Distributed Termination Detection with Sticky State Indicators

Friedemann Mattern

Department of Computer Science, University of Kaiserslautern,
P.O. Box 3049, D 6750 Kaiserslautern, Fed. Rep. Germany

mattern@informatik.uni-kl.de

July 29, 1990

Distributed termination detection is a "prototype problem" of the theory of distributed computing which has gained considerable interest in recent years[1]. It is closely related to other important problems such as deadlock detection [CMH83, Na86], garbage collection [TTL88, TM90], and snapshot computation [CL85, Ma90]. The main purpose of this short note is not to introduce yet another distributed termination detection algorithm, but to demonstrate that straightforward solutions to the problem can be based on simple ideas with informal and intuitive (but yet correct) proofs.

We adopt the usual model of a distributed system consisting of n processes $P_1, ..., P_n$ communicating by synchronous messages:

1. A process is always in one of two states, *passive* or *active*.

2. An active process may activate another (passive) process by sending a message to it.

3. At any time, a process (which is not currently[2] engaged in a send operation) may change from active to passive. (To become active again, however, it must be reactivated by an active process.)

It is usually assumed that initially at least one process is active. If at some instant of time all processes are (simultaneously) passive, the system has reached a stable state, it is *terminated*. The task of a termination detection algorithm is to detect this global property. Since in a distributed system it

[1] See, e.g., [DFG83, To84] and the reference list in [Ma87].

[2] Note that activations of other processes are not necessarily "immediate". A synchronous send operation, however, is blocking—the sender therefore remains active at least until the message has been received by the receiver.

is impossible to inspect the states of all processes simultaneously even when using synchronous messages, this is a non-trivial problem.

For a first attempt to solve the problem, assume that each process P_i has a *state indicator* S_i always correctly reflecting the state of the process. Then an initiator may start a *control wave* which visits all processes and returns the values of the state indicators. (More efficiently, it could only return the "accumulated" value *passive* if all processes were passive, and *active* otherwise.) The control wave can be implemented in various ways; for example by a token circulating on a (virtual) ring connecting all processes, by a sequential or parallel distributed graph traversal scheme such as the echo algorithm [Ch82], by a (virtual) broadcast scheme on a spanning tree, or by any other total algorithm with feedback [Te90]. Unfortunately, however, the values of the state indicators collected in that way are of no use: because of possible reactivations of processes "behind the back" of the wave, the observation that all processes were passive when being inspected by the wave does not imply that all processes were passive simultaneously. An algorithm which announces global termination when it found all S_i passive could wrongly detect termination!

Fortunately, the simple scheme sketched above can easily be transformed into a correct algorithm. Assume now that the state indicators S_i are "sticky" in the following sense: If a process P_i is activated, the value of S_i becomes (or remains) active. If a process becomes passive, however, S_i "sticks" to active. Clearly, if at some moment before the start of the control wave some process P_j was active, the algorithm will not announce global termination because the value of S_j is *active* when it is eventually collected by the wave. As a matter of fact, the implicit semantics of the sticky state indicator ensures the *safety property* (i.e., no "false termination" is detected). Unfortunately, however, termination will never be announced in the scheme as it stands! In order to guarantee *liveness* ("termination will eventually be detected after its occurrence") it is necessary to repeatedly start a control wave and to periodically reset the sticky state indicators to the true values of their processes' states without compromising the safety property. Obviously, the following two properties hold:

(Safety) If at the start of the wave a process was active (and the sticky state indicators are not reset *during* the wave), termination will not be announced. (Or equivalently: If termination is announced, the system was already terminated when the wave was started.)

(Liveness) If at the start of the wave the value of each sticky state indicator S_i is *passive* (and consequently all processes are passive and the system is terminated), termination will be announced at the end of the wave.

To see the liveness property, observe that since the system is terminated

(as a consequence of the assumption), the values of the sticky state indicators will not be changed. Hence the wave will find all state indicators *passive* and will consequently announce termination. Note that the assumptions trivially hold if the state indicators are reset to their true values *after* termination occurred (and the final wave is started thereafter).

The principle presented above allows several concrete variants which can easily be implemented. If in a sequence of control waves the next wave is started only after completion of the previous one, it is probably most appropriate to reset the state indicator of a process directly after collecting its value (i.e., actually during but logically after the wave). In that way, no extra wave is necessary for the resetting of the state indicators. Furthermore, to minimize the number of control waves it is reasonable to visit the processes in a "lazy" way—a wave visiting an active process is only propagated when the process becomes passive.

The scheme can also be adapted to *asynchronous communications.* One possibility is to acknowledge each message and to consider a process to be engaged in a send operation while the acknowledgement is not received (thus simulating synchronous communication for which the acknowledgement is implicit). Obviously, this can be realized by locally counting sent messages and received acknowledgements. It is also possible to use indirect acknowledgements and to batch acknowledgements, this technique is used for example in the vector counter algorithm [Ma87].

Finally, there is a great variety of wave algorithms on which our generic scheme can be based [Te90, Ra90]. For rings with a circulating token it is similar to the well-known algorithm by Dijkstra et al. [DFG83] (whereas in that algorithm send-flags are used, our scheme implicitly uses receive-flags), for spanning trees Topor presented a variant in [To84]. An interesting challenge is the design of efficient and "intelligent" waves which try to avoid the visit of processes that have not been reactivated since the last visit.

Conclusions. We presented a simple generic and intuitively correct termination detection scheme together with an informal correctness argument based on the "sticky indicator" paradigm. It shows again that distributed termination detection is non-trivial but not necessarily complicated. We leave it to the reader to formalize the idea and to give thorough and complete proofs for concrete variants of the sticky state indicator scheme.

References

[Ch82] Chang, E.J.H, *Echo Algorithms: Depth Parallel Operations on General Graphs,* Trans. Softw. Eng. 8 (1982) 391–401.

[CL85] Chandy, K.M., L. Lamport, *Distributed Snapshots: Determining Global States of Distributed Systems*, ACM Trans. on Computer Systems 3 (1985) 45–56.

[CMH83] Chandy, K.M., J. Misra, L.M. Haas, *Distributed Deadlock Detection*, ACM Trans. on Computer Systems 1 (1983) 144–156.

[DFG83] Dijkstra, E.W., W.H.J. Feijen, A.J.M. van Gasteren, *Derivation of a Termination Detection Algorithm for Distributed Computations,* Inf. Proc. Lett. 16 (1983) 217–219.

[Ma87] Mattern, F., *Algorithms for Distributed Termination Detection*, Distributed Computing 2 (1987) 161–175.

[Ma90] Mattern, F., *Efficient Distributed Snapshots and Global Virtual Time Algorithms for Non–FIFO Systems*, Tech. Rep. SFB124–24/90, Kaiserslautern University, 1990.

[Na86] Natarajan, N., *A Distributed Scheme for Detecting Communication Deadlocks*, IEEE Trans. on Software Engineering SE–12 (1986) 531–537.

[Ra90] Raynal, M., Helary, J.-M., *Control and Synchronisation of Distributed Systems and Programs*, Wiley, 1990

[Te90] Tel, G., *Total Algorithms*, Technical Report RUU–CS–88–16, Dept. of Computer Science, Utrecht University, 1988. Also in: Algorithms Review 1 (1990) 13–42.

[TM90] Tel, G., Mattern, F., *The Derivation of Distributed Termination Detection Algorithms from Garbage Collection Schemes*, Technical Report RUU–CS–90–24, Dept. of Computer Science, Utrecht University, 1990.

[To84] Topor, R.W., *Termination Detection for Distributed Computations*, Inf. Proc. Lett. 18 (1984) 33–36.

[TTL88] Tel, G., R.B. Tan, J. van Leeuwen, *The Derivation of Graph Marking Algorithms from Distributed Termination Detection Protocols*, Science of Computer Programming 10 (1988) 107–137.