HAL
archives-ouvertes.fr

The causal ordering abstraction and a simple way to implement it

Michel Raynal, André Schiper

▶ To cite this version:

Michel Raynal, André Schiper. The causal ordering abstraction and a simple way to implement it. [Research Report] RR-1132, INRIA. 1989. <inria-00075427>

HAL Id: inria-00075427
https://hal.inria.fr/inria-00075427

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1132

Programme 3
Réseaux et Systèmes Répartis

THE CAUSAL ORDERING ABSTRACTION AND A SIMPLE WAY TO IMPLEMENT IT

Michel RAYNAL
André SCHIPER

Décembre 1989

RR-1132

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

THE CAUSAL ORDERING ABSTRACTION AND A SIMPLE WAY TO IMPLEMENT IT

Publication Interne n° 501 - Novembre 1989 - 12 Pages

Michel RAYNAL
IRISA
Campus de Beaulieu
F - 35042 Rennes Cedex
raynal@irisa.fr

André SCHIPER
Ecole Polytechnique Fédérale de Lausanne
Département d'Informatique
CH - 1015 Lausanne
schiper@elma.epfl.ch.bitnet

ABSTRACT

Control in distributed systems is mainly introduced to reduce non-determinism. This non-determinism is due on one hand to the asynchronous execution of the processes located on the various sites of the system, and on the other hand to the asynchronous nature of the communication channels. In order to get rid of part of the asynchronism due to the communication channels, a new ordering relation, known as causal ordering, has been introduced by Birman. After having shown the usefulness of causal ordering, we propose a natural implementation, based simply on counting the emitted messages. A proof of the correctness of the algorithm is also given.

LE CONCEPT D'ORDONNANCEMENT CAUSAL ET UNE MISE EN OEUVRE SIMPLE

RESUME

Une grande partie du contrôle réalisé dans les systèmes répartis a pour but de réduire le non-déterminisme de leurs comportements. Une cause de celui-ci est l'asynchronisme des processus placés sur les divers sites du système et l'asynchronisme des canaux de communication. Afin de s'affranchir en partie de ce dernier asynchronisme un nouvel ordre sur les réceptions de messages a été introduit par Birman : l'ordonnancement causal. Après avoir rappelé l'intérêt de cet ordre, une mise en oeuvre particulièrement simple en est proposée ; sa correction est également prouvée.

THE CAUSAL ORDERING ABSTRACTION AND A SIMPLE WAY TO IMPLEMENT IT

Michel RAYNAL
IRISA
Campus de Beaulieu
F - 35042 Rennes Cedex
raynal@irisa.fr

André SCHIPER
Ecole Polytechnique Fédérale de Lausanne
Département d'Informatique
CH - 1015 Lausanne
schiper@elma.epfl.ch.bitnet

ABSTRACT

Control in distributed systems is mainly introduced to reduce non-determinism. This non-determinism is due on one hand to the asynchronous execution of the processes located on the various sites of the system, and on the other hand to the asynchronous nature of the communication channels. In order to get rid of part of the asynchronism due to the communication channels, a new ordering relation, known as causal ordering, has been introduced by Birman. After having shown the usefulness of causal ordering, we propose a natural implementation, based simply on counting the emitted messages. A proof of the correctness of the algorithm is also given.

1. CONTROLLING NON-DETERMINISM

Controlling non-determinism is one of the essential tasks in computer systems. In centralized systems the non-determinism is caused by interrupts. In distributed systems the non-determinism is due to the asynchronous execution of the parallel processes together with the asynchronous nature of the communication channels linking them together; it is also due to the failures that can affect the different entities of the system.

In order to master the asynchronism of the processes and so solve the related problems (take for example the resource allocation problem) it is usual to introduce synchronization mechanisms [Raynal 88]. In ordrer to master the asynchronism of the communication channels it is possible to build network synchronizers which force processes and channels to progress in synchronized steps [Awerbuch 85, Helary 90]. This corresponds to defining a virtual distributed machine in which the undesirable behaviour due to non-determinism has been suppressed. The same is true for communication protocols, whose objective is to get rid of the non-reliable channels: communication protocols define a "reliable channel" abstraction which is implemented on top of a non-reliable channel. The alternate bit protocol [Barlett 69] or Stenning's protocol [Stenning 76] are such examples: both build a channel without message loss, duplification or desequencing on top of a channel that can lose or duplicate messages, or even, in the case of Stenning's protocol, that can desequence them. The reliable channel abstraction, like the synchronization mechanisms, make it possible to get rid of the undesirable non-deterministic behaviours.

In this paper we consider a reliable distributed system, and are concerned by the realization of a communication scheme that make it possible to get rid of a particular non-deterministic behaviour. More precisely, we are concerned with the abstraction called "causal ordering", as proposed by the Isis system [Birman 87]. The paper is organized in the following way. In section 2 we define causal ordering and show its usefulness. In section 3 we show an easy and

PAPIER RECUPERE ET RECYCLE

natural way to implement causal ordering (which differs from the Isis implementation), and give in section 4 a proof of its correctness.

2. CAUSAL ORDERING

2.1 Definition

Causal ordering of events in a distributed system (an event corresponding to the emission of a message, the reception of a message, or an internal action) is based on the well known "happened before" relation, noted -> [Lamport 78]. If E1 and E2 are two events, then A -> B iff one of the following conditions is true:

i) A and B are two events occurring on the same site, A before B;

ii) A is the emission of a message, and B corresponds to the reception of the same message;

iii) there exists an event C such that A -> C and C -> B.

The delivery of a message depends in principle only on the fact that it has been sent; in other words, the delivery does not depend on the state of the system at reception time. In the case of FIFO channels, the reception of a message M partly depends on the state of the system: it depends on the state at emission time of the channel through which the message is received. So the FIFO channel abstraction reduces the non-determinism of a channel. The causal ordering corresponds to reducing the non-determinism globally on all of the channels: the delivery of a message is dependent on the state of the system as known by the sender at emission time. Consider two events E1 = SEND(M1) and E2 = SEND(M2). Causal ordering is respected if:

if (E1 -> E2) and (M1, M2 have same destination)
then RECEIVE(M1) -> RECEIVE(M2)

In other words, in case of a "happened before" relation between two emissions to a same destination, there exists a "happened before" relation between the delivery of the messages. For example, on figure 1, message M1 must be delivered on site S3 before message M3.

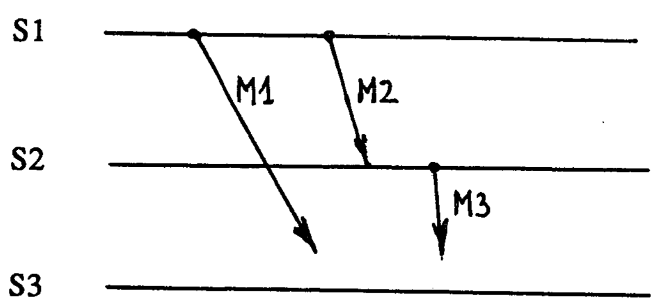

Fig. 1 Causal ordering ensures that M1 is received before M3.

2.2 Examples of the Usefulness of Causal Ordering

Management of a replicated data

Consider a data X replicated on various sites. In order to ensure mutual consistency of the various copies x1, x2, ..., the updates to these copies must take place in the same order. In particular this introduces the need for mutual exclusion on the updates done by two different sites. This can be solved using a token; when in possession of the token, a site can update his copy of X and broadcasts the update to all other sites having a copy of X (fig. 2). Causal ordering ensures that all sites will receive all the updates in the same order (the update Wi will not be delivered before the update Wj, for i < j), which ensures mutual consistency of the copies [Birman 88, Joseph 86]. The token ensures total ordering of the updates on X and causal ordering ensures that all the copies xi are updated in this same order.

