Probabilistic Causal Contexts for Scalable CRDTs

Pedro Henrique Fernandes
ProDEI, Universidade do Porto & INESC TEC
Porto, Portugal
up202201271@edu.fe.up.pt

Carlos Baquero
Universidade do Porto & INESC TEC
Porto, Portugal
cbm@fe.up.pt

Abstract

Conflict-free Replicated Data Types (CRDTs) are useful to allow a distributed system to operate on data even when partitions occur, and thus preserve operational availability. Most CRDTs need to track whether data evolved concurrently at different nodes and needs to be reconciled; this requires storing causality metadata that is proportional to the number of nodes. In this paper, we try to overcome this limitation by introducing a stochastic mechanism that is no longer linear on the number of nodes, but whose accuracy is now tied to how much divergence occurs between synchronizations. This provides a new tool that can be useful in deployments with many anonymous nodes and frequent synchronizations. However, there is an underlying trade-off with classic deterministic solutions, since the approach is now probabilistic and the accuracy depends on the configurable metadata space size.

CCS Concepts: • **Networks**; • **Computer systems organization** → **Distributed architectures**; **Availability**; **Redundancy**; **Reliability**; Fault-tolerant network topologies; • **Mathematics of computing** → **Probabilistic algorithms**; **Probabilistic representations**; • **Information systems** → **Storage architectures**; **Distributed storage**; • **Computing methodologies** → Simulation evaluation;

Keywords: Conflict-free Replicated Data Types (CRDTs), Bloom filters, eventual consistency

1 Introduction

The CAP theorem [3, 7] states that only two of the three properties of shared-data systems can be achieved simultaneously - data consistency, system availability, and tolerance to network partitions. In view of the inevitability of the occurrence of network partitions and the need to counter them, particularly in large-scale, geographically distributed systems, only one other property remains - either availability or consistency.

In order to achieve high availability, the model of consistency in a system must be relaxed. One of such models is eventual consistency [12, 13]. Under the conditions defined by this model, replicas are able to serve requests immediately, whether they mutate the state at the receiving replica or not. Replies are based on local state at the time of service and happen with no need for synchronizations with remote replicas. The effects of mutating operations can be shared asynchronously, at some point in the future, allowing replicas to converge to the same state. As such, synchronizations among replicas allow clients to operate on the same data objects by issuing requests to different replicas. The frequency of synchronization has direct impact on how stale local states can get, thus influencing the quality expected from replies. In spite of the possibility of replying with out of date information, many use cases allow for some flexibility, making eventual consistency a viable option.

Conflict-free Replicated Data Types (CRDTs) are data types intended for distributed use that employ the eventual consistency model, offering deterministic convergence of state, even in environments with concurrent events. Due to the fact that some operations are not commutative, such as the addition and removal of the same element from a set, these data types may include a conflict resolution mechanism, to deterministically decide on the resulting state of a replica. CRDTs can take the shape of registers, counters, sets, maps, sequences, among others - each containing an appropriate conflict resolution strategy [9]. Among the different categories of CRDTs there are Causal CRDTs, which depend on knowledge of known events to provide adequate semantics to some data types.

Causal contexts, the structures at the heart of causal CRDTs, record events known by their respective replicas. In spite of being widely applicable and providing excellent support for compaction of information on known events, existing causal contexts hold an amount of data that grows linearly in relation to the number of replicas in the system. Given the fact that this property limits scalability, our work intends to explore the use of probabilistic structures as replacements of the classic, deterministic, and error-free causal contexts.

Additional background knowledge on CRDTs and causal contexts appears in Section 2. Section 3 introduces probabilistic structures for set representation.

Furthermore, due to the characteristics of the probabilistic structures considered, event identifiers may no longer need to be formed in the same way as they did previously, potentially reducing their overall size by doing away with the need to identify their origin replica. This anonymity, in addition to smaller states for causal CRDTs, provides the additional benefit of removing the requirement for management of identities and membership in the system. This may be especially useful in networks with some degree of dynamicity. All of these are objectives of the newly proposed design, presented in Section 4.

The evaluation of the conceived solution, in Section 5, aims to inform the reader of the potential of such an approach

regarding memory consumption and frequency of errors, so as to guide potential further implementations or evolutions of this work.

Lastly, Section 6 contains concluding remarks and suggestions of future work that might improve the results or expand our work.

The main contributions of our work are the following:

- Recognition of the scalability limiting factors in existing solutions and assessment of alternative designs based on probabilistic structures;
- The identification of hindrances accompanying the use of Age-Partitioned Bloom Filters (APBFs), that could apply to similar probabilistic structures, and conception of filter union strategies to provide additional robustness to the existing design in the context of the present work;
- The reimplementation of Age-Partitioned Bloom Filters in Rust, with the adequate adaptations [5]. This intends to offer the benefit of added memory safety and possible improvements in efficiency and memory management when compared with the original implementation;
- The analysis of the memory consumption and correctness of a concrete probabilistic CRDT, with the aim of determining potential use cases and informing future implementations under realistic conditions.

