Journal of Southeast University (English Edition) Vol. 22, No. 3, pp. 435 – 438 Sept. 2006 ISSN 1003—7985

Parallel algorithm of generating closure of RDFS source

Miao Zhuang[1] Zhang Yafei[2] Lu Jianjiang[1] Song Zilin[1]

([1]Institute of Command Automation, PLA University of Science and Technology, Nanjing 210007, China)

([2]Training Department, PLA University of Science and Technology, Nanjing 210007, China)

Abstract: To decrease the time of generating a closure, a parallel algorithm of generating the closure of a resource description framework schema (RDFS) source is presented. In the algorithm, RDFS triples in the source are classified according to the forms of triples in the entailment rules and it reduces the scope of searching for specific triples. The dependence among the classes of triples is analyzed. Based on the classification, the initial RDFS source is partitioned into several subsets. The subsets are distributed to each process, and the closure is generated in parallel by applying the RDFS entailment rules. Parallel generating the closure of an RDFS source takes less time and increases efficiency.

Key words: parallel; closure; resource description framework (RDF) graph; RDFS source

The resource description framework (RDF) semantics specification provides a model-theoretic description of the semantics of RDF and resource description framework schema (RDFS)[1]. It also contains RDFS entailment lemma, describing a set of entailment rules. In use of these rules, the closure of an RDFS source can be generated and efficient reasoning services can be provided.

Several proposals have explored the generation of closure for inference on RDF models. Lassila used the entailment rules in the RDF semantics for RDF schema reasoning[2]. However, the approach is limited to a subset of the RDF schema semantics. The SWI-PROLOG semantic library also provides limited support for RDF schema reasoning[3]. In the infrastructure, the transitive closure of sub-class and sub-property relations can be accessed directly. Christophides et al.[4] presented a similar approach to creating special index structures for answering queries about hierarchical relations in RDF models. Broekstra proposed a pruning iterative forward chaining algorithm to compute the deductive closure of an RDF model[5]. The algorithm is implemented in the Sesame system[6].

As the size of RDFS sources becomes larger and larger, serially generating the closure takes too much time and memory. A better way is to generate the closure in parallel. In this paper, a parallel algorithm of generating the closure of an RDFS source is proposed to decrease time and increase efficiency.

Received 2006-04-10.

Foundation item: The Weaponry Equipment Foundation of PLA Equipment Ministry (No. 51406020105JB8103).

Biographies: Miao Zhuang (1976—), male, graduate; Zhang Yafei (corresponding author), male, doctor, professor, yf_zhang888@sina.com.

1 Background

1.1 RDFS entailment rules

RDFS extends RDF by assigning an externally specified semantics to specific resources. It is only because of these external semantics that RDF schema is useful. These semantics cannot be captured in RDF, whereas they can be revealed by applying RDFS entailment rules to derive new statements. The RDFS entailment rules are shown in Tab. 1.

By applying the rules in Tab. 1, more statements can be inferred and the semantics implied by an RDFS source are represented in the form of explicit statements.

1.2 Definition of closure

An RDFS source is a set of RDF(S) triples. A set of triples is defined as an RDF graph[7]. An RDF model can also be seen as a set of statements or as the graph induced by these statements. For reasons of convenience, we do not distinguish the term RDFS source, the RDF graph and the RDF model in the rest of this paper.

When talking about an RDF model, we find that it contains two types of statements, explicit and implicit. The explicit statements refer to the statements contained in the model. The implicit statements refer to what are contained implicitly, which can be made explicit by the rules in Tab. 1. We call the set of all statements, both explicit and implicit, the closure of a model.

Tab. 1 RDFS entailment rules

Rules	If T (the triple set) contains	Then add
1	$\langle s\ p\ l\rangle$, where l is a plain literal (with or without a language tag).	$\langle \_:n$ rdf: type rdfs: Literal$\rangle$, where $\_:n$ identifies a blank node allocated to l.
2	$\langle p$ rdfs: domain $c\rangle$, $\langle s\ p\ o\rangle$	$\langle s$ rdf: type $c\rangle$
3	$\langle p$ rdfs: range $c\rangle$, $\langle s\ p\ r\rangle$	$\langle r$ rdf: type $c\rangle$
4	$\langle s\ p\ o\rangle$ $\langle s\ p\ r\rangle$	$\langle s$ rdf: type rdfs: Resource$\rangle$ $\langle r$ rdf: type rdfs: Resource$\rangle$
5	$\langle p_1$ rdfs: subPropertyOf $p_2\rangle$, $\langle p_2$ rdfs: subPropertyOf $p_3\rangle$	$\langle p_1$ rdfs: subPropertyOf $p_3\rangle$
6	$\langle p$ rdf: type rdf: Property$\rangle$	$\langle p$ rdfs: subPropertyOf $p\rangle$
7	$\langle p_1$ rdfs: subPropertyOf $p_2\rangle$, $\langle s\ p_1\ o\rangle$	$\langle s\ p_2\ o\rangle$
8	$\langle c$ rdf: type rdfs: Class$\rangle$	$\langle c$ rdfs: subClassOf rdfs: Resource$\rangle$
9	$\langle c_1$ rdfs: subClassOf $c_2\rangle$, $\langle s$ rdf: type $c_1\rangle$	$\langle s$ rdf: type $c_2\rangle$
10	$\langle c$ rdf: type rdfs: Class$\rangle$	$\langle c$ rdfs: subClassOf $c\rangle$
11	$\langle c_1$ rdfs: subClassOf $c_2\rangle$, $\langle c_2$ rdfs: subClassOf $c_3\rangle$	$\langle c_1$ rdfs: subClassOf $c_3\rangle$
12	$\langle p$ rdf: type rdfs: ContainerMembershipProperty$\rangle$	$\langle p$ rdfs: subPropertyOf rdfs: member$\rangle$
13	$\langle s$ rdf: type rdfs: Datatype$\rangle$	$\langle s$ rdfs: subClassOf rdfs: Literal$\rangle$

Definition 1 The closure of an RDF graph G is the graph defined by the set of all triples that are implied by G. Denote the closure of a graph G as $c(G)$.

The RDF semantics specification suggests that an RDFS closure is defined to be the graph resulting from the following processes[1]:

① Add T to all the RDF and RDFS axiomatic triples;

② Apply rule lg(Literal generalization rule: if T contains *uuu aaa lll*, then *add uuu aaa* _ : *nnn*. Where _ : *nnn* identifies a blank node allocated to the literal *lll* by this rule.) to any triple containing a literal until the graph is unchanged by the rule;

③ Apply rules rdf2 and rdfs1 until the graph is unchanged;

④ Apply rule rdf1, rule gl (Literal instantiation rule: if T contains *uuu aaa* _ : *nnn*, then *add uuu aaa lll*.) and the remaining RDFS entailment rules until the graph is unchanged.

After the process above, all the implicit statements of an RDFS source can be presented explicitly in the form of new-derived statements. However, in the context of generating the closure of an RDFS source, it is enough to apply the RDFS entailment rules instead of the whole process above. Broekstra et al. proposed a pruning iterative forward chaining algorithm[5]. A similar idea is adopted to deal with the rules and a parallel algorithm is proposed to generate the closure.

2 Algorithm Design

2.1 Classifying RDFS triples

The algorithm in Ref. [5] did not distinguish among different triples and treated all those equally, thereby searching in the whole triple set for the statements needed. Given a certain rule with two premise statements, when one statement triggering the rule is captured, the other statement has to be searched in all of the triples, thus affecting the efficiency of the algorithm. In fact, for some rules, it is not necessary to search in all triples for the given statement. For instance, for rule 1, it is enough to search for the trigger statement like $\langle s\ p\ l\rangle$ instead of $\langle s\ p\ o\rangle$.

We modify the strategy of searching for the given triple. According to the forms of triples occurring in the entailment rules, we classify the RDFS triple set T into several subsets. The subsets are shown in Tab. 2.

Tab. 2 Types of triples in the entailment rules

Set T_i	Type of triples
T_1	$\langle s\ p\ l\rangle$
T_2	$\langle \_:n$ rdf: type rdfs: Literal$\rangle$
T_3	$\langle p$ rdfs: domain $o\rangle$
T_4	$\langle s\ p\ o\rangle$
T_5	$\langle s$ rdf: type $o\rangle$
T_6	$\langle s$ rdfs: range $o\rangle$
T_7	$\langle s$ rdf: type rdfs: Resource$\rangle$
T_8	$\langle s$ rdfs: subPropertyOf $o\rangle$
T_9	$\langle s$ rdf: type rdf: Property$\rangle$
T_{10}	$\langle s$ rdf: type rdfs: Class$\rangle$
T_{11}	$\langle s$ rdfs: subClassOf rdfs: Resource$\rangle$
T_{12}	$\langle s$ rdfs: subClassOf $o\rangle$
T_{13}	$\langle p$ rdf: type rdfs: ContainerMembershipProperty$\rangle$
T_{14}	$\langle p$ rdfs: subPropertyOf rdfs: member$\rangle$
T_{15}	$\langle s$ rdf: type rdfs: Datatype$\rangle$
T_{16}	$\langle s$ rdfs: subClassOf rdfs: Literal$\rangle$

The relation among 16 triple sets is as follows. T_1, T_3, T_5, T_6, T_8 and T_{12} are all the subsets of T_4. T_3, T_5, T_6, T_8 and T_{12} are not intersectant to each other. T_2, T_7, T_9, T_{10}, T_{13} and T_{15} are all the subsets of T_5 and not intersectant to each other. T_{14} is the subset of T_8.

Both T_{11} and T_{16} are the subsets of T_{12} and not intersectant to each other. After classifying the RDFS triples, the scope of searching for the triples triggering rules becomes smaller and thus efficiency is improved.

2.2 Partitioning the initial RDFS source

Before parallel generating the closure, the number of processors has to be determined. Our goal of designing the parallel algorithm is to decrease the execution time. We determine that every single processor deals with one entailment rule and, therefore, 13 processors are needed.

With respect to the parallel generation of the closure, partitioning the initial source and distributing them to each process is a basic step. It is not optimal to divide the RDFS triples into several subsets with the same size and distribute them to every processor. Each processor deals with one entailment rule, and dividing the triples averagely cannot ensure that all the triples needed by a given rule are stored in one individual processor, which leads to great number of messages among processes.

To decrease the number of messages, we analyze the minimal triple sets each process needs. Observing each entailment rule and the triple sets occurring in the rule, we can capture the dependence among the triple sets. The dependence is shown in Fig. 1. Where "Rule, T_j" below beelines or beside arcs with arrow denotes that applying rule i, a statement of the triple set in circle and a statement in T_j may derive a new statement belonging to the triple set pointed by the arrow. "Rule 4, (2)" denotes that by applying this rule, one statement can derive two new statements.

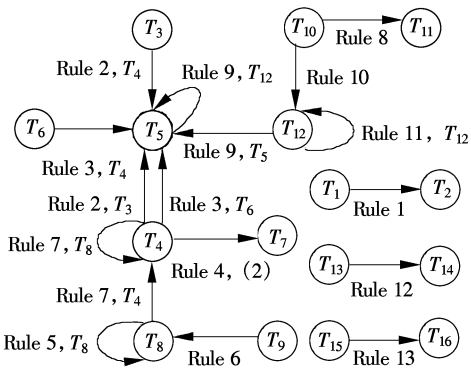

Fig. 1 The dependence among triple sets

In Fig. 1, we discover that for a certain RDFS source, the number of triples in some sets, for example, T_{13} and T_{15}, is constant and applying rules 12 and 13 once respectively is enough. Moreover, the sequence of applying these rules has no effect on the results of applying other rules.

According to the types of triple sets occurring in each RDFS entailment rule, we partition the RDFS triples into several subsets. For instance, T_3 and T_4 are the triple sets that trigger rule 2, so we distribute both T_3 and T_4 into the same triple subset, called D_2, and so on.

3 Parallel Generating a Closure

The key step of generating the closure of an RDFS source is applying the entailment rules to derive statements and exchange the statements needed by different rules among the processes. First, each process obtains the partitioned subset of initial RDFS triples and stores it locally. Secondly, applying the given rule, each process produces new statements. Thirdly, each process sends the new-derived statements to the processes requiring them and receives the statements from other processes by function MPI_SEND( ) and MPI_RECV( ) respectively. Finally, each process gathers all the triple subsets to the array RsClosure by function MPI_Gather( ). The parallel algorithm based on the message passing interface (MPI) is shown as follows:

```
Input: Subset of RDFS source D_i (i = 1, 2, ..., 13).
Output: The closure of an RDF graph (RsClosure).
/* Parameter: nproc (the number of process), myid (process label), newTri (the triple derived), recvTri (the triple from other processes) */
Methods:
{
  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &nproc);
  MPI_Comm_rank (MPI_COMM_WORLD, &myid);
  Data_input (D_i);
  MPI_Barrier (MPI_COMM_WORLD);
      /* Synchronize all processes */
  If (myid = =0)
      startwtime = MPI_Wtime ();
  new = 1;
  while (new ! =0)
      {
        if (triggered(myid + 1))
          {
          newTri = applyRule(myid + 1);
          /* Applying rule myid + 1, derive new triples */
          typeOfTriple = classify (newTri);
          /* Classify the new-derived triples */
          needproc = partition (typeOfTriple);
/* Determine the number and ID of the processes needing it */
          for (i = 0; i < needproc. num; i++);
          MPI_SEND (newTri, strlen (newTri), MPI_UNSIGNED_CHAR, needproc. id[i], i, MPI_COMM_WORLD);
/* Send the new-derived triple to the process needing it */
          }
        else
          new = 0;
        MPI_RECV (recvTri, MaxTriLen, MPI_UNSIGNED_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
```

```
        /* Receive the triples sent by other processes */
        If status = = MPI _ SUCCESS
          {
           typeOfTriple = classify ( newTri) ;
           /* Classify the triples received */
           add _ tri ( recvTri, setOfTri( typeOfTriple) ) ;
  /* Add triples received into corresponding triple set */
          new = 1;
          }
        else
          new = 0;
      }
If ( myid = = 0)
{
  MPI _ Gather ( sOfTri, RsClosure, MPI _ COMM _ WORLD) ;
  /* Gather all the triple sets distributed in each processor, obtaining the closure */
  endwtime = MPI _ Wtime ( ) ;
  printf ( "wall clock time = % f\n", endwtime - startwtime) ;
  Return RsClosure;
  }
  MPI _ Finalize ( ) ;
}
```

4 Conclusion

We present a parallel algorithm of generating the closure of an RDFS source. After classifying the triples, according to the dependence among all types of triple sets, we partition the initial RDFS source into subsets and distribute them to each process. For each process executing one RDFS entailment rule, the closure is generated in parallel. Classifying RDFS triples can reduce the scope of searching for the trigger statements, therefore, parallel generating the closure makes it possible to increase efficiency.

Our parallel algorithm is suitable for generating the closure of an RDFS source of large size. In the use of the algorithm, each processor has to store all triple sets that may trigger a given rule. With the increase of new-derived statements, the quantity of messages among processors grows large and thus performance decreases. The research on how to solve this problem is the future work.

References

[1] Hayes P. RDF semantics [EB/OL]. (2004-02-10) [2006-01-30]. http://www. w3. org/TR/2004/REC-rdf-mt-20040210/.

[2] Lassila O. Taking the RDF model theory out for a spin [A]. In: *Lecture Notes in Computer Science*[C]. Springer-Verlag, 2002, **2342**: 307 - 317.

[3] Wielemaker J, Schreiber G, Wielinga B. Prolog-based infrastructure for RDF: scalability and performance [A]. In: *Lecture Notes in Computer Science*[C]. Springer-Verlag, 2003, **2870**: 644 - 658.

[4] Christophides V, Plexousakis D, Scholl M, et al. On labeling schemes for the semantic web [A]. In: *Proceedings of the* 12*th International World Wide Web Conference* [C]. Budapest, Hungary, 2003. 544 - 555.

[5] Broekstra J. Storage, querying and inferencing for semantic web languages [D]. Amsterdam, Netherlands: Vrije University, 2005.

[6] Broekstra J, Kampman A, van Harmelen F. Sesame: a generic architecture for storing and querying RDF and RDF schema [A]. In: *Lecture Notes in Computer Science*[C]. Springer-Verlag, 2002, **2342**: 54 - 68.

[7] Hayes P. RDF model theory [EB/OL]. (2004-02-10) [2006-01-30]. http://www. w3. org/TR/2002/WD-rdf-mt-20020429/.

RDFS 数据源的并行闭包生成算法

苗 壮[1] 张亚非[2] 陆建江[1] 宋自林[1]

([1] 解放军理工大学指挥自动化学院,南京 210007)

([2] 解放军理工大学训练部,南京 210007)

摘要:为了减少闭包的生成时间,提出了一种 RDFS 数据源闭包的并行生成算法. 该算法基于 RDFS 推理规则生成闭包,根据三元组的形式对数据源中的 RDFS 三元组进行分类,缩小了特定形式三元组的直找范围;分析了各类三元组间的推理依赖关系,并按照每条推理规则所需的三元组将初始的数据源划分成多个子集,将子集导入各个并行的进程中,并应用 RDFS 推理规则并行地生成闭包. 并行的闭包生成算法可以有效地减少运行时间以提高闭包生成的效率.

关键词:并行;闭包;RDF 图;RDFS 数据源

中图分类号:TP182