行政院國家科學委員會專題研究計劃成果報告

分散式互斥與物件追蹤

Distributed Mutual Exclusion and Object Tracking

計劃類別：☑個別型計劃　　□整合型計劃

計劃編號：NSC 89—2213—E—002—021

執行期間：民國 88 年 8 月 1 日至民國 89 年 7 月 31 日

個別型計劃：計劃主持人：蔡益坤

處理方式：　☑可立即對外提供參考
□一年後可對外提供參考
□兩年後可對外提供參考
(必要時，本會得展延發表時限)

執行單位：國立臺灣大學資訊管理學系

中華民國 90 年 2 月 20 日

摘　要

本計畫的內容在探討分散式計算領域中的兩個基本問題----分散式互斥以及物件追蹤 (或定位)。分散式互斥問題存在的歷史較久，也因此被研究得較爲透徹；物件定位問題則因其在行動物件管理上的關鍵角色，近幾年來也受到相當的重視。當系統中各程序之間的連結組態必須被納入考慮時，現有的分散式互斥演算法都利用了所謂 token-based 的策略。由於代表唯一特權的 token 與行動式物件的特性類似，token-based 的分散式互斥演算法往往也可以在稍事修改後應用於解決物件定位的問題。

我們對現有的 token-based 分散式互斥演算法做比較分析，將各演算法的觀念及特色做了詳細的對照。同時，爲了能夠處理重製物件的定位問題，我們在考慮網路連結組態的情形下，進一步探討比互斥更廣義的共容互斥 (*k*-exclusion) 問題。我們提出兩個解決共容互斥問題的演算法，並說明如何將這些演算法應用於追蹤重製的行動物件。

關鍵詞：目錄服務、分散式演算法、容錯性、行動物件、互斥問題、網路協定、物件定位、重製物件、路由協定。

ABSTRACT

This research concerns two fundamental problems in distributed computing---- mutual exclusion and object tracking. The mutual exclusion problem has a longer history and is more extensively studied, while object tracking is a key problem in the management of mobile objects which has attracted much attention recently. For a variant of the mutual exclusion problem where the network topology is taken into account, all existing distributed solutions use the idea of a unique token. It turns out that these token-based solutions for mutual exclusion can also be adapted for object tracking, as the token behaves very much like a mobile object does.

We present a comparative survey of existing token-based mutual exclusion algorithms contrasting their basic ideas. To handle objects with replications, we go on to consider the more general k-exclusion problem which seems not to have been studied in a network setting. We propose two solutions for the k-exclusion problem and show how they can be adapted for tracking replicated mobile objects.

Keywords: Directory Service, Distributed Algorithms, Fault Tolerance, Mobile Objects, Mutual Exclusion, Network Protocols, Object Tracking, Replicated Objects, Routing Protocols.

Distributed Mutual Exclusion and Object Tracking (Final Report of NSC 89-2213-E-002-021)

Yih-Kuen Tsay
Department of Information Management
National Taiwan University
E-mail: tsay@im.ntu.edu.tw

February 20, 2001

Abstract

This research concerns two fundamental problems in distributed computing—mutual exclusion and object tracking. The mutual exclusion problem has a longer history and is more extensively studied, while object tracking is a key problem in the management of mobile objects which has attracted much attention recently. For a variant of the mutual exclusion problem where the network topology is taken into account, all existing distributed solutions use the idea of a unique token. It turns out that these token-based solutions for mutual exclusion can also be adapted for object tracking, as the token behaves very much like a mobile object does.

We present a comparative survey of existing token-based mutual exclusion algorithms contrasting their basic ideas. To handle objects with replications, we go on to consider the more general k-exclusion problem which seems not to have been studied in a network setting. We propose two solutions for the k-exclusion problem and show how they can be adapted for tracking replicated mobile objects.

Keywords: Directory Service, Distributed Algorithms, Fault Tolerance, Mobile Objects, Mutual Exclusion, Network Protocols, Object Tracking, Replicated Objects, Routing Protocols.

1 Introduction

A distributed system stores and manages various shared resources so that they can be conveniently accessed by users of the system. We refer to an instance of any of these resources as an *object*. The mutual exclusion problem is fundamental in such a system, as a shared object typically may be accessed by one user at a time to ensure consistency. Locating an object so as to deliver messages (such as operation requests) intended for the object is also fundamental; the problem becomes more complicated when objects may move. We first outline the problem setting and explain the similarity between the two problems. We then highlight the main results of this report along with related work.

Networks of Processes

As a general model of typical distributed systems we consider networks of processes whose topology is given by an undirected graph, where the nodes represent the processes and the edges represent the communication links. A communication cost is associated with each link. Two neighboring processes (that are connected by a communication link) may communicate with each other by exchanging messages directly over the link that connects the two processes. A message from one process to another non-neighboring process will have to be routed through other processes.

Mutual Exclusion

Under the network model, the privilege to exclusively access a shared object, or enter the critical section (in the terminology of the mutual exclusion problem), is typically materialized by the possession of a unique token. The token is initially held by one of the processes. To obtain the privilege, a process sends out a request that is relayed to the the token holder. An algorithm for the mutual exclusion problem essentially needs to address (1) how a request is forwarded to the token holder and (2) how processes change their states, while the token is being transfered to the requesting process, to reflect the new location of the token.

Mutual exclusion in a network of processes is very different from that in a shared memory system. Algorithms that assume a uniform cost of communication between pairs of processes are unlikely to be efficient for a network of processes.

Object Tracking

The task of locating an object in a distributed system is usually performed by a directory service or name service of the system [CDK94]. The relevant directory service may be centralized at a particular server or distributed across a number of servers or even the entire system. Disregarding these variations, certain distributed data structure has to be maintained to keep track of the objects so that a request can be routed to the intended object. Forwarding a request to the node where a particular mobile object resides is very similar to forwarding a request to the token holder in a mutual exclusion algorithm. The distributed data structure of a mutual exclusion algorithm intuitively can be adapted as the distributed directory for tracking a mobile object.

Most existing systems that support mobile objects (such as a cellular network) use home-based tracking strategies. In a home-based directory, every object has a fixed home and all messages for an object (or at least the first of messages in the same session) must be routed through its home. When an object moves, it reports its new location to its home which updates the routing table accordingly. This type of protocols is vulnerable to node or link failures. An object becomes inaccessible when its home fails or communications to its home are blocked.

Replicating the home is one possible solution. However, stationary replicated homes may not respond efficiently to a dynamically changing network.

Main Results

We present a comparative survey of existing token-based mutual exclusion algorithms contrasting their basic ideas. To handle objects with replications, we go on to consider the more general k-exclusion problem where at most k processes are allowed in the critical section at any time. The k-exclusion problem was first defined by Fischer [FLBB79]. A number of algorithms have been suggested [ADGS94, AM94, AV99], all designed for *shared-variable* systems. To the best of our knowledge, the problem has not been studied in a network setting. Based on the idea of a token-based mutual exclusion algorithm, we derive two k-exclusion algorithms for the network model. We then show how these algorithms can be adapted for tracking replicated mobile objects.

Related Work

Van de Snepscheut [VdS87] was probably the first to give solutions to the mutual exclusion problem in a *general* network of processes, extending the earlier work for rings by Martin [Mar85]. He first assumed that the network is a tree and then extended the solution to one for a general network. His idea was to orient the edges so that they point to the process with the token; when the token moves, the directions of the edges are updated accordingly. Raymond [Ray89] later presented an algorithm based on a spanning tree of the network. The algorithm is identical to the restricted solution of Van de Snepscheut. However, he gave a more detailed analysis of the average case message complexity, which is $O(\log N)$, and showed how node failures may be handled.

