Published in the Proceedings of the Brazilian Symp. nn Computer Networks (SBRC), Belo Horizonte, May/1995

Causal Order Protocols for Group Communication

Raimundo José de Araújo Macêdo
Universidade Federal da Bahia - LaSiD[1]
Salvador, Bahia, Brasil
email: macedo@ufba.br

ABSTRACT

Group communication (or multicast) is a powerful abstraction that can be used whenever groups of distributed processes cooperate for the execution of a given task. Due to the uncertainties inherent to distributed systems (emerging from communication or process failures), multicast messages may arrive in an inconsistent order at different destination processes. Further complications will arise when groups overlap (i.e. a process is allowed to belong to distinct groups). In this paper, we address the problem of causal order violation in message delivery. We present three causal order protocols for overlapping groups with different trade-offs between message space overhead[2] and message delivery delays. The first protocol favours message space overhead (small one) with a longer message delivery delay. The second protocol favours message delivery delay but with a larger message space overhead. The third protocol is a compromise solution between message delivery delay and message space overhead, comparing favourably with respect to existing solutions.

Key works: group communication, multicast protocols, causal order, logical clocks, distributed algorithms, distributed systems, network protocols.

1. Introduction

Group communication (or multicast) is a powerful abstraction that can be used whenever groups of distributed processes cooperate for the execution of a given task such as committing a distributed data base transaction, or to achieve fault-tolerance or better performance (by replication) [Bir91]. In group communication, processes usually communicate in a group basis where a message is sent to a group of processes, rather than just to one process, which is the case in point-to-point communication. With a group is usually associated a name to which application processes will refer, making transparent the location of the distributed processes forming the group. Due to the uncertainties inherent to distributed systems (emerging from communication or process failures), group communication protocols have to face situations where, for instance, a sender process fails when a multicast is underway or where messages arrive in an inconsistent order at different destination processes.

[1]Laboratório de Sistemas Distribuídos (LaSiD). End: Prédio do CPD/UFBa, Campus de Ondina, CEP 40.170-110, Salvador, Bahia, Brasil. Fax: (071) 247 9907

[2] The protocol related information contained in a multicast message

Further complications will arise when groups overlap (i.e. a process is allowed to belong to distinct groups). As observed in [Lam78], messages exchanged by distributed processes can be partially ordered according to their causal origin. In this paper, we address the problem of causal order violation in message delivery, caused by umpredictable message transmission delays. Process and communication failures have been addressed elsewhere [Mac94, EMS95].

To illustrate the need of causal order preserving delivery, consider for example, a computer based conferencing application where users may simultaneously participate in different conversations (or groups). Suppose a given multi-group user generates a message m' in a group B as a consequence of a message m delivered to the same user in group A. Other multi-group users participating simultaneously in groups A and B, would then require that m' be delivered only after m has been delivered (otherwise, m' may make no sense). Since message m potentially caused message m', we say that they are causally related [Lam78]. For correct delivery of messages m and m', we require a protocol which delivers messages respecting their causal origin or in causal order.

Causal order protocols for group communication have been proposed by several authors [BSS91, PBS89, ADKM92, MR93]. However, overlapping groups are only addressed in [BSS91, MR93]. In [BSS91] is described an efficient causal order protocol called CBCAST. The CBCAST protocol delivers messages efficiently regarding message delivery delays. However, when complex group overlapping structures are considered, CBCAST can lead to a (potentially) large message space overhead (this will be examined in more details). In this paper, we present three causal order protocols for overlapping groups with different trade-offs between message space overhead and message delivery delays. The first of our causal order protocols favours message space overhead (small one) with a longer message delivery delay. The second protocol favours message delivery delay but with a larger message space overhead. The third protocol is a compromise solution between message delivery delay and message space overhead, producing a trade-off similar to the protocol presented in [MR93]; it produces, however, a faster message delivery than [MR93] when single groups are considered.

We begin by discussing the causal order abstraction in section 2. Section 3 states our system model and failure assumptions. In section 4 we summarize the basic timestamping mechanism upon which our protocols as well as other mechanisms have been developed. Section 5 presents the causal order protocols. Finally, section 6 concludes the paper.

2. Preserving Causal Order in Group Communication

2.1 Event Ordering in Distributed Systems

A process execution consists of a sequence of events, each event corresponding to the execution of an action by a process. Within a given process, events are naturally ordered by the sequence they happen. However, ordering events from distinct processes is not possible unless they execute communication actions among themselves. An example of a communication action executed by a process, say p, can be to send a multicast message, say m, to a group that is recorded in m as $m.g$. The corresponding event will be denoted as $send_p(m)$. Similarly, we denote the event of a process q, belonging to the group $m.g$, receiving m as $receive_q(m)$. Then, we can define a 'happened before' relation, denoted as '$\rightarrow$', on *send* and *receive* events in a given set of system events. Thus, when a, b, and c are three distinct events in a subset of system events, each referring to either *send* or *receive* events,

(i) if a comes before b in the same process, then $a \rightarrow b$. e.g., if $send_p(m)$ comes before $send_p(m')$, then $send_p(m) \rightarrow send_p(m')$;

(ii) if a is a $send_p(m)$ and b is $receive_q(m)$, then $a \rightarrow b$; and

(iii) if $a \rightarrow b$ and $b \rightarrow c$, than $a \rightarrow c$.

A message m will be said to have potentially *caused* m', if $send(m) \rightarrow send(m')$, and distinct messages m and m' will be said to be *concurrent* if neither $send(m) \rightarrow send(m')$ nor $send(m') \rightarrow send(m)$ is true. Hence, the relation '$\rightarrow$' establishes a partial order of events in a distributed system [Lamt78]. For notational simplicity, when m and m' are two distinct multicasts, $m \rightarrow m'$ will denote that $send(m) \rightarrow send(m')$.

When some specific delivery order is required, received messages may have to be retained for later delivery until certain ordering conditions are satisfied (otherwise, the delivery order stated may be violated). Thus, we need to define $delivery_p(m)$ as the event of delivering message m to process p.

2.2 Causal Order Delivery

If an event a "happened before" event b (i.e., a $\rightarrow$ b), then, event b may have been caused or influenced by event a. If a and b are two message multicasts, to respect causality, the delivery of a must precede delivery of b, in all common destinations of a and b. Below is the specification of causal order delivery in overlapping groups.

Causal order delivery : $send_p(m) \rightarrow send_r(m')$, $p \in m.g$ and $r \in m'.g \Rightarrow deliver_q(m) \rightarrow deliver_q(m')$, for all q, $q \in m.g \cap m'.g$.

In figure 1, we illustrate causal delivery in a 3-member process group. The process group is formed by processes p_1, p_2, and p_3. The horizontal lines represent the passage of time and oblique lines message flows between processes (arrows touching the lines represent delivered messages). Process p_1 multicast message m_1 and process p_2, after delivering m_1, multicast m_2. Note that p_3 receives m_2 first but its delivery is delayed until m_1 is received and delivered.

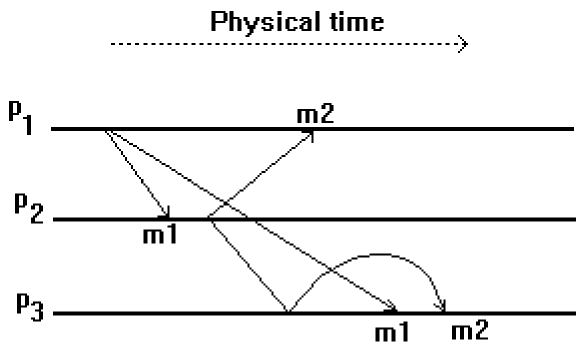

Figure 1 - Causal Order Delivery

3. The System Model and Failure Assumptions

We assume a set of sequential processes which are distributed possibly on distinct processors or sites and communicate with each other only by exchanging messages. We assume a transport layer that provides lossless, uncorrupted, and sequenced message delivery between any pair of functioning processes (FIFO assumption)[3]. Once a message has been sent by the transport layer, it can take an potentially unbounded amount of time to be received at the destination (i.e. we assume an asynchronous system). For simplicity, in this paper we assume a failure-free environment (i.e., processes do not crash or misbehave). In [Mac94] we present mechanisms for coping with dynamic changes in the membership (due to failures such as process crashes and network patitioning) that can be used with the causal order protocols we will describe. We also assume that all the members of a group are lively: a member will eventually multicast a new message. This assumption can be removed by the introduction of a simple mechanism based on local time-outs [MES93b].

4. An Overview of the Causal Blocks Model

In what follows we briefly describe our basic timestamping mechanism upon which the causal order protocols are developed. We have named this mechanism the Causal Blocks model which is a framework for developing group communication protocols with different ordering and reliability requirements [MES93a].

Consider the existence of a group, $g = \{p_1, p_2, , ... , p_n\}$. Each process p_i maintains a logical clock called the Block Counter and denoted as BC_i. BC_i is an integer variable and its value increases monotonically. When g is created, the BC_i of every p_i is initialized to any finite integer, and without loss of generality, we will suppose that they are all initialized to zero. Transmitted messages are timestamped with Block Counters, and, as it is the case in Lamport's Logical Clock, timestamping using Block Counters will respect causality. That is, if $m \rightarrow m'$, then the timestamp associated with m' will be larger than the timestamp associated

[3] This can be realised by introducing sequential numbers to messages, with positive acknowledgement and retransmission of missing messages (TCP/IP provides such a functionality).

with m. However, the reverse will not always hold. If two messages have different timestamp values, they may be concurrent. Unlike Lamport's Logical Clock that is advanced on send/receive events, we have decided to advance Block Counters on send/delivery events, since a transmitted message can only be causally dependent on previously delivered ones.

Just before a process p_i multicasts a message m, it advances BC_i by one. The contents of the incremented BC_i is assigned to m as its block-number in the message field m.b. BC_i may also be advanced by p_i on $delivery_i(m)$ if the current value of BC_i is less than m.b. Thus, the two events under which BC_i may be advanced are:

CA1 (Counter advances during $send_i(m)$): Before p_i multicasts m, it increments BC_i by one, and assigns the incremented value to m.b; and,

CA2 (Counter advances during $delivery_i(m)$): Before pi delivers m, it sets $BC_i = \max\{BC_i, m.b\}$.

Based on *CA1* and *CA2* we can state the three following properties possessed by block-numbers of multicast messages. For notational simplicity, we denote $send_{pi}$ as simply $send_i$.

pr1 : $send_i(m) \rightarrow send_i(m') \Rightarrow m.b < m'.b$.

pr2 : for any m, P_j $m.g$: $deliver_j(m) \rightarrow send_j(m') \Rightarrow m.b < m'.b$; and,

pr3 : for all $m', m'' : m'.b = m''.b \Rightarrow m'$ and m'' are concurrent.

The properties *pr1* and *pr2* follow directly from *CA1* and *CA2*, respectively. Together they imply that for any distinct m, m': send(m) $\rightarrow$ send(m') $\Rightarrow m.b < m'.b$. The property *pr3* states that distinct messages multicast with the same block-number are necessarily concurrent and these messages must have been multicast by distinct processes, as *CA1* forbids two send events to occur in a given process with the same value of BC.

Finally, notice that the reverse of *pr1* is not always true. That is, if there are two distinct messages m and m' such that m $\rightarrow$ m', not necessarily m.b < m'.b is true. To verify this statement, just notice that a sequence of send events will increment the Block Counter irrespective of the existence of received (but not delivered) messages.

Causal Blocks and the Block Matrix

Consider a process group $g = \{p_1, p_2, ... , p_n\}$. Using *pr3*, p_i constructs Causal Blocks to represent concurrent messages it sent/received with the same block-number. Construction of Causal Blocks leads to the notion of Block Matrix which can be viewed as a convenient way of representing sent and received messages with different block-numbers. A Causal Block is a vector of size equal to n = |G|. Whenever a process p_i receives or sends a multicast message with a new block-number, say B, it creates an empty vector of length n; for any message multicast with block-number B, it sets the i^{th} entry of the vector to '+'; and, for any multicast message received with block-number B from another process p_j, $j \neq i$, it sets the j^{th} entry of the vector to '+'. Causal Blocks, maintained by a process in this way, will have the following property:

PR1 : in a given Causal Block, only concurrent messages are represented.

When group communication is active, i.e. member processes continually send multicast messages, the number of Causal Blocks constructed will grow. Causal Blocks maintained by a process are arranged in the increasing order of the message block-number they represent, giving rise to a matrix which is called the *Block Matrix* and denoted as BM. Thus, BM[B] will represent the Causal Block for message block-number B. Referring to BM of any member process in G, we can state another property of Causal Blocks,

PR2: if m and m', represented in BM[B] and BM[B'], respectively, are causally related such that $m \rightarrow m'$, then B < B'.

	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3	+		+	+		
4	+				+	
5		+				

Figure 2 - The Block Matrix of a 6-member Group Process

Figure 2 shows the Block Matrix of a 6-member process group. It represents all messages sent/received by the process which owns this particular matrix. Since messages arrive at destinations with presumably distinct delays, the BM matrices of different processes will not necessarily be the same in a given moment of physical time. The BM matrix showed in figure 2, indicates, for example, that the block-numbers of the last messages received from processes p_1 and p_2, are 4 and 5, respectively.

Block Completion

By *PR1*, all messages which belong to a given Causal Block, say BM[B], are concurrent. From *PR2*, any causal order message delivery based solely on block numbers will require that a received message represented in BM[B], B > 1, be delivered only after all multicast messages which can be represented in BM[B'], for all B' < B, have been delivered.

To enable a member process to accurately determine that a given block completely represents all messages which can be represented in it, we use the notion of *block completion*.

A Causal Block BM[B], $B \geq 1$, will be said to be *complete*, if and only if for all j, $1 \leq j \leq n$, the j^{th} entry of BM[B] either (i) has '+' or (ii) is blank and there exists B', B'>B such that the j^{th} entry of BM[B'] has '+'. By (i), it is ensured that if p_j has sent a message with block-number B, it is received. By *CA1* and *CA2*, p_j will not produce two distinct multicast messages with the same block-number. Since messages from a given sender process are received in FIFO order, (ii) guarantees that p_j has not multicast any message with block-number B. In the example shown in figure 2, block 2 is complete because processes p_2, p_5, and p_6 have sent a message with block-number 2, and processes p_1, p_3, and p_4 have sent a message with block-number 3.

From *PR2* it is obvious that the existence of BM[B], B > 1, also implies the existence of some Causal Block BM[B'], for some B' < B (B' = B - 1 if Block Counters are incremented by 1). Since messages are assumed to be transmitted in FIFO order, Causal Blocks in the BM maintained by a process will complete in the sequentially increasing order of block-numbers they represent.

5. The Causal Order Protocols

If two messages m and m' are causally related ($m \rightarrow m'$), a causal order protocol will deliver m' to a process p only after m has been delivered to p. If the delivery of m' is delayed only if there exist a message m'', such that $m'' \rightarrow m'$ and m'' has not yet been delivered to p, we say that the causal order protocol delivers messages as early as possible. Causal order protocols have to compromise between delivery speed (as early as possible principle) and the size of message timestamps containing causality information [MR93]. Small timestamps usually means slower delivery. The first causal order protocol for process groups, the CBCAST protocol [BJ87], was based on the idea of message piggybacking. In CBCAST, causally related messages were piggybacked on top of transmitted messages; this made the protocol quite expensive in terms of message space overhead. In the second version of CBCAST [BSS91b], a message to be transmitted is timestamped with a vector clock [Mat89, Fid91, BSS91b] which is a compact representation of a causal history within a process group. Vector clocks lead to a precise causal dependency representation between transmitted messages, and it is a lower-bound in message space overhead in order to get delivery as early as possible [Cha91, Ray92]. When groups overlap, causal order protocols based purely on vector clocks [BSS91b] are quite expensive in terms of message space overhead. Typically, vector clocks related to all groups in the system have to be transmitted on top of transmitted messages. Although some optimisations are possible (e.g., using compression - [BSS91b]), the high cost is unavoidable to guarantee fast causal delivery (as early as possible delivery). In order to avoid the high cost of vector clocks for overlapping process groups, some compromise must be made, trading a potentially slower delivery with smaller timestamps.

Using the concept of block completion for causal message delivery, when a message m arrives at a destination process p, it can only be delivered after the Causal Block (m.b - 1) becomes complete in p. As we have previously stated, some of the messages with block-number less than m.b may actually be concurrent to m (the reverse of *pr1* is not always true).

The waiting time introduced by block completion can be avoided if we can deduce the precise set of messages m depends on. In the next section, we shall show how Causal Block numbers can be used to precisely represent causal relationship between transmitted messages, leading to a faster causal order protocol. We will then present three protocols for causal order delivery in overlapping process groups, in the context of causal blocks[4], and describes a trade-off solution in more details. The first protocol, called Slow causal protocol, is purely based on message block-numbers. It is optimum in message space overhead but may introduce extra delay time for message delivery because it relies on block-completion. The second protocol, the Fast causal protocol is based on a vector called the Global Last Delivered Vector (GLDV). It reduces message delivery delay (delivery as early as possible) but has a higher message space overhead. The third protocol, the Relative causal order protocol, uses the Last Delivered Vector (LDV) combined with the block-completion concept and is a trade-off solution between message space overhead and delivery delay.

Sections 5.2 discusses causal order delivery in overlapping process groups. We describe the Slow, Fast, and Relative causal order protocols, in sections 5.3, 5.4, and 5.5, respectively.

5.1 Representing Causal Relationship precisely using Causal Blocks Numbers

Consider that instead of timestamping a message m with just its block-number, we introduce a timestamp that includes also the block-numbers of the last delivered message from each of the group members by the time m is sent. When m is received at a destination process p, it can be delivered immediately after the messages represented by those block-numbers have also been delivered at p.

Consider that each member process p_i of a group g maintains a vector, called the Last Delivered Vector and denoted as LDV, whose size is the size of g. The contents of LDV, together with the message block-number, will be used to timestamp sent messages. At any time, LDV[j] of p_i, $1 \leq j \leq n$, will indicate the largest block number of the message from p_j that p_i has delivered. Assume that m.b and m.s denote the block-number and the sender process identifier associated with message m, respectively. Thus, LDV of p_i will satisfy the following at any given time,

$$\text{LDV}[j] = \max\{\text{m.b} \mid \text{m.s} = p_j \text{ and } \mathit{deliver}_i(m) \text{ has occurred}\}.$$

With respect to its LDV, p_i will carry out the following:

A1 : just before $\mathit{send}_i(m)$: m.ldv = LDV,

A2 : just after $\mathit{send}_i(m)$: LDV[i] = m.b (or BC_i), and

A3 : just after $\mathit{deliver}_i(m)$: LDV[m.s] = m.b,

[4] Mechanisms such as a membership protocol and a flow control scheme have been developed in the context of Causal Blocks [
protocols described in this paper.

Consider the LDV vectors maintained by processes p_i and p_j, and denoted as LDV_i and LDV_j, respectively. The values of LDV_i and LDV_j are compared using the following rules:

C1 : $LDV_i \leq LDV_j \Leftrightarrow \forall k : LDV_i[k] \leq LDV_j[k]$

C2 : $LDV_i < LDV_j \Leftrightarrow LDV_i \leq LDV_j$ and $\exists k : LDV_i[k] < LDV_j[k]$

For causal message delivery, a received message *m* is delivered to p_i immediately after the following delivery condition is true:

DC : for all p_j *m.g*, $m.LDV[j] \leq LDV_i[j]$.

By the above condition, any message that p_i sends is immediately deliverable at p_i. Hence, we will assume that for *m, m.s* = p_i, $send_i(m)$ is immediately followed by $deliver_i(m)$.

LDV vectors have similar properties to vector clocks [BSS91b]. The difference is due to the fact that an entry of a given LDV represents the Block Counter value of a given group process (i.e. the process logical time), whereas an entry of a vector clock indicates how many events have happened in a given process by a given moment of its computation. In other words, $LDV_i[j]$ indicates not only the last message sent by process p_j and delivered by p_i, but also informs p_i that p_j has sent/delivered messages up to Causal Block $LDV_i[j]$. This p_i's knowledge of p_j's logical time progress, as we will show in this chapter, provides our protocol with an efficient way of dealing with causal message delivery in the presence of group overlapping. Below we state some properties of LDV vectors.

LDV vector Properties

Consider messages *m1* and *m2* timestamped as m1.LDV and m2.LDV, respectively. Based on the previous definitions we state the following properties *pv1, pv2*, and *pv3* possessed by LDV vector timestamps.

pv1 : $m1.b = \max(m1.LDV[j], 1 \leq j \leq n) + 1$;

When a message m1 is sent, its block-number m1.b is given by the incremented value of the current Block Counter (by *CA1*). Thus, to verify *pv1*, we only need to show that the value of a Block Counter $BC_i = \max(LDV_i[j], 1 \leq j \leq n)$. Notice that for i = j, $LDV_i(j)$ is only updated to equalise BC_i (by *A2*). For $i \neq j$, by *A3*, $LDV_i(j)$ is updated to m1.b of a delivered message m1. By *AC2*, BC_i will maintain the maximum value between the previous value of BC_i and m1.b ($LDV_i(j)$ after *A3*). Hence, $BC_i = \max\{LDV_i[j], 1 \leq j \leq n\}$ holds, and therefore, *pv1*.

pv2 : $m1 \rightarrow m2 \Leftrightarrow m1.LDV < m2.LDV$ and $m1.b \leq m2.LDV[i]$

