Consiglio Nazionale delle Ricerche

STITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

A GRACEFULLY DEGRADABLE ALGORITHM FOR BIZANTINE AGREEMENT

F. Di Giandomenico, M. L. Guidotti, F. Grandoni
L. Simoncini

Nota interna B4-49
Settembre 1986

A GRACEFULLY DEGRADABLE ALGORITHM FOR BYZANTINE AGREEMENT.

F. Di Giandomenico*, M. L. Guidotti*, F. Grandoni**, L. Simoncini**

* Dipartimento di Informatica, Pisa, Italy.
** IEI - CNR, Pisa, Italy

ABSTRACT.

An algorithm for the Byzantine Agreement without authentication in a set of n processes is presented. This algorithm has the peculiarity of being more efficient as less malicious the behaviour of the faulty processes is and less the number of actual faulty processes is. If the number of actually faulty processes is less of t/2 (t is the maximum allowable number of faulty processes) it is shown that the proposed algorithm converges very quickly to the agreement. In a system composed of 3t processes and one sender process, if at most $\lfloor t/2 \rfloor + 1$ faulty processes are present (except the sender), reaching the agreement requires 3 rounds and $2(n-1)$ messages exchanged per process. A comparison with known algorithms based on similar hypotheses is performed.

1.-INTRODUCTION

The Byzantine Generals problem arises in systems composed of a number, n, of independent, cooperating processes, at most t of which can fail, exhibiting an unpredictable behaviour.

A given process S in the system, called the sender, broadcasts a value $v \in V$ to all the other processes. Each of them must agree on a value in the set $V \cup \{d\}$, where d is a default value, in such a way that:

IC1: All correct processes agree on the same value; and

IC2: If the sender S is correct, then all correct processes in the system agree on the value sent by S.

If it is not required that all processes determine their value at the same time, the problem is referred to as " Eventual Byzantine Agreement" (EBA) [DRSa]. The algorithm presented in this paper gives a solution to this subclass of the general problem.

In the system considered here the following assumptions hold:

* Any process can directly communicate with any other process;
* Communications lines are reliable;
* A receiving process can always identify the sender process;
* Processes step through synchronous phases. This assumption allows detecting missing messages.

No authentication technique is required for messages.

The algorithm POM (Pruned OM) presented here is based on the same idea inspiring the algorithm OM described in [LSP]: the agreement on the value sent by S is pursued by reaching agreement on the value that each process in the system received from S. But in POM the information carried by messages is analyzed during each algorithm step, trying to detect the conditions that guarantee the agreement on a unique value. As a consequence, processing time and information exchange required by POM depend on the actual number of faulty processes, and on how these processes behave. The upper bounds to the values of these parameters are the same as in OM, that is t+1 "rounds" and $O(n^t)$ respectively; but the main result of the present work is that, if the actual number of faulty processes is lower than t/2 the complexity of POM is considerably lower than that of OM, as it will be shown in Section 4.

The complexity of POM will be compared to relevant solutions to the same problem presented in the literature. The algorithms presented in [DRSb], named D1 and D2, and the randomized algorithms reported in [CC],[P], referred to as CC and P, will be examined.

In Section 2 the proposed algorithm POM is described; in Section 3 the correctness proof is given and in Section 4 the complexity of POM is evaluated in comparison with known algorithms based on similar hypotheses.

2. THE POM ALGORITHM

2.1. Preliminary definitions

In the same way as in OM, the agreement value pertinent to S is determined by a process p on the basis of the agreement values obtained for any other process which received a message from S. The activity executed by p to obtain this result will be referred to as activity of p "in the S context".

For each process q that partecipates to the agreement on the value sent by S, the activity of p in the S context implies the activity of p in the context Sq, which in turn implies the activity of p in the contexts Sqr, where r is a process which partecipates to the agreement on the value re-sent by q ...

More generally a context will be named "Slr", where l is a string (possibly of null length) of process names which have re-sent, in the given order, the message received from S until r, and r is the process which has retransmitted to p the message from S upon which p is determining the agreement value.

If l is of null length, r may coincide with S; in this case the actual context is S.

Note that, owing to the algorithm structure, Slr is a string without repetitions.

During the execution of POM, p is concurrently active in several contexts, so that the agreement value found in the context

Slr is used by p to obtain the result pertaining to the context Sl, to which Slr belongs.

