The Distributed Snapshot of K.M.Chandy and L. Lamport

We consider a distributed system of the form of a strongly connected, finite, directed graph, of which each vertex is a machine and each edge a uni-directional first-in-first-out buffer of sufficient capacity. (Strongly connected means that there is a directed path from any vertex to any vertex.)

A distributed computation consists of a succession of so-called "atomic actions"; each atomic action is performed by one of the machines: it changes the state of that machine, accepts at most one message per input buffer of that machine, and sends at most one message per output buffer of that machine. The buffers have no clairvoyance, i.e. a message can only be accepted after it has been sent.

For a message-accepting action to take place, the messages in question must have arrived; arrival of messages, however, only enables an atomic action to take place and never forces it to take place. Each message in a buffer will eventually be accepted by the machine to which this buffer leads. (The friendly daemon that resolves each machine's nondeterminacy so as to ensure the implied fairness is not considered part of the system.)

Between any two successive atomic actions the state of the distributed computation is determined by (i) the state of each machine, and (ii) the string of messages contained in each buffer.

A so-called "stable predicate" is such that, if it holds in some state, it will hold in all possible later states. If it holds, we say that "stability" has been reached. The purpose of the distributed snapshot algorithm of K.M. Chandy and L.Lamport is to collect such state information that, on account of it, stability can be detected.

The distributed snapshot algorithm is superposed upon the distributed algorithm such that, while the distributed algorithm evolves from state SO

to state S1, it collects the description of a so-called "snapshot state" SSS with the properties that --though there need not have been a single moment at which it occurred-- SSS is a state that is possible after S0 and S1 is a state that is possible after SSS . Hence, if stability has been reached in S0 , SSS satisfies the stable predicate; conversely, if SSS satisfies the stable predicate, stability has been reached in S1 .

In our first description of the snapshot algorithm each machine, each atomic action, and each message is either white or red. Each atomic action gets the colour of the machine that performs it, each message gets the colour of the action that sends it. In state S0 , all machines and all messages are white. As part of the execution of the snapshot algorithm each machine turns from white to red once, i.e. each machine's individual history is a sequence of white actions, followed by a sequence of red actions. So, eventually, all machines and all actions are red. From the onwards, no white messages are sent; since all white messages in the buffers will be accepted in due time, eventually all messages in the buffers are red as well. State S1 is a state in which all machines and all messages are red.

For the time being by magic --about which more later-- , the moments at which machines turn red are chosen in such a way that no red message is accepted in a white action. We note at this stage that such a choice of moments is possible: if, for instance, all machines turned red simultaneously, each machine would be red before the first red message entered the system and, hence, no red message would be accepted by a white machine.

The snapshot state SSS consists of

(i) for each machine its state at the moment of its transition from white to red, and

(ii) for each buffer the string of white messages accepted from it in red actions.

Our purpose is to show the existence of an equivalent computation -- i.e. equivalent with respect to the distributed computation and with respect

to the snapshot algorithm-- with its snapshot state as one of its intermediate states, i.e. with S0 as initial state, with SSS as intermediate state, and with S1 as final state. From that existence follows what had to be shown, viz. that SSS is a possible state after S0 and that S1 is a possible state after SSS .

The equivalent alternative consists of all the white actions in their original order, followed by all the red actions in their original order. To show its equivalence to the original distributed computation we observe that two successive atomic actions from different machines commute unless the first one sends a message that is accepted by the second one. From this it follows that a red action and a subsequent white one commute: from their colours we firstly deduce that such two actions are performed by different machines, and secondly deduce that, messages produced by the first one being red, the first one does not send a message accepted by the second. By interchanging them we derive a computation that with respect to the distributed algorithm is equivalent to the one before the interchange; colours of actions and messages having been left as they were, the property that no red message is accepted in a white action still holds and the snapshot algorithm, being defined in terms of colours, yields the same snapshot state.

Such an interchange, however, reduces the number of "inversions" by 1 (an inversion being a pair of differently coloured actions such that the red one takes place before the white one). Hence a finite number of such interchanges yields the equivalent alternative in which all white actions precede all the red ones. And here all machines can turn red between the last white and the first red action; the system state at that moment is evidently the state yielded by the snapshot algorithm.

* * *

We now turn our attention to two details of implementation. Firstly, we have to implement the "magic" that sees to it that each machine turns red at an appropriate moment. Secondly, we have to see to it that each machine can record the proper string of messages for each of its input buffers.

Let us begin with the last requirement. A red machine has to record for each of its input buffers the string of accepted messages up to and including the last white message. We are not allowed to translate that into "up to and excluding the first red message" since that red message need not exist. Therefore: instead of colouring the messages, we extend the repertoire of messages with a special one, called the "marker", with the convention that on each buffer the messages --if any-- that precede the marker are deemed white and the messages --if any-- that follow the marker are deemed red; like the other messages, the markers participate in the first-in-first-out regime of the buffers, and, upon turning red, each machine sends over each of its output buffers a marker before sending anything else over that output buffer.

The markers can also be used to implement the magic. Since each red message is preceded by a marker, machines turn red in time if they do so upon accepting their first marker while still white. The snapshot algorithm is initiated by (at least) one machine turning red (and, accordingly, sending a marker over each of its output buffers). Since each machine is reachable via a path from the initiator(s), and each message sent is eventually accepted, all machines will turn red in due time. Since each machine turns red once, each buffer carries precisely one marker. Hence, in each machine it is known when collection of the local snapshot information has been completed, viz. when over each of its input buffers a marker has been accepted. The local snapshot information can then be sent to a central point --probably the only initiator-- where it can be subjected to the test whether the snapshot state satisfies the stable predicate.

* * *

The above algorithm is a creation of K.Mani Chandy of the University of Texas (Austin) and Leslie Lamport of SRI International (Menlo Park); the former of the two told it to me on the 1st of July 1983. Being otherwise engaged during July and August, I returned to it in September. Several sessions of the Tuesday Afternoon Club were devoted to it; first we reconstructed the algorithm, and then we experimented with various ways of reasoning about it. The contributions of C.S.Scholten in the later stages of those experiments are

gratefully acknowledged. No one else, however, can be blamed for any shortcoming of the above text, which was written in São Paulo, Brasil.

It can be argued that we have been overspecific by viewing the original distributed algorithm as a linear sequence of atomic actions in time: no one cares about the relative order of actions that might take place concurrently. Yet I believe the linearization responsible for the fact that we could carry out the argument without the introduction of subscripts or any other nomenclature to distinguish between the machines. The linearization plays a role very similar to the choice of an otherwise arbitrary coordinate system in analytical geometry.

In comparison to earlier distributed algorithms for the detection of termination, the snapshot algorithm described above is very beautiful in the sense that it is applicable independently of the specific stable predicate.

Additional Remark. In the above we have shown that if no red message is accepted in a white action, the individual speeds can be adjusted so as to let all colour transitions of the machines take place simultaneously. The condition is sharp: if a red message is accepted in a white action, its sender turns red intrinsically prior to its acceptor. (End of Additional Remark.)

Plataanstraat 5
5671 AL NUENEN
The Netherlands

7 november 1983

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow