IEEE 11th International Symposium on Programming and Systems, ISPS'13
http://ieeexplore.ieee.org/document/6581478/
DOI: 10.1109/ISPS.2013.6581478

p2pCoSU: a P2P Sparql/update for collaborative authoring of triple-stores

Hafed Zarzour*
Department of Computer Science
Mohamed Cherif Messaadia University
41000, Souk-Ahras, Algeria
hafed.zarzour@gmail.com

Mokhtar Sellami
LABGED, Department of Computer Science
Badji Mokhtar University
23000, Annaba, Algeria.
m.sellami@dgrsdt.dz

Abstract— **An important topic within Computer Supported Collaborative Work is collaborative editing or authoring system, which has been an interesting research area by the release of Web 2.0 products including Social Networks, Wikipedia, CMS, Google Docs, Blogs and many, many more. SPARQL/UPDATE is emerging as a reaction to this challenge. However, the current standard allows only a single authoring of triple-stores and does not provide transparent mechanism to support collaborative authoring. Furthermore, maintaining consistency between distributed triple-stores executing concurrent operations is a difficult problem. To solve this problem, this paper proposes a novel p2pCoSU solution that supports collaborative authoring for P2P semantic triple-stores and ensures causality, consistency and intention preservation criteria. We evaluate and compare the performance of the proposed p2pCoSU and related approaches by simulation; the results show that our solution is efficient and scalable.**

Keywords—Collaborative Editing; Similarity; Distributed System;Triple-Stores; P2P Networks; SPARQL UPDATE.

I. INTRODUCTION

An important topic within Computer Supported Collaborative Work (CSCW) is collaborative editing or authoring system, which has been an interesting research area by the release of Web 2.0 products including Social Networks, Wikipedia, CMS, Google Docs, Blogs and many, many more. Collaborative editing system can be defined as a system that allows different participants, geographically distributed, to work together on shared data and modify it by generating concurrent operations.

In Peer-to-Peer (P2P) collaborative editing system, shared data that participants are working on is replicated so that each participant works on local replica that exists on each peer. A participant at each peer works on the shared data by performing an operation to the local replica, which will change the data state. To allow all peers to get the latest state of the data, any operation generated at one peer has to be propagated to all other peers. A local operation is an operation generated by the local peer, whereas a remote operation is an operation generated by another peer and received as a result of the operation broadcast. Local or remote operations are executed in such a way that all copies are identical and the participant's intentions are preserved. At each peer, participant has to execute the local or remote operations in such a way that the data replicas are consistent and the achievement of the participant's intentions are guaranteed. Optimistic replication is largely used as a solution to provide data availability for these systems [1]. Such system is said to be correct if it ensures CCI model [2] that means Causality, Consistency, and Intention preservation.

Semantic Web technologies allow describing and exchanging knowledge via the web with the use of standard specifications such as Resource Description Framework (RDF) and SPARQL/UDDATE. Both standards provide excellent way to create and deploy an infrastructure for the Web 2.0. Where RDF represents resources with triples (subject, predicate, object), SPARQL/UPDATE [3] represents the current World Wide Web Consortium (W3C) proposed recommendation for an RDF update language. SPARQL/UPDATE reuses a syntax of the SPARQL query language for RDF and defines updating operations of RDF data to update triples from of the target graph. Updating operations are provided as inserting new triples into an RDF graph and deleting known triples from a graph. However, the current standard of SPARQL/UPDATE does not take into account neither the collaborative editing aspect, nor the P2P networks aspect.

A first solution to support a collaborative editing for existing SPARQL/UPDATE is to propose a mechanism that allows all concurrent operations to commute. The main challenge for this solution is to show practically that the insert and delete operations of the same triple can commute.

Commutative Replicated Data Type (CRDT) [4] is a convergence philosophy invented as a new generation of technique that ensures consistency maintenance of replica in a collaborative editing system without any difficulty over P2P networks. This method supposes that all concurrent operations commute.

In this paper, we propose a new model for building P2P SPARQL/UPDATE that allows collaborative editing and ensures causality, consistency and intention preservation criteria for semantic data type defined as new CRDT. The model is particularly appropriate for scalable collaborative editing of distributed triple stores. With our approach, existing SPARQL/UPDATE becomes collaborative, flexible, more efficient, and can be deployed on large-scale computer networks, especially for P2P networks.

The remainder of this paper is organized as follows. Section 2 reviews the backgrounds and the most recent related works for P2P semantic collaborative editing systems. Section 3 details the proposed solution p2pCoSU. Section 4 contains the experimental evaluation, while Section 5 concludes this work.

II. BACKGROUNDS AND RELATED WORK

A domain of replication in semantic P2P system has been addressed by several researchers. This section reviews the existing work in this domain, discusses their limitations and drawbacks. MIDAS-RDF [5] is a distributed P2P RDF/S store which supports a publish/subscribe model [6] that enables remote peers to selectively subscribe to RDF content index structure. RDFGrowth [7] is based on semantic data sharing where only one peer can modify the shared knowledge while others can read them [8]. However, collaborative editing notion is different from sharing of semantic data.

Edutella [9] presents P2P platform for semantic data based on metadata. Its mechanism focuses on querying RDF metadata stored in distributed RDF stores. A replication service is proposed as complements local storage by replicating in additional peers to achieve metadata persistence / availability and workload balancing while maintaining metadata integrity and consistency. However, they do not mention how to replicate and synchronize metadata.

RDFSync [10] is an algorithm for synchronizing a semantic data. Semantic data is defined as RDF graphs where each RDF graph is decomposed unequivocally into minimal subsets of triples and canonically represented by ordered lists of the identifiers. To ensure the synchronization, the difference is performed between the source and the target of the ordered list. However, it is not explicitly specified what happens in the case of concurrent updates on copies.

