

## A NEW ALGORITHM TO IMPLEMENT CAUSAL ORDERING

André SCHIPER, Jorge EGGLI, Alain SANDOZ

Ecole Polytechnique Fédérale de Lausanne  
Département d'Informatique  
CH-1015 Lausanne, Switzerland  
schiper@elma.epfl.ch.bitnet

### Abstract

This paper presents a new algorithm to implement causal ordering. Causal ordering was first proposed in the ISIS system developed at Cornell University. The interest of causal ordering in a distributed system is that it is cheaper to realize than total ordering. The implementation of causal ordering proposed in this paper uses logical clocks of Mattern-Fidge (which define a partial order between events in a distributed system) and presents two advantages over the implementation in ISIS: (1) the information added to messages to ensure causal ordering is bounded by the number of sites in the system, and (2) no special protocol is needed to dispose of this added information when it has become useless. The implementation of ISIS presents however advantages in the case of site failures.

Keywords: distributed algorithms, ordering.

### 1. Introduction

The notion of global time does not exist in a distributed system. Each site has its own clock, and it is impossible to order two events  $E_1$  and  $E_2$  occurring on different sites of the system unless they communicate. It is however often necessary to order events in a distributed system.

One possible construction of a total ordering of events in a distributed system is described in [Lamport 78]. It is built using logical clocks defined in the same paper. To progress however, the algorithm requires each site to have received at least one message from every other site in the system, which means systematic acknowledgements of messages.

ie for which there  
algorithm has the  
ion (ie no clean  
which allows  
the cycle to abort  
e propagated,

by the large  
Several are  
with operational  
extensively tested,  
tedious derivations  
n made, not to  
e those aspects of  
ts developed for  
tion of the

ty Based Probe  
ng of the 7th

ified Priority Based  
EE Trans on Soft

Computing Surveys,

clock Detection  
buted Computing.

Detection



There does however exist a weaker ordering than total ordering: causal ordering. The implementation of such an ordering needs less message exchanging (no acknowledgements like the ones above are needed), and can prove to be sufficient in some applications.

This causal ordering should not be confused with the causality in the definition of logical clocks, which we call here "causal timestamping". Let us give an example enabling us to distinguish causal ordering from causal timestamping. Suppose an event  $\text{SEND}(M_1)$ , corresponding to the site  $S_1$  sending message  $M_1$ , and timestamped with logical time  $T_1$ . Suppose then a second event  $\text{SEND}(M_2)$ , with timestamp  $T_2$ , occurring on site  $S_2$  after  $S_2$  has received message  $M_1$ . Lamport's logical clocks ensure that  $T_1 < T_2$ . Thanks to this "causal timestamping", event  $\text{SEND}(M_1)$  precedes event  $\text{SEND}(M_2)$  for every site in the system which will ever know of these events. This does not say anything about the order in which the messages  $M_1$  and  $M_2$  arrive at any given site in the system. Causal ordering of the events  $\text{SEND}(M_1)$  and  $\text{SEND}(M_2)$  means that every recipient of both  $M_1$  and  $M_2$  receives message  $M_1$  before message  $M_2$ . This is not automatically the case in a distributed system, as shown in figure 1, where site  $S_3$  gets message  $M_2$  before message  $M_1$ , even though event  $\text{SEND}(M_1)$  occurs before event  $\text{SEND}(M_2)$ .

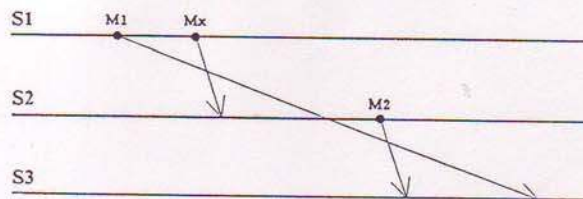


Figure 1. An example of the violation of causal ordering.

Causal ordering is described in [Birman 87] and has been implemented in the ISIS system developed at Cornell University [Birman 88a]. The implementation of causal ordering which we present here differs however from that of ISIS. It presents two advantages: (1) the information added to messages to ensure causal ordering of events is bounded (in the sense defined in section 4.2) by the number of sites in the system, and (2) the implementation does not require any complicated algorithm to clean up this additional information. The implementation of ISIS presents however advantages in the case of site failures. Causal ordering is also achieved through the *conversation abstraction* [Peterson 87]; this implementation however, uses explicit "send before" relations between messages, which is not the case in ours. The rest of the paper is organized in the following way: in section 2, we formally define causal ordering and show the usefulness of this notion. In section 3, we briefly present the idea of the implementation of causal ordering in ISIS. Finally in section 4, we develop a new algorithm to implement causal ordering.

## 2. Causal

Causal ord  
we classifi  
we shall s  
R, iff any

1.  $E_1$  a

2.  $E_1$  o  
the

With the  
and  $E_2 = S$

$E_1^c$

To illustr  
on every  
the replic  
a token.  
the local  
figure 2)

In the  
Preceden  
 $\text{SEND}(W$   
causal or  
in which  
ensures g

It is imp  
total, bu  
independ  
of this c  
updates  
then  $X_j$ ,  
of even  
expensiv  
impleme



## 2. Causal ordering of events

Causal ordering is linked to the relation "happened before" between events, noted " $\rightarrow$ ", which we classically define as the transitive closure of the relation  $R$  described below (to simplify, we shall speak of sites rather than processes). Two events  $E_1$  and  $E_2$  are related according to  $R$ , iff any of the following two conditions is true:

1.  $E_1$  and  $E_2$  are two events occurring on the same site,  $E_1$  before  $E_2$ ;
2.  $E_1$  corresponds to the sending of a message  $M$  from one site, and  $E_2$  corresponds to the reception of the same message  $M$  on any other site.

