Reducing False Causality in Causal Message Ordering

Pranav Gambhire and Ajay D. Kshemkalyani

Dept. of EECS, University of Illinois at Chicago, Chicago, IL 60607-7053, USA
{pgambhir, ajayk}@eecs.uic.edu

Abstract. A significant shortcoming of causal message ordering systems is their inefficiency because of false causality. False causality is the result of the inability of the "happens before" relation to model true causal relationships among events. The inefficiency of causal message ordering algorithms takes the form of additional delays in message delivery and requirements for large message buffers. This paper gives a lightweight causal message ordering algorithm based on a modified "happens before" relation. This lightweight algorithm greatly reduces the inefficiencies that traditional causal message ordering algorithms suffer from, by reducing the problem of false causality.

1 Introduction

In a distributed system, causal message ordering is valuable to the application programmer because it reduces the complexity of application logic and retains much of the concurrency of a FIFO communication system. Causal message ordering is defined using the "happens before" relation, also known as the causality relation and denoted $\longrightarrow$, on the events in the system execution [11]. For two events $e1$ and $e2$, $e1 \longrightarrow e2$ iff one of the following conditions is true: (i) $e1$ and $e2$ occur on the same process and $e1$ occurs before $e2$, (ii) $e1$ is the emission of a message and $e2$ is the reception of that message, or (iii) there exists an event $e3$ such that $e1 \longrightarrow e3$ and $e3 \longrightarrow e2$.

Let $Send(M)$ denote the event of a process handing over the message M to the communication subsystem. Let $Deliver(M)$ denote the event of M being *delivered* to a process after it is been received by its local communication subsystem. The system respects *causal message ordering* (CO) [3] iff for any two messages M_1 and M_2 sent to the same destination, $(Send(M_1) \longrightarrow Send(M_2)) \Longrightarrow (Deliver(M_1) \longrightarrow Deliver(M_2))$. In Figure 1, causal message ordering is respected if message M_1 is delivered to process P_3 before message M_3.

When a message arrives out of order with respect to the above definition, a causal message ordering system buffers it and delivers it only after all the messages that should be seen before it in causal order, have arrived and have been delivered.

Causal message ordering is very useful in several areas such as managing replicated database updates, consistency enforcement in distributed shared memory,

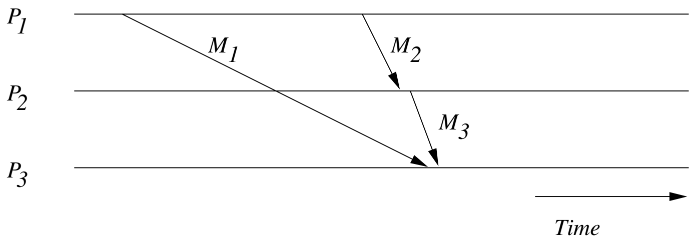

Fig. 1. Causal message ordering.

enforcing fair distributed mutual exclusion, efficient snapshot recording, global predicate evaluation, and data delivery in real-time multimedia systems. It has been implemented in systems such as Isis [3], Transis [1], Horus, Delta-4, Psync [12], and Amoeba [9]. The causal message ordering problem and various algorithms to provide such an ordering have been studied in several works such as [3,4,10,13,14,16] which also provide a survey of this area.

A causal message ordering abstraction and its implementation were given by Raynal, Schiper and Toueg (RST) [13]. For a system with n processes, the RST algorithm requires each process to maintain a $n \times n$ matrix - the $SENT$ matrix. $SENT[i,j]$ is the process's best knowledge of the number of messages sent by process P_i to process P_j. A process also maintains an array $DELIV$ of size n, where $DELIV[k]$ is the number of messages sent by process P_k that have already been delivered locally. Every message carries piggybacked on it, the $SENT$ matrix of the sender process. A process P_j that receives message M with the matrix SP piggybacked on it is delivered M only if, $\forall i$, $DELIV[i] \geq SENT[i,j]$. P_j then updates its local $SENT$ matrix $SENT_j$ as: $\forall i \forall j$, $SENT_j[k,l] = max(SENT_j[k,l], SP[k,l])$. Several optimizations that exploit topology, communication pattern, hardware broadcast, and underlying synchronous support are surveyed in [10] which also identifies and formulates the necessary and sufficient conditions on the information for causal message ordering and their optimal implementation.