2 Why "Data Types"?

The words "Data Types" in the title refer to Conflict-free Replicated Data Types (CRDTs), presented and formally defined in [9, 10]. CRDTs are data types that provide deterministic results, even in the presence of concurrent executions. This determinism allows the data types to be conflict-free.

As their name suggests, CRDTs allow for replication of data. In particular, they employ optimistic replication, also known as *eventual consistency*.

Each node that holds an instance of a CRDT accepts updates without synchronization with remote instances. The data replicas are able to store is kept consistent with other replicas through asynchronous synchronization and incorporation of remote message data into the local state.

Although solutions guided by the principles of eventual consistency existed previously, their implementations were ad hoc and error-prone. In the presence of conflicting concurrent updates, these solutions could require a consensus and roll-back for arbitration.

The formalization of CRDTs came to define sufficient conditions for convergence, which is the mechanism that enables eventual consistency. These data types are capable of ensuring absence of conflicts by leveraging mathematical properties, such as inflation in a semi-lattice and commutativity. Due to this, CRDTs are guaranteed to converge, even in spite of failures.

Conflict-free Replicated Data Types are divided into two major distinct design classes - state-based and operation-based. State-based designs include one other major class - delta-state CRDTs. These avoid sharing the complete state. The work described in this article makes use of state-based designs.

2.1 State-based CRDTs

The state-based approach for replicated objects has every replica executing each update request, initially only modifying its own local state. Through synchronization, occasionally, each replica sends its local state to a set of other replicas, which merge the received state with their own, by using a *merge* function that deterministically reconciles the states.

The *merge* function, that aims to maintain convergence, is defined as a *join*: a least upper bound over a join-semillatice. Joins have some characteristics that make the state-based designs capable of handling scenarios such as the integration of duplicate messages or differently ordered deliveries at different replicas. Figure 1 presents these.

$$a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c \qquad \text{(associativity)}$$
$$a \sqcup b = b \sqcup a \qquad \text{(commutativity)}$$
$$a \sqcup a = a \qquad \text{(idempotency)}$$

Figure 1. Properties of *join*

Examples of such *join* operations can be seen in Figure 2. It shows a Set CRDT, with concurrent additions and removals of elements. *Joins* occur when nodes have to integrate remote states into their local copies.

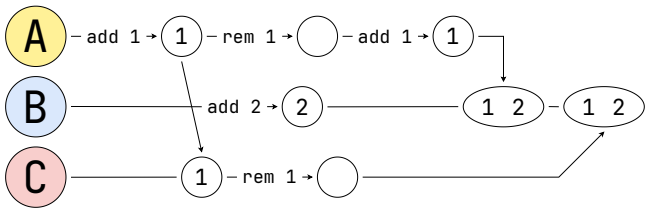

Figure 2. Example execution of an Add-Wins Set CRDT

Eventually, every update request and its effects will be registered at every other replica, which by the convergence guarantees, will make every replica have equal state. This claim assumes that the set of known updates is the same among all of the replicas.

CRDTs following this approach have the benefits of idempotency and lack of need for message ordering over the operation-based approach. These benefits are carried to the delta-state versions. Due to the fact that state can grow unbounded for some data types, synchronizing by sending the

full state to known replicas in the system may not be an ideal solution. Delta-state CRDTs aim to solve this issue.

2.2 Causal Context

Causal CRDTs, presented in [1], are a class of CRDT designs whose states are made up of a dot store and a causal context. The function of a dot store is to hold information specific to the data type, such as values and relevant metadata. The causal context, also known as the causal history or dot context, is used to remember what events are known by an instance of a distributed object.

More specifically, a causal context is an ever-growing set of event identifiers. As [1] states, causal CRDTs make use of globally unique identifiers to tag locally occurring events. These identifiers are assigned only to specific update events, such as, for instance, add(e) operations to insert elements into a set. This is done in order to track knowledge of the updates and their effects across the replicas of the distributed object.

The construction of these identifiers is usually done in a manner that aims to facilitate the compaction of a set of them. With this in mind, the strategy that is normally followed is appending the value of a locally unique, monotonically increasing counter to a globally unique replica identifier. This way, by incrementing the local counter after every assignment, a given replica i in the system is able to generate unique values, starting with the pair $(i, 1)$, followed by $(i, 2)$, successively repeating the process for any operation the design deems worthy of an identifier. Each one of these globally unique identifiers is also known as a dot.

2.2.1 Representation and Compaction.

By choosing an appropriate design for the dots, like the one that was previously mentioned, the lossless compaction of a causal context becomes possible. If that was not the case, the memory consumption of the set would grow linearly for every added dot. For a system that intends to run indefinitely, the size of the causal context could grow unbounded, which would eventually compromise the usefulness of causal CRDT designs.

Through synchronization with others, one instance is able to construct sequences of produced dots for each of the replicas in the system. When using an algorithm that guarantees causal consistency of the underlying CRDT, there is a single contiguous initial interval for every replica; one of such is the anti-entropy algorithm introduced in [1].

For each replica $i \in \mathbb{I}$ ($\mathbb{I}$ being the set of replica identifiers), the contiguous initial interval of known dots from any replica $j \in \mathbb{I}$ can be represented by a single value, given by $\max_j(c_i)$. This is possible because the causal consistency guarantee given by the aforementioned algorithm ensures that, from replica j, every value from the initial one up to the one given by $\max_j(c_i)$ is already known. To test the inclusion of a dot from j, with local sequence number n, in the causal context of i, the following condition is sufficient:

$$1 \leq n \leq \max_j(c_i) \implies (j, n) \in c_i \tag{1}$$

The $\max_i(c)$ function, defined in Figure 3, returns the greatest value in the causal context c from the sequence generated by replica i. Along with it, there is a definition for function $\text{next}_i(c)$, which is useful for producing dots in the previously described fashion. The dots in the following definition of causal context are pairs of replica IDs and integers ($\mathbb{I} \times \mathbb{N}$) that serve as globally unique identifiers, as presented in the introduction to this Section, 2.2.

$$\begin{aligned} \text{CausalContext} &= \mathcal{P}(\mathbb{I} \times \mathbb{N}) \\ \max_i(c) &= \max(\{n \mid (i, n) \in c\} \cup \{0\}) \\ \text{next}_i(c) &= (i, \max_i(c) + 1) \end{aligned}$$

Figure 3. Causal Context

The conclusions here stated mean that the causal context could be represented as a compact vector, similar to a version vector [8]. If replica IDs coincide with indices in the vector, with each replica being assigned a single index, the causal context becomes especially compact. Otherwise, a simple map $\mathbb{I} \hookrightarrow \mathbb{N}$ suffices.

If the algorithm that supports the causal CRDT design is one that does not guarantee causal consistency (such as the basic anti-entropy algorithm from [1]), the same compaction principle can be applied, albeit with a caveat. Without the guarantee of causal consistency, there is no assurance that all known dots from a given replica form a contiguous sequence. However, the initial sequence of known dots from a given replica can be encoded in a structure similar to the ones already described; the remaining straggler dots can be stored in a set. The size of the set of stragglers is not expected to be problematic, because as synchronizations occur, straggler dots eventually find their way into the compact representation, which means that the set is expected to be kept small.

An observation that can be made about this compaction scheme is that the size of the causal context grows linearly with the number of replicas. Furthermore, there is a reliance on replicas identifying themselves in the underlying algorithm. The solution described in this work does away with the need for these, while introducing a probabilistic dimension. The probabilistic dimension inevitably adds a number of errors, that ideally, should be kept small.

3 Why "Probabilistic"?

When in use in a CRDT design, the causal context is a set of dots at its core, even if it can be encoded in a structure that differs from a typical set implementation, as stated in 2.2.1.

The structures used in current designs represent the set of dots in a compact, deterministic, and therefore, error-free fashion. The property of determinism in current causal contexts can be observed in two ways:

- If a dot is indeed in the set, the structure will always report it as present, which means that there are no false negatives.
- Similarly and expectedly, if a dot is not in the set, the structure will report it as absent. Thus, there is an assurance of no false positives.

The compact, error-free solutions perfectly fit many of the common use cases for CRDTs. In spite of the frequent adequacy of existing methods and structures, some limiting factors can be found in their usage for specific use cases.

The major catalyst for our work was the fact that causal contexts in current designs make use of structures similar to version vectors [8]. One attribute of these is the storage of an amount of data that is linear with the number of replicas present in the system.

There is also a dependency on identifying replicas, which may not always be desirable. Usage of identification can be seen in the existing implementations of causal contexts, as evidenced by the anti-entropy algorithm in [1]. This is used to achieve a causally consistent delivery when synchronizing with delta-states. However, anonymity throws causally consistent delivery out of the picture in that same algorithm. This happens due to the fact that it becomes no longer possible to track which deltas have been delivered to which remote replicas, rendering impossible their error-free garbage collection from the deltas' buffer. Thus, by striving to achieve an anonymous design, some possibilities are lost, including the use of such an algorithm. Losses of this nature limit some anonymous designs to more basic algorithms, such as the basic anti-entropy algorithm from [1].

Dynamic environments are another potential generator of limitations. When overlays are dynamic and often changing, replicas may have to frequently refresh their set of neighbours and manage the entries of their causal contexts. The need for entry management stems not only from failures and intentional departures, but also from the arrival of previously unknown replicas. When it happens, it is essential to create an entry for the newly observed remote replica, in order to track known dots that originated there. Entry management can also be useful for garbage collection, in order to remove stale entries when faced with scenarios where a replica crashes or goes away for indefinite time. Depending on the underlying structure used to encode the causal context and the frequency of changes in the overlay, entry management may be more or less of a performance concern, as a result of resizing, for instance.

Dynamic environments are a known challenge to δ-CRDTs in particular, as briefly mentioned in [4]. Due to the fact that there may be no information about the last deltas acknowledged by a given new neighbour, emission of the full state may now be required more frequently. This need is observable in the above-mentioned anti-entropy algorithm in [1].

The solution described in this article aims to improve the existing solutions in scenarios where at least some of the limitations stated above are felt. The attempt makes use of a probabilistic structure to represent the set of dots, in place of deterministic ones. The use of a probabilistic structure spawns a number of errors, due to the inherent nature of stochastic methods. Nevertheless, if their occurrence can remain limited, they can allow the use of CRDT designs, namely ones that synchronize by sending state to neighbours, in scenarios where they would otherwise be discouraged or even impossible to use.

In the remainder of this Section we aim to provide context to the probabilistic representation of sets. We cover one of the two first structures of this kind, which came to be known as the Bloom Filter [2], and the variant of it that is used in the final design, the Age-Partitioned Bloom Filter [11].

3.1 Bloom Filters

Bloom Filters were introduced by Burton H. Bloom [2] in 1970 for the purpose of representing sets probabilistically. They utilize a hash area consisting of N individual addressable bits, that can be seen as 1-bit cells. The initial state of the bits has them all set to 0.

On insertion, each element is mapped by each of a set of hash functions to an address in the hash area. From the resulting set of addresses, every corresponding bit is then set to 1.

To evaluate the membership of an input element, an address is calculated by each of the hash functions. The element is said to be in the set if all of the bits addressed are set to 1.

As is the case with the deterministic, error-free methods, inserted elements will always be identified as present by queries, meaning that there are no false negatives. In contrast, if an element has not been inserted in these structures, it can still be reported as present, generating a false positive. False positives can occur if all the bits in the addresses calculated for an absent element are set to 1. This can happen if the addressed bits were previously set by insertions of other elements.

Given the fact that the size of the hash area is limited and since queries rely on the existence of bits set to 0 to identify absent elements, there is a point of saturation beyond which too many false positives occur. Burton H. Bloom identified this point of saturation, considering it to be reached when "half the bits in the hash field are 1 and half are 0".

Two main points influence saturation:

- The smaller the bit space, the faster it gets saturated.
- The more hash functions used, the more possible distinct addresses are generated to represent an element,

which improves distinction between elements and therefore decreases the occurrence of false positives. However, too many hash functions saturate the bit space at a higher rate and increase the likelihood of a false positive occurring, suggesting there is a balance to be found regarding the number of hash functions in use, and subsequently, the number of bits representing an element.

One key idea regarding filter construction is that a greater allowable error frequency enables the usage of a smaller area of memory, with the exact opposite being also true, indicating the existence of tunable parameters in implementations. These enable the creation of filters according to the needs of the applications that make use of them.

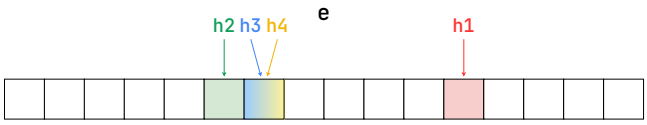

Figure 4. Insertion of element e in a Bloom Filter using 4 hash functions

3.2 Age-Partitioned Bloom Filters

Age-Partitioned Bloom Filters (APBFs) [11] differ from the original Bloom Filters by aiming to represent probabilistically a sliding window over a stream of elements, instead of a fixed set of elements. APBFs can be seen as a collection of Partitioned Bloom Filters, known as slices. Partitioned Bloom Filters do away with collisions of the initial approach by dividing the hash space into distinct segments, one for each hash function. This distinction can be seen in Figure 5.

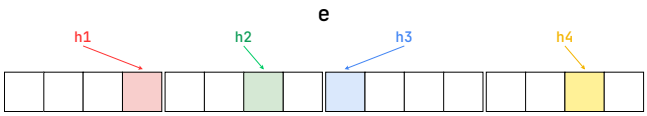

Figure 5. Insertion of element e in a Partitioned Bloom Filter using 4 hash functions

APBFs are characterized by three structural parameters - k, l and m. An APBF is made up of a sequence of $k + l$ equally sized slices of m bits each, from s_0 to s_{k+l-1}. The first k slices, from s_0 to s_{k-1}, are the ones active for insertions. The remaining l slices, from s_k to s_{k+l-1}, are relevant for storage capacity, allowing the storage of older batches. Over time, as slices reach their saturation points, they are shifted within the filter, promoting the aging of elements. Slices hold batches of elements named generations and, upon a generation change, the oldest slice is removed and a new one is added. Consequently, this fact allows making room for new elements and forgetting the oldest ones, maintaining the size of the filter. These characteristics can be observed in the diagram of Figure 6.

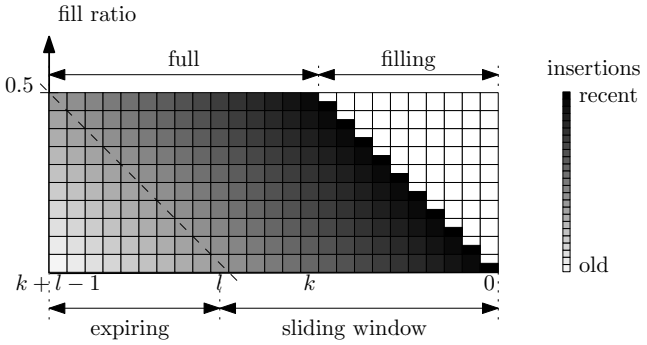

Figure 6. Fill ratios of the slices of an APBF

APBFs natively support two operations - addition of elements to the set, add(e), and element presence queries, query(e). The add(e) operation sets bits at locations $s_i[h'_i(e)]$ to 1, for every i such that $0 \leq i < k$. Due to the possible shifting of slices, queries must look for sequences of k slices containing the queried element, from s_0 up to s_{k+l-1}. Hence, a query returns true iff there is an initial index j, where $0 \leq j \leq l$, such that for all i, where $j \leq i < j + k$, $s_i[h'_i(e)] = 1$.

4 Contribution Design and Implementation

Our solution uses a state-based Add-Wins Set CRDT (Figure 2) to test the viability of a probabilistic design. In such a design, the deterministic structure used to represent the causal context (as described in 2.2) is swapped for a probabilistic one. In our particular case, we use an Age-Partitioned Bloom Filter. This is done so that older known event identifiers are forgotten over time, keeping the size of the causal context within bounds.

As the substitute of the traditional deterministic causal context, an APBF or any other probabilistic structure that aims to provide such functionality must take into account the necessary operations - insertions, queries and unions. Considering the inexact nature of stochastic methods, the use of an APBF as a dot context generates an expected volume of errors, which must be minimized in order to negatively affect the consistency between replicas as little as possible.

The essential operations of insertion and querying are natively supported by an APBF. Although the union functionality is indispensable, probabilistic structures for set representation do not usually allow such a feature. To counter this inadequacy, we devised some union strategies for the APBF.

4.1 Union Strategies

Assuming the hash functions and structural parameters (size, number of slices, etc.) are exactly the same between the filters

to be combined, a simple bitwise OR operation combines the contents of filter cells with matching indices.

In an original Bloom Filter (Figure 4), with just one hash area shared by a number of hash functions, the zone to merge corresponds to the whole filter. Alternatively, Partitioned Bloom Filters are divided into slices (Figure 5). In these constructions, slices are individual zones that have to be matched according to their dimensions and fixed hash functions. The number of slices of both filters should also be the same, so that each slice has a match, allowing the fill ratios of slices of the same filter to grow roughly evenly.

Three possible strategies devised for the APBF combine the following subsets of its contents:

- All slices
- Active slices
- Current generation

The merge zone in the strategy that proposes the combination of all slices corresponds to the complete filter. As such, it requires the APBF not to reuse hash functions internally, meaning that each of its slices must be tied to a different hash function.

In contrast, the local merge zone in the strategies that merge only the active slices or only the current generation is formed by the set of active slices. Therefore, within the sequence of k active slices, there are k distinct hash functions. For the remaining l, APBFs merging with any one of these strategies can employ the reuse of hash functions, which allows reusing computed hashes in membership queries.

As their descriptions suggest, the difference between the active-slices and current-generation strategies is that in the former a local active slice subsumes the content of a matching remote active slice, while in the latter a local active slice merges only the content of the current generation of its remote match. Knowing which of the set bits of an active slice correspond to the current generation is made possible by storing a duplicate bit space for every active slice. With that purpose in mind, insertions affect both the slice itself and the respective current generation slice. Upon generation change, the current generation slice is cleared, guaranteeing that older set bits are not included in future content mergers.

4.2 Random Event Identifiers

Elements to be shared in the system - either update operations or their effects - should be uniquely identified in the system. This identification allows testing knowledge of their existence at remote replicas, in order to determine if and how they should be delivered and potentially integrated into the local state. The construction of the identifiers is covered in 2.2. Alternatively, the IDs can contribute to the probabilistic dimension of the solution. As a means to achieve enough uniqueness in the system, IDs could be obtained by generating a fixed length sequence of bytes at random.

The random option has the benefit of potentiating anonymity in the system, which is one of the main drivers of the work here proposed. Anonymity, for instance, can be a desired characteristic if the underlying distributed system has significant churn, which is not dealt with easily in existing solutions.