Naimi *et al.* [NTA96] presented yet another tree-based solution. Unlike the previous algorithms, the edge set of the tree maintained by their algorithm changes over time. They assumed that the network topology is a complete graph and were also able to derive an average case message complexity of $O(\log N)$. Demmer and Herlihy [DH98] proposed the so-called arrow distributed directory protocol for keeping track of mobile objects in a distributed system. The algorithmic technique is essentially a combination of those of Van de Snepscheut and Naimi *et al.*

Mullender and Vitányi [MV88] formalized the general problem of distributed match-making. They showed that mutual exclusion and object tracking (or name service, in their terminology) can be formulated as special instances of the distributed match-making problem. However, the formulations seem to be biased toward particular types of algorithms. Peleg [Pel93] defined the concept of distance-dependent directories and presented efficient constructions of such directories that may be used to implement a name service. Awerbuch and Peleg [AP95] designed directory services that are organized as a hierarchy of subdirectories. They showed that the communication overhead of their tracking mechanism is close to optimal. Plaxton *et al* [PRR97] presented a

randomized algorithm for accessing shared objects that tends to satisfy each access request with a nearby copy. They also considered dynamic changes in both the network and the set of object copies.

2 Token-Based Algorithms for Mutual Exclusion

We surveyed four token-based mutual exclusion algorithms, namely Van de Snepscheut [VdS87], Raymond [Ray89], Naimi *et al.* [NTA96], and Demmer and Herlihy [DH98]. The four algorithms are similar in that they all maintain distributed tree-like structures (plus some auxiliary structures) that guide the routing of a request to the token holder or an eventual token holder. Van de Snepscheut went further to extend the idea to using more general directed acyclic graphs; however, he did not pursue the fault-tolerance potential of the extended algorithm.

Van de Snepscheut and Raymond

Raymond's algorithm and the tree-version of Van de Snepscheut's algorithm are in fact identical. The algorithms start with a spanning tree of the network whose root is the node holding the unique token. A node wishing to enter the critical section must first acquire the token. As the token travels around the tree, the root changes its location accordingly.

It is convenient to think of each tree edge as being oriented such that the edge points from the child to the parent. Every node thus has exactly one directed path pointing to the root. When a node wants to enter the critical section, it sends a request along the first edge of the unique directed path. The request is routed along the unique path to the root (every node in the path participates in relaying the request). When the request eventually arrives, the root can simply send the token along the path that the request has traveled (in the reverse order). The direction of an edge is reversed when the token passes through it and arrives at the other end (in accordance with the view that every tree edge points from the child to the parent). The more elaborate part is in the handling of simultaneous requests.

After sending out a request (either as an originator or a leg in relay), a node may subsequently receive some other requests from its other children. Any of these subsequent requests will be put in a local queue of the node. When the token arrives in response to the outstanding request, the node enters the critical section if it was the originator of the request. Otherwise, the node passes the token via the edge from which the outstanding request was received and, if there is any request in the local queue, the node removes the first from the queue and sends it over the same edge (which now points to the root, where the token is).

More about Van de Snepscheut

In his original paper, Van de Snepscheut actually did not talk of spanning trees. He first considered tree-shaped networks and devised the previously described mechanism of routing requests and the token and he then extended the idea to general networks. His final algorithm orients the edges of the network such that the network becomes a directed acyclic graph. The token holder is the single sink of the directed acyclic graph. Unlike in the tree case, a node now may have more than one directed path pointing to the token holder. The routing scheme is modified as follows.

When a node wants to enter the critical section, it sends a request along the first edge of one of the directed paths to the token holder. The next node in the path will either relay the request along one of its outgoing edges or put the request in the local queue. As before, the token is routed in the reverse order along the path that the request has traveled. The direction of each edge incident to a node is updated to point to the node, when it receives a token; this maintains acyclicity and also ensures that the token holder is the only sink of the entire graph.

