The cost of reasoning with RDF updates

Sana Al Azwari John N. Wilson
Department of Computer & Information Sciences, University of Strathclyde, Glasgow, UK,
{sana.al-azwari,john.n.wilson}@strath.ac.uk

***Abstract*—Many real world RDF collections are large compared with other real world data structures. Such large RDF collections evolve in a distributed environment. Therefore, these changes between RDF versions need to be detected and computed in order to synchronize these changes to the other users. To cope with the evolving nature of the semantic web, it is important to understand the costs and benefits of the different change detection techniques. In this paper, we experimentally provide a detailed analysis of the overall process of RDF change detection techniques namely: explicit change detection, forward-inference change detection, backward-inference change detection and backward-inference and pruning change detection. The results show that pruning is relatively expensive by comparison with inferencing.**

I. Introduction

Resource Description Framework (RDF) is an annotation language that provides a graph-based representation of information about Web resources in the Semantic Web. Because RDF content is shared between different agents, a common interpretation of the terms used in annotations is required. This interpretation is typically provided by an ontology expressed as RDF Schema (RDFS) or Web Ontology Language (OWL). The schema provides additional semantics for the basic RDF model. Changes in the domain that are reflected by evolution of the ontology may require changes in the underlying RDF data. Since RDF is designed to easily integrate large data collections from diverse data sources, changes to RDF may occur in locations distributed across a network. In a client-server environment, an on-demand update service would require efficient generation and transfer of RDF updates. Alternatively in a peer-base network where RDF is updated locally, the updates would also need to be generated using the least possible resources. In both of these scenarios, it is necessary to establish the most cost-effective way of generating RDF updates.

Initial proposals for change detection schemes are based on comparing two RDF graphs and computing the differences between them to generates a set of differences (*delta*). These differences represent a set of change operations (i.e. insertions and deletions) that transform one RDF graph into another [1]. This change detection method generates the deltas using set arithmetic for RDF graphs. For example, If M and M' are RDF models, then the delta that transforms M to M' is modelled as a set of triple insertions and triple deletions where insertions is the set difference $M' - M$ and deletions is $M - M'$.

This is a straightforward technique. However, it only handles the syntactic level of RDF and generates the explicit differences between two RDF models. In addition, the cost of storing the delta or passing it between nodes in a distributed environment using this method is linear with the size of the differences between two RDF models [1]. Therefore, several approaches for calculating the delta have been proposed based with a view to minimizing the delta size in order to reduce the required bandwidth and storage space for updating RDF data collections [5], [8], [2], [4]. These approaches aim to minimize the delta size by exploiting the semantics of RDF data. The minimisation process requires the application of RDF inference rules under the RDFS specification to the triple set [3]. This RDF closure process results in the derivation of new triples. Considering this closure (i.e. inferred set of triples) when calculating the set-differences between two RDF models may reduce the size of the produced delta.

The basic operations in change detection techniques are the set-difference and inference operations. A classification of change detection techniques can be based on the inference strategy (i.e. the computation of the closure). The existing strategies are Explicit delta (ΔE); Closure delta (ΔC); the Dense delta (ΔD); the Dense & Closure delta (ΔDC)and the Explicit & Dense delta (ΔED). These approaches are inference based strategies apart from the ΔE which is syntactic-based. Due to limited space in this paper, only ΔE and ΔED are explained since these techniques are used extensively in the experimental methodology.

As explained above, ΔE calculates the delta in the syntactic level of RDF. Given two RDF models M and M' each of which represents set of triples $t(SPO)$, the ΔE that transforms M to M' is the set of triple insertions and triple deletions that is computed by:

$$\begin{aligned}\Delta E(M, M') = \{del(t) | t \in M - M'\} \\ \cup \{Ins(t) | t \in M' - M\}\end{aligned} \quad (1)$$

In contrast, ΔED, obtains a set of insertion triples by performing the set-difference operation $M' - M$. The set of deletions is obtained by computing the closure of M' (denoted as $C(M')$) first and then performing the set-difference operation. ΔED is computed as follows:

$$\begin{aligned}\Delta ED(M, M') = \{del(t) | t \in M - C(M'))\} \\ \cup \{Ins(t) | t \in M' - M)\}\end{aligned} \tag{2}$$

