Making Linked Open Data Writable with Provenance Semirings

Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, Olivier Corby

Making Linked Open Data Writable with Provenance Semirings

Luis-Daniel Ibáñez[1], Hala Skaf-Molli[1], Pascal Molli[1], and Olivier Corby[2]

[1] LINA, University of Nantes
{luis.ibanez,hala.skaf,pascal.molli}@univ-nantes.fr
[2] INRIA Sophia Antipolis-Méditerranée
olivier.corby@inria.fr

Abstract. Linked Open Data cloud (LOD) is essentially read-only, restraining the possibility of collaborative knowledge construction. To support collaboration, we need to make the LOD *writable*. In this paper, we propose a vision for a writable linked data where each LOD participant can define updatable materialized views from data hosted by other participants. Consequently, building a writable LOD can be reduced to the problem of SPARQL self-maintenance of Select-Union recursive materialized views. We propose TM-Graph, an RDF-Graph annotated with elements of a specialized provenance semiring to maintain consistency of these views and we analyze complexity in space and traffic.

1 Introduction

Linked Open Data cloud (LOD) [1] is essentially read-only. As pointed out by T. Berners-Lee in [2], we need a *writable* Linked Data to allow collaborative knowledge building among LOD participants *i.e.* a third participant can update any dataset available in the LOD. A straightforward solution is to allow write access through SPARQL Update using web access control protocol [3]. Thus, a LOD participant has to manage concurrent updating and roll-back procedures in case of malicious modifications. A direct write permission could overload a participant. Existing proposals [3,4,5,6] apply the copy-modify-merge paradigm to LOD. A third participant can make a copy of a pertinent dataset and pushes back local updates to the dataset maintainer for approbation and merging. These approaches ensure eventual consistency of copies, *i.e.*, if all updates are propagated to all participants, then all copies are identical. However, they have two main drawbacks: (i) the lack of support for partial replication of the original datasets *i.e.* if a subset of a general knowledge base such as DBpedia is required in a collaboration, then the whole dataset of DBpedia has to be copied, leading to a waste of resources storing and synchronizing unnecessary data. (ii) Eventual consistency requires a connected graph of participants and eventual delivery of all updates to all participants. This is a very strong hypothesis for LOD.

[3] http://www.w3.org/wiki/WebAccessControl

Collaborative Data Sharing Systems (CDSS) [7] support partial replication and do not require a connected graph to maintain consistency. Each participant can declare materialized views on other participants datasets, cycles are supported. CDSS use provenance semirings [8] to maintain consistency of views. However, existing CDSS are mostly relational and views are not self-maintainable [9] *i.e.* CDSS require to re-execute the query of the view to ensure data consistency. This could overload the datasources, especially, those with a large number of collaborators. Moreover, CDSS scale poorly and impose a global ordering on data synchronization which is unrealistic in the LOD context.

In this paper, we propose a vision of a writable LOD by adapting the CDSS approach for LOD. The specific contributions are:

1. We define the Macro-View Self-Maintenance problem (MaS) as the consistency guarantee for collaboration networks on the LOD with partial replication, and show that it is less restrictive than the eventual convergence guaranteed by [3] and [4].
2. We propose TM-Graph as a solution for the MaS problem. TM-Graph annotates RDF-triples with elements of the Trio-Monoid, a kind of provenance semiring [8], in a similar way to CDSSs, but without the imposition of a global reconciliation order [10].
3. We give the worst case complexities in space and traffic of TM-Graph in. Both depend on the network's density and the selectivity of the Macro-Views. The space complexity is also dependent on the probability of concurrent insertions of the same data by different participants.

Section 2 describes a use case in a collaborative LOD. Section 3 describes the needed preliminary definitions. Section 4 formalizes the Macro-View Self-Maintenance (MaS) problem. Section 5 proposes TM-Graph a solution for the MaS problem. Section 6 details complexity analysis. Section 7 summarizes the state of art and related work. Finally, section 8 presents the conclusions and outlines the future work.

2 Use Case

We suppose four participants: *DBpedia* with its general knowledge base. *FranceFacts* aiming to be a knowledge base related to France, *Scientists* aiming to be a knowledge base about scientists, and *Collaborator* is running by an individual that aiming to have knowledge base in various areas (analogous to a Wikipedia user that collaborates in a broad class of articles).

The following sequence of actions illustrated in Figure 1 describes an expected collaboration scenario to do enrichment and evolution of the shared knowledge.

1. Both $FranceFacts$ and $Scientists$ start by copying the information relevant to their domain available in DBpedia, e.g. all the triples with subject or object *dbpedia:France* and all entities that are known for something. They also need to know when the data in the sources has changed to work on a fresh one. They need *Partial replication* with *Change notification*.