Naimi *et al.*

Naimi *et al.*'s algorithm also uses a tree structure, but their tree is for routing requests only and is more dynamic with a changing set of edges. An additional distributed queue tells the token holder and subsequent holders which node is the next in line waiting for the token. The root of the tree is not intended to be the token holder, but rather the node at the tail of the queue, where some other process may be added. In other words, the tree provides routing information for a process to join the distributed waiting queue. They assume that the network is a complete graph. Initially, the tree is a star-shaped one with the token holder as the root and the parent of every other node (any spanning tree would do); the token holder is also the only node, the head, and the tail of the queue.

When a node wants to enter the critical section, it sends a request to its parent. It then regards itself as the new root (by nullifying the pointer to parent) expecting that, once its request is received by the current root, it will be added to the queue and become the new tail. Other nodes on the directed path to the current root, upon receiving the request, make the requester their new parent. As the request travels to its destination, tree edges are removed and added and the tree is temporarily split into two smaller trees (which are also dynamic). Finally, the current root inserts the requester behind itself in the queue (by setting a local pointer to the requester) and also make the requester its new parent, turning itself into a non-root node and thus merging the two smaller trees back into one. The head of queue, after holding the token for a finite amount of time, will pass out the token to the next node and remove itself from the queue. Every node in the queue eventually will get the token.

More than one processes may be trying to enter the critical section. Though the overall changes to the tree and the queue may be more complicated, processes behave just as described above. A new root may receive some other request even before it is actually added to the queue; it handles the request just as the current root would do. The isolated segment of queue gets hooked to the distributed queue when the new root is eventually inserted. A root (in each of its incarnations) allows just one process to be added behind it in the queue. Once such an addition occurs, the root has got a new parent, i.e., the requester; a second request will be forwarded to its new parent (which may receive yet another request at an earlier time and will have to forward the second request to the earlier requester).

The assumption of a complete communication network makes it possible for a node to switch its parent to any other node and hence for the tree to change so dynamically. Practically, this assumption boils down to working in a general network with an underlying routing service. The "physical link" between each pair of nodes corresponds to the "logical link" (realized by the routing service) between the pair in the general network. Demmer and Herlihy, to be introduced next, also assumes an underlying routing service.

Demmer and Herlihy

Demmer and Herlihy's algorithm closely resembles that of Naimi *et al.* However, their tree has a fixed set of edges which is more like the tree in Van de Snepscheut's and Raymond's algorithms. They also use a distributed queue for lining up the processes waiting for the token which is identical to that of Naimi *et al.*; the tree tells where the tail of queue is. An underlying routing service is assumed for transmitting the token; as we pointed out earlier, the routing service provides a logical complete communication network that Naimi *et al.* assumed.

Their algorithm starts with a spanning tree of the network whose root is the node holding the unique token. Changes to the orientation of the edges occur while requests are processed. When a node wants to enter the critical section, it sends a request to its parent and then regards itself as the new root. The algorithm differs from Naimi *et al.*'s in how a node changes its parent. Each of the intermediate nodes on the directed path to the current root, after forwarding the request to the next node, makes the precedent node its new parent (the direction of the corresponding edge is reversed). Finally, the current root inserts the requester behind itself in the queue and also makes the precedent node its new parent. When this is done, the direction of the path that the request has traveled is reversed, pointing to the new root.

More than one processes may be trying to enter the critical section. But again, like in Naimi *et al.*'s algorithm, although the overall changes to the tree and the queue may be more complicated, processes behave just as described above.

3 Algorithms for k-Exclusion

We now consider the k-exclusion problem. Based on the idea of Van de Snepscheut's mutual exclusion algorithm, we derive two k-exclusion algorithms.