The order of the basic set-difference and inference operations can be used to categorise RDF change detection methods as forward-inference or backward-inference approaches. The forward-inference approach follows the inference-then-difference strategy which, in the case of ΔED, computes the entire closure of M' first and then calculates the set-differences. In contrast, the backward-inference approach uses the difference-then-inference strategy. That is, instead of computing the entire closure of M', in the case of ΔED, this method calculates first the set-differences $M - M'$ and $M' - M$, and then checks every triple $t \in (M - M')$ and removes it if it can be inferred in the M' set as follows:

$$\text{Remove } t \text{ from } (M - M') \text{ if } t \in C(M') \tag{3}$$

Therefore, instead of pre-computing the full closure in advance, this method infers only triples related to the result of $(M - M')$. This would be expected to improve the time and space required in change detection in comparison with forward inferencing models. As explained above, the calculation of RDF closure is based on applying the RDFS entailment rules provided by the RDFS semantics specification. The entailment rules infer new RDF statements based on the presence of other statements. Only the rules that play a crucial roles in minimizing the delta size are used [4].

However, although the backward-inference method is applied to infer only relevant triples, some triples might be unnecessary for change detection. This provides an opportunity for pruning unnecessary triples prior to invoking backward inferencing [4]. Pruning allows for the skipping of some irrelevant triples derived from the set-difference operation $(M - M')$ during change detection with a consequent potential improvement in efficiency. After pruning, the remaining triples are checked to see if they can be inferred in the other set. By contrast to the approach presented in [2]. The proposed change detection technique follows *difference-pruning-inference strategy*. The general rule for pruning is that if the subject or object of a triple does not exist in M' then this triple cannot be inferred in M', therefore, this triple is pruned before the inference process begins.

The semantic content of RDF data can be exploited by both pruning and inferencing to provide opportunities for improving the performance of updates to RDF data collections. However the relative merits of these approaches are not clear. Where data sets are large and subject to frequent updates, both processes may require significant computing time. The contribution of this paper is an analysis of the costs and benefits of performance improvements by pruning and inferencing respectively when these operations are carried out over large real-world data sets.

II. Related work

Existing RDF change detection tools include Prompt-Diff [5] which compares the structure of RDF versions based on heuristic matchers. Although this method overcomes the problem of different serialization of RDF data it does not handle the semantic property of RDF. Several change detection approaches have been proposed to handle the semantic level of RDF and these provide the opportunity of minimizing the delta size [8], [1], [9]. Since most of the data in RDF versions remains unchanged [6], an alternative approach to forward-chaining inference is the backward-chaining inference [2] which instead of computing the full closure of RDF model, computes only the relevant triples. Choosing backward inference over forward inference is a trade-off between storage space and query processing, respectively. Hybrids that combine the benefits of both strategies have also been explored [7]. There is also evidence that pruning the delta can reduce its size in the context of backward-chaining inference strategies [4].

In general, approaches to managing RDF versions are focused on part of the problem such as minimizing delta size or the number of inferred triples rather than the overall performance of the update process. An efficient approach for updating RDF knowledge bases that is designed with the overall performance in mind provides an approach to addressing the growing size and complexity of RDF data stores. In the light of these challenges, the overall aim of this work is to gain insight into the strengths and weaknesses of different approaches to managing RDF updates by providing a detailed experimental analysis of their performance. Its contribution is a comparative study of different strategies for using semantic content to generate RDF updates in the context of the overall performance of the process.

III. Approach

This work provides a detailed analysis of different change detection techniques including: explicit change detection (denoted as EC), forward-inference change detection (denoted as FC), backward-inference change detection (denoted as BC) and pruning-and-backward-inference change detection (denoted as PBC). EC performs ΔE for computing the deltas, while FC, BC and PBC perform ΔED.

The RDF triple store was based on the properties subClassOf, subPropertyOf, Type, and Triple which holds triples that contain any other property type. In addition, there are three extra tables which store the result of performing the set-difference operations $M - M'$ and $M' - M$, these tables are Del table and Ins table respectively, and the third table is Inf to store inferable triples. Using these tables, changes between two RDF models are detected and updated as follows:

First, the differences between M and M' are computed using the set-difference operations $M - M'$ and $M' - M$, the result is stored in the Del table and the Ins table, respectively.

Next, in the case of ED, the triples in the Del table and the Ins table are the delta. Therefore, all the triples in the Del table are removed from M, and all the triples in the Ins table are inserted into M. This step transforms M to M'.

The computation of delta in FC follows the *inference-than-difference strategy*. Therefore, before computing the differences and updating M, there is an inference process that involves the calculation of the full closure in M'.

In the case of BC, a *difference-than-inference strategy* is followed so the differences between the two models are computed first and instead of calculating the full closure of M' the inference process checks only the triples in the Del table if it can be inferred in M' by applying the inference rules explained in Section1. If any triple is inferred in M' this triple is removed from the Del table.

In contrast, in PBC, prior to the inference process, some of the triples in the Del table may be pruned. The pruning process checks every triple in the Del table if both the subject and the object of the triple is existed in M' as a subject and object, respectively, then the triple may be able to be inferred in M', and therefore this triple is inserted to the Ins table. After the pruning process, only the triples in the Inf table are included in the inference process and not all the triples in the Del table. It is worth mentioning that not all the rules are applied to each triple, but only the rules that correspond to the property of the triple.

IV. Results and discussion

All experiment were performed on Intel(R) Xeon(R) CPU X3470 @ 2.93GHz - 1 cpu with 4 cores and hyperthreading, Ubuntu 12.04 LTS operating system and 16GB memory. All change detection techniques were implemented using Java with Jena. In order to achieve accurate results, Just In Time (JIT) compilation and garbage collection were controlled. All measurements are averages of elapsed time for ten independent runs recorded after a sequence of five runs to control class loading.

MySQL was used to store RDF versions and deltas. The data set used contained both the Gene Ontology(GO) vocabulary and associations between GO terms and gene products including the Uniprot TaxonomyThis data set was chosen because it is frequently updated with a new version of it being released every month. The data set includes the first release from each year between 2005 and 2014. Annual versions were chosen to increase the gap between versions and therefore get more differences between the RDF triple collections. Using this data set, the oldest version (i.e.the 2005 version) was transformed to the other different versions starting from the 2006 version and ending with the 2014 version. This gradually increase the number of differences between the versions when measuring the delta size with a consequent effect on the performance of the different change detection methods.

	(1)		**(2)**		
	Delta Size		Triples used in reasoning		
year-range	ΔE	ΔED	FC	BC	PBC
2005-2006	43136	42770	45400	817	562
2005-2007	116710	116228	52449	1457	888
2005-2008	189253	188512	59995	2880	1434
2005-2009	210372	209334	66264	4086	1969
2005-2010	237510	236190	74389	4754	2433
2005-2011	265609	264221	81538	5513	2790
2005-2012	308594	307163	87751	5895	2883
2005-2013	348819	347292	99425	6735	3471
2005-2014	367233	365629	104209	7080	3638

TABLE I: Triple statistics. (1) The total number of changes collected over the period. (2) The count of triples participated in the inference process

In Table I column 1 shows the delta size generated by the EC method using ΔE. The delta is defined as a set of deleted triples and inserted triples to be applied to the base data structure to convert it into the updated structure . Since the inference-based approaches (FC, BC and PBC) use the ΔED, the delta reductions produced by these methods are identical. Reasoning over RDF dataset reduces the size of the delta (Table I column 1). Table I column 2 shows the number of triples participated in the inference process. FC has the highest number of triples used in the inference process as a result of calculating the full closuer. This number is reduced by $\sim$94% when changes were detected using BC. PBC, on the other