Let p_i and p_j be the processes which sent m1 and m2, respectively. By the definition of the relation '$\rightarrow$', if $m1 \rightarrow m2$, then either i = j and m2 was sent after m1, or m2 was sent after P_j has delivered m1. In the former case, by *A2*, $LDV_i[i]$ will be incremented after m1 be sent, thus $m2.LDV > m1.LDV$ and $m1.b \leq m2.LDV[i]$ will hold. In the latter case, notice that m1 can only be delivered in p_j after LDV_j be larger or equal than m1.LDV. Because the block-number of a message is always larger than all block-numbers present in its timestamp vector (by *pv1*), then, after delivering m1, by *A3,* $LDV_j(i)$ is updated to m1.b and then will be larger than m1.LDV[i]. Therefore, by *A1*, the inequality m2.LDV > m1.LDV will hold. Because $LDV_j(i)$ is updated to m1.b before being used as m2's timestamp, then m1.b ≤ m2.LDV[i] will also hold. Hence, it follows that if m1 → m2, then m1.LDV < m2.LDV and m1.b ≤ m2.LDV[i]. The reverse of *pr2*, that is, if m1.LDV < m2.LDV and m1.b ≤ m2.LDV[i], then *m1 → m2,* follows from the fact that when m2 is timestamped to be transmitted, all messages with block-numbers up to those present in m2.LDV will have already been delivered in p_j, including *m1* (note that *m1.b ≤ m2*.LDV[i]). Thus, if m1.LDV < m2.LDV and m1.b ≤ m2.LDV[i], then indeed *m1 → m2*.

Notice that by using LDV vectors as timestamps, the potential causal relationship between messages is as precisely represented as in vector clocks [BSS91]. In practice, this precise representation of potential causality makes possible fast causal message delivery. That is, the delivery of a message *m* is only delayed if there is another message m' such that *m' → m* and *deliver(m')* has not happened yet.

If two messages *m1* and *m2* are not causally related they are said to be concurrent messages, and we will denote m1 || m2. Concurrent messages can be precisely defined by their timestamps as follows:

pv3 : m1 || m2 ⇔ neither {m1.LDV < m2.LDV and m1.b ≤ m2.LDV[i]}
nor {m2.LDV < m1.LDV and m2.b ≤ m1.LDV[j]} holds.

If m1 || m2, neither m1 → m2 nor m2 → m1 hold. In the former case, when m2 was sent m1 had not been delivered yet. Thus, by *A3*, $LDV_j[i]$ will only progress to m1.b when m1 is delivered in p_j. Thus, m2.LDV[i] < m1.b. The same reasoning applies for the latter case (when m2 → m1 does not hold).

5.2 Causal Order in Overlapping Process Groups

When process groups overlap, a message m received by a multi-group member may causally depend on messages sent in other groups distinct from the one m was sent to. Thus, timestamping a message with a single group vector (such as LDV or Vector Clocks) is not enough to represent causal information. Consider two groups g_x and g_y, such that $g_x \cap g_y \neq \emptyset$ (i.e. g_x and g_y overlap). A message m sent in g_x by a process in $g_x \cap g_y$, may depend on previous messages sent in g_y. Thus, m has to carry causal information not only about g_x but also about g_y so that any other member of $\{g_x \cap g_y\}$ will be able to deliver m properly.

Overlapped groups can be represented as a graph of groups whose structure can vary in time due to the changes on the group memberships. Consider the existence of process groups $g_i = \{p_1, p_2, ... ,p_n\}$. The graph O of overlapped groups is defined as follows: one vertice of O represents a given process group, and there will be an edge linking two vertices whenever the corresponding groups overlap. That is, O = (V,E), where V is the set of all groups g_i and $(g_j,$

$g_k) \in E$ iff $g_j \cap g_k \neq \varnothing$. In the particular case where there is no cycles in O, it suffices for a message m to carry causal information about all groups the sender of m is a member of. However, in the general case, causal dependences may propagate through cycles [BSS91] and messages have to carry global causal information (i.e., causal information of all groups in O). For instance, consider the 4 groups $g_1 = \{p_1, p_2, p_3, p_4\}$, $g_2 = \{p_3, p_4, p_5, p_6\}$, $g_3 = \{p_5, p_6, p_7, p_8\}$, and $g_4 = \{p_1, p_2, p_7, p_8\}$. The graph O of overlapping groups (figure 3) is given by $V = (g_1, g_2, g_3, g_4)$ and $E = \{e_1 = (g_1, g_2), e_2 = (g_2, g_3), e_3 = (g_3, g_4), e_4 = (g_1, g_4)\}$. Consider that p_1 sends the message m_1 in g_1. After delivering m_1, p_3 sends m_2 in g_2. Then, p_6 delivers m_2 and sends m_3 in g_3. And finally, p_7 delivers m_3 and sends m_4 in g_4. Since $m_1 \rightarrow m_4$, p_2 has to deliver m_1 before m_4. Unless the information '$m_1 \rightarrow m_2 \rightarrow m_3$' is passed through the groups to p_7, it cannot deduce '$m_1 \rightarrow m_4$' and send this causal information together with m_4.

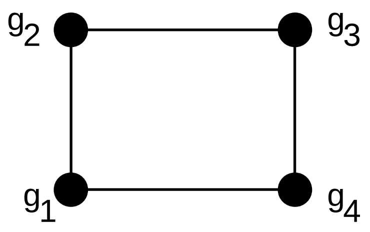

Figure 3 - Graph O of overlapped process groups

5.3 The Slow Causal Order Protocol

In the slow causal order protocol, a received message m can immediately be delivered, provided that all blocks BM[B], B < m.b, are complete and its messages delivered. Consider process p_i, and let $G_i = \{g_x \mid p_i \in g_x\}$. To run the protocol, a process p_i has - in addition to BC_i and a $BM_{x,i}$ for each g_x in G_i - a vector, called the Completed Blocks Vector and denoted as CBV_i, which is of size $|G_i|$ and is maintained in the following manner: for each g_x in G_i, $CBV_i[x]$ indicates the largest number of complete blocks in $BM_{x,i}$. Process p_i also maintains the value of $\min\{CBV_i[x] \mid g_x \in G_i\}$ in a variable B_{min}. CBV_i is initialized to [0,0,...,0], B_{min} to 0, and every time a '+' is added to any $BM_{x,i}$ both are updated if possible. Figure 4 shows the simplified msg-delivery sub-process intended for causal order delivery.

```
msg-delivery: /* executed after a message has been received */
{
 do
   if $B_{min} = \min\{CBV[x] \mid g_x \in G\} \rightarrow$
        deliver all sent and received, but not delivered m such that $m.b = B_{min} + 1$
        if $m.b > BC_i \rightarrow BC_i := m.b$; /* update the block-counter */
     /* the received message completed one or more blocks */
         $B_{min} := \min\{CBV[x] \mid g_x \in G\}$;
         deliver all sent and received, but not delivered m such that $m.b \le B_{min} + 1$
         if $m.b > BC_i \rightarrow BC_i := m.b$; /* update the block-counter */
     fi
   od
}
```

Figure 4 - The msg-delivery sub-process for slow causal order delivery

In figure 4, B_{min} represents the largest block-number of complete (and delivered) Causal Blocks for all groups p_i is a member of. Thus, any received message with block-number $B_{min} + 1$ can immediately be delivered to p_i with the assurance that all possible causal dependent messages have been already delivered. On the other hand, messages with block-numbers $B_{min} + K$, $K > 1$, will have to wait for block-completion to occur. Otherwise, causal order may be violated. For instance, if a message m, $m.b = B_{min} + 5$, is received, it can not be immediately delivered since another message m', $m'.b = B_{min} + K$, $1 \le K \le 4$ and $m.s \ne m'.s$, can be subsequently received. Therefore, m will have to wait for the block $BM[B_{min} + 4]$ to become complete in order to be safely delivered. When more than one block get complete, messages are delivered following the increasing order of their block-numbers. Safety and liveness correctness proves for the Slow Causal order protocol can be found in [Mac94].

5.4 The Fast Causal Order Protocol

To obtain fast causal order delivery, we extend the group based Last Delivered Vector presented in section 4.1 into a Global Last Delivered Vector suitable for a multi-group environment. Consider the existence of distinct process groups. In order to construct the GLDV vector, each process maintains a unique Block-Counter BC, despite the number of groups it may belong to. To represent sent/received messages, each process p_i maintains a Block-Matrix $BM_{i,x}$ for each group g_x that p_i is a member of. Updating BC_i follows the rules CA1 and CA2. Also, each process p_i maintains a Global Last Delivered Vector ($GLDV_i$) for keeping the largest block-number of messages sent/delivered by all processes (belonging to groups p_i is a member of or not).

Suppose the existence of a multi-group process p_i. Thus, $GLDV_i$ is initialised with the entries corresponding to all processes belonging to groups p_i is also a member of. Transmitted messages are timestamped with the current value of $GLDV_i$. So, before p_i transmits a message m, it assigns the current value of $GLDV_i$ to m.gldv. Just after m is sent, the vector entry corresponding to p_i is updated with m.b (i.e. $GLDV_i$ [i] = m.b). When m is received by another process, say p_j, the local $GLDV_j$ is updated to include all entries in m.gldv that are not present in $GLDV_j$. So, actually, $GLDV_j$ will maintain the largest block-number of messages sent/delivered by processes of groups in the overlapping structure O. To simplify presentation, let us assume that when a given GLDV is initialised, there will be an entry in it for each process in O and they will be set to zero before any message is transmitted or delivered. Notice that like the LDV vectors (section 5.1), the GLDV vectors are constructed by maintaining the largest block-number of messages sent/delivered by processes. Thus, the comparison rules and properties of LDV vectors presented in section 5.1 can also be applied for GLDV vectors. Consider a given process p_i, and let $G_i = \{g_x \mid p_i \in g_x\}$. Let S be the set of all processes belonging to groups p_i is also a member of. That is, $S = \{p_k \mid p_k \in g_x \text{ and } g_x \in G_i\}$. On receiving a message m, p_i must proceed with the following test for delivery:

Delivery test : if ($\forall\, p_k \in S$, $GLDV_i[k] \geq$ m.gldv[k]) then deliver(m);

After delivering the message m, $GLDV_i$ is updated as follows:

```
Updating GLDV:
    for any process $p_k \notin S$, $GLDV_i$ [k] := max(m.gldv[k], $GLDV_i$[k]);
    $GLDV_i$[m.s] := m.b;
```

The delivery test guarantees that a received message m is only delivered after all messages represented in m.gldv, corresponding to those processes in S, have already been delivered by p_i. After delivery of m, the entry of $GLDV_i$ corresponding to the sender of m (m.s) is updated with the block-number of the message delivered (m.b). The other entries of $GLDV_i$ not corresponding to processes in S, will keep the maximum values between the values in $GLDV_i$ and m.gldv, so that a future transmitted message m' by p_i will carry in m'.gldv all inter-group causal information known by p_i by the time m' is multicast.

Message space overhead when using the GLDV vectors

Consider the existence of a set of process groups whose overlapping structure is given by the graph O, previously defined. The size of GLDV, |GLDV|, corresponds to the number of processes participating in groups in O. It can be trivially verified that $|GLDV| < \sum |g_i|,\ g_i \in V$, if $E \neq \emptyset$, where $|g_i|$ is the size of group g_i. In CBCAST [BSS91], a message is timestamped with |V| vector clocks (i.e. the number of groups in O), leading to a message space overhead of

$\sum |g_i|$, $g_i \in V$. Thus, the CBCAST[5] overhead corresponds to the GLDV upper-bound overhead, that will only happen when there is no overlapping in O ($E = \emptyset$). Hence, the more groups overlap, the smaller is the overhead of Fast causal protocol using GLDV when compared with the CBCAST.

5.5 The Relative Causal Order Protocol

We now combine the group based LDV vectors with the concept of block-completion to develop a trade-off solution. We will first describe the protocol for the non-overlapping scenario. We will then extend the protocol to work with overlapping process groups, proving its correctness.

To represent ordering causal information between messages sent in a given group, we will make use of the Last Delivered Vector (LDV) introduced in section 5.1. Thus, a process p_i will maintain a Block Counter $BC_{x,i}$ and a Block Matrix $BM_{x,i}$ for every group g_x, it is a member of. The updating of $BM_{x,i}$ and $BC_{x,i}$ will follow the conditions *CA1* and *CA2* restricted to a given group. Besides that, p_i will also maintain a Last Delivered Vector - LDV_i, for each group p_i is a member of. $LDV_{x,i}$ will be updated as specified in the conditions A1, A2, and A3 of section 4.1, and will be used to precisely represent causality between messages sent in a given group (see property *pv2* of section 5.1).

Every message multicast will be timestamped with, in addition to m.b, the current value of the LDV_i vector as m.ldv. The algorithm of the send(m) primitive for a process p_i is given below, in figure 5, where g_x represents the destination group of message m, m.g.

```
send(m: non-null-message);
{
  m.s := i;
  m.b := $BC_{x,i}$ + 1; $BC_{x,i}$ := m.b;
  m.ldv := $LDV_{x,i}$;
  $LDV_{x,i}$[i] := $BC_{x,i}$;
  $BM_{x,i}$[m.b] := '+';
  multicast m to other members of m.g;
}
```

Figure 5 - The send primitive

5. Without considering the compression mechanisms presented in [Birman91b].

The receive process (figure 6) will receive messages from the transport layer and represent them in the appropriate Block Matrix. After that, it signals the deliver process which will try then to deliver the received message. Delivering a message creates conditions for received (and not delivered) messages to be delivered. So, after a message has been delivered, the deliver process is recursively activated until there is no message that can be delivered. A received message m taken from the transport layer and intended for p_i will be immediately represented in the block matrix BM_i related to the group m.g but delivery will only happen when the m.ldv is greater than or equal to $LDV_{i,x}$ (g_x = m.g). Figures 6 and 7 show the algorithmic descriptions of the receive and deliver sub-processes, respectively.

```
msg-receive;
{
  do
    receive m from the transport layer;
    update appropriate BM; /* $BM_{i,x}$[m.b] := m; */
    signal msg-delivery sub-process; /* signal the deliver sub-process */
  od
}
```

Figure 6 - The msg-receive sub-process

```
msg-deliver;
{
   if $\exists$ m in $BM_{i,x}$ | m is not marked delivered and m.ldv $\geq$ $LDV_{i,x}$ $\rightarrow$
     {
        mark m delivered and deliver m to $p_i$;
        signal msg-deliver sub-process; /* try recursively to deliver another message */
     }
}
```

Figure 7 - The msg-deliver sub-process

Maintaining inter-group causal information

Each process p_i maintains a block-counter $BC_{i,x}$ and a block-matrix $BM_{i,x}$ for each group g_x process p_i is a member of. Updating of the Block-Counter and Block-Matrix follows the same rules for the non-overlapping case described in the last paragraph. Thus, $BC_{i,x}$ maintains the largest block-number of messages sent or delivered in group g_x and received/sent messages of g_x are represented in $BM_{i,x}$. Just before a message m is sent in group g_x, it is timestamped with the incremented value of $BC_{i,x}$ as m.b. A destination process, say p_j, on receiving m will know that when m was sent group g_x had progressed in logical time up to block-number m.b. Thus, any message m', $m \rightarrow m'$, subsequently transmitted by p_j to a group g_y, $y \neq x$, will also carry this inter-group causal information so that any process $p_r \in g_x \cap g_y$, $r \neq j$, will deliver m' only after the Causal Block $BM_{r,x}$[m.b] is complete and its messages delivered. Therefore, enforcing the delivery of m before delivery of m'.

For assessing which Causal Blocks per group should be complete before a received message can be delivered, processes will maintain a vector containing the block-numbers of the largest sent/delivered message per group (the GBN vector), and this information will be transmitted together with any sent message.

The GBN vector

Let $GBN_i[1..k]$ be the process p_i's vector of Group Block Numbers, where k is the number of groups[6]. $GBN_i[j]$ represents the block number of last message sent/delivered in group g_j. Every message sent by process p_i will carry in m.gbn the current value of GBN_i. GBN_i is set up with zeros when p_i starts and will be updated in the following way:

Just after a message m is sent by p_i,

GBN_i[m.g] := m.b*;*

Just after p_i delivers a message m,

$GBN_i[j] := \max\{GBN_i[j], m.gbn[j]\}$, $j \neq m.g$;

GBN_i[m.g] := m.b;

By using the vector GBN, we can now state the message delivery conditions for a given multi-group process p_i. A received message m of g_x can only be delivered if the two following conditions *C01* and *C02* are true.

Conditions for Causal Ordering

CO1 : $m.ldv \leq LDV_{x,i}$;

CO2 : m.gbn[y] is complete in $BM_{y,i}$, and its messages delivered, for all g_y such that p_i is a member of and $y \neq x$.

By applying the above conditions, protocols satisfying causal order message delivery can be developed. For proving the correctness of the protocol, we have to show that a sent message is eventually delivered (liveness property) and it does so without violating causal order (safety property). These proves can be found elsewhere [Mac94].

6. In fact, k changes dynamically. Thus, in a real implementation, GBN will be a list of (Gid, BN) pairs, where BN is the block message of the group identified by Gid.

6 Concluding remarks

We have presented three approaches for causal order message delivery in overlapping process groups. They represent different trade-offs between delivery delay and message space overhead costs. The Slow causal protocol requires the smallest timestamp (the message block-number) but can potentially introduce extra delays since Causal Blocks have to be complete before message delivery. The Fast causal protocol is based on the GLDV vector which is a precise representation of causal dependence between transmitted messages in a multi-group environment. In Fast causal, message delivery occurs as early as possible. That is, the delivery of a message m is only delayed if there is a message m', such that m' $\rightarrow$ m and m' has not been delivery yet. On the other hand, the Fast causal protocol imposes the highest message space overhead among the approaches presented. The Relative causal protocol is a trade-off solution. It provides as early as possible delivery in a uni-group environment but some extra delay is possible in a multi-group environment due to the need of block-completion. When compared with the Fast causal protocol, it provides a smaller timestamp with a potentially slower delivery time. When compared with The Slow causal protocol, the Relative causal provides a faster message delivery but with a higher message space overhead.

To our best knowledge, in the existing literature published up to date, causal order protocols for overlapping groups have only been addressed in [BSS91b, MR93]. When complex group structures are considered, the protocol in [BSS91b] can lead to a large message space overhead: vector clocks for all groups (no matter whether the sender process belongs to them or not) have to be transmitted with any transmitted message. The protocol presented in [MR93] optimizes message space overhead, transmitting only one integer (rather than a vector) per group. However, it imposes delay overhead at all destination processes (re-synchronization messages have to be transmitted so as the local logical clocks of destination processes reach the values indicated in the integers contained in the transmitted messages), even if a given destination process belongs only to one group . The Relative causal order protocol transmits one integer per group and also one vector (the LDV vector) corresponding to the destination group of the transmitted message. Thus, while it optimizes message space overhead, it will not impose any extra delay at destination processes which do not belong to multiple groups, for the LDV vector will allow delivery as early as possible.

An implementation of the Relative causal order protocol is underway. We expect that this implementation together with other mechanisms based on the Causal Blocks representation (a total order protocol[7], a membership protocol, and a flow control scheme [Mac94]), will form a solid substratum for the development of reliable distributed applications.

[7] Messages are delivered in the same global order and respecting causality. A prototype of the total order protocol together with implemented and tested over a set of networked UNIX machines.

References

[ADKM92]

Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki, "Transis: A Communication Sub-System for High Availability", 22nd International Symposium on Fault-Tolerant Computing (FTCS-22nd), Boston, July 1992.

[BJ7]

Birman, K. and Joseph, T. "Reliable Communication in the Presence of Failures", ACM Transactions on Computers Systems 5, 1 (Feb. 1987)

[Bir91]

Kenneth P. Birman, "The Process Group Approach to Reliable Distributed Computing", Technical Report TR91-1216, Department of Computer Science, Cornell University, July, 1991.

[BSS91]

K. Birman, A. Shiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", ACM Transactions On Computer Systems, Vol. 9, No 3, August 1991, pp. 272-314.

[Cha91]

B. Charron-Bost, "Concerning the size of Logical Clocks in Distributed Systems", Inf. Proc. Letters, Vol. 39, (1991), pp. 11-16.

[EMS95]

Paul Ezhilchelvan, Raimundo A. Macêdo, Santosh K. Shrivastava, "Newtop: a Fault-Tolerant Total Order Multicast Protocol", 15th International Conference on Distributed Computing Systems, Vancouver-Canada, June 1995. IEEE.

[Fid91]

C. J Fidge, "Logical time in distributed computing systems, IEEE Computer, Vol. 24,8 (1991).

[Lam78]

Lamport, L., "Time, clocks, and ordering of events in a distributed system", Commun. ACM, 21, 7 (July 1978), pp. 558-565.

[MES93a]

Raimundo A. Macêdo, Paul Ezhilchelvan, Santosh K. Shrivastava, "Modelling Group Communication using Causal Blocks", 5th European Workshop on Dependable Computing, Lisbon, February, 1993.

[MES93b]

Raimundo A. Macêdo, Paul Ezhilchelvan, Santosh K. Shrivastava, "Newtop: a Total Order Multicast Protocol Using Causal Blocks", Broadcast deliverable report, Volume I, First open Broadcast workshop, Newcastle, october, 1993.

[Mac94]

Raimundo A. Macêdo, "Fault-Tolerant Group Communication Protocols for Asynchronous Systems", Ph.D. Thesis, Department of Computing Science, University of Newcastle upon Tyne, 1994.

[Mat89]

F. Mattern, "Time and global states in distributed systems", In Proc. of the International Workshopom Parallel and Distributed Algorithms, North-Holland, Amsterdam, 1989.

[MR93]

Achour Mostefaoui and Michel Raynal, "Causal Multicasts in Overlapping Groups: Towards a Low Cost Approach", In Proc. of the 4th IEEE Int. Conference on Future Trends of Distributed Systems, pp. 136-142, Lisboa, September 1993.

[PBS89]

L. L. Peterson, N. Bucholz, and R Schlichting, "Preserving and using context information in interprocess communication", ACM Transactions on Computer Systems, Vol. 7, No 3, August 1989, pp. 217-246.

[Ray92]

Michel Raynal, "About Logical Clocks for Distributed Systems", Operating Systems Review, SIGOPS, vol. 26, Number 1, January, 1992.