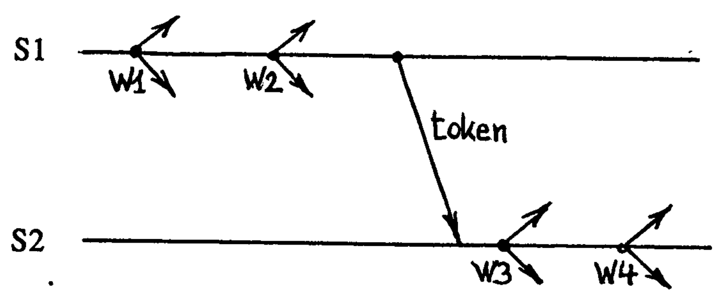

Fig. 2 Management of a replicated data

Consider now a second data Y, controlled by a second token, and two sites S1, S2 (with copies x1, y1, resp. x2, y2). Both sites see the same sequence of updates on x1, x2 and on y1, y2. The interleaving of these sequences can nevertheless be different on S1 and S2. In other words, causal ordering does not define a total order.

Monitoring a distributed system

Consistent observation of a distributed system is another example of the usefulness of causal ordering. Consider a number of sites monitoring events occurring locally, and suppose that all this information is of interest to an observer site OS. If the monitoring information collected by the sites is sent to OS respecting causal ordering, the information will be received in a coherent order (i.e. in an order not violating the events precedence order).

Resource allocation

Consider the mutual exclusion problem, or more generally a resource allocation problem. This can be solved by a resource allocator residing on a site, which receives allocation requests in a FIFO queue. With this scheme, the requests are honoured in the reception order. This might however be considered unsatisfactory in some cases: it might be desirable to honour requests not in their reception order, but in their emission order, i.e. if two requests R1 and R2 are such that R1 -> R2, then R1 should be honoured before R2. This is again a causal ordering problem. Note that this example, in the particular case of mutual exclusion, was used by Lamport in his paper introducing logical clocks [Lamport 78]. Lamport's solution was to duplicate the resource allocator on each site (each site being concerned with all the requests and having to acknowledge them). This can be avoided using causal ordering!

3. IMPLEMENTATION OF CAUSAL ORDERING

3.1 Related Work

The Isis implementation of causal ordering [Birman 87] is straightforward: a message carries with it all the history of the communications that have preceded. Consider again figure 1. Message M3, sent by S2 to S3, carries the following information about the past: message M1, with the information that it was sent to S3, and message M2 with the information that it was sent to S3. When receiving M3, site S3 thus also receives a copy of M1: if not already delivered, M1 will be delivered at that time, i.e. before M3. To prevent unbounded growth of the information added to a message, a mechanism must be added to inform of messages that have been received and delivered: these messages need not be sent any more. This mechanism is however rather complex. Anyhow, the size of the information added to messages is unbounded.

Another implementation of causal ordering is given in [Schiper 89]. In this implementation, control information (and not messages as in Isis) is added to the messages. This control information allows the destination site of a message M to know if there are messages that have to be delivered before M, in order to respect causal ordering: if this is the case, M is not immediately delivered. The control information is composed of a bounded number of pairs (*destination site*, *vector time*), where *"vector time"* is a time defining a partial ordering of the events of a distributed system [Fidge 88, Mattern 88]. The implementation proposed in this paper has some similarities with this implementation. It does not however use vector times, and is thus much more easy to understand.

3.2 A Simple Implementation

To implement causal ordering we give every site control information representing the site's perception of the system state (more precisely, the site's perception of the communications). This information will allow each site to decide when a received message can be delivered. Moreover, information added to every message will allow the receiving site, when a message is delivered, to update its perception of the system state.

