

Analysis of the Matrix Event Graph Replicated Data Type

Florian Jacob

Karlsruhe Institute
of Technology
Institute of Telematics
florian.jacob@kit.edu

Carolin Beer

Karlsruhe Institute
of Technology
Institute of Telematics
carolin.beer@student.kit.edu

Norbert Henze

Karlsruhe Institute
of Technology
Institute of Stochastics
henze@kit.edu

Hannes Hartenstein

Karlsruhe Institute
of Technology
Institute of Telematics
hannes.hartenstein@kit.edu

November 13, 2020

Abstract

Matrix is a new kind of decentralized, topic-based publish-subscribe middleware for communication and data storage that is getting popular particularly as a basis for secure instant messaging. In comparison to traditional decentralized communication systems, Matrix replaces pure message passing with a replicated data structure. This data structure, which we extract and call the Matrix Event Graph (MEG), depicts the causal history of messages. We show that this MEG represents an interesting and important replicated data type for general decentralized applications that are based on causal histories of publish-subscribe events: we show that a MEG possesses strong properties with respect to consistency, byzantine attackers, and scalability. First, we show that the MEG provides Strong Eventual Consistency (SEC), and that it is available under partition, by proving that the MEG is a Conflict-Free Replicated Data Type for causal histories. While strong consistency is impossible here as shown by the famous CAP theorem, SEC is among the best known achievable trade-offs. Second, we discuss the implications of byzantine attackers on the data type's properties. We note that the MEG, as it does not strive for consensus, can cope with $n > f$ environments with n total participants of which f show byzantine faults. Furthermore, we analyze scalability: Using Markov chains we study the width of the MEG, defined as the number of forward extremities, over time and observe an almost optimal evolution. We conjecture that this property is inherent to the underlying spatially inhomogeneous random walk.

1. Introduction

Matrix¹ is a specification of protocols and their behavior for a middleware that provides communication and data services for decentralized applications. While the size of its public federation is still comparatively small, its utilization rises quickly, and several organizations are deploying large, private federations. Currently, Matrix

is mainly used as the basis of a decentralized instant messaging protocol employed by the French government, the Mozilla foundation, the Federal Defense Forces of Germany, and others.

Matrix implements topic-based publish-subscribe services based on a federated architecture. Similar to e-mail or XMPP, clients attach themselves to a Matrix server, their so-called homeserver, which represents them in the Matrix network. Servers with clients subscribed to a specific topic (called room in Matrix parlance) form a federation to exchange published events independent of other topics. Events can be either communication events or state update events on the stored data. In the instant messaging use case, topics are employed for group or one-to-one communication rooms, communication events are used for instant messages, while the stored data is used for persistent information like room membership or room description.

In contrast to e-mail or XMPP, Matrix replaces pure message passing with a replicated, per-topic data structure that stores the causal history of events. As Matrix servers can thereby synchronize their room's full causal histories, the Matrix approach promises increased decentralized system resilience: After a network partition, a server has significantly stronger means to recover the complete state of the room, i.e., to avoid loss of events. While this increased level of system resilience has been observed by practitioners, the underlying replicated data type has not yet been analyzed thoroughly.

In this paper, we first extract and abstract the Matrix Event Graph replicated data type from the Matrix specification and denote it by MEG. A MEG is a Directed, Acyclic Graph (DAG) made up of vertices which represent communication and data storage update events, and directed edges which stand for potential causal relations between events.

Because the graph represents the potential causal order of events, a correct graph is inherently cycle-free. Appending new events is the only write operation supported by the Matrix Event Graph, which makes it

¹<https://matrix.org/>, <https://matrix.org/spec/>

append-only — and a candidate for Distributed Ledger Technologies. Thus, the MEG can be considered as a fundamental concept for various applications that are based on causal histories, ranging from decentralized crowdsensing databases in Internet of Things scenarios over decentralized collaboration applications to decentralized push notification systems. Since, for Distributed Ledger Technologies, it has been conjectured that consistency, decentralization, and scalability cannot be achieved simultaneously [26, 19], our analysis focuses on these aspects.

As main contribution we therefore provide an analysis of the degree to which the MEG fulfills consistency, deployability in decentralized scenarios, and scalability:

Consistency: In accordance with the CAP theorem [7], and since Matrix provides availability and partition tolerance, the MEG necessarily has to sacrifice strong consistency. We show that Matrix provides Strong Eventual Consistency by proving that the MEG is a Conflict-Free Replicated Data Type (CRDT) [22] for causal histories.

Decentralization: We discuss the implications of byzantine attackers on the specific type of CRDT that the MEG represents. The avoidance of consensus is the primary reason that allows the MEG CRDT to facilitate $n > f$ environments with n total participants of which f exhibit byzantine faults.

Scalability: The inherent probabilism of uncoordinated, concurrent updates on a MEG is the main challenge for the analysis of the MEG with respect to scalability. We are interested in the width of the MEG in terms of the number of forward extremities, i.e. ‘vertices without children’, over time. We study the width of the MEG using a formalization by means of Markov chains. We observe that the MEG does not degenerate, and conjecture that this non-degeneracy is inherent to the underlying spatially inhomogeneous random walk.

This paper is structured as follows: We start with a more detailed description of how the MEG works and the problem statement in Section 2. Section 3 presents related work and background on replicated data types. Assumptions and architecture are given in Section 4. The inner working of the MEG is formalized in Section 5, which is then used to prove that it is a Conflict-Free Replicated Data Type. In Section 6, we perform a reality check of the utilized assumptions of Section 5 and discuss how the MEG can be made byzantine fault tolerant. Section 7 formalizes the stochastic behavior of the width of the MEG and provides evidence that the width always evolves to a near-optimal value, and does so fast. We conclude the paper in Section 8.

2. MEG: Overview and Problem Statement

In the following and for illustration purposes, we often make use of the instant messaging use case of Matrix, but we want to emphasize that the Matrix Event Graph

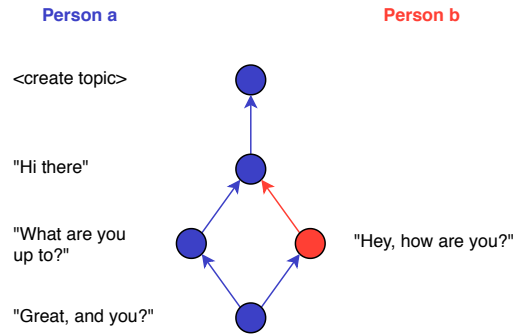


Figure 1.: Basic example of a MEG

is a general replicated data type for append-only causal histories of publish-subscribe events. We also typically focus our studies on a single MEG instance, and therefore a single broadcast domain associated with that MEG. However, several independent MEGs can coexist. A sample MEG is exhibited in Fig. 1.

General MEG setup. As mentioned before, a MEG is a Directed, Acyclic Graph (DAG). One MEG represents the message history and attributes of a group or 1:1 chat, and it is replicated independently by all participating servers. Upon creation, the DAG consists of only a single vertex, the *root vertex*. Each vertex in the DAG corresponds to an application-defined publish-subscribe event, e.g., to a text message or temperature reading. Edges represent potential causal relationships between events: When a new vertex is added, it is appended to the existing DAG through one or more outgoing edges. These edges point towards vertices that had no incoming edges before, i.e., the newest events in causal history, which we from now on call the *forward extremities* of the DAG. The selection of forward extremities is done according to the current knowledge of the adding replica. This potential causal relationship is known as the *happened before* relationship², as defined by Lamport [13]: For $a \leftarrow b$, we say a happened before b . Edges thereby form a partial order that is consistent with the causal order in which events took place.

In addition to being directed, acyclic, and representing the causal order of events, the MEG is also weakly connected since all newly added vertices have at least one outgoing edge. The root vertex, as the only vertex without outgoing edges, is therefore the unique minimal element of the partial order represented by the DAG. DAGs with this specific structure are called *rooted* [16].

