Causal Broadcast Protocol for
Very Large Group Communication Systems

Saul POMARES HERNANDEZ, Jean FANCHON, Khalil DRIRA, Michel DIAZ

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

7, Av. du colonel Roche - 31077 Toulouse – France

{spomares, fanchon, khalil, diaz}@laas.fr

Abstract

The paper presents a causal protocol for very large group communication systems. The protocol does not make any assumption concerning the group structure nor the network topology. It aims to minimize and control the flow of causal information through two mechanisms. First, it reduces the amount of causal information (CI) timestamped per message by avoiding the transmission of redundant information. The second original mechanism we propose controls and bounds the size of transmitted CI, without losing causality, through the use of additional control messages. Our protocol guarantees a great flexibility, as well as the minimization of exchanged causal information. In addition, it accelerates the knowledge of the system concerning its causal past.

Key words: Group Communication, Causality, Vector Clocks, Large Size Groups, Message-Passing.

1. Introduction

In Group Communication Systems (GCS), causal ordering protocols are an essential tool to exchange information. The use of causal ordering provides built-in message synchronization and reduces the non-determinism in a distributed computation. It is an extension of the FIFO channel property to a group of participants. Messages are delivered to each member following the causal order of their emission. The concept of causal ordering is of considerable interest to the design of distributed systems and can be found in applications of several domains, such as distributed cooperative engineering, teleconferencing [19], multimedia systems [3], mobile computing [17], real-time [9] and security domains [21].

The causal delivery of the messages in a group communication protocol is usually ensured by sending causality information (CI) together with the messages of the protocol. The main problem is the amount of information emitted to enforce causal ordering: in a group G with n members, this amount is $\theta(n)$ [5]; for large values of n, the bound of $\theta(n)$ is prohibitively high.

Furthermore, causal constraints may dramatically slow down the message delivery in large groups. This is due to the asynchronism and to the different communication delays.

In this paper, we present an efficient causal broadcast protocol that aims to control, minimize and optimize the transmission of causal information. The protocol does not make any assumptions concerning the group structure nor the network topology. Furthermore, the protocol does not use causal servers to disseminate information. All the participants are considered to be peers.

The paper proceeds as follows: Section 2 discusses related work. In Section 3 the contributions of this article are presented. Section 4 describes the System Model. It provides notations, and presents the causal order background for the paper. Section 5 presents, in detail, our protocol, along with its correctness proof. The conclusions and future works are discussed in Section 6.

2. Related work

We distinguish two approaches for solving the problem concerning the amount of information for large groups in causal communication protocols:

i. The first approach consists in bounding in advance the amount of causal information transmitted [8,24]. It has the great disadvantage that it no longer completely preserves causality. The approach can provide a high level of ordering accuracy, but only under appropriate circumstances. It is useful in systems where placing an ordering on concurrent events only impacts performance, and not correctness. Such works include, for example, consistency maintenance schemes and resource allocation algorithms [26].

ii. The second approach attempts to reduce the size of causal information transmitted, without losing causal dependencies. We can classify two categories:

ii.a The first category does not make any assumptions concerning the group structure nor the topological network structure. This category is only concerned with finding the necessary and sufficient conditions to enforce causal message ordering, and/or finding an optimal encoding scheme to represent and transmit this information. Examples of protocols of this category include the work of Kshemkalyani and Singhal [10]. Their work proposes to only piggyback onto every message the bare minimum information concerning all prior causal messages that either are not yet guaranteed to be delivered in causal ordering, or are not known to have been already delivered. A second example concerns protocols that use vector timestamps. In these protocols, it is possible to only send the positions that have changed between the previously sent vector and the current one [23,4]. For this category, in the worst scenario, the complexity and bound of control information for broadcast remains $\theta(n)$[5,22,12]. Protocols of this category have two main advantages. First, they can be used in applications that can be configured at run time (dynamic groups). Second, they are easily scalable.

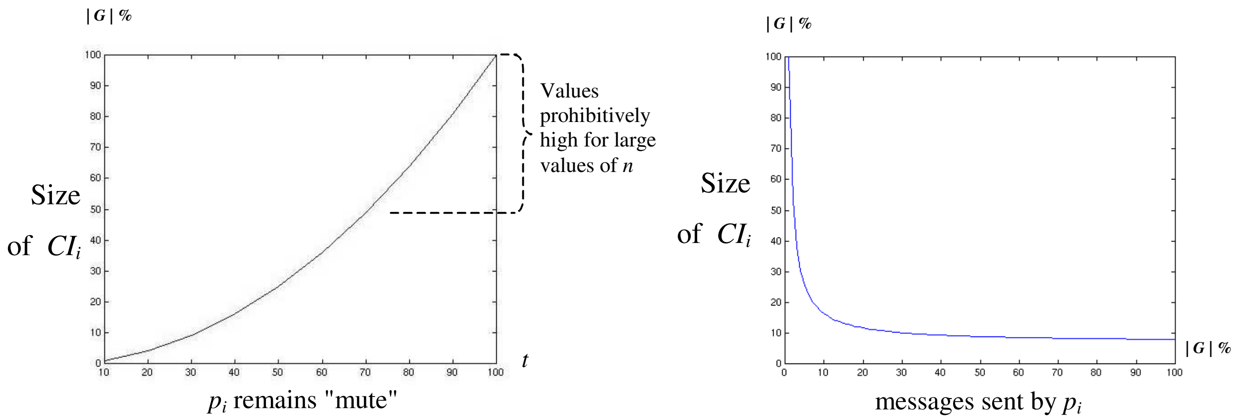
ii.b The second category is composed of solutions that assume certain network topology and/or communication patterns [20], and/or certain group structure [1]. The main disadvantage for protocols that assume certain *network topology* is that they are usable in applications where the topology is relatively static or can be computed at compile or configuration time. Protocols of this type are absolutely inappropriate to applications where it is not possible to know in advance when, where, nor who (user, host, process, agent, etc) will participate, (e.g. almost all the cooperative internet applications). An example of a protocol that assumes certain *network topology* is the work of Rodrigues and Verísimo [20], who use causal separators as a barrier to filter all information concerned with the messages exclusively addressed to the backward set. A causal zone (set of process bounded by one or more causal separators) never sends CI outside of its boundaries. An example of a protocol that assumes certain *group structure* is proposed by Baldoni et al. [1], who split the participating processes into local groups, and use causal servers to disseminate messages across these groups. Thus, the amount of control information is reduced to the local group size. The disadvantage of this technique is that the use of causal servers adds a delay to the delivery of messages that need to cross local group boundaries. Therefore, this technique is not suitable for applications with time constrains. The protocols that fall into this second category are not easily scalable.

3. Contributions of the paper

In this paper, we present an efficient causal broadcast protocol that aims to control, minimize and optimize the transmission of causal information. Its main features and contributions are:

- The protocol does not make any assumptions concerning the group structure nor the network topology. Therefore, it has the advantages listed in Section 2 ii.a: configuration at run time and scalability. *However, the bound never reaches $\theta(n)$.* Furthermore, to prevent any delay in the messages delivery, the protocol does not use causal servers to disseminate information (unlike those presented in Section 2 ii.b). All the participants are considered to be peers.
- The *minimisation of the causal information* transmitted is achieved by avoiding the transmission of redundant information. It is based on the *Propagation Constraints* (PC) established by Kshemkalyani et al. [12].
- A new mechanism introduces *additional Light Control Messages* (LCMs[1]) *that only*

[1] In the sequel, we'll use the term *messages* to refer to application messages (messages carrying information concerning the application). The term *LCM messages* strictly refers to additional control messages used in our protocol.



a) b)

Fig. 1 Average behavior of the amount of CI_i for a) inactive processes (general case) and b) active processes (best case)

carry causal information. These LCM messages may be sent at any moment and allow a control on the way CI is transmitted during executions. The control is achieved particularly in both the size of CI per message and in the timing of the emissions. It is important to notice that the overhead induced by these new messages is minimized by the fact that their mutual causal relationships are not taken into account by the protocol. Only their causal relationships with the original messages are considered. They only need to be delivered in FIFO order.