3.1 Algorithm 1

Algorithm 1 is essentially a generalization of the tree version of Van de Snepscheut's algorithm. The basic ideas are as follows. Initially, the k tokens are distributed arbitrarily among the nodes of the network. Each node records the number of tokens it holds and, for each tree edge, it also records the number of tokens held by the nodes on the subtree that the edge leads to. If a node wants to enter the critical section and does not have a free token, it sends a request along a tree edge with the highest (and nonzero) number of "free" tokens. A receiving node of the request that is without a free token passes the request also along a tree edge with the highest number of free tokens. The request eventually will reach either a node with a free token or a node where further forwarding will be fruitless. In the first case, the token backtracks the path that the request has traveled. In the second case, the request is put into the local queue of the last receiving node and will be forwarded when it emerges to the head of queue and there is a tree edge with nonzero free tokens.

To identify an edge with the highest number of "free" tokens, each node maintains two variables t_e and r_e for each incident edge e. The variable t_e indicates "the number of tokens in the subtree that e leads to", while r_e indicates "the number of outstanding requests sent along edge e". The number $t_e - r_e$ is then the number of "free" tokens on the subtree that e leads to.

It is interesting to note that Algorithm 1 degenerates into the mutual exclusion algorithm of Van de Snepscheut when k equals 1.

3.2 Algorithm 2

Algorithms that utilize a spanning tree of the network have a few drawbacks. Whenever a communication link that is a tree edge fails, the network is partitioned. Tokens cannot be exchanged between two partitions until the failed link recovers. Moreover, as messages are always routed through the tree edges, some links may become communication bottlenecks. To strengthen tolerance toward link failures and to reduce chance of bottlenecks, we propose a second k-exclusion algorithm that maintains a directed acyclic graph (DAG) instead of a spanning tree, again inspired by the work of Van de Snepscheut.

Algorithm 2 works as follows. Initially, a *primary* token is assigned to one of the processes in the network. The algorithm maintains a directed acyclic graph of the network such that all edges are directed toward the primary token. Every request is routed along the directed edges

until it reaches the primary token holder; any path may be chosen if there exist more than one paths to the primary token. If the primary token is not in use, it is sent to the requester; the direction of the edges change accordingly as the token moves. Otherwise, the primary token holder generates a *secondary* token and sends it to the requester; the movement of a secondary token does not change the direction of an edge. A counter in the primary token records the number of secondary tokens currently in the system so as to guarantee that there are at most $k-1$ of them at a time. Any process holding either the primary token or a secondary token may enter the critical section.

We next describe what should be done to a token when a process leaves the critical section. If the token is primary and the queue of requests is not empty, the process sends it to the first requester in the queue and forwards all other requests in the queue to the new token holder. If it is secondary, the process sends it to the primary token holder, which will destroy the token and increment the secondary token count.

It is not necessary for a request to be always routed to the primary token. The request may be intercepted by a node that has relayed a secondary token on one of its incoming edges. When the secondary token is returned and arrives at this node, the token can be sent directly to the originator of the intercepted request (without being returned to the primary token holder). Intercepting a request reduces the number of exchanged messages, but may increase the response time (depending how quickly the secondary token is released by its holder).

4 Adaptations for Object Tracking

There are many possibilities regarding what a process wants from an object. However, our main concern is for a process to find the whereabout of a mobile object. We assume that a requesting process always wants the object to be moved to its own site and then performs a read or write operation on it. Any object tracking solution needs to meet the following requirements.

- (Safety) There is at most one (legal) copy of the object in the system at any time.
- (Liveness) A requesting process will eventually acquire the object.

With the object replaced by a token, the above two requirements become those for a token-based solution to mutual exclusion. The distributed data structure of a token-based algorithm can therefore be adapted as the distributed directory for tracking a mobile object.

An object may sometimes be replicated to enhance availability. We assume that some other mechanism controls the replication of an object and is not part of the object tracking problem; however, the number of copies never exceeds a predefined bound k. The requirements for tracking replicated objects are as follows:

- (Safety) There are at most k (legal) copies of the object in the system at any time.
- (Liveness) A requesting process will eventually acquire a particular copy of the object that it requested. (This does not rule out the possibility that a process may want just any copy of the object for making a read operation.)

