Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

A DOMINO-EFFECT FREE RECOVERY ALGORITHM: FORMAL SPECIFICATION

D. Briatico, A. Ciuffoletti, L. Simoncini

Nota interna B84-07

Giugno 1984

A DOMINO-EFFECT FREE RECOVERY ALGORITHM: FORMAL SPECIFICATION.

D. Briatico (*), A. Ciuffoletti (*), L. Simoncini (**)

* Dipartimento di Informatica, Universita` di Pisa
** Istituto di Elaborazione dell'Informazione, CNR, Pisa.

ABSTRACT

Recovery techniques may be distinguished on the basis of the time when the recovery lines are built:

at the time of recording the recovery point,

at the time of rollback;

Consequently we distinguish "planned" and "unplanned" policies for determining recovery lines.
With an unplanned policy a "domino effect" can occur, with increase of rollback activity and unknown amount of process elaboration to be undone.
The planned policy is usually intended as being static, in the sense that the recovery lines are a priori established at design time.

In this paper an algorithm for "dynamic" planning of recovery line is formally specified. We shall define a computational model for a distributed system of communicating processes using asynchronous message passing and shall describe the recovery algorithms by means of axioms.

A DOMINO-EFFECT FREE RECOVERY ALGORITHM: FORMAL SPECIFICATION.

D. Briatico (*), A. Ciuffoletti (*), L. Simoncini (**)

* Dipartimento di Informatica, Universita` di Pisa
** Istituto di Elaborazione dell'Informazione, CNR, Pisa.

SUBJECT INDEX: Dependable Software Development/ Recovery Techniques.

SUBAREA: Robust Programming.

MAILING ADDRESSES: Daniele Briatico,
c/o SELENIA s.p.a., Via S. Maria 83
56100 PISA, ITALY
- tel. +39 50 43023

Augusto Ciuffoletti,
c/o SELENIA s.p.a., Via S. Maria 83,
56100 PISA, ITALY
- tel. +39 50 43023

Luca Simoncini,
I.E.I. - C.N.R., Via S. Maria 46,
56100 PISA, ITALY
- tel. +39 50 43023 // +39 50 500159
- telex. 590305 IEICNR I
- telex. 500371 CNUCE I

THE PAPER HAS BEEN CLEARED THROUGH ALL AUTHORS' AFFILIATIONS

THE PAPER WILL BE PRESENTED BY LUCA SIMONCINI

MAIL ALL CORRESPONDENCE TO : LUCA SIMONCINI,
I.E.I. - C.N.R.,
VIA S.MARIA 46,
56100 PISA
ITALY

1. Introduction.

This paper is concerned with error recovery in distributed systems.

Such systems may be generally viewed as composed by a set of independent components, called processes, exchanging information through message passing.

With respect to the independence of the processes, we will suppose that the message passing mechanism is asynchronous, as contraposed to synchronous mechanisms defined as in ADA /ADA80a/ or CSP /HOA78a/ or DP /HAN78a/. Thus we allow that messages may be shuffled during the propagation between the sender and the receiver, and we consider that communications are made reliable by lower levels.

In a distributed system, the operation of error recovery is aimed to the restoration of a consistent state after the failure of a subset of components of the system.

Two methods are generally referenced for the recovery of a system: backward and forward; in this paper we are essentially concerned with the first one, for which we formally specify an algorithmic solution, applicable to distributed system.

We refer to backward methods when one of the process constituent the system or a substitute of it can resume a previously saved state. We call this action rollback and the saved state recovery point.

Thus we can identify two distinct phases in a backward recovery algorithm: the first which is devoted to the creation of the recovery points, and which takes place during the normal operation of the system; the second, which is aimed to the resumption of a correct state through the use of the information contained in the recovery points.

However, the application of backward error recovery algorithms in distributed systems is hindered by the occurrence of "domino effect". This consists in the impossibility of restoring an acceptable state starting from the recovery points available on the different components.

Two different attitudes can be identified about the time when the systems builds the recovery line, joining recovery points of the processes, which constitutes a consistent state of the system:

- at the time of the recording of the recovery point;

- at the time of the rollback.

The first attitude involves that recovery points are recorded according to certain rules that ensure that all of them belong to a recovery line. In this case we speak of a "planned" existence of recovery points /RAN75a/.

The other attitude involves that recovery points are recorded regardless to the generation of recovery lines: the generation of recovery lines is attempted at the time of the rollback. This attempt is not necessarily successful, as a domino effect can make useless part of the recorded recovery points /AND79a/. The probability and the amount of the domino effect is design dependent /RUS80a/ ,/MER78a/.

In this paper we shall consider the first attitude. This one requires a coordination among the recovery points of the processes. This is directed by the rule used to determine recovery lines; this rule can be deduced from the specific internal activity of the system /LAM81a/; in this case the policy for the generation of the recovery points is statically decided and included in the specification of the system. Yet, such a deduction may be difficult and extremely critical. Examples of such kind of static coordination may be considered the transactions.

In this paper we shall discuss a generalized decision rule, based on a decision dynamically adapted to the activity of the system.

As the recovery points are created through coordination among independent processes, we shall call them "planned recovery points" as suggested by Anderson in /AND79a/; but we stress that what we are going to talk about has nothing to do with conversations /RAN75a/ or transactions. At least, not directly.

We will formally specify a data structure (i.e. objects and rules of use) aimed to the storage of informations related to the recovery lines and their consistency. The axioms for this structure ensure that a consistent recovery line may be found for every recovery point.

As we want to preserve the asynchronism between the send and the receive of a message, compensation actions may be associated to a recovery line. This compensation consists in repeating the receive of a message sent before the recovery line and received after it; we will specify more formally these conditions in the following. For the present, it is enough to say that the amount of compensation required

is limited to a subset of the messages exchanged in the system.

The axioms we give are reasonably applicable to a distributed system, and a typical application is described in the companion paper /BRI83c/ and in /BRI83a/.

The next section is organized in 4 steps:

- A formal definition of what a parallel computing system is considered to be;
- a definition of recovery line in such a system
- The definition of a data structure associated to each process, as a set of composite objects and some invariant axioms;
- A theorem that claims that from a set of data structures can be deduced a recovery line .

2. The formal model.

In this section a formal model is defined aimed to the description of the system computation. Our model describes only the aspects of the computation which are related to the recovery line planning.

As we consider a system supporting a parallel computations, we will describe it with a set of independent processes Q. The interactions among processes are modeled by message passing.

More formally, we say that each process c in Q is composed by a sequence of events, each being identified by:

a) the name of the process which produces it, and

b) its position in the sequence of events produced by that process.

To this purpose we will define a function (E) from the pairs (Q,N) of process names and natural numbers to the domain of events (Ev):

$$E: Q \times N \rightarrow Ev$$

Thus an event will be described by E(c,i) with c in Q and i in N.

Further, as we must identify several kinds of event, we need a function that maps events into a set of "types":

type: Ev -> {send, receive, set-rp, don't care}

The inverse of the function is:

~type: {send, receive, set-rp, don't care} -> P(Ev)

and identifies a partition on the set of events.
In the following we will express with P(S) the set of the subsets of the set S (set of parts of S).

Two processes may interact, exchanging information; the interaction between two processes is modeled by a function (I) which associates, to a pair of related send and receive events, the information (message) exchanged from the first to the second.

The I function is partial and is defined from the pairs of events to the set of all possible messages M:

I: Ev x Ev -> M

To be an interaction function, the I must satisfy some axioms:

1st I axiom: Let snd = E(sndr,i)
rcv = E(rcvr,j)
I(snd,rcv) is defined => sndr is different from rcvr
type(snd) = send
type(rcv) = receive

2nd I axiom: Let snd, rcv, e3 in Ev
I(snd,rcv) is defined => I(rcv,e3), I(snd,e3) are undefined.

3rd I axiom: Let e1 in Ev
type(e1)=send => exists e2, I(e1,e2) is defined
and type(e1)=receive => exists e2, I(e2,e1) is defined.

4th I axiom: Let snd, rcv in Ev
let S = { e | e -> snd and
type(e) = set-rp }
let R = { e | e -> rcv and
type(e) = set-rp }
I(snd,rcv) is defined => #S =< #R

The first axiom states that a process cannot interact with itself, while the second states that a message is exchanged from a single send to a single receive, with neither duplication nor broadcast. The third

axiom requires that every interaction is complete, in the sense that its unique send and its unique receive are contained in the description of the system. In practice this requirement can be made far less stringent, as explained in the companion paper /BRI83c/.

The fourth axiom has a specific relevance for what is concerning the avoidance of domino effect; it requires that for each send-receive pair, the number of set-rp preceding the receive is greater or at least equal to the number of set-rp preceding the send. This requirement must be enforced by a appropriate implementation of the receive. A pragmatic solution to this problem could be the dynamic execution of an appropriate number of set-rp events, when the process "perceives" that the following event should be a receive of a message from a process with a greater number of set-rp events. We note that a set-rp event has no direct side effects on other components.

We have thus introduced an "a posteriori" formal description of a system on the basis of the set of components of the system (Q), a set of events (Ev) which describes the behaviour, associated time by time to each component by E; a specific behaviour can involve the exchange of information from a component to another described by a message (I). Thus a system is described by the quintuple (Q,Ev,M,E,I).

Now we define two derived total functions, that will be useful for the formal description of the problem of recovery line planning.

The first is defined on the set of all send events, and returns the receiver of the message:

receiver: ~type(send) -> Q

receiver function: the function receiver associates to an event snd in ~type(send) the process rcvr such that:
exists i, I(snd,E(rcvr,i)) is defined.

The second is defined on the set of all receive events, and gives the sender of the message and the message itself:

sender: ~type(receive) -> Q x M

sender function: the function sender associates to an event rcv in ~type(receive)

a) the process sndr such that:
exists i, I(E(sndr,i),rcv) is defined.

b) the message m such that,
m = I(E(sndr,i),rcv).

We have thus given some formal tools for the description of a parallel computation. Based on these tools, we will now formally define the concept of causality inside the system.

2.1. Causality relations between events.

Now let us introduce three relations to express the logic-temporal causality notion among events.

Relation ->: Let E(p,i) and E(q,j) be two events:
E(p,i) -> E(q,j) <=> p=q and i<j

Relation ^: Let snd and rcv be two events:
snd ^ rcv <=> I(snd,rcv) is defined.

The relations -> and ^ induces a partial ordering and expresses the temporal based cause-effect concept existent in Q.

2.2. The recovery line.

However, the conditions for a set of recovery points to be a recovery line are somewhat more complex; in fact the recovery points (corresponding in our model to set-rp events) belonging to a recovery line must be consistent between each other with respect to the interactions that happened before and after the set-rp event.

Such a requirement is rather difficult to be expressed without appropriate formal tools; now we introduce the definition of recovery line in terms of the model explained in the previous section.

This definition is slightly different from others in literature (/AND79a/,/RAN78a/) and is adequate to asynchronous environments. In

particular we make the hypothesis that messages can be received again, with a compensation action as outlined in the introduction, to obtain a consistent state of the information flow.

Suppose that a set of recoverable messages exists, that allows to recover the related information flow only by re-executing the reception.

Given a set S of events, we define the set of processes whose events are contained in S as:

$$p(S) = \{ a \mid \text{exists } i, E(a,i) \text{ is in } S\}$$

and the set of events of S belonging to a process a as:

$$S(a) = \{ e \mid e=E(a,i), E(a,i) \text{ is in } S\}$$

Definition of recovery line:

A set of set-rp events RLE and a set of recoverable messages RLM is a RECOVERY LINE for the set of processes A contained in Q, if:

1. for every e1 = E(p,i), e2 = E(q,j) in RLE, not (p = q);

2. Let snd ^ rcv
   sndr is in p(RLE) and RLE(sndr) -> snd
   =>
   rcvr is in p(RLE) and RLE(rcvr) -> rcv
   and
   I(snd,rcv) is in RLM
   <=>
   rcvr is in p(RLE) and RLE(rcvr) -> rcv
   and
   not (sndr is in p(RLE) and RLE(sndr) -> snd)

The first requirement specifies that a process cannot have two set-rp events in a recovery line.

The second is composed of two parts and must be justified on the bases of our assumption about the system. The first part requires that if the following situation holds on the sender:

... - set-rp - ... - send - ...

(time advances in this --> direction, ... represents arbitrary

sequence of events)

then on the receiver must hold:

... - set-rp - ... - receive - ...

The second specifies a necessary and sufficient condition for the messages necessary to restore a consistent information flow among the processes. For this, two conditions must hold: the receive must follow the set-rp event in the receiver:

... - set-rp - ... - receive - ...

and the send must not precede the set-rp in the sender:

... - send - ... [- set-rp -] ...

(the [ ] indicate that set-rp may happen or not).

This definition has two important properties. The first is that the recovery does not involve all the processes of the system, but only those which received information produced after a set-rp event of the recovery line. The second is that only a subset of the messages exchanged in the system must be saved to make possible a consistent recovery. The conditions that bind such subset are necessary and sufficient, and no further minimization is made possible.

2.3. The planned recovery points

We define a structure aimed to maintain information useful to the determination of recovery lines. This structure is the Planned Recovery Point (PRP) and is composed by:

T: a timestamp (a natural number);

PE: a subset of Q;

MS: a set of messages;

Associated with each event in the system there is a set (PRPset) of PRPs, representing the recovery points available to the process at the time when the event occurred.

Further, for the computation of the timestamp, we will suppose that each process keeps a counter T.

We define the total functions PRPset and T that formalize these associations as:

```
PRPset: Ev -> P(N x P(Q) x P(M))

T: Ev -> N
```

The functions PRPset and T satisfies the following axioms:

let PRPset(e1)[n] be the PRP associated at the event e1 = E(p,i) with PRP.T = n;

1st axiom:
T(E(p,0)) = 0 for all p in Q.

2nd axiom:
type(e1) = set-rp
=>
PRPset(E(p,i+1))[T(E(p,i)) + 1] is:
T= T(E(p,i)) + 1
PE = {}
M = {}
T(E(p,i+1)) = T(E(p,i)) + 1.

3rd axiom:
snd ^ rcv
=>

a. T(E(sndr,i+1)) = T(snd)
T(E(rcvr,j+1)) = T(rcv)

b. for every t =< T(snd),
and
PRPset(E(rcvr,j+1))[t].PE = PRPset(rcv)[t].PE U {sndr}

c. for every t such that T(snd) < t <= T(rcv),
PRPset(E(rcvr,j+1))[t].MS =
PRPset(rcv)[t].MS U {m}

4th axiom:
type(E(p,i)) = don't care
=>
   for every t, PRPset(E(p,i),t) = PRPset(E(p,i+1),t)
   and
   T(E(p,i+1)) = T(E(p,i))

The function T has the following properties:

THEOREM 1

For every distinct e1,e2:

(a) e1 ^ e2 => T(e1) =< T(e2)

(b) e1 -> e2 => T(e1) =< T(e2)

COROLLARY

For every distinct e1, e2:

a. T(e1) < T(e2) => e1 -> e2

b. if type(e1) = set-rp
    e1 -> e2 => T(e1) < T(e2)

c. if type(e2) = set-rp
    T(e1) =< T(e2) => e1 -> e2

Now we show that appropriate sets of RPRs identify recovery lines of a system:

THEOREM 2.
Let (Q,Ev,M,I,E) be a concurrent system;
    End be the set of events maximum of Ev for ->;
    P be a subset of Q;
    n be a natural such that
      for every p in P, T(End(p)) >= n;
    A be the minimum subset of Q such that:
      q is in P
      or
        p is in A and q is in PRPset(End(p))[n].PE
        =>
        q is in A

The set of events:
 RLE = {e | e is in ~type(set-rp)
            T(e) = n-1 }
 and
 p(RLE) = A
and the set of messages:
 RLM = U(p in A) PRPset(End(p))[n].M is a recovery line.

The theorem 2 shows that, starting from the set of "current" PRPsets (that is those related to the most recent events), it is possible, given a set of PRPs of identical timestamp n, to find a recovery line which contains them, without the possibility for a domino effect to occur.

From this we can conclude that virtually each PRP of the system is contained in some recovery line; in fact, to find a recovery line passing from the k-th PRP of process p, it is enough to apply the rule described in the theorem 2 for a P = { p } and n = k.

It is important to note that, to keep consistent its recovery points, it is enough that a process is aware of its own send, receive and set-rp events, and that it receives, together with each message, the timestamp associated to the corresponding send event.

We give now an example which shows how the correlation among recovery points in a set of processes evolves in different instants in its history.

Let us consider Fig. 1.

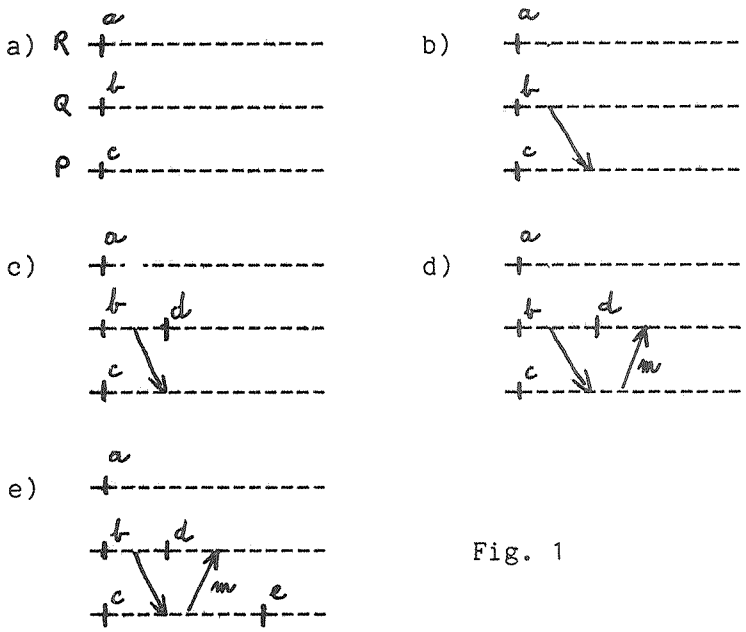

Fig. 1

The system is composed by processes P,Q and R.

In Fig. 1a these processes independently produce set-rp events. For what has been shown in Theorem 2, the available recovery lines identified by a set of recovery points RLE and a set of messages RLM are:

({a},{}) ({b},{}) ({c},{}) ({a,b},{})
({a,c},{}) ({b,c},{}) ({a,b,c},{})

Which of these recovery lines is to be used in case of failure depends on the set of processes which is involved in the failure, that is the P set in theorem 2.

In Fig. 1b after the pair of send-receive events among processes P and Q the recovery point b cannot any more used independently by c and viceversa. Therefore the available recovery lines are:

({a},{}) ({b,c},{}) ({a,b,c},{})

In Fig. 1c after the set-rp event in Q, the available recovery lines are:

({a},{}) ({b,c},{}) ({a,b,c},{})

({d},{})

In Fig. 1d after the send-receive among processes P and Q, the available recovery lines are:

({a,b,c},{}) ({d},{m})

Finally, in Fig. 1e, after the set-rp event e in P, the available recovery lines are:

({a,b,c},{}) ({d},{m}) ({e},{}) ({d,e),{m})

All these recovery lines are consistent with the definition given in sect. 2.2; the snapshots of the system in Fig. 1 are consistent with the axioms given in sect. 2.

The important thing to note is that each process can, on the basis of the local knowledge of the events it produces, adapt dynamically the correlation among its recovery points and the others in the system, to keep usable recovery lines at any instant.

These characteristics allows us to claim that the management of the PRPsets can be implemented in a distributed way. For a further insight of implementative aspect, we refer to the companion paper /BRI83c/ and to /BRI83a/.

3. Conclusions.

From the previous theorem we can conclude that, once PRPset is kept consistent with their specification and the system respects its axioms, it is possible to find a recovery line for each of the recovery points of the system.

Further, the axioms are defined in a way which is compatible with a decentralized implementation, where each process keeps consistent its own PRPset structure with the one of other processes.

A relevant open problem is the coordination of the discarding of recovery points; in fact the solution we give tends to keep all the recovery points utilizable, with an associated cost in terms of storage overhead; it could be quite valuable to be able to select which of the recovery points are really useful, and must be kept consistent.

Others open problems are the coordination of recovery points for the determination of a recovery line in presence of spare processes and

the unreliability of communications.

All these problems are currently object of our research, with the aim of extending the use of the model in such side fields.

4. REFERENCES

/ADA80a/ "Reference Manual for the ADA Programming Language ", US Dept. of Defense, July 1980.

/AND79a/ T. Anderson, P.A. Lee, S.K. Shrivastava "System Fault Tolerance", in "Computing Systems Reliability: an advanced course", ed. T. Anderson, B. Randell, Cambridge Univ. Press, Cambridge, 1979, pp. 153-210.

/BRI83a/ D. Briatico, "Trattamento dei guasti decentralizzato per sistemi multimicroprocessori: aspetti formali ed implementativi per il prototipo MuTEAM", graduation thesis, Pisa University, Nov. 1983.

/BRI83c/ D. Briatico, A. Ciuffoletti, L. Simoncini, L. Strigini, "Error Detection / Fault Treatment in the MuTEAM system", submitted for publication, Nov. 1983.

/HAN78a/ P. Brinch Hansen "Distributed Processes: a concurrent programming concept", comm. of the ACM, vol. 21, n.11, Nov. 1978, pp. 934-941.

/HOA78a/ C.A.R. Hoare, "Communicating Sequential Processes", Communications of the ACM, Vol. 21, No. 8, 1978, pp. 668-679.

/LAM81a/ B.W. Lampson "Atomic Transactions" in "Distributed Systems: Architecture and Implementation", B.W. Lampson, M. Paul, H.J Siegert, Eds., Lecture Notes in Computer Science, Vol. 105, Springer-Verlag, pp 246-264.

/MER78a/ P.M. Merlin, B. Randell "State Restoration in Distributed Systems" 8-th Annual International Conference on Fault Tolerant Computing, pp 129-134, June 21-23, Paris.

/RAN75a/ B. Randell "System Structure for Software Fault Tolerance", IEEE Transactions on Software Engineering, Vol. SE-1, No 2, June 1975, pp. 220-232.

/RAN78a/ B. Randell, P.A. Lee, P.C. Treleaven, "Reliability Issues in Computing System Design", Computing Surveys, Vol. 10, No. 2, 6-1978, pp. 123-165.

/RUS80a/ D.L. Russel, "State Restoration in Systems of Communicating Processes", IEEE Transactions on Software Engineering, Vol. SE-6, No. 2, Mar. 1980, pp. 183-194.