GLOBAL QUIESCENCE DETECTION BASED ON CREDIT DISTRIBUTION AND RECOVERY

Friedemann MATTERN

Department of Computer Science, University of Kaiserslautern,
P.O. Box 3049, D 6750 Kaiserslautern, Federal Republic of Germany

Keywords: quiescence detection, distributed termination, termination detection, distributed control, distributed algorithm, distributed computing

Abstract. We present a new, yet very simple principle for global quiescence detection. All active processes and all messages of the underlying computation have a share of a credit initially distributed by the environment. The sum of all credit shares is kept invariant. When the environment has recovered the whole credit, it can conclude that no processes are active and no messages are in transit. We show how the principle can be implemented efficiently.

1. Introduction

Consider a distributed system consisting of n processes $P_1,...,P_n$ communicating by so called *activation messages.* A process is always in one of two states, *active* or *passive.* The system behaves according to the following rules:

(1) Only active processes may send activation messages.

(2) A process may change from active to passive at any time.

(3) A process may change from passive to active only on receipt of an activation message.

If all processes are passive and no activation message is in transit, the system has reached a stable state, it is *quiescent.* The detection of this property is an important and non-trivial control problem for distributed systems. Typical instances of the quiescence property are termination of distributed algorithms, global communication deadlock, and end of a phase of a distributed multiphase algorithm. Beginning with [2] and [3] numerous solutions based on various principles have been published in recent years [4]. Typical solutions to the quiescence problem consist in superimposing a *detection algorithm* using *control messages* on the underlying computation and possibly modifying the communication protocol of the activation messages in some way which does not affect or obstruct the underlying computation. If (and only if) the underlying distributed computation eventually becomes quiescent, the detection algorithm should announce this fact some finite time later.

Initially, the system is supposed to be quiescent. As in [2] we assume the existence of a special process called the *environment* which 'spontaneously' starts the underlying computation by sending activation messages to some processes. Otherwise the environment does not take part in the underlying computation, i.e., it starts the computation only once and does not receive activation messages. (It does, however, play an active role in testing the quiescence criterion by receiving and processing control messages.) When the underlying computation has reached a quiescent state, the environment will eventually detect this fact. As will be seen in Section 5, however, there exists a *symmetric* variant of the algorithm where any process may detect global quiescence.

The main advantage of the proposed scheme is its simplicity as compared to other algorithms. Furthermore, it is readily implementable, has low message complexity, can cope with dynamic processes, and does not require synchronous communication or FIFO channels.

2. The detection principle

Our solution to the quiescence detection problem is based on a very simple principle. Processes and activation messages can hold a certain *share of credit* $\in \mathbb{Q}^+$. Upon start of the underlying computation the environment distributes a credit of total value 1 to the processes. The detection algorithm must ensure the following properties:

(a) At any time, the sum of all credit shares held by processes, messages, and the environment is 1.

(b) When a process is active it holds a credit share > 0.

(c) An activation message which is in transit holds a credit share > 0.

After the start of the underlying computation the environment tries to get back the credit shares. For that purpose we assume that by *control messages* the processes can send part or all of their credit shares to the environment. If it has recovered the whole sum, the environment announces global quiescence. The following theorem asserts that no "false quiescence" is detected.

THEOREM 1 (Safety). *If the environment has regained the whole credit sum, the system is quiescent.*

PROOF. If the whole credit is concentrated at the environment, then because of invariant (a) no other process and no message has a share in the credit. Because of (b) and (c), then all processes are passive and no activation messages are in transit. □

The guarantee that the whole credit is recollected at the environment some finite time after the system is quiescent requires a suitable realization of the credit collection scheme. It is easily verified that because the environment does not redistribute any credit share it regains (and because message transmission times are assumed to be finite), the following two rules ensure the desired *liveness property*.

R1. When a process becomes passive it transmits its credit share to the environment.

R2. When an activating message with credit share C arrives at an *active* process, C is transmitted to the environment.

Furthermore, any correct implementation must observe properties (a) - (c). A possible implementation could be based on the following rules:

R3. When an activation message with credit share C arrives at a *passive* process, C is transferred to the activated process.

R4. When an active process with credit share C sends an activation message, the process keeps ½C and the message gets the other half.

Clearly, the number of control messages (generated by R1 and R2) is equal to the number of activation messages of the underlying computation. The *message complexity* is therefore identical to the Dijkstra/Scholten algorithm [2]. By changing rule R2 (adding C to the credit share of the process instead of sending it to the environment), however, the number of control messages can be reduced. We do not advocate this possibility because the addition of fractions may lead to technical problems. If a process sends an activation message immediately before becoming passive, it may also transfer its *whole* credit with the activation message.

