Paper to

Seminar Peer-to-Peer Netzwerke

Summer term 2009

RDFS Reasoning and Query Answering in distributed environments

Supervisor: Liaquat Ali

July 2009

Christine Haas

Contents

1 Introduction

As the semantic web has become more and more popular there is a growing need for distributed RDFS reasoning to ensure scalability of Semantic Web applications. This paper focuses on DHT-based RDF stores and presents two alorithms - Forward Chaining and Backward Chaining - for RDFS reasoning and Query Answering in such distributed environments.

1.1 RDF

The Resource Description Framework (RDF) is a language for representing information about recources in the World Wide Web [1]. This is done by so-called RDF statements that consist of a subject, a predicate and an object. A RDF statement can also be written as a triple (s, p, o) where s is the subject, p the property and o the object of the statement.

1.2 RDFS and RDFS reasoning

An extension of RDF is the RDF Vocabulary Description Language: RDF Schema (RDFS) [2] which provides means to define classes, properties and hierarchies thereof.
Through RDFS reasoning it is possible to infer new knowledge from static information, for that it is a very important feature for Semantic Web Applications. For RDFS reasoning the RDFS entailment rules of RDF Semantics [3] are used. By applying these rules, new triples can be inferred in a RDF store.
In Figure 1 there are the main entailment rules for RDFS reasoning and computing of the transitive closure listed. The first four rules compute all the instances of a class, the next two rules compute the transitive closure of a property hierarchy and finally the last two rules the transitive closure of a class hierarchy.
So suppose the follwoing two triples are stored in a RDF store: $(artist, rdfs : subClassOf, person)$ $(painter, rdfs : subClassOf, artist)$
Then triple $(painter, rdfs : subClassOf, person)$

can be inferred by applying the rules for computing the transitive closure of class hierarchies (rule 7 and 8).

Rule	Head	Body
1	type(X,Y)	triple(X,rdf:type,Y)
2	type(X,Y)	triple(X,P,Z),triple(P,rdfs:domain,Y)
3	type(X,Y)	triple(Z,P,X), triple(P,rdfs:range,Y)
4	type(X,Y)	type(X,Z), subclass(Z,Y)
5	subProperty(X,Y)	triple(X,rdfs:subPropertyOf,Y)
6	subProperty(X,Y)	triple(X,rdfs:subPropertyOf,Z), subProperty(Z,Y)
7	subClass(X,Y)	triple(X,rdfs:subClassOf,Y)
8	subClass(X,Y)	triple(X,rdfs:subClassOf,Z), subClass(Z,Y)

Figure 1: RDFS entailment rules

2 RDFS Reasoning and Querying in distributed environments

In [4] there are two algorithms for RDFS reasoning and query answering presented that can cope with distributed environments: Forward Chaining and Backward Chaining. This chapter first introduces the storing protocol that both algorithms are built upon and then describes the algorithms in detail.

2.1 Storing protocol

Both algorithms are build upon DHT-based RDF stores. Distributed Hash Tables(DHT), which are common in many Peer-to-Peer Networks, are overlay networks which attempt to solve a simple lookup problem: Given a data item x, the goal is to find the node where x is stored. The storage protocol that is used was originally presented in [5]. Each RDF triple is stored three times in the DHT, once for each element of the triple.

2.2 Forward Chaining

Forward Chaining (FC) is a data driven approach where all triples are precomputed and sent to the network to be stored. Each time a triple is sent to a node to be stored, the node computes all inferred triples according to the entailment rules. Then the node sends the inferred triples to the network to be stored.
Figure 2 shows by a small RDFS class hierarchy how FC works. At the beginning, before running FC, only the RDF triples, that are not in bold, are inserted in the local databases of the nodes in the network whereas the triples in bold are the triples, that are inferred during the execution of FC. The underlined parts of the RDF triples are the keys, that lead to a specific node.
In the first iteration of FC, in the local databases of nodes $n2$ and $n6$ the new RDF triples, $(musician, rdfs : subClassOf, person)$ and $(musician, rdfs : subClassOf, person)$, can be inferred by applying entailment rules 7 and 8.
In the second iteration node $n2$ and node $n8$ both infer triple $(drummer, rdfs : subClassOf, person)$. This results in the fact, that the same triple is sent twice to the network to be stored. Generating redundancies is a severe drawback of FC. The algorithm stops after the third iteration because no new triples can be inferred in the local databases of the nodes.
Since all triples are precomputed the queries that query for all instances of a certain class C only need one message to the node that is responsible for class C. To achieve this, high storage load caused by generated redundancies and high network traffic while storing, have to be accepted. Also, before querying, a huge amount of triples are inferred and sent to the network to be stored without the garantee that these triples are needed in the querying part.

2.3 Backward Chaining

Backward Chaining(BC) is one solution to overcome the drawbacks of FC, is presented in [4]. In contrast to FC, Backward Chaining only evaluates RDFS

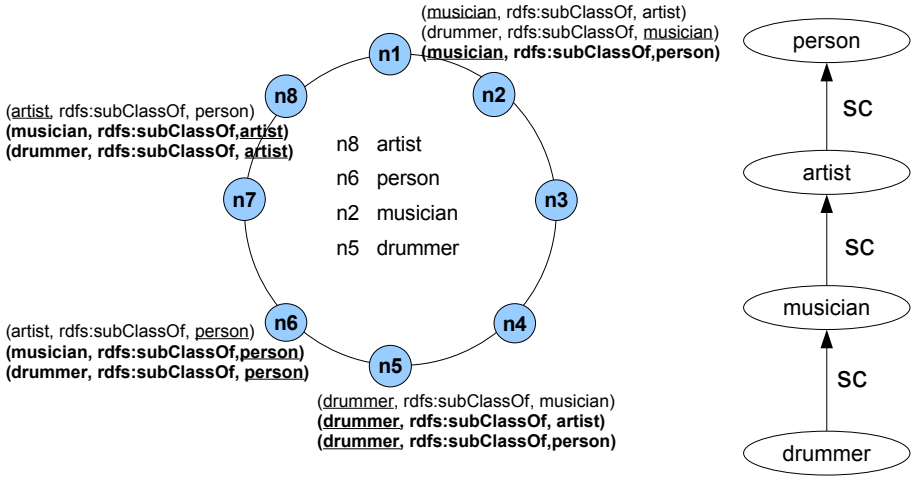

Figure 2: Example of FC

entailment rules at query processing time. The problem that arises is the processing of recursive rules in a distributed RDF store. To solve that problem, rule adornements are used to get a good order which subgoals in a collection of rules to evaluate first.

Definition 1. *An adornement of a predicate p with n arguments is an ordered string a of k's, b's and f's of length n, where k indicates an argument which is the key, b indicates a bound argument which is not the key and f is a free argument.*[4]

Figure 3 shows the adorned RDFS entailment rules.

Rule	Head	Body
1	$type^{kf}$ (X, Y)	$triple^{kbf}$ (X, rdf:type, Y)
2	$type^{kf}$ (X, Y)	$triple^{kff}$ (X, P, Z), $triple^{fbf}$ (P, rdfs:domain, Y)
3	$type^{kf}$ (X, Y)	$triple^{ffk}$ (Z, P, X), $triple^{fbf}$ (P, rdfs:range, Y)
4	$type^{kf}$ (X, Y)	$triple^{kbf}$ (X, rdf:type, Z), $subClass^{ff}$ (Z, Y)
5	$type^{fk}$ (X, Y)	$triple^{fbk}$ (X, rdf:type, Y)
6	$type^{fk}$ (X, Y)	$triple^{fff}$ (X, P, Z), $triple^{fbk}$ (P, rdfs:domain, Y)
7	$type^{fk}$ (X, Y)	$triple^{fff}$ (Z, P, X), $triple^{fbk}$ (P, rdfs:range, Y)
8	$type^{fk}$ (X, Y)	$type^{ff}$ (X, Z), $triple^{fbk}$ (Z, rdfs:subClassOf, Y)
9		
10	$subProperty^{kf}$ (X,Y)	$triple^{kbf}$ (X, rdfs:subPropertyOf, Y)
11	$subProperty^{kf}$ (X,Y)	$triple^{kbf}$ (X, rdfs:subPropertyOf, Z), $subProperty^{ff}$ (Z, Y)
12	$subProperty^{fk}$ (X,Y)	$triple^{fbk}$ (X, rdfs:subPropertyOf, Y)
13	$subProperty^{fk}$ (X,Y)	$subProperty^{ff}$(X, Z), $triple^{fbk}$ (Z, rdfs:subPropertyOf, Y)
14		
15	$subClass^{kf}$ (X,Y)	$triple^{kbf}$ (X, rdfs:subClassOf, Y)
16	$subClass^{kf}$ (X,Y)	$triple^{kbf}$ (X, rdfs:subClassOf, Z), $subClass^{ff}$ (Z, Y)
17	$subClass^{fk}$ (X,Y)	$triple^{fbk}$ (X, rdfs:subClassOf, Y)
18	$subClass^{fk}$ (X,Y)	$subClass^{ff}$(X, Z), $triple^{fbk}$ (Z, rdfs:subClassOf, Y)
19		

Figure 3: Adorned RDFS entailment rules

If an adorned RDFS entailment rule consists of two subgoals, BC chooses first the subgoal that has a k in its adornement. In any case, there will be such a predicate, since one of the arguments of the head predicate is going to be the key that lead to that certain node. So this body predicate with the k in its adornement can always be evaluated locally. After evaluating this body predicate there will be values returned that can be passed to the second body predicate.