Adding a new vertex to the source replica. The replica that creates an event on behalf of a client and appends it as a vertex is called *source replica*. When it adds a vertex, the corresponding event could be causally

²Note that Lamport defines a *happened before* b as $a \rightarrow b$. In this paper, we actually use the converse relation $b \rightarrow a$, as common for Distributed Ledger Technologies, so that new references can be stored as part of the new vertex, and old vertices can be kept immutable. It follows that for $b \rightarrow a$, we say a is the parent of b .

related to previous events. Thus, all forward extremities should be included as edges. Replicas can experience a high number of forward extremities caused by latencies or partitions, and malicious replicas could forge events with a high number of parents. However, certain algorithms executed on the MEG do not scale well with the number of parent events, i.e., they can become very resource intensive, especially when old parts of the MEG are referenced as parents [8]. In practice, the maximum number of parent events therefore has to be restricted to a finite value d . If there are more than d forward extremities, a replica selects a subset of size d for the new event. For the potential causal order relation in the MEG still to be consistent with the actual causal order, clients have to inform the replica about actual causal dependencies so that those are included as parents.

Updating all replicas. Beyond appending the new vertex to the local DAG, the source replica also needs to synchronize with the other replicas. The replica sends a DAG update that consists of the new vertex and edges to all replicas using a broadcast protocol. On reception of an update, replicas append the new vertex to their DAG via the new edges selected by the source replica as soon as all required parent vertices exist in the local replica. In case the parent vertices are not (yet) available, the update is buffered until they are.

Dealing with concurrent updates. When clients at two different replicas concurrently invoke updates, each replica thinks of their vertex as the single next step in causal history represented by their DAGs, i.e., both deviate from the last consistent DAG state. In case of continuous synchronization failure, e.g. due to a network partition, additional client updates will enlarge the inconsistency between the replicas' DAGs and lead to two causally independent chains of events, built from the last synchronized event. Both replicas will continue to try to synchronize their state with other replicas. When the partition heals, all replicas will eventually receive all updates. As depicted in Fig. 1, instead of trying to find a linear order of updates and to solve conflicts with rollbacks, the concurrent DAG states are merged by attaching both causally independent chains of events to the last synchronized event, i.e., by forking the DAG. This acceptance of concurrency in the data type itself by only providing a partial order on events is the core idea of the Matrix Event Graph. It is also the basis for our proof of conflict-freedom in Section 5. A fork in the DAG introduced by concurrency will lead to two causally independent forward extremities. Following the attachment rules for new vertices, a replica that has received and appended both causally independent chains to its DAG selects both as parents for a new vertex. In terms of graphs, this means that the new vertex will join both chains again, which marks that the period of concurrency and causal independence is over, and reduces the number of forward extremities by one.

Problem statement. The way in which concurrency is handled in a MEG as well as the use of vari-

ous parameters as outlined above give rise to the key research questions addressed in this paper: Which consistency guarantees can application developers expect from a MEG — and under which assumptions do they hold? And: Can the width of the MEG degenerate? The preceding explanations describe how the MEG is available under partition, and how it tries to achieve Eventual Consistency, as conjectured by the Matrix developers [5]. In this paper, we provide a proof of Strong Eventual Consistency in Section 5. In Section 6, we relax the employed assumptions, particularly on the communication primitive. In addition, the overview above showed that if the number of vertex parents is restricted to d and selected randomly, the evolution of the number of forward extremities u , i.e., the width of the DAG, is non-trivial in concurrent environments. In Section 7, we explore whether for arbitrary start values of u , if k replicas continuously select d parents independently and then synchronize the new vertices, the width of the DAG converges in a sufficiently small number of iterations. In addition, we explore how the choice of the number of parent vertices d affects the speed of convergence.

Not in scope of this paper: While we make assumptions on and deal with the underlying broadcast communication primitive, we consider the topic of broadcast communication per se beyond the scope of this paper. Moreover, Matrix employs an access control system for MEGs, which we will not consider further, but which has been examined in [10].

3. Related Work & Background

Jacob et al. investigated quantitative aspects of the public Matrix federation, and found scalability problems with the broadcast communication currently employed by Matrix [9]. However, they did not investigate the scalability and other properties of the replicated data structure itself. The access control system of Matrix, which builds on top of the MEG, was very recently studied in [10]. Privacy and usability aspects of Matrix, along with a CRDT-based vision on how to improve this situation in federated networks in general, are the topic of [1].

In the field of replicated data types, Shapiro et al. introduced the category of Conflict-Free Replicated Data Types (CRDTs), together with a new consistency model provided by the category, namely Strong Eventual Consistency [22]. Following the initial definition, new papers mostly focused on implementations of the data type like the JSON-CRDT by Kleppmann et al. [12], or extended the base concept of CRDTs [4].

The initial CRDT concept was overhauled in cooperation with the original authors in [18]. We will mainly use the new CRDT terminology introduced there.

3.1. Consistency Models

The inherent trade-off between *Consistency* and *Availability* in the presence of network partitions in dis-

tributed systems led to the definition of a variety of consistency models. A well-known consistency model is *Eventual Consistency* (EC), which provides the following guarantees [22]:

- *Eventual Delivery*: An update applied by one correct replica is eventually applied by every correct replica.
 - *Termination*: Every invoked method terminates.
 - *Convergence*: Correct replicas that applied the same set of updates eventually reach equivalent states.
- Strong Eventual Consistency (SEC)* builds on top of EC, and strengthens Convergence [22]:
- *Strong Convergence*: Correct replicas that applied the same set of updates have equivalent states.

Whether two states are equivalent is application-dependent. In our case, the state of two replicas is equivalent if their graphs consist of identical vertices and edges. Note that “the same *set* of updates” means that while the updates are identical, they might be received or applied in different order. The key difference between Convergence and Strong Convergence is that with Convergence, replicas may coordinate with other replicas to find agreement on their state even after having applied updates. Especially if the ordering of updates matters, this can lead to rollbacks. With Strong Convergence, the agreement has to be immanent and implicit.

3.2. Conflict-Free Replicated Data Types

Conflict-Free Replicated Data Types (CRDTs) were first formalized in [22]. CRDTs are an abstract data structure that allows for optimistic update execution (cf. [20]) while guaranteeing conflict-freedom upon network synchronization. The system model of CRDTs is based on a *fail-silent* abstraction with a Causal Order Reliable Broadcast communication protocol (see Section 4). For objects that implement a CRDT in a system with n replicas, Shapiro et al. show that SEC is ensured for up to $n - 1$ replica failures [22].

Two conceptually different, but equally expressive types of CRDTs are the *operation-based* and the *state-based* CRDT. Replicas implement functions to be invoked by clients to access or modify the state. The key difference between operation- and state-based CRDTs lies in the way of synchronization: In state-based CRDTs, all replicas periodically send their full state to all other replicas which then merge states. In contrast, operation-based CRDTs only synchronize upon changes. Source replicas transmit state changes resulting from a client invocation as operations. In Section 5, we show that the MEG is an operation-based CRDT.

Operation-based CRDTs implement functions that can be classified as **update** or **query**. A **query** function returns information on the current state of the replica. Their counterpart, **update** functions, modify the state. They comprise two steps: At first, a **generator**³ step is executed by the source replica. It is side-effect-free, but

³Originally introduced as *prepare-update*

returns an *operation*, i. e., an encapsulation of the state changes. A common example of a **generator** step is the creation of a unique object identifier for **update** functions that add an object to the state. The second step is called **effector**⁴ step, it must be executed at every replica. Thus, the source replica transmits the generated operation to all replicas using broadcast. Upon reception of an operation, each replica executes the **effector** step locally and applies the resulting changes to their state. [23]

In general, the data structure of a CRDT cannot maintain a specific shape or topology, such as a DAG, as concurrent updates could violate invariants. Specific implementations of CRDTs can overcome this restriction however, for example shown by the *Operation-based Add-only monotonic DAG* described in [21]. Their implementation allows clients to collaboratively edit a DAG, by adding vertices and edges in separate updates. Topology preservation is enforced by rejection of new edges that violate the current partial order of the DAG. In a similar vein, the MEG is designed in a way that preserves its topology as rooted DAG inherently, which we will show in Subsection 5.2.

4. Assumptions and Architecture

We assume a finite and known set of replicas, each storing a full local copy of the MEG.

Assumptions. We make use of two failure models, both based on the *asynchronous* timing assumption, which means that no upper bounds on computation or network transmission times are given. The *fail-silent* model [3, p. 63] implies that faulty replicas can crash-stop at any time, while the remaining replicas have no means to reliably distinguish failure from communication or processing delays, i. e., the fault is ‘silent’. The *fail-silent-arbitrary* model [3, p. 64] allows for arbitrary, i. e. byzantine, behavior of faulty replicas. This includes intentionally malicious behavior. In this model, ‘silent’ also means that replicas cannot detect whether another replica currently adheres to the protocol or not.