- In the present paper, the LCM messages are used to bound the amount of CI per message. However, they may be emitted on different criteria than the size of CI. We emphasize that the bound on CI used here can be modified at any moment, depending on local or global conditions. To our knowledge, our proposed mechanism is the first to provide the property of bounding the number of entries of message timestamps without loss of accuracy in causal dependencies.

The benefits of our work are depicted in Fig.1. In the approaches of the same category as our protocol [12,13], when a process remains "mute," (without sending messages), the amount of CI not yet emitted can grow (Fig. 1 a). If a process takes an active part in the exchange of messages, this amount remains small (Fig. 1 b). The way to control this amount consists in forcing a process to generate a LCM message each time that the CI becomes too high. The LCM message, like a normal message, acts as a reset that reduces to zero the amount of CI recorded by the sender.

Besides their effect on the amount of CI emitted per message and recorded per process, the LCM messages anticipate the exchange of CI. Therefore, they diminish the asynchronism of the system. We discuss in Section 6 their possible impact on the detection of the loss of messages and site failures.

4. Preliminary definitions and notations

4. 1 The System Model

We assume that a group is composed of processes $G = \{p_1, p_2, p_3 \ldots p_n\}$, where each group and each process has a unique identifier. The system has no global clock, and the group participant processes do not share memory. They communicate only by asynchronous messages, and each message is sent to the entire group (meaning, it is broadcast). The communication channels are non-FIFO. The message propagation delay is finite, but unpredictable. The underlying communication medium is reliable (in future works, this condition will not be necessary). A process p fails only in a fail-stop manner.

Similarly to the approach of Hélary et al. [8], we distinguish events that need causal reinforcement from those which do not. Nevertheless, to be generic, we distinguish only two *relevant group* events with causal relationships, namely, *message send* and *message delivery,* and one *irrelevant group* event with FIFO relationships, namely, *LCM message send.*

A message send event, denoted *send*(m), is a procedure that is enacted according to the following steps:

1. The message sent is placed in the output buffer of the process.
2. The message is timestamped with the corresponding causal information.

Notations	Description
$VT(p_i)$	Vector time of process p_i.
(i, a)	Denotes any type of occurring event at local time a at process p_i.
G	Denotes the group of processes involved.
$\|G\|$	Denotes the number of processes in group G
CI_i	Local list at process i records causal information since the last $send(m_{i,a})$ event. The elements of CI are pairs (process number, sequence number)
H	Structure timestamped that carry the necessary causal information for relevant messages and LCM messages.
$m_{i,t}$	Message sent by process p_i at local time t. The subscripts of $m_{i,t}$ can be dropped if there is not ambiguity.
$m_{i,t}.Dests$	Set of destinations of multicast message $m_{i,t}$ (In our protocol $Dests = G$)
$nm_{i,t}$	*LCM* message sent by process p_i at local time t.
$LCM\_send_i(m)$	Event at which a LCM message nm is sent by process p_i
$send_i(m)$	Event at which a message m is sent by process p_i.
$Delivery_j(m)$	Event at which message m is delivered at process p_j.
b	The "*b-bound*" value (See Section 5.2)

Table 1. Explanation of the notations used

3. It is subsequently broadcasted to all the group participants by an underlying network protocol.

A message delivery event, denoted *delivery*(m), is a procedure that includes the following steps:

1. The message that arrives is placed in the input buffer of the process.
2. Verification of the causal dependencies of the message takes place.
3. If the causal dependencies are satisfied, then the message is delivered to the application; otherwise, the message is stored in the local buffer.

A LCM message send event, denoted *LCM_send*(m), is processed following the same steps as the *send*(m) event.

Table 1 summarizes all the notations we will use in the paper to describe our protocol.

4.2 Causal Dependency Information

Our delivery causal ordering (CO) protocol is based on logical clocks, called *vector clock,* proposed by Mattern and Fidge[14,6]. The receiving site uses vector time to determine whether some message(s) needs to be delivered before the current message is delivered. The sending site uses the *Propagation Constraints* (PC) established by Kshemkalyani et al. [12] to transmit the bare minimum causal dependency information using the propagation constraint principle.

4.2.1 Vector Time

The vector time has been commonly used to determine the causal relationships between messages [4,20,2,15,18]. It is denoted $VT(p_i)$ on process p_i. Its size is $n = |G|$.