In context of data consistency approaches in distributed systems, several solutions are developed based on Operation transformation (OT)[11, 12] for supporting a range of collaboration functionalities in advanced collaborative distributed systems. However, Oster et al. [13] prove that all previously proposed transformations violate the commutative functions. In addition, there are no transformation functions for semantic data are available particularly for SPARQL/UPADATE.

Recently, Common replicated data type (CRDT) [14, 15-16, 17], has been developed as a new class of methods to ensure convergence without any synchronization requirement. This approach states that all concurrent operations commute, allowing copies to execute operations in different orders with the guarantee that the copies will be identical at the end of collaborative session. CRDT provides a simple solution which can be interesting for data replication and consistency in P2P networks. Currently there is a collection of algorithms that develop CRDTs for different data structure such as: Lgoot [18], TreeDoc [14] and XML–CRDT [19]. In the literature, several CRDTs have been designed to support collaborative editing of semantic data.

SWOOKI [20] is P2P semantic wiki that couples two domains: a semantic wikis domain and P2P wikis domain where users can add a semantic annotation in wiki pages. The users of SWOOKI collaborate for writing semantic annotations by editing wiki pages. This system is structured on distributed nodes where each node corresponds to a hosting server that contains replicated pages of semantic wiki. The semantic data are stored in RDF repositories. To edit the triple data, add and remove operations are used for inserting and deleting a RDF triple respectively. The insert operation is interpreted by the fact to increment the counter element that is associated to each triple. When the occurrence of a given triple is equal to zero, it will be permanently removed from a semantic replica. However, when a delete operation is invoked, this solution can fail in ensuring the consistency condition between peers. This gives a counter example for triples-stores and implies an inconsistent situation.

C-Set [21] is a data structure defined as CRDT for sets that can be integrated within a semantic store in order to provide P2P synchronization of autonomous semantic store. The main idea of C-set is to assign a counter to each triple of set for tracking how many times a triple t has been added or removed. To this end, four operations are defined on this set. The delete operation del() can performed locally and sends remote delete operation rdel() that is executed remotely. The ins() is an insert operation executed locally. It sends remote insert operation rins() that is executed remotely. However, they do not mention how to ensure the causality and preserve the intention of operations. Although c-set has been designed to ensure consistency, it violates the operations intentions especially when it comes to mutually execute remote delete operations on the same triples that locally have already been removed several times then reinserted.

In [22] authors define many CRDTs having a set structure, Grow Only Set (G-Set), Last Writer Wins Set (LWW-element-Set) and Observed Remove Set (OR-Set). In a G-Set, there is only an insertion operation where each element can be inserted and not deleted from the set. The reconciliation Principe is based on simple set union, since union is commutative. In a LWW-element-Set, A timestamp is attached to each element. If an element is not already exists, a local operation updates its timestamp and adds it to the set and cannot be scalable. In an Observed Remove Set (OR-Set) each element is associated to a set of unique tag. A local add creates a tag for the element and a local remove deletes all the tag of the element. However, Set ignores the intention of remove operations, LWW-element-Set is not enable to scale since it uses the tombstone mechanism and OR-Set requires transparent mechanism of unique tag generation between different sites.

In summary, P2P Semantic system studies are based on data querying and sharing. No solutions have been proposed for collaborative editing semantic triples-stores integrating SPARQL/UPDATE potentialities. Consequently, they do not ensure the CCI consistency and take into account concurrent editing of distributed triple-stores on P2P networks.

III. P2PCoSU PROPOSITION

p2pCoSU is a P2P network of semantic store nodes that allows several users, using their own site, to collaboratively edit a shared triple store without the need for physical proximity, in order to produce a group-intended final triple store. Each peer maintains a replica of the shared semantic store and each user has full access to his/her local replica. When a peer updates its local copy of data, it broadcasts a corresponding operation to all other users such that all users can view the update reflected in their local replicas. The broadcasted operation realizes the intention of the user who initiates. The main idea of this approach is to define a new CRDT for semantic store of SPARQL/UPDATE where all concurrent operations commute without any merge or integration algorithms, and it does not require a central server.

A. Data structure

A semantic store is composed of the set of RDF data where each RDF data consists of triples of the form (subject, property, object). Each triple has a number of attributes, such as its visibility in the semantic store, insert and delete counters.

Users work on a semantic store by adding triples to the semantic store, deleting triples from the semantic store, and modifying the existing triples in the semantic store. As opposed to previously defined approaches, the notion of increment-counter for RDF triples is introduced in our research as explained in the following definition.

Definition 1: Given a triple t and a set of triples S, any function $f: S \rightarrow N$, where N is a set of all natural numbers. The value of $f(t)$ is said to be the counter-increment function of the triple t, it is equal to the number of times *t* occurs in S. The counter increment function of an element can be either a zero, or a positive number.

Definition 2: An insert counter-increment is the function that represents the number of insert operations performed on the triple t, denoted by *fa(t).*

Definition 3: A delete counter-increment is the function that represents the number of delete operation performed on the triple *t*, denoted by *fd(t).*

Insert and delete counter-increment can serve as a mechanism of storage of the performed operations regardless of their type remote or local. The size of the insert/delete counter-increment is not taken into account since it it has not effect on the consistency results.

Definition 4: Let S be a set. A set with double counter-increment function 2MSet is just a triple *(S; fa; fd)* where *S* is a set, *fa* and *fd* are insert and delete counter-increment functions respectively. 2MSet is the empty set with double counter-increment function if for all $x \in S; fa(x) = fd(x) = 0$.

The notion of a set with double counter-increment function being proposed in this study is a generalization and a fusion of two sets the first one corresponds to the sequence of the removal operations and the second corresponds to the sequence of insertion operation performed on a set of triples. Both counter-increment function permit to obtain the same result when the system is on idle state. In other words, all sites that replicate a same initial data and afterwards execute the same set of the data, even in a different order, will have identical results since there are variables which maintain the effect of any executed operation.

Definition 5: Let *A = (S; fa; fd)* be a set with double counter-increment function. A visibility is a boolean value that allows to all x ∈ A to be visible or not to end users. This value depends directly on the difference between *fa* and *fd*. The visibility function of a triple *t* is expressed as:

$$f(fa(t), fd(t)) = \begin{cases} True, fa(t) > fd(t) \\ false, fa(t) \le fd(t) \end{cases}$$

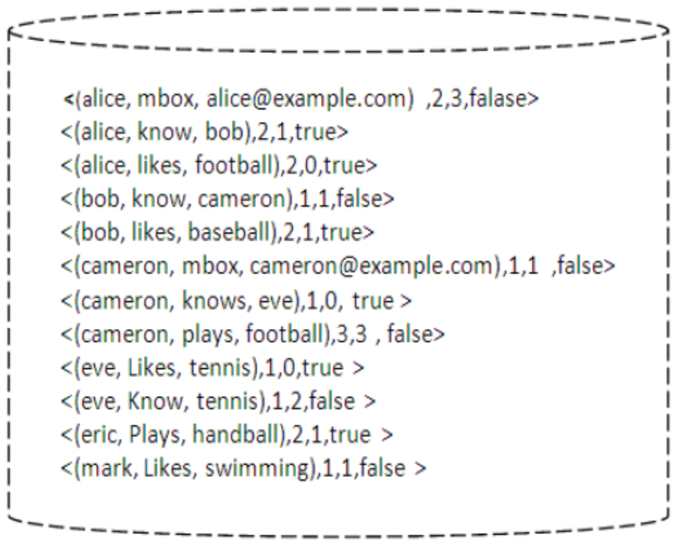
The visibility is a boolean that represents if the triple is visible or not. A triple is never really removed it is just noted as invisible.

Definition 6: An RDF store plus, denoted by R^+, is a repository used for storing RDF triples. It is a pair *(A; V)*, where A is a set with double counter-increment function and V is *V : A → Bool*.

Figure 1 shows the logical view of an RDF store plus R^+, and how to compute the visibility of each triple from an insert and delete counter-increment functions. All possible cases are presented in this sample, only the second, third, fifth, seventh, ninth, and eleventh triples appear to user because they have an insert counter-increment greater than a delete counter-increment, thus the visible RDF store plus contains (alice, know, bob), (alice, likes, football), (bob, likes, baseball), (cameron, knows, eve), (eve, likes, tennis), and (eric, plays, handball).

This mechanism of RDF store plus construction ensures convergence and consistency in any case. Therefore, deferent users have the same RDF store plus when each triple is added or removed, because the comparison between an insert and delete counter-increment functions, used for computing the visibility, is the same in any site.

Fig. 1. Logical view of RDF store plus R^+



<(alice, mbox, alice@example.com) ,2,3,falase>
<(alice, know, bob),2,1,true>
<(alice, likes, football),2,0,true>
<(bob, know, cameron),1,1,false>
<(bob, likes, baseball),2,1,true>
<(cameron, mbox, cameron@example.com),1,1 ,false>
<(cameron, knows, eve),1,0, true >
<(cameron, plays, football),3,3 , false>
<(eve, Likes, tennis),1,0,true >
<(eve, Know, tennis),1,2,false >
<(eric, Plays, handball),2,1,true >
<(mark, Likes, swimming),1,1,false >

Each operation of a user has to be broadcast to all other users such that all users can view the operation reflected in their local replicas. Each distributed operation realizes the intention of the user who initiates it, such that when the operation is executed in another RDF store plus replica, the operation is reflected correctly.

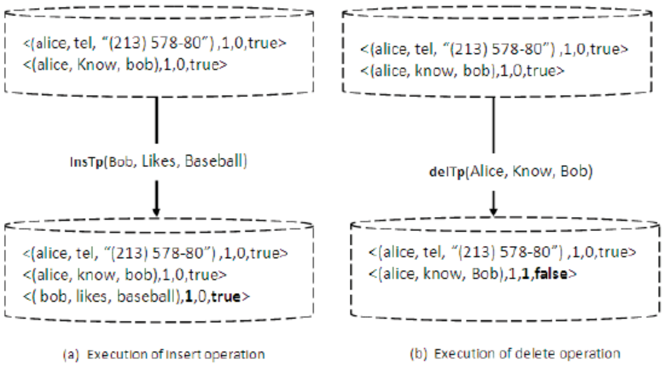
B. Updating operations

A user works on an RDF store plus by adding, deleting, and modifying triples of the RDF store plus. Every update intended by the user is realized by an operation.

In a collaborative editing system, there are two generic operation primitives that affect an RDF store: insert and delete. Meanwhile, the update operation can be considered or made equivalent as a delete of the existing value to be updated followed by an insert of the new value. The two generic operation primitives used in the RDF store plus are: insTp(t) and delTp(t). where insTp(t) is used to insert a triple t in the RDF store, delTp(t) is used to remove the triple t from the RDF store plus. An update request consists of two operations, including a triple to be deleted and a triple to be added. In other words, the execution of an update operation, changing a triple t1 to a new value of t2, consists of the sequential execution of delTp(t1) followed by insTp(t2). After each execution of local insert or delete operation of the triple t, the visibility is computed, and the corresponding remote operation is broadcast to all other sites in order to be executed. The insertion and deletion of triples group can be expressed as series execution of insTp() and delTp() opeartions. In this work, we suppose that the execution of any operation is atomic and it will be never violated.

The figure 2 illustrates a sample of RDF store plus before and after executing a delete and insert operations. The initial state of RDF store plus includes only two triples (alice, tel, "(213) 578-80") and (alice, know, bob) with visibilities equal to true.

Fig. 2. Execution example of update operations on RDF Store plus model

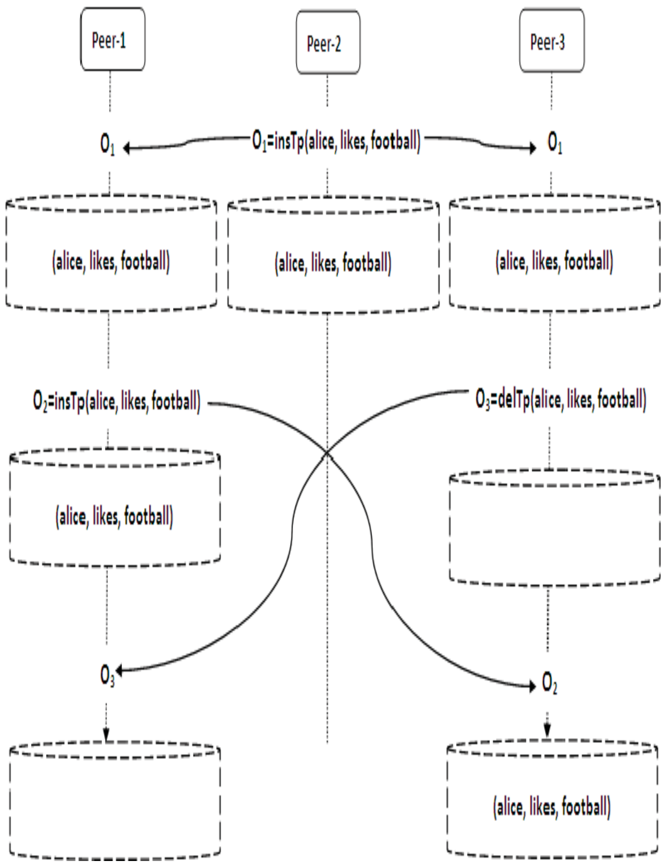


<(alice, tel, "(213) 578-80") ,1,0,true>
<(alice, Know, bob),1,0,true>
InsTp(Bob, Likes, Baseball)
<(alice, tel, "(213) 578-80") ,1,0,true>
<(alice, know, bob),1,0,true>
<(bob, likes, baseball),1,0,true>
(a) Execution of insert operation
<(alice, tel, "(213) 578-80") ,1,0,true>
<(alice, know, bob),1,0,true>
delTp(Alice, Know, Bob)
<(alice, tel, "(213) 578-80") ,1,0,true>
<(alice, know, Bob),1,1,false>
(b) Execution of delete operation

In figure 2 (a), insert operation of a new triple is performed in order to add (bob, likes, baseball) to RDF store plus. This operation occurs to a new state of RDF sore plus that contains the inserted triple. Since the triple to be added does not exist in the initial state, it is inserted and initialized to 1 and 0. The first value corresponds to the insert counter-increment whilst the second corresponds to the delete counter-increment. In Figure 2 (b), the delete operation is performed for removing the triple (alice, know, bob). Since the triple to be deleted is already in the initial state, its deleted counter-increment is incremented, and its visibility is now set to hidden, thus, the triple (alice, know, bob) is masked.

Figure 3 presents a counter-example, where an existing SPARQL/UPADTE with operations insert(triple) and delete(triple) does not commute and the eventual consistency is violated.

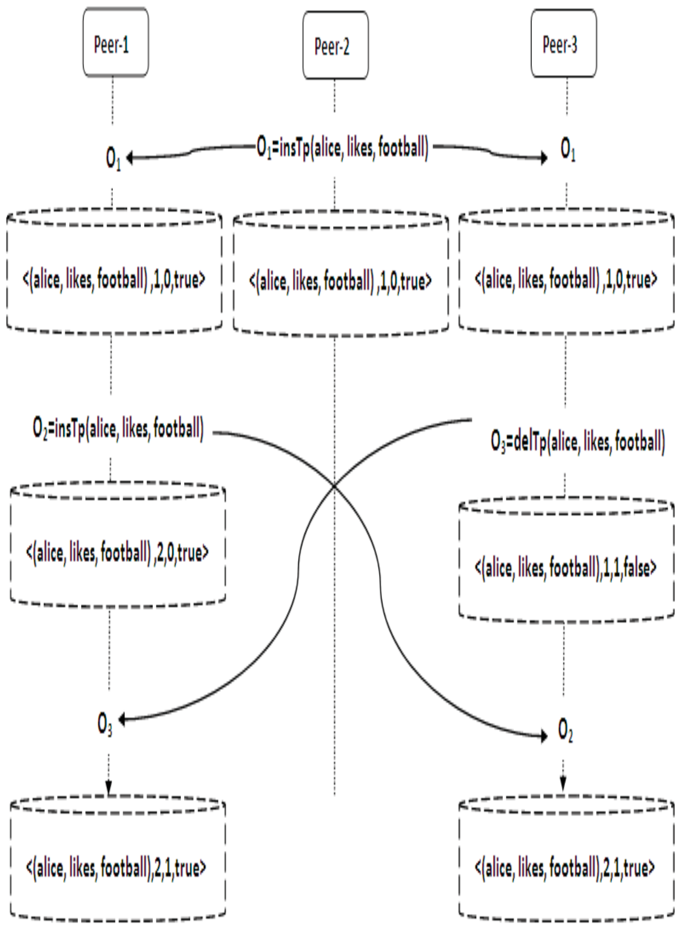
Fig. 3. An existing SPARQL/UPADTE



To explain the concepts of our solution clearly and illustrate its advantages, the same scenario is executed twice, one time for an existing SPARQL/UPADTE (see fig. 3) and the other for a p2pCoSU solution (see fig. 4). Consider the scenario outlined in figure 4. Three users at three distributed peers are to collaboratively edit an RDF store plus using p2pCoSU where each user has their own local replica. At the beginning, the local RDF stores plus are empty. The user at peer-2 inserts triple t=(alice, likes, football) (O1). This operation is broadcasted to peer-1 and peer-3. The user at peer-1 inserts another triple t (O2) and the user at peer-3 removes it (O3). When O2 arrives at peer-3, it will be executed by incrementing to 2 the insert counter-increment of t in the local

RDF store plus replica. Since the triple t is already in RDF store plus of peer-3 it will be unmasked after obtaining the new value of the visibility. When O3 arrives and executed at peer-1, the delete counter-increment will be incremented. The visibility is recomputed and its state becomes visible to the user. When O2 and O3 are broadcasted and executed mutually at peer-1 and peer-3, the end results of all replicas are identical and consistent. Thus, convergence is ensured. It should be noted that the Peer-2, shown in figure 4, converges also to the same results obtained at Peer-1 and Peer-3 contrary to the case of the Peer-2 in figure 3 which diverges.

Fig. 4. Convergence scenario with p2pCoSU approach



Peer-1
Peer-2
Peer-3
O1=insTp(alice, likes, football)
O1
O1
<(alice, likes, football) ,1,0,true>
<(alice, likes, football) ,1,0,true>
<(alice, likes, football) ,1,0,true>
O2=insTp(alice, likes, football)
O3=delTp(alice, likes, football)
<(alice, likes, football) ,2,0,true>
<(alice, likes, football),1,1,false>
O3
O2
<(alice, likes, football),2,1,true>
<(alice, likes, football),2,1,true>

C. Algorithmes

In collaborative editing systems, when a user updates the local copy, the site generates an operation that realizes the user's task. The generated operation is immediately executed at the local copy. It is then published to all other users in order to be executed. Algorithms 1 and 2, presented in figure 5 and 6 respectively, describe the procedures invoked by a site during this phase. The function insTp(t) is a local operation that allows to user to interpret the insertion intention of a new triple in two steps: first, the insertion in local level is performed, then the remote operation is sent to all peers.

Fig. 5. Insert algorithm

```
1:procedure insTp(t)          >t is the triple to be inserted
2:  add(t);
3   broadcastInsTp(t);
4: end procedure
```

In the same way, the local delete operation delTp(t) is used for removing the triple t from the local RDF store plus replica. After that, a corresponding remote operation is generated and distributed to different peers via a network.

Fig. 6. Delete algorithm

```
1:procedure DelTp(t)          >t is the triple to be deleted
2:  rmv(t);
3   broadcastDelTp(t);
4: end procedure
```

Both functions broadcastInsTp() and broadcastDelTp() guarantee to deliver successfully the local operations of insert and delete triple to all peers in order to be executed. These functions communicate respectively with integrateInsTp() and integrateDelTp() that ensure the retrieving and execution of the remote operations. The broadcast mechanism used in the delete and insert algorithms serves to guarantee the diffusion of all local executed operations to other peers of the network. If some of the broadcast packets or recipients fail, the system replays the broadcasting of the failed operation.

To add or remove a triple, two counters are used to implement a set with double counter-increment function. The first one corresponds to insert counter-increment whilst the second corresponds to the delete counter-increment. Both counters are associated to every triple, the values of these counters represent the number of delete and insert occurrences of the triple in the RDF store plus. During the insert operation (see Figure 7), the counter of the triple to be inserted is incremented and the function update(t) is invocated if it is already in RDF store plus. Elsewhere, a new triple is created < t; 1; 0; true >, where 1 and 0 are initial values of insert and delete counter-increment functions respectively. The default visibility value is true and the triple will appear to the user.

In the case of the delete operation (see Figure 8), if the triple to be deleted was already inserted in the RDF store plus, the counter that represents the delete counter-increment is incremented, then the visibility value is computed by calling to update() function, (see Figure 9). When a user executes a delete operation of a triple that does not exist or has not been inserted, the corresponding counter will be equal to one. Thus, this triple will be inserted in the RDF store plus but it is hidden.

During executing any local or remote operation, an update(t) algorithm computes the difference between the insert and the delete counter-increment of the triple t. According to the obtained value, the triple t will be masked or no. Algorithm 5 shows the procedure invoked by a peer during this phase.

Fig. 7. Add algorithm

```
1:procedure add(t)
2:  if t ∈ $R^+$ then
3 :    fa(t)++;
4 :    update(t);
5:   else
6:   fa(t)=1;
7:   fd(t)=0;
8:  ins(t, fa(t), fd(t),true);
9: end procedure
```

Fig. 8. rmv algorithm

```
1:procedure rmv(t)
2:  if t ∈ $R^+$ then
3 :    fd(t)++;
4 :    update(t);
5:   else
6:   fa(t)=0;
7:   fd(t)=1;
8:  ins(t, fa(t), fd(t),false);
9: end procedure
```

The remote operation being propagated by a peer will arrive at another peer. Whenever a remote operation is received, the peer applies the operation to its local copy in such a way so as to maintain the intention of the user who initiates the operation as well as to guarantee that the RDF store plus replicas are identical at all peers. At arrival of the remote operation at a peer, receiveInsTp(t) (see Figure 10) or receiveDelTp(t) (see Figure 11) are invoked for executing add(t) or rmv(t) according to the type of delivered operation.

Fig. 9. Update algorithm

```
1:procedure update(t)
2:  if fa(t)- fd(t) >0 then
3 :    v=true;
4 :   else
5:    v=false;
6: end procedure
```

Fig. 10. Receive insert algorithm

```
1:procedure receiveInsTp(t)
2:  add(t);
3: end procedure
```

Fig. 11. Receive delete algorithm

```
1:procedure receiveDelTp(t)
2:  rmv(t);
3: end procedure
```

D. CCI model correctness

The proof that p2pCoSu ensures CCI consistency model is straightforward. In order to guarantee causality of the generated operations, there are many causal diffusions for scalable systems such as [23]. A scalable causally algorithm [24] can be used to verify the causal consistency by tracking and checking before the execution of each operation at any sites. However, B-Set has a very interesting characteristic that ensures eventual consistency without any requirement of causal delivery or receive.

The visibility variable, which is associated to each triple, is calculated base on the difference between the insert and delete counter-increment function, as difference operation is commutative in the set of integers, according to [4] p2pCoSu ensures eventual consistency.

Since the effect of each generated operation is preserved in local or remote site by the introduction of the insert and delete counter-increment functions combined with the visibility propriety, the intention operation is respected. An expanded proof will be done in future work.

IV. EVALUATION

In this section, we present the experimentation we have made in order to compare p2pCoSu to existing solutions.

To evaluate the performance of our system, we edit the FOAF (Friend of a friend) dataset to describe social network within a virtual organization. For this reason we have implemented p2pCoSu using the ARQ API's of the open source JENA Framework [25] that implements the W3C standard SPARQL/Update language for data manipulation. We have assessed the effectiveness of our technique by examining the similarity between two different replicas performing a set of concurrent operations. We measure the similarity for C-Set [21], semantic part of SWOOKI [20], SPRAQL/UPDATE [2], and p2pCoSU. The similarity between two replicas A and B having a set structure is measured by the following function:

$$g(A,B)=\begin{cases}\dfrac{g(A\cap B)}{g(A\cup B)}\times 100, g(A\cup B)\neq 0\\ \\ 100\%, g(A\cup B)\end{cases}$$

Where *g(A ∩ B)* is a function that returns the cardinality of the union of A and B whilst *g(A∪B)* returns the cardinality of the intersection of A and B. For instance, let us consider two sets of triples *A={(alice, know, bob),(bob, know, cameron)}* and *B={(alice, know, bob)}*. The similarity between these sets is equal to 50% because *g(A ∩ B)* =1 and *g(A ∪B)*=2.

Figure 12 shows the similarity calculated between two stores of two different users executing concurrent operations following four approaches: p2pCoSU, C-Set, SWOOKI, and SPARQL/UPDATE. At the beginning, both replicas are identical and the similarity equals to 100% in all considered approaches. After executing the first set of concurrent updating, the similarity of SPARQL/UPDATE and SWOOKI decreases linearly to a minimum and begin to increase again. The p2pCoSU similarity remains constant to 100% all along the editing session, while the similarity of C-Set continuously decreases. Finally, compared to C-Set, SWOOKI, and SPARQL/UPDATE, p2pCoSU achieves better similarity in any case.

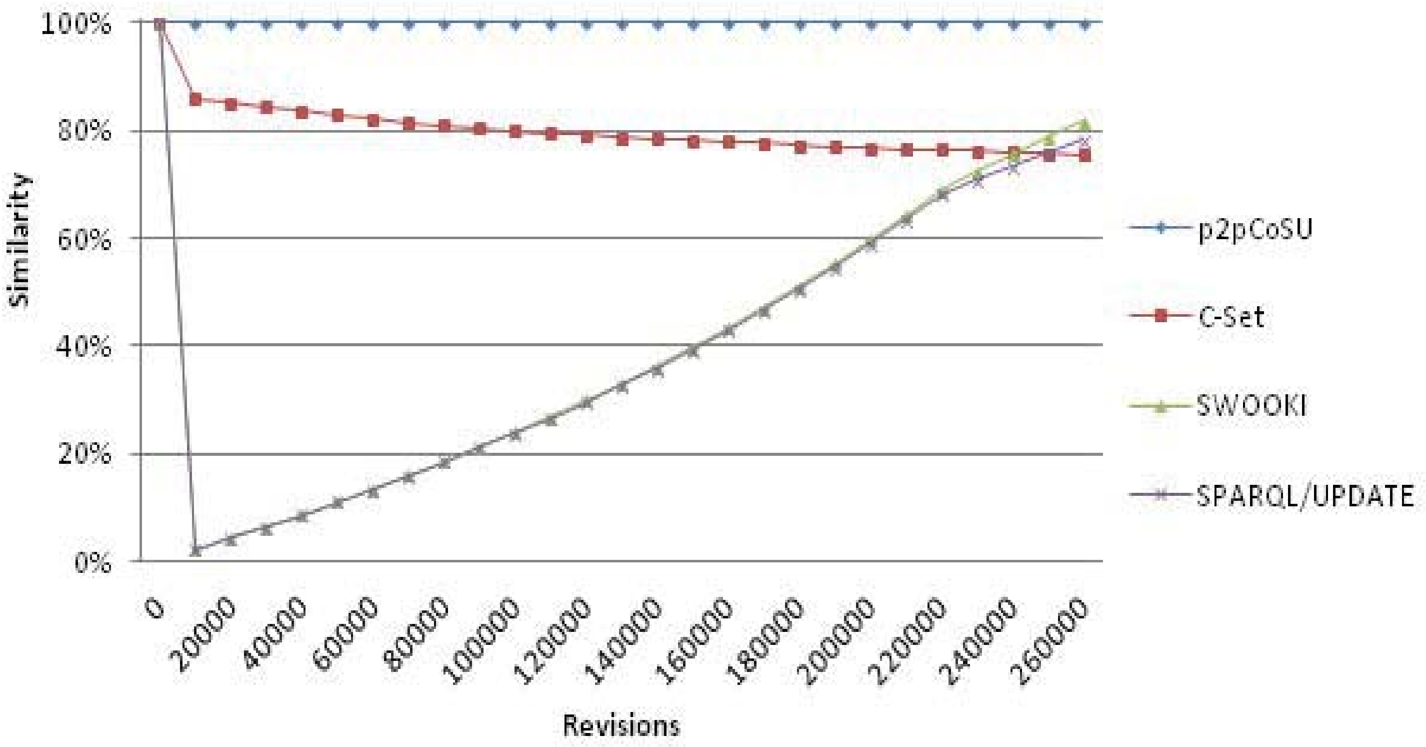


Fig. 12. Similarity between two semantic stores executing concurrent operations

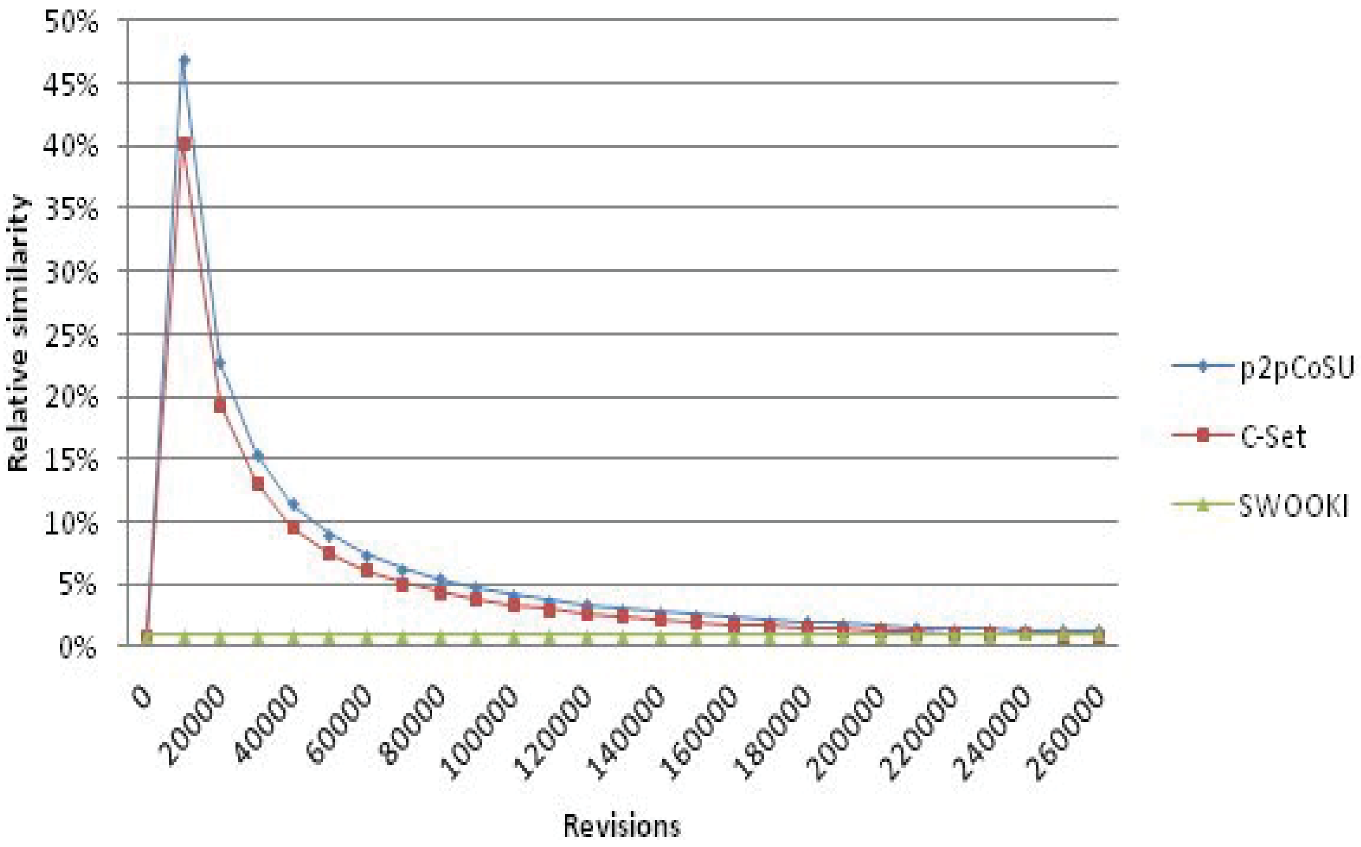
Figure 13 shows the relative similarity of the three different approaches. The relative similarity is a similarity of each approach divided by the similarity of existing SPARQL/UPDATE without any modification. We notice that p2pCoSU is most efficient in terms of improvement compared to SPARQL/UPADTE. Finally, the p2pCoSu similarity is superior to the Basic SPARQL/UPADTE similarity while a relative similarity of C-Set is inferior and SWOOKI has a poor improvement.

As consequence, p2pCoSU is well-suited for such semantic store editing since it remains the best improvement over concurrent updating.

V. Conclusion

In this paper, we have presented p2pCoSU approach for scalable collaborative editing of distributed semantic stores that uses a new commutative replicated data type. p2pCoSU is designed for large-scale decentralized networks that ensures CCI consistency model. It is a general approach that can be used with any set data type and can be a thrust for future research in this area. We have validated the p2pCoSU approach on triples collaborative editing of FOAF dataset. The experiment results demonstrate that the p2pCoSU is scalable and more efficient as it can cope well when more operations are performed. The experimentation also shows that p2pCoSU has better performances than the existing SPARQL/UPDATE, SWOOKI and C-Set approaches. In the future, we plan to integrate p2pCoSU algorithm in a distributed semantic wiki. We are currently working on semantic MediaWiki[26]. We are also working on a set undo for the p2pCoSU algorithm.

Fig. 13. Relative similarity



REFERENCES

[1] V. Martins, E. Pacitti, M. El Dick, and R. Jimenez-Peris, "Scalable and Topology-Aware Reconciliation on P2P Networks", Distrib Parallel Databases, vol. 24(1), pp.1-43, 2008.

[2] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems", ACM Transactions on Computer-Human Interaction, vol.5(1), pp.63-108, 1998.

[3] SPARQL 1.1 Update, http://www.w3.org/TR/sparql11-update/, accessed 01 January 2013.

[4] N. M. Preguic J. M. Marques, M. Shapiro, and M. Letia, "A commutative replicated data type for cooperative editing", International Conference On Distributed Computing Systems, ICDCS, IEEE Computer Society, pp.395-403, 2009.

[5] G. Tsatsanifos, D. Sacharidis, T. Sellis, On Enhancing Scalability for Distributed RDF/S Stores, in: Proceedings of International Conference on International Conference on Extending Database Technology, EDBT, pp.141-152, 2011.

[6] P.A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl, "Publish/subscribe for rdf based p2p networks", The SemanticWeb: Research and Applications, pp.182-197, 2004.

[7] G. Tummarello, C. Morbidoni, J. Petersson, P. 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.

[8] G. Tummarello, C. Morbidoni, R. Bachmann-Gmur, and O. Erling, "Rdfsync, Efficient remote synchronization of rdf models", International Semantic Web and Asian Semantic Web Conference, ISWC/ASWC, pp.537-551, 2007.

[9] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, "Edutella: a p2p networking infrastructure based on rdf", 11th international conference on World Wide Web, pp. 604-615, 2002.

[10] B. Quilitz, and U. Leser, "Querying Distributed RDF Data Sources with SPARQL", European Semantic Web Conference on The Semantic Web: Research and Applications", pp. 524-538, 2008

[11] C. Sun, and C. S. Ellis, "Operational transformation in real-time group editors: issues, algorithms, and achievements", ACM Conference on Computer Supported CooperativeWork, pp. 59-68, 1998.

[12] M. Cart, and J. Ferri, "Asynchronous reconciliation based on operational transformation for p2p collaborative environments", International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom, IEEE Computer Society, pp 127-138, 2007.

[13] G. Oster, P. Urso, P. Molli, and A. Imine, "Proving correctness of transformation functions in collaborative editing systems", LORIA – INRIA Lorraine, Research Report RR-5795, Dec. 2005. Available: http://hal.inria.fr/inria-00071213/

[14] N. Preguica, J.M. Marques, M. Shapiro, and M. Letia, "A commutative replicated data type for cooperative editing", 29th IEEE International Conference on Distributed Computing Systems, ICDCS '09, pp.395–403, Washington, DC, USA, 2009.

[15] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types", Xavier Défago, Franck Petit, and Vincent Villain, editors, Stabilization, Safety, and Security of Distributed Systems, Lecture Notes in Computer Science, vol. 6976, pp. 386–400, Springer Berlin / Heidelberg, 2011.

[16] We. Yu, “A string-wise CRDT for group editing”, 17th ACM international conference on Supporting group work, pp.141-144,New York, NY, USA, 2012

[17] Hafed Zarzour, Mokhtar Sellami, “B-Set: a synchronization method for distributed semantic stores”, International Conference on Complex Systems, ICCS'12, November 5 - 6, 2012.

[18] S. Weiss, P. Urso, and P. Molli, “Logoot-Undo: Distributed Collaborative Editing System on P2P Networks”, IEEE Transactions on Parallel and Distributed Systems, vol. 21(8), pp.1162-1174, 2010.

[19] S. Martin, P. Urso, and S. Weiss, “Scalable XML Collaborative Editing with Undo”, International Conference on Cooperative Information System, CoopIS, 2010.

[20] H. Skaf-Molli, C. Rahhal, P. Molli, “Peer-to-peer Semantic Wikis”, International Conference on Database and Expert Systems Applications, DEXA, pp.196-213, 2009.

[21] K. Aslan, H. Skaf-Molli, P. Molli, and S. Weiss, “C-set: a commutative replicated data type for semantic stores”. RED: Fourth International Workshop on Resource Discovery, At the 8th Extended Semantic Web Conference, ESWC, pp. 123-130, 2011.

[22] M. Shapiro, N. Preguica, C. Baquero, and M. Zawirski, “A comprehensive study of Convergent and Commutative Replicated Data Types”, Research Report RR-7506, INRIA, January 2011.

[23] S. Kawanami, T. Nishimura, T. Enokido, and M. Takizawa, “A Scalable Group Communication Protocol with Global Clock”, AINA, pp. 625-630, 2005.

[24] W. Lloyd, M-J. Freedman, M Kaminsky, D-G. Andersen, “Don't settle for eventual: scalable causal consistency for wide-area storage with COPS”, SOSP, pp. 401-416, 2011.

[25] Jena - A Semantic Web Framework for Java, http://jena.source.com, 2012

[26] M. Krtzsch, D. Vrandecic, M. Vlkel, H. Haller, and R. Studer, “SemanticWikipedia”, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5(4), pp.251-261, 2007.