A lightweight asynchronous algorithm for causal delivery using extra message intertion.

Authors:

Cesar Sanchez
<Cesar.Sanchez@ACM.org>
Stanford University

Angel Alvarez
<Angel.Alvarez@ACM.org>
Technical University of Madrid-Spain

Contact information:

Cesar Sanchez
PO Box 12334
Stanford, CA 94309
USA

Angel Alvarez
ETSI Telecomunicacion - UPM
Ciudad Universitara s/n
Madrid
Spain

A lightweight asynchronous algorithm for causal delivery using extra message insertion

César Sánchez
<Cesar.Sanchez@ACM.org>
Stanford University

Ángel Álvarez
<Angel.Alvarez@ACM.org>
Technical University of Madrid-Spain

Abstract

This paper presents an algorithm to solve the problem of causal delivery with messages of size bounded to $O(k)$ where the parameter $n < k \leq n^2$ is a constant fixed by the programmer. The algorithm makes use of the concept of matrix clocks. It works by inserting some extra messages in the network in a clever manner to create stronger conditions than causal delivery. These extra messages simplify the clock matrices, which can then be coded even in linear space if so desired. The drawback of the algorithm is that message delivery may sometimes get delayed waiting for these extra messages to arrive, even if all the conditions strictly imposed by the causality of the application are satisfied. The paper proves the correctness of the algorithm presented and compares it with other solutions proposed for the causal ordering problem.

Keywords: Asynchronous distributed systems, logical clocks, causal order, causal delivery, matrix clocks.

1 Previous work

A pioneer work in distributed systems with groups of processes was ISIS [1], which was one of the first systems to include causal message ordering. The first implementation of ISIS included all the information about the causal past in every message, and thus the size of messages was in principle unbounded, short of regularly pruning the causal past. In [2] Raynal, Schiper and Toueg proposed an elegant extension of the logical [3] and vector [4] clock algorithms that solved the problem with bounded size messages. In the original algorithm with matrix clocks, the size of messages was $\Theta(n^2)$, n being the number of processes. This high space complexity can be a problem for practical implementations.

Shingal and Ksenkalyany [5] proposed an alternative method that reduced the average complexity but still had a worst case space complexity of $\Theta(n^2)$. Mattern [6] proposed an algorithm that drastically reduced the size but with the penalty of a tight synchronization and therefore lower network utilization and concurrency.

2 Definitions

We consider an asynchronous distributed system as a set of processes that can communicate only by sending and receiving messages through a network. We assume that the network does not duplicate messages and that every message will eventually arrive at its destination. And no other assumptions, such as FIFO message ordering, exist.

In particular, we assume there is no global clock or a bound in the drift among processes' local clocks. We consider a set of events of interest that happen inside every process. This set is assumed to be totally ordered according to the local clock of the process. We use S_i to denote the set of events in the computation of process P_i, and $\prec_i$ to refer to the ordering relation among the events of S_i.

We consider the set of messages M sent in a given computation. Given a message $m \in M$ we refer to $send(m)$ as the event of sending the message M and to $recv(m)$ as the event of delivering the message at its destination. We also use $from(m)$ and $to(m)$ to refer to the processes which send and receive m, respectively. Consequently, $send(m)$ is an event local to process $from(m)$ while $recv(m)$ is local to $to(m)$. We refer to Γ as the partial order relation imposed by the fact that $send(m)$ happens before the $recv(m)$. That is, for each message $m \in M : (send(m), recv(m)) \in \Gamma$.

Definition 1 (happened before relation). *Given two events $e_1, e_2 \in S$ we say that e_1 happened before e_2, and we use the notation $e_1 \prec e_2$, if*

- $(e_1, e_2) \in \bigcup_i \prec_i$
- $(e_1, e_2) \in \Gamma$ *, or*
- $\exists c \ : \ (e_1 \prec c) \wedge (c \prec e_2)$.

In a real computation the relation $\prec$ is an irreflexive partial order.

Definition 2. *A computation respects causal order when $\forall m, m' \in M$:*

$$\left.\begin{array}{l} to(m) = to(m') \\ send(m) \prec send(m') \end{array}\right\} \Rightarrow recv(m) \prec recv(m')$$

3 Lightweight algorithm using message insertion

In the algorithm presented in [2] every process P_k stores a matrix $M_k : n \times n$ and sends this matrix attached to every message. P_k also keeps track of a vector $DELIV_k[1..n]$ that stores the number of messages sent from every other process to P_k and delivered so far. The initial values of both M_k and $DELIV_k$ are zero.

