srCE: a collaborative editing of scalable semantic stores on P2P networks

Hafed Zarzour*

Department of Computer Science,
University of Mohamed Cherif Messaadia,
Souk-Ahras, Algeria
Email: hafed.zarour@gmail.com
*Corresponding author

Mokhtar Sellami

Laboratoire sur la Gestion Electronique de Document (LabGED),
Department of Computer Science,
University of Badji Mokhtar,
Annaba, Algeria
Email: mokhtar.sellami@nasr-dz.org

Abstract: Commutative Replicated Data Type (CRDT) is a convergence philosophy invented as a new generation of technique that ensures consistency maintenance of replica in collaborative editors without any difficulty over Peer-to-Peer (P2P) networks. This technique has been successfully applied to different data representation types in scalable collaborative editing for linear, tree document structure and semi-structured data types but not yet on set data type ensuring Causality, Consistency and Intention (CCI) preservation criteria. In this paper, we propose a srCE approach, a novel CRDT for a set structure to facilitate the collaborative and concurrent editing of Resource Description Framework (RDF) stores in large scale by different members of virtual community. Our approach ensures CCI model and is not tied to a specific case and therefore can be applied for any document that complies to set structure. A prototype implementation using Friend of a Friend (FOAF) data sets with and without the srCE model illustrates significant improvement in scalability and performance.

Keywords: collaborative editing; CRDT; CCI model; RDF store; scalability; P2P networks.

Reference paper should be made as follows: Zarzour, H. and Sellami, M. (2013) 'srCE: a collaborative editing of scalable semantic stores on P2P networks', *Int. J. Computer Applications in Technology,* Vol. 48, No. 1, pp.1–13.

Biographical notes: Hafed Zarzour is currently working as an Assistant Professor at the Computer Science Department of Souk-Ahras University in Algeria. He holds an MS degree in Computer Science from Annaba University since 2008. He is now preparing for his PhD degree at the University of Annaba. His research interests include development environments, collaborative distributed systems and semantic web.

Mokhtar Sellami is currently a full Professor and Research Director in Computer Science at the University of Annaba, Algeria. He received his PhD degree in Computer Science from Grenoble University, France, in 1979. He later gained his State Doctorate degree in 1989 from the University of Annaba, Algeria, within the same field. He had worked at the European Scientific Project in Research and Information Technology (1986–1989). His research interests include distribute collaborative environment and knowledge engineering, document analysis and handwriting recognition.

1 Introduction

Collaborative editing systems offer to members of virtual community that are geographically distributed, flexible solution to concurrently edit and share data of different types. This is in order to obtain identical result via large-scale computer networks, especially for cloud computing and Peer-to-Peer (P2P) networks, which form a massively parallel computer system with distributed control, processing and information (Miriam and Easwarakumar, 2012). The major benefits include reducing errors and increasing productivity by minimising task completion time. Moreover, collaborative editing systems offer flexibility and convenience where it is easy for users to contribute from anywhere and anytime in the world with effective and efficient work processes that help in developing different viewpoints.

The semantic web community uses the Resource Description Framework (RDF) as a universal data model which stores information as a graph. An RDF is stored as a set of triples. Each triple has three components <subject, predicate, object> where the subject is the source, the predicate and object correspond to its target and label, respectively. The RDF data model offers the possibility to derive new knowledge from explicit and background knowledge (Tsatsanifos et al., 2011). The semantic P2P platform is considered as an association which couples the technology of semantic web into P2P networks and keeps the characteristics of P2P systems. A large number of users and resources lead to the need for scalable systems (Masmoudi et al., 2011).

Nowadays, developing an efficient and scalable system for collaborative editing of RDF triples on P2P networks becomes a key challenge for many semantic web environments. The main idea is to replicate RDF data at the local RDF stores for all peers collaborating together. The most significant issue in these systems is how to ensure consistency of replicas. This is a very difficult process to implement due to the fact that many issues can be encountered as conflict of editing, reconciliation and concurrency. Such a system is considered as correct and sound if it preserves the Causality, Consistency and Intention (CCI) model (Sun et al., 1998) criteria defined as follows:

1 Causality: the execution order of all operations is performed in the same way on each copy.

2 Convergence: when the system is idle, all copies are identical.

3 Intention: The expected effect of a delete and insert operation must be observed on all copies.

A P2P collaborative editing environment for distributed semantic stores should take into account the challenging difficulties discussed earlier to ensure consistency of replicated data across all collaborating users. This is after conducting various operations where commutativity, concurrency and scalability must be addressed. To overcome such challenging problem, we propose a novel scalable RDF store collaborative editing approach called srCE. The method is not only for distributed storing of RDF triples but further for supporting concurrent editing operations at large scale. The approach is a new class of Commutative Replicated Data Type (CRDT) (Preguic et al., 2009) for set structure that ensures the CCI consistency model explained earlier. In this approach, two multi-sets are typically used: the first one contains added triples, whilst the second contains deleted triples. The resulting set contains coherent data which are obtained by computing the difference between the multiplicities of both added and deleted stores.

The remainder of this paper is organised as follows: Section 2 reviews the most recent background and related works for RDF collaborative editing systems. Section 3 details the proposed solution that is srCE. Section 4 formally establishes the correctness of our approach according to CCI Model. The prototype implementation and experimental results with analysis are presented in Section 5. Finally, conclusions along with future works are drawn in Section 6.

2 Background and related work

Because of the importance of collaborative editing systems and their usage in a number of applications such as social networks, blogs, wikis and media reporting, it becomes a necessity for researchers to invest time and efforts developing a scalable system that helps people to work together concurrently and consistently.

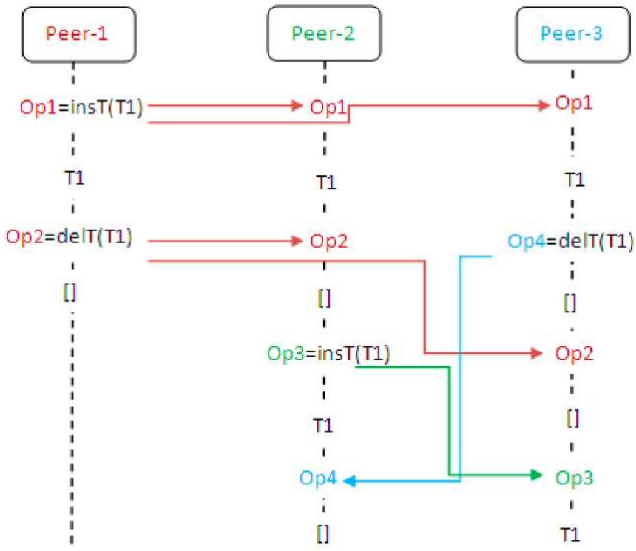
2.1 Problem description

To illustrate the challenging nature of this problem, we will identify a conflict in a collaborative editing system of an RDF set structure and show the counter example of existing solutions proposed in this context.

Let us consider three users on three different peers who are working concurrently for editing an RDF store. The RDF store is described as a set of RDF triples where each triple contains three components `<subject, predicate, object>`.

The users are working in their peers being called *peer-1, peer-2* and *peer-3,* respectively, as illustrated in Figure 1, with their sequential activities. Initially, each peer owns a copy of the same shared RDF store. The first user performs the operations as *Op1 = insT(T1)* for inserting the triple T1 and *Op2 = delT(T1)* in order to delete T1, these operations are broadcast to *peer-2* and *peer-3*. In *peer-2, Op1* and *Op2* are executed sequentially. After that the second user updates his local copy of the RDF store by *Op3 = insT(T1)* which inserts a new triple T1. This operation is published immediately. In *peer-3,* when *Op1* is executed, the third user performs *Op4 = delT(T1)* before *Op2*. During this step, *Op4* is sent to *peer-2*. Then, *Op3* and *Op4* are executed, respectively, on *peer-2* and *peer-3*.