Cheriton and Skeen [6] pointed out several drawbacks of the causal message ordering paradigm and these were further discussed in [2,5,15]. The most significant was that every implementation of a CO system has to deal with *false causality.* False causality is the insistence of the system to impose a particular causality ordering of events even though the application semantics do not require such an ordering.

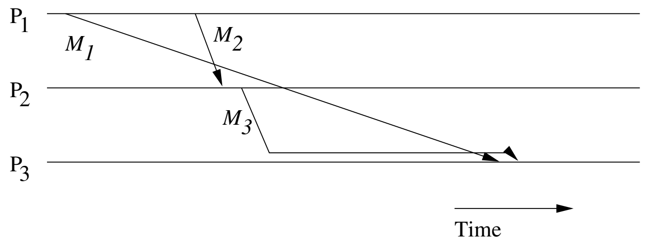

Fig. 2. False causality. No semantic causal dependence between $Send(M_1)$ and $Send(M_3)$.

In Figure 2, $Send(M_1) \longrightarrow Send(M_3)$. Assume that $Send(M_1)$ and $Send(M_3)$ are not causally related according to the application semantics. Now suppose M_3 reaches P_3 before M_1 does. In a causal message ordering system, it is required that $Deliver(M_1) \longrightarrow Deliver(M_3)$ and hence M_3 is buffered till M_1 is received and delivered. $Send(M_1)$ and $Send(M_3)$ are not semantically related and semantically P_3's behavior does not depend on the order in which it receives and is delivered M_1 and M_3. Hence, buffering of M_3 is really unnecessary. This buffering is wasteful of system resources and the withholding of message delivery unnecessarily delays the system execution.

This paper addresses the topic of reduction of false causality in causal message ordering systems. We propose an algorithm in which the incidence of false causality is much lower than in a conventional causal message ordering system that is based purely on the "happens before" relationship.

The notion of false causality arising from the "happens before" relation itself has been identified in several other contexts earlier, even before Cheriton and Skeen pointed out its drawbacks in the performance of causal message ordering algorithms. When Lamport defined the "happens before" relation $\longrightarrow$ [11], he had pointed out that "$e1 \longrightarrow e2$ means that it is *possible* for event $e1$ to causally affect event $e2$" but $e1$ and $e2$ need not necessarily have any semantic dependency. Fidge proposed a clock system to track true causality more accurately in a system with multithreaded processes [7]. However, this scheme and its variants are very expensive. Tarafdar and Garg pointed out the drawbacks of false causality in detecting predicates in distributed computations [17].

Section 2 gives the system model and a framework to design relations that have varying degrees of false causality. Section 3 presents a practical and easy to implement partial order relation based on the framework of Section 2, that reduces many of the false causal relationships modeled by the $\longrightarrow$ relation. Based on this new relation, the section then presents a lightweight algorithm that reduces false causality in causal message ordering. Section 4 concludes.

2 System Model

A distributed system is modeled as a finite set of n processes communicating with each other by asynchronous message passing over reliable logical channels. There is no shared memory and no common clock in the system. We assume that channels deliver messages in FIFO order. A process execution is modeled as a set of events, the time of occurrence of each of which is distinct. An event at a process could be a message send event, a message delivery event, or an internal event. A message can be multicast at a send event, in which case it is sent to multiple processes. A distributed computation or execution is the set of all events ordered by the "happens before" relation $\longrightarrow$ [11], also defined in Section 1. Define $e1 \overset{=}{\longrightarrow} e2$ as $(e1 \longrightarrow e2) \vee (e1 = e2)$. The j^{th} event on process P_i is denoted as e_i^j. Each process P_i has a default initialization event e_i^0. The set of all events E forms a partial order $(E, \longrightarrow)$. The causal past of an event e is denoted $EC(e) = \{e' \mid e' \overset{=}{\longrightarrow} e\}$. The causal past of an event e_i projected on

P_j is denoted $EC_j(e_i) = \{e'_j \mid e'_j \xrightarrow{=} e_i\}$. For any event e_i^k, define a vector *event count* $ECV(e_i^k)$ of size n such that $ECV(e_i^k)[j] = |EC_j(e_i^k)| - 1$. $ECV(e_i^k)[j]$ gives the number of computation events at P_j in the causal past of e_i^k.

False causality is defined based on the true causality partial order relation $\xrightarrow{s}$ on events; the $\xrightarrow{s}$ relation is the analog of the $\longrightarrow$ relation, that accounts only for semantically required causality, and is defined similar to that in [17].

Definition 1. *Given two events* $e1$ *and* $e2$*,* $e1$ *semantically depends on* $e2$*, denoted as* $e1 \xrightarrow{s} e2$*, iff the action taken at* $e2$ *depends on the outcome of* $e1$*.*

We assume that a message delivery event is always semantically dependent on the corresponding message send event. Furthermore, if $e1$ and $e2$ are on different processes and $e1 \xrightarrow{s} e2$, then $\exists\ M \mid e1 \xrightarrow{s} \mathit{Send}(M) \xrightarrow{s} \mathit{Delivery}(M) \xrightarrow{s} e2$. With this interpretation, $\xrightarrow{s}$ is the transitive closure of a "local semantically depends on" relation and the "happens before" imposed by message send and corresponding delivery events.

By substituting $\xrightarrow{s}$ for $\longrightarrow$ in the traditional definition of causal message ordering, the resulting definition will be termed "semantic causal ordering". In contrast, the traditional causal message ordering will be termed the "happens before causal ordering". We will also refer to the causal message ordering problem as simply the causal ordering problem.

If the only ordering imposed on events is to respect the semantic causality relation defined above, then there is no performance degradation due to false causality. In other words, if messages M_1 and M_2 are sent to the same destination, then M_1 need not be delivered before M_2 if $\mathit{Send}(M_1) \not\xrightarrow{s} \mathit{Send}(M_2)$, but M_1 will be delivered before M_2 if $\mathit{Send}(M_1) \xrightarrow{s} \mathit{Send}(M_2)$. This model defines causality among events based on the application semantics. An implementation of this model requires that the semantic causal dependencies of each event be available. Existing programming language paradigms do not permit such specifications and neither does an API for such a specification seem practical. It is possible for a compiler to extract such information by analyzing data dependencies. Alternatively, analogous to Fidge [7] and Tarafdar-Garg [17], we can assume that this information is computable using techniques given in [7]. In fact, as we will show, the causal ordering algorithm we propose requires *only* that the following information is available: for any event, information about the most recent local event on which the event has a semantic dependency.

Definition 2. *Given two events* $e1$ *and* $e2$*,* $e1$ *weakly causes* $e2$*, denoted as* $e1 \xrightarrow{w} e2$*, iff* $e1 \not\xrightarrow{s} e2 \wedge e1 \longrightarrow e2$*.*

For a computation $(E, \longrightarrow)$ and a complete specification of the true causality $(E, \xrightarrow{s})$ in the computation, the amount of false causality is the size of the $\xrightarrow{w}$ relation, which is $\longrightarrow(E \times E) \setminus \xrightarrow{s}(E \times E)$.

Ideally, it is desired to implement the $\xrightarrow{s}$ relation, and have $\xrightarrow{w}$ be the empty relation on $E \times E$. The difficulties in having the programmer specify the

$\xrightarrow{s}$ relation are given above. Though a compiler or an alternate mechanism can identify the exact set of all local and nonlocal events that semantically precede the current event to implement true causality, the overhead of tracking such a set of events as the computation progresses is nontrivial. Therefore, our objective is to approximate $\xrightarrow{s}$ at a low cost to make an implementation practical, and minimize the size of the $\xrightarrow{w}$ relation on $E \times E$. To this end, we introduce the vector MCV to track the latest event at each process such that if the happens-before causal order for a message presently sent is enforced only with respect to messages sent in the computation prefix identified by such events, then semantic causal order is not violated. The vector MCV naturally identifies a computation prefix denoted MC. Thus, for a message sent at any event e, the vector $MCV(e)$ ensures that if happens-before causal ordering is guaranteed with respect to messages sent in $MC(e)$, then semantic causal ordering is guaranteed with respect to all messages sent in the causal past of the event e.