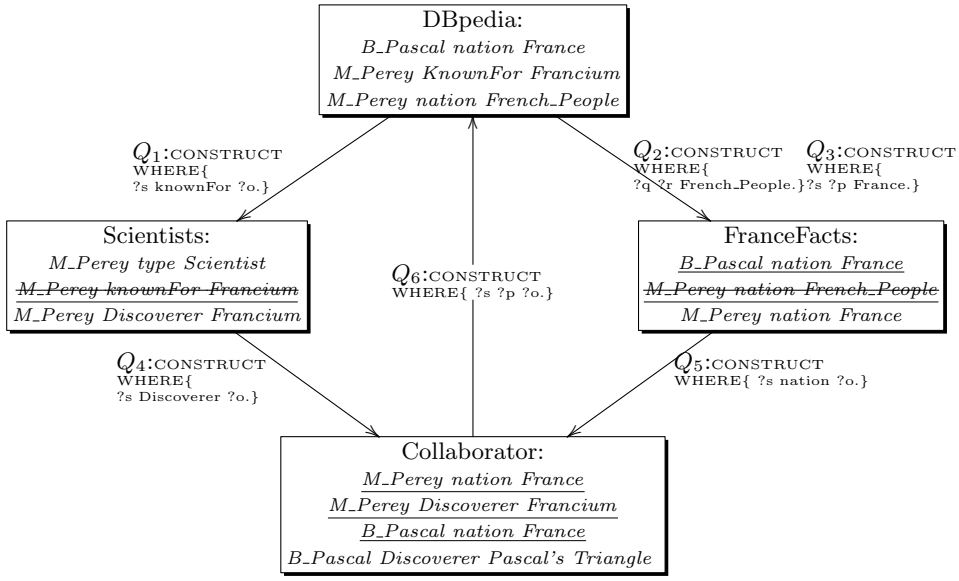


Fig. 1: Use Case of collaboration. The copying of data is formalized as a CONSTRUCT query. An underline indicates a triple replicated from other participant and a strike-through a deleted triple. For readability, prefixes are omitted.

2. *Scientists* believes that the *dbpprop:discoverer* property is more accurate than *knownFor* to describe the association between the chemical elements and the scientists that discovered them, and updates her local copy to reflect it.
3. *FranceFacts* notices that some of the triples she has copied have none or strange values of the *nationality* property, e.g., *dbpedia:French_People* instead of *dbpedia:France*[4]. She updates her copy to fix it.
4. *Collaborator* queries DBpedia to list all French discoveries and their discoverers, but many of her expected results are missing. A quick look at the DBpedia pages allows her to know that is for the reasons independently tackled by *Scientists* and *FranceFacts*. She decides to replicate data from these two participants and confirms that her query runs except for her favorite discovery, the Pascal's Triangle. She decides to add the needed triples for him to appear. When finished, she writes to DBpedia's maintainers explaining the problem and pointing the data that they need to have for correctly answer the query.
5. *DBpedia* acknowledges the error and decides to replicate the corrections made by *Collaborator*. Now, a cycle is formed between a subset of the updates of the four participants. *Scientists* had discovered the same errors at the same time and corrected them so she decides not to copy anything from

[4] To further convince the reader that the problem is real, we invite her/him to check the DBpedia 3.9 pages of Marguerite Perey, Irène Joliot-Curie and Louis De Broglie

Collaborator[5]. However, as she is subscribed to DBpedia changes, those updates will arrive to her indirectly. In this case, they need *Consistency Guarantees* in the occurrence of *Concurrent* operations

In this use-case, the copy-modify-merge approaches [3,4,5,6] will not allow view definitions and will replicate all data on all participants i.e. each participant will have a copy of DBpedia. Moreover, they will ensure eventual consistency only when the graph is get connected *i.e.* after step 4 of the scenario, all triples that belongs to the intersection of all views should be identical and there is no guarantee on remaining triples.

On the other hand, CDSS approaches [7] can support view definition but in the relational model, not on RDF data. They support semantic heterogeneity but views are not always self-maintainable i.e. maintainable only with the current materialization state and the incoming updates [11]. Finally, CDSS scales poorly and requires that sites reconcile in sequence i.e. two sites cannot reconcile in parallel.

In this paper, we want to support this use-case, with self-maintainable views and coordination synchronization mechanism. These requirements aim to respect high autonomy constraints of LOD participants.

3 Preliminaries

Let IRI be an infinite set of Internationalized Resource Identifiers, LIT a set of Literals and $Blank$ another set of IRIs[6]. All three sets are pairwise disjoint. An *RDF-triple*, or simply *triple*, is a 3-tuple $(s, p, o) \in (IRI \cup Blank) \times IRI \times (IRI \cup Blank \cup LIT)$. s, p and o are called the subject, predicate and object of the RDF-triple, respectively. An *RDF-Graph* is a set of RDF-Triples. An *RDF Graph Store* is a set $\{DG, (iri_1, G_1), \ldots, (iri_n, G_n)\}$ where DG is an RDF-Graph called *Default Graph* and the pairs (iri_n, G_n) represent the association between an IRI iri_n and an RDF-Graph G_n called *Named Graph*. Note that all IRIs are different, i.e., $i \neq j \equiv iri_i \neq iri_j$.

To ease the manipulation of Graph Stores, we use the N-Quads [12] notation. Instead of a set of sets, there is a set of *quads* (s, p, o, gn) where s, p, o corresponds to the subject, predicate and object of an RDF-Triple and gn to the graph's name IRI, with the δ symbol denoting the Default Graph.

A CONSTRUCT SPARQL 1.1 Query [13] query returns a single RDF graph specified by a graph template. The RDF graph is the union of the RDF triples resulting from substituting variables in the graph template by the corresponding value in the solutions of a given query.

In order to fully support Graph Stores, we need to extend the CONSTRUCT query definition to return a Graph Store instead of an RDF-Graph. Our solution

[5] And if *Scientists* is an Open Data source, she probably does not know who has copied her data.

[6] The skolemization of blank nodes is a feature of the RDF 1.1 working draft, see `http://www.w3.org/TR/rdf11-concepts/#section-skolemization`.

will work exactly the same way if we only consider triples within a unique RDF-Graph. For the sake of generality, we consider that CONSTRUCT queries return N-Quads for the remainder of the paper.

The Federated Query W3C Recommendation [14] defines the keyword SERVICE that allows the execution of a portion of a SPARQL query against a remote Graph Store equipped with a SPARQL endpoint.

SPARQL Update 1.1 [15] is the W3C recommendation to update RDF Graph Stores. It has two types of operations, Graph Update operations for the insertion and deletion of RDF graphs, and Graph Management operations for named graphs. Graph update operations can be *grounded*, i.e. specifying exactly the triples to add or delete, or with query pattern specified in a WHERE clause, triples are added to or removed from the Graph Store based on the bindings of the query pattern.

Without loss of generality, we consider that each Linked Data participant holds one RDF-Graph Store accessible via a SPARQL endpoint, to abbreviate, we will refer to them as *Stores* or *Participants*. To track changes done by a participant, we follow the postulates of [16], namely: The RDF-Triple is the smallest directly manageable piece of knowledge, an RDF statement can only be added or removed, the two basic types of updates are addition and removal of an RDF-Triple, and each update turns the RDF-Graph into a new state. This means that we decompose the SPARQL Update 1.1 operations issued on the RDF-Graph into individual added or removed quads. Concerning Graph Maintenance operations, we assume that the Graph Store do not register empty named graphs, i.e., each time a triple is inserted to a non-existent named graph, the graph will be created, and when a graph has no more triples left, is automatically deleted. We refer the reader to the section 3.2 of [15] for further details.

4 Macro-View Self-Maintenance (MaS) Problem

A participant needs to copy subsets of data from other participants to locally execute queries and updates. The act of copying from one participant is formalized in the following definition:

Definition 1 (Collab-View). *Let S, T two stores, let Q a SPARQL CONSTRUCT federated query defined at T such that:*

1. *Is of the form CONSTRUCT WHERE* [7].
2. *It has only one query pattern to match, i.e., in its WHERE clause.*
3. *There is only one occurrence of the SERVICE keyword, having as parameter the URL of the endpoint of S, and encompassing its query pattern,* i.e*, the triple pattern of Q is matched only against S.*

We call $C = (Q, S \to T)$ a Collab-View for T in S, where Q is a query as defined above, T is the target *and S is the* source *of the Collab-View.*

We say that a Collab-View is full *if $eval(Q, S) = S$.*

[7] `http://www.w3.org/TR/sparql11-query/#constructWhere`

Note that Collab-Views are equivalent to relational select (without projection nor join) views. Figure 1 shows the Collab-Views for the example described in section 2. *Collaborator* has two Collab-View C_1 and C_2 defined as: $C_1 = (Q_4, Scientists \rightarrow Collaborator)$ and $C_2 = (Q_5, FranceFacts \rightarrow Collaborator)$.

The directed graph $G = (N, E)$ where the nodes N is the set of the participants and the edges E is the set of directed edges between sources and targets of all Collab-View is called *Collab-Net*. The *Collab-Net* G of the example of Figure 1 is: $G = (\{DBpedia, Scientists, FranceFacts, Collaborator\}, \{(Scientists \rightarrow Collaborator), (FranceFacts \rightarrow Collaborator), (DBpedia \rightarrow Scientists), (DBpedia \rightarrow FranceFacts), (Collaborator \rightarrow DBpedia)\}$

Definition 2 (Macro-View). *Let T a store, $C = \{Q_i, S_i \rightarrow T\}$ is the set of Collab-View where T is the target and S_i is a source. The* Macro-View *of T, M_T is defined as the union of the results of the execution of all Collab-View in C, $M_T = \bigcup_{q \in Q_i} eval(q, S_i)$. The set of stores S_i is called the* sources *of M_T.*

The Macro-View of the participant *Collaborator* in the figure 1 is :

$$M_{Collaborator} = eval(Q_4, Scientists) \cup eval(Q_5, FranceFacts)$$

We expect Macro-Views to be self-maintainable i.e. maintainable with only incoming concerning updates. An update operation $op(p)$ on a quad p executed at a store S ($op@S$) concerns a Collab-View C ($op@S \rhd C$) where $C = (Q, S \rightarrow T)$ iff $p \in eval(Q, S)$.

Our scientific problem is the maintenance of Macro-Views, i.e., recursive select-union views defined as SPARQL CONSTRUCT queries, modulo the local operations.

Definition 3 (Macro-View Self-Maintenance). *Let P be a participant holding a materialization of a Macro-View M_P, $Mat(M_P)$. For each $S \in sources(M_P)$, P receives the update operations δ_i such that $\exists C = (Q, S \rightarrow P) : \delta_i \rhd C$. The Materialized Macro-View self-Maintenance problem is to compute the new value of M_P using only the previous materialization $Mat(M_P)$ and the set of $\delta_i s$* ,i.e. , *without contacting the sources.*

In short, a participant defining a Macro-View will receive all the updates that concern the Collab-Views that compose the Macro-View. We define now our consistency criteria for Collab-Net as follows.

Definition 4 (Collab-Net Consistency). *Let (P, Q) a Collab-Net. Assume each $p \in P$ maintains a sequence Δ_p with all its local updates, and another Δ_{M_p} for the updates arriving from the sources of the Macro-View it maintains, Δ_p and Δ_{M_p} are disjoints. The Collab-Net is* consistent *iff when the system is idle and all updates have been delivered, every materialized Macro-View at each p is equal to the evaluation against the sources modulo local p operations:*

$$(\forall p \in P : eval(M_p) = apply(Mat(M_p), \Delta_p^{-1}))$$

Where Δ_p^{-1} is the sequence of the inverse operations of Δ_p in reverse order.

For example, in the use-case of the previous section, the Collab-Net Consistency should ensure that the state of *Collaborator* must be equal to the union of the evaluation of m_3 and m_4 in the respective sources minus locate update, i.e., the insertion of the triples related to *Blaise Pascal*. In section 5, we propose an algorithm that ensures the consistency of a Collab-Net.

Note also that the use of provenance in the annotations has the benefit of providing extra information about the quads in the dataset. This information could be used to assess trust or to estimate the quality of the data.

5 TM-Graph: Annotating Quads with a Provenance Monoid

We propose an algorithm that using annotations with elements of a provenance semiring to solve the self-maintenance of Macro-View without the need of a global ordering for synchronization.

Provenance semirings have been already studied in the context of RDF and SPARQL queries [17,18]. Briefly, for the positive fragment of the SPARQL algebra including CONSTRUCT, the semirings studied for the relational model can be reused. As a Macro-View is equivalent to a query with SELECT and UNION, both operators belonging to the positive fragment, we need only to use the monoid $(K, +, 0)$ *i.e.* we do not need the * operator of the semiring because Macro-Views are join-less. We slightly adapt the semantics of annotated RDF-Graphs described in [17] to RDF-Graph Stores and quads: given that K is disjoint with IRI LIT and $Blank$, a K-annotated quad has the form: $(s, p, o, gn) \hookrightarrow k$ where $k \in K$. A K-Annotated RDF-Graph Store is a finite set of K-annotated quads.

Definition 5 (TM-Graph). *We specialize the Trio(X) provenance semiring [19], into the Trio-Monoid* $(\mathbb{N}(K), +, 0)$ *where:*

- *K is the set of quad identifiers defined as a pair $(ID, tick)$, where ID is a globally unique store identifier and tick is a natural number.*
- *$\mathbb{N}(K)$ is the set of polynomials with indeterminates in K and coefficients in $\mathbb{N}$.*

We call the graph annotated with Trio-Monoid elements a TM-Graph

When a store S executes an insertion, the quad is annotated with the identifier $(id(S), S.tick)$, *i.e*, the identifier of the store and the local logical time of insertion, we call this the *timestamp* of S. To union many

Collab-Views in a Macro-View, we define the union between TM-Graph as follows:

Definition 6 (TM-Graph Union). *Let G_1 and G_2 two TM-Graph, the union $G_1 \uplus G_2$ is computed as follows*

$$\begin{aligned} G_1 \uplus G_2 = &\{(s,p,o,gn) \hookrightarrow k \mid (s,p,o,gn) \hookrightarrow k \in G_1 \wedge (s,p,o,gn) \notin G_2\} \cup \\ &\{(s,p,o,gn) \hookrightarrow k \mid (s,p,o,gn) \hookrightarrow k \in G_2 \wedge (s,p,o,gn) \notin G_1\} \cup \\ &\{(s,p,o,gn) \hookrightarrow k + k' \mid (s,p,o,gn) \hookrightarrow k \in G_1 \wedge (s,p,o,gn) \hookrightarrow k' \in G_2\} \end{aligned}$$

To maintain the Macro-View, we model also the inserts and deletes operations as annotated quads, in the spirit of Z-Relations [20]; Z-Relations incorporate set difference to the positive relational algebra. This allows an unified representation of instances and Δs, and the resolution of the view maintenance problem as a query rewriting problem: the Collab-View query is rewritten to a query against the current materialization and the Δs. For TM-Graph G_1, G_2, we define the difference $G_1 \wr G_2$ by using the additive inverse of the Trio-Monoid:

$$G_1 \wr G_2 = \{(s,p,o,gn) \hookrightarrow k \,|\, (s,p,o,gn) \hookrightarrow k \in G_1 \wedge (s,p,o,gn) \hookrightarrow k \notin G_2\} \cup$$
$$\{(s,p,o,gn) \hookrightarrow max(0, k - k') \,|\, (s,p,o,gn) \hookrightarrow k \in G_1 \wedge (s,p,o,gn) \hookrightarrow k' \in G_2\}$$

When a quad annotation is 0, it is deleted from the store. This truncation enables the set semantics in relational data, which is our intended effect with TM-Graph. Another important observation for Z-Relations is that the difference can be expressed as an union thanks to the identity: $k - k' = k + (-k')$.

For every Collab-View $C = (Q, S \to T)$, we construct a sequence of annotated quads Δ_C based on the sequence of updates executed at S as follows:

- If S inserts $(s,p,o,gn) \hookrightarrow k$, then $(s,p,o,gn) \hookrightarrow k$ is appended to Δ_C.
- If S deletes $(s,p,o,gn) \hookrightarrow k$, then $(s,p,o,gn) \hookrightarrow -k$ is appended to Δ_C.

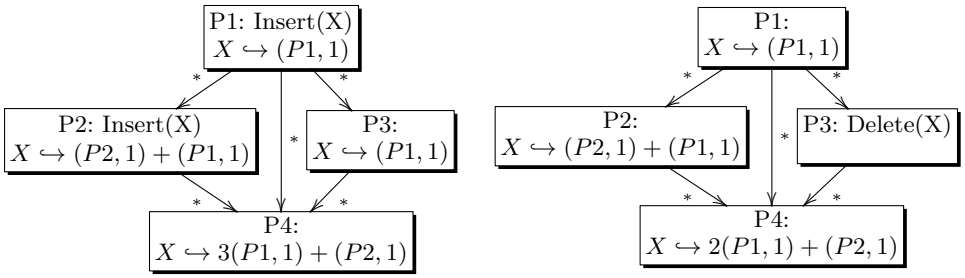
Therefore, given a Materialized Macro-View $Mat(M)$ and the set of Δs of each of its Collab-View, $\{\Delta_{C_1}, \ldots, \Delta_{C_n}\}$ the expression of $Mat(M)$ self-maintenance is simply:

$$Mat(MV)' = Mat(MV) \uplus \Delta_{C_1} \uplus \cdots \uplus \Delta_{C_n}$$

Figure 2 shows a Collab-Net with four participants and six Collab-Views. Assume that the insertion of an element X concerns all Collab-View and consider that participants $P1$ and $P2$ insert X (2a). The instance of X in $P1$ is annotated with a $P1$'s timestamp and the instance of X in $P2$ with a $P2$'s timestamp, each time the operation traverses a Collab-View, the target unions the annotated quad with its current state. Intuitively, each monomial in the annotation counts the number of times that the insertion of the quad arrived from the creator of an instance of X to this participant through a different simple path. Thereafter, consider that $P3$ deletes X (2b), $P4$ will only delete the terms of the annotation that come from $P3$.

When a Collab-View is declared, it will be executed once at the source and then materialized at the target, later, the source will push any concerning updates to the target. Upon reception of updates, the target will apply them immediately. If a target is also the source of another Collab-View, it will push local updates *and* the updates received from its sources. We make two assumptions about the update transfer: is *reliable*, i.e., that operations do not get lost; and it guarantees that if an operation a concerning the Collab-View happened before another concerning operation b, then a will be delivered to the target before b.

However, when the network has cycles, an update could be forwarded infinitely. To take cycles in account, we redefine an operation op as : $op = (q, p)$



(a) Concurrent insertion in TM-Graph

(b) Deletion in TM-Graph

Fig. 2: TM-Graph in action. The sum of indeterminates tracks the concurrent insertion of the same quad, the coefficient tracks the number of times a quad created by the same participant arrived through different paths(2a).Thereafter, when P3 deletes X (2b), P4 "substracts" the annotation coming from P3.

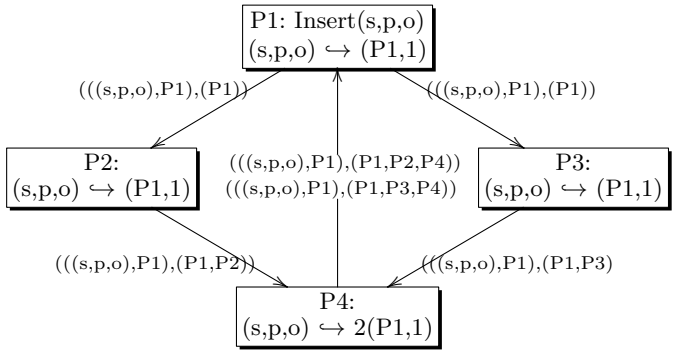


Fig. 3: Detection of cycles in a Collab-Net. Starting with the insertion of (s,p,o) by P1, operations sent carry on the path they have walked. P1 can detect that operations coming from P4 have already been executed and ignore them.

```
//Δ_M: Set of sequences of operations from sources,
//S: Current state, ID: Store's identifier
Input: Δ_M, S, ID

foreach Δ_m ∈ Δ_M:
  foreach op ∈ Δ_m:
    if ID ∉ op.p:
      S ⊎ op.q
      append(op.p, ID)
      publish(op)
```

Specification 1.1: Synchronization algorithm executed at each participant

where q is the annotated quad and p is a sequence of store identifiers representing the *path* followed by the operation. Operations are published into Δs as explained above. If the operation has not cycled, it is executed, tagged as having "passed" by this store by appending the ID to the path and published; otherwise, is ignored. Local operations are tagged with the ID of the store before being published. Figure 3 illustrates the path tagging and the cycle detection.

Specification 1.1 describes the synchronization algorithm that every participant executes when it receives a sequence of updates. After checking the operation has not cycled, applies the $\uplus$ operator, adds himself to the path of the operation and forwards it.

6 Complexity Analysis

In this section we analyze the complexity in space and traffic of TM-Graph, answering the question: *how much space and traffic costs to guarantee the Collab-Net consistency ?*. Note that, in execution time, algorithm 1.1 is linear in the number of operations received.

TM-Graph's overhead in space comes from the size of each quad's annotation. Each indeterminate of the Trio Monoid is the size of a timestamp , $ts = (ID, tick)$. A tick can be represented as an unsigned long (8 bytes). For the ids, we can consider that the top-level domain of a linked data organization uniquely identifies it. To optimize, each store can maintain a table associating to each domain a number, instead of directly storing a potentially long string.

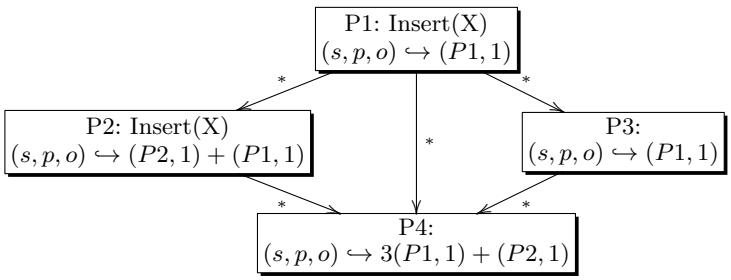


Fig. 4: Illustration of Space Overhead. The annotation of (s,p,o) in P4 has two terms because (s,p,o) was inserted concurrently in two stores (P1,P2) from which there is a path to P4 where the insertion of (s,p,o) concerns all edges. The indeterminate $(P1,1)$ has a coefficient of 3 because there are three paths from P1 to P4 such that the insertion of (s,p,o) concerns all edges in them.

The number of terms β in the annotation of a quad q stored in a participant P is equal to the number of concurrent insertions of q in different participants that arrive to P. An update operation *op* arrives to P from S iff there is a path in the Collab-Net from S to P such that for all Collab-Views C in the path

op concerns C. For example, in figure 4, the annotation of (s, p, o) in P4 has two terms because (s, p, o) was inserted concurrently in two stores (P1,P2) from which there is a path to P4 where the insertion of (s, p, o) concerns all edges.

Note also that even if there are many paths such that the operation concerns all edges, β augments only in the case of concurrent insertions, i.e., the predecessors of P insert the same data independently of each other. The worst case for β is a strongly connected Collab-Net CN with all full Collab-Views, i.e., a full replication scenario, where every participant inserts the same triples concurrently. In this case the annotation of each triple will have $|CN|$ terms.

The coefficient ρ of an indeterminate in the annotation of a quad q in a participant S is equal to the number of times that the update operation that inserts q arrives through different simple paths from the generator of the insertion of q to S. For example, in figure 4 the indeterminate $(P1, 1)$ has a coefficient of 3 because there are three paths from P1 to P4 such that the insertion of (s, p, o) concerns all edges in them: The direct one, and through P2 and P3 respectively.

The worst case for ρ is a Collab-Net that forms a complete graph with all Collab-Views full. In this case the insertion of a triple will concern all Collab-Views an it will arrive $N!$ times to each other participant.

Let $sizeOf$ be a function that returns the space needed to store an object x, ts_i and ρ_i the indeterminate and coefficient of the i-th term. The size of the annotation of a quad for a Collab-Net of N participants is given by:

$$\sum_{i=1}^{\beta}(sizeOf(ts_i) + sizeOf(\rho_i)),\ \ 1 \leq \beta \leq N,\ \ 1 \leq \rho_i \leq N!$$

Assuming a constant $sizeOf(ts)$, the worst case is $N*(sizeOf(ts)+sizeOf(N!)$. For example, in figure 4 the overhead of (s, p, o) at P4 is $sizeOf\,(P1, 1) + sizeOf(3) + sizeOf\,(P2, 1) + sizeOf(1)$

Note that we don't need to store a factorial amount of terms, but a sequence of numbers that in the worst case are factorial. This means that even if ρ is very high, the space used to store it is rather low.

In terms of the number of messages exchanged to attain the View Maintenance, our solution is optimal: after receiving all the operations from the δs, the maintenance is assured modulo local operations. On the other hand, the overhead in the size of the message varies depending of the operation. Insertions have a constant overhead of $sizeOf(ts)$, deletions' overhead equals to the size of the annotation of the deleted quad. However, recall that operations need to be tagged with the path they have "walked", to detect cycles and avoid infinite forwarding.

The length of the path of an operation *op* generated at a participant P, that we call $\phi(op)$, is bounded by the length of the longest simple path starting from P such that *op* concerns all edges. For example, for the Collab-Net depicted in figure 3, the update sent by P1 concerns all Collab-Views in the Collab-Net. As the Collab-Net in this case is strongly connected, the operation will "walk" all the cycles starting at $S1$: $(S1, S2, S3)$ and $(S1, S3, S4)$.

The worst case for ϕ is a Collab-Net where it exists a hamiltonian path starting at each participant and all Collab-Views are full. In this case, all operations will carry in the last hop of the path, a sequence with the identifiers of all the participants of the Collab-Net.

