Bounded Version Vectors

José Bacelar Almeida[⋆], Paulo Sérgio Almeida, and Carlos Baquero[⋆⋆]

Departamento de Informática, Universidade do Minho
{jba,psa,cbm}@di.uminho.pt

Abstract. Version vectors play a central role in update tracking under optimistic distributed systems, allowing the detection of obsolete or inconsistent versions of replicated data. Version vectors do not have a bounded representation; they are based on integer counters that grow indefinitely as updates occur. Existing approaches to this problem are scarce; the mechanisms proposed are either unbounded or operate only under specific settings. This paper examines version vectors as a mechanism for data causality tracking and clarifies their role with respect to vector clocks. Then, it introduces bounded stamps and proves them to be a correct alternative to integer counters in version vectors. The resulting mechanism, bounded version vectors, represents the first bounded solution to data causality tracking between replicas subject to local updates and pairwise symmetrical synchronization.

1 Introduction

Optimistic replication is a critical technology in distributed systems, in particular when improving availability of database systems and adding support to mobility and partitioned operation [18]. Under optimistic replication, data replicas can evolve autonomously by incorporation new updates into their state. Thus, when contact can be established between two or more replicas, mutual consistency must be evaluated and potential divergence detected.

The classic mechanism for assessing divergence between mutable replicas is provided by *version vectors* which, since their introduction by Parker et al. [14], have been one of the cornerstones of optimistic data management. Version vectors associate to each replica a vector of integer counters that keeps track of the last update that is known to have been originated in every other replica and in the replica itself. The mechanism is simple and intuitive but requires a state of unbounded size, since each counter in the vector can grow indefinitely.

The potential existence of a bounded substitute to version vectors has been overlooked by the community. A possible cause is a frequent confusion of the roles played by *version vectors* and *vector clocks* (e.g. [17, 18]), that have the same representation [14, 5, 13], together with the existence of a minimality result by Charron-Bost [4], stating that vector clocks are the most concise characterization of causality among process events.

[⋆] Partially supported by FCT project POSI/ICHS/44304/2002.
[⋆⋆] Supported in part by FCT under grant BSAB/390/2003.

Operation Init():

$$(\mathsf{V}_i^k)' = 0.$$

Operation Upd(a):

$$(\mathsf{V}_i^k)' = \begin{cases} \mathsf{V}_i^k + 1 & \text{if } i = k = a; \\ \mathsf{V}_i^k & \text{otherwise.} \end{cases}$$

Operation Sync(a, b):

$$(\mathsf{V}_a^k)' = (\mathsf{V}_b^k)' = \mathsf{V}_a^k \sqcup \mathsf{V}_b^k.$$

Fig. 1. Semantics of version vector operations

In this article we show that a bounded solution is possible for the problem addressed by version vectors: the detection of mutual inconsistency between replicas subject to local updates and pairwise symmetrical synchronization. We present a mechanism, *bounded stamps*, that can be used to replace integer counters in version vectors, stressing that the minimality result that precludes bounded vector clocks does not apply to version vectors. Due to space limitations, proofs of Lemmas 1 and 3 are omitted. See [1] for full details.

2 Data Causality

Data causality on a set of replicas can be assessed via set inclusion of the sets of update events known to each replica. Data causality is the pre-order defined by:

$$r_a \leq r_b \quad \text{iff} \quad U_a \subseteq U_b \ ,$$

being U_a and U_b the sets of update events (globally unique events) known to replicas r_a and r_b.

When tracking data causality with version vectors in an N replica system, one associates to each replica $r_i \in \{r_0, \ldots, r_{N-1}\}$ a vector V_i of N integer counters. The order on version vectors is the standard pointwise (coordinatewise) order:

$$\mathsf{V}_a \leq_{\mathsf{V}} \mathsf{V}_b \quad \text{iff} \quad \forall k.\, \mathsf{V}_a^k \leq \mathsf{V}_b^k \ ,$$

where V_i^k denotes component k of vector V_i.

The operations on version vectors, formally presented in Fig. 1, are as follows:

Initialization (Init()) establishes the initial system state. All vectors are initialized with zeroes.

Update (Upd(a)) an update event in replica r_a increments V_a^a.

Synchronization (Sync(a, b)) synchronization of r_a and r_b is achieved by taking the pointwise join (greatest element) of V_a and V_b.

This classic mechanism encodes data causality because comparing version vectors gives the same result as comparing sets of known update events. For all runs and replicas r_a and r_b:

$$r_a \leq r_b \quad \text{iff} \quad U_a \subseteq U_b \quad \text{iff} \quad \mathsf{V}_a \leq_{\mathsf{V}} \mathsf{V}_b \ .$$

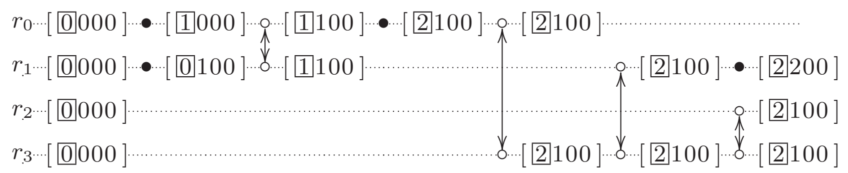
r_0 [0000] [1000] [1100] [2100] [2100]
r_1 [0000] [0100] [1100] [2100] [2200]
r_2 [0000] [2100]
r_3 [0000] [2100] [2100] [2100]

Fig. 2. Version Vectors: example run, depicting slice 0 counters by a boxed digit

Figure 2 shows a run with version vectors in a four replica system. Updates are depicted by a "•" and synchronization by two "◦" connected by a line.