Definition 3. *For any event e, define vector $MCV(e)$ and set $MC(e)$ to have the following properties.*

- *The* maximum causality vector *$MCV(e)$ is a vector of length n with the following properties:*
  - *(Containment:) $\forall j$, $MCV(e)[j] \leq ECV(e)[j]$, and*
  - *(Semantic dependency satisfaction:) $e_j^l \xrightarrow{s} e_i^k \Longrightarrow l \leq MCV(e_i^k)[j]$*
- *$MC(e) = \{e_j' \mid e_j' \longrightarrow e_j^{MCV(e)[j]}\}$, i.e., $MC(e)$ is the computation prefix such that the latest event of this prefix at each process P_j is $MCV(e)[j]$.*

We now make the following Proposition 1 which holds because there is no event that is not in $MCV(e)[j]$ that semantically precedes the event e.

Proposition 1. *For any event e, if every message sent by each P_j among its first $MCV(e)[j]$ events is delivered in happens-before causal order with respect to any messages sent at event e, then every message sent in $ECV(e)$ is delivered in semantic causal order with respect to any messages sent at event e.*

Observe that in general, there are multiple values of MCV that will satisfy Definition 3. Any formulation of MCV consistent with Definition 3 can be used by a causal ordering implementation to reduce false causality. Clearly, different formulations of MCV reduce the false causality to various degrees and can be implemented with varying degrees of ease. Two desirable properties of a good formulation of MCV are:

- It should eliminate as much of the false causality as possible.
- It should be implementable with low overhead.

Happens-before causal ordering of a message sent at event e needs to be enforced only with respect to messages sent in the computation prefix $MC(e)$. To implement this causal ordering, for each event e_i^k, process P_i needs to track the number of messages sent by each process P_j up to event $e_j^{MCV(e_i^k)[j]}$ to every other process. As observed by process P_i at e_i^k, the count of all such messages sent by each P_j up to $e_j^{MCV(e_i^k)[j]}$ to every other process P_l can be tracked by a matrix $SENT[1\ldots n, 1\ldots n]$, where $SENT[j,l]$ is the number of messages sent by P_j up to $e_j^{MCV(e_i^k)[j]}$ to P_l. Analogously, we track the count of all messages sent by each P_j up to $e_j^{ECV(e_i^k)[j]}$ to each other process P_l, by using the matrix $SENT\_ECV[1\ldots n, 1\ldots n]$.

For a traditional causal ordering system, $MCV(e) = ECV(e)$ and this implementation exhibits the negative effects caused by the maximum amount of false causality. Note that here $SENT = SENT\_ECV$ and the matrix $SENT$ as we have defined then degenerates to the matrix $SENT$ as defined in [13].

3 Algorithm

3.1 Preliminaries

We propose the following formulation of vector $MCV(e)$, that is easy to implement and gives a good lightweight solution to the false causality problem. The definition uses the *max* function on vectors, which gives the component-wise maximum of the vectors.

Definition 4. *1. Initially,* $\forall i$, $MCV(e_i^0) = [0,\ldots,0]$.

2. For an internal event or a send event e_i^k,

$$MCV(e_i^k) = \begin{cases} ECV(e_i^k) & \textit{if}\ \exists\ e_q^p \mid MCV(e_i^{k-1})[q] < p \leq ECV(e_i^k)[q]\ \wedge\ e_q^p \xrightarrow{s} e_i^k, \\ MCV(e_i^{k-1}) & \textit{otherwise} \end{cases}$$

3. For a delivery event e_i^k *of a message sent at* e_m^r,

$$MCV(e_i^k) = max(MCV(e_i^{k-1}), MCV(e_m^r))$$

Observe that the only way $e_q^p \xrightarrow{s} e_i^k$, where $q \neq i$, is if $\exists\ e_i^{k'}, e_q^{p'}$ such that $e_i^{k'} \xrightarrow{s} e_i^k$, $e_i^{k'} = Deliver(M)$, $e_q^{p'} = Send(M)$, and $e_q^p \xrightarrow{s} e_q^{p'}$. Also observe that the MCV of an event is the *max* of the ECVs of one or more events in its causal past, and therefore, analogous to EC, if $Deliver(M)$ belongs to the MC of some event, then $Send(M)$ also belongs to the MC of that event. Based on the above two observations, it is possible to simplify the condition test in Definition 4.2 as follows.

Definition 5. *1. Initially,* $\forall i$, $MCV(e_i^0) = [0,\ldots,0]$.

2. For an internal event or a send event e_i^k,

$$MCV(e_i^k) = \begin{cases} ECV(e_i^k) & \textit{if}\ \exists\ e_i^p \mid MCV(e_i^{k-1})[i] < p \leq k\ \wedge\ e_i^p \xrightarrow{s} e_i^k, \\ MCV(e_i^{k-1}) & \textit{otherwise} \end{cases}$$

3. For a delivery event e_i^k *of a message sent at* e_m^r,

$$MCV(e_i^k) = max(MCV(e_i^{k-1}), MCV(e_m^r))$$

Informally, the above formulation of MCV identifies the following events at each process. *For any event e_i^k, (I) $MCV(e_i^k)[j]$, $j \neq i$, identifies the latest event e_j such that some event at P_i occurring causally after e_j and in $EC(e_i^k)$ depends semantically on e_j; (II) If e_i^k depends semantically on some event at P_i that occurs after $MCV(e_i^{k-1})[i]$, then $MCV(e_i^k)$ identifies e_i^k at P_i; otherwise it identifies $MCV(e_i^{k-1})[i]$ at P_i.* Causal ordering of a message sent at e_i^k needs to be enforced only with respect to messages sent in the computation prefix up to these identified events.

Lemma 1 shows that this formulation of MCV is an instantiation of Definition 3. Specifically, it states that each event that semantically happens before e_i^k belongs to the computation prefix up to the events indicated by $MCV(e_i^k)$.

Lemma 1. *Definition 5 satisfies the "Containment" property and the "Semantic Dependency Satisfaction" property of MCV described in Definition 3, i.e., ($\forall j$, $MCV(e)[j] \leq ECV(e)[j]$) and $e_j^l \xrightarrow{s} e_i^k \Rightarrow l \leq MCV(e_i^k)[j]$.*

The formulation of MCV in Definition 5 is easy to implement because we can observe the following property.

Property 1. From Definition 5, it follows that

1. At a send event e_i^k, determining $MCV(e_i^k)$ requires $MCV(e_i^{k-1})$ and identifying the most recent local event on which there is a semantic dependency of e_i^k. This information can be stored locally at a process.
2. At a delivery event e_i^k of a message sent at event e_m^r, determining $MCV(e_i^k)[j]$ requires $MCV(e_i^{k-1})[j]$ and $MCV(e_m^r)[j]$. $MCV(e_m^r)[j]$ can be piggybacked on the message sent at e_m^r.

The following property based on Definition 5 implies that the entries in the $SENT$ matrix at a process are monotonically nondecreasing in the computation.

Property 2. $MCV(e_i^k)[j] \geq MCV(e_i^{k-1})[j]$ i.e., $MCV(e)[j]$ is monotonically nondecreasing at any process.

3.2 Algorithm to Reduce False Causality

Definition 5 of MCV is realized by the algorithm presented in Figure 3. This algorithm is based on the causal ordering abstraction of Raynal, Schiper and Toueg [13] because it provides a convenient base to express the proposed ideas. The proposed ideas can be superimposed on more efficient causal ordering algorithms such as those proposed and surveyed in [10].

Recall that in [13], each send event and each delivery event updates the $SENT$ matrix to reflect the maximum available knowledge about the number of messages sent from each process to every other process in the causal past. This implies that any message M has to be delivered in causal order with respect to all the messages sent in the causal past of the event $Send(M)$, even though there may be no semantic dependency between M and these messages. This is a source of false causality which can be minimized by the lightweight algorithm proposed here.

Each process P_i maintains the data structures $DELIV_i$, $SENT\_CONC_i$ and $SENT\_PREV_i$, described below.

- $DELIV_i$: array[1..n] of integer.
  $DELIV_i[j]$ is P_i's knowledge of the number of messages from process P_j that have been delivered to P_i thus far. It is initialized to zeros.
- $SENT\_PREV_i$: array[1..n,1..n] of integer.
  $SENT\_PREV_i[j,l]$ at e_i^k is P_i's knowledge of the number of messages sent from P_j to P_l up to the event $e_j^{MCV(e_i^k)[j]}$. It is initialized to zeros.
- $SENT\_CONC_i$: array[1..n,1..n] of integer.
  $SENT\_CONC_i[j,l]$ at e_i^k is P_i's knowledge of the number of messages sent from P_j to P_l after the event $e_j^{MCV(e_i^k)[j]}$. It is initialized to zeros.

Recall that the $n \times n$ matrix $SENT\_ECV$ at e_i^k, defined in Section 2, gave the count of all messages sent by each P_j up to $e_j^{ECV(e_i^k)[j]}$ to each other process P_l, in $SENT\_ECV[j,l]$. The two matrices $SENT\_PREV$ and $SENT\_CONC$ at any process have the invariant property that $SENT\_PREV_i[j,l] + SENT\_CONC_i[j,l] = SENT\_ECV_i[j,l]$, as will be shown in Lemma 2. The row $SENT\_PREV_i[j,\cdot]$ reflects the row $SENT\_ECV_i[j,\cdot]$ up to the event $e_j^{MCV(e_i^k)[j]}$, whereas the row $SENT\_CONC_i[j,\cdot]$ reflects the row $SENT\_ECV_i[j,\cdot]$ after that event. The challenge is to maintain the $SENT\_PREV$ and $SENT\_CONC$ matrices at a low cost so as to retain the above property.

The causal ordering algorithm that minimizes false causality is given in Figure 3. At a message send event, steps (E1)-(E7) are executed atomically. Step (E1) determines based on the sender's semantics if the message being sent, and implicitly all future messages sent in the system, should be delivered in causal order with respect to all messages sent so far, i.e., whether $MCV(e_i^k) = ECV(e_i^k)$. Specifically, the test *should_Semantically_Precede* uses two inputs: $MCV(e_i^{k-1})$ and the latest local event on which there is a local dependency (Property 1). These two inputs are used to check for Definition 5.2, i.e., to determine whether there exists an event e_i^p such that $e_i^{MCV(e_i^{k-1})[i]} \longrightarrow e_i^p \wedge e_i^p \xrightarrow{s} e_i^k$, in which case $MCV(e_i^k)$ will be greater than $MCV(e_i^{k-1})$. If $MCV(e_i^k)$ is greater than $MCV(e_i^{k-1})$, then the condition *should_Semantically_Precede* becomes *true* and $MCV(e_i^k)$ should be set to $ECV(e_i^k)$. In this case, steps (E2)-(E5) update the matrices $SENT\_PREV$ and $SENT\_CONC$ to reflect this; otherwise $SENT\_PREV$ and $SENT\_CONC$ are left unchanged. Step (E6) sends the message with the two matrices piggybacked on it. Step (E7) updates $SENT\_CONC$ to reflect the message(s) just sent.

When a message M, along with the sender's $SENT\_PREV$ and $SENT\_CONC$ matrices SP and SC piggybacked on it, is received by a process, M can be delivered only if the number of messages delivered locally so far is greater than or equals the number of messages sent to this process as per SP (step (R1)). Steps (R2)-(R8) are executed atomically. The message gets deliv-

```
Data structures at $P_i$
D1. $DELIV_i$: array [1..n] of integer
D2. $SENT\_PREV_i$: array [1..n,1..n] of integer
D3. $SENT\_CONC_i$: array [1..n,1..n] of integer

Emission of message M from $P_i$ to $P_j$
E1. if ( $should\_Semantically\_Precede$) then
E2.     for $k = 1$ to $n$ do
E3.         for $l = 1$ to $n$ do
E4.             $SENT\_PREV_i[k,l] = SENT\_PREV_i[k,l] + SENT\_CONC_i[k,l]$
E5.             $SENT\_CONC_i[k,l] = 0$
E6. Send($M, SENT\_PREV_i, SENT\_CONC_i$)
E7. $SENT\_CONC_i[i,j] = SENT\_CONC_i[i,j] + 1$

Reception of (M, SP$_M$, SC$_M$) at $P_j$ from $P_i$
R1. Wait until ($\forall k, SP_M[k,j] \leq DELIV_j[k]$)
R2. Deliver M
R3. $SC_M[i,j] = SC_M[i,j] + 1$
R4. $DELIV_j[i] = DELIV_j[i] + 1$
R5. for $k = 1$ to $n$ do
R6.     for $l = 1$ to $n$ do
R7.         $SENT\_CONC_j[k,l] = \max(SENT\_CONC_j[k,l] + SENT\_PREV_j[k,l],$
                $SP_M[k,l] + SC_M[k,l]) - \max(SENT\_PREV_j[k,l], SP_M[k,l])$
R8.         $SENT\_PREV_j[k,l] = \max(SENT\_PREV_j[k,l], SP_M[k,l])$
```

Fig. 3. Causal message ordering algorithm to minimize false causality.

ered in step (R2). Steps (R3)-(R4) update the data structures SC and $DELIV$ to reflect that the message was sent and has now been delivered, respectively. Steps (R5)-(R8) update the data structures $SENT\_CONC$ and $SENT\_PREV$. These steps ensure that $SENT\_PREV$ reflects the maximum knowledge about the messages that were sent by events in the MC of the current event, while $SENT\_CONC$ reflects the maximum knowledge about the messages that were not sent by events in the MC of the current event.

3.3 Correctness Proof

We state some lemmas and the main theorem that prove the correctness of the algorithm. See the full paper for details [8].

Lemma 2 gives the invariant among $SENT\_PREV$, $SENT\_CONC$ and $SENT\_ECV$ at any event.

Lemma 2. $SENT\_PREV_i + SENT\_CONC_i = SENT\_ECV_i$

Lemma 3 states that the $SENT\_PREV$ matrix reflects exactly all the messages sent by various processes up to $MCV(e)$. This includes all the send events that semantically precede all local events up to the current event.

Lemma 3. *The messages sent by* P_j *until* $e_j^{MCV(e_i^k)[j]}$ *correspond exactly to the messages represented by* $SENT\_PREV_i[j,\cdot]$ *at* e_i^k.

Lemma 4 states that messages represented by the matrix $SENT\_CONC$ are the messages sent concurrently in terms of semantic dependency with respect to the current event. These are the messages with respect to which no "happens-before" causal ordering needs to be enforced in order to meet the semantic causal ordering requirements.

Lemma 4. *The messages represented by $SENT\_CONC_i[j, \cdot]$ at e_i^k correspond exactly to the messages sent in the left-open right-closed duration $(MCV(e_i^k)[j],$-$ECV(e_i^k)[j]]$.*

Theorem 1. *The algorithm given in Figure 3 implements causal message ordering with respect to the relation $\xrightarrow{s}$.*

3.4 Algorithm Analysis

With the proposed approach, any message M multicast at event e_i should be delivered in causal order with respect to all the messages represented by $SENT\_$-$PREV$, i.e., sent in the computation prefix $MC(e_i)$. Messages represented by $SENT\_CONC$ are those with respect to which no false causality is imposed (follows from Lemma 4 and Proposition 1). The amount of false causality in enforcing causal delivery of M is the size of $\xrightarrow{w}(MC(e_i) \times MC(e_i))$, in contrast to the traditional causal ordering where the amount of false causality is the size of $\xrightarrow{w}(EC(e_i) \times EC(e_i))$. While $\xrightarrow{w}(MC(e_i) \times MC(e_i))$ may still be large (although smaller than that for the traditional approach), indicating that much false causality may still exist, this is not so on close analysis. With the proposed approach, false causality is potentially imposed only with respect to some of the messages sent up to MCV. Each $MCV(e_i^k)[j]$ will usually be much less than $ECV(e_i^k)[j]$. Hence, false causality is potentially imposed only with respect to some of the messages sent in the more distant past. In practice, message delivery times tend to have an exponential distribution. Hence, messages sent in the distant past up to the events indicated by $MCV(e_i^k)$ would have been delivered with high probability and the present message could most likely be delivered as soon as it arrives and without any buffering. Only in the case that some message sent in the distant past has not been delivered, and a false causality exists on such a message, that the proposed algorithm will unnecessarily delay the present message and require some buffering. We expect that such a case will have a low probability of occurrence, and when it occurs, the extra delay incurred by the imposed false causality will be small.

The computational complexity of the proposed algorithm is the same as that of the RST algorithm. The $O(n^2)$ extra computation in steps (R5)-(R7) is the same order of magnitude as in the RST algorithm. The $O(n^2)$ extra computation in steps (E1)-(E5) is of the same order of magnitude as in the RST algorithm. In terms of space complexity, the RST algorithm requires $n^2 \times m$ bits of storage space and message overhead, where m is the size of the message counter, whereas the proposed algorithm requires $2n^2 \times m$ bits of storage space and message overhead, which are comparable.

The only additional overhead is to implement Definition 5. By Property 1, this requires the MCV vector of the previous local event and the identity of the latest local event on which there is a semantic dependency. The former information is already computed; the latter can be assumed to be available as in [7,17] or can be extracted from compiler data.

4 Concluding Remarks

False causality in causal message ordering reduces the performance of the system by unnecessarily delaying messages and requiring large buffers to hold the delayed messages. We presented an efficient algorithm for implementing causal ordering that eliminates much of the false causality. In particular, the algorithm eliminates the false causality in the near *past* of any message send event. Some false causality with respect to messages sent in the more distant past exists. It is expected that such false causality will have a minimal degradation on the performance of the ideal causal ordering implementation because messages sent in the more distant *past* will have been delivered with high probability and cause buffering of the present message with low probability. The implementation presented here is lightweight and requires about the same order of magnitude overhead as the baseline RST algorithm, with minimal additional support.

Acknowledgements

This work was supported by the U.S. National Science Foundation grants CCR-9875617 and EIA-9871345.

References

1. Y. Amir, D. Dolev, S. Kramer and D. Malki, Transis: A communication sub-system for high-availability, *Proc. 22nd International Symposium on Fault-tolerant Computing*, IEEE Computer Society Press, 337-346, 1991.
2. K. Birman, A response to Cheriton and Skeen's criticism of causal and totally ordered communication, *Operating Systems Review*, 28(1): 11-21, Jan. 1994.
3. K. Birman, T. Joseph, Reliable communication in the presence of failures, *ACM Transactions on Computer Systems*, 5(1): 47-76, Feb. 1987.
4. K. Birman, A. Schiper and P. Stephenson, Lightweight causal and atomic group multicast, *ACM Transactions on Computer Systems*, 9(3): 272-314, Aug. 1991.
5. J. Caroll, A. Borshchev, A deterministic model of time for distributed systems, *Proc. Eighth IEEE Symposium on Parallel and Distributed Processing*, 593-598, Oct. 1996.
6. D.R. Cheriton, D. Skeen, Understanding the limitations of causally and totally ordered communication, *Proc. 11th ACM Symposium on the Operating Systems Principles*, 44-57, Dec. 1993.
7. C. Fidge, Logical time in distributed computing systems, *IEEE Computer*, 24(8): 28-33, Aug. 1991.

8. P. Gambhire, Efficient Causal Message Ordering, M.S. Thesis, University of Illinois at Chicago, April 2000.
9. F. Kaashoek, A. Tanenbaum, Group communication in the Ameoba distributed operating system, *Proc. Fifth ACM Annual Symposium on Principles of Distributed Computing*, 125-136, 1986.
10. A. Kshemkalyani, M. Singhal, Necessary and sufficient conditions on information for causal message ordering and their optimal implementation, *Distributed Computing*, 11(2), 91-111, April 1998.
11. L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, 21(7): 558-565, July 1978.
12. L.L. Peterson, N.C. Bucholz and R.D. Schlichting, Preserving and using context information in interprocess communication, *ACM Transactions on Computer Systems* 7(3), 217-246, 1989.
13. M. Raynal, A. Schiper, S. Toueg, The causal ordering abstraction and a simple way to implement it, *Information Processing Letters* 39:343-350, 1991.
14. L. Rodrigues, P. Verissimo, Causal separators and topological timestamping: an approach to support causal multicast in large-scale systems, *Proc. 15th IEEE International Conf. on Distributed Computing Systems*, May 1995.
15. R. van Renesse, Causal controversy at Le Mont St. Michel, *Operating Systems Review*, 27(2):44-53, April 1993.
16. A. Schiper, A. Eggli, A. Sandoz, A new algorithm to implement causal ordering, *Proc. Third International Workshop on Distributed Systems*, Nice, France, LNCS 392, Springer-Verlag, 219-232, 1989.
17. A. Tarafdar, V. Garg, Addressing false causality while detecting predicates in distributed programs, *Proc. 18th IEEE International Conf. on Distributed Computing Systems*, 94-101, May 1998.