Conflict-free Replicated Hypergraphs

Aruna Bansal
AN & SK School of Information Technology
Indian Institute of Technology Delhi
Delhi, India
email: aruna.bansal@cse.iitd.ac.in

***Abstract*—Hypergraphical structures provide a natural mathematical way to represent the richness of diversified data and complex relationships in hierarchical, relational, navigational, and semi-structured settings, e.g., bibliographic paper submissions, mHealth, and social media applications. These applications operate in distributed environments with a requirement of availability while coping with high network latencies. Replication is the commonly used approach to achieve a high degree of availability, facilitating local query processing. However, replication requires expensive (and often infeasible) concurrency control to ensure consistency. In this work, we specify the *well-formed* Hypergraphs as a Conflict-free Replicated Data Type (HgCRDT), which is a commutative replication-based approach expressed in terms of two 2P-Sets, the latter comprising mutable hyperedges.**

Index Terms*—*Hypergraphs; semi-structured data; complex relationships; well-formed structures; higher-ordered relationships; eventual consistency; data replication; conflict-free replicated data types.

I. Introduction

Hypergraphs are generalized graphs denoted as a pair (N, E) where N is a set of vertices, and E is a set of hyperedges which are arbitrary nonempty subsets of N [1]. Hypergraphs are interesting mathematical structures with application in databases [1]–[4]. Hypergraphs are better-suited than graphs and relational databases to represent complex relationships between hierarchical, navigational, semi-structured data and metadata found in various applications [5]–[8]. *Complex relationships* connect and represent multiple entities and/or relations (to formulate higher-order relations) describing a group of similar entities or a structure. We can find the natural occurrence of complex relationships represented via hyperedges in several applications and data sets, including co-authorship, co-citation, social networks, email networks [6], biological processes [9] [10], and patients' medical history [5]. Higher-order relations can be easily accommodated in hypergraphical structures by employing *higher-order hyperedges* to connect other hyperedges. Traditional data models for data and complex relationships are optimized for particular types of queries and data; for instance, a tabular representation of relational databases is optimized for structured data and relational algebraic queries; and a graph representation of graph databases is optimized for data stored in nodes and edges, as well as navigation and neighborhood queries. Hypergraphs can be used to combine the properties of various data models to cope with the semi-structured, hierarchical, complex, higher-order relationships inherent in such data.

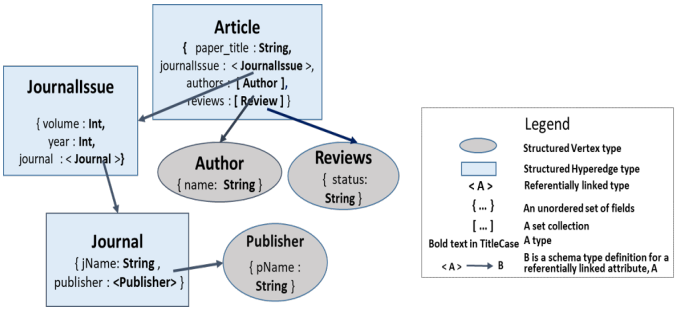

Figure 1. Example: a hypergraph structure capturing an article relationship.

Consider a motivating scenario in which prospective authors submit papers to a journal that are subjected to reviews before being accepted for publication in a journal. The end result is a published paper or a journal article viewed as a relationship between the authors, a collection of reviews, and a journal issue. Journals usually have multiple *issues*; therefore, a journal issue has its publishing year and volume and links the journal (that further relates to a publisher). Note the italicised *issue* refers to journal issue. The paper is an implicit and essential part of the article. Figure 1 depicts various entities (i.e., `Author`, `Reviews`, and `Publisher`) and relationships (i.e., `Article`, `JournalIssue`, and `Journal`) of this scenario in a structured hypergraph via structured vertices and structured hyperedges, respectively, each with a set of fields as their components that are initialized as null.

In this type of real-world scenario, the submission, review, and publication process of conferences/journals are often carried out at many distant domain sites, with each site notifying the other sites of the article's revised status for the next stage. These sites may span over a large geographical area bearing diverse network connections. Therefore, information availability is highly required along with network latencies. Since replication provides availability at the expense of *strong consistency* between the copies, that further necessitate synchronization [11]. Therefore, a weaker notion of consistency is required to ensure the consistency of replicated copies.

We are familiar with consensus algorithm [12] [13] that resolves conflicts between updates, however, at the expense of high reconciliation cost. Here, *Conflict-free Replicated Data Types* or *CRDTs* is a reasonable choice for maintaining consistency in highly dynamic environments [14] [15]. CRDTs address the twin requirements of availability of data (for

 ISBN: 978-1-61208-969-0

efficient local query processing) and operation under network partitioning without requiring complicated concurrency control mechanisms while offering *strong eventual consistency* [16].

Contributions: The purpose of this paper is to discuss how to distribute higher-ordered hypergraphs to multiple replicas in a scalable manner where rejoining replicated copies of hypergraphs from distributed databases is possible without any loss of information. We believe that our work's novelty adds a new dimension to use the rich hypergraphs in distributed settings. Other data distribution challenges, such as security and privacy, are beyond the scope of this paper. We omit specifics about our ongoing implementation to emphasize the suitability of our proposed hypergraphs to be used with conventional CRDTs. On the other hand, the implementation introduces a database paradigm for modeling, storing, retrieving, distributing, and encoding our envisioned hypergraphs.

We aim to leverage the novelty of CRDTs with hypergraph structures and semantics to provide consistent updating and propagation of hypergraphical information across multiple replicas in the previously-stated distributed settings while also ensuring data availability and network latency. Therefore, we extend the existing portfolio of CRDTs [14], [15], [17]–[19] to embrace *well-formed higher-order recursively-defined mutable hypergraph* as a new CRDT: Hypergraph CRDT or HgCRDT. While previous constructions of a CRDT in others have been graphical, hierarchical, list-oriented, key-value based, we believe this is the first instance of well-formed higher-order hypergraphs. In particular, the hyperedges are *mutable* (discussed in Section III-A), in that the set of atoms they connect can be changed. We propose a hypergraph *atom*, a logical term to refer to a vertex or a hyperedge. The mutability of the atoms within a CRDT merits special attention. The hyperedges allow the nesting of hyperedges and are built on references, making their members independent.

In the rest of this paper, we overview background and some related work in Section II. In Section III, we introduce hypergraphs in HgCRDT, and the HgCRDT approach. Next, in Section IV, we present the specification of the HgCRDT that incorporates query, add, remove and modify operations on hypergraphical atoms. A proof-of-correctness showing how concurrent processes meet convergence conditions (essential for eventual consistency) is given in Section V. Finally, in Section VII, we summarise our contributions as well as potential research directions.

II. Background and Related Work

This section will begin by briefly introducing hypergraphs and related work. After that, we will discuss the background of CRDTs and the research aligned with this paper.

A. Hypergraphs

A **hypergraph** is a generalized graph where hyperedges connect more than two vertices. A traditional hypergraph is specified as a pair (N, E) where N is a set of vertices, and E is a set of hyperedges, which are arbitrary nonempty subsets of N as given in [1].

Hypergraphs have been studied since 1980 by various researchers. A few significant work includes: [1] expresses the relational database schemes as hyperedges for ensuring certain degrees of acyclicity (such as α-acyclicity, β-acyclicity, and γ-acyclicity); [20] introduces *Hypernode model* based on nested graphs; [7] proposes *GROOVY*, an object-oriented database model formalized using hypergraphs; [21] proposes a framework for mapping a generic hierarchical/network/relational db model into another using hypergraph; and [22] introduces a schema-oriented graph model with properties and labels using hyper-nodes and hyper-edges.

Existing research on hypergraphs in distributed settings includes, *HyperX* [23] (a scalable hypergraph framework which works in distributed graph settings converting hypergraphs into graphs using a layer built atop Spark), and *Trinity* [24] (a hypergraph database and computation platform over distributed memory cloud).

B. CRDTs

The CRDTs manage distributed replicas of *mutable data* with minimal synchronization and without using complex concurrency control protocols [15]. In CRDTs, different replicas of a data structure can be locally read and written to, replicating the data/operations asynchronously at the distributed locations. The approach applies to data type representations in which the operations performed are conflict-free while ensuring *strong eventual consistency* [16], allowing local modification to the data and then immediately returning to computation.

The CRDTs are designed to work in an underlying *reliable causally-ordered broadcast communication protocol*, in which a source replica (the replica that sends its update information to other replicas) delivers its messages to each downstream replica (the recipient replica) exactly once in an order consistent with happened-before [15]. Maintaining *causal consistency* via a reliable delivery mechanism helps to further reduce inconsistencies between replicas by restricting the operations seen in possibly different orders at the replicas to only *concurrent operations*. Therefore, the same replica can simultaneously send and receive different or redundant messages.

A CRDT specifies an internal data structure representation (called the *payload*), and an collection of *interface operations*, comprising the *query operations* which interrogate the state of the data object and return a value, and the *update operations* which change the internal state of the data object. Both query and update operations may specify *preconditions* that must be satisfied for them to be invoked. Query operations can be performed purely locally, without any need for synchronization and communication with other replicas. Update operations do not return any value, and involve two phases- first, the source site *prepares the parameters* for the updates to be performed at the various replicas; then these changes are *effected* at the various replicas atomically, *immediately* at the source site, and *asynchronously* propagated to the other sites. Usually,

the effect-phase parameters sent to all downstream sites are identical to those at the source phase.

CRDT implementations are classified into Convergent Replicated Data Type (CvRDT) and Operation-based Commutative Replicated Data Type (CmRDT). CvRDTs is a passive replication approach where all necessary information that needs to be replicated is captured by a state which is further transmitted to all the replicas. On the other hand, CmRDTs is an active replication approach where an update operation occurs at the source replica and then is replicated to all the downstream replicas by transmitting operations and performed locally. The eventual transmission of the entire state in CvRDTs may be costly for large data structures. In contrast, operation-based CRDTs transmit only the update operations, which are typically small. However, the communication infrastructure used for CmRDTs must ensure that all operations on a replica are delivered to the other replicas, without duplication, but in any order.

The portfolio of CRDTs [14] [15] includes a variety of interesting data types such as counters, registers, sets, graphs, lists [18] [19], and maps [25]. Of particular interest are the operations-based 2P2P Graph CRDTs [14] [15], since their payload consists of two 2P-Sets for adding and removing vertices and edges where the edge sets are dependent on the vertex sets.

