METHODODOLOGICAL CONSTRUCTION OF RELIABLE DISTRIBUTED ALGORITHMS

by

L. Shira and M. Rodeh*

Technical Report #361

March 1985

* IBM Israel Scientific Center, Haifa, Israel
METHODOLOGICAL CONSTRUCTION OF RELIABLE DISTRIBUTED ALGORITHMS

by

Liuba Shriya
Computer Science Dept., Technion, Haifa, Israel

and

Michael Rodeh
IBM Israel Scientific Center, Haifa, Israel

March 1985

ABSTRACT

The problem of designing distributed algorithms which operate in the presence of undetectable link failures is considered. A methodology is suggested for transforming distributed algorithms into reliable algorithms. The transformation is considered at two levels: reliable transmission through every single link, and as an alternative a reliable implementation of high level communication primitives. Application of the suggested methodology yields reliable implementation for distributed termination and k-selection.
1. Introduction

In the context of distributed computing, fault tolerance can either deal with the failure of processing elements, or with the failure of communication links or with both. We concentrate on link failures.

In an asynchronous system, it is in general unpredictable when an operation will end. In particular, a very slow link is sometimes hard to distinguish from a malfunctioning one. To model such a behavior we assume that failures are undetectable. Under such an assumption, the communication network must have redundancy by way of bi-connectivity or otherwise communication cannot be guaranteed. Thus we are after passive redundant systems, in which there is no need to observe malfunction [HOP80]. Notice that under such circumstances, no recovery can take place as there is no way to initiate any recovery procedures.

Failures may be circumvented in various levels of granularity. Some systems have redundancy at the component level [HOP80]. Looking at the individual links as the basic system components this approach leads to redundancy in the routing protocols so as to ensure message transfer through each of the individual links.

Securing peer-to-peer communication is another common approach which lends itself naturally to dynamic-routing techniques (e.g. [MS79]). Unfortunately, all these schemes assume that failures are detectable.

We do not know of any approach to reliability other than the one described in [S85] and [R84] which has a higher granularity level than peer-to-peer communication. Fortunately enough, the scheme described there is passive, so that it is applicable to our setup.

We define two natural communication primitives, called announcement and consultation. These primitives have efficient reliable implementation. The primitives are shown to be useful for the design of reliable algorithms for the problem of distributed termination (e.g. see [FRA80]) and k-selection (e.g. see [AHU]).
The alternative of securing communication on the individual link level is also discussed.

2. The computation model

Consider a network $N = (P, L)$ of $n = |P|$ asynchronous processors connected by $l = |L|$ links. The processors are engaged in solving some computational problem by executing their individual programs and by exchanging messages. We also assume that:

1. Each processor has a unique name known to its immediate neighborhood. Also, one of the processors, the leader, is distinguished from the others.
2. The time required for a message to pass through a link is unpredictable and unbounded.
3. Links have unbounded buffers.
4. Messages pass through links in a FIFO order.

2.1. The model of faults

The model of faults assumes that at most one link may fail. A malfunctioning link may recover and fail again. The impact of failures is the total loss of some of the messages which were waiting in the link buffer at the time of failure. Even if the link will later recover, lost messages never reappear.

Consider the graph defined by the non-faulty links and processors. The link connectivity of this graph is essential for the implementation of the algorithms we deal with. Thus, in the sequel we assume that the network is at least bi-connected.

2.2. The complexity measure

The complexity measure we are referring to in evaluating the efficiency of algorithms is the total number of messages of "reasonable size" which are
transmitted through the non-faulty links during the execution of an algorithm. The size of the additional processor memory dedicated to deal with faults is also considered.

3. Reliable algorithms

We are interested in algorithms that operate correctly even if during their execution a link may fail. In the sequel we refer to such algorithms as reliable algorithms.

Designing reliable algorithms is a work of art. Transforming an existing algorithm and making it reliable may be easier. We propose a scheme for transforming an arbitrary distributed algorithm (called source algorithm) into a reliable algorithm. This is done by securing the algorithm at a single link level: Then, we define two useful high level communication primitives and suggest an efficient securing scheme for them. In this way a source algorithm which is expressed in terms of the defined primitives may be transformed mechanically into a reliable algorithm.

3.1. Securing communication at a single link level

Consider a distributed algorithm $A$ which is executed by the processors of a network $N = (P,L)$. Let $u$ and $v$ be two processors connected by a link and assume that during the execution of $A$, $u$ sends a message $m$ to $v$. In the presence of a faulty link the message $m$ may get lost. Therefore, at least two copies of $m$ must be sent via link disjoint routes. A straightforward approach is to broadcast $m$ to $v$. To broadcast $m$, $u$ adds the identity of $v$ to $m$ and passes it to all its neighbours which, upon receiving the message, pass it further. As the network is at least bi-connected, at least one of the copies of $m$ will eventually arrive to $v$. Special provisions must be taken to ensure that every processor $u$ sends $m$ to every neighbour at most once. If the above provisions are taken then the cost of
 sending \( m \) reliably is \( O(|L|) \) messages. Securing all the messages of \( A \) in this way yields a reliable implementation for \( A \). The message complexity of this reliable implementation is \( O(f |L|) \), where \( f \) is the message complexity of the source algorithm \( A \).

In order to ensure that during a broadcast of a message \( m \), every processor sends this message to every neighbour at most once, the initiator of \( m \) may append a sequence number to it, while every processor remembers the highest sequence number of a message it has already sent. In cases where several broadcasts may proceed simultaneously in a reliable implementation of an algorithm \( A \) (meaning that in the source algorithm \( A \) messages are initiated simultaneously), this scheme requires \( O(n) \) local storage per process.

Achieving reliability via broadcasting is applicable to any source algorithm. A natural question is whether the efficiency of this scheme may be improved. It turns out that broadcasting is a rather common pattern of communication in distributed computing and may be viewed as a high-level communication primitive.

In the next section this primitive (called \textit{announcement}) and another one, called \textit{consultation}, are discussed and an efficient reliable implementation for them is suggested. These results are applicable to the link reliability problem as well.

3.2. High level communication primitives

High level communication primitives may serve as building blocks for the construction of distributed algorithms. The following communication primitives implicitly appear in many distributed algorithms.

1. \textit{Announcement} - a message sent by one processor to all the other processors of the network. We deliberately refrain from using the word \textit{broadcasting} to distinguish the announcement abstraction from any specific algorithm which performs a broadcast operation.
2. Consultation is a global computation of the network. More formally, a basic function is a non-trivial associative and commutative binary function \( f : D \times D \rightarrow D \); \( f_n \) is an \( n \)-consultation function built upon \( f \) if, for every \( a_0, a_1, \ldots, a_{n-1} \in D \):

\[
f_n(a_0, a_1, \ldots, a_{n-1}) = f(a_0, f(a_1, f(a_2, \ldots, f(a_{n-2}, a_{n-1})))).
\]

Consultation is the task of computing distributedly the \( n \)-consultation function \( f_n(a_0, a_1, \ldots, a_{n-1}) \), such that \( a_i \) is stored at \( P_i \), and the resulting value is obtained at the leader.

Consultation may be viewed as a computation over a set of values distributed in the network which can be implemented by a single pass which starts at the leaves of any spanning tree and ends at its root.

Many distributed algorithms use announcement and consultation as subroutines. Therefore, providing a reliable efficient implementation of these primitives (without a direct reliable support of the more basic communication primitives underlying them) could be of great help towards the development of reliable, efficient distributed algorithms.

3.2.1. Implementing the primitives in a reliable environment

Consider first the implementation of announcement and consultation in an environment in which no failures occur. By employing a single spanning tree rooted at the leader, both can be implemented rather efficiently: Announcement is easy as the leader may broadcast a message to each of its neighbors asking them to send that message on towards the leaves. Consultation is easy too as it is initiated at the leaves and proceeds towards the leader. It can be implemented very efficiently by collecting the accumulated partial computations from all the children of a processor \( p \), computing an intermediate result at \( p \) and then sending it to the parent processor. Obviously this scheme breaks down when links fail.
In the sequel we suggest an implementation of the two communication primitives which resist a single link failure. First the simple case of a ring network is considered. Then general networks are discussed. Finally, several application of the reliable implementations are proposed.

3.2.2. Faulty rings

On a ring, announcement is easy - the leader sends a message to both directions; every processor gets at least one of the copies.

Consultation is more complicated and is done by binary search. First a high level description of the implementation is given and then the algorithm is presented.

Initially, the whole ring is unconsulted; (the consulted part of the ring is the interval of processors that have already participated in the consultation). Invarianlly, the yet unconsulted part is a continuous section of the ring. The algorithm proceeds in phases. Each phase reduces by half the size of the yet unconsulted part of the ring.

In a phase \( \pi \), the leader sends a \( \text{start} \) command to the midst vertex \( v \) of the unconsulted part of the ring, via both directions. Upon receiving a copy of the start command in a phase \( \pi \), \( v \) initiates a consultation towards both directions. Upon receiving a consultation message from one direction the leader enters a new phase, recomputes the midst of the currently unconsulted part of the ring and issues a next \( \text{start} \) command.

The messages issued during the \( \pi \)-th phase of the consultation algorithm are therefore \( \langle \text{start}, \pi, v \rangle \) and \( \langle \text{consult}, \pi, v' \rangle \), where \( v' \) specifies the destination of the start message and \( v' \) is the partial result of the consultation built over \( f \) computed in the processors which this message has passed. All types of messages are forwarded by every processor from their initiator to their destination.
To formally describe the consultation algorithm the processors are numbered in a clockwise order starting at the leader which is denoted by $P_0$. For the sake of simplicity, we assume that $n = 2^k$ for some natural number $k$. Detailed local programs for the processors on the ring are presented in figure 1.

It can be easily verified that during the execution of the algorithm at each phase $i$:

1. at least one $consult$ message reaches $P_0$, reducing by half the yet unconsulted part of the ring. Thus after $\log n$ phases the entire ring is covered.

2. at most one processor can receive the two $start$ messages designated to it. Furthermore, at most two $consult$ messages reach $P_0$. Thus at most $2n$ messages are transmitted.

The message complexity of the algorithm is at most $2n \cdot \log n$.

3.2.3. Networks with arbitrary shape

In this section the reliable implementation of announcement and consultation in general networks is discussed. In the case of general networks the reliable announcement requires $O(|L|)$ message transmissions, while the straightforward implementation of consultation obtained by each processor performing an announcement, requires $O(n \cdot |L|)$ message transmissions. A more efficient solution may be obtained by preparing a reliable communication scheme beforehand.

A ring is a sparse bi-connected graph. Therefore, developing an efficient announcement algorithm is easy. To reduce the cost of announcement in general networks, we can build a sparse bi-connected spanning subgraph first:

Itai and Rodeh [IR78] showed that a bi-connected spanning subgraph with $O(n)$ links is contained in every bi-connected graph with $n$ vertices. This subgraph is obtained by augmenting any depth first search (DFS) tree with one lowest
comment: The program for the leader \( P_0 \):

\[
l := 1; \ cl := \text{undefined}; \ r := n-1; \ cr := \text{undefined}; \ \pi := 1;
\]

while \( l < r - 1 \) do

\[
m := \left\lfloor \frac{r+l}{2} \right\rfloor;
\]

send <start, \pi, m> to both \( P_1 \) and to \( P_{n-1} \);

wait until a message <consult, \pi, g> arrives;

if the message has arrived from \( P_i \) then begin \( cl := g; \ l := m \) end;

else begin \( cr := g; \ r := m \) end;

\[
\pi := \pi + 1
\]

end;

if \( cl = \text{undefined} \) then return \( \left( f(a_0, cl) \right) \)

else if \( cr = \text{undefined} \) then return \( \left( f(a_0, cl) \right) \)

else return \( \left( f(a_0, f(cl, cr)) \right) \);

comment: The program for a processor \( P_i \) other than the leader;

Note: only the first <start> message in a phase is responded.

Initially: \( \pi := 0; \)

wait until a message arrives;

assume that a message arrived from the neighbour \( P_e \);

if the message is <start, j, m> then

if \( j > \pi \) then

begin

\[
\pi := j;
\]

if \( i = m \) then send <consult, j, a_i> to both \( P_e \) and \( P_{2i-1} \);

else send <consult, j, m> to \( P_{2i-1} \);

end

else skip

else if the message is <consult, j, g> then send <consult, j, f (g, a_i) > to \( P_{2i-1} \);

---

Figure 1: Detailed local programs for consultation on a ring.
frond for every vertex for which fronds exist. We call this sparse subgraph \( \text{DFSLF} \).

Reducing a general network to the sparse \( \text{DFSLF} \) reduces the cost of reliable announcement on it to \( O(n) \) messages and the cost of reliable consultation to \( O(n^2) \) messages. Obviously, a \( \text{DFSLF} \) is useful also for securing transmission through individual links.

### 3.2.4. Constructing \( \text{DFSLF} \) reliably.

A reliable distributed \( \text{DFSLF} \) algorithm may be derived from the original \( \text{DFS} \) algorithm [HT'73,T'72]. This algorithm adjoins vertices, one by one, to the \( \text{DFS} \) tree assigning a sequence number to a vertex as it joins the tree. These numbers are used to establish the lowest frond for every vertex that has one. There is a single center of control which moves from a vertex to one of its neighbors. When the control is at some vertex \( v \), \( v \) inspects its neighbors to find a vertex which does not yet belong to the tree: If such a vertex exists, the center of control is moved there, the direct link to it is marked (this vertex joins the structure and is assigned a sequence number. Otherwise, the control is moved (backtracks) to the parent of \( v \) and a lowest frond from \( v \) is established and marked. If the control cannot backtrack, since \( v \) has no parent in the tree (i.e., \( v \) is the root) the algorithm terminates successfully.

**Moving** the control reliably from a vertex \( v \) to another vertex \( u \) is done by \( v \) announcing the identity of \( u \).

**Establishing** a lowest frond is done by \( v \) announcing the identity of the vertex \( u \) with the smallest \( \text{DFS} \) number. Then both \( u \) and \( v \) mark the direct link between them as being the lowest frond.

**Inspecting** the neighbors reliably by a vertex \( v \) is done by \( v \) sending an inquiring message to all its \( d \) neighbors and waiting for \( d - 1 \) answers. These answers are the \( \text{DFS} \) numbers for vertices already in the structure and negative answers for others. After \( d - 1 \) answers have arrived, \( v \) announces the inquiring
message designated to the vertex \( u \) whose answer is missing and waits for an answer from \( u \). \( u \) also sends its answer by announcement.

The message complexity of the DFSLF algorithm is derived as follows: The message complexity of announcement is \(|L|\). Thus moving control, establishing the lowest frond, and inspection each take \( O(|L|) \) messages per processor. Each pass of the center of control results either in a new vertex joining the DFSLF or in a backtrack through which a lowest frond is established. There are \( n \) vertices and at most \( n - 1 \) lowest fronds in the DFSLF. Thus, the overall message complexity is \( O(n \cdot |L|) \).

Note, that there is only a single center of activity in the DFSLF construction algorithm. Therefore only one processor at a time may initiate a broadcast. As a result, constant local space is sufficient for efficient broadcast provision.

3.2.5. Further improvements

In a recent paper [IR84] Itai and Rodeh have given a generalization of the scheme described in Section 3.2.2..

For a spanning tree \( T \) rooted at \( r \) let \( T[p] \) denote the path in \( T \) from \( p \) to \( r \). We say that the spanning trees \( T \) and \( S \) satisfy the \( 2 \)-tree condition for links if for every processor \( p \), \( T[p] \) and \( S[p] \) are link-disjoint. Itai and Rodeh have shown that every bi-connected network contains two such trees with respect to every root \( r \).

For peer-to-peer communication the following \( 2 \)-tree protocol for links is suggested: For a processor \( u \) to send a message to another processor \( v \), \( u \) first sends the message to the root on both trees. At least one of the two copies will arrive. Upon arrival of a message the root creates two copies and sends them to the target processor \( v \) again through the two trees. As before, at least one of the two messages will arrive.

In [IR84] \( s-t \) numbering techniques [ET76] (see also [E]) are used to impose a linear order among the processors of the network. This linear order is somewhat
similar to the linear clockwise order of the processors of a ring. We omit the
details here and only state that by employing s-t numbering in a way similar to the
implementation of consultation on a ring; consultation functions may be computed
in $O(n \log n)$ messages.

4. Applying the reliable primitives

In this section we derive reliable algorithms from known distributed algorithms by using reliable implementation of consultation and announcement.

4.1. Reliable distributed termination detection

The problem of distributed termination is that of transforming a globally
stable state (a state in which every processor is idle) to a termination state (in
which all the processors know that no message is ever going to arrive). This prob-
lem has been posed and solved by Francez [FR80] and has been reworked many
times since then.

Consider Francez and Rodêh's solution to the distributed termination prob-
lem [FR82]. Their algorithm consists of phases. In every phase three waves of mes-
sages flow through an arbitrary spanning tree.

1. $W_1$ finds out if all the processors are locally stable.
2. $W_2$ enables $W_3$.
3. $W_3$ verifies that the processors are still locally stable.

Francez and Rodêh describe their algorithm in terms of communication on a
spanning tree. However, what they really do is to use announcement for $W_2$ and
consultation for $W_1$ and $W_3$. Thus their scheme may become reliable for the price
of a factor of $O(\log n)$. 
4.2 Reliable $k$-selection

$k$-selection is a problem that has attracted many researchers. First we concentrate on the algorithm by Blum et al. [BFPRT] (also in [AHU, Algorithm 3.63]). The algorithm has a bag $B$ of elements as input.

4.2.1 The algorithm due to Blum et al.

1. If the bag $B$ is small (say no more than 50 elements); find its $k$-th element by sorting and return it. Otherwise execute the following steps.

2. Divide the elements of $B$ into bags of size 5, so that at most one bag has less than 5 elements. The elements in this special bag are called leftovers.

3. Find the median of each of the 5-element bags by sorting and form a set $M$ of all the medians.

4. Obtain the median $m$ of the set $M$ by applying the algorithm recursively to $M$.

5. Partition $B$ into 3 subbags $BL$, $BE$, $BG$ which contain the elements of $B$ which are smaller, equal or larger than $m$, respectively.

Let $B'$ be defined as follows:

$$B' = \begin{cases} BL & \text{if } |BL| > k \\ BE & \text{if } |BL| < k < |BL| + |BE| \\ BG & \text{if } |BL| + |BE| < k \end{cases}$$

if $B' = BE$ then return $m$ as a result. Otherwise, compute the value $k'$ defined by:

$$k' = \begin{cases} k & \text{if } B' = BL \\ k - |BL| - |BE| & \text{if } B' = BG \end{cases}$$

6. Apply the algorithm recursively to find the $k'$-th element of $B'$, returning it as a result.

Assume now that the elements of $B$ are distributed among the processors of a network. Various distributed implementations of the algorithm in this setup appear in [SFR83]. To assure reliability let us discuss each step at a time.
1. Counting the number of elements in \( B \) is a simple consultation operation. If the number of elements is small they can all be shipped to the leader (another consultation operation) and the median is computed there.

2. Dividing the elements of \( B \) into bags of size 5 is a consultation operation by itself. Note that it is associative and commutative.

3. This step is done locally in each of the processors.

4. Applying the algorithm recursively requires announcement (for the sake of synchronization). The distribution of \( m \) among the processors is done by another announcement. The other details of the recursive invocation are identical to the ones described in [SFR83].

5. Partitioning \( B \) to yield \( B' \) is done locally, while finding \( k' \) requires counting which is done by consultation.

6. Distributing \( k' \) and the recursive invocation of Step 6 is done by announcement.

We hope that this sketchy description is sufficient to convince that a distributed and reliable implementation of the algorithm by Blum et al. is indeed easy. Still, it leads to non-trivial complexity results: By using the complexity analysis of [SFR83] and the suggested reliable implementation of consultation and announcement, an \( O(|B|^{0.9115} \cdot n \cdot \log n) \) message complexity may be derived.

4.2.2. Probabilistic \( k \)-selection

Finding the \( k \)-th element of a bag can be done probabilistically by partitioning the bag with respect to a randomly chosen element, and proceeding in a fashion very similar to that described in Section 4.2. Choosing at random an element \( m \) out of a bag of \( b = |B| \) elements in a non-reliable network is the crucial problem here. We concentrate on the distributed random choice procedure.
suggested in [SFR83] for the case of a reliable network.

The algorithm assumes the existence of a spanning tree constructed over the processors of the network. Each processor assigns a fixed order to its own elements and to its children. Assume that a processor knows \( t_1, t_2, \ldots, t_d \) - the respective numbers of elements which reside in each of its \( d \) sub-trees (where \( d \) being its degree in the spanning tree). The random choice proceeds as follows: The root randomly chooses an integer \( i \) in the range \( 1 \) to \( b \). To find the corresponding element, it checks whether this element is one of the \( t \) elements which reside in its own memory. This happens if \( i \leq t \). Otherwise, the root sends a message LOCATE \((j)\) to the \( f \)-th child, where \( j \) is the smallest positive integer of the form 
\[
i - t - t_1 - \cdots - t_{f-1}.
\]
Upon reception of a message LOCATE \((j)\), the receiving processor acts similarly to the root. When an element has been located, it is sent to the root, to serve as the partitioning element.

The above distributed random choice procedure cannot be directly expressed as a composition of consultation and announcement operations because the element location procedure is not commutative and associative. For such and similar cases the methodological approach described in section 3.1 suggests securing communication at link transmission level. Unfortunately in the case of the \( k \)-selection algorithm this introduces significant message complexity degradation. We therefore take an alternative approach - redesign the element location procedure and give it a direct reliable implementation.

We first consider the special case of a ring network. To locate the element, instead of using the order defined by the nonreliable spanning tree, the algorithm uses the clockwise order defined by the ring. The processors are assigned sequence numbers in this order by the root initiating a numbering message in both directions. A following binary search type procedure is then carried out by the root over the ring to locate the processor containing the \( t \)-th number in the total order defined by the local orders and the ring order.
The procedure operates in phases. At each phase the root splits the ring into two parts and counts the number of elements at each part. The splitting is done by the root announcing the sequence number of the split point. The counting is done by the two vertices near the split point initiating a counting message towards the root. At least one of the countings succeeds in reaching the root. Thus the root knows in what part of the ring the processor containing the \( i \)-th number resides and places the next split point in the middle of this part. \( \log n \) phases are sufficient to locate the processor containing the \( i \)-th element. At each phase \( O(n) \) messages are transmitted. The message complexity of the reliable random choice procedure is thus \( O(n \log n) \).

For the case of a general network, s-t numbering can be used to determine a linear order among the processors, and then binary search is applicable. Again, at most \( O(n \log n) \) messages are sent.

5. Conclusion

In this paper a transformational approach is suggested to the design of reliable algorithms. We start with non-reliable algorithms, identify certain high level communication patterns and use reliable implementing for them to obtain reliable algorithms. The resulting algorithms are in many cases efficient and nontrivial. For the cases were the available repertoire of reliable high level primitives is not sufficient to express the algorithm, low level reliable announcement may be used.

6. Acknowledgements

We would like to thank Danny Dolev, Nissim Francez, Oded Goldreich and Alon Itai for very helpful discussions.
References

[AHU]

[BFPRT73]

[E]

[ET76]

[FR82]

[FRA80]

[HOP80]

[HT73]

[IR78]

[IR84]

[MS79]

[S85]

[SFR83]

[T72]