2.1 Version vector slices

All operations over version vectors exhibit a pointwise nature: a given vector position is only compared or updated to the same position in other vectors, resulting from all information about updates originated in replica r_k being stored in component k of each version vector. This allows a decomposition of the replicated system into N *slices*, where each slice represents the updates that were originated in a given replica. Slice i for an N replica system is made up of the i^{th} component of each version vector:

$$\langle \mathsf{V}_0^i, \ldots, \mathsf{V}_{N-1}^i \rangle \ .$$

This means that data causality in N replicas can be encoded by the concatenation of the representation for each of the N slices. It also means that it is enough to concentrate on a subproblem: encoding the distributed knowledge about a single source of updates, and the corresponding version vector slice (VVS). The source of updates increments its counter and all other replicas keep potentially outdated copies of that counter; this subproblem amounts to storing a distributed representation of a total order.

For the remainder of the paper we will concentrate, for notational convenience and without loss of generality, on finding a bounded representation for slice 0. In the run presented in Fig. 2 this slice is shown using boxed counters.

2.2 On version vectors and vector clocks

Asynchronous distributed systems track causality and logical time among communicating processes by means of several mechanisms [12, 19], in particular vector clocks [5, 13]. While structurally equivalent to version vectors, vector clocks serve a distinct purpose. Vector clocks track causality by establishing a strict partial order on the events of processes that communicate by message passing, and are known to be the most concise solution to this problem [4]. Vector clocks, being a vector of integer counters, are unbounded in size, but so is the number of events that must be ordered and timestamped by them. In short, *vector clocks order an unlimited number of events occurring in a given number of processes.*

If we consider the role of version vectors, data causality, there is always a limit to the number of possible relations that can be established on the set of replicas. This limit is independent on the number of update events that are considered on any given run. For example, in a two replica system $\{r_a, r_b\}$ only four cases can occur: $r_a = r_b$, $r_a < r_b$, $r_b > r_a$ and $r_a \parallel r_b$. If the two replicas are already divergent the inclusion of *new* update events on any of the replicas does not change their mutual divergence and the corresponding relation between them. In short, *version vectors order a given number of replicas, according to an unlimited number of update events.*

The existence of a limited number of relations is a necessary but not sufficient condition for the existence of a bounded characterization mechanism. A relation, which is a global abstraction, must be encoded and computed through local operations on replica pairs without the need for a global view. This is one of the important properties of version vectors.

3 Informal presentation

We now give an informal presentation of the mechanism and give some intuition of how it works and how it accomplishes its purpose. Having shown that it is enough to concentrate on a subproblem (a single source of updates) and the corresponding slice of version vectors, we now present the stamp that will replace, in each replica, the integer counter of the corresponding version vector.

For problem size N, i.e. assuming N replicas, with r_0 the "primary" where updates take place and $r_1, \ldots, r_{N-1}$ the "secondary" replicas, we represent a stamp by something like

$$\boxed{\begin{array}{l} \mathbf{c}\ \mathrm{b}\ \mathrm{a} \\ \mathbf{c}\ \mathrm{a} \\ \mathbf{a} \\ \mathbf{c}\ \mathrm{a} \end{array}}\ .$$

It has a representation of bounded size, as it consists of N rows, each with at most N symbols (letters here), taken from a finite set $\mathcal{L}_N$. An example run consisting of four replicas is presented in Fig. 3.

A stamp is, in abstract, a vector of totally ordered sets. Each of the N components (rows in our notation) represents a total order, with the greatest element on the left (the first row above means $c > b > a$). In a stamp for replica r_i, row i ($i \in \{0, \ldots N-1\}$) is what we call the *principal order* (displayed with a gray background), while the other rows are the *cached orders*. (Thus, the stamp above would belong to replica r_3.) The cached order in row j represents the principal order of replica j at some point in time, propagated to replica i (either directly or indirectly through several synchronizations).

The greatest element of the principal order (on the left, depicted in bold over gray) is what we call the *principal element*. It represents the most recent update (in the primary) known by the replica. In a representation using an infinite total ordered set instead of $\mathcal{L}_N$ nothing more would be needed. This element can be thought of as "corresponding" to the value of the integer counter in version vectors. The left column in a stamp (depicted in bold) is what we

