Adaptive plausible clocks

Anders Gidenstam Marina Papatriantafilou
Department of Comp. Science, Chalmers University of Technology
SE-412 96 Göteborg, Sweden
[ptrianta|andersg]@cs.chalmers.se

***Abstract*— Having small-sized logical clocks with high causal-ordering accuracy is useful, especially where (i) the precision of the knowledge of the causal dependencies among events implies savings in time overhead and (ii) the cost of transmitting Full Vector clock timestamps —that precisely characterise the causal relation— is high. Plausible clocks can be used as timestamps to order events in a distributed system in a way that is consistent with the causal order as long as the events are causally dependent. In this work we study the accuracy of plausible clocks, which is measured by the number of causally independent event pairs that they relate. We introduce the Non-Uniformly Mapped R-Entries Vector (NUREV) clocks, a general class of plausible clocks, which allow the use of clock vectors with a small number of entries and which also allow each process in the system to use a different mapping between process-ids and clock-entry indices, the idea being that dynamic mappings allow self-tuning and adaptation to improve the accuracy of the clocks. Furthermore, we analyse the ways that these clocks may relate causally independent event pairs. Our analysis resulted in a set of conclusions and the formulation of new, adaptive algorithms for plausible clocks, with improved accuracy, even when the number of clock entries is very small, which is important in peer-to-peer communication systems.**

I. INTRODUCTION

To observe consistent states in a distributed execution, it is desirable to be able to establish some order among processes' local states or among events of the execution. Certain consistency requirements demand the ability to produce total orderings. Examples include sequentially consistent distributed shared memory, distributed FIFO queues, among others. Where total orderings are not a must, as for several modern peer-to-peer applications [1], [2], relaxed consistency guarantees such as *causal consistency* can imply higher flexibility and lower overhead [3], [4], [5].

To be able to determine the causal relation between events, i.e. whether they have a cause-effect relation or are independent, we can *timestamp* the events using some clock value. For asynchronous systems without physical clocks, *logical clocks* can be used as mechanisms for timestamping events, in a way so that event orderings based on incremental timestamp values are consistent with causal precedence. *Vector clock* timestamps [6], [7], [8] can capture the exact causal relation at the cost of $O(N)$ clock entries per timestamp for a system of N processes. Since any logical clock that can determine the exact causal relation between events in the system directly from the timestamps requires that a timestamp of size $O(N)$ [9] is included in every message, it is important to investigate the *accuracy* of logical clocks that *approximate* the causal relation and use timestamps with fixed, small size, thus enhancing the scalability of the system. There is another *trade-off* that can be made to gain fixed-size timestamps, namely *time and functionality*: in [10] a timestamping system that characterises causality and has fixed-size timestamps is described. However, in that case only a dedicated checker process, which needs to be notified about all events in the system, can determine the causal relation between events —and sometimes it has to delay its decision until related notifications have arrived.

Torres-Rojas and Ahamad in [4] formalised the notion of *plausible clocks*. A plausible clock can *always determine the order of causally related events correctly* but *may order events which are actually concurrent*. Lamport's logical clocks [11] are an example of plausible clocks. The *inaccuracy* of a plausible clock algorithm is measured by the number of ordering errors it makes in an execution. In [4] a couple of plausible clock algorithms were introduced, namely the R-Entries Vector (REV) clocks and the k-Lamport clocks. A method for combining plausible clocks was also introduced: the combined clock is also plausible and has at least the same or potentially improved accuracy, at the cost of having size equal to the sum of the sizes of its components. The experimental evaluation of the algorithms in [4] has shown that they have improved accuracy compared to Lamport clocks, with the REV clock showing consistently better behaviour. As pointed out in the same paper, the actual accuracy of a plausible clock may depend on a number of factors, such as the size of the system and the execution, the communication patterns among the processes and possibly more. The analysis and evaluation of these possible dependencies, as well as their implications in the design of plausible clock algorithms were left as issues for future research.

The results of our work make significant steps in the investigation of these issues. More specifically, we extend the notion of reduced size vector clocks to include clocks where each process chooses its own mapping between process-ID's and clock entries and furthermore we allow these mappings to change dynamically during the lifetime of the system. We call this class of clocks (which includes the aforementioned R-Entries Vector clock and the Full Vector clock algorithms) *Non-Uniformly Mapped R-Entries Vector* (NUREV) clocks and show that all NUREV clocks are plausible.

The idea behind NUREV clocks is that *dynamic mappings* allow *self-tuning* and *adaptation* of the clocks in order to improve their accuracy. We analyse the ways that plausible clocks may relate causally independent event pairs. Our analysis shows that it is both the communication patterns, as well as the actual clock values — in particular the value-differences among related entries — that influence the accuracy of the clock. These conclusions implied a set of criteria on which to base decisions, which, in turn, resulted in new adaptive mapping strategies, MINDIFF and ROV-MRS, that offer high causal-ordering accuracy.

The experimental evaluation of the performance of our proposed methods agrees well with the conclusions of the analysis part and also shows promising results from the applicability point of view, namely that adaptive plausible clocks with very small number of entries can give very good event-ordering accuracy. In our evaluation we include both *peer-to-peer* communication systems, which are the main target applications for such algorithms, as well as *client-server* communication systems, to enhance the understanding.

After describing our system model, we present our results, in the aforementioned order. We conclude by a discussion of the results and future research issues in this direction.

II. MODEL AND DEFINITIONS

We follow the standard model used in the related literature. The system consists of N sequential processes which are identified by distinct identity numbers, denoted by $p_1, \ldots, p_N$. The processes communicate with each other using messages. The communication is point-to-point and fault-free. There is no physical clock accessible to processes. We make no assumptions about the relative speeds of the processes or the communication channels. Each system execution is a set H of *events*. An event can be the sending of a message, the reception of a message or a local step taken by some process. In each system execution, each process p_i executes a sequence of events, which is called its *local history* for that execution and is denoted by $H_i = e_i^1, e_i^2, e_i^3, \ldots$. We consider the *happened-before* a.k.a. *causal precedence* relation $\rightarrow$ on the set H of all the events of a system execution, as defined by Lamport [11].

Plausible Time-Stamping Systems are mechanisms for timing events in a way that event orderings based on incremental clock values are consistent with causal precedence. The following are the formal definitions paraphrased from [4]:

For a global history H of a distributed system, a *Time-Stamping System* (*TSS*) P is a pair $\left(\langle S, \stackrel{P}{\rightarrow}\rangle, P.\mathbf{stamp}\right)$, where: S is a set of timestamp values (whose details are left open by this definition); $\stackrel{P}{\rightarrow}$ is an irreflexive and transitive relation defined on the elements of S such that $\langle S, \stackrel{P}{\rightarrow}\rangle$ is a strict partial order; $P.\mathbf{stamp}$ is the timestamping function that maps H to S. We write $P.\mathbf{stamp}(a)$, or shorter $P(a)$, to represent the timestamp of event a.

For $u, v \in S$ define: (i) $v \stackrel{P}{=} u$ iff $v = u$; and (ii) $v \stackrel{P}{\parallel} u$ iff $\neg(v \stackrel{P}{=} u) \wedge \neg(v \stackrel{P}{\rightarrow} u) \wedge \neg(u \stackrel{P}{\rightarrow} v)$. Further, we write: $a \stackrel{P}{=} b$ iff $P(a) \stackrel{P}{=} P(b)$, i.e. when P believes that a, b are the same event; $a \stackrel{P}{\rightarrow} b$ iff $P(a) \stackrel{P}{\rightarrow} P(b)$, i.e. when P believes that a precedes b; and $a \stackrel{P}{\parallel} b$ iff $P(a) \stackrel{P}{\parallel} P(b)$, i.e. when P believes that a and b are concurrent.

A time-stamping system P is *plausible* if $\forall a, b \in H$: (i) $a = b$ iff $a \stackrel{P}{=} b$; and (ii) if $a \rightarrow b$ then $a \stackrel{P}{\rightarrow} b$. The second condition is also known as the *weak clock condition*. A time-stamping system *characterises causality* if $\forall a, b \in H$: (i) $a = b$ iff $a \stackrel{P}{=} b$; (ii) $a \rightarrow b$ iff $a \stackrel{P}{\rightarrow} b$; and (iii) $a \| b$ iff $a \stackrel{P}{\parallel} b$. This set of requirements is also known as the *strong clock condition*.

To measure the amount of inaccuracy of a plausible time-stamping system we use the proportion of all pairs of concurrent events in a history that are believed to be causally related by the time-stamping system; i.e.:

$$error(P, H) = \frac{|(a,b) \in H \times H : (a\|b) \wedge (a \stackrel{P}{\rightarrow} b)|}{|(a,b) \in H \times H : a\|b|}.$$

Note that [4] uses the ratio against *all* event pairs instead. We claim that taking the ratio against the number of *concurrent* event pairs is a more precise measure of the accuracy of an plausible clock algorithm since a plausible clock never orders ordered event pairs wrongly.

III. NON-UNIFORMLY MAPPED VECTOR CLOCKS

The *R-Entries Vector Clock* (REV) introduced in [4] is similar to a vector clock but has only R ($\leq N$) entries whereas a full vector clock has N entries. Each process is associated with one entry in the clock vector by a mapping function, $f(\cdot)$, which in the case of the REV clock is *process-ID* mod R.

To address the question, pointed out in [4], of how the choice of mapping function affects the performance of a fixed size vector clock, we introduce the class of Non-Uniformly Mapped R-Entries Vector clocks (NUREV), which generalise the REV clock by allowing each process to use its *own mapping* between process-IDs and clock-entry indices and to *change this mapping* over time. These two generalisations make it possible to define an extended REV clock where each clock may automatically tune itself to perform better for the current communication patterns in the system. Formally a NUREV time-stamping system is a tuple $\left(\langle S, \stackrel{NUREV}{\rightarrow}\rangle, NUREV.\mathbf{stamp}\right)$ where

- S is a set of tuples of the form $\langle i, V_i, f_i\rangle$ where i is an integer that identifies each process in the system, V_i is a $1-$dimensional vector of R integers and f_i is a function from process-ID to entry index $\{1, \ldots, R\}$ (f_i may be different in different tuples).
- $NUREV.\mathbf{stamp}$ is defined by the rules

NUREV0) Initial value:

$$\begin{aligned} i &= \text{unique process-ID} \in \{1, \ldots, N\} ; \\ f_i &= \text{some mapping function} ; \\ V_i[f_i(j)] &= 0 \; \forall j \in 1, \ldots, N \end{aligned}$$

NUREV1) When a send or local event with timestamp $\langle i, V_i^+, f_i^+\rangle$ is generated:

$$\begin{aligned} f_i^+ &= \text{updated } f_i ; \\ V_i^+[r] &= 0 \; \forall r \in \{1, \ldots, R\} ; \\ V_i^+[f_i^+(j)] &= \max(V_i[f_i(j)], V_i^+[f_i^+(j)]) \\ & \quad \forall j \in \{1, \ldots, N\} ; \\ V_i^+[f_i^+(i)] &= V_i^+[f_i^+(i)] + 1 \end{aligned}$$

NUREV2) When a message with timestamp $\langle s, V_s, f_s\rangle$ is received:

$$\begin{aligned} f_i^+ &= \text{updated } f_i \ ; \\ V_i^+[r] &= 0 \ \forall r \in \{1,\ldots,R\} \ ; \\ V_i^+[f_i^+(j)] &= \max(V_i[f_i(j)], V_s[f_s(j)], \\ &\qquad V_i^+[f_i^+(j)]) \ \forall j \in \{1,\ldots,N\} \ ; \\ V_i^+[f_i^+(i)] &= V_i^+[f_i^+(i)] + 1 \end{aligned}$$

- Let $\langle i, V_i, f_i\rangle, \langle j, V_j, f_j\rangle \in S$ then: $\langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle j, V_j, f_j\rangle \Leftrightarrow$
$((i = j \wedge V_i[f_i(j)] < V_j[f_j(j)]) \vee (i \neq j \wedge$
$(\forall k.V_i[f_i(k)] \leq V_j[f_j(k)]) \wedge (V_i[f_i(j)] < V_j[f_j(j)])))$

Intuitively, the NUREV TSS applies the ordinary Vector Clock update and comparison rules to the extended version of the R-entry vectors, using the mapping function to access the clock values. Note that *the NUREV rules themselves do not impose any restrictions on how the mapping function f_i looks nor on how it is changed when updated.* Observe also that the ordinary Vector Clock is a NUREV clock with a fixed mapping function that equals the identity function on N entries and the REV clock of [4] is a NUREV clock with the mapping function fixed to the *process-ID* mod R function.

One important property of the NUREV class of logical clocks is that they all are plausible clocks.

Theorem 3.1: NUREV clocks are plausible Time-Stamping Systems.

Proof: First we need to prove that NUREV is a Time-Stamping System and, then, that NUREV is plausible. The former is proved by showing that the relation $\overset{NUREV}{\rightarrow}$ is irreflexive and transitive. The later is proved by showing that NUREV satisfies the definition of a plausible clock.

Let $\langle i, V_i, f_i\rangle \in S$ and assume that $\langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle i, V_i, f_i\rangle$. Then we have

$$\langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle i, V_i, f_i\rangle \quad \Leftrightarrow \quad (i = i \wedge V_i[f_i(i)] < V_i[f_i(i)])$$

which is a contradiction. So $\overset{NUREV}{\rightarrow}$ is irreflexive.

To prove that *NUREV* is transitive let $\langle i, V_i, f_i\rangle, \langle j, V_j, f_j\rangle, \langle k, V_k, f_k\rangle \in S$ and assume that $\langle i, V_i, f_i\rangle, \overset{NUREV}{\rightarrow} \langle j, V_j, f_j\rangle$ and $\langle j, V_j, f_j\rangle \overset{NUREV}{\rightarrow} \langle k, V_k, f_k\rangle$. The proof is done by case analysis on the sources of the events.

$$\begin{aligned} (i = j = k) \quad &\Rightarrow \quad (i = j \wedge V_i[f_i(j)] < V_j[f_j(j)]) \wedge \\ &\qquad (j = k \wedge V_j[f_j(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad (i = k \wedge V_i[f_i(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad \langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle k, V_k, f_k\rangle \end{aligned}$$

$$\begin{aligned} (i \neq j = k) \quad &\Rightarrow \quad (i \neq j \wedge \forall l.V_i[f_i(l)] \leq V_j[f_j(l)] \wedge \\ &\qquad V_i[f_i(j)] < V_j[f_j(j)]) \wedge \\ &\qquad (j = k \wedge V_j[f_j(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad (i \neq k \wedge \forall l.V_i[f_i(l)] \leq V_k[f_k(l)] \wedge \\ &\qquad V_i[f_i(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad \langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle k, V_k, f_k\rangle \end{aligned}$$

$$\begin{aligned} (j \neq i = k) \quad &\Rightarrow \quad (i \neq j \wedge \forall l.V_i[f_i(l)] \leq V_j[f_j(l)] \wedge \\ &\qquad V_i[f_i(j)] < V_j[f_j(j)]) \wedge \\ &\qquad (j \neq k \wedge \forall l.V_j[f_j(l)] \leq V_k[f_k(l)] \wedge \\ &\qquad V_j[f_j(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad (i = k \wedge V_i[f_i(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad \langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle k, V_k, f_k\rangle \end{aligned}$$

$$\begin{aligned} (k \neq i = j) \quad &\Rightarrow \quad (i = j \wedge V_i[f_i(j)] < V_j[f_j(j)]) \wedge \\ &\qquad (j \neq k \wedge \forall l.V_j[f_j(l)] \leq V_k[f_k(l)] \wedge \\ &\qquad V_j[f_j(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad (i \neq k \wedge \forall l.V_i[f_i(l)] \leq V_k[f_k(l)] \wedge \\ &\qquad V_i[f_i(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad \langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle k, V_k, f_k\rangle \end{aligned}$$

$$\begin{aligned} (i \neq j \neq k) \quad &\Rightarrow \quad (i \neq j \wedge \forall l.V_i[f_i(l)] \leq V_j[f_j(l)] \wedge \\ &\qquad V_i[f_i(j)] < V_j[f_j(j)]) \wedge \\ &\qquad (j \neq k \wedge \forall l.V_j[f_j(l)] \leq V_k[f_k(l)] \wedge \\ &\qquad V_j[f_j(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad (i \neq k \wedge \forall l.V_i[f_i(l)] \leq V_k[f_k(l)] \wedge \\ &\qquad V_i[f_i(k)] < V_k[f_k(k)]) \\ &\Rightarrow \quad \langle i, V_i, f_i\rangle \overset{NUREV}{\rightarrow} \langle k, V_k, f_k\rangle \end{aligned}$$

Thus the NUREV clock is a Time-Stamping System since $\overset{NUREV}{\rightarrow}$ is both irreflexive and transitive.

We need now to proceed by proving that the NUREV TSS is plausible.

Let $a, b \in H$ be two arbitrary events such that $NUREV(a) = \langle i, V_i, f_i\rangle$ and $NUREV(b) = \langle j, V_j, f_j\rangle$.

It is easy to see that

$$\forall a, b \in H : a = b \Leftrightarrow a \overset{NUREV}{=} b.$$

If a and b occurred at the same site then $\overset{P}{\rightarrow}$ will establish their causal relation correctly by comparing $V_i[f_i(j)]$ and $V_j[f_j(j)]$.

Consider the case when a and b occurred at different sites. If $a \rightarrow b$ then from the definition of $NUREV.\mathbf{stamp}$ we have that $\forall l.V_i[f_i(l)] \leq V_j[f_j(l)]$ and $V_i[f_i(j)] < V_j[f_j(j)]$ must hold. This is also what $\overset{NUREV}{\rightarrow}$ requires so

$$a \rightarrow b \quad \Rightarrow \quad a \overset{NUREV}{\rightarrow} b$$

and thereby proving the NUREV TSS to be plausible. ∎

IV. Analysis of Event Orderings using Plausible Clocks

To come up with a gnomon on which to base decisions for which processes should share vector entries, we should identify the cases where such a clock may incorrectly order a pair of concurrent events and draw conclusions about what a mapping function should be aiming at, to recognise as many concurrent events as possible. The results in this section are formalised for NUREV clocks. They can be adapted to any plausible clock in which a notion of mapping between processes and clock entries can be defined.

We assume a global time model, where $t(e_i)$ denotes the global time instant when an event e_i happened. If $T = t(e_i)$ and e_i's timestamp is $\langle i, V_i, f_i\rangle$ we denote this instance of f_i by f_i^T. This assumption is not intended to be used by the algorithms, but only for the sake of arguing about the clocks' behaviour in this analysis. For any event e_i with timestamp $\langle i, V_i, f_i\rangle$, we introduce (again, for the analysis' sake) a corresponding N-entry vector $xp(V_i)$, where $xp(V_i)[j] = V_i[f_i(j)]$. We call $xp(V_i)$ the *expanded form of* V_i. For two vectors xp_1, xp_2 we say that $xp_1 > xp_2$ iff $xp_1[i] \geq xp_2[i]$ for all i and there is at least one entry j such that $xp_1[j] > xp_2[j]$.

For the following, consider e_i, e_j be two arbitrary events of processes i, j $(i \neq j)$ in a system execution and let $\langle i, V_i, f_i\rangle, \langle j, V_j, f_j\rangle$ be their corresponding NUREV timestamps.

Lemma 4.1: If $V_i[f_i(i)] > V_j[f_j(i)]$ then $e_i \overset{NUREV}{\not\to} e_j$ and $e_i \not\to e_j$.

Proof: (sketch) The lemma follows from the definition of NUREV clocks. ∎

Lemma 4.2: If $(V_i[f_i(i)] > V_j[f_j(i)]) \wedge (V_i[f_i(j)] < V_j[f_j(j)])$ then $e_i \overset{NUREV}{\|} e_j$ and $e_i \| e_j$.

Proof: (sketch) Apply the previous lemma (4.1) in both directions. ∎

Notice that for Full Vector clocks and if the timestamp of each event includes the identity of the process that executed it, the condition in the latter lemma is both necessary and sufficient for the two events being concurrent [12]. In the following we analyse the conditions for NUREV to incorrectly relate a pair of concurrent events. Let us first define some terms that will be useful in the rest of this section. Given an event e_i and its corresponding NUREV timestamp $\langle i, V_i, f_i\rangle$, we call:

- $V_i[f_i(i)]$: e_i's *own key*.
- $V_i[f_i(j)]$ $(i \neq j)$: p_i's *presumption of the own key of* p_j's *latest event preceding* e_i. (For the limit case when there is no event by p_j preceding e_i, consider the *initialising event* by each process, that sets all the vector entries to 0, to precede the first events in the executions of all processes).

Lemma 4.3: If $V_i[f_i(i)] \leq V_j[f_j(i)]$ then $e_i \to e_j$ or $e_i \| e_j$ and there exists some event e_k timestamped by $\langle k, V_k, f_k\rangle$ by a process p_k $(k \neq i)$ such that $e_k \to e_j$ and $V_i[f_i(i)] \leq V_k[f_k(k)]$ and $f_j^T(i) = f_j^T(k)$, where $T = t(e'_j) : e_k \to e'_j \to e_j$ or $e'_j = e_j$.

Proof: (sketch) From lemma 4.1 it follows that $e_j \not\to e_i$. Then either $e_i \to e_j$ or $e_i \| e_j$. In the latter case, assume, towards a contradiction that there is not such event e_k as stated in the lemma. The contradiction can be reached by using the plausibility property of the clock and the fact that the mapping function may map several processes to some entry. ∎

Intuitively, the lemma states that clocks advance because of precedence or because of merging entries, i.e. that using NUREV clocks, e_j's presumption of the own key of p_i's latest event preceding e_j can be an *inflated value*, since p_i might share an entry in p_j's clock with another process in the meanwhile. Note that when using Full Vector clocks, e_j's presumption of the own key of p_i's latest event e_i preceding e_j is exactly equal to e_i's own key [12]. Figure 1 graphically describes the possible cases of a pair of events' actual relation and how NUREV clocks may capture it.

Lemma 4.4: If $e_i \to e_j$ and there is no event e'_i by p_i such that $e_i \to e'_i \to e_j$ and $V_i[f_i(i)] = a$ and $V_j[f_j(i)] = a + B$ (for some $B > 0$) then for all e_i^l $(1 \leq l \leq x, x \leq B)$ such that $e_i^l \| e_j$ and that $e_i \to e_i^1 \to \ldots \to e_i^l \to \ldots \to e_i^x$, it is possible that $e_i^l \overset{NUREV}{\to} e_j$.

Proof: (sketch) Since $e_i \to e_j$, $xp(V_i) < xp(V_j)$. Process p_i may increase its vector by modifying only its own entry $(f_i(i))$ only by one in each such e_i^l, thus causing their timestamp vectors to compare similarly with V_j. Since $V_j[f_j(i)] = a + B$ this can cause NUREV timestamps to order e_j with up to B consecutive events of p_i which follow e_i and which are concurrent with e_j (cf. Figure 2). ∎

This means that if e_j's presumption of the own key of p_i's last preceding event is *inflated* by B, this may result in e_j to be NUREV-ordered with a maximum number of B events of p_i which are in fact concurrent with e_j. Since the length of such a sequence of events e_i^l as described in the lemma is related to B, let us call it a B_{dep}*-error-prone sequence* of p_i caused by the event pair (e_i, e_j) (cf. Figure 2). Moreover, let us define *time-to-hear-back*(e_i, p_j) as follows: if $e_i^1, \ldots, e_i^h (h \geq 1)$ is the minimal sequence of p_i's events between (i) event e_i by p_i that precedes an event e_j of p_j such that there is no event e'_i by p_i which $e_i \to e'_i \to e_j$ and (ii) an event e_i^{h+1} $(h \geq 0)$ such that $e_j \to e_i^{h+1}$, then *time-to-hear-back*$(e_i, p_j) = h$. If there is no such event e_i^{h+1}, we consider the last event of p_i in the execution, instead. In other words, after an event e_i that is directly preceding an event (e_j) of p_j, the *time-to-hear-back*(e_i, p_j) is the time, i.e. number of events, for p_i to hear back from p_j, i.e. to reach an event e_i^{h+1} that is directly preceded by e_j.

Corollary 4.1: Given the precondition in lemma 4.4 and the corresponding B_{dep}*-error-prone sequence*, the length of the sequence is bounded from above by *time-to-hear-back*(e_i, p_j).

Proof: (sketch) The timestamp of e_j (or of a subsequent event that causes the precedence between e_j and e_i^{h+1}) informs p_i about the inflation of e_i's own value by p_j. Moreover, this precedence will actually establish order among the events of the processes (cf. Figure 2). ∎

The last two lemmas imply that the length of a B_{dep}*-error-prone sequence* of some p_i caused by some (e_i, e_j) is bounded from above by (i) B, the difference between e_i's own key and

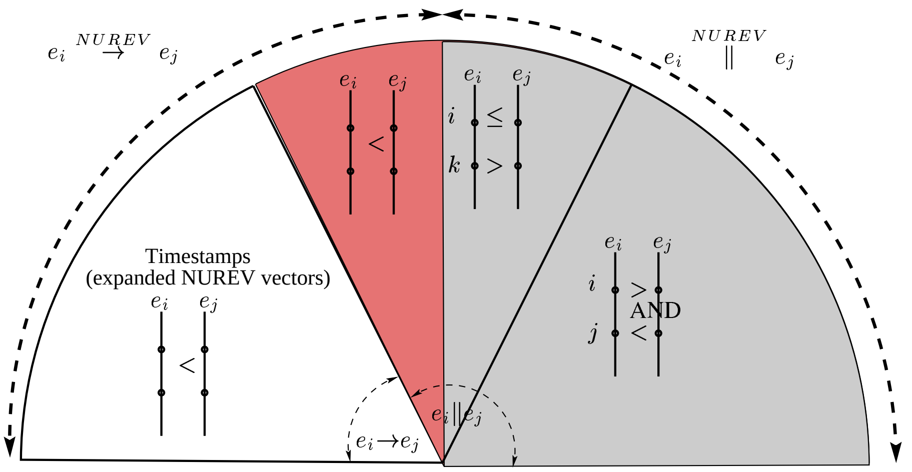

Fig. 1. Graphical representation of a pair of events' actual relation and how NUREV clocks capture it

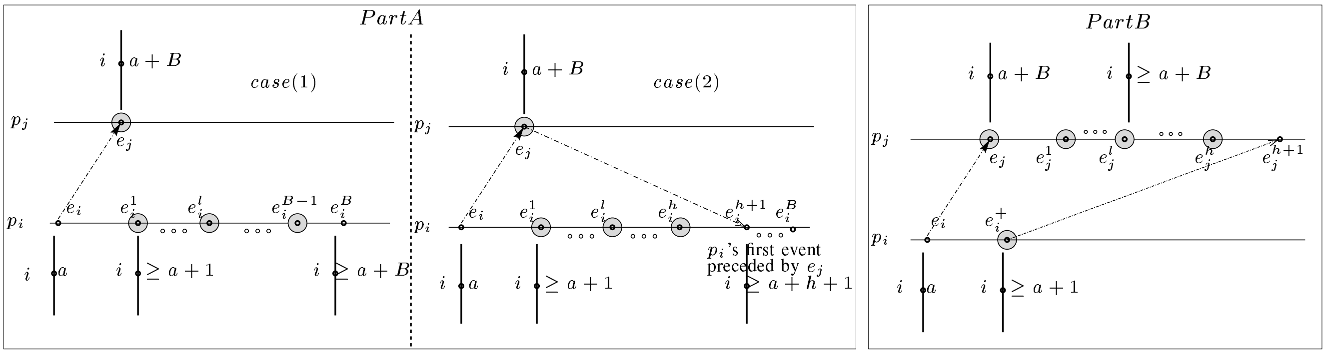

Fig. 2. *Possible errors by e_j's inflated (by $B > 0$) presumption of the own key of e_i (p_i's last event preceding e_j). Vertical lines represent the events' xp's. Circles indicate events of p_i and p_j that are pairwise concurrent but may be NUREV-ordered.* **Part A:** *The corresponding B_{dep}-*error-prone sequence *is bounded by B (case 1) and the* time-to-hear-back(e_i, p_j) *(case 2).* **Part B:** *The corresponding B_{indep}-*error-prone sequence *is bounded from above by the* time-to-hear-again(p_i, e_j).

e_j's presumption of it and (ii) the *time-to-hear-back*(e_i, p_j).

Lemma 4.5: Consider the case that $e_i \rightarrow e_j$ and there is no event e_i' by p_i such that $e_i \rightarrow e_i' \rightarrow e_j$ and $V_i[f_i(i)] = a$ and $V_j[f_j(i)] = a + B$ (for some $B > 0$). If there exists e_i^+ such that $e_i \rightarrow e_i^+$ and $e_i \| e_j^1$ and $e_i^+ \overset{NUREV}{\rightarrow} e_j^1$, where $e_j^1 = e_j$ or $e_j \rightarrow e_j^1$, then $e_i^+ \overset{NUREV}{\rightarrow} e_j^l$ for each e_j^l such that $e_j^1 \rightarrow e_j^l$ and $e_j^l \| e_i^+$.

Proof: (sketch) Let V_e denote the vector of the timestamp of an event e. If $e_i^+ \overset{NUREV}{\rightarrow} e_j^1$ then from the definition of NUREV clocks follows at $xp(V_{e_i^+}) < xp(V_{e_j^1})$. Still from the same definition and the plausibility property follows that $xp(V_{e_j^+}) < xp(V_{e_j^l})$, which implies the lemma. ∎

This means that if e_j's presumption of the own key of p_i's last preceding event (e_i) is *inflated* by B, this may result in some event(s) e_i^+ of p_i subsequent to e_i to be NUREV-ordered with a sequence of events of p_j subsequent to e_j but actually concurrent with e_i^+ (cf. Figure 2). Since the length of such a sequence of e_j^l events as described in the lemma is not related to B, let us call it a B_{indep}-*error-prone sequence* of p_j caused by (e_i, e_j). Moreover, let us define *time-to-hear-again*(p_i, e_j) to be the length h of the sequence of p_j's events between e_j and the event e_j^{h+1} ($h \geq 0$) which is the first event by p_j that is preceded both by the event e_i and the event e_i^+ by p_i, where e_i is the event by p_i directly preceding e_j, i.e. there is no event e_i' by p_i such that $e_i \rightarrow e_i' \rightarrow e_j$, and e_i^+ is the event by p_i that is directly preceded by e_i. If there is no such event e_j^{h+1}, we consider the last event of p_j in the execution, instead. Note that for infinite executions this implies that *time-to-hear-again*(p_i, e_j) can have an infinite value. In other words, after an event (e_i) of p_i that is directly preceding an event e_j (of p_j), the *time-to-hear-again*(p_i, e_j) is the time, i.e. number of events, to hear again from p_i, i.e. to have the first event e_j^{h+1} at p_j which is directly preceded by an event of p_i subsequent to e_i.

Corollary 4.2: Given the precondition in lemma 4.5 and the corresponding B_{indep}-*error-prone sequence*, the length of the sequence is bounded from above by *time-to-hear-again*(p_i, e_j).

By combining lemmas 4.4 and 4.5 and their resulting corollaries, we get that: if during the clock update of an event e_j the presumed value of p_i's last preceding event e_i is inflated by $B > 0$, then the corresponding B_{dep}*-error-prone sequence* of p_i caused by (e_i, e_j) and the potential B_{indep}*-error-prone sequence* p_j caused by (e_i, e_j), together imply a total number of possibly NUREV-ordered concurrent events which may be as high as the product of the length of the two sequences. The total number of errors implied by the inflated value is bounded from above by:

$$\begin{aligned} &min\{\textit{time-to-hear-back}(e_i, p_j), B\} \times \\ &\qquad \textit{time-to-hear-again}(p_i, e_j) \leq \\ &\qquad B \times \textit{time-to-hear-again}(p_i, e_j) \end{aligned}$$

Hence, considering a process p_j at the point when it must decide the new mapping when executing an event e_j, the goals of its adaptive mapping function to minimise the number of NUREV-ordered concurrent event pairs should be:

Inflation

In the timestamp of e_j the sharing of entries must be arranged so that the inflation of the resulting timestamp's presumed values for each process' latest preceding event is kept minimal.

Next-Contact

In the timestamp of e_j the sharing of entries must be such that: the longer the time intervals (length of event sequences) until p_j and p_i communicate again after e_j (directly, or indirectly, via another process in the system), the smaller inflation is permitted in e_j's presumption about p_i's last preceding event.

The first conclusion implies that values with large differences should not be assigned to share the same entry, since the lower one must be equalised to the higher one to maintain plausibility. The second conclusion implies that an optimal adaptive mapping may need information about the *future* in order to decide how processes should share entries in a timestamp value.

V. MINDIFF NUREV CLOCK

Armed with the conclusions above one can see that errors are introduced when a process' presumption of the own key of another process is inflated, so a good plausible clock should try to minimise this inflation.

There are two operations on a NUREV clock where inflation can occur. The first operation is the local step or send operation where inflation can occur only if the local process shares its clock entry with another process. This can be avoided by always assigning a clock entry for the local process' own exclusive use. The second case is the receive operation, where it is impossible to avoid inflation in all cases since a clock with R entries cannot hold more than R different values.

We now introduce the MINDIFF NUREV clock that attempts to minimise the inflation at each receive operation. Let us take a closer look at the exact information available to a process when updating its clock at a message arrival. The major steps in the algorithm presented below are also visualised by the example in Figure 3. Let $\langle i, V_i, f_i\rangle$ be the clock of process i when a timestamp $\langle j, V_j, f_j\rangle$ is received from process j. First we define a full-sized clock vector W_i that is produced by merging the local clock and the received timestamp without inflation, where each entry also is marked with its owning process, i.e.

$$\begin{aligned} W_i[k] &= \max(V_i[f_i(k)], V_j[f_j(k)]) \ \forall k \in \{1, \ldots, N\}; \\ W_i[k].id &= k \ \forall k \in \{1, \ldots, N\}. \end{aligned}$$

Note that there can be at most $2R$ different values in W_i since V_i and V_j can hold only R different values each.

The next step is to decide which processes should share entries in the updated clock vector. If we sort the entries in W_i (excluding the process' own entry) in increasing order we can easily compute a cost matrix that reflects how much inflation different ways of mapping the processes to the R clock entries cause. Let $\hat{W}_i$ be the sorted version of W_i, then the lower triangular cost matrix $(C_{k,l})$ is defined as follows:

$$\begin{aligned} C_{k,k} &= 0, \ k \in \{1, \ldots, N\}; \\ C_{k,l} &= C_{k,(l+1)} + \hat{W}_i[k] - \hat{W}_i[l] \text{ for } l < k, \end{aligned}$$

where $C_{k,l}$ is the cost of letting the set $\{p_n | n = \hat{W}_i[r].id \text{ for } k \leq r \leq l\}$ of processes share the same entry in the updated clock vector.

In the MINDIFF clock we try to minimise the sum of the inflation suffered by all process entries. A minimal-cost-assignment of processes to clock entries corresponds to a partitioning of the sorted sequence $\hat{W}_i$ into $R-1$ blocks (one entry is the process' own entry).

Note that since $\hat{W}_i$ contains at most $2R$ different values it is trivial to compress the size of the cost matrix from $N \times N$ to $2R \times 2R$ by joining the $\hat{W}_i[k]$:s that have the same value and adjust the corresponding cost entry accordingly. The sorting of the values in $\hat{W}_i$ can be made by a linear merge ($2R$ steps), if one maintains the clock entries sorted in the timestamps.

In Figure 4 we present a new algorithm that finds a good K-partitioning of a L-element sequence in $O(K \cdot L)$ steps. Initially the algorithm places the $K-1$ partition boundaries in at the first $K-1$ of the $L-1$ places where a partition boundary can be placed. Let b_k be the position of the k-th partition boundary and let b_0 and b_K denote position 0 and L, respectively. Define the cost of the partitioning $b_1, \cdots, b_{K-1}$ as

$$cost(b_1, \cdots, b_{K-1}) = \sum_{k=1}^{K} C_{b_k, b_{k-1}+1}.$$

The partitioning algorithm proceeds by selecting the right-most partition boundary and moving it to the right until it it reaches a local cost minimum. Then it selects the next partition boundary to the left and moves it to the right until it either finds a local minimum or reaches the first boundary. This procedure is repeated for all boundaries until no changes occur. Note that since the boundaries only move to the right, the algorithm needs at most $O(K \cdot L)$ steps before it terminates.

Once the partitioning of the processes into $R-1$ blocks is determined the new mapping function is determined by letting

Clock p_1

$\{p_1\}$,	15
$\{p_6\}$,	0
$\{p_3, p_4, p_7\}$,	7
$\{p_2, p_5\}$,	20

Timestamp from p_3

$\{p_3\}$,	10
$\{p_1, p_4\}$,	3
$\{p_2, p_6\}$,	9
$\{p_5, p_7\}$,	17

Full-sized updated clock

$\{p_1\}$	16
$\{p_2\}$	20
$\{p_3\}$	10
$\{p_4\}$	7
$\{p_5\}$	20
$\{p_6\}$	9
$\{p_7\}$	17

Cost matrix

		p_4	p_6	p_3	p_7	p_2	p_5
		7	9	10	17	20	20
p_4	7	0					
p_6	9	2	0				
p_3	10	4	1	0			
p_7	17	25	15	7	0		
p_2	20	34	21	10	3	0	
p_5	20	34	21	10	3	0	0

Updated clock p_1

$\{p_1\}$,	16
$\{p_4\}$,	7
$\{p_3, p_6\}$,	10
$\{p_2, p_5, p_7\}$	20

Fig. 3. Example of MINDIFF clock update at message reception.

$b_0 \leftarrow 0$; $b_K \leftarrow L$; $b_k \leftarrow k$ for $k = 1, \ldots, K-1$
repeat
  for $k = 1, \ldots, K-1$ **do**
    while $cost(b_1, \cdots, b_k, \cdots, b_{K-1}) > cost(b_1, \cdots, b_k + 1, \cdots, b_{K-1})$ **and** $b_k + 1 < b_{k+1}$ **do** $b_k \leftarrow b_k + 1$
until no b_k changed.

Fig. 4. The K-partitioning algorithm.

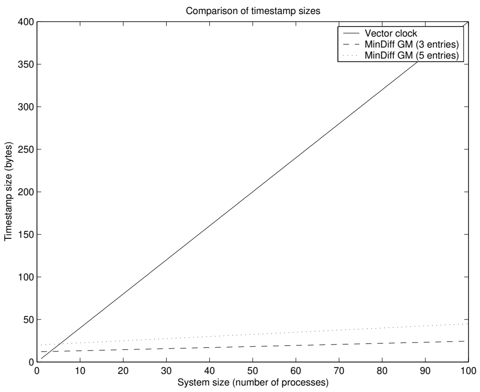

Fig. 5. Growth of timestamp size. The size of a clock entry is 4 bytes.

the processes in each block share an entry, and the new value for entry r is simply $\hat{W}_i[b_r]$.

To be able to store any arbitrary mapping that comes as a result of the grouping of entries according to the MINDIFF clock, the total space requirement of a timestamp would be R clock entries and N values of $\log_2 R$ bits each. This is significantly smaller than the N clock entries required by the full-size vector clock — cf. figure 5 for a comparison of the growth of the corresponding timestamp values as the system size grows. Still, it is challenging to study, for mappings with constant-size representation, how to follow the conclusions of our analysis, in order to avoid, as much as possible, to introduce orderings of concurrent events.

VI. ROV-MRS NUREV CLOCK

Consider a NUREV clock that at each process, maps $R-1$ of the other processes to exclusive clock entries and all other processes to the remaining clock entry, called the "others entry". The owner of a clock and the source process of a timestamp is always assigned an exclusive clock entry, for the same reason as in the MINDIFF clock. We call this clock the *R-others vector clock* (ROV). The formal description of the ROV clock algorithm is given in figure 6.

Let $ROV = \left(\langle S, \stackrel{ROV}{\rightarrow} \rangle, ROV.\mathbf{stamp}\right)$ where

- S is a set of tuples of the form $\langle i, V_i, f_i \rangle$ where i is an integer that identifies each process in the system, V_i is a $1-$dimensional vector of R integers and f_i is a function from process-IDs to $\{1, \ldots, R\}$ such that i and at most $R-2$ other process-IDs are bijectively mapped to $\{1, \ldots, R-1\}$ and all other process-IDs are mapped to R
- $ROV.\mathbf{stamp}$ is defined by the rules
  ROV0) Initial value:
  $$\begin{aligned} i &= \text{unique process} - \text{ID} \in \{1, \ldots, N\} \; ; \\ f_i &= \{i \mapsto 1, \forall j \neq i. j \mapsto R\} \; ; \\ V_i[f_i(j)] &= 0 \; \forall j \in 1, \ldots, N \end{aligned}$$
  ROV1) Before a send or local event with timestamp $\langle i, V_i^+, f_i \rangle$ is generated:
  $$\begin{aligned} f_i & \quad \text{is not changed} \; ; \\ V_i^+[f_i(i)] &= V_i[f_i(i)] + 1 \end{aligned}$$
  ROV2) When a message with time-stamp $\langle s, V_s, f_s \rangle$ is received:
  $$\begin{aligned} f_i^+ &= \text{updated } f_i \; ; \\ V_i^+[r] &= 0 \; \forall r \in \{1, \ldots, R\} \; ; \\ V_i^+[f_i^+(j)] &= \max(V_i[f_i(j)], V_s[f_s(j)], \\ & \qquad V_i^+[f_i^+(j)]) \; \forall j \in \{1, \ldots, N\} \; ; \\ V_i^+[f_i^+(i)] &= V_i^+[f_i^+(i)] + 1 \; ; \end{aligned}$$
- Let $\langle i, V_i, f_i \rangle, \langle j, V_j, f_j \rangle \in S$ then:

$$\begin{aligned} \langle i, V_i, f_i \rangle \stackrel{ROV}{\rightarrow} \langle j, V_j, f_j \rangle \quad \Leftrightarrow \quad & (i = j \wedge V_i[f_i(j)] < V_j[f_j(j)]) \vee \\ & (i \neq j \wedge (\forall k. V_i[f_i(k)] \leq V_j[f_j(k)]) \\ & \wedge (V_i[f_i(j)] < V_j[f_j(j)])) \end{aligned}$$

Fig. 6. Definition of the R-others vector clock. Note that the set of allowed mapping functions is restricted to those where only entry R is shared by more than one process. The operation updating the mapping should only produce mappings from that set.

This mapping-class has constant-size representation, but the issue of which processes should be allocated to exclusive clock entries in each process' clock remains to be solved. Following the conclusions from the analysis section, we propose the following strategy:

Most Recent Senders (ROV-MRS) mapping: In this policy the $R-2$ last processes the process received messages from are mapped to exclusive entries in the clock. If there are less than $R-2$ such processes, that is, there are some unused exclusive entries, those entries are allocated to processes that had exclusive entries in the most recently received timestamps. All other processes (apart from the process itself) are mapped to the others-entry. Since each process assigns exclusive entries to its most recent senders, it will thus add *no inflation* to their entries. Since these processes might take longer than others to send something again (*time-to-hear-again* can be longer than the other processes), such a mapping would agree well with the [Next-Contact]-conclusion of the analysis.

It is worth pointing out that considering the opposite strategy, namely a Least Recent Senders (ROV-LRS) mapping, the results from section IV imply that: a process which is a non-sender for long time might send a message soon (i.e. *time-to-hear-again* can be short) and hence establish order among events, so it may not need an exclusive entry and that entry could be used to prevent other errors instead. This argues against such a choice.

VII. Experimental evaluation

We first study *peer-to-peer* communication systems, the main target application domain for such algorithms. As shown in section IV, the communication patterns play a special role in the accuracy of the time-stamping systems. We study the *client-server* communication systems separately, as they have very different communication patterns. In particular, considering that direct communication will only take place between the clients and the server(s), the servers have a key-role, since it is they who are likely to cause and propagate a large portion of the errors. A parallel consideration is that the servers may also play a key-role in the *actual ordering* of the events (requests). However, for the purpose of enhancing the understanding of plausible clocks, we wish to discuss their accuracy, as well.

In the discussion we use the terms and conclusions of section IV. The analysis outcome is compared with the outcome of the experimental evaluation. The experiments were conducted by creating system histories of simulated distributed systems and annotating the events with timestamps from a number of different plausible clocks and also a full vector clock to measure the accuracy of the plausible clocks[1].

The plausible clocks we focus on are: the *R-Entries Vector Clock* (REV) [4], our *R-Others Vector Clock* with the *Most Recent Senders* (ROV-MRS) dynamic mapping policy and our MinDiff clock. The k-Lamport clocks [4] were not included as their behaviour is such that: their accuracy for small k is rather low, while after the first few values of k, adding more entries in the vector, the accuracy does not improve. This is explained by our analysis, since these clocks do not keep per-process information in the clock. Similar is the effect of combining k-Lamport clocks with other plausible clocks using the methodology in [4]. Our experiments confirmed these conclusions. An experimental study on how the performance of the combination of REV and k-Lamport clock depends on different system parameters, is presented in [13].

A. Peer-to-peer communication

Experiment description: The system history is generated by letting each process randomly select whether to send or receive a message or to perform a local step. The destination of each message is selected at random. Figure 7 presents results from peer-to-peer experiments.

Discussion of results: The *R-Others Vector Clock* with the *Most Recent Senders* dynamic mapping policy (ROV-MRS) shows good accuracy even for a very small number of clock entries (i.e. small R). Recall that for this clock each process assigns unique entries to its most recent senders, thus adding no inflation to their entries. Since these processes might take longer than others to send something again (*time-to-hear-again* can be long), the performance result conforms with the [Next-Contact]-conclusion of the analysis. From the point of view of the history length, as also indicated by figure 7, when the history becomes longer, the accuracy-improvement curve for these clocks tends to become less sharp. This is explained via the fact that as the history becomes longer, a process will need more entries to save more accurate information about its recent senders.

Regarding MinDiff's performance, as expected from the analysis and its design to follow the [Inflation]-conclusion, it is very close to that of full vector clocks even for very small number of clock entries. Combined with figure 5, which compares the MinDiff's and Full Vector Clock's timestamp sizes, the results look promising from the applicability point of view, especially when considering the scalability wanted in peer-to-peer systems. From the intellectual-challenge point-of-view, constant-size representation of mapping functions may deserve more investigation.

B. Client-server communication

Experiment description: The system history is generated by letting each client randomly decide whether to send a request, receive a response or do a local step. The server processes requests immediately and in FIFO order. Figure 8 shows results from client-server executions.

Discussion of results: Consider the servers' and the clients' perspectives separately:

• When a *server* receives a message from a client: (i) The *time-to-hear-back* (from the server) for that client is likely to be very small, since in most cases the client simply waits for the server's reply. Minimising the inflation is the key to accuracy here, justifying the MinDiff clock. (ii) The *time-to-hear-again* from that client might be large, hence it may be better to have unique entries for each of the recently Requesting clients following conclusion [Next-Contact] of the analysis section IV, justifying the ROV-MRS policy. However, later in the execution, that client may be displaced in the server's

[1]The interested reader can find additional experiments to those presented here on our homepage http://www.cs.chalmers.se/∼andersg/Clocks/.

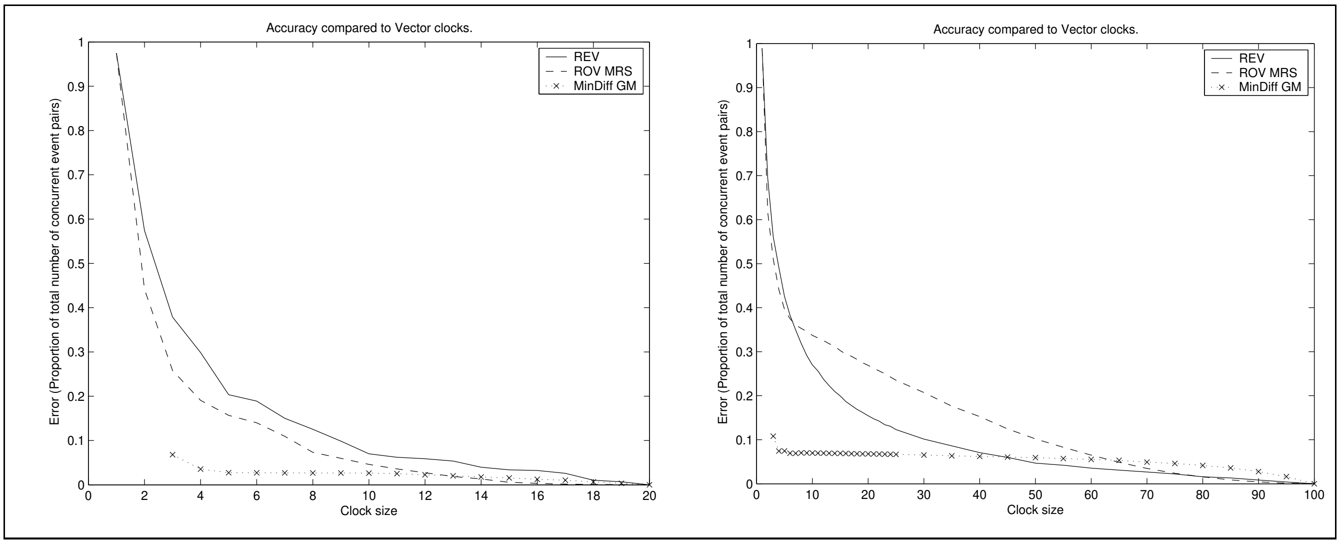

Fig. 7. 20 and 100-process peer-to-peer systems. The local history length of each process is 40 and 100 events. The number of concurrent event pairs is 284 698 and 44 930 478 and the total number of event pairs is 327 645 and 50 486 176, respectively. (Note that clock size is the number of clock entries.)

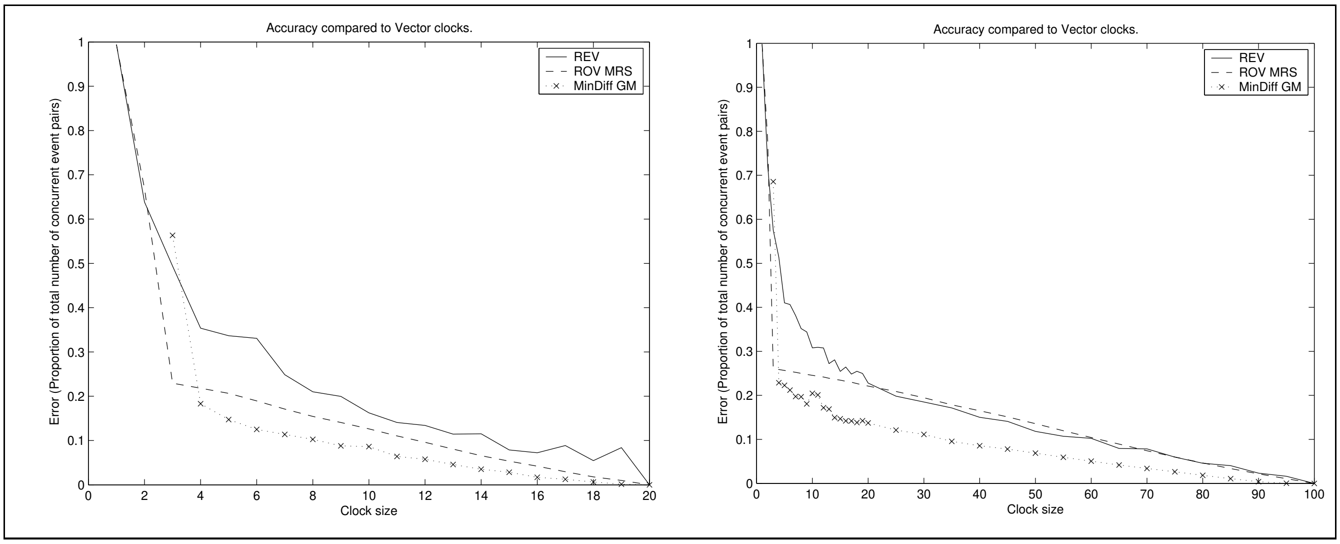

Fig. 8. 1-server-19-client and 1-server-99-client systems. Local history length of each client process is 100 events. The number of concurrent event pairs is 1 032 675 and 28 661 424 and the total number of event pairs is 3 260 181 and 85 850 856, respectively.

clock by new, more recently requesting clients (if there are more than $R-2$ of them). In that case it will have to share the others-entry with all other processes, resulting in inflation of its value or the other processes' values. (In the many servers case and if the nodes form clusters (disjoint or with small overlaps), the method should be appropriate, especially if the cluster size is smaller than or close to R.) (iii) Since the *time-to-hear-again* (from the same client) can be *arbitrarily large* any (even small) inflation can cause *arbitrarily many errors* in any of the algorithms (cf. lemma 4.5). MINDIFF (which uses the minimisation of inflation as a tool) tends to show better performance than REV (which uses the arbitrary mapping as a tool).

• The latter argument, from the perspective of a *client* shows that: When the client inflates the value of another client, both the *time-to-hear-again* and *time-to-hear-back* between the two clients depend on when the next requests from *both* the clients will be issued (they communicate indirectly, via the server). These values can be arbitrarily large, even unbounded if any of the two clients stops issuing requests. When a client inflates the value of the server, the client may have some estimation on the time-to-hear-back (server from client) and time-to-hear-again (client from server) values, depending on whether it knows when it will send its next request.

A general conclusion is that the particular communication patterns play a very significant role in the client-server communication case. An algorithm that could give guarantees would need information about the future, as concluded in section IV. If such information is available or predictable, it may be possible to have even better performance from within applications, by adopting next-contact-aware conditions in the update of the mapping functions, e.g. knowledge or estimation of the request frequency.

VIII. Discussion

Logical clocks have been studied extensively in the distributed computing literature. Still there are aspects of them which need to be discovered to enhance performance and scalability towards satisfying needs of future distributed systems, e.g. in the context of scalable multicast and collaborative applications in peer-to-peer systems. This paper makes significant steps for an in-depth study on the accuracy of vector timestamps with fixed and small number of entries, aiming at scalable solutions for large systems. The work builds on the the work of Torres-Rojas and Ahamad [4], where the notion of plausible clocks and some plausible clock algorithms were introduced.

In particular, our contributions are the following: (i) We introduce the Non-Uniformly Mapped R-Entries Vector (NUREV) clocks, a general class of clocks that extends and includes the R-Entries Vector (REV) clocks algorithm of [4] and the Full Vector clocks. With NUREV clocks each process in the system can use a different mapping between process-ids and clock-entry indices, the idea being that dynamic mappings may allow self-tuning and adaptation to improve the accuracy of the clocks. (ii) We prove that NUREV clocks are plausible. This makes it easier to design new adaptive plausible clock algorithms. (iii) Furthermore, we analyse the ways that these clocks may relate causally independent event pairs. Our analysis resulted in a set of criteria to concentrate on in order to improve performance. (iv) These, in turn, resulted in new adaptive mapping strategies, MinDiff and ROV-MRS, that show very competitive performance for small clock/timestamp sizes, that is, where it matters in practice. The experimental evaluation of the performance of our proposed methods agrees with the conclusions of the analysis part and also shows promising results from the applicability point of view.

Our work points to some new issues that need investigation. One of them is the issue of constant-size representation of mapping functions, for example while our MinDiff clock provides outstanding performance for small timestamps, it is, strictly speaking, not a fix-sized clock. Although from the applicability point of view, as is shown in the paper, this is no obstacle to the performance and scalability in the system, from the intellectual point of view, it can be a challenging issue. Possible directions to investigate it include the use of appropriate approximated mappings (e.g. low-pass filters or polynomials) or the use of smaller sets of mapping functions that can be represented in constant space.

Other important and challenging research issues that follow from this research are (i) to study the performance and accuracy of plausible clocks from an information-theory point of view, (ii) to bound the size of the vector entries and (iii) to consider dynamic group sizes and other varying parameters.

Acknowledgement

We wish to express our thanks to Philippas Tsigas for the informative and fruitful discussions and to Niklas Elmqvist, Ha Hoai Phuong and Håkan Sundell for their feedback on a draft of this paper.

References

[1] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems," *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 1, pp. 63–108, 1998.

[2] Martin Mauve, "Consistency in replicated continuous interactive media," in *Proc. of ACM CSCW'00 Conference on Computer-Supported Cooperative Work*, 2000, pp. 181–190.

[3] Mustaque Ahamad, Phillip W. Hutto, and Ranjit John, "Implementing and programming causal distributed shared memory," in *Proc. of the 11th International Conference on Distributed Computing Systems*. May 1991, pp. 274–281, IEEE Computer Society.

[4] Francisco J. Torres-Rojas and Mustaque Ahamad, "Plausible clocks: Constant size logical clocks for distributed systems," *DISTCOMP: Distributed Computing*, vol. 12, 1999.

[5] Antonio Fernández, Ernesto Jiménez, and Vicente Cholvi, "On the interconnection of causal memory systems," in *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC-00)*. 2000, pp. 163–170, ACM Press.

[6] Friedemann Mattern, "Virtual time and global states of distributed systems," in *Proc. of the International Workshop on Parallel and Distributed Algorithms*, M. Cosnard, Ed. Oct. 1988, Elsevier.

[7] Colin Fidge, "Logical time in distributed computing systems," *Computer*, vol. 24, no. 8, pp. 28–33, Aug. 1991.

[8] Frank B. Schmuck, "The use of efficient broadcast protocols in asynchronous distributed systems," Technical Report TR88-928, Cornell University, Computer Science Department, Aug. 1988.

[9] Bernadette Charron-Bost, "Concerning the size of logical clocks in distributed systems," *Information Processing Letters*, vol. 39, no. 1, pp. 11–16, July 1991.

[10] Roberto Baldoni and Giovanna Melideo, "k-dependency vectors: A scalable causality-tracking protocol," in *Proc. of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2003, pp. 219–226.

[11] Leslie Lamport, "Time, clocks, and the ordering of events in a distributed system," in *Communications of the ACM*, July 1978, pp. 558–565.

[12] Özalp Babaoğlu and Keith Marzullo, "Consistent global states of distributed systems: Fundamental concepts and mechanisms," in *Distributed Systems*, Sape Mullender, Ed., pp. 55–96. Addison-Wesley, 1993.

[13] Francisco J. Torres-Rojas, "Performance evaluation of plausible clocks," *Lecture Notes in Computer Science*, vol. 2150, pp. 476–481, 2001.