It may seem that the addition of another probabilistic facet to a system where stochastic mechanisms are already used can increase the error rate. Here, the error probability added by the random dot solution can be reduced by increasing the length of the randomly generated ID. However, these random dots are intended for use in probabilistic data structures, as is the case in our work. Considering the fact that probabilistic data structures represent their input elements probabilistically, whatever uniqueness they had upon insertion will be lost, even if they were perfectly unique to begin with. The loss of uniqueness is manifested in collisions of outputs. It is assumed and expected that the loss of uniqueness generated by the probabilistic structures will be bigger than that caused by the random generation of identifiers, potentially by some orders of magnitude. Due to this, the increase in the error rate by using random dots is not expected to be noticeable.

One further point of consideration lies in memory consumption. Memory consumption should be comparatively limited in a probabilistic structure, since the amount of memory used to represent an inserted element does not depend on its size. In current solutions, the amount of memory consumed by the structure representing the causal context depends on two factors: the size of the identifiers in use and the number of replicas represented. The impact of the size of dots in the space complexity of a causal context is often not an issue, because the compaction scheme offsets any added burden, usually rendering the use of probabilistic structures unattractive. Nevertheless, for a sufficiently large number of replicas represented in the classic causal context there could be an advantage for stochastic methods, including random dots.

5 Results

The results of the present Section were obtained in a simulation environment [6]. The simulator sets up a network according to different parameters and compares the deterministic and probabilistic versions of a Set CRDT.

5.1 Memory Consumption

In order to evaluate the memory consumption of the proposed solution, one has to take into consideration the characteristics of the deterministic and probabilistic structures, their contents, and the network where the CRDTs operate.

In a completely static network, where the set of nodes remains unchanged over time, a deterministic solution could employ the use of a simple vector of counters as the causal context, such as the one described in 2.2.1, where every

index corresponds globally to the same replica. Furthermore, if there is no guarantee of a causally consistent delivery in the system, a supporting set of straggler dots must be represented, contributing to a variable and expectedly minor increase in the amount of memory used.

Assuming a size of 64 bits per counter, allowing the continuous operation of the system without concerns regarding overflows, Figure 7 depicts the slowly increasing use of memory as more replicas are included in the system. When compared with the APBF with capacity for 2048 items, they only match in number of bytes used at around 750 replicas in the network, while the one with capacity for 4096 elements only matches at above 1400 replicas.

In a static network, there is a clear disadvantage when comparing the probabilistic approach to the error-free deterministic one. However, when the network becomes dynamic, the set of known replicas progressively increases. Over sufficient time, the number of entries in the context may increase to values that justify the use of a probabilistic structure.

In addition, in a dynamic network, a vector of counters may not be a flexible enough structure, due to the management of entries and the mapping of indices to replicas. As an alternative, trading the compact size for more flexibility, a map could be used, with replica IDs as keys and counters as values. Given the fact that 128-bit sized UUIDs are commonly used for identification, due to the very low probability of collision of generated values, Figure 7 assumes their use for replica naming.

One other aspect to take into account is the fact that map implementations may allocate more memory than that used by the total of its entries. This aspect has practical relevance and thus must not be ignored. The plot line that considers the memory overhead of the map in Figure 7 relates to the HashMap implementation in the standard library of the Rust language at the time of writing.

Following these changes, the match in memory usage between the classic deterministic structure and the probabilistic one happens at around 250 replicas for the filter with capacity for 4096 dots and at around 125 for the one with space for 2048. In a dynamic network with sufficient churn rate, these figures do not seem unreasonable and support the potential validity of the use of probabilistic structures within causal CRDTs.

The APBFs in Figure 7 use the Current Generation union strategy, which means that every active slice in the filter has a duplicate, used to store only the current generation. This fact further increases the memory consumption of the APBF approach in the comparison.

5.2 APBF Error Rate

The calculation of the error rate requires a comparison of the contents of a classic deterministic Add-Wins Set CRDT with those of an equivalent probabilistic one that saw the same operations. An error is a new inconsistency, which is a difference between the deterministic and probabilistic sets that did not exist in the previous synchronization round. The results show the percentage of the total number of synchronizations that caused new inconsistencies.

Figure 8 presents the percentage of synchronizations that generate new inconsistencies between the classic and the probabilistic versions of a set, for each of the APBF union strategies, in a network with 16 replicas, some churn (1% chance of a node leaving, with new one taking its place) and where 80% of the operations are synchronizations (the remaining 20% are evenly distributed between adds and removes). With such frequent synchronizations, error rates for all strategies are limited, hovering around 2/3%. Also present in the Figure is the fact that when an APBF is set to have a smaller false positive rate (defined by $10^{-\text{error}}$), the error rate remains similar.