Informally, $M_k[i, j]$ represents the knowledge of process P_k about the number of messages sent by P_i to P_j. It can also be interpreted as the number of messages sent from P_i to P_j in the current causal past.

The values evolve as follows:

- When a message is sent from P_i to P_j, first M_i is attached to the message, and then $M_i[i, j]$ is incremented by one.

- When a message is received, the receiving process P_j checks whether it has delivered all the past messages necessary to ensure causal order, namely the column $M_m[\_, j]$ in message m. If this check is not satisfied then the message is enqueued until all the missing messages have arrived.
- When a process P_j delivers a message m sent from P_i, it updates its local M_j with the maximum values, $M_j[x, y] = max\{M_m[x, y], M_j[x, y]\}$, and increments $DELIV_j[i]$ by one. We will represent this kind of multi-cell matrix updating as $M_j[\_, \_] = max\{M_m[\_, \_], M_j[\_, \_]\}$.

This simple and elegant algorithm can be understood from another point of view.

The matrix that a process maintains and sends is equivalent to a set Φ of "constraints". These constraints are logical predicates that can be evaluated. If upon receiving a message all the constraints carried by it are satisfied (all the predicates evaluate to true), then the message is delivered. Otherwise, the message is enqueued until all the constraints are satisfied.

In the case of the matrix algorithm the set of constraints can be expressed as $\Phi = \{\phi_{ij} : PID = P_j \Rightarrow DELIV_j[i] \geq M[i, j]\}$. Then, every entry in the matrix is equivalent to a constraint, and the algorithm proposed in this paper is derived from the following observations:

- Upon receiving a message a given process P_k only needs to evaluate the constraints $\{\phi_{ik} | \forall i\}$ since the others are trivially satisfied (the antecedent is false).
- All the zeros in the matrix will be trivially satisfied in every receiver (i.e. they correspond to true constraints) because $\forall j \; DELIV_j[i] \geq 0$, since it starts with 0 and increases monotonically as the computation proceeds. As a result, there is no need to store and send these trivial constraints.

It follows from the above that if the matrix is sparse enough, a good coding method can be used, for example a tuple (i, j, n) for every $M[i, j] = n$ with $n > 0$.

Our algorithm forces zeros in the matrices in the following ways:

- The column of the matrix of any process P_k related to itself (namely $M_k[\_, k]$) is always zero. In the basic matrix algorithm, $M_k[\_, k] = DELIV_k[\_]$; all the constraints imposed by this column have already been satisfied, so there is no need to keep track of them any longer.
- After a message m is sent from P_i to P_j, the column $M_i[\_, j]$ is set to zeros except the value indicating the message m, $M_i[i, j]$. This can be done safely because the delivery of m will imply the satisfaction of the constraints associated with $M_i[\_, j]$.
- The original idea of our proposal is to *force* stronger constraints than the ones implied by the flow of the computation, when necessary. Informally, when a process P_i detects that its matrix M_i has too many non-zero values, it selects the process P_j that is causing the most "uncertainty" (most non-zero values in its column of the matrix) and sends an extra message to this process. This message only contains the constraints that the receiver should fulfill, i.e. $M_i[\_, j]$. After sending this extra message, the sender deletes that column with the exception of the value indicating this message, the same as for regular user messages. The process has a

threshold value k ($n < k \leq n^2$) that represents the maximum number of non-zero entries a process allows in its matrix clock before it starts clearing entries by sending extra messages.

Note that the extra messages do not belong to the happened before relation $\prec$ and are only used to force additional constraints.

As a particular case, consider the algorithm with threshold parameter $k = n+1$. Whenever a process P_i detects that its matrix clock contains $n+1$ non-zero entries it knows that at least one column contains at least two non-zero entries. P_i then selects the column j with the maximum number of non-zero entries, and sends a message to P_j with that column attached. It continues by deleting the column $M_i[\_, j]$ with the exception of entry $M_i[i, j]$, which is incremented to indicate the extra message. The constraint set that this new matrix represents is stronger than the set represented by the previous matrix, for any evaluation in the causal future.

