Large Scale Temporal RDFS Reasoning Using MapReduce

Chang Liu[1] **Guilin Qi**[2] **Yong Yu**[1]
[1] Shanghai Jiao Tong University, {liuchang, yyu}@apex.sjtu.edu.cn
[2] Key Laboratory of Computer Network and Information Integration of State Education Ministry
School of Computer Science and Engineering, Southeast University, China, gqi@seu.edu.cn
Postal Address: 311 Yifu Building, 800 Dongchuan Road, Shanghai, China, 200240
Phone: 86-21-54745879

Abstract

In this work, we build a large scale reasoning engine under temporal RDFS semantics using MapReduce. We identify the major challenges of applying MapReduce framework to reason over temporal information, and present our solutions to tackle them.

Introduction

The Resource Description Framework (RDF) is one of the major representation standards for the Semantic Web, and RDF Schema (RDFS) is introduced as RDF's vocabulary description language.

Representing temporal information is crucial for many semantic web applications. People, for example, might live in different places over time, ontologies might have multiple versions, and semantic services might have temporal constraints. Currently, there have been many works considering the representation (Gutierrez, Hurtado, and Vaisman 2007) and the querying issue (Tappolet and Bernstein 2009)of temporal RDF. As discussed in (Tappolet and Bernstein 2009), without an efficient engine to reason with the temporal information, the exponential growth of the number of temporal relations will make the materialization of triple form unpractical. A recent work reported millions of triple with temporal information (Hoffart et al. 2011) and the number of triples with temporal information is still increasing. It is reasonable to expect more temporal RDF triples to be handled by semantic web applications. None of existing work, however, considered the large scale reasoning problem.

Urbani et al. have applied MapReduce techniques to perform large scale RDFS reasoning (Urbani et al. 2009). However, to reason temporal information, some optimizations for their MapReduce algorithms, such as the duplicate-prevention mechanism, are not applicable any more. Our previous work (Liu et al. 2011) has considered fuzzy values when applying MapReduce techniques. However, handling temporal information is more challenging than fuzzy information, as fuzzy values can be totally ordered but temporal intervals cannot. In this work, we first identify the some challenges of applying MapReduce algorithms to perform large scale temporal RDFS reasoning and then propose our solutions to tackle them.

Temporal RDFS

Syntax

A temporal triple is in form of $(s, p, o)[t, e]$ where (s, p, o) is a triple, and t and e are time points such that $t \leq e$. Intuitively a temporal triple $(s, p, o)[t, e]$ represents that the triple (s, p, o) is true in the time interval $[t, e]$. We allow $t = e$ to express a time point as a time interval. Furthermore, we allow t to be $-\infty$ and e to be $+\infty$ to express unbounded intervals such as $(-\infty, 0]$, $[0, +\infty)$, and $(-\infty, +\infty)$. A temporal RDF graph G is defined to be a set of temporal triples.

Semantics

Given a temporal RDF graph G, and a time point t, we define a *time projection* to be an RDF graph G_t such that $G_t = \{(s, p, o) : \exists (s, p, o)[l, e] \in G, t \in [l, e]\}$. We call a mapping M from a time point $t \in \mathbb{Z}$ to an interpretation as an *RDFS interpretation mapping*, if for all $t \in \mathbb{Z}$, $M(t)$ is an RDFS interpretation. Given an RDFS interpretation mapping M, we say that M satisfies G if for all $t \in Z$, $M(t)$ satisfies G_t. For the space limitation, we do not give the definition of an RDFS interpretation. The formal discussion can be found in our technical report[1].

Entailment Rules

Given two temporal RDF graphs G_1 and G_2, we say that G_1 temporal RDFS entails G_2, denoted as $G_1 \models_{tD} G_2$, if any RDFS interpretation mapping M that satisfies G_1, also satisfies G_2. There is a sound and complete entailment rule set for the temporal RDFS semantics. A typical rule is listed in the following:

t-rdfs2 $\quad (p, \texttt{domain}, u)[t_1], (v, p, w)[t_2] \Rightarrow (v, \texttt{type}, u)[t_1 \cap t_2]$

Here `domain` and `type` are RDFS vocabularies. t_1 and t_2 are two time intervals, and $t_1 \cap t_2$ calculates the intersection of them. The *temporal RDFS closure* of a temporal RDF graph G can be computed by iteratively applying the temporal RDFS rules until the fixpoint is reached. Given

[1] Available at `http://apex.sjtu.edu.cn/apex_wiki/temporal`

G_1 and G_2, the entailment relation can be determined by checking if G_2 is a subgraph of the temporal RDFS closure of G_1.

Large Scale Temporal RDFS Reasoning

We are able to encode a temporal RDFS entailment rule into a MapReduce program similar to the treatment of RDFS entailment rules given in (Urbani et al. 2009). This kind of implementation, however, will suffers some challenging problems which make the large scale temporal reasoning impossible. In this section, we first identify these challenging problems, and then we present our solutions to tackle them.

Challenges and solution

Duplicates The straightforward implementations will result in many duplicates. As a result, the graph will expand greatly. Storing the huge size of graph may lead to both a waste of storage space and a waste of computation time. The same problem appears when dealing with two fuzzy triples with the same triple part. For fuzzy domain, we can simply store the larger fuzzy degree. For temporal domain, however, there is no full order so that we are not always able to compare two time intervals. In the following we will introduce a compact representation of the time intervals to tackle this problem.

Efficient Implementation The naive implementation will encounter some problems and violate the principles to develop efficient MapReduce programs. For example, the naive implementation of rule t-rdfs2 will introduce an unnecessary shuffling process and will treat the values in the *reduce* function as a set instead of a stream.

Shortest Path Calculation In temporal RDFS semantics, there are two rules, i.e. t-rdfs5 (subproperty) and t-rdfs11 (subclass), that require iterative calculations. When we treat each temporal triple as a weighted edge in the RDF graph, calculating the closure by applying these two rules is essentially a variation of the all-pairs shortest path calculation problem. Unlike the fuzzy domain which has a total order, the domain of time interval has only a partial order. Thus, we need to design new efficient algorithms for this problem.

Compact Representation

We use a compact representation to store temporal triples. Given a set of temporal triple $(s, p, o)[t_i, e_i]$, for $i = 1, ..., k$ such that $t_i \leq e_i < t_{i+1}$ holds for all $1 \leq i < k$, we store them as $(s, p, o)[l]$ where $l = (t_1, e_1), ..., (t_k, e_k)$. Furthermore t_1 can be $-\infty$ and e_k is allowed to be $+\infty$. We call l a *compound time interval*.

In such compact representation, the *intersection* and *union* operations of two compound time intervals can be implemented by algorithms with linear time complexity.

Loading schema triples into memory

As discussed above, a rule, e.g. t-rdfs2, can be encoded into a MapReduce program. However, there are two drawbacks in the naive implementation. Firstly, there is a very expensive shuffling process before the reducers are launched. Secondly, in each reducer, we have to load all values of key/value pairs generated by mappers with the same key into the memory. If there are too many values, e.g., the number of (v, w) for a given p is too large, the reducer can easily fail.

We find that in the condition of each of these rules, there is at least one temporal triple that is a schema triple. In practice, we can assume that the number of schema triples is relatively small. Therefore, we can load them into the memory, and perform the join in the mappers such that we can avoid the above two drawbacks.

Calculating the class and property hierarchy

It is easy to see that rule t-rdfs11 needs to recursively calculate the transitive closure of `sc` triples, e.g. triples with `sc` as their predicates. However, since they are schema triples, as we have discussed previously, we can load them into the memory in order to avoid the reiterative execution of the MapReduce program of rule t-rdfs11. Furthermore, we can see that calculating the `sc` closure by applying rule t-rdfs11 is indeed a variation of the all-pairs shortest path calculation problem. We can develop an algorithm to calculate the closure within $O(n^4)$ where n is the number of `sc` triples. Even though the complexity seems a little high, the algorithm runs very fast in real applications.

Conclusion

In this work, we presented the first work to handle the problem of large scale temporal RDFS reasoning using MapReduce. We identified some unique challenges and gave solutions to tackle them. In the future, we will conduct experiments to study the performance of our engine.

Acknowledgement

Guilin Qi is partially supported by NSFC (61003157), Jiangsu Science Foundation (BK2010412), Excellent Youth Scholars Program of Southeast University, and Doctoral Discipline Foundation for Young Teachers in the Higher Education Institutions of Min- istry of Education (No. 20100092120029).

References

Gutierrez, C.; Hurtado, C. A.; and Vaisman, A. 2007. Introducing time into rdf. In *Proc. of TKDE' 07*, 207 – 218.

Hoffart, J.; Suchanek, F. M.; Berberich, K.; Lewis-Kelham, E.; de Melo, G.; and Weikum, G. 2011. Yago2: exploring and querying world knowledge in time, space, context, and many languages. In *Proc. of WWW' 11*, 229–232.

Liu, C.; Qi, G.; Wang, H.; and Yu, Y. 2011. Large scale fuzzy pD^* reasoning using mapreduce. In *Proc. of ISWC' 11*, 405–420.

Tappolet, J., and Bernstein, A. 2009. Applied temporal rdf: Efficient temporal querying of rdf data with sparql. In *Proc. of ESWC' 09*, 308–322.

Urbani, J.; Kotoulas, S.; Oren, E.; and Harmelen, F. V. 2009. Scalable distributed reasoning using mapreduce. In *Proc. of ISWC' 09*, 374 – 389.