The join algorithm: ordering messages in replicated systems†

L.V. Mancini, G. Pappalardo

Abstract

The need to ensure correct input-output behaviour and a higher level of fault-masking in the case of real-time systems has led designers to consider the applications of N-Modular Redundancy (NMR) in the construction of software. This approach permits redundant systems to be robust with respect to failures in replicated processors, and also permits the use of software fault tolerance techniques such as N-version programming. In order to ensure consistent behaviour of all nonfaulty replicated processors, these must process input requests in the same order. A suitable distributed algorithm, the 'join algorithm', is proposed that allows nonfaulty processors to agree on the order in which their input requests will be processed.

† Appeared in Proc. of IFAC SAFECOMP ’86, Sarlat, France.

Series Editor: M.J. Elphick

©1988 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

MANCINI, Luigi Vincenzo

The join algorithm: ordering messages in replicated systems

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1988.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 256.)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
PAPPALARDO, Giuseppe.

Abstract

The need to ensure correct input-output behaviour and a higher level of fault-masking in the case of real-time systems has led designers to consider the applications of N-Modular Redundancy (NMR) in the construction of software. This approach permits redundant systems to be robust with respect to failures in replicated processors, and also permits the use of software fault tolerance techniques such as N-version programming. In order to ensure consistent behaviour of all nonfaulty replicated processors, these must process input requests in the same order. A suitable distributed algorithm, the 'join algorithm', is proposed that allows nonfaulty processors to agree on the order in which their input requests will be processed.

About the author

Mr. L.V. Mancini has been at the Computing Laboratory since May, 1985 as a Research Associate.

Mr. G. Pappalardo has been at the Computing Laboratory since April, 1986 as a Research Associate.

Suggested keywords

AGREEMENT
DISTRIBUTED PROCESSING
MAJORITY VOTING
RELIABILITY
REPLICATED PROCESSING

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.64404 001.6425
U.D.C. 519.687 681.322.06
The Join Algorithm: Ordering Messages in Replicated Systems

L. Mancini and G. Pappalardo
Computing Laboratory, The University, Newcastle upon Tyne, NE1 7RU, U.K.

ABSTRACT

Abstract. The need to ensure correct input-output behaviour and a higher level of fault-masking in the case of real-time systems has led designers to consider the application of N-Modular Redundancy (NMR) in the construction of software. This approach permits redundant systems to be robust with respect to failures in replicated processors, and also permits the use of software fault tolerance techniques such as N-version programming. In order to ensure consistent behaviour of all nonfaulty replicated processors, these must process input requests in the same order. A suitable distributed algorithm, the 'join algorithm', is proposed that allows nonfaulty processors to agree on the order in which their input requests will be processed.

Index Terms - majority voting, replicated processing, distributed processing, reliability, agreement.

appeared in Proc. of IFAC SAFECOMP ’86, Sarlat, France.
1. Introduction

The need to ensure correct input-output behaviour and a high level of fault-masking in the case of real time systems has led designers to consider the application of N-Modular Redundancy ($N \geq 2$) in the construction of software. Examples of such systems are NASA's Space Shuttle with $N = 4$ [10], and SRI's SIFT system with $N \geq 3$ [4].

This approach to fault-tolerance has demonstrated the possibility of obtaining reliable computations through the replication of programs on $N$ computers and the use of a reliable decision algorithm. The decision algorithm may utilize a subset only of all the $N$ results for a decision; e.g., the first result that passes an acceptance test may be chosen. It is also possible that an acceptable decision result cannot be determined, and a higher level recovery procedure must be invoked. The decision algorithm may be entrusted to a unique component, but is often implemented $N$ times - once for each computation in which the decision result is used. In this case, only one computation is affected by the failure of any one implementation, such as a majority voter in SIFT [4].

Modular redundancy in the form of replication of processing modules with majority voting is one of the best known techniques for tolerating failures in concentrated processing. In the present paper we shall explore the application of this form of redundancy to distributed systems and describe a fully distributed protocol to prevent the occurrence of the so called sequencing failures [7]. We will consider our system to be composed of a number of nodes fully connected by means of redundant communication channels. A node will be composed of a number of tasks and voters in a classical NMR (N-Modular Redundant) configuration.

Major goals addressed by the solutions presented here are:
- to provide fault tolerant support for a wide class of programs;
- to allow greater asynchrony between executions of program replications;
- to allow software fault tolerance techniques, such as N-version programming [1] to be incorporated into distributed NMR systems;
- to decrease the number of message exchanges with respect to other solutions proposed in the literature.

The paper is structured as follows. In Section 2 we present a distributed architecture for replicated processing; in Section 3 an agreement algorithm is proposed for application in the implementation of distributed replicated systems; in Section 4 an example is presented that demonstrates how the algorithm works in a concrete case. Some concluding remarks are drawn in the last section.

2. An Architecture for Replicated Processing

The architecture of the system under consideration consists of a number of NMR nodes connected to form an arbitrary graph and communicating only by message passing. In Fig. 1 two directly connected nodes N_i, N_k are shown. The degree of replication shown per node is three, i.e. each node consists of three modules; module k (1 ≤ k ≤ 3) is further composed of a voter-task combination V_k, T_k. The function of a voter V_k is to perform majority voting on incoming messages; messages that have been voted are enqueued into the voted message queue VMQ_k. The function of a task T_k is to pick a message from VMQ_k, process it and transmit 3 (in general N) copies of the result to the NMR nodes attached to it.

With reference to Fig. 1, a failure of a single voter-task combination of N_i can be masked if at least two voters of N_k are nonfaulty, provided that (1) messages reach N_k uncorrupted
and (2) all nonfaulty modules of $N_i$ show the same input-output behaviour. (In general, fault masking is guaranteed if the majority of modules of each node is nonfaulty). Condition (2) above holds if the following sequencing condition SEQ is verified.

SEQ: All non-faulty modules of an NMR node process voted messages in an **identical** order.

Indeed, it is assumed that at any time each task maintains some state information, determined by its past input, which affects its subsequent behaviour. For example, if some nonfaulty task $T_k$ of $N_i$ processes message $m_1$ followed by $m_2$ and some other non-faulty task $T_j$ of $N_i$ processes $m_3$ followed by $m_2$, then results obtained by $T_k$ and $T_j$ for message $m_2$ need not be identical. Violation of the sequencing condition SEQ in a NMR node has been termed a sequencing failure [7].

The condition SEQ is particularly hard to meet in a concurrent processing environment, as it can be appreciated by this simple example. Let $N_k$ be a NMR node which can receive results from two different nodes $N_i$ and $N_j$ (Fig. 2). Suppose that $N_i$ and $N_j$ send
their result messages at about the same time to $N_k$, and that messages can experience variable delays during transmission.

$N_i$ $\quad$ $N_j$ $\quad$ $N_k$

Fig. 2. NMR node $N_k$ receives messages from nodes $N_j$, $N_i$.

It is thus possible that voters of $N_k$ receive messages in a different order: this will cause voted messages to be enqueued in different orders in the queues VMQs of $N_k$.

There are four possible ways of coping with sequencing failures:

(i) Specific scheduling algorithms are designed to ensure that non-faulty voters of an NMR node insert voted messages in their VMQs in an identical order [11].

(ii) The atomic message broadcast facility [2] is employed. If all non-faulty processors use this facility for broadcasting their messages to their receivers, then all non-faulty voters of a node are guaranteed to receive messages in an identical order, thereby preventing the possibility of sequencing failures.

(iii) Non-faulty processors of an NMR node periodically execute an agreement algorithm to ensure that VMQs are identical [8]. (This approach turns out to be an optimised version of the atomic broadcast based approach mentioned earlier).

(iv) Occurrence of sequencing failures are detected as exceptions and specific exception handlers are provided for recovering from such failures [7].
The first three approaches have one feature in common: they prevent the occurrence of sequencing failures; in the fourth approach sequencing failures are permitted, but there is a provision for their detection and recovery. The approach taken in this paper is the third, it requires neither the synchronisation of the replicated programs executed by redundant processors, nor a planned scheduling. It is worth noting that the agreement algorithm presented makes no assumption about the behaviour of failed modules: arbitrary faulty behaviour (a commission failure [3]) may be tolerated.

3. The Join Algorithm

In order to ensure the condition SEQ, the processors of a NMR node must agree (by a distributed algorithm) on the order in which messages in the queues VMQ will be processed. Such an algorithm must be reliable, guaranteeing the agreement also in the presence of faulty components; to this purpose the one proposed here with the name "join algorithm" exploits the signed message algorithm for interactive consistency presented in [5]. Below the Join algorithm executed by each processor is specified in Pascal as procedure Join.

We shall assume that the following global data structures are defined in the environment of join. N is the number of processors in the NMR-node. Each message is composed of a value and a unique identifier, the latter specifying the sender NMR-node and a sequence number. Voted messages are enqueued by voters into the message queue \texttt{vmq}, from which they are extracted and appended to the queue \texttt{omq} that feeds the processing task, in such a way that requirement SEQ is respected.
CONST N = {N is the number of processors in the NMR-node}

TYPE identifier = RECORD
  sender: sendertype;
  seqnumber: integer
END;

message = RECORD
  id: identifier;
  info: value;  {may be NULL}
END;

messagequeue = QUEUE OF message;

VAR vmq: messagequeue
  {voted messages queue};
omq: messagequeue;
  {ordered messages ready for processing by tasks}

Procedure Join is started at given time intervals by all replicated processors of a node simultaneously. This requires that all nonfaulty processors within a node have synchronised clocks. Join behaves as follows.

1. It first locks vmq, to prevent the voter to feed new messages while the next two steps are carried out.

2. A timer is started that expires after a time interval $\Delta$ equal to the maximum delay that messages directed to the node can experience.

3. The set myids of the identifiers of the messages in vmq is computed.

4. vmq is unlocked.

5. The processor engages with the others in a byzantine agreement upon each processor's value of the set myids. The version of byzantine agreement used, being based on signatures [9] is guaranteed to be successful under the assumption that the majority of
processors is nonfaulty; we shall assume that the communication between processors satisfies all the requirements given in [5] for consensus to be reached. By performing this agreement all nonfaulty processors compute exactly the same array allmyids; moreover, if processor n is not faulty, then allmyids[n] is the set of message identifiers received by n, i.e. n's local value of myids.

6. The set

\[ \text{commonids} = \bigcup \{ \text{allmyids}[n], \ 1 \leq n \leq N \} \]

is computed. It will be the same for all nonfaulty processors, like the vector allmyids from which it is computed. Myids may be a proper subset of commonids for two reasons: either (1) due to a difference in communication delays the processor has not yet received a message msg that some other nonfaulty processor has received, or (2) some faulty processor has broadcast an identifier id that does not appear in any received message. After the timeout \( \Delta \) (set at step 2) has expired, all messages like msg will have been received, voted and inserted in vmq; conversely, no messages with identifiers like id will have been received.

7. The processor orders the identifiers in commonids according to a priority among senders which is assumed to be statically fixed, and appends the corresponding messages (if any) to the queue omq removing them from vmq; if no message corresponds to an identifier like id at step 6, a message with this identifier and information NULL is inserted into omq. Since each nonfaulty processor's commonids are all equal and equally ordered, so will be their omq's, which guarantees the fulfillment of requirement SEQ.

It is worth noting that ordering messages according to a (fixed) precedence relation may be viewed as a particular strategy for resolving nondeterminism in distributed systems [8].
PROCEDURE Join;

VAR myids, commonids: SET OF identifier;
allmyids: ARRAY[1..N] OF SET OF identifier;

BEGIN lock(vmq);
StartTimer;
myids := Identifiers(vmq);
unlock(vmq);
InteractiveConsistency(myids,allmyids);
Union(allmyids,commonids);
Sort(commonids);
WaitTimer;
Transfer(commonids,vmq,omq)
END;

PROCEDURE
Union(ids:ARRAY[1..N] OF SET OF identifier;
VAR commonids: SET OF identifier);

VAR processor: 1..N;

BEGIN commonids := [];
   FOR processor := 1 TO N
      DO commonids :=
         commonids + ids[processor]
   END;

PROCEDURE StartTimer;

{sets a timer to the maximum message transmission delay}

PROCEDURE WaitTimer;

{returns after the timer set by StartTimer expires}

FUNCTION Identifiers(q: messagequeue):
SET OF identifier;

{returns the set of identifiers of messages enqueued in q}
PROCEDURE InteractiveConsistency
  (myids: SET OF identifiers;
   VAR allmyids:
   ARRAY[1..N] OF
   SET OF identifier);

{all nonfaulty processors compute exactly the same array allmyids; moreover, if processor n is not faulty, allmyids[n] is the set of message identifiers received by n, i.e. its local variable myids}

PROCEDURE Sort(VAR s: SET OF identifier);

{sorts q according to a fixed priority, equal for all processors}

PROCEDURE
  Transfer(commonids: SET OF identifier;
   VAR vmq.omq: QUEUE OF message);

{Transfer dequeues from vmq those messages whose identifiers are in commonids and enqueues them to omq, in the order specified by commonids; if there is no message in vmq corresponding to an identifier in commonids, then a NULL message is enqueued to omq}

4. A Simple Example

In order to clarify the algorithm of the previous section, we shall observe it at work in a concrete situation. We shall assume the topology of Fig. 2 and a degree of replication equal to three as in Fig.1.

Let the three queues vmq of $N_k$ be empty at time $t_0$ (Fig. 3.a) and $N_i$ and $N_j$ issue the messages of identifiers ($i,1$), ($i,2$), and ($j,1$) respectively between $t_0$ and $t_1$, the instant at which $N_k$ starts performing the join algorithm. We recall that, under the assumption that at least two modules in each processor are nonfaulty, any message present in any of the
vmqs at time $t_1$ must be correct. It is worth recalling that due to the different communication delays: (1) a queue may contain messages which another queue has not received yet, and (2) different queues may receive messages in a different order. Therefore the situation of the vmq's at $t_1$ may well be that depicted in Fig. 3.b.

At $t_1$ the join algorithm on the queues in Fig. 3.b is simultaneously started by the three modules of node $N_k$. Each module starts a timer, computes the set myids of its identifiers, and exchanges myids with the other modules, recording module n's value of myids in allmyids[n] (procedure InteractiveConsistency). Assuming that only module 2 is faulty, the byzantine agreement used ensures that modules 1 and 3 compute the same allmyids, e.g. those in Fig. 3.c; note that allmyids[2] has taken the fictitious value (h,5) in the nonfaulty modules 1 and 2 owing to a particularly malicious behaviour of the faulty module 2, but more likely allmyids[2] will simply contain a null value. Procedure Union is then invoked by each module to compute the sets commonids shown in Fig 3.d.

After the timer (equal to the maximum communication delay) has expired, module 1 will have received and voted the message of identifier (j,1) and module 3 will have received and voted the message of identifier (i,2). Indeed, these messages had certainly been sent by nodes $N_i$ and $N_j$, if one of the nonfaulty modules of $N_k$ has received them, and must therefore be received by all the other modules eventually. On the other hand, no message corresponding to the identifier (h,5) may be received. Thus, at this stage each nonfaulty module holds in its queue vmq a voted copy of the message corresponding to each correct identifier in commonids (Fig. 3.e). Ordering commonids, e.g. according to the fixed priority $i>j>h$, and transferring messages from vmq to omq in the same order will now yield the queues omq shown in Fig. 3.f and ensure that messages sent to node $N_k$ are processed in the same order by its nonfaulty modules 1 and 3.
As a final remark, we note that an extremely malicious faulty module might forge the identifier id of a message that has not yet arrived before the join algorithm is started, but will reach only one of the modules 1 and 3 before the timeout expires. In practice, the added fault probability due to such a failure can turn out to be so negligible that it is justifiable to ignore it. It is not difficult, however, to cope with it; two possible approaches are: taking the majority of the vector allmyids, rather than the union, or using signatures to detect malicious modules.

5. Concluding Remarks

A general strategy has been presented to prevent sequencing failures in distributed replicated systems. It has several virtues, as discussed below.

- it permits the adoption of software fault tolerance techniques; instead of merely running identical copies of a process, it gives the opportunity to run processes having different implementations but satisfying the same specification. As shown in [1] this guards against software faults;

- it allows different strategies for resolving nondeterminism, as shown in [8];

- the algorithm proposed is based on message passing, which is a natural choice for a distributed system built on a local area network, such as a highly reliable real-time system;

- it allows two levels of asynchrony: (1) the different NMR nodes in the distributed replicated system do not need a global clock, clock synchronization is only required within each NMR node among its replicated modules; (2) only the replicated Join procedures must synchronise within a node, whereas actual message processing can be carried out asynchronously;
the overhead due to the messages exchanged in the byzantine agreement may be minimized by increasing the sizes of the voted message queues on which the agreement is performed.

Further study is needed in order to optimise the performance of the solution proposed. It is quite likely, however, that the overhead due to the join algorithm may be minimised by employing a special unit dedicated to executing it.

We conclude by drawing the reader's attention to related research on replicated systems which is under way at the University of Newcastle upon Tyne. A formal specification of correctness requirements for replicated systems is discussed in [8]. In [8] the problem posed by non-determinacy in application programs and techniques for coping with it are discussed. The adoption of a given fault tolerant algorithm is often suggested by assumptions about the behaviour of failed components: in [3] the authors present a fault classification that has been developed for specifying faulty behaviour of components with replicated responses. Plans are currently being formulated to construct an experimental test bed for trying out ideas on replicated processing.

Acknowledgements

The authors are grateful to Prof. T. Anderson, and Prof. S. Shrivastava for their comments and suggestions. This work was supported by the Royal Signals and Radar Establishment of the Ministry of Defence.
References