To summarize, TM-Graph's performance is affected in order of importance by: (i) Collab-Net's density, the less dense, the better. (ii) Collab-View Selectivity, the more selective, the better. (iii) Probability of many participants inserting the same data concurrently, the less, the better. In short, the farther the Collab-Net is from a Full Replication scenario, the better performance of TM-Graph.

We argue that Linked Open Data is a collection of datasets of different domains with low overlapping, meaning that the probability of having concurrent insertions of the same data is rather low. Our context is aimed to partial replication, where participants copy fragments of the original dataset to work only with their subset of interest, so we can expect selective Collab-Views.

Nevertheless, note that the worst cases for β and ϕ have only a cost linear in the size of the Collab-Net, and that storing very high values of ρ represents a low cost in space. For example, a C++ unsigned long long can hold a positive integer up to $2^{64} - 1$ in 64 bits.

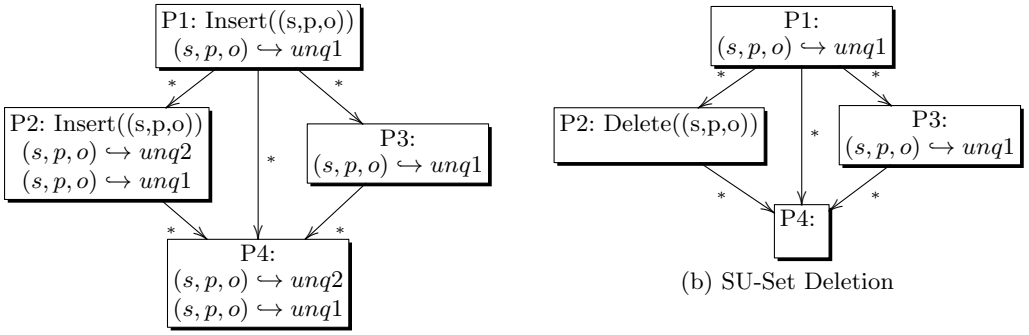
7 Related Work

The first use of CONSTRUCT queries as views in RDF-Graphs appeared in [21], however, [21] lies on avoiding replication and be able to define declaratively RDF-Graphs with data belonging to different stores. As a consequence, [21] does not support update operations. [22] also advocates that views on Semantic Web should be RDF-Graphs, an extension to SPARQL is proposed to handle real use cases from the bioinformatics domain.

Proposals in [3,4,5,6] allow collaborative editing of datasets by applying the copy-modify-merge paradigm. A third participant take a copy of pertinent datasets and push back updates to datasets maintainers for approbation and merging. Compared to our proposal, such approaches do not support partial replication and view-maintenance. Nevertheless, if the Collab-Net is connected and all Macro-Views are full, then both systems are comparable i.e. all systems will ensure eventual consistency. However, MaS defines a stronger level of consistency than eventual consistency. We illustrate this using SU-Set [3].

SU-Set [3] is a Conflict-Free Replicated Data Type, a type whose operations; when concurrent; do not conflict and guarantees eventual consistency. Another CRDT for RDF-Graph is srCE [4]. SU-Set tags inserted RDF-Triples with an unique *id*, together with a connected network that guarantees eventual delivery of updates, guarantees that when all updates have been delivered, all participants will have identical data.

The eventual consistency criterion used in [3] assumes that *all* operations executed by one participant will eventually reach *all* participants. This approach cannot be used to solve the more general MaS problem, as illustrated in figure 5. The annotation of a triple is only the *id* of the participants that (concurrently)



(a) SU-Set Concurrent Insertion

(b) SU-Set Deletion

Fig. 5: SU-Set cannot solve the MaS problem. After a concurrent insertion (5a) When P2 deletes X, it destroys both instances, (5b), P4 will also destroy both upon reception of the update by P2, violating MaS, as the sources P1 and P3 still have X and P4 has not executed any local operation.

inserted it, in the example, for S in $S4$ there is $S1$ and $S2$. Alternative paths are not taken into account, so when $S2$ deletes X, $S4$ will also delete it, breaking MaS as X still exists in $S3$.

Concerning complexities, when graph is connected and macro-views not selective, TM-Graph is more expensive than SU-Set in space. SU-Set has an space complexity of $\beta * sizeOf(ts)$, meaning that the extra consistency level of MaS costs that when the network graph's density is high, the coefficients of the annotation can overflow. In traffic, SU-Set does not require the cycle detection, but it offloads to the network the causal delivery of updates, which can have hidden costs, like the maintenance and sending of a vector clock.

Collaborative Data Sharing Systems (CDSS) like Orchestra [7] or YouTopia [23] allow collaborative editing by propagating updates on peers through schema mappings. They reduce the problem of to a combination of the well-known problems of data exchange, view maintenance and view adaptation.

Macro-Views can be seen as a restricted type of mappings from a schema to itself, making possible the use of Orchestra. However this has the following shortcomings for our context: (i) The implementation is for the relational model. We need RDF. (ii) Supports a wider type of views, but at the cost of sacrificing their self-maintainability. (iii) It imposes a global ordering on data synchronization. A global epoch number advances each time a participant publishes local updates. Imports and queries are with respect to the data available at the specific epoch where the import starts [10]. This means that the action of synchronization is blocking and that some central control is needed to assign the order of execution. (iv) The stronger consistency guarantees of Orchestra needs the storage of larger provenance information in extra tables, hindering the self-maintenance.

8 Conclusion and Future Work

Making Linked Open Data writable is important issue for transforming Linked Data into a read/write space and allow continuous improvement of the quality of data. In this paper, we proposed a vision of a writable linked data where each LOD participant can define updatable materialized views from data hosted by other participants. Consequently, building a writable LOD can be reduced to the problem of self-maintenance of materialized views. We formalized this problem as the Macro-View Self-Maintenance (MaS) problem. Macro-Views were restricted to Select-Union recursive views to preserve self-maintainability of views and respect the high autonomy of LOD participants.

We proposed TM-Graph as solution for the MaS problem. TM-Graph is a coordination-free protocol that synchronizes macro-views using only the incoming updates from the data sources. TM-Graph annotates quads with elements of a provenance semiring. The number of terms in the annotation corresponds to the number of concurrent insertions of the same quad. Each term has a coefficient that corresponds to the number of simple paths from a quad's origin to the current participant. Consequently, TM-Graph will be efficient for sparse collaboration networks with low overlapping between view definitions.

Compared to copy-modify-merge approaches, TM-Graph supports disconnected graphs and partial replication, thus, a higher consistency level. Compared to CDSS approaches, TM-Graph is adapted for the high autonomy of LOD participants thanks to its coordination-free protocol and self-maintainability of Macro-Views.

Future works will address experimental validations of TM-Graph with various setups on our ongoing prototype [8] and using the provenance stored in the annotations for trust assessing and quality estimation. We also plan to study the performance of TM-Graph on different Collab-Net dynamics, raising the problem of view-adaptation [9]. Finally, in a Collab-Net, each participant can be part of the LOD federation and improve the global availability of data in LOD. However, participants can provide the same data with different degree of divergence that can degrade the performance of existing federated query engines. We want to able to compute a divergence metric in order to help federated query engines to reduce traffic while improving data availability.

References

1. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Morgan and Claypool (2011)
2. Berners-Lee, T., O'Hara, K.: The read-write linked data web. Philosophical Transactions of the Royal Society (2013)
3. Ibáñez, L.D., Skaf-Molli, H., Molli, P., Corby, O.: Live linked data: Synchronizing semantic stores with commutative replicated data types. International Journal of Metadata, Semantics and Ontologies **8**(2) (2013)

[8] `https://code.google.com/p/live-linked-data/`

4. Zarzour, H., Sellami, M.: srce: a collaborative editing of scalable semantic stores on p2p networks. Int. J. of Computer Applications in Technology **48**(1) (2013)
5. Cassidy, S., Ballantine, J.: Version control for rdf triple stores. In: ICSOFT. (2007)
6. Sande, M.V., Colpaert, P., Verborgh, R., Coppens, S., Mannens, E., de Walle, R.V.: R&wbase:git for triples. In: Linked Data on the Web Workshop. (2013)
7. Karvounarakis, G., Green, T.J., Ives, Z.G., Tannen, V.: Collaborative data sharing via update exchange and provenance. ACM TODS (August 2013)
8. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Principles of Database Systems (PODS). (2007)
9. Chirkova, R., Yang, J.: Materialized views. Foundations and Trends in Databases **4**(4) (2011)
10. Taylor, N.E., Ives, Z.G.: Reliable storage and querying for collaborative data sharing systems. In: International Conference on Data Engineering (ICDE). (2010)
11. Gupta, A., Jagadish, H., Mumick, I.S.: Data integration using self-maintainable views. In: EDBT. (1996)
12. Cyganiak, R., Harth, A., Hogan, A.: N-quads: extending n-triples with context. Technical report, DERI (July 2008)
13. W3C: SPARQL 1.1 Query Language. (March 2013)
14. W3C: SPARQL 1.1 Federated Query. (march 2013)
15. W3C: SPARQL 1.1 Update. (March 2013)
16. Ognyanov, D., Kiryakov, A.: Tracking changes in rdf(s) repositories. In: International Conference on Knowledge Engineering and Management (EKAW). (2002)
17. Geerts, F., Karvounarakis, G., Christophides, V., Fundulaki, I.: Algebraic structures for capturing the provenance of sparql queries. In: EDBT/ICDT. (2013)
18. Damásio, C., Analyti, A., Antoniou, G.: Provenance for sparql queries. In: International Semantic Web Conference (ISWC). (2012)
19. Green, T.J.: Containment of conjunctive queries on annotated relations. In: International Conference on Database Theory (ICDT). (2009)
20. Green, T.J., Ives, Z.G., Tannen, V.: Reconcilable differences. Theory of Computer Systems **49**(2) (2011)
21. Schenk, S., Staab, S.: Networked graphs: A declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In: WWW. (2008)
22. Shaw, M., Detwiler, L.T., Noy, N., Brinkley, J., Suciu, D.: vsparql: A view definition language for the semantic web. Journal of Biomedical Informatics **44** (2011)
23. Kot, L., Koch, C.: Cooperative update exchange in the youtopia system. In: International Conference on Very Large Data Bases (VLDB). (2009)