When p has determined the value v in V U {d} as the result for the context Srl, it will be said that "p reached termination on the value v in the context Srl". When p has reached the termination in a context Sl' "containing" Slr, that is Sl' is a prefix of Slr, it will be said that "p reached indirect termination in the context Slr".

Relevant information obtained by the process p in its activity in the context Slr is collected into the tuples R(Slr), A(Slr), T(Slr), having dimension equal to the number of partners in Slr. The meaning of the tuples is as follows:

R(Slr)(q) is the message value that process q has received from r and subsequently retransmitted to p; if q ≡ p this is the value that p has received directly from r.

A(Slr)(q) is the value on which the process p has reached termination in the context Slrq ; if q ≡ p this is again the value received by p directly from r.

T(Slr)(q) is the value of a special message, called "termination message", which q sends to p when q succeeds in determining the agreement value in the context Slr.

The agreement algorithm uses the following functions, defined over a domain of tuples of values in V U {d}; if m is the dimension of the generic tuple M, the number of elements actually specified in the tuple will be indicated by m' (0<=m'<=m).

maj(M) is a majority function over M:

$$maj(M)=\begin{cases} v \in V \text{ if } fr(v,M)\geq \lceil (m+1)/2 \rceil \\ d \text{ if for all } v \in V,\ fr(v,M)+(m-m') < \lceil (m+1)/2 \rceil \\ \text{undefined otherwise} \end{cases}$$

where fr(v,M) is the number of occurrencies of v in the tuple M.
maj(M) is not defined if m-m' >= ⌈(m+1)/2⌉; it is always defined if M is completely specified (m'=m).

majq(M) is a special majority function over M:

$$majq(M)=\begin{cases} v \in V \text{ if } fr(v,M)\geq \lceil (m+1)/2 \rceil +(t-1) \\ d \text{ if for all } v \in V \text{ in } M\ fr(v,M)< \lceil (m+1)/2 \rceil -(t-1) \\ \text{undefined otherwise} \end{cases}$$

The function majq is only defined over completely specified tuples. If majq(M) is defined then majq(M)=maj(M).

2.2 Termination in the Slr context

In the POM algorithm termination in the context Slr can occur in three different ways:

1. R-termination
   Process p reaches R-termination in the context Slr on the value v ∈ V U {d} if

$$v=majq(R(Slr))$$

2. A-termination
   Process p reaches A-termination in Slr on the value v ∈ V U {d} if

$$v=maj(A(Slr))$$

3. T-termination
   Process p reaches T-termination in Slr on the value v ∈ V U {d} if

$$fr(v,T(Slr)) >= t+1$$

The value v obtained at the termination in the context Slr is used by p to determine the agreement value in the context Sl.

If Slr ≡ S, p has completed the execution of the algorithm POM(t); the final value obtained is the result of the agreement.

Upon termination in the context Slr, the process p sends a special message to each process, which is still active in the same context; this message signals the termination of p, and carries the termination value. The special message is needed to distinguish the behaviour of the (correctly) terminated process from that of the faulty ones, which can manifest their fault by stopping sending messages.

After receiving a termination message from p, the process q uses the termination value in place of the values required from p in the sequel of the algorithm, in the context Slr in which p reached termination. This by no means implies that special reliance is credited to the termination value. Moreover, q sends to p no more messages relative to the context Slr.

The termination value is used by q until it receives from p a new termination message, pertaining to a context Sl' containing Slr.

2.3. Active and passive processes

The processes in the system are arranged in two sets, as in [DFFLS]:

* the set NA of active processes, composed by S and 3t more processes;

* the set NP of passive processes, composed by all remaining processes.

The active processes execute the full algorithm, seeking the

agreement value relative to S by determining the values relative to the other processes in NA.

Passive processes wait for receiving termination messages in the context S, and are allowed to reach T-termination only.

This behaviour may entail the need of two more message-exchange rounds, namely up to t+2, for the execution of POM by all the processes in the system.

The introduction of passive processes allows a further reduction in the message traffic, the full algorithm being executed by only the minimum number of processes required to tolerate the expected number of faults t.
Moreover, partitioning a system designed to be t-faults tolerant into the sets NA and NP increases the probability that not all the actual faults be in NA, so favoring the conditions where POM performs better.

2.4. The POM algorithm

In the description of the algorithm the following functions are used:

* SENDERS(p,k):
  determines the set of prefixes having format Slr, such that: i) p has not yet terminated in Slr, and ii) r is a sender to p in the round k-1.

* RECEIVERS(Slrp):
  determines the set of processes that partecipate to the agreement in the context Slr; this set does not include the processes already terminated in Slr, namely processes q such that T(Sl')(q) is defined, where Sl' is a prefix of Slr.

* SELECT(ENDMSG):
  this function selects from the set of termination messages (which is its argument) all termination messages relative to the generic context Slr, such that no termination message exists for any context Sl', where Sl' is a prefix of Slr.

To keep the notation simple, the tuples R, A, T relative to process p will be indicated by Rp, Ap, Tp respectively.

The normal messages sent by process p have the format: "Slrp: v".

The termination messages relative to the context Sla have the format: "*Sla: v".

The notation M(q)=? means that the q-th element in the tuple M is not defined.

```
     POM(t) /executed by process p/

k:=1;

TERMINATED   is a boolean variable having initial value "false";

if p belongs to NA

     then CURRENTCONTEXTS  is an empty set of prefixes (i.e.  con-
          texts);
          ENDMSG is an emty set of termination messages;
          while     k <= t+1 and not TERMINATED do
                    A_ROUND(k);
                    k := k+1
          od;

     else while     k <= t+2 and not TERMINATED do
                    P_ROUND(k);
                    k := k+1
          od.


     P_ROUND(k)

for all "*S: v" /received from q ∈ NA in the round k/ do
     T(S)(q) := v
od;
if it exists v ∈ V U {d} such that fr(v,T(S)) >= t+1
     then "the agreement value for S is v";
          TERMINATED := true;

     A_ROUND(k)

if   k=1
     then if p=S
               then send "S: v" to all processes in NA-{S};
                    TERMINATED := true

               else if   not received "S: v"
                         then Rp(S) := Ap(s) := d
                         else Rp(S) := Ap(S) := v
     else

------------ sending messages in the round k ------------

          for all "*Sla: v" in SELECT(ENDMSG) do
               if   Sla=S
                    then send "*S: v" to RECEIVERS(Slap) U NP
                    else send "*Sla: v" to RECEIVERS(Slap)
          od;

          ENDMSG := ∅;

          CURRENTCONTEXTS := SENDERS(p,k);
```

```
if CURRENTCONTEXTS = ∅
     then TERMINATED := true;
          round k is terminated

for all Slr in CURRENTCONTEXTS do
     send "Slrp: R(Slr)(p)" to RECEIVERS(Slrp)
od;

--------- receiving messages in the round k ------------

for all msgs /received from a process q/ do
     if msg is "Slrq: v" and Slr is in CURRENTCONTEXTS
          then R(Slr)(q) := v ;
     if msg is "*Sla: v"
          then for all Slal' in CURRENTCONTEXTS do
               R(Slal')(q) := v ;
               T(Sla)(q) := v
               od ;
od ;

--------- substitution  of values for already-------------
terminated or faulty processes

for all Slr in CURRENTCONTEXTS do
     for all R(Slr)(q) = ? where q is in NA-{Slrp} do
          if  T(Sl')(q) ~= ?  where Sl' is  the  minimal
                                lenght prefix of Slr
               then R(Slr)(q) := T(Sl')(q)
               else R(Slr)(q) := d
     od ;
od ;

if k < t+1
     then for all Slr in CURRENTCONTEXTS do
               for all q in NA - {Slrp} do
                    R(Slrq)(p) := R(Slr)(q) ;
                    A(Slrq)(p) := R(Slr)(q)
               od ;
          od ;

--------- T - termination ---------------------------

for all "*Sla: v" received in the round k do
     if it exists v ∈ V U {d} such that
                         fr(v,T(Sla))>=t+1
          then insert "*Sla: v" in ENDMSG ;
               T(Sla)(p) := v ;
               if Sla ≡ S
                    then "the  agreement value for S
                          is v" ;
                         round k is terminated

                    else A(Sl)(a) := v
od;
```

```
--------- R-termination ---------------------------

          for all Slr in CURRENTCONTEXTS do
               if majq(R(Slr)) = v
                    then insert "*Slr: v" in ENDMSG;
                         T(Slr)(p) := v;
                         if Slr ≡ S
                              then  "the agreement value for S
                                     is  v";
                              round k is terminated

                              else A(Sl)(r) := v
          od;

--------- A-termination ---------------------------------------------------

          if k = t+1
               then for all Slr in CURRENTCONTEXTS where p has not
                    yet terminated  do
                    A(Slr)(q) := R(Slr)(q)
                    od;

          for all A(Sla) where Sla is a prefix proper to round h,
                         1<=h<=k-1, such that T(Sla)(p) = ?  do
               if   maj(A(Sla)) = v
                    then insert "*Sla: v" in ENDMSG;
                         T(Sla)(p) := v;
                         if   Sla ≡ S
                              then "the agreement value for S
                                     is v";
                                  round k is terminated

                              else A(Sl)(a) := v
          od;
```

3. CORRECTNESS PROOF

In the following, the part of the POM(t) algorithm relative to the agreement in the context Slr will be indicated by POM(t)¦Slr.

Let L be the length of the string Slr (1 <= L <= t+1): then the active processes need at most k=t+2-L rounds to complete POM(t)¦Slr.

The proof will start by showing the correctness of POM(t) as executed by the processes belonging to the set NA of active processes, which has cardinality 3t+1 by definition.

LEMMA 1

Let Slr be the context relative to the agreement on the value that the non-faulty process r has received through the path Sl and re-sent to all processes in NA-{Slr}; let v ∈ V be the value re-sent by r.

If at most t processes in NA-{Slr} are faulty, and the non-faulty processes that have indirectly terminated in Slr, if any, have terminated on the value v, then any other non-faulty process in NA-{Slr} which has not indirectly terminated in Slr reaches termination in Slr on v by executing POM(t)|Slr.

Proof : by induction on k which is the maximum number of rounds of POM(t)|Slr.

Induction basis: k = 1

Every process p, which is non-faulty and is not indirectly terminated, does terminate in Slr on the value received from r, by executing POM(t)|Slr. Since r sent the value v, this is also the value on which p terminates.

Induction step: k > 1

Assume that the lemma holds for k-1.

Let p be a process which is non-faulty and not indirectly terminated in Slr.

Since p is non-faulty and, from the hypothesis, v is the value substituted in place of that pertaining to non-faulty processes indirectly terminated in Slr, then

$$Rp(Slr)(q) = v$$

for every non-faulty process in the context Slr.

It follows that the number of elements of Rp(Slr) having value v is not less than the number of non-faulty processes.

Since |NA-{Slr}| = 3t+1-L and in this case 3t+1-L >= 2t+1 since L <= t, then the non-faulty processes constitute a majority in the set of processes partecipating to the agreement in Slr, that is

$$maj(Rp(Slr)) = v$$

Now it will be shown that p terminates on the value v, independently of the kind of termination reached in Slr by executing POM(t)|Slr.

a) Process p R-terminates in Slr.

Since maj(Rp(Slr)) = v, if p has R-terminated then

$$majq(Rp(Slr)) = v$$

therefore the termination value is v.

b) Non-faulty processes that have terminated in Slr before p, if any, have R-terminated.

There are two cases:

b1) Process p A-terminates in Slr.

The algorithm execution by p in the contexts Slrq depends on:

* the termination values relative to contexts Sl', where Sl' is a prefix of Slr, if they exist. These values can only be equal to v, by hypothesis.

* the termination values relative to the context Slr, which are the algorithm result produced by processes already terminated in Slr. These processes can only have R-terminated. As shown in the point a) above, all these values must be equal to v.

It follows that all processes that have indirectly terminated in Slrq have found the value v.

Since no more than t faulty processes can be in the set NA-{Slrq}, where q is non-faulty, the induction hypotesis holds, so that it can be concluded that

$$Ap(Slr)(q) = v$$

for each non-faulty process q in NA-{Slrq}, if this element is actually computed by p.

Moreover,

$$Ap(Slr)(p) = v$$

since p has received the value v directly from r.

At last, since the non-faulty processes are the majority in the set NA-{Slr}, it follows that

$$maj(Ap(Slr)) = v$$

b2) Process p T-terminates in Slr.

Recall that the T-termination rule implies that at least one sender out of the t+1 ones which have sent the same value in termination messages as recorded in Tp(Slr) is non-faulty; let q be such a process.

Since in this case q can only have R-terminated, hence on the value v as shown in the point a) above, the process p also terminates in Slr on v.

c) The non faulty processes which have terminated in Slr before p have all found the value v.

Two cases are distinguished: A-termination and T-termination.

c1) Process p A-terminates in Slr.

The algorithm execution by process p in the contexts Slrq

depends on:

* termination values relative to contexts Sl', where Sl' is a prefix of Sl; by hypothesis these values are all equal to v;

* the termination values relative to the context Slr, which are the algorithm result produced by processes already terminated in Slr; in this case these values too are all equal to v.

Therefore the termination value v is determined by all the non-faulty processes which reach indirect termination in Slrq.

Then, in the same way as in the case b1, it is possible to show that p terminates in Slr on the value v.

c2) Process p T-terminates in Slr.

The proof follows the same pattern as in b2.

[] QED

The following theorem shows that POM(t) satisfies IC2.

THEOREM 2

For every t, the POM(t) algorithm satisfies IC2 in a system having more than 3t processes, at most t of which are faulty.

Proof

Consider the non-trivial case where the sender S is non-faulty; let v ∈ V be the value sent by S to all other processes.

The thesis follows by applying the Lemma 1 to the context S.

[] QED

Lemma 3 states that the occurrence of the conditions that allow a non-faulty process to R-terminate in a context Slr on a value v implies that all the other non-faulty processes in the system will terminate on the same value v, regardless of the kind of termination they will arrive at.

LEMMA 3

Let Slr be the context relative to the agreement on the value that the process r has received through the path Sl and re-sent to all processes in NA-{Slr}; moreover, suppose that no process seeking agreement in the context Slr be indirectly terminated in Slr.

If at most t processes out of those in NA-{Slr} are faulty, and a non-faulty process p ∈ NA-{Slr} R-terminates in Slr on v ∈ V, then all other non-faulty processes in NA-{Slr}, which have not yet indirectly terminated, do terminate in Slr on the same

value v.

Proof

Let m be the majority in NA-{Slr}.

Firstly it will be shown that, if the process p has R-terminated in Slr on the value v, then :

* It exists a subset M⊆NA-{Slr} such that |M| >= m

* Every process in M is non-faulty

* For every non-faulty process q ∈ NA-{Slr}:
  if q' ∈ M then Rq(Slr)(q') = v

Hence maj(Rq(Slr)) = v

In fact, since

majq(Rp(Slr)) = v

it follows that

fr(v,Rp(Slr)) >= m+(t-1)

Two cases can be distinguished.

i. Process r is non-faulty.

Then, for every non-faulty process q' ∈ M, where M is the set of non-faulty processes in NA-{Slr},

Rq(Slr)(q') = v

As the non-faulty processes in NA-{Slr} constitute a majority, then

|M| >= m,
maj(Rq(Slr)) = v

ii. Process r is faulty.

In this case the number of faulty processes in NA-{Slr} is at most (t-1); hence, in the worst case, process p may have received the (correct) value v from each of them, whether each other non-faulty process q may have received from them a value different from v.

Since Rp(Slr) and Rq(Slr) may differ only by elements coming from faulty processes it follows that:

* fr(v,Rq(Slr)) >= fr(v,Rp(Slr))-(t-1) >= m

and then

maj(Rq(Slr)) = v

for each non-faulty process q in NA-{Slrp}

* Define the set M as:

M = {q|Rp(Slr)(q) = v and q is non-faulty};

then

|M| >= fr(v,Rp(Slr))-(t-1) >= m

This result allows to proceed with the proof, by considering the different termination modes in Slr, for a non-faulty process q.

a) q has R-terminated.

Since the function majq(Rq(Slr)) is defined, then:

majq(Rq(Slr)) = maj(Rq(Slr)) = v.

b) All non faulty processes already terminated in Slr (before q) have R-terminated.

As proved in the preceding point a), the termination value for all these processes is v.

If q A-terminates in Slr, then, applying Lemma 1 to the contexts Slrq', q' ∈ M, it follows that:

Aq(Slr)(q') = v for every q' ∈ M

since all processes indirectly terminated in Slr have the termination value v.

As the set M is composed by the majority of processes seeking agreement in Slr it follows that:

maj(Aq(Slr)) = v

If q T-terminates in Slr, there are at least t+1 equal elements in Tq(Slr); at least one of them must be sent by a non-faulty process, and, as proved above, it must be equal to v. Then the same value is assumed by q as termination value for the context Slr.

c) All non-faulty processes, terminated in Slr before q, have terminated on the value v.

Consider the case where q A-terminates in Slr.

The algorithm execution by q in any context Slrr', where r'∈ NA-{Slrq}, depends on the termination values of non-faulty processes relative to the context Slr; these values can only be equal to v.

By applying Lemma 1 to every context Slrq', q' ∈ M, it follows that:

$Aq(Slr)(q') = v$ and
$maj(Aq(Slr)) = v$

If q T-terminates in Slr, the same reasoning as in point b) above proves that the termination value is v.

[] QED

The following theorem proves that POM(t) satisfies IC1.

THEOREM 4

For every t, let NA be a set, having cardinality $n > 3t$, composed by processes which execute the POM(t) algorithm to reach agreement on the value sent by a process $S \in NA$; no more than t processes can be faulty. If a non-faulty process $p \in NA-\{S\}$ terminates for S on the value $v \in V \cup \{d\}$ and no other non-faulty process has terminated before p (in a preceding round), then any non-faulty process in $NA-\{S\}$ will terminate for S on v.

Proof: by induction on t

Induction base: $t = 0$

The thesis is true since there are no faulty processes.

Induction step: $t >= 1$

Assume that the theorem holds for t-1; it has to be proved that it holds for t.

Observe first that if S is non-faulty then Theorem 4 follows from Theorem 2. Therefore in the following the sender S is assumed to be faulty.

1.- The termination value is $v \in V$.

The process p can have terminated for S in executing POM(t) exclusively by R- or A-termination.

1a.- R-termination.

Since, from the hypothesis, $n >= 3t+1$, the thesis follows from Lemma 3.

1b.- A-termination.

In this case $v = maj(Ap(S))$.

Let $I(p) = \{r | Ap(S)(r)=v\}$.
The activity of p in the contexts Sr, for every $r \in I(p)$, does not depend on termination values for S originated by non-faulty processes.

Any other non-faulty process q will only A- or T-terminate for S, and only in the same round as that of p or later.

1b1.- q terminates in the same round as p.

Since no non-faulty process can have terminated for S before p and q, process q cannot T-terminate, so it can only A-terminate. The activity of q in any context Sr did not depend on termination values originated by non-faulty processes.

In this case the execution of POM(t)¦Sr is equivalent to the execution of POM(t-1) by the processes seeking the agreement on the value sent by r. Then from the induction hypothesis it follows that q terminates in Sr on v by executing POM(t)¦Sr if $r \in I(p)$ and Aq(S)(r) is defined.

This implies that

$$I(p) \subseteq I(q)$$

at least potentially, meaning that q needs not be actually terminated in every Sr such that $r \in I(p)$, if this has not been necessary to determine maj(Aq(S)).

Since

$|I(p)| \geq \lceil n/2 \rceil$ (i.e. the majority over a (n-1) elements tuple) then

$maj(Aq(S)) = v$

1b2.- q terminates for S after p.

In this case q can both A- and T-terminate.

Suppose, in both cases, that the round k where q has terminated be the first round where some non-faulty process does terminate for S, after the round where p itself has terminated.

Examine first the case where q has A-terminated.

The activity of q in the contexts Sr, $r \in I(p)$, where it is still acting in the round k, depends on the termination values for S originated by p and possibly by other processes terminated in the same round as p. These values, which are all equal to v as just shown, are used by q in place of the messages that q would receive from those processes in the contexts Sr.

Note that in this way q uses informations that are extraneous to the context Sr for what concerns the "origin", whereas the information contents are still pertinent, as p (and the other processes, if any) has already terminated in Sr on v . Therefore their termination for S does not influence the activity of q in Sr with improper values.

This observation allows to establish again that the execution of POM(t)¦Sr in every context Sr is equivalent, as far as q is concerned, to the execution of POM(t-1). Therefore it is possible to apply the induction hypothesis to conclude that q terminates on v in each context Sr, such that $r \in I(p)$ and Aq(S)(r) is defined.

Hence, at least potentially:

$$I(p) \subseteq I(q)$$
$$maj(Aq(S)) = v$$

As just shown in the case of p, the termination of q for S does not influence the activity of other processes in the contexts Sr by means of improper values.

Examine now the case where q has T-terminated.

Then at least one of the (t+1) equal-valued termination messages for S received by q has been sent by a non-faulty process q'; since q' has terminated before q, it must have A-terminated in the same round as p. Therefore the value found by q', and hence by q, is v. As in the preceding cases, the termination of q does not introduce extraneous values in the contexts Sr, $r \in I(p)$.