Local information of a site

Every site manages the following two variables (where n is equal to the number of sites in the system):

REC: array [1..n] of integer; (* initially set to 0 *)

SENT: array [1..n,1..n] of integer; (* initially set to 0 *)

We will use the notation RECi and SENTi to refer to the variables managed by site Si. So on site Si, the variable RECi[j] represents the number of messages sent by Sj (and delivered); the variable SENTi[k,l] represents the knowledge of Si of the number of messages sent (but not necessarily delivered) from Sk to Sl.

Behaviour of a site

The behaviour of a site Si is expressed by the following two rules, governing the emission and the reception of a message.

i) *Emission of a message M from Si to Sj:*

SENTi[i,j] := SENTi[i,j] + 1;
send (M, SENTi) to Sj;

Thus M is sent together with a copy of SENTi, i.e. with site Si's view of the system state.

ii) *Reception of (M,ST_M) sent from Sj to Si (where ST_M represents the control information "SENT" carried by the message M):*

wait (RECi[j]+1 = ST_M[j,i]) and (for all k≠j: RECi[k] ≥ ST_M[k,i]);
delivery of M;
RECi[j] := RECi[j] + 1;
for all k,l: SENTi[k,l] := max (SENTi[k,l], ST_M[k,l]);

Thus a message M sent by Sj can be delivered only if:

- all the messages sent by Sj before M have been delivered. This is expressed by the first part of the wait condition: RECi[j]+1 = ST_M[j,i]. This condition ensures that the communication channels are perceived as FIFO;

- all other causally preceding messages, whose existence is revealed by ST_M, have been delivered. This is expressed by RECi[k] ≥ ST_M[k,i] for all k≠j.

Once a message has been delivered, the site's perception of the system state is updated in an obvious way.

Before going further to prove the algorithm, let's notice that the algorithm is easy to adapt to the multicast case. The emission rule i) has simply to be replaced by the following rule i'):

i') *Emission of a message M from Si to the set of sites S=[Sj1,...,Sjm]:*

for all k in [j1,...,jm]: SENTi[i,k] := SENTi[i,k] + 1;
send (M, SENTi) to every member of the set S;

4. PROOF OF CORRECTNESS

The correctness of the above algorithm will be proved in the usual two steps. We will first prove in 4.1 that causal ordering is not violated: this corresponds to the safety condition. In 4.2 we will then prove the liveness of the algorithm.

4.1 Safety

To prove the safety of the algorithm we need the following intermediate result:

Proposition 1. If neither a message M1, nor a message M such that SEND(M1) -> SEND(M), has been delivered to a site Si, then RECi[j] < ST_M1[j,i] (where j represents the sending site Sj, and ST_M1 the control information carried with message M1).

Proof. First notice that only the messages sent by Sj are to be considered, because only these messages can increment RECi[j]. So consider that M was sent by Sj. All the messages sent by Sj after message M1 are such that SEND(M1) -> SEND(M). Thus if none of these messages has been delivered to Si, we have: the number of messages from Sj delivered to Si is less than the number of messages sent to Si by Sj up to the sending of M1, in other words RECi[j] < ST_M1[j,i].

The safety condition is equivalent to the following assertion: as long as a message M1 has not been delivered to Si, no message M2 such that SEND(M1) -> SEND(M2) can be delivered. The proof will be made by induction on the number of messages delivered on the destination site Si.

i) *Base step.*

The first message received by Si cannot be M2. To prove this there are two cases to consider:

1) M2 is sent from Sj to Si (as message M1). In this case ST_M2[j,i] ≥ 2. Thus RECi[j]+1 ≠ ST_M2[j,i] (because RECi[j] is initially 0), and M2 cannot be delivered (first part of the wait condition not satisfied).

