C-Set : a Commutative Replicated Data Type for Semantic Stores

Khaled Aslan, Pascal Molli, Hala Skaf-Molli, Stéphane Weiss

C-Set : a Commutative Replicated Data Type for Semantic Stores

Khaled Aslan[1] and Pascal Molli[1] and Hala Skaf-Molli[1] and Stephane Weiss[2]

[1] University of Nantes, France
[2] INRIA Rennes, France

Abstract. Web 2.0 tools are currently evolving to embrace semantic web technologies. Blogs, CMS, Wikis, social networks and real-time notifications, integrate ways to provide semantic annotations and therefore contribute to the linked data and more generally to the semantic web vision. This evolution generates a lot of semantic datasets of different qualities, different trust levels and partially replicated.
This raises the issue of managing the consistency among these *replicas*. This issue is challenging because semantic data-spaces can be very large, they can be managed by autonomous participants and the number of replicas is unknown.
A new class of algorithms called *Commutative Replicated Data Type* are emerging for ensuring eventual consistency of highly dynamic content on P2P networks. In this paper, we define C-Set a CRDT specifically designed to be integrated in Triple-stores. C-Set allows efficient P2P synchronisation of an arbitrary number of autonomous semantic stores.

Keywords: *scalability, synchronisation, peer-to-peer, consistency, replication*

1 Introduction

Web 2.0 tools are currently evolving to embrace semantic web technologies. Blogs, CMS, Wikis, social networks and real-time notifications, integrate ways to provide semantic annotations and therefore contribute to the linked data [1] and more generally to the semantic web vision. This evolution generates a lot of semantic datasets of different qualities, different trust levels and partially replicated.

The problem of updating and synchronizing data in the semantic web has been raised by Berners-Lee in [10]. Efficient synchronization of semantic stores organized in a peer-to-peer network can leverage problems of scalability and allows different autonomous participant to collaboratively combine, improve and enrich semantic datasets.

Synchronizing semantic stores is challenging. Semantic data-spaces can be very large, they can be managed by autonomous participants and the number of replicas is unknown, potentially high. Only few replication algorithms can fulfill these constraints and they all belong to the optimistic replication class

of algorithms [9]. An optimistic replication model considers an unknown number of sites, each site has a copy a shared object. An object can be modified by applying locally an operation. Next, this operation is broadcasted to other sites in order to be re-executed. The system is correct if it ensures the eventual consistency property [5] i.e. all replicas are identical when the system is idle. Thomas write rule [5] was the first algorithm to ensure eventual consistency in duplicated databases. However, Thomas write rule requires the knowledge of the number of participants (in order to provide a safe garbage collection scheme). This constraint is not compatible with our context.

Recently, a new class of optimistic replication algorithms called CRDTs is emerging [7,14]. CRDT stands for *Commutative Replicated Data Types*. CRDTs propose to define new abstract data type that provides commutative operations. CRDTs ensure eventual consistency regardless of the order of operations at reception. CRDT has been defined for arrays, linear sequence and tree.

In this paper, we define a specific data type for sets called C-Set that can be applied for a triple store and we prove that this type ensures eventual consistency. With a traditional set, operations insert/delete do not commute. C-Set provides commutative insert, delete operations, does not require causal reception of operation and can leverage some problems of garbage collection. C-Set is designed to be integrated within a semantic store in order to provide P2P synchronisation of autonomous semantic store.

The paper is organized as follow: Section 2 presents backgrounds and related works. Section 3 defines a new CRDT type for set designed for triple stores. Section 4 discusses our approach. Section 5 summarizes contributions and presents future work.

2 Background and related work

Many previous work on replication in semantic P2P systems focused on sharing RDF resources. They did not enable collaborative working for maintaining RDF stores. In the context of data sharing, some sites publish their RDF data while other sites consume this data. So there is no concurrent modifications and/or operations. While collaborative systems consider a shared object between the peers. This imply that the peers can modify the object concurrently and then they can run a consolidation algorithm to ensure the consistency of the shared object.

Tim Berners-Lee and Dan Connolly proposed an ontology for the distribution of differences between RDF graphs called *Delta* [10]. In their approach they rely on the standard serialization of RDF graphs into text files then running a *diff* between the resulted text files. However, the authors do not detail how eventual consistency can be efficiently reached by their algorithms.

RDFGrowth [12] and Publish/Subscribe Networks [4] focus on semantic data sharing where only one peer can modify the shared knowledge while others can read them. However, as it was mentioned earlier, sharing is different from collaboration. In sharing, some peers publish data while others can only read these

data and concurrent updates are not managed. In collaborative working environments, some peers publish data, others can read and write these data and a synchronization algorithm integrates concurrent updates. Collaborative environments improve the quality of data, the experience of the collaborative Wikipedia demonstrates this.

Edutella [6] proposes a RDF-based metadata infrastructure for P2P applications. It focuses on querying RDF metadata stored in distributed RDF repositories. The objective is providing access to distributed digital educational resources. Edutella distinguishes between two kinds of peers, simple and super peer. Simple peers provide data resource along with its schema. Super peers are used for several purposes, including data mediation, integration and query routing. Edutella proposes a replication service. However, it is not mentioned how to replicate and synchronize metadata.

RDFSync [11] synchronizes a target RDF graph with a source one. RDF graphs are decomposed unequivocally into minimal subsets of triples (Minimum Self-Contained Graphs MSGs) and canonically represented by ordered lists of the identifiers (hashes) of its composing MSGs. The synchronization algorithm performs a *diff* between the source and the target of the ordered list of MSGs. RDFSync can perform three kinds of synchronization, in the Target Growth Sync (TGS) the target becomes equal to the merge of both graphs, in the Target Erase Sync (TES) the target deletes unknown information by the source and finally in Target Change Sync (TCS) the target becomes equal to the source. Figure 1 shows an example of RDF graph synchronization using RDFSync.

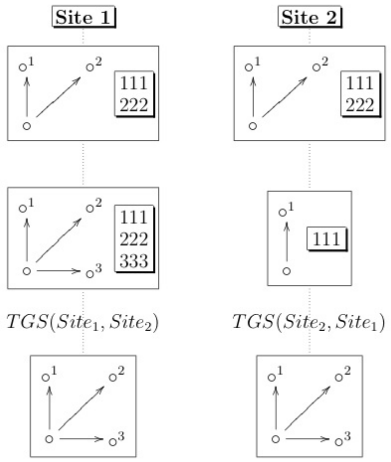

Fig. 1: RDF synchronization using RDFSync

RDFPeers [3] is a scalable distributed RDF repository. It is based on a structured P2P network. To enable faults tolerance, RDFPeers uses partial replication of RDF data. Every RDF triple is stored at three nodes of the network. However, it is not explicitly specified what happens in the case of concurrent updates on copies.

Thomas write rule [5] present techniques by which a number of loosely coupled processes can maintain duplicate copies of a database, despite the unreliability of their only means of communication. The copies of the database can be kept consistent. However, in order to remove the old deleted entries "garbage collection" they propose the following scheme: each site could notify the other sites whenever it hears about a deletion. If these notifications are transmitted in order with the "normal" sequence of modifications, then upon receipt of such a notification a site can be sure that the sending site has delivered any outstanding assignments to the deleted entry, has marked it as deleted, and will not generate any new assignments to it. This implies the knowledge of all the sites in the system. This constraint is not compatible with the P2P networks context.

In summary, P2P Semantic Web replication researches focus on knowledge sharing and querying. No algorithm have been designed for collaborative editing of RDF graphs. Consequently, they do not take into account concurrent updates on RDF graphs.

3 C-Set definition

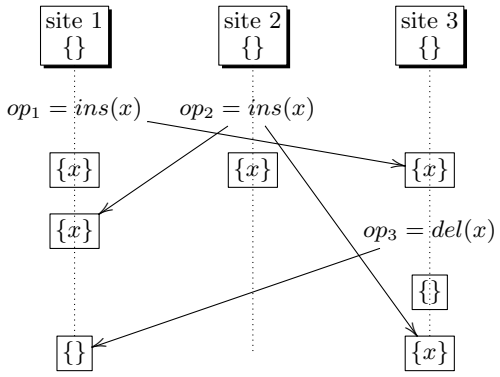

Fig. 2: A Traditional set counter-exemple

A CRDT is a data structure where all concurrent operations commute, so they do not require merge algorithms or integration procedure to enforce consistency. The replicas of a CRDT converge automatically, without complex concurrency control. Examples of algorithms that implement this data type are Woot [7],

TreeDoc [8], Logoot [13] and Logoot-Undo [14]. None of the existing algorithms handles the set data type.

A set with operations $insert(element)$ and $delete(element)$ is not a CRDT. The counter-example is presented in figure 2. The example illustrates how three sites can receive operations op_1, op_2, op_3 in different order. Site 1 executes the sequence $[op_1; op_2; op_3]$ while Site 3 executes the sequence $[op_1; op_3; op_2]$. If the execution of operations does not commute, then eventual consistency is violated i.e. the set on site 1 is empty while the set on site 3 contains $\{x\}$.

3.1 C-Set Data Structure

We want to define a CRDT for sets data type, where each element in this set corresponds to an RDF triple. We represent the set S of elements as a set of a couple of $(e : element, x : \mathbb{Z})$. We define on this set four operations : $ins(e : element)$, $del(e : element)$, $rins((e : element, k : \mathbb{Z}))$ and $rdel((e : element, k : \mathbb{Z}))$ detailed in the following section.

3.2 C-Set Algorithms

The operation $ins(e : element)$ can be executed locally immediately. It generates and sends remote insert operation $rins((e : element, k : \mathbb{Z}))$ that is executed remotely. In our model we give the *ins* operation precedence over the *del* operation. So whenever an *ins* operation is executed it compensate the effect of all the previously received *del* operations.

```
ins(e : element):
if (∃k ∈ ℤ : (e, k) ∈ S) then
 if (k ≤ 0)
    S = (S/{(e, k)}) ∪ {(e, 1)};
    send(rins((e, +|k| + 1)));
 else if (k > 0)
        S = (S/{(e, k)}) ∪ {(e, k + 1)};
        send(rins((e, +1)));
      endif
 endif
else
  S = S ∪ {(e, 1)};
  send(rins((e, +1)));
endif
```

Algorithm 1: *ins* algorithm

```
rins((e : element, k : ℤ*)):
if (∃i ∈ ℤ : (e, i) ∈ S) then
  S = (S/{(e, i)}) ∪ {(e, k + i)};
else
  S = S ∪ {(e, k)};
endif
```

Algorithm 2: *rins* algorithm

The operation $del(e : element)$ is executed locally. It generates and sends the remote delete operation $rdel((e : element, k : \mathbb{Z}^*))$ that is executed remotely.

```
del(e : element):
if (∃k ∈ ℤ : (e, k) ∈ S)
  if (k ≤ 0) then
     S = (S/{(e, k)}) ∪ {(e, k − 1)};
     send(rdel((e, −1)));
  else // k>0
    S = S/{(e, k)};
    send(rdel((e, −k)));
  endif
endif
```

Algorithm 3: *del* algorithm

```
rdel((e : element, k : ℤ*)):
if (∃i ∈ ℤ : (e, i) ∈ S)
        S = (S/{(e, i)}) ∪ {(e, i + k)};
else
        S = S ∪ {(e, +k)};
endif
```

Algorithm 4: *rdel* algorithm

Figure 3 shows an execution example of C-Sets. The first execution corresponds to the example presented in figure 2. Site 1 executes sequence $[op_1; op_2; op_3]$ which in fact executes $[(+1) + (+1) + (-1)]$ on the counter associated to x value in the C-Set. When the same operations are executed in a different order on site 3, the sequence $[op_1; op_3; op_2]$ force the computation of $[(+1) + (-1) + (+1)]$ on the counter associated to x value in the C-Set. Of course both sites converges. An alternative execution is also presented in figure 3. This execution illustrates how the operation op_3 can trigger the sending of operation $rdel(x, -2)$.

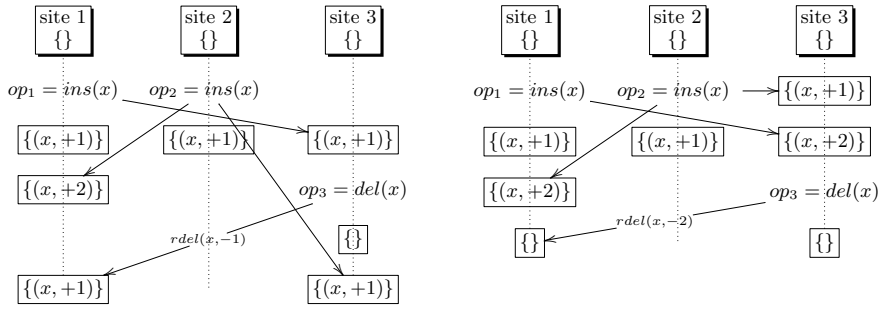

Fig. 3: C-Set : Two convergent executions

The proof that C-Set preserves eventual consistency is straightforward. On each site, C-Set generates sequence of additions of elements that belong to $(\mathbb{Z}, +)$. As addition in $(\mathbb{Z}, +)$ is commutative, C-Set converge.

4 Discussions

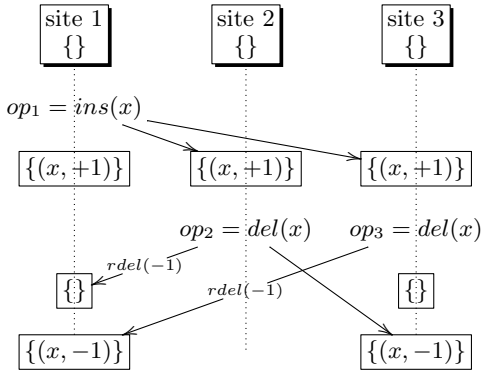

Fig. 4: C-Set and Tombstones

C-Sets is a CRDT data type for sets that ensures eventual consistency. However, as shown in figure 4, C-Set, as the Thomas write rule, relies on tombstones. This means that a deleted element remains in the store. In the example in figure 4 the final value $\{(x,-1\}$ is clearly a tombstone. Garbage collecting tombstones is a challenging issue in a dynamic P2P network as in our context. It requires each site to have knowledge about the state of all the others sites. Consequently it makes the synchronisation dependant of the number of sites and requires procedures to join and leave synchronisation groups.

An interesting property of C-Sets is that tombstones can be removed locally on one site without communication with others sites when counters associated to set elements are equal to 0. Tombstones will remain if same elements are concurrently deleted on several sites which is not the more frequent scenario. Extensive experimentation is needed to prove the efficiency of this hypothesis.

5 Conclusion and perspectives

In this paper we presented C-Set : a CRDT for sets that ensure eventual consistency. C-Set is designed to be integrated within a semantic store in order to provide P2P synchronisation of autonomous semantic store. C-Sets is more efficient than Thomas write rule, especially in managing the delete operations.

Future work will integrate C-Set within an existing semantic store. Next, we will be able to to evaluate the overhead of C-Sets using real-world semantic web data such as DBpedia [2].

References

1. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 4(2):1–22, January 2009.
2. Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, September 2009.
3. Min Cai and Martin Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In *Proceedings of the 13th international conference on World Wide Web*, pages 650–657. ACM, 2004.
4. P.A. Chirita, Stratos Idreos, Manolis Koubarakis, and Wolfgang Nejdl. Publish/-subscribe for rdf-based p2p networks. *The Semantic Web: Research and Applications*, pages 182–197, 2004.
5. P. Johnson and R. Thomas. RFC677: The maintenance of duplicate databases. 1976.
6. Wolfgang Nejdl, Boris Wolf, Changtao Qu, and Stefan Decker. EDUTELLA: a P2P networking infrastructure based on RDF. *Proceedings of the*, 2002.
7. Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data Consistency for P2P Collaborative Editing. In *Conference on Computer-Supported Cooperative Work*, 2006.
8. Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A Commutative Replicated Data Type for Cooperative Editing. *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 395–403, June 2009.
9. Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, March 2005.
10. Berners-Lee Tim and Connolly Dan. Delta: an ontology for the distribution of differences between RDF graphs. *http://www.w3.org/DesignIssues/Diff*, 2004.
11. Giovanni Tummarello, Christian Morbidoni, R. Bachmann-Gmur, and Orri Erling. RDFSync: efficient remote synchronization of RDF models. *Proceedings of the ISWC/ASWC2007*, pages 537–551, 2007.
12. Giovanni Tummarello, Christian Morbidoni, Joakim Petersson, Paolo Puliti, and F. Piazza. RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications. *The First International Workshop on Peer-to-Peer Knowledge Management*, 2004.
13. Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot : a scalable optimistic replication algorithm for collaborative editing on p2p networks. In *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2009.
14. Stephane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8), 2010.