The reasoning followed so far in the case of q "first terminated" after p can be replicated for the process q' "first terminated" for S after q, since it was proved that processes terminated in the same or in a preceding round as q do not originate values effectively extraneous in the contexts Sr, where $r \in I(p)$.

The thesis follows by carrying out this iteration.

2.- The termination value is v=d.

This means that p has terminated for S on d.

Suppose, by contradiction, that a non-faulty process q be terminated for S on $v \neq d$.

Then, as a consequence of point 1.- above, all non-faulty processes, including p, will terminate for S on v, thus contradicting the hypotheses.

Therefore all non-faulty processes do terminate for S on d.

[] QED

As a conclusion of the correcteness proof, observe that the algorithm POM(t) executed by processes in NA terminates in finite time, since it is composed by t+1 rounds, each of finite duration.

In fact, if the non-faulty process $p \in NA$ has not yet terminated in the t-th round, in the (t+1)-th round it completes all the A tuples relative to contexts Slr, Slr having length t, where it has not yet terminated.

Note that if, as in this case, Ap(Slr) is completely specified, then the function maj(Ap(Slr)) is defined. Therefore all the agreement values relative to all the mentioned contexts Slr, where p has not indirectly terminated, are defined. These values allow the complete specification of the tuples Ap(Sl); then maj(Ap(Sl)) is defined, and so forth until maj(Ap(S)) is found, which is the final result of the algorithm.

The following theorem shows that the execution of POM(t) by the passive processes gives the correct result.

THEOREM 5

The algorithm POM(t) as executed by passive processes satisfies the conditions IC1 and IC2.

Proof

Let p be a non-faulty process in NP.

Process p do terminate for S.
In fact, all non-faulty processes in NA reach termination for S no later than the (t+1)-th round, and, in the next round, they send the termination messages to all active and passive processes. Then, no later than the (t+2)-th round p receives the termination messages for S by all non-faulty processes in NA. Their number is at least 2t+1, and they have all terminated for S on the same value, as proved by the Theorem 4.
Therefore p T-terminates for S no later than the (t+2)-th round.

Recall that the T-termination value v is such that

$$fr(v,Tp(S)) >= t+1$$

This implies that there exists at least one non-faulty process in NA which have sent to p a termination message for S carrying the value v.

Then the correctness of POM(t) as executed by processes in NP follows by the correctness of POM(t) as executed by processes in NA, already proved by Theorems 2 and 4.

[] QED

4.- COMPLEXITY OF POM

In this section an evaluation of POM is given, in terms of parameters usually used in the literature to measure the complexity of Byzantine Agreement algorithms, namely:

* System redundancy.
  It is the number of processes the system must be composed of to tolerate up to t simultaneous faults.

It has been shown in [PSL] that 3t+1 is the lower bound to this parameter, for the algorithms which do not make use of authentication techniques.

* Number of rounds.
  This parameter gives a measure of the time required by the algorithm to complete.

If f indicates the number of faults actually present in the system, in [DRSb] it has been shown that the lower bound for this parameter is f+2, for EBA algorithms.

* Number of messages.
  This is the number of messages exchanged by all processes in the algorithm execution.

The exact evaluation of POM will be carried out only for some significant cases, since the detailed execution depends, as in any other EBA algorithm, on the actual number of faults and on their effects.

Systems composed by (3t+1) active processes only will be considered; in the more general case where passive processes are present, in computing the complexity figures one must take into account the n-(3t+1) termination messages sent to passive processes by each active processes, and possibly one more round as required by passive processes to terminate.

A comparison among POM and the algorithms D1, D2, CC, P, will be carried out and summarized in several tables.

The minimum time required by POM to complete is 3 rounds; this is also the minimum time for almost all other EBA algorithms. This happens when every non-faulty process can R-terminate for S; in this case each non-faulty process sends 2(n-1) messages.

The complexity parameters of POM when the actual number of faults in the system is f<t can be precisely given only if- f<=⌊(t+1)/2⌋ or f<=⌊t/2⌋+1, in the cases where the sender S is faulty or not, respectively.

In fact under such conditions the R-termination can not be impaired by the behaviour of faulty processes, and it can occur directly in the context S, or anyway in a number of contexts contained in S such that the total activity is remarkably reduced. As a consequence, in these cases the complexity of POM is much lower than that of OM.

The following table shows the number of messages computed for POM and OM for the case where the sender S and ⌊t/2⌋ more processes are faulty, for some value of n. The first column relative to POM reports the number of messages sent by a non-faulty process in the better case, that is when it can terminate in 3 rounds. The second column refers to one of the worst cases, when no process can terminate in a context Slr, where Slr is a prefix composed by names of faulty processes only. In this case the R-termination rule allows any non-faulty process to terminate in contexts Slr, where Sl is a prefix of length >= 1 composed by faulty processes only, and r is a non-faulty process. At round f+2 every process can terminate in any context where it is still active, since such contexts are indicated by prefixes ending with the name of a non-faulty process.

n=3t+1	f-1	POM		OM
13	2	24	1001	9031
16	2	30	1480	266644
19	3	36	6483	9714769
22	3	42	9501	419592000
25	4	48	48191	19385395155
28	4	54	64370	1200316978276
31	5	50	383653	78537881912680

Observe, comparing the values obtained for POM and OM, that the better use of the information made by POM reduces considerably the amount of information to be exchanged.

The next two tables compare the complexity of POM against that of the mentioned EBA algorithms. In both tables the column headed by "messages" contains the order of the total number of messages sent by a non-faulty process in the pertinent algorithm, given that the number of processes partecipating to the agreement be the minimum.

It should be noted also that in the number n the sender is included and that the sender terminates its activity in POM in the first round; in this round it sends its value to all other n-1 processes.

The number of rounds has been evaluated taking into account the initial round of the sender; it is an exact number for POM, D1, D2, while for P and CC, since these protocols use a probabilistic mechanism, a lower bound can only be given.

The values of n, number of rounds and of messages in the case of non-faulty sender and of at most $\lfloor (t+1)/2 \rfloor$ faulty processes are shown in the next table.

	n	rounds	messages
POM	$3t + 1$	3	$O(n)$
D1	$3t + 1$	5	$O(n^2)$
D2	$2t^2+3t+5$	2	$O(n)$
P	$3t + 1$	>= 5	$O(n)$
CC	$3t + 1$	3	$O(n)$

From this table we see that POM is either better or comparable to the other algorithms with respect to the number of messages. D2 is the algorithm which shows the lowest number of rounds, but a larger redundancy is needed for the number of partner processes.

The next table shows the results in the case in which the sender is the only faulty process in the system.

	n	rounds	messages
POM	3t + 1	4	$O(n^2)$
D1	3t + 1	7	$O(n^2)$
D2	$2t^2+3t+5$	4	O(n)
P	3t + 1	>= 5	O(n)
CC	3t + 1	>= 3	O(n)

The random algorithms have a number of messages which is lower than that of the other algorithms; in any case it should be noted that it is not possible to define an upper bound on the number of rounds for the random algorithms. The number of messages exchanged in D2 is comparable with that one of POM, since in D2 there is a larger redundancy and therefore both D2 and POM are $O(t^2)$ for what concerns the number of exchanged messages.

When the number of faults is greater than t/2, it may happen that no process in POM can early stop; in this case the POM has a behaviour which coincides with OM.

REFERENCES

[CC] B. Chor, B. Coan, "A Simple and EFficient Randomized Byzantine Agreement Algorithm", 3rd Symposium on Reliable Software in Distibuted and Data Base Systems, 1984.

[DFFLS] D. Dolev, M. Fisher, R. Fowler, N. Lynch, R. Strong, "An Efficient Algorithm for Byzantine Agreement Without Authentication", IBM Research Report RJ3428, 1982.

[DRSa] D. Dolev, R. Reischuk, R. Strong, " 'Eventual' is earlier than 'Immediate' ", IBM Research Report RJ3632, 1982.

[DRSb] D. Dolev, R. Reischuk, R. Strong, "Early Stopping in Byzantine Agreement", IBM Research Report RJ3915, 1983.

[F] M. Fischer, "The Consensus Problem in Unreliable Distributed Systems", Proc. of the International Conference on Foundation of Computing Theory, Sweden, 1983.

[LSP] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem", ACM Trans. on Programming Languages and Systems, Vol. 4, n. 3, July 1982.

[P] K. J. Perry, "Randomized Byzantine Agreement", 3rd Symposium on Reliable Software in Distributed and Data Base Systems, 1984.

[PSL] M. Pease, R. Shostak, L. Lamport, "Reaching Agreement in the Presence of Faults", Journal of ACM, Vol. 27, n. 2, April 1980.