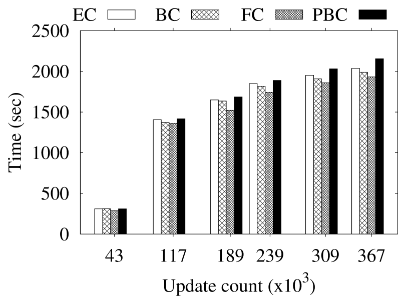

Fig. 1: Total update time

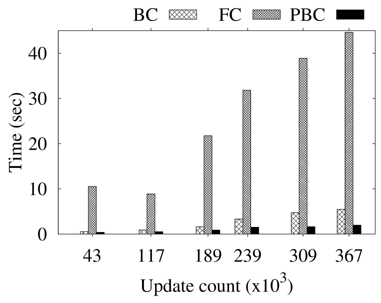

Fig. 2: Inference time

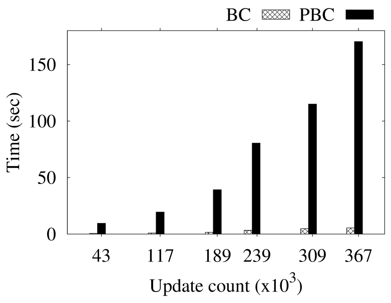

Fig. 3: Total reasoning time

hand, can further prune ∼47% of these triples in BC.

In addition to the delta size and the number of triples in the inference process, the performance of the four approaches was also analysed by measuring their execution time. Reasoning time consists of the time to complete both inferencing and pruning. Change detection time is the sum of set difference time and reasoning time. In addition to change detection, the time to apply the changes to the original data set (application time) was also recorded. Figure 1 shows that the total time (consisting of the sum of all components including application time increases with the size of the update set. From Figure 2 it can be seen that the inference time in PBC exceeds that of BC by about 1.4-2.7 times , and of FC by about 18.7-32.5 times as a result of the pruning process. However, although pruning RDF has efficiently reduced the inference time, the pruning operation in PBC adds to the overall reasoning time (and consequently change detection) as shown in Figure 3. Moreover, the prunning time appears to increase significantly with the increase in the number of pruned triples. Results from these experiments show that change detection using PBC has a penalty of about 17.3-31.1 by comparison with BC when pruning time is counted as part of the reasoning process.

V. Conclusion and future work

Updating RDF is a crucial problem and as the understanding of the update process grows, new approaches for its optimization also emerge. The work presented in this paper contributes to this growing understanding by presenting an in depth analysis of the update process. Based on our experimental results, we found that pruning RDF has a significant effect on reducing the inference time. However, pruning the RDF requires additional time for applying the pruning rules, and this time increases as the size of the structural differences between the RDF versions increases. The inference time is decreased ∼95.8% in BC than in PBC where a pre-inference computation is required for pruning. Therefore, this time increases the change detection time and the RDF update overall time. Moreover, the delta size is still the same as pruning RDF does not reduce the amount of data that need to be stored and exchanged over the network.

The results reported here use RDF/S as a platform for reasoning. The richer semantic context available in OWL ontologies provides the opportunity for further reductions in the delta size. Taking advantage of the time consumed by pruning to apply more complex rules could further reduce the amount of data to be stored and exchanged which is the aim of our future work. In addition, since the GO data set does not contain "subPropertyOf" relationship, use synthetic data to evaluate the performance of the change detection approaches using different change ratios. Moreover, each relationship, such as "subClassOf", "subPropertyOf" and "Type", will be analysed separately to measure the complexity of the inference rules related to these relationships.

References

[1] T. Berners-Lee and D. Connolly. Delta: an ontology for the distribution of differences between RDF graphs, 2004.

[2] J. Broekstra and A. Kampman. Inferencing and truth maintenance in RDF schema. *PSSS*, 2003.

[3] P. Hayes and B. McBride. RDF semantics, 2004.

[4] D. Im et al. Backward inference and pruning for RDF change detection using rdbms. *J. Info. Science*, 39(2):238–255, 2013.

[5] N. Noy and M. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. *AAAI/IAAI*, 2002:744–750, 2002.

[6] N. Noy and M. Musen. Ontology versioning in an ontology management framework. *Intelligent Systems*, 19(4):6–13, 2004.

[7] W. Shen and Y. Qu. An RDF storage and query framework with flexible inference strategy. In *Frontiers of WWW Research and Development-APWeb 2006*, pages 166–175. Springer, 2006.

[8] M. Völkel and T. Groza. Semversion: An RDF-based ontology versioning system. In *Proc. IADIS Int. Conf. WWW/Internet*, volume 2006, page 44, 2006.

[9] D. Zeginis et al. On computing deltas of RDF/S knowledge bases. *ACM Trans on the Web (TWEB)*, 5(3):14, 2011.