4 Proof of correctness

We prove safety and liveness separately. We use $M_1 \leq M_2$ as the usual $\leq$ between matrices. Given an event e and the sets of constraints Φ_1 and Φ_2, we say $\Phi_1 \Rightarrow_e \Phi_2$ whenever the satisfaction of all the constraints Φ_1 implies the satisfaction of all the constraints Φ_2, for any time in the casual future of e. If Φ_2 holds whenever Φ_1 holds, no matter what moment of time, we say $\Phi_1 \Rightarrow \Phi_2$.

Safety We first prove the following claims:

Proposition 1. *Let M_1 and M_2 be the matrices corresponding to the sets of propositions Φ_1 and Φ_2. If $M_1 \geq M_2$ then $\Phi_1 \Rightarrow \Phi_2$.*

Proof. Consider a process P_i, and a time in its execution such that Φ_1 is satisfied. Then $DELIV_i[\_] \geq M_1[\_, i]$ because Φ_1 is satisfied. As long as $M_1 \geq M_2$, $DELIV_i[\_] \geq M_2[\_, i]$, so P_i also satisfies Φ_2, as we wanted to show. .■

Lemma 2. *Let $e \in S_i$ be an arbitrary event in P_i, Φ_1 be the set of constraints stored in P_i before e, and Φ_2 be the set of constraints after e. Then $\Phi_2 \Rightarrow \Phi_1$.*

Proof. Let M_1 be the matrix associated with Φ_1 and M_2 the matrix associated with Φ_2. We now consider the different possible cases of e separately:

- e is the delivery of a user message m from P_j. Then $M_2[\_, l] = max\{M_1[\_, l], M_m[\_, l]\} \quad \forall l \neq i$ and $M_2[\_, i] = M_1[\_, i] = \bar{0}$. Therefore $M_2 \geq M_1$. By proposition 1, $\Phi_2 \Rightarrow \Phi_1$.
- e is the delivery of an extra message m from P_j. Here $M_2 = M_1$, and therefore $M_2 \geq M_1$. By proposition 1, $\Phi_2 \Rightarrow \Phi_1$.
- e is the sending of a user or extra message m to P_j. Then $M_2[\_, j] = 0$ except $M_2[i, j] = M_1[i, j] + 1$, and $M_2[\_, l] = M_1[\_, l]$ for $l \neq j$. Consider now an arbitrary process P_k where Φ_2 is satisfied. We distinguish two cases:
  - $P_k \neq P_j$. Let us take an arbitrary constraint in Φ_1. If it is not related to P_k then it is trivially true. If it is related, let l be such

that ϕ_{lk} is that constraint. We know that $M_2[l,k] = M_1[l,k]$ because $k \neq j$, and hence $\phi_{lk} \in \Phi_2$. Since Φ_2 is satisfied then ϕ_{lk} is satisfied. Therefore the whole set Φ_1 is satisfied in P_k.

– $P_k = P_j$. The only nontrivial constraint related to P_j in Φ_2 is ϕ_{ij} (the rest of $M_j[\_, j]$ has been reset to 0 by the algorithm). The satisfaction of this constraint implies that m has been delivered. Consequently, the constraints that were attached to m have been satisfied, and so $\{\phi_{lj} \in \Phi_1, \forall l\}$ is satisfied. Therefore Φ_1 is satisfied in P_k. .■

Lemma 3. *Let $m \in M$ be a user message and let Φ_{sent} be the set of constraints before send(m), Φ_m the ones carried in m, and Φ_{deliv} the ones after the delivery of the message. Then $\Phi_{deliv} \Rightarrow_{recv(m)} \Phi_m \Rightarrow \Phi_{sent}$.*

PROOF. $\Phi_m \Rightarrow \Phi_{sent}$ is trivial since $M_{sent} = M_m$. Now, let P_j be $to(m)$. After the delivery $M_{deliv}[\_, l] = max\{M_{prev}[\_, l], M_m[\_, l]\} \ \ \forall l \neq j$. Moreover, all $M_m[\_, j]$ are satisfied at any time in the causal future of $recv(m)$ (because in any other process they are trivially satisfied, and in P_j, $DELIV_j[\_] \geq M_m[,j]$ and monotonically increasing). Consequently, given a moment in the future of the delivery, if Φ_{deliv} is satisfied then Φ_m is satisfied. Therefore $\Phi_{deliv} \Rightarrow_{recv(m)} \Phi_m$ as we wanted to show. .■

Now we are ready to prove safety:

Theorem 4 (safety). *Let $m, m' \in M$ with $send(m) \prec send(m')$ and $to(m) = to(m')$. Then $recv(m) \prec recv(m')$.*

PROOF. It can be proven that if $a \prec b$ then there is a causal channel between a and b, that is, a succession of events $\{e_i\}_{i=1..n}$ with $e_1 = a$, $e_n = b$, such that either

- $e_i = send(m), e_{i+1} = recv(m)$ for some m, or
- $e_i \prec_k e_{i+1}$, for some process P_k and $\nexists c \in S_k \ : \ e_i \prec_k c \prec_k e_{i+1}$.

Consider the causal channel $\{e_i\}$ between $send(m)$ and $send(m')$. The two possibilities are:

- If $e_1 = send(m)$ and $e_2 = recv(m)$ then $recv(m) \prec send(m')$. Hence, since $send(m') \prec recv(m')$ we have $recv(m) \prec recv(m')$ as we wanted to show.
- If $e_1 \prec_k e_2$ it is possible to show by induction over the length of the causal channel that $\Phi_{e_n} = \Phi_{send(m')} \Rightarrow_{recv(m')} \Phi_{e_1}$ (using lemmas 2 and 3 appropriately). Therefore, immediately after $recv(m')$ all the constraints in Φ_1 are satisfied, and particularly m must have been delivered. As long as no event occurred after the delivery of m' (we are reasoning about *right after*), and $to(m) = to(m')$ we have that $recv(m') \not\prec_j recv(m)$. Then, as long as $\prec_j$ is a total order $recv(m) \prec_j recv(m')$ as we wanted to show. .■

Liveness The following proposition is useful to prove liveness.

Proposition 5. *Let M be the message number n sent from P_i to P_j. If $DELIV_j[i] < n$ then M has not been delivered to P_j.*

Theorem 6. *The algorithm satisfies liveness.*

PROOF. Consider the relation $<$ between messages (user or extra) defined by: $m < m'$ iff $send(m) \prec send(m')$. Now, for the sake of contradiction assume there is a process P_i with M_i the nonempty set of messages that cannot be delivered at the end of the computation. The relation $<$ defined above is a nonempty finite partial order, so consider m a lower bound of M_i. Let ϕ_{ki} be a constraint that is not satisfied by P_i (there has to be some, otherwise m would have been delivered), and call $n = M_i[k, i]$. This means that $DELIV_i[k] < n$. Then consider m_n the n-th message sent from P_k to P_i. From proposition 5, m_n has not been delivered to P_i. It can be proven from $\phi_{ki} = n$ that $send(m_n) \prec send(m)$. This implies that $m_n < m$, which contradicts the fact that m is a lower bound of the non-delivered messages into P_i. .■

5 Future work

Some of the lines worth exploring in the future include:

- Check the impact of the number of extra messages in practice. Probably a schema like the one used in [7] can be used to reduce the number of extra messages inserted in the network.
- Evaluate the total size of the messages in a given computation in relation to the total size of the so called *optimal* algorithm proposed in [5].
- Make an analysis of the influence of using different values of k for different processes in the system.
- Evaluate the penalty (in time) in practical computations introduced by the need to wait for the arrival of the extra messages.

References

[1] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, Aug 1991.

[2] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inf. Processing Letters*, 39:343–350, 1991.

[3] L Lamport. Time, clocks and the ordering of events in a distributed system. *Comm. of the ACM*, 7(21):558–565, 1978.

[4] F. Mattern. Time and global states of distributed systems. In *Proc. Int'l Workshop on Parallel and Distributed Algorithms*, pages 215–226, Bonas, France, 1988. North-Holland, Amsterdam.

[5] M. Singhal and A. D. Kshemkalyani. Necessary and sufficient conditions for causal message ordering and their optimal implementation. Technical Report CISRC-7/95-TR33, Dep. of Computer Science, Ohio State University, Jul 1995.

[6] F. Mattern and S. Fünfrocken. A non-blocking lightweight implementation of causal order message delivery. Technical Report TR-VS-95-01, Dep. of Computer Science, Technical Univ. of Darmstadt, Mar 1995.

[7] A. Mostefaui and O. Theel. Reduction of timestamp sizes for causal event ordering. Technical Report 1062, IRISA, Nov 1996.