Possible and Impossible Vector Clock Sets

Esteban Meneses and Francisco J. Torres-Rojas

Abstract— **It is well known that vector clocks capture perfectly the causality relationship among events in a distributed system. However, there are some interesting properties of vector clocks that are still to be explored. In particular, we are interested in discovering whether there is an efficient procedure for deciding if a given set of vector clocks is contained in some distributed history. We call this the *possible vector clock set problem*.**

Index Terms— **Distributed Computing, Logical Clocks, Vector Clocks.**

I. Introduction

SOME relationships among events happening in real life can be deduced just by comparing the times when they happened. Typically, if event **a** occurred before event **b** we can say that **a** is potentially a cause for **b**. The same happens with events in a distributed system. The general problem here is to find a mechanism such that, given two arbitrary events **a** and **b**, it can be established which the causality relationship between them is or, if it were the case, conclude that both events are *concurrent* [6].

Vector clocks [3], [5] define a technique to determine precisely the causality relationship among events in a distributed system, including the concurrent case. Given a distributed system with *N* sites, each site keeps a *N* entry vector of integers where the *j*-th entry accounts for the quantity of events occurred in site *j* that are known to this site. When event **a** happens at certain site, it is timestamped with the current vector clock of the site. There are a few, but precise, rules for updating local clocks when a message in sent or received. By comparing the vector clocks assigned to any two arbitrary events, it can be precisely determined the causality relation between them. Besides this well known property of vector clocks, there are still a number of interesting questions about vector clocks that are worthy of exploring [8].

Esteban Meneses (esteban.meneses@predisoft.com), Predisoft and *Centro de Investigación en Computación e Informática Avanzada* (*CIenCIA*), Costa Rica. Francisco J. Torres-Rojas (torres@ic-itcr.ac.cr), Costa Rica Institute of Technology and *Centro de Investigación en Computación e Informática Avanzada* (*CIenCIA*), Costa Rica.

In particular, we propose the *possible (or impossible) vector clock set problem*, where, given an arbitrary, finite set of vector clocks, we are to find a distributed history that contains events timestamped with the vector clocks of the set. As we will show, there exist *impossible sets*, for which there are no distributed history that contains them.

In Section II, some basic concepts about vector clocks are reviewed. The problem of possible sets of vector clocks is presented in Section III. Some properties of these logical clocks, useful for the problem at hand, are explained in Section IV. A preliminary approach to finding a distributed history that contains a given set of vector clocks is explored in Section V, and Section VI gives an example of the proposed technique. Finally, conclusions and future work are listed in Section VII.

II. Vector Clocks

Let's assume a distributed system with *N* sites, where all the communications are made through message exchange. There are three kinds of events: internal, **send** and **receive**. The *local history* $\mathcal{H}_i$ for site *i* is the total-ordered sequence of events $\mathcal{H}_i$=$e_{i1}e_{i2}$... that are executed at site *i*. The *global* or *distributed history* $\mathcal{H}$ for the distributed system is the partially ordered set of events occurring at every site in the system.

Leslie Lamport [4] proposed the concept of *logical clocks*, i.e., a mapping between events in a distributed history $\mathcal{H}$ and integer numbers that can be used to detect some of the causal relationships between events. Although they are very easy to implement, *Lamport clocks* cannot capture all the causality information. For example, concurrency between events is poorly detected. These clocks are consistent with causality, but they do not characterize it [6].

Vector clocks [3], [5], [2], [6] consist of a mapping between events in a distributed history $\mathcal{H}$ and integer vectors. Each site *i* keeps its own vector clock V_i with *N* entries, where *N* accounts for the number of sites in the system. Entry $V_i[i]$ maintains the local clock of site *i*, while $V_i[j]$ keeps track of the activity at site *j*, from the point of view of site *i*.

Each time an event occurs at site *i*, its local clock ticks and a vector clock is associated with that event. Also, every message sent in the system is piggybacked with the vector clock corresponding to the **send** event.

Site *i* updates its vector clock obeying the following rules:

- $V_i[j] = 0, 0 \leq j \leq N-1$ - Initial value.
- Each time an internal event occurs: $V_i[i] = V_i[i] + \Delta$ (typically Δ=1)
- When a message with timestamp *T* is received: $V_i[j] = max(V_i[j], T[j]), 0 \leq j \leq N-1$
  $V_i[i] = V_i[i] + \Delta$

Let **a** $\in \mathcal{H}_i$. We say that **a** has timestamp *V*(**a**), where *V*(**a**) is the value of V_i at the instant when **a** was executed.

Vector clocks are compared following these rules:

- $v = w \Leftrightarrow v[j] = w[j], 0 \leq j \leq N-1$
- $v \leq w \Leftrightarrow v[j] \leq w[j], 0 \leq j \leq N-1$
- $v < w \Leftrightarrow v \leq w$ and $\exists$ *j* such that $v[j] < w[j]$
- $v \parallel w \Leftrightarrow \exists$ k such that $v[k] < w[k]$ and $\exists$ *j* such that $v[j] > w[j]$

Mattern showed in [5] that there is an isomorphism between timestamps obtained from a vector clock when events are executed, and the causality relationship among events in $\mathcal{H}$. Thus, vector clocks satisfy the *Strong Clock Condition*, where $\forall$ **a** and **b** $\in \mathcal{H}$:

1) **a** = **b** $\Leftrightarrow$ *V*(**a**) = *V*(**b**)
2) **a** $\rightarrow$ **b** $\Leftrightarrow$ *V*(**a**) $<$ *V*(**b**)
3) **a** $\parallel$ **b** $\Leftrightarrow$ *V*(**a**) $\parallel$ *V*(**b**)

Given two vector clocks *v* and *w*, only one of four cases might happen: *v* = *w*, *v* $\rightarrow$ *w*, *w* $\rightarrow$ *v* or *v* $\parallel$ *w*. Following the comparisons defined above and using the fact that the relationship $\rightarrow$ generates a partial order, a Hasse diagram is obtained when drawing all those relationships [7]. Vector clocks have several interesting properties, some of which have been studied in [2], [8], [9].

Definition 1: If *v* and *w* are two vector clocks, we say that *t* is the *maximum* of them, denoted as *maximum*(*v*,*w*), if these conditions hold:

- *v* $\rightarrow$ *t* and *w* $\rightarrow$ *t*
- It doesn't exist a vector clock z such that $v \rightarrow z \wedge w \rightarrow z \wedge z \rightarrow t$

The definition for *minimum* is analogous to the one given for *maximum*.

It can be easily verified that given vector clocks *v* and *w*, *t* is *maximum*(*v*,*w*) iff *t*[*k*] = *max*(*v*[*k*],*w*[*k*]), $0 \leq k \leq N-1$, and that *u* is *minimum*(*v*,*w*) iff *u*[*k*] = *min*(*v*[*k*],*w*[*k*]), $0 \leq k \leq N-1$.

The next two lemmas are needed for the main result of next section:

Lemma 1: No other site has more updated information about the activity of Site *i* (i.e., a larger value in its *i*-th entry) than Site *i* itself.

Proof: From the rules defined previously, it is evident that site *i* is the only one that increments $V_i[i]$, and the value of this entry in any other site is either zero, or the value sent (directly or indirectly) by site *i*, which in turn could have incremented it since then. ■

Lemma 2: Let **a**,**b** $\in \mathcal{H}$. In particular, let **a** $\in \mathcal{H}_k$, i.e., **a** was executed at site *k*. Then:
a $\rightarrow$ **b** $\Leftrightarrow$ *V*(**a**)[*k*] $\leq$*V*(**b**)[*k*].

Proof: This is a corollary of the isomorphism between vector clocks and events in the distributed history, and the comparison of vector clocks defined previously. Since **a** occurred at site *k* and it is causally before **b**, then, when **b** occurs, the timestamp of site *k* is at least the one **a** had, and vice versa. This result is usually known as the *Simple Strong Clock Condition.* ■

III. The Possible Vector Clock Set Problem

Not every arbitrary set of vector clocks is valid or possible, in the sense that it might be impossible to find a distributed history $\mathcal{H}$ that contains a subset of events such that their vector clocks correspond to the ones in the given set. In other words, certain combinations of vector clocks can not coexist in the same distributed history.

Theorem 1 (Deserted Zone): Let $a_1, a_2, ..., a_n$ be *n* events in $\mathcal{H}$ associated with the *n* vector clocks $v_1, v_2, ..., v_n$, respectively. Also, let the *n* events be concurrent with each other. If *t*=*maximum*($v_1, v_2, ..., v_n$), then, it cannot exist an event **z** $\in \mathcal{H}$ such that $v_i$$\rightarrow$*V*(**z**)$\rightarrow$*t*, for any *i*.

Proof: By contradiction. Let's assume that there exists an event **z** $\in$ *H* such that $v_i$$\rightarrow$*V*(**z**)$\rightarrow$*t* for some *i*. Without loss of generality, let *i* be

1, and let $\mathbf{z}$ happen on site k, i.e., $\mathbf{z} \in \mathcal{H}_k$. Now, using Lemma 1, we have $v_1[k] < V(\mathbf{z})[k]$. Given that $V(\mathbf{z}) \rightarrow t$, then $v_1[k] < V(\mathbf{z})[k] \leq t[k] = max(v_1[k], v_2[k], ..., v_n[k])$, which means definitively that $V(\mathbf{z})[k] \leq v_p[k]$, for some $p \in \{2,3,...,n\}$. In virtue of Lemma 2 and the isomorphism between vector clocks and events, $V(\mathbf{z})[k] \leq v_p[k]$ implies that $\mathbf{z} \rightarrow \mathbf{a}_p$, but by hypothesis $\mathbf{a}_1 \rightarrow \mathbf{z}$, and, therefore, $\mathbf{a}_1 \rightarrow \mathbf{a}_p$, which contradicts the fact that $\mathbf{a}_1$ and $\mathbf{a}_p$ are concurrent. Hence, $\mathbf{z}$ cannot exist. ■

Let's consider the case with *n*=2, where $\mathbf{a}_1$ and $\mathbf{a}_2$ are two concurrent events with vector clocks v_1 and v_2, respectively. If *t*=*maximum*(v_1, v_2), then v_1, v_2 and *t* define a kind of "deserted zone", where no event might occur. Hence, Theorem 1 affirms that there cannot be any event $\mathbf{z}$ in the same distributed history that contains concurrent events $\mathbf{a}_1$ and $\mathbf{a}_2$, with its vector clock lying in the deserted zone induced by $\mathbf{a}_1$ and $\mathbf{a}_2$. Notice, however, that there could exist many integer vectors which have this property. Nevertheless, none of them can be associated to any event in the same distributed history.

From the result of Theorem 1, we realize the existence of certain *impossible* sets of vector clocks. For instance, the set {<100, 0, 100>, <0, 100, 0>, <50, 100, 0>} is *impossible*. Thus, if presented with a set of vector clocks, it makes sense to ask ourselves whether or not this set of vector clocks is a *possible* subset of timestamps seen in a distributed history. This question is what we called the *possible vector clock set problem.*

In its more general form, this problem can be stated as:

Given a set of vector timestamps, decide whether or not there is a distributed history containing them. There is not information available about the site where each vector clock is supposed to have occurred.

There could be a version of the problem where the sites corresponding to certain vector clocks in the set are known, while other vector clocks are free to be assigned to any site:

Given a set of vector timestamps, some of them assigned to a specific sites, decide whether or not there is a distributed history containing them all.

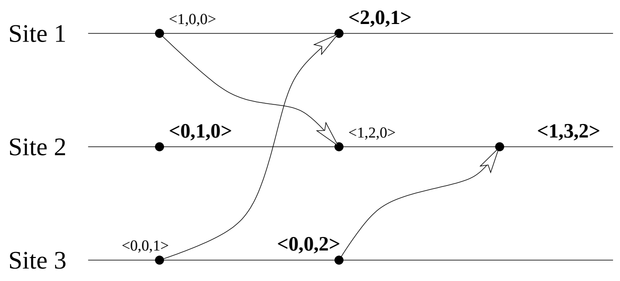

Fig. 1. Underlying distributed history for set A

Finally, the, apparently, "softest" version of the problem pins down all the vector clocks to known sites:

Given a set of vector timestamps, each one assigned to a specific site, decide whether or not there is a distributed history containing them.

As an example of the latest statement of the problem, consider the vector clock set $\mathcal{A}$ over a distributed system with 3 sites:

$\mathcal{A}$ = { [<2,0,1>, *site 1*],
[<0,1,0>, *site 2*],
[<1,3,2>, *site 2*],
[<0,0,2>, *site 3*] }

To demonstrate that $\mathcal{A}$ is a *possible* set, we just have to show a distributed history $\mathcal{H}$ which contains every element of $\mathcal{A}$ as a timestamp. In this case, $\mathcal{A}$ is possible, because there are multiple distributed histories that satisfy the requirement. Figure 1 shows an example of such a distributed history (the timestamps from $\mathcal{A}$ are written in bold).

On the other hand, the vector clock set $\mathcal{B}$ is *impossible*:

$\mathcal{B}$ = { [<2,0,1>, *site 1*],
[<0,1,0>, *site 2*],
[<0,1,1>, *site 2*],
[<1,3,2>, *site 2*],
[<0,0,2>, *site 3*] }

To prove that $\mathcal{B}$ is an impossible set, consider the concurrent vector clocks v_1=$< 0, 1, 0 >$ and v_2=$< 2, 0, 1 >$, both in $\mathcal{B}$. Let's say that these vector clocks correspond to events $\mathbf{a}_1$ and $\mathbf{a}_2$. Following Theorem 1, if we compute the *maximum* between v_1 and v_2, we obtain *t*=$< 2, 1, 1 >$. Then, there can not exist an event $\mathbf{z}$ in the same distributed history as $\mathbf{a}_1$ and $\mathbf{a}_2$, such that its associated vector clock $V(\mathbf{z})$

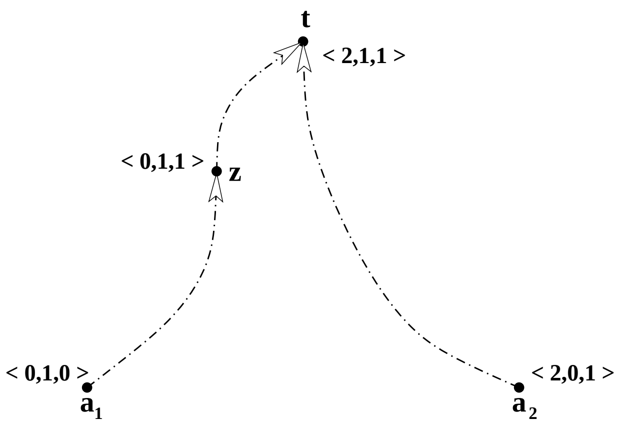

Fig. 2. Impossible set due to deserted zone theorem

has the property: $v_1 < V(\mathbf{z}) < t$ or $v_2 < V(\mathbf{z}) < t$. However, vector clock $< 0,1,1 >$ in $\mathcal{B}$ happens to be between $< 0,1,0 >$ and t=$< 2,1,1 >$, which renders set $\mathcal{B}$ as impossible. Figure 2 shows a graphic representation of this particular situation.

IV. Intermediate Results

In this section, we consider certain properties of vector clocks and some impossibility results, which are useful to explore the problem of possible vector clock sets.

Lemma 3: Let timestamp *v* correspond to a event in $\mathcal{H}_i$. Then, for every $j \neq i$ and $v[j] \neq 0$, the $v[j]$-th event of $\mathcal{H}_j$ is a **send**.

Proof: Left to the reader. ■

Thus, if every timestamp in set $\mathcal{S}$ is associated with some site, then, just by checking these timestamps, a *necessary* set of **send** events in the distributed history can be deduced. However, this resultant group of **send** events might not be sufficient for building the underlying distributed history. Let v_j and v_k be two timestamps corresponding to events happening in sites j and k, respectively. Also, let $v_j[i] = v_k[i] = x$ ($i \neq j$ and $i \neq k$). As Lemma 3 established, the x-th event in Site *i* must be a **send**. In order to construct timestamps v_j and v_k, this **send** must reach, directly or indirectly, sites j and k. Since our model does not allow multicasting, the x-th event in Site *i* must be directed to, let's say, Site *j*, and from there to Site k with an extra **send**. We call this kind of **send** a *deferred send*.

Lemma 4: Given a vector clock set $\mathcal{S}$ where each timestamp is associated with some site, if two timestamps (occurring in different sites) coincide in their *i*-th entry and neither of them occurred in Site *i*, a *deferred* **send** is required in the underlying distributed history.

Proof: It is straightforward given the previous discussion. ■

Although the aim of a *deferred* **send** is to provide time information indirectly, this functionality could be fulfilled by one of the regular **send**s derived from Lemma 3. Thus, it could be the case in an underlying distributed history, that some *deferred* **send** is collapsed with another **send** event.

The following lemma proves the existence of impossible vector clock sets with just one element:

Lemma 5: The vector clock set $\mathcal{S}$=$\{< 1,1,1 >\}$ is impossible.

Proof: Let's suppose, without loss of generality, that timestamp $< 1,1,1 >$ occurred at Site *1*. So, entry 1 is justified by being the local stamp, which means that this vector clock corresponds to the first event of this site. Lemma 3 indicates that the first event of both Site *2* and Site *3* must be **send** events. Notice that the first event of Site *1* must be a **receive**. Assume, for instance, that it receives a message from Site *2*. Then, its third entry can not be 1, since there is "no time" for such information to be transmitted. Note that a deferred **send** is not sufficient either, because if Site *3* sends a message to Site *2*, then the first event of Site *2* should be a **receive** event, being unable to send the required information to Site *1*. ■

Theorem 2: Consider a distributed system with N sites. Let $\mathcal{S} = \{v_1, v_2, ..., v_k\}$ be a set with $k \leq$ N vector clocks concurrent to each other and let t = *maximum*(v_1,v_2,...,v_k). There cannot be v_i ($1 \leq i \leq k$) such that $v_i[j] < t[j]$ for all *j* = 1,2,...,N.

Proof: By contradiction. Assume that such v_i exists, and that it occurred at Site *p*. So, by assumption, $v_i[p] < t[p]$. Since *t* is the *maximum* of the vector set, there must exist some v_j, such that $v_j[p] = t[p]$. The only way to explain this value of entry $v_j[p]$ is via a chain of messages starting with a **send** event **a** at Site *p*. Let $V(\mathbf{a})$ be the vector clock of **a**. Clearly, $V(\mathbf{a}) < v_j$, because event **a** causally precedes the event with timestamp v_j. Now, **a** and the event with timestamp v_i come from the same site *p*, and since $v_i[p] < V(\mathbf{a})[p]$, we have $v_i < V(\mathbf{a}) < v_j$, which contradicts the premise that all vector clocks are concurrent. ■

V. An Exhaustive Approach

By using an exhaustive strategy, it can be possible to decide whether or not there is an underlying distributed history for some vector clock set. We analyze a solution for the "softest" version, where each timestamp is associated with some specific site. Strategies for the other two versions of the problem can be obtained by extending this one.

We are looking for a set $\mathcal{A}$ of pairs of events of the form $< Site$: **send**, $Site$: **receive** $>$, where **send** and **receive** indicate the relative positions of these events in each local history. In fact, any distributed history can be characterized by the number of sites, the number of events in each site and set $\mathcal{A}$. Thus, the algorithm basically tests a series of alternatives for the underlying history, until a set $\mathcal{A}$ that produces all required timestamps in the distributed history is obtained.

Given a vector clock set $\mathcal{S} = \{v_1, v_2, ..., v_m\}$ with *m* timestamps, each one associated to some site, the following steps will find an underlying distributed history:

Step 1: The number of sites *N* is equal to the length of every timestamp in $\mathcal{S}$.

Step 2: Let *t* = *maximum*$(v_1, v_2, ..., v_m)$, then $t[i]$ is the number of events in site *i*, for *i*=1,2,...,N.

Step 3: Obtain all the necessary **send** events set $\mathcal{R}$, using Lemma 3.

Step 4: Obtain all the *deferred* **send** events set $\mathcal{D}$, using Lemma 4.

Step 5: Associate every element of $\mathcal{R}$ and $\mathcal{D}$ with appropriate **receive** events, computing set $\mathcal{A}$

If timestamp *v* appears in Site *i*, it is known that it was the $v[i]$-th event in this site. Let's call **a** this event. Now, we have to justify all the other entries in *v*. Lemmas 3 and 4 establish the quantity of **send** events, being deferred or not, that must reach site *i* in order to justify timestamp *v* for event **a**.

Definition 2: The *receive block* of event **a** includes the preceding events of **a** in the same site and **a** itself, which can be used to receive information for justify timestamp of event **a**.

Figure 3 shows the *receive blocks* for two different events **a** and **b**, occurring in the same site *i*, and whose timestamps must be justified. Notice that if these timestamps share the j-th entry (with $j \neq i$), then j-th must not be justified for **a** in the receive

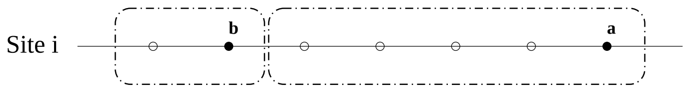

Fig. 3. Receive blocks for events **b** and **a**, respectively

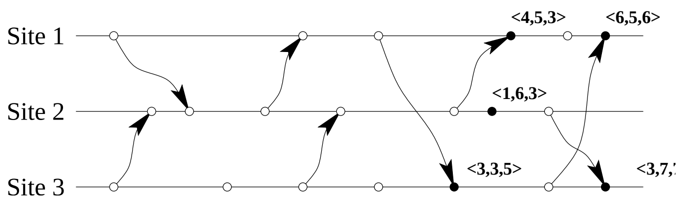

Fig. 4. Original distributed history

block for **a**. This entry, on the contrary, has to be kept and not overridden by a message with large j-th entry in its timestamp.

Finally, we have all the pieces to solve the "softest" version of the problem. Notice that the difference between versions of the problem is the fact that site information is inexistent, incomplete or complete. When information about sites is inexistent, this is a particular case of being incomplete. Then, if we have an algorithm for detecting possibility or impossibility when site information is complete, we can iterate the algorithm over the options that are created by associate the timestamps without site information with a particular one. Those possibilities could be huge, but finite.

VI. Example: Building underlying distributed history

This section gives an example of the exhaustive approach introduced in the previous section. Suppose we are given a set $\mathcal{S}$ of timestamps taken from the distributed history in Figure 4 (obviously, we do not know this history):

$\mathcal{S}$ = { [<4,5,3>, *site 1*],
[<6,5,6>, *site 1*],
[<1,6,3>, *site 2*],
[<3,3,5>, *site 3*],
[<3,7,7>, *site 3*] }

Let's call the events associated with the previous timestamps **a**,**b**,**c**,**d** and **e**, respectively.

Now, the algorithm from section V will be applied to verify if this set has at least one underlying or *witness* distributed history. First of all, it is known that N=3, so the system have 3 sites where we are to accommodate all the required **send** and **receive** events. Secondly, the *maximum* for all timestamps in $\mathcal{S}$ is $t = max(< 4,5,3 >, < 6,5,6 >, < 1,6,3 >, < 3,3,5 >, < 3,7,7 >) =< 6,7,7 >$, which means that there are 6 events in Site *1*, 7 in Site *2* and 7 in Site *3*.

Using Lemma 3, the set $\mathcal{R}$ is obtained. Thus, local events {1,3} in Site *1* must be **send**s. Also, events {3,5,7} in Site *2* and events {3,6} in Site *3*. The next table summarizes these results:

Event	Site	Number	Blocks
s_1	1	1	**c**
s_2	1	3	**d,e**
s_3	2	3	**d**
s_4	2	5	**a,b**
s_5	2	7	**e**
s_6	3	3	**a,c**
s_7	3	6	**b**

This table shows the name of the event, the site where it occurred, the local number of event and finally the blocks which it must reach. This last column is very important, since the *raison d'être* for each **send** event is to reach some block and thus provide some event with the required components for its timestamp.

Intuitively, one could think that if some **send** event has two or more blocks in its last column, then a deferred **send** is required. However, this is not necessarily the case. Events s_1,s_3,s_5 and s_7 have just one reaching block. On the other side, events s_2 and s_4 have two reaching blocks. Nevertheless, in each case, the two events occurred in the same site, so it is sufficient for the **send** to reach the first of the two events. Moreover, it is necessary to avoid another **send** event to reach these sites with more updated information. Then, for instance, s_2 have just to reach block **d** and this local timestamp will be propagated into block **e**. But, one must avoid any other **send** event coming from Site *1* and occurring after s_2.

The last argument cannot be applied to s_6, so it must be considered inserting a deferred **send** which reaches block **a** and then block **c** or vice versa. Therefore, Lemma 4 has been taken into account and set $\mathcal{D}$ has been built.

Figure 5 depicts a visual scheme of the last situation. It shows all the events with timestamps in $\mathcal{S}$ and the required **send** events. Note that the *deferred*

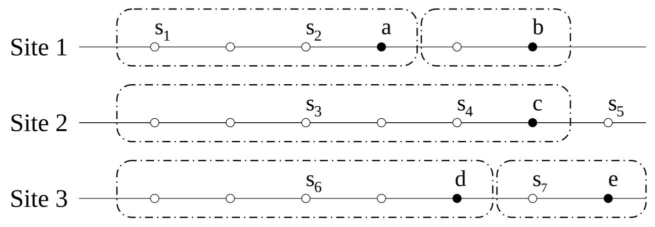

Fig. 5. Scheme for building the witness distributed history

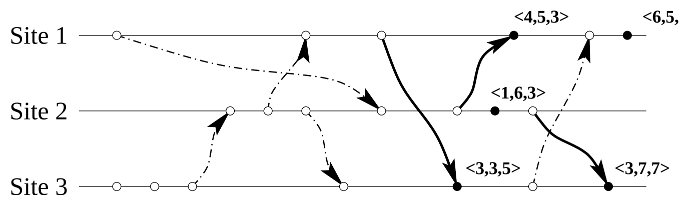

Fig. 6. Witness distributed history

send is not shown, because its event number must no be fixed *a priori*.

Finally, applying an exhaustive assignment for **send** events to its reaching blocks and testing if the required timestamps are generated, we verify the possibility of $\mathcal{S}$. One of the witness distributed histories appears in figure 6. Note that it is different from the original history of figure 4. In fact, there could exist many underlying distributed histories for a given set $\mathcal{S}$. However this new history keeps some of the original messages, which are depicted with solid lines. The new messages were drawn with dotted lines.

As it can be appreciated, the *deferred* **send** is located in Site *2* as the second event. Nevertheless, it is not necessary, because all the information it was supposed to provide, was carried by **send** event s_4.

VII. Conclusions and Future Work

Vector clocks have been a helpful tool in determining the causality relations among events in a distributed system. Knowing their properties will ensure a more reliable time stamping system, as we can check additional requirements. For instance, if we had an efficient procedure for determining if some set of vector clocks is possible or not, then we can reject a report of timestamps if they constitute an impossible or falsified set.

We are interested in classifying the process of deciding whether some vector clock set is possible or impossible. This problem is clearly soluble, at least using an exhaustive strategy. But, it would be important to verify if this problem is P or NP-complete. Thus, future work is focused on discovering the nature of the possible set problem on vector clocks. It must be defined if it has a polynomial time solution or if it can only be solved by an exhaustive approach. Besides, there are several interesting properties and theorems about possible and impossible sets that deserve to be explored (e.g., closure properties, special cases, geometric interpretations, etc.).

REFERENCES

[1] Ahamad, M. et al. *Causal memory: definitions, implementation and programming*. Distributed Computing, 1995.

[2] Baldoni, R. and Raynal, M. *Fundamentals of Distributed Systems: A Practical Tour of Vector Clocks Systems*. IEEE Distributed Systems Online, February, 2002.

[3] Fidge, C. *Logical Time in Distributed Computing Systems*. Computer, Vol 24(8), August, 1991.

[4] Lamport, L. *Time, Clocks, and the Ordering of Events in a Distributed System*. Communications of the ACM, Vol 21(7):558-565, July, 1978.

[5] Mattern, F. *Virtual Time and Global States of Distributed Systems*. Proceedings of the International Workshop on Parallel and Distributed Algorithms, 215-226, 1989.

[6] Schwarz, R. and Mattern, F. *Detecting Causal Relationships in Distributed Systems: In Search of the Holy Grail*. Distributed Computing, 1994.

[7] Torres-Rojas, F. *Partially Ordered Sets and Logical Clocks for Distributed Systems*. Proceedings of Conferencia Latinoamericana en Informática, 2000.

[8] Torres-Rojas, F. and Meneses, E. *Algunas Propiedades Interesantes de los Relojes Vectoriales*. Proceedings of Jornadas Chilenas de Computación, 2003.

[9] Yang, Z. and Marsland T.A. *Global States and Time in Distributed Systems*. IEEE Computer Society Press, 1994.