Causal Delivery with Vector Clocks

0 Introduction

In a relatively recent publication [1] an algorithm is presented for the problem known as *causal delivery*, but in Section 4.11 its authors commit a major logical blunder when they discuss what they call "Weak Gap-Detection"; this is a property of the shape $P \Rightarrow Q$, whence they conclude: "Thus, no undelivered message exists [which is $\neg Q$] if [an expression for $\neg P$].". Because of this blunder, and because of other reasons for dissatisfaction with this presentation, among which lack of explicitness about the rules of the game and a nomenclature that is far from sober, I decided to try to solve this problem on my own. Hence this note.

The logical error can be a symptom of two causes: either the algorithm is correct, but its presentation is not, or both are incorrect. It so happens that my solution yields essentially the same algorithm, save minor differences. Surprisingly enough –for me, as a layman in this area–, I do not encounter the slightest need to formulate such a thing as a "Gap-Detection" property.

Although I have tried to formalize my presentation no more than required for the sake of clarity, it is an order of magnitude more formal than in [1].

1 Preliminaries

In order not to disrupt the flow of the main story, we first introduce a few concepts that turn out to be useful.

1.0 Closed sets

We call a subset V of some fixed universe *closed* with respect to a given binary relation $<$ (on that universe), iff for every element y in V all values less than y are elements of V as well; in formula:

$$closed{\cdot}V \quad \equiv \quad (\forall x,y : y{\in}V : x{<}y \Rightarrow x{\in}V) \quad .$$

Simple properties are that the universe itself is closed and that closedness is preserved under set union, that is:

$$closed{\cdot}V \wedge closed{\cdot}W \quad \Rightarrow \quad closed{\cdot}(V{\cup}W) \quad .$$

For example, if the universe is the natural numbers, and $<$ their usual ordering, then the closed subsets are, besides the universe itself, the finite intervals $[0,n)$, for all natural n. In this case, a finite closed set is represented –fully determined– by its *size*, and, we have:

$$\begin{array}{rcl} [0,m) \cup [0,n) & = & [0, m \,\mathsf{max}\, n) \\ [0,m) \subseteq [0,n) & \equiv & m \le n \end{array}$$

Closedness of sets is connected to transitivity of the relation $<$, in the following way. For every y we can define a set V_y by:

$$x \in V_y \quad \equiv \quad x < y \qquad .$$

Then we have that transitivity of $<$ is equivalent to $(\forall y :: \mathit{closed} \cdot V_y)$.

1.1 Projections

For set V and a predicate P defined on V, the subset of all elements of V that satisfy P is denoted by $V{\uparrow}P$ ("V projected onto P"):

$$x \in (V{\uparrow}P) \quad \equiv \quad x \in V \wedge P \cdot x \qquad .$$

Notice that, actually, projection is a non-concept: if we interpret sets as predicates, as in the above definition, projection is just conjunction of predicates. Obvious properties of projection are:

$$\begin{array}{rcl} \{x\}{\uparrow}P & = & \mathsf{if}\ P\cdot x \rightarrow \{x\}\ \ []\ \ \neg(P\cdot x) \rightarrow \phi\ \ \mathsf{fi} \\ (V \cup W){\uparrow}P & = & V{\uparrow}P \cup W{\uparrow}P \end{array}$$

Finally, if, in a universe Ω, V is closed with respect to a relation then so is $V{\uparrow}P$ in the subuniverse $\Omega{\uparrow}P$.

2 Causal delivery

We consider a distributed system consisting of concurrently operating processes. In each process so-called *events* take place; these events are strictly linearly ordered, that is, each process in itself is sequential. We use $\triangleright$ for sequential precedence: for events e and f in the *same* process $e \triangleright f$ means that e takes place before f. Sequential precedence is a total ordering:

$$(\forall e, f :: \ e = f \ \vee \ e \triangleright f \ \vee \ f \triangleright e) \qquad .$$

An event in a process is either an *internal* event, which represents a local state change of the process, or a *send* event, which represents sending a message to one (explicitly identified) other process in the system, or a *receive* event, which represents receiving a message sent earlier by some other process in the system. Every message sent eventually *arrives*, that is, becomes eligible for reception, at its destination, but we impose no upper bound on the time this takes, nor do we require messages to arrive in the same order in which they were sent. (What I call "reception" here, is called "delivery" in [1].)

Causal delivery is a generalization of *first-in first-out delivery*; the latter is the property that any two messages sent by one (same) process to another (same) process are received in the same order in which they were sent. The generalization is that we also consider messages sent by different processes: any two messages, whether or not sent by the same process, may or may not be *causally related*; a process has the property of causal delivery if it receives its messages in the same order in which they were sent, but only so for messages that are causally related. As we will see, for messages from the same sender, causal precedence coincides with ordinary sequential precedence; hence, causal delivery implies first-in first-out delivery.

Internal events are of no relevance to our discussion, which is why we ignore them in what follows. The contents of messages are irrelevant too, but their identities are important. Every send or receive event involves a message and, conversely, every message *uniquely* identifies one send event (including the process in which it takes place) and one receive event (including the process in which it takes place). Therefore, we can identify all events in the system by means of the messages. Based upon this observation we introduce the following nomenclature.

Throughout this note, we use variables i, j, and h to denote messages, and variables p and q to denote processes. Now we use:

$send_i$	for the event by which message i is sent
$sndr_i$	for the process in which $send_i$ takes place
$receive_j$	for the event by which message j is received
$rcvr_j$	for the process in which $receive_j$ takes place

causal precedence is a binary relation on the set of events; we denote it by $\rightarrow$, with the interpretation that $e \rightarrow f$ means that event e "causally precedes" event f. Postponing the definition of $\rightarrow$ for a while, we first use it to define *causal delivery*. This notion restricts the order in which messages can be received by a process; causal delivery is a property of individual processes: some processes in the system may have it whereas others may not.

definition: Process q has the property of causal delivery iff:

$$(\forall i,j : rcvr_i = q \wedge rcvr_j = q : \; send_i \rightarrow send_j \;\Rightarrow\; receive_i \triangleright receive_j\,) \quad .$$

□

remark: This definition slightly differs from the one given in [1], where $receive_i \rightarrow receive_j$ is used instead of $receive_i \triangleright receive_j$. Because, for events within the *same* process, causal precedence happens to coincide with sequential ordering, my replacing $\rightarrow$ by $\triangleright$ is both harmless and advantageous: now $\rightarrow$ becomes a relation on send events only, which allows for a simpler definition.

□

3 Causal precedence

In the above definition of causal delivery $\rightarrow$ only occurs in the formula $send_i \rightarrow send_j$; therefore, we only need define $\rightarrow$ as a relation on send events. Because of the one-to-one correspondence between send events and messages, we can now also define $\rightarrow$ as a relation on messages, instead of send events. That is, by way of abbreviation we write:

$$(\forall i,j :: \; i \rightarrow j \;\equiv\; send_i \rightarrow send_j) \quad .$$

Relation $\rightarrow$ is the transitive closure of the –my jargon– *basic precedence* relation, denoted by $\cdot\!>$. Informally, $send_i$ basically precedes $send_j$, notation: $i \cdot\!> j$, if either $send_i$ or $receive_i$ takes place (in the same process) before $send_j$:

definition: Relation $\cdot\!>$, on messages, is defined as follows, for all i,j :

$$\begin{aligned} i \cdot\!> j \;\equiv\;& (sndr_i = sndr_j \wedge send_i \triangleright send_j) \;\vee \\ & (rcvr_i = sndr_j \wedge receive_i \triangleright send_j) \end{aligned}$$

□

definition: As the transitive closure of $\cdot\!>$ relation $\rightarrow$ is most easily defined recursively, that is, for all i,j :

$$i \rightarrow j \;\equiv\; i \cdot\!> j \;\vee\; (\exists h :: \; i \rightarrow h \wedge h \cdot\!> j\,)$$

□

lemma 0 : Within the same sender causal precedence is the same as sequential precedence:

$$(\forall i,j : sndr_i = sndr_j : \; i \rightarrow j \;\equiv\; send_i \triangleright send_j\,)$$

proof: omitted (for the time being).
□

For our purposes, it is convenient to observe that the relations $\cdot\!>$ and $\rightarrow$ can also be represented by sets B_j and C_j, containing the basic and causal predecessors of j, thus:

$$\begin{array}{lcl} i \in B_j & \equiv & i \cdot\!> j \\ i \in C_j & \equiv & i \rightarrow j \end{array}$$

Reformulated in terms of C the above definition of $\rightarrow$ then assumes the following shape, for all j:

$$C_j \;=\; B_j \cup (\cup h : h{\in}B_j : C_h) \quad .$$

This definition makes explicit that the set of causal predecessors of a message j is fully determined by the set B_j of its basic predecessors.

In the same way the relation $\triangleright$ on receive events within a single process can also be represented by sets D_j, such that D_j is the set of messages received earlier than j by the same process; that is, for all i, j:

$$i \in D_j \quad \equiv \quad rcvr_i = rcvr_j \;\wedge\; receive_i \triangleright receive_j \quad .$$

In terms of the sets thus introduced, the definition of causal delivery can now be reformulated as:

definition: Process q has the property of causal delivery iff:

$$(\forall j : rcvr_j{=}q : \; (\forall i : i{\in}C_j \wedge rcvr_i{=}q : i{\in}D_j)\,) \quad .$$

□

4 Specification of the problem

The problem to be solved now is the implementation of causal delivery for a single, dedicated process in the system. We call this dedicated process the *monitor* and we identify it by the name μ. The monitor distinguishes itself from the other processes only by the requirement that it should have the property of causal delivery:

$$(\forall j : rcvr_j{=}\mu : \; (\forall i : i{\in}C_j \wedge rcvr_i{=}\mu : i{\in}D_j)\,) \quad .$$

To make this formula more manageable we introduce predicate R_μ, defined by:

$$(\forall j :: \; R_\mu{\cdot}j \;\; \equiv \;\; rcvr_j = \mu \,) \quad ,$$

such that we can reformulate the causal delivery property for μ by means of projection, as defined in Section 1:

$$(\forall j : R_\mu{\cdot}j : \; C_j \!\uparrow\! R_\mu \subseteq D_j \,) \quad .$$

5 An abstract solution

Because B_j is the set of messages sent or received by the same process from which j is sent, this set can be computed locally by that process. For this purpose we introduce, for every process p, a variable H_p –the *history* of process p–. Similarly, we introduce a single variable K_μ in the monitor –we are only interested in D_j for $rcvr_j = \mu$–. These variables are manipulated and used as follows:

$$\begin{array}{rl}
\textsf{initially}: & (\forall p :: H_p = \phi) \;\wedge\; K_\mu \,=\, \phi \\
\textsf{send } j \textsf{ from } p: & B_j := H_p \;\; ; \;\; H_p := H_p \cup \{j\} \\
\textsf{receive } j \textsf{ in } p \, (p \neq \mu): & H_p := H_p \cup \{j\} \\
\textsf{receive } j \textsf{ in } \mu: & \{ \; D_j = K_\mu \; \} \; H_\mu := H_\mu \cup \{j\} \;\; ; \;\; K_\mu := K_\mu \cup \{j\}
\end{array}$$

According to the specification the monitor has the property of causal delivery if every message j received by it satisfies $C_j \!\uparrow\! R_\mu \subseteq D_j$. Because the operation receive j in μ has preassertion $D_j = K_\mu$ we may use K_μ for D_j, so the requirement means that the operation must have $C_j \!\uparrow\! R_\mu \subseteq K_\mu$ as guard:

```
receive $j$ in $\mu$:   if  $C_j \uparrow R_\mu \subseteq K_\mu$  $\rightarrow$  $H_\mu := H_\mu \cup \{j\}$
                                          ; $K_\mu := K_\mu \cup \{j\}$
                      fi
```

Because of the guard, the monitor maintains as invariant:

P_0: $\quad (\forall j : j \in K_\mu : \; C_j \!\uparrow\! R_\mu \subseteq K_\mu \,) \quad .$

This solution is mathematically correct, but not very efficient: the problem is to construct an efficient implementation for the expression $C_j \!\uparrow\! R_\mu \subseteq K_\mu$. Here we have to take into account that the system is distributed, which means that the system has no globally accessible storage, where, for example, the sets C_j could be kept. Instead, set C_j must be sent together with message j to its destination: the process sending j has the information needed to compute C_j whereas the process receiving j uses C_j. Because C_j was defined by:

$$C_j \;=\; B_j \cup (\cup h : h{\in}B_j : C_h) \quad ,$$

and because B_j, when p is the sender, obtains its value from H_p, we equip process p with an additional variable G_p, coupled to H_p by the invariant:

P_1: $\quad (\forall p :: G_p \;=\; H_p \cup (\cup h : h{\in}H_p : C_h)\,)$

For the invariance of P_1 the operations must be extended accordingly:

```
             initially :   $(\forall p :: H_p = \phi \wedge G_p = \phi) \;\wedge\; K_\mu = \phi$
          send $j$ from $p$ :   $C_j := G_p \;;\; H_p := H_p \cup \{j\} \;;\; G_p := G_p \cup \{j\}$
receive $j$ in $p\,(p{\neq}\mu)$ :   $H_p := H_p \cup \{j\} \;;\; G_p := G_p \cup \{j\} \cup C_j$
        receive $j$ in $\mu$ :   if  $C_j{\uparrow}R_\mu \subseteq K_\mu \;\rightarrow\; H_\mu := H_\mu \cup \{j\}$
                                        ; $G_\mu := G_\mu \cup \{j\} \cup C_j$
                                        ; $K_\mu := K_\mu \cup \{j\}$
                     fi
```

Having played their role in the formulation of invariant P_1 variables H_p have now become superfluous, so they can be eliminated:

```
             initially :   $(\forall p :: G_p = \phi) \;\wedge\; K_\mu = \phi$
          send $j$ from $p$ :   $C_j := G_p \;;\; G_p := G_p \cup \{j\}$
receive $j$ in $p\,(p{\neq}\mu)$ :   $G_p := G_p \cup \{j\} \cup C_j$
        receive $j$ in $\mu$ :   if  $C_j{\uparrow}R_\mu \subseteq K_\mu \;\rightarrow\; G_\mu := G_\mu \cup \{j\} \cup C_j$
                                        ; $K_\mu := K_\mu \cup \{j\}$
                     fi
```

*   *   *

The only place where set C_j is used is in the expression $C_j{\uparrow}R_\mu$; therefore, we can replace all variables that play a role in the computation of C_j by their projections onto R_μ. This is a standard transformation: we introduce new variables E_j and F_p that take over the roles of C_j and G_p, as prescribed by the invariants:

P_2: $\quad (\forall j :: E_j \;=\; C_j{\uparrow}R_\mu\,)$

P_3: $\quad (\forall p :: F_p \;=\; G_p{\uparrow}R_\mu\,)$

Using the projection properties from Section 1 the above program can now be reformulated in terms of the new variables:

```
                  initially :   $(\forall p :: F_p = \phi) \;\wedge\; K_\mu = \phi$
send $j$ from $p$ to $q$ $(q \neq \mu)$ :   $E_j := F_p$
         send $j$ from $p$ to $\mu$ :   $E_j := F_p$ ; $F_p := F_p \cup \{j\}$
      receive $j$ in $p$ $(p \neq \mu)$ :   $F_p := F_p \cup E_j$
            receive $j$ in $\mu$ :   if  $E_j \subseteq K_\mu \;\rightarrow\; F_\mu := F_\mu \cup \{j\} \cup E_j$
                                                ; $K_\mu := K_\mu \cup \{j\}$
                               fi
```

6 Implementation

According to lemma 0 –Section 3– causal precedence amounts to sequential precedence for messages sent by the same process. Because causal precedence is transitive, all sets E_j in our solution are closed –Section 1– with respect to causal precedence, and so are their projections onto the predicates that distinguish messages by their senders. More formally, we define predicates S_p by:

$$(\forall j ::\; S_p{\cdot}j \;\equiv\; sndr_j = p\,) \quad ,$$

and we partition E_j into sets $E_j{\restriction}S_p$, for all p. Notice that $E_j{\restriction}S_p$ is the set of messages, sent by process p to process μ, that causally precede j.

Similarly, $K_\mu{\restriction}S_p$ is the set of messages received by process μ from process p. Because of invariant P_0, set K_μ is closed with respect to $\rightarrow$, and so are the sets $K_\mu{\restriction}S_p$. Obviously, we now have:

$$E_j \subseteq K_\mu \;\equiv\; (\forall p ::\; E_j{\restriction}S_p \subseteq K_\mu{\restriction}S_p\,) \quad .$$

Because they contain messages sent by a single process only, because hence these messages are linearly ordered, and because they are closed, we can represent these sets by their *sizes*. Therefore, we introduce variables e_j and k_μ with, for all j, p:

$$e_j{\cdot}p \;=\; (\#i ::\; i \in E_j{\restriction}S_p\,)$$
$$k_\mu{\cdot}p \;=\; (\#i ::\; i \in K_\mu{\restriction}S_p\,)$$

In more direct terms, these relations can be rewritten, of course, as:

$$e_j{\cdot}p \;=\; (\#i ::\; i \rightarrow j \wedge rcvr_i = \mu \wedge sndr_i = p\,)$$
$$k_\mu{\cdot}p \;=\; (\#i ::\; i \in K_\mu \wedge sndr_i = p\,)$$

The condition $E_j \subseteq K_\mu$ can now be encoded as:

$$(\forall p :: \, e_j{\cdot}p \le k_\mu{\cdot}p) \quad \text{, or, for short: } e_j \le k_\mu \quad .$$

To implement this we must also represent sets F_p by their sizes, so we also introduce variables f_p , with, for all p, q :

$$f_p{\cdot}q \; = \; (\#i :: \, i{\in}F_p \wedge rcvr_i = \mu \wedge sndr_i = q) \quad .$$

In terms of the new variables, our solution becomes, where max is used for element-wise maximum:

```
                 initially :   $(\forall p :: f_p = 0) \;\wedge\; k_\mu = 0$
send $j$ from $p$ to $q$ $(q{\neq}\mu)$ :   $e_j := f_p$
          send $j$ from $p$ to $\mu$ :   $e_j := f_p$ ; $f_p{\cdot}p := f_p{\cdot}p + 1$
     receive $j$ in $p$ $(p{\neq}\mu)$ :   $f_p := f_p$ max $e_j$
     receive $j$ from $p$ in $\mu$ :   if $e_j \le k_\mu$ → $f_\mu := f_\mu$ max $e_j$
                                          ; $f_\mu{\cdot}p := f_\mu{\cdot}p + 1$
                                          ; $k_\mu{\cdot}p := k_\mu{\cdot}p + 1$
                                 fi
```

7 Epilogue

This is my first attempt to present the algorithm for causal delivery in a systematic way. With the degree of formality employed here I am much more confident in the correctness of the algorithm than I was after reading [1]. (To be honest, the presentation in [1] also suffers from lack of precision.) To smoothen the presentation it probably will pay to develop the little theory of closed sets and their projections a little further.

That the authors of [1] have felt the need to introduce a so-called "Gap-Detection" property might be caused by their failing to observe that, as a result of causal delivery, the set of messages delivered to the monitor is closed with respect to causal precedence –cf. invariant P_0– . This observation is crucial to the design, because it is this property that allows this set to be represented by a vector –variable k_μ– as well.

References

[1] Ö. Babaoğlu, K. Marzullo, *Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms*, in: S. Mullender (ed.), *Distributed Systems*, Addison-Wesley (1993, 2nd ed.).

Eindhoven, 20 october 1995

Rob R. Hoogerwoord
department of mathematics and computing science
Eindhoven University of Technology
postbus 513
5600 MB Eindhoven