The general algorithm of vector time is as follows:

1. Initially, $VT(p_i)[j] = 0\ \forall j=1\ldots n$.
2. For each event *send*(m) at p_i,
   $VT(p_i)[i] = VT(p_i)[i] + 1$.
3. Each multicast message by process p_i is timestamped with the updated value of $VT(p_i)$.
4. For each event $delivered_j(m')$, p_j modifies its vector time in the following manner:
   $VT(p_j)[k]=max(VT(p_j)[k],VT(m')[k])\ \forall k=1\ldots n$.

The way to determine causal relationships between messages [4] $m \rightarrow m'$ using vector timestamps $VT(m)$ and $VT(m')$ is:

$$m \rightarrow m' \text{ iff } VT(m) < VT(m') \text{ where}$$
$$VT(m) < VT(m') \text{ if } \forall k\text{: } VT(m)[k] \leq VT(m')[k]$$
$$\text{and } \exists k\text{: } VT(m)[k] < VT(m')[k]$$

Note. We maintain and use the vector time on each process to check causal relationships at the reception of messages. However, the vector time is not timestamped.

4.2.2 Propagation constraints

The Propagation Constraints (PC) specify the sufficient and necessary conditions on the information propagation to enforce causal ordering [12].

Definition. The information $d \in m_{i,a}.Dests$ concerning a message $m_{i,a}$ sent to d must propagate along all causal paths, starting at event (i, a) up to, and only up to, the earliest events (j, b) on any such path, such that either:

Vector Time	Necessary and sufficient causal information
1. Initially	
i. $VT(p_i)[j] = 0 \; \forall j{:}1\ldots n$.	ii. $CI_i \leftarrow \emptyset$
2. For each event *send*(*m*) at p_i	
i. $VT(p_i)[i] = VT(p_i)[i] +1$	ii. $H_m \leftarrow CI_i$
3. Broadcast *send*(*m*) $m= (i,\ t_i{=}VT(p_i)[i],\ mess,\ H_m)$	
	i. $CI_i \leftarrow \emptyset$ — From PC 2. The p_i does not have to send (case broadcast) any entry of CI_i which is guaranteed that it will be delivered in CO.
4. For each event *delivery*(*m*) at p_j, $m=(k,\ t_k,\ mess,\ H_{m'})$	
To enforce causal order delivery of m i. **if not** $(t_k = VT(p_j)[k] +1 \wedge t_l \leq VT(p_j)[l] \;\; \forall (l, t_l) \in H_m)$ **then** *wait* **else**	
ii. $VT(p_j)[k] = VT(p_j)[k] +1$	iii. a) **if** $\exists ci_{s,t'} \in CI_j \mid k = s$ **then** $CI_j \leftarrow CI_j \setminus \{ ci_{s,t'} \}$ b) $CI_j \leftarrow CI_j \cup \{(k, t_k)\}$

Table 2. Listing of the basic causal protocol.

PC 1. $delivery_d(m_{i,a}) \overset{=}{\rightarrow} (j, b)$, i.e. it is known that $m_{i,a}$ has been delivered or

PC 2. $\exists\, (k, c) \mid (i, a) \rightarrow (k, c) \rightarrow (j, b) \wedge d \in m_{k,c}.Dests$ i.e. it is guaranteed that it will be delivered in CO.

The first PC means that the message $m_{i,a}$ is known to have been delivered in causal order at event (j, b) to destination $d = j$. Therefore, in any future multicast message to destination d, the message does not need to carry information concerning $d \in m_{i,a}.Dests$.

The second PC means that there exists an event (k, c) that would ensure the causal delivery of the present message (j, b) with respect to (i, a). If (k, c) is causally delivered with respect to (i,a), the information about $d \in m_{i,a}.Dests$ does not need to be sent.

5. The Causal Protocol

We present our protocol in two parts. The first part, Section 5.1 (Basic Causal Protocol) describes the protocol without LCM messages. Its main property is to avoid the transmission of redundant causal information. In the second part, Section 5.2, we present the extension of the basic causal protocol with LCM messages and bounded transmitted CI.

5.1 The Basic Causal Protocol (First Mechanism)

In order to reduce the amount of information, the basic causal protocol does not transmit causal information, which is already guaranteed to be delivered in CO (propagation constraint PC 2). Therefore, the only causal information timestamped in each message is updated by the messages received between the last send event and the current one. It is recorded in each process k in the structure CI_k, where each entry (process number i, sequence number t_i) corresponds to the last send event $(i,\ t_i)$ at the process p_i of a message received at p_k.

First, we begin to define the structures used:

- The causal information structure CI_i is a set of entries $(k,\ t_k)$. Each entry denotes a message received by p_i sent by process p_k at clock $t_k = VT(p_k)[k]$.
- The message structure $m_{i,ti}$ is a tuple $m_{i,ti} = (i,\ t_i,\ mess,\ H_m)$, where i is the process identifier, $t_i{=}VT(p_i)[i]$ is the local sequence message counter at node i, $mess$ is the content, and H_m contains the causal information. H_m is a copy of CI_i, when $m_{i,ti}$ is emitted.

<table>
<tr><th>Vector Time</th><th>Necessary and sufficient causal information</th></tr>
<tr><td colspan="2">5. From the protocol listed in Table 2, 4. iii. b)</td></tr>
<tr><td></td><td>4 iii. b) $CI_j \leftarrow CI_j \cup \{(k, t_k)\}$
i. $e \leftarrow |CI_i|$</td></tr>
<tr><td colspan="2">6. if $e > b$ then
(For each automatic event $LCM\_send(nm)$ at p_i)</td></tr>
<tr><td>i. $VT(p_i)[i] = VT(p_i)[i] + 1$</td><td>ii. $H_{nm} \leftarrow CI_i$</td></tr>
<tr><td colspan="2">7. Broadcast LCM_send(nm)
$nm = (i, t_i = VT(p_i)[i], H_{nm})$</td></tr>
<tr><td></td><td>i. $CI_i \leftarrow \emptyset$ From PC 2. The p_i does not have to send (case broadcast) any entry of CI_i which is guaranteed that it will be delivered in CO.</td></tr>
<tr><td colspan="2">8. For each event delivery(nm) at p_j,
$nm = (k, t_k, H_{nm})$</td></tr>
<tr><td colspan="2">To enforce causal order delivery of nm
i. if not ($t_k = VT(p_j)[k] + 1 \wedge t_l \leq VT(p_j)[l] \; \forall \, (l, t_l) \in H_{nm}$) then wait
else</td></tr>
<tr><td>ii. $VT(p_j)[k] = VT(p_j)[k] + 1$</td><td>Note: Between LCM messages nm, our protocol does not guarantee causal delivery. It only guarantees FIFO property.</td></tr>
</table>

Table 3. Listing of the causal protocol with bounded entries.

As we can see in Table 2, the basic protocol is composed of two parts:

- The right part uses the principles of Kshemkalyani and Singhal described in Section 4.2.2.
- The left part uses the algorithm of vector time described in Section 4.2.1.

A site uses vector time to maintain accuracy in the causal history information of the messages sent, and it uses the principles proposed by Kshemkalyani and Singhal to transmit the necessary and sufficient causal dependency information.

5.2 Introducing LCM messages and bounding the number of entries of message timestamp (Second Mechanism)

We present the algorithm introducing LCM messages (Table 3), which follows Table 2. The condition ($e<b$ in Table 3.6) for the emission of LCM messages is a bound on the size of CI. Each time that $e > b$, where $e=|CI_i|$ and $1 \leq b \leq n$ is a bound for e, the process p_i is forced to automatically generate a *LCM* send event $LCM\_send(nm_{i,ti})$ (See Table 3.5.i and 3.6). (Hélary et al. have first introduced this «*b-bound*» constraint in [8]). In the previous basic protocol, the number of entries in CI_i is not bounded; it can grow until $\theta(n)$. The algorithm works even if the condition $e<b$ is replaced by random (see proof below). This means that the bound can be modified dynamically during the execution.

The function of a LCM message $nm_{i,ti}$ is to carry the causal information stored in CI_i from the last $send_i(m)$ event. And then, like a $send_i(m)$ event, the $LCM\_send(nm_{i,ti})$ event resets CI_i to $\emptyset$. A LCM send message $nm_{i,ti}$ is not processed in the same way as a send message m_{i,t_i} (see Table 2.4 and 3.8). Both are delivered in causal order, but LCM messages do not modify the structure *CI* of the receiver.

The causal protocol with bound message timestamps is shown in Table 3.

The proposition below states that this algorithm ensures the causal delivery of messages, even in the presence of (any number of) LCM messages. These LCM messages can be emitted at any moment, independently of the value of b.

The causal (or precedence) relation between events is denoted $\rightarrow$. We recall that it is the smallest transitive relation containing the local (and total) orderings on the events of each single process and the sets of pairs of events $\{send_i(m) \rightarrow delivery_k(m)\ ,\ k=1 \ldots n\}$ of emissions and receptions of messages.

Proposition:

If m and m' are two application messages such that $send_i(m) \rightarrow send_j(m')$ then for any process p_k, $k=1...n$, $delivery_k(m)$ occurs before $delivery_k(m')$ i.e. $delivery_k(m) \rightarrow delivery_k(m')$

Proof: Given $m = (i, t_i, mess, H_m)$ and $m' = (j, t_j, mess, H_{m'})$ such that $send_i(m) \rightarrow send_j(m')$.

Two cases may occur:

I) First case: $i=j$, the two messages are sent by the same process p_i. The FIFO order delivery between p_i and any process p_k is ensured by [**4. i**][2] (and [**8.i**]), and this implies that $delivery_k(m) \rightarrow delivery_k(m')$.

II) Second case: $i \neq j$. We prove this case by induction on the distance $d(m,m')$ between m and m', i.e. the maximal length n of a sequence of messages $m=m_0..m_i..m_n=m'$ such that $send(m_i) \rightarrow send(m_{i+1})$, $i=0,n-1$. (This distance must be finite for m' to be sent!)

The case $d(m,m')= 1$ is considered below. Otherwise, for some process p_l and some message m'', we have $send_i(m) \rightarrow send_l(m'') \rightarrow send_j(m')$, $d(m'',m')= 1$ and $d(m,m'')= d(m,m') -1$. The induction hypothesis implies that for any process p_k, $delivery_k(m) \rightarrow delivery_k(m'')$ and $delivery_k(m'') \rightarrow delivery_k(m')$, which implies $delivery_k(m) \rightarrow delivery_k(m'') \rightarrow delivery_k(m')$.

Case $d(m,m')= 1$ (Fig 2): this implies that $delivery_j(m) \rightarrow send_j(m')$. We can also restrict to the case where m' is the first application message sent by p_j after $delivery_j(m)$, because if we prove that $delivery_k(m) \rightarrow delivery_k(m')$ holds in that case, then for a message m'' such that $send_j(m') \rightarrow send_j(m'')$, the case **I**) above ensures that $delivery_k(m') \rightarrow delivery_k(m'')$ and finally $delivery_k(m) \rightarrow delivery_k(m') \rightarrow delivery_k(m'')$.

Thus, we suppose that $delivery_j(m) \rightarrow send_j(m')$, and m' is the first application message sent by p_j after $delivery_j(m)$. After $delivery_j(m)$, we have $VT(p_j)[i]=(t_i)$ and $(i,t_i) \in CI_j$ [**4. ii** and **4.iii**]. There are only two events which may modify or remove (i,t_i) from CI_j before $send_j(m')$ occurs:

1) A message $m'' = (i, t, mess'', H_{m''})$ or $m'' = (i, t, H_{m''})$ is received by p_j. In that case $t > t_i$ (due to FIFO order) and $(i,t) \in CI_j$ after the reception of m''[**4.ii** and **4.iii**].

2) A LCM message sent by p_j empties CI_j. Let $m''=(j, t_j, H_{m''})$ be the first LCM message sent by p_j, such that $delivery_j(m) \rightarrow LCM\_send_j(m'') \rightarrow send_j(m')$, then (see the case 1), $(i,t) \in H_{m''}$ for some $t \geq t_i$ [**6. ii** and **7.i**].

Then, after $delivery_j(m)$, when $send_j(m')$ is executed, two situations may occur:

A) No message has been sent by p_j. $H_{m'}$ has a component $(i,t) \in H_{m'}$ such that $t \geq t_i$ and the message m' must wait to be delivered to process p_k that $VT(p_k)[i] \geq t \geq t_i$ [**4. i**]. This is possible only if m has been received by p_k ($VT(p_k)[i]$ is modified only on receptions of messages from p_i in FIFO order, [**4. ii** and **8.ii**]) and it ensures that $delivery_k(m) \rightarrow delivery_k(m')$.

B) Otherwise, $H_{m'}$ does not carry an entry for the process i, due to a LCM message $m''=(j, t_j'', H_{m''})$ such that $delivery_j(m) \rightarrow LCM\_send_j(m'') \rightarrow send_j(m')$ and such that for some $t \geq t_i$, $(i,t) \in H_{m''}$ (Case 2 above). This message m'' must wait to be delivered to process p_k that $VT(p_k)[i] \geq t \geq t_i$ [**8. i**], and this is possible only if m has been received by p_k. Due to the FIFO ordering, we must have that $delivery_k(m'') \rightarrow delivery_k(m')$. We conclude that $delivery_k(m) \rightarrow delivery_k(m'') \rightarrow delivery_k(m')$.□

This situation is depicted in Fig. 2.

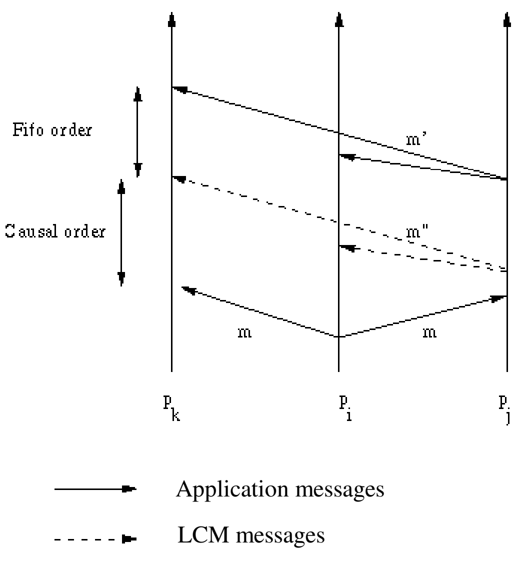


Fig. 2 Example of maintenance of causality between LCM and Application messages.

---

[2] The references in braces [] in this proof refer to the algorithm in the Tables 2 and 3.

6. Conclusions and Future Work

In this paper, we have presented a Causal Broadcast Protocol, which minimizes and controls the transmission of the causal information. One original mechanism presented bounds and splits the causal information by using LCM messages. We use LCM messages in a way that does not increase the causal dependencies, but maintains those between the other messages. The protocol guarantees the minimization of the transmitted causal information and a great flexibility. It also accelerates the knowledge of the system about its causal history.

Moreover, the use of LCM messages offers the additional advantage of a possible anticipation in the detection of the loss of messages. A LCM message sent by a process p_i after the reception of a standard message m may reach a process p_j while m has not yet been received by p_j. Furthermore, the periodic emission of LCM messages may anticipate the detection of site failures.

Our future work involves the extension of the causal protocol to support multi-group communications, including overlapping groups.

In addition, we are currently developing a mechanism that will dynamically control and optimize the emission of LCM messages. The main objective of this mechanism is to dynamically calculate a trade-off between the number of LCM messages and the timestamp size. It will do this by taking into account the changing characteristics of the group system.

This protocol has been implemented in Java and is intended to be used in Collaborative Distributed Applications.

7. References

[1] R. Baldoni, R. Friedman, R. van Renesse, "The Hierarchical Daisy Architecture for Causal Delivery", 17th IEEE International Conference on Distributed Computing Systems, May 1997.

[2] R. Baldoni, A. Mostefaoui and M. Raynal "Causal Delivery of Messages with Real-Time Data in Unreliable Networks ", *Journal of Real-time Systems*, Vol. 10, no. 3, 1996.

[3] R. Baldoni, R.Prakash, M Raynal., M. Singhal, "Efficient Causally Ordered Communications for Multimedia Real-time Applications", in Proceedings of the 4th International Symposium on High Performance Distributed computing, pp 140-147, Washington D.C., Aug. 1995.

[4] K. Birman, A. Schiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", *ACM Trans. Compt. Syst*. Vol. 9, No. 3, pp. 272-314, Aug. 1991.

[5] B. Charron-Bost, "Concerning the Size of Logical Clocks in Distributed Systems", *Inf Process Lett* 39:11-16 (1991).

[6] C.A. Fidge, "Timestamps in Message-Passing Systems that Preserve Partial Ordering", *Australian Computer Science Communications*, Vol 10, no. 1 : 56-66, 1988.

[7] B. Glade, K. Birman, R Cooper, and R. van Renesse. "Light-Weight Process Groups in the ISIS System", *Distributed System Engineering*, (1):29-36, 1993.

[8] J. Hélary, G. Melideo, M. Raynal, " Tracking Causality in Distributed Systems: a Suite of Efficient Protocols", in Proc. of the 7th International Colloquium on Structural Information and Communication on Complexity, *Carleton Scientific*, June 2000.

[9] R. Jacinto Montes, S. Pomares Hernández, F. Ramos,"A Protocol for WorkGroup in Industrial Environments Based on MMS", International Conference On Principles Of Distributed Systems (OPODIS), Paris France, 1998.

[10] A. D. Kshemkalyani et M. Singhal, "An Optimal Algorithm for Generalized Causal Message Ordering", in Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96), pp. 87-87, ACM, May 1996.

[11] A. Khesemkalyani, "Causality and Atomicity in Distributed Computations", In *Distributed Computing* 11: 169-189, 1998.

[12] A. D. Kshemkalyani and M. Singhal, "Necessary and sufficient conditions on information for causal message ordering and their optimal implementation", *Distributed Computing* 11: 91-111, 1998.

[13] L. Lamport, "Time, clocks and the ordering of events in distributed systems", *Communications ACM* 21(7): 558-565, 1998.

[14] F. Mattern, "Virtual Time and Global States of Distributed Systems ", *Parallel and Distributed Algorithms*, North-Holland, 215-226, 1989.

[15] A. Mostefaoui and M. Raynal, "Causal multicast in overlapping groups towards a low cost approach", IEEE Workshop on Future Trends of Distributed Computer Systems, pp 136-142, September 1995.

[16] D. Powell, editor. "Delta-4 : A Generic Architecture for Dependable Distributed Computing". *ESPRIT Research Reports*. Srpinger Cerlag, November 1991.

[17] R. Prakash, M. Raynal, M.Singhal, " An Effient Causal Ordering Algorithm for Mobile Computing Environments", Technical report, Parallel Architectures Dept., Inst. Nat. de Rech. En Inf. et en Aut., France, 1995.

[18] M. Raynal, A. Shiper, S. Toueg, "The Causal Ordering Abstraction and a Simple way to Implement it", *Inf. Process Lett* 39(6): 343-350, 1991.

[19] K. Ravindran, B Prasad, "Communication Structures and Paradigms for Distributed Conferencing Applications", 12th IEEE Int. Conf. On Distributed Computing Systems, May 1992.

[20] L. Rodrigues and P. Veríssimo, "Causal Separators and Topological Timestamping: an Approach to Support Causal Multicast in Large-Scale Systems", in the Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, British Columbia, Canada, May 1995.

[21] F. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial", *ACM Compt. Surveys*, vol 22, No. 4, pp 299-319, Dec. 1990 .

[22] R. Schwarz and F. Mattern, "Detecting Causal Relationships in Distributed Computations: in Search of the Holy Grail", *Distributed Computing* 7: 149-174, 1994.

[23] M. Singhal, A. Kshemkalyani, "An Efficient Implementation for Vector Clocks", *Inf Process Letter* 43: 47-52, 1992.

[24] F.J. Torres-Rojas, M. Ahamad, "Plausible Clocks: Constant Size Logical Clocks for Distributed Systems, *Distributed Computing*, 12(4):179-196,1999.

[25] F. Torres-Rojas, M. Ahamad, "Plausible Clocks: Constant Size Logical Clocks for Distributed Systems", *Distributed Computing* 12: 179-195, 1999.