A potential problem with the scheme seems to be the partition of the credit share (R4). The values quickly become very small (i.e., the denominators of the fractions become large) and impossible to handle; especially the accumulation at the environment may pose problems. Note that because of their limited range and because of rounding errors floating point values are not appropriate. Fortunately, it is possible to use the (negative) logarithm of the values instead. This leads to feasible solutions as will be seen in the next sections.

3. A realization of the principle

The quiescence detection principle can be implemented in a straightforward way. Credit shares of processes are represented by a local integer variable CREDIT whose value stands for the credit share $2^{-\text{CREDIT}}$. Whenever an activation message is sent by a process, the credit share is split up (R4):

```
CREDIT := CREDIT + 1 ;
send <CREDIT,...> to ... ;
```

When a process becomes passive, it sends its credit share to the environment (R1):

```
send <CREDIT> to ENV ;
active := false ;  /* CREDIT := ∞ ; */
```

If an activation message with credit share CR arrives at a process, CR is transmitted to the environment (R2) or it is transferred to the process (R3) by the following atomic action:

```
if active then send <CR> to ENV ;
          else active := true ; CREDIT := CR ;
fi ;
```

Several optimizations are conceivable for messages arriving at an *active* process. If CR = CREDIT, it is not necessary to send CR to the environment, CREDIT := CREDIT − 1 would be the appropriate action. The idea can be generalized by using a local set S which stores a limited number of different credit shares. When an activation message is sent, a member of S would be used if $|S|>1$. If an activation message with a credit share $CR \in S$ arrives, S can be updated by simulating a binary addition. When the process becomes passive, the whole set S must be transferred to the environment.

In order to detect global quiescence, the environment keeps book on the *missing credits*. Conceptually, it subtracts an incoming credit CR by updating a local variable D (initialized to 1 after the start of the underlying computation): $D := D - 2^{-CR}$. It announces global quiescence if $D = 0$. Because it is unrealistic to assume that reasonable large values of CR can be handled in that way, another bookkeeping technique is proposed. The environment keeps a dynamic set DEBTS of integer values so that $\sum_{d \in DEBTS} 2^{-d} = D$. DEBTS is initialized to {0}. Whenever a control message $<CR>$ is received by the environment, DEBTS is updated as follows:

```
K := CR ;
while $K \notin DEBTS$ begin
    DEBTS := $DEBTS \cup \{K\}$ ;
    K := $K - 1$ ;
end ;
DEBTS := $DEBTS - \{K\}$ ;
if $DEBTS = \emptyset$ then quiescent fi ;
```

The correctness of the scheme (i.e., its equivalence to "$D := D - 2^{-CR}$; **if** $D = 0$...") is easily verified by observing that it constitutes an explicit realization of the canonical binary subtraction algorithm.

The following theorem guarantees the practicability of the scheme.

THEOREM 2. *At any instant, the cardinality of DEBTS is bounded by the number of credit shares currently distributed over the active processes, activation messages and control messages.*

PROOF. Let $DEBTS = \{d_1, ..., d_n\}$. The missing total credit $D = \sum_{i=1}^{n} 2^{-d_i}$ is distributed over the active processes, activation messages, and control messages in such a way that all credit shares are also negative powers of two. Since each d_i is unique in DEBTS, it is the most compact partition of D by (negative) powers of two. □

Notice that because of the implementation of rule R4, the value of each member of DEBTS is bounded by the total number of activation messages.

4. A modified principle

We now describe the realization of a variant of the principle which transmits credit shares by *control messages* only. The advantages of the variant are that no alteration of activation messages is necessary and that the amount of memory at the environment can be reduced as will be shown in Section 5. Furthermore, the actual number of control messages can be less than m (while still being bounded by 3m), where m denotes the number of activation messages.

For simplicity, we assume that a control message is only processed when the receiver is passive and that no other messages are accepted while it is processing a control message. To reduce the number of control messages, a process should remain active as long as it "knows" (based on its local state) that global quiescence has not been reached. We also assume that control messages do not overtake activation messages on the communication channels. This property can easily be realized on non-FIFO channels by equipping each communication channel with an incoming and outgoing counter for activation messages. A control message contains the number of activation messages which have previously been sent over the channel. It is only then accepted at the other end when at least that number of activation messages have been received. Notice that the contents of activation messages are not relevant to the detection algorithm, therefore activation messages may bypass other messages.

The principle of the variant can be characterized as follows. Whenever a process is activated by an activation message, it is guaranteed that a control message will visit that process *later*, some time after it has become passive. For that purpose, the control messages trail the activation messages, splitting up when necessary. When they all no longer "see" any activation message ahead of them, global quiescence can be concluded.

Note that the proof of Theorem 1 does not immediately apply because activation messages and active processes do no longer have a direct share in the credit. However, it is easily verified that the credit transmitted by the control message (which will eventually follow an activation message) takes the place of the credit which is otherwise transmitted by the activation message. Active processes always get a credit $\neq 0$ (for a rather short time) after becoming passive.

In order to trace the activation messages, each process has a local set ACTIVATED (initialized to $\emptyset$) which is updated whenever an activation message is sent to any other process P_k:

$$\text{ACTIVATED} := \text{ACTIVATED} \cup \{k\}.$$

When a control message $<$CREDIT$>$ arrives at a (passive) process, CREDIT is sent to the environment if ACTIVATED $= \emptyset$, otherwise CREDIT is divided and sent to the processes stored in ACTIVATED. Then ACTIVATED is reset to $\emptyset$.

Instead of partitioning an incoming CREDIT by successively halving it, this is done in a more ingenious way leading to an even distribution. Let n denote the cardinality of ACTIVATED and define $C_{ij} = \lfloor \log(i+j-1) \rfloor$ (all logarithms are to base 2). Then the j-th element of ACTIVATED gets $2^{-C_{nj}}$ parts of CREDIT, i.e., for j=1,...,n a control message with the value C_{nj} + CREDIT (standing for $2^{-C_{nj}} 2^{-\text{CREDIT}}$) is sent to the j-th process of ACTIVATED. (As an example, for $n = 5$ the factors $^1/_4$, $^1/_4$, $^1/_4$, $^1/_8$, $^1/_8$ are generated). Note that the canonical definition $C_{ij} = j$ if $j < i$, and $C_{ij} = j-1$ else $(j = i)$ would lead to much faster growing credit values. The following theorem asserts that invariant (a) of Section 2 is observed:

Theorem 3. $\forall\, n \geq 1: \sum_{j=1}^{n} 2^{-C_{nj}} = 1$

Proof (Sketch). Induction on n by making use of the following easily verified lemmata

Lemma 1. $\forall\; n \geq 1: \lfloor \log 2n \rfloor = \lfloor \log(2n+1) \rfloor$

Lemma 2. $\forall\; n \geq 1: 2^{-C_{n+1,n}} + 2^{-C_{n+1,n+1}} = 2^{-C_{n1}}$ □

The values C_{ij} can be more easily computed using the following theorem:

Theorem 4. $$\forall_{i \geq 1,\, 1 \leq j \leq i}: \lfloor \log(i+j-1) \rfloor = \begin{cases} \lceil \log i \rceil - 1 & \textit{if } \; j \leq 2^{\lceil \log i \rceil} - i \\ \lceil \log i \rceil & \textit{otherwise} \end{cases}$$

Proof (Sketch).

case a) $i = 2^k$ for some $k \in \mathbb{N}$. (Then $2^{\lceil \log i \rceil} = i$ and $\lceil \log i \rceil = \lfloor \log i \rfloor$ ...)

case b) $i \neq 2^k$ for any $k \in \mathbb{N}$. (Then $\lceil \log i \rceil - 1 = \lfloor \log i \rfloor$ ...) □

This leads to the following algorithm which is executed atomically whenever a process is passive and receives a control message $<$CREDIT$>$. Note that "$C := \lceil \log(N) \rceil$" in line 3 can easily be computed by "$C := 0$; **while** $2^C < N$ **do** $C := C+1$;" because C is the smallest integer k so that $2^k \geq N$.

```
1.  $N := |ACTIVATED|$ ;
2.  **if** $N = 0$ **then send** $\langle CREDIT \rangle$ **to** ENV ;
3.  **else** $C := \lceil \log(N) \rceil$ ;  $J := 1$ ;
4.      **for all** $P \in$ ACTIVATED **begin**
5.          **if** $J \leq 2^C - N$ **then send** $\langle CREDIT + C - 1 \rangle$ **to** P ;
6.                           **else send** $\langle CREDIT + C \rangle$ **to** P ;
7.          **fi** ;
8.          $J := J+1$ ;
9.      **end** ;
10.     $ACTIVATED := \emptyset$ ;
11. **fi** ;
```

5. Optimizations and variants

Section 3 described a method for accumulating credit shares CR at the environment by using a set DEBTS for keeping book on the the missing shares. However, if it can be guaranteed that the values CR do not become too big, i.e., if $CR \leq LIMIT$ for some constant LIMIT, it is possible to realize $D := D - 2^{-CR}$ in a more direct way. By "shifting" D by LIMIT (and consequently initializing it to 2^{LIMIT}) the updating action becomes $D := D - 2^{LIMIT - CR}$. Because $LIMIT - CR \geq 0$, the computation can then be based on the built-in integer arithmetic if 2^{LIMIT} is not larger than the maximum integer value. Otherwise, the binary subtraction operating on a sufficiently large bit vector must be programmed explicitly.

Fortunately, it is possible to guarantee $CR \leq LIMIT$ for the variant described in Section 4 by adding a test "**if** $CREDIT + C \geq LIMIT$..." between lines 3 and 4. If the test succeeds, the process returns CREDIT to the environment and requests a larger share (i.e., a smaller value of CREDIT). The local control algorithm can then be aborted by skipping lines 4 - 11. When the environment receives the returned credit share CREDIT it *raises the total credit* by $D := D + 2^{LIMIT} - 2^{LIMIT - CREDIT}$ and sends a "fresh" credit value 0 to the process. Notice that this credit raising (requiring two extra control messages) should be a rare action. Anyhow, LIMIT must be small enough to avoid an arithmetic overflow when raising the credit.

An interesting variant which might reduce the number of control messages results from introducing *control rounds* and the possibility of "early returning" of the credit share to the environment. Assume that all control messages are tagged with the sequence number of the current control round and that a process remembers the highest number it has encountered so far. Whenever a control message $<$CREDIT$>$ arrives at a process for which ACTIVATED$\neq\emptyset$ but which has already been visited by a control message of the current round, CREDIT is immediately returned to the environment together with a request to revisit this process during the next round. When the environment has recovered the whole credit, a new control round may be started (possibly after waiting some time) if some processes remain to be visited. The environment simply sends control messages to these processes as if it had directly activated them. Since the number of control messages per round is bounded, the environment does not get overwhelmed with control messages. Therefore, there is no danger that the environment becomes a serious bottleneck. This is an important advantage of the control round variant.

Notice that the environment can start the detection algorithm by sending control messages to *all* processes. Those that did not take part in the underlying computation will immediately return their credit share. Since thereby each process can act as the environment without knowing the processes which initially became active, we can dispense with the notion of an environment—any process may start the underlying computation and any process may initiate the detection algorithm. (The control messages must then carry the identity of the initiating process in order to be able to return the credit shares). However, if several processes concurrently start the detection algorithm, global quiescence can only be concluded if *all* of them agree. Such an agreement scheme has been worked out by Shavit and Francez in their symmetric variant of the Dijkstra/Scholten principle [5].

6. Discussion

The principle of the first variant described in Sections 2 and 3, where credit values are "piggybacked" on the activation messages, seems to be more natural than sending credits later, after the activation messages. However, the variants described in Sections 4 and 5 may need less control messages and less memory for the accumulation of the credit shares; they are also more symmetric since any process can act as the environment.

The space complexity of the algorithm should not pose any problems in practice. The value of CREDIT is bounded by the total number of activation messages and can be coded by about log CREDIT bits. Furthermore, CREDIT can be bounded by some constant LIMIT as has been shown in Section 5.

The overall principle of the detection algorithm is reminiscent of the diffusing computations scheme [2]. Whereas in that scheme an explicit acknowledgement for each message is used, here the environment is directly notified whenever a process becomes passive.

The idea of using a splitting credit scheme and maintaining the sum of the values invariant has also been used in the reference counting scheme of an elegant distributed garbage collection algorithm [1]. In fact, distributed garbage collection and global quiescence detection are related problems; in [6] a connection between the two problems is established by showing how garbage collection algorithms can be derived almost mechanically from a distributed termination detection protocol. As Tel has noted [7], Bevan’s garbage collection scheme [1] can easily be transformed to a quiescence detection algorithm whose principle is similar to the one described here.

ACKNOWLEDGEMENTS

I am grateful to Christian Beilken, Uli Lemberg, Monika Mevenkamp, and Ronny Mohr for fruitful discussions and useful comments on an earlier draft of this paper. Special thanks to Gerard Tel for drawing my attention to Bevan’s garbage collection algorithm, and to Mike Spenke for suggesting to collect the missing credit shares instead of counting directly the credit shares returned to the environment.

References

[1] BEVAN D.I. (1987) *Distributed Garbage Collection Using Reference Counting.* In: J.W. de Bakker et al., Proc. PARLE – Parallel Architectures and Languages Europe, Springer-Verlag LNCS 259, pp. 176-187

[2] DIJKSTRA E.W., SCHOLTEN C.S. (1980) *Termination Detection for Diffusing Computations.* Information Processing Letters 11:1, pp. 1-4

[3] FRANCEZ N. (1980) *Distributed Termination.* ACM Trans. on Prog. Lang. and Sys. 2:1, pp. 42-55

[4] MATTERN F. (1987) *Algorithms for Distributed Termination Detection.* Distributed Computing 2:3, pp. 161-175

[5] SHAVIT N., FRANCEZ N. (1986) *A New Approach to Detection of Locally Indicative Stability*. In: L. Kott (ed.), Proc. 13th ICALP, Springer-Verlag LNCS 226, pp. 344-358

[6] TEL G., TAN R.B., VAN LEEUWEN J. (1988) *The Derivation of Graph Marking Algorithms from Distributed Termination Detection Protocols.* Science of Computer Programming 10:2, pp. 107-137

[7] TEL G. (1988) *Personal communication, April 1988.*