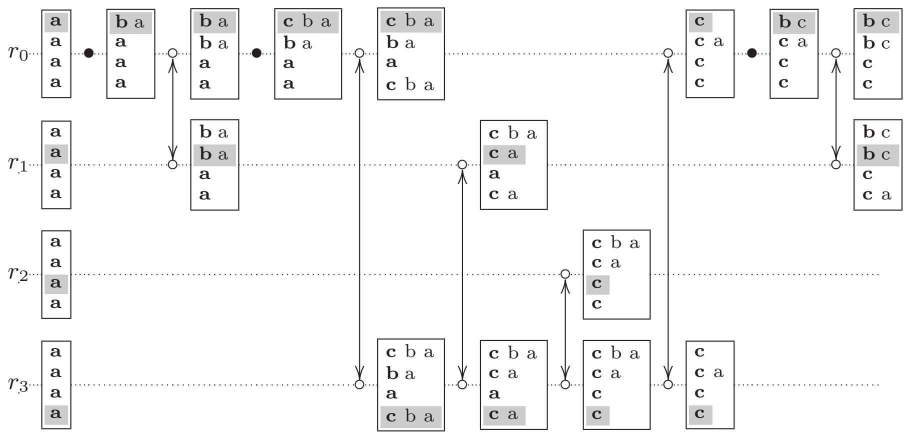
Fig. 3. Bounded stamps: example run

call the *principal vector*; it is made up of the greatest element of each order (row). It represents the most recent local knowledge about the principal element of each replica (including itself). In a stamp, there is a relationship between the principal order and the principal vector: the elements in the principal vector are the same ones as in the principal order. In other words, the set of elements in the principal vector is ordered according to the principal order.

3.1 Comparison and synchronization as well defined local operations

As we will show below, the mechanism is able to compare two stamps by a local operation on the respective principal orders. No global knowledge is used: not even a global order on the set of symbols $\mathcal{L}_N$ is assumed. For comparison purposes $\mathcal{L}_N$ is simply an unordered set, with elements that are ordered differently in different stamps. As an example, the comparison of

$$r_0 = \boxed{\begin{array}{l} \mathbf{b}\ \mathrm{c} \\ \mathbf{c}\ \mathrm{a} \\ \mathbf{c} \\ \mathbf{c} \end{array}} \quad \text{with} \quad r_1 = \boxed{\begin{array}{l} \mathbf{c}\ \mathrm{b}\ \mathrm{a} \\ \mathbf{c}\ \mathrm{a} \\ \mathbf{a} \\ \mathbf{c}\ \mathrm{a} \end{array}}$$

involves looking at $\boxed{\mathbf{b}\ \mathrm{c}}$ and $\boxed{\mathbf{c}\ \mathrm{a}}$, and gives $r_0 > r_1$.

When synchronizing two stamps, in the positions of the two principal elements, the resulting value will be the maximum of the two principal elements; the rest of the resulting principal vector will be the pointwise maximum of the respective values. The comparisons are performed according to the principal orders of the two stamps involved.

It is important to notice that, in general, it is not possible to take two arbitrary total orders and merge them into a new total order. As such, it could be thought that computing the maximum as mentioned above is ill defined. As we will show, several properties of the model can be exploited that make these operations possible and well defined. We will also show that it is possible to totally order the elements in the resulting principal vector, i.e. to obtain a new principal order.

The update of cached-orders is trivial: if the element in the principal vector is updated to a new value, the whole cached order is updated to the corresponding one in the other replica; otherwise is remains as before.

3.2 Garbage collection for symbol reuse

The boundedness of the mechanism is only possible through symbol reuse. When an update operation is performed, instead of incrementing an integer counter, some symbol is chosen to become the new principal element. By using a finite set of symbols $\mathcal{L}_N$, an update will eventually reuse a symbol that was already used in the past to represent some previous update that has been synchronized with other replicas.

However, by reusing symbols, an obvious problem arises that needs to be addressed: the symbol reuse cannot compromise the well-definedness of the comparison operations described above. As an example, it would not be acceptable that, due to reuse, the principal orders of two stamps end up being **a** b c and **c** a, as it would not be possible to overcome the ambiguity between $a > b > c$ and $c > a$ and to infer which one is the greatest stamp.

To address the problem, the mechanism implements a distributed "garbage collection" of symbols. This is accomplished through the extra information in the cached orders. As we will show, any element in the principal order/vector of any replica is also present in the primary replica (in some principal or cached order). This is the key property towards symbol reuse: when an update is performed, any symbol which is not present in the primary replica is considered "garbage" and can be (re)used for the new principal element.

As an example, in Fig. 3, when the final update occurs, symbol b can be used for the new principal element because it is not present in the primary replica. Notice that the scheme only assures that b does not occur in the principal orders/vectors. In this example b occurs in some cached orders of replicas r_1 and r_2, but this is not a problem because those elements will not be used in comparisons; the "old" b will not be confused with the "new" b.

3.3 Synopsis of formal presentation

The formal presentation and proof of correctness will make use of an unbounded mechanism which we call the *counter mode principal vectors* (CMPV). This auxiliary mechanism represents what the evolution of the principal vector would be if we could afford to use integer counters. The mechanism makes use of the total order on natural numbers and does not encode orders locally. In Fig. 4 we present part of the run in Fig. 3 using the counter mode mechanism.

The bulk of the proof consists in establishing several properties of the CMPV model that allow the relevant comparison operations to be computed in a well-defined way using only local information. The key idea is that, exploiting these

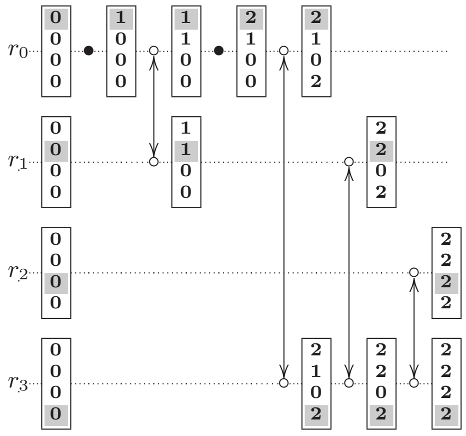

Fig. 4. Counter mode principal vectors

properties, bounded stamps can be seen as an encoding of CMPV using a finite set $\mathcal{L}_N$, where the principal orders are used to encode the relevant order information.

4 Counter Mode Principal Vectors

Version Vector Slices (VVS) rely on an unbounded totally ordered set – the natural numbers. Their unbounded nature is actually a consequence of adopting a predetermined order relation (and hence globally known) to capture data causality among replicas. To overcome this, we enrich VVS in a way that order judgments become, in a sense, local to each replica. In this way, it will be possible to dynamically encode the causality order and open the perspective of bounding the "counters" domain.

For a replica index a, its stamp in the CMPV model is denoted by C_a and defined as the tuple $\langle a, \mathsf{a} \rangle$ where a is a vector of integers with size N – the *principal vector* for C_a (see Fig. 4). The value in position k of vector a is denoted by a^k and represents the knowledge of stamp C_a concerning the most recent update known by stamp C_k. The element a^a plays a central role since it holds C_a's view about the more recent update – this is essentially the information contained in VVS counters and we call it the *principal element* for stamp C_a.

Figure 5 defines the semantics of the operations in the CMPV model.[1] Symbol $\sqcup$ denotes the join operation under integer ordering (i.e. taking the maximum element). Notice that the order information is only required to perform the synchronization operation. Moreover, comparisons are always between principal elements or pointwise (between the same position in two principal vectors).

[1] Recall that the problem under consideration is restricted to slice 0. In particular, this implies that one considers only update events for replica 0.

Operation Init():

$$(\mathsf{a}^k)' = 0.$$

Operation Upd(0):

$$(\mathsf{a}^k)' = \begin{cases} \mathsf{a}^k + 1 & \text{if } a = k = 0; \\ \mathsf{a}^k & \text{otherwise.} \end{cases}$$

Operation Sync(a, b):

$$(\mathsf{a}^k)' = (\mathsf{b}^k)' = \begin{cases} \mathsf{a}^a \sqcup \mathsf{b}^b & \text{if } k \in \{a, b\}; \\ \mathsf{a}^k \sqcup \mathsf{b}^k & \text{otherwise.} \end{cases}$$

Fig. 5. Semantics of operations in CMPV

Occasionally, it will be convenient to write $\mathsf{a} \sqcup \mathsf{b}$ for the result of the synchronization on stamps C_a and C_b (i.e. the principal vector of one of these stamps after synchronization).

A *trace* consists of a sequence of operations starting with Init() and followed by an arbitrary number of updates and synchronizations. In the remainder, when stating properties in the CMPV, we will leave implicit that they only refer to reachable states, i.e. states that result from some trace of operations. Induction over the traces is the fundamental tool to prove invariance properties, as the following simple facts about CMPV.

Proposition 1. *For every stamp C_a, C_b and index k,*

1. $\mathsf{a}^b \leq \mathsf{b}^b$,
2. $\mathsf{a}^a \leq \mathsf{0}^0$,
3. $\mathsf{a}^k \leq \mathsf{a}^a$.

Proof. Simple induction on the length of traces. □

Given stamps C_a and C_b we define their *data causality order under CMPV* ($\leq_{\mathsf{C}}$) as the comparison of their principal elements:

$$\mathsf{C}_a \leq_{\mathsf{C}} \mathsf{C}_b \quad \text{iff} \quad \mathsf{a}^a \leq \mathsf{b}^b \ .$$

By Fig. 5 it can be seen that the computation of principal elements only depends upon principal elements. Moreover, if we restrict the impact of the operations to the principal element we recover the VVS semantics. This observation leads immediately to the correctness of CMPV as a data causality encoding for slice 0:

$$\mathsf{C}_a \leq_{\mathsf{C}} \mathsf{C}_b \quad \text{iff} \quad \mathsf{V}_a^0 \leq_{\mathsf{V}} \mathsf{V}_b^0 \ .$$

This result is not surprising since CMPV was defined as a semantics preserving extension of VVS.

Next we will show that the additional information contained in the CMPV model makes it possible to avoid relying on the integer order, and to replace it with a locally encoded order. For this, we will use a non-trivial invariant on the global state given by the following lemma.

Lemma 1. *For every stamp C_a and C_b and index k,*

$$\mathsf{a}^a \le \mathsf{b}^b \quad \textit{and} \quad \mathsf{b}^k \le \mathsf{a}^k \quad \textit{implies} \quad \mathsf{a}^k \in \mathsf{b} \ .$$

Recall that the order information is only required to perform the synchronization operation. Moreover, comparisons are always between principal elements or pointwise (between the same position in two principal vectors). In the following we will show that these comparisons can be performed without relying on integer order as long as we can order the elements in the principal vector of each stamp individually.

Comparison between principal elements reduces to a membership testing.

Proposition 2. *For every stamp C_a, C_b,*

$$\mathsf{a}^a \le \mathsf{b}^b \quad \textit{iff} \quad \mathsf{a}^a \in \mathsf{b} \ .$$

Proof. $\Longrightarrow$ If $\mathsf{a}^a \le \mathsf{b}^b$ then, by Proposition 1(1) we have that $\mathsf{b}^a \le \mathsf{a}^a$ and so, by Lemma 1, $\mathsf{a}^a \in \mathsf{b}$.

$\Longleftarrow$ If $\mathsf{a}^a \in \mathsf{b}$ then, by Proposition 1(3) we have that $\mathsf{a}^a \le \mathsf{b}^b$. □

For a stamp C_a, let us denote by $\le^{\mathsf{a}}$ the restriction of the intrinsic integer order to the values contained in the principal vector a:

$$x \le^{\mathsf{a}} y \quad \text{iff} \quad x \le y \ \text{ and } \ x \in \mathsf{a} \ \text{ and } \ y \in \mathsf{a} \ .$$

Using these orderings, we define new ones that are appropriate to perform the required comparisons. For stamps C_a and C_b, let their combined order $\le^{\mathsf{ab}}$ be defined as:

$$\begin{aligned} x \le^{\mathsf{ab}} y \quad \text{iff} \quad & (\mathsf{b}^b \in \mathsf{a} \ \text{ and } \ (x \in \mathsf{a} \Rightarrow x \le^{\mathsf{a}} y)) \ \text{ or} \\ & (\mathsf{a}^a \in \mathsf{b} \ \text{ and } \ (x \in \mathsf{b} \Rightarrow x \le^{\mathsf{b}} y)) \ . \end{aligned}$$

For convenience, we also define the corresponding join operation $\underset{\mathsf{ab}}{\sqcup}$ as:

$$x \underset{\mathsf{ab}}{\sqcup} y = \begin{cases} y & \text{if } x \le^{\mathsf{ab}} y, \\ x & \text{otherwise.} \end{cases}$$

The following proposition establishes the claimed properties for this ordering.

Proposition 3. *For every stamp C_a and C_b and index k,*

1. $\mathsf{a}^a \le \mathsf{b}^b$ *iff* $\mathsf{a}^a \le^{\mathsf{ab}} \mathsf{b}^b$,
2. $\mathsf{a}^k \le \mathsf{b}^k$ *iff* $\mathsf{a}^k \le^{\mathsf{ab}} \mathsf{b}^k$.

Proof. (1) Follows directly from Propositions 1 and 2.

(2) $\Longrightarrow$ Let $\mathsf{a}^k \le \mathsf{b}^k$. When $\mathsf{b}^b \le \mathsf{a}^a$ Proposition 2 guarantees that $\mathsf{b}^b \in \mathsf{a}$ and, by Lemma 1, we have $\mathsf{b}^k \in \mathsf{a}$ and then $\mathsf{a}^k \le^{\mathsf{a}} \mathsf{b}^k$, which establishes $\mathsf{a}^k \le^{\mathsf{ab}} \mathsf{b}^k$. The case $\mathsf{a}^a < \mathsf{b}^b$ is trivial since, either $\mathsf{a}^k \in \mathsf{b}$ (in which case $\mathsf{a}^k \le^{\mathsf{b}} \mathsf{b}^k$), or $\mathsf{a}^k \notin \mathsf{b}$ and so $\mathsf{a}^k \le^{\mathsf{ab}} \mathsf{b}^k$. $\Longleftarrow$ Let $\mathsf{a}^k \not\le \mathsf{b}^k$ (that is, $\mathsf{b}^k < \mathsf{a}^k$). The proof proceeds as in the previous implication. □

Restricted orders can be explicitly encoded (e.g. by a sequence) and can be easily manipulated. We now show that when a synchronization is performed, all the elements in the resulting principal vector were already present in the more up-to-date stamp. This means that the restricted order that results is a restriction of the one from the more up-to-date stamp.

Proposition 4. *Let* C_a *and* C_b *be stamps and* $\mathsf{C}_x = \mathsf{C}_a \sqcup \mathsf{C}_b$*. If* $\mathsf{a}^a \leq \mathsf{b}^b$ *then, for all* k*,*

$$\mathsf{x}^k \in \mathsf{b} \ .$$

Proof. For the pointwise join $\mathsf{x}^k = \mathsf{a}^k \sqcup \mathsf{b}^k$: if $\mathsf{a}^k \leq \mathsf{b}^k$ then $\mathsf{x}^k = \mathsf{b}^k \in \mathsf{b}$; if $\mathsf{b}^k \leq \mathsf{a}^k$ then, by Lemma 1, $\mathsf{a}^k \in \mathsf{b}$. Otherwise, note that the resulting principal element (b^b) is already in b. □

These observations together with the fact that the global state can only retain a bounded amount of integer values (an obvious limit is N^2) opens the way for a change in the domain from the integers in the CMPV model to a finite set.

5 Bounded Stamps

A migration from the domain of integer counters in CMPV to a finite set $\mathcal{L}_N$ is faced with the following difficulty: the update operation should be able to choose a value, that is not present in any principal vector, for the new principal element in the primary.

Adopting a set $\mathcal{L}_N$ sufficiently large (e.g. with N^2 elements) guarantees that such a choice exists under a global view. The problem lies in making that choice using only the information in the state of the primary. To overcome this problem we make a new extension of the model that allows the primary to keep track of all the values in use in the principal vectors of all stamps.

We will present this new model parameterized by a set $\mathcal{L}_N$ (the symbol domain), a distinguished element $\mathbf{0} \in \mathcal{L}_N$ (the initial element), and an oracle for new symbols $\text{new}(-)$ (satisfying an axiom described below). For each replica index a, its local state in the bounded stamps model is denoted by B_a and defined as $\langle a, \underline{\mathsf{a}}, \boxed{a} \rangle$ where:

- a is the replica index;
- $\underline{\mathsf{a}}$ is a vector of values from $\mathcal{L}_N$ with size N – the principal vector;
- $\boxed{a}$ is a vector of N total orders, encoded as sequences, representing the full bounded stamp.

This last component contains all the information in the principal vector, the principal order and the cached orders. Although the principle vector $\underline{\mathsf{a}}$ is redundant (as each component $\underline{\mathsf{a}}^k$ is also present in the first position of each $\boxed{a}^k$), it is kept in the model for notational convenience in describing the operations and in establishing the correspondence between the models.

The intuitive idea is that the state for each stamp keeps an explicit representation of the restricted orders. More precisely, for stamp B_a, the sequence

Operation Init():

$$(\underline{\mathsf{a}}^k)' = \mathbf{0},$$
$$(\boxed{a}^k)' = \langle \mathbf{0} \rangle.$$

Operation Upd(0):

$$(\underline{\mathsf{0}}^0)' = \mathrm{new}(\boxed{0}),$$
$$(\boxed{0}^0)' = \mathrm{new}(\boxed{0}) \cdot \boxed{0}^0_{|(\underline{\mathsf{0}})'}.$$

Operation Sync(a, b):

$$(\underline{\mathsf{a}}^k)' = (\underline{\mathsf{b}}^k)' = \begin{cases} \underline{\mathsf{a}}^a \underset{\mathsf{ab}}{\sqcup} \underline{\mathsf{b}}^b & \text{if } k \in \{a, b\}, \\ \underline{\mathsf{a}}^k \underset{\mathsf{ab}}{\sqcup} \underline{\mathsf{b}}^k & \text{otherwise,} \end{cases}$$

if $k \in \{a, b\}$:

$$(\boxed{a}^k)' = (\boxed{b}^k)' = \begin{cases} \boxed{b}^b_{|(\underline{\mathsf{b}})'} & \text{if } \underline{\mathsf{a}}^a \in \underline{\mathsf{b}}, \\ \boxed{a}^a_{|(\underline{\mathsf{a}})'} & \text{otherwise,} \end{cases}$$

if $k \neq a$ and $k \neq b$:

$$(\boxed{a}^k)' = \begin{cases} \boxed{b}^k & \text{if } (\underline{\mathsf{a}}^k)' \neq \underline{\mathsf{a}}^k, \\ \boxed{a}^k & \text{otherwise,} \end{cases}$$
$$(\boxed{b}^k)' = \begin{cases} \boxed{a}^k & \text{if } (\underline{\mathsf{b}}^k)' \neq \underline{\mathsf{b}}^k, \\ \boxed{b}^k & \text{otherwise.} \end{cases}$$

Fig. 6. Semantics of operations on BS model

$\boxed{a}^a$ contains precisely the elements of $\underline{\mathsf{a}}$ ordered downward (first element is $\underline{\mathsf{a}}^a$). From that sequence one easily defines the restricted order for stamp B_a, what we call *principal order* to emphasize its explicit nature.

$$x \leq_{\mathsf{B}^\mathsf{a}} y \quad \text{iff} \quad x = y \ \text{ or } \ \langle y, x \rangle = \boxed{a}^a_{|\{x,y\}} \ ,$$

where $l_{|m}$ denotes the sequence l restricted to the elements in m, i.e. $\langle x \mid x \in l \ \text{ and } \ x \in m \rangle$. The combined order $\leq^{\mathsf{ab}}$ and associated join are defined precisely as in counter mode, that is

$$\begin{aligned} x \leq^{\mathsf{ab}} y \quad \text{iff} \quad & (\underline{\mathsf{b}}^b \in \underline{\mathsf{a}} \wedge (x \in \underline{\mathsf{a}} \Rightarrow x \leq_{\mathsf{B}^\mathsf{a}} y)) \ \text{ or} \\ & (\underline{\mathsf{a}}^a \in \underline{\mathsf{b}} \wedge (x \in \underline{\mathsf{b}} \Rightarrow x \leq_{\mathsf{B}^\mathsf{b}} y)) \ . \end{aligned}$$

The other sequences in $\boxed{a}$ keep information about (potentially outdated) principal orders of other stamps – these are called the *cached orders.*

Figure 6 gives the semantics for the operations in this model. The oracle for new symbols new$(-)$ is a function that gives an element of $\mathcal{L}_N$ satisfying the following axiom:

$$\text{For every stamp } \mathsf{B}_a, \qquad \mathrm{new}(\boxed{0}) \notin \underline{\mathsf{a}} \ .$$

The argument $\boxed{0}$ in the oracle new$(-)$ intends to emphasize that the choice of the new symbol should be made based on the primary local state.