We call a replica *correct* if it is non-faulty. A fault is the failure to adhere to the protocol. Additionally, in the fail-silent model, a replica is also considered faulty if it is crashing infinitely often, remains crashed forever or loses its memory upon recovery. [3]

The formal CRDT-proof that we give in Section 5 is based on the stricter assumption of a *fail-silent* model. In Section 6 we extend the claims to the *fail-silent-arbitrary* model.

Furthermore, we make use of two broadcast abstractions in this work. Firstly, we use *Reliable Broadcast*. Informally, this abstraction provides a set of properties that guarantee that eventually, the same set of messages is received by all correct replicas, even if the sending replica fails [3].

- *Validity*: If a correct replica sends a message m , then

⁴Originally introduced as *effect-update*

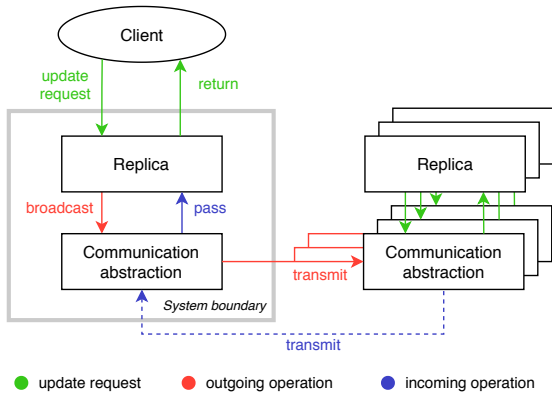


Figure 2.: An update request by a client invokes the generator of an update function at the replica, which creates an update operation. This update operation is then transmitted to all replicas, including the calling replica itself, through the communication abstraction. The communication abstraction enforces guarantees about incoming operations, e.g. on their ordering.

it eventually receives m .

- *No duplication*: Messages are received only once.
- *No creation*: If a replica receives a message m with sender p , then m was previously sent by p .
- *Agreement*: If a message m is received by some correct replica, m is eventually received by every correct replica.

The other, more powerful, abstraction is called *Causal Order Reliable Broadcast*. It extends the guarantees of Reliable Broadcast by also preserving the *causal order* of messages [3]:

- *Causal Delivery*: For any message m_1 and m_2 where the broadcast of message m_1 happened before (cf. [13]) the broadcast of message m_2 , m_2 is only received by replicas that have already received m_1 .

The formal CRDT-proof in Section 5 is based on the *Causal Order Reliable Broadcast* abstraction. In Section 6 we relax this assumption to *Reliable Broadcast* — even in byzantine scenarios — while maintaining the CRDT properties.

Architecture. As we can see in Fig. 2, each *client* is attached to a single *replica* in which it trusts. The client can request functions of class `query` or `update` at their replica, as defined in Subsection 3.2. As part of executing an `update` function, the source replica distributes operations, i.e., encoded state changes, to all replicas using a broadcast *communication abstraction*.

A more granular architectural view is provided in Fig. 3. Inside a replica, the *Reference Monitor* is the entry point for incoming requests from clients and operations from remote replicas. It serves as a gate keeper to prevent further processing of operations or requests that violate the protocol or, in a byzantine setting, originate from unauthorized or unauthenticated parties. Operations and requests that pass the Reference Monitor

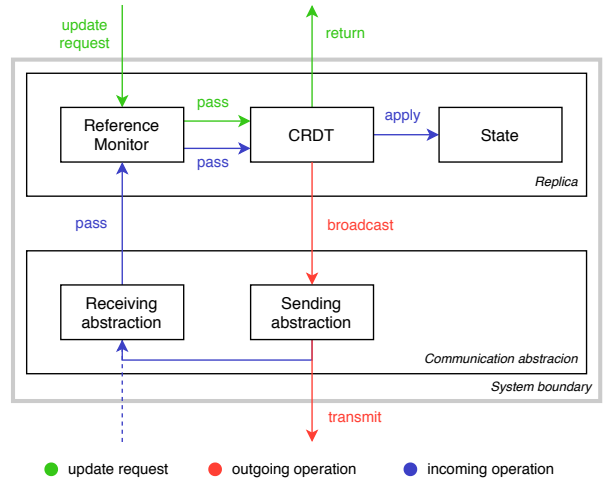


Figure 3.: Inner workings of the source replica and communication abstraction when receiving an update request. After entering the replica through the Reference Monitor, it is passed to the CRDT. The CRDT encodes the state changes as an operation which is then broadcasted to all replicas using the communication abstraction. Incoming update operations, again, pass the Reference Monitor before being processed at the CRDT component. The CRDT then applies them to the local state of the replica.

are handed to the *CRDT*. The CRDT can read and modify the state of the replica and is thus the core logic module of the replica. In case of a `query` request, it accesses the state and returns the desired value. For `update` requests, the generator of the update function encapsulates state changes into an operation that is passed to the communication abstraction. The CRDT then returns to the client to indicate success. The communication abstraction sends the update operation to all replicas, including the calling replica.⁵ These update operations then trigger the local update effector which applies the changes to the state of the replica.

5. The MEG as CRDT

Building upon the overview given in Section 2, we formalize the MEG as an operation-based shared object. We show that the MEG is a CRDT and thereby provides Strong Eventual Consistency (SEC). The underlying assumption for this section is a fail-silent model with Causal Order Reliable Broadcast. This is in accordance with the assumptions used by Shapiro et al. for CRDTs (cf. Subsection 3.2) [23].

5.1. Formalization of the MEG

To define the Matrix Event Graph as a CRDT, we adopt the formal definition introduced with the concept of operation-based CRDTs in [23, 22] and use the pseudo

⁵While, depending on the specific communication abstraction, this is not required in an actual implementation, it is important on a conceptual level to ensure that the guarantees hold.

code notation by Preguiça [17].

An object is formally defined as (S, s^0, q, t, u, P) : S is the space of possible per-replica states, and $s^0 \in S$ is the initial state of every replica. q is the set of query functions. update functions are composed of a generator step t and an effector step u . The effector u may contain a delivery precondition P , which must be fulfilled before an operation is being processed further. Notably, P only delays the execution, it does not abort the effector step. When a replica with state $s \in S$ executes a step u , we denote this as $s \bullet u$, which yields a new state. As shorthand for the state at replica i , we write $s_i \in S$.

We provide a pseudo code implementation of the MEG as an operation-based CRDT in Listing 1. A vertex is a tuple (e, w) that represents an event in the MEG. w is a unique identifier for the event, whereas e contains the actual event. Edges represent a potential causal relationship between child and parent vertex. The state is a DAG, defined through a set of vertices and a set of edges. Initially (s^0), it consists of a single vertex and no edges. The query functions `lookup`, `hasChild`, `getExtremities` and `getState` allow to access the replica state without modification. `lookup` checks whether a vertex with a given identifier is part of the current state. Similarly, `hasChild` checks for the existence of child vertices for a given vertex. `getExtremities` returns the current set of forward extremities, whereas `getState` returns the state. The update function `add` is used to append new events to the MEG. Its generator step t_{add} takes the event e as input argument. Based on the state of the source replica at that time, a set L of forward extremities is created. Lastly, a unique identifier w is chosen. The parameters w , e and L , and a reference to the update function `add` are returned together and constitute the update operation.

The effector u_{add} is invoked by the operation that was created in the generator step. Once the delivery precondition P is fulfilled, the new vertex (e, w) and the new edges $((e, w), (e_p, w_p))$ for each $(e_p, w_p) \in L$ are added to the state, i.e., the set of vertices and edges, respectively. Since `add` is the only update function, we will drop it as a subscript for the steps t and u from now on.

5.2. Preservation of the DAG topology

As mentioned in Subsection 3.2, the preservation of a specific shape, such as a DAG, is not possible in a generic way for CRDTs. We now show that the MEG always preserves the desired data structure of a rooted DAG by design as Lemma 2.

Lemma 1. *There is at least one forward extremity at any time after initialization of the MEG.*

Proof. By induction.

Base case: After initialization of the MEG, the DAG consists of a single root and no edges. Therefore, the

Listing 1.: Pseudo code implementation of the Matrix CRDT. `query` and `update` indicate the type of the respective functions, `generator` and `effector` denote the two steps of an update function. `pre` is the delivery precondition P .

```

state set  $S = (V, E)$  // vertices  $V$  consist of
    event  $e$  and uid  $w$ :  $(e, w)$ ,  $E$  are edges:
     $E \subseteq V \times V$ 
initial  $(\{(e_0, w_0)\}, \emptyset)$ 
query lookup (uid  $w$ ) : boolean
    return  $\exists((e', w') \in V) : w' == w$ 
query hasChild (vertex  $(e, w)$ ) : boolean
    return  $\exists((e', w') \in V) : ((e', w'), (e, w)) \in E$ 
query getExtremities () : list of vertices
    return  $L = \bigcup_{(e, w) \in V: \text{not hasChild}((e, w))} \{(e, w)\}$ 
query getState () : set
    return  $S$ 
update add
    generator (event  $e$ )
        let  $L = \text{getExtremities}()$ 
        let  $w = \text{unique}()$ 
        return add,  $(e, L, w)$ 
    effector (event  $e$ , list of vertices  $L$ , uid
         $w$ )
        pre:  $\forall(e_p, w_p) \in L: \text{lookup}(w_p)$ 
             $V = V \cup \{(e, w)\}$ 
             $E = E \cup \bigcup_{(e_p, w_p) \in L} \{(e, w), (e_p, w_p)\}$ 

```

root is a forward extremity as it has no incoming edges. *Induction step:* Given a valid MEG, executing `add` appends a new vertex with only outgoing edges. Thus, that new vertex is a forward extremity. \square

Lemma 2. *The MEG maintains the properties of a rooted DAG at all times: (i) single root, (ii) acyclicity, and (iii) weak connectedness.*

Proof. By induction.

Base case: The initial state s^0 contains a single vertex and no edges. This MEG therefore is a rooted DAG.

Induction step: Given replicas i with state $s_i = (V_i, E_i)$, where s_i is a rooted DAG, an arbitrary source replica r is selected. As part of the generator step t , the set of forward extremities is determined as L , and a unique identifier w created. By Lemma 1, $|L| > 0$. Since t is side-effect-free, the MEG remains unchanged.

Consequently, the execution of the effector step u is triggered at each replica i . u awaits the fulfillment of the delivery precondition P , which ensures that s_i contains all parents that are referenced by L . Finally, applying u yields the new replica states s'_i :

$$s'_i = (V_i \cup \{(e, w)\}, E_i \cup \bigcup_{(e_p, w_p) \in L} \{(e, w), (e_p, w_p)\}).$$

Since all new edges are outgoing from the new vertex (e, w) , no new cycles can be formed, and existing roots remain roots. No new roots or isolated vertices have been added as the new vertex has outgoing edges. Because all s_i were assumed to be rooted DAGs, all s'_i must be rooted DAGs. \square

5.3. Proof of CRDT properties

Now, we show that MEG implements an operation-based CRDT and thus guarantees SEC. We structure the proof by the SEC properties Strong Convergence, Eventual Delivery, and Termination (cf. Subsection 3.1).

Strong Convergence. For Strong Convergence, we need to show commutativity of concurrent updates and causal order reception of operations for noncommutative updates.

Commutativity for updates is determined by the commutativity of their operations. Two updates (t, u) and (t', u') commute, iff for any reachable state $s \in S$ for which the delivery precondition P is satisfied for both u and u' : (i) P is still satisfied for u in $s \bullet u'$, and (ii) $s \bullet u \bullet u' \equiv s \bullet u' \bullet u$. [22]

Lemma 3. *Once an update operation satisfies P for some state s , it will continue to satisfy P for any state s' following s .*

Proof. Consider any update operation $u(e, L, w)$ that satisfies P in some state $s = (V, E)$. Applying an arbitrary operation $u(e', L', w')$ to s yields a new state s' :

$$\begin{aligned} s' &= s \bullet u(e', L', w') \\ &= (V \cup \{(e', w')\}, E \cup \bigcup_{(e_p, w_p) \in L'} \{(e', w'), (e_p, w_p)\}) \end{aligned}$$

P being satisfied in s implies that it remains satisfied for s' :

$$\begin{aligned} \forall (e_p, w_p) \in L : (e_p, w_p) \in V \\ \Rightarrow \forall (e_p, w_p) \in L : (e_p, w_p) \in V \cup \{(e', w')\} \end{aligned}$$

□

Lemma 4. *Any two operations $u(e_i, L_i, w_i)$ and $u(e_j, L_j, w_j)$ commute with each other.*

Proof. We consider any state $s = (V, E)$ and two update operations $u(e_i, L_i, w_i)$, $u(e_j, L_j, w_j)$ that both satisfy P in s .

As shown in Lemma 3, after applying one operation, the other operation still satisfies P . It remains to show that the resulting states are equivalent, regardless of the order in which the effectors are executed. Since u only performs a union of the edge and vertex sets, by commutativity of the union operator, commutativity of u follows: $s \bullet u(e_i, L_i, w_i) \bullet u(e_j, L_j, w_j) \equiv s \bullet u(e_j, L_j, w_j) \bullet u(e_i, L_i, w_i)$ □

As we have shown, MEG updates are commutative and Strong Convergence is guaranteed. This is possible because all required properties of the MEG are preserved by design (cf. Lemma 2).

Since the MEG encodes causal relations as edges in the data structure, the delivery precondition P can ensure that these dependencies are respected without sacrificing commutativity.

Eventual Delivery. For Eventual Delivery, we need to show that P is eventually satisfied for all operations.

Lemma 5. *P is immediately satisfied on causally ordered message reception.*

Proof. P ensures that all referenced parents are part of the local **state**. Since `getExtremities` selects all parents from the current **state**, P must be satisfied at the source replica after the **generator** step. Once satisfied, P remains satisfied since vertices are never removed. Therefore, receiving all causally preceding operations is sufficient to satisfy P at every replica. Consequently, having causal order message reception, P is immediately satisfied on reception. □

Termination. Given the implementation in Listing 1, we can see that there are no loops or recursive calls in either of the functions, therefore, they will eventually exit. Knowing that P is immediately satisfied given causal order message reception, as shown in Lemma 5, we can conclude that Termination holds.

Conclusion. We have shown Termination and eventual satisfaction of P . Lemma 4 shows commutativity of concurrent updates. Therefore, all properties of an operation-based CRDT are met by the MEG.

6. Relaxation of Assumptions and Reality Check for Byzantine Settings

In this section, we evaluate the assumptions we have used for the CRDT proof of the MEG in Section 5 and relax them wherever possible without violating previously shown guarantees. We show that Matrix currently provides no SEC because of its unreliable broadcast protocol. However, when having a Reliable Broadcast abstraction that provides Validity and Agreement, the MEG can provide SEC in byzantine $n > f$ environments with n total and f faulty participants. This is possible since conflicts, created by byzantine replicas that share different update operations with different replicas, can always be resolved.

6.1. Relaxation of the Broadcast Assumptions

In Section 5, we assumed a Causal Order Reliable Broadcast abstraction, which is commonly used with CRDTs. Yet in reality, the communication abstraction employed by Matrix provides much weaker guarantees. We thus revisit the assumptions and show that the Causal Delivery property of the broadcast abstraction is not necessary⁶ and can be removed without violating Strong Convergence for safety as well as Eventual Delivery and Termination for liveness (cf. Section 3 for definition and Section 5 for fulfillment).

Strong Convergence. To provide Strong Convergence, replicas must receive noncommutative update operations in their causal order. As every update operation commutes with every other, as shown in Lemma 4,

⁶The No Duplication property is also not necessary: Because each vertex has a unique identifier w , and outgoing edges cannot be added afterwards, it suffices to make the effector conditional on the presence of the vertex in the replica state to gain idempotent effectors that can cope with multiple receptions of identical operations.

Strong Convergence does not require any ordering guarantees by the communication abstraction.

Eventual Delivery. In Lemma 5, we used the Causal Delivery property to show that the delivery precondition P is immediately satisfied. However, Eventual Delivery only requires that correct update operations received by a replica *eventually* satisfy P , so that they can eventually be applied.

It therefore remains to show that the delivery precondition P is eventually satisfied without Causal Delivery. Given an update operation, P is satisfied if all referenced parents are part of the `state` of a replica. If an operation satisfies P at some point in time, it continues to satisfy P thereafter, because the MEG is an append-only data structure. As per Lemma 5, P is satisfied for any given operation after the `generator` step at the source replica finishes. Therefore, all referenced parents must have been previously added to the `state` and therefore be part of some update operation. If an update operation does not satisfy P at some replica due to reordering of operations by the broadcast abstraction, replicas can delay and buffer the update operation until P is satisfied. Owing to the Validity and Agreement properties of the broadcast abstraction (cf. Section 4), all missing update operations are eventually received by all correct replicas. As correct replicas apply all operations that they received and that satisfy P , all parents must eventually be part of their `state`. Consequently, for correct replicas, P must eventually be satisfied for every update operation.

Termination. Since all method executions terminate, and since we have shown that in the new setting, P is eventually satisfied for all operations, the Termination property still holds.

Thus, the MEG only requires a weak form of Reliable Broadcast, and does not depend on Causal Delivery.

6.2. Tolerating Byzantine Failures

In the following, we replace the *fail-silent* failure model with the *fail-silent-arbitrary* model. We assume that the adversary cannot permanently block broadcast communication between two correct replicas. In a system with n replicas, the adversary can induce byzantine faults in up to f replicas with $n > f$. This means that a client’s trusted replica might be the only correct replica in the system. As the MEG does not strive for consensus, it is able to cope with such a hostile environment. To model the capabilities of byzantine replicas in a distributed systems that implement a CRDT, Zhao et al. introduce a three-part threat model [27] which consists of attacks on the membership service, malicious updates, and attacks on the Reliable Broadcast service. To keep focus on the MEG, we will only touch on the issues related to the membership service and malicious updates, and put the attack on the Reliable Broadcast service at the center of attention.

Membership service. With respect to a membership service, we assume a known set of replicas that does

not change. Still, we want to note that attacks on the membership service for dynamic groups may prevent replicas from receiving some or all update operations, which could affect Eventual Delivery. We consider this as an important, but somewhat separate topic.

Malicious updates. Malicious replicas could attempt to inject updates into the data structure that are not compliant with the protocol. In general, to address threats from malicious updates, the Reference Monitor is the endpoint for all external interfaces of the replica. It ensures authorization, authentication, integrity, and general protocol compliance of incoming operations. Update operations that pass the Reference Monitor can therefore be handled like non-byzantine, i.e., correct operations. A serious attack could be based on non-unique event identifiers. However, unique event identifiers can be ensured in a byzantine environment by generating event identifiers from the event data using a collision-resistant hash function. This way, Reference Monitors can verify whether an event identifier is valid by recomputing the hash themselves. To prevent the injection of unauthorized update operations, impersonation needs to be prevented as well. This can be achieved by means of asymmetric encryption, i.e., by cryptographically signing update operations and a Public Key Infrastructure that is trusted by all correct replicas. Signatures also ensure integrity of update operations, so that update operations that are not directly received from the source replica cannot be altered unobtrusively. Therefore, authenticated update operations allow us to drop the No Creation property of the broadcast abstraction, as the Reference Monitor can now identify forged or tampered update operations itself. The creation of operations that are not protocol compliant, such as events with non-existing or non-(con)current extremities as parents, might incur load on performance, but does not threaten the correct operation of the MEG.

Attacks on the Reliable Broadcast Abstraction. Attacks on the Reliable Broadcast abstraction may lead to correct replicas that receive different operations, potentially causing permanent divergence in replica states. While fail-silent-arbitrary Reliable Broadcast algorithms exist (cf. [3, p. 121]), they are generally difficult to scale to many replicas, as communication complexity increases in the number of replicas. However, we do not require all of their properties due to the commutative and conflict-free nature of the MEG. Using asymmetric encryption, the No Creation property is not required and the broadcast abstraction is left to provide Validity and Agreement. As Validity is only concerned with correct sending replicas, faulty replicas can mainly attack Agreement by performing equivocation, i.e. broadcasting different update operations to different replica subsets, or not broadcasting an update operation to all replicas [14]. We show that equivocation, a costly problem in fail-arbitrary Reliable Broadcast algorithms, is not an issue for the Matrix Event Graph due to its distinct structure. We recall

that for Agreement, an operation that is received by some correct replica eventually has to be received by every correct replica.

Under the assumption that malicious replicas have no means to fabricate a hash collision, they can only send operations with different event identifiers when trying to create inconsistencies. However, due to the conflict-free nature of an operation-based CRDT, both operations can be received and processed by correct replicas. A byzantine replica that performs equivocation can therefore be modeled as two replicas that crash while sending independent update operations. Therefore, the broadcast abstraction only has to ensure that eventually, *any* operation received at some correct replica will be received at every correct replica.

In Matrix, Validity is provided since source replicas immediately apply update operations to their local state. However, with respect to Agreement, Matrix replicas use a ‘best-effort broadcast’ that is implemented via unicast transmissions to all replicas. This alone does not provide Agreement even in fail-silent systems without byzantine attackers, as a failing replica could only provide a limited number of correct replicas with the update operation. To mitigate this issue, Matrix uses a backfilling mechanism which allows replicas to specifically request missing operations from other replicas. It is used when a replica receives an update operation for which the parents are not part of the replica state. With this mechanism, Matrix achieves Agreement under the assumption of constant MEG progress, i.e., a never-ending stream of (arbitrary low-frequency) new update operations from other replicas. However, if / for as long as the progress come to a halt, Agreement, and thus Eventual Delivery, is violated⁷.

Therefore, Matrix does only provide Agreement and thereby SEC under the assumption of constant progress. One could now replace the best-effort broadcast with a gossip-based broadcast protocol that is scalable and robust, as suggested in [9]. While this alone is not sufficient to ensure Agreement without constant progress, the efficient gossip-based broadcast could be used by replicas to periodically broadcast their current set of forward extremities to all other replicas, which then could trigger backfilling. This addition would guarantee probabilistic Agreement, and therefore SEC for the MEG implementation of Matrix.

7. Scalability: Width of the MEG over Time

In this section, we study the evolution of the width of the MEG over time. While we verified our results with Monte-Carlo simulations, we decided to go for an analytical approach to deliver a precise mathematical

⁷In the Matrix reference replica implementation Synapse, this issue has been raised in the developer community [24]. Correct replicas will now take note of unreachable homeservers and retry synchronization once they become available eventually [25]. Faulty senders still require constant progress.

problem definition and treatment. In Sections 5 and 6, we assumed that *all* forward extremities known to a replica are used as parents for new vertices created by the replica. In this case, the number of forward extremities is reduced as much as possible whenever a new vertex is created. However, as noted in Section 2, honest replicas can experience a high number of forward extremities after a partition, and malicious replicas could deliberately create events with a high number of parents. This is problematic from a performance perspective because checks, particularly of the Reference Monitor, are resource intensive, especially when old parts of the MEG are referenced, but are needed for every parent [8]. Thus, for reasons of performance, the number of parents of a new vertex is restricted to a finite value d in practice. If there are more than d forward extremities, a replica selects a random subset of parents of size d for the new vertex. In this section, we provide evidence that the width of the MEG still converges⁸ to the the number k of participating replica times a small factor when all k replica repeatedly and concurrently add a new vertex.

We model the evolution of the width of the MEG as follows. We assume that vertices are added in rounds. A round consists of two steps: First, each of the k replicas concurrently adds a new forward extremity and thereby ‘eliminates’ d forward extremities which are used as parents. Second, all replicas synchronize their new extremities and reach a consistent state. The overall number of eliminated extremities depends on the amount of *overlap* between the parent choices of different replicas. As we are interested in scaling k while keeping d low, we assume $k > d$. As forward extremities cannot be eliminated effectively if a new forward extremity has only one parent, we assume $d > 1$. The model also accepts an arbitrarily high number of forward extremities u_0 as starting condition. We analyze the sequence of number of forward extremities u_i by a mean value analysis.

Please note that this model maximizes uncoordinated concurrency in Step 1 and, thus, models a worst case scenario: More new vertices per replica in Step 1, i.e., a higher frequency of updates by clients or prolonged periods of network partition, would eliminate more than d overlap-free forward extremities, but not add additional ones. Also, if replicas would be aware of the eliminations of other replicas, their forward extremity choices could be done more overlap-free.

7.1. Stochastic Process

We represent the concurrent updates in Step 1 of each round as a stochastic urn model. The initial number of forward extremities u is described by u initial red balls, while the number of newly linked parent vertices d is

⁸Please note that when we discuss convergence in this section, convergence is related to the number of forward extremities. In the previous CRDT-related section, convergence is related to propagation of states.

the number of balls taken out during a drawing by a replica. The update generator execution of the k replicas lead to the conduction of k independent drawings that can be modeled by sequential drawings with the use of black balls: the balls drawn by a replica are replaced by black balls and put back to the urn. Therefore, after k replicas have performed Step 1, the black balls indicate the number of selected parent vertices. After each round, the black balls are replaced by red ones again and the next round starts with the current number of red balls.

We let the random variable $R_{d,k}(u)$ denote the total number of removed forward extremities, while $u - R_{d,k}(u)$ denotes the number of forward extremities that ‘survived’ for the subsequent urn experiment. With this urn experiment, we build a stochastic process for the behavior of the number of forward extremities. We derive the expectation and the variance of $R_{d,k}(u)$, and we provide a recursion formula for the distribution of $R_{d,k}(u)$. We discuss the implications on MEGs in Subsection 7.3.

Let the random variable U_n describe the number of balls in the urn after $n \in \mathbb{N}_0$ rounds. Let u_0 be the initial number of balls in the urn, then $U_0 = u_0$ and $U_{n+1} = U_n + k - R_{d,k}(U_n)$. As $(U_n)_{n \in \mathbb{N}_0}$ is a sequence of random variables, it is a stochastic process (cf. e.g. [6]). We are interested in whether convergence can be *expected*, and, if yes, how fast convergence is reached. The process is a spatially inhomogeneous random walk, specifically a time-homogeneous Markov chain (cf. e.g. [15]) with state space $M_U = \mathbb{N}^+$:

$$\forall n \in \mathbb{N}_0 \quad \forall u_0, \dots, u_{n+1} \in M_U :$$

$$\begin{aligned} \mathbb{P}(U_{n+1} = u_{n+1} | U_0 = u_0, \dots, U_{n-1} = u_{n-1}, U_n = u_n) \\ = \mathbb{P}(U_{n+1} | U_n = u_n) \Rightarrow \text{memorylessness} \end{aligned}$$

with transition matrix: $P_{i,j} = \mathbb{P}(U_n = j | U_{n-1} = i) = \mathbb{P}(R_{d,k}(i) = k - (j - i))$ and transition probability: $\forall n \in \mathbb{N}_0 \forall l, m \in M_U : \mathbb{P}(U_n = j | U_{n-1} = i) = \mathbb{P}(U_1 = j | U_0 = i)$. Thus, the transitions are independent of n and the process is time-homogeneous.

A positive recurrent, aperiodic and irreducible Markov chain has a stationary distribution, i.e., a fixed point of the transition function in which the probabilities for the next state do not change with state transitions.

If we assume $u_0 \in [0, k - 1]$, then $u_1 > k$, as no more than u_0 balls can be drawn, but k balls get added. Therefore, states $[0, k - 1]$ are transient, and one can remove them from the chain. The remaining states are irreducible and aperiodic: As the next state increment in one round is in $[k - k \cdot d, k - d]$, every other state can be reached in a finite number of iterations. However, it is unclear whether the states are transient, i.e., visited only once, or positively recurrent, i.e., have a finite expected time until they are visited repeatedly. This represents an open problem and is left for future work.

7.2. Properties of Random Variable $R_{d,k}(u)$

As stated before, let $R_{d,k}(u)$ denote the total number of red balls that showed up in a single round of k independent drawings of size d from an urn of size u . Initially,

the urn contains only red balls ($r = u$) and no black balls ($b = 0$). A *drawing* means taking d balls from the urn at random, where $d < u$. The drawing ends by replacing each red ball with a black ball and then returning all d balls back into the urn.

We now provide the expectation (a) and the variance (b) of $R_{d,k}(u)$, and a recursion formula (c) for the distribution of $R_{d,k}(u)$. For the proof, see Appendix A.

Theorem 1. *For the random variable $R_{d,k}(u)$, we have:*

$$a) \quad \mathbb{E}(R_{d,k}(u)) = d \cdot \frac{1 - p^k}{1 - p}, \quad k \geq 1,$$

where

$$p = \frac{u - d}{u} \quad (1)$$

is the retention probability.

b)

$$\begin{aligned} \mathbb{V}(R_{d,k}(u)) = & \frac{vud}{1-p} \left(\frac{1-w^{k-1}}{1-w} - p^{k-1} \cdot \frac{1-(w/p)^{k-1}}{1-w/p} \right) \\ & - \frac{vd^2}{(1-p)^2} \left(\frac{1-w^{k-1}}{1-w} - 2p^{k-1} \frac{1-(w/p)^{k-1}}{1-w/p} \right. \\ & \left. + p^{2(k-1)} \frac{1-(w/p^2)^{k-1}}{1-w/p^2} \right), \end{aligned}$$

where

$$v = \frac{d(u-d)}{u^2(u-1)}, \quad w = \frac{(u-d)(u-d-1)}{u(u-1)}. \quad (2)$$

c) If $k \geq 2$ then

$$\mathbb{P}(R_{d,k}(u) = j) = \sum_{\ell=0}^d \frac{\binom{u-(j-\ell)}{\ell} \binom{j-\ell}{d-\ell}}{\binom{u}{d}} \cdot \mathbb{P}(R_{d,k-1}(u) = j-\ell).$$

7.3. Implications for the MEG and Conjecture

The formula for the expectation of $R_{d,k}(u)$ allows for statements on the expected convergence behavior of the MEG in the presence of concurrent updates by different replicas. In addition, the formula for the variance of $R_{d,k}(u)$ shows the deviation from expected convergent behavior. For Figure 4, we use these formulas to calculate the expected development and deviation of forward extremities U_n over the number of rounds for varying k but fixed d . To plot the calculations, we put different realizations of U_n against the expected value of U_{n+1} , via $\mathbb{E}(U_{n+1}) = U_n + k - \mathbb{E}(R_{d,k}(U_n))$. The dashed line is $U_{n+1} = U_n$, so its intersection with the colored lines mark their fixed points. In the area below the dashed line, $\mathbb{E}(U_{n+1}) < U_n$, the urn contents are expected to decrease, in accordance with the plotted standard deviation. The change from linear to constant curves (for decreasing U_n , i.e. from right to left) show the switch from likely overlap-free choices to overlapping choices, which decrease the urn contents less. It shows that for any plotted realization of U_n , we either expect a decreasing urn value (below the dashed line), or a transition to the fixed point. Therefore, the plotted configurations

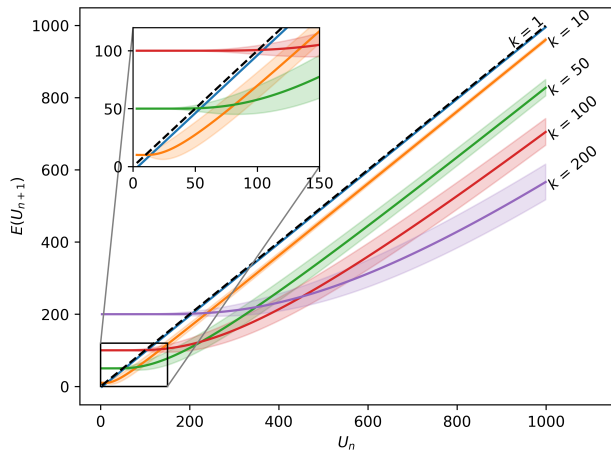


Figure 4.: Expectation for the next urn content $\mathbb{E}(U_{n+1})$ for different realizations of U_n , $d = 5$, and varying k . Points below the dashed line of $U_n = \mathbb{E}(U_{n+1})$ mean that the urn content is expected to decrease, points above mean that an increase is expected. For visibility, the plotted standard deviation is increased by the factor 5. Please note that when the curves are followed from right to left, they change from a linear slope to a constant value close to k .

show convergence. In addition, the variance is very low. We observe that the convergence of the width of the graph appears to be almost optimal, i.e., the fixed point is near k .

Synapse, the reference implementation of a Matrix replica, recently activated a feature to force the depletion of forward extremities by sending empty ‘dummy’ events using the same parent selection rules as regular events⁹ with $d = 10$, as soon as there are more than 10 forward extremities present [11]. This fact allows to take advantage of the convergence in periods of missing updates, and brings reality closer to our model.

To gain insights into the influence of d , we use the expectation of U_n via $\mathbb{E}(U_{n+1}) = \mathbb{E}(U_n) + k - \mathbb{E}(R_{d,k}(U_n))$, and calculate the number of rounds n until $\mathbb{E}(U_n) - \mathbb{E}(U_{n+1}) < 1$. This is equivalent to the number of rounds after which $\mathbb{E}(R_{d,k}(\mathbb{E}(U_n))) \geq k$ holds, i.e., the number of rounds after which we expect to eliminate a number of forward extremities in Step 1 that is less than or equal to the number of forward extremities that we add in Step 2. Fig. 5 shows that, while the number of rounds until convergence is reached directly depends on the choice of d , there are diminishing returns. The highest gain in time until convergence is between $d = 2$ and $d = 3$, while there is much less difference between $d = 6$ and $d = 10$. With optimal choice of forward extremities, i.e., $u \gg k \cdot d$, convergence speed is nearly $k \cdot (1 - d)$, and therefore the number of rounds until convergence is nearly proportional to $\frac{1}{1-d}$. Synapse employs $d = 5$ with

⁹Note that Synapse actually takes 5 random forward extremities and 5 of the newest forward extremities, which are not independent between replicas.

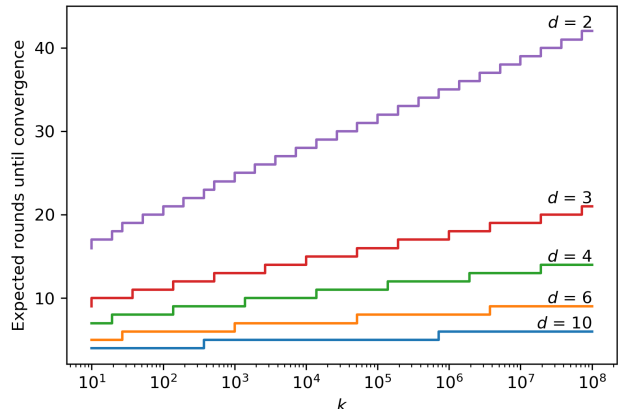


Figure 5.: Expected number of rounds until convergence for varying d and k , starting at $u_0 = 100 \cdot k$. While convergence speed increases with d , the returns in the number of rounds to reach convergence diminish.

$k \lesssim 10^3$, which we can confirm as a good compromise in convergence speed performance using our formulas in Figs. 4 and 5.

With small u , bad choices, i.e., overlapping choices for parents are made, but because u is small, they don’t harm convergence permanently. With large u , the probability for overlapping choices grows smaller and smaller, and convergence speed is linear.

We therefore conjecture that regardless of the exact choice of k and d , the process converges for any start value u to a stationary value near k in a finite number of rounds. The derived properties of $R_{d,k}(u)$ are important building blocks to eventually prove this conjecture. The convergence speed depends on the choice of d , but values larger than 3 are subject of diminishing returns.

In practice, this means that if the conjecture holds, the MEG possesses a self-stabilization property [2] in the sense that if transient faults lead to a high number of forward extremities (a high u), a correct system converges to a stable number of forward extremities near k in a finite number of rounds, and remains stable as if the fault had never occurred.

8. Conclusion

In this paper, we extracted and abstracted the replicated data type employed by Matrix, and proved that it represents a Conflict-Free Replicated Data Type. Therefore, the Matrix Event Graph provides Strong Eventual Consistency, a fact that in particular indicates that all correct replicas that applied the same set of updates are in equivalent state — immediately and without any further agreement procedure. This proof gives fundamental insights into why the Matrix system shows good resilience and scalability in the number of replicas in practice. It therefore makes the underlying replicated data type an attractive candidate as a basis for other decentralized applications. In addition, we analyzed the challenges for systems with byzantine actors and

showed that the properties of the Matrix Event Graph facilitate a byzantine-tolerant design, especially due to equivocation tolerance. However, design and analysis of an appropriate underlying broadcast protocol with the identified properties remain topics for future research. Furthermore, we formalized and studied the evolution of the width of the graph as a spatially inhomogeneous random walk. Our observations let us conjecture that the width of the graph always converges independently of the specific system parameters, and does so fast.

In summary, we believe that the Matrix system and similar systems are highly relevant in real-world scenarios, and that their scientific understanding is of utmost importance. We hope that our results advance understanding as well as proper real-world setup of those systems, and can serve as a basis for further research.

A. Proof of Properties of $R_{d,k}(u)$

For a series of drawings $R_{d,k}(u)$, we write Z_k for the number of red balls that show up in the k th drawing, so that $R_{d,k}(u) = Z_1 + \dots + Z_k$.

a) In what follows, let $k \geq 2$. Under the condition $R_{d,k-1}(u) = r$, the urn contains $u-r$ red and r black balls. Thus, the conditional distribution of Z_k given $R_{k-1} = r$ is the hypergeometric distribution $\text{Hyp}(d, u-r, r)$, which implies

$$\mathbb{E}(Z_k | R_{d,k-1}(u) = r) = d \cdot \frac{u-r}{u}.$$

Since $R_{d,k}(u) = R_{d,k-1}(u) + Z_k$, we have $\mathbb{E}(R_{d,k}(u)) = \mathbb{E}(R_{d,k-1}(u)) + \mathbb{E}(Z_k)$. Moreover,

$$\begin{aligned} \mathbb{E}(Z_k) &= \mathbb{E}[\mathbb{E}(Z_k | R_{d,k-1}(u))] = \mathbb{E}\left[d \cdot \frac{u - Z_{k-1}}{u}\right] \\ &= d - \frac{d}{u} \cdot \mathbb{E}(R_{d,k-1}). \end{aligned}$$

It follows that

$$\begin{aligned} \mathbb{E}(R_{d,k}(u)) &= \mathbb{E}(R_{d,k-1}(u)) + d - \frac{d}{u} \cdot \mathbb{E}(R_{d,k-1}(u)) \\ &= d + p \mathbb{E}(R_{d,k-1}(u)). \end{aligned}$$

Together with $\mathbb{E}(R_{d,1}(u)) = d$, we now obtain by induction over k

$$\mathbb{E}(R_{d,k}(u)) = d \sum_{j=0}^{k-1} p^j = d \cdot \frac{1-p^k}{1-p},$$

as was to be shown. Notice that

$$\lim_{k \rightarrow \infty} \mathbb{E}(R_{d,k}(u)) = \frac{d}{1-p} = u.$$

This result is not surprising, since in the long run each of the red balls will have shown up.

b) The proof uses the general fact that, for random variables X and Y , the variance of X can be calculated according to the formula $\mathbb{V}(X) = \mathbb{E}[\mathbb{V}(X|Y)] + \mathbb{V}(\mathbb{E}[X|Y])$, i.e., the variance of X is the sum of the expectation of the conditional variance of X given Y and the variance of the conditional expectation of X given Y . In our

case, we put $X = R_{d,k}(u)$ and $Y = Z_{k-1}$, where $k \geq 2$, and obtain

$$\begin{aligned} \mathbb{V}(R_{d,k}(u)) &= \mathbb{V}(R_{d,k-1}(u) + Z_k) \\ &= \mathbb{E}[\mathbb{V}(R_{d,k-1}(u) + Z_k | R_{d,k-1}(u))] \\ &\quad + \mathbb{V}(\mathbb{E}[R_{d,k-1}(u) + Z_k | R_{d,k-1}(u)]). \end{aligned} \quad (3)$$

Since $\mathbb{V}(R_{d,k-1}(u) + Z_k | R_{d,k-1}(u)) = \mathbb{V}(Z_k | R_{d,k-1}(u))$ and the conditional distribution of Z_k given $R_{d,k-1}(u)$ is the hypergeometric distribution $\text{Hyp}(d, u - R_{d,k-1}(u), R_{d,k-1}(u))$, it follows that

$$\begin{aligned} \mathbb{V}(Z_k | R_{d,k-1}(u)) &= d \cdot \frac{u - R_{d,k-1}(u)}{u} \\ &\quad \cdot \left(1 - \frac{u - R_{d,k-1}(u)}{u}\right) \left(1 - \frac{d-1}{u-1}\right) \\ &= \frac{d}{u^2} \left(1 - \frac{d-1}{u-1}\right) \\ &\quad \cdot (u - R_{d,k-1}(u)) R_{d,k-1}(u). \end{aligned} \quad (4)$$

Moreover, we have

$$\begin{aligned} \mathbb{E}[R_{d,k-1}(u) + Z_k | R_{d,k-1}(u)] &= R_{d,k-1}(u) + \mathbb{E}[Z_k | R_{d,k-1}(u)] \\ &= Z_{k-1} + d \cdot \frac{u - R_{d,k-1}(u)}{u} \\ &= d + \left(1 - \frac{d}{u}\right) R_{d,k-1}(u). \end{aligned}$$

