* rdf:type rdfs:Class) → (*c* rdfs:subClassOf *c*)
	11	(*c* rdfs:subClassOf *d*) (*d* rdfs:subClassOf *e*) → (*c* rdfs:subClassOf *e*)
	12	(*p* rdf:type rdfs:ContainerMembershipProperty) → (*p* rdfs:subPropertyOf rdfs:member)
	13	(*c* rdf:type rdfs:Datatype) → (*c* rdfs:subClassOf rdfs:Literal)

and attributes related to them. Similarly, triple patterns forming rules can also be divided into two types: schema and instance triple patterns. Schema triples are generally small and static to a given ontology so as to be maintained in main memory while instance triples may continue to grow as much as not to be maintained in main memory. So, schema-only rules could be processed sufficiently on rete framework, but rules having instance triple patterns could not be processed on rete when the patterns are too generic.

For example, the generic triple pattern (*u p v*) of rdfs2 in Table 1 could be matched to all given triples, but only small part of them could be joined with specific triples matched to the remaining triple pattern (*p* rdfs:domain *c*) due to the common variable '*p*'. Indexing mechanisms such as hashing are usually applied to check such consistency efficiently, but they also require large memory spaces. More badly, as the target ontology grows, such indexing size also grows and may not be maintained in main memory. To solve this issue, we take following approaches according to types of rules:

- Schema-only rules (i.e., rdfs 5, 6, 8, 10, 11, 12, and 13, and owl-horst 9, 10, 12a, 12b, 12c, 13a, 13b, and 13c) are processed fully on rete framework.
- Generic-only rules (i.e., rdf 1 and 2, rdfs 1, 4a, and 4b, owl-horst 5a and 5b) are replaced and processed fully using dictionary which encodes all unique terms of input triples.

Table 2. OWL Horst rules[1][8]

id	entailment rules
1	(*p* rdf:type owl:FunctionalProperty) (*u p v*) (*u p w*) → (*v* owl:sameAs *w*)
2	(*p* rdf:type owl:InverseFunctionalProperty) (*u p w*) (*v p w*) → (*u* owl:sameAs *v*)
3	(*p* rdf:type owl:SymmetricProperty) (*u p v*) → (*v p u*)
4	(*p* rdf:type owl:TransitiveProperty) (*u p v*) (*v p w*) → (*u p w*)
5a	(*u p v*) → (*u* owl:sameAs *u*)
5b	(*u p v*) → (*v* owl:sameAs *v*)
6	(*u* owl:sameAs *v*) → (*v* owl:sameAs *u*)
7	(*u* owl:sameAs *v*) (*v* owl:sameAs *w*) → (*u* owl:sameAs *w*)
8a	(*p* owl:inverseOf *q*) (*u p v*) → (*v q u*)
8b	(*p* owl:inverseOf *q*) (*u q v*) → (*v p u*)
9	(*c* rdf:type owl:Class) (*c* owl:sameAs *d*) → (*c* rdfs:subClassOf *d*)
10	(*p* rdf:type rdf:Property) (*p* owl:sameAs *q*) → (*p* rdfs:subPropertyOf *q*)
11	(*u p v*) (*u* owl:sameAs *x*) (*v* owl:sameAs *y*) → (*x p y*)
12a	(*c* owl:equivalentClass *d*) → (*c* rdfs:subClassOf *d*)
12b	(*c* owl:equivalentClass *d*) → (*d* rdfs:subClassOf *c*)
12c	(*c* rdfs:subClassOf *d*) (*d* rdfs:subClassOf *c*) → (*c* owl:equivalentClass *d*)
13a	(*p* owl:equivalentProperty *q*) → (*p* rdfs:subPropertyOf *q*)
13b	(*p* owl:equivalentProperty *q*) → (*q* rdfs:subPropertyOf *p*)
13c	(*p* rdfs:subPropertyOf *q*) (*q* rdfs:subPropertyOf *p*) → (*p* owl:equivalentProperty *q*)
14a	(*c* owl:hasValue *v*) (*c* owl:onProperty *p*) (*u p v*) → (*u* rdf:type *c*)
14b	(*c* owl:hasValue *v*) (*c* owl:onProperty *p*) (*u* rdf:type *c*) → (*u p v*)
15	(*c* owl:someValuesFrom *d*) (*c* owl:onProperty *p*) (*u p v*) (*v* rdf:type *d*) → (*u* rdf:type *c*)
16	(*c* owl:allValuesFrom *d*) (*c* owl:onProperty *p*) (*u p v*) (*u* rdf:type *c*) → (*v* rdf:type *d*)

- Rules related to 'owl:sameAs' (i.e., owl-horst 6, 7, and 11) are replaced with *sameAs* table storing all same terms, defining a canonical term among them, and replacing all same term occurrences with their canonical ones.
- Remaining rules having combination of schema and instance triple patterns (i.e., rdfs 2, 3, 7, and 9, owl-horst 1, 2, 3, 4, 8a, 8b, 14a, 14b, 15, and 16) are processed first on rete to be materialized into schema-specific rules and then each materialized rule is processed on distributed and parallelized Hadoop framework.

The first and second ones are straightforward and the third one is similar to the approach of WebPIE[1]. So, the detailed explanation of them is omitted here. For the last one, our previous work[5] introduced rete-based framework that materializes some of the rules (i.e., rdfs 2, 3, 7, and 9, owl-horst 4, and 8a) into schema-specific rules and this paper extends the work into other rules in OWL Horst and incorporates Hadoop framework[9] additionally to deal with billions of triples.

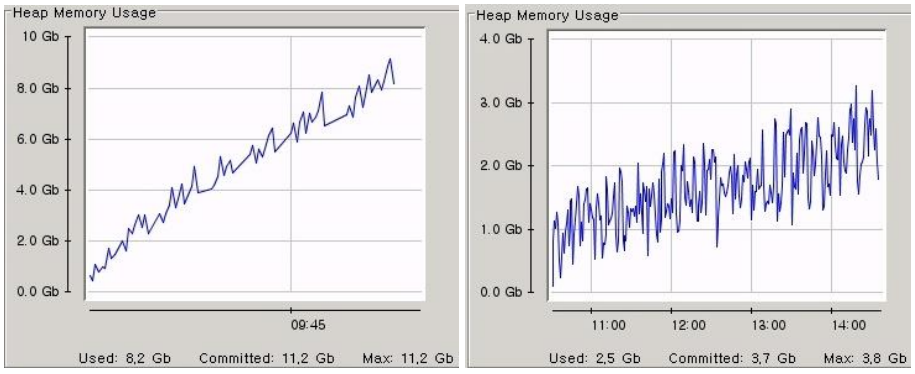
For example, when a given ontology defines *n* functional properties, $p_1,\ldots,p_n$, our rete-based reasoning framework will materialize the rule owl-horst1 into following *n* rules: (*u* p_i *v*) (*u* p_i *w*) → (*v* owl:sameAs *w*) (here, i = 1,…,n). These rules can be implemented in one pair of a mapper and a reducer as in WebPIE[1] but having p_i as parameters. In addition, when input triples are stored and indexed with six possible

combinations of subject, predicate and object using Hbase[10], one of column-based, no-SQL databases, the input format of the mapper can be implemented to selectively fetch triples only matched to the corresponding pattern. Finally, we can combine rules having a common conclusion (e.g., rdfs 2, 3, and 9) and implement one pair of a mapper and a reducer to efficiently eliminate duplicates derived from different rules.

4 Experiments

To demonstrate the feasibility of the proposed materialization approach, we first checked memory usages according to materialization. Fig. 1 shows that the memory without materialization is exhausted quickly even though the size of data is quite small, while the memory with materialization is consumed smoothly. We also compared the elapsed time in reasoning with and without materialization using LUBM[11]. The result in Fig. 2 shows that materialization improves the reasoning remarkably and even makes it linear to the size of data.