2) M2 is sent from Sk (k≠j). Here ST_M2[j,i] > 0, and thus on the destination site Si, RECi[j] < ST_M2[j,i]. So M2 cannot be delivered (second part of the wait condition not satisfied).

ii) *Induction step.*

We suppose that M1 is not delivered on Si, and that the nth message delivered is not a message M2 such that SEND(M1) -> SEND(M2). We will prove that the same is true for the n+1st message. Again there are two cases to consider:

1) M2 is sent from Sj to Si (as message M1). From the induction hypothesis and proposition 1, we have RECi[j] < ST_M1[j,i]. Further, we have ST_M2[j,i] > ST_M1[j,i], as M2 is also sent from Sj. So, after delivery of the nth message on Si we have:

RECi[j] < ST_M1[j,i] < ST_M2[j,i]

This prevents M2 from being the next message delivered on Si (first part of the wait condition not satisfied).

2) M2 is sent from Sk (k≠j). Here we have the same inequalities as above, except that ST_M1[j,i] ≤ ST_M2[j,i] (because M2 is not sent by Sj). Thus after delivery of the nth message on Si we have:

RECi[j] < ST_M1[j,i] ≤ ST_M2[j,i]

i.e. RECi[j] < ST_M2[j,i], which again prevents M2 from being the next message delivered on Si (second part of the wait condition not satisfied).

4.2 Liveness

To prove liveness we will suppose that there exists a message M sent from Sj to Si that is never delivered, and show that we end up with a contradiction. Message M comes with the control information ST_M[x,y]. For x≠j, ST_M[x,i] counts the number of messages sent by Sx to Si that causally precede M, and ST_M[j,i]-1 counts the number of messages sent by Sj to Si that causally precede M. The number of messages that causally precede M is thus bounded. In absence of failure, and after a finite time, all these messages will have arrived, and message M will be deliverable (see the wait conditions).

So, if M cannot be delivered, there exists a message M', such that SEND(M') -> SEND(M), which is never delivered. The same reasoning applied to M can again be applied to M', and so on. As the number of messages causally preceding M is finite, we end up with a message M(n) which has no message causally preceding it. So M(n) will be delivered, which shows the contradiction.

5. CONCLUSION

In this paper, causal ordering has been justified as a means for reducing the non-determinism due to the asynchronism of communication channels. Its usefulness has been shown by various examples, and a simple implementation has been given. With this implementation, the destination site of a message can know, when receiving a message, if it can be immediately delivered, or if causally preceding messages are still underway. In contrast to the Isis implementation, the information added to messages to ensure causal ordering is here bounded, more precisely bounded by n^2, where n is the number of sites. The same bound is obtained in [Schiper 89], but the given implementation is less easy to understand. In contrast, the proposed implementation is simply based on counting the messages emitted. In a certain sense, this implementation extends the concept of sequence numbers used in networks [Stenning 76]. More generally, the counting techniques are usual in many solution to synchronization problems, starting from semaphores in centralized systems (to solve competition or synchronization problems), up to counters in distributed systems used to obtain consistent global information (for example distributed termination [Helary 87, Mattern 87]) or to ensure coherent global decisions (e.g. distributed mutual exclusion [Raynal 86]).

REFERENCES

[Awerbuch 85] B.Awerbuch, "Complexity of Networks Synchronization", Journal of the ACM, Vol 32, No 4 (Oct. 85), pp 804-823.

[Barlett 69] K.A.Barlett, R.A.Scantlebury, P.T.Wilkinson, "A note on reliable full duplex transmission over half duplex links, Comm. ACM, Vol 12, No 5 (May 69), pp 260-261.

[Birman 87] K.Birman, T.Joseph, "Reliable Communications in Presence of Failures", ACM TOCS, Vol 4, No 1 (Feb 86), pp 54-70.

[Birman 88] K.Birman, "Exploiting Replication", in Lectures Notes Arctic'88, Tromso, July 1988.

[Fidge 88] C.Fidge, "Timestamps in Message-Passing Systems That Preserve the Partial Ordering", Proc. of the 11th Australian Computer Science Conference, Univ of Queensland, Feb 1988.

[Helary 87] J.M.Helary, C.Jard, N.Plouzeau, M.Raynal, "Detection of Stable Properties in Distributed Applications", Proc. 6th annual ACM Symp. on PODC (Aug 87), pp 125-136.