Data causality ordering under the Bounded Stamps model is defined by

$$\mathsf{B}_a \leq_{\mathsf{B}} \mathsf{B}_b \quad \text{iff} \quad \underline{\mathsf{a}}^a \in \underline{\mathsf{b}} \ .$$

The correctness of the proposed model follows from the observation that, apart from the cached orders used for the symbol reuse mechanism, it is actually an encoding of the CMPV model. To formalize the correspondence between both models, we introduce an encoding function $[\![-]\!]_{-}$ that maps each integer in the CMPV model into the corresponding symbol (in $\mathcal{L}_N$) in the state resulting from a given trace. This map is defined recursively on the traces.

$$\begin{aligned} [\![n]\!]_{\mathrm{Init}()} &= \mathbf{0}, \\ [\![n]\!]_{\alpha\,\cdot\,\mathrm{Upd}(0)} &= \begin{cases} \mathrm{new}(\boxed{0}_\alpha) & \text{if } n = \left|\alpha_{|\mathrm{Upd}(0)}\right| + 1, \\ [\![n]\!]_\alpha & \text{otherwise,} \end{cases} \\ [\![n]\!]_{\alpha\,\cdot\,\mathrm{Sync}(x,y)} &= [\![n]\!]_\alpha . \end{aligned}$$

Where $\left|\alpha_{|\mathrm{Upd}(0)}\right|$ is the number of update events in α, $\boxed{0}_\alpha$ is the bounded stamp for the primary after trace α, and $\mathrm{new}(\boxed{0}_\alpha)$ gives a canonical choice for the new principal element on the primary after the update. When we discard the cached orders, the semantics of operations given in Fig. 6 are precisely the ones in CMPV (Figure 5) affected by the encoding map. Moreover, the principal orders are encodings for the restricted orders presented in the previous section.

Lemma 2. *For an arbitrary trace α, replicas index a and b:*

1. $\underline{\mathsf{a}}^k = [\![\mathsf{a}^k]\!]_\alpha$,
2. $[\![\mathsf{a}^i]\!]_\alpha = [\![\mathsf{a}^j]\!]_\alpha$ *implies* $\mathsf{a}^i = \mathsf{a}^j$,
3. $x \leq^{\mathsf{a}} y$ *iff* $[\![x]\!]_\alpha \leq_{\mathsf{B}^{\mathsf{a}}} [\![y]\!]_\alpha$.

Proof. This results from a simple induction on the length of traces. When the last operation was Init() it is trivial. When it was Upd(0), the result follows from the induction hypothesis and the axiom for the oracle new(−). When it was Sync(x, y) the result follows from induction hypothesis, the fact that definitions on both models coincide since $\leq^{\mathsf{ab}}$ computes the required joins (Proposition 3), and the correctness of the new restricted orders (Proposition 4). □

As a simple consequence of the previous result, we can state the following correctness result.

Proposition 5. *For any arbitrary trace α and replica indexes a and b we have*

$$\mathsf{B}_a \leq_{\mathsf{B}} \mathsf{B}_b \quad \textit{iff} \quad \mathsf{C}_a \leq_{\mathsf{C}} \mathsf{C}_b \ .$$

Proof. Immediate from Lemma 2 and the definitions of $\leq_{\mathsf{B}}$ and $\leq_{\mathsf{C}}$. □

It remains to instantiate the parameters of the model. A trivial but unbounded instantiation would be: set $\mathcal{L}_N$ as the integers, $\mathbf{0}$ as value 0 and $\mathrm{new}(\boxed{0}) = \underline{0}^0 + 1$. In this setting, principal orders would be an explicit representation of counter mode restricted orders. Obviously, we are interested in bounded instantiations of $\mathcal{L}_N$. To show that such instantiations exists, we introduce the following lemma that puts in evidence the role of cached orders.

Lemma 3. *For every stamp* B_a *there exists an* i *such that*

$$\boxed{a}^a \subseteq \boxed{0}^i \ .$$

We are now able to present a bounded instantiation for the model. Let $\mathcal{L}_N$ be a totally ordered set with N^2 elements (we have observed by model-checking that not all N^2 elements are needed, but this is enough for our purpose of proving boundedness; the total order is here only to avoid making non-deterministic choices). We define:

$$\begin{aligned} \mathbf{0} &= \sqcap \mathcal{L}_N, \\ \mathrm{new}(\boxed{a}) &= \sqcap\{x \mid x \in \mathcal{L}_N \ \text{ and } \ x \notin \boxed{a}\}. \end{aligned}$$

Lemma 3 guarantees that new($\boxed{0}$) satisfies the axiom. It follows then that it acts as an encoding of counter mode model (Proposition 5). Thus we have constructed a bounded model for the data causality problem in a slice, which generalizes, by concatenating slices, to the full data causality problem addressed by version vectors.

6 Related Work

On what concerns bounded replacements for version vectors there is, up to our knowledge, no previous solution to the problem. The possible existence of a bounded substitute to version vectors was referred in [2] while introducing the version stamps concept. Version stamps allow the characterization of data causality in settings where version vectors cannot operate, namely when replicas can be created and terminated autonomously.

There have been several approaches to version vector compression. Update coalescing [15] takes advantage of the fact that several consecutive updates issued in isolation in a single replica can be made equivalent to a single large update. Update coalescing is intrinsic in bounded stamps since sequence restriction in the update operation discards non-propagated symbols. Dynamic compression [15] can effectively reduce the size of version vectors by removing a common minimum from all entries (along each slice). However, this technique requires distributed consensus on all replicas and therefore cannot progress if one or more replicas are unreachable. Unilateral version vector pruning [17] avoids distributed consensus by allowing unilateral deletion of inactive version vectors entries, but relies on some timing assumptions on the physical-clock's skew.

Lightweight version vectors [9] develop an integer encoding technique that allows a gradual increase of integer storage as counters increase. This technique is used in conjunction with update coalescing to provide a dynamic size representation. Hash histories [10] track data causality by collecting hash fingerprints of contents. This representation is independent of the number of replicas but grows in proportion to the number of updates.

The minimality of vectors clocks as a characterization of Lamport causality [12], presented by Charron-Bost [4] and recently re-addressed in [7], indicates particular runs where the full expressiveness of vectors clocks is required.

However there are cases in which smaller representations can operate: Plausible Clocks [20] offer a bounded substitute to vectors clocks that are accurate in a large percentage of situations and may be used in settings were deviations only impacts performance and not correctness; Resettable Vector Clocks [3] allow a bounded implementation of vector clocks under a specific communication pattern between processes.

The collection of cached copies of the knowledge in other replicas has been explored before in [6, 21] and used for optimization of message passing strategies. This concept is sometimes referred to as matrix clocks [16]. These clocks are based on integer counters and are similar to our intermediate "counter mode principal vector" representation.

7 Conclusions

Version vectors are the key mechanism in the detection of inconsistency and obsolescence among optimistically replicated data. This mechanism has been used extensively in the design of distributed file systems [11, 8], in particular for data causality tracking among file copies. It is well known that version vectors are unbounded due to their use of counters; some approaches in the literature have tried to address this problem.

We have brought the attention to the fact that causally ordering a limited number of replicas does not require the full expressive power of version vectors. Due to the limited number of configurations among replicas, data causality tracking does not necessarily imply the use of unbounded mechanisms. As a consequence, Charron-Bost's minimality of vector clocks cannot be transposed to version vectors. The key to bounded stamps was defining an intermediate unbounded mechanism and showing that it was possible to perform comparisons without requiring a global total order. Bounded stamps were then derived as an encoding into a finite set of symbols. This required the definition of a non-trivial symbol reuse mechanism that is able to progress even if an arbitrary number of replicas ceases to participate in the exchanges. This mechanism may have a broader applicability beyond its current use (e.g. log dissemination and pruning) and become a building block in other mechanisms for distributed systems.

Bounded version vectors are obtained by substituting integer counters on version vectors by bounded stamps. It represents the first bounded mechanism for detection of obsolescence and mutual inconsistency in distributed systems.

References

1. José Bacelar Almeida, Paulo Sérgio Almeida, and Carlos Baquero. Bounded version vectors. Technical Report UMDITR2004.01, Departamento de Informática, Universidade do Minho, July 2004.
2. Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte. Version stamps – decentralized version vectors. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pages 544–551. IEEE Computer Society, 2002.

3. A. Arora, S. S .Kulkarni, and M. Demirbas. Resettable vector clocks. In *19th Symposium on Principles of Distributed Computing (PODC'2000), Portland, 2000*. ACM, 2000.
4. Bernadette Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39:11–16, 1991.
5. Colin Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *11th Australian Computer Science Conference*, pages 55–66, 1989.
6. Michael J. Fischer and A. Michael. Sacrificing serializability to attain high availability of data. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 70–75. ACM, 1982.
7. V. K. Garg and C. Skawratananond. String realizers of posets with applications to distributed computing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'01)*, pages 72–80. ACM, 2001.
8. Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Gerald J. Popek, and Dieter Rothmeier. Implementation of the ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71. USENIX, June 1990.
9. Yun-Wu Huang and Philip Yu. Lightweight version vectors for pervasive computing devices. In *Proceedings of the 2000 International Workshops on Parallel Processing*, pages 43–48. IEEE Computer Society, 2000.
10. Brent ByungHoon Kang, Robert Wilensky, and John Kubiatowicz. The hash history approach for reconciling mutual inconsistency. In *Proceedings of the 23nd International Conference on Distributed Computing Systems (ICDCS)*, pages 670–677. IEEE Computer Society, 2003.
11. James Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transaction on Computer Systems*, 10(1):3–25, February 1992.
12. Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
13. Friedemann Mattern. Virtual time and global clocks in distributed systems. In *Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.
14. D. Stott Parker, Gerald Popek, Gerard Rudisin, Allen Stoughton, Bruce Walker, Evelyn Walton, Johanna Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. *Transactions on Software Engineering*, 9(3):240–246, 1983.
15. David Howard Ratner. *Roam: A Scalable Replication System for Mobile and Distributed Computing*. PhD thesis, 1998. UCLA-CSD-970044.
16. Frédéric Ruget. Cheaper matrix clocks. In *Proceedings of the 8th International Workshop on Distributed Algorithms*, pages 355–369. Springer Verlag, LNCS, 1994.
17. Yasushi Saito. Unilateral version vector pruning using loosely synchronized clocks. Technical Report HPL-2002-51, HP Labs, 2002.
18. Yasushi Saito and Marc Shapiro. Optimistic replication. Technical Report MSR-TR-2003-60, Microsoft Research, 2003.
19. R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 3(7):149–174, 1994.
20. F. J. Torres-Rojas and M. Ahamad. Plausible clocks: constant size logical clocks for distributed systems. *Distributed Computing*, 12(4):179–196, 1999.
21. G. T. J. Wuu and A. J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'84)*, pages 232–242. ACM, 1984.