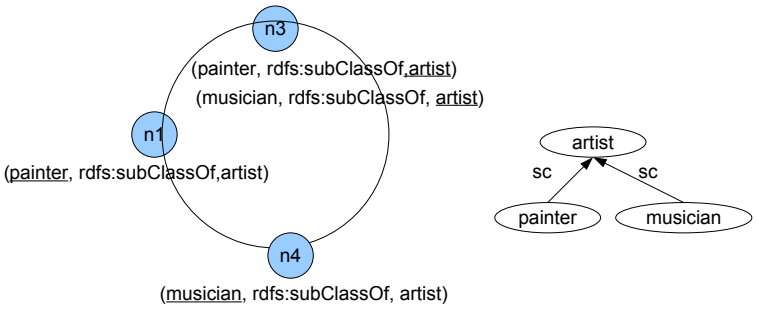

Figure 4: Example of a class hierarchy and how its triples can be stored

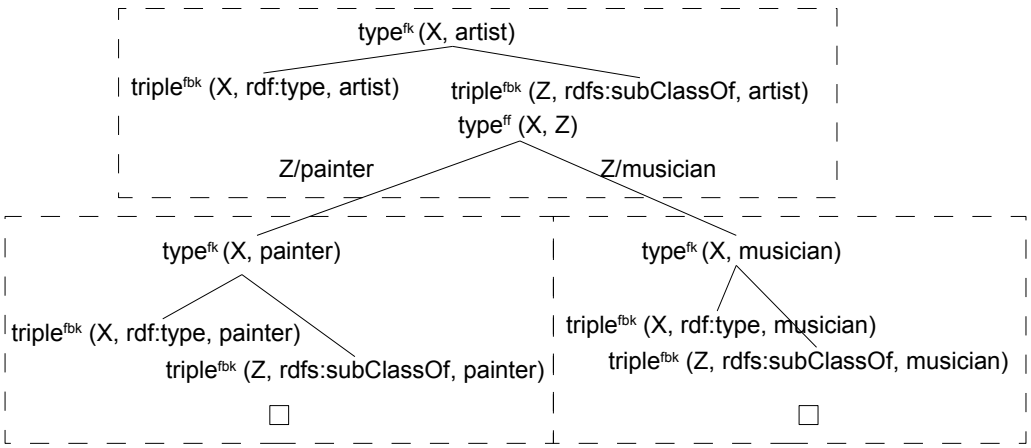

Figure 5: Proof tree of BC for example in Figure 4 and query for all instances of class artist

Figure 5 shows the proof tree of BC for the example in Figure 4. The task is to find all instances of class artist: $(X, rdf : type, artist)$. Since node $n3$ is the responsible node for key artist, the request to evaluate this tripple pattern will be sent to this node.
BC starts from the adorned predicate $type^{fk}(X, artist)$ in node $n3$. This predicate is checked against the adornement rules. Rules 5 and 9 can be found. Rule 5 can be evaluated locally in node $n3$, since it only has one predicate in its body. In rule 9 the predicate $triple^{fbk}(Z, rdfs : subClassOf, Y)$ is evaluated first, because it has a k in its adornement. So the two bindings for Z are painter

and musician. Then BC goes on in the two nodes responsible for keys painter ($n1$) and musician ($n4$). Since evaluation of RDFS entailment rules is only done at querying time, only these triples are inferred that are needed for querying. So only a constant number of messages are sent to the network while storing and storage load remain constant.

3 Comparison of FC and BC

In FC the network traffic increases as the depth of the RDF graph increases whereas in BC the number of messages that are sent is independent from the RDF schema.
The storage load in FC increases with the depth of the RDF graph because of the generated redundancies. In BC, only the triples are inferred, that are needed for query answering so the storage load remains constant. Query answering in FC only needs one step, since all inferred triples are precomputed before. Query processing time in BC depends on the depth of the RDF graph.

References

[1] *RDF Primer , W3C Recommendation 10 February 2004.* `http://www.w3.org/TR/rdf-primer/`

[2] *RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation 10 February 2004.* `http://www.w3.org/TR/rdf-schema/`

[3] *RDF Semantics , W3C Recommendation 10 February 2004.* `http://www.w3.org/TR/rdf-mt/`

[4] Kaoudi, Zoi ; Miliaraki, Iris ; Koubarakis, Manolis: RDFS Reasoning and query answering on top of DHTs.

[5] Cai, M. ; Frank, M.: RDFPeers: a scalable distributed RDF repository based on structured peer-to-peer network. (2004)