With the relation  $\rightarrow$ , we can formally define the causal ordering of two events  $E_1 = \text{SEND}(M_1)$  and  $E_2 = \text{SEND}(M_2)$ , noted  $E_1^c \rightarrow E_2$ , as follows:

$E_1^c \rightarrow E_2$  iff (if  $E_1 \rightarrow E_2$ , then any recipient of both  $M_1$  and  $M_2$  receives message  $M_1$  before message  $M_2$ )

To illustrate the usefulness of causal ordering, consider the handling of some replicated data on every site of a distributed system [Joseph 86, Birman 88b]. Each site controls one copy of the replicated data and can update it. To ensure mutual exclusion of updates, let's introduce a token. The site in possession of the token can update the data. Every write operation  $W_i$  on the local copy performed by a site  $S$  is immediately broadcasted to every other site (see figure 2).

In the example of figure 2, we have  $\text{SEND}(W_1) \rightarrow \text{SEND}(W_2) \rightarrow \text{SEND}(W_3) \rightarrow \text{SEND}(W_4)$ . Precedence of  $\text{SEND}(W_2)$  over  $\text{SEND}(W_3)$  is ensured by the sending of the token, since (1)  $\text{SEND}(W_2) \rightarrow \text{SEND}(\text{token})$ , and (2) reception of the token happens before  $\text{SEND}(W_3)$ . The causal ordering ensures that every site receives the updates in the same order (i.e. the order in which they happened initially). So every site updates its local copy in that order, which ensures global consistency of the set of copies.

It is important to realize that this ordering between events in the distributed system is not a total, but only a partial ordering. To see this, just consider a second replicated data, modified independently from the first one, and controlled by another token. Let's note  $X_j$  the updates of this data. Causal ordering of the events  $\text{SEND}(X_j)$ , ensures again that every site sees these updates in the same order. However, one site  $S$  may well receive first some operation  $W_i$  and then  $X_j$ , whereas a second site  $S'$  might receive  $X_j$  before  $W_i$ . This shows that causal ordering of events is weaker than total ordering. Construction of a total ordering is however more expensive to achieve in terms of number of messages exchanged. The low cost of the implementation of causal ordering makes it an interesting tool for the development of



distributed applications. Note that in the example above, if  $W$  and  $X$  are independent, there is no need for causal ordering of  $SEND(W_i)$  and  $SEND(X_j)$ .

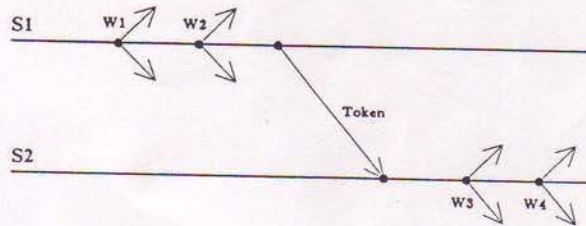


Figure 2. Handling of replicated data using causal ordering.

### 3. Implementation of causal ordering in ISIS

The implementation of causal ordering in ISIS is described in [Birman 87]. However, ISIS implements causal ordering together with atomic broadcasts (atomic broadcasts ensure that a broadcasted message is received by all sites that do not fail, or by none). For clarity, we shall only be interested here in the realization of causal ordering. The idea is the following: every message  $M$  carries along with itself every other message sent before  $M$  it might know of. To achieve this, every site  $S$  handles a buffer (noted  $BUFF\_S$ ) which contains, in their order of emission, every message received or sent by  $S$  (that is, every message preceding any future message emitted by  $S$ ). Sending a message  $M$  from  $S$  to any site  $S'$  will require the following actions: message  $M$  is first inserted into buffer  $BUFF\_S$ , a packet  $P$  is then built containing all the messages in  $BUFF\_S$ , and finally this packet is sent to the destination site  $S'$  of  $M$ . When it arrives at  $S'$ , the following actions are executed for every message in  $P$ : (1) if the message is already in buffer  $BUFF\_S'$  (every message is given a unique id), it has already arrived at  $S'$  and is ignored. Else the message is inserted in  $BUFF\_S'$ ; (2) every message in the packet, of which  $S'$  is the destination site, is delivered to  $S'$  in the correct order.

As an example, consider figure 1. The packet sent from site  $S_1$  to  $S_2$  (resulting from the emission of message  $M_x$ ) contains, in order, messages  $M_1$  and  $M_x$ . The packet sent from  $S_2$  to  $S_3$  (resulting from the emission of message  $M_2$ ) contains messages  $M_1$ ,  $M_x$ , and  $M_2$  (transmission of  $M_x$  is not necessary, but does take place if the algorithm described in [Birman 87] is respected). So message  $M_1$  is carried from site  $S_1$  to  $S_3$  over two different paths:  $\langle S_1, S_3 \rangle$  and  $\langle S_1, S_2, S_3 \rangle$ . In this way, site  $S_3$  will always receive message  $M_1$  before message  $M_2$ .

The algorithm, as described here, still has one major drawback: the information contained in  $BUFF\_S$  increases indefinitely. Some protocol must be added to retrieve obsolete messages from the buffers. The simplified idea is the following: periodically each site  $S$  independently

builds a r  
broadcaste  
S along w  
acknowled  
messages  
sent from  
identified  
before it  
to S any  
from its c  
another s  
the replic

### 4. Another

#### 4.1. Some

The basic  
every me  
M arrives  
yes, mess  
message  
question:  
arrives at

If we cor

Propos  
tim  
tha

Proof: c  
follows  
violated,  
greater t  
which co

However  
timestam  
does not  
necessary



builds a request packet  $P$  containing the ids of messages in its buffer  $BUFF\_S$ . Packet  $P$  is broadcasted to every other site. When some site  $S'$  receives the packet, it notes the source site  $S$  along with the identifiers (the corresponding messages must not be sent to  $S$  any more!) and acknowledges back to  $S$ . When  $S$  has received an acknowledgement from every site, the messages identified in packet  $P$  can be deleted from  $BUFF\_S$ . This is because, if messages sent from  $S'$  to  $S$  are received in the order of emission, every message that could have been identified in  $P$  and nevertheless sent from  $S'$  to  $S$  meanwhile, will have been received by  $S$  before it receives the acknowledgement; afterwards, these messages will not be sent from  $S'$  to  $S$  any more. Note that the protocol initialized by  $S$  does not allow  $S'$  to delete messages from its own buffer: some message  $M$  identified in  $P$  could still be on the way to  $S'$ , sent by another site  $S''$ . After deleting message  $M$  from  $BUFF\_S'$ ,  $S'$  would not be able to recognize the replicated message  $M$ .

#### 4. Another algorithm to implement causal ordering

##### 4.1. Some reflexions on the violation of causal ordering

The basic idea of our algorithm is the following. Rather than carrying around with a message every message which precedes it, let's try to answer the following question when a message  $M$  arrives at a site  $S$ : will any message preceding  $M$  arrive at  $S$  in the future? If the answer is yes, message  $M$  must not be delivered immediately. It will only be delivered to  $S$  when every message causally preceding  $M$  has arrived. For the moment, let's try to answer an easier question: is it possible to know that the causal ordering has been transgressed when a message arrives at a site?

If we consider Lamport's logical clocks, we can state the following proposition:

*Proposition 1: if the causal order has been violated, then there exists a message  $M$ , timestamped  $T(M)$ , which arrives at destination  $S$  when the local time  $T(S)$  is greater than  $T(M)$ .*

*Proof:* consider two messages  $M_1$  and  $M_2$  sent to  $S$ , such that  $SEND(M_1) \rightarrow SEND(M_2)$ . It follows from the definitions that  $T(M_1) < T(M_2)$ . Suppose the causal ordering has been violated, i.e.  $M_2$  arrives at  $S$  before  $M_1$ . After delivery of  $M_2$ , the logical time at  $S$  becomes greater than  $T(M_2)$ :  $T(M_2) < T(S)$ . Therefore, when  $M_1$  arrives at  $S$ ,  $T(S)$  is greater than  $T(M_1)$ , which completes the proof.

However, the converse of Proposition 1 is not true, as shown in figure 3 where a message  $M$ , timestamped  $T(M)$ , arrives at site  $S$  which has logical time  $T(S)$  such that  $T(M) < T(S)$ . This does not mean that the causal order has been violated, which shows that  $T(M) < T(S)$  is a necessary, but not sufficient, condition for causal ordering violation.



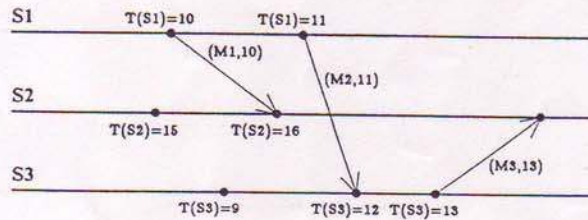


Figure 3. Transgression of causality cannot be detected with Lamport's logical clocks.

So there is no way of answering our second question about causality violation knowing only  $T(S)$  and  $T(M)$ . The problem with Lamport's logical clocks is that they define a total order, whereas there exists only a partial ordering of the events in a distributed system.

A logical clock defining a weaker, partial order is the tool which will be sufficient to infer that the causal ordering was transgressed. This logical clock was recently proposed in [Mattern 89] and [Fidge 88]. For the sake of clarity, let's rapidly recall the principle. The logical time is defined by a vector of length  $N$ , where  $N$  is the number of sites in the system. We will note this logical time  $VT$  (vector time),  $VT(S)$  for the logical time on site  $S$ , and  $VT(M)$  for the timestamp of message  $M$ . The logical time of a site evolves in the following way (see figure 4):

- when a local event occurs at site  $S_i$ , the  $i^{th}$  entry to the vector  $VT(S_i)$  is incremented by one:  $VT(S_i)[i] := VT(S_i)[i] + 1$ .
- when  $S_i$  receives a message  $M$ , timestamped  $VT(M)$ , the rule states:
  - for  $j=i$ ,  $VT(S_i)[j] := VT(S_i)[j] + 1$ ;
  - for  $j \neq i$ ,  $VT(S_i)[j] := \max(VT(S_i)[j], VT(M)[j])$ .

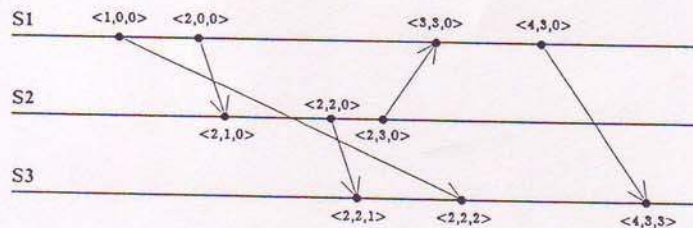


Figure 4. Mattern-Fidge logical clocks.

We also define the ordering relation " $<$ " between logical vector times as follows:  $VT_1 < VT_2$  iff  $VT_1[i] \leq VT_2[i]$ , for all  $i$ . This relation is trivially reflexive, antisymmetric and transitive. Having defined relation  $<$ , it is possible to show that, given two events  $E_1$  and  $E_2$ , then  $E_1 \rightarrow E_2$  iff

$VT(E_1) < VT(E_2)$   
S. In other words,  $VT(E_1) < VT(E_2)$  implies  $E_1 \rightarrow E_2$ .

The logical clock is a proposition.

Proposition: A message  $M$  is received at site  $S_i$  iff  $VT(M) < VT(S_i)$ .

Proving this proposition leads to the proof of the need for a logical clock.

Lemma: A message  $M$  is received at site  $S_i$  iff  $VT(M) < VT(S_i)$ .

Using this lemma, we can prove the violation of causality.

Proof: Suppose  $E_1 \rightarrow E_2$  but  $VT(E_1) \geq VT(E_2)$ . (recall  $S_i$  sends  $M$  at  $E_1$  and receives  $M$  at  $E_2$ ).

4.2. The logical clock.

We are now describing the algorithm which will buffer a message in size  $i$  in the system. The entries of the algorithm are ordered as follows:

- the first entry is the message  $M$ .
- the second entry is the logical time  $VT(M)$ .
- the third entry is the site  $S_i$  which sent  $M$ .



$VT(E_1) < VT(E_2)$ , where  $VT(E)$  is the value of  $VT(S)$  just after occurrence of event  $E$  on site  $S$ . In other words, events  $E_1$  and  $E_2$  are concurrent iff  $\text{not}(VT(E_1) < VT(E_2))$  and  $\text{not}(VT(E_2) < VT(E_1))$ .

The logical time in the system being so defined, we now proceed to prove the following proposition:

*Proposition 2: the causal ordering of events in the system is violated iff there exists a message  $M$ , timestamped  $VT(M)$ , which arrives at destination  $S$  when local time  $VT(S)$  is such that  $VT(M) < VT(S)$ .*

Proving the implication "causal ordering violation  $\Rightarrow$  there exists  $M$ ,  $VT(M) < VT(S)$ " is similar to the proof given above concerning Lamport's logical clocks. To prove the converse, we need the following lemma (the proof is given in [Schiper 89]):

*Lemma 1: consider an event  $E_1$ , timestamped  $VT(E_1)$ , occurring at site  $S_i$ . For every event  $E_2$  such that  $VT(E_1)[i] \leq VT(E_2)[i]$ ,  $E_1 \rightarrow E_2$  is true.*

Using lemma 1, we can now prove "there exists  $M$ ,  $VT(M) < VT(S) \Rightarrow$  causal ordering violation".

*Proof:* Suppose a message  $M$  was sent to site  $S$  by site  $S_i$ . If  $VT(M) < VT(S)$ , then in particular,  $VT(M)[i] \leq VT(S)[i]$ . Consider  $M'$  the message which made  $VT(S)[i]$  take its current value (recall  $S \neq S_i$ ). Then  $VT(M')[i] = VT(S)[i]$ . By lemma 1, it follows from  $VT(M)[i] \leq VT(M')[i]$  that  $\text{SEND}(M) \rightarrow \text{SEND}(M')$ . Since  $M'$  arrived at  $S$  before  $M$ , the causal ordering has been violated.

#### 4.2. The causal ordering algorithm

We are now going to present our algorithm for achieving causal ordering. As in the algorithm described in section 3, we also associate with each site  $S$  a buffer, noted  $\text{ORD\_BUFF\_S}$ , which will be sent along with the messages emitted by  $S$ . However, the contents of this buffer are not messages, but ordered pairs  $(S', VT)$ , where  $S'$  is a destination site of some message and  $VT$  a timestamp vector. Unlike the earlier buffer  $\text{BUFF\_S}$ , this one is bounded in size in the following sense: it holds at most  $(N-1)$  pairs, where  $N$  is the number of sites in the system. Note however that the time vectors in this buffer are not bounded, since their entries depend on the number of events having occurred on each site. The existence of an algorithm enabling to bound the time vectors in the system is an open problem. The causal ordering algorithm is composed of three parts:

- the insertion of a new pair in  $\text{ORD\_BUFF\_S}$  when site  $S$  sends a new message;
- the delivery of a message and the merge of the accompanying buffer with the destination site's own buffer;

knowing only  
a total order,  
n.  
efficient to infer  
proposed in  
principle. The  
in the system.  
on site  $S$ , and  
the following

incremented

s:  $VT_1 < VT_2$  iff  
insensitive. Having  
then  $E_1 \rightarrow E_2$  iff



- the deletion of obsolete pairs in the site's buffer (this part needs no exchange of messages).

#### 4.2.1. Emission of a message

Consider a message  $M$ , timestamped  $VT(M)$  the logical time of emission, and sent from site  $S_1$  to site  $S_2$ . The contents of  $ORD\_BUFF\_S_1$  are sent along with the message. After the message is sent, the pair  $(S_2, VT(M))$  is inserted in  $ORD\_BUFF\_S_1$  (note that this pair was not sent with  $M$ ). This information will be sent along with every message  $M'$  emitted from  $S_1$  after message  $M$ . The meaning of this pair in the buffer is that *no message carrying the pair can be delivered to  $S_2$  as long as the local time of  $S_2$  is not such that  $VT(M) < VT(S_2)$* . This ensures that any message  $M'$ , emitted after  $M$  with destination  $S_2$ , will not be delivered before  $M$ , because the *only* way for  $S_2$  to have a logical time greater than  $VT(M)$ , is to receive message  $M$  or a message  $M'$  which depends causally on  $M$ , i.e.  $SEND(M) \rightarrow SEND(M')$ . However, no message emitted after  $M$  will be delivered to  $S_2$  before its logical time is greater than  $VT(M)$ .

What happens if  $ORD\_BUFF\_S_1$  already contains a pair  $(S_2, VT)$  when  $(S_2, VT(M))$  is to be inserted in the buffer? The older pair can simply be discarded. To see this, we need the following lemma (see also point 4.2.2 and [Schiper 89] for a proof):

*Lemma 2: for any site  $S$ , and any pair  $(S', VT)$  in buffer  $ORD\_BUFF\_S$ , the logical time  $VT(S)$  at  $S$  is such that:  $VT < VT(S)$ .*

Since, by lemma 2,  $VT$  is a time vector such that  $VT < VT(M)$ , the pair  $(S_2, VT)$  becomes obsolete after the insertion of the pair  $(S_2, VT(M))$ . It follows that an ordering buffer contains at most  $(N-1)$  pairs, that is at most one for every site different from the site it is associated with.

Note that for the protocol to be correct there is no need to suppose that messages sent between two given sites arrive in the order in which they are emitted. If a message  $M_2$  was to overtake message  $M_1$ , then  $M_2$  would carry knowledge of  $M_1$  in its accompanying buffer, and so the protocol ensures proper delivery independently of the order of arrival of messages.

Let's complete this point by noting that the algorithm adjusts well to the case of broadcasting a message  $M$  to a set  $DEST$  of destination sites. The solution consists of sending to every site  $S$  in  $DEST$ , in the buffer accompanying  $M$ , all the pairs  $(S', VT(M))$ , where  $S'$  is in  $DEST$  and  $S' \neq S$ .

#### 4.2.2. Arr

Suppose a  
no order  
figure in  
 $VT < VT(S)$

When a r  
accompan  
time. Cor  
accompan  
introduce  
if a pair  
following

- $(S, V$
- $(S, V$

Conjunct  
 $VT_{SUP} = s$   
 $VT < VT_{SU}$

Note fina  
another m

#### 4.2.3. Exa

Figure 5  
algorithm  
arrives at  
which the  
 $SEND(M)$   
of arrival  
 $SEND(M)$

#### 4.2.4. Del

As has a  
bounded.  
solution c  
delivered.  
 $S_2$ , then  
 $VT < VT(M)$   
since the

$\sup(VT_1, V$



#### 4.2.2. Arrival of a message

Suppose a message  $M$  arrives at its destination site  $S_2$ . If the buffer accompanying  $M$  contains no ordered pair  $(S_2, VT)$ , then the message can be delivered. If such a pair does however figure in that buffer (there is at most one), message  $M$  cannot be delivered to  $S_2$  as long as  $VT < VT(S_2)$  is not true.

When a message can be delivered to site  $S_2$ , two actions must be undertaken: (1) merge the accompanying buffer with the destination site's own buffer, and (2) update the site's logical time. Consider the first point. Suppose the existence of a pair  $(S, VT)$ ,  $S \neq S_2$ , in the buffer accompanying message  $M$ . If  $ORD\_BUFF\_S_2$  does not contain any pair  $(S, \dots)$ , then  $(S, VT)$  is introduced in the buffer (if  $S = S_2$ , the pair need not be introduced in  $ORD\_BUFF\_S_2$ ). Now, if a pair  $(S, VT_1)$  already figures in the site's buffer, the meaning of these two pairs is the following:

- $(S, VT_1)$ : no message can be delivered to  $S$  as long as  $VT_1 < VT(S)$  is not true;
- $(S, VT)$ : no message can be delivered to  $S$  as long as  $VT < VT(S)$  is not true.

Conjunction of these two conditions can be translated into a single pair  $(S, VT_{SUP})$  where  $VT_{SUP} = \sup(VT_1, VT)$ <sup>1</sup>. Indeed,  $VT_{SUP}$  is the smallest time vector such that  $VT_1 < VT_{SUP}$  and  $VT < VT_{SUP}$ .

Note finally that delivery of a message forces the local time to progress, so that delivering of another message may be possible.

#### 4.2.3. Example

Figure 5 shows an example illustrating a few typical situations solved by the causal ordering algorithm. In this figure, the end of an arrow points to the moment at which a message arrives at a destination site, whereas the corresponding circled number indicates the order in which the messages are delivered. Consider messages  $M_3$  and  $M_5$  sent to site  $S_4$ . The events  $SEND(M_3)$  and  $SEND(M_5)$  are concurrent, so messages  $M_3$  and  $M_5$  are delivered in the order of arrival. Now consider messages  $M_1$ ,  $M_4$  and  $M_6$ , sent to site  $S_3$ . The order of emission is  $SEND(M_1) \rightarrow SEND(M_4) \rightarrow SEND(M_6)$ . The messages are delivered in this order.

#### 4.2.4. Deletion of an obsolete pair in the ordering buffer

As has already been indicated, the number of pairs in the ordering buffer of a site is bounded. It can however be interesting to delete obsolete pairs from a buffer. The simplest solution consists of comparing message timestamps with buffer timestamps when messages are delivered. Namely, if message  $M$ , timestamped  $VT(M)$  and sent by site  $S_1$ , is delivered to site  $S_2$ , then compare  $VT(M)$  with the time vector of the pair  $(S_1, VT)$  in  $ORD\_BUFF\_S_2$ . If  $VT < VT(M)$ , then the pair has become obsolete and can be deleted from the local buffer, since the local time on  $S_1$  is already greater than  $VT$ .

<sup>1</sup>  $\sup(VT_1, VT_2)[i] = \max(VT_1[i], VT_2[i])$







delivered to S until M<sub>1</sub> itself has been delivered to S. This is what we are going to show. But first let's infer a couple of remarks from the algorithm.

*Remark 1:* by definition of the relation "happened before", it follows from SEND(M<sub>1</sub>)→SEND(M<sub>2</sub>) that there exists some maximal sequence E<sub>0</sub>,...,E<sub>m</sub> of events such that SEND(M<sub>1</sub>)=E<sub>0</sub>→E<sub>1</sub>→...→E<sub>m-1</sub>→E<sub>m</sub>=SEND(M<sub>2</sub>) (we define maximality as follows: for all 0≤k<m, and for every event E not in the sequence, E<sub>k</sub>→E→E<sub>k+1</sub> is not true). This sequence need not be unique, but does exist. Then either event E<sub>1</sub> is the delivery of M<sub>1</sub> on its destination site S, or, by maximality, it is a local event on the emission site S<sub>i</sub> of M<sub>1</sub>. The first case will not be considered in the proof since we will suppose message M<sub>1</sub> has not yet been delivered. In the second case, the pair (S,VT(M<sub>1</sub>)) has been introduced in ORD\_BUFF\_S<sub>i</sub> when event E<sub>1</sub> occurs. By point 4.2.2, we then have VT(M<sub>1</sub>)<VT where (S,VT) is the pair accompanying message M<sub>2</sub> to S.

*Remark 2:* consider again message M<sub>1</sub> and site S<sub>i</sub> sending M<sub>1</sub>. As suggested in sections 4.1 and 4.2.1, the only way for any other site S' to have a local time with VT(M<sub>1</sub>)[i]≤VT(S')[i] (or more generally such that VT(M<sub>1</sub>)<VT(S')) is to have received a message M such that SEND(M<sub>1</sub>)→SEND(M).

We can now proceed with the proof. We are going to show by induction on the events of destination site S that until message M<sub>1</sub> sent by S<sub>i</sub> is delivered, none of these events is the delivery of a message M such that SEND(M<sub>1</sub>)→SEND(M).

*Base step:* consider E(S)<sub>1</sub> the first event on destination site S and suppose E(S)<sub>1</sub> is not the delivery of M<sub>1</sub>. Before event E(S)<sub>1</sub> occurs, VT(S)[i]=0, as every other component of VT(S). For every message M in the system such that SEND(M<sub>1</sub>)→SEND(M), M is accompanied by a pair (S,VT), and VT[i]≥1>VT(S)[i] (by remark 1, and since E(S)<sub>1</sub> is not the delivery of message M<sub>1</sub>). So VT<VT(S) is not true. Therefore, in application of the algorithm, event E(S)<sub>1</sub> is not the delivery of message M.

*Induction step:* now consider E(S)<sub>n</sub> the n<sup>th</sup> event on S and assume as induction hypothesis that none of the preceding events E(S)<sub>1</sub>,...,E(S)<sub>n-1</sub> on S is the delivery of a message M such that SEND(M<sub>1</sub>)→SEND(M). If M<sub>1</sub> has not yet been delivered, it follows from this hypothesis and remark 2 that VT(E(S)<sub>n-1</sub>)[i]<VT(M<sub>1</sub>)[i]. Now remark 1 says that, if VT is the time vector of the pair (S,VT) accompanying a message M with SEND(M<sub>1</sub>)→SEND(M), then VT(M<sub>1</sub>)[i]≤VT[i], so VT is not such that VT<VT(E(S)<sub>n-1</sub>)=VT(S). The algorithm then ensures that E(S)<sub>n</sub> is not the delivery of a message M such that SEND(M<sub>1</sub>)→SEND(M).

We can thus infer that, as long as message M<sub>1</sub> has not been delivered to S, no message happening after M<sub>1</sub> can be delivered to that site.

ing algorithm. This  
e going to show in  
s the proof of the

①  
<1,0,1,0>

respects the causal  
(M<sub>1</sub>)→SEND(M<sub>2</sub>),  
nt statement is: if  
(M<sub>1</sub>)→SEND(M<sub>2</sub>) is



2. *Liveness*: To complete the proof, we must still show the liveness of our algorithm, i.e. that in the absence of failures every message in the system is indeed delivered.

*Proof*: ad absurdo. Suppose some message  $M$  has arrived at site  $S$ , and is never delivered. At the time of arrival,  $M$  was accompanied by a pair  $(S, VT)$  such that, for some  $i$ ,  $VT(S)[i] < VT[i]$ . The number of messages that must be delivered to  $S$  before  $M$  is finite (it is smaller than the sum of  $(VT[i] - VT(S)[i])$  over all such  $i$ ). In the absence of failures and after some finite time, all these messages will have arrived at  $S$ . If every such message had been delivered, then we would have  $VT(S) > VT$  and  $M$  could be delivered: contradiction. (This is because if  $VT(S) > VT$  is not true, again  $VT(S)[i] < VT[i]$ , for some  $i$ . Let's call  $n = VT[i]$ . Then the  $n^{\text{th}}$  event on site  $i$  was the emission of a message  $M'$  for  $S$ . If  $M'$  has been delivered to site  $S$ , then  $VT(S)[i] > VT(M')[i] > VT[i]$ : contradiction.)

So there exists at least another message  $M'$  which will not be delivered to  $S$  and should be before  $M$ . If  $(S, VT')$  is the pair corresponding to  $S$  in the accompanying buffer of  $M'$ , then  $VT' < VT$  and  $VT'[i] < VT[i]$  for some  $i$ . We can thus apply the same reasoning to  $M'$  as to  $M$ , which completes the proof by finite decreasing induction.

#### 4.3. Failures

Up to this point, we have not considered failures (this is because our implementation preserves the causal ordering even in the case of failures). Some failures however can have surprising effects. Consider figure 6, where message  $M_1$  is sent from site  $S_1$  to site  $S_2$  before message  $M_2$  is sent to site  $S_3$ . A communication failure might prevent message  $M_1$  from arriving at its destination, but not message  $M_2$  from arriving at  $S_3$ , though sent afterwards. To see this, consider the following sequence: message  $M_1$  is sent to  $S_2$ , but arrives with a parity error; then  $M_2$  is sent but site  $S_1$  breaks down before retransmission of  $M_1$  is done.

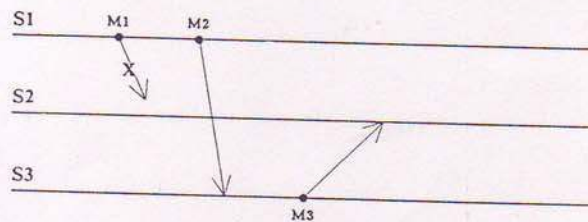


Figure 6. Possible effect of the failure of site  $S_1$ .

What effect does this have on the causal ordering algorithm? Referring to figure 6, we see that message  $M_3$  will arrive at site  $S_2$  together with a pair  $(S_2, VT(M_1))$ . If  $M_1$  is not delivered,  $M_3$  will never be! As a matter of fact, site  $S_3$  will never be able to communicate

with site  $S_2$  a message from reason, any s communicati ordering in t

Solutions to rollback mec delivery of failure, since different pa This exampl without rollb

Let's show H (in the exam agree to the time. Consid failure, each

- consid deliver
- consid These and no
- consid Si of t

When consi globally the this event n site  $S$  such introduce a however, th

#### 5. Conclusi

We have sh message, an ordering.  $S$  delivered to pair  $(S, VT$  Compared



with site  $S_2$  again (meaning messages from  $S_3$  will never be delivered to  $S_2$ ), since every other message from  $S_3$  to  $S_2$  will pile up behind message  $M_3$ , waiting for message  $M_1$ . For the same reason, any site having received a message from site  $S_3$  will be prevented by the algorithm of communicating with site  $S_2$ . This of course is not a satisfying way to implement causal ordering in the case of failures.

Solutions to this problem can be conceived, but, as we will see, they need some sort of rollback mechanism to be introduced (i.e. in figure 6, for site  $S_3$  to recover a state preceding delivery of message  $M_2$ ). Let's note that the ISIS implementation resists to this kind of failure, since message  $M_1$  (which is at the heart of the problem) is sent to site  $S_2$  along two different paths:  $\langle S_1, S_2 \rangle$  and  $\langle S_1, S_3, S_2 \rangle$ , so that message  $M_3$  cannot arrive at  $S_2$  before  $M_1$ . This example clearly suggests that the only way to completely solve the problem of failures without rollback is an implementation like the one of ISIS.