Figure 1 Scenario of RDF set editing conflict (see online version for colours)



At the end of the execution of concurrent modifications, the copies of the RDF store diverge. That is because *insert* and *delete* for the same triple in the context of set structure are not commutative. In addition, the intentions of insert and delete operations are not preserved.

The conflicts of RDF collaborative editing systems occur when a pair of users concurrently modifies the same triple of an RDF repository. This is because the removal and insertion of the same triple do not commute. This can be expressed as follows:

$$ins(T) \succ\succ del(T) \neq del(T) \succ\succ ins(T).\ (\succ\succ: followed\ by)$$

If two operations affect the same element, they are potentially in conflict. To resolve this conflict, it must be decided which of the operations is to be taken into account, while the other will be ignored or preserved in the same order of execution for all peers. Indeed, this does not make sense as the result will correspond to neither of the authors' intentions.

2.2 From operation transformation to CRDT approach

For the context of collaborative editing systems, the Operation Transformation (OT) (Ellis and Gibbs, 1989) has been identified as an optimistic consistency control approach that maintains coherence of the replicas of the shared data. To repair the inconsistency, each remote operation is transformed before execution when users generate and perform an operation locally. Sun et al. (1998) proved that the mechanism of transformation functions must satisfy two conditions C1 and C2 (Ressel et al., 1996) to achieving convergence. A number of algorithms have been proposed based on OT approach. The most well-known of these algorithms include GOTO (Sun and Ellis., 1998), GOT (Sun et al., 1998), SOCT2 (Suleiman et al., 1998), SOCT4 (Vidot et al., 2000) and MOT2 (Cart and Ferri, 2007). While these algorithms have significant differences, all of them use a state vector associated with any element at any site.

Because of the use of vector clocks, such algorithms are known for their inability to scale as well as the fact that their correctness is hard for verification (Preguic et al., 2009). This is mainly because remote operations are inefficient as well as history buffers are likely to grow for larger memberships (Roha et al., 2011). To ensure consistency, MOT, GOTO and SOCT2 require that their transformation functions satisfy both conditions C1 and C2 (Ressel et al., 1996). The GOT method imposes neither of these conditions, but a relation of total order is required between operations and an undo/do/redo scheme. In SOCT4 framework, only C1 is necessary and C2 is replaced by a continuous global order of operations execution. SOCT2 is typically peer-to-peer collaborative environment that ensures the CCI model defined earlier. However, SOCT2 is designed only for text document structure. Further, there are no transformation functions for semantic data are available particularly for set structure.

Recently, CRDT (Preguic et al., 2009) framework is proposed as a new approach that is scalable and ensures consistency of replicas without synchronising. The approach offers control for complex concurrency by defining specific types appropriated to each data type that are commutative for any performed set of operations in order to guarantee identical results. CRDT algorithms initially designed for P2P asynchronous collaboration are suitable for real-time collaboration (Ahmed-Nacer et al., 2011).

Weiss et al. (2010) suggest Logoot a CRDT for linear structure, which provides a unique position identifier for each line in order to allow operations to commute. When an insert operation is performed, a new position is generated between the position of the previous line and next line. To achieve convergence on this data type, a total order is used between lines in the document. Unfortunately, Logoot data model uses tombstones that make the document grow without limits.

TreeDoc (Preguic et al., 2009) is another sequence CRDT designed for cooperative text editing. It uses a binary tree to represent and store the document. In this method, tombstones are used to keep track of the deleted lines. Since the complexity of the proposed technique of delete tombstones, it cannot be integrated in P2P networks.

Martin et al. (2010) present a CRDT for semi-structured data type to edit XML data. This proposition treats both components of XML structure, children elements and attributes. To allow concurrent operations to commute, a timestamp identifier is used and defined as a couple of site identifier and generated operation number. In this approach, it is demonstrated how to ensure the consistency but not how to preserve the intention and the causality of CCI model.

In summary, operation transformation-based algorithms are not suitable for P2P network and they can be applied only for applications that use a linear representation of the document (Ignat and Norrie, 2002). Therefore, there is no transformation defined for semantic stores. CRDT has been successfully applied to different data representation types in scalable collaborative editing for linear data type (text document) (Weiss et al., 2010), tree document structure data type (Preguic et al., 2009) and semi-structured data type (Martin et al., 2010) but not yet on semantic data type having a set structure and ensuring CCI model.

2.3 Distributed RDF systems on P2P

Many distributed RDF systems have been implemented to support collaborative storing, indexing and querying RDF documents (Tummarello et al., 2007). Most of them use distributed shared RDF repositories. The approach that supports peer-to-peer semantic wikis is known as SWOOKI (Skaf-Molli et al., 2009), which allows users to add semantic annotations in wiki pages. Users collaborate not only for writing wiki pages but also for writing semantic annotations. The semantic wiki is deployed on P2P servers where each peer hosts a copy of all semantic wiki pages as well as an RDF repository for the semantic data. Two

operations on the RDF repositories are defined for updating the RDF data: the first one for adding a statement and increments the occurrence associated with it. The second operation is used for physical deleting from the local RDF repository when the occurrence is equal to zero. However, this solution can fail in ensuring the consistency condition between peers particularly when a delete operation is used. For instance, when two peers perform the sequence of operations given in the following order:

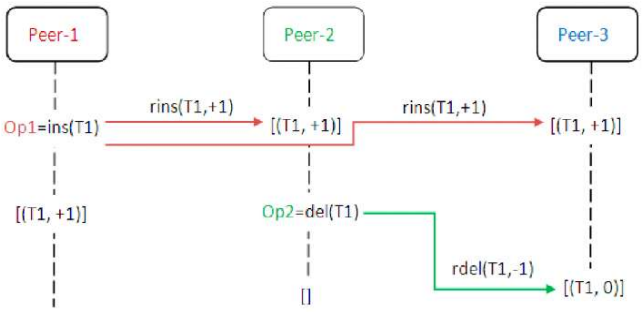
$$Peer1 : insT(T) \succ\succ delT(T) \succ\succ insT(T) \succ\succ delT(T) \Rightarrow [T]$$

$$Peer2 : insT(t) \succ\succ delT(t) delT(t) \succ\succ insT(t) \Rightarrow .[]$$

This leads to a divergence of RDF local repositories and provides inconsistency in two peers in such a way that one with statement *T* and the other without *T*.

C-Set (Aslan et al., 2011) proposes a CRDT for set data type, where four operations are defined on this set. The delete operation *del(T)* can performed locally and sends remote delete operation *rdel(T, i)* that is executed remotely. The *ins(T)* is an insert operation executed locally. It sends remote insert operation *rins(T, i)* that is executed remotely. However, they did not mention how to ensure the causality and preserve the intention of operations. In addition, their proposition can lead to conflicts of replicas and generate divergent executions. This occurs practically where a delete operation is performed after executing a remote insert operation as illustrated in Figure 2.

Figure 2 Divergence after executing modifications in C-Set proposition (see online version for colours)



RDFPeers (Cai and Frank, 2004) is one of the first distributed RDF repositories, which takes into account the peer-to-peer constraints using the MAAN (Cai et al., 2004) method. It stores and indexes each triple by specifying its subject, predicate and object. However, RDFPeers lacks the ability for supporting collaborative update operations.

Quilitz and Leser (2008) presented a querying platform of an RDF graph based on SPARQL that offers a single interface for querying multiple and distributed SPARQL endpoints. The architecture of a mediator-based system is used for providing transparent query access to multiple data sources and making query federation transparent to the client. Data sources are described by service descriptions. A service description language enables the query engine to decompose a query into sub-queries, each of which can be answered by an individual service.

SAHA 3 (Kurki and Eero, 2010) is an RDF metadata editor for collaborative annotation, which can be used to create and publish semantic content instantly on the semantic web. SAHA 3 is scalable to large data sets of objects and incorporates a simple publishing platform for building end user search portals with full-text and multifaceted search as well as online chat services. However, it supports only collaborative simultaneous RDF editing. Hence, when the user begins to edit resources, these resources are locked for other users.

Tsatsanifos et al. (2011) propose a distributed P2P RDF/S store called MIDAS-RDF, that is built on the top of a distributed multi-dimensional system for a large-distribution network (Dzafic et al., 2012), where each RDF triple is represented as a four-dimensional key. Furthermore, MIDAS-RDF supports a publish-subscribe model that enables remote peers to selectively subscribe to RDF content index structure.

Shapiro et al. (2011) present different set CRDTs, 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 principle 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 does not already exist, 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 with a set of unique tags. A local add creates a tag for the element and a local remove deletes all the tags of the element. However, G-Set ignores the intention of remove operations, LWW-element-Set is not allowed to scale since it uses the tombstone mechanism and OR-Set requires transparent mechanism of unique tag generation between different sites.

Recently, SU-Set (Ibanez et al., 2012) is proposed as a CRDT for RDF graphs based on OR-Set (Shapiro et al., 2011) that supports the SPARQL 1.1 Update operation and guarantees consistency. SU-Set is designed to serve as base for an RDF-Store CRDT that could be implemented in an RDF engine. Since OR-Set considers only insertion and deletion of single elements, it is not possible to apply OR-Set directly to SPARQL Update. Therefore, SU-Set modifies the operations to send the relevant set of triples to affect one by one, but that could flood the network with traffic considering the potential size of an RDF-graph. However, SU-Set relies on causal delivery of the underlying network, which is challenging and can pose problems in highly dynamic platforms. In addition, the intention of editing operations has not been defined.

Methods reviewed in this section are limited to sharing, querying and synchronising distributed RDF repositories. Further, due to the challenging nature of this problem, the requirements for concurrent updating of RDF triples are not fully addressed as none of these systems is known to satisfy the criteria of CCI consistency model.

3 srCE approach

In this research, we present a novel approach for collaborative editing called the srCE. The proposed method is a new class of CRDT to collaboratively edit triple stores of RDF in large scale. The concept of CRDT discussed in the work of Preguic et al. (2009) ensures scalability and consistency of replicas without synchronising. The optimistic replication strategy employed in srCE aims to ensure that peers can access RDF data without a priori synchronisation. In this framework, a new data type of RDF is defined where all concurrent operations of users from different sites are set to be commutative in a way that they can be performed at any given order. Every operation is performed locally then propagated to other peers in order to be executed. The operations should be executed concurrently in different orders and converge to the same RDF data in all peers. Our approach is not tied to a specific case and therefore can be applied for any document that complies to set structure such as database systems.

3.1 Data model

As opposed to previously defined approaches, the notion of multiplicity for RDF triples is introduced in our research as explained in this section. The multiplicity bears the number of occurrences for a given function performed by a peer.

Definition 1: *An RDF store is a repository used for storing RDF triples. It is a multi-set defined as a pair (T, f), where T is set of triples and f is a multiplicity function. For any t ∈ T then f(t) is the multiplicity of t where f:T→N, N = 1, 2, 3, ….*

The concept of multi-set being proposed in this study is a generalisation of the set usage. While a set must contain only one occurrence of a triple, a multi-set may contain multiple occurrences of the same triple. For instance, the multi-set *M1* is written as:

M1 = {<"bob","knows","alice">,<"bob","knows","alice" >, <"bob","knows","eve">}

It can be redefined using the multi-set concept as:

M1 = {(< "bob","knows","alice">, 2),

(<"bob","knows","eve">, 1)}

Where in this case, the first RDF triple has a multiplicity of two meanwhile the multi-set *M2* is defined as follows:

M2 = {<"bob","knows","alice">,

<"bob","mbox","bob@example.com">}

is redefined as:

M2 = {(<"bob","knows","alice">,1),

(<"bob","mbox","bob@example.com">, 1)}.

Definition 2: *Added RDF store, denoted by A, is an RDF store which contains all triples added by the user along with the multiplicity values.*

Definition 3: *Deleted RDF store, denoted by D, is an RDF store which contains all triples combined with the multiplicity f(t) removed by the user.*

Added and deleted RDF store can serve as an increment-only counter. The increment-only counter is useful for our context; it is used to count the number of insertion or removal of each triple for intention preservation in collaborative editing systems.

Definition 4: *Resulting RDF store, denoted by R, is the resulting set that includes all triples contained in the added RDF store A such that the multiplicity values of such triples are greater than their corresponding in the deleted RDF store D in a way that triples that their multiplicity values in D are greater than or equal to those of A are disregarded. In other words, R=A–D = {t/t ∈ A ∧ $f_A(t) > f_D(t)$}, $f_A(t)$ and $f_D(t)$ are, respectively, the multiplicity of t in A and D. if t ∉ D, $f_D(t) = 0$.*

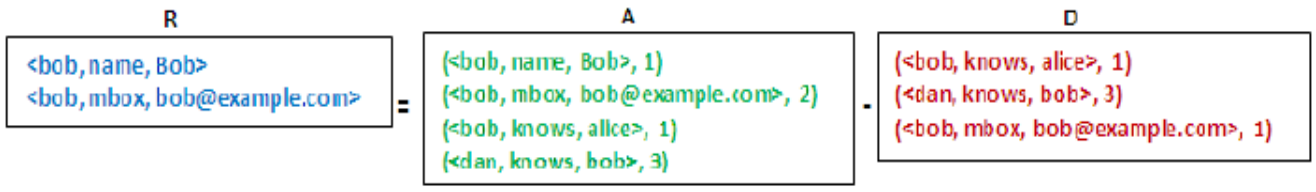
Figure 3 shows how to obtain the resulting RDF store from an added and deleted RDF stores.

All possible cases are presented in this sample, only the first and the second triples of *A* appear in *R* because they have multiplicity greater than the same triples in *D*. Thus, the resulting RDF store contains *<bob, name, Bob>* and *<bob, mbox,bob@example.com>*. The first triple does not get deleted as its multiplicity is one in the added RDF store whilst it does not exist in the deleted RDF store. Meanwhile, the second triple has a multiplicity value in the added RDF store which is greater than the one in the deleted store. This mechanism of resulting RDF store construction ensures convergence and consistency in any case. Therefore, different users should have the same RDF store when each triple is added or removed.

3.1.1 Intention model

Intentions of insert and delete operations of semantic data, particularly in the context of sets and multi-sets, have never been clearly defined. In fact, we have given below clear and accurate definitions of intentions of these operations.

Figure 3 Construction example of resulting RDF store (see online version for colours)



Definition 5: *Intention of insert operation is defined as* $f(t) = f(t) + 1$ *and* $A_{n+1} = A_n \cup \{t\}$ *where the multiplicity of triple t is incremented inside the added RDF store A. The initial value of f(t) is 0,* A_n *and* A_{n+1} *are, respectively, the added RDF stores in two different consecutive states where* A_{n+1} *includes the union of* A_n *and the added triple t.*

Definition 6: *Intention of delete operation is defined as* $f(t) = f(t) + 1$ *and* $D_{n+1} = D_n \cup \{t\}$ *where the multiplicity of triple t is incremented, the initial value of f(t) is 0,* D_n *and* D_{n+1} *are, respectively, the deleted RDF stores in two different consecutive states where* D_{n+1} *includes the union of* D_n *and the deleted triple t.*

If the triple *t* to be added or deleted does not exist in *A* or *D* so it is added directly to *A* or *D* with multiplicity one otherwise its multiplicity is incremented.

3.2 Editing operations

There are two basic editing operations that affect an RDF store in a collaborative editing system: 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 insert and delete functions are defined as follows:

1 *InsT*(*t*): is an update operation in which the triple *t* is added in the added RDF store *A*.

2 *DelT*(*t*): is an update operation in which the triple *t* is added in the deleted RDF store *D*.

To explain these concepts clearly from a point of view of data structure and concurrent editing operations, the following scenario is proposed which is shown in Figure 4: two users at two distributed peers or sites are to edit the same RDF replica where each user has his own copy.

Let us consider two triples *T1=<bob, knows, alice>* and *T2=<bob, name, Bob>*, the initial state already contains *T1* and *T2*. At the beginning, both copies are identical. The first user adds *T1* and the second removes it.

After executing *Op1* on *peer-1,* the multiplicity of *T1* in *A* is incremented to *2* and the resulting RDF store *R* contains the same triples of *R* because *D* is empty. When *Op2* is executed on *peer-2,* the *T1* is inserted to *D* with multiplicity one, the *R* contains only *T2*. When *Op1* and *Op2* are broadcasted and executed mutually, the resulting RDF store is identical.

Let us consider again the scenario presented in Figure 1. When the delete operation *Op4* is retrieved and performed on *peer-2,* the multiplicity of the triple *T1* in deleted RDF store *D* is incremented to 2. When the insert operation *Op3* is integrated on *peer-3,* the multiplicity of the corresponding triple is incremented.

The consistency between the two resulting RDF stores on *peer-2* and *peer-3* is ensured. We can observe that after executing concurrent modifications, *Peer-2* and *Peer-3* now converge and the last resulting RDF store is the same (Figure 5).

Figure 4 Convergence states of concurrent operations (see online version for colours)

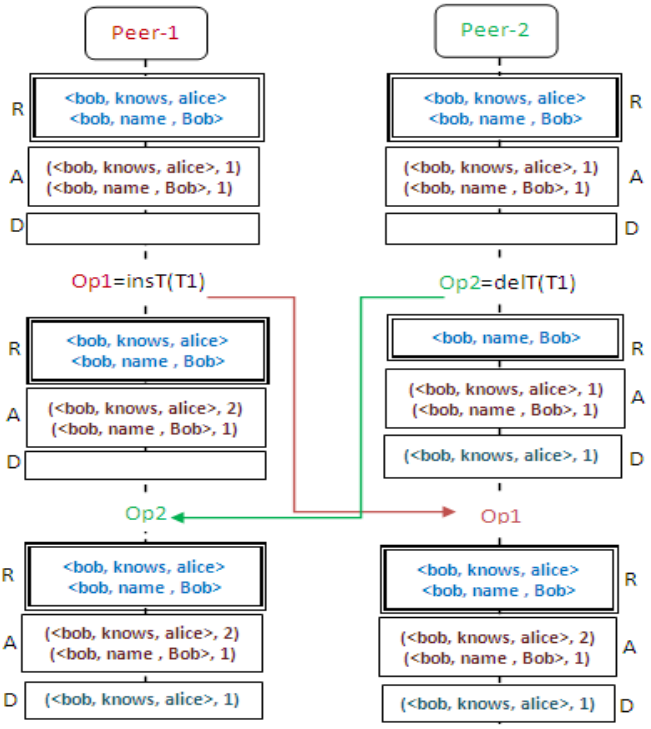
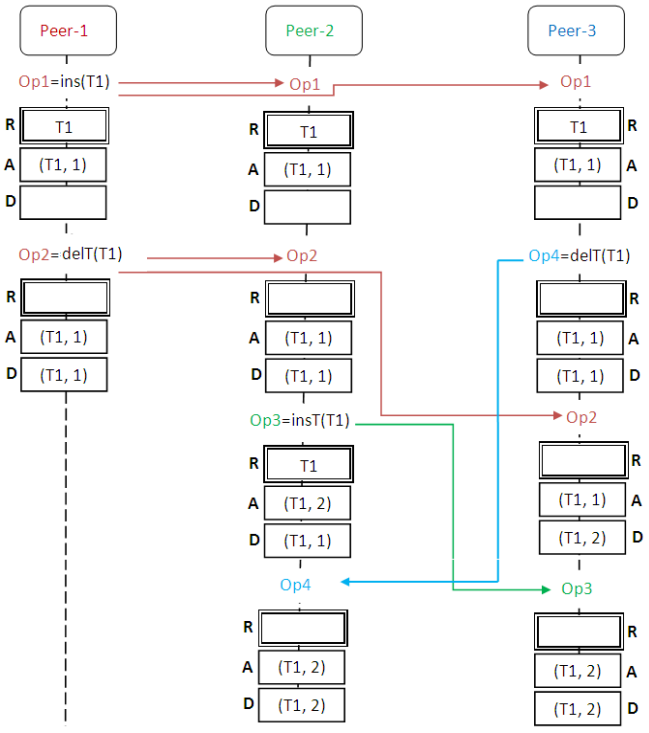


Figure 5 Convergence after using srCE method (see online version for colours)



3.3 Algorithms

Having explained the different concepts for the newly proposed srCE model, the implementation is discussed in this section along with the different procedures and functions being deployed.

The procedure *Execute(op)* outlined in Algorithm 1 which is defined mainly to perform a given operation and thereafter dispatch them to other peers. The procedure takes an operation *op* as an input argument. The *op* is defined as a data structure with two main components that are: the operation type and the triple. The function *Execute* runs the operation *op* locally and afterwards, it propagates the *op* immediately to other peers in order to be executed remotely. This ensures happens-before execution.

Algorithm 1 The algorithm for local operation execution

```
1: procedure EXECUTE(op) ▷ op is a local operation
2:     Run(op)
3:     Broadcast(op)
4: end procedure
```

The dispatch of a given operation during execution is done via the broadcast statement which ensures the delivery of an input operation to all peers. The *Broadcast* procedure, which takes an operation as an argument, guarantees that any local operation is successfully propagated and executed for all peers. When a remote operation is received by a peer, it would triggers automatically the function *Receive(op)* which performs the operation delivered via the *Broadcast* at the remote agent. The *Receive(op)* listed in Algorithm 2 takes an operation as an argument and invokes the *Run(op)* procedure.

Algorithm 2 The algorithm for retrieving remote operation

```
1: procedure RECEIVE(op) ▷ op is a remote operation
2:     Run(op)
3: end procedure
```

The *Run()* function listed in Algorithm 3 is the place where all operations sent or received are executed according to their type. As every operation has a type which is either delete or add, the *Run* procedure initially checks if there is an addition operation so that the insert function of the triple is invoked; otherwise, the delete function of the triple is executed. This is being illustrated in the following pseudo-code.

Algorithm 3 The algorithm of the execution of local or remote operation

```
1: procedure RUN(op)
2:     if op.type =Insert then
3:         InsertTriple(op.triple)
4:     else if op.type =Delete then
5:         DeleteTiple(op.triple)
6:     end if
7: end procedure
```

InsertTiple(triple) allows initially to test if the triple already exists in added RDF store *A*. If it exists, its multiplicity is incremented, otherwise it is added to *A* with multiplicity one. At the end, the resulting RDF store *R* is computed automatically after each execution of local or remote operation.

Algorithm 4 The algorithm for triple insert

```
1: procedure INSERTTRIPLE(triple) ▷ A is a local added
   RDF store
2:     if triple ∈A then
3:         triple.multiplicity ← triple.multiplicity+1;
4:         A.setTriple(triple)
5:     else
6:         insT(triple)
7:     end if
8:     ConstructResultingRDF(triple)
9: end procedure
```

The function *DeleteTriple* outlined in Algorithm 5 has the same behaviour as the previous function except that the multi-set used is the deleted RDF store *D*.

Algorithm 5 The algorithm for triple delete

```
1: procedure DELETETRIPLE(triple) ▷ D is a local deleted
   RDF store
2:     if triple ∈D then
3:         triple.multiplicity ← triple.multiplicity+1;
4:         D.setTriple(triple)
5:     else
6:         insT(triple)
7:     end if
8: ConstructResultingRDF(triple)
9: end procedure
```

The method *setTripleMultiplicity(t.multiplicity)* used by *A* or *D* allows to update the multiplicity of triple *t* by a new value *t.multiplicity*.

ConstructResultingRDF(triple) returns always a value of type set for *R* which includes consistent triples obtained after integrating concurrent modifications on each state at different sites.

Algorithm 6 The algorithm for computing result set

```
1: function CONSTRUCTRESULTINGRDF(triple)
2:     R← nil                                   ▷ R is empty
3:     for each triple of A do
4:         if A.getTripleMultiplicity(triple) >
       D.getTripleMultiplicity(triple) then
5:             insT(triple, R)
6:         end if
7:     end for
       return R
8: end function
```

The method *getTripleMultiplicity(t)* of *A* and *D* returns the multiplicity of the triple *t*. If *t* does not exist in *D*, then *getTripleMultiplicity(t)* returns 0.

4 Correctness of the approach

In this section, we show that srCE ensures the CCI consistency model. An interesting property of srCE is that

the causality is not required to ensure eventual consistency since the delete operation can be executed before the start of the execution for insert operation of the same triple. This is due to the fact that every pair of operations commutes for concurrent or non-concurrent operations. A probabilistic causal broadcast (Eugster et al., 2003) coupled with causal barriers (Prakash et al., 1997) or a scalable causal broadcast (Kshemkalyani and Singhal, 1998) can be used to ensure causality.

The following theorem states that srCE is a CRDT and ensures consistency criteria. A CRDT is a data type where all concurrent operations commute with one another (Preguic et al., 2009).

Theorem 1: *If R=A–D then srCE ensures consistency*.

We define the precedence function by $\succ\succ$, for example *Op1* $\succ\succ$ *Op2,* this means that *Op1* happens-before *Op2*.

Proof: We firstly prove that every concurrent operation pairs commutes (insert/insert, delete/delete, insert/delete).

Let $R_0 = A_0 - D_0$ be an initial state. Add a new triple to A or D is expressed mathematically as union operation, for example: $A_0 \cup t1 = A_1$.

1 Insertion operations commute

- insT(t1) $\succ\succ$ insT(t2):

  insT(t1) $\succ\succ$ insT(t2)$\Rightarrow$($A_0 \cup$ {t1}) $\cup$ t2= A_1

  So, $R_1 = A_1 - D_0$

- insT(t2) $\succ\succ$ insT(t1):

  insT(t2) $\succ\succ$ insT(t1) $\Rightarrow$($A_0 \cup t2$) $\cup$ t1 = A_2

  So, $R2 = A2 - D$o

We have $A_1 = A_2$ because union is commutative. Then $R_1 = R_2$, the resulting RDF store includes exactly the same set of triples. Therefore, insertion operations commute.

2 Delete operations commute

- delT(t1) $\succ\succ$ delT(t2):

  delT(t1) $\succ\succ$ delT(t2)$\Rightarrow$($D_0 \cup t1$) $\cup$ $t2 = D_1$

  So, $R_1 = A_0 - D_1$

- delT(t2) $\succ\succ$ delT(t1):

  delT(t2) $\succ\succ$ delT(t1) $\Rightarrow$($D_0 \cup t2$) $\cup$ t1 = D_2

  So, $R_2 = A_0 - D_2$

We have $D_1 = D_2$ because union is commutative. Then $R_1 = R_2$, after executing the two delete operations, the deleted RDF store, D, includes the same triples. Furthermore, the resulting RDF store is the same in two cases. Thus, delete operations commute

3 Delete and insertion operations commute

- insT(t1) $\succ\succ$ delT(t2):

  insT(t1)=$A_0 \cup$ t1 = A_1

  delT(t2)=$D_0 \cup t2 = D_1$

  So, $R_1 = A_1 - D_1$

- delT(t2) $\succ\succ$ insT(t1):

  delT(t2)=$D_0 \cup t2 = D_2$

  insT(t1)=$A_0 \cup t1 = A_2$

  So, $R_2 = A_2 - D_2$

We have $A_1 = A_2$ and $D_1 = D_2$, then $R_1 = R_2$.

Both resulting RDF stores obtained are identical. Thus, insertion and delete operations commute.

All concurrent operations couples commute. Thus srCE data type is CRDT. According to Preguic et al. (2009), srCE ensures consistency.

The following theorem states that insert and delete operations intentions are respected.

Theorem 2: *srCE ensures intentions*.

Proof: The main aim to introduce the multiplicity approach is to ensure intention preservation. The use of the new concepts of added and deleted RDF stores defined as multi-sets where the effect of every operation is observed in the resulting RDF store by a multiplicity counter associated with each triple added or deleted. Therefore, the intentions are preserved.

5 Experimental results

In this section, the experiment methodology carried out during the course of this research is described. Then we present the results of our extensive evaluation.

5.1 Methodology

In this subsection, we describe the srCE prototype and its basic principles as well as the experimental set-up.

5.1.1 Prototype description

A prototype of srCE is designed and implemented in Java as an extension to SPARQL/UPDATE (http://www.w3.org/TR/sparql11-update/). The current World Wide Web Consortium (W3C) proposed recommendation for an RDF update language. It reuses a syntax of the SPARQL Query Language for RDF and supports updating operations of RDF data from the target graph. Updating operations are provided as inserting new triples into an RDF graph and deleting known triples from a graph.

In particular, we use the ARQ module of the open source JENA framework (http://jena.sourceforge.net/) that

implements the W3C standard SPARQL/Update language for data manipulation. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine (Yao et al., 2011). The JENA framework is used for building semantic web applications and uses a notion of model for dealing with a set of triples that can be created from the local or remote file. In the ARQ mechanism, the RDF content created and generated by update operations can be stored in memory as Turtle (Terse RDF Triple Language) format. This RDF format is considered as a set of all triples added or deleted by the users.

The main task of srCE strategy is to maintain consistency between any two RDF stores performing any two concurrent operations, such that any triple changed in the first RDF store will be also changed in the second. To maintain this consistency, we focus on the following concepts.

In addition to the main file that contains the result of reconciliation at anytime, there are two other auxiliary files. The first corresponds to inserted triples and the second corresponds to removed triples. An integer is added to each triple to count the occurrence of the latter. In other words, the integer variable is coupled with a triple to form the multi-set. These files correspond to added and deleted RDF store, respectively.

Each local or remote operation is firstly performed in the first auxiliary files then the main file which is visible by the user is built from these files based on the function defined in Section 3.1 to ensure convergence.

5.1.2 Experimental set-up

To facilitate the evaluation of srCE, we used the FOAF (an acronym of Friend of a friend) data set for editing concurrently in order to describe social network within a virtual community. The FOAF project (http://www.foafproject.org/) is about creating a web of machine-readable homepages describing people, the links between them and their activities. FOAF is expressed using RDF data. The creation and editing of FOAF requires that the sequence of concurrent operations is performed. Each person is described by their name, email address denoted as mbox as well as the people that he/she knows. Thus, there are three elements for a person which are: name, mbox and knows.

The following listing shows an example of a FOAF profile written in Turtle format, it states that Bob is the name of person bob. His email address is bob@exemple.net and he knows Alice which is a name of person resource.

Listing 1: FOAF profile written in Turtle format

```
1 @prefix:
2          <http://people.example.com/>.
3 @prefix foaf:
4          <http:// xmlns.com/foaf/0.1/>.
5 : bob
6    foaf : knows : alice;
7    foaf : mbox <mailto: bob@example.net>;
8    foaf : name "Bob"
```

The Turtle model may be written in direct triple notation like this:

Listing 2: Triple notation

```
1 <http://people.example.com/bob>
2    <http://xmlns.com/foaf/0.1/knows>
3    <http://people.example.com/alice>
4 <http://people.example.com/bob>
5    <http://xmlns.com/foaf/0.1/mbox>
6    <mailto:bob@example.net>
7 <http://people.example.com/bob>
8     <http://xmlns.com/foaf/0.1/name>
9    “Bob”
```

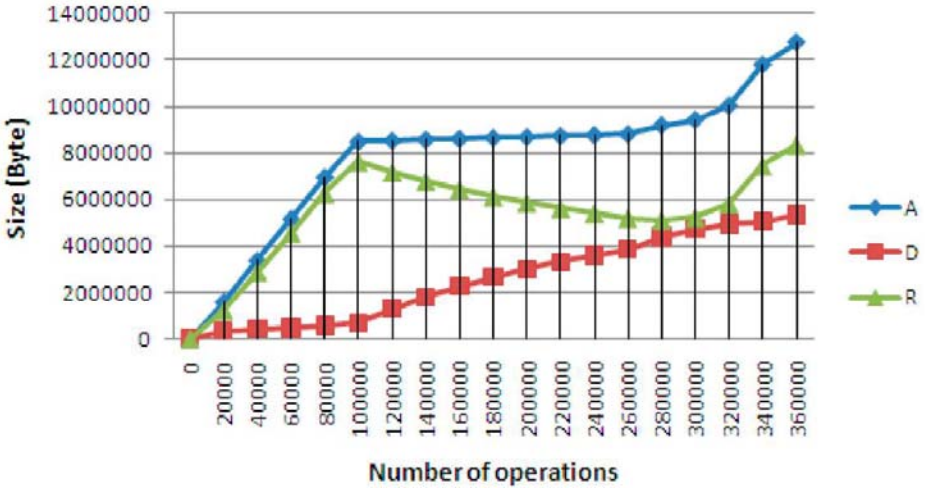
We have assessed the effectiveness of our technique by examining the size of trace files. Initially, we automatically generate a set of updating operations that include more than 360,000 operations as a series of inserts and deletes of triples. Thereafter, we perform these operations and evaluate the size with and without the srCE approach. For operation without srCE model, we used directly the ARQ default execution strategies without any modifications, i.e. the problem of inconsistency of replicas due to the inherent difficulty of concurrent modifications is not taken into account. Using the same revisions, all modifications are executed using our model and simply re-executed in SPARQL/update. Finally, we read different information of the execution from the property files.

5.2 Experimental results

To assess the scalability of the resulting RDF store *R* in relationship with both the secondary added and deleted RDF stores *A* and *D,* respectively, the system implemented using Java is executed to generate the files for the data set stores by executing a large number of updating operations reaching to 360,000 operations. The experiment is conducted during the collaborative session of FOAF triples. The sizes of added and deleted RDF store increase, respectively, with successive operations. The results of such experiment are shown in Figure 6. The x-axis represents the number of performed operations whilst the y-axis represents the size of generated *A, D* and *R* files in bytes.

It can be observed that the size of *R* grows when the size of *A* increases and that of *D* also increases, but slowly as shown in the first 100,000 operations. Since there are more delete operations of the triples, the size of *D* starts to increase while the size of *R* decreases though *A* continuously grows. When *A* and *D* approach each other, *R* reaches its minimum. Once *A* and *D* diverge, *R* increases again. The increase in size of *A* and *D* is related to the existence of triples inserted or deleted, i.e. if the triple does already exist, the size of the increase is almost marginal. So, we can say that *R* increases proportionally with the divergence of *A* and *D*.

Figure 6 Relationship between R, A and D RDF stores (see online version for colours)



To further analyse the relationship between the resulting *R*, added *A* and deleted *D* RDF stores as well as to show the effects of *A* and *D* against *R,* another experiment is carried out to measure the relative size *S* as the size of FOAF triples computed as the size of visible RDF triples divided by the size of multi-set of deleted and inserted RDF triples. This is expressed in the following equation.

$$S = \frac{|R|}{|A| + |D|} \tag{1}$$

where $|R|$, $|A|$ and $|D|$ are the sizes of triples for a resulting, added and deleted RDF stores, respectively. In a similar way, a large number of operations are applied increasingly reaching 360,000 updating operations. The results of such experiment are shown in Figure 7 where the y-axis shows the value of relative size *S*. During the life cycle of RDF store the size of the visible triples is always inferior to the sum of the sizes for the secondary files. The triples which are visible represent the consistency state. In other words, they are identical on all peers of virtual community. In the interval 20,000 and 70,000 the percentage reaches its maximum, this is due of minimising of *D* and maximising of *A*. Thereafter, the size of *R* is small compared to *A* and *D*. Thus, the larger number of operations, the lesser is the relative size.

To evaluate the effectiveness of the proposed method compared to existing SPARQL/Update, we conducted a comparative experiment where we apply an increasingly large number of updating operations to the FOAF data set using SPARQL/update as well as SPARQL/Update with the proposed srCE approach. We measure for both experiments the metric size of visible resulting RDF store as well as the size of the generated FOAF data set. The results are shown in Figure 8. During the editing session, the size of triples for the store that contains the consistency data, generated by SPARQL/update with srCE extension, remains inferior to the triples store created by traditional SPARQL/Update. This last requires more space for saving and using its triples store and gives inconsistent results.

Figure 7 Relative size of RDF store (see online version for colours)

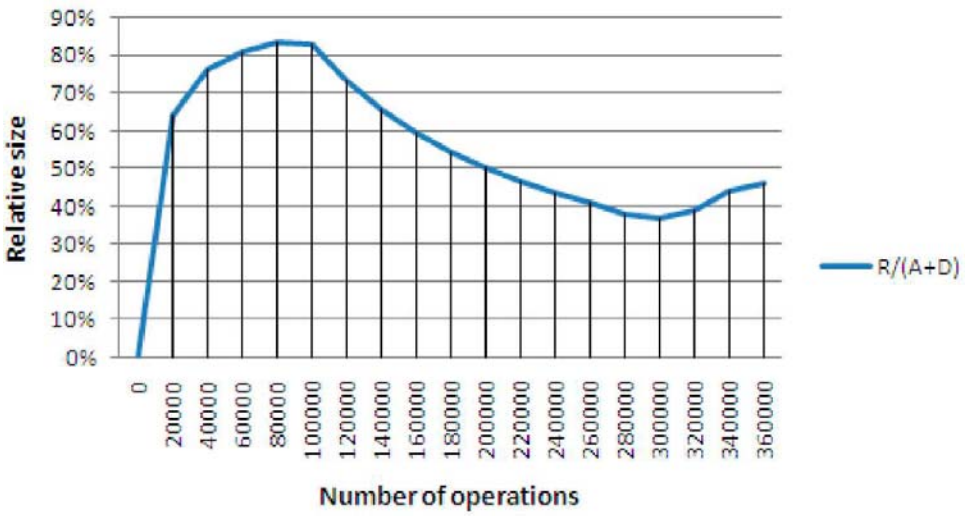


Figure 8 Visible resulting RDF store size with and without srCE model (see online version for colours)

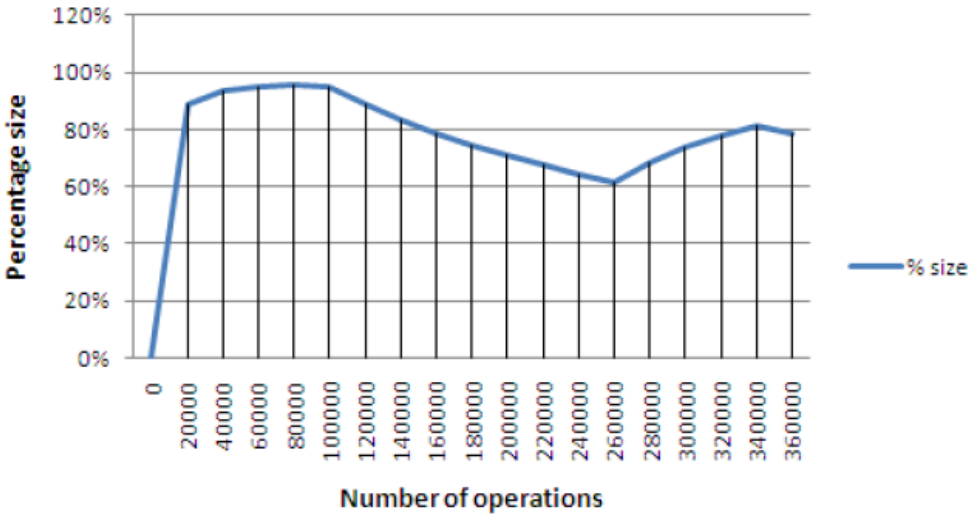
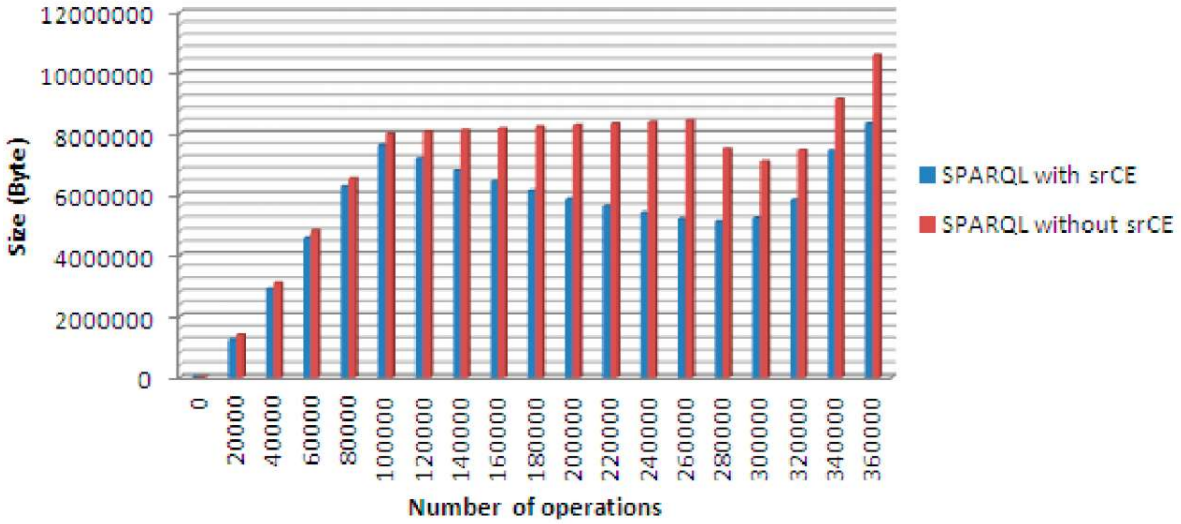


Figure 9 Percentage sizes of visible resulting triples obtained from srCE divided by resulting triples without srCE model (see online version for colours)



Furthermore, a different analysis is derived from the comparative data obtained in the previous experiment by generating the relationship between the two cases computed as the ratio of triples sizes for SPARQL with srCE over SPARQL/Update without srCE. The results are shown in Figure 9. Compared with the performed updating operations of both versions of SPARQL/Update: classical and srCE-based, the results in Figure 9 prove that the trace size decides the performance of our approach. Indeed, the srCE is more efficient when more operations are performed. Thus, it is scalable when the number of operations largely grows.

The srCE method scales in terms of the number of copies. The number of copies is not a factor in each component of srCE model. There is no total order on operations and no consensus process. Additionally to this, the srCE anatomy is independent of the number of peers or replicas. The only requirement to ensure consistency of the srCE is to perform the same set of operations in all sites, the execution order of operations within system is not important because all operations commute without any control since a delete operation can be received before or after an insert.

6 Conclusion and further work

This paper presents the srCE method which is a novel CRDT for RDF set structure that supports collaborative editing. srCE is designed for large-scale decentralised networks that ensure CCI consistency model. In a way that causality, convergence and intention of operations are all preserved and guaranteed. The major property of the proposed approach is to ensure the commutativity of the executed operations by distinguishing between the different

operations depending on their type. Introducing the concept of using added and deleted data set stores for the executed operations, whilst the final and consistent across all peers' results should appear in the resulting RDF store *R*. 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.

A number of experiments are conducted to assess the efficiency and scalability of our approach by editing triples collaboratively and concurrently from the FOAF data set. A prototype is implemented using Java programming language as an extension to SPARQL/Update that supports concurrent operations. The experimental results have demonstrated that the srCE is scalable and more efficient as it can cope well when more operations are performed. The experimentation also shows that SPARQL/Update with srCE model has better performance than without srCE. Further our approach is independent from the number of peers, replicas as well as the number of submitted operations regardless of their order. This is one of the merits of the approach compared to existing methods, which require the total order.

For future work, the srCE model can be applied in a wide range of applications including semantic wikis such as Semantic MediaWiki (Krtzsch et al., 2007) in order to allow users to take advantage of semantic technologies. Further, we plan to develop and integrate the undo component for the srCE approach. More interesting, our method can be utilised with a number of tools and systems which therefore enhances their capabilities for editing structured content in a distributed and collaborative environment. For instance, DBpedia (Bizer et al., 2009) a tool used for extracting structure content, can be extended to further build knowledge by collaboratively editing content of wikis.

Finally, the integration of the newly proposed srCE model with SPARQL/Update can be standardised as it takes into account the collaborative updating aspect in order to emerge as new generation of Web 2.0 tools that embrace semantic web technologies.

References

Ahmed-Nacer, M., Ignat, C-L., Oster, G., Roh, H-G. and Urso, P. (2011) 'Evaluating CRDTs for real-time document editing', *ACM Symposium on Document Engineering,* pp.103–112.

Aslan, K., Molli, P. and Skaf-Molli, H. (2011) 'C-Set: a commutative replicated data type for semantic stores', *Proceedings of 8th Extended Semantic Web Conference, pp.123–130.*

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R. and Hellmann, S. (2009) 'DBpedia a crystallization point for the web of data', *Journal of Web Semantics*, Vol. 7, No. 3, pp.154–165.

Cai, M. and Frank, M-R. (2004) 'RDFpeers: a scalable distributed RDF repository based on a structured peer-to-peer network', *Proceedings of International Conference on World Wide Web,* pp.650–657.

Cai, M., Frank, M-R., Chen, J. and Szekely, P-A. (2004) 'MAAN: a multi-attribute addressable network for grid information services', *Journal of Grid Computing,* Vol. 2, No. 1, pp.3–14.

Cart, M. and Ferri, J. (2007) 'Asynchronous reconciliation based on operational transformation for p2p collaborative environments', *Proceedings of International Conference on Collaborative Computing: Networking, Applications and Worksharing,* CollaborateCom, IEEE Computer Society, pp.127–138.

Dzafic, I., Neisius, H-T. and Mohapatra, P. (2012) 'High performance power flow algorithm for symmetrical distribution networks with unbalanced loading', *International Journal of Computer Applications in Technology*, Vol. 43, No. 2, pp.179–187.

Ellis, C-A. and Gibbs, S-J. (1989) 'Concurrency control in groupware systems', *Proceedings of ACM International Conference on Management of Data,* pp.399–407.

Eugster, P-T., Guerraoui, R., Handurukande, S-B., Kouznetsov, P. and Kermarrec, A-M. (2003) 'Lightweight probabilistic broadcast', *ACM Transactions on Computer Systems,* Vol. 21, No. 4, pp.341–374.

Ibanez, L.D., Skaf-Molli, H., Molli, P. and Corby, O. (2012) 'Synchronizing semantic stores with commutative replicated data types', *Proceedings of 21st International Conference Companion on World Wide Web*, pp.1091–1096.

Ignat, C-L. and Norrie, M-C. (2002) 'Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems', *Proceedings of 4th International Workshop on Collaborative Editing*, CSCW, IEEE Distributed Systems online.

Krtzsch, M., Vrandecic, D., Vlkel, M., Haller, H. and Studer, R. (2007) 'SemanticWikipedia', *Journal of Web Semantics: Science, Services and Agents on the World Wide Web,* Vol. 5, No. 4, pp.251–261.

Kshemkalyani, A-D. and Singhal, M. (1998) 'Necessary and sufficient conditions on information for causal message ordering and their optimal implementation', *Distributed Computing,* Vol. 11, No. 2, pp.91–111.

Kurki, J. and Eero, H. (2010) 'Collaborative metadata editor integrated with ontology services and faceted portals', *Proceedings of CEUR Workshop, Workshop on Ontology Repositories and Editors for the Semantic Web,* ORES, the Extended Semantic Web Conference.

Martin, S., Urso, P. and Weiss, S. (2010) 'Scalable XML collaborative editing with undo', *Proceedings of International Conference on Cooperative Information System,* CoopIS,

Masmoudi, N.K., Rekik, C., Djemel, M. and Derbel, N. (2011) 'Hierarchical control for discrete large-scale complex systems by intelligent controllers', *International Journal of Computer Applications in Technology*, Vol. 42, No. 1, pp.1–12.

Miriam, D.D.H. and Easwarakumar, K.S. (2012) 'HPGRID: a new resource management architecture with its topological properties for massively parallel systems', *International Journal of Computer Applications in Technology*, Vol. 43, No. 2, pp.155–167.

Prakash, R., Raynal, M. and Singhal, M. (1997) 'An adaptive causal ordering algorithm suited to mobile computing environments', *Journal of Parallel and Distributed Computing,* Vol. 41, No. 2, pp.190–204.

Preguic, N-M., Marques, J-M., Shapiro, M. and Letia, M. (2009) 'A commutative replicated data type for cooperative editing', *Proceedings International Conference On Distributed Computing Systems,* IEEE Computer Society, pp.395–403.

Quilitz, B. and Leser, U. (2008) 'Querying distributed RDF data sources with SPARQL', *Proceedings of European Semantic Web Conference on The Semantic Web: Research and Applications,* pp.524–538.

Ressel, M., Nitssche-Ruhland, D. and Gunzenhuser, R. (1996) 'An integrating, transformation-oriented approach to concurrency control and undo in group editors', *Proceedings of ACM International Conference on Computer Supported Cooperative Work*, CSCW, Boston, pp.288–297.

Roha, H., Jeon, M., Kim, J-S. and Lee, J. (2011) 'Replicated abstract data types: building blocks for collaborative applications', *Journal of Parallel and Distributed Computing,* Vol. 71, No. 3, pp.354–368.

Shapiro, M., Preguica, N., Baquero, C. and Zawirski, M. (2011) *A Comprehensive Study of Convergent and Commutative Replicated Data Types*, Research Report RR-7506, INRIA.

Skaf-Molli, H., Rahhal, C. and Molli, P. (2009) 'Peer-to-peer semantic wikis', *Proceedings of International Conference on Database and Expert Systems Applications,* pp.196–213.

Suleiman, M., Cart, M. and Ferri, J. (1998) 'Concurrent operations in a distributed and mobile collaborative environment', *Proceedings of International Conference on Data Engineering,* IEEE Computer Society, pp.36–45.

Sun, C. and Ellis, C-S. (1998) 'Operational transformation in real-time group editors: issues, algorithms, and achievements', *Proceedings of ACM Conference on Computer Supported Cooperative Work,* pp.59–68.

Sun, C., Jia, X., Zhang, Y., Yang, Y. and Chen, D. (1998) 'Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems', *ACM Transactions on Computer-Human Interaction,* Vol. 5, No. 1, pp.63–108.

Tsatsanifos, G., Sacharidis, D. and Sellis, T. (2011) 'On enhancing scalability for distributed RDF/S stores', *Proceedings of International Conference on International Conference on Extending Database Technology,* pp.141–152.

Tummarello, G., Morbidoni, C., Bachmann-Gmur, R. and Erling, O. (2007) 'RDFsync: efficient remote synchronization of RDF models', *Proceedings of International Semantic Web and Asian Semantic Web Conference,* pp.537–551.

Vidot, N., Cart, M., Ferri, J. and Suleiman, M. (2000) 'Copies convergence in a distributed real-time collaborative environment', *Proceedings of ACM Conference on Computer Supported Cooperative Work,* pp.171–180.

Weiss, S., Urso, P. and Molli, P. (2010) 'Logoot – undo: distributed collaborative editing system on P2P networks', *IEEE Transactions on Parallel and Distributed Systems,* Vol. 21, No. 8, pp.1162–1174.

Yao, Q., Sun, Y.Q. and Wang, H.Y. (2011) 'A novel approach to global software development for chartered enterprises', *International Journal of Computer Applications in Technology*, Vol. 40, No. 3, pp.149–159.