[Helary 90] J.M.Helary, M.Raynal, "Synchronization and Control of Distributed Systems and Programs", J.Wiley, 1990, 200p.

[Joseph 86] T.Joseph, K.Birman, "Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems, ACM TOCS, Vol 4, No 1 (Feb 86), pp 54-70.

[Lamport 78] L.Lamport, "Time, Clocks and the Ordering of Events in a Distributed Sytem", Comm. ACM, Vol 21, No 7 (July 78), pp 558-565.

[Mattern 87] F.Mattern, "Algorithms for Distributed Termination Detection", Distributed Computing, Vol 2, No 3 (1987), pp 161-175.

[Mattern 88] F.Mattern, "Time and Global States of Distributed Systems", Proc. Int. Workshop on Parallel and Distributed Algorithms, Bonas, France (Oct 88), North-Holland, pp 215-226.

[Raynal 86] M.Raynal, "Algorithms for Mutual Exclusion", the MIT Press, 1986, 107p.

[Raynal 88] M.Raynal, "Distributed Computation and Networks: concepts, tools and algorithms", the MIT Press, 1988, 166p.

[Schiper 89] A.Schiper, J.Eggli, A.Sandoz, "A New Algorithm to Implement Causal Ordering", Proc. 3rd Int. Workshop on Distributed Algorithms, Nice (Sept 89), Springer Verlag, LNCS 392, pp 219-232.

[Stenning 76] W.Stenning, "A Data Transfer Protocol", Computer Network, Vol 1 (1976), pp 99-110.

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 492 SPARSE MATRIX MULTIPLICATION ON VECTOR COMPUTERS
Jocelyne ERHEL
20 Pages, Septembre 1989.

PI 493 THE SUPERIMPOSITION OF ESTELLE PROGRAMS : A TOOL FOR THE IMPLEMENTATION OF OBSERVATION AND CONTROL ALGORITHMS
Benoît CAILLAUD
30 Pages, Septembre 1989.

PI 494 EMPLOI DU TEMPS : PROBLEME MATHEMATIQUE OU PROBLEME POUR LA PROGRAMMATION EN LOGIQUE AVEC CONTRAINTES
Xavier COUSIN
64 Pages, Septembre 1989.

PI 495 NUMERICAL METHODS IN MARKOV CHAIN MODELING
Bernard PHILIPPE, Youcef SAAD, William J. STEWART
46 Pages, Septembre 1989.

PI 496 PLANIFICATION EN UNIVERS MONO ET MULTI-AGENTS (Définitions, concepts, objectifs, présentation d'un planificateur)
Philippe PORTEJOIE
92 Pages, Octobre 1989.

PI 497 LE TRAITEMENT D'EXCEPTIONS - ASPECTS THEORIQUES ET PRATIQUES
Valérie ISSARNY
80 Pages, Octobre 1989.

PI 498 IMPLEMENTATION D'UN LANGAGE DE PROGRAMMATION LOGIQUE D'ORDRE SUPERIEUR AVEC MALI
Pascal BRISSET
28 Pages, Octobre 1989.

PI 499 QUANTIFICATION IMAGE PAR LA METHODE DE LA VRAISEMBLANCE DU LIEN (A.V.L.) AVEC UN CODAGE PREORDONNANCE
Israël-César LERMAN, Nadia GHAZZALI
60 Pages, Octobre 1989.

PI 500 EFFICACITE DE LA METHODE DES PUISSANCES UNIFORMISEES POUR LES CHAINES DE MARKOV RAIDES
Haïscam ABDALLAH, Raymond MARIE
18 Pages, Octobre 1989.

PI 501 THE CAUSAL ORDERING ABSTRACTION AND A SIMPLE WAY TO IMPLEMENT IT
Michel RAYNAL, André SCHIPER
12 Pages, Novembre 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique