ELSEVIER

Information Processing Letters 66 (1998) 175–180

Information Processing Letters

Efficient token-based control in rings ☆

Esteban Feuerstein [a,1], Stefano Leonardi [b,*], Alberto Marchetti-Spaccamela [a,2], Nicola Santoro [c,3]

[a] *Departamento de Computación, FCEyN, Universidad de Buenos Aires and Instituto de Ciencias, Universidad de General Sarmiento, Buenos Aires, Argentina*

[b] *Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy*

[c] *School of Computer Science, Carleton University, Ottawa, Ontario, Canada, K1S 5B6*

Received 24 March 1998
Communicated by F. Dehne

Abstract

In this paper we deal with the efficiency of *token-based strategies* for the basic problem of controlling the allocation of a shared resource in a ring of n processing entities. We propose new protocols that allow a bounded number of exchanged messages per access request to the resource, while this amount is unbounded for classical solutions. We also guarantee all the requests to be served within a maximum delay. The new proposed protocols are *request-message-based strategies*, in that a process entity sends a message to "inform" the token of the access request. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Distributed computing; Amortized analysis

1. Introduction

Consider the problem of controlling the allocation of a shared resource in a ring of n processing entities.

A common solution to this problem is by means of a *permission* (*control*) *token*: a *free* token is passed from one entity to the next in the ring according to a pre-defined set of rules (protocol) understood and adhered by all the entities; to access the resource, an entity must first obtain a free token and convert it into a *busy* token; once the use of the resource is terminated, the busy token is destroyed and a new free token is created.

This mechanism, which we shall here call *circular token-based control*, was the primary motivation for the study of the *election* problem [10], and it is obviously used to solve the *mutual exclusion* problem [13].

Circular token-based control is used in a wide variety of communication problems other than *ring networks*. For example, data transmission in *token rings* [1,14] and access to transmission medium in *bus networks* [3] are solved by arranging the entities in a "logical ring" and then applying the above described mechanism; a conceptually similar situation is found

☆ A short abstract of this paper appears in the *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, 1996. The work is partially supported by EU ESPRIT Long Term Research Project ALCOM-IT under contract no. 20244, and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative", EEC project KIT-DYNDATA, by Natural Science and Engineering Research Council Research Grant A2415 and by University of Buenos Aires' program "Programa Investigadores Jóvenes".

* Corresponding author. Email: leon@dis.uniroma1.it.

[1] Email: efeuerst@dc.uba.ar.

[2] Email: alberto@dis.uniroma1.it.

[3] Email: santoro@scs.carleton.ca.

PII: S0020-0190(98)00054-4

in *hub polling systems* [3]. It is interesting to note that all these different situations use exactly the same "hot-potato" solution: an entity holding the token will pass it along the ring as soon as it no longer needs it (e.g., see [2,7,13]).

An extensive amount of literature exists on the performance of the resulting system when circular token-based control is used in token rings, bus networks and hub polling systems (e.g., [3,5,12]). On the other hand, surprisingly little is known about the efficiency of the use of this mechanism. In fact, within the context of *distributed computing*, the research has mostly focused on detection of token loss [10,11] and on self-stabilization aspects (e.g., [4,6,8,9]).

In this paper we are concerned with the very *basic* question of how efficiently can token-based control be implemented in a ring network.

We consider the case of a single resource and a single token in a fault-free ring. Requests for the resource are generated by the entities in an "on-line" fashion: each entity can generate a request for the resource at any time, and neither the location nor the time of the requests is known a priori. To use the resource, an entity needs a special message, the token T, which the processors hand from one to the other around the ring.

The goal is that of minimizing the number of *messages per request* necessary to allocate the resource to a given set of requests. The solution must be *fair*, every request will be granted within finite time.

As mentioned above, the solution used in literature is the obvious one, characterized by the following rule: an entity holding the token will pass it along the ring as soon as it no longer needs it. According to this rule, if an entity which did not request the token receives it, it will immediately forward it. An important drawback of this solution is that the token is circulating even if no processes ask the resource. In this case the number of messages exchanged may be *unbounded* even for a finite set of requests. This fact has immediate *negative practical consequences*: an exceedingly large amount of communication might be spent to manage a seldomly used resource.

We present two new protocols, that we call Q and D, for circular token-based control. Both algorithms guarantee a *bounded* number of messages per request, by implementing a *request-message-based strategy* that let the token circulate only if it is informed of a process entity asking the resource. Our proposals do not require to piggyback any information about the processors identifiers on token messages.

The analysis of the efficiency of the proposed protocols uses two distinct measures.

The first measure is the *average number of messages-per-request* necessary to satisfy a sequence of requests or, equivalently, the *worst case ratio* between the total number of (token and request) messages and the number of requests in the sequence. We will use *amortized analysis* techniques [17] for this first measure.

The second measure is the *service traffic*, that is the worst case number of messages that are exchanged in the network between the time in which a processor sends the request message and the time in which a processor receives the token. The service traffic must be bounded in order to prevent *starvation*.

Results of this paper. Let n be the number of processing entities (or *processors*) in the ring.

We first prove the existence of an $n-1$ lower bound on both average number of messages per request and service traffic.

We then propose and analyze two algorithms. The first solution, protocol Q, requires only n messages per request, and the service traffic is bounded by $\frac{3}{2}n^2$, for which this protocol is almost *optimal* with respect to the number of messages per request.

The second solution, algorithm D, allows a lower service traffic, namely $3n-3$, at the expenses of a slight increase in the number of messages per request that is $2n$. This protocol achieves optimality, up to a constant factor, with respect to both measures.

2. Definitions and preliminary results

The n processors are assumed to be logically structured in a ring in which the communication takes place in a single direction, say clockwise. Let $p_0, p_1, \ldots, p_{n-1}$ be the set of processors. For $i = 0, \ldots, n-2$ processor p_i may communicate directly with processor p_{i+1} and processor p_{n-1} is connected to processor p_0.

We assume that the requests messages are anonymous, i.e., they carry no information about the processor that generated them. Without loss of generality, we

also assume that a processor can ask again the resource only after that the previous request has been satisfied. Processors have neither shared memory nor a common clock, and their speeds are unrelated.

The easy classical strategy [13], that we call S, prescribes that a processor receiving the token from the previous processor in the ring can either enter the critical section, or pass the token to the next processor in the ring. The following theorem, whose trivial proof is left to the reader, motivates our search for request-message-based protocols.

Theorem 1. *For every set of requests algorithm S may require an unbounded number of messages.*

We first show the following lower bounds on the number of messages per request and service traffic.

Theorem 2. *Any algorithm for the token distribution problem in a unidirectional ring with anonymous requests messages requires at least $n-1$ messages per request and the service traffic is at least $n-1$.*

Proof. Assume the token at position p_i when no request is pending. A request at processor p_{i-1} (p_{n-1} if $i=0$) is then issued. Any algorithm must exchange at least $n-1$ messages to bring the token to the pending request, and the service delay is also at least $n-1$. The process can be continued with a new request at p_{i-2} (p_{n-1} if $i=2$) once the token has been brought to the previous request, and so on. □

In the sequel of this paper we describe and analyze our algorithms Q and D.

3. Algorithm Q

The idea of this algorithm is that every processor willing the token sends a request through the ring. When the token receives a request message, it moves to search a processor that made a request. The resource token T contains a counter that is increased when a new request message is received, while it is decreased when a request is served. The token "searches" processors with pending requests while the counter is positive. The behavior of a processor p is formally described as follows:

(1) When p needs the resource then:
- If p has T then it enters the critical section.
- If p has not T then it sends a request message to the next processor.

(2) When p receives a request message then:
- If p has T then it increases the counter and passes T to the next processor.
- If p has not T then the request message is passed to the next processor.

(3) When p receives T then:
- If p has a pending request then it enters the critical section in which the token counter is decreased by 1; at the exit of the critical section p passes T to the next processor if the token counter is greater than 0.
- If p has not a pending request then T is passed to the next processor in the ring.

We show that Q serves all the requests with an almost optimal number of messages per request, and with bounded service traffic.

Theorem 3. *Algorithm Q serves all the requests. It requires at most n messages per request and the service traffic is bounded by $\frac{3}{2}n^2$.*

Proof. A request message is circulating for each request not yet registered in the token counter. The token stops moving when the counter is 0, so every request message will eventually reach the token and "push" it till the next processor with a pending request. The first part of the claim is then proved.

To prove the bound on the number of messages per request we define R to be the set of pending requests and M to be the set of request messages circulating in the network. We abuse of notation and indicate with R and M also the number of requests of R and M.

For each pending request $r \in R$ we denote by $d_1(r)$ the clockwise distance from the processor currently storing T to the processor with pending request r, while for each request message $m \in M$ we denote with $d_2(m)$ the clockwise distance from the processor currently holding m to the processor currently storing T. Moreover, let C be the current value of the token counter. Clearly, $R = M + C$. Let

$$\Phi = \sum_{r \in R} d_1(r) + \sum_{m \in M} d_2(m)$$

be a potential function. The amortized cost of a message is defined as $1 + \Delta\Phi$, where 1 is the cost of the message to the algorithm and $\Delta\Phi$ is the variation of the potential function due to the exchange of the message. We study the amortized cost of any exchanged message:

- The token is passed to the next processor. In this case, by definition of Q, the value of the token counter is $C \geqslant 1$. The potential function is increased for each request message in the network and it is decreased for each pending request. Then, the variation of the potential function is $\Delta\Phi = +M - R = -C \leqslant -1$ and the amortized cost is at most 0.
- A request message is passed to the next processor. The potential function decreases by 1 since the distance between the exchanged request message and the token is decreased by 1. Therefore the amortized cost is 0.

On the other hand, each new request increments Φ exactly by n. As $\Phi \geqslant 0$, the total variation in the potential for a sequence of k requests cannot be less than $-kn$, and therefore we get that the total number of messages cannot exceed the maximum value of the potential function, which gives an average of at most n messages per request.

The service traffic measures the maximum number of request messages exchanged in the networks between the time at which a processor, say p_x, sends a request message, and the time at which the processor receives T.

Without loss of generality, we can assume that all the messages are originated by requests coming after the request of p_x.

Consider a pending request $r \in R$ issued by a processor whose position is between p_x and the position of T. We are only interested in the messages that are exchanged until token T reaches position p_x, so we define $d'(r)$ to be the clockwise distance between T and p_x rather than the clockwise distance between T and the processor that issued request r. Therefore, we define a different potential function

$$\Phi' = \sum_{r \in R} d'_1(r) + \sum_{m \in M} d_2(m).$$

With the same analysis as before, we can show that the amortized cost of every message is smaller or equal than 0.

Let us now consider the maximum variation of the potential function. Requests issued by processors with position between T and p_x (including p_x), give a contribution of n to the potential function; and there may be at most $n - 1$ of those requests. Requests issued by processors between p_x and T contribute exactly with their distance to p_x (by definition of Φ' their contribution is given by their distance to T plus the distance from T to p_x). There are at most $n - 2$ of these requests.

Overall, the maximum variation of Φ' (and therefore the service delay), is bounded by

$$n(n-1) + \frac{(n-2)(n-1)}{2} = \frac{3}{2}n^2 - \frac{5}{2}n + 1.$$

Actually, this bound is also tight. Following the proof it is possible to construct a worst case example that achieves precisely that value. □

4. Algorithm D

The second algorithm we propose is called D. It allows a lower service traffic with a slight increase of the number of messages per request. D is also a request-message-based strategy but, differently from Q, the request message is stopped when it is recognized that the processor issuing the request will receive the token anyway.

In this algorithm the resource token T contains a bit that memorizes one of two possible states: *active* and *check*. In the *active* state the resource token T is searching for a request in the ring. In the *check* state the resource token makes an extra tour in the ring to check if it is aware of all the requests (this extra tour is called the *check tour*). For this purpose, T contains a counter to establish whether the tour in the ring has been completed.

Each processor p contains a local variable M_p whose value is 1 if a request message has passed through the processor after that T has passed for the last time, and 0 otherwise.

The behavior of a processor p that executes algorithm D is as follows:

(1) When p needs the resource there are two possibilities:
   - If p has T then it enters the critical section.
   - If p has not T then:

- If $M_p = 0$ then it sends a request message to the next processor and sets $M_p = 1$.
- If $M_p = 1$ then no message is sent.

(2) If p receives a request message while it has T then M_p is set to 0, the token state to *active* and T is passed to the next processor.

(3) If p receives a request message while it has *not* T then:
- If $M_p = 0$ then the request message is passed to the next processor in the ring and M_p is set to 1.
- If $M_p = 1$ then nothing is done (the request message is stopped).

(4) If p receives T with state *active* then:
- If p has a pending request then it enters the critical section in which it sets M_p to 0, the token counter to $n - 1$ and the token state to *check*. At the end of the critical section T is passed to the next processor.
- If p has not a pending request then M_p is set to 0 and T is passed to the next processor.