The existing work in the direction of higher-order CRDT includes Riak [25], a distributed NoSQL key-value data store that defines maps as a CvRDT; JSON data structure [26] that composes lists, maps, and registers to embed JSON data types as a CRDT; Logoot [18] that uses a sparse non-mutable *n*-ary tree to nest ordered lists; and higher-order patterns [27]. Causal Graphs [28] illustrates a hierarchical graph-oriented CRDT that represents ordered trees into Causal Graphs. Furthermore, Delta CRDT [29] [30] discusses CRDTs that encode CRDTs using delta mutations of state-based CRDTs. Deltas are temporarily stored in a buffer instead of propagating the entire state to the remote replicas.

An instance of CRDTs employed in databases is the use of *SU_Sets*, a CRDT to handle RDF-Graphs and the SPARQL 1.1 Update operations [31] [32]. The underlying CRDT used in that work is an Operations-based OR-Set of database triples. While the insert and delete operations involve *sets* of elements, these are of a pre-defined atomic element type, in contrast to our higher-order hypergraphs where the set of a hyperedge may include hyperedges belonging to the same hypergraph. More interesting is the insert-delete operation, which uses a *multiset* of mappings when preparing sets of triples to delete and insert into the database.

III. Representation of HgCRDTs

A. Hypergraphs in HgCRDT

We use hypergraphs in HgCRDT, where a (higher-order) *hypergraph* is a collection of *schematic & typed vertices* V and *hyperedges* H. We propose a term hypergraph *atoms* to abstractly refer to the schematic typed vertices and hyperedges. Vertices are assumed to be primitive and represent entities, whereas a *directed* hyperedge is of the form $he(U)$, which connects a *set* of *atoms* U. We formally define hypergraphs in HgCRDT as follows:

Definition 1 (Hypergraphs in HgCRDT). *A hypergraph G in HgCRDT is defined as a collection of hypergraph objects composed of (V, H), where*

- *V is a finite set of vertex objects defined as: $V = \{v_1, v_2, ..., v_n\}$, $n \geq 0$, with each $v_i \in V$ containing only scalar data (such as String, Int, Float, Boolean); and*
- *H is a finite set of hyperedge objects used to represent a relationship, and is defined as: $H = \{he_1, he_2, ..., he_m\}$, $m \geq 0$. Each hyperedge object $he \in H$ connects a finite set of atoms specified as $U = \{u_1, u_2, ..., u_k\}, k \geq 0$, where each $u_i \in (V \cup H)$. The hyperedge he is added to H when the following constraints satisfies avoiding any cycle and self-loop for every $u_i \in he.U$, where $u_i \in H$:*
  1) $u_i \neq he$,
  2) $\forall\ he' \in H$:
     a) $u_i \notin he'.U$
     b) $\forall\ he'' \in he'.U :\ he'' \notin he$ □

Hyperedges are a non-trivial data type, supporting relational structure, hierarchical data, and higher-order relations. A hypergraph is **well-founded** if for every new hyperedge he to be added in H, every atom $u \in he.U$ must exist in $(V \cup H)$. As a consequence, a hyperedge cannot appear within its own set. Notably, we treat atoms as typed objects with a unique implicit identity, avoiding the need to store the entire hyperedge where hyperedge members are themselves (independent) objects. As shown in Figure 2, a few hypergraph objects where, e.g., the independent objects for a journal issue, a set of authors, and a set of reviews are all referentially tied to an article hyperedge object using the implicit object it. Additionally, it aids in the formation of *well-formed* and *acyclic* hypergraphical structures. A hyperedge is said to be **well-formed** when added to H if on adding atom $u \in he.U$, u does not form any cycle and self-loop. Significantly, *acyclic structures* facilitate query optimization and minimize query response time. Hyperedges are **mutable**, in that we permit the set of atoms to be modified. Moreover, hypergraphs are particularly well-suited for replication since a hyperedge's projection containing only some of its set's atoms is still a hyperedge.

Further, similar to vertices that represent entities, each hyperedge retains a set of *internal attributes* (usually of scalar types) that define the properties of a relationship. The internal attributes are different than the referential attributes (i.e., $he.U$) used to define the parts of a relationships based on which the relationship exists. In Figure 1, `paper_title` in `Article`, and `JName` in `Journal` hyperedges are internal attributes. Furthermore, two hyperedges with the same referential set U and carrying the same value are *not* the same. The implicit object ids, the hyperedge type, and the internal attributes of the hyperedges make the hyperedges different. In this paper,

we skip the internal attributes of vertices and hyperedges, and emphasize on using only the referential attributes U in a hyperedge.

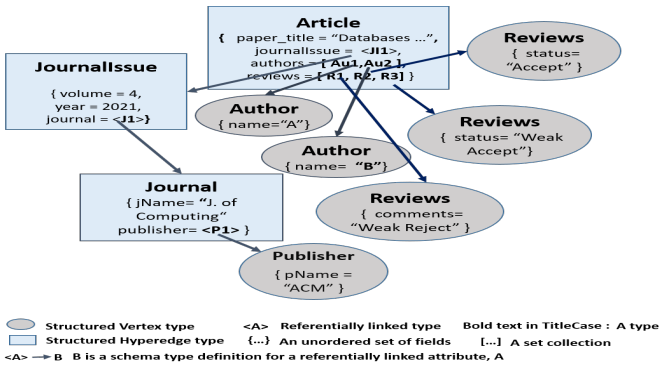

Figure 2. A few hypergraph objects generated for the hypergraph structure of Figure 1.