Therefore, the second summand figuring in (3) equals

$$\left(1 - \frac{d}{u}\right)^2 \mathbb{V}(R_{d,k-1}(u)).$$

Since $\mathbb{V}(R_{d,k-1}(u)) = \mathbb{E}[R_{d,k-1}^2(u)] - (\mathbb{E}R_{d,k-1}(u))^2$, (4) yields

$$\begin{aligned} \mathbb{E}[\mathbb{V}(Z_k | R_{d,k-1}(u))] &= \frac{d}{u^2} \left(1 - \frac{d-1}{u-1}\right) \\ &\quad \cdot (u \mathbb{E}(R_{d,k-1}(u)) \\ &\quad - \mathbb{V}(R_{d,k-1}(u)) - (\mathbb{E}R_{d,k-1}(u))^2). \end{aligned}$$

We thus obtain the recursion formula

$$\begin{aligned} \mathbb{V}(R_{d,k}(u)) &= v \cdot \mathbb{V}(R_{d,k-1}(u)) \\ &\quad + \frac{d}{u^2} \cdot \left(1 - \frac{d-1}{u-1}\right) \\ &\quad \cdot \left(u \mathbb{E}(R_{d,k-1}(u)) - (\mathbb{E}R_{d,k-1}(u))^2\right) \end{aligned}$$

with v given in (2), from which the result follows by straightforward calculations. Notice that $\mathbb{V}(R_{d,1}(u)) = 0$ ($R_{d,1}(u)$ is the constant d), and that $\lim_{k \rightarrow \infty} \mathbb{V}(R_{d,k}(u)) = 0$. The latter convergence is clear from the fact that, in the long run, all red balls will have been drawn.

c) The result follows from the fact that the event $\{R_{d,k} = j\}$ is the union of the pairwise disjoint events $\{R_{d,k-1}(u) = j - \ell, Z_k = \ell\}$, $\ell = 0, 1, \dots, d$, and the fact that the conditional distribution of $R_{d,k}(u) (= R_{d,k-1}(u) + Z_k)$ given $R_{d,k-1}(u) = j - \ell$ is the hypergeometric distribution $\text{Hyp}(d, u - (j - \ell), j - \ell)$.

B. The hypergeometric distribution

Suppose an urn contains b black and w white balls. If m balls are drawn completely at random without replacement, then the number X of black balls drawn has the hypergeometric distribution $\text{Hyp}(m, b, w)$, i.e., we have

$$\mathbb{P}(X = j) = \frac{\binom{b}{j} \binom{w}{m-j}}{\binom{b+w}{m}}, \quad j = 0, 1, \dots, m,$$

where we put $\binom{s}{\ell} := 0$ if $s < \ell$. Expectation and variance of X are given by

$$\begin{aligned} \mathbb{E}(X) &= m \cdot \frac{b}{b+w}, \\ \mathbb{V}(X) &= m \cdot \frac{b}{b+w} \cdot \left(1 - \frac{b}{b+w}\right) \left(1 - \frac{m-1}{b+w-1}\right), \end{aligned}$$

respectively.

Acknowledgment

We thank the Matrix developers for their ingenious system design, and Alexander Marsteller for many hours of differential equation analysis.

References

- [1] A. Auvolat. “Making Federated Networks More Distributed”. In: *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. 2019, pp. 383–384.
- [2] Baruch Awerbuch et al. “Self-stabilization by local checking and global reset”. In: *Distributed Algorithms*. Ed. by Gerard Tel and Paul Vitányi. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 326–339. ISBN: 978-3-540-48799-9.
- [3] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. 2nd. Springer Publishing Company, Incorporated, 2011. ISBN: 9783642152597. DOI: 10.1007/978-3-642-15260-3.
- [4] Kevin De Porre et al. “A generic replicated data type for strong eventual consistency”. In: *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC 2019* (2019). DOI: 10.1145/3301419.3323974.
- [5] The Matrix.org Foundation. *Matrix Specification: Architecture*. 2019. URL: <https://matrix.org/docs/spec/#architecture>.
- [6] Robert G Gallager. *Discrete stochastic processes*. Vol. 321. Springer Science & Business Media, 2012.
- [7] Seth Gilbert and Nancy Lynch. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services”. In: *SIGACT News* 33.2 (2002-06), pp. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601. URL: <https://doi.org/10.1145/564585.564601>.
- [8] Matthew Hodgson. *Forward extremities accumulate and lead to poor performance*. 2017. URL: <https://github.com/matrix-org/synapse/issues/1760> (visited on 2020-05-31).
- [9] Florian Jacob, Jan Grashöfer, and Hannes Hartenstein. “A Glimpse of the Matrix: Scalability issues of a new message-oriented data synchronization middleware”. In: *Proc. ACM 20th Int. Middleware Conference Demos and Posters*. 2019, pp. 5–6.
- [10] Florian Jacob et al. “Matrix Decomposition: Analysis of an Access Control Approach on Transaction-Based DAGs without Finality”. In: *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies. SACMAT ’20*. 2020, pp. 81–92. DOI: 10.1145/3381991.3395399.
- [11] Erik Johnston. *Synapse: Add experimental option to reduce extremities*. 2019-06-19. URL: <https://github.com/matrix-org/synapse/pull/5480> (visited on 2020-06-01).
- [12] Martin Kleppmann and Alastair R Beresford. “A conflict-free replicated JSON datatype”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017), pp. 2733–2746.
- [13] Leslie Lamport. “Time, Clocks, and the Ordering of Events in a Distributed System”. In: *Communications of the ACM* 21.7 (1978), pp. 558–565. ISSN: 15577317. DOI: 10.1145/359545.359563.
- [14] Mads Frederik Madsen and Søren Debois. “On the Subject of Non-Equivocation: Defining Non-Equivocation in Synchronous Agreement Systems”. In: *Proceedings of the 39th Symposium on Principles of Distributed Computing. PODC ’20*. Virtual Event, Italy: Association for Computing Machinery, 2020, pp. 159–168. ISBN: 9781450375825. DOI: 10.1145/3382734.3405731. URL: <https://doi.org/10.1145/3382734.3405731>.
- [15] Michael Mitzenmacher and Eli Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.
- [16] John C. Platt, Nello Cristianini, and John Shawe-Taylor. “Large Margin DAGs for Multiclass Classification”. In: *Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 547–553.
- [17] Nuno Preguiça. “Conflict-free Replicated Data Types: An Overview”. In: *arXiv preprint arXiv:1806.10254* (2018). URL: <http://arxiv.org/abs/1806.10254>.
- [18] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. “Conflict-Free Replicated Data Types CRDTs”. In: *Encyclopedia of Big Data Technologies* (2019), pp. 491–500. DOI: 10.1007/978-3-319-77525-8_185.

- [19] Mayank Raikwar, Danilo Gligoroski, and Goran Velinov. “Trends in Development of Databases and Blockchain”. In: *arXiv preprint arXiv:2003.05687* (2020).
- [20] Yasushi Saito and Marc Shapiro. “Optimistic replication”. In: *ACM Computing Surveys* 37.1 (2005), pp. 42–81. ISSN: 03600300. DOI: 10.1145/1057977.1057980.
- [21] Marc Shapiro et al. *A comprehensive study of Convergent and Commutative Replicated Data Types*. Tech. rep. RR-7506. 2011. URL: <https://hal.inria.fr/inria-00555588>.
- [22] Marc Shapiro et al. “Conflict-free replicated data types”. In: *Symposium on Self-Stabilizing Systems*. Springer. 2011, pp. 386–400.
- [23] Marc Shapiro et al. “Conflict-free Replicated Data Types. Rapport de recherche 7686”. In: (2011). DOI: 10.1007/978-3-319-77525-8_185.
- [24] Richard van der Hoff. *Homeservers don’t catch up with missed traffic until someone sends another event*. 2017. URL: <https://github.com/matrix-org/synapse/issues/2528> (visited on 2020-04-02).
- [25] Olivier ‘reivilibre’ Wilkinson. *Catch-up after Federation Outage*. 2020. URL: <https://github.com/matrix-org/synapse/pull/8272> (visited on 2020-10-30).
- [26] K. Zhang and H. Jacobsen. “Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 1337–1346.
- [27] Wenbing Zhao et al. “Byzantine fault tolerance for collaborative editing with commutative operations”. In: *IEEE International Conference on Electro Information Technology (2016)*, pp. 246–251. ISSN: 21540373. DOI: 10.1109/EIT.2016.7535248.