(5) If a processor p receives T with state *check* then:
- If p has a pending request then it enters the critical section in which M_p is set to 0 and the token counter is decremented by 1. Once the critical section is terminated, T is passed to the next processor if the token counter is bigger than 0.
- If p has no pending request then M_p is set to 0, the token counter is decremented by 1 and, if it is greater than 0, T is passed to the next processor.

The following theorem proves that the performance of D is only a constant factor away from the optimum, both for the number of messages per request and for the service traffic. The constant-factor increase in the number of messages required by D with respect to Q is due to the additional check tour necessary to prevent starvation.

Theorem 4. *Algorithm D serves all the requests. It requires at most $m = 2n$ messages per request and the service traffic is at most $3n - 3$.*

Proof. We first prove that D serves all the requests. When a process p needs the token, T's state may be *active* or *check*. In the first case, T will arrive to p before it finishes the check tour. If the state is *check* then there are two possibilities: either T will pass through p before ending the check tour, in which case we are done, or it has already passed through p (this includes the case in which the token has finished the check tour and it is idle). But then a request message will eventually reach T and produce at least one complete tour on the ring.

Let's turn now to the amortized number of messages per request. A request message will traverse at most n edges before it is stopped or it reaches the token; after that only the token must move. Every check tour requires n additional messages. In the worst case a check tour may be done for each request, and therefore at most $2n$ messages are sent.

As for the service traffic, three different types of messages may be sent between the time a processor, say p_x, generates a request and the time it is served. The first type is determined by requests generated from processors between p_x and the token position, following the clockwise order. The number of this kind of messages is at most $n - 1$. The second type of messages is determined by requests generated by processors whose position is between the token position and p_x, following the clockwise order. The number of this second kind of messages is also bounded by by $n - 1$ since each edge of the ring will be traversed by at most one of them. Finally, we have to count at most $n - 1$ token messages, which gives altogether a maximum of $3n - 3$ messages. □

References

[1] D. Andrews, G. Schulz, A token-ring architecture for local area networks: An update, in: Proc. IEEE COMPCON, Fall 1982.

[2] ANSI/IEEE Standards 802.2-1985, 802.3-1985, 802.4-1985, 802.5-1985.

[3] D. Bertsekas, R. Gallager, Data Networks, 2nd ed., Prentice-Hall, 1992.

[4] J.E. Burns, J. Pachl, Uniform self-stabilizing rings, ACM Trans. Programming Languages and Systems 11 (2) (1989) 330–344.

[5] W. Bux, Performance issues in local area networks, IBM J. 23 (4) (1984).

[6] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Comm. ACM 17 (11) (1974) 643–644.

[7] F. Halsall, Data Communications, Computer Networks and Open Systems, 3rd ed., Addison-Wesley, 1992.

[8] A. Israeli, M. Jalfon, Token management schemes and random walks yield self-stabilizing mutual exclusion, in: Proc. 9th ACM Symposium on Principles of Distributed Computing, 1990, pp. 119–129.

[9] H. Kakugawa, M. Yamashita, Uniform randomized self-stabilizing mutual exclusion on unidirectional ring under unfair c-deamon, Proc. 2nd Workshop on Self-Stabilizing Systems, 1995, pp. 14.1–14.13.

[10] G. Le Lann, Distributed systems—Towards a formal approach, in: B. Gilchrist (ed.), Inform. Processing, Vol. 77, North-Holland, 1977, pp. 155–160.

[11] J. Misra, Detecting termination of distributed computations using markers, in: Proc. 2nd ACM Symposium on Principles of Distributed Computing, 1983, pp. 290–294.

[12] V. Rego, L.M. Ni, Analytic models of cyclic service systems and their application to token-passing local networks, IEEE Trans. Computers C-37 (10) (1988) 1224–1234.

[13] M. Raynal, Distributed Algorithms and Protocols, John Wiley & Sons, 1988.

[14] F.E. Ross, FDDI—A tutorial, IEEE Commun. Mag. 24 (1986) 10–17.

[15] M. Singhal, A taxonomy of distributed mutual exclusion, J. Parallel Distrib. Comput. 18 (1993) 94–101.

[16] I. Suzuki, T. Kasami, A distributed mutual exclusion algorithm, ACM Trans. Comput. Syst. 3–4 (1985) 344–349.

[17] R.E. Tarjan, Amortized computational complexity, SIAM J. Alg. Discrete Meth. 6 (2) (1985) 306–318.