B. Hypergraph as a CRDT (HgCRDT)

Due to the anticipated size of hypergraphs (for instance, a conference may include sub-conferences, workshops, and a few journals; or a journal/conference may receive a large volume of submissions [33]), we prefer the operations-based commutative (CmRDT) approach over a state-based (CvRDT) or a delta state-based approach as in [30], as transferring the hypergraph state between replicas and merging would be prohibitively expensive.

The communication model of the HgCRDT is similar to that of the CRDTs [15], in that operations are sent in an ordered causal fashion. We employ two 2-phase sets (2P-Sets [15]), i.e., those where elements can be removed after addition, but cannot be reintroduced, as the payload for our hypergraph data type- one each for its vertices V and hyperedges H. Other variants of CRDT Sets are possible, such as OR-Sets, though the commutativity properties need to be carefully verified for each such choice.

We are already familiar with the existing 2P2P graph-based CRDT described in [14]. To facilitate the adoption of our technique, we use the template provided by the 2P2P-Graph specification for hypergraphs. Hypergraphs are generalized graphs dealing with more complex structures than graphs, hierarchies, and maps. The richness of our hypergraphs makes our work different compared to the existing 2P2P-Graph CRDT. Our proposed hypergraph specification uses two tombstone sets (or remove sets: VR, HR) to represent the 2P-Sets, which relaxes in some instances the requirement for a causal order of delivery, and thus permits some additional asynchrony.

Also, note that in hypergraphs, vertices are the base case for atoms (which also include hyperedges) and that hyperedges relate the atoms of a *set* to each other. The novelty of this work lies in this treatment of such well-founded recursive hypergraphical structures. Another novelty is that the set incident on a hyperedge is itself *mutable* 2P-Set. Hyperedges have the following form, in which object references are used to store the set rather than the complete hyperedge itself (in the implementation).

$$he(\textbf{mutable}\ atom\ set\ U)$$

Consequently, hyperedges are mutable, as we may add and remove atoms incident on the hyperedge. The use of tombstone sets allows deletion of an atom from a set; however, since the atoms are implemented as typed objects having their implicit identity, the atoms persist across such modifications. The usage of implicit object identities explains why traditional CRDT models like Key-Value pairs and maps are not suitable for encoding hypergraphs, even after some transformation.

IV. Specification of HgCRDTs

The HgCRDT specification comprises a list of local query operations and global commutative update operations to *add, remove* individual or a set of atoms and *modify* hyperedges. Vertex *modify* is a trivial operation, and therefore, we ignore it in this report. Note that the notion of sources and downstream sites is not statically fixed.

In continuation to our previous example, Figure 3 illustrates a distribution scenario using the HgCRDT framework involving its update operations to capture the journal article's submission, review, and publication processes among three distinct copies. Each update operation begun at a source replica is propagated to subsequent downstream replicas. Note that vertices and hyperedges are introduced to the system in case of no earlier existence, and the outcome is an article with the associated entities and other relationships, as seen in Figure 2. The operation delays affect the payload of a replica in case any other operation needs its prior delivery. As seen in Figure, replica 1 initiates add operations for two vertices for authors A and B and an article hyperedge, which is then shared with replicas 2 and 3. Similarly, replica 2 adds three review vertices. Meanwhile, replica 3 has a vertex for the publisher and two hyperedges for the journal and *issue*. Now, adding the set of reviews, and journal issue at the respective replica (i.e., replicas 2 and 3), necessitates modifying the article hyperedge. Sharing the change operation by both the replicas leads to a concurrent arrival at replica 1. However, this issue is resolved according to the causal order of delivery that reflects both the changes in the article. To add, we assume that a special *issue* of the same journal is introduced to which the article is best suited. Note that the previous instance of the journal issue hyperedge cannot be deleted due to its reliance on the article hyperedge. Thus, the modification of the new *issue* to the article hyperedge enables the deletion of the first *issue*.

Next, we describe the HgCRDT specification that contains a few keywords- *initial* specifies initial values of payload sets at every replica; *let* marks non-mutating statements; *query* and *Update* indicate a non-mutable, and a mutable operations respectively; and *pre* specifies preconditions that must be satisfied for an operation to be invoked. Each update operation has two phases: *prepare at source*, and *effect at downstream*. The initial phase illustrates that an argument is locally prepared by the *source* replica to be delivered to *downstream* replicas.

 ISBN: 978-1-61208-969-0

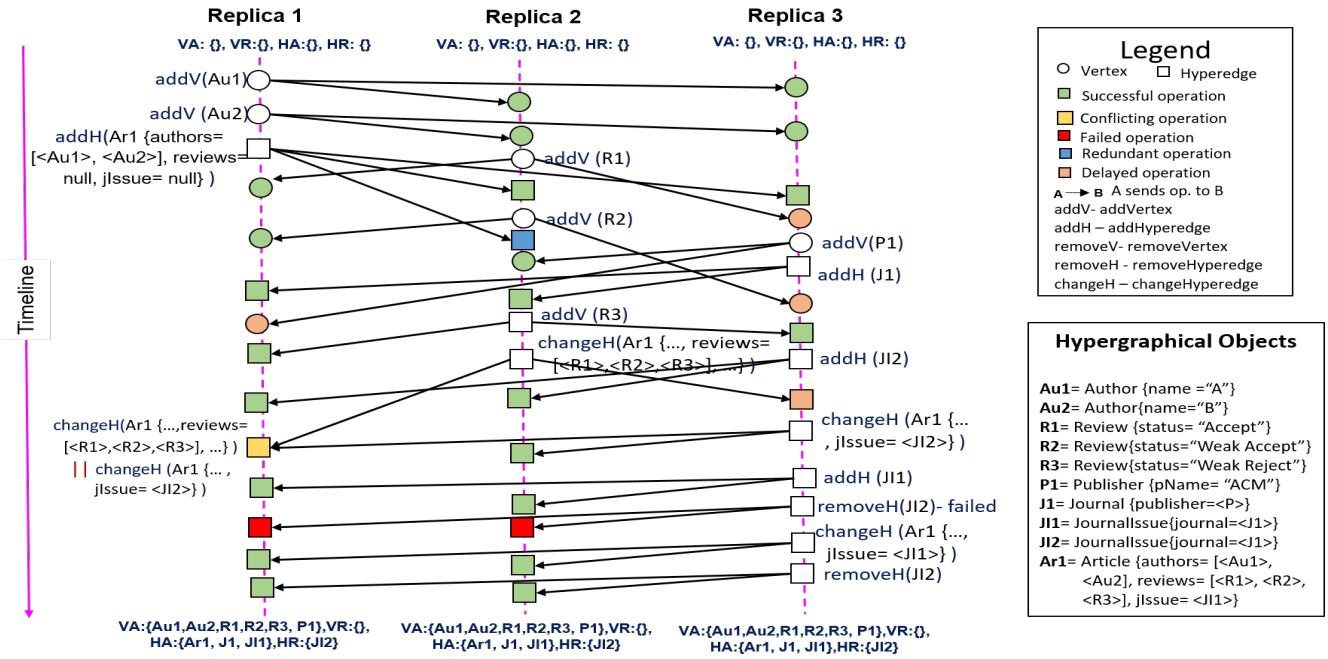

Figure 3. Example: A distribution of HgCRDT operations between three replicas capturing the formation of a journal article relationship (from the paper submission to its publication). Only objects are displayed here, with limited scalar values.

A downstream replica executes the later phase that atomically and asynchronously uses the received argument prepared by the source replica.

a) ***Query operations (Figure 4):****:* The *payload* initializes the local and mutating state of a replica. In HgCRDT, the payload consists of four sets: VA, VR, HA, and HR for adding and removing vertices and hyperedges. The query operations are performed locally at each replica. These operations provide the extensional observational criteria for identifying/distinguishing between the state of the mutable object (hypergraph in this case).

lookupAtom checks for the presence of an atom, whether a vertex (*lookupVertex*) or a hyperedge (*lookupHyperedge*), as the case may be, in the hypergraph. The *lookupAtom* operation is lifted to sets of atoms using conjunction. In the *lookupHyperedge* query, the precondition checks for the existence of all atoms in the set. Since we permit the set to be mutable, the payload sets HA, HR only contain reference-based structures for the hyperedges, and the set is accessed by dereferencing. *within* operation checks that the given hyperedge should be acyclic. Therefore, it recursively checks if a given atom appears within a given hyperedge.

b) ***Update operations****:* are global operations that are defined using the novelty of the CRDT approach. As in, the *source replica* initiates operation and prepares the update information to send to *downstream replicas*. The operation is then *effected* immediately at the source, and if the parameter is non-trivial, also sent asynchronously but reliably to the downstream locations, where it is affected atomically. Causal delivery reduces the need for commutativity to only the concurrent operations, handling the dependency of the hyperedge

$\triangleright$ VA : *vertex add set*, VR : *vertex remove set*,
HA : *hyperedge add set*, HR : *hyperedge remove set*
payload set $VA,\ VR,\ HA,\ HR$
initial ϕ, ϕ, ϕ, ϕ

query $lookupAtom$ (atom a) : boolean b
if a is a vertex: $lookupVertex\ (a)$
otherwise if a is a hyperedge: $lookupHyperedge\ (a)$

query $lookupAtomSet$ (atom set S) : boolean b
let $b = \left(\bigwedge_{\forall u \in\ S} lookupAtom(u) \right)$

query $lookupVertex$ (vertex v) : boolean b
let $b = (v \in (VA \setminus VR))$

query $lookupHyperedge$ (hyperedge $he(U)$): boolean b
let $b = (lookupAtomSet(U) \wedge\ he(U) \in (HA \setminus HR))$

query $within$ (atom a, hyperedge $he(U)$): boolean b
let $b = \begin{cases} true & \text{if } a = he(U) \vee \\ & \quad a \in U \vee \exists\, he(U')\ s.t. \\ & \quad lookupHyperedge(he(U')) \\ & \quad \wedge\ a \in U' \\ & \quad \wedge\ within(he(U'), he(U)) \\ false & \text{otherwise} \end{cases}$