Let's show how failures could be treated in our context. Once a failure has been discovered (in the example of figure 6, most likely by site  $S_2$ ) the remaining working sites must first agree to the time of failure, and then take appropriate actions to rewind their own logical time. Consider a failure of site  $S_i$  (site number  $i$ ). To reach a global agreement on the time of failure, each site  $S$  must proceed as follows:

- consider the set  $P = \{(S, VT_j)\}$  of all pairs accompanying a message  $M_j$  waiting to be delivered to  $S$ ;
- consider then the subset  $P_k = \{(S, VT_{jk})\}$  of  $P$  of pairs such that  $VT_{jk}[i] > VT(S)[i]$ . These pairs carry evidence of some message, emitted by the broken down site  $S_i$ , and not yet delivered to  $S$ ;
- consider finally the number  $MIN(S) = \min_k \{VT_{jk}[i]\}$ .  $MIN(S)$  indicates the number on  $S_i$  of the oldest event  $SEND(M)$  such that message  $M$  was never delivered to site  $S$ .

When considering the minima of  $MIN(S)$  over all the remaining working sites  $S$ , we get globally the oldest event  $SEND(M)$  on  $S_i$  such that message  $M$  was never delivered. Call  $N$  this event number on  $S_i$ . The failure must have occurred on site  $S_i$  after event  $(N-1)$ . Every site  $S$  such that  $VT(S)[i] > (N-1)$  must rewind its clock. A general way of doing this is to introduce a rollback mechanism. Depending on the considered application's semantic however, there could exist a cheaper solution (or no solution at all).

## 5. Conclusion

We have shown in this paper how pairs  $(S, VT)$ , composed of the destination site of some message, and of a Mattern-Fidge logical time vector, make it possible to ensure causal ordering. Such a pair  $(S, VT)$  carried by a message  $M$  says that the message cannot be delivered to site  $S$  before the local time  $VT(S)$  has become greater than  $VT$ . Actually, the pair  $(S, VT)$  indicates that at least one message preceding  $M$  must still be delivered to  $S$ . Compared to this, the implementation of ISIS forces any given message to carry along every

algorithm, i.e. that

never delivered. At that, for some  $i$ , the  $M$  is finite (it is of failures and after a message had been contradiction. (This is call  $n = VT[i]$ ). Then  $s$  been delivered to

to  $S$  and should be buffer of  $M'$ , then ing to  $M'$  as to  $M$ ,

our implementation  $s$  however can have  $S_1$  to site  $S_2$  before t message  $M_1$  from igh sent afterwards. , but arrives with a a of  $M_1$  is done.

to figure 6, we see  $M_1$ ). If  $M_1$  is not able to communicate



causally preceding message in the system, whereas in our scheme, the message carries only some bounded information concerning their existence. On the other hand, we have seen that the implementation of ISIS does not need any special mechanism to treat failures, which can also be of advantage depending on the considered application. Actually a precise quantitative evaluation of the costs of these algorithms should be done. Depending on the characteristics of the application (semantics, real time aspects, etc...) the better suited algorithm could be chosen. We do not rule out the possibility of an algorithm combining advantages of both the ISIS system and our own implementation. Moreover, and independently from these considerations, we think that the proposed causal ordering algorithm will contribute to a better understanding of ordering problems in a distributed system, and, in particular, of the relation of causality.

#### Acknowledgments

We would like to thank the referees for their useful comments.

#### References

- [Birman 87] K.Birman, T.Joseph, "Reliable Communications in Presence of Failures", ACM Trans. on Computer Systems, Vol 5, No 1 (Feb 1987), pp 47- 76.
- [Birman 88a] K.Birman et al., "ISIS - A Distributed Programming Environment", Cornell University, June 1988.
- [Birman 88b] K.Birman, "Exploiting Replication", in Lectures Notes Arctic'88, Tromso, July 1988.
- [Fidge 88] C.Fidge, "Timestamps in Message-Passing Systems That Preserve the Partial Ordering", Proc. of the 11th Australian Computer Science Conference, Univ of Queensland, Feb 1988.
- [Joseph 86] T.Joseph, K.Birman, "Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems", ACM Trans. on Computer Systems, Vol 4, No 1 (Feb 1986), pp54-70.
- [Lamport 78] L.Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM, Vol 21, No 7 (July 1978), pp 558-565.
- [Mattern 89] F.Mattern, "Time and Global States of Distributed Systems", Proc. of the International Workshop on Parallel and Distributed Algorithms, Bonas, France, October 1988, North-Holland 1989.
- [Peterson 87] L.Peterson, "Preserving Context Information in an IPC Abstraction", IEEE Proc. of the 6<sup>th</sup> Symp. on Reliability in Distributed Software and Database Systems, March 1987.
- [Schiper 89] A.Schiper, J.Eggl, A.Sandoz, "A New Algorithm to Implement Causal Ordering", Rapport Interne 89/02, EPFL-Laboratoire de Systèmes d'Exploitation, April 1989.

SYN

Pa

+ Comput

\* Couran

#### Abstract

The symm  
indistinguish  
provide her  
rectional  
only  $O(n)$   
still  $O(n)$   
of [Burns,  
lity, cont  
elected is

#### 1. Intro

We addr  
ption that  
[Angluin,  
blem (call  
exists, it  
known to t  
minate wit  
requires  
their prot  
presented  
assumption  
ging  $O(n)$   
derickson,  
simultane

We prov  
nism. Its  
has to be  
for distri  
manage to  
names from  
trade a s

This work  
and by the