Demmer and Herlihy have outlined a solution for tracking replicated objects in [DH98]. They assumed that one copy of the object is designated as the *primary* copy. A process wishing to write the object must first acquire the primary copy and then "invalidate" the rest $k-1$ secondary copies (without actually moving them). They tackled the object tracking problem directly rather than adapting exiting solutions to exclusion problems. However, their solution is essentially a combination of a mutual exclusion algorithm and a $(k-1)$-exclusion algorithm. The mutual exclusion algorithm is used for acquiring the primary copy, while the $(k-1)$-exclusion algorithm is for acquiring or invalidating a secondary copy. Algorithm 1 may play the role of $(k-1)$-exclusion algorithm in Demmer and Herlihy's solution.

Another solution can be obtained by adapting Algorithm 2. When a node wishes to write the object, it sends a special request to ask for the primary token. The special request can only be satisfied by the primary token. After getting the primary token, it waits until all secondary tokens (there are $k-1$ or less out there) are returned. The option of intercepting a request, as discussed earlier, will have to be removed though.

5 Concluding Remarks

Operations on an object can be more sophisticated than what we have considered. For example, copies of a mobile object may be created or destroyed dynamically. To what extent solutions to exclusion problems can be applied to object tracking deserve further study.

Algorithms that use a spanning tree are vulnerable to link or node failures. These include the mutual exclusion algorithms we reviewed and Algorithm 1 for k-exclusion. Van de Snepscheut's algorithm and Algorithm 2 have better fault-tolerance, as they utilize a directed acyclic graph. The algorithms still work even if some links fail as long as the failures do not lose the token or disconnect the network. However, both algorithms lack a mechanism for handling link recoveries.

Acknowledgment

This is a joint work with Mr. Cheng-Chung Huang. His Masters thesis contains a preliminary, yet more detailed, account of the results presented in this report.

References

[ADGS94] Y. Afek, D. Dolev, E. Gafni, and N. Shavit. A bounded first-in-first-enabled solution to the l-exclusion problem. *ACM Transactions on Programming Languages and Systems*, 16(3):939–953, 1994.

[AM94] J. Anderson and M. Moir. Using k-exclusion to implement resilient, scalable shared objects. In *Proceedings of the 13th ACM Symposium on Principles of Distributed Computing*, pages 141–150, 1994.

[AP95] B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the ACM*, 42(5):1021–1058, September 1995.

[AV99] K. Alagarsamy and K. Vidyasankar. Fair and efficient mutual exclusion algorithms. In *Proceedings of the 12th International Symposium on Distributed Computing, LNCS 1499*, pages 166–179, 1999.

[CDK94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, 1994.

[DH98] M.J. Demmer and M.P. Herlihy. The arrow distributed directory protocol. In *Proceedings of the 12th International Symposium on Distributed Computing, LNCS 1499*, pages 119–134, 1998.

[FLBB79] M.J. Fischer, N. Lynch, J. Burns, and A. Borodin. Resource allocation with immunity to process failure. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 78–92, 1979.

[Mar85] A.J. Martin. Distributed mutual exclusion on a ring of processors. *Science of Computer Programming*, 5:265–276, 1985.

[MV88] S.J. Mullender and P.M.B. Vitányi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[NTA96] M. Naimi, M. Tréhel, and A. Arnold. A log(n) distributed mutual exclusion algorithm based on path reversal. *Journal of Parallel and Distributed Computing*, 34:1–13, 1996.

[Pel93] D. Peleg. Distance-dependent distributed directories. *Information and Computation*, 103:270–298, 1993.

[PRR97] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[Ray89] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, August 1989.

[VdS87] J.L.A. Van de Snepscheut. Fair mutual exclusion on a graph of processes. *Distributed Computing*, 2:113–115, 1987.