Figure 4. Query Operations

update $addAtom$ (atom a)
  if a is a vertex: $addVertex\ (a)$
  otherwise: $addHyperedge\ (a)$

update $addVertex$ (vertex v)
  prepare at source (v)
  effect at downstream (v)
    $VA := VA \cup \{v\}$

update $addVertexSet$ (vertex set X)
  prepare at source (X)
  effect at downstream (X)
    $\forall v \in X\ : (VA := VA \cup \{v\})$

$\triangleright\ he(atom\ set\ U)$
update $addHyperedge$ (hyperedge $he(U)$)
  prepare at source $(he(U))$
    pre $lookupAtomSet(U)$
  effect at downstream $(he(U))$
    pre $lookupAtomSet(U)$
    $HA := HA \cup \{he(U)\}$

Figure 5. Add Operations

update $removeAtom$ (atom a)
  if a is a vertex: $removeVertex(a)$
  otherwise : $removeHyperedge(a)$

update $removeVertex$ (vertex v)
  prepare at source (v)
    pre $lookupVertex(v)$
      $\wedge\ \forall\ (he\{U\} \in (HA \setminus HR)) :$
      $\neg U.lookupVertex(v)$
  effect at downstream (v)
    pre $addVertex(v)$ delivered
    $VR := VR \cup \{v\}$

update $removeHyperedge$ (hyperedge $he(U)$)
  prepare at source $(he(U))$
    pre $lookupHyperedge(he(U)) \wedge$
      $\forall\ (he\{U'\} \in (HA \setminus HR)) :$
      $\neg U'.lookupHyperedge(he(U))$
  effect at downstream $(he(U))$
    pre $addHyperedge(he(U))$ delivered $\wedge$
      $\forall (he\{U'\} \in (HA \setminus HR)) :$
      $\neg U'.lookupHyperedge(he(U))$
    $HR := HR \cup \{he(U)\}$

Figure 6. Remove Operations

2P-Set on the vertex 2P-Set.

Add Operations (Figure 5): The *addAtom* operation adds a vertex or a hyperedge, depending on the kind of atom specified. Adding a set of hyperedges can be realized by iterating the *addHyperedge* operation. Note that when adding a hyperedge, all atoms in its set must exist, and thus the

$\triangleright\ he(\mathbf{mutable}\ atom\ set\ U)$
update $changeHyperedge$ (hyperedge $he(U)$,
    atom set S^+, S^-)
  prepare at source ($he(U)$, atom set S^+, S^-)
    pre $lookupAtomSet(S^+)$
      $\wedge\ U.lookupAtomSet(S^-)$
      $\wedge\ lookupHyperedge(he(U))$
      $\wedge\ \forall(x \in S^+) :\ \neg\ within(\ x,\ he(U))$
  effect at downstream ($he(U)$, atom set S^+, S^-
    pre $addHyperedge(he(U))$ delivered
      $\wedge\ lookupAtomSet(S^+)$
      $\wedge\ \forall(x \in S^+) :\ \neg\ within(\ x,\ he(U))$
    $\forall(x \in S^-) :\ U.removeAtom(x);$
    $\forall(x \in S^+) :\ U.addAtom(x);$

Figure 7. Modify Operation

corresponding add operations for all these atoms must have been delivered earlier.

Remove Operations (Figure 6): We can only delete an atom incident on a hyperedge after the hyperedge itself has been removed. Note that deleting an atom (whether vertex or hyperedge) requires that it should not be incident on *any* hyperedge (should not be in the set of any hyperedge). Thus the precondition ensures that it cannot possibly appear within any higher-order hyperedge. We do not present here the *remove* operations lifted to a set of atoms.

Modify Operations (Figure 7): It is always possible to modify a hyperedge in a hypergraph by deleting the existing edge and replacing it with the modified edge. It requires ensuring that any atoms present (recursively) within the new set of the new hyperedge must already exist (and must not be the hyperedge itself).

However, since hyperedges are complex structures, this implementation is expensive. Instead, we specify the modification of a hyperedge by the addition or removal of atoms in a set via *changeHyperedge* operation. Note that we now require a set to *itself* be *mutable* 2P2P-Set. The vertices and hyperedges in the sets $U.V, U.H$ are respectively subsets of the two 2P-Sets V, H of the global hypergraph object. By global hypergraph, we mean the replica's state consisting of payload sets, irrespective of any particular hyperedges. The global hypergraph objects V and H may be represented by payload VA, VR, and HA, HR in the tombstone implementation, respectively.

The *changeHyperedge* operation takes an existing hyperedge $he(U)$, and the atom sets S^+, S^- that are to be added to and removed from the set U. For simplicity, assume that $S^+ \cap S^- = \emptyset$, $S^+ \cap\ U = \emptyset$ and $S^- \subseteq U$. For readability, we use the set operations of intersection and subset. These conditions can be expressed in terms of the query operations. Note that in the precondition of *changeHyperedge*, we need to check that the set S^+ being added should exist in the (global) hypergraph, whereas the set being deleted S^- should already be in the *mutable* set of the given hyperedge. Note, in the **effect** phase, the atoms from the various sets are removed/added to the set of the hypergraph. Observe that the

 ISBN: 978-1-61208-969-0

atoms are only removed from the set of the hyperedge, but *not* from the (global) hypergraph because hyperedges are formed using references of existing other atoms.

V. Proof of Correctness

Most of the arguments related to 2P-Sets and 2P2P-Graphs [15] carry over in the proof that this specification implements a CRDT. It is easy to show that *add* operations or *remove* operations on unrelated atoms naturally commute. If, however, an atom appears (recursively) within the set of another atom, then adding the second atom must causally follow the addition of the first atom. The delivery order ensures it. The reverse holds for remove operations. Concurrent $add(u)$ and $remove(u)$ operations on the same atom u, and concurrent $remove(u)$ and $add(w)$ [or $add(u)$ and $remove(w)$] operations where there is a some relationship between u, w, are dealt with using the 2P-Set conditions [15], the conditions on adding or removing atoms, and transitivity.

Operations other than the remove operations are independent of the *changeHyperedge*. The tombstone set will ensure that removal prevails over modifications. Modifications to different hyperedges commute. Consider two concurrent modifications to the same hyperedge with changes S_1^+, S_1^- and S_2^+, S_2^- respectively. We claim that the operations can safely commute (refer to the Lemma 1), resulting in set $(U \cup S_1^+ \cup S_2^+) \setminus (S_1^- \cup S_2^-)$. Atoms appearing in the corresponding add set (or removal set) pose no problem. The assumptions about the sets of atoms being added or removed from a given set within each operation allow the commutation.

Lemma 1. *Concurrent* $changeHyperedge(he,\ S_1^+,\ S_1^-)$ *and* $changeHyperedge(he,\ S_2^+,\ S_2^-)$ *commute.*

Proof. According to the $changeHyperedge$ operation, a set of atoms S^+ are added to, and a set of atoms S^- are removed from a hyperedge. Therefore:

$$changeHyperedge\ (he,\ S_1^+,\ S_1^-) = U \cup S_1^+ \text{ and } U \setminus S_1^- = (U \cup S_1^+) \setminus S_1^-$$

Similarly,

$$changeHyperedge\ (he,\ S_2^+,\ S_2^-) = U \cup S_2^+ \text{ and } U \setminus S_2^- = (U \cup S_2^+) \setminus S_2^-$$

The concurrent execution of both the change operations on each replica on the same hyperedge results:

$$changeHyperedge\ (he,\ S_1^+,\ S_1^-)\ ||\ changeHyperedge\ (he,\ S_2^+,\ S_2^-) = (U \cup S_1^+ \cup S_2^+) \setminus (S_1^- \cup S_2^-)\ ||\ (U \cup S_2^+ \cup S_1^+) \setminus (S_2^- \cup S_1^-)$$

Further, the commutative *set-union* operation makes the results equivalent:

$$(U \cup S_1^+ \cup S_2^+) \setminus (S_1^- \cup S_2^-) \equiv (U \cup S_2^+ \cup S_1^+) \setminus (S_2^- \cup S_1^-)$$

Therefore, modification of concurrent operations to the same hyperedge commute. □

VI. Discussion

Our proposed HgCRDT framework has been implemented in our hypergraph-oriented database system. The system works in the realm of an underpinning object-oriented framework, supporting object re-usability, complex objects, data abstraction, encapsulation, and typing-like features. We use *well-defined schema* and *types* to build hypergraphs where higher-order relationships are formulated on top of other existing relationships and entities without violating schematic acyclic dependencies.

Our system ensures the consistency of hypergraph objects among all the replicas of a distributed domain in its *Consistency layer*, after which each replica immediately stores the objects in its local database. Currently, the distribution process works in a multi-threaded environment (#6 threads). The system stores and retrieves hypergraph objects from its storage and retrieval layers that are built atop *HyperGraphDB* [34]. HyperGraphDB is a general-purpose, portable, extensible, and typed data storage mechanism. We use HyperGraphDB to exploit object-level sharing in higher-order and n-ary relationships.

VII. Conclusions

We proposed hypergraphs as a natural candidate structure for representing semi-structured, hierarchical, navigational, complex, higher-order relationships in distributed computing settings. We introduced and specified a new CRDT, a well-formed higher-order recursively-defined mutable hypergraph named HgCRDT, where hypergraphs were modeled using user-defined schema and system-defined object-oriented types. In HgCRDT, the hyperedges themselves were mutable. The HgCRDT is an operation-based specification of 2P2P sets, which works with tombstone sets.

An extension of our approach introduces and implements partial replication in the HgCRDT in a hypergraph-oriented database model built atop HyperGraphDB. However, we have omitted the specification and details pertaining to the partial replication for clarity of exposition. We are in the process of formalizing our approach to prove it algebraically, giving a detailed mathematical proof of our approach; and studying the performance of the replicated hypergraphs, particularly the scalability of the approach, and evaluating the time and space complexity when dealing with a variety of large hypergraphs on real data. We also intend to compare our approach with other possible Hypergraph specifications such as state-based, and other Set CRDTs-based variations, e.g., OR Set.

Acknowledgments: I wish to acknowledge fruitful discussions and collaborated work of Sanjiva Prasad, IIT Delhi.

References

[1] R. Fagin, "Degrees of acyclicity for hypergraphs and relational database schemes," *Journal of the ACM (JACM)*, vol. 30, no. 3, pp. 514–550, 1983.

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis, "On the desirability of acyclic database schemes," *Journal of the ACM (JACM)*, vol. 30, no. 3, pp. 479–513, 1983.

 ISBN: 978-1-61208-969-0

[3] M. Yannakakis, "Algorithms for acyclic database schemes," in *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, ser. VLDB '81. VLDB Endowment, 1981, p. 82–94.

[4] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, "Directed hypergraphs and applications," *Discrete applied mathematics*, vol. 42, no. 2-3, pp. 177–201, 1993.

[5] S. Prasad, "Designing for scalability and trustworthiness in mhealth systems," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2015, pp. 114–133.

[6] M. M. Wolf, A. M. Klinvex, and D. M. Dunlavy, "Advantages to modeling relational data using hypergraphs versus graphs," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2016, pp. 1–7.

[7] M. Levene and A. Poulovassilis, "An object-oriented data model formalised through hypergraphs," *Data & Knowledge Engineering*, vol. 6, no. 3, pp. 205–224, 1991.

[8] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, p. 1, 2008.

[9] S. Klamt, U.-U. Haus, and F. Theis, "Hypergraphs and cellular networks," *PLoS computational biology*, vol. 5, no. 5, pp. 243 – 255, 2009.

[10] E. Ramadan, A. Tarafdar, and A. Pothen, "A hypergraph model for the yeast protein complex network," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 189–.

[11] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process." Massachusetts Inst of Tech Cambridge lab for Computer Science, Tech. Rep., 1982.

[13] A.-M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel, "The icecube approach to the reconciliation of divergent replicas," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, 2001, pp. 210–218.

[14] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," INRIA Centre Paris-Rocquencourt, Research Report RR-7506, 2011.

[15] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Stabilization, Safety, and Security of Distributed Systems*, X. Défago, F. Petit, and V. Villain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 386–400.

[16] V. B. Gomes, M. Kleppmann, D. P. Mulligan, and A. R. Beresford, "Verifying strong eventual consistency in distributed systems," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 109, 2017.

[17] R. Brown, Z. Lakhani, and P. Place, "Big(ger) Sets: decomposed delta CRDT Sets in Riak," in *Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data*. ACM, 2016, p. 5.

[18] S. Weiss, P. Urso, and P. Molli, "Logoot-undo: Distributed collaborative editing system on p2p networks," *IEEE transactions on parallel and distributed systems*, vol. 21, no. 8, pp. 1162–1174, 2010.

[19] N. Preguiça, J. M. Marquès, M. Shapiro, and M. Letia, "A commutative replicated data type for cooperative editing," in *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2009, pp. 395–403.

[20] M. Levene and A. Poulovassilis, "The hypernode model and its associated query language," in *Proceedings of the 5th Jerusalem Conference on Information Technology, 1990.'Next Decade in Information Technology'*. IEEE, 1990, pp. 520–530.

[21] M. Owrang and L. L. Miller, "An approach for integration of data processing in a distributed enviornment," in *Proceedings of the 17th conference on ACM Annual Computer Science Conference*, 1989, pp. 358–367.

[22] F. Laux, "The typed graph model," in *The Twelfth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA)*, pp. 13 – 19. [Online]. Available: https://www.iaria.org/conferences2020/DBKDA20.html

[23] W. Jiang, J. Qi, J. X. Yu, J. Huang, and R. Zhang, "Hyperx: A scalable hypergraph framework," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 909–922, 2018.

[24] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 505–516.

[25] R. Brown, S. Cribbs, C. Meiklejohn, and S. Elliott, "Riak dt map: A composable, convergent replicated dictionary," ser. PaPEC. ACM, 2014.

[26] M. Kleppmann and A. R. Beresford, "A Conflict-Free Replicated JSON Datatype," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2733–2746, 2017.

[27] A. Leijnse, P. S. Almeida, and C. Baquero, "Higher-Order Patterns in Replicated Data Types," in *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data*. ACM, 2019.

[28] A. Hall, G. Nelson, M. Thiesen, and N. Woods, "The causal graph crdt for complex document structure," in *Proceedings of the ACM Symposium on Document Engineering*, 2018.

[29] P. S. Almeida, A. Shoker, and C. Baquero, "Efficient State-based CRDTs by Delta-mutation," in *International Conference on Networked Systems*. Springer, 2015, pp. 62–76.

[30] P. Sérgio Almeida, A. Shoker, and C. Baquero, "Delta state replicated data types," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 162–173, 2018.

[31] L. D. Ibáñez, H. Skaf-Molli, P. Molli, and O. Corby, "Synchronizing semantic stores with commutative replicated data types," in *Proceedings of the 21st International Conference on World Wide Web*. ACM, 2012.

[32] L.-D. Ibáñez, H. Skaf-Molli, P. Molli, and O. Corby, "Live linked data: synchronising semantic stores with commutative replicated data types," *International Journal of Metadata, Semantics and Ontologies 4, 8*, vol. 8, no. 2, pp. 119–133, 2013.

[33] "Acl2020: General conference statistics," https://acl2020.org/blog/general-conference-statistics/, 2008, [Online; accessed on 06-April-2022].

[34] B. Iordanov, "Hypergraphdb: a generalized graph database," in *International conference on web-age information management*, ser. WAIM'10. Springer, 2010, pp. 25–36.