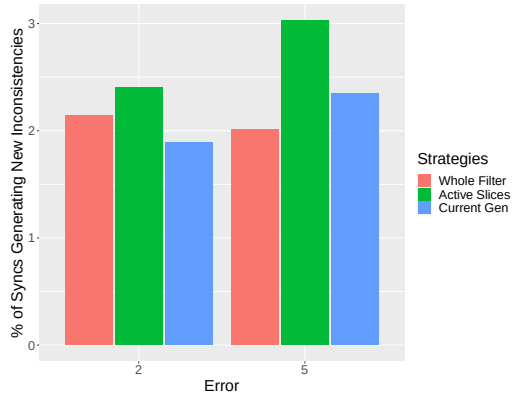

Figure 8. Percentage of error-inducing syncs by union strategy - *error* $[2, 5]$ and 80% syncs

Figure 9 shows the error rates when the synchronization rate drops to 40% (the remaining 60% are evenly distributed between adds and removes). With fewer synchronizations, each of them is expected to set a greater number of bits to 1, increasing the fill ratio in bigger chunks, in comparison with the previous case of Figure 8. By increasing the fill ratio in bigger chunks at once, the detection of saturation of a slice might arrive too late, producing a number of slices with fill ratio values over the designated limit. Furthermore, with more elements inserted at once into the window by the union of slices, the more are expected to move out of the window by slice shifts, which can also contribute to an increased number of errors.

As a result, Figure 9 shows an increase in error rate. The *Active Slices* and *Current Generation* strategies produce very similar proportions of erroneous synchronizations, at around 20% for APBFs built for false positive probabilities of 10^{-2} and 10^{-5}. The *Whole Filter* strategy produce nearly 60% error-inducing synchronizations for ABPFs with false positive

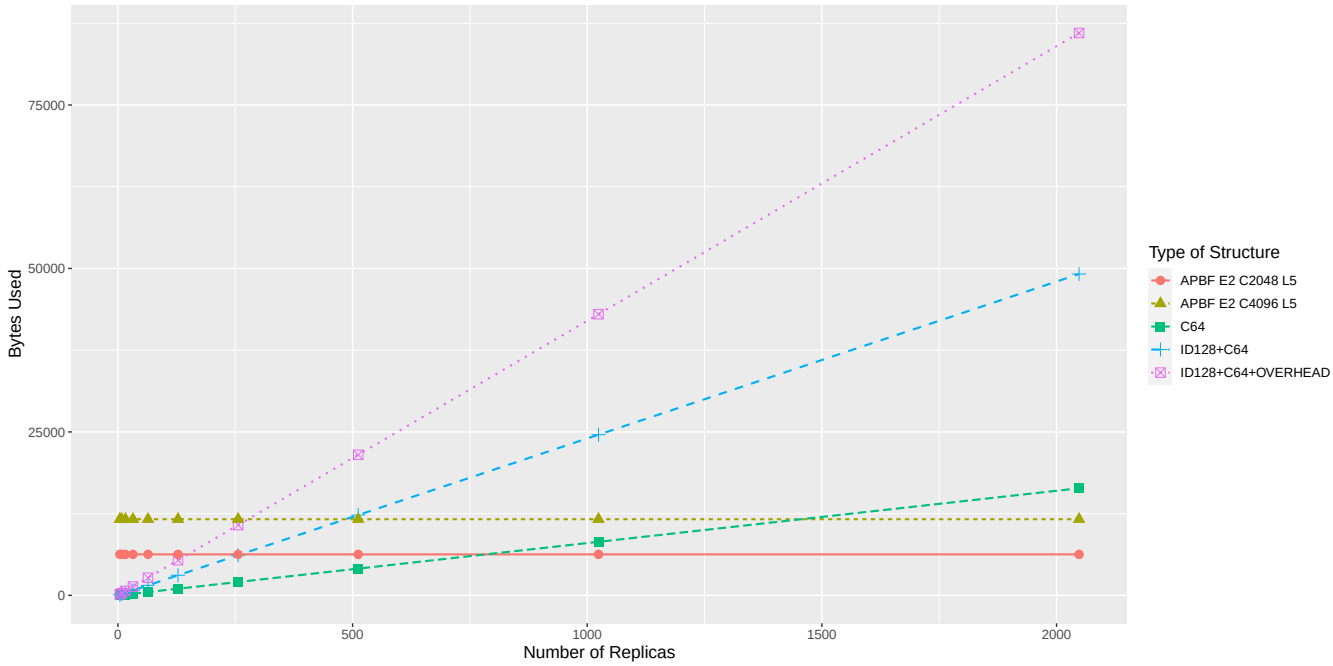

Figure 7. Memory Consumption

probability of 10^{-2} and nearly 40% for ABPFs made for the false positive probability of 10^{-5}.

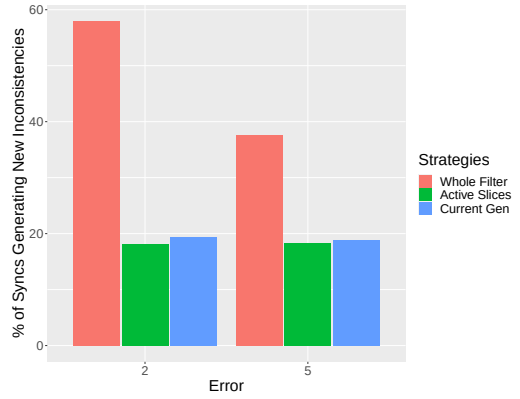

Figure 9. Percentage of error-inducing syncs by union strategy - *error* [2, 5] and 40% syncs

In conclusion, when synchronizations are very frequent, APBF union strategies other than *Current Generation* can produce comparable results. However, due to the lower percentages of inconsistent state it exhibits in suboptimal conditions, *Current Generation* seems to be the more versatile strategy.

6 Conclusion

Our work explored the viability of probabilistic structures at the core of causal CRDTs.

The probabilistic structure of choice was the Age-Partitioned Bloom Filter (APBF) [11] due to its ability to operate on a sliding window over a stream, keeping track of more recently observed elements, and forgetting older ones, thus avoiding saturation or unbounded growth.

Owing to the APBF's lack of native support for union of contents, adaptations were needed so that it could support every operation demanded of a causal context, the particular structure of the classic causal CRDT design targeted to be replaced.

Even considering the expected errors, related to false positives (inherent to Bloom Filter variants) and oblivion of events that still held relevance in the system (inherent to sliding window solutions), the use of APBFs in causal CRDTS showed some promise. In networks with some degree of dynamic behaviour, the probabilistic version of causal CRDTs was capable of providing space savings when compared to the classic design, while keeping the effect of errors somewhat limited. These characteristics, in addition to the lack of need for identifying replicas in the system, may offer additional flexibility over the classic solution, which may be useful for some use cases.

6.1 Future Work

One expected source of improvements over the APBF-based solution would be the conception of a structure capable of

representing a set probabilistically that not only allows insertions and membership queries, but also supports other set operations, with an emphasis on set unions. The same structure should follow the sliding window model, to enable improved scalability and continuous operation. Even though the errors resulting from the oblivion of still-useful elements seem inevitable, there is some expected margin for improvement, which could have very positive effects on the results.

In addition, bearing in mind that state CRDTs are in use, the possibility of sharing delta-states during synchronizations, with support for transitive sharing of state, could align the work described in this document with the state of the art. However, the accuracy of synchronization with deltas might only be sufficient with a probabilistic structures with an improved union operation, like the one suggested previously. Moreover, due to the nature of the existing anti-entropy algorithms for delta-CRDTs, there could be some benefits in exploring what an algorithm that considers that replicas are anonymous would be like, due to the need for identification in the context of garbage collection of deltas.

Acknowledgments

This work is financed by the ERDF - European Regional Development Fund, through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme under the Portugal 2020 Partnership Agreement, and by National Funds through the FCT - Portuguese Foundation for Science and Technology, I.P. on the scope of the UT Austin Portugal Program within project BigHPC, with reference POCI-01-0247-FEDER-045924.

References

[1] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2018. Delta state replicated data types. *J. Parallel and Distrib. Comput.* 111 (2018), 162–173.

[2] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[3] Eric A Brewer. 2000. Towards robust distributed systems. In *PODC*, Vol. 7. Portland, OR, 343477–343502.

[4] Vitor Enes. 2017. *Efficient Synchronization of State-based CRDTs.* Master's thesis. Universidade do Minho. https://vitorenes.org/page/other/msc-thesis.pdf

[5] Pedro Henrique Fernandes. 2021. *Age-Partitioned Bloom filter in Rust.* Retrieved February 27, 2023 from https://github.com/A77377/filte-rs

[6] Pedro Henrique Fernandes. 2021. *Simple simulator used for comparison between classic and probabilistic AWOR-Sets.* Retrieved February 27, 2023 from https://github.com/A77377/probabilistic-aworset-sim

[7] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News* 33, 2 (2002), 51–59.

[8] D Stott Parker, Gerald J Popek, Gerard Rudisin, Allen Stoughton, Bruce J Walker, Evelyn Walton, Johanna M Chow, David Edwards, Stephen Kiser, and Charles Kline. 1983. Detection of mutual inconsistency in distributed systems. *IEEE transactions on Software Engineering* 3 (1983), 240–247.

[9] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. A comprehensive study of convergent and commutative replicated data types. (2011).

[10] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*. Springer, 386–400.

[11] Ariel Shtul, Carlos Baquero, and Paulo Sérgio Almeida. 2020. Age-Partitioned Bloom Filters. *arXiv preprint arXiv:2001.03147* (2020).

[12] Douglas B Terry, Alan J Demers, Karin Petersen, Mike J Spreitzer, Marvin M Theimer, and Brent B Welch. 1994. Session guarantees for weakly consistent replicated data. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*. IEEE, 140–149.

[13] Werner Vogels. 2008. Eventually consistent. *Queue* 6, 6 (2008), 14–19.