Especially, rete-reasoning without materialization consumed and exhausted memories very quickly even for small size of data. However, materialization solved this issue effectively by maintaining only schema triples in memory and leaving reasoning of instance rules to Hadoop framework.



(a) without materialization, LUBM(20) (b) with materialization, LUBM(1000)

Fig. 1. Memory usages with and without materialization

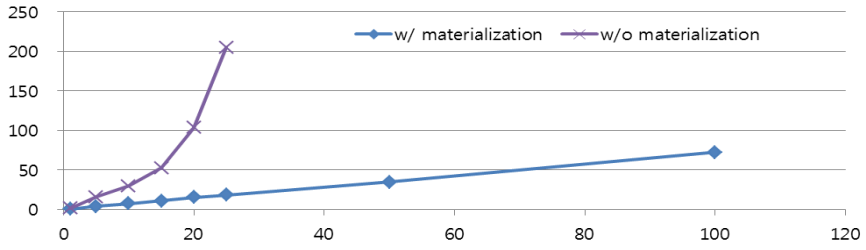


Fig. 2. The elapsed time in reasoning with and without materialization according to the size of data (LUBM)

5 Conclusion

This paper explained a rete-based reasoning method that materializes RDFS and OWL-Horst rules when an ontology schema is given and then can combine with Hadoop framework to deal with billions of triples. Each generic instance rule is materialized into several schema-specific rules, which can be implemented in a pair of a mapper and a reducer. Each mapper can selectively fetch input triples matched to its pattern using Hbase and rules having a common conclusion can be combined into a reducer to efficiently eliminate duplicate derivations from different rules.

The combination with Hadoop framework is being implemented and will be tested to check how much our method could improve reasoning performance, comparing to WebPIE[1] in near future.

Acknowledgement

This work was supported by the IT R&D program of MSIP/KEIT. [2014044034002, High performance database solution development for integrated big data monitoring and analysis]

References

1. Urbani, J., Kotoulas, S., Maassen, J., Harmelen, F., Bal, H.: WebPIE: A Web-scale Parallel Inference Engine using MapReduce. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 10, 59--75 (2012)
2. Forgy, C.L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence 19(1), 17--37 (1982)
3. Doorenbos, R.B.: Production Matching for Large Learning Systems. Ph.D Thesis, Carnegie Mellon University, Pittsburgh, PA (1995)
4. Özacar, T., Öztürk, Ö., Ünalir, M.O.: Optimizing a Rete-based Inference Engine using a Hybrid Heuristic and Pyramid based Indexes on Ontological Data. Journal of Computers 2(4), 41--48 (2007)
5. Lee, S., Jung H., Kim, P., You, B.-J.: Dynamically Materializing Wild Pattern Rules Referring to Ontology Schema in Rete Framework. In Proceedings of the 1st Asian Workshop on Scalable Semantic Data Processing (AS2DP) (2009)
6. RDF Semantics, available at: http://www.w3.org/TR/rdf-mt
7. OWL Web Ontology Language Semantics and Abstract Syntax, available at: http://www.w3.org/TR/owl-semantics
8. Horst, H.J.: Completeness, Decidability and Complexity of Entailment for RDF Schema and a Semantic Extension Involving the OWL Vocabulary. Journal of Web Semantics 3(2-3) 79--115 (2005)
9. Apache Hadoop, available at http://wiki.apache.org/hadoop.
10. Um, J.-H., Lee, S., Kim, T.-H., Jeong, C.-H., Seo, K., Park, J., Jung, H.: MapReduce-based Bulk-Loading Algorithm for Fast Search for Billions of Triples, In Proceedings of the 9th KIPS International Conference on Ubiquitous Information Technologies and Applications (CUTE 2